**Module 2 : Beginning Python Basics**

**The print statement**

Open vs code and go to file and click on New text file and save as

Write 1st program of python Hello World

Print(“Hello world”)

* print is a built-in function.
* Anything inside the parentheses (in quotes) is displayed.
* You can print **strings**, **numbers**, **variables**, and even **expressions**.

1. Printing a String

print("Welcome to Python!")

2. Printing Numbers

print(2025)

3. Printing Multiple Items

print("The answer is", 42)

4. Printing with Expressions

print("2 + 3 =", 2 + 3)

5. X=10

Y=20

Sum=x+y

Print(sum)

**Comments**

Comments are lines in your code that are not executed. They're used to explain code, make notes, or temporarily disable parts of code.

Types of Comments in Python

**1.Single-line Comments**

Use the # symbol.

# This is a single-line comment

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

**2. Multi-line Comments (Block Comments)**

Python doesn't have a true multi-line comment syntax, but you can use multiple # lines or triple quotes (''' or """) as a workaround.

# This is a multi-line

# comment that explains

# something important

"""

This is also considered

a multi-line comment by many,

but it's actually a multi-line string

that’s not assigned to a variable.

"""

**Python Data Structures & Data Types**

**Python has several built-in data types which are categorized as follows:**

**1. Numeric Types**

int – Integer (e.g., 10, -5)

float – Floating point number (e.g., 3.14, -0.001)

complex – Complex number (e.g., 2 + 3j)

**2. Text Type**

str – String (e.g., "hello", 'Python')

**3. Sequence Types**

list – Ordered, mutable collection (e.g., [1, 2, 3])

tuple – Ordered, immutable collection (e.g., (1, 2, 3)), Cannot be changed after creation

range – Immutable sequence of numbers (e.g., range(1, 5))

**4. Set Types**

set – Unordered, mutable, unique elements (e.g., {1, 2, 3})

frozenset – Immutable version of a set

**5. Mapping Type**

generally refers to applying a function to each item in an iterable (like a list) and getting a new iterable (typically with transformed data).

dict – Key-value pairs (e.g., {'a': 1, 'b': 2})

**6. Boolean Type**

bool – True or False

**7. None Type**

NoneType – Represents absence of value (None)

**Python Data Structures**

Data structures are ways of organizing and storing data efficiently.

**1. List**

Ordered, mutable, allows duplicates.

Defined with []

Supports indexing, slicing, and various methods (append(), remove(), etc.)

fruits = ['apple', 'banana', 'cherry']

**2. Tuple**

Ordered, immutable.

Faster than lists for iteration.

coordinates = (10.0, 20.0)

**3. Set**

Unordered, mutable, no duplicates.

Useful for membership testing and eliminating duplicates.

unique\_numbers = {1, 2, 3}

**4. Dictionary:-**A dictionary in Python is a collection of key-value pairs. It lets you store and retrieve data by a key.Efficient for lookups.

Example

student = {

"name": "Rahul",

"age": 20,

"grade": "A"

}

print(student["name"])

print(student["age"])

Adding a new item

student["city"] = "Delhi"

print(student)

Changing a value

student["grade"] = "A+"

print(student)

Deleting a key-value pair

del student["age"]

print(student)

Looping through a dictionary

for key, value in student.items():

print(key, ":", value)

1. **Numeric Types**

# Integer

a = 10

print(type(a)) # <class 'int'>

# Float

b = 3.14

print(type(b)) # <class 'float'>

# Complex

c = 2 + 3j

print(type(c)) # <class 'complex'>

1. **Text Type**

text = "Hello, Python!"

print(type(text)) # <class 'str'>

print(text.upper()) # HELLO, PYTHON!

1. **Sequence Types**

List (Mutable, Ordered)

fruits = ['apple', 'banana', 'cherry']

fruits.append('mango')

print(fruits) # ['apple', 'banana', 'cherry', 'mango']

Tuple (Immutable, Ordered)

coordinates = (10.5, 20.5)

print(coordinates[0]) # 10.5

Range (Immutable, Sequence of Numbers)

for i in range(3):

print(i) # 0, 1, 2

1. **Set Types**

# Creating a set

fruits = {"apple", "banana", "cherry"}

print(fruits) # Output might be in any order

# Adding an item

fruits.add("orange")

print(fruits)

# Trying to add a duplicate (won't change the set)

fruits.add("apple")

print(fruits)

# Removing an item

fruits.remove("banana")

print(fruits)

# Checking membership

print("apple" in fruits) # True

print("banana" in fruits) # False

# Set operations

set1 = {1, 2, 3}

set2 = {3, 4, 5}

print(set1.union(set2)) # {1, 2, 3, 4, 5}

print(set1.intersection(set2)) # {3}

creating a set from list

my\_list = [1, 2, 2, 3, 4, 4, 5]

my\_set = set(my\_list)

print(my\_set) # {1, 2, 3, 4, 5}

**5. Mapping Type**

map(function, iterable)

Example-1

numbers = [1, 2, 3, 4, 5]

# Square each number

def square(x):

return x \* x

squared = map(square, numbers)

print(list(squared)) # Output: [1, 4, 9, 16, 25]

Example-2

a = [1, 2, 3]

b = [4, 5, 6]

# Add elements from both lists

result = map(lambda x, y: x + y, a, b)

print(list(result)) # Output: [5, 7, 9]

Example-3

numbers = [1, 2, 3, 4, 5]

squared = [x\*\*2 for x in numbers]

print(squared) # Output: [1, 4, 9, 16, 25]

1. **Boolean data type**

In Python, the boolean data type represents one of two values:

* True
* False

Example-1

# Boolean variables

is\_active = True

# Using booleans in a condition

if is\_active:

print("User is active.")

else:

print("User is not active.")

Example-2

x = 10

y = 5

print(x > y) # True

print(x == y) # False

print(x < 20) # True

**7. None Type**

* None is not the same as 0, False, or an empty string "".
* It is a singleton (only one instance exists).

Example 1: Assigning None to a variable

x = None

print(x) # Output: None

print(type(x)) # Output: <class 'NoneType'>

Example 2: Using None in conditions

x = None

if x is None:

print("x has no value") # Output: x has no value

**String Operations in Python**

Common string operations include concatenation, slicing, repetition, searching, replacing.

1. Concatenation (+)

Joining two or more strings.

str1 = "Hello"

str2 = "World"

result = str1 + " " + str2

print(result) # Output: Hello World

1. Repetition (\*)

str1 = "Hi! "

print(str1 \* 3) # Output: Hi! Hi! Hi!

1. Slicing

Extracting a portion of a string using indices.

str1 = "PythonProgramming"

print(str1[0:6]) # Output: Python

print(str1[-6:]) # Output: gramming

1. Length (len())

str1 = "Hello"

print(len(str1)) # Output: 5

1. **Membership (in, not in)**

Checking if a substring exists in a string.

print("Python" in "Python Programming") # Output: True

print("Java" not in "Python Programming") # Output: True

1. String Methods

s = " hello world "

print(s.strip()) # Remove leading/trailing whitespace -> "hello world"

print(s.upper()) # " HELLO WORLD "

print(s.lower()) # " hello world "

print(s.replace("world", "Python")) # " hello Python "

**Simple Input & Output**

In Python, **input** is taken using the input() function, and **output** is displayed using the print() function.

Example: Simple Input and Output

# Taking input from the user

name = input("Enter your name: ")

age = input("Enter your age: ")

# Displaying the output

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

print("You are", age, "years old.")

**Note:** By default, input() returns data as a **string**. If you need numeric input, use int() or float():

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

print("Double the number is:", num \* 2)

**Simple Output Formatting**

Simple output formatting in Python can be done using:

1. **Using f-strings**

name = "Amit"

age = 30

print(f"My name is {name} and I am {age} years old.")

1. **Using format() method**

name = "Amit"

age = 30

print("My name is {} and I am {} years old.".format(name, age))

1. **Using % operator**

name = "Amit"

age = 30

print("My name is %s and I am %d years old." % (name, age))

**Operators in python**

**operators** are special symbols or keywords used to perform operations on variables and values.

1. **Arithmetic Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| **+** | **Addition** | **a + b (e.g., 5 + 3 = 8)** |
| **-** | **Subtraction** | **a - b (e.g., 5 - 3 = 2)** |
| **\*** | **Multiplication** | **a \* b (e.g., 5 \* 3 = 15)** |
| **/** | **Division** | **a / b (e.g., 5 / 2 = 2.5)** |
| **//** | **Floor Division** | **5 // 2 = 2** |
| **%** | **Modulus (remainder)** | **5 % 2 = 1** |
| **\*\*** | **Exponentiation** | **2 \*\* 3 = 8** |

a = 10

b = 3

print(a + b) # 13

print(a \*\* b) # 1000

1. **Comparison Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| **==** | **Equal to** | **a == b** |
| **!=** | **Not equal to** | **a != b** |
| **>** | **Greater than** | **a > b** |
| **<** | **Less than** | **a < b** |
| **>=** | **Greater than or equal** | **a >= b** |
| **<=** | **Less than or equal** | **a <= b** |

print(10 > 5) # True

print(10 == 3) # False

1. **Logical Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| **and** | **True if both are true** | **a > 5 and b < 10** |
| **or** | **True if at least one is true** | **a > 5 or b < 2** |
| **not** | **Reverse the result** | **not(a > 5)** |

x = 10

print(x > 5 and x < 15) # True

print(not x == 10) # False

1. **Assignment Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| **=** | **a = 5** | **-** |
| **+=** | **a += 2** | **a = a + 2** |
| **-=** | **a -= 2** | **a = a - 2** |
| **\*=** | **a \*= 2** | **a = a \* 2** |
| **/=** | **a /= 2** | **a = a / 2** |

a = 5

a += 3 # a = a + 3 => 8

print(a)