CMPS 5P
Introduction to Programming in Python
Programming Assignment 4

In this assignment you will write a Python program that reads a positive integer n from user input, then prints out the first n prime numbers. 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. A positive integer p is called *prime* if its only positive divisors are 1 and p. The one exception to this rule is the number 1 itself, which is considered to be non-prime. A positive integer that is not prime is called *composite*. Euclid showed that there are infinitely many prime 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 test a number m for primality, but perhaps the simplest is to do a sequence of trial divisions. Begin by dividing m by 2. If it divides evenly (i.e. has zero remainder), then m is not prime, otherwise divide by 3. Continue in this fashion dividing by 4, then 5, etc. If at any point m is found to be divisible by a number d in the range $2 \le d \le m-1$, then halt, and conclude that m is composite. Otherwise, conclude that m is prime. However a moment's thought shows that one need only do trial divisions by numbers that are themselves prime. For instance if a trial division by 2 fails (i.e. has non-zero remainder showing that m is odd), then a trial division by 4, 6, 8 or any even number must also fail. Therefore to test a number m for primality do trial divisions by *prime* numbers less than m.

A little more thought shows that it is not necessary to go all the way up to $m-1$, and in fact one need only divide m by primes p in the range $2 \le p \le \sqrt{m}$. To see this, suppose $m > 1$ is composite. Then there exist positive integers a and b such that $1 < a < m$, $1 < b < m$, and $m = ab$. If both $a > \sqrt{m}$ and $b > \sqrt{m}$, then $m = \sqrt{m} \cdot \sqrt{m} < a \cdot b = m$, a contradiction. Hence either $a \le \sqrt{m}$ or $b \le \sqrt{m}$, and it follows that some prime satisfying $2 \le p \le \sqrt{m}$ divides m. Therefore if no such prime exists, then m must itself be prime.

Your goal in this project is to implement this process in Python. You will write a function called `isPrime()` with the heading

```
def isPrime(m, L):
```

where m is the number to be tested for primality and L is a list containing sufficiently many primes to do the testing. This function will return `True` or `False` according to whether m is prime or composite. At the time `isPrime(m, L)` is called, the list L must contain (at least) all primes p in the range $2 \le p \le \sqrt{m}$. For instance to test $m = 53$ for primality, one does successive trial divisions by 2, 3, 5 and 7. We go no further since $11 > \sqrt{53}$. Thus when `isPrime(53, L)` is called we must have $\text{len}(L) \ge 4$ and $L[0] = 2$, $L[1] = 3$, $L[2] = 5$, $L[3] = 7$. The return value in this case would be `true` since all these divisions fail. Similarly to test $m = 143$ divide by 2, 3, 5, 7 and 11 (since $13 > \sqrt{143}$). Thus when `isPrime(143, L)` is called it must be that $\text{len}(L) \ge 5$ and $L[0] = 2$, $L[1] = 3$, $L[2] = 5$, $L[3] = 7$, $L[4] = 11$. The return is `false` in this case since 11 divides 143. Function `isPrime()` will contain a loop that steps through the list L doing trial divisions, terminating when either a trial division succeeds (in which case `false` is returned), or when the next prime in L is greater than $\sqrt{m}$ (in which case `true` is returned.)

The main program in this project (i.e. that part of the program outside of any function) will prompt for and receive a positive integer giving the number of primes to generate. In the context of this main program we will refer to the list of primes as `PrimeList`, though you may use any variable name you like. `PrimeList` will contain only a small number of primes initially. Your main program will enter a loop that appends new primes to `PrimeList` until it is of the required length, then print out the contents of the list in the format described below. Observe that `PrimeList` plays a dual role in this project. On the one hand it is used to collect, store and print the output data. On the other hand, it is passed to function `isPrime()` to test new integers for primality. Whenever `isPrime()` returns `true`, the newly discovered prime is appended to `PrimeList`. This process works since as explained above, the primes needed to test an integer m range only up to $\sqrt{m}$, and all of these primes (and more) will be stored in `PrimeList` at the time m is tested. It is necessary to initialize `PrimeList` to hold at least one prime, say `PrimeList = [2]`, or `PrimeList = [2, 3]`.

The following is an outline of the steps to be performed by the main program.

- Prompt for and read a positive integer n giving the number of primes to generate.
- Initialize `PrimeList` to store the first several primes in order.
- Enter a loop that calls function `isPrime()` on successive integers, appends newly discovered primes to `PrimeList`, and halts when `PrimeList` is of the required length.
- Print the contents of `PrimeList` 10 to a line, separated by single spaces. Note that if n is not a multiple of 10, then the last line of output will contain fewer than 10 primes.

Your program, which will be called Primes.py, will produce output matching the sample runs below. (As usual `%` signifies the terminal prompt.)

```
% python Primes.py

Enter the number of Primes to compute: 10

The first 10 primes are:
2 3 5 7 11 13 17 19 23 29

% python Primes.py

Enter the number of Primes to compute: 97

The first 97 primes are:
2 3 5 7 11 13 17 19 23 29
31 37 41 43 47 53 59 61 67 71
73 79 83 89 97 101 103 107 109 113
127 131 137 139 149 151 157 163 167 173
179 181 191 193 197 199 211 223 227 229
233 239 241 251 257 263 269 271 277 281
283 293 307 311 313 317 331 337 347 349
353 359 367 373 379 383 389 397 401 409
419 421 431 433 439 443 449 457 461 463
467 479 487 491 499 503 509
```

```
% python Primes.py

Enter the number of Primes to compute: 285

The first 285 primes are:
2 3 5 7 11 13 17 19 23 29
31 37 41 43 47 53 59 61 67 71
73 79 83 89 97 101 103 107 109 113
127 131 137 139 149 151 157 163 167 173
179 181 191 193 197 199 211 223 227 229
233 239 241 251 257 263 269 271 277 281
283 293 307 311 313 317 331 337 347 349
353 359 367 373 379 383 389 397 401 409
419 421 431 433 439 443 449 457 461 463
467 479 487 491 499 503 509 521 523 541
547 557 563 569 571 577 587 593 599 601
607 613 617 619 631 641 643 647 653 659
661 673 677 683 691 701 709 719 727 733
739 743 751 757 761 769 773 787 797 809
811 821 823 827 829 839 853 857 859 863
877 881 883 887 907 911 919 929 937 941
947 953 967 971 977 983 991 997 1009 1013
1019 1021 1031 1033 1039 1049 1051 1061 1063 1069
1087 1091 1093 1097 1103 1109 1117 1123 1129 1151
1153 1163 1171 1181 1187 1193 1201 1213 1217 1223
1229 1231 1237 1249 1259 1277 1279 1283 1289 1291
1297 1301 1303 1307 1319 1321 1327 1361 1367 1373
1381 1399 1409 1423 1427 1429 1433 1439 1447 1451
1453 1459 1471 1481 1483 1487 1489 1493 1499 1511
1523 1531 1543 1549 1553 1559 1567 1571 1579 1583
1597 1601 1607 1609 1613 1619 1621 1627 1637 1657
1663 1667 1669 1693 1697 1699 1709 1721 1723 1733
1741 1747 1753 1759 1777 1783 1787 1789 1801 1811
1823 1831 1847 1861 1867

%
```

What to turn in

Submit the file Primes.py to the assignment name pa4 in the usual way. This project is somewhat more complex than previous assignments, so get started early and ask questions if anything is less than clear.