Unit 2
Functions in Python

Defining a Function:

Function kya hota hai?

Function ek **block of code** hota hai jo kisi specific task ko perform karta hai.

Jab bhi zarurat ho, aap us function ko call karke use kar sakte hain instead of same code baar-baar likhne ke.

Function define kaise karte hain?

Python me function define karne ke liye def keyword use hota hai:

```
def function_name():
    # code block
```

Example:

```
def greet():
    print("Hello, welcome!")
```

Yahan greet function hai.

Calling a Function:

Function Call kya hota hai?

Function call ka matlab hota hai function ko execute karna (chalana)

Jab aap function define kar lete hain, tab tak wo sirf memory me hota hai.
Usko run karne ke liye **call karna padta hai**.

Syntax of Function Call

```
function_name()
```

Basic Example

```
def greet():
    print("Hello Rahul")

greet()   # function call
```

✓ Output:

Hello Rahul

Yahan greet() call hone par function execute hua.

Types of Functions:

Python me functions ko mainly **2 major categories** me divide kiya jata hai:

1️⃣ **Built-in Functions**

2️⃣ **User-defined Functions**

Iske alawa kuch special types bhi hote hain (jaise lambda, recursive, etc.)

1️⃣ Built-in Functions

Ye wo functions hote hain jo Python me already defined hote hain.
Aapko inhe banane ki zarurat nahi hoti.

Examples:

```
print("Hello")
len("Rahul")
type(10)
```

Common Built-in Functions:

- print() → output show karta hai
- len() → length batata hai
- type() → data type batata hai
- int(), float(), str() → type conversion

Advantages:

✓ Time save hota hai
✓ Reliable hote hain
✓ Direct use kar sakte hain

2️⃣ User-defined Functions

Ye wo functions hote hain jo **programmer khud banata hai**

Syntax:

```
def function_name():
    # code
```

Example:

```
def greet():
    print("Hello Rahul")

greet()
```

Advantages:

✓ Custom logic likh sakte hain
✓ Code reuse hota hai
✓ Large program ko manage karna easy

Function Arguments:

Function Arguments kya hote hain?

Arguments wo values hoti hain jo hum function ko call karte waqt pass karte hain.
Ye function ko input provide karti hain.

Parameters vs Arguments

Term	Meaning
Parameter	Function define karte waqt use hone wale variables
Argument	Function call karte waqt di gayi values

Example:

```
def greet(name):   # name = parameter
    print("Hello", name)

greet("Rahul")    # "Rahul" = argument
```

Types of Function Arguments

Python me arguments ke **main 4 types** hote hain:

1️⃣ **Positional Arguments**

Values **order (position)** ke according pass hoti hain

```
def student(name, age):
    print(name, age)

student("Rahul", 20)
```

✓ Output:

Rahul 20

⚠ Order change karne par output bhi change ho jayega:

student(20, "Rahul") # wrong order

---

2 Keyword Arguments

Argument ko **parameter ke naam se** pass karte hain
Order matter nahi karta

```
def student(name, age):
    print(name, age)

student(age=20, name="Rahul")
```

✓ Output:

Rahul 20

---

3 Default Arguments

Function define karte waqt **default value set** kar sakte hain
Agar argument pass nahi kiya to default use hoga

```
def greet(name="Guest"):
    print("Hello", name)

greet()
greet("Rahul")
```

✓ Output:

Hello Guest
Hello Rahul

---

4 Variable Length Arguments

Jab number of arguments fix nahi ho

(a) *args (Non-keyword arguments)

Multiple values ko tuple me store karta hai

```
def numbers(*num):
    print(num)

numbers(1, 2, 3, 4)
```

✓ Output:

(1, 2, 3, 4)

(b) **kwargs (Keyword arguments)

Multiple values ko dictionary me store karta hai

```
def info(**data):
    print(data)

info(name="Rahul", age=20)
```

✓ Output:

{'name': 'Rahul', 'age': 20}

Argument Passing Methods

1 Required Arguments

Ye compulsory hote hain

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

add(2, 3)
```

⚠ Agar missing hua:

```
add(2)  # Error
```

---

2 Mixed Arguments

Different types combine kar sakte hain

```
def func(a, b=10, *args, **kwargs):
    print(a, b, args, kwargs)

func(1, 2, 3, 4, x=5, y=6)
```

Important Rules

✓ Positional arguments pehle aate hain
✓ Keyword arguments baad me aate hain

✓ Default arguments last me hone chahiye

✓ *args aur **kwargs end me use hote hain

Common Errors

❌ Missing arguments

❌ Wrong order

❌ Duplicate values

```
def test(a, b):
   print(a, b)

test(2, a=3)  # Error (duplicate value)
```

Anonymous Functions:

Anonymous function wo function hota hai jiska **koi naam nahi hota**
Python me ise **lambda function** kehte hain

✓ Ye small aur one-line functions hote hain

Syntax of Lambda Function

lambda arguments: expression

lambda keyword use hota hai function define karne ke liye

Basic Example

```
add = lambda a, b: a + b
print(add(2, 3))
```

✓ Output:

5

Yahan lambda function do numbers add kar raha hai

Without Assigning (Direct Use)

```
print((lambda a, b: a + b)(5, 3))
```

✓ Output:

8

Lambda vs Normal Function

Normal Function:

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

Lambda Function:

```
lambda a, b: a + b
```

Lambda short aur compact hota hai

Lambda Function ke Use Cases

1 map() ke saath

Har element par function apply karta hai

```
nums = [1, 2, 3, 4]
result = list(map(lambda x: x*2, nums))
print(result)
```

✓ Output:

```
[2, 4, 6, 8]
```

---

2 filter() ke saath

Condition ke basis par filter karta hai

```
nums = [1, 2, 3, 4, 5]
result = list(filter(lambda x: x % 2 == 0, nums))
print(result)
```

✓ Output:

```
[2, 4]
```

---

3 sorted() ke saath

Custom sorting ke liye use hota hai

```
data = [(1, 3), (2, 1), (4, 2)]
result = sorted(data, key=lambda x: x[1])
print(result)
```

✓ Output:

```
[(2, 1), (4, 2), (1, 3)]
```

Multiple Arguments

```
mul = lambda a, b, c: a * b * c
print(mul(2, 3, 4))
```

✓ Output:

24

Lambda Function ki Limitations

❌ Sirf **ek expression** likh sakte hain
❌ Multiple statements nahi likh sakte
❌ Complex logic ke liye suitable nahi

Advantages

✓ Short aur simple
✓ Less code (one-line)
✓ Quick operations ke liye best

Disadvantages

❌ Readability kam ho sakti hai
❌ Debugging difficult hoti hai
❌ Complex tasks ke liye use nahi karna chahiye

Important Points

✓ lambda ek expression return karta hai (automatically)
✓ return keyword use nahi hota
✓ Function ka naam nahi hota
✓ Mostly temporary use ke liye hota hai

Lists & Tuple:

Introduction to List and Tuple:

1️⃣ List in Python

List ek **ordered collection** hoti hai elements ki
Ye **mutable (changeable)** hoti hai

Matlab: aap list ke data ko modify kar sakte hain

List ka Syntax

list_name = [element1, element2, element3]

Example

my_list = [1, 2, 3, "Deepak", 4.5]
print(my_list)

List Features

Ordered (index based)
Mutable (change ho sakti hai)
Different data types store kar sakti hai
Duplicate values allow karti hai

2️⃣ Tuple in Python

Tuple bhi ek **ordered collection** hai
Ye **immutable (change nahi hota)** hota hai

Matlab: ek baar create hone ke baad change nahi kar sakte

Tuple ka Syntax

tuple_name = (element1, element2, element3)

Example

my_tuple = (1, 2, 3, "Deepak")
print(my_tuple)

Tuple Features

Ordered
Immutable
Duplicate values allow karta hai
Faster than list

Accessing List and Tuple:

1️⃣ Accessing List Elements

Indexing kya hota hai?

List me har element ka ek **index (position number)** hota hai
Index **0 se start hota hai**

Example

my_list = [10, 20, 30, 40, 50]

Index	Value
0	10

Index	Value
1	20
2	30
3	40
4	50

Access by Index

```
print(my_list[0])  # 10
print(my_list[2])  # 30
```

Negative Indexing

Negative index last se start hota hai

```
print(my_list[-1])  # 50
print(my_list[-2])  # 40
```

Index	Value
-1	Last element
-2	Second last

Slicing (Range Access)

List ka ek part access karna

```
print(my_list[1:4])  # [20, 30, 40]
```

Format:

```
list[start : end]
```

✓ start included hota hai
✓ end excluded hota hai

Step Slicing

```
print(my_list[0:5:2])  # [10, 30, 50]
```

Format:

```
list[start:end:step]
```

Reverse List

```
print(my_list[::-1])  # reverse list
```

Nested List Access

```
my_list = [[1, 2], [3, 4], [5, 6]]

print(my_list[0])     # [1, 2]
print(my_list[0][1])   # 2
```

---

2 Accessing Tuple Elements

Tuple ka access bilkul list jaisa hi hota hai
Difference sirf itna hai ki tuple **immutable** hota hai

Example

```
my_tuple = (10, 20, 30, 40, 50)
```

Access by Index

```
print(my_tuple[0])  # 10
print(my_tuple[3])  # 40
```

Negative Indexing

```
print(my_tuple[-1])  # 50
```

Slicing

```
print(my_tuple[1:4])  # (20, 30, 40)
```

Step Slicing

```
print(my_tuple[::2])  # (10, 30, 50)
```

Nested Tuple Access

```
my_tuple = ((1, 2), (3, 4))

print(my_tuple[1])     # (3, 4)
print(my_tuple[1][0])   # 3
```

Operations:

1️⃣ List Operations

List **mutable** hoti hai, isliye isme kaafi operations perform kar sakte hain.

---

1. Adding Elements

append() → end me add karta hai

```
my_list = [1, 2, 3]
my_list.append(4)

print(my_list)  # [1, 2, 3, 4]
```

insert() → specific position par add karta hai

```
my_list.insert(1, 10)

print(my_list)  # [1, 10, 2, 3, 4]
```

extend() → multiple elements add karta hai

```
my_list.extend([5, 6])

print(my_list)  # [1, 10, 2, 3, 4, 5, 6]
```

---

2. Removing Elements

remove() → value se remove karta hai

```
my_list.remove(10)
```

pop() → index se remove karta hai

```
my_list.pop(2)
```

clear() → sab elements delete

```
my_list.clear()
```


---

3. Updating Elements

```
my_list = [10, 20, 30]
my_list[1] = 50

print(my_list)  # [10, 50, 30]
```

---

4. Searching & Counting

index() → position find karta hai

```
my_list = [1, 2, 3, 2]
print(my_list.index(2))  # 1
```

count() → occurrence count karta hai

```
print(my_list.count(2))  # 2
```

---

5. Sorting & Reversing

sort()

```
my_list = [3, 1, 2]
my_list.sort()

print(my_list)  # [1, 2, 3]
```

reverse()

```
my_list.reverse()
```

---

6. List Operators

Concatenation (+)

```
a = [1, 2]
b = [3, 4]

print(a + b)  # [1, 2, 3, 4]
```

Repetition (*)

```
print([1, 2] * 3)  # [1, 2, 1, 2, 1, 2]
```

Membership (in / not in)

```
print(2 in [1, 2, 3])  # True
```

Length

```
len(my_list)
```

---

2 Tuple Operations

Tuple **immutable** hota hai, isliye limited operations hote hain.

1. Accessing

```
t = (1, 2, 3)
print(t[0])
```

---

2. Tuple Operators

Concatenation (+)

```
t1 = (1, 2)
t2 = (3, 4)

print(t1 + t2)  # (1, 2, 3, 4)
```

Repetition (*)

```
print((1, 2) * 2)  # (1, 2, 1, 2)
```

Membership

```
print(2 in (1, 2, 3))  # True
```

Length

```
len(t)
```

---

3. Built-in Functions on Tuple

```
t = (3, 1, 2)

print(max(t))  # 3
print(min(t))  # 1
print(sum(t))  # 6
```

---

4. Counting & Index

```
t = (1, 2, 2, 3)

print(t.count(2))   # 2
print(t.index(2))   # 1
```

Key Difference in Operations

Feature	List	Tuple
Add	✓ Yes	❌ No
Remove	✓ Yes	❌ No
Update	✓ Yes	❌ No

Feature	List	Tuple
Sort	✓ Yes	❌ No
Immutable	❌ No	✓ Yes

Working with List and Tuple:

Working with List and Tuple in Python

"Working with" ka matlab hai:

✓ Create karna
✓ Access karna
✓ Modify (sirf list)
✓ Loop chalana
✓ Operations perform karna

1️⃣ Working with List

1. Creating a List

```
my_list = [10, 20, 30, 40]
```

2. Accessing Elements

```
print(my_list[0])   # 10
print(my_list[-1])  # 40
```

3. Modifying List (Mutable)

```
my_list[1] = 50
print(my_list)  # [10, 50, 30, 40]
```

4. Adding Elements

```
my_list.append(60)
my_list.insert(1, 15)
```

5. Removing Elements

```
my_list.remove(30)
my_list.pop()
```

6. Looping through List

Using for loop

```
for item in my_list:
    print(item)
```

Using index

```
for i in range(len(my_list)):
    print(my_list[i])
```

---

7. List Comprehension (Important 🔥)

Short way to create list

```
squares = [x**2 for x in range(5)]
print(squares)
```

✓ Output:

[0, 1, 4, 9, 16]

---

8. Nested List Working

```
matrix = [[1, 2], [3, 4]]

for row in matrix:
    for item in row:
        print(item)
```

2️⃣ Working with Tuple

1. Creating Tuple

```
my_tuple = (10, 20, 30)
```

2. Accessing Elements

```
print(my_tuple[0])
print(my_tuple[-1])
```

3. Tuple is Immutable

```
# ❌ Not allowed
my_tuple[1] = 50
```

4. Looping through Tuple

```
for item in my_tuple:
    print(item)
```

5. Tuple Unpacking

```
a, b, c = my_tuple
print(a, b, c)
```

6. Nested Tuple

```
t = ((1, 2), (3, 4))

for i in t:
    for j in i:
        print(j)
```

3️⃣ Common Operations (Both List & Tuple)

1. Length

```
len(my_list)
len(my_tuple)
```

2. Membership

```
print(20 in my_list)
print(50 not in my_tuple)
```

3. Concatenation

```
print([1, 2] + [3, 4])
print((1, 2) + (3, 4))
```

4. Repetition

```
print([1, 2] * 2)
print((1, 2) * 2)
```

Real-Life Use Cases

List Use:

✓ Shopping cart
✓ Student marks list
✓ Dynamic data storage

Tuple Use:

✓ Fixed data (coordinates)
✓ Database records (read-only)
✓ Configuration values

List vs Tuple in Working

Feature	List	Tuple
Modify	✓ Yes	❌ No
Speed	Slow	Fast
Memory	More	Less
Use	Dynamic	Fixed

Function and Methods:

1 Function kya hota hai?

Function ek **independent block of code** hota hai
Ye kisi object se directly tied nahi hota

Syntax

```
def function_name():
  # code
```

Example

```
def greet():
  print("Hello!")

greet()
```

Function Features

✓ Independent hota hai
✓ def keyword se define hota hai
✓ Reusable hota hai
✓ Arguments le sakta hai

---

2 Method kya hota hai?

Method ek function hi hota hai jo **kisi object ke saath attached hota hai**

Ye object ke through call hota hai

Syntax

```
object.method()
```

Example

```
my_list = [1, 2, 3]

my_list.append(4)   # method
```

Yahan append() ek **list method** hai

Method Features

✓ Object ke saath bound hota hai
✓ Dot (.) operator se call hota hai
✓ Specific object ke liye kaam karta hai

Function vs Method (Difference)

Feature	Function	Method
Definition	Independent	Object ke saath
Call	function()	object.method()
Example	print()	list.append()
Use	General purpose	Specific object

Built-in Functions vs Methods

Built-in Function Example

```
my_list = [1, 2, 3]

print(len(my_list))   # function
```

Method Example

```
my_list.append(4)    # method
```

Common Functions

✓ print() → output
✓ len() → length
✓ type() → data type
✓ sum() → total

Common List Methods

my_list = [1, 2, 3]

append()

my_list.append(4)

insert()

my_list.insert(1, 10)

remove()

my_list.remove(2)

pop()

my_list.pop()

sort()

my_list.sort()

String Methods (Example)

text = "hello"

print(text.upper()) # HELLO
print(text.lower()) # hello

Important Concept

Function → general use
Method → object-specific behavior

Example Together

my_list = [3, 1, 2]

print(len(my_list)) # function
my_list.sort()   # method

print(my_list)

Real-Life Analogy

Function = ek general tool (jaise calculator)
Method = specific tool (jaise washing machine ka button)

Dictionaries:

Working with Dictionaries:

Dictionary kya hota hai?

Dictionary ek **collection of key-value pairs** hoti hai
Har value ek **unique key** ke saath store hoti hai

✓ Format:

{key: value}

Example

```
student = {
   "name": "Deepak",
   "age": 20,
   "course": "BCA"
}
```

Yahan:

- name, age, course = keys
- "Deepak", 20, "BCA" = values

Properties of Dictionary

1. Ordered (Python 3.7+)

Elements insertion order maintain karte hain

2. Mutable

Dictionary ko change/update kar sakte hain

3. Keys Unique hoti hain

Duplicate keys allowed nahi

```
d = {"a": 1, "a": 2}
print(d)  # {'a': 2}
```

4. Values Duplicate ho sakti hain

```
d = {"a": 1, "b": 1}
```

5. Mixed Data Types

```
d = {"name": "Deepak", "marks": [90, 80], "active": True}
```

Working with Dictionary

1. Creating Dictionary

d = {"a": 1, "b": 2}

2. Accessing Values

print(d["a"]) # 1

Using get()

print(d.get("b")) # 2

Difference:

- d["x"] → error agar key na ho
- d.get("x") → None return karega

3. Adding Elements

d["c"] = 3

4. Updating Values

d["a"] = 10

5. Removing Elements

pop()

d.pop("b")

popitem()

d.popitem() # last item remove karega

del

del d["a"]

clear()

d.clear()

---

6. Looping in Dictionary

Keys

```
for key in d:
   print(key)
```

Values

```
for value in d.values():
    print(value)
```

Key-Value Pair

```
for key, value in d.items():
    print(key, value)
```

---

7. Nested Dictionary

```
students = {
    "s1": {"name": "A", "age": 20},
    "s2": {"name": "B", "age": 21}
}
print(students["s1"]["name"])
```

Dictionaries Functions/ Methods:

Dictionary ke saath jo built-in functions aur methods use hote hain unse hum data ko access, update aur manage karte hain.

1️⃣ keys() Method

Kya karta hai?

Dictionary ke **saare keys return karta hai**

```
d = {"name": "Rahul", "age": 20}

print(d.keys())
```

✓ Output:

```
dict_keys(['name', 'age'])
```

Ye ek view object return karta hai (list nahi)

2️⃣ values() Method

Kya karta hai?

Dictionary ke **saare values return karta hai**

```
print(d.values())
```

✓ Output:

```
dict_values(['Rahul', 20])
```

3️⃣ items() Method

Kya karta hai?

key-value pairs return karta hai (tuple form me)

print(d.items())

✓ Output:

dict_items([('name', 'Rahul'), ('age', 20)])

4️⃣ get() Method

Kya karta hai?

Key ki value safely return karta hai

print(d.get("name")) # Rahul
print(d.get("marks")) # None

Difference:

- d["marks"] → Error
- d.get("marks") → None

5️⃣ update() Method

Kya karta hai?

Dictionary ko update karta hai (add + modify dono)

d.update({"age": 25, "city": "Lucknow"})
print(d)

✓ Output:

{'name': 'Rahul', 'age': 25, 'city': 'Lucknow'}

6️⃣ pop() Method

Kya karta hai?

Specific key ko remove karta hai

d.pop("age")
print(d)

Return bhi karta hai removed value

7️⃣ popitem() Method

Kya karta hai?

Last inserted item remove karta hai

d.popitem()

Output tuple me aata hai:

('city', 'Lucknow')

8️⃣ clear() Method

Kya karta hai?

Puri dictionary empty kar deta hai

d.clear()
print(d)

✓ Output:

{}

9️⃣ copy() Method

Kya karta hai?

Dictionary ki copy banata hai

new_d = d.copy()

Original aur copy alag hoti hain

🔟 setdefault() Method

Kya karta hai?

Agar key exist nahi karti to add karta hai
Agar exist karti hai to value change nahi karta

d = {"name": "Rahul"}

d.setdefault("age", 20)
print(d)

✓ Output:

{'name': 'Rahul', 'age': 20}

1️⃣1️⃣ fromkeys() Method

Kya karta hai?

Multiple keys ke saath ek new dictionary banata hai

keys = ["a", "b", "c"]

d = dict.fromkeys(keys, 0)
print(d)

✓ Output:

{'a': 0, 'b': 0, 'c': 0}

1️⃣2️⃣ Built-in Functions

len()

len(d)

Number of items return karta hai

---

type()

type(d)

---

sorted()

sorted(d)

Keys ko sorted order me deta hai

Summary Table

Method	Use
keys()	Keys return karta hai
values()	Values return karta hai
items()	Key-value pairs
get()	Safe access
update()	Add/modify

Method	Use
pop()	Specific remove
popitem()	Last remove
clear()	Empty dictionary
copy()	Copy create
setdefault()	Default add
fromkeys()	New dictionary create

Module:

Module kya hota hai?

Module ek **file (.py file)** hoti hai jisme Python code (functions, variables, classes) likha hota hai

Iska use:
Code ko organize karne ke liye
Reusability ke liye

Simple Definition

Module = Python file containing reusable code

Example

Agar aap ek file banate hain:

mymodule.py

```
def greet(name):
    return "Hello " + name
```

Ye ek module ban gaya

Types of Modules

1. Built-in Modules

Python ke andar already available hote hain

Examples:

- math
- random
- datetime

```
import math

print(math.sqrt(16))
```

2. User-defined Modules

Jo aap khud banate hain

```
# mymodule.py
def add(a, b):
   return a + b
```

3. Third-party Modules

External libraries (install karni padti hain)

Examples:

- numpy
- pandas

Important Module Functions

dir()

Module ke andar kya-kya hai dikhata hai

```
import math
print(dir(math))
```

help()

Documentation show karta hai

```
help(math)
```

---

name Variable (Important)

Ye batata hai file directly run hui hai ya import hui hai

```
if __name__ == "__main__":
   print("This is main file")
```

Agar file directly run hogi tab hi ye execute hoga

Importing Module:

Importing module ka matlab hai:
kisi dusri Python file (module) ke code ko apne program me use karna

---

Basic Syntax

import module_name

Example

import math
print(math.sqrt(16))

✓ Output:

4.0

Yahan math module import kiya gaya hai

Import karne ke Different Tarike

1. Simple Import

import math
print(math.sqrt(25))

Har function ke saath module name likhna padta hai

---

2. Specific Import

from math import sqrt
print(sqrt(25))

Direct function use kar sakte hain (math likhne ki zarurat nahi)

---

3. Multiple Import

from math import sqrt, pi
print(sqrt(16))
print(pi)

---

4. Alias (Short Name)

import math as m
print(m.sqrt(36))

Short name use karne ke liye

---

5. Import All

from math import *

Saare functions import ho jate hain
⚠ Use karna avoid karein (confusion ho sakta hai)

User-defined Module Import

Agar aapne khud module banaya hai:

mymodule.py

```
def greet():
   print("Hello")
```

Import in another file

```
import mymodule
mymodule.greet()
```

Import ka Flow kaise hota hai?

Python module ko dhundhta hai:

1. Current directory
2. Installed libraries
3. System paths

Important Points

✓ Module ek hi baar load hota hai
✓ Multiple baar import karne par repeat load nahi hota
✓ Import se code reusable ban jata hai

Advantages of Importing Modules

Code reuse
Time saving
Organized code
Large projects easy

Math Module:

Math Module kya hota hai?

Ye ek **built-in module** hai jo mathematical operations ke liye use hota hai

Isme advanced math functions hote hain

Import kaise karein?

```
import math
```

Important Functions

1. sqrt() → square root

print(math.sqrt(16)) # 4.0

2. pow() → power

print(math.pow(2, 3)) # 8.0

3. ceil() → next integer (upar wala)

print(math.ceil(4.2)) # 5

4. floor() → lower integer

print(math.floor(4.8)) # 4

5. factorial()

print(math.factorial(5)) # 120

6. fabs() → absolute value

print(math.fabs(-10)) # 10.0

Important Constants

print(math.pi) # 3.14159...
print(math.e) # 2.718...

Uses:

Scientific calculations
Engineering formulas
Geometry problems

Random Module:

Random Module kya hota hai?

Ye module **random values generate** karne ke liye use hota hai

Games, OTP, simulations me use hota hai

Import

import random

Important Functions

1. random() → 0 se 1 ke beech value

```
print(random.random())
```

2. randint(a, b) → random integer

```
print(random.randint(1, 10))
```

3. randrange()

```
print(random.randrange(1, 10))
```

4. choice() → random element

```
list1 = [1, 2, 3, 4]
print(random.choice(list1))
```

5. shuffle() → list ko mix karta hai

```
random.shuffle(list1)
print(list1)
```

6. uniform() → float value

```
print(random.uniform(1, 5))
```

Uses:

Game development
OTP generation
Lottery system
Random testing

Packages:

Package kya hota hai?

Package ek **folder hota hai jisme multiple modules hote hain**

Ye large projects ko organize karta hai

Structure Example

```
my_package/
│
 ├── module1.py
 ├── module2.py
 └── __init__.py
```

---

init.py kya hota hai?

Ye file batati hai ki ye folder ek package hai
(Python 3 me optional hai but important concept hai)

Package ko import kaise karein?

import my_package.module1

Specific function import

from my_package.module1 import func_name

Difference: Module vs Package

Feature	Module	Package
Definition	Single file	Folder
Content	Functions, classes	Multiple modules
Use	Small code	Large project

Composition and The Distribution Utility:

Composition in Python (Object-Oriented Concept)

Composition kya hota hai?

Composition ek OOP concept hai jisme ek class dusri class ko **include (contain)** karti hai

Simple words me:
“Has-A relationship”

Example (Real Life)

Car has Engine
Computer has CPU

Python Example

```
class Engine:
    def start(self):
        print("Engine started")

class Car:
    def __init__(self):
        self.engine = Engine()   # composition
```

```
    def start_car(self):
        self.engine.start()

c = Car()
c.start_car()
```

Output:

Engine started

Key Points

Ek class ke andar dusri class ka object hota hai
Code reuse hota hai
Classes loosely coupled hoti hain

Advantages

Reusability
Flexibility
Better code structure

Composition vs Inheritance

Feature	Composition	Inheritance
Relationship	Has-A	Is-A
Flexibility	High	Low
Example	Car has Engine	Dog is Animal

Distribution Utility in Python

Ye concept generally **Python packages distribute karne ke liye use hota hai**

Matlab:
Aap apni Python library / project ko dusron ke saath share kar sakte hain

---

Distribution Utility kya hota hai?

Ye tools aur process hote hain jo Python code ko:
Package

Distribute
Install

karne me help karte hain

Common Tools

1. setuptools

Python package banane ke liye use hota hai

2. distutils

Old tool (ab deprecated ho raha hai)

3. wheel

Package ka binary format (.whl file)

---

setup.py Example

```
from setuptools import setup, find_packages

setup(
    name="myproject",
    version="1.0",
    packages=find_packages(),
)
```

Package Build kaise karein?

python setup.py sdist

Ye source distribution banata hai

Install kaise karein?

pip install myproject

PyPI kya hota hai?

Python Package Index (online store jahan packages upload hote hain)

Advantages of Distribution

Code share kar sakte hain
Reuse easy
Install simple ho jata hai
Professional development me use hota hai