Introduction

Definition :- Python is a high-level, interpreted, general-purpose programming language that is widely used for web development, data analysis, artificial intelligence, automation, software development, and more

- Created by Guido van Rossum.
- Released in 1991.

Python used for

- Web Development
- Data Science & Analytics
- Machine Learning & AI
- Automation & Scripting
- Game Development
- Desktop Applications
- Networking & Cybersecurity
- Web Scraping
- Cloud & Big Data
- Internet of Things (IoT)

What can python do ?

- Build web applications on servers
- Automate workflows and repetitive tasks
- Connect to databases and manage files
- Handle big data and perform complex math
- Rapid prototyping and production-ready software
- Develop machine learning and AI models
- Create data visualizations and dashboards
- Perform web scraping and data extraction
- Build desktop and GUI applications
- Work with cloud services and IoT devices

Why did we use python ?

- Works on multiple platforms: Windows, Mac, Linux, Raspberry Pi, etc.
- Simple, readable syntax similar to English.
- Write programs in fewer lines compared to many other languages.

- Runs on an interpreter – code executes immediately, enabling fast prototyping.
- Supports multiple programming paradigms: procedural, object-oriented, functional.
- Huge standard library with ready-to-use modules.
- Strong community support and extensive documentation.
- Easily integrates with other languages like C, C++, and Java.
- Great for web development, data science, AI, machine learning, and automation.
- Open-source and free to use.
- Supports scripting and rapid application development.
- Easy to learn for beginners and powerful for experts.

Python Command Line

- Sometimes, the fastest and simplest way to test a small piece of Python code is to run it directly without saving it to a file, that's why we used the command line for the fastest test of code.
- Open terminal
- ```
  umesh2809@Umeshs-MacBook-Air ~ % python3
  Python 3.13.5 (v3.13.5:6cb20a219a8, Jun 11 2025, 12:23:45) [Clang 16.0.0 (clang-1600.0.26.6)] on darwin
  Type "help", "copyright", "credits" or "license" for more information.
  >>> print("Hello Umesh")
  Hello Umesh
  ```
- Used exit() function to exit that command line interpreter.

Python Indentation

- Indentation refers to the space to the starting of the code or program.
- Indentation in python is very important and used to show block of code but in other programming languages indentation is used only for readability.
  ```

Variables

- Variables are containers for storing data values.

Variable Names

1. Camel case
   Ex. addTwoNum

2. Pascal case
   Ex. AddTwoNum

3. Snake case
   Ex. add_two_num

Simultaneous Assignment of Variables

Python allows developers to assign multiple values to multiple variables simultaneously.
Ex.

```
a, b, c = 2, 3,1
```

Single value to multiple variables

Python allows us to assign single values to the multiple variables.
Ex.

```
a = b = c = 4
```

Unpacking Elements from a Collection

We can unpack the collection and assign the from that collection to the variables
Ex.

```
collectionName = [1,2,3,4]
x, y, z = collectionName
print(x)
print(y)
print(z)
```

Two type of variables

1. Global Variables
    - Created outside the function
    - Used by anywhere outside the function as well as inside the function
    - Ex

```
a = "sham"
def greet():
   print("Hello", a)
greet()
```

    - When we write the variable in the function with the same name, it will represent it as a local variable not as a global variable.

```
a = "Global" # global variabl
def greet():
  a= "Local" #local variable
  print("Hello", a)
greet()
print("Hello ",a)
```

- Global keyword
    - When we declare a variable into the function with the global keyword it will become the global variable, which can be accessed outside and also inside the function.

```
num1 = 5
def addition():
   global num2
   num2 = 10

addition()
print("addition of num1 and num2 :- ",num1 + num2)
```

- global keyword is also used in function to change the value of global variable.

```
name = "India"
print("Name before changing :-", name)
def ChangeGlobalVariable():
  global name
  name = "Bharat"

ChangeGlobalVariable()
print("Name after changing :- ",name)
```

2. Local Variables
   - Created inside the function
   - Used only in the function, outside access is not allowed.

```
def greet():
 a= "Local" #local variable
 print("Hello", a)
greet()
```

Strings

- A set of characters is called a string.
- Surrounded by single quotation or double quotation.

```
singleQuotation = 'Hello' #single quotation
doubleQuotation = "Python"  #double quotation

```

- Quotes inside quotes
  - We can use quotes inside single or double quotes to highlight a sentence.

```
print("It's me, born to win.")
print("Hello we are youth of new 'India'")
print('I want to learn "Python"')
```

- Multiline String.
  - We can assign multiline string to a variable using triple quotes.

```
lineOfMind = '''When You Expect
more than others expectations,
it hurts more than others expectations'''
print(lineOfMind)
```

- String As Array in Python
  - Like many other programming languages string in python work as an array.
  - Python does not have the char data type the single character value is consider as the string with length 1.
  - Square brackets “[ ]” are used to access the characters from the given string.

```
Name = "Karan"
print(Name[0])
```

- Function To Find String Length
  - We used **len()** function to find the length of the string.

```
Name = "Karan"
print("Length Of the value store into the Name variable :-", len(Name))
```

- Check Is String Is Present Or Not
  - Used “in” to check string is present, It gives True if string is present

```
name = "Maharashtra"
print("rashtr" in name)
```

  - Used “not in” to check the string is not present. It gives True if string is not present

```
name = "Maharashtra"
print("text" not in name)
```

- Slicing Strings
  - You can slice the string into the small part and used it by using slicing string process.
  - Syntax for string slicing
  - **string[start : stop : step]**
  - **Start :-** From where to start the slicing
  - **Stop :-** From where stop the slicing
  - **Step :-** Condition to do the slicing. It is default 1.

- Basic Slicing

  - In the basic slicing we only used **Start** and **Stop**

  1.

```
name = "Maharashtra"
print(name[0:4])
```

  -In the above example 0 is start point which is include in slicing & 4 is stop point which is exclude and step is default i.e. 1.

  2.

```
name = "Maharashtra"
print(name[0:])
```

- In the above example 0 is start point and we did not mention stop point to print the whole string. End point will be the “length + 1” of that string.

3.

```
name = "Maharashtra"
print(name[:4])
```

- We do not mention the start point just Stop point mention to print the string till “stop point -1”

4.

```
name = "Maharashtra"
print(name[:])
```

- Do not mention start and stop point so it take by default 0 for start point and length+1 for stop point.

5.

```
name = "Maharashtra"
print(name[0:6:2])
```

- used step to skip the characters from string

6.

```
name = "Maharashtra"
print(name[-6:-1])
```

- Use a negative index to slice the string from right side.

- For reverse traversal

```
name = "Maharashtra"
print(name[::-1])
```

- We used -1 for step to reverse the string

- Modify Strings

  - Python has some inbuilt method to modify the string or to handle the string.

    1. **Upper Case**

```
name = "Maharashtra"
print(name.upper())
```

    - This method updates all characters of the string into the upper case.

2. Lower Case

```
name = "Maharashtra"
print(name.lower())
```

- This method update all character of the string into the lower case.

3. Remove white spaces

```
name = "          Hello Maharashtra          "
print(name.strip())
```

- It removes the white space from start and end of the string

4. Replace method

```
name = "Hello Maharashtra"
print(name.replace("a", "A"))
```

- Replace new character instead of old one

5. split method

```
name = "Hello Maharashtra proud to be your son"
print(name.split())
```

- It split every word into as a list

- String Concatenation
  - Is the process to add two string as a single string

```
str1 ="Hello"
str2 = "Maharashtra"
print(str1+str2)
```

- String Format
  - We used it to print the string variable value into the output

```
str1 ="Hello"
str2 = "Maharashtra"
print("str1 value is :- "+ str1 +"yes it is correct")
print(f"str2 value is {str2} yes it is correct")
```

- Escape Characters

Code	Result	Example
\’	Single Quote	`name = 'Hello It\'s Aman'` `print(name)`
\\	Backslash	`txt = "This will insert one \\ (backslash) into your statement."` `print(txt)`
\n	New Line	`str = "Hello\nWorld!"` `print(str)`
\r	Carriage Return	`str = "Hello\rWorld!"` `print(str)`
\t	Tab	`str = "Hello\tWorld!"` `print(str)`
\b	Backspace	`str = "Hello \bWorld!"` `print(str)`

- String Method

capitalize()	Converts the first character of the string to uppercase and the rest to lowercase.
casefold()	Converts the entire string to lowercase, more aggressive than lower() (useful for comparisons).
center(width)	Returns the string centered in a field of given width with spaces (or custom fill).
count(substring)	Returns the number of occurrences of a specified substring.
encode()	Encodes the string into bytes using a specified encoding (default is UTF-8).
endswith(substring)	Returns True if the string ends with the specified substring.

expandtabs(tabsize)	Replaces tab characters (\t) with spaces using the specified tab size.
find(substring)	Searches for the substring and returns the lowest index where it is found. Returns -1 if not found.
format()	Formats specified values in a string using placeholders {}.
format_map(mapping)	Similar to format(), but uses a mapping (like a dictionary) for replacements.
index(substring)	Same as find() but raises a ValueError if the substring is not found.
isalnum()	Returns True if all characters are alphanumeric (letters or numbers).
isalpha()	Returns True if all characters in the string are alphabetic.
isascii()	Returns True if all characters in the string are ASCII (0–127).
isdecimal()	Returns True if all characters are decimal numbers (0–9).
isdigit()	Returns True if all characters are digits (including Unicode digits).
isidentifier()	Returns True if the string is a valid Python identifier (variable name).
islower()	Returns True if all alphabetic characters in the string are lowercase.
isnumeric()	Returns True if all characters are numeric (including digits, fractions, etc.).
isprintable()	Returns True if all characters in the string are printable.
isspace()	Returns True if the string contains only whitespace characters.
istitle()	Returns True if the string is in title case (each word starts with an uppercase letter).
isupper()	Returns True if all alphabetic characters in the string are uppercase.
join(iterable)	Joins the elements of an iterable (like a list) into a single string using the original string as a separator.
ljust(width)	Returns a left-justified version of the string with specified width.
lower()	Converts all characters in the string to lowercase.
lstrip()	Removes leading (left-side) whitespace or specified characters from the string.
maketrans()	Creates a translation table for use with the translate() method.
partition(separator)	Splits the string into a tuple of three parts: before, separator, and after.
replace(old, new)	Replaces all occurrences of the old substring with a new one.
rfind(substring)	Returns the highest index where the substring is found. Returns -1 if not found.
rindex(substring)	Same as rfind() but raises a ValueError if the substring is not found.
rjust(width)	Returns a right-justified version of the string with specified width.

rpartition(separator)	Splits the string into a tuple of three parts, starting the search from the right.
rsplit(separator)	Splits the string from the right side using the specified separator.
rstrip()	Removes trailing (right-side) whitespace or specified characters from the string.
split(separator)	Splits the string into a list using the specified separator. Default is space.
splitlines()	Splits the string at line breaks (\n) and returns a list.
startswith(substring)	Returns True if the string starts with the specified substring.
strip()	Removes leading and trailing whitespaces or specified characters.
swapcase()	Swaps the case of each character: lowercase becomes uppercase and vice versa.
title()	Converts the first letter of each word in the string to uppercase.
translate(table)	Returns a string where characters are replaced using a translation table.
upper()	Converts all characters in the string to uppercase.
zfill(width)	Pads the string on the left with zeros (0) until it reaches the specified width.

Boolean

- Booleans represent only two values either True or False.
- When you compare two values it gives the boolean value.

```
a=10
b = 9
print(a > b)
print(a == b)
print(a < b)
```

- In python most values are evaluated as True

```
print(bool(10))
print(bool("Sham"))
```

- Zero and null are not evaluated as True, it is evaluated as False

```
print(bool(0))
print(bool())
```

Operators

- Operators are used to perform the operations on the operands.
- `print(15 + 10)`
- 15 and 10 are operands and + is operator.

- List of Operators.
    1. Arithmetic operators
    2. Assignment operators
    3. Comparison operators
    4. Logical operators
    5. Identity operators
    6. Membership operators
    7. Bitwise operators

1. Arithmetic Operators

- We used arithmetic operators to do the basic mathematical operations

Operator	**Name**	**Example**	**Description**
+	Addition	x + y	Adds two values.
-	Subtraction	x - y	Subtracts right operand from the left.
*	Multiplication	x * y	Multiplies two values.
/	Division	x / y	Divides left operand by the right and returns a float.
%	Modulus	x % y	Returns the remainder of the division.
*	Exponentiation	x ** y	Raises x to the power of y.
//	Floor Division	x // y	Divides and returns the integer part (ignores remainder).

2. Assignment Operators

- Assignment operators used to assignment value to the variable

Operator	Name	Example	Description
=	Assignment	x = 5	Assigns the value on the right to the variable on the left.
+=	Add and Assign	x += 3	Adds 3 to x and assigns the result to x. (x = x + 3)
-=	Subtract and Assign	x -= 3	Subtracts 3 from x and assigns the result to x.
*=	Multiply and Assign	x *= 3	Multiplies x by 3 and assigns the result to x.
/=	Divide and Assign	x /= 3	Divides x by 3 and assigns the result (float) to x.
%=	Modulus and Assign	x %= 3	Gets the remainder of x divided by 3, assigns to x.

3. Comparison Operator

- Used to compare the two operands
- Always give results either True or False.

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

4. Logical Operator

- Logical operators are used to combine two or more conditions and return a result based on their truth values (True or False).

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10

or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverses the result	not(x < 5 and x < 10)

- For and
  - If both are true, only then the output is True otherwise False.

A	B	A and B
TRUE (1)	TRUE (1)	TRUE (1)
TRUE (1)	FALSE (0)	FALSE (0)
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

- For or
  - When both are False, then only output is False.

A	B	A or B
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

- For not

A	not A
TRUE	FALSE
FALSE	TRUE

5. Identity Operator
   - Identity operators are used to compare the memory locations of two objects, i.e., whether they refer to the same object in memory.

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

6. Bitwise Operator

- **Bitwise operators** are used to perform operations on the **binary (bit-level) representation** of integers.
- They compare or manipulate individual bits of numbers, making them useful for tasks involving low-level data processing, such as in embedded systems or optimization tasks.

Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	x & y
|	OR	Sets each bit to 1 if one of two bits is 1	X | y
^	XOR	Sets each bit to 1 if only one of the bits is 1	x ^ y
~	NOT	Inverts all the bits	~x
<<	Left Shift	Shifts bits left, filling with zeros (x << 2 is like multiplying by 4)	x << 2
>>	Right Shift	Shifts bits right, discarding right bits (x >> 2 is like dividing by 4)	x >> 2

- Table for XOR

Bit A	Bit B	A ^ B (Result)
0	0	0
0	1	1
1	0	1
1	1	0

Operator Precedence

- Operator precedence determines the order in which operators are evaluated in an expression when there are multiple operators involved.
- Precedence orders are given below from high precedence to low.

Operator(s)	Description
()	Parentheses
**	Exponentiation
*, /, //, %	Multiplication, division, floor division, modulus
+ , -	Addition and subtraction
<<, >>	Bitwise left and right shifts
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
==, !=, >, >=, <, <=, is, is not, in, not in	Comparisons, identity, and membership operators
not	Logical NOT
and	Logical AND
or	Logical OR

Python Collections

- 4 collection data types in python.
    - List
    - Tuple
    - Set
    - Dictionary

List

```
stdName = ["Chetan", "Yogesh", "Nilima"]
```

- Built in data type,
- Used to store multiple items into the single variable.
- Declare using square brackets “[ ]”.
- It is ordered, changeable and allows duplicate values.
- **Ordered** :- The item in the list has a fixed order, when we want to add the new item to the last of the list, the order of other items remains the same.
- **Changeable :-** We can change the list, means we can add and remove the list item.
- We can find the length of the list using the len**()** function.
- List items contain data with different data types.

```
stdName = ["Chetan", 2, True, 2.5, 1+7j]
print(stdName)
```

- We can use **list()** constructor to create a list, or to convert into the list.

```
stdName = list(("Chetan", 2, True, 2.5, 1+7j))
print(stdName)
```

- Just one thing, list() always contains two brackets.
- Indexing of lists starts from zero.
- Can access data using index.

```
stdName = ["Chetan", "Yogesh", "Nilima"]
print(stdName[0])
```

- We can access data using the -ve index also.

```
stdName = ["Chetan", "Yogesh", "Nilima"]
print(stdName[-1])
```

- List Slicing :-

- We can specify the range in the list slicing where to start and where to stop and print that part of the list also. Start is included and stop is excluded.

```
stdName = ["Chetan", "Yogesh", "Nilima", "Shivani", "Mayur","Pratiksha", "Lankesh", "Gayatri"]
print(stdName[2:5])
```

- Leaving out the start value the range starts from the zero and if leaving out the stop value the range will go the length of the list and print the whole list.
- For checking if an item is in the list or not we used **in** i.e. membership operator.
- We can change list items using the list indexing.

```
stdName = ["Chetan", "Yogesh", "Nilima", "Shivani", "Mayur","Pratiksha", "Lankesh", "Gayatri"]
stdName[0] = 1
print(stdName)
```

- Change items using range.

```
stdName = ["Chetan", "Yogesh", "Nilima", "Shivani", "Mayur","Pratiksha", "Lankesh", "Gayatri"]
stdName[0:2] = [2,3]
print(stdName)
```

- Insert Item

- We can insert the new item on the specific index using **insert()** method

```
stdName = ["Chetan", "Yogesh", "Nilima", "Shivani", "Mayur","Pratiksha", "Lankesh", "Gayatri"]
stdName.insert(0,1)
print(stdName)
```

- Append item

- We can add new item to the last of the list using **append()** method

```
stdName = ["Chetan", "Yogesh", "Nilima", "Shivani", "Mayur","Pratiksha", "Lankesh", "Gayatri"]
stdName.append("KGF")
print(stdName)
```

- Extend the list

- Want to add two list with each other or other collection to list we used **extend()** method.

```
stdName = ["Chetan", "Yogesh", "Nilima", "Shivani", "Mayur","Pratiksha", "Lankesh", "Gayatri"]
stdRollNo = [1,2,3,4,5]
stdName.extend(stdRollNo)
print(stdName)
```

```
stdName = ["Chetan", "Yogesh", "Nilima", "Shivani", "Mayur","Pratiksha", "Lankesh", "Gayatri"]
stdRollNo = (1,2,3,4,5)
stdName.extend(stdRollNo)
print(stdName)
```

```
stdName = ["Chetan", "Yogesh", "Nilima", "Shivani", "Mayur","Pratiksha", "Lankesh", "Gayatri"]
stdRollNo = {9,5,6,8}
stdName.extend(stdRollNo)
print(stdName)
```

- For dictionary only keys will extend

```
stdName = ["Chetan", "Yogesh", "Nilima", "Shivani", "Mayur","Pratiksha", "Lankesh", "Gayatri"]
stdRollNo = {"x":20, "y":40}
stdName.extend(stdRollNo)
print(stdName)
```

- Remove list item from list

- We used **remove()** to remove the specific item from list.
- If there are two same item in list remove() method will remove the first occurrence only.

```
stdName = ["Chetan", "Yogesh", "Nilima", "Shivani", "Mayur","Pratiksha", "Lankesh", "Gayatri"]
stdName.remove("Chetan")
print(stdName)
```

- The **pop()** method will remove the item from the specified index.
- When we do not specify the index **pop()** method will remove the last item of list

```
stdName = ["Chetan", "Yogesh", "Nilima", "Shivani", "Mayur","Pratiksha", "Lankesh", "Gayatri"]
```

```
stdName.pop(0)
print(stdName)
```

- **del** keyword also removes the specified index item from the list.
- The del keyword can also delete the list completely.

```
stdName = ["Chetan", "Yogesh", "Nilima", "Shivani", "Mayur","Pratiksha", "Lankesh", "Gayatri"]
del stdName[7]
print(stdName)
```

- **clear()** method will clear the list and make the list empty.

```
stdName = ["Chetan", "Yogesh", "Nilima", "Shivani", "Mayur","Pratiksha", "Lankesh", "Gayatri"]
stdName.clear()
print(stdName)
```

- Sort list

- We used the sort() method for sorting the items into the list of strings by alphabetic order.

```
name = ["yogesh", "Ganesh","Kaveri","Neha","Arbaj"]
name.sort()
print(name)
```

- When a list contains an integer value it sorts by ascending order.

```
name = [100, 80,30,101, 5]
name.sort()
print(name)
```

- Sort by descending order

- We can sort the list into descending order using **reverse = True**.

```
name = [100, 80,30,101, 8]
name.sort(reverse=True)
print(name)
```

- If a list contains both capital and small letter word, **sort()** will sort the capital letter word first then goes towards the small letter word.

```
name = ["aman","karan","z","y","A", "Z"]
```

```
name.sort()
print(name)
```

- If we have to sort a list containing both capital and small letters together, we use str.lower() as the key.

```
name = ["aman","karan","z","y","A", "Z","C"]
name.sort(key = str.lower)
print(name)
```

- We can use the **reverse()** inbuilt method to reverse the list.

```
name = ["aman","karan","z","y","A", "Z","C"]
name.reverse()
print(name)
```

- We cannot copy the list by using
  List2 = List1
- This just gives the reference of List 1 to the List2, changes of List1 will directly reflect to List2.

```
name1 = ["aman","karan","z","y","A", "Z","C"]
name2 = name1
print(name1)
name1.pop()
print(name2)
```

- To copy the list into another list we used the copy**()** inbuilt method.

```
name1 = ["aman","karan","z","y","A", "Z","C"]
copyList = name1.copy()
name1.pop()
print(copyList)
print(name1)
```

- ## Join Two List

  We can use the “+” Operator to join or concatenate two lists.

```
name1 = ["aman","karan","z","y","A", "Z","C"]
name2 = [1,2,3,4,5]
```

```
addList = name1 + name2
print(addList)
```

List of inbuilt methods used in List.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Adds the elements of a list (or any iterable) to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Tuples

- Tuples are used to store multiple items in a single variable.
- Used to store collection of data.
- Tuple is ordered and unchangeable, allowing duplicate values.
- Tuples are written with round brackets.

```
name = ("Karan","Chetan","Kavya")
print(name)
```

- Tuple items are indexed, start from 0.
- We can use len() to find the length of the function

- Create a tuple with one item.

```
name = ("Karan",)
print(name)
```

- We can add data of multiple datatypes in tuple

- tuple() constructor :- used to make tuples.

```
thistuple = tuple([1,2,3,4,5])
print(thistuple)
print(type(thistuple))
```

- Access data from tuple

- We can access tuple items by using indexes into the square brackets.
- We can also use negative indexes for accessing the data.
- We can use a range of indexes to access data using the specified range, start index is included and stop is excluded.
- When we do not pass start value it will start from 0
- When we do not pass stop value it will go till end of the list
- For checking if an item exists in tuple we used **in** keyword.

- Update Tuple Value

- Tuple is unchangeable or immutable but we can change it by following some steps.
- tuple ⇒ convert to list ⇒ update list ⇒ convert it to tuple.
- We can add tuples with each other by using the "+=" operator, there is no extended method to add tuples with each other, we can add multiple tuples using the same.
- We can delete tuples completely by using the **del** keyword.

- Use of * (asterisk) in tuple

- When we unpackaging the tuple and the tuple item is more than the assigned variable then we use an asterisk to the last variable so the remaining will be assigned as a list.

```
number = (1,2,3,6,7,8,0)
(a,b,*c) = number
print(a)
print(b)
print(c)
```

- Tuple Methods

Method	Description
count()	Returns how many times a particular value appears in the tuple
index()	Finds the position of the first occurrence of a specified value in the tuple

Sets

- Used to store multiple items in a single variable.
- Set is an unordered, unchangeable and unindexed collection.
- The set is unordered because the order of items in the set is not fixed, but when we use a set with int value it performs ordered behavior.
- We can add and remove data from the set but it is unchangeable because it is unindexed.
- Used {} for set representation
- No duplication allowed in set.
- True and 1 are treated as the same, when we use the same set it looks like duplicate.
- We can use the len**()** method to check how many items the set has used.
- The set can be of any data type.
- We can use **set()** to make a set.

- Access set items.

- We cannot access set items using index, but we can access it by using a for loop.

- Add items into the set.

- We can use the **add()** method to add data into the set.
- If we want to add two sets or combine two sets we can use the **update()** method, we can add any other object to the set using it.

- Remove items from the set.

- We can use **remove() or discard()** method to remove data from the set.
- When we remove by using remove() method and If item does not exist then it raises the error.
- When we remove by using the discard() method and If item not exist then it not raise the error.
- We can also use the **pop()** method to remove the data, but it remove random data from the set.

- **clear()** method gets empty set i.e. remove all items from the set.
- **del** keyword is used to delete the set.

- **Join set**
- **union() and update()** will join the set with all the values.
- union() creates a new set after join, update() do not create new set after join process.
- We can use | instead of union() method, we can join only set using | not other data types.
- We can also join multiple sets using **union()** method and |.
- union() method allows us to join sets with other collections like tuple or list.
- **intersection()** keep only duplicate values
- It returns a new set after applying intersection() on two sets, we can use & operator instead of intersection() method.
- We can use intersection() to set with other type of collection but not with &
- **intersection_update()** this method also returns the duplicate elements but it will update the old set and not create any new set.
- **difference()** will return a new set which keeps the items that are in the first set but not in the second set.
- We can use the "-" operator instead of the difference() method.
- The "-" operator only allows you to join sets with sets, and not with other data types like tuple, list etc..
- **difference_update()** method will also keep the items from the first set that are not in the other set, but it will change the original set instead of returning a new set.
- **symmetric_difference()** keep all items except duplicates
- Can use ^ operator instead of symmetric_difference() method.

Dictionaries

- Dictionaries are used to store data in key : value pair.

```
data = {
  "name" : "Sachin",
  "RollNo" : 18,
  "Address" : "Niphad",
  "Class" : "12th"
}
print(data)
```

- A dictionary is a collection which is ordered, changeable and does not allow duplicates.
- Dictionary are represented using curly brackets {}
- We store data using key value pairs. Access using key only.

```
data = {
  "name" : "Sachin",
  "RollNo" : 18,
  "Address" : "Niphad",
  "Class" : "12th"
}
print(data["name"])
```

- Duplicate values are not allowed in the dictionary.

```
data = {
  "name" : "Rakesh",
  "class" : "SSC",
  "address":"Nashik",
  "address":"Niphad"
}
print(data)
```

- Used len() function to find the length of the dictionary.
- Values of dictionary data can be of any data type.
- We used the type() function to confirm the type of given dictionary.
- We used dict() constructor to convert other collection into dictionary.
- Access item using key.
- get() method can also used to access the data

```
thisdict = (("name","umesh"),("address","niphad"))
dictionary = dict(thisdict)
print(dictionary.get("name"))
```

- key() method return list of all keys into the dictionary.
- values() method return list of all values from the dictionary.
- item() method returns all values of the dictionary with keys in the form of tuple.
- We can use it to check if the keys are in the dictionary or not.

- Can change the value by referring to the key.
- For updating the value or item in the dictionary we can use update() method.

```
data = {
  "name":"Umesh",
  "address":"Niphad",
  "college":"kkw"
}
data.update({"rollNo":12})
print(data)
```

- Add Items into the Dictionary.

- We can add items into the dictionary using assigning value to the key of that dictionary.

```
data = {
  "name":"Umesh",
  "address":"Niphad",
}
data["hello"]="umesh"
print(data)
```

- **Removing items from the dictionary.**
- We used pop() to remove the item from the dictionary.

```
data = {
  "name":"Umesh",
  "address":"Niphad",
  "college":"kkw"
}
data.pop("name")
print(data)
```

- popitem() method removes the last inserted item from the dictionary.

```
data = {
  "name":"Umesh",
  "address":"Niphad",
  "college":"kkw"
}
data.popitem()
print(data)
```

- Using the del keyword we remove the item specified with the key.

```
data = {
  "name":"Umesh",
  "address":"Niphad",
  "college":"kkw"
}
del data["address"]
print(data)
```

- The del keyword can also delete the whole dictionary.

```
data = {
  "name":"Umesh",
  "address":"Niphad",
  "college":"kkw"
}
del data
print(data)
```

- clear() method uses the empty dictionary.

```
data = {
  "name":"Umesh",
  "address":"Niphad",
  "college":"kkw"
}
data.clear()
print(data)
```

- Copy a dictionary

- We can not copy the dictionary directly as
  Dict1 = Dict2
  These refer the Dict2 to the Dict1
  Changes in Dict2 will refer into the Dict1.
- To make a direct copy of one dictionary to another variable we used the copy() method.
- We can also copy the dictionary by using dict() operator.

```
data = {
  "name":"Umesh",
  "address":"Niphad",
  "college":"kkw"
}
data1 = dict(data)
```

```
data.popitem()
print(data1)
print(data)
```

- ## Dictionary Methods

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Conditional Statements

If Statement

- Usually if a statement is used to check the single condition.

```
a = 15
if a <= 20:
  print(f"{a} is less than 20")
```

If ... Else Statement

- We used these statements to check two conditions.
- If one condition fails then the other will execute automatically.

```
a = 15
if a <= 20:
  print(f"{a} is less than 20")
else:
  print(f"{a} is not less than 20")
```

Elif Statement

- In Python, elif is used to check another condition if the previous if (or elif) condition was not true

```
a = -10
if a < 0:
  print("Number is negative")
elif a > 0:
  print("Number is positive")
else:
  print("Number is zero")
```

Shortcut If

- We can write if statement in one line

```
a = -1
if a < 0: print("Number is negative")
```

Shortcut If … else

- We can write if and else into one line.

```
a = 19
print("Number is negative") if a < 0  else print("number is positive")
```

- And
  - and is a logical operator which is used to combine the two conditions into the if else statement.

```
a = 5
b = 10
if a < b and a > b:
  print("Both conditions are true")
```

  - If both conditions are True then and only then the if statement will execute.
- Or
  - or is a logical operator which is used to combine the two conditions into the if else statement.

```
a = 5
b = 10
c = 15
if a < b or b > c or a > c:
  print("at least one condition is true")
```

  - While using or, at least one condition will be true.
- Not
  - It used to reverse the result of the conditional statement.

```
a = 5
b = 10
c = 15
if not a > b:
```

```
    print("a is not less than b")
```

Nested If

- We can use the if statement inside the if statement.

```
gender = "female"
age = 24
if gender == "female":
  if age >= 18:
      print("She can vote")
```

Pass Statement

- If we have to write the if statement but do not want to write anything into it just give the pass.
- Or if you do not want to execute that block we can use a pass statement.

```
a = 33
b = 100
if b > a:
   pass
else:
  print("Pass execute")
```

Match Statement

- It is the same statement as switch case in other languages.
- We used a match statement when we have to use multiple conditions and do not want to use the if else statement, It selects one block of code from many blocks.
- Syntax :-

```
match expression:
  condition 1:
      block of code
  condition 2:
      block of code
```

- When the execution starts the value of expression is compared to every condition.

```
name = "Shubham"
match name:
    case "Umesh":
        print("Name of candidate is Umesh")
    case "Chetan":
        print("Name of candidate is Chetan")
```

- Default Value

- We used default value or default case when there is no match case then and only then the default case will execute.
- We used (_) underscore for the default case.

```
name = "Rajesh"
match name:
    case "Umesh":
        print("Name of candidate is Umesh")
    case "Chetan":
        print("Name of candidate is Chetan")
    case "Shubham":
        print("Name of candidate is Shubham")
    case _:
        print("There is no any candidate with these name")
```

- The value default gives error if we do not place it in last so keep it at the last case

- Combine Values

- you can check multiple values in one case using the pipe (|) as an OR operator.

```
name = input("Enter name:-")
match name:
    case "Umesh" | "Rajesh" | "Mukesh":
```

```
        print("Candidate into 1st Category")
    case "Chetan" | "Shubham" | "Akash":
        print("Candidate into 2nd Category")
    case _:
        print("There is no any candidate with these name")
```

- If we want to add extra conditions then we can use the if statement into the case part or in the condition part.

```
name = input("Enter name:-")
age = 20
match name:
    case "Umesh" | "Rajesh" | "Mukesh" if age == 18:
        print("Candidate into 1st Category")
    case "Chetan" | "Shubham" | "Akash" if age >=20:
        print("Candidate into 2nd Category")
    case _:
        print("There is no any candidate with these name")

```

Loops

- There are two loops in python
    - While loop
    - For loop

1. While Loop

- In a while loop we can execute a set of statements till the condition is true.

```
i = 1
while(i <= 20):
    if i % 2 == 0:
        print(i)
    i += 1
```

- When we do not increment the value of i, the loop will go infinitely.

- Break Statement

- It used to break the while loop at that point.

```
i = 1
while(i <= 20):
    if i % 2 == 0:
        print(i)
        break
    i += 1
```

- Continue Statement

- To skip the iteration which is true on that condition and move to the next iteration.

```
i = 1
while(i <= 20):
```

```
    i += 1
    if i == 4:
        continue
    print(i)
```

- **else**
- We run the block of code once when the condition no is not true.

```
i = 1
while(i <= 20):
    i += 1
    if i == 4:
        continue
    print(i)
else:
    print("4 is number which was skip above")
```

2. For Loop

- when you want to repeat a block of code a certain number of times or go through each item in a collection (like a list, tuple, string, or range).

```
name = ["umesh","rakesh","ramesh"]
for i in name:
    print(i)
```

- In python for loop does not need initialize indexing variable

- Looping through String

- We can add loop through the string and print one by one character of that string

```
name = "Umesh"
for i in name:
    print(i)
```

- Break Keyword.

- We used break keyword to stop the iteration at that point.

```
name = ["umesh","rakesh","ramesh"]
for i in name:
    if i == "rakesh":
        break
    print(i)
```

- Continue statement

- Using the continue statement we can stop the current iteration and move on to the next iteration.

```
name = ["umesh","rakesh","ramesh"]
for i in name:
    if i == "rakesh":
        break
    print(i)
```

- range() function

- To iterate the loop for a specific number of times.

1. range(5)

- Iterate the loop till the 5 starting from zero and increment by 1 by default.

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

2. range(1,5)

- Loop will iterate from 1 to 6 but 6 not included for the iteration, and increment will be by 1 by default.

```
for i in range(1,5):
    print(i)
```

3. range(1, 6, 2)

- Loop will iterate from 1 to 6, but 6 is not included and increment will be done by 2 steps.

```
for i in range(1, 6, 2):
    print(i)
```

- Else in for loop

- else part of the for loop will be executed once when the loop execution finished.

```
for x in range(10):
  print(x)
else:
   print("Finally finished the for loop!")
```

- It is not executed if the for loop will stop by the break statement.

- Nested For Loops

- Nested for loop is a loop inside the loop.
- Inner loop will be executed at least one time for every iteration of the outer loop.

```
students = ["umesh", "rakesh", "kranti"]
for student in students:
   for marks in range(80, 101, 10):
       print(student, marks)
```

- pass statement

- The pass statement is a placeholder that does nothing.
- It's used when a statement is required syntactically, but you don't want to execute any code yet.

```
students = ["umesh", "rakesh", "kranti"]
for student in students:
   for marks in range(80, 101, 10):
      #logic is not decided yet
      pass
```

Functions

- A function is a block of code which runs when we call that function by object.
- We can pass the data into the function we called it as parameters.
- A function can return data as a result.

- How to create a function ?

- **def** keyword is used to define a function in python.

```
def greeting():
    print("Hello Umesh")
```

- Declaring The Function.

- For calling the function we write the function name with parenthesis.

```
def greeting():
    print("Hello Umesh")
greeting()
```

- Arguments Used In Functions.

- Information can be passed to the function called arguments.
- Arguments are passed after the function name into the parenthesis. Arguments can be more than one but separated by commas.
- Below is the example in which we pass the argument and print them using the object.

```
def greeting(fname):
    print("My name is ",fname)
greeting("Aman")
greeting("Anushka")
greeting("Sham")
```

- Name arguments and parameters are used for the same meaning “To pass the information to the function.”

```
def greeting(fname):                # fname is the parameter
    print("My name is ",fname)
greeting("Aditya")              # Aditya is the argument
```

- Number of Arguments and Number of Parameters

- Number of Arguments == Number of Parameters.

```
def greeting(fname, lname):                # fname and lname is
the parameter
    print("My name is ",fname,"",lname)
greeting("Aditya","Shinde")                  # Aditya and
Shinde is the argument
```

- If the number of arguments and parameters are not equal then we get the error.
- *missing 1 required positional argument*

- Below are some important concepts related to function arguments and usage:

1. Arbitrary Arguments (`*args`)

- If you don’t know how many arguments will be passed into your function, use *args. This collects all extra positional arguments into a **tuple**

```
def my_function(*kids):
    print("Kids name are :- ",kids)
    print("Name of Second Kid :- ",kids[1])
my_function("Rahul","Tejas","Radha","Raju","Kaveri")
```

- Here the kids are in tuple form.

2. Keyword Arguments (`kwargs`)

- You can pass arguments using key = value syntax.
- Order does not matter because arguments are matched by their names.

```
def my_function(boy1, boy2, boy3):
    print("Name of First boy :- ",boy1)
    print("Name of Second boy :- ",boy2)
    print("Name of Third boy :- ",boy3)
my_function(boy3 = "Chetan", boy2 = "Rakesh", boy1 =
"Kedar")
```

3. Arbitrary Keyword Arguments (**kwargs)

- If you don't know how many keyword arguments will be passed, use **kwargs.
- This collects all extra keyword arguments into a **dictionary**.

```
def StudentInfo(**kwargs):
    print("First name of student :- ",kwargs["firstName"])
    print("Middle name of student :- ",kwargs["middleName"])
    print("Last name of student :- ",kwargs["lastName"])
StudentInfo(firstName = "Chetan", middleName = "Uttam",
lastName = "Khairnar")
```

4. Default Parameter Value

- You can assign default values to parameters.
- If no value is passed, the default is used.

```
def select(lang = "Python"):
    print("My favorite programming language is ",lang)
select("Java")
select("C")
select()
select("C++")
```

5. Passing a List as an Argument

- Any data type can be passed to a function (string, number, list, dictionary, etc.).

```
def select(Organization):
    for x in Organization:
        print(x)
companyName = ["TCS","Tata Technologies","WIPRO","Infosys"]
select(companyName)
```

6. Return Values

- Functions can return results using the **return** keyword.

```
num1 = int(input("Enter the number :- "))
def Add(num1):
    return 5 + num1
print(Add(num1))
```

7. The `pass` Statement

- Functions cannot be empty, If you want an empty function (maybe to define later), use **`pass`**.

```
num1 = int(input("Enter the number :- "))
num2 = int(input("Enter the number :- "))
def Add(num1):
    pass
print("First Function return the addition ",Add(num1))
def Sub(num2):
    return num1 + num2
print("Second function return the subtraction :-
",Sub(num2))
```

Function Decorator

- Function that takes another function as an input and adds extra features to it.
- Decorator in python is a special function that modifies or adds extra features to another function, without changing its code.
- The main objective of decorator functions is we can extend the functionality of existing functions without modifying that function.
- Very common in python Framework like Django, Flask, etc.
- Ex. :- Covering our phone with the phone cover does not change its functionality but it now has extra protection and style.
- Syntax for Decorator

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

- Example :-

```
def showroomEnquiry(func):
    def Enquiry():
        print("Enquiry start...!")
        func()
        print("Enquiry End...!")
    return Enquiry
@showroomEnquiry
def carModel():
    print("This car model is very good..!")
carModel()
```

Recursion

- A function **calls itself** again and again until it reaches a point where it stops.
- This "stopping point" is called the **base case**. Without it, the function would call itself forever.

Lambda

- Is a small anonymous function.
- It takes any number of arguments but only one expression.
- Syntax for lambda Function

```
x = lambda variable : expression
```

- Example :-

```
add = lambda x,y : x+y

print(add(2,5))
```

Python Arrays

- Python does not have an inbuilt array. But list in python work as an array.
- Arrays in python can store multiple data into the single variable.

```
cityName = ["Nashik","Pune","Mumbai","Thane"]
```

What is an Array?

- Array is a special variable used to store multiple data at a time.

Access the elements from an array.

- We can access data from array by using array indexing.

```
cityName = ["Nashik","Pune","Mumbai","Thane"]
print(cityName[0])
```

We can also modify the value of array using array indexing

- We can modify the array by assigning updated value to the index.

```
cityName = ["Nashik","Pune","Mumbai","Thane"]
print("Old Array :-",cityName)
cityName[0]="Pimpri Chinchawad"
print("New Array :-",cityName)
```

- By using the **len()** method we can find the length of the array. (number of item in that array)

```
cityName = ["Nashik","Pune","Mumbai","Thane"]
print("Old Array :-",cityName)
print("Length of array :-",len(cityName))
```

- The length of an array is always one more than the highest index of the array.

Looping array element

```
cityName = ["Nashik","Pune","Mumbai","Thane"]
```

```
for i in cityName:
   print(i)
```

Adding Data to the Array

- Used append() to add data into array

```
- cityName = ["Nashik","Pune","Mumbai","Thane"]
- cityName.append("Niphad")
- print(cityName)
```

Removing data from array using pop() and remove() method

- We can remove data using the pop() method in two ways.

  1. Mention index number to remove data on that index.

```
studentName = ["Umesh","Chetan","Akash","Rutik"]

studentName.pop(0)

print(studentName)
```

  2. Do not mention any index directly; use the pop method it will delete the last data of the array.

```
studentName = ["Umesh","Chetan","Akash","Rutik"]

studentName.pop()

print(studentName)
```

- We can also remove data using remove method by directly mentioning the data into the remove() method.

```
studentName = ["Umesh","Chetan","Akash","Rutik"]
studentName.remove("Umesh")
print(studentName)
```

Array Methods

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Modules

- A group of functions, variables and classes saved to a file, which is nothing but a module.
- Every python file(.py) acts as a module.
- **practice.py**

```
num = 1001

def add(a, b):
    print("Addition is ",a+b)


def sub(a,b):
    print("Substraction is ",a-b)


def product(a,b):
    print("Multiplication is ",a*b)
```

- **demo.py**

```
import practice

print(practice.num)

practice.add(2,3)

practice.product(2,3)
```

- In the demo file we can import the module file by using import keyword, and used the function and variables from that module file.
- We can access the variable like **modulefileName.variableName**
- Also can access the function like
- **moduleFileName.functionName**
- Whenever we use a module file for our program the compile file for that module will be created and stored into the hard disk permanently.

- Renaming the module at the time of importing the module.

- Ex. import practice as p

- Here practice is original module and p is the renaming of that module
- We can access the members from the module file by using the renaming part of the module.

```
import practice as p
print(p.num)
p.add(2,3)
p.product(2,3)
```

- We can import a particular member directly by using (from… import….)
- The advantage is we can access members directly without writing the module name.

```
from practice import add, sub, product, num
print(num)
add(2,3)
product(2,3)
sub(2,3)
```

- We can also import all module from module file by using asterisk sign (*) like below.
- **from practice import***

```
from practice import*
print(num)
add(2,3)
product(2,3)
sub(2,3)
```

Various ways of importing modules into the file.

- import modulename
- import module1, module2, module3..
- import module1 as m1
- import module1 as m1, module1 as m2, module3 as m3
- from module import member
- from module import member1, member2, member3
- from module import member1 as x
- from module import*

Member aliasing

```
from demo import num as n, add as a
print(n)
a(2,3)
```

- In these we try to alias the member of the module file.
- When we define the alias name for the member we have to use that alias name only, for accessing that member, if not then it gives an error.

Working with math module:

- Python provides inbuilt module **math**.
- This module defines several functions which can be used for mathematical operations.
- **Below are some examples of a math module.**

Sr. No	Function / Constant	Example	Used For
1	math.pi	math.pi → 3.14159	Value of π
2	math.e	math.e → 2.71828	Value of e (Euler's number)
3	math.inf	math.inf	Infinity
4	math.nan	math.nan	Not a Number
5	math.ceil(x)	math.ceil(4.3) → 5	Rounds up to nearest integer
6	math.floor(x)	math.floor(4.9) → 4	Rounds down to nearest integer
7	math.trunc(x)	math.trunc(4.7) → 4	Removes decimal part
8	math.fabs(x)	math.fabs(-5) → 5.0	Absolute value
9	math.sqrt(x)	math.sqrt(16) → 4.0	Square root
10	math.isqrt(x)	math.isqrt(17) → 4	Integer square root
11	math.pow(x, y)	math.pow(2, 3) → 8.0	Power (x^y)
12	math.exp(x)	math.exp(2) → 7.389	e^x
13	math.log(x)	math.log(10) → 2.302	Natural log (base e)
14	math.log10(x)	math.log10(100) → 2	Logarithm base 10
15	math.log2(x)	math.log2(8) → 3	Logarithm base 2
16	math.factorial(n)	math.factorial(5) → 120	Factorial (n!)
17	math.comb(n, r)	math.comb(5, 2) → 10	Combinations (nCr)
18	math.perm(n, r)	math.perm(5, 2) → 20	Permutations (nPr)
19	math.gcd(a, b)	math.gcd(12, 18) → 6	Greatest Common Divisor
20	math.lcm(a, b)	math.lcm(12, 18) → 36	Least Common Multiple
21	math.sin(x)	math.sin(math.radians(90)) → 1.0	Sine (in radians)

22	math.cos(x)	math.cos(math.radians(0)) → 1.0	Cosine
23	math.tan(x)	math.tan(math.radians(45)) → 1.0	Tangent
24	math.asin(x)	math.degrees(math.asin(1)) → 90.0	Inverse Sine
25	math.acos(x)	math.degrees(math.acos(1)) → 0.0	Inverse Cosine
26	math.atan(x)	math.degrees(math.atan(1)) → 45.0	Inverse Tangent

- We can use the help() method to find the extra information about the module.

```
import helper
help(helper)
```

Use of Random Module.

1. random() function

- This module defines several functions to generate random numbers.
- This function alway generate value between 0 to 1, 1 is excluded from it

```
from random import*
for i in range(2):
    print(random())
```

2. randint() function

- To generate a random integer number between a given range.
- Start and end range number are inclusive

```
from random import*
for i in range(5):
    print(randint(1,10))
```

3. uniform() function

- We used uniform() function to return a random float number between a given range of numbers.

```
from random import*
for i in range(5):
    print(uniform(1,10))
```

4. randrange() function

- It returns a random number from range
  Start <= x < stop,
  Start ⇒ start argument is optional and default value is 0
  Step ⇒ step argument is optional and default value is 1.
- randrange(5) generate the number from 0 to 4

```
from random import*
for i in range(5):
    print(randrange(5))
```

- randrange(1,6) generates the number from 1 to 5.

```
from random import*
for i in range(5):
    print(randrange(1,6))
```

- randrange(1,10,2) generates the number from 1,3,5,7,9

```
from random import*
for i in range(5):
    print(randrange(1,10,2))
```

5. choice() function

- This function does not return any random number, it returns the random object from the list or tuple.

```
from random import*
name = ["ketan","kaveri","kavya","kajol","kartiki"]
for i in range(3):
    print(choice(name))
```

File Handling

- We used file handling for storing our data permanently for the future purpose.
- Files are a very common storage area to store our data.

Two Types Of File

1. Text File

- We can use this file for storing text data.
- We used the .txt extension for storing text files.

2. Binary File

- We used binary files to store data like images, video files, audio files etc..

Opening a file.

- Before the read and write operation we have to open the file. We used **open()** python inbuilt function to open the file
- But at the time of opening the file we have to specify the mode for opening the file like below.
  F = open(filename, mode)

Closing a file.

- After completing the operation on file we have to close the file, it is highly recommended to close the file.
- For this we have to use the **close() function** for closing the file.

- Allowed modes in python are as follows.

1. r ⇒ Open an existing file for read operation.
   - If the specified file is not exist then it gives error, **"FoundNotFoundError"**

```
# First, make sure demo.txt exists with some text
f = open("demo.txt", "r")
print(f.read())    # Reads and prints file content
f.close()
```

2. w ⇒ Open an existing file for writing operation.
   - If an existing file contains some data it overrides the new data, if the file does not exist then it creates a new file.

```
f = open("demo.txt", "w")
f.write("This text will replace old data")
f.close()
```

- **Note :- if we want to add multiple lines then we have to use writelines() function**

```
f = open("newfile.txt", "w")
f.writelines("This file \n is created \n with x mode.")
f.close()
```

3. a ⇒ Open an existing file for an append operation.
    - It won't override the existing data, insert data after the existing data. If the specified file is not available then it will create a new file.

```
f = open("demo.txt", "a")
f.write("\nThis new line is added at the end")
f.close()
```

4. r+ ⇒ To read & write an existing file.
    - It does not override the existing data, these modes insert the new data at the starting of the file.

```
f = open("demo.txt", "r+")
print("Before:", f.read())    # Reads file
f.seek(0)                     # Move cursor to start
f.write("NewData")            # Overwrites from beginning
f.close()
```

5. w+ ⇒ To write & read data.
    - It overrides the existing data in the file.

```
f = open("demo.txt", "w+")
f.write("Fresh content only")    # Overwrites everything
f.seek(0)                        # Move cursor back to start
print("After:", f.read())        # Reads what we just wrote
f.close()
```

6. a+ ⇒ To append and read the data from the existing file.
    - It does not override the data, File pointer place at end of file.

```
f = open("demo.txt", "a+")

f.write("\nAppended new line here!")   # Adds at end

f.seek(0)                              # Move to start

print(f.read())                        # Read full file

f.close()
```

7. x ⇒ To create a new file only.
    - "x" stands for exclusive creation.
    - If a file already exists python gives an error.
    - Safe than other modes which create files, if files do not exist. Because it won't override anything.

```
f = open("newfile.txt", "x")

f.write("This file is created with x mode.")

f.close()
```