Final Lab Exam

CS211 – Data Structures & Algorithms
Usman Institute of Technology
Fall 2020

Task 1: WordSpell

English language is amalgamation of 26 standard characters. The characters are either in upper case or lower case. The words in english are made up from these characters and are categorized into Nouns, Pronouns, Verbs, Adverbs, Adjectives etc.

Given a set of words in a text file, write a function **WordSpell()** that takes a text file as input parameter and returns a list of words that start from a specific spelling.

Input data:

- A text file named as Words.txt containing more than 1000 words.

Output data:

- A python list that contains the words starting from specific spelling.

Constraints:

- The words should only be string values.
- If there are any special characters in between the words, your function should omit those special characters.
- Function should be able to process both upper- and lower-case letters

```
def WordSpell(file, spell):
    // your code goes here

Example:
spell = "pre"
print(WordSpell(file,spell))

#the function should return

['predefined', 'predestination', 'precaution']
```

Task 2: Jukebox

In this task we are going to make our own Music Jukebox. This jukebox consists of a customized song playlist. Your jukebox is multi-functional and provide a number of functionalities.

The most important functionality is storing songs in the list. The jukebox should allow us to adding and removing songs in our list. It also allows us to marking and unmarking songs as favorites. Interestingly, the box automatically adds a song into favorite if it is played more than thrice. Obliviously, there should be a functionality to play any specific song or playing the entire list or only favorite songs. It also returns the list of songs in the sorted format.

This functionality required for a huge list of songs. The functionality of your implementation would be tested against a huge list of songs. Therefore, developers decided to implement using a Linked List. In this case, a linked list will be list of songs. We will be maintaining two linked lists: one for playlist and another for favorites songs. We will be having a class to store information about a song such as song name and singer. Another class will be required to providing mechanism for adding and removing songs in playlist and favorites.

Implementation:

You are required to implement:

1. Create a class **MySongs** that stores the information of song name, artist's name and the number of times the song played.
2. Create a class **JukeNode** to create a node for the linked list.
3. Create a class **MyJukeBox** and write the following functions in the class.
    a. Write a function **InsertSong()** that inserts a song in the list. The insertion should be done in O(1).
    b. Write a function **PlaySong()** that plays a specific song from the list. This function will print the -playing <song name>- on the screen (replace <song name> with the title of the song. We should also count the number of a times a song is played.
    c. Write a function **DeleteSong()** that deletes a song from the list. Make sure if a song is deleted from the list then it should also be deleted from favorites. The function should print " '<song name>' has been deleted" on the screen after deletion (replace <song name> with title of the song).
    d. Write a function **AddtoFav()** that adds a song to favorite. If a song is played more than 3 times then it automatically adds into favorite list. The function should print " '<song name>' has been added to favorites" on the screen after deletion (replace <song name> with title of the song).
    e. Write a function **PlayList()** that plays all songs of the list one by one such that the song which added first must play first.
    f. Write a function **PlayFavt()** that plays all songs that are added in favorites such that the song which marked favorite first must play first.
    g. Write a function **GetFav()** that returns a python list containing objects of class MySongs for only those songs which are added in the favorite.

h. Write a function **SortSongs()** that returns a python list containing object of class MySongs sorted alphabetically w.r.t artist's names.

Sample Output:

```
See the attached template (.py file) provided with the exam for the
exact name and signature of each function.

Example:
Obj = MyJukebox()
Obj.InsertSong("Radioactive","Imagine Dragons")
Obj.InsertSong("Girls Like you", "Maroon5")
Obj.InsertSong("Demons","Imagine Dragons")
Obj.PlaySong("Demons")
Obj.PlaySong("Girls Like you")
Obj.PlaySong("Girls Like you")
Obj.PlaySong("Girls Like you")
Obj.AddtoFav()
songlist = Obj.SortSongs()
Obj.DeleteSong("Radioactive")

The results should be:

-playing Demons-
-playing Girls Like you-
-playing Girls Like you-
-playing Girls Like you-

'Girls Like you' has been added to favorites
Radioactive has been deleted from the list.
```

Task 3: Queuing System

We have to implement a queuing system for a Bank of Digital Currency. The bank is first of its kind and people are taking a lot of interest in the bank. The bank decided to open different branches across the country to facilitated their customers. The management has to decide the number of counters and their types for each branch. Instead of deciding randomly, they decided to take advantage of simulations for this purpose. They have generated the list of customers with their type and time taken by each customer. You have to build a queueing system to simulate in order to calculate different values.

For this task, you have to build simulation only for four counters. The bank can have four kind of customers as given below. Our system will designate a separate queue for each type of customer.

- Queue 1: People want to open a new account
- Queue 2: People want to deposit cash into the account
- Queue 3: People want to inquire about the account information
- Queue 4: People want to close their account.

We will be providing a list of customers with their type and time taken by the respective customer. Since, the space in the bank is limited so the security guard of the bank is allowing only 10 people per queue. The rest of the people have to wait outside of the bank. So, each queue will have a length 10 customers. If a customer arrives and the respective queue is full then we consider that customer will not wait and go away. The bank operates from 9:00 am to 5:00 pm with lunch break of one hour. So, each counter/queue can server 420 minutes. We assume that customer arrives as the we start the day.

Our simulation should provide the following analysis:

- The number of customers deal for each type of counter/queue
- Average time taken by customers for each type of counter/queue
- Maximum length of a queue during any time of the day
- Number of customers we could not serve as queue was full (turned away)

Implementation:

1. Create a class **BankQueue** with the following functions:
    a. **Constructor**: takes two parameters typeId and size. The typeId describe an identifier for the queue and size is the size of the queue i.e. number of customers can be stored by the queue.
    b. **EnqueueCustomer**: takes a number as parameter which shows the time/minutes required
    c. **DequeueCustomer**: returns a customer from the queue as per FIFO
    d. **IsFull:** returns true if queue has number of customers equal to size of the queue, otherwise returns false.
    e. **IsEmpty**: returns true if queue is empty, otherwise returns false.
2. Create a class **BankSimulation** with the following functions:
    a. **Constructor:** takes a parameter filename: a filename with path to load customer data (description of the file format is given below)

b. **Process:** Process all customers one by one as per the requirements mentioned above.
c. **CustomersServed**: returns a python list of size 4 where each element shows the total number of customers served for each type of queue. First entry shows for queue type 1 and so on.
d. **AverageTime**: returns a python list of size 4 where each element shows the taken by customers for each type of queue. First entry shows for queue type 1 and so on.
e. **MaxLength:** returns a python list of size 4 where each element shows the maximum length of each type of queue. First entry shows for queue type 1 and so on.
f. **NotServerd:** returns a python list of size 4 where each element shows the number of customers not served for type of queue. First entry shows for queue type 1 and so on.

File Format

Each line of the file contains two integers X and Y where X is type of customers and Y is the time in minutes taken by the respective customer

$1 \leq X \leq 4$

$1 \leq Y \leq 60$

Sample File:

```
1 5
2 1
4 4
1 2
3 5
3 1
1 3
4 8
4 1
2 9
1 4
```

Sample Output:

```
Example:
Obj = BankSimulation("myfile.txt")
Obj.Process()
print(Obj.CustomersServed())
print(Obj.AverageTime())
print(Obj.MaxLength())
print(Obj.NotServerd())
[4, 2, 2, 3]
[3.5, 5, 3, 4.66]
[4, 2, 2, 3]
[0, 0, 0, 0]
```

Task 4: Emirates

A data set of emperors along with their years they died have been given in a text file attached with this document. Write a function **SortEmpire()** that takes list of objects of empire. The object consists of empire name and the year. The function should return a python list of containing objects in chronological order. The complete task should be executed in O(n).

File Format

Each line of the file contains two values N and Y separated by comma. N is the name of the emperors and Y is the year they died.

Sample File:

```
Tsiimba Mukuumbi, 1894
Cyrus the Great, 530
Bardiya, 522
Xerxes I, 486
Bivula Bangi, 1929
```

```
def SortEmpire(file):
    // your code goes here

Example:
print(SortEmpire("emperors.txt"))

#the function should return
Xerxes I, 486
Bardiya, 522
Cyrus the Great, 530
Tsiimba Mukuumbi, 1894
Bivula Bangi, 1929
```