CS:1210 Practice Problem Set 4

Complete before Tuesday, Feb 17th

1. Without executing this program on a computer, figure out what output it produces.

```
n = 1
while n < 6:
    m = n
    line = ""
    while m > 1:

        if m % 2 == 0:
            m = m/2
        else:
            line = line + str(m) + " "
            m = 3*m + 1
    line = line + str(1)
    print line

    n = n + 2
```

2. Write down the boolean value that each of these expressions evaluates to. For expressions containing the variable `x`, assume that the value of `x` is 13.

   (a) `(10 != 200) and (True)`

   (b) `(not (x < 15)) or (x > 10)`

   (c) `not(not(x != 13))`

   (d) `(10 < 20) and ((20 < 30) or (20 != 20))`

   (e) `(x == 10) or ((x < 10) or (not (x > 20)))`

   (f) `abs(5 - 25) < 15`

   (g) `(int(len("hello")/3.0) == 1) and (not(len("hi") >= 2))`

   (h) `not(not(not False)) or (False or True)`

   (i) `(5 < 10) and ((10 < 5) or ((3 < 18) and not (8 < 18)))`

   (j) `(not (5 < 10)) or (not (3 != 4) and not (8 > 11))`

3. Look up the meaning of the following functions defined in the `math` module. You can find documentation for the `math` module at `https://docs.python.org/3/library/math.html` and after reading the documentation, you should also try playing with these in the Python shell.

   - `ceil(x)`
   - `factorial(x)`
   - `floor(x)`
   - `trunc(x)`
   - `pow(x, y)`

Now we want you to be able to evaluate the following expressions away from the computer. Write down the value and type of each expression.

(a) `math.ceil(5.75) - math.floor(5.75)`

(b) `math.ceil(5) - math.floor(5.0)`

(c) `math.trunc(10.5)/3`

(d) `math.pow(2, 3) - math.pow(3, 2)`

(e) `math.factorial(5)/10`

(f) `math.ceil(math.sqrt(20))`

(g) `math.floor(math.log10(50))`

4. Start with the code we wrote to solve the *primality testing* problem and turn this into a boolean function called `isPrime` takes a positive integer as an *argument* and returns `True` if the argument is a prime; `False` otherwise.

5. Write a function called `numPrimes` that takes a nonnegative integer argument, say N, and returns the number of primes less than or equal to N. Your function should repeatedly call the function `isPrime` (see previous problem).

6. Write a function called `isPerfectSquare` that takes a nonnegative integer as an argument and returns True if the argument is a *perfect square* and `False` otherwise. Thus the function call `isPerfectSquare(100)` should return `True` and the function call `isPerfectSquare(60)` should return `False`.

There are two more problems in the following pages.

7. Here is a partially completed program that repeatedly prompts the user for a positive integer and outputs all the factors of that integer. The program repeats this until the user types `done`. The program outputs the factors of each given positive integer in one line. Here is an example interaction between the program and the user. The user enters the positive integers 22, 31, and 64 followed by `done`.

```
Enter a positive integer: 22
Factors: 1 2 11 22
Enter a positive integer: 31
Factors: 1 31
Enter a positive integer: 64
Factors: 1 2 4 8 16 32 64
Enter a positive integer: done
```

The program below has two blanks that need to be filled.

```
# repeat until user types "done"
while True:
    inputString = input("Enter a positive integer: ")

    # Check if inputString is done and if so break out of loop

    if ________________________:

        break

    # This part of the code processes a positive integer
    n = int(inputString)
    factor = 1 # tracks potential factors of n
    # The string variable outputString is used to construct
    # the line of output with all factors of n
    outputString = "Factors: "

    # loop through all potential factors
    while factor <= n:
        if n % factor == 0:
            # Update the outputString

            ------------------------------------------------------

        factor = factor + 1

    print(outputString)
```

8. Here is a partially completed program that aims to solve the following problem. The user types in a sequence of positive integers, one per line, ending with the number 0. The program reads in this sequence and counts and outputs the number of pairs of consecutive numbers that are in increasing order. An example interaction of this program with the user is given below.

```
Type a positive int (zero if done) 20
Type a positive int (zero if done) 23
Type a positive int (zero if done) 25
Type a positive int (zero if done) 20
Type a positive int (zero if done) 19
Type a positive int (zero if done) 9
Type a positive int (zero if done) 10
Type a positive int (zero if done) 0
3
```

The program outputs 3 because it detects three pairs of consecutive numbers in increasing order: (i) 20, 23, (ii) 23, 25, and (iii) 9, 10. The program below has two blanks to fill. Your task is to fill in these blanks.

```
# Variable used to read input numbers
current = int(input("Type a positive int (zero if done)."))
# tracks the number just prior to the most recently read number
previous = current
# counter to track the number of consecutive, increasing pairs
numIncreasingPairs = 0
while current != 0:
    if current > previous:
        numIncreasingPairs = numIncreasingPairs + 1

    # Update previous (Blank 1)

    ---------------------------------------------

    # Update current (Blank 2)

    ------------------------------------------------------------

print(numIncreasingPairs)
```