CSE 20
Beginning Programming in Python
Programming Assignment 5

In this assignment you will write a Python program that reads a positive integer n from user input, then prints out all of the prime numbers that are less than or equal to n. We begin by reviewing a few topics from arithmetic. An integer m is said to be *divisible* by another (non-zero) integer d if and only if there exists an integer k such that $m = kd$. Equivalently, m is divisible by d if and only if the remainder of m upon (integer) division by d is zero. In this case we say that d is a *divisor* of m. Every positive integer m is a divisor of itself since $m = 1 \cdot m$, and for the same reason, 1 is a divisor of every positive integer. An integer $p > 1$ is called *prime* if its only positive divisors are 1 and p. Note that 1 itself is not considered to be prime. A positive integer that is not prime is called *composite*. Euclid showed that there are infinitely many prime numbers. Since every even number other than 2 is composite (being divisible by 2), there are also infinitely many composite numbers. The prime and composite sequences begin as follows.

Primes: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, …

Composites: 1, 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 22, 24, 25, 26, 27, 28, …

There are many ways to find prime numbers. In this project you will use a method called the *Sieve of Eratosthenes* to find all primes up to a given magnitude. Let us suppose that we wish to find all the primes that are less than or equal to 50. First list all the numbers from 1 to 50.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

We have colored 1 red, signifying that it is not prime. The next unshaded number must be prime. We therefore mark 2 as prime by coloring it green, and shade each of its multiples red since they, being divisible by 2, cannot be prime.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

Again, the next unshaded number must be prime (since it is not a multiple of 2). We shade 3 green, and all of its multiples red. Notice that half of these multiples are already red, since they are also multiples of 2.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

The next unshaded number, which is 5, must be prime (since it is not a multiple of 2 or 3). We color 5 green, and all of its multiples red. Note that most of them have already been eliminated. In fact, the first multiple of 5 not already shaded red is 25.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

Continuing in the same manner, the next unshaded number must be prime (since it is not a multiple of 2, 3 or 5). Accordingly, we color 7 green and eliminate all of its multiples by coloring them red. Notice the only such multiple less than 50 which is not already red is 49.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

Now consider the next unshaded number, which is 11, and must be prime (since it is not a multiple of 1, 2, 5 or 7). But notice that all of its multiples less than 50 have already been eliminated, so it's not necessary to color any multiple of 11 red. Why is this? If there were an unshaded multiple of 11, then the "other factor" in that multiple could not be 2, 3, 5 or 7, since they have already been eliminated. The smallest such multiple of 11 is $11^2 = 121$. But 121 is larger than our upper limit 50. Likewise, any unshaded multiple of 13 would have to be at least $13^2 = 169$, also too big. The same goes for the remaining unshaded numbers. Therefore, we can halt the process now, and color everything else green.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

The set of prime numbers that are less than or equal to 50 is therefore

$$\text{Primes} = \{2,3,5,7,11,13,17,19,23,29,31,37,41,43,47\},$$

exactly those numbers shaded greed. Those shaded red are the composite numbers in the range 1 to 50.

$$\begin{aligned}\text{Composites} = \{&1,4,6,8,9,10,12,14,15,16,18,20,21,22,24,25,26,27,\\ &28,30,32,33,34,35,36,38,39,40,42,44,45,46,48,49,50\}.\end{aligned}$$

In general, suppose we wish to find all primes in the range $1 \cdots n$. We eliminate 1, mark 2 as prime, and eliminate all multiples of 2 that are less than or equal to n. We repeat the steps as above: move to the next open candidate, mark it as prime, and mark its multiples that are less than or equal to N as composite. We stop when we reach a prime p satisfying $p^2 > n$, since all multiples of p less than p^2 have already been eliminated. Here are several websites that discuss this method further.

https://www.storyofmathematics.com/sieve-of-eratosthenes
https://www.visnos.com/demos/sieve-of-eratosthenes
https://primes.utm.edu/glossary/page.php?sort=SieveOfEratosthenes

Your goal in this project is to implement this process in Python. You will write two functions called `makeSieve()` and `getIndices()` with headings

```
def makeSieve(n):
```

and

```
def getIndices(L, x):
```

respectively. A call to `makeSieve(n)` will return a list S of length $n + 1$ containing Boolean values True and False. The returned list will satisfy

$$S[k] = \begin{cases} \text{True} & \text{if } k \text{ is prime} \\ \text{False} & \text{if } k \text{ is not prime} \end{cases}$$

for each index k in the range $0 \leq k \leq n$. For instance, if $n = 20$, then S will be the list

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
F	F	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F

indicating that the set of primes less than or equal to 20 are the indices $\{2, 3, 5, 7, 11, 13, 17, 19\}$. In other words, each list element $S[k]$ identifies its index k as either prime (True) or composite (False). Note that index 0 is included only since lists must begin at index zero. To accomplish this, the call `makeSieve(n)` will create a list of length $n + 1$, whose first two elements are False, and whose remaining $n - 1$ elements are True. It will then implement the procedure described above, systematically assigning False to each composite index in the range $2 \leq k \leq n$.

The function call `getIndices(L, x)` will return a list of integers, in increasing order, consisting of those indices of the list `L` at which the value `x` is stored. For instance `getIndices([1, 2, 3, 2, 1], 2)` will return the list `[1, 3]`. As a more relevant example, if `S` is the list returned by `makeSieve(10)`, then `getIndices(S, True)` will return the list `[2, 3, 5, 7]`, which are the prime numbers in the range 1 to 10. Study the examples LinearSearch1.py and LinearSearch2.py for hints on how to implement `getIndices()`.

Function `main()` in this assignment will prompt the user for a positive integer n. If the user enters a non-positive integer, your program will continue to prompt for a positive integer, until one is entered. It will then call `makeSieve(n)`, then use the returned list to create two lists. The first being a list of prime numbers in the range 1 to n, and the second, a list of composite numbers in the same range. It will then print out the list of primes separated by spaces, with 10 primes to a line, followed by the list of composites formatted in the same way.

Your source file for this project will be called Sieve.py. An example run of the program is included below. As usual, `$` represents the command line prompt on your platform.

```
$ python3 Sieve.py

Enter a positive integer: -1
Please enter a positive integer: 0
Please enter a positive integer: 50

There are 15 prime numbers in the range 1 to 50:

2 3 5 7 11 13 17 19 23 29
31 37 41 43 47

There are 35 composite numbers in the range 1 to 50:

1 4 6 8 9 10 12 14 15 16
18 20 21 22 24 25 26 27 28 30
32 33 34 35 36 38 39 40 42 44
45 46 48 49 50

$
```

Observe the instances of blank lines in the output, and how the prompt changes when the user begins by entering a non-positive integer. As usual, your program must follow this format exactly to receive full credit.

You will follow the form of GeneralTemplate.py posted in Examples on the class webpage. Another file called SieveTemplate.py is included in the folder Examples/pa4. It contains function headings, and some comments in `main()` giving a rough outline of program logic. You can start the project by downloading this file, changing its name and filling in the details, being sure to test each function as you go.

Two more files are included in Examples/pa4 called TestSieve.py and TestSieveOut. TestSieve.py is a python program that can be used to test the correctness of your function `makeSieve()`. The expected output of this file is contained in TestSieveOut. Place these files in the same directory as your program Sieve.py, then run TestSieve.py. Compare the output to TestSieveOut. If they do not match, something is wrong with your program. If they do match, it does not prove that your function `makeSieve()` is fully correct, but it provides evidence of correctness.

What to turn in
Submit your source code file Sieve.py to the assignment pa4 on Gradescope before the due date. This project is somewhat more complex than previous assignments, so get an early start and seek help if anything is unclear.