CS:1210 Homework 3

Due via ICON on Friday, Feb 27th, 4:59 pm

What to submit: Your submission for this homework will consist of four text files, named `hw3a.py`, `hw3b.py`, `hw3c.py`, and `hw3d.py`. The `.py` files should contain Python code that solve Problems (a), (b), (c), and (d) respectively. These files should each start with with a comment block containing your name, section number, and student ID. You will get no credit for this homework if your files are named differently, have a different format, and if your files are missing your information. Your program should all be well documented, i.e., have useful comments. We will discuss what constitutes good documentation and coding style in class. Part of good documentation is choosing meaningful names for your variables. The files `hw3a.py`, `hw3b.py`, and `hw3d.py` should contain the functions `farthestConsecutivePrimes`, `nearHalfInteger`, and `twoDRandomWalk` respectively, and no main programs. In addition, the file `hw3a.py` should contain the function `isPrime`. The file `hw3c.py` should contain the function `nearHalfInteger` and a main program.

(a) Write a function `farthestConsecutivePrimes` that takes a positive integer parameter, say N, and returns a *list* $[m, n]$ of consecutive primes $m < n \leq N$ such that the gap between them, i.e., $n - m$ is maximum among all pairs of consecutive primes less than or equal to N. For example, the function call `farthestConsecutivePrimes(150)` should return the list `[113, 127]` because these two consecutive primes that are less than or equal to 150 have the largest gap (i.e., $14 = 127 - 113$) compared to all other pairs of consecutive primes less than or equal to 150. The function `farthestConsecutivePrimes` should repeatedly call the boolean function `isPrime` that determines if a given positive integer is a prime. **Note:** We have not talked about lists in Python yet, but starting next week we will spend the rest of the semester talking about Python lists. For this function, you don't need to know anything about lists, except that once your function has figured out `m` and `n`, it can end with the statement `return [m, n]`.

(b) Let us call a floating point number a *near half integer* if it is within 1/1000 of a fraction $\frac{n}{2}$, for some integer n. In other words, a floating point number is a *near half integer* if it is within 1/1000 of some number in $H = \{\ldots, -1.5, -1, -0.5, 0, 0.5, 1, 1.5, \ldots\}$. Note that H contains all integers and all "halfs." Write a function called `nearHalfInteger` that takes a a floating point number f as a parameter and if f is a near half integer then the function returns the number in H that f is closest to; otherwise `nearHalfInteger` returns the Python constant `None`.

(c) Write a program that starts by prompting the user for a positive integer, let us call this n. The program then reads n floating point numbers input by the user (typed one in each line) and outputs the number of near half integers in the input. Here is an example interaction between the program and the user; the last line below is produced by the program.

```
5
3.00004
4.51
-9.99999999
-3.49999
11.8
The number of near half integers is 3
```

Your program will need to repeatedly call the function `nearHalfInteger` from (b).

(d) Write a function called `twoDRandomWalk` that simulates a 2-dimensional random walk. This function starts off a "robot" at point $(0, 0)$ and then in each step of the random walk the robot moves 1 step in one of 4 directions (north, south, east, west) chosen at random with equal probability. Imagine that there is a $2n \times 2n$ square "barrier" centered around point $(0, 0)$ and the random walk ends when the robot reaches any point on this barrier More precisely, for any given positive integer n, the barrier is defined by the vertical lines $x = n$, $x = -n$, and the horizontal line $y = n$, and $y = -n$. The function should return the number of steps the robot took before ending the random walk. The function should have the following header:

```
def twoDRandomWalk(n = 100, printOn = False):
```

where the first parameter `n` specifies the barrier, while the second argument `printOn` tells the function whether it should do its work quietly or whether it should print the locations of the robot and it travels. In other words, if `printOn` is `True` then the function prints the robot locations as it moves. This is in addition to returning the length of the random walk. If `printOn` is `False` then the function prints nothing and simply returns the length of the random walk.