Government Girls Polytechnic Bilaspur

Subject Name: Scripting Language (Python)
Subject Code: 2022372(022)
Semester : 3rd

Prepared By:
Mr. Nuresh Kumar Dewangan
Lecturer
Department of Computer Science & Engineering

Unit-1
Introduction to Python Programming

Features of Python - Python एक **high-level, interpreted, and general-purpose programming language** है, जिसे 1991 में **Guido van Rossum** ने develop किया था। यह modern programming के लिए बहुत popular है क्योंकि इसकी syntax **simple और readable** है।

1. Simple and Easy to Learn

- Python का syntax बहुत ही **simple और English-like** है।
- Beginners के लिए सीखना आसान है क्योंकि इसमें **complex syntax** नहीं होता।
- Example:

```
print("Hello, World!")
```

- बस एक line में output generate हो जाता है।

2. Interpreted Language

- Python **interpreted language** है, मतलब code **line by line execute** होता है।
- इसे run करने के लिए **compile** करने की जरूरत नहीं होती।
- इसकी वजह से **debugging और testing** आसान हो जाती है।

3. High-Level Language

- Python एक **high-level language** है, मतलब programmer को **memory management, hardware details** की चिंता नहीं करनी पड़ती।
- आप सीधे **logic और algorithms** पर focus कर सकते हैं।

4. Dynamically Typed Language

- Python में variables **declare करते समय type define करने की जरूरत नहीं** होती।
- Interpreter **automatically variable type detect** करता है।
- Example:

```
x = 10      # integer
x = "Hello" # string
```

5. Object-Oriented Programming (OOP) Support

- Python **object-oriented programming** support करता है।
- इसमें **classes, objects, inheritance, encapsulation, polymorphism** use किए जा सकते हैं।
- Example:

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

```
s1 = Student("Nuresh")
print(s1.name)
```

6. Open Source and Free

- Python **open-source** language है।
- इसका source code free में available है और आप इसे **modify और distribute** कर सकते हैं।

7. Portable / Platform Independent

- Python **cross-platform language** है।
- Python code **Windows, Linux, MacOS** सभी systems पर बिना change किए run किया जा सकता है।

8. Extensive Standard Library

- Python में **built-in library** बहुत बड़ी है, जिससे **file handling, networking, web development, database, GUI** आदि tasks आसानी से किये जा सकते हैं।
- Example:

```
import math
print(math.sqrt(16))  # 4.0
```

9. Integrated / Extensible

- Python को **other languages** (C, C++, Java) के साथ integrate किया जा सकता है।
- आप Python code में C/C++ modules import करके **performance improve** कर सकते हैं।

10. GUI and Web Development Support

- Python **Tkinter, PyQt** जैसी libraries provide करता है GUI applications बनाने के लिए।
- Web development के लिए Python में **Django, Flask** frameworks use होते हैं।

11. High-Level Data Structures

- Python में **lists, tuples, dictionaries, sets** जैसी powerful data structures available हैं।
- इनका use करके complex data easily handle किया जा सकता है।

```
fruits = ["apple", "banana", "mango"]
person = {"name": "Nuresh", "age": 23}
```

12. Extensible and Embeddable

- Python code को C/C++ में embed किया जा सकता है।
- Python को **other languages के programs में** integrate किया जा सकता है।

13. Interpreted and Interactive

- Python का interactive mode beginners के लिए बहुत useful है।
- आप Python shell में **directly commands test** कर सकते हैं।

```
>>> print("Hello")
Hello
```

14. Robust and Scalable

- Python का syntax और dynamic typing इसे **robust** बनाते हैं।
- Large-scale applications, web applications, AI, Machine Learning, Data Science आदि में Python widely used है।

15. Community Support

- Python की **community बहुत बड़ी और active** है।
- Internet पर Python के लिए **tutorials, documentation, forums** easily available हैं।

Working Modes of Python - Python एक **high-level, interpreted language** है और इसे use करने के लिए **different working modes** available हैं। इन modes के माध्यम से programmers Python code को **write, execute और test** कर सकते हैं।

Python के मुख्य दो working modes हैं:

1. Interactive Mode (Python Shell / REPL)- Interactive mode वह mode है जिसमें आप **directly Python commands को type करके execute** कर सकते हैं।
इसे **Python Shell** या **REPL (Read-Eval-Print Loop)** भी कहते हैं।

Features:

1. हर command **line by line execute** होती है।
2. Output तुरंत screen पर show होता है।
3. Errors को तुरंत identify किया जा सकता है।
4. Beginners के लिए यह mode बहुत useful है क्योंकि आप Python syntax और commands **practice** कर सकते हैं।

How to use Interactive Mode:

- Command Prompt / Terminal में type करें:

```
python
```

- Python shell open हो जाएगा:

```
>>>
```

- अब commands type करें:

```
>>> print("Hello, World!")
Hello, World!
>>> 5 + 3
8
```

Advantages of Interactive Mode:

- Immediate output देखने के लिए best।
- Quick testing के लिए ideal।
- Beginners को learning में help करता है।

Disadvantages:

- Large programs execute करना मुश्किल।
- Code save नहीं होता, अगर shell close हो गया तो lost हो जाता है।

2. Script Mode (File Mode / Program Mode) - **Script mode में Python program को** .py file **में लिखकर execute किया जाता है।**
यह mode large programs और projects **के लिए use किया जाता है।**

Features:

1. Python code को **text file (.py)** में save किया जाता है।
2. Program **एक साथ execute** होता है।
3. Error debugging interactive mode की तुलना में delayed होती है क्योंकि **पूरा code run** होने के बाद errors दिखते हैं।

How to use Script Mode:

1. किसी text editor में Python code लिखें:

```
# file: example.py
print("Hello from script mode!")
a = 10
b = 5
print("Sum =", a + b)
```

2. File save करें .py extension के साथ।
3. Command Prompt / Terminal में execute करें:

```
python example.py
```

4. Output screen पर दिखाई देगा:

```
Hello from script mode!
Sum = 15
```

Advantages of Script Mode:

- Large programs और projects के लिए best।
- Code **save और reuse** किया जा सकता है।
- Program structure maintain करने में easy।

Disadvantages:

- Code run करने से पहले save करना जरूरी है।
- Errors runtime के दौरान ही दिखाई देते हैं।

Comparison Between Interactive Mode and Script Mode:

Feature	Interactive Mode	Script Mode
Execution	Line by line	Entire program at once
File Required	No	Yes (.py)
Suitable For	Beginners, quick testing	Large programs, projects
Output	Immediate	After complete execution
Saving Code	Not possible	Possible

Variables in Python - Python में **variables** programming का एक **basic और important concept** है। Variables का use **data को store करने और manipulate करने** के लिए किया जाता है।

Definition:

Variable एक **name (identifier)** है जो किसी **memory location** को represent करता है और उसमें **value store** की जाती है।

- Variable data को **temporarily memory में रखता है**।
- Python में variable **dynamically typed** होते हैं, मतलब **type explicitly declare करने की जरूरत नहीं होती।**

Syntax:

```
variable_name = value
```

Rules for Naming Variables in Python:

1. Variable name **letters (a-z, A-Z), digits (0-9), और underscore (_) में** हो सकता है।
2. Variable name **digit से start नहीं** होना चाहिए।
3. Variable name में **special characters (@, $, %, etc.) नहीं** हो सकते।
4. Python keywords जैसे if, for, while variable names नहीं हो सकते।
5. Variable names **case-sensitive** होते हैं:

```
a = 10
A = 20
print(a)  # 10
print(A)  # 20
```

Types of Variables in Python:

1. Local Variable

- Function या block के अंदर define किया गया variable **sirf उसी block/function** में accessible होता है।

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

greet()  # Output: Hello
# print(message)  # Error: message is local
```

2. Global Variable

- Function या block के बाहर define किया गया variable **पूरे program में accessible** होता है।

```
x = 100  # Global variable

def show():
    print(x)

show()  # Output: 100
print(x)  # Output: 100
```

3. Constants

- Python में officially **constant variables** नहीं होते।
- Conventionally **UPPERCASE letters** से constants represent किए जाते हैं।

```
PI = 3.14159
GRAVITY = 9.8
```

Dynamic Typing in Python:

- Python variables **dynamically typed** होते हैं।
- इसका मतलब है कि **variable का type run-time पर detect** होता है।

```
x = 10       # Integer
x = "Hello"  # Now string
print(x)     # Output: Hello
```

Examples of Variables:

```
name = "Nuresh"      # String variable
age = 23             # Integer variable
height = 5.8         # Float variable
is_student = True    # Boolean variable

print("Name:", name)
print("Age:", age)
print("Height:", height)
print("Student:", is_student)
```

Output:

```
Name: Nuresh
Age: 23
Height: 5.8
Student: True
```

Important Points About Python Variables:

1. Variables **memory location को reference करते हैं।**
2. **Dynamic typing** के कारण same variable में different type values assign की जा सकती हैं।
3. Python में variables **case-sensitive** होते हैं।
4. Variables को **use करने से पहले define करना जरूरी** है।
5. Good practice: Variable names **meaningful और descriptive** होने चाहिए।

Data Types in Python - Python में **Data Types** programming का एक महत्वपूर्ण concept है। Data Type define करता है कि **variable में stored data का type क्या है और उस data के साथ कौन-कौन से operations perform किए जा सकते हैं।**

Python एक **dynamically typed language** है, इसका मतलब है कि **variable का type run-time पर detect होता है** और programmer को explicitly declare करने की जरूरत नहीं होती।

1. Numeric Data Types

Numeric data types में **numbers** को store किया जाता है। Python में तीन main numeric types हैं:

a) int (Integer)

- Whole numbers को represent करता है।
- Fraction या decimal number int में store नहीं हो सकता।
- Example:

```
age = 23
year = 2025
print(type(age))  # <class 'int'>
```

b) float (Floating Point Number)

- Decimal numbers को represent करता है।
- Fractional values और scientific notation support करता है।
- Example:

```
height = 5.8
pi = 3.14159
print(type(height))  # <class 'float'>
```

c) complex

- Complex numbers में **real और imaginary part** होता है।
- Imaginary part के लिए **j या J** use किया जाता है।
- Example:

```
z = 2 + 3j
print(type(z))  # <class 'complex'>
```

2. Sequence Data Types

Sequence data types में **ordered collection of items** होते हैं।

a) str (String)

- Text या characters को represent करता है।
- Single, double, या triple quotes में लिखा जा सकता है।
- Example:

```
name = "Nuresh"
greeting = 'Hello'
multiline = """This is
a multi-line
string"""
print(type(name))  # <class 'str'>
```

b) list

- Ordered collection of **mutable elements**।
- Square brackets [ ] use किए जाते हैं।
- Example:

```
fruits = ["apple", "banana", "mango"]
fruits[0] = "orange"  # Mutable
print(fruits)  # ['orange', 'banana', 'mango']
```

c) tuple

- Ordered collection of **immutable elements**।
- Round brackets ( ) use किए जाते हैं।
- Example:

```
coordinates = (10, 20)
# coordinates[0] = 15  # Error: tuple is immutable
print(type(coordinates))  # <class 'tuple'>
```

d) range

- Sequence of numbers generate करने के लिए use होता है।
- Example:

```
numbers = range(1, 6)
print(list(numbers))  # [1, 2, 3, 4, 5]
```

3. Set Data Types

Set में **unique और unordered elements** store होते हैं।

- Curly braces { } use किए जाते हैं।
- Duplicates automatically remove हो जाते हैं।
- Example:

```
colors = {"red", "green", "blue", "red"}
print(colors)  # {'red', 'green', 'blue'}
print(type(colors))  # <class 'set'>
```

4. dict (Dictionary)

- Key-value pairs store करता है।
- Curly braces { } और colon : use होते हैं।
- Keys unique होती हैं और values किसी भी type की हो सकती हैं।
- Example:

```
student = {"name": "Nuresh", "age": 23, "city": "Bilaspur"}
print(student["name"])  # Nuresh
print(type(student))  # <class 'dict'>
```

5. Boolean Data Type

- True या False values को represent करता है।
- Often **condition check और logical operations** में use होता है।
- Example:

```
is_student = True
passed = False
print(type(is_student))  # <class 'bool'>
```

6. None Type

- None represent करता है **no value / null value**।
- Often **placeholders** या **empty variable** के लिए use होता है।
- Example:

```
x = None
print(type(x))  # <class 'NoneType'>
```

Type Conversion in Python (Type Casting)

Python में variables का type **convert** किया जा सकता है:

```
x = 10      # int
y = float(x) # float
z = str(x)   # string
print(y, z)
```

Keywords in Python - Python में **keywords** programming का बहुत important concept हैं। Keywords वह **reserved words** होते हैं जिनका Python language में **special meaning** होता है। इन्हें **variable, function या identifier** के रूप में use नहीं किया जा सकता।

Definition:

Keywords Python की predefined words हैं, जिन्हें language internally use करती है।

- ये **syntax और structure** define करते हैं।
- Python में currently लगभग **36+ keywords** हैं (version 3.x के अनुसार)।

Example: if, else, while, for, def, return, class

Features of Python Keywords:

1. **Reserved Words**: ये words program में किसी और purpose के लिए use नहीं किए जा सकते।
2. **Case Sensitive**: Python keywords **lowercase letters** में लिखे जाते हैं। Uppercase version valid नहीं होगा।

```
If = 10   # Invalid
if = 10   # Valid (used in syntax)
```

3. **Predefined Meaning**: हर keyword का Python interpreter के लिए **special purpose** होता है।
4. **Cannot be Used as Identifiers**: Variable या function name के रूप में keywords use नहीं किए जा सकते।

List of Common Python Keywords:

Keyword	Purpose
if	Condition check के लिए
else	Alternate condition
elif	Multiple conditions check
for	Looping के लिए
while	Looping के लिए
break	Loop को stop करने के लिए

continue	Loop के next iteration में jump करने के लिए
def	Function define करने के लिए
return	Function से value return करने के लिए
class	Class create करने के लिए
import	Module import करने के लिए
from	Module के specific part को import करने के लिए
as	Module alias देने के लिए
try	Exception handling के लिए
except	Exception catch करने के लिए
finally	Exception के बाद हमेशा execute होने वाला block
pass	Empty statement
with	Context manager के लिए

Note: Python के हर version में कुछ new keywords add हो सकते हैं।

How to Check All Python Keywords:

Python में **keyword module** use करके सभी keywords देख सकते हैं।

```
import keyword
print(keyword.kwlist)
print("Total Keywords:", len(keyword.kwlist))
```

Example of Using Keywords:

```
# if-else keyword example
age = 18
if age >= 18:
   print("Adult")
else:
   print("Minor")

# for loop keyword example
for i in range(1, 6):
   print(i)

# def and return keyword example
def add(a, b):
   return a + b

print(add(5, 3))  # Output: 8
```

Operators in - Python में **Operators** programming का fundamental concept है। Operators का use **variables और values के बीच operations perform** करने के लिए किया जाता है।

Definition:

Operators वो symbols हैं जो Python interpreter को बताते हैं कि **दो values के बीच कौन-सा operation करना है**।

- Example: +, -, *, /, %
- Operators **expressions** create करने के लिए use होते हैं।

Syntax:

```
result = a + b
```

Types of Operators in Python:

Python में operators को **7 main categories** में divide किया जा सकता है:

1. Arithmetic Operators (गणितीय operators)

- Numbers के साथ **mathematical operations** perform करते हैं।

Operator	Description	Example	Output
+	Addition	5 + 3	8
-	Subtraction	5 - 3	2
*	Multiplication	5 * 3	15
/	Division (float)	5 / 2	2.5
//	Floor Division (integer division)	5 // 2	2
%	Modulus (remainder)	5 % 2	1
**	Exponent	2 ** 3	8

Example:

```
a = 10
b = 3
print(a + b)  # 13
print(a ** b) # 1000
```

2. Comparison / Relational Operators (तुलनात्मक operators)

- दो values की तुलना करते हैं और **Boolean (True/False)** result देते हैं।

Operator	Description	Example	Output

==	Equal to	5 == 5	True
!=	Not equal	5 != 3	True
>	Greater than	5 > 3	True
<	Less than	5 < 3	False
>=	Greater or equal	5 >= 5	True
<=	Less or equal	5 <= 3	False

Example:

```
x = 10
y = 20
print(x < y)   # True
print(x != y)  # True
```

3. Logical Operators (तार्किक operators)

- Boolean values के साथ **logical operations** perform करते हैं।

Operator	Description	Example	Output
and	Returns True if both conditions are True	True and False	False
or	Returns True if any one condition is True	True or False	True
not	Reverses the Boolean value	not True	False

Example:

```
a = True
b = False
print(a and b)  # False
print(a or b)   # True
print(not a)    # False
```

4. Assignment Operators (आवंटन operators)

- Variable को **value assign** करने और modify करने के लिए use होते हैं।

Operator	Description	Example
=	Assigns value to a variable	x = 5
+=	Adds and assigns	x += 3 → (same as x = x + 3)
-=	Subtracts and assigns	x -= 2 → (same as x = x - 2)
*=	Multiplies and assigns	x *= 4
/=	Divides and assigns	x /= 2
%=	Modulus and assigns	x %= 3
**=	Exponent and assigns	x **= 2
//=	Floor divides and assigns	x //= 2

Example:

```
x = 10
x += 5
print(x)  # 15
```

5. Bitwise Operators (बाइनरी operators)

- Integers के **binary representation** पर operate करते हैं।

Operator	Description	Example	Output
&	Bitwise AND	5 & 3	1
|	Bitwise OR	5 | 3	7
^	Bitwise XOR	5 ^ 3	6
~	Bitwise NOT	~5	-6
<<	Left Shift	5 << 1	10
>>	Right Shift	5 >> 1	2

6. Membership Operators (सदस्यता operators)

- Sequence में किसी **element की presence** check करते हैं।

Operator	Description	Example	Output
in	Returns True if a value is present in a sequence	'a' in 'apple'	True
not in	Returns True if a value is not present in a sequence	'b' not in 'apple'	True

Example:

```
fruits = ["apple", "banana"]
print("apple" in fruits)      # True
print("mango" not in fruits)  # True
```

7. Identity Operators (पहचान operators)

- Check करते हैं कि **दो variables same object को reference** कर रहे हैं या नहीं।

Operator	Description	Example
is	Returns True if both variables refer to the same object in memory	x is y

is not	Returns True if both variables refer to different objects in memory	x is not y

Example:

```
x = [1,2]
y = x
z = [1,2]
print(x is y)   # True
print(x is z)   # False
```

Python में **input और output** operations के लिए built-in functions use किए जाते हैं। सबसे common functions हैं: **input(), raw_input() और print()**।

1. input() Function

Definition:

- Python 3 में **input() function** user से data **console/keyboard** से लेने के लिए use होता है।
- यह function **user द्वारा entered value को string type** में return करता है।

Syntax:

```
variable = input("Prompt message: ")
```

Example:

```
name = input("Enter your name: ")
print("Hello,", name)
```

Output:

```
Enter your name: Nuresh
Hello, Nuresh
```

Important Points:

1. User input हमेशा **string** type में return होता है।
2. अगर numeric input चाहिए तो **type casting** करनी पड़ती है।

```
age = int(input("Enter your age: "))
print("Your age is", age)
```

2. raw_input() Function

Note: raw_input() **Python 2** में use होता था।
Python 3 में raw_input() remove कर दिया गया और उसका काम **input() function** करता है।

Definition:

- Python 2 में raw_input() function **keyboard से input** लेता है और **string type** में return करता है।

Example in Python 2:

```
name = raw_input("Enter your name: ")
print "Hello,", name
```

Output:

```
Enter your name: Nuresh
Hello, Nuresh
```

- Python 2 में **input() function** actually input को **evaluate** करता था, मतलब अगर user number type करता तो number का type detect होता।

3. print() Function

Definition:

- Python में **print() function** का use **output display करने के लिए** किया जाता है।
- Python 2 में print statement थी, Python 3 में इसे **function** बना दिया गया है।

Syntax:

```
print(object(s), sep=' ', end='\n')
```

Parameters:

1. object(s) – output के लिए value(s)
2. sep – multiple objects को separate करने के लिए (default: space)
3. end – output के end में कौन सा character लगे (default: newline \n)

Example:

```
# Single value
print("Hello, World!")

# Multiple values
x = 10
y = 20
print("Sum of x and y:", x + y)
```

```
# Using sep and end
print("Python", "Programming", sep="-", end="!")
```

Output:

```
Hello, World!
Sum of x and y: 30
Python-Programming!
```

Unit-2
Control Structure

Control Flow Statements in Python

Python में control flow statements ऐसे statements होते हैं जो यह तय करते हैं कि program का execution किस direction में या किस order में होगा। मतलब — कौन सा block of code कब और कितनी बार चलेगा।

Types of Control Flow Statements

Python में मुख्यतः तीन प्रकार के control flow statements होते हैं:

- Conditional Statements/Decision-Making Statement (निर्णयात्मक कथन)
- Looping Statements/iteration Statements (दोहराव वाले कथन)
- Jump Statements/branching Statements (जम्प या transfer statements)

1. Conditional Statements

ये statements किसी condition (शर्त) पर depend करते हैं। अगर condition true है तो एक block चलेगा, नहीं तो दूसरा।

Syntax:

```
if condition:
    statement1
elif another_condition:
    statement2
else:
    statement3
```

Example:

```
x = 10
if x > 0:
    print("Positive number")
elif x == 0:
    print("Zero")
else:
    print("Negative number")
```

Explanation:

अगर x positive है तो पहला block execute होगा,
अगर x zero है तो दूसरा,
अन्यथा तीसरा block।

2. Looping Statements

ये statements किसी block of code को बार-बार चलाने के लिए इस्तेमाल किए जाते हैं, जब तक कोई condition पूरी होती है।

Python में दो main loops हैं:

(a) for loop

किसी sequence (जैसे list, tuple, string) पर iterate करने के लिए।

```
for i in range(5):
    print(i)
```

Output:

0

1

2

3

4

(b) while loop

जब तक कोई condition true रहती है, तब तक चलता रहता है।

```
i = 1
while i <= 5:
    print(i)
    i += 1
```

3. Jump Statements

ये statements loop या condition के normal flow को बदल देते हैं।

(a) break

Loop को तुरंत रोक देता है।

```
for i in range(10):
    if i == 5:
        break
    print(i)
```

Output: 0 1 2 3 4

(b) continue

Current iteration को छोड़कर अगले पर चला जाता है।

```
for i in range(5):
    if i == 2:
        continue
    print(i)
```

Output: 0 1 3 4

(c) pass

कुछ नहीं करता, बस placeholder की तरह काम करता है।

```
for i in range(5):
    pass  # to be implemented later
```

Decision Making Statements

Python में Decision Making Statements का उपयोग किसी condition (शर्त) के आधार पर यह तय करने के लिए किया जाता है कि कौन-सा code block execute होगा। यानी अगर कोई condition True होती है, तो एक particular statement चलेगा, वरना दूसरा।

Program का control flow इन statements के जरिए conditionally change किया जा सकता है।

Types of Decision Making Statements in Python

Python में मुख्यतः चार प्रकार के Decision Making Statements होते हैं:

1. if statement – यह केवल तब execute होता है जब दी गई condition True हो।
2. if-else statement – अगर condition True है तो एक block execute होगा, वरना दूसरा।
3. if-elif-else ladder – इसका उपयोग तब किया जाता है जब कई conditions को check करना हो।
4. nested if statement – जब एक if के अंदर दूसरा if statement होता है।

1. if Statement (Simple if Statement)

if statement का उपयोग किसी condition को check करने के लिए किया जाता है। अगर condition True होती है तो statement execute होता है, अगर condition False होती है तो statement skip कर दिया जाता है।

यह सबसे basic और fundamental decision-making statement है।

Syntax:

```
if condition:
    statement(s)
```

condition: कोई logical या comparison expression जो True या False return करता है।

अगर condition True है → indented block execute होता है।

अगर False है → कुछ नहीं होता।

Working:

Python सबसे पहले condition को evaluate करता है।

अगर result True है → नीचे का indented code block run होगा।

अगर result False है → program control if block को छोड़ देगा।

Example:

```
x = 10
if x > 5:
    print("x is greater than 5")
```

Output:

```
x is greater than 5
```

Explanation:

यहाँ condition x > 5 True है, इसलिए message print हुआ।
अगर x = 3 होता तो कुछ भी print नहीं होता।

Uses / Advantages:

- जब केवल एक simple condition check करनी हो।
- Small decision-based programs में useful होता है।
- Login verification, validation आदि जैसे tasks में उपयोगी।

2. if-else Statement

if-else statement दो alternatives देता है —
अगर condition True है → if block execute होगा,
अन्यथा → else block execute होगा।

यह binary decision (Yes/No type) situations में उपयोग होता है।

Syntax:

```
if condition:
    statement(s)
else:
    statement(s)
```

Working:

Condition evaluate होती है।

अगर True → if block execute।

अगर False → else block execute।

Example:

```
num = int(input("Enter a number: "))

if num % 2 == 0:
    print("Even Number")
else:
    print("Odd Number")
```

Output (for input 7):

Odd Number

Explanation:

num % 2 == 0 condition False है, इसलिए else block चला।

Uses / Advantages:

- जब हमें दो options में से एक चुनना हो।
- जैसे — Pass/Fail, Yes/No, True/False decision।
- Simple decision-based branching के लिए helpful।

3. if-elif-else Statement

if-elif-else statement का उपयोग तब किया जाता है जब हमें multiple conditions check करनी हों।
Program ऊपर से नीचे तक हर condition check करता है और पहली True condition पर रुक जाता है।

Syntax:

```
if condition1:
    statement1
elif condition2:
    statement2
elif condition3:
    statement3
else:
    statement4
```

Working:

सबसे पहले if condition check होती है।

अगर True है → वही block execute होता है।

अगर False है → अगली elif condition check होती है।

अगर कोई भी True नहीं हुई → else block execute होता है।

Example:

```
marks = int(input("Enter your marks: "))


if marks >= 85:
    print("Grade A")
elif marks >= 70:
    print("Grade B")
elif marks >= 50:
    print("Grade C")
else:
    print("Fail")
```

Output (for input 72):

Grade B

Explanation:

पहली condition (marks >= 85) False,

दूसरी (marks >= 70) True,
इसलिए "Grade B" print हुआ और बाकी conditions check नहीं की गईं।

Uses / Advantages:

- Multiple decision making में helpful।
- Grading system, Menu-based program आदि में उपयोगी।
- Code को साफ और logical बनाता है।

4. Nested if Statement

जब किसी if block के अंदर एक और if statement लिखा जाता है,
तो उसे Nested if Statement कहते हैं।

यह तब उपयोग होता है जब हमें एक decision के अंदर दूसरा decision लेना हो।

Syntax:

```
if condition1:
    if condition2:
        statement(s)
    else:
        statement(s)
else:
    statement(s)
```

Working:

- Outer if की condition check होती है।
- अगर True है → अंदर वाले (inner if) की condition check होती है।
- अगर दोनों True हैं → inner block execute होता है।
- अगर कोई False है → respective else block चलता है।

Example:

```
x = 10
y = 20

if x > 0:
    if y > 10:
        print("Both x and y are positive and y is greater than 10")
    else:
        print("x positive but y not greater than 10")
```

```
else:
    print("x is not positive")
```

Output:

Both x and y are positive and y is greater than 10

Explanation:

Outer if (x > 0) True और inner if (y > 10) भी True,
इसलिए पहला print statement चला।

Uses / Advantages:

- Complex conditions में उपयोगी।
- जब decision multiple levels पर depend करता हो।
- जैसे eligibility check (age + citizenship) आदि में।

5. Short Hand if / if-else Statement

Python में अगर condition छोटी हो, तो उसे एक ही line में लिखा जा सकता है।
इसे Short Hand if Statement या Ternary Expression भी कहते हैं।

Syntax (for short if):

if condition: statement

Syntax (for short if-else):

statement1 if condition else statement2

Example 1 (short if):

```
x = 5
if x > 0: print("Positive number")
```

Example 2 (short if-else):

```
a = 10
b = 20
print("a is greater") if a > b else print("b is greater")
```

Output:

```
b is greater
```

Uses / Advantages:

- Code को concise (छोटा) बनाता है।
- Simple conditions के लिए perfect।
- One-liner statements या small scripts में helpful।

Looping Statements in Python

Looping statements वो होते हैं जिनसे हम किसी block of code को बार-बार (repeatedly) execute कर सकते हैं जब तक कोई condition True रहती है।
Python में मुख्यतः दो प्रकार के looping statements होते हैं —

- while loop
- for loop

1. while Loop

while loop तब तक चलता है जब तक दी गई condition True रहती है।
जैसे ही condition False हो जाती है, loop रुक जाता है।

Syntax:

```
while condition:
    # body of loop
```

Example:

```
i = 1
while i <= 5:
    print("Hello Python")
    i = i + 1
```

Output:

```
Hello Python
Hello Python
Hello Python
Hello Python
Hello Python
```

Explanation:

सबसे पहले i = 1

condition i <= 5 True है, तो print होता है

हर बार i बढ़ता है (i = i + 1)

जब i 6 हो जाता है, condition False → loop रुक जाता है।

2. for Loop

for loop किसी sequence (जैसे list, string, tuple, range आदि) के हर element पर iterate करता है।

Syntax:

```
for variable in sequence:
    # body of loop
```

Example:

```
for i in range(1, 6):
    print("Number:", i)
```

Output:

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
```

Explanation:

range(1,6) → 1 से 5 तक numbers देता है
हर बार i की value बदलती है और loop तब तक चलता है जब तक sequence खत्म न हो जाए।

3. Nested Loop

जब एक loop के अंदर दूसरा loop होता है, उसे nested loop कहते हैं।

Syntax:

```
for i in range(1, 4):
    for j in range(1, 3):
        print(i, j)
```

Output:

```
1 1
1 2
2 1
```

2 2

3 1

3 2

Explanation:

बाहरी loop (i) हर बार चलता है और उसके अंदर वाला (j) हर बार पूरी तरह चलता है। इससे हमें table या pattern बनाने में मदद मिलती है।

While loop

while loop एक indefinite loop है जो तब तक एक block को repeatedly execute करता है जब तक दी गई condition True रहती है। यानी iteration की संख्या पहले से निश्चित नहीं होती; loop की termination runtime condition पर निर्भर करती है।

Syntax:

```
while condition:
    statement(s)
```

Working:

- सबसे पहले condition evaluate की जाती है।
- यदि condition True है → indent किए हुए statements execute होते हैं।
- statements पूरे होने के बाद control फिर से condition पर वापस जाता है।
- जब condition False हो जाती है → loop terminate हो जाता है और control loop के बाद वाले statement पर चला जाता है।

Example:

```
n = 5
i = 1
sum = 0
while i <= n:
    sum += i       # sum = sum + i
```

```
    i += 1          # increment to avoid infinite loop
print("Sum of first", n, "natural numbers is", sum)
```

Output:

Sum of first 5 natural numbers is 15

Explanation:

शुरुआत में i = 1, condition i <= 5 True → sum में 1 जोड़ा गया।

हर iteration के बाद i को i += 1 से बढ़ाया जाता है। यह increment बहुत जरूरी है वरना loop infinite बन जाएगा।

जब i = 6 हो जाता है, condition False और loop खत्म। अंत में accumulated sum print होता है।

Advantages / Use-cases:

- जब iterations की संख्या unknown हो और termination किसी condition पर depend करे (उदाहरण: user input से, sentinel value, file read until EOF)।
- menu-driven programs, socket/server loops, polling आदि में उपयोगी।

Common pitfalls & best practices:

- Infinite loop का खतरा — ensure करें कि loop body में condition को बदलने वाला code मौजूद हो।
- यदि external input पर depend कर रहे हों तो validation रखें।
- break/continue का सही इस्तेमाल करें: break से loop तुरंत बंद होगा; continue current iteration skip कर अगले पर जाएगा।

for loop

for loop एक definite loop / iterator-based loop है जो किसी sequence/iterable (जैसे list, tuple, string, range, dict इत्यादि) के हर element पर एक-एक करके iterate करता है। यहाँ iterations की संख्या sequence पर निर्भर करती है और प्रायः पहले से ज्ञात होती है।

Syntax:

```
for variable in sequence:
    statement(s)
```

Working:

- Python sequence से पहला element लेकर उसे variable में assign करता है।
- indented block execute होता है।
- अगले element पर जाता है और step 2 repeat होता है।
- जब sequence खत्म हो जाता है, loop terminate होता है और control loop के बाद के statements पर चला जाता है।

उदाहरण 1 — range के साथ (common use):

```
for i in range(1, 6):   # 1 से 5
    print("Square of", i, "is", i*i)
```

Output:

```
Square of 1 is 1
Square of 2 is 4
Square of 3 is 9
Square of 4 is 16
Square of 5 is 25
```

उदाहरण 2 — list के साथ:

```
fruits = ["apple", "banana", "mango"]

for fruit in fruits:
    print("I like", fruit)
```

Output:

```
I like apple

I like banana

I like mango
```

Explanation:

- range(1,6) iterable है; for loop हर बार i को next value देता और body execute करता।
- जब iterable समाप्त हो जाता है, loop अपने आप रुक जाता है — programmer को manually increment करने की आवश्यकता नहीं होती।

Advantages / Use-cases:

- Sequences, collections, और iterators पर simple और readable iteration।
- indexing की आवश्यकता नहीं जब तक index चाहिए हो (तब enumerate() उपयोग करें)।
- list comprehension के साथ combined होकर concise code बनता है।

Common pitfalls & best practices:

- loop variable को loop के बाहर mutate न करें ताकि side-effects से बचा जा सके।
- यदि आप loop में sequence modify कर रहे हैं (जैसे list से elements remove करना), तो unexpected behavior आ सकता है — ऐसी स्थिति में copy पर iterate करें या list comprehension का उपयोग करें।
- heavy nested loops performance प्रभावित कर सकते हैं; complexity का ध्यान रखें।

nested loop

Nested loops वे loops होते हैं जिनमें एक loop के अंदर दूसरा loop (या अनेक) लिखा होता है। हर बार outer loop की एक iteration पर inner loop पूरी तरह execute होता है। Nested loops का उपयोग multi-dimensional data (जैसे matrices), pattern printing, और Cartesian product calculations के लिए होता है।

Syntax:

```
for outer in outer_sequence:
    for inner in inner_sequence:
        statement(s)
```

(यहाँ for के स्थान पर while भी हो सकता है — nested while/for-while combinations वैध हैं।)

Working:

- Outer loop का पहला iteration शुरू होता है।
- Inner loop पूरी तरह से execute होता है (inner sequence के सभी elements के लिए)।
- Inner loop समाप्त होने पर control outer loop के next iteration पर आता है।
- यह प्रक्रिया तब तक चलती है जब outer sequence समाप्त हो जाए।

उदाहरण 1 — multiplication table (2D iteration):

```
for i in range(1, 4):      # outer: 1 to 3
    for j in range(1, 4):  # inner: 1 to 3
        print(i, "x", j, "=", i*j)
    print("---")
```

Output:

```
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
---
2 x 1 = 2
```

```
2 x 2 = 4
2 x 3 = 6
---
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
---
```

उदाहरण 2 — pattern printing (triangular pattern):

```
n = 4
for i in range(1, n+1):
    for j in range(i):
        print("*", end="")
    print()
```

Output:

```
*
**
***
****
```

Explanation:

- प्रत्येक outer iteration के लिए inner loop j की पूरी range पर चलता और एक row/line बनता।
- Outer loop rows को नियंत्रित करता है और inner loop columns/characters को बनाता है।

Advantages / Use-cases:

- 2D arrays / matrices के elements पर iteration (matrix multiplication, traversal)।
- pattern designing, nested data structures processing (list of lists), combinatorial problems (Cartesian product)।

Complexity और pitfalls:

- Time Complexity: अगर outer loop m बार और inner loop n बार चलता है तो कुल iterations ≈ m * n होंगे; इसलिए nested loops आसानी से time-consuming बन जाते हैं। बड़े data पर performance का ध्यान रखें।
- Memory और CPU का careful management जरूरी है; जहाँ possible हो, algorithmic optimization (e.g., flattening, vectorization, using library functions like NumPy) पर विचार करें।
- अधिक nesting (3+ levels) से code readability कम होती है — ऐसे में function में logic निकालना (refactor) बेहतर रहता है।

Jumping Statements

Jumping statements वे statements होते हैं जो loop या program के normal flow of execution को बदल देते हैं।
इनका प्रयोग तब किया जाता है जब हमें loop को छोड़ना (exit), skip करना (continue) या अस्थायी रूप से खाली रखना (pass) हो।

Types of Jumping Statements in Python:

- break statement
- continue statement
- pass statement

1. break Statement

break statement का उपयोग loop को अचानक समाप्त (terminate) करने के लिए किया जाता है, चाहे condition True ही क्यों न हो।

Syntax:

```
for variable in sequence:
    if condition:
        break
    statement(s)
```

Example:

```
for i in range(1, 10):
    if i == 5:
        break
    print(i)
```

Output:

```
1
2
3
4
```

Explanation:

जब i == 5 हुआ, break statement ने loop को बीच में ही रोक दिया।
बाकी iterations नहीं चले।

Uses:

- किसी specific condition पर loop को बीच में रोकने के लिए।
- Searching algorithms या input validation में, जब desired result मिल जाए।

2. continue Statement

continue statement loop की current iteration को skip कर अगले iteration पर चला जाता है।

Syntax:

```
for variable in sequence:
    if condition:
        continue
    statement(s)
```

Example:

```
for i in range(1, 6):
    if i == 3:
        continue
    print(i)
```

Output:

1

2

4

5

Explanation:

जब i == 3, उस iteration में print() statement skip हुआ और control अगली iteration (i=4) पर चला गया।

Uses:

- जब कुछ particular values या conditions को ignore करना हो।
- Input filtering या skipping invalid data के लिए उपयोगी।

3. pass Statement

pass एक null statement है — यह कुछ नहीं करता।
इसे placeholder के रूप में लिखा जाता है ताकि syntax error न आए जब code बाद में लिखा जाना हो।

Syntax:

```
if condition:
    pass
```

Example:

```
for i in range(5):
    if i == 2:
        pass
    else:
        print("Number:", i)
```

Output:

```
Number: 0
Number: 1
Number: 3
Number: 4
```

Explanation:

जब i == 2, pass statement execute हुआ लेकिन कुछ नहीं किया।
Program बिना error के अगले iteration पर चला गया।

Uses:

- जब किसी loop, function, या class का structure लिखना हो लेकिन body अभी define न की गई हो।
- "To be implemented later" situations में helpful।

break Statement

break statement का उपयोग loop को अचानक (prematurely) terminate करने के लिए किया जाता है।

जब Python interpreter को break मिलता है, तब वो तुरंत loop से बाहर निकल जाता है, और control loop के बाद की statement पर चला जाता है।

Normally, loops (like for or while) तब तक चलते हैं जब तक उनकी condition True रहती है।

लेकिन अगर किसी special condition पर loop को बीच में ही रोकना हो, तो हम break statement का प्रयोग करते हैं।

यह अक्सर तब useful होता है जब हमें किसी specific condition मिलने पर loop को रोकना हो, जैसे searching operation या किसी condition match होने पर।

Syntax:

```
for variable in sequence:
    statement
    if condition:
        break
    statement
```

while loop के साथ:

```
while condition:
    statement
    if condition2:
        break
    statement
```

Example 1: Using break in for loop

```
for i in range(1, 11):
    if i == 5:
        break
    print(i)
```

Output:

```
1
2
3
4
```

यहाँ loop normally 1 से 10 तक चलना चाहिए था,
लेकिन जब i == 5 हुआ, तब break execute हुआ और loop turant terminate हो गया।

Example 2: Using break in while loop

```
i = 1
while i <= 10:
    if i == 6:
        break
    print(i)
    i += 1
```

Output:

```
1
2
3
4
5
```

जैसे ही i == 6 हुआ, loop break हो गया और आगे नहीं चला।

continue Statement

continue statement का उपयोग किसी particular iteration को skip करने के लिए किया जाता है।

यह loop को नहीं रोकता, बल्कि उस iteration को छोड़कर next iteration पर jump कर जाता है।

जब continue statement execute होता है, तो Python current iteration की remaining statements को skip कर देता है और सीधे loop की next iteration पर चला जाता है। यह तब useful है जब किसी condition पर हम कुछ specific values को skip करना चाहते हैं।

Syntax:

```
for variable in sequence:
    if condition:
        continue
    statement
```

Example 1: Using continue in for loop

```
for i in range(1, 6):
    if i == 3:
        continue
    print(i)
```

Output:

```
1
2
4
5
```

यहाँ जब i == 3 हुआ, तब continue ने उस iteration को skip कर दिया,
इसलिए 3 print नहीं हुआ।

Example 2: Using continue in while loop

```
i = 0
while i < 5:
    i += 1
    if i == 2:
        continue
    print(i)
```

Output:

```
1
3
4
5
```

यहाँ जब i == 2 हुआ, तब continue ने print(i) को skip किया, इसलिए 2 print नहीं हुआ।

pass Statement

pass statement एक null operation statement है।
यह कुछ नहीं करता — बस एक placeholder की तरह काम करता है ताकि syntax error ना आए।

कभी-कभी प्रोग्राम बनाते समय हमें किसी function, loop या condition का structure बनाना होता है, लेकिन logic बाद में लिखना होता है। ऐसे में अगर हम body खाली छोड़ दें, तो Python error देता है।

इसलिए वहाँ हम pass statement लिख देते हैं जो कुछ नहीं करता, बस interpreter को बताता है कि "यहाँ future में code आएगा।"

Syntax:

```
if condition:
    pass
```

या

```
for i in range(5):
    pass
```

Example 1: In if statement

```
x = 10
if x > 5:
    pass
else:
    print("x is small")
```

Output:

(No output, because pass did nothing)

यहाँ if block खाली था, लेकिन हमने pass लिखा ताकि error न आए।

Example 2: In loop

```
for i in range(3):
    pass
print("Loop executed successfully")
```

Output:

Loop executed successfully

Loop चला लेकिन कुछ काम नहीं किया क्योंकि pass कुछ नहीं करता।

Difference between break and continue Statement

Basis of Difference	break Statement	continue Statement
Definition	break statement का उपयोग loop को completely terminate (रोकने) के लिए किया जाता है।	continue statement का उपयोग loop की current iteration को skip करके next iteration पर जाने के लिए किया जाता है।
Functionality / Working	जब break execute होता है, तो loop तुरंत खत्म हो जाता है और control loop के बाहर की statement पर चला जाता है।	जब continue execute होता है, तो loop की उस iteration की बाकी statements skip हो जाती हैं, और control loop के next iteration पर चला जाता है।
Control Transfer	Control loop के बाहर चला जाता है।	Control loop के अंदर ही रहता है, अगली iteration पर चला जाता है।
Use Case / Purpose	तब use करते हैं जब किसी specific condition पर loop को पूरी तरह से रोकना हो।	तब use करते हैं जब किसी specific condition पर कुछ iterations को skip करना हो।
Loop Execution	Loop execution terminate हो जाता है, और आगे नहीं चलता।	Loop execution continue रहता है, बस current iteration skip हो जाती है।
Effect on Loop Counter	break के बाद loop counter का कोई महत्व नहीं रहता क्योंकि loop समाप्त हो जाता है।	continue loop counter को increment/decrement होने देता है, और loop चलता रहता है।
Placement / Syntax	आमतौर पर if condition के अंदर लगाया जाता है ताकि किसी condition पर loop रोका जा सके।	आमतौर पर if condition के अंदर लगाया जाता है ताकि किसी condition पर iteration skip की जा सके।

Example (for loop)	for i in range(1,6):
	  if i==3:
	  break
	  print(i)
	
	Output: 1 2
		for i in range(1,6):
		  if i==3:
		  continue
		  print(i)
		
		Output: 1 2 4 5

Output Behavior	Output वहीं तक print होता है जहाँ तक break execute नहीं हुआ।	Output में वो values print होती हैं जिन पर continue condition false रही (skipped value missing होती है)।
Common Usage	Searching में जब desired element मिल जाए तब break से loop रोक देते हैं।	Filtering में जब किसी unwanted value को skip करना हो तब continue use करते हैं।

Unit-3
Sequence, Dictionary & Built in Function

Sequence Datatype- Sequence Datatype Python का एक ऐसा datatype होता है जिसमें multiple values / elements को एक specific order (क्रम) में store किया जाता है। यानि इसमें data एक के बाद एक sequence में रखा जाता है और हर element की अपनी position (index) होती है।

Ordered Data Structure
Sequence datatype में data हमेशा order में store होता है।
जो element पहले add किया गया, उसका index छोटा होगा और बाद में add किए गए elements का index बढ़ता जाएगा।

Indexing Supported
Sequence datatype में हर element को index number से access किया जा सकता है।
Index हमेशा 0 से start होता है
Negative indexing भी possible होती है (जैसे -1 last element को refer करता है)

Multiple Data Store करने की Capability
Sequence datatype एक साथ एक से ज्यादा values store कर सकता है।
ये values:
Same datatype की हो सकती हैं
Different datatype की भी हो सकती हैं (number, string, boolean आदि)

Iteration Possible (Loop Support)
Sequence datatype पर for loop या while loop आसानी से लगाया जा सकता है।
क्योंकि data ordered होता है, इसलिए element-by-element traversal possible होता है।

Slicing Feature Available
Sequence datatype में slicing का concept होता है, जिससे हम sequence का कोई specific part निकाल सकते हैं।
उदाहरण: start index से end index तक का data

Length Known होती है
Sequence datatype की length निकाली जा सकती है, यानी उसमें कितने elements हैं यह पता किया जा सकता है।

Common Operations Supported
Sequence datatype पर कई common operations possible होते हैं, जैसे:

- Concatenation (दो sequence जोड़ना)
- Repetition (sequence को repeat करना)
- Membership checking (in, not in)

Python में Sequence Datatype के Types

1. String
2. List
3. Tuple

String - String Python का एक sequence datatype है, जिसका उपयोग text / characters के group को store करने के लिए किया जाता है।
String में हर character का अपना index position होता है और string ordered होती है।

Python में string को single quotes ' ', double quotes " " या triple quotes ''' ''' / **""" """** में लिखा जाता है।

String कैसे Create करते हैं? (Creation of String)

Python में string create करने के तरीके:

```
name = "Python"

city = 'Bilaspur'

msg = """Welcome to Python Programming"""
```

Triple quotes का उपयोग **multi-line string** के लिए किया जाता है।

String को Access कैसे करते हैं? (Accessing String)

(a) Indexing द्वारा

String के हर character को index से access किया जाता है।

```
s = "Python"

print(s[0])   # P

print(s[3])   # h

print(s[-1])  # n
```

Positive index left से start होता है (0 से)
Negative index right से start होता है (-1 से)

(b) Slicing द्वारा

String का कोई part निकालने के लिए slicing use करते हैं।

```
s = "Programming"
print(s[0:6])   # Progra
print(s[3:])    # gramming
print(s[:7])    # Program
print(s[-4:])   # ming
```

String को Update कैसे करते हैं? (Updating String)

Python में string immutable होती है, यानी एक बार string create होने के बाद उसके characters को directly change नहीं किया जा सकता।

Wrong way:

```
s = "Python"
s[0] = 'J'   # Error
```

Correct way (New string बनाकर):

```
s = "Python"
s = "J" + s[1:]
print(s)   # Jython
```

यानी update करने के लिए नई string बनानी पड़ती है।

String के Built-in Functions (Methods)

1. len()

String की length बताता है।

```
s = "Python"
print(len(s))   # 6
```

2. lower()

String को lowercase में convert करता है।

```
print("PYTHON".lower())   # python
```

3. upper()

String को uppercase में convert करता है।

```
print("python".upper())   # PYTHON
```

4. capitalize()

String का first character capital बनाता है।

```
print("python".capitalize())   # Python
```

5. title()

हर word का पहला letter capital करता है।

```
print("python programming language".title())
# Python Programming Language
```

6. replace(old, new)

String के किसी part को replace करता है।

```
s = "I like Java"
print(s.replace("Java", "Python"))
```

7. find()

Substring की **starting index** return करता है।

```
s = "Programming"
print(s.find("gram"))   # 3
```

अगर value नहीं मिले तो -1 return करता है।

8. count()

Substring कितनी बार आया है, यह बताता है।

```
s = "banana"
print(s.count("a"))   # 3
```

List - List Python का एक sequence datatype है जिसका उपयोग multiple values / elements को एक ही variable में store करने के लिए किया जाता है। List ordered, mutable (changeable) होती है और इसमें different datatypes के elements store किए जा सकते हैंऔर यह duplicate values को भी hold कर सकती है।

Python में list को square brackets [ ] के अंदर define किया जाता है।

List Creation (List कैसे बनाते हैं)

List Python का एक mutable sequence datatype है, जिसमें multiple elements ordered form में store होते हैं।
List को square brackets [ ] के अंदर define किया जाता है और इसमें same या different datatype के elements हो सकते हैं।

```
a = [10, 20, 30]
b = ["Python", "Java", "C"]
c = [1, "Hello", 3.14, True]
empty_list = []
```

List ordered होती है और duplicate values allowed होती हैं।

Accessing List Elements (List को Access करना)

(a) Indexing द्वारा

List का हर element एक **index number** से access किया जाता है (index 0 से start).

```
lst = [10, 20, 30, 40]
print(lst[0])    # 10
print(lst[2])    # 30
print(lst[-1])   # 40
```

(b) Slicing द्वारा

List का कोई specific part निकालने के लिए slicing use की जाती है।

```
lst = [1, 2, 3, 4, 5]
print(lst[1:4])   # [2, 3, 4]
print(lst[:3])    # [1, 2, 3]
print(lst[3:])    # [4, 5]
```

Updating List Elements (List को Update करना)

List mutable होती है, इसलिए elements को directly change किया जा सकता है।

(a) Single Element Update

```
lst = [10, 20, 30]
lst[1] = 25
print(lst)   # [10, 25, 30]
```

(b) Multiple Elements Update (Slicing)

```
lst = [1, 2, 3, 4, 5]
lst[1:4] = [20, 30, 40]
print(lst)
```

Adding Elements in List (List में Element जोड़ना)

1. append()

List के end में single element add करता है।

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

2. insert()

Specific index position पर element add करता है।

```
lst = [10, 20, 30]
lst.insert(1, 15)
```

3. extend()

एक list के multiple elements दूसरी list में add करता है।

```
lst = [1, 2]
lst.extend([3, 4, 5])
```

Removing / Deleting Elements from List

1. remove()

Value के basis पर element delete करता है (first occurrence).

```
lst = [10, 20, 30, 20]
lst.remove(20)
print(lst)
```

2. pop()

Index के basis पर element delete करता है और deleted element को return भी करता है।

```
lst = [10, 20, 30]
x = lst.pop(1)
```

```
print(x)     # 20
print(lst)   # [10, 30]
```

pop() बिना index के last element delete करता है।

3. del keyword (Single Element)

Index के आधार पर element delete करता है।

```
lst = [10, 20, 30, 40]
del lst[2]
print(lst)
```

4. clear()

पूरी list को empty कर देता है लेकिन list object बना रहता है।

```
lst = [1, 2, 3]
lst.clear()
print(lst)   # []
```

5. Deleting the Entire List

पूरी list को memory से delete करने के लिए del keyword use किया जाता है।

```
lst = [10, 20, 30]
del lst
```

इसके बाद lst variable exist नहीं करता और access करने पर error आएगा।

List के Built-in Functions / Methods

len()

List में कुल कितने elements हैं, यह बताता है।

```
lst = [10, 20, 30, 40]
print(len(lst))   # 4
```

count()

किसी specific element की **frequency** बताता है।

```
lst = [1, 2, 2, 3, 2]
print(lst.count(2))   # 3
```

index()

किसी element का **index position** return करता है।

```
lst = [10, 20, 30]
print(lst.index(20))   # 1
```

अगर element list में नहीं है तो error आता है।

sort()

List के elements को **ascending order** में arrange करता है।

```
lst = [4, 2, 1, 3]
lst.sort()
print(lst)
```

Descending order:

```
lst.sort(reverse=True)
```

reverse()

List के elements का order **reverse** कर देता है।

```
lst = [1, 2, 3]
lst.reverse()
print(lst)
```

copy()

List की shallow copy बनाता है।

```
a = [10, 20, 30]
b = a.copy()
print(b)
```

max()

List का maximum value return करता है।

```
lst = [10, 50, 30]
print(max(lst))   # 50
```

min()

List का minimum value return करता है।

```
print(min(lst))   # 10
```

sum()

Numeric list के सभी elements का sum देता है।

```
lst = [10, 20, 30]
print(sum(lst))   # 60
```

sorted()

Original list को change किए बिना new sorted list return करता है।

```
lst = [3, 1, 2]
new_lst = sorted(lst)
print(new_lst)
print(lst)
```

Tuple - Tuple Python का एक sequence datatype है, जिसका उपयोग multiple elements को ordered form में store करने के लिए किया जाता है।
Tuple round brackets ( ) में define की जाती है और इसमें same या different datatype के elements हो सकते हैं।

Important Point:
Tuple immutable होती है, यानी tuple create होने के बाद उसके elements को change (update), insert या delete नहीं किया जा सकता।
Tuple duplicate values को allow करती है और indexing, slicing को support करती है।

Tuple Creation (Tuple कैसे बनाते हैं)

```
t1 = (10, 20, 30)
t2 = ("Python", "Java", "C")
t3 = (1, "Hello", 3.14, True)
single = (10,)      # single element tuple
empty = ()
```

Single element tuple के लिए comma **,** जरूरी होता है।

Accessing Tuple Elements (Tuple को Access करना)

(a) Indexing द्वारा

```
t = (10, 20, 30, 40)
print(t[0])   # 10
print(t[2])   # 30
print(t[-1])  # 40
```

(b) Slicing द्वारा

```
t = (1, 2, 3, 4, 5)
print(t[1:4])   # (2, 3, 4)
print(t[:3])    # (1, 2, 3)
print(t[3:])    # (4, 5)
```

Updating Tuple Elements (Tuple को Update करना)

Direct update possible नहीं है, क्योंकि tuple immutable होती है।

```
t = (10, 20, 30)
t[1] = 25      # Error
```

Correct Method (New tuple बनाकर):

```
t = (10, 20, 30)
t = (10, 25, 30)
print(t)
```

यानी update करने के लिए नई tuple create करनी पड़ती है।

Inserting Elements in Tuple (Tuple में Element Insert करना)

Tuple में direct insert / append possible नहीं है (immutable).

Workaround Method:

Tuple को पहले list में convert करते हैं, फिर element add करके वापस tuple बना देते हैं।

```
t = (10, 20, 30)
lst = list(t)
lst.append(40)
t = tuple(lst)
print(t)
```

Deleting Elements from Tuple

(a) Single Element Delete

Tuple से single element delete करना directly possible नहीं है।

Workaround:

```
t = (10, 20, 30)
lst = list(t)
lst.remove(20)
t = tuple(lst)
print(t)
```

(b) Deleting the Entire Tuple (पूरी Tuple Delete करना)

पूरी tuple को delete करने के लिए del keyword का उपयोग किया जाता है।

```
t = (10, 20, 30)
del t
```

इसके बाद t variable exist नहीं करता।

Tuple के Built-in Functions / Methods

1. len()

Tuple में total elements की संख्या बताता है।

```
t = (10, 20, 30)
print(len(t))
```

2. max()

Tuple का maximum element return करता है।

```
t = (10, 50, 30)
print(max(t))
```

3. min()

Tuple का minimum element return करता है।

```
print(min(t))
```

4. sum()

Numeric tuple का sum return करता है।

```
print(sum(t))
```

5. count()

Element कितनी बार आया है, यह बताता है।

```
t = (1, 2, 2, 3)
print(t.count(2))
```

6. index()

Element का index number return करता है।

```
t = (10, 20, 30)
print(t.index(20))
```

Difference between List and Tuple

Basis	List	Tuple
Definition	List Python का एक mutable sequence datatype है जो multiple elements को ordered form में store करता है।	Tuple Python का एक immutable sequence datatype है जो multiple elements को ordered form में store करता है।
Syntax	Square brackets [ ] का उपयोग होता है।	Round brackets ( ) का उपयोग होता है।
Mutability	Mutable होती है, यानी elements को change, add या delete किया जा सकता है।	Immutable होती है, यानी create होने के बाद elements को change, add या delete नहीं किया जा सकता।

Update Operation	List के elements को directly update किया जा सकता है।	Tuple के elements को directly update नहीं किया जा सकता।
Insertion / Deletion	Append, insert, remove, pop जैसे methods available होते हैं।	Direct insertion या deletion possible नहीं है।
Performance	List comparatively slower होती है क्योंकि mutable है।	Tuple comparatively faster होती है क्योंकि immutable है।
Memory Usage	List ज़्यादा memory consume करती है।	Tuple कम memory consume करती है।
Built-in Methods	List में methods की संख्या ज़्यादा होती है।	Tuple में methods limited होते हैं (count(), index() आदि)।
Data Modification Risk	Data accidentally change हो सकता है।	Data safe रहता है क्योंकि modification allowed नहीं।
Use Case	जब data frequently change करना हो।	जब data fixed / constant रखना हो।
Duplicate Values	Duplicate values allowed होती हैं।	Duplicate values allowed होती हैं।
Indexing & Slicing	Indexing और slicing supported है।	Indexing और slicing supported है।
Creation Speed	Creation थोड़ा slow होता है।	Creation थोड़ा fast होता है।
Example	lst = [10, 20, 30]	tup = (10, 20, 30)

Set - Set Python का एक unordered collection datatype है, जिसका उपयोग unique elements को store करने के लिए किया जाता है।
Set curly braces { } में define किया जाता है और इसमें duplicate values allowed नहीं होतीं।

Important Points:

1. Set unordered होता है (fixed index नहीं होता)
2. Set में indexing और slicing possible नहीं है
3. Set mutable होता है
4. Set में same datatype या different datatype के elements हो सकते हैं

Set Creation (Set कैसे बनाते हैं)

```
s1 = {10, 20, 30}

s2 = {"Python", "Java", "C"}

s3 = {1, "Hello", 3.14, True}
```

```
empty_set = set()     # empty set
```

{} empty dictionary बनाता है, empty set नहीं।

Accessing Set Elements (Set को Access करना)

Set unordered होता है, इसलिए indexing possible नहीं है।

Wrong:

```
print(s1[0])   # Error
```

Correct way – Loop का उपयोग:

```
for x in s1:
    print(x)
```

Membership check:

```
print(20 in s1)    # True
```

Updating Set Elements (Set को Update करना)

Set में direct update (replace) possible नहीं है, क्योंकि elements unordered होते हैं।

Direct update:

```
s1[0] = 100   # Error
```

Update का मतलब होता है:

- पुराने element को remove करना
- नया element add करना

```
s1.remove(10)

s1.add(100)
```

Adding Elements in Set (Addition)

1. add()

Set में single element add करता है।

```
s = {1, 2, 3}

s.add(4)
```

2. update()

Multiple elements (list, tuple, set) को add करता है।

s = {1, 2}

s.update([3, 4, 5])

Deleting Elements from Set (Deletion)

1. remove()

Specific element delete करता है।

s = {1, 2, 3}

s.remove(2)

Element न होने पर error देता है।

2. discard()

Element delete करता है लेकिन element exist नहीं करने पर error नहीं देता ।

s.discard(5)

3. pop()

Random element delete करता है।

s = {10, 20, 30}

s.pop()

4. clear()

Set को empty कर देता है।

s.clear()

5. Deleting the Entire Set

पूरे set को delete करने के लिए del keyword use होता है।

s = {1, 2, 3}

del s

Set के Built-in Functions / Methods

len(s) # number of elements

max(s) # maximum element

min(s) # minimum element

sum(s) # sum of elements (numeric)

Set Operations (Important Built-in Methods)

Set का सबसे बड़ा advantage यह है कि इसमें mathematical set operations directly perform किए जा सकते हैं। नीचे सबसे important aur commonly used set methods को detail में explain किया गया है।

union() – Union Operation

Union दो या दो से अधिक sets के सभी unique elements को combine करता है।
Common elements एक बार ही आते हैं।

Syntax:

```
set3 = set1.union(set2)
```

Example:

```
a = {1, 2, 3}

b = {3, 4, 5}


print(a.union(b))

# {1, 2, 3, 4, 5}
```

Original sets change नहीं होते।

intersection() – Intersection Operation

Intersection केवल common elements return करता है जो दोनों sets में मौजूद हों।

Syntax:

```
set3 = set1.intersection(set2)
```

Example:

```
a = {1, 2, 3}
b = {3, 4, 5}

print(a.intersection(b))
# {3}
```

Non-common elements remove हो जाते हैं।

difference() – Difference Operation

Difference पहले set के वो elements return करता है जो दूसरे set में नहीं होते।

Syntax:

```
set3 = set1.difference(set2)
```

Example:

```
a = {1, 2, 3}
b = {3, 4, 5}

print(a.difference(b))
# {1, 2}
```

a - b जैसा काम करता है।

symmetric_difference() – Symmetric Difference

ऐसे elements return करता है जो या तो पहले set में हों या दूसरे में,
लेकिन दोनों में common न हों।

Syntax:

```
set3 = set1.symmetric_difference(set2)
```

Example:

```
a = {1, 2, 3}

b = {3, 4, 5}


print(a.symmetric_difference(b))

# {1, 2, 4, 5}
```

Common element (3) remove हो जाता है।

Dictionary - Dictionary Python का एक built-in mapping datatype है, जिसमें data को key **:** value pair के form में store किया जाता है।
Dictionary curly braces { } में define की जाती है और हर key unique होती है, जबकि values duplicate हो सकती हैं।

Important Points:

- Dictionary ordered होती है (Python 3.7+ से)
- Dictionary mutable होती है
- Dictionary में indexing नहीं होती, access हमेशा key से होता है
- Keys immutable datatypes (int, str, tuple) होते हैं

Dictionary Creation (Dictionary कैसे बनाते हैं)

(a) Normal way

```
student = {
    "roll": 101,
    "name": "Amit",
    "marks": 85
}
```

(b) dict() function से

```
emp = dict(id=1, name="Rahul", salary=25000)
```

(c) Empty Dictionary

```
d = {}
```

Accessing Dictionary Elements (Access करना)

Dictionary में data key–value pair में store होता है, इसलिए access हमेशा key के through किया जाता है। नीचे main data access methods को detail में समझाया गया है।

1. get() Method

get() method का उपयोग किसी key की value safely access करने के लिए किया जाता है।

Syntax:

```
dict_name.get(key)
```

Example:

```
student = {"roll": 101, "name": "Amit", "marks": 85}


print(student.get("name"))   # Amit
print(student.get("age"))    # None
```

Important Points:

अगर key exist नहीं करती तो error नहीं देता

Default value भी दे सकते हैं

```
print(student.get("age", "Not Available"))
```

2. keys() Method

keys() method dictionary की सभी keys का view object return करता है।

```
print(student.keys())
```

Loop में use:

```
for k in student.keys():
    print(k)
```

3. values() Method

values() method dictionary की सभी values return करता है।

```
print(student.values())
```

Example:

```
for v in student.values():
    print(v)
```

4. items() Method

items() method dictionary के **key–value pairs** को tuple के form में return करता है।

```
print(student.items())
```

Loop के साथ:

```
for k, v in student.items():
    print(k, "=", v)
```

Updating Dictionary Elements (Update करना)

Existing key की value change करना update कहलाता है।

```
student["marks"] = 90
print(student)
```

Adding Elements in Dictionary (Addition)

New key : value pair add करने को addition कहते हैं।

(a) Direct Assignment

```
student["age"] = 20
```

(b) update() method

```
student.update({"city": "Bilaspur"})
```

Deleting Elements from Dictionary (Deletion)

1. del keyword

Specific key delete करता है।

```
del student["roll"]
```

2. pop()

Key के आधार पर element delete करता है और उसकी value return करता है।

```
x = student.pop("marks")
print(x)
```

3. popitem()

Last inserted item delete करता है।

```
student.popitem()
```

4. clear()

पूरी dictionary empty कर देता है।

```
student.clear()
```

5. Delete Entire Dictionary

पूरी dictionary delete करने के लिए del keyword।

```
del student
```

Other Common Dictionary Methods (अन्य महत्वपूर्ण Methods)

len() Function

Dictionary में total कितने key–value pairs हैं, यह बताता है।

```
print(len(student))
```

copy() Method

Dictionary की **shallow copy** बनाता है।

```
new_student = student.copy()
```

Original dictionary change नहीं होती।

Python calendar Module - Python में calendar module का उपयोग date, month, year से related information display करने और calendar-based operations करने के लिए किया जाता है।
इस module को use करने से पहले इसे import करना जरूरी है।

```
import calendar
```

Calendar Module का Use

- Month calendar print करना
- Year calendar print करना
- Leap year check करना
- Month / Year की details निकालना
- Weekday पता करना

calendar.month(year, month)

दिए गए year और month का calendar string format में return करता है।

```
import calendar

print(calendar.month(2025, 1))
```

Use case:

किसी specific month का calendar print करना

calendar.calendar(year)

पूरे year का calendar string format में return करता है।

```
print(calendar.calendar(2025))
```

पूरे साल का calendar देखने के लिए use होता है।

calendar.monthcalendar(year, month)

दिए गए month का calendar list of lists के form में return करता है।

```
print(calendar.monthcalendar(2025, 1))
```

Important points:

हर inner list = एक week

Monday से start होता है

जिस दिन date नहीं होती, वहाँ 0 होता है

Mostly programming logic में use होता है।

calendar.weekday(year, month, day)

दिए गए date का weekday number return करता है।

```
print(calendar.weekday(2025, 1, 1))
```

Output:

0 → Monday

1 → Tuesday

6 → Sunday

calendar.isleap(year)

Check करता है कि दिया गया year leap year है या नहीं।

```
print(calendar.isleap(2024))   # True
print(calendar.isleap(2025))   # False
```

Leap year:

4 से divisible

100 से divisible नहीं (except 400)

calendar.leapdays(y1, y2)

दो years के बीच कितने leap years हैं, यह return करता है।

```
print(calendar.leapdays(2000, 2025))
```

y1 include नहीं होता, y2 include होता है।

Python time module - Python का time module system time और date के साथ काम करने के लिए use होता है। यह time-based operations**,** delay, और timestamp calculation में helpful है।

```
import time
```

time.time()

Current time (Unix timestamp) return करता है।

Unix timestamp = 1 January 1970 (epoch) से अब तक seconds में time।

```
import time

print(time.time())
```

Output: float value (seconds including decimals)

time.localtime([seconds])

Given seconds का local time structure (struct_time) में return करता है।

```
t = time.localtime()

print(t)

print(t.tm_year, t.tm_mon, t.tm_mday)  # year, month, day
```

Default: time.time() का current time
tm_hour, tm_min, tm_sec भी access कर सकते हैं

time.ctime([seconds])

Seconds का human-readable format return करता है।

```
print(time.ctime())
```

Without argument → current time
With argument → specific timestamp

time.sleep(seconds)

Program को delay / pause करने के लिए use होता है।

```
print("Start")
time.sleep(5)  # 5 seconds wait
print("End")
```

Use case: Animation, countdown, rate limiting

range() Function - range() function का use sequence of numbers generate करने के लिए होता है। यह immutable sequence return करता है। Mostly loops में use किया जाता है, जैसे for loop।

Syntax Variants:

```
range(stop)
range(start, stop)
range(start, stop, step)
```

Parameters:

Parameter	Description	Parameter
start	Sequence का first number (default = 0)	start
stop	Sequence का last number **exclude** होता है	stop
step	Increment / decrement value (default = 1)	step

Examples:

1. Single Argument (stop)

```
for i in range(5):
    print(i)
```

Output:

0 1 2 3 4

0 से start, 5 तक (5 exclude), step = 1

2. Two Arguments (start, stop)

```
for i in range(3, 8):
    print(i)
```

Output:

3 4 5 6 7

3. Three Arguments (start, stop, step)

```
for i in range(2, 11, 2):
    print(i)
```

Output:

2 4 6 8 10

Step = 2 → हर next number 2 increase

Negative step possible:

```
for i in range(10, 0, -2):
    print(i)
```

Output:

10 8 6 4 2

Key Points:

- range() list नहीं return करता, बल्कि range object return करता है।
- Convert to list: list(range(5)) → [0, 1, 2, 3, 4]
- Immutable होता है → elements change नहीं कर सकते।
- Memory efficient होता है, क्योंकि lazy evaluation use करता है।

round() Function - round() function का use floating point number को nearest integer या specific decimal places तक round करने के लिए होता है।

Syntax:

round(number[, ndigits])

number → float या integer

ndigits → decimal places (optional, default = 0)

Examples:

1. Round to nearest integer

```
x = 4.7
print(round(x))
```

Output:

5

```
y = 4.3
print(round(y))
```

Output:

4

2. Round to specific decimal places

```
x = 3.14159
print(round(x, 2))
```

Output:

```
3.14
```

```
y = 2.71828
print(round(y, 3))
```

Output:

```
2.718
```

Unit-4
Function & File Handling

Module - Module Python का एक file होता है जिसमें functions, variables, classes और executable statements defined होते हैं।
Simple words में, module = reusable Python code।

- Module का extension हमेशा .py होता है
- Large program को small, manageable parts में divide करने के लिए modules use किए जाते हैं
- Code reuse, readability, और maintainability बढ़ती है

Example:

```
math.py   # this is a module
```

Why Modules are Used

- Code reuse possible होता है
- Program structured और clean रहता है
- Large projects को manage करना आसान
- Namespace conflicts से बचाव
- Debugging आसान होती है

Types of Modules in Python

1. Built-in Modules

Python के साथ पहले से available होते हैं
Examples:

- math
- sys
- time
- calendar
- random

```
import math

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

2. User-defined Modules

User खुद create करता है

Example:

my_module.py

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

Use in another file:

```
import my_module
print(my_module.add(10, 20))
```

How to access module

1. import - import keyword का use किसी complete module को current program में include करने के लिए किया जाता है।

Syntax:

```
import module_name
```

Working:

पूरा module memory में load होता है

Access करने के लिए dot operator (.) use करना पड़ता है

```
import math
print(math.pi)
print(math.factorial(5))
```

Multiple Modules Import:

```
import math, time, calendar
```

Import with Alias (as)

Long module names को short बनाने के लिए

```
import math as m

print(m.sqrt(25))
```

Advantages of import

- Namespace clear रहता है
- Name conflict से बचाव
- Readability better

2.from - from keyword का use module के specific members (function / variable / class) को directly import करने के लिए किया जाता है।

Syntax:

```
from module_name import member
```

Example:

```
from math import sqrt, pi

print(sqrt(36))

print(pi)
```

यहाँ math. लिखने की जरूरत नहीं

math - math module Python का built-in module है, जो mathematical operations और calculations करने के लिए functions और constants provide करता है।
इस module का उपयोग scientific calculations**,** engineering**,** statistics**,** और data science में extensively होता है।

Use करने से पहले import करना जरूरी है:

```
import math
```

Why math module is used

- Accurate mathematical calculations
- Advanced mathematical functions (trigonometry, logarithm, power)
- Ready-made constants (π, e)
- Faster and optimized calculations

math.pi

π (pi) का value देता है

```
import math
print(math.pi)
```

math.sqrt(x)

x का square root देता है

```
print(math.sqrt(25))   # 5.0
```

math.pow(x, y)

x की power y देता है (float return करता है)

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

** operator से अलग, यह हमेशा float देता है

math.fabs(x)

x का absolute value (float) देता है

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

math.factorial(x)

x! (factorial) देता है

```
print(math.factorial(5))  # 120
```

Only non-negative integers allowed

math.floor(x)

x से छोटा या बराबर nearest integer

```
print(math.floor(4.9))  # 4
```

math.ceil(x)

x से बड़ा या बराबर nearest integer

```
print(math.ceil(4.1))   # 5
```

Trigonometric Functions (Angles in Radians)

All trigonometric functions radians में काम करती हैं।

- math.sin(x)
- math.cos(x)
- math.tan(x)

```
print(math.sin(math.pi/2))  # 1.0
```

math.gcd(a, b)

Greatest Common Divisor

```
print(math.gcd(12, 18))  # 6
```

math.lcm(a, b)

Least Common Multiple (Python 3.9+)

```
print(math.lcm(12, 18))  # 36
```

random - random module Python का built-in module है, जो random (यादृच्छिक) numbers और random selection generate करने के लिए use होता है।
यह module games**,** simulations**,** lottery systems**,** sampling**,** और testing में widely use किया जाता है।

Use करने से पहले import करना जरूरी है:

```
import random
```

Why random module is used?

- Random numbers generate करने के लिए
- List / tuple / string से random element select करने के लिए
- Data shuffling के लिए
- Games और simulations बनाने के लिए

Seeding Function

```
random.seed(value)
```

Random numbers को reproducible बनाने के लिए seed set करता है।

```
random.seed(10)

print(random.random())
```

Same seed → same output sequence

Random Number Generation Functions

random.random()

0.0 से 1.0 के बीच (1.0 exclude) random float value return करता है।

```
import random

print(random.random())
```

Output:

$0.0 \leq x < 1.0$

random.randint(a, b)

a और b के बीच random integer देता है (a और b दोनों include)।

```
print(random.randint(1, 6))
```

Dice roll simulation के लिए useful

random.randrange(start, stop, step)

range() की तरह random number generate करता है।

```
print(random.randrange(0, 10, 2))
```

Even numbers में से random selection

Random Selection Functions

random.choice(sequence)

Given sequence (list, tuple, string) में से single random element select करता है।

```
colors = ["red", "green", "blue"]
print(random.choice(colors))
```

Set directly allowed नहीं (convert to list first)

Shuffling Function

random.shuffle(list)

List के elements को random order में shuffle करता है।

```
nums = [1, 2, 3, 4, 5]
random.shuffle(nums)
print(nums)
```

Original list change होती है (in-place)

Function - Function Python का एक block of code होता है, जिसे एक specific task perform करने के लिए लिखा जाता है और जरूरत पड़ने पर बार-बार call किया जा सकता है। Function program को modular**,** reusable और easy to manage बनाता है।

Example:

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

Why Functions are Used?

- Code reusability बढ़ती है
- Program readable और structured होता है
- Large program को छोटे parts में divide किया जा सकता है
- Debugging और maintenance आसान
- Repetition of code avoid होती है

Function के Parts (Structure of Function)

```
def function_name(parameters):
    statements
    return value
```

Part	Description
def	Function define करने का keyword
function_name	Function का नाम
parameters	Input values
statements	Function body
return	Output value (optional)

Types of Functions

1. Built-in Functions

Python में पहले से defined होते हैं
Examples: print(), len(), type(), range()

2. User-defined Functions

Programmer द्वारा बनाए जाते हैं

```
def square(x):
    return x * x
```

User Defined Function - User Defined Function वह function होता है जिसे programmer खुद अपनी आवश्यकता के अनुसार define करता है, ताकि किसी specific task को perform किया जा सके।
ऐसे functions program को reusable, modular, और easy to understand बनाते हैं।

Built-in functions (जैसे print(), len()) Python में पहले से मौजूद होते हैं, जबकि user defined functions programmer द्वारा बनाए जाते हैं।

Function Definition - Function definition का मतलब है function का structure और behavior तय करना।
Python में function define करने के लिए def keyword का उपयोग किया जाता है।

Syntax (Function Definition):

```
def function_name(parameters):
    statements
    return value
```

Example:

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

यहाँ:

def → function define करने का keyword

add → function name

a, b → parameters

return → result वापस करता है

Function calling - जब तक function call नहीं होता, function execute नहीं होता।

```
result = add(10, 20)

print(result)
```

Execution steps:

- Function call होता है
- Arguments (10, 20) parameters (a, b) में जाते हैं
- Function body execute होती है
- return value वापस करता है

User Defined Function की आवश्यकता (Need of User Defined Function)

1.Code Reusability (कोड का पुनः उपयोग)

User defined function बनाने का सबसे बड़ा reason है code reuse।
अगर कोई task program में बार-बार करना है, तो हर बार same code लिखने के बजाय उसे function में लिखकर multiple times call किया जा सकता है।

Example:

```
def add(a, b):

   return a + b
```

अब add() function को program में कहीं भी reuse किया जा सकता है।

2.Modularity (प्रोग्राम को छोटे भागों में बाँटना)

Large program को समझना और manage करना मुश्किल होता है।
User defined functions program को small, logical modules में divide कर देते हैं।

Benefit:

- Program structured बनता है
- Logic clearly divided रहता है

3.Readability & Clarity (कोड की समझ बढ़ाना)

Functions meaningful names के साथ code को self-explanatory बना देते हैं।
calculate_salary() पढ़कर ही समझ आ जाता है कि function क्या करता है।

4.Easy Debugging (डिबगिंग आसान होती है)

Function में error होने पर, पूरे program को check करने की जरूरत नहीं केवल उसी function को debug करना होता है

5.Easy Maintenance & Update (आसान बदलाव)

अगर logic change करना हो, Function में एक बार change करो ,Change automatically हर जगह reflect होगा

Example:
Tax calculation logic change हुआ → सिर्फ calculate_tax() function update करना होगा।

6.Parameter Passing Facility

User defined functions में, Different values pass करके Same logic को different situations में use कर सकते हैं

Example:

```
def area(r):
    return 3.14 * r * r
```

Function के चार प्रकार (Based on Arguments & Return Type)

Python में functions को arguments (parameters) और return value के आधार पर चार प्रकार में classify किया जाता है।

1.Function with Arguments and with Return Value

Definition: ऐसे functions जो arguments accept करते हैं और result return भी करते हैं।

Syntax:

```
def function_name(arguments):
    statements
    return value
```

Example:

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

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

Explanation:

- a, b → arguments
- return → calculated result वापस करता है
- Most commonly used type

2.Function with Arguments and without Return Value

Definition: ऐसे functions जो arguments accept करते हैं, लेकिन कोई value return नहीं करते। ये functions output directly print करते हैं।

Syntax:

```
def function_name(arguments):
    statements
```

Example:

```
def greet(name):
    print("Welcome", name)

greet("Omkar")
```

Explanation:

- Input लिया जाता है
- Result print होता है
- return keyword नहीं होता

3.Function without Arguments and with Return Value

Definition: ऐसे functions जो कोई argument accept नहीं करते, लेकिन value return करते हैं।

Syntax:

```
def function_name():
    statements
    return value
```

Example:

```
def get_pi():
    return 3.14

x = get_pi()
print(x)
```

Explanation:

- No input required
- Fixed value या internally calculated value return करता है

4.Function without Arguments and without Return Value

Definition: ऐसे functions जो ना arguments लेते हैं और ना ही return value देते हैं**।**
ये functions केवल task perform करते हैं।

Syntax:

```
def function_name():
    statements
```

Example:

```
def show_msg():
    print("Hello Python")

show_msg()
```

Explanation:

- No input
- No output
- Only display या process

Exception - Exception वह runtime error होती है जो program के execution के दौरान आती है और normal flow को disturb कर देती है। Python में जब कोई error आती है और उसे handle नहीं किया जाता, तो program terminate (crash) हो जाता है। Exception एक ऐसी runtime error है जो program के execution के समय उत्पन्न होती है और program के normal flow को बाधित करती है।

Error vs Exception

Error	Exception
Compile time या logical mistake	Runtime problem
Programmer की गलती	User input / environment की वजह से
Handle नहीं हो सकती	Handle की जा सकती है
SyntaxError	ZeroDivisionError

Exception कब आती है? (When Exceptions Occur)

Exception निम्न परिस्थितियों में आती है:

- गलत user input
- Invalid calculation
- File / database access problem
- Network / hardware failure

Examples:

```
print(10 / 0)      # ZeroDivisionError

print(int("abc")) # ValueError
```

Exception Handling - Exception handling वह mechanism है, जिससे program runtime errors को handle करके crash होने से बचता है और normal execution continue रखता है।

Python में exception handling के लिए used keyword:

1. try
2. except
3. else
4. finally

Exception Handling की आवश्यकता

Program Crash होने से बचाने के लिए

जब program के execution के दौरान runtime error (exception) आती है, तो exception handling के बिना program तुरंत terminate हो जाता है।
Exception handling का use करने से program crash होने के बजाय safely handle हो जाता है।

Example:

Divide by zero → program crash
Exception handling → proper message + program continue

Normal Flow of Program बनाए रखने के लिए

Exception आने पर program का normal flow disturb हो जाता है।
Exception handling से, Error handle हो जाती है, Program आगे execute होता रहता है

Result:

Controlled execution
Unexpected termination नहीं होती

User-Friendly Error Messages देने के लिए

Default error messages user के लिए confusing और technical होते हैं।
Exception handling से meaningful और simple messages दिखा सकते हैं।

Example:

```
print("Please enter a valid number")
```

instead of

```
ValueError: invalid literal for int()
```

Robust और Reliable Program बनाने के लिए

Robust program वह होता है जो, Invalid input, unexpected situations, Runtime errors को safely handle कर सके।

Exception handling से program more reliable और professional बनता है।

Debugging और Error Identification आसान बनाने के लिए

Exception handling से, Error का exact reason पता चलता है Which part failed यह समझ आता है | Exception as e से error message store कर सकते हैं।

Resource Management के लिए (File, Database, Network)

Files, database connections, network resources को, Properly close करना जरूरी होता है |

Exception handling के साथ finally block ensure करता है कि, Error आए या न आए, Resource हमेशा release हो

Example:

```
finally:
    file.close()
```

Large Applications & Real-Time Systems में जरूरी

Banking, hospital, railway, airline systems जैसे applications में, Program crash acceptable नहीं होता | Exception handling इन systems को fault tolerant बनाता है।

Exception Handling Keywords in Python

try - try block में वह code लिखा जाता है जहाँ exception आने की possibility होती है। Python पहले try block को execute करता है।

Example:

```
try:
    x = int(input("Enter a number: "))
    y = int(input("Enter another number: "))
```

```
    print(x / y)
```

Explanation:

- User input ले रहा है
- Division हो रहा है
- अगर error आएगा, तो control except block में चला जाएगा

except - except block में वह code लिखा जाता है जो exception आने पर execute होता है। यह program को crash होने से बचाता है।

Example:

```
except ZeroDivisionError:
    print("Zero se divide nahi kar sakte")
```

Complete Example:

```
try:
    a = int(input())
    b = int(input())
    print(a / b)
except ZeroDivisionError:
    print("Division by zero not allowed")
```

Explanation:

- अगर b = 0 होगा
- ZeroDivisionError आएगा
- Message print होगा, program crash नहीं होगा

else - else block तभी execute होता है जब try block में कोई exception नहीं आती।

Example:

```
try:
    x = int(input("Enter number: "))
    print(10 / x)
except ZeroDivisionError:
```

```
    print("Zero not allowed")
else:
    print("Division successful")
```

Explanation:

- Error नहीं आया → else execute
- Error आया → else skip
- Use case: जब success logic अलग से लिखना हो

finally - finally block हमेशा execute होता है, चाहे exception आए या न आए।

Example:

```
try:
    f = open("data.txt", "r")
    print(f.read())
except FileNotFoundError:
    print("File not found")
finally:
    print("File operation completed")
```

Explanation:

- File मिले या न मिले
- finally block execute होगा
- Mostly resource cleanup के लिए use

Example:

```
finally:
    f.close()
```

raise - raise keyword का use करके programmer खुद exception generate करता है।

Example:

```
age = int(input("Enter age: "))
if age < 18:
```

```
    raise ValueError("Age must be 18 or above")
```

Explanation:

- Condition false हुई
- Programmer ने manually error raise किया
- Useful for business rules

All Keywords Together (Combined Example)

```
try:
    age = int(input("Enter age: "))
    if age < 18:
        raise ValueError("Underage not allowed")
    print("Access granted")
except ValueError as e:
    print("Error:", e)
else:
    print("No exception occurred")
finally:
    print("Program execution completed")
```

Flow Explanation:

try → risky code

raise → condition fail → exception

except → error handled

else → execute only if no error

finally → always execute

Unit-5
GUI Programming

GUI (Graphics User Interface)

1. GUI का full form Graphical User Interface होता है।
2. यह एक ऐसा interface है जो user को electronic devices जैसे computer और tablet के साथ graphic elements के माध्यम से interact करने की सुविधा देता है।
3. इसमें information को display करने के लिए icons, menus और other graphical representations का उपयोग किया जाता है, जबकि text-based commands का उपयोग नहीं किया जाता।
4. ये graphical elements user को mouse या अन्य input devices का उपयोग करके computer को commands देने और functions select करने में सक्षम बनाते हैं।

Need of GUI (Graphical User Interface)

GUI (Graphical User Interface) की आवश्यकता आधुनिक कंप्यूटर सिस्टम और applications में इसलिए होती है क्योंकि यह user और computer के बीच interaction को सरल, प्रभावी और user-friendly बनाता है। नीचे GUI की आवश्यकता को विस्तार से समझाया गया है—

1. User Friendly Interface

GUI में commands टाइप करने की बजाय icons, buttons, menus, windows का उपयोग किया जाता है। इससे non-technical users भी आसानी से computer और software का उपयोग कर सकते हैं। CLI (Command Line Interface) की तुलना में GUI समझना बहुत आसान होता है।

2. Easy Learning and Usage

GUI applications को सीखने में कम समय लगता है क्योंकि user को complex commands याद नहीं करनी पड़तीं। Visual representation के कारण user जल्दी समझ जाता है कि कौन-सा option किस काम के लिए है।

3. Reduces Human Errors

Command line में गलत command टाइप होने से errors हो सकते हैं, जबकि GUI में pre-defined options होते हैं। इससे गलत input की संभावना कम हो जाती है और system ज्यादा reliable बनता है।

4. Faster Interaction

Mouse, touch या keyboard shortcuts के माध्यम से user बहुत जल्दी actions perform कर सकता है। Drag and drop, click operations जैसी सुविधाएँ काम की speed बढ़ाती हैं।

5. Better Visual Representation

GUI data को charts, graphs, images, icons और animations के रूप में दिखा सकता है। इससे information ज्यादा clear और attractive बनती है और decision making आसान होती है।

6. Multi-Tasking Support

GUI multiple windows और applications को एक साथ manage करने की सुविधा देता है। User multiple tasks को parallel तरीके से perform कर सकता है, जैसे एक साथ document edit करना और browser चलाना।

7. Standardization and Consistency

GUI applications में common standards follow किए जाते हैं, जैसे menu structure, icons और dialog boxes। इससे एक software सीखने के बाद दूसरे software को use करना आसान हो जाता है।

8. Suitable for Modern Applications

आज के modern applications जैसे banking software, educational apps, mobile apps और web applications GUI पर ही आधारित हैं। Touch-based devices (mobile, tablets) के लिए GUI अत्यंत आवश्यक है।

Advantages of GUI (Graphical User Interface)

GUI (Graphical User Interface) कंप्यूटर और user के बीच interaction को आसान, तेज और प्रभावी बनाता है। इसके अनेक advantages हैं, जो आधुनिक computer systems और applications में इसे अत्यंत उपयोगी बनाते हैं।

1. User Friendly Interface

GUI में icons, menus, buttons और windows का उपयोग किया जाता है, जिससे user को commands याद रखने की आवश्यकता नहीं होती। यह beginners और non-technical users के लिए बहुत आसान होता है।

2. Easy to Learn and Use

GUI applications को सीखना बहुत सरल होता है क्योंकि सभी options visually दिखाई देते हैं। Point-and-click method के कारण user कम training में ही software को efficiently use कर सकता है।

3. Reduces Errors

GUI में user को limited और predefined options दिए जाते हैं, जिससे गलत input या syntax errors की संभावना कम हो जाती है। इससे system ज्यादा reliable और safe बनता है।

4. Faster Work and Productivity

Mouse clicks, shortcuts और drag-and-drop जैसी सुविधाओं के कारण काम जल्दी होता है। GUI user की productivity बढ़ाता है और time की saving करता है।

5. Better Visual Presentation

GUI information को graphics, images, charts, graphs और animations के माध्यम से प्रस्तुत करता है। इससे data ज्यादा clear, attractive और understandable बनता है।

6. Multi-Tasking Support

GUI एक ही समय में multiple windows और applications को run करने की सुविधा देता है। User आसानी से different tasks के बीच switch कर सकता है।

7. Consistency and Standardization

GUI में common standards follow किए जाते हैं, जैसे menu structure, icons और dialog boxes। इससे एक application सीखने के बाद दूसरी GUI-based applications को use करना आसान हो जाता है।

8. Suitable for Modern and Touch-Based Devices

GUI mobile phones, tablets और touch screen devices के लिए अत्यंत उपयुक्त है। Modern applications जैसे online banking, e-learning, e-governance और mobile apps GUI पर ही आधारित होते हैं।

Python GUI Development Options

Python GUI (Graphical User Interface) develop करने के लिए कई options provide करता है। इनमें से सबसे important नीचे दिए गए हैं:

1. PyQt5

- PyQt5 Python के लिए एक graphical user interface (GUI) framework है।
- यह developers के बीच काफी popular है और GUI को coding या QT designer की सहायता से create किया जा सकता है।
- QT Development framework एक visual framework है जो widgets को drag and drop करके user interface build करने की सुविधा देता है।
- यह एक free और open source binding software है और cross-platform application development framework के रूप में implement किया गया है।
- इसका उपयोग Windows, Mac, Android, Linux और Raspberry Pi पर किया जाता है।

2. Python Tkinter

- Tkinter desktop applications develop करने के लिए Python की सबसे popular GUI libraries में से एक है।
- यह TK और Python standard GUI framework का combination है।
- Tkinter विभिन्न प्रकार के widgets provide करता है जैसे: labels, buttons, text boxes, checkboxes जिनका उपयोग GUI application में किया जाता है।
- Button control widgets का उपयोग applications display और develop करने के लिए किया जाता है, जबकि Canvas widget का उपयोग application में shapes जैसे: lines, polygons, rectangles, draw करने के लिए किया जाता है।
- इसके अतिरिक्त, Tkinter Python की built-in library है, इसलिए इसे अन्य GUI frameworks की तरह install करने की आवश्यकता नहीं होती।

3. PySide2

- तीसरी Python GUI library जिसके बारे में हम चर्चा कर रहे हैं वह PySide2 है, जिसे QT for Python भी कहा जाता है।
- Qt for Python, Qt के official Python bindings (PySide2) provide करता है, जिससे Python applications में Qt APIs का उपयोग संभव हो पाता है। इसमें एक binding generator tool (Shiboken2) भी होता है, जिसका उपयोग C++ projects को Python में expose करने के लिए किया जाता है।
- Qt for Python LGPLv3/GPLv3 और Qt commercial license के अंतर्गत उपलब्ध है।

4. Kivy

- Kivy एक open source Python library है, जिसका उपयोग innovative user interfaces वाले applications जैसे multi-touch apps के rapid development के लिए किया जाता है।
- Kivy निम्न platforms पर run करता है: Linux, Windows, OS X, Android, iOS, Raspberry Pi, आप same code को सभी supported platforms पर run कर सकते हैं। यह natively कई inputs, protocols और devices को support करता है जैसे:WM_Touch, WM_Pen, Mac OS X Trackpad, Magic Mouse, Linux Kernel HID
- Kivy MIT license के अंतर्गत 100% free है।
- Toolkit में 20 से अधिक widgets उपलब्ध हैं और इसके कई parts C language में लिखे गए हैं।

5. wxPython

- wxPython Python programming language के लिए एक cross-platform GUI toolkit है।
- यह Python programmers को robust और highly functional GUI वाले programs easily create करने की सुविधा देता है।
- इसे Python extension modules के रूप में implement किया गया है जो popular wxWidgets cross-platform library (C++) के GUI components को wrap करता है।
- Python और wxWidgets की तरह wxPython भी Open Source है।
- wxPython एक cross-platform toolkit है, जिसका अर्थ है कि same program बिना modification के multiple platforms पर run करता है। Currently supported platforms हैं: Microsoft Windows, Mac OS X, macOS, Linux

Tkinter

- Tkinter Python की standard GUI library है।
- Python जब Tkinter के साथ combine किया जाता है तो GUI applications create करने का fast और easy तरीका प्रदान करता है।
- Tkinter, Tk GUI toolkit के लिए एक powerful object-oriented interface provide करता है।

Creating GUI Application using Tkinter

Tkinter का उपयोग करके GUI application create करना एक आसान task है। इसके लिए नीचे दिए गए steps perform करने होते हैं:

- Tkinter module import करें।
- GUI application का main window create करें।
- GUI application में एक या अधिक widgets add करें।
- User द्वारा trigger किए गए events को handle करने के लिए main event loop enter करें।

Example –

```
import tkinter

top = tkinter.Tk()

# Code to add widgets will go here...

top.mainloop()
```

Output -

Tkinter Widgets

Tkinter various controls provide करता है, जैसे buttons, labels और text boxes जो GUI application में used होते हैं। इन controls को commonly widgets कहा जाता है। यहां ये widgets हैं साथ ही निम्न तालिका में brief description है -

Tkinter Widgets Table:

Sr.No.	Widget	Description
1	Button	Button widget आपके application में buttons display करने के लिए used होता है।
2	Canvas	Canvas widget आपके application में shapes draw करने के लिए used होता है, जैसे lines, ovals, polygons और rectangles।
3	Checkbutton	Checkbutton widget options की संख्या को checkboxes के रूप में display करने के लिए used होता है। User एक समय में multiple options select कर सकता है।
4	Entry	Entry widget user से values accept करने के लिए single-line text field display करने के लिए used होता है।
5	Frame	Frame widget अन्य widgets organize करने के लिए container widget के रूप में used होता है।
6	Label	Label widget अन्य widgets के लिए single-line caption provide करने के लिए used होता है। यह images भी contain कर सकता है।
7	Listbox	Listbox widget user को options की list provide करने के लिए used होता है।
8	Menubutton	Menubutton widget आपके application में menus display करने के लिए used होता है।

9	Menu	Menu widget user को various commands provide करने के लिए used होता है। ये commands Menubutton के inside contained होते हैं।
10	Message	Message widget user से values accept करने के लिए multiline text fields display करने के लिए used होता है।
11	Radiobutton	Radiobutton widget options की संख्या को radio buttons के रूप में display करने के लिए used होता है। User एक समय में केवल एक option select कर सकता है।
12	Scale	Scale widget slider widget provide करने के लिए used होता है।
13	Scrollbar	Scrollbar widget various widgets को scrolling capability add करने के लिए used होता है, जैसे list boxes।
14	Text	Text widget multiple lines में text display करने के लिए used होता है।
15	Toplevel	Toplevel widget separate window container provide करने के लिए used होता है।
16	Spinbox	Spinbox widget standard Tkinter Entry widget का variant है, जिसका use fixed number of values से select करने के लिए किया जा सकता है।
17	PanedWindow	PanedWindow एक container widget है जिसमें किसी भी number के panes हो सकते हैं, horizontally या vertically arranged।
18	LabelFrame	labelframe एक simple container widget है। इसका primary purpose complex window layouts के लिए spacer या container के रूप में act करना है।
19	tkMessageBox	यह module आपके applications में message boxes display करने के लिए used होता है।

Python Tkinter Geometry

Tkinter geometry वह method specify करती है जिसका use करके widgets display पर represented होते हैं। Python Tkinter निम्नलिखित geometry methods provide करता है।

- The pack() method
- The grid() method
- The place() method

Python Tkinter pack() method

- pack() widget का use widget को block में organize करने के लिए होता है।
- Python application में pack() method का use करके added positions widgets को method call में specified various options का use करके controlled किया जा सकता है।
- However, controls कम होते हैं और widgets generally less organized manner में added होते हैं।

Syntax -

```
widget.pack(options)
```

Example -

```
from tkinter import *

parent = Tk()

redbutton = Button(parent, text="Red", fg="red")

redbutton.pack(side=LEFT)

greenbutton = Button(parent, text="Black", fg="black")

greenbutton.pack(side=RIGHT)

bluebutton = Button(parent, text="Blue", fg="blue")

bluebutton.pack(side=TOP)

blackbutton = Button(parent, text="Green", fg="red")

blackbutton.pack(side=BOTTOM)

parent.mainloop()
```

Output –

Python Tkinter grid() method

- grid() geometry manager widgets को tabular form में organizes करता है।
- हम method call में rows और columns को options के रूप में specify कर सकते हैं।
- हम किसी widget का column span (width) या rowspan(height) भी specify कर सकते हैं।
- यह widgets को python application में place करने का more organized way है।

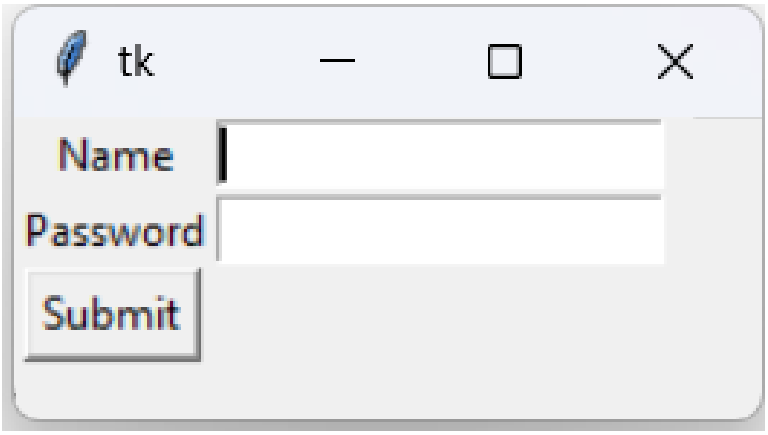
Syntax -

```
widget.grid(options)
```

Example -

```
from tkinter import *

parent = Tk()

name = Label(parent, text="Name").grid(row=0, column=0)

e1 = Entry(parent).grid(row=0, column=1)

password = Label(parent, text="Password").grid(row=1, column=0)

e2 = Entry(parent).grid(row=1, column=1)

submit = Button(parent, text="Submit").grid(row=3, column=0)

parent.mainloop()
```

Output –



Python Tkinter place() method

place() geometry manager widgets को specific x और y coordinates पर organizes करता है।

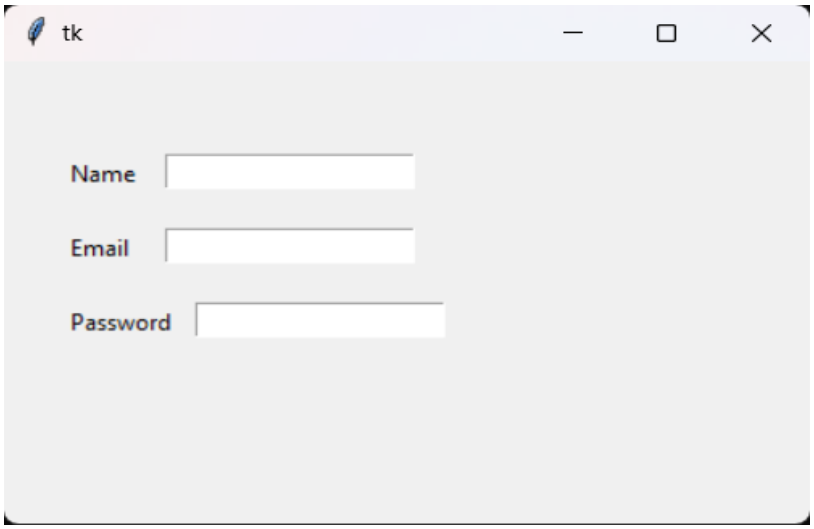
Syntax -

```
widget.place(options)
```

Example -

```
from tkinter import *
top = Tk()
top.geometry("400x250")
name = Label(top, text="Name").place(x=30, y=50)
email = Label(top, text="Email").place(x=30, y=90)
password = Label(top, text="Password").place(x=30, y=130)
e1 = Entry(top).place(x=80, y=50)
e2 = Entry(top).place(x=80, y=90)
e3 = Entry(top).place(x=95, y=130)
top.mainloop()
```

Output –



Python tkinter Button

- Button widget का use python application में various types के buttons add करने के लिए होता है।
- Python हमें हमारी requirements according button के look को configure करने की allow करता है।
- Various options requirements के depending upon set या reset किए जा सकते हैं।
- हम किसी button के साथ एक method या function भी associate कर सकते हैं जो button press होने पर called होती है।

Syntax -

W = Button(parent, options)

Possible options की एक list नीचे दी गई है।

Button Widget Options:

SN	Option	Description
1	activebackground	यह button का background represent करता है जब mouse button पर hover करता है।
2	activeforeground	यह button का font color represent करता है जब mouse button पर hover करता है।
3	bd	यह pixels में border width represent करता है।
4	bg	यह button का background color represent करता है।
5	command	यह function call पर set होता है जो function call होने पर scheduled होती है।
6	fg	Button का foreground color।
7	font	Button text का font।
8	height	Button की height। Height textual lines के लिए text lines की संख्या या images के लिए pixels की संख्या में represented होती है।
9	highlightcolor	Highlight का color जब button के पास focus होता है।
10	image	यह button पर displayed image पर set होता है।
11	justify	यह illustrate करता है कि multiple text lines किस way से represented होती हैं। यह left justification के लिए LEFT, right

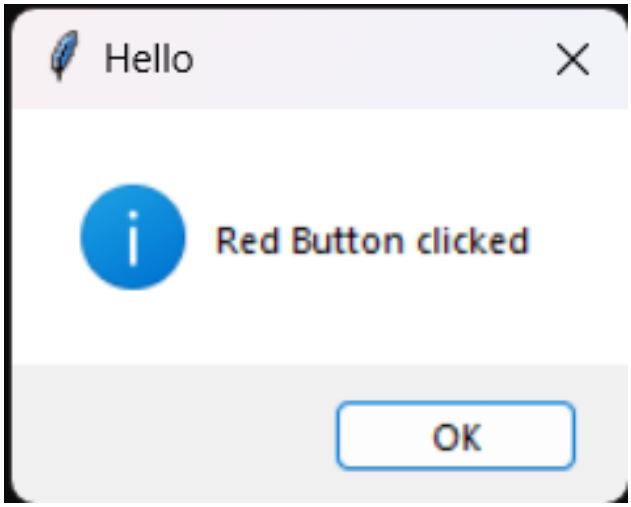
		justification के लिए RIGHT, और center justification के लिए CENTER पर set होता है।
12	padx	Horizontal direction में button का additional padding।
13	pady	Vertical direction में button का additional padding।
14	relief	यह border का type represent करता है। यह SUNKEN, RAISED, GROOVE, और RIDGE हो सकता है।
15	state	यह option button को unresponsive बनाने के लिए DISABLED पर set होता है। ACTIVE button की active state represent करता है।
16	underline	Button text को underlined बनाने के लिए इस option को set करें।
17	width	Button की width। यह textual buttons के लिए letters की संख्या या image buttons के लिए pixels के रूप में exist करती है।
18	wraplength	यदि value positive number पर set की जाती है, तो text lines इस length के within fit होने के लिए wrapped होंगी।

Example -

```
from tkinter import *

from tkinter import messagebox

top = Tk()

top.geometry("200x100")


def fun():

   messagebox.showinfo("Hello", "Red Button clicked")


b1 = Button(top, text="Red", command=fun, activeforeground="red",
activebackground="pink", pady=10)

b2 = Button(top, text="Blue", activeforeground="blue",
activebackground="pink", pady=10)

b3 = Button(top, text="Green", activeforeground="green",
activebackground="pink", pady=10)

b4 = Button(top, text="Yellow", activeforeground="yellow",
activebackground="pink", pady=10)
```

```
b1.pack(side=LEFT)
b2.pack(side=RIGHT)
b3.pack(side=TOP)
b4.pack(side=BOTTOM)
top.mainloop()
```

Output –



Python tkinter Canvas

- Canvas widget structured graphics को python application में add करने के लिए used होता है।
- यह python application में graph और plots draw करने के लिए used होता है।

Syntax -

```
w = Canvas(parent, options)
```

Possible options की एक list नीचे दी गई है।

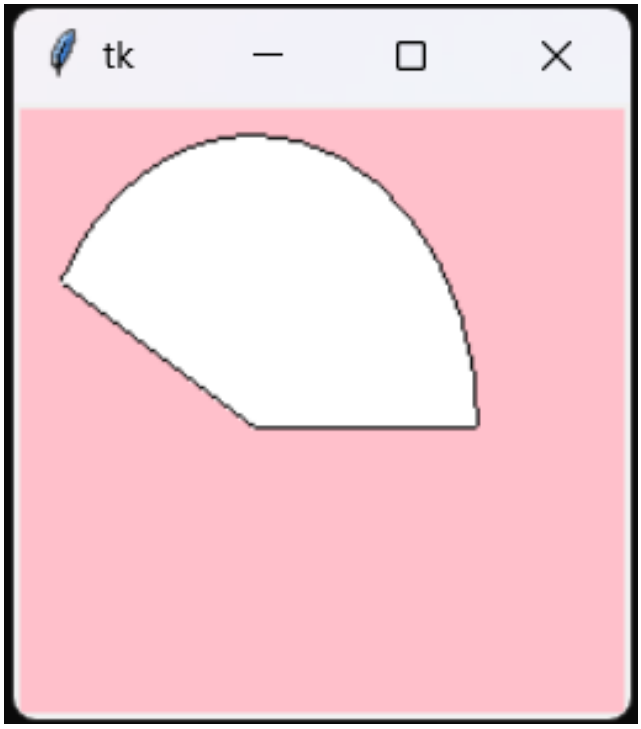
Canvas Widget Options:

SN	Option	Description
1	bd	यह border width represent करता है। Default width 2 है।
2	bg	यह canvas का background color represent करता है।
3	confine	Canvas को scroll region के outside unscrollable बनाने के लिए set किया जाता है।
4	cursor	Cursor को arrow, circle, dot, etc. के रूप में canvas पर used किया जाता है।
5	height	यह vertical direction में canvas का size represent करता है।
6	highlightcolor	यह highlight color represent करता है जब widget focused होता है।
7	relief	यह border का type represent करता है। Possible values हैं SUNKEN, RAISED, GROOVE, और RIDGE।
8	scrollregion	यह coordinates represent करता है जो tuple के रूप में specified होते हैं जिसमें canvas का area होता है।
9	width	यह canvas की width represent करता है।
10	xscrollincrement	यदि यह positive value पर set होता है। Canvas केवल इस value के multiple पर placed होता है।
11	xscrollcommand	यदि canvas scrollable है, तो यह attribute horizontal scrollbar की .set() method होनी चाहिए।
12	yscrollincrement	xscrollincrement की तरह works करता है, लेकिन vertical movement govern करता है।
13	yscrollcommand	यदि canvas scrollable है, तो यह attribute vertical scrollbar की .set() method होनी चाहिए।

Example -

```
from tkinter import *

top = Tk()

top.geometry("200x200")

c = Canvas(top, bg="pink", height=200, width=200)

arc = c.create_arc((5, 10, 150, 200), start=0, extent=150, fill="white")

c.pack()

top.mainloop()
```

Output –



Some common drawing methods:

Creating an Oval:

```
oval = C.create_oval(x0, y0, x1, y1, options)
```

Creating an arc:

```
arc = C.create_arc(x0, y0, x1, y1, start=0, extent=110, options)
```

Creating a Line:

```
line = C.create_line(x0, y0, x1, y1, ..., xn, yn, options)
```

Creating a polygon:

```
polygon = C.create_polygon(x0, y0, x1, y1, ..., xn, yn, options)
```

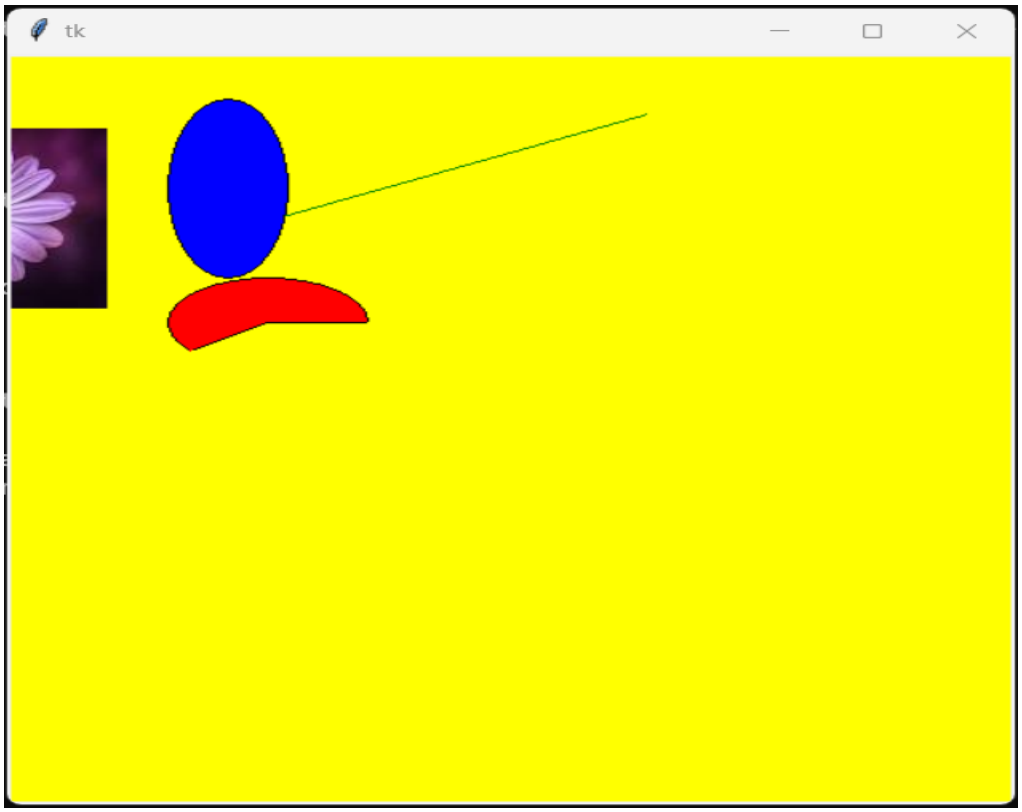
Creating an image:

```
filename = ImageTk.PhotoImage(Image.open("download.jpeg"))
image = C.create_image(50, 50, anchor=NE, image=filename)
```

Example -

```
from tkinter import *
from PIL import Image, ImageTk
root = Tk()
C = Canvas(root, bg="yellow", height=500, width=500)
line = C.create_line(108, 120, 320, 40, fill="green")
arc = C.create_arc(180, 150, 80, 210, start=0, extent=220, fill="red")
oval = C.create_oval(80, 30, 140, 150, fill="blue")
filename = ImageTk.PhotoImage(Image.open("download.jpeg"))
image = C.create_image(50, 50, anchor=NE, image=filename)
C.pack()
mainloop()
```

Output –



Python Tkinter Checkbutton

- Checkbutton user की choices को track करने के लिए used होता है जो application को provided की जाती हैं।
- अन्य words में, हम कह सकते हैं कि Checkbutton on/off selections implement करने के लिए used होता है।
- Checkbutton text या images contain कर सकता है।
- Checkbutton mostly user को many choices provide करने के लिए used होता है जिनमें से user को एक या अधिक choose करने की need होती है।
- यह generally many of many selections implement करता है।

Syntax -

```
w = Checkbutton(master, options)
```

Possible options की एक list नीचे दी गई है।

Checkbutton Widget Options:

SN	Option	Description
1	activebackground	यह background color represent करता है जब checkbutton cursor के under होता है।
2	activeforeground	यह checkbutton का foreground color represent करता है जब checkbutton cursor के under होता है।
3	bg	Button का background color।
4	bitmap	यह button पर एक image (monochrome) display करता है।
5	bd	Corner के around border का size।
6	command	यह एक function से associated होता है जो checkbutton की state change होने पर called की जाती है।
7	cursor	Mouse pointer cursor name में change हो जाएगा जब यह checkbutton पर over होता है।
8	disableforeground	यह color है जो disabled checkbutton के text को represent करने के लिए used होता है।
9	font	यह checkbutton का font represent करता है।
10	fg	Checkbutton का foreground color (text color)।
11	height	Checkbutton की height (lines की संख्या)। Default height 1 है।
12	highlightcolor	Focus highlight का color जब checkbutton focus के under होता है।
13	image	Checkbutton को represent करने के लिए used image।
14	justify	यह specify करता है कि यदि text multiple lines contain करता है तो text का justification क्या है।

15	offvalue	Associated control variable default रूप से 0 पर set होती है यदि button unchecked है। हम unchecked variable की state को किसी अन्य में change कर सकते हैं।
16	onvalue	Associated control variable को onvalue पर set किया जाता है यदि button checked है।
17	padx	Button के text का horizontal padding।
18	pady	Button के text का vertical padding।
19	relief	Border का type। Default FLAT है, अन्य options हैं SUNKEN, RAISED, GROOVE, RIDGE।
20	selectcolor	Color जो selected होने पर checkbutton के background के रूप में displayed होता है।
21	selectimage	Image जो selected होने पर checkbutton पर displayed होती है।
22	state	Button की current state। NORMAL, ACTIVE, या DISABLED हो सकती है।
23	underline	Text में underline करने के लिए character का index।
24	variable	Button की state को track करने वाली control variable।
25	width	Button की width characters में।
26	wraplength	Text wrap करने की maximum length।

Methods

जिन methods को Checkbuttons के साथ called किया जा सकता है, उनका वर्णन निम्न तालिका में किया गया है।

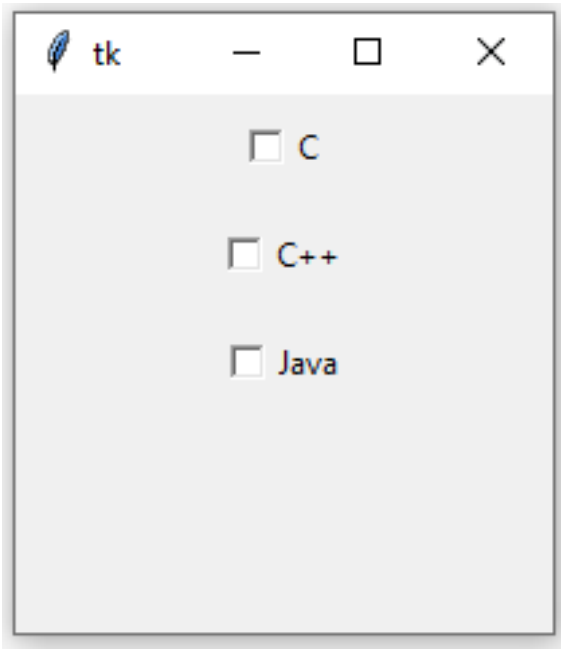
SN	Methods	Description
1	deselect()	Checkbutton को turn off करने के लिए called किया जाता है।
2	flash()	Checkbutton active और normal colors के between flashed होता है।
3	invoke()	यह checkbutton से associated method को invoke करेगा।
4	select()	Checkbutton को turn on करने के लिए called किया जाता है।
5	toggle()	विभिन्न Checkbuttons के between toggle करने के लिए used होता है।

Example -

```
from tkinter import *

top = Tk()

top.geometry("200x200")

checkvar1 = IntVar()
```

```
checkvar2 = IntVar()

checkvar3 = IntVar()


chkbtn1 = Checkbutton(top, text="C", variable=checkvar1, onvalue=1,
offvalue=0, height=2, width=10)

chkbtn2 = Checkbutton(top, text="C++", variable=checkvar2, onvalue=1,
offvalue=0, height=2, width=10)

chkbtn3 = Checkbutton(top, text="Java", variable=checkvar3, onvalue=1,
offvalue=0, height=2, width=10)


print(checkvar1)


chkbtn1.pack()

chkbtn2.pack()

chkbtn3.pack()

top.mainloop()
```

Output –



Python Tkinter Label

- Label का use container box specify करने के लिए होता है जहां हम text या images place कर सकते हैं।
- इस widget का use user को python application में used अन्य widgets के बारे में message provide करने के लिए होता है।
- Various options हैं जिन्हें Label में shown text या text के part को configure करने के लिए specified किया जा सकता है।

Syntax -

w = Label(master, options)

Possible options की एक list नीचे दी गई है।

Label Widget Options:

SN	Option	Description
1	Anchor	यह widget को provided size के within text की exact position specify करता है। Default value CENTER है, जो specified space के within text को center करने के लिए used होता है।
2	bg	Widget के behind displayed background color।
3	bitmap	यह bitmap को graphical object पर set करने के लिए used होता है जिसे specified किया गया है, ताकि label text के instead graphics represent कर सके।
4	bd	यह border की width represent करता है। Default 2 pixels है।
5	cursor	Mouse pointer specified cursor के type में change हो जाएगी, i.e., arrow, dot, etc.
6	font	Widget के inside written text का font type।
7	fg	Widget के inside written text का foreground color।
8	height	Widget की height।
9	image	Label के रूप में shown की जाने वाली image।
10	justify	यह text की orientation represent करने के लिए used होता है यदि text multiple lines contain करता है। यह left justification के लिए LEFT, right justification के लिए RIGHT, और center justification के लिए CENTER पर set हो सकता है।
11	padx	Text का horizontal padding। Default value 1 है।
12	pady	Text का vertical padding। Default value 1 है।
13	relief	Border का type। Default value FLAT है।
14	text	यह string variable पर set होता है जिसमें एक या अधिक lines का text हो सकता है।

15	textvariable	Widget के inside written text control variable StringVar पर set होता है ताकि इसे accessed और changed किया जा सके।
16	underline	हम text के specified letter के under एक line display कर सकते हैं। इस option को उस letter की number पर set करें जिसके under line displayed होगी।
17	width	Widget की width। इसे characters की संख्या के रूप में specified किया जाता है।
18	wraplength	Label text के रूप में केवल एक line होने के instead, हम इसे lines की संख्या में break कर सकते हैं जहां प्रत्येक line में इस option को specified characters की संख्या होती है।

Python Tkinter Entry

- Entry widget single line text-box provide करने के लिए used होता है user को user से एक value accept करने के लिए।
- हम user से text strings accept करने के लिए Entry widget का use कर सकते हैं।
- इसका use केवल user से text की एक line के लिए किया जा सकता है।
- Text की multiple lines के लिए, हमें text widget का use करना चाहिए।

Syntax -

```
w = Entry(parent, options)
```

Possible options की एक list नीचे दी गई है।

Entry Widget Options:

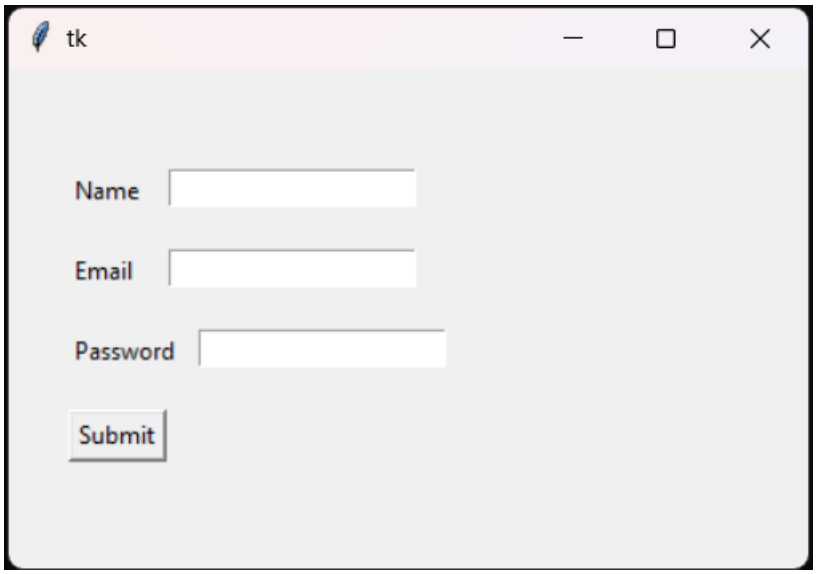
SN	Option	Description
1	bg	Widget का background color।
2	bd	Pixels में widget की border width।
3	cursor	Mouse pointer arrow, dot, etc. पर set cursor type में change हो जाएगी।
4	exportselection	Entry box के inside written text default रूप से clipboard में automatically copied हो जाएगी। हम इसे not copy करने के लिए exportselection को 0 पर set कर सकते हैं।
5	fg	यह text का color represent करता है।
6	font	यह text का font type represent करता है।
7	highlightbackground	यह traversal highlight region में display करने के लिए color represent करता है जब widget के पास input focus नहीं होता है।

8	highlightcolor	यह traversal highlight rectangle के लिए use किया जाने वाला color represent करता है जो widget के around drawn होता है जब इसके पास input focus होता है।
9	highlightthickness	यह एक non-negative value represent करता है जो widget के around draw की जाने वाली highlight rectangle की width indicate करती है जब इसके पास input focus होता है।
10	insertbackground	यह insertion cursor द्वारा covered area में background के रूप में use किया जाने वाला color represent करता है। यह color normally widget के लिए normal background को override करेगा।
11	insertborderwidth	यह insertion cursor के around draw की जाने वाली 3-D border की width indicate करने वाली non-negative value represent करता है। Value का Tk_GetPixels को acceptable forms में से कोई भी हो सकता है।
12	insertofftime	यह एक non-negative integer value represent करता है जो indicate करती है कि insertion cursor प्रत्येक blink cycle में "off" रहने के लिए milliseconds की संख्या। यदि यह option zero है, तो cursor नहीं blinks: यह हर समय on रहता है।
13	insertontime	एक non-negative integer value specify करता है जो indicate करती है कि insertion cursor प्रत्येक blink cycle में "on" रहने के लिए milliseconds की संख्या।
14	insertwidth	यह value indicate करती है जो insertion cursor की total width represent करती है। Value का Tk_GetPixels को acceptable forms में से कोई भी हो सकता है।
15	justify	यह specify करता है कि text कैसे organized है यदि text multiple lines contain करता है।
16	relief	यह border का type specify करता है। इसका default value FLAT है।
17	selectbackground	Selected text का background color।
18	selectborderwidth	Selected task के around display की जाने वाली border की width।
19	selectforeground	Selected task का font color।
20	show	Entry text को string के instead किसी अन्य type के show करने के लिए used होता है। उदाहरण के लिए, password stars (*) का use करके typed की जाती है।
21	textvariable	Entry से text retrieve करने के लिए StringVar के instance पर set होता है।
22	width	Displayed text या image की width।
23	xscrollcommand	Entry widget horizontal scrollbar से linked हो सकती है यदि हम चाहते हैं कि user actual width of widget से अधिक text enter करे।

Example -

```
from tkinter import *

top = Tk()

top.geometry("400x250")

name = Label(top, text="Name").place(x=30, y=50)

email = Label(top, text="Email").place(x=30, y=90)

password = Label(top, text="Password").place(x=30, y=130)


submitbtn = Button(top, text="Submit", activebackground="pink",
activeforeground="blue").place(x=30, y=170)


e1 = Entry(top).place(x=80, y=50)

e2 = Entry(top).place(x=80, y=90)

e3 = Entry(top).place(x=95, y=130)

top.mainloop()
```

Output –



Python Tkinter Frame

- Python Tkinter Frame widget widgets के group को organize करने के लिए used होता है।
- यह एक container की तरह act करता है जिसका use अन्य widgets को hold करने के लिए किया जा सकता है।
- Screen के rectangular areas python application में widgets organize करने के लिए used होते हैं।

Syntax -

```
w = Frame(parent, options)
```

Possible options की एक list नीचे दी गई है।

Frame Widget Options:

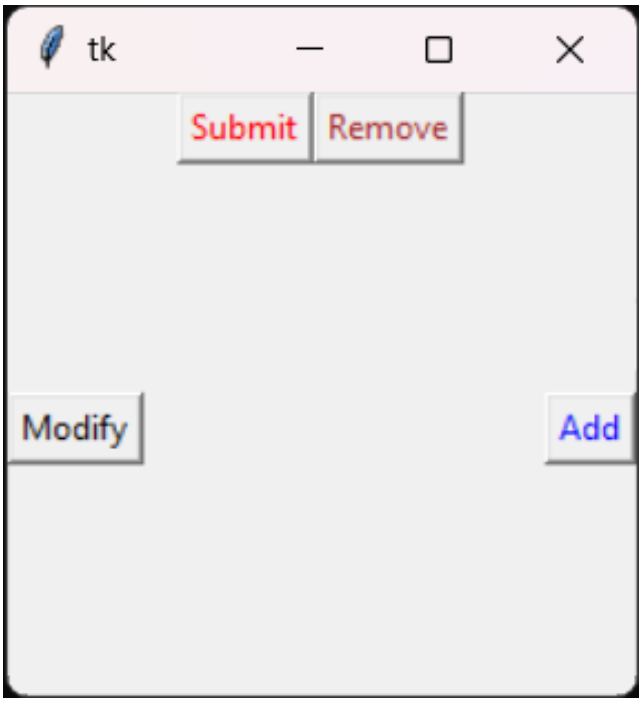
SN	Option	Description
1	bd	यह border width represent करता है।
2	bg	Widget का background color।
3	cursor	Mouse pointer arrow, dot, etc. जैसे different values पर set cursor type में change हो जाती है।
4	height	Frame की height।
5	highlightbackground	Background color का color जब यह focus के under होता है।
6	highlightcolor	Text color जब widget focus के under होता है।
7	highlightthickness	Border के around thickness specify करता है जब widget focus के under होता है।
8	relief	Border का type specify करता है। Default value FLAT है।
9	width	Widget की width represent करता है।

Example -

```
from tkinter import *

top = Tk()

top.geometry("140x100")

frame = Frame(top)

frame.pack()

leftframe = Frame(top)
```

```
leftframe.pack(side=LEFT)

rightframe = Frame(top)

rightframe.pack(side=RIGHT)

btn1 = Button(frame, text="Submit", fg="red", activebackground="red")

btn1.pack(side=LEFT)

btn2 = Button(frame, text="Remove", fg="brown", activebackground="brown")

btn2.pack(side=RIGHT)

btn3 = Button(rightframe, text="Add", fg="blue", activebackground="blue")

btn3.pack(side=LEFT)

btn4 = Button(leftframe, text="Modify", fg="black",
activebackground="white")

btn4.pack(side=RIGHT)

top.mainloop()
```

Output –



Python Tkinter Menu

- Menu widget python application में various types के menus (top level, pull down, और pop up) create करने के लिए used होता है।
- Top-level menus वे होते हैं जो parent window के title bar के just under displayed होते हैं।
- हमें Menu widget का new instance create करने और add() method का use करके इसमें various commands add करने की need होती है।

Syntax -

w = Menu(top, options)

Possible options की एक list नीचे दी गई है।

Menu Widget Options:

SN	Option	Description
1	activebackground	Widget का background color जब widget focus के under होता है।
2	activeborderwidth	Widget के border की width जब यह mouse के under होता है। Default 1 pixel है।
3	activeforeground	Widget का font color जब widget के पास focus होता है।
4	bg	Widget का background color।
5	bd	Widget की border width।
6	cursor	Mouse pointer arrow या dot जैसे cursor type में change हो जाती है जब यह widget पर hovers करता है।
7	disabledforeground	Widget का font color जब यह disabled होता है।
8	font	Widget के text का font type।
9	fg	Widget का foreground color।
10	postcommand	Postcommand किसी भी function पर set हो सकता है जो mouse menu पर hovers करने पर called होता है।
11	relief	Widget के border का type। Default type RAISED है।
12	image	Menu पर image display करने के लिए used होता है।
13	selectcolor	Color जो checkbutton या radiobutton display करने के लिए used होता है जब वे selected होते हैं।
14	tearoff	Default रूप से, menu में choices position 1 से place लेना start करती हैं। यदि हम tearoff = 1 set करते हैं, तो यह 0th position से place लेना start करेगी।
15	title	इस option को window के title पर set करें यदि आप window के title को change करना चाहते हैं।

Menu Widget Methods:

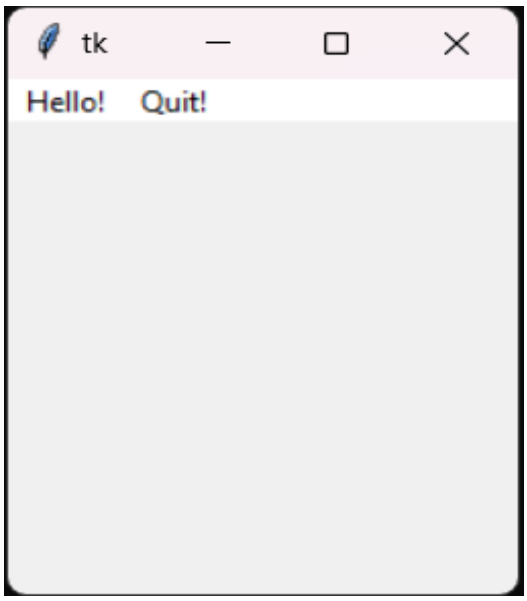
SN	Method	Description
1	add_command(options)	Menu items को menu में add करने के लिए used होता है।
2	add_radiobutton(options)	यह method radiobutton को menu में add करता है।
3	add_checkbutton(options)	यह method checkbuttons को menu में add करने के लिए used होता है।
4	add_cascade(options)	Parent menu को given menu associate करके hierarchical menu create करने के लिए used होता है।
5	add_separator()	Menu में separator line add करने के लिए used होता है।
6	add(type, options)	Specific menu item को menu में add करने के लिए used होता है।
7	delete(startindex, endindex)	Specified range में exist menu items delete करने के लिए used होता है।
8	entryconfig(index, options)	Given index द्वारा identified menu item को configure करने के लिए used होता है।
9	index(item)	Specified menu item का index get करने के लिए used होता है।
10	insert_separator(index)	Specified index पर separator insert करने के लिए used होता है।
11	invoke(index)	Specified index पर given choice से associated function invoke करने के लिए used होता है।
12	type(index)	Index द्वारा specified choice का type get करने के लिए used होता है।

Example 1 -

```
from tkinter import *

top = Tk()

def hello():

   print("hello!")

# create a toplevel menu

menubar = Menu(top)

menubar.add_command(label="Hello!", command=hello)

menubar.add_command(label="Quit!", command=top.quit)

# display the menu
```

```
top.config(menu=menubar)
top.mainloop()
```

Output –

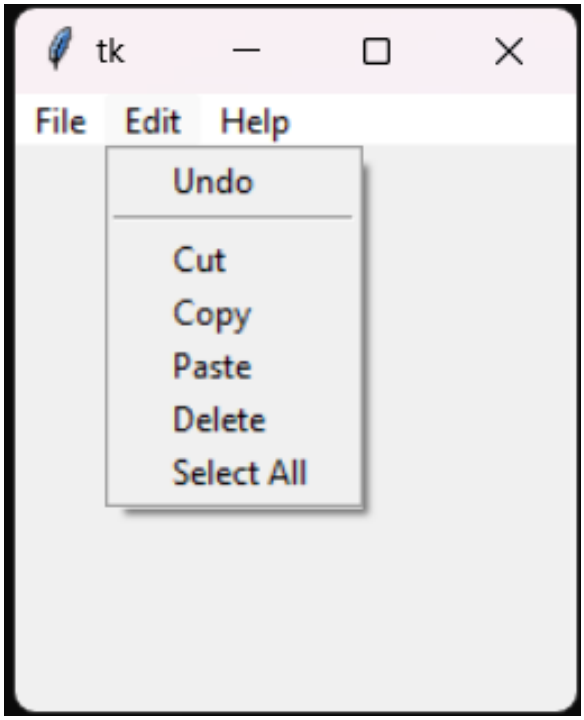


Example 2-

```
from tkinter import Toplevel, Button, Tk, Menu
top = Tk()
menubar = Menu(top)
file = Menu(menubar, tearoff=0)
file.add_command(label="New")
file.add_command(label="Open")
file.add_command(label="Save")
file.add_command(label="Save as...")
file.add_command(label="Close")
file.add_separator()
file.add_command(label="Exit", command=top.quit)
menubar.add_cascade(label="File", menu=file)
```

```
edit = Menu(menubar, tearoff=0)
edit.add_command(label="Undo")
edit.add_separator()
edit.add_command(label="Cut")
edit.add_command(label="Copy")
edit.add_command(label="Paste")
edit.add_command(label="Delete")
edit.add_command(label="Select All")
menubar.add_cascade(label="Edit", menu=edit)
help = Menu(menubar, tearoff=0)
help.add_command(label="About")
menubar.add_cascade(label="Help", menu=help)
top.config(menu=menubar)
top.mainloop()
```

Output –



Python Tkinter Text

- Text widget का use Python application पर text data show करने के लिए होता है। However, Tkinter हमें Entry widget provide करता है जो single line text box implement करने के लिए used होता है।
- Text widget का use multi line formatted text को various styles और attributes के साथ display करने के लिए होता है।
- Text widget mostly user को text editor provide करने के लिए used होता है।

Syntax

```
w = Text(top, options)
```

Options की एक list जो Text widget के साथ used की जा सकती है, नीचे दी गई है।

SN	Option	Description
1	bg	Widget का background color।
2	bd	Widget की border width represent करता है।
3	cursor	Mouse pointer specified cursor type में change हो जाती है, i.e. arrow, dot, etc.
4	exportselection	Selected text window manager में selection में exported होता है। हम इसे 0 पर set कर सकते हैं यदि हम नहीं चाहते कि text exported हो।
5	font	Text का font type।
6	fg	Widget का text color।
7	height	Lines में widget का vertical dimension।
8	highlightbackground	Highlightcolor जब widget के पास focus नहीं होता है।
9	highlightthickness	Focus highlight की thickness। Default value 1 है।
10	highlightcolor	Focus highlight का color जब widget के पास focus होता है।
11	insertbackground	Insertion cursor का color represent करता है।
12	insertborderwidth	Cursor के around border की width represent करता है। Default 0 है।
13	insertofftime	Milliseconds में time amount जिसके during insertion cursor blink cycle में off होता है।
14	insertontime	Milliseconds में time amount जिसके during insertion cursor blink cycle में on होता है।
15	insertwidth	Insertion cursor की width represent करता है।
16	padx	Widget का horizontal padding।
17	pady	Widget का vertical padding।
18	relief	Border का type। Default SUNKEN है।

19	selectbackground	Selected text का background color।
20	selectborderwidth	Selected text के around border की width।
21	spacing1	यह specify करता है कि text की प्रत्येक line के above कितनी vertical space दी जाती है। Default 0 है।
22	spacing2	यह option specify करता है कि जब कोई logical line wraps होती है, तब displayed lines of text के between कितनी extra vertical space add की जाए। Default 0 है।
23	spacing3	यह specify करता है कि text की प्रत्येक line के below कितनी vertical space insert की जाए।
24	state	यदि state DISABLED पर set होती है, तो widget mouse और keyboard के प्रति unresponsive हो जाता है।
25	tabs	यह option control करता है कि tab character का use text को position करने के लिए कैसे किया जाता है।
26	width	Characters में widget की width represent करता है।
27	wrap	यह option wider lines को multiple lines में wrap करने के लिए used होता है। इस option को WORD पर set करें ताकि lines उस word के after wrap हों जो available space में fit होता है। Default value CHAR है जो line को उस character पर break करती है जो too wide हो जाती है।
28	xscrollcommand	Text widget को horizontally scrollable बनाने के लिए, हम इस option को Scrollbar widget की set() method पर set कर सकते हैं।
29	yscrollcommand	Text widget को vertically scrollable बनाने के लिए, हम इस option को Scrollbar widget की set() method पर set कर सकते हैं।

Text Widget Methods:

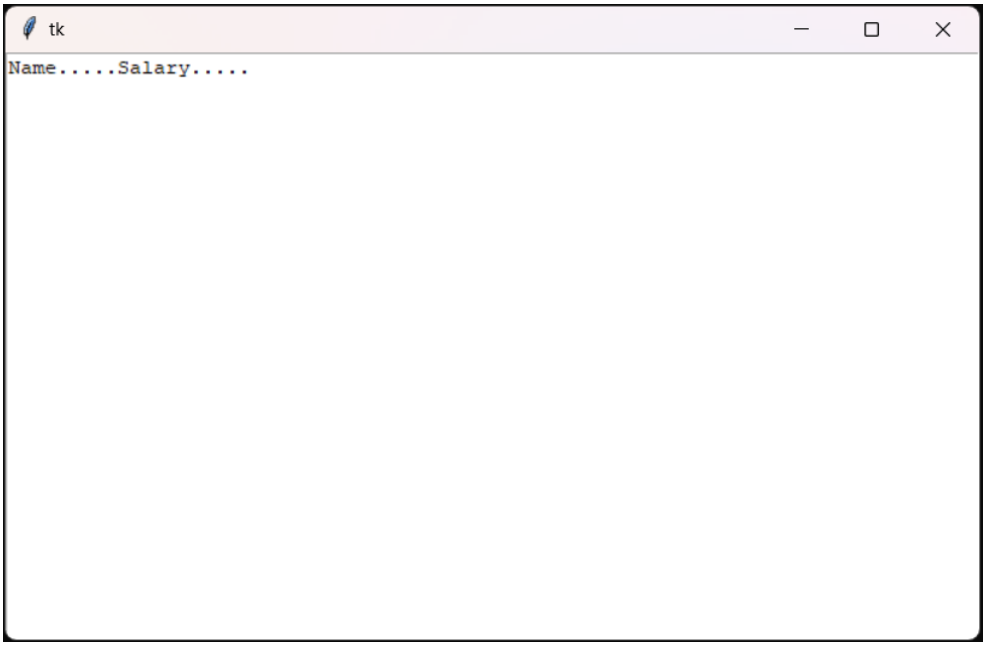
हम Text widget के साथ निम्नलिखित methods का use कर सकते हैं।

SN	Method	Description
1	delete(startindex, endindex)	यह method specified range के characters delete करने के लिए used होता है।
2	get(startindex, endindex)	यह specified range में present characters return करता है।
3	index(index)	Specified index का absolute index get करने के लिए used होता है।
4	insert(index, string)	Given index पर specified string insert करने के लिए used होता है।
5	see(index)	यह एक boolean value true या false return करता है depending upon कि specified index पर text visible है या नहीं।

Example -

```
from tkinter import *
top = Tk()
text = Text(top)
text.insert(INSERT, "Name.....")
text.insert(END, "Salary.....")
text.pack()
top.mainloop()
```

Output –



PanedWindow Widget

- PanedWindow widget एक Container widget की तरह act करता है जिसमें एक या अधिक child widgets (panes) होते हैं horizontally या vertically arranged।
- Child panes user द्वारा resized किए जा सकते हैं, mouse का use करके sashes के रूप में known separator lines move करके।
- PanedWindow python applications में different layouts implement करने के लिए used होता है।

Syntax -

w = PanedWindow(master, options)

Possible options की एक list नीचे दी गई है।

PanedWindow Widget Options:

SN	Option	Description
1	bg	Widget का background color represent करता है जब इसके पास focus नहीं होता है।
2	bd	Widget का 3D border size represent करता है। Default option specify करता है कि trough में कोई border नहीं होता है जबकि arrowheads और slider में 2-pixel border size होता है।
3	borderwidth	Widget की border width represent करता है। Default 2 pixel है।
4	cursor	Mouse pointer specified cursor type में change हो जाती है जब यह window पर over होता है।
5	handlepad	यह option handle और sash के end के between distance represent करता है। Horizontal orientation के लिए, यह sash के top और handle के between distance है। Default 8 pixels है।
6	handlesize	Handle का size represent करता है। Default size 8 pixels है। However, handle हमेशा एक square होगा।
7	height	Widget की height represent करता है। यदि हम height specify नहीं करते हैं, तो यह child window की height द्वारा calculated होगी।
8	orient	Orient HORIZONTAL पर set होगा यदि हम child windows को side by side place करना चाहते हैं। इसे VERTICAL पर set किया जा सकता है यदि हम child windows को top to bottom place करना चाहते हैं।
9	relief	Border का type represent करता है। Default FLAT है।
10	sashpad	प्रत्येक sash के around done की जाने वाली padding represent करता है। Default 0 है।
11	sashrelief	प्रत्येक sash के around border का type represent करता है। Default FLAT है।
12	sashwidth	Sash की width represent करता है। Default 2 pixels है।
13	showhandle	यह True पर set होता है handles display करने के लिए। Default value false है।
14	Width	Widget की width represent करता है। यदि हम widget की width specify नहीं करते हैं, तो यह child widgets के size द्वारा calculated होगी।

PanedWindow Methods:

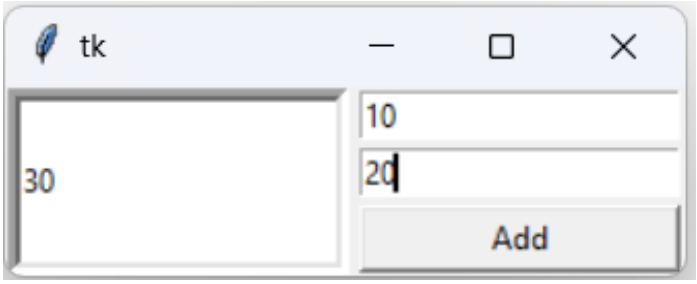
PanedWindow के साथ associated निम्नलिखित methods हैं।

SN	Method	Description
1	add(child, options)	Parent window में window add करने के लिए used होता है।
2	get(startindex, endindex)	यह method specified range में present text get करने के लिए used होता है।
3	config(options)	Widget को specified options के साथ configure करने के लिए used होता है।

Example –

```
from tkinterimport *

def add():

a = int(e1.get())

b = int(e2.get())

leftdata= str(a+b)

left.insert(1,leftdata)

w1 = PanedWindow()

w1.pack(fill = BOTH, expand = 1)

left = Entry(w1, bd = 5)

w1.add(left)

w2 = PanedWindow(w1, orient = VERTICAL)

w1.add(w2)

e1 = Entry(w2)

e2 = Entry(w2)

w2.add(e1)

w2.add(e2)

bottom = Button(w2, text = "Add", command = add)

w2.add(bottom)

mainloop()
```

Output -



Tkinter messagebox

- messagebox module python applications में message boxes display करने के लिए used होता है।
- Various functions हैं जो application requirements के depending upon relevant messages display करने के लिए used होती हैं।

Syntax -

```
messagebox.function_name(title, message [, options])
```

Parameters -

1. function_name: यह एक appropriate message box function represent करता है।
2. title: यह एक string है जो message box के title के रूप में shown होती है।
3. message: यह string है जो message box पर message के रूप में displayed होती है।
4. options: Various options हैं जिनका use message dialog box configure करने के लिए किया जा सकता है।

1. showinfo()

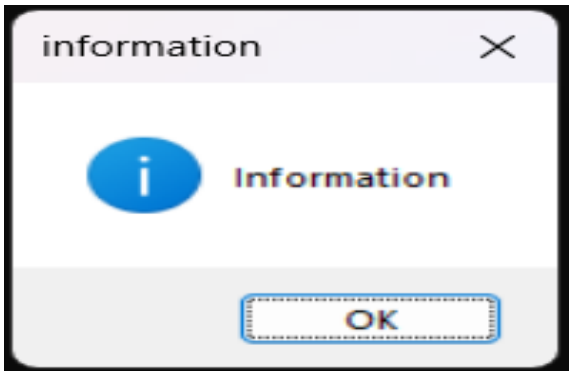
showinfo() messagebox वहां used होता है जहां हमें user को कुछ relevant information show करने की need होती है।

Example -

```
from tkinter import *

from tkinter import messagebox

top = Tk()
```

```
top.geometry("100x100")
messagebox.showinfo("information", "Information")
top.mainloop()
```

Output –



2. showwarning()

यह method user को warning display करने के लिए used होता है। निम्नलिखित example consider करें।

Example -

```
from tkinter import *
from tkinter import messagebox
top = Tk()
top.geometry("100x100")
messagebox.showwarning("warning", "Warning")
top.mainloop()
```

Output –

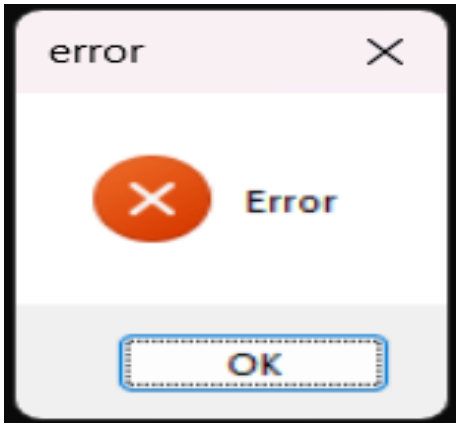
3. showerror()

यह method user को error message display करने के लिए used होता है। निम्नलिखित example consider करें।

Example -

```
from tkinter import *
from tkinter import messagebox
top = Tk()
top.geometry("100x100")
messagebox.showerror("error", "Error")
top.mainloop()
```

Output -



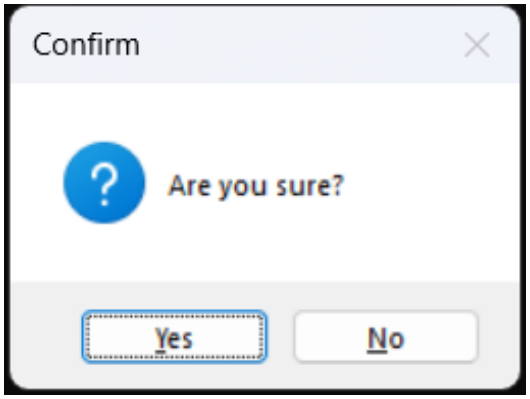
4. askquestion()

यह method user से कुछ question ask करने के लिए used होता है जिसका answer yes या no में दिया जा सकता है। निम्नलिखित example consider करें।

Example -

```
from tkinter import *
from tkinter import messagebox
top = Tk()
top.geometry("100x100")
messagebox.askquestion("Confirm", "Are you sure?")
top.mainloop()
```

Output –



5. askokcancel()

यह method user की action को confirm करने के लिए used होता है कुछ application activity के regarding। निम्नलिखित example consider करें।

Example -

```
from tkinter import *
from tkinter import messagebox
top = Tk()
top.geometry("100x100")
messagebox.askokcancel("Redirect", "Redirecting you to www.javatpoint.com")
top.mainloop()
```

Output –

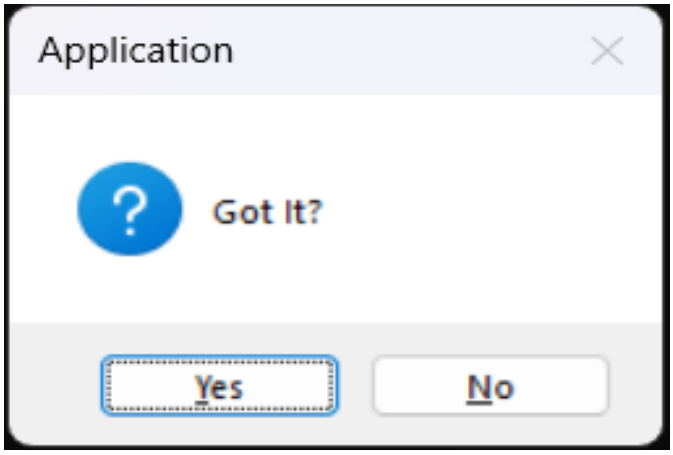
6. askyesno()

यह method user से कुछ action के बारे में ask करने के लिए used होता है जिसके लिए user yes या no में answer कर सकता है। निम्नलिखित example consider करें।

Example -

```
from tkinter import *
from tkinter import messagebox
top = Tk()
top.geometry("100x100")
messagebox.askyesno("Application", "Got It?")
top.mainloop()
```

Output –



7. askretrycancel()

यह method का use user से यह पूछने के लिए होता है कि क्या कोई particular task फिर से करना है या नहीं। निम्नलिखित example consider करें।

Example -

```
from tkinter import *
from tkinter import messagebox
top = Tk()
top.geometry("100x100")
messagebox.askretrycancel("Application", "try again?")
top.mainloop()
```

Output –