CS 100
Final: Practice Exam 3

Multiple choice questions. 4 points each.

Question 1

```
greeting = 'Hi ya!'

def greet(greeting):
    print(greeting)

print(greeting)
greet('Hello')
```

a) `Hi ya!`
   `Hi ya!`

b) `Hi ya!`
   `Hello`

c) `Hello`
   `Hi ya!`

d) `Hello`
   `Hello`

e) none of the above

Question 2

```
subjects = ['Superman', ['Batman', 'Robin'], ['Blossom', 'Bubbles', 'Buttercup']]
print(subjects[1:][0][1])
```

a) `a`

b) `l`

c) `o`

d) `u`

e) none of the above

Question 3

```
foods = ['soup', 'waffle', 'pizza']
for food in foods:
    food[0].upper()
print(foods[0], foods[1], foods[2])
```

a) `SOUP waffle pizza`

b) `soup waffle pizza`

c) `Soup Waffle Pizza`

d) `SOUP WAFFLE PIZZA`

e) none of the above

Question 4

```
swift = 'May you live all the days of your life.'
match = swift.count("you")
words = swift.split()
for i in range(0, len(words)):
    if i == len(words[i]):
        match += 1
print(match)
```

a) `1`

b) `2`

c) `3`

d) `4`

e) none of the above

Question 5

```
def pantsOnFire(words, length):
    index = 0
    while index < length:
        if len(words[index]) != 1:
            index += 1
            continue
        index *= 4
    return index

aesop = ['There', 'is', 'no', 'believing', 'a', 'liar', 'even', 'when', 'he', 'speaks',
'the', 'truth']
print(pantsOnFire(aesop, len(aesop)))
```

a) `3`

b) `12`

c) no return because this is an infinite loop

d) `IndexError: list index out of range`

e) none of the above

Question 6

```
def counterbalance(aList):
    left = 0
    right = -1
    for element in aList:
        if aList[left] != aList[right]:
            return element
        left += 1
        right -= 1
    return None

bools = [1 != 1, True, not not True, False and not True, True or not True, False]
print(counterbalance(bools))
```

a) `1 != 1`
b) `not not True`
c) `False`
d) `True`
e) none of the above

Question 7

```
print(int('3.141593'))
```

a) `IndexError: string index out of range`
b) `NameError: name 'int' is not defined`
c) `TypeError: int() argument must be a number`
d) `ValueError: invalid literal for int() with base 10: '3.141593'`
e) none of the above

Question 8

```
def symCharacter(s):
    if len(s) % 3 == 2:
        return None
    else:
        mid = len(s) // 2
        for i in range(mid):
            if s[i] == s[-i-1]:
                s *= 2
                break
            elif s[mid] in s[i:]:
                continue
            return s[i]
    return s

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

a) `None`
b) `g`
c) `gigglesgiggles`
d) `giggles`
e) none of the above

Question 9

```
classification = [{1:'Tiger'}, {2:'Giraffe'}, {3:'Peacock'}]
print((classification[1][0]))
```

a) `G`

b) `T`

c) `KeyError: 0`

d) `KeyError: 1`

e) none of the above

Question 10

The lines below are the content of the file named *`me.txt`*.

```
it's all about me me me
it's all about me me me
```

What is the output after the execution of the following code?

```
def wordLineCount(inFile):
    accum = 0
    inF = open(inFile, 'r')
    for line in inF:
        uniqueWords = []
        for word in line.split():
            if word not in uniqueWords:
                uniqueWords.append(word)
                accum += 1
    inF.close()
    return accum

print(wordLineCount('me.txt'))
```

a) `4`

b) `5`

c) `8`

d) `12`

e) none of the above

Question 11A (12 points)

A college directory consists of the names and email addresses of all students (one email address per student).

Define a *CollegeDirectory* class, write a docstring for the class, and define the following three methods:

1. An **initialization** method. The initialization method should:
   - take a string parameter, *name*, and assign it to the instance attribute *name* of the college directory being created
   - create an instance attribute named *students* for the college directory being created and initialize it to the empty dictionary
2. A method named ***addStudent***. The method *addStudent* should take two string parameters:
   - *studentName* (assume that every name is unique)
   - *emailAddress*

   The method *addStudent* should add *studentName* as a key in *students* and *emailAddress* as its value.
3. A method named ***getEmailList***. This method should return a list of all enrolled students and their email. Each student name should be separated from the student's email address by a space and the email should be delimited by angle brackets. For example, the following would be a correct entry in a returned email list: `"Hester Prynne <hprynne@hawthorne.edu>"`.

Question 11B (8 points)

Assume that correct code for the class *CollegeDirectory* in Question 11A has been saved in a file named *college_directory.py*. Write code that uses the class and method definitions to perform the following tasks:

1. import the *CollegeDirectory* module
2. create a college directory for a college named `Hawthorne`
3. add students `Hester Prynne` (email address `hprynne@hawthorne.edu`) and `Roger Chillingworth` (email address `rchillingworth@hawthorne.edu`) to the directory of Hawthorne College
4. print out the email list for Hawthorne College

Question 12 (20 points)

Write a function named *starCounter* that takes three parameters:

1. a dictionary named *starDictionary*. Each key in *starDictionary* is the name of a star (a string). Its value is a dictionary with two keys: the string `'distance'` and the string `'type'`. The value associated with key `'distance'` is the distance from our solar system in light years. The value associated with key `'type'` is a classification such as `'supergiant'` or `'dwarf'`.
2. a number, *maxDistance*
3. a string, *classification*

The function *starCounter* should compute the number of stars in *starDictionary* that are within *maxDistance* of our solar system and are of type *classification* and return that number. Note that the word "within" means "inside"; as such, it would not be inclusive of the boundary.

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

```
>>> starDictionary = {'Polaris': {'distance': 430, 'type': 'supergiant'}, 'Alpha
Centauri': {'distance': 4.37, 'type': 'spectral'}}
>>> print(starCounter(starDictionary, 10, 'spectral'))
1
```

Question 13 (20 points)

Write a function named *lineStats*. The function *lineStats* takes three 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
3. *threshold*, a non-negative integer

The function *lineStats* 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 that contain at least *threshold* characters. A word is defined as a sequence of characters, including letters, digits and punctuation marks, separated from the next word by whitespace characters (space, tab, or newline).
2. the number of capitalized words

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

```
Go ask Alice when she's ten feet tall
And call Alice when she was just small
Go ask Alice, I think she'll know
```

Then the function call:

```
lineStats('alice.txt', 'aliceStats.txt', 4)
```

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

```
5 2
5 2
4 3
```