**Programming Methodologies**

* **As far as application development is concerned ( in industry) there are 2 famous programming methodologies / paradigms/ techniques / approaches**

1. **Procedural approach( structured )**
2. **Object oriented approach**

**Drawbacks of procedure oriented approach**

**In procedural oriented style of application development the major drawback is data and the code that processes the data is not properly organized. It leads to following problems**

1. **Data processing is not secured**
2. **Complexity can’t be handled**

**Object Oriented Approach**

**Features of object oriented programming:**

**We have 4 standard object oriented features as far as application development is concerned**

**1) Encapsulation**

**2) Abstraction**

**3) Polymorphism**

**4) Inheritance**

1. **Encapsulation & abstraction cover the drawbacks of procedure oriented approach**
2. **Polymorphism & inheritance offer additional flexibility**

**Encapsulation: The major drawback of procedural orientation is overcome through Encapsulation. i.e., Data and code that acts upon data is properly organized through encapsulation**

**“The process of combining data and code that acts upon data into one single unit of organization so that both (data & code) are safe from outside interference is known as Encapsulation”**

**Encapsulation prevents free flow of data in application. It leads to secured data processing by the eligible code of the same application**

**Abstraction: The process of showing only the essential functionality of the entity by hiding the implementation details at that time for that purpose**

**Class**

* **Class & objects are the means through which the theoretical object oriented feature called “encapsulation” is implemented**
* **As far as user defined data type creation is concerned a class is similar to structure of ‘C’**
* **A copy of a class is an Object**
* **A class is the basis of encapsulation**
* **A Class is plan , proposal or blue print**
* **A class is a user defined data type**

**Object**

* **A physical realization of a class is known as an object.**
* **An object implements encapsulation**
* **An object is fundamental unit of data storage area in an object oriented system**

**Generally object can have 3 things**

**State:it is the data of the object. Properties & their current values put together call it as object’s state**

**Behaviour: code of the object i.e., functions of the object is nothing but its behaviour**

**Identity: it is the name of the object with which it is uniquely identified**

**How to define a Class?**

**We can define a class by using class keyword.**

**Syntax:**

**class className:**

**''' documenttation string '''**

**variables:instance variables,static and local variables**

**methods: instance methods,static methods,class methods**

**convention: says that while naming a class ... first letter should capital**

**& if class name contains more than one word, every word first letter should be capital. How ever it is not compulsory but it is a good programming practice if u follow the convention**

**Documentation string represents description of the class. Within the class doc string is always optional. We can get doc string by using the following 2 ways.**

**1) print(classname.\_\_doc\_\_)**

**2) help(classname)**

**Example:**

**class Employee:**

**''''' This is Employee class'''**

**print(Employee.\_\_doc\_\_)**

**help(Employee)**

**Within the Python class we can represent data by using variables.**

**There are 3 types of variables are allowed.**

**1) Instance Variables (Object Level Variables)**

**2) Static Variables (Class Level Variables)**

**3) Local variables (Method Level Variables)**

**Q) Which variable can be declared as instance variable?**

**Ans:**

* **if the any variable’s value varied from object to obj: then such type of variables must be declared as instance variables.**
* **For every object a separate copy of instance variables will be created.**

**Q) How to declare/create instance variables?**

**Ans: in python instance variables must be created in the following way:**

**self.instance\_variable\_name**

**Q) Where we can declare instance variable?**

**1) Inside Constructor by using self variable**

**2) Inside Instance Method by using self variable**

**3) Outside of the class by using object reference variable**

**Q) What is a method?**

**ans: any function defined inside a class is nothing but method**

**Types of Methods:**

**Inside Python class 3 types of methods are allowed**

**1) Instance Methods**

**2) Class Methods**

**3) Static Methods**

**1) Instance Methods:**

* **Inside method implementation if we are using instance variables then such type of methods are called instance methods.**
* **Syntax to create instance method:**

**def fun\_name(self):**

**body of the method**

* **By using self variable inside method we can able to access instance variables.**

**What is an Object:**

* **Pysical existence of a class is nothing but object. We can create any number of objects for a class.**
* **Syntax to Create Object: referencevariable = classname()**
* **Ex: e1=Employee()**

**What is Reference Variable?**

* **The variable which can be used to refer object is called reference variable.**
* **By using reference variable, we can access properties and methods of object.**

**Program-1:**

**Wap to create one employee , give data to the employee and display the same**

**class Employee:**

**def setdata(self):**

**self.eno=101**

**self.ename='pavan'**

**self.salary=25000**

**def display(self):**

**print("..........EMPLOYEE INFORMATION")**

**print(f"Employee Number : {self.eno}")**

**print(f"Name : {self.ename}")**

**print(f"Salary : {self.salary}")**

**e1=Employee()**

**e1.setdata()#method calling**

**e1.display()**

**output:**

**..........EMPLOYEE INFORMATION**

**Employee Number : 101**

**Name : pavan**

**Salary : 25000**

**Q) what is the drawback with our first object oriented program?**

**Ans: if we create "n" no.of objects for employee class ,unfortunately all the "n" objects can get the same state**

**Q) How to overcome the above drawback?**

**Ans: redefine the functionality of setdata() method in such a way that while calling this**

**method proper inputs must be supplied as an arguments so that even though if u create "n" no.of objects, they can have different states**

##### **Self variable in Python:**

**In the above program display,setdata method has a parameter named ‘self’. Let’s talk about it:**

1. **self is a default variable that refers to a current class object or the current instance(object) of a class.**
2. **Generally, by using self, we can initialize the instance variables inside the constructor/method**
3. **By using a self variable, we can access instance variables and instance methods.**
4. **It can also be some other name rather than ‘self’, but the most commonly used and preferred is ‘self’.**
5. **It contains the address of the current object being referred.**
6. **self is not a keyword**

##### **Q) Can we create more than one object to a class?**

**Ans: Yes, the main concept of OOPs is class acts as a template for the objects. So, any number of objects can be created for one class.**

**Program-2:**

**#wap to overcome the drawback of the first obj oriented program**

**class Employee:**

**def setdata(self,eid,name,sal):**

**self.eno=eid**

**self.ename=name**

**self.salary=sal**

**def display(self):**

**print("......... EMPLOYEE INFO")**

**print(f"EMPLOYEE NO : {self.eno}")**

**print(f"EMPLOYEE NAME : {self.ename}")**

**print(f"EMPLOYEE SALARY : {self.salary}")**

**e1=Employee()**

**e1.setdata(101,'pavan',25000)**

**e1.display()**

**e2=Employee()**

**e2.setdata(102,'kumar',89999)**

**e2.display()**

**output:**

**......... EMPLOYEE INFO**

**EMPLOYEE NO : 101**

**EMPLOYEE NAME : pavan**

**EMPLOYEE SALARY : 25000**

**......... EMPLOYEE INFO**

**EMPLOYEE NO : 102**

**EMPLOYEE NAME : kumar**

**EMPLOYEE SALARY : 89999**

**#wap to create one employee,set necessary info,display employee details**

**#implement validating the data**

**class Employee:**

**def setemp(self):**

**self.eno=eval(input("Enter employee id"))**

**while(self.eno<0):**

**print("Invalid input.... enter +ve values only")**

**self.eno=eval(input("Enter employee id"))**

**self.ename=input("Enter name : ")**

**self.sal=eval(input("enter salary : "))**

**while(self.sal<0):**

**print("Invalid input.... enter +ve values only")**

**self.sal=eval(input("enter salary : "))**

**def dispemp(self):**

**print(25\*"\*")**

**print("Employee details")**

**print(25\*"\*")**

**print(f"Employee Id : {self.eno}")**

**print(f"Name of the Employee : {self.ename}")**

**print(f"Salary of an employee : {self.sal}")**

**e1=Employee()**

**e1.setemp()**

**e1.dispemp()**

##### **Q) What are local variables?**

**Ans: Variables that are created with in a method/function**

**In the above program self,eid,name,sal will be treated as local variables to setdata() method**

##### **Q) What happens if name clash occurs between instance variable & local variable?**

**Ans: nothing will happen**

##### **Q) What is the scope of the local variable?**

**Ans: scope refers to the portion of the program where we can access a variable. Scope of the local variable is limited to the method/function in which it is created**

**Program-3:**

**calling instance method with keyword arguments  
# defining a method with default arguments  
class Employee:  
 def setdata(self,eno,salary,ename='guest'):  
 self.eno=eno  
 self.ename=ename  
 self.salary=salary  
 def display(self):  
 print("......... EMPLOYEE INFO")  
 print(f"EMPLOYEE NO : {self.eno}")  
 print(f"EMPLOYEE NAME : {self.ename}")  
 print(f"EMPLOYEE SALARY : {self.salary}")  
e1=Employee()  
e1.setdata(101,salary=25000)  
e1.display()  
e2=Employee()  
e2.setdata(ename='aaswith',eno=102,salary=8999)  
e2.display()  
  
Output:**

**......... EMPLOYEE INFO**

**EMPLOYEE NO : 101**

**EMPLOYEE NAME : guest**

**EMPLOYEE SALARY : 25000**

**......... EMPLOYEE INFO**

**EMPLOYEE NO : 102**

**EMPLOYEE NAME : aaswith**

**EMPLOYEE SALARY : 8999  
 *Program 4:***

***#wap to create a student , read the data dynamically& display the same*class Student:  
 def setstudent(self,rno,sname,total):  
 self.rno=rno  
 self.sname=sname  
 self.total=total  
 self.calGrade()  
 def calGrade(self):  
 if self.total>=900:  
 self.grade='A'  
 elif self.total>=800:  
 self.grade='B'  
 elif self.total>=700:  
 self.grade='C'  
 else:  
 self.grade='D'  
 def displaystudent(self):  
 print("STUDENT INFORMATION..........")  
 print(f"ROLL NO ={self.rno}")  
 print(f"NAME = {self.sname}")  
 print(f"TOTAL MARKS = {self.total}")  
 print(f"GRADE = {self.grade}")  
roll=eval(input("Enter Roll no : "))  
name=input("Enter name of the student : ")  
total=eval(input("Enter marks : "))  
s1=Student()  
s1.setstudent(roll,name,total)  
s1.displaystudent()  
  
Output:**

**Enter Roll no : 1**

**Enter name of the student : pavan**

**Enter marks : 568**

**STUDENT INFORMATION..........**

**ROLL NO =1**

**NAME = pavan**

**TOTAL MARKS = 568**

**GRADE = D  
  
*Program 5:***

**#scope of a local variable  
class Test:  
 def f1(self):  
 x=10  
 def f2(self):  
 y=20  
 print(f"{x} + {y} = {x+y}")  
t1=Test()  
t1.f2()**

**Output:**

**Traceback (most recent call last):**

**File "D:\advpython\localvariable.py", line 9, in <module>**

**t1.f2()**

**File "D:\advpython\localvariable.py", line 7, in f2**

**print(f"{x} + {y} = {x+y}")**

**NameError: name 'x' is not defined  
  
*Program 6:***

**'''**

**creation of multiple objects & reading the data dynamically: process must be continued as long as the user entered the choice:yes & stop immediately if user choice is "no"**

**'''**

**from time import \***

**class Book:**

**def setBook(self,title,author,pages,cost):**

**self.title=title**

**self.author=author**

**self.pages=pages**

**self.cost=cost**

**def show(self):**

**print("........ BOOK INFO:")**

**print(f"TITLE : {self.title}")**

**print(f"AUTHOR : {self.author}")**

**print(f"NO.OF PAGES : {self.pages}")**

**print(f"COST : {self.cost}")**

**sleep(5)**

**lst\_books=[]**

**while True:**

**t=input("Enter Title of the Book : ")**

**au=input("Enter Author Name : ")**

**while((np:=eval(input("Enter no.of pages : ")))<0):**

**pass**

**while((price:=eval(input("Enter cost of the book : ")))<0):**

**pass**

**b=Book()**

**b.setBook(t,au,np,price)**

**lst\_books.append(b)**

**choice=input(".... Do u want to create one more book:::Yes|no : ")**

**if choice=='no':**

**break**

**for book in lst\_books:**

**book.show()**

***Program 7:***

**#create a new data type: vehicle: make,model,cost**

**#create multiple objects for vehicle**

**# the process should be continued as long as the user choice : yes**

**class Vehicle:**

**def setvehicle(self,make,model,cost=59999):**

**self.make=make**

**self.model=model**

**self.cost=cost**

**def display(self):**

**print(25\*"\*")**

**print("VEHICLE INFORMATION...........")**

**print(25\*"\*")**

**print(f"VEHICLE MAKE: {self.make}")**

**print(f"MODEL : {self.model}")**

**print(f"COST : {self.cost}")**

**vlist=[]**

**while True:**

**vmake=input("Enter vehicle make : ")**

**vmodel=input("Enter vehicle model : ")**

**vcost=eval(input("Enter cost of the vehicle : "))**

**v1=Vehicle()**

**v1.setvehicle(vmake,vmodel,vcost)**

**vlist.append(v1)**

**while (choice:=input("R u interested to create one more vehicle yes | no : ")).lower() not in ["yes",'y','s','no','n']:**

**print("invalid response.....")**

**if choice.lower()=='no' or choice.lower()=='n':**

**break**

**for vobj in vlist:**

**vobj.display()**

**Q: How to create instance variable outside the class?**

**ans: using obj reference**

**ex:**

**e1=Employee()**

**outside the class: e1.rno=101**

**Q: How to know instance variables information of an object?**

**ans: refvariable.\_\_dict\_\_**

**ex: e1.\_\_dict\_\_**

***Program 8:***

**'''wap to demonstrate \_\_dict\_\_ attribute of an object & how to create instance variable outside the class**

**demonstrate that no.of instance variables varied from object to object**

**'''class Test1:**

**def m1(self):**

**self.x=10**

**self.y=20**

**t1=Test1()**

**print("No.of instance variables of t1 : ",len(t1.\_\_dict\_\_))**

**t1.m1()**

**print("No.of instance variables of t1 : ",len(t1.\_\_dict\_\_))**

**print(t1.\_\_dict\_\_)**

**t1.z=30**

**print("No.of instance variables of t1 : ",len(t1.\_\_dict\_\_))**

**print(t1.\_\_dict\_\_)**

**t2=Test1()**

**print("No.of instance variables of t2 : ",len(t2.\_\_dict\_\_))**

**t2.m1()**

**print("No.of instance variables of t2 : ",len(t2.\_\_dict\_\_))**

**print(t2.\_\_dict\_\_)**

**Output:**

**No.of instance variables of t1 : 0**

**No.of instance variables of t1 : 2**

**{'x': 10, 'y': 20}**

**No.of instance variables of t1 : 3**

**{'x': 10, 'y': 20, 'z': 30}**

**No.of instance variables of t2 : 0**

**No.of instance variables of t2 : 2**

**{'x': 10, 'y': 20}**

**Q: How to restrict accessing an instance variable from outside of the class?**

**Ans: Any variable declaration preceded by two underscores is by default treated as private variables & we can access them inside the class happily but we can’t access them outside the class**

**Q: Can we declare an instance method as private method?**

**Ans: yes**

***Program 9:***

**#demonstrate private method concept**

**class Test2:**

**def m1(self):**

**self.a=10**

**self.\_\_b=20 # private variable**

**def \_\_display(self): # private method**

**print(f"{self.a=}\t{self.\_\_b=}")**

**t=Test2()**

**t.m1()**

**t.\_\_display()**

**Output:**

**Traceback (most recent call last):**

**File "D:/advpython/Test2.py", line 10, in <module>**

**t.\_\_display()**

**AttributeError: 'Test2' object has no attribute '\_\_display'**

**Q: Is it allowed in python to create an instance method with the class name?**

**Ans: yes**

***Program 10:***

***#class & method with the same name***

**class Sample:**

**def Sample(self):**

**print("Instance method execution.....")**

**s1=Sample()#object creation for Sample class**

**s1.Sample()# call to the instance method Sample()**

**Output:**

**Instance method execution.....**

**Q: If we create a normal function & class with the same name what happens?**

**Ans: Python considers only the last declaration**

***Program 11:***

**#guess the output**

**def Example():**

**print("wish function execution.......")**

**class Example:**

**def m1(self):**

**print("m1 method of Example class execution....")**

**e1=Example() # Object Creation**

**print(e1) # printing the details of an object**

**Output:**

**<\_\_main\_\_.Example object at 0x000001A8D1859F70>**

***Program 12:***

**#guess the output**

**class Example:**

**def m1(self):**

**print("m1 method of Example class execution....")**

**def Example():**

**print("wish function execution.......")**

**e1=Example() # function call statement which returns None**

**print(e1)**

**Output:**

**wish function execution.......**

**None**

## Python \_\_str\_\_(): This method returns the [string](https://www.journaldev.com/14385/python-string) representation of the object. This method is called when [print()](https://www.journaldev.com/15182/python-print) or str() function is invoked on an object. Purpose of \_\_str\_\_() is to generate a printable string representation of the object that is informal in nature and human-readable in nature.

***Program 13:***

**#wap to override \_\_str\_\_ method**

**class Computer:**

**def giveinfo(self):**

**self.brand='LENOVO'**

**self.processor='CORE I5'**

**self.ram=8**

**self.hd='2 TB'**

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

**return f'Brand = {self.brand}\nProcessor ={self.processor}\nRam = {self.ram}\n Hard disk capacity = {self.hd}'**

**c1=Computer()**

**c1.giveinfo()**

**print(c1)**

**Output:**

**Brand = LENOVO**

**Processor = CORE I5**

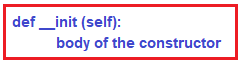
**Ram = 8**

**Hard disk capacity = 2 TB**

## ****Constructors in Python:****

1. **Constructor is a specialized method of a class**
2. **Generally, the constructor is used for creating instance variables & initial intializations that are required during the object creation.**
3. **In python, constructor is a method with the name ‘\_\_init\_\_’.**
4. **The method of the first parameter should be ‘self’ (referring to the current object).**

**Syntax:**

****

1. **In any programming language, constructor is a method which, by default, is invoked whenever an instance (object) for a class is created. It’s not required to explicitly invoke or call it.**

***Program 14:***

**class Person:**

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

##### **print("Constructor is executed automatically.....")**

##### **p1=Person()**

**Output:**

**Constructor is executed automatically.....**

**Conclusion: we just created an object for Person and haven’t called any method. But, something printed to the console, which is done by the constructor “\_\_init\_\_”. Hence, we can conclude that it is automatically called at the time of object creation**

##### **Q: Is constructor mandatory in Python?**

**No, it’s not mandatory for a class to have a constructor. It completely based on our requirement whether to have a constructor or not. At the time of object creation if any initialization is required then we should go for a constructor, else it’s not needed. Without a constructor also, the python program is valid.**

**Note: The constructor is not mandatory to be included. In case, if we haven’t included a constructor then python will, internally, include an empty constructor. We can check this by using dir(class\_name) built in method.**

***Program 15:***

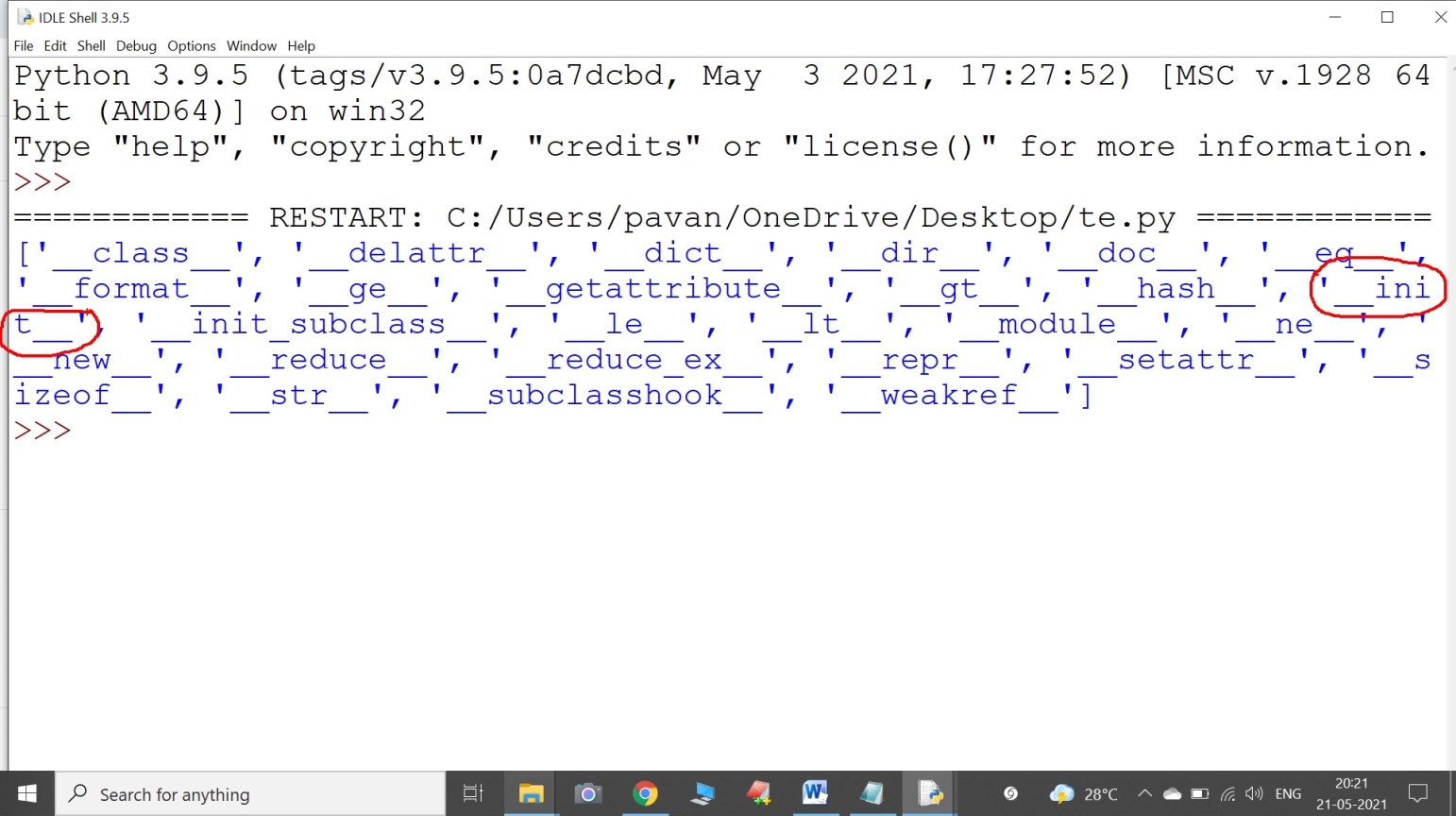
**‘’’for every python program pvm provides one constuctor if u dont define any type of constructor in your class’’’**

**class Test:**

**pass**

**print(dir(Test))**

**Output:**

****

##### **Q:Can a constructor be called explicitly?**

**Ans: Yes, we can call constructor explicitly with object name. But since the constructor gets executed automatically at the time of object creation, it is not recommended to call it explicitly.**

***Program 16:***

**#explicit call to the constructor**

**class Person:**

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

**print("Constructor is executed automatically.....")**

**def m1(self):**

**print("m1 method.....")**

**p1=Person()**

**p1.m1()**

**p1.\_\_init\_\_()**

**p1.\_\_init\_\_()**

**p1.\_\_init\_\_()**

**p1.\_\_init\_\_()**

**Output:**

**Constructor is executed automatically.....**

**m1 method.....**

**Constructor is executed automatically.....**

**Constructor is executed automatically.....**

**Constructor is executed automatically.....**

**Constructor is executed automatically.....**

**Types of constructors :**

* **Default/Non parameterized constructor :The default constructor is simple constructor which doesn’t accept any arguments.It’s definition has only one argument which is a reference to the instance being constructed.**
* **Parameterized constructor : constructor with parameters is known as parameterized constructor.The parameterized constructor take its first argument as a reference to the instance being constructed known as self and the rest of the arguments are provided by the programmer.**

##### **Q:How many parameters can constructor have?**

**Ans: Constructor can have any number of parameters depending on the requirement. There is no limit for it. All the values which we need or want to be initialized during the object creation should be passed on to the constructor**

***Program 17:***

**#non parameterized constructor**

**class Person:**

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

**self.name='pavan'**

**self.course='advaned python'**

**self.exp=17**

**self.salary=25000**

**def show(self):**

**print(f"NAME : {self.name}")**

**print(f"COURSE : {self.course}")**

**print(f"EXPERIENCE : {self.exp}")**

**print(f"SALARY : {self.salary}")**

**p1=Person()**

**p1.show()**

**Output:**

**NAME : pavan**

**COURSE : advaned python**

**EXPERIENCE : 17**

**SALARY : 25000**

***Program 18:***

**#wap on parameterized constructor**

**from time import \***

**class Course:**

**def \_\_init\_\_(self,cname,duration,fac,cost):**

**self.cname=cname**

**self.duration=duration**

**self.fac=fac**

**self.cost=cost**

**def courseinfo(self):**

**print("DETAILS OF THE COURSE...........")**

**print(f"COURSE NAME : {self.cname}")**

**print(f"DURATION : {self.duration}")**

**print(f"FACULTY: {self.fac}")**

**print(f"COST : {self.cost}")**

**sleep(5)**

**courses=[]**

**while True:**

**name=input("Enter course name : ")**

**dur=input("Duration of the course : ")**

**faculty=input("Enter faculty name : ")**

**cost=eval(input("Enter cost : "))**

**c1=Course(name,dur,faculty,cost)#calling parameterized constructor**

**courses.append(c1)**

**choice=input("Do u Want to Continue.... Yes|No : ")**

**if(choice.lower()=='no' or choice.lower()=='n'):**

**break**

**for obj in courses:**

**obj.courseinfo()**

**Output:**

**Enter course name : CORE PYTHON**

**Duration of the course : 30 DAYS**

**Enter faculty name : M.PAVAN**

**Enter cost : 500**

**Do u Want to Continue.... Yes|No : Y**

**Enter course name : ADV PYTHON**

**Duration of the course : 60 DAYS**

**Enter faculty name : M.PAVAN**

**Enter cost : 1500**

**Do u Want to Continue.... Yes|No : no**

**####################**

**DETAILS OF THE COURSE...........**

**COURSE NAME : CORE PYTHON**

**DURATION : 30 DAYS**

**FACULTY: M.PAVAN**

**COST : 500**

**####################**

**DETAILS OF THE COURSE...........**

**COURSE NAME : ADV PYTHON**

**DURATION : 60 DAYS**

**FACULTY: M.PAVAN**

**COST : 1500**

##### **Static Variables in Python: If the value of a variable is not changing from object to object, such types of variables are called static variables or class level variables. We can access static variables either by class name or by object name. Accessing static variables with class names is highly recommended than object names.**

***Program 19:***

**#STATIC VARIBLE DEMONSTRATION**

**class Student:**

**college\_name='ST.ANNS' #declaration of static variable**

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

**self.name=name**

**self.id=id**

**s1=Student('pavan', 1)**

**s2=Student('aaswith', 2)**

**print("Studen1 info:")**

**print("Name: ", s1.name)**

**print("Id : ", s1.id)**

**print("College name n : ", Student.college\_name)**

**print("\n")**

**print("Studen2 info:")**

**print("Name: ",s2.name)**

**print("Id : ",s2.id)**

**print("College name : ", s2.college\_name)**

**Output:**

**Studen1 info:**

**Name: pavan**

**Id : 1**

**College name n : ST.ANNS**

**Studen2 info:**

**Name: aaswith**

**Id : 2**

**College name : ST.ANNS**

##### **Declaring static variables in Python:**

**We can declare static variable in the following ways,**

1. **Inside class and outside of the method**
2. **Inside constructor**
3. **Inside instance method**
4. **Inside class method**
5. **Inside static method**

##### **Declaring static variable inside class and outside of the method:**

**Generally, we can declare and initialize static variable within the class and outside of the methods. This is the preferred way.**

##### **Declaring static variable inside constructor:**

**We can declare and initialize static variables within the constructor by using class’s name.**

**Program 20:**

**# Declaring static variable inside a constructor**

**class Demo:**

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

**Demo.b=20**

**d = Demo()**

**print(Demo.\_\_dict\_\_)**

**Output:**

**{'\_\_module\_\_': '\_\_main\_\_', '\_\_init\_\_': <function Demo.\_\_init\_\_ at 0x00000221B55B8E50>, '\_\_dict\_\_': <attribute '\_\_dict\_\_' of 'Demo' objects>, '\_\_weakref\_\_': <attribute '\_\_weakref\_\_' of 'Demo' objects>, '\_\_doc\_\_': None, 'b': 20}**

##### **Declaring static variable inside instance method:**

**We can declare and initialize static variable inside instance method by using class Name just as we have done in the constructor.**

##### **Declaring static variable inside class method:**

**We can declare and initialise static variable inside class method in two ways, one is using class name, other is using cls pre-defined variable**

##### **What is class method?**

**A method inside a class with @classmethod decorator is called a class method. Inside the class method we can declare and initialize static variables by using the class’s name**

**We can also do the initialization in the class method using the cls variable. cls is predefined variable in python. We should pass the cls parameter to the class methods just as we use self for instance methods.**

**Program 21:**

**#Declaring static variable inside class method**

**class Demo:**

**a=20**

**@classmethod**

**def m2(cls):**

**Demo.b=30**

**print("a = ",Demo.a)**

**print("b = ",cls.b)**

**Demo.m2()**

**print(Demo.\_\_dict\_\_)**

**Output:**

**a = 20**

**b = 30**

**{'\_\_module\_\_': '\_\_main\_\_', 'a': 20, 'm2': <classmethod object at 0x0000021EA4C89F70>, '\_\_dict\_\_': <attribute '\_\_dict\_\_' of 'Demo' objects>, '\_\_weakref\_\_': <attribute '\_\_weakref\_\_' of 'Demo' objects>, '\_\_doc\_\_': None, 'b': 30}**

##### **Static Methods in Python:**

**The static methods, in general, utility methods. Inside these methods we won’t use any instance or class variables. No arguments like cls or self are required at the time of declaration.**

1. **We can declare static method explicitly by using @staticmethod decorator.**
2. **We can access static methods by using class name or object reference**

##### **Declaring static variable inside static method:**

**We can declare and initialize static variables inside a static method by using class name**

##### **Setter and Getter methods in Python:**

**Setter methods can be used to set values to the instance variables. They are also known as mutator methods. Their main aim is to only set values to instance variables, hence they don’t return anything.**

**Syntax:**

**Setter and Getter methods in Python**

**Getter methods are used to get the values of instance variables. They return the value in the instance variable. Getter methods also known as accessor methods.**

**Syntax:**

**Types of Class Methods in Python**

**Program 22:**

**#mini banking application**

**from random import \***

**from time import \***

**from sys import \***

**class Account:**

**bank='SBI' # STATIC / CLASS VARIABLE**

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

**self.name=input("Enter name : ")**

**self.\_\_balance=eval(input("Enter initial deposit amount : "))**

**while self.\_\_balance<0:**

**print("INVALID AMOUNT")**

**self.\_\_balance=eval(input("Enter initial deposit amount : "))**

**print("Your Account number generation is under process......")**

**sleep(5)**

**self.accno=Account.generateaccnumber()**

**print("\*\*\*\*\*\*\*\*\*SUCCESSFULLY NEW ACCOUNT CREATED\*\*\*\*\*\*\*\*\*")**

**self.displayacc()**

**def displayacc(self):**

**print("\*\*\*\*\*\*\*\*\*\*ACCOUNT DETAILS\*\*\*\*\*\*\*\*\*\*")**

**print(f"BANK NAME : {Account.bank}")**

**print(f"ACCOUNT HOLDER NAME : {self.name}")**

**print(f"ACCOUNT NUMBER : {self.accno}")**

**print(f"BALANCE IN THE A/C : {self.getbalance()}")**

**sleep(5)**

**@staticmethod #optional**

**def generateaccnumber():**

**t=""**

**for i in range(11):**

**t=t+str(randint(0,9))**

**return int(t)**

**def setbalance(self,amt):**

**self.\_\_balance=amt**

**def getbalance(self):**

**return self.\_\_balance**

**def deposit(self,amt):**

**self.setbalance(self.getbalance()+amt)**

**print(".......... After deposit operation Your balance is updated to Rs ",self.getbalance())**

**def withdraw(self,amt):**

**if(self.getbalance()<amt):**

**exit("...... TRANSACTION DECLINED DUE TO INSUFFICIENT FUNDS")**

**else:**

**print("...... Please Collect Your cash of Rs : ",amt)**

**sleep(3)**

**self.setbalance(self.getbalance()-amt)**

**print("...... After withdraw operation Your balance is updated to Rs ",self.getbalance())**

**list\_accs=[]**

**while True:**

**a1=Account()**

**list\_accs.append(a1)**

**choice=input("Do u want to create one more account??Yes|No : ")**

**if choice.lower()=='no' or choice.lower()=='n':**

**break**

**choices=[1,2,3,4]**

**while True:**

**print(".............1.Print Account Information")**

**print(".............2.Deposit")**

**print(".............3.withdraw")**

**print(".............4.EXIT")**

**option=eval(input("Enter Your option : "))**

**if option not in choices:**

**exit("INVALID OPTION")**

**elif option==4:**

**exit("THANK U VISIT AGAIN")**

**else:**

**ano=eval(input("Enter Account number : "))**

**for aobj in list\_accs:**

**if aobj.accno==ano:**

**if option==1:**

**aobj.displayacc()**

**break**

**elif option==2:**

**damt=eval(input("Enter deposit amount : "))**

**while damt<0:**

**print("INVALID AMOUNT")**

**damt=eval(input("Enter deposit amount : "))**

**aobj.deposit(damt)**

**break**

**elif option==3:**

**wamt=eval(input("Enter withdraw amount : "))**

**while wamt<0 or wamt%100!=0:**

**print("INVALID AMOUNT / AMOUNT MUST BE IN MULTIPLES OF 100 ONLY")**

**wamt=eval(input("Enter withdraw amount : "))**

**aobj.withdraw(wamt)**

**break**

**else:**

**print("INVALID ACCOUNT NUMBER")**

**Output:**

**Enter name : pavan**

**Enter initial deposit amount : 1000**

**Your Account number generation is under process......**

**\*\*\*\*\*\*\*\*\*SUCCESSFULLY NEW ACCOUNT CREATED\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*ACCOUNT DETAILS\*\*\*\*\*\*\*\*\*\***

**BANK NAME : SBI**

**ACCOUNT HOLDER NAME : pavan**

**ACCOUNT NUMBER : 6248409095**

**BALANCE IN THE A/C : 1000**

**Do u want to create one more account??Yes|No : yes**

**Enter name : aaswith**

**Enter initial deposit amount : 2000**

**Your Account number generation is under process......**

**\*\*\*\*\*\*\*\*\*SUCCESSFULLY NEW ACCOUNT CREATED\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*ACCOUNT DETAILS\*\*\*\*\*\*\*\*\*\***

**BANK NAME : SBI**

**ACCOUNT HOLDER NAME : aaswith**

**ACCOUNT NUMBER : 59243946264**

**BALANCE IN THE A/C : 2000**

**Do u want to create one more account??Yes|No : n**

**========================================**

**.............1.Print Account Information**

**.............2.Deposit**

**.............3.withdraw**

**.............4.EXIT**

**Enter Your option : 2**

**Enter Account number : 6248409095**

**Enter deposit amount : -999**

**INVALID AMOUNT**

**Enter deposit amount : 100**

**.......... After deposit operation Your balance is updated to Rs 1100**

**.............1.Print Account Information**

**.............2.Deposit**

**.............3.withdraw**

**.............4.EXIT**

**Enter Your option : 3**

**Enter Account number : 59243946264**

**Enter withdraw amount : 3**

**INVALID AMOUNT / AMOUNT MUST BE IN MULTIPLES OF 100 ONLY**

**Enter withdraw amount : 1245**

**INVALID AMOUNT / AMOUNT MUST BE IN MULTIPLES OF 100 ONLY**

**Enter withdraw amount : 1200**

**...... Please Collect Your cash of Rs : 1200**

**...... After withdraw operation Your balance is updated to Rs 800**

**========================================**

**.............1.Print Account Information**

**.............2.Deposit**

**.............3.withdraw**

**.............4.EXIT**

**Enter Your option : 1**

**Enter Account number : 111123**

**INVALID ACCOUNT NUMBER**

**Q: how to access members of one class inside another class?**

**Ans: in two ways we can access members of one class inside another class**

**1) By Composition (Has-A Relationship)**

**2) By Inheritance (IS-A Relationship)**

**Program 23:**

**#how to access member(s) of one class in another class**

**class Employee:**

**def \_\_init\_\_(self,eid,ename,sal):**

**self.eid=eid**

**self.\_\_ename=ename**

**self.sal=sal**

**def display(self):**

**print("...............EMPLOYEE INFO............")**

**print(f"EMPLOYEE ID : {self.eid}")**

**print(f"EMPLOYEE NAME : {self.\_\_ename}")**

**print(f"SALARY : {self.sal}")**

**class Manager:**

**@staticmethod**

**def update\_salary(eobj):**

**eobj.sal+=10000**

**print("Based on your performance u got increment.......")**

**eobj.display()**

**e1=Employee(5789,'M.PAVAN',25000)**

**e1.display()**

**Manager.update\_salary(e1)**

**Output:**

**...............EMPLOYEE INFO............**

**EMPLOYEE ID : 5789**

**EMPLOYEE NAME : M.PAVAN**

**SALARY : 25000**

**Based on your performance u got increment.......**

**...............EMPLOYEE INFO............**

**EMPLOYEE ID : 5789**

**EMPLOYEE NAME : M.PAVAN**

**SALARY : 35000**

## Inner Class in Python:

* **A class defined in another class is known as inner class or nested class.**
* **We can make our code even more object oriented by using inner class**
* **A single object of the class can hold multiple sub-objects. We can use multiple sub-objects to give a good structure to our program.**

## Q: Why inner class?

* **For the grouping of two or more classes. Suppose we have two classes remote and battery. Every remote needs a battery but battery without remote won’t be used**
* **With the help of the inner class or nested class, we can hide the inner class from the outside world.**
* **Hence, Hiding the code is another good feature of the inner class.**
* **By using the inner class, we can easily understand the classes because the classes are closely related. We do not need to search for classes in the whole code, they all are almost together.**

**Program 24:**

**#how to create inner class**

**class Car:**

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

**print("Iam a car.... i can run on the road")**

**class Engine:**

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

**print("Hello car without engine you are no more")**

**def m1(self):**

**print("inner class instance method")**

**c1=Car()**

**e1=c1.Engine() # object for inner class... e1 is a reference to inner class object**

**e1.m1()**

**Output:**

**Iam a car.... i can run on the road**

**Hello car without engine you are no more**

**inner class instance method**

**Program 25:**

**class Body:**

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

**print("body got created")**

**self.i1=self.Head()**

**self.i2=self.i1.Brain()**

**class Head:**

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

**print("head got created")**

**class Brain:**

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

**print("brain got created")**

**o=Body()**

**Output:**

**body got created**

**head got created**

**brain got created**

##### **Garbage Collection in Python:**

1. **In old languages like C++, programmers are responsible for both creation and destruction of objects.**
2. **Usually a programmer should take very much care to create objects, and sometimes he may neglect destruction of useless objects.**
3. **Due to this negligence, at a certain point of time there may be a chance, total memory can be filled with useless objects which create memory problems and total application will be down with Out of memory error.**
4. **But in Python, internally a program will run in background always to destroy useless objects.**
5. **So, the chances of failing a Python program with memory problems are very little.**
6. **This program is nothing but Garbage Collector.**

##### **Objective:**

* **The main objective of Garbage Collector is to destroy useless objects. If an object does not have any reference variable, then that object is eligible for Garbage Collection.**
* **By default, the Garbage collector is enabled, but we can disable it based on our requirement. In order to do so, we need to import the gc module.**

##### **Methods:**

1. **gc.isenabled() – This method returns True if garbage collector is enabled**
2. **gc.disable() – This function is used to is disable the garbage collector explicitly**
3. **gc.enable() – This function is used to is enable the garbage collector explicitly**

**Program 26:**

**# gc module, enabling & disabling gc explicitly**

**import gc**

**print(gc.isenabled())**

**gc.disable()**

**print(gc.isenabled())**

**gc.enable()**

**print(gc.isenabled())**

**Output:**

**True**

**False**

**True**

##### **Destructors in Python:**

* **destructor is a method in python**
* **in python always name of the the destructor must be \_\_del\_\_()**
* **In Python, destructors are not needed as much needed in C++ because Python has a garbage collector that handles memory management automatically.**
* **Destructors are called automatically by garbage collection just before bject gets destroyed**
* **The main purpose of destructor is to perform cleanup activities, closing database connecitons etc**

**Program 27:**

**#destructor demonstration-example1**

**class Test:**

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

**print("constructor executed")**

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

**print("fulfilling the last wish of the object..... good bye for ever")**

**t1=Test()**

**del t1**

**Output:**

**constructor executed**

**fulfilling the last wish of the object..... good bye for ever**

**Program 28:**

**#destructor demonstration-example2**

**class Test:**

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

**print("constructor executed")**

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

**print("fulfilling the last wish of the object..... good bye for ever")**

**t1=Test()**

**t2=t1**

**print("program is still running.............")**

**del t1**

**print("program is over......")**

**Output:**

**constructor executed**

**program is still running.............**

**program is over......**

**Inheritance**

* **Acquiring features from existing entities**
* **It is a mechanism of creating new classes from existing class & there by acquiring properties & behavior from them**
* **Establishing parent child relationship between two classes is nothing but Inheritance**
* **The existing class is called a super class or base class or parent class.**
* **The new class is called a subclass or derived class or child class.**
* **While declaring subclass, we need to pass super class name into subclass’s parenthesis**

**Ex:**

##### **>>> class A:**

##### **pass**

##### **>>> class B(A):**

##### **pass**

##### **>>> issubclass(B,A)**

##### **True**

##### **>>> issubclass(A,object)**

##### **True**

##### **>>> issubclass(A,B)**

##### **False**

##### **>>> issubclass(B,object)**

##### **True**

##### **Advantages of Inheritance:**

1. **The main advantage of inheritance is code re-usability.**
2. **Time taken for application development will be less.**
3. **Redundancy (repetition) of the code can be reduced.**

**Key points of inheritance:**

* **Super classes are generalized classes**
* **Sub classes are specialized classes**
* **In a hierarchy of classes, higher the class the more generalized it is. Similarly lower the level of the class more specialized it becomes**

**Types of Inheritance: There are five types of inheritances, and they are as follows.**

* **Simple Inheritance (or) Single Inheritance**
  + **Multi-Level Inheritance**
  + **Hierarchical Inheritance**
* **Multiple Inheritance**
* **Hybrid Inheritance**

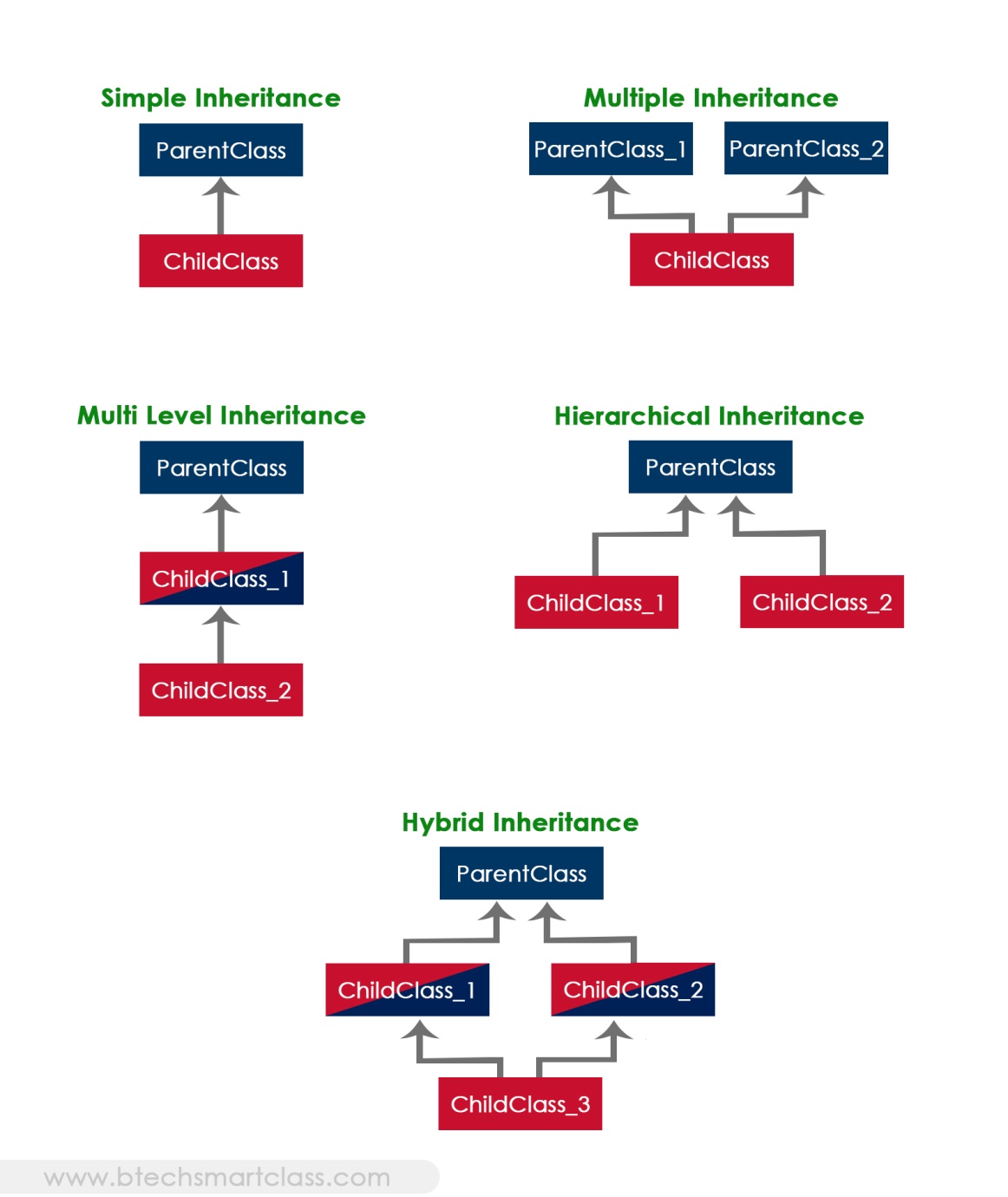
**Single Inheritance: is that form of inheritance in which a class inherits from only one parent class**

**Single inheritance has 2 special cases**

1. **Hierarchical Inheritance – Super class contains more than one sub class**
2. **Multilevel Inheritance – Here sub class acts as intermediate super class for its subclass**

**Multiple Inheritance: If a class is inheriting from more than one super class then it is known as multiple inheritance.**

**Hybrid inheritance: This form combines more than one form of inheritance. Basically, it is a blend of more than one type of inheritance.**



Q: **What is object class?**   
**Ans: Like**[**Java Object class**](https://www.geeksforgeeks.org/object-class-in-java/)**, in Python (from version 3.x), object is root of all classes.   
In Python 3.x, “class Test(object)” and “class Test” are same.**

**>>> class Test:**

**... pass**

**...**

**>>> issubclass(Test,object)**

**True**

**Program 29:**

**#wap to prove that child class will get everything from parent class**

**class Test:**

**a=10 #static variable**

**def m1(self):**

**self.b=20**

**print("Test class|instance method")**

**@classmethod**

**def m2(cls):**

**print("Test class|class method")**

**@staticmethod**

**def m3():**

**print("Test class|static method")**

**class Sample(Test):**

**pass**

**s1=Sample()**

**print(s1.a)**

**s1.m1()**

**print(s1.b)**

**s1.m2()**

**s1.m3()**

**Output:**

**10**

**Test class|instance method**

**20**

**Test class|class method**

**Test class|static method**

##### **CONSTRUCTORS in INHERITANCE:**

**By default, the super class‟s constructor will be available to the subclass.**

**Program 30:**

**#wap to prove that even constructor also inherited from parent to child**

**class P:**

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

**print("Parent class|constructor")**

**class C(P):**

**pass**

**c1=C()**

**Output:**

**Parent class|constructor**

##### **Q: If child class and super class both have constructors, then?**

**Ans: If child class and super class both have constructors, if you create an object to child class then child class constructor will be executed. While creating object for a class, that class’s constructor is first priority.**

**Program 31:**

**#If child class and super class both have constructors, then?**

**class P:**

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

**print("Parent class|constructor")**

**class C(P):**

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

**print("Child class|constructor")**

**c1=C()**

**Output:**

**Child class|constructor**

##### **Q: Can we call super class constructor from child class constructor?**

**Ans: Yes, we can call super class constructor from child class constructor by using super() function. super() is a predefined function which is useful to call the superclass constructors, variables and methods from the child class.**

**Program 32:**

**#Can we call super class constructor from child class constructor?**

**class P:**

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

**print("Parent class|constructor")**

**class C(P):**

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

**super().\_\_init\_\_()**

**print("Child class|constructor")**

**c1=C()**

**Output:**

**Parent class|constructor**

**Child class|constructor**

**Program 33:**

**#demo program on Inheritance**

**class Person:**

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

**self.name=name**

**self.age=age**

**def display(self):**

**print(f"Name : {self.name}\nAge : {self.age}")**

**class Employee(Person):**

**def \_\_init\_\_(self,name,age,eid,salary):**

**super().\_\_init\_\_(name,age)**

**self.eid=eid**

**self.salary=salary**

**def displayemp(self):**

**print("\nEMPLOYEE INFORMATION..................")**

**self.display()**

**print(f"Employee id : {self.eid}\nSalary : {self.salary}")**

**class Student(Person):**

**def \_\_init\_\_(self,name,age,total,rno):**

**super().\_\_init\_\_(name,age)**

**self.total=total**

**self.rno=rno**

**def display\_stu(self):**

**print("\nSTUDENT INFORMATION..................")**

**self.display()**

**print(f"Roll No : {self.rno}\nTotal : {self.total}")**

**e1=Employee('pavan',25,198,25000)**

**s1=Student('aaswith',12,500,3)**

**e1.displayemp()**

**s1.display\_stu()**

**Output:**

**EMPLOYEE INFORMATION..................**

**Name : pavan**

**Age : 25**

**Employee id : 198**

**Salary : 25000**

**STUDENT INFORMATION..................**

**Name : aaswith**

**Age : 12**

**Roll No : 3**

**Total : 500**

* **If the same method is inherited from both parent classes, then Python will always consider the order of Parent classes in the declaration of the child class.**

**Program 34:**

**#multiple inheritance**

**#case-1**

**class A:**

**def m1(self):**

**print("class A|m1 method")**

**class B:**

**def m1(self):**

**print("class B|m1 method")**

**class C(B,A):**

**pass**

**c1=C()**

**c1.m1()**

**Output:**

**class B|m1 method**

**Program 35:**

**#multiple inheritance**

**#case-2**

**class A:**

**def m1(self):**

**print("class A|m1 method")**

**class B:**

**def m1(self):**

**print("class B|m1 method")**

**class C(A,B):**

**pass**

**c1=C()**

**c1.m1()**

**Output:**

**class A|m1 method**

**Program 36:**

**#multiple inheritance**

**#case-3**

**class A:**

**def m1(self):**

**print("class A|m1 method")**

**class B:**

**def m1(self):**

**print("class B|m1 method")**

**class C(B,A):**

**def m1(self):**

**print("class C|m1 method")**

**c1=C()**

**c1.m1()**

**Output:**

**class C|m1 method**

**Note: in the above case the search will take place as follows.**

**Class C -> class B -> class A->class object**

## DLR Algorithm:

**Incase of multiple inheritance method resolution will search depth first and then go left to right**

## MRO Algorithm:

* **Also known as C3 algorithm**
* **It is a better version of DLR algorithm**
* **Python provides two ways to get the method resolution order of a class - \_\_mro\_\_ attribute or mro() method**

**Program 37:**

**#wap to display MRO of a class**

**class A:**

**def myname(self):**

**print(" I am a class A")**

**class B(A):**

**def myname(self):**

**print(" I am a class B")**

**class C(A):**

**def myname(self):**

**print("I am a class C")**

**class D(B, C):**

**pass**

**print(D.mro())**

**print(C.\_\_mro\_\_)**

**Output:**

**[<class '\_\_main\_\_.D'>, <class '\_\_main\_\_.B'>, <class '\_\_main\_\_.C'>, <class '\_\_main\_\_.A'>, <class 'object'>]**

**(<class '\_\_main\_\_.C'>, <class '\_\_main\_\_.A'>, <class 'object'>)**

**Q: What is the formual to find MRO of any class?**

**Ans: MRO(X) = X+Merge(MRO(P1),MRO(P2),...,ParentList)**

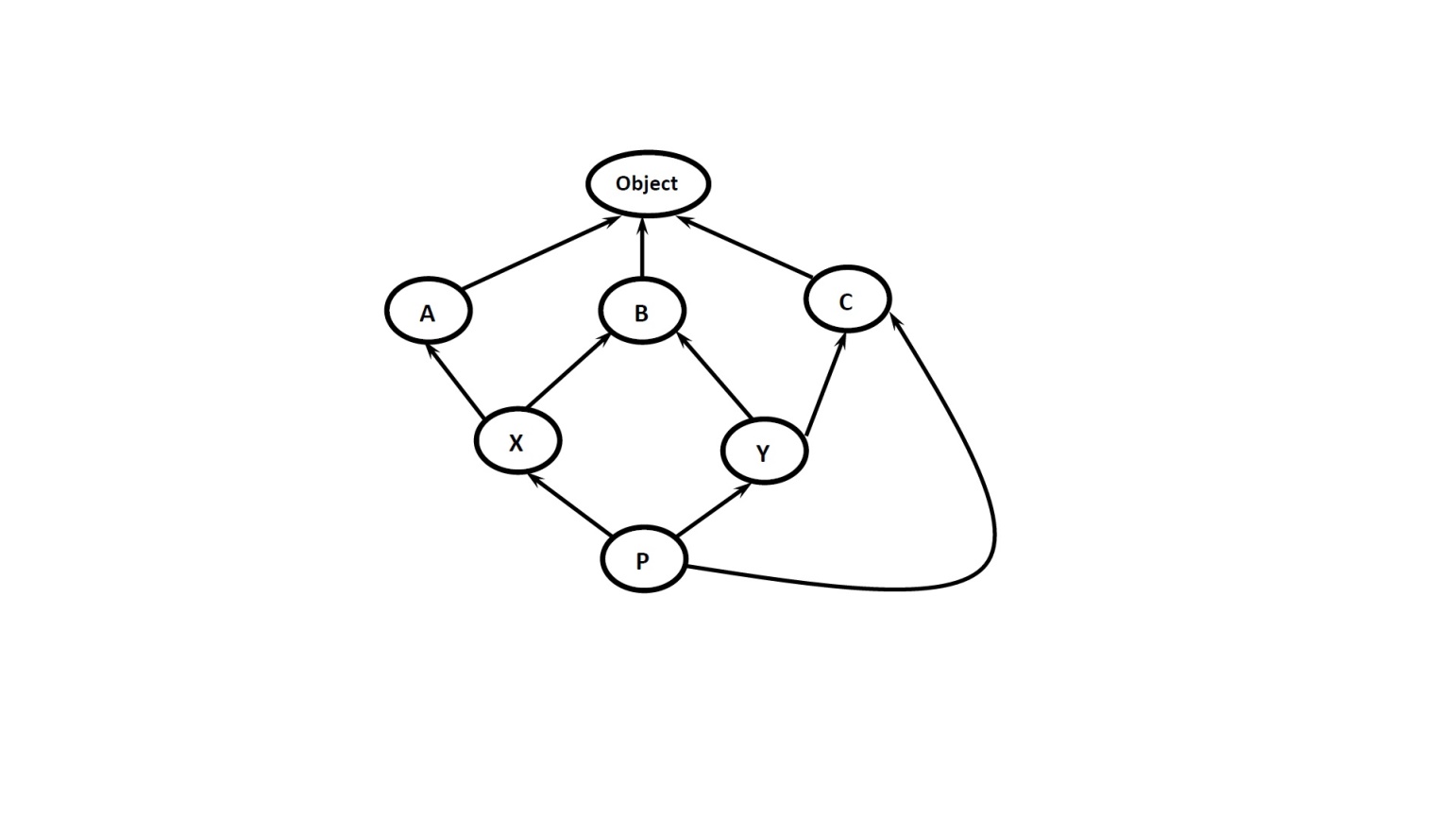
**Head Element vs Tail Terminology:**

* **Assume C1,C2,C3,...are classes.**
* **In the list: C1C2C3C4C5....**
* **C1 is considered as Head Element and remaining is considered as Tail.**

**How to find Merge:**

* **Take the head of first list**
* **If the head is not in the tail part of any other list, then add this head to the result and remove it from the lists in the merge.**
* **If the head is present in the tail part of any other list, then consider the head element of the next list and continue the same process.**

**Demo Program-1 for Method Resolution Order:**

****

**mro(A)=A,object**

**mro(B)=B,object**

**mro(C)=C,object**

**mro(X)=X,A,B,object**

**mro(Y)=Y,B,C,object**

**mro(P)=P,X,A,Y,B,C,object**

**Finding mro(P) by using C3 Algorithm:**

**Formula: MRO(X) = X+Merge(MRO(P1),MRO(P2),...,ParentList)**

**mro(p) = P+Merge(mro(X),mro(Y),mro(C),XYC)**

**= P+Merge(XABO,YBCO,CO,XYC)**

**= P+X+Merge(ABO,YBCO,CO,YC)**

**= P+X+A+Merge(BO,YBCO,CO,YC)**

**= P+X+A+Y+Merge(BO,BCO,CO,C)**

**= P+X+A+Y+B+Merge(O,CO,CO,C)**

**= P+X+A+Y+B+C+Merge(O,O,O)**

**= P+X+A+Y+B+C+O**

**Program 38:**

**#find out mro of a particular class**

**class A:pass**

**class B:pass**

**class C:pass**

**class X(A,B):pass**

**class Y(B,C):pass**

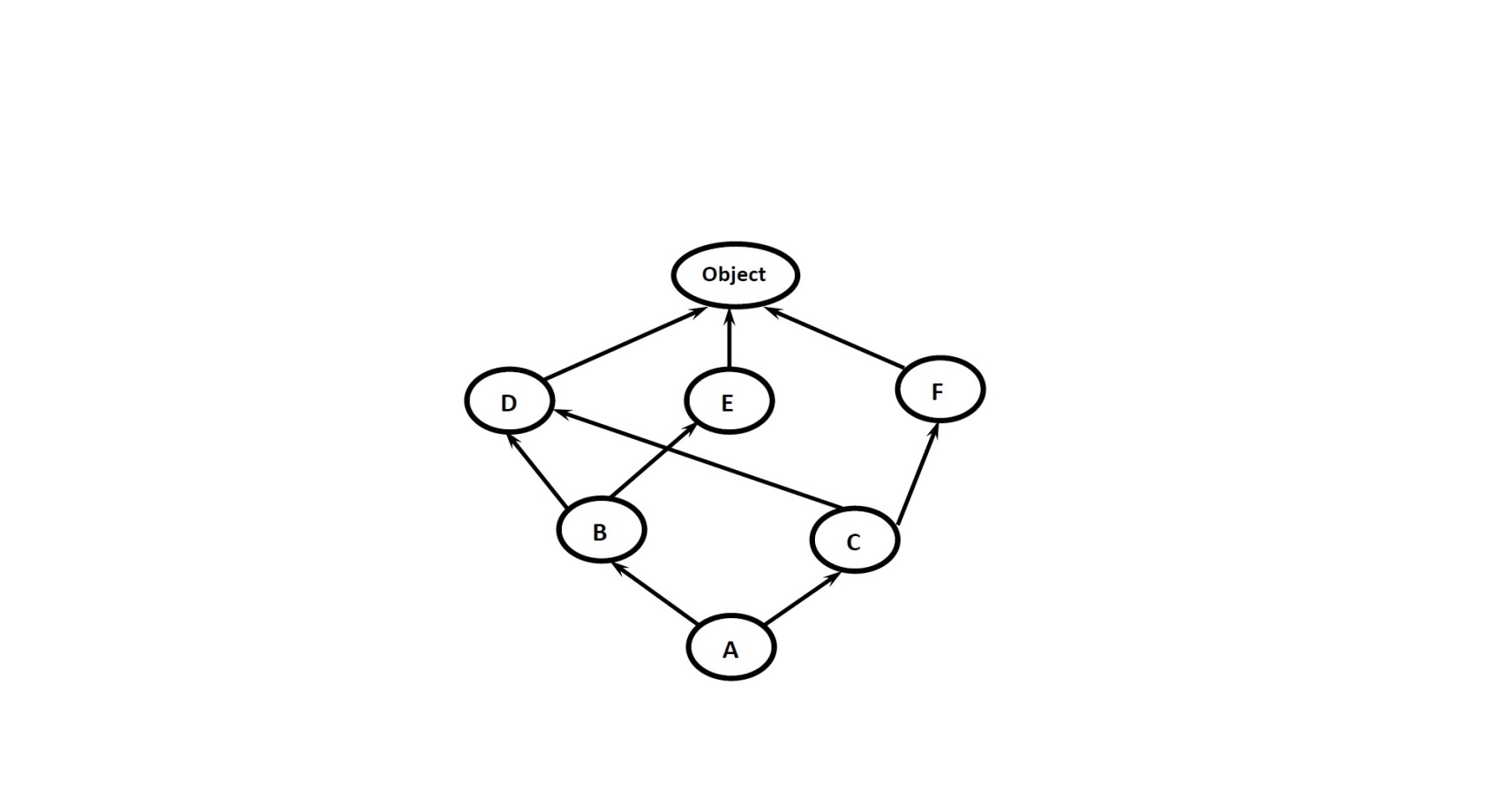
**class P(X,Y,C):pass**

**print(P.mro())**

**Output:**

**[<class '\_\_main\_\_.P'>, <class '\_\_main\_\_.X'>, <class '\_\_main\_\_.A'>, <class '\_\_main\_\_.Y'>, <class '\_\_main\_\_.B'>, <class '\_\_main\_\_.C'>, <class 'object'>]**

**Demo Program-2 for Method Resolution Order:**

****

**mro(o) = object**

**mro(D) = D,object**

**mro(E) = E,object**

**mro(F) = F,object**

**mro(B) = B,D,E,object**

**mro(C) = C,D,F,object**

**mro(A) = A+Merge(mro(B),mro(C),BC)**

**= A+Merge(BDEO,CDFO,BC)**

**= A+B+Merge(DEO,CDFO,C)**

**= A+B+C+Merge(DEO,DFO)**

**= A+B+C+D+Merge(EO,FO)**

**= A+B+C+D+E+Merge(O,FO)**

**= A+B+C+D+E+F+Merge(O,O)**

**= A+B+C+D+E+F+O**

**Program 39:**

**#find out mro of A**

**class D: pass**

**class E:pass**

**class F:pass**

**class B(D,E):pass**

**class C(D,F):pass**

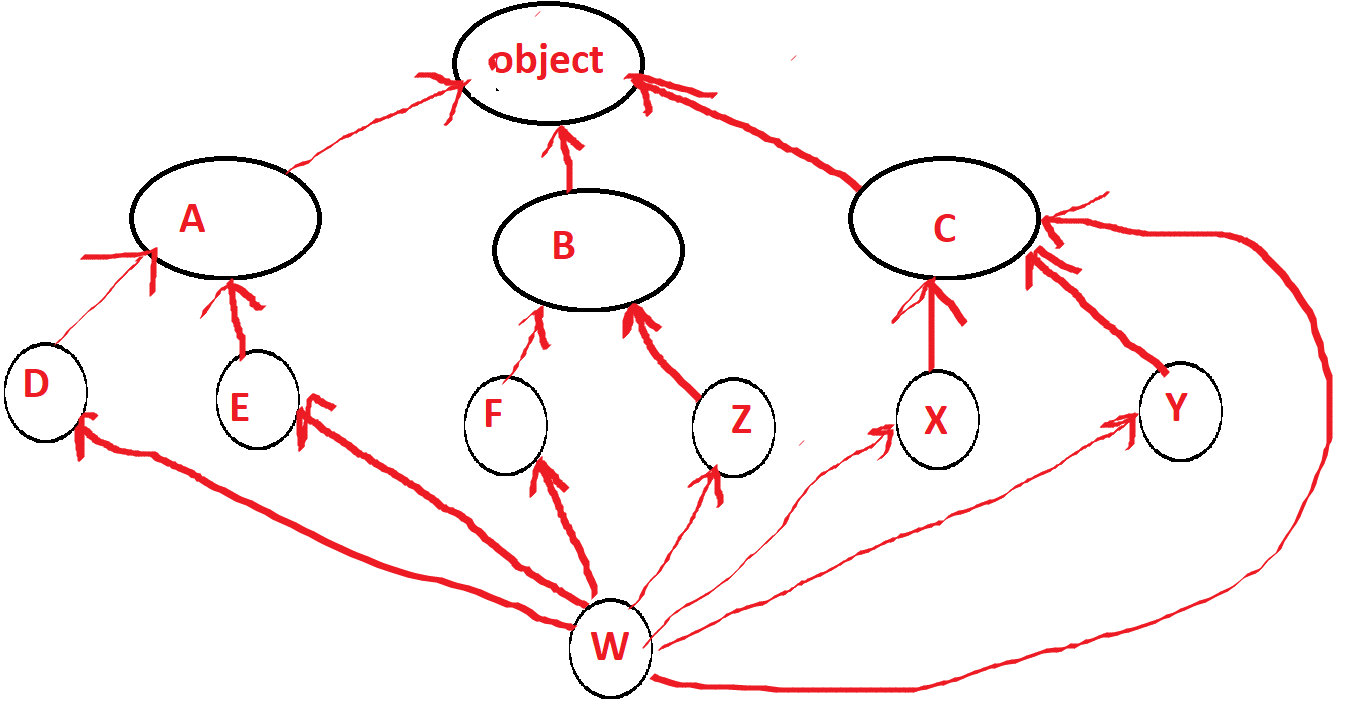
**class A(B,C):pass**

**print(A.\_\_mro\_\_)**

**Output:**

**(<class '\_\_main\_\_.A'>, <class '\_\_main\_\_.B'>, <class '\_\_main\_\_.C'>, <class '\_\_main\_\_.D'>, <class '\_\_main\_\_.E'>, <class '\_\_main\_\_.F'>, <class 'object'>)**

**Demo Program-3 for Method Resolution Order:**

****

**Program 40:**

**#Find out MRO of W**

**class A:pass**

**class B:pass**

**class C:pass**

**class D(A):pass**

**class E(A):pass**

**class F(B):pass**

**class Z(B):pass**

**class X(C):pass**

**class Y(C):pass**

**class W(D,E,F,Z,X,Y):pass**

**print(W.\_\_mro\_\_)**

**Output:**

**(<class '\_\_main\_\_.W'>, <class '\_\_main\_\_.D'>, <class '\_\_main\_\_.E'>, <class '\_\_main\_\_.A'>, <class '\_\_main\_\_.F'>, <class '\_\_main\_\_.Z'>, <class '\_\_main\_\_.B'>, <class '\_\_main\_\_.X'>, <class '\_\_main\_\_.Y'>, <class '\_\_main\_\_.C'>, <class 'object'>)**

##### **Super() Function in Python:**

**super() is a predefined function in python. By using super() function in child class, we can call,**

1. **Super class constructor.**
2. **Super class variables.**
3. **Super class methods.**

##### **Q: Which scenarios super() function is required?**

**Ans: When both superclass and child class may have the same method names, same variable names, some scenarios may come where we want to use both of them. In such a case, using the super() function we can call the parent class method.**

**Program 41:**

**#how to access parent class method from child class using super() function**

**class Test:**

**def marry(self):**

**print("You must marry subbamma:::::",id(self))**

**class Child(Test):**

**def marry(self):**

**print("i will marry only krithika setty:::",id(self))**

**super().marry()**

**c1=Child()**

**c1.marry()**

**Output:**

**i will marry only krithika setty::: 2697740798848**

**You must marry subbamma::::: 2697740798848**

**Polymorphism in Python: An entity behaving differently in different situations is**

**Known as polymorphism**

**• One interface multiple forms is the key feature of polymorphism**

**• Polymorphism offers extensibility of code**

**• If offers flexibility in application development**

##### **Duck Typing Philosophy of Python:**

### In duck typing, while creating a data, it’s not required to declare the argument type explicitly. At runtime, based on provided value the type will be considered automatically. Since Python is considered as a Dynamically Typed Programming Language, it follows Duck Typing.

**“If it walks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck.”**

**Duck Typing originates from the above saying.**

**Program 42:**

**#wap to demonstrate duck typing philosophy in python**

**class Pycharm:**

**def execute(self):**

**print("...............Executing the program with Pycharm")**

**print("compiling")**

**print("linking")**

**class MyEditor:**

**def execute(self):**

**print("...............Executing the program with MyEditor")**

**print("spell checking....")**

**print("internal verification.....")**

**print("Compiling")**

**print("linking")**

**class Laptop:**

**def code(self,ide):**

**ide.execute()**

**lap1=Laptop()**

**ide=Pycharm()**

**lap1.code(ide)**

**ide=MyEditor()**

**lap1.code(ide))**

**Output:**

**...............Executing the program with Pycharm**

**compiling**

**linking**

**...............Executing the program with MyEditor**

**spell checking....**

**internal verification.....**

**Compiling**

**Linking**

## Operator Overloading:

* **Operator overloading in Python is the ability of a single operator to perform more than one operation based on the class (type) of operands**
* **For example, the + operator can be used to add two numbers, concatenate two strings or merge two lists. This is possible because the + operator is overloaded with int and str classes.**

**>>> 10+20**

**30**

**>>> int.\_\_add\_\_(10,20)**

**30**

**>>>**

* **In Python, when any operator is used, a special function is internally invoked by the compiler for that particular operator.**
* **Python methods that have double underscores before and after their names are called Magic methods or Special functions**.
* **By changing this magic methods code, we can extend the functionality of the operator**.

## Magic methods in Python

|  |  |  |
| --- | --- | --- |
| **OPERATOR** | **EXPRESSION** | **MAGIC METHOD** |
| **Addition** | **b1 + b2** | **\_\_add\_\_()** |
| **Subtraction** | **b1 – b2** | **\_\_sub\_\_()** |
| **Multiplication** | **b1 \* b2** | **\_\_mul\_\_()** |
| **Division** | **b1 / b2** | **\_\_truediv\_\_()** |
| **Power** | **b1 \*\* b2** | **\_\_pow\_\_()** |
| **Floor division** | **b1 // b2** | **\_\_floordiv\_\_()** |
| **Modulo operator** | **b1 % b2** | **\_\_mod\_\_()** |
| **Bitwise left shift** | **b1 << b2** | **\_\_lshift\_\_()** |
| **Bitwise right shift** | **b1 >> b2** | **\_\_rshift\_\_()** |
| **Bitwise NOT** | **~b1** | **\_\_invert\_\_()** |
| **Bitwise AND** | **b1 & b2** | **\_\_and\_\_()** |
| **Bitwise OR** | **b1 | b2** | **\_\_or\_\_()** |
| **Bitwise XOR** | **b1 ^ b2** | **\_\_xor\_\_()** |
| **Less than** | **b1 < b2** | **\_\_lt\_\_()** |
| **Less than equal to** | **b1 <= b2** | **\_\_le\_\_()** |
| **Greater than** | **b1 > b2** | **\_\_gt\_\_()** |
| **Greater than equal to** | **b1 >= b2** | **\_\_ge\_\_()** |
| **Equal to** | **b1 == b2** | **\_\_eq\_\_()** |
| **Not equal to** | **b1 !=  b2** | **\_\_ne\_\_()** |

**Program 43:**

**#wap to demonstrate operator overloading**

**class Book:**

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

**self.cost=cost**

**def \_\_add\_\_(first,second):**

**return first.cost+second.cost**

**b1=Book(250)**

**b2=Book(120)**

**print("Cost of the two books is Rs: ",b1+b2)**

**Output:**

**Cost of the two books is Rs: 370**

**Program 44:**

**#wap to overload the following operators: + , \* , <**

**class Employee:**

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

**self.name=name**

**self.salary=salary**

**def \_\_mul\_\_(e1,t1):**

**return e1.salary\*t1.months**

**def \_\_add\_\_(emp1,emp2):**

**return emp1.salary+emp2.salary**

**def \_\_lt\_\_(emp1,emp2):**

**return emp1.salary<emp2.salary**

**class TimeSheet:**

**def \_\_init\_\_(self,name,months):**

**self.name=name**

**self.months=months**

**e1=Employee('pavan',12789)**

**e2=Employee('kumar',5000)**

**t1=TimeSheet('pavan',5)**

**print("Salary to be paid by the company : ",e1\*t1)**

**print("Salary of two employees : Rs : ",e1+e2)**

**print("Salary of employee1 < Salary of employee2????",e1>e2)**

**Output:**

**Salary to be paid by the company : 63945**

**Salary of two employees : Rs : 17789**

**Salary of employee1 < Salary of employee2???? True**

**Program 45:**

**#wap how to add b1+b2+b3 where b1,b2,b3 are Book class objects**

**class Book:**

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

**self.cost=cost**

**def \_\_add\_\_(first,second):**

**return Book(first.cost+second.cost)**

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

**return f"Cost of All Books Rs {self.cost}"**

**b1=Book(250)**

**b2=Book(120)**

**b3=Book(1500)**

**print(b1+b2+b3)**

**Output:**

**Cost of All Books Rs 1870**

##### **Method overloading in Python:**

* **If 2 methods have the same name but different types of arguments, then those methods are said to be overloaded methods.**
* **But in Python Method overloading is not possible. If we are trying to declare multiple methods with the same name and different number of arguments, then Python will always consider only the last method.**

**Program 46:**

**#wap demonstrate that python consider only the last method,if u define more than one method with same name**

**class Demo:**

**def m1(self):**

**print('no-arg method')**

**def m1(self, a):**

**print('one-arg method')**

**def m1(self, a, b):**

**print('two-arg method')**

**d= Demo()**

**d.m1()**

**d.m1(10)**

**d.m1(10,20)**

**Output:**

**Traceback (most recent call last):**

**File "D:\advpython\inheritance\demo.py", line 11, in <module>**

**d.m1()**

**TypeError: m1() missing 2 required positional arguments: 'a' and 'b'**

##### Q: **How we can handle overloaded method requirements in Python??**

**Ans: Most of the time, if a method with a variable number of arguments is required then we can handle it with default arguments or with a variable length of argument methods.**

**Program 47:**

**#How we can handle overloaded method requirements in Python??**

**#with variable lenght of arguments**

**class Demo:**

**def m1(self,\*vals):**

**print(f"Method is called with {len(vals)} no.of arguments")**

**sum=0**

**for ele in vals:**

**sum+=ele**

**print("Sum of the values {}".format(sum))**

**d= Demo()**

**d.m1()**

**d.m1(10)**

**d.m1(10,20)**

**Output:**

**Method is called with 0 no.of arguments**

**Sum of the values 0**

**Method is called with 1 no.of arguments**

**Sum of the values 10**

**Method is called with 2 no.of arguments**

**Sum of the values 30**

**Program 48:**

**#How we can handle overloaded method requirements in Python??**

**#with default arguments**

**class Demo:**

**def m1(self,x=None,y=None,z=None):**

**if x!=None and y!=None and z!=None:**

**print("Method called with 3 arguments")**

**print("Result : ",x+y+z)**

**elif x!=None and y!=None:**

**print("Method called with 2 arguments")**

**print("Result : ",x+y)**

**else:**

**print("Method called with only one argument")**

**print("Please provide 2 or 3 arguments")**

**d= Demo()**

**d.m1(10)**

**d.m1(10,20)**

**d.m1(10,20,30)**

**Output:**

**Method called with only one argument**

**Please provide 2 or 3 arguments**

**Method called with 2 arguments**

**Result : 30**

**Method called with 3 arguments**

**Result : 60**

##### **Constructor Overloading in Python:**

* **Constructor overloading is also not possible in Python. If we define multiple constructors, only the last constructor will be considered.**
* **But based on our requirement we can declare constructor with default arguments and variable length arguments.**

##### **Overriding in Python:**

* **Redefining the functionality of parent given method within the class child class**
* **All the members available in the parent class, those are by-default available to the child class through inheritance.**
* **If the child class is not satisfied with parent class implementation, then child class is allowed to redefine that method in the child class based on its requirement**
* **This concept is called overriding. Overriding concept applicable for both methods and constructors.**

# Abstract in Python:

##### **Types of Methods in Python:**

**Based on the implementation the methods can be divided into two types:**

1. **Implemented methods.**
2. **Un-implemented method.**

###### **Implemented methods:**

**A method which has a both method name and method body, that method is called an implemented method. They are also called concrete methods or non-abstract methods.**

###### **Un-implemented methods:**

**A method which has only method name and no method body, that method is called an unimplemented method. They are also called as non-concrete or abstract methods.**

**Abstract method:**

* **Body less/definition less/unimplemented methods are treated as abstract methods**
* **Abstract methods are meaningful in the context of inheritance**
* **Abstract methods, in python, are declared by using @abstractmethod decorator.**
* **@abstractmethod decorator presents in abc module. We should import the abc module in order to use the decorator.**
* **Since abstract method is an unimplemented method, we need to put a pass statement, else it will result in error.**

**Ex:**

**@abstractmethod**

**def m1(self):**

**pass**

* **For abstract methods, implementation must be provided in the subclass of abstract class.**

##### **Abstract Classes in Python:**

* **Every abstract class in Python should be derived from the ABC class which is present in the abc module**
* **Abstract class can contain Constructors, Variables, abstract methods, non-abstract methods, and Subclass.**
* **Abstract class can have zero abstract methods also**
* **If an abstract class contains atleast one abstract method we can’t instantiate the object**
* **If child class doesn’t implement parent class given abstract methods then for child class also we can’t create the object**
* **If any class is inheriting ABC class, and that class doesn’t contain an abstract method, then happily we can create an object to that class.**
* **If any class is not inheriting ABC class, then we can create an object for that class even though it contains an abstract method.**

**Program 49:**

**#wap to demonstarte object creation not possible for abstract class if it contains atleast one abstract method**

**from abc import \***

**class Test(ABC):**

**@abstractmethod**

**def m1(self):**

**pass**

**t1=Test()**

**Output:**

**Traceback (most recent call last):**

**File "D:\advpython\abstract\first.py", line 7, in <module>**

**t1=Test()**

**TypeError: Can't instantiate abstract class Test with abstract method m1**

**Program 50:**

**#demonstrate abstract class & method**

**from abc import \***

**class Database(ABC):**

**@abstractmethod**

**def connect(self):**

**pass**

**class Oracle(Database):**

**def connect(self):**

**print("got connection to oracle database")**

**obj=Oracle()**

**obj.connect()**

**Output:**

**got connection to oracle database**

##### **Q: Can abstract classes contain more subclasses?**

**Ans: Yes, an abstract class can contain more than one subclass. If different child classes require different kinds of implementations, in that case an abstract class can contain more than one subclass**

## ****Interfaces in Python:****

* **Imagine there is an abstract class which contains only abstract methods and doesn’t contain any concrete methods, such classes are called Interfaces.**
* **Therefore, an interface is nothing but an abstract class which can contains only abstract methods.**
* **In python there is no separate keyword to create an interface**
* **Interface can contain:**
  + **Constructors**
  + **Variables**
  + **abstract methods**
  + **sub class**
* **Object creation is not possible for the interface class (abstract class).**
* **As we know, abstract methods should be implemented in the subclass of interface (Abstract Class). We can create objects for the child class of interface to access implemented methods**.

**Program 51:**

**#demonstrate interface concept**

**from abc import \***

**class DBInterface(ABC):**

**@abstractmethod**

**def connect(self):**

**pass**

**@abstractmethod**

**def disconnect(self):**

**pass**

**class Oracle(DBInterface):**

**def connect(self):**

**print('Connecting to Oracle Database...')**

**def disconnect(self):**

**print('Disconnecting to Oracle Database...')**

**class Sybase(DBInterface):**

**def connect(self):**

**print('Connecting to Sybase Database...')**

**def disconnect(self):**

**print('Disconnecting to Sybase Database...')**

**dbname=input('Enter Database Name either Oracle or Sybase:')**

**classname=globals()[dbname]**

**x=classname()**

**x.connect()**

**x.disconnect()**

**Output:**

**Enter Database Name either Oracle or Sybase:Oracle**

**Connecting to Oracle Database...**

**Disconnecting to Oracle Database...**

**Note:** The globals()[str] is a predefined function, it will convert the string ‘str’ into a class name and returns the classname.

##### **Q: When should we go for interfaces?**

**Ans: Since, Interfaces will not contain implemented methods, when we don‟t know anything about implementation of requirements, then we should go for interfaces.**

##### **Q: When should we go for abstract class?**

**Ans: An abstract class is a class which can contains few implemented methods and few unimplemented methods as well. When we know about requirements partially, but not completely, then we should go for abstract class.**

##### **Q: When should we go for concrete class?**

**Ans: Concrete class is a class which is fully implemented. It contains only implemented methods. When we know complete implementation about requirements, then we should go for concrete class.**

**Exception Handling in Python**

* **In application development & execution we encounter 2 kinds of errors**

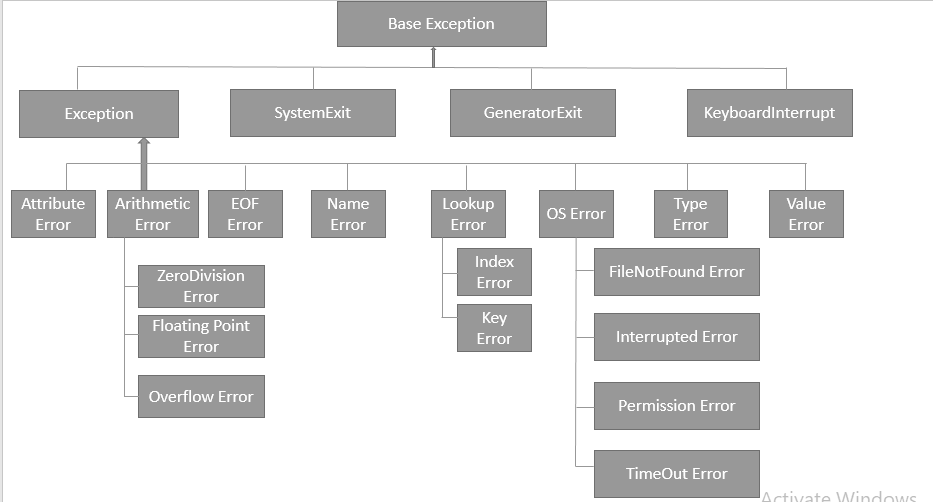
**1) Compilation errors 2) Runtime errors**

* **If the application is violating that compile time rules that is syntactically if the application is   
   wrong compiler reports the error**
* **During program execution if runtime environment rules are violated exceptions are raised by the PVM**
* **An Exception is a runtime error**
* **An exception is an object in python that represents an abnormal event occurred during the program execution**
* **Exceptions are dangerous as far as java application execution is concerned**
* **When an exception is raised, program gets terminated abnormally**
* **If a program is terminated abnormally, the following problems occur**
* **End user loses previous work. No chance of saving it**
* **End user does not get any clue about what went wrong**
* **Resources given to the application are not submitted back gracefully**
* **The process of dealing with exceptions & thereby preventing dangers caused by exceptions is known as exception handling**
* **Exception makes the application robust.**
* **To implement exception handling in python we have 2 kinds of support**

**1) language support 2) API support**

* **As far as language support is concerned we have following keywords**
  1. **try 2) except 3) raise 4) finally**
* **As far as API support is concerned we have standard exception classes given in python**

**Hierarchy of Exception classes**



##### **Default Exception Handing in Python:**

* **In python, for every exception type, a corresponding class is available and every exception is an object to its corresponding class.**
* **Whenever an exception occurs, Python Virtual Machine (PVM) will create the corresponding exception object and will check for handling code.**
* **If handling code is not available, then Python interpreter terminates the program abnormally and prints corresponding exception information to the console. The rest of the program won’t be executed.**

**Program 52:**

**#wap to demonstrate default exception handling in python**

**fp=open(fname:=input("Enter file name : "),"r+")**

**print("FILE OPENED SUCCESSFULLY")**

**fp.close()**

**Output:**

**Enter file name : hello.xlx**

**Traceback (most recent call last):**

**File "D:\practice\python\exceptions\four.py", line 2, in <module>**

**fp=open(fname:=input("Enter file name : "),"r+")**

**FileNotFoundError: [Errno 2] No such file or directory: 'hello.xlx**

**Program 53:**

**#wap to demonstrate default exception handling in python**

**x=eval(input("Enter numerator value : "))**

**y=eval(input("Enter denominator value : "))**

**print("Result of division : ",x/y)**

**print("division is over")**

**Output:**

**Enter numerator value : 10**

**Enter denominator value : 0**

**Traceback (most recent call last):**

**File "D:\practice\python\exceptions\second.py", line 4, in <module>**

**print("Result of division : ",x/y)**

**ZeroDivisionError: division by zero**

**Program 54:**

**#wap to read two int value from the command line & display the result of division**

**from sys import \***

**x=int(argv[1])**

**y=int(argv[2])**

**print("Result of division : ",x/y)**

**print("division is over")**

**Output:**

**D:\practice\python\exceptions>py third.py 10 5**

**Result of division : 2.0**

**division is over**

**D:\practice\python\exceptions>py third.py 10**

**Traceback (most recent call last):**

**File "D:\practice\python\exceptions\third.py", line 4, in <module>**

**y=int(argv[2])**

**IndexError: list index out of range**

##### **Q:How to Handle Exception in Python?**

**Ans: Using try except statements we can handle exception in python.**

**try keyword:**

* **try is one of the 4 keywords of exception handling mechanism**
* **“try” is used to create a block of statements**
* **In “try” block we place doubtful code of our application that may raise exceptions**

**try:**

**statement-1**

**statement-2**

**statement-3**

**Note: The code in the try block is purely the logic of the application. If that code doesn’t cause any runtime error, try has no influence in the application execution**

* **If exception is raised in the try block, instead of program getting terminated try block is terminated**

**except:**

* + **except block should be preceded by another except block or try block**
  + **Between try & except block no statement is allowed**
  + **One try block can have any no. of except blocks. However at any moment only one except block gets executed**
  + **During program execution if abnormal event is generated PVM calibrates the kind of abnormality and encapsulates the abnormality i.e., it creates the obj of corresponding exception**
  + **Now PVM throws that object. If appropriate except block is available then code in the except block will be executed**
  + **If exception is not generated in the “try” block control never comes to except block**

**Program 55:**

**#wap to demonstrate how to implement exception handling using try,except keyword**

**try:**

**fp=open(fname:=input("Enter file name : "),"r+")**

**print("FILE OPENED SUCCESSFULLY")**

**fp.close()**

**except FileNotFoundError:**

**print("File doesnt exist with the specified name")**

**print("Please provide valid name")**

**Output  
Enter file name : abx.xyz**

**File doesnt exist with the specified name**

**Please provide valid name**

**Program 56:**

**#wap to demonstrate try except mechanism**

**#wap to read two values dynamically , print result of the division**

**try:**

**x=eval(input("Enter numerator value : "))**

**y=eval(input("Enter denominator value : "))**

**print("Result of division : ",x/y)**

**print("division is over")**

**except ZeroDivisionError:**

**print("Denominator should not be zero")**

**Output:**

**Enter numerator value : 25**

**Enter denominator value : 0**

**Denominator should not be zero**

**Control Flow in try-except:**

**statement-1**

**statement-2**

**try:**

**statement-3**

**statement-4**

**statement-5**

**statement-6**

**except:**

**statement-7**

**statement-8**

**statement-9**

**statement-10**

**Case-1: if there is no exception**

**Statement : 1,2,3,4,5,6,9,10 executed & normal termination**

**Case -2: exception raised statement-1**

**Abnormal termination**

**Case -3: exception raised statement-2**

**Statement-1 executed & then abnormal termination**

**Case -4: exception raised statement-3 and corresponding except block not found**

**Statement -1,2 executed & then abnormal termination**

**Case -5: exception raised statement-3 and corresponding except block found**

**Statement -1,2,7,8,9,10 executed & then abnormal termination**

##### **Printing Exception Information in Python:**

* **In the above examples, in the exception handling code, we are printing our custom message**
* **Rather than that, we can also print exception information by creating a reference to the exception**

**Program 57:**

**#wap to demonstrate how to print exception information**

**try:**

**print(13/0)**

**except ZeroDivisionError as msg:**

**print(f"Exceptin Information :::: {msg}")**

**Output:**

**Exceptin Information :::: division by zero**

##### **try with multiple except Blocks in python:**

* **In python, try with multiple except blocks are allowed.**
* **Sometimes, there may be the possibility that a piece of code can raise different exceptions in different cases.**
* **In such cases, in order to handle all the exceptions, we can use multiple except blocks.**
* **So, for every exception type a separate except block we have to write.**

**Program 58:**

**#wap to demonstrate try with multiple except blocks**

**try:**

**from sys import \***

**x=int(argv[1])**

**y=int(argv[2])**

**print("Result of division : ",x/y)**

**print("division is over")**

**except ZeroDivisionError:**

**print("second argument shouldnot be zero")**

**except ValueError:**

**print("Enter integers only")**

**except IndexError:**

**print("supply both the arguments")**

**Output:**

**D:\practice\python\exceptions>py seven.py 10**

**supply both the arguments**

**D:\practice\python\exceptions>py seven.py 10 0**

**second argument shouldnot be zero**

**D:\practice\python\exceptions>py seven.py 10 two**

**Enter integers only**

##### **ONE except BLOCK – MULTIPLE EXCEPTIONS:**

* **Rather than writing different except blocks for handling different exceptions, we can handle all of them in one except block.**
* **The only thing is we will not have the flexibility of customizing the message for each exception.**
* **Rather, we can take the reference to the exception and print its information**

**Syntax:**

**except (Exception1, Exception2, Exception3……):**

**or**

**except (Exception1, Exception2, Exception3……) as msg:**

**Parenthesis are mandatory, and this group of exceptions is internally considered as tuple.**

**Program 59:**

**#wap to demonstrate one exception block-multiple exceptions**

**try:**

**from sys import \***

**x=int(argv[1])**

**y=int(argv[2])**

**print("Result of division : ",x/y)**

**print("division is over")**

**except (ZeroDivisionError,ValueError,IndexError) as msg:**

**print("Exception information:::",msg)**

**Output:**

**D:\practice\python\exceptions>py eight.py 10**

**Exception information::: list index out of range**

**D:\practice\python\exceptions>py eight.py 10 two**

**Exception information::: invalid literal for int() with base 10: 'two'**

**D:\practice\python\exceptions>py eight.py 10 0**

**Exception information::: division by zero**

##### **Default except Block in Python:**

* **We can use default except block to handle any type of exceptions.**
* **It’s not required to mention any exception type for the default block.**
* **Whenever we don’t have any idea of what expectation the code could raise, then we can go for default except block.**
* **We know that we can have multiple except blocks for a single try block.**
* **If the default except block is one among those multiple ones, then it should be at last, else we get SyntaxError.**

**Program 60:**

**#wap to demonstrate default except block**

**try:**

**from sys import \***

**x=int(argv[1])**

**y=int(argv[2])**

**print("Result of division : ",x/y)**

**print("division is over")**

**except:**

**print("Please Provide valid input")**

**Output:**

**D:\practice\python\exceptions>py nine.py 10**

**Please Provide valid input**

**D:\practice\python\exceptions>py nine.py 10 0**

**Please Provide valid input**

**D:\practice\python\exceptions>py nine.py five two**

**Please Provide valid input**

**Program 61:**

**# wap to demonstrate if program contains multiple except blocks & default except is one among them , program raises syntax error if default except is the first one**

**try:**

**from sys import \***

**x=int(argv[1])**

**y=int(argv[2])**

**print("Result of division : ",x/y)**

**print("division is over")**

**except:**

**print("Please Provide valid input")**

**except ZeroDivisionError:**

**print("2nd argument should not be zero")**

**Output:**

**D:\practice\python\exceptions>py ten.py**

**File "D:\practice\python\exceptions\ten.py", line 7**

**print("division is over")**

**^**

**SyntaxError: default 'except:' must be last**

**finally keyword:**

* **This keyword is used to create a block of statements .**
* **The nature of finally block is that its code is guaranteed for execution in exception generated case or non generated case**
* **Therefore in the finally block we write resources releasing code**

**For Ex: closing the files & closing the database connections**

##### **Q:Why not ‘try except’ block for clean-up activities?**

**try block: There is no guarantee like every statement will be executed inside the try block. If an exception is raised at the second line of code in the try block at, then the remaining lines of code in that block will not execute. So, it is not recommended to write clean up code inside the try block.**

**except block: There is no guarantee that an except block will execute. If there is no exception then except block won’t be executed. Hence, it is also not recommended to write clean up code inside except block**

##### **else Block in Python:**

* **We can use else blocks with try-except-finally blocks.**
* **The else block will be executed if and only if there are no exceptions inside the try block.**
* **If an exception is raised inside the try block, then the except block will get executed but the else block won’t.**

**Program 62:**

**#wap to demonstrate finally & else block**

**try:**

**fp=None**

**fp=open(fname:=input("Enter file name : "),"r")**

**print("Content of the file..........\n",fp.read())**

**except FileNotFoundError:**

**print("File Doesnt exist with the specified name")**

**except:**

**print("During File I/O operations some thing went wrong")**

**else:**

**print("All operations on File completed successfully")**

**finally:**

**if fp!=None:**

**fp.close()**

**print("File closed.......??? ",fp.closed)**

**Output:**

**D:\practice\python\exceptions>py 12.py**

**Enter file name : test.txt**

**Content of the file..........**

**hi iam learning adv python course**

**All operations on File completed successfully**

**File closed.......??? True**

**Userdefined/Custom Exceptions**

* **Our own created exceptions are known as userdefined exceptions**

**Use of userdefined exceptions:**

* **Inorder to develop userdefined exception our own class must be a child of builin Exception class**

**Ex:**

**Class VotingException (Exception):**

**pass**

* **Sometimes the input given to the application doesn’t violate runtime rules. Therefore PVM doesn’t raise any exception**
* **But that input violates our business rules.**
* **Whenever business rule is violated we want to halt the processing instantly**
* **The best way to achieve this creating our own exception & raise it**
* **We can raise a custom exception by using the keyword ‘raise’.**

**Ex:**

**raise VotingException**

**Program 63:**

**#wap to demonstrate custom/userdefined exception**

**class VotingException(Exception):**

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

**print("Ineligible To Vote")**

**try:**

**age=eval(input("Enter Age : "))**

**if age<18:**

**raise VotingException()**

**else:**

**print("Eligible To Vote")**

**except VotingException:**

**print("Please enter age >=18 only")**

**Output:**

**Enter Age : 15**

**Ineligible To Vote**

**Please enter age >=18 only**

**Enter Age : 18**

**Eligible To Vote**

##### **Different control flow cases of try except finally in python:**

**try:**

**stmt-1**

**stmt-2**

**stmt-3**

**except:**

**stmt-4**

**finally:**

**stmt-5**

**stmt-6**

**Case-1: If there is no exception**

**1,2,3,5,6 Normal Termination**

**Conclusion: If there is no exception, then try and finally blocks will execute and except block won’t execute, leading to normal termination**.

**Case-2: If an exception raised at stmt2 and the corresponding except block matched**

**1,4,5,6 Normal Termination**

**Conclusion: If an exception is raised inside the try block and the except block is handling that corresponding exception then try, except and finally blocks will execute, leading to normal termination.**

**Case-3: If an exception raised at stmt2 but the corresponding except block not matched**

**1,5,6, Abnormal Termination**

**Conclusion: If an exception is raised inside the try block and the except block is not handling that corresponding exception then try and finally blocks will execute, leading to abnormal termination.**

**Case-4: If an exception raised at stmt4 then it is always abnormal termination but before that finally block will be executed.**

**Conclusion: If an exception is raised inside the try block, then flow goes to the except block. And in the except block, assume, the particular exception is handled. Instead of handling exceptions, if the block itself raises another exception then the program will terminate abnormally. In this case also, before the program termination finally block will be executed.**

**Case-5: If an exception raised at stmt-5 or at stmt-6 then it is always abnormal termination**

**Conclusion: If an exception is raised inside the finally block, then it’s always abnormal termination.**

**Various possible Combinations of try-except-else-finally:**

**1) Whenever we are writing try block, compulsory we should write except or finally block.i.e without except or finally block we cannot write try block.**

**2) Wheneever we are writing except block, compulsory we should write try block. i.e except without try is always invalid.**

**3) Whenever we are writing finally block, compulsory we should write try block. i.e finally without try is always invalid.**

**4) We can write multiple except blocks for the same try,but we cannot write multiple finally blocks for the same try**

**5) Whenever we are writing else block compulsory except block should be there. i.e without except we cannot write else block.**

**6) In try-except-else-finally order is important.**

**7) We can define try-except-else-finally inside try, except, else and finally blocks. i.e nesting of try-except-else-finally is always possible.**

**Iterable:**

* **Iterable is anything you can loop over with a for loop.**
* **An object is called an iterable if u can get an iterator out of it.**
* **Calling iter() function on an iterable gives us an iterator.**

**Iterator:**

* **An iterator is an object representing a stream of data.**
* **It returns the data one element at a time.**
* **Iterables supports only iter() function.But iterators supports both iter() and next() function.**
* **Calling next() function on iterator gives us the next element**
* **If the iterator is exhausted(if it has no more elements), calling next() raises StopIteration exception.**
* **Iterators are also iterables but not vice versa**

# Limits of Iterator:

* **We can only go forward in an iterator.**
* **We can’t make a copy of it.**
* **No way to get the previous element.**
* **We can’t reset the iterator.**

**Program 64:**

**# How to make object of our own class as iterator object**

**# wap to generator account numbers as long as account number doesnt start with "0"**

**from random import \***

**from time import \***

**class AccNumbers:**

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

**return self**

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

**accno=''**

**for i in range(11):**

**accno+=str(randint(0,9))**

**if(not accno.startswith("0")):**

**return accno**

**else:**

**raise StopIteration**

**obj=AccNumbers()**

**i=1**

**for ele in obj:**

**print(f"{i}) {ele}")**

**i+=1**

**sleep(2)**

**Output:**

**1) 98111425889**

**2) 16852798877**

**3) 96302608778**

**4) 74848405874**

**5) 49742531456**

**6) 58892956151**

**Program 65:**

**#wap to generate 5 random mail ids  
# demonstrate how make object of our own class as iterator  
from random import \*  
from time import \*  
class MailIds:  
 def \_\_init\_\_(self):  
 self.data='abcdefghikjklmnopqrstuvwxyz'  
 self.count=0  
 def \_\_iter\_\_(self):  
 return self  
 def \_\_next\_\_(self):  
 temp=''  
 for i in range(10):  
 temp+=choice(self.data)  
 self.count+=1  
 if(self.count<=5):  
 return temp+"@gmail.com"  
 else:  
 raise StopIteration  
obj=MailIds()  
i=1  
for ele in obj:  
 print(f"{i}) {ele}")  
 i+=1  
 sleep(3)**

**Output:**

**1) qmpxiftmfk@gmail.com**

**2) iednptkbsx@gmail.com**

**3) pszksvvzwl@gmail.com**

**4) jonezbzznn@gmail.com**

**5)** [**tbxzyqwouc@gmail.com**](mailto:tbxzyqwouc@gmail.com)

**Regular Expression in Python**

* **The Regex or Regular Expression is a way to define a pattern for searching or manipulating strings. We can use a regular expression to match, search, replace, and manipulate inside textual data.**

##### **Application areas of Regular Expressions:**

1. **To develop validation frameworks/validation logic.**
2. **To develop Pattern matching applications (ctrl-f in windows, grep in UNIX, etc)**
3. **To develop Translators like compilers, interpreters, etc**
4. **To develop digital circuits**
5. **To develop communication protocols like TCP/IP, UDP, etc**

## The re module:

* **RE module, a built-in Python module that provides all the required functionality needed for handling patterns and regular expressions.**

## Python regex methods:

**The Python regex module consists of multiple methods, below is the list of regex methods and their meaning**.

| **Method** | **Description** |
| --- | --- |
| [**re.compile('pattern')**](https://pynative.com/python-regex-compile/) | **Compile a regular expression pattern provided as a string into a re.Pattern object.** |
| [**re.search(pattern, str)**](https://pynative.com/python-regex-search/) | **Search for occurrences of the regex pattern inside the target string and return only the first match.** |
| [**re.match(pattern, str)**](https://pynative.com/python-regex-pattern-matching/) | **Try to match the regex pattern at the start of the string. It returns a match only if the pattern is located at the beginning of the string.** |
| [**re.fullmatch(pattern, str)**](https://pynative.com/python-regex-pattern-matching/#h-re-fullmatch) | **Match the regular expression pattern to the entire string from the first to the last character.** |
| [**re.findall(pattern, str)**](https://pynative.com/python-regex-findall-finditer/) | **Scans the regex pattern through the entire string and returns all matches.** |
| [**re.finditer(pattern, str)**](https://pynative.com/python-regex-findall-finditer/#h-finditer-method) | **Scans the regex pattern through the entire string and returns an iterator yielding match objects.** |
| [**re.split(pattern, str)**](https://pynative.com/python-regex-split/) | **It breaks a string into a list of matches as per the given regular expression pattern.** |
| [**re.sub(pattern, replacement, str)**](https://pynative.com/python-regex-replace-re-sub/) | **Replace one or more occurrences of a pattern in the string with a replacement.** |
| [**re.subn(pattern, replacement, str)**](https://pynative.com/python-regex-replace-re-sub/#h-re-s-subn-method) | **Same as re.sub(). The difference is it will return a tuple of two elements. First, a new string after all replacement, and second the number of replacements it has made.** |

**Program 66:**

**#wap to demonstrate regular expression concept**

**import re**

**msg='pavan sir can handle python course & pavan can handle c,c++ not only that pavan can handle java also'**

**p1=re.compile("pavan")**

**matches=p1.finditer(msg)**

**print(type(matches))**

**for match in matches:**

**print("match found at index : ",match.start(),"& ends at index : ", match.end(),"matched string : ",match.group())**

**Output:**

**<class 'callable\_iterator'>**

**match found at index : 0 & ends at index : 5 matched string : pavan**

**match found at index : 37 & ends at index : 42 matched string : pavan**

**match found at index : 74 & ends at index : 79 matched string : pavan**

###### **Note:**

1. **start(): This method when called on the match object will returns the start index of the match.**
2. **end(): This method when called on the match object will return end+1 index of the match.**
3. **group(): This method will return the matched string**

**Note: Instead of creating a pattern object and then searching for a pattern, we can directly pass patterns as arguments to finditer () function.**

**Ex:**

**matches=re.finditer("pavan",msg)**

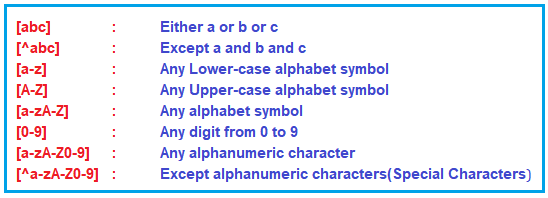
## Regex Metacharacters:

* **We can use both the special and ordinary characters inside a regular expression.**
* **For example, Most ordinary characters, like 'A', 'p', are the simplest regular expressions; they match themselves.**
* **You can concatenate ordinary characters, so the “pavan” pattern matches the string 'pavan'.**
* **Apart from this we also have special characters. For example, characters like '|', '+', or '\*', are special.**
* **Special metacharacters don’t match themselves. Instead, they indicate that some rules**
* **Special characters affect how the regular expressions around them are interpreted.**

| **Metacharacter** | **Description** |
| --- | --- |
| **. (DOT)** | **Matches any character except a newline.** |
| **^ (Caret)** | **Matches pattern only at the start of the string.** |
| **$ (Dollar)** | **Matches pattern at the end of the string.** |
| **\* (asterisk)** | **Matches 0 or more repetitions of the regex.** |
| **+ (Plus)** | **Match 1 or more repetitions of the regex.** |
| **? (Question mark)** | **Match 0 or 1 repetition of the regex.** |
| **[] (Square brackets)** | **Used to indicate a set of characters. Matches any single character in brackets. For example, [abc] will match either a, or, b, or c character.** |
| **| (Pipe)** | **used to specify multiple patterns. For example, P1|P2, where P1 and P2 are two different regexes.** |
| **\**  **(backslash)** | **Use to escape special characters or signals a special sequence. For example, If you are searching for one of the special characters you can use \ to escape special meaning for them.** |
| **[^...]** | **Matches any single character that is not in brackets.** |
| **(...)** | **Matches whatever regular expression is inside the parentheses. For example, (abc) will match to substring 'abc'** |

## Character class:

* **A "character class", or a "character set", is a set of characters put in square brackets.**
* **The regex engine matches only one out of several characters in the character class or character set.**
* **We place the characters we want to match between square brackets.**
* **For example if you want to match any vowel, we use the character set [aeiou].**
* **A character class or character set matches only a single character. The order of the characters inside a character class or set does not matter. The results are identical.**
* **We use a hyphen inside a character class to specify a range of characters**



##### **Predefined Character classes in Python:**

* **The following special sequences have a pre-defined meaning and make specific common patterns more comfortable to use.**

| **Special Sequence** | **Meaning** |
| --- | --- |
| [**\d**](https://pynative.com/python-regex-special-sequences-and-character-classes/#h-special-sequence-d-and-d) | **Matches to any digit. Short for character classes [0-9].** |
| **\D** | **Matches to any non-digit. short for [^0-9].** |
| **\s** | **Matches any whitespace character. short for character class [ \t\n\v\r\f].** |
| **\S** | **Matches any non-whitespace character. Short for [^\t\n\v\r\f].** |
| **\w** | **Matches any alphanumeric character. Short for character class [a-zA-Z0-9].** |
| **\W** | **Matches any non-alphanumeric character. Short for [^a-zA-Z0-9]** |

## Regex Quantifiers:

## We use quantifiers to define quantities.

## A quantifier is a metacharacter that determines how often a preceding regex can occur. you can use it to specify how many times a regex can repeat/occur.

**For example, We use metacharacter \*, +, ? and {} to define quantifiers.**

| **Quantifier** | **Meaning** |
| --- | --- |
| **\*** | **Match 0 or more repetitions of the preceding regex. For example, a\* matches any string that contains zero or more occurrences of 'a'.** |
| **+** | **Match 1 or more repetitions of the preceding regex. For example, a+ matches any string that contains at least one a, i.e., a, aa, aaa, or any number of a's.** |
| **?** | **Match 0 or 1 repetition of the preceding regex. For example, a? matches any string that contains zero or one occurrence of a.** |
| **{m}** | **Matches only m copies of the preceding regex. For example, p{3} matches exactly three 'p' characters, but not four.** |
| **{m, n}** | **Match m to n repetitions of the preceding regex. For example, a{m,n} matches m number of a’s and maximum n number of a’s** |

**Program 67:**

**#demonstrate search() method**

**import re**

**msg=''' yesterday i went for long drive in car without licence. Police stopped me & asked for license. then i replied them i didnt carry my license '''**

**match=re.search("licence|license",msg)**

**print(match)**

**Output:**

**<re.Match object; span=(48, 55), match='licence'>**

**Program 68:**

**#write an expression for 10/11/12 digit mobile number**

**#7013088104 - 10 digit**

**#07013088104 - 11**

**# 917013088104 -12**

**import re**

**mobno=input("Enter 10 or 11 or 12 digit mobile number : ")**

**match=re.fullmatch("[6-9]\d{9}|0[6-9]\d{9}|91[6-9]\d{9}",mobno)**

**if match!=None:**

**print("it is a valid india mobile number")**

**else:**

**print("Invalid mobile number")**

**Output:**

**Enter 10 or 11 or 12 digit mobile number: 917013088104**

**it is a valid india mobile number**

**Note: in the above example RE(Regular Expression) can be written like the following also**

**RE for 10/11/12 digit mobile number: "[6-9]\d{9}|0[6-9]\d{9}|91[6-9]\d{9}"**

**Program 69:**

**#$ is a meta character, but if u want to treat it as normal literal it should be preceded by \ in writing RE**

**import re**

**match=re.search("\$15","book cost is $15")#regex engine**

**print(match)**

**Output:**

**<re.Match object; span=(13, 16), match='$15'>**

**Program 70:**

**#demonstrate findall() method**

**import re**

**msg=''' yesterday i went for long drive in car without licence. Police stopped me & asked for license.**

**then i replied them i didnt carry my license '''**

**match1=re.findall("licence|license",msg)#return type of findall is list of matched strings**

**print(match1)**

**match2=re.findall("licen[cs]e",msg)**

**print(match2)**

**Output:**

**['licence', 'license', 'license']**

**['licence', 'license', 'license']**

**Program 71:**

**#demonstrate meta character $**

**import re**

**pattern=re.compile("pavan$",flags=re.I)**

**match1=pattern.search("CORE & ADV PYTHON BOTH BATCHES HANDLED BY PAVAN")**

**print(match1)**

**match2=re.search("pavan[0-9]+$","CORE & ADV PYTHON BOTH BATCHES HANDLED BY pavan123")**

**print(match2)**

**Output:**

**<re.Match object; span=(42, 47), match='PAVAN'>**

**<re.Match object; span=(42, 50), match='pavan123'>**

**Program 72:**

**#demonstrate sub() & subn() methods**

**import re**

**txt="""pavan.sacet@gmail.com,**

**clangbypavan@gmail.com,**

**javabypavan@gmail.com,**

**pythonbypavan@hotmail.com,**

**salesforcebypavan@yahoo.co.in"""**

**print(25\*"#")**

**s1=re.sub("@gmail.com","@jntuk.edu.in",txt)**

**print(s1)**

**print(25\*"#")**

**s2=re.subn("@gmail.com","@yahoo.co.in",txt)**

**print(type(s2))**

**print("Modified string : ",s2[0])**

**print("No.of substitutions : ",s2[1])**

**print(25\*"#")**

**Output:**

**#########################**

**pavan.sacet@jntuk.edu.in,**

**clangbypavan@jntuk.edu.in,**

**javabypavan@jntuk.edu.in,**

**pythonbypavan@hotmail.com,**

**salesforcebypavan@yahoo.co.in**

**#########################**

**<class 'tuple'>**

**Modified string : pavan.sacet@yahoo.co.in,**

**clangbypavan@yahoo.co.in,**

**javabypavan@yahoo.co.in,**

**pythonbypavan@hotmail.com,**

**salesforcebypavan@yahoo.co.in**

**No.of substitutions : 3**

**Program 73:**

**#wap to check car registration number is valid or not as per INDIAN standards**

**#AP,TS,TN,KA,OR : FIRST TWO LETTERS MUST BE ALAPHABETS**

**# AFTER STATE REPRESENTATION, 2 DIGITS ARE COMPULSORY**

**#AFTER 2 DIGITS: IT MUST HAVE EITHER 1 OR 2 ALPHABETS**

**#FINALLY IT MUST ENDS WITH: 4 DIGITS**

**#want to know AP OR TS vehicle**

**import re**

**regno=input("Enter car registration number : ")**

**m1=re.fullmatch("[A-Z]{2}\d{2}[A-Z]{1,2}\d{4}",regno,flags=re.I)**

**if m1==None:**

**print("INVALID CAR REGISTRATION NUMBER")**

**else:**

**print("VALID CAR NUMBER IN INDIA")**

**m2=re.match("AP|TS",regno,flags=re.I)**

**if m2==None:**

**print("Car doesnt belongs to AP or TS it belongs to some other state")**

**else:**

**print("CAR from either AP/TS")**

**if m2.group().lower()=='ap':**

**print("GIVEN CAR REGISTRATION NUMBER BELONGS TO AP")**

**else:**

**print("GIVEN CAR REGISTRATION NUMBER BELONGS TO TS")**

**Output:**

**Enter car registration number : xyz01ab4569**

**INVALID CAR REGISTRATION NUMBER**

**Enter car registration number : KA12A2555**

**VALID CAR NUMBER IN INDIA**

**Car doesnt belongs to AP or TS it belongs to some other state**

**Enter car registration number : AP39CC1234**

**VALID CAR NUMBER IN INDIA**

**CAR from either AP/TS**

**GIVEN CAR REGISTRATION NUMBER BELONGS TO AP**

**Program 74:**

**#WHAT IS THE USE OF flags=re.I**

**import re**

**txt="""%$# the first season of IPL was played in 2008. the second season of iPl was in 2009. last season of ipl was played in**

**2018 won by CSK. csk won the Ipl title in 2010 and 2011 as well. MI has also ipL won the title 3 times in 2013, 2015 & 2017**

**"""**

**pattern=re.compile('ipl',flags=re.I)**

**matches=pattern.findall(txt)**

**for ele in matches:**

**print(ele)**

**Output:**

**IPL**

**iPl**

**ipl**

**Ipl**

**ipL**

**Program 75:**

**#demonstrate split() method of re module**

**import re**

**txt="pavan,praveen both are dealing courses for $1500 is it ? true"**

**L=re.split("[,$?]",txt)**

**print(type(L))**

**print("orginal text : ",txt)**

**print(L)**

**Output:**

**<class 'list'>**

**orginal text : pavan,praveen both are dealing courses for $1500 is it ? true**

**['pavan', 'praveen both are dealing courses for ', '1500 is it ', ' true']**

**Program 76:**

**#demonstrate webscraping**

**import re**

**from urllib import request**

**response=request.urlopen("https://www.jntuk.edu.in/contacts/")**

**data=response.read()**

**txt=str(data)**

**mails=re.findall("\w+@jntuk.edu.in",txt)**

**for ele in set(mails):**

**print(ele)**

**Output:**

**de@jntuk.edu.in**

**sdc@jntuk.edu.in**

**registrar@jntuk.edu.in**

**ce@jntuk.edu.in**

**nss@jntuk.edu.in**

**vc@jntuk.edu.in**

**ramesh\_biotech@jntuk.edu.in**

[**chiefengineer@jntuk.edu.in**](mailto:chiefengineer@jntuk.edu.in)

**Multithreading in Python**

**Q) What are the similarities b/w multitasking and multithreading?**

**1) Both are meant for performing multiple jobs at a time**

**2) OS should provide support in terms of time slicing/time sharing & context switching**

**3) Optimum utilization of CPU cycles is the criteria**

**Q) What is the difference b/w multitasking & multithreading?**

* **To achieve multitasking, we need not to do any special programming. OS completely takes care of it. Where as in multithreading, we have to code specially to achieve multiple jobs getting done concurrently.**
* **‘n’ jobs need ‘n’ processes incase of multitasking i.e., multitasking is a process based, multiple jobs performing concept. Where as in multi threading one single process can perform multiple jobs concurrently**

**Thread: A single sequential flow of control in a process is a Thread (or) An independent path of execution in a process**

**We have 2 kinds of applications**

1. **Single Threaded (single flow)**
2. **Multi Threaded ( multiflow)**

* **An application is said to be multithreaded if it has multiple flows of control. In each flow we can perform one task.**
* **When ever a single application is required to perform more than one job at a time, that application is a right candidate to be made multithreaded application**

**Ex: server application, Banking application**

* **A thread is a sub process**
* **In python a thread is an object**
* **Developing multithreaded application is main goal of multithreading**
* **Python’s contribution to multithreading is that it has brought system level programming to application level**

##### **Q) How to implement Multithreading in Python?**

**Ans: Where-ever a group of independent jobs are available, then it is highly recommended to execute them simultaneously instead of executing one by one** **using the concept of multithreading.**

* **Python provides one inbuilt module named “threading” to provide support for implementing multithreading concepts.**
* **So, developing multi-threaded Programs is very easy in python.**
* **The key point to remember is that, every Python Program by default contains one thread which is nothing but MainThread.**
* **The threading module contains function current\_thread() which returns the current executing Thread object.**
* **To set the name/get the name we can use a property called “name”**
* **Program 77:**

**#wap to print current thread details & also change the default name of the current thread**

**from threading import \***

**for i in range(20):**

**print("Hello")**

**for x in range(100,121):**

**print(x)**

**t1=current\_thread()**

**print("Current Thread Name : ",t1.name())**

**t1.name=”pavan”**

**print("After name change......Current Thread Name : ",t1.name())**

##### **Different Ways to Create a Thread in Python:**

**There are two ways in which we can create a thread in python. They are as follows:**

1. **Creating a Thread with pre-defined ‘Thread’ class**
2. **Creating our own thread class by inheriting Thread class**

##### **Creating a Thread using Thread class:**

**There is a pre-defined class named ‘Thread’ available within the ‘threading’ module. We can create threads by using that inbuilt class.**

**Program 78:**

**# first job: printing Hello message for 20 times**

**#second job: print 100 to 120**

**#implement multithreading**

**from threading import \***

**def wish():**

**for i in range(5):**

**print(t1.name,"Job is printing ::: Hello")**

**t1=Thread(target=wish)**

**t1.start()**

**ct=current\_thread()**

**for x in range(27,35):**

**print(ct.name+" job is printing ::: ",x)**

**Output:**

**Thread-1 Job is printing ::: Hello**

**Thread-1 Job is printing ::: Hello**

**Thread-1 Job is printing ::: Hello**

**MainThread job is printing ::: 27**

**MainThread job is printing ::: 28**

**Thread-1 Job is printing ::: Hello**

**Thread-1 Job is printing ::: Hello**

**MainThread job is printing ::: 29**

**MainThread job is printing ::: 30**

**MainThread job is printing ::: 31**

**MainThread job is printing ::: 32**

**MainThread job is printing ::: 33**

**MainThread job is printing ::: 34**

**Explanation:**

* **We created an object ‘t1’ for the class ‘Thread’ with the target as wish() method.**
* **Whenever we call start() method using the object ‘t1’, the thread execution starts.**
* **The wish() method will be executed parallely along with the other parts of the program.**
* **If multiple threads are present in our program, then we cannot expect exact execution order in output. Because of this we cannot provide exact output order for the above program. It varies from machine to machine while running**

##### **Advantage of multithreading:**

**The main advantage of multi-tasking is to improve performance of the system by reducing response time. Let’s see it in the example below**

**Program 79:**

**#without multithreading**

**from random import \***

**from time import \***

**def generate\_accnos():**

**for i in range(5):**

**accno=''**

**for x in range(11):**

**accno=accno+str(randint(0,9))**

**print("11 digit Account number ::: ",accno)**

**sleep(1)**

**def mailIds():**

**for i in range(5):**

**mid=''**

**for j in range(6):**

**mid=mid+chr(choice(range(97,123)))**

**print("Gmail Id suggested ::: ",mid+"@gmail.com")**

**sleep(1)**

**st=time()**

**generate\_accnos()**

**mailIds()**

**et=time()**

**print("Time taken to execute both the tasks : ",et-st)**

**Output:**

**11 digit Account number ::: 78107859463**

**11 digit Account number ::: 82014674845**

**11 digit Account number ::: 12114830724**

**11 digit Account number ::: 40794960075**

**11 digit Account number ::: 29600995472**

**Gmail Id suggested ::: wierra@gmail.com**

**Gmail Id suggested ::: iwacgp@gmail.com**

**Gmail Id suggested ::: aqfpmk@gmail.com**

**Gmail Id suggested ::: syumme@gmail.com**

**Gmail Id suggested ::: fqpxdd@gmail.com**

**Time taken to execute both the tasks : 10.09019160270691**

**Program 80:**

**#with multithreading... time factor**

**from threading import \***

**from random import \***

**from time import \***

**def generate\_accnos():**

**for i in range(5):**

**accno=''**

**for x in range(11):**

**accno=accno+str(randint(0,9))**

**print("11 digit Account number ::: ",accno)**

**sleep(1)**

**def mailIds():**

**for i in range(5):**

**mid=''**

**for j in range(6):**

**mid=mid+chr(choice(range(97,123)))**

**print("Gmail Id suggested ::: ",mid+"@gmail.com")**

**sleep(1)**

**st=time()**

**t1=Thread(target=generate\_accnos)**

**t2=Thread(target=mailIds)**

**t1.start()**

**t2.start()**

**t1.join()**

**t2.join()**

**et=time()**

**print("Time taken to execute both the tasks : ",et-st)**

**Output:**

**11 digit Account number ::: 67447261308**

**Gmail Id suggested ::: ipcaat@gmail.com**

**Gmail Id suggested ::: bolhws@gmail.com**

**11 digit Account number ::: 63111144640**

**11 digit Account number ::: 01637884952**

**Gmail Id suggested ::: hfvwge@gmail.com**

**11 digit Account number ::: 30880117112**

**Gmail Id suggested ::: wujzaj@gmail.com**

**11 digit Account number ::: 87100258143**

**Gmail Id suggested ::: jtgirs@gmail.com**

**Time taken to execute both the tasks : 5.052481174468994**

##### **join() method:**

* **If we want a particular thread to wait for another thread to complete, then we can use join() method.**
* **In the above program main thread will wait until the two child threads gets completed.**

**Program 81:**

**#with multithreading... time factor**

**from threading import \***

**from random import \***

**from time import \***

**class Bank:**

**def generate\_accnos(self):**

**for i in range(5):**

**accno=''**

**for x in range(11):**

**accno=accno+str(randint(0,9))**

**print("11 digit Account number ::: ",accno)**

**sleep(1)**

**class Gmail:**

**def mailIds(self):**

**for i in range(5):**

**mid=''**

**for j in range(6):**

**mid=mid+chr(choice(range(97,123)))**

**print("Gmail Id suggested ::: ",mid+"@gmail.com")**

**sleep(1)**

**acc=Bank()**

**g1=Gmail()**

**st=time()**

**t1=Thread(target=acc.generate\_accnos)**

**t2=Thread(target=g1.mailIds)**

**t1.start()**

**t2.start()**

**t1.join()**

**t2.join()**

**et=time()**

**print("Time taken to execute both the tasks : ",et-st)**

**Output:**

**11 digit Account number ::: 22808605994**

**Gmail Id suggested ::: agbeep@gmail.com**

**Gmail Id suggested ::: mquozd@gmail.com**

**11 digit Account number ::: 72486539899**

**Gmail Id suggested ::: klvpfo@gmail.com**

**11 digit Account number ::: 02368763757**

**Gmail Id suggested ::: ciwgzt@gmail.com**

**11 digit Account number ::: 25511978739**

**Gmail Id suggested ::: lxxwna@gmail.com**

**11 digit Account number ::: 13823296019**

**Time taken to execute both the tasks : 5.050032138824463**

**Program 82:**

**#how to pass arguments to a function where executing a function is the job of thread**

**from threading import \***

**def isEven(values):**

**for x in values:**

**print(f"{x} is even") if x%2==0 else print(f"{x} is odd")**

**nums=[14,21,20,10,5,45,78,99,100,11,433]**

**t1=Thread(target=isEven,args=(nums,))**

**t1.start()**

**for i in range(20):**

**print("How are you???")**

**Output:**

**14 is even**

**21 is odd**

**20 is even**

**How are you???**

**How are you???**

**10 is even**

**5 is odd**

**45 is odd**

**78 is even**

**99 is odd**

**100 is even**

**11 is odd**

**433 is odd**

**How are you???**

**How are you???**

**How are you???**

**How are you???**

##### **Creating a Thread class by inheriting Thread class:**

* **our class should inherit Thread predefined class in python to behave like Thread**
* **Predefined Thread class contains run() method**
* **In our child class we need to override the run() method with our required functionality.**
* **Whenever we call start() method then automatically run() method will be executed and performs our job.**

**Program 83:**

**#how to make our own class as a Thread class**

**from threading import \***

**class MyThread(Thread):**

**def run(self):**

**ct=current\_thread()**

**if ct.name=='first':**

**for i in range(5):**

**print(ct.getName(),"::Learning Advanced Python")**

**else:**

**for x in range(5):**

**print(ct.getName(),"::Core python course already completed")**

**t1=MyThread()**

**t2=MyThread()**

**t1.name="first"**

**t2.name="second")**

**t1.start()**

**t2.start()**

**Output:**

**first ::Learning Advanced Python**

**first ::Learning Advanced Python**

**first ::Learning Advanced Python**

**second ::Core python course already completed**

**second ::Core python course already completed**

**second ::Core python course already completed**

**second ::Core python course already completed**

**second ::Core python course already completed**

**first ::Learning Advanced Python**

**first ::Learning Advanced Python**

##### **Thread Identification Number (ident):**

**For every thread internally, a unique identification number will be available. We can access this by using implicit variable “ident”.**

##### **active\_count() method:**

**This function returns the number of active threads currently running.**

**Program 84:**

**#what is the identification number of a thread**

**# is it possible to restart a thread:no**

**from threading import \***

**from time import \***

**def wish():**

**print(current\_thread().name," is saying Hello")**

**sleep(2)**

**print(current\_thread().name," job is over")**

**t1=Thread(target=wish,name='child-1')**

**t2=Thread(target=wish,name='child-2')**

**t3=Thread(target=wish,name='child-3')**

**t4=Thread(target=wish,name='child-4')**

**ct=current\_thread()**

**t1.start()**

**t2.start()**

**t3.start()**

**t4.start()**

**print("No.of Active Threads : ",active\_count())**

**sleep(10)**

**print("child-1 identification number : ",t1.ident)**

**print(ct.getName()," identification number : ",ct.ident)**

**print("No.of Active Threads : ",active\_count())**

**Output:**

**child-1 is saying Hello**

**child-2 is saying Hello**

**child-3 is saying Hello**

**child-4 is saying Hello**

**No.of Active Threads : 5**

**child-2 job is over**

**child-4 job is over**

**child-1 job is over**

**child-3 job is over**

**child-1 identification number : 16052**

**MainThread identification number : 4776**

**No.of Active Threads : 1**

##### **enumerate() function:**

**This function returns a list of the objects of all active threads currently running.**

## ****Daemon Threads in Python:****

* **When a program is running there are some processes, related to the program, that will be running in the background supporting the program execution.**
* **Such threads or processes which are running in the background are called Daemon Threads.**
* **The main objective of Daemon Threads is to provide support for Non-Daemon Threads (like main thread)**

##### **Example: Garbage Collector**

* **Whenever Main Thread runs with low memory, immediately PVM runs Garbage Collector to destroy useless objects and to provide free memory, so that Main Thread can continue its execution without having any memory problems.**
* **We can check whether the thread is Daemon or not, by using isDaemon() method of Thread class and also by using daemon property.**
* **We have to set a thread as a Daemon before starting it. Once a thread starts, we cannot change its Daemon nature. If we try to change, we will get RuntimeError: cannot set daemon status of active thread**
* **The main thread is always Non-Daemon thread**

**Note:**

1. **When we start the execution of a program, the Main Thread will automatically start. We don’t explicitly start the Main Thread.**
2. **Since, by default, the Main Thread is a Non Daemon, and it starts automatically at the start to program execution, we can’t change its Non Daemon nature to Daemon.**
3. **Hence, the Main Thread will always be a Non Daemon thread.**
4. **The threads, which we create in the program, will inherit their Daemon nature from the parent thread, by default.**
5. **If the Parent Thread is Daemon, then child thread is also Daemon by default.**
6. **If the Parent Thread is Non-Daemon, then ChildThread is also Non-Daemon by default.**
7. **Whenever the last Non-Daemon Thread terminates, automatically all Daemon Threads will be terminated**

**Program 85:**

**#program on daemon behaviour of threads**

**from threading import \***

**t1=Thread(name='child-1')**

**ct=current\_thread()**

**print(f"{ct.name} is Daemon Thread?????",ct.daemon)**

**print(f"{t1. name } is Daemon Thread ????",t1. daemon)**

**t1. daemon=True**

**print(f"{t1. name } is Daemon Thread ????",t1. daemon)**

**t1.start()**

**Output:**

**MainThread is Daemon Thread????? False**

**MainThread is Daemon Thread????? False**

**child-1 is Daemon Thread ???? False**

**child-1 is Daemon Thread ???? True**

##### **Synchronization in Python:**

* **If multiple threads are executing simultaneously on object or data then there may be a chance of data inconsistency problems.**

##### **Let’s understand it through an example:**

**Consider a couple who is having a Joint account and both are having their ATM cards. They come to different ATMs and try to withdraw some amount at the same time. Let’s say the total balance in the account is 500 and Wife tries to withdraw 450 and the husband tries to withdraw 100. When they swipe the card for withdrawing money, the balance shown will be 500. Two threads will be created for the transaction, out of which only one thread should be successful and the other should fail. If both the threads get successful then its a loss to the bank. So, the threads should be in synchronization so that one fails and the other wins.**

**Program 86:**

**#inconsistent data**

**from threading import \***

**from time import \***

**class Account:**

**def \_\_init\_\_(self,ano,bal):**

**self.ano=ano**

**self.bal=bal**

**def withdraw(self,wamt):**

**print("Who Got chance????",current\_thread().name)**

**if self.bal>=wamt:**

**print(f"Please collect your cash from {self.ano} of Rs : {wamt}")**

**sleep(5)**

**self.bal-=wamt**

**sleep(5)**

**print(f"After withdraw operation........Balance in Accno {self.ano} is updated to Rs : {self.bal}")**

**else:**

**print(f"insufficient funds in your account {self.ano}")**

**class MyThread(Thread):**

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

**super().\_\_init\_\_()**

**self.acc=acc**

**def run(self):**

**acc.withdraw(600)**

**acc=Account(12345678,1000)**

**t1=MyThread(acc)**

**t2=MyThread(acc)**

**t1.setName("first thread")**

**t2.setName("second thread")**

**t1.start()**

**t2.start()**

**Output:**

**Who Got chance???? first thread**

**Please collect your cash from 12345678 of Rs : 600**

**Who Got chance???? second thread**

**Please collect your cash from 12345678 of Rs : 600**

**After withdraw operation........Balance in Accno 12345678 is updated to Rs : -200**

**After withdraw operation........Balance in Accno 12345678 is updated to Rs : -200**

##### **Q) How to overcome data inconsistency problems?**

**Ans: We can solve these inconsistency problems by synchronizing the threads such that they will be executed one by one. The main application areas of synchronization are,**

1. **Online Reservation system**
2. **Funds Transfer from joint accounts etc**.

##### **Synchronization By using Lock concept in python:**

* **Locks are the most fundamental synchronization mechanism provided by the threading module. We can create Lock object as follows,**

**l=Lock()**

* **The Lock object can be held by only one thread at a time. If any other thread wants the same lock then it will have to wait until the other one releases it. It’s similar to waiting in line to book a train ticket, public telephone booth etc.**

**acquire() method: A Thread can acquire the lock by using acquire() method  
 l.acquire()  
release() method: A Thread can release the lock by using release() method.  
 l.release()**

**Note: Only the thread currently holding the lock is allowed to call the release() method thread, otherwise we will get Runtime Error saying, RuntimeError: release unlocked lock**

**Program 87:**

**#thread synchronization with lock concept**

**from threading import \***

**from time import \***

**class Account:**

**def \_\_init\_\_(self,ano,bal):**

**self.ano=ano**

**self.bal=bal**

**def withdraw(self,wamt):**

**L.acquire()**

**print("Who Got chance????",current\_thread().name)**

**if self.bal>=wamt:**

**print(f"Please collect your cash from {self.ano} of Rs : {wamt}")**

**sleep(5)**

**self.bal-=wamt**

**sleep(5)**

**print(f"After withdraw operation........Balance in Accno {self.ano} is updated to Rs : {self.bal}")**

**else:**

**print(f"insufficient funds in your account {self.ano}")**

**L.release()**

**class MyThread(Thread):**

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

**super().\_\_init\_\_()**

**self.acc=acc**

**def run(self):**

**acc.withdraw(600)**

**acc=Account(12345678,1000)**

**L=Lock()**

**t1=MyThread(acc)**

**t2=MyThread(acc)**

**t1.name="first thread"**

**t2.name="second thread"**

**t1.start()**

**t2.start()**

**Output:**

**Who Got chance???? first thread**

**Please collect your cash from 12345678 of Rs : 600**

**After withdraw operation........Balance in Accno 12345678 is updated to Rs : 400**

**Who Got chance???? second thread**

**insufficient funds in your account 12345678**

**Walrus Operator**

* **Introduced in the version 3.8**
* **Walrus-operator is another name for assignment expressions**
* **it is a way to assign to variables within an expression using the notation NAME := expr**
* **The name is due to its similarity to the eyes and tusks of a Walrus on its side.**

**Ex1:**

**simple assignment operator: We are all familiar with how to assign a value to a variable. We do so using the simple assignment operator:**

**num = 15**

**And if we wanted to print the value of this variable using the print function, we can pass in the variable num as follows:**

**print(num)**

**# 15**

**By using Walrus Operator:**

**print(num:=15)**

**Ex2:**

**Without walrus operator:**

**msg=input("Enter a name : ")**

**while msg!='':**

**print("Excellent "+msg)**

**msg=input("Enter a name : ")**

**By using Walrus operator:**

**while (msg:=eval("Enter a name : "))!='':**

**print("Excellent "+msg)**

**Ex3:**

**Without Walrus operator:**

**vals=[5,2,1,8,3,6]**

**new\_list={x:x\*\*3 for x in vals if x\*\*3>150}**

**>>> new\_list**

**{8: 512, 6: 216}**

**With Walrus operator:**

**vals=[5,2,1,8,3,6]**

**new\_list={x:y for x in vals if (y:=x\*\*3)>150}**

**>>> new\_list**

**{8: 512, 6: 216}**

**Ex4:**

**Without Walrus Operator:**

**>>> from math import \***

**>>> half\_r=10**

**>>> print(f"Radius = {2\*half\_r} & area = {pi\*2\*half\_r\*2\*half\_r}")**

**Radius = 20 & area = 1256.6370614359173**

**With Walrus Operator:**

**>>>half\_r=10**

**>>> print(f"Radius = {(r:=2\*half\_r)} & Area = {pi\*r\*r}")**

**Radius = 20 & Area = 1256.6370614359173**

**Note: by using walrus operator if we assign a value to a variable within f string then that assignment expression must be enclosed within () so that u can use that variable anywhere in the program**

**FILE HANDLING**

* **As the part of programming requirement, we have to store our data permanently for future purpose. For this requirement we should go for files.**
* **Files are very common permanent storage areas to store our data.**

**Types of Files:**

**There are 2 types of files**

**1. Text Files:**

**Usually we can use text files to store character data**

**eg: abc.txt**

**2. Binary Files:**

**Usually we can use binary files to store binary data like images,video files, audio files etc...**

##### **Opening a File in Python:**

**Before performing any operation (like read or write or append) on the file, first we must open that file programmatically. In order to open the file, we have a predefined function open()in python. At the time of opening a file, we must specify the mode, which represents the purpose of opening a file.**

**Syntax: f = open(filename, mode)**

**When we execute the above command with proper filename and mode, then a file object will be returned, which we are storing in variable ‘f’. Operation performed on file object is the operation performed on file.**

##### **Closing a File in Python:**

**After completing our operations on the file, it is highly recommended to close the file programmatically. For this we have a predefined function close().**

**Syntax: f.close()**

##### **File Modes:**

1. **r**  **open an existing file for read operation. The file pointer is positioned at the beginning of the file.If the specified file does not exist then we will get FileNotFoundError.This is default mode.**
2. **w**  **open an existing file for write operation. If the file already contains some data then it will be overridden. If the specified file is not already avaialble then this mode will create that file.**
3. **a**  **open an existing file for append operation. It won't ovewrite existing data.If the specified file is not already avaialble then this mode will create a new file.**
4. **r+**  **To read and write data into the file. The previous data in the file will not be deleted.The file pointer is placed at the beginning of the file.**
5. **w+**  **To write and read data. It will overwrite existing data.**
6. **a+**  **To append and read data from the file.It wont override existing data.**
7. **x**  **To open a file in exclusive creation mode for write operation. If the file already exists then we will get FileExistsError.**

**Note: All the above modes are applicable for text files. If the above modes suffixed with**

**'b' then these represents for binary files.**

**Eg: rb,wb,ab,r+b,w+b,a+b,xb**

##### **Properties of File Object in Python:**

**The file object which we get when we open a file has many predefined methods which can be called to get the information about the file and the mode.**

1. **f.name → Name of opened file mode**
2. **f.mode → Mode in which the file is opened**
3. **f.closed → Returns a boolean value whether the file is closed or not**
4. **f.readable() → Returns a boolean value whether the file is readable or not**
5. **f.writable() → Returns a boolean value whether the file is writable or not**

**How to check a particular file exists or not?**

* **We can use os library to get information about files in our computer.**
* **os module has path sub module,which contains isFile() function to check whether a particular file exists or not?**

**os.path.isfile(fname)**

**Program 88:**

**#wap to create a new file in the current working directory & close it immediately**

**fp=open("sample.txt","w")**

**fp.close()**

**Program 89:**

**# how to know whether file exist or not with the specified name**

**from os.path import \***

**fname=input("Enter file name : ")**

**if(isfile(fname)):**

**print("FILE ALREADY EXIST WITH THE SPECIFIED NAME")**

**else:**

**fp=open(fname,"w")**

**print("NEW FILE IS CREATED IN THE CURRENT WORKING DIRECTORY SUCCESSFULLY")**

**fp.close()**

**Output:**

**D:\practice\python\file>py 2.py**

**Enter file name : test.doc**

**NEW FILE IS CREATED IN THE CURRENT WORKING DIRECTORY SUCCESSFULLY**

**D:\practice\python\file>py 2.py**

**Enter file name : test.doc**

**FILE ALREADY EXIST WITH THE SPECIFIED NAME**

**Program 90:**

**"""**

**Write a program to open a new file but with the specified name if already file**

**exists program should produce a message: "file exist with specified name, please**

**specify some other name**

**"""**

**from os.path import \***

**while isfile(fname:=input("Enter name of the file : "))==True:**

**print("Hello Boss File already exist with the specified name.... Try with another name")**

**fp=open(fname,"w")**

**print(f"NEW FILE CREATED WITH THE NAME {fname}")**

**fp.close()**

**Output:**

**D:\practice\python\file>py test.py**

**Enter name of the file : test.doc**

**Hello Boss File already exist with the specified name.... Try with another name**

**Enter name of the file : sample.txt**

**Hello Boss File already exist with the specified name.... Try with another name**

**Enter name of the file : input1.txt**

**NEW FILE CREATED WITH THE NAME input1.txt**

**Program 91:**

**#demonstrate x mode**

**fp=open(fname:=input("enter file name : "),"x")**

**print(f"NEW FILE CREATED WITH THE NAME {fname}")**

**fp.close()**

**Output:**

**D:\practice\python\file>py test.py**

**enter file name : input1.txt**

**Traceback (most recent call last):**

**File "D:\practice\python\file\test.py", line 2, in <module>**

**fp=open(fname:=input("enter file name : "),"x")**

**FileExistsError: [Errno 17] File exists: 'input1.txt**

**D:\practice\python\file>py test.py**

**enter file name : another.txt**

**NEW FILE CREATED WITH THE NAME another.txt**

**Program 92:**

**f=open("abc.txt", 'w')**

**print("File Name: ", f.name)**

**print("File Mode: ", f.mode)**

**print("Is File Readable: ", f.readable())**

**print("Is File Writable: ", f.writable())**

**print("Is File Closed : ", f.closed)**

**f.close()**

**print("Is File Closed : ", f.closed)**

**Output:**

**File Name: abc.txt**

**File Mode: w**

**Is File Readable: False**

**Is File Writable: True**

**Is File Closed : False**

**Is File Closed : True**

**Writing data to text files:**

* **We can write character data to the text files by using the following 2 methods.**

**write(str)**

**writelines(list of lines)**

**Note: While writing data by using write() methods, compulsory we have to provide line seperator(\n), otherwise total data should be written to a single line.**

**Program 93:**

**#wap to open a new file , copy the data to the file & close it**

**f=open("wish.txt", 'w')**

**f.write("Welcome\n")**

**f.write("to\n")**

**f.write("python world\n")**

**print("Data written to the file successfully")**

**f.close()**

**Output:**

**Data written to the file successfully**

**File Content: wish.txt**

**Welcome**

**to**

**python world**

##### **Appending Content:**

**In the above program, data present in the file will be overridden every time if we run the program. Instead of overriding, if we want append the data then we should open the file in append mode as follows:**

**Syntax: f = open(“wish.txt”, “a”)**

**Program 94:**

**f=open("wish.txt", 'a')**

**f.write("Learning")**

**f.write(" Python is")**

**f.write(" very easy")**

**print("Data written to the file successfully")**

**f.close()**

**Output:**

**Data written to the file successfully**

**File Content: wish.txt**

**Welcome**

**to**

**python world**

**Learning Python is very easy**

**Writelines(argument) method in python:**

**We can write lines of text into the file by using writelines method**

**Program 95:**

**f=open("names.txt", 'w')**

**list=["Ramesh\n" ,"Arjun\n", "Senthil\n", "Vignesh"]**

**f.writelines(list)**

**print("List of lines written to the file successfully")**

**f.close()**

**Output:**

**List of lines written to the file successfully**

**Content of the File:::: names.txt:**

**Ramesh**

**Arjun**

**Senthil**

**Vignesh**

**Reading Character Data from text files:**

* **We can read character data from text file by using the following read methods.**

**read(): To read total data from the file**

**read(n): To read 'n' characters from the file**

**readline(): To read only one line**

**readlines():To read all lines into a list**

**Sample File::: abc.txt content**

**Hello everyone!!**

**This topic is very important**

**Please don't sleep**

**Once this topic done**

**Then happily you can sleep**

**Thanks for cooperating**

**Program 96:**

**#wap to read the data from a file using read() method**

**fp=open(fname:=input("Enter file name : "),"r")**

**data=fp.read()**

**print(data)**

**fp.close()**

**Output:**

**Enter file name: abc.txt**

**Hello everyone!!**

**This topic is very important**

**Please don't sleep**

**Once this topic done**

**Then happily you can sleep**

**Thanks for cooperating**

**Program 97:**

**#wap to read the data from a file line by line using readline() method**

**fp=open(fname:=input("Enter file name : "),"r")**

**l1=fp.readline()**

**print(l1,end="")**

**l2=fp.readline()**

**print(l2,end="")**

**l3=fp.readline()**

**print(l3,end="")**

**fp.close()**

**Output:**

**Enter file name: abc.txt**

**Hello everyone!!**

**This topic is very important**

**Please don't sleep**

**Program 98:**

**#wap to read all lines from the file using readlines()**

**fp=open(fname:=input("Enter file name : "),"r")**

**total\_lines=fp.readlines()**

**print(len(total\_lines))**

**for line in total\_lines:**

**print(line,end="")**

**fp.close()**

**Output:**

**Enter file name: abc.txt**

**6**

**Hello everyone!!**

**This topic is very important**

**Please don't sleep**

**Once this topic done**

**Then happily you can sleep**

**Thanks for cooperating**

**The seek() and tell() methods:**

**tell():**

* **We can use tell() method to return current position of the cursor(file pointer) from beginning of the file.**
* **The position(index) of first character in files is zero just like string index.**
* **We can use seek() method to move cursor(file pointer) to specified location.**

**f.seek(offset, fromwhere)**

* **offset represents the number of positions**
* **fromwhere specifies from which location**

**Note: allowed value for fromwhere is only zero ( means from the beginning)**

**The with statement:**

* **With is a keyword in python**
* **The with statement can be used while opening a file.**
* **We can use this to group file operation statements within a block.**
* **The advantage of with statement is it will take care closing of file,after completing all operations automatically even in the case of exceptions also, and we are not required to close explicitly.**

**Content of the File: test.txt**

**Welcome**

**to**

**Python**

**Program 99:**

**#demonstrate tell() method**

**f=open("test.txt", "r")**

**print(f.tell())**

**print(f.read(2))**

**print(f.tell())**

**print(f.read(3))**

**print(f.tell())**

**Output:**

tell method in Python

**Program 100:**

**#wap to demonstrate seek() method**

**data="SampurneshBabu movie is excellent"**

**f=open("abc.txt", "w")**

**f.write(data)**

**with open("abc.txt", "r+") as f:**

**text=f.read()**

**print(text)**

**print("The Current Cursor Position: ",f.tell())**

**f.seek(24)**

**print("The Current Cursor Position: ",f.tell())**

**f.write("Britania Bisket")**

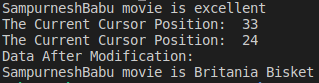
**f.seek(0)**

**text=f.read()**

**print("Data After Modification:")**

**print(text)**

**Output:**



**Program 101:**

**#wap to count no.of lines,characters,word in a file**

**fp=open(fname:=input("Enter file name : "))**

**total\_data=fp.readlines()**

**print(total\_data)**

**print("NO.of lines : ",len(total\_data))**

**cc=0**

**wc=0**

**for line in total\_data:**

**cc=cc+len(line)**

**wc=wc+len(line.split())**

**print("Character count : ",cc)**

**print("No.of words : ",wc)**

**fp.close()**

##### **sys.exit(0): It can be used to exit the system without executing the rest of the program. The argument represents status code and 0 means normal termination and it is the default value.**

## ****Working with binary files in Python:****

* **It is very common requirement to read or write binary data like images, video files, audio files etc.**

**Program 102:**

**#read the image from a file, copy that to some other binary file**

**fp1=open("d:\\posters\\1.jpg","rb")**

**fp2=open("pavan.jpg","wb")**

**data=fp1.read()**

**fp2.write(data)**

**fp1.close()**

##### **Working with csv files in Python:**

**csv means Comma separated values. As the part of programming, it is a very common requirement to write and read data wrt csv files. Python provides a csv module to handle csv files.**

**Program 103:**

**#wap to copy the data to csv file**

**import csv**

**with open("student.csv","w",newline='') as fp:**

**w=csv.writer(fp)**

**w.writerow(['rno','name','marks','avg'])**

**n=eval(input("Enter no.of students : "))**

**for i in range(n):**

**sn=eval(input("Enter roll numer: "))**

**sname=input("Enter student name : ")**

**mks=eval(input("Enter marks : "))**

**avg=mks/4**

**w.writerow([sn,sname,mks,avg])**

**Program 104:**

**#how to read the data from csv file**

**from csv import \***

**with open("student.csv") as fp:**

**fr=reader(fp)**

**for line in fr:**

**if len(line)!=0:**

**for word in line:**

**print(word,end="\t")**

**print()**

**pickling & unpickling:**

**Python pickle module is used for serializing and de-serializing python object structures. The process to converts any kind of python objects (list, dict, object of our own class etc.) into byte streams (0s and 1s) is called pickling or serialization or flattening or marshalling. We can converts the byte stream (generated through pickling) back into python objects by a process called as unpickling.**

**Program 105:**

**#pickle module**

**#dump(),load()**

**from pickle import \***

**class Employee:**

**def \_\_init\_\_(self,eid,ename,sal,exp):**

**self.eid=eid**

**self.ename=ename**

**self.sal=sal**

**self.exp=exp**

**def display(self):**

**print(f"eid={self.eid}\nname={self.ename}\nsalary={self.sal}\nexperience={self.exp}")**

**e1=Employee(145,"pavan",25000,19)**

**with open("emp.dat","wb") as fp:**

**dump(e1,fp)**

**with open("emp.dat","rb") as fr:**

**t=load(fr)**

**t.display()**

**GENERATOR**

* 1. **A generator is a special type of function which does not return a single value, instead, it returns an iterator object with a sequence of values.**
  2. **In a generator function, a yield statement is used rather than a return statement.**
  3. **we can say that if the body of any function contains a yield statement, it automatically becomes a generator function.**

### yield vs. return:

* **The yield statement is responsible for controlling the flow of the generator function. It pauses the function execution by saving all states and yielded to the caller. Later it resumes execution when a successive function is called. We can use the multiple yield statement in the generator function.**
* **The return statement returns a value and terminates the whole function and only one return statement can be used in the function.**

## Advantages of Generators:

**There are various advantages of Generators. Few of them are given below:**

### 1. Easy to implement

**Generators are easy to implement as compared to the iterator. In iterator, we have to implement \_\_iter\_\_() and \_\_next\_\_() function.**

**2. Memory efficient**

**Generators are memory efficient for a large number of sequences. The normal function returns a sequence of the list which creates an entire sequence in memory before returning the result, but the generator function calculates the value and pause their execution. It resumes for successive call. An infinite sequence generator is a great example of memory optimization.**

1. **Generate Infinite Sequence**

**The generator can produce infinite items. Infinite sequences cannot be contained within the memory and since generators produce only one item at a time**

**Example 1:**

**#wap to generate first n natual numbers using generator concept**

**def numbers(n):**

**i=1**

**while i<=n:**

**yield i**

**i=i+1**

**for x in numbers(100000000000000000000000000000000000000000000000):**

**print(x)**

**Example 2:**

**#wap to generate fibonacci series using generator**

**def fib(count):**

**a,b=0,1**

**while a<=count:**

**yield a**

**a,b=b,a+b**

**print("FIBONACCI SERIES..........")**

**for x in fib((n:=eval(input("Enter a value ")))):**

**print(x)**

**Python Database Connectivity**

**(PDBC)**

**Every application requires someplace to store the data like customer’s information, billing information, calling information, etc. To store the data we need storage areas. There are two storage areas normally. They are:**

1. **Temporary Storage Areas**
2. **Permanent Storage Areas**

##### **Temporary storage area:**

**These are the Memory Areas where Data will be stored temporarily. For example, Python objects like list, tuple, dictionary. Once the Python program completes its execution then these objects will be destroyed automatically, and data will be lost.**

##### **Permanent Storage Areas:**

**Also known as Persistent Storage Areas. Here we can store Data permanently. For example File Systems, Databases, Data warehouses, Big Data Technologies, etc**

##### **Limitations of File Systems:**

1. **We cannot store a huge amount of Information.**
2. **There is no Query Language support and hence operations will become very complex.**
3. **There is no Security for Data.**
4. **There is no Mechanism to prevent duplicate Data. Hence there may be a chance of data Inconsistency Problems.**

##### **Overcoming limitations:**

**To overcome the above Problems of File Systems, we should go for Databases.**

##### **Python Database Programming:**

**Sometimes as the part of Programming requirement, we have to connect to the database programmatically and we have to perform several operations like,**

1. **creating tables,**
2. **inserting data,**
3. **updating data,**
4. **deleting data,**
5. **selecting data etc.**

**The above operations in the database will be performed using SQL, a query language. In Python, we use some modules and send the queries/SQL commands to the database.**

**Python provides inbuilt support for several databases like**

1. **Oracle,**
2. **MySql,**
3. **SqlServer,**
4. **GadFly,**
5. **SQLite, etc**

**The support python provides for each database is through different modules for each like cx\_Oralce module for communicating with Oracle database, pymssql module for communicating with Microsoft SQL Server.**

##### **Standard steps for Python database Programming::**

1. **import database-specific module**
2. **Establish a Connection towards the database with credentials in a secured way.**
3. **Create a Cursor object**
4. **Use In-built methods to execute the SQL queries**
5. **Commit or rollback**
6. **Fetch the result from the Cursor**
7. **Close the resources**

##### **Step1: import database-specific module**

**We need to import specific database modules to work with the database. The modules can be imported using the import keyword. For example, if we are importing the cx\_Orcale module, then**

**import cx\_Oracle**

##### **Step2: Establish a connection**

**We can establish a connection between the python program and database by using the connect() method. If we call this connect() method then it returns a Connection object. For example,**

**con = cx\_Oracle.connect(“username/password@systeminfo”)**

**We need to provide the credentials, username, and password, in order to establish a connection to the database.**

##### **Step3: Create a Cursor object:**

**After creating a connection object then the next step is to create a Cursor object. With the connection object in the previous step, we have to call the cursor() method, which returns the Cursor object. For example,**

**cursor=con.cursor()**

##### **Step4: Execute the SQL queries**

**To execute SQL queries we can use three predefined methods based on the requirement,  
execute(“SQL query”): This method is used to execute a single SQL query  
cursor.execute()**

**executescript(“SQL queries”): This method is used to execute SQL queries which are separated by semicolon.  
cursor.executescript()**

**executemany(): This method is used to execute parameterized queries.  
cursor.executemany(“SQL queries”)**

##### **Step5: commit or rollback**

**We can commit or rollback the changes based on requirements in case of DML queries like insert, update, and delete queries. We can call commit or rollback methods by using a connection object.**

**commit(): This method saves the changes into the database. This should be called using connection object  
con.commit()**

**rollback(): This method rolls the temporary changes back. This should also be called using connection object  
con.rollback()**

##### **Step6: Fetch the results from the Cursor object**

**We can fetch or get the result from the cursor object in case of select queries. While fetching the result we can use three types of methods based on the requirement.**

**fetchone(): This method is used to fetch one record. Internally this method returns one record as a tuple. If there are no more records, then it returns None  
cursor.fetchone()**

**fetchmany(number of records): By using this method, we can fetch or get a number of records. This method accepts a number of records (integer value) to fetch and returns a tuple of tuples where each record itself is a tuple. If there are no more records, then it returns an empty tuple.  
cursor.fetchmany(2)**

**fetchall(): to fetch all records at a time**

##### **Step7: Close the resources:**

**In the above steps, we have created a connection object and then a cursor object. It’s our responsibility to close these as a part of good programming practice.**

**Closing cursor: cursor.close()  
Closing connection: con.close()**

##### **Summary:**

**The following is the list of important methods which can be used for python database programming**

1. **connect()**
2. **cursor()**
3. **execute()**
4. **executescript()**
5. **executemany()**
6. **commit()**
7. **rollback()**
8. **fetchone()**
9. **fetchall()**
10. **fetchmany(n)**
11. **close()**

**Since these methods are common for all databases they won’t be change from database to database.**