

CMPSC 102
Discrete Structures
Fall 2018

Lab 1 Assignment:
Prime Number Computation Using Python

Submit deliverables through your assignment GitHub repository bearing your name. Place source code in src/ and output in output/ directories

Objectives

To enhance the understanding of Python by completing a program which tests for prime numbers by opening a text file to extract numbers for primality tests. The output of this program is a count of primes and composites from the text file.

GitHub Starter Link

<https://classroom.github.com/a/6XNF7KYp>

To use this link, please follow the steps below.

- Click on the link and accept the assignment
- Once the importing task has completed, click on the created assignment link which will take you to your newly created github repository for this lab,
- Clone this repository (bearing your name) and work locally
- As you are working on your lab, you are to commit and push regularly. The commands are the following.

```
- git commit <nameOfFile> -m ‘‘Your notes about commit here’’  
- git push
```

Reading Assignment

Please read Chapters 1, 2 in the course book and the second week’s slides.

Additional Materials

Please locate your `sandbox` directory for this lab where you will find two files:

- A partially completed python source code; `isPrimeCounter.py`.
- A text file containing values to check for primality; `dataFile.txt`

About Prime Numbers

A prime number is a whole number greater than 1 whose only factors are 1 and itself. The number, 2, is the only even number which is also prime. A factor is a whole number (integer) that can be evenly divided into another number with no remainder. For instance, the number 6 is not prime because it can be evenly divided by the factors; 2 and 3 without a remainder. Mathematically speaking, we say, $2 \mid 6$, *two divides six* and $3 \mid 6$, *three divides six*. Because there are numbers; 2 or 3 that can be used to divide into six without a remainder, 6 cannot be a prime number. Instead, 6 is said to be a *composite* number. Numbers that have more than two factors are called composite numbers. The number 1 is neither prime nor composite.

The first few prime numbers are 2, 3, 5, 7, 11, 13, 17, 19, 23 and 29. Shown in Figures 1, 2 and 3, the occurrence of these primes appears to form specific types of patterns in the real number system. An important use of primes in computer science is that they are hard to find. There is no equation (that we know of) to directly test for primality of numbers. Instead, all potential divisors, less than the prime number, must be manually tested to verify that there never a remaining value after division. This type of test has been used to determine performance in hardware and algorithms. In addition, the determination of prime numbers is a process which can be done using simple mathematical approaches using a machine. The measurement of the task is made by ascertaining how much time has elapsed between the start and the end of the test and can be used to classify machines in performance groups.

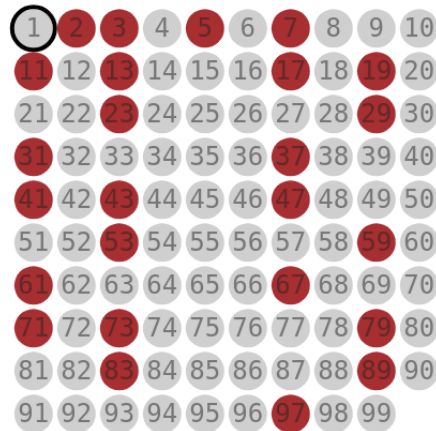


Figure 1: Prime numbers highlighted and formatted by grid. Image available from: <http://labs.minutelabs.io/Number-Constellations/>

Lab Work

The test for primality is to ascertain whether there exists *any* divisor for a particular number. If there are *no* divisors found for a test value, then one may conclude that the it is prime, otherwise the test value has to be composite.

In this lab, you are going to complete the given python code (see below) by adding the if

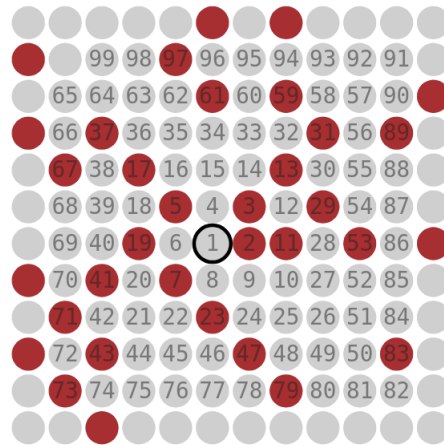


Figure 2: Prime numbers highlighted and formatted in an Ulam Spiral. Image available from: <http://labs.minutelabs.io/Number-Constellations/>

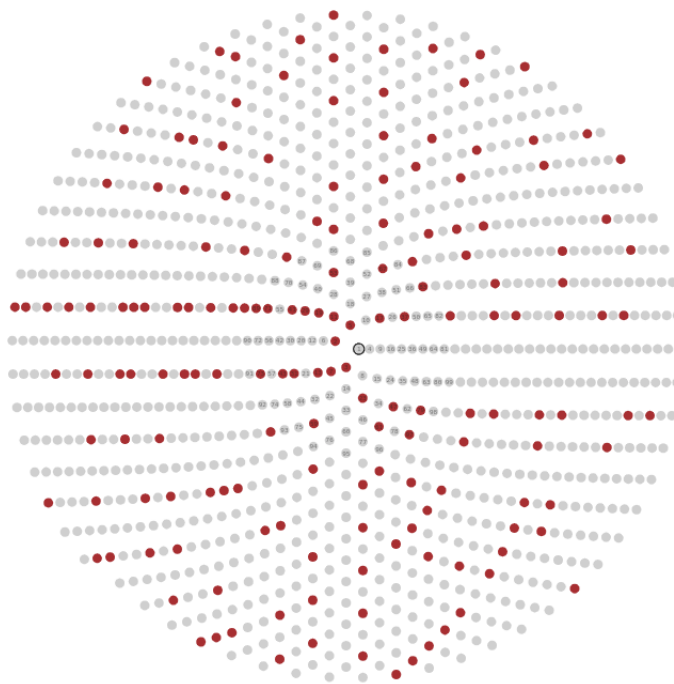


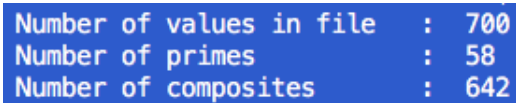
Figure 3: Prime numbers highlighted and formatted in a Sacks Spiral. Image available from: <http://labs.minutelabs.io/Number-Constellations/>

statements used for the test. You will use your completed code to test the primality of the 700 arbitrarily chosen numbers in the accompanying text file (`dataFile.txt`) which is composed of one test value per line.

Your completed program is to proceed line by line to test each test values for primality. When a prime number is found, it will be counted, as shown in Figure 4. Once the program has terminated (i.e., completed a test for all the numbers of the text file), your program will print to the screen the following information.

- The total number of values that was tested: in this case, the number will be 700 (i.e., the number of test values in the text file.)
- The number of discovered primes.
- The number of composite numbers.

For each of these outputs, you will notice that there is already a print statement in the code that you are to use. Your output statements should look like those of Figure 4. Remember, the number of values in the file is equal to the summation of the primes and composites (combined.)



```
Number of values in file : 700
Number of primes       : 58
Number of composites   : 642
```

Figure 4: The output of your code has to resemble the following. Note: after a prime number from the text file has been discovered, it is printed to the screen. At the end of the program, the number of primes and composites from the list, as shown. The values in the file should be the addition of the number of primes and the composites. Your tallies will be different from those featured here.

During the execution of your program, you are to tally the number of discovered primes and composites and report them at the termination of the program to the user. Your output will be exactly look like that of 4 where the number of primes and composites from the list are displayed.

Partial Code

```
#!/usr/bin/env python3

# original author: obonhamcarter@allegheny.edu
# your name:
# date 5 Aug 2018

data = open("dataFile.txt")
numOfPrimes_int = 0
numOfNonPrimes_int = 0
numOfNums_int = 0

for line_str in data:
    numOfNums_int += 1
```

```

try: # if the line contains something other than an integer
    n_int = int(line_str)
except ValueError: # catch the non-integer
    pass

print("    < Your testing algorithm goes here. Remember to use the variables >")
print("    < which are used at the end of the code to describe how many >")
print("    < primes and composites there are >")

print("    Number of values in file      : ",numOfNums_int)
print("    Number of primes                : ",numOfPrimes_int)
print("    Number of composites            : ",numOfNonPrimes_int)

```

How Your Algorithm Should Work

In order to test for primality for x , in the set of numbers $\{2, \dots, (x-1), x\}$ are to be tested as potential divisors. In other words, we shall determine whether there is *any* number between 2 and $x-1$ which divides evenly into x (and has no remainder.) For a number to be prime, then no divisor will exist to have a zero remainder (i.e., no number will divide evenly into the number). The steps of this operation are shown below for the test of divisors for 8, and a list of potential divisors, $n \in \{7, 6, 5, 4, 3, 2\}$. Note that we can exclude the x (itself) and 1 for the test.

- test $n = 7$; $8\%7 = 1$
 - test $n = 6$; $8\%6 = 2$
 - test $n = 5$; $8\%5 = 3$
 - test $n = 4$; $8\%4 = 0$
 - Stop; 4 is a factor of 8 and there is no remainder.
- Conclusion: this number is not prime.

Note: when we have a zero-remainder for a particular division, we know that we have found a factor of x and so we can immediately conclude that x is *not* prime and move on with the next number to check for primality.

Required Deliverables

Submit deliverables through your assignment GitHub repository bearing your name. Place source code in **src/** and the output text file **output/** directories.

1. Your completed and working python code (containing comments for your algorithm) which successfully counts the number of primes and composites from the text file.

2. A text file (called “results.txt”) containing the exact output of your program – your counts of primes and composites will be different from those of Figure 4 but should be outputted to the screen in a similar format.