CSE 101 - Introduction to Programming
Assignment 3
March 2022

GENERAL INSTRUCTIONS

1. In this assignment, you need to write the code for every problem in different '**.py**' files.
2. You are only allowed to use the basic modules available in Python. You cannot use any external modules, libraries, or APIs.
3. You will create a document, citing all the online and offline resources, that you have used, in completing this assignment.
4. Your code will be checked for plagiarism against the code of your classmates as well as some sample codes available online. The institution's policies will strictly apply.
5. This time, questions will be added in phases, i.e, further questions would be added. You will be notified regarding this.
6. Start early. Resolve all your doubts with the TAs, *2 days* before the deadline.

SUBMISSION INSTRUCTIONS

1. Your code files must be renamed as A3_2021xxx_{Problem_no}.py. For example, A3_2021001_1.py, would be the python file for problem 1 for roll no. 2021001.
2. Your document with references to resources used must be in pdf format and named A3_Resources_2021xxx.pdf
3. All these files (code files and the reference pdf file) must be put in a zipped folder named A3_2021xxx.zip
4. The zip file containing everything relevant to this assignment must be submitted before the deadline on google classroom.

PROBLEMS

Q1.

Remember the problem **Transformations!!** from Assignment 2, where we were performing transformations on an n-point 3D shape. This problem is an extension of that problem with certain modifications.

Firstly, if you didn't create different functions for different transformations, you need to change your code so that there are at least these 3 functions -

1. **scale(x, y, z, sx, sy, sz)** which returns scaled x', y' and z' according to sx, sy and sz.
2. **translate(x, y, z, tx, ty, tz)** which returns translated x', y' and z' according to tx, ty and tz.
3. **rotate(x, y, z, axis, phi)** which returns rotated x', y' and z' according to axis and phi.

You also have to create a function **matmul(A, B)** which returns the matrix multiplication of matrices A and B.

After these modifications are done, your task is to create a **A3_2021xxx_1_test.py** which will test the correctness of the code. In **Q1test.py** there will be a total of 4 functions -

1. **test_matmul(A, B, true_C)** which calculates the matrix multiplication of matrices A and B using the **matmul(A, B)** function and tests the returned matrix C with true_C. If they match, it returns True, and False otherwise.
2. **test_scale(x, y, z, sx, sy, sz, true_x, true_y, true_z)** which scales x, y, z in accordance with sx, sy and sz using the **scale(x, y, z, sx, sy, sz)** function and tests the returned x', y', z' with true_x, true_y, true_z respectively. If they match, it returns True, and False otherwise.
3. **test_translate(x, y, z, tx, ty, tz, true_x, true_y, true_z)** which translates x, y, z in accordance with tx, ty and tz using the **translate(x, y, z, tx, ty, tz)** function and tests the returned x', y', z' with true_x, true_y, true_z respectively. If they match, it returns True, and False otherwise.
4. **test_rotate(x, y, z, axis, phi, true_x, true_y, true_z)** which rotates x, y, z in accordance with axis and phi using the **rotate(x, y, z, axis, phi)** function and tests the returned x', y', z' with true_x, true_y, true_z respectively. If they match, it returns True, and False otherwise.

These matching are expected to be done **only** using assertions.

You will be making multiple test cases for all the functions and will test your code's correctness. This problem will help you in understanding how a judge on a competitive programming website works.

Q2.

Cleaning services are crucial to our lifestyle, aesthetics and overall hygiene. Your task is to design a **class** for a cloth and apparel cleaning service. The class would be named, **LaundryService.** This class would be used to create **Customer Specific** details, and would contain methods/functions for:

→ **__init__():** The parameters passed to this function, during the object instantiation would be:

Parameter	Appropriate Type/ Values
Name of customer	String
Contact of customer	Numeric [integer]
Email	Alphanumeric [string]
Type of cloth	"Cotton", "Silk", "Woolen" or "Polyester"
Branded	A boolean
Season	"Summer" or "Winter" [string]

Inside __init__(), you will be assigning an **id** to a new customer. This **id** would be global variable and **should be incremented automatically** when a new customer is created, and the incremented value must be assigned as the **id** of the new customer.

→ **customerDetails():** This will print out the details of a customer in the order:

1. Their name
2. Their contact no.
3. Their email
4. Type of cloth they deposited
5. Whether the cloth is branded or not

##No need to print out the season

→ **calculateCharge():** This will calculate the charge and return, based upon the following factors:

1. Type of cloth: Cotton → 50 Rupees, Silk → 30 Rs., Woolen → 90 Rs., Polyester → 20 Rs.
2. Branded: if True, **1.5** times the charge from step 1. Otherwise the charge is unchanged
3. Season: if it is "Winter" charge is halved from step 2. Otherwise the charge is **2 times** the charge from step 2.

→ **finalDetails():** This function will call customerDetails() and calculateCharge() functions within itself. Furthermore, based upon the charge returned by the calculateCharge(), this function will further print the total charge and the expected day of return.

1. If total charge is greater than 200, it will print out "To be returned in 4 days"
2. Otherwise, it will print out "To be returned in 7 days"

After creation of the class, you need to able to create multiple objects of type **LaundryService.**
The parameters to be passed for the initialization would be taken from the user. Finally you will be calling findDetails() method from the object created.
Caveat: for the input for **brand,** you can take **0** or **1** from the user. It can be passed **False** or **True**, therewise during instantiation.

```
Name of customer: Saira
Contact No.: 9876543210
Email: saira@hanjimail.com
Type of cloth: Woolen
Branded?: 1
Season: Winter

The customer specific details:

Customer Id: 1
Customer Name: Saira
COntact: 9876543210
Email: saira@hanjimail.com
Type of cloth: Woolen
Branded or not True
Total charge: 45.0
To be returned in 7 days!
```

Another instantiation of a new object in the same running code (student are free to iterate for a given number of objects or they can restrict the number of objects for atleast 4) would give the result as:

```
Name of customer: Elton John
Contact No.: 8787878787
Email: elton@ohhoo.com
Type of cloth: Cotton
Branded?: 0
Season: Summer

The customer specific details:

Customer Id: 2
Customer Name: Elton John
COntact: 8787878787
Email: elton@ohhoo.com
Type of cloth: Cotton
Branded or not True
Total charge: 100
To be returned in 7 days!
```

The output can deviate in formatting and sequence of statements. This one is for reference only.

Q3.

Bank of IIIT-D is a newly opened bank that caters to the needs of the institute. The faculty, students and all the staffs can open a bank account under their name. The main purpose of this bank is to maintain some emergency funds for the user in case the need arises. There is no system to provide rate of interests to the users, for now.

A blueprint design/prototype has to be formulated for the bank. This task has come under the responsibility and accountability of Dr. Pankaj Jalote and Dr. Raghava Mutharaju. As banking services need to be robust and trustworthy, the professors need someone who could be relied upon, to build the prototype. Can they rely upon you?

Your task is to create a menu driven program. First of all, the code starts with some welcome message, and then asks the users to provide **their username, password, and initial amount** to be deposited into the bank. Then a menu driven approach has now to be utilized for these 4 tasks:

1. Asking user to deposit a sum **into** their account
2. Asking user to withdraw a sum **from** their account
3. Asking the user if they want to see their bank statement, i.e, their past transaction history
4. If the user still wish to continue.

Here comes the detailed implementation that should work behind the scenes to facilitate the previously mentioned tasks.

You need to create a class called **BankAccount**. The class BankAccount has 5 methods inside it. These are as follows:

- **__init__()**
- **deposit()**
- **withdraw()**
- **authenticate()**

- **bankStatement()**

1→ __init__():
This method, the usual constructor, accepts 4 parameters during object instantiation. These are the username, password, current balance. Remember, that the current balance, just after the instantiation, is the initial amount provided by the user in the menu driven code. It also creates a file named **'u.txt',** where **u** is the username itself. This file contains the transaction history of a specific user. This will be a unique file created for every new user. You can assume that no two users have the same name.

2→ authenticate()
This method will ask the user to input " provide their secret password". It would return True or False depending upon whether the provided password matches with the password given, during the instantiation. This method is necessary as multiple authentications are required during a session. It would be called everytime a user wants to deposit, withdraw or view his bank statement.

3→ deposit()
This method will accept the amount to be deposited, as the parameter. It will be calling ***authenticate()*** to verify the user. You need to consider one caveat! There would be at most **2 wrong attempts** while entering a password. You will be using **assert,** to ensure the number of attempts does not exceed 3. If it does, an AssertionError would be raised with the message "**Too many wrong attempts!!**". This AssertionError must be handled using **exception handling.**
The current balance would be updated, by **adding** the amount **to** it.
The user specific file '**u.txt**' would be again opened, and a text would be **appended** to the file. The text would be of the form:
Timestamp The amount of Rupees 100.0 has been added Current balance → 600.0

For the timestamp you can use the datetime module directly, whose working is as follows:

```
>>> from datetime import datetime
>>> print(datetime.now())
2022-03-24 08:41:51.494869
```

Finally you will be closing the file to ensure no error or glitch persists.

4 → withdraw()
This method will accept the amount to be withdrawn or debited, as the parameter. Again it will also be calling ***authenticate()*** to verify the user. Similar to **deposit(),** number of attempts allowed would be at most 3, and you would be using **assertion** to ensure the same.

The current balance would be updated, by **subtracting** the amount **from** it. Again a caveat!! You need to first ensure that the amount to be withdrawn is less than the current balance, otherwise withdrawal would not be possible with a message "**Low balance!! Cannot be debited at this time!**" and then the function would simply return.

If the current balance is sufficient, then user specific '**u.txt**' file would be re-opened and a text would be appended to it, of the form:

Timestamp The amount of Rupees 100.0 has been debited Current balance → 600.0

Finally, the user specific file will be closed for the aforementioned rationale.

5 → bankStatement()
This method does not accept any parameter. It also calls ***authenticate()*** for verification and **assert** to ensure the number of attempts to 3, similar to the steps followed in the previous two methods.
Again the user specific file '**u.txt**' will be opened in **read** mode. It will print out the transaction history line by line.

For this question, students' can implement password validation and email address, as per their choice.

One sample run for the said implementation has been attached below:

```
Welcome to the Bank of IIIT-D

Enter Name: Shashwat
Enter Password: Vaibhav
Starting balance for your account: 500
Select Your Option:
1. Deposit
2. Withdraw
3. Bank Statement
Enter your choice: 1
Provide the amount to be deposited: 100
Provide your secret password: Y
Provide your secret password: Vaibhav
Do you wish to continue? Y/N : Y
Select Your Option:
1. Deposit
2. Withdraw
3. Bank Statement
Enter your choice: 2
Provide the amount to be withdrawn: 200
Provide your secret password: Vaibhav
Do you wish to continue? Y/N : Y
Select Your Option:
1. Deposit
2. Withdraw
3. Bank Statement
Enter your choice: 1
Provide the amount to be deposited: 300
Provide your secret password: Vaibhav
Do you wish to continue? Y/N : Y
Select Your Option:
1. Deposit
2. Withdraw
3. Bank Statement
Enter your choice: 2
Provide the amount to be withdrawn: 100
Provide your secret password: Vvvv
Provide your secret password: VavV
Provide your secret password: Vaibhav
Do you wish to continue? Y/N : Y
Select Your Option:
1. Deposit
2. Withdraw
3. Bank Statement
Enter your choice: 2
Provide the amount to be withdrawn: 1000
Provide your secret password: Vaibhav
Low Balance!! Cannot be debited at this time!
Do you wish to continue? Y/N : Y
Select Your Option:
1. Deposit
2. Withdraw
3. Bank Statement
Enter your choice: 3

Provide your secret password: Vaibhav

Hey Shashwat! Your transactions history:

2022-03-24 03:08:30.930703 The amount of Rupees 100.0 has been added  Current balance -> 600.0
2022-03-24 03:09:19.979754 The amount of Rupees 200.0 has been debited  Current balance -> 400.0
2022-03-24 03:09:46.060572 The amount of Rupees 300.0 has been added  Current balance -> 700.0
2022-03-24 03:10:27.644359 The amount of Rupees 100.0 has been debited  Current balance -> 600.0

Do you wish to continue? Y/N : N
>>>
```

Also the text file has been generated :

With the content:

Shashwat.txt - Notepad

File Edit Format View Help

```
2022-03-24 03:08:30.930703 The amount of Rupees 100.0 has been added  Current balance -> 600.0
2022-03-24 03:09:19.979754 The amount of Rupees 200.0 has been debited  Current balance -> 400.0
2022-03-24 03:09:46.060572 The amount of Rupees 300.0 has been added  Current balance -> 700.0
2022-03-24 03:10:27.644359 The amount of Rupees 100.0 has been debited  Current balance -> 600.0
```

Q4.

Given a list of people objects, create a function **sort_attribute()** that sorts the list by an attribute name. The attribute to sort by will be given as a string, one among “firstname”, “lastname”, “age”.

You need to create a class called **Person**. The class Person has **2** methods inside it. These are as follows:

- **__init__()**
  This method, the usual constructor, accepts 3 parameters during object instantiation
  - firstname (string)
  - lastname (string)
  - age (int)

- **object_info()**
  This method will print all 3 parameters of an instance of a class space separated, in the order firstname, lastname and age.

For k queries that supply a string among “firstname”, “lastname” or “age”, sort the information for ‘n’ users for that particular attribute and return the person ids in ascending order for that sort of attribute. Print the menu as shown below in sample testcase and use that everytime

Input format:

The first line takes input ‘n’ for n information lines and ‘k’ for k queries
Each of the ‘n’ lines takes 3 space separated inputs: Firstname, Lastname and Age (integer)
Each of the next ‘k’ lines takes string inputs that could be “firstname”, “lastname” or “age”

Output Format:

Each of the k lines consists of a line separated output of **object_info()** function of class Person. You can also implement **__repr__()** function in place of object_info().

Sample Menu (white), Input (italics) and Output (bold):

Welcome to the Application!

Specify number of people to be enrolled: *3*

Enter Firstname for Person 1: *Michael*
Enter Lastname for Person 1: *Smith*
Enter Age for Person 1: *40*

Enter Firstname for Person 2: *Alice*
Enter Lastname for Person 2: *Waters*
Enter Age for Person 2: *21*

Enter Firstname for Person 3: *Zoey*
Enter Lastname for Person 3: *Jones*
Enter Age for Person 3: *29*

Specify number of Queries: *3*
Query 1: *firstname*
Order:
Alice Waters 21
Michael Smith 40
Zoey Jones 29

Query 2: *lastname*
Order:
Zoey Jones 29
Michael Smith 40
Alice Waters 21

Query 3: *age*
Order:
Alice Waters 21
Zoey Jones 29
Michael Smith 40

Thanks for using the Application, if you wish to restart, press “Y” else press “N” to exit: *N*
Exits!

Q5.

Placement Season is about to start at IIITD. You have to design an application to simplify the procedure. You have to make two classes - Student and Company.

Functions for the Student Class are described Below:

1. **__init__** : This function takes 3 parameters: Name (String input), cgpa (Float input) and branch (String input) of the student. Possible branches include CSE, CSAM, ECE. You have to initialise another variable isPlaced to False.
2. **isEligible** : This function takes 1 parameter: Object of the Company. It checks if the student is eligible for the given company i.e.
   - the student has cgpa >= the required cgpa for the Company,
   - the company is hiring from the student's branch and
   - the student is unplaced.

   If the student is eligible for that company, then print "Student {name} is eligible for Company {name}". If the student is not eligible, then print "Student {name} is not eligible for Company {name}"
3. **apply** : This function takes 1 parameter: Object of the company. It will add the student to the list of students who have applied for that particular Company.
4. **getsPlaced** : This will change the status of the student from unplaced to placed.

Functions for the Company Class are described Below:

1. **__init__** : This function takes 4 parameters: Name (String input), requiredcgpa (float input which is the least cgpa required to become eligible), branches (list of branches from which

the company will hire) and positionOpen (which is the maximum number of students that the company will hire). You also have to initialise another variable appliedStudents as an empty list and a variable applicationOpen to True.

2. **hireStudents** : This function does not take any parameters and will hire top k students from the list of appliedStudents based on their cgpa (k represents the number of positions open). When the company hires a particular student, you have to change the status of that student to placed. Now print the names of the students that the company has hired and the number of positions open for the company. This function in the end will call the function closeApplication.

3. **closeApplication** : This function also does not take any parameter and this will close the application of the Company and print the number of students that the company has hired.

After defining these classes, take inputs of atleast 5 students and 2 companies. Now you have to iterate on all the students and check if he or she is eligible for the company. If the student is eligible, then call the apply function for that student. After iteration on all the students, call the function hireStudents for company 1. Now again iterate on all the students and check if they are eligible and then call the function hireStudents for company 2.

NOTE: Remember while taking the input for the **cgpa** of the student, you need to use exception handling to ensure that the cgpa lies between 0 and 10. If the cgpa lies outside these bounds, throw exception using **assertion statement** of your choice (ex → “The cgpa is invalid”, or “Kindly provide correct and valid cgpa”, etc).

Sample Run:

```
Enter the number of students
5
Enter name of Student 1
Naman
Enter cgpa of student 1
10
Enter branch of student 1
CSE
Enter name of Student 2
Pankil
Enter cgpa of student 2
9.6
Enter branch of student 2
CSE
Enter name of Student 3
Himanshu
Enter cgpa of student 3
5.9
Enter branch of student 3
CSE
Enter name of Student 4
Vishwesh
Enter cgpa of student 4
6.9
Enter branch of student 4
CSAM
Enter name of Student 5
Mayank
Enter cgpa of student 5
8.0
Enter branch of student 5
ECE
Enter the number of Companies
2
Enter name of Company 1
Microsoft
Enter required cgpa of Company 1
6.0
Enter space separated branches of Company 1
CSE CSAM
Enter number of positions Open of Company 1
2
Student Naman is eligible for the Company Microsoft
Student Pankil is eligible for the Company Microsoft
Student Himanshu is not eligible for the Company Microsoft
Student Vishwesh is eligible for the Company Microsoft
Student Mayank is not eligible for the Company Microsoft

The company Microsoft has hired the following Students :
Naman
Pankil
The company hired 2 students

Enter name of Company 2
Amazon
Enter required cgpa of Company 2
1.0
Enter space separated branches of Company 2
CSE CSAM ECE
Enter number of positions Open of Company 2
5
Student Naman is not eligible for the Company Amazon
Student Pankil is not eligible for the Company Amazon
Student Himanshu is eligible for the Company Amazon
Student Vishwesh is eligible for the Company Amazon
Student Mayank is eligible for the Company Amazon

The company Amazon has hired the following Students :
Mayank
Vishwesh
Himanshu
The company hired 3 students
```