Lab 13: Python

ESS 112 - Programming I

International Institute of Information Technology – Bangalore

Submission: LMS by 22 Feb 23:59:59

Submission Instructions: Submit your code in a separate file on LMS.

Exercises:

1. Write a python program to multiply each number of a list with a given number using lambda function.
   Sample Input:
   1 2 3 4
   2
   Sample Output:
   2 4 6 8

2. Write a program that takes a list of integers and returns a list of all the unique pairs of numbers whose product is even. Use list comprehension to implement your solution.

   Sample Input:
   1, 2, 3, 4, 5

   Sample Output:
   [(1, 2), (1, 4), (2,3), (2, 4), (2, 5), (3, 4), (4, 5)]

3. Write a program that takes a list of integers and returns a list of all the possible subarrays (contiguous subsequences) of length of two, sorted in descending order by the product of their elements. Use list comprehension to implement your solution.
   Sample Input:
   1, 2, 3, 4

   Sample Output:
   [(3, 4), (2,3), (1,2)]

4. Implement a Taxi class with subclasses MicroTaxi, MiniTaxi, SedanTaxi. Following properties should be provided: 1. rateCard 2. taxiAgencyDetails

   Place the above properties in the appropriate classes. Explain your decision in the form of code comments.

   Implement a class Ride with attributes: vehicleType (MicroTaxi, MiniTaxi or SedanTaxi), and travelDistance. and a property fare that is computed using the vehicleType and travelDistance attributes.

5. Implement a Person with attributes: name, parent, children.

   Implement a class Family (essentially a tree of Persons) with an attribute headOfFamily as its root node. Let there be the following methods: • headOfFamily • allNodes • searchNode(n), n being a node. • allAncestors(n), n being a node. • parent(n), n being a node. • depth Add more methods and properties if required. Use Person class specifying the parent and children of each person. Display the tree at the end.

6. Write a program that asks the user to input a list of numbers and then calculates the average of those numbers. Your program should handle the following exceptions:
   a. If the user enters a non-numeric value, your program should catch the ValueError exception and print an error message to the user.
   b. If the user enters a negative number, your program should catch the ValueError exception and print an error message to the user.
   c. If the user enters an empty list, your program should catch the ValueError exception and print an error message to the user.
   d. If the calculation fails for any other reason (such as division by zero), your program should catch the Exception exception and print a generic error message to the user.

   Your program should keep asking the user for input until a valid list of numbers is entered. Once a valid list is entered, your program should calculate the average and print the result.

7. (Bonus question – optional) Design python program to demonstrate a possible implementation of simplistic bank ATM. To check your balance, you enter your account number. The bank sends an OTP to your registered mobile, and if you enter the OTP correctly in the ATM, it prints out your current balance.
   Your solution should follow the class structure described below:
   1. ATM: contains the main and is responsible for taking the user input and sending it to the Bank, using the methods of Bank listed below
   2. Bank: has a list of Customers and Accounts. It uses an OTPService to send OTP and also to verify the OTP. Checking balance for an account is thus a 2 step process: the ATM sends a request to the Bank to send an OTP to the registered mobile number associated with the customer for this account, and then the ATM sends a second request to the Bank to verify the OTP. Bank provides the following methods:

   - addCustomer(mobileNumber) returns a unique customer ID
   - addAccount(customerID,balance) returns a unique account ID. Balance is the initial balance for this account
   - sendOTPforBalanceCheck(accountNumber)
   - verifyOTPandGetBalance(accountNumber, OTP) returns the balance if the OTP is correct for this account number

   3. Customer: Each customer has a unique ID as well as a registered mobile number
   4. Account: An account consists of the Customer ID, the Account ID and the balance. Customers can have multiple accounts
   5. OTPService: sends an OTP to a requested mobile number, and also verifies if an OTP matches what was sent out for a particular request. Provides the following methods:

   - sendOTP(mobileNumber) - sends OTP (prints OTP to standard out)

- verifyOTP(mobileNumber, OTP) - returns true if the OTP matches the outstanding OTP for this mobile number

For simplicity, all unique numbers/IDs mentioned above are integers: Customer ID, Account ID, OTP, are each managed independently, and each is a set of auto-generated sequential numbers. Customer IDs start from 1, Account IDs start from 101, OTP's start from 1001.

The ATM class can access only the Bank class and its methods. All other classes are hidden from it.

OTPService encapsulates the mechanism of sending out OTP's and verifying them for a given mobile number. One implementation is for this class to maintain the list of pairs so that it can later verify the OTP. Note that for security, the combination of <mobileNumber, OTP> should not be accessible by any other class.

The main reads the standard input and processes each line as follows, based on the 1st character of the input line:

- if the first character is C, then this is to add a new customer. The next field is an integer, the mobile number of the customer

- if the first character is A, then a new account is created. The next field the Customer ID of the customer owning this account, and the third field is the balance in the account

- if the first character is B, then a request to send OTP is to be sent to the bank. The next field is the account ID whose balance is being queried

- if the first character is V, then the OTP is to be verified. The next 2 fields are the account ID and the OTP to be verified

- if the first character is E, then that is the end of input

Sample Input

```
C 12345
C 23456
C 34567
C 45678
A 1 1000
A 2 2000
A 3 3000
A 2 4000
A 3 5000
A 3 6000
B 101
V 101 1001
B 103
B 102
```

```
V 103 1002
B 104
V 102 1003
V 104 1004
B 106
V 106 1002
V 106 1005
V 106 1005
E
```

Sample Output

```
OTP 1001 to 12345
Balance 1000
OTP 1002 to 34567
OTP 1003 to 23456
Balance 3000
OTP 1004 to 23456
Invalid OTP
Balance 4000
OTP 1005 to 34567
Invalid OTP
Balance 6000
Invalid OTP
```