WHAT IS PYTHON?

Python is a high-level, general-purpose programming language known for its readability and versatility. It's widely used in various fields, including web development, data science, machine learning, and software development. Python is an interpreted language, meaning code is executed line by line. Its simple syntax, similar to English, makes it beginner-friendly and popular among experienced programmers.

Here's a more detailed breakdown:

- **Versatile and General-Purpose:**

Python isn't specialized for a particular task but can be used in many applications.

- **Readability and Beginner-Friendly:**

Its syntax is designed to be easy to understand, even for those new to programming.

- **Interpreted Language:**

Python code is executed directly, line by line, without needing to be compiled first.

- **Object-Oriented:**

Python supports object-oriented programming principles, allowing for modular and reusable code.

- **Large Standard Library:**

Python comes with a vast collection of pre-written code modules for various tasks.

- **Widely Used:**

It's a popular choice for web development, data analysis, machine learning, and software development.

- **Cross-Platform:**

Python runs on various operating systems like Windows, macOS, and Linux

WHAT IS MODULES AND PIP IN PYTHON?

In Python, modules are reusable blocks of code that extend the language's functionality. pip is the package manager for Python, used to install and manage these external modules.

Modules:

- **Reusability:**

Modules are designed to be reused in different parts of a program or even in different programs.

- **Organization:**

They help organize code, making it easier to maintain and understand.

- **Extensibility:**

Modules provide access to a wide range of functionalities without requiring you to write the code yourself.

- **Examples:**

Standard Python modules like math, random, or datetime are pre-installed with the language. External modules can be installed using pip, such as numpy, pandas, or requests.

pip (Package Installer for Python):

- **Installation:** pip is the primary tool for installing Python packages, including external modules.
- **Dependency Management:** It helps manage dependencies between different packages.
- **Virtual Environments:** pip can be used with virtual environments to isolate project dependencies.
- **Command-line usage:** The basic command is pip install <module_name>

COMMENTS IN PYTHON

A **comment** is a line in your code that is **ignored by Python when your program runs**. It's used to **explain code** or leave **notes for yourself or others**.

1. Single-line Comment

Use # at the beginning of the line.

```
# This is a single-line comment

print("Hello, World!")  # This prints a message
```

2. Multi-line Comment

Python doesn't have a true multi-line comment, but you can use triple quotes (`'''` or `"""`) to simulate it.

```
'''
This is a
multi-line comment
'''
print("Welcome to Python")
```

WHAT IS AN ESCAPE SEQUENCE?

Escape sequences start with a **backslash** \, and are used to **represent special characters** in a string that can't be typed directly.

Escape Sequence	**Meaning**	**Example**
\n	New Line	"Hello\nWorld"
\t	Tab Space	"Name:\tAlice"
\\	Backslash	"This is a backslash: \\"
\'	Single Quote	'It\'s a sunny day'
\"	Double Quote	"He said, \"Hi!\""

VARIABLES AND DATA TYPES IN PYTHON

What is a Variable?

A **variable** is a **name** that stores a value in memory.
It acts like a **container** for data.

Rules for Naming Variables:

- Must start with a letter or underscore (_)
- Cannot start with a number
- Can only contain letters, numbers, and underscores
- **Case-sensitive** (name ≠ Name)

Python Data Types

Python has built-in data types to classify different kinds of data.

1. Basic Data Types in Python:

Data Type	**Example**	**Description**
int	10, -5, 1000	Whole numbers
float	3.14, -2.0	Decimal numbers
str	"Hello"	Text - (string of characters)
bool	True, False	Boolean values

2.Collection Data Types:

Data Type	**Example**	**Description**
list	[1, 2, 3]	Ordered, changeable, allows duplicates
tuple	(1, 2, 3)	Ordered, unchangeable, allows duplicates
set	{1, 2, 3}	Unordered, no duplicates
dict	{"name": "Alice", "age": 25}	Key-value pairs

EXERCISE:1

1. **Add two numbers**
2. **Find the square and cube of a number**
3. **Calculate area of a rectangle**
4. **Convert temperature (Celsius ↔ Fahrenheit)**
5. **Simple interest calculator**

TYPECASTING IN PYTHON

Typecasting in Python means converting one data type into another. It's useful when you want to perform operations between different data types or format data in a specific way.

There are two types of typecasting in Python:

---

1. IMPLICIT TYPECASTING

Python automatically converts one data type to another without your involvement.

```
a = 5        # int
b = 2.0      # float

result = a + b
print(result)         # Output: 7.0
print(type(result))   # Output: <class 'float'>
```

2. EXPLICIT TYPECASTING

You manually convert the data type using type functions like:

- `int()` – converts to integer
- `float()` – converts to float
- `str()` – converts to string
- `bool()` – converts to boolean

Examples:

```
# String to int
num_str = "10" #print("Hello" 6,9,0,sep="$",end="JJA")
num_int = int(num_str)
print(num_int)             # Output: 10

# Float to int
f = 7.9
print(int(f))              # Output: 7 (decimal part is removed)

# Int to string
x = 100
print(str(x))              # Output: '100'

# Int to float
y = 5
print(float(y))            # Output: 5.0
```

TAKING INPUT FROM USER

BASIC EXAMPLE:

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

USING TYPECASTING:

```
age = int(input("Enter your age: "))    # Converts input string to integer
print("You are", age, "years old.")
```

EXERCISE

1. **Take your full name as input and print: "Nice to meet you, <full name>!"**
2. **Input three numbers and print their total sum.**
3. **Take two floating-point numbers and print their multiplication result.**
4. **Take your birth year and current year as input, then print your age.**
5. **Input the length of one side of a square and print its area and perimeter.**
   (Area = side × side, Perimeter = 4 × side)

OPERATORS IN PYTHON

1. Arithmetic Operators (Used for mathematical operations)

Operator	**Description**	**Example**
`+`	Addition	`5 + 3 = 8`
`-`	Subtraction	`5 - 2 = 3`
`*`	Multiplication	`4 * 2 = 8`
`/`	Division	`8 / 2 = 4.0`
`//`	Floor Division	`8 // 3 = 2`
`%`	Modulus (remainder)	`7 % 3 = 1`
`**`	Exponent (power)	`2 ** 3 = 8`

---

2. Comparison Operators (Returns True or False)

Operator	**Description**	**Example**
`==`	Equal to	`5 == 5 → True`
`!=`	Not equal to	`5 != 3 → True`
`>`	Greater than	`6 > 2 → True`
`<`	Less than	`2 < 5 → True`
`>=`	Greater than or equal	`5 >= 5 → True`
`<=`	Less than or equal	`4 <= 6 → True`

---

3. Assignment Operators (Used to assign values to variables)

Operator	Description	Example
`=`	Assign	`x = 5`
`+=`	Add and assign	`x += 3 → x = x + 3`
`-=`	Subtract and assign	`x -= 2 → x = x - 2`
`*=`	Multiply and assign	`x *= 4`
`/=`	Divide and assign	`x /= 2`
`//=`	Floor divide and assign	`x //= 3`
`%=`	Modulus and assign	`x %= 2`
`**=`	Exponent and assign	`x **= 2`

---

4. Logical Operators (Used to combine conditional statements)

Operator	Description	Example
`and`	Returns `True` if both are true	`x > 5 and x < 10`
`or`	Returns `True` if at least one is true	`x < 3 or x > 7`
`not`	Reverses the result	`not(x > 5)`

WHAT ARE STRINGS?

In python, anything that you enclose between single or double quotation marks is considered a string. A string is essentially a sequence or array of textual data. Strings are used when working with Unicode characters.

Example:

```
name = "Ayush"

print("Hello, " + name)
```

Output: Hello, Ayush

```
print('He said, "I want to eat an apple".')
```

Multiline Strings:

If our string has multiple lines, we can create them like this:

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,

sed do eiusmod tempor incididunt

ut labore et dolore magna aliqua."""

print(a)
```

Accessing Characters of a String:

In Python, string is like an array of characters. We can access parts of string by using its index which starts from 0.
Square brackets can be used to access elements of the string.

```
print(name[0])

print(name[1])
```

Looping through the string

We can loop through strings using a for loop like this:

```
for character in name:

    print(character)
```

Above code prints all the characters in the string name one by one!

STRING SLICING & OPERATIONS ON STRING

Length of a String:

We can find the length of a string using len() function.

Example:

```
fruit = "Mango"
len1 = len(fruit)
print("Mango is a", len1, "letter word.")
```

Output:

Mango is a 5 letter word.

String as an array

A string is essentially a sequence of characters also called an array. Thus we can access the elements of this array.

Example:

```
pie = "ApplePie"
print(pie[:5])
print(pie[6])        #returns character at specified index
```

Output:

```
Apple
i
```

Note: This method of specifying the start and end index to specify a part of a string is called slicing.

Slicing Example:

```
pie = "ApplePie"
print(pie[:5])      #Slicing from Start
print(pie[5:])      #Slicing till End
print(pie[2:6])     #Slicing in between
print(pie[-8:])     #Slicing using negative index
```

Output:

```
Apple
Pie
pleP
ApplePie
```

Loop through a String:

Strings are arrays and arrays are iterable. Thus we can loop through strings.

Example:

```
alphabets = "ABCDE"
for i in alphabets:
    print(i)
```

Output:

```
A
B
C
D
E
```

STRING METHODS

Python provides a set of built-in methods that we can use to alter and modify the strings.

1. **upper() :**

   The upper() method converts a string to upper case.

   Example:

   ```
   str1 = "AbcDEfghIJ"
   print(str1.upper())
   ```

   Output:

   ```
   ABCDEFGHIJ
   ```

2. **lower()**

   The lower() method converts a string to lower case.

   Example:

   ```
   str1 = "AbcDEfghIJ"
   print(str1.lower())
   ```

   Output:

   ```
   Abcdefghij
   ```

3. **strip() :**

   The strip() method removes any white spaces before and after the string.

   Example:

```
str2 = " Ayush Spoon "
print(str2.strip)
```
Output:

Ayush Spoon

4. rstrip() :

The rstrip() removes any trailing characters in right side of string.

Example:
```
str3 = "Hello !!!"
print(str3.rstrip("!"))
```
Output:

Hello

5. replace() :

The replace() method replaces all occurences of a string with another string. Example:
```
str2 = "Silver Spoon"
print(str2.replace("Sp", "M"))
```
Output:

Silver Moon

6. split() :

The split() method splits the given string at the specified instance and returns the separated strings as list items.

Example:
```
str2 = "Silver Spoon"
print(str2.split(" "))      #Splits the string at the
whitespace " ".
```
Output:

['Silver', 'Spoon']

7. capitalize() :

The capitalize() method turns only the first character of the string to uppercase and the rest other characters of the string are turned to lowercase. The string has no effect if the first character is already uppercase.

Example:
```
str1 = "hello"
capStr1 = str1.capitalize()
print(capStr1)
str2 = "hello WorlD"
capStr2 = str2.capitalize()
print(capStr2)
```
Output:

Hello

Hello world

8. center() :

The center() method aligns the string to the center as per the parameters given by the user.

Example:
```
str1 = "Welcome to the Console!!!"
print(str1.center(50))
```
Output:
```
        Welcome to the Console!!!
```
We can also provide padding character. It will fill the rest of the fill characters provided by the user.

Example:

```
str1 = "Welcome to the Console!!!"
print(str1.center(50, "."))
```
Output:

```
...........Welcome to the Console!!!............
```

9. count() :

The count() method returns the number of times the given value has occurred within the given string.

Example:

```
str2 = "Abracadabra"
countStr = str2.count("a")
print(countStr)
```
Output:

```
4
```

10. endswith() :

The endswith() method checks if the string ends with a given value. If yes then return True, else return False.

Example :

```
str1 = "Welcome to the Console !!!"
print(str1.endswith("!!!"))
```
Output:

```
True
```
We can even also check for a value in-between the string by providing start and end index positions.

Example:

```
str1 = "Welcome to the Console !!!"
print(str1.endswith("to", 4, 10))
```
Output:

```
True
```

11. find() :

The find() method searches for the first occurrence of the given value and returns the index where it is present. If given value is absent from the string then return -1.

Example:

```
str1 = "He's name is Dan. He is an honest man."
print(str1.find("is"))
```
Output:

```
10
```
As we can see, this method is somewhat similar to the index() method. The major difference being that index() raises an exception if value is absent whereas find() does not.

Example:

```
str1 = "He's name is Dan. He is an honest man."
print(str1.find("Daniel"))
```
Output:

```
-1
```

12.index() :

The index() method searches for the first occurrence of the given value and returns the index where it is present. If given value is absent from the string then raise an exception.

Example:

```
str1 = "He's name is Dan. Dan is an honest man."
print(str1.index("Dan"))
```
Output:

13

As we can see, this method is somewhat similar to the find() method. The major difference being that index() raises an exception if value is absent whereas find() does not.

Example:

```
str1 = "He's name is Dan. Dan is an honest man."
print(str1.index("Daniel"))
```

Output:

ValueError: substring not found

13.isalnum() :

The isalnum() method returns True only if the entire string only consists of A-Z, a-z, 0-9. If any other characters or punctuations are present, then it returns False.

Example 1:

```
str1 = "WelcomeToTheConsole"
print(str1.isalnum())
```

Output:

True

14.isalpha() :

The isalnum() method returns True only if the entire string only consists of A-Z, a-z. If any other characters or punctuations or numbers(0-9) are present, then it returns False.

Example :

```
str1 = "Welcome"
print(str1.isalpha())
```

Output:

True

15.islower() :

The islower() method returns True if all the characters in the string are lower case, else it returns False.

Example:

```
str1 = "hello world"
print(str1.islower())
```

Output:

True

16.isprintable() :

The isprintable() method returns True if all the values within the given string are printable, if not, then return False.

Example :

```
str1 = "We wish you a Merry Christmas"
print(str1.isprintable())
```

Output:

True

17.isspace() :

The isspace() method returns True only and only if the string contains white spaces, else returns False.

Example:

```
str1 = "        "       #using Spacebar
print(str1.isspace())
str2 = "        "       #using Tab
print(str2.isspace())
```

Output:

```
True
True
```

18.istitle() :

The istitile() returns True only if the first letter of each word of the string is capitalized, else it returns False.

Example:

```
str1 = "World Health Organization"
print(str1.istitle())
```

Output:

```
True
```

Example:

```
str2 = "To kill a Mocking bird"
print(str2.istitle())
```

Output:

```
False
```

19.isupper() :

The isupper() method returns True if all the characters in the string are upper case, else it returns False.

Example :

```
str1 = "WORLD HEALTH ORGANIZATION"
print(str1.isupper())
```

Output:

```
True
```

20.startswith() :

The endswith() method checks if the string starts with a given value. If yes then return True, else return False.

Example :

```
str1 = "Python is a Interpreted Language"
print(str1.startswith("Python"))
```

Output:

```
True
```

21.swapcase() :

The swapcase() method changes the character casing of the string. Upper case are converted to lower case and lower case to upper case.

Example:

```
str1 = "Python is a Interpreted Language"
print(str1.swapcase())
```

Output:

```
pYTHON IS A iNTERPRETED lANGUAGE
```

22.title() :

The title() method capitalizes each letter of the word within the string.

Example:

```
str1 = "He's name is Dan. Dan is an honest man."
print(str1.title())
```

Output:

```
He'S Name Is Dan. Dan Is An Honest Man.
```

IF-ELSE STATEMENTS

if-else is used in Python to make decisions in your program.

It helps your program take different actions based on conditions (True/False).
Based on this, the conditional statements are further classified into following types:

- ❖ if
- ❖ if-else
- ❖ if-else-elif
- ❖ nested if-else-elif.

Structure:
```
if condition:
    # code to run if condition is True
else:
    # code to run if condition is False
```

Example1:
```
age = 21
age_normal = 18
if (age> age_normal):
    print("You can drive car.")
else:
    print("You can not drive car.")
```

Output:
You can not drive car.

Example2:
```
marks = 75

if marks >= 90:
    print("Grade: A")
elif marks >= 75:
    print("Grade: B")
elif marks >= 60:
    print("Grade: C")
else:
    print("Fail")
```
Output:
```
Grade: B
```

MATCH CASE STATEMENT

match-case is used to check **multiple conditions** (like switch-case in C, C++, Java).
It was introduced in **Python 3.10** and later versions

Important Points:

- Use `match-case` only in **Python 3.10 or newer**.
- `case _`**:** works like the **default** case.
- You can use `|` to check **multiple values** in a single case.
- Use `case _ if condition:` for **conditional matching**.

Example1:

```
fruit = "apple"

match fruit:
    case "apple":
        print("It's red or green.")
    case "banana":
        print("It's yellow.")
    case _:
        print("Unknown fruit.")
```

Example2:

```
value = 2

match value:
    case 1 | 2 | 3:
        print("Between 1 and 3")
    case 4:
        print("It's four")
    case _:
        print("Something else")
```

Example3:

```
number = 10

match number:
    case _ if number % 2 == 0:
        print("Even number")
    case _:
        print("Odd number")
```

INTRODUCTION TO LOOPS

Sometimes a programmer wants to execute a group of statements a certain number of times. This can be done using loops. Based on this loops are further classified into following main types;

- for loop
- while loop

The for Loop

for loops can iterate over a sequence of iterable objects in python. Iterating over a sequence is nothing but iterating over strings, lists, tuples, sets and dictionaries.

Example1: Iterating over a string:

```
name = 'Abhishek'
for i in name:
    print(i, end=", ")
```

Output:

```
A, b, h, i, s, h, e, k,
```

Notes: Similarly, we can use loops for lists, sets and dictionaries.

Example2: Iterating over a list:

```
colors = ["Red", "Green","Blue","Yellow"]
for x in colors:
    print(x)
```

Output:

```
Red
Green
Blue
Yellow
```

range():

What if we do not want to iterate over a sequence? What if we want to use for loop for a specific number of times?
Here, we can use the range() function.

Example:

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

Output:

```
0
1
2
3
4
```

Example3:

```
for k in range(4,9):
    print(k)
```

Output:

```
4
5
6
7
8
```