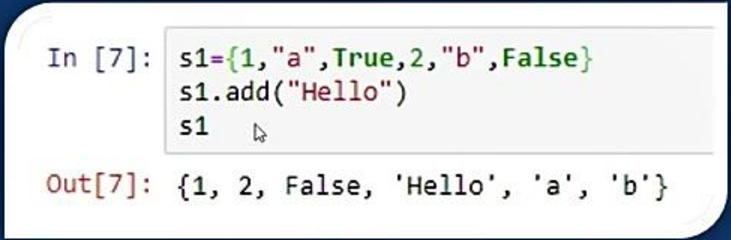
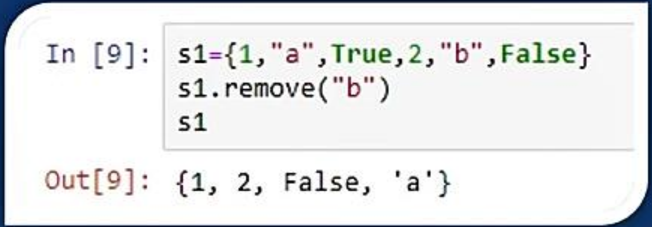
Set Operations

Update one dictionary's elements with another



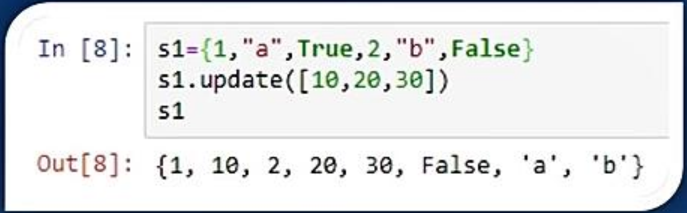
```
In [7]:  s1={1,"a",True,2,"b",False}
         s1.add("Hello")
         s1

Out[7]:  {1, 2, False, 'Hello', 'a', 'b'}
```

Removing an element



```
In [9]:  s1={1,"a",True,2,"b",False}
         s1.remove("b")
         s1

Out[9]:  {1, 2, False, 'a'}
```

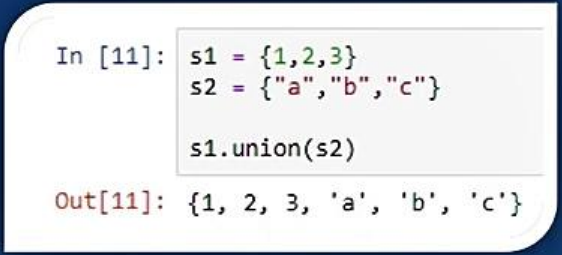
Updating multiple elements



```
In [8]:  s1={1,"a",True,2,"b",False}
         s1.update([10,20,30])
         s1

Out[8]:  {1, 10, 2, 20, 30, False, 'a', 'b'}
```

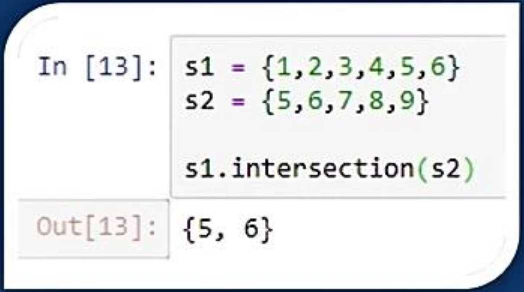
Set Functions

Union of two sets



```
In [11]: s1 = {1,2,3}
         s2 = {"a","b","c"}

         s1.union(s2)

Out[11]: {1, 2, 3, 'a', 'b', 'c'}
```

Intersection of two sets



```
In [13]: s1 = {1,2,3,4,5,6}
         s2 = {5,6,7,8,9}

         s1.intersection(s2)

Out[13]: {5, 6}
```

If Statement

If
It’s raining:
Sit inside

else
Go out and Play Football

$\longrightarrow$

If Statement

If
It's raining:
Sit inside

else
Go out and Play Football

$\longrightarrow$

If Statement

greatlearning
Learning for Life

If
Marks > 70:
Get Ice-cream

else
Give Practice Test

$\longrightarrow$

if…else Pseudo Code

```
If(condition){
Statements to be executed….
}

else{
Statements to be executed….
}
```

Looping Statements

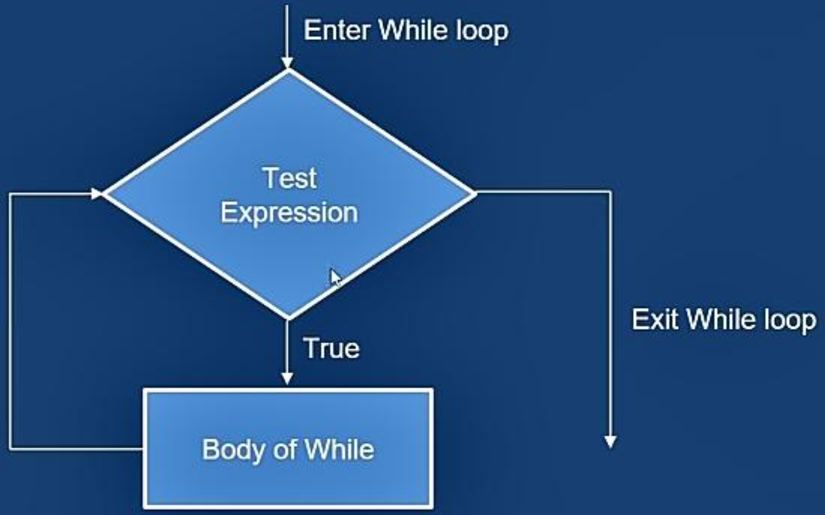
Looping statements are used to repeat a task multiple times

Keep filling this bucket with a mug of water *while* it is not full

Looping Statements

Looping Statements

While Loop

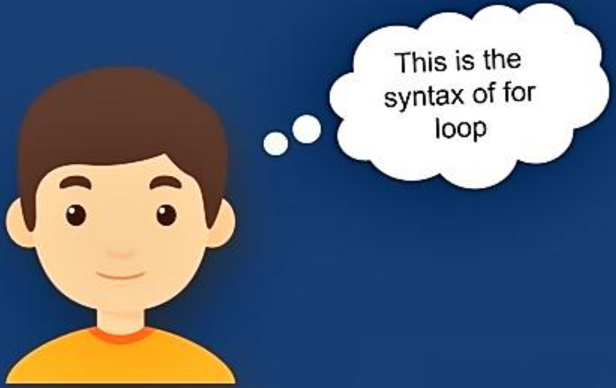


Syntax:

```
while condition:
        Execute Statements
```

For Loop

For Loop is used to iterate over a sequence(tuple, list, dictionary..)



```
for val in sequence:
              Body of for
```

Functions in Real Life

greatlearning
Learning for Life

Eating

Running

Cycling

Python Functions

Function is a block of code which performs a specific task

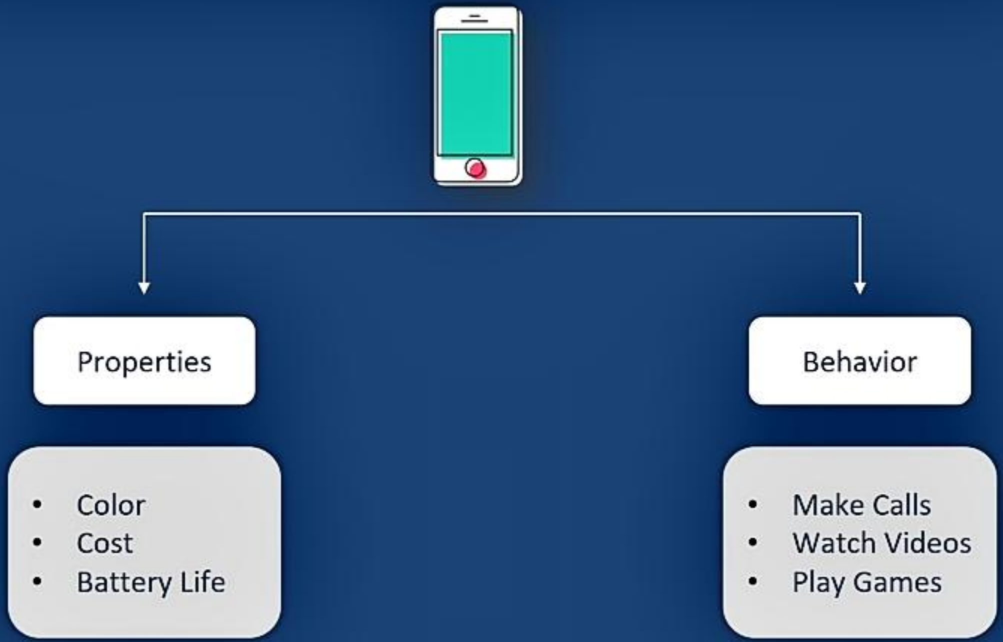
Deposit	$\longrightarrow$ Function to deposit money
Withdraw	$\longrightarrow$ Function to withdraw money
Balance	$\longrightarrow$ Function to check balance

Python Object Oriented Programming

greatlearning
Learning for Life

Classes

Class is a template/blue-print for real-world entities



Properties

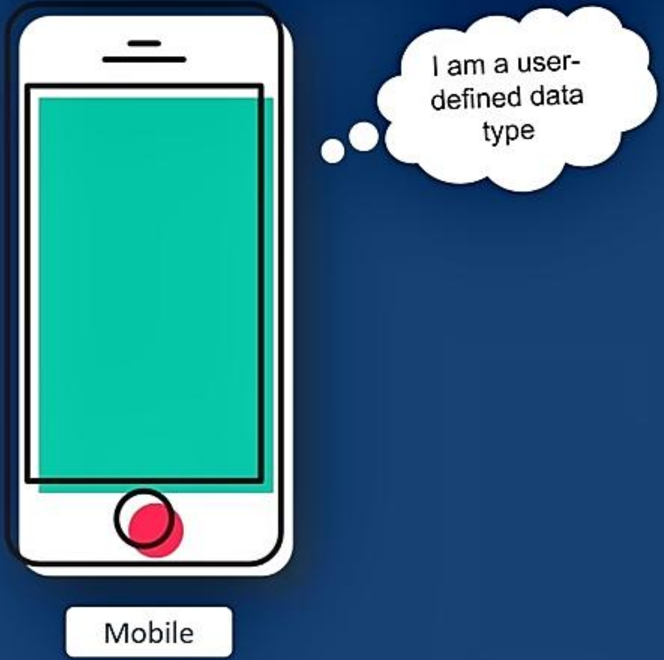
- Color
- Cost
- Battery Life

Behavior

- Make Calls
- Watch Videos
- Play Games

Class in Python

Class is a user-defined data-type



int

float

bool

str

Objects

Objects are specific instances of a class

Apple
Motorola
Samsung

Objects in Python

Specific instances of Mobile data type

Apple

Motorola

Samsung

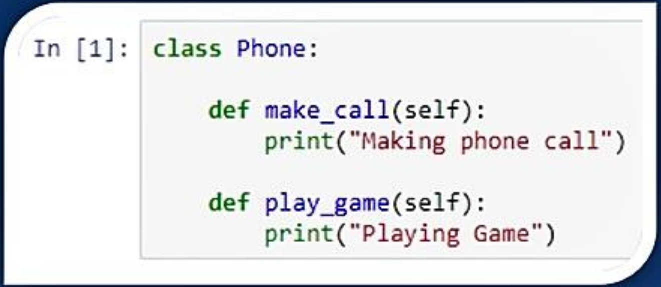
a = 10

b = 20

c = 30

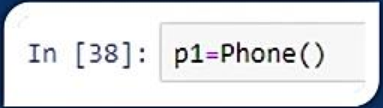
Specific instances of integer data type

Creating the first Class



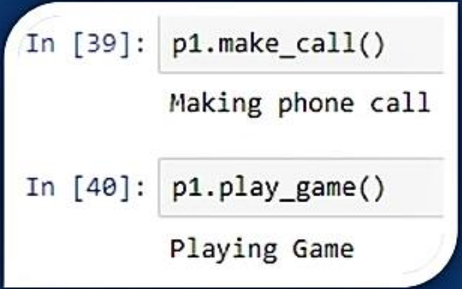
```
In [1]: class Phone:

            def make_call(self):
                print("Making phone call")

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

Creating the ‘Phone’ class



```
In [38]: p1=Phone()
```

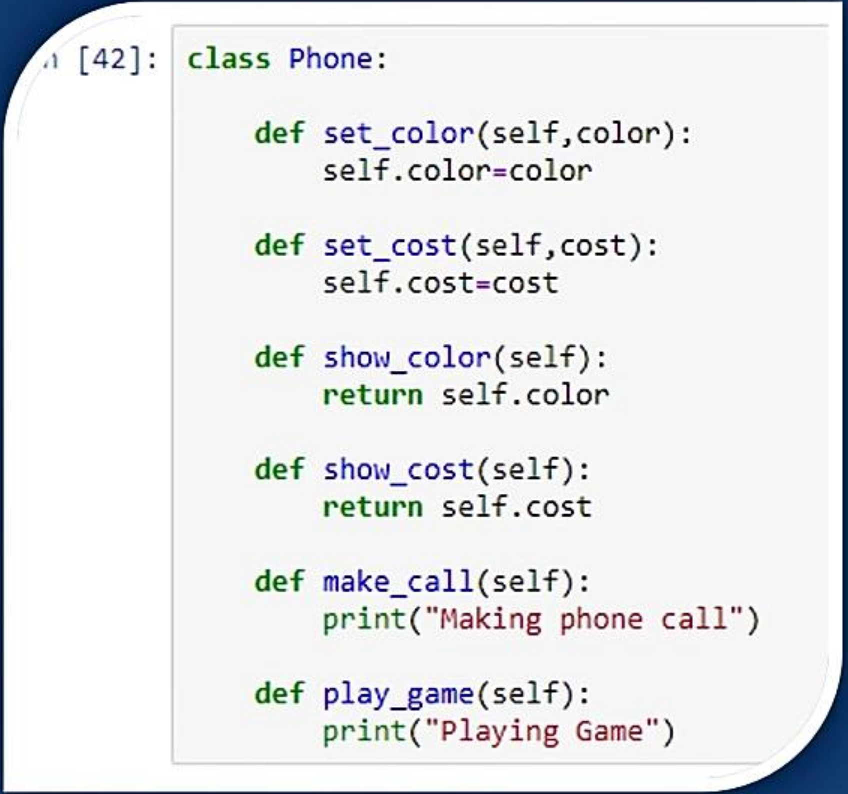
Instantiating the ‘p1’ object



```
In [39]: p1.make_call()

         Making phone call

In [40]: p1.play_game()

         Playing Game
```

Invoking methods through object

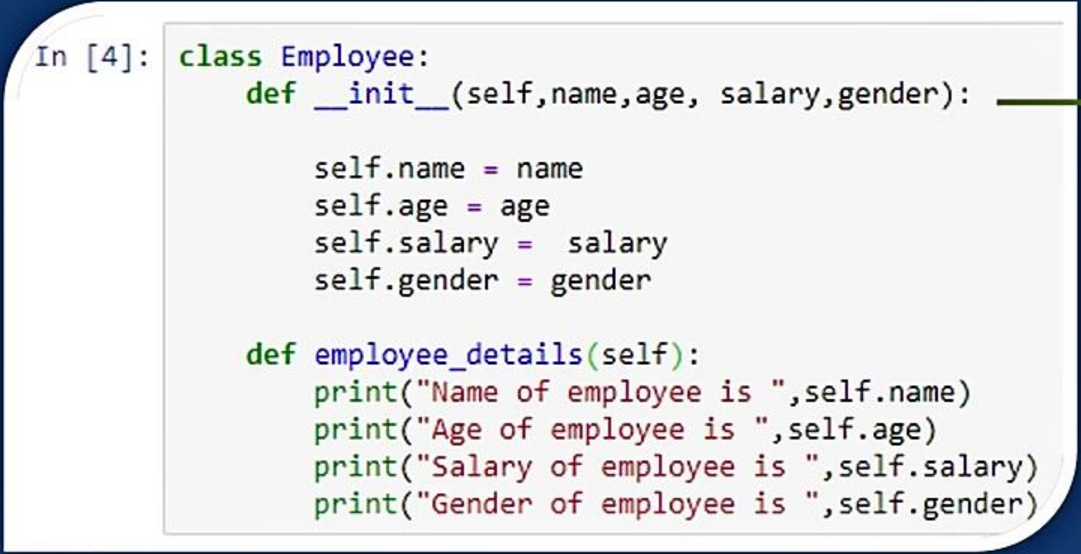
Adding parameters to the class



```
[42]: class Phone:

          def set_color(self,color):
              self.color=color

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

          def show_color(self):
              return self.color

          def show_cost(self):
              return self.cost

          def make_call(self):
              print("Making phone call")

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

Setting and Returning the attribute values

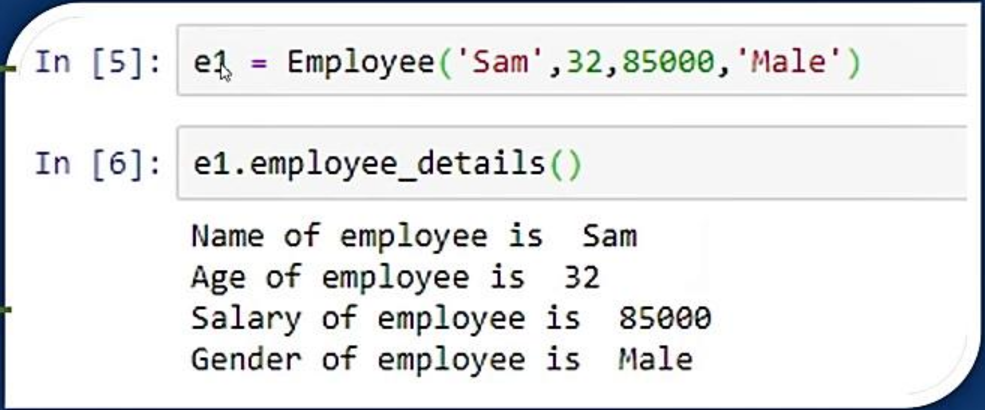
Creating a class with Constructor



```
In [4]: 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 employee is ",self.name)
                print("Age of employee is ",self.age)
                print("Salary of employee is ",self.salary)
                print("Gender of employee is ",self.gender)
```

init method acts as the constructor

Instantiating Object



Instantiating the 'e1' object

```
In [5]: e1 = Employee('Sam',32,85000,'Male')
```

Invoking the 'employee_details' method

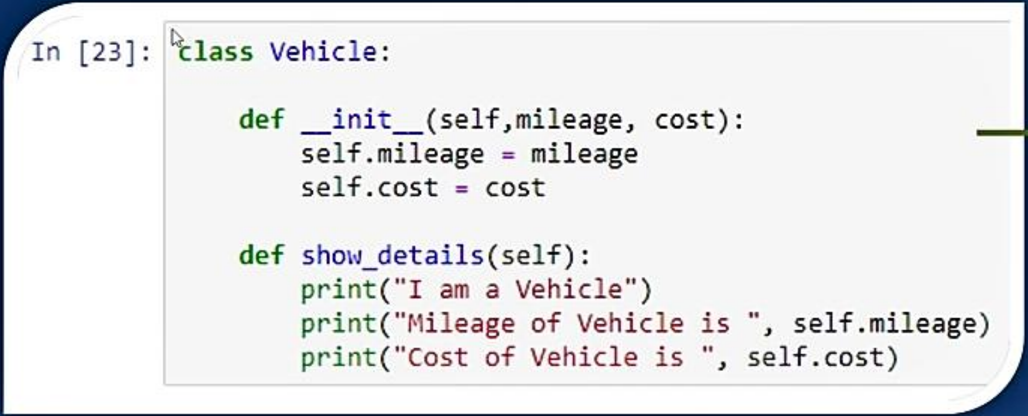
```
In [6]: e1.employee_details()

Name of employee is  Sam
Age of employee is  32
Salary of employee is  85000
Gender of employee is  Male
```

Inheritance in Python

With inheritance one class can derive the properties of another class

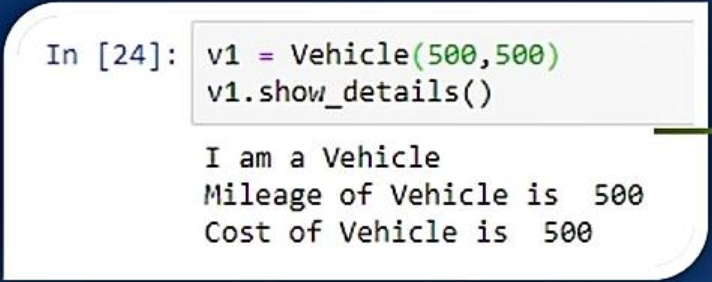
Man inheriting features from his father

Inheritance Example



```
In [23]: class Vehicle:

             def __init__(self,mileage, cost):
                 self.mileage = mileage
                 self.cost = cost

             def show_details(self):
                 print("I am a Vehicle")
                 print("Mileage of Vehicle is ", self.mileage)
                 print("Cost of Vehicle is ", self.cost)
```

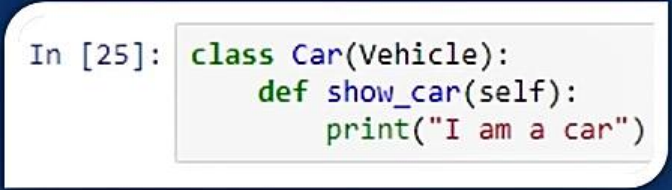
Creating the base class



```
In [24]: v1 = Vehicle(500,500)
         v1.show_details()

         I am a Vehicle
         Mileage of Vehicle is  500
         Cost of Vehicle is  500
```

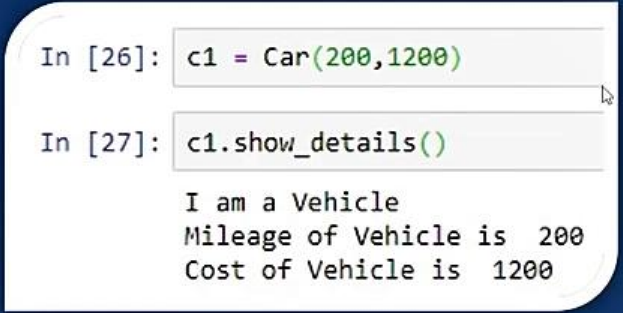
Instantiating the object for base class

Inheritance Example



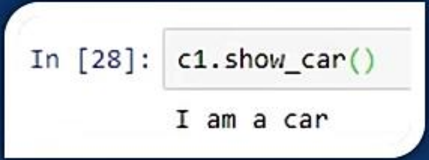
```
In [25]: class Car(Vehicle):
             def show_car(self):
                 print("I am a car")
```

Creating the child class



```
In [26]: c1 = Car(200,1200)

In [27]: c1.show_details()

         I am a Vehicle
         Mileage of Vehicle is  200
         Cost of Vehicle is  1200
```

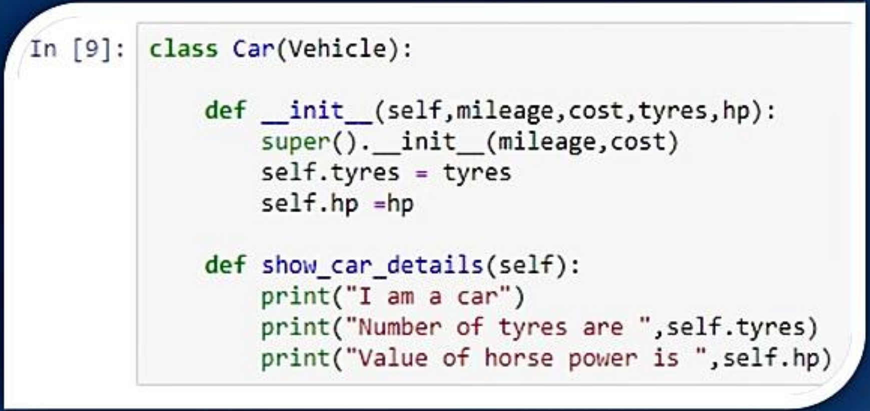
Instantiating the object for child class



```
In [28]: c1.show_car()

         I am a car
```

Invoking the child class method

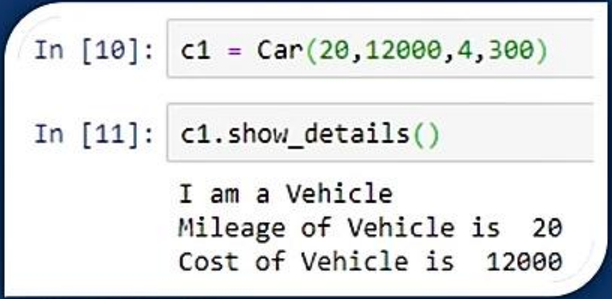
Over-riding init method



```
In [9]: class Car(Vehicle):

            def __init__(self,mileage,cost,tyres,hp):
                super().__init__(mileage,cost)
                self.tyres = tyres
                self.hp =hp

            def show_car_details(self):
                print("I am a car")
                print("Number of tyres are ",self.tyres)
                print("Value of horse power is ",self.hp)
```

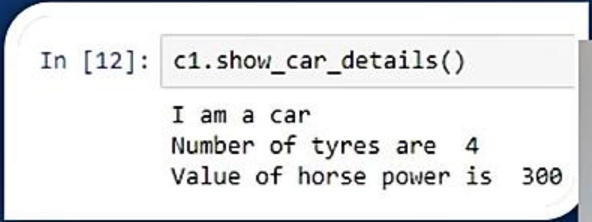
$\longrightarrow$ Over-riding init method

Invoking show_details()
method from parent class



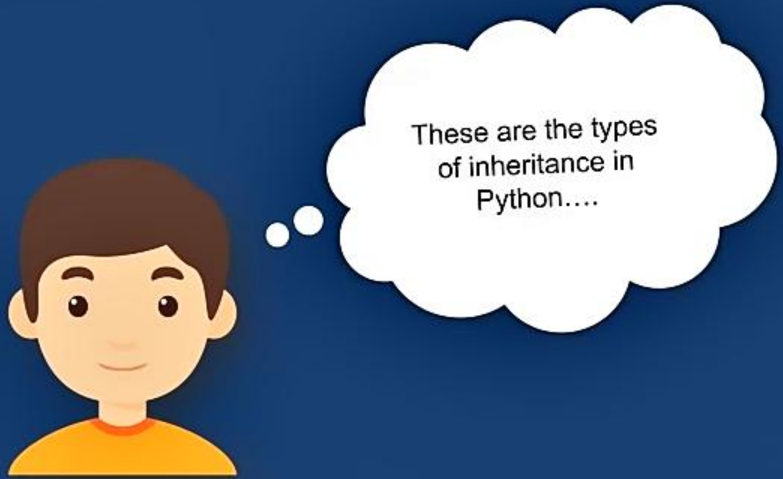
```
In [10]: c1 = Car(20,12000,4,300)

In [11]: c1.show_details()

         I am a Vehicle
         Mileage of Vehicle is  20
         Cost of Vehicle is  12000
```

Invoking show_car_details()
method from child class



```
In [12]: c1.show_car_details()

         I am a car
         Number of tyres are  4
         Value of horse power is  300
```

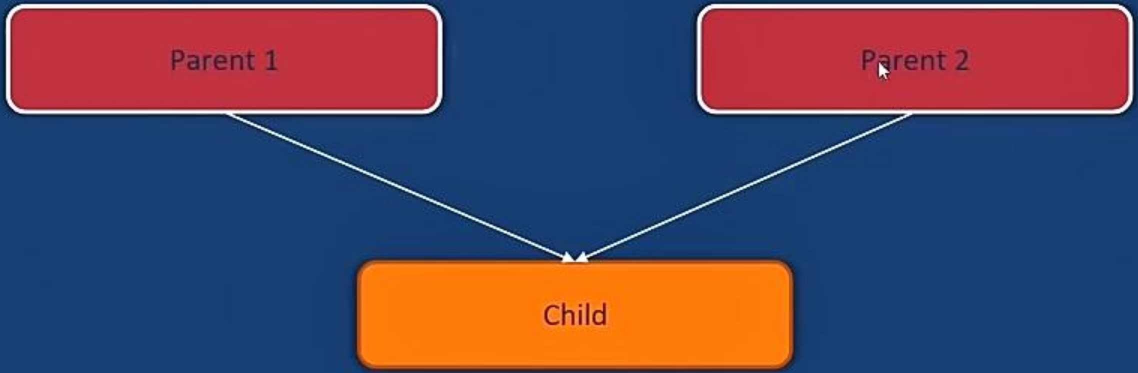
Types of Inheritance



- Single Inheritance
- Multiple Inheritance
- Multi-level Inheritance
- Hybrid Inheritance

Multiple Inheritance

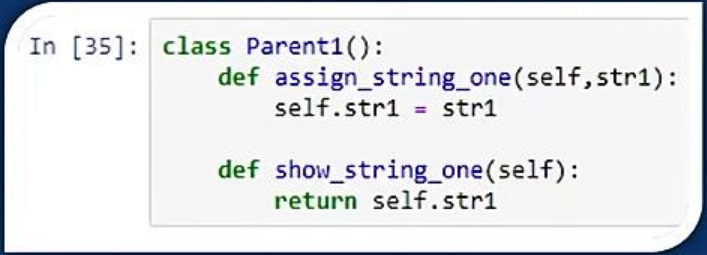
In multiple inheritance, the child inherits from more than 1 parent class



Multiple Inheritance in Python

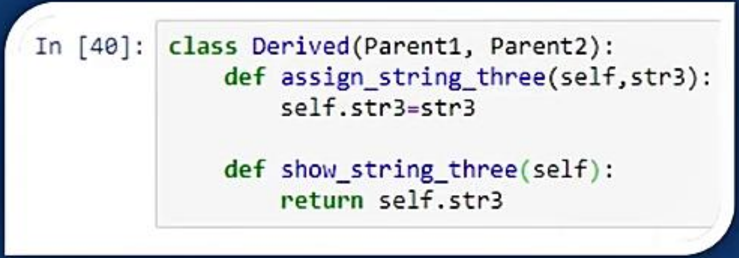
greatlearning
Learning for Life

Parent Class One



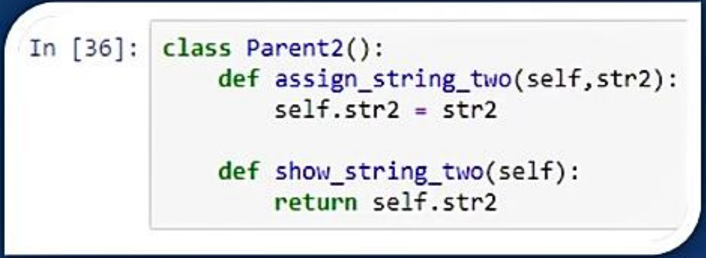
```
In [35]: class Parent1():
             def assign_string_one(self,str1):
                 self.str1 = str1

             def show_string_one(self):
                 return self.str1
```

Child Class



```
In [40]: class Derived(Parent1, Parent2):
             def assign_string_three(self,str3):
                 self.str3=str3

             def show_string_three(self):
                 return self.str3
```

Parent Class Two

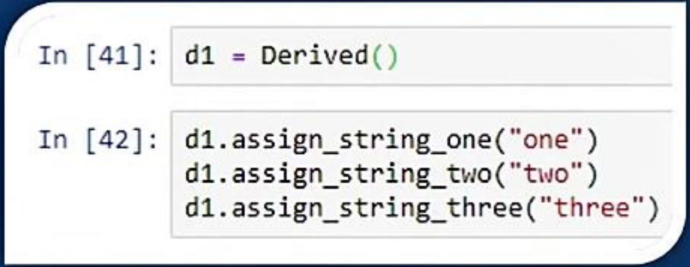


```
In [36]: class Parent2():
             def assign_string_two(self,str2):
                 self.str2 = str2

             def show_string_two(self):
                 return self.str2
```

Multiple Inheritance in Python

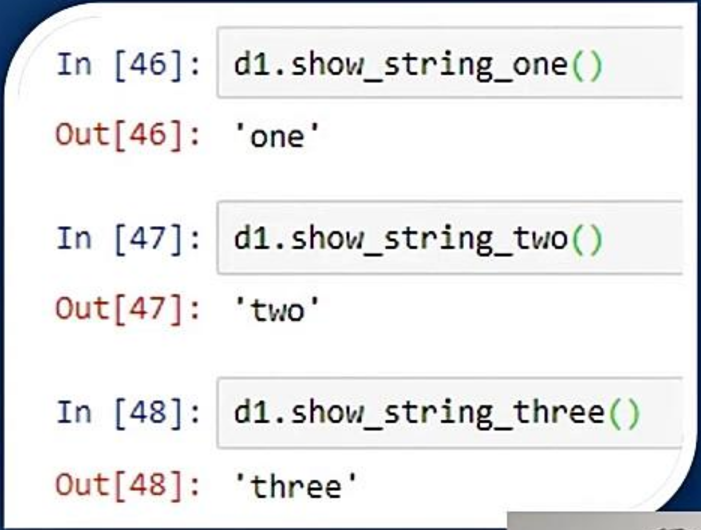
greatlearning
Learning for Life

Instantiating object of child class



```
In [41]: d1 = Derived()

In [42]: d1.assign_string_one("one")
         d1.assign_string_two("two")
         d1.assign_string_three("three")
```

Invoking methods



```
In [46]: d1.show_string_one()

Out[46]: 'one'

In [47]: d1.show_string_two()

Out[47]: 'two'

In [48]: d1.show_string_three()

Out[48]: 'three'
```