Manual - Python Basics

Objective

The objective of this session is to provide a comprehensive introduction to the fundamentals of Python programming. By the end of this session, you will have a solid understanding of Python's history, features, and advantages, as well as how to work with expressions, operators, data types, and basic data structures.

Table of Contents

1. Introduction to Python

History of Python

Python was created by Guido van Rossum and first released in 1991. It was designed with an emphasis on code readability and simplicity, making it accessible for beginners while being powerful enough for advanced tasks.

Features of Python

- **Easy to Read, Learn, and Write:** Python's syntax is designed to be intuitive and its code easy to read, which makes it a great choice for beginners.
- **Interpreted Language:** Python is executed line-by-line, which makes debugging easier.
- **Dynamically Typed:** Variables in Python do not need explicit declaration to reserve memory space. The declaration happens automatically when a value is assigned to a variable.
- **Vast Standard Library:** Python's standard library supports many common programming tasks such as connecting to web servers, searching text with regular expressions, and reading and modifying files.
- **Extensible:** Python can be extended with modules written in C or C++.
- **Large Community and Ecosystem:** Python has a large, active community and a vast ecosystem of libraries and frameworks.

Advantages of Python

- **Productivity and Speed:** Python enhances productivity by offering simplicity in coding, which allows developers to write code faster.
- **Open Source:** Python is free to use and distribute, including for commercial purposes.
- **Cross-Platform Compatibility:** Python can run on various operating systems such as Windows, MacOS, and Linux.
- **Third-Party Modules:** The Python Package Index (PyPI) hosts thousands of third-party modules, making Python highly versatile.

Variables and Data Types

- **Variables**: Containers for storing data values. Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when a value is assigned to a variable.
    - Example:

```
x = 5
y = "Hello, World!"
```

- **Data Types**: Python has various standard data types including:
    - **Numeric Types**: int, float, complex
        - Example:

```
int_var = 10
float_var = 20.5
complex_var = 1 + 2j
```

    - **Sequence Types**: list, tuple, range
        - Example:

```
list_var = [1, 2, 3, 4]
tuple_var = (1, 2, 3, 4)
range_var = range(5)  # generates numbers from 0 to 4
```

    - **Mapping Type**: dict
        - Example:

```
dict_var = {"key1": "value1", "key2": "value2"}
```

    - **Set Types**: set, frozenset
        - Example:

```
set_var = {1, 2, 3, 4}
frozenset_var = frozenset([1, 2, 3, 4])
```

- **Boolean Type**: bool
  - Example: `bool_var = True`
- **Text Type**: str
  - Example:`string_var = "Hello, Python!"`

Data Types	Class	Description
Numeric	int, float	Holds numeric values
String	str	Holds sequence of characters
Sequence	list, tuple	Holds collection of items
Mapping	dict	Holds data in key-value form
Boolean	bool	Holds either True or False
Set	set	Hold collection of unique items

2. Expressions and Operators

• Working with Arithmetic Operators

Arithmetic operators are used to perform mathematical operations between numeric values. These operators are used to perform mathematical operations:

- + : Addition
- - : Subtraction
- * : Multiplication
- / : Division
- // : Floor Division
- % : Modulus
- ** : Exponentiation

Addition (+)

- **Usage**: Adds two numbers.
- **Example**:

```
result = 10 + 5  # result is 15
```

Subtraction (-)

- **Usage**: Subtracts the right operand from the left operand.
- **Example**:`result = 10 - 5  # result is 5`

Multiplication (*)

- **Usage**: Multiplies two numbers.
- **Example**:

```
result = 10 * 5  # result is 50
```

Division (/ and //)

- **Usage**: Divides the left operand by the right operand.
  - `/` performs floating-point division.
  - `//` performs integer (floor) division.
- **Example**:

```
result = 10 / 3  # result is 3.3333...
result = 10 // 3  # result is 3
```

Modulus (%)

- **Usage**: Returns the remainder of the division.
- **Example**:`result = 10 % 3  # result is 1`

Exponentiation (**)

- **Usage**: Raises the left operand to the power of the right operand.
- **Example**:`result = 2 ** 3  # result is 8`

• Assignment Operators

These operators are used to assign values to variables:

- = : Assign
- += : Add and assign
- -= : Subtract and assign
- *= : Multiply and assign
- /= : Divide and assign
- //= : Floor divide and assign
- %= : Modulus and assign
- **= : Exponentiate and assign

Examples:

```
x = 5
x += 3  # x is now 8
x -= 2  # x is now 6
x *= 4  # x is now 24
x /= 3  # x is now 8.0
x //= 2 # x is now 4.0
x %= 3  # x is now 1.0
x **= 3 # x is now 1.0
```

- ## Comparison Operators

Comparison operators compare the values on either side of the operator and return a boolean result (`True` or `False`).

These operators are used to compare two values:

- == : Equal to
- != : Not equal to
- > : Greater than
- < : Less than
- >= : Greater than or equal to
- <= : Less than or equal to

Equal to (==)

- **Usage**: Checks if two values are equal.
- **Example**:

```
result = (10 == 10)  # result is True
```

Not equal to (!=)

- **Usage**: Checks if two values are not equal.
- **Example**:`result = (10 != 5)  # result is True`

Greater than (>)

- **Usage**: Checks if the left operand is greater than the right operand.
- **Example**:`result = (10 > 5)  # result is True`

Less than (<)

- **Usage**: Checks if the left operand is less than the right operand.
- **Example**:`result = (5 < 10)  # result is True`

Greater than or equal to (>=)

- **Usage**: Checks if the left operand is greater than or equal to the right operand.
- **Example**:`result = (10 >= 5)  # result is True`

Less than or equal to (<=)

- **Usage**: Checks if the left operand is less than or equal to the right operand.
- **Example**:

```
result = (5 <= 10)  # result is True
```

• Logical Operators

Logical operators are used to combine conditional statements. These operators are used to combine conditional statements:

- and : Returns True if both statements are true
- or : Returns True if one of the statements is true
- not : Reverses the result, returns False if the result is true

and

- **Usage**: Returns `True` if both statements are `True`.
- **Example**:

```
result = (10 > 5) and (5 < 10)  # result is True
```

or

- **Usage**: Returns `True` if at least one of the statements is `True`.
- **Example**:

```
result = (10 > 5) or (5 > 10)  # result is True
```

not

- **Usage**: Reverses the result of the condition.
- **Example**:

```
result = not (10 > 5)  # result is False
```

---

3. Understanding the type() Function and Type Inference

The type () Function

The type () function is used to determine the type of a variable or value.

Examples:

```
print (type (5))          # <class 'int'>
print (type (5.0)         # <class 'float'>
print(type("Hello"))      # <class 'str'>
print (type ([1, 2, 3]))  # <class 'list'>
print (type ((1, 2, 3)))  # <class 'tuple'>
print (type ({1: 'a', 2: 'b'})) # <class 'dict'> → Dict means dictionary
```

Type Inference

Python automatically infers the type of variable based on the value assigned to it. You don't need to explicitly declare the type.

Examples:

```
x = 5      # x is inferred as an int
y = 3.14   # y is inferred as a float
z = "hi"   # z is inferred as a str
```