Getters and Setters Method

In [ ]:
```
Getters and Setters --> these methods are used to access private variables of a class outside the class by external user.
                        (These methods are instance method)
                    --> These methods are also used for Data Validations.
```

Getter Method

In [ ]:
```
Getter means access or getting the value of private variable.

Syntax of Getter Method: getting the value of private variable.

    def get_variable_name(self):
        return self.__variable
```

Setter Method

In [ ]:
```
Setter means  set the values of private variables.



Syntax of Setter Method: set the value of private variable

    def set_variable_name(self,x):
        self.__variable_name=x
```

Example of Getter and Setter Method

In [3]:
```
#demo
class Bank:
    bank_name="Indian Bank"
    def __init__(self,balance=0):
        self.__balance=balance

    #Getter Method
    def get_balance(self):
        print("Getter method called")
        return self.__balance

    #Setter Method
    def set_balance(self,x):
        print("setter method called")
        self.__balance=x

Indian_Bank = Bank()
Indian_Bank.set_balance(100)
print(Indian_Bank.get_balance())
```

```
setter method called
getter method called
100
```

Pillers of Object Oriented Programming:

In [ ]:
```
There are Four Pillers of Object oriented Programming :

    Encapsulation --> Binding the data and functions together as a single enitity(class).
                  --> Encapsulation is a mechanism of wrapping the data (variables) and code acting on the data (methods)
                      together as a single unit

    Polymorphism --> The word "polymorphism" means having many forms. In simple words, we can define polymorphism as
                     the ability of a message to be displayed in more than one form.
                 --> A real-life example of polymorphism is a person who at the same time can have different characteristics

    Inheritance --> Inheritance allows us to define a class that inherits all the methods and properties from another class.

    Abstraction --> Hiding the irrelevant thing from the user.
                --> Abstraction in python is defined as a process of handling complexity by hiding unnecessary information
                    from the user.
```

Inheritance

In [ ]:
```
Inheritance --> All the variables and methods that are available to the parent class will automcatically
                available to the child class through inheritance
            --> Parent and child relation will be implemented with the help of Inheriatnce.

Advantage of inheritance:
    Code Reuseability.
```

Terminologies related to inheritance

In [ ]:
```
Terminologies related to inheritance:

    --> Parent class/Base class/Super class --> Is that class which is being inherited
    --> child class/derived class/Sub class --> is that class which is inheriting the properties and behaviour from parent
        class
```

Example

In [6]:
```
#demo
class Traditional_Phone:
    def call(self):
        print("Calling functionality is there")
    def message(self):
        print("Message functionality is there")
    def Game(self):
        print("Gaming functionality is there")
    def Calculator(self):
        print("Calculator functionality is present")

class SmartPhone(Traditional_Phone):
    def camera(self):
        print("Camera functionality is there")
    def VideoCall(self):
        print("VideoCalling functionality is there")


x=SmartPhone()
x.call()
x.message()
x.Game()
x.Calculator()
x.camera()
x.VideoCall()
print("-------------------------------------")
y=Traditional_Phone()
y.call()
y.message()
y.Game()
y.Calculator()
```

```
Calling functionality is there
Message functionality is there
Gaming functionality is there
Calculator functionality is present
Camera functionality is there
VideoCalling functionality is there
-------------------------------------
Calling functionality is there
Message functionality is there
Gaming functionality is there
Calculator functionality is present
```

Types of Inheriatance:

In [ ]:
```
Multiple Types of Inheriatance:
    Single Level Inheriatance
    Multi Level Inheriatance
    Hierarchical Inheriatance
    Multiple Interitence
```

Single Level Inheritance

In [ ]:
```
Single Inheritance --> The concept of inheriting the properties and behaviour from
                       one parent class and one child class is known as single inheriatnce.

Example:
    Father and You
```

Implementation of Single Level Inheritance

In [7]:
```
class parent:
    def m1(self):
        print("parent class m1 method")
class child(parent):
    def m2(self):
        print("Child class m2 method")
c=child()
c.m1()
c.m2()
x=parent()
x.m1()
```

```
parent class m1 method
Child class m2 method
parent class m1 method
```

Multi level Inheritance

In [ ]:
```
Multilevel Inheriatnce --> If you want inheriting the properties of multiple class
                           into a single class that type of inheritance is knownas multilevel inheriatnce

Example:
    GrandFather  ---> Father ----> You
```

Implementation of Multilevel Inheritance

In [8]:
```
class GrandFather:
    def property(self):
        print("Purchased a property")

    def Farmhouse(self):
        print("FarmHouse")

    def gold(self):
        print("Gold")

class Father(GrandFather):
    def Car(self):
        print("Car")

    def Share(self):
        print("Share")

class You(Father):
    def Bike(self):
        print("Bike")
    def Laptop(self):
        print("Laptop")


x=You()
x.Bike()
x.Laptop()
x.Car()
x.Share()
x.FarmHouse()
```

```
Bike
Laptop
Car
Share
FarmHouse
```

Hierachical Inheriatance

In [ ]:
```
Hierachical Inheriatance: if we want to inherit properties and behaviour of parent class to multiple child class
                          then such type of inheriatnce is known as hierachical inheriatance.

Example --> Traditional_Phone --> Smartphone
            Tradtional Phone  --> smartWatch
```

Implementation of Hierachical Inheriatance

In [9]:
```
class Traditional_Phone:
    def call(self):
        print("Calling Funcctionality is there")
    def message(self):
        print("Messging Funcctionality is there")
    def Calculator(self):
        print("Calculator Funcctionality is there")
class SmartPhone(Traditional_Phone):
    def camera(self):
        print("Camera Functionality is there")
    def music(self):
        print("Music Funcctionality is there")
    def VideoCalling(self):
        print("VideoCalling Funcctionality is there")
    def radio(self):
        print("Radio Funcctionality is there")
class SmartWatch(Traditional_Phone):
    def calorie(self):
        print("Calorie Funcctionality is there")
    def step_counting(self):
        print("Step Counting Funcctionality is there")
    def heart_beat(self):
        print("HeartBeat Funcctionality is there")

x=SmartWatch()
x.calorie()
x.heart_beat()
x.call()
x.message()
```

```
Calorie Funcctionality is there
HeartBeat Funcctionality is there
Calling Funcctionality is there
Messging Funcctionality is there
```

Multiple Inheritance

In [ ]:
```
Multiple Inheritance --> if you want to inherit the properties and behaviour of multiple parent classes
                         into one single child class then that type of inheriatnce is known as multiple inheriatnce.


Example:  Traditional_Phone                 Ipod
                          SmartPhone
```

Implementation of Multiple Inheritance

In [10]:
```
class Traditional_Phone:
    def call(self):
        print("Calling Funcctionality is there")
    def message(self):
        print("Messging Funcctionality is there")
    def Calculator(self):
        print("Calculator Funcctionality is there")


class Ipod:
    def music(self):
        print("Music Funcctionality is there")

class SmartPhone(Traditional_Phone,Ipod):
    def camera(self):
        print("Camera Functionality is there")
    def music(self):
        print("Music Funcctionality is there")
    def VideoCalling(self):
        print("VideoCalling Funcctionality is there")
    def radio(self):
        print("Radio Funcctionality is there")
x=SmartPhone()
x.camera()
x.music()
x.call()
```

```
Camera Functionality is there
Music Funcctionality is there
Calling Funcctionality is there
```

In [11]:
```
class Traditional_Phone:
    def call(self):
        print("Calling Funcctionality is there")
    def message(self):
        print("Messging Funcctionality is there")
    def Calculator(self):
        print("Calculator Funcctionality is there")
    def music(self):
        print("Music Funcctionality - Traditional is there")


class Ipod:
    def music(self):
        print("Music Funcctionality IPod is there")

class Iphone:
    def security(self):
        print("More Secure")

class SmartPhone(Ipod,Traditional_Phone,Iphone):
    def camera(self):
        print("Camera Functionality is there")
    def VideoCalling(self):
        print("VideoCalling Funcctionality is there")
    def radio(self):
        print("Radio Funcctionality is there")
y=SmartPhone()
y.music()
y.security()
```

```
Music Funcctionality IPod is there
More Secure
```

Note

In [ ]:
```
In Case of inheritance if we have same method in parent class as well in child class and
we are creating child class object then PVM will check that method first in child class then first parent class and
then second parent class if the method is not present in all parent and child class then we will get an error.
```