CS:1210 Homework 1

Due via ICON on Monday, Feb 2nd, 4:59 pm

What to submit: Your submission for this homework will consist of one pdf file and three text files, named `hw1a.pdf`, `hw1b.py`, `hw1c.py`, and `hw1d.py`. The pdf file should contain your responses to Problem (a) and the `.py` files should contain Python programs for Problems (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 (e.g., docx), and if your files are missing your information.

(a) Make sure that Python 3 and Wing IDE 101 have been correctly downloaded onto your computer. Then download (or cut-and-paste) the program `intToBinary5.py` and run it in the Wing IDE. When prompted for a nonnegative integer, type `7987`.

  (i) What is the output produced by your program?

  (ii) Write down the entire sequence of values that the variables `n` and `suffix` take on during the execution of the program with input `7987`. Note that the first value of the variable `n` is `7987` and the first value of `suffix` is `""`. You can either figure this out by hand or you can insert `print` statements in appropriate locations and have your program produce this information.

(b) Here is a program, that I call `binaryToDecimal.py`, that takes as input a nonnegative binary integer and is supposed to output the equivalent decimal number. For example, if the input is `11001` then the output should be `25`. Thus this program does the "opposite" of what `intToBinary5.py` does.

```
# Get input
n = int(input("Input a nonnegative binary integer: "))

dec = 0 # we will build the decimal equivalent in this variable
placeValue  = 1 # the place value of the right most bit is 1


# Repeatedly extract bits from n, right-to-left
while n > 0:
    placeValue = placeValue * 2 # update place value
    bit = n % 10 # get the rightmost bit
    dec = dec + bit * placeValue # update dec by value of new bit
    n = n / 10 # everything except the last digit is reassigned to n

print(dec)
```

Unfortunately, `binaryToDecimal.py` does not quite behave as it is supposed to. I have separately posted this program on the course website for you to download and play with. The program contains two small "errors" and if you fix these, it should run correctly. Correct the errors in the program and submit the corrected program.

(c) Write a program that takes as input a real number x and a nonnegative integer t and outputs the value of the series:

$$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots + \frac{x^t}{t!}.$$

Here is an example illustrating the interaction between this program and the user:

```
Input a real number: 2.0
Input a nonnegative integer: 3
6.333333333333333
```

In the above example, $1 + 2.0 + (2.0)^2/2! + (2.0)^3/3! = 1 + 2.0 + 2.0 + 8.0/6 = 6.3333...$

Here is an informal description of the algorithm you are required to use. First note that the series you need to compute contains $t+1$ terms, with the first term being 1. So initialize a variable called `series` to 1; this will be the variable that will eventually contain the answer you need to output. Now set up a `while`-loop that executes t times. Such a loop would look like this:

```
counter = 1
while counter <= t:
    ...
    ...
    counter = counter + 1
```

In each execution of the loop, you will need to construct the next term of the series and add it to the variable `series`. Now note that each term in the series consists of a numerator and a denominator. The numerator can be constructed by multiplying the previous numerator by x and the denominator can be constructed by multiplying the previous denominator by `counter`.

(d) Write a program that takes as input a positive real number s and a positive real number ε and outputs the approximate value of $\sqrt{s}$ to within the tolerance ε. In other words, the output of your program should be a number s' such that $|s' - \sqrt{s}| \leq \varepsilon$. (Thus the "distance" between the output s' and the correct answer $\sqrt{s}$ should be at most ε.)

Your program should implement a very old algorithm for approximating the square root of numbers, called the *Babylonian Method* (also attributed to the Greek mathematician Hero of Alexandria and therefore sometimes called *Hero's Method*). Here is the pseudocode for this algorithm.

```
1. start with an arbitrary initial guess $x_0 \neq 0$ and let $i = 1$.
2. compute $x_i$ as the average of $x_{i-1}$ and $s/x_{i-1}$.
3. if the difference between $x_i$ and $x_{i-1}$ is less than $\varepsilon$, output $x_i$ and halt.
4. else increment $i$ and return to step (2).
```