Unit-3

Functions: Functions, Built-in Functions, User Defined Functions, recursive functions, Scope of a Variable Python and **OOP:** Defining Classes, Defining and calling functions passing arguments, Inheritance, polymorphism, Modules– date time, math, Packages. **Exception Handling**- Exception in python, Types of Exception, User-defined Exceptions.

Functions:-

A function is a block of code that performs a specific task. function improves efficiency and reduces errors because of the reusability of a code. once we create a function, we can call it anywhere and anytime. the benefit of using a function is reusability.

--> Reusability means You can write a function once and use it many times without rewriting the same code again and again.

Reusability of code:-

using the same code again and again instead of writing it multiple times.

code reusability means write once, use many times.

Ex:-

```
a=10
b=20
print(a+b)
c=5
d=7
print(c+d)
```

-->> same logic written again and again.

Ex:-

```
def add(x,y):
    return x+y
print(add(10,20))
print(add(5,2))
```

-->> same code reused many times.

creating a function:-

* function blocks start with the def keyword. after that the function name and parentheses () are used.
* an argument or parameters should be pass inside the parentheses.

* any function's code block begins with a colon(:) and is indented.

Syntax:-

```
def function_name(parameter1,parameter2):
   # Function body
   return value
```

function_name= function name is the name of the function. we can give any name to function.

parameter= parameter is the value passed to the function. we can pass any number of parameters. function body uses the parameters value to perform an action.

function_body= the function body is a block of code that performs some tasks.

return value:- return value is the output of the function.

Ex:-

```
def message():
   print("shri gnanambica degree college..")
message()
```

Ex:-

```
def add(num1,num2):
   print("Number 1:",num1)
   print("Number 2:",num2)
   addition=num1+num2
   return addition
res=add(2,3)
print(res)
```

Types of functions:-

there are two types of functions. 1). predefined or built-in function 2). user-defined function.

1). predefined function:-

it is also called Built-in Functions are the functions that are already provided by Python, so we can use them directly without writing their code and without defining them using def.

Ex:- range(), id(), type(), input(), eval(), len(), sum(), max(), min()etc..

2). user-defined functions:-

A user-defined function is a function that is created by the programmer using the def keyword to perform a specific task.
when a function is written for a specific task, those functions are called as user-defined functions.

Ex:-

```
def greet():
   print("Hello how are you?")
greet()
```

function call:-

we may call a function to perform a task once it has been specified. for calling a function, we type the name of the function and the necessary parameters.

Parameters:

->Parameters are the variable names used in the function definition.
->They receive the values from arguments.

Arguments:

->Arguments are the actual values we pass during function call.
->They send values to parameters.
Ex:-

```
def my_function():
   print("this function has created")
my_function()    ---> function calling.
```

Ex:-

```
def add(a, b):
   print(a + b)
add(10, 20)
```

Here:
10 and 20 are arguments
a and b are parameters

Ex:-

```
def my_name(name):
   print(name)
my_name('Vishnu')   --> function calling with argument.
```

Ex:-

```
# function for sum of three numbers
def sum_three_numbers(num1,num2,num3):
   return num1+num2+num3
x=sum_three_numbers(10,20,30)   # --> function calling with three arguments
print(x)
```

Functional Parameters and Arguments in Python:-

Parameters are the variables defined in the function definition.
They act as placeholders for values.

Arguments are the actual values we pass to the function when calling it.

Ex:-

```
# two parameters F_name and L_name
def name_function(F_name, L_name):    #function definition
   print(F_name, L_name)
name_function("Shri","Gnanambica")    #function calling.
```

Ex:-

```
def add(a,b):
   print(a+b)
add(5,2)
```

What is return? :-

->return is a keyword in Python.
->It is used inside a function.
->It is used to send a value back from the function.
->After return is executed, the function stops.
->The returned value can be stored in a variable or used in calculations.

Ex:-

```
def add(a, b):
   return a + b

x = add(10, 20)
print(x)
```

Difference Between print() and return():-

1) print()

->print() is used to show the output on the screen.
->It is mainly used for display purpose.
->It does not give any value back.
->We cannot store or use its output later.

2) return()

->return is used to send a value back from a function.
->It is used for storing and using results later.
->It gives a value back (returns value).
->We can store the returned value in a variable.

print()	**return**
Shows result on screen	Sends result back from function
Used only for displaying output	Used to send value from a function
Does not return any value	Returns a value
Output cannot be stored in a variable	Returned value can be stored in a variable
Mainly used for debugging or display	Mainly used for further processing in program

scope of a variable:-

The scope of a variable in Python is defined as the specific area or region where the variable is accessible to the user. The scope of a variable depends on where and how it is defined. In Python, a variable can have either a global or a local scope.

Local Variable:-

A local variable is a variable that is declared inside a function and its scope is limited to that function only.

It cannot be accessed outside the function.

Ex:-

```
def my_function():
   x = 10   # local variable
   print(x)
my_function()
```

Global Variable :-

A global variable is a variable that is defined outside any function and is available throughout the entire program.

It can be used inside and outside functions.

Ex:-

```
x = 50   # global variable
def show():
   print(x)
show()
print(x)
```

Global Keyword:-

The global keyword is used in Python to declare that a variable inside a function refers to a global variable. Normally, variables created inside a function are treated as local variables. If we want to change the value of a global variable inside a function, we must use the global keyword.

Without using the global keyword, any assignment to a variable inside a function creates a new local variable, and the global variable remains unchanged.

Example:

```
x = 10
def change():
   global x
   x = 20
change()
print(x)
```

Output: 20

Real-Life Example: Classroom

Global Variable (Common to everyone)
School name
School rules
Holiday list
These are same for all classrooms

Why global?

Because everyone can see and use them
Local Variable (Specific to one place)
Student's exam paper
Student's marks sheet
Bench number in one classroom
Only that student / that class can use it

Why local?

Because it is needed only in one place

Real-Life Example: Bank

Global Variable
Bank name
Interest rate
IFSC code
Used by all branches & all customers
Local Variable
Your account balance shown at counter
Transaction amount you enter
Used only for that transaction

Real-Life Example: Mobile Phone

Global Variable
Phone language
Date & time
Battery percentage
Every app can read these

Local Variable
OTP entered in one app
Message you type in WhatsApp
Used only inside that app

OOP (Object-Oriented Programming):-

Object-Oriented Programming (OOP) is a programming paradigm that uses classes and objects to design applications. It represents real-world entities in the form of objects. OOP provides features like encapsulation, inheritance, polymorphism, and abstraction, which help in code reusability, security, and better program structure. Python supports OOP and allows programmers to build modular and maintainable applications.

1. class
2. objects
3. polymorphism
4. Encapsulation
5. inheritance
6. Data Abstraction.

class:-

A class in Python is a blueprint used to create objects. It defines the properties (variables) and behaviors (methods) of objects. Classes help in organizing code, improving reusability, and representing real-world entities.
we can create a class using class keyword followed by class_name
class class_name:

Ex:-

```
class Student:
    pass
```

-class → keyword
-Student → class name
-pass → empty class

object:-

An object in Python is an instance of a class. It represents a real-world entity and contains data and methods defined in the class. Objects are created using the

class name, and each object has its own separate data. Objects are used to access variables and methods of a class.

How to write and use an object in Python:-

STEP 1: Create a Class

Before creating an object, we must have a class.

```
class Student:
   def display(self):
      print("Hello, I am a student")
```

-This class just defines what a student can do.

STEP 2: Create an Object

Now we create an object from the class.

```
s1 = Student()
```

Explanation:
Student → class name
() → creating an object
s1 → object name
This line means: Create a real student object

STEP 3: Use the Object to Call a Method

```
s1.display()
```

What happens:

s1 → object
display() → method
Python sends s1 automatically to self
Output:

```
Hello, I am a student
```

Ex:-

```
class student:
   def display(self):
      print("Hello world")
s1=student()
s1.display()
```

Ex:-

```
class College:
   name = "Sgdc"

clg = College() # Creating object from class
print(clg.name) # Accessing the class
```

Ex:-

```
class Person:
   def __init__(self, name, age):
      self.name = name
      self.age = age

p1 = Person("Sri Ram", 30)
p2 = Person("Hanuman", 25)

print(p1.name, p1.age)  #Sri Ram  30
print(p2.name, p2.age)  #Hanuman  25
```

What is self in Python?:-

self is a reference to the current object of the class.
It is used to:

Access object variables
Access object methods
Store data inside the object

What is __init__() in Python?:-

__init__() is a special method called a constructor.

It runs automatically when an object is created.

Purpose of __init__():-

To initialize object variables
To assign values to object
To prepare object for use

self.name = name

Means:
- Take the value from parameter name
- Store it inside the object variable self.name

Ex:-

```
class Car:
    def __init__(self,brand,model,year):
        self.brand=brand
        self.model=model
        self.year=year
a1=Car('toyota','fortuner',2019)
print(a1.brand,a1.model,a1.year)
```

Inheritance in Python:-

Inheritance is an Object-Oriented Programming (OOP) concept where one class (child class) can acquire properties and methods from another class (parent class).

Syntax:-

```
class Parent:
    # properties and methods

class Child(Parent):
    # additional properties and methods
```

Parent class:-

Parent Class (also called Base Class or Super Class) is the class whose properties and methods are inherited by another class.

Parent class = Giver Class

Child Class:-

A Child Class (also called Derived Class or Sub Class) is the class that inherits properties and methods from another class.

Child class = Receiver Class

Real-Life Example:-

Think about:
- Father → has house
- Son → automatically gets that house

Here:
Father = Parent class
Son = Child class

Basic Syntax:-

```
class Parent:
    pass
class Child(Parent):
    pass
```

Ex:-

```
class Animal:
    def speak(self):
        print("Animal makes a sound")

class Dog(Animal):
    def bark(self):
        print("Dog barks")
d = Dog()
d.speak()   # Inherited method
d.bark()    # Child method
```

Explanation:

Animal is Parent class
Dog is Child class
Dog inherits method speak()
So we can call d.speak() even though it's not inside Dog class

Output:

Animal makes a sound
Dog barks

Types of Inheritance in Python:-

There are 5 types of inheritance in Python:

1. Single Inheritance
2. Multiple Inheritance
3. Multilevel Inheritance
4. Hierarchical Inheritance
5. Hybrid Inheritance

1. Single Inheritance:-

Single Inheritance is a type of inheritance in which a single child class derives features from a single parent class.
It is called single inheritance because:
There is only one parent class
There is only one child class
The child depends on only one parent

Ex:-

```
class Father:
    def show(self):
        print("This is father class")
class Son(Father):
    def display(self):
        print("This is son class")
s = Son()
s.show()
s.display()
```

2. Multiple Inheritance:-

Multiple Inheritance is a type of inheritance in which one child class inherits properties and methods from two or more parent classes.

Simple words lo:
One Child → Many Parents

Syntax:-

```
class Parent1:
   pass
class Parent2:
   pass
class Child(Parent1, Parent2):
   pass
```

Ex:-

```
class camera:
   def click_photo(self):
      print("photo clicked")
class musicplayer:
   def play_music(self):
      print("music playing")
class smartphone(camera,musicplayer):
   def phone(self):
      print("VIVO V60E smartphone")
p1=smartphone()
p1.click_photo()
p1.play_music()
p1.phone()
```

Ex:-

```
class Father:
   def s1(self):
      print("father property")
class Mother:
   def s2 (self):
```

```
        print("mother property")
class Son(Father, Mother):
    def s3(self):
        print("son property")
s4=Son()
s4.s1()
s4.s2()
s4.s3()
```

3. Multilevel Inheritance:-

Multilevel inheritance is a type of inheritance where a class inherits from another class, and that class itself inherits from another class. It forms a chain-like structure and helps in creating hierarchical relationships in programs.

Ex:-

Grandparent
↓
Parent
↓
Child

It is called multilevel inheritance because inheritance happens in multiple levels (or stages) instead of just one level.

Level 1 → Grandparent
Level 2 → Parent
Level 3 → Child

Syntax:-

```
class Grandparent:
    pass

class Parent(Grandparent):
    pass

class Child(Parent):
    pass
```

Ex:-

```
class Grandfather:
    def land(self):
        print("Grand fahter land")
class Father(Grandfather):
    def house(self):
        print("father house")
class Son(Father):
    def bike(self):
        print('son bike')
s=Son()
s.land()
s.house()
s.bike()
```

Explanation:-

Son inherits from Father
Father inherits from Grandfather
So Son can access:
land()
house()
bike()
This is multilevel inheritance.

Ex:-

```
class Person:
    def speak(self):
        print('person can speak')
class Employee(Person):
    def work(self):
        print('Employee works')
class Manager(Employee):
    def manage(self):
        print('manager manages team')
m=Manager()
m.speak()
m.work()
m.manage()
```

Here:
Manager inherits from Employee
Employee inherits from Person
So Manager gets all features

4. Hierarchical Inheritance:-

Hierarchical inheritance is a type of inheritance where multiple child classes inherit from a single parent class. It forms a tree-like structure and promotes code reusability.
One parent → Many children

Why It Is Called Hierarchical?

The word hierarchical means arranged in a hierarchy (tree structure).

Since one parent class branches into multiple child classes like a tree, it is called hierarchical inheritance.

Basic Syntax:-

```
class Parent:
   pass
class Child1(Parent):
   pass
class Child2(Parent):
   pass
```

Here:

Child1 inherits from Parent
Child2 inherits from Parent
Both share the same parent.

Ex:-

```
class Animal:
   def eat(self):
      print("Animal eats")
```

```
class Dog(Animal):
    def bark(self):
        print("Dog barks")
class Cat(Animal):
    def walk(self):
        print("Cat walk")
d = Dog()
c = Cat()
d.eat()
d.bark()
c.eat()
c.walk()
```

Explanation:-

Animal is the parent class
Dog and Cat are child classes
Both Dog and Cat can use eat() method
Dog has its own method bark()
Cat has its own method walk()

Ex:-

```
class Shape:
    def info(self):
        print('this is shape')
class Circle(Shape):
    def area(self):
        print("this is circle")
class Rectangle(Shape):
    def rect(self):
        print('this is rectangle')
a1=Circle()
b1=Rectangle()
a1.info()
a1.area()
b1.info()
b1.rect()
```

5. Hybrid Inheritance:-

Hybrid Inheritance is a combination of two or more types of inheritance.
It may combine:
Single Inheritance
Multiple Inheritance
Multilevel Inheritance
Hierarchical Inheritance
When these inheritance types are combined in one program, it is called Hybrid Inheritance.

Why It Is Called Hybrid?:-

The word Hybrid means “mixture”.
So, When we mix different inheritance types together, it is called Hybrid Inheritance.

Ex:-

```
class A:
   def show_A(self):
      print('Class A')
class B:
   def show_B(self):
      print('Class B')
class C(A):
   def show_C(self):
      print('Class C')
class D(B,C):
   def show_D(self):
      print('Class D')
obj=D()
obj.show_A()
obj.show_B()
obj.show_C()
obj.show_D()
```

Explanation:-

B and C inherit from A → (Hierarchical)
D inherits from B and C → (Multiple)
So this becomes Hybrid Inheritance.

Ex:-

```
class Student:
    def stu_info(self):
        print('Student Information')
class Sports(Student):
    def sports_marks(self):
        print("sports marks is 87")
class Arts(Student):
    def arts_marks(self):
        print("arts makrs is 90")
class Result(Sports, Arts):
    def total(self):
        print("total calculated")
r=Result()
r.stu_info()
r.sports_marks()
r.arts_marks()
r.total()
```

Explanation:-

Sports and Arts inherit from Student (Hierarchical).
Result inherits from Sports and Arts (Multiple).
So this is Hybrid Inheritance.

Polymorphism in Python:-

polymorphism means 'many forms' (poly means many and morphism means fomrs). in python, polymorphism is an OOP concept that allows the same function or method name to perform different actions depending on the object that is calling it.
one name --> many behaviors
Ex:-

```
class Dog:
    def sound(self):
        print('dog barks')
class Cat:
    def sound(self):
        print('cat meows')
```

```
d=Dog()
c=Cat()
d.sound()
c.sound()
```

here:-
- same method name- sound()
- different behaviour - dog barks, cat meows
  this is polymorphism.

Types of Polymorphism in Python:-

1). Method Overriding (Runtime Polymorphism)
2). Operator overloading
3). Method overloading
4). Duck Typing(Function Polymorphism)

1). Method overriding:-

method overriding happens when a child class provides a different implementation of a method that is already defined in its parent class.
Method overriding is also called Runtime polymorphism. Because the method that gets executed is decided at runtime, not a compile time.

Ex:-

```
class Animal:
    def sound(self):
        print("Animals make sounds")
class Dog(Animal):
    def sound(self):
        print("Dog barks")
class Cat(Animal):
    def sound(self):
        print("Cat meows")
a = Animal()
d = Dog()
c = Cat()
a.sound()
d.sound()
c.sound()
```

Explanation:

Animal class has method sound()
Dog class inherits Animal
Dog defines its own sound() → This overrides parent method
When d.sound() is called → Python executes Dog version
This decision happens at runtime
Ex:-

```
class Student:
    def result(self):
        print("General student result")

class BtechStudent(Student):
    def result(self):
        print("B.Tech student result calculated with internal + external marks")

class DegreeStudent(Student):
    def result(self):
        print("Degree student result calculated with semester marks")

s1 = BtechStudent()
s2 = DegreeStudent()

s1.result()
s2.result()
```

Operator Overloading:-

Operator Overloading is an OOP concept in Python that allows us to give special meaning to operators (like +, -, *, >, etc.) when they are used with user-defined objects.

Magic Method?

Magic methods are special methods in Python that start and end with double underscores and are automatically called by Python to perform special operations like object creation, operator overloading, and printing.

Operator Method

+	__add__()
-	__sub__()
*	__mul__()
/	__truediv__()
==	__eq__()
>	__gt__()
<	__lt__()

Ex:-

```
class Student:
   def __init__(self, marks):
      self.marks=marks
   def __add__(self,other):
      total=self.marks+other.marks
      print(total)
s1=Student(50)
s2=Student(60)
s1+s2
```

Ex:-

```
class A:
   def __init__(self,a):
      self.a=a
   def __add__(self,b):
      return self.a+ b.a
ob1=A(1)
ob2=A(2)
ob3=A('Shri')
ob4=A(' Gnanambica')
print(ob1+ob2)
print(ob3+ob4)
```

3). Method overloading:-

Method Overloading is a concept where multiple methods have the same name but different parameters. Python does not directly support it, but it can be achieved using default arguments or variable-length arguments.

python does not support method overloading by default. if you define multiple methods with the same name , only the latest definition will be used.

Ex:-

```
def product(a,b):
   p=a*b
   print(p)
def product(a,b,c):
   p=a*b*c
   print(p)
product(4,5,2)
```

the earlier definition product(a,b) gets overwritten. if you call product(4,5) it will be raise an error because the latest version expect 3 arguments.

Method Overloading using Default Arguments:-

A default argument means:

-> If the user does not give a value,
-> Python uses a default value.

Ex:-

```
class Demo:
   def add(self, a, b=0):
      print(a + b)
d = Demo()
d.add(10)      # Only one argument
d.add(10, 20)   # Two arguments
```

Explanation:-

def add(self, a, b=0):

Here:

a → compulsory argument

b=0 → default argument

If user gives only one value:

d.add(10)
Then:
a = 10
b = 0 (default value)
Result:
10 + 0 = 10
If user gives two values:
d.add(10, 20)
Then:
a = 10
b = 20
Result:
10 + 20 = 30

Variable Length Argument:-

Variable Length Arguments allow a function to accept any number of inputs instead of a fixed number of parameters.

Normally, when we define a function, we must specify how many arguments it can take. But sometimes we do not know how many values the user will pass. In such cases, we use variable length arguments.

In Python, this is done using the * symbol (for example, *args).

Ex:-

```
def fun(*names):
   for i in names:
      print(i)
fun('Ram','sita','hanuman')
fun('sgdc','svu')
fun('mpl')
fun()
```

Ex:-

```
def fun(*d):
   for i in d:
      print(i)
fun('Ram',10,'TPT')
```

```
fun(10,20,40)
fun(10,20)
fun()
```

Ex:-

```
def add(*numbers):
    print(sum(numbers))
add(10)
add(10,20)
add(10,20,30)
add(10,20,30,40)
```

4.Duck Typing(Function Polymorphism):-

Duck typing is created by defining the same method in different classes and using them in a common function without checking their type.

how to create Duck typing in python:-

there is no special keyword for duck typing.

1). create different classes.

2). give them the same method name.

3). use them in one common function.

step:-1

create Different classes.

```
class Duck:
    def speak(self):
        print('this is duck class')
class Dog:
    def speak(self):
        print("this is dog class")
```

(Both classes have the same method name speak()).

step:2

create a common function

```
def make_sound(animal):
    animal.speak()
```

step:3

pass objects

```
d=Duck()
dg=Dog()
make_sound(d)
make_sound(dg)
```

ex:-

```
class Duck:
    def fly(self):
        print('duck is flying')
class Airplane:
    def fly(self):
        print('airplane is flying')
def start_flying(obj):
    obj.fly()
d=Duck()
a=Airplane()
start_flying(d)
start_flying(a)
```

Ex:-

```
class Creditcard:
    def pay(self, amount):
        print('paid',amount,'using credit card')
class UPI:
    def pay(self,amount):
        print('paid', amount, 'using UPI')
class Paypal:
    def pay(self,amount):
        print('paid',amount,'using paypal')
def make_payment(method,amount):
    method.pay(amount)
cc=Creditcard()
up=UPI()
pp=Paypal()
```

```
make_payment(cc,85000)
make_payment(up,52000)
make_payment(pp,5200)
```

Encapsulation in Python:-

Encapsulation is the process of wrapping data and methods together in a class and restricting direct access to some of the object's components using private variables and public methods.

Example:-

A capsule medicine
Inside the capsule → medicine is safely covered
Outside → only the capsule is visible

Similarly in Python:
Data (variables) are kept inside a class
Methods control how that data is accessed

How Python Implements Encapsulation?:-

Python uses Access Specifiers

Access specifiers define how class members (variables and methods) can be accessed from outside the class. They help in implementing encapsulation by controlling the visibility of data. There are three types of access specifiers:

1). Public Members
2). protected members
3). Private members

1). Public Members:-

Public members are variables or methods that can be accessed from anywhere inside the class, outside the class or from other modules. By default, all members in Python are public. They are defined without any underscore prefix (e.g., self.name).

Ex:-

```
class Student:
   def __init__(self,name):
      self.name=name
s=Student('Ravi')
```

```
print(s.name)
```

2. Protected members

Protected members are variables or methods that are intended to be accessed only within the class and its subclasses. They are not strictly private but should be treated as internal. In Python, protected members are defined with a single underscore prefix (e.g., self._name).

Ex:-

```
class Student:
   def __init__(self,name):
      self._name=name
s=Student('Ravi')
print(s._name)
```

3. Private Members:-

Private members are variables or methods that cannot be accessed directly from outside the class. They are used to restrict access and protect internal data. In Python, private members are defined with a double underscore prefix (e.g., self.__salary)

Ex:-

```
class Student:
   def __init__(self,name):
      self.__name=name
s=Student('Ravi')
print(s.__name)
```

output:-

```
AttributeError: 'Student' object has no attribute '__name'
```

To access private data , we use
Getter and Setter methods.

Getter → Read data
Setter → Modify data safely

Ex:-

```
class Person:
    def __init__(self,age):
        self.__age=age
    def get_age(self):
        return self.__age
    def set_age(self,new_age):
        self.__age=new_age
p=Person(20)
print('current age:',p.get_age())
p.set_age(27)
print('update age:', p.get_age())
```

Explanation:-

1. Class Definition

```
class Person:
```

A class named Person is created.
It represents a blueprint for creating objects.

2. Constructor Method

```
def __init__(self, age):
    self.__age = age
```

__init__() is a constructor.
It runs automatically when an object is created.
self.__age is a private variable.
Double underscore (__) makes it private.
Private variables cannot be accessed directly outside the class.

3. Getter Method

```
def get_age(self):
    return self.__age
```

Getter method is used to read private data.
It returns the value of __age.
It does not modify the value.

Used to safely access private variables.

4. Setter Method
```
def set_age(self, new_age):
    self.__age = new_age
```

Setter method is used to update private data.
It changes the value of __age.
It does not return anything.
Used to safely modify private variables.

Object Creation
```
p = Person(24)
```
Object p is created.
Constructor runs automatically.
__age is set to 24.

First Print Statement
```
print('current age:', p.get_age())
```

Calls getter method.
Returns 24.

Output:
current age: 24

Updating Value
```
p.set_age(30)
```

Calls setter method.
Updates private variable to 30.

Second Print Statement
```
print('updated age:', p.get_age())
```
Getter is called again.
Returns updated value 30.

Output:
updated age: 30

Recursive Function in Python:-

A recursive function is a function that calls itself to solve a problem.
It is mainly used when a problem can be divided into smaller similar subproblems.

Basic Structure:-

```
def function_name(parameters):
    if condition:   # Base Case
        return result
    else:
        return function_name(smaller_problem)  # Recursive Call
```

Every recursive function must have two main parts:

Base Case:

A condition that stops the recursion and returns a result without making any further calls. This prevents infinite loops and the associated RecursionError in Python.

Recursive Case:

The part of the function that calls itself with a modified, smaller input, moving the problem closer to the base case.

Ex:

```
def fun(n):
    if n==0:
        return 0
    print('Haiii')
    fun(n-1)
fun(3)
```

Ex:- (factorial number)

```
def fact(n):
    if n==0 or n==1:
        return 1
    else:
        return n *fact(n-1)
print(fact(3))
```

Constructor in Python

A constructor in Python is a special method of a class that is automatically executed when an object of the class is created. It is mainly used to initialize the instance variables of the class.

In Python, the constructor is defined using the __init__() method. When an object is created, the __init__() method runs automatically and assigns values to the object variables.

Syntax:

```
class ClassName:
    def __init__(self):
        # initialization code
```

Example:-

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age

s1 = Student("Ram", 23)
print(s1.name, s1.age)
```

Types of Constructors in Python:

1). Default Constructor:-

A default constructor is a constructor that does not take any parameters (except self). It is used to initialize objects with default values.

Ex:-

```
class Student:
    def __init__(self):
        self.name = "Devendra"
        self.age = 23
    def display(self):
        print("Name:", self.name)
        print("Age:", self.age)
s1 = Student()
s1.display()
```

2. Parameterized Constructor :-

A parameterized constructor is a constructor that accepts parameters. It is used to initialize objects with different values.

Ex:-

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def display(self):
        print("Name:", self.name)
        print("Age:", self.age)
s1 = Student("Devendra", 23)
s2 = Student("Ravi", 22)
s1.display()
s2.display()
```

Modules in Python:-

a module can be represent as if it contains the classes, objects, variables, function then it is said to be a module. it helps us reduce code in different programs.

a module is a python file that contains reusable code and can be improved into another python program using the import keyword.

Types of Modules:-

in python, modules are mainly classified into two types:

1). Built-in modules(standard library modules)

2).User-Defined Modules.

1). Built-in modules:-

these modules are already provided by python. we don't need to create them- just import and use them.

Examples:-

math, random, datetime, os, sys

math module:-

the math module is a built-in module in python. it provides mathematical functions like square root, factorial, power, pi.

Ex:-

```
import math
print("Square root of 25:", math.sqrt(25))
print("Factorial of 5:", math.factorial(5))
print("Value of pi:", math.pi)
print("2 power 3:", math.pow(2, 3))
```

Random module:-

the random module is a built-in module used to generate random numbers or select random values.

it is useful in games, OTP generation, password generation etc.

Ex:-

```
import time
import random
for i in range(5):
   time.sleep(1)
   print(random.randint(1000,5000))
```

Ex:-

```
import random
print(random.choice([10,45,78,52,81]))
```

Date and Time module:-

python provides the datetime module to work with date and time.

it helps us to get current date, get current time, format date and time.

Ex:- (Get current Date and Time)

```
import datetime
now=datetime.datetime.now()
print('current date and time:', now)
```

Ex:- (Get only date)

```
import datetime
today=datetime.date.today()
print("today date is:", today)
```

Ex:- (Get only time)

```
import datetime
current_time=datetime.datetime.now().time()
print('current time is:',current_time)
```

Ex:-

```
import calendar
print(calendar.calendar(2022))
```

Ex:-

```
# Import the built-in calendar module
import calendar

# Specify the year and month
year = 2024
month = 2 # February

# Display the calendar for the specified month and year
print(calendar.month(year, month))
```

OS module:-

the os module is a built-in python module used to interact with the operating system.

it helps us to: work with files and folders, get current directory, create or remove directories, run system-related operations.

Example 1:- (Get current working directory)

```
import os
print(os.getcwd())
```

(getcwd() → Returns the current working directory path)

Example 2:- (list files in a folder)

```
import os
print(os.listdir())
```

(listdir() → Shows all files and folders in the current directory.)

Example 3: Create a New Folder

```
import os
os.mkdir("NewFolder")
```

mkdir() → Creates a new directory.

Example 4: Remove a Folder

```
import os
os.rmdir("NewFolder")
```

rmdir() → Removes a directory.

sys module:-

the sys module is a built-in python module used to interact with the python interpreter and system-specific parameters.

it helps us to: access command-line arguments, exit from a program, check python version, view module search path.

Ex:- (check python version)

```
import sys
print(sys.version)
```

(sys.version → Shows the Python version you are using.)

2). user-Defined Modules:-

A user-defined module is a module that is created by the programmer.

It is simply a Python file (.py file) that contains functions, variables, or classes, and can be reused in other programs.

When we create our own .py file and import it into another file, it is called a user-defined module.

Ex:-1

Step 1: Create a Module File

mymodule.py

```
def addition(a,b):
    return a+b
def sub(a,b):
    return a-b
```

step 2:- use the module in another file.

main.py

```
import mymodule
a=float(input('Enter Number1:'))
b=float(input('Enter Number2:'))
option=int(input('Enter 1 for addition or enter 2 for sub:'))
if option==1:
    print(mymodule.addition(a,b))
elif option==2:
    print(mymodule.sub(a,b))
else:
    print('you have choosen wrong option..')
```

Ex:2

student.py

```
def student_info(name,roll):
    return name, roll
def check_pass(mark):
    if mark >=35:
        return 'pass'
    else:
        return 'Fail'
```

main.py

```
import student
info=student.student_info('Samarasimha',2026)
print(info)
result=student.check_pass(98)
print('Result is:',result)
```

Packages in Python:-

packages in python are a way of organizing related modules into a structured directory. a package is basically a folder contains multiple python modules and a special file called __init__.py (used to treat the folder as a package).

to use a module from a package, we use the import statement, for example import package_name.module_name.

How to create a package in python:-

1. create a folder (package name)

Ex:- college (this folder name becomes your package name).

2. create __init__.py file

(this tells python this folder is a package and it can be empty).

3. create modules inside package

inside college folder create python files.

college/

-> __init__.py

-> student.py

-> teacher.py

4. write code inside modules

student.py

```
def student_details(name,roll):
   print('student name:', name)
   print('roll number:',roll)
```

teacher.py

```
def teacher_details(name, subject):
   print('teacher name:', name)
   print('subject:', subject)
```

5. create main file outside package and import package in main.py

main.py

```
from college.student import student_details
from college.teacher import teacher_details
student_details('Raghu',1254)
teacher_details('Devendra', 'python')
```

Exception Handling in Python:-

An exception is an error that occurs during program execution(runtime error). when an exception occurs, the normal flow of the program stops and python shows an error message.

Exception handling in python is a mechanism used to handle runtime errors so that the program does not crash suddenly. an exception is an error that occurs during the execution of a program , such as dividing by zero, accessing invalid index or opening a non-existing file.
python provides special keywords like try, except, else and finally to handle exceptions.

types of Exception:-

1). Built-in Exception
2). User-defined Exception

1). Built-in Exception

These are standard exceptions already present in Python's standard library to handle common errors that occur during program execution.

1. ZeroDivisionError:-

a ZeroDivisionError is raised when an attempt is made to divide a number by zero.

Ex:-

```
try:
   result=10/0
except ZeroDivisionError:
   print('Error, cannot divide by zero')
```

2. ValueError:

a valueerror is raised when a function receives an argument of the correct type but with an invalid value.

Ex:-

```
try:
   num=int('abc')
except ValueError:
   print('Error, invalid input,please enter a number')
```

3. IndexError:-

an indexerror is raised when an attempt is made to access an element of a list or other sequence that is out of range.

Ex:-

```
try:
   my_list=[1,2,3]
   print(my_list[5])
except IndexError:
   print('Error, list index out of range')
```

4. TypeError:-

a TypeError occurs when an operation is performed on an inappropriate data type.

Ex:-

```
try:
   result='hello'+10
except TypeError:
   print('Error, cannot add a string and an integer')
```

5. KeyError:-

a KeyError is raised when an attempt is made to access a key that does not exist in a dictionary.

Ex:-

```
try:
   my_dict={'name':'Ravi'}
   print(my_dict['age'])
except KeyError:
   print('Error: key not found in dictionary')
```

try:-

the main functionality of the try block is to enclose risky code that may generate exceptions and transfer control to the except block if an error occurs.

except:-

the except block is used to catch and handle exceptions raised in the try block. it prevents the program from crashing and allows normal program execution.

Ex:-

```
try:
   num=int('abc')
   print(num)
except ValueError:
   print('please enter a valid number')
```

Ex:-

```
a=10
b=0
try:
   res=a/b
   print(res)
except ZeroDivisionError:
   print('you cannot divide by zero please enter valid number..')
```

Explanation:

Code inside try may cause an error.
If division by zero happens → ZeroDivisionError
except block handles it.

else:-

the else block in exception handling executes only when no exception occur in the try block. it is used to write code that should run when the program executes successfully without errors.

Ex:-

```
try:
   marks=int(input('Enter marks:'))
except ValueError:
   print('Invalid input')
else:
   print('Marks recorded successfully.')
```

--> if input is correct -- else runs
--> if wrong input -- only except runs

finally:-

the finally block is used to execute important cleanup code that must run whether an exception occurs or not. it always executes after the try and except blocks.

Ex:-

```
try:
   a=10
   b=0
   print(a/b)
except ZeroDivisionError:
   print('cannot divide by zero')
finally:
   print('Execution completed')
```

2.User Defined Exceptions:-

User defined exceptions are custom exceptions created by programmers when built-in exceptions are not sufficient.

They are created by defining a class that inherits from the Exception class. The raise keyword is used to generate the exception and it can be handled using try and except.

Ex:-

```
class Age Error(Exception):
   pass

age = int(input("Enter your age: "))

try:
   if age < 18:
      raise AgeError("You are not eligible")
   else:
      print("You are eligible")
except AgeError as e:
   print(e)
```

Important Questions:-

Section	Question No.	Question
5 Marks (Short Answer Questions)	1	Write a short note on scope of variables.
	2	Write a short note on packages in Python.
	3	Explain types of errors in Python.
	4	Explain user-defined functions.
	5	Write about constructors in Python.
	6	Write a short note on Try–Except–Else–Finally blocks.
10 Marks (Long Answer Questions)	1	Write about the types of functions in Python in detail.
	2	Explain the classes and objects in Python.
	3	Define Object-Oriented Programming and explain different types of OOP features with examples.
	4	Write about polymorphism and explain different types of polymorphism with suitable examples.
	5	Define exceptions in Python and explain different types of exceptions with suitable examples.

B. Devendra MSc-CS
Dept of Computers
Shri Gnanambica Degree College(A)