CS 100 Fall 2017 Midterm 1

Monday, October 9, 2017

There are 13 questions on this test. Record your answers to the first 10 questions by circling a letter below. Answer questions 11, 12 and 13 on the attached pages. We have also provided separate scratch pages for writing trial code for the programming problems and tracing code for the multiple choice problems. Work on the scratch pages will not be graded. The value of each question is

1-10 multiple choice (4 points each)
11-13 programming (20 points each)

There is a penalty of one point if your name is not clearly legible and one point if your section is not correct..

Allocate your time accordingly. You may receive partial credit for questions 11, 12 and 13. Answer them as completely as you can. If you finish early, use the extra time to double check your work. When you are done, hand in this answer sheet (including programming question solutions) and the scratch pages and sign the exam attendance sheet.

You may use the summary of Python language elements that is provided. You may not use other notes, books or electronic devices. If you have brought a cell phone or other mobile device you must leave it with the proctor during the exam.

Good luck!

Name (print clearly) ______________________________

Student ID ______________________ Section (see below) __________

001 MW 8:30; **003** TR 1:00; **005** TR 10:00; **007** WF 1:00; **009** TF 2:30; **011** TF 4:00; **013** MW 10:00; **015** MW 11:30; **017** TR 8:30; **019** TR 1:00; **021** MW 11:30; **023** TR 11:30; **025** TR 4:00; **101** T 6:00; **103** W 6:00; **105** R 6:00; **107** R 6:00; **109** F 6:00; **H01** MW 1:00; **H03** WF 10:00; **H05** TR 8:30

Q1 a b c d e

Q2 a b c d e

Q3 a b c d e

Q4 a b c d e

Q5 a b c d e

Q6 a b c d e

Q7 a b c d e

Q8 a b c d e

Q9 a b c d e

Q10 a b c d e

Write code for Questions 11A and 11B here. Use vertical lines for indentation.

Write code for Question 12 here. Use vertical lines for indentation.

Write code for Question 13 here. Use vertical lines for indentation.

Questions 1-10 (4 points each)

Question 1

```
true_count = 0
false_count = 0
if true_count < 0:
    true_count -= 1
    false_count += 1
else:
    true_count -= 2
    false_count += 2
if true_count > false_count:
    true_count -= 4
    false_count += 4
elif true_count == false_count:
    true_count -= 8
    false_count += 8
else:
    true_count -= 16
    false_count += 16
print(true_count, false_count)
```

Hint: value += 1 is the same as value = value+1 and value -= 1 is the same as value = value-1

a) -16 16
b) -17 17
c) -18 18
d) -1 1
e) none of the above

Question 2

```
doors = ['hello', 'I', 'love', 'you']
remix = doors[-1] + doors[1] + doors[-2]
print(remix)
```

a) Index Error: list index out of range
b) youhellolove
c) youIlove
d) lovehelloI
e) none of the above

Question 3

```
polonius = 'This above all'
print(polonius[1:2])
```

a. T
b. This
c. above
d. above all
e. none of the above

Question 4
```
import turtle
s = turtle.Screen()
t = turtle.Turtle()
for i in range(4):
    t.forward(100)
    t.right(90)
    t.forward(100)
```

a) a straight line
b) two adjacent sides of a square
c) three sides of a square
d) a square
e) none of the above

Question 5
```
day = 'Monday'
mix_type = [['Every dog'], 'must have', 'his', day]
print(mix_type[-1])
```

a) IndexError: list index out of range
b) day
c) dog
d) y
e) none of the above

Question 6
```
def rithmetic(n1, n2):
    for i in range(2):
        r = n1%n2
        q = n1//n2
        n1 = r
        n2 = q
    return (n1,n2)

result = rithmetic(10,3)
print(result)
```

a) ZeroDivisionError: division by zero
b) (1,0)
c) (3,0)
d) (1.0, 0.3)
e) none of the above

Question 7

```
warming = True
highground = True

if or if not warming:
    print('sun will rise')
if warming:
    print("many cat5's")
elif not warming and highground:
    print('no flooding')
```

a) SyntaxError: invalid syntax
b) sun will rise
c) sun will rise
   many cat5's
d) sun will rise
   no flooding
e) none of the above

Question 8

```
test = 'common'
repeats = 0
for letter in test:
    if test.count(letter) > 1:
        repeats += test.count(letter)

print(repeats)
```

a) 8
b) 0
c) 2
d) 4
e) none of the above

Question 9

```
def operate(aSeq, aVal):
    if aVal in aSeq:
        return aVal
    if [aVal] in aSeq:
        return [aVal]
    if str(aVal) in aSeq:
        return str(aVal)
    if [str(aVal)] in aSeq:
        return [str(aVal)]

zeroes = [0, [0], '0', ['0']]
print(operate(zeroes, 0))
```

a) [0]
b) None
c) 0, [0], '0', ['0']
d) 0
e) none of the above

Question 10

```
def test(seq):
    prev = ''
    accum = ''
    for current_item in seq:
        if current_item in prev:
            accum += current_item
        prev += current_item
    return accum

print(test('hallelujah'))
```

a) None
b) ''
c) hallelujah
d) llah
e) none of the above

Question 11A (8 points)

Write a function named *double_circle* that uses turtle graphics to draw two circles of specified radius that share a common point of origin. (Hint: you may use the turtle method *circle*.) The function *double_circle* takes two parameters:

1. *t*, a turtle that is used to draw the circles. The turtle *t* may initially be at any location on the screen and in any orientation and may be either up or down.
2. *radius*, an integer that is the radius of each of the circles

The function *double_circle* should:

1. draw two circles, beginning at the initial position and orientation of *t*
2. leave *t* in its initial position and orientation when it returns

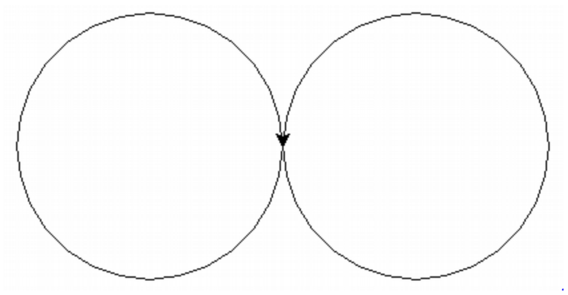
For full credit, you must perform repeated operations using a loop.

For example, the following is an example of correct graphical output.

```
import turtle

scrn = turtle.Screen()
turt = turtle.Turtle()
turt.right(90)

double_circle(turt, 100)
```



Question 11B (12 points)

Write a function named *circle_line* that uses turtle graphics and the function *double_circle* from Question 11a to draw a sequence of double circles that lie along a line. Each successive double circle changes in size relative to the preceding one by a fixed multiplier.

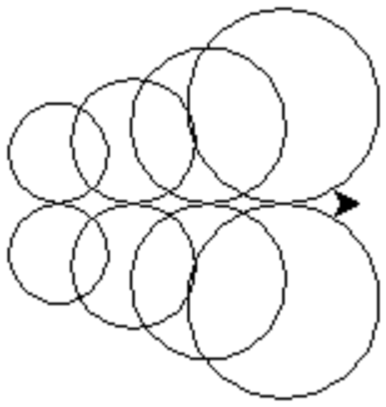
The function *circle_line* takes five parameters:

1. *t*, a turtle that is used to draw
2. *radius*, an integer that is the radius of the first double circle
3. *multiplier*, a positive floating point number that is the ratio of the radii of successive double circles
4. *num_repeats*, an integer that is the number of double circles to draw
5. *separation*, an integer that is the distance between the common point of origin of successive double circles

The function *circle_line* should call the function *double_circle* repeatedly to draw a series of double circles.

For example, the following would be correct graphical output.

```
import turtle
scrn = turtle.Screen()
turt = turtle.Turtle()
circle_line(turt, 20, 1.25, 4, 30)
```



Question 12 (20 points)

Write a function named *partial_symmetry* that takes a single parameter, *string_list* (a list of non-empty strings). The function *partial_symmetry* should return a list of the strings in *string_list* that meet ***either*** of these two requirements.

Requirement 1. The string is of even length ***and*** the string begins and ends with the same character.

Requirement 2. The string is of odd length.

For example, the following would be correct input and output.

```
pirates = ['dead', 'men', 'tell', 'no', 'tales']
print(partial_symmetry(pirates))
['dead', 'men', 'tales']
```

Question 13 (20 points)

Write a function named *greet_student*. The function *greet_student* takes three parameters

1. *state*, a string that is a two-letter state abbreviation (e.g., MI, NY, NJ)
2. *in_msg*, a string message that is printed for in-state students
3. *out_msg*, a string message that is printed for out-of-state students

The function *greet_student* should

1. ask the user for their name. (Assume that the user responds with valid input.)
2. ask the user what state they are from. (Assume that the user responds with valid input.)
3. print out a personalized response, using the appropriate message (depending on whether they are in-state or out-of-state) and their name
4. return a list containing two strings -- the user's name and the state they are from

For example, the following would be correct input and output

```
>>> in_msg = 'You get in-state tuition'
>>> out_msg = 'Welcome to our state'
>>> student = greet_student('NJ', in_msg, out_msg)
What is your name? Rocky
What state are you from? MI
Welcome to our state Rocky
>>> print(student)
['Rocky','MI']
```