Week-10, Graded Programming

Problem 1

Question

Write a class named `Point` having object attributes `x` and `y` taken as parameters respectively.

- `x` and `y` should be zero if no parameter is passed.
- Write a method for `Point` class named `move` that takes two arguments `dx` and `dy` as input and increments the `x` and `y` attributes by `dx` and `dy` respectively.
- Write a method for the `Point` class named `value` that returns a tuple of `x` and `y`. It should not print anything.
- Write a method for the `Point` class named `duplicate` that returns a new `Point` object having the same `x` and `y`. It should not print anything.

Note: Obtaining input and printing is not required.

Answer

```
class Point:
    def __init__(self, x=0, y=0): # constructor for class with default arguments
        self.x = x # assigning the value of x to object attribute x
        self.y = y # assigning the value of y to object attribute y

    def move(self, dx, dy): # method with two parameters
        self.x += dx # incrementing the value of x by dx
        self.y += dy # incrementing the value of y by dy

    def value(self):
        return (self.x, self.y) # return the tuple with the value of x and y

    def duplicate(self):
        return Point(self.x, self.y) # return a new object of Point class with the value of x and y
```

Suffix visible

```
pt1 = Point()
pt2 = pt1.duplicate()

inp1 = input().strip().split(',')
inp2 = input().strip().split(',')

pt1.move(int(inp1[0]),int(inp1[1]))
pt2.move(int(inp2[0]),int(inp2[1]))

print(f'Point 1: {pt1.value()}')
print(f'Point 2: {pt2.value()}')
```

Test case

Public

Input

```
1,1
4,5
```

Output

```
Point 1: (1, 1)
Point 2: (4, 5)
```

Private

Input

```
-3,4
0,1
```

Output

```
Point 1: (-3, 4)
Point 2: (0, 1)
```

Problem 2

Question

Write a class named `Point` having object attributes `x` and `y` taken as parameters respectively. (Use code from the previous question)

- `x` and `y` should be zero if no parameter is passed.
- Write a method for `Point` class named `move` that takes two arguments `dx` and `dy` as input and increments the `x` and `y` attributes by `dx` and `dy` respectively.
- Write a method for the `Point` class named `value` that returns a tuple of `x` and `y`. It should not print anything.
- Write a method for the `Point` class named `duplicate` that returns a new `Point` object having the same `x` and `y`. It should not print anything.

Write a class named `Line` having object attributes `startPoint` and `endPoint` taken as parameters respectively.

- `startPoint` and `endPoint` are objects of `Point` class.
- Write a method for `Line` class named `length` to return the length of the line formed by the points `startPoint` and `endPoint`.
- Write a method for `Line` class named `slope` to return the slope of the line formed by the points `startPoint` and `endPoint`. Return `math.inf` if the slope is infinite.

Note: Obtaining input and printing is not required.

Answer

```
import math
class Point:
    def __init__(self, x=0, y=0): # constructor for class with default arguments
        self.x = x # assigning the value of x to object attribute x
        self.y = y # assigning the value of y to object attribute y

    def move(self, dx, dy): # method with two parameters
        self.x += dx # incrementing the value of x by dx
        self.y += dy # incrementing the value of y by dy

    def value(self): # method with no parameter
        return (self.x, self.y) # return the tuple with the value of x and y

    def duplicate(self):
        return Point(self.x, self.y) # return a new object of Point class with the value of x and y

class Line:
    def __init__(self, startPoint, endPoint): # constructor for class with default arguments
        self.startPoint = startPoint # assigning the value of startPoint to object attribute startPoint
        self.endPoint = endPoint # assigning the value of endPoint to object attribute endPoint

    def length(self):
```

```
        x1, y1 = self.startPoint.value() # using value function of Point object to obtain x and y as tuple from startPoint
        x2, y2 = self.endPoint.value() # using value function of Point object to obtain x and y as tuple from endPoint
        return ((x2 - x1)**2 + (y2 - y1)**2)**0.5 # returning the absolute distance

    def slope(self):
        x1, y1 = self.startPoint.value()
        x2, y2 = self.endPoint.value()
        if x1 != x2: # preventing the zero division error
            return (y2 - y1) / (x2 - x1)
        else:
            return (math.inf)
```

Suffix visible

```
pt1 = Point()
pt2 = Point(3,3)

pt1.move(1,1)
pt2.move(1,2)

print(f'Point 1: {pt1.value()}')
print(f'Point 2: {pt2.value()}')

l1 = Line(pt1, pt2)
print(f'Line length: {l1.length():.2f}')
print(f'Line slope: {l1.slope():.2f}')
```

Test case

Public

Input

```
1,1
4,5
```

Output

```
Point 1: (1, 1)
Point 2: (4, 5)
Line length: 5.00
Line slope: 1.33
```

Private

Input

```
-3,4
0,1
```

Output

```
Point 1: (-3, 4)
Point 2: (0, 1)
Line length: 4.24
Line slope: -1.00
```

Problem 3

Question

Create a class `TimeConverter` that receive `time` in seconds at the time of object creation and has the following method:

- `Second_to_Minutes` that converts the value of `time` into minutes and returns the output in `minutes min second sec` format. For example: if seconds is 170, the method should return `2 min 50 sec`.
- `Second_to_Hours` that converts the value of `time` into hours and returns the output in `hours hr minutes min and seconds sec` format. For example: if seconds is 170, the method should return `0 hr 2 min 50 sec`
- `Second_to_Days` that converts the value of `time` into days and returns the output in `days days hours hr minutes min and seconds sec` format. For example: if seconds is 170, the method should return `0 days 0 hr 2 min 50 sec`

Sample Input 1

```
a = TimeConverter(35)
print(a.Second_to_Minutes())
print(a.Second_to_Hours())
print(a.Second_to_Days())
```

Sample Output 1

```
0 min 35 sec
0 hr 0 min 35 sec
0 days 0 hr 0 min 35 sec
```

Sample Input 2

```
a = TimeConverter(6582)
print(a.Second_to_Minutes())
print(a.Second_to_Hours())
print(a.Second_to_Days())
```

Sample Output 2

```
109 min 42 sec
1 hr 49 min 42 sec
0 days 1 hr 49 min 42 sec
```

Sample Input 3

```
a = TimeConverter(257865)
print(a.Second_to_Minutes())
print(a.Second_to_Hours())
print(a.Second_to_Days())
```

Sample Output 3

```
4297 min 45 sec
71 hr 37 min 45 sec
2 days 23 hr 37 min 45 sec
```

You only have to create the class. Do not create any object of the class. Objects will be created internally to verify the answer.

Answer

```
class TimeConverter:# Create class TimeConverter
    def __init__(self,time):# Create constructor for TimeConverter
        self.time=time

    def Second_to_Minutes(self):# Method for convert Second_to_Minutes
        m = self.time//60 # Calculate the minutes from second
        s = self.time % 60 # Calculate the remaining second that cannot be convert into minuts.
        return (str(m)+' min '+str(s)+' sec')
    def Second_to_Hours(self): # Method for convert Second_to_Hours
        m = self.time//60 # Calculate the minutes from second
        s = self.time % 60 # Calculate the remaining second that cannot conver into minuts.
        h = m // 60 # Calculate the hours from minuts
        m = m % 60 # Calculate the remaining minuts that cannot be convert into hours.
        return (str(h)+' hr '+str(m)+' min '+str(s)+' sec')
    def Second_to_Days(self):# Method for convert Second_to_days
        m = self.time//60 # Calculate the minutes from second
        s = self.time % 60 # Calculate the remaining second that cannot conver into minuts.
        h = m // 60 # Calculate the hours from minuts
        m = m % 60 # Calculate the remaining minuts that cannot be convert into hours.
        d = h // 24 # Calculate the days from hours
        h = h % 24 # Calculate the remaining hours that cannot be convert into days.
        return (str(d)+' days '+str(h)+' hr '+str(m)+' min '+str(s)+' sec')
```

Suffix visible

```
sec = int(input())
a = TimeConverter(sec)
print(a.Second_to_Minutes())
print(a.Second_to_Hours())
print(a.Second_to_Days())
```

Testcases

Public

Input 1

```
35
```

Output 1

```
0 min 35 sec
0 hr 0 min 35 sec
0 days 0 hr 0 min 35 sec
```

Input 2

```
6582
```

Output 2

```
109 min 42 sec
1 hr 49 min 42 sec
0 days 1 hr 49 min 42 sec
```

Input 3

```
257865
```

Output 3

```
4297 min 45 sec
71 hr 37 min 45 sec
2 days 23 hr 37 min 45 sec
```

Input 4

```
1257865
```

Output 4

```
20964 min 25 sec
349 hr 24 min 25 sec
14 days 13 hr 24 min 25 sec
```

Private

Input 1

```
0
```

Output 1

```
0 min 0 sec
0 hr 0 min 0 sec
0 days 0 hr 0 min 0 sec
```

Input 2

```
36000
```

Output 2

```
600 min 0 sec
10 hr 0 min 0 sec
0 days 10 hr 0 min 0 sec
```

Input 3

```
654768
```

Output 3

```
10912 min 48 sec
181 hr 52 min 48 sec
7 days 13 hr 52 min 48 sec
```

Input 4

```
43455444
```

Output 4

```
724257 min 24 sec
12070 hr 57 min 24 sec
502 days 22 hr 57 min 24 sec
```

Problem 4

Question

Write a class `UserLoginInfo` that has two variables:

- `UserName` that contains the username
- `old_passwords` : a list that holds the history of the user's passwords. The last item of the `old_passwords` is the user's current password.

Object creation format

```
u1 = UserLoginInfo(Username,Password)
```

At the time of object creation Username assign to `UserName` and Password will be added in to `old_passwords` list without check any validation.

Rules for valid passwords are:

- Length of password should be greater than 7
- The first character should be a letter and in upper case
- The remaining characters are either numbers or letters or a combination of both.

The class `UserLoginInfo` should have the following methods:

- `RetrievePassword` that returns the current password of the user.
- `ChangePassword(New_Password)` that receives a string `New_Password` and updates the user's current password with `New_Password`.
    - The `ChangePassword` the method should only change the password if the `New_Password` is a valid password that is different from the all of the user's old passwords . If this operation is successful, return the message `Password updated successfully` .
    - If `New_Password` is exists in the list of `old_passwords`, then return the message `Password already used`.
    - if `New_Password` is not a valid password then return the message `Invalid password`
- `Login(Username, Password)` that receives strings `UserName` and `Password` and returns a message `Welcome Username` if the `Username` and `Password` are matched to the current username and current password respectively, otherwise, return `Username or Password incorrect` message.

You only have to create the class. Do not create any object of the class. Objects will be created internally to verify the answer.

Answer

```
class UserLoginInfo: # Create class UserLoginInfo
    def __init__(self,UserName,Password):# Create constructor for
UserLoginInfo
        # Assign object variables
        self.old_passwords=[Password]
        self.UserName = UserName
    # Create method to check password validation
```

```
    def is_valid(self,pwd):
        # Length of password should be greater than 7
        if len(pwd) < 8:
            return False
        # First letter should be in uppercase
        if pwd[0].istitle()!=True:
            return False
        # Password should contain only (alphabets or numerics or both)
        if pwd.isalnum()!= True:
            return False
        else:
            return True
    # Create method fo change password
    def ChangePassword(self,newpass):
        # Check password is already used in previous
        if newpass in self.old_passwords:
            return 'Password already used'
        else:
            # Check validation of password
            if self.is_valid(newpass)==True:
                self.old_passwords.append(newpass)
                return 'Password updated successfully'
            else:
                return 'Invalid password'
    # Method for retrieve current password
    def RetrievePassword(self):
        return self.old_passwords[-1]
    # Method for login
    def Login(self,user,pas):
        # Check username and current password are matched with input or not.
        if pas==self.old_passwords[-1] and user == self.UserName :
            return 'Welcome '+str(user)
        else:
            return 'Username or Password incorrect'
```

Suffix (Visible)

```
u1 = UserLoginInfo(input(),input())
a = u1.ChangePassword(input())
b = u1.ChangePassword(input())
c = u1.ChangePassword(input())
d = u1.Login(input(),input())
e = u1.Login(input(),input())
f = u1.RetrievePassword()
print(a)
print(b)
print(c)
print(d)
print(e)
print(f)
```

Testcases

Public

Input

```
Amit
Amit1234
amit12345
Amit12345
Amit12345
amit
Amit12345
Amit
Amit12345
```

Output

```
Invalid password
Password updated successfully
Password already used
Username or Password incorrect
Welcome Amit
Amit12345
```

Private

Input-1

```
Ravi
Ravi12345
Ravi@12345
Ravi123
Ravi12345678
Ravi
Ravi123
Ravi
Ravi12345678
```

Output

```
Invalid password
Invalid password
Password updated successfully
Username or Password incorrect
Welcome Ravi
Ravi12345678
```