Primitive Data Types in Python

Why Do We Need Data Types in Python?

1. Efficient Memory Usage: Python assigns memory based on the type and size of the value.

2. Type-Specific Operations: Each data type supports specific operations (e.g., you can add numbers but not mix strings and integers directly).

3. Error Prevention: Type declarations help catch bugs and invalid operations early.

4. Code Clarity: Knowing variable types makes the code easier to read and maintain.

5. Performance Optimization: Python internally optimizes execution depending on the type of data.

1. int (Integer)

Stores whole numbers without a decimal point.

Example: a = 10

Operations: +, -, *, //, %, **, comparison (==, >, <)

Size: Dynamic in Python (unlimited precision), depends on value

2. float (Floating Point)

Stores numbers with a decimal point.

Example: b = 3.14

Operations: +, -, *, /, //, %, round(), comparison

Size: Typically 24 bytes (platform dependent)

3. str (String)

Stores a sequence of characters enclosed in quotes.

Example: name = 'Alice'

Operations: + (concatenation), * (repetition), slicing, indexing, len(), in, not in

Size: Depends on length; each character ~1 byte (ASCII) or more (Unicode)

Primitive Data Types in Python

4. bool (Boolean)

Stores either True or False.

Example: flag = True

Operations: and, or, not, comparison operators

Size: 28 bytes in CPython implementation

5. complex (Complex Number)

Stores a complex number in the form a + bj.

Example: z = 2 + 3j

Operations: +, -, *, /, abs(), real, imag

Size: Usually 32 bytes