Practice makes your coding skill perfect. So, whenever you are given an exercise, please try and practice it.

Remember that computer coding skills need a systematic approach.

Therefore, every week's learning is built on the current week and previous week learning.

Lab 7 Classes and Objects

1. Class Product

a. Define a class named **Product** that holds the following data about a product in attributes **id**, **name**, and **price**.

Include the following:

- A constructor to initialise the product's id, name and price,
- A function to set the price,
- A function to display the object for inspection purpose.

b. Write statements to

- Create a product
- Display it
- Get its id, name, and price
- Increase the product's price by some percentage

2. The Sheep Counter

In this question, you will model a sheep counter. A sheep counter is a hand-held device that shepherds often use to count sheep. The counter can count from 0 to 999. It has two buttons: the click button and the clear button. A press on the click button will increase the counter value by 1. A press on this button when the counter value is 999 would return the counter to 0. A press on the clear button will set the counter value to 0. The sheep counter can be represented by a class with one attribute (value) and two operations (click and clear). For ease of testing, we will let our counter count up to 10 only.

a. Create the class.

b. Write statements to test it.

Note: We can declare a variable to hold the maximum counter value in the class:

```
class Counter:
    MAX_COUNT =
    999
    ---
```

Functions of the class can access this maximum value as

Counter.MAX_COUNT

Also, to make it easy to do the testing, you can set the maximum count to 10, for example.

3. Class Student

a. Define a class named **Student** that holds the following data about a student in attributes **id**, **name**, and **marks**. Attribute **marks** is a list that holds the marks of three assessment components, assignment 1, assignment 2 and the exam, with the weights of 30%, 30% and 40%, respectively. Include the following:

- A constructor to initialise the student's id, name and marks,
- A method to change a particular mark (the mark to be changed is specified by its index in the list),
- A method to display the student's details for inspection purpose.

b. Write statements to

- Create a student
- Display all of the student's details
- Get the student's id, name and marks (one by one)
- Change at least the mark of one assessment component.

In your testing, observe the following usage-conforming rules:

- The student id is a non-empty string
- The student's name is a non-empty string
- **marks** are a list of three float values between 0 and 100, inclusive

state your assumptions, if any.

4. Class Catalog (A Class to Maintain a Collection of Products)

This question assume you have created the **Product** class for Question 1. Define class **Catalog** which has attribute **productList** to maintain a collection of products.

It has functions

- To display the products in the list (for inspection)
- To search for a product by its product id. This method takes an id and returns the Product object with that id. If the product is not in the list, it should return **None**.
- To add a product to the list. The method must ensure that the id's of the products in the list are unique.
- To change the price of a product. This method takes two parameters: one for the id and one for the new price.
- To delete a product with a specified id.
- To retrieve the list of products whose names contains a specified search word (a string). If there is no such product, the method returns an empty list.

Test your class thoroughly.

5. Menu-Driven Program

Write a program named **CatalogMenu**, which provides the user with the following options:

1. To add a product
2. To change the price of a product
3. To delete a product with a specified id
4. To display all the products

Q. To quit

The menu program has a **Catalog** object. Before presenting the menu to the user, the program reads the details of the products and add them to product list of the Catalog. To add a product, the program gets product's id, name and price from the user and then apply method **addProduct** to the **Catalog** object. Similarly, for other option 2 and 3.

7. Fibonacci Problem – The Object-Oriented Way

The Fibonacci problem can be formulated in terms of a colony of rabbits. This colony starts with one pair. The rabbits are mature when they are two months old. A pair of mature rabbits gives birth to another pair every month. At any time, the colony has a number of new-born pairs, of one-month old pairs, and of mature pairs.

At the beginning of the first month, the colony has

- 1 new-born pair
- 0 one-month old pair
- 0 mature pair
- and the total population of 1 pair

At the beginning of month 2, it has

- 0 new-born pair
- 1 one-month old pair
- 0 mature pair
- and the total population of 1 pair

At the beginning of month 3, it has

- 1 new-born pair
- 0 one-month old pair
- 1 mature pair
- and the total population of 2 pair

Define a **RabbitColony** class, which has three attributes **newBorn** for number of new-born pairs, **oneMonth** for number of one-month pairs, and **mature** for number of mature pairs. Test your class. Then use it to print out the first 12 terms of the Fibonacci sequence: 1 1 2 3 5 8 . . .

8. **Magic Functions**

Below is the class **BankAccount** presented in the lectures. Python allows us to add special behaviours to the class using what is known as the magic function. We can use magic function to provides a string representation of object for the **BankAccount** class as follows:

```
class BankAccount:
    def __init__(self, accountNr, customerName, balance = 0):
        self.accountNr = accountNr
        self.customerName = customerName
        self.balance = balance

    def __repr__(self):
        return "BankAccount<nr: " + str(self.accountNr) +  \
            ", name: " + str(self.customerName) +          \
            ", balance: " + str(self.balance) + ">"

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        self.balance -= amount
```

Write another version of the **Product** class in Question 1. Add __repr__ and __str__ functions to provide a string representation of object

9. **Product Class - The version that prevents misuse**

Consider the class you define for Question 1. Suppose the following are assumptions for the intended usage of the class:

- The product's id and name should not be changed.
- To change the price, method **setPrice** should be used.
- **id** and **name** are non-empty strings
- **price** is a positive float value.
- **qty** is a non-negative integer.

Using private attributes, define a new version of the class that cannot be misused. Any "illegal" attempt with the constructor or set methods will print a warning message.

Lab07 – Inheritance

1. **Review Questions**

a. How to declare that a class B is a subclass of class A?

b. How to call a method in the superclass?

c. Suppose the superclass A has attribute x and a constructor to initialize x. Suppose the subclass B has additional attribute y and a constructor to initialize x and y. Define the constructor for class B.

d. What is polymorphism?

e. Let **x** be a reference to an object and let **C** be a class. What is returned by

i. **type(x) is C**?

ii. **isinstance(x, C)**?

2. Book and Dictionary

Consider the class

```
class Book:

    def __init__(self, title, nrPages):
        self.title = title
        self.nrPages = nrPages

    def __repr__(self):
        return "Book<: " + "title:" + str(self.title) + \
               ", nrPages: " + str(self.nrPages) + ">"
```

Here is a basic test of the class:

```
>>>     b = Book ("The Arts of Programming", 150)
>>>     b
    Book<: title: TheArts of  Programming, nrPages:     150>
```

Define class **Dictionary** as a subclass of **Book**. The subclass has an additional attribute **nrEntries** to record the number of entries the dictionary has. Test your **Dictionary** class.

3. Book and Dictionary – Code Reuse

Let us start with this class:

```
1  class   Book:
2
3      def  __init__(self,       title,      nrPages):
4          self.title       = title
5          self.nrPages     = nrPages
6
7      def  getDetails(self):
8          return   "title:      "  + str(self.title)          + \
9                   ",  nrPages:    "  + str(self.nrPages)
10
11     def  __repr__(self):
12         return   self.__class__.__name__        + \
13                  "<"  + self.getDetails()       + ">"
```

Perform the following tests to see how the class definition works:

```
>>> b = Book("The Arts of Programming", 150)
>>> b.getDetails()
' title: The Arts of Programming, nrPages: 150'
>>> b
Book<title: The Arts of Programming, nrPages: 150>
```

In the above class,

- Method **getDetails** returns a string showing attribute names and values.
- Method __repr__ calls **getDetails** to get the attribute names and values (line 13)
- Expression self.__class__.__name__ returns the name of the class.

Next, consider this class:

```
1  class Dictionary(Book):
2
3      def __init__(self, title, nrPages, nrEntries):
4          super().__init__(title, nrPages) self.nrEntries = nrEntries
5
6      def getDetails(self):
7          return \
8              super().getDetails()          + \
9              ", nrEntries:      " + str(self.nrEntries)
10
11     # Inherit __repr__ from superclass
12
```

In the above definition of the subclass,

- Method **getDetails** calls the method with the same names in the superclass to get details about title and number of pages.
- We do not need to define method __repr__. The subclass inherits this method from the superclass

Perform the following tests to see how the class definition works:

```
>>> d = Dictionary("Essential English Dictionary", 200, 30000)
>>> d.getDetails()
' title: Essential English Dictionary, nrPages: 200, nrEntries:
    30000'
>>> d
Dictionary<title: Essential English Dictionary, nrPages: 200,
    nrEntries: 30000>
```

The two classes clearly demonstrate the usefulness of code reuse.

4. Classes Payment and Subclasses

a. Define class **Payment** that has an attribute **amount** (intended to be of type **float**) that stores a payment amount.

In addition to __init__ and __repr__, include method **printDetails** to print on the screen the payment details as shown in the example below:

```
PAYMENT
Amount:        400
```

b. Define class **CashPayment** as a subclass of **Payment**.

This class redefines **printDetails** to display the payment details as shown in the example below:

```
PAYMENT
Amount:        400
By cash
```

c. Define class **CreditCardPayment** as a subclass of **Payment**. This class has attribute **creditCardNr** for the credit card number.

This class redefines **printDetails** to display the payment details as shown in the example below:

```
PAYMENT
Amount:        400
By credit  card:  12345678
```

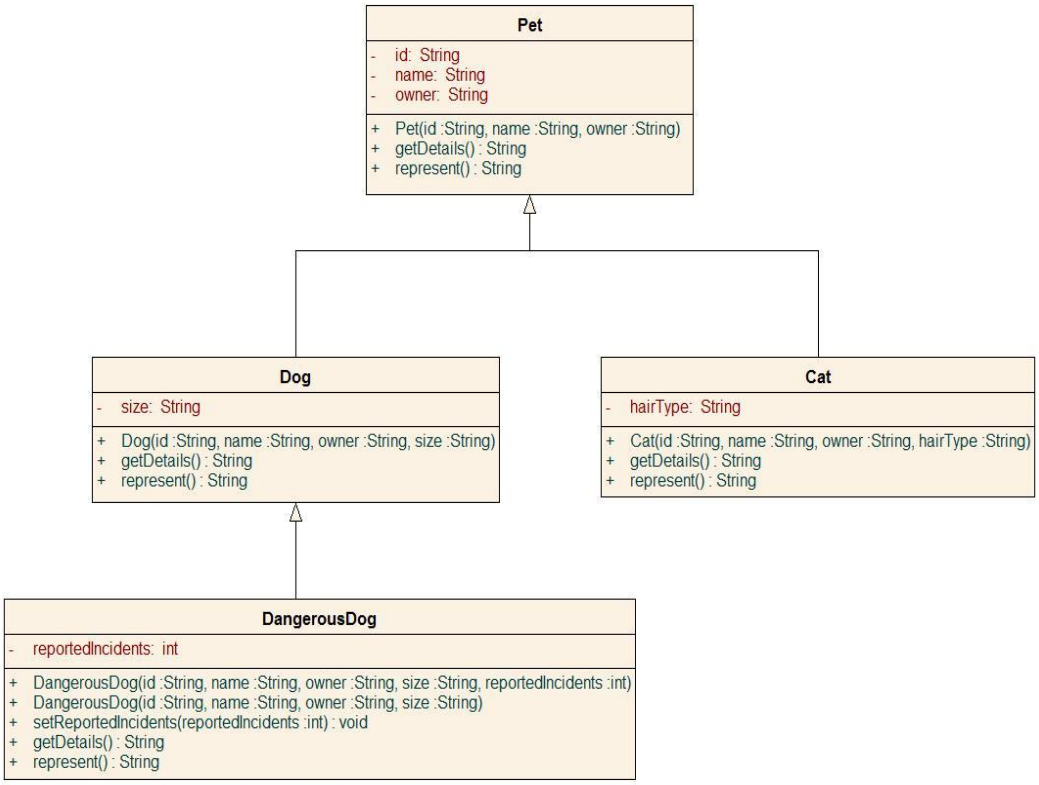
d. A list of payments is created as shown below:

```
p1 = CashPayment(100)
p2 = CreditCardPayment(200,        "AB123")
p3 = CashPayment(200)
p4 = CreditCardPayment(100,        "CD456")
payments  =     [p1, p2,  p3,   p4]
```

Write statements to calculate and display the total payment amount of all the payments by credit cards.

5. Pets, Dogs and cats

An inheritance hierarchy for pets is given below:



- The size of a dog can be “small”, “medium” or “large”.
- The hair type of a cat can be “SH” (short hair) or ‘LH” (long hair).
- When a dangerous dog’s details are entered, initially the number of reported (attack) incidents may be missing. In this case a default value of 0 is recorded.
- For each class, the **getDetails** method returns a String showing attribute names and attribute values, the attribute names and values are separated by colons, and the attributes are separated by commas.

A. Implement class Pet. Test it with a test program.

B. Implement class Dog and test it.

C. Repeat for the two remaining classes.

D. Add statements to create a list of pets and then to display all the dogs (including the dangerous ones).

E. Add statements to display all the dogs, excluding the dangerous ones.

F. Add statements to display the total number of reported incidents involving the dangerous dogs.

6. **Polymorphism example**
   Consider the following two different classes:

```
class Cat:
    def __init__(self, name, color):
        self.name = name
        self.color = color
    def info(self):
        print(f"I am a cat. My name is {self.name}. My color is {self.color}")
    def make_sound(self):
        print("Cat_sound")
class Dog:
    def __init__(self, name, color):
        self.name = name
        self.color = color
    def info(self):
        print(f"I am a dog. My name is {self.name}. My color is {self.color}")
    def make_sound(self):
        print("Dog_sound")
cat1 = Cat("Kitty", "black")
dog1 = Dog("Fluffy", "white")
```

Write common interface function using Polymorphism concept to execute **make_sound ()** method in both classes.