Last name ______________________

First name ______________________

LARSON—MATH 255–CLASSROOM WORKSHEET 10
Scatter Plots & Recursion.

1. Log in to your Sage/Cocalc account.
   (a) Start the Chrome browser.
   (b) Go to `http://cocalc.com` and sign in.
   (c) You should see an existing Project for our class. Click on that.
   (d) Click "New", call it **c10**, then click "Sage Worksheet".
   (e) For each problem number, label it in the Sage cell where the work is. So for Problem 1, the first line of the cell should be `#Problem 1`.

   Step Functions and Scatter Plots

   Given a list L of pairs (x, y) you can plot the *step function* that holds y constant from one x to the next with `plot_step_function(L)`.

2. Try `plot_step_function([(x,x) for x in [3..9]])`
3. Try `plot_step_function([(i,sin(i)) for i in [5..20]])`
4. Try `plot_step_function([(i*.2,sin(i*.2)) for i in [5..100]])`

   Given a list L of pairs (x, y) you can plot the *scatter plot* that consists just of those points with `scatter_plot(L)`.

5. Try `scatter_plot([(0,1),(2,4),(3.2,6)])`
6. Try `scatter_plot([(x,x) for x in [5..20]])`
7. Try `scatter_plot([(x,x**2) for x in [-5..5]])`
8. Try `scatter_plot([(i*.2,sin(i*.2)) for i in [5..100]])`
9. Define a function `points(x)` that plots all the points (1,2), (2,3), . . . (x,x+1). Use `scatter_plot()`.

   Recursion

   A **recursive** function is a function that calls itself. It must always have a *base case* so that the recursion eventually stops.

10. Here is an example of a recursive definition of the *factorial* function. The base case here is the case where the input is 0 or 1.

```
def facto1(n):
    if n==0 or n==1:
        return 1
    else:
        return n*facto1(n-1)
```

11. Now try `facto1(0)`, `facto1(1)`, `facto1(2)`, `facto1(3)`, and `facto1(10)`.

12. It is often intuitive to define a function recursively, but usually the same function can be defined without recursion. Here is a function `facto2(n)` that does the same thing as `factorial(x)` but is **not** recursive. Test it to make sure it gives the same results.

```
def facto2(n):
    result=1
    if n==0:
        return result
    for i in [1..n]:
        result=result*i
    return result
```

Try `facto2(0)`, `facto2(1)`, `facto2(2)`, `facto2(3)`, and `facto2(10)`.

13. Write a function facto3(x) that prints x, and returns 1 if x=1 else returns x*facto3(x-1). Test it!

14. The *gcd* of 2 non-negative integers is their *greatest common divisor*. The following recursive function calculates the gcd of integers a and b using the fact (which can be proved) that, if $a \geq b$ then $\gcd(a, b) = \gcd(a - b, b)$. It uses the fact that $\gcd(0, a) = \gcd(a, 0) = a$, for any non-negative integer a, as the base case.

```
def gcd(a,b):
  if a==0 or b==0:
      return max(a,b)
  else:
      return gcd(max(a,b)-min(a,b),min(a,b))
```

Try $\gcd(0, 5)$, $\gcd(2, 5)$, $\gcd(5, 5)$, $\gcd(10, 5)$, $\gcd(50, 51)$, $\gcd(50, 55)$, and $\gcd(1234, 5678)$.

15. The `gcd()` function does not actually test that the input numbers are non-negative. Add a test to your code, so that if either a or b is negative, the program prints an error message.

16. **Investigate**. Start with any positive integer x. If x is even divide by 2. If x is odd, multiply by 3 and add 1. Repeat. Try this for several initial starting numbers x. What happens?

    Can you write code to continue this investigation (and see if the pattern persists)?