Inheritance

# inheritance

class Employee:

    def \_\_init\_\_(self,name,id):

        self.name = name

        self.id = id

    def showDetails(self):

        print(f"The name of Employee is {self.name} and id is {self.id}")

# child class of Employee

class Programmer(Employee):

    def showLanguage(self):

        print(f"The default language is python.")

e = Employee("okasha",123)

e.showDetails()

e2 = Programmer("Muhammad Zain Zameer",42804)

e2.showLanguage()

e2.showDetails()

Access Modifiers(public, private, protected)

First of all in python there is no such thing as public, private and protected. But uske baad bhi hum baat krte hein python mein inki.

Access modifiers comes whenever we talk about object-oriented programming.

Lets understand in general what these terms meant:

Public: access kia jaskta hai bahir se ek object ke attribute ko.

Private: bahir se access nh kiye jaskte

Protected: can be accessed from the class or from the subclass but not outside the class.

Public:

# Access Modifiers

# Public

class Employee:

    def \_\_init\_\_(self):

        self.name = "Harry"

e = Employee()

print(e.name) # accessible from outside

Private

# Private: Access nahi kiye jasakein

# we can specify double underscore

class Employee:

    def \_\_init\_\_(self):

        self.\_\_name = "Harry"

e = Employee()

print(e.\_\_name) # this will throw error

but still I can access the name attribute in python, It is only for the convention to indicate that a variable or method should be considered private by prefixing its name with a double underscore(\_). This is known as weak internal use indicator and it is a convention only, not a strict rule.

Now how can I access indirectly the private variable ?

# Private: Access nahi kiye jasakein

# we can specify double underscore

class Employee:

    def \_\_init\_\_(self):

        self.\_\_name = "Harry"

e = Employee()

print(e.\_Employee\_\_name) # name mangling

we can use object.\_ClassName\_\_attributeName to access the attribute even if it’s a private.

This is known as name mangling.

class Employee:

    def \_\_init\_\_(self):

        self.\_\_name = "Harry"

e = Employee()

print(e.\_\_dir\_\_()) # we can use this which can show also \_Employee\_\_name so you can access the private data, -> programmer accidently variable ko change na krde isiliye name mangling krdi jati hai

same as we can use double underscore for function to indicate a private function.

class Student:

    def \_\_init\_\_(self, age, name):

        self.\_\_age = age      # An indication of private variable

    def \_\_funName(self):  # An indication of private function

        self.y = 34

        print(self.y)

obj = Student(21,"Harry")

print(obj.\_Student\_\_age)

obj.\_Student\_\_funName()

Protected

In object-oriented programming (OOP), the term "protected" is used to describe a member (i.e., a method or attribute) of a class that is intended to be accessed only by the class itself and its subclasses. In Python, the convention for indicating that a member is protected is to prefix its name with a single underscore (\_). For example, if a class has a method called \_my\_method, it is indicating that the method should only be accessed by the class itself and its subclasses.

It's important to note that the single underscore is just a naming convention, and does not actually provide any protection or restrict access to the member. The syntax we follow to make any variable protected is to write variable name followed by a single underscore (\_) ie. \_varName.

class Student:

    def \_\_init\_\_(self):

        self.\_name = "Harry"

    def \_funName(self):      # protected method

        return "CodeWithHarry"

class Subject(Student):       #inherited class

    pass

obj = Student()

obj1 = Subject()

# calling by object of Student class

print(obj.\_name)

print(obj.\_funName())

# calling by object of Subject class

print(obj1.\_name)

print(obj1.\_funName())

Remember that these all are just conventions, which means that maybe someone say to you that one underscore values like (\_name) should be treated as private variables, so its just naming conventions there is no built in rules that are defined in python for them because in python there are no public, private and protected.

Double underscore lagane se python mangling krdega.

Baki sb normal hai.

Static methods

So static methods are those utility methods that are not associated with the class or instance. Like they are independent and maybe you want some complex calculation function to be shipped to the programmer that is going to use your class maybe you want to do that. So you can create static methods by using a decorater @staticmethod and now you can call it with an object or by using direcltly class name like below:

# static methods

class Math:

    def \_\_init\_\_(self,num):

        self.num = num

    def addtonum(self,n):

        self.num = self.num + n

    # utility functions: these are helper functions

    @staticmethod # not associated with class or instance

    def addZain(a,b):

        return a+b

a = Math(2)

print(a.num)

a.addtonum(3)

print(a.num)

print(a.addZain(7,2)) # calling static by object

print(Math.addZain(7,2)) # calling static by direct class name

so in interview you might get question that do we need to pass self as argument in every function defined inside the class? The answer is no.

Instance vs class variables

Just for understanding that how lets say when I call emp1.showDetails() then how it works ?

class Employee:

    def \_\_init\_\_(self,name):

        self.name = name

    def showDetails(self):

        print(f"The name of the employee is {self.name}")

emp1 = Employee("Arya")

emp1.showDetails()

# now we can see output on the screen

# we can do something similar like this

Employee.showDetails(emp1) # this is same as above emp1.showDetails()

#  emp1.showDetails() == Employee.showDetails(emp1)

# whenever we are calling showDetails function then ek argument jara hai woh hai object isiliye we write self

Now lets talk about instance vs class variables.

Instance variables woh variable hote hein joh instance se associated hote hein na key class se.

class Employee:

    def \_\_init\_\_(self,name):

        self.name = name # instance associated

        self.raiseAmount = 0.02 # instance associated

    def showDetails(self):

        print(f"The name of the employee is {self.name} and the raise amount is {self.raiseAmount}")

emp1 = Employee("Arya")

emp1.raiseAmount+=180 # changing the instance variable

emp1.showDetails() # The name of the employee is Arya and the raise amount is 180.02

emp2 = Employee("Rohan")

emp2.showDetails() # The name of the employee is Rohan and the raise amount is 0.02

Class variables woh hote joh har instance mein ap chahte ho ke same rahe like generic rahe jesa ke company name .

class Employee:

    companyName = "Goldman Sachs" # class associated / class variable

    def \_\_init\_\_(self,name):

        self.name = name # instance associated

        self.raiseAmount = 0.02 # instance associated

    def showDetails(self):

        print(f"The name of the employee is {self.name} and the raise amount in {self.companyName} is {self.raiseAmount}")

emp1 = Employee("Arya")

emp1.raiseAmount+=180 # changing the instance variable

emp1.showDetails() # The name of the employee is Arya and the raise amount in Goldman Sachs is 180.02

emp2 = Employee("Rohan")

emp2.showDetails() # The name of the employee is Rohan and the raise amount in Goldman Sachs is 0.02

class Employee:

    companyName = "Goldman Sachs" # class associated / class variable

    def \_\_init\_\_(self,name):

        self.name = name

        self.raiseAmount = 0.02

    def showDetails(self):

        print(f"The name of the employee is {self.name} and the raise amount in {self.companyName} is {self.raiseAmount}")

emp1 = Employee("Arya")

emp1.companyName = "Goldman Sachs India" # now write here we can see that the company name for arya is changed but for Rohan company name is still the same why? this is because when you wrote something like this first it will take a look at instance that is there any instance variable called "companyName" ? now we are calling emp1.companyName means we are creating a new instance variable and assigning it a value. but what if it didn't found that variable in instance variables then it will look for class variables it will start search for class variables for it

emp1.showDetails()

emp2 = Employee("Rohan")

emp2.showDetails()

# another point

# when i write something like

print(Employee.companyName) # output: Goldman Sachs

# but can i change this company name ?

Employee.companyName = "Facebook" # yes you can

# lets print rohan show details to see if companyname really changed or not?

emp2.showDetails() # yes it did changed

now when I change rohan’s company like emp2.companyName = “Google” what will happen that an instance variable will be created and it will be used in showDetails function.

Another example to demonstrate that:

class Employee:

    companyName = "Goldman Sachs"

    noOfEmployees = 0

    def \_\_init\_\_(self,name):

        self.name = name

        self.raiseAmount = 0.02

        Employee.noOfEmployees += 1

    def showDetails(self):

        print(f"The name of the employee is {self.name} and the raise amount in {self.companyName} is {self.raiseAmount}")

emp1 = Employee("Okasha")  # + 1

emp2 = Employee("Okasha2") # + 1

emp3 = Employee("Okasha3") # + 1

emp4 = Employee("Okasha4") # + 1

print(emp1.noOfEmployees) # 4 (Total 4 employees)

Class Methods

# Class Methods

class Employee:

    def show(self):

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

now that we know ke koi bhi method joh hota hai class ke andar woh instance ke liye banta hai na ke class ke liye.

# Class Methods

class Employee:

    company = "Apple"

    def show(self):

        print(f"The name is {self.name} and company is {self.company}")

    def changeCompany(cls,newCompany): # now what this function is doing is it is acting as a normal function and this function is working for the instance for now not for the class

        cls.company = newCompany

e1 = Employee()

e1.name = "Harry"

e1.show()

e1.changeCompany("Tesla") # this will change the company name for instance # not for the class

e1.show() # on instance you can see Tesla

print(Employee.company) # Apple

now if I write self,cls or harry , it doesn’t care. Yeh phele argument ko joh aya hai usko object smjhta hai or udhr changeCompany ne us instance mein instance create krdia company name ka or company set krdi. Now agr mein chahta hoon ke cls joh hai woh object ke bajaye class ko reference kre take class variable change hosake. I can use @classmethod decorator.

# Class Methods

class Employee:

    company = "Apple"

    def show(self):

        print(f"The name is {self.name} and company is {self.company}")

    @classmethod

    def changeCompany(cls,newCompany):

        cls.company = newCompany

e1 = Employee()

e1.name = "Harry"

e1.show()

e1.changeCompany("Tesla")

e1.show()

print(Employee.company) # Tesla (now company name changed)

Class methods as alternative constructors

class Employee:

    def \_\_init\_\_(self,name,salary):

        self.name = name

        self.salary = salary

e = Employee("Harry",12000)

print(e.name,e.salary)

this was our ideal scenario of constructors but lets say that user say that user will give argument in this form

class Employee:

    def \_\_init\_\_(self,name,salary):

        self.name = name

        self.salary = salary

string = "Zain-12000"

now we need to extract data from a string and pass it to constructor, so what can we do is use a split function:

class Employee:

    def \_\_init\_\_(self,name,salary):

        self.name = name

        self.salary = salary

string = "Zain-12000"

e = Employee(string.split("-")[0],string.split("-")[1])

print(e.name,e.salary)

but you are not happy after doing whole lotta circus

lets say bohot sari information extract krni humne toh hum kia bar bar split method ko call krte rahein gein kia? Code kitna ugly dikhega right

inshort what we want ke har format ke data ko handle kia jaye agr constructor mein “harry”,1200 pass horha toh bhi handle ho or agr string format mein data araha toh tab bhi handle ho

class Employee:

    def \_\_init\_\_(self,name,salary):

        self.name = name

        self.salary = salary

    @classmethod

    def fromStr(cls,str):

        return cls(string.split("-")[0],string.split("-")[1])

string = "harry-12000"

e = Employee.fromStr(string)

print(e.name,e.salary)

jitna bhi messy code tha sara ka sara class mein le jakr patak dia

class Employee:

    def \_\_init\_\_(self,name,salary):

        self.name = name

        self.salary = salary

    @classmethod

    def fromStr(cls,str):

        name,salary = str.split("-")

        # return cls(string.split("-")[0],string.split("-")[1])

        return cls(name,salary)

string = "jacob-12000"

e = Employee.fromStr(string)

print(e.name,e.salary)

another alternative ke name,salary = split krdo toh magic hojata

we can create alternative constructors using @classmethod decorator that can handle different formats of input.

Dir,dict,help

They are used for object info inspection.

Dict is an attribute, you will understand later.

The dir() method: return the list of all the attributes and methods(including dender methods).

What are dender methods you will get into that later on.

l = [1,2,3,4]

print(dir(l)) # will extract each and every method of list that can be used in it

the dict() method and help() method:

class Employee:

    companyName = "Meta"

    def \_\_init\_\_(self):

        self.name = "Zain" #default given

        self.data = "Null"

    def updateQuery(self):

        print("Updating code 0012 query: 01001")

e1 = Employee()

print(e1.\_\_dict\_\_) # will return the instance variables

print(Employee.\_\_dict\_\_) # will return all class variables

print(help(Employee)) # will tell you details about the class, what attributes are in and what functions are in

another example of dict method:

class Employee:

    companyName = "Meta"

    def \_\_init\_\_(self):

        self.name = "Zain" #default given

        self.data = "Null"

    def updateQuery(self):

        self.query = 1001

        print("Updating code 0012 query: 01001")

e1 = Employee()

e1.updateQuery()

print(e1.\_\_dict\_\_)

Super method

class parentClass:

    def parent\_method(self):

        print("This is a parent method.")

class ChildClass(parentClass):

    def parent\_method(self):

        print("This is a parent method from child class.")

    def child\_method(self):

        print("This is a child method.")

        # self.parent\_method() # instance ke pass agr parent\_method hai apna agr toh use call kro nahi toh parent class mein jao wahan se yeh method call kro

        super().parent\_method() # parent ki class ka parent\_method() call kro

child\_object = ChildClass()

child\_object.child\_method()

super() function in python is used to refer to the parent class.

We can sometime use super() for constructors, like below:  
now lets say we create a constructor for parent class which takes name and id and we create its subclass called programmer in which we also create a constructor that takes old arguments plus a language this time,

class Employee:

    def \_\_init\_\_(self,name,id):

        self.name = name

        self.id = id

class Programmer(Employee):

    def \_\_init\_\_(self,name,id,lang):

        self.name = name

        self.id = id

        self.lang = lang

rohan = Employee("Rohan Das",420)

harry = Programmer("Harry",234,"Python")

print(rohan.name, rohan.id)

so I just copy and pasted above constructor to its subclass. But lets say there are some complex string parsing going on in parent class , will I still be copying all that code or use some smart trick?

Well smart trick is down below , you can use super method

class Employee:

    def \_\_init\_\_(self,name,id):

        self.name = name

        self.id = id

class Programmer(Employee):

    def \_\_init\_\_(self,name,id,lang):

        super().\_\_init\_\_(name,id)

        self.lang = lang

rohan = Employee("Rohan Das",420)

harry = Programmer("Harry",234,"Python")

print(rohan.name, rohan.id)

print(harry.name, harry.id, harry.lang)

Magic/Dunder methods

Just like \_\_init\_\_ humne like kabhi isko likh kr call nh kia tha, hum bs object create krte the or yeh khud hi execute hojata tha. Just like len() method, we never called \_\_len\_\_ we just call len()

class Employee:

    name = "Harry"

e = Employee()

print(e.name)

print(len(e)) # it will throw an error because for this object no len function is defined

now lets define len function for this object

class Employee:

    name = "Harry"

    def \_\_len\_\_(self):

        total = 0

        for c in self.name:

            total+=1

        return total

e = Employee()

print(e.name)

print(len(e))

we defined \_\_len\_\_() but we called len() , that’s why they are known as magic methods.

Just like we don’t call \_\_init\_\_ its just automatically invoked when we create objects that’s why they known as constructors.

Now lets say I want ke agr ‘e’ ko print kroon toh uski ache se representation ho, because abhi e ko print kroonga toh object ayega samne Employee ka. So I can do something like this:

class Employee:

    def \_\_init\_\_(self,name):

        self.name = name

    def \_\_len\_\_(self):

        total = 0

        for c in self.name:

            total+=1

        return total

    def \_\_str\_\_(self):

        return (f"The name of the employee is {self.name}")

e = Employee("Harry")

print(e) # ab object nh ayega , ab ek formatedd string ayegi because humne \_\_str\_\_ method define krdia

now same as \_\_str\_\_ there is another thing called \_\_repr\_\_

class Employee:

    def \_\_init\_\_(self, name):

        self.name = name

    def \_\_len\_\_(self):

        total = 0

        for c in self.name:

            total += 1

        return total

    def \_\_str\_\_(self):

        return f"Employee Name: {self.name} (Used for print)"

    def \_\_repr\_\_(self):

        return f"Employee('{self.name}')  # Used for debugging"

e = Employee("Harry")

# first it will look for str, if no str then falls back to repr

# even though you use str() if there is no str defined then it will look for repr to run it

# Using \_\_str\_\_

print(str(e))   # Calls \_\_str\_\_, meant for users

# Using \_\_repr\_\_

print(repr(e))  # Calls \_\_repr\_\_, meant for developers

# Default behavior of print()

print(e)        # Falls back to \_\_str\_\_

# Default behavior in interactive mode (simulated)

e  # In an interactive Python shell, this would call \_\_repr\_\_

lets discuss \_\_call\_\_ method now:

class Employee:

    def \_\_init\_\_(self,name,id):

        self.name = name

        self.id = id

e = Employee("zain",42804)

e() # this will throw error that it is not callable

so we do need to define call function:

class Employee:

    def \_\_init\_\_(self,name,id):

        self.name = name

        self.id = id

    def \_\_call\_\_(self):

        print("hey im the call method.")

e = Employee("zain",42804)

e()

now it will execute.

Now lets discuss a practical use case of a \_\_call\_\_ method:

class MultiplyBy:

    def \_\_init\_\_(self, factor):

        self.factor = factor  # Store the multiplier

    def \_\_call\_\_(self, numbers):

        return [num \* self.factor for num in numbers]

# Create an instance and call it like a function

multiplier = MultiplyBy(10)

data = [1, 2, 3, 4, 5]

print(multiplier(data))  # Output: [10, 20, 30, 40, 50]

there are other dender methods also in python to discover .