LARSON—MATH 353–CLASSROOM WORKSHEET 13
More 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 **c13**.

   Primes & Gauss

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

3. Use `scatter_plot` to *visualize* the density of the primes as values of n get larger and larger (that is, the fraction of the numbers in [1..n] that are prime).

   What we see is that the density appears to go down as n increases. Gauss conjectured that $f(n) = \frac{1}{log(n)}$ "approximates" the density of primes. We'll test that function and also try to put some meat on what an approximation should be.

4. $f(x) = \frac{1}{log(x)}$ is a decreasing function. Is it a good approximation? Plot it on the same graph as your graph of the density of primes. Try:
   ```
   scatter_plot([(x,density_of_primes(x)) for x in [2..100]]) +
   scatter_plot([(x,1/log(x)) for x in [2..100]], facecolor="red")
   ```

5. Now try to increase the range of values from [2..100] to [2..1000] or [2..10000] or try other intervals like [100000..101000]. Does that help?

6. Maybe the difference (or the absolute value of the difference) between these functions, `density_of_primes(n)-1/log(n)` decreases as n increases? Investigate that.

7. What else might it mean for $\frac{1}{log(n)}$ to approximate the density of the primes up to n?

Recursion

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

8. The Fibonacci sequence F_n is defined as follows $F_0 = 0$, $F_1 = 1$ and $F_n = F_{n-1} + F_{n-2}$ for $n > 1$. Here is a recursive function `fibonacci(n)` that computes the nth Fibonacci number.

```
def fibonacci(n):
    if n==0:
        return 0
    if n==1:
        return 1
    else:
        return fibonacci(n-1)+fibonacci(n-2)
```

   Test your function to make sure that it works.

9. Evaluate what you get for `timeit("fibonacci(10)")`, `timeit("fibonacci(20)")` , and `timeit("fibonacci(25)")` .

10. Define a non-recursive (iterative) function `fibonacci2(n)` that computes the n^{th} Fibonacci number.

11. Evaluate what you get for `timeit("fibonacci2(10)")`, `timeit("fibonacci2(20)")` , and `timeit("fibonacci2(25)")` .

    The recursive `fibonacci(n)` function we defined takes a very long time to respond for $n = 30$ and may never respond for $n = 40$. Now try `fibonacci2(40)` and `fibonacci2(400)`. Why does the iterative function work while the recursive function does not?

12. Solve the equation $\frac{a+b}{a} = \frac{a}{b}$, for a and b. Find $\frac{a}{b}$. Get a 10-digit approximation for this quantity (this is the Golden Ratio).

13. Define a function `fib_ratio(n)` which returns the ratio of the $(n+1)^{th}$ Fibonacci number to the n^{th}. find `fib_ratio(10)` and `fib_ratio(100)`. Compare this answer to your previous answer. What can you conjecture?

14. **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—c13 worksheet attached" (so that it will be properly recorded).

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