Module-3

Python Introduction

1. Keywords
2. Identifiers
3. Variables, Constants, Literals
4. Datatypes & Type conversions
5. Operators
6. Comments

Keywords

What you will Learn ?

a = 100

int a = 100;

Keywords

Keywords are predefined, reserved words used in Python programming.

We cannot use a keyword as a variable name, function name, or any other identifier.

List of Keywords

- None
- break
- except
- in
- raise
- False
- await
- else
- import
- pass
- and
- continue
- for
- lambda
- try
- True
- class
- finally
- is
- return
- as
- def
- from
- nonlocal
- while
- async
- elif
- if
- not
- with
- assert
- del
- global
- or
- yield

Identifiers

Identifiers

Identifiers are the name given to variables, classes, methods, etc.

Variable Identifiers

python

```python
age = 25
name = "John"
count = 0
total_price = 100.50
is_student = True
```

Method/Function Identifiers

python

```python
def calculate_area(length, width):
    return length * width

def print_greeting(name):
    print("Hello, " + name + "!")
```

Class Identifiers

python

```python
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age

class Calculator:
    def add(self, a, b):
        return a + b
```

Rules

- Start with a Letter or Underscore
- Followed by Letters, Digits, or Underscores
- Avoid Starting with a Digit
- No Special Characters(!, @, $, etc.)
- Case-Sensitive (e.g: age, Age, and AGE)
- Avoid Using Keywords

Valid & Invalid Identifiers

- name
- age
- _count
- total_price
- is_student
- calculate_area
- student_info
- MAX_VALUE

- 3d_model (starts with a digit)
- @result (contains special character)
- for (a Python keyword)
- my-name (contains a hyphen, not allowed)

Variables

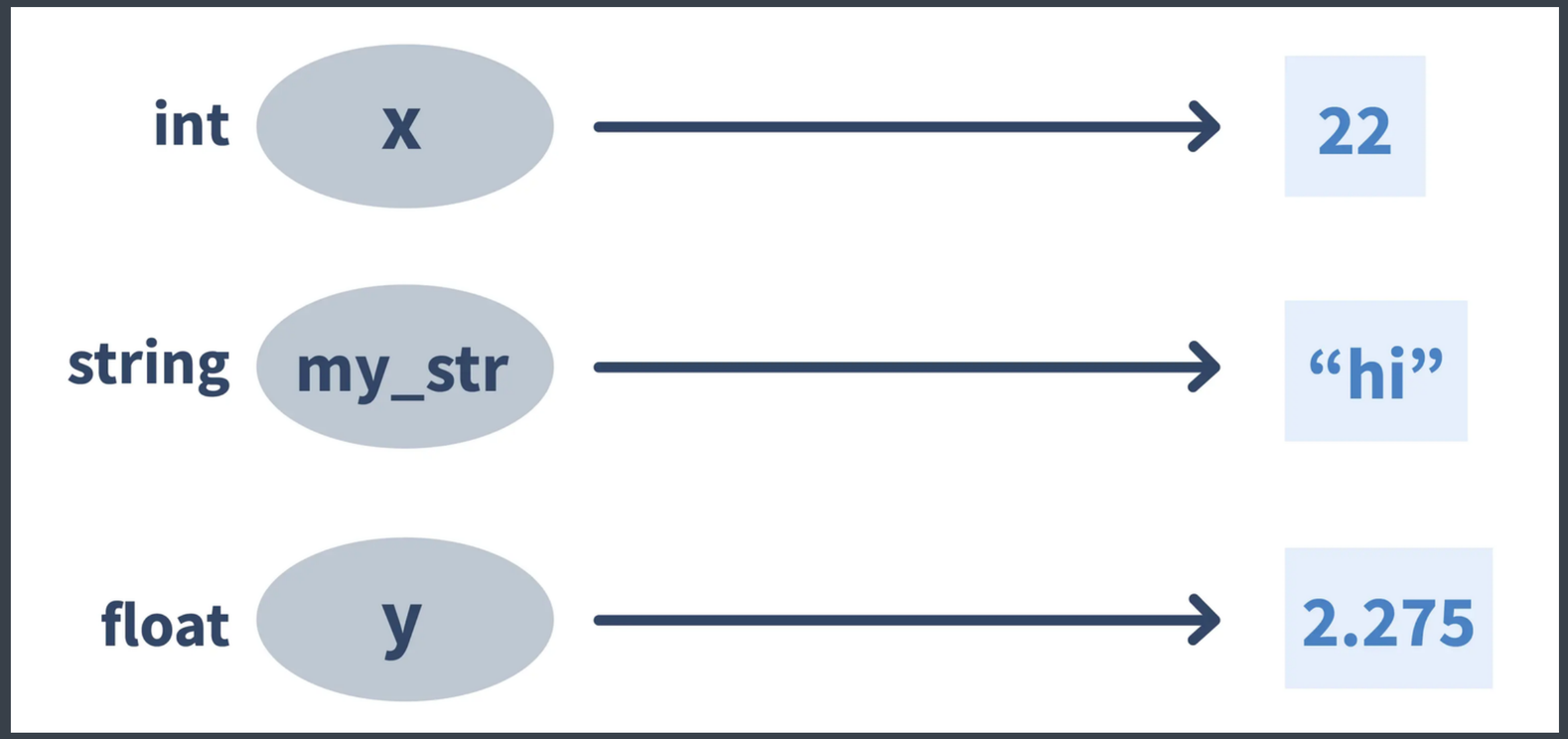
Variables

In programming, a variable is a container (storage area) to hold data.

python

```python
age = 25
name = "John"
count = 0
total_price = 100.50
is_student = True
```

Variables



Rules

- Start with a Letter or Underscore
- Followed by Letters, Digits, or Underscores
- Avoid Starting with a Digit
- No Special Characters(!, @, $, etc.)
- Case-Sensitive (e.g: age, Age, and AGE)
- Avoid Using Keywords

Valid & Invalid Variables

- name
- age
- _count
- total_price
- is_student
- calculate_area
- student_info
- MAX_VALUE

- 3d_model (starts with a digit)
- @result (contains special character)
- for (a Python keyword)
- my-name (contains a hyphen, not allowed)

Variables vs Identifiers

Identifiers are names given to entities like variables, functions, etc., while variables are a specific type of identifier used to store and manipulate data in a program

Constants

Constants

A constant is a special type of variable whose value cannot be changed.

```
# import constant file we created above
import constant

print(constant.PI) # prints 3.14
print(constant.GRAVITY) # prints 9.8
```

Literals

Literals

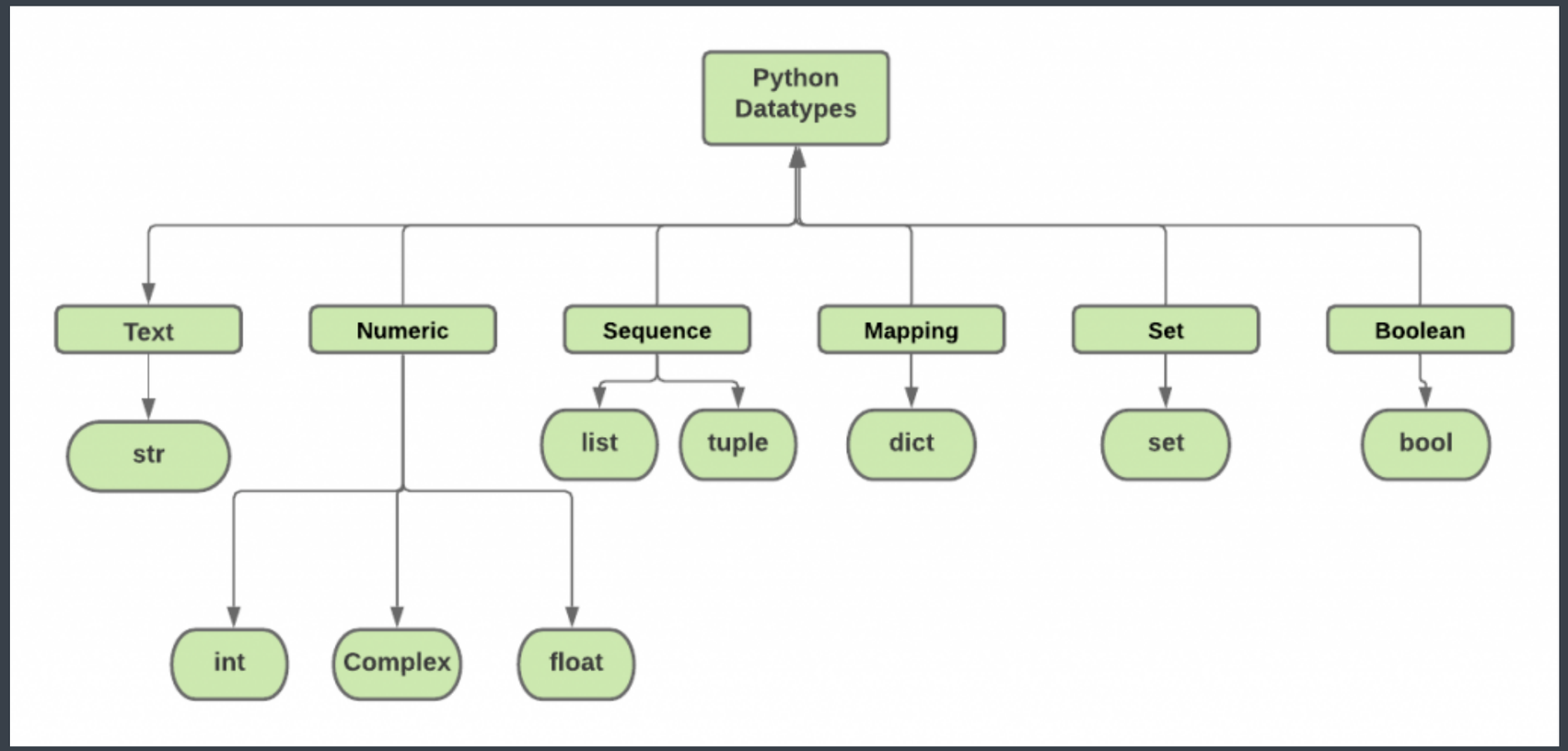
Literals are representations of fixed values in a program. They can be numbers, characters, or strings, etc.

Types of Literals

1. Numeric Literals
2. String Literals
3. Boolean Literals
4. None Literal
5. Bytes and Bytearray Literals
6. Raw String Literals
7. Numeric Separator
8. Collection Literals:
   - List, Tuple, Set, Dictionary

Data Types

What you will Learn ?



Text Data Type

String

str represents strings of characters, enclosed in single, double, or triple quotes.e.g:

'hello'
"world"
'''hello world'''

Numeric Data Types

int

Represents integer values
(e.g., 1, -10, 100)

float

Represents floating point or decimal values
(e.g., 3.14, -2.5)

complex

complex values (e.g., 3.14j, 1-2i)

Sequence Data Types

Tuple

- ordered
- immutable
- separated by commas
- enclosed in parentheses
- (e.g., (1, 2, 3)).

List

- ordered
- mutable
- separated by commas
- enclosed in square brackets
- (e.g., [1, 2, 3])

Mapping Data Types

dict

Represents unordered collections of key-value pairs, enclosed in curly braces (e.g., {'name': 'John', 'age': 30})..

Set Data Types

set

Represents unordered collections of unique elements, enclosed in curly braces (e.g., {1, 2, 3})..

Boolean Data Types

bool

bool represents boolean values, which can be either True or False. Often used for logical operations and control flow.

"True"
"False"

None Data Types

None

Represents a special object that indicates the absence of a value or a "null" value.

Type conversions

Type Conversion

In programming, type conversion is the process of converting data of one type to another.
For example: converting int data to str.

Type Conversion

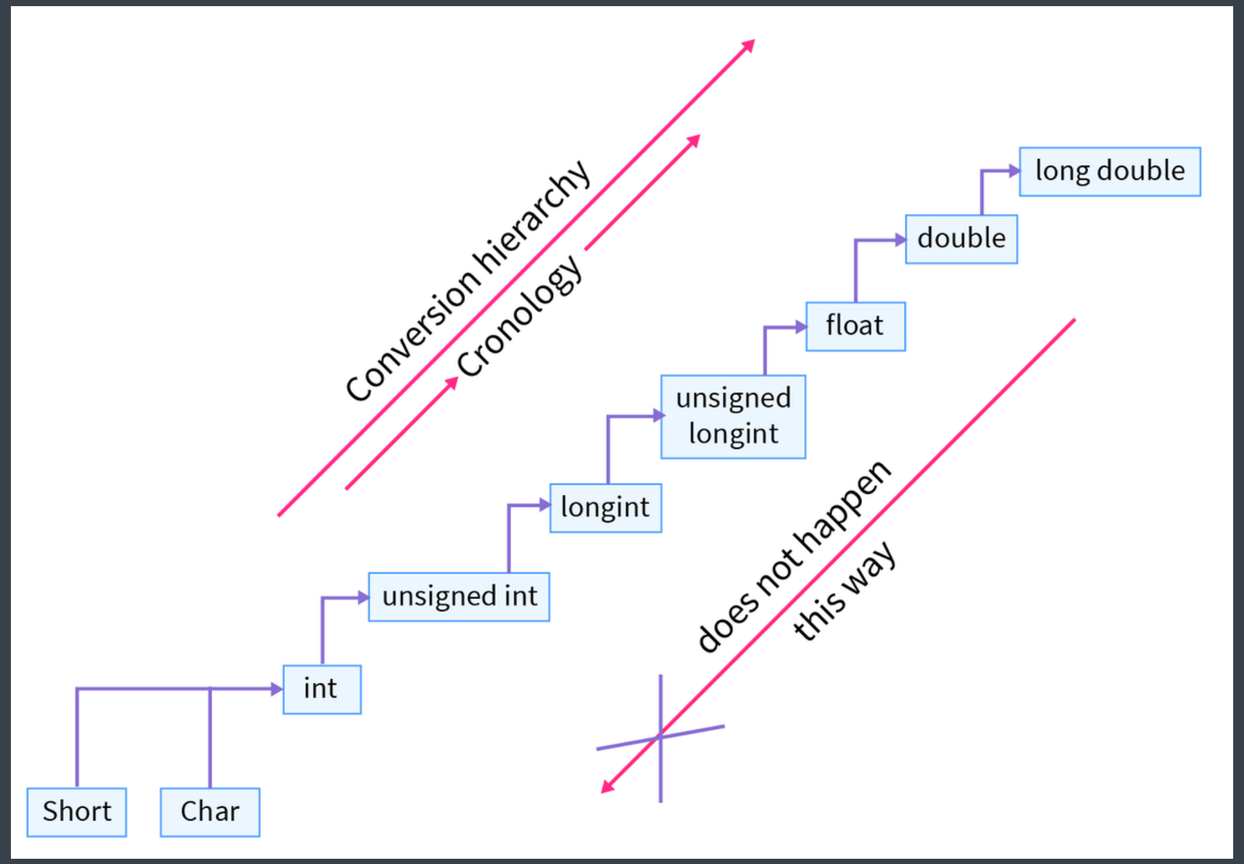
Implicit Type Conversion

Python automatically converts one data type to another.

Explicit Type Conversion

Users convert the data type of an object to required data type.

Conversion Hierarchy



Built-in type conversion functions

- int(): Converts to an integer.
- float(): Converts to a floating-point number.
- str(): Converts to a string.
- bool(): Converts to a boolean.
- list(): Converts to a list.
- tuple(): Converts to a tuple.
- set(): Converts to a set.
- dict(): Converts to a dictionary.

Hands on Examples

python

```python
num = 5 + 2.0
```

python

```python
# Typecasting to int
num_int = int(5.7)

# Typecasting to float
num_float = float("3.14")

# Typecasting to string
message = str(42)

# Typecasting to boolean
is_true = bool(1)
is_false = bool(0)
```

Value ERROR

```
non_numeric_string = "Hello"
num_int = int(non_numeric_string)
print("Converted value:", num_int)
```

```
Error: invalid literal for int() with base 10: 'Hello'
```

Introduction to ASCII Values

The "ASCII value" of a character is a number that represents that character in the computer's memory.

ord() & chr()

In Python, you can use the ord() function to find the ASCII value of a character.

python

```python
char_a = 'A'
char_b = 'B'
ascii_value_a = ord(char_a)
ascii_value_b = ord(char_b)
print("ASCII value of 'A' is:", ascii_value_a)
print("ASCII value of 'B' is:", ascii_value_b)

ascii_value = 65
character = chr(ascii_value)
print("Character of ASCII value 65 is:", character)
```

Answer

```
ASCII value of 'A' is: 65
ASCII value of 'B' is: 66
Character of ASCII value 65 is: A
```

What you have Learnt ?

1. Datatypes
2. Type Conversion
3. ASCII Values

Operators

Agenda

Topics Covered

- Arithmetic operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Bitwise Operators
- Special Operators

Arithmetic Operators

- + Addition: Adds two operands.
- - Subtraction: Subtracts the right operand from the left operand.
- * Multiplication: Multiplies two operands.
- / Division: Divides the left operand by the right operand (returns a float).

Arithmetic Operators

```python
# Addition
a = 5
b = 10
result_addition = a + b
print("Addition:", result_addition)

# Subtraction
c = 7
d = 3
result_subtraction = c - d
print("Subtraction:", result_subtraction)
```

```python
# Multiplication
e = 3
f = 4
result_multiplication = e * f
print("Multiplication:", result_multiplication)

# Division
g = 10
h = 3
result_division = g / h
print("Division:", result_division)
```

Arithmetic Operators

- // Floor Division: Divides the left operand by the right operand and rounds down to the nearest integer (returns an integer).
- % Modulo: Returns the remainder of the division.
- ** Exponentiation: Raises the left operand to the power of the right operand

Arithmetic Operators

```python
# Integer Division (Floor Division)
i = 10
j = 3
result_integer_division = i // j
print("Integer Division (Floor Division):", result_integer_division)

# Modulo (Remainder)
k = 10
l = 3
result_modulo = k % l
print("Modulo (Remainder):", result_modulo)

# Exponentiation
m = 2
n = 3
result_exponentiation = m ** n
print("Exponentiation:", result_exponentiation)
```

Arithmetic Operators Ans

```python
# Integer Division (Floor Division)
i = 10
j = 3
result_integer_division = i // j
print("Integer Division (Floor Division):", result_integer_division)

# Modulo (Remainder)
k = 10
l = 3
result_modulo = k % l
print("Modulo (Remainder):", result_modulo)

# Exponentiation
m = 2
n = 3
result_exponentiation = m ** n
print("Exponentiation:", result_exponentiation)
```

Assignment Operators

- = Assignment: Assigns the value on the right to the variable on the left.
- += Add and Assign: Adds the right operand to the variable on the left and assigns the result to the variable.
- -= Subtract and Assign: Subtracts the right operand from the variable on the left and assigns the result to the variable.

Assignment Operators

```
# Assignment Operator (=)
x = 10
print("Value of x:", x)

# Add and Assign (+=)
y = 5
y += 3
print("Value of y after Add and Assign:", y)

# Subtract and Assign (-=)
z = 8
z -= 2
print("Value of z after Subtract and Assign:", z)
```

Assignment Operators

- *= Multiply and Assign: Multiplies the variable on the left by the right operand and assigns the result to the variable.
- /= Divide and Assign: Divides the variable on the left by the right operand and assigns the result to the variable.
- //= Floor Divide and Assign: Floor divides the variable on the left by the right operand and assigns the result to the variable.

Assignment Operators

```
# Multiply and Assign (*=)
a = 4
a *= 2
print("Value of a after Multiply and Assign:", a)

# Divide and Assign (/=)
b = 10
b /= 2
print("Value of b after Divide and Assign:", b)

# Integer Divide and Assign (//=)
c = 15
c //= 4
print("Value of c after Integer Divide and Assign:", c)
```

Assignment Operators

- %= Modulo and Assign: Calculates the remainder of the division and assigns it to the variable.
- **= Exponentiate and Assign: Raises the variable on the left to the power of the right operand and assigns the result to the variable.

Assignment Operators

```
# Modulo and Assign (%=)
d = 12
d %= 5
print("Value of d after Modulo and Assign:", d)

# Exponentiate and Assign (**=)
e = 2
e **= 3
print("Value of e after Exponentiate and Assign:", e)
```

Comparision Operators

- == Equal to: Checks if two values are equal.
- != Not equal to: Checks if two values are not equal.
- > Greater than: Checks if the left operand is greater than the right operand.
- < Less than: Checks if the left operand is less than the right operand.

Comparision Operators

```python
# Comparison Operator: Equal to (==)
x = 5
y = 10
result_equal = x == y
print("Is x equal to y?", result_equal)

# Comparison Operator: Not Equal to (!=)
a = 7
b = 7
result_not_equal = a != b
print("Is a not equal to b?", result_not_equal)

# Comparison Operator: Greater than (>)
num1 = 15
num2 = 10
result_greater_than = num1 > num2
print("Is num1 greater than num2?", result_greater_than)

# Comparison Operator: Less than (<)
p = 4
q = 8
result_less_than = p < q
print("Is p less than q?", result_less_than)
```

Comparision Operators

- >= Greater than or equal to: Checks if the left operand is greater than or equal to the right operand.
- <= Less than or equal to: Checks if the left operand is less than or equal to the right operand.

Comparision Operators

```
# Comparison Operator: Greater than or Equal to (>=)
m = 5
n = 5
result_greater_equal = m >= n
print("Is m greater than or equal to n?", result_greater_equal)

# Comparison Operator: Less than or Equal to (<=)
r = 10
s = 15
result_less_equal = r <= s
print("Is r less than or equal to s?", result_less_equal)
```

Logical Operators

- and Logical AND: Returns True if both conditions are True.
- or Logical OR: Returns True if at least one condition is True.
- not Logical NOT: Returns True if the condition is False, and vice versa.

Logical Operators

```python
# Logical Operator: AND
x = True
y = False

result_and = x and y
print("Result of x AND y:", result_and)

# Logical Operator: OR
a = True
b = False

result_or = a or b
print("Result of a OR b:", result_or)

# Logical Operator: NOT
p = True

result_not = not p
print("Result of NOT p:", result_not)
```

Membership Operators

- in Membership: Returns True if the value is present in the sequence.
- not in Negated Membership: Returns True if the value is not present in the sequence.

Membership Operators

```python
# List of fruits
fruits = ['apple', 'banana', 'orange', 'grape']

# Check if 'apple' is in the list
is_apple_in_list = 'apple' in fruits
print("'apple' is in the list:", is_apple_in_list)

# Check if 'watermelon' is not in the list
is_watermelon_not_in_list = 'watermelon' not in fruits
print("'watermelon' is not in the list:", is_watermelon_not_in_list)
```

Identity Operators

- is Identity: Returns True if both variables point to the same object.
- is not Negated Identity: Returns True if the variables point to different objects.

Identity Operators

```python
# Variables with the same value
name1 = "John"
name2 = "John"

# Check if name1 and name2 refer to different objects in memory
result_is_not = name1 is not name2
print("name1 is not name2?", result_is_not)

# Variables with different values
num1 = 10
num2 = 20

# Check if num1 and num2 refer to different objects in memory
result_is = num1 is not num2
print("num1 is not num2?", result_is)
```

Bitwise Operators

- Bitwise AND (&): Sets each bit to 1 if both bits are 1.
- Bitwise OR (|): Sets each bit to 1 if at least one of the bits is 1.
- Bitwise XOR (^): Sets each bit to 1 if only one of the bits is 1.
- Bitwise NOT (~): Inverts the bits, changing 1 to 0 and 0 to 1.

Bitwise Operators

```python
# Bitwise AND (&) Operator
result_and = 6 & 3
print("Bitwise AND:", result_and)

# Bitwise OR (|) Operator
result_or = 6 | 3
print("Bitwise OR:", result_or)

# Bitwise XOR (^) Operator
result_xor = 6 ^ 3
print("Bitwise XOR:", result_xor)

# Bitwise NOT (~) Operator
result_not = ~6
print("Bitwise NOT:", result_not)
```

Bitwise Operators

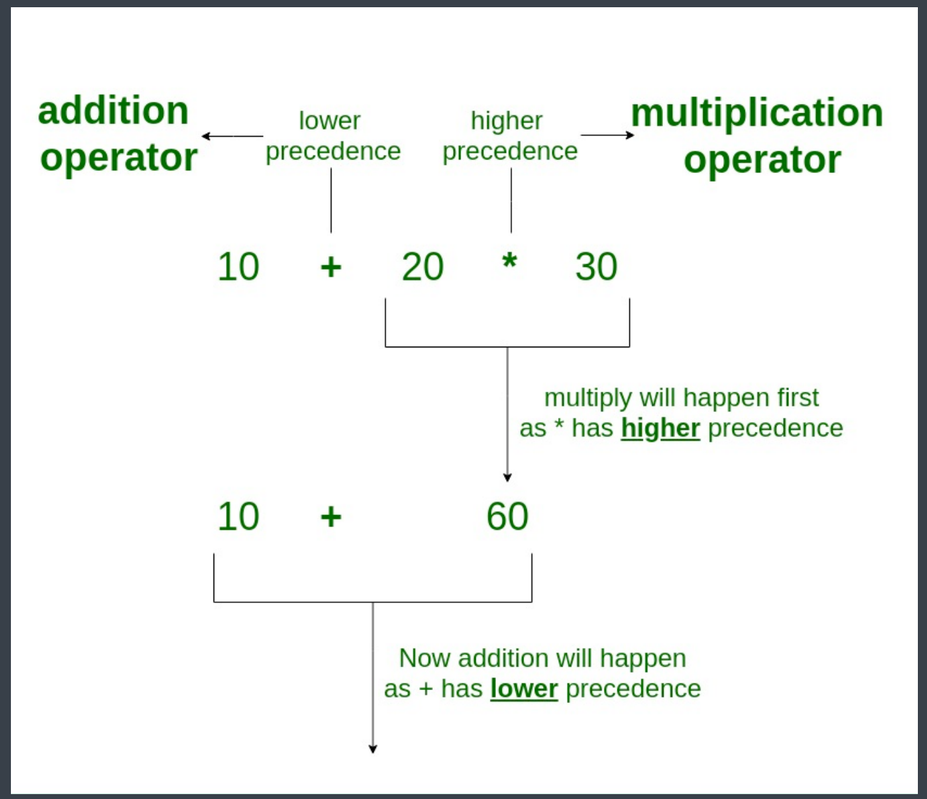
- Bitwise Left Shift (<<): Shifts the bits to the left by a specified number of positions.
- Bitwise Right Shift (>>): Shifts the bits to the right by a specified number of positions

Bitwise Operators

```
# Bitwise Left Shift (<<) Operator
result_left_shift = 2 << 2
print("Bitwise Left Shift:", result_left_shift)

# Bitwise Right Shift (>>) Operator
result_right_shift = 32 >> 2
print("Bitwise Right Shift:", result_right_shift)
```

Python Operator Precedence



Python Operator Precedence

Python Operator Precedence

Precedence	Operator Sign	Operator Name
Highest	**	Exponentiation
	+x, -x, ~x	Unary positive, unary negative, bitwise negation
	*, /, //, %	Multiplication, division, floor, division, modulus
	+, -	Addition, subtraction
	<<, >>	Left-shift, right-shift
	&	Bitwise AND
	^	Bitwise XOR
	|	Bitwise OR
	==, !=, <, <=, >, >=, is, is not	Comparison, identity
	not	Boolean NOT
	and	Boolean AND
Lowest	or	Boolean OR

What you have Learnt ?

1. Arithmetic operators
2. Assignment Operators
3. Comparison Operators
4. Logical Operators
5. Bitwise Operators
6. Operator Precedence

Comments

Comments

In computer programming, comments are hints that we use to make our code more understandable.

Comments are completely ignored by the interpreter.

Types of Comments

Single-line comments

These comments begin with a # symbol and continue until the end of the line.

Multi-line comments

Docstrings are enclosed in triple quotes (''' or """) and can span multiple lines.