22C:16 Homework 3

Due via ICON on Wednesday, Feb 16th, 4:59 pm

1. Consider this Python program. Write down the *type* of the variable `x` after each assignment statement. Ideally, you should be able to answer this question without typing any of this on a computer, but you should feel free to use a computer if that helps you.

```
import math
import sys
x = sys.maxint - 100
x = 2*x
x = x - x
x = int(x)
x = math.sqrt(sys.maxint)
```

2. Write down the values of each of these expressions. Make sure you pay attention to whether the answer has type `int` or `float`.
   (a) `math.trunc(8.97)`
   (b) `math.ceil(8.97) - math.floor(8.97)`
   (c) `round(8.97, 1)`
   (d) `abs(math.ceil(-8.97))`
   (e) `int(math.ceil(math.log(565656, 10)))`

3. Recall how we figured out the running time of our primality testing program on some inputs by using functions from the `time` module. For this problem, I want you to perform some experiments that will help you understand the running time of your program better.
   (a) Run your primality testing program on ten 15-digit primes and report the average running time (with the average taken over the ten runs).
   (b) Run your primality testing program on ten 16-digit primes and report the average running time (with the average taken over the ten runs). How does the running time for 16-digit primes compare with the running time for 15-digit primes? Explain what you see in a sentence.
   (c) Instead of stopping when `factor` reaches $\sqrt{n}$, suppose we let `factor` travel all the way to $n - 1$. How long would you program now take to test a 15-digit prime? Since this might take too long, I suggest you start up your program and let it run for 5 minutes (as measured by your watch) and then restart your shell. So your answer should be either (i) a running time in seconds, if your program completes within 5 minutes, or (ii) a message telling us that your program did not complete within 5 minutes.

   I found an easy-to-use website with collections of primes of various sizes:
   `http://www.alpertron.com.ar/googolm1.pl`

   Feel free to use this or any other source of primes.

4. 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 minimum, maximum, and average of these numbers. Here is an example interaction between the program and the user.

```
Enter the positive integer that denote the length of your sequence: 5
Now input your sequence, one number per line.
3.4
4.5
-9.23
3.456
11.8
The minimum is -9.23
The maximum is 11.8
The average is 2.7852
```

   Your program does not need to do any "error checking" and can assume that the user will correctly follow the program's instructions.

5. Let us call an floating point number a *near integer* if it is within 1/1000 of an integer. Write a program that interacts with the user in a manner quite similar to the previous program, except that it eventually outputs the number of near integers in the input sequence. Here is an example interaction between the program and the user.

```
Enter the positive integer that denote the length of your sequence: 5
Now input your sequence, one number per line.
3.00004
4.5
-9.99999999
3.456
11.8
The number of near integers is 2
```

   Your program does not need to do any "error checking" and can assume that the user will correctly follow the program's instructions.
