LARSON—MATH 353–CLASSROOM WORKSHEET 11
Recursion—Density of Primes.

1. Log in to CoCalc.
   (a) Start the Chrome browser.
   (b) Go to `https://cocalc.com`
   (c) Login (**your VCU email address** is probably your username).
   (d) You should see an existing Project for our class. Click on that.
   (e) Click "New", then "Worksheets", then call it **c11**.

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

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

11. It is often intuitive to define a function recursively, but usually the same function can be defined without recursion. Here is a function `factorial2(n)` that does the same thing as `factorial(x)` but is **not** recursive. Test it to make sure it gives the same results.

```
def factorial2(n):
    result=1
    if n==0:
        return result
    for i in [1..n]:
        result=result*i
    return result
```

12. 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 $\gcd(1234, 5678)$.

Primes

13. Write a function density_of_primes(n) that **returns the number of primes** in the integers up to n, divided by n. Test it.

14. Use `scatter_plot` to *visualize* the density of the primes as values of n get larger and larger.

15. **Getting your classwork recorded**

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

(a) Click the "Make pdf" (Adobe symbol) icon and make a pdf of this worksheet. (If Cocalc hangs, click the printer icon, then "Open", then print or make a pdf using your browser).

(b) Send me an email with an informative header like "Math 353—c11 worksheet attached" (so that it will be properly recorded).

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