Python

TECHNOLOGY

Operators

Learning Objectives

By the end of this lesson, you will be able to:

- Understand basic operators
- Analyze different type of operators

TECHNOLOGY

Operators in Python

Operators in Python

In Python, an operator is a symbol or a keyword that performs an operation on one or more operands (variables or values) and returns a result.

There are several types of operators in Python:

1. Arithmetic operators

2. Comparison operators

3. Logical operators

Operators in Python

4. Bitwise operators

5. Assignment operators

6. Membership operators

7. Identity operators

By using these operators, we can perform different types of operations in Python and make our programs more efficient and effective.

TECHNOLOGY

Types of Operators in Python

Types of Operators

In Python, operators can be categorized into several types based on their functionality.

Here are the main types of operators in Python:

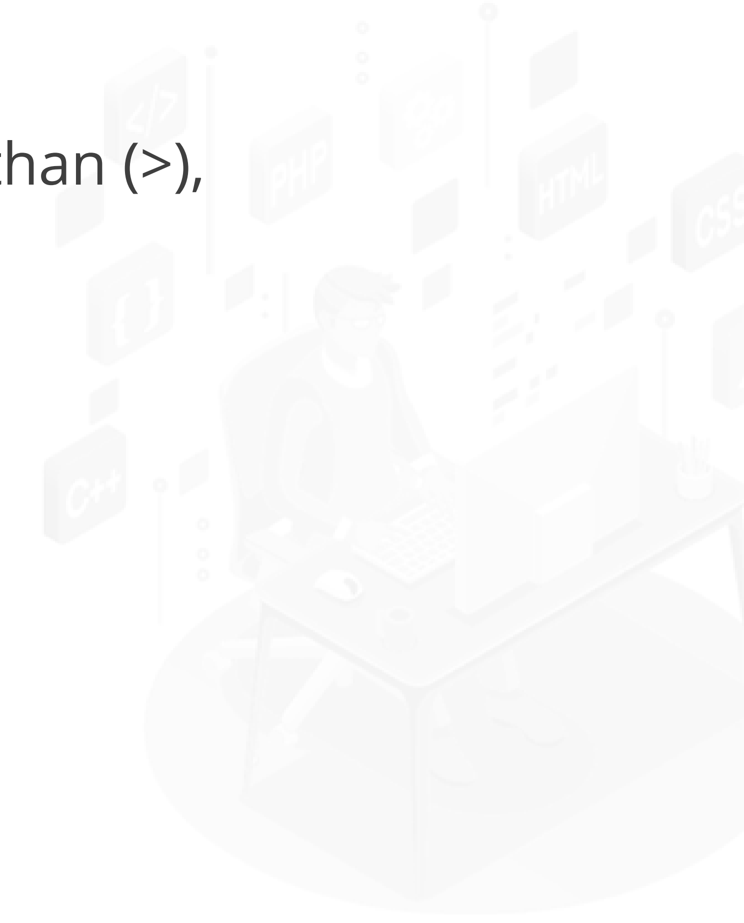
1. Arithmetic operators: These operators are used to perform arithmetic operations on numeric values.

Examples of arithmetic operators include addition (+), subtraction (-), multiplication (*), division (/), modulus (%), and exponentiation (**).

Types of Operators

2. Comparison operators: These operators are used to compare two values and return a Boolean value indicating whether the comparison is true or false.

Examples of comparison operators include equal to (==), not equal to (!=), greater than (>), less than (<), greater than or equal to (>=), and less than or equal to (<=).



Types of Operators

3. Logical operators: These operators are used to perform logical operations on Boolean values.

Examples of logical operators include **and, or,** and **not.**

4. Bitwise operators: These operators are used to perform bitwise operations on integers. Examples of bitwise operators include **bitwise AND (&), bitwise OR (|), bitwise XOR (^), bitwise NOT (~), left shift (<<),** and **right shift (>>).**

5. Assignment operators: These operators are used to assign values to variables.

Examples of assignment operators include **=, +=, -=, *=, /=, %=, **=, &=, |=, ^=, <<=, and >>=.**

Types of Operators

6. Membership operators: These operators are used to test whether a value is a member of a sequence or collection.

Examples of membership operators include in and not in.

7. Identity operators: These operators are used to test whether two objects are the same object or not.

Examples of identity operators include is and is not.

TECHNOLOGY

Arithmetic Operators in Python

Arithmetic Operators

Arithmetic operators in Python are used to perform arithmetic operations on numeric values.

Here are the main arithmetic operators in Python:

1. **Addition (+):** Adds two operands together.

2. **Subtraction (-):** Subtracts the second operand from the first operand.

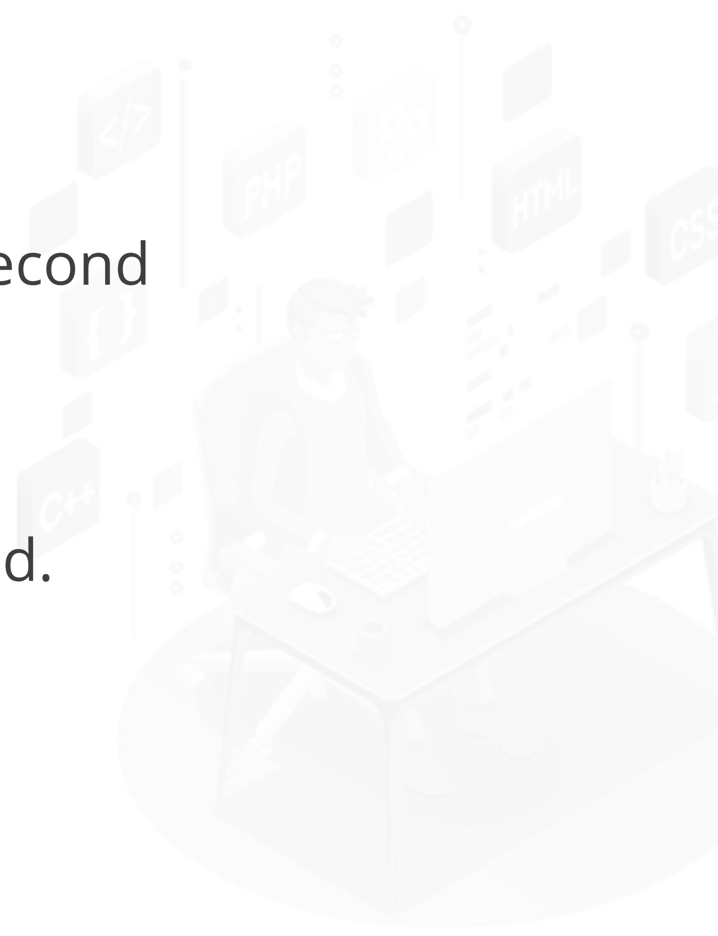
3. **Multiplication (*):** Multiplies two operands together.

Arithmetic Operators

4. Division (/): Divides the first operand by the second operand.

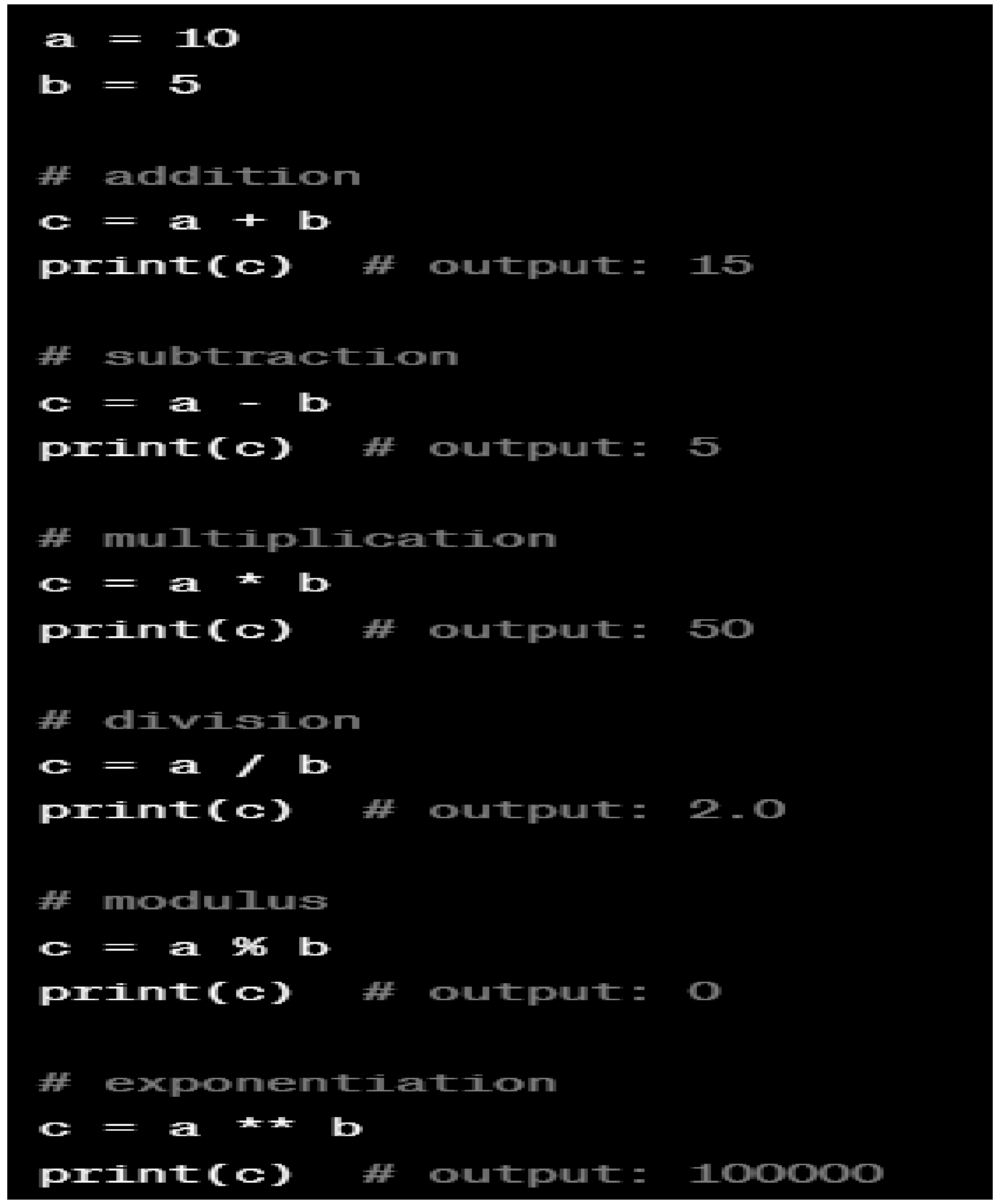
5. Modulus (%): Returns the remainder of the division of the first operand by the second operand.

6. Exponentiation (**): Raises the first operand to the power of the second operand.



Arithmetic Operators

Here's an example code snippet that demonstrates the use of arithmetic operators in Python:



```
a = 10
b = 5

# addition
c = a + b
print(c)  # output: 15

# subtraction
c = a - b
print(c)  # output: 5

# multiplication
c = a * b
print(c)  # output: 50

# division
c = a / b
print(c)  # output: 2.0

# modulus
c = a % b
print(c)  # output: 0

# exponentiation
c = a ** b
print(c)  # output: 100000
```

In this example, we assign the values 10 and 5 to variables a and b, respectively. We then perform arithmetic operations using the arithmetic operators and assign the result to the variable c. Finally, we print the value of c for each operation.

TECHNOLOGY

Assignment Operators in Python

Assignment Operators

Assignment operators in Python are used to assign values to variables. They are a shorthand way of writing an expression that assigns a value to a variable and then uses that variable in an operation.

Here are the main assignment operators in Python:

1. **Assignment operator (=):** Assigns a value to a variable.

2. **Addition assignment (+=):** Adds the value on the right-hand side to the variable on the left-hand side and assigns the result to the variable on the left-hand side.

3. **Subtraction assignment (-=):** Subtracts the value on the right-hand side from the variable on the left-hand side and assigns the result to the variable on the left-hand side.

Assignment Operators

4. Multiplication assignment (*=): Multiplies the value on the right-hand side by the variable on the left-hand side and assigns the result to the variable on the left-hand side.

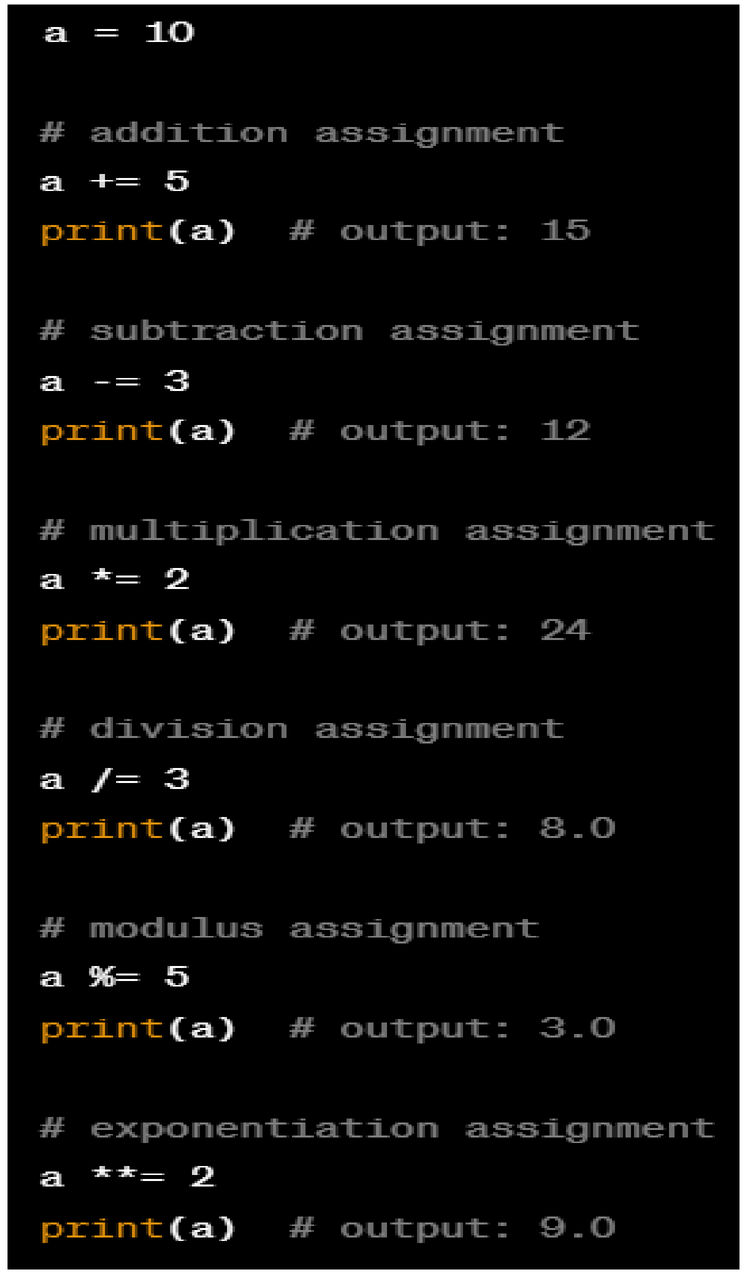
5. Division assignment (/=): Divides the variable on the left-hand side by the value on the right-hand side and assigns the result to the variable on the left-hand side.

6. Modulus assignment (%=): Calculates the remainder of the division of the variable on the left-hand side by the value on the right-hand side and assigns the result to the variable on the left-hand side.

7. Exponentiation assignment (=):** Raises the variable on the left-hand side to the power of the value on the right-hand side and assigns the result to the variable on the left-hand side.

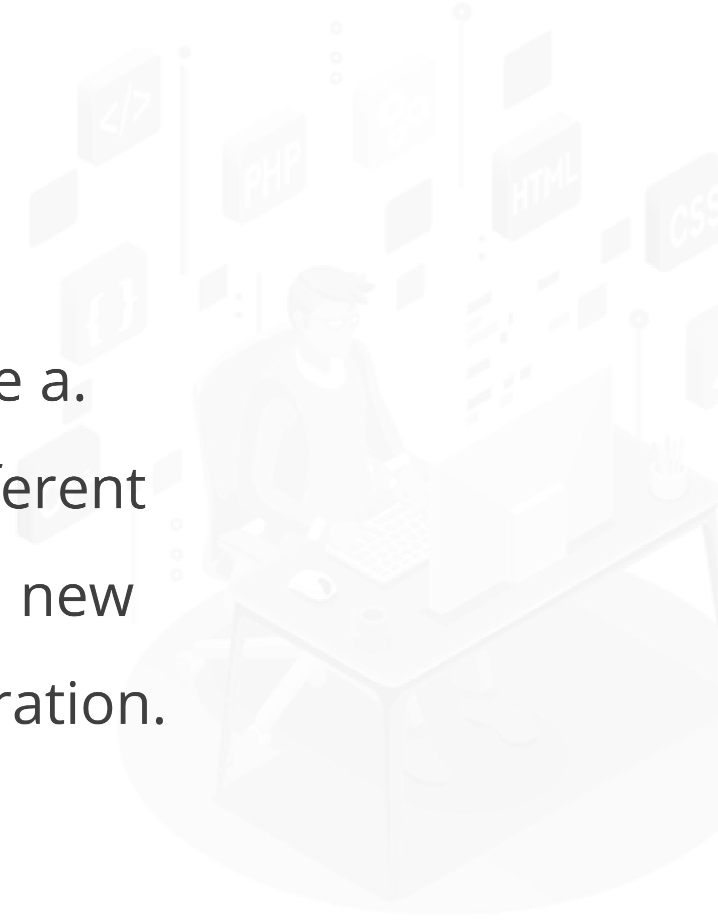
Assignment Operators

Here's an example code snippet that demonstrates the use of assignment operators in Python:



```
a = 10

# addition assignment
a += 5
print(a)   # output: 15

# subtraction assignment
a -= 3
print(a)   # output: 12

# multiplication assignment
a *= 2
print(a)   # output: 24

# division assignment
a /= 3
print(a)   # output: 8.0

# modulus assignment
a %= 5
print(a)   # output: 3.0

# exponentiation assignment
a **= 2
print(a)   # output: 9.0
```

In this example, we first assign the value 10 to variable a. We then use the assignment operators to perform different operations on the value of a, and assign the result to a new value of a. Finally, we print the value of a for each operation.



TECHNOLOGY

Logical Operators in Python

Logical Operators

Logical operators in Python are used to perform logical operations on Boolean values (True and False).

There are three logical operators in Python:

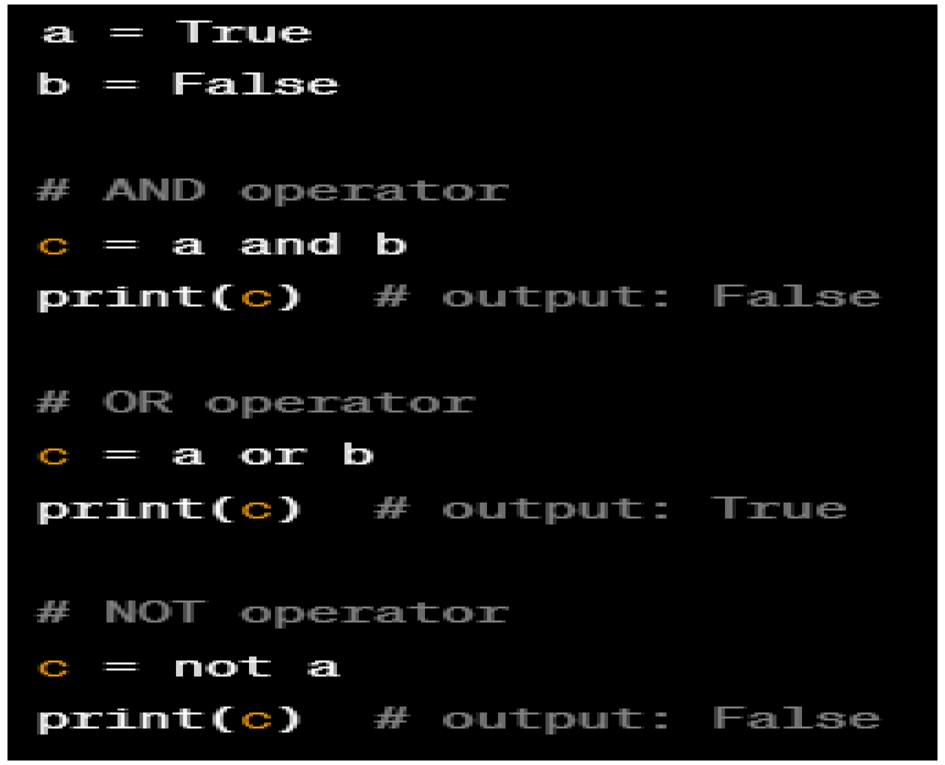
1. AND operator (and): Returns True if both operands are True, otherwise it returns False.

2. OR operator (or): Returns True if at least one of the operands is True, otherwise it returns False.

3. NOT operator (not): Returns the opposite of the operand. If the operand is True, it returns False, and vice versa.

Logical Operators

Here's an example code snippet that demonstrates the use of logical operators in Python:



```
a = True
b = False

# AND operator
c = a and b
print(c)  # output: False

# OR operator
c = a or b
print(c)  # output: True

# NOT operator
c = not a
print(c)  # output: False
```

In this example, we assign True and False values to variables a and b, respectively. We then perform logical operations using the logical operators and assign the result to the variable c. Finally, we print the value of c for each operation.

TECHNOLOGY

Bitwise Operators in Python

Bitwise Operators

Bitwise operators in Python are used to perform operations on binary representations of integers.

There are six bitwise operators in Python:

1. Bitwise AND operator (&): Returns 1 in each bit position for which both corresponding bits in the operands are 1, otherwise it returns 0.

2. Bitwise OR operator (|): Returns 1 in each bit position for which at least one of the corresponding bits in the operands is 1, otherwise it returns 0.

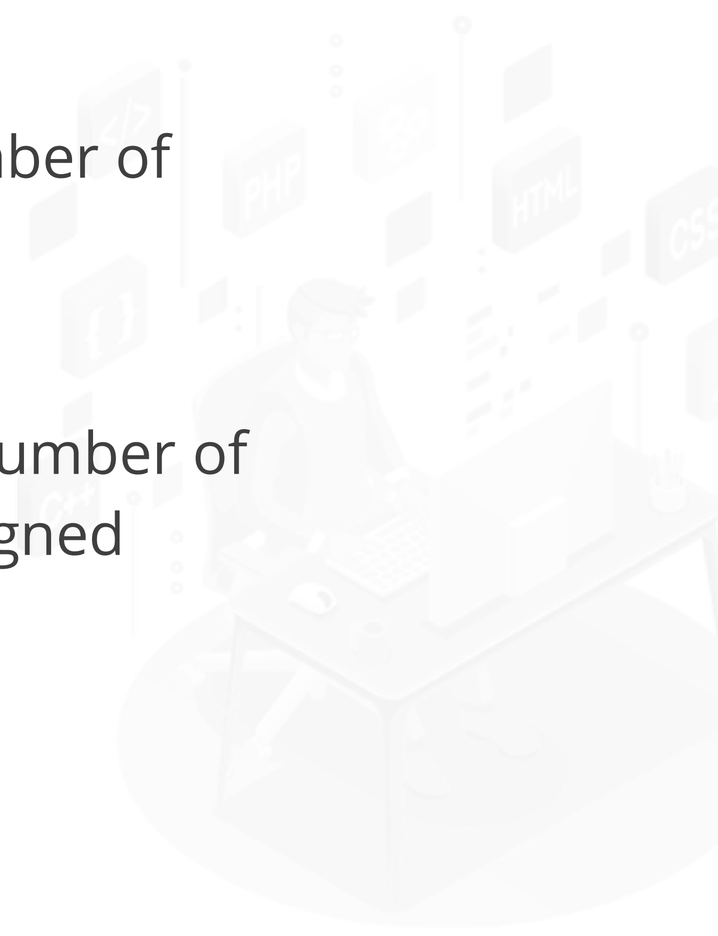
3. Bitwise XOR operator (^): Returns 1 in each bit position for which only one of the corresponding bits in the operands is 1, otherwise it returns 0.

Bitwise Operators

4. Bitwise NOT operator (~): Returns the complement of the operand, i.e., it flips all the bits.

5. Left shift operator (<<): Shifts the bits of the first operand to the left by the number of positions specified by the second operand. The shifted bits are filled with 0.

6. Right shift operator (>>): Shifts the bits of the first operand to the right by the number of positions specified by the second operand. The shifted bits are filled with 0 for unsigned numbers and with the sign bit for signed numbers.



TECHNOLOGY

Relational Operators in Python

Relational Operators

Relational operators in Python are used to compare two values and return a Boolean value (True or False) depending on whether the comparison is true or false.

There are six relational operators in Python:

1. Equal to operator (==): Returns True if both operands are equal, otherwise it returns False.

2. Not equal to operator (!=): Returns True if both operands are not equal, otherwise it returns False.

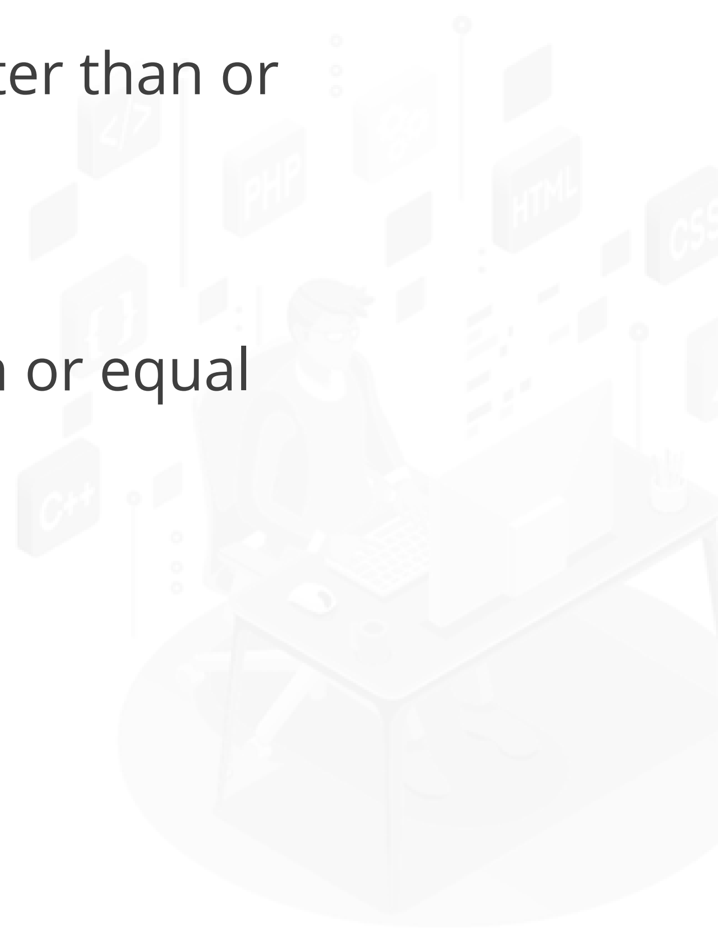
3. Greater than operator (>): Returns True if the left operand is greater than the right operand, otherwise it returns False.

Relational Operators

4. Less than operator (<): Returns True if the left operand is less than the right operand else, it returns False.

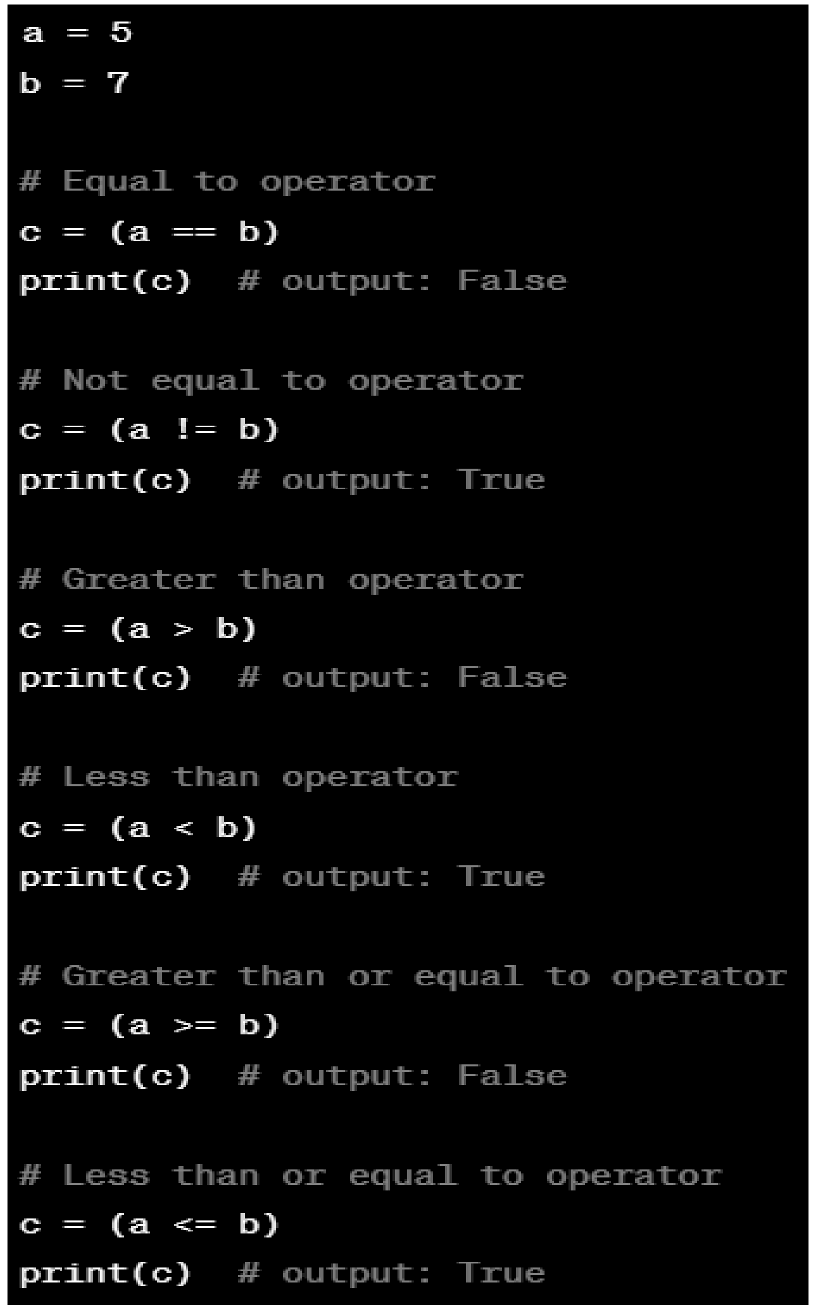
5. Greater than or equal to operator (>=): Returns True if the left operand is greater than or equal to the right operand else, it returns False.

6. Less than or equal to operator (<=): Returns True if the left operand is less than or equal to the right operand else, it returns False.



Relational Operators

Here's an example code snippet that demonstrates the use of relational operators in Python:



```
a = 5
b = 7

# Equal to operator
c = (a == b)
print(c)  # output: False

# Not equal to operator
c = (a != b)
print(c)  # output: True

# Greater than operator
c = (a > b)
print(c)  # output: False

# Less than operator
c = (a < b)
print(c)  # output: True

# Greater than or equal to operator
c = (a >= b)
print(c)  # output: False

# Less than or equal to operator
c = (a <= b)
print(c)  # output: True
```

In this example, we assign the values 5 and 7 to variables a and b, respectively. We then perform relational operations using the relational operators and assign the result to the variable c. Finally, we print the value of c for each operation.

TECHNOLOGY

Special Operators in Python

Special Operators

Special operators in Python perform special operations on Python data types, such as membership and identity testing.

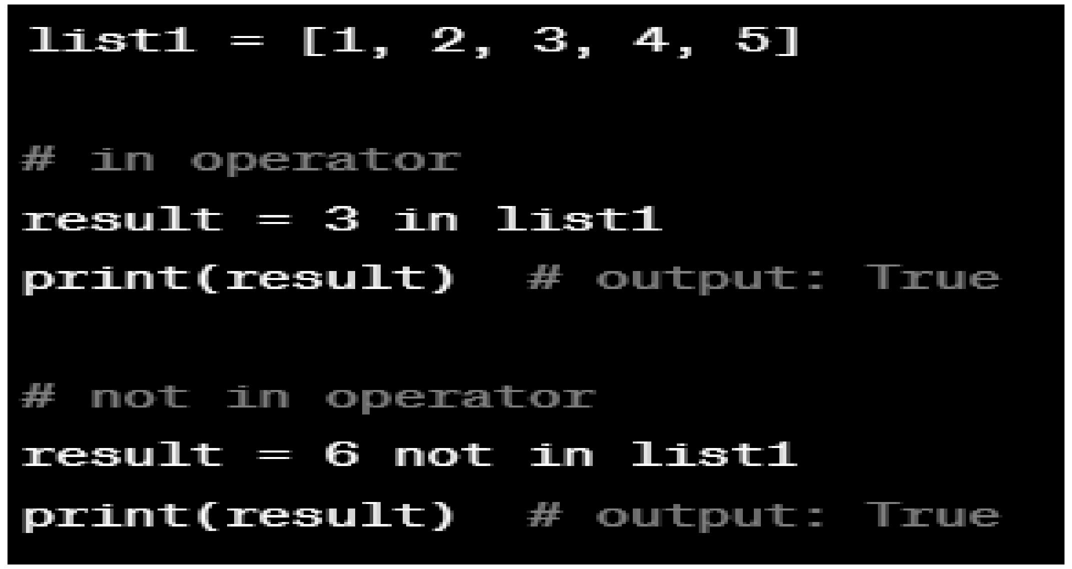
There are two types of special operators in Python:

1. Membership operators: Membership operators are used to test if a sequence is present in an object. There are two membership operators in Python:

- ` **in**` operator: Returns True if a value is found in the sequence, otherwise, it returns False.
- ` **not in**` operator: Returns True if a value is not found in the sequence, otherwise, it returns False.

Special Operators

Here's an example code snippet that demonstrates the use of membership operators in Python:



```
list1 = [1, 2, 3, 4, 5]

# in operator
result = 3 in list1
print(result)  # output: True

# not in operator
result = 6 not in list1
print(result)  # output: True
```

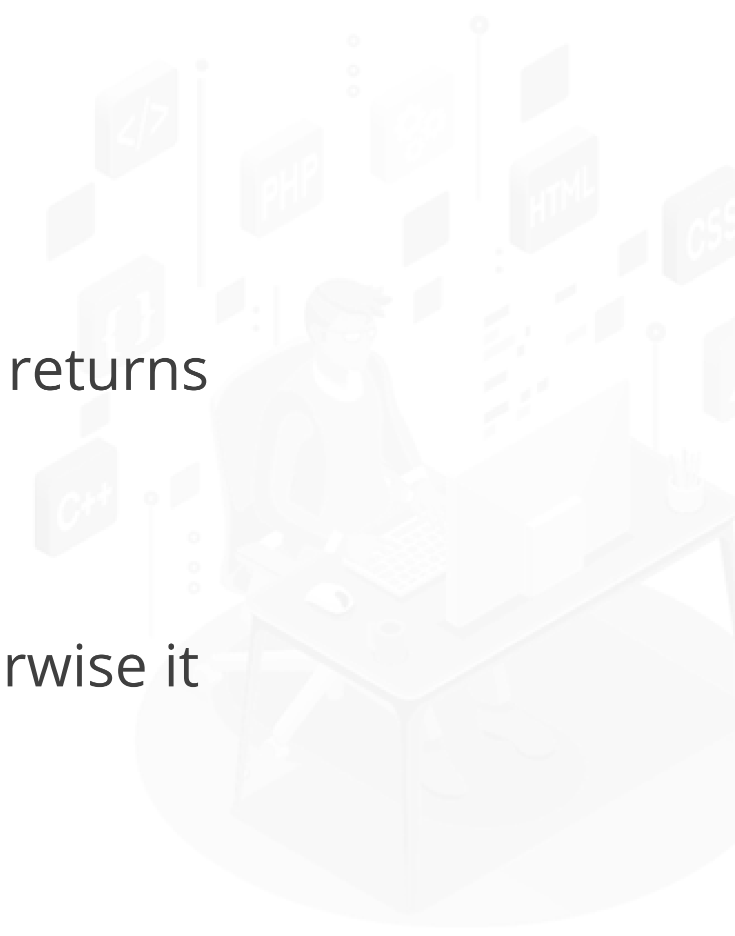
This example defines a list **`list1`** with five elements. We then use the **`in`** operator to check if the value 3 is present in the list, and the **`not in`** operator to check if the value 6 is not present in the list.

Special Operators

2. Identity operators: Identity operators are used to compare the objects' memory locations to check whether they are the same.

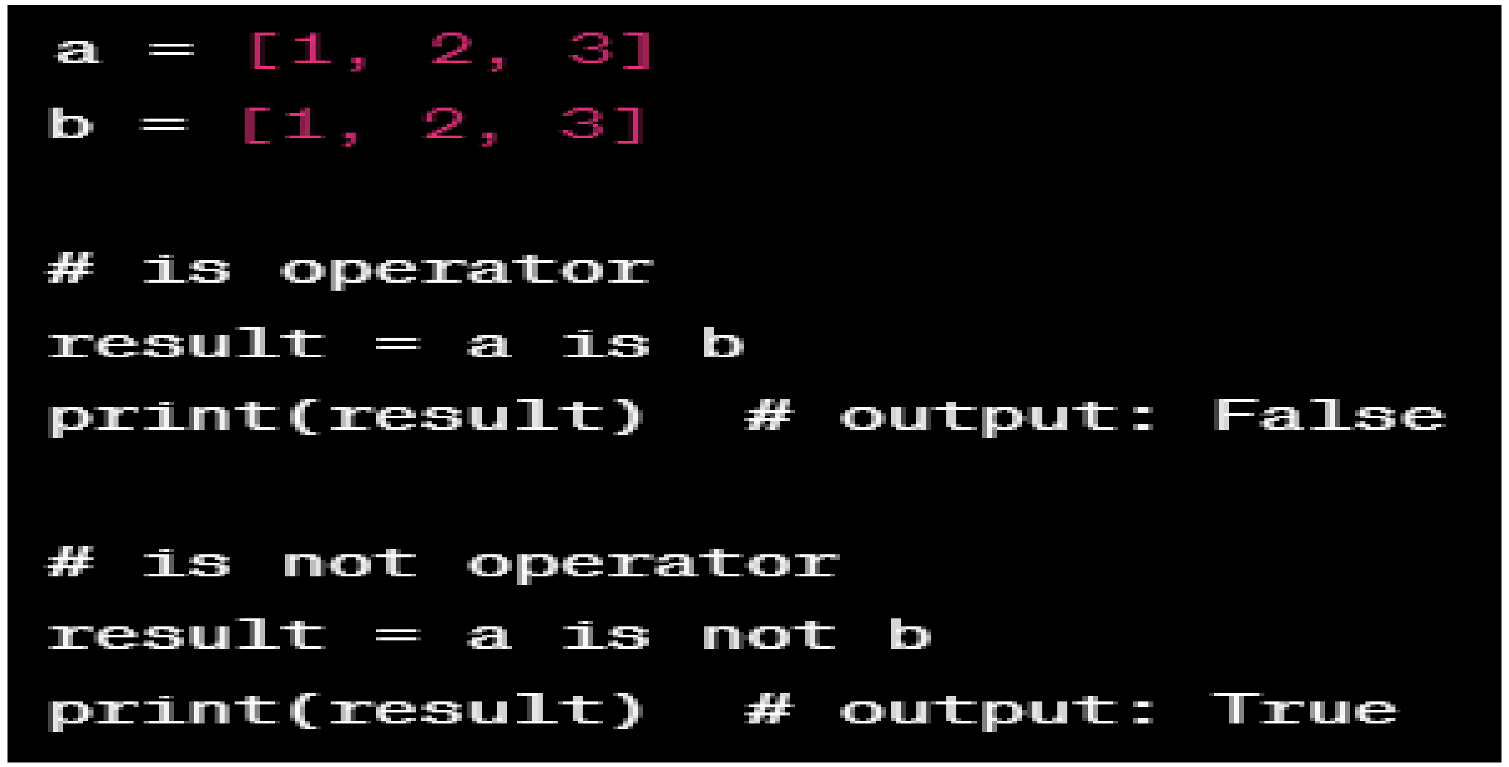
There are two identity operators in Python:

- **\`is\` operator:** Returns True if both operands are the same object, otherwise it returns False.
- **\`is not\`** operator: Returns True if both operands are not the same object, otherwise it returns False.



Special Operators

Here's an example code snippet that demonstrates the use of identity operators in Python:



```
a = [1, 2, 3]
b = [1, 2, 3]

# is operator
result = a is b
print(result)  # output: False

# is not operator
result = a is not b
print(result)  # output: True
```

In this example, we define two lists `` `a` `` and `` `b` `` with the same values. However, since they are two different objects with different memory locations, the **`is`** operator returns False. The **`is not`** operator returns True since the two lists are not the same object.

TECHNOLOGY

Membership Operators in Python

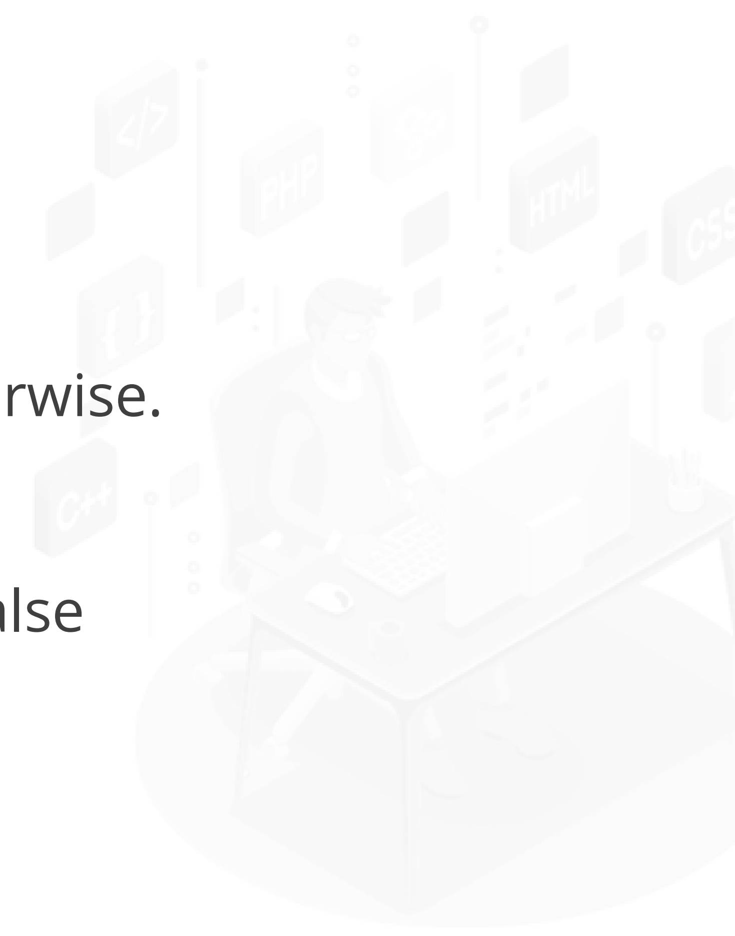
Membership Operators

Membership operators in Python are used to test whether a value is a member of a sequence or not.

There are two membership operators in Python:

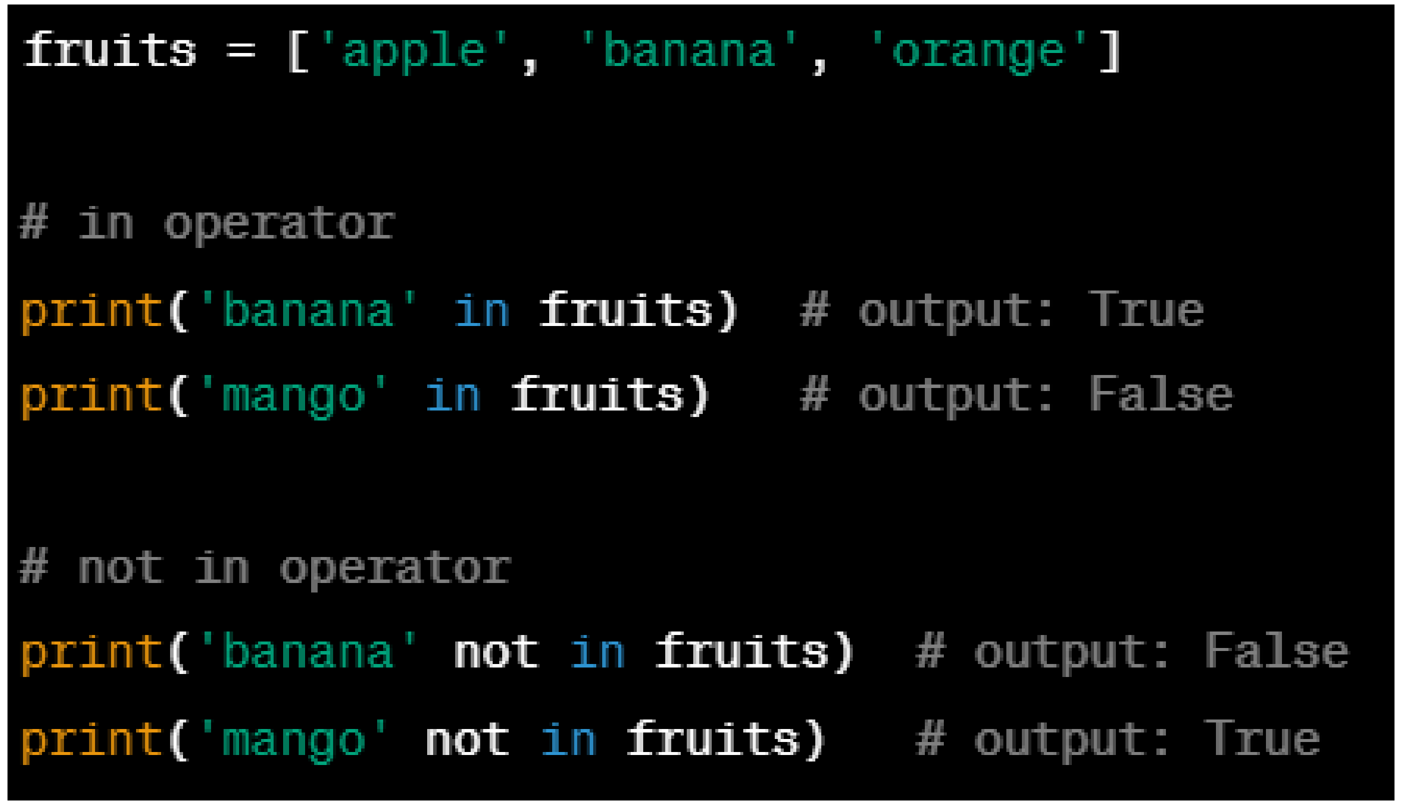
1. `**in operator**`: It returns True if a value is found in the sequence, and False otherwise.

2. `**not in operator**`: It returns True if a value is not found in the sequence, and False otherwise.



Membership Operators

Here's an example code snippet that demonstrates the use of membership operators in Python:



```
fruits = ['apple', 'banana', 'orange']

# in operator
print('banana' in fruits)  # output: True
print('mango' in fruits)   # output: False

# not in operator
print('banana' not in fruits)  # output: False
print('mango' not in fruits)   # output: True
```

In this example, we define a list `**fruits**` containing three fruits. We then use the `**in**` operator to check if **'banana'** is present in the list, and **'mango'** is not present in the list.

We also use the `**not in**` operator to check if **'banana'** is not present in the list, and **'mango'** is present in the list.

TECHNOLOGY

Identity Operators in Python

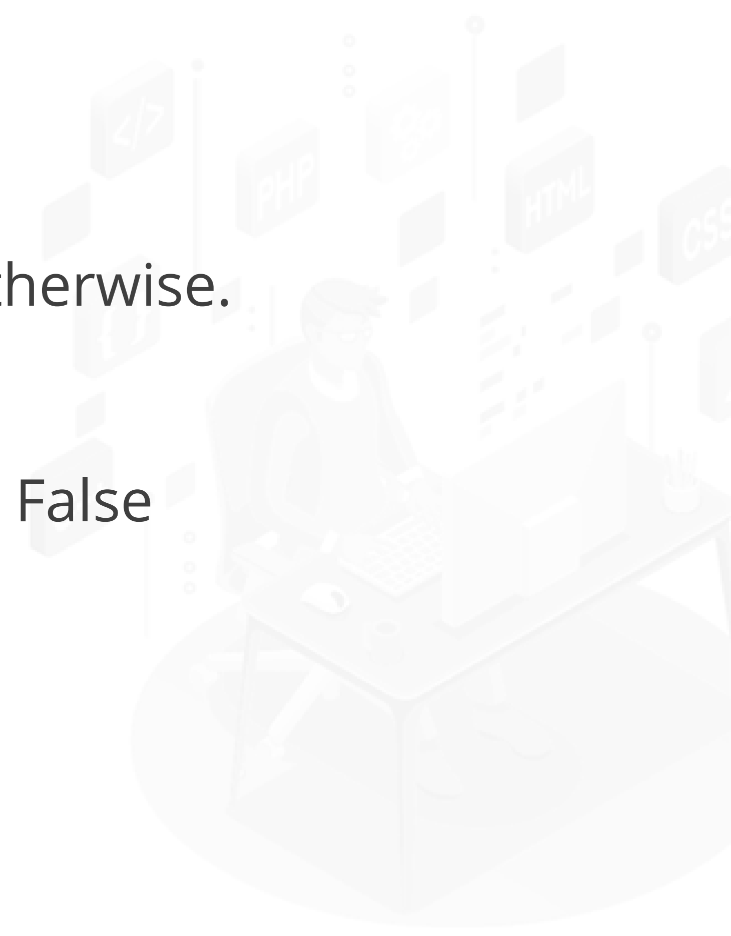
Identity Operators

Identity operators in Python are used to compare the memory locations of two objects to check if they are the same or not.

There are two identity operators in Python:

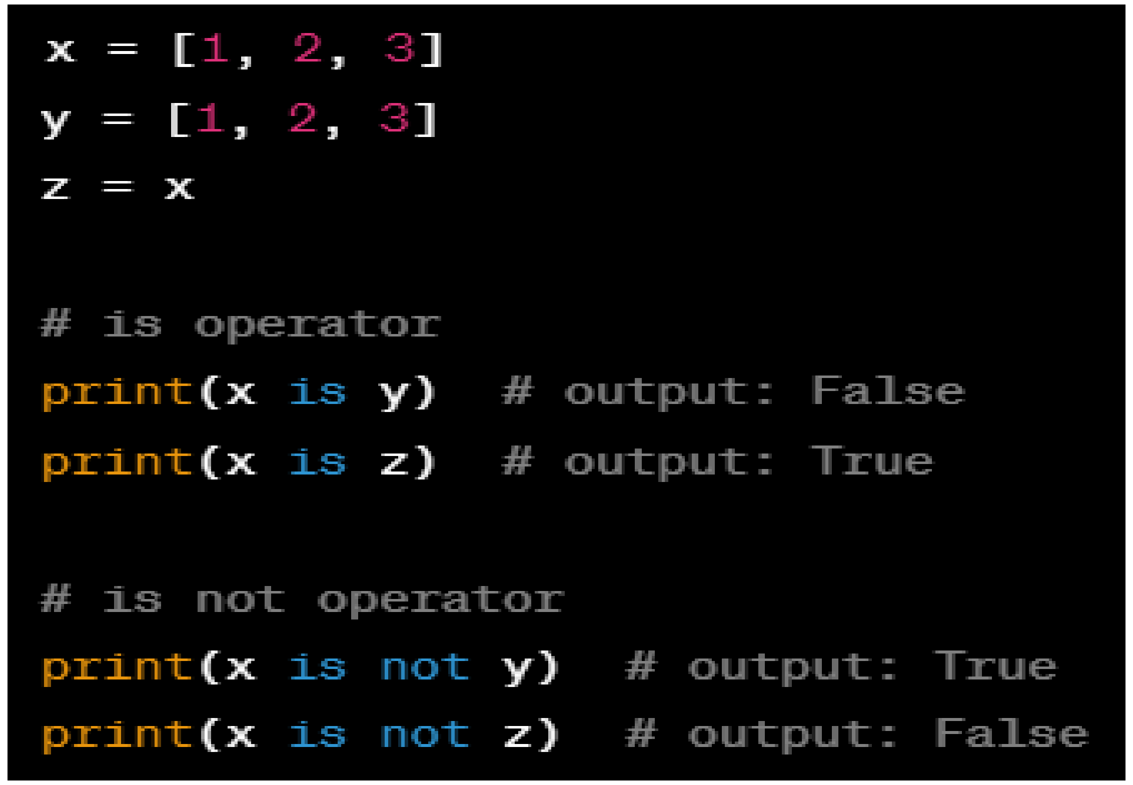
1. `**is operator**`: It returns True if both operands are the same object, and False otherwise.

2. `**is not operator**`: It returns True if both operands are not the same object, and False otherwise.



Identity Operators

Here's an example code snippet that demonstrates the use of identity operators in Python:



```python
x = [1, 2, 3]
y = [1, 2, 3]
z = x

# is operator
print(x is y)  # output: False
print(x is z)  # output: True

# is not operator
print(x is not y)  # output: True
print(x is not z)  # output: False
```

In this example, we define three lists `x`, `y`, and `z`. Lists `x` and `y` contain the same values, but they are two different objects with different memory locations.

Identity Operators

The `**is**` operator returns False when comparing `**x**` and `**y**`. However, list `**z**` is assigned to the same memory location as `**x**`, so the `**is**` operator returns True when comparing `**x**` and `**z**`.

The `**is not**` operator returns True when comparing `**x**` and `**y**`, and False when comparing `**x**` and `**z**`.

Key Takeaways

- Operators can be categorized into several types based on their functionality.
- Arithmetic operators in Python are used to perform arithmetic operations on numeric values.
- Logical operators in Python are used to perform logical operations on Boolean values (True and False).
- Identity operators in Python are used to compare the memory locations of two objects to check if they are the same or not.