



Inspiring Excellence

Course Code:	CSE111
Course Title:	Programming Language II
Homework No:	11
Topic:	Inheritance
Submission Type:	Will be notified later on.
Resources:	<ol style="list-style-type: none">1. Class lectures2. BuX lectures<ol style="list-style-type: none">a. English:<ol style="list-style-type: none">i. Inheritance: hereb. Supplementary:<ol style="list-style-type: none">i. Inheritance: here

Task 1

A multinational company has two special types of regular employees. One is Foreign employees and another one is Part time employees. Design the Employee (parent), Foreign_employee(child) and Parttime_employee(child) classes so that the following output is produced. The Foreign_employee and Parttime_employee classes should inherit the Employee class. Note that:

- Basic salary of a Regular, Foreign employee is 30,000 and for Part-time employees basic is 15,000.
- Regular employees get 10% increment on their salary and Foreign employees get 15% increment on their basic salary.
- Employees from the HR department will collect their work distribution load from the manager, and others will collect their work distribution load from the HR department.

Driver Code	Output
<pre> print("1-----") emp1=Employee("Nawaz Ali", 102, "Marketing") print("2-----") emp1.employeeDetails() print("3-----") emp1.workDistribution("Marketing") print("4-----") emp1.increment() emp1.employeeDetails() print("5-----") f_emp=Foreign_employee("Nadvi", 311, "Human Resource") f_emp.employeeDetails() print("6-----") f_emp.workDistribution("Human Resource") print("7-----") f_emp.increment() f_emp.employeeDetails() print("8-----") p1_emp=Part_time_employee("Asif", 210, "Sales") p2_emp=Part_time_employee("Olive", 223, "Accounts") print("9-----") p1_emp.employeeDetails() print("10-----") p1_emp.workDistribution("Sales") print("11-----") p2_emp.increment() </pre>	<pre> 1----- 2----- Name: Nawaz Ali, Dept Marketing Employee id: 102, Salary: 30000 3----- Collect work distribution loads from the HR department. 4----- Name: Nawaz Ali, Dept Marketing Employee id: 102, Salary: 33000.0 5----- Name: Nadvi, Dept Human Resource Employee id: 311, Salary: 30000 Employee Type: Foreign 6----- Collect work distribution details from the manager. 7----- Name: Nadvi, Dept Human Resource Employee id: 311, Salary: 34500.0 Employee Type: Foreign 8----- 9----- Name: Asif, Dept Sales Employee id: 210, Salary: 15000 Employee Type: Part Time 10----- Collect work distribution loads from the HR department. 11----- Sadly, there is no increment for the part time </pre>

<pre>print("12-----") p2_emp.employeeDetails()</pre>	<pre>employees!! 12----- Name: Olive, Dept Accounts Employee id: 223, Salary: 15000 Employee Type: Part Time</pre>
--	--

Task 2

Write the **ScienceExam** class so that the following code generates the output below:

<pre>class Exam: def __init__(self,marks): self.marks = marks self.time = 60 def examSyllabus(self): return "Maths , English" def examParts(self): return "Part 1 - Maths\nPart 2 - English\n" engineering = ScienceExam(100,90,"Physics","HigherMaths") print(engineering) print('-----') print(engineering.examSyllabus()) print(engineering.examParts()) print('=====') architecture = ScienceExam(100,120,"Physics","HigherMaths","Drawing") print(architecture) print('-----') print(architecture.examSyllabus()) print(architecture.examParts())</pre>	<pre>OUTPUT: Marks: 100 Time: 90 minutes Number of Parts: 4 ----- Maths , English , Physics , HigherMaths Part 1 - Maths Part 2 - English Part 3 - Physics Part 4 - HigherMaths ===== Marks: 100 Time: 120 minutes Number of Parts: 5 ----- Maths , English , Physics , HigherMaths , Drawing Part 1 - Maths Part 2 - English Part 3 - Physics Part 4 - HigherMaths Part 5 - Drawing</pre>
--	---

Task 3

Write the **PokemonExtra** class so that the following code generates the output below:

```
class PokemonBasic:

    def __init__(self, name = 'Default', hp = 0,
weakness = 'None', type = 'Unknown'):
        self.name = name
        self.hit_point = hp
        self.weakness = weakness
        self.type = type

    def get_type(self):
        return 'Main type: ' + self.type

    def get_move(self):
        return 'Basic move: ' + 'Quick Attack'

    def __str__(self):
        return "Name: " + self.name + ", HP: " +
str(self.hit_point) + ", Weakness: " + self.weakness

print('\n-----Basic Info:-----')
pk = PokemonBasic()
print(pk)
print(pk.get_type())
print(pk.get_move())

print('\n-----Pokemon 1 Info:-----')
charmander = PokemonExtra('Charmander', 39, 'Water',
'Fire')
print(charmander)
print(charmander.get_type())
print(charmander.get_move())

print('\n-----Pokemon 2 Info:-----')
charizard = PokemonExtra('Charizard', 78, 'Water',
'Fire', 'Flying', ('Fire Spin', 'Fire Blaze'))
print(charizard)
print(charizard.get_type())
print(charizard.get_move())
```

OUTPUT:

```
-----Basic Info:-----
Name: Default, HP: 0, Weakness: None
Main type: Unknown
Basic move: Quick Attack

-----Pokemon 1 Info:-----
Name: Charmander, HP: 39, Weakness: Water
Main type: Fire
Basic move: Quick Attack

-----Pokemon 2 Info:-----
Name: Charizard, HP: 78, Weakness: Water
Main type: Fire, Secondary type: Flying
Basic move: Quick Attack
Other move: Fire Spin, Fire Blaze
```

Task 4

A renowned Bakery shop recently launched cheesecakes into their cakes menu. Cheesecakes will have all the general attributes of the regular cakes but it has some special features. Design the **Cakes** (parent) and **Cheese_Cakes** (child) classes so that the following output is produced. Note that:

- 1kg regular cake price is 1200 Taka and 1 kg cheese-cake price is 1500 Taka
- As cheese-cakes are newly launched, they need user feedback. For this reason, if a customer gives feedback on cheese-cakes he'll get 10% discounts on his next purchase.

Write the classes **Cakes** and **Cheese_Cakes** to generate the following output.

Driver Code:	Output:
<pre>order_1=Cakes("Chocolate",500) order_2=Cakes("Vanilla",800) print("(1)*****") order_1.cake_details() print("(1.1)*****") print(Cakes.order_list) print("(2)*****") order_2.add_customization("Zero","Butter Cream") order_2.cake_details() print("(3)*****") Cakes.give_feedbacks("Chocolate Cake","Very Delicious") Cakes.give_feedbacks("Chocolate Cake","Yummy") print("(4)*****") Cakes.show_feedbacks() print("(5)*****") ch_order1=Cheese_Cakes("Red velvet",700) ch_order1.cake_details() print("(6)*****") ch_order1.add_customization() print("(7)*****") ch_order2=Cheese_Cakes("Blue Berry",900,"No Bake") ch_order2.cake_details() print("(8)*****") print(Cakes.order_list) print("(9)*****") Cheese_Cakes.give_feedbacks("Red velvet Cheese Cake","Average") Cheese_Cakes.give_feedbacks("Blue Berry Cheese Cake","Liked it") print("(10)*****") Cakes.show_feedbacks()</pre>	<pre>(1)***** Flavor: Chocolate Cake, Weight: 500 gm Sweetness: Moderate sugar, Cream Type: Whipped Cream Price: 600.0 Taka (1.1)***** {'Chocolate Cake 500gm': 1, 'Vanilla Cake 800gm': 1} (2)***** Flavor: Vanilla Cake, Weight: 800 gm Sweetness: Zero sugar, Cream Type: Butter Cream Price: 960.0 Taka (3)***** Thanks for your valuable feedback! Thanks for your valuable feedback! (4)***** {'Chocolate Cake': ['Very Delicious', 'Yummy']} (5)***** Flavor: Red velvet Cheese Cake, Weight: 700 gm Sweetness: Moderate sugar, Cream Type: Cream Cheese Cake Type:baked, Price: 1050.0 Taka (6)***** Sorry! No customization available for cheese cakes. (7)***** Flavor: Blue Berry Cheese Cake, Weight: 900 gm Sweetness: Moderate sugar, Cream Type: Cream Cheese Cake Type:No Bake, Price: 1350.0 Taka (8)***** {'Chocolate Cake 500gm': 1, 'Vanilla Cake 800gm': 1, 'Red velvet Cheese Cake 700gm': 1, 'Blue Berry Cheese Cake 900gm': 1} (9)***** Thanks for your valuable feedback! You will get 10% discount for your next purchase!</pre>

	<p>Thanks for your valuable feedback!</p> <p>You will get 10% discount for your next purchase!</p> <p>(10)*****</p> <p>{'Chocolate Cake': ['Very Delicious', 'Yummy'], 'Red velvet Cheese Cake': ['Average'], 'Blue Berry Cheese Cake': ['Liked it']}</p>
--	---

Task 5

1	<code>class A:</code>
2	<code>temp = 3</code>
3	<code>def __init__(self):</code>
4	<code>self.sum = 0</code>
5	<code>self.y = 0</code>
6	<code>self.y = A.temp - 1</code>
7	<code>self.sum = A.temp + 2</code>
8	<code>A.temp -= 2</code>
9	
10	<code>def methodA(self, m, n):</code>
11	<code>x = 0</code>
12	<code>n[0] += 1</code>
13	<code>self.y = self.y + m + A.temp</code>
14	<code>A.temp += 1</code>
15	<code>x = x + 2 + n[0]</code>
16	<code>n[0] = self.sum + 2</code>
17	<code>print(f"{x} {self.y} {self.sum}")</code>
18	

19	<code>class B(A):</code>
20	<code> x = 1</code>
21	<code> def __init__(self, b = None):</code>
22	<code> super().__init__()</code>
23	<code> self.sum = 2</code>
24	<code> if b == None:</code>
25	<code> self.y = self.temp + 1</code>
26	<code> B.x = 3 + A.temp + self.x</code>
27	<code> A.temp -= 2</code>
28	<code> else:</code>
29	<code> self.sum = self.sum + self.sum</code>
30	<code> B.x = b.x + self.x</code>
31	<code> def methodB(self, m, n):</code>
32	<code> y = [0]</code>
33	<code> self.y = y[0] + self.y + m</code>
34	<code> B.x = self.y + 2 + self.temp - n</code>
35	<code> self.methodA(self.x, y)</code>
36	<code> self.sum = self.x + y[0] + self.sum</code>
37	<code> print(f"{self.x} {y[0]} {self.sum}")</code>

Write the output of the following code:

<pre>x = [23] a1 = A() b1 = B()</pre>	Output:		
	x	y	sum

<pre>b2 = B(b1) a1.methodA(1, x) b2.methodB(3, 2) a1.methodA(1, x)</pre>			