UNIT – IV

OBJECT ORIENTED PROGRAMMING

4.1: Object Oriented Programming :- In all the programs up to now, we have wrote our programs is with functions i.e.., which contains blocks of statements and that manipulate data. This is called procedure-oriented way of programming.

There is another way of organizing your program which is to combine data and functionality and wrap it inside something called an object. This is called object oriented programming.

Definition of OOPL :- OOPL refers to a type of high level computer programming language in which programmers define not only the data type of a data structure, but also the types of operations that can be applied on data. In this way, the data. In this way, the data structure becomes an object that includes both data and operations.

4.2:Features of Object Oriented Programming :- Python supports different features of object oriented programming

1.Like other general-purpose programming languages, Python is also an object-oriented language since its beginning.

2.Python supports object-oriented concepts like Class, Object, Inheritance, Polymorphism, Data Abstraction, Encapsulation and different methods.

3.**Class :-** Python supports the concept of class, that can be defined as a collection of objects. It is a logical entity that has some specific attributes and methods.

4.**Object :-** The Object is an entity that has state and behavior. It may be real-world object like mouse, keyboard, chair, table etc. Everything in python is an object and almost everything has attributes and methods.

5.**Inheritence :-** Inheritence is the most important aspect of object-oriented programming. By using Inheritence, we can create a class which uses all the properties and behaviors of another class. The new class is known as a derived class or child class. The main use of inheritance is to reuse the code again and again.

6.**Polymorphism :-** The word polymorphism has been derived from Greek word, here poly means many and morphs means forms. That means which has many forms by single. That means one operation can be performed in many ways.

7.**Encapsulation :-** Encapsulation is also an important aspect of object-oriented programming. It is used to restrict access to methods and variables. In encapsulation code and data are wrapped together within a single unit.

8.**Data Abstraction :-** Data abstraction and encapsulation both are often used as synonyms, because data abstraction is achieved through encapsulation.

9. It also provider data hiding , so it is more secure than any other programming language .

10. Python object oriented is platform independent language because we can write one time and can run for no of times .

11. It is very easy to understand , to write and to read.

12 . Python contains automatic garbage coveter .

13. It simulator the real world entity , so real world problems can be easily solved through oops .

14. Python also supports exception handling .

15. Python oops concept is revisable again and again .

16 . It is so Independent that means when one program is developed on one as that program can also run-on other operation system also .

4.3:Class :- A class is the main accept of object-oriented programming , infect a class is the basic building block of python .

~ We can also define a class is a blueprint for creating objects . In python everything is going to be treat as object insert .

~ A class can be simply said as it is a sketch (prototype) of a house . It contains all the details about the floors , doors , windows etc. . Based on these descriptions we build the house .

How to create a class :- Python has very simply syntax for defining a class . The syntax can be given as

```
Class className :
        <decstring>
        <statements
              -
              -
```

From the above syntax it is clear that the class definition is quite similar to function definition . To define a class we use the keyboard called “class” and followed by the className and a colon(:) operator as just.

~ The statements , in the class definition can be of any sequential instructions , decision control statements , aloop statements , and can even include function definitions .

~ The variables defined in a class are caved as class variables .

~ The functions defined in a class are caved as class methods .

~ Class variables and class methods together can be caved as class members .

~ The first statement in the class is a docstring . It serves as a comment in the class . This is an optical statement to use .

~ Class definitions can appear anywhere in program, but they are usually written after the import statement .

Example :- Class newclass :

```
                a = 10
                b = 20
        print(newclass-a)
        print(newclass-b)
```

Output :- 10

~ As soon as we define a class , a new class object is created with the same name . This class object access us to access the different attributes present in the class .

4.4:Creating objects :- On object is an instance of a class . We can create no of objects in a program . Every object contains some data and functions . After class creation the next job is to create an object . The object then access class variables and class methods by using the dot(.) operator . The syntax to create object is an follows :

```
Object_name =
class_name()
```

Example :- obj = animals()

From the above code we have created an object to the class animals .

~ We can also access variables present inside the class as

```
Objectname . variablename
```

Example :- Class animals :

```
        a = int(input("Enter first no"))(
        b = int(input("Enter second no"))
        c = a + b
    obj = animals()
    print("a is " , obj . a)
    print("b is" , obj . b)
    print("resent is " , obj . c)
```

Output :- a is 23

b is 32

result is 55

4.5:Class methods and self-variable :- Class methods are nothing at functions . The functions which are written in the class can be called as class methods . Class methods start with the keyboard called ‘def” , and method name followed by parametheses . by colen(:)) operators .

```
Def meyhodname () :
```

Syntax

Example :- Class ABC :

```
    Var = 10
    Def display():
        Print(“I am class method”)

ObJ = ABC()
Print(obj.var)
Obj.dsplay()
```

~ When you save and run the above program you are going to set error .

So, to avoid errors class methods must have the default and first argument with the name as “self” in the class method parameters list. This “self” is the first argument variable that is added to the beginning of the parameters list.

Here the self argument variable is nothing but the object.

Example:-class abc:

```
    def display(self):
        print(“hello class method”)
obj=abc()
obj.display()
```

Now the above class method will looks as follows

```
        obj.display()
        def display(obj):
            print("hello class method")
```

- You may have noticed self parameter in function definition. inside the class but,we called the method simply as obj.display() without any arguments but still is worked.
- This is because, whenever an object calls its method, the object itself is passed as the first argument.
- These means that even if a method that takes no arguments it has to defined a name called "self".
- Similarly a function defined to accept one parameter will actually take two arguments, one-the self variable and another the parameters.
- So the class method uses self, they require an object. For this reason they are after referred as "instance methods".

Eg:-

```
class ABC:
        def createName(self,name):
            self.name=name
            print("name of create method is",name)
    obj=ABC()
    obj.createName("Tony")
```

O/P:-name of create method is Tony

- self is nothing but temporary storage for object.
- self is not a keyword in python.

4.6:**Methods**:- A method is nothing but a function in oop. A function in a class can be called as methods. These methods have the same procedure as like functions. The methods will be identified with the same keyword called "def".

Syntax of method:- def methodName():

In python programming language there are mainly three types of methods exist. They are

1. instance method
2. class method
3. static method

1.Instance method:- An instance method is nothing but an ordinary method written with in the class. These instance method will take self as an argument in the parameters. Python provides its value

The instance method should be properly indented.

Ex: class animals: Instance method

```
    Cat=meow
    def sound(self):
        print("I am Instance method of animals")
obj=animals()
print(obj.cat)
obj.sound()
```

O/P: meow

I am instance method of animals.

The instance method takes object, so they work only with the help of objects.

4.7:Class Methods: Class methods are little different from these ordinary methods. We can easily identify a class method in the class. The class methods take first argument in the parameters as "cls" but not the "self". We can use classmethod decorator "@classmethod" for class definition. The syntax is as follows

Syntax: @classmethod

```
    def fun(cls, arguments…):
        statements.
```

The above class method syntax looks as ordinary method in the class.

A class method is a method that is bound to a class rather than its object. That means the class method does not require an object to call the class methods.

The creation of object for class method is optional.

```
Eg:     class person:
            @classmethod
             def age(cls):
                    print("i am class method age")
        person.age( )
```

o/p: i am class method age

from the above code it is clear that there is no necessary to create an object to call the classmethods,we can call the class methods with the classname itself.If you want to call the class method with object it is accessible,we don't get any error.

The class variable argument will be passes as an argument to the parameter that it is going to take class name as a parameter rather the class object.

That means it takes "person" class from the above program and takes that methods and calls the methodname with classname .

Class methods are widely used for factory methods which returns a class object(like constructor) for different usecases.

```
Eg:  class person:
         def val(self,name,age):
                self.name=name
                self.age=age
                print("name is:",name)
                print("age is:",age)
@classmethod
def volume(cls,name,age)
```

```
        cls.name=name
        cls.age=age
         print("classmet name is",age)
         print("classmet name is ",name)
obj=person()
obj.vol("Tony",24)
person.volume(20,"Rock")
```

O/P:- Name is Tony

age is 24

classmet name is Rock

classmet age is 20

4.8:Static methods:- Static methods are special case of methods in python. Static methods belongs to the class. But static methods knows nothing about the class and they just deals with its parameters static methods want take arguments as "self" or "cls". They have normal arguments, so simple they can be called as like ordinary methods. But the difference is that these methods can be called with classname or with object name.

Eg:- classname.staticmethodName()

(or)

Objectname.staticmethodName()

In static method we can pass any no.of arguments. So the static methods can be identified in the class very easily with the

help of decorator in python.

Syntax: @staticmethod

def methodName(arguments...)

statements

—

—

*static method will not have any permissions to access the class variables and its functionality even they belongs to class.

*static methods are mainly useful when you don't want to change or override a specific implementation of a method.

Eg: class person:

```
    def vol(self)
        print("Iam instance method")
  eclass method
    def area(name,age):
        print(name)
        print(age)
obj=person()
obj.vol()
person.volume()
person.area('Tony',25)
```

output: Iam instancemethod

Iam class method

Tony

25

Eg: class animal:

```
    var="welcome"
     def __len__(self):
```

```
                    return self.var
obj=animal()
print("The length of var is",len(obj.var))
```

output: The length of var is 7.

4.9:Constructor method: In python to initialize the variables in the class we have constructor. Constructor have a special name in python it can be called as __init__(). The init constructor always start with __(two underscores) constructor name followed by underscores of two(__) and then parenthesis.

*Constructors are mainly useful to initialize all the variables in the class.

*when you create a class without constructor in python, the interpreter automatically creates a default constructor for you that does not do anything.

Syntax: def __init__():

*Constructors also take one argument, "that argument can be called as self".

*The main aim of constructor is to assign values to instance variables that the objet need when it starts. Constructors verify that there are enough resources for the object and perform any start-up task.

Ex: class abc:

```
    def __init__(self):
        print("I am Constructor")
abc()
```

O/P: I am Constructor.

- Constructor are the first statements that are executed in the class.
- Constructors can be called without creating objects also.

There are two types of constructors in the class

1. Default Constructor
2. Parameterized Constructor

1.Default Constructor: The constructor that does not contain any arguments are called as default constructor.

Ex: class person:

```
        def __init__(self):
            print("I am Constructor")
            print("I am a default Constructor")
    person()
```

O/P: I am Constructor

I am a default Constructor

2.Parameterized Constructor: Parameterized constructor are used to take variables that initializes and to use in the class.

Ex:

```
Class abc:
    def __init__(self,val,name,val1):
        self.val=val
        self.name=name
        self.val1=val1
        print("the value is:",val)
        print("the name is:",name)
        print("the val1 is:",val1)
    def operation(self):
        op=self.val*self.val1
        print("the result of operation in method is:",op)
obj=abc(10,'raj',30)
obj.operation()
```

Output:- The value is : 10

The name is : raj

The val1 is : 30

The result of operation in method is : 300

- The init method is automatically involved when the object of the class is created.
- To pass valus to the constructor we have to pass values to the class. as constructors initializes values to the class.
- Constructors can take no. of arguments in init method.
- The default argument is the self variable in constructor.

4.10:Inheritance: The main aim of inheritance is a re-usability in python supports the concept of re-using existing classes.

So, the technique of creating a new class from an existing class is called inheritance. The existing class is called the base class and the new class is known as derived classes are created by first inheriting the data and the methods of the base class and then adding new specialized data and functions in it.

- In this process of inheritance, the base class remains unchanged.
- Inheritance follows the "Is-a" relationship.

Syntax of Inheritance:

```
class Baseclass:
        Body of the baseclass
Class Derivedclass(Baseclass):
        Body of derived class
```

- From the above syntax derived class or subclass inherits the features from the superclass or Baseclass.
- Inheritance follows the concept of Top-down approach.

In Top-down approach generalized classes are designed first and then specialized classes are derived by inheriting/extending the generalized classes.

Eg:

```
class parent:
    def__init__(self,name,age):
        Self.name=name
        Self.age=age
    def parentmethod(self):
        print(self.name)
        print(self.age)
class child:
    def __init__(self,name,age,exp,area):
        parent.__init__(self,name,age)
        self.exp=exp
        self.area=area
    def childmethod(self):
        parent.parentmethod(self)
        print(self.exp)
        print(self.area)
    obj=child('raj',20,45,70)
    obj.childmethod()
```

output:

```
raj
20
30
70
```

In inheritence there is no necessity to create an object for the super class.

We have to create object for the subclass.

All the constructors and methods in inheritence concept has to be inherited to the subclass.

4.11:Super( ) method:

Super keyword acts like a built-in-functions in python.The super built-in-function returns a proxy object that allows you to parent class.

Instead of calling the super class constructor and super class method names with the classname,they cal also be called with a built-in-function called super( ) in the childclass.

Syntax for the built-in-function.

Super( ).__init__(argument1,argument2)

Or

Super( ).methodName( )

With the help of super keyword function we can inherit super class method into the child class and also we can herit parent class instance methods into subclass.

Eg:The following example shows how to use super keyword or super( ) built-in-function in program.

```
Class person
    def__init__(self,name,age):
        self.name = name
        self.age = age
        print("Iam superclass variable",self.name)
        print("Iam superclass variable",self.age)
    def personmethod(self):
        print("Iam superclass method",self.name)
```

```
            print("Iam superclass method",self.age)
class  man(person):
        def __init__(self,name,age,height,weight):
            super(). __init__(name,age)
            self.height = height
            self.weight = weight
            print("Iam sub class var",self.height)
            print("Iam sub class var",self.weight)
         def  mankind(self):
              super().personmethod()
              print("Iam child method")
obj = man("rajesh",23,165,66)
obj.mankind()
```

o/p:- Iam superclass variable : rajesh

Iam superclass variable : 23

Iam subclass variable : 165

Iam sub class variable : 66

Iam superclass method : rajesh

Iam superclass method : 23

Iam child method

4.11.1:Types of inheritance:- In Python programming language there are different types of inheritance exists.Some of them are as follows:

1.Single Inheritance

2.Multiple Inheritance

3.Multi – level Inheritance

4. Multi path Inheritance

1. Single inheritance :- In Single Inheritance a class can be derived from a single base class.This type of inheritance can also be called as normal inheritance.It contains ony one base class

Base class

Sub class

Eg: class boy :

```
    def boymethod (self ):
        print("Iam super class method")
class girl :
    def girlmethod(self ):
        boy.boymethod (self )
        print("Iam sub class method")
obj = girl( )
obj.girlmethod( )
```

o/p:- I am super class method

I am sub class method

2.Mutiple Inheritance: When a derived class inherits features from more than one base class it is called "multiple" inheritance.The derived class has all the features of both the base classes and also it can contain new features.

Syntax:

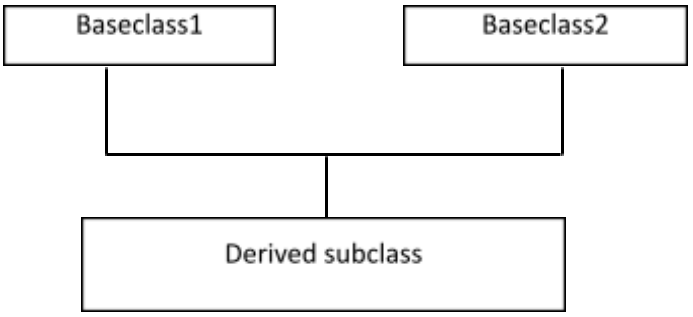
```
Class Baseclass:
    Block of statements
```

Class BaseClass:

Block of statements

Class Derived subclass(Baseclass1,Baseclass2):

Block of statements



- In multiple inheritance any attribute is first searched in current derived class.If it is not found here,then the search continues into parent classes using depth first search technique,that is in left-right fashion without searching same class twice.
- Ex:

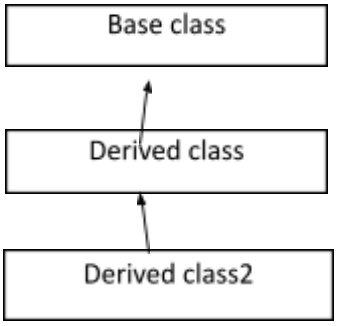
```
Class coral:
        def coralmethod(self):
                                print("I am coral method")
  class anemol:
        def coralmethod(self):
                                 print("I am second coral method")


Class animal(anemol,oral):
        Def animalmethod(self):
                Print("I am sub class")
obj.animal()
obj.coralmethod()
```

Output:

I am second coral method.

Multiple Inheritance:

The technique of deriving a class from an already derived class is called multi-level-inheritance.



- From the above figure the baseclass acts as a baseclass for derived classs which this class inturns acts like as a baseclass for derived class2.The derived class 1 has the features of baseclass plus its own features.
- In multiple-inheritance,number of levels can go up to any number based on the requirement.

Syntax:

```
Class Base:
        Statements
Class Derived(Base):
        Statements
Class Derived2(Derived1):
        Statements
```

- In multi-level inheritance the attributes are first searched in the current class(Derived classs).If it is not found there,then the derived class is searched,if it is not found then it goes to Baseclass.

Ex:

```
Class person:
        def name(self):
                print("I am name")
class teacher(person):
        def qualify(self):
                print("I am a phd teacher")
class hod(teacher):
        def experience(self):
                print(" I am an experienced teacher")
obj.hod()
obj.name()
obj.qualify()
obj.experience()
```

OUTPUT:

I am name

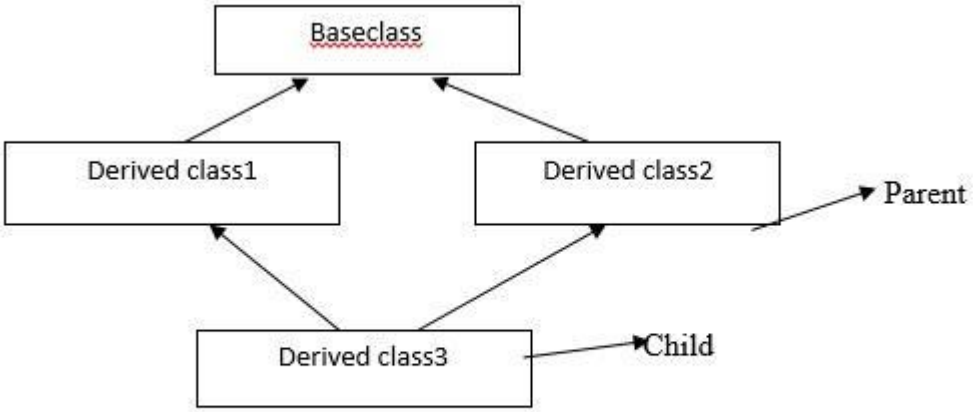
I am a phd teacher

I am a experienced teacher

4.Multi-path Inheritance:

Deriving a class from there derived classes that are these derived classes acts as a like base class.These classes in term derives from otherclasses.This other class can be called as a Baseclass.This type of concept

can be called as multi-path inheritance.



Ex:

```
Class student:
        def name(self):
                print("name")
class teacher(student):
        def quali(self):
                print("I am Phd")
class hod(student):
        def experience(self):
                print("I am experienced")
class result(hod,teacher):
        def eligibi(self):
                self.experience()
                self.quali()
                self.name()
obj.result()
obj.eligible()
```

OUTPUT:

I am experienced

I am phd

Name

4.12: OVERRIDING METHODS: overriding and overloading are the best examples for the concept of polymorphism so polymorphism is the best form of object oriented programming

Definition of polymorphism:

Polymorphism is the word derived from Greek language which poly means “many” and morphism means “forms” that means which has many forms.Polymorphism refers to the same name with many implementations.

Polymorphism contains two types of methods

1.Overriding methods

2.Overloading methods

1.**OVERRIDING METHODS:**

Overiding is nothing but the same name should exists in the parent class and also in the subclass.

- Overriding is possible only in the super class and also only in subclass that means one method with operation() is defined in the super class and the same method name called operation() should also be defined in subclasss.then we can say that overriding has been done.
- The overridden methods in two clases should contain the same methodname.but with different operation.
- When the methods are overridden the subclass method only is going to be executed by the interpreter.
- The method should be overridden if you want a different functionality in your subclass method name.

- When a method is overidden the superclass method is not considered at all.

Syntax for overriding:

Class classname:

def methodname(self):

……..

……..

OVERIDING

Class classname2:

def methodname(self):

Ex:

```
Class boy:
    def boyfriend(self):
        self.var=5
        Print("I am superclass value",self.var)
Class young:
    Def boyfriend(self):
        self.var=6
        Print("I am subclass value",self.var)
obj.young()
Obj.boyfriend()
```

OUTPUT:

I am subclass value:6

- With the help of overriding concept we can also override constructors.

4.13:ABSTRACTION:

Abstraction is the synonym for datahiding.Here datahiding can be implemented in methods by writing no code in the methods.That means empty methods are written in one class.These methods are implemented in another class with the help of inheritance.

- Therefore an abstraction class just serve as Template for other classes by defining a list of methods that class must implement.
- In abstract classes no object should be created for the base class.
- Object is created for the implementation class.
- In python to say an abstract method we use ”Notimplemented error” inside method definitions.
- So when we use “NotImplemented error” in method there is no necessity to create objects and tells to implement that the method is implemented in subclass.
- The abstract class is this an “interface definition”.

In heritance, we say that a class implements an interface, if it inherits from the class which specifies that interface.

Example: To illustrate concept of Abstract class

```
class built:
        def taste(self):
                raise NotImplementedError()
        def rich(self):
                raise NotImplementedError()
        def color(self):
                raise NotImplementedError()
class mango(built):
        def taste(self):
                return “sweet”
        def rich(self):
                return “vitamin A”
        def color(self):
                return “yellow”
obj = mango()
```

```
print(obj.taste(), obj.rich(), obj.color())
```

Output: sweet

vitamin A

yellow

4.14:Errors and Exceptions:-

Error:- An error is a term to describe any issue that arises unexpectedly. Basically there are two types of errors exist.

1. Syntax error 2. Logical Errors

1. Syntax Errors:- Syntax error arises due to poor understanding of the language. That means this type of errors comes because of grammar mistake or spelling mistakes. Hence this type of error can be detected and solved easily.

Example:

```
i = 0
if(i==0)
print(i)
```

- SyntaxError: invalid syntax

In the above program we have missed (:) after condition.

2. Logical Errors:- The other type of error known as logical errors, this occurs due to poor understanding of problem and its solution. Logical errors occurs due to wrong algorithm or logic. In some cases, logical errors may lead to divide by zero or accessing an item in a list where the index of the item is outside the bounds of list.

Example:

```
>>5/0
ZeroDivisionError: IntegerDivision or Modulo by zero.
```

Exceptions:- Even if a statement is syntactically correct, it may still cause an error when executed. Such errors that occur at run time are known as exceptions. An exception is an event which occurs during the execution of a program and disrupts the normal flow of a program instructions.

- When a program raises an exception, it must handle the exception, otherwise the program will be immediately terminated.
- Exceptions are again divided into two types
  1. Synchronous Exception
  2. Asynchronous Exception

1. **Synchronous Exceptions:** These exceptions are like dividing a number by zero, array index out of bound etc.
2. **Asynchronous Exception:** These exceptions like an interrupt from keyboard, hardware malfunction or disk failure etc.
   - In all the above 2 cases if they were not handled than an appropriate message will be displayed what had happened.

4.14:Handling Exceptions:- Whenever we have written a program, if we get any errors than we have to handle the exceptions. So, in python we can handle exceptions by using try block and except block.

- A critical operation which can raise exception is placed inside the try block and code that handles exception is written in except block.

Try and Except blocks:

Syntax:

```
try:
        Statements
except ExceptionName:
        Statements
```

The try statement works as follows:

- **Step 1**: First the try block statements between the try and except keywords is executed.
- **Step 2**:If no exception occurs then except block is skipped.
- **Step 3**:If an exception occurs try block handles.
- **Step 4**:Then except block will be executed after try block.

If try block what ever the logic you want to write and you think the logic is going to get error then you have to use try block.

Eg:

```
no=int(input("Enter the numerator:"))
deno =int(input("Enter the deno:"))
try:
    quo=no/deno
    print("Quotient:",quo)
except ZeroDivisionError:
    print("Denominator cannot be zero")
```

Output:

```
Enter the numerator:5
Enter the deno:0
Denominator cannot be zero
```

Multiple except blocks: We can write multiple except blocks for a single try block. The block which matches with the exception generated will get executed. A try block can be associated with more than one except block to specify handlers for different exceptions. However only one except block is executed.

Syntax:

```
try:
    operations are done in this block
except Exception1:
    if there is exception 1,then execute this block
except Exception2:
    if there is exception 2,then execute this block
-   -   -
else:
    If there is no exception execute this block
```

In the syntax else part is optional part.If you have written else block then it has to follow all the except block then else block has to be executed only if the try clause do not raise an exception.

Eg:

```
try:
    no=int(input("Enter no:"))
    print(no**2)
except(KeyboardInterruptError):
    print("You should enter a no")
except(ValueError):
    print("value error")
else:
    print("No error in program")
```

Output:

```
Enter no:2
4
No error in program
```

Multiple Exceptions in a single block :- Instead of writing multiple except blocks in program.we can write single except block with multiple handle exceptions.

Ex:-

```
try:
    No=int(input("Enter number"))
    Print(No ** 2)
Except(keyboardInterrupt,valueError,TypeError):
    Print("Please check before you entering…..")
Print("Bye')
```

O/P:- Enter number 3

9

Bye

Except block without Exception:- You can even specify an except block without mentioning any exception in except block.

Syntax :- try:

```
        Statement
    except:
        statement
    else:
        Statement
```

Ex:-

```
try:
    no=int(input("Enter a number"))
    print(no**2)
except:
    print("Enter Correct number")
else:
    print("No Error")
```

O/P:- Enter a number 3
9
No Error

4.15: Finally Block:- The try block has another optional block called finally which is used to define clean-up actions that must be executed under all circumstances. The finally block is always executed before leaving the try block . That means the statements written in finally block are executed irrespective of whether an exception has occurred or not .

Syntax :- try:

```
        Statement
    finally:
        This will be always executed.
```

Ex:- try:

```
        no=int(input("Enter a number"))
O/P:-   Enter a number 6
        print(no**2)                                        36
    except:                                                 I am optional
        print("You should enter a number")                  No Error
    else:
        print("I am optional ")
    finally:
        print("No Error")
```

⟶ We can write multiple try block , and we should not write finally block after try block

4.16:RAISING EXCEPTIONS:-

When we write a python program that does not contain any error,then that program is going to get output.Even though the program does not contain any errors the user may intentionally raise an error in the program.

So,to raise an error in the program,python provides a keyword called "raise" keyword.The general syntax for raise keyword is as follows

Syntax: raise[Exception[,args[,traceback]]]

From the above syntax by using the raise keyword we can raise the exceptions intentionally in the program.

Here,exception is the name of the exception to be raised.

(example,typeerror is an exception)

Example: raise Typeerror or raise Valueerror

.args is optional and specifies a value for the exception argument.

Example: raise Exception(‘Hello!World”)

If args is not specified,then the exception argument is none.

The final argument traceback is optional and if present,the traceback object is used for the exception.

Example: To raise an exception

```
Try:                                                    Output:  10
    no=10                                                       Exception occured
    print(no)
    raise ValueError
except:
    print(“Exception occurred”)
```

The most important thing in python is that we can re raise an exception in the except block.The re-raised exception can be handled in except block only.

Example:

```
try:
    raise NamEerror
except:
    print(“Re-Raising the exception”)
    raise
```

Output: Re-Raising the Exception

```
Traceback(most recent call last):
File”c:\python34\Try:py”,Line2 in module
Raise NameError
NameError
```

4.17:Instantiating Exeptions: Instantiation exception means creating a object to an exception.After creating object we can add arguments to the object before we raise exception.These arguments can be used to give additional information about the error.

To instantiate the exception,the the except block may specify a variable after exception name.The variable then becomes an exception instance.

Example: try:

```
        Raise Exception('Hello','world')
except Exception as errorobj:
        print(type(errorobj))      #Exception Instance.
        Print(errorobj.args)     #arguments stored in .args
        Print(errorobj)
        arg1,arg2=errorobj.args
        print('arguement1='arg1)
        print('arguement2='arg2)
```

Output: <type 'exceptions.Ecxception>

```
('hello','world')
arguement1=hello
arguement2=world
```