CS:1210 Exam 1

Feb 20th, 6:30 pm to 8:30 pm

Instructions:

- This is an open notes exam and you have 2 hours to complete it. There are 4 problems in the exam and these appear on 7 pages. The exam is worth 100 points (10% of your grade); the first two problems are worth 20 points each and the last two are worth 30 points each.
- Make sure that you do not have any electronic devices (laptops, phones, calculators, dictionaries, etc.) on your desk; you are not allowed to access these during your exam.
- Write as neatly as you can.
- Show all your work, especially if you want to receive partial credit.

Name:

Circle your discussion section:

A01 (8:30-9:20, 221 MLH)	A02 (9:30-10:20 213 MLH)	A03 (11-11:50, 116 MH)
A04 (12:30-1:20, 66 SH)	A05 (3:30-4:20, 248 JH)	A06 (5-5:50, 346 JH)

1. [**20 points**] Write down the *value* and *type* of each of these expressions. Assume that the modules `math` and `random` have already been imported prior to the evaluation of these expressions.

   (a) `float(str(54/10) + "03")`

   (b) `(random.randint(0, 10)*10 <= 100) or (random.randint(0, 10) < 5)`

   (c) `int(str(543 % 10) + str(543 // 10))`

   (d) `math.sqrt(16) + 3`

(e) `(not not not True) and True`

(f) `76 // 10 + 76 / 10`

(g) `int(5 + random.random())//2`

(h) `(153 < 153) or (not (175 != 198))`

(i) `"The answer is"+ " " + str(0.5)`

(j) `str(15.0//3 + 3//2)`

2. **[20 points]** You are given some code and asked to answer questions about this code.

(a) What output does this program produce?

```
n = 5
while n < 10:
    m = n - 1
    while m < 2*n:
        print(n, m)
        if m % 3 == 0:
            break
        m = m + 2

    print("***")
    n = n + 1
```

(b) What output does this program produce? The function `foo1` uses certain variables; for each variable used in `foo1`, write down whether that variable is a (i) parameter, (ii) local variable, or (iii) global variable. Answer the same question for `foo2`.

```
def foo1(m):
    n = 3
    return m + n - y


def foo2(p):
    global y
    y = y - 2
    return p*p + y

# main program
y = 13
while True:
    if y % 5 != 0:
        print(foo2(foo1(y)))
        print(foo1(foo2(4)))
    else:
        break
```

3. **[30 points]** In this problem, you are given two partially completed programs. Your task is to complete each program.

   (a) Here is a primitive, partially completed, program that is meant to test the typing skills of a user. The user is required to repeatedly type one of the following three words correctly: "`mongoose`" or "`squirrel`" or "`possum`". Each word needs to be typed in a new line and it does not matter which of these three words the user types. Your program should let the user keep typing until the user makes 3 mistakes. As soon as the user makes the third mistake your program should terminate with a message giving the percentage of words correctly typed by the user. (Note that this is 100 times the number of correctly typed words divided by the number of typed words.)

   The program contains two "blanks" where code is missing. Your task is to write this code.

```
error = 0 # tracks the number of errors
count = 0 # tracks the total number of words typed by user
while True:
    word = input()
    count = count + 1

    # Increment variable error if the user has made an error
    # Blank 1
    if _______________________________________________________:

        error = error + 1

    if error == 3:
        break

# Output percentage of correct words
# Blank 2
print("Percentage of correct words:", _____________________________)
```

(b) For Homework 1, you implemented an algorithm to compute the square root of a given positive real number. A similar method could be used to compute the *cube root* of a positive real number. Given positive floating point numbers s and ε, the following algorithm (described in pseudocode) will output a real number s' such that $|s' - \sqrt[3]{s}| \leq \varepsilon$. In other words, the algorithm will output s' that is quite close (i.e., within distance ε) to the actual cube root of s.

Pseudocode:

(a) start with the initial guess $x_0 = s/3$ and let $i = 1$.

(b) compute x_i using the formula

$$x_i = \frac{1}{3}\left(\frac{s}{x_{i-1}^2} + 2 \cdot x_{i-1}\right).$$

(c) if the difference between x_i and x_{i-1} is less than or equal to ε, output x_i and halt.

(d) else increment i and return to step (2).

For example, suppose that we want to compute the cube root of $s = 60$. Then, $x_0 = 20$ and $x_1 = 1/3 \cdot (60/400 + 2 \cdot 20) = 1/3 \cdot (40.15) = 13.38333$. Values of $x_2, x_3, \ldots$ can be computed in a similar manner from previous values in the sequence.

Given below is a partially completed function called `cubeRoot` that implements the above algorithm. Your task is to fill in the three blanks in the code.

```
def cubeRoot(s, e):
    oldX = s/3 # initial guess
    # The next value of x
    # Blank 1

    newX = _______________________________________

    while True:
        # If the new x and the oldx are close
        # enough, then return an answer
        # Blank 2

        if _______________________________________

            return newX

        # Update newX and oldX
        # Blank 3
        oldX = newX

        newX = ____________________________________
```

4. **[30 points]** For each part of this problem, you are required to write a function.

   (a) Write a function called `encode` that takes as parameters a positive integer n and a nonnegative integer k and uses k to encode n. The encoding process works as follows: to each digit of n, the value k is added to produce a new digit, and this new digit replaces the old digit. So for example if $n = 243$ and $k = 2$ then the encoding process would yield 465. If the result of adding k to a digit of n yields a result with more that one digit, we simply keep digit in the one's place and throw away the remaining digits. For example, if $n = 287$ and $k = 5$ then the encoding process would yield 732. (This is because 7 + 5 = 12 and we only keep the 2; 8 + 5 = 13 and we only keep the 3; and 2 + 5 = 7.)

   Use the following function header:

```
def encode(n, k):
```

   The function should return the encoded number as a string. For example, the function call `encode(596, 5)` should return the string "`041`".

   Note: My code for this function is 7 lines long; your code should not be too much longer.

(b) Write a boolean function `isPerfectCube` that takes a positive integer n as a parameter and returns `True` if n is a *perfect cube*. Otherwise, the function returns `False`. Note that a positive integer n is a *perfect cube* if there is some integer m such that $m^3 = n$. The first five perfect squares are 1, 8, 27, 64, and 125. Thus, if $n = 64$, the function should return `True`, but if $n = 35$, the function should return `False`.

To do its work, the function `isPerfectCube` should call the function `cubeRoot` that was discussed in Problem 3(b). Now note that `cubeRoot` will only return an *approximate* cube root. So for example, `cubeRoot(64, 0.001)` will return a floating point number that could be anywhere between 7.999 and 8.001.

Notes: (i) The correctness of your answer for this problem is completely independent of the correctness of your answer for Problem 3(b). (ii) My code for this function is 3 lines long; your code should not be too much longer.