B

Name: SOLUTION

Email Address (UW Net ID): @uw.edu

Section:

CSE 160 Winter 2023 - Final Exam

Instructions:

- You have **110 minutes** to complete this exam.
- The exam is **closed book**, including no calculators, computers, phones, watches or other electronics.
- You are allowed a single sheet of notes for yourself.
- We also provide a syntax reference sheet.
- Turn in *all sheets* of this exam, together and in the same order when you are finished.
- When time has been called, you must put down your pencil and stop writing.
    - **Points will be deducted if you are still writing after time has been called.**
- 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.
- You may ask questions by raising your hand, and a TA will come over to you.

Good luck!

Question	**Points**
Question 1	6
Question 2	6
Question 3	4
Question 4	4
Question 5	4
Question 6	4
Question 7	8
Extra Credit	1
TOTAL	**36**

1) [6 pts] For each of the blanks (marked by "# ____") at the end of the lines of code below, write a boolean expression that defines what values would need to be passed into the function (as the variable **x**) in order for the matching line of code to be reached. You can assume that the values passed in as x will always be integers. Feel free to use mathematical notation. If a branch is unreachable, write UR. For example,

```
def example(x):
    x = x + 1
    if x > 3 or x < 3:
        print('Branch 1 reached!')        # x != 2
    elif x == 2:
        print('Branch 2 reached!')        # UR
```

```
### Part A:
def part_a(x):
    x = x * 2
    if x > 0:
        print('Branch 1 reached!')          # x > 0
    elif x > 1:
        print('Branch 2 reached!')          # UR
    else:
        print('Branch 3 reached!')          # x <= 0

### Part B:
def part_b(x):
    x = x % 2
    if x > 1:
        print('Branch 1 reached!')          # UR
    elif x < 0:
        print('Branch 2 reached!')          # UR
    else:
        print('Branch 3 reached!')          # All x

### Part C:
def part_c(x):
    y = x % 5
    if y > 3:
        print('Branch 1 reached!')          # x % 5 == 4
        if x < 0:
            print('Branch 1.5 reached!')    # x % 5 == 4 && x < 0
        elif x == 0:
            print('Branch 1.75 reached!')   # UR
    else:
        print('Branch 2 reached!')          # x % 5 != 4
        z = x / 5
        if 5 * z == x:
            print('Branch 2.5 reached!')    # x % 5 == 0
        elif y == 0:
            print('Branch 2.75 reached!')   # UR
```

2) [6 pts] Write a function called **`favorite_food(filename)`** that takes the name of a file as a string parameter. Each line of the file is in the form:

```
TA_name food_item1 food_item2 ...
```

The TA_name and each food_item are strings (containing no spaces or punctuation). The number of food items given for each TA is variable (some TAs only like one food item, while some TAs like many food items). Additionally, some TAs decided to add more items they liked later to the file, so there can be multiple lines in the file with the same TA and additional favorite food items. Note, food items are unique and never repeated.

Here are the contents of a sample input file, `TAFavoriteFood.txt`:

```
Sneh Pizza Pasta
Wen Popcorn
Annalisa Sushi Cake Cookies Coffee
Max Teriyaki
Sneh Gyro
```

Your function should read in the given file and return a dictionary mapping each TA to a list of their favorite food(s). You may assume the file name provided is valid and that the file is formatted as described and includes at least one valid line.

Expected output when calling **`favorite_food("TAFavoriteFood.txt")`**:

```
{"Sneh": ["Pizza", "Pasta", "Gyro"], "Wen": ["Popcorn"], "Annalisa":
["Sushi", "Cake", "Cookies", "Coffee"], Max: ["Teriyaki"]}
```

```
def favorite_food(filename):
    # Write your code here
    file = open(filename)
    result = {}

    for line in file:
        list_line = line.split()
        TA = list_line[0]
        food = list_line[1:]
        if TA not in result:
            result[TA] = food
        else:
            result[TA].extend(food)

    file.close()

    return result
```

3) [4 pts] For each nested data structure, write the one line of code that would print the string "Dog". You can assume that "Dog" will only appear once. You must use the given data structure; writing `print("Dog")` will be marked as 0 points.

```
nested_data_1 = {"Canine": ["Puppy", "Hound", "Dog"], "Feline": ["Kitten",
"Cat"], "Swine":["Hog", "Pig"]}
```

```
print(nested_data_1["Canine"][2])
```

```
nested_data_2 = {"Johnsons": {"Butterscotch": "Cat"}, "Smiths": {}, "Garcias":
{"Guppy": "Fish", "Scout": "Dog"}}
```

```
print(nested_data_2["Garcias"]["Scout"])
```

```
nested_data_3 = [{"France": {"Population": 638000, "Area": 9870, "Pet": "Fish"}},
{"United States": {"Population": 323000000, "Area": 5870000, "Pet": "Dog"}},
{"Russia": {"Population": 143000000, "Area": 66000000, "Pet": "Cat"}}]
```

```
print(nested_data_3[1]["United States"]['Pet'])
```

```
nested_data_4 = [["Building", "Cars", "Lights", "Loud"], ["Dog", "Cow", "Sheep",
"Tractor", "Field"], ["Boat", "Fishing", "Creek", "Quiet"]]
```

```
print(nested_data_4[1][0])
```

4) [4 pts] Consider the following class definition:

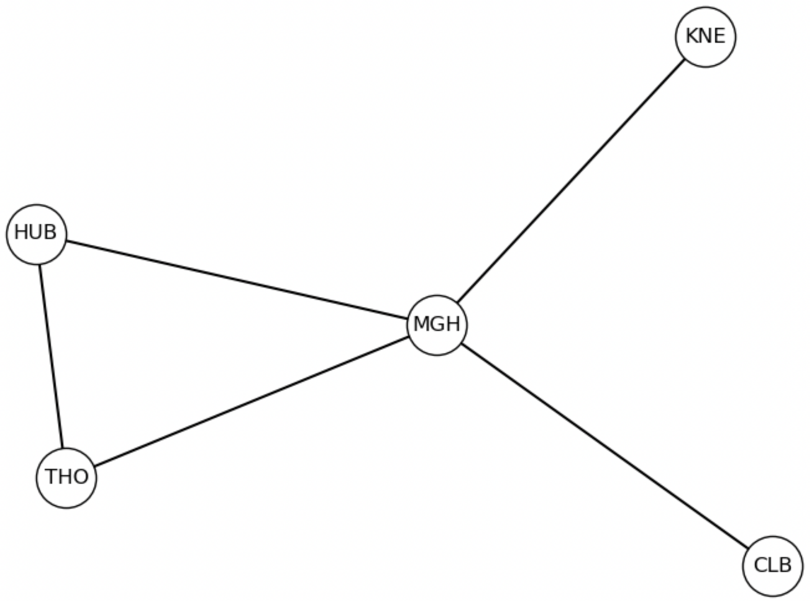
```
from operator import itemgetter
class RestaurantRating:
     def __init__(self, name):
          self.rating = {}
          print(name + "'s top restaurants list")

     def add_rating(self, restaurant_name, rating):
          self.rating[restaurant_name] = rating

     def get_rating(self):
          r1 = sorted(self.rating.items())
          r2 = sorted(r1, key=itemgetter(1), reverse=True)
          final = []
          for pair in r2:
               final.append(pair[0])
          return final
```

For each of the below code snippets, write the expected output.

```
# Snippet A
rating1 = RestaurantRating("The Ave")
rating1.add_rating("Chi-Mac", 4.0)
rating1.add_rating("Cafe on the Ave", 4.3)
rating1.add_rating("Korean Tofu House", 4.3)
print(rating1.get_rating())
# Write Snippet A's output here:

The Ave's top restaurants list
["Cafe on the Ave", "Korean Tofu House", "C hi-Mac"]

# Snippet B
rating2 = RestaurantRating("UVillage")
rating2.add_rating("Dough Zone", 4.0)
rating2.add_rating("Din Tai Fung", 4.0)
rating2.add_rating("Elemental", 4.2)
print(rating2.get_rating())
# Write Snippet B's output here:

UVillage's top restaurants list
["Elemental", "Din Tai Fung", "Dough Zone"]
```

5) [4 pts] The data file called "Building_names.txt" has been provided to you, and contains pairs of building name abbreviations. This file contains:

```
KNE    MGH
CLB    MGH
THO    MGH
HUB    MGH
HUB    THO
```



You may assume that there are only two columns like above, separated by four spaces ("whitespace"). Each line of the file represents an edge in a graph. In the space provided below, write the code to read the file, create a graph, and then plot and show the graph. "`networkx`" and "`matplotlib.pyplot`" are imported for you below. Make sure to use `.draw_networkx(graph)` to draw the graph and `.show()` to show the graph. The resulting graph should look like what's shown to the right.

```
import networkx as nx
import matplotlib.pyplot as plt

file = open("Building_names.txt")
map = nx.Graph()
for line in file:
   name = line.split()
   map.add_edge(name[0], name[1])
nx.draw_networkx(map)
plt.show()
file.close()
```

6) [4 pts] Your friend wants you to see the code that they've written. While you look at their code, you start having flashbacks of the flake8 style errors that you lost in your homework submissions for CSE 160. Given this code, circle four general style errors (not just those that flake8 would catch). Then for each style error, write a sentence (also noting the line number) explaining why it is an error (i.e., which style guideline is it violating?) and provide a possible solution.

Line Numbers

```
1  myFavoriteSongs = ["OMG", "Rocketeer", "Clarity", "Plastic Love", "Jam & Butterfly"]
2  count = 0
3  for i in range(0,len(myFavoriteSongs),1):
4      x = myFavoriteSongs[i]
5      if (len(x) > 7) == True:
6          count = count+1
7      else:
8          count = count
9  print(count, "of my favorite songs have long titles!")
```

Possible errors and their line numbers include:

- **Line 1: lower_case_snake violation. Change to my_favorite_songs**
- **Line 3: range(0,len(myFavoriteSongs),1) should be reduced to range(len(myFavoriteSongs))**
  - **OR: There needs to be spaces between each parameter**
- **Line 4: undescriptive variable name. Change to something like "song"**
- **Line 5: boolean zen violation (== True is redundant). Change to if len(x) > 7:**
- **Line 6: Spaces around operators. Change to count = count + 1 or count += 1**
- **Lines 7,8 : This else statement doesn't do anything. Remove it**

7) [8 pts]

Part A: Write a function called **`election_results`** that takes a single parameter (**`vote_counts`**, a dictionary) that maps candidate names to how many votes they received. The function should then return a list that represents the individual votes. For example if vote_counts is defined as

```
vote_counts = {"john" : 4, "johnny" : 3, "jackie" : 2, "jamie" : 4}
```

then a function call of

```
election_results(vote_counts)
```

would return the list

```
["john", "john", "john", "john", "johnny", "johnny", "johnny", "jackie",
 "jackie", "jamie", "jamie", "jamie", "jamie"]
```

The order of the returned votes does not matter.

```
def election_results(vote_counts):
    # Write your code here

    list = []

    for person in vote_counts :
        for i in range(vote_counts[person]):
            list.append(person)

    return list
```

Part B: Write **two** distinct tests, in the form `assert <condition>` (e.g., `assert 1 != 2`), that will validate that the function you wrote in Part A works as expected. Each test should test a different case than the other. Then, below those tests, write one to two sentences that describe why those test cases are different from each other.

Case 1:

```
assert election_results({}) == []
```

Explanation: empty dictionary returns empty list

Case 2:

```
assert election_results({“john”: 1}) == [“john”]
```

Explanation: One item in dictionary returns corresponding list

Extra credit (1pt): If you created your own programming language, what would you call it? Then, draw a logo for this new programming language. Any answer covering both parts will get full credit.