**Day 1: Introduction to Python Programming**

**What is Python: Overview and Features**

Python is a versatile and powerful high-level programming language designed to be easy to read and simple to implement. Its design philosophy emphasizes code readability with the use of significant indentation, which helps developers write clean and logical code.

**Key Features:**

• **Readable Syntax:** Python’s syntax is designed to be clean and easy to understand. For example, indentation is used to define code blocks instead of braces.

• **Interpreted Language**: Python code is executed line-by-line, which simplifies debugging and testing.

• **Dynamically Typed**: Variables in Python are not bound to a specific type. This means you can reassign variables to different data types without declaring their type.

• **Object-Oriented:** Python supports object-oriented programming (OOP), allowing for encapsulation, inheritance, and polymorphism.

• **Extensive Standard Library:** Python includes a rich standard library that provides modules and functions to handle various tasks like file I/O, system calls, and more.

• **Cross-Platform Compatibility:** Python runs on various operating systems including Windows, macOS, and Linux.

**Identifiers and Keywords**

**Identifiers** are names given to various program elements such as variables, functions, classes, and objects. Identifiers must follow these rules:

• Start with a letter (A-Z or a-z) or an underscore (\_).

• Can be followed by letters, digits (0-9), or underscores.

• Are case-sensitive, meaning variable and Variable are different identifiers.

**Keywords** are reserved words that have special meanings in Python and cannot be used as identifiers. They include:

• False, None, True – Boolean values and special constants

• and, or, not – Logical operators

• if, else, elif – Conditional statements

• for, while, break, continue – Loop control

• def, class – Function and class definitions

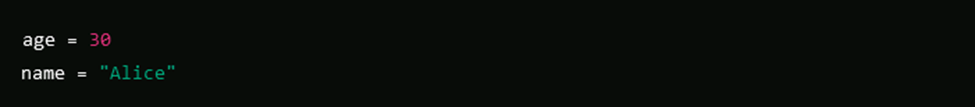
**Statements and Expressions**

• Statement: A statement is an instruction that the interpreter can execute. For example, x = 5 is an assignment statement.

• Expression: An expression is a combination of variables, constants, operators, and functions that computes a value. For example, 5 + 3 is an expression that evaluates to 8.

**Variables and Operators**

Variables are containers for storing data values. They are created by assigning a value to a name:



**Operators** perform operations on variables and values. Types of operators include:

* **Arithmetic Operators**: Perform mathematical operations.
* `+` (addition), `-` (subtraction), `\*` (multiplication), `/` (division), `%` (modulus), `` (exponentiation), `//` (floor division)
* The modulo operation (abbreviated “mod”, or “%” in many programming languages) is the remainder when dividing. For example, “5 mod 3 = 2” which means 2 is the remainder when you divide 5 by 3.
* Floor division in Python is a mathematical operation, also known as integer division. The floor division in Python is performed by using the // operator. It divides the first number by another and rounds down the result to the nearest whole number and makes it equal to the math. floor() function.

For example, if we divide 10 by 3 using floor division, we get:10 // 3 = 3. Here, the quotient is 3.33, but since we are using floor division, the result is rounded down to the nearest integer, 3.

* **Comparison Operators**: Compare two values and return a Boolean result.

`==` (equal to), `!=` (not equal to), `>` (greater than), `<` (less than), `>=` (greater than or equal to), `<=` (less than or equal to)

* **Logical Operators:** Combine Boolean values.

`and`, `or`, `not`

* **Assignment Operators**: Assign values to variables.

`=`, `+=`, `-=`, `\*=`, `/=`

**Precedence and Associativity:**

* **Precedence**: Determines the order in which operations are performed. For example, multiplication and division have higher precedence than addition and subtraction.
* **Associativity**: Determines the order of operations for operators of the same precedence. For most operators, the associativity is left-to-right. For exponentiation, it's right-to-left.

Precedence and associativity are fundamental concepts that determine how expressions are evaluated in Python.

**Precedence**

Operator precedence defines the order in which different operations are performed in an expression. Operators with higher precedence are evaluated before operators with lower precedence.

Here’s a brief overview of operator precedence from highest to lowest:

1. Parentheses: `()`

2. Exponentiation: `\*\*`

3. Unary + and -: `+`, `-`

4. Multiplication, Division, Floor Division, Modulus: `\*`, `/`, `//`, `%`

5. Addition and Subtraction: `+`, `-`

6. Bitwise Shift Operators: `<<`, `>>`

7. Bitwise AND: `&`

8. Bitwise XOR: `^`

9. Bitwise OR: `|`

10. Comparison Operators: `==`, `!=`, `<`, `>`, `<=`, `>=`

11. Logical NOT: `not`

12. Logical AND: `and`

13. Logical OR: `or`

**Associativity**

Associativity determines the direction in which operators of the same precedence level are evaluated. In Python:

- Left Associative: Most operators (like `+`, `-`, `\*`, `/`) are left associative, meaning they are evaluated from left to right.

- Example: `2 - 3 + 5` is evaluated as `(2 - 3) + 5`.

- Right Associative: Some operators (like exponentiation ``) are right associative, meaning they are evaluated from right to left.

- Example: `2 3 2` is evaluated as `2 (3 2)`.

**Example**

Consider the expression:



**Step-by-step evaluation:**

1. Evaluate `4 2` (exponentiation) → `16`

2. Evaluate `3 \* 16` (multiplication) → `48`

3. Evaluate `(1 + 2)` (parentheses) → `3`

4. Evaluate `2 + 48` (addition) → `50`

5. Evaluate `50 - 3` (subtraction) → `47`

So, `result` will be `47`.

Understanding precedence and associativity helps you predict how complex expressions will be evaluated and can help prevent logical errors in your code! If you have any further questions, feel free to ask!

**Data Types**

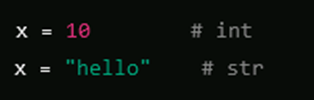
Python is a dynamically and strongly typed language. The main data types in Python are:

* Numeric Types: Used for numerical values.
* `int`: Integer values (e.g., `10`, `-5`)
* `float`: Floating-point values (e.g., `3.14`, `2.718`)
* `complex`: Complex numbers (e.g., `1+2j`)

* Sequence Types: Ordered collections of items.
* `str`: Strings, which are sequences of characters (e.g., `"hello"`, `'world'`)
* `list`: Mutable sequences (e.g., `[1, 2, 3]`)
* `tuple`: Immutable sequences (e.g., `(1, 2, 3)`)
* Mapping Type: Used for key-value pairs.
* `dict`: Dictionary (e.g., `{"name": "Alice", "age": 30}`)
* Set Types: Unordered collections of unique elements.
* `set`: A set of elements (e.g., `{1, 2, 3}`)
* `frozenset`: An immutable set (e.g., `frozenset([1, 2, 3])`)

* Boolean Type: Represents truth values.
* `bool`: Boolean values (`True` or `False`)

**Dynamic Typing**: Variables are not bound to a specific type and can change type as needed:



Strong Typing: Python does not automatically convert between incompatible types. For instance, attempting to add a string to an integer will raise an error:

