CS100 Spring 2017 Final

May 11, 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) _____

002 MR 11:30; **004** MW 8:30; **006** MR 1:00; **008** WF 10:00; **010** MR 2:30; **012** TF 4:00; **014 TF** 2:30; **018** WF Z 1:00; **020** TR 10:00; **102** W 6:00 (If you are not sure, check your section on the signout sheet.)

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

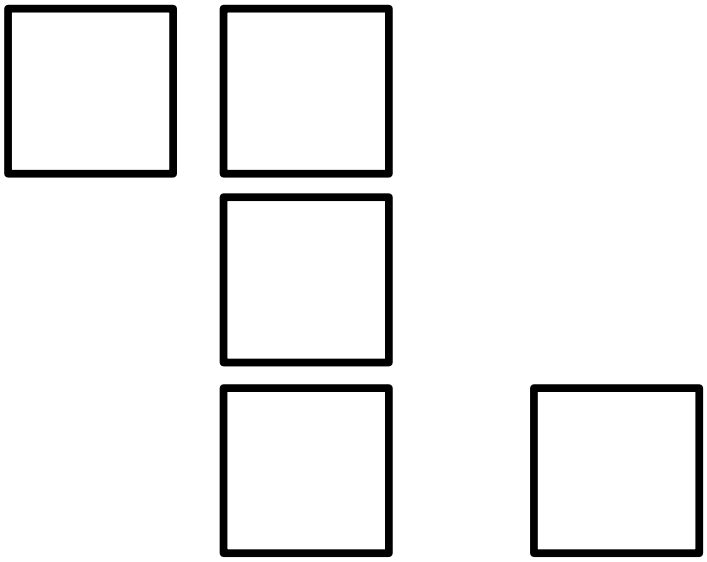
Q6 a b c d e

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.

Multiple choice questions. Four points each.

Question 1

```
def dictTest(s):
    d = {}
    for chr in s:
        chrCount = s.count(chr)
        if chr not in d:
            d[chr] = 1
        else:
            d[chr] += 1
            return len(d)
    return -1

print(dictTest('happy'))
```

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

Question 2

```
def notSubStr(a, b):
    for i in range(1, len(b)+1):
        sub = b[:i]
        if sub not in a:
            return sub
    return a

word0 = 'ads'
word1 = 'ado'
print(notSubStr(word0, word1))
```

a) ad
b) ado
c) ads
d) '' (the empty string)
e) none of the above

Question 3

```
digits = {'one':1, 'z':0, 'zero':0, 'two':2}
print(digits[0][1])
```

a) e
b) zero
c) ValueError: duplicate value 0
d) z
e) none of the above

Question 4

```
def aSym(aString):
    words = aString.split()
    left = 0
    right = -1
    for word in words:
        if words[left] != words[right]:
            return word
        left += 1
        right -= 1
    return ""

s = 'all for one trumps one for all'
print(aSym(s))
```

a) all
b) all for one
c) trumps
d) '' (the empty string)
e) none of the above

Question 5

```
word = 'off'
prev = ""
for letter in word:
    if prev == "":
        prev = letter
        out = ""
        continue
    elif letter == prev:
        prev = letter
        out = letter
        break
    else:
        prev = letter
        out = word[0]
print(out)
```

a) o
b) f
c) of
d) "" (the empty string)
e) none of the above

Question 6

```
for i in range(1,2):
    for j in range(1):
        print(i, j, end = "")
# Hint: end = "" suppresses the endline character of the print statement
```

a) 1
b) 1 0
c) 1 1
d) no output
e) none of the above

Question 7

```
chuck = "He fell on his knees on the bar room floor"

def property(t):
    wordList = t.split()
    d = {}
    for word in wordList:
        length = len(word)
        if len(word) not in d:
            d[length] = 1
        else:
            d[length] += 1
    return len(d)

print(property(chuck))
```

a) 5
b) 10
c) KeyError: 0
d) 4
e) none of the above

Question 8

```
bools = [True or not True, True, not not True, not False, True or False]
output = 0
for i in range(len(bools)):
    if bools[i]:
        output += 1
    else:
        output *= 2
print(output)
```

a) 4
b) 6
c) 8
d) 3
e) none of the above

Question 9

```
subjects = ['compsci', ['algebra', 'calculus'], ['physics', 'chem', 'bio']]
print(subjects[1][0][1])
```

a) p
b) h
c) physics
d) IndexError: string index out of range
e) none of the above

Question 10

```
huck = ["ain’t", "no", "trouble", "to", "do", "wrong"]
indx = 0
while len(huck[indx]) < len(huck):
    indx += 1

print(indx)
```

a) 2
b) 6
c) 7
d) 1
e) none of the above

Question 11A (8 points)

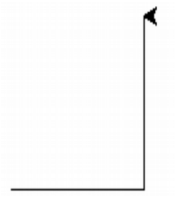
Write a function named *halfSquare* that uses turtle graphics to draw two adjacent sides of a square. The function *halfSquare* takes two parameters:

1) *t*, a turtle that is used for drawing
2) *length*, an integer that is the length of a side

The function *halfSquare* should begin to draw at the initial position and orientation of *t*. The turtle should turn 90 degrees after drawing each side. Do not make any assumptions about the initial up/down state of *t*. For full credit you must use a loop for repeated operations.

Below is an example of a half square drawn by this function

```
import turtle
s = turtle.Screen()
aTurt = turtle.Turtle()
halfSquare(aTurt,100)
```



Question 11B (12 points)

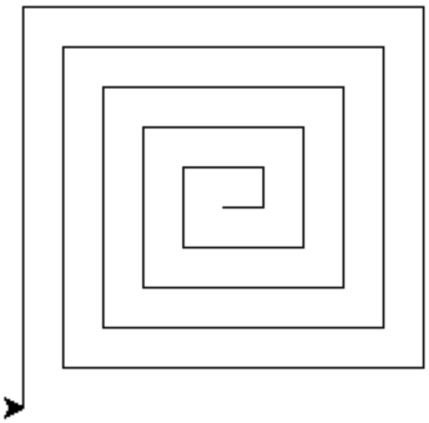
Write a function named *spiral* that repeatedly calls the function *halfSquare* from part A of this question to draw a series of half squares of increasing size. Each successive half square begins where the previous one ends. Begin drawing at the initial location of the turtle.

The function *spiral* takes 3 parameters:

1) *t*, a turtle used for drawing
2) *increment*, the side length of the first half square and the increase in length of successive half squares
3) *num*, an integer that is the number of half squares to draw

For example, the drawing below would be correct output.

```
spiral(aTurtle, 20, 10)
```



Question 12 (20 points)

Write a function named *lineStats*. The function *lineStats* takes two string parameters: the name of an input file and the name of an output file.

The function *lineStats* should read and analyze each line of the input file and write two statistics about the line to a corresponding line of the output file. The statistics to compile for each line are

1) the number of unique words
2) the number of unique characters (including the space character but not including the endline character)

Upper and lower case characters are the same ('At' and 'at' are the same word, for example). The input file contains only upper and lower case letters and white space.

If the input file contains the following lines:

Step right up step right up step right up
Everyone a winner bargains galore

Then an output file with the following lines would be correct:

words 3 chars 11
words 5 chars 15

Question 13 (20 points)

Write a function named *vowelEndings* that takes a string, *text*, as a parameter.

The function *vowelEndings* returns a dictionary *d* in which the keys are all the vowels that are the last letter of some word in *text*. The letters a, e, i, o and u are vowels. No other letter is a vowel. The value corresponding to each key in *d* is a list of all the words ending with that vowel. No word should appear more than once in a given list. All of the letters in *text* are lower case.

The following is an example of correct output:

```
>>> t = 'today you are you there is no one alive who is you-er than you'
>>> vowelEndings(t)
{'u': ['you'], 'o': ['no', 'who'], 'e': ['are', 'there', 'one', 'alive']}
```