Imam Abdulrahman Bin Faisal University
Deanship of Preparatory Year and Supporting Studies

Basics of Programming through Python

Object Oriented Programming

Introduction to Programming

COMP102

Term 3-2022-2023

Learning outcomes

- **Describe the difference** between structural programming and object-oriented programming.
- **Justify** the philosophy of object-oriented design and the concepts of inheritance .
- **Design** and implement simple programs in an object-oriented programming language.

Introduction

- Two basic programming paradigms:
  - Procedural
    - Organizing programs around functions or blocks of statements which manipulate data.
  - Object-Oriented
    - combining data and functionality and wrap it inside what is called an object.

Object Oriented Programming is a way of computer programming using the idea of “objects” to represents data and methods. It is also, an approach used for creating neat and reusable code instead of a redundant one.

Main difference between Object-Oriented and Procedural Oriented Programming

Object-Oriented Programming (OOP)	Procedural-Oriented Programming (Pop)
It is a bottom-up approach	It is a top-down approach
Program is divided into objects	Program is divided into functions
Makes use of Access modifiers ‘public’, private’, protected’	Doesn’t use Access modifiers
It is more secure	It is less secure
Object can move freely within member functions	Data can move freely from function to function within programs
It supports inheritance	It does not support inheritance

Object-Oriented Framework

- Classes and objects are the two main aspects of object-oriented programming.
- A **class** creates a new *type*.
- Where **objects** are *instances* of the class.
- Objects can store data using ordinary variables that *belong* to the object.
- Objects can have functionality by using functions that *belong* to the class. Such functions are called methods.
- This terminology is important because it helps us to differentiate between a function which is separate by itself and a **method** which belongs to an object.

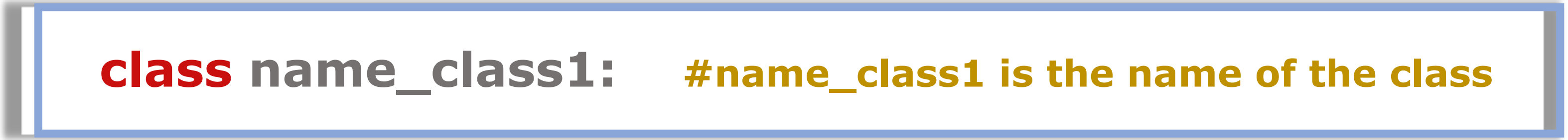
General OOP Rules

1. Everything around you is an object
2. Each object contains properties (attributes) and functions (actions or methods)
3. Object is <u>an instance of class</u>

What are Classes and Objects?

- A class is a collection of objects, or you can say it is a blueprint of objects defining the common attributes and behavior.

Class is defined under a “Class” Keyword.



```
class name_class1:    #name_class1 is the name of the class
```

The Attributes and methods of the class are listed in an indented block.

New Terminology to learn

- Class
- Object
- Instance
- Attributes
- Methods
- Inheritance
- **SELF PARAMETER**

Class and attributes(properties)

Class_name	**Attributes**
Table	WIDTH,HEIGHT
student	Name,id,age,…
Cat	Color, age,type
car	Color,brand,yearManf,speed
rectangle	Length, width

Every class you write in Python has two basic features: attributes and methods.

Attributes are the individual things that differentiate one object from another. They determine the appearance, state, or other qualities of that object. They belongs to the class and an object belongs to a class.

Classes Copy Objects

A Python program consists of one or more classes

Example of class:

class Student:

description of student goes here

(Attributes / Methods)...

Attributes	*Methods*
Sname	updateInformation
Sbirthdate	CalculateGrade
Saddress	DisplayInformation

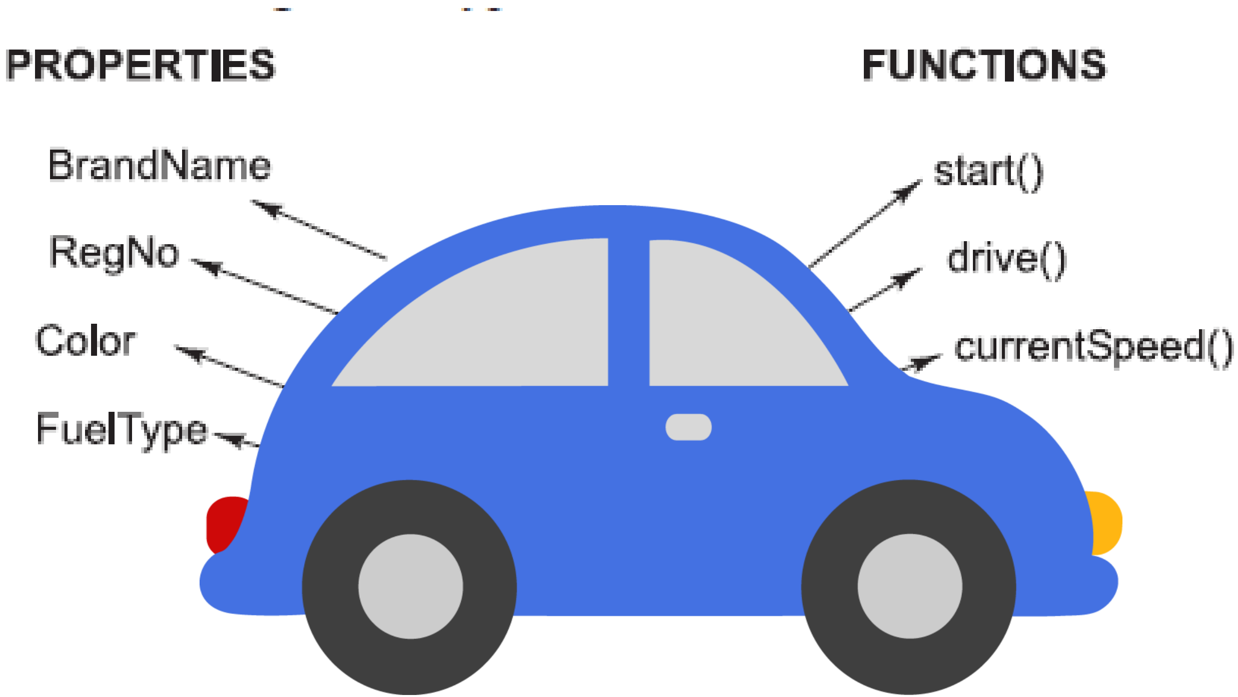
A class is an abstract description of objects
An object is an instance of a class

Some objects of Student class:

Attributes	*Methods*
Mohamed	updateInformation
2-10-2003	CalculateGrade
Dammam	DisplayInformation

Attributes	*Methods*
Ahmed	updateInformation
3-5-2002	CalculateGrade
Khobar	DisplayInformation

Example of class (properties and methods)



PROPERTIES

BrandName

RegNo

Color

FuelType

FUNCTIONS

start()

drive()

currentSpeed()

Create a Class

Problem:

Create a class named MyClass, with a property named x:

Solution:

```
class MyClass:
  x = 2023
```

Create an Object

Now we can use the class named **MyClass** to create objects:

Problem:

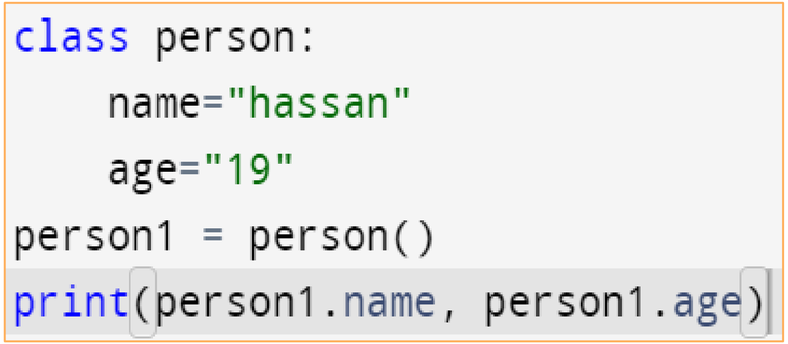
Create an object named p1, and print the value of x:

Solution:

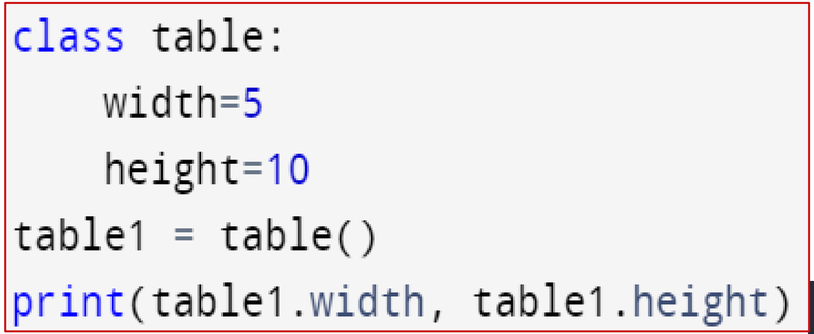
```
class MyClass:
    x=2023
p1 = MyClass()
print(p1.x)
```

Create classes and objects: Example

- Create class named Person with two attributes name and age?
- Create class named table with two attributes width and height?
- Create objects for each class?



```
class person:
    name="hassan"
    age="19"
person1 = person()
print(person1.name, person1.age)
```



```
class table:
    width=5
    height=10
table1 = table()
print(table1.width, table1.height)
```

Exercise 1

Problem:

Write a Python class named Student with two attributes student_id, student_name:

Create an object named **s1**, and print the id and the name of the student

Solution:

```
class Student:
    student_id='22000345'
    student_name = 'Fatimah’
#main
s1 = Student()
print(s1.student_id)
print(s1.student_name)
```

Create object(s)

How to create many objects that belong to the same CLASS but with different values??

Example:

The objects Person1,Person 2,Person 3 are instances of the class PERSON???

The __init__() Function

- The previous examples are classes and objects in their simplest form and are not really useful in real life applications.
- To understand the meaning of classes we have to understand the built-in function __init__() function.
- All classes have a function called __init__(), which is always executed when the class is being initiated.
- Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created.

__init __ () Function

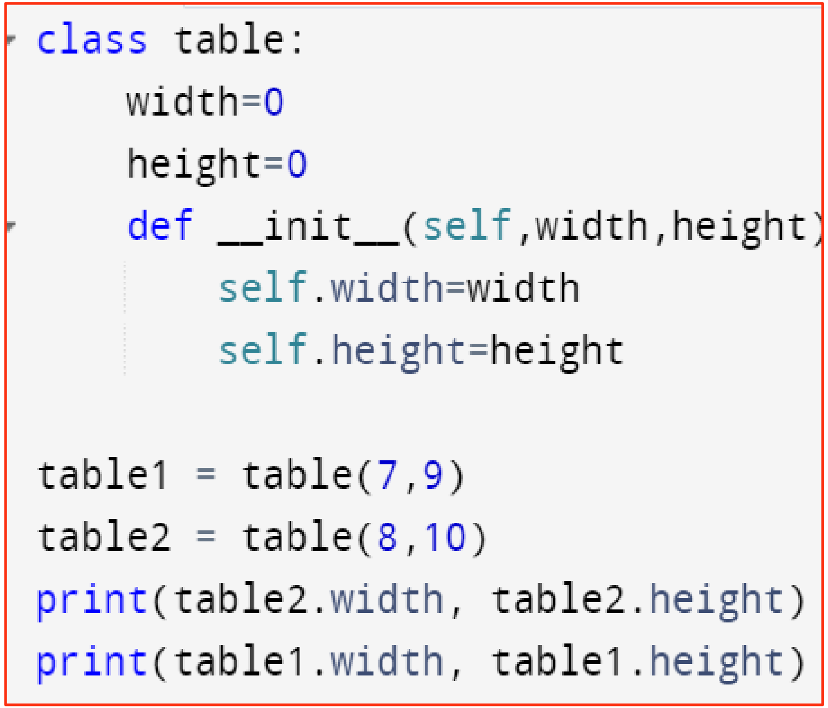
- All classes have a task named __init __ (), which always comes when the class starts.
- Using __init ()__ to set the value of the item object, or other activities that need to be done when the product is created.

__init__(parameters) is the special method that initializes an individual object. This method runs automatically each time an object of a class is created.

__init__ serves as a constructor for the class. Usually does some initialization work.

Example of __init__()

- When you define __init__() in a class definition, its first parameter should be self.
- The self parameter refers to the individual object itself. It is used to fetch or set attributes of the particular instance



```
class table:
    width=0
    height=0
    def __init__(self,width,height)
        self.width=width
        self.height=height

table1 = table(7,9)
table2 = table(8,10)
print(table2.width, table2.height)
print(table1.width, table1.height)
```

Object Methods

```
def name(self, parameter, ..., parameter):
        statements
```

- The **self** parameter is a reference to the current instance of the class and is used to access variables that belongs to the class.
- **self** *must* be the first parameter to any object method represents the "implicit parameter"
- It does not have to be named **self** , you can call it whatever you like, but it must be the first parameter of any function in the class.

Exercise 2

Problem:

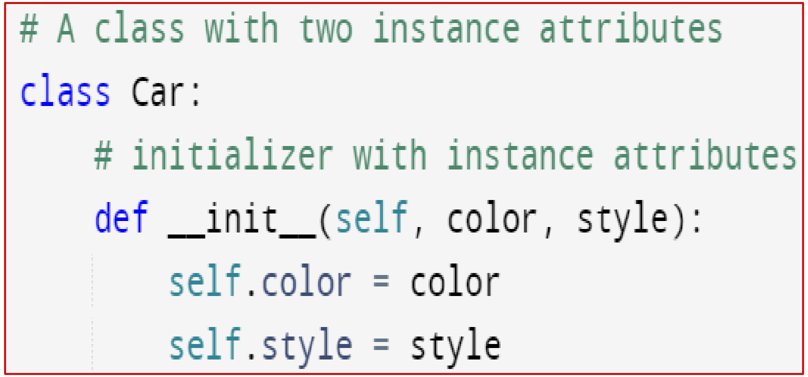
Create a class named Person, use the __init__() function to assign values for name and age.

Solution:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person(“Ahmad", 36)

print(p1.name)
print(p1.age)
```

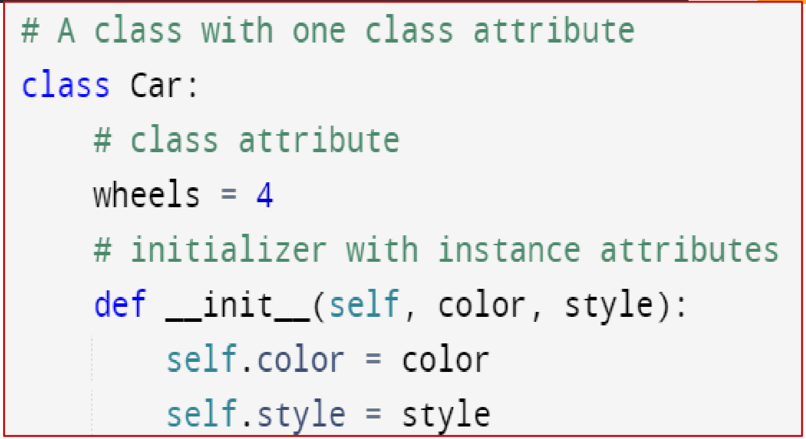
Note: The __init__() function is called automatically every time the class is being used to create a new object.

Difference between Class attribute/Instance(Object) Attribute



```python
# A class with two instance attributes
class Car:
    # initializer with instance attributes
    def __init__(self, color, style):
        self.color = color
        self.style = style
```

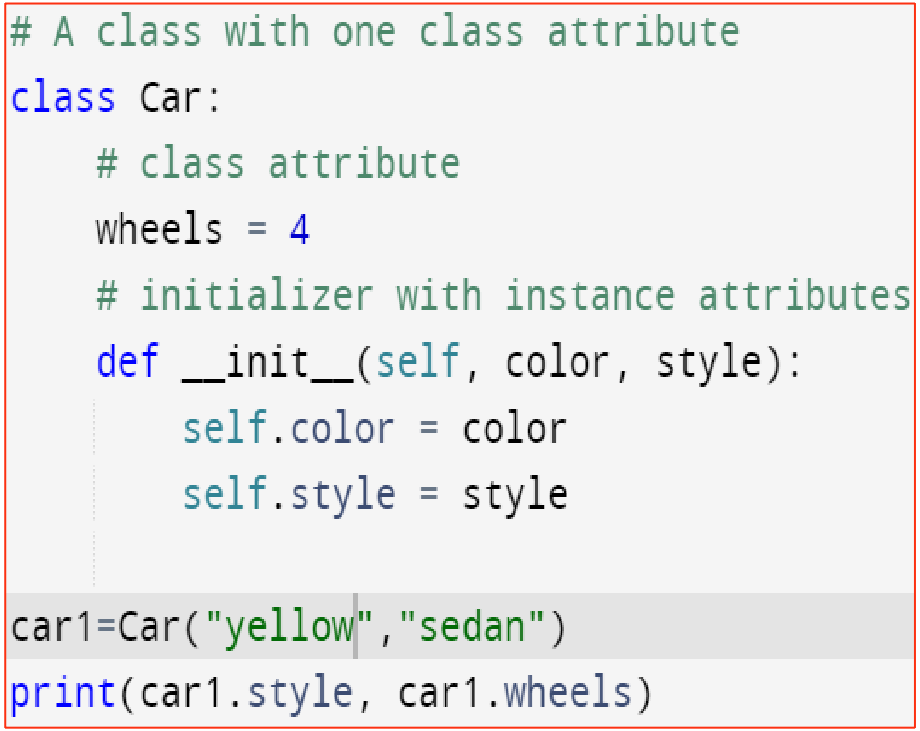
- **The instance attribute is a variable that is unique to each object (instance). Any changes made to the variable don’t reflect in other objects of that class.**
- **In the case of our Car class, each car has a specific color and style.**



```python
# A class with one class attribute
class Car:
    # class attribute
    wheels = 4
    # initializer with instance attributes
    def __init__(self, color, style):
        self.color = color
        self.style = style
```

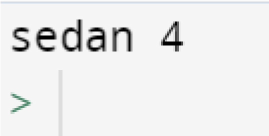
- **The class attribute is a variable that is same for all objects. Any changes made to that variable will reflect in all other objects.**
- **In the case of our Car class, each car has 4 wheels.**

Class attributes: Example



```python
# A class with one class attribute
class Car:
    # class attribute
    wheels = 4
    # initializer with instance attributes
    def __init__(self, color, style):
        self.color = color
        self.style = style

car1=Car("yellow","sedan")
print(car1.style, car1.wheels)
```

- *We only define color and style attributes. The wheels is 4 for all instance by default.*
- *Expected output:*



```
sedan 4
>
```

Exercise

Problem:

Write a Python program to create an instance of a specified class Student with **instance attributes**:

- ✓ Student_id
- ✓ Student_name
- ✓ Class_name

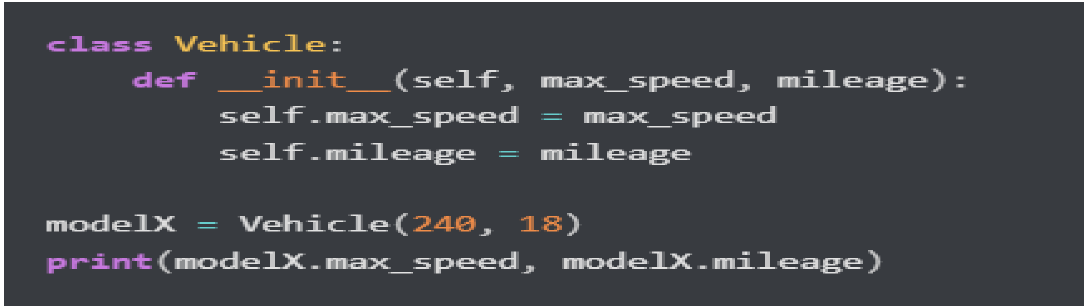
Solution:

```
class Student:
  def __init__(self, student_id, student_name, 
class_name):
     self.student_id = student_id
     self.student_name = student_name
     self.class_name = class_name
student = Student(‘22000345’, ‘FATIMAH', ‘SF')
print(student.student_id )
print(student.class_name )
```

Exercise: create a vehicle class

Problem : Write a Python program to create a Vehicle class with max_speed and mileage instance attributes.

Solution:



```
class Vehicle:
    def __init__(self, max_speed, mileage):
        self.max_speed = max_speed
        self.mileage = mileage

modelX = Vehicle(240, 18)
print(modelX.max_speed, modelX.mileage)
```

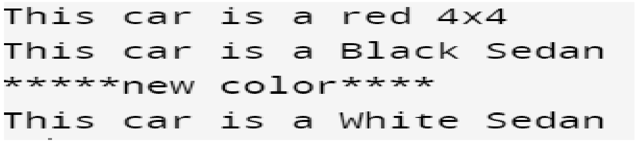
Methods and Classes

- Performing a task in a program requires a method.
- In Python, we create a program unit called a class to house the set of methods that perform the class's tasks.
- An object is referred to as an instance of its class.
- Reuse of existing classes when building new classes and programs saves time and effort.
- Reuse also helps you to build more reliable and effective systems, because existing classes and components have extensive *testing*, *debugging* and *performance.*

Try and check

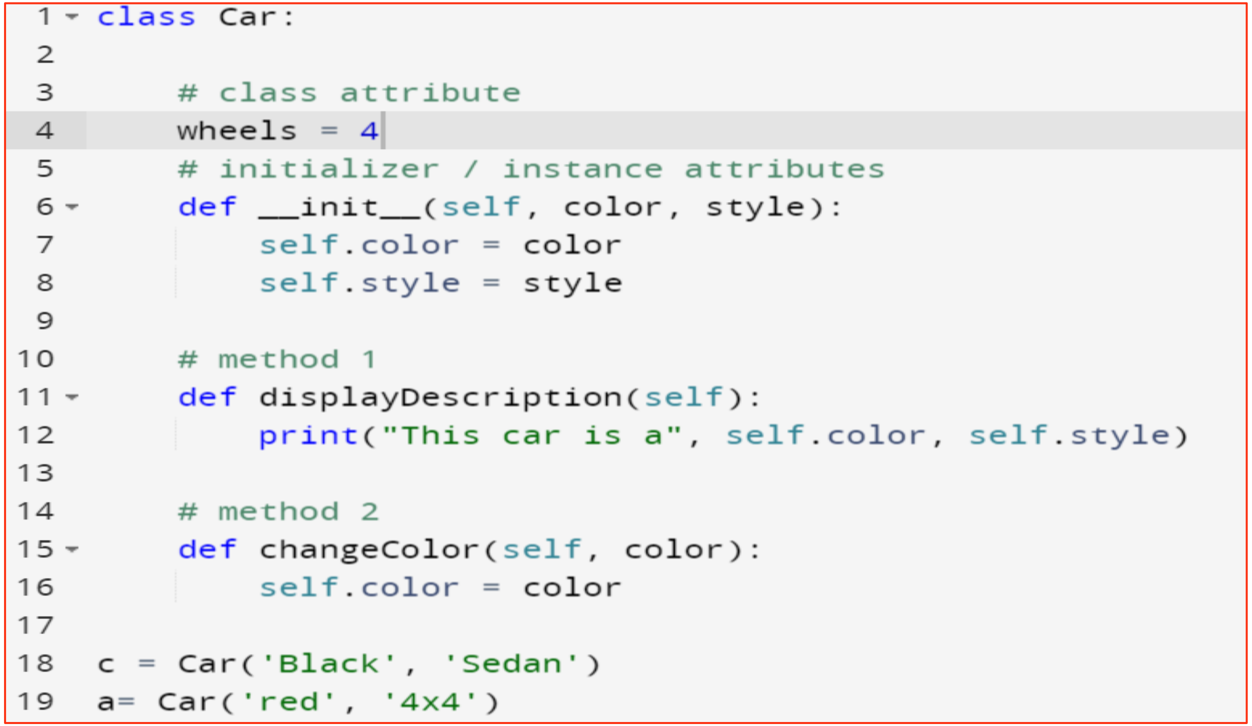
- Create class car with instance attributes: color and style and class attribute wheel
- Create method displayDescription() to print color and style of the car
- Create method changeColor() to set new color of a car

- Create 2 objects from the class car :car1 and car2
- Display the description of the 2 objects
- Change the color of the second object to white
- Display the description of the second objects
- Example of output:



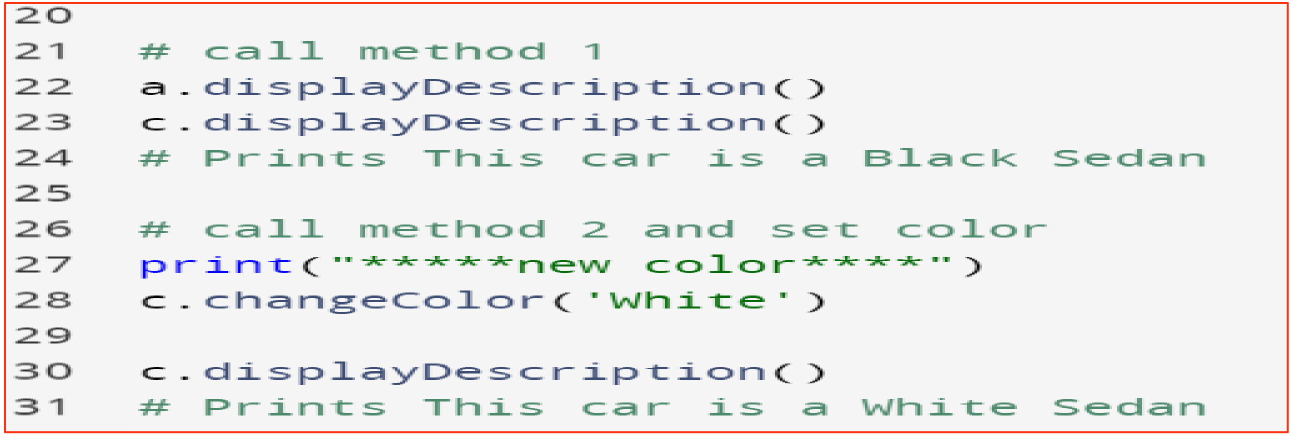
```
This car is a red 4x4
This car is a Black Sedan
*****new color****
This car is a White Sedan
```

Solution (1/2)



```python
class Car:

    # class attribute
    wheels = 4
    # initializer / instance attributes
    def __init__(self, color, style):
        self.color = color
        self.style = style

    # method 1
    def displayDescription(self):
        print("This car is a", self.color, self.style)

    # method 2
    def changeColor(self, color):
        self.color = color

c = Car('Black', 'Sedan')
a= Car('red', '4x4')
```

Solution (2/2)



```python

# call method 1
a.displayDescription()
c.displayDescription()
# Prints This car is a Black Sedan

# call method 2 and set color
print("******new color*****")
c.changeColor('White')

c.displayDescription()
# Prints This car is a White Sedan
```

Exercise: methods

Problem:

Add to the class person a method that prints a greeting, and execute it on the p1 object:

Solution:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " +
self.name)

p1 = Person(“Ahmad", 36)
p1.myfunc()
```

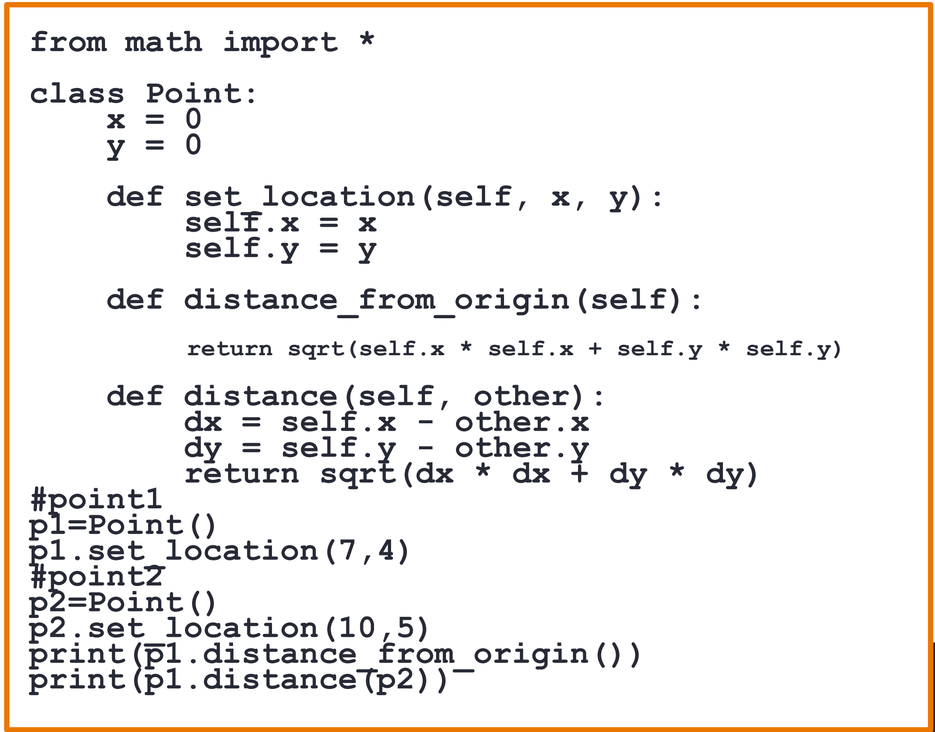
Exercise: class point

Problem:

Create class point with 2 attributes x and y, and create the following methods: **distance(), set_location(),**

and distance_from_origin()

Create two points **p1 and p2** . Call all the methods and print the location, distance between 2 points and distance from the origin.



```
from math import *

class Point:
    x = 0
    y = 0

    def set_location(self, x, y):
        self.x = x
        self.y = y

    def distance_from_origin(self):

        return sqrt(self.x * self.x + self.y * self.y)

    def distance(self, other):
        dx = self.x - other.x
        dy = self.y - other.y
        return sqrt(dx * dx + dy * dy)
#point1
p1=Point()
p1.set_location(7,4)
#point2
p2=Point()
p2.set_location(10,5)
print(p1.distance_from_origin())
print(p1.distance(p2))
```

Exercise: Display()

Problem:

Write a Python class named Student with two attributes student_id, student_name. Create a function to display the all attributes and their values in Student class.

Create an object S1.

Solution:

```
class Student:
    student_id ="220020202"
    student_name ="fatimah"
    def display(self):
          print(self.student_id,"\n",
          self.student_name)
#main
S1=Student()
S1.display()
```

Exercise: Create a Rectangle class

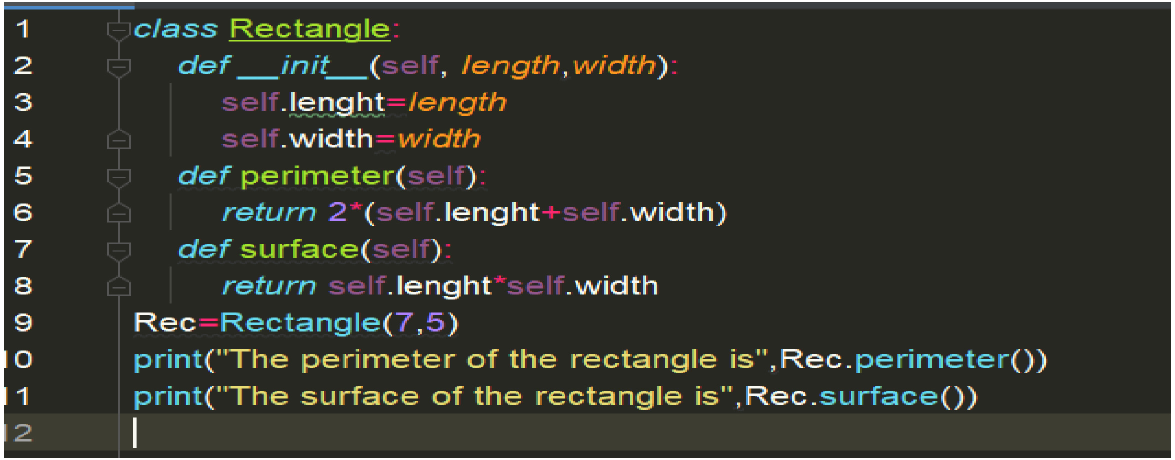
Problem

- ❑ Create a Rectangle class with 2 attributes(length , width) and 2 methods Perimeter and surface.

Note:

- ❑ **Perimeter() method** is used to calculate the perimeter of the rectangle.
- ❑ **Surface() method** is used to calculate the surface of the rectangle.

Solution



```
class Rectangle:
    def __init__(self, length,width):
        self.lenght=length
        self.width=width
    def perimeter(self):
        return 2*(self.lenght+self.width)
    def surface(self):
        return self.lenght*self.width
Rec=Rectangle(7,5)
print("The perimeter of the rectangle is",Rec.perimeter())
print("The surface of the rectangle is",Rec.surface())
```

Try and check

Problem

1-Create a Python class named **BankAccount** which represents a bank account, having the following attributes: **accountnumber , name of the account owner , balance.**

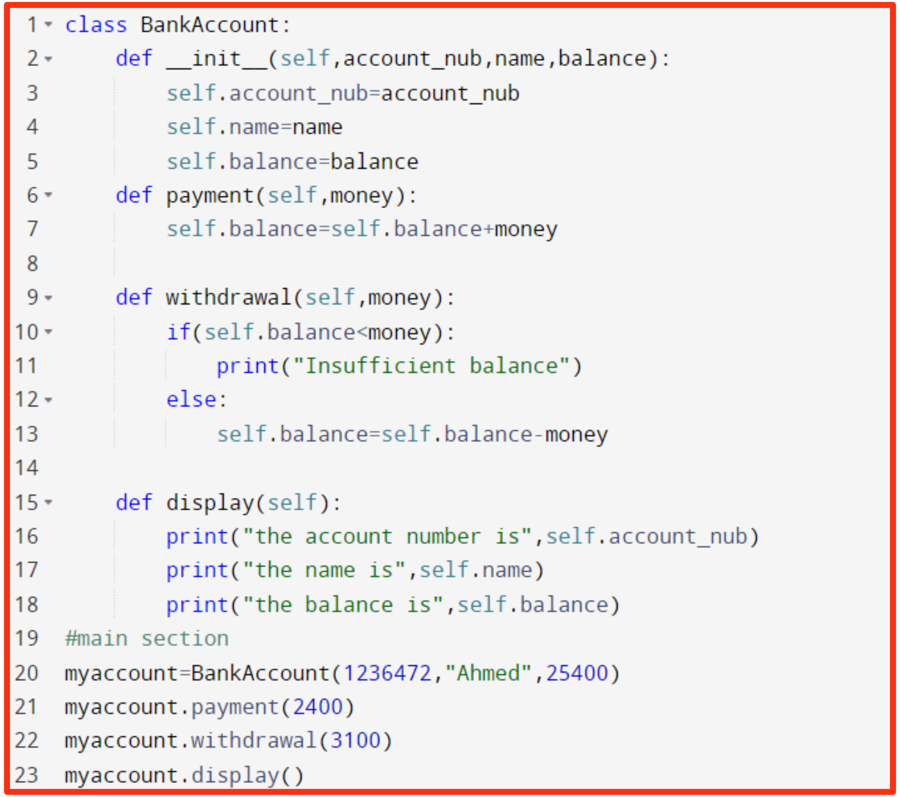
2-Create a constructor having as parameters: **accountnumber, name, balance**

3-Write a **Payment() method** that handles the payments.

4-Write a **Withdrawal() method** that handles withdrawals.

5-Write a **display() method** to display the account details

Solution



```python
class BankAccount:
    def __init__(self,account_nub,name,balance):
        self.account_nub=account_nub
        self.name=name
        self.balance=balance
    def payment(self,money):
        self.balance=self.balance+money

    def withdrawal(self,money):
        if(self.balance<money):
            print("Insufficient balance")
        else:
            self.balance=self.balance-money

    def display(self):
        print("the account number is",self.account_nub)
        print("the name is",self.name)
        print("the balance is",self.balance)
#main section
myaccount=BankAccount(1236472,"Ahmed",25400)
myaccount.payment(2400)
myaccount.withdrawal(3100)
myaccount.display()
```

Object-Oriented programming methodologies

Inheritance

Inheritance

- Ever heard of this dialogue from relatives “you look exactly like your father/mother” the reason behind this is called ‘inheritance’.
- From the Programming aspect, It generally means “inheriting or transfer of characteristics from parent to child class without any modification”.

Inheritance in Python

- One of the major benefits of object-oriented programming is **reuse** of code
- One of the ways this is achieved is through the inheritance mechanism.
- Inheritance can be best imagined as implementing a *type and subtype* relationship between classes.
- The new class is called the derived/child class and the one from which it is derived is called a parent/base class.

Create a Parent Class(step 1)

Problem:

Create a class named Person, with firstname and lastname properties, and a printname method

Solution:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

#Use the Person class to create an object,
and then execute the printname method:

x = Person("Ahmad","Hamza")
x.printname()
```

Create a Child Class (step 2)

To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class:

```
class child(parent:
```

Problem:

Create a class named Student, which will inherit the properties and methods from the Person class

Solution:

```
class Student(Person):
  pass
```

Note: Use the pass keyword when you do not want to add any other properties or methods to the class.

Create an object (step 3)

Problem:

Use the Student class to create an **object**, and then execute the **printname** method (inherited from the parent class person)

Solution:

```
x = Student("Mustapha", "Ahmed")
x.printname()
```

Add the __init__() Function in the child class

- We have created **a child class** that inherits all the properties and methods from its parent.
- We want to add the __init__() function to the child class (instead of the pass keyword).
- The __init__() function is called automatically every time the class is being used to create a new object.

Add the __init__() Function

Problem:

Add the __init__() function to the Student class:

Solution:

```
class Student(Person):
  def __init__(self, fname,
lname):
    #add properties etc.
```

Add the __init__() Function

- When you add the __init__() function, the child class **will no longer inherit** the parent's __init__() function.
- The child's __init__() function **overrides** the inheritance of the parent's __init__() function.

Add the __init__() Function

- To keep the inheritance of the parent's **__init__()** function, add a call to the parent's __init__() function.

Example:

```
class Student(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)
```

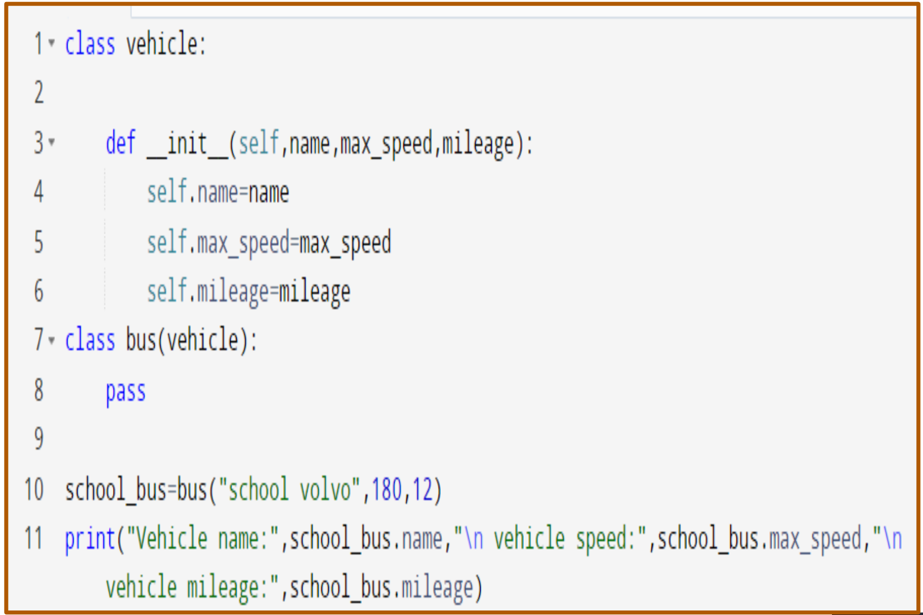
- To add a new attribute age to the child class:

```
class Student(Person):
  def __init__(self, fname, lname,age):
    Person.__init__(self, fname, lname)

    self.age=age
```

Try and check

Problem:

Create a Bus object that will inherit all the variables and methods of the parent Vehicle class and display it.

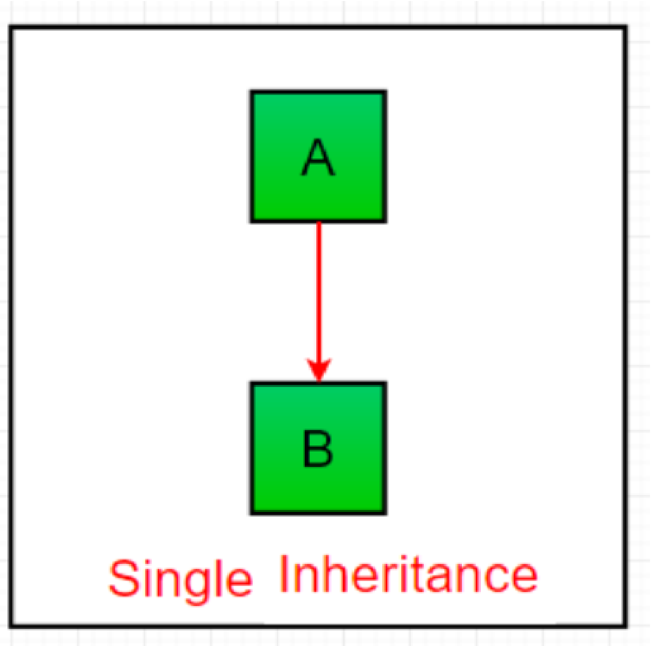


```
class vehicle:

    def __init__(self,name,max_speed,mileage):
        self.name=name
        self.max_speed=max_speed
        self.mileage=mileage
class bus(vehicle):
    pass

school_bus=bus("school volvo",180,12)
print("Vehicle name:",school_bus.name,"\n vehicle speed:",school_bus.max_speed,"\n vehicle mileage:",school_bus.mileage)
```

Object-Oriented programming methodologies

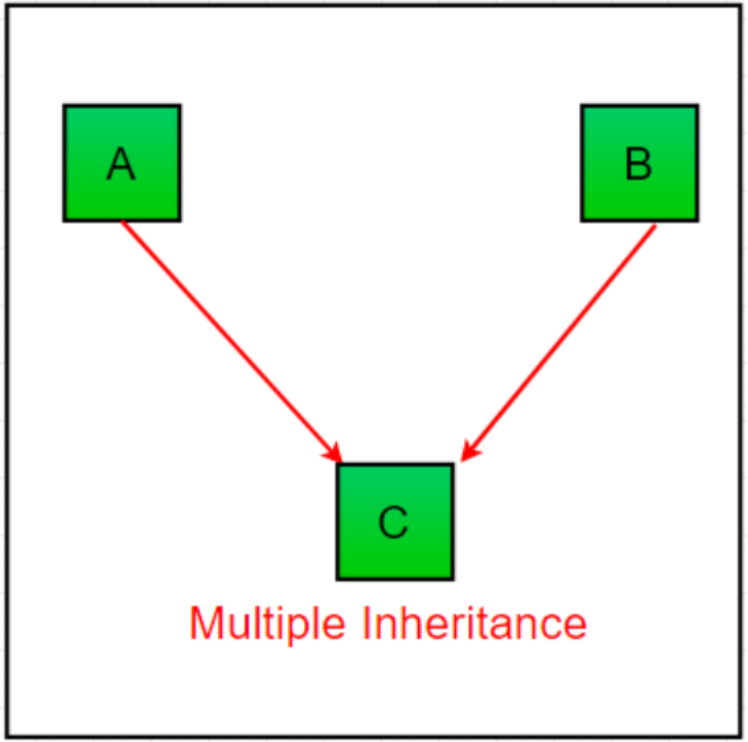
Types of Inheritance

Single Inheritance: parent and child



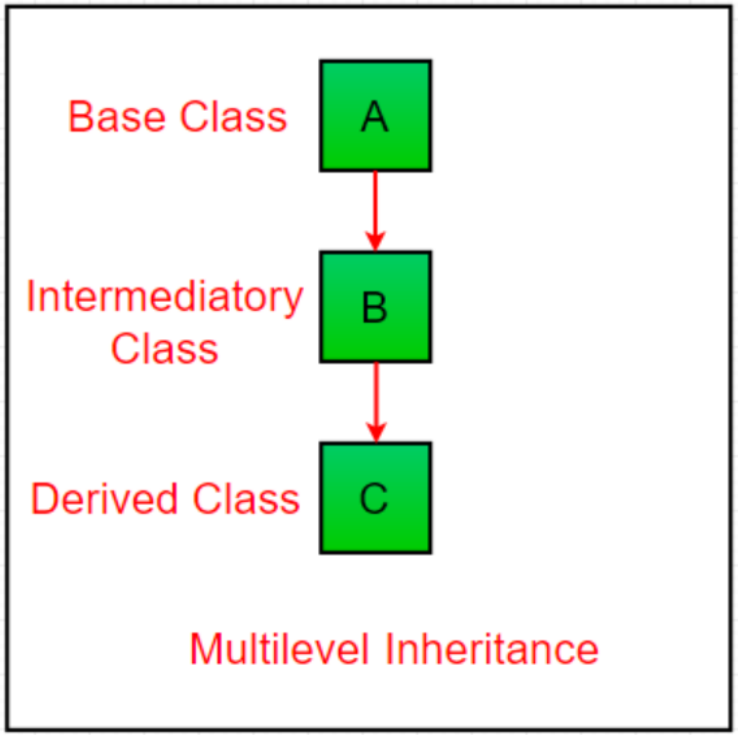
```
class Parent:
    def func1(self):
        print("This function is in parent class.")
# Derived class
class Child(Parent):
    def func2(self):
        print("This function is in child class.")

 # main
object = Child()
object.func1()
object.func2()
```

Multiple Inheritance:2 base classes+1 derived class



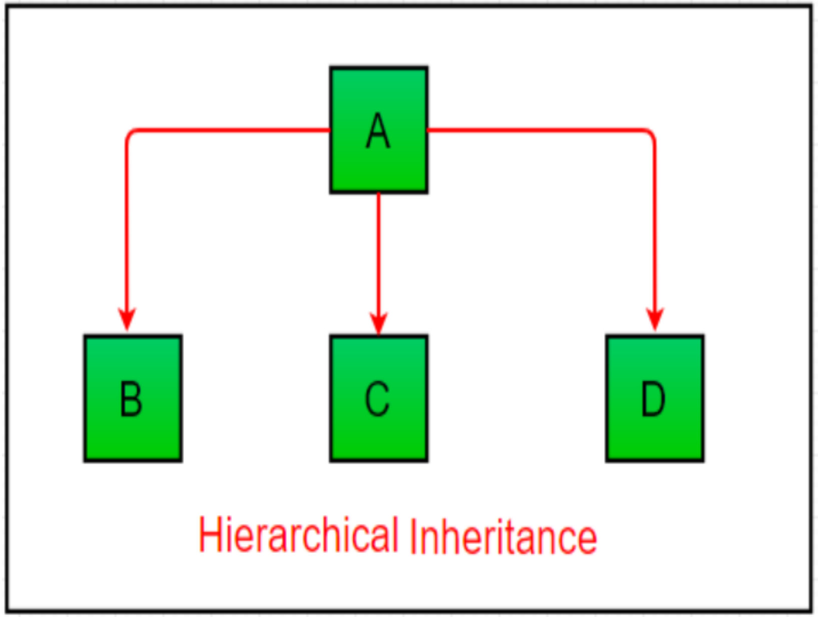
```
# Base class1
class Mother:
    mothername = ""
    def mother(self):
        print(self.mothername)
# Base class2
class Father:
    fathername = ""
    def father(self):
        print(self.fathername)
# Derived class
class Son(Mother, Father):
    def parents(self):
        print("Father :", self.fathername)
        print("Mother :", self.mothername)
# Driver's code
s1 = Son()
s1.fathername = “Ahmad"
s1.mothername = “Zaineb"
s1.parents()
```

Multilevel Inheritance: This is similar to a relationship representing a child and a grandfather.



```
# Base class
class Grandfather:
    def __init__(self, grandfathername):

        self.grandfathername = grandfathername
# Intermediate class
class Father(Grandfather):
    def __init__(self, fathername, grandfathername):
        self.fathername = fathername
        # invoking constructor of Grandfather class
        Grandfather.__init__(self, grandfathername)


# Derived class
class Son(Father):
    def __init__(self, sonname, fathername, grandfathername):
        self.sonname = sonname
        # invoking constructor of Father class
        Father.__init__(self, fathername, grandfathername)
    def print_name(self):
        print('Grandfather name :', self.grandfathername)
        print("Father name :", self.fathername)
        print("Son name :", self.sonname)
# main
s1 = Son('Prince', 'Rampal', 'Lal mani')
print(s1.grandfathername)
s1.print_name()
```

Hierarchical Inheritance: we have a parent (base) class and two child (derived) classes.



```
# Base class
class Parent:
    def func1(self):
        print("This function is in parent class.")
# Derived class1
class Child1(Parent):
    def func2(self):
        print("This function is in child 1.")
# Derived class2
class Child2(Parent):
    def func3(self):
        print("This function is in child 2.")
# main code
object1 = Child1()
object2 = Child2()
object1.func1()
object1.func2()
object2.func1()
object2.func3()
```

Single Inheritance:

Single level inheritance enables a derived class to inherit characteristics from a single parent class.

```
class employee1:
    def __  init __(self, name, age, salary):
        self.name = name
        self.age    =  age
        self.salary     =  salary

class   childemployee     (employee1):#This is a child class

    pass



emp1 = employee1(‘Ahmad',22,1000)
print(emp1.age)
emp2=childemployee("mona",25,2000)
print(emp2.salary)
```

Note:
We can, also, use in the body of the child class

```
  pass   Or
def __init__(self, name,age,salary):
    employee1.__init__(self,
name,age,salary)
```

Multilevel Inheritance(1/2)

Multi-level-inheritance: **enables a derived class to inherit properties from an immediate parent class which in turn inherits properties from his parent class.**

Example:

```
class employee:
    def __init__(self,name,age,salary ):
        self.name = name
        self.age  = age
        self.salary  = salary
class childemployee1(employee):#First child class
        pass
```

Note:
We can, also, use in the body of the first child class

```
  pass   Or
def __init__(self, name,age,salary):
     employee.__init__(self, name,age,salary)
```

Multilevel Inheritance(2/2)

```
class childemployee2(childemployee1):#Second child class
        pass




emp1 = employee(‘Ahmad',22,1000)
emp2 = childemployee1(‘Sarah',23,2000)


print(emp1.age)
print(emp2.age)
```

Note:
We can, also, use in the body of the second child class

```
  pass   Or
def __init__(self, name,age,salary):
    childemployee1.__init__(self,
name,age,salary)
```

Output: 22,23

Hierarchical Inheritance:(1/2)

Hierarchical Inheritance: Hierarchical level inheritance enables more than one derived class to inherit properties from a parent class.

Example:

```
#Hierarchical Inheritance
 class employee:
     def __init__(self, name, age, salary):
         self.name = name
         self.age = age
         self.salary  = salary
```

Hierarchical Inheritance:(2/2)

```
class childemployee1(employee):
      pass



class childemployee2(employee):
     pass



emp1 = employee(' Ali ',22,1000)
emp2 = employee(' Mohamad ',23,2000)
```

Note:
We can, also, use in the body of the first child class

```
   pass   Or
def __init__(self, name,age,salary):
    employee.__init__(self,
name,age,salary)
```

Note:
We can, also, use in the body of the Second child class

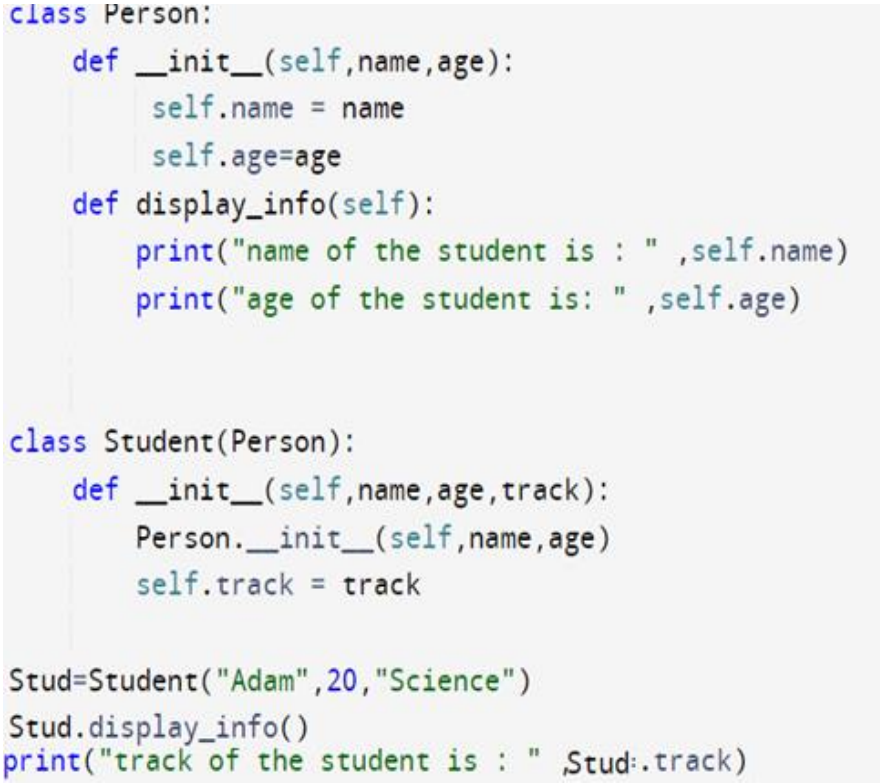
```
   pass   Or
def __init__(self, name,age,salary):
    employee.__init__(self,
name,age,salary)
```

More examples: Single Inheritance

- Create **class person** with attributes: Name, age and method **display_Info()**
- Create **class student** that inherits all the methods and properties from class Person and add to this class **new instance attribute named “track”**
- Create 2 students and display their information

More examples: Single Inheritance

- Create **class person** with attributes: Name, age and method **display_Info()**
- Create **class student** that inherits all the methods and properties from from class Person and add to this class **new instance attribute named "track"**
- Create 2 students and display their information

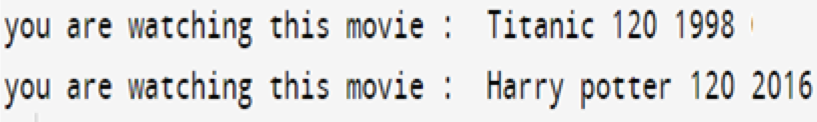


```
class Person:
    def __init__(self,name,age):
         self.name = name
         self.age=age
    def display_info(self):
        print("name of the student is : " ,self.name)
        print("age of the student is: " ,self.age)


class Student(Person):
    def __init__(self,name,age,track):
        Person.__init__(self,name,age)
        self.track = track

Stud=Student("Adam",20,"Science")
Stud.display_info()
print("track of the student is : " ,Stud.track)
```

More examples: Hierarchical Inheritance

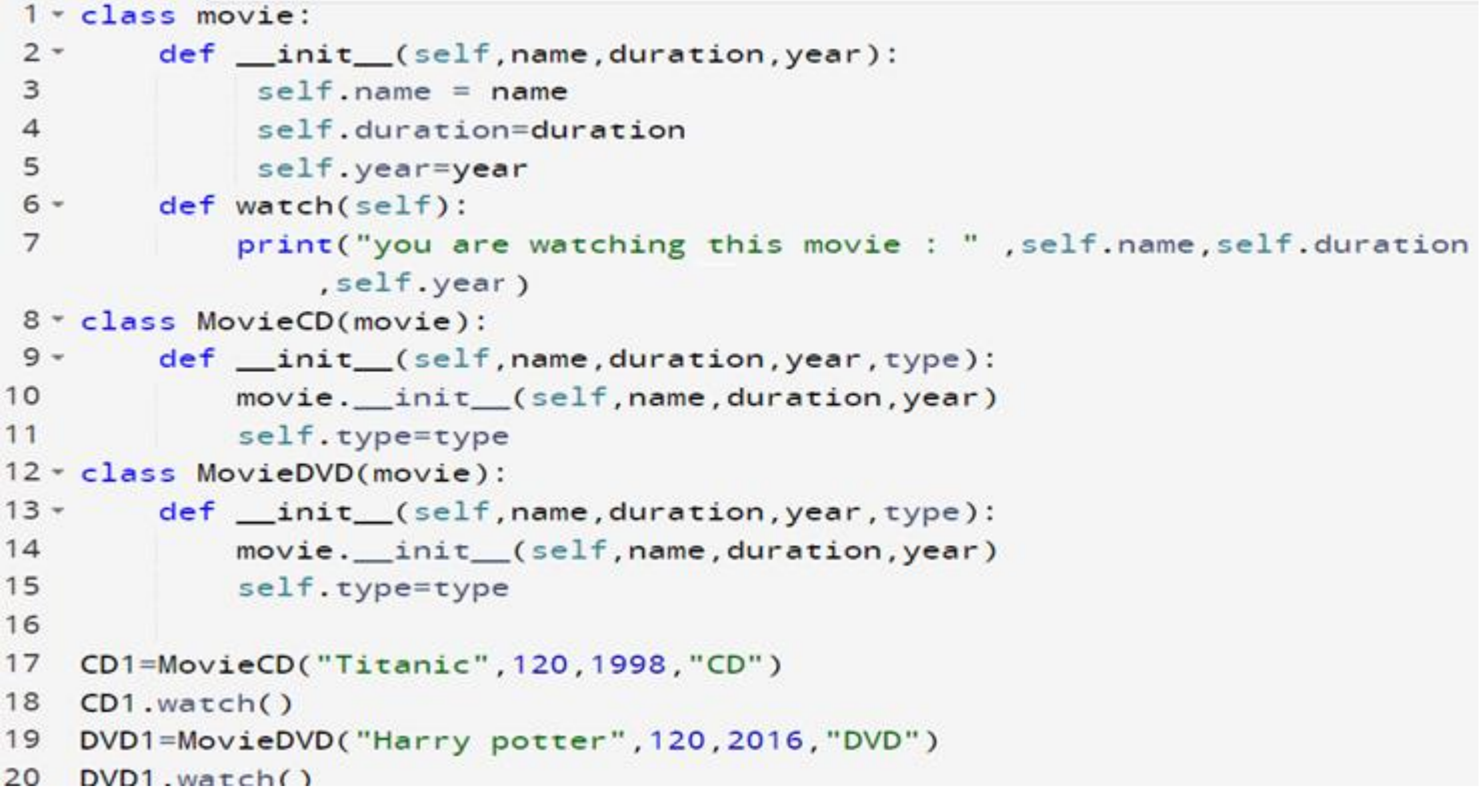
- Create **class Movie** with attributes: Name, duration,year and method **watch() to display all info of the movie**
- Create **class MovieCD** that inherits all the methods and properties from class Movie. Add **the attribute TYPE**
- Create **class MovieDVD** that inherits all the methods and properties from class Movie. Add **the attribute Type**

- Create 1 object from class MovieCD and display the information
- Create 1 object from class MovieDVD and display the information



```
you are watching this movie :  Titanic 120 1998
you are watching this movie :  Harry potter 120 2016
```

More examples: Hierarchical Inheritance

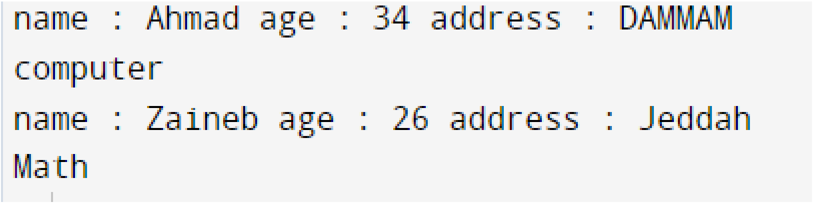


```
class movie:
    def __init__(self,name,duration,year):
         self.name = name
         self.duration=duration
         self.year=year
    def watch(self):
        print("you are watching this movie : " ,self.name,self.duration,self.year)
class MovieCD(movie):
    def __init__(self,name,duration,year,type):
        movie.__init__(self,name,duration,year)
        self.type=type
class MovieDVD(movie):
    def __init__(self,name,duration,year,type):
        movie.__init__(self,name,duration,year)
        self.type=type

CD1=MovieCD("Titanic",120,1998,"CD")
CD1.watch()
DVD1=MovieDVD("Harry potter",120,2016,"DVD")
DVD1.watch()
```

More examples: Multiple Inheritance

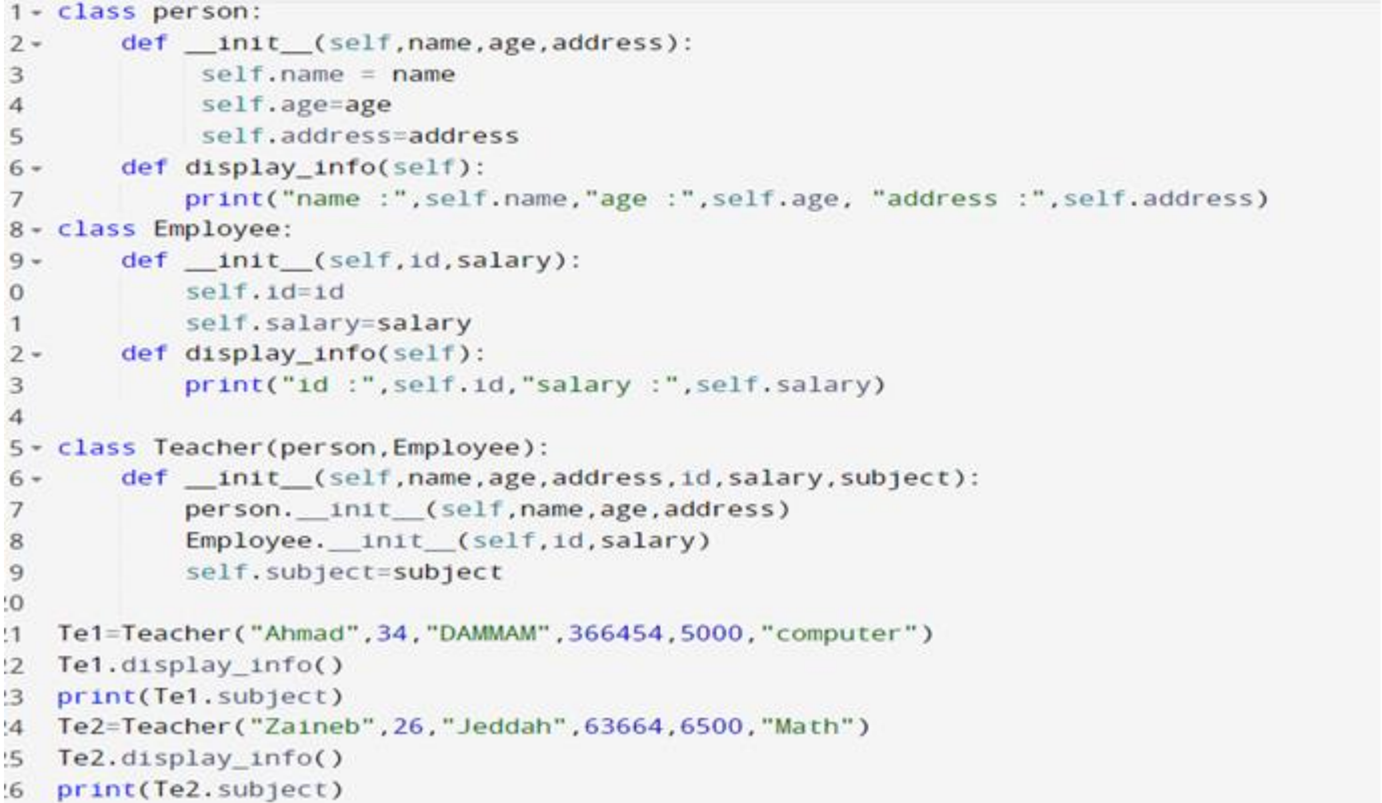
- Create **class person** with attributes: Name, Age,address and method **display_info() to display all info**
- Create **class Employee** with attributes: ID, salary and method **display_info() to display all info**
- Create **class Teacher** that inherits all the methods and properties from **classes person and Employee.** Add **the attribute Subject**

- Create 2 objects from class teacher and display their information and the subject



```
name : Ahmad age : 34 address : DAMMAM
computer
name : Zaineb age : 26 address : Jeddah
Math
```

More examples: Multiple Inheritance

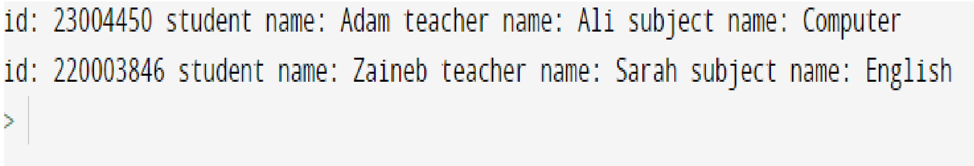


```python
class person:
    def __init__(self,name,age,address):
        self.name = name
        self.age=age
        self.address=address
    def display_info(self):
        print("name :",self.name,"age :",self.age, "address :",self.address)
class Employee:
    def __init__(self,id,salary):
        self.id=id
        self.salary=salary
    def display_info(self):
        print("id :",self.id,"salary :",self.salary)

class Teacher(person,Employee):
    def __init__(self,name,age,address,id,salary,subject):
        person.__init__(self,name,age,address)
        Employee.__init__(self,id,salary)
        self.subject=subject

Te1=Teacher("Ahmad",34,"DAMMAM",366454,5000,"computer")
Te1.display_info()
print(Te1.subject)
Te2=Teacher("Zaineb",26,"Jeddah",63664,6500,"Math")
Te2.display_info()
print(Te2.subject)
```

More examples: Multilevel Inheritance

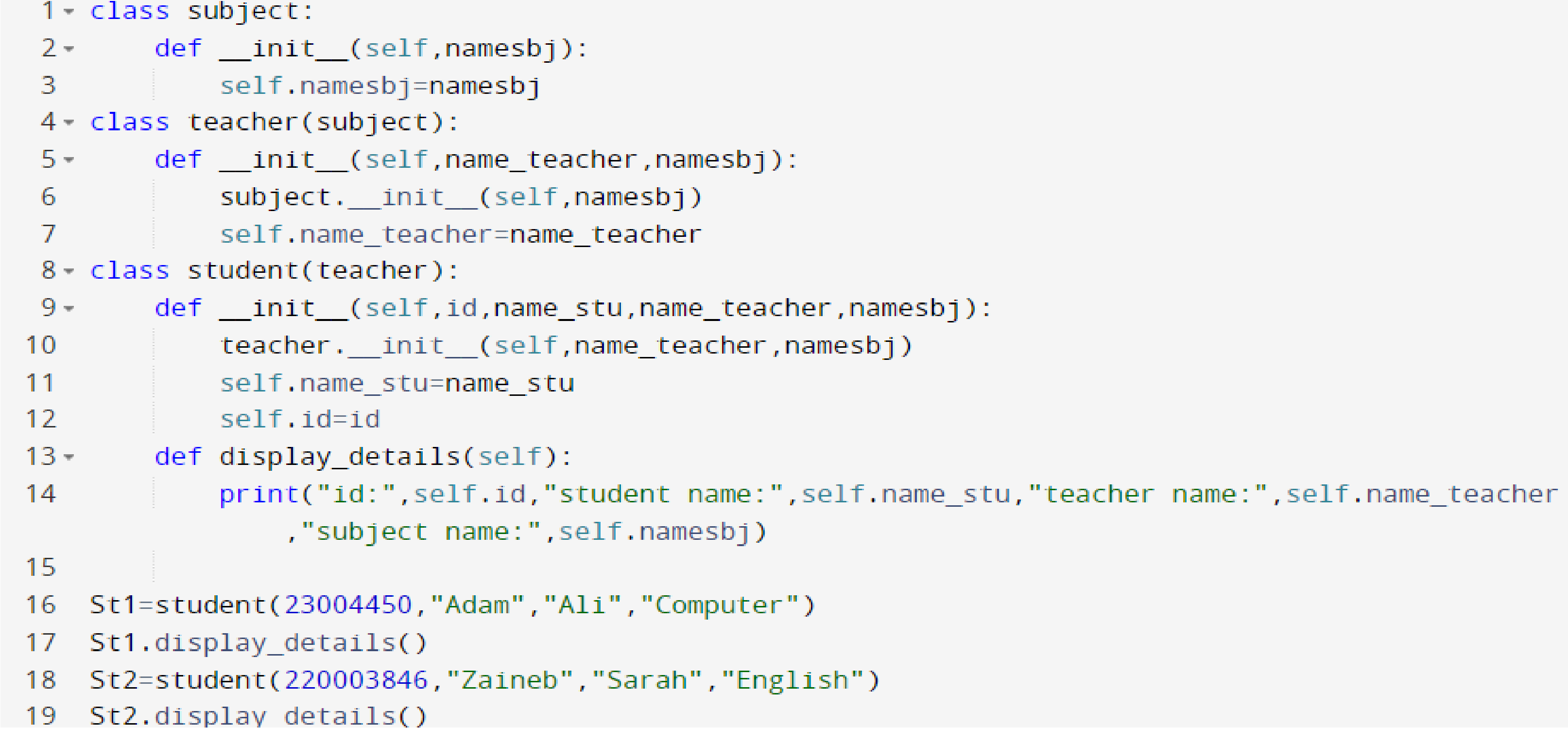
- Create **class subject** with attribute: **namesbj**
- Create **class Teacher** that inherits the attributes **of class subject** and add the **attribute name_teacher**
- Create **class Student** that inherits **subj_name** and **name_teacher**. Add the following attributes:**id,name_stu**
- Create **the method Stu_details** to print all details of the student.

- Create two students and display all their information.



```
id: 23004450 student name: Adam teacher name: Ali subject name: Computer
id: 220003846 student name: Zaineb teacher name: Sarah subject name: English
>
```

More examples: Multilevel Inheritance



```
class subject:
    def __init__(self,namesbj):
        self.namesbj=namesbj
class teacher(subject):
    def __init__(self,name_teacher,namesbj):
        subject.__init__(self,namesbj)
        self.name_teacher=name_teacher
class student(teacher):
    def __init__(self,id,name_stu,name_teacher,namesbj):
        teacher.__init__(self,name_teacher,namesbj)
        self.name_stu=name_stu
        self.id=id
    def display_details(self):
        print("id:",self.id,"student name:",self.name_stu,"teacher name:",self.name_teacher,"subject name:",self.namesbj)

St1=student(23004450,"Adam","Ali","Computer")
St1.display_details()
St2=student(220003846,"Zaineb","Sarah","English")
St2.display_details()
```

Thanks!

Any questions?