COSC 101, Final Exam
Fall 2019

Honor Code

I agree to comply with the spirit and the rule of the Colgate University Academic Honor Code during this exam and throughout the final exam period. I will not discuss the contents of this exam with other students until the exam period is over, and affirm that I have neither given or received inappropriate aid on this exam.

Signature: _____

Printed Name: _____

Write your name; do not open the exam until instructed to do so.

You have 120 minutes to complete this exam.

There are 7 questions and a total of 90 points available for this exam. Don't spend too much time on any one question.

Since indentation is important in Python, please be sure that your use of indentation is obvious for any code you write.

If you want partial credit, show as much of your work and thought process as possible.

If you run out of space for answering a question, you can continue your answer on one of the scrap pages at the end of the exam. If you do so, be sure to indicate this in two places: (1) below the question, indicate which scrap page contains your answer, and (2) on the scrap page, indicate which question you are answering.

Question:	1	2	3	4	5	6	7	Total
Points:	5	10	10	15	15	15	20	90
Score:								

1. (5 points) Assume that the following statements have already been executed:

```
a=1.6
b='Fall2019'
c=3
```

For each of the following expressions, evaluate the expression and write the resulting value, or identify the error in the code that would prevent it from running.

(a) `b*2`

Solution:

```
'Fall2019Fall2019'
```

(b) `type(round(a, 2))`

Solution:

```
<type 'float'>
```

(c) `int(b)`

Solution:

```
Error: invalid literal for int() with base 10: 'Fall2019'
```

(d) `str(round(a)+len(b))`

Solution:

```
'10'
```

(e) `c*2%3+a-2**c**2//10`

Solution:

```
-49.4
```

2. (10 points)

3. (10 points) Assume that the following statements have already been executed:

```
a = [5, 6, 'string', ['3', 4, 9]]
b = {2:3, 5:6, 8:4}
c = (1,2,3,4,5)
```

For each of the following expressions, evaluate the expression and write the resulting value, or identify the error in the code that would prevent it from running.

(a) `a[-1:][0][2] - a[0]`

Solution:

```
4
```

(b) `a[1] in b`

Solution:

```
False
```

(c) `c.append(6)`

Solution:

```
Error: 'tuple' object has no attribute 'append'
```

(d) `a[2] * 3`

Solution:

```
'stringstringstring'
```

(e) `c[2] == b[2]`

Solution:

```
True
```

(f) `len(c) + len(b) - len(a)`

Solution:

```
4
```

(g) `sum(b)`

Solution:

```
15
```

(h) `max(b.values())`

Solution:

```
6
```

(i) `a[2].split('i')`

Solution:

```
['str', 'ng']
```

(j) `a[3] + list(b)`

Solution:

```
['3', 4, 9, 2, 5, 8]
```

4. (a) (5 points) What is the output of the following program?

```
def f3(a, b):
    s = ""
    for c in a:
        if c != b:
            s = c + s
    return s

s1 = "banana"
s2 = "a"
print(f3(s1,s2))
```

Solution:

```
nnb
```

(b) (5 points) What is the output of the following program?

```
def f2(x):
    new_list = []
    for (k,v) in x.items():
        new_list.append(v[-1])
        x[k] = new_list

a = [5, 6, 7]
b = [3, 9]
c = [11]
d = {1: a, 2: b, 3: c}
f2(d)
print(d)
```

Solution:

```
{1: [7, 9, 11], 2: [7, 9, 11], 3: [7, 9, 11]}
```

(c) (5 points) What is the output of the following program?

```
def f1(L):
    if len(L) <= 1:
        return L
    x = L[-1] + L[0]
    y = L[1:-1]
    return [x] + f1(y)

alist = [1, 2, 3, 4]
blist = f1(alist)
print(alist)
print(blist)
```

Solution:

```
[1, 2, 3, 4]
[5, 5]
```

5. (15 points) For this problem, select one line of code from each of the pairs of lines of code below and reorder them to solve the following problem:

> Write a function `meldict` that takes two dictionaries as parameters and combines them into a single dictionary. The newly combined dictionary should be returned. Since there may be keys in common to each dictionary, each value in the new dictionary should be a list containing all values that are related to a particular key. In addition, your function should make sure that the list items (values for each key) are in sorted order.
>
> For example, `meldict({'a':4, 'b':2, 'c':3}, {'a':1, 'c':7})` should return the dictionary `{'a':[1,4], 'b':[2], 'c':[3, 7]}`.

A1
```
            newdict[k].append(v)
```
A2
```
            newdict[k] = newdict[k] + v
```

B1
```
        else:
```
B2
```
        elif:
```

C1
```
    return newd
```
C2
```
    return newdict
```

D1
```
            newdict[k].sort()
```
D2
```
            sorted(newdict[k])
```

E1
```
        if v in newdict:
```
E2
```
        if k in newdict:
```

F1
```
        newdict[v] = [ d1[k] ]
```
F2
```
        newdict[k] = [ d1[k] ]
```

G1
```
def meldict(d1, d2):
```
G2
```
def meldict(d):
```

H1
```
    for k in d2.keys():
```
H2
```
    for k,v in d2.items():
```

I1
```
    newd = []
```
I2
```
    newdict = {}
```

J1
```
            newdict[k] = [ v ]
```
J2
```
            newdict[v] = [ k ]
```

K1
```
    for k in d1:
```
K2
```
    for k in d1.values():
```

Select only 11 lines of code from above, and only one line from each pair. You may fill

in line identifiers (*e.g.*, E2) below, or write out the code.

Solution:

- G 1
- I 2
- K 1
- F 2
- H 2
- E 2
- A 1
- D 1
- B 1
- J 1
- C 2

6. (15 points) Your friend claims they can hack into anyone's account and you want to prove them wrong. Write a function `hack_accounts` that your friend will use to attempt their hack. The function takes as a parameter a dictionary of users, where the keys are usernames and the values are passwords. For example:

```
users = {'rdougherty':'1l2k5a?', 'hsamadian':'l3m7l1!',\
             'mesmith':'abc123!', 'jsommers':'9s4v7b#'}
```

The function should allow your friend up to three guesses of the password for each user in the dictionary. If they successfully guess the password of any user the function will display a congratulatory message and return `True`. If they are unable to guess any password correctly the function display a failure message and return `False`.

Solution:

```
def hack_accounts(users):

    for username in users:
        print("Attempting to hack user " + username + ":")

        password = input("Password: ")
        attempt = 1
        while password != users[username] and attempt < 3:
            attempt += 1
            password = input("Password: ")

        if password != users[username]:
            print("Unable to access system as " + username + ".\n")
        else:
            print("Successfully logged in as " + username + ".")
            return True

    print("\nUnable to hack into system.")
    return False
```

7. (20 points) You are tasked with processing a given data set about surface sea ice concentrations. The data is stored in a file called `concentrations.bin`, and the data is in the following format:

$$x, y, day, concentration$$

where x, y are the surface coordinates (in degrees; x is latitude, y is longitude), day is an integer between 0 and 364 (indicating the day number), and $concentration$ is a value between 0 and 1, representing that location's ice coverage for that day. For example, the file can look like this (where each point and each day is separated by a new line):

```
0,1,0,0.5
0,1,1,0.8
0,1,2,0.3
... <suppresed lines> ...
0,1,364,0.67
0,2,0,0.8
```

You can assume that each x, y coordinate will have 365 lines (one for each day). We will determine which of these points have very small year-round concentration and are physically close to each other.

Write a function called `extractCloseOutliers` that takes as parameter `eps` (a number between 0 and 1) and `threshold` (a positive number), where you will perform the following tasks:

- Read the contents of the file `concentrations.bin`.
- Create a list L of coordinates with average concentration at most `eps` in value at least 300 days of the year.
- Among these coordinates, find all pairs of coordinates c_1, c_2 with distance at most `threshold`. Here, the distance between c_1, c_2 is the Euclidean distance: if c_1 has coordinates (x_1, y_1) and c_2 has coordinates x_2, y_2, then the distance is $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.
- Write all of these pairs of coordinates to a file `closePoints.txt`, where each line written is a space-separated list of the coordinates. If the pair is $c_1 = (x_1, y_1)$ and $c_2 = (x_2, y_2)$, you will write $x_1\ y_1\ x_2\ y_2$ to the file.

Solution:

```
import math

def splitCoords(f):
    d = {}
    for line in f:
        data = line.split(',')
        coord = (int(data[0]), int(data[1]))
        conc = float(data[-1])
```

```
        if coord in d:
            d[coord].append(conc)
        else:
            d[coord] = [conc]
    return d


def lowConcCoords(coords_dict, eps):
    L = []
    for coord in coord_dict:
        if max(coord_dict[coord]) <= eps:
            L.append(coord)
    return L


def distance(c1, c2):
   return math.sqrt( (c1[0] - c2[0])**2 + (c1[1] - c2[1])**2)


def findPairs(L, threshold):
    pairs = []
    for i in range(len(L)-1):
        for j in range(i+1, len(L)):
            if distance(L[i], L[j]) <= threshold:
                pairs.append((L[i], L[j]))
    return pairs


def writePairs(pairs):
    f = open("closePoints.txt", 'w')
    for c1,c2 in pairs:
        f.write(f'{c1[0]} {c1[1]} {c2[0]} {c2[1]}\n')
    f.close()


def extractCloseOutliers(eps, threshold):
    f = open('concentrations.bin', 'r')
    coords_dict = splitCoords(f)
    L = lowConcCoords(coords_dict, eps)
    pairs = findPairs(L, threshold)
    writePairs(pairs)
    f.close()
```

(This page is intentionally blank. Label any work with the corresponding problem number.)

(This page is intentionally blank. Label any work with the corresponding problem number.)