Last name _____

First name _____

LARSON—MATH 353–CLASSROOM WORKSHEET 09
Conjecturing in CoCalc/SAGE.

1. Sign in to your CoCalc account.
   (a) Start the Chrome browser.
   (b) Go to `https://cocalc.com`
   (c) Log in to your account.
   (d) You should see an existing Project for our class. Click on that.
   (e) Make sure you are in your Home directory (if you put files in the Handouts directory they could be overwritten.)
   (f) Click "New", then "Jupyter Notebook", then call it **353-c09**.
   (g) Make sure you have SAGE as the *kernel*.

Review: the Ring of Integers Mod n

   (a) We can define the *ring of integers mod* 3 ($\mathbb{Z}/3\mathbb{Z}$) in Sage with `R = Integers(3)`. To see what they look like, try `list(R)`.
   (b) 1 in every ring is different. In $\mathbb{Z}/3\mathbb{Z}$, $1 + 1 + 1 = 0$. To see this in Sage, we need to tell it *which* "1" we mean. Try $a = R(1)$ and then evaluate $a + a + a$.
   (c) What is the *multiplicative order* of an element in $\mathbb{Z}/10\mathbb{Z}$?
   (d) What is Euler's ϕ function?

We will set up the conjecturing program by following the steps at:
`http://nvcleemp.github.io/conjecturing/`

2. The `Conjecturing` program .zip file is in your 353-handouts directory.
3. *Move* the program to your `Home` directory.
4. Click "New" and then "Terminal" to get a Sage terminal window.
5.
```
~$ unzip conjecturing-0.13.zip
~$ cd conjecturing-0.13
~/conjecturing-0.13$ make
```
6. Look in your `Home` directory. You should see a `conjectures.py` file and an `expressions` file.
7. Switch back to your `353-c09.ipynb` Jupyter notebook and test your installation. Run: `load("conjecturing.py"` and then `conjectures()` with no inputs. The interpreter will either complain that there's no procedure with that name (bad) or that the procedure needs inputs (good).

8. Code and test each of the following procedures.

```
def digits10(n):
    return len(n.digits(10))

def digits2(n):
    return len(n.digits(2))

def count_divisors(n):
    return len(divisors(n))

def count_prime_divisors(n):
    return len(factor(n))

def number(n):
    return n
```

9. Here is a minimal test for generating upper-bound conjectures for an integer n.

```
invariants = [number, digits10, digits2]
objects = [2, 3, 10]
invariant_of_interest = invariants.index(number)
conjectures = conjecture(objects, invariants, invariant_of_interest,
   upperBound = True, debug = True)
```

   If a conjecture is true, the only way to be certain is to *prove* it. If it is false, the only way to be certain of that is to find an example that demonstrates falsity (a *counterexample*).

10. What conjectures do you get? (Are they true? If not find a counterexample and add it to Sage. Then re-run to get new conjectures.)

11. We can get better conjectures by adding more integer invariants. What can we add?

Facts about the Produced Conjectures

(1) **Truth**. They are TRUE for every input object.
(2) **Significance**. Each conjecture, when added to the list of conjectures, was "better" for at least one input object than any previously stored conjecture.

Property Conjectures
For integers (or any other object-type) we can conjectures necessary or sufficient conditions for an integer to have that property.

```
properties = [is_prime, is_even]
property_of_interest = properties.index(is_prime)
objects = [3]
propertyBasedConjecture(objects, properties, property_of_interest)
```

12. What conjectures do you get? (Are they true? If not find a counterexample and add it to Sage. Then re-run to get new conjectures.)

Getting your classwork recorded

When you are done, before you leave class...

(a) Click the "Print" menu choice (under "File") and make a pdf of this worksheet (html is OK too).

(b) Send me an email (`clarson@vcu.edu`) with an informative header like "Math 353 - c09 worksheet attached" (so that it will be properly recorded).

(c) Remember to attach today's classroom worksheet!