CS 100
Final: Practice Exam 2

Multiple choice questions. 4 points each.

Question 1

```
def analyzeString(s):
    d = {}
    for character in s:
        chrCount = s.count(character)
        if chrCount not in d:
            d[chrCount] = [character]
        else:
            d[chrCount].append(character)
    return d

print(analyzeString('indeed'))
```

a) `2`

b) `4`

c) `{1: ['i', 'n'], 2: ['d', 'e', 'e', 'd']}`

d) `{'i': 1, 'n': 1, 'd': 2, 'e': 2}`

e) none of the above

Question 2

```
def notIn(a, b):
    indexA = 0
    while True:
        if b not in a[indexA:]:
            return a[indexA:]
        else:
            indexA += 1
    return ""

firstString = 'aha'
secondString = 'a'
print(notIn(firstString, secondString))
```

a) this is an infinite loop that never returns

b) `""` (the empty string)

c) `a`

d) `None`

e) none of the above

Question 3

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

a) `z`
b) `o`
c) `['one','zero']`
d) `['zero','one']`
e) none of the above

Question 4

```
def syms(text):
    wordList = text.split()
    rtnList = []
    for i in range(len(wordList)):
        if wordList[i] == wordList[-i-1]:
            rtnList.append(wordList[i])
    return rtnList

s = 'all for one and one for all'
print(syms(s))
```

a) `['all', 'for', 'one', 'and', 'one', 'for', 'all']`
b) `['all', 'for', 'one', 'one', 'for', 'all']`
c) `['all', 'for', 'one', 'and']`
d) `['all', 'for', 'one']`
e) none of the above

Question 5

```
burns = ["a", "man's", "a", "man", "for", "all", "that"]
out = []
for i in range(len(burns)-2):
    if burns[i] == burns[i+2]:
        continue
    elif burns[i][0] in burns[i+2]:
        out.append(burns[i])
        break
    else:
        out.append(burns[i+2])

print(out)
```

a) `[]`
b) `IndexError: list index out of range`
c) `["man"]`
d) `["man's"]`
e) none of the above

Question 6

```
input = 'boohoo'
output = ""
for i in range(2):
    output += input[i]
    for j in range(1):
        output += input[j]
print(output)
```

a) bb

b) bbo

c) bbobb

d) bboobo

e) none of the above

Question 7

```
beatles50 = "You say you want to have a revolution"
def makeDict(t):
    wordList = t.split()
    d = {}
    for word in wordList:
        final = word[-1]
        if final not in d:
            d[final] = [word]
        else:
            d[final].append(word)
            break
    return d

print(len(makeDict(beatles50)))
```

a) 7

b) 8

c) 1

d) 2

e) none of the above

Question 8

```
bools = [False, True, True and not False, not False, True or not True, True or False]

output = 1
for expr in bools:
    if expr:
        output *= 2
    else:
        output += 1
print(output)
```

a) 8

b) 11

c) 64

d) 96

e) none of the above

Question 9

For this question assume that a file named *len.txt* exists and has the following content

Everybody knows that the dice are loaded
Everybody rolls with their fingers crossed

```
def repeatCount(inFile):
    inF = open(inFile)
    text = inF.read().split()
    words = []
    repeats = 0
    for word in text:
        if word not in words:
            words.append(word)
        else:
            repeats += 1
    inF.close()
    return repeats

print(repeatCount('len.txt'))
```

a) 2

b) 3

c) 12

d) 13

e) none of the above

Question 10

```
karl = 'they have a world to win'
freq = {}
for thing in karl.split():
    freq[len(thing)] = karl.count(thing)
print(len(freq))
```

a) `5`

b) `6`

c) `24`

d) `TypeError: object of type 'int' has no len()`

e) none of the above

Question 11A (12 points)

In a university we structure course offerings into sections (e.g. CS100_002). Each section has a list of zero or more students enrolled in it. Write a definition line for a class named *Section* and a one-line docstring that describes what a section is.

Write definitions for the following methods in the *Section* class:

1. An **initialization** method. The initialization method should:
   - take a single parameter of type string, *section_id*, and assign it to the instance attribute *section_id* of the section being created
   - create an instance attribute named *enrolled_students* for the section being created and initialize it to the empty list
2. A method named ***enroll***. This method should take the name of a student as a string parameter and add it to the list of students for that section. You may assume that every name is in "First Name Last Name" format and that every name is unique.
3. A method named ***is_enrolled***. This method should take the name of a student as a string parameter. If the student is in the section's list of enrolled students, *is_enrolled* should return `True`, otherwise it should return `False`.

Question 11B (8 points)

Assume that the code for the class *Section* (Question 11A) has been saved in a file named *section.py*. Write code that performs the following tasks (each task takes one line):

1. import the module that defines the class *Section*
2. create a section with ID `Math111_101`
3. enroll students `Joe Josephson` and `Mary Smith` in `Math111_101`
4. check whether `Mary Josephson` is enrolled in `Math111_101` and print the result

Question 12 (20 points)

Write a function named *inverse* that takes a single parameter, a dictionary. In this dictionary each key is a student, represented by a string. The value of each key is a list of courses, each represented by a string, in which the student is enrolled.

The function *inverse* should compute and return a dictionary in which each key is a course and the associated value is a list of students enrolled in that course.

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

```
>>> student_courses = {'Elise':[],'Nelson':['CS100','MATH111'],'Justin':['CS100']}
>>> print(inverse(student_courses))
{'CS100': ['Nelson', 'Justin'], 'MATH111': ['Nelson']}
```

Question 13 (20 points)

Write a function named *fileStats*. The function *fileStats* 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 function *fileStats* should read and analyze each line of the input file and write two statistics, separated by a space, about the line to a corresponding line of the output file.

The two statistics for each line are:

1. the number of words
2. the number of digits (digits are the characters in positive integers: 0, 1, 2, ..., 9)

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

```
February has 28 days or 29 in a leap year
January and March have 31 days
April has 30 days
```

Then the function call:

```
fileStats('monthStats.txt', 'monthStatsOut.txt')
```

should produce an output file *monthStatsOut.txt* with the following content:

```
10 4
6 2
4 2
```