CS 100 Spring 2016 Final

Wednesday, May 11, 2016

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 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 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. W

Good luck!

Name (print clearly) ____

Student ID ______________________________ Section (see below) ____________

02 TF 10:00 Vaks; **04** MR 2:30 Cohen; **06** MR 2:30 Diallo; **08** WF 10:00 Diallo; **10** MW 10:00 Spirollari; **12** TF 4:00 Vaks; **14** TF 2:30 Vaks; **16** MR 10:00 Diallo; **18** WF 1:00 Diallo; **102** W 6:00 Vaks

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.

Question 1
```
def symCharacter(s):
    if len(s) == 0:
        return None
    else:
        mid = len(s)//2
        for i in range(mid):
            if s[i] == s[-i-1]:
                return s[-i]
    return ""

t = 'panama'
print(symCharacter(t))
```

a) the empty string
b) None
c) a
d) NameError: name 's' is not defined
e) none of the above

Question 2
```
def dups(txt):
    rtnVal = []
    txtList = txt.split()
    for word in txtList:
        if word not in rtnVal:
            rtnVal.append(word)
    return rtnVal

t = "One fish, two fish."
print(dups(t))
```

a) ['One', 'two']
b) ['One', 'fish', 'two']
c) ['One', 'fish', 'two', 'fish']
d) ['One', 'fish,', 'two', 'fish.']
e) none of the above

Question 3
```
chars = {'odd prime':3, 0:'unsigned', 3:'loneliest','additive identity':0}
print((chars[3][0]))
```

a) o
b) s
c) a
d) IndexError: index out of range
e) none of the above

Question 4
```
aDict = {1:'zero', 'zero':0}
print(aDict[0])
```

a) 'zero'
b) 1
c) (1:'zero')
d) KeyError: 0
e) none of the above

Question 5

```
declaration = ['I', 'am', 'the', 'one']

num = 0
for idx in range(len(declaration)):
    if len(declaration[idx]) == idx:
        num += 1
        num *= 2
print(num)
```

a) 0
b) 2
c) 6
d) 14
e) none of the above

Question 6

```
def breakTest(t):
    tList = t.split()
    rtnVal = 0
    pos = 0
    for word in tList:
        if tList.count(word) > 1:
            rtnVal += 1
            continue
        break
    return rtnVal

s = 'truth is beauty and beauty is truth'
print(breakTest(s))
```

a) 0
b) 1
c) 2
d) 3
e) none of the above

Question 7

```
def trackChars(aString):
    d = {}
    for char in aString:
        if char not in d:
            d[char] = 1
        else:
            d[char] += 1
    return d

print(len(trackChars('hello world')))
```

a) 8
b) 9
c) 10
d) 11
e) none of the above

Question 8

```
T = True
F = False
exprs = [T or not T, not(not T), not(F and T), not(T or F)]
value = 1
multiplier = 1
for expr in exprs:
    multiplier += 1
    if expr == True:
        value += multiplier
print(value)
```

a) 15
b) 10
c) 11
d) 6
e) none of the above

Question 9

```
import turtle
t = turtle.Turtle()
for i in range(4):
    if i%3 == 0:
        t.forward(100)
        t.left(120)
    elif i//3 == 1:
        t.forward(100)
        t.left(60)
    else:
        t.left(60)
```

a) a zigzag line
b) two sides of an equilateral triangle
c) an equilateral triangle
d) a straight line
e) none of the above

Question 10
The lines below are the content of the file named *paradox.txt*.

A paradox?
A startling paradox

After the execution of the following code, what is the content of the file *out.txt*?

```
def vowelCount(inFile, outFile):
    vowels = 'aeiouAEIOU'
    inF = open(inFile, 'r')
    outF = open(outFile, 'w')

    for line in inF:
        vowelCount = 0
        for letter in line:
            if letter in vowels:
                vowelCount += 1
        outF.write(str(vowelCount)+'\n')
    inF.close()
    outF.close()

print(vowelCount('paradox.txt', 'out.txt'))
```

a) 2
   3
b) 10
c) 4
   6
d) FileNotFoundError: No such file or directory: 'paradox.txt'
e) none of the above

Question 11a (6 points)

Write a function named *midline* that draws a line based on two passed parameters

1) *t*, a turtle to use to draw
2) *length*, the length of the line to draw

The turtle *t* is initially at the midpoint of the line that *midline* draws. When *midline* returns, *t* should be in its initial position and orientation. Do not make any assumptions about the initial position or orientation of *t*.

Question 11b (14 points)

Write a function named starburst that uses turtle graphics to draw a number of lines of increasing length that share a common midpoint. Each line is offset from the previous line by a specified angle.

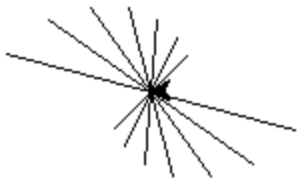
The function starburst must repeatedly call the function midline.

The function starburst takes 5 parameters:

1) *t*, a turtle used for drawing
2) *num*, the number of lines to draw
3) *init*, the length of the first line
4) *delta*, the multiple by which successive lines increase in length
5) *angle*, the counterclockwise rotation between successive lines

For example if *starburst* is called by the following code, the drawing below would be correct output.

```
import turtle
s = turtle.Screen()
t = turtle.Turtle()
t.left(45)
starburst(t, 7, 50, 1.2, 20)
```



Question 12 (20 points)

Write a function named *mostFrequent* that takes two parameters:

1) *inFile*, a string that is the name of an input file
2) *outFile*, a string that is the name of an output file

The input file *inFile* exists when *mostFrequent* is called; *mostFrequent* must create *outFile*. The input file contains only lower case letters and white space.

The function *mostFrequent* identifies the letter(s) that appear most frequently on each line of inFile and writes them to a corresponding line of *outFile*. If more than one letter on a line has the same frequency, they should each be written to *outFile*. Each letter should be written to a line only once.

For example, if the file *moreYouKnow.txt* contains the following lines:

the more that you read
the more things you will know

then the function call

```
mostFrequent('moreYouRead.txt', 'moreYouReadOut.txt')
```

should produce an output file *moreYouReadOut.txt* with the following lines:

te
o

Hint: the space character " " is not a letter and so can not be a most frequent letter.

Question 13a (6 points)
Write a function named *longEnough* that takes two parameters:

1) *s*, a string
2) *threshold*, a non-negative integer

The function *longEnough* should return True if the length of *s* is at least *threshold*, and False otherwise.

For example, the function call longEnough('Amen', 4) should return True.

Question 13b (14 points)
Write a function named longWords that takes two parameters:

1) *t*, a string
2) *i*, a non-negative integer

The string *t* contains only lower case letters and white space.

The function *longWords* should return a dictionary in which the keys are all words in *t* that contain at least *i* characters. The value corresponding to each key is the number of times that that word occurs in *t*.

The function *longWords* should call the function *longEnough* to determine whether a word should be a key in the returned dictionary.

For example, the following is a correct input/output pair

```
>>> text = 'one fish two fish red fish blue fish'
>>> print(longWords(text, 4))
{'blue': 1, 'fish': 4}
```