Last name _____

First name _____

LARSON—MATH 750–SAGE WORKSHEET 01
Getting Started with Sage/Colcalc.

1. Create a Sage/Cocalc account.
   (a) Start the Chrome browser.
   (b) Go to `http://cocalc.com`
   (c) Login. You should see an existing Project for our class. Click on that.
   (d) If you don't see a Project, ask me and we'll get the right email address associated with your account.
   (e) Click "New", then "Worksheets", then call it **s01**.

Taylor polynomials

We're going to look at the behavior of the Taylor polynomials at 0 for the sine function.

2. First let's plot the sine function (on a somewhat arbitrary interval). Run (evaluate): `plot(sin(x),(x,-1,10)`.
3. Now let's let f be a shorter name for our sine function. Run: `f(x)=sin(x)`. Check `f(0)` and `f(pi/2)`, etc, to see what you get (and that it agrees with what you expect).
4. Now let's calculate the first few Taylor polynomials. Run `f.taylor(x,0,1)`, then `f.taylor(x,0,2)`, then `f.taylor(x,0,3)`.
5. Now let's give these functions names and graph them along with the sine. Run: `f1=f.taylor(x,0,1)`. This assigns the name $f1$ to the first Taylor polynomial. Then run: `plot(sin(x),(x,-1,10),ymax=2)+plot(f1(x),(x,-1,10),ymax=2,color="green")`.
6. Do the same thing for the second and third Taylor polynomials.
7. Now let's put it all together. Define the 10^th Taylor polynomial, give it a name and plot it on the same coordinate system as sine.
8. Repeat with the 50^{th} Taylor polynomial. (Wow!)
9. Now let's see how good these approximations are. Let's find $\sin(47°)$. Sage doesn't know degrees—its trig function calculations assume radians. What will you evaluate?
10. Now find the values of the Taylor polynomials you defined for 47°. How good are these approximations?

Sets in Sage

11. One way to generate the power set of the set $\{1, 2, 3, 4\}$ is to run: **Subsets([1,2,3,4])**. Hmmm, the output is opaque.

12. Lets give this collection a name $S4$ and run through the collection with a *for* loop and have it print out the elements:

```
S4=Subsets([1,2,3,4])
for x in S4:
    print x
```

13. Sets in Sage have a built-in subset relation. Let's define a couple of sets and check. `Set` is Sage's set *constructor*. Run: `S=Set([3,4])`. This defines S. Run: `S` to see what that set is. Now define a set T which will be the set $\{3, 4, 5\}$.

14. Many common mathematical objects are built-in to Sage and have common *methods* built-in. Testing if one set is a subset of another is a built-in Set method. Run: `S.issubset(T)`. Now check if T is a subset of S.

15. Now we have a family of sets $S4$ and a way to test for inclusion. We can now write our own functions to check the three poset properties. Run:

```
def reflexive(X):
    for x in X:
        if not x.issubset(x):
            return False
    return True
```

Then run: `reflexive(S4)` to check that our family of sets is reflexive.

16. Check if $S4$ (with inclusion) is anti-symmetric.

```
def anti_symmetric(X):
    for x in X:
        for y in X:
            if x.issubset(y) and y.issubset(x) and not x == y:
                return False
    return True
```

17. Can you write a function to check if a family of sets X is transitive? Make sure it works on $S4$

Graphs in Sage

Sage includes the `graphs` class which contains a number of *methods*. Some of these include constructors for making well-known graphs. "pete" is an arbitrarily chosen name in what follows. We could use G or anything else instead!

18. Evaluate:

```
pete=graphs.PetersenGraph()
pete.show()
```

Here is a list of common graphs that have pre-built constructors: `http://doc.sagemath.org/html/en/reference/graphs/sage/graphs/graph_generators.html`

Find another one of these that interests you and `show` it.