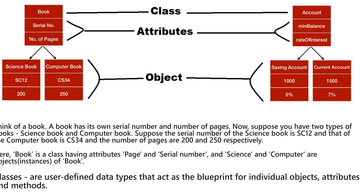


Think of a book. A book has its own serial number and number of pages. Now, suppose you have two types of books - Science book and Computer book. Suppose the serial number of the Science book is SC12 and that of the Computer book is CS34 and the number of pages are 200 and 250 respectively.

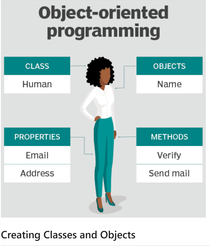
Here, 'Book' is a class having attributes 'Page' and 'Serial number', and 'Science' and 'Computer' are objects(instances) of 'Book'.

Classes - are user-defined data types that act as the blueprint for individual objects, attributes and methods.

Objects - are instances of a class

Methods - are functions that are defined inside a class that describe the behaviors of an object.

Attributes - are defined in the class template and represent the state of an object. Objects will have data stored in the attributes field.



Creating Classes and Objects

In [1]:
```
class Phone: #the 1st letter of the class is always capits - Phone and not phhone.

    def make_call(self): # Method-1
        print("Calling")

    def play_game(self): # Method -2
        print("Playing")
```

Creating/instantiating and object p1

In [2]:
```
p1 = Phone()
```

Invoking methods through objects

In [4]:
```
p1.make_call()
```

```
Calling
```

In [5]:
```
p1.play_game()
```

```
Playing
```

Adding parameter to the class

In [22]:
```
class Phone:
    def set_colour(self,colour):
        self.colour=colour

    def set_cost(self,cost):
        self.cost=cost

    def show_colour(self):
        return self.colour

    def show_cost(self):
        return self.cost

    def make_call(self):
        print("calling")

    def play_game(self):
        print("Playing")
```

In [23]:
```
p2 = Phone()
```

In [24]:
```
p2.set_colour("Blue")
p2.set_cost(2000)
```

In [25]:
```
p2.show_colour()
```

```
Blue
```

In [26]:
```
p2.show_cost()
```

```
2000
```

In [27]:
```
p2.make_call()
```

```
calling
```

In [28]:
```
p2.play_game()
```

```
Playing
```

Creating a class with constructor

In [35]:
```
class Employee:
    def __init__(self,name,age,salary,gender):
        self.name = name
        self.age = age
        self.salary = salary
        self.gender = gender

    def employee_details(self):
        print("Name of the employee is ", self.name)
        print("Age of the employee is ", self.age)
        print("Salary of the employee is ", self.salary)
        print("Gender of the employee is ", self.gender)
```

In [36]:
```
# Now instantiate the object
```

In [38]:
```
e1 = Employee('Shila',24,25000,'Female')
e1.employee_details()
```

```
Name of the employee is  Shila
Age of the employee is  24
Salary of the employee is  25000
Gender of the employee is  Female
```

OOP Inheriatnce

With inheritance, one class can derive (inherit) the properties of another class.

In [39]:
```
class Vehicle:                          #This is the base class
    def __init__(self,milage,cost):
        self.milage = milage
        self.cost = cost
    def show_details(self):
        print("This is a vehicle")
        print("Milage of the vehicle is ", self.milage)
        print("The cost of the vehicle is", self.cost)
```

In [40]:
```
v1 = Vehicle(20,20000) #Instantiating the object for base calss
v1.show_details()
```

```
This is a vehicle
Milage of the vehicle is  20
The cost of the vehicle is 20000
```

In [41]:
```
class Car(Vehicle): #Creating a child class enheriting parent class's(super class's) features.
    def show_car(self):
        print("Toyota Corola")
```

In [44]:
```
c1 = Car(25,25000) #Instatiating the object for child class
c1.show_details()
c1.show_car()  #Invoking the child method
```

```
This is a vehicle
Milage of the vehicle is  25
The cost of the vehicle is 25000
Toyota Corola
```

over-riding init method

adding extra attributes in child class

In [43]:
```
class Car(Vehicle):
    def __init__(self,milage,cost,tyres,hp):#Here tyres and hp are added in child class Car
        super().__init__(milage,cost)
        self.tyres = tyres
        self.hp = hp
    def show_car_details(self):
        print("Toyota Corola")
        print("Number of tyres = ", self.tyres)
        print("Horse power of the car is = ", self.hp)
```

In [47]:
```
    c1 = Car(25,25000,4,999)
    c1.show_details() #invoking method from parent class
```

```
This is a vehicle
Milage of the vehicle is  25
The cost of the vehicle is 25000
```

In [49]:
```
c1.show_car_details() #invoking method from child class
```

```
Toyota Corola
Number of tyres =  4
Horse power of the car is =  999
```

Types of Inheritnace

1.Single Inheritance (child class inherits from one parent class, discussed above)

2.Multiple Inheritance (Child class inherits from more than one parent class)

In [75]:
```
class Parent1(): # Parent class 1
    def assign_string_one(self,str1):
        self.str1 = str1
    def show_string_one(self):
        return self.str1

class Parent2(): # Parent class 2
    def assign_string_two(self,str2):
        self.str2 = str2
    def show_string_two(self):
        return self.str2

class Derived (Parent1,Parent2): # Child class inheriting from both parents
    def assign_string_three(self,str3):
        self.str3 = str3
    def show_string_three(self):
        return self.str3
```

In [76]:
```
#Instantiating object (d1) of child class
d1 = Derived()
d1.assign_string_one("One")
d1.assign_string_two("Two")
d1.assign_string_three("Three")
```

In [78]:
```
#Invoke method of parent 1
d1.show_string_one()
```

Out[78]:
```
'One'
```

In [80]:
```
#Invoke method of parent 2
d1.show_string_two()
```

Out[80]:
```
'Two'
```

In [81]:
```
#Invoke method of child
d1.show_string_three()
```

Out[81]:
```
'Three'
```

3.Multi level Inheritance (Parent to child, child to grand child relationship)

In [83]:
```
#Parent class
class Parent:
    def get_name(self,name):
        self.name =  name
    def show_name(self):
        return self.name
```

In [90]:
```
#Child class
class Child(Parent):
    def get_age(self,age):
        self.age = age
    def show_age(self):
        return self.age
```

In [93]:
```
#Grand child class
class Grand_Child(Child):
    def get_gender(self,gender):
        self.gender = gender
    def show_gender(self):
        return self.gender
```

In [94]:
```
#Instantiating object go of class grand_child
go = Grand_child()
go.get_name("Shila")
go.get_age(24)
go.get_gender("Female")
```

In [95]:
```
#Invoke method of of parent
go.show_name()
```

Out[95]:
```
'Shila'
```

In [98]:
```
#Invoke Method of child
go.show_age()
```

Out[98]:
```
24
```

In [99]:
```
#Invoke Method of grand child
go.show_gender()
```

Out[99]:
```
'Female'
```

4.Hybrid Inheritance

In [ ]:
```

```