Last name ______________________

First name ______________________

LARSON—MATH 756–SAGE WORKSHEET 06
The CONJECTURING Program.

1. Login to your Sage/Cocalc account.
   - (a) Start the Chrome browser.
   - (b) Go to `http://cocalc.com`
   - (c) Login. You created a new Project for our class. Click on that.
   - (d) Click "New", then "Worksheets", then call it **s06**.

   Organizing Your Functions

   As we go you can (should) cut and paste any needed relevant definitions from previous worksheets. One useful thing you can do is put all of your functions together is a `.sage` file and load them. Then you will always know what definitions you have and that they are actually in memory at that time (the variables in Sage worksheets lose their state after some amount of time if they are not being used).

   In **s05** we created a file "independence.sage" as a location to place all of the functions we have defined (and we divided that file up into graphs, invariants, properties, and auxiliary functions). My copy is in our shared project and can always be copied too.

2. Run `load("independence.sage")` and then evaluate `pete.show()` and `independence_number(pete)` to see that the loaded functions are working.

   Setting Up the Conjecturing Program

   We will use the CONJECTURING program to make various kinds of graph theoretic conjectures. Setting up the program involved downloading a file to your CoCalc project, unzipping it, and running `make`. The CoCalc set-up directions at:
   `math1um.github.io/objects-invariants-properties/#install-conjecturing-2`.
   When you are successfully set-up you will see files "conjecturing.py" and "expressions" (a compiled C binary file) in your CoCalc project.

   Upper Bound Conjectures

   The CONJECTURING program takes as inputs a list of graphs, a list of graph invariants, the index of the graph invariant to investigate (produce conjectured bounds for), and whether the user wants upper or lower bounds.

3. Here is a first example for producing upper bounds for the independence number of a graph. The conjectures are guaranteed to be true for all input graphs.

```
objects = [pete,k_2_3]

invariants = [independence_number, matching_number,
Graph.order, Graph.size, Graph.radius, Graph.diameter]

investigate = invariants.index(independence_number)

for conj in conjecture(objects, invariants, investigate, upperBound =True):
    print conj
```

4. For each conjecture, either it is true or there is a counterexample. Can you prove or find a counterexample for either conjecture?

5. Find a counterexample to one of these conjectures, code it up, give it a name, add it to the list of `objects` and rerun the code block. Notice that the falsified conjecture is no longer produced.

6. Now try the following code block. We will add more invariants in the hopes of getting better conjectures.

```
objects = [pete,k_2_3]

invariants = [independence_number, matching_number,
fractional_independence_number, critical_independence_number,
Graph.order, Graph.size, Graph.radius, Graph.diameter,
Graph.lovasz_theta]

investigate = invariants.index(independence_number)

for conj in conjecture(objects, invariants, investigate, upperBound =True):
    print conj
```

   This time some of the conjectures are true. We can keep the program from producing these by adding them as `theory`. The produced conjectures must be better for at least one input graph object than the existing `theory`—so the program can't reproduce these true conjectures.

7. Evaluate:

```
objects = [pete,k_2_3]

invariants = [independence_number, matching_number,
fractional_independence_number, critical_independence_number,
Graph.order, Graph.size, Graph.radius, Graph.diameter, Graph.lovasz_theta]

investigate = invariants.index(independence_number)

theorems = [fractional_independence_number, Graph.lovasz_theta]

conjs = conjecture(objects, invariants,investigate, upperBound = True,
theory = theorems):
for conj in conjs:
    print conj
```

   Hmmm... What happened?

8. Rerun the last line, turning on the `debug` parameter:

```
conjecture(objects, invariants,investigate, upperBound = True,
theory = theorems, debug = True)
```

   So to get interesting conjectures we need some example of a graph g where the independence number does not equal `min(fractional_independence_number(g), g.lovasz_theta())`. Can you think of one on your own?

   We can also let the computer search for an example. If there is no result for graphs on 5 vertices, keep trying for graphs with 6, 7, 8... vertices.

9. Evaluate:

```
alpha_f = fractional_independence_number
for g in graphs.nauty_geng("5 -c"):
    if independence_number(g) != min((alpha_f(g),g.lovasz_theta()):
        print g.graph6_string()
```

10. What graph did you just produce?

11. Now add this graph to your list of graph `objects` and rerun the CONJECTURING program. What do you get?

12. How can you proceed?

Lower Bound Conjectures

We can also generate lower bound conjectures for any invariant: just turn the `upperBound` parameter to `False`.

13. Evaluate:

```
objects = [pete,k_2_3]

invariants = [independence_number, matching_number,  Graph.order,
Graph.size, Graph.radius, Graph.diameter]

investigate = invariants.index(independence_number)

for conj in conjecture(objects, invariants, investigate, upperBound = False):
    print conj
```

14. Find proofs or counterexamples.

15. We can continue this investigation in the same way: add graphs that are counterexamples, search for counterexamples, add theorems, and rerun the program. Another thing we can do is add invariants.

16. CONJECTURE^{TM} and Play! See if you can generate a non-trivial conjecture.