CMPS 6100 Lab 01

In this lab, we will practice programming fundamentals by exploring an endlessly fascinating topic: prime numbers.

Some prompts will require you to edit `main.py` and others will require answers to go in `answers.md`.

Refer back to the README.md for instruction on git, how to test your code, and how to submit properly to get all the points you've earned.

Generating Prime Numbers

Our first goal is to implement a prime number generator. Given an upper bound, our program will print all prime numbers less than that bound and count how many of them there are.

We will implement this in a piecewise fashion, building up to the generator.

As you know, a prime number is an integer which is only divisble by 1 and itself.

1. In `main.py`, implement the function `is_divisible_by` which takes in two parameters `num` and `i` and returns `True` if `num` is divisible by `i` and `False` otherwise. **(3 pts)**
2. Test that your function is correct by calling from the command-line `pytest main.py::test_is_divisible_by`
3. Implement a function `is_prime` which takes in as a parameter a number to test, and returns `False` if that number is divisible by any number in the range of 2 to the square root of the number, and `True` otherwise. **(5 pts)**
4. Test that your function is correct by calling from the command-line `pytest main.py::test_is_prime`
5. Now that you have `is_prime` implemented, implement a function `generate_primes` which takes in as a parameter an upper bound and prints out all primes less than the upper bound.
6. Now implement a function `count_primes`. This is very similar to `generate_primes` except that rather than printing the primes, it counts them. `count_primes` takes in as a parameter and upper bound and returns the number of primes less than that bound. **(7 pts)**
7. Test that your function is correct by calling from the command-line `pytest main.py::test_count_primes`

Twin Primes

Twin primes are pairs of primes that differ only by two. For example, (11,13), (17,19), and (29,31) are all pairs of twin primes. It is known that there are an infinite number of primes, but it is not known whether there are an inifite number of twin primes, even after millenia of thought.

In this part of the lab, we will examine twin primes.

Let's start by implementing a twin-prime generator.

There is a fact about prime numbers which you can exploit to simplify your implementation.

> All prime numbers other than 2 and 3 are one more or less than a multiple of 6.

Don't believe me, look at the primes! Between 11 and 13 is twelve, 17 and 19: 18. 23 is one less than 24. And so on!

Why is this true? Because all other numbers are divisible by two or three (or both).

This fact suggests an algorithm to generate twin primes: For all multiples of 6, check to see if both $6n - 1$ and $6n + 1$ are prime. If so, they are a twin prime pair.

That said, don't forget the twin prime pair (3, 5) which is the exception to the above.

8. Implement a function `generate_twin_primes` which takes in as a parameter an upper bound and prints out all pairs of twin primes less than that bound. A pair should be printed only if both its primes are less than the bound.
9. As we did for basic prime numbers, now lets count them. Implement a function `count_twin_primes` which takes in as a parameter an upper bound and returns the number of pairs of twin primes less than that bound. A pair should only be included if both its primes are less than the bound. **(7 pts)**
10. Test that your function is correct by calling from the command-line `pytest main.py::test_count_twin_primes`
11. Now try to generate large pairs of twin primes. What is largest pair of twin primes that you were able to generate, and how long did it take? You can benchmark your code using python's `time` module:

    ```
    import time

    start = time.time()
    generate_twin_primes(10)
    end = time.time()

    elasped_time_ms = (end - start) * 1000
    print("Elapsed Time: {:.2f} milliseconds".format(elasped_time_ms))
    ```

    Enter your answer in answers.md. (3 pts)

Wrapping Up

Before pushing to github, comment out all code and function calls that you make outside of the defined functions. Otherwise, they may break the GitHub tests. Benchmarking your twin prim generation is especially problematic as you will cause the testing process to time out.

Epilogue

A major topic of interest in number theory are prime and twin prime counting functions. While it is beyond of the scope of this lab and course - this isn't a number theory course after all - with your functions, you could reproduce, verify, and explore some of the known (and unknown) patterns and relationships.

For our purposes, prime numbers were a simple, yet fascinating, domain in which to practice our programming skills. If you like these types of problems, check out Project Euler. The problems in this lab didn't come from Project Euler, but it wouldn't be surprising to find them there.

Lastly, if you found this lab fascinating or are interested in Math in general, you may enjoy the YouTube channels 3Blue1Brown and Numberphile.