Eesti Teadusagentuur
Estonian Research Council

TeaMe

Rakenduste loomise ja programmeerimise alused

Valikkursus gümnaasiumitele

Jüri Vilipõld
Kersti Antoi
Irina Amitan

Rakenduste loomise ja programmeerimise alused

valikkursus gümnaasiumitele
õpik

Jüri Vilipõld, Kersti Antoi, Irina Amitan

2013

Kujundanud Viivi Jokk
TTÜ informaatikainstituut

Sisu

Eessõna

Käesoleval ajal on kõik inimtegevuse valdkonnad (teadus, tehnoloogia, kunst, sport, olme, meelelahutus ja muud) väga tihedalt seotud informatsiooni ja infotehnoloogia kasutamisega. Tänapäevase ning tuleviku infoühiskonna olulisteks ja keskseteks nähtusteks on infotöötlus ja programmjuhtimisega süsteemid – arvutid ja arvutivõrgud, programmjuhtimisega seadmed ja tööpingid, robotid ja sisseehitatud protsessoritega seadmed, telefonid ja teised sideseadmed, … Arvestades eeltoodut, kasvab ka vajadus infotehnoloogia ekspertidele, kes projekteerivad, loovad ja hooldavad erinevaid IKT süsteeme ja nende tarkvara. Tänapäeval peavad enamike erialade spetsialistid mitte ainult oskama kasutada infotehnoloogia vahendeid, vaid suutma ka püstitada ülesandeid oma valdkonna probleemide lahendamiseks. Vajadusel tuleb osaleda ka vastavates projektides, näiteks oma valdkonna mudelite ja algoritmide loomisel. Samal ajal täheldatakse nii meil kui mujal, et kooli lõpetajate huvi infotehnoloogia erialadel õppimiseks võiks olla suurem ning õppima asujate tase on erinev.

Viimasel ajal on mitmes riigis (näiteks USA ja Suurbritannia) läbi viidud põhjalikud uuringud olukorrast infotehnoloogia ja arvutiteaduse õpetamisest koolides ning käivad aktiivsed arutelud sellel teemal. Praeguseks on jõutud järeldusele, et olukord ei vasta aja nõuetele ja arengutendentsidele ning õpetamise sisu vajab olulisi muudatusi. Nende uuringute alusel on loodud koolide arvutiõpetuse jaoks uued kontseptsioonid ja standardid ning soovitused õppekavade uuendamiseks. On jõutud järeldusele, et koolis ei piisa ainult infotehnoloogia vahendite kasutama õppimisest, vaid peab tundma ka nende töö põhimõtteid ja loomise meetodeid. Selle näiteks on USA arvutiõpetajate assotsiatsiooni (CSTA) poolt koolide jaoks koostatud arvutiteaduse rahvuslik standard ja tüüpõppekava ning Suurbritannia koolide arvutiteaduse õppekava kontseptsioon. Viimases on määratletud nõuded arvutialaste teadmiste ja oskuste kohta erinevate kooliastmete jaoks alates 1. astmest (vanus 5–7 aastat). Nendes materjalides on tehtud ettepanek muuta vastav õppeaine koolides kohustuslikuks.

Seoses koolide ja ülikoolide arvutiõpetuse aktuaalsete probleemide aruteludega ning uute kontseptsioonide ja suundade määramisega on võetud kasutusele mõiste *Computational Thinking* (*CT*). *CT* on kesksel kohal ka näiteks ülalpool nimetatud standardis ja kontseptsioonis ning paljudes teistes materjalides, kuid Eesti keeles vastavat terminit veel ei ole – pakuksime selleks terminiks infotehnoloogiline või infoloogiline lähenemisviis.

Infotehnoloogilise lähenemisviisi (*CT*) põhiolemus seisneb oskuses formuleerida probleeme sellisel kujul, et neid oleks võimalik efektiivselt lahendada infotehnoloogia vahendite abil. Sellega seonduvad järgmised mõisted ja tegevused: abstraktsioon, süsteemide struktuuri, oleku ja käitumise modelleerimine, süsteemi-analüüs, rakenduste ja infosüsteemide disain, algoritmimine ja programmeerimine jms.

Järgnevalt on toodud mõned lingid, kust on võimalik saada täiendavat informatsiooni: Wikipedia, CSTA, ISTE (*The International Society for Technology in Education*), Google.

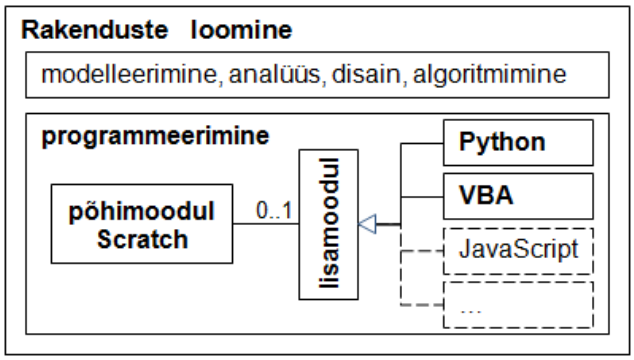
Üheks tähelepanuväärsemaks nähtuseks on järjest tekkivad ning kiiresti arenevad ja levivad programmeerimiskeeled, mis on loodud spetsiaalselt programmeerimise õppimiseks/õpetamiseks nagu Alice, Scratch, BYOB, AgentSheets, Greenfoot, Kodu, MS Small Basic, StarLogo, MIT App Inventor. Neid nimetatakse ka hariduslikeks programmeerimiskeelteks (Educational programming languages). Õppe-otstarbelisi programmeerimiskeeli on loodud ja kasutatud juba ammu – esimeste seas olid näiteks BASIC (1964), Logo (1967), Pascal (1970). Mõnedest neist kasvasid välja ka nö tööstuslikud variandid nagu Visual Basic (VB) ja Visual Basic for Application (VBA).

Uue põlvkonna hariduslikud programmeerimiskeeled on enamikus graafilised ehk visuaalsed programmeerimiskeeled. Tegemist on lihtsa süntaksiga keelte ning mugava kasutajaliidesega

keskkondadega, kus programm pannakse kokku peamiselt hiire abil. Taolistes süsteemides on osutatud suurt tähelepanu atraktiivsusele ja multimeedia (graafika, heli, videod) kasutamisele ning võimalusele lihtsalt ja kiirelt luua mänge, animatsioone, koomikseid, taieseid, esitlusi, omandades selle abil programmeerimise olemuse, põhimõisted ja meetodid märgatavalt kiiremini, kui alustades kohe „suurte“ proffidele mõeldud keeltega. Sellega on arvestatud ka antud kursuse sisu, struktuuri ja kasutatavate vahendite valikul.

Kursuse põhieesmärk on rakenduste loomise ja programmeerimise põhimõtete omandamine. Rõhuasetus on just rakenduste loomisel, mitte lihtsalt programmeerimisel mingis keeles. Rakenduste loomine on mõnevõrra laiem valdkond, mis sisaldab ka programmeerimist. See on tänapäeval tihedalt seotud selliste tegevustega nagu modelleerimine (süsteemide ja objektide struktuur, omadused ja tegevused nendega), rakenduste disain (andmed, kasutajaliides, protseduurid ja koostöö nende vahel) ning algoritmide koostamine. Tuleks rõhutada, et programmeerimisoskus vähemalt ühes programmeerimiskeeles – parem kui mitmes – on rakenduste loomiseks ja nende tööpõhimõtete mõistmiseks absoluutselt vajalik. Arvestades kursuse mahtu, kulub suurem osa ajast programmeerimise põhikontseptsioonide ja oskuste omandamisele.

Kursuse struktuuri ja põhikomponente kajastab allolev joonis.



Programmeerimise osa skeem tutvustab veidi modelleerimiskeele UML põhimõtteid süsteemide struktuuri kirjeldamisel, st programmeerimise osa koosneb põhimoodulist, milles kasutatakse Scratchi. Sellele võib, aga ei pea (kordus 0..1) järgnema lisamoodul.

Moodulite orienteeruvad mahud: põhimoodul 15–35 tundi, lisamoodul 0–20 tundi. Lisamooduli keele ja moodulite mahud (kokku 35 tundi) valib kool, arvestades õppijate õppesuunda, kusjuures lisamoodul võib ka puududa. Lisamooduli kasutamise üheks eesmärgiks on õppijatele tekstipõhise programmeerimiskeele tutvustamine.

Rakenduste loomise meetodeid, vahendeid ja põhifaase (ülesande püstitus, analüüs, disain ja programmeerimine) vaadeldakse ülesannete lahendamisel kogu kursuse jooksul. Läbivalt käsitletakse ka modelleerimist ja algoritmimist. Modelleerimises tutvustatakse objektorienteeritud lähenemisviisi ja unifitseeritud modelleerimiskeelt UML. Objekte, klasse ning vastavaid diagramme käsitletakse lühidalt ja lihtsustatult. Põhjalikumalt vaadeldakse protsesside ja algoritmide esitamist tegevusdiagrammide ja algoritmikeele (pseudokoodi) abil. Kursuse üldosa materjalid on eraldi failides: modelleerimine, rakenduste loomine ja algoritmimine. Nende poole tuleb pöörduda kursusejooksul korduvalt. Kohe alguses oleks otstarbekas tutvuda modelleerimise olemuse ja põhimõistetega, sest neid kasutatakse praktiliselt kõikides teistes materjalides.

Põhimooduli programmeerimiskeele valimisel on arvestatud, et programmeerimise õppimisel ei ole primaarne mitte programmeerimiskeel, vaid algoritmimine, modelleerimine ja disain. Seepärast peaks esimese keelena kasutama võimalikult lihtsa süntaksi ja struktuuriga programmeerimiskeelt, mis võimaldaks keskenduda nimetatud tegevustele ning pühenduda algoritmimise ja programmeerimise üldistele põhimõtetele ja kontseptsioonidele. Programmeerimiskeele valikul on arvestatud keele levikut ja sarnasust teiste programmeerimiskeeltega.

Põhimoodulis on programmeerimiskeeleks **Scratch**, **milles** pööratakse suurt tähelepanu atraktiivsusele ning multimeedia kasutamisele. Lihtsalt ja kiirelt on võimalik luua mänge, animatsioone, koomikseid,

esitlusi jms. Samas on võimalik mugavalt realiseerida ka arvutuslikke, andmetöötluse ja arvutigraafika algoritme.

Scratchi kodusaidil, loodud Massachusettsi Tehnoloogia Instituudis (MIT), on suurel hulgal õppematerjale, näiteid ja demosid. Programmeerimiskeele Scratchi asemel või lisaks sellele võib kasutada ka programmeerimissüsteemi BYOB (SNAP!), mis on loodud Berkeley Ülikoolis. BYOB on täielikult ühilduv Scratchiga, omades veel mõningaid täiendavaid võimalusi.

Scratch ja BYOB leiavad üha laiemat kasutamist nii koolides kui ka ülikoolides. USA arvutiõpetajate assotsiatsiooni (*CSTA*) poolt koostatud gümnaasiumite arvutiteaduse õppekava (vt õppekava) aine „Sissejuhatus programmeerimisse" tugineb täielikult Scratchile. Scratch ja BYOB leiavad kasutamist arvutiõpetuse algkursustes ka mitmes maailma juhtivas ülikoolis (nt Harvard, Berkeley). Programmeerimiskeelte populaarsust kajastavas Tiobe indeksis, kus on toodud 100 enamkasutatavat keelt, on Scratch tavaliselt esimese 30 hulgas.

Scratchi kasutajaliides, abiinfosüsteem ja kasutamisjuhend on olemas ka eesti keeles.

Lisamoodulites käsitletakse rakenduste loomist erinevate programmeerimiskeelte abil. Üldiselt ei peaks piirduma ainult ühe programmeerimiskeelega, vaid võiks tutvuda veel vähemalt ühega neist, valides sobiva lisamooduli. Tulevikus võiks lisamooduleid olla 4–5. Valikus võiksid olla mõned sellised keeled nagu: JavaScript, PHP, Visual Basic jms. Esialgu on materjalid kahe programmeerimiskeele kohta, milleks on Python ja VBA.

Python on vabavaraline üldotstarbeline objektorienteeritud lihtne ja võimas programmeerimiskeel, mis on loodud 1991. aastal. Selle populaarsus ja kasutamise ulatus (VI–VIII koht Tiobe indeksis) on kasvanud just viimastel aastatel. Pythonit kasutatakse paljudes koolides ja ülikoolides esimese programmeerimiskeelena. Pythoni üheks oluliseks omaduseks on lihtsus. Lausete struktuur on selge ja kompaktne, milles puuduvad spetsiifilised eraldajad lausete struktuuri määramiseks. Puudub väärtuste ja muutujate deklareerimine ning struktuurandmete jäiga ja fikseeritud struktuuri kirjeldamine, mis on iseloomulik enamikule programmeerimiskeeltele. Ka andmetüüpide ja andmestruktuuride käsitlemine on lihtne ja dünaamiline.

VBA (*Visual Basic for Application*) on ka paljudes koolides kasutatavas tarkvaras MS Office, pakkudes lihtsaid vahendeid rakenduste loomise, programmeerimise ja modelleerimise õppimiseks objektorienteeritud keskkonnas. VBA põhineb üldotstarbelisel programmeerimissüsteemil Visual Basic (**VB**), mis oma kasutusulatuselt on samal tasemel kui PHP ja Python. **VBA** on üks enimkasutatav vahend **dokumendipõhiste rakenduste** loomisel ja arendamisel Microsofti toodetes: Excel, Word, PowerPoint, Access, Visio jt ning ka mitmetes teiste firmade toodetes: CorelOffice, CorelDRAW, AutoCAD, Imagineer, … VBAga lähedased arendusvahendid on ka OpenOffice'is ja IBM SmartSuite ja paljudes teistes rakendustes. Käesolevas õpikus vaadeldakse VBA kasutamist Microsoft Excelis, mille eesmärgiks ei ole mitte programmeerimine Excelis, vaid Scratchis omandatud programmeerimisoskuste ja -teadmiste süvendamine. VBA ja Excel pakuvad selleks suurepäraseid võimalusi.

Objektorienteeritud modelleerimine

Objektorienteeritud (OO) lähenemisviis domineerib tarkvara, rakenduste ja infosüsteemide loomisel, arendamisel ja kasutamisel. Selle olemus seisneb reaalsete ja abstraktsete süsteemide ja objektide oleku ja käitumise modelleerimises arvutil tarkvara-objektide abil.

Objektorienteeritud lähenemisviisi olemus ja põhimõisted

Objektorienteeritud (OO) lähenemisviis domineerib tarkvara, rakenduste ja infosüsteemide loomisel, arendamisel ja kasutamisel. Selle olemus seisneb nii reaalsete kui abstraktsete süsteemide ja objektide oleku ning käitumise modelleerimises **tarkvaraobjektide** abil.

Tarkvaraobjekt on omavahel seotud andmete ja programmide (protseduuride) kogum. Arvuti süsteemi- ja rakendustarkvara põhineb samuti tarkvaraobjektidel, milleks on dokumendid, vormid, aknad, tööriistaribad, ohjurid jms. Uuemad programmeerimiskeeled on objektorienteeritud, võimaldades luua ja kasutada tarkvaraobjekte.

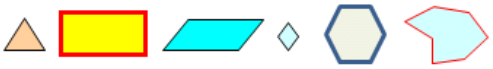
OO lähenemisviisi põhimõisted on järgmised: **objekt**, **klass**, objekti **omadused** ja **tegevused**, **seosed** objektide vahel, **sündmused.**

Objekt on piiritletud osa süsteemist ning teda võib vaadelda omaette tervikuna, milleks on **konkreetne** ese, seade, isik, hoone, asutus, fail, dokument, graafiline kujund jms. Objektid on näiteks Juku jalgratas, Pille jalgratas, Peetri arvuti, õpilane Juku Naaskel, Juku pere suvila, Juku pall, Tallinna Tehnikaülikooli kampus, must kolmnurk, konkreetne dokument jms.

Klass on ühetüübiliste objektide hulk – abstraktsioon ehk üldmõiste: jalgratas, arvuti, õpilane, suvila, hoone, ülikool, Exceli tööleht, Wordi dokument või muud.

Konkreetne objekt on klassi eksemplar (ka ilming, isend, olem).

Klassi Jalgratas eksemplarid



Klassi Hulknurk eksemplarid

Omadused (atribuudid) on karakteristikud, mis identifitseerivad objekti ja iseloomustavad selle olekut, väljanägemist, …

Kasutatavate omaduste valik sõltub rakenduse liigist, eesmärkidest, …

Konkreetse ratta atribuudid: liik, mark, mõõtmed, värv, käikude arv, olek, …

Tarkvaraobjekti ratas atribuudid müügisüsteemis: samad, mis eelmisel rattal ning lisaks veel näiteks tarnija, kauplus, …

Arvutisimulatsioonis on atribuutideks (pildi) suurus, kiirus, asukoht, …

Graafikaobjektil: laius, kõrgus, asukoht ekraanil, täitevärv, pindala, ümbermõõt, …

Tegevused (meetodid, operatsioonid), mida sooritab objekt ise või sooritatakse objektiga. Valik sõltub süsteemi liigist, eesmärkidest, …

Rataste müük: ratta lisamine või eemaldamine, andmete muutmine, …

Simulatsioon: kiiruse suurendamine, pidurdamine, peatamine, …

Graafikaobjekt: teisaldamine, kopeerimine, mõõtmete muutmine, täitevärvi muutmine, …

Sündmused. Objekt võib reageerida teatud välistele mõjudele: muudab olekut, käivitab tegevuse, …

Konkreetne ratas: kokkupõrge, …

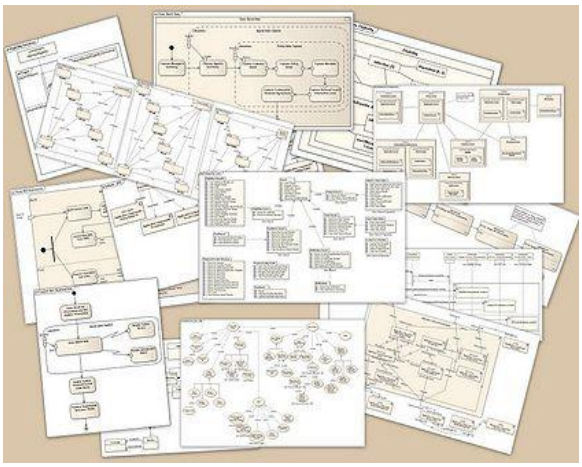
Simulatsioon: vajutus tühikule – alusta sõitu, hiireklõps – peatu jms.

Seosed (suhted, relatsioonid). Objektil võivad olla seosed teiste objektidega. Seosed on erineva olemuse ja tähendusega:

- omandisuhe: jalgratas kuulub Jukule, arvuti kuulub Peetrile,
- töösuhe: Peeter Kask töötab SEB pangas,
- sisaldavuse suhe: protsessor on arvutis, tööleht leht kuulub töövihikusse.

Unifitseeritud modelleerimiskeel UML

Objektorienteeritud graafiline keel on süsteemide visuaalseks analüüsiks, kavandamiseks, loomiseks ja dokumenteerimiseks. Keeles on kindel valik lihtsad graafilisi kujutisi (sümboleid) infosüsteemide ja rakenduste erinevate komponentide, olemite ja tegevuste tähistamiseks ning elementide ühendamiseks. Sümbolite sisse paigutatakse kindla tähenduse ja struktuuriga tekstid. Loodavate diagrammide abil saab kirjeldada süsteemi struktuuri, olekuid, tegevusi jms ning see on suhtlemisvahendiks informaatikute ja teiste erialade spetsialistide vahel.



Kasutatavate diagrammide tüübid ning nende detailiseerimise aste sõltuvad süsteemi iseloomust ja arendusprotsessi faasist: ülesande püstitus, analüüs, projekteerimine (disain) jms.

Süsteemide struktuuri, oleku ja käitumise kirjeldamiseks on erinevat tüüpi **diagramme** (skeeme ehk mudeleid):

- klassi- ja objektidiagrammid,
- tegevusdiagrammid,
- kasutusjuhtude diagrammid,
- olekudiagrammid,
- …

Peamised UML diagrammid ülesande püstituse, analüüsi ja disaini faasis on **klassi**- ja **tegevusdiagrammid**.

Klassid ja objektid

UMLi diagrammidel esitatakse klass ristkülikuna, millel on üldjuhul on kolm sektsiooni:

- nimi,
- atribuudid (omadused),
- operatsioonid (tegevused, meetodid).

Nendest kohustuslik on ainult nimi.

Järgnevas näites on toodud kooli infosüsteemi klassi **Õpilane** lihtsustatud kirjeldus.

Õpilane
kood eesnimi perenimi sünniaeg aadress klass …
lisamine() eemaldamine() muutmine() …..

Klassi nimi. Nimi peab algama suurtähega.

Atribuudid (**omadused** ehk **parameetrid**)

Igal atribuudil on **nimi**. Võib olla näidatud ka **tüüp**: arv, tekst, kuupäev jm.

Konkreetsetel objektidel on omadusel kindel väärtus:

K20567, Juku, Naaskel, 26.10.2001, Spordi 5-9, 4B, …

K20573, Pille, Pilt, 23.06.2001, Teatri 7-13, 4A, …

Operatsioonid (**tegevused** ehk **meetodid**) Esitatakse tavaliselt kujul **nimi()**.

Siin on meetodite tähendus järgmine: **lisamine()** – uue õpilase lisamine, **eemaldamine()** – õpilase eemaldamine, **muutmine()** – õpilase andmete muutmine.

Arvuti
mark protsessor taktsagedus sisemälu_maht kõvaketta_maht hind / arv …
lisamine() eemaldamine() muutmine() ...

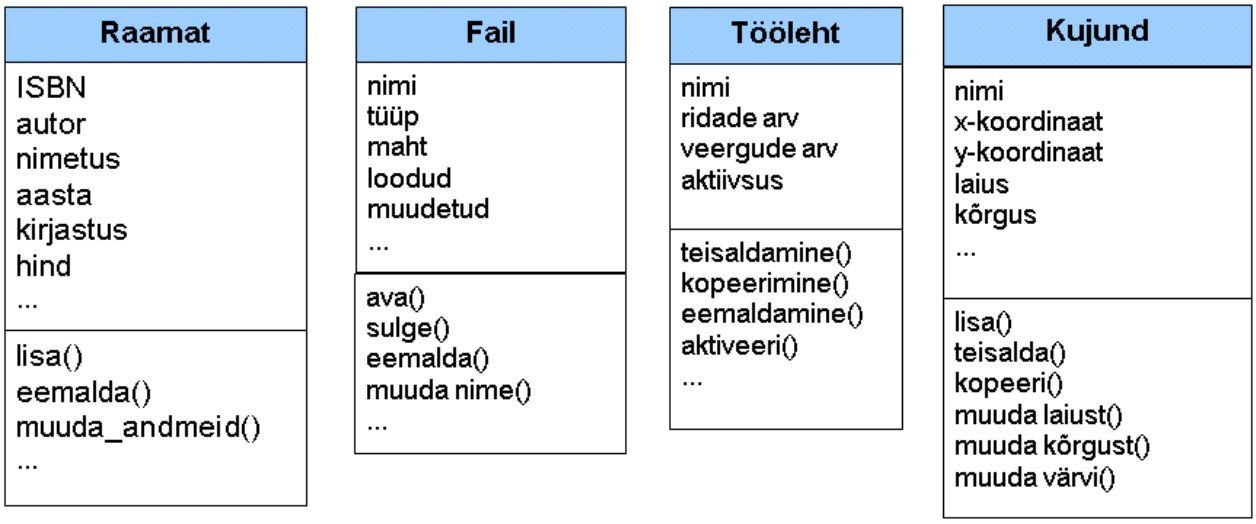
Siin on veel ühe klassi kirjelduse näide: klass **Arvuti** müügisüsteemis.

Üheks omaduseks on olemasolevate arvutite arv antud grupis.
Tegemist on nn muutuva ehk arvutatava atribuudiga. Sama marki arvutite lisamisel või eemaldamisel muudetakse omadust arv. Tuletatava omaduse tunnusena on kaldkriips (/) omaduse nimetuse ees.

Allpool on toodud näiteks tabel, mis sisaldab andmeid arvutite (objektid) kohta: arvutite omadusi.

Mark	**Protsessori tüüp**	**Taktsagedus**	**Sisemälu, MB**	**Kõvaketas, GB**	**Hind, €**	**Arv**
Aragorn	AMD Athlon64	3,0	512	160	416	13
Balrog	Intel Celeron	2,8	256	80	349	21
Eldar	Intel Xeon	3,0	512	160	388	7
Elend	Intel Celeron	4,0	2048	400	1083	0
Faram	AMD Athlon64	3,0	512	210	499	12
Frodo	AMD Athlon64	3,0	4072	500	1108	25
Gandalf	Intel Xeon	3,2	512	160	708	2
Kalvar	Intel Xeon	3,0	512	160	583	7
Marvin	Intel Celeron	3,0	512	160	499	0
Rohan	AMD Athlon64	3,5	512	160	491	0
Sauron	Intel Xeon	2,5	256	180	283	5

Järgnevad on veel mõned klasside kirjeldamise näited.



Klass Ristkülik süsteemis Geomeetria:

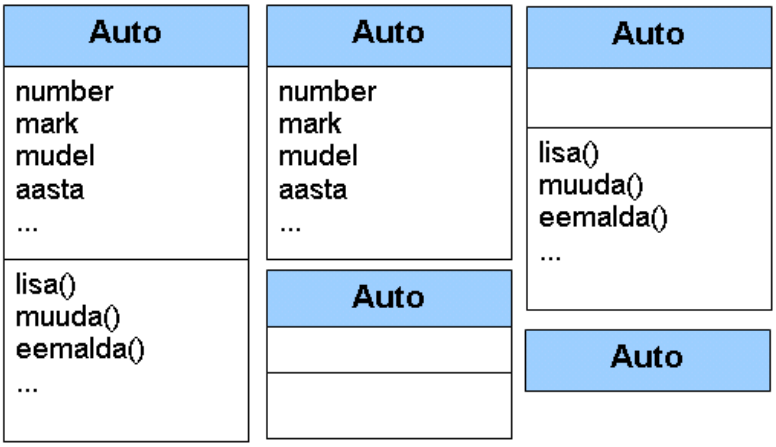
Ristkülik
laius kõrgus **/** pindala **/** ümbermõõt **/** diagonaal ...
leia_pindala() leia_ümber() leia_diagonaal() muuda() ...

$$S = a \cdot b$$
$$P = 2(a + b)$$
$$d = \sqrt{a^2 + b^2}$$

NB! Sisestatavad andmed: laius (a) ja kõrgus (b) ning
tuletatavad omadused: pindala (S), ümbermõõt (P) ja diagonaal (d).
Viimaste ees võib tuletatava omaduse tunnusena olla **/** (kaldkriips), aga pole kohustuslik.

Klassisümbolite erinevad esitusviisid

Antud juhul võib kasutada erinevaid variante: tühjad sektsioonid, ärajäetud sektsioonid jms, kuid sageli kasutataksegi ainult nime (diagrammidel).



Seosed ja klassidiagrammid

Süsteemi kuuluvatel objektidel on omavahel teatud seosed – relatsioonid. Seoseid (relatsioon, suhe) näidatakse diagrammidel joonte abil. Klassidiagramm näitab klasse ja võimalikke seoseid nende objektide vahel. Reeglina on igas seoses ainult kaks klassi. Seosele võib anda nime, mida võib näidata skeemil. Seose mõlema otsa juures võib esitada korduse, mis näitab, mitu antud klassi objekti võib olla seotud teise klassi objektiga. Korduste tüüpvariandid ja nende esitus UMLis on järgmised:

n – kindel arv, näiteks 1, 4, 256

n1, n2, n3, ... – loend, näiteks 2, 4, 6

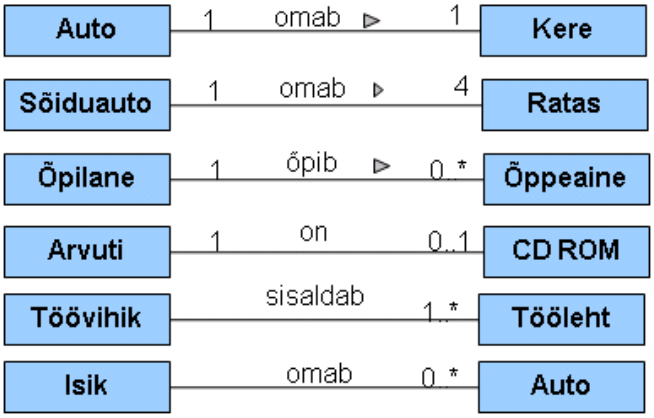
0..1 või 0, 1 – null või üks

* – suvaline hulk

1..* – üks või rohkem

0..* – null või rohkem

Seose nime ja korduse väärtuse 1 võib jätta ära.



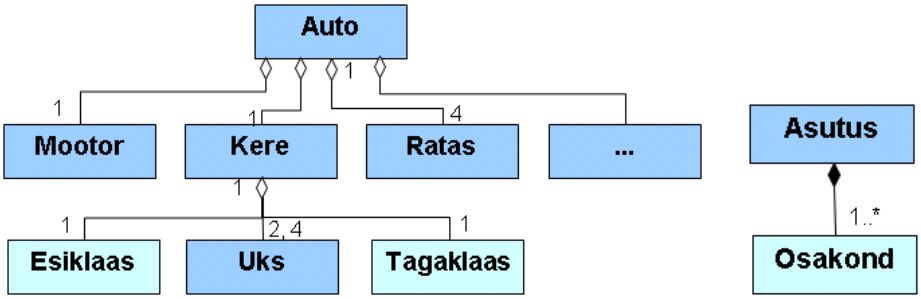
Eristatakse järgmisi seoste liike:

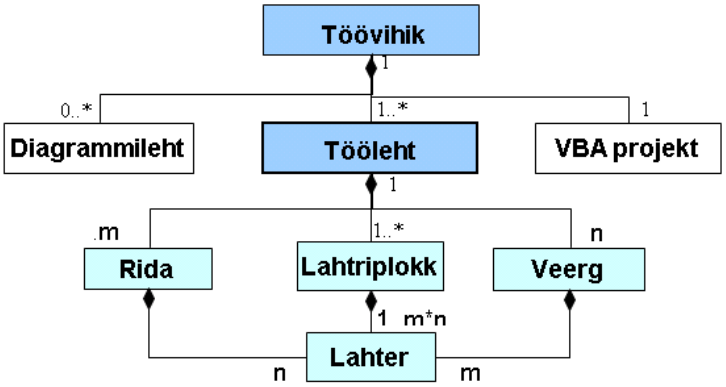
- agregatsioon
- assotsiatsioon
- üldistus

Agregatsioon ja kompositsioon

Agregatsioon ehk sisalduvuse seos. Üks objekt sisaldab teisi. Tervik ja osad. Terviku poolel võib olla täitmata romb ◊ (seda ei pea tingimata olema).

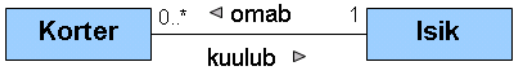
Kompositsioon – agregatsiooni erijuht – osa saab eksisteerida ainult tervikus, mille tähistuseks on täidetud romb ♦. Vt järgnevat skeemi.

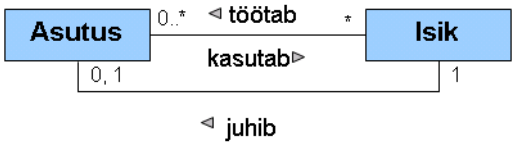




Assotsiatsioon

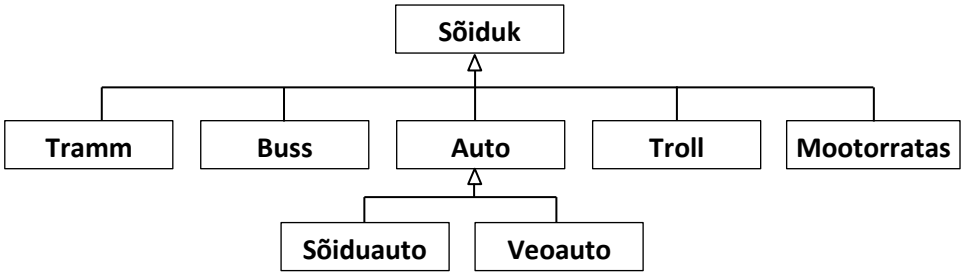
Assotsiatsioon – suvaline mõtet omav seos (suhe) objektide vahel – omandisuhe, töösuhe jms. Assotsiatsiooni erijuht on agregatsioon.



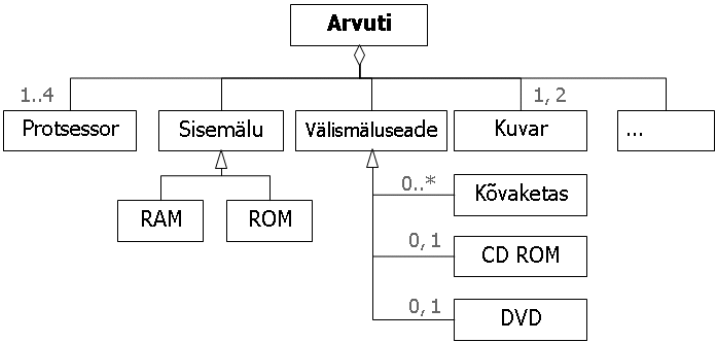


Üldistus (*generalization*)

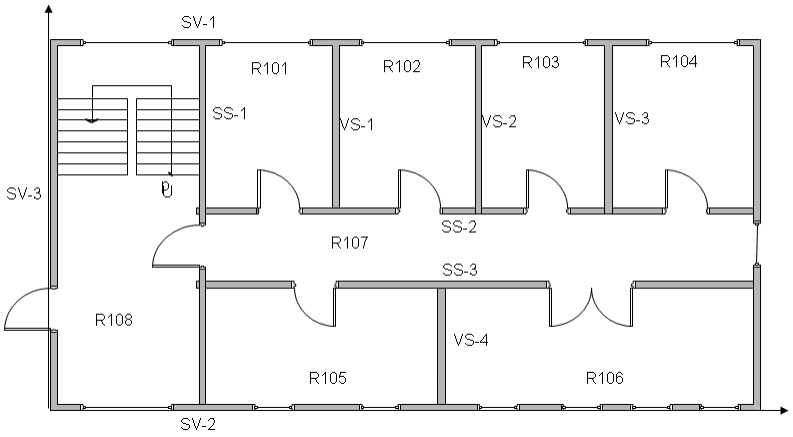
Üldistus – esivanema (ülemklassi) ja järglaste (alamklasside) seos. Järglased pärivad esivanema omadused ning võivad omada täiendavaid omadusi. Kujund △ – esivanema poole.

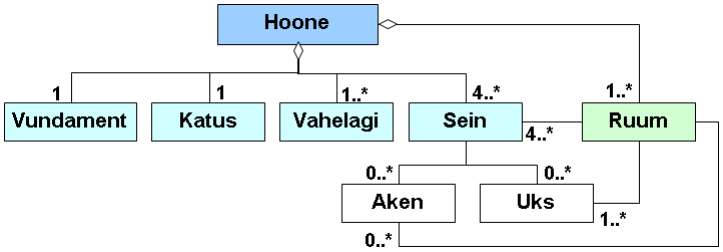


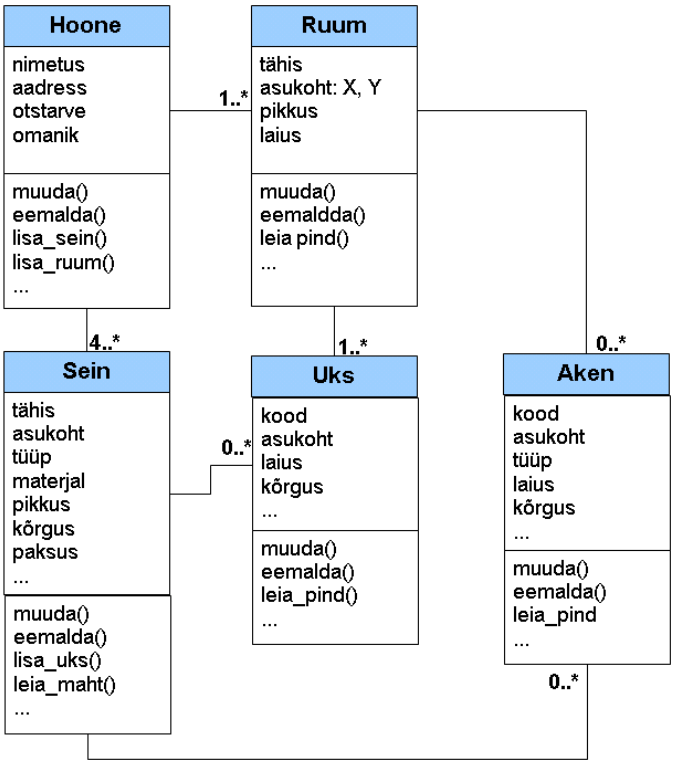
Veel mõned klassidiagrammide näited:



Hoone põhiklassid:



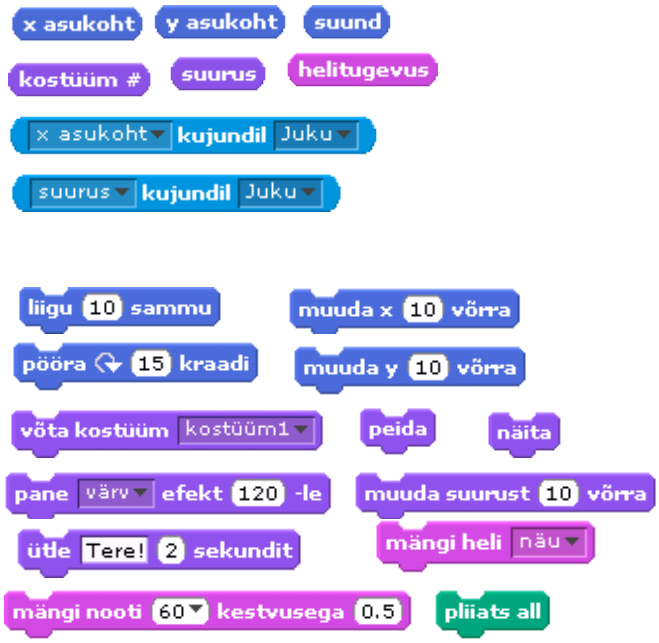




Objektid ja klassid süsteemis Scratch

Klass Sprait Scratchis

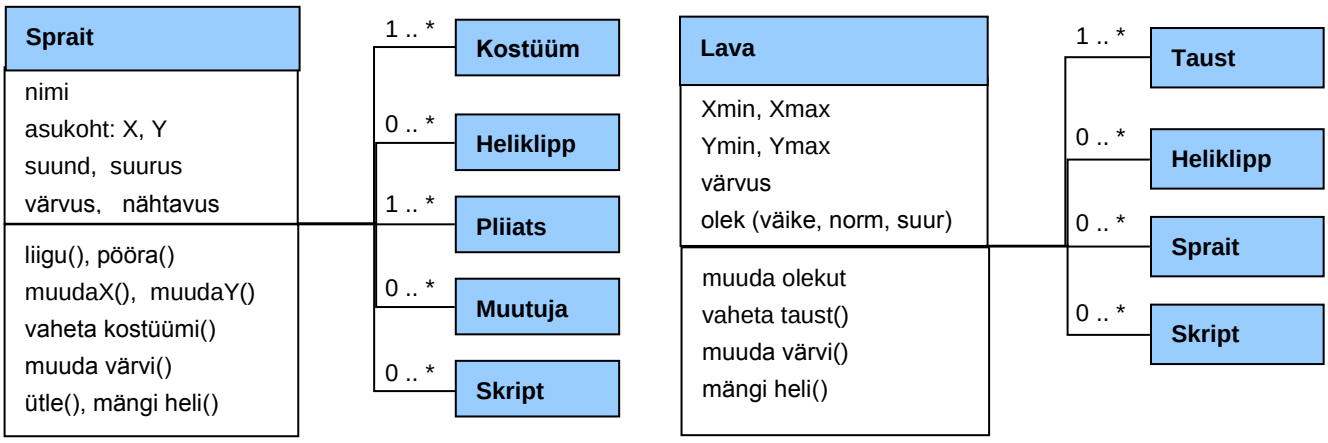
Sprait
nimi asukoht: x,y suund suurus värvus nähtavus
liigu() pööra() muudaX() muudaY() vaheta kostüümi() muuda värvi() ütle() mängi heli() ...



Sprait (*sprite*) on universaalne graafikaobjektide klass. Spraitidel on kindel valik **omadusi**: nimi, asukoht, suurus, jms. Spraitidega saab **käskude** ehk **plokkide** (**meetodite**) abil määrata erinevaid tegevusi: liigutada, pöörata, muuta suurust jms. Käskudest saab moodustada programmiüksuseid – **skripte.**

Lava on ekraani piirkond, kus toimub spraitide tegevus.

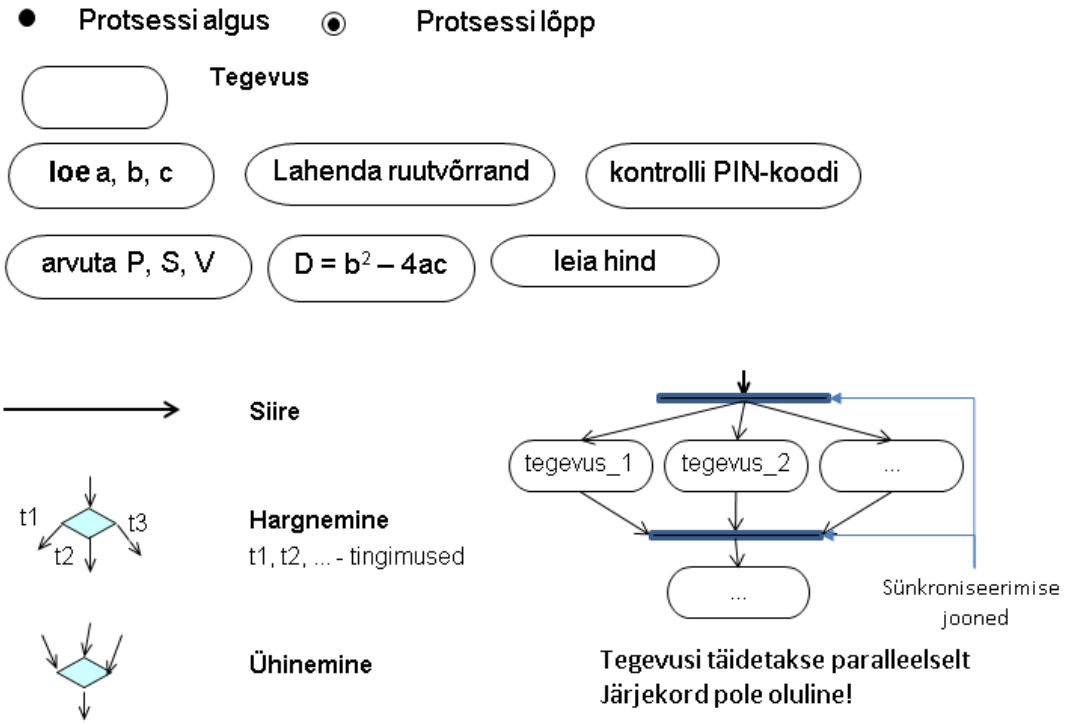
Klass Sprait ja sellega seotud klassid:



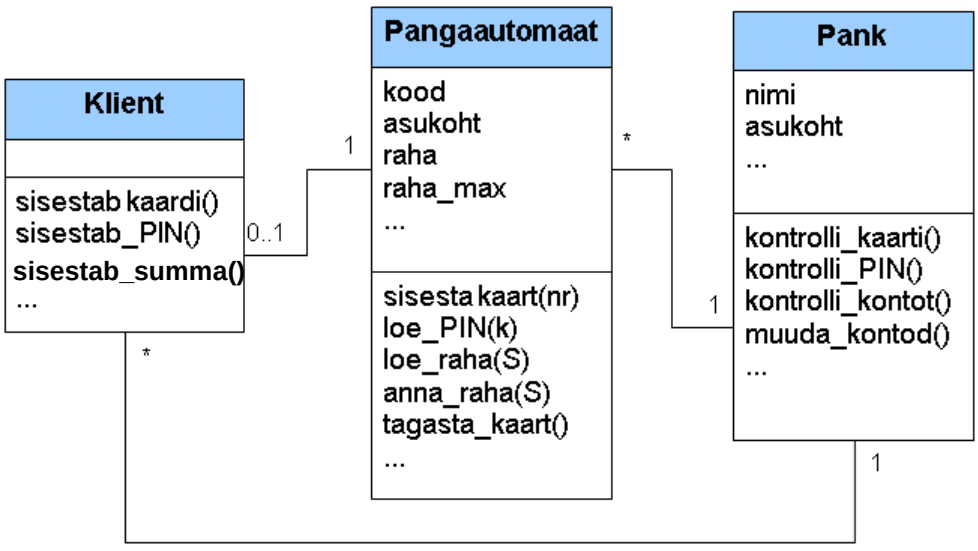
UML tegevusskeemid

Tegevusskeem ehk tegevusdiagramm (*Activity Diagram*) esitab graafiliselt antud protsessi või töö täitmiseks vajalikke tegevusi ja nende järjekorda. Tegevusskeemi kasutatakse äri- ja tööprotsesside ning algoritmide kirjeldamiseks.

Standardkujundid (sümbolid)

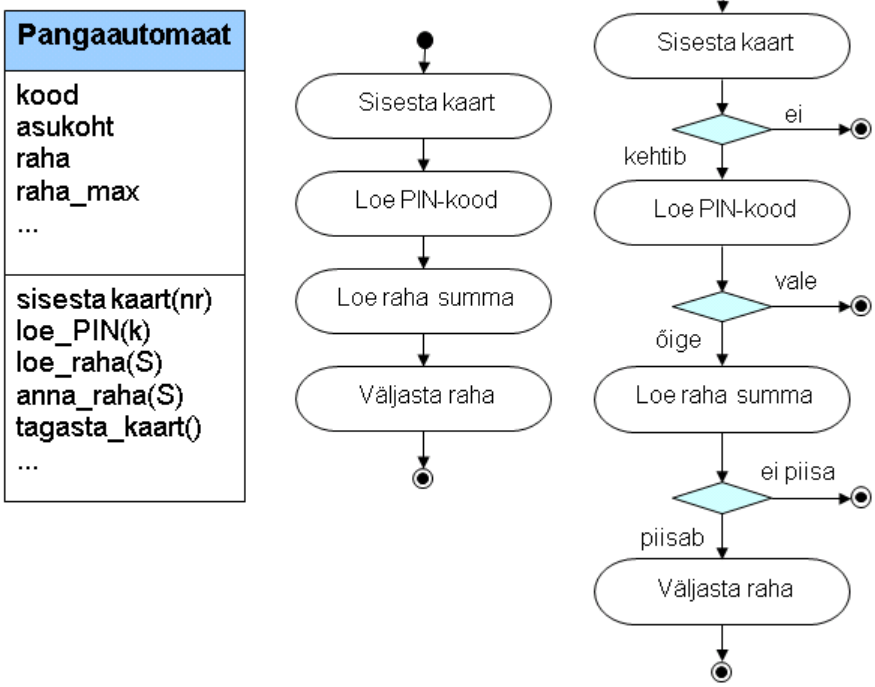


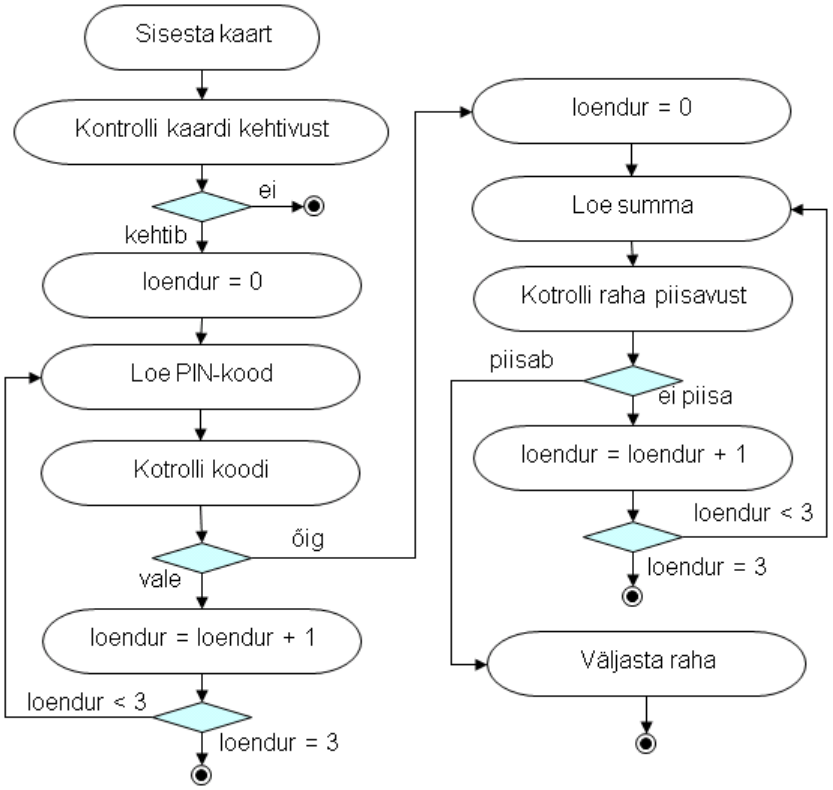
Pangaautomaatide süsteem



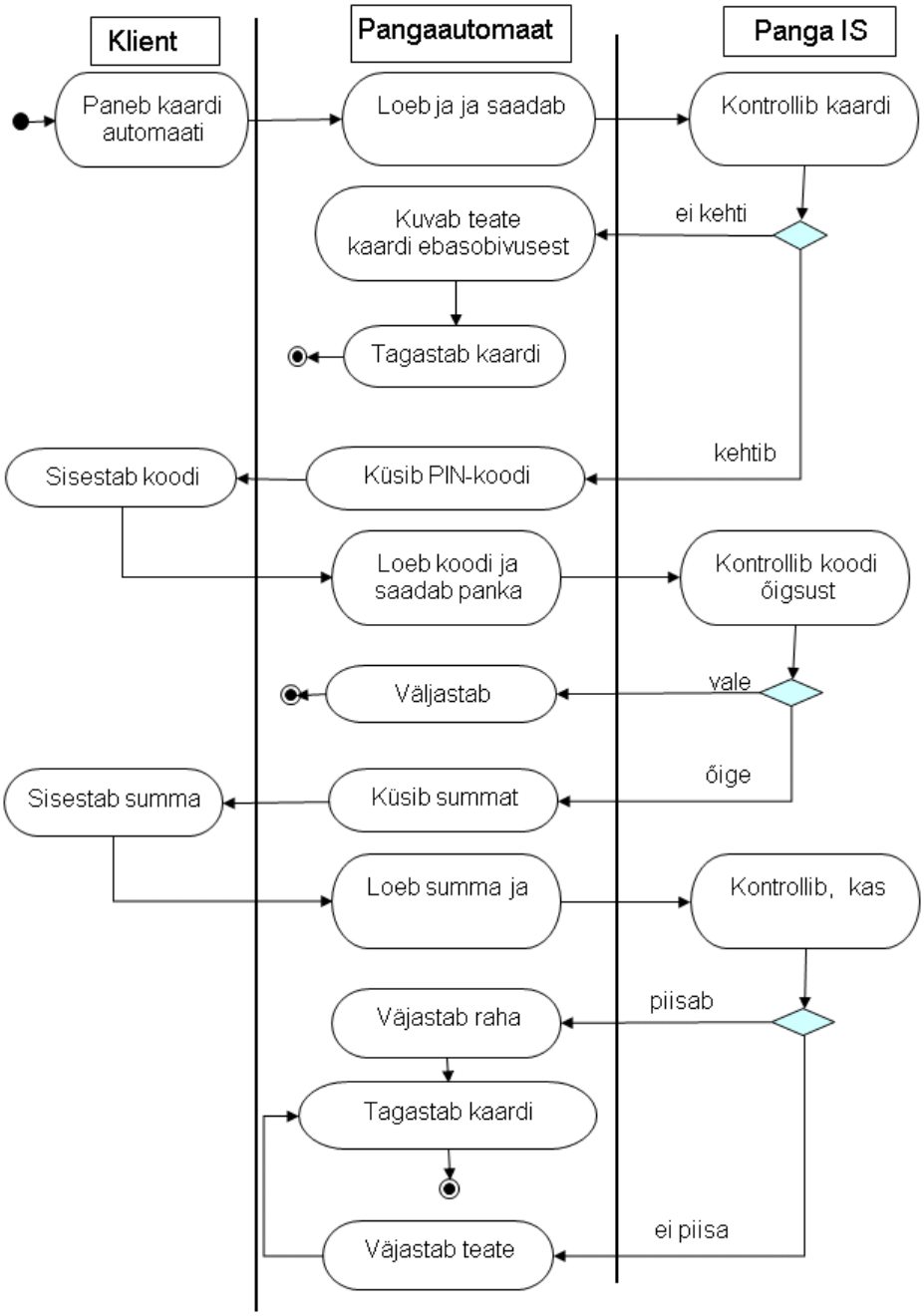
Pangaautomaat

Klass ja erineva täpsusega tegevusskeemid:



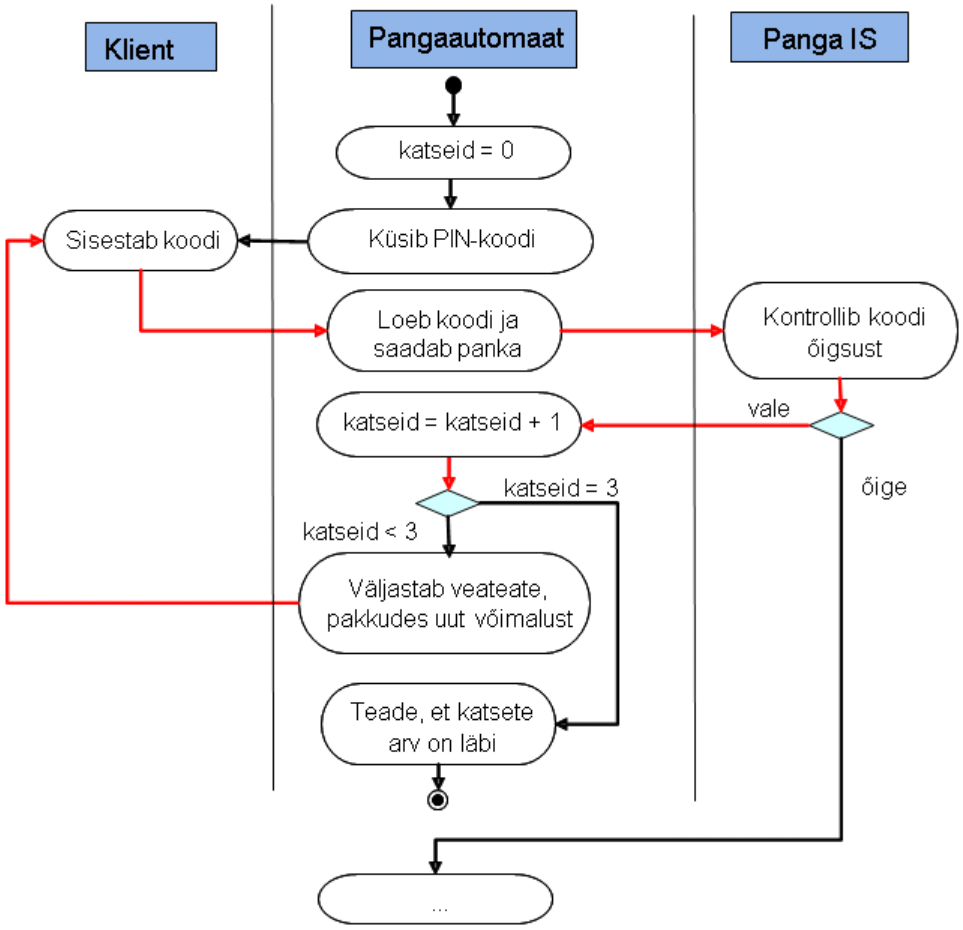


Tegevuste jaotus objektide vahel

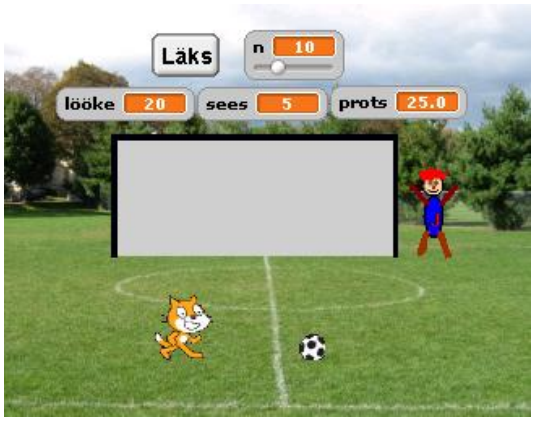


Pangaautomaat. PIN koodi kontroll

Lubatud on kolm katset:



Rakendused



Rakendus (*Application*) on tarkvara kogum antud liiki ülesannete lahendamiseks või tegevuste täitmiseks arvutil. Samas tähenduses kasutatakse ka mõisteid programm, rakendusprogramm, programmipakett.

Rakenduse olemus

Rakendus (*Application*) on tarkvara kogum antud liiki ülesannete lahendamiseks või tegevuste täitmiseks arvutil. Samas tähenduses kasutatakse ka lähedasi mõisteid: programm, rakendusprogramm, programmipakett.

Eristada võib rakenduste kahte põhiliiki:

- üldotstarbeline rakendustarkvara (rakendusprogrammid): veebilehitsejad, tekstitöötluse-, tabeli-, graafikaprogrammid, …
- spetsialiseeritud rakendused (erineva otstarbe ja mahuga, realiseeritakse erinevate vahenditega ja erinevates keskkondades): võrrandite lahendamine, palgaarvestus, detailide projekteerimine, mängud, …

Peamised tegevused rakenduste loomisel:

- modelleerimine
- analüüs
- projekteerimine (disain)
- algoritmimine
- programmeerimine (kodeerimine)
- testimine
- dokumenteerimine

Rakenduse põhikomponendid

Rakenduses on üldjuhul omavahel tihedalt seotud:

- programmid
- andmed ja objektid
- kasutajaliides

Programm

Programm on käskude (korralduste, instruktsioonide, **lausete**) kogum, mis määrab, milliseid tegevusi peab arvuti täitma **andmete** ja **objektidega**, tagades tavaliselt ka kasutajaliidese töö.

Programmide koostamiseks kasutatakse spetsiaalseid **programmeerimiskeeli** ja -**süsteeme**. Igas keeles on piiratud valik **käske** ehk **lauseid** ning nende esitamiseks ja täitmiseks on kindlad reeglid. Programm võib koosneda mitmest üksusest. Erinevates keeltes kasutatakse nende jaoks erinevaid nimetusi: **protseduur**, **funktsioon**, **skript**.

Tänapäeva arvutid saavad täita programme, mis on esitatud antud arvutitüübi jaoks ettenähtud **masinkeeles**. See koosneb kahendsüsteemis esitatud elementaarsetest käskudest nagu liitmine, lahutamine, võrdlemine jms. Programmide tõlkimiseks masinkeelde kasutatakse spetsiaalseid programme, mida nimetatakse **translaatoriteks**. On kahte liiki translaatoreid – **kompilaatorid** ja **interpretaatorid**. Kompilaator tõlgib (transleerib) terve programmi enne täitmist masinkeelde. Interpretaator tõlgib programmi täitmise ajal. Millist transleerimise viisi kasutatakse, sõltub keelest ja programmeerimissüsteemist. Mitmed süsteemid võimaldavad mõlemat varianti: rakenduse loomise ajal kasutatakse interpretaatorit, valmisprogramm aga transleeritakse masinkeelde, sest selline programm töötab kiiremini ja ei vaja enam programmeerimissüsteemi, millega ta on loodud. Translaator on programmeerimissüsteemi üheks peamiseks osaks. Lisaks translaatorile sisaldab see tavaliselt spetsiaalset **redaktorit**, mille abil sisestatakse ja redigeeritakse programme, vahendeid kasutajaliideste loomiseks, silumise ja testimise vahendeid jms.

Järgnevas näites on toodud sarnase sisuga programmid kolmes programmeerimiskeeles: Scratch, Visual Basic ja Python. Programmid esitavad kasutaja poolt etteantud arvu (**n**) korrutamisülesandeid, hindavad vastuseid ja annavad üldhinnangu tulemustele.


```
Sub Korrutamine()
 Dim a, b, n, vastus, vigu, k
 n = InputBox("Mitu ülesannet?")
 vigu = 0
 For k = 1 To n
   a = Int(2 + Rnd() * 8)
   b = Int(2 + Rnd() * 8)
   vastus = InputBox(a &" * "&b)
   If Int(vastus) <> a * b Then
      MsgBox "Vale!"
      vigu = vigu + 1
   End If
Next k
If vigu = 0 Then
  MsgBox "Tubli! Kõik oli õige!"
  Call Tubli (Shapes("Juku"),3)
 Else
    MsgBox "Vigu oli " & vigu
End If
End Sub
```

```
import random
def Korrutamine() :
  n = int(input("Mitu ?"))
  vigu = 0
  for k in range(n) :
     a = random.randint(2, 9)
     b = random.randint(2, 9)
     t = str(a) + " * " + str(b)
     vastus = input(t + "=>")
     if int(vastus) != a * b :
        print ("Vale!")
        vigu += 1
  if vigu == 0 :
     print ("Kõik on õige!")
  else :
     print ("Vigu oli ", vigu)

Korrutamine()
```


```
Sub Tubli(kuju, n)
  Dim i
  For i = 1 To n
    kuju.IncrementTop -30
    paus 0.3
    kuju.IncrementTop 30
    paus 0.5
  Next i
End Sub
```

Lisaks põhitegevusele on Scratchis ja VBAs kasutusel protseduur, mis realiseerib lihtsa animatsiooni (graafika-objekt teeb mõned hüpped), kui kasutaja on kõik õigesti vastanud.

Programmid on väga tihedalt seotud **andmetega**, sest kasutavate andmete liigist ja organisatsioonist sõltuvad otseselt programmi sisu ja kasutatavad vahendid. Järgnevalt esitletakse andmeid põhjalikumalt. Programminäited on toodud vaid Scratchis ja Visual Basicus.

Andmed

Andmed on informatsiooni esitus kujul, mis võimaldab seda säilitada, töödelda ja edastada programmjuhtimisega süsteemide (eeskätt arvutite) abil. Kasutatakse erinevat liiki andmeid ja neil võib olla erinev organisatsioon.

Andmete liigid

Peamised andmete liigid on

- märkandmed: arvud, tekstid, …
- graafikaandmed: pildid, joonised, diagrammid, …
- heliandmed: kõne, muusika, …

Liigist sõltub andmete esitus arvuti seadmetes ning võimalikud tehted (operatsioonid) ja tegevused nendega. Kõik andmed esitatakse arvuti mälus kodeerituna – kahendarvudena.

Märkandmeid kasutatakse praktiliselt kõikides rakendustes. Nende esitamiseks arvutis on esmaseks märkhaaval kodeerimine kahendarvude abil: igale märgile (täht, number jms) vastab kindel kahendarv (näiteks A on 01000001, B on 01000010, 3 on 00110011, + on 00101011). Kaheks peamiseks **kodeerimissüsteemiks** on ASCII ja UNICODE. Arvud, millega tehakse matemaatilisi operatsioone, võivad olla salvestatud spetsiaalsetes vormingutes: täisarvud**,** püsikomaarvud ja ujukomaarvud (reaalarvud).

Graafikaandmete loomiseks ja redigeerimiseks saab kasutada erinevaid programme. On olemas terve hulk spetsiaalseid graafika-, pilditöötluse- ja joonestusprogramme (Paint, GIMP, Adobe Photoshop, AutoCAD, …), mille töö tulemusi saab salvestada erinevates vormingutes failidesse. Sõltuvalt vormingust on graafikafailidel kindlad tüübitunnused nagu PNG, JPEG, GIF, PSD, DWG, …

Graafikaobjekte saab luua ja kasutada ka paljudes üldotstarbelistes rakendusprogrammides (nt tekstitöötluse-, esitluse- ja tabeliprogrammides) või neid importida ning lisada siis dokumenti.

Loodavates rakendustes saab graafikaandmeid importida olemasolevatest graafikafailidest kas rakenduse loomise või täitmise ajal. Graafikaobjektideks on näiteks ka Scratchi spraitide kostüümid ja lava taustad. Mitmed programmeerimissüsteemid võimaldavad luua graafilisi kujutisi ka vahetult täitmise ajal. Scratchis saab näiteks joonistusi teha grupi Pliiats ja Liikumine käskudega.

Heliandmed salvestatakse samuti erinevates failivormingutes – WAV, MP3, MID, … Heliklippe saab kasutada mitmetes rakendustes. Programme on ka heli salvestamiseks ning muusika loomiseks ja redigeerimiseks.

Andmete organisatsioon

Organisatsiooni järgi võib andmed jagada järgmistesse gruppidesse:

- muutujad ehk mäluväljad,
- andmekogumid,
- objektid.

Muutujad

Muutuja ehk **mäluväli** (öeldakse ka mälupesa jms) on koht arvuti mälus, mis koosneb teatud hulgast baitidest, kuhu programm saab salvestada mingi **väärtuse**: arvu, teksti jms. **Viitamiseks** väljale (muutujale) kasutatakse programmis selle **nime**.

Muutujat võib käsitleda objektina.

Nimetus **muutuja** tekkis programmeerimiskeelte loomise algaastatel, mil arvutitel lahendati peamiselt matemaatilisi ülesandeid ja kajastab seda, et mäluvälju kasutatakse muutuvate suuruste väärtuste salvestamiseks, st välja väärtused võivad olla erinevad.

Rakenduse koostajale ei pruugi olla tähtis, kus asub väli või mis kujul täpselt salvestatakse selles väärtused ja milline on välja pikkus (baitide arv väljas). Olgu öeldud, et viimane omadus võib siiski mõnikord olla oluline ning programmi koostamisel saab seda teatud määral reguleerida.

Muutujaid võib ette kujutada kui nimega tähistatud lahtrit tabelis või kirje väljana.

eesnimi	perenimi	vanus	pikkus	kaal	
Juku	Naaskel	10	153	42	…

eesnimi	perenimi	vanus	pikkus	kaal
Juku	Naaskel	10	153	42

Programmides **Korrutamine** (vt lk 29) on kasutusel kuus muutujat: **a, b, k, n, vastus, vigu.**

Omadused (atribuudid)

Nimi identifitseerib mälupesa (muutuja). Enamikes programmeerimiskeeltes on nimede esitamiseks üsna täpsed reeglid. Enamasti nõutakse, et nimi võib sisaldada ainult **tähti** ja **numbreid** ning peab algama tähega. Erandina on lubatud kasutada **allkriipsu** tühiku asendajana mitmeosalistes nimedes, sest tühikute kasutamine nimedes on tavaliselt keelatud. Füüsilisel tasemel on muutuja üheks omaduseks ka aadress – välja esimese baidi aadress.

Muutuja
nimi väärtus väärtuste tüüp välja pikkus skoop
loomine() väärtuse omistamine() väärtuse lugemine() eemaldamine()

Väärtus on mäluväljas salvestatud muutuja väärtus (arv, tekst, …). Igal ajahetkel saab muutujal olla ainult üks väärtus.

Väärtus tekib või muutub siis, kui see omistatakse muutujale programmi täitmisel vastava käsu toimel. Suureks erandiks on selles osas Scratch, kus saab muutujale väärtuse omistada ka enne programmi täitmist, kasutades muutuja monitori liugurit.

Iga muutujaga saab programmis seostada tüübi: tekst, täisarv, reaalarv, tõeväärtus jms. Tüübist sõltub ka välja pikkus. Võimalik valik sõltub programmeerimiskeelest. Paljudes keeltes (Java, C, Pascal) on tüübi määramine kohustuslik, kuid mitmes keeles ei ole see vajalik (Visual Basic) või pole isegi võimalik (Scratch, JavaScript, Python). Kui muutuja tüübi määramine ei ole vajalik (tüüp on määramata), valib väärtuste esitusviisi ja väljade pikkused translaator. Programmi koostamisel (näiteks väärtuste leidmist kirjeldavates avaldistes) peab aga arvestama muutujate väärtuste liigiga (arv, tekst, tõeväärtus).

Skoop on muutuja **määramispiirkond**. Enamikes programmeerimissüsteemides on muutujal (aga näiteks ka andmekogumil) vähemalt kaks võimalikku skoopi – terve rakendus või konkreetne protseduur või objekt, ning räägitakse globaalsetest ja lokaalsetest muutujatest. Esimesel juhul saab muuta ning kasutada muutuja väärtusi kõikides programmi protseduurides või skriptides, teisel juhul on muutujale juurdepääs ainult kindlas protseduuris või objektis.

Tegevused

Loomine: muutuja loomine seisneb sellele mäluvälja eraldamises arvuti mälus. Enamikes programmeerimiskeeltes toimub see programmi täitmise ajal. Erandiks on Scratch, kus muutujad luuakse enne programmi täitmist.

Eemaldamine: muutuja eemaldamine seisneb mälu vabastamises. Enamike süsteemide korral toimub see automaatselt programmi või protseduuri töö lõppemisel. Mõned keeled võimaldavad eemaldada muutujaid täitmise ajal. Scratchis saab muutujaid eemaldada nö „käsitsi“.

Väärtuse omistamine ja **väärtuse lugemine** on kaks põhitegevust, mida arvuti saab täita. Väärtuse omistamine seisneb mingi kindla väärtuse salvestamises antud muutuja väljas. Tüüpiliselt eelneb sellele väärtuse tuletamine (leidmine) etteantud avaldise alusel või lugemine väliskeskkonnast. Üheks peamiseks vahendiks väärtuse leidmisel ja salvestamisel on **omistamislause**, millel on enamikes programmeerimis-keeltes järgmine kuju:

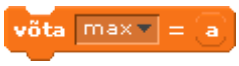
muutuja = avaldis

Scratchis kasutatakse **omistamisplokki**: 

Siin on **muutuja** esindatud oma nimega, märki „=“ nimetatakse **omistamissümboliks** (mitte võrdusmärgiks), **avaldis** määrab väärtuse leidmise eeskirja. Avaldise väärtuse leidmisel kasutatakse tavaliselt konstante ja muutujate väärtusi. Avaldise erijuhuks on ka konstant ja muutuja.

S = 0: k = 1: nimi = „Juku Naaskel“

siin omistatakse muutujatele konstantide väärtused, st salvestatakse need vasakus pooles näidatud muutujate väljades

max = a 

loetakse muutuja **a** väärtus ja salvestatakse see muutujas **max** (toimub kopeerimine)

D = b * b – 4 * a * c

loetakse muutujate **a**, **b** ja **c** väärtused,leitakse avaldise väärtus ja omistatakse muutujale **D,** st salvestatakse selle väljas

k = k + 1 

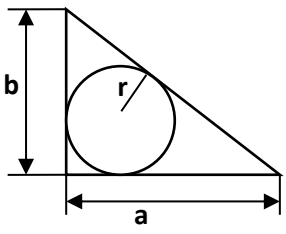
loetakse muutuja **k** väärtus, liidetakse sellele **1** ja tulemus salvestatakse tagasi muutujasse **k**

n = n – 5 

loetakse muutuja **n** väärtus, lahutatakse sellest **5**, tulemus salvestatakse tagasi muutujasse **n**

Siin toodud näidetes esinevad avaldistes lisaks muutujatele ka **konstandid**. Konstandi väärtus kirjutatakse vahetult programmi lausesse (avaldisse).

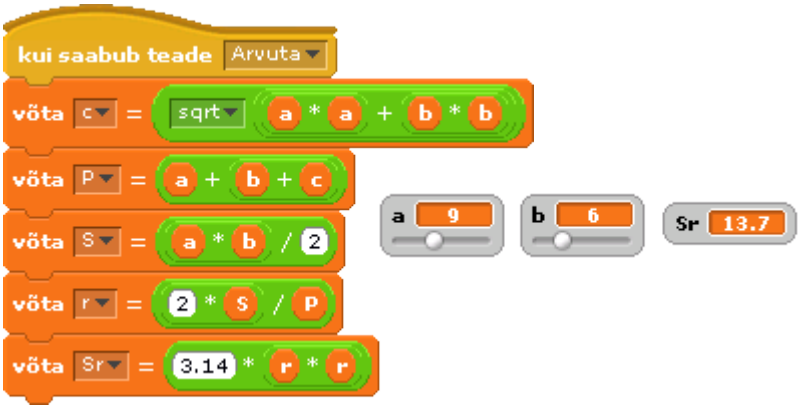
Näide: Kolmnurga siseringi pindala leidmine



Näites on toodud arvutusliku iseloomuga Scratchi ja Visual Basicu programmid. Need leiavad täisnurkse kolmnurga siseringi pindala (**Sr**) antud kaatetite pikkuste (**a, b**) järgi:

$$c = \sqrt{a^2 + b^2}; \quad P = a + b + c; \quad S = a \cdot b / 2; \quad r = 2 \cdot S / P; \quad Sr = \pi \cdot r^2$$

Põhiosa programmides koosneb omistamislausete jadast. Mõlemas programmis on kasutusel muutujad **a, b, c, P, S, r, Sr**. VBA protseduuris on kasutatud muutujate kirjeldamiseks Dim lauset, muutujatele tüüp võetakse automaatselt.




```
Sub Arvuta ()
  Dim a, b, c, p, S, r, Sr
   a = Val(InputBox ("anna a"))
   b = Val(InputBox ("anna b"))
   c = Sqr (a * a + b * b)
   P = a + b + c
   S = a * b / 2
   r = 2 * S / P
   Sr = 3.14 * r * r
   MsgBox "Pindala= " & Sr
End Sub
```

Scratchi skriptis eeldatakse, et muutujate **a** ja **b** väärtusi muudetakse liugurite abil ning tulemus (siseringi pindala) kuvatakse muutuja **Sr** monitoris.

Visual Basic programm loeb sisendboksidest ja salvestab vastavates muutujates kaatetite pikkused (a, b). Järgnevad omistamislaused leiavad ja salvestavad vastavates muutujates hüpotenuusi (c) ümbermõõdu (P), kolmnurga pindala (S), siseringi raadiuse (r) ja siseringi pindala (Sr). Lause **MsgBox** kuvab muutuja Sr väärtuse teateboksis.

Andmekogumid

Andmekogumeid nagu loendid, tabelid, massiivid, kirjed, kollektsioonid, nimistud jms, saab moodustada ja kasutada (sõltuvalt programmeerimissüsteemist) nii märk-, graafika- kui ka heliandmete jaoks. Eriti iseloomulik on andmekogumite kasutamine märkandmete puhul. Siin tutvume kahe lihtsama ja enam-kasutatava kogumiliigiga: **loendid** ja **tabelid**.

Loendid

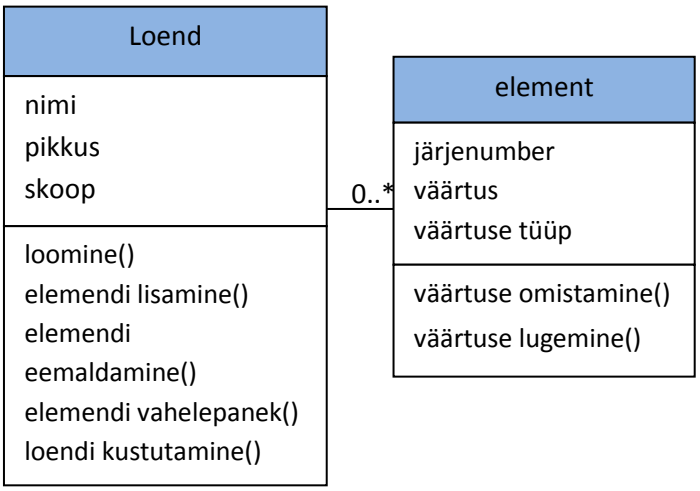
Loend ehk ühemõõtmeline massiiv (öeldakse ka **järjend** või **vektor**) kujutab endast järjestatud elementide (muutujate) kogumit, mis tähistatakse programmis ühe nimega. Viitamiseks elementidele kasutatakse loendi nime koos **indeksiga** (järjenumbriga). Näites on toodud kaks loendit. **T** sisaldab töötajate nimesid, **Palk** nende töötajate palkasid.

T

Kask	**Kuusk**	**Mänd**	**Paju**	**Saar**	**Tamm**	**Vaher**			
1	2	3	4	5	6	7	8	9	10
920	**670**	**1120**	**990**	**1040**	**1230**	**848**			

Palk

Loendi **põhiomadusteks** on **nimi**, **pikkus** ja **skoop**. Nimede jaoks kehtivad enamikes programmeerimiskeeltes samad reeglid nagu muutujate jaoks. Loendi pikkuseks on elementide arv selles. Loendite elementide numeratsioon algab tavaliselt ühest või nullist. Mõnedes programmeerimiskeeltes saab kirjel-duses määrata indeksi minimaalse ja maksi-maalse väärtuse. Scratchis on indeksi minimaalne väärtus alati 1, maksimaalne väärtus ei ole piiratud. Skoobi tähendus on samasugune nagu muutujal.



Loendi (massiivi) loomine seisneb selle elementidele mäluväljade eraldamises, mis tüüpiliselt tehakse programmis oleva kirjelduse (deklaratsiooni) alusel programmi täitmise ajal. Erandiks on jällegi Scratch, kus loendi loomine toimub „käsitsi". Esialgu elemendid puuduvad. Elementide lisamise, eemaldamise ja vahelepaneku käske ei pruugi igas keeles olla. Sel juhul tuleb taolised operatsioonid eraldi programmeerida. Scratchis on need käsud olemas.

Loendi element vastab oma olemuselt muutujale – tegemist on mäluväljaga. Oluliseks erinevuseks on see, et loendi elementidel ei ole eraldi nimesid, neile vastavad **järjenumbrid**. Elemendi järjenumbrid võivad muutuda, kui elemente lisatakse vahele või neid eemaldatakse. Põhimõtteliselt võivad massiivi elementidel olla erinevad tüübid (liigid), kuid tavaliselt on elementide tüübid siiski ühesugused. Väärtuste omistamine massiivi elementidele ja väärtuste lugemine toimub analoogselt muutujatega, erinevus on vaid viitamise viisis – loendi nimi koos indeksiga:

nimi(*indeks*) või **nimi[*indeks*]** või 

Siin **nimi** on loendi nimi, **indeks** – elemendi järjenumber, mis võib olla kas konstant, muutuja või avaldis. Mõnes keeles paigutatakse indeks ümarsulgudesse, teistes nurksulgudesse. Scratchis kasutatakse viitamiseks loendi elementidele spetsiaalset plokki. Mõned näited:

T(3) = „Känd" 

Loendi **T** kolmandale elemendile omistatakse väärtus Känd.

p = Palk(1) + Palk(k)

Loendi **Palk** esimesele elemendile liidetakse sama loendi element numbriga **k** ja saadud väärtus omistatakse muutujale **p**.

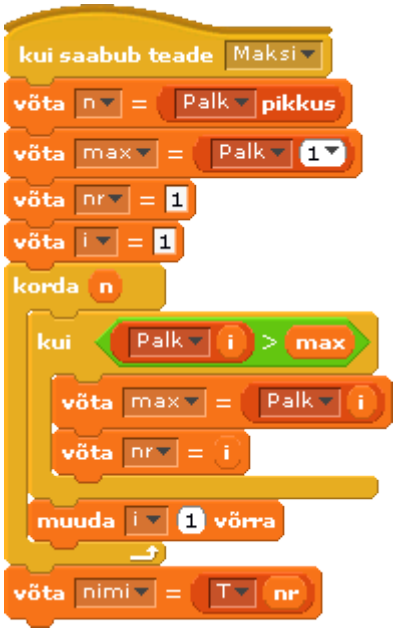
Allpool on toodud Scratchi skript ja Visual Basicu protseduur, mis leiavad loendis (massiivis) **Palk** olevate palkade aritmeetilise keskmise. Esitatud on ka Scratchi loendid **T** ja **Palk**.


```
Sub Keskmine()
  Dim k, n, Sum, kesk
  Dim Palk(1 To 10)
  Loe_Palk Palk, n
  Sum = 0
   For k = 1 To n
     Sum = Sum + Palk(k)
   Next k
   kesk = Sum / n
   MsgBox kesk
End Sub
```

Scratchi skript teeb kõigepealt kindlaks loendite pikkused (elementide arvu loendites). Loendid on ühesuguse pikkusega, seega pole oluline, millise järgi pikkus määratakse. Põhiosa käskudest on seotud loendi **Palk** elementide summa leidmisega. Summa (muutuja **Sum**) algväärtus võetakse alguses null. Korduses liidetakse seejärel summale järjest juurde iga töötaja palk. Viitamiseks loendi erinevatele elementidele kasutatakse muutujat **k**, mille väärtust suurendatakse igal korduse sammul ühe võrra alates ühest kuni töötajate arvuni (muutuja **n** väärtus). Summa jagatakse töötajate arvuga ja saadud väärtus omistatakse muutujale **kesk**.

Visual Basicus toimub kõik analoogselt. Massiivide asukoht ei ole praegu oluline (see võib olla vormil, Exceli töölehel, failides vm). Protseduurid Loe_Palk ja Loe_PT hangivad massivide elementide väärtused.

Allpool on toodud Scratchi skript ja Visual Basicu protseduur, mis leiavad maksimaalse palga ja seda saava töötaja nime. Keskel on toodud protseduuri **algoritm**.




```
protseduur Maksi
  loe T,  Palk
  n = Palk.pikkus
  max =Palk(1)
  nr = 1
  kordus i = 1..n
    kui Palk(i) > max siis
       max = Palk(i)
       nr = i
    lõpp kui
  lõpp kordus
  nimi = T(nr)
  kuva nimi, max
```

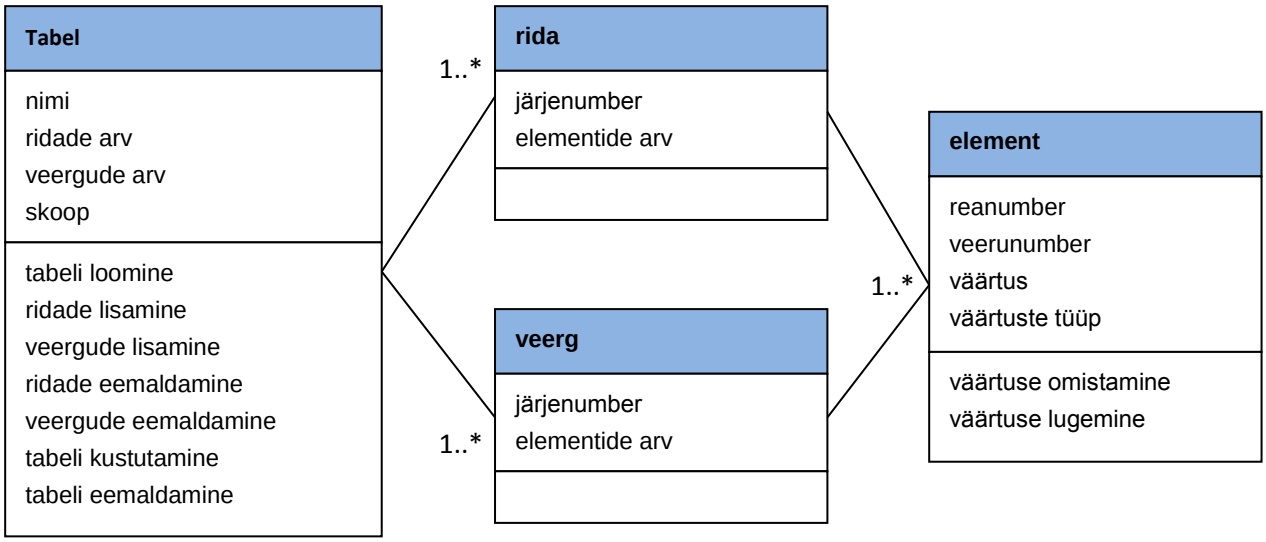
```
Sub Maksi()
   Dim i, n, max, nr, nimi
   Dim T(1 To 10), Palk(1 To 10)
   Loe_TP T, Palk, n
   max = Palk(1)
   nr = 1
   For i = 1 To n
      If Palk(i) > max Then
         max = Palk(i)
         nr = i
      End If
   Next i
   nimi = T(nr)
   MsgBox nimi & ", palk: " & max
End Sub
```

Tabel

Tabel on väga sageli kasutatav andmekogum. Tabeli korral on tegemist kahemõõtmelise massiiviga. Viitamiseks tabeli elementidele kasutatakse nime koos kahe indeksiga:

nimi(rida, veerg)

Näiteks A(1, 1), A(i, j), …



NB! Scratchis tabeleid kasutada ei saa.

Objektid

Objektorienteeritud programmeerimissüsteemides kasutatakse tarkvaraobjekte, mis modelleerivad reaalsete objektide olekut ja käitumist või esindavad teatud tarkvarakomponente. Objektideks võivad olla mitmesugused andmekogumid, dokumendid ja nende osad, vormid, ohjurid jms. Kitsamas tähenduses on objekt lihtsalt teatud andmeüksus, millega programm saab täita mingeid tegevusi. Üheks tüüpiliseks objektiks on näiteks graafiline kujutis ehk graafikaobjekt.

Laiemas tähenduses on objekt tarkvaraüksus, milles on ühendatud andmed ja protseduurid (programmid), mis määravad tegevusi nendega. Iga objektiga on seotud teatud valik **atribuute** ehk **omadusi** (näiteks graafikaobjekti jaoks on nendeks nimi, mõõtmed, asukoht ekraanil jms) ning **meetodeid,** mille abil määratakse tegevusi (operatsioone) antud tüüpi objektiga (näiteks graafikaobjekti puhul asukoha muutmine ekraanil, pööramine, värvuse muutmine jms). Omadused kujutavad endast objekti sisemuutujaid: igale omadusele vastab mäluväli, kus säilitakse selle jooksvat väärtust. Meetodid on protseduurid, mille abil määrtakse tegevusi objektidega. Tüüpiliselt muutuvad selle tulemusena objekti omadused. Näiteks meetodid graafikaobjekti teisaldamiseks (asukoha muutmiseks) muudavad selle asukohta määravaid koordinaate. Objekti saab panna reageerima teatud **sündmustele** – hiireklõps, vajutus kindlale klahvile, kokkupuude teise objektiga, teate saabumine teiselt objektilt jms. Sündmuse esinemisel käivitub automaatselt vastav protseduur või skript. Kui süsteem toetab paralleelprotsesse (nagu Scratch), siis võib sündmuse esinemisel käivituda ka mitu protsessi.

Ühetüübilised objektid kuuluvad ühte **klassi**. Klass on üldistus ehk abstraktsioon, mis määratleb (kirjeldab) antud klassi kuuluvate objektide olemuse ja kasutamise. See fikseerib objektide **omaduste** ja **sündmuste** valiku, mis on ühesugune kõikidel antud klassi kuuluvatel objektidel, sisaldades protseduure **meetodite** realiseerimiseks ning malli, mille alusel luuakse uus objekt.

Koostades objektorienteeritud keskkonnas oma rakendusi, saab rakenduse looja kasutada oma programmides olemasolevaid objekte või luua uusi objekte klasside määratluste alusel. Paljud süsteemid võimaldavad defineerida ja luua ka oma klasse. Viitamiseks objektide omadustele ja meetoditele kasutatakse enamikes süsteemides taolisi konstruktsioone:

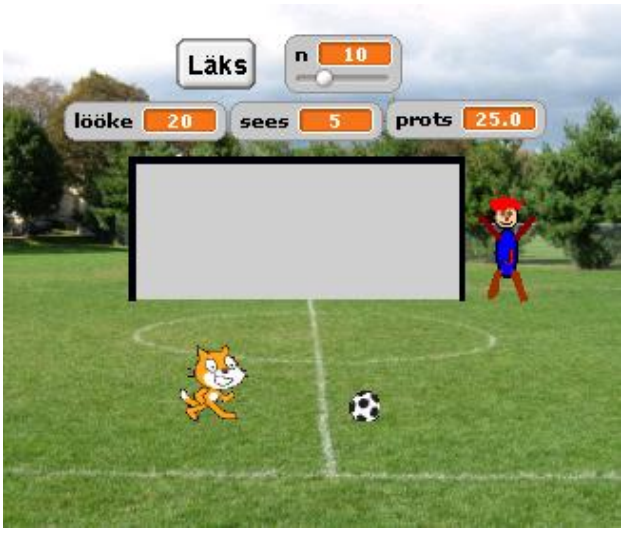
objekt.omadus ja **objekt.meetod** ***argumendid***

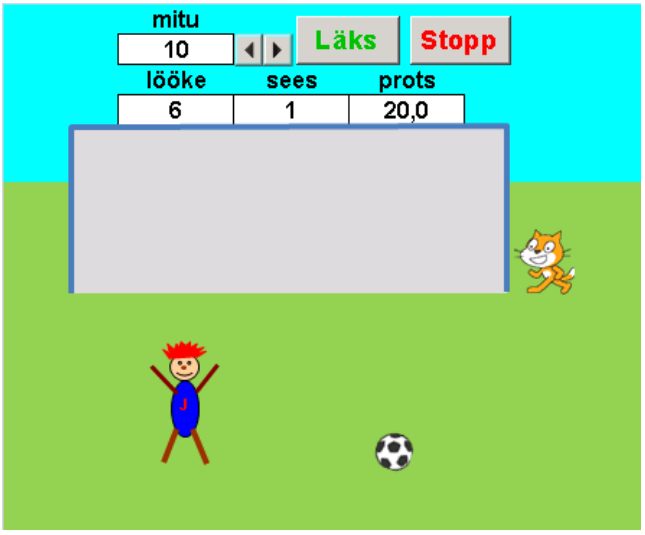
Siin on objekt **viit objektile** (võib olla ka muutuja); omadused ja meetodid esitatakse nende nimede abil. Meetodites kasutatakse sageli argumente, mis määravad omaduste uued väärtused, väärtuste muutuse, tegevuse tüübi, …

Sprait
nimi asukoht: X, Y suund, suurus värvus, nähtavus, …
liigu(), pööra() muudaX(), muudaY() vaheta_kostüümi() muuda_värvi() ütle(), mängi_heli(), …

Scratchis objekti mõistet otseselt ei kasutata, kuid praktiliselt on tegemist objektorienteeritud süsteemiga. Kõrval on toodud klassi **Sprait** kirjeldus, milles on esitatud valik omadusi ja meetodeid. Spraidid, lava ja selle taustad ning muud elemendid (pliiats, heliklipid jms) kujutavad endast objekte. Skriptide loomiseks kasutatavad käsud vastavad oma olemuselt meetoditele. Scratchis on objektide, nende omaduste, meetodite ja sündmuste kasutamine viidud sellisele tasemele, et programmi koostaja ei peagi nendest eriti täpselt teadma. Kuna skriptid tehakse Scratchis konkreetse spraidi jaoks, ei ole käskudes viidet objektile nagu oli näidatud ülalpool.

Näites on toodud Scratchi ja VBA (Excelis) programmid, mis imiteerivad pealelööke väravale. Programmid loendavad löökide ja tabamuste arvu ning leiavad tabavusprotsendi. Mõlemas programmis on neli graafikaobjekti: **Kraps**, **Juku**, **pall** ja **värav**. Scratchi programmis teeb pealelööke **Kraps**, VBAs **Juku**. Vaata demosid: Scratch, VBA (Excel).



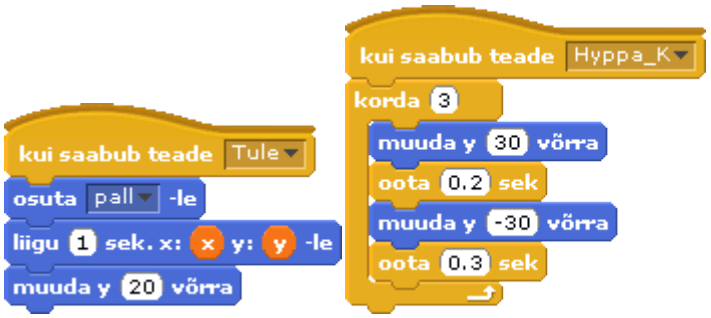


Palli skript

VBA peaprotseduur

```
Sub Trenn()
[lööke] .Value = 0
[sees].Value = 0
Do Until [lööke] = [n]
                                    ' pall platsile
  x = 40 + Rnd() * 400
  y = 160 + Rnd() * 150
  Liigu_XY pall, x, y, 10
                                    ' Juku palli juurde
  Liigu_XY Juku, x, y, 5
  Juku.IncrementTop -50
  paus 1
                                    ' pall värava suunas
  x = 40 + Rnd() * 400
  y = 20 + Rnd() * 200
  Liigu_XY pall, x, y, 20
  [lööke] = [lööke] + 1
  If On_Puude(pall, värav) Then
    [sees] = [sees] + 1
    Hyppa Juku, 3, 30
  Else
    Hyppa Kraps, 4, 20
  End If
  [prots] = [sees] / [lööke] * 100
  paus 0.5
Loop

pall.Left = Juku.Left
pall.Top = Juku.Top + Juku.Height
End Sub
```

Krapsu skriptid



VBA alamprotseduur

```
Sub Hyppa(kuju, n, h)
  Dim i
  For i = 1 To n
    kuju.IncrementTop  -h
    paus 0.2
    kuju.IncrementTop  h
    paus 0.4
  Next  i
End Sub
```

Tegevused on mõlemas programmis analoogsed. Kasutaja saab valida löökide arvu, st muutuja **n** väärtuse.

Tegevusi korratakse **n** korda:

1. **pall** viiakse sujuvalt juhuslikku kohta platsil,
2. lööja (**Kraps** või **Juku**) liigub **palli** juurde,
3. **pall** viiakse juhuslikku kohta **värava** piirkonnas,
4. suurendatakse löökide arvu ühe võrra,
5. kui **pall** puudutab **väravat**,
6. suurendatakse muutuja **sees** väärtust ühe võrra,
7. lööja (**Kraps** või **Juku**) teeb mõned hüpped,
8. vastupidisel juhul teeb hüppeid teine osaleja (**Juku** või **Kraps**),
9. arvutatakse protsent.

Kui löögid tehtud, viiakse **pall** lööja juurde ja lõpetatakse programmi töö.

Scratchis kasutatakse **Krapsu** viimiseks **palli** juurde **Krapsu** skripti **Tule** ning hüpete tegemiseks vastavalt **Krapsu** ja **Juku** skripte **Hyppa_K** ja **Hyppa_J** (ei ole näidatud).

VBAs kasutatakse hüpete juhtimiseks ühte parameetritega protseduuri **Hyppa**(kuju, n, h)**,** mille poole pöördutakse kaks korda erinevate argumentide komplektiga.

Scratchis kuuluvad skriptid kindlale spraidile ja määravad käsuplokkide (meetodite) abil tegevusi ainult antud spraidi jaoks. Sellepärast viiteid objektidele (spraitidele) skriptides ei kasutatagi.

VBAs saab protseduur määrata tegevusi mitme objektiga ning seepärast peab käsus alati näitama objekti. Tegevuste määramiseks kasutatakse **meetodeid** (*IncrementTop*), **omadusi** (Left**,** *Top*) ja spetsiaalseid **protseduure**: **Liigu_XY** (vastab Scratchi käsule [**liigu** *t* **sek x**…, **y**…]), **On_Puude** jms. Kõrvaloleval pildil on toodud väike valik klassi **Shape** (kujund ehk graafikaobjekt) omadusi ja meetodeid.

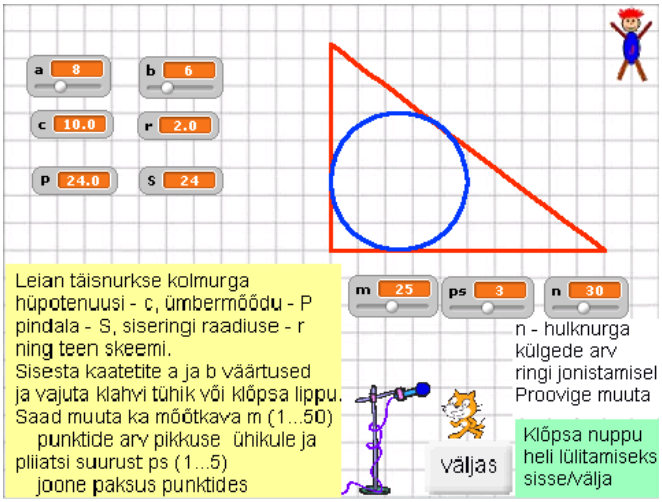
Shape
Name Left, Top, Rotation Width, Height Visible, …
IncrementLeft () IncrementTop () IncrementRotation () Copy (), Cut() Select (), …

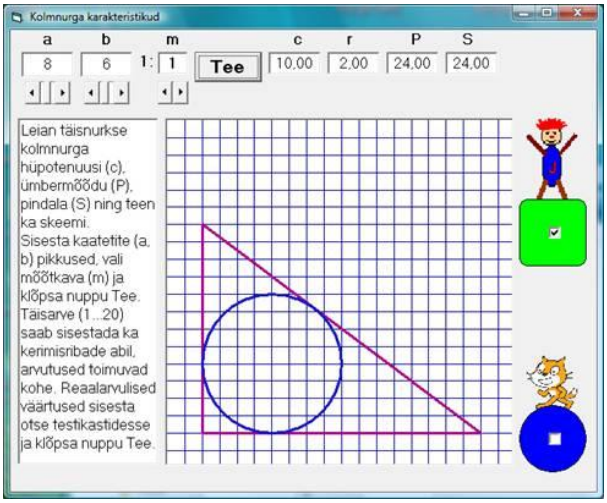
Lisaks graafikaobjektidele on VBA programmis kasutusel neli töölehe lahtrit: [**lööke**], [**sees**], [**prots**] ja [**n**], mille näol on samuti tegemist objektidega. Konstruktsioon [***lahtri_nimi***] on üks võimalikest viitamisviisidest lahtritele VBA programmides. Lahtril on üsna palju omadusi ja meetodeid: **Name, Value** (väärtus), **Address** jm. Omadus **Value** on lahtri jaoks vaikimisi võetav väärtus ning selle võib programmis ära jätta.

Siin ei ole näidatud väikest pealisehitust VBA programmile, mis määratleb objekti muutujad ja seob nendega vastavad objektid.

Kasutajaliides

Kasutajaliides sisaldab vahendeid, mille abil kasutaja saab rakendusega suhelda – näha tulemusi, muuta algandmeid, anda vajalikke korraldusi jms. Programmeerimissüsteemides on liideste loomiseks spetsiaalsed vahendid. Tavaliselt tehakse kasutajaliides rakenduse loomise faasis, kuid programmis võivad olla võimalused liidese muutmiseks ka täitmise ajal.





Pildil on näha Scratchis ja Visual Basicus loodud rakenduste kasutajaliidesed. Programm võimaldab leida kaatetite väärtuste järgi täisnurkse kolmnurga hüpotenuusi, ümbermõõdu, pindala ja siseringi raadiuse ning teeb ka vastava joonise. Erinevate andmeliikide kasutamise ning mõningate juhtimisega seotud tegevuste demonstreerimiseks on programmidesse lisatud ülesande lahendamise seisukohast mittevajalikke elemente – animatsioone ning Scratchis lisaks ka heliefekte. Vt programmi täitmist Scratchi näite veebilehelt.

Rakenduste loomise vahendid

Üldotstarbelised programmeerimiskeeled ja -süsteemid

Java, C, **C++**, **C#**, **Visual Basic**, **Pascal**, **Fortran**, … võimaldavad luua **autonoomseid rakendusi**. Programmi saab transleerida masinkeelde ja selle kasutamiseks pole enam vaja programmeerimissüsteemi, millega antud rakendus loodi. Lihtsamal juhul on Windowsis tegemist nn EXE-failiga. Kasutajaliidese realiseerimiseks kasutatakse tüüpiliselt vorme.

Üldotstarbelised rakendusprogrammid

Enim kasutatakse **tarkvarakomplekte** MS Office, OpenOffice, CorelOffice, mis sisaldavad tabeli-, teksti-, esitlus- ja andmebaasirakendusi, ning graafikaprogramme nagu AutoCAD, CorelDRAW jt. Enamik kaasaegsetest rakendusprogrammidest sisaldavad ka arendamisvahendeid, millest tuntuimaks on *Visual Basic for Application* (**VBA**). Lisaks Microsofti toodetele kasutatakse VBA-d enam kui 200-s rakendusprogrammis, seal hulgas ka sellistes tuntud süsteemides nagu AutoCAD, CorelOffice ja CorelDRAW.

Üldotstarbelised rakendusprogrammid võimaldavad luua nn dokumendipõhiseid rakendusi, mis luuakse mingi baasrakenduse keskkonnas. Võib eristada kolme põhivarianti:

- **Kasutatakse ainult baasrakenduse vahendeid** (tabelarvutustarkvara, graafikaprogrammid ja andmebaasisüsteemid). Näiteks oleks tabelarvutustarkvara, mis võimaldab tabeleid luua, kasutada valemeid, suurt hulka sisefunktsioone, diagramme jmt; võimaldavad luua efektiivseid

ja paindlikke rakendusi ka arendusvahenditeta. Vt näidet: rakendus ruutvõrrandite lahendamiseks.

- **Kasutatakse baasrakenduse vahendeid ja arendusvahendeid** (nt VBA). Baasrakendust kasutatakse eeskätt kasutajaliidese loomiseks, andmete kujundamiseks ja vormindamiseks. Näited Excel+VBA: hunt, kits ja kapsas.
- **Kasutatakse ainult arendusvahendeid**, kusjuures baasrakendus on siis nö konteiner arendusvahendi programmide hoidmiseks. Kasutajaliidesena kasutatakse sellisel juhul tavaliselt vorme. Rakenduse üleviimine ühest rakendusest teise, millel on sama arendusvahend (nt VBA), on võrdlemisi lihtne. Järgnevalt on võimalik vaadata paari väikest näidet, mis on realiseeritud VBA abil Excelis, Wordis ja PowerPointis.

Võrgurakenduste arendusvahendid

Võrgurakendused põhinevad veebi- ehk HTML-dokumentidel. Nende loomiseks on palju erinevaid vahendeid, HTML-redaktoreid nagu **Front Page**, **Dreamweaver**, **HTML Kit** jpt. Veebidokumentide dünaamilisuse suurendamiseks saab kasutada mitmeid skriptikeeli **PHP**, **JavaScript**, **VBScript**, **Python**. Võrgurakenduste loomisel saab kasutada ka üldotstarbelisi programmeerimiskeeli nt **Java**, **C#**, **Visual Basic**, **Net** jt.

Matemaatikaprogrammid

MathCAD, **MathLab**, **Mathematica**, **Maple**, **Wiris** jms, mis sisaldavad rikkaliku valikut vahenditest arvutuslike rakenduste loomiseks. Neis on võimalik luua ka graafikuid ning kasutada ka **sisseehitatud programmeerimisvahendeid**.

Modelleerimiskeeled ja -süsteemid

Rakenduste modelleerimiseks kasutatakse analüüsi- ja disainifaasis üha enam spetsiaalseid keeli, millest üheks enimkasutatavaks on UML (***U**nified **M**odeling **L**anguage*).

Rakenduste loomise põhietapid

Allpool on esitatud rakenduste loomise põhietapid ja nende lühiiseloomustus. Analoogsed etapid esinevad ka muude toodete (ehitised, masinad, …) loomisel.

Ülesande püstitus	põhifunktsioonid, nõuded, tingimused, piirangud, ...
Analüüs	lahendamise üldpõhimõtted, põhiandmed, -objektid ja -tegevused, kontseptuaalsed mudelid, realiseerimisvahendid, ...
Projekteerimine (disain)	kasutajaliidese kavandamine ja loomine, täpsemad mudelid, andmete ja programmide struktuur, algoritmid, dokumenteerimine, …
Realisatsioon	kasutajaliidese loomine, valemite, skriptide ja programmide koostamine, silumine, testimine, dokumenteerimine, ...

Etappide sisu ja mahud sõltuvad rakenduse iseloomust, suurusest, keerukusest, kasutatavast metoodikast ja vahenditest, teostajast või meeskonnast jms. Lihtsate ja väikeste rakenduste korral võivad kõik etapid sulanduda ühte ja taanduda praktiliselt realisatsioonile (programmeerimisele). Suurte ja keeruliste rakenduste korral võivad analüüsi- ja disainietapid olla tunduvalt mahukamad kui realisatsioonietapp. Taoliste rakenduste puhul eristatakse mõnikord kolme osa: ülesande püstitus, projekteerimine ja realiseerimine (ehitamine). Ülesande püstitus võib tulla ühelt firmalt (tellijalt), projekteerimise ja realiseerimise võivad täita erinevad meeskonnad (firmad).

Näites on toodud klassikaline loogikaülesanne ehk mäng „Hunt, kits ja kapsas“, mis on tuntud juba sajandeid erinevate rahvaste folklooris ning sellel on mitmeid variatsioone. Kaasajal on loodud ka palju selle teemaga seotud arvutimänge. Tüüpvariant on järgmine: mees rändab koos hundi, kitse ja kapsaga ning peab saama koos oma seltskonnaga paadiga (parvega) üle jõe, arvestades teatud kitsendusi, mis on esitatud ülesande püstituses (vt ka demo).

Ülesande püstitus

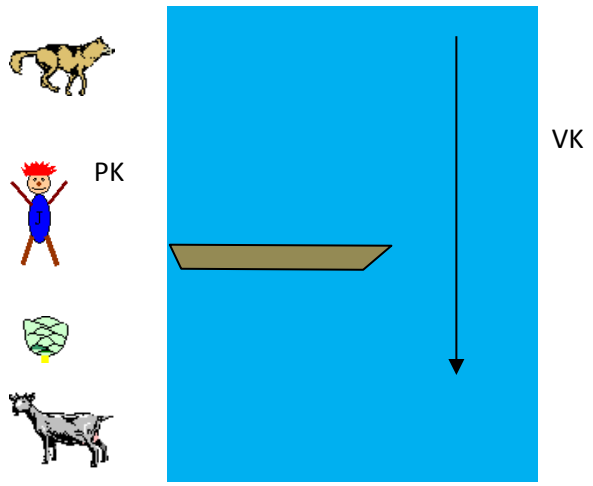
Koostada rakendus, mis võimaldab mängida arvutil mängu „Hunt, kits ja kapsas“.

Juku peab minema üle jõe hundi, kitse ja kapsaga, arvestades järgmist:

- paati mahub lisaks Jukule ainult üks „tegelane“,
- ilma Jukuta paat ei sõida,
- kui kits jääb ilma Jukuta koos kapsaga, sööb kits kapsa ära,
- kui hunt jääb ilma Jukuta kitsega, sööb hunt kitse ära.

Kes ja millal läheb paati või tuleb maha ning millal ja kellega paat ja Juku sõidavad, määrab mängija (kasutaja). Võiks fikseerida ka aja, mis kulub kõigi üle jõe viimiseks.

Analüüs



Põhiobjektideks on **paat, Juku, hunt, kits** ja **kapsas**. Alguses on paat parema kalda (PK) juures ning kõik tegelased on paremal kaldal.

Ülesandel on kaks lahendust:

(1) Juku viib **kitse** vasakule kaldale, tuleb tagasi ja viib üle **hundi** ning toob **kitse** tagasi. Viib üle **kapsa**, tuleb tagasi ja viib vasakule kaldale **kitse**.

(2) Juku viib **kitse** vasakule kaldale, tuleb tagasi ja viib **kapsa** vasakule kaldale ning toob **kitse** tagasi. Siis viib **hundi** üle, tuleb tagasi ja viib vasakule kaldale **kitse**.

Esitame esimese lahenduse stsenaariumi (algoritmi) veidi detailsemalt.

Juku viib kitse vasakule kaldale

Juku tuleb paadiga tagasi.

Juku viib hundi vasakule kaldale.

Juku toob kitse tagasi.

Juku viib kapsa vasakule kaldale.

Juku tuleb tagasi.

Juku viib kitse vasakule kaldale

Toodud stsenaarium annab mängijale üsna täpse tegutsemisjuhendi, kuid jätab rakenduse realiseerimise seisukohalt mitmeid lahtisi otsi. Ei ole selged paatimineku ja mahatuleku reeglid. Näiteks, kas kaks sõitjat lähevad paati või tulevad maha ühekaupa (järjestikku) või korraga? Kui pealeminek või mahatulek toimub järjestikku, võivad tekkida olukorrad, kus kits ja kapsas või hunt ja kits jäävad koos ühte kohta ilma Jukuta.

Näiteks, kui alguses läheb paati kõigepealt Juku, ongi nii hundil kui kitsel vaba voli – Juku on paadis, nemad kaldal. Võib minna lahti suureks söömiseks – kits sööb kapsa ja hunt sööb kitse. Kas arvestada, et Juku on samuti kaldal ja takistab söömist? Et teha mängu raskemaks (huvitamaks) otsustame nii, et pealeminek ja mahatulek toimub alati ühekaupa mängija käsu peale ning kitse ja kapsast või hunti ja kitse ei tohi mingil juhul kusagile jätta ilma Jukuta. Üleviimise detailsem stsenaarium (algoritm) oleks järgmine:

1. Kits paati
2. Juku paati (hunt kapsast ei söö)
3. Paat Juku ja kitsega vasakule kaldale
4. Kits maha vasakule kaldale (Juku jääb paati)
5. Paat Jukuga paremale kaldale
6. Hunt paati (Juku ei pea maha tulema)
7. Paat Juku ja hundiga vasakule kaldale
8. Juku maha vasakule kaldale (enne hunti!)
9. Hunt maha vasakule kaldale (Juku on kohal)
10. Kits paati (enne Jukut, muidu hunt sööb ära)
11. Juku paati
12. Paat Juku ja kitsega paremale kaldale
13. Juku maha paremal kaldal (enne kitse)
14. Kits maha paremale kaldale
15. Kapsas paati (enne Jukut)
16. Juku paati
17. Paat Juku ja kapsaga vasakule kaldale
18. Kapsas maha vasakule kaldale
19. Paat Jukuga paremale kaldale
20. Kits paati
21. Paat Juku ja kitsega vasakule kaldale
22. Juku maha vasakule kaldale
23. Kits maha vasakule kaldale (kõik on üle!)

Nagu näha, on nüüd kirjas üsna palju tegevusi, mis arvestavad, et pealeminek ja mahatulek toimuks ühekaupa ja teatud paare ei jäetaks kunagi kahekesi. Kui mängija tegutseb täpselt selle stsenaariumi järgi,

ei saa tekkida olukorda, kus paati püütakse panna üle kahe tegelase. Kuid kasutaja võib ju mitte reegleid teada või eksida. Kuidas võiks sel juhul toimida? Kaks põhivarianti: rakendus ei luba viia ülearust tegelast paati või paat läheb ülearuse tegelase korral põhja (koos paadisolejatega) ning mängu peab alustama uuesti. Valime teise variandi.

Stsenaariumi alusel saab määrata ka objektide põhitegevused ja -omadused, mida saab arvestada rakenduse kavandamisel ja realiseerimisel.

Paat
nimi koht X, Y mitu
algseaded sõit vasakule sõit paremale kas_palju põhjaminek

Paadi omadust **nimi** saab kasutada paadile viitamiseks. Omaduse **koht** abil määratakse paadi asukoht: vasak kallas, parem kallas, jõel, **X, Y** on paadi koordinaadid, mida saab kasutada näiteks sihtkoha määramisel. Omadus **mitu** näitab tegelaste arvu paadis: alguses võetakse selle väärtuseks 0 ning seda muudetakse iga kord, kui mõni läheb paati (+1) või tuleb maha (-1).

Meetod **algseaded** paneb paika mõned algväärtused: asukoht – parem kallas, sõitjate arv (omaduse **mitu** väärtus) – 0 jms. Meetodid **sõit vasakule** ja **sõit paremale** juhivad paadi liikumist vastavalt vasakule või paremale. Meetod **kas palju** kontrollib, kui paati tuleb uus tegelane, kas sõitjate arv ei ole suurem kui kaks. Kui on, käivitatakse meetod **põhjaminek**.

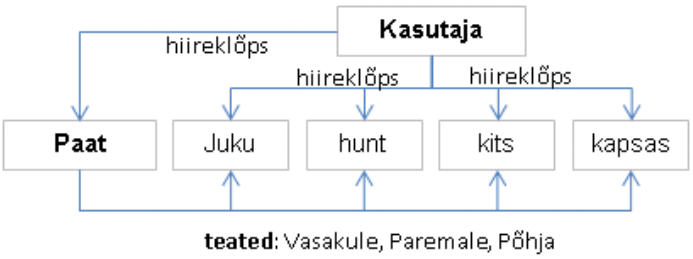
Tegelane
nimi koht X, Y nähtavus
algseaded maha paremal maha vasakul paati sõit vasakule sõit paremale põhjaminek

Neljal objektil **Juku**, **hunt**, **kits** ja **kapsas** (siin kasutatakse nende jaoks ühisnimetust **Tegelane)**, on ühesugune valik omadusi ja meetodeid. Võib rääkida ka klassist **Tegelane**, milles on neli objekti (eksemplari).

Omadust **nimi** kasutatakse viitamiseks konkreetsele objektile, **koht** näitab tegelase asukohta: paremal kaldal, paadis, vasakul kaldal, **X, Y** on objekti koordinaadid, mida kasutatakse liikumiste sihtkohtade määramiseks. Omadust **nähtavus** võib kasutada objekti peitmiseks, kui see läheb koos paadiga põhja.

Meetod **algseaded** viib objekti alguspunkti paremal kaldal ning võtab omadusele **koht** vastava väärtuse jms. Meetodid **maha paremal** ja **maha vasakul** viivad objekti vastavasse punkti paremal või vasakul kaldal, **paati** viib objekti paati. Meetodid **sõit vasakule** ja **sõit paremale** viivad objekti vastavale kaldale, kui objekt on paadis. Meetod **põhjaminek** teeb objekti nähtamatuks, kui paat läheb põhja.

Üheks rakendusega seotud oluliseks tegelaseks on **kasutaja** ehk **mängija**, sest just tema määrab oma tegevusega, mida peab tegema üks või teine objekt ning juhib kogu mängu. Rakenduse töökorraldus võiks olla järgmine. Mängija klõpsab hiirega objekti. Selle vastav protseduur valib hetke seisust lähtudes tegevuse, kasutades objekti omadust **koht**. Kui näiteks klõpsatakse paremal kaldal olevat **paati** ning **Juku** on paadis, liigub **paat** vasakule kaldale. Paadiga koos liigub vasakule ka **Juku**. Kui paadis on veel keegi, liigub ka see. Selle, et sõit toimub, ja sõidu suuna, teatab **paat**. Näiteks, kui klõpsatakse kaldal olevat **hunti** ja **paat** on sama kalda juures, läheb **hunt** paati. Kui paat on teise kalda juures, siis hiireklõps ei mõju.



Sellega on määratletud rakenduse töötamise üldised põhimõtted, fikseeritud põhiobjektid ning nende omadused ja meetodid ning tehtud kindlaks koostöö objektide vahel. Analüüsietapi võib lugeda lõppenuks ja saab asuda disaini (projekteerimise) juurde. Paneme tähele, et seni pole olnud juttu realiseerimisvahendi(te)st. See on üsna tüüpiline analüüsifaasi korral, sest siin vaadeldavad põhimõtted ja mudelid ei peaks olema seotud kindla programmeerimiskeelega.

Sageli valitakse realiseerimisvahendid just analüüsietapi tulemuste alusel, kuid kui realiseerimisvahendid on juba ette teada, võib teatud määral nendega arvestada.

Projekteerimine ehk disain

Disainifaasis peaks juba arvestama kindlate realiseerimisvahenditega, kuid mitmed aspektid võivad olla üldise iseloomuga ja need täpsustatakse realiseerimisfaasis. Siia kuluvad eeskätt kasutajaliidese mitmed elemendid ja algoritmid. Praegu arvestame, et realiseerimine toimub Scratchis.

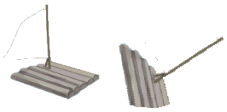
Alustame kasutajaliidesest, mis kujundatakse laval. Lava taustaks on jõe kujutisega pilt. Spraidid (graafika-objektid) on praegu ka kasutajaliidese elemendid, sest mängija kasutab neid hiirega klõpsates vajalike tegevuste määramiseks.

Laval on mõned teated ja juhendid. Samuti on siin aja (muutuja **aeg**) monitor.

Objektid (spraidid)

Põhimõtteliselt on tegemist samade põhiobjektidega. On tehtud mõningaid mitteolulisi muudatusi ja täiendusi:

- paadi asemel on siin **parv**, lihtsalt pilt on teine. Nimi oleks võinud jääda ka endiseks, kuid praegu on seda siiski muudetud. Spraidil on kaks kostüümi: parv nö normaalasendis ja ümberläinud parv.



- **Juku** – sama, mis enne. Üks kostüüm. Juku kasutamine erineb mõnevõrra teiste tegelaste kasutamisest. Formaalselt viib tema teisi üle: paat ilma Jukuta ei liigu. Muus osas erinevusi pole.

- **hunt** – üks kostüüm. Sööb ära kitse, kui on sellega ilma Jukuta, kapsast ei söö.

- **kits** ja **kapsas** – mõlemal on kaks kostüümi. Teised kostüümid on tekstiteated: „Hunt sõi kitse ära!“ ja „Kits sõi kapsa ära!“.

- mõned abispraidid: juhendid, teated jmt.

Muutujad

- **koht_P** – parve asukoht: 1 – paremal kaldal, 2 – vasakul kaldal. Täpsemalt öeldes: parv on parema või vasaku kalda juures. Vastab parve (paadi) omadusele **koht**. Globaalne muutuja. Algväärtus 1. Muudab parve põhiskript.
- **koht** – tegelaste asukohad, neli lokaalset muutujat. Väärtused: 1 – paremal kaldal (maas), 2 – vasakul kaldal, 3 – parvel. Muutujad vastavad tegelaste omadustele **koht**. Algväärtusteks on 1.
  NB! Võivad olla ka globaalsed muutujad, siis peavad neil olema erinevad nimed.

- **mitu** – tegelaste arv parvel. Esindab parve vastavat omadust. Globaalne muutuja, algväärtus 0, suurendatakse ühe võrra iga kord, kui mingi tegelane läheb parvele ning vähendatakse ühe võrra, kui keegi läheb maha. Kui väärtus läheb suuremaks kahest, läheb parv koos sellel olevate tegelastega põhja.
- **aeg** – näitab jooksvat aega. Globaalne muutuja.
- **tun** – tunnus, mis näitab, kas mäng käib (tun=1) või mitte (tun=0). See on globaalne muutuja. Algväärtus on 1, kui mäng käivitatakse; omistatakse null, kui mäng lõppeb normaalselt: parv läheb põhja, kits sööb kapsa või hunt sööb kitse ära. Kui tun=0, ei järgne klõpsamisele mingeid tegevusi.
- **t** – ülesõidu aeg sekundites on globaalne muutuja, mida saab muuta programmi seadistamisel paadi skriptis **algus**.

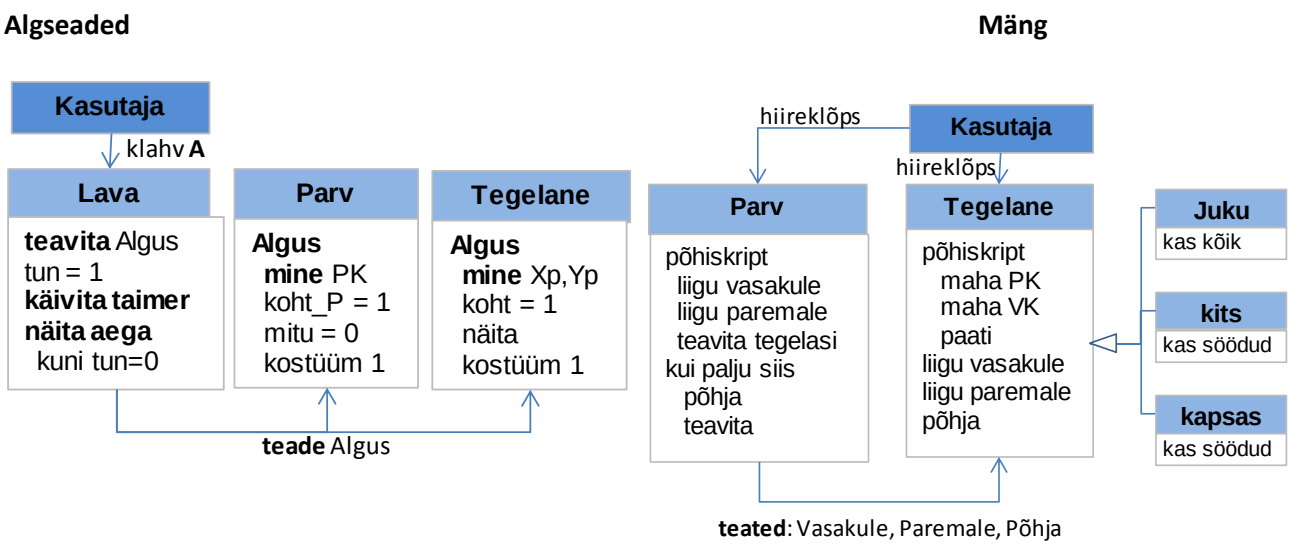
Programm

Scratchi programm koosneb **skriptidest**. Skript kuulub kindlale **spraidile** või **lavale** ning ühe objekti skript ei saa määrata tegevusi teise objektiga, kuid objekt (skript) võib väljastada teate käivitamaks teise skripti, mis saab täita vajalikud tegevused.

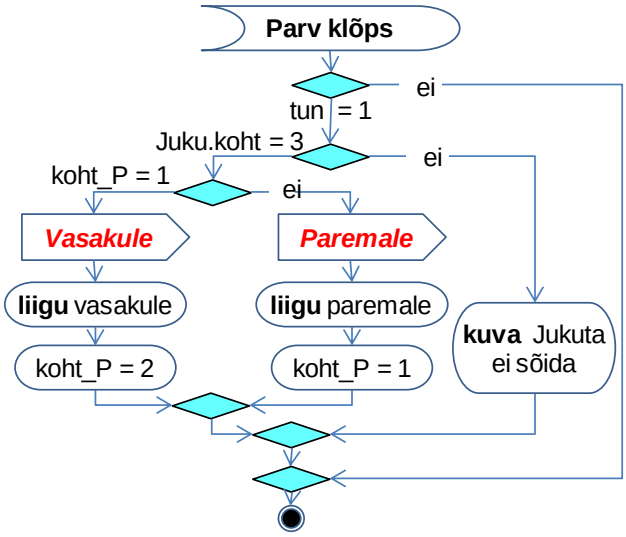
Antud juhul võib programmis eristada kahte osa: **algseaded** ja **mäng**.

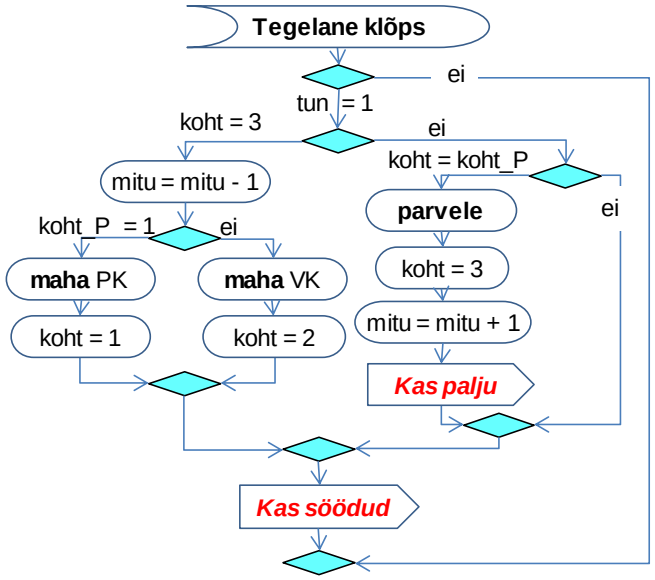
Algseaded tehakse uue mängu algul, kui kasutaja vajutab klahvi **A**. Seda võib teha suvalisel hetkel, katkestades jooksva mängu. Klahvisündmusele reageerib **Lava**. Vastav skript väljastab teate **Algus**. Teate võtavad vastu **parv** ning **Tegelased** (**Juku, hunt, kits** ja **kapsas**). Parve skript **Algus** viib parve parema kalda juurde (**mine PK**), võtab asukohta näitava muutuja **koht_P** väärtuseks on **1** ja muutuja **mitu** väärtuseks **0**.

Kõikide tegelaste tegevused on analoogsed. Skeemil tähendab '**mine Xp, Yp**' objekti viimist parema kalda juurde – punkti koordinaatidega **Xp, Yp**. Erinevatel objektidel on need erinevad, et nad ei kataks üksteist. Konkreetsed väärtused pannakse paika realisatsiooni ajal Scratchis. Käsk **koht = 1** tähendab, et vastava tegelase asukohta määravale muutujale omistatakse väärtus **1**.

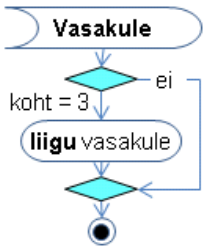


Mängu käigu määrab mängija: arvestades hetkeseisu, vajadust ja reegleid, klõpsab ta vastavat objekti. Allpool on toodud parve ja tegelaste põhiskriptide algoritmid.



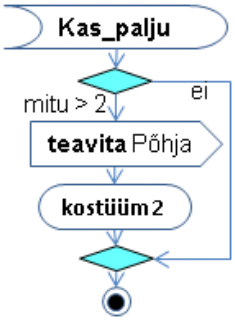


Kui klõpsatakse parve, kontrollitakse, kas mäng käib (**tun = 1**) ning kui **jah**, vaadatakse, kas **Juku** on parvel (**Juku.koht = 3**). Kui on, tehakse kindlaks, kas parv on paremal (**koht_P = 1**) või vasakul kaldal. Vastavalt sellele toimub ka parve liikumine. Enne seda saadab **parv** teate, mille võtavad vastu kõik tegelased, kuid sellele reageerivad ainult parvel olevate tegelaste (**koht = 3**) vastavad skriptid. Näites on toodud skeem liikumiseks vasakule. Käsk **liigu** realiseeritakse Scratchi plokiga [**liigu t sek. x**..., **y**...] (sujuv liikumine) või **[mine x..., y**...] (kohene asukoha vahetus).



Parve ja sellel olevate tegelaste liikumine peab toimuma sünkroonselt.

Kui klõpsatakse tegelast ja muutja **tun** väärtus on **1** (mäng käib), tehakse kindlaks, kas tegelane on parvel (**koht = 3**) või kaldal. Kui tegelane on parvel, siis vastavalt parve asukohale (näitab muutuja **koht_P** väärtus), läheb tegelane maha paremale (**maha PK**) või vasemale kaldale (**maha VK**); seejuures muudetakse vastavalt ka asukohta näitava muutuja väärtust ja vähendatakse ühe võrra muutuja **mitu** väärtust. Realisatsioonis võib asukoha muutmiseks kasutada Scratchi käsku [**liigu t sek**, **x**..., **y**...] (sujuv liikumine) või [**mine x**..., **y**...] (kohene). Kui tegelane ei ole parvel ja on parvega samal kaldal, läheb ta parvele. Kui tegelane ja parv on erinevatel kallastel, ei tohiks hiireklõps mõjuda.



Kui keegi läheb parvele, suurendatakse muutuja **mitu** väärtust ühe võrra ja saadetakse teade **Kas_palju**. Teate võtab vastu **parv** ja kontrollib muutuja **mitu** väärtust. Kui see on suurem kui 2, läheb parv põhja. Parvele võetakse teine kostüüm, parvel olnud tegelased peidetakse. Mängu tuleb alustada uuesti algusest.

Iga kord, kui toimub kas tegelase minek parvele või mahaminek, saadetakse teade **Kas söödud**. Seda töötlevad **kapsa** ja **kitse** vastavad skriptid, sest neil on oht saada sööduks. Näiteks **kitse** skriptis kontrollitakse, kas **kitsel** ja **hundil** on sama **koht**, kui **jah** ning **Juku koht** ei lange sellega kokku, loetakse **kits** sööduks: võetakse teine kostüüm teatega „Kits on söödud!“ ning muutujale **tun** omistatakse väärtus **0**, mis tähendab mängu lõppu. Seda teeb lava vastav skript. **Kapsaga** toimub kõik analoogselt. Ka **Jukul** on üks lisaskript, mis teeb kindlaks, kas kõik on viidud üle jõe.

```
Kas_söödud
  kui kits.koht = hunt.koht siis
    kui kits.koht <>Juku.koht = 3 siis
      võta kostüüm 2
      tun = 0
```

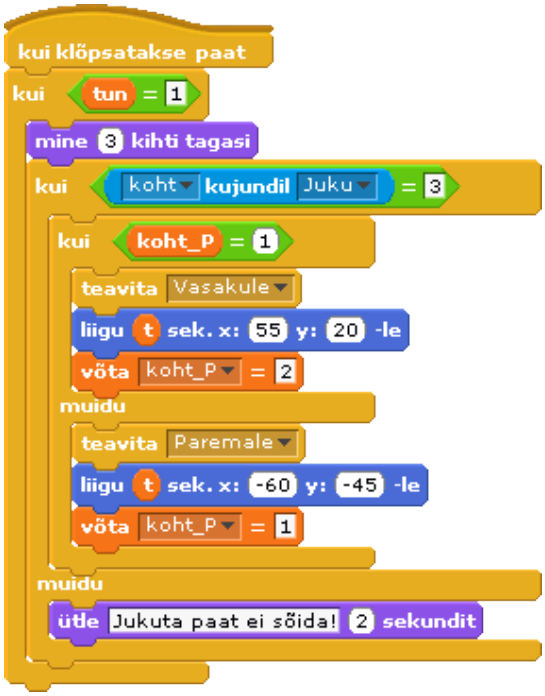
Realisatsioon

Realisatsioonifaasis võidakse teha mõningad täpsustused. Praegu otsustame, et objektide olekute (asukohtade) fikseerimiseks kasutame lokaalseid muutujaid **koht**, ainult parve jaoks kasutame globaalset muutujat **koht_P**, sest sellele tuleb tihti viidata teiste spraitide skriptides. Ülesõitmine toimub sujuvalt, pealeminek ja mahatulek on momentsed. Loomulikult pannakse realiseerimisel paika ka konkreetsed koordinaadid.

Siin on esitatud **Lava** skriptid. Kõik algab skriptist, mis reageerib klahvi **A** vajutusele. See paneb muutuja **tun** väärtuse korraks nulliks, teeb väikese pausi (eelmise mängu peatamiseks) ja väljastab teate **Algus**, mille võtab vastu **Lava** skript **Algus** ja ka tegelaste samanimelised skriptid.

See kõik viib mängu algseisu.

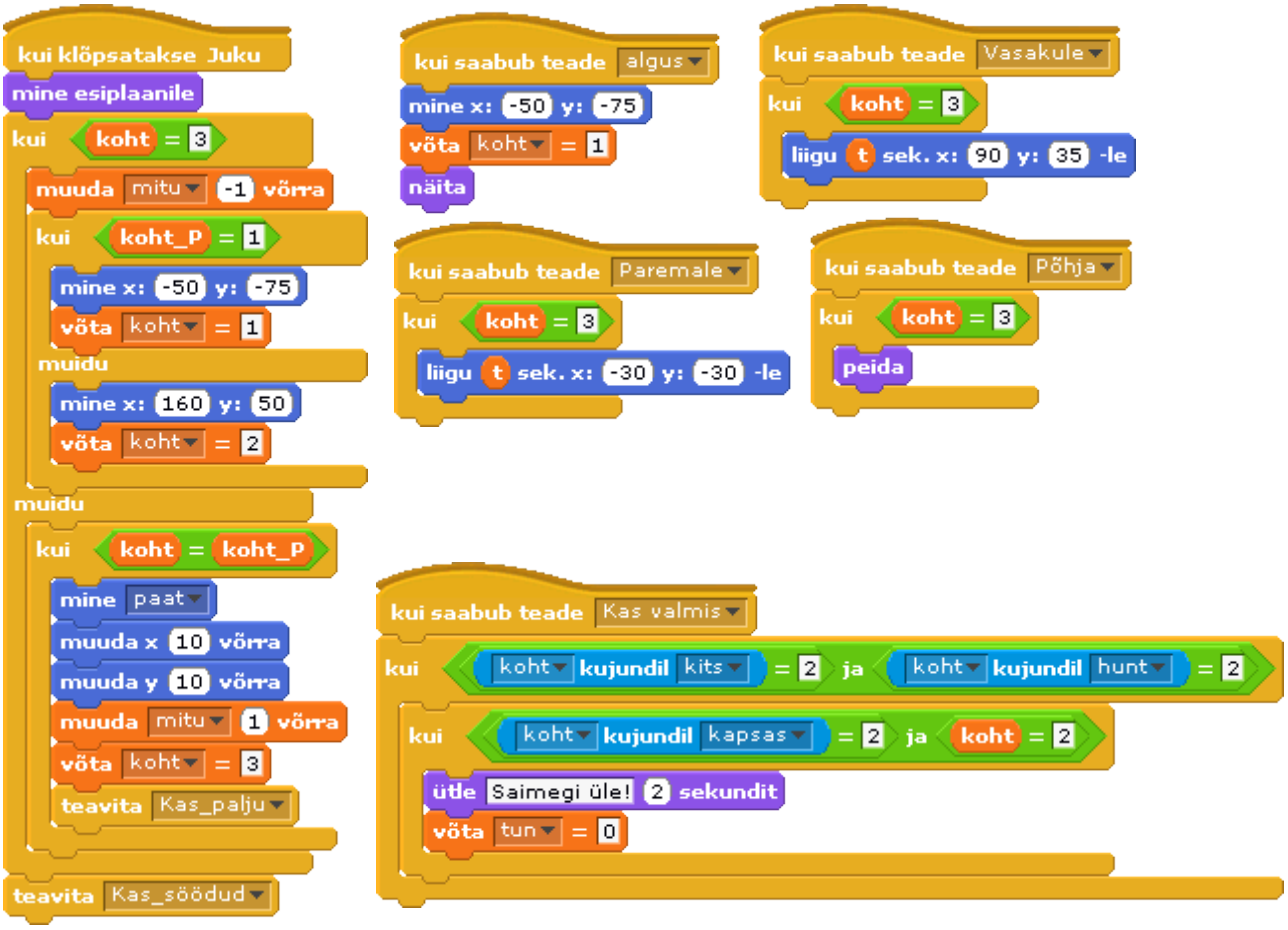
Parve skriptid



Parve koordinaadid vasaku kalda ääres on (-60; -45). Neid kasutatakse skriptis **Algus** ja põhiskriptis liikumisel paremale.

Liikumised toimuvad sujuvalt. See määratakse käskudega [**liigu t sek x**...**y**...] Siin on muutuja **t** väärtus 1, mis on omistatud skriptis **Algus** ning määrab liikumise aja.

Juku skriptid



Kõikide tegelaste skriptid on väga sarnased ja neid saab luua kopeerimise teel. Koopiates tuleb teha mõningaid muudatusi, mis seisneb peamiselt koordinaatide muutmises.

Juku skriptid erinevad teistest kõige rohkem. Jukut nihutatakse peale parvele minekut käskudega [**muuda x**…] ja [**muuda y**…], et see ei sattuks kohakuti üleviidava tegelasega. Teistel ei ole see vajalik. Siin on ka üks eriskript: **Kas valmis**, mida teistel ei ole. Skript kontrollib, kas kõik on üle viidud. Selleks peab kõigil olema muutuja **koht** väärtus 2 (vasak kallas).

Ühe spraidi skriptides saab viidata teiste spraitide lokaalsetele muutujatele. Vaadeldav skript kuulub Jukule, kelle lokaalsele muutujale **koht** viidatakse otse nime 

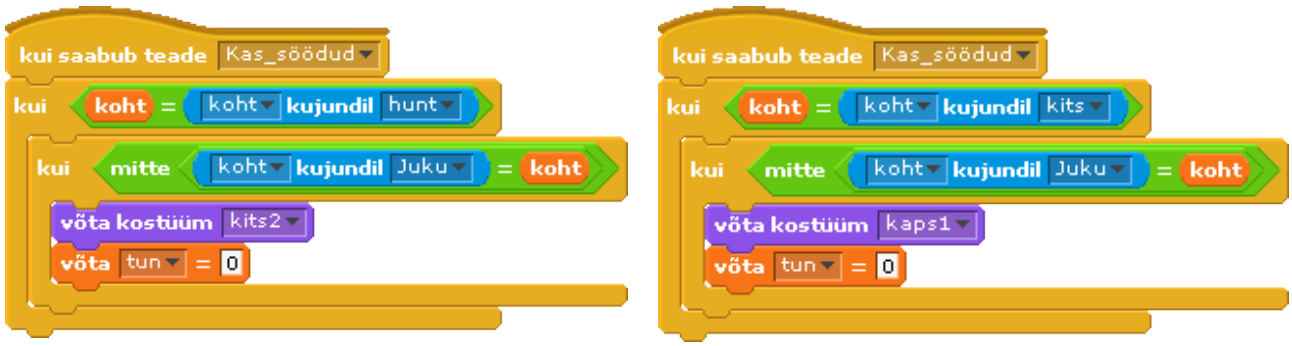
abil, teise spraidi sama nimega muutujale viitamiseks kasutatakse grupi **Andurid** plokki 

.

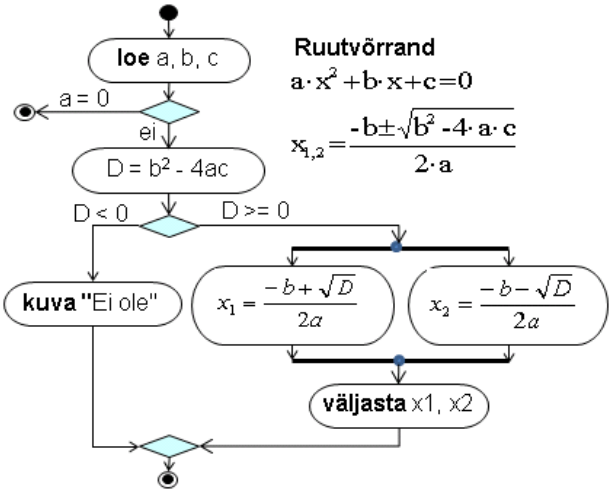
Kitsel ja kapsal on skriptid, mis kontrollivad võimalikku söömise fakti. Ka neis kasutatakse viitamist teise spraidi muutujatele.

Hundi ja kitse üldised skriptid

Kitse ja kapsa skriptid, mis kontrollivad söömist



Algoritmimine



Algoritm on sammsammuline tegevusjuhis, juhend, eeskiri mingi tegevuse sooritamiseks või eesmärgi saavutamiseks. Algoritm määrab, milliseid tegevusi, milliste andmetega ja millises järjekorras peab täitma. Mõiste „algoritm“ pärineb matemaatikast, kuid seda kasutatakse ka mujal: retseptid, juhendid, stsenaariumid – kõik need on eeskirjad e algoritmid. Eriti palju kasutatakse seda mõistet seoses rakenduste loomise ja programmeerimisega.

Algoritmi olemus

Algoritm on sammsammuline tegevusjuhis, juhend, eeskiri mingi tegevuse sooritamiseks või eesmärgi saavutamiseks. Algoritm määrab, milliseid tegevusi, milliste andmetega ja millises järjekorras peab selleks täitma. Mõiste algoritm pärineb matemaatikast, kuid seda kasutatakse ka mujal: retseptid, juhendid, stsenaariumid – kõik need on eeskirjad ehk algoritmid. Eriti palju kasutatakse seda mõistet seoses rakenduste loomise ja programmeerimisega. Võib öelda, et programmeerimise sisu ja kunst seisneb suurel määral just algoritmide loomises. Programmi teksti kirjutamist nimetatakse **kodeerimiseks**. Programm on üks võimalikest algoritmi esitusviisidest. Algoritmi sisu ja esitusviis sõltub täitjast, olgu selleks siis inimene, arvuti, robot vms.

Siin arvestatakse, et täitjaks on **programmjuhtimisega seade** (nt arvuti) – protsessori ja mäluga varustatud seade, millel võivad olla võimalused (täpne koosseis ei ole oluline) infovahetuseks väliskeskkonnaga. Algoritmi võib sellisel juhul vaadelda loodava programmi mudelina, so tegevuste kirjeldusena üldisel kujul, mis otseselt ei sõltu konkreetsest programmeerimiskeelest. Algoritm on eeskätt mõeldud inimesele ning on orienteeritud ülesande lahendamise sisule ja lahenduse kirjelduse ülevaatlikkusele. Põhimõtteliselt võiks silmas pidada, et algoritmi saaks realiseerida erinevate programmeerimiskeelte abil. Algoritme kasutatakse nii rakenduste loomisel kui ka nende dokumenteerimisel näiteks kas või selleks, et neid hiljem realiseerida mõne muu vahendiga.

Sageli kasutatakse rakenduse loomisel algoritmi järk-järgulist detailiseerimist. Algfaasis võib algoritm olla üsna üldisel kujul, kuid arenduse käigus see täpsustub. Lõppfaasis peaks algoritm üsna täpselt arvestama kasutatavate andmete liike ja organisatsiooni ning tegevusi, mida arvuti saab täita andmetega. Tüüpiliselt detailiseeritakse algoritm tasemele, millelt oleks lihtne koostada programmi.

Andmed algoritmides

Nii nagu programmid, on ka algoritmid ja nende esitus otseselt sõltuvad kasutatavate andmete liigist ja organisatsioonist. Edaspidi eeldame, et arvuti saab salvestada ja töödelda erinevat liiki andmeid: märkandmeid (arvud, tekstid, tõeväärtus, …) ning graafika- ja heliandmeid. Algoritmides saab kasutada muutujaid, andmekogumeid (loendid, massiivid, tabelid) ja objekte.

Skalaarandmed esitatakse algoritmides konstantidena ja muutujatena. Nende olemus ja käsitlus on põhimõtteliselt sama nagu programmides, kuid formalismi on vähem, nt ei kasutata eriti tüüpe (erineva täpsusega täis- ja reaalarve). Liikidega (arv, tekst, tõeväärtus) peab arvestama näiteks tehete ja funktsioonide kasutamisel. Konstandid kirjutatakse otse algoritmi, muutujad esitatakse nimede abil. Võiks arvestada, et algoritmis kasutatavad nimed sobiksid kasutamiseks ka programmis. Muutujate kasutamisel algoritmides peab arvestama, et programmis hakkavad neile vastama mälupesad ning omistamine seisneb väärtuse salvestamises arvuti mälus muutujale eraldatud pesas. Peamine korraldus muutujale väärtuse andmiseks on **omistamine**, ning selle võib esitada kujul:

muutuja = avaldis

Avaldiste esitamisele algoritmides erilisi nõudeid ja piiranguid ei seata. Oluline on, et see oleks inimesele arusaadav ja vastaks väärtuste liigile, näiteks aritmeetikatehteid ja matemaatikafunktsioone saab rakendada ainult arvudele.

Andmekogumite puhul arvestame esialgu **ühemõõtmeliste massiivide** ehk **loenditega**, mis on järjestatud elementide (muutujate) kogum. Loend (massiiv) tähistatakse ühe nimega, viitamiseks elementidele kasutatakse nime koos indeksiga: V(1), Y(k), X(i + 1) jms.

Objektide osas arvestame praegu, et tegemist on süsteemis määratletud ja realiseeritud objektide klassidega (nagu on Scratchis ja dokumendipõhistes süsteemides). Siia kuuluvad eeskätt graafikaobjektid. Teatud juhtumeil võivad tulla mängu ka dokumendid ja nende osad, vormid, dialoogiboksid, aknad jms.

Objekti käsitlemisel algoritmis peab mingil viisil viitama objektile, tema omadustele ning meetoditele. Viitamiseks objektile kasutatakse **nime**. Kui algoritm kehtib ainult kindla objekti jaoks (nagu Scratchis), siis võib objekti nime algoritmis jätta ära. Viitamiseks omadustele ja meetoditele kasutatakse määratud, kokkulepitud või üldarusaadavaid nimesid. Meetodite puhul kasutatakse sageli ka argumente.

pall.X = 0; pall.liigu x0, y0; pööra 180; muudaX 10

Sprait
nimi x asukoht, y asukoht suund, suurus värvus, nähtavus, ...
liigu(), pööra() muudaX(), muudaY() vaheta kostüümi() muuda värvi() ütle(), mängi heli(), ...

Näites on toodud valik Scratchi põhiobjektide (spraitide) omadusi ja meetodeid. Analoogsed omadused ja meetodid on graafikaobjektidel ka mitmetes teistes süsteemides (omaduste ja meetodite nimed on erinevad). Siin omadused **x asukoht** ja **y asukoht** määravad spraidi asukoha laval (X ja Y on koordinaadid). Omadus **suund** näitab spraidi pööret ning objekti liikumise suunda. Spraidi meetodid on realiseeritud käsuplokkide abil. Üldjuhul ei pea Scratchis realiseerimiseks mõeldud algoritmides tingimata kasutama täpselt käsuplokkide nimesid, kuid teatud määral tasub sellega siiski arvestada, kui algoritmi detailiseerimisel on jõutud viimasele tasemele. Algoritmi koostaja võib spetsifitseerida tegevused ja nende sisu. Koostaja võib ise määratleda ka oma objektid (**klassid**) ja spetsifitseerida nende jaoks omadused ja meetodid.

Üldiselt peaks algoritmi juurde kuuluma andmete spetsifikatsioon, milles näidatakse kasutatavad muutujad, andmekogumid ja objektid.

Algoritmide esitusviisid

Algoritmide esitamiseks kasutatakse erinevaid viise ja vahendeid:

- tavaline tekst,
- valemid,
- plokkskeemid,
- UML tegevusdiagrammid,
- Algoritmikeeled.

Antud materjalis kasutatakse peamiselt **UML**i tegevusdiagramme ja algoritmikeelt. **UML**i diagrammides kasutatakse teatud standardseid sümboleid ja kujundeid. Antud juhul kõiki standardite detaile väga täpselt ei arvestata. Peamised tegevusdiagrammide komponendid on järgmised:

● **protsessi** (protseduuri, skripti) **algus** **protsessi lõpp** (katkestus)

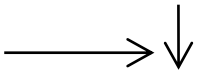
tegevus — Tegevusskeemide põhielement. Täidetav tegevus. Võib esitada erineva detailsusastmega

leia hind | koosta arve | lahenda võrrand | arvuta pindala

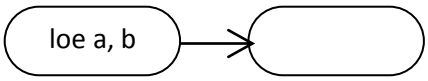
pall platsi keskele | pall lenda | **liigu** 20 sammu | **pööra** 30 kraadi

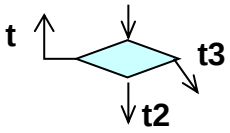
loe a, b, c | **arvuta** S, P | **kirjuta** S, P | **kuva** teade

S = 2πr(r + h) | D = b2 – 4ac | nimi = „Juku“ | S = 0; k = 0



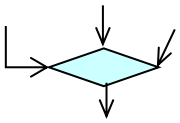
Siire. Näitab järgmist tegevust.





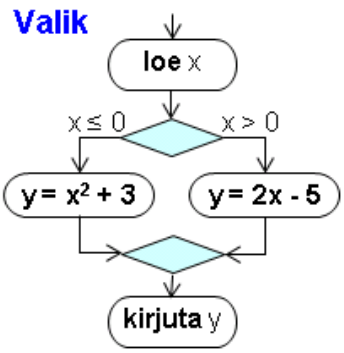
Hargnemine

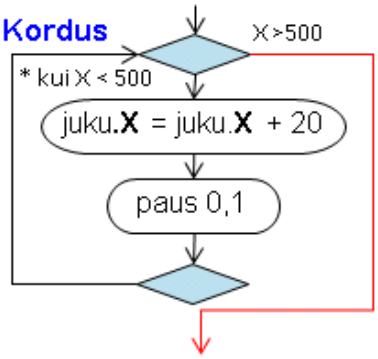
Tüüpiliselt üks sisend ja mitu väljundit
t1, t2, t3 – tingimused.



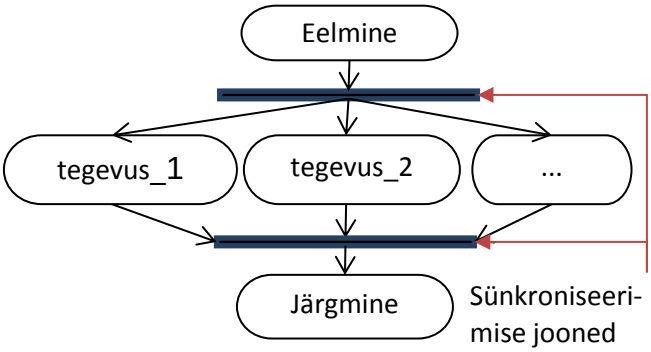
Ühinemine

Tüüpiliselt mitu sisendit ja üks väljund





Paralleelprotsessid ja tegevuste sünkroniseerimine



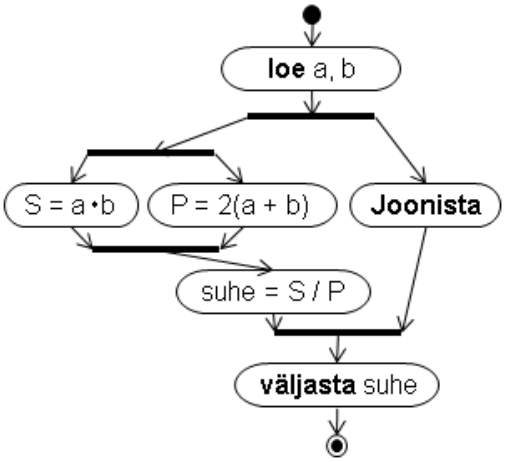
Sünkroniseerimise jooned

Tegevusi võib täita **paralleelselt**. Seda ei pea tegema, kui süsteem ei toeta paralleelsust või sellest ei ole erilist kasu. Ühtlasi tähendab see, et tegevuste täitmise järjekord pole oluline!

Algoritmikeeles esitatakse tegevused sarnaselt programmeerimiskeele lausetele, kuid olulisemalt nõrgema formalismiga ning nõuetega süntaksi õigsusele. Tegemist on kokkuleppega, mida rakendatakse algoritmide kirjeldamiseks näiteks töömeeskonnas, koolis või õpetamisel. Seejuures kasutatakse mõningaid kindla tähendusega sõnu ja fraase nagu võtmesõnad programmeerimiskeeltes või tegevusskeemides: **loe**, **kirjuta**, **kui**, **kordus**, …

Allpool on mõned algoritmide näited, mis on esitatud **UML** tegevusskeemide ja algoritmikeele abil.

Antud on ristküliku külgede pikkused (a, b). Leida selle pindala (S), ümbermõõt (P) ning pindala ja ümbermõõdu suhe. Joonistada ka ristkülik.



protseduur Joonista
mine 0, 0
pliiats alla
liigu a, 0
liigu a, b
liigu 0, b
liigu 0, 0

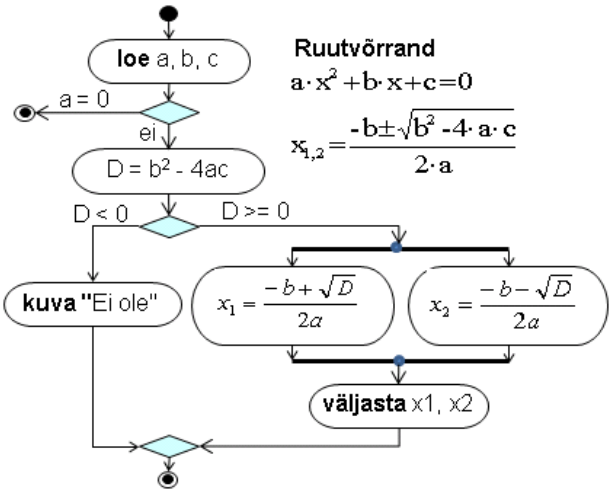
protseduur Rist
loe a, b
teavita Joonista
S = a * b
P = 2(a + b)
suhe = S / P
väljasta suhe

UML diagramm näitab, et arvutused ja joonistamine võivad toimuda paralleelselt, kuid enne peab lugema algandmed (a, b). Sellise variandi võimaldab realiseerida Scratch. Peale algandmete lugemist käivitatakse teatega „Läks“ samaaegselt joonestamise ja arvutamise skriptid. Need võivad töötada paralleelselt, sest ei sõltu üksteisest.

Antud ülesande juures aja kokkuhoidu praktiliselt ei teki ning lähenemisviis demonstreerib lihtsalt põhimõttelisi võimalusi. Skeem näitab, et ka S ja P arvutused võivad toimuda paralleelselt, kuid selle realiseerimiseks ei ole mingit mõtet. Selline esitus näitab, et nende kahe suuruse arvutamise järjekord ei ole oluline – S ja P arvutamise käsuplokkide järjekord võib olla programmis suvaline, kuid mõlemad peavad eelnema muutuja **suhe** väärtuse leidmisele.

Muutuja **suhe** väärtuse väljastamiseks ei ole joonestamise lõpetamise ootamine tingimata vajalik, sest see võib toimuda kohe peale arvutuste lõppu. Võite vaadata ka Scratchi projekti, kus on mõningaid täiendusi joonise mastaabi valimisel, mis ei ole kajastatud algoritmis.

Järgnevalt on esitatud ruutvõrrandi lahendamise algoritm. Peale algandmete (a, b, c) lugemist toimub kontroll, kas **a=0**; kui jah, siis programmi töö katkestatakse. Jätkamisel leitakse kõigepealt **D** väärtus. Kui **D<0**, lahend puudub, vastupidisel juhul leitakse ja väljastatakse **x1** ja **x2**. Vaatamiseks on demo.



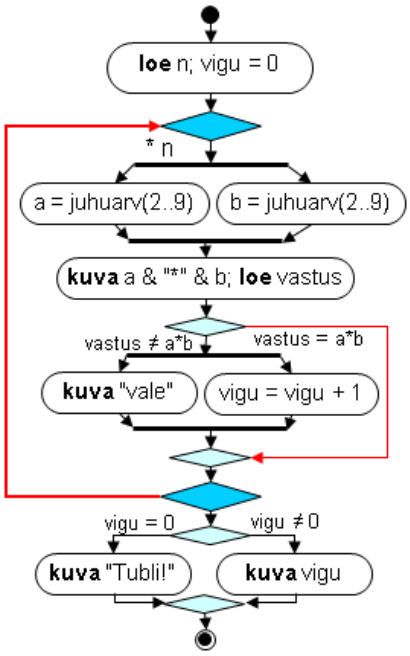
Ruutvõrrand

$$a \cdot x^2 + b \cdot x + c = 0$$

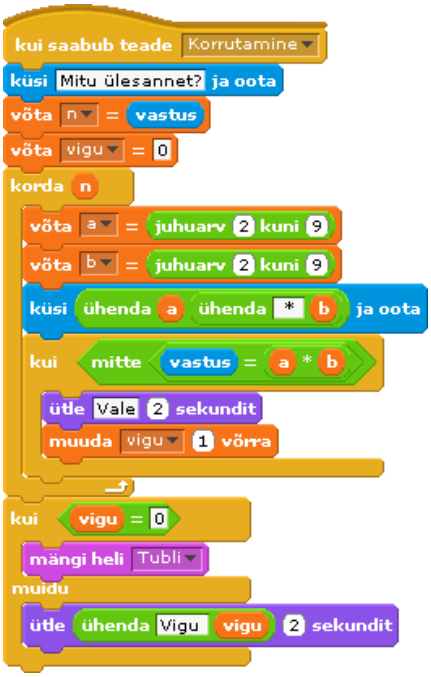
$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

protseduur Ruutvrd
loe a, b, c
kui a = 0 **siis lõpp**
D = b² - 4ac
kui D < 0 **siis**
kuva "Juured puuduvad"
muidu
$x_1 = \frac{-b + \sqrt{D}}{2a}$ $x_2 = \frac{-b - \sqrt{D}}{2a}$
kuva x1, x2
lõpp kui
lõpp Ruutvrd

Allpool on esitatud **algoritm** ja Scratchi **skript** rakenduse jaoks, mis esitab **n korrutamisülesannet**, kontrollib kasutaja vastuseid ja annab hinnangu. Vt ka Scratchi projekti.




```
protseduur Korrutamine
  loe n
  vigu = 0
  kordus n korda
      a ja b juhuarvud 2..9
      kuva küsimus
      loe vastus
      kui vastus <> a* b siis
        kuva "Vale"
        vigu = vigu + 1
  lõpp kordus
  kui vigu = 0 siis
      kuva  "Tubli!"
  muidu
      kuva vigu
  lõpp kui
```



Tüüptegevused märkandmetega

UML	Algoritmikeel	Scratch	Visual Basic (VB) Python

Väärtuste leidmine ja salvestamine (omistamine)

Leitakse avaldise väärtus ja salvestatakse see antud muutujas

UML	Algoritmikeel	Scratch	Visual Basic (VB) Python
muutuja = avaldis	muutuja = avaldis		muutuja = avaldis

muutuja – lihtmuutuja nimi, massiivi element: nimi(indeks), nimi(rida, veerg)

avaldis – eeskiri väärtuse leidmiseks: operandid, tehtemärgid, funktsioonid
arvavaldised, tekstavaldised, ajaavaldised, võrdlused, loogikaavaldised

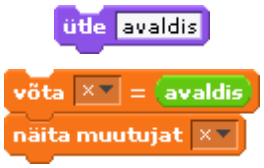
Väärtuste lugemine väliskeskkonnast

Loetakse väärtused väliskeskkonnast (boks, vorm, dokument, fail, …) ja salvestatakse muutujates

UML	Algoritmikeel	Scratch	Visual Basic (VB) Python
sisesta muutuja,	**küsi** muutuja, … **loe** muutuja, … **sisesta** muutuja, …		**Visual Basic** muut = InputBox() **Python** muut = input()

Väärtuste kirjutamine väliskeskkonda

Leitakse avaldise väärtus ja kuvatakse või kirjutatakse väliskeskkonda (boks, vorm, dokument, fail, …)

UML	Algoritmikeel	Scratch	Visual Basic (VB) Python
väljasta avaldis	**kuva** avaldis **kirjuta** avaldis		**Visual Basic** MsgBox avaldis **Python** print(avaldis, …)

Tüüptegevused objektidega

UML	Algoritmikeel	Scratch	Visual Basic Python

Objektide omaduste lugemine ja muutmine

Viitamiseks objekti omadusele kasutatakse tüüpiliselt konstruktsiooni objekt.omadus

UML	Algoritmikeel	Scratch	Visual Basic Python
muutuja = objekt.omadus objekt.omadus=avaldis	muutuja = objekt.omadus objekt.omadus = avaldis		X = auto.Left auto.Left = a+10

Objektide meetodite rakendamine

Viitamiseks objekti meetodile kasutatakse tüüpiliselt konstruktsiooni objekt.meetod *argumendid*

UML	Algoritmikeel	Scratch	Visual Basic Python
objekt.meetod arg-d	objekt.meetod arg_d		auto.IncrementLeft 10 auto.setx(120)

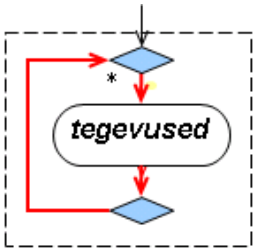
Protsesside juhtimine

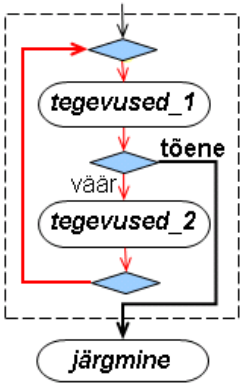
Protsesside juhtimine seisneb tegevuste täitmise järjekorra määramises. Eristatakse nelja liiki protsesse:

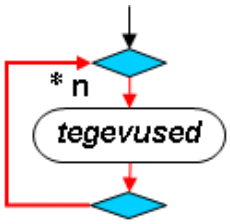
- järjestikune protsess ehk jada,
- tsükliline protsess ehk kordus,
- hargnev protsess ehk valik,
- paralleelne protsess.

Protsesside kirjeldamiseks kasutatakse vastavaid kokkuleppeid, korraldusi või käske. Järjestikuste protsesside määramiseks mingeid spetsiaalseid vahendeid ei kasutada – eeldatakse, et tegevusi täidetakse selles järjekorras nagu need on algoritmis esitatud.

Kordused

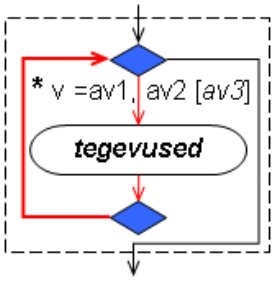
UML	Algoritmikeel	Scratch	Visual Basic	Python
Lõputu kordus				
	kordus *tegevused* lõpp kordus		Do *laused* Loop	while True: *laused*
Lõputu kordus katkestusega				

	kordus
	tegevused 1
	kui *ting* siis välju
	tegevused 2
	lõpp kordus
	NB! Katkestamise tingimusi võib olla mitu!
			Do
			laused 1
			If *ting* Then
			Exit Do
			End If
			laused 2
			Loop
				while True:
				laused 1
				if ting:
				break
				laused 2

Etteantud korduste arvuga kordus				
	kordus *n* korda *tegevused* lõpp kordus **NB!** Tegevuste seas võivad olla katkestajad!		For i = 1 T0 n *laused* Next i **NB!** VBs näitab käsu jätkumist _	for i in \ range(n): *laused* **NB!** Pythonis näitab käsu jätkumist ****

UML	Algoritmikeel	Scratch	Visual Basic	Python

Juhtmuutujaga kordus



kordus *v* = a1..a2 [,a3]

tegevused

lõpp kordus

Juhtmuutuja ***v*** saab väärtusi a1-st a2-ni sammuga a3

Puudub. Saab modelleerida näiteks taoliselt

For v=a1 To a2 _
[Step a3]

laused

Next v

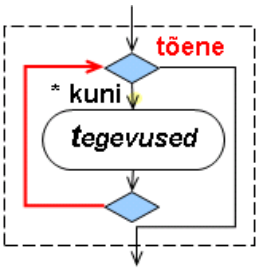
NB! VBs näitab käsu jätkumist _

for v in \
range \
(a1,a2,a3):

laused

NB! Pythonis näitab käsu jätkumist ****

Eelkontrolliga kordus kuni tingimus saab tõeseks



kordus kuni *ting*

tegevused

lõpp kordus

korratakse, kuni tingimus **saab tõeseks**

Do Until *ting*

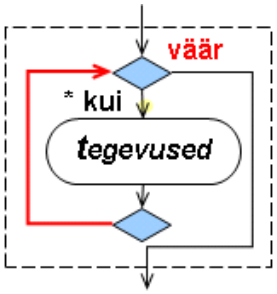
laused

Loop

while not *ting*:

laused

Eelkontrolliga kordus kui tingimus on tõene



kordus kui *ting*

tegevused

lõpp kordus

korratakse, kui tingimus **on tõene**

puudub

Do While *ting*

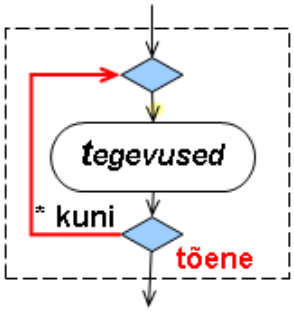
laused

Loop

while *ting:*

laused

Järelkontrolliga kordus kuni tingimus saab tõeseks



kordus

tegevused

lõpp kuni *ting*

korratakse, kui tingimus pole tõene (kuni saab tõeseks)

puudub

Do

laused

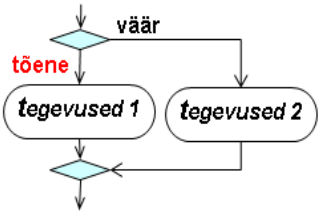
Loop Until *ting*

puudub

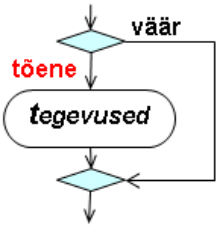
Valikud

Valik kahest

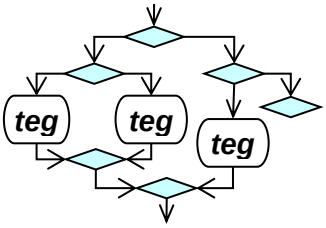
UML	Algoritmikeel	Scratch	Visual Basic	Python

	kui *tingimus* siis
	tegevused 1
	muidu
	tegevused 2
	lõpp kui
			If *tingimus* Then
			laused 1
			Else
			laused 2
			End If
				if *tingimus*:
				laused 1
				else :
				laused 2

Valik ühest

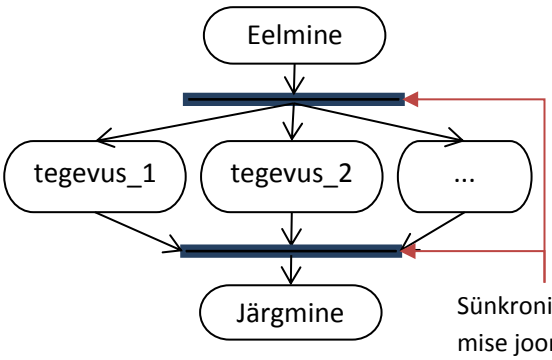
	kui *tingimus* siis
	tegevused
	lõpp kui
	
	kui *tingimus* siis *tegevus*
			If *tingimus* Then
			laused
			End If
			
			If *ting* Then *lause*
				if *tingimus*:
				laused
				
				if *ting: lause*

Mitmene valik

	kui *ting* siis
	kui ting siis
	tegevused 1
	muidu
	tegevused 2
	muidu
	kui *ting* siis *tegev*
	lõpp kui
			If *ting* Then
			laused
			[ElseIf *ting* Then
			laused]
			…
			[Else
			laused]
			End If
				if *ting*:
				laused
				[elif *ting*:
				laused]
				…
				[else:
				laused]

Üldjuhul võib valiku tegemiseks olla suvaline hulk tingimusi. Erinevates programmeerimiskeeltes on mitmese valiku jaoks olemas spetsiaalsed laused.

Paralleelsed protsessid

UML	Algoritmikeel	Scratch	Visual Basic
 Sünkroniseerimise jooned	Kokkuleppeline esitusviis	Ühetüübilised päiseplokid 	puudub Python siin ei vaadelda

Mitmest üksusest koosnevad rakendused

Programmid koosnevad sageli mitmest omavahel seotud üksusest – protseduurist, funktsioonist või skriptist. Olenevalt sellest vastab igale osale omaette algoritm. Programmi üksuste st ka algoritmi üksuste vahel toimub koostöö:

- pöördumine ühest üksusest teise poole,
- andmevahetus üksuste vahel.

Võib eristada kahte liiki üksusi:

- funktsioonid,
- protseduurid.

Erinevates programmeerimiskeeltes kasutatakse üksuste jaoks erinevaid nimetusi ja teatud erinevusi on ka nende kasutamisel, kuid üldised põhimõtted on samad.

Funktsioonid. Parameetrid ja tagastatav väärtus

Funktsioonid on mõeldud peamiselt väärtuste leidmiseks (tuletamiseks). Tüüpiline funktsioon leiab ja **tagastab ühe väärtuse**.

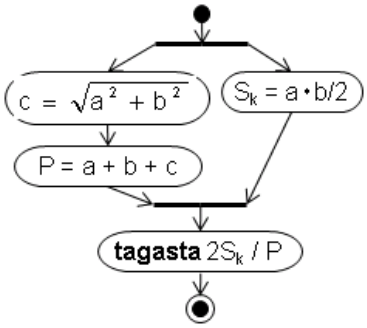
Allpool on toodud funktsioon, mis leiab täisnurkse kolmnurga siseringi raadiuse.

Esitatud on UML tegevusdiagrammi kaks varianti ning funktsiooni esitus algoritmikeeles, Pythonis, Visual Basicus, Scratchi versioonis 1.4 ja BYOBis.

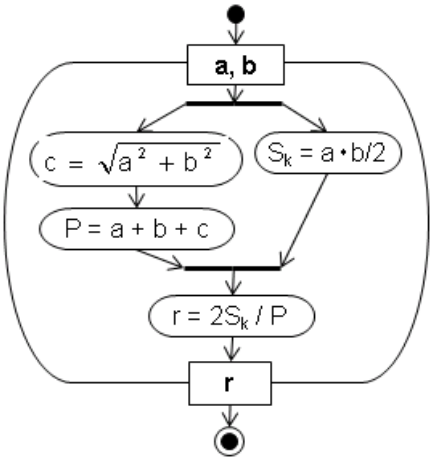
Viimase kolme kasutamises juures on näidatud ka ringi pindala leidmist.

Täisnurkse kolmnurga siseringi raadius: variant 1

parameetrid: a, b – kaatetid
tagastatav väärtus - r



Täisnurkse kolmnurga siseringi raadius: variant 2. Parameetrid ja tagastatav väärtus on skeemil



Algoritmikeel

funktsioon *Sirira*(a, b)
c = **sqrt**($a^2 + b^2$)
P = a + b + c
S_k = a*b/2
tagasta 2*S_k / P

Pythoni funktsioon

```
import math
def Sirira(a, b):
    c = math.sqrt(a*a+b*b)
    P = a+b+c
    Sk = a*b/2
    return 2*Sk/P
```

Kasutamine (pöördumine)
print(**Sirira**(4,5))

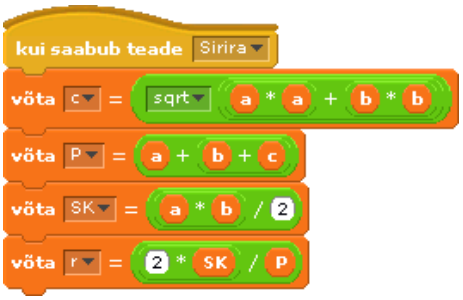
VB funktsioon

```
Function Sirira(a, b)
Dim c, P, Sk
c = Sqr(a ^ 2 + b ^ 2)
P = a + b + c
Sk = a * b / 2
Sirira = 2 * Sk / P
End Function
```

Kasutamine

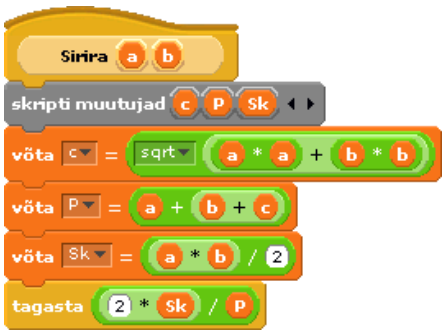
```
r = Sirira(k1, k2)

Sr = 3.14 * r * r
```

Scratch versioon 1.4



Kasutamine

BYOB



Kasutamine

Funktsioonides saab kasutada sisendandmete esitamiseks **parameetreid** ehk tinglikke muutujaid, mis saavad väärtused vastavatelt **argumentidelt** funktsiooni poole pöördumisel. Parameetreid kasutatakse (koos funktsiooni sisemuutujatega) tulemuse leidmiseks. Toodud näites on parameetriteks kaatetite **a** ja **b** pikkused.

Nagu öeldud, leiab ja tagastab funktsioon tavaliselt **ühe väärtuse**, milleks toodud näites on raadius **r**. Tagastatav väärtus määratletakse vastavas protseduuris ühel või teisel moel. Üsna levinud on ***return*** (tagasta) lause kasutamine (Pythonis) või tulemuse omistamine funktsiooni nimele (VB-s). UML skeemidel võib parameetreid ja tagastatavat väärtust näidata skeemi ümbritseva kujundi servadel. Algoritmikeeles, Pythonis, VB-s ning enamikes programmeerimiskeeltes on parameetrid funktsiooni päises funktsiooni nime järel sulgudes.

Scratchi versioonis 1.4 funktsioone (seega ka parameetreid ja argumente) ei ole. Neid saab modelleerida tavalise skriptina, mille võiks teha eraldi spraidi jaoks. Parameetritele ja tagastatavale väärtusele vastavad suurused on otstarbekas määratleda globaalsete muutujate abil (siin **a**, **b**, **r**) ning nö abisuurused lokaalsete muutujate abil (siin **c, P, Sr**).

Scratchi uuemates versioonides peaks saama kasutada ka funktsioone, parameetreid ja argumente, luues vastavaid plokke. Praegu arendavad seda versiooni Berkeley ülikool ja MIT koos BYOB nime all.

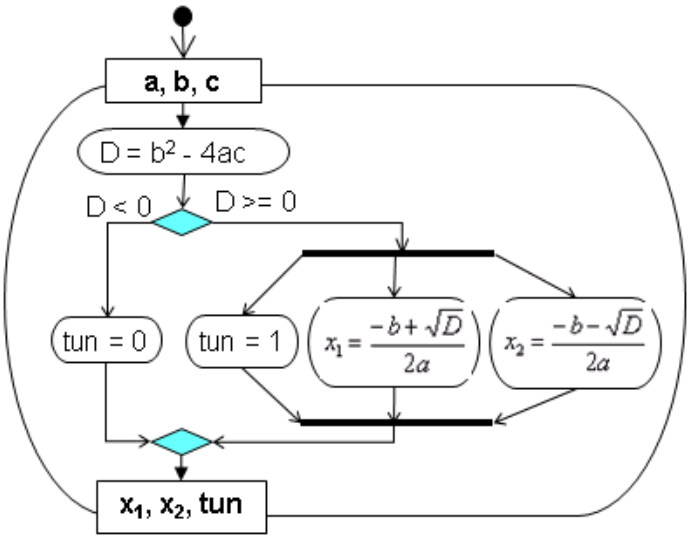
Funktsiooni kasutamine toimub nn **funktsiooniviida** abil, mis esitatakse kujul:

f_nimi(argumendid)

f_nimi – funktsiooni nimi; **argumendid** – parameetritele vastavad väärtused, mis võivad olla esitatud konstantidena, muutujatena või avaldistena. Argumentide arv ja järjestus peab vastama parameetritele.

Protseduurid. Sisend- ja väljundparameetrid

Protseduurid on tunduvalt üldisema iseloomuga kui funktsioonid, võivad teha suvalisi tegevusi, mida antud programmeerimiskeeles saab kasutada, sh leida näiteks ka väärtusi. Allpool esitatud näites on algoritm ruutvõrrandi lahendamiseks realiseeritud parameetritega protseduurina.



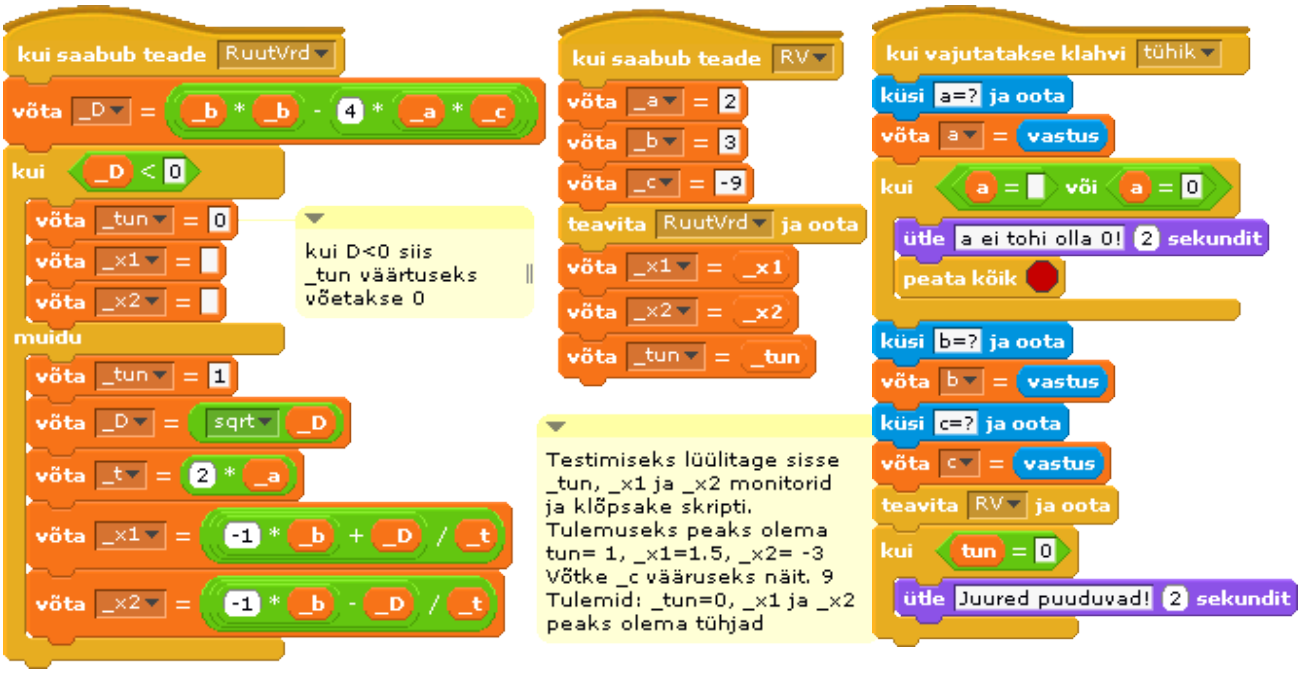

```
protseduur Ruutvrd (a, b, c, x1, x2, tun)
 D = b² - 4ac
 kui D < 0 siis
  tun = 0
 muidu
   tun = 1
   x1 = (-b+√D)/2a    x2 = (-b-√D)/2a
 lõpp kui
lõpp Ruutvrd
```

Protseduuri **Ruutvrd** sisendparameetriteks on ruutvõrrandi kordajad **a**, **b** ja **c**, väljundparameetriteks on lahendid **x1** ja **x2** (kui need leiduvad) ja tunnus **tun**, mis näitab, kas lahendeid on (tun = 1) või ei ole (tun = 0).

Ruutpea kujutab endast peaprotseduuri, mis loeb algandmed, kontrollides kohe, kas **a** = 0. Kui jah, siis kuvatakse vastav teade ja katkestatakse programmi täitmine. Kui **a** ei ole 0 (null), siis loetakse ka **b** ja **c** väärtused ning pöördutakse alamprotseduuri **Ruutvrd** poole.

```
protseduur Ruutpea
  loe a
  kui a = 0 siis
    kuva „a ei tohi olla 0!“
    stopp
  lõpp kui
  loe  b, c
  Ruutvrd a, b, c, x1, x2, tunnus
  kui tunnus = 0 siis
    kuva “Lahendid puuduvad!“
  muidu
     kuva x1, x2
lõpp kui
```

Scratchi kasutatud versioonis 1.4 ei ole parameetrite ja argumentide kasutamine võimalik, kuid seda saab modelleerida.



Võib teha spetsiaalse spraidi, millel on kaks skripti. Esimene **RuutVrd** realiseerib ruutvõrrandi lahendamise ülaltoodud algoritmi järgi, kusjuures kasutatakse spraidi lokaalseid muutujaid, mille nimed algavad siin allkriipsuga, et neid oleks lihtsam eristada tavalistest muutujatest. **RV** on liidese mall. Nende skriptidega spraiti on otstarbekas eksportida, selleks et hiljem saaks seda lisada suvalisse projekti ning sellega siduda.

Sisendparameetritele **_a**, **_b** ja **_c** ja väljundparameetritele **_x1**, **_x2**, **_tun** seatakse vastavusse argumendid: **a**, **b** ja **c**, ning **y1**, **y2** ja **tun**.

Näites on toodud skript, mis loeb kordajad ja käivitab skripti **RV**. See seob omavahel parameetrid ning argumendid ja käivitab skripti **RuutVrd**. Võimalik on tutvuda ka vastava projektiga.

Näitena on toodud sama algoritmi realiseerimine Visual Basicuga:

```
Sub RuutPea()
  Dim a, b, c, y1, y2, tun
   a = InputBox("Anna a")
   If a = "" Or a = "0" Then
     MsgBox "a ei tohi olla 0!": End
   End If
   b = InputBox("Anna b")
   c = InputBox("Anna c")
   RuutVrd a, b, c, y1, y2, tun
   If tun = 0 Then
     MsgBox "Lahend puudub"
   Else
     MsgBox "y1="&y1& " y2="&y2
   End If
End Sub
```

```
Sub RuutVrd(a,b,c, x1,x2, tun)
  Dim D
  D = b ^ 2 - 4 * a * c
  If D < 0 Then
     tun = 0
  Else
     tun = 1
     x1 = (-b + Sqr(D)) / (2 * a)
     x2 = (-b - Sqr(D)) / (2 * a)
  End If
End Sub
```

RuutVrd on klassikaline parameetritega protseduur:
a, **b**, **c** – sisendparameetrid,
x1, **x2**, **tun** – väljundparameetrid.

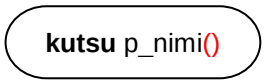
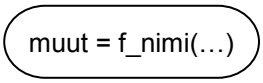
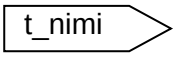
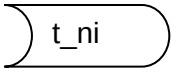
Sisendparameetrid saavad väärtused samanimelistelt argumentidelt, mida kasutades leitakse tulemused ja omistatakse need parameetritele **tun**, **x1**, **x2**.

Parameetrite väärtused omistatakse vastavalt argumentidele **y1**, **y2**, **tun**.

Peaprotseduuris on muutujate nimedeks **y1** ja **y2** näitamaks, et parameetrite ja argumentide nimed ei pea olema samad.

Pöördumine ühest üksusest teise poole

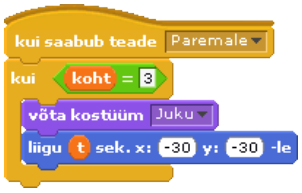
Allpool on toodud tüüpilised elemendid koostöö kirjeldamiseks programmiüksuste vahel.

UML	Algoritmikeel	Scratch	Visual Basic	Python
Pöördumine protseduuri poole				
	**kutsu** p_nimi(arg, …)	 (1.4)  ver (2.0)	**Call p_nimi** (arg, …) **p_nimi arg, …**	**f_nimi** (…)
Pöördumine funktsiooni poole				
	muut = f_nimi(…)	Scratchi vers 1.4 ei ole  (2.0)	muut = f_nimi(…)	
Teadete tekitamine				
 	**teavita** t_nimi	 	puudub	
Teadete vastuvõtmine				
	**teade** t_nimi			**puudub**

Rakenduste loomine Scratchiga

Scratch on uue põlvkonna graafiline programmeerimissüsteem, mis võimaldab lihtsalt ja kiiresti omandada algoritmimise ja programmeerimise põhimõtteid ning tüüpilisi vahendeid ja tegevusi, mida kasutatakse ka tavapärastes tekstipõhistes programmeerimiskeeltes.

Scratchis pannakse programm kokku käsuplokkidest hiire abil. Süntaksiprobleemid selles keeles praktiliselt puuduvad.



Sissejuhatus programmeerimissüsteemi Scratch

Scratch on uue põlvkonna graafiline programmeerimissüsteem, mis on loodud Massachusetts'i Tehnoloogiainstituudis (*Massachusetts Institute of Technology – MIT*) programmeerimise õppimiseks ja õpetamiseks. Scratchi versioon 1.4 on ilmunud 2009. aastal, millest alates algas süsteemi kiire levik.

Scratchi saab tasuta alla laadida aadressilt http://scratch.mit.edu. Seal leidub suurel hulgal õppe- ja abimaterjale tõlgituna erinevatesse keeltesse (sh eesti keelde) ning näiteid. Scratchi serverid pakuvad ka pilveteenuseid – paari hiireklõpsuga saab oma rakenduse (projekti) tõsta „pilve" ja täita seal Java apletina või Flashi klipina. Vastavad teisendused tehakse serveris automaatselt. Praegu on serveris juba mitu miljonit projekti, mida võib vabalt alla laadida, uurida ja remiksida. Uues versioonis on olemas võimalus Scratchi rakenduste loomiseks ja täitmiseks brauseris. Lisaks toimuvad arendustööd selliste vahendite loomiseks, mis võimaldavad kasutada Scratchi rakendusi ka nutiseadmetel nagu tahvelarvutid ja mobiiltelefonid.

Scratchis on lihtne ja mugav rakenduste loomise keskkond ja kasutajaliides. Süsteemiga on kaasas rikkalik valik graafika- ja heliobjekte, kuid neid saab lisada ka oma failidest ja internetist. Pilte saab teha veebikaameraga ja heliklippe salvestada rakenduste loomise käigus, kui arvutil on mikrofon. Oma graafikaobjektide loomiseks või olemasolevate muutmiseks on võimalik kasutada keskkonda sisseehitatud joonistamisredaktorit. Scratchis on pööratud suurt tähelepanu atraktiivsusele ning multimeedia kasutamisele – lihtsalt ja kiirelt on võimalik luua mänge, animatsioone, koomikseid, esitlusi jms. Samas on võimalik lahendada ka arvutuslikke, andmetöötluse- ja graafikaülesandeid: nt realiseerida, uurida ja visualiseerida klassikalisi algoritme massiividega (summade, keskmiste, ekstreemumite leidmine, erinevad otsimise ja sorteerimise meetodid demo), teha joonistusi ja diagramme sealhulgas funktsioonide graafikuid jms.

Scratchil on juba üsna mitmeid järglasi ja analooge. Neist tähelepanuväärseim on Berkeley ülikoolis arendatav süsteem **BYOB** (*Build Your Own Blocks*) (http://byob.berkeley.edu).

See on ühilduv Scratchiga ja momendil on oma võimaluste poolest isegi viimasest veidi ees. BYOB võimaldab luua oma plokke (käske), mis vastavad oma olemuselt Scratchi plokkidele ja kujutavad endast parameetritega funktsioone ja protseduure (skripte). BYOBis saab luua ka oma objekte ja klasse.

BYOB võimaldab transleerida enda ja ka Scratchi projekte masinkoodi, saades Windowsi EXE-faili. Praegu on lõpetamisel BYOBi versioon, mis võimaldab luua rakendusi brauseris. Süsteemi nimeks peaks siis saama SNAP!. Uues versioonis peaks ka Scratchil olema analoogsed võimalused.

Veel üheks Scratchi modifikatsiooniks, mis omab võrreldes Scrathiga täiendavaid võimalusi, on **Panther** (http://pantherprogramming.weebly.com).

On tekkinud mitmeid graafilisi programmeerimissüsteeme, mis on Scratchiga sarnased.

Kõigepealt võiks nimetada suurfirma Google poolt arendatavat süsteemi **Blockly** (http://code.google.com/p/blockly). Ka siin kasutatakse Scratchi käsuplokkidele sarnaseid elemente ning skriptid pannakse hiire abil kokku otse brauseris. Süsteemi üheks huvitavaks eriomaduseks on võimalus protseduure tõlkida erinevatesse tekstipõhistesse programmeerimiskeeltesse nagu Python, JavaScript, Dart ja XML. Üsna sarnane Blocklyga on ka **MIT App Inventor** (http://appinventor.mit.edu), mis on ette nähtud Androidi rakenduste loomiseks mobiilidele. Veel üks omapärane eksperimentaalne süsteem on **Calico** (vt http://calicoproject.org), kus ühes ja samas kasutajaliideses saab kasutada erinevaid programmeerimissüsteeme: graafilist Scratchiga sarnanevat süsteemi, keelt **Jigsaw**, Pythonit, Rubyt, robotite programmeerimise vahendit Myro, …

Scratch ja BYOB leiavad järjest laiemad kasutamisest programmeerimise õpetamise esimeste keeltena. Need sobivad igas vanuses õppuritele, alates põhikoolide algklassidest kuni kolledžite ja ülikoolide programmeerimise sissejuhatavate kursusteni. Käsitletavate teemade ja lahendatavate ülesannete valikul tuleb muidugi arvestada õppijate vanust, taset, õpetamise eesmärke jm. Peab ütlema, et valikuvõimalusi ja variatsioone on siin tunduvalt rohkem kui traditsiooniliste programmeerimiskeelte kasutamisel.

USA arvutiõpetajate assotsiatsiooni (CSTA) poolt koostatud arvutiteaduse tüüpõppekavas (ülevaade, kirjeldused) põhineb aine „Sissejuhatus programmeerimisse“ täielikult Scratchil. Scratchi või BYOBiga alustatakse programmeerimise kursust näiteks CS50 Harvardi Ülikoolis (vt https://www.cs50.net/psets), Wisconsini Ülikoolis (vt link) jt. Berkeley Ülikoolis põhineb mitteinformaatikute kursus „The Beauty and Joy of Computing“ BYOBil ja Scratchil (vt http://bjc.berkeley.edu).

Esitatud kursuse antud osa eesmärgiks ei ole programmeerimiskeele Scratch õppimine. Scratch on siin lihtsalt vahendiks rakenduste loomise ning programmeerimise ja algoritmimise aluste omandamisel. Scratchi abil on võimalik omandada kiirelt kõik peamised programmeerimise, algoritmimise, modelleerimise ja rakenduste loomise põhimõtted ja tüüptegevused: eri liiki andmete (märk-, graafika-, heli- ja videoandmed) olemus ja kasutamise iseärasused, operatsioonid andmetega ja avaldised, andmete sisestamine ja väljastamine, joonistamise põhimõtted, muutujad ja massiivid, protsesside liigid ja nende kirjeldamine. Jada, kordus, valik ja paralleelsus, programmide jagamine üksusteks ning koostöö ja andmevahetuse korraldamine nende vahel, rakenduste kasutajaliideste loomise põhimõtted, objektorienteeritud lähenemisviisi ja modelleerimise olemus. Kuigi Scratch ei toeta antud momendil kõiki objektorienteeritud (OO) programmeerimise võimalusi (tegemist on nn objektipõhise süsteemiga), võimaldab ta tutvustada ja osaliselt rakendada OO lähenemisviisi.

Scratchi materjalide esitatamine kahel tasemel

Võimaldamaks õppijal saada kiiresti ülevaade Scratchist, on esimesel tasemel toodud lühivaade Scratchi kasutamisest rakenduste loomiseks. Sellega on soovitatav tutvuda kohe alguses, paralleelselt esimeste harjutustega.

Antud osaga on linkidega seotud lisamaterjalid (teine tase), kust saab täiendavat informatsiooni vastava teema kohta. Lisamaterjalidele võivad olla viited praktikumide töölehtedele.

Modelleerimine	Rakenduste loomine	Algoritmimine
Scratchi põhiobjektid	Skriptide loomine	Protsesside juhtimine
Muutujad	Rakenduste loomine Scratchiga	Andmed ja avaldised
Joonistamine	Loendid	Algoritmid loenditega

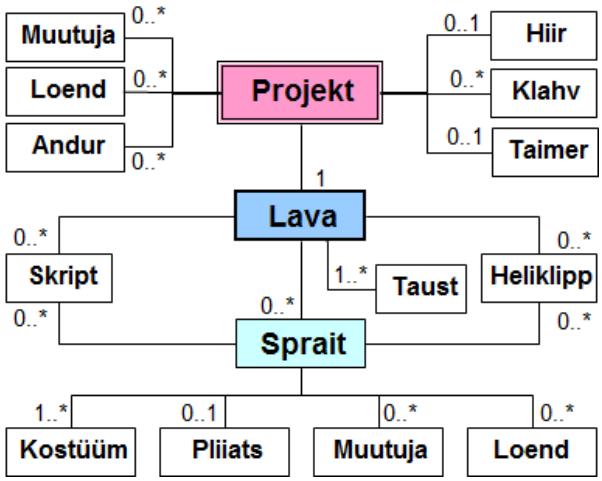
Ülevaade Scratchi rakenduste komponentidest ja vahenditest

Selles jaotises on lühiülevaade Scratchi põhikontseptsioonidest, komponentidest ja vahenditest. Vajadusel saab, kasutades tekstis toodud linke, põhjalikumat infot lisamaterjalidest.

Scratchi rakendusi nimetatakse **projektideks**. Projekti põhikomponentideks on spraidid ja skriptid, mille abil saab luua väga erineva otstarbe, sisu ja struktuuriga projekte (vt näiteid Scratch wiki ja Scratchi koduleht).

Scratchi projekti struktuur ja põhiobjektid

Scratchi projekt koosneb omavahel seotud objektidest. Kasutatavad **objektid** ja **seosed** nende vahel on esitatud UML klassidiagrammina (vt modelleerimine).



Seostel toodud kordused näitavad, mitu objekti ühest klassist võivad olla seotud teise klassi objektiga. Vaikimisi on kordus 1.

Projektis on alati üks ja ainult üks lava (kordus 1), millel võib asuda suvaline arv spraite, sh ka mitte ühtegi (kordus 0..*). Selline kordus on mitmel teisel objektil: skript, heliklipp, muutuja jms.

Laval on alati vähemalt üks taust ja spraidil vähemalt üks kostüüm (1..*). Kordus 0..1 näitab, et kasutusel võib olla ainult üks objekt või mitte ühtegi (hiir, taimer, pliiats).

Projekti loomisel ja täitmisel saab kasutada ja muuta objektide **omadusi** (spraidi koordinaadid, suurus, nähtavus, muutuja väärtus jms). Scratchi käsuplokid on objektide **meetodid**, mille abil saab määrata tegevusi spraitide ja nende alamobjektidega (pliiats, heli).

Lisamaterjal: Scratchi põhiobjektid, Scratch wiki.

Spraidid

Scratchi rakenduste kesksteks komponentideks on **objektid**, mida nimetakse **spraitideks** (inglise k ***sprite*** – haldjas või vaim). Programmide (skriptide) abil saab määrata mitmesuguseid tegevusi spraitidega: muuta asukohta, suurust, värvust, panna kõndima, tantsima, tekitama helisid, arvutama, joonistama jms. Süsteemiga tuleb kaasa üsna suur valik spraite, mis on süstematiseeritud liikide kaupa (inimesed, loomad, asjad, …) erinevatesse kaustadesse. Iga spraidiga on seotud teatud valik **omadusi** nagu nimi, asukoht laval (x, y), suurus, värvus, … ning **meetodeid**, mis on realiseeritud käsuplokkidena. Spraitide loomisel saab kasutada ka oma graafikaobjekte PNG, GIF, JPG või BMP ning animeeritud GIF-faile. Spraitide kostüüme võib luua ja muuta **joonistamisredaktori** abil (vt Kasutamisjuhend ja Scratch wiki).

Spraidid tegutsevad ekraanil piirkonnas, mida nimetatakse **lavaks**.

Spraidil võib olla mitu **kostüümi** ehk **kuvandit** (*costume*) – tüüpiliselt spraidi erinevad variandid e teisikud. Praktiliselt kujutab sprait endast graafiliste kujundite (kostüümide) kogumit. Kui sprait on nähtav, siis on nähtav üks tema kostüümidest e kuvanditest. Vahetades kostüüme programmi käskude abil teatud

sagedusega, saab tekitada animatsiooniefekte nagu kõndimine, tantsimine või muud. Kostüüme võib kasutada ka muul viisil, näiteks erinevate teadete kuvamiseks.

Lisamaterjal: Objektid Scratchis, Kasutamisjuhend, Scratch wiki.

Skriptid ja käsud

Scratchi projektis kasutatakse programmiüksusi, mida nimetakse **skriptideks** (*script*). Skript koosneb **käskudest** ehk **lausetest**, mis on realiseeritud graafiliste plokkidena. Plokid (käsud) on jagatud funktsioonide (liikumine, juhtimine, helid jm) alusel gruppidesse. Käsud jagunevad **liht**- ja **liitkäskudeks**, millest viimased sisaldavad teisi liht- ja/või liitkäske.

Plokid ühendatakse omavahel hiire abil tegevuste toimumise järjekorras. Skript algab **päiseplokiga**, mis on seotud skripti käivitamisviisiga: rohelise lipu klõpsamine, vajutus klahvile klaviatuuril, teate saabumine jms.

Skript kuulub alati ühele kindlale spraidile (või lavale). Ühel spraidil võib olla suvaline hulk skripte. Skriptid saavad teha omavahel koostööd teadete abil.

Programmi jagamine mitmeks omavahel koostööd tegevaks üksuseks on programmeerimises väga tähtsaks põhimõtteks. See võimaldab luua paindlikke ja ökonoomseid rakendusi, kasutada üksusi korduvalt nii samas kui ka teistes rakendustes, täita tegevusi paralleelselt, …

Lisamaterjal: Skriptid Scratchis, Kasutamisjuhend, Scratch wiki.

Scratchis on koostöö skriptide vahel rajatud **teate saatmisele** ühe skripti poolt ja selle **vastuvõtmisele** teiste skriptide poolt. Käsuga [**teavita *nimi***] või [**teavita *nimi* ja oota**] väljastatakse teade (signaal), mille võib vastu võtta ja käivituda suvaline arv skripte päisega [**kui saabub teade *nimi***], kus ***nimi*** on sama mis oli käsus [**teavita** …]. Tegemist võib olla sama spraidi skriptidega või teise spraidi või lava skriptidega. Teate vastuvõtnud skriptid alustavad tööd paralleelselt.

Näiteks võib mitmel spraidil olla pildil esitatud skript, mis kõik hakkavad tööle, kui mingi skript saadab teate [**teavita *hüppa***…]. Sama päisega (nimega) skriptid võivad olla ka erineva sisuga.

Kui käsus [**teavita** …] on täiend **oota**, siis peatatakse teate saatnud skripti täitmine, kuni lõpeb kõikide teate vastuvõtnud skriptide täitmine ning peale seda jätkub antud skripti töö. Kui täiendit **oota** käsus ei ole, jätkab teate saatnud skript oma tööd kohe pärast teate saatmist ja töötab paralleelselt teatele reageerinud skriptidega.

Rakenduste loomise ja kasutamise liidesed

Scratchi kasutajaliides (projekti loomise liides) kujutab endast ekraanipiirkondade kogumit, kus asuvad vahendid rakenduste loomiseks. Selle abil realiseeritakse tavaliselt rakenduse kasutajaliides, mille abil kasutaja suhtleb valmisrakendusega. Lisamaterjal: Kasutamisjuhend, Scratch wiki.

Pildil on Scratchi rakenduste loomise liidese põhielemendid. Akna ülaservas on menüü ja tööriistariba.

Liidese aknas võib eristada järgmisi alasid: **lava**, **spraitide loetelu** (kasutatud spraitide ja lava ikoonid), **aktiivse spraidi jooksev info**, **skriptide** ala, **kostüümide** ala, **helide** ala, **käskude plokid** ja vastavad **plokkide grupid**.

Lava kohal asuvate nuppude abil saab valida lava kolme oleku (suuruse) vahel: **normaalne**, **väike** ja **esitlus**. Suuruse valik muudab liidese erinevate alade jaotust. Esitluse olekus on täisekraanil nähtav ainult lava ning käivitamise ja peatamise nupud. Selles olekus ei saa lisada uusi spraite ja skripte.

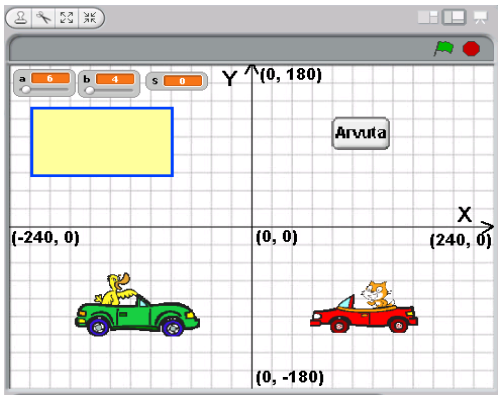
Rakenduse kasutamise liides luuakse laval. Liidese elementideks võivad olla taustad, aktiivsed ja passiivsed spraidid, käsunupud, klahvid, muutujate monitorid, …

Teistes süsteemides kasutatakse liidesena tavaliselt vorme ja dialoogibokse.

Lava

Lava (*Stage*) on ekraani piirkond, kus tegutsevad kõik spraidid.

Lisamaterjal: Objektid Scratchis, Kasutamisjuhend, Scratch wiki.



Laval võib olla mitu **tausta**. Vaikimisi on selleks valge taust. Taustu saab lisada, eemaldada, muuta. Taust on graafikaobjekt (pilt) ning oma olemuselt analoogne spraidi kostüümiga.

Koordinaattelgede algus on lava keskel. Lava on laiuseks on 480 pikselit (x_min = -240, x_max = 240) ja kõrguseks 360 pikselit (y_min = -180, y_max = 180).

Tegemist on tingliku ühikuga, mille suurus sõltub ekraani suurusest ja kasutatavast eraldusvõimest.

Andmed

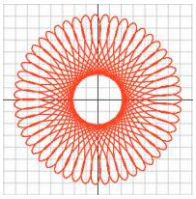
Scratchi rakendustes saab kasutada **graafika**-, **heli**- ja **märkandmeid**. Lisamaterjal: Andmed ja avaldised.

Graafikaandmed

Scratchis võib esineda kahte liiki graafikaandmeid: **graafikaobjektid** ja **graafilised kujutised**.

Graafikaobjektid imporditakse graafikafailidest või luuakse Scratchi joonistamisredaktoriga projekti loomise faasis. Formaalselt on tegemist kas lava taustadega või spraitide kostüümidega. Objektide loomiseks võib kasutada PNG, GIF, JPG, BMP faile.

Lisamaterjal: Scratchi põhiobjektid



Graafilised kujutised luuakse projekti täitmise ajal. Selleks kasutatakse käske gruppidest **Pliiats** ja **Liikumine**. Grupi **Pliiats** käsud võimaldavad määrata pliiatsi omadusi: värv, joone paksus, olek (pliiats üleval või all) jms. Grupi **Liikumine** käskude abil muudetakse joonistava spraidi asukohta.

Scratchis kasutatakse nn **kilpkonnagraafikat**. Meetod ja nimetus võeti kasutusele aastaid tagasi programmeerimissüsteemis **Logo**, mis leiab ka praegu laialdast kasutust, eriti programmeerimise õpetamisel. on kasutusel mitmetes programmeerimissüsteemides nagu Python, BYOB, Basicu mitmed versioonid. Kilpkonnagraafika elemente kasutatakse praktiliselt kõikides programmeerimissüsteemides.

Lisamaterjal: Joonistamine, Scratch wiki.

Heliandmed

Scratchis võib eristada ka kahte liiki heliandmeid: **heliobjektid** ja **hetkhelid** ehk helindid.

Heliobjektid on heliklipid, mis imporditakse või salvestatakse projekti loomise ajal. Saab kasutada WAV ja MP3 vormingus faile.

Heliklippe saab esitada grupi **Heli** käskudega [**mängi heli** *nimi*] ja [**mängi heli** *nimi* **kuni valmis**].

Hetkhelid tekitatakse arvuti poolt vastavate käskude toimel. Scratchis kasutatakse selleks käske [**mängi nooti** ...] ja [**mängi trummi** ...]. Mõlemal juhul saab valida erinevaid instrumente, heli tugevust ja tempot.

Lisamaterjal: Scratchi põhiobjektid, Scratch wiki.

Märkandmed

Scratchis kasutatakse järgmisi märkandmeid: **arvud**, **tekstid** ja **tõeväärtused**.

Arvude jaoks ei ole Scratchis erinevaid vorminguid ja tüüpe, mis on iseloomulikud teistele programmeerimiskeeltele. Tegemist on tavaliste täis- ja reaalarvudega, kusjuures formaalset erinevust nende vahel ei ole. Murdosa eraldamiseks täisosast tuleb reaalarvudes kasutada punkti. Sisendväljas, kus peaks olema arv (näiteks [liigu ... sammu], [oota ... sek]) võib reaalarvu sisestamisel kasutada ka koma, mis muudetakse automaatselt punktiks. Arve võib esitada käsuplokkides ja avaldistes konstantidena ja sisestada klaviatuurilt. Arvude jaoks on neli põhitehet: liitmine, lahutamine, korrutamine, jagamine (**+** , **-** , ***** , /), võrdlustehted, mis esitatakse vastavate plokkide abil, ning teatud valik matemaatilisi funktsioone.

Tekstandmete väärtuseks on Scratchis, nagu ka teistes programmeerimissüsteemides, suvaliste märkide jada. Tekste saab kasutada ainult mõnedes plokkides teadete väljastamiseks, dialoogide korraldamiseks, … Tekst sisestatakse klaviatuurilt konstandina otse plokki, omistatakse muutujale või loendi elemendile. Tekstide jaoks on olemas ka mõned tehted: ühendamine, märkide eraldamine, võrdlemine.

Tõeväärtuste ehk loogikasuuruste jaoks on ainult kaks väärtust ehk loogikakonstanti: tõene (*true*), väär (*false*). Ilmutatud kujul neid Scratchis praktiliselt ei kasutata. Loogikaväärtused tekivad võrdlustes ja loogikaavaldistes, mida kasutatakse tingimuste esitamiseks.

Konstandid ja muutujad

Konstandi väärtus kirjutatakse otse käsuplokki. Programmi täitmisel ei saa konstandi väärtust muuta. Arvus võivad olla ainult numbrid, miinusmärk ja murdosa eraldaja. Tekstkonstante ei paigutata mingite piirajate (jutumärgid või ülakomad) vahele nagu seda peab tegema tekstipõhistes programmeerimiskeeltes.

Muutuja on üheks kesksemaks mõisteks kõikides programmeerimissüsteemides. Selle tähendus programmeerimises ei ole päris sama, mis on muutujal matemaatikas. Programmeerimises mõistetakse muutuja all **nimega varustatud kohta arvuti mälus** (**mälupesa** või mäluvälja), kuhu programm saab salvestada väärtusi. Salvestatud väärtust saab hiljem kasutada näiteks uue väärtuse leidmiseks.

Lisamaterjal: Muutujad Scratchis.

Scratchis luuakse muutujad „käsitsi“ enne programmi täitmist grupis **Muutujad** asuva korraldusega **Tee muutuja**. Muutuja loomisel kuvatakse dialoogiboks, milles saab näidata muutuja nime ja skoobi. Skoop määrab, kas muutuja on kättesaadav kõikidele spraitidele ja lavale või ainult ühele spraidile. Iga muutuja jaoks luuakse automaatselt monitor ehk näidik, mis kuvab laval muutuja jooksva väärtuse arvu või tekstina. Monitori saab ka peita. Monitori näit võib olla tavaline, suur või liuguriga (kerimisribaga), mille abil saab muutuja arvväärtust muuta käsitsi.

Monitoride näited:

tavalised: 

, 

; suured: 

; liuguriga: 

.

Scratchi skriptides saab muutujatele omistada väärtusi **omistamiskäskudega**:

Avaldis määrab väärtuse leidmise eeskirja. Selle erijuhtudeks on ka konstant ja muutuja. Käsu täitmisel leitakse avaldise väärtus ja tulemus võetakse muutuja väärtuseks. Mõned näited:

Enamikes tekstipõhistes programmeerimiskeeltes ja algoritmides esitatakse omistamiskorraldus järgmiselt:

muutuja = ***avaldis***

näiteks: vahe = pikkus – kaal, P = 2 * (a + b)

Operatsioonid märkandmetega. Avaldised

Avaldise abil kirjeldatakse eeskiri vajaliku väärtuse leidmiseks: määratakse tehted ja operatsioonid ning nende täitmise järjekord. Avaldiste koostamiseks kasutatakse vastavaid tehete ja funktsioonide plokke.

Lisamaterjal: Andmed ja avaldised.

Arvavaldised

Arvude jaoks on neli põhitehet, mis esitatakse Scratchis vastavate plokkide abil:

Tehete järjekord määratakse plokkide paigutusega avaldises. Oluline on arvestada, et ühes plokis olev avaldise osa vastab sulgudes olevale avaldisele. Näiteks plokk  vastab avaldisele (a + b).

Arvandmete jaoks on Scratchis teatud valik **matemaatikafunktsioone**, mis esitatakse avaldistes taoliste plokkide abil nagu nt  .

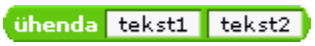
Vasakpoolses väljas saab valida funktsiooni nime: abs, sqrt, sin, cos, tan, asin, acos, atan, ln, log, e^, 10^.

Argumendiks võib üldjuhul olla avaldis, erijuhul muutuja või konstant:

 $\sqrt{a^2+b^2}$  sin(x + 45) – argument kraadides.

Tekstavaldised

Scratchis ei ole eriti palju vahendeid tegevuste täitmiseks tekstidega.

Tekstide ühendamiseks saab kasutada plokki 

Mõlemas väljas võivad olla omakorda ühendamise plokid.

Plokk  võimaldab leida antud teksti pikkuse, st märkide arvu tekstis.

Plokk  tagastab antud numbriga (nr) sümboli.

Protsesside juhtimine

Scratchis kujutab protsessi kirjeldus käsuplokkide gruppi, milleks võib olla terve skript või osa sellest. Sõltumata kasutatavatest vahenditest võib ülesannete lahendamisel eristada nelja liiki protsesse:

- järjestikune protsess ehk jada,
- tsükliline protsess ehk kordus,
- hargnev protsess ehk valik,
- paralleelsed protsessid.

Lisamaterjal: Protsesside juhtimine, Scratch wiki.

Järjestikune protsess ehk jada

Järjestikuse protsessi korral täidetakse kõik käsud täpselt sellises järjekorras nagu need on esitatud. Käske ei jäeta vahele ega toimu ka kordamisi. Mingeid erilisi vahendeid protsessi määratlemisel ei ole vaja, sest täitmine toimub nö loomulikus järjekorras.

Toodud näide kujutab järjestikust protsessi. Programm küsib ja loeb ristküliku külgede pikkused **a** ja **b**, arvutab selle pindala **S** ja ümbermõõdu **P** ning leiab pindala ja ümbermõõdu suhte – **suhe**.

NB! käskude järjekorra valimisel peab arvestama, et neid täidetakse täpselt selles järjekorras, nagu need on esitatud programmis. Algandmete lugemise käsud peavad eelnema pindala ja ümbermõõdu arvutamisele ning viimased omakorda suhte leidmisele. Samal ajal ei oma tähtsust **a** ja **b** lugemise omavaheline järjekord ning pindala **S** ja ümbermõõdu **P** arvutamise järjekord.

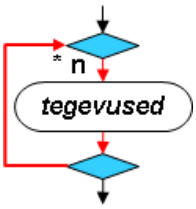
Tsüklilised protsessid

Tsükliline protsess seisneb tegevuste (käskude) korduvas täitmises. Programmis tuleb määrata reeglid ja tingimused protsessi täitmiseks. Scratchis on korduste kirjeldamiseks mitu käsuplokki, mis võimaldavad arvestada erinevat tüüpi protsessidega.

Lisamaterjal: Algoritmimine.

Etteantud korduste arvuga kordus

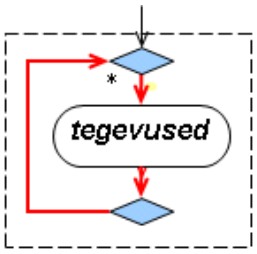
See esitatakse Scratchis plokiga [**korda** *n*], kus ***n*** on kordamiste arv. Viimane võib olla esitatud konstandi, muutuja või avaldise abil. Allpool on toodud selle kordusetüübi esitus ka UML diagrammide ja algoritmikeele (pseudokoodi) abil.



kordus *n* **korda**
tegevused
lõpp kordus

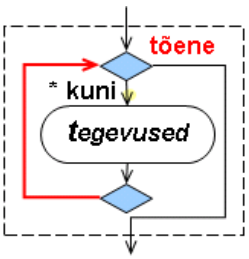
Lõputu kordus

Formaalselt täidetakse ploki sees olevaid käske lõputult, kuid praktiliselt katkestatakse kordus ühel või teisel viisil. Korratavate käskude seas võib olla näiteks käsk [**peata skript**], mis täidetakse teatud tingimuse täitumisel ja see lõpetab lõputut kordust sisaldava skripti töö. Korduse võib katkestada käsk [**peata kõik**], mis asub samas või mõnes teises skriptis. Alati saab korduse katkestada punase nupuga, mis lõpetab terve rakenduse töö.

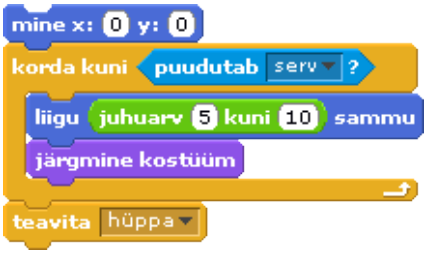


kordus
tegevused
lõpp kordus

Eelkontrolliga tingimuslik kordus



kordus kuni *tingimus*
tegevused
lõpp kordus



Selle korduseliigi korral **korratakse** tegevusi seni, **kuni** antud tingimus **saab tõeseks**.

Tingimuste esitamiseks kasutatakse loogikaavaldisi, mis esitatakse võrdluste ja loogikatehete plokkide abil:

Loogikaavaldise tulemuseks saab olla ainult tõeväärtus: tõene (*true*) või väär (*false*).

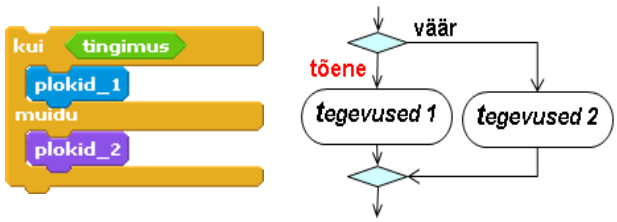
Hargnevad protsessid

Hargnevas protsessis valitakse üks mitmest võimalikust tegevusest, sõltuvalt antud tingimustest.

Vt ka Algoritmimine.

Scratchis on kaks valikuplokki: valik kahest ja valik ühest. Esimesel juhul täidetakse kahest võimalikust plokkide grupist üks. Teisel juhul, sõltuvalt tingimuse väärtusest, antud plokid kas täidetakse või jäetakse vahele. Iga plokk võib omakorda sisaldada valikuplokke. Tingimuste esitamiseks kasutatakse loogikaavaldisi, mis esitatakse võrdluste ja loogikatehete plokkide abil:

Loogigaavaldise väärtuseks saab olla ainult tõeväärtus: tõene (*true*) või väär (*false*).



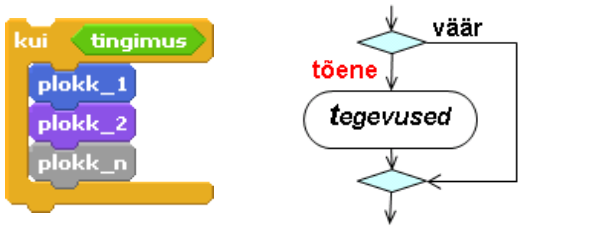
kui *tingimus* **siis**

tegevused 1

muidu

tegevused 2

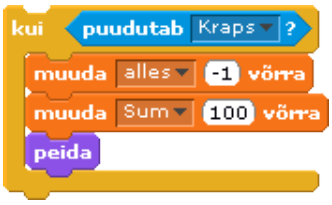
lõpp kui



kui *tingimus* **siis**

tegevused

lõpp kui



Paralleelsed protsessid

Kasutaja saab korraga käivitada mitu skripti, mille päises on plokk [**kui klõpsatakse** ...] või [**kui vajutatakse klahvi** ...]. Lisaks juba töötavatele skriptidele võib kasutaja käivitada (kui rakenduses on ette nähtud) täiendavaid skripte, mis alustavad tööd paralleelselt eelmistega.

Skriptidest saab paralleelseid protsesse käivitada käsuplokiga [**teavita** *nimi*] ja [**teavita** *nimi* **ja oota**]. Korraga hakkavad tööle kõik skriptid, mille päises on plokk [**kui saabub teade** *nimi*], kus *nimi* on sama nagu käsus teavita. Tegemist võib olla sama spraidi skriptidega või teise spraidi omadega.

Sündmused

Süsteemis on ette nähtud erinevate sündmuste kontroll ja reaktsioon nendele. Sündmusi saab tekitada kasutaja hiire ja klahvidega ning skriptid teadete saatmisega. Üheks enim esinevaks reaktsiooniks sündmustele on skriptide käivitamine, mis on seotud päiseplokkidega. Scratchi põhiobjektid

[**kui klõpsatakse** (rohelist lippu)]

[**kui klõpsatakse** *spraiti*]

[**kui klõpsatakse** *lava*]

[**kui vajutatakse** *klahvi*]

[**kui saabub** *teade*]

Sündmuste kontrolli ja reaktsioone neile saab skriptides teostada andurite ja vastavate tingimuste kasutamisega:

<puudutab objekti>

<hiir all?>

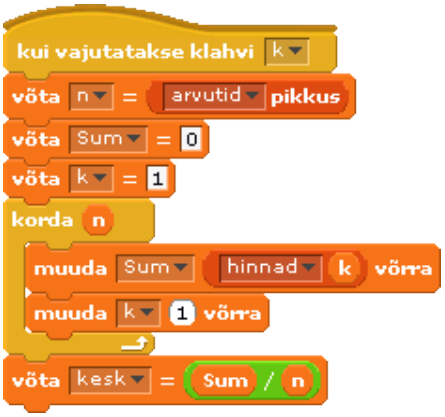
<klahv ... all?>

(taimer)

Loendid

Loend (*List*) on järjestatud mäluväljade kogum, mis tähistatakse ühe nimega. Loendi elementidele viidatakse nime ja indeksi abil: arvutid 1, hinnad k.

Käsuga **Tee loend** luuakse tühi loend ja laval tekib loendi monitor. Loendi elemente saab lisada, muuta ja eemalda nö "käsitsi", importida tekstifailist ja eksportida tekstifaili. Elemente saab lisada, eemaldada ja muuta vastavate käskude abil skriptides.



Selliseid andmekogumeid nimetatakse sageli ka ühemõõtmelisteks massiivideks.

Näites on toodud kahe loendi monitorid ja skript, mis leiab loendi **hinnad** elementide aritmeetilise keskmise, mis omistatakse muutujale **kesk**. Skript teeb kindlaks elementide arvu loendites (loendite pikkused) ja omistab selle muutujale **n**. Plokis [**korda** n] liidetakse järjest loendi **hinnad** elemendid (**hinnad** järjenumbriga **k**) muutujale **Sum**, saades hindade summa. Selle jagamisel muutuja **n** väärtusega saadakse keskmine.

Lisamaterjalid: Loendid 1, Loendid 2, Scratch wiki.

Scratchi rakenduste kasutamisviisid ja projektide vormingud

Scratchi rakendusi saab kasutada (täita) erineval viisil.

Keskkond võib asuda kliendi oma arvutis või serveris. Scratchi projekti fail on laiendiga **sb**. Programmi täitmist korraldab **interpretaator**, mis transleerib (tõlgib) skriptid ühekaupa masinkeelde ja suunab koodi täitmiseks protsessorisse. Näide: hunt kits kapsas.sb. Fail laetakse serverilt ning automaatselt avatakse Scratchis.

Scratchi projekti saab siduda html-dokumendiga märgendiga <applet…>. Süsteem loob märgendi alusel projektist Java apleti ja paigutab selle html-dokumenti (vt link). Viimase laadimisel brauserisse aktiveeritakse aplet. Näide: hunt kits kapsas.html.

Projekti täitmine „pilves“ – Scratchi kodulehel *MIT* serveris. Scratchi projekti saab serverisse üles laadida menüükäsuga „jaga projekti internetis“. Käsuga „mine Scratchi kodulehele“ saab minna Scratchi avalehele (scratch.mit.edu). Kui kasutaja on loonud serveris konto, kuvatakse tema projektide loetelu, kust saab

valida täitmiseks sobiva projekti. See esitatakse Java apletina või Flashi klipina. Kasutaja võib lisada mingisse dokumenti lingi serveris asuvale (oma või võõra) projektile.

Näiteks http://scratch.mit.edu/projects/vilip/2828630. Aadressi algus on standardne, lõpus on kasutajanimi (vilip) ja projekti järjenumber (2828630), mille süsteem fikseerib automaatselt üleslaadimise hetkel. Aadressi saab näiteks kopeerida, kui projekt muudetakse serveris aktiivseks.

EXE-faili kasutamine (ainult MS Windows). Tegemist on masinkeelse programmiga, mille täitmiseks ei ole vaja Scratchi ega muid vahendeid. Scratch ise praegu exe-faile ei tee, kuid selleks saab kasutada BYOBi. Viimane võimaldab avada ja täita Scratchi faile ning kompileerida need exe-failideks.

Näide: hunt kits kapsas.exe.

Lähitulevikus peaks saama võimalikuks Scratchi rakenduste loomine ja täitmine otse brauseris.

Näide: Tutvus

Rakendus demonstreerib mitmeid Scratchi võimalusi ja vahendeid ning programmeerimise põhilisi tegevusi nagu graafika- ja heliobjektide kasutamine, märkandmete väljastamine ja sisestamine, joonistamine, korduste ja valikute kirjeldamine, muutujate, loendite ja juhuslike arvude kasutamine, lihtsate tekst- ja arvavaldiste kasutamine, tegevuste jagamine ning koostöö korraldamine skriptide vahel.

Demo: tutvus.sb, tutvus.html, tutvus.java, tutvus.exe

Ülevaade rakendusest

Stsenaarium*: Laval on tegelased: Kraps, Mari, Juku, robot Robi ja pall. Programmi käivitamisel Kraps teretab, esitleb end, teeb salto ja küsib kasutaja nime. Kasutaja võib sisestada nime või loobuda sellest. Viimasel juhul muudab Kraps pahameelest oma värvi, väljendab kahetsust, kustutab eelmised andmed, peidab ennast ära ning lõpetab programmi töö. Kui aga kasutaja sisestab nime (põhimõtteliselt suvaline tekst: vähemalt üks märk), kiidab Kraps kasutajat ning tegelased esitlevad end, tehes seejuures mõned väikesed trikid: hüpped, saltod, tantsud jm. Seejärel pakub Juku kasutajale teenust: saada teada oma „saledus“. Kui kasutaja soovib seda, küsib Juku tema käest pikkuse ja kaalu (massi), leiab kehamassiindeksi (pikkus/kaal^2) ja selle alusel „saleduse“ ning laseb abispraidil joonistada andmete ümber raami. Kui kasutaja ei soovi teada saledust, eemaldab abisprait muutujate monitorid ja raami. Enne töö lõppu jätavad tegelased hüvasti, kuvades vastavad teated ja tehes veel mõned trikid. Vahepeal poisid veidi togivad ja lennutavad palli.*

Lava kujundus (taust) ei oma erilist tähtsust, see võib ka puududa. Praegu on laval mitu tausta, millest on nähtav üks. Hiireklõpsuga laval saab neid vahetada. Laval on ka muutujate monitorid, milles kuvatakse kasutaja **nimi**, **pikkus**, **kaal**, **indeks** ja **saledus**. Laval on oma skriptid, üks neist töötab pidevalt paralleelselt teiste skriptidega ja muudab lava tausta värve.

Rakenduses on neli põhispraiti: **Kraps**, **Mari**, **Juku** ja **Robi** ning üks abisprait, mis joonistab ja haldab muutujate monitore (kuvab ja peidab). Konkreetsed spraitide kujud ja nimed ei oma tähtsust. **Kraps** on oma kahe kostüümiga kohe laval olemas. **Mari** ja selle kostüümid on imporditud spraitide hoidlast. Samuti on hoidlast imporditud **Robi**, millele on kopeerimise ja redigeerimisega lisatud üks kostüüm. **Juku** on imporditud GIF-failist. Kui aga lugeja realiseerib ise antud rakenduse, võib ta valida mõned teised spraidid.

Rakenduses on mitu skripti. Pildil on toodud üks Krapsu skriptidest, mida võib nimetada ka **peaskriptiks**. Sellest algab programmi töö ja siit käivitatakse teised skriptid. Peaskript alustab tööd, kui kõpsatakse rohelist lippu, samaaegselt käivitub ka lava skript, mis muudab lava taustavärve ja töötab kogu aeg paralleelset teiste skriptidega.

Edasi vaatame programmi tööd, järgides peaskripti, mille käske täidetakse järjest. Alguses on mitu lihtkäsku, kuid põhiosa skriptist moodustab liitkäsk [**kui** *tingimus…*], mis sisaldab omakorda mitut lihtkäsku. Sõltuvalt sellest, kas kasutaja sisestas oma nime või mitte, täidetakse ainult üks käsugrupp. Skriptis on mitu käsku [**teavita** *teade*…], mis käivitavad ühe või mitu alamskripti ja peatavad vahepeal peaskripti täitmise.

Siin ja edaspidi, viitamaks käskudele tekstis, paigutame need märkide **[ ]** vahele milles mõnikord on näidatud käsu kõik elemendid, teinekord ainult käsu algus või nimi.

Sissejuhatavad tegevused

Kõigepealt teeb käsk [**näita**] Krapsu nähtavaks, juhul kui ta oli peidus. See on vajalik programmi käivitamisel olukorras, kus kasutaja loobus eelmisel käivitamisel nime sisestamisest ja Kraps oli peidus. Käsk [**mängi heli** *näu*] käivitab heliklipi „näu“, mis on Krapsul kohe olemas. Vajadusel saab heliklippe importida infoalas valitavalt vahelehelt **Helid**. Käsk [**ütle** …] väljastab Krapsu tervituse, kuvades antud teksti jutumullis Krapsu juures.

Käsk [**teavita** *hei_hop* **ja oota**] saadab süsteemile teate, käivitamaks skripte, mille alguses on plokk [**kui saabub teade** *hei_hop*]. Praegu on Krapsul üks selline skript, mis paneb ta tegema saltot. Alamskripti käskude täitmise järel jätkub peaskripti töö.

Andmete sisestamise käsk [**küsi** *tekst* **ja oota**] peatab skripti täitmise, kuvab *teksti* spraidi juures olevas jutumullis ning kuvab lava alumises servas tekstivälja, kuhu sisestada väärtus. Kui kasutaja tipib selle ning vajutab klahvi **Enter**, võetakse antud väärtus sisemuutuja ***vastus*** väärtuseks ja skripti töö jätkub. Kui kasutaja vajutab kohe klahvile Enter, on tegemist nn tühja väärtusega. Käsuga [**võta** *nimi* = *vastus*] omistatakse muutujas *vastus* väärtus muutujale *nimi*.

Järgneb liitkäsk [**kui** *tingimus* (käsud1) **muidu** (käsud2)], mille abil saab kirjeldada **valikuid** ehk hargnevaid protsesse. Praegu on tegemist **kahendvalikuga** – kahest alternatiivsest võimalusest valitakse üks, olenevalt võrdluse abil esitatud tingimuse väärtusest. Kui tingimus on **tõene**, täidetakse käsud1 ning käsud2 jäetakse vahele. Vastupidisel juhul täidetakse käsud2 ning käsud1 jäetakse vahele. Antud juhul tehakse võrdluse abil kindlaks, kas muutujal *nimi* on väärtus või mitte.

Kasutaja loobus jätkamisest

Kui tingimus on tõene (muutuja *nimi* väärtus on tühi), täidetakse **kui**-harus olevad käsud. Käsk [**teavita** puhasta] käivitab spraidi **abi** vastava skripti, mis peidab muutujate **pikkus**, **kaal**, **indeks** ja **saledus** monitorid ning kustutab raami. Käsk [**pane** *värv* **efekt** 100-le] muudab spraidi värvi, kasutades värvikoodi (väärtused on vahemikus 0 kuni 200). Praegu kasutatakse sinist värvi. Käsk [**ütle** Kahju!…] kuvab teate ja järgnev käsk [**peida**], mis peidab Krapsu ära. Kuna **muidu**-haru käsud jäetakse vahele, siis järgnevalt täidetakse käsk [**peata kõik**], mis lõpetab kogu rakenduse töö.

Kasutaja jätkab

Käsu [**kui**…] teise haru esimene käsk [**ütle**…] on juba tuttav. Võrreldes eelmiste seda tüüpi käskudega, on siin kasutusel **tekstavaldis**, mis on moodustatud plokiga 

ning kus ühendatakse tekstkonstant „Tubli, “ ja muutuja **nimi** väärtus.

Tekstide ühendamiseks tekstipõhistes süsteemides kasutatakse märki **+** või **&**. Näiteks Pythonis ja Visual Basicus pannakse selline avaldis kirja nii: „Tubli, “ + nimi, VBs ka „Tubli, “ & nimi.

Käsk [**teavita** alustame] **ja selle alusel käivitatud tegevused**

Järgmine käsutüüp [**teavita** alustame **ja oota**] esines juba ka eespool, kui käivitus üks Krapsu skript. Praegu on tegemist üldisema juhuga, mil käivitub korraga neli skripti, sest päisega [**kui saabub teade** alustame] algav skript on praegu igal spraidil. Skriptid töötavad paralleelselt. Kuna käsus **teavita** on täiend **oota**, siis peatatakse peaskripti täitmine, kuni lõpeb kõikide alamskriptide töö, ning peale seda jätkub antud skripti täitmine. Vaatleme nüüd nendes protsessides osalevate spraitide vastavaid skripte ja tegevusi, mis on üles ehitatud suhteliselt sarnaselt.

Alustame. Juku ja Robi skriptid

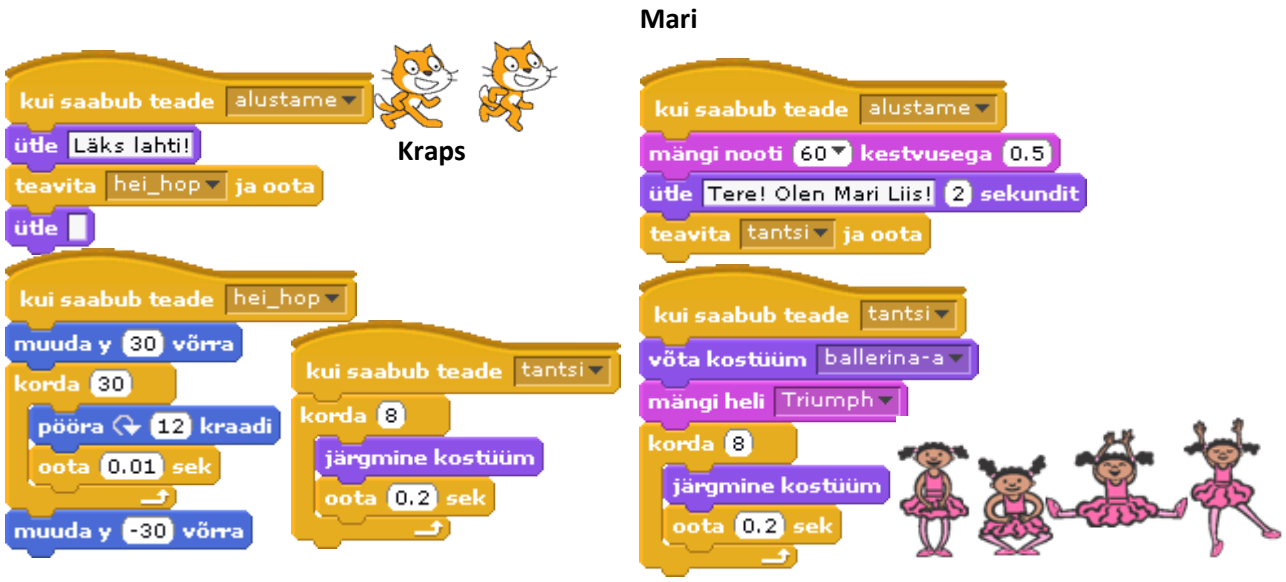
Juku skript **alustame** tekitab käsuga [**oota** 2 **sek**] kõigepealt väikese pausi, et alustada tegevusi teistest veidi hiljem. Seejärel Juku esitleb end ja saadab teate „hüppa“, mis on mõeldud tema enda skriptile – see paneb Juku hüppama. Skriptis on plokiga [**korda** 3] määratud etteantud korduste arvuga kordus, st et ploki sees olevaid käske täidetakse kolm korda. Iga kord suurendatakse Juku y-koordinaati 40 pikseli võrra, tehakse väike paus, vähendatakse koordinaati 40 pikseli võrra ja tehakse jälle paus. Tulemusena liigub Juku kolm korda üles ja alla, imiteerides hüppamist. Käsk [**teavita** löök **ja oota**] käivitab palli skripti, mis viib palli Juku juurest Krapsu juurde ja sealt tagasi. Vahepeal lendab pall ka üles.

Juku

Robi

Robi skriptid on analoogsed. Käsuga [**mängi heli** robi] käivitatakse salvestatud heliklipp. Salvestada saab vahelehel **Helid** korraldusega **Lindista**, kui arvutil on mikrofon. Kui see puudub, võib võtta süsteemi heliklippide hoidlast suvalise klipi. Midagi ei juhtu, kui klippi üldse ei ole – heli lihtsalt ei mängita. Skript **liigu** paneb **Robi** kostüüme vahetama (visuaalsed efektid) ning horisontaalselt liikuma, suurendades ja vähendades x-koordinaati. Ka siin on plokk **korda**, mis tagab sisemiste käskude kordamise kaks korda. Robi teine kostüüm on saadud esimesest kasutades Scratchi joonistamisredaktorit.

Alustame. Krapsu ja Mari skriptid



Muusika ja tantsulembese **Mari** skriptides demonstreeritakse heliandmete kasutamist ja tantsu imiteerimist kostüümide vahetamise teel. Skriptis **alustamine** mängitakse lihtsalt üks noot, kasutades Scratchis olevat lihtsat helide süntesaatorit. Käsus [**mängi nooti** *number* **kestvusega** *aeg*] saab näidata noodi numbri (48 kuni 72) ja kestvuse sekundites. Kes tunneb noote, võib proovida ka komponeerida meloodiat. Plokiga [**mängi trummi** …] saab imiteerida ka löökpillide helisid.

Skriptis **tantsi** võetakse Mari jaoks esimene kostüüm ballerina-a (igal kostüümil on nimi). Seda tehakse igaks juhuks, arvestades et programmi eelneval katkestamisel võib Mari jääda suvalisse olekusse. Käsuga [**mängi heli** *Triumph*] käivitatakse heliklipp ning skripti töö jätkub muusika saatel. On olemas ka käsk [**mängi heli** *nimi* **kuni valmis**]. Selle käsu korral skripti töö peatatakse, mängitakse heliklipp lõpuni ning seejärel jätkatakse tööd.

Tantsu simuleerimiseks kasutatakse siin plokki [**korda** 8]. Igal kordamisel vahetatakse kostüümi ja tehakse väike paus. Kuna kordamisi on praegu 8, käiakse kostüümid kaks korda läbi. Kui jõutakse viimase kostüümini, alustatakse uuesti esimesest.

Skript **tantsi**, ilma kostüümi valimiseta ja heliklippi käivitamiseta, on ka **Krapsul**. Kui Mari skript **alustame** saadab teate **tantsi**, hakkab tööle ka Krapsu samanimeline skript.

Krapsu skript **hei_hop** paneb teda tegema saltot. Kraps liigub üles 30 pikselit, teeb ühe tiiru ja liigub tagasi alla. Paneme tähele, et 360-kraadine pööre ei ole määratud ühe käsuga: [**pööra** 360 **kraadi**], vaid 30 korda täidetava käsuga [**pööra** 12 **kraadi**] (30 * 12 = 360), et pööre oleks nähtav (jälgitav). Kui kasutada esimest varianti, siis pööret näha ei oleks. Realiseeritud variandi korral, kus igale pöördenurga muutusele (12 kraadi) järgneb ka väike paus, jaguneb liikumine kaadriteks. Taoline põhimõte on animatsioonides üldkasutatav. Liikumise kiirust ja sujuvust saab reguleerida vaadeldaval juhul nurga muutusega ja pausi pikkusega. Paneme tähele, et pause kasutatakse ka teistes animatsioonidega seotud skriptides: **hüppa** (Juku), **tantsi** (Mari ja Kraps), **liigu** (Robi).

Skriptides **hüppa**, **liigu**, **tantsi** ja **hei_hop** kasutatakse etteantud korduste arvuga kordust, mis on üks lihtsamaid ja sagedamini kasutatav korduse tüüp kõikides programmeerimiskeeltes. Palli skriptides demonstreeritakse tingimuslike korduste kasutamist. Lava skriptis esineb ka lõputu kordus.

Palli skriptid

Kaheks peamiseks palli skriptiks on **löök** ja **lenda**. Esimene viib palli pööreldes vasakule Krapsuni ja tagasi Juku juurde. Teine viib palli üles lava servani ja alla Juku juurde tagasi. Mõlemas skriptis kasutatakse kaks korda tingimuslikku kordust. Esimeses skriptis on tingimus määratud nö puutesündmustega – puudutab Kraps, puudutab Juku. Teises on tingimus esitatud lava serva puudutamisega (üles liikudes) ja jõudmisega antud kõrguseni (alla liikudes). Pallil on ka mõned klahvisündmustele reageerivad skriptid: [**kui vajutatakse klahvi** vasak nool], [kui **vajutatakse klahvi** ülesnool] ja [**kui klõpsatakse** pall], millest on näidatud ainult esimene.

Saleduse leidmine

Käsk [**teavita** *saledus* **ja oota**] käivitab rakenduse teise osa, kus põhitegelaseks on Juku. Skriptis **saledus** küsitakse, kas kasutaja soovib teada oma saledust. Kui kasutaja sisestab tähe **e**, peidetakse muutujate **pikkus**, **kaal**, **indeks** ja **saledus** monitorid ja skripti töö peatakse. See tähendab, et täitmisjärg läheb tagasi peaskriptile. Kui aga sisestatakse midagi muud, käivitakse järjest käsud [**teavita** ...].

Juku skript **pöörle** paneb Juku tegema saltot. Põhimõtelist on tegemist sama asjaga nagu Krapsu skriptis **hei_hop** ning skripti pole siin näidatud.

Saleduse leidmisel on kasutusel neli põhimuutujat: **pikkus**, **kaal**, **indeks** ja **saledus**. Esimese kahe väärtused sisestab kasutaja, teised leiab skript **hinnang**.

Algandmete sisestamist korraldab skript **loe_alg**. Kuid muutujate (pikkus ja kaal) väärtusi saab muuta ka liugurite abil. Skriptis **loe_alg** on ette nähtud võimalus, et olemasolevaid muutujate väärtusi ei muudeta. Kui käsu **küsi** täitmisel väärtust ei sisestata, vaid vajutatakse kohe klahvile Enter, jääb muutuja väärtus endiseks. Väärtust muudetakse ainult siis, kui tegemist ei olnud tühja väärtusega.

Saleduse leidmisel leitakse kõigepealt kehamassi indeks (kaal/pikkus^2) ja omistatakse see muutujale **indeks**. Kuna pikkus sisestatakse sentimeetrites, kehamassi indeksi leidmisel aga peab see olema meetrites, jagatakse muutuja **pikkus** väärtus 100-ga. Skriptis kasutatakse abimuutujat **v**, millele omistatakse pikkuse väärtus meetrites, et järgnev käsk oleks lühem.

Kehamassiindeks k_{ind} leiab sageli kasutamist inimese saleduse või täidluse hindamisel. Tavaliselt kasutatakse taolist skaalat:

$$k_{ind} < 18 - \text{kõhn}; 18 = k_{ind} < 25 - \text{normaalne}; 25 = k_{ind} \le 30 - \text{ülekaal}; \; k_{ind} > 30 - \text{suur ülekaal}$$

Siin kasutavas skriptis ei arvestata viimast kriteeriumi ($k_{ind} > 30$ – suur ülekaal). Lugeja võiks proovida see lisada.

Muutujate monitorid on sageli olulisteks kasutajaliidese elementideks. Antud rakenduses on ette nähtud monitoride peitmine ja kuvamine vastavalt olukorrale. Seda teevad spraidi **abi** skriptid: **puhasta** ja **loe_alg**, vastavate teadete saamisel.

Skript **puhasta** kustutab ka raami, mis on joonistatud monitoride ümber.

Joonistamine

Joonistada võib suvaline sprait. Iga spraidiga on alati seotud **pliiats**, mis võib olla üleval – spraidi liikumisel joont ei teki, või all – liikumisel tekib lavale joon. Joonistamisel kasutatakse peamiselt käske gruppidest **Pliiats** ja **Liikumine**.

Pliiatsi käskudega saab määrata joone värvi, varjundi ja paksuse ning muuta pliiatsi olekut, kas üleval või all. Praegu valitakse pliiatsi värv ja suurus juhuslikult.

Liikumise käskudega määratakse joonte asukoht laval. Käsk [**mine** x: … y: …] viib spraidi kohe antud punkti. Käsuga [**liigu** t sek. x: … y: …] saab määrata spraidi sujuva liikumise. Ühikuks on piksel.

Hüvastijätt

Selles tegevuses osalevad kõik tegelased, kusjuures tegu on paralleelsete protsessidega. Siin demonstreeritakse ka lihtsamaid võimalusi loendite kasutamiseks.

Loendis nimega **T** on tekstid, millest valitakse iga spraidi jaoks juhuslik väärtus, mis kuvatakse käsus **ütle**. Kõikide spraitide skriptides päisega [**kui saabub teade** *hüvasti*] on käsk

Loendi **T** elemendi indeksiks võetakse juhuslik arv vahemikust 1 kuni **m**, kus **m** on elementide arv loendis ning see leitakse spraidi **abi** skriptis.

Lava skriptid

Laval on kaks skripti. Üks käivitub koos Krapsu põhiskriptiga, kui klõpsatakse rohelist lippu, muutes lava värvi.

Formaalselt on tegemist lõputu kordusega, praktiliselt aga lõpetab skripti töö Krapsu põhiskripti käsk [**peata kõik**]. Alati saab kogu rakenduse töö katkestada ka lava kohal asuva punase nupuga.

Värvi muudetakse sagedusega üks kord sekundis. Värvi kood valitakse juhuslikult.

Teine skript, mis käivitub hiireklõpsuga laval, koosneb ühest käsust: 

Andmed

Järgnevalt vaadeldakse objekte, põhimuutujaid ja rakenduse struktuuri.

Objektid (**spraidid**)
Kraps – juhib ja korraldab rakenduse tööd ja teeb igasuguseid trikke (kaks kostüümi, seitse skripti)

Mari – kultuuritöötaja: tantsib ja teeb muusikat (neli kostüümi, kolm skripti)

Juku – sportlane ja matemaatik, arvutab ja hindab saledusi (üks kostüüm, seitse skripti)

Robi – robot, tutvub inimestega, õpib arvutama faktoriaale (kaks kostüümi, kuus skripti)

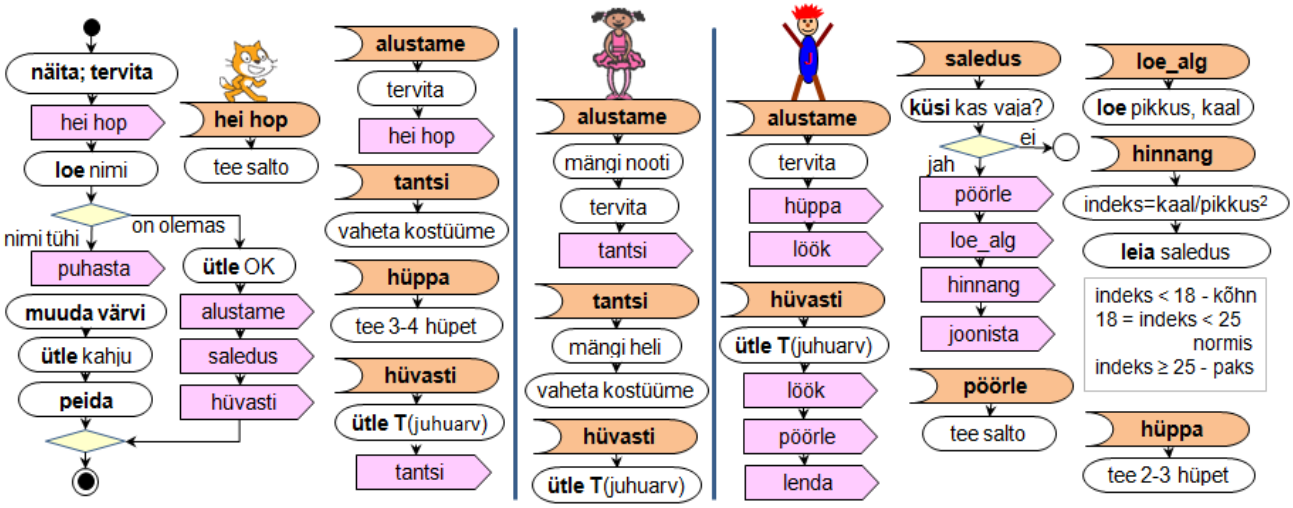
pall – teda togivad ja pilluvad **Juku** ja **Kraps** (üks kostüüm, viis skripti)

abi (peidetud) – joonistab ja teeb muid abitöid (üks kostüüm, neli skripti)

Põhimuutujad: nimi, pikkus, indeks, saledus

Rakenduse struktuur

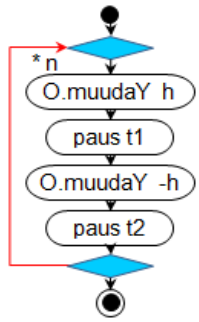
On näidatud üldvaade kolme spraidi skriptidest. Koostööd peegeldavad teavitamise korraldused ja vastuvõtvate skriptide päised. Vt Rakendused.



Algoritmid

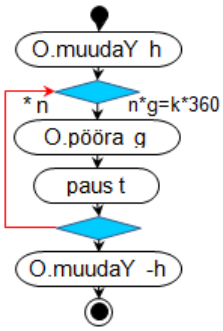
Näiteks on toodud mõned algoritmid UML tegevusdiagrammidena ja pseudokoodis. Vt Algoritmimine.

hüppa



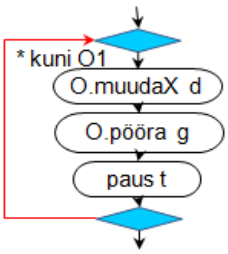

```
prots hüppa
kordus n korda
    objekt.muudaY h
    paus t1
    objekt.muudaY -h
    paus t2
```

salto



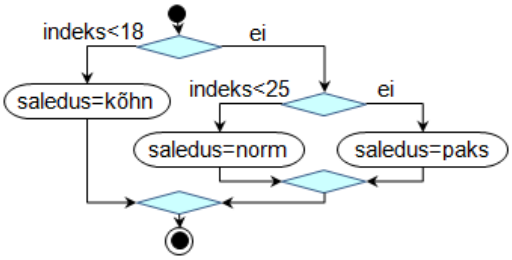

```
prots salto
objekt.muudaY h
kordus n korda
    objekt.pööra g
    paus t
lõpp kordus
objekt.muudaY -h
```

löök




```
prots löök
kordus kuni objekt_1
    objekt.muudaY h
    objekt.pööra g
    paus t
```

hinnang




```
prots hinnang
kui indeks<18 siis
   saledus = kõhn
muidu
   kui indeks<25 siis
      saledus = norm
   muidu
      saledus = paks
```

Rakenduste disainist ja dokumenteerimisest

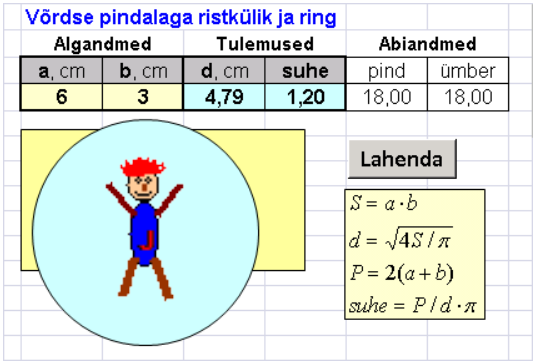
Suuremate rakenduste korral on otstarbekas enne skriptide koostamist täpsustada ülesande püstitust, viia läbi rakenduse analüüs ja disain (projekteerimine) ning koostada projekti dokumentatsioon:

- määratleda andmed,
- fikseerida spraitide funktsioonid,
- seosed nende vahel ja tegevused,
- koostada algoritmid,
- kavandada kasutajaliides.

Rakenduse realiseerimise käigus dokumentatsioon enamasti täiustub ning viimasena vormistatakse selle lõplik variant, mida läheb vaja rakenduse kasutamisel ja arendamisel.

Lisamaterjal: Rakendused.

Sissejuhatus VBAsse



VBA – *Visual Basic for Application*, mis on kavandatud dokumendipõhiste rakenduste loomiseks ja arendamiseks, on sarnane üldotstarbelisele programmeerimissüsteemile Visual Basic (VB). Antud materjalis vaadeldakse VBA kasutamist Exceli keskkonnas, arvestades, et lugeja oskab juba mingil määral programmeerida ja on tuttav Scratchiga. Eelteadmised Excelist ei ole vajalikud.

Sissejuhatus VBAsse

Rakenduste arendussüsteem **VBA** – ***V**isual **B**asic for **A**pplications*, põhineb üldotstarbelisel programmeerimissüsteemil Visual Basic (**VB**), mis leiab laialdast kasutamist eriti rakendustarkvara ja veebirakenduste loomisel. Kasutusulatuselt on ta umbes viiendal–kuuendal kohal. Temast eespool on vaid Java ja C-pere keeled (C, C++, C#), mis on eeskätt süsteemprogrammeerimise keeled. Enam-vähem samas ulatuses on PHP ja Pythoni kasutamine. Basicu variante ja dialekte on üsna palju. Neist nn tööstusliku tarkvara loomiseks leiab aga kasutamist peamiselt Visual Basic ja selle alusel loodud VBA.

VBA esindab **dokumendipõhiste rakenduste** loomise ja arendamise vahendit. Kui pidada silmas programmeerimiskeelt, siis on see VB-s ja VBA-s praktiliselt sama. Kõik põhilaused, andmete käsitlus, andmete liigid ja tüübid on mõlemas süsteemis ühesugused. Erinevus seisneb peamiselt selles, et VBA on mõeldud kasutamiseks koos mõne rakendustarkvaraga ning selle abil loodud rakendused saavad töötada ainult vastava programmi keskkonnas. Visual Basicu näol on aga tegemist üldotstarbelise programmeerimissüsteemiga ning selle abil luuakse autonoomseid rakendusi. Samal ajal saab VB-d kasutada ka dokumendipõhiste rakenduste loomisel.

VBA peamised rakendusvaldkonnad on üldotstarbelised rakendusprogrammid: tabeli- ja tekstitöötluse programmid, esitlus- ja graafikasüsteemid, andmebaasirakendused. VBA leiab laialdast kasutust eeskätt Microsofti toodetes – Excel, Word, PowerPoint, Access, Visio, Project. VBA on kasutusel ka sellistes süsteemides nagu Corel Office, Corel Draw, AutoCAD, Imagineer jm. VBAga lähedased arendusvahendid on ka süsteemides OpenOffice ja IBM SmartSuite ja paljudes teistes.

Siin vaadeldakse VBA kasutamist MS Exceli keskkonnas. Tahaks rõhutada, et eesmärgiks ei ole Exceli programmeerimine, vaid Scratchis omandatud programmeerimise oskuste ja teadmiste süvendamine ja laiendamine tekstipõhiste, objektorienteeritud programmeerimist toetavate vahendite abil. VBA ja Excel pakuvad selleks suurepäraseid võimalusi. Olgu märgitud, et VBA võeti esmakordselt kasutusele just Excelis (1995).

Võrreldes enamike teiste programmeerimiskeeltega on VBAl (võiks öelda Visual Basicul) oluliselt lihtsam struktuur ja süntaks ning samal ajal on siin rikkalik valik objektorienteeritud vahendeid töötamiseks graafikaobjektide, tabelite ja massiividega. VBA lausete ja Scratchi käsuplokkide vahel on suur sarnasus, eriti protsesside juhtimise ja graafikaandmete kasutamise osas. Lihtsuses saab Visual Basicuga võistelda ehk ainult Python. Kuid Pythonis on mõnevõrra keerulisem ja töömahukam kasutajaliideste loomine, eriti kui kasutatakse tabeleid, massiive ja graafikaobjekte.

Exceli töölehed ja töövihikud pakuvad lihtsaid ja mugavaid vahendeid kasutajaliideste loomiseks ja andmete, sh graafikaobjektide, salvestamiseks ja säilitamiseks. Exceli töölehte võib teatud määral võrrelda Scratchi lavaga. Kuid võrreldes viimasega on töölehel praktiliselt piiramatud mõõtmed ning kindlad mõõtühikud, mis võimaldavad teha ka mõõtkavas joonistusi, kasutades ühikutena näiteks sentimeetreid. Töölehe lahtrid aga võimaldavad salvestada suuri andmehulki üksikväärtustena, tabelitena ja massiividena. Kusjuures kõik andmete paigutamisega ja vormindamisega seotud küsimused saab lihtsalt lahendada Exceli vahenditega, mis vähendab oluliselt vajadust tegeleda sellega loodavas programmis.

Töötamiseks VBAga Exceli keskkonnas siin pakutavas lähenemisviisis ei pea eriti palju tundma Excelit (kuigi halba see muidugi ei tee). Praktiliselt piisab oskustest määrata nimesid lahtritele ja lahtriplokkidele, lisada ja eemaldada töölehti ja lahtriplokke, salvestada ja avada töövihikuid. Veidi tuleb kokku puutuda ka joonestamisvahendite kasutamisega ja vormindamisega.

Eeldatakse, et lugeja juba omab baasteadmisi programmeerimisest ning on töötanud Scratchiga. VBA ja programmeerimise üldisi mõisteid ja vahendeid püütakse selgitada ja kinnistada Scratchist omandatud teadmiste ja oskuste abil.

Kiirtutvus VBAga

Programmide ja protseduuride liigid ning struktuur

VBA toega rakendus võimaldab hallata kasutajaliidest ning määrata operatsioone ja tegevusi andmete ja baasrakenduse (Excel, Word, AutoCAD jm) objektidega:

- **andmed** – arvud, tekstid, kuupäevad jms,
- **objektid** – Exceli töövihikud, töölehed, lahtrid, graafilised kujundid, Wordi dokumendid jms.

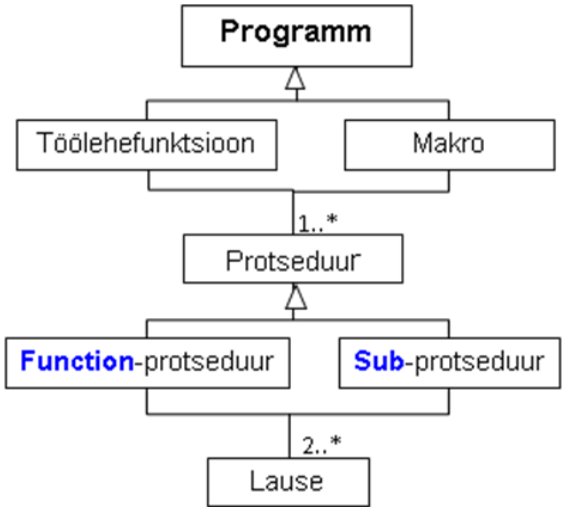
Excelis saab kasutada kahte liiki VBA programme:

- **makrod** – võimaldavad määratleda suvalisi tegevusi, käivitatakse näiteks käsunupuga,
- **töölehefunktsioonid** – võimaldavad leida ainult väärtusi, saab kasutada Exceli valemites.

Üldjuhul võib programm koosneda mitmest **protseduurist**. Protseduure on kahte liiki:

- **Sub** – protseduurid ehk alamprogrammid,
- **Function** – protseduurid ehk funktsioonid.

Mitmeprotseduurilises programmis on üks protseduur alati **peaprotseduur**, teised on **alamprotseduurid**.



Viimased võivad omakorda kasutada teisi alamprotseduure jne. Programmi täitmine algab peaprotseduurist sellekäivitamisel. Alamprotseduur hakkab tööle siis, kui selle poole pöördutakse peaprotseduuris või teises alamprotseduuris. Makro peaprotseduur on alati **Sub**-protseduur ja töölehefunktsioonil **Function**-protseduur. Alamprotseduurid võivad olla mõlemal suvalised. Lihtsamal juhul koosneb makro ühest **Sub**-protseduurist, töölehefunktsioon ühest **Function**-protseduurist.

NB! Töölehefunktsioone saab kasutada ainult Excelis.

Protseduur koosneb lausete (korralduste) jadast. Lausete abil määratakse vajalikud tegevused ja nende täitmise järjekord, esitatakse programmi ja protseduuride struktuur, kirjeldatakse andmeid jm.

Laused koosnevad võtmesõnadest (kindla tähendusega ja kasutamiskohaga sõnad: **Sub**, **Dim**, **End**, **If** jm), nimedest, avaldistest jm.

Protseduuride tüüpstruktuur

Makro peaprotseduur	**Sub**-tüüpi alamprotseduur	**Function**-protseduur – suvaline
Parameetriteta Sub-**protseduur**	Võivad olla sisend- ja väljund-parameetrid	Võivad olla ainult sisend-parameetrid
Sub *nimi* ( ) *laused* [ja *kommentaarid*] **End Sub**	**Sub *nimi*** (**[*parameetrid*]**) [*laused* ja *kommentaarid*] **End Sub**	**Function *f_nimi*** (**[*parameetrid*]**) [*laused ja kommentaarid*] *f_nimi = tagastatav väärtus* **End Function**
NB! Makro peaprotseduuril ei tohi olla parameetreid! Nime järel peavad olema tühjad sulud!	**Pöördumine** Call-**lausega:** [Call] *nimi* ([*argumendid*])	Pöördumine **funktsiooniviidaga**: *f_nimi* ([*argumendid*])

NB! Siin ja edaspidi tähendavad **nurksulud**, et nende vahel olev element võib puududa. Näiteks ei pea igal protseduuril olema parameetreid. Sellisel juhul puuduvad pöördumisel ka argumendid.

Andmevahetuseks protseduuride vahel saab kasutada **parameetreid** ja **argumente**. **Sub**-protseduuridel võivad olla **sisend-** ja **väljundparameetrid**. Viimastest igaüks tagastab ühe väärtuse. Funktsioonil võivad olla ainult sisendparameetrid: tulemus ehk **tagastatav väärtus** (saab olla ainult üks) omistatakse funktsiooni

NIME NIMELISELE muutujale ja selle kaudu tagastatakse väljakutsuvale protseduurile. Argumentide arv, liik ja järjekord peab vastama parameetritele. Andmevahetuseks võib kasutada ka **globaalseid muutujaid**. Protseduuride sees kasutatakse sisemuutujaid ehk **lokaalseid muutujaid**. Scratchi skriptid vastavad enam-vähem protseduuridele.

Protseduurid salvestatakse moodulilehtedele ehk **moodulitesse**, mis asuvad töövihiku eraldi osas **VBA-projektis**, kasutades Visual Basicu **redaktorit** (**VBE** – *Visual Basic Editor*), mis toetab programmide sisestamist ja redigeerimist, kontrollides operatiivselt keele süntaksit, pakkudes spikreid, abiinfot jms.

Makrode käivitamiseks on mitmeid erinevaid võimalusi. **Makro** saab käivitada alati Exceli menüü vahekaardilt *Developer* (**Arendaja**). Käivitamise kiirendamiseks ja hõlbustamiseks võib paigutada töölehele käsunupu või mingi graafikaobjekti ning siduda makro sellega. **Töölehefunktsiooni** käivitamine toimub Exceli valemist **funktsiooniviida** abil.

Protseduuride täitmist korraldab spetsiaalne programm – VBA **interpretaator** (translaator). Saades korralduse mingi protseduuri täitmiseks, eraldab interpretaator protseduurile täitmise ajaks arvuti mälus **tööpiirkonna** (**programmiplokk** ja **andmeplokk**). Programmiplokki tuuakse vastava protseduuri tekst ning kontrollitakse selle süntaksit. Vea korral väljastatakse veateade ja täitmine katkestatakse. Kui vigu ei ole, eraldatakse andmeplokis mälupesad (väljad) protseduuri poolt kasutatavate muutujate väärtuste ajutiseks säilitamiseks. Seejärel tõlgib interpretaator protseduuri laused masinkeelde, suunab selle täitmiseks protsessorisse ning tagastab tulemused andmeplokki.

Näide: Tutvus

Inimese pikkuse ja massi (kaalu) alusel leitakse tema kehamassi indeks:

k_{ind} = mass/pikkus^2 , mass (kaal) – kg, pikkus – m

ja selle alusel antakse hinnang tema nö saleduse kohta, arvestades järgmisi kriteeriume:

$$k_{ind} < 18 - \text{kõhn}; 18 \le k_{ind} \le 25 - \text{normaalne}; \; 25 < k_{ind} \le 30 - \text{ülekaal}; \;\; k_{ind} > 30 - \text{suur ülekaal}$$

Näite abil teeme tutvust VBA põhimõistete ja vahenditega: andmed ja objektid, protseduurid, laused, andmete sisestamine ja väljastamine, valikute ja korduste kirjeldamise lihtsamad võimalused jms.

Vaadeldakse kolme põhivarianti:

- Makro. Interaktiivne andmevahetus dialoogibokside abil,
- Makro. Algandmed ja tulemused töölehe lahtrites, andmevahetus töölehtedega,
- Töölehefunktsioonid. Andmed ja valemid töölehel.

Variant 1. Ülesande püstitus ja kasutajaliides

Programm kuvab tutvustava teate ning küsib ja loeb kasutaja nime. Seejärel küsitakse, kas kasutaja tahab lasta leida oma kehamassi indeksi. Kui jah, siis programm küsib ja loeb kasutaja poolt sisestatud pikkuse ja massi, leiab ja kuvab kehamassi indeksi ning annab hinnangu selle kohta. Kui kasutaja ei soovi sellist infot, väljastab programm kahetsuse ja lõpetab töö.

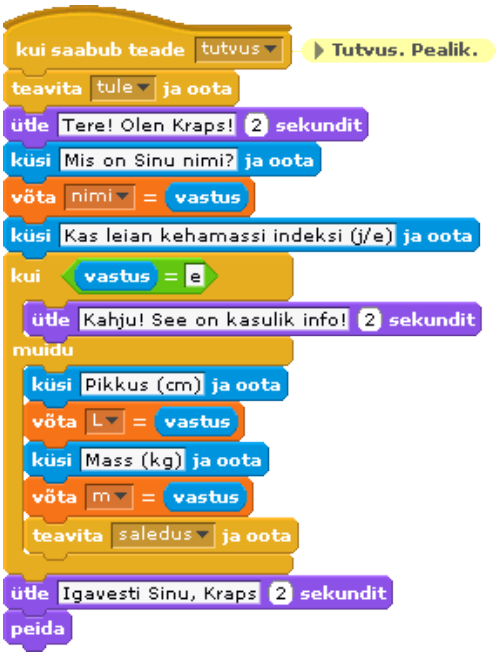
Kasutajaliides asub töölehel, sisaldades rohkem elemente kui otseselt vaja. Piisaks ju ainult käsu-nupust programmi käivitamiseks, kuid isegi selleta saaks hakkama. Töölehel on ka kaks graafika-objekti: Juku ja Kraps. Nende abil tutvustatakse graafikaobjektide kasutamise põhimõtteid. Enne programmi töö lõppu teevad nad salto ning nende asukoht töölehel muutub.

Kui lugeja tahab proovida antud programmi käima panna, võib skeemil näidatud pildikeste asemel võtta suvalised graafikaobjektid. Võib näiteks joonistada mingi kujundi, kasutades Exceli korraldust **Insert Shapes** või lisades pildi failist korraldusega **Insert Picture**. Oluline on, et nimed vastaksid programmis olevatele nimedele (vt programmi viimased read). Nime panemiseks või muutmiseks teha pilt aktiivseks ja tippida nimekasti valemirea vasakus servas vajalik nimi, seejärel vajutada kindlasti klahvile **Enter**. Vt jaotist „Graafilised kujundid – klass Shape“.

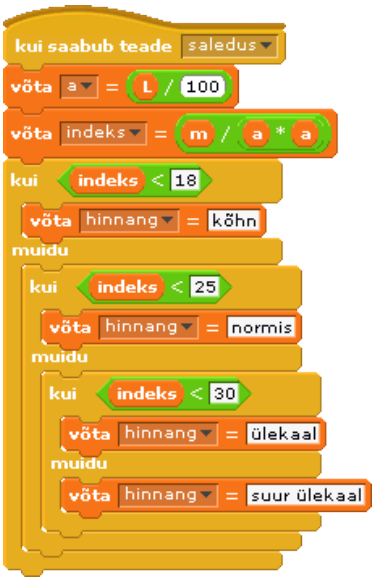
Programm

Programm (makro) koosneb kolmest protseduurist: peaprotseduur **Tutvus** ning alamprotseduurid **saledus** (**Function**-protseduur) ja **Liigu** (**Sub**-protseduur). Viitamiseks graafikaobjektile kasutatakse konstruktsiooni Shapes(“nimi“).

```
Sub Tutvus() 'peaprotseduur
   Dim nimi, v, pikkus, mass, indeks 'muutujad
   MsgBox  "Tere! Olen Juku!"   ' teate kuvamine
   nimi = InputBox("Sinu nimi => ")   ' nime lugemine
   If nimi = "" Then MsgBox "Kedagi ei ole!": End
   v = InputBox("Leian keha indeksi (1 - jah  / 0 - ei )? ")
   If Val(v) <>  1 Then  ' valiku algus
      MsgBox nimi + "! Kahju! Ehk mõtled veel?"
      Exit Sub
   Else              ' alternatiiv
      pikkus = InputBox (nimi & "! Sinu pikkus (cm) ")
      mass = InputBox ("aga mass/kaal (kg) ")
      indeks = mass / (0.01 * pikkus) ^ 2
      MsgBox nimi & "! Sinu indeks on:" & Round(indeks,1)
      MsgBox nimi & "! Oled " & saledus (pikkus, mass)
   End If
   MsgBox "Kohtumiseni! Igavesti Sinu, Juku ja Kraps!"
   Call Liigu (Shapes("Juku"), 200, 100)  'AP kutsung
   Liigu Shapes ("Kraps"), 100, 50 ' Call on ära jäetud
End Sub
```



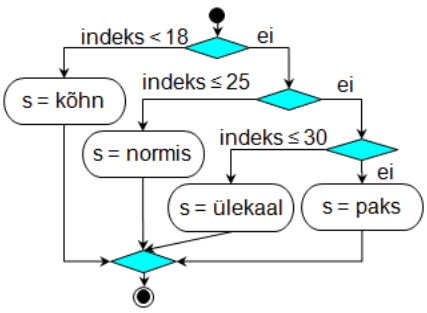

```
Function saledus(L, m)
' hinnang massiindeksi alusel
 ' L – pikkus, m – mass
  Dim indeks
  indeks = m / (L / 100) ^ 2
  If indeks < 18 Then
     saledus = "kõhn"
  Else
     If indeks <= 25 Then
        saledus = "normis"
     Else
        If indeks <= 30 Then
           saledus = "ülekaaluline"
        Else:
           saledus = "suur ülekaal"
        End If
     End If
  End If
End Function
```



Parameetrite **L** ja **m** abil antud pikkuse ja massi väärtustest arvutatakse kehamassiindeks.

Selle alusel leitakse hinnang, mis omistatakse funktsiooni **NIME NIMELISELE** muutujale.

Valik tehakse mitmetasemelise **If**-lause abil



```
Sub Liigu (kuju, x, y)
 ' kuju pöörlemine ja asukoht
   Dim i
   For i = 1 To 360
      kuju.IncrementRotation 1
      DoEvents
   Next i
   kuju.Left = x * Rnd()  ' vasak
   kuju.Top = y * Rnd()  ' ülemine
End Sub
```

Parameetriteks on graafikaobjekt **kuju** ja selle sihtkoha koordinaadid (x, y). Konkreetne kuju ja koordinaatide maksimaalsed väärtused antakse pöördumisel argumentide abil.

Kuju teeb pöörde 360°. **For**…**Next**-lause määrab **korduse**: i muutub 1 kuni 360 ja iga kord täidetakse korduses olevad laused.

Meetod **IncrementRotation** muudab kuju pöördenurka (siin iga kord 1°). Käsk **DoEvents** tagab, et pöörlemine oleks nähtav.

Omaduste **Left** ja **Top** väärtused määravad kuju asukoha. Funktsioon **Rnd**() annab juhuarvu vahemikus 0 kuni 1.

Programmide sisestamine, redigeerimine ja käivitamine

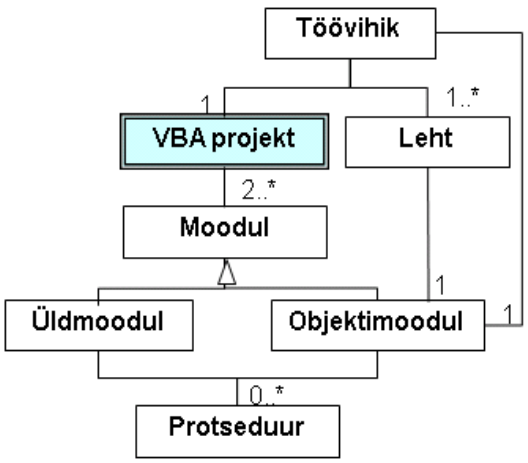
Soovitame kopeerida programmi VBA redaktorisse, panna see käima ning katsetada ja täiendada. Alguses võib graafikaobjektid ja protseduuri **Liigu** ära jätta. Viimased kaks rida peaprotseduuris võib sellisel juhul muuta kommentaarideks, lisades nende ette ülakomad. Programmi **peaprotseduuri** ja alamprotseduuri **Liigu** võiks salvestada **lehemoodulisse** (vt allpool), millel asub kasutajaliides, funktsiooni **saledus**, aga üldmoodulisse, mis oleks kasulik ka järgmiste variantide jaoks.

VBA projekti struktuur

Programmi protseduurid salvestatakse VBA projekti moodulitesse Visual Basicu redaktori (VBE – *Visual Basic Editor*) abil. Redaktori aktiveerimiseks saab kasutada mitmeid erinevaid viise. Esmakordsel pöördumine võib kasutada käsku (nuppu) **Visual Basic** vahekaardilt **Developer**. Redaktori saab käivitada ka klahvikombinatsiooniga **Alt+F11**. VBE keskkonda pääseb, klõpsates hiire parempoolse nupuga lehelipikul ja valides ilmuvast objektimenüüst korralduse **View Code**, mis viib antud töölehe moodulisse. Kui Visual Basic redaktorit on antud seansis juba kasutatud, asub tegumiribal ikoon, mis võimaldab kohe pääseda redaktoriaknasse.

VBA projekti põhikomponentideks on **moodulid**, kuhu saab salvestada protseduure. Kaheks peamiseks mooduli liigiks on **objektimoodulid** ja **üldmoodulid**.

Oma objektimoodul on igal lehel ja ka töövihikul. Need luuakse ja eemaldatakse koos töövihiku ja lehe loomisega. **Lehemoodulisse** salvestatakse tavaliselt sellised protseduurid, mis on tihedalt seotud antud lehega, kasutades näiteks ainult antud lehe objekte, lehe ja selle alamobjektide sündmusprotseduure jms.

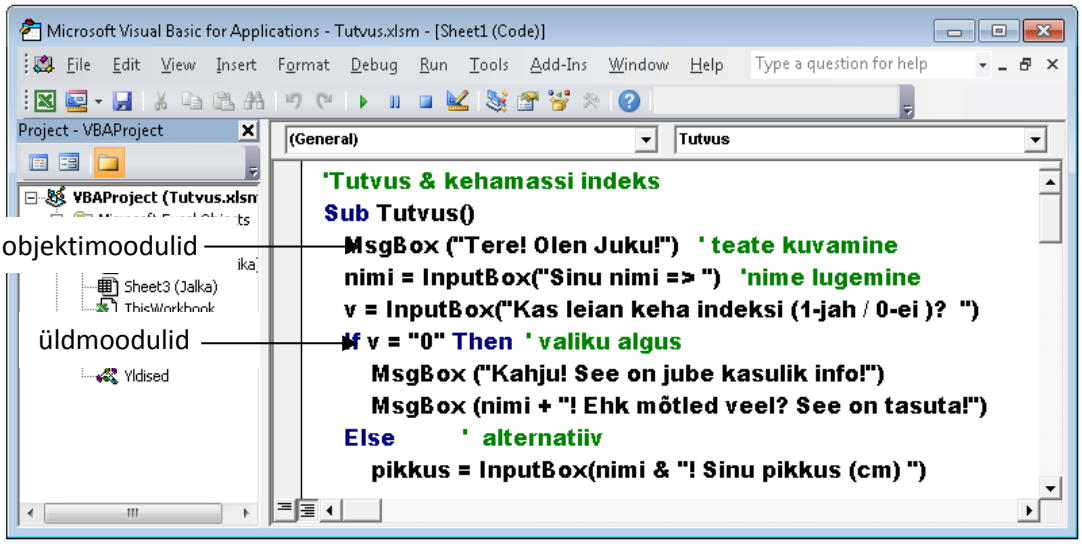


Üldmoodulisse võib salvestada suvalisi protseduure. Tavaliselt paigutatakse neisse üldise iseloomuga protseduure, mis on seotud mitme lehega. Üldmooduleid algselt töövihikus ei ole, kuid neid saab alati lisada korraldusega **Insert Module** ja nende arv ei ole piiratud. Vaikimisi on üldmoodulite nimed **Module1**, **Module2**, …, mida saab muuta omaduste aknas (*Properties Window*).

Üldmooduleid saab ka eemalda.

NB! Töölehefunktsioonid peab tingimata salvestama **üldmoodulitesse**.

Redaktori kasutajaliides



Redaktori akna ülemises servas on **menüü** ja **standardriistariba**. Ekraani vasakus servas asub **projektihalduri** aken, milles on kuvatud moodulite puukujuline struktuur (kaustad ja moodulid) iga projekti (avatud töövihiku) jaoks. Alguses on objektimoodulite (lehtede ja töövihiku) moodulite kaust ***Microsoft Excel Objects*** ja seejärel (kui on) – üldmoodulite kaust ***Modules***. Vajaliku mooduli aktiveerimiseks peab tegema topeltklõpsu selle nimel. Samas piirkonnas on alguses tavaliselt ka põhiobjektide (lehed, töövihik, moodulid) omaduste (atribuutide) aken ***Properties Window*** (pildil ei ole näidatud), mille võib esialgu sulgeda. Vajadusel saab selle kuvada korraldusega **View**, **Properties Window**.

Põhiosa ekraanist võtab enda alla **koodi-** ehk **programmiaken**, kuhu sisestatakse mooduli protseduurid. Protseduuride arv ühes moodulis ei ole piiratud. Pildil on aktiivne töölehe **Sheet1** (**Tutvus**) moodul, milles on makro **Tutvus** protseduurid.

Lehemooduli jaoks on projektihalduris toodud kaks nime: VBA sisenimi ja välisnimi ehk lihtsalt nimi (sulgudes). Sisenimi on algselt esitatud kujul **SheetN**, kus **N** on lehe järjenumber, mida suurendatakse automaatselt lehtede lisamisel. Sisenimesid saab muuta omaduste (*Properties*) aknas. Välisnimi on näha lehe nimelipikul ning seda saab muuta Exceli aknas või VBA protseduuris. Kui nimesid ei ole muudetud, langevad sise- ja välisnimi kokku.

Programmi teksti sisestamisel kontrollib VBA redaktor tavaliselt automaatselt ridade kaupa lausete süntaksi õigsust. Kui antud rea mingis lauses on süntaksivigu, siis kuvatakse üleminekul uuele reale vastav teade, muudetakse reas oleva teksti värvust (vaikimisi punaseks) ning paigutatakse kursor vea eeldatavale asukohale, mis ei pruugi olla alati päris täpne.

Tüüpilised süntaksivead on võtmesõnade vale esitus, tehtemärkide ärajätmine avaldistes (eeskätt korrutamistehtes), eraldajate (tühikud, komad, sulud, jutumärgid, …) ärajätmine vajalikes kohtades, tühikute kasutamine nimedes või võtmesõnades jms.

VBA redaktori tugi on väga kasulik objektide omaduste ja meetodite sisestamisel. Kui peale objektiviita sisestada punkt, pakub redaktor antud klassi objektide omaduste ja meetodite loendit vajaliku elemendi valimiseks, hoides nii kokku aega ja vähendades vigade tekkimise võimalust.

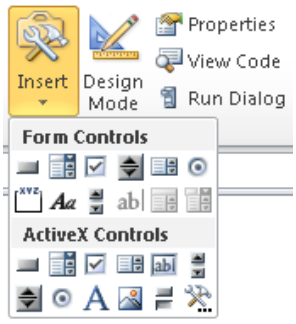
Informatsiooni objektide (klasside) ning nende omaduste ja meetodite kohta saab objektisirvija ***Object Browser*** abil ning VBA abiinfosüsteemist ***Microsoft Visual Basic Help***. Viimasest saab informatsiooni ka programmeerimiskeele lausete, funktsioonide ja muude elementide kohta. Informatsiooni saamiseks konkreetse võtmesõna, omaduse, meetodi jmt kohta, võib kursori viia vastava elemendi kohale ja vajutada klahvile **F1**.

Makrode käivitamine

Makro käivitamine toimub enamasti Exceli aknas, kusjuures tavaliselt peab muutma aktiivseks töölehe, millel paiknevad makro poolt kasutatavad objektid ja andmed.

Käivitamiseks võib alati kasutada dialoogiboksi **Macro**, mille saab kuvada käsuga **Macros** vahekaardil **Developer** (**Arendaja**), kui see ei ole peidetud, kuid võib kasutada ka kiirkäsku **Alt+F8**. Seejärel kuvatakse dialoogiboks **Macro**. Kuvatud loetelust saab valida vajaliku makro ning peale klõpsamist nupule **Run** asub VBA selle täitmisele.

Makro kiireks käivitamiseks võib paigutada töölehele makroga seotud käsunupu või mingi graafikaobjekti. Makro käivitatakse hiireklõpsuga vastaval nupul või graafikaobjektil.



Käsunuppe on kahte tüüpi: Exceli vormi (***Form***) käsunupud ja **ActiveX** käsunupud. Viimased on uuemad ja pakuvad rohkem võimalusi, kuid ActiveX käsunupu loomine ja sidumine makroga on mõnevõrra keerulisem. Alguses võiks proovida **Form** käsunuppe, kuid üsna pea võiks üle minna **ActiveX** käsunuppudele. Nii ühe kui teise loomiseks kasutatakse vahekaardi **Developer** (arendaja) grupi **Controls** (juhtelemendid) käsku **Insert** (lisa). Selle klõpsamisel kuvatakse kaks tööriistakasti: **Forms Controls** (vormi juhtelemendid) ja **ActiveX Controls** (ActiveX juhtelemendid), kus saab teiste Windowsi ohjurite (juhtijate) seast valida käsunupu.

Makro sidumine Form käsunupuga

Valida tööriistakastist käsunupp ja hiireklikiga lisada see töölehele. Ilmub dialoogiboks **Assign Macro** (makro omistamine), kus kuvatakse olemasolevate makrode loetelu, millest saab:

- valida vajalik makro ja klõpsata nuppu OK
- muuta nupul olevat teksti (ei ole kohustuslik)

Makro sidumine ActiveX käsunupuga

Valida tööriistaboksist käsunupp ja hiireklikiga lisada see töölehele. Nupul on tekst CommandButtonN, kus N on 1, 2, … , tähistades sama tüüpi nupu järjenumbrit antud töölehel. Kui tegemist on esimese (ainukese) nupuga, siis N on 1 ning sisse lülitub disainirežiim (vt nupp **Design Mode**).

Nupu sidumine loodud makroga

1. Tehes topeltklõpsu makronupul, kuvatakse antud lehe moodul, millesse on lisatud sündmusprotseduuri mall:

   Private Sub ***CommandButtonN_Click*()**

   End Sub

2. Protseduuri alguse ja lõpu vahele lisatakse käivitatava protseduuri nimi. Seejärel tuleks siirduda tagasi töölehele.
3. Muuta nupul olevat teksti (muutmine ei ole kohustuslik, kuid reeglina seda tehakse).
4. Klõpsata nuppu **Properties** (Atribuudid), mille tulemusena kuvatakse nupu omadused (*Properties*).
5. Asendada reas **Caption** tekst **CommandButtonN** oma soovitud tekstiga.

Makro sidumine graafikaobjektiga

1. Lisada töölehele suvaline kujund (**Insert** , **Shapes**), mille võib ka importida.
2. Teha kujundil hiire paremklikk ning valida ilmuvast objektimenüüst korraldus **Assign Macro**.
3. Kuvatud dialoogiboksist **Assign Macro** (nagu **Forms** käsunupu korral) valida makro ja klõpsata nuppu **OK**.

Kui **Forms** käsunupuga või graafikaobjektiga on seotud makro, käivitab hiireklikk sellel vastava makro koheselt. Nupu või graafikaobjekti asukoha, suuruse vms muutmiseks on vaja hiirekliki tegemise ajal hoida all klahvi **Ctrl**.

Lausete struktuur ja põhielemendid

Laused on korraldused (käsud), mis määravad tegevusi. Lausetel on keelereeglitega määratud kindel otstarve, struktuur – **süntaks** ja täitmise reeglid – **semantika**. Lausete põhielemendid on järgmised:

- **Võtmesõnad**: kindla asukohaga, kujuga ja tähendusega sõnad (Sub, Function, If, Else, …)
- **Funktsiooniviidad**: Round(indeks,1), MsgBox("Tere! "), InputBox("mass"), saledus(pikkus, mass)
- **String**- ja **arvkonstandid**: "normaalne", "Sinu nimi => ", "juku", 100, 0.01, -31.71 jms
  NB! Stringkonstandid paigutatakse jutumärkide vahele
  NB! reaalarvudes on murdosa eraldajaks punkt
- **Protseduuride**, **muutujate**, **parameetrite** nimed: saledus, Liigu, v, pikkus, L, m, x1, x_1, a13s1
  Nimede esitamiseks kehtivad ühtsed reeglid:

> **Nimi** võib koosneda ühest tähest või tähtede, numbrite ja allkriipsude jadast, mis algab tähega. Nimes ei tohi olla tühikuid. Suur- ja väiketähti ei eristata.

- **Avaldised**: mass / (0.01*pikkus) ^2, nimi & "! Sinu pikkus (cm) ", nimi = " ", indeks < 18
- Avaldiste põhiliikideks on arvavaldised, stringavaldised ja loogikaavaldised (erijuht: võrdlus)
- **Tehtesümbolid** ehk **operaatorid**: ^, *, /, +, -, =, <=
- **Piirajaid** ja **eraldajad**: ( ) , ” : .

Laused jagunevad **lihtlauseteks** ja **liitlauseteks**. Liitlause võib sisaldada teisi liht ja/või liitlauseid. Liitlauseteks on vaadeldavas näites **if**-laused peaprotseduuris ja funktsioonis **saledus** ning **for**-lause protseduuris **Liigu**. Kõik teised on lihtlaused.

Muutujate deklareerimine. Dim-lause

Dim nimi, v, pikkus, mass, indeks, lahter

See lause määratleb – öeldakse ka deklareerib – **muutujad**. Lihtsamal juhul antakse **Dim**-lauses muutujate nimede loetelu, kusjuures iga muutuja jaoks võib näidata tema väärtuste tüübi – muutuja tüübi.

Näiteks:

Dim nimi **As String**, v **As Integer**, pikkus **As Double**, mass, indeks

Muutujate tüübid näidatakse vastavate võtmesõnadega: muutuja **nimi** tüübiks on string (tekst või sõne), **v** tüübiks on täisarv, **pikkus** tüübiks on reaalarv. Kui muutuja tüüp pole **Dim**-lauses antud nagu ülaltoodud näites muutujad **mass** ja **indeks**, valib süsteem (VBA interpretaator) selle ise.

Enamasti ei ole lihtsamates programmides muutujate tüübi määramine vajalik ning sellest saadav mälumahu kokkuhoid märkimisväärne. Hiljem näeme, et tüüpide määramisest on teatud kasu objektimuutujate jaoks programmide sisestamisel ja redigeerimisel.

Muutujate deklareerimine ei ole Visual Basicus kohustuslik, kuid muutujate loetelu andmine (ka ilma tüüpi määramata) aitab lihtsustada võimalike vigade peilimist.

Vt jaotist „Andmete liigid ja tüübid“.

Enne protseduuri täitmist eraldab VBA interpretaator igale muutujale oma tööpiirkonnas mäluvälja ehk pesa. Programmi täitmise ajal saab vastavate lausete toimel salvestada nendesse pesadesse väärtusi. Vajadusel saab hiljem lugeda neid väärtusi näiteks uute väärtuste leidmiseks.

Scratchis luuakse muutujad „käsitsi“, kasutades grupi **Muutujad** käsku **Tee muutuja**. Laval kuvatakse monitor, kus näeb muutuja jooksvat väärtust. Monitori saab ka peita. Muidu on sama lugu nagu enne – ikka mälupesa, ei midagi muud.

Andmete sisestamine ja väljastamine dialoogiboksidega

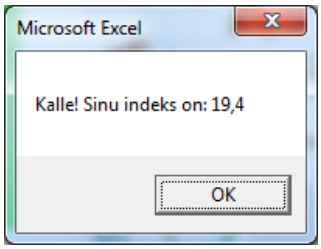
Suurem osa näite peaprotseduuri lausetest on seotud andmete sisestamisega ja väljastamisega dialoogibokside abil (vt jaotist „Dialoogibokside kasutamine“). Siin näidatakse nende lihtsamaid võimalusi.

Andmete kuvamiseks kasutatakse protseduuri **MsgBox**, millel lihtsamal juhul on järgmine kuju:

MsgBox *avaldis*

Üldjuhul on tegemist arv- või stringavaldisega, kuid väga tihti on tegemist lihtsalt stringiga, mille abil kuvatakse teade, näiteks:

MsgBox "Tere! Olen Juku!" või **MsgBox** "Hello, World!“



Kõige tüüpilisem on stringavaldiste kasutamine, mis moodustatakse stringkonstantidest ning arv- või stringmuutujatest sidurdamistehte abil.

Sidurdamistehte operaator on **&** või **+** (erinevus on vaid eri tüüpi operandide kasutamisel).

MsgBox nimi + "! Sinu indeks on:" & **Round**(indeks, 1) või **MsgBox** "pindala = " & S

Scratchis kasutatakse sel puhul käsku 

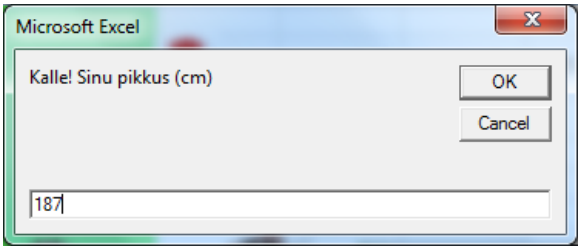
Stringavaldise saab panna kokku plokiga 

Andmete lugemiseks klaviatuurilt kasutatakse omistamislause erijuhtu, kus paremas pooles on VBA sisefunktsioon **InputBox**:

muutuja = **InputBox** (teade, …)

Esimene (kohustuslik) argument **teade** esitatakse enamasti stringkonstandi abil, kuid see võib olla ka stringavaldis. Näiteid vaadeldavast protseduurist:

nimi = **InputBox** ("Sinu nimi => ") : pikkus = **InputBox** (nimi & "! Sinu pikkus (cm) ")



Lause täitmisel programmi töö peatakse ja kuvatakse dialoogiboks, milles on teade ja väli, kuhu saab sisestada väärtuse. Peale nuppu **OK** klõpsamist, omistatakse antud väärtus muutujale ja programmi töö jätkub. Kui väärtust ei sisestata ja klõpsatakse **OK** või **Cancel**, omistatakse muutujale tühi väärtus!

Sisendboksist loetud arv on salvestatud muutuja väljas tekstivormingus (vt jaotist „Dialoogibokside kasutamine“). Selliselt esitatud arvuga võib tekkida probleeme aritmeetika ja võrdlemise operatsioonide täitmisel. Seetõttu on sageli otstarbekas teisendada väärtus arvuvormingusse, kasutades näiteks funktsiooni **Val**. Nii on seda tehtud näiteks võrdluses lauses **If** Val(v) <> 1 **Then**. Kui on kindlalt teada, et väärtust ei kasutada liitmisoperatsioonides ja võrdlustes, võib väärtuse jätta ka teisendamata.

Scratchis kasutatakse andmete lugemiseks klaviatuurilt plokki küsi teade ja oota. Väärtus salvestatakse sisemuutujas **vastus**. Vajaduse korral saab selle omistada suvalisele muutujale, kasutades käsku võta muutuja = vastus

Omistamisest ja omistamislausest

Omistamist kasutatakse programmides väga sageli.
Lause üldkuju enamikes tekstipõhistes programmeerimiskeeltes on:

muutuja = avaldis

Lause täitmisel leitakse avaldise väärtus ja saadud tulemus omistatakse antud muutujale: st salvestatakse muutuja väljas ehk pesas. Avaldise erijuhtudeks on konstant, muutuja ja funktsiooniviit.
Üldkujul kujul esinevad omistamislaused näite peaprotseduuris ja funktsioonis **saledus**:
indeks = mass / (0.01 * pikkus) ^ 2 ja indeks = m / (L / 100) ^ 2

Omistamislause erijuhtu vaadeldi andmete sisestamisel funktsiooniga **InputBox**. Samuti on erijuhuks omistamislause **saledus = *konstant*** funktsioonis **saledus**. Siin omistatakse väärtus funktsiooni NIME NIMELISELE muutujale.

Üheks võimaluseks on ka väärtuse omistamine objekti omadusele, nagu näiteks

kuju. Left = x * Rnd()

– objekti omaduse **Left** väärtuseks võetakse avaldise väärtus.

Scratchis on kaks omistamiskäsku:
võta muutuja = avaldis ja muuda muutuja avaldis võrra

Teine on erijuht, mille saab panna kirja ka nii: [võta muutuja = (muutuja + avaldis)]

Valikud ja If-lause

Kõikides programmeerimiskeeltes on valik (**If**-lause) üks peamisi protsesside juhtimisega seotud tegevusi. Enamikes keeltes on vähemalt kaks **If**-lause varianti, mis võimaldavad teha valiku ühest (laused kas täidetakse või jäetakse vahele) või valiku kahest (täidetakse üks kahest käsuplokist). Paljudes programmeerimiskeeltes on vahendid mitmese valiku kirjeldamiseks. Visual Basicus on olemas kõik kolm varianti, mis esinevad ka järgnevas näites.

Peaprotseduuris on kasutusel valik ühest ja valik kahest. Valiku korral ühest on kaks varianti: üherealine lause ja mitmerealine lause. Esimest varianti tasub kasutada ainult erijuhul – **If** -lausesse kuulub ainult üks või kaks lihtlauset.

Valik ühest

Üherealine If-lause

```
If nimi = "" Then MsgBox "Kedagi ei ole!": End
```

Mitmerealine If-lause

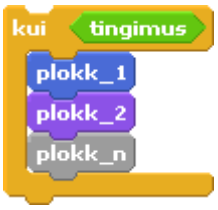
```
If v = "" Then
   MsgBox "Kahju!"
   Exit Sub
End If
```

Üherealine If-lause

```
If tingimus Then laused
```

Mitmerealine If-lause

```
If tingimus Then
   laused
End If
```



Valik kahest

```
If  Val(v ) = 0 Then   ' kui tingimus tõene
   MsgBox nimi + "! Ehk mõtled veel?"
  Exit Sub
Else                   ' tingimus väär
   pikkus = InputBox (…)
    …
End If
```

```
If  tingimus Then
   laused_1
Else
   laused_1
End If
```

Mitmese valikuga on tegemist funktsioonis **saledus**. Sellist valikut saab kirjeldada mitmel erineval viisil, kuid alati saab kasutada ühes **If**-lauses teisi **If**-lauseid. Praegu on esimese taseme **If**-lause **If**-harus üks lihtlause, **else**-harus (ehk **else**-osalauses) on kasutatud järgmise taseme **If**-lauset ja viimases veel ühte. Sisemisi lauseid võib jaotada harude vahel ka teisiti. Antud juhul võib näiteks panna esimese taseme **If**-harusse ning ka **else**-harusse veel **If**-lause.

Visual Basicus on aga veel üks võimalus, milleks on **ElseIf**-osalausete kasutamine.

Allpool on toodud mõned vaadeldava **If**-lause võimalikud esitamise variandid.

Mitmene valik

```
If indeks < 18 Then
   saledus = "kõhn"
Else
   If indeks <= 25 Then
      saledus = "normis"
   Else
      If indeks <= 30 Then
         saledus = "ülekaaluline"
      Else
         saledus = "suur ülekaal"
      End If
   End If
End If
```

```
If indeks <= 25 Then
   If indeks < 18 Then
      saledus = "kõhn"
   Else
      saledus = "normis"
   End If
Else
   If indeks <= 30 Then
      saledus = "ülekaaluline"
   Else
      saledus = "suur ülekaal"
   End If
End If
```

```
If indeks < 18 Then
   saledus = "kõhn"
ElseIf indeks <= 25 Then
   saledus = "normis"
ElseIf indeks <= 30 Then
   saledus = "ülekaaluline"
Else
   saledus = "suur ülekaal"
End If
```

Kasutajafunktsioonid

Kasutaja saab luua oma funktsioone, mille kasutamine programmides toimub analoogselt sise-funktsioonidega. Funktsiooni peaprotseduuriks on ***Function***-protseduur:

```
Function nimi ( [parameetrid] )
  protseduuri keha
  nimi = avaldis
End Function
```

Parameetritega esitatakse funktsiooni **sisendandmed** (kui vaja).

Näiteks on funktsiooni päis järgmine:
Function *saledus*(*L*, *m*)
Funktsiooni nimeks on **saledus**, parameetriteks: **L** – pikkus (cm) ja **m** – mass (kg).

Tüüpiliselt **funktsioon leiab ja tagastab ühe väärtuse**. Tagastatav väärtus omistatakse funktsiooni **NIME NIMELISELE** muutujale. Selliseid omistamisi võib olla mitu, kuid tagastatakse ainult üks väärtus (viimasena omistatu). Funktsiooni töö lõpetab tavaliselt lause **End Function**, kui täitmisjärg jõuab selleni. Funktsiooni töö saab katkestada lausega **Exit Function.** Tulemus ja täitmisjärg tagastatakse kohta, kust toimus pöördumine **funktsiooniviida** abil.

Funktsiooniviit esitatakse kujul: *nimi* (*argumendid*), kus *nimi* on funktsiooni nimi ning *argumendid* annavad parameetrite väärtused. Argumentide arv, tüübid ja järjekord peavad vastama parameetritele.

Funktsioon **saledus** tagastab väärtuse, mille leiab (valib) **If**-lause muutuja **indeks** väärtuse alusel. Viimane omakorda sõltub parameetrite **L** ja **m** väärtustest. Pöördumine funktsiooni poole toimub peaprotseduuri lausest:

```
MsgBox nimi & "! Oled " & saledus (pikkus, mass)
```

Parameetritele **L** ja **m** vastavad argumendid on esitatud muutujate **pikkus** ja **mass** abil.

Oma funktsioone saab kasutada ka töölehe valemites. Sellisel juhul peab protseduur asuma üldmoodulis.

Objektid, omadused ja meetodid

Protseduuris **Liigu** kasutatakse graafikaobjekte (vt „Graafilised kujundid – klass Shape“). Igal objektil on teatud hulk omadusi ja meetodid, mida saab kasutada tegevuste määramiseks. Omaduste ja meetodite valik ning nende nimetused sõltuvad objekti tüübist ehk klassist. Graafikaobjektide ehk kujundite (klass *Shape*) omadusteks on näiteks nimi (*Name*), vasaku ja ülemise serva koordinaadid (*Left* ja *Top*) jms. Objektide asukoha muutmiseks, pööramiseks, kopeerimiseks, … on mitmeid meetodeid.

Viitamiseks objektidele on erinevaid võimalusi. Graafikaobjektide puhul on põhivariant:

Shapes(*nimi*),

kus *nimi* on objektile töölehel määratud nimi, mis esitatakse stringkonstandi abil nagu näiteks Shapes(“Juku“).

Viitamiseks saab kasutada ka muutujaid.

Sub-tüüpi alamprotseduurid

Sarnaselt *Function*-protseduuridega on *Sub*-tüüpi alamprotseduurid mõeldud ühe kindlapiirilise (alam)ülesande täitmiseks. Erinevalt peaprotseduurist võivad neil olla ka parameetrid. Kirjeldatud protseduur algab lausega:

Sub *nimi* ([*parameetrid*])

Protseduuride käivitamiseks peaprotseduurist, teisest alamprotseduurist või ka funktsioonist, kasutatakse pöördumislauset ehk **Call**-lauset:

Call *nimi* ([*argumendid*]) või ***nimi*** [*argumendid*].

Esitatud variandid on samaväärsed. Siin ***nimi*** on protseduuri nimi; *argumendid* (mis võivad ka puududa) peavad vastama parameetritele. Võtmesõna **Call** ärajätmisel peab ära jätma ka sulud.

Allpool esitatud näites on kasutusel protseduur **Liigu** (kuju, x, y), mille parameetrid on järgmised: **kuju** – graafikaobjekt, **x** ja **y** – kuju sihtkoha koordinaadid (arvud). Protseduuri poole pöördutakse kaks korda lausetega **Call *Liigu***(Shapes("Juku"), 200, 100) ja ***Liigu*** Shapes("Kraps"), 100, 50.

Scratchi skriptid on üsna sarnased VBA protseduuridega ning plokk [**teavita** *teade* **ja oota**] on sarnane pöördumislausega (**Call**-lausega), kuid päris samad need ei ole, sest VBA pöördumislausega saab käivitada vaid ühe konkreetse protseduuri.

Scratchi praeguses versioonis ei ole parameetreid ja argumente. Scratchis saab samaaegselt (paralleelselt) käivitada mitu protseduuri:

Scratchis kasutaja funktsioone praegu veel ei ole.

Makro Tutvus. Variant 2 – tööleht

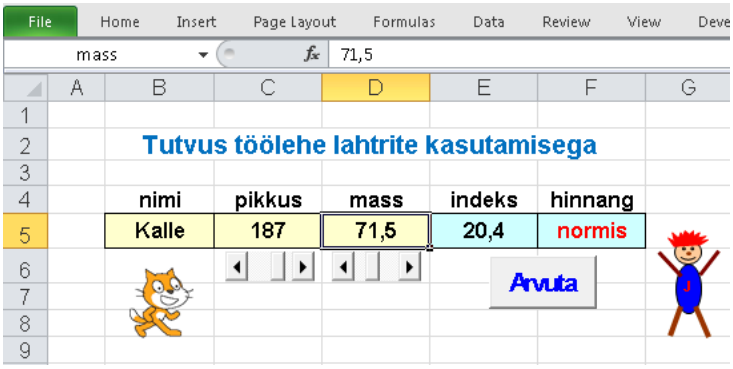
Võimalus kasutada töölehti andmete säilitamiseks ja kasutajaliideste tegemiseks lihtsustab rakenduste loomist, vähendades selleks kuluvat aega, eriti kui on vaja kasutada tabeleid. Viimased on aga kõikides eluvaldkondades üheks peamiseks vahendiks andmete korrastatud esitamiseks. Olgu märgitud, et sageli kujutavad endast andmebaasid omavahel seotud tabeleid.

Kasutajaliides

Allpool toodud näite abil tutvustame lihtsamaid võimalusi kasutajaliidese loomiseks töölehel.

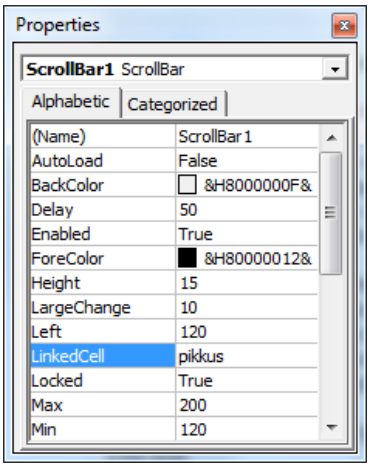
Kõigepealt tuleb valida kohad (lahtrid ja lahtriplokid) algandmetele ja tulemustele, kasutades mitmesuguseid abi- ning kujunduselemente nagu pealkirjad, päised, selgitused, teated jms. Lisaks nimetatutele võib kasutada ka mitmesuguseid vahendeid andmete sisestamiseks ja protseduuride käivitamiseks. Nendeks võivad olla kerimisribad, märkeruudud, käsunupud, …

Kuna programmides kasutatakse lahtritele ja lahtriplokkidele viitamiseks nimesid, siis üheks oluliseks tööks enne programmi kirjutamist on nimede määramine.



Vaadeldaval juhul võiks kasutajaliides olla näiteks järgmine. Algandmete jaoks on lahtrid **nimi**, **pikkus** ja **mass**, tulemuste jaoks **indeks** ja **hinnang**. Pikkuse ja massi väärtuste sisestamiseks on võimalus kasutada kerimisribasid. Kasutades programmis viitamiseks lahtritele nimesid, ei ole kasutajaliidese täpsel asukohal erilist tähtsust.

Inimese nime lülitamine algandmete hulka on üsna tinglik, sest ülesande lahendamiseks ei ole see otseselt vajalik. Formaalselt pole ülesande lahendamise seisukohast olulised ka pealkirjad lahtrite kohal. Need on vajalikud kasutajale, mõistmaks, kus mingid andmed on. Loomulikult ei ole mingit praktilist väärtust **Juku** ja **Krapsu** pildikestel, kuid mõnikord kasutatakse selliseid „kaunistusi“. Siin on need selleks, et näidata, kuidas saab protseduure siduda graafikaobjektidega.



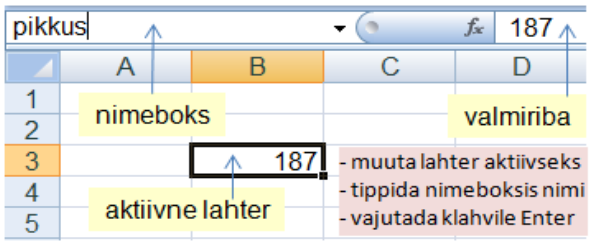
Kerimisriba (*ScrollBar*) saab lisada töölehele vahekaardi *Developer* (Arendaja) grupi *Controls* (Juhtelemendid) käsuga *Insert* (Lisa). Samas on ka mitmed teised kasutajaliidese komponendid nagu **märkeruudud**, **komboboksid**, **käsunupud** jms.

Käsuga *Properties* (Atribuudid) kuvatavas aknas saab määrata omaduste väärtused: minimaalne (Min) ja maksimaalne (Max) väärtus, lahter valitud väärtuse salvestamiseks (*Linked Cell*), muutmise sammud (*Small Change*, *Large Change*).

Nimede määramisest lahtritele

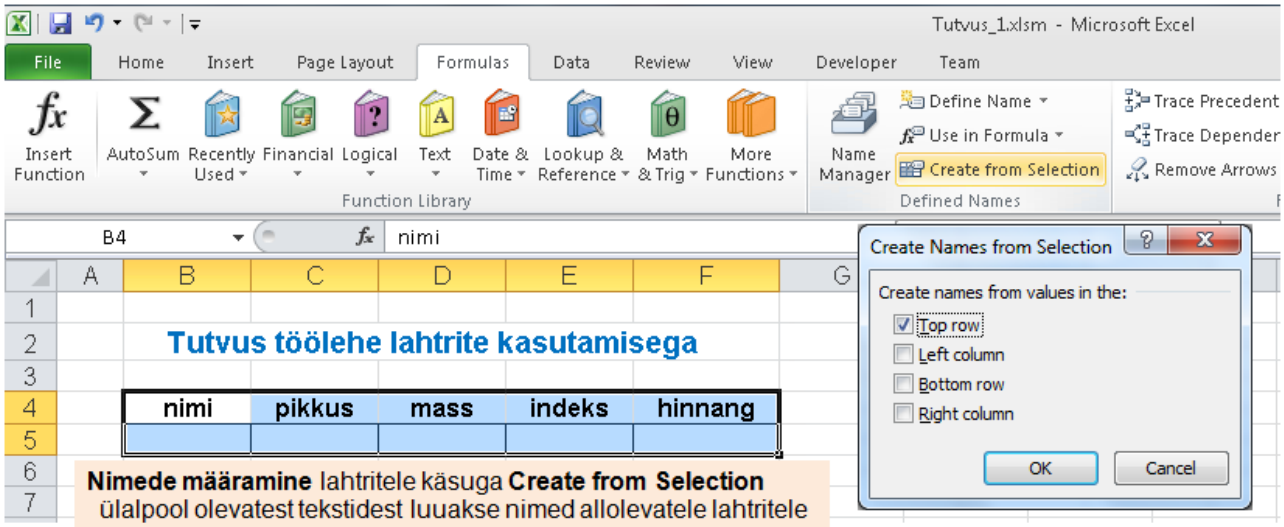
Nimede määramiseks lahtritele ja lahtriplokkidele on mitmeid võimalusi. Üheks lihtsamaks, kuid mitte alati kõige efektiivsemaks, on **nimeboksi** kasutamine.

Nimeboks (*Name Box*) asub vasakul valemiriba kõrval. Sisuliselt võiks seda nimetada nime ja aadressi boksiks, sest nime puudumisel kuvatakse selles lahtri aadress. Kui lahtril (või lahtriplokil) on nimi, kuvatakse nimi. Boksi saab kasutada ka nimede määramiseks. Nimede esitamise reeglid on peaaegu samasugused nagu muutujate puhul: **nimi** võib koosneda ühest tähest või tähtede, numbrite ja allkriipsude jadast, mis algab tähega, kuid on ka piiranguid – nimi ei tohi kokku langeda lahtri aadressiga nagu näiteks A1, X13, AS365 vmt. Nimedena ei tohi kasutada üksikuna tähti **c** ja **r**, sest need on kasutusel kiirtähisena aktiivse lahtri rea (*Row*) või veeru (*Column*) valimiseks.



Nimed saab määrata korraga ka mitmele lahtrile, kasutades nende kohal või kõrval olevaid tekste. Selleks on vahelehel *Formulas* (Valemid) grupis *Defined Names* (Määratletud nimed) käsk *Create from Selection* (Loo valikust). Eelnevalt tuleb valida sidus lahtriplokk: nimetatavad lahtrid ja lahtrid selgitavate tekstidega. Peale käsu sisestamist kuvatakse boks *Create Names from Selection* (Nimede loomine valikust), kus saab näidata (kinnitada) nimedeks kasutatavate tekstide asukoha.

Alloleval pildil on näidatud nimede määramine kõrvuti asuvatele lahtritele.



Vahelehel *Formulas* asub ka nupp ***Name Manager*** (Nimehaldur) vastava dialoogiboksi kuvamiseks, milles saab nimesid määrata ja täita ka muid tegevusi nendega (eemaldada, muuta, …).

Nimed võivad olla **globaalsed** või **lokaalsed**. Globaalsed nimed kehtivad terves töövihikus ning sama nime võib töövihikus olla ainult üks. Lokaalne nimi kehtib ainult ühel kindlal töölehel. Selliseid nimesid võib töövihikus olla ka mitu. Esimest korda nime määramisel on see vaikimisi globaalne.

Programm

Allpool olev näiteprogramm koosneb kahest protseduurist: peaprotseduur **Tutvus_2** ja funktsioon **saledus**, mis on põhimõtteliselt samad nagu eelmises variandis. Siin on toodud versioonis kasutatakse **If**-lauses **ElseIf**-osalauseid.

Kui lugeja tahab programmi katsetada, võiks selleks kasutada sama töövihikut, kus on eelmine variant. Programmi peaprotseduuri võiks salvestada selle lehe moodulis, kus asub kasutajaliides või üldmoodulis. Funktsiooni võiks paigutada üldmoodulisse kui see seal juba ei ole.

Peaprotseduuris on neli muutujat: **nimi**, **L**, **mass**, **indeks**. Protseduur loeb kõigepealt töölehelt kasutaja nime ning kuvab tervituse. Seejärel loeb töölehe lahtritest pikkuse ja massi väärtused ning omistab need muutujatele **L** ja **mass**. Omistamislause **indeks = mass / (L / 100) ^ 2** leiab kehamassiindeksi väärtuse ja salvestab selle muutujas **indeks**. Järgnev lause kirjutab väärtuse muutujast töölehe lahtrisse. Peaprotseduuri viimase lause paremas pooles on funktsiooniviit pöördumiseks funktsiooni **saledus** poole, mis tagastab hinnangu, mis väljastatakse otse töölehele.

NB! Muutujate nimed võivad, aga ei pea kokku langema lahtrite nimedega.

```
Sub Tutvus_2()
  ' Kehamassi indeks. Pealik
  Dim nimi, L, mass, indeks  ' muutujad
  nimi = Range("nimi").Value   ' loeb väärtuse lahtrist nimega nimi ja salvestab muutujas nimi
  MsgBox "Tere, " & nimi & "! Hakkan arvutama!"  ' kuvab teate dialoogiboksis
  L = Range("pikkus").Value   ' loeb väärtuse lahtrist pikkus ja salvestab muutujas L
  mass = Range("mass")       ' loeb lahtrist mass, omaduse Value võib jätta ära
  indeks = mass / (L / 100) ^ 2  ' avutatakse avaldise väärtus, tulemus omistatakse muutujale indeks
  Range("indeks").Value = indeks ' muutuja indeks väärtus kirjutatakse töölehe lahtrisse indeks
  Range("hinnang") = saledus(L, mass)  ' funktsiooni poolt leitav väärtus kirjutatakse lahtisse hinnang
End Sub
```

```
Function saledus (L, m):
  ' hinnang massiindeksi alusel
  ' L - pikkus, m - mass
  Dim indeks
  indeks = m / (L / 100) ^ 2
  If indeks < 18 Then
    saledus = "kõhn"
  ElseIf indeks <= 25 Then
    saledus = "normis"
  ElseIf indeks <= 30 Then
     saledus = "ülekaaluline"
  Else
      saledus = "suur ülekaal"
  End If
End Function
```

```
' Lugeja võiks proovida käivitada need protseduurid
Sub Vaheta(K1 As Shape, K2 As Shape)
  Dim x1, x2
  x1 = K1.Left:  x2 = K2.Left
  K1.Left = x2:  K2.Left = x1
  poorle K1:  pöörle K2
End Sub

Sub pöörle (kuju As Shape)
  Dim i
  For i = 1 To 360
    kuju.IncrementRotation 1
    DoEvents
  Next i
End Sub
```

Põhilisteks tegevusteks peaprotseduuris on väärtuste lugemine töölehelt ja tulemuste väljastamine (kirjutamine) töölehele. Rakenduste loomisel tabelarvutusprogrammide keskkonnas omab see olulist tähtsust, sest töölehed on mugavad ja lihtsalt hallatavad vahendid andmete salvestamiseks ja säilitamiseks. Lugemise ja kirjutamise põhivariandid (esinevad ka näites) on järgmised:

lugemine: muutuja = Range(“nimi“)[.Value]
kirjutamine: Range(“nimi“)[.Value] = avaldis

Formaalselt on tegemist lahtri omaduse *Value* (väärtus) lugemise ja muutmisega. Kuna *Value* on lahtri jaoks nn vaikimisi võetav omadus, võib selle ära jätta, sest nii saab programmi teksti lühendada.

Vt ka jaotist „Andmete lugemine töölehelt ja kirjutamine töölehele“.

Viiteid lahtritele (formaalselt lahtri omadusele *Value*) võib otse kasutada ka avaldistes:

```
Range("indeks").Value = Range("mass") / (Range("pikkus") / 100) ^ 2
```

Selline lause on pikem ja ebaülevaatlikum ning selle täitmine nõuab ka oluliselt rohkem aega. Praegu ei ole see oluline, kuid suurte andmehulkade korral peab sellega arvestama. Otstarbekas on kinni pidada järgmisest põhimõttest: andmete lugemine ja kirjutamine peab olema eraldatud andmete töötlemisest.

Tutvus. Variant 3 – töölehefunktsioonid

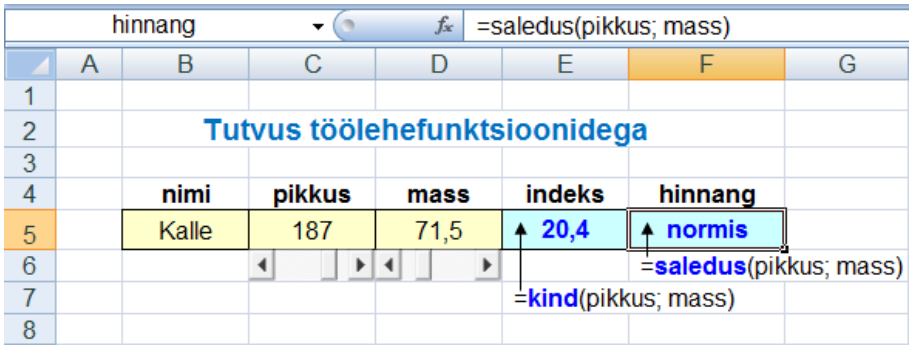
Selles variandis kasutame töölehefunktsioone – VB funktsioone –, mille poole pöördutakse töölehe lahtrites olevatest valemitest.

Olgu näiteks vaja leida kehamassi indeks ja hinnang selle väärtuse kohta. Hinnangu leidmiseks on olemas funktsioon ***saledus***. Indeksi leidmiseks varem eraldi funktsiooni ei olnud ning see tuleb koostada.

```
Function kind(L, m)
  kind = m / (L / 100) ^ 2
End Function
```

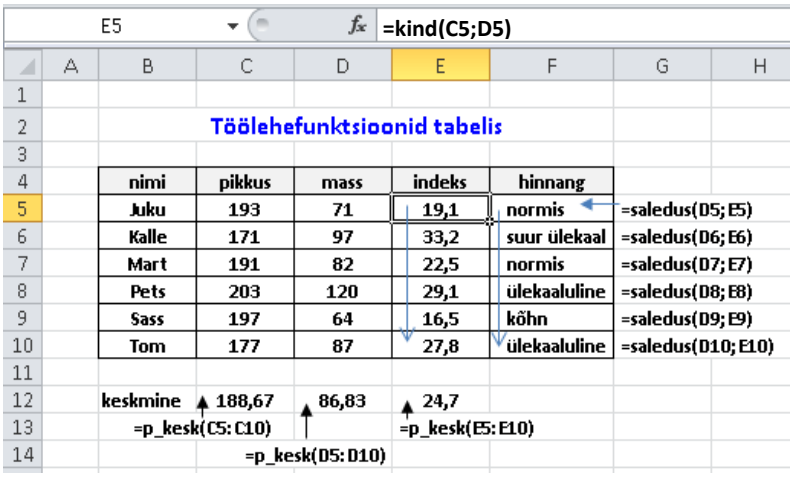
Tegemist on väga lihtsa üherealise, sisuga funktsiooniga. Vastavad arvutused võiks realiseerida ka Exceli valemiga.

NB! Et funktsiooni saaks kasutada töölehel, peab see olema salvestatud üldmoodulis. Eeldades, et lahtritele on määratud nimed, võib funktsioonide abil esitada arvutused nii nagu näidatud allpool.



Töölehefunktsioonide kasutamine tabelis

Töölehefunktsioone on mugav kasutada ka Exceli tabelis. Olgu vaja leida kehamassiindeks ja hinnang mitme isiku jaoks (näit meeskond, klass vmt). Lisaks on vaja leida pikkuste, masside ja indeksite aritmeetilised keskmised.




```
Function p_kesk (piirkond As Range)
  ' piirkonna keskmine
  Dim lahter, n, S
  n = piirkond.Cells.Count
  S = 0
  For Each lahter In piirkond
    S = S + lahter.Value
  Next lahter
  p_kesk = S / n
End Function
```

Lause **For Each** täitmisel korratakse **For** ja **Next** vahel olevaid lauseid antud kogumi (siin lahtriplokk ehk piirkond) iga elemendi jaoks.

Keskmise leidmiseks on funktsioon **p_kesk**, mida saab kasutada töölehelt. Selle peab salvestama üldmoodulisse. Ka Excelis on olemas funktsioon keskmise leidmiseks: AVERAGE(prk).

Viidad funktsioonidele **kind** ja **saledus** sisestatakse tabeli esimese rea (rivi) lahtritesse neli ja viis, kasutades argumentide esituseks aadresse. Valemid kopeeritakse allapoole, järgmistesse lahtritesse. Viidad funktsioonile **p_kesk** sisestatakse vastava veeru (tulba) alla, esitades argumendi lahtrite vahemikuna: C5:C10, D5:D10, E5:E10.

Makro kasutamine tabelis

Rakendusi Exceli keskkonnas on mõnikord otstarbekas realiseerida selliselt, et Exceli enda vahendeid (valemeid, funktsioone jmt) ei kasutata. Kõik töötlusega seotud põhitegevused täidetakse VBA protseduuride abil. Kasutatakse ainult töölehti andmete salvestamiseks ja säilitamiseks ning Exceli vormindamise ja kujundamise vahendeid.

Makro tabelis Arvuta

nimi	pikkus	mass	indeks	hinnang
Juku	193	71	19,1	**normis**
Kalle	171	97	33,2	**suur ülekaal**
Mart	191	82	22,5	**normis**
Pets	203	120	29,1	**ülekaaluline**
Sass	197	64	16,5	**kõhn**
Tom	177	87	27,8	**ülekaaluline**
keskmine	**188,67**	**86,83**	**24,7**	

Näites on toodud eelmises jaotises vaadeldud ülesanne kehamassi indeksi, saleduse ja mitme isiku keskmiste leidmiseks. Selleks kasutatakse funktsioone kind, saledus ja p_kesk, mis olid mängus eelmistes näidetes. Need käivitatakse makrost. Töölehel ei ole nüüd üldse valemeid, sest kõik arvutused teeb VBA makro.

Kasutaja sisestab nimed, pikkused ja massid tabelisse ja käivitab makro. Isikute arv (st ridade arv tabelis) võib ka olla suvaline.

Taolistes ülesannetes on olulised järgmised küsimused:

- kuidas programm teeb kindlaks tabeli asukoha ja mõõtmed: ridade ja veergude arv,
- kuidas viidata tabelile ja selle elementidele: lahtrid, read (rivid) ja veerud (tulbad).

Ülalnimetatud küsimuste lahendamisel on oluline koht lahtritele või lahtriplokkidele määratud nimedel. Programmi laused on lühemad, kui tabelile (lahtriplokile) saab viidata objektimuutuja abil, mis määratakse **Set**-lauses.

Näites eeldatakse, et nimi **andmed** on määratud tabeli kehale (ilma päiseta) ning tulpadele **pikkus**, **mass** ja **indeks**.

Tabeli sidumine muutujaga võimaldab viidata tabeli lahtritele indeksite abil:

muutuja(*rivi*, *tulp*)

kus *rivi* ja *tulp* on indeksid: rivi (rea) ja tulba (veeru) numbrid: 1, 2, …

```
Sub Statistika()
  Dim T As Range, m, i
  Set T = Range("andmed")
  m = T.Rows.Count
  For i = 1 To m
    T(i, 4) = kind(T(i, 2), T(i, 3))
    T(i, 5) = saledus(T(i, 2), T(i, 3))
  Next i
  T(m + 2, 1) = "keskmine"
  T(m + 2, 2) = p_kesk(Range("pikkus"))
  T(m + 2, 3) = p_kesk(Range("mass"))
  T(m + 2, 4) = p_kesk(Range("indeks"))
End Sub
```

Muutuja **T** tüübiks on **Range** ja sellega seotakse piirkond (lahtriplokk) nimega **andmed**. Kasutades omadusi **Rows** ja **Count** tehakse kindlaks rivide arv (**m**) tabeli kehas.

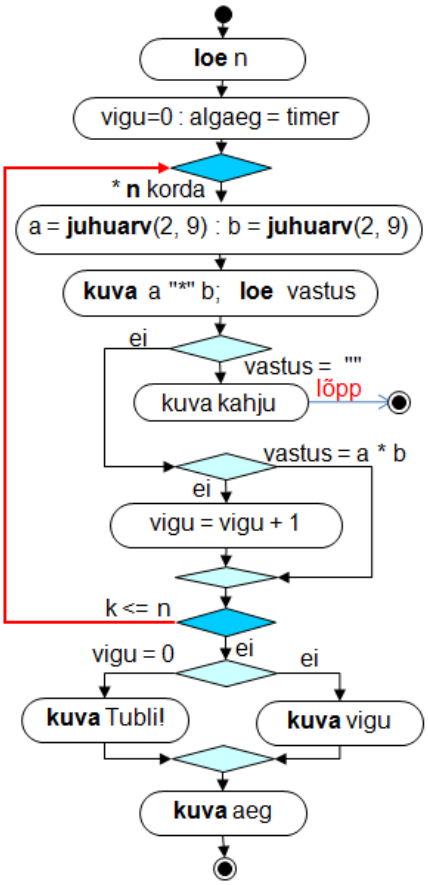
For-lause määrab korduse: **i** väärtust (rea järjenumbrit) muudetakse 1 kuni **n** ning iga **i** väärtuse korral täidetakse **For** ja **Next** vahel olevad laused.

T(m+2, *tulp***)** määrab lahtri, mis asub tulbas ***tulp*** (1, 2, 3, 4) ja rivis **m+2**. Tabeli all jäetakse tühjaks üks rida.

Funktsioon **p_kesk**, kasutades argumendina tulba (lahtriploki) nime, leiab selle keskmise.

Näide. Korrutamistabel. For-lause

Programm (makro) koosneb ühest **Sub**-protseduurist ning imiteerib korrutamise trenni. Kõigepealt küsitakse ülesannete arvu **n**, seejärel esitatakse **n** korrutamisülesannet ning hinnatakse vastuseid ja tehakse kindlaks valede vastuste arv. Programm fikseerib ka aja.




```
Sub Korrutustabel()    ' Korrutamise harjutamine
 Dim n, vigu, a, b, k, _
       vastus, algaeg, aeg
 n = InputBox("Mitu ülesannet? ")
 vigu = 0
 Randomize
 algaeg = Timer()  ' jooksev arvutiaeg
 For k = 1 To n     ' korduse algus
   a = 2 + Round(7 * Rnd())
   b = 2 + Round(7 * Rnd())
   vastus = InputBox(a & " * " & b)
   If vastus = "" Then  ' kui tühi väärtus
      MsgBox "Kahju, et loobusid!"
      Exit Sub   ' katkestab korduse
   End If
   If Int(vastus) <> a * b Then
      MsgBox "Vale!"
      vigu = vigu + 1
   End If
 Next k
 If vigu = 0 Then
   MsgBox "Tubli! Kõik oli õige!"
 Else
    MsgBox ("Vigu oli " & vigu)
 End If
 aeg = Round(Timer() - algaeg, 2)
 MsgBox ("Aeg: " & aeg & " sek")
End Sub
```

Programmis on kasutusel muutujad:

- **n** – ülesannete arv,
- **a**, **b** – juhuslikud arvud vahemikus 2 kuni 9 küsimuste esitamiseks,
- **vastus** – kasutaja poolt pakutav vastus,
- **vigu** – vigade arv,
- **k** – juhtmuutuja **For**-lauses,
- **aeg** – ülesandele kulunud aeg.

Korduse kirjeldamiseks kasutatakse programmis nn juhtmuutujaga kordust ehk **For**-lauset, mille enam-kasutatav variant on järgmine:

For *muutuja* = *algväärtus* **To** *lõppväärtus*
    laused
Next *muutuja*

muutujat nimetatakse siin juhtmuutujaks. Täitmisel muudetakse järjest juhtmuutujat algväärtusest kuni lõppväärtuseni (sammuga üks) ja iga juhtmuutuja väärtuse korral täidetakse **For** ja **Next** vahel olevad laused.

See kordus on oma olemuselt üsna sarnane Scratchi plokile [**korda** *n*], kuid viimasest veidi üldisem. Scratchi plokist erineb **For**-lause peamiselt automaatselt muudetava **juhtmuutuja** olemasolu poolest. Antud programmis juhtmuutujat **k** küll ei kasutata, kuid süntaksi reeglite tõttu peab juhtmuutuja alati olema. Töös tabelite ja massiividega kasutatakse juhtmuutujat sageli järjenumbrina (rea või veeru numbrina).

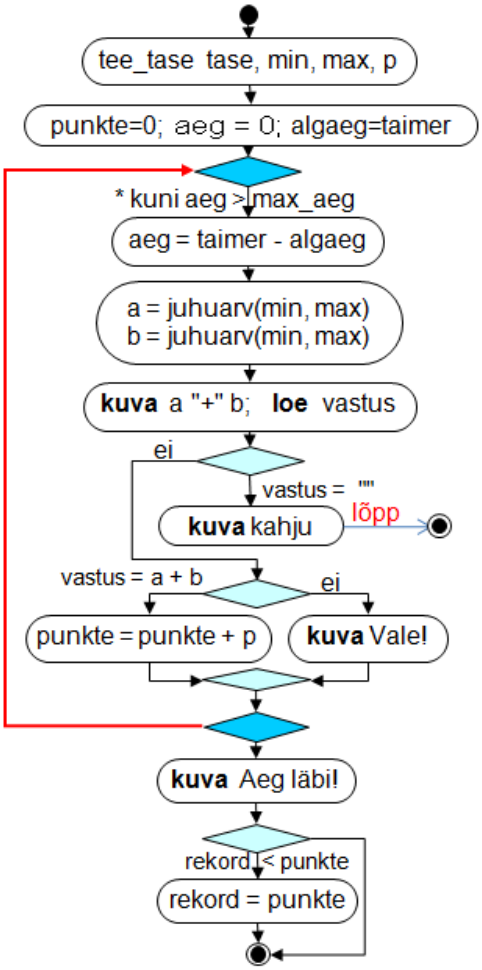
Kuna antud programmi kasutajaliides põhineb dialoogibokside kasutamisel ning ei pruugita Exceli vahendeid, saab selle panna tööle ka teistes rakendusprogrammides, mis toetavad Visual Basicut.

Näide Pranglimine. Do Until-lause

See programm võimaldab harjutada liitmist.

Programm esitab etteantud aja jooksul (praegu 45 sek) liitmise ülesandeid ja kontrollib vastuseid, andes õigete vastuste eest punkte. Kasutaja saab valida kolme taseme vahel, millest sõltub arvude suurus ja ühe tehte punktide arv (vt protseduuri **Tee_tase**).

Töölehel on lahtrid nimedega: **tase**, **punkte**, **aeg** ja **rekord**. Programmi muutujad on loetletud **Dim**-lauses. Töölehel on ka väsimatud abilised – **Juku** ja **Kraps**, kes raiskavad veidi aega karistuseks vale vastuse eest.




```
Sub Prangi_1()
  ' Liitmise trenn
  Const max_aeg = 45   ' maksimaalne aeg
  Dim algaeg, a, b, vastus, mini, maxi, p, punkte
  Range("punkte") = ""
  Tee_tase Range("tase"), mini, maxi, p
  Randomize
  algaeg = Timer():  Range("aeg") = 0
  Do Until Range("aeg") > max_aeg
     Range("aeg") = Timer() - algaeg
     a = mini + Int(Rnd() * (maxi - mini + 1))
     b = mini + Int(Rnd() * (maxi - mini + 1))
     vastus = InputBox(a & " + " & b)
     If vastus = "" Then MsgBox "Kahju!": End
     If Int(vastus) = a + b Then
        Range("punkte") = Range("punkte") + p
     Else
        MsgBox "Vale!": Call Ai_ai
     End If
  Loop
  MsgBox "Aeg on läbi "
  If Range("punkte") > Range("rekord") Then
    Range("rekord") = Range("punkte")
    MsgBox "Õnnitlen! Uus rekord!"
  End If
End Sub
```

```
Sub pöörle(kuju As Shape)
   Dim i
   For i = 1 To 360
      kuju.IncrementRotation 1
      DoEvents
   Next i
End Sub
```

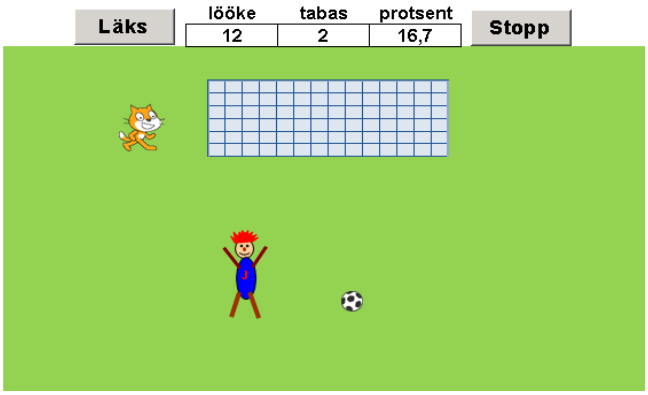
```
Sub Tee_tase(tase, mini, maxi, p)
  If tase = 1 Then
     mini = 1: maxi = 10: p = 2
  ElseIf tase = 2 Then
      mini = 10: maxi = 20: p = 3
  Else
     mini = 20: maxi = 30: p = 5
  End If
End Sub
```

```
Sub Ai_ai()
  pöörle Shapes("Juku")
  pöörle Shapes("Kraps")
End Sub
```

Põhiosa peaprotseduurist moodustab kordus, mida juhitakse **Do Until** *tingimus* … **Loop**-lause abil. Lauseid korduses täidetakse seni, kuni *tingimus* (jooksev aeg on suurem kui max_aeg) saab tõeseks.

Näide Jalka

Rakendus, milles on tegemist peamiselt objektidega – kujundid (graafikaobjektid) ja töölehe lahtrid. Võrdluseks on toodud ka Scratchi programm.



Rakenduses imiteeritakse jalgpallitrenni. Piltidel on toodud VBA (vasakul) ja Scratchi kasutajaliidesed. Programmi põhiobjektideks on **Juku** (pealelööja rollis) ja **pall**. Lisaks on kasutusel suhteliselt passiivne objekt **värav** – ruuduline ristkülik. Programmis kontrollitakse **palli** sattumisi **värava** piirkonda. Töölehel on „vaatlejana“ ka **Kraps**, kes on Scratchi rakenduses pealelööja rollis. Tulemuste fikseerimiseks kasutatakse töölehe lahtreid nimedega **lööke**, **tabas** ja **protsent**. Scratchis on samade nimedega muutujad.

```
Sub Trenn()
  Dim pall As Shape, J As Shape 'objektimuutujad
  Set pall = Shapes("pall") ' objektide sidumine
  Set J = Shapes("Juku")  'muutujatega, sama
  Range("lööke") = 0: Range("tabas") = 0 ' algväärtused
  Randomize  ' juhuarvude generaator algseisu
  Do  ' lõputu korduse algus
     pall.Left = Rnd() * 400          ' palli vasak serv
     pall.Top = 150 + Rnd() * 150   ' palli ülemine serv
     Call Hyppa(pall, 4, 30)  ' kutsub protseduuri Hyppa
     J.Left = pall.Left : J.Top = pall.Top ' Juku palli juurde
     paus 0.5
     pall.Left = Rnd * 400 : pall.Top = Rnd * 150 ' löök
     Range("lööke") = Range("lööke") + 1
     If On_sees(pall, Shapes("värav")) Then
        Range(“tabas“) = Range(“tabas“)+ 1
        Call Hyppa(J, 2, 40)  ' kutsub protseduuri Hyppa
        Call Hyppa(Shapes("Kraps"), 3, 20)
     End If
     [protsent] = [tabas] / [lööke] * 100  ' [ ]=Range( )
     paus 2
  Loop  ' korduse lõpp
End Sub
```

Programmis on realiseeritud järgmine stsenaarium (algoritm). Käivitamisel nupuga **Läks** korratakse lõputult järgmisi tegevusi. Pall viiakse väljaku alumisse poolde, kus see teeb mõned hüpped. Juku „jookseb“ palli juurde, lühikesele pausile järgneb „löök“ – pall viiakse väljaku ülemisse poolde ning juhuk kui pall sattus väravasse, suurendatakse tabamuste arvu ning Juku ja Kraps teevad mõned hüpped. Arvutatakse tabamuste protsent ning kõik kordub. Programmi töö saab katkestada nupuga **Stopp**.

Palli asukoha muutmiseks kasutatakse selle omadusi **Left** ja **Top**. Nende muutmiseks kasutatakse juhuarve. VBAs peab kasutama otsest viitamist objektidele ning nende omadustele ja meetoditele. Scratchis kuulub iga skript kindlale objektile (spraidile) ja saab määrata ainult selle tegevusi. VBA protseduur aga on objektidest sõltumatu ja saab määrata suvaliste objektide tegevusi ja muuta nende omadusi.

VBA programm koosneb peaprotseduurist **Trenn** ja kolmest alamprotseduurist: **Hyppa**, **paus** ja **On_sees**.

Peaprotseduuris on **Dim**-lausega määrtletud muutujad **pall** ja **J**. Tegemist on **objektimuutujatega** (**viitmuutujatega**), mida kasutatakse viitamiseks objektidele **pall** ja **Juku**. Objekt seotakse muutujaga **Set**-lausega. Standardne viitamisviis graafikaobjektidele on **Shapes(“***nimi***“).**

Lausetega Range("lööke") = 0: Range("tabas") = 0 antakse töölehe lahtritele nimedega **lööke** ja **tabas** väärtus null. Standardne viitamisviis nimedega lahtritele on Range ("nimi"). Alternatiivina võib kasutada ka konstruktsiooni **[***nimi***]** (vt lause [protsent] = [tabas] / [lööke] * 100 protseduuri lõpus). Sel juhul ei tohi lahtri nimi kokku langeda muutuja nimega samas protseduuris.

Protseduuri põhiosa moodustab liitlausega määratletud **Do**…**Loop** lõputu kordus. Palli viimiseks platsi alumisse poolde kasutatakse lauseid:

```
pall.Left = Rnd() * 400          ' palli vasak serv juhuarv 0-st kuni 400-ni
pall.Top = 150 + Rnd() * 150  ' palli ülemine serv 150-st kuni 300-ni.
```

Mõõtühikuks on point (punkt – 1/72 tolli ehk umbes 0,35 mm). Objekti omaduse muutmiseks kasutatakse omistamislauset kujul:

```
objekt.omadus = avaldis
```

Objekt on esitatud muutuja abil (pall), alternatiivina võiks olla Shapes(“pall“). Omadused esitatakse nimedega. VB sisefunktsioon Rnd() tagastab juhusliku reaalarvu vahemikus 0 kuni 1. Tühjad sulud funktsiooni nime järel võib ka ära jätta.

Palli viimine platsi ülemisse poolde toimub omaduste ja juhuarvude abil analoogselt:

```
pall.Left = Rnd * 400 : pall.Top = Rnd * 150    ' löök, palli  ülemine serv 0-st 150-ni
```

NB! Ühel real võib olla mitu lauset. Eraldajaks on koolon.

Juku viimiseks palli juurde võrdsustatakse vastavad omadused: J.Left = pall.Left : J.Top = pall.Top

```
Function On_sees(O_1, O_2)
' Tõene, kui objekt O_1 on O_2 sees
 Dim v1, p1, y1, a1, v2, p2, y2, a2
  v1 = O_1.Left: p1 = v1 + O_1.Width
  y1 = O_1.Top: a1 = y1 + O_1.Height
  v2 = O_2.Left: p2 = v2 + O_2.Width
  y2 = O_2.Top: a2 = y2 + O_2.Height
  If v1 > v2 And p1 < p2 And _
     y1 > y2 And a1 < a2 Then
   On_sees = True
  Else
   On_sees = False
  End If
End Function
```

Peale palli viimist platsi ülemisse poolde suurendatakse löökide arvu ja kontrollitakse, kas pall on värava sees või mitte. Selleks kasutatakse abifunktsiooni **On_sees**(O_1, O_2), mis tagastab väärtuse ***True*** (tõene), kui objekt O_1 on täielikult objekt O_2 sees. Oma olemuselt vastab funktsioon Scratchi plokile <**puudutab** *objekt*>, ainult siin peab O_1 olema täielikult O_2 sees.

Protseduur illustreerib funktsioonide kirjeldamise põhimõtteid. VBA funktsioon saab leida ja tagastada ühe ja ainult ühe väärtuse. Tagastatav väärtus omistatakse funktsiooni nimele. Funktsioonil on tüüpiliselt olemas parameetrid, mis esindavad sisendandmeid. Viimased saavad väärtused pöördumisel vastavatelt argumentidelt.

Pauside tekitamiseks kasutatakse abiprotseduuri paus, mille poole saab pöörduda lausega:

paus *pikkus*

pikkus – pausi pikkus sekundites. Protseduur vastab täielikult Scratchi käsule [**oota**].

```
Sub paus(pp)
' paus pikkusega pp sek
  Dim pl  ' pausi lõpp
  pl = Timer() + pp
  Do Until Timer() > pl
    DoEvents
  Loop
End Sub
```

Protseduur esindab parameetritega protseduuri. Parameeter ***pp*** saab väärtuse pöördumisel vastavalt argumendilt, mis annab konkreetse pausi pikkuse sekundites.

Muutuja **pl** väärtuseks võetakse pausi lõpu jaoks väljaarvutatud aeg. VBA sisefunktsioon **Timer**(), mis on Scratchi ploki [**taimer**] analoog, annab arvuti jooksva kellaaja. Lause **Do Until** … **Loop** määrab tingimusliku korduse, mis vastab [**korda kuni** ...] plokile Scratchis. Igal kordamisel täidetakse lause **DoEvents**, mis võimaldab Windowsil pausi ajal täita ootel olevad tegevused. Kordus lõpeb, kui aeg ületab pausi lõpuaja **pl**.

Parameetritega protseduuri näiteks on ka protseduur **Hyppa**, mille abil imiteeritakse objektide, milleks siin on **Juku**, **Kraps** ja **pall**, hüppamist – objekt tõuseb korduvalt (**n** korda) ülespoole ja laskub alla.

```
Sub Hyppa(kuju, n, h)
' objekt kuju hüppab
  Dim i
  For i = 1 To n
    kuju.IncrementTop  -h
    paus 0.2
    kuju.IncrementTop h
    paus 0.3
   Next i
End Sub
```

Protseduuril on kolm sisendparameetrit: **kuju** – graafikaobjekt, **n** – hüpete (korduste) arv, **h** – hüppe kõrgus. Parameetrid saavad väärtused vastavatelt argumentidelt. Protseduuri poole pöördumine toimub siin kolm korda lausetega:

Call Hyppa(pall, 4, 30) : **Call** Hyppa(J, 2, 40) : **Call** Hyppa(Shapes("Kraps"), 3, 20)

Esimesel korral on esimeseks argumendiks (vastab parameetrile *kuju*) **pall**, teisel **Juku**, kolmandal **Kraps**. Kuna objekt **Kraps** ei ole seotud muutujaga, kasutatakse viitamiseks konstruktsiooni Shapes("Kraps"). Igal pöördumisel on ka parameetritele **n** ja **h** vastavatel argumentidel erinevad väärtused.

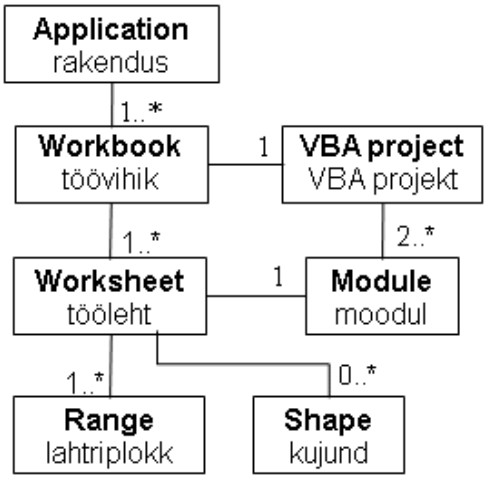
Parameetritega protseduuri poole saab pöörduda korduvalt, kasutades erinevaid argumente. See võimaldab vähendada programmi mahtu.

Scratchi praeguses versioonis puudub võimalus parameetrite ja argumentide kasutamiseks. Sellepärast peab moodustama (kopeerima) kolm peaaegu ühesugust skripti vastavate spraitide jaoks. Näidatud on neist kaks – palli ja Krapsu skriptid. Väikesed erinevused (korduste arv, hüppe kõrgus ja pausi pikkus) pannakse paika projekti loomise ajal.

Exceli objektide kasutamine VBAs

Rakenduse struktuur ja põhiobjektid

Exceli rakendus kujutab endast omavahel seotud objektide kogumit: töövihikud, töölehed, lahtriplokid, graafilised kujundid jms. Ühetüübilised objektid kuuluvad ühte klassi. Igal klassil on kindel nimi: Workbook, Worksheet, Range, Shape jne. VBA toega Exceli rakenduses võib olla kasutusel mitu töövihikut. Töövihik võib koosneda ühest või mitmest töölehest. Töölehti saab lisada ja eemaldada nii „käsitsi“ Exceli vastavate käskudega kui ka VBA programmi käskude abil. Igal töölehel on üle 15 miljardi lahtri (!), millest saab moodustada praktiliselt piiramatu hulga lahtriplokke. Lahtriplokina on käsitletav suvaline lahtrite kogum. Töölehe pinnale saab paigutada praktiliselt piiramatu hulga graafikaobjekte ehk kujundeid. Neid saab joonestada Exceli juurde kuuluvate joonestusvahenditega ja importida graafikafailidest. Tegevusi graafikaobjektidega saab täita „käsitsi“ ja programmi käskudega.



Klasse on Excelis tunduvalt rohkem, kui skeemil tooduid. Näiteks võib lisaks töölehtedele kasutada diagrammilehti. Diagramme saab paigutada ka töölehe pinnale, töölehele saab panna ka mitmesuguseid ohjureid (juhtelemente) – kerimisribad, märkeruudud, komboboksid jms. Siin puutume vähemalt esialgu nö programmilisel tasemel kokku ainult graafiliste kujunditega, lahtriplokkide ja ohjuritega. Piirdume ainult rakendustega, kus on kasutusel üks töövihik. Selles võib olla praktiliselt piiramatu hulk töölehti ning igal töölehel võib olla omaette rakendus. Vaatamata nimetatud piirangutele, saab luua väga sisukaid ja mahukaid rakendusi. Kui põhioskused omandatud, saab vajaduse korral laiendada kasutatavaid vahendeid ja tegevusvaldkonda.

Scratchi skriptid on alati seotud kindla objektiga (spraidi või lavaga) ning seega pole vajadust ega isegi võimalust neile otseselt viidata. VBA protseduur aga ei ole seotud konkreetse objektiga ning üks protseduur saab määrata tegevusi mitme objektiga ning seetõttu peab alati näitama objekti, millega vastav tegevus täidetakse. Viitamiseks objektidele kasutatakse klassi nimesid, kinni pidades teatud reeglitest.

Objektidel on kindel valik **omadusi** ja **meetodeid**. Ühe klassi objektidel on sama valik. Igal omadusel ja meetodil on kindel nimi, mida kasutatakse neile viitamiseks. Viitamisel kasutatakse erinevates programmeerimissüsteemides laialt levinud viisi, kus omadused ja meetodid seotakse objektiga **punkti** abil:

objekt.**omadus** ja *objekt*.**meetod** [*argumendid*]

Objekt on viit objektile. Esitus sõltub objekti klassist ja viitamisviisist. Omadused ja meetodid esitatakse nimede abil. Mõnedel meetoditel võivad olla ka argumendid.

Mõned näited:

Shapes(“auto“).**Left** = 0	kujundi **auto** omaduse **Left** väärtuseks võetakse 0.
Shapes(“pall“).**IncrementTop** 20	meetod muudab kujundi ülemise serva koordinaati 20 võrra.
Range(“punkte“). **Value** = 0	lahtri **punkte** väärtuseks võetakse 0.

Mõnikord kasutatakse ka alamobjekte ja nende omadusi. Näiteks graafilise kujundi värvuse määramiseks kasutatakse järgmist konstruktsiooni: Shapes(“ufo“).**Fill.ForeColor.SchemeColor** = 3.

Täpsemalt vaadeldakse neid küsimusi vastava klassi objektide kirjeldamisel.

Tööleht – klass *Worksheet*

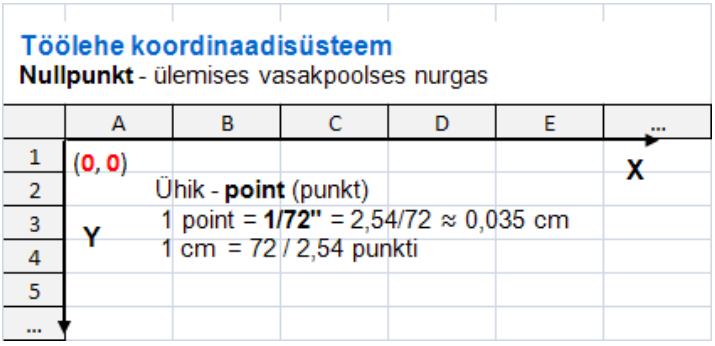
VBA programmide abil saab määrata tegevusi Exceli töölehtedega ja ka töövihikutega, mille koosseisu lehed kuuluvad, kuid neid võimalusi praegu eriti põhjalikult ei vaadelda. Töölehti ja töövihikuid kasutatakse antud juhul peamiselt VBA rakenduste konteineritena (hoidlatena) ning kasutajaliideste komponentidena. Töölehele paigutatakse graafikaobjektid, millega manipuleerivad programmid ning töölehe lahtreid kasutatakse programmide poolt kasutatavate andmete salvestamiseks ja säilitamiseks, aga ka kasutajaliideste loomisel.

Tööleht
nimi ridade arv veergude arv laius, kõrgus aktiivne lahter
aktiveerimine() teisaldamine() kopeerimine() eemaldamine() peitmine() ...

Protseduurid salvestatakse töölehe moodulisse või üldmoodulisse. Viimasel juhul eeldatakse, et programmi töö ajal on aktiivne kasutajaliidesega tööleht, kus asuvad VBA protseduuride poolt kasutatavad objektid. Ühes töövihikus võib olla realiseeritud mitu sõltumatut rakendust. Sellisel juhul peaks need asuma erinevatel lehtedel, kuigi põhimõtteliselt ei ole välistatud ka ühise lehe kasutamine. Salvestades töövihiku välismällu, saame säilitada nii rakenduse töö tulemused kui ka programmid.

Siin ei vaadelda ka töölehe omaduste (näiteks nime) muutmist programmide abil. Seda saab teha lihtsalt klõpsates lehe lipikut (lehesakki) ja tippides vajaliku nime. Praktiliselt ei pea arvestama isegi töölehe selliste omadustega nagu ridade ja veergude arv ning lehe mõõtmed, sest need on meie lihtsate rakenduste jaoks meeletult suured (ridu üle miljoni, veerge ca 16 tuhat). Töölehe virtuaalne laius on ca 300 meetrit ja kõrgus lausa 5 kilomeetrit (!). Võrrelge kasvõi Scratchi lavaga.

Olulised on **töölehe koorinaadisüsteem** ja **mõõtühikud**, eriti töötamisel graafikaobjektidega. Andmed nende kohta on toodud pildil. Koordinaatsüsteemi nullpunkt on töölehe ülemises vasakpoolses nurgas. Horisontaaltelg (X) on suunatud paremale, vertikaaltelg (Y) alla. Pikkuse ühikuks on trükindusest pärit üksus **point** (punkt), mis võrdub 1/72 tolliga ehk ca 0,35 millimeetriga.



Erinevalt Scratchist, saab VBA protseduuridega teha ka nö mõõtkavas jooniseid ja skeeme, kasutades vajadusel ka vastavaid mõõtühikute (ja koordinaatide) teisendusi.

Mõnikord on vaja viidata VBA programmis töölehele, selleks võib kasutada järgmisi konstruktsioone:

Sheets(*nimi***)** või **Worksheets(***nimi***)**, mis on samaväärsed.

Nimi esitatakse tüüpiliselt stringkonstandina: Sheets(“*Plats*“), Worksheets(“*rada_1*“).

Teatud määral tutvume siin **töölehe sündmustega**, mida saab lihtsalt kasutada paindlike ja dünaamiliste rakenduste loomiseks. Objektisündmuste käsitlus erinevates programmeerimissüsteemides on üsna ühesugune.

Graafilised kujundid – klass Shape

Viitamine objektidele

Graafikaobjekte saab töölehele lisada Exceli käskudega vahelehe ***Insert*** (Lisa) grupist ***Illustrations*** (illustratsioonid). Käsk ***Shapes*** (kujundid) võimaldab lisada baaskujundeid: sirgjoone lõik, murdjoon, ristkülik, hulknurk, ovaal jm. Kujunditest saab moodustada liitkujundeid, redigeerida ja vormindada neid. Käsk **Picture** (Pilt) võimaldab importida kujundeid graafikafailidest. Graafikaobjekte saab lisada, redigeerida ja vormindada ka VBA käskudega.

Kõik töölehel paiknevad objektid: *MS Drawing* abil tehtud joonised, imporditud pildid ja teiste rakenduste objektid, hõlmatud diagrammid, ohjurid ehk juhtelemendid (käsunupud, kerimisribad, märkeruudud jm), kuuluvad klassi **Shape**. Neile viitamiseks võib kasutada erinevaid viise. Üks esmaseid ja peamisi on:

Shapes (*nimi***)**,

kus **nimi** on objekti (kujundi) nimi. Tüüpiliselt esitatakse nimi tekstikonstandina, mis paigutatakse jutumärkide vahele. Nimi on kujundil alati olemas. Excel moodustab nime kujundi loomisel või importimisel. Vaikimisi on nime kuju ***tüüp number***, kus tüüp sõltub kujundi tüübist (alamklassist) ja number on tüübist sõltumatu järjenumber, mis suureneb automaatselt iga uue kujundi lisamisel. Mõned näited:
Rectangle 1, Rectangle 13, Oval 4, Straight Connector 18, Picture 7.

Aktiivse kujundi nime võib näha **nimeboksis**. Viimast saab kasutada ka nime muutmiseks. Selleks peab muutma kujundi aktiivseks (klõpsates seda), tippima nimeboksis nime ja vajutama klahvile Enter. Omapoolse nime määramine on sageli otstarbekas, sest süsteemi poolt pandud nimed on tüüpiliselt üsna pikad ja ei ütle midagi objekti olemuse kohta antud rakenduses. Näiteks sirgjoone lõiku nimetakse *Straight Connector* (sirgkonnektor, Exceli varasemates versioonides oli *Line*).

Kujunditele viitamise näited:
Shapes(“Rectangle13”), Shapes (“ring”), Shapes (“pall”), Shapes (“Juku”)

Kui protseduur ei asu selle töölehe moodulis, millel on kujund, peab viidale kujundile eelnema ka viit lehele. Seda peab näiteks kasutama siis, kui protseduur asub üldmoodulis.
Sheets (“Mäng”).Shapes(“pall”), ActiveSheet.Shapes(“Juku”)

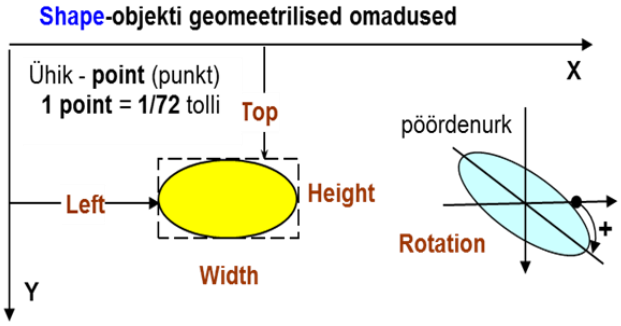
Esimesel juhul viidatakse konkreetsele lehele, kasutades selle nime. Teisel juhul viidatakse aktiivsele lehele, kasutades rakenduse omadust **ActiveSheet**.

Objekti saab **Set**-lause abil siduda muutujaga. See võimaldab lühendada viitu. **Set**-lause üldkuju on
Set *muutuja* = *objekt*

Mõned näited: **Set** *auto* = Shapes("auto") : **Set** *J* = Shapes("Juku").

Shape-objekti põhiomadused

Name	nimi
Left	vasak serv
Top	ülemine serv
Width	laius
Height	kõrgus
Rotation	pöördenurk
Visible	nähtavus
Fill.ForeColor	esiplaani värvus



Kujundi täitevärvuse muutmiseks võib kasutada järgmist omaduse määrangut:

objekt.**Fill.ForeColor.SchemeColor** = värvi number

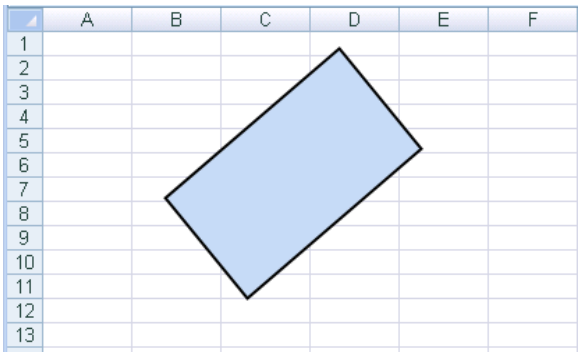
Värvikoodid on järgmised: **0** – must, **1** – valge, **2** – punane, **3** – roheline, **4** – sinine jne kuni 80.
Proovige teha protseduur, mille abil saab näha värvikoode.

Shape-objekti põhimeetodid

IncrementLeft *dx*	vasaku serva juurdekasv	**Select**	valimine (aktiveerimine)
IncrementTop *dy*	ülemise serva juurdekasv	**Cut**	lõikamine
IncrementRotation *dn*	pöördenurga juurdekasv	**Copy**	kopeerimine
ScaleHeight *k, False*	kõrguse skaleerimine	**Delete**	eemaldamine
ScaleWidth *k, False*	laiuse skaleerimine		

Näiteid

Objekti omaduste muutmine. Töölehe suvalises kohas asub graafikaobjekt nimega **kast**. Protseduur viib kujundi punkti (0; 0) ja annab sellele kindlad mõõtmed (100 * 50 punkti).



```
Sub Koju()
  Shapes("kast").Rotation = 0
  Shapes("kast").Left = 0
  Shapes("kast").Top = 0
  Shapes("kast").Width = 100
  Shapes("kast").Height = 50
  Shapes("kast").Fill.ForeColor.SchemeColor = 0
End Sub
```

Objekti sidumine muutujaga. **Meetodite kasutamine**. Töölehe suvalises kohas asub graafikaobjekt nimega **kast**. Protseduuri igakordsel täitmisel teeb kujund ühe sammu paremale ja alla, pöördub ning muudab suurust ja värvust. Koordinaatide ja pöördenurga muutused ning skaleerimise teguri võib valida suvaliselt. Värvuse määramiseks kasutatakse juhuarve. Et vähendada programmi teksti pikkust, on objekt seotud muutujaga.

```
Sub Liigu_1()
  Dim K As Shape
  Set K = Shapes("kast")
  K.IncrementLeft 10
  K.IncrementTop 5
  K.IncrementRotation 15
  K.ScaleWidth 1.03, False
  K.ScaleHeight 1.03, False
  K.Fill.ForeColor.SchemeColor = Rnd() * 80
End Sub
```

Muutuja **K** on seotud kujundiga **kast**. Tänu millele lüheneb lausete pikkus. Võrdle eelmise näitega. Kuna muutuja **K** on deklareeritud tüübiga **Shape**, pakub süsteem sisestamisel selle objekti omaduste ja meetodite loetelu (spikrit), kui objektimuutuja järel sisestatakse punkt.

Tegevuste määramisel kasutatakse peamiselt meetodeid. Esimesel kolmel juhul võib kasutada ka omadusi.

Näiteks: K.IncrementLeft 10 => K.Left = K.Left + 10

Kujundi sujuv liikumine. Protseduur **Liigu_XY** viib antud kujundi sujuvalt punkti koordinaatidega x, y. Protseduur on analoogne Scratchi käsuplokile [**liigu** t **sek** x: … y: …] .

```
Sub Liigu_XY(kuju As Shape, x, y, d, p)
  Dim xj, yj, L, dx, dy, i
  xj = kuju.Left:  yj = kuju.Top
  L = Sqr((x - xj) ^ 2 + (y - yj) ^ 2)
  If L = 0 Then Exit Sub
  dx = d * (x - xj) / L: dy = d * (y - yj) / L
  For i = 1 To L / d
    kuju.IncrementLeft dx
    kuju.IncrementTop dy
    paus p
  Next i
End Sub
```

Parameetrid: **kuju** – graafiline kujund (klass Shape), **x**, **y** – sihtkoha koordinaadid, **d** – liikumise samm, **p** – pausi pikkus sammude vahel. Kahele viimasele parameetrile vastavate argumentide väärtustega saab reguleerida liikumise kiirust.

Muutujad: **xj**, **yj** – kujundi algkoordinaadid, **L** – kaugus sihtkohast, **dx**, **dy** - sammu projektsioonid, **i** – abimuutuja.

Protseduur leiab kujundi kauguse sihtkohast **L** ning sammu projektsioonid **dx** ja **dy**. Liikumist juhitakse **For**-lause abil, kus juhtmuutuja lõppväärtuseks on **L/d**.

Antud protseduuri abil võiks näiteks muuta sujuvaks **palli** ja **Juku** liikumised näites „Jalka“.

Lahtrid ja lahtriplokid – klass Range

Lahtriplokina ehk Range-objektina võib käsitleda suvalist töölehe lahtrite kogumit. Üksiklahter kujutab endast lahtriploki erijuhtu: ühest lahtrist koosnev lahtriplokk. Esialgu piirdumegi peamiselt üksikute lahtrite kasutamisega. Viitamiseks lahtrile on mitmeid võimalusi, põhivariant on järgmine:

Range (*nimi*),

kus ***nimi*** on lahtri nimi, mis tüüpiliselt esitatakse stringkonstandina. Nimi määratakse lahtrile tavaliselt Exceli vahenditega, kuid seda saab teha ka VBA protseduuri abil. Mõned viitamise näited:

Range (“*a*”), Range (“*pindala*”), Range(“*aeg*”), Range (“*lööke*“).

Alternatiivina võib kasutada konstruktsiooni [***nimi***], kus lahtriploki nimi (ilma jutumärkideta) paigutatakse nurksulgude vahele: [*a*], [*pindala*], [*aeg*], [*lööke*] (protseduuris ei tohi sel juhul olla samanimelisi mujutujaid) ning töölehe omadust Cells(rida, veerg) või Cells(nr)

Kui on vaja viidata teise lehe lahtritele, peab viidale lahtrile eelnema viit lehele, näiteks:

Sheets(“Hinnakirjad“).Range(“Arvutid“), Sheets(“Maksud“).Range(“tulumaks“)

Lahtritele ja lahtriplokkidele nimede määramiseks on mitmeid viise. Kõige lihtsamaks on **nimeboksi** kasutamine. Nimede määramine toimub siin analoogselt graafikakujunditele nimede panemisega. Valida välja lahter või lahtriplokk, tippida nimeboksis nimi ja vajutada klahvi Enter. Vahelehel *Formulas* (Valemid) on käsk *Name Manager* (Nimehaldur), mille alusel kuvatav dialoogiboks sisaldab mitmeid võimalusi nimede määramiseks ja muutmiseks. Nimed saab määrata korraga mitmele lahtrile käsuga *Create from Selection* (Loo valikust).

Range-objektil on suur hulk omadusi, järgnevalt on toodud ainult mõned olulisemad:

Address lahtriploki (lahtri) aadress

Name lahtriploki (lahtri) nimi

Value lahtris salvestatud väärtus

Formula lahtris olev valem

Height lahtriploki (lahtri) kõrgus

Width lahtriploki (lahtri) laius

Left lahtriploki vasaku serva kaugus töölehe vasakust servast

Top lahtriploki ülemise serva kaugus töölehe ülemisest servast

VBA programm saab lugeda töölehe lahtrites olevaid väärtusi, leida nende alusel tulemused ja kirjutada need vastavatesse lahtritesse. Kasutaja, sisestades lahtrisse mingi väärtuse, muudab selle omadust *Value*. Sama teeb ka VBA protseduur lahtrisse väärtuse kirjutamisel.

Omadus Value on lahtriploki nn vaikimisi võetav omadus. See tähendab, et kui omadust objekti järel ei ole näidatud, arvestatakse, et tegemist on omadusega Value.

Seega
Range (“pikkus“).Value = 13 ja
Range (“pikkus“) = 13 on samaväärsed

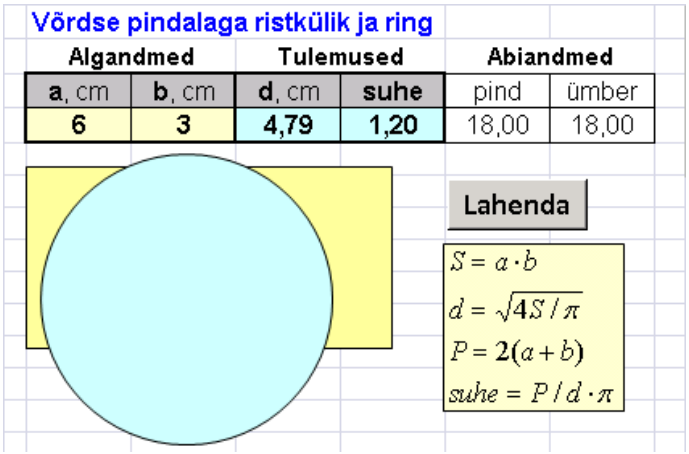
Lahtriploki meetodeid kasutatakse näiteks toodud rakendustes suhteliselt harva. Nimetame siin vaid mõned: valimine (Select), kopeerimine (Copy), eemaldamine (Delete).

Näide: Võrdse pindalaga ristkülik ja ring

Programm leiab läbimõõdu ringi jaoks, mille pindala on võrdne antud ristküliku pindalaga ning arvutab ristküliku ümbermõõdu ja ringjoone pikkuse suhte. Makro muudab ka ristküliku ja ringi mõõtmeid vastavalt andmetele. Mõõtühikud on sentimeetrites. Selle näiteks on toodud kolm varianti.

Variant 1 – kasutatakse ainult objektide omadusi

Rakenduse kasutajaliides on toodud alloleval pildil. Töölehel on nimedega lahtrid andmete jaoks. Algandmed: ristküliku mõõtmed **a** ja **b**; tulemused: ringi läbimõõt **d** ja **suhe** ning abiandmed: ristküliku pindala **pind** ja ümbermõõt **ümber**.



Samuti on töölehel graafilised kujundid: ristkülik ja ring nimedega **rist** ja **ring**. Olgu märgitud, et kujundite esialgsed mõõtmed ei ole üldse olulised, need paneb paika programm.

Rakendus demonstreerib kasutajaliidese elemente Exceli töölehel. Väga olulised on siin nimedega lahtrid, kuhu saab salvestada vajalikke andmeid. Vormindamiseks saab kasutada Exceli vahendeid. Ka graafilised kujundid lisatakse Exceli käskudega.

```
Sub Rist_Ring_1()
' Võrdsete pindalaga ristkülik ja ring. Arvutused
  Range("pind").Value = Range("a").Value * Range("b").Value
  Range("ümber") = 2 * (Range("a") + Range("b"))
  Range("d") = Sqr(4 * Range("pind") / 3.14159)
  Range("suhe") = Range("ümber") / (3.14159 * Range("d"))
' Kujundite mõõtmete muutmine
  Shapes("rist").Width = Range("a") * 72 / 2.54
  Shapes("rist").Height = Range("b") * 72 / 2.54
  Shapes("ring").Width = Range("d") * 72 / 2.54
  Shapes("ring").Height = Shapes("ring").Width
End Sub
```

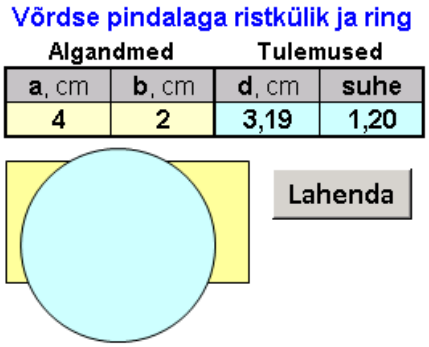
Programm koosneb ühest protseduurist, milles kasutatakse ainult objektide omadusi. Kõik andmed (algandmed, tulemused ja abiandmed) on töölehe lahtrites. Töölehel on ka programmi käivitamisnupp.

Programm loeb lahtritest **a** ja **b** külgede pikkused (formaalselt on lahtrite **a** ja **b** omadused *Value*), leiab nende korrutise ning kirjutab selle lahtrisse **pind** (formaalselt omistab väärtuse lahtri omadusele Value).

Kuna omadus Value võetakse ka vaikimisi, on see edaspidi ära jäetud. Järgmised kolm lauset leiavad ümbermõõdu, läbimõõdu ja suhte väärtused ning kirjutavad need töölehe vastavatesse lahtritesse. Viimased neli lauset panevad paika kujundite mõõtmed. Arvestatakse, et mõõtmed on sentimeetrites. Tegur 72 / 2.54 teisendab sentimeetrid ekraaniühikutesse: 1 punkt = 1 / 72 tolli.

Variant 2 – muutujate kasutamine

Rakendus lahendab sama ülesannet, mida tegi eelmine variant, kuid muudab lisaks ka kujundite täitevärvust. Siin kasutatakse muutujaid, kuhu salvestatakse algandmete, tulemuste ja abiandmete väärtused, tänu millele ei ole abiandmete salvestamiseks vaja eraldi lahtreid töölehel.



Viitamiseks kujunditele kasutatakse objektimuutujaid.

Tegemist on üsna tüüpilise VBA toega rakendusega Excelis arvutusliku iseloomuga ülesannete jaoks. Algandmete sisestamiseks ja säilitamiseks kasutatakse töölehe lahtreid, kust programm loeb need VBA muutujatesse. Arvutuste ajal asuvad kõik vajalikud andmed: algandmed, abiandmed ja tulemid, VBA tööpiirkonna andmeplokis. Vajalikud tulemused kirjutab programm töölehele. Soovi korral saab andmed hõlpsasti printida. Andmete ja ka programmi säilitamiseks ei ole vajalikud eraldi meetmed vaid piisab töövihiku salvestamisest. Andmete vormindamise saab teha lihtsalt ja kiiresti Exceli vahenditega.

```
Sub Rist_Ring_2()
' Võrdsete pindalaga ristkülik ja ring
  Const pi = 3.14159, cm_p = 72 / 2.54
  Dim a, b, d, S, P, suhe ' muutujad
' Algandmete lugemine töölehelt
  a = Range("a"): b = Range("b")
' Arvutamine
  S = a * b: P = 2 * (a + b)
  d = Sqr(4 * S / pi): suhe = P / (pi * d)
' Tulemite kirjutamine töölehele
  Range("d") = d: Range ("suhe") = suhe
' Kujundite mõõtmete ja värvuse muutmine
  Dim rist As Shape, R As Shape
  Set rist = Shapes("rist"):  Set R = Shapes("ring")
  rist.Width = a * cm_p: rist.Height = b * cm_p
  R.Width = d * cm_p: R.Height = d * cm_p
  rist.Fill.ForeColor.SchemeColor = 70 * Rnd()
  R.Fill.ForeColor.SchemeColor = 70 * Rnd()
End Sub
```

Lausega **Const** määratletakse kaks nimega konstanti: **pi** ja **cm_p**. Viimast kasutatakse sentimeetrite teisendamiseks punktidesse. **Dim** lause abil on määratletud muutujad. Nende väärtuse tüüp ei ole näidatud ning VBA valib need ise.

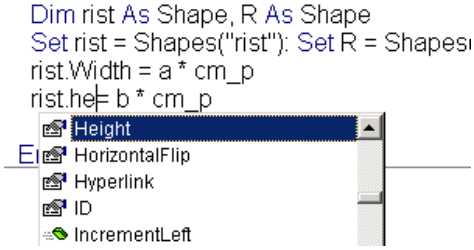
Algandmed loetakse töölehe lahtritest nimedega **a** ja **b** ning salvestatakse muutujatesse **a** ja **b**.

Arvutamisel kasutatakse muutujatesse salvestatud algandmete väärtusi ja muutujaisse salvestatakse ka abiandmete ja tulemite väärtused. Tulemused kirjutatakse muutujatest töölehe lahtritesse.

Deklareeritakse objektimuutujad **rist** ja **R** ning **Set**-lause abil seotakse nendega töölehel asuvad kujundid **rist** ja **ring**, mis võimaldab lühendada järgnevates lausetes viiteid objektidele.

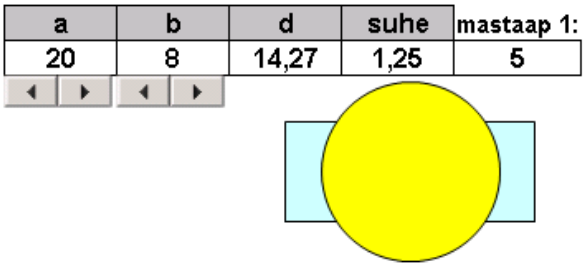
Viitamiseks graafikaobjektidele kasutatakse muutujaid ja ka omaduste väärtused on praegu salvestatud muutujates. Viimases kahes lauses muudetakse kujundite täitevärvust. Värvide numbrid määratakse juhuarvude abil.

Objektimuutujate jaoks on tüübi esitamine **Dim**-lauses (siin **As Shape**) kasulik programmilausete sisestamisel. Kui tüüp on määratud, pakub VB editor pärast objektiviida järele punkti sisestamist vastava objekti omaduste ja meetodite loetelu (spikrit), millest saab lihtsalt ja kindlalt valida vajaliku omaduse või meetodi.



Variant 3 – alamprotseduuride kasutamine

Antud rakendus võimaldab lisaks eelmistes tehtutele kasutada mastaapi, muuta algandmeid spinnerite abil ja paigutada kohakuti ristküliku ja ringi keskpunktid.



Programm demonstreerib parameetrite nimede kasutamist objektidele viitamisel. Samuti tutvustatakse siin sündmuste ja sündmusprotseduuride kasutamist.

```
Sub Rist_Ring_3()
' Võrdse pindalaga ristkülik ja ring
  Const pi = 3.14159
  Dim a, b, S, P, d, mas
  a = Range("a"):  b = Range("b")
  mas = Range("mastaap")
  S = a * b:    P = 2 * (a + b)
  d = Sqr(4 * S / pi)
  Range("d") = d
  Range("suhe") = P / (pi * d)
  Tee_skeem Shapes("rist"), _
      Shapes("ring"), a, b, d, mas
End Sub
```

```
Sub Tee_skeem(rist As Shape, R As Shape, a, b, d, m)
' Kujundite mõõtmete ja asukoha muutmine
  Const cm_p = 72 / 2.54
  rist.Width = a * cm_p / m
  rist.Height = b * cm_p / m
  R.Width = d * cm_p / m
  R.Height = R.Width
  R.Left = rist.Left + rist.Width / 2 - R.Width / 2
  R.Top = rist.Top + rist.Height / 2 - R.Height / 2
  rist.Fill.ForeColor.SchemeColor = 70 * Rnd()
  R.Fill.ForeColor.SchemeColor = 70 * Rnd()
End Sub
```

Programm koosneb kahest protseduurist. Peaprotseduur loeb töölehelt algandmed, teeb arvutused ja käivitab alamprotseduuri **Tee_skeem**, andes viimasele argumentidena viidad graafikaobjektidele **rist** ja **ring** ning nende mõõtmed ja mastaabi. Viitamisel kujunditele kasutatakse parameetrite nimesid ja see on täiesti analoogne muutujate kasutamisele.

Sündmused ja sündmusprotseduurid

Süsteem reageerib mõnede objektidega seotud sündmustele (näiteks hiireklõps), käivitades vastava sündmusprotseduuri, kui selline on varem ette valmistatud. Selles jaotises vaatleme lühidalt töölehe ja mõnede AxtiveX juhtelementide (käsunupud, spinnerid, …) sündmusprotseduure.

```
Private Sub CommandButton1_Click()
  Call Rist_Ring_3  ' võib lihtsalt Rist_Ring_3
End Sub

Private Sub CommandButton2_Click()
  MsgBox "Tere!"
  nimi = InputBox("Mis on Sinu?")
  MsgBox nimi & "? See on väga ilus nimi!"
End Sub
```

Jaotises „Käsunuppude loomine ja kasutamine“ vaadeldi makro sidumist **ActiveX**-käsunupuga.

Selleks kasutatakse sündmusprotseduuri päisega

Sub CommandButtonN_Click()

N on nupu järjenumber (CommandButtonN on ActiveX-objekti nimi).

Kui käivitamist vajav protseduur on juba olemas, paigutatakse sündmusprotseduuri ainult pöördumislause – käivitatava protseduuri nimi.

Soovi korral võib vajalikud laused paigutada otse sündmusprotseduuri sisse nagu on näidatud protseduuris **CommandButton2_Click**().

Käsunupu jaoks on veel terve rida sündmusi, millele see võib reageerida vastava protseduuri abil. Igal sündmusel on kindel nimi: **DblClick**, **MouseDown**, **MouseUp** jms, mis ilmub automaatselt sündmus-protseduuri malli päises, kui sündmus valitakse loetelust.

Analoogselt toimub sündmusprotseduuride kasutamine teiste töölehele paigutatavate ohjurite jaoks. Olgu märgitud, et ohjureid saab kasutada ka ilma VBA kasutamiseta.

```
Private Sub Spinner_a_Change()
  Rist_Ring_3
End Sub
```

Eespool toodud näites olid kasutusel spinnerid  , mille abil saab muuta lahtrite **a** ja **b** väärtusi. Spinner seotakse lahtriga omaduse **LinkedCell** abil.

```
Private Sub Spinner_b_Change()
  Rist_Ring_3
End Sub
```

Sündmusprotseduur **Change** võimaldab käivitada vastavad tegevused kohe, kui muudetakse lahtri väärtust (formaalselt spinneri väärtust).

Ka töölehega on seotud terve rida sündmusi, mis võimaldavad käivitada vastavad sündmusprotseduurid:

- **Activate** – lehe aktiveerimine,
- **Deactivate** – lehe deaktiveerimine (lahkumine antud lehelt),
- **Change** – muutus mingi lahtri väärtus,
- **SelectionChange** – muutus aktiivseks uus lahter.

```
Private Sub Worksheet_Activate()
  MsgBox "Tere! Kuidas hästi elad?"
  MsgBox "Hakkame tööle!"
End Sub
```

Kui tööleht, mille moodulis asub antud sündmusprotseduur, muutub aktiivseks, käivitub sündmusprotseduur **Worksheet_Activate** ja kuvatakse toodud teated.

```
Private Sub Worksheet_Deactivate()
   Salvesta
   End
End Sub
```

Kui lahkutakse antud töölehelt, rakendub sündmusprotseduur **Worksheet_Deactivate**, käivitatakse protseduur **Salvesta** ja pärast selle töö lõppu katkestatakse käsuga **End** kõikide protseduuride töö.

Sündmus **Change** tekib, kui muutub töölehe mõne lahtri sisu. Uue väärtuse võib lahtrisse kirjutada kasutaja või programm. Parameetri Target väärtuseks saab protseduur muudetud lahtri.

```
Private Sub Worksheet_Change(ByVal Target As Range)
  If Target = Range("a") Or Target = Range("b") Or _
     Target = Range("maastap") Then  Rist_Ring_3
  End If
End Sub
```

Kui muudetud lahter oli **a**, **b** või **mastaap**, käivitatakse programm **Rist_Ring_3**.

Kui taolist kontrolli ei tehta, tekib töölehele kirjutamisel korduv sündmusprotseduuri käivitamine.

Sündmus **SelectionChange** tekib, kui muutub aktiivne lahter. Lahtrikursori asukohta muudetakse kas hiireklõpsuga, nooleklahvidega või programmis käsuga **Select**. Uue aktiivse lahtri (Range-objekt) saab protseduur parameetri **Target** väärtusena. Viimase omadusi ja meetodeid saab kasutada programmis.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
   Dim J As Shape
   Set J = Shapes("Juku")
   J.Left = Target.Left + Target.Width / 2 - J.Width / 2
   J.Top = Target.Top - J.Height + Target.Height
End Sub
```

Kui muutub aktiivse lahtri asukoht, viiakse kujund **Juku** aktiivse lahtri kohale. Seega liigub **Juku** koos lahtrikursoriga.

NB! Sündmusprotseduuride käivitamisevõimaluse saab välja (ja uuesti sisse) lülitada objekti Application omaduse EnableEvents abil. Kui Application.EnableEvents = False, siis sündmusprotseduurid ei käivitu.

Töö tabelitega

Tabel asub töölehe ristkülikukujulises sidusas lahtriplokis. Rakenduse loomisel peab arvestama, et tabel võib asuda töölehe suvalises piirkonnas ning erinevatel kasutamiskordadel võivad tabelil olla erinevad mõõtmed. Tegevuste määramiseks tabelitega on sageli vaja viidata programmis tervele tabelile ja/või selle elementidele: lahtrid, rivid (read), tulbad (veerud), suvalised osad tabelis.

Andmete määratlemiseks on otstarbekas anda tabelile või selle mingi(te)le lahtri(te)le nimed.

Viit- ehk objektimuutuja kasutamine (omistamine **Set**-lausega) võimaldab kasutada programmis oluliselt lühemaid lauseid.

Tabeli mõõtmete ja asukoha määramisel saab kasutada lahtriploki vastavaid omadusi.

Omadus	**Selgitus, näited**
Rows / Columns	piirkonna kõik rivid / tulbad
Rows(*rn*) **Columns(*tn*)**	rivi/tulp etteantud numbriga: prk.Rows(1), prk.Rows(i), prk.Columns(13), prk.Columns(k+3)
Cells	piirkonna kõik lahtrid: k = prk.Cells.Count: Range(“mass“).Cells.Count
Cells(*rn*, *tn*)	lahter rivis *rn*, tulbas *tn*: x=prk.Cells(1, 1): **For** i=1 **To** m: S = S + T.Cells(i, 3): **Next** i .Cells võib jätta ära: x = prk (1, 1): **For** i=1 **To**: S=S+ T(i, 3): **Next** i
CurrentRegion	sidus mittetühi ristkülikukujuline piirkond lahtri (lahtriploki) ümber: Set T = Range(“talg“).CurrentRegion
Offset(rnihe, tnihe)	nihutatud piirkond: T.Offset(1, 2), T.Offset(1, 0)
Resize(*m*, *n*)	mõõtmete muutmine: T.Resize(5, 4), T.Resize(1, n)-tulp, T.Resize(m, 1)-rivi
Cells.Count **Rows.Count** **Columns.Count**	lahtrite, ridade, veergude arv piirkonnas (tabelis): k = Range(“nimed“).Cells.Count: m = A.Rows.Count: n = A.Columns.Count

Vaatame jaotises „Makro kasutamine tabelis“ toodud ülesannet, kus leiti kehamassiindeks, saledus ning pikkuse, massi ja indeksi aritmeetiline keskmine mitme isiku jaoks. Erinevalt eelmisest, kus tabeli ja selle elementide (tulpade) identifitseerimiseks kasutati mitut nime, on siin mängus ainult üks nimi: tabeli esimese lahtri (ülemine vasakpoolne lahter) nimi **tabalg**. Tabeli määratlemisel mängib olulist rolli omadus **CurrentRegion**, mis määratleb mittetühja piirkonna lahtri (või lahtriploki) ümber. See lihtsustab üsna oluliselt tööd nn dünaamiliste (muutuvate mõõtmetega) tabelitega. Tulpade määratlemisel kasutatakse omadusi **Offset** ja **Resize**.

```
Sub Statistika1()
  Dim T As Range, m, i
  Set T = Range("tabalg").CurrentRegion
  m = T.Rows.Count
  For i = 2 To m
    T(i, 4) = kind(T(i, 2), T(i, 3))
    T(i, 5) = saledus(T(i, 2), T(i, 3))
  Next i
  T(m + 2, 1) = "keskmine"
  T(m + 2, 2) = p_kesk(T.Offset(1, 1).Resize(m - 1, 1))
  T(m + 2, 3) = p_kesk(T.Offset(1, 2).Resize(m - 1, 1))
  T(m + 2, 4) = p_kesk(T.Offset(1, 3).Resize(m - 1, 1))
End Sub
```

Omadusega **CurrentRegion** määratud piirkond seotakse muutujaga **T**.

Lause m = T.Rows.Count leiab rivide arvu tabelis **m**. **NB!** Sisaldab ka päiserida. Korduses on esimese rivi numbriks **2**: tabeli keha esimene rivi.

Omadusega Offset(1, *tnihe*) määratakse piirkond, mille algus on 1 rivi allpool ja *tnihe* tulpa paremal T algusest.

Mõõtmed on samad nagu T-l (m * 5). Neid muudetakse omadusega Resize(m-1, 1).

Saadud tulbad on argumendiks pöördumistel funktsiooni **p_kesk** poole.

Andmed ja avaldised VBAs

Andmete liigid ja tüübid

Andmete põhiliikideks VBAs on **stringid** (sõned ehk tekstid), **arvud**, **ajaväärtused** ja **tõeväärtused**. Iga andmeliigi jaoks on määratletud võimalikud väärtused ja nende diapasoon, lubatavad tehted ja operatsioonid.

Arvutisüsteemides on iga andmeliigi jaoks ette nähtud kindel esitusviis ehk vorming, mida kasutatakse väärtuste salvestamiseks mälus ja operatsioonide täitmisel protsessoris. Vorminguga on määratletud väärtuste salvestamiseks eraldatavate mäluväljade struktuur ja pikkused. Ühe andmeliigi jaoks võib olla kasutusel mitu erinevat vormingut ehk tüüpi.

Stringi väärtuseks võib olla suvaline märgijada, mida saab arvutis esitada. Märkide valik ja nende arvutisisene esitus põhineb standarditega fikseeritud kodeerimissüsteemidel. Kasutatakse ASCII või Unicode kooditabelite süsteemi. ASCII süsteemis vastab igale märgile kindel 8-bitine arv (kood), mis esitatakse ühe baidi abil, Unicode'is – 16 bitine (2 baiti). Stringi (sõne) pikkusele (märkide arvule stringis) VBs praktikas segavaid piiranguid ei ole.

Arvude salvestamiseks kasutatakse erinevaid vorminguid. Need võivad olla esitatud üldises tekstivormingus (ASCII-koodis), kus igale numbrile eraldatakse üks bait. Kuna aga erinevate märkide hulk on arvude esituses üsna väike (numbrid, arvu märk ja võimalik murdosa eraldaja), siis on nende salvestamiseks ja töötlemiseks ette nähtud erivormingud, mis on üldisest tekstivormingust ökonoomsemad. Täisarvude ja reaalarvude jaoks kasutatakse fikseeritud pikkusega välju ning erinevaid esitusviise.

Täisarvud teisendatakse arvutis kahendsüsteemi ning esitatakse kahendnumbrite (bittide) jadana, ühte bitti kasutatakse arvu märgi esitamiseks. Arvu maksimaalne väärtus sõltub temale eraldatud välja pikkusest max = $\mathbf{2^{n-1} - 1}$, kus **n** on välja pikkus bittides. Kasutatakse **kahe-** ja **neljabaidiseid** välju (16 või 32 bitti), millele vastavad arvude maksimaalsed väärtused $2^{15} - 1 = 32\,767$ ja $2^{31} - 1 = 2\,147\,483\,647$.

Reaalarvud esitatakse mantissi ja eksponendi abil: arv = m•p^n, kus m on mantiss (arvu tüvi), n – eksponent (astendaja) ja p - arvusüsteemi alus (2, 10 või 16). Mantiss esitab arvu numbreid, eksponent koma mõttelist asukohta. Kasutatakse nelja- ja kaheksabaidiseid välju, millele vastavad esitustäpsused 6–7 numbrikohta (ühekordne täpsus) ja 15–16 numbrikohta (topelttäpsus) ning maksimaalsed väärtused 10^{37} ja 10^{307}.

Ajaväärtus koosneb üldjuhul kuupäevast ja kellaajast. Need salvestatakse ühe reaalarvuna. Arvu täisosa näitab päevade arvu alates 01.01.1900, murdosa esitab kellaaja alates kesköösst. VBs saab kasutada aja-väärtusi alates 01.01.100 kuni 31.12.9999.

Tõeväärtusi on kaks: **tõene** (*True*) ja **väär** (*False*). VBs esitatakse need täisarvudena – väärtusele **väär** vastab **0**, väärtusele **tõene** vastab tavaliselt **-1**, kuid sellena käsitletakse ka suvalist nullist erinevat väärtust.

Programmi koostamisel saab määrata muutujate väärtuste jaoks sobivad esitusviisid ehk tüübid, kasutades spetsiaalseid deklareerimislauseid, millest peamine on **Dim**-lause.

Järgnev **Dim**-lause deklareerib nelja muutuja väärtuste tüübid

Dim n **As Integer**, *a* **As Single**, *b* **As Double**, *nimi* **As String**

Lause määrab, et muutuja **n** väärtusteks võivad olla täisarvud, **a** väärtusteks ühekordse täpsusega reaalarvud, **d** väärtusteks topelttäpsusega reaalarvud ning muutuja **nimi** väärtusteks stringid. Arvmuutujate jaoks tulenevad siit ka väljade pikkused: **n** – 2 baiti, **a** – 4 baiti ning **b** – 8 baiti. Stringmuutuja **nimi** välja pikkus on muutuv ning sõltub talle omistatavatest väärtustest.

Andmetüüp määratletakse deklaratsioonis võtmesõna abil. Igale tüübile vastab kindel väärtuste esitusviis ja diapasoon ning välja pikkus. Mitme andmetüübi määramiseks võib kasutada ka **tüübitunnuseid** - kindla tähendusega sümboleid, mis lisatakse muutuja nime lõppu. Näiteks on järgnev lause samaväärne eelmisega

Dim n%, a!, b#, nimi$

Andmete põhitüübid VBAs

Tüübi nimetus	**Tüübi tähis**	**Väärtuse tüüp**	**Välja pikkus (Bait)**	**Väärtuste diapasoon**
Integer	%	Täisarv	2	-32 768 ... 32 767
Long	&	Pikk täisarv	4	-2 147 483 648 ... 2 147 483 647
Single	!	Ühekordse täpsusega reaalarv	4	suurim väärtus umbes 10^{37} täpsus 6..7 numbrikohta
Double	#	Topelttäpsusega reaalarv	8	suurim väärtus umbes 10^{307} täpsus 15..16 numbrikohta
String	$	String (tekst)	märkide arv + 10 B	0 kuni ca 2 mln märki
Date		Kuupäev	8	01.01.100 ... 31.12.9999
Boolean		Tõeväärtus	2	True või False
Variant arvuga		Suvaline arvu tüüp	16	suurim väärtus umbes 10^{307} täpsus 15...16 numbrikohta
Variant stringiga		String (tekst)	märkide arv + 22 B	0 kuni umbes 2 mln märki
Objekti-muutujad		Viit objektile. Määratletakse **Set**-lausega	4	

Vaikimisi, s.t kui muutuja tüüp ei ole deklareeritud, võetakse tema tüübiks universaalne andmetüüp **Variant**. Taolistele muutujatele võib omistada suvalist tüüpi väärtusi. Väljade pikkused võetakse nende jaoks varuga ning väärtuse esitusviisi valib interpretaator sõltuvalt omistatava väärtuse tüübist. Näiteks arvu salvestamiseks, sõltumata tema tüübist ja suurusest, eraldatakse alati väli pikkusega 16 baiti. **Variant**-tüübi kasutamine arvandmete jaoks on üldiselt ebaökonoomne nii mäluruumi kasutamise kui ka programmi töökiiruse poolest. Selle tüübi korral on vajalik mälumaht keskmiselt 3...5 korda suurem ning arvutuste kiirus 2...3 väiksem kui spetsiaalselt arvude jaoks ettenähtud tüüpide kasutamisel. Objektimuutujate jaoks määratakse tüüp klassi nime abil: Range, Shape, Worksheet jne.

Skalaarandmed

Konstandid

Konstandi väärtus näidatakse programmis ning ta ei muutu programmi täitmise ajal. Iga andmeliigi jaoks on ette nähtud kindlad konstantide esitamise reeglid.

Arvkonstandid esitatakse tavaliste kümnendarvudena või kümne astmega. Reaalarvudes kasutatakse murdosa täisosast eraldamiseks punkti.

13 -345 647.234 -35.67 2.1E6 = 2.1×10^{6} 1e-20 =10^{-20}

Stringkonstandis kasutatakse suvalisi märke, mida saab arvutisüsteemis esitada. Konstandi väärtus paigutatakse jutumärkide vahele.

"a" "Pindala" "x1=" "Mis on Sinu nimi?" “Ei”

Ajakonstant sisaldab üldjuhul kuupäeva ja kellaaega. Konstant paigutatakse numbrimärkide (#) vahele. Väärtuste esitamiseks kasutatakse erinevaid vorminguid, põhivariant on järgmine: # kk/pp/aa tt:mm:ss #

05/13/99 14:23:45 # # 12/24/99 # # 13:45

Tõeväärtusi on ainult kaks, need esitatakse võtmesõnade **True** (**tõene**) ja **False** (**väär**) abil.

Võib kasutada nimeta ja nimega konstante. **Nimeta konstant** esitatakse otse avaldises või lauses

5 * (a^2 + b^2), 3.14159 * d^2 / 4, "Summa=" & S

Nimega konstant deklareeritakse (määratletakse) **Const**-lause abil, mille struktuur on järgmine:

Const nimi [**As** tüüp] = väärtus { , nimi [**As** tüüp] = väärtus }

tüüp esitatakse tüübi nimetust tähistava võtmesõna abil ***Integer***, ***Long***, ***Single***, ***Double***, ***String***, ***Boolean***, ***Date*** või ***Variant*.** Kui tüüp pole näidatud, siis võetakse selleks ***Variant***. Olgu märgitud, et tüübi otsest määratlemist konstantide jaoks kasutatakse võrdlemisi harva. Deklaratsioonide näiteid

Const pi = 3.14159, Nmax = 1000, pea = "Viktoriin"

Konstandi nime võib kasutada erinevates avaldistes ja lausetes viitamiseks vastavale väärtusele

L = 2 * pi * r : S = pi * r^2.

Deklaratsioonis konstandi väärtuse muutmisel muutub see (tema skoobi piires) kõikjal, kus tema nimi esineb. Erinevalt muutujast ei saa nimega konstandile omistada väärtusi programmi täitmise ajal.

Konstante võib deklareerida protseduuri sees ning mooduli alguses väljaspool protseduure. Esimesel juhul saab neid kasutada ainult protseduuris, kus need on deklareeritud. Teisel juhul saab neid kasutada kõikides antud mooduli protseduurides, kus sama nime pole deklareeritud muuks otstarbeks.

VBAs on rida sisekonstante. Igal taolisel konstandil on kindel tähendus, väärtus ja nimi. Neid kasutatakse kindlates kohtades ja kindlas tähenduses, peamiselt mõne sisefunktsiooni argumendina. Kõikide sisekonstantide nimed algavad tähtedega vb:

vbBlack, vbRed, vbOK, vbCancel, vbYes, vbSunday

Muutujad

Muutujad võib jagada kolme rühma:

- lihtmuutujad ehk skalaarmuutujad,
- objektimuutujad ehk viitmuutujad,
- struktuurmuutujad.

Lihtmuutuja (ehk lihtsalt muutuja) on nimega varustatud koht arvuti mälus - järjestikku baitide rühm: mälu**väli** ehk mälu**pesa**, kuhu programm saab täitmisel ajal salvestada väärtusi (arve, tekste jm) ja hiljem kasutada neid näiteks uute väärtuste leidmiseks või ka muuks otstarbeks.

Väljad muutujatele eraldab VBA interpretaator oma tööpiirkonna andmeplokis. Välja pikkus (baitides) sõltub muutuja tüübist, mis määratakse programmis otseselt või kaudselt. Igal ajahetkel saab muutujal olla ainult üks väärtus. Väärtuse **salvestamist** antud muutuja pesas nimetatakse muutujale väärtuse **omistamiseks**. Seni kuni muutujale pole omistatud väärtust, on see määramatu.

Objektimuutujale eraldatakse samuti VBA tööpiirkonnas mäluväli ehk pesa. Välja pikkus on alati neli baiti. Erinevalt tavamuutujast (ehk lihtmuutujast) ei salvestata sinna väärtusi, vaid nn **viit** objektile: objekti omaduste vektori aadress mälus. Tänu selle, saab muutuja nime abil viidata objektile ning selle omadustele ja meetodile. See on reeglina oluliselt lühem ja lihtsam, kui otsene viitamine.

Struktuurmuutujale (massiivmuutuja jm) vastab mitu omavahel seotud ja teatud organisatsiooni omavat mälupesa (elementi). Massiivmuutujaid ehk massiive käsitletakse jaotises „Massiivid“.

Protseduuride **sisemuutujad** võib kasutusele võtta **deklareerimata**, esitades nende nimed lausete sellistes kohtades, kus muutujate kasutamine on lubatud (näiteks omistamislausetes). Selle kohta öeldakse, et muutujad **on deklareeritud kaudselt**. Kui deklaratsioonid puuduvad, teeb VBA interpretaator konteksti järgi kindlaks kõik protseduuris esinevad muutujad, eraldab neile väljad ja kasutab neid protseduuri täitmise ajal. Muutuja tüübiks võetakse **Variant**.

Enamasti on otstarbekas määratleda muutujad otse, kasutades vastavaid **deklaratsioonilauseid**. Deklaratsiooniga saab määrata järgmised muutuja omadused:

- nimi,
- tüüp,
- skoop ehk määramispiirkond,
- väärtuste eluiga.

Peamiseks lauseks muutujate deklareerimiseks on **Dim**-lause, mille kuju on järgmine:

Dim nimi [**As** tüüp] [, nimi [**As** tüüp]] …

kus tüüp esitatakse lihtmuutujate jaoks tüübi nimetust tähistava võtmesõna ***Integer*, *Long*, *Single*, *Double*, *String*, *Boolean*, *Date*** või ***Variant*** abil. Objektimuutujate jaoks esitatakse tüüp klassi nime abil: *Range*, *Shape*, *Worksheet* jmt. Kui tüüpi pole näidatud, siis võetakse selleks *Variant*.

Deklaratsioonide näiteid

Dim *a* **As Single**, *b* **As Single**, *n* **As Integer**, *x* **As Double**, *t* **As Boolean**

Dim *enimi* **As String**, *pnimi* **As String**, *sünniaeg* **As Date**, *palk* **As Double**

Mõnede tüüpide korral saab muutuja tüübi **Dim**-lauses määrata ka spetsiaalsete tüübitunnuste abil:

$ - ***String***, % - ***Integer***, & - ***Long***, ! – ***Single***, # - ***Double***.

Kasutades tüübitunnuseid, võib ülaltoodud **Dim**-laused esitada kompaktsemalt järgmiselt:

Dim *a*!, *b*!, *n*%, *x*#, *t* **As Boolean**

Dim *enimi$*, *pnimi$*, *sünniaeg* **As Date**, *palk#*

NB! Tõeväärtustel ja ajaväärtustel tüübitunnused puuduvad, nende jaoks kasutatakse tüübi määramiseks, kui vaja, vastavaid võtmesõnu.

NB! Tüüp näidatakse iga muutuja jaoks eraldi tema nime järel. Näiteks kehtib lauses

Dim *a, b, c* **As Double**, *d, e* **As Double**

tüüp **Double** ainult muutuja *c* ja *e* jaoks, teiste muutujate (*a*, *b* ja *d*) tüübiks võetakse **Variant**. Võtmesõnade kasutamisel kipuvad **Dim**-laused venima üsna pikaks, kui muutujaid on rohkem. Sellepärast on tüübitunnuste kasutamine sageli mugavam. Näiteks eelpool toodud lause, kui tahetakse kõikide muutujate jaoks määrata tüüp **Double**, näeb tüübitunnuse kasutamisel välja järgmiselt:

Dim *a*#, *b*#, *c*#, *d*#, *e*#

Järgnev **Dim**-lause

Dim *J* **As** Shape, *pall* **As** Shape, *e_tab* **AS** Range

määratleb kaks objektimuutujat viitamiseks graafikaobjektidele (*J* ja *pall*) ja ühe muutuja viitamiseks lahtriplokile (*e_tab*).

Et objektimuutujaid saaks kasutada viitamiseks, peab need tingimata siduma **Set**-lausete abil vastavate objektidega, näiteks:

Set *J* = Shapes(“Juku“) : **Set** *pall* = ActiveSheet.Shapes(“Juku“)

Set *e_tabel* = Range(“Edetabel“)

Kuigi muutujate deklareerimine ei ole VBs kohustuslik, on soovitav seda siiski teha. Kusjuures **lihtmuutujate** jaoks ei ole tüüpide määratlemine eriti vajalik, sest saavutav mõnebaidine kokkuhoid pole tänapäeva arvutite jaoks nimetamisväärne. Hõlbustamaks tähevigade leidmist nimedes, on otstarbekas siiski deklareerida kõik muutujad (võib ilma tüüpideta) ja paigutada mooduli algusesse korraldus:

Option Explicit

mis määrab, et muutujate deklareerimine on antud moodulis kohustuslik. Kui moodulis esineb deklareerimata muutuja, väljastab VB käivitamisel veateate:

Variable not defined – muutuja ei ole defineeritud – ja märgistab deklareerimata muutuja nime.

Objektimuutujate korral on tüübi (ehk klassi nime) esitamine **Dim**-lauses väga kasulik. Kui protseduuri teksti koostamisel sisestada peale objektiviita punkt, näitab redaktor antud klassi objektide omaduste ja meetodite loetelu, kust saab valida vajaliku elemendi. See hoiab kokku aega ja vähendab vigade võimalust.

Andmete skoop ja väärtuste eluiga

Andmete **skoop** (ingl *scope*) määratleb protseduurid ja moodulid, millest on juurdepääs andmeüksusele (parameetrile, nimega konstandile või muutujale) tema nime abil. Eristatakse kolme taset:

- protseduuri tase,
- mooduli tase,
- projekti tase.

Protseduuritasemega andmeid nimetatakse ka **lokaalseteks andmeteks**, mooduli- ja projektitasemega andmeid **globaalseteks andmeteks**.

Parameetrite jaoks on **skoobiks** alati ainult see **protseduur**, mille päises (**Sub**- või **Function**-lauses) need on esitatud (deklareeritud) ning seda muuta ei saa. Konstantide ja muutujate skoope saab määrata deklaratsioonidega.

Protseduuritaseme andmeid ehk lokaalseid andmeid saab kasutada ainult antud protseduuris. Need deklareeritakse protseduuri sees: konstandid **Const-**lause, muutujad **Dim**-lausega. Lihtmuutujaid saab kasutada protseduuri sees ka deklareerimata. Antud protseduuris on deklareerimata muutuja lokaalne siis, kui samanimelist muutujat või konstanti pole deklareeritud globaalsel tasemel.

Moodulitaseme andmed ehk mooduli ühisandmed deklareeritakse mooduli alguses. Nendele on juurdepääs kõikidest mooduli protseduuridest, milles ei ole deklareeritud samanimelist andmeüksust. Konstandid deklareeritakse **Const**-lausega, muutujad **Dim**-lausega.

Projektitaseme andmeid saab kasutada kõikides moodulites ja protseduurides, kus ei ole deklareeritud samanimelist andmeüksust. Need deklareeritakse suvalise **üldmooduli** alguses lausega **Public**.

Muutujate väärtuste eluiga tähendab perioodi, mille jooksul need väärtused eksisteerivad. Lokaalsete muutujate väärtuste eluiga on vaikimisi antud protseduuri täitmise aeg. Pärast protseduuri töö lõppu tema sisemuutujate väärtused kustutatakse. Et muutuja väärtus säiliks, peab tema deklareerimiseks kasutama **Dim**-lause asemel **Static**-lauset. Globaalsete muutujate väärtused säilivad seni, kuni täidetakse **End**-lause või suletakse rakendust sisaldav fail.

Avaldised ja VBA sisefunktsioonid

Avaldiste struktuur ja liigid

Avaldis määrab, mis tehted (operatsioonid) ja millises järjekorras on vaja väärtuse leidmiseks täita. Üldjuhul koosneb avaldis:

- operandidest,
- tehtesümbolitest,
- ümarsulgudest.

Erijuhul võib avaldis koosneda ainult ühest operandist. **Operandideks** võivad olla:

- konstandid,
- lihtmuutujad,
- objektide omadused,
- struktuurmuutujate elemendid,
- funktsiooniviidad.

Nimeta konstandi väärtus (arv, string jm) esitatakse otse avaldises. Nimega konstant deklareeritakse lausega **Const** ning avaldises kasutatakse selle nime.

Lihtmuutujad esitatakse samuti nimede abil, struktuurmuutujate elementide esitus sõltub andmekogumi liigist (massiiv, kirje jmt). Objekti omadus esitatakse kujul:

objekt [.*omadus*]

kus *omadus* esitatakse vastava nime abil: Value, Left, Top, Address jne. Vaikimisi võetava omaduse võib ära jätta, näiteks omadus *Value Range*-objektil.

Funktsiooniviit esitatakse kujul:

nimi (*argument* [, *argument*] …)

kus *nimi* on VBA sisefunktsiooni või kasutajafunktsiooni nimi (Sin, Sqr, CDbl jmt). VBA funktsioonide nimed ei kuulu reserveeritud võtmesõnade hulka, kuid muuks otstarbeks neid kasutada ei ole mõistlik, sest see võib tekitada segadust. Funktsiooniviites esinev *argument* näitab funktsioonile edastatavat väärtust. Argumendid võivad olla esitatud avaldiste abil. Argumentide arv, tüüp ja esitusjärjekord sõltuvad konkreetsest funktsioonist.

Tehted (operatsioonid) ja tehtemärgid (operaatorid):

- aritmeetika; ^ , * , / , \ , Mod , + , -
- stringitehted; & või +
- võrdlused; = , <> , < , <= , > , >=
- loogika; Not, And, Or

Tehete liigid on siin toodud prioriteetide kahanemise järjekorras. Aritmeetika- ja loogikatehete prioriteedid kahanevad vasakult paremale. Avaldise väärtuse leidmisel arvestatakse tehete prioriteete liikide vahel ning aritmeetika- ja loogikatehete puhul ka liigi sees. Üldjuhul võivad avaldises esineda tehted kõikidest liikidest.

Avaldises a + b > c **And** a + c > b **And** b + c > a esinevad aritmeetika-, võrdlus- ja loogikatehted. Väärtuse leidmisel täidetakse kõigepealt aritmeetika-, siis võrdlus- ning lõpuks loogikatehted.

Tehete järjekorra muutmiseks võib kasutada ümarsulge. Sulgudes asuva avaldise väärtus leitakse eraldi. Ümarsulgudes esitatakse ka funktsiooniviitade argumendid.

Sõltuvalt andmete liigist, kasutatavatest operatsioonidest ja leitava väärtuse liigist võib avaldised jagada järgmistesse rühmadesse: arvavaldised, tekst- ehk stringavaldised, loogikaavaldised.

Arvavaldised ja matemaatikafunktsioonid

Arvavaldiste operandide väärtusteks on arvud ning neis kasutatakse aritmeetikatehteid ning funktsioone, mis tagastavad arvväärtusi. Aritmeetikatehted ja nende prioriteedid on järgmised.

Prioriteet	**Tehtesümbolid** (operaatorid)	**Selgitus**
1	^	Astendamine a^n
2	-	Unaarne miinus -a * -b + c
3	* , /	Korrutamine, jagamine a * b, a / b
4	\	Jagatise täisosa a \ b, 13 \ 5 = 2
5	Mod	Jagatise jääk a Mod b, 13 Mod 5 = 3
6	+ , -	Liitmine, lahutamine a + b, a - b

Tehete prioriteetide rakendamise näiteid

-3^2 * 5 + 18 / 2 * 3 = -9 * 5 + 9 * 3 = -45 + 27 = -18,

(-3)^2 * 5 + 18 / (2 * 3) = 9 * 5 + 18 / 6= 48

4^3^2 = 64^2 = 4096 64^1/3 = 641/3 = 64/3 64^(1/3) = = 4

Matemaatikafunktsioonid

Funktsioon	**Selgitus**: **a – arvavaldis**
Sqr(a)	Ruutjuur Sqr(b^2 - 4*a*c) = (b^2 - 4*a*c)^(1/2)
Log(a)	Naturaallogaritm (ln a) Log(a) / Log(10) = log10a NB! Kümnendlogaritmi VBAs ei ole!
Exp(a)	e^a (e = 2,71828…) (Exp(-x) + Exp(2 * x)) / 2 = $(e^{-x} + e^{2x}) / 2$
Abs(a)	Absoluutväärtus Abs((a-x)/(a+x))
Sin(a), Cos(a), Tan(a)	Sin(x)+Cos(2*x)+Tan(x^2)-Cos(2*x)^2 = sin x+cos 2x+tan x^2 -$\cos^2$ 2x Argument radiaanides
Atn(a)	arctan radiaanides ($-\pi/2<x<\pi/2$). Atn(a/Sqr(1-a^2))=arcsin a 2*Atn(1) - Atn(a/Sqr(1-a^2)) = arccos a. 4*Atn(1) = π
Sgn(a)	Arvu märk Sgn(5)=1, Sgn(0)=0 (a=0), Sgn(-5)= -1 (a<0)
Rnd()	Juhuslik arv x: 0 <= x < 1. n=Int((b-a+1)*Rnd()+a) - täisarv vahemikus a <= n <= b

Teisendusfunktsioonid

Funktsioon	**Selgitus**: **a – arv- või tekstavaldis**
Asc(a)	Esimese märgi ASCII-kood. Asc(“Abi“)=65, Asc(1)=48
CDbl(a)	Teisendus topelttäpsusega realarvuks. | a | < 10^{307}
CInt(a)	Teisendus täisarvuks. -32768 ≤ a ≤ 32767. Ümardatakse.
CLng(a)	Teisendus pikaks täisarvuks. $-2^{31} \le a \le 2^{31}$
CSng(a)	Teisendus ühekordse täpsusega reaalarvuks. | a | < 10^{37}
CStr(a)	Teisendus stringiks
CVar(a)	Teisendus Variant tüüpi
Val(a)	Teksti teisendus arvuks. Murdosa jäetakse ära.
Int(a)	Lähim täisarv, mis on väiksem kui a Int(4.9)=4, Int(-4.9)=-5
Fix(a)	Arvu täisosa Fix(4.9) = 4 Fix(-4.9) = -4
Round(a [, mp])	Ümardab väärtuse mp numbrikohani murdosas
Format(a, for)	Arvu teisendamine etteantud vormingusse. for – vorming. Peamine variant “0.0{0}“. Võimaldab määrata murdosa pikkuse: Format(45.67375, "0.000") = > 45.674

Stringavaldised ja funktsioonid

Stringavaldiste operandide väärtuseks on stringid, neis võib kasutada stringitehet ja stringifunktsioone. **Stringitehet** & nimetatakse **sidurdamiseks**. See võimaldab ühendada stringe. Sidurdamiseks tuleks teisendada arvud tekstivormingusse (operaatori & kasutamisel teeb VB seda automaatselt). Sidurdustehte operaatorina võib kasutada ka märki + , kui mõlemad operandid on stringid.

Näiteid

"Peeter" & " " & "Kask" annab Peeter Kask, 35.7 & " " & 2.5 annab 35.7 2.5

Kui S=5378.75, x1=2.538, x2=-1.34, siis

"Summa= " & S annab Summa= 5378.75,

"x1= " & x1 & " x2= " & x2 => x1= 2.538 x2= -1.34

Stringifunktsioonid

s – stringavaldis, näidetes S = “Visual Basic“

Left(s, n)	eraldab stringist n vasakpoolset märki. Left(S,6)=Visual
Right(s, n)	eraldab n parempoolset märki. Right(S, 5)=Basic
Mid(s, m, n)	eraldab n märki alates märgist m. Mid(S, 8, 3)=Bas
Len(s)	leiab stringi pikkuse Len(S) = 12
InStr([n,] s1, s2)	leiab positsiooni, millest algab string s2 stringis s1. n näitab otsingu algust, vaikimisi 1. InStr(S, “a”) = 5
UCase(s)	muudab väiketähed suurtähtedeks. UCase(S)=VISUAL BASIC
LCase(s)	muudab suurtähed väiketähtedeks. LCase(S)=visual basic
Space(n)	moodustab n tühikust koosneva stringi. Space(80)
String(n, märk)	moodustab stringi, mis sisalda n näidatud märki
Ltrim(s), Rtrim(s), Trim(s)	eemaldavad tühikut vastavalt stringi algusest, lõpust ning lõpust ja algusest Trim(Inputbox("Sisestage nimi"))
Chr(kood)	ASCII koodile vastav märk Chr(65) = A
Asc(s)	esimesele märgile vastav ASCII kood Asc("A") = 65
Val(s)	teisendab stringi arvuks. Murdosa jäetakse ära.
Str(s)	teisendab arvu stringiks Str(3.14159*r*r)
Format(a, for)	teisendab väärtuse etteantud formaadiga tekstiks Format(34.36732,"0.00") => 34.37

Operatsioonides stringidega kasutatakse ka mõningaid erilauseid:
Mid(s1, m[, n]) = s2

Asendab stringis **s1** **n** märki, alates positsioonist **m**, märkidega stringist **s2**. Kui **n** puudub, kasutatakse stringi **s2** tervikuna.

LSet s1 = s2 salvestab stringi **s2** stringi **s1** algusesse (vasakule).

RSet s1 = s2 salvestab stringi **s2** stringi **s1** lõppu (paremale).

Võrdlused ja loogikaavaldised

Võrdlused on käsitletavad loogikaavaldiste erijuhtudena, nende kuju on:

avaldis1 operaator *avaldis2*

Võrdlusoperaatorid on: = , <> , < , <= , > , >=

avaldis1 ja *avaldis2* on arv- või stringavaldised. Ühes võrdluses esinevad avaldised peaks üldjuhul kuuluma samasse liiki. Võrdluses võib olla ainult üks operaator. Võrdluse tulemiks on alati tõeväärtus **True** (tõene) või **False** (väär).

Võrdluste näiteid
x <= 0, b*b - 4*a*c < 0, x*x + y*y > r*r, UCase(vastus) = "EI"

NB! Stringide võrdlemisel eristatakse suur- ja väiketähti!

Loogikaavaldise üldkuju on järgmine:

avaldis LTS *avaldis* [LTS *avaldis*]…

Siin on *avaldis* võrdlus- või loogikaavaldis ja LTS loogikatehte sümbol. Peamised loogikatehted on Or, And ja Not. Nende tähendused on

Or – või. Tehte **a** Or **b** väärtus on **tõene** (***True***), kui vähemalt ühe operandi väärtus on tõene, vastupidisel juhul on tulem **väär** (***False***).

And – ja. Tehte **a** And **b** tulem on **tõene** (***True***) ainult siis, kui mõlema operandi väärtused on tõesed, vastupidisel juhul on tehte tulem **väär** (***False***).

Not – mitte. Tehte Not **a** tulem on **tõene** (***True***) siis, kui a väärtus on väär (***False***) ja **väär** (***False***) vastupidisel juhul.

Loogikaavaldiste näiteid

x >= 2 And x <= 13, x < 2 Or x >13

a+b > c And a+c > b And b+c > a

Mõned loogikafunktsioonid

IsDate(a)	Tõene, kui argument (a) on kuupäev
IsEmpty(a)	Tõene, kui argumendi väärtus on tühi
IsMissing(param)	Tõene, kui parameeter puudub
IsNumeric(a)	Tõene, kui avaldise väärtus on arv
IsObject(a)	Tõene, kui argumendiks on objekt

Ajaavaldised ja ajafunktsioonid

Ajaväärtus sisaldab üldjuhul kahte osa: kuupäev ja kellaaeg. Ajaväärtused salvestatakse reaalarvudena: täisosa – päeva järjenumber alates baasajast (01:01:1900), murdosa – kellaaeg alates kesköösst päeva osades.

Ajaväärtuste kui reaalarvudega saab täita põhimõtteliselt suvalisi operatsioone, kuid praktilist tähendust omavad ainult liitmine, lahutamine ja võrdlemine:

aeg2 – aeg1 NOW – #01/01/2000#

kuupäev2 – kuupäev1 Date – sünniaeg

kellaaeg2 – kellaaeg1 finaeg – startaeg

kuupäev + arv Date+100

kuupäev – arv Date-100

jooksev aeg – algaeg Timer() – algaeg

NB! Kahe ajaväärtuse vahe saadakse päevades, see on arv, mille täisosa on päevade arv kahe ajaväärtuse vahel ja murdosa kellaaeg päeva osades.

Mõned ajafunktsioonid

Now()	Jooksev aeg arvutis kujul pp.kk.aaaa hh:mm:ss
Date()	Jooksev kuupäev kujul pp.kk.aaaa
Time()	Jooksev kellaaeg kujul hh:mm:ss
Timer()	Kesköösst möödunud aeg sekundites täpsusega 0,01 sek
Year(aeg)	Aasta number ajaväärtuses
Month(aeg)	Kuu number ajaväärtuses
Day(aeg)	Päeva number ajaväärtuses
Hour(aeg)	Tunnid ajaväärtuses
Minute(aeg)	Minutid ajaväärtuses
Second(aeg)	Sekundid ajaväärtuses
DateSerial(a, k, p)	Teeb kuupäeva aasta (a), kuu (k) ja päeva (p) alusel
TimeSerial(h, m, s)	Teeb kellaja tundide (h), minutite (m) ja sekundite (s) alusel

Omistamine ja omistamislause

Omistamise olemus ja omistamislause põhivariandid

Omistamine on üks fundamentaalsemaid tegevusi, mida arvuti saab programmi toimel täita. See seisneb mingi väärtuse või aadressi **salvestamises** arvuti sisemälu etteantud väljas või pesas – muutujas. Välja (pesa) eelmine väärtus (kui oli) kaob.

Üheks peamiseks vahendiks **väärtuse salvestamiseks** (omistamiseks) on kõikides programmeerimiskeeltes **omistamislause**. Tüüpiliselt eelneb **väärtuse** salvestamisele selle **leidmine** (tuletamine) etteantud avaldise abil. Kusjuures avaldise väärtuse leidmisele kaasneb enamasti muutujate ja/või omaduste varem salvestatud **väärtuste lugemine**. Omistamine võib kaasneda ka mõnede muude lausete ja meetodite täitmisele, näiteks korduslause **For**...**Next**.

VBAs võib eristada omistamislause kasutamise kuut põhivarianti:

- väärtuse omistamine lihtmuutujale,
- väärtuse omistamine objekti omadusele,
- viida omistamine objektimuutujale,
- väärtuse omistamine struktuurmuutuja (massiiv, kirje) elemendile,
- viida omistamine struktuurmuutuja elemendile,
- ühe struktuurmuutuja omistamine teisele struktuurmuutujale.

Järgnevas peatume esimesel kahel.

Väärtuse omistamine lihtmuutujale ja objekti omadusele

Esimese kahe omistamise ja vastavate omistamislausete olemus on praktiliselt sama. Muutujale eraldatav mäluväli (pesa) kujutab endast objekti, mille omadust **väärtus** omistuslause täitmisel muudetakse. Objekti igale omadusele eraldatakse omaduste vektoris mäluväli, kuhu omistamisel salvestatakse vastav väärtus.

Esimese kahe variandi jaoks on omistamislausete kujud järgmised:

muutuja = *avaldis*
objekt.*omadus* = *avaldis*

Siin *muutuja* esitatakse nime abil, objekt määratakse objektiviida või objektimuutuja abil ning *omadus* näidatakse vastava nime abil. Mõlemal juhul on avaldise olemus ja esitamise reeglid samad. Tuletame meelde, et avaldiste operandideks võivad olla konstandid, muutujad, objektide omadused, massiivi elemendid, funktsiooniviidad (funktsioonid) VBA sisefunktsioonidele või kasutaja funktsioonidele. Avaldis võib koosneda ka ainult ühest operandist.

Omistamislausete näiteid:

k = 0: x = 2.5: pi = 3.14159: nimi = "A. Kask": auto.Left = 25

Erijuht: muutujale või omadusele omistatakse konstandi väärtus – konstandi väärtus salvestatakse programmiplokist andmeplokki või omaduse vektori vastavasse mäluvälja.

NB! Öeldakse, et „k-le **omistatakse** null“ või „x-le **omistatakse** 2.5“, aga mitte „k **võrdub** nulliga“ või „x **võrdub** 2.5-ga“.

x = y : ring.Top = Shapes(“rist“).Top : ufo.Left = x : a = Range(“a“)

Erijuht: muutujale või omadusele omistatakse teise muutuja või omaduse väärtus – paremas pooles oleva muutuja või omaduse väärtus kopeeritakse vasakus pooles oleva muutuja või omaduse mäluvälja.

NB! Ka siin on õige öelda, et „y-i väärtus omistatakse x-le“ (lühemalt: „y omistatakse x-le“), aga mitte „x võrdub y-ga“.

x = a + i * h : y = 3 * Sin(x) - Sqr(x^4 + 5)

Üldjuht: leitakse paremas pooles oleva avaldise väärtus ja tulemus omistatakse vasakus pooles olevale muutujale, st salvestatakse antud muutuja mäluväljas (pesas).

k = k + 1: S = S + y: n = n – k : F = F * k: auto.Left = auto.Left + h

Erijuht: sama muutuja või omadus esineb omistuslause vasakus ja paremas pooles. Tegemist on muutuja või omaduse uue väärtuse leidmise ja asendamisega eelmise (jooksva) väärtuse alusel. Näiteks lause **k = k + 1** täitmisel loetakse k jooksev väärtus, liidetakse sellele 1 ja saadud tulemus võetakse k uueks väärtuseks, so k väärtust suurendatakse ühe võrra.

Tahaks rõhutada, et omistamislauset ei tohi mitte mingil juhul samastada võrrandi, võrdusega või valemiga, mida alguses sageli kiputakse tegema välise sarnasuse tõttu (eriti omistamisel lihtmuutujale).

Võrduses või võrrandis on vasak ja parem pool samaväärsed.

Näiteks võrdused **x = y** ja **y = x** on samaväärsed aga omistamislaused **x = y** ja **y = x** on lausa vastupidise toimega. Olgu näiteks pesade x ja y sisu vastavalt 5 ja 1. Lause **x = y** täitmisel kopeeritakse arv 1 pesast y pesasse x ja peale seda on mõlemas pesas väärtus 1, väärtus 5 pesas x kaob. Kui aga tegemist on omistamislausega **y = x**, kopeeritakse väärtus (5) pesast x pesasse y ja mõlemas pesas on nüüd väärtus 5.

Veel kaks sarnast aga vastupidise toimega lauset

Juku.Left = pall.Left ja **pall.Left = Juku.Left**

Esimese juhul "viiakse" objekt Juku objekti pall juurde: Juku vasak serv võetakse võrdseks palli vasaku servaga, teisel juhul, vastupidi – "tuuakse" pall Juku juurde.

Näiteks omistamislaused **2 = x** ja **x + 5 = y** on mõttetud ja lubamatud. Ei saa omistada väärtust konstandile või avaldisele ehk salvestada (!) konstandis või avaldises midagi. Omistuslause vasakus pooles võib olla ainult muutuja nimi või objekti omadus!
Võrrandid **k = k + 1** või **n = n – k** ei oma mingit mõtet matemaatikas: ei saa mingi suurus olla võrdne iseendaga pluss või miinus midagi, väljaarvatud triviaalne juht, kui see midagi võrdub nulliga.

Andmete lugemine töölehelt ja kirjutamine töölehele

Reeglina on otstarbekas arvutustes kasutatavad lähteandmed lugeda eelnevalt töölehelt VB muutujatesse. Muutujate kasutamine võimaldab loobuda vahetulemuste ja abiandmete salvestamisest töölehele. Ka tulemused on tüüpiliselt otstarbekas salvestada muutujatesse ja sealt kirjutada töölehele. Muutujate kasutamine suurendab programmide töökiirust kümneid ja isegi sadu kordi. See on väga oluline, kui on tegemist suuremate ülesannete ja andmehulkadega.

Andmevahetuseks Exceli töölehtede ja VBA vahel saab kasutada erinevaid vahendeid. Siinpakutavad on üksikväärtuste korral ühed lihtsamad ja enim kasutatavad. Tegemist on erikujuliste omistamislausetega.

Väärtuste lugemiseks töölehelt võib kasutada järgmise kujuga omistamislauset:
muutuja = Range("nimi").Value või muutuja = Range("nimi") või muutuja = [nimi]

Formaalselt on tegemist töölehe lahtri omaduse Value lugemisega ja selle salvestamisega VB muutujasse. Kuna omadus Value on lahtri jaoks vaikimisi võetav omadus, jäetakse see tavaliselt ära, näiteks:

a = Range("a"): b = Range("b"): L = Range("pikkus")

Väärtuste kirjutamiseks töölehele võib kasutada järgmise kujuga lauset:
Range("nimi").Value = *avaldis* või Range ("nimi") = *avaldis* või [nimi] = *avaldis*

Formaalselt on tegemist töölehe lahtri omaduse Value muutmisega. Üldjuhul leitakse avaldise väärtus ja saadud tulemus kirjutatakse töölehe lahtrisse. Ka siin võib omaduse Value ära jätta.

NB! Kui muutuja nimi langeb kokku lahtri nimega ei saa kasutada viitamiseks lahtritele konstruktsiooni [nimi], vaid peab kasutama põhivarianti Range ("nimi").

Dialoogibokside kasutamine

Dialoogibokse võib kasutada teadete väljastamiseks ning üksikväärtuste lugemiseks ja väljastamiseks. Siin vaadeldakse nende kasutamise lihtsamaid võimalusi.

VBA siseprotseduur MsgBox võimaldab väljastada teateid ja üksikuid väärtusi. Tema lihtsaim variant on:

MsgBox *stringavaldis*

Lause täitmisel peatub programmi täitmine ning kuvatakse teateboks stringavaldise väärtusega ja nupuga OK. Kui kasutaja klõpsab nuppu OK või vajutab klahvile Enter, eemaldatakse boks ja programmi töö jätkub.

Üksikute väärtuste lugemiseks võib kasutada VBA sisefunktsiooni InputBox, mille põhivariant on järgmine:

muutuja = InputBox(teade [,päis, pakkumine])

Funktsioonil on üks kohustuslik argument - teade, mis võib üldjuhul olla stringavaldis. Lause täitmisel peatatakse programmi töö ning ilmub sisendboks, milles on kuvatud argumendi teade väärtus. Dialoogiboksis on tekstiväli, kuhu kasutaja saab tippida sisestatava väärtuse, ning kaks nuppu OK ja Cancel. Kui kasutaja klõpsab nuppu **OK** või vajutab klahvile **Enter**, omistatakse tekstiväljas olev väärtus antud muutujale. Kui kasutaja klõpsab nuppu **Cancel** või vajutab klahvile **Esc**, omistatakse muutujale tühi string.

Argumendi **päis** väärtus (kui ta esineb) kuvatakse sisendboksi päises. Argument **pakkumine** on mõeldud vaikimisi võetava väärtuse soovitamiseks. See kuvatakse sisendboksi tekstiväljas ja kui kasutaja ei asenda seda, vaid klõpsab kohe nuppu OK, võetakse see muutuja väärtuseks. Näiteks kuvab lause

n = InputBox ("Katsete arv", "Korrutamine", 13)

sisendboksi teatega „Katsete arv“, boksi päises on tekst „Korrutamine“ ning tekstiväljas arv 13. Kui kasutaja klõpsab kohe nuppu **OK**, võetakse muutuja **n** väärtuseks 13.

Sisendboksist loetavat väärtust käsitletakse alati **tekstina**. Arvude puhul võib see põhjustada probleeme, kui arve kasutatakse **liitmistehetes**. Kuna stringide sidurdamise tehtesümbolina on lubatud kasutada ka märki **+**, võib arvude liitmise asemel toimuda nende sidurdamine.

Näiteks makro **Test1** täitmisel, kui **a** väärtuseks sisestada 20 ja **b** väärtuseks 10, kuvatakse järgmine vastus: Keskmine=1005. Lause c = (a + b) / 2 täitmisel sidurdatakse ***a*** väärtus ***b*** väärtusega ja saadakse 2010, mis jagamisel kahega annab tulemuseks 1005.

Et vältida taolisi asju, peaks sisestatud väärtuse teisendama arvuvormingusse:
muutuja = **Val**(InputBox(teade [,päis, pakkumine]))

Makros **Test2**, kus arvude sisestamisel kasutatakse funktsiooni **Val**, toimub arvude liitmine ning samade väärtuste korral (20 ja 10) väljastatakse vastus: Keskmine = 15. Kui on tegemist täisarvudega, võib **Val** asemel kasutada funktsiooni **Int**, reaalarvude korral aga funktsiooni **CDbl**.

Makrod demonstreerivad ka võimalikku reaktsiooni juhule, kui kasutaja klõpsas nuppu **Cancel**.

```
Sub Test1()
a = InputBox("a", , 20)
If a = "" Then End
b = InputBox("b", , 10)
If b = "" Then End
c = (a + b) / 2
MsgBox "Keskmine=" & c
End Sub
```

```
Sub Test2()
 a = Val(InputBox("a", , 20))
 If a = 0 Then End
 b = Val(InputBox("b", , 10))
 If b = 0 Then End
 c = (a + b) / 2
 MsgBox "Keskmine=" & c
End Sub
```

Juhtimine

Valikud ja valikulaused

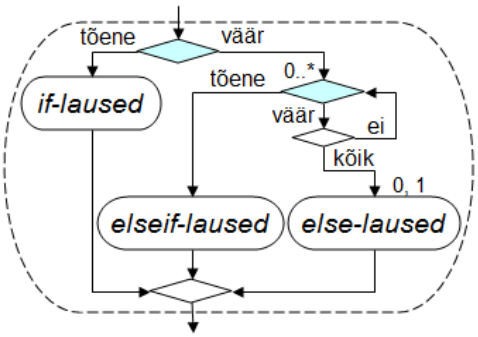
Valikulaused võimaldavad määrata tegevuste (lausete) valikulist täitmist sõltuvalt etteantud tingimustest ja kriteeriumitest. Need kujutavad endast liit- ehk struktuurlauseid, sisaldades teisi liht- ja/või liitlauseid.

VBAs on kaks If-lauset: mitmerealine ja üherealine.

Lausete struktuur ja täitmise põhimõtted

Mitmerealise If-lause üldkuju ja täitmise põhimõte on järgmised:

If *tingimus* **Then**
    [*if_laused*]
[**ElseIf** *tingimus2* **Then**
    elseif_laused
[**Else**
    else_laused]
End If



If-, **ElseIf**-, **Else**- ja **End If**-osalaused peavad olema eraldi ridadel. *Tingimused* esitatakse võrdluste või loogikaavaldiste abil. Laused võivad olla suvalised liht- ja liitlaused, sh ka **If**-laused.

Lause täitmisel kontrollitakse kõigepealt tingimust **If**-osalauses, kui see on tõene, täidetakse *if_laused*, kõik ülejäänu jääb vahele. Vastupidisel juhul kontrollitakse järjest tingimusi **ElseIf**-osalausetes (kui neid on olemas) ning kui leitakse **esimene tõene**, täidetakse järgnevad laused, kõik ülejäänu jääb vahele. Kui ükski tingimus ei ole tõene, täidetakse *else_laused* (kui need on olemas).

Ülaltoodud **If**-lause üldist varianti nimetatakse sageli ka mitmeseks valikuks: mitmest võimalikust tegevuste rühmast valitakse tingimuste alusel välja üks.

Lause üldkujust tulenevad ka kaks sageli kasutatavat varianti, mis võimaldavad määratleda valiku kahest ja valiku ühest.

Kahendvalik

If *tingimus* **Then**
    laused_1
Else
    laused-2
End If

Valik ühest

If *tingimus* **Then**
    laused
End If

Sama tüüpi valikuid võimaldab kirjeldada ka üherealine **If**-lause, mille üldkuju on:

If *tingimus* **Then** *laused_1* [**Else** *laused_2*]

Üherealist lauset tasub kasutada siis, kui sisemised laused on lühikesed lihtlaused. Näiteks:

```
If a > b Then m = a Else m = b
If x > 0 Then k = k + 1: Sp = Sp + x
If y > max Then max = y
If n > nmax Then Exit Do
```

Loomulikult saab toodud laused esitatada mitmerealistena, näiteks esimene lause näeks välja nii:

```
If a > b Then
  m = a
Else
  m = b
End If
```

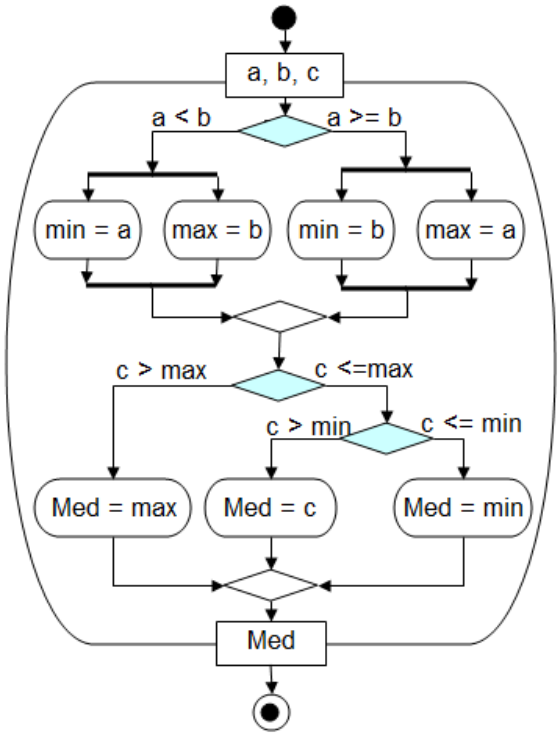
Nagu näha, tuleb siin ridu juurde, sest **If**-, **Else**- ja **End If**-osalaused peavad olema eraldi ridadel. Üherealisel lausel peavad aga kõik elemendid olema ühel real ja **End If**-osalauset ei ole ja ei tohi olla.

Näited

Näide: Kolme arvu mediaan

Leida (valida välja) kolmest arvust suuruse poolest keskmine ehk **mediaan**. Allpool on toodud ülesande lahendamise tegevusskeem (algoritm) ja VBA funktsioon **Med**. Funktsioonil on kolm parameetrit: **a**, **b** ja **c**. Kasutusel on kaks mälupesa (muutujat) **min** ja **max**. Kõigepealt omistatakse muutujale **min** minimaalne väärtus esimesest kahest arvust (**a** ja **b**) ja muutujale **max** maksimaalne väärtus nendest. Edaspidise jaoks ei ole tähtis kumb arvudest (**a** või **b**) on suurem.

Edasi võrreldakse **min** ja **max** väärtusi kolmanda arvuga **c**. Kui **c** on suurem kui **max**, on tegemist olukorraga, kus min ≤ max ≤ c ja **mediaaniks** on **max**. Kui aga c ≤ max, on kaks võimalust. Kui c > min on min ≤ c ≤ max ja **mediaaniks** on **c**, kui ei ole, siis c ≤ min ≤ max ja **mediaaniks** on **min**.




```
Function Med(a, b, c)
  Dim max, min
  If a < b Then
    min = a: max = b
  Else
    min = b: max = a
  End If
  If c > max Then
    Med = max
  ElseIf c > min Then
    Med = c
  Else
    Med = min
  End If
End Function
```

Funktsioonis on kasutusel kaks mitmerealist **If**-lauset. Esimene esindab nö klassikalist kahendvalikut, teises on kasutusel ka **ElseIf**-osa.

```
Function Med1(a, b, c)
  Dim max, min
  If a < b Then min = a: max = b Else min = b: max = a
  If c > max Then Med1 = max Else If c > min Then Med1 = c Else Med1 = min
End Function
```

Võrdluseks on toodud sama ülesande jaoks funktsioon **Med1**, kus kasutatakse kahte üherealist **If**-lauset. Võib tähele panna, et kuigi funktsioon **Med1** on kompaktsem (eeskätt ridade arvude poolest), on selle loetavus ja ülevaatlikus halvem kui eelmisel funktsioonil, seda eriti viimase lause puhul.

```
Function Med2(a, b, c)
  Dim max, min
  If a < b Then min = a: max = b _
          Else min = b: max = a
  If c > max Then Med2 = max _
             Else If c > min Then Med2 = c _
                             Else Med2 = min
End Function
```

Loetavust ja ülevaatlikkust saab pika rea puhul parandada, kasutades käsu jätkumise tunnust: tühik ja allkriips (_). Näiteks on toodud sama funktsiooni jaoks veel üks variant. Funktsioonis **Med2** on kasutusel samuti kaks üherealist **If**-lauset, mis on liigendatud mitmele reale.

Nagu juba varem öeldud, peaks eelistama mitmerealist **If**-lauset.

Näide: Ruutvõrrand

Programm leiab ruutvõrrandi $ax^2 + bx + c = 0$ reaalarvulised lahendid x1 ja x2. Arvestatakse võimalusega, et lahendid võivad puududa, kui diskriminant $D = b^2 – 4ac$ on väiksem kui 0. Kontrollitakse ka, et **a** ei oleks null.

Ruutvõrrandi lahendamine on realiseeritud parameetritega protseduuriga **Ruut_Vrd**. Sisendparameetriteks on võrrandi kordajad **a**, **b** ja **c**, väljundparameetriteks lahendid **x1** ja **x2** (kui need on) ja abimuutuja **tun**. Viimase väärtuseks on 1 (lahendid on olemas) või 0 (lahendid puuduvad).

Kõigepealt leitakse protseduuris diskriminandi **D** väärtus. **If**-lauses kontrollitakse, kas **D** on nullist väiksem, kui jah, võetakse muutuja **tun** väärtuseks 0 ja **Else**-osa laused jäävad täitmata. Vastupidisel juhul võetakse muutuja **tun** väärtuseks 1 ja leitakse **x1** ja **x2** väärtused.

```
Sub Ruut_Vrd(a, b, c, x1, x2, tun)
  Dim D
  D = b ^ 2 - 4 * a * c
  If D < 0 Then
    tun = 0
  Else
    tun = 1
    x1 = (-b - Sqr(D)) / (2 * a)
    x2 = (-b + Sqr(D)) / (2 * a)
  End If
End Sub
```

```
Sub Ruut_Pea()
  Dim a, b, c, x1, x2, tunnus
  a = Range("a")
  If a = 0 Then MsgBox "a ei tohi olla 0!": End
  b = Range("b"): c = Range("c_")
  Ruut_Vrd a, b, c, x1, x2, tunnus
  If tunnus = 0 Then
    x1 = "": x2 = ""
    MsgBox "Lahendid puuduvad!"
  End If
  Range("x_1") = x1: Range("x_2") = x2
End Sub
```

Peaprotseduuris arvestatakse, et võrrandi kordajad on töölehe lahtrites nimedega **a**, **b** ja **c_**. Protseduur loeb väärtused töölehelt ja salvestab vastavates muutujates. Kõigipealt loetakse **a** väärtus ning kui see võrdub nulliga, kuvatakse teade ja programmi töö lõpetatakse. Siin on kasutusel üherealine **If**-lause, milles on kaks lihtlauset: **MsgBox** ja **End**. Need mõlemad täidetakse, kui tingimus on tõene, või jäetakse vahele, kui tingimus ei ole tõene. Peale alamprotseduuri täitmist kontrollitakse **If**-lauses muutuja tunnus väärtust. See saab väärtuse parameetrilt tun ja saab olla 0 või 1. Kui tunnus võrdub nulliga, omistatakse muutujatele **x1** ja **x2** tühi väärtus ning kuvatakse vastav teade. Kui see ei ole 0, jäävad kehtima tagastatud väärtused. Viimased laused kirjutavad töölehe lahtritesse **x1** ja **x2** väärtused, sõltumata sellest, kuidas need tekkisid.

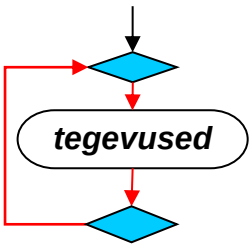
Kordused

VBs on korduste kirjeldamiseks mitu lauset:

- **Do...Loop**-lause
- **For**-lause
- **For Each**-lause
- Lausetel on mitu varianti, eriti kehtib see **Do**…**Loop** lause kohta

Do … Loop-lause. Lõputu kordus

Do
laused
Loop



kordus
tegevused
lõpp kordus

Do ja **Loop** vahel olevate lausetega määratud tegevusi täidetakse põhimõtteliselt lõputult. Korduse saab lõpetada täites protseduuri, milles on **End**-lause või vajutades klahvidele **Ctrl+Break**. Katkestamiseks võib korduse sees olla käsk **Exit Do**.

Näide: Foor 1

Imiteeritakse foori ühe sektsiooni tööd. Töölehel on graafikaobjekt (näiteks ring) nimega **tuli** ning käsunupud **Tööle** ja **Stopp**. Protseduuri **Tööle** moodustab lõputut kordust määrav **Do**…**Loop**-lause. Pidevalt toimub pöördumine protseduuri **Muuda** poole, mis muudab **tule** värvi. Parameetriteks ja argumentideks on värvi number ja põlemise kestus. Foori saab välja lülitada, käivitades protseduuri **Aitab**.

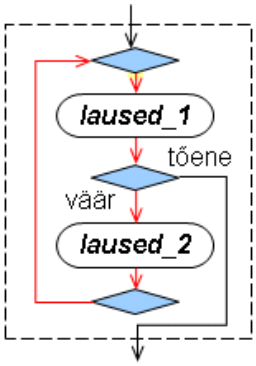
```
Sub Tööle ()
  Do
    Muuda  2, 5  ' punane, 5 sek
    Muuda  5, 2  ' kollane, 2 sek
    Muuda  3, 4  ' roheline, 4 sek
    Muuda  5, 2  ' kollane, 2 sek
  Loop
End Sub
```

```
Sub Muuda (värv, p)
   Shapes("tuli").Fill.ForeColor.SchemeColor = värv
   paus p
End Sub
Sub Aitab ()
   Muuda 0, 1 ' must
  End
End Sub
```

```
Sub paus(pp)
  Dim pl
  pl = Timer() + pp
  Do
    DoEvents
  Loop While Timer() < pl
End Sub
```

Do … Loop-lause. Lõputu kordus katkestusega

Do
laused_1
If *tingimus* **Then Exit Do**
laused_2
Loop



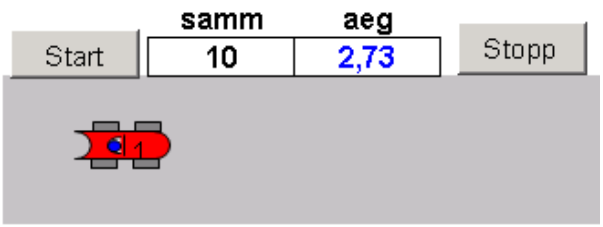
kordus
tegevused 1
kui tingimus **siis** välju
tegevused 2
lõpp kordus

Korduse moodustab **Do**…**Loop**-lause. Selle sees peab olema vähemalt üks valikulause, mis sisaldab katkestamiskäsku **Exit Do** (võib olla ka **Exit Sub** või **Exit Function**). Igal kordamisel kontrollitakse tingimust. Kui see on tõene, katkestab **Exit Do** korduse ja täitmine jätkub **Do…Loop**-lausele järgnevast lausest. Laused **Exit Sub** või **Exit Function** katkestavad vastava protseduuri töö.

Katkestamise lauseid võib olla rohkem kui üks ning tingimus võib olla antud üldisemal kujul.

NB! Scratchis lõpetab käsk [**peata skript**] skripti töö, aga ainult korduse katkestamiseks käsku ei ole.

Näide: Auto 1



Töölehel on kujundid nimedega **auto** ja **fin**. Programmi toimel „sõidab“ auto töölehe vasakust servast finišijooneni (fin). Kasutaja saab töölehe lahtris anda sammu, millest sõltub liikumise kiirus. Programm kirjutab lahtrisse **aeg** sõitmise aja.

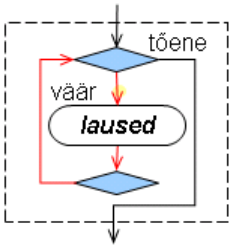
```
Sub Auto_1()
  Dim auto As Shape, algaeg, h
  Set auto = Shapes("auto")
  h = Range("samm")
  auto.Left = 0: algaeg = Timer()
  Do
     Range("aeg") = Timer() - algaeg
     auto.IncrementLeft h / 2 + Rnd() * h
     If auto.Left > Shapes("fin").Left Then Exit Do
     paus 0.01
  Loop
  MsgBox "Olen kohal!"
End Sub
```

Lahtrist samm loetakse väärtus ja omistatakse muutujale ***h***. Auto vasaku serva väärtuseks võetakse null ning muutuja algaeg väärtuseks taimeri jooksev väärtus.

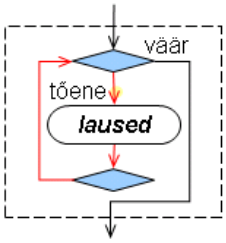
Põhiosa tegevustest toimub **Do**…**Loop** korduses. Igal kordamisel kirjutatakse lahtrisse aeg jooksev aeg. Auto vasaku serva koordinaati muudetakse juhuarvu abil, kasutades muutuja h väärtust. **If**-lauses kontrollitakse, kas auto vasak serv on suurem finišijoone vasakust servast. Kui jah, katkestatakse kordamine ja kuvatakse teade „Olen kohal!“

Do … Loop-lause. Eelkontrolliga ja järelkontrolliga kordused

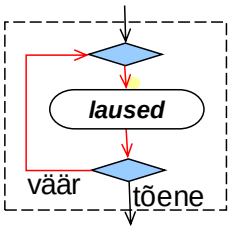
Do Until *tingimus*
  laused
Loop



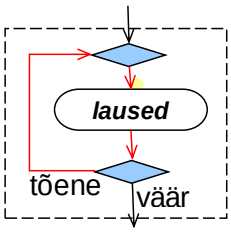
Do While *tingimus*
  laused
Loop



Do
  laused
Loop Until *tingimus*



Do
  laused
Loop While *tingimus*



Tegemist on nelja väga sarnase korduste määratlemise lausega. Kui **Do**-osalauses kasutatakse täiendit **Until**, toimub kordamine seni, kuni tingimus saab tõeseks. Täiendi **While** korral toimub kordamine, kui tingimus on tõene. Järelkontrolliga lausetes täidetakse grupis olevaid lauseid vähemalt üks kord. Eelkontrolliga korduses võivad need jääda ka täitmata, kui tingimus on **Until**-tüüpi korduses kohe tõene või väär **While**-tüüpi korduse korral. Milline variant valida, sõltub sageli programmi koostaja soovist. Lausete seas võib olla üks või mitu **katkestamislauset**: Exit Do, Exit Sub või Exit Function.

Toodud näidetes on protseduuris **Auto_2** kasutusel eelkontrolliga Until-kordus, protseduuris **Auto_3** on järelkontrolliga **Do While**-kordus. Esimesel juhul, kui auto on juba sihtkohas, ei liigu see programmi käivitamisel paigast: tingimus on kohe tõene. Teisel juhul samas olukorras teeb auto ühe sammu ja siis selgub, et tingimus on väär ja kordus lõpetatakse.

```
Sub Auto_2()
  Dim auto As Shape, algaeg, h
  Set auto = Shapes("auto")
  h = Range("samm")
  auto.Left = 0: algaeg = Timer()
  Do Until auto.Left  >= Shapes("fin").Left
     Range("aeg") = Timer() - algaeg
     auto.IncrementLeft h / 2 + Rnd() * h
     paus 0.01
  Loop
 End Sub
```

```
Sub Auto_3()
  Dim auto As Shape, algaeg, h
  Set auto = Shapes("auto")
  h = Range("samm")
  auto.Left = 0: algaeg = Timer()
  Do
     Range("aeg") = Timer() - algaeg
     auto.IncrementLeft h / 2 + Rnd() * h
     paus 0.01
  Loop While auto.Left  <  Shapes("fin").Left
End Sub
```

Näide: Palli liikumine

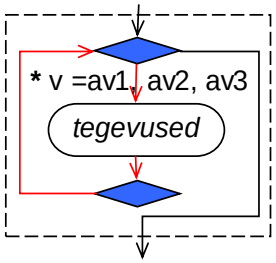
Protseduurid, mis juhivad palli lendu üles aeglustusega (kuni kiirus jõuab nulli) ning kukkumist kiirendusega kuni pall jõuab maale.

```
Sub Yles()
  ' Liikumine üles aeglustusega
  Dim h, d, pall As Shape
  Set pall = Shapes("pall")
  h = 30: d = 1
  Do Until h <= 0 Or pall.Top <= 0
     pall.IncrementTop -h
     h = h - d
     paus 0.1
  Loop
 End Sub
```

```
Sub Alla()
  ' Liikumine alla kiirendusega
  Dim h, d, pall As Shape
  Set pall = Shapes("pall")
  h = 0: d = 1
  Do While pall.Top < Shapes("maa").Top
     pall.IncrementTop h
     h = h + d
     paus 0.1
  Loop
End Sub
```

For ... Next-lause. Juhtmuutujaga kordus

```
For v = av1 To av2 [ Step av3 ]
    laused_1
   [Exit For]
    laused_2
Next v
```




```
kordus v = a1..a2 [,a3]
    tegevused
lõpp kordus
```

v on nn juhtmuutuja. **av1**, **av2** ja **av3** võivad üldjuhul olla avaldised. **av1** on juhtmuutuja algväärtus, **av2** lõppväärtus, **av3** muutmise samm. Kui sammu ei ole antud, võetakse selle väärtuseks 1.

Näide: Auto ringliiklus

Programm imiteerib auto ringliiklust. Kasutusel on kolm graafikaobjekti auto, Juku ja Kraps, kolm lahtrit: ringe, ring ja aeg ning neli muutujat: auto, ringe, ring ja algaeg.

```
Sub Auto_4()
  Dim auto As Shape, ringe, ring, algaeg
  Set auto = Shapes("auto")
  auto.Left = 0
  ringe = Range("ringe") ' ringide arv
  algaeg = Timer()
  For ring = 1 To ringe  ' For-lause algus
    Range("ring") = ring  ' ringi number
    Do Until auto.Left > 1000
      Range("aeg“) = Timer() - algaeg
      auto.IncrementLeft  20 + Rnd() * 10
      paus 0.02
    Loop
    auto.Left = 0
  Next ring                ' For –lause lõpp
  Hyppa Shapes("Juku"), 3 , 40
  Hyppa Shapes("Kraps"), 5 , 30
End Sub
```

Auto viiakse töölehe vasakusse serva. Töölehe lahtrist **ringe** loetakse väärtus ja omistatakse muutujale **ringe**. Seda kasutatakse **For**-korduses juhtmuutuja **ring** lõppväärtusena.

Igal kordamisel liigub auto töölehe vasakust servast 1000 punkti kaugusele. Ringi number (juhtmuutuja väärtus), kirjutatakse lahtrisse **ring**. Liikumise juhtimiseks kasutatakse **Do Until**-kordust. Jooksev aeg kirjutatakse lahtrisse **aeg**.

```
Sub Hyppa(kuju, n, h)
Dim i
For i = 1 To n
  kuju.IncrementTop -h
  paus 0.1
  kuju.IncrementTop h
  paus 0.1
Next i
End Sub
```

Peale sõidu lõppu teevad Juku ja Kraps mõned hüpped protseduuri **Hyppa** toimel, kus kasutatakse samuti **For-**kordust.

Näide: Naturaalarvude ruutude summa

```
Function Rea_Sum(n1, n2)
  Dim k, S
  S = 0
  For k = n1 To n2
    S = S + k ^ 2
  Next k
  Rea_Sum = S
End Function
```

Funktsioon **Rea_Sum** leiab naturaalarvude ruutude summa alates algväärtusest **n1** kuni **n2**. Algus ja lõpp antakse parameetritele vastavate argumentidega pöördumisel.

Põhiosa funktsioonist moodustab juhtmuutujaga kordus. Kõigepealt võetakse muutuja ***S*** algväärtuseks 0. **For**-lauses muudetakse juhtmuutuja ***k*** väärtust alates **n1**-st **n2**-ni ning liidetakse järgmise väärtuse ruut summale. Vastavalt VB reeglitele, omistatakse tagastatav väärtus funktsiooni nimele.

Näide „Ühemuutuja funktsiooni väärtuste aritmeetiline keskmine“

Allpool on toodud kaks funktsiooni, mis võimaldavad leida ühemuutuja funktsiooni **y = F(x)** väärtuse lõigul [a, b]. Lõik jagatakse **n** võrdseks osaks pikkusega **h** = (b – a)/n. Keskmise saamiseks leitakse funktsiooni väärtuste summa ja jagatakse punktide arvuga **n+1**. Funktsiooni väärtuste $\mathbf{y_i}$ leidmisel muudetakse **x** väärtusi sammuga **h** alates **a** väärtusest kuni **b** väärtuseni. Eeldatakse, et funktsiooni väärtuse leidmiseks antud **x** väärtuse jaoks on olemas VB funktsioon F(x).

```
Function Fkesk_1(a, b, h)
  Dim  S, x, n
  S = 0 : n = 0
  For x = a To b + h / 2 Step h
    S = S + F(x)
    n = n + 1
  Next x
  Fkesk_1 = S / n
End Function
```

```
Function Fkesk_2(a, b, n)
  Dim i, S, h, x
  h = (b - a) / n: S = 0
  For i = 0 To n
    x = a + i * h
    S = S + F(x)
  Next i
  Fkesk_2 = S / (n + 1)
End Function
```

Funktsioonis **Fkesk_1** on parameetriteks otspunktid **a** ja **b** ning samm **h**.

Funktsioonis **Fkesk_2** on sammu asemel jaotiste arv **n**.

Funktsioonis **Fkesk_1** on **For-**lause juhtmuutujaks reaalarvuline suurus **x**, mille algväärtus on **a**, lõppväärtus **b** ja muutumise samm on **h**.

Kuigi teoreetiliselt on **x** lõppväärtus **b**, siis praktiliselt on selleks võetud **b + h/2**, st pool sammu rohkem. Asi on selles, et avaldisega **(b–a)/n** leitav sammu **h** tegelik väärtus saab tulla ka veidi erinev teoreetilisest, milleks võib olla lõpmatu murd. Selle tõttu võib **x** väärtuse suurendamisel korduses tekkida olukord, kus viimane väärtus, mis peaks teoreetiliselt olema võrdne **b**-ga, on sellest natuke suurem ja sellisel juhul viimast juhtmuutuja väärtust ei kasutata. Näiteks praegu jääks viimane punkt keskmise arvutamisest välja. Poole sammu lisamine lõppväärtusele tagab, et viimane väärtus leiab igal juhul kasutamist.

Näide: Ühemuutuja funktsiooni maksimaalne väärtus

```
Function Fmax_1(a, b, n)
  Dim h, x, y, maks, i
  h = (b - a) / n
  maks = F(a)
  For i = 1 To n
    x = a + i * h
    y = F(x)
    If y > maks Then maks = y
  Next i
  Fmax_1 = maks
End Function
```

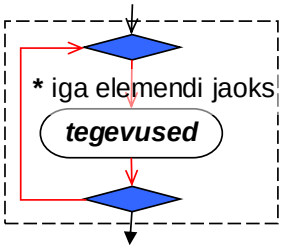
Funktsioon **Fmax_1** leiab etteantud ühemuutuja funktsiooni maksimaalse väärtuse antud lõigul [a; b]. Funktsiooni parameetriteks on lõigu otsapunktid a ja b ning jaotiste arv n.

Maksimumi algväärtuseks võetakse funktsiooni väärtus vasakus otsapunktis **a**. **For**-lauses arvutatakse järjest funktsiooni väärtusi jaotuspunktides. Iga leitud **y** väärtust võrreldakse maksimumi jooksva väärtusega (maks) ja kui **y** väärtus on suurem, võetakse see uueks maksimumi väärtuseks. Viimane muutujale **maks** omistatud väärtus ongi tulemuseks ja see omistatakse funktsiooni nimele.

For Each-lause

For Each-lause võimaldab täita mingid tegevused antud kogumi iga elemendi jaoks. Kogumiks võib olla Exceli objektide kollektsioon (lahtriplokk, töölehed, kujundid jne) või massiiv.

```
For Each element In kogum
  laused_1
  [Exit For]
  laused_2
Next element
```




```
kordus Iga elemendi jaoks kogumis
  tegevused
lõpp kordus
```

Kogumi element esitatakse objektimuutuja või **Variant**-tüüpi muutuja abil. Lausete seas võib olla üks või mitu **Exit For**-lauset (reeglina mingi valikulause sees), mis võimaldavad katkestada korduse ja anda täitmisjärje järgmisele lausele enne kordamiste nö normaalset lõppu. Nii korduse kui ka protseduuri või programmi lõpetamiseks võib kasutada ka lauseid **Exit Sub**, **Exit Function** või **End**.

Täitmine: **For Each**- ja **Next**-osalausete vahel olevaid lauseid täidetakse iga kogumi elemendi korral, kui mõni lause nagu **Exit For**, **End** vmt, ei katkesta täitmist.

Näide: Lahtriploki elementide summa

```
Function SumP(prk As Range)
  Dim lahter As Range, S
  S = 0
  For Each lahter In piirkond
    S = S + lahter.Value
  Next lahter
  SumP = S
End Function
```

Funktsioon **SumP** leiab etteantud piirkonna (lahtriploki) lahtrite väärtuste summa. Piirkond esitatakse parameetri prk abil, mille tüübiks on Range.

Funktsioonis on määratletud kaks muutujat:

lahter – Range-tüüpi objektimuutuja viitamiseks piirkonna igale lahtrile,

S – lihtmuutuja summa jooksva ja lõppväärtuse salvestamiseks.

Summa kogumiseks kasutatakse **For Each**-lauset. Igal kordamisel liidetakse järgmise lahtri väärtus summale – muutuja S väärtusele.

Eespool toodud funktsiooni võib esitada ka lihtsamalt, saavutades samal ajal suurema universaalsuse.

```
Function SumPM (kogum)
  Dim element, S
  S = 0
  For Each element In kogum
    S = S + element
  Next element
  SumPM = S
End Function
```

Funktsioon **SumPM** võimaldab leida nii Exceli töölehe piirkonna kui ka VB massiivi elementide summa.

Siin on parameetri kogum tüübiks **Variant**. Sellele vastavaks argumendiks võib olla nii Exceli töölehe lahtriplokk kui ka VB massiiv. Samuti on siin **Variant**-tüüpi ka muutuja element. Lauses S = S + element on jäetud ära omadus Value. See ei olnud vajalik ka eelmises funktsioonis, kuna lahtriploki jaoks võetakse see vaikimisi.

Näide: Maksimaalne element piirkonnas või massiivis

```
Function MaxPM(kogum)
 Dim elem, maks
 maks = -10 ^ 20
 For Each elem In kogum
   If elem > maks Then maks = elem
 Next elem
 MaxPM = maks
End Function
```

Funktsioon MaxPM võimaldab leida maksimaalse väärtuse Exceli lahtriplokis või VBA massiivis.

Parameeter **kogum** on **Variant**-tüüpi. Sama tüüpi on ka muutuja **elem**, mida kasutatakse viitamiseks piirkonna või massiivi elementidele, ja muutuja **maks**, mida kasutatakse maksimaalse elemendi jooksva ja lõppväärtuse salvestamiseks. Maksimumi **maks** algväärtuseks võetakse siin väga väike arv (-10^{20}), mis on kindlasti väiksem kõigist väärtustest massiivis. Alternatiiviks on võtta maksimumi algväärtuseks kogumi esimene element.

For Each-lauses võrreldakse igat elementi elem maksimumi jooksva väärtusega maks ja kui mingi elemendi väärtus on sellest suurem, võetakse element uueks maksimumi väärtuseks: omistatakse muutujale maks, asendamaks eelmist väärtust.

Näide: Väärtuse asukoha otsimine vektoris

Funktsioon Otsi_Nr leiab otsitava väärtusega võrdse väärtuse järjenumbri antud vektoris Vek. Kui otsitavaga võrdset väärtust vektoris ei ole, tagastab funktsioon väärtuse 0.

Funktsioonil on kaks parameetrit: **x** – otsitav väärtus, **Vek** – vektor, milles toimub otsimine

```
Function Otsi_Nr(x, Vek)
 Dim k, elem
 k = 1
 For Each elem In Vek
   If elem = x Then
       Otsi_Nr = k
       Exit Function
   End If
   k = k + 1
 Next elem
 Otsi_Nr = 0
End Function
```

Otsitava väärtuse järjenumbri k algväärtuseks võetakse 1. **For Each**-lause igal kordamisel võrreldakse jooksvat elementi elem otsitava väärtusega **x**.

Kui need on võrdsed, võetakse **k** jooksev väärtus funktsiooni väärtuseks ja katkestatakse nii **For Each**-lause kui ka funktsiooni täitmine.

Vastupidisel juhul suurendatakse **k** väärtust ühe võrra ja jätkatakse kordamist. Kui kõik elemendid on läbi vaadatud ja ükski ei võrdunud otsitavaga, lõpeb kordamine ja tulemuseks võetakse 0.

Näide: Graafikaobjektide asukoha muutmine

For Each-lause kasutamisel võib tekkida olukord, kus lauses määratavaid tegevusi ei pea täitma kõikide antud kollektsiooni liikmetega. Eriti tüüpiline on see graafikaobjektide kollektsiooni (klassi) Shapes puhul, sest sellesse kollektsiooni kuuluvad kõik töölehe pinnal paiknevat objektid nagu diagrammid, joonised, pildid, ohjurid (sh käsunupud) ja isegi lahtrite kommentaarid.

Vajalike objektide valimiseks võib kasutada erinevaid tingimusi:

- objektid asuvad mingis kindlaks alas (piirkonnas),
- kuuluvad kindlasse liiki (joonis, pilt, ohjur vms),
- nende nimed algavad mingite kindlate märkidega vms.

Makros **Liikuge** eeldatakse, et töölehe kindlas piirkonnas, ristkülikus nimega **plats**, asub mingi hulk suvalist tüüpi graafilisi kujundeid. Nende sees on ka ovaalid või ringid, mille nimed algavad tekstiga „Oval“.

```
Sub Liikuge()
  Dim kuju As Shape, plats As Shape, x, y
  Set plats = Shapes("plats")
  Do
    For Each kuju In Shapes
      If On_Sees(kuju, plats) And _
        Left(kuju.Name, 4) = "Oval" Then
        x = plats.Left + Rnd() * (plats.Width - kuju.Width)
        y = plats.Top + Rnd() * (plats.Height - kuju.Height)
        kuju.Left = x: kuju.Top = y
        paus 0.1
      End If
    Next kuju
  Loop
```

Makro **Liikuge** muudab juhuarvude abil ainult ovaalide asukohta.

For Each-lause on paigutatud lõpmatut kordust määrava **Do**...**Loop**-lause sisse. Programmi töö katkestamiseks on protseduur, mis sisaldab **End**-lauset.

For Each-lause määrab korduse üle kõikide kujundite töölehel – kollektsioon Shapes. Sisemine **If**-lause valib aga ainult need, mis asuvad kujundi **plats** sees ja mille nimede alguses on „Oval“. Liigutatavate kujundite asukoht määratakse juhuslike arvudega, arvestades ka nende mõõtmeid, nii et need jääksid platsi piiridesse.

Massiivid

Järgnevalt vaadeldakse massiivide olemust ja põhiomadusi, massiivide deklareerimist ning viitamist massiivi elementidele.

Massiivide olemus ja põhiomadused

Massiiv ehk massiivmuutuja on ühetüübiliste elementide (väärtuste) järjestatud kogum. Massiiv tähistatakse ühe nimega. Pöördumiseks tema üksikelemendi poole kasutatakse nime koos indeksitega, mis näitavad elemendi asukohta massiivis. Massiivi igale elemendile eraldatakse arvuti mälus eraldi väli (pesa). Massiivi põhiomadused on:

- nimi
- dimensioon
- indeksite rajad
- elementide tüüp
- dünaamilisus

Massiivi nime esitamiseks kehtivad samad reeglid nagu teiste nimede jaoks.

Dimensioon ehk mõõt iseloomustab massiivi organisatsiooni – määrab elementide paigutuse kogumis ning indeksite arvu, mis on vajalik elementide asukoha määramiseks. **Ühemõõtmeline** massiiv on näiteks nummerdatud elementide jada, kus elemendi asukoha jadas saab määrata ühe indeksi (järjenumbri) abil. Sellist massiivi nimetatakse sageli ka vektoriks või jadaks. Selle analoogiks Excelis on töölehe rida või veerg või nende osad. Scratchis vastab sellele loend. **Kahemõõtmelises** massiivis, mida nimetatakse ka maatriksiks, moodustavad elemendid tabeli. Elemendi asukoht on üheselt määratav kahe indeksi abil, milleks on rea- ja veerunumber. Excelis vastab kahemõõtmelisele massiivile tööleht või ristkülikukujuline lahtriplokk. Võib kasutada ka kolme ja enama dimensiooniga massiive. Maksimaalne dimensioonide arv VBAs võib olla 60!

Indeksite rajad näitavad iga dimensiooni jaoks indeksi minimaalse ja maksimaalse väärtuse. Indeksi minimaalseks väärtuseks võib põhimõtteliselt olla suvaline arv, kuid enamasti on selleks 0 või 1. Indeksi maksimaalne väärtus ei ole piiratud.

Elementide tüüp määrab nende esitusviisi ehk vormingu. Formaalselt peab massiivi kõigil elementidel olema ühesugune tüüp. Kui selleks on **Variant**, võib ühes massiivis säilitada erinevat liiki väärtusi – arve, tekste, kuupäevi jms.

Dünaamilisus. Selle omaduse järgi jagunevad massiivid kahte liiki:

- fikseeritud massiivid
- dünaamilised massiivid

Fikseeritud massiivile eraldatakse **koht** (pesad) **mälus enne** selle **protseduuri täitmist**, kus massiiv on määratletud, ja selle **rajad peavad** olema määratud **konstantide abil**. **Dünaamilisele** massiivile eraldatakse **mälu protseduuri täitmise ajal** ning massiivi **rajade** määramiseks **võib** kasutada **muutujaid** ja **avaldisi**.

Enamasti kasutatakse dünaamilisi massiive, sest nende abil saab luua universaalsemaid ja paindlikumaid programme, mis ei sõltu massiivide mõõtmetest.

Massiivide deklareerimine

Massiivide omadused määratakse deklaratsiooniga. Kõik massiivi muutujad peavad tingimata olema deklareeritud. Deklareerimisviis sõltub teatud määral sellest, kas on tegemist fikseeritud või dünaamilise massiiviga.

Fikseeritud massiive kirjeldatakse **Dim**-lausega:

Dim nimi (rajad, rajad …) [**As** tüüp] …

Rajad võib deklaratsioonis esitada kujul [min **To** max], kus **min** on dimensiooni indeksi minimaalne väärtus (alumine raja) ja **max** sama indeksi maksimaalne väärtus (ülemine raja). Need väärtused **peavad** olema esitatud täisarvuliste **konstantide abil**. Rajade arv näitab massiivi dimensiooni. Kui indeksi minimaalväärtust ei ole näidatud, võetakse selleks null. Korraldusega **Option Base 1** mooduli alguses saab määrata, et vaikimisi võetav indeksite minimaalne väärtus moodulis on 1.

Massiivi elementide tüüp määratakse samade **võtmesõnade** abil nagu lihtmuutujatel: **Integer**, **Double** vms. Võib kasutada ka **tüübitunnuseid**: %, # jne. Kui tüüp puudub, siis valitakse selleks **Variant**.

Deklaratsioonide näiteid

Dim X(100) **As Long**, Y(100) **As Double**, nimed(300), V(1 **To** 500)

Lauses on deklareeritud neli ühemõõtmelist massiivi. Massiivides **X** ja **Y** on mõlemas 101 elementi (i = 0, 1, ... 100), elementide tüübiks on esimeses massiivis **Long**, teises **Double**. Massiivis **nimed** on 301 elementi ja massiivis V 500 elementi. Mõlema massiivi elementide tüübiks on **Variant**.

Dim T%(5, 8), A(1 **To** 20, 1 **To** 30) **As Double**, palgad(1 **To** 1000, 1 **To** 8)

Selles lauses on deklareeritud kolm kahemõõtmelist massiivi. Massiivis **T** on 6 rida (rn = 0, 1, ... , 5) ja 9 veergu (vn = 0, 1, ... , 8), elementide tüübiks on **Integer**. Massiivis **A** on 20 rida ja 30 veergu, elementide tüüp on **Double**. Massiivis palgad on 1000 rida ja 8 veergu.

Dünaamilisi massiive võib deklareerida korduvalt. Alguses, kui massiivi mõõtmed ei ole veel teada, võib selle deklareerida **Dim**-lausega, milles dimensioon ja rajad ei ole määratud ning nime järel on tühjad sulud

Dim Kor( ), V( ), W( )

Taolise deklaratsiooni alusel massiivile mälus kohta ei eraldata ja seda ei saa kasutada. Dünaamilise massiivi dimensiooni ja rajade määramine ning mälu eraldamine toimub **ReDim**-lausega

ReDim nimi (rajad, rajad, …) [**As** tüüp] …

Massiivi deklaratsiooni kuju on sama nagu **Dim**-lauses, ainult siin **võib rajade määramisel kasutada muutujaid ja avaldisi**.

Näiteks määrab järgmine lause

ReDim V(2 * k + 1), W(2 * k + 1), Kor(1 **To** m, 1 **To** n)

massiivide **V**, **W** ja **Kor** mõõtmed ja eraldab neile mälu, kui eelnevalt on muutujatele **k**, **m** ja **n** omistatud väärtused.

Viitamine massiivi elementidele ja massiividele

Viitamine massiivi elementidele toimub kujul

nimi (*indeks*, *indeks*, …)

Selles on **nimi** on massiivi nimi ja **indeks** konstant, muutuja või avaldis, mis näitab vastava indeksi väärtust. Indeksite arv peab võrduma massiivi dimensiooniga ja indeksite väärtused peavad olema deklaratsioonis määratud rajade piirides.

Näiteid eespool deklareeritud massiivide jaoks:

X(0), X(13), Y(100), X(i), Y(2 * j - 1), T(0, 0), A(i, j), A(1, k + 2)

Massiivi elementidele võib omistada väärtusi ja neid võib kasutada avaldistes analoogselt lihtmuutujatega. Näiteks leiavad laused

```
    For i = 0 To n
      Y(i) = Sin(3 * X(i))^2 - Cos(X(i) / 2) + 2 * X(i)
    Next i
```

funktsiooni y = $\sin^2(3x)$ - cos(x/2) + 2x väärtused ja salvestavad need massiivi **Y**, kasutades muutuja väärtusi massiivis **X**.

```
Sub Kopi(A(), B(), m, n)
  Dim i, j
  For i = 1 To m
    For j = 1 To n
      B(i, j) =  A(i, j)
    Next j
  Next i
End Sub
```

Protseduur **Kopi** omistab elementhaaval maatriksi **A** elemendid maatriksi **B** elementidele (omistab maatriksi **A** maatriksile **B** ehk kopeerib maatriksi **A** maatriksisse **B**). Protseduurile edastatakse parameetrite abil maatriksid ja nende mõõtmed: **m** – ridade arv, **n** – veergude arv. Kuna **A** ja **B** on kahemõõtmelised massiivid, kasutatakse viitamiseks nende elementidele kahte indeksit: **i** – rea number, **j** – veeru number.

Massiivi elementidele saab viidata ka **For Each**-lause muutuja abil.

```
Function Pos_Kesk(A)
  Dim elem, S, n
  S = 0: n = 0
  For Each elem In A
    If elem > 0 Then  S = S + elem : n = n + 1
  Next element
  If n = 0 Then   Pos_Kesk = 0 _
          Else Pos_Kesk = S / n
End Function
```

Funktsioon **Pos_Kesk** võimaldab leida suvalise massiivi positiivsete elementide aritmeetilise keskmise.

Protseduuri parameeter on **Variant**-tüüpi ja pöördumisel võib sellele vastata suvalise dimensiooni ja mõõtmetega massiiv.

Enam veel – seda funktsiooni saab kasutada ka töölehe lahtriploki summa leidmiseks.

For Each-lauses saab massiivi elemente ainult kasutada (lugeda), kuid mitte muuta.

Viitamiseks tervele massiivile kasutatakse massiivi nime indeksiteta, kuid seda saab teha ainult teatud erijuhtudel:

- Parameetrites ja argumentides. Nimele võivad järgneda ka tühjad sulud.
- Ühe massiivi omistamisel teisele. Massiivide dimensioonid, rajad ja tüübid peavad olema ühesugused. Näiteks ülaltoodud protseduuri asemel võiks kasutada hoopis ühte lihtsat omistamislauset: B = A.
- Massiivi omistamisel töölehe lahtriplokile (piirkonnale) või vastupidi.

Näide: Operatsioonid vektoritega

Programmid **Mas_1** ja **Mas_2** teevad põhimõtteliselt seda sama – genereerivad funktsiooni Rnd() abil juhuslikest arvudest (1 kuni 100) kaks vektorit (ühemõõtmelist massiivi) **X** ja **Y**, liidavad elementhaaval need vektorid, saades tulemuseks vektori **Z**. Kõik kolm vektorit kirjutatakse töölehele, kasutades baaslahtrina lahtrit nimega **V_alg**. Lõpuks leiavad mõlemad programmid vektori **Z** elementide aritmeetilise keskmise ja kirjutavad selle töölehele.

Väliselt erinevad programmid suhteliselt vähe, kuid sisuline erinevus on põhimõtteline. Esimeses programmis kasutatakse **fikseeritud** massiive, teises **dünaamilisi**. Programmis **Mas_1** kasutatakse nii massiivide rajade määramisel kui ka **For**-lausetes juhtmuutuja **i** lõppväärtuse esitamisel nimega konstanti **n**, mille väärtuseks on võetud 10. Elementide arvu muutmiseks peab muutma programmi. Programmis **Mas_2** loetakse vektorite elementide arv **n** töölehelt. Seega võib igal täitmisel ette anda suvalise arvu elemente programmi muutmata. Vektorite kirjutamiseks kasutatakse selles programmis kompaktsemat varianti.

```
Sub Mas_1()
' Fikseeritud massiivid
Const n = 10
Dim X(n), Y(n), Z(n), i, S, kesk
 ' Vektorite genereerimine
For i = 1 To n
  Y(i) = Int(Rnd() * 100) + 1
  X(i) = Int(Rnd() * 100) + 1
Next i
' Vektorite liitmine
For i = 1 To n
   Z(i) = Y(i) + X(i)
Next i
' Vektorite kirjutamine töölehele
For i = 1 To n
   Range("V_alg").Cells(i, 1) = X(i)
   Range("V_alg").Cells(i, 2) = Y(i)
   Range("V_alg").Cells(i, 3) = Z(i)
Next i
' Keskmise leidmine
S = 0
For i = 1 To n
   S = S + Z(i)
Next i
kesk = S / n
Range("keskmine") = kesk
End Sub
```

```
Sub Mas_2()
' Dünaamilised massiivid
 Dim i, n, S, kesk, prk
 n = Range("elemente")
 ReDim X(n), Y(n), Z(n)
 ' Vektorite genereerimine
 For i = 1 To n
   Y(i) = Int(Rnd() * 100) + 1
   X(i) = Int(Rnd() * 100) + 1
 Next i
 ' Vektorite liitmine
 For i = 1 To n
    Z(i) = Y(i) + X(i)
 Next i
 ' Vektorid töölehele
 Set prk = Range("V_alg")
 For i = 1 To n
    prk(i, 1) = X(i)
    prk(i, 2) = Y(i)
    prk(i, 3) = Z(i)
 Next i
 ' Keskmise leidmine
 S = 0
 For i = 1 To n
    S = S + Z(i)
 Next i
 kesk = S / n
 Range("keskmine") = kesk
End Sub
```

```
Sub Mas_3()
Dim n, kesk, prk As Range
n = Range("elemente")
ReDim X(n), Y(n), Z(n)
Tee_Vek X(), n, 1, 100
Tee_Vek Y(), n, 1, 100
Liida_Vek X, Y, Z, n
Set prk = Range("prk")
prk.ClearContents
Kir_Vek X, n, prk(1, 1)
Kir_Vek Y, n, prk(1, 2)
Kir_Vek Z, n, prk(1, 3)
kesk = Kesk_Vek(Z, n)
Range("keskmine") = kesk
End Sub
```

```
Sub Tee_Vek(V(), n, a, b)
 Dim i
 For i = 1 To n
   V(i) = a+Int(Rnd()* (b - a + 1))
 Next i
End Sub
```

```
Sub Liida_Vek(V1, V2, V3, n)
  Dim i
  For i = 1 To n
    V3(i) = V1(i) + V2(i)
  Next i
End Sub
```

```
Function Kesk_Vek(V, n)
  Dim i, S
  For i = 1 To n
    S = S + V(i)
  Next i
  Kesk_Vek = S / n
End Function
```

```
Sub Kir_Vek(V, n, prk)
  Dim i
  For i = 1 To n
    prk(i) = V(i)
  Next i
End Sub
```

Näide: Maatriksi ridade aritmeetiline keskmine

```
Sub Mas_3()
  Dim m, n, i, j, S, prk As Range
  m = Range("ridu") :  n = Range("veerge")
  ReDim A(1 To m, 1 To n), KV(1 To m)
  For i = 1 To m :  For j = 1 To n
     A(i, j) = Int(Rnd() * 100) + 1
  Next j  :  Next i
  For i = 1 To m
    S = 0
    For j = 1 To n : S = S + A(i, j) :  Next j
    KV(i) = S / n
  Next i
  Range("M_alg").Resize(m, n).Value = A
  For i = 1 To m
     Range("M_alg").Cells(i, n + 2) = KV(i)
  Next i
End Sub
```

Programm **Mas_3** leiab ristkülikmaatriksi **A** iga rea elementide aritmeetilise keskmise ja salvestab saadud keskmised vektoris **KV**. Kasutatakse dünaamilisi massiive. Maatriksi ridade ja veergude arv loetakse töölehelt.

Massiivide deklareerimisel on **ReDim**-lauses alumised rajad esitatud ilmutatud kujul. Andmete kirjutamisel töölehele määratakse maatriksi ja vektori asukoht töölehel ühe lahtri **M_alg** abil.

Lausega

Range("M_alg").Resize(m, n).Value = A

omistatakse kahemõõtelise massiivi A väärtused sobivate mõõtmetega lahtriplokile kasutades lahtriploki omadust Resize(m, n).

Vektori asukoht töölehel määratakse samuti lahtri **M_alg** suhtes. Vektor kirjutatakse töölehe veergu, mis asub n + 2 lahtri võrra sellest lahtrist paremal. Vektori esimene element kirjutatakse töölehe samale reale, millel on lahter **M_alg**.

Massiivide kasutamine parameetrite ja argumentidena

Massiivide kasutamist parameetrina esines ka eespool olnud näidetes, kuid seal ei peatutud nende esitamise reeglitel ega vaadeldud massiivide kasutamist argumentidena. Käesolevas jaotises esitatakse ülevaade nendest küsimustest.

Parameetermassiiv deklareeritakse protseduuri päises järgmiselt: nimi [( )] [**As** tüüp]

Parameetermassiivi nimele järgnevad tühjad sulud. Juhul kui parameeter on **Variant**-tüüpi, võib sulud ära jätta. **Dimensiooni** ja **rajasid** siin näidata **ei tohi**.

Järgnevalt on esitatud protseduuride päised
Sub *Mat_Vek* (A(), B(), C(), m, n) ja **Sub *Mat_Vek*** (A, B, C, m, n)

on põhimõtteliselt samaväärsed. Mõlemal juhul on parameetrite tüübiks **Variant**, kuid teisel juhul ei ole näha, et **A**, **B** ja **C** on massiivid. Seda võib näha protseduuris, kust toimub pöördumine antud protseduuri poole. Muide, parameetermassiivi deklaratsioonist ei ole näha ka seda, kas tegemist on fikseeritud või dünaamilise massiiviga. Ka see pannakse paika protseduuris, kust toimub pöördumine.

Parameetermassiivile **vastavaks argumendiks** peab olema **massiiv**. Argumentide loetelus võivad selle nime järel olla tühjad sulud sõltumata tüübist, kuid need ei ole kohustuslikud.

```
Sub Demo_Mas()
  Dim r, v
  r = Range(”ridu”) : v = Range(”veerge”)
  ReDim T(1 To r, 1 To v), V1(1 To v), V2(1 To r)
  ...
  Mat_Vek T(), V1(), V2(), r, v
 ’ Mat_Vek T, V1, V2, r, v
  ...
End Sub
```

Toodud protseduuri fragmendis on näha andmete deklareerimine ja pöördumine alamprotseduuri poole. Massiivid **T**, **V1** ja **V2** on dünaamilised – need on deklareeritud **ReDim**-lausega. Massiivide rajad on määratud muutujatega **r** ja **v**, mille väärtused loetakse eelnevalt töölehelt. Protseduuris on näidatud võimalikud pöördumised alamprotseduuri **Mat_Vek** poole, mille päis on esitatud ülalpool. Teine variant on esitatud kommentaarina. Pöördumistes kasutatakse

Variant-tüüpi argumentmassiive **T**, **V1** ja **V2**, mis vastavad parameetritele **A**, **B** ja **C**. Argumendid on deklareeritud peaprotseduuris. Tühjade sulgude kasutamine argumentides ei ole kohustuslik.

Protseduuri päises **Sub Mat_Vek1** (A#(), B#(), C#(), m, n) on massiivide (elementide) tüübiks määratud reaalarvud – **Double**. Sellisel juhul peavad massiivide nimedele **tingimata** järgnema **tühjad sulud**. Kuna argumentide tüübid peavad vastama parameetritele, peab protseduuris, kust pöördumine toimub, olema massiivid deklareeritud samade tüüpidega, näiteks nii: **ReDim** T#(1 **To** r, 1 **To** v), V1#(1 **To** v), V2#(1 **To** r).
Pöördumisel võivad sulud olla, kuid need ei ole kohustuslikud:
Mat_Vek T(), V1(), V2(), r, v või Mat_Vek T, V1, V2, r, v

Mitmeprotseduurilistes programmides deklareeritakse massiivid, mida kasutatakse erinevates alam-protseduurides, reeglina peaprotseduuris, sest selle tööpiirkond eksisteerib programmi töö lõpuni. Alamprotseduuris deklareeritud massiivid eemaldatakse peale selle töö lõppu.

Massiivi kasutamisel parameetritena tekib üsna sageli vajadus kasutada selle rajasid – indeksite minimaalseid ja maksimaalseid väärtusi erinevate dimensioonide jaoks. Seda läheb vaja näiteks korduste kirjeldamiseks. Rajade kindlakstegemiseks kasutatakse kahte viisi:

- alamprotseduur teeb ise kindlaks massiivide vajalikud rajad,
- rajad esitatakse antud protseduuris parameetrina, nende tegelikud väärtused edastab kutsuv protseduur argumentide abil,

Massiivide rajade leidmiseks on VBAs vastavad funktsioonid:
LBound(massivi_nimi [, dimensioon]) ja **UBound**(massivi_nimi [, dimensioon]).

Funktsioon **LBound** leiab indeksi minimaalse väärtuse antud dimensiooni jaoks, **UBound –** maksimaalse. Dimensiooni number esitatakse teise argumendi abil. Kui argument puudub, loetakse selle väärtuseks 1.

Vaatleme võrdluseks kahte funktsiooni vektoris (ühemõõtmelises massiivis) minimaalse väärtuse leidmiseks. Esimeses funktsioonis on üks parameeter – vektor **V**. Kasutades ülalpool vaadeldud funktsioone, leitakse massiivi rajad **n1** ja **n2**, mida kasutatakse **For**-lauses juhtmuutuja **i** alg- ja lõppväärtuse määramiseks.

```
Function MinV_1(V())
  Dim i, n1, n2, mini
  n1 = LBound(V) :   n2 = UBound(V)
  mini = V(n1)
  For i = n1 + 1 To n2
    If V(i) < mini Then mini = V(i)
  Next i
  MinV_1 = mini
End Function
```

```
Function MinV_2(V(), n1, n2)
  Dim i, mini
  mini = V(n1)
  For i = n1 + 1 To n2
    If V(i) < mini Then mini = V(i)
  Next i
  MinV_2 = mini
End Function
```

```
Function MinV_3(V(), n)
  Dim i, mini
  mini = V(1)
  For i = 2 To n
    If V(i) < mini Then mini = V(i)
  Next i
  MinV_3 = mini
End Function
```

Teises funktsioonis on parameetrites lisaks vektorile näidatud ka indeksi minimaalne ja maksimaalne väärtus. Funktsioon **MinV_2** on laiemate võimalustega kui **MinV_1**, sest seda saab kasutada vektori elementide suvalises vahemikus [**n1**; **n2**]. Praktikas on indeksi minimaalne väärtus enamasti 1. Funktsioon **MinV_3** arvestab sellega. Siin on parameetriteks vektor **V** ja indeksi maksimaalne väärtus **n** (tavaliselt ka elementide arv vektoris). Indeksi minimaalne väärtus (1) on antud funktsioonis fikseeritud.

Massiivide lugemine töölehelt ja kirjutamine töölehele

Töötades Exceli keskkonnas sageli tekib vajadus lugeda massiivide elemente töölehelt ja kirjutada neid sinna. Allpool on toodud tüüpprotseduurid massiivide lugemiseks ja kirjutamiseks. Parameetrile **prk** võib vastata kas piirkond või selle esimene lahter. Eeldatakse, et indeksite minimaale väärtus on **1**.

Lugemine töölehelt

```
Sub Loe_Tab(A, m, n, prk)
  Dim i, j
  For i = 1 To m
    For j = 1 To n
      A(i, j) = prk(i, j)
    Next j
  Next i
End Sub
```

```
Sub Loe_Tulp(V, n, prk)
  Dim i
  For i = 1 To n
    V(i) = prk(i, 1)
  Next i
End Sub
```

```
Sub Loe_Rivi(V, n, prk)
  Dim i
  For i = 1 To n
    V(i) = prk(1, i)
  Next i
End Sub
```

Kirjutamine töölehele

```
Sub Kir_Tab(A, m, n, prk)
  Dim i, j
  For i = 1 To m
    For j = 1 To n
      prk(i, j) = A(i, j)
    Next j
  Next i
End Sub
```

```
Sub Kir_Tulp (V, n, prk)
  Dim i
  For i = 1 To n
    prk(i, 1) = V(i)
  Next i
End Sub
```

```
Sub Kir_Rivi(V, n, prk)
  Dim i
  For i = 1 To n
    V(i) = prk(1, i)
  Next i
End Sub
```

Viitamiseks töölehe lahtritele kasutatakse protseduurides konstruktsiooni **prk.Cells(i, j)** asemel kompaktsemat varianti **prk(i, j)**.

Ühemõõtmeliste massiivide lugemisel/kirjutamisel eristatakse nende paigutust töölehel, kusjuures rivi on horisontaalne, tulp vertikaalne. Paneme tähele indeksite kasutamist töölehe piirkonna jaoks. Formaalselt käsitletakse piirkonda kahemõõtmelisena. Juhul kui parameetrile **prk** seatakse vastavusse piirkond (mitte selle esimene lahter), võib kasutada nii tulba kui ka rivi jaoks samu protseduure, kus viitamiseks töölehe piirkonnale kasutatakse ühte indeksit.

```
Sub Loe_Vek(V, n, prk)
  Dim i
  For i = 1 To n
    V(i) = prk(i)
  Next i
End Sub
```

```
Sub Kir_Vek(V, n, prk)
  Dim i
  For i = 1 To n
    V(i) = prk(i)
  Next i
End Sub
```

Tutvumine Pythoniga

Python on lihtne kuid võimas programmeerimiskeel, mis leiab üha laiemat kasutamist väga erineva iseloomuga rakenduste loomiseks. Tegemist on vabavaralise tarkvaraga.

Pythonit kasutatakse sageli ka programmeerimise õpetamiseks koolides ja ülikoolides.

Sissejuhatus Pythonisse

Python on üldotstarbeline, objektorienteeritud, väike, võimas, lihtne ja lõbus – nagu väidavad keele loojad ja arendajad – vabavaraline programmeerimiskeel, mis on loodud 1991. aastal. Autor on Guido van Rossum Hollandist. Nimi on võetud inglise koomikute grupi Monty Python nime järgi.

Python leiab laialdast kasutamist erinevat liiki tarkvara loomisel, muuhulgas ka veebirakenduste juures ning dokumendipõhistes rakendustes. Kasutamise ulatuselt on ta võrreldav PHP ja Visual Basicuga. Neist kõrgemal on vaid sellised keeled nagu Java ja C-pere keeled (C, C++, C#), mis on eeskätt süsteem-programmeerimise keeled.

Üheks oluliseks Pythoni omaduseks on lihtsus. Selles osas on ta võrreldav Scratchi, Visual Basicu ja VBAga. Keelel on väga lihtne süntaks, milles puuduvad igasugused spetsiifilised eraldajad lausete struktuuri määramiseks võtmesõnade või looksulgudega, nagu näiteks Pascalis, C-s, Javas või mitmetes teistes keeltes. Lausete struktuur on selge ja kompaktne, selles ei ole väärtuste ja muutujate deklareerimist ega struktuurandmete jäiga ja fikseeritud struktuuri kirjeldamist, mis on iseloomulik enamikule programmeerimiskeeltele. Andmetüüpide ja andmestruktuuride käsitlemine on lihtne ja dünaamiline. See on põhjuseks ka Pythoni üha ulatuslikumas kasutamises programmeerimise algkursuste esimese keelena.

Lausete ja võtmesõnade hulk keeles on üsna väike. Suur osa tegevusi määratakse funktsioonide ja objektide meetodite abil. Süsteemi tuuma on sisse ehitatud rida **standardmooduleid**, mis sisaldavad suurt hulka väga erineva otstarbega funktsioone ja protseduure, klasside määranguid ning meetodeid. Väga lihtsalt saab kasutada ka lisamooduleid, mis on koostatud erinevate autorite ja firmade poolt. Kasutaja saab ka ise koostada oma funktsioone ja klasse ning moodustada neist oma mooduleid.

Pythoni rakendustes saab kasutada erinevat liiki andmeid: märkandmed (arvud, tekstid, ajaväärtused ja tõeväärtused), graafikaandmed (pildid, skeemid, joonised), heliandmeid ja multimeedia vahendeid, lugeda andmeid andmebaasidest ja kirjutada neid andmebaasi. Lihtsalt saab kasutada erineva organisatsiooniga andmeid: loendid, massiivid, maatriksid, sõnastikud, failid jms. Võimalik on luua ja töödelda erineva struktuuriga dokumente.

Töö programmide sisestamise, redigeerimise ja silumisega toimub lihtsas ja hõlpsasti käsitlevaks kasutaja-liideses. Programme saab koostada ja redigeerida ka suvalise tekstiredaktoriga.

On mitmeid Pythoni realisatsioone. Peamine ja enamkasutatav on CPython, mis on realiseeritud programmeerimiskeeles C ja töötab enamikes operatsioonisüsteemides: Windows, Linux, Unix, Mac Os, ... Keele arendamisega tegeleb ühing Python Software Foundation (PSF). Veebilehelt http://python.org/ saab Pythoni alla laadida ning seal on ka suurel hulgal viiteid abi- ja õppematerjalidele. CPythoni jaoks arendatakse kahte haru: **Python 2** (momendil viimane versioon 2.7) ja **Python 3** (viimane versioon 3.3), mis on küll teatud määral erinevad, kuid erinevused ei ole eriti suured.

Antud materjal on orienteeritud keelele Python 3. See on osa kursusest „Rakenduste loomise ja programmeerimise alused“ ja ei ole mõeldud päris algajatele. Eeldatakse, et õppur oskab juba teatud määral programmeerida, näiteks Scratchi või BYOB (SNAP) abil. Pythoni kursust võib vaadelda kui Scratchi kursuse järge. Üldiselt ei ole Scratchi tundmine siiski tingimata vajalik, kuid see on kasulik. Arvestame, et lugeja tunneb selliseid programmeerimise põhimõisteid nagu muutuja, avaldis ja omistamine, objekt ning selle omadused ja meetodid, protsesside juhtimine: jada, kordus ja valik. Pythoni olemust ja põhielemente püüame selgitada näidete varal. Soovitame lugejal neid Pythonis proovida, katsetada, uurida ja täiendada. Materjalis ei käsitleta kaugeltki kõiki Pythoni võimalusi, tegemist on sissejuhatusega sellesse imelisse maailma.

Pythoni kohta on üsna palju õpikuid paberkandjal ja loomulikult veebimaterjale. Arvestades keele kiiret arengut ja internetis üha laiemalt levivaid interaktiivse õppimise tehnoloogiaid, olgu järgnevalt nimetatud mõned õppevahendid.

Põhjalik ja mahukas, kuid sujuvalt algav ja pidevalt uuendatav, õppematerjalide komplekt. Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers. How to Think Like a Computer Scientist. Materjalid on HTML- ja PDF-vormingus, lisaks näidete komplektid.

Eelnevaga on otseselt seotud interaktiivne kursus: Brad Miller and David Ranum, How to Think Like a Computer Scientist: Interactive Edition (Using Python 3.x). Tekst põhiliselt eelmisest materjalist. Ineraktiivne Pythoni interpretaator harjutuste koostamiseks ja täitmiseks otse veebidokumendis, ekraanivisioonid, enesetestid jm.

Standfordi Ülikooliga seotud avatud veebikursuste süsteem Udacity. David Evans. Intro to Computer Science (CS101). Building a Search Engine. Pythonil põhinev intensiivne interaktiivne kursus. Ekraanivisioonid, Interaktiivne Pythoni interpretaator, enesetestid jms.

Khan Academy. Computer Science. Introduction to programming and computer science. Ekraanivisioonidel põhinev Pythoni lühikursus.

Kiirtutvus Pythoniga

See jaotis on programmi struktuuri ja komponentide tutvustamiseks.

Programmi struktuur ja komponendid

Näide: Tutvus

Programm **Tutvus** küsib kasutajalt nime ning seda, kas ta tahab lasta leida oma keha indeksi: mass/pikkus^2, mass (kaal) – kg, pikkus – m. Kui kasutaja sisestab vastuseks numbri "**0**", väljastab programm kahetsuse ja lõpetab töö. Kui kasutaja sisestab väärtuse "**1**", küsib programm kasutaja pikkuse ja massi ning leiab ja kuvab kehamassi indeksi ja hinnangu selle kohta. Paremal on toodud algoritmid Scratchis.

```
def saledus(L, m): # funktsioon
  """ Annab massiindeksi alusel hinnangu
   toitumusaste kohta. Parameetrid
   L - pikkus, m - mass """
  indeks = m / (L / 100) ** 2
  if indeks < 18:
     hinnang = "kõhn"
  else:
     if indeks <= 25:
        hinnang =  "normis"
     else:
        if indeks <= 30:
           hinnang =  "ülekaaluline"
        else:
           hinnang =  "suur ülekaal"
  return hinnang
# Tutvus. Peaprotseduur
print("Tere! Olen Python!")  # teate kuvamine
nimi = input('Sinu nimi => ')  # nime lugemine
v = input("Leian kehamassindeksi (1-jah / 0-ei )?  ")
if  v == "0" :  # valiku algus
   print ("Kahju! See on jube kasulik info!")
   print (nimi + "! Ehk mõtled veel? See on tasuta!")
else:          # alternatiiv
   pikkus = int(input(nimi + "! Sinu pikkus (cm) => "))
   mass = float(input("aga mass/kaal (kg) => "))
   indeks = round(mass / (0.01 * pikkus) ** 2, 1)
   print (35 * "=")  # prindib "joone" 35-st märgist "="
   print (nimi + "! Sinu massiindeks on:" + str(indeks) )
   print (nimi,  "! Oled ", saledus(pikkus, mass) )
print ('Kohtumiseni! Igavesti Sinu, Python!')
```

Soovitame lugejal kopeerida toodud programm Pythoni kasutajaliidesese ja katsetada seda ise. Kindlasti tuleks üle vaadata, kas liitlausetes (plokkides) on kõigil käskudel ühesugune taane.

Allpool on toodud mõned selgitused.

Programm ehk moodul koosneb üldjuhul mitmest üksusest: skriptist, protseduurist, funktsioonist. Näites koosneb programm kahest protseduurist ehk skriptist: **peaprotseduur** ja **alamprotseduur** ehk **funktsioon**. Protseduurid koosnevad **lausetest** (käskudest) ja **kommentaaridest**.

Kommentaarid algavad sümboliga **#**, võivad asuda eraldi real või rea lõpus ja ei avalda mingit mõju programmi täitmisele. Kommentaariks võib pidada ka kolmekordsete jutumärkide (*"""*) või ülakomade (*'''*) vahel asuvaid stringe (tekste), nagu praegu on funktsiooni **saledus** alguses.

Laused on korraldused/käsud, mis määravad vajalikke tegevusi. Lausetel on keele reeglitega määratud kindel otstarve ja struktuur. Need võivad sisaldada

- **võtmesõnu**: kindla asukoha, kuju ja tähendusega sõnad (if, else, return, def),
- **funktsiooniviitasid**: saledus(...), round(...), print(...), input(...),
- **string**- ja **arvkonstante**: *"hinnang"*, *"Sinu nimi => "*, 100, 0.01 jms,
- funktsioonide, muutujate ja parameetrite **nimesid**: saledus, round, v, pikkus, L, m,
- **avaldisi**: m / (L / 100) ** 2, mass / (0.01 * pikkus) ** 2, 35 * "=", indeks < 18,
- **operaatoreid** (tehtemärke): +, *, =, ==, <=,
- **piirajaid** ja **eraldajaid**: ( ) , ” : .

NB! Pythonis eristatakse suur- ja väiketähti.

Programmi (mooduli) täitmine algab peaprotseduurist. Funktsioon alustab tööd (käivitub), kui toimub pöördumine selle poole. Peaprotseduur asub lõpus, selle algust ja lõppu kuidagi ei tähistata.

Funktsioon algab alati päisega:

def *nimi* (parameetrid):

millele järgnevad järgmistel ridadel taandega protseduuri keha laused.

NB! päise lõpus peab tingimata olema koolon (:). Parameetritega esitatakse funktsiooni sisendandmed.

Praegu on funktsiooni päis järgmine:

def saledus(*L*, *m*)**:**

Funktsiooni nimeks on **saledus**, parameetriteks: L – pikkus (cm) ja m - mass (kg).

Väga sageli leiab ja tagastab funktsioon ühe või mitu väärtust. Väärtuste tagastamiseks kasutatakse **return**-lauset, mis lõpetab funktsiooni töö ning tagastab tulemuse(d) ja täitmisjärje kohta, kust toimus pöördumine funktsiooniviida abil. Funktsiooniviit esitatakse kujul: ***nimi*** (*argumendid*), kus ***nimi*** on funktsiooni nimi ning ***argumendid*** annavad parameetritele väärtused. Funktsioon **saledus** tagastab muutuja **hinnang** väärtuse, mille leiab (valib) **if**-lause muutuja **indeks** väärtuse alusel. Viimane omakorda sõltub parameetrite **L** ja **m** väärtustest. Pöördumine funktsiooni poole toimub peaprotseduuri lausest:

print (nimi, *"Oled "*, **saledus**(pikkus, mass))

Parameetritele **L** ja **m** vastavad argumendid on esitatud muutujate pikkus ja mass abil.

Funktsioonid **print**(), **input**() ja **round**() on Pythoni sisefunktsioonid.

Üheks Pythoni süntaksi oluliseks eripäraks on taanete kohustuslik ja reeglipärane kasutamine liitlausetes. Viimased sisaldavad teisi liht ja/või liitlauseid. Näites on kasutusel liitlaused **if** ja **else** nii peaprotseduuris kui ka funktsioonis **saledus**, mis on ka ise liitlause (**def**-lause).

Liitlause sisemised laused peavad algama taandega lause päise suhtes, milleks on soovituslikult 3–4 positsiooni, et struktuur oleks selgesti hoomatav. Ühesugusega tasemega laused peavad algama samast positsioonist.

Allpool on toodud näiteks peaprotseduuri **if**-lause.

NB! **else** ei ole omaette lause vaid **if**-lause osa, mis ei ole kohustuslik. Omaette **else** esineda ei saa, talle peab alati vastama üks ja ainult üks **if**-lause. Sellepärast nimetatakse seda sageli ka **else**-osalauseks.

```
if int(vastus) == 0 :  # valiku algus
    print ("Kahju! See on jube kasulik info!")
    print (nimi + "! Ehk mõtled veel? See on tasuta!")
else :
    pikkus = int(input(nimi+"! Sinu pikkus (cm) => "))
    mass = float(input("aga mass/kaal (kg) => "))
    …
    print ("Oled " + saledus(pikkus, mass) + "!")
print ('Kohtumiseni! Igavesti Sinu, Python!')
```

if lause

else lause

if-lausesse kuulub kaks lihtlauset, mis paiknevad päise suhtes taandega. Sellega seotud **else**-osalause peab algama täpselt samast positsioonist.

else-lausesse kuulub kuus lihtlauset, millel kõigil on täpselt ühesugune taane.

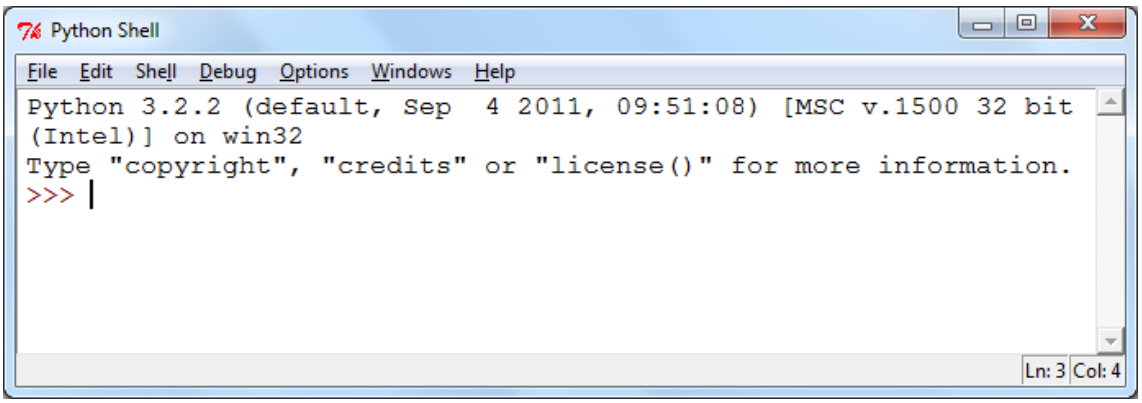
Viimasel print-lausel ei ole taanet, sest see ei kuulu **if**-lausesse.

Funktsioonis **saledus** on mitu erineva tasemega lauset. Funktsioon **def** ise kujutab liitlauset, mille sees on kaks liitlauset: **if**- ja **else**-lause. Esimeses on üks lihtlause, teises kaks liitlauset, mis on samuti **if**- ja **else**-laused. Viimases on omakorda **if**- ja **else**-lause. Taanded ja ainult taanded määravad liitlausete sisalduvuse tasemeid. Täpsemalt on liitlausete struktuurist juttu jaotises **Liitlaused**.

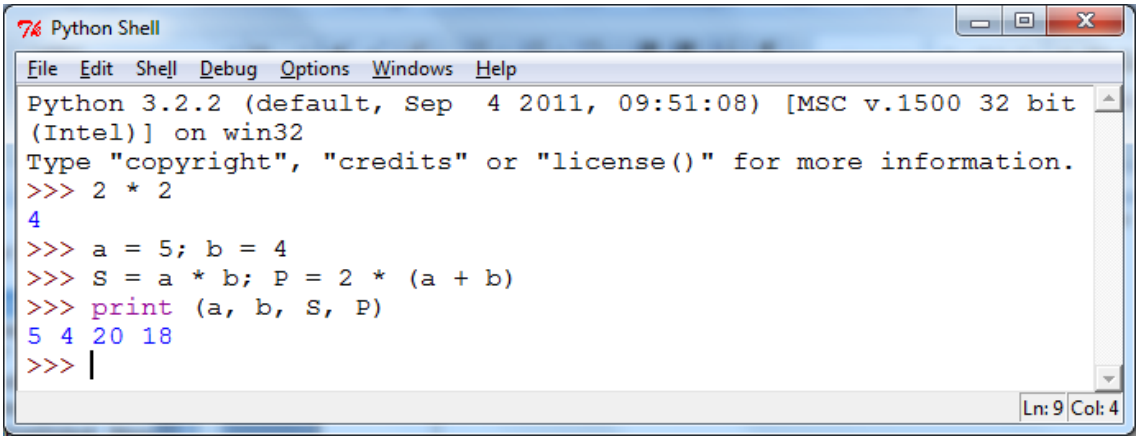
Programmide sisestamine ja täitmine

Programmide sisestamiseks, redigeerimiseks ja täitmiseks kasutatakse tavaliselt interaktiivset kasutaja-liidest **IDLE** (***I**nteractive **D**eve**L**opment **E**nvironment*), mille haldur kuulub Pythoni põhikomplekti, kuid on ka mitmeid teisi Pythoni kasutajaliideseid nagu näiteks PyScripter, mille peab eraldi alla laadima. IDLE käivitamiseks võib kasutada Windowsi Start-menüüst järgmist korraldust:
Start > Programs > Python 3.x > IDLE (Python GUI).

Kuvatakse interaktiivse kasutajaliidese aken **Python Shell**.



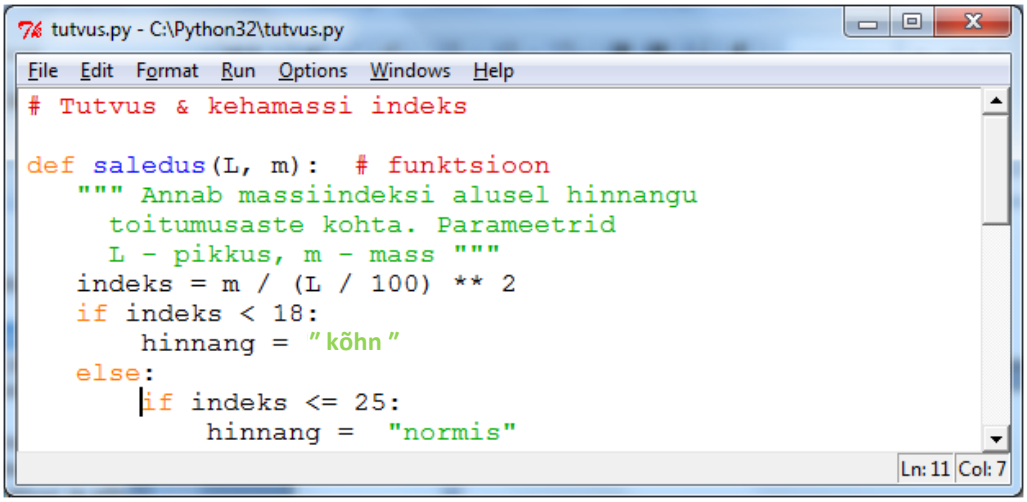
Shelli aknasse (käsuaknasse) väljastatakse tulemused ja teated. Akent võib kasutada interaktiivseteks arvutusteks nn kalkulaatori režiimis. Käsureale, mille alguses on sümbolid **>>>**, saab sisestada väärtusi, avaldisi, lauseid ja programmi fragmente.



Selles programmis on kõigepealt leitud avaldise 2*2 väärtus, mis, nagu selgub, on ka Pythoni jaoks ikka veel 4. Edasi omistatakse muutujale **a** väärtus 5 ja muutujale **b** väärtus 4. Tegemist võiks olla siin näiteks ristküliku külgede pikkustega. Järgnevate omistamislausetega leitakse ristküliku pindala (**S**) ja ümbermõõt (**P**). Lausega **print** väljastatakse ekraanile **a**, **b**, **S** ja **P** väärtused.

Selline töörežiim võimaldab mitmesuguste operatiivsete arvutuste tegemist näiteks programmide silumiseks, aga ka keelevahenditega tutvumiseks ja nende uurimiseks, kuid seda kasutatakse ka päris algajatele mõningate põhimõistete (andmete tüübid, avaldised, muutujad, …) selgitamiseks. Meie sellel pikemalt ei peatu, soovitame õppuril endal katsetada.

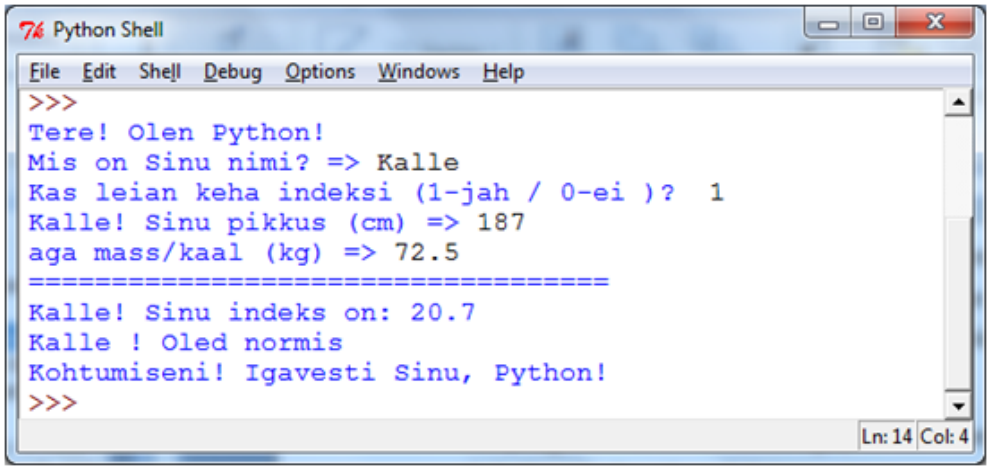
Programmi sisestamiseks tuleb File-menüüst anda käsk New Window. Ilmub **redaktori aken**, kuhu saab sisestada või kopeerida programmi. Tegemist on spetsiaalse tekstiredaktoriga, mis toetab Pythoni programmide sisestamist – taanete tegemist, programmi elementide ilmestamist, abiinfo ja spikrite pakkumist jms. Vaadeldava programmi algus on allpool toodud redaktori aknas. Redaktor kuvab erinevad programmi elemendid – kommentaarid, võtmesõnad, stringid jms – erinevate värvidega, kui programmifail on salvestatud laiendiga **.py**.



Programmi ehk mooduli saab käivitada redaktorist käsuga **Run** > **Run Module** või klahviga **F5**. Programmi täitmist korraldab spetsiaalne süsteemiprogramm – Pythoni interpretaator. Programm (fail) peab olema enne täitmist salvestatud.

Kui programmis on süntaksivigu, kuvab interpretaator veateate ja näitab ka vea eeldatava koha. Kui vigu ei ole, alustab interpretaator programmi täitmist, muutes aktiivseks käsuakna (Shelli akna), kus kuvatakse programmi teated ja tulemused. Vead võivad tekkida ka täitmisel, kui näiteks sisestatakse ebasobivad andmed vms.

Vaadeldava programmi jaoks on allpool näidatud Shelli aken variandi jaoks, kui kasutaja nimega Kalle soovis lasta arvutada kehamassiindeksi.

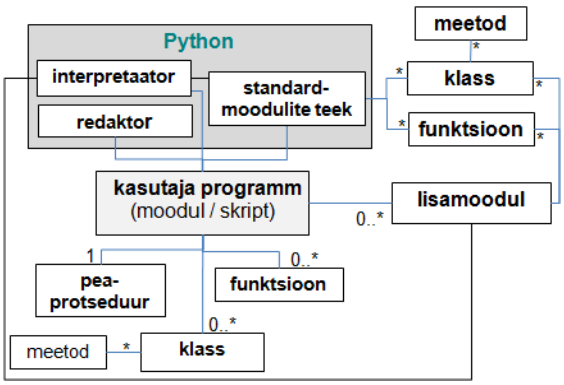


Programmide töötlemisel on kasulik võimalus pöörduda Help-menüüst käsuga Python Docs või klahviga **F1** Pythoni dokumentatsioonileheküljele (Python v3.x.x. documentation), mis vastab kasutatavale versioonile ning sisaldab hulgaliselt informatsiooni Pythoni kohta.

Moodulid, funktsioonid, klassid ja meetodid

Pythoni programm (öeldakse ka moodul või skript) võib üldjuhul koosneda mitmest üksusest. Alati on olemas **peaprotseduur**, milles võib olla suvaline hulk **funktsioone** ehk **protseduure** ja kasutaja poolt määratletud **klasse**. Funktsiooni näide (saledus) oli programmis **Tutvus**. Funktsioonide loomise põhimõtteid vaadeldakse jaotises **Funktsioonid** ning kasutatakse läbi kogu käesoleva materjali. **Klasside** loomist ei käsitleta.

Reeglina kasutatakse programmis lisaks oma protseduuridele ja klassidele ka vahendeid Pythoni moodulitest. Tavaliselt on tegemist **standardmoodulite teeki** (*The Python Standard Library*) kuuluvate moodulitega, mis laetakse alla koos süsteemiga. Lisaks standardmoodulitele saab kasutada lisamooduleid, mis ei kuulu nimetatud teeki. On suur hulk mooduleid, mis on loodud erinevate autorite ja firmade poolt ning reeglina tasuta allalaaditavad Internetist. Kasutaja saab lihtsalt moodustada ka oma mooduleid ja kasutada neid analoogselt olemasolevate moodulitega.



Mooduli nagu ka kasutaja loodud programmi komponentideks võivad olla funktsioonide ja klasside definitsioonid ehk lühemalt öelduna **funktsioonid** ja **klassid**. Lihtsamal juhul koosneb moodul ainult funktsioonidest. **Klass** määratleb ühetüübiliste objektide **omadused** ja **tegevused**. Võimalikud tegevused määratakse nn **meetodite** abil, mis oma olemuselt kujutavad funktsioone ja erinevad tavafunktsioonidest selle poolest, et neid saab kasutada ainult antud klassi kuuluvate objektidega.

Pythoni rakenduste loomine on praktiliselt alati seotud standardmoodulite teegis asuvate vahendite kasutamisega. Osa selles olevatest funktsioonidest ja klassidest kuuluvad **sisseehitatud** (*built-in*) funktsioonide ja klasside hulka. **Sisefunktsioonide** kasutamiseks ei ole vaja mingeid spetsiaalseid meetmeid. Soovitud kohta, näiteks avaldisse, kirjutatakse funktsiooniviit: funktsiooni nimi, millele

tüüpiliselt järgnevad sulgudes argumendid. Sellised on näites **Tutvus** esinevad funktsioonid: **int**, **float**, **str**, **input**, **print** ja **round**. Järgneb veidi modifitseeritud lause programmist **Tutvus**:

print (nimi + *"!"*, *"Sinu massiindeks on:"* + **str (round (**indeks, 2)))

Funktsiooni **print** argumentideks on siin kaks stringavaldist.

NB! Python 3-s on **print** funktsioon, Python 2-s on see lause.

Näite teises argumendis on kaks sisefunktsiooni: **round**(r_arv) – ümardab reaalarvu etteantud täpsuseni ja **str**(arv) – teisendab arvu tekstivormingusse.

Põhjalikku informatsiooni standardteegi moodulite ja neisse kuuluvatest vahenditest, sh ka sise-funktsioonidest, saab Pythoni dokumentatsiooni (IDLE Help, Python Docs) materjalist Library Reference. Suur osa standardmoodulites olevaid funktsioone ning klasse ja nende meetodeid ei kuulu aga sisseehitatud vahendite hulka. Nendeks on ka üldlevinud matemaatikafunktsioonid sqrt(), sin(), cos(), log() jt, mille kasutamiseks peab **importima** oma projekti vastavad moodulid või nende komponendid.

import *moodul1*, *moodul2*, …

moodul esitatakse mooduli nime abil. Näiteks: **import** math, turtle, time, random.

Käsu alusel imporditakse projekti moodulid. Formaalselt on imporditud moodulid objektid, mille nimedeks on praegu moodulite nimed, seal kirjeldatud konstandid ja funktsioonid on objekti atribuudid ja meetodid.

Viitamiseks objekti (mooduli) meetoditele kasutatakse konstruktsiooni:

objekt.***meetod***(*argumendid*),

kus ***objekt*** esitatakse nime abil, ***meetod*** on meetodi (funktsiooni) nimi, ***argumendid*** esindavad meetodi täitmisel kasutatavaid andmeid. Argumentide tähendus, arv ja järjestus sõltuvad meetodist (funktsioonist).

Järgnevas on toodud mõned viitamise näited:

math.sqrt(x ** 2 + y ** 2), **math.**sin(**math.**pi * x), **turtle.**pendown(), **turtle.**setpos(-50, 100), **turtle**.left(45), **random.**randint(1, 100), algaeg = **time**.clock(), **time**.sleep(0.1)

Importimiseks võib kasutada ka:

from *moodul* **import** *

Sel juhul tehakse kättesaadavaks *mooduli* kõikide funktsioonide, meetodite ja klasside nimed ja neile viitamisel mooduli (objekti) nime ei kasutata.

from math **import** * ; **from** turtle **import** *

Kasutamise näited:

```
y = sqrt(log10(5 * cos(x + 3));

pendown();

pencolor('red');

forward(120)
```

Näeme, et kasutamine toimub samamoodi nagu sisefunktsioonide puhul. Kuigi teine variant on kompaktsem, kasutatakse rohkem esimest. Siin on ühelt poolt tegemist traditsioonidega ja „hea tooni“ reeglitega. Teiselt poolt võib aga suurtes programmides, kus on kasutusel palju funktsioone paljudest moodulitest, teise variandi korral tekkida nimede kokkulangemise tõttu segadusi. Uurimaks standardmoodulite olemust ja nende kasutamist, võiks teha mõned harjutused Shelli aknas.

```
>>> sqrt(169)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in
<module>
    sqrt(169)
NameError: name 'sqrt' is not defined
>>>
```

Kõigepealt ruutjuur 169-st. Python kuvab veateate väites, et nimi '**sqrt**' ei ole defineeritud. Sama teade ilmub ka siis, kui taolist asja kasutatakse programmis ilma **import** käsuta; programmi töö katkestatakse.

```
>>> import math
>>> math.sqrt(169)
13.0
>>> a = 2; b = 3; c = -9
>>> math.sqrt(b ** 2 - 4 * a * c)
9.0
```

Imporditakse moodul **math**, mis sisaldab üldlevinud matemaatikafunktsioone nagu sqrt(), sin(), cos(), log() jmt. Nimi **math** (koos punktiga) peab eelnema igale funktsiooniviidale.

```
>>> c = 9
>>> math.sqrt(b ** 2 - 4 * a * c)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in
<module>
    math.sqrt(b ** 2 - 4 * a * c)
ValueError: math domain error
```

Näide süsteemi reaktsioonist andmetele, mille puhul mingit operatsiooni teha ei saa. Praegu on tegemist katsega leida ruutjuur negatiivsest arvust.

```
>>> 3 * math.sin(math.pi *
b/7)/math.pi ** 2
0.29634255008464655
```

Ka konstant π on olemas moodulis **math**: math.pi

```
>>> from math import *
>>> 3 * sin(pi * b / 7) / pi ** 2
0.29634255008464655
```

Kui importimiseks kasutatakse käsku **from**, ei ole funktsiooniviitades mooduli nime.

Näide: Korrutustabel

Programm võimaldab harjutada korrutamist. Kõrval on toodud algoritm.

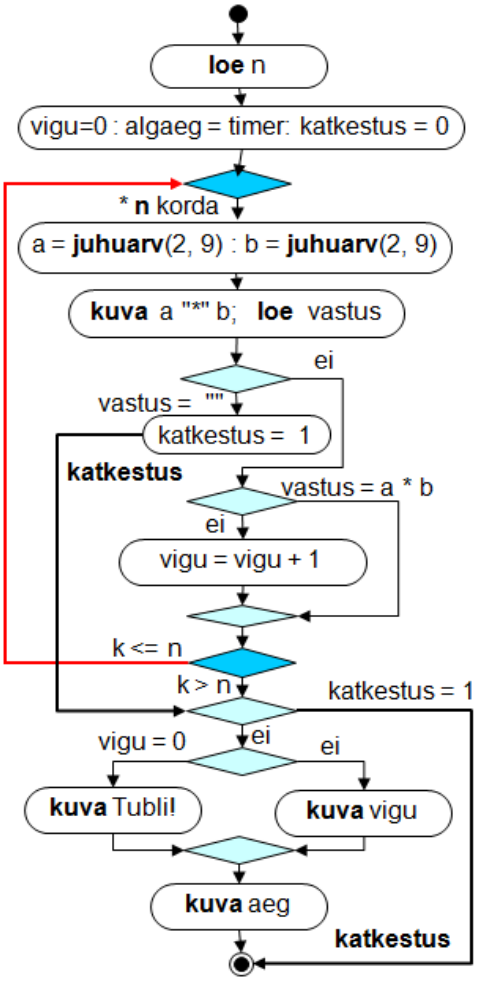
Näide illustreerib skalaarandmete ja avaldiste ning valikute kasutamist, tutvustab moodulite importimist ja korduste kasutamise põhimõtteid jms.

Esimesena sisestatakse ülesannete arv **n**. Programmi põhiosa moodustab kordus, mis kuvab järjest **n** ülesannet, loeb kasutaja vastused ja hindab neid.

Lõpus kuvatakse üldhinnang ja kulunud aeg. Kasutaja võib katkestada töö tühja väärtuse sisestamisega.

Tegevuste sisuline pool, mis eriti ei sõltu kasutavast keelest, on järgneval toodud UML-tegevusskeemil.

UML-tegevusskeem




```
import random, time  # moodulite import
def korrutustabel():
  ''' Korrutamistabeli harjutamine.
     Küsib ülesannete arvu ja esitab
     juhuarvude (2...9) abil ülesandeid.
     Fikseerib vigade arvu ja aja. '''
  n = int(input("Mitu ülesannet? "))
  vigu = 0; katkestus = False
  algaeg = time.clock()  # jooksev arvutiaeg
  for k in range(n) :  # korduse algus
     a = random.randint(2, 9)
     b = random.randint(2, 9)
     abi = str(k+1) + '. ' + str(a) + ' * ' + str(b)
     vastus  = input(abi + "=> " )
     if vastus == "":  # kui tühi väärtus
        katkestus = True
        break  # katkestab korduse
     if int(vastus) != a * b :
        print ("Vale!")
        vigu += 1       # vigu = vigu + 1
        time.sleep(2)  # karistuseks paus 2 sek
        # korratava ploki lõpp
  if katkestus == True:
     print ("Kahju, et loobusid!")
     return      # lõpetab programmi töö
  if vigu == 0 :
    print("Tubli! Kõik oli õige!")
  else :
     print ("Vigu oli ", vigu)
  aeg =  round(time.clock() - algaeg, 2)
  print ("Aeg: " + str(aeg) + " sek")

korrutustabel()
```

Kasutusel on 9 muutujat: ***n*** – ülesannete arv, ***a*** ja ***b*** – tegurid (juhuarvud vahemikus 2...9), ***vastus*** – kasutaja poolt pakutav vastus, ***vigu*** – valede vastuste arv, ***abi*** – abimuutuja teate kuvamiseks (algoritmis ei ole kajastatud), ***algaeg*** – alguse aeg, ***aeg*** – kulunud aeg, ***katkestus*** – katkestamise tunnus: *False* – ei katkestatud (skeemil 0), *True* – katkestati, sisestati tühi väärtus (skeemil 1).

Praktiliselt kogu programm kujutab endast ühte protseduuri, millel ei ole parameetreid ja mis ei tagasta ka mingeid väärtusi. Väljaspool seda on ainult käsk **import** ja pöördumislause **korrutustabel**().

Käsuga (lausega) **import** imporditakse Pythoni standardteegist moodulid **random** ja **time**. Mooduli **random** meetodit **randint**(min, max) kasutatakse juhuslike täisarvude genereerimiseks etteantavas vahemikus. Mooduli **time** meetod **clock** annab aja, mis on möödunud programmi töö algusest, meetod **sleep**(pp) võimaldab tekitada etteantud pikkusega (sek) pausi.

Praegu kasutatava importimise variandi korral peab viitamiseks vajalikule funktsioonile (tekkinud objekti meetodile) kasutama konstruktsiooni: ***objekt.meetod***(*argumendid*).

Programmi keskseks osaks on **kordus**, mille kirjeldamiseks kasutatakse Pythoni **for**-lause ühte levinumat varianti etteantud kordamise arvuga korduste määratlemiseks (täpsemalt vt jaotist „For-lause“, lk 211), mille üldkuju on järgmine:

for *muutuja* **in range (n):**

laused

n on alati täisarvuline muutuja, mida nimetatakse sageli ka **juhtmuutujaks**. Funktsioon **range**(n) moodustab täisarvude jada 0 kuni n-1. Seega omandab **for**-lauses juhtmuutuja järjest väärtused alates 0-st kuni n-1ni sammuga 1, kusjuures muutuja iga väärtuse korral täidetakse ***laused***. Kokku täidetakse kordusesse kuuluvaid lauseid **n** korda.

Programmis on ette nähtud võimalus korduse katkestamiseks kasutaja poolt. Kui lause

vastus = **input**(abi + ″=> ″)

täitmisel ei sisestata väärtust, vaid vajutatakse kohe klahvile Enter, võetakse muutuja ***vastus*** väärtuseks tühi väärtus. Kui järgnev **if**-lause fikseerib taolise olukorra, võetakse muutuja ***katkestus*** väärtuseks **True** ja **break**-lause katkestab korduse, andes täitmisjärje **for**-lausele järgnevale lausele:

```
if katkestus == True:     # ==True võib ära jätta
    print ("Kahju, et loobusid!")
    return      # lõpetab protseduuri töö
```

Paneme tähele loogikamuutuja ***katkestus*** kasutamist ja katkestuse töötlemist. Enne korduse algust võetakse muutuja väärtuseks **False**. Kui toimub katkestus, võetakse muutuja uueks väärtuseks **True**. Siin arvestatakse, et korduse ja kogu programmi töö võib lõppeda kahel erineval viisil – normaalne lõpp, kui täidetakse kõik ***n*** ülesannet (***katkestus*** = **False**), või programmi (korduse) töö katkestatakse (***katkestus*** = **True**). Viimasel juhul ei ole vaja teha ka kokkuvõtet. Sellekohase otsuse teeb ülaltoodud **if**-lause. Kui tingimus ***katkestus*** == **True,** lõpetab **return**-lause antud funktsiooni (protseduuri) ja sellega ka kogu programmi töö.

Andmete sisestamiseks ja väljastamiseks kasutatakse funktsioone **input** ja **print**. Funktsiooni **input** poolt tagastatava väärtuse teisendamiseks tekstivormingust täisarvuvormingusse on funktsiooni **int**.

Väärtuste teisendamisega on tegemist ka lauses abi = **str**(k+1) + ′. ′ + **str**(a) + ′ * ′ + **str**(b)

Funktsiooni **str** abil teisendatakse avaldise **k+1** ning muutujate **a** ja **b** väärtused tekstideks. See on vajalik tekstavaldises, kus arvud ühendatakse stringidega. Muutuja **abi** on kasutusel lihtsalt järgmise lause lühendamiseks.

Andmetest: väärtused ja tüübid, konstandid ja muutujad, omistamine ja avaldised

Märkandmete väärtused ja tüübid. Klassid

Pythonis programmides on kõik **väärtused** (tekstid, arvud, tõeväärtused) **objektid** ja kuuluvad kindlasse **klassi**. Igale klassile vastab kindel nimi. Märkandmete põhitüüpideks ehk -klassideks on:

- **stringid** – klass **str**
- **täisarvud** – klass **int**
- **ujupunktarvud** (reaalarvud) – klass **float**
- **tõeväärtused** – klass **bool**.

Igale väärtusele eraldatakse programmi täitmisel ajal koht (väli ehk pesa) arvuti mälus. Väärtuse esitusviis ja välja pikkus (baitides) sõltub väärtuse tüübist (klassist).

```
>>> type("Tere")
<class 'str'>
>>> type(13)
<class 'int'>
>>> type("13")
<class 'str'>
>>> type(21.53)
<class 'float'>
>>> type('21.73')
<class 'str'>
>>> type(2 * 2)
<class 'int'>
>>> type(2 * 2 == 4)
<class 'bool'>
>>>
```

Tutvumiseks väärtuste klassidega ehk tüüpidega sisestage prooviks Shellis mõned funktsioonid **type**. Mõisted „**tüüp**“ ja „**klass**“ on samaväärsed.

Funktsioon **type**(*väärtus*) tagastab väärtuse tüübi ehk klassi nagu näidatud kõrval. Kui argument on esitatud avaldisega, leitakse selle väärtus ja tagastatakse saadud tulemuse klass (tüüp). Paneme tähele, et **jutumärkide** või **ülakomade** vahele paigutatud arve käsitletakse stringidena.

Erinevat tüüpi väärtuste salvestamiseks arvuti mälus kasutatakse erinevaid **vorminguid**. Stringid (klass **str**) esitatakse **tekstivormingus**: igale märgile (sümbolile) vastab kindel kahendkood. Arvud võivad olla salvestatud nii tekstivormingus (klass **str**) kui ka spetsiaalsetes täisarvude ja realarvude jaoks ettenähtud püsipunkt- ja ujupunkt-vormingutes (klassid **int** ja **float**).

Tekstivormingus arvudega arvuti matemaatikatehteid teha ei saa! Kui seda on vaja, peab väärtuse teisendama funktsiooniga **int** või **float** arvuvormingusse.

Lihtsamal juhul, milleks on ka toodud näide, esitatakse märkandmed programmides üksikväärtustena (skalaaridena) – **konstantide** ja **muutujatena**.

Konstandid

Konstantide väärtused esitatakse otse programmis. Nende esitusviis sõltub väärtuse tüübist (klassist). Väärtuse muutmiseks peab tegema muudatuse programmis.

Stringkonstandid paigutatakse tavaliselt **jutumärkide** või **ülakomade** (apostroofide) – nn piirajate – vahele:

"kõhn", "Tere, olen Python!", 'Mis on Sinu nimi?' , "e", "0", "12", '-63.5'

Piirajad konstandi väärtuse hulka ei kuulu. Jutumärgid ja ülakomad on piirajatena samaväärsed. Ühe konstandi jaoks peavad piirajad olema samad. Ühte tüüpi piirajate vahel oleva stringi sees võivad olla teist tüüpi piirajad, nagu näiteks:

```
nimi = input ( 'romaani "Kevade" autor => ' )
```

Kasutatakse ka kolmekordsete piirajate """ või ''' vahel asuvaid stringe. Sellel variandil on mõningad täiendavaid võimalusi ja eesmärke. Väärtus võib paikneda mitmel real ja täita näiteks ka pikema kommentaari rolli. Eriline tähendus on aga taolisel stringil funktsiooni alguses, mida mitmed Pythoni redaktorid võimaldavad kuvada Help-käsu abil (info funktsiooni kohta). Sellist stringi nimetakse **dokumendistringiks** (*docstring*).

NB! Stringkonstandina esitatud arv (näiteks: "0", "12", '-63.5') salvestatakse tekstivormingus – igale sümbolile vastab kindel kood. Sellises vormingus salvestatud arvudega aritmeetikaoperatsioone täita ei saa. Olgu märgitud, et funktsiooni **input** poolt tagastatav väärtus on alati tekstivormingus. Kui sisestatud arve kasutatakse arvutustes, tuleb need teisendada funktsioonide **int** või **float** abil vastavasse arvuvormingusse. Paneme tähele võrdluse esitust varasema näite peaprotseduuri **if**-lauses:

```
…
v= input("Leian keha indeksi(1-jah /0- ei) ")
if  v ==  "0" :
…
```

Siin arvestatakse, et **input**-lausega sisestatud muutuja ***v*** väärtus on tekstivormingus. Sellepärast on võrdlemisel ka väärtus 0 (null) esitatud tekstivormingus.

Arvkonstandid esitatakse programmides enamasti tavaliste kümnendarvudena või eksponentvormingus. Reaalarvudes kasutatakse murdosa eraldamiseks **punkti:**

13, 74600, -21, 73.0, 73.5902, 2.1e6 = 2.1×10^{6} , 1e-20 =10^{-20}

Loogikakonstant ehk tõeväärtus esitatakse võtmesõnade **True** või **False** abil. Kasutamist võis näha programmis **korrutustabel** lk 163.

Muutujad. Omistamine, omistamislause, avaldised

Pythonis mõistetakse muutuja all **nime**, mis **viitab väärtusele** (objektile). Vaadeldava programmi peaprotseduuris on viis muutujat: **nimi**, **v**, **pikkus**, **mass**, **indeks**, funktsioonis **saledus** on kaks muutujat: **indeks** ja **hinnang**. Funktsioonis on kasutusel ka kaks **parameetrit**: **L** ja **m**. Parameetrid on eri liiki muutujad, mis erinevad nö tavamuutujatest otstarbe ja väärtuse saamise viisi poolest. Need esitatakse programmis samuti nimede abil.

Nimi võib koosneda ühest tähest või tähtede, numbrite ja allkriipsude (_) jadast, mis peab algama tähega või allkriipsuga. Teisi sümboleid (kaasaarvatud tühikud) nimes olla ei tohi. Pythonis eristatakse nimedes suur- ja väiketahti. Need reeglid kehtivad kõikide nimede kohta (funktsioonide, loendite, klasside nimed):

nimi, v, pikkus, L, x_1, Ab_t3_st_8, _algus, täht

Erinevalt paljudest teistest programmeerimiskeeltest (Pascal, C, Java) ei pea Pythonis deklareerima muutujaid ja määratlema kirjeldustes nende tüüpe. Seda isegi ei saa teha. Pythoni muutuja luuakse programmi töö ajal **omistamislause** täitmisel, kui muutuja nimi esineb esimest korda omistamislause vasakus pooles. Sellega luuakse ja salvestatakse ka muutuja esimene (võimalik, et ainuke) väärtus. Omistamislause põhivariant esitatakse järgmiselt :

muutuja = ***avaldis***

Selles lauses on ***muutuja*** muutuja nimi, ***avaldis*** annab eeskirja väärtuse leidmiseks. Avaldise erijuhuks on konstant, muutuja ja funktsiooniviit.

hinnang = *"kõhn"*, *indeks* = *mass* / (0.01 * *pikkus*) ** 2 ; *nimi* = **input**(*'Sinu nimi => '*)

pealik = *"Juku"*; *palk* = 0; *n* = 13; *x* = 2.73; *y* = -5; *dist* = (x**2 + y**2)**0.5

Proovige Shelli aknas muutujate ja avaldiste kasutamist, sisestades näiteks allolevad korraldused.

```
>>> x = 2.73; y = -5
>>> dist = (x**2 + y**2)**0.5
>>> dist
5.696744684466735
>>> s = (x**2 - 3*y)**0.5 - 4*(x + 3) / 2
>>> s
-6.721550886629678
>>>
```

Luuakse kaks muutujat: ***x*** ja ***y*** ning omistatakse neile väärtused: 2.73 ja -5.

Luuakse muutuja ***dist***, arvutatakse avaldise väärtus ja omistatakse see muutujale.

Luuakse muutuja ***s***, arvutatakse avaldise väärtus ja omistatakse see muutujale.

Aritmeetika põhitehted ja nende prioriteedid:

****** astendamine

*****, **/** korrutamine ja jagamine

+ , **-** liitmine ja lahutamine

Toodud näidetes on mitmeid **arvavaldisi**. Nende operandide väärtusteks on arvud ja operatsioonid määratakse aritmeetikatehetega. Programmis on ka neli **stringavaldist**.

Lauses

print (nimi + "! Sinu massiindeks on:" + **str(**indeks))

olevas avaldises toimub kolme stringi ühendamine (liitmine): muutuja ***nimi*** väärtus (näiteks Kalle), konstandi *"! Sinu massiindeks on:"* ja muutuja ***indeks*** väärtus (näiteks 20.7).

Avaldisega

print(35 * "=")

korratakse stringi "=" 35 korda ning kuvatakse 35-st märgist koosnev „joon". Täpsemalt avaldistest vt jaotises „Avaldiste struktuur ja liigid".

Tutvumine loenditega

Loend ehk ühemõõtmeline massiiv kujutab endast järjestatud väärtuste (objektide) kogumit. Loend tähistatakse ühe nimega, viitamiseks elementidele kasutatakse nime koos nurksulgudes asuva indeksiga (järjenumbriga) – ***nimi***[i*ndeks*]: V[0], V[i], V[2*k+1]. Indeks võib olla konstant, muutuja või avaldis. Elementide nummerdamine (indeks) algab alati nullist!

Funktsioon **len**(loendi_nimi) võimaldab leida loendi pikkuse – elementide arvu loendis.

Näide: Meeskond

On antud mingi meeskonna liikmete nimed ja pikkused. Rakendus teeb järgmist:

- kuvab (prindib) kõikide mängijate nimed ja pikkused
- leiab ja kuvab mängijate keskmise pikkuse
- leiab nende liikmete keskmise pikkuse, kellel see on suurem üldisest keskmisest
- leiab ja kuvab kõige pikema mängija pikkuse ja nime

Luuakse kaks loendit: nimed ja P. Esimene koosneb stringidest, teine arvudest.

Esimese **for**-lause abil prinditakse meeskonna kõikide liikmete nimed ja pikkused ning leitakse pikkuste summa. Jaganud leitud summa n-ga, saadakse keskmine pikkus.

```
# Loendite kasutamise demo, loendite loomine
nimed = [ 'Haab', 'Kask', 'Mänd', 'Paju', 'Saar', 'Tamm' ]
P = [ 193, 181, 178, 203, 197, 177]
n = len(nimed) # elementide arv loendis (pikkus)
# loendite kuvamine ja summa leidmine
print ("Meeskonnas on ", n, "liiget")
print (" ", "Nimi", "Pikkus")  # tabeli päis
summa = 0
for i in range(n) :
    print (i + 1, nimed [i], P[i])
    summa = summa + P[i]
kesk  = summa / n  # keskmine pikkus
print ("Keskmine pikkus:", round(kesk, 2))
# Üle keskmiste keskmine
print ("\nÜle keskmise pikkuse")
summa = k = 0  # üks väärtus kahele muutujale
print (" ", "Nimi", "Pikkus")
for i in range(n):
   if P[i] > kesk:
      print (nimed [i], P[i])
      summa = summa + P[i]; k += 1
kesk = summa / k
print ("Üle keskmise keskmine:", round(kesk, 2))
# Kõige pikem
maks = P[0]; nr = 0
for i in range(n):
   if P[i] > maks: maks = P[i]; nr = i
print ('\nPikim on', nimed[nr], '-', maks, 'cm')
```

Programmi esimese osa väljundil on kuju:

```
Meeskonnas on  6 liiget
  Nimi Pikkus
1 Haab 193
2 Kask 181
… …   …
6 Tamm 177
Keskmine pikkus: 188.17
```

Teine **for**-lause prindib keskmisest pikemate meeskonna liikmete nimed ja pikkused.

Leitakse ka nende pikkuste summa ja arv (k), mis on vajalikud vastava keskmise leidmiseks.

```
Üle keskmise pikkuse
Nimi  Pikkus
Haab  193
Paju   203
Saar  197
Üle keskmiste keskmine: 197.67
```

Kolmas **fo**r-lause leiab maksimaalse elemendi loendis P ja selle järjenumbri.

```
Pikim on Paju – 203 cm
```

Programmi lausete struktuur ja põhielemendid

Kokkulepped süntaksi esitamiseks

Keelekonstruktsioonide kirjeldamisel kasutatakse teatud kokkuleppeid, mis võimaldavad näidata lausete ja nende elementide esitusviise kompaktselt ja ühemõtteliselt. Taolisi leppeid kasutatakse laialdaselt formaalsete keelte süntaksi kirjeldamiseks.

Võtmesõnad, **tehtesümbolid**, **piirajad** ja **eraldajad** moodustavad tavaliselt keelekonstruktsiooni püsiva osa, need peavad olema esitatud programmis täpselt nendes kohtades ja sellisel kujul, nagu on näidatud kirjelduses.

Keelekonstruktsioonide muutuvad komponendid võib valida programmi koostaja, arvestades nende esitusreegleid. Kirjeldustes on need toodud kaldkirjas: *nimi, avaldis, lause* jne. Muutuv osa võib olla esitatud üldkujul, hiljem täpsustatakse seda täiendavate kirjeldustega.

Nurksulgudes [*a*] olev element võib esineda keelekonstruktsiooni antud kohas, kuid ei ole kohustuslik. Kui nurksulgudes asuvale elemendile järgneb punktiir ([*a*]…), tähendab see, et antud element võib puududa või korduda suvaline arv kordi.

Püstkriipsu | kasutatakse tähenduses "või", selle abil esitatakse võimalikud variandid.

Näiteks funktsiooni päise kirjelduse nö esimesel tasemel võib formaalselt esitada järgmiselt:

def *nimi* ([*parameetrid*]) :

Sellest võib välja lugeda, et **päis** peab algama võtmesõnaga **def**, sellele järgneb **funktsiooni *nimi***, edasi tulevad **sulud**, mille sees võivad olla ***parameetrid*** ning päise lõpus peab olema **koolon**. Võib ka järeldada, et isegi kui parameetrid puuduvad, peavad tühjad sulud ikkagi olema.

Täpsustame parameetrite esitamise reeglit:

parameetrid ::= *parameeter* [, *parameeter*] …

Mõnikord kasutatakse kirjelduses märke ::= tähenduses „on“. Kirjeldusest järeldub, et parameetreid võib olla üks või mitu (suvaline hulk), kusjuures mitme parameetri korral eraldatakse need üksteisest komadega.

parameeter ::= *nimi* [= *väärtus*]

väärtus ::= *string* | *täisarv* | *reaalarv* | *tõeväärtus*

*nim*i ::= *täht* | _ [*täht* | *number* | _] …

Nimi võib koosneda ühest tähest või allkriipsust või tähtede, numbrite või allkriipsude jadast, mis algab tähe või allkriipsuga.

Toome näiteks veel programmi rea määratluse:

[*lause* [**;** *lause*]…] [*# kommentaar*]

Ühel real võib olla mitu lauset, mis eraldatakse semikoolonitega. Rea lõpus võib olla kommentaar, mis peab algama märgiga **#**. Real võib olla ainult üks või mitu lauset või ainult kommentaar. Rida võib olla ka tühi.

Lausete põhielemendid

Laused on korraldused (käsud), mis määravad vajalikke tegevusi. Lausetel on keelereeglitega määratud kindel otstarve ja struktuur. Lausetes võivad esineda:

- **võtmesõnad**: kindla asukoha, kuju ja tähendusega sõnad ja lühendid: if**,** else**,** elif**,** return**,** for**,** in**,** while**,** break**,** continue**,** True**,** False**,** and**,** or**,** not**,** def**,** class jm
- **viited funktsioonidele** ja **meetoditele**, [moodul**.**] nimi([argumendid]): saledus(L, m), korruta(), print(*“Summa=“*, S), input(*'a='*), random.randint(2, 9), time.clock(), math.sin(x)**,** penup()
- **avaldised**: m / (L / 100) ** 2, str(a) + ' * ' + str(b), indeks < 18
- **konstandid**: *"kõhn"*, *'Sinu nimi => '*, 100, 18, 0.01, 9, True
- muutujate, parameetrite ja objektide **nimed**: *a*, *b*, *v*, *pikkus*, *indeks*, *ind*, *L*, *m* , *radom, time*
- **tehtemärgid**: +, -, /, *, **, =, ==, <=, +=
- **piirajaid** ja **eraldajaid**: ( ) ” “ : . ;
- **…**

Struktuuri järgi jagunevad laused kahte gruppi: **lihtlaused** ja **liitlaused**.

Lihtlaused

Lihtlause, nagu ütleb nimi, ei sisalda teisi lauseid. Mõned lihtlausete näited:

- **omistamislause**, *muutuja* = *avaldis* : x = 0; k = 1; y = 2*math.sin(x)+3*math.cos(x-2); nimi = *“Juku“*; mid = ideaal(L, vanus, sugu); k = k + 1; x = x + h; L = float(input(*'pikkus='*));
- **pöördumislause**, [*moodul***.**] *nimi*([*argumendid*]): saledus(L, m); korrutamistabel()
  print(*“Summa=“*, S, *“keskmine=“*, k); random.randint(2, 9); time.clock(); math.sqrt(x); penup()
- **return**-lause, return [avaldis [, avaldis]…]: return; return pindala; return x1, x2, tun
- **break**-lause, **break**
- …

Liitlaused

Liitlaused sisaldavad teisi liht- ja/või liitlauseid. Liitlausete korral on Pythonis ja ka teistes programmeerimiskeeltes probleemiks lausete struktuuri kindlakstegemine, seda nii inimese kui ka interpretaatori jaoks – millised laused kuuluvad antud lause sisse, kus lõpeb üks lause ja algab teine jms.

Programmeerimiskeeltes kasutatakse liitlausete struktuuri täpseks fikseerimiseks erinevaid lahendusi. Vaatleme neid **if**-lause näitel, mille olemus ja tööpõhimõte on sarnane kõikides keeltes. Scratchis on asi lahendatud väga lihtsalt ja selgelt: sisemised plokid paigutatakse **kui** ja **muidu** harudesse. Visual Basicus ja mõnes teises keeles (Ada), määratakse **Else-**haru ja **If**-lause lõpp spetsiaalse **End If**-osalause abil. Mitmes keeles (C-pere, Java, Pascal) kasutatakse liitlausete struktuuri fikseerimiseks **looksulge** või võtmesõnade paare: **begin** … **end**. Analoogsed probleemid on seotud ka teiste liitlausetega.

Scratch, BYOB

Visual Basic, Ada

If *tingimus* **Then**
  laused 1
Else
  laused 2
End If
järgmine lause

C-pere**, Java**

If *tingimus* {
  laused_1
}
else {
  laused_2
}
järgmine lause

Python

if *tingimus* **:**
  laused_1
else :
  laused_2
järgmine lause

Pythonis on struktuuri ja lause lõpu määramisel oluline roll kohustuslikel **taanetel**. Tulemus on põhimõtteliselt lihtne, ülevaatlik ja kompaktne, kuid nõuab veidi harjumist ja tähelepanu. Ülaloleval skeemil peab *järgmise lause* taane olema samasugune kui **if**-lausel ja **else**-osalausel. Kui aga taane on võrdne *laused_2* taandega, kuulub *järgmine lause* **else**-osalausesse ja programm ei tööta päris õigesti. Teistsugust taanet olla ei saagi, sest Python kontrollib taandeid ja kui need ei sobi, kuvab veateate.

Taolise skeemiga puutusime me kokku juba esimeses näites **Tutvus**, milles oli valiku kirjeldamine **if**- ja **else**-lause abil. Kui viimasel lausel **print** (*'Kohtumiseni! Igavesti Sinu, Python!'*) oleks taane, mis võib olla võrdne ainult eelmise lause taandega, kuuluks see **else**-osalausesse. Sellisel juhul kuvatakse teade ainult siis, kui tingimus **if**-lauses on väär, kuid seda peab kuvama alati.

Liitlause esitamise põhivariant Pythonis on järgmine:

liitlause päis :
  liitlause keha – plokk

Päise esitus sõltub lause tüübist: **if**-, **else**-, **for**-, **while**-, **def**-lause jm.; lõpus peab tingimata olema **koolon :**. **Liitlause keha** on plokk, mis võib koosneda liht- ja liitlausete jadast. Kõikidel sama tasemega lausetel peab olema ühesugune taane.

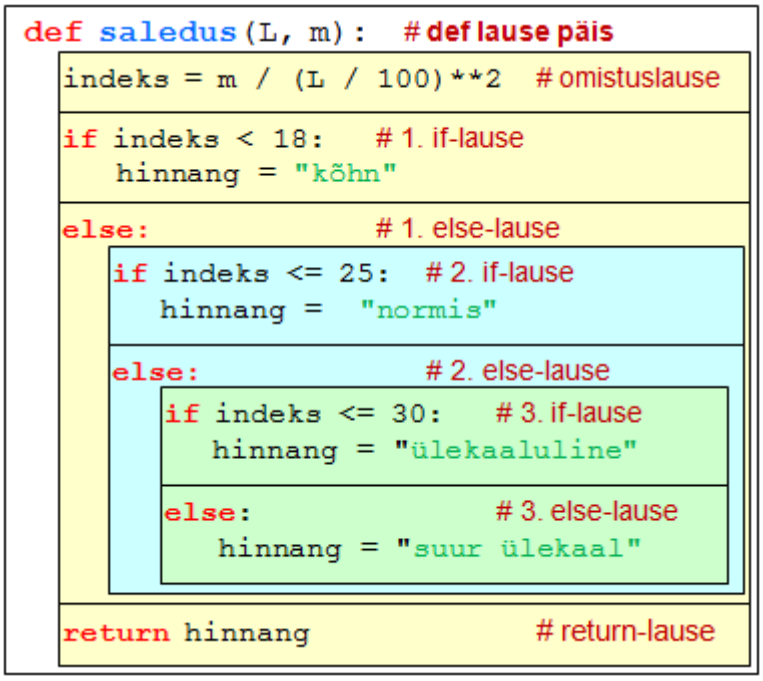
On lubatud, kuid mitte eriti soovitatav, liitlause esitamine ühel real järgmisel kujul:

liitlause päis : lihtlause [**;** lihtlause] **…**

Sellist esitusviisi võiks erandkorras kasutada juhul, kui liitlausesse kuulub 1 kuni 2 lühikest lihtlauset, näiteks

  if a > 0 **:** S += a **;** n +=1

Järgnevalt käsitleme üldisemat juhtu – funktsiooni **saledus**, mis kujutab formaalselt ühte **def**-liitlauset.



def-lausesse kuulub antud juhul neli lauset: omistamislause, **if**-lause ja **else**-osalause ning **return**-lause.

Esimene ja viimane on lihtlaused, teine ja kolmas liitlaused. Kõigil neljal lausel peab olema kindlasti ühesugune taane, **if**- ja **else**-lause puhul on tegemist nende päistega, sest siselausetel on juba järgmise tasemega taanded. **else**-osalausesse kuuluvad omakorda **if**- ja **else**-laused ja viimasesse veel järgmised.

Kui **return**-lause taane oleks võrdne teise **else**-osalause taandega, kuuluks see viimase sisse. Sellisel juhul kui tingimus indeks<18 on tõene, jääb **return**-lause täitmata ja hinnangut ei tagastata.

Taanetel on oluline koht ka liitlause ilmekuse ja loetavuse seisukohalt ja seda kõikides programmeerimis-keeltes. Keeltes, kus taanded ei ole kohustuslikud, soovitatakse neid loetavuse parandamise mõttes siiski kasutada. Näites on toodud sama funktsioon Visual Basicus ilma taaneteta (formaalselt lubatud) ja taanetega. Võrdluseks on kõrval veel kord toodud see funktsioon Pythonis.

```
Function Saledus(L, m)
indeks = m / (L / 100) ^ 2
If indeks < 18 Then
hinnang = "kõhn"
Else
If indeks <= 25 Then
hinnang = "normis"
Else
If indeks <= 30 Then
hinnang = "ülekaaluline"
Else
hinnang = "suur ülekaal"
End If
End If
End If
saledus = hinnang
End Function
```

```
Function Saledus(L, m):
  indeks = m / (L / 100) ^ 2
  If indeks < 18 Then
    hinnang = "kõhn"
  Else
    If indeks <= 25 Then
      hinnang = "normis"
    Else
      If indeks <= 30 Then
        hinnang = "ülekaaluline"
      Else
        hinnang = " suur ülekaal "
      End If
    End If
  End If
  saledus = hinnang
End Function
```

```
def saledus(L, m):
  indeks = m / (L / 100) ** 2
  if indeks < 18:
    hinnang = "kõhn"
  else:
    if indeks <= 25:
      hinnang = "normis"
    else:
      if indeks <= 30:
        hinnang = "ülekaaluline"
      else:
        hinnang = "suur ülekaal"
  return hinnang
```

Sügav sisaldavuse aste liitlauses vähendab programmi teksti ilmekust ja raskendab loetavust. Võimaluse korral peaks püüdlema sisaldavuse taseme vähendamisele.

```
def saledus(L, m):
  indeks = m / (L / 100) ** 2
  if indeks < 18:
     hinnang = "kõhn"
  elif: indeks <= 25:
     hinnang =  "normis"
  elif: indeks <= 30:
      hinnang =  "ülekaaluline"
  else:
      hinnang =  "suur ülekaal"
  return hinnang
```

Valiku kirjeldamisel **if**-lausega on Pythonis võimalus kasutada **elif**-osalauseid. Lause täitmisel kontrollitakse kõigepealt tingimust **If**-osalauses, kui see on tõene, täidetakse if_osalause tegevused, kõik ülejäänu jääb vahele. Vastupidisel juhul kontrollitakse järjest tingimusi **elif**-osalausetes (kui neid on olemas) ning kui leitakse **esimene tõene**, täidetakse järgnevad tegevused, kõik ülejäänu jääb vahele. Kui ükski tingimus ei ole tõene, täidetakse else_osalause tegevused (kui need on).

Kõrvaltoodud funktsiooni **saledus** variandil ehk liitlausel **def** on ainult kaks sisaldavuse astet. Selles on omistamislause, **if**-lause, kaks **elif**-osaluset, **else**-osalause ja **return**-lause. Kõikides sisalduvates liitlausetes on ainult üks lihtlause.

Toodud funktsiooni võiks esitada ka järgmiselt:

```
def saledus(L, m) :
  indeks = m / (L / 100) ** 2
  if    indeks < 18 :   return  "kõhn"
  elif  indeks <= 25 :  return  "normis"
  elif  indeks <= 30 :  return  "ülekaaluline"
  else :                return  "suur ülekaal"
```

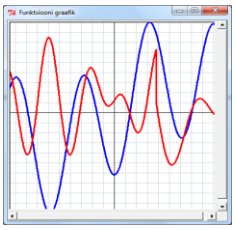
Siin on loobutud muutujast ***hinnang*** ning tagastatav väärtus esitatakse kõikidel juhtudel otse **return**-lauses. Viimane aga on esitatud vastavas üherealises liitlauses.

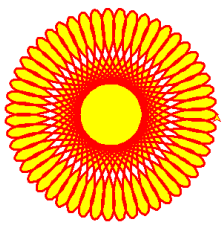
Protseduuride kasutamine on üheks sageli kasutatavaks võimaluseks sisaldavuse taseme vähendamiseks ning programmide struktuuri ja loetavuse parandamiseks. Kui näiteks loobuda programmis **Tutvus** funktsiooni **saledus** kasutamisest ja paigutada vastavad tegevused peaprotseduuri, suureneks selle sisaldavuse tase ning jälgitavus halveneks.

Veidi kilpkonnagraafikast

Pythoni kooseisu kuulub moodul **turtle**, mis sisaldab nn „kilpkonnagraafika“ vahendeid. Joonestamise operatsioone täidab spetsiaalne graafikaobjekt – ***turtle*** (kilpkonn) – Scratchi spraidi analoog. Kilpkonni (objekte) võib olla mitu ja neil võivad olla erinevad kujud. Vaikimisi on kilpkonni üks ja sellel on nooletaoline kuju ➤. Kilpkonnaga on seotud **pliiats** ja objektil on suur hulk meetodeid ehk käske, mille abil saab seda liigutada, pöörata, muuta pliiatsi suurust, värvust jms.

Kui programmis on joonistamiskäske, kuvatakse automaatselt graafikaaken ***Python Turtle Graphics***. Programmis saab määrata akna suuruse. Kui seda ei tehta, valib süsteem mõõtmed ise, arvestades arvuti ekraani suurust. Mõõtühikuks on piksel. Koordinaatsüsteemi nullpunkt on akna keskel.





Iseenesest on kilpkonna kasutamine üsna selge ja lihtne ning selle mõistmiseks ja kasutamiseks eriti suuri eelteadmisi vaja ei ole. Tegemist on joonistamise robotiga. Vastavate meetodite abil nagu **forward**(d), **setpos**(x, y), saab muuta selle asukohta, pöörata – **left**(nurk), **right**(nurk), muuta pliiatsi suurust ja värvi – **pensize**(w) ja **pencolor**(v) jm. Üldine juhtimine ja vajalikud arvutused tehakse tavaliste Pythoni lausete abil nagu kordused, valikud jms.

Järgnevalt on toodud paar lihtsat näidet. Lugeja võiks proovida neid Pythonis käivitada ning täiendada ja muuta.

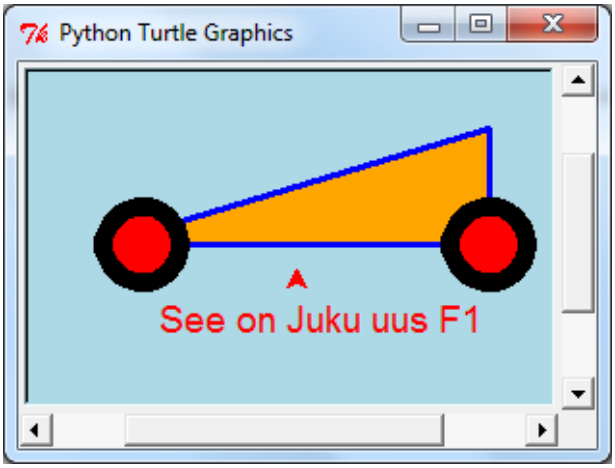
Lisainfot saab hankida Pythoni dokumentatsioonist ja jaotisest „Graafikaandmed ja graafikavahendid Pythonis“, lk 196.

Näide: Superauto ehk Juku F1

```
from turtle import *
setup(300, 200)       # akna mõõtmed
bgcolor("lightblue")  # akna põhja värv
# auto kere (kolmnurga) joonistamine
penup(); setpos(-100,0); pensize(3)
pencolor("blue"); fillcolor("orange")
begin_fill()  # alusta täitmist
pendown(); forward(200);   left(90)
forward(60) ;  setpos(-100, 0)
end_fill()    # lõpeta täitmine
# ratta (täidetud ringi) joonistamine
setpos(-60, 0)  # pliiats ringi algpunkti
color('black', 'red'); pensize(10)
begin_fill(); circle(20); end_fill()
# teise ratta joonistamine
penup(); setpos(120,0); pendown()
begin_fill(); circle(20); end_fill()
# teksti kirjutamine graafikaaknasse
penup(); setpos(-70, -50) # teksti algus
pencolor('red')
write ("See on Juku uus F1", font = ('Arial', 14) )
home()  # koju – akna keskpunkti  (0, 0)
left(90); backward(13)  #  vasakule, alla
mainloop()  # graafikaaken ooteseisu
```

Programm joonistab mingi autotaolise kujundi: täidetud kolmnurk – kere, ringid – rattad.

Kilpkonna meetodite importimiseks moodulist **turtle** kasutatakse käsku **from**, tänu millele saab kasutada meetodeid ilma viiteta objektile (kilpkonnale).



Pliiatsi värvi ja täitevärvid saab määrata eraldi meetoditega **pencolor**(jv) ja **fillcolor**(tv) või meetodiga **color**(jv, tv) koos. Värvid esitatakse nimetuste (tekstikonstatide) abil: "red", "blue" , ...

Käsud **forward**(d) ja **backward**(d) viivad kilpkonna praegusest asukohast edasi või tagasi, arvestades suunda, mis on alguses paremale (0°).

Suunda muuta saab käskudega **left**() ja **right**().

Käsk **setpos**(x, y) viib punkti (x, y), suunda muutmata.

Ringi keskpunkt on pliiatsist (kilpkonnast) vasakul, **r** piksli kaugusel, näites r=20.

Näiteks kui pliiats on punktis (-60, 0) ja suund on 90°, on ringi keskpunkt (-80, 0).

Näide: Võrdse pindalaga ristkülik ja ring

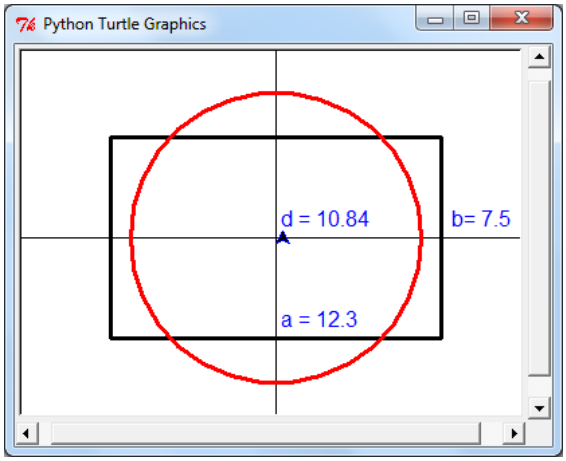
Rakendus leiab ristküliku külgede **a** ja **b** alusel ringi läbimõõdu **d**, mille pindala **S** on võrdne ristküliku pindalaga. Leitakse ka ristküliku ümbermõõdu ja ringjoone pikkuse **suhe** ning tehakse skeem.

Kasutatakse järgmisi seoseid: S = a * b, P = 2 (a + b), $d = \sqrt{4 \cdot s / \pi}$, suhe = P/(π * d)

```
from math import *
from turtle import *

# Algandmete lugemine
a = float(input("laius => "))
b = float(input("kõrgus => "))
m = float(input("mastaap => "))
#  Arvutamine
S = a * b;   P = 2 * (a + b)
d = sqrt(4 * S / pi);  suhe = P / (pi * d)

# väljund Shelli
print ( "laius =", a, " kõrgus =", b)
print ( "pindala =", round(S, 2))
print ( "läbimõõt =", round(d, 2))
print ( "suhe =", round(suhe, 2))

# Joonistamine
speed(0)  # joonistamise kiirus max !
# teljed. W - akna laius, H - kõrgus
W = window_width();  H = window_height()
penup(); setpos(-W/2, 0); pendown(); setpos(W/2, 0)
penup(); setpos(0, -H/2); pendown(); setpos(0, H/2)
# ristkülik
penup(); pensize(3)
am = a * m; bm = b * m   # korrutamine mastaabiga
setpos(-am / 2, -bm / 2)  # alumisse vasakusse nurka
pendown()
# üles, paremale, alla, algusesse
setpos(-am / 2, bm/2);  setpos(am / 2, bm/2)
setpos(am / 2, -bm /2);  setpos(-am / 2, -bm / 2)
penup()
# ring, keskpunkt vasakul kilpkonnast kaugusel  r(d/2)
pencolor( "red"); setpos(m * d / 2, 0)  # r = m * d / 2
setheading(90);  pendown(); circle(m * d/2)

# teksti kirjutamine graafikaaknasse
pencolor ( "blue")
penup(); setpos(am/2+ 5, 5)
write(" b= " + str(b), font=( "Arial", 12))
penup(); setpos(5, -bm/2 + 5)
write("a = " + str(a), font=( "Arial", 12))
penup(); setpos(5, 5)
write( "d = " + str(round(d, 2)), font=("Arial", 12))
mainloop()  # graafikaaken ooteseisu
```

Kuna joonestamise ühik (piksel) on üsna väike, kasutatakse siin mastaabitegurit ***m***: pikslite arv (ca 20–30) pikkusühikule. Sellega saab reguleerida joonise mõõtmeid graafikaaknas.

Peale algandmete sisestamist arvutatakse muutujate **S**, **P**, **d** ja **suhe** väärtused ning väljastatakse algandmed ja tulemused käsuaknasse. Graafikaakna mõõtmed (W – laius, H – kõrgus) valib Python, programm kasutab neid telgede joonestamisel.



Ristküliku joonestamisel viiakse pliiats (kilpkonn) kõigepealt ristküliku alumisse vasakpoolsesse nurka. Siit liigutakse üles, paremale, alla ja algpunkti tagasi.

Enne ringi joonestamist viiakse kilpkonn punkti (m*d/2) ja pööratakse üles (90 kraadi). Käsk **circle**(**r**) joonistab ringi raadiusega **r**, mille **keskpunkt** on **r** ühikut kilpkonnast **vasemal**.

NB! Kilpkonna suund on üles ▲.

Käsk **write**(tekst, …) võimaldab kirjutada teksti alates kilpkonna asukohast. Arvud peab enne väljastamist teisendama tekstivormingusse funktsiooniga **str**(). Saab anda kirja tüübi ja suuruse.

Kasutajafunktsioonid ja moodulid

Kasutajafunktsioonide ehk **protseduuridega** tutvusime juba esimeses näites. Järgnevas püüame laiendada selle valdkonna teadmisi ja oskusi.

Funktsiooni olemus, struktuur ja põhielemendid

Kaasajal mängivad protseduurid ja funktsioonid rakenduste loomisel erakordselt olulist rolli. Vähegi tõsisemates programmides kasutatakse selle jaotamist üksusteks ehk komponentideks, sest nii saab luua paindlikumaid, töökindlamaid ning paremini hallatavaid ja häälestuvaid rakendusi ja süsteeme. Kord loodud protseduure saab aga korduvalt kasutada erinevates rakendustes.

Mõni sõna terminitest. **Funktsioonid** olid algselt (ja mitmetes süsteemides nagu Fortran, Pascal, ADA, Basic, Scratch ka praegu) mõeldud ainult ühe väärtuse (arv, tekst, kuupäev, tõeväärtus, ...) leidmiseks ja tagastamiseks. Programmiüksusi, mida kasutatakse suvaliste tegevuste kirjeldamiseks, nimetatakse **protseduurideks** või **alamprogrammideks**. Pythonis nagu ka mitmetes teistes programmeerimis-süsteemides nimetatakse kõiki protseduure funktsioonideks. Funktsioonid võivad määrata suvalisi tegevusi (joonestada, sisestada ja kuvada väärtusi, sisaldada andmevahetust failide ja andmebaasidega, ...) ning leida ja tagastada väärtusi – ühe või mitu. Mitmed autorid nimetavad Pythoni programmiüksusi vahel funktsioonideks, vahel protseduurideks ning nii ka siin.

Funktsiooniga/protseduuriga on seotud kaks aspekti: funktsiooni definitsioon ehk määratlus ja pöördumine funktsiooni poole. Formaalselt – süntaksi seisukohalt – kujutab funktsiooni definitsioon endast liitlauset, mille üldkuju on järgmine:

def nimi ([parameetrid]):
    [*docstring*]
    lause
    [lause]
    …

Funktsiooni päis algab võtmesõnaga **def**, millele järgneb nimi ja sulgudes parameetrid. Tühjad sulud peavad olema ka siis, kui parameetreid ei kasutata. Päise lõpus peab tingimata olema **koolon**.

Funktsiooni sisu ehk keha koosneb lausetest, mis peavad paiknema päise suhtes **taandega**. Otstarbekas on panna algusesse kolmekordsete piirajate vahele nn dokumendistring (vt lk 165).

Parameetrid esindavad funktsiooni sisendandmeid ja esitatakse kujul: *parameeter* [**,** *parameeter*] …, kus *parameeter* ::= *nimi* [**=** *väärtus*]. Parameetrid võivad esindada skalaarseid suurusi, loendeid, objekte ja funktsioone. Väärtus näitab vaikimisi võetavat väärtust – kui vastav argument pöördumises puudub, kasutatakse parameetrile omistatud väärtust. Parameetrite loetelus peavad parameetrid, mille jaoks pole esitatud vaikeväärtust, asuma eespool, vaikeväärtustega parameetrid on loetelu lõpus.

Funktsiooni poole pöördumises peab argumentide esitamisel arvestama parameetrite järjekorda, kui ei kasutata parameetrite nimesid. Kohustuslik on esitada vaikeväärtuseta parameetritele vastavad argumendid.

Järgnev näiteprotseduur joonistab ruudu küljepikkusega **a**, alguspunktiga (**x**, **y**), milleks on alumine vasak-poolne nurk. Vaikimisi võrduvad **x** ja **y** nulliga. Parameeter **w** määrab joone paksuse. Funktsioon leiab ja tagastab ruudu ümbermõõdu ja pindala.

```
from turtle import *
def ruut (a, w, x = 0, y = 0) :
  ''' ruut küljega a; x, y – koh
    w – joone paksus'''
  penup(); setpos(x, y)
  pensize(w); pendown()
  for i in range(4):
    forward(a)
    left (90)
  P = 4 * a; S = a * a
  return  P, S
```

Mõned võimalikud pöördumised:

```
ruut(100, 2)  # ainult nimedeta kohustulikud argumendid
ruut(150, y = -80, w = 5) # 1. nimeta, teised nimedega
ruut(x = -130, a = 200, w = 3) # kõik nimedega, osa puuduvad
# NB! nimedega argumendid võivad suvalises järjekorras
P, pind = ruut(30, 3)   #  P ja pind = tagastatavad väärtused
print ("pindala", pind)  # kasutatakse (kuvatakse) ainult ühte
```

Parameetrite nimed (näites: a, w, x, y), aga samuti funktsiooni sees kasutatavad muutujad (P ja S) omavad tähendust ja on kättesaadavad ainult antud protseduuris. Öeldakse, et **protseduur lokaliseerib** oma **parameetrid** ja **sisemuutujad**.

Pöördumine funktsiooni poole esitatakse kujul:

[*muutuja* [, *muutuja*] … =] ***nimi*** ([*argument* [, *argument*] …])

Siin on ***nimi*** funktsiooni nimi. Argumentide abil antakse vastavatele parameetritele tegelikud väärtused. Väärtuste esitamiseks võib kasutada konstante, muutujaid ja avaldisi. Argumentide esitamisel võib kasutada parameetrite nimesid, mida võib teha ka positsioonilistele parameetritele vastavate argumentide jaoks. Selliste argumentide esitusviisi korral ei ole nende järjekord oluline.

Vasak pool (*muutujad*) võib pöördumislauses olla, kui funktsioon tagastab **return**-lausega väärtusi. Muutujate arv peab võrduma tagastatavate väärtuste arvuga. Paneme tähele, et pöördumisel ei pea funktsiooni poolt tagastatavaid väärtusi tingimata vastu võtma.

Näite esimeses kolmes pöördumises ei võeta vastu tagastatavaid väärtusi. Seda tehakse ainult viimases pöördumises: P, pind = ruut(30, 3). Kuna **return**-lause tagastab kaks väärtust, on vasakus pooles samuti kaks muutujat. Kuvatakse ainult pindala.

return-lause üldkuju on järgmine:

return *avaldis* [, *avaldis*] …]

Lause **return** lõpetab funktsiooni töö ja tagastab avaldis(t)e väärtuse(d), andes täitmisjärje punkti, kust toimus pöördumine. Kas tagastatavad väärtused ka vastu võetakse, sõltub pöördumise viisist. Funktsioonis võib olla suvaline arv **return**-lauseid.

Mitmeprotseduurilistes programmides on andmevahetusel protseduuride vahel oluline roll, sest üks protseduur ei pääse ligi – sageli ei tohigi – teise protseduuri andmetele. Andmevahetuseks kasutatakse erinevaid viise. Peamisteks on just siin vaadeldud võimalused: parameetrite ja argumentide mehhanism ning väärtuste tagastamine **return**-tüüpi lausetega. Erinevates programmeerimiskeeltes kasutatakse mõnevõrra erinevaid viise.

Paljudes keeltes (Fortran, Pascal, Basic, C-pere, Java jt) jagunevad parameetrid kolme rühma: **sisendparameetrid**, **väljundparameetrid** ja **sisend-väljund parameetrid**.

- **sisendparameetrid** saavad väärtused argumentidelt pöördumisel, neid kasutatakse protseduuri täitmisel, kuid neid ei muudeta (tavaliselt ei saagi muuta),
- **väljundparameetritele** omistatakse väärtused protseduuri täitmise ajal ja need väärtused omistatakse vastavatele argumentidele,
- **sisend-väljund parameetrid** omavad väärtusi enne pöördumist, neid kasutatakse ja muudetakse protseduuri täitmisel.

Pythonis on ainult sisendparameetrid, mida protseduuri täitmisel ei muudeta. Väljundparameetrite ja sisend-väljund parameetrite puudumist kompenseerib võimalus tagastada **return**-lausega mitu väärtust. Teistes keeltes saab taolisel viisil tagastada ainult ühe väärtuse.

Esitatu kehtib üksikväärtuste ehk skalaarandmete kohta. Praktiliselt kõikides keeltes, sh ka Pythonis, saab kasutada omamoodi sisend-väljund parameetritena massiive ja loendeid. Protseduur saab lugeda ja muuta parameetriks oleva loendi elementide väärtusi ja pöördunud protseduur saab muudetud väärtused kätte.

Üheks andmevahetuse võimaluseks enamikus keeltes on nn globaalsete andmete kasutamine, millele on juurdepääs mitmel protseduuril. Ka Pythonis on olemas lause **global**, millega saab kuulutada andmed globaalseteks. Arvestades mitmesuguseid võimalikke kõrvalmõjusid, ei ole taoline andmevahetuse viis eriti levinud ega mõistlik.

Andmevahetuseks kasutatakse ka faile (andmebaase ja dokumente), eriti siis, kui tegemist on suuremate andmehulkadega. Nii võib näiteks üks protseduur kirjutada andmed faili, teine protseduur neid aga sealt lugeda.

Ning lõpuks lausetest, mis moodustavad funktsiooni keha, milles võivad olla suvalised liht- ja liitlaused, sh ka **def**-laused. See tähendab, et Pythonis võib üks protseduur sisaldada teisi protseduure. Viimast võimalust kasutatakse suhteliselt harva.

Näide: Kolmnurga pindala

Funktsioon leiab kolmnurga pindala külgede pikkuste alusel. Arvestatakse võimalusega, et kolmnurka moodustada ei saa. Pindala leidmiseks kasutatakse Heroni valemit:

$$S = \sqrt{p(p-a)(p-b)(p-c)}$$, kus p = (a+b+c)/2.

Kui tingimus a+b<=c või b+c<=a või a+c<=b on tõene, siis kolmnurka moodustada ei saa.

```
def K_pind(a, b, c):
    ''' Kolmnurga pindala. a, b, c – küljed.
        Tagastab pindala või -1 '''
    if a + b <= c or b + c <= a or a + c <= b :
        return -1 # kolmnurka ei ole
    else:
        p = (a + b + c) / 2
        S = (p * (p -a) * (p - b) * (p - c)) ** (0.5)
        return S
# Funktsiooni katsetamine
a, b, c = eval (input("külgede pikkused => "))
  # korraga võib lugeda mitme muutuja väärtused
  # eval teisendab väärtused arvuvormingusse
pind = K_pind(a, b, c)
if pind == -1:
    print("Kolmnurka ei ole!", a, b, c)
else:
    print("küljed:", a, b, c, " pind=", round(pind, 3))
```

Funktsiooni töö testimisel demonstreeritakse võimalust korraga mitme muutuja väärtuse sisestamiseks. Sisestuslause võib arvude korral olla esitatud kujul:

muutuja [, *muutuja*]… = **eval**(**input**([teade]))

Mitme väärtuse sisestamisel eraldatakse need üksteisest komadega.

Funktsioon **eval**() on üldiselt ette nähtud avaldise väärtuse leidmiseks. Tegemist võib olla ka tekstavaldisega. Kui tegemist on tekstivormingus arvudega, teisendatakse need arvuvormingusse, analoogselt funktsioonidega **int**() ja **float**().

Näide: Ruutvõrrand

Järgnevalt on toodud funktsioon ruutvõrrandi $ax^2 + bx + c = 0$ lahendite **x1** ja **x2** leidmiseks. Arvestatakse võimalusega, et need võivad puududa. Kasutatakse spetsiaalset tunnust: kui lahendid puuduvad, võetakse tunnuse väärtuseks 0, muidu 1.

```
import math
def ruutvrd(a, b, c):
  ''' Ruutvõrrand. a, b, c  - kordajad.
     Tagastab tun = 1 | 0 – on ei ole
     x1, x2 – lahendid (kui on)'''
  D = b * b - 4 * a * c
  if D < 0:
    tun = 0; x1=""; x2=""
  else:
     D = math.sqrt(D)
     tun = 1
     x1 = (-b - D) / (2 * a)
     x2 = (-b + D) / (2 * a)
  return tun, x1, x2
```

```
# Peaprotseduur
print ("Lahendan ruutvõrrandeid!")
print ("Sisesta kordajad; a, b, c")
a = float(input( "Sisesta a => "))
while (a == 0):
  print ( " a ei tohi olla 0!!!")
  a = float(input("anna uus a => "))
b = float(input( "anna b => "))
c = float(input( "ja nüüd c => "))
tunnus, x1, x2 = ruutvrd(a, b, c)
if tunnus != 0:
  print("Siin need on:", x1, x2)
else:
  print("Lahendid puuduvad!!!")
```

Funktsiooni parameetriteks on ruutvõrrandi kordajad **a**, **b** ja **c**. Protseduur leiab diskriminandi ***D*** väärtuse. Kui see on negatiivne, omistatakse muutujale ***tun*** väärtus 0 ja muutujatele ***x1*** ja ***x2*** tühjad väärtused.

Peaprotseduur kontrollib kordajate sisestamisel ***a*** väärtust ja seni, kui see on null, küsib uut väärtust.

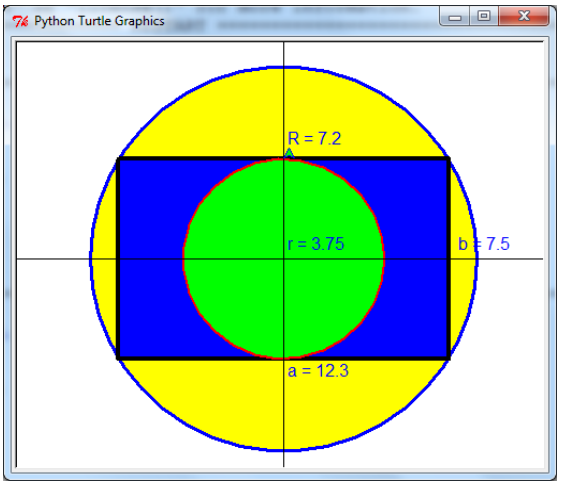
Pöördumisel funktsiooni poole lausega **tunnus, x1, x2 = ruutvrd(a, b, c)** arvestatakse, et see tagastab kolm väärtust.

Näide: Ristküliku karakteristikud

Rakendus leiab ristküliku külgede (**a** ja **b**) alusel selle pindala (**S**), ümbermõõdu (**P**), diagonaali (**d**), siseringi ja ümberringi raadiused (**r** ja **R**) ning ringide pindalad (**S1** ja **S2**). Programm joonistab ka ristküliku ja ringid. Programmi väljundid on näidatud allpool.

Python Shell

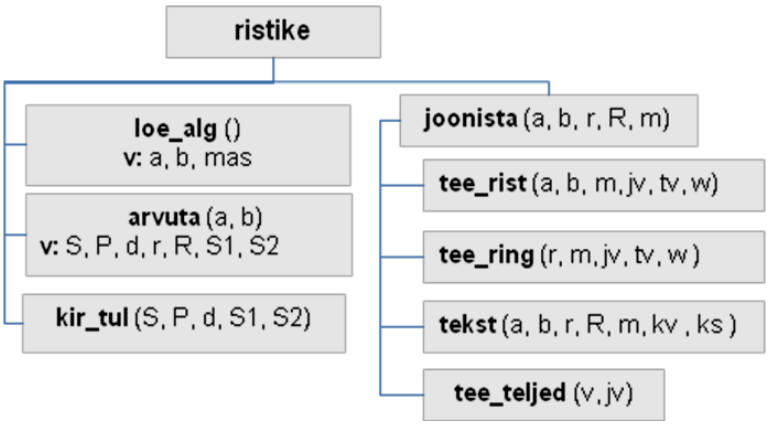
```
Python 3.2.2 (default, Sep  4 2011, 09:51:08)
>>>
Ristküliku omadused: pind, ümbermõõt jm
laius 12.3
kõrgus 7.5
mastaap (vaikimisi 25)
pindala = 92.25
ümbermõõt = 39.6
diagonaal = 14.41
siseringi pind = 44.18
ümberringi pind = 163.0
```



Programm on jagatud protseduurideks (funktsioonideks), arvestades täidetavate tegevuste iseloomu nagu algandmete sisestamine, arvutused, tulemuste kuvamine ja joonistamine. Antud juhul on tegemist väikese ülesandega, kus protseduuridest silmnähtavat kasu ei ole, kuid ettekujutus põhimõttest siiski tekib. Üheks eesmärgiks protseduuride kasutamisel on nendevaheline koostöö ja andmevahetuse korraldamise kinnistamine.

NB! Programmi jaotamine protseduurideks on rakenduse disaini juures üks olulisemaid tegevusi.

Peaprotseduuril **ristike** on neli alamprotseduuri: **loe_alg**, **arvuta**, **kir_tul** ja **joonista**. Viimasel on omakorda neli alamprotseduuri. Andmevahetuseks protseduuride vahel kasutatakse parametreid ja argumente.




```
from math import *
from turtle import *
def loe_alg ():
    ''' algandmete lugemine '''
    laius = float ( input ("laius ") )
    korgus = float ( input ("korgus "))
    mas = input("mastaap (vaikimisi 25) ")
    if mas == "":
        mas = 25
    else:  mas = int (mas)
    return laius, korgus, mas

def arvuta (a, b):
    ''' ristküliku põhiomadused '''
    S = a * b; P = 2 * (a + b)
    d = sqrt(a**2  +  b**2);  R = d / 2
    r = b / 2
    if a < b: r = a / 2    # r – pool lühemast küljest
    S1 = pi * r**2;  S2 = pi * R**2;
    return  S, P, d, r, R, S1, S2

def kir_tul (S, P, d, S1, S2):
    ''' tulemuste kirjutamine Shelli aknasse '''
    print ('pindala =', round(S, 2))
    print ('ümbermõõt =', round(P, 2))
    print ('diagonaal =', round(d, 2))
    print ('siseringi pind =', round(S1, 2))
    print ('ümberringi pind =', round(S2, 2))

def joonista (a, b, r, R, mas = 1):
    ''' joonistamise pealik '''
    tee_ring (R, mas, w = 3, jv="blue", tv="yellow")
    tee_rist (a, b, mas, w = 4, tv = "blue")
    tee_ring (r, mas, w = 2, jv="red", tv = "green")
    tee_teljed ()
    tekst (a, b, r, R, mas)
    mainloop()
```

Importimiseks kasutatakse käsku **from**, mis võimaldab kasutada funktsoonidele ja meetoditele viitamiseks kompaktsemat varianti.

Protseduur **loe_alg** loeb algandmed, milleks on ristküliku külgede pikkused. Parameetreid protseduuril ei ole. Tagastatavaid väärtusi on kolm: **laius**, **korgus** ja **mas** (mastaap).

Mastaabi jaoks on ette nähtud vaikimisi võetav väärtus: **25**. Kui kasutaja sisestab tühja väärtuse, omistatakse muutujale **mas** väärtus 25.

Funktsioon **arvuta**, leiab ja tagastab **return**-lause abil seitse väärtust. Funktsiooni sisendparameetriteks on külgede pikkused **a** ja **b**.

Tagastatavad väärtused: **S** – pindala, **P** – ümbermõõt, **d** – diagonaal, **r** ja **R** – siseringi ja ümberringi raadiused, **S1** ja **S2** – siseringi ja ümberringi pindalad.

Protseduur **kir_tul** kirjutab tulemused Shelli aknasse, kasutades Pythoni sisefunktsiooni **print**.

Väärtuste ümardamiseks kasutatakse Pythoni sisefunktsiooni **round**.

Protseduur **joonista** korraldab andmete väljastamist graafikaaknasse. Mastaabi jaoks on ette nähtud vaikimisi võetav väärtus, milleks on **1**.

Alamprotseduurides: **tee_ring**, **tee_rist**, **tee_teljed** ja **tekst** on mõnedel parameetritel vaikeväärtused. Pöördumisel on vastavad argumendid esitatud kasutades parameetrite nimesid. Erinevatel pöördumistel kasutatakse joone värvi (**jv**) ja paksuse (**w**) ning täitevärvi (**tv**) erinevaid väärtusi, mis asendavad parameetrite esialgsed väärtused.

```
def tee_teljed (w = 1, jv = "black"):
    ''' X ja Y teljed, w-joone paksus, jv -värv '''
    W = window_width();  H = window_height()
    pensize(w); pencolor(jv)
    penup(); setpos(-W/2, 0); pendown();
    setpos(W/2, 0);   penup();
    setpos(0, - H/2); pendown(); setpos(0, H/2)
```

Protseduur **tee_teljed** joonestab koordinaatteljed, mis läbivad akna keskpunkti (0, 0). Kuna pöördumisel ei ole joone paksust (**w**) ja joone värvi (**jv**) esitatud, jäävad kehtima parameetrite vaikeväärtused. Vastavate meetodite abil teeb protseduur kindlaks akna laiuse ja kõrguse ning omistab nende väärtused muutujatele **W** ja **H**, mida kasutatakse telgede joonestamisel.

```
def tee_rist (a, b, m=1, jv = 'black', tv = "", w=1):
    ''' ristkülik a*b, m - mastaap, jv – joone värv,
      tv - täitevärv, w - joone paksus'''
    am = a * m; bm = b * m
    penup(); setpos(-am / 2, -bm / 2)
    color(jv, tv); pensize (w); pendown()
    begin_fill()
    setpos(-am /2, bm/2); setpos(am/2, bm/2)
    setpos(am/2, -bm/2); setpos(-am/2, -bm/2)
    end_fill()
```

Protseduur **tee_rist** joonistab ristküliku külgede pikkustega **a** ja **b** ning keskpunktiga (0, 0). Protseduur pakub mõnevõrra rohkem võimalusi võtmevormingus esitatud parameetritega, kui kasutatakse antud programmis.

Mastaabi (**m**) vaikimisi võetavaks väärtuseks on 1. Kui vastav argument pöördumisel puudub, kasutatakse joonestamisel ühikuna pikselit. Joone jaoks saab kasutada erinevat paksust (**w**) ja värvi (**jv**). Kujund võib olla täidetud etteantud värviga või mitte (tv = "" – tühi).

```
def tee_ring (r, m = 1, jv = 'black', tv = "", w = 1):
    '''ring, r - raadius, m - mastaap, jv – joone värv
      tv - täitevärv, w - joone paksus'''
    penup(); setpos(m * r, 0); color(jv, tv)
    pensize(w); pendown()
    setheading(90); begin_fill()
    circle(m * r); end_fill()
```

Protseduur **tee_ring** joonistab ringi raadiusega **r** ja keskpunktiga (0, 0). Arvestatakse, et joonestamisel on ringi keskpunkt vasakul pliiatsist (kilpkonnast) kaugusel **r**. Pliiats viiakse käsuga **setpos** punkti (m*r, 0) ja suunaks määratakse käsuga **setheading** 90 kraadi. Parameetrite (**m**, **jv**, **tv** ja **w**) jaoks kasutatakse sama põhimõtet, mis oli ristküliku jaoks.

```
def tekst (a, b, r, R, m = 1, kv = "blue", ks = 12):
    ''' testi kirjutamine graafikaaknasse '''
    pencolor (kv); penup(); setpos(a * m/2 + 10, 5)
    write("b = " + str(b), font=( "Arial", ks))
    penup(); setpos(5, -b * m / 2-20)
    write("a = " + str(a), font=( "Arial", ks))
    penup(); setpos(5, 5)
    write( "r = "+str(round(r, 2)), font=( "Arial", ks))
    penup(); setpos(5, m * b/2 + 10)
    write( "R="+str(round(R, 2)), font=("Arial", ks))
```

Protseduur **tekst** kirjutab külgede pikkused (**a**, **b**) ja raadiused (**r**, **R**) graafikaaknasse.

Kasutusel on kolm võtmevormingus parameetrit:
m – mastaap,
kv – kirja värv (= joone värv) ja
ks – kirja suurus (pikselites).

Pliiats viiakse käsuga **setpos** vajalikku kohta ning kuvatav tekst esitatakse käsus **write** string-avaldise abil, kusjuures saab määrata ka kirja tüübi ja suuruse.

```
# peaprotseduur
print ('Ristküliku omadused: pind, ümbermõõt jm')
laius, korgus, mas = loe_alg()
S, P, d, r, R, S1, S2 = arvuta (laius, korgus)
kir_tul(S, P, d, S1, S2)
joonista (laius, korgus, r, R, mas)
```

Peaprotseduur käivitab järjest vastavad alamprotseduurid, edastab neile argumentide abil vajalikud väärtused ning võtab vastu protseduuride **loe_alg** ja **arvuta** tagastatavad väärtused.

Universaalne funktsioon Ristkülik

Ülalpool vaadeldud näites oli funktsioonidel **tee_rist** ja **tee_ring** oluline puudus – loodavate jooniste asukoht oli jäigalt seotud graafikaakna keskkohaga.

```
import turtle
def rist (kk,  x,  y,  a,  b,  m = 1,
          jv = 'black',  tv = "",  w = 1):
   ''' Joonistab ristküliku. Parameetrid:
     kk – kilpkonn,  m – mastaap: pikseleid  ühikule
     x, y – algus; a, b – laius ja kõrgus
     jv – joone värv, tv – täitevärv
     w – joone paksus, vaikimisi 1 piksel    '''
   x = m * x; y =  m * y; am = m * a; bm = m * b
   kk.penup(); kk.setpos(x, y)
   kk.width(w); kk.color(jv, tv)
   kk.pendown(); kk.begin_fill()
   kk.setpos(x + am, y); kk.setpos(x+ am, y +bm)
   kk.setpos(x, y + bm); kk.setpos(x, y)
   kk.end_fill()

aken = turtle.Screen()
J = turtle.Turtle()
rist (J, -150, -300,300,400, tv = "gray")
rist (J, -170, 80,340,20, tv = "darkgray")
rist (J, 0, 200, 120, 25, jv = 'gray', tv = 'white')
rist (J, 0, 225, 120, 25, jv = 'black', tv = 'black')
rist (J, 0, 250, 120, 25, jv = 'blue', tv = 'blue')
rist (J, 0 ,100, 6, 100, tv = "brown")
# aed
rist (J,-300, -260, 590, 10, jv = 'black', tv = 'green')
x = -300; y = -280; a = 10; b = 70
for i in range(30): # lipid
  rist(J, x, y, a, b, jv = 'black', tv = 'yellow')
  x = x + 20
rist(J,-300, -220, 590, 10, jv = 'black', tv = 'brown')
aken.mainloop()
```

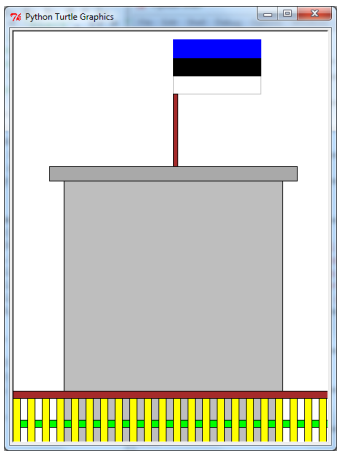
Antud juhul eeldatakse, et pöördumisel antakse **kk**-le vastav objekt (kilpkonn), vt näide.

Siin on võetud kasutusele parameetrid **x** ja **y**, mis määravad ristküliku alguspunkti: alumine vasakpoolne nurk. See võimaldab paigutada kujundi suvalisse kohta aknas.

Parameetri **m** (mastaap) vaikeväärtus on 1, st kui pöördumisel vastavale argumendile väärtust ei anta. Mõõtmed esitatakse pikslites. Enne joonistamist korrutatakse **x**, **y**, **a** ja **b** väärtused **m**-ga.

Kui pöördumisel parameetrile **tv** (täitevärv, vaikimisi tühi väärtus) vastavat argumenti ei anta, siis kujundit ei täideta

Pöördudes korduvalt protseduuri **rist** poole, saadakse selline pilt



Kasutajamoodul Inimene.py

Lihtsamal ja enamkasutatavamal juhul kujutab kasutajamoodul endast ühes failis (laiendiga **.py**) asuvat funktsioonide kogumit. Fail peab asuma seda kasutava programmiga samas kaustas või kaustas, kust Pythoni interpretator seda otsida oskab. Failis asuvad funktsioonid (protseduurid) saavad kasutatavaks käskude **import** või **from** täitmise järel.

Toome näiteks mooduli **inimene.py**, mis sisaldab funktsioone inimese omaduste leidmiseks: ideaalne mass, rasvaprotsent, ruumala jms. Vajalikud algandmed ja valemid on toodud allpool.

Algandmed:

L – pikkus (cm), **m** – mass (kg), **v** – vanus (aasta), **sugu** – mees/naine

Omadused:

$\mathbf{m_{id}}$ – ideaalne mass (kg), **r** – rasvaprotsentsus (%), ρ – tihedus(kg/m^3), **V** – ruumala(dm^3), **S** – pindala (m^2)

$$m_{id} = \begin{cases} (3 \cdot L - 450 + v) \cdot 0.225 + 40{,}5 \text{ naine} \\ (3 \cdot L - 450 + v) \cdot 0.250 + 45{,}0 \text{ mees} \end{cases} \quad r = \begin{cases} \frac{m - m_{id}}{m} \cdot m + 22 \text{ naine} \\ \frac{m - m_{id}}{m} \cdot m + 15 \text{ mees} \end{cases} \quad \rho = 8{,}9 \cdot r + 11 \cdot (100 - r)$$

$$V = \frac{1000 \cdot m}{\rho}, \quad S = \frac{(1000 \cdot m)^y \cdot L^{0,3}}{3118{,}2}, \text{ kus } y = (35{,}75 - \log_{10} m)/53{,}2, \; kind = m/(L/100)^2$$

Toodud funktsioonid on ühtlasi täiendavateks näideteks funktsioonide loomise ja kasutamise kohta.

Mõnel juhul on tegemist nö mitmetasemeliste funktsioonidega – üks funktsioon pöördub teise funktsiooni poole, see pöördub omakorda järgmise taseme funktsiooni poole jne.

```
import math
from datetime import *

def ideaal(L, v, sugu):
    ''' Inimese ideaalne mass (kg).
        L-pikkus (cm), v-vanus(aasta)
        sugu - n(naine), m(mees) '''
    abi = 3 * L - 450 + v
    if sugu[0].lower() == 'n' :
        mid = abi * 0.225 + 40.5
    else:
        mid = abi * 0.250 + 45.0
    return mid
```

Moodul **datetime** sisaldab meetodeid ja funktsioone tegevuste täitmiseks kuupäevade ja kellaegadega.

Tingimuse kontrollimisel eraldatakse parameetri sugu väärtusest esimene märk (indeksiga 0): sugu[0], teisendatakse see meetodiga lower() väiketäheks ja võrreldakse konstandiga *'n'*. See võimaldab esitada soo ühe väike- või suurtähega (n, m, N, M) või sõnaga (näit naine, Naine, mees, …) . Tegelikult omab tähtsust ainult see, kas esimene või ainukene täht on *'n'* (väike või suur) või mitte. Kui jah, eeldatakse, et tegemist on naisega, kui on midagi muud, kasutatakse mehe jaoks ette nähtud valemit.

```
def rasvaprots(L, m, v, sugu):
    ''' Rasvaprotsent(%), L-pikkus (cm),
        m - mass(kg), v-vanus(aasta),
        sugu - n(naine), m(mees) '''
    mid = ideaal(L, v, sugu)
    abi = (m - mid) / m * 100
    em = sugu[0].lower()
    if em == 'n':  return abi + 22
    else:          return abi + 15
```

Funktsioon **rasvaprots** pöördub eelmise funktsiooni poole ideaalse massi (**mid**) leidmiseks, kasutades argumentidena parameetrite kaudu saadud väärtusi.

Siin omistatakse parameetri **sugu** väiketäheks teisendatud esimene märk muutujale **em**, et võrdlus oleks lühem.

Tagastatav väärtus esitatakse avaldisena otse **return**-lauses.

```
def tihedus(L, m, v, sugu):
    ''' Tihedus (kg/m3). L-pikkus (cm),
        m - mass(kg); v - vanus(aasta)
        sugu - n(naine), m(mees) '''
    rasv = rasvaprots(L, m, v, sugu)
    return 8.9 * rasv + 11 * (100 - rasv)
```

Funktsioon **tihedus**, mille poole pöördutakse mõnest teisest protseduurist (näiteks funktsioonist **ruumala**), pöördub funktsiooni **rasvaprots** poole. Viimane pöördub omakorda funktsiooni **ideaal** poole.

```
def ruumala (L, m, v, sugu):
    ''' Ruumala (dm3/ltr). L-pikkus (cm),
        m - mass(kg); v-vanus(aasta)
        sugu - n(naine), m(mees) '''
    ro = ro = tihedus(L, m, v, sugu)
    return 1000 * m / ro
```

Siin toimub juba neli järjestikust pöördumist, kui arvestada ka pöördumist antud funktsiooni poole.

Selline suur pöördumiste ahel ei pruugi alati sobida, sest teeb funktsioonid üksteisest sõltuvaks. Vajadusel võib ühendada tegevused ühte protseduuri (vt **ruumala2**)

```
def ruumala2(L, m, v, sugu):
    ''' Ruumala (dm3/ltr). L-pikkus (cm),
        m - mass(kg); v-vanus(aasta)
        sugu - n(naine), m(mees) '''
    abi = 3 * L - 450 + v
    if sugu[0].lower() == 'n':
        mid = abi * 0.225 + 40.5
        rasv = (m - mid) / m * 100 + 22
    else:
        mid = abi * 0.250 + 45.0
        rasv = (m - mid) / m * 100 + 15
    ro = 8.9 * rasv + 11 * (100 - rasv)
    return 1000 * m / ro
```

Selles näites toodud funktsioon ühendab endas tegevused, mis olid eespool realiseeritud eraldi funktsioonidena. Nüüd on kõik tegevused ühes funktsioonis ning teisi funktsioone ei lähe vaja.

Soovi korral võib lisada tagastatavate väärtuste hulka ka **mid**, **rasv** ja **ro** väärtused.

```
def pindala(L, m):
    ''' Pindala (m2). L-pikkus (cm),
        m - mass(kg) '''
    y = (35.75 - math.log10(m)) / 53.2
    S = (1000 * m)**y * L ** 0.3 / 3118.2
    return S
```

Siin toimub lihtsalt väärtuste leidmine omistamislausete abil. Kasutataks funktsiooni **log10** moodulist **math**.

```
def saledus(L, m):
    ''' Kehamassi indeks ja hinnang.
        L-pikkus(cm),m - mass(kg) '''
    kind = m / (L/100)**2
    if   kind < 18:     hind = "kõhnake"
    elif kind <= 25:    hind = "normis"
    elif kind <= 30:    hind = "ülekaal"
    else:               hind = "suur ülekaal"
    return kind, hind
```

Selle funktsiooni analoog oli esimeses näiteprogrammis **Tutvus**. Praegu on kehamassiindeksi ja selle põhjal hinnangu andmine pandud ühte funktsiooni, mis tagastab kaks väärtust.

```
def sugu(kood):
    ''' isikukoodi alusel
    tagastab sugu'''
    em = kood[0]
    if int(em) % 2 == 0:
        return 'naine'
    else:
        return 'mees'
```

Siin ning järgnevates funktsioonides kasutatakse viitamist stringide elementidele indeksite abil, vt jaotist „Stringid (sõned) ja stringavaldised“, lk 189.

Isikukoodi esimene number näitab isiku sugu ja ka sünniaja sajandit. Paaritud numbrid (1, 3, 5) näitavad sugu „mees“, paarisnumbrid (2, 4, 6) „naine“.

```
def saeg(kood):
    ''' sünnikuupüev
        isikukoodi alusel'''
    en = kood[0]
    if en == '1' or en == '2' :
        ab = 1800
    elif en == '3' or en == '4' :
        ab = 1900
    elif en == '5' or en == '6' :
        ab = 2000
    else:
        return 0  # olematu sajand
    aasta2 = int(kood[1:3])
    aasta = ab + aasta2
    kuu = int(kood[3:5])
    paev = int(kood[5:7])
    skp = date(aasta, kuu, paev)
    return skp
```

Funktsioon tagastab isiku sünnikuupäeva kujul:

aasta – kuu – päev näiteks: 1989 – 03 – 25

Isiku sünniaasta kaks viimast numbrit on koodi 2. ja 3. positsioonis. Esimesed kaks numbrit (seotud sajandiga) on kodeeritud esimeses numbris: 1 ja 2 – 1800, 3 ja 4 – 1900, 5 ja 6 – 2000. Funktsioon eraldab kõigepealt koodi esimese numbri (indeks 0) ja omistab selle muutujale **en**. Edasi omistatakse sõltuvalt **en** väärtusest muutujale **ab** väärtus 1800, 1900 või 2000. Sellele liidetakse arv, mis saadakse koodi 2. ja 3. numbri alusel.

NB! Viitamist stringi lõikudele vt jaotisest „Stringid (sõned) ja stringavaldised“.

Mooduli **datetime** funktsioon **date** moodustab kuupäeva kolmest osast koosneva liitväärtusena. Väärtus tagastatakse **return**-lausega.

```
def vanusK(kood):
    ''' vanus aastates ja püevades
        isikukoodi alusel '''
    sa = saeg(kood)
    tana = date.today()
    vp = (tana - sa).days  # vanus päevades
    va = vp/365.25         # vanus aastades
    return va, vp
```

Funktsioonid **vanusK** ja **vanusA** tagastavad inimese vanuse aastates ja päevades kasutades vastavalt isikukoodi või sünnikuupäeva.

Funktsioonides kasutatakse mooduli **datetime** vahendeid.

```
def vanusA(paev, kuu, aasta):
    ''' vanus aastates ja püevades
        sünnikuupäeva alusel '''
    sa = date(aasta, kuu, paev)
    tana = date.today()
    vp = (tana - sa).days  # vanus päevades
    va = vp/365.25         # vanus aastades
    return va, vp
```

Siin antakse parameetritena isiku sünni kuupäev, kuu ja aasta. Funktsiooniga **date** moodustatakse kuupäev standardvormingus, mida saab kasutada tehetes kuupäevadega.

Järgnevalt on toodud näide mooduli kasutamise kohta.

```
from inimene import *
def isik():
    ik = input("Anna oma isikukood ")
    while len(ik) != 11:
        ik = input ("Peab olema 11 märki! Anna uus ")
    L = float(input("pikkus "))
    m = float(input("mass "))
    s = sugu(ik)
    va, vp = vanusK(ik)
    imas = ideaal(L, va, s)
    ruum = ruumala (L, m, va, s)
    print ("Oled sündinud", saeg(ik))
    print ("Sinu vanus on", round(va), "aastat")
    print ("Ideaalne mass", round(imas, 2))
    print ("ruumala", round(ruum, 2))
    kind, hind = saledus(L, m)
    print ("massiindeks", round(kind), "oled", hind)

isik()
```

Tuletame meelde, et moodul **inimene.py**, mille funktsioonid on toodud ülalpool, peab asuma samas kaustas, kus on programm. Importimiseks võib kasutada käsku **import** või **from**. Siin kasutatakse käsku **from**, mille puhul pole vaja viitamist moodulile.

Programm loeb algandmed:
isikukood (**ik**), pikkus (**L**), mass (**m**), sugu (**s**)

Ja teeb mõned pöördumised mooduli **inimene** funktsioonide poole ning kuvab tulemused.

Kasutajamoodul funktsioon.py

Moodul sisaldab funktsioone, mis võimaldavad leida kasutaja poolt antud ühemuutuja funktsiooni F(x) mitmesuguseid omadusi (maksimaalne ja minimaalne väärtus ning nende asukohad, integraal ja pindala, nullkohad) etteantaval lõigul ja joonistada funktsiooni graafiku.

```
def F_max(F, a, b, n):
    ''' Funktsiooni F maks
     ja selle asukoht vx '''
    h = (b - a) / n
    maxi = F(a); vx = a
    for i in range(n + 1):
        x = a + i * h
        y = F(x)
        if  y > maxi:
            maxi = y
            vx = x
    return maxi, vx
```

```
def F_min(F, a, b, n):
    ''' Funktsiooni F min
     ja selle asukoht vx '''
    h = (b - a) / n
    mini = F(a); vx = a
    for i in range(n + 1):
        x = a + i * h
        y = F(x)
        if y < mini:
            mini = y
            vx = x
    return mini, vx
```

Parameeter **F** on nii selles kui ka järgmistes protseduurides antud punktis funktsiooni väärtuse leidmise protseduuri nimi.

Protseduurid on peaaegu identsed, ainuke erinevus on **if**-lause võrdlusmärgis.

Mõlemad funktsioonid tagastavad kaks väärtust: **maxi** või **mini** ja **vx** (vastav x).

Väärtusi **maxi** ja **mini** kasutatakse koordinaatsüsteemi määramisel graafikaakna jaoks.

```
def integraal(F, a, b, n):
    ''' Määratud integraal
        trapetsivalemiga'''
    h = (b - a) / n
    if h <= 0: return
    S = (F(a) + F(b)) / 2
    for i in range(1, n):
        S = S + F(a + i * h)
    return h * S
```

```
def pindala(F, a, b, n):
    ''' Pindala
        trapetsivalemiga'''
    h = (b - a) / n
    if h <= 0: return
    S = (abs(F(a))+abs(F(b))) / 2
    for i in range(1, n):
        S = S + abs(F(a + i * h))
    return h * S
```

Näites on tegemist sarnaste funktsioonidega, milles kasutatakse trapetsivalemit:

$S = h(y_0/2 + y_1 + y_2 + \ldots + y_{n-1} + y_n/2)$

Määratud integraal kujutab endast pindala – arvestatakse funktsiooni väärtuste märki.

Pindala leidmisel kasutatakse absoluutväärtusi.

```
def nullid(F, x0, xn, n,
           eps = 0.00001) :
    ''' Nullkohad lõigul [x0, xn],
        poolitusmeetodiga'''
    h = (xn – x0) / n
    y1 = F(x0)
    if y1 == 0 : print (x0)
    for i in range(1, n+1) :
        x = x0 + i * h
        y2 = F(x)
        if  y2 == 0 : print (x)
        if y1 * y2 < 0 :
            xk=F_pool(F, x-h, x, eps)
            print (round(xk, 3))
        y1 = y2
```

```
def F_pool(F, a, b, \
           eps = 0.00001) :
    ''' Nullkoht lõigul [a, b],
        poolitusmeetod '''
    y1 = F(a)
    while b - a > eps :
        c = (a + b) / 2
        y2 = F(c)
        if y2 == 0: return c
        if y1 * y2 > 0 :
            a = c
        else :
            b = c
    return c
```

Funktsioon **F_pool**() leiab (täpsustab) nullkoha lõigul [a; b] etteantava täpsusega **eps**. Eeldatakse, et nullkoht antud lõigul on olemas. Kasutatakse poolitusmeetodit.

Protseduur nullid() leiab kõik nullkohad lõigul [x0, xn]. Järjest arvutatakse ja võrreldakse funktsiooni naaberväärtusi. Kui need on erineva märgiga (y1*y2 < 0), on vahemikus nullkoht. Selle väärtuse täpsustamiseks kasutatakse funktsiooni **F_pool**.

```
def fungraaf(F, x0, xn, n, w = 2):
    ''' funktsiooni F graafik lõigul [x0; xn] '''
    mini, vx =  F_min (F, x0, xn, n)
    maxi, vx =  F_max(F, x0, xn, n)
    setworldcoordinates(x0, mini, xn, maxi)
    penup(); setpos (x0, 0)   # X - telg
    pendown(); setpos(xn,0)
    penup(); setpos (0, mini)  # Y - telg
    pendown(); setpos(0,maxi)
    x = x0   # jaotised X-teljel
    while x < xn: # jaotised X-teljel
        penup();   setpos(x,-0.2)
        pendown(); setpos(x,0.2)
        x = x + 1
    # graafik
    h = (xn - x0)/n; pencolor('blue')
    penup(); setpos(x0,F(x0))
    pendown(); width(w)
    for k in range(n + 1):
        x = x0 + k * h
        y = F(x)
        setpos(x, y)
```

```
def funtab(F, x0, xn, n):
    ''' funktsiooni F tabel
        lõigul [a; b] '''
    h = (xn - x0) / n
    for k in range(n + 1):
        x = x0 + k * h
        y = F(x)
        print (round (x, 2), "\t", round(y, 4))
```

Protseduur **funtab** arvutab ja kuvab Shelli aknas antud lõigul uuritava funktsiooni väärtused.

Protseduur **fungraaf** joonistab funktsiooni graafiku antud lõigule. Oluline roll on siin mooduli **turtle** meetodil:

setworldcoordinates(x0, mini, xn, maxi),

mis määrab kasutaja koordinaatsüsteemi, näidates akna alumise vasakpoolse nurga (x0, mini) ja ülemise parempoolse nurga (xn, maxi) väärtused.

Protseduur joonistab koordinaatteljed ja jaotised X-teljele.

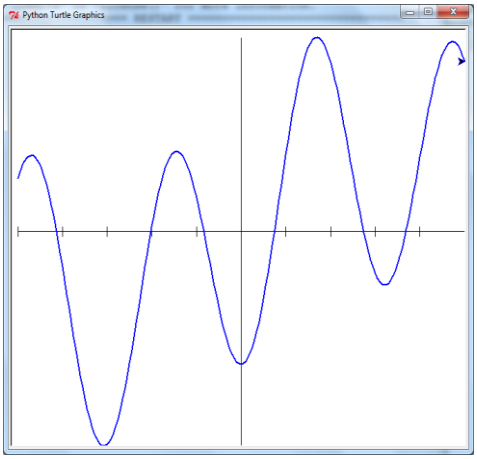
Graafiku enda joonistamine toimub **for**-lause juhtimisel, kus muutuja x väärtusi muudetakse järjest ja arvutatakse muutuja y väärtused ning joonistatakse vastavad lõigud.

Näide funktsiooni kasutamise kohta, vt „Kasutajamoodul funktsioon.py“, lk 185.

```
from math import *
from turtle import *
from funktsioon import *

def F1(x):
  return 3*sin(x /2)+5 * cos(2 * x+3)

a = -5; b = 5; n1 = 10; n2 = 200
funtab (F1, a, b, n1)
mini, x1 = F_min(F1, a, b,n2)
maxi, x2 = F_max(F1, a, b,n2)
print ("Y min:", round(mini, 3),
                round(x1, 2))
print ("Y max:", round(maxi, 3),
                 round(x2, 2))
integ = integraal(F1, a, b, n2)
pind = pindala(F1, a, b, n2)
print ("Integ:", round(integ,3))
print ("pind:", round(pind,3))
nullid(F1, a, b,n2)
fungraaf (F1, a, b, n2)
mainloop()
```

Kasutaja võib defineerida Pythoni funktsiooni matemaatilise ühemuutuja funktsiooni väärtuse leidmiseks antud punktis x. Selle funktsooni nimi (siin **F1**) antakse argumendina pöördumisel mooduli funktsioonide poole.

Tulemuseks on järgmine graafik:




```
-5.0      1.9741
-4.0      -1.3096
-
3.0       -1.5632
4.0       2.75
5.0       6.3327
Y min: -7.998 -3.07
Y max: 7.22 1.69
Integ: 2.691
pind: 35.778
-4.134
-2.019
-0.838
0.746
2.741
3.69
```

Märkandmed ja tegevused nendega

Pythoni programmides kasutatakse kolme liiki märkandmeid:

- arvud: täisarvud – klass int, reaalarvud ehk ujupunktarvud – klass float,
- tekstid (stringid ehk sõned)– klass str,
- tõeväärtused – klass bool.

Väärtuse liigist sõltuvad ka võimalikud tegevused andmetega.

Avaldised ja funktsioonid

Avaldis määrab ära tehted (operatsioonid) ja nende sooritamise järjekorra.

Avaldiste struktuur ja liigid

Üldjuhul koosneb avaldis:

- operandidest,
- operaatoritest ehk tehtesümbolitest,
- ümarsulgudest.

Erijuhul võib avaldis koosneda ainult ühest operandist. Operandideks võivad olla:

- konstandid,
- lihtmuutujad,
- struktuurmuutujate elemendid,
- funktsiooniviidad ja meetodid.

Lihtmuutujad esitatakse nimede abil, struktuurmuutujate elementide esitus sõltub andmekogumi liigist.

Funktsiooniviit ja **meetod** esitatakse kujul:

[***objekt***.] ***nimi*** ([*argumendid*]),

kus ***nimi*** on Pythoni sisefunktsiooni või kasutajafunktsiooni nimi. Sisefunktsioonide nimed ei kuulu reserveeritud võtmesõnade hulka, kuid muuks otstarbeks neid kasutada ei ole mõistlik, vältimaks segaduse tekkimist. Funktsiooniviites esinev ***argument*** näitab funktsioonile edastatavat väärtust. Argumendid võivad olla esitatud avaldiste abil. Argumentide arv, tüüp ja esitusjärjekord sõltuvad konkreetsest funktsioonist.

Tehted ehk **operaatorid** jagunevad nelja rühma:

- **aritmeetikatehted**: ** , * , / , // , % , + , -
- **stringitehted** +, *
- **võrdlustehted**: == , != , < , <= , > , >=
- **loogikatehted**: not, and, or

Üldjuhul võib ühes ja samas avaldises esineda tehteid kõikidest liikidest. Avaldise väärtuse leidmisel arvestatakse tehete prioriteete liikide vahel ning aritmeetika- ja loogikatehete puhul ka liigi sees. Tehete liigid on siin esitatud prioriteetide kahanemise järjekorras. Aritmeetika- ja loogikatehete prioriteedid kahanevad vasakult paremale. Avaldises **a + b > c** and **a + c > b** and **b + c > a** esinevad aritmeetika-, võrdlus- ja loogikatehted.

Väärtuse leidmisel täidetakse kõigepealt aritmeetika-, siis võrdlus- ning lõpuks loogikatehted.

Tehete järjekorra muutmiseks võib kasutada ümarsulge, kusjuures sulgudes oleva avaldise väärtus leitakse kõigepealt (eraldi). Ümarsulgudes esitatakse ka funktsiooniviidete ja meetodite argumendid.

Sõltuvalt andmete liigist ning kasutatavatest tehetest ja leitava väärtuse liigist võib avaldised jagada järgmistesse rühmadesse:

- arvavaldised
- stringavaldised
- loogikaavaldised

Arvavaldised ja matemaatikafunktsioonid

Arvavaldiste operandide väärtusteks on arvud ning neis kasutatakse aritmeetikatehteid ning funktsioone, mis tagastavad arvväärtusi.

Aritmeetikatehted ja nende prioriteedid on järgmised.

Prioriteet	**Tehte sümbolid**	**Selgitus**
1	**	astendamine a**n
2	-	unaarne miinus -a * -b + c
3	* **ja** /	korrutamine ja jagamine a * b, a / b
3	//	täisarvuline jagamine a // b, 13 // 5 = 2
3	%	jagatise jääk a % b, 13 % 5 = 3
4	+ **ja** -	liitmine ja lahutamine a + b, a - b

Võrdse prioriteediga tehteid täidetakse järjest vasakult paremale. Erandiks on astendamine, kus tehteid täidetakse paremalt vasakule. Tehete järjekorda saab reguleerida ümarsulgudega.

Tehete prioriteetide rakendamise näiteid:
-3**2 * 5 + 18 / 2 * 3 = -9 * 5 + 9 * 3 = -45 + 27 = -18,

(-3)**2 * 5 + 18 / (2 * 3) = 9 * 5 + 18 / 6= 48

4**2**3 = 48 = 65 536 64**1/3 = 64^1/3 = 64/3 64**(1/3) = 4

Matemaatikafunktsioonid ja konstandid

Matemaatikafunktsioonid kuuluvad moodulisse **math** ja esitatakse programmis kujul **math**.nimi(a) või nimi(a), olenevalt mooduli importimise viisist.

sqrt(a)	ruutjuur sqrt(b**2 - 4*a*c) = (b**2 - 4*a*c)**(1/2)
log(a)	naturaallogaritm (ln a) log(a) / log(10) = $\log_{10}$a.
log10(a)	kümnendlogaritm
exp(a)	e^a (e = 2,71828…) (exp(-x) + exp(2 * x)) / 2 = (e^{-x} + e^{2x}) / 2
abs(a)	absoluutväärtus abs ((a - x) / (a + x))
sin(a), cos(a), tan(a)	sin(x)+cos(2*x)+tan(x**2)-cos(2*x)**2 **NB!** argument on radiaanides
asin(a), acos(a), atan(a)	arkusfunktsioonid. Radiaanides (-π/2<x<π/2). atan(a/sqrt(1-a**2))=asin a , 4*atan(1) = π
pi, e	pi = π, e – naturaallogaritmi alus: 2.718281828459045

Teisendusfunktsioonid

str(a)	teisendus stringvormingusse
int(a)	teisendus täisarvuks
float(a)	teisendus reaalarvuks
round(a, n)	ümardamine: round (13.74615, 2) = 13.75
trunc(x)	täisosa
ceil(x)	vähim

Stringid (sõned) ja stringavaldised

Stringavaldiste operandide väärtusteks on stringid (sõned), milles võib kasutada stringitehteid ja stringifunktsioone. Stringitehet **+** nimetataks ka **sidurdamiseks**. See võimaldab ühendada stringe ja ka arve. Viimased peab teisendama funktsiooniga **str** tekstivormingusse.

Näiteid:
"Peeter" + " " + "Kask" annab Peeter Kask, str(35.7) + ” ” + str(2.5) annab 35.7 2.5

Kui S=5378.75, x1=2.538, x2=-1.34, siis

"Summa= " + str(S) annab Summa= 5378.75,

"x1= " + str(x1) + " x2= " + str(x2) => x1= 2.538 x2= -1.34

Stringandmete jaoks on rida funktsioone ja meetodeid. Stringe (sõnesid) käsitletakse järjestatud märkide jadana. Olgu näiteks antud stringid:

s = *'See on string'* ja isik = *"Juku Naaskel"*

Neid võib kujutada järgmiste märkide jadadena:

s

S	e	e		o	n		s	t	r	i	n	g
0	1	2	3	4	5	6	7	8	9	10	11	12

isik

J	u	k	u		N	a	a	s	k	e	l
0	1	2	3	4	5	6	7	8	9	10	11

Märkide nummerdamine algab alati nullist. Funktsioon **len**(string) annab märkide arvu stringis

len(*'Juku'*) = > 4; len(s) => 13; len(isik) => 12

Viitamiseks üksikule märgile kasutatakse nime koos indeksiga: nimi [indeks]

s[0] => S, s[2] => e, s[5] => n, s[12] => g

isik[0] => J, isik[2] => k, isik[5] => N, isik[11] => l

Saab viidata stringi osadele, kasutada stringi **lõikeid** (*slices*). Lõige esitatakse kujul:

nimi [m : n]

siin **m** on lõigu algus (kaasarvatud), **n** – lõigu lõpp (väljaarvatud) – tegelikult on lõpp (n - 1).

s[0 : 3] => *'See'*, s[4 : 6] => *'on'*,

s[7 :] => *'string'*, s[: 7] = > *'See on '*

Stringe saab omavahel liita (+) ja korrutada (*) arvuga

'Tere, ' + isik[0 : 4] + *'!'* => *'Tere, Juku!'*

isik[: 4] + *' on '* + 3 * *'väga '* + *'tubli!'* => *'Juku on väga väga väga tubli!'*

NB! Stringi väärtust saab muuta ainult tervikuna!

isik = *'Kalle'*; isik = s - on võimalikud

isik[0] = *'k'* , isik[5 :] = *'Kärmas'* - annavad vea!

Võib luua uue stringi taoliselt:

isik2=*“”*; isik2 = isik2 + isik[: 4] + *' Kärmas'* => *'Juku Kärmas'*

Operatsioon (tehe) **str1 in str2** annab tulemuseks **True**, kui str1 sisaldub str2-s.

Näiteks lause **if** 'a' **in** tekst: k = k + 1 suurendab muutuja k väärtust kui täht 'a' on antud tekstis olemas.

Mõned stringide meetodid

Esitatakse kujul: **string**.meetod()

count (*a_str*)	loendab alamstringi esinemise arvu stringis
find (*a_str*)	leiab alamstringi (*a_str*) alguse (indeksi) stringis
lower (), upper ()	teisendab teksti koopia väike- või suurtähtedeks
lstrip(), rstrip(), strip()	eemaldab juhtsümbolid vasakult, paremalt või kõik
split()	jagab lause sõnadeks, moodustades loendi

"Loendab alamstringi esinemise arvu stringis".**count**(*"string"*) => 2

Tühiku asukoht stringis isik: isik**.find**(" ") => 4

Eesnime ja perenime eraldamine stringist isik:

eesnimi = isik[0 **:** isik**.find**(" ")]; perenimi = isik[isik**.find**(" ") + 1 **:**]

eesnimi – lõige algusest (0) kuni tühikuni, perenimi – lõige peale tühikut stringi lõpuni.

*"Juku"***.upper**() => *'JUKU'*

" *tekst* "**.strip**() = 'tekst'

>>> *"Jagab lause sõnadeks, moodustades loendi"*.**split**()

[*'Jagab'*, *'lause'*, *'sõnadeks,'*, *'moodustades'*, *'loendi'*]

Erisümbolid

Teksti sees võivad olla mitteprinditavad juhtsümbolid, mis avaldavad teatud mõju teksti väljanägemisele. Need esitatakse teksti sees sümboli **** (langkriips) järel. Taolisi sümboleid on mitmeid, siin nimetame ainult paari:

\n - uus rida printimisel või kirje lõpp failis

\t - tabuleerimine printimisel

```
print ("Igaüks\neraldi\nreale");   # iga sõna prinditakse eraldi reale
print (x, "\t", y, "\t", z);          # jäävad suuremad vahed
```

Paar näidet. Funktsioon **mitu_tais** teeb kindlaks täishäälikute arvu antud tekstis, funktsioon **ilma_tais** tagastab antud teksti (stringi) ilma täishäälikuteta.

```
def mitu_tais(txt):
    ''' täishäälikute arv '''
    tais = 'AEIOUÕÄÖÜ'
    mitu = 0
    for t in txt:
        if t.upper() in tais:
            mitu += 1
    return mitu
```

```
def ilma_tais(txt):
    ''' ilma täishäälikuteta '''
    tais = 'AEIOUÕÄÖÜ'
    uus = ''
    for t in txt :
        if t.upper() not in tais:
            uus += t
    return uus
```

Mõlemas funktsioonis on moodustatud täishäälikuid sisaldav stringkonstant **tais**, mis sisaldab ainult suurtähti. Antud tekstis aga võivad olla nii suur- kui ka väiketähed, mida arvestatakse meetodi **upper** kasutamises.

```
tekst = input("Anna tekst ")
n = mitu_tais(tekst)
print ("Täishäälikuid oli:", n)
print (ilma_tais(tekst))
```

Funktsioonides tasub panna tähele **for**- ja **if**-lauseid. Kuna string **txt** kujutab endast jada, siis kontrollitakse järjest selle kõikide märkide sisalduvust stringis **tais**, kasutades **if**-lauses operaatorit **in**.

Vt ka näiteid moodulis „Inimene“, lk 184.

Võrdlused ja loogikaavaldised

Võrdlused on käsitletavad loogikaavaldiste erijuhtudena, nende kuju on järgmine:

avaldis1 **tehtesümbol** *avaldis2*

Tehtesümbolid (operaatorid) on järgmised: **==** , **!=** , **<** , **<=** , **>** , **>=**

Avaldised *avaldis1* ja *avaldis2* on arv- või stringavaldised. Ühes võrdluses esinevad avaldised peavad kuuluma samasse liiki. Võrdluse tulemiks on alati tõeväärtus **True** (tõene) või **False** (väär). Võrdluste näiteid

x <= 0, b * b - 4*a*c < 0, x * x + y * y > r * r, vastus.upper() == "EI"

NB! Stringide võrdlemisel eristatakse suur- ja väiketähti!

Loogikaavaldise üldkuju on järgmine:

avaldis **LTS** *avaldis* {**LTS** *avaldis*}

Siin on *avaldis* võrdlus või loogikaavaldis ja **LTS** loogikatehte operaator. Peamised loogikaoperaatorid on **or**, **and** ja **not**. Nende tähendused on:

or – või: tehte a **or** b väärtus on tõene (**True**), kui vähemalt ühe operandi väärtus on tõene, vastupidisel juhul on tulem väär (**False**).

and – ja: tehte a **and** b tulem on tõene (**True**) ainult siis, kui mõlema operandi väärtused on tõesed, vastupidisel juhul on tehte tulem väär (**False**).

not – mitte: tehte **not** a tulem on tõene (**True**) siis, kui a väärtus on väär (**False**) ja väär (**False**) vastupidisel juhul.

Loogikaavaldiste näiteid:

x >= 2 **and** x <= 13, x < 2 **or** x > 13 , a + b > c **and** a + c > b **and** b + c > a

Omistamine ja omistamislause

Omistamine on üks fundamentaalsemaid tegevusi, mida arvuti saab programmi toimel täita. See seisneb mingi väärtuse salvestamises arvuti sisemälus etteantud väljas või pesas. Välja (pesa) eelmine väärtus, kui see oli, kaob.

Üheks peamiseks vahendiks väärtuse salvestamiseks (omistamiseks) on kõikides programmeerimiskeeltes omistamislause. Tüüpiliselt eelneb väärtuse salvestamisele selle leidmine (tuletamine) etteantud avaldise abil. Kusjuures avaldise väärtuse leidmisega kaasneb enamasti muutujate ja/või omaduste varem salvestatud väärtuste lugemine. Omistamine võib kaasneda ka mõnede muude lausete ja meetodite täitmisele.

Omistamise ja vastavate omistamislausete olemus on praktiliselt sama. Ühelt poolt kujutab muutujale eraldatav mäluväli (pesa) endast objekti, mille omadust *väärtus* omistamislause täitmisel muudetakse. Objekti igale omadusele eraldatakse omaduste vektoris mäluväli, kuhu omistamisel salvestatakse vastav väärtus.

Omistamislause põhivariant on järgmine:

muutuja = *avaldis*

Ühe lausega saab omistada väärtused ka mitmele muutujale:

muutuja [, *muutuja*]… = *avaldis* [, *avaldis*]…

Igale muutujale erinev väärtus: y, z, w = x, 2 * x + 3, sin(x) + cos(x)

muutuja [= *muutuja*]… = *avaldis*

Üks väärtus mitmele muutujale: PS = pn = NS = nn = 0

Muutuja esitatakse nime abil. Tuletame meelde, et avaldiste operandideks võivad olla konstandid, muutujad, objektide omadused, massiivi elemendid, funktsiooniviidad (funktsioonid) sisefunktsioonidele või kasutaja koostatud funktsioonidele. Avaldis võib koosneda ka ainult ühest operandist.

Omistamislausete näiteid

k = 0; x = 2.5; v = w = z = 13; nimi = *"A. Kask"*

Erijuht: muutujale omistatakse konstandi väärtus.

x = y ; t = v; ymax = y

Erijuht: muutujale omistatakse teise muutuja väärtus, st paremas pooles oleva muutuja väärtus kopeeritakse vasakus pooles oleva muutuja väärtuseks.

x = a + i * h ; y = 3 * math.sin(x) – math.sqrt(x^4 + 5)

Üldjuht: leitakse paremas pooles oleva avaldise väärtus ja tulemus omistatakse vasakus pooles olevale muutujale, st salvestatakse antud muutuja mäluväljas (pesas).

k = k + 1; S = S + y;: n = n – k; F = F * k

k += 1; S += y; n -= k; F *= k *# võib kasutada lühendatud variante*

Erijuht: sama muutuja esineb omistamislause vasakus ja paremas pooles. Tegemist on uue väärtuse leidmisega ja asendamisega eelmise (jooksva) väärtuse alusel. Näiteks lause k = k + 1 täitmisel loetakse k jooksev väärtus, liidetakse sellele 1 ja saadud tulemus võetakse k uueks väärtuseks, st k väärtust suurendatakse ühe võrra.

S, P, d = a * b, P = 2 * (a + b), math.sqrt(a**2 + b**2)

Lauses leitakse kolm väärtust ja omistatakse muutujatele S, P ja d.

Lause **a, b = b, a** vahetab muutujate **a** ja **b** väärtused.

Omistamislaused **2 = x** ja **x + 5 = y** on aga täiesti mõttetud ja lubamatud! Ei saa omistada väärtust konstandile või avaldisele ehk salvestada (!) konstandis või avaldises midagi. Omistamislause vasakus pooles võib olla ainult muutuja nimi või objekti omadus!

Andmete väljastamine ekraanile

Andmete väljastamisega (öeldakse ka kuvamisega, printimisega, trükkimisega) ja sisestamisega (öeldakse ka lugemisega) tuleb vähemal või suuremal määral kokku puutuda peaaegu iga rakenduse loomise juures. Programmid peavad tüüpiliselt väljastama tulemusi ja harilikult vajavad nad ka algandmeid ülesande lahendamiseks. Tegemist on vastusuunaliste, kuid teatud mõttes sarnaste tegevustega. Pythonis on rikkalik vahendite kogum andmete sisestamiseks ja väljastamiseks erineval kujul. Järgnevalt vaadeldakse vaid kõige lihtsamaid võimalusi.

Andmete väljastamiseks Shelli aknasse saab kasutada funktsiooni **print**:

print ([*argument* [, *argument*]…])

Lihtsaimal juhul võib funktsioonil (lausel) olla kuju: **print()** ja see väljastab tühja rea.

Üsna sageli kasutatakse funktsiooni teadete väljastamiseks.

Järgnevas näites on kasutusel mitu stringkonstanti:

print(*"Tere,"* , *'mina olen Python.'* , *"Mis on Sinu nimi?"*).

Siin on tegemist kolme argumendiga (stringiga). Kuvamisel lisab Python väärtuste vahele ühe tühiku. Sama pildi saame, kui kasutame ühte argumenti:

print (*“Tere, mina olen Python. Mis on Sinu nimi?“*).

Tavaliselt on argumentideks mitu väärtust ning konstandid, muutujad ja avaldised võivad esineda läbisegi:

print (*"laius="*, a, *"kõrgus="*, b, *"pindala="*, a * b)

On olemas vahendid vormindamiseks, kuid neil me ei peatu. Üheks praktiliseks probleemiks võib olla arvu üleliia suur murdosa pikkus, sellisel juhul võib kasutada ümardamist funktsiooni **round**() abil. Näiteks:

print (*"x="*, **round**(x, 2), *"y="*, **round**(y, 4))

Kasulikuks võimaluseks on spetsiaalne element end = " " argumentide loetelu lõpus; see blokeerib ülemineku uuele reale pärast antud rea väljastamist.

Allpool toodud laused kirjutavad tulemused ühele reale järgmiselt: 0 1 4 9 16 25 36 49 64 81

```
for k in range(10):
    print (k * k, end = " ")
```

Andmete sisestamine klaviatuurilt

Andmete sisestamiseks saab kasutada funktsiooni **input**(), mille põhivariant on järgmine:

muutuja = **input** ([teade])

teade (ei ole kohustuslik) võib olla esitatud tekstikonstandi või -avaldise abil, näiteks:

vastus = **input**(str(a) + " + " + str(b) + " = ")

Arvud peavad taolistes avaldistes olema teisendatud funktsiooniga **str**() tekstivormingusse.

Selle täitmisel kuvatakse **teade** (kui on) ja programm jääb ooteseisu. Peale seda, kui kasutaja sisestab väärtuse ja vajutab klahvile **Enter**, omistatakse väärtus muutujale ja töö jätkub. Kui kasutaja vajutab kohe Enter-klahvile omistatakse muutujale tühi väärtus.

Oluline on silmas pidada, et funktsiooniga **input** sisestatud väärtused on esitatud **tekstivormingus**.

Proovige käsuaknas (Shelli aknas) mõningaid sisestamisega seotud tegevusi.

```
>>> x = input("x = ")
x = tere
>>> print (x, type(x))
tere <class 'str'>
>>>
```

Vastuseks funktsiooniga **input** kuvatud teatele sisestatakse tekst **tere**. Funktsioon **print** kuvab x-i väärtuse ja selle tüübi:

```
tere <class 'str'>
```

Võiks öelda, et kõik on ootuspärane.

```
>>> x = input("x = ")
x = 13
>>> print (x, type(x))
13 <class 'str'>
>>>
```

Vastuseks funktsiooniga **input** kuvatud teatele sisestatakse arv **13**. Funktsioon **print** kuvab x-i väärtuse ja selle tüübi:

```
13 <class 'str'>
```

Nagu näeme, on x väärtuse tüübiks samuti 'str'.

```
>>> x = input("x = ")
x =
>>> print (x, type(x))
 <class 'str'>
>>>
```

Vastuseks funktsiooniga **input** kuvatud teatele ei sisestata midagi, vaid vajutatakse kohe klahvile Enter. Funktsioon **print** ei kuva x kohal midagi (tegemist on tühja väärtusega), tüübiks on ikka 'str':

```
<class 'str'>
```

```
>>> x = input("x = ")
x = 13
>>> x / 3
…
  x / 3
TypeError: unsupported operand
type(s) for /: 'str' and 'int'
>>>
```

Vastuseks funktsiooniga **input** kuvatud teatele sisestatakse arv **13**. Edasi proovitakse jagada x väärtust 3-ga. Vastuseks kuvab Python veateate, et operandid tüübiga 'str' ja 'int' ei sobi tehte / jaoks.

Lausega **input** sisestatud väärtus on seega alati tekstivormingus (tüüp **str**). Selliste arvväärtustega arvutusi teha ei saa ning sellepärast kasutataks sageli teisendusfunktsioone **int** ja **float**, näiteks:

n = **int** (**input**(*"Mitu küsimust?"*)); a = **float**(**input**(*"kaugus"*))

NB! Kui teisendamiseks kasutatakse funktsiooni **int**, kuid sisestatav väärtus on aga reaalarv (arvul on murdosa), siis tekib viga. Funktsiooni **float** korral täisarvu sisestamisel viga ei teki ning sellepärast on sageli parem kasutada teisendamiseks funktsiooni **float**.

Mõnikord on otstarbekas sisestamisel kasutada teisendamiseks funktsiooni

eval(**input**(*teade*)).

Funktsioon on stringina ette antud avaldise väärtuse leidmiseks. Sisestamisel teeb see kindlaks sisestava väärtuse tüübi ja teisendab selle vastavasse vormingusse. Võiks proovida näiteks järgmisi tegevusi.

```
>>> x = eval(input("x = "))
x = 13
>>> print (x, type(x))
13 <class 'int'>
>>>
```

Vastuseks teatele sisestatakse täisarv 13 ja see teisendatakse täisarvuks 'int'.

```
>>> x = eval(input("x = "))
x = 13.7
>>> print (x, type(x))
13.7 <class 'float'>
```

Vastuseks teatele sisestatakse reaalarv 13.7 ja see teisendatakse tüüpi 'float'.

```
>>> x = eval(input("x = "))
x = tere
Traceback (most recent call last):
…
  x = eval(input("x = "))
…
NameError: name 'tere' is not defined
>>>
```

Vastuseks teatele sisestatakse tekst tere.

Kuvatakse veateade, nimi tere on määratlemata. Süsteem käsitleb sisendit nimena, sest interpreteerib seda avaldisena.

```
>>> x = eval(input("x = "))
x = 'tere'
>>> print (x, type(x))
tere <class 'str'>
>>>
```

Sisestatav väärtus on paigutatud ülakomade vahele – *'tere'*. Formaalselt on tegemist stringkonstandiga, mille tüübiks on 'str'.

```
>>> x = eval(input("x = "))
x = 3 * 13 – 21          # sisestatakse avaldis,
>>> print (x, type(x))  # mis sisaldab
18 <class 'int'>         # konstante
>>> a = 13
>>> x = eval(input("x = "))
x = a      # sisestatakse muutuja nimi
>>> print (x, type(x))
13 <class 'int'>
>>> x = eval(input("x = "))
x = 3 * a - a / 2 + 100  # sisestatakse avaldis,
>>> print (x, type(x))   # mis sisaldab
132.5 <class 'float'>    # muutujaid ja konstante
```

Funktsioon **eval** võimaldab siin kasutajal sisestada avaldise väärtuse arvutamiseks.

Avaldistes võib kasutada muutujate, objektide ja protseduuride nimesid, mis on kättesaadavad antud protseduuris või skriptis.

Funktsiooni **eval** saab kasutada ka mitme väärtuse korraga sisestamiseks (eraldatakse üksteisest komadega). Lause esitatakse järgmisel kujul: *muutuja* [, *muutuja*]… = **eval**(**input**([*teade*]))

Graafikaandmed ja graafikavahendid Pythonis

Üldised põhimõtted

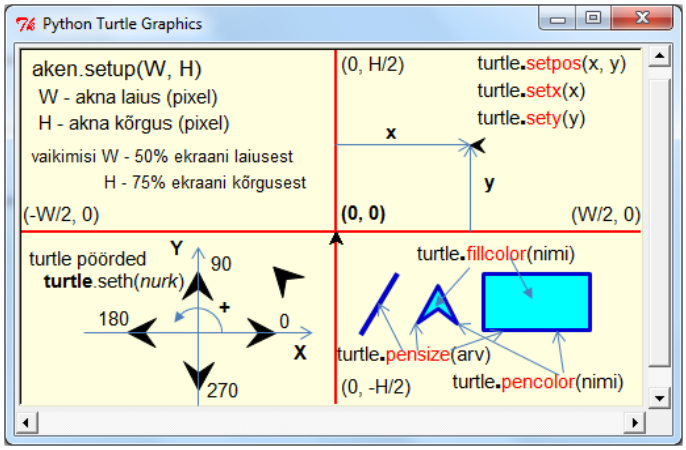
Graafikaandmete (pildid, joonised, skeemid, digrammid, graafikud jms) loomiseks ja töötlemiseks on suur hulk erinevaid tarkvarasid, alates lihtsatest joonistamisprogrammidest ja lõpetades pilditöötlemise, kujundamise joonestamis- ja projekteerimissüsteemidega. Graafikaandmed (graafikaobjektid) leiavad laialdast kasutamist ka rakendusprogrammide dokumentides ning veebidokumentides. Arvutigraafikas eristatakse kahte liiki graafikat: **rastergraafika** ja **vektorgraafika**. Rastergraafikas luuakse graafiline kujutis pikselitest – väikestest värvilistest punktidest. Veidi lihtsustades võib öelda, et vektorgraafikas kasutatakse kujutiste loomiseks sirgjoonte lõike ja neist moodustatavaid kujundeid – kolmnurgad, ristkülikud, ringid, … .

Enamikes programmeerimissüsteemides on olemas vahendid graafikaandmete kasutamiseks, kuid ka loomiseks. Pythoni standardmoodulite teeki kuulub moodul **turtle**, milles sisalduvate vahendite abil saab luua ja töödelda graafikaandmeid nn **kilpkonnagraafika** stiilis. Meetod ja nimetus võeti kasutusele aastaid tagasi programmeerimissüsteemis **Logo**, mis on päevakorral ka praegu, eriti programmeerimise õpetamisel. Kilpkonnagraafika on kasutusel mitmetes programmeerimissüsteemides nagu Scratch, BYOB, Basicu mitmed versioonid jms. Selle elemente on praktiliselt kõikides programmeerimissüsteemides, sest tegemist on arvutigraafika ja eriti vektorgraafika algelementidega. Joonistamisel lähtutakse tõsiasjast, et suurema osa kujunditest (joonised, graafikud, diagrammid, …) saab moodustada sirgjoone lõikudest. Vajaduse korral kasutatakse ka kujundite täitmist värviga.

Moodulis **turtle** sisalduvate vahendite abil saab luua joonistusi, teatud määral manipuleerida olemasolevate graafikaobjektidega (piltidega) ning luua ka animatsioone. Kuid eeskätt on see vahend siiski mõeldud joonistamiseks, võimaldades suhteliselt kiiresti ja lihtsalt omandada joonistamise ja joonestamise programmeerimise alused ja põhimõtted. Graafika programmeerimine aitab kaasa üldiste programmeerimisoskuste süvendamisele. Peab märkima, et ka robotite programmeerimisel leiavad kasutamist mitmed analoogilised põhimõtted ning isegi sarnased käsud.

Joonestamise operatsioone täidab spetsiaalne graafikaobjekt – ***turtle*** (kilpkonn) – Scratchi spraidi analoog. Kilpkonna (objekte) võib olla mitu ja neil võivad olla erinevad kujud. Vaikimisi eksisteerib üks kilpkonn, millel on nooletaoline kuju ➤. Kilpkonna rollis võib kasutada ka teisi graafilisi kujundeid, nt gif-vormingus pilte. Joonistused tekivad spetsiaalses graafikaaknas *Python Turtle Graphics*, mis luuakse automaatselt, kui programmis täidetakse esimene graafikakäsk (*turtle* meetod).

Käsuga **setup** saab määrata akna suuruse, mille mõõtmed antakse pikslites. Pikslil ei ole fikseeritud suurust – see sõltub ekraani suurusest ja eraldusvõimest. Koordinaatsüsteemi nullpunkt asub akna keskel. Oluline roll on kilpkonna suunal ehk pöördel (omadus **heading**). Mitmed liikumise ja pööramise meetodid: **forward**, **backward**, **left** jt arvestavad suuna hetkeväärtust. Alguses viiakse kilpkonn akna nullpunkti ja suunaks võetakse 0° (paremale).



Kilpkonnaga on seotud **pliiats**, mis võib liikumisel jätta jälje. Nähtamatu pliiats liigub koos kilpkonnaga ning tal on kaks võimalikku olekut: all (*pendown*) – jätab liikumisel joone, üleval (*penup*) – joont ei teki. Saab määrata pliiatsi suuruse (*pensize*) ehk joone paksuse ja pliiatsi (joone) värvuse (*pencolor*).

Kilpkonna (sisuliselt pliiatsi) asukoha ja suuna muutmiseks on rida käske (meetodeid), mis toimivad kahel erineval viisil:

- uus asukoht määratakse jooksva suuna ja ette antava distantsiga (d) – käsud forward(*d*), backward(*d*)
- uus asukoht määratakse sihtkoha koordinaatidega (x, y) – setpos(x, y) või goto(x, y), setx(x), sety(y)

Viit **turtle**-objekti meetodile ehk **kilpkonnagraafika käsk** esitatakse kujul: [***objekt*.**] ***meetod*** ([argumendid])

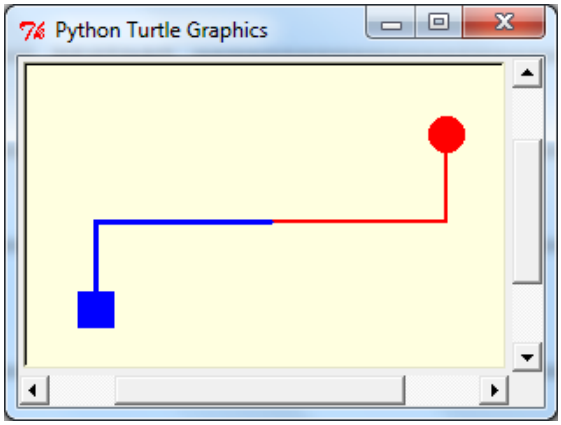
- objekt esitatakse nime turtle või muutuja nime abil, kui eelnevalt on objektiga seotud muutuja muutuja = turtle
- ***meetod*** määrab tegevuse
- ***argumendid*** näitavad tegevuse täitmiseks vajalikke andmeid, mis mõnedel meetoditel puuduvad

Käsu **from** kasutamisel importimise juures ei ole viita objektile vaja. Kuigi viida kasutamise korral on programm kompaktsem, eelistatakse sageli otsest viidet objektile, sidudes selle muutujaga. See variant on üldjuhul paindlikum, võimaldades näiteks paralleelselt kasutada mitut erinevat kilpkonna. Sageli on otstarbekas siduda muutujaga ka graafikaaken. Allpool olevas näites on algseaded programmi jaoks, kus kasutatakse kahte kilpkonna: kati ja mati. Ühtlasi demonstreerib see ka mõnede meetodite kasutamist.

```
import turtle
# akna seaded
aken = turtle.Screen()
aken.setup(300, 200)   # akna mõõtmed pikslites
aken.bgcolor('lightyellow')  # tausta värv
# kilpkonn kati
kati = turtle.Turtle()  # uus turtle objekt kati
kati.shape("circle")   # kati pildiks ring
kati.color("red")
kati.pensize(2)
kati.forward(100)
kati.left(90)
kati.forward(50)
# kilpkonn mati
mati = turtle.Turtle()   # uus turtle objekt mati
mati.shape("square")  # mati pildiks ruut
mati.color("blue")
mati.pensize(3)
mati.forward(-100)
mati.left(-90)
mati.forward(50)
aken.mainloop()  # aken ooteseisu
```

Screen() on meetod, mis loob klassi **Screen** kuuluva objekti. Omistamislausega seotakse loodud objekt muutujaga **aken**.

Samamoodi loob meetod **Turtle**() klassi **Turtle** kuuluva objekti ning omistamislause seob selle muutujaga, mille nimi on näidatud lause vasakus pooles.



Valik mooduli turtle meetodeid

Järgnevalt on esitatud valik **turtle** meetodeid (käske), kuid need ei hõlma sugugi kõiki võimalusi. Täiendavat informatsiooni selle teema kohta võib hankida Pythoni dokumentatsioonist.

Akna seaded	**Selgitused**
setup(W, H)	akna laius (W) ja kõrgus (H). aken.setup(600, 400) – 600*400 pix
bgcolor(värv)	tausta värv : *'black'*, *'red'*, *'white'*, *'green'* jmt
bgpic(nimi.gif)	tausta pilt – gif-fail, peab olema programmiga samas kaustas

setworldcoordinates(xv, ya, xp, yy) – kasutaja koordinaadid: xv – x vasak, ya – y alumine,
xp – x parem, yy – y ülemine

Kipkonna põhiomaduste küsimine ja muutmine

xcor(), ycor(); heading()	x- ja y-koordinaat; jooksev suund
showturtle(); hideturtle()	näita või peida, võib lühemalt: st(); ht()
shape(nimi)	Kilpkonna kujund: *'arrow'*, *'turtle'*, *'circle'*, *'square'*, *'triangle'*, *'classic'*
addshape(nimi.gif)	pildi lisamine gif-failist, mis peab olema samas kaustas programmiga.

Kilpkonna liikumine

forward(d); backward(d)	liigu edasi või tagasi **d** pikslit jooksvas suunas, arvestades **d** märki
setpos(x, y) | goto(x, y)	mine punkti (**x**, **y**), meetodid on samaväärsed
setx(x), sety(y)	määra **x**- või **y**-koordinaat;
home()	algseis: keskpunkti (0, 0), **heading**(90) ➤
right(nurk); left(nurk)	pööra paremale või vasakule **nurk** kraadi + pöörab vastupäeva
setheading(nurk)	võtta suunaks **nurk** kraadi
circle(r, …)	ring raadiusega **r turtle**'st vasakul (arvestades suunda) 

Pliiatsi omaduste määramine

pendown(); penup()	pliiats alla või üles
pensize(w) | size(w)	pliiatsi suurus (joone paksus), meetodid on samaväärsed
pencolor(v), fillcolor(v)	pliiatsi värv, täitevärv. v - värv: *'black'*, *'red'*, *'white'*, *'green'* jmt
pencolor(pv, tv)	pliiatsi (joone) värv (pv) ja täitevärv (tv)
begin_fill(), end_fill()	täitmise algus, täitmise lõpp
clear()	kustutab antud kilpkonna joonistused

Lugemine ja kirjutamine

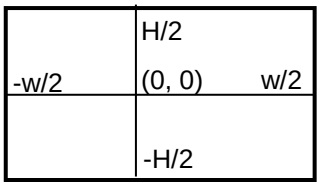
textinput(päis, teade)	teksti lugemine dialoogiboksist
numinput (päis, teade, …)	arvu lugemine dialoogiboksist
write (tekst, …font(…), …)	teksti kirjutamine graafikaaknasse, pliiats eelnevalt vajaliku kohta

Graafikamoodul kujud.py

Näitemoodulis kujud.py on funktsioonid elementaarkujundite (sirgjoone lõik, kolmnurk, ristkülik, …) joonistamiseks. Nende abil saab luua erinevaid pilte ja jooniseid. Lugeja võiks proovida toodud kollektsiooni täiendada.

```
def teljed (a, jv = "black", w = 1):
    ''' X ja Y teljed. a - aken
        jv-joone värv, w - paksus '''
    kk = turtle.Turtle()
    W = a.window_width();  H = a.window_height()
    kk.penup(); kk.setpos(-W/2, 0); kk.pendown();
    kk.pensize(w); kk.pencolor(jv)
    kk.setpos(W/2, 0);   kk.penup()
    kk.setpos(0, - H/2); kk.pendown()
    kk.setpos(0, H/2)
```

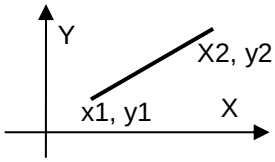
Graafikaaken **a** on määratletud pöörduvas protseduuris. Telgede joonestamiseks on eraldi kilpkonn (kk). Protseduur teeb kindlaks akna mõõtmed: W – laius, H – kõrgus.




```
def tekst (kk, tekst, x, y,
           kv = 'black', ks = 12, kt = 'normal'):
    ''' tekst graafikaaknasse. x, y – koht,
        kv – kirja värv, ks – suurus, kt – tüüp'''
    kk.penup(); kk.setpos (x, y); kk.pencolor(kv)
    kk.write(tekst, font=('Arial', ks, kt))
```

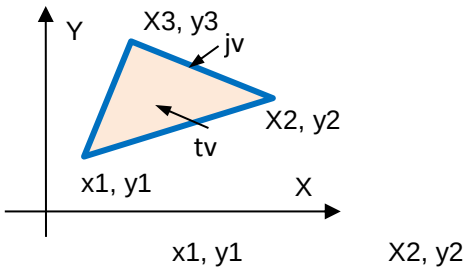
Teksti kirjutamisel graafikaaknasse saab määrata kirja värvuse, suuruse ja tüübi: *normal* | ***bold*** | *italic*.

```
def joon (kk, x1, y1, x2, y2, m=1, jv = 'black', w=1):
    ''' sirgjoone lõik (x1, y1) - (x2, y2), kk-kilpkonn
        m - mastaap,jv - joone värv,  w - paksus '''
    kk.penup(); kk.pencolor(jv); kk.pensize(w)
    kk.setpos(x1 * m, y1 * m); kk.pendown()
    kk.setpos(x2 * m, y2 * m)
```

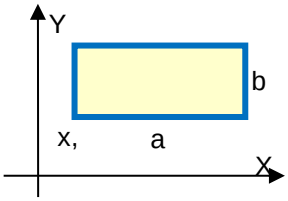


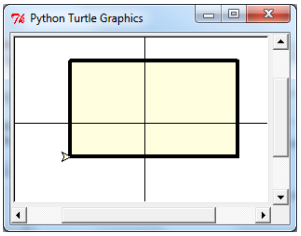

```
def kolmnurk (kk, x1, y1, x2, y2, x3, y3,
              m = 1, jv = 'black', tv ='', w = 1):
    ''' kolmnurk tippudega (x1,y1),(x2,y2),(x3,y3),
        kk-kilpkonn, m-mastaap, jv-joone värv,
        tv – täitevärv,  w- joone paksus '''
    kk.penup(); kk.color(jv, tv); kk.pensize(w)
    kk.begin_fill()
    joon(kk, x1, y1, x2, y2, m ,jv, w)
    joon(kk, x2, y2, x3, y3, m ,jv, w)
    joon(kk, x3, y3, x1, y1, m ,jv, w)
    kk.end_fill()
```

Kolmnurk moodustatakse kolmest sirglõigust, kasutades protseduuri **joon**.



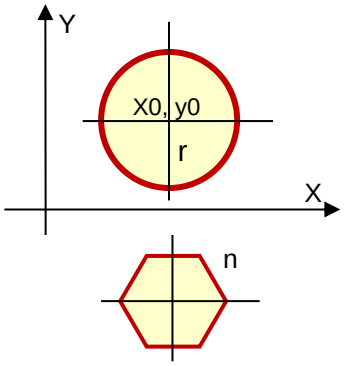
```
def rist (kk, x, y, a, b, m = 1,
        jv = 'black', tv = "", w = 1):
    ''' Ristkülik.  kk - kilpkonn,  m – mastaap,
      x, y – algus; a, b - laius ja kõrgus
      jv - joone värv, tv – täitevärv,  w - paksus '''
    x = m*x; y =  m*y; am = m * a; bm = m * b
    kk.penup(); kk.setpos(x, y)
    kk.width(w); kk.color(jv, tv)
    kk.pendown(); kk.begin_fill()
    kk.setpos(x + am, y)
    kk.setpos(x + am, y + bm)
    kk.setpos(x, y + bm); kk.setpos(x, y)
    kk.end_fill()
```





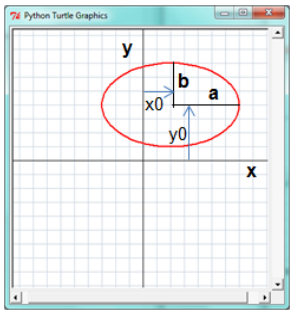

```
def  ring (kk,  x0, y0, r, m=1,
        jv = "black", tv = "", w=1, n=""):
    """ ring või hulknurk.
        kk - joonistaja, m - mastaap,
        x0, y0 - ringi keskpunkt,   r - raadius,
        jv - joone värv, tv – täitevärv, w - paksus
        n - hulknurga külgede arv, kui tühi - ring """
    x0 = m * x0; y0 = m * y0; rm = m * r
    kk.penup(); kk.goto(x0+rm, y0)
    kk.seth (90) ;  kk.pendown(); kk.pensize(w);
    kk.pencolor(jv); kk.fillcolor(tv)
    kk.begin_fill()
    if  n == "":    kk.circle(rm)
    else:           kk.circle(rm, steps = n)
    kk.end_fill()
```

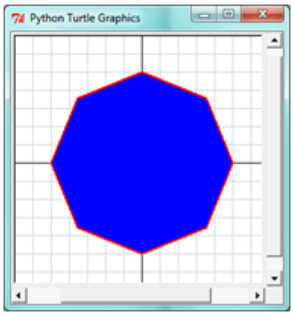
Protseduuri peamiseks elemendiks on mooduli **turtle** meetod **circle**, mis võimaldab joonistada ka korrapäraseid hulknurki, kui parameetrile **steps** anda mitte tühi väärtus.




```
def ellips (kk, x0, y0, a, b,
            jv="black", tv = "", w = 1, n = 64):
    """ elips, ring või hulknurk.
        kk - joonistaja, m - mastaap,
        x0, y0 - keskpunkt,   a, b - raadiused,
        jv - joone värv, tv – täitevärv, w - paksus
        n - hulknurga külgede arv """
    kk.color(jv, tv); h = 360/n
    kk.penup(); kk.setpos(x0+a, y0)
    kk.pendown(); kk.width(w);
    kk.begin_fill()
    for k in range(n):
        fi = k * h * pi / 180
        x = x0 + a * cos(fi)
        y = y0 + b * sin(fi)
        kk.setpos(x, y)
    kk.end_fill()
```

Protseduur võimaldab joonestada ellipsi, ringi või hulknurga. Selle aluseks on ellipsi võrrand parameetrilisel kujul. Joonistamisel joonestatakse järjest n kaart. Kui n on suhteliselt väike (<20) tekib hulknurk. Kujundi kuju saab määrata a ja b väärtusega. Kui a = b on tegemist ringi või võrdkülgse hulknurgaga.





```
def lill(kk, x0, y0, a, b,
         jv = "black", tv = "", w = 1):
  ''' x0, y0 – keskpunkt, a - kroonlehe pikkus
    b – kroonlehtede arv '''
  kk.color(jv, tv)
  kk.penup(); kk.setpos(x0,y0)
  kk.pendown(); kk.width(w)
  kk.begin_fill()
  for fi in range(181):
     fir = fi * pi / 180
     ro = a * sin (fir * b)
     x = x0 + ro *  sin(fir)
     y = y0 + ro * cos(fir)
     kk.setpos(x, y)
  kk.end_fill()
```

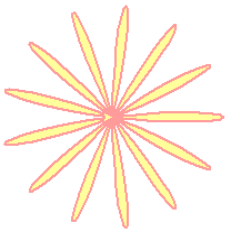
See protseduur joonistab lilletaolise kujundi.

Protseduuris kasutatakse võrrandit polaar-koordinaatides, joone koordinaadid arvutatakse järgmiste võrrandite abil:

$\rho = a*\sin(b*\varphi)$

$x = \rho*\cos(\varphi)$

$y = \rho*\sin(\varphi)$.



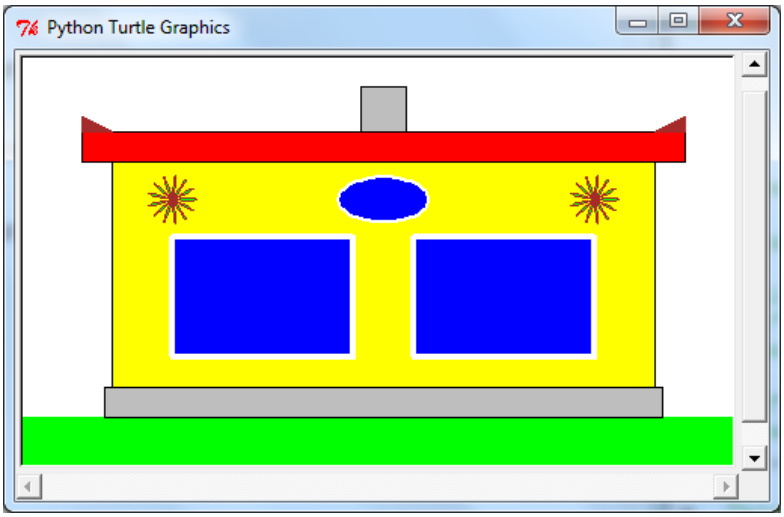
Muutuja ***a*** väärtuseks on kroonlehe pikkus ja ***b*** väärtuseks kroonlehtede arv.

lill (J, 0, 0, 100, 13, jv = "red", tv = "yellow", w=2)

Näide: Ehitame maja

Näide demonstreerib mooduli **kujundid** kasutamist. Moodulis olevatest „detailidest“ pannakse kokku (vist küll mõningate liialdustega) maja pilt.

```
from kujud import *
aken = turtle.Screen()
aken.setup(500, 300)
J = turtle.Turtle()
# maa
rist (J,-240, -140, 480, 40, jv = "green", tv = "green")
# vundment
rist (J,-185, -100, 370, 20, jv = "black", tv = "gray")
# sein ja kaunistused
rist (J,-180, -80, 360, 150, jv = "black", tv = "yellow")
lill (J, 140, 45, 16, 13, jv = "brown", tv = "green")
lill (J, -140, 45, 16, 13, jv = "brown", tv = "green")
# katus ja servad
rist (J, -200, 70, 400, 20, jv = "black", tv = "red")
kolmnurk(J, -200, 90, -180, 90, -200, 100, jv = 'brown', tv = 'brown')
kolmnurk(J, 180, 90, 200, 90, 200, 100, jv = 'brown', tv = 'brown')
# aknad
rist (J, -140, -60, 120, 80, jv = "white", tv =  "blue", w=4)
rist ( J, 20, -60, 120, 80, jv = "white", tv =  "blue", w=4)
ellips (J, 0, 45, 30, 15,jv = "white", tv =  "blue", w=2)
# korsten
rist (J,-15, 90, 30, 30, jv = "black", tv = "gray")
aken.exitonclick()  # lõpp, kui klõpsatakse akent
```



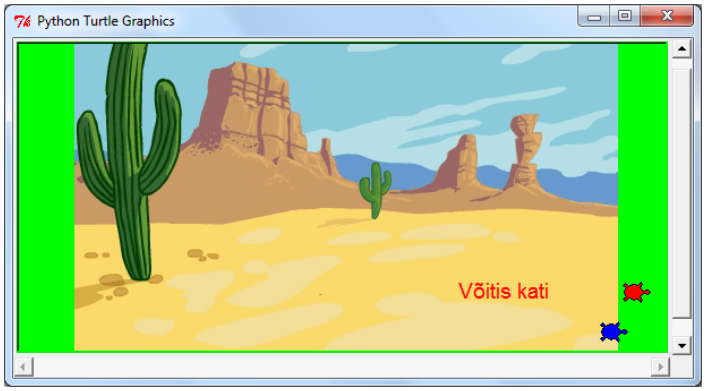
Animatsioonide näited

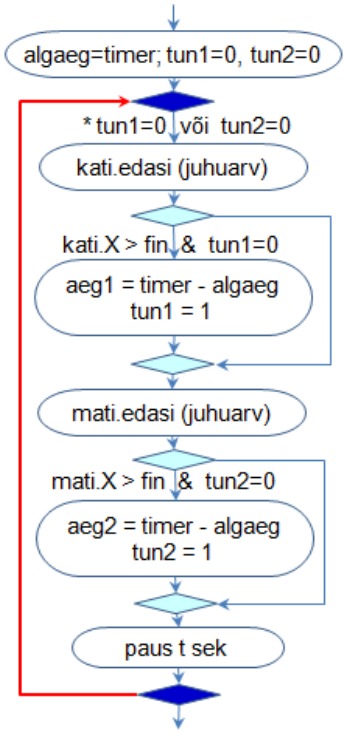
Näide: Suur võidujooks

Programm demonstreerib lihtsamaid võimalusi animatsiooni realiseerimiseks kilpkonnagraafika vahenditega, aga lisaks ka muud nagu taustapildi lisamine, objekti sidumine muutujaga, objektide kloonimine (paljundamine), mitme **turtle**-objekti, **while**-korduse ja ajaarvestuse kasutamine.

Kaks **turtle**-objekti – **kati** ja **mati –**, pannakse liikuma akna vasakust servast juhuslike sammudega etteantud kauguseni **fin**. Fikseeritakse kohalejõudmise ajad ja nende alusel tehakse kindlaks võitja.

```
import turtle, time
from random import *
aken = turtle.Screen(); W = 600; H = 300
aken.setup(W, H); aken.bgcolor("green")
aken.bgpic("aafrika.gif")  # taustapilt
kati = turtle.Turtle()  # luuakse kilpkonn kati
kati.shape("turtle")  # kati pildiks kilpkonn
kati.fillcolor("red"); kati.penup(); kati.ht()
kati.setpos(-W / 2 + 40, -H/2+70); kati.st()
mati = kati.clone()  # kloonitakse mati!!!
mati.fillcolor("blue"); mati.sety(-H/2 + 35)
time.sleep(1)  # paus 1 sek
# algab suur võidujooks
fin = W/2 - 90 # finishjoone x-koordinaat
algaeg = time.clock()  # aja algväärtus
h = 20; tun1 = tun2 = 0
while tun1 == 0 or tun2 == 0: # kordus
   kati.forward(randint(h/2, h))  # samm
   if kati.xcor() > fin and tun1 == 0 :
      aeg1 = time.clock() -  algaeg; tun1 = 1
      # fikseeritakse kati aeg
   mati.forward(randint(h/2, h))  # samm
   if mati.xcor() > fin and tun2 == 0:
      aeg2 = time.clock() -  algaeg; tun2 = 1
      # fikseeritakse mati aeg
   time.sleep(0.01)   # paus 0.01 sek
# kes võtis?
if aeg1 < aeg2:  v = "kati";  yt = kati.ycor()
else:            v = "mati";  yt = mati.ycor()
# teksti kirjutamine graafikaaknasse
kohtunik = turtle.Turtle(); kohtunik.ht()
kohtunik.penup()
kohtunik.goto(100, yt - 10)
kohtunik.pencolor("red")
kohtunik.write("Võitis " + str(v))
print(aeg1, aeg2)
aken.exitonclick()
```





Kasutatakse kolme **turtle**-objekti, mis on seotud järgmiste muutujatega: **kati**, **mati** ja **kohtunik.**

Lausega: kati = turtle.Turtle(), luuakse uus **turtle**-objekt ja seotakse see muutujaga **kati**.

Käsuga kati.shape("turtle") määratakse, et **kati** graafiliseks kujutiseks on kilpkonnataoline pilt.

Pildi värvuseks võetakse punane ja **kati** viiakse ekraani vasakusse serva.

Käsuga **clone**() kloonitakse **katist mati** – kõik **kati** omadused lähevad **matile**. Edasi muudetakse **mati** värvi ja y-koordinaati.

Programmi keskseks osaks on tingimuslik kordus, millega juhitakse objektide liikumist. Tegevusi korratakse seni, kuni tingimus on tõene. Oluline koht on siin muutujatel **tun1** ja **tun2**, mille väärtuseks võetakse alguses 0. Kui võistleja jõuab lõppu (x>fin), fikseeritakse tema aeg ja vastava tunnuse väärtuseks võetakse 1, millega tagatakse, et võistleja aeg fikseeritakse ainult üks kord, kui vastav tunnus võrdub nulliga. Kordust ja ka mõlemaid **if**-lauseid täidetakse kuni jooksu on lõpetanud mõlemad võistlejad.

Näide: Suur võidusõit

Näide on sarnane eelmisega, kuid imiteeritakse kahe auto võidusõitu.

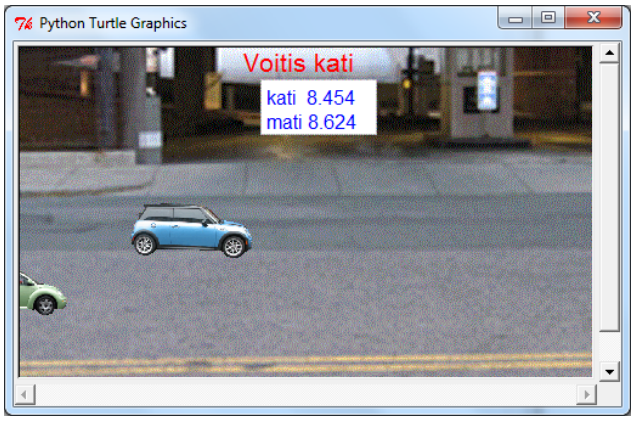
```
import turtle, time
from random import *
aken = turtle.Screen(); W = 1000; H = 600
aken.setup(W, H); dist = W
aken.bgpic("rada.gif")          # tausta pilt
aken.addshape("car_1.gif")  # lisa auto 1
aken.addshape("car_2.gif")  # lisa auto 2
kati = turtle.Turtle()        # luua objekt kati
kati.shape("car_1.gif")   # auto 1 kati pildiks
kati.penup(); kati.hideturtle(); kati.setx(-W/2)
mati= kati.clone()           # kloonida mati
mati.shape("car_2.gif")  # auto 2 mati pildiks
mati.sety(-50)
kati.st(); mati.st()    # peida kati ja mati
kati.speed(0); mati.speed(0)
ringe = aken.numinput("Suursoit", "Mitu ringi?", 3)
dist = 1200; h = 30; r1 = r2 = tun1 = tun2 = 0
algaeg = time.clock()  # aja algväärtus
while tun1 == 0 or tun2 == 0:
  kati.forward(randint(h / 2, h))  # kati samm
  if kati.xcor()  > dist :          # kas ring täis?
    kati.setx(-W / 2); r1 +=1   # algus, lisa ring
    if r1 == ringe and tun1 == 0: # kas kõik?
      aeg1 = time.clock()- algaeg; tun1 = 1
  mati.forward(randint(h / 2, h))  # mati samm
  if mati.xcor() > dist:              # kas ring täis?
    mati.setx(-W / 2); r2 +=1   # algus, lisa ring
    if r2 == ringe and tun2 == 0: # kas kõik?
      aeg2 = time.clock()- algaeg; tun2 = 1
  time.sleep(0.001)  # paus 0.01 sek
if aeg1 < aeg2:     v = "kati"   # kes võitis?
else:                 v = "mati"
# tekst graafikaaknasse
kohtunik=turtle.Turtle(); kohtunik.ht()
kohtunik.pu(); kohtunik.setx(-60);
kohtunik.sety(125)
kohtunik.pd(); kohtunik.pencolor("red")
kohtunik.write(" Voitis " + str(v), font=("Arial", 16))
kohtunik.pencolor("blue")
kohtunik.pu(); kohtunik.goto(-35,100);
kohtunik.write("kati  " + str(round(aeg1,3)))
kohtunik.pu(); kohtunik.goto(-35,80);
kohtunik.write("mati " + str(round(aeg2,3)))
aken.mainloop()
```

Kilpkonna võib esitada suvalise gif-vormingus pildiga. Pildi fail peab olema samas kaustas, kus asub programm.

Käsuga **aken.addshape**(fail) lisatakse objekt aknasse ja käsuga **nimi.shape**(fail) seotakse see konkreetse kilpkonnaga.

Erinevalt eelmisest näitest, sõidavad siin autod etteantud arvu ringe: iga kord, kui auto jõuab kaugusele **dist,** viiakse see akna vasakusse serva.

Allpool on toodud sõidu juhtimise algoritm pseudo-koodis.




```
loe ringe
dist = 1200; h = 30; r1 = r2 = tun1 = tun2 = 0
algaeg = timer
kordus seni kui tun1 = 0 või tun2 = 0
  kati.edasi(juhuarv)
  kui kati.X > dist siis
    kati.X = algus; r1 = r1 +1
    kui r1 = ringe siis
      aeg1 = timer - algaeg; tun1=1
  lõpp kui
  mati.edasi(juhuarv)
  kui mati.X > dist siis
    mati.X = algus; r2 = r2 +1
    kui r2= ringe siis
      aeg2 = timer - algaeg; tun2=1
  lõpp kui
  paus pp
lõpp kordus
```

Juhtimine

Juhtimiseks kasutatakse valikuid ja korduseid.

Valikud ja valikulaused

Valikulaused võimaldavad määrata tegevuste (lausete) valikulist täitmist sõltuvalt etteantud tingimustest ja kriteeriumitest. Need kujutavad endast liit- ehk struktuurlauseid, mis võivad sisaldada teisi liht- ja/või liitlauseid.

Lausete struktuur ja täitmise põhimõtted

if-lause üldkuju ja täitmise põhimõte on järgmised:

if *tingimus*:

if-laused

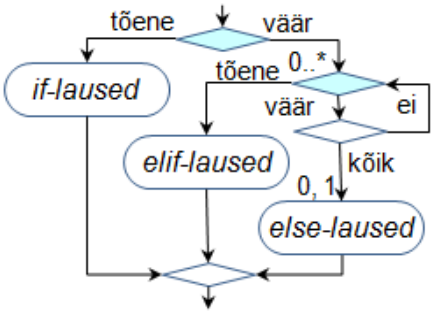
[**elIf** *tingimus*:

elif-laused]

…

[**else** :

else-laused]



Lause koosneb ühest **if**-osast ning võib sisaldada suvalise arvu **elif** (else if) osalauseid ja ühe else-osalause.

Tingimused esitatakse võrdluste või loogikaavaldiste abil. **Laused** võivad olla suvalised liht- ja liitlaused, sh ka **If**-laused. Struktuuri esitamiseks peab kasutama taandeid!

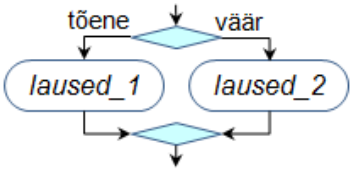
Lause täitmisel kontrollitakse kõigepealt tingimust **if**-lauses, kui see on tõene, täidetakse *if_laused*, kõik ülejäänu jääb vahele. Vastupidisel juhul kontrollitakse järjest tingimusi **elif**-osalausetes (kui neid on olemas) leidnud esimene tõese, täidetakse järgnevad laused ning kõik ülejäänu jääb vahele. Kui ükski tingimus ei ole tõene, täidetakse *else_laused* (kui need on olemas).

Ülaltoodud **if**-lause üldist varianti nimetatakse sageli ka mitmeseks valikuks – mitmest võimalikust tegevuste rühmast valitakse tingimuste alusel välja üks.

Lause üldkujust tulenevad ka kaks sageli kasutatavat varianti, mis võimaldavad määratleda valiku kahest ja valiku ühest.

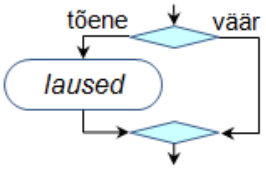
Kahendvalik

if *tingimus* **:**
laused_1
else :
laused_2



Valik ühest

if *tingimus* **:**
laused



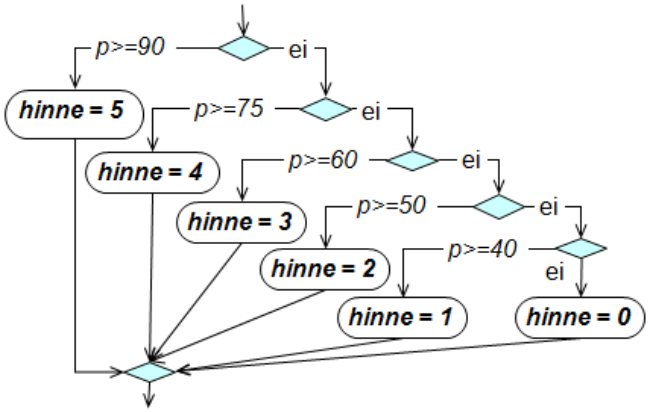
Näide: Punktid ja hinded

Mingi eksami, testi, … hinne määratakse saadud punktide alusel nagu siin esitatud.

Tuleb koostada funktsioon, mis tagastaks vastavalt saadud punktidele hinde.

p = 90–100, hinne = 5
p = 75– 89, hinne = 4
p = 60– 74, hinne = 3
p = 50– 59, hinne = 2
p = 40– 49, hinne = 1
p < 40 hinne = 0

Järgnevalt on toodud selleks kaks varianti: tavaline esitusviis ja kompaktne.

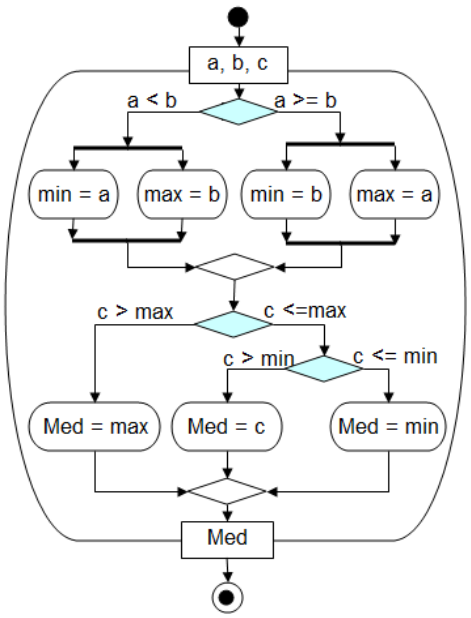



```
def hinne1 (p):
  if p >= 90:
    hinne = 5
  elif  p >= 75:
    hinne = 4
  elif p >= 60:
    hinne = 3
  elif  p >= 50:
    hinne = 2
  elif  p >= 40:
    hinne = 1
  else:
    hinne = 0
  return hinne
```

```
def hinne2 (p):
  if p >= 90:   return 5
  elif p >= 75: return 4
  elif p >= 60: return 3
  elif p >= 50: return 2
  elif p >= 40: return 1
  else:         return 0
```

Näide: Kolme arvu mediaan

Leida kolmest arvu mediaan (suuruse mõttes keskmine väärtus). Allpool on toodud ülesande lahendamise tegevusskeem (algoritm) ja Pythoni funktsioon **med**. Funktsioonil on kolm parameetrit: **a**, **b** ja **c**. Kasutusel on kaks muutujat: **min** ja **max**. Kõigepealt omistatakse muutujale **min** väiksem väärtus esimesest kahest arvust (**a** ja **b**) ja muutujale max suurem väärtus. Edaspidise jaoks ei ole tähtis, kumb arvudest **a** või **b** on suurem või väiksem.




```
def med (a, b, c):
  if a < b :
    min = a; max = b
  else:
    min = b; max = a
  if c > max:
    return max
  elif c > min:
    return c
  else:
    return min
```

```
def med(a, b, c):
  if a < b: min = a; max = b
  else:     min = b; max = a
  if c > max:   return max
  elif c > min: return c
  else:         return min
```

Edasi võrreldakse **min** ja **max** väärtusi kolmanda arvuga. Kui **c** on suurem kui **max**, on tegemist olukorraga, kus **min ≤ max ≤ c** ja mediaaniks on **max**. Kui aga **c ≤ max**, on kaks võimalust: **c > min**, siis **min ≤ c ≤ max** ja mediaaniks on **c**, kui ei ole, siis mediaaniks on **min.**

Kordused

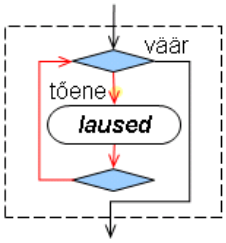
Pythonis on korduste kirjeldamiseks kaks lauset: **while**-lause, **for**-lause. Nendel lausetel on mitu varianti.

Eelkontrolliga while-lause

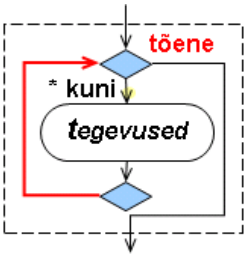
Põhivariant

while *tingimus* :
    laused
    [**break**]
    laused

Kordus töötab, kuni tingimus on tõene.



Scratchi eelkontrolliga kordus töötab seni, kuni tingimus saab tõeseks.



Sisemiste lausete kuuluvus **while**-lausesse määratakse taandega. Kõikide sisemiste lausete taane peab olema võrdne esimese lause omaga, kui need ei kuulu järgmise taseme lause sisse. Lause sees võivad olla **break**-laused, mis võivad katkestada kordamise ning on tavaliselt mingi tingimuslause sees.

NB! Sarnane eelkontrolliga tingimuslik lause töötab Scratchis mõnevõrra teisiti. Pythoni **while**-korduse täitmisel korratakse tegevusi siis, kui tingimus on tõene. Sellise reegliga kordusi (korratakse kuni tingimus on tõene) nimetatakse sageli **while**-tüüpi korduseks. Scratchi **korda kuni** ploki täitmine kestab aga seni kuni tingimus **saab tõeseks**. Sellist kordust (korratakse kuni tingimus saab tõeseks) nimetatakse **until**-tüüpi korduseks.

Näide 1: Start

```
from time import *
def start (k):
  print ("Valmis olla!")
  while k > 0:
    print (k, end = "  ")
    k = k - 1
    sleep(1)
  print ("Start!")
start (10)
```

Antud funktsioon kuvab pöördumise **start** (10) järel järgmised teated.

Valmis olla!
10 9 8 7 6 5 4 3 2 1 Start!

print-lause väljastab teate „Valmis olla!“. Edasi algab kordus.

while-lause igal täitmisel kontrollitakse tingimust **k** > 0. Kui see on tõene, kuvatakse **k** väärtus, lahutatakse **k** väärtusest 1, tehakse paus 1 sek ja kõik kordub kuni **k** saab võrdseks nulliga, st tingimus pole enam täidetud. Peale seda kuvatakse tekst „Start!“.

Näide 2: Arvu arvamine

```
import random
y = random.randint(1, 100)
x = int(input("Arva ära arv 1...100. : "))
k = 1
while x != y:
   k += 1
   if x < y:
      x = int(input('Vähe! Proovi veel: '))
   else:
      x = int(input('Palju! Proovi veel: '))
print ('Korras! Katseid oli: ', k)
```

Väljund võiks olla järgmine:

```
>>>
Arva ära arv 1...100. : 50
Palju! Proovi veel: 30
Palju! Proovi veel: 10
Vähe! Proovi veel: 20
Palju! Proovi veel: 15
Vähe! Proovi veel: 17
Korras! Katseid oli: 5
>>>
```

Muutujale ***y*** omistatakse juhuarv vahemikust 1–100. Kasutaja peab selle arvu ära arvama võimalikult väikese katsete arvuga. Programm toetab veidi kasutajat, teatades, kas pakutud arv ***x*** on väiksem või suurem arvuti poolt mõeldud arvust ***y***. Programm loendab ka katsete arvu, kasutades selleks muutujat ***k***.

Tingimus **while**-lauses on määratud võrdlusega x != y. See tähendab, et kordus kestab seni, kuni kasutaja ei ole sisestanud õiget vastust. Igal kordamisel kontrollitakse kõigepealt tingimust. Kui kasutaja poolt pakutud arv ***x*** ei võrdu arvuti arvuga ***y*** (tingimuse väärtus on **tõene)**, siis täidetakse sisemised käsud. Kui tingimuse väärtus on väär (x = y), katkestatakse kordamised ning täidetaks **while**-lausele järgnev lause, mis väljastab teate ja katsete arvu ***k***. Kuna tingimuse kontroll toimub kohe korduse alguses, peab x esimene väärtus olema sisestatud enne korduse algust.

Näide: Funktsiooni tabuleerimine

Rakendus võimaldab arvutada ja kuvada etteantud funktsiooni väärtused etteantaval lõigul, mis on jagatud **n** võrdseks osaks

```
from math import *
def Fy(x) :   return 3 * sin(2*x)-5 * cos(x/3)
def tabuleerimine (a, b, n) :
   h = (b - a) / n
   x = a
   while x <= b + h/2:
      print (x, round (Fy(x), 3))
      x = x + h
print ("Funktsiooni tabuleerimine")
x0 = float (input ("algus => "))
xn = float (input ("lõpp => "))
n = int (input ("jaotisi => "))
tabuleerimine (x0, xn, n)
```

```
Funktsiooni tabuleerimine
algus => 0
lõpp => 5
jaotisi => 10
0.0–5.0
0.5–2.406
1.0–1.997
1.5–3.965
2.0–6.2
2.5–6.239
3.0–3.54
3.5 0.005
4.0 1.792
4.5 0.883
5.0 -1.153
```

Peaprotseduur loeb algandmed (x0, y0 ja n) ning käivitab funktsiooni **tabuleerimine** (a, b, n), edastades sellele vastavad argumendid. Funktsioon leiab sammu **h** ning võtab **x** algväärtuseks **a**.

Järgnevas **while**-lauses arvutatakse järjest ja kuvatakse funktsiooni väärtus ning muudetakse **x** väärtust sammu **h** võrra. Peale igat muutust kontrollitakse tingimust **x <= b + h/2**. Teoreetiliselt on **x** lõppväärtuseks **b**, kuid **x** muutmisel reaalarvulise sammuga võib tekkida olukord, kus viimane (**b**-le vastav) väärtus võib olla veidi suurem **b**-st ning viimane **y** väärtus jääb leidmata. Selle tõttu on tingimuses võetud lõppväärtus poole sammu võrra suuremaks, tagamaks, et viimane väärtus leitaks alati.

```
print ("Funktsiooni tabuleerimine")
x0 = float (input ("algus => "))
xn = x0
while x0 == xn :
  xn = float(input ("lõpp => "))
  if x0 == xn: print ("x0 ja xn ei tohi olla võrdsed!")
n = 0
while n == 0:
 n = int (input ("jaotisi => "))
 if n == 0: print ("jaotiste arv ei tohi olla null!")

tabuleerimine(x0, xn, n)
```

Antud ülesande lahendamisel on oht sattuda lõputusse kordusse. Kui kasutaja sisestab x0 ja xn jaoks ühesugused väärtused, saadakse sammu h väärtuseks null, mille tõttu **while**-korduse täitmisel x väärtus ei muutu ja tööd ei lõpetata. Öeldakse, et programm jäi „lõputusse tsüklisse“, millest saab väljuda, kasutades klahvikombinatsiooni Ctrl+C.

Vältimaks sellise olukorra tekkimist, tuleks juba algandmete sisestamisel kontrollida selle variandi võimalikkust ning, kui kasutaja sisestab xn jaoks x0-ga võrdse väärtuse, paluda sisestada uus väärtus.

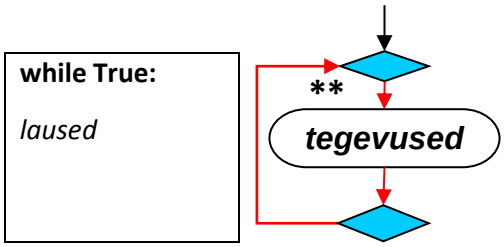
Tegemist on üsna tüüpilise ülesandega algandmete kontrollimiseks sisestamise käigus ja väärtuse võimaliku korduva sisestamisega.

Antud juhul eksisteerib veel teise võimaliku vea oht. Kui **n** väärtuseks sisestatakse 0, tekib sammu arvutamisel jagamine nulliga ja programmi töö katkestatakse. Kuna see ei ole soovitatav, kasutatakse selliste tegevuse täitmiseks enamasti **while**-lauset ja sellist skeemi nagu ülalpool esitatud.

Algselt omistatakse meelega muutujatele **xn** ja **x0** sellised väärtused, et tingimus oleks tõene ja toimuks sisestamine. Kui sisestatatakse muutujatele võrdsed väärtused, väljastatakse teade. Kuna tingimus on siis tõene, pakutakse võimalust uue väärtuse sisestamiseks. Seda tehakse seni, kuni sisestatakse muutujatele erinevad väärtused. Sarnaselt toimub kontroll ja sisestamine ka muutuja **n** jaoks.

Mõnikord on teatud põhjustel vaja meelega tekitada antud kohas lõputu kordusga sarnane tsükkel, mis katkestataks näiteks teise paralleelselt toimiva protsessi toimel. Sel juhul tuleks samuti kasutada **while**-lauset.

Lõputu kordus: while-lause



kordus

tegevused

lõpp kordus

Lõputu korduse saab **while**-lause abil määrata tingimusega, mille väärtus on alati **tõene**. Kõige lihtsam on kirjutada lausesse lihtsalt tõeväärtus **True**. Kasutatakse ka taolisi võrdlusi nagu 1 == 1 või muud sellist. Korduses olevate lausetega määratud tegevusi täidetakse sellisel juhul põhimõtteliselt lõputult. Korduse saab lõpetada vajutusega klahvidele Ctrl+C. **NB!** Selle klahvikombinatsiooni kasutamine võib osutada vajalikuks mõnikord ka siis, kui programm mingi vea tõttu tööd ei lõpeta. Sel juhul öeldaksegi, et programm jäi tsüklisse.

Lõputu kordus leiab kasutust erilise iseloomuga rakendustes, sest reeglina peab programmi töö lõppema loomulikul teel, st programmis määratud tingimuste alusel.

Lõputu kordus katkestusega: break-lause

```
import random
def arva():
  print ("Arva ära arv 1...100.")
  y = random.randint(1, 100)
  k = 0
  while True:
    k = k + 1
    if k > 13:
      print ("Katsete arv on täis!")
      tun = 0; break
    x = input("Paku arv => ")
    if x == "" :
       tun = 0; print("Kahju! "); break
    x = int(x)
    if x == y : tun = 1; break
    if x < y : print ("Vähe!")
    else:  print ("Palju!")
  if tun == 1:
    print("Korras! Katseid oli:", k)

arva()
```

Ette võib tulla olukordi, kus põhjuseid korduse katkestamiseks on mitu ja need asuvad erinevates kohtades. Sellistel juhtudel on kasulik kordamiste aluseks võtta lõputu kordus ning näha ette korduse katkestamine **break-l**ause abil, kui tekib vastav sündmus, mida kontrollitakse valikulauses. Muuhulgas saab selliselt modelleerida ka nn **järelkontrolliga kordust**, mille jaoks on mitmetes keeltes spetsiaalsed laused.

Esitatud näites on toodud arvu arvamise mängu uus variant. Siin on lisatud tingimus, et katsete arv ei tohi olla suurem ette antud arvust (programmis on selleks võetud 13). On arvestatud võimalusega, et mängija katkestab mängu, vajutades arvu sisestamise asemel lihtsal klahvi **Enter**, st tagastatakse tühi väärtus.

Antud juhul on korduse lõpetamiseks kolm võimalikku varianti:

- arv on ära arvatud, st x = = y
- katsete arv on ületatud, st k > 13
- kasutaja loobus, st x = = ''

For-lause

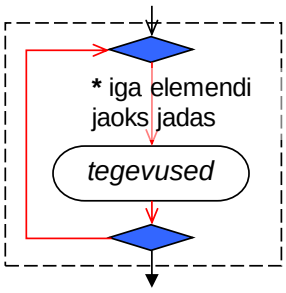
For-lause abil saab kirjeldada erineva täitmise põhimõtetega kordusi. Lause üldkuju on järgmine:

for *element* **in** *jada*

laused

[**if** *tingimus* : **break**]

laused



kordus Iga elemendi jaoks jadas

tegevused_1

kui *tingimus* **välju**

tegevused_2

lõpp kordus

Tegevusi täidetakse teatud jada või kogumi iga elemendi jaoks. Jada või kogumi määratlemiseks võib kasutada erinevaid viise ja vahendeid. Üheks kõige lihtsamaks on funktsioon **range**, mille üldkuju on järgmine:

range ([algus,] lõpp [, samm])

Näiteks määrab range(3, 12, 2] loetelu, mille elementideks on väärtused 3, 5, 7, 9, 11. Kui puudub algus, võetakse see võrdseks nulliga ja kui samm ei ole näidatud, võetakse see võrdseks ühega. Jada viimane liige võetakse ühe võrra väiksemaks kui lõpp. Näiteks, kui n = 5, siis tekitatakse järgmised jadad

range(n) => 0, 1, 2, 3, 4; **range**(1, n) => 1, 2, 3, 4; **range**(n, 8) => 5, 6, 7

Funktsiooni range abil saab kirjeldada etteantud kordamiste arvuga kordusi. Näiteks lause

```
for k in range(7):
print ('Tere!')
```

väljastab seitse korda teksti „Tere!“.

Loendite (massiivide) töötlemisel on väga levinud **for**-lause kasutamine. Kui määratletud on järgmine loend

X = [3, 5, 8, 10],

siis lause

for elem **in** X: **print** (elem**2, end = " ")

kuvab jada 9 25 64 100.

Näide: Naturaalarvude ruutude summa

```
def nats2(n1, n2):
    S = 0
    for k in range(n1, n2+1):
        S = S + k * k
    return S

print (nats2(5,13))
```

Funktsioon **nats2** leiab naturaalarvude ruutude summa alates algväärtusest **n1** kuni arvuni **n2**. Algus ja lõpp antakse pöördumisel parameetritele vastavate argumentidega.

Põhiosa funktsioonist moodustab **for**-lause. Kõigepealt võetakse muutuja ***S*** algväärtuseks 0, kusjuures **for**-lauses muudetakse juhtmuutuja **k** väärtust alates n1 kuni n2+1 ning liidetakse summale järgmise väärtuse ruut. Tulemuseks tagastatakse **S**.

Näide: Funktsiooni tabuleerimine ja maksimaalne väärtus

```
from math import *

def Fy(x):    return 3 * sin(2 * x + 3)

def Fz(x):
    return 3 *sin(2 * x + 3)-5 * cos(x / 3)

def tabuleerimine (F, a, b, n):
    h = (b - a) / n
    print ('Funktsioon:', F)
    for i in range(n + 1):
        x = a + i * h
        print ("%3d  %5.2f %10.4f" %(i, x, F(x)))

def Fmax(F, a, b, n = 1000):
    h = (b - a) / n
    ymax = F(a)
    for i in range(n + 1):
        x = a + i * h  # x punktis i
        y = F(x)
        if y > ymax: ymax = y
    return ymax

print ("Funktsiooni tabuleerimine")
x0 = float (input ("algus => "))
xn = float(input ("lõpp => "))
n = int(input ("jaotisi => "))
tabuleerimine (Fy, x0, xn, n)
print ("suurim Fy:", Fmax(Fy, x0, xn))
tabuleerimine (Fz, x0, xn, n)
print ("suurim Fz.", Fmax(Fz, x0, xn))
```

See ülesanne oli varem lahendatud ühe funktsiooni **Fy** jaoks **while**-lause baasil (vt lk 209). Siin kasutatakse korduse juhtimiseks **for**-lauset ning töödeldakse veel teist funktsooni **Fz**. Lisatud on ka üks lisategevus – maksimaalse väärtuse leidmine.

Peaprotseduur loeb algandmed **x0**, **xn**, **n** ning käivitab järjest protseduurid **tabuleerimine** ja **Fmax** funktsioonide **Fy** ja **Fz** jaoks.

tabuleerimine arvutab ja kuvab argumendi ja parameetrina antud funktsiooni väärtused määratud lõigul.

Tulemuste kuvamisel kasutatakse vormindamisstringi:
%*w*d – täisarv w positsiooni,
%*w*.*m*f – reaalarv w positsiooni, murdosa pikkus m.

Fmax leiab määratud lõigul funktsiooni maksimaalse väärtuse. Selles protseduris on parameetri **n** vaikeväärtus 1000. Kui vastavat argumenti pöördumisel ei anta (ja siin seda ei tehta), kasutatakse vaikeväärtust. Praegu tähendab see seda, et maksimumi arvutamisel kasutatakse jaotiste arvu 1000, mis tagab piisavalt suure täpsuse.

Programm demonstreerib võimalust funktsiooni (protseduuri) kasutamiseks parameetrina ja argumendina. Protseduuride **tabuleerimine** ja **Fmax** poole pöördutakse kaks korda. Esimesel korral on argumendiks funktsioon **Fy**, teisel **Fz**. Protseduurid **tabuleerimine** ja **Fmax** on selles mõttes universaalsed, et need ei ole seotud konkreetse kasutatava funktsiooniga – see antakse pöördumisel.

Loendid

Loend (*List*) ehk **ühemõõtmeline massiiv** on järjestatud väärtuste kogum. Sellist kogumit tähistatakse ühe nimega, tema elementidele viidatakse nime ja indeksi (järjenumbri) abil. Loendite käsitlemises on üsna palju ühist stringidega, kuid loendi igale elemendile eraldatakse mälus eraldi väli (pesa) ja loendi elemente saab muuta neid lisades, eemaldades, asendades, ...

Loendi olemus ja põhiomadused

Loendi elementide järjenumbrid algavad alati nullist. Viitamine elementidele toimub indeksnime abil järgmiselt:
loendi_nimi [*indeks*], indeks = 0, 1, 2, ...

Indeks paigutatakse nurksulgudesse [ ] ja see võib olla esitatud kui konstant, muutuja või avaldis.

Loendi pikkuse – elementide arvu loendis – saab teha kindlaks funktsiooniga len(loendi_nimi).

T = [*'Kask'*, *'Kuusk'*, *'Mänd'*, *'Paju'*, *'Saar'*, *'Tamm'*, *'Vaher'*]

P = [920, 670, 1120, 990, 1040, 1230, 848]

0, 1, 2, 3, 4, 5, 6

Loendis **T** on töötajate nimed, loendis **P** palgad; len(T) = len(P) = 7

T[0] = Kask T[1] = Kuusk; T[4] = Saar; P[0] = 920; P[4] = 1040; P[6] = 848

Saab viidata ka loendi elementide vahemikele: nimi [m : n]

T[1 : 4] => [*'Kuusk'*, *'Mänd'*, *'Paju'*]; P[4:] => [1040, 1230, 848]

Loendeid saab liita (ühendada): [*'kass'*, *'koer'*] + [*'lõvi'*, *'hunt'*] => [*'kass'*, *'koer'*, *'lõvi'*, *'hunt'*] ja korrutada: 3 * [*'kass'*, *'koer'*] => [*'kass'*, *'koer'*, *'kass'*, *'koer'*, *'kass'*, *'koer'*]

100 * [0] tekitab 100-st nullist koosneva massiivi.

Tühi loend luuakse lausega **nimi** = [ ], mis oleks vajalik näiteks uue loendi loomisel:

```
V = [ ]
for k in range (10):
    V.append(k**2 + 13)
```

Siin luuakse esiteks tühi loend **V** ja seejärel lisatakse sellesse 10 liiget.

Valik loendite meetodeid

append(x)	Lisab x-i (väärtus või objekt) loendi lõppu. T.append(*'Mets'*). Loendi T lõppu lisatakse väärtus 'Mets'
extend(loend)	Lisab loendile teise loendi. P.extend([2100, 1750, 2300]). P lõppu lisandub 3 väärtust
pop()	Tagastab viimase elemendi ja eemaldab selle loendist. maha = T.pop(). maha= *'Paju'*, T-st eemaldatakse
insert(nr, x)	Paneb väärtuse x numbriga määratud elemendi ette. T.insert(2, *'Lepp'*). Nimi *'Lepp'* elemendi nr. 2 ette
index(x)	Tagastab x-i asukoha (indeksi) loendis. k = T.index(*'Kuusk'*). k-le omistatakse väärtus 1
sort()	Sorteerib loendi elemendid kasvavas järjekorras

Näiteid loenditega

Eesti–inglise tõlketest

Loendis nimega **eesti** on sõnad eesti keeles, loendis **inglise** – inglise keeles. Programm palub järjest tõlkida eestikeelsed sõnad inglise keelde, teeb kindlaks õigete vastuste arvu ja protsendi.

```
def test_1 ():
  eesti  =  ["kass", "koer", "hiir", "lehm", "karu"]
  inglise = ["cat",  "dog",  "mouse", "cow", "bear"]
  print ("Tõlgi inglise keelde")
  n = len (eesti)
  oige = 0
  for k in range(n):
    vastus = input (eesti [k] + "=> ")
    if (vastus == inglise [k]):
      oige = oige + 1
    else:
      print ("Vale!")
  return oige / n * 100
print ("protsent:", test_1())
```

Programm leiab funktsiooni **len** abil elementide (sõnade) arvu loendites ning omistab selle muutujale **n**. Seejärel omistatakse muutujale **oige** algväärtus 0. Korduses kuvatakse järjest sõnu loendist **eesti**, kasutades elementidele viitamiseks indeksit (muutujat) **k**. Kasutaja antud vastust võrreldakse sama järjenumbriga sõnaga loendis **inglise**. Kui vastus on õige, suurendatakse muutuja **oige** väärtust, vastupidisel juhul kuvatakse teade „Vale!“. Kui kõik sõnad on läbitud, leitakse õigete vastuste protsent.

Eesti–inglise tõlketest 2

```
def test_2():
  eesti = ["kass", "koer", "hiir", "lehm", "karu"]
  inglise = ["cat",  "dog",  "mouse", "cow", "bear"]
  print ("Tõlgi inglise keelde")
  oige = 0; k = 0
  for sona in eesti:
    vastus = input ( sona + "=")
    if (vastus == inglise [k]): oige += 1
    else: print ("Vale!")
    k += 1
  return oige / len (eesti) * 100
```

Selles näites on eelmise testi ülesanne lahendatud teisiti.

for-lauses kasutatakse jada määramisel loendit **eesti**.

Vastavalt sellise **for**-lause käsitlemise reeglitele täidetakse sisemised laused loendi iga elemendi korral ning viitamiseks elementidele ei kasutata indekseid.

Selliseid korduseid nimetatakse mõnedes programmeerimiskeeltes **for each**-kordusteks ehk korduseks kollektsiooni või hulgaga. Viitamiseks teise loendi (inglise) elementidele kasutatakse indeksit (k).

Näide: Tõlge inglise–eesti

```
def trans_IE() :
   eesti = ["kass", "koer", "hiir", "lehm", "karu"]
   inglise = ["cat",  "dog",  "mouse", "cow", "bear"]
   sona = input ("anna sõna => ")
   nr = -1; k = 0
   for elem in inglise :
      if sona == elem :
         nr = k; break
      k += 1
   if nr == -1:
      print ("Sellist sõna minul ei ole!")
   else :
      print ("Siin see on: ", eesti [nr])
```

Kui kasutaja annab sõna inglise keeles, siis funktsioon peab tõlkima selle eesti keelde.

Kasutades **for**-lauset, tehakse otsimine loendis **inglise**. Muutujale **nr** omistatakse väärtus -1 ning muutujale **k** algväärtus 0. Sisestatud sõna võrreldakse järjest loendi **inglise** jooksva elemendiga. Kui need on samad (võrdsed), omistatakse muutujale **nr** muutuja **k** väärtus ja kordus katkestatakse. Kui sõnad ei võrdu, suurendatakse **k** väärtust ühe võrra. Kui korduse lõppemisel jäi muutuja **nr** väärtuseks -1, tähendab see, et otsitavat sõna loendis pole.

Loendi sisestamine klaviatuurilt

Funktsioon **loe_vek** võimaldab luua loendi, sisestades selle elemendid klaviatuurilt.

```
def loe_vek(V, n):
   for k in range(n):
      elem = input("anna element => ")
      V.append(elem)


X = [ ]
n = int(input("mitu => "))
loe_vek(X, n)
for k in range(n):
   print (X[k])
```

Funktsiooni parameetriteks on loend (vektor) ja elementide arv selles. **For**-lauses küsitatakse järjest väärtusi ja lisatakse meetodi **append** abil sisestatud väärtused loendi lõppu. Elementide tüüp võib olla suvaline – **string**, **int** või **float**.

Kasutamise näide:
Korraldusega X = [] luuakse tühi loend **X**. Sisestatakse elementide arv **n** ja käivitatakse funktsioon.

Kontrolliks väljastatakse loodud vektor ekraanile.

Loendi loomine juhuarvudest

```
import random
def tee_vek(V, n, mini, maxi):
  for i in range(n):
    elem = random.randint(mini, maxi)
    V.append(elem)
X = [ ]
n = int(input("mitu => "))
tee_vek(X, n, -100, 100)
for k in range(n):
  print (X[k], end = " ")
```

Programmide katsetamiseks ja testimiseks kasutatakse sageli juhuslikke arve, mille hulk võib olla suvaline.

Selles näites antakse funktsioonile **tee_vek** parameetritena ette loend **V**, kuhu salvestatakse genereeritavad juhuarvud, nende hulk **n** ja arvude piirid **mini** ja **maxi**.

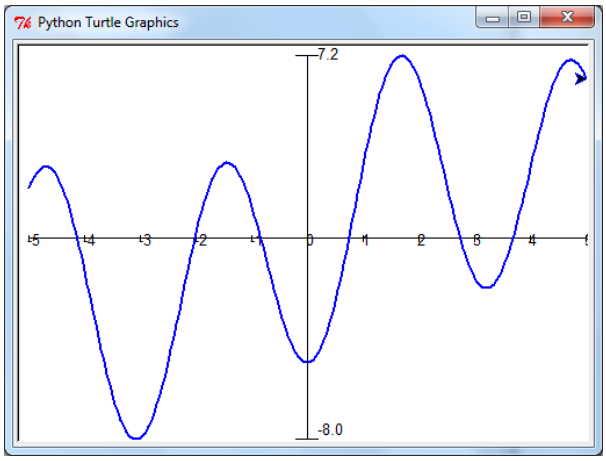
Antud juhul genereeritakse täisarve.

Konkreetne vektor (**X**), elementide arv **n** ja piirid (-100, 100) antakse peaprotseduurist pöördumisel argumentidena.

Funktsiooni uurimine

Rakendus võimaldab teha suvalise ühemuutuja funktsiooni graafiku etteantaval lõigul, arvutada ja kuvada argumendi ja funktsiooni väärtused jaotiste arvuga määratud punktides, leiada samal lõigul mõned funktsiooni karakteristikud: minimaalse ja maksimaalse väärtuse ning nendele vastavad x väärtused, määratud integraali ja pindala ning nullkohad. Argumendi ja funktsiooni väärtused salvestatakse loenditesse (massiividesse), millest viimaseid kasutatakse karakteristikute leidmisel ja graafiku tegemisel.

Allpool on toodud tulemuste näited funktsiooni F(x) = 3 * sin(x / 2) + 5 * cos(2 * x + 3) jaoks lõigul [-5; 5], toodud graafikaakna koopia ning Shelli aknas kuvatavad x, y ja karakteristikute väärtused. Allpool on toodud programmi protseduurid ja kommentaarid nende kohta. Vt ka sama ülesande realisatsiooni lihtmuutujatega mooduli **funktsioon** abil.



x	y	
-5.0	1.974	y min: -7.993 x: -3.05
-4.0	-1.31	y max: 7.22 x: 1.7
-3.0	-7.942	integraal: 2.6906
-2.0	0.177	pindala: 35.7778
-1.0	1.263	nullkohad
0.0	-4.95	-4.13438
1.0	2.857	-2.01871
2.0	6.294	-0.838
3.0	-1.563	0.74594
4.0	2.75	2.74154
5.0	6.333	3.6896

```
from math import *
from turtle import *

def Fy(x):
    return 3 * sin(x / 2) +  5 * cos(2 * x + 3)

def Fz(x,  p = 2):
    if (x <= p):
        return 5 * cos(3 * x) * sin(x/2) + 1
    else:
        return 7 * sin(2 * x - 1) * cos(x  / 3)
```

Kasutaja võib uurida suvalise ühemuutuja funktsiooni käitumist. Selleks peab ta koostama vastava protseduuri funktsiooni väärtuse leidmiseks sellisel kujul nagu on näidatud vasakul ning peale programmi käivitamist sisestama programmi teadete vastuseks vajalikud algandmed, sh ka funktsiooni nime.

Vt allpool protseduure **loe_alg** ja **pealik** (lk 217, 219).

```
def pealik():
    '''peaprotseduur'''
    F, a, b, n, n2 = loe_alg()
    tee_tabel(F, a, b, n)
    X = tee_mas_X(a, b, n2)
    Y = tee_mas_Y(F, X)
    mini, maxi = tee_kar(X, Y)
    AL = 800; AK = 600
    aken(AL, AK, a, b, n, mini, maxi)
    graafik (X, Y, "blue")
    mainloop()
```

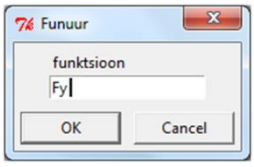
Programm on jagatud järgmisteks protseduurideks:

loe_alg () – algandmete sisestamine
tee_tabel (F, a, b, n) – x ja y väärtused käsuaknasse
tee_mas_X (a, b, n2) – loendi (massiivi) X loomine
tee_mas_Y (F, X) – loendi (massiivi) Y loomine
tee_kar (X, Y) – karakteristikute leidmine ja kirjutamine Shelli
min_nr (V) – min vektoris ja selle järjenumber
max_nr (V) – max vektoris ja selle järjenumber
integraal (X, Y) – määratud integraal antud lõigul
pindala (X, Y) – pindala antud lõigul
aken(AL, AK, a, b, n, mini, maxi) – graafikaakna seadistamine
graafik (X, Y, *"blue"*) – graafiku joonistamine

```
def loe_alg():
    ''' Algandmete lugemine. F – funktsiooni nimi,
    a, b – lõigu otsapunktid, n, n2 – jaotiste arvud '''
    F = eval(textinput("Funuur", "funktsioon"))
    a = numinput("Funuur", "lõigu algus", -5)
    b = numinput("Funuur", "lõigu lõpp", 5)
    n = int(numinput("Funuur", "jaotisi tabelis", 10))
    n2 = int(numinput("Funuur", "jaotisi graafikul", 200))
    return  F, a, b, n, n2
```

Algandmete sisestamiseks kasutatakse graafikamooduli **turtle** sisendbokse.

Lause täitmisel kuvatakse boks, kuhu kasutaja saab sisestada väärtuse.



Esimese lausega sisestatakse uuritava funktsiooni nimi.

```
def tee_tabel(F, a, b, n):
    ''' argumendi ja funktsiooni väärtused Shelli aknasse.
      F – funktsioon, a, b – lõigu otsapunktid, n - jaotisi  '''
    h = (b -a) / n            # tabuleerimise samm
    print ("  x \t", "  y")    # tabeli päis Shelli
    for i in range(n+1):   # jaotise number ( i ) 0...n
        x = a + i * h          # x- i jooksev väärtus
        y = F(x)               # y- i jooksev väärtus
        print (round(x, 2), "\t", round(y, 3))  # x, y Shelli
```

Protseduur arvutab argumendi ja funktsiooni väärtused ning kirjutab need ka käsuaknasse.

Parameetri F väärtus – kasutaja poolt protseduuri **loe_alg** täitmisel sisestatud funktsiooni nimi, tuleb peaprotseduurist. Jaotiste arv **n** valitakse lõigu pikkust arvestades. Näiteks **n** on 10...20 ja lõigu pikkus 10...20 ühikut.

```
def tee_kar(X, Y):
    ''' karakteristikute leidmine ja kirjutamine Shelli
      aknasse. X – argumendi, Y- funktsiooni vekorid'''
    mini, nr = min_nr(Y)
    print ("mini Y", round(mini, 3), " x:", round(X[nr], 2))
    maxi, nr = max_nr(Y)
    print ("maks Y:", round(maxi, 3), " x:", round(X[nr], 2))
    print ("integral:", round(integral(X, Y), 4))
    print ("pindala:", round(pindala(X, Y), 4))
    NV = nullid(X, Y)
    print ("nullkohad")
    for x in NV:
        print (round(x, 5))
    return mini, maxi
```

Protseduur käivitab alamprotseduurid: **min_nr**(Y), **max_nr**(Y), **integral**(X, Y), **pindala**(X, Y) ja **nullid**(X, Y) ja kuvab viimaste poolt tagastatud väärtused käsuaknas.

Karakteristikute leidmiseks kasutatakse protseduurides **tee_mas_X**(x0, xn, n) ja **tee_mas_Y**(F, X) arvutatud ja loendites (vektorites) X ja Y salvestatud väärtusi.

Karakteristikute piisava täpsusega väärtuste saamiseks peab punktide arv (elementide arv vektorites X ja Y) olema piisavalt suur – lõigul pikkusega 10–20 ühikut vähemalt 100...200 või enam.

```
def tee_mas_X(x0, xn, n):
    ''' argumendi väärtuste vektori loomine.
      x0, xn – otsapunktid,   n – jaotiste arv '''
    h = (xn - x0) / n       # x muutmise samm
    X = [ ]                 # tühi loend
    for i in range(n+1):  # jaotise number ( i ) 0…n
        x = x0 + i * h          # x- i jooksev väärtus
        X.append(x)             # lisatakse x loendisse
    return X
```

Protseduur arvutab argumendi (x) väärtused ja salvestab tagastatavas loendis (massiivis) X.

for-lauses muudetaks jaotamispunkti **i** väärtust 0 kuni n.

NB! Funktsioon **range**(n+1) tekitab väärtuste jada: 0, 1, 2, ... n.

```
def tee_mas_Y(F, X):
    ''' argumendi  massiivi Y loomine
      loendis X olevate väärtuste alusel'''
    Y = [ ]
    for x in X:  # x väärtusteks võetakse X-i elemente
        Y.append(F(x))
    return Y
```

Protseduur arvutab ja salvestab tagastatavas loendis **Y** funktsiooni väärtused, kasutades protseduuri, mille nimi tuleb anda parameetri **F** väärtuseks. Argumendi (x) väärtused on varem salvestatud loendis **X**. **for**-korduse täitmisel võetakse muutuja **x** väärtusteks järjest elemente loendist **X** ning leitakse vastav funktsiooni väärtus.

```
def max_nr(V):
    ''' maks vektoris ja
      selle indeks '''
    n = len(V)
    maxi = V[0]; nr = 0
    for i in range (n) :
        if  V[i] > maxi :
            maxi = V[i]; nr = i
    return maxi, nr
```

```
def min_nr(V):
    ''' min vektoris ja
      selle indeks '''
    n = len(V)
    mini = V[0]; nr = 0
    for i in range (n) :
        if  V[i] < mini :
            mini = V[i]; nr = i
    return mini, nr
```

Need protseduurid on peaaegu identsed, peamine erinevus on **if**-lauses olevas võrdlusmärgis.

Mõlemad funktsioonid tagastavad kaks väärtust: **maxi** või **mini** ja **nr**, mis on maxi või mini asukoht (järjenumber) loendis.

maxi ja **mini** väärtusi kasutatakse kasutaja koordinaatsüsteemi määramisel graafikaakna jaoks.

```
def integral(X, Y):
    '''määratud integraal
      trapetsvalemiga'''
    n = len(X)
    h = X[1] - X[0]
    S = (Y[0] + Y[n-1]) / 2
    for i in range(1, n - 1):
        S = S + Y[i]
    return h * S
```

```
def pindala(X, Y):
    ''' pindala joone ja
      X-telje vahel'''
    n = len(X)
    h = X[1] - X[0]
    S=(abs(Y[0])+abs(Y[n-1]))/2
    for i in range(1, n - 1):
        S = S + abs(Y[i])
    return h * S
```

Tegemist on funktsiooniga, milles kasutatakse trapetsvalemit:

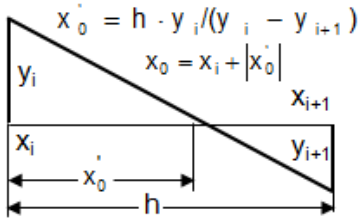
$S = h * (y_0/2 + y_1 + y_2 + \ldots + y_{n-1} + y_n/2)$

Määratud integraal kujutab algebraliselt pindala, arvestatakse funktsiooni väärtuste märki.

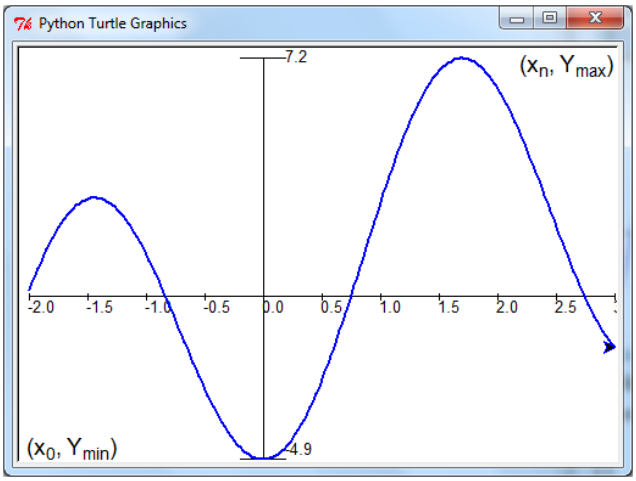
Pindala arvutamiseks kasutatakse absoluutväärtusi.

```
def nullid(X, Y):
  ''' nullkohtade leidmine'''
  NV = [ ]; n = len(X)
  h = X[1] - X [0]  # samm
  for i in range (n - 1):
    if Y[i] == 0: NV.append(X[i])
    if Y[i] * Y[i + 1] < 0:
      x0 = h * Y[i] / (Y[i] - Y[i+ 1])
      x0 = X[i] + abs(x0)
      NV.append (x0)
  return NV
```

Vaadeldaval lõigul (a, b) võib olla suvaline hulk nullkohti. Eeldades, et jaotiste arv on piisavalt suur ehk lõigu alamjaotiste pikkus (h) on väike (näiteks 0,1 … 0,01 ühikut), võib toimida järgmiselt. Vaadata järjest läbi funktsioonide väärtused (loend Y) ning võrrelda naaberpunktide väärtusi. Kui väärtused on erineva märgiga (yi·yi+1<0), siis kahe punkti vahel on nullkoht, mis leitakse esitatud valemitega, arvestades skeemi.




```
def aken(W, H, x0, xn, n, Ymin, Ymax):
  ''' Aken ja teljed. W akna laius, H - kõrgus
     x0 - lõigu algu, xn - lõpu lõpp;
     Ymin, Ymax - funktsioonide min, max '''
  setup(W, H);  speed(0)
  setworldcoordinates(x0, Ymin, xn,  Ymax)
  penup(); setpos (x0, 0);
  pendown(); setpos(xn, 0)
  penup(); setpos (0, Ymin)
  pendown(); setpos(0, Ymax)
  x = x0; hx=(xn-x0)/n  # kriipsude samm
  hk =(Ymax - Ymin)/100  # kriipsu pikkus
  while x <= xn:
    penup();   setpos(x, -hk)
    pendown(); setpos(x,  hk)
    penup(); setpos(x, -3 * hk)
    write (str(round(x)), font=('Arial', 10))
    x = x + hx
  # markerid Y-teljele
  penup(); setpos(-0.2 , Ymax); pendown()
  setpos(0.2 , Ymax); penup()
  setpos(0.2, Ymax-2*hk); pendown()
  write (str(round(Ymax, 1)),font=('Arial', 10))
  penup(); setpos(-0.2 , Ymin); pendown()
  setpos(0.2 , Ymin)
  write (str(round(Ymin, 1)), font=('Arial', 10))
```



Protseduur määrab graafikaakna mõõtmed pikselites, kasutaja koordinaadid, joonistab teljed, teeb jaotised X-teljele ning **Ymax** ja **Ymin** markerid Y-teljele. Väga oluline on turtle meetod setworldcoordinates(x0, Ymin, xn, Ymax), millega määratakse nn maailma ehk kasutaja koordinaadid, nii et graafik mahuks parasjagu aknasse. Süsteem ise valib kasutatavad ühikud ning programmi koostaja ei pea väga palju sellega tegelema.

```
def graafik(X, Y, jv = "red", w = 2 ):
    n = len(X); speed(0)
    penup(); setpos(X[0],Y[0]); pendown()
    pencolor(jv); width(w)
    for i in range(n):
      setpos(X[i], Y[i])

pealik ()
```

Graafiku (joone) joonistamine, kui aken on fikseeritud ning argumendi ja funktsiooni väärtused asuvad loendites, on juba võrdlemisi lihtne. Pliiats viiakse punkti (X[0],Y[0]) ning edasi liigutakse järjest joone ühest punktist teise. Selleks, et graafiku joon oleks sile, peab jaotiste arv olema suhteliselt suur: 100–200 või enam.

Näide: Loto

```
import random
def loto():
  ''' Rakendus imiteerib loto mängu '''
  nmax = int (input("Maks number ? "))
  np = int (input("Numbreid piletil? "))
  nl = int (input("Numbreid loosimisel? "))
  # teeb pileti numbrid
  P = tee_num(np, nmax)
  sort_jada (P)
  print (P)
  # teeb loosimise numbrid
  L = tee_num(nl, nmax)
  L.sort()
  print (L)
  # leiab ühesugused väärtused
  T = yhised(P, L)
  print ("Tabas: ", len(T))
  print(T)
```

Rakendus võimaldab luua kaks komplekti juhuslikke arve etteantavas vahemikus. Arvud salvestatakse vastavates loendites: üks sisaldab pileti numbreid, teine loosimise numbreid. Numbrid ühes loendi piires ei kordu. Selleks, et kasutajal oleks lihtsam võrrelda ja kontrollida pileti ja loosimise numbreid, kuvab programm need sorteerituna kasvavas järjestuses. Programm teeb kindlaks kokkulangevate numbrite arvu ja kuvab need eraldi loendis.

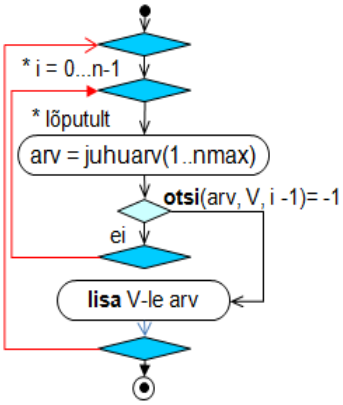
Siin on kasutusel mitu tüüpalgoritmi: erinevate juhuarvude genereerimine, väärtuste otsimine massiivides, massiivi elementide sorteerimine, kokkulangevate väärtuste leidmine massiivides. Toodud on erinevad variandid.

Maks number ? 36
Mitu numbrit piletil? 5
Mitu numbrit loosimisel? 10
[7, 21, 24, 25, 29]
[1, 3, 7, 15, 20, 24, 27, 29, 33, 35]
Tabas: 3
[7, 24, 29]

```
def tee_num(n, nmax):
  ''' Loob vektori V n erinevast
     juhuarvust [1..nmax].
     Kasutab funktsiooni otsi()'''
  V = [ ]
  for i in range(n):
    while True:
      arv = random.randint(1, nmax)
      if otsi(arv, V, i - 1) == -1:
        V.append (arv); break
  return V
```

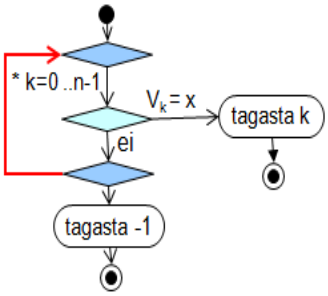
Protseduur loob **n** üksteisest erinevat arvu vahemikus 1…nmax ja salvestab need loendis V.

for-lause muudab elemendi järjenumbrit 0..n-1. Luuakse juhuarv vahemikus 1..nmax ning kontrollitakse funktsiooniga **otsi**, kas selline arv on eespool olemas. Kui on, genereeritakse uus arv, kui ei – salvestatakse **arv** loendis **V** ja genereeritakse järgmine arv




```
def otsi(x, V, n):
  ''' otsib x asukohta V-s 0..n
     kui ei ole, tagastab -1 '''
  for k in range(n + 1):
    if x == V[k]: return k
  return -1
```

Funktsioon otsib väärtusega **x** võrdset väärtust loendist **V**, alates algusest kuni elemendini numbriga **n**. Võimalik, et ka kuni lõpuni. Kui vastav väärtus on olemas, tagastab selle järjenumbri, kui ei, siis tagastab väärtuse -1.



```
def tee_num_1(n, nmax):
    ''' Loob vektori n erinevast
        juhuarvust [1..nmax]. Kasutab
        random-meetodit shuffle() '''
    arvud = list(range(1, nmax + 1))
    random.shuffle(arvud)
    return arvud[: n]
```

Antud protseduuris kasutatakse juhuslike üksteisest erinevate väärtuste genereerimiseks Pythoni vastavaid vahendeid.

Lausega arvud = list(range(1, nmax + 1)) luuakse loend järjestikustest väärtustest 1…nmax.

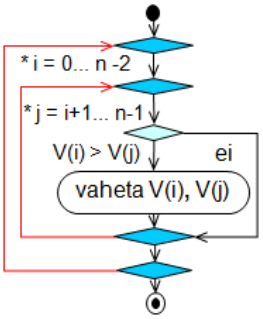
Mooduli **random** meetodiga **shuffle** pannakse need juhuslikku järjekorda. Käsk **arvud[: n]** eraldab loendist **arvud** esimesed **n** arvu.

```
def sort_jada(V):
    '''sordib V jadameetodiga'''
    n = len(V)
    for i in range (n-1):
        for j in range(i+1, n):
            if V[i] > V[j]:
                V[i], V[j] = V[j], V[i]
```

Kõige lihtsam sorteerimise meetod. Võrreldakse elemente Vi (i=0..n-2) järgnevate elementidega Vj (j=i+1..n-1).

Kui Vi > Vj, vahetakse elemendid.

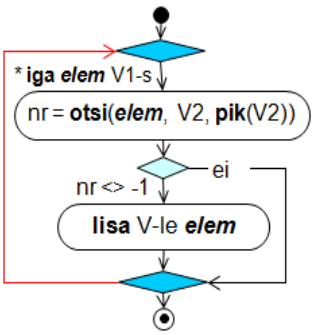
Järgmises jaotises on näiteks toodud veel sorteerimisalgoritme.




```
def yhised(V1, V2):
    '''leiab yhesugused väärtused
       vektorites V1 ja V2 ning
       salvestab need vektoris V '''
    V = [ ]
    for elem in V1:
        nr = otsi(elem, V2, len(V2) - 1)
        if nr != -1: V.append(elem)
    return V

loto()
```

Võrreldakse väärtusi kahes vektoris **V1** ja **V2**. Võetakse järjest elemente vektorist **V1** ja otsitakse võrdset väärtust vektoris **V2**. Kui leitakse kokkulangev väärtus, lisatakse element vektorisse **V**.

Väärtuste otsimisel kasutatakse funktsiooni **otsi**.



Mõned sorteerimisalgoritmid

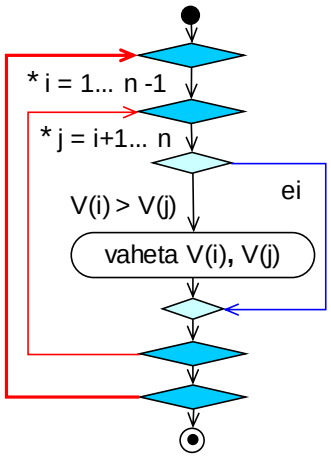
Sorteerimisalgoritmide demodega saab tutvuda järgmiselt lingilt Demod.

Jadasortimine

Jadasortimine on kõige lihtsam sortimismeetod, mille aeg on enam-vähem võrreldav **mullsortimisega** (täpsemalt allpool). Jadasortimises võrreldakse elemente V_i (i = 1..n-1) järgnevate elementidega V_j (j = i+1..n-1). Kui $V_i > V_j$, siis vahetakse need elemendid omavahel. Elementide arv vektoris on **n**.

Sellise meetodi korral tõstetakse väiksemad väärtused järjest ülespoole.

Täitmise aeg on proportsionaalne $\mathbf{n^2}$.



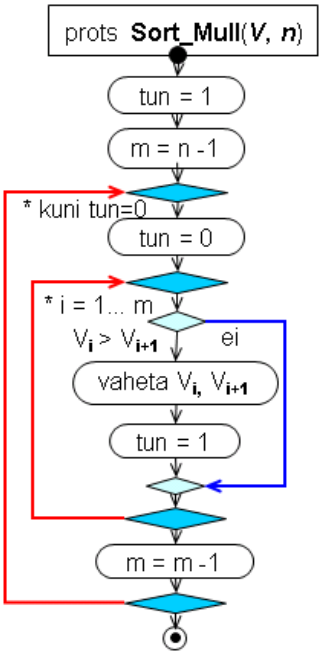

```
protseduur Sort_jada(V, n)
kordus i = 1, n - 1
   kordus j = i + 1, n
      kui V(i) > V(j) siis
         abi = V(i);  V(i) = V(j)
         V(j) = abi
      lõpp kui
   lõpp kordus
lõpp kordus
```

```
def sort_jada(V):
  n = len(V)
  for i in range (n-1):
    for j in range(i+1, n):
     if (V[i] > V[j]):
       V[i], V[j] = V[j], V[i]
```

Mullsortimine

Mullsortimine on üks vanemaid sortimisalgoritme, mille kasutamisel vaadatakse massiiv korduvalt läbi. Iga kord võrreldakse naabreid ning kui V(i) > V(i+1), vahetakse need omavahel ümber. Väiksemad väärtused tõusevad ülespoole, suuremad laskuvad allapoole. Läbivaatamine kestab seni, kuni ei tehta enam ühtegi vahetust.

See meetod on 2–3 korda aeglasem kui valiksortimine ning üldjuhul ka veidi aeglasem kui jadasortimine. Kiire on see meetod siis, kui väärtused on peaaegu järjestatud.



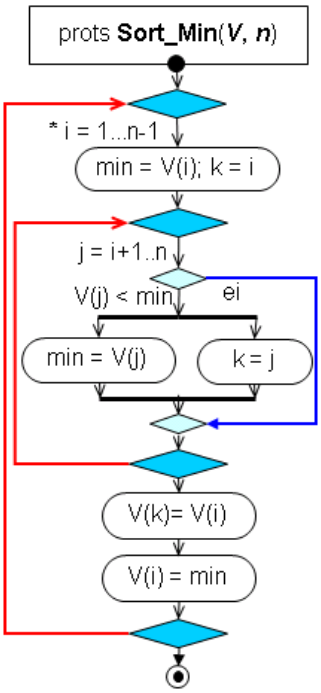

```
protseduur sort_mull(V, n)
 tun = 1
 m = n - 1
 kordus kuni tun = 0
   tun = 0
   kordus i = 1... m
     kui V(i) > V(i + 1) siis
       abi = V(i)
       V(i) = V(i + 1)
       V(i + 1) = abi
       tun = 1
     lõpp kui
     m = m - 1
 lõpp kordus
```

```
def sort_mull(V):
   n = len(V)
   m = n - 1
   tun = 1
   while (tun != 0):
     tun = 0
     for i in range (m):
       j = i + 1
       if (V[i] > V[j]):
         V[i], V[j] = V[j], V[i]
         tun = 1
     m = m – 1
```

Valiksortimine

Selle meetodi korral muudetakse indeksi **i** väärtust vahemikus 1 kuni **n**-1 ning iga **i** väärtuse korral leitakse minimaalne element vektori osas järjenumbritega **i** kuni **n** ja vahetatakse see elemendiga number **i**.

Valiksortimine on 2–3 korda kiirem kui mullsortimine.



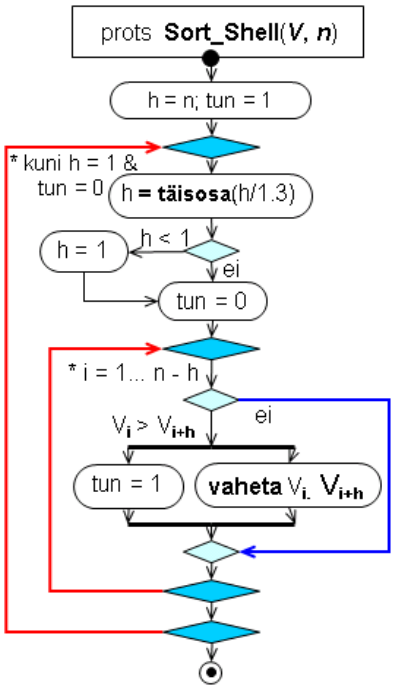

```
protseduur sort_min(V, n)
 kordus i = 1 kuni n - 1
   min = V(i)
   k = i
   kordus j = i + 1 kuni n
     kui V(j) < min siis
       min = V(j)
       k = j
     lõpp kui
   lõpp kordus
   V(k) = V(i)
   V(i) = min
 lõpp kordus
```

```
def sort_min(V):
  n = len(V)
  for i in range(n-1):
     min = V[i]
     k = i
     for j in range(i+1, n):
       if  V[j] < min:
         min = V[j]
         k = j
     V[k]= V[i]
     V[i] = min
```

Shelli meetod

See meetod on sarnane mullsortimisele, kuid siin kasutatakse **muutuvat sammu**, mis on alguses suur, edaspidi järjest väheneb. Keerukus sõltub oluliselt sammu valikust, kuid üks paremaid sammu muutmise reegleid on $n_i = n_{i-1} / 1{,}3$.

Shelli meetodi keerukus on umbes $n*\log_2 n$, mis on oluliselt kiirem eelmistest, eriti kui n väärtus on suur.
n = 1000 – umbes 10 korda, n = 100 000 – umbes 500 korda, n = 1 000 000 – umbes 5000 korda!




```
prots Sort_Shell(V, n)
   h = n
   tun = 1
   kordus kuni h=1 & tun=0
     h = täisosa(h/1,3)
     kui h < 1 siis h = 1
     tun = 0
     kordus i = 1... n - h
       kui V(i) > V(i + h) siis
         abi = V(i)
         V(i) = V(i + h)
         V(i + h) = abi
         tun = 1
       lõpp kui
     lõpp kordus
   lõpp kordus
```

```
def sort_shell(V):
  n = len (V)
  h = n
  while True:
    h = int(h // 1.3)
    if h < 1: h = 1
    tun = 0
    for i in range (0, n-h):
      if  V[i] > V[i + h]:
        V[i], V[i + h] = V[i + h], V[i]
        tun = 1
    if (h == 1 and tun == 0 ): break
```

Tabelid ja maatriksid

Nii tavaelus kui ka paljudes programmeerimiskeeltes kasutatakse tabelit kui ülevaatlikku andmekogumit, mis koosneb järjestatud ridadest ja veergudest. Tabeli lahtrites võivad olla arvud, stringid, … väärtused. Lahter võib olla ka tühi. Programmis **Meeskond** kasutatavad andmed meeskonna liikmete kohta võiks esitada tabelina nagu näidatud allpool. Selliselt esitatud andmestruktuure nimetatakse ka kahemõõtmelisteks massiivideks. Viitamiseks massiividele ja tabelitele kasutatakse nime koos kahe indeksiga, milleks on reanumber ja veerunumber: **nimi**[*rida*][*veerg*].

Tabel **T**

nimi	**pikkus**	
Haab	193	0
Kask	181	1
Mänd	178	2
Paju	203	3
Saar	197	4
Tamm	177	5
0	1	

Maatriks **A**

5	7	-9	4	8	0
-3	-8	1	-9	-4	1
6	.3	2	3	4	2
1	-6	3	4	-8	3
0	1	2	3	4	

T[rida][veerg] **A**[rida][veerg]

T[0, 0], T[3, 1], T[5][1] A[0, 0], A[0, 2], A[2,1], A[4, 4]

Pythonis esitatakse tabelid ja maatriksid loendite loenditena, kus loendi elemendid võivad olla ka loendid. Näites **Meeskond** olid andmed meeskonna liikmete kohta esitatud kahe lihtloendiga: **nimed** ja **P** (pikkused).

```
nimed =  [ 'Haab', 'Kask', 'Mänd', 'Paju', 'Saar', 'Tamm' ]
P =  [  193,   181,   178,   203,   197,   177  ]
```

Tabelina ehk loendite loendina võib sama info esitada järgmiselt:

```
T = [ ['Haab', 193], ['Kask', 181], ['Mänd', 178], ['Paju', 203], ['Saar', 197], ['Tamm', 177 ] ].
```

Tabelis olevad loendid on kõik võrdse pikkusega: elementide arv kõikides tabeli ridades on sama.

Järgnevas näites on toodud mõned protseduurid vaadeldava tabeliga.

```
def fkesk(T, v):
  '''veeru keskmine'''
  m = len(T)
  S = 0
  for i in range(m):
     S = S + T[i][v]
  return S / m
```

```
def fmaks(T, v):
  ''' max veerus v'''
  m = len(T)
  maks = T[0][v]
  nr = 0
  for i in range(1, m):
    if T[i][v] > maks:
      maks = T[i][v]
      nr = i
  return maks, nr
```

```
T = [['Haab', 193], ['Kask', 181], ['Mänd', 178], \
 ['Paju', 203],    ['Saar', 197], ['Tamm', 177 ] ]

print("Meeskonna liikmed")
kuva_tab(T)
kesk = fkesk(T,1)
print ("\nkeskmine pikkus:", round(kesk, 1))
maxi, nr = fmaks(T,1)
print ("pikim:", T[nr])
```

```
def kuva_tab(T):
  ''' tabeli kuvamine '''
  m = len(T); n = len(T[0])
  for i in range(m):
    for j in range(n):
      print (T[i][j], end = ' ')
    print()
```

```
Meeskonna liikmed
Haab 193
Kask 181
Mänd 178
Paju 203
Saar 197
Tamm 177
keskmine pikkus: 188.2
pikim: ['Paju', 203]
```

Funktsioon **fkesk**(T, v) leiab antud numbriga veeru (v) elementide aritmeetilise keskmise, **fmaks**(T, v) – maksimaalse elemendi ja rea numbri, kus see element asub. Mõlemad funktsioonid on mõnevõrra üldisemad kui vaja antud ülesande jaoks, ning vastava karakteristiku leidmine on kirjeldatud mitte konkreetse veeru (1), vaid suvalise veeru jaoks.

Ridade arvu tabelis – alamloendite arvu loendis – teeb kindlaks funktsioon lausega m = len(T).

Protseduuris **kuva_tab**(T), mis kuvab parameetrina **T** esitatud loendi tabelina, kasutatakse ka tabeli veergude arvu. Protseduur on antud ülesande vajadustest universaalsem, set ridade ja veergude arv võib olla suvaline. Paneme tähele elementide paigutamise määramist tabelina **print**-funktsioonides. Funktsioonis **print** (T[i][j], end = ' ') kasutatakse argumenti end = '', mis blokeerib automaatse ülemineku peale funktsiooni täitmist. Tabeli ühe rea elemendid paigutatakse järjest ekraani ühele reale. Iga kord peale sisemise korduse lõppu täidetakse funktsioon **print**(), mis viib uuele reale.

Näide: Operatsioonid maatriksitega

Järgnevalt on toodud mõned näited protseduuridest, mis täidavad erinevaid operatsioone maatriksiga nagu näiteks maatriksi loomine juhuslikest arvudest, maatriksi kuvamine, maatriksi ridade ja veergude leidmine jms.

```
def tee_mat(m, n, a = -10, b = 10):
    ''' maatriksi loomine juhuarvudest (a…b) '''
    A = [ ]  # tühi loend
    for i in range(m):
        rida = [ ]  # tühi loend uue rea jaoks
        for j in range(n):     # arvude loomine
            rida.append (randint(a, b))  # lisamine
        A.append(rida)   # rea lisamine loendisse
    return A   # tagastab loodud maatriksi

def ri_sum(A): # maatriksi ridade summad
    V=[ ]  # tühi vektor summade jaoks
    m = len(A); n = len(A[0])  # read ja veerud
    for i in range(m):  # iga rea jaoks 0…m-1
        S = 0  # rea i summa nulli
        for j in range(n): # iga veeru jaoks 0…n-1
            S = S + A[i][j]  # liita element summale
        V.append(S)      # summa vektorisse
    return V   # tagastab loodud maatriksi
```

```
def kuva_mat_R(A):
    ''' maatriksi kuvamine ridade kaupa '''
    m = len(A)  # ridade arv maatriksis
    for i in range(m):
        print (A[i])   # rea i kuvamine
def kuva_vek(V):
    ''' vektori kuvamine vertikaalselt '''
    m = len(V)   # elementide arv vektoris
    for i in range(m):
        print (V[i])    # elemendi i kuvamine

def ve_sum(A): # maatriksi veergude summad
    V=[ ]  # tühi vektor summade jaoks
    m = len(A); n = len(A[0])  # read ja veerud
    for j in range(n):  # iga veeru jaoks 0…n-1
        S = 0   # veeru j  summa nulli
        for i in range(m):  # iga rea jaoks 0…m-1
            S = S + A[i][j]    # liita element summale
        V.append(S)       # summa vektorisse
    return V     # tagastab loodud maatriksi
```

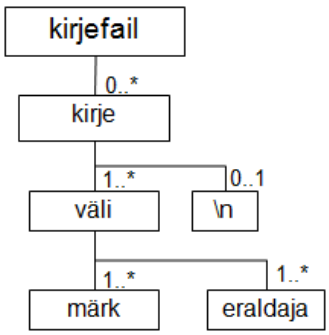
```
# peaprotseduur
M = tee_mat(4, 5)
# maatriks kuvatakse kolmel erineval kujul
print (M); kuva_tab(M); kuva_mat_R(M)
# veergude summade leidmine ja kuvamine
print("veergude summad")
VS = ve_sum(M); print(VS)
# max ja number antud veerus
vrg =int(input("Anna veerg, mille max "))
maxi, nr = fmaks(M, vrg)
print ("maksi veerus", vrg, maxi, "number:", nr)
# ridade summade leidmine ja kuvamine
print("ridade summad")
RS = ri_sum(M); print(RS)
```

```
[ [6, -4, -9, 5, -3], [-4, -1, 6, 2, -5],
 [7, -3, 2, 6, 4], [-1, 5, 10, -6, 9] ]
6    -4   -9   5    -3
-4   -1   6    2    -5
7    -3   2    6    4
-1   5    10   -6   9
[6, -4, -9, 5, -3]
[-4, -1, 6, 2, -5]
[7, -3, 2, 6, 4]
[-1, 5, 10, -6, 9]
veergude summad
[8, -3, 9, 7, 5]
Anna veerg, mille max 3
maksi veerus 3 6 number: 2
ridade summad
[-5, -2, 16, 17]
-5
-2
16
17
```

Failid

Failide tüübid ja struktuur

Faile kasutatakse erinevat liiki informatsiooni säilitamiseks arvutis. Käesolevas jaotises vaadeldakse **tekstifaile**, mida kasutatakse laialdaselt andmete salvestamiseks. Tekstifailidel võib olla erinev organisatsioon, kuid lihtsamal juhul kujutab tekstifail märkide jadast koosnevat jadafaili. Väga sageli kasutatakse ka **kirjefaile**. Selline tekstifail koosneb **kirjetest** ehk **ridadest**. Kirje koosneb ühest või mitmest **väljast**, mis on üksteisest eraldatud ühe või mitme tühiku või tabuleerimissümboliga (\t). Kirje lõpus on (mittenähtav) reavahetuse sümbol (\n), millega peab arvestama failide ja nende kirjete töötlemisel. Sellise struktuuriga faile kasutatakse tabelite salvestamiseks. Samuti saab seda kasutada ka tabelitöötlus-programmide andmebaasides. Tekstifailide töötlemine (lugemine ja kirjutamine) toimub tavaliselt kirjete kaupa.



Failide avamine ja sulgemine

Kaheks põhitegevuseks failidega on andmete **lugemine** ja **kirjutamine**. Enne lugemist või kirjutamist peab faili avama. Pythonis kasutatakse selleks lauset:

failiobjekt = **open**(*failinimi*, *töötlusviis*)

failiobjekt – objektimuutuja, *failinimi* – faili täisnimi: [tee]nimi.txt

Näiteks: palgad.txt või C:/raamatupidamine/palgaarvestus/palgad.txt

Kui fail on programmiga samas kaustas, ei ole teed vaja näidata. Tekstifailide üldlevinud failitüübiks on **.txt**, selle vaatamiseks ja ka koostamiseks on mugav kasutada üldlevinud tekstiredaktoreid, kuid põhimõtteliselt võib kasutada suvalist tüübitunnust.

Töötlusviisi põhivariantideks on:
"**w**" – kirjutamine (*write*) , "**r**" – lugemine (*read* – võetakse ka vaikimisi)

Faili avamisel luuakse nn **failiobjekt** (öeldakse ka failimuutuja), millel on rida meetodeid operatsioonide määramiseks: write(), read(), readline().

fail = **open**(*"puud.txt"*, *"w"*) – fail puud.txt avatakse kirjutamiseks ning failiobjekti nimeks on ***fail***.
fob = **open**(*"puud.txt"*) – fail puud.txt avatakse lugemiseks, failiobjekti nimeks on ***fob***.

Peale töötlemist peab faili sulgema meetodiga close(): failiobjekt.close().

Andmete kirjutamine faili

Andmete tekstifaili kirjutamise põhivariantideks on:

failiobjekt.**write**(*string*)
print([*avaldis*, ...], *sep* = ' ', *end* = *'\n'*, *file = failiobjekt*)

Esimesel juhul on tegemist failiobjekti meetodiga. Teisel juhul **print**-funktsiooniga, mida oleme siiani kasutanud andmete väljastamisel käsuaknasse. Ainukeseks erinevuseks on parameeter **file**, millele pöördumisel vastab failiobjekt. Väljastamisel ekraanile seda ei kasutada või esitatakse kujul file = sys.stdout.

```
fail_1 = open("ankeet.txt", "w")
tee_fail(fail_1,"Juku", 163, 54, 13)
fail_1.close()
```

Esimene lause avab faili nimega **ankeet.txt**. Kui sellise nimega fail on anud kaustas olemas, selle sisu kustutatakse. Allpool on toodud faili loomise protseduuri **tee_fail** kolm varianti.

```
def tee_fail(fm, nimi, L, m, v):
    fm.write("See on tähtis fail")
    fm.write("nimi: " + nimi)
    fm.write("pikkus: " + str(L))
    fm.write("mass: " + str(m))
```

Kuna meetodi **write** argumendiks peab olema üks string, siis peab selle moodustama stringavaldise abil elementidest.

Protseduuri täitmisel tekib ühest reast (kirjest) koosnev fail: See on tähtis failnimi: Jukupikkus: 163mass: 54

```
def tee_fail(fm, nimi, L, m, v):
    fm.write("See on tähtis fail" + "\n")
    fm.write("nimi: " + nimi + "\n")
    fm.write("pikkus: " + str(L)+ "\n")
    fm.write("mass: " + str(m)+ "\n")
```

Selleks et iga kirje läheks eraldi reale, peab stringi lõppu panema reavahetuse sümboli "\n".

Faili tekib neli eraldi ridadel asuvat kirjet.

```
See on tähtis fail
nimi: Juku
pikkus: 163
mass: 54
```

```
def tee_fail(fm, nimi, L, m, v):
    print("See on tähtis fail", file = fm)
    print("nimi: ", nimi, file = fm)
    print("pikkus: ", L, file = fm)
    print("mass: ", m, file = fm)
```

Selles variandis kasutatakse **print**-funktsiooni. Saadud faili struktuur on sama nagu eelmises variandis, kuid siin ei pea koostama stringavaldist, sest funktsioon **print** lubab kasutada mitut argumenti, mis ei pruugi olla stringid.

Ka reavahetus (kirje lõpp) lisandub automaatselt.

Üldiselt on **print**-funktsiooni kasutamine mõnevõrra mugavam ja lihtsam.

Järgnevalt on toodud näide funktsiooni **print** kasutamisest andmete kirjutamisel nii ekraanile kui ka faili. Kasutatakse ühte protseduuri, milles argumendi **file** väärtusega määratakse, kuhu andmed väljastatakse.

```
from math import *
import sys

def Fy(x):
    return 3 * sin(x / 2)+ 5 * cos(2 * x + 3)

def tee_tab(F, a, b, n, fm):
    ''' funktsiooni tabuleerimine ja
        väljastamine ekraanile või faili '''
    if fm == "":  fm = sys.stdout
    h = (b -a) / n
    for i in range(n+1):
        x = a + i * h
        y = F(x)
        print (round(x, 2), "\t",
               round(y, 4), file = fm)
''' funktsiooni tabuleerimise peaprotseduur '''
x0 = float(input("lõigu algus "))
xn = float(input("lõigu lõpp "))
n = int(input("jaotisi "))
kuhu = int(input("kuhu? 1-ekraan, 2-fail "))
if kuhu == 1:
    fm = ""
else:
    fm = open ("funtab.txt", "w")
tee_tab(Fy, x0, xn, n, fm)
if kuhu == 2: fm.close()
```

Moodul **sys** sisaldab vahendeid juurdepääsuks mõnedele süsteemisisestele funktsioonidele ja objektidele, mida kasutab peamiselt interpretaator (sh standardsisend ja -väljund).

Oluline roll on siin parameetril **fm**. Kui väljund toimub ekraanile, on vastava argumendi väärtuseks tühi tekst. Parameetrile **fm** vastab omakorda argument **file** funktsioonis **print**. Protseduuris kontrollitakse **fm** väärtust. Kui see on pöördumisel tühi, võetakse tema väärtuseks **stdout** (standardväljund) – väljund läheb ekraanile. Väljastamisel faili on vastavaks väärtuseks failiobjekt, kusjuures fail peab olema eelnevalt avatud lugemiseks.

Peaprotseduur loeb algandmete (x0, xn ja n) väärtused ja küsib, kuhu tahetakse suunata väljund – ekraanile või faili. Vastavalt sellele võetakse **fm** jaoks kas tühi väärtus või funktsiooniga **open** määratav failiobjekt.

Kui väljastamine toimus faili, sulgeb selle peaprotseduuri viimane lause.

Sageli toimub andmete kirjutamine faili loenditest või tabelitest. Järgnev näide demonstreerib andmete kirjutamist loenditest **nimed** ja **pikkused**, mis olid kasutusel näites **Meeskond**.

```
def faili(nimed, pikkused):
    n = len(nimed)
    fm = open("meeskond.txt", "w")
    for i in range(n):
        print(nimed[i], pikkused[i], file = fm)
    fm.close()

nimed = [ 'Haab', 'Kask', 'Mänd', 'Paju', 'Saar', 'Tamm' ]
pikkused  =  [ 193, 181, 178, 203, 197, 177 ]
faili (nimed, pikkused)
```

Loendid **nimed** ja **pikkused** on määratletud peaprotseduuris.

Funktsiooniga **open** avatakse meeskond.txt. Seda faili kasutatakse ka järgnevates näidetes andmete lugemisel.

Kirjutamisel kasutatakse funktsiooni **print**.

```
Haab 193
Kask 181
Mänd 178
Paju 203
Saar 197
Tamm 177
```

Andmete lugemine failist

Andmete lugemiseks failist saab kasutada mitmeid failiobjekti meetodeid ja ka muid vahendeid. Järgnevalt vaatleme mõnda varianti neist:

- faili lugemine korraga stringmuutujasse: string = failiobjekt.**read**()
- faili lugemine korraga loendisse: loend = failiobjekt.**readlines**()
- kirjete kaupa korduslauses: **for** kirje **in** failiobjekt: [tegevused kirjega]

Kogu faili lugemist kasutatakse tavaliselt väikeste ja lihtsa struktuuriga failide korral.

```
fob = open("ankeet.txt", "r")
string = fob.read()
fob.close()
print (string, type(string))
```

Loetakse terve fail (näites ankeet.txt) ja omistatakse see ühele stringmuutujale. Stringi kuuluvad ka mittenähtavad erisümbolid '\n', mille tõttu kuvatakse kirjed eraldi ridadel.

```
See on tähtis fail
nimi: Juku
pikkus: 163
mass: 54
 <class 'str'>
```

```
fob = open("ankeet.txt", "r")
loend = fob.readlines()
fob.close()
print (loend, type(loend))
```

Tervest loetud faili sisust moodustatakse loend, mis koosneb faili kirjete elementidest:

```
['See', 'on', 'tähtis', 'fail', 'nimi:', 'Juku', 'pikkus:', '163', 'mass:', '54']
<class 'list'>
```

Vaadeldava faili sisu saab näiteks kuvada. Enamasti toimub failide lugemine ja sellega kaasnev töötlemine kirjete kaupa. Siin võib eristada kolme põhivarianti:

- loetakse järjest kirjeid ja neis sisalduvate andmete põhjal leitakse kohe vajalikud suurused,
- loetakse järjest kirjeid ja neist moodustatakse üks või mitu lihtloendit ehk vektorit,
- loetakse järjest kirjeid ja neist moodustatakse liitloend ehk tabel.

```
def loe_fail1():
  fm=open("meeskond.txt", "r")
  S = 0; n = 0
  for kirje in fm:
     rida = kirje.split()
     S += eval(rida[1])
     n += 1
     print (rida)
  print ("kesk", S / n, 1)
  fm.close()

loe_fail1()
```

Esitatud protseduur loeb failist **meeskond.txt** kirjeid ja leiab nende alusel keskmise pikkuse. **for**-lauses omistatakse muutujale **kirje** järjest faili kirjeid. Meetod **split** moodustab sõnadest loendi **rida**, milles on antud juhul kaks elementi: **nimi** ja **pikkus**. Iga järgnev kirje asendab loendi **rida** eelmised väärtused.

Pikkuse väärtus (loendi teine element järjenumbriga 1) lisatakse summale **S** ja suurendatakse muutuja **n** väärtust. Väärtused loendist **rida** kuvatakse ekraanil.

```
['Haab', '193']
['Kask', '181']
['Mänd', '178']
['Paju', '203']
['Saar', '197']
['Tamm', '177']
kesk 188.2
```

Faili töötlemisel näidatud viisil jääb peale protseduuri täitmist mällu ainult viimane kirje.

```
def loe_fail2():
   ''' moodustatakse kaks loendit '''
   nimed = [ ]       # tühjad
   pikkused = [ ]    # loendid
   fm= open("meeskond.txt", "r")
   for kirje in fm:
     rida = kirje.split()
     nimed.append(rida[0])
     pikkused.append(int(rida[1]))
   print (nimed,pikkused )
   fm.close()
   nimed, pikkused = \
         lisa_kirjed(nimed, pikkused)
   print (nimed, pikkused)
   faili (nimed, pikkused)  # tagasi faili

def lisa_kirjed(nimed, pikkused):
   while True:
     kas = input("kas lisada j-jah, e-ei ")
     if kas == 'e': break
     nimi = input("anna nimi ")
     pikkus = eval(input("pikkus "))
     nimed.append(nimi)
     pikkused.append(pikkus)
   return nimed, pikkused

loe_fail2()
```

Antud protseduur loeb faili kirjete kaupa ja moodustab andmetest kaks loendit (vektorit): **nimed** ja **pikkused**. Peale seda võib kasutaja lisada mõned kirjed ja andmed kirjutatakse faili tagasi.

Igast kirjest moodustatakse kaheelemendiline loend **rida**, mille üks element lisatakse loendisse **nimed** ja teine loendisse **pikkused**. Loendid on sellised:

```
['Haab', 'Kask', 'Mänd', 'Paju', 'Saar', 'Tamm']
[193, 181, 178, 203, 197, 177]
```

Pöördumine protseduuri **lisa_kirjed** poole.

Loendid on peale lisamist ja enne kirjutamist tagasi faili järgmised:

```
['Haab', 'Kask', 'Mänd', 'Paju', 'Saar', 'Tamm', 'Kuusk', 'Mets']
[193, 181, 178, 203, 197, 177, 187, 205]
```

Protseduur pakub võimalust kirjete lisamiseks faili. Kui kasutaja vastab küsimusele "j" , küsitakse nime ja pikkust.

```
kas tahad lisada j-jah, e-ei j
anna nimi Kuusk
pikkus 187
kas tahad lisada j-jah, e-ei j
anna nimi Mets
pikkus 205
kas tahad lisada j-jah, e-ei e
```

```
def loe_fail3():
   ''' kirjetest moodustatakse tabel '''
   T = [ ]  # tühi loend(tabel)
   fm = open("meeskond.txt", "r")
   for kirje in fm:
     rida = kirje.split()   # kirje loendiks
     T.append(rida)       # tabelisse
   fm.close()
   print (T)
   kuva_tab(T)
```

Igast kirjest luuakse loend **rida** ja see lisatakse loendisse **T**. Tulemusena tekib tabelit esindav loendite loend.

```
[['Haab', 193], ['Kask', 181], ['Mänd', 178],
['Paju', 203], ['Saar', 197], ['Tamm', 177]]
Haab 193
Kask 181
Mänd 178
Paju 203
Saar 197
Tamm 177
keskmine .  188.17
```

```
def kuva_tab(T):
  ''' tabeli kuvamine '''
  m = len(T); n = len(T[0])
  for i in range(m):
    for j in range(n):
      print (T[i][j], end = ' ')
    print()

loe_fail3()
```

Protseduur kuvab loendi tabelina.