**OBJECT ORIENTED PROGRAMMING**

**Key Principles:**

Encapsulation

Inheritance

Abstraction

Polymorphism

These 4 features are used using Classes and Objects.

**CLASSES :**

* Classes allow us to create our own data types that is user defined data types.
* They act as a blue print for our object.
* They allow us to create variable and methods for a particular object that are defined.

Eg: Class Product

Id

Name

Price

Based on the class product, we can create n no of objects.

**Class:**

Create a class in PyDev using <class name.py>

To define fields, we can do that directly inside a constructor method that will initialize those fields.

\_\_init\_\_ is a method which is inbuilt method, available for every class and it take a key word called self.

self point to the current object that is being created of this particular class.

**Code**

class **Product**:

#DEFINED THE INIT METHOD WHICH WILL CONSTRUCT AN OBJECT

def **\_\_init\_\_**(*self*):

*self*.product\_id=102

*self*.product\_name=*"LG OLED TV"*

*self*.product\_size=65

*self*.product\_price=1079.50

#CREATING AN OBJECT FOR PRODUCT

prod = Product()

# PRINT THE VALUES

print(*"PRODUCT CODE : "*, prod.product\_id)

print(*"PRODUCT MODEL : "*, prod.product\_name)

print(*"PRODUCT SIZE : "*, prod.product\_size)

print(*"PRODUCT PRICE : "*, prod.product\_price)

**Output**

PRODUCT CODE : 102

PRODUCT MODEL : LG OLED TV

PRODUCT SIZE : 65

PRODUCT PRICE : 1079.5

**CREATING PARAMETERIZED CONSTRUCTOR:**

**Code**

class **Course**:

def **\_\_init\_\_**(*self*, cname, cratings):

*self*.course\_name=cname

*self*.course\_ratings=cratings

c1=Course(*"Programming in Python"*,[1,2,3,4,5])

print(c1.course\_name)

print(c1.course\_ratings)

c2=Course(*"Programming in Java"*,[1,2,3,4,5])

print(c2.course\_name)

print(c2.course\_ratings)

**Output**

Programming in Python

[1, 2, 3, 4, 5]

Programming in Java

[1, 2, 3, 4, 5]

**DEFINING AN INSTANCE METHOD:**

**Code**

class **Course**:

def **\_\_init\_\_**(*self*, cname, cratings):

*self*.course\_name=cname

*self*.course\_ratings=cratings

def **average**(*self*):

noOfRatings=len(*self*.course\_ratings)

avgRatings=sum(*self*.course\_ratings)/noOfRatings

print(*"Average Ratings for course - "*, *self*.course\_name, *" : "*, avgRatings)

c1=Course(*"Programming in Python"*,[1,2,3,4,5])

print(c1.course\_name)

print(c1.course\_ratings)

c1.average()

c2=Course(*"Programming in Java"*,[1,2,3,4,5])

print(c2.course\_name)

print(c2.course\_ratings)

c2.average()

**Output**

Programming in Python

[1, 2, 3, 4, 5]

Average Ratings for course - Programming in Python : 3.0

Programming in Java

[1, 2, 3, 4, 5]

Average Ratings for course - Programming in Java : 3.0

**MUTATOR AND ACCESSOR METHODS:**

**Code**

class **Programmer**:

def **setfName**(*self*,nam):

*self*.efName=nam

def **getfName**(*self*):

return *self*.efName

def **setSalary**(*self*, sal):

*self*.efsalary = sal

def **getSalary**(*self*):

return *self*.efsalary

def **setTechnology**(*self*,tech):

*self*.eTech=tech

def **getTechnology**(*self*):

return *self*.eTech

p1 = Programmer()

p1.setfName(*"Kamal Kumar JesuRanjan"*)

p1.setSalary(85000)

p1.setTechnology([*"C# .Net"*, *"Java"*, *"Python"*])

print(*"Programmer Full Name : "*,p1.getfName())

print(*"Programmer Salary : "*,p1.getSalary())

print(*"Programmer Technologies : "*,p1.getTechnology())

**Output**

Programmer Full Name : Kamal Kumar JesuRanjan

Programmer Salary : 85000

Programmer Technologies : ['C# .Net', 'Java', 'Python']

**INVOKING THE INSTANCE OF A METHOD:**

**Code**

class **Product**:

def **\_\_init\_\_**(*self*):

*self*.product\_id=102

*self*.product\_name=*"LG OLED TV"*

*self*.product\_size=65

*self*.product\_price=1079.50

def **display**(*self*):

print(*"PRODUCT CODE : "*, *self*.product\_id)

print(*"PRODUCT MODEL : "*, *self*.product\_name)

print(*"PRODUCT SIZE : "*, *self*.product\_size)

print(*"PRODUCT PRICE : "*, *self*.product\_price)

#CREATING AN OBJECT FOR PRODUCT

prod = Product()

# PRINT THE VALUES BY INVOKING THE DIAPLY FUNCTION

prod.display()

**Output**

PRODUCT CODE : 102

PRODUCT MODEL : LG OLED TV

PRODUCT SIZE : 65

PRODUCT PRICE : 1079.5

**METHODS VS CONSTRUCTORS**

**METHODS CONSTRUCTORS**

Name can be any name Name must be \_\_init\_\_

Will be executed when invoked Automatically invoked by Python

using object runtime

Invoked any number of times Invoked only one per the object

Usually write the Business logic Declare and Initilise the

Variables of the class

**DEINE A STATIC METHOD:**

**Code**

class **Student**:

#majorsub IS A STATIC VARIABLE, WHICH IS NOT CHANGED

majorSub=*"Computer Science"*

def **\_\_init\_\_**(*self*, rNo, fNm, lNm):

*self*.rollNo=rNo

*self*.firstName=fNm

*self*.lastName=lNm

def **printStudent**(*self*):

print(*"Student Roll No : "*, *self*.rollNo)

print(*"Student First Name : "*, *self*.firstName)

print(*"Student Last Name : "*, *self*.lastName)

print(*"Student Major subject : "*, *self*.majorSub)

#CALLING THE INSTANCE OF Student

stud1 = Student(101,*"Kumar"*,*"Jesu"*)

stud1.printStudent()

# IF THE STATIC NAME HAS TO BE CALLED THEN WE CAN ALSO CALL THE CLASS NAME AND THE OBJECT NAMES INSTEADOF CREATING INSTANCE

print(*"Calling Directly from Class Name without creating instance : "*,Student.majorSub)

**Output**

Student Roll No : 101

Student First Name : Kumar

Student Last Name : Jesu

Student Major subject : Computer Science

Calling Directly from Class Name without creating instance : Computer Science

**DEFINE STATIC FIELD AND STATIC METNOD**

**EGS TO COUNT THE NUMBER OF INSTANCES :**

**Code**

class **ObjectCounter**:

#STATIC FIELD

noOfObjects = 0

#CONSTRUCTOR

def **\_\_init\_\_**(*self*):

ObjectCounter.noOfObjects+=1

# CREATE STATIC METHOD

*@staticmethod*

def **displayCount**():

print(*"No of Instances created : "*, ObjectCounter.noOfObjects)

# INITIALIZE 5 INSTANCES

obj1=ObjectCounter()

obj2=ObjectCounter()

obj3=ObjectCounter()

obj4=ObjectCounter()

obj5=ObjectCounter()

#PRINT THE COUNT

ObjectCounter.displayCount()

**Output**

No of Instances created : 5

**CREATE AN INNER CLASS:**

**Code**

class **Car**:

#DEFINE CONSTRUCTOR

def **\_\_init\_\_**(*self*,make,year):

*self*.carMake=make

*self*.carYear=year

def **displayCar**(*self*):

print(*"The Car Make is "*,*self*.carMake)

print(*"The Year of Car is "*,*self*.carYear)

#CREATING INNER CLASS

class **CarPrice**:

#DEFINE CONSTRUCTOR FOR INNER CLASS

def **\_\_init\_\_**(*self*,price):

*self*.carPrice=price

def **displayCarPrice**(*self*):

print(*"The Price of the car is "*,*self*.carPrice)

car1=Car(*"Audi"*,*"232"*)

carp1=car1.CarPrice(47500)

car1.displayCar()

carp1.displayCarPrice()

print(*"======================"*)

car2=Car(*"BMW"*,*"241"*)

carp2=car2.CarPrice(57800)

car2.displayCar()

carp2.displayCarPrice()

print(*"======================"*)

car3=Car(*"Volvo"*,*"242"*)

carp3=car3.CarPrice(65780)

car3.displayCar()

carp3.displayCarPrice()

**Output**

The Car Make is Audi

The Year of Car is 232

The Price of the car is 47500

======================

The Car Make is BMW

The Year of Car is 241

The Price of the car is 57800

======================

The Car Make is Volvo

The Year of Car is 242

The Price of the car is 65780

**GARBAGE COLLECTION:**

**Meaning**

In Python, there is a garbage collector that keeps running in the background as a part of Python runtime and it will clean up our objects when the program is done and when the objects are no longer used.

**GC METHODS:**

**Code**

import gc

print(gc.isenabled())

gc.disable()

print(gc.isenabled())

gc.enable()

print(gc.isenabled())

**Output**

True

False

True

**DESTRUCTORS:**

**Code**

from debugpy.\_vendored.pydevd.stubs.\_django\_manager\_body import none

class **Product**:

#DEFINE CONSTRUCTOR

def **\_\_init\_\_**(*self*):

*self*.product\_id=102

*self*.product\_name=*"LG OLED TV"*

*self*.product\_size=65

*self*.product\_price=1079.50

#DEFINE DESTRUCTOR

def **\_\_del\_\_**(*self*):

print(*"Cleaning the Memory using Destructor"*)

def **display**(*self*):

print(*"PRODUCT CODE : "*, *self*.product\_id)

print(*"PRODUCT MODEL : "*, *self*.product\_name)

print(*"PRODUCT SIZE : "*, *self*.product\_size)

print(*"PRODUCT PRICE : "*, *self*.product\_price)

#CREATING AN OBJECT FOR PRODUCT

prod = Product()

prod.display()

prod = None

prod1 = Product()

prod1.display()

prod1 = None

**Output**

PRODUCT CODE : 102

PRODUCT MODEL : LG OLED TV

PRODUCT SIZE : 65

PRODUCT PRICE : 1079.5

Cleaning the Memory using Destructor

PRODUCT CODE : 102

PRODUCT MODEL : LG OLED TV

PRODUCT SIZE : 65

PRODUCT PRICE : 1079.5

Cleaning the Memory using Destructor

**EGS: PATEIENT CLINICALS USECASE:**

**Code**

class **Patient**:

def **\_\_init\_\_**(*self*,pname,page,psex,phone):

*self*.patientName=pname

*self*.patientAge=page

*self*.patientSex=psex

*self*.patientmobile=phone

*self*.patientClinicals=[]

def **addClinicalData**(*self*,pClinical):

*self*.patientClinicals.append(pClinical)

class **Clinical**:

def **\_\_init\_\_**(*self*, compName, compValue):

*self*.testType=compName

*self*.testValue=compValue

pat1= Patient(*"Kumar"*,48,*"Male"*,*"087-7684507"*)

cl1 = Clinical(*"Blood Pressure"*,*"78/105"*)

pat1.addClinicalData(cl1)

cl2 = Clinical(*"Heartrate"*,82)

pat1.addClinicalData(cl2)

print(*"Patient Name : "*, pat1.patientName)

print(*"Patient Age : "*, pat1.patientAge)

print(*"Patient Gender : "*, pat1.patientSex)

print(*"Patient Mobile : "*, pat1.patientmobile)

print(*"======= PATIENT TEST RESULTS =========="*)

for eachClVal in pat1.patientClinicals:

print(*"Patient Test Type : "*, eachClVal.testType)

print(*"Patient Test Value : "*, eachClVal.testValue)

**Output**

Patient Name : Kumar

Patient Age : 48

Patient Gender : Male

Patient Mobile : 087-7684507

======= PATIENT TEST RESULTS ==========

Patient Test Type : Blood Pressure

Patient Test Value : 78/105

Patient Test Type : Heartrate

Patient Test Value : 82

**QUIZ**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated