22C:16 CS:1210 Exam 1

Feb 22nd, 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.
- 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 section:

SCA (M, W evening)	A01 (9:30-10:20)	A02 (11-11:50)	A03 (12:30-1:20)
	A04 (2:00-2:50)	A06 (3:30-4:20)	

1. [**30 points**] For each of the following expressions, first determine if evaluating it would cause Python to report an error; if so, explain in a sentence what it was about the expression that caused the error. Otherwise, if you think the expression will evaluate in Python in an error-free manner, write down the *value* that the expression evaluates to and its *type*. Assume that the modules `math`, `sys`, and `random` have already been imported prior to the evaluation of these expressions. Further, assume that the value of `sys.maxint` is `9223372036854775807`.

   (a) `len(str(random.randint(1, 5)*10)) != 3`

   (b) `str(5 < len("Exam1" + "CS") or 100/0) + "False"`

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

(d) `sys.maxint + 2 - 5`

(e) `(math.ceil(20.56) - math.floor(6.90))/10`

(f) `3 + (-2**(3/2))**3 - 3`

(g) `3.0 * float(raw_input("Enter an integer:  "))`
(Assume that user inputs the expression `2 + 5`, when prompted)

(h) `not bool(0.1) or not (100 and not False)`

(i) `math.sqrt(math.sqrt(25)+4)*10/20**0`

(j) `10 < 2 or "hello" + "hi" - "ok"`

2. [**32 points**] For each of the programs below, write down the output produced by the program.

(a) What output does this program produce?

```
x = 15
while x < 100:
    y = x + 40
    while (x < y) :
        if (y % 10) == 5 :
            y = y + 15
        else :
            print x, y
            y = y - 35
    x = x + 30
```

(b) What output does this program produce?

```
x = 64
y = 40

count = 0
while abs(x - y) > 0:
    print x, y
    if x > y:
        x = x - y
    else:
        y = y - x

    if count > 4:
        break
    count = count + 1

print x, y
```

(c) What output does this program produce? Please pay attention to all of the print statements.

```
m = 24
n = 30

upperBound = n
if m <= n:
    upperBound = m
print upperBound

factor = 2
maxFactor = 1
while factor <= upperBound:
    if m % factor == 0 and n % factor == 0:
        maxFactor = factor
        print maxFactor

    factor = factor + 1
```

(d) What output does this program produce?

```
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
```

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

   (a) Here is a partially completed program that simulates 1 million rolls of three six-sided dice and reports the number of times *exactly* two out of the three dice rolls were identical. For example, if the first die rolled 3, the second rolled 4, and the third rolled 3, we would count this event. However, if all three dice rolled 3, we would not count this event. Also, if the first die rolled 3, the second rolled 4 and the third rolled 5, we would not count this event because all three dice rolls are distinct.

   The code below is complete except for two boolean expressions that are missing from `if`-statements. Your task is to understand our code and supply the missing pieces. Please read our comments carefully – they provide key hints.

```
import random

counter = 0 # tracks the number of rolls

# numTimes tracks the number of times exactly two
# of the three rolls are identical
numTimes = 0

# roll three six-sided dice million times
while counter < 1000000:
    roll1 = random.randint(1, 6)
    roll2 = random.randint(1, 6)
    roll3 = random.randint(1, 6)

    # First check if at least one pair of the dice rolls
    # have identical values; this allows either two or all three
    # dice rolls to have identical values.

    if ___________________________________________________________:

        # Then check to make sure that not all three rolls are
        # identical

        if _______________________________________________________:

            numTimes = numTimes + 1

    counter = counter + 1

print numTimes
```

(b) 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 = raw_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
```

4. [**30 points**] Suppose that we have defined a boolean function called `relativePrimes` with the following function header:

```
def relativePrimes(m, n):
```

The function takes two positive integers `m` and `n` and returns `True` if they are relatively prime to each other; the function returns `False` otherwise. Recall that two positive integers are *relatively prime* if the only common factor they share is 1. For example, 9 and 14 are relatively prime because nothing larger than 1 evenly divides both 9 and 14. Thus, the function call `relativePrimes(9, 14)` will return `True`. On the other hand, the function call `relativePrimes(24, 60)` will return `False` because (for example) 12 divides both 24 and 60.

(a) Define a function called `smallerRelativePrimes` that takes a single parameter `N` and returns the number of positive integers smaller than `N` that are relatively prime to `N`. For example, `smallerRelativePrimes(14)` should return 6 because 1, 3, 5, 9, 11, and 13 are the positive integers less than 14 that are relatively prime with respect to 14.

The header for this function should be

```
def smallerRelativePrimes(N):
```

and in the body of the function you should be making repeated calls to the given function `relativePrimes(m, n)`. Note that you do not have to define `relativePrimes(m, n)`.

(b) What you see below is a partially completed 2-line function called `isPrime` that takes a positive integer `n` and returns `True` if `n` is prime; and returns `False` otherwise. Fill in the missing expression to complete the function. You should aim to construct this expression by using the function `smallerRelativePrimes` defined in the previous problem.

```
def isPrime(n):
    return _______________________________________________
```