Week-8 Graded Programming

Problem 1

Question

Write a **recursive** function `reverse` that takes a list as input and returns a list with elements in reverse order.

Function Argument	Function Return
a list of numbers and/or strings	a list in reversed order of original list
[1, 2, 3, 4]	[4,3,2,1]
['a', 'b', 'c', 'd']	['d', 'c', 'b', 'a']

```
# Recursive function to reverse a list
def reverse(input_list):
    # function body
```

Note: The program internally check for a recursive implementation of the function. The student need not accept any input or print output to the console. The student should only write the body of the function.

Test Cases

Public

Input	Output
1, 2, 3, 4	[4, 3, 2, 1]
'a', 'b', 'c', 'd'	['d', 'c', 'b', 'a']

Private

Input	Output
4, 3, 2, 1	[1, 2, 3, 4]
'd','c','b','a'	['a', 'b', 'c', 'd']
'aa', 2, 1, 'd'	['d', 1, 2, 'aa']
1, 1, 1, 0, 0, 0	[0, 0, 0, 1, 1, 1]

Answer

```python
# Recursive function to reverse a list
def reverse(input_list):
    # Base case 1 - input is an empty list, return an empty list
    if len(input_list) == 0:
        return []
    # Base case 2 - input is a list of single element, return the same list
    elif len(input_list) == 1 :
        return input_list
    # Recursive case - calls another instance of the same function with different argument
    # The first part of return contains last element of the input list
    # The second part of return calling same function on a sub list
    # the sub list is the input list with the last element removed
    return [input_list[len(input_list) - 1]] + reverse(input_list[:len(input_list) - 1])
```

Problem 2

Question

Write a **recursive** python function `max_element` that accepts a list as input and returns the maximum element stored in it. If the list only has only one value, then this value is the maximum value. The list can be either a list of numbers or a list of strings.

Function Argument	Function Return
a list of numbers or a list of strings	maximum element
[1,2,3,4]	4
['abc','b','c','d']	'd'

```
# Recursive function to find the maximum element in a list
def max_element(input_list):
    # function body
```

Note: The program internally check for a recursive implementation of the function. The student need not accept any input or print output to the console. The student should only write the body of the function.

Test Cases

Public

Input	Output
1, 2, 3, 4	4
'abc', 'b', 'c', 'd'	'd'

Private

Input	Output
4, 3, 2, 1, 2, 4	4
'd','c','b','a'	'd'
'ab c', 'ab d', 'a', 'd'	'd'
-2, -1, -1, 0, 0, 0	0

Answer

```python
# Recursive function to find the maximum element in a list
# it recursively applies the function till the list has only one element left and returns the max value
# it takes two arguments and uses max() to find maximum between two values,
def max_element(input_list):
    # Base case - input is a list of single element, return the element
    if len(input_list) == 1 :
        return input_list[0]

    # Recursive case - applies max builtin function of two values
    # The first part contains first element of the input list
    # The second part calling same function on a sub list excluding the first element
    else:
        return max(input_list[0], max_element(input_list[1:len(input_list)]))
```

Problem 3

Question

Part A: `simple_sort` is a non recursive function that take a list of numerical values and returns a sorted list. The returned list is in ascending order. The argument list can contain duplicate values. If the two values are same, they appear together (one after another) in the sorted list.

See the below table for `simple_sort`.

Function Argument	Function Return
a list of numeric values	a list of numeric values in ascending order
[1, 2, 3, 5, 8, 9]	[1, 2, 3, 5, 8, 9]
[8, 7, 5, 1, 2, 2, 0]	[0, 1, 2, 2, 5, 7, 8]

Part B: A function `simple_search` is a **recursive** function. It takes two inputs :

1. a sorted list from `simple_sort`
2. an item (a numerical value)

It searches for an item in the sorted list, lets say `item_list` in following ways:

- Step 1:
    - If the list has more than one element: check the middle most element in the sorted list, if this is the item being searched, item is said to be found and function returns bool literal `True`.
    - If the list is having only one element and it does not match with item being searched, the function returns bool literal `False`.
- Step 2:
    - If the item is not found in step-1, the function checks the item in a sublist of list used in step-1
        - If middle element < item: the item is searched in the sub list to the right of the middle item
        - If middle element > item: the item is searched in the sub list to the left of the middle item

Step 1 and Step 2 is repeated until the element is found or the list is exhausted.

See the below table for `simple_search`.

Function Argument	Function Return
a list of numeric values a numeric value	a boolean literal
[1, 2, 3, 5, 8, 9] 3	True
[0, 1, 2, 2, 5, 7, 8] 6	False

```
# Part A: sorts numbers in ascending order (or non descending order if duplicate elements are present)
def simple_sort(item_list):
    # function body

# Part B: Recursive function to find an element is present in a list or not
def simple_search(item_list, item):
    # function body
```

Note: The program internally check for a recursive implementation of a given function. The student need not accept any input or print output to the console. The student should only write the body of the function.

Test Cases

Public

Input	Output
1, 2, 3, 5, 8, 9 3	True
8, 7, 5, 1, 2, 2, 0 6	False

Private

Input	Output
1, 1, 2, 3, 5, 8, 9 1	True
8, 4, 7, 7, 5, 1, 2, 0 6	False
1.1, 2.2, 0.0, 0.0, 9.9, 8.8, 9.0 0	True
1.1, 2.2, 0.0, 0.0, 9.9, 8.8, 9.0 1.0	False

Answer

```
# a simple sorting function based on the minimum value
# Part A: sorts in ascending order (or non descending order if duplicate elements are present)
def simple_sort(item_list):
    sorted_list = []
    for item in item_list[:]:
        # find the minimum value in the item list and append it to sorted_list
        sorted_list.append(min(item_list))
        # remove the minimum value in the item list
        item_list.remove(min(item_list))
```

```
    return sorted_list

# Part B: Recursive function to find if an element is present in a list or not
def simple_search(item_list, item):
    # start position in the item list
    start = 0
    # end position in the item list
    end = len(item_list)
#     print(f"start={start}, end={end}")
#     print(item_list)
    # verify the item in sublist till start index is lesser than or equal to end index
    if start < end:
        mid = (start + end) // 2
        # if the middle element is the element being searched, update found as True
        if item_list[mid] == item:
            return True
        # if the item is smaller than mid value, left sublist is searched
        elif item_list[mid] > item:
            end = mid
            # recursive call to simple search function
            return simple_search(item_list[start:end], item)
        # if the item is smaller than mid value, right sublist is searched
        elif item_list[mid] < item:
            start = mid + 1
            # recursive call to simple search function
            return simple_search(item_list[start:end], item)
    else:
        return False
```

Non Recursive Solution

```
# a simple sorting function based on minimum value
# Part A: sorts in ascending order (or non descending order if duplicate elements are present)
def simple_sort(item_list):
    sorted_list = []
    for item in item_list[:]:
        # find the minimum value in the item list and append it to sorted_list
        sorted_list.append(min(item_list))
        # remove the minimum value in the item list
        item_list.remove(min(item_list))
    return sorted_list

# Part B: Non Recursive binary search function to find if an element is present in a list or not
def simple_search(item_list, item):
    # start position in the item list
    start = 0
    # end position in the item list
    end = len(item_list)-1
    # a flag to mark if the element is found or not
    found = False
```

```
    # verify the item in sublist till the start index is lesser than or
equal to the end index
    while start <= end and (not found):
        mid = (start + end) // 2
        # if the middle element is the element being searched, update found
as True
        if item_list[mid] == item :
            found = True
        else:
            # if the item is smaller than mid value, left sublist is
searched
            if item < item_list[mid]:
                end = mid - 1
            # if the item is smaller than mid value, right sublist is
searched
            else:
                start = mid + 1
    return found
```

Problem 4

Question

You are given a movie dataset in the form of a dictionary. The key of dictionary is the movie name and the value is a list. First element of this list is the box office collection in millions and the second element is the year of release.

```
movies_db =
{
    "Avatar":[2_847, 2009],
    "Avengers: Endgame": [2_797, 2019],
    "Titanic": [2_187, 1997],
    "Star Wars: The Force Awakens": [2_068, 2015],
    "Avengers: Infinity War": [2_048, 2018],
    "Jurassic World": [1_671, 2015],
    "The Lion King": [1_656, 2019],
    "The Avengers": [1_518, 2012],
    "Furious 7": [1_516, 2015],
    "Frozen II": [1_450, 2019]
}
```

You need to implement the functions given below in the code area. These functions are called in the suffix code (the shaded region) in the code area.

- `add_movie_to_boxoffice(movies_db, new_movie)`:
  - adds a movie `new_movie` to the movie database `movies_db`.
  - It returns the updated `movies_db`.
  - The `new_movie` argument is a tuple containing `name, collection, year` of a movie
    - for example, `new_movie = ('Star Wars: The Last Jedi', 1_332, 2017)`
- `total_collection(movies_db)`: returns total collection by all movies in `movies_db`
- `average_collection(movies_db)`: returns average collection by all movies in `movies_db` rounded up to 2 decimal points
- `num_of_movies_above_average_movies(movies_db)`: returns the number of movies with box office collection more than the average box office collection in `movies_db`
- `highest_grossing_movie_year(movies_db)`: returns the year of the movie with highest box office collection in `movies_db`

Internally, the program read a test case number and configures the `movies_db` with some movie data.

The expected output from the program displays:

> - Line-1: total collection,
> - Line-2: average collection,
> - Line-3: number of movies with above average box office collection,
> - Line-4: year of highest grossing film

```
# add a new movie to movie database
def add_movie_to_boxoffice(movies_db, new_movie):
    # function body
```

```
4
5  # returns the year of highest grossing movie in movie database
6  def highest_grossing_movie_year(movies_db):
7      # function body
8
9  # returns total collection of all movies in movie database
10 def total_collection(movies_db):
11     # function body
12
13 # returns average collection of all movies in movie database
14 def average_collection(movies_db):
15     # function body
16
17 # returns number of movies in movie database which has box office collection
   more than average
18 def num_of_movies_above_average_movies(movies_db):
19     # function body
```

Note: The student need not accept any input or print output to the console. The student should only write the body of the function.

Suffix Code Block

```
1  # Suffix Code Block - start
2  movies_db = add_movie_to_boxoffice(movies_db, new_movie)
3  print(total_collection(movies_db))
4  print(average_collection(movies_db))
5  print(num_of_movies_above_average_movies(movies_db))
6  print(highest_grossing_movie_year(movies_db))
7  # Suffix Code Block - end
```

Test Cases

Public

Input	Output
1	20917 1901.55 5 2009

Private

Input	Output
2	1714 131.85 4 2016
3	752 75.2 3 2017

Answer

```
# add a new movie to movie database
def add_movie_to_boxoffice(movies_db, new_movie):
    name, collection, year = new_movie
    # creating a movie entry in the movies database dictionary
    movies_db[name] = [collection, year]
    return movies_db

# returns the year of highest grossing movie in movie database
def highest_grossing_movie_year(movies_db):
    # finding highest collection and movie year with highest collection
    movies_data_list = list(movies_db.values())
    top_collection = 0 # movies_data_list[0][0]
    top_movie_year = 0 # movies_data_list[0][1]
    # looping through each movie in the movies_db
    for movie in movies_db:
        collection = movies_db.get(movie)[0]
        year = movies_db.get(movie)[1]
        # keep updating top_collection and top_movie_year as to store movie
of highest collection
        if collection > top_collection:
            top_collection = collection
            top_movie_year = year
    return top_movie_year

# returns total collection of all movies in movie database
def total_collection(movies_db):
    total_budget = 0
    # loop through all the movies and add the earning to total_budget
    for movie in movies_db:
        total_budget = total_budget + movies_db[movie][0]
    return total_budget

# returns average collection of all movies in movie database
def average_collection(movies_db):
    total_collection = 0
    # loop through all the movies and add the earning to total_budget
    for movie in movies_db:
        total_collection = total_collection + movies_db[movie][0]
    # divide total_budget by number of movies to get the average budget
    average_collection = total_collection / len(movies_db)
    return round(average_collection, 2)

```

```python
# returns number of movies in movie database which has box office collection more than average
def num_of_movies_above_average_movies(movies_db):
    abov_avg_movie_count = 0
    # loop through all the movies and count movies with higher earning than average collection
    for movie in movies_db:
        collection = movies_db[movie][0]
        if collection > average_collection(movies_db):
            abov_avg_movie_count += 1
    return abov_avg_movie_count
```

Problem 5

Question

Below table shows various subjects and related topics noted by some student. A topic may have been mistakenly noted multiple times for a subject. For example "list" in python.

Subject	Stored in Variable	A list of topics (can vary for each test case)
python	subject_topics[0]	["list","functions","variables","booleans","list","exceptions","conditions","input", "loops"]
java	subject_topics[1]	["strings", "variables","input", "exceptions", "integers","booleans", "loops"]
pdsa	subject_topics[2]	["algorithm", "variables", "complexity", "tree", "stack", "queue"]
dbms	subject_topics[3]	["join","floats","integers", "constraints","aggregate", "select", "where"]

The variable `subject_topics` is a nested list and stores the subjects data as shown in the table. Each element of `subject_topics` is a list of some topics.

Write a function `trending` that accepts `subject_topics` as input and does the following:

- Collate the topics present across all the subjects and store them in a list named `common_topics_list`. If a particular topic appears in multiple subjects, then retain only one entry for it. In other words, this list should not contain any duplicates.
- Find out the number of occurrences of each topic in `common_topics_list` across all subjects,
    - if a topic appears in two subjects, its occurrence will taken as 2.
    - a topic with highest occurrence is called "top trending topic"
    - a topic with minimum occurrence is called "least trending topic"
- Return the count of top trending and count of least trending topics separated by comma.
    - for above example the expected return value should be: `1,15`
    - that is, there is 1 topic which is top trending and there are 15 topics which are least trending

Evaluation Format

- You do not have to accept input from the user or print output to the console.
- You only have to fill the body of the following function.

```
def trending(subject_topics):
    '''
        Arguments: subject_topics (nested list)
        Returns: count_top_trending, count_least_trending (integer, integer)
    '''
```

Test Cases

Public

Input	Output
1	1,15

Private

Input	Output
2	1,13
3	2,15

Answer

```python
# implementation of function `trending`
def trending(subject_topics):
    # find common topics, it converts topics of each subject into a set to remove duplicates
    common_topics = set(subject_topics[0])
    for topics in subject_topics:
        common_topics = common_topics.union(set(topics))
    # convert it to a list and store in common_topics_list
    common_topics_list = list(common_topics)
    #print(common_topics_list)

    # find occurance of each topic in all subjects
    common_topics_count = {}
    # loop through each topic in common topics list
    for topic in common_topics_list:
        # intialize each topic count to 0
        common_topics_count[topic] = 0
        # loop through the topics of each subject
        # and increment the topic count by 1 if it appears in the topics of the subject
        for topics in subject_topics:
            if topic in topics:
                common_topics_count[topic] += 1

    # find topics with maximum occurrences (most trending)
    # get the maximum number of occurance from common_topics_count dictionary
    top_count = max(common_topics_count.values())
    top_topics = []
    # loop through common_topics_count and find the topics with max occurance
    for key, value in common_topics_count.items():
        if value == top_count:
            top_topics.append(key)

    # find topics with minium occurances (least trending)
```

```
    # get the minimum number of occurance from common_topics_count
dictionary
    bottom_count = min(common_topics_count.values())
    bottom_topics = []
    # loop through common_topics_count and find the topics with min
occurance
    for key, value in common_topics_count.items():
        if value == bottom_count:
            bottom_topics.append(key)
    # print most trending and least trending topics count
    return len(top_topics), len(bottom_topics)
```