COSC 101, Final Exam
Spring 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	15	15	10	15	20	90
Score:								

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

```
p = 8
q = 6.7
r = 4
```

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) `p / r`

Solution:

```
2.0
```

(b) `min(round(q), p)`

Solution:

```
7
```

(c) `str(p) * int( q // r )`

Solution:

```
'8'
```

(d) `int(q) % p`

Solution:

```
6
```

(e) `int(q) // (p - 2 * r)`

Solution:

```
Error: integer division by zero
```

2. (10 points) Assume that the following statements have already been executed:

```
a = 'Spring2019'
b = [ a[1], a[-3:], a[-2], a.split('i')]
c = { a : 4, 'a' : 3, b : 5, 'd': 2, 'e': -2}
```

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) `c[a]`

Solution:

```
4
```

(b) `('Spr' in b) == False`

Solution:

```
True
```

(c) `a[-20::-2]`

Solution:

```
''
```

(d) `c.get('Spring2019',2)`

Solution:

```
4
```

(e) `4 in c.values()`

Solution:

```
True
```

(f) `'Spring2019'[-7::-1]`

Solution:

```
'irpS'
```

(g) `'*'.join(b[:2])`

Solution:

```
'p*019'
```

(h) `b[ c[ a[ c['e'] ] ] - c[b[1][0]]]`

Solution:

```
Error
```

(i) `b[-1][1][2:]`

Solution:

```
2019
```

(j) `sum(c.values())`

Solution:

```
12
```

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

```
def g(lst):
    if len(lst) < 2:
        return []
    else:
        if lst[0]==len(lst):
            return [lst[0] * 2] + g(lst[2:])

print(g([5, 4, 3, 2, 1]))
```

Solution:

```
[10, 6]
```

(b) (5 points) What is printed by running the code?

```
def func(dictIn, s):
    print(d,s)
    for chr in s:
        dictIn[chr]=chr
        s=s+'1'
    return dictIn,s

s='ab'
d={'a':1, s:2}
print(func(d,s))
print(d,s)
```

Solution:

```
{'a': 1, 'ab': 2} ab
({'a': 'a', 'ab': 2, 'b': 'b'}, 'ab11')
{'a': 'a', 'ab': 2, 'b': 'b'} ab
```

(c) (5 points) What is printed by the following program?

```
def a(lst):
    strt = []
    for i in range(len(lst)):
        if [i] == lst[i:i+1]:
            strt += [i]
    return strt

print(a([2, 1, 0, 3, 0]))
```

Solution:

```
[1, 3]
```

4. (15 points total) This question has three parts based on the following function.

```
def myFunction(s):
    n = "" #empty string
    for ch in s.lower():
        if ch not in 'aeiou':
            n += ch
    return n
```

(a) (5 points) What does `myFunction('All students')` return?

Solution: `'ll stdnts'`

(b) (5 points) Rewrite `myFunction(s)` using a **`while`** loop instead of a **`for`** loop.

Solution:

```
def myFunction_while(s):
    n = "" #empty string
    i = 0
    while i < len(s):
        if s[i].lower() not in 'aeiou':
            n += s[i]
        i += 1
    return n
```

(c) (5 points) Rewrite `myFunction(s)` *recursively*, using no loops.

Solution:

```
def myFunction_recursive(s):
    if len(s) == 0:
        return ""
    r = v(s[1:])
    if s[0].lower() not in 'aeiou':
        r = s[0] + r
    return r
```

5. (10 points total) This question has two parts.

   (a) (5 points) Write a function `removerepetitions` that receives a list which may contain repeated items and removes the repeated items (keeps only one instance of each item) and returns `None`. You must use nested for loop.

```
Example:
myList=[1,2,1,3,2,2]
removerepetitions(myList)
print(myList)     # has to print [1,2,3]


def removerepetitions (lst):
    #your code here
```

Solution:

```
def removerepetitions(lst):
    copy = lst[:]
    for temp in copy:
        for i in range(lst.count(temp) - 1):
            lst.remove(temp)
```

(b) (5 points) Write the same function in a different way. This time the function has to return a new list instead of updating the same list. You must not use nested for loop. Instead, write an auxiliary function `removeItem(item, lst)` that removes all repetitions of a given item is a given list (keeps one instance) and use it in your function `removerepetitions(lst)`.

```
Example:
myList = [1,2,1,3,2,2]
print(removerepetitions(myList))    # has to print [1,2,3]
print(myList)                       # has to print [1,2,1,3,2,2]
```

Solution:

```
def removeItem(item, lst):
    for i in range(lst.count(item) - 1):
            lst.remove(item)


def removerepetitions(lst):
    copy = lst[:]
    for temp in lst:
        removeItem (temp, copy)
    return copy
```

6. (15 points) Write a function `rearrange()` that receives a dictionary parameter *rawdict* (example below).

```
rawdict ={('Nike', 'Puma'): 'Sport', ('Burger King',):'Food',
          (1, 1.2, 5):'Num', ('Adidas',):'Sport'}
```

The function should use the information in *rawdict* to create and return a new dictionary *labels* (example below).

```
labels ={'Sport':['Nike','Puma','Adidas'], 'Food':['Burger King'],
          'Num':[1,1.2,5]}
```

In *rawdict*, keys are tuples and values are strings describe the items in the tuple. In *labels*, keys are the label values from *rawdict* and values are the list of all items which are associated with that label.

After this, write a function `summarize()` which receives a dictionary parameter in the format returned by `rearrange()`. The function should return create and return another dictionary *summary* (example below).

```
summary = {'Sport': [('Nike', 1), ('Puma', 1), ('Adidas', 1)],
         'Food': [('Burger King', 1)], 'Num': [(1, 1), (1.2, 1), (5, 1)]}
```

In this dictionary, the keys are the labels and the values are lists of tuples with item names and counts of times each item appeared with the corresponding label.

You should write at least one *helper function* to keep the amount of code in `summarize()` short and modular.

Solution:

```
def rearrange(rawdict):
    labels = {}
    for key, value in rawdict.items():
        if value in labels:
            labels[value] += list(key)
        else:
            labels[value] = list(key)
    return labels

def countrepeated(lst):
    visited = []
    result = []
    for i in lst:
        if i not in visited:
            result.append((i, lst.count(i)))
            visited.append(i)
    return result
```

```
def summarize(labels):
    result = {}
    for key, val in  labels.items():
        result[key] = countrepeated(val)
    return result
```

7. (20 points total) This question has two parts and is continued on the following pages.

   Consider a weather data file in which each line of the file is either the name of a location (in the format `"#City, STATE"`), temperature information for the above city (in the format `"Month day, low, high"`), or a blank line. For example, a short weather data file `weather.txt` contains the following:

```
# Hamilton, NY
December 4, 27, 48
December 5, 41, 52
December 6, 33, 39


# Williamstown, MA
December 4, 24, 28
December 5, 42, 55

December 6, 26, 55

# Chicago, IL
December 4, 44, 65
December 5, 27, 44
December 6, 26, 35
```

   According to this file, the high temperature in Williamstown on December 6th was 55 and the low in Chicago on December 5th was 27.

   Your program will be written as multiple functions, described on the following pages.

(a) (10 points) Write a function `get_temps` that takes as a parameter the name of a weather data file. This function will return a dictionary where the keys are locations and the values are lists containing all of the temperatures from that location. For example:

```
>>> print( get temps ('weather.txt'))
{'Hamilton , NY' : [27, 48, 41, 52, 33, 39],
'Williamstown , MA' : [24, 28, 42, 55, 26, 55],
'Chicago , IL': [44, 65, 27, 44, 26, 35]}
```

Your function should return an empty dictionary if the file is not found. (Hint: Python raises a `FileNotFound` error if it is unable to open a file.) You can assume that opened files will be in the format described on the previous page.

Solution:

```
def get_temps(filename):
    try:
        d = {}
        f = open(filename, 'r')
        for line in f:

            if line.startswith("#"):
                k = line.strip("# ")
                d[k] = []

            elif line:
                low = int(line.split(',')[-2].strip())
                high = int(line.split(',')[-1].strip())
                d[k].append(low)
                d[k].append(high)

        f.close()
        return d

    except FileNotFoundError:
        return {}
```

(b) (10 points) Write a function `temp_report` that takes as a parameter the name of a weather data file. This function will return a string 'report' describing the data from the file. For example:

```
>>> temp report('weather.txt'))
Of the 3 locations: Chicago, IL was the warmest with a high
temperature of 65 and Williamstown, MA was the coldest
with a low temperature of 24.
```

You are required to use the `get_temps` function from part (a). You can assume the function works as described (regardless of whether your answer is correct or not). You must also write at least one additional helper function to use in your `temp_report`.

Solution:

```
def get_highlow(temps):
    high, low = temps[0], temps[0]
    for t in temps:
        if t > high:
            high = t
        if t < low:
            low = t
    return high, low

def temp_report(filename):
    all_temps = get_temps(filename)

    # accumulators
    best_high_loc = all_temps.keys()[0]
    best_low_loc = all_temps.keys()[0]
    best_high, best_low = get_highlow(all_temps[best_high_loc])

    for loc in all_temps:
        loc_temps = all_temps[loc]
        loc_high, loc_low = get_highlow(loc_temps)

        if loc_high > best_high:
            best_high = loc_high
            best_high_loc = loc

        if loc_low < best_low:
            best_low = loc_low
            best_low_loc = loc

    print(f'Of the {len(all_temps)} locations: {best_high_loc}
        was the warmest with a high temperature of {best_high}
        and {best_low_loc} was the coldest with a low
        temperature of {best_low}')
```

(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.)