



Inspiring Excellence

Course Code:	CSE111
Course Title:	Programming Language II
Homework No:	04
Topic:	Instance method and overloading
Submission Type:	Hard Copy
Resources:	<ol style="list-style-type: none">1. Class lectures2. BuX lectures<ol style="list-style-type: none">a. English: https://www.youtube.com/watch?v=eUQZYzszZf8b. Supplementary: https://shorturl.at/uwENV

Task 1

Design the **Student** class with the necessary properties so that the given output is produced for the provided driver code. Use constructor overloading and method overloading where necessary.

Hint:

- A student having **cgpa** ≥ 3.5 and **credit** > 10 is **eligible for scholarship**.
 - A student having **cgpa** ≥ 3.7 is eligible for **Merit based scholarship**
 - A student with **cgpa** ≥ 3.5 but < 3.7 is eligible for **Need-based scholarship**.

Driver Code	Given Output
<pre>print('-----') std1 = Student("Alif", 3.99, 12) print('-----') std1.checkScholarshipEligibility() print('-----') std1.showDetails() print('-----') std2 = Student("Mim", 3.4) std3 = Student("Henry", 3.5, 15, "BBA") print('-----') std2.checkScholarshipEligibility() print('-----') std3.checkScholarshipEligibility() print('-----') std2.showDetails() print('-----') std3.showDetails() print('-----') std4 = Student("Bob", 4.0, 6, "CSE") print('-----') std4.checkScholarshipEligibility() print('-----') std4.showDetails()</pre>	<pre>----- ----- Alif is eligible for Merit-based scholarship. ----- Name: Alif Department: CSE CGPA: 3.99 Number of Credits: 12 Scholarship Status: Merit-based scholarship ----- ----- Mim is not eligible for scholarship. ----- Henry is eligible for Need-based scholarship. ----- Name: Mim Department: CSE CGPA: 3.4 Number of Credits: 9 Scholarship Status: No scholarship ----- Name: Henry Department: BBA CGPA: 3.5 Number of Credits: 15 Scholarship Status: Need-based scholarship ----- ----- Bob is not eligible for scholarship. ----- Name: Bob Department: CSE CGPA: 4.0 Number of Credits: 6 Scholarship Status: No scholarship</pre>

Task 2

Design the **Foodie** class with the necessary properties so that the given output is produced for the provided driver code. You can follow the notes below:

1. Your code should work for any number of strings passed to **order()** method.
2. Total spent by a foodie is calculated by adding the total prices of all the ordered foods and the waiter's tips (if any).
3. Global variable 'menu' can be accessed directly from inside the class.

Driver Code	Output
<pre>menu = {'Chicken Lollipop':15,'Beef Nugget':20,'Americano':180,'Red Velvet':150,'Prawn Tempura':80,'Saute Veg':200} f1 = Foodie('Frodo') print(f1.show_orders()) print('1-----') f1.order('Chicken Lollipop-3','Beef Nugget-6','Americano-1') print('2-----') print(f1.show_orders()) print('3-----') f1.order('Red Velvet-1') print('4-----') f1.pay_tips(20) print('5-----') print(f1.show_orders()) f2 = Foodie('Bilbo') print('6-----') f2.order('Prawn Tempura-6','Saute Veg-1') print('7-----') f2.pay_tips() print('8-----') print(f2.show_orders())</pre>	<pre>Frodo has 0 item(s) in the cart. Items: [] Total spent: 0. 1----- Ordered - Chicken Lollipop, quantity - 3, price (per Unit) - 15. Total price - 45 Ordered - Beef Nugget, quantity - 6, price (per Unit)- 20. Total price - 120 Ordered - Americano, quantity - 1, price (per Unit)- 180. Total price - 180 2----- Frodo has 3 item(s) in the cart. Items: ['Chicken Lollipop', 'Beef Nugget', 'Americano'] Total spent: 345. 3----- Ordered - Red Velvet, quantity - 1, price (per Unit)- 150. Total price - 150 4----- Gives 20/- tips to the waiter. 5----- Frodo has 4 item(s) in the cart. Items: ['Chicken Lollipop', 'Beef Nugget', 'Americano', 'Red Velvet'] Total spent: 515. 6----- Ordered - Prawn Tempura, quantity - 6, price (per Unit)- 80. Total price - 480</pre>

	Ordered - Saute Veg, quantity - 1, price (per Unit)- 200. Total price - 200 7----- No tips to the waiter. 8----- Bilbo has 2 item(s) in the cart. Items: ['Prawn Tempura', 'Saute Veg'] Total spent: 680.
--	---

Task 3

Design the **Department** class with the necessary properties so that the given output is produced for the provided driver code.

Hints:

1. Your code should work for any number of integers passed to the `add_students()` method. The method will calculate the average number of students if the number of integers passed is equal to the number of classes.
2. Your code should work for any number of Department objects passed to the `merge_Department()` method.
3. The average students of the mega department in the `merge_Department()` method are calculated in this way -

Total students of mega department = mega department average * mega department sections + department 1 average * department 1 sections + department 2 average * department 2 sections + department 3 average * department 3 sections +

Average students of mega department = (Total students of mega department / mega department sections)

Driver Code	Output
<pre> d1 = Department() print('1-----') d2 = Department('MME Department') print('2-----') d3 = Department('NCE Department', 8) print('3-----') d1.add_students(12, 23, 12, 34, 21) print('4-----') d2.add_students(40, 30, 21) print('5-----') d3.add_students(12, 34, 41, 17, 30, 22, 32, 51) </pre>	<pre> The ChE Department has 5 sections. 1----- The MME Department has 5 sections. 2----- The NCE Department has 8 sections. 3----- The ChE Department has an average of 20.4 students in each section. 4----- The MME Department doesn't have 3 sections. 5----- The NCE Department has an average of 29.88 students in each section. 6----- </pre>

```

print('6-----')
mega = Department('Engineering
Department', 10)
print('7-----')
mega.add_students(21,30,40,36,10,32,27,
51,45,15)
print('8-----')
print(mega.merge_Department(d1, d2))
print('9-----')
print(mega.merge_Department(d3))

```

The Engineering Department has 10 sections.

7-----

The Engineering Department has an average of 30.7 students in each section.

8-----

ChE Department is merged to Engineering Department.

MME Department is merged to Engineering Department.

Now the Engineering Department has an average of 40.9 students in each section.

9-----

NCE Department is merged to Engineering Department.

Now the Engineering Department has an average of 64.8 students in each section.

Task 4

Design the **Shopidify** class such that users can create 2 types of account `guest_accounts` and `user_accounts` to shop from the online e-commerce site.

Now create the methods and constructors using overloading concepts to facilitate the online shopping procedure.

Use constructor overloading for handling the `guest_accounts` and `user_accounts`.

[You are not allowed to change the driver code.]

Tester Code	Output
<pre># Test the Shopidify class guest_account = Shopidify() print("1xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx") john_account = Shopidify("John") print("2xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx") guest_account.add_to_cart("Air Jordan", 2) guest_account.add_to_cart("Luffy Action Figure") guest_account.display_cart() print("3xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx") john_account.add_to_cart(["Chocolate Chip Cookies", 3, "Goku Action Figure", 2, "Dumbbells-5kg", 2]) john_account.display_cart() print("4xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx") guest_account.add_to_cart("Air Jordan") guest_account.display_cart() print("5xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx") guest_account.checkout() print("6xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx") guest_account.display_history() print("7xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx") john_account.checkout() print("8xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx") john_account.display_history() print("9xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")</pre>	<pre>Welcome to Shopidify 1xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx Welcome John to Shopidify 2xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx Items in the cart for Guest: - Air Jordan: 2x - Luffy Action Figure: 1x 3xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx Items in the cart for John: - Chocolate Chip Cookies: 3x - Goku Action Figure: 2x - Dumbbells-5kg: 2x 4xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx Items in the cart for Guest: - Air Jordan: 3x - Luffy Action Figure: 1x 5xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx Checkout completed for Guest 6xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx Purchase history for Guest: Transaction 1: - Air Jordan: 3x - Luffy Action Figure: 1x 7xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx Checkout completed for John 8xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx Purchase history for John: Transaction 1: - Chocolate Chip Cookies: 3x - Goku Action Figure: 2x - Dumbbells-5kg: 2x 9xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx</pre>

Task 5

Write the **Author** class with the required methods to give the following outputs as shown.

Write your code here

Do not change the following lines of code.

```
a1 = Author()
print("=====")
a1.addBook("Ice", "Science Fiction")
print("=====")
a1.setName("Anna Kavan")
a1.addBook("Ice", "Science Fiction")
a1.printDetail()
print("=====")
a2 = Author("Humayun Ahmed")
a2.addBook("Onnobhubon", "Science Fiction")
a2.addBook("Megher Upor Bari", "Horror")
print("=====")
a2.printDetail()
a2.addBook("Ireena", "Science Fiction")
print("=====")
a2.printDetail()
print("=====")
```

=====

A book can not be added without author
name

=====

Number of Book(s): 1

Author Name: Anna Kavan

Science Fiction: Ice

=====

=====

Number of Book(s): 2

Author Name: Humayun Ahmed

Science Fiction: Onnobhubon

Horror: Megher Upor Bari

=====

Number of Book(s): 3

Author Name: Humayun Ahmed

Science Fiction: Onnobhubon, Ireena

Horror: Megher Upor Bari

=====

Task 6

1	<code>class Lab4:</code>
2	<code> def __init__(self):</code>
3	<code> self.x = 3</code>
4	<code> self.y = 2</code>
5	<code> self.sum = 5</code>
6	<code> def methodA(self, x):</code>
7	<code> self.y = self.sum + self.x - x</code>
8	<code> self.sum = x - self.y</code>
9	<code>d = Lab4()</code>
10	<code>d.sum = self.sum + self.methodB(d)</code>
11	<code>print(self.x, self.y, self.sum)</code>
12	<code>return d</code>
13	<code> def methodB(self, t, z = 4):</code>
14	<code> y = 2</code>
15	<code> t.x = self.x + self.sum</code>
16	<code> y = y + t.x - t.y</code>
17	<code> self.sum = t.x + t.y + y - z</code>
18	<code> if z == 4:</code>
19	<code> return y</code>
20	<code>print(t.x, t.y, self.sum)</code>
21	<code>p = t.methodA(y)</code>
22	<code>print(t.x, self.y, p.sum)</code>

<pre>obj = Lab4() obj2 = obj.methodA(4) obj.methodB(obj2, 10)</pre>	Output:		

Task 7

1	<code>class Test4:</code>
2	<code> def __init__(self):</code>
3	<code> self.sum, self.y = 0, 0</code>
4	<code> def methodA(self):</code>
5	<code> x, y = 0, 0</code>
6	<code> msg = [0]</code>
7	<code> msg[0] = 5</code>
8	<code> y = y + self.methodB(msg[0])</code>
9	<code> x = y + self.methodB(msg, msg[0])</code>
10	<code> self.sum = x + y + msg[0]</code>
11	<code> print(x, y, self.sum)</code>
12	<code> def methodB(self, *args):</code>
13	<code> if len(args) == 1:</code>
14	<code> mg1 = args[0]</code>
15	<code> x, y = 0, 0</code>
16	<code> y = y + mg1</code>
17	<code> x = x + 33 + mg1</code>
18	<code> self.sum = self.sum + x + y</code>
19	<code> self.y = mg1 + x + 2</code>
20	<code> print(x, y, self.sum)</code>
21	<code> return y</code>
22	<code> else:</code>
23	<code> mg2, mg1 = args</code>
24	<code> x = 0</code>
25	<code> self.y = self.y + mg2[0]</code>
26	<code> x = x + 33 + mg1</code>
27	<code> self.sum = self.sum + x + self.y</code>
28	<code> mg2[0] = self.y + mg1</code>
29	<code> mg1 = mg1 + x + 2</code>
30	<code> print(x, self.y, self.sum)</code>
31	<code> return self.sum</code>

<pre>t3 = Test4() t3.methodA() t3.methodA() t3.methodA() t3.methodA()</pre>	x	y	sum