Inspiring Excellence

Course Code:	**CSE111**
Course Title:	**Programming Language II**
Classwork No:	**03**
Topic:	**OOP (Classes and Objects)**
Number of tasks:	**6**

Classwork Part

Task 1

You have been hired by a bank to develop cutting-edge banking software. As part of your task,
(i) Design a class called **BankAccount** that represents a bank account that produces the expected output.

Driver code

```
account1 = BankAccount("Bilbo", "Savings")
print("======================================")
print(f"User Name: {account1.user_name}")
print(f"Balance: {account1.balance}")
print(f"Account Type:", account1.account_type)
print("======================================")
account2 = BankAccount("Frodo", "Business")
print(f"User Name: {account2.user_name}")
print(f"Balance: {account2.balance}")
print(f"Account Type: {account2.account_type}")
print("======================================")
```

Output:

```
======================================
User Name: Bilbo
Balance: 1.0
Account Type: Savings
======================================
User Name: Frodo
Balance: 1.0
Account Type: Business
======================================
```

(ii) Now change the balance of account1 to 15.75 taka and account2 to 700.50 taka. Finally, print the updated details with their respective user names.

Output:

```
New account balance of Bilbo is 15.75
New account balance of Frodo is 700.5
```

Task 2

You are the proud owner of a magnificent mango orchard where you have different varieties of mango trees such as Fazlee, Langda, Harivanga, Himsagar, etc. But this year, demand for Gopalbhog and Amrapali is too high. To cater to this unsatisfied demand in the future, you have decided to plant these two varieties in your orchard.

(i) Now, based on the given driver code, you need to design a **MangoTree** class.
Initially, when you plant a tree, it will have a height of 1 meter and the number of mangoes will be 0.

Driver code

```
mangoTree1= MangoTree("Gopalbhog")
# Display the details of the mango tree
print("====================================")
print("Mango Tree Details:")
print(f"Variety: {mangoTree1.variety}")
print(f"Height: {mangoTree1.height} meter(s)")
print(f"Number of mangoes on the tree: {mangoTree1.number_of_mangoes}")
print("====================================")
mangoTree2= MangoTree("Amrapali")
# Display the details of the mango tree
print("Mango Tree Details:")
print(f"Variety: {mangoTree2.variety}")
print(f"Height: {mangoTree2.height} meter(s)")
print(f"Number of mangoes on the tree: {mangoTree2.number_of_mangoes}")
print("====================================")
```

Output:

```
====================================
Mango Tree Details:
Variety: Gopalbhog
Height: 1 meter(s)
Number of mangoes on the tree: 0
====================================
```

Mango Tree Details:
Variety: Amrapali
Height: 1 meter(s)
Number of mangoes on the tree: 0
==========

(ii) Suppose 5 years have passed and these small trees have grown larger and started to bear fruit.

- A mango tree roughly grows 3 meters in one year.
- Amrapali bears 15 mangoes per meter and Gopalbhog bears 10 mangoes per meter.

Now, update the height and number_of_mangoes of both mangoTree1 and mangoTree2, respectively, in the driver code using the formula mentioned above. And print the updated details.

Output:

Updated details after 5 years:
==========

Variety: Gopalbhog
Height: 16 meter(s)
Number of mangoes on the tree: 160
==========

Variety: Amrapali
Height: 16 meter(s)
Number of mangoes on the tree: 240
==========

Task 3

Design the class “Contacts” so that the code produces the expected output. You are not allowed to change the given driver code below.

Driver code

```
names = ["Emergency", "Father", "Bestie"]
numbers = ["999", "01xx23", "01xx87", "01xx65", "01xx43"]

m1 = Contacts(names, numbers)
print("Saved Contacts:", m1.contactDict)
print("----------------------------------------")

names.append("Mother")
numbers.pop()

m2 = Contacts(names, numbers)
print("Saved Contacts:", m2.contactDict)
```

Output:

```
Contacts cannot be saved. Length Mismatch!
Saved Contacts: {}
----------------------------------------
Contacts saved successfully.
Saved Contacts: {'Emergency': '999', 'Father': '01xx23', 'Bestie': '01xx87', 'Mother': '01xx65'}
```

Subtasks:

Suppose, after running the above code, you again write the following line:

m1 = Contacts(names, numbers)

1. What will happen after executing this line?
2. Since m1 was already created in the 3rd line of the driver code, will the newly created m1 have the same reference now? Explain by writing/modifying a few lines of code.
   [Hint: Store the previously created m1 in a temporary variable and compare it with the new one.]

Task 4

Design the class **"Student"** so that the code produces the expected output. You are not allowed to change the given driver code below.
[Hint:

- If a student's CGPA is less than 2.00 then s/he is under probation. Probation students can not take less than 1 course and more than 2 courses; otherwise, his/her advising request will be denied.
- If a student's CGPA is greater than or equal to 2.00, the student will fall under the regular category. Regular students must take at least 3 courses and at most 5 courses, or else his/her advising request will be denied.
- If a student's advising request is denied, that means his/her currently taken courses are not finalized. So, the value should be set to 0.]

Driver code

```
s1 = Student("Clark", 3.45, 4)
print(f"Name: {s1.name}\nCGPA: {s1.cgpa}\nCourses Taken: {s1.courses_taken}")
print(f"Student Status: {s1.student_status}\nAdvising Status: {s1.advising_status}")
print("--------------------------------------------------------------------------------")
s2 = Student("Barry", 1.93, 2)
print(f"Name: {s2.name}")
print(f"Student Status: {s2.student_status}\nAdvising Status: {s2.advising_status}")
print("--------------------------------------------------------------------------------")
s3 = Student("Diana", 2.91, 2)
print(f"Advising Status: {s3.advising_status}\nCourses Taken: {s3.courses_taken}")
print("--------------------------------------------------------------------------------")
s4 = Student("Bruce", 1.52, 5)
print(f"Advising Status: {s4.advising_status}\nCourses Taken: {s4.courses_taken}")
```

Output:

```
All the best, Clark, for the upcoming semester.
Name: Clark
CGPA: 3.45
Courses Taken: 4
Student Status: Regular
Advising Status: Approved
--------------------------------------------------------------------------------
Study hard this time, Barry.
Name: Barry
```

Student Status: Probation
Advising Status: Approved

Hello Diana, You are a regular student and have to take between 3 to 5 courses.
Advising Status: Denied
Courses Taken: 0

Sorry, Bruce, you are on probation and cannot take more than 2 courses.
Advising Status: Denied
Courses Taken: 0

Task 5

Design the **CellPackage** class and write suitable driver code to produce the output:
Subtasks:

- (#1) **Assign** the arguments into appropriate attributes: **data**, **talk_time**, **messages**, **cashback**, **validity** and **price** via a parameterized constructor. All the attributes should be of int data type. Note that **data** is stored in *Megabytes* (*1 GB = 1024 MB*) and the **cashback** amount is calculated from a percentage value.
- (#2,3,4) **Implement** driver code to display all the information of a package. **Check** if any particular attribute does not exist (is equal to 0), do not print that attribute. Attributes **validity** and **price** are always printed.

```
# Driver Code
# Subtask 1: Write the CellPackage Class

pkg = CellPackage(150, '6 GB', 99, 20,
'7%', 7)
print('==========Package
1============')
# Subtask 2: Check each attribute and
print

pkg2 = CellPackage(700, '35 GB', 700, 0,
'10%', 30)
print('==========Package
2============')
# Subtask 3: Check each attribute and
print

pkg4 = CellPackage(120, '0 GB', 190, 0,
'0%', 10)
print('==========Package
3============')
# Subtask 4: Check each attribute and
print
```

```
============Package 1============
Data = 6144 MB
Talktime = 99 Minutes
SMS/MMS = 20
Validity = 7 Days
--> Price = 150 tk
Buy now to get 10 tk cashback.
============Package 2============
Data = 35840 MB
Talktime = 700 Minutes
Validity = 30 Days
--> Price = 700 tk
Buy now to get 70 tk cashback.
============Package 3============
Talktime = 190 Minutes
Validity = 10 Days
--> Price = 120 tk
```

TASK 6

Write the output of the following code:

		Output
1	`class Human:`	
2	`    def __init__(self):`	
3	`        self.age = 0`	
4	`        self.height = 0.0`	
5		
6	`h1 = Human()`	
7	`h2 = Human()`	
8	`h1.age = 21`	
9	`h1.height = 5.5`	
10	`print(h1.age)`	
11	`print(h1.height)`	
12	`h2.height = h1.height - 3`	
13	`print(h2.height)`	
14	`h2.age = h1.age`	
15	`h1.age += h1.age`	
16	`print(h1.age)`	
17	`h2 = h1`	
18	`print(h2.age)`	
19	`print(h2.height)`	
20	`h1.age += h1.age`	
21	`h2.height += h2.height`	
22	`print(h1.age)`	
23	`print(h1.height)`	
24	`h2.age += h2.age`	
25	`h1.age = h2.age`	

26	`print(h2.age)`	

Homework Part

Homework No:	**03**
Topic:	**OOP(Classes and objects)**
Submission Type:	**Hard Copy (Only submit the part of the code that you have been instructed to write. DO NOT write any given code.)**
Resources:	**1. Class lectures** **2. BuX lectures** **a. English: https://shorturl.at/dhjAZ** **b. Supplementary: https://shorturl.at/wMPRU**

Task 1

Suppose you are hired by the authority of a library that only keeps antique books. They want you to build a management system to keep track of the books. They have given an instruction manual:

1. Only books that are antique enough are kept in the library. [**hint:** ***analyze the driver code and the output to find the minimum age of a book to be considered antique***.]

2. Library has 3 floors. Floor 0,1 and 2. The comparatively antique books are stored on the higher floors. Books that were published less than 200 years ago are stored on floor 0, published at least 200 years ago are stored on floor 1, and published at least 400 years ago are stored on floor 2.

Now, design a class called '**Book**' using a parameterized constructor so that after executing the driver code, the desired result shown in the output section will be printed.

Hint: ***You can use the following lines of code to find out the current year.***

import datetime #Import this module at the top of your code.

today = datetime.date.today() #This will give you today's date.

year = today.year #Extracting the year from today's date.

#Driver Code

```
book1= Book('The Act', 'Ferguson', 1924)
print(f"{book1.author} wrote the book '{book1.name}'.")
```

```
print(f"This book was published in
{book1.year_of_publication}.")
print(f"This book is {book1.status}")
print("-------------------------------------")
book2= Book('Flame', 'Nolan', 1932)
print(f"{book2.author} wrote the book '{book2.name}'.")
print(f"This book was published in
{book2.year_of_publication}.")
print(f"This book is {book2.status}")
print("-------------------------------------")
book3= Book('Norms', 'Alfred', 1832)
print(f"{book3.author} wrote the book '{book3.name}'.")
print(f"This book was published in
{book3.year_of_publication}.")
print(f"This book is {book3.status}")
print("-------------------------------------")
book4= Book('Apex', 'Samson', 1923)
print(f"{book4.author} wrote the book '{book4.name}'.")
print(f"This book was published in
{book4.year_of_publication}.")
print(f"This book is {book4.status}")
print("-------------------------------------")
book5= Book('Habitat', 'Eden', 1723)
print(f"{book5.author} wrote the book '{book5.name}'.")
print(f"This book was published in
{book5.year_of_publication}.")
print(f"This book is {book5.status}")
print("-------------------------------------")
book6= Book('Apocalypto', 'Menez', 1603)
print(f"{book6.author} wrote the book '{book6.name}'.")
print(f"This book was published in
{book6.year_of_publication}.")
print(f"This book is {book6.status}")
```

Output:

Checking the book.
Ferguson wrote the book 'The Act'.
This book was published in 1924.
This book is Rejected. The book is not antique enough.

Checking the book.

Nolan wrote the book 'Flame'.
This book was published in 1932.
This book is Rejected. The book is not antique enough.

Checking the book.
Alfred wrote the book 'Norms'.
This book was published in 1832.
This book is Accepted. The book is stored on floor: 0.

Checking the book.
Samson wrote the book 'Apex'.
This book was published in 1923.
This book is Accepted. The book is stored on floor: 0.

Checking the book.
Eden wrote the book 'Habitat'.
This book was published in 1723.
This book is Accepted. The book is stored on floor: 1.

Checking the book.
Menez wrote the book 'Apocalypto'.
This book was published in 1603.
This book is Accepted. The book is stored on floor: 2.

TASK 2

Design a class called **Pokemon** using a parameterized constructor so that after executing the following line of code the desired result shown in the output box will be printed.

The first object along with print has been done for you, you also need to create other objects and print accordingly to get the output correctly.

Subtasks:

1. Design the **Pokemon** class using a parameterized constructor.
   The 5 values that are being passed through the constructor are respectively:
   1. pokemon 1 name,
   2. pokemon 2 name,
   3. pokemon 1 power,
   4. pokemon 2 power and
   5. damage rate

2. Create an object named **team_bulb** and pass the values `'bulbasaur', 'squirtle', 80, 70, 9` respectively.
3. Use print statements accordingly to print the desired result of **team_bulb**.

[You are not allowed to change the code below]

```
# Write your code for class here

team_pika = Pokemon('pikachu', 'charmander', 90, 60, 10)
print('=======Team 1=======')
print('Pokemon 1:',team_pika.pokemon1_name,
team_pika.pokemon1_power)
print('Pokemon 2:',team_pika.pokemon2_name,
team_pika.pokemon2_power)
pika_combined_power = (team_pika.pokemon1_power +
team_pika.pokemon2_power) * team_pika.damage_rate
print('Combined Power:', pika_combined_power)

# Write your code for subtask 2 and 3 here
```

Expected Output:

```
=======Team 1=======
Pokemon 1: pikachu 90
Pokemon 2: charmander 60
Combined Power: 1500
=======Team 2=======
Pokemon 1: bulbasaur 80
Pokemon 2: squirtle 70
Combined Power: 1350
```

TASK 3

Part A

Write the **box** class so that the given drive code gives the expected output.

[You are not allowed to change the code below]

```
# Write your class code here


print("Box 1")
b1 = box([10,10,10])
```

Expected Output:

```
Box 1
Creating a Box!
========================
```

```
print("=========================")
print("Height:", b1.height)
print("Width:", b1.width)
print("Breadth:", b1.breadth)
volume = b1.height * b1.width *
b1.breadth
print(f"Volume of the box is {volume}
cubic units.")
print("-------------------------")
print("Box 2")
b2 = box((30,10,10))
print("=========================")
print("Height:", b2.height)
print("Width:", b2.width)
print("Breadth:", b2.breadth)
volume = b2.height * b2.width *
b2.breadth
print(f"Volume of the box is {volume}
cubic units.")
b2.height = 300
print("Updating Box 2!")
print("Height:", b2.height)
print("Width:", b2.width)
print("Breadth:", b2.breadth)
volume = b2.height * b2.width *
b2.breadth
print(f"Volume of the box is {volume}
cubic units.")
print("-------------------------")
print("Box 3")
b3 = b2
print("Height:", b3.height)
print("Width:", b3.width)
print("Breadth:", b3.breadth)
volume = b3.height * b3.width *
b3.breadth
print(f"Volume of the box is {volume}
cubic units.")
```

```
Height: 10
Width: 10
Breadth: 10
Volume of the box is 1000
cubic units.
-------------------------
Box 2
Creating a Box!
=========================
Height: 30
Width: 10
Breadth: 10
Volume of the box is 3000
cubic units.
Updating Box 2!
Height: 300
Width: 10
Breadth: 10
Volume of the box is 30000
cubic units.
-------------------------
Box 3
Height: 300
Width: 10
Breadth: 10
Volume of the box is 30000
cubic units.
```

Part B

After the given driver code, if we run the following lines of code:

```
one = (b3 == b2)

b3.width = 100
two = (b3 == b2)
```

1. What will be the values of the variables `one` and `two`? Explain your answer briefly in text.
2. What will be the value of `b2.width`? Has that value changed since the driver code ran? If yes, explain why in brief text.

TASK 4

Design the **Order** class so that it generates the expected output for the given Driver code. The **Order** class has an attribute named ***items***, which is a list that is created in the following pattern:

[i_1, q_1, p_1, i_2, q_2, p_2, i_3, q_3, p_3, . . .].

Here, ***i_x***, ***q_x,*** and ***p_x*** refer to the **item name**, **quantity ordered,** and **subtotal price** of the **x-th ordered item** respectively.

[You are not allowed to change the code below]

```
# Write your class code here

menu = {
    'Chicken_Cheeseburger' : 249,
    'Mega_Cheeseburger' : 289,
    'Fries' : 139,
    'Hot_Wings' : 99,
    'Rice_Bowl' : 299,
    'Soft_Drinks' : 50
}

order1 = Order(menu, "Chicken_Cheeseburger-2, Fries-3,
Soft_Drinks-3")
print(order1.items)
print()
```

```
print('-'*35)
print('Item           x Quantity :   Price')
print('--------------   --------   -------')

index = 0
total = 0
while index < len(order1.items):
  item = order1.items[index]
  quantity = order1.items[index+1]
  price = order1.items[index+2]

  print(f'{item:20} x {quantity:2} : {price:7.2f}')
  total += price
  index += 3 # Going to next item

print('-'*35)
print(f'Total:                      {total:7.2f}')
print('-'*35)
```

Expected Output:

```
['Chicken_Cheeseburger', 2, 498, 'Fries', 3, 417, 'Soft_Drinks', 3,
150]

-----------------------------------
Item           x Quantity :   Price
--------------   --------   -------
Chicken_Cheeseburger x  2 :  498.00
Fries                x  3 :  417.00
Soft_Drinks          x  3 :  150.00
-----------------------------------
Total:                      1065.00
-----------------------------------
```

TASK 5

		Output
1	`class Student:`	
2	`    def __init__(self):`	
3	`        self.name = None`	
4	`        self.cgpa = 0.0`	
5	`s1 = Student()`	
6	`s2 = Student()`	
7	`s3 = None`	
8	`s1.name = "Student One"`	
9	`s1.cgpa = 2.3`	
10	`s3 = s1`	
11	`s2.name = "Student Two"`	
12	`s2.cgpa = s3.cgpa + 1`	
13	`s3.name = "New Student"`	
14	`print(s1.name)`	
15	`print(s2.name)`	
16	`print(s3.name)`	
17	`print(s1.cgpa)`	
18	`print(s2.cgpa)`	
19	`print(s3.cgpa)`	
20	`s3 = s2`	
21	`s1.name = "old student"`	
22	`s2.name = "older student"`	
23	`s3.name = "oldest student"`	
24	`s2.cgpa = s1.cgpa - s3.cgpa + 4.5`	
25	`print(s1.name)`	
26	`print(s2.name)`	
27	`print(s3.name)`	
28	`print(s1.cgpa)`	
29	`print(s2.cgpa)`	
30	`print(s3.cgpa)`	