Assignment 2 (Deadline Extended to Sunday, 10th Nov 2024, 11:59:59 pm)

- Rules are the same as Assignment 1, e.g. to be done individually (discussing the problems with each other is fine), code to be uploaded, marks, bonus marks policy, etc.
- You can use all concepts of Python discussed in class.
- Code for each question should be put in one file named <RollNo>_Q1.py, <RollNo>_Q2.py, …. Your code should have a function test() (or _test()) in which you will have all the test cases you write for the functions in your program. You will zip the files for all the questions and submit the **<RollNo>.zip** file on GC.
- ***Only some problems are given now - so you can get started. More problems will be added (Questions 8 to 10 added on 1-Nov).***

==

Question1. (Easy, ˜40LOC)

The canteen in the Institute maintains a table of prices of items, like:

1. Samosa: 15
2. Idli: 30
3. Maggie: 50
4. Dosa, 70

   …

For the program you have to write, set the menu in your program by this statement (feel free to add more items).

menu = [("Samosa", 15), ("Idli", 30), ("Maggie", 50), ("Dosa", 70), ("Tea", 10), ("Coffee", 20), ("Sandwich", 35), ("ColdDrink", 25)]

Maggie, 1, Rs 50
Samosa, 5, Rs 75

TOTAL, 6 items, Rs 125

Question2. (Medium. ~50 LOC.)
Write a program that will repeatedly take (for a student) input of this type: course_no, number_of_credits, and grade_received (eg. CSE101 4 A). These inputs are for the courses the student has done in the semester. If no input is given (i.e. just a return), that means no courses are left. From this data, print the transcript for the semester for the student as:

Course_no: grade
Course_no: grade
….

SGPA: n.nn (two decimal places)

In the transcript, the records should be printed such that they are sorted by the course_no (hint: when sorting a list of tuples/lists, the default sorting is with the first element of the tuple/list).

The course numbers are an alphanumeric string with capital letters in the start and digits at the end (e.g. CSE101, CSSS21), credits can be 1, 2 or 4, and grade can be A+ (10), A(10), A-(9), B(8), B-(7), C(6), C-(5), D(4), F(2) (the number in () is their numeric equivalent for SGPA calculation.) The SGPA is computed as (sum of: credits*grade/ total_credits). If any input is not valid, print a suitable message ("improper course no", "incorrect credit", or "incorrect grade") and ignore that input.

Suggestions: A list of tuples/lists may be a good structure to use. Read the input as tuples into a list. When inputs are done, process the list.
First just write code to read the input, split it into a list, and pass it to a function to check for errors (recall string has operations like isdigit() to check if the string is integer). If input is valid, add to a list of tuples (initially, to check the course name structure just check if it is an alphanumeric string - later write a function to check its valid structure - requires a bit of logic). Independently write a function which, given a list of tuples, can compute the SGPA. Then combine the two.

Question3: (Hard, 70 LOC)

Before you graduate, you get the signature from all your friends in your yearbook - which will be stored as a dictionary for the program. In this dictionary, the keys are the name of other students in the batch; the value is either 0 (for not signed) or 1 (for signed). Like you, all other students are also doing the same - and there is a dictionary entry for each. For creating this dictionary, input is given in a file - if there are N students, then the file contains:

name1:
name2, 1
name3, 0
name4, 1
…
name2:
name1, 0
name3, 0
name4, 0

…

(Like this data is there for all the graduating students).

Write a program to determine who has the most signatures and who has the least (if there are more than one for max/min - print all).

Suggestion. Initially manually create a dictionary yearbook = {name1: {...}, name2: {....}, …}, and then determine students with most/least signatures. Then write a function to read the file and create this dictionary.

Question4. (Medium, ˜50 LOC)

The goal of this game is for the user (player) to guess the 5 character word in a limited number of chances.

Pick up at least 50 5 character words (e.g. from https://7esl.com/5-letter-words/) and save them in a list or any other structure (you can assign them in a statement, i.e. hard code them). Select a word at random from this list - for this, generate a random number between 0 and length of the list of words using the random number generator randint() provided in python (Pls look it up - it is quite easy) - this is the index in the list for the word the user has to guess.

Now prompt the user to input a five character string as his/her guess. After the guess, the program should show the user the chars from the guess string which are in correct places, and correct chars in wrong places, by outputting:

--a-d (if a and d were in the input string and are in these two places in the word),
other characters present: b (if b is present in the word)

Let the user repeatedly give guesses, till either the guess is correct, or the number of tries is 6.

Bonus: Accept from the user only valid words as guesses. For this, use an available online dictionary API to check if the given string is a word or not or create a file of valid words and check in it. And if not, prompt the user and do not count this attempt.

Question5. Medium, 40 LOC.
You are given a list of coordinates (each as a x,y tuple) representing a 2D shape. If the shape has N coordinate, create a Nx3 matrix with the first column having the x values, 2nd column having the y values, and the third column being 1 for all. This represents the matrix for the shape.

You have to scale this 2D shape. For scaling, take as input cx and cy (scaling parameters) from the user, and form a matrix of the type:

$$\begin{bmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

For scaling the shape, multiply the matrix of the shape with this matrix. Finally, return the new shape in the form of a list of coordinates - which will be the first two columns of the resultant matrix.

Suggestion. While you can easily find matrix multiplication on the net, try to first write your own function for this - use online help only if needed.

Question 6. (Hard, 100 LOC)

Write a program to create your own personal address book, in which each entry has: Name, address, phone number, and email address. In the program, the address book should be a dictionary - the key of the dictionary has to be the **name** of the person, while the value has to be another dictionary with **address**, **phone**, and **email** as its keys. Initially, you can assume that people in your directory have unique names. Then modify the program to handle multiple persons with same name.(One way to handle this is to keep name as the key, but the value may be a list of dictionaries - one for each person)

Your program should provide the following operations: (i) insert a new entry, (ii) delete an entry (iii) find all matching entries given a partial name, (iv) find the entry with a phone number or email, and (v) exit. When the program exits, it writes the current dictionary in a file (addrbook.txt) and when it is started again next time, it reads the existing address book.
AS
Suggestion: To create an address book with some entries, write a small script which will call the add function to add a few entries to create a dictionary.

Bonus. Write a program to merge address book of your friend with yours. For this, you can store the address book as a json or as a list of dictionary items - if both you and your friend agree to the structure, merging will be easier.

Question 7. (Hard, ˜70.)
Given a text file (pages.txt) which has lines of the form:

URLnn, init_importance: text containing some URLnn.

Each line represents a page with the first URLnn being its URL and init_importance is a number between 0 and 1.0, which gives the initial importance of this page. The URLnn in the rest of the line refers to the URL links to other pages that this page contains. (To simplify, instead of giving a full path, we are using URLnn, and instead of having separate files for each page, we are giving the text of a page as a line in the file). Example:

URL00, 0.5: This is page zero, and has references to URL01, URL09, and also to URL08. It may have repeated references - so there are two references to URL09.

URL02, 0.6: This is another page (page is represented as a line in this). This has reference to URL05, URL04, and URL00
…

Pages are ranked according to their overall importance. Let the total number of unique links in a page i be links[i] (i.e. these many unique pages this page refers to). The overall importance of a page i is sum over all the pages (j) which have a link to page i of: init_importance[j]/ links[j] (i.e. all the pages to which the page j has a link are distributed the importance of this page equally)

Your program has to find the highest ranking N pages (N can be set in a variable or taken as input) for a given input file.

It seems natural to create a dictionary with page URL as the index, and having its init_importance, overall importance, the set of URLs it accesses, etc in it.

Note. This simplified version has only one round of computation for the overall importance from the initi_importance. The ranking algorithm actually takes the current importance (starting with init_importance, which is often set to 1) and then updates the importance repeatedly till the changes are very small. If you want, you can extend your program to implement this.

Question 7a. Bonus problem.
You can work in groups of 2 (both students can submit the same code - at the top of the code have a comment stating names of all students who worked together.)

Write an interesting application using some common Python libraries (e.g plotting library, GUI library, requests). The application should be at least 75 LOC.

Question 8 (Medium, ~60-70 LOC)

You are organizing a meeting with Alice, Bob, and Cameron. Each person has a schedule for the day with multiple booked meeting slots. The goal is to find a common 30-minute free time slot where all three can attend a meeting.

- The available meeting hours are from 9:00 AM to 5:00 PM (17:00).
- Each person's meeting schedule is given in a format like this: `["15:00-16:00", “9:10-10:10",12:00-13:00"]`.
- If a common 30-minute slot is available, print the slot and the names of all participants.
- If there is no common slot, but some conflicts (i.e., one or two people are free, while the others are not), print the conflict with the names of those who cannot attend the free time slot.
- If no common slot exists, return that there is no available time for all participants.

Input:

- Three lists (one for each participant) containing their meeting schedules for the day, in 24-hour format as strings.

Output:

- The available 30-minute time slot with all attendees.
- If there is a conflict, print the conflicting attendees and the available time slot.

Example Input:

```
alice_schedule = ["15:00-16:00", "12:00-13:15"]
bob_schedule = ["14:00-14:30", "12:30-13:30"]
cameron_schedule = ["09:00-10:00", "15:30-16:30"]
```

Bonus part [50 LOC]

Conflicts occur only when two or more candidates **share the same free slot** and no **uncommon (unique)** free slots are available for those candidates. The goal is to:

1. Find three unique, non-overlapping free slots for Alice, Bob, and Cameron.
2. If two candidates share only one common free slot and no other unique slots are available for them, print a conflict message indicating the candidates who have the conflict and the shared time slot.
3. If conflicts are not there for all, print the free slots for each person or print sorry this person today is not free.

Question 9. (Medium, ~80 LOC)

You need to create a Python-based Directory System where words are stored hierarchically according to their letters. The system will support basic operations like adding, deleting, searching etc. based on their structure. Here's a breakdown of the requirements:

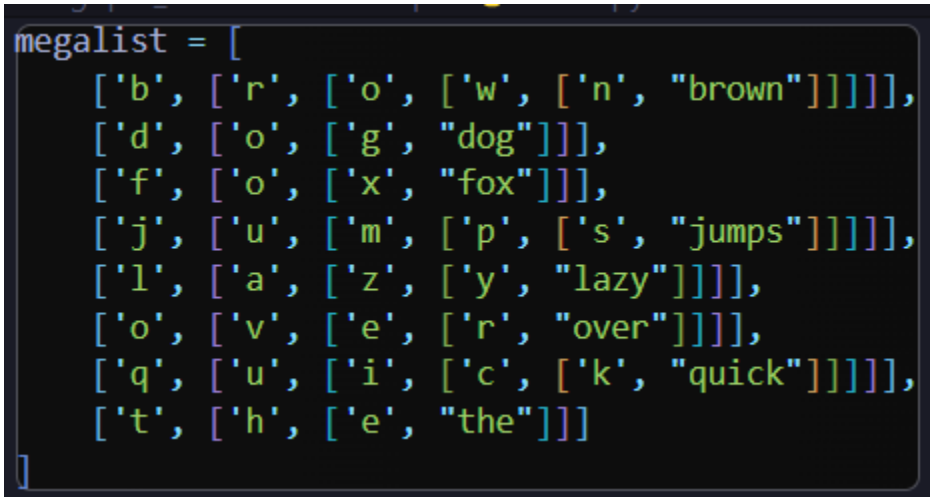
Directory Structure:

- **Mega List:** A list of sublists, each corresponding to a letter of the alphabet ('a' to 'z'). Words are represented as nested lists, where each sublist stores subsequent letters. For example, the word "hello" is stored in the sublist corresponding to 'h', and under it, there will be further nested sublists for 'e', 'l', etc.

  Sample mega list for words app, apple, bat, game, gamete:

  ```
  megalist =
  [['a',['p',['p',"app"['l',['e',"apple"]]]]],['b',['a',['t',"bat
  "]]],['g',['a',['m',['e',"game",['t',['e',"gamete"]]]]]]]
  ```

Sample mega list for words The, quick, brown, fox, jumps, over, the, lazy, dog:



```
megalist = [
    ['b', ['r', ['o', ['w', ['n', "brown"]]]]],
    ['d', ['o', ['g', "dog"]]],
    ['f', ['o', ['x', "fox"]]],
    ['j', ['u', ['m', ['p', ['s', "jumps"]]]]],
    ['l', ['a', ['z', ['y', "lazy"]]]],
    ['o', ['v', ['e', ['r', "over"]]]],
    ['q', ['u', ['i', ['c', ['k', "quick"]]]]],
    ['t', ['h', ['e', "the"]]]
]
```

Functionalities:

1. **Create Directory:**
    - Parse a given list of space-separated words and store each word hierarchically in the mega list based on its letters.
2. **Interactive Input System:**
    - Allow users to input letters interactively and display all words in the directory starting with the typed prefix.
3. **Count Words Containing a Substring:**
    - Traverse the directory to count all words containing a specified substring.
4. **Add Word:**
    - Insert a new word into the directory in its correct position based on its letters.
5. **Delete Word:**
    - Locate a word in the directory and remove it from the structure.

Question10 (Answers to this question are to be added in a file named <***RollNo_Q10.pdf***> in the zip file that you submit)**.**

(i) Please go through the following functions:

```
def update_listA(lst):
    for i in lst:
        if i% 2 == 0:
            lst.insert(lst.index(i),2*i)
            print(lst)
        else:
            i = i + 1
```

```
            print(lst)
    return lst

def update_listB(lst):
    for i in range(len(lst)):
        if lst[i] % 2 == 0:
            lst.insert(i,2*lst[i])
            print(lst)
        else:
            lst[i] = lst[i] + 1
            print(lst)
    return lst
```

Will they both give the same answer to the question below? Elaborate on your answer. Also, explain what you think the code does.

```
Example usage
my_list = [1, 2, 3, 4]

#call to update_listA or update_listB, as applicable
updated_list = update_list(my_list)
print(updated_list)
```

(ii) Consider the following initializations:

```
tup = ()
dic = {}
lis = []
tup1 = (tup,dic,lis,5)
```

What will print(tup1) will give for each of the below snippets separately

a. `tup1.append(5)`
b. `tup1[1][1] = []`
   `tup1[1][1].append(5)`
c. `tup1[0].append(5)`
d. `tup1[2].append(5)`
e. `tup1 = tup`
f. `tup1[3] = tup1[3]`

(iii) If `lst = [10,20,30,40].` Do any two of the operations output the same value? If yes, what is your explanation ?

```
a. lst[1:3] = 5*[0]
b. lst[-1:3] = 5*[0]
c. lst[1:-3] = 5*[0]
d. lst[-1:-3] = 5*[0]
```

(iv) Consider the following function written to rotate the list lst by k values. Identify any errors or useless pieces of code that are in the above snippet. Explain.

```
def rotate_list(lst, k):
    if not lst:
        return lst
    if k not in lst:
        return lst
    k = k % (len(lst) -1)
    return lst[k:] + lst[:k]

# Example usage
my_list = [1, 2, 3, 4, 5]
k = 12
rotated_list = rotate_list(my_list, k)
print(rotated_list)
```

(v) Consider the following function:

```
def fxn(data, k):
    if k <= 0:
        return data

    if isinstance(data, list):
        return [fxn(data, k - 1)]
    elif isinstance(data, tuple):
        return (fxn(data, k - 1))
```

Explain what the above code does for lists and tuples. Write outputs for

a. fxn([ ], 5)
b. fxn(( ), 6)

Will the answer to a and b remain same even if I change the second parameter

Is the first **if** condition checking k <= 0 redundant?

(vi) Predict the output of the following code snippet.

```
dict_a = {1:'a',2:'b'}
```

```
dict_b = {2:'c',3:'b'}
dict_b[1] = (dict_a[2] != 'dict_b[2]')
dict_b[2] = (dict_a[2] == dict_b[2])
dict_a.update(dict_b)

print(dict_a,dict_b)
```

(vii) Consider the following function written to sort a list of numbers: The function has an error.

```
1.def sortList(a):
2.    n = len(a)
3.    if n <= 1:
4.         return
5.    for i in range(1, n):
6.         key = a[i]
7.         j = i-1
8.         while j >= 1 and key < a[j]:
9.              a[j+1] = a[j]
10.             j -= 1
11.        a[j+1] = key
12.   return a
```

a) Write an assert statement that will pass (Your input list should have at least 4 numbers)
b) Write an assert statement that will reveal the error (Your input list should have at least 4 numbers).
c) Give the correction to be made so that assert statement in b) will pass

(viii) Consider the following code written to convert algebraic expressions in infix format to postfix format using the precedence rules of operators:

Examples:

a) **Input =** “a+b”; **Output =** “ab+”
b) **Input =** “a*b+c”; **Output =** “ab*c+”
c) **Input =** “a*(b+c)”; **Output =** “abc+*”
d) **Input =** “a+b*c”; **Output =** “abc*+”
e) **Input =** “(a+b)*c”; **Output =** “ab+c*”
f) **Input =** “a*b^c”; **Output =** “abc^*”

```
# Function to return precedence of operators
def prec(c):

    if c == '^':
```

```
        return 3
    elif c == '/' or c == '*':
        return 2
    elif c == '+' or c == '-':
        return 1
    else:
        return -1

# Function to perform conversion of expressions
def expr_conv(s):
    st = []
    result = ""

    for i in range(len(s)):
        c = s[i]

        # If the scanned character is an operand, add it to the output string.
        if (c >= 'a' and c <= 'z') or (c >= 'A' and c <= 'Z') or (c >= '0' and c <= '9'):
            result += c

        # If the scanned character is an '(', push it to the list st used as a stack.
        elif c == '(':
            st.append('(')

        # If the scanned character is an ')', perform pop on st and add the popped character
       # to the output string from the stack until an '(' is encountered.
        elif c == ')':
            while st[-1] != '(':
                result += st.pop()
            st.pop()

        # If an operator is scanned
        else:
            while st and (prec(c) < prec(st[-1]) or prec(c) == prec(st[-1])):
                result += st.pop()
            st.append(c)

    # Pop all the remaining elements from the stack
    while st:
        result += st.pop()

    print(result)
    return(result)
```

a. Write assertion statements for each of the following. Assume that the function has no bug and the assertion statements will pass.

   I. A valid infix expression with at least 4 operators, of which 3 have three different precedence values

   II. A valid infix expression with at least 5 operators and two pairs of parentheses/brackets to override the precedence rules.

   III. A valid infix expression with each of the five operators (+, -, /, *, ^) occurring at least once and at least one pair of parentheses to override the precedence rules

b. What review comment will you give to the programmer, if you are asked to test the code with input expression strings such as “abc”, “abc+”, “+-*/” that are not valid infix expressions ? (You are not expected to correct the code)