CSE 160 - Final Exam Solution

Autumn 2023

Name: _____

UW Email: _____@uw.edu

- **Do not open this exam before the exam begins and close the booklet when time is called. Starting early or working after time is subject to a deduction.**
- You have **110 minutes** to complete this exam.
- This exam contains 10 pages (including this page) and 8 questions (7 exam questions, 1 extra-credit question).
- A Python syntax reference sheet is provided that includes documentation that we have covered in class.
- You are additionally allowed a 8.5" x 11" double-sided cheat sheet. All other materials besides writing utensils should be put away before the exam starts. This includes all devices like phones, calculators, and smartwatches.
- You may only use parts and features of Python that have been covered in class.
- All questions assume Python version 3.7, as we have been using all quarter.

Question	Points	Score
1	4	
2	5	
3	5	
4	6	
5	4	
6	5	
7	6	
Total:	35	

1. Consider the following code:

```
def mystery(lst):
    result = [i ** 2 if i % 2 == 1 else i // 2 for i in lst]
    return result
```

For the below problems, you will be asked to provide a call to the `mystery()` function. One example call of this function is: `mystery([0, 0])`, where the parameters provided inside of the function can differ.

(a) (2 points) Provide an example call to the `mystery()` function that would return the list `[7, 49, 0, 1]`.

There are a variety of answers to this problem. A few are listed below:

```
mystery([14, 7, 0, 1])
mystery([14, 98, 0, 2])
```

(b) (2 points) Provide an example call to the `mystery()` function that would return the list `[25, 9, 16, 81]`.

There are a variety of answers to this problem. A few are listed below:

```
mystery([5, 3, 32, 9])
mystery([50, 18, 32, 162])
```

2. (5 points) Write a function `set_combination` that takes in a list of lists `list_of_lists` and returns a set where the elements appear at least once in an inner list.

   For example:

   - `set_combination([[]])` should return `set()` (an empty set).
   - `set_combination([[1], [2], [3]])` should return `{1, 2, 3}`.
   - `set_combination([[2, 4], [4, 6], [6, 8]])` should return `{2, 4, 6, 8}`.

```
def set_combination(list_of_lists):
    # Your code goes here

    common_set = set()
    for lst in list_of_lists:
        common_set = common_set | set(lst)
    return common_set
```

3. Consider the following class definition:

```
from operator import itemgetter

class MusicShow:
    def __init__(self, show_name):
        self.show_name = show_name
        self.performers = []

    def add_artist(self, artist, members):
        self.performers.append((artist, members))
        print(self.show_name, "added", artist)

    def print_top_list(self, num=3):
        top_list = sorted(self.performers, key=itemgetter(0), reverse=True)
        top_list = sorted(top_list, key=itemgetter(1))
        print(top_list[:num])
```

(a) (2 points) For the below code snippet, write out the expected printed output.

```
music_bank = MusicShow("Music Bank")
music_bank.add_artist("wayv", 6)
music_bank.add_artist("enhypen", 7)
music_bank.add_artist("ive", 6)
music_bank.add_artist("ateez", 8)
music_bank.print_top_list()
```

```
Music Bank added wayv
Music Bank added enhypen
Music Bank added ive
Music Bank added ateez
[("wayv", 6), ("ive", 6), ("enhypen", 7)]
```

(b) (3 points) For the below code snippet, write out the expected printed output.

```
the_show = MusicShow("The Show")
artists = ["iu", "red velvet", "sejeong", "le sserafim", "aespa"]
members = [1, 5, 1, 5, 4]

for i in range(len(artists)):
    the_show.add_artist(artists[i], members[i])

the_show.print_top_list(4)
```

```
The Show added iu
The Show added red velvet
The Show added sejeong
The Show added le sserafim
The Show added aespa
[("sejeong, 1), ("iu", 1), ("aespa", 4), ("red velvet", 5)]
```

4. (6 points) Suppose you are given a file named `"gpa.csv"` where there are three columns in the CSV file: `student_name`, `class_name`, and `gpa`.

   You should write code to print out a dictionary where the keys are each student name in the csv file and the values are a list of tuples, where each tuple consists of two elements `(class_name, gpa)`, where `class_name` is a string and `gpa` is a float. If a student appears multiple times in the file, the dictionary's list value associated with the student will contain multiple tuples of class names with their respective GPAs.

   For example, given an example `"gpa.csv"` file that contains the following content:

```
student_name,class_name,gpa
Lucas,CSE 160,3.0
Sheamin,INFO 340,3.8
Suh Young,CSE 160,3.8
Suh Young,CLAS 205,4.0
```

   You should print out the dictionary:

```
{"Lucas": [("CSE 160", 3.0)], "Sheamin": [("INFO 340", 3.8)],
"Suh Young": [("CSE 160", 3.8), ("CLAS 205", 4.0)]}
```

```
import csv # You can use this library in your code
# Your code goes here
```

Solution using csv.DictReader:

```
f = open("gpa.csv")
gpa_csv = csv.DictReader(f)
gpa_dict = dict()

for row in gpa_csv:
    name = row["student_name"]
    class_name = row["class_name"]
    gpa = float(row["gpa"])

    gpa_dict.setdefault(name, [])
    gpa_dict[name].append((class_name,
                           gpa))

print(gpa_dict)
```

Solution not using csv.DictReader:

```
f = open("gpa.csv")
gpa_dict = dict()

line_num = 0
for line in f:
    if line_num != 0:
        line = line.split(",")
        name = line[0]
        class_name = line[1]
        gpa = float(line[2])

        gpa_dict.setdefault(name, [])
        gpa_dict[name].append((class_name,
                               gpa))
    line_num += 1

print(gpa_dict)
```

5. (4 points) A student wrote a function that calculates if the average age in `age_list` is greater than the given parameter `age`. The function returns `True` if the average age is greater than `age`, `False` otherwise.

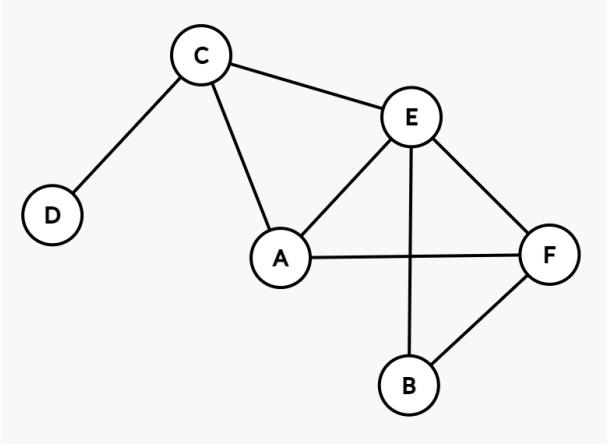
   However, the student also made some style issues. Identify four style issues that the student should fix and the corresponding change to fix the issue.

```
def checkAverageAge(age, age_list):
    sum = 0
    for curr_age in age_list:
        sum = sum + curr_age
    if sum/len(age_list) > age:
        return True
    else:
        return False
```

Style Issue	Solution
The function name, `checkAverageAge`, is not in snake case.	Change function name to `check_average_age`.
No docstring specified for function.	Add docstring describing the parameters (`age, age_list`), behavior
The variable name `sum` is also a built-in Python function.	Change variable name to another name, such as `curr_sum`.
There are no spaces between the operator `/`.	Add a space between the operator so the expression is `sum / len(age_list)`.
Bad boolean zen for last four lines of code.	Directly return `sum / len(age_list)` instead of using `if` statements.

6. (5 points) Write a function `max_edges` that takes in a graph `g` and returns a the value of the node that has the maximum edges. You can assume that only one node has the maximum amount of edges. You should not use the `max` function to solve this problem.

   For example, given this graph `g`:



   You should return `"E"`, because that node has the most amount of edges in the graph (four edges) compared to other nodes.

```
import math
import networkx as nx

def max_edges(g):
    # Your code goes here


    max_node = ""
    max_count = -math.inf

    for node in g.nodes():
        curr_count = len(set(g.neighbors(node)))
        if curr_count > max_count:
            max_node = node
            max_count = curr_count
    return max_node
```

7. Write three test cases that test **different behaviors** using `assert` statements for each of the following functions given its docstring:

   (a) (3 points) `get_max_key(my_dict)`: Given a dictionary `my_dict` where keys are strings and values are floats, returns the key that has the corresponding maximum value. You can assume `my_dict` will only contain one key with the maximum float value. Returns `None` if `my_dict` is empty.

   There are a variety of answers to this problem. A few are listed below:

```
# Tests when my_dict is empty.
assert get_max_key({}) is None

# Tests when there is one key-value pair in my_dict.
assert get_max_key({"A": 1.0}) == "A"

# Tests when there are multiple key-value pairs in my_dict.
assert get_max_key({"A": 0.5, "B": 1.0, "C": 1.5}) == "C"
```

   (b) (3 points) `count_chars(lst, char)`: Given a list of strings `lst` and a single-character string `char`, return a dictionary where the keys are each individual element in `lst` and the value is the respective count of the character `char` for each key (case-sensitive). Return an empty dictionary if `lst` is empty.

   There are a variety of answers to this problem. A few are listed below:

```
# Tests when lst is empty.
assert count_chars([], "a") == {}

# Tests when there is one element in lst, char does not occur in list
assert count_chars(["hello"], "b") == {"hello": 0}

# Tests when there is one element in lst, char occurs in list
assert count_chars(["hello"], "l") == {"hello": 2}

# Tests when there are multiple elements in list
assert count_chars(["hello", "world"], "l") == {"hello": 2, "world": 1}
```