Name:___[Edited for CSE 160 26wi]_____

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

CSE 160 Winter 2023 - Final Exam KEY

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 up to this point.
- 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!

****Modified from original to be in scope for 26wi**

1) For each of the blanks (marked by "# ____") at the end of the lines of code below, write what value(s) would need to be passed into the function (as the variable **x**) in order for the matching line of code to be reached. 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) 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
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"], Max: ["Teriyaki"]}
```

```
def favorite_food(filename):
    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) For each nested data structure, write the one line of code that would print the string "Dog" by accessing values or indexing into the structure. You can assume that "Dog" will only appear once.

```
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) Consider the following class definition:

```
class RestaurantRating:
    def __init__(self, name):
        self.rating = {}

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

Add a method `get_top_rated` that returns the top two rated restaurants as a tuple, with the highest-rated first and the second-highest second. If there are multiple restaurants with the same highest rating, you may return any two and place them in any order. If there are less than two restaurants in the object, still return a tuple with None values.

Examples:

```
rating1 = RestaurantRating("The Ave")
rating1.get_top_rated()
>>> (None, None)
rating1.add_rating("Chi-Mac", 4.0)
rating1.get_top_rated()
>>> (“Chi-Mac”, None)
rating1.add_rating("Cafe on the Ave", 4.3)
rating1.get_top_rated()
>>> (“Cafe on the Ave”, “Chi-Mac”)
```

```
def get_top_rated(self):
    top1 = None
    top2 = None
    r1 = float('-inf')
    r2 = float('-inf')
    for restaurant, rating in self.rating.items():
        if rating > r1:
            top2 = top1
            r2 = r1
            top1 = restaurant
            r1 = rating
        elif rating > r2:
            top2 = restaurant
            r2 = rating
    return (top1, top2)
```

5) The data file called "campus_paths.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
```

Each row represents a direct path between one building on campus to another. For example, there exists a path between KNE and MGH and vice versa, but there is no path between KNE and THO. You may assume that each row in the file contains exactly two buildings separated by a space.

Read in the file and create a dictionary that maps each building to a list of other buildings that are reachable. Note that paths are bidirectional! So for the above example, your code should product the following dictionary:

```
{'KNE': ['MGH'],
 'MGH': ['KNE', 'CLB', 'THO', 'HUB'],
 'CLB': ['MGH'],
 'THO': ['MGH', 'HUB'],
 'HUB': ['MGH', 'THO']}
```

```
paths = {}
file = open("campus_paths.txt")
for line in file:
    b1, b2 = line.strip().split()
    if b1 not in paths:
        paths[b1] = []
    if b2 not in paths:
        paths[b2] = []
    paths[b1].append(b2)
    paths[b2].append(b1)
file.close()
```

6) Your friend is writing and using a class called Song to keep track of some of her favorite songs. But looking at her code, there are 4 bugs! Annotate her code to fix the bugs.

```
class Song:
      def __init__(self, title, artist):
            self.title = title
            self.artist = artist
      def title_length(self):
            return len(self.title)

song1 = Song(“OMG”, “New Jeans”)
song2 = Song(“Jam & Butterfly”, “DPR Live”)
my_favorite_songs = [song1, song2, Song(“Daylight”, “Taylor Swift”)]
count = 0
for i in range(0, len(my_favorite_songs)):
      if my_favorite_songs[i].title_length() >= 7:
            count = count + 1
print("Songs with long titles: " + str(count))
```

7) Write a function called **`election_results`** that takes a single parameter (**`vote_counts`**) and returns a list which expands the votes count.

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

election_results(vote_counts)
```

Should return

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

```
def election_results(vote_counts):
    res_list = []

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

    return res_list
```