What is an OOPs:

-🡪 It is about: Class, Object, Inheritance, Polymorphism, Data Abstraction and Data Encapsulation.

A screenshot of a computer

AI-generated content may be incorrect.

Why OOPs:

* To group related functions (methods)
* To create a template/blueprint
* OOP’s is a concept where characteristic and functions of a real-life object is packaged as a single entity in the code.

#!/bin/python

import os

def get\_details\_for\_each\_tomcat(server\_xml):

    global tcf,th

    tcf=server\_xml

    th=os.path.dirname(os.path.dirname(server\_xml))

    return None

def display\_details():

    print(f'The configuration file:{tcf}\n The th home:::{th}')

    return None

def main():

    tomcat7="/home/Automation/tomcat7/conf/server.xml"

    tomcat9="/home/Automation/tomcat9/conf/server.xml"

    get\_details\_for\_each\_tomcat(tomcat7)

    display\_details()

    get\_details\_for\_each\_tomcat(tomcat9)

    display\_details()

    return None

if \_\_name\_\_=="\_\_main\_\_":

    main()

Output:

$ python Automation1.py

The configuration file:/home/Automation/tomcat7/conf/server.xml

The th home:::/home/Automation/tomcat7

The configuration file:/home/Automation/tomcat9/conf/server.xml

The th home:::/home/Automation/tomcat9

Code2:

def main():

    tomcat7="/home/Automation/tomcat7/conf/server.xml"

    tomcat9="/home/Automation/tomcat9/conf/server.xml"

    get\_details\_for\_each\_tomcat(tomcat7)

    get\_details\_for\_each\_tomcat(tomcat9)

    display\_details()

    display\_details()

    return None

Output:

$ python Automation1.py

The configuration file:/home/Automation/tomcat9/conf/server.xml

The th home:::/home/Automation/tomcat9

The configuration file:/home/Automation/tomcat9/conf/server.xml

The th home:::/home/Automation/tomcat9

Code 3:

Self keyword in class

Outside no self

A screenshot of a computer

AI-generated content may be incorrect.

1. To create objects🡪tomcat7,tomcat9
2. Separate memory locations for different objects
3. Grouping related functions in one name
4. Template: Related functions to that object

#!/bin/python

import os

class Tomcat():

    def get\_details\_for\_each\_tomcat(self,server\_xml):

        self.tcf=server\_xml

        self.th=os.path.dirname(os.path.dirname(server\_xml))

        return None

    def display\_details(self):

        print(f'The configuration file:{self.tcf}\n The th home:::{self.th}')

        return None

def main():

    tomcat7=Tomcat()

    tomcat9=Tomcat()

    tomcat7.get\_details\_for\_each\_tomcat("/home/Automation/tomcat7/conf/server.xml")

    tomcat9.get\_details\_for\_each\_tomcat("/home/Automation/tomcat9/conf/server.xml")

    tomcat7.display\_details()

    tomcat9.display\_details()

    tomcat9.display\_details()

    tomcat7.display\_details()

    print(tomcat7.tcf)

    print(tomcat9.tcf)

if \_\_name\_\_=="\_\_main\_\_":

    main()

Output:

$ python Automation1.py

The configuration file:/home/Automation/tomcat7/conf/server.xml

The th home:::/home/Automation/tomcat7

The configuration file:/home/Automation/tomcat9/conf/server.xml

The th home:::/home/Automation/tomcat9

The configuration file:/home/Automation/tomcat9/conf/server.xml

The th home:::/home/Automation/tomcat9

The configuration file:/home/Automation/tomcat7/conf/server.xml

The th home:::/home/Automation/tomcat7

/home/Automation/tomcat7/conf/server.xml

Class and Object attributes:

* Class is a template/blueprint to create an object
* Class is the combination of attributes and methods
* We can define attributes for class and objects

class emp():

    def get\_name\_age\_salary(self,name,age,salary):

        self.name=name

        self.age=age

        self.salary=salary

        return None

    def display\_details(self):

        print(f'name:{self.name}\nage:{self.age}\nsalary:{self.salary}')

        return None

emp1=emp()

emp2=emp()

emp1.get\_name\_age\_salary("John",34,45000)

emp2.get\_name\_age\_salary("Ram",45,56000)

# print(dir(emp1))

# 'age', 'display\_details', 'get\_name\_age\_salary', 'name', 'salary']

print(emp1.name) # John

emp1.display\_details()

# name:John

# age:34

# salary:45000

Cod2:

* Class attributes are access inside of class 🡪 self.
* Object attributes are access inside of class or outside of class by class.variable

emp.count

class emp():

    count=0

    def get\_name\_age\_salary(self,name,age,salary):

        self.name=name

        self.age=age

        self.salary=salary

        self.get\_count\_objs()

        return None

    def get\_count\_objs(self):

        emp.count=emp.count+1

    def display\_details(self):

        print(f'name:{self.name}\nage:{self.age}\nsalary:{self.salary}')

        return None

emp1=emp()

emp2=emp()

emp1.get\_name\_age\_salary("John",34,45000)

emp2.get\_name\_age\_salary("Ram",45,56000)

# print(dir(emp1))

# 'age', 'display\_details', 'get\_name\_age\_salary', 'name', 'salary']

print(emp1.name) # John

emp1.display\_details()

# name:John

# age:34

# salary:45000

print(emp.count) #2

Output:

$ python Automation2.py

John

name:John

age:34

salary:45000

2

Constructor of class:

* A Constructor is a special method that is called by default whenever you can create an object from class

class Emp:

    def \_\_init\_\_(self):

        print("This is a init method...")

    def display(self):

        print("This is display method...")

emp1=Emp()

emp2=Emp()

# emp1.display() # This is display method...

# emp.2display() # This is display method...

OUTPUT:

# #This is a init method...

# # This is a init method...

Code 2:

class Emp:

    count=0

    def \_\_init\_\_(self):

        Emp.count=Emp.count+1

    def display(self):

        print("This is display method...")

emp1=Emp()

emp2=Emp()

print(Emp.count)  #2

Code 3:

class Emp(object):

    # def name\_salary(self,name,salary):

    #     self.name=name

    #     self.salary=salary

    #     return None

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

        self.name=name

        self.salary=salary

        return None

    def display(self):

        print(f'The name is ::{self.name}\n The salary is :: {self.salary}')

        return None

emp1=Emp("Ramu",56000)

# emp1.name\_salary("Ramu",56000)

emp2=Emp("Narendra",90000)

# emp2.name\_salary("Narendra",90000)

emp1.display()

emp2.display()

# Output:

# $ python constructor.py

# The name is ::Ramu

#  The salary is :: 56000

# The name is ::Narendra

#  The salary is :: 90000

Simple python script with /without OOPS concepts:

Destructor of a class:

class Emp(object):

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

        self.salary=salary

        self.age=age

    def \_\_del\_\_(self):

        print("Delete object")

emp1=Emp(15000,65)

Output:

Delete object

A screenshot of a computer

AI-generated content may be incorrect.

Polymorphism and Inheritance:

* Same method name in diff class called Polymorphism

A white background with blue text

AI-generated content may be incorrect.

class Tomcat(object):

    def \_\_init\_\_(self,home,version):

        self.home=home

        self.version=version

        return None

    def display(self):

        print("This is from  tomcat class.....")

        print(self.home)

        print(self.version)

        return None

class Apache(object):

    def \_\_init\_\_(self,home,version):

        self.home=home

        self.version=version

        return None

    def display(self):

        print("This is from  Apache class.....")

        print(self.home)

        print(self.version)

        return None

tomcat\_obj=Tomcat("/home/tomcat9","7.6")

apache\_obj=Apache("/etc/httpd","2.4")

tomcat\_obj.display()

apache\_obj.display()

Output:

$ python Polymorphism.py

This is from tomcat class.....

/home/tomcat9

7.6

This is from Apache class.....

/etc/httpd

2.4

Inheritance:

A screenshot of a computer

AI-generated content may be incorrect.

Save a lot of time for duplicating the code

class Tomcat(object):

    def \_\_init\_\_(self,home,version):

        self.home=home

        self.version=version

        return None

    def display(self):

        print("This is from  tomcat class.....")

        print(self.home)

        print(self.version)

        return None

class Apache(Tomcat):

    def \_\_init\_\_(self,home,version):

        self.home=home

        self.version=version

        return None

tomcat\_obj=Tomcat("/home/tomcat9","7.6")

apache\_obj=Apache("/etc/httpd","2.4")

tomcat\_obj.display()

apache\_obj.display()

Output:

$ python inheritance\_auto.py

This is from tomcat class.....

/home/tomcat9

7.6

This is from tomcat class.....

/etc/httpd

2.4

Encapsulation:

class Person(object):

    def assign(self,name,age):

        self.name=name

        self.age=age

        return None

    def display(self):

        print(self.name,self.age)

        return None

per1=Person()

per1.assign("john",21)

# #per1.display() #John , 21

# print(per1.name) # Jo hn

# print(per1.age)  # 21

Output:

$ python encapsulation.py

Traceback (most recent call last):

File "encapsulation.py", line 31, in <module>

print(per1.\_\_age)

AttributeError: 'Person' object has no attribute '\_\_age'

Code 2:

class Person(object):

    def assign(self,name,age):

        self.name=name

        self.\_\_age=age

        return None

    def \_\_display(self):

        print(self.name,self.age)

        return None

per1=Person()

per1.assign("john",21)

# print(per1.\_\_age)

print(per1.\_\_display())

Output:

$ python encapsulation.py

Traceback (most recent call last):

File "encapsulation.py", line 32, in <module>

print(per1.\_\_display())

AttributeError: 'Person' object has no attribute '\_\_display'

Code 3:

class Person(object):

    def assign(self,name,age):

        self.name=name

        self.\_\_age=age

        self.\_\_display()

        return None

    def \_\_display(self):

        print(self.name,self.\_\_age)

        return None

per1=Person()

per1.assign("john",21)

# print(per1.\_\_age)

# print(per1.\_\_display())

Output:

$ python encapsulation.py

john 21

A screenshot of a computer

AI-generated content may be incorrect.