Content 12

**Inheritance**

Inheritance is a way of creating a new class from an existing class.

**Syntax:**

class Emoloyee: #Base Class

#Code

class Programmer(Employee): #Derived or child class

#Code

We can use the methods and attributes of Employee in Programmer object.

Also, we can overwrite or add new attributes and methods in the Programmer class.

##### **Type of Inheritance**

1. Single inheritance
2. Multiple inheritance
3. Multilevel inheritance

**Single Inheritance**

Single inheritance occurs when child class inherits only a single parent class.

Base -> Derived

**Code For single inheritance (advance Code):**

# Simple Inheritance

class Employee:

    name1= "Arman"

    salary1 = "45k"

class programmer(Employee):

    def \_\_init\_\_(self,name2,salary2,language2):

        self.name2= name2

        self.salary2= salary2

        self.language2 = language2

    def display(self):

        print(f"The name is: {self.name2}")

        print(f"The salary is: {self.salary2}")

        print(f"The language is: {self.language2}")

#simple use of base class

print("The Employee's Details...")

e = Employee()

print(e.name1)

print(e.salary1)

#cinstructor use in derived class

print("\n\nThe Progammers Details...")

p = programmer("Aamaan","40k","python")

p.display()

#derived class use

print("\n\nDetails of Employee class by Derived class....")

print(p.name1)

print(p.salary1)

**Output:**

The Employee's Details...

Arman

45k

The Progammers Details...

The name is: Aamaan

The salary is: 40k

The language is: python

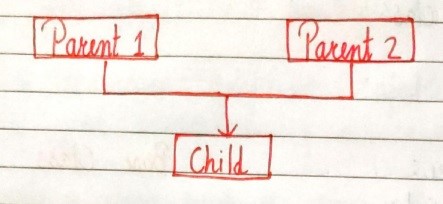
Details of Employee class by Derived class....

Arman

45k

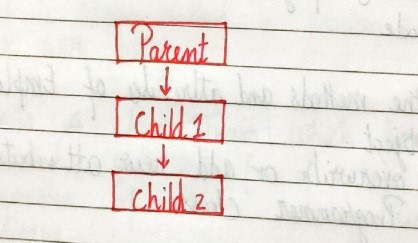
**Multiple Inheritance**

Multiple inheritance occurs when the child class inherits from more than one parent class.



**Multilevel Inheritance**

When a child class becomes a parent for another child class.



**Super() method**

Super method is used to access the methods of a super class in the derived class.

super().\_\_init\_\_()  #Calls constructor of the base class

**Code For super Class:**

class Employee:

    work = "Operator"

    def work\_details(self):

        print("My Work is:~ ",e.work)

        print("The work is hard but salary is good...")

class teacher(Employee):

    work= "Teaching"

    def work\_details(self):

        super().work\_details()

        print("\n")

        print("My Work is:~ ",t.work)

        print("The work is good but salary is low...")

class Programmer(teacher):

    work="Computer programming"

    def work\_details(self):

        super().work\_details()

        print("\n")

        print("My Work is:~",p.work)

        print("The work is best and salary is also good...")

e = Employee()

t= teacher()

p = Programmer()

p.work\_details()

**Output:**

My Work is:~ Operator

The work is hard but salary is good...

My Work is:~ Teaching

The work is good but salary is low...

My Work is:~ Computer programming

The work is best and salary is also good...

**Class methods**

A class method is a method which is bound to the class and not the object of the class.

@classmethod decorator is used to create a class method.

Syntax to create a class method:

@classmethod

def (cls, p1, p2):

#code

**Code For Class method (Changing the attribute of class):**

class Employee:

    name = "Aaman"

    salary = "450k"

    # changing the Salary

    @classmethod

    def change\_salary(cls, new\_amt):

        print(f"changing the salary amount '{e.salary}' with '{new\_amt}'")

        cls.salary = new\_amt

    @classmethod

    def change\_name(cls, new\_name):

        print(f"changing the name '{e.name}' with '{new\_name}'")

        cls.name = new\_name

e = Employee()

#changing the salary

print("Previous salary: ", e.salary)

e.change\_salary("200k")

print(f"The changed Salary is: {e.salary} ")

#changing the name

print("\n\n")

print("The prevoius name is: ", e.name)

e.change\_name("Shaalam")

print("The changed name is: ", e.name)

**Output:**

Previous salary: 450k

changing the salary amount '450k' with '200k'

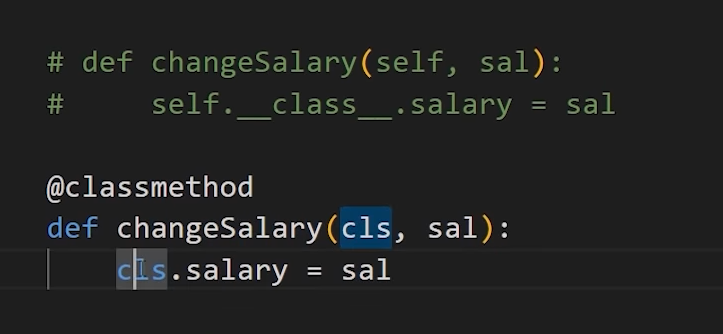
The changed Salary is: 200k

The prevoius name is: Aaman

changing the name 'Aaman' with 'Shaalam'

The changed name is: Shaalam

**Alternate Method for Changing:**



**@property decorators**

Consider the following class

class Employee:

@property

def name(self):

return self.ename

if e = Employee() is an object of class employee, we can print (e.name) top print the ename/call name() function.

**@.getters and @.setters**

The method name with @property decorator is called getter method.

We can define a function + @name.setter decorator like below:

@name.setter

def name(self, value):

self.ename = value

**Operator overloading in Python**

Operators in python can be overloaded using dunder methods.

These methods are called when a given operator is used on the objects.

Operators in python can be overloaded using the following methods:

p1 + p2 -> p1.\_\_add\_\_(p2)

p1 – p2 -> p1.\_\_sub\_\_(p2)

p1 \* p2 -> p1.\_\_mul\_\_(p2)

p1 / p2 -> p1.\_\_truediv\_\_(p2)

p1 // p2 -> p1.\_\_floordiv\_\_(p2)

**Other dunder/magic methods in Python**

\_\_str\_\_() -> used  to set what gets displayed upon calling str(obj)

\_\_len\_\_() -> used to set what gets displayed upon calling .\_\_len\_\_() or len(obj)