Python Operators

Introduction:

In this article, we are discussing Python Operators. The operator is a symbol that performs a specific operation between two operands, according to one definition. Operators serve as the foundation upon which logic is constructed in a program in a particular programming language. In every programming language, some operators perform several tasks. Same as other languages, Python also has some operators, and these are given below -

* Arithmetic operators
* Comparison operators
* Assignment Operators
* Logical Operators
* Bitwise Operators
* Membership Operators
* Identity Operators
* Arithmetic Operators

Arithmetic Operators

Arithmetic operators used between two operands for a particular operation. There are many arithmetic operators. It includes the exponent (\*\*) operator as well as the + (addition), - (subtraction), \* (multiplication), / (divide), % (reminder), and // (floor division) operators.

Consider the following table for a detailed explanation of arithmetic operators.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| **+ (Addition)** | It is used to add two operands. For example, if a = 10, b = 10 => a+b = 20 |
| **- (Subtraction)** | It is used to subtract the second operand from the first operand. If the first operand is less than the second operand, the value results negative. For example, if a = 20, b = 5 => a - b = 15 |
| **/ (divide)** | It returns the quotient after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a/b = 2.0 |
| **\* (Multiplication)** | It is used to multiply one operand with the other. For example, if a = 20, b = 4 => a \* b = 80 |
| **% (reminder)** | It returns the reminder after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a%b = 0 |
| **\*\* (Exponent)** | As it calculates the first operand's power to the second operand, it is an exponent operator. |
| **// (Floor division)** | It provides the quotient's floor value, which is obtained by dividing the two operands. |

**Program Code:**

Now we give code examples of arithmetic operators in Python. The code is given below -

1. a = 32    # Initialize the value of a
2. b = 6      # Initialize the value of b
3. **print**('Addition of two numbers:',a+b)
4. **print**('Subtraction of two numbers:',a-b)
5. **print**('Multiplication of two numbers:',a\*b)
6. **print**('Division of two numbers:',a/b)
7. **print**('Reminder of two numbers:',a%b)
8. **print**('Exponent of two numbers:',a\*\*b)
9. **print**('Floor division of two numbers:',a//b)

**Output:**

Now we compile the above code in Python, and after successful compilation, we run it. Then the output is given below -

Addition of two numbers: 38

Subtraction of two numbers: 26

Multiplication of two numbers: 192

Division of two numbers: 5.333333333333333

Reminder of two numbers: 2

Exponent of two numbers: 1073741824

Floor division of two numbers: 5

AD

Comparison operator

Comparison operators mainly use for comparison purposes. Comparison operators compare the values of the two operands and return a true or false Boolean value in accordance. The example of comparison operators are ==, !=, <=, >=, >, <. In the below table, we explain the works of the operators.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| == | If the value of two operands is equal, then the condition becomes true. |
| != | If the value of two operands is not equal, then the condition becomes true. |
| <= | The condition is met if the first operand is smaller than or equal to the second operand. |
| >= | The condition is met if the first operand is greater than or equal to the second operand. |
| > | If the first operand is greater than the second operand, then the condition becomes true. |
| **<** | If the first operand is less than the second operand, then the condition becomes true. |

**Program Code:**

Now we give code examples of Comparison operators in Python. The code is given below -

1. a = 32      # Initialize the value of a
2. b = 6       # Initialize the value of b
3. **print**('Two numbers are equal or not:',a==b)
4. **print**('Two numbers are not equal or not:',a!=b)
5. **print**('a is less than or equal to b:',a<=b)
6. **print**('a is greater than or equal to b:',a>=b)
7. **print**('a is greater b:',a>b)
8. **print**('a is less than b:',a<b)

**Output:**

Now we compile the above code in Python, and after successful compilation, we run it. Then the output is given below -

Two numbers are equal or not: False

Two numbers are not equal or not: True

a is less than or equal to b: False

a is greater than or equal to b: True

a is greater b: True

a is less than b: False

Assignment Operators

Using the assignment operators, the right expression's value is assigned to the left operand. There are some examples of assignment operators like =, +=, -=, \*=, %=, \*\*=, //=. In the below table, we explain the works of the operators.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| = | It assigns the value of the right expression to the left operand. |
| += | By multiplying the value of the right operand by the value of the left operand, the left operand receives a changed value. For example, if a = 10, b = 20 => a+ = b will be equal to a = a+ b and therefore, a = 30. |
| -= | It decreases the value of the left operand by the value of the right operand and assigns the modified value back to left operand. For example, if a = 20, b = 10 => a- = b will be equal to a = a- b and therefore, a = 10. |
| \*= | It multiplies the value of the left operand by the value of the right operand and assigns the modified value back to then the left operand. For example, if a = 10, b = 20 => a\* = b will be equal to a = a\* b and therefore, a = 200. |
| %= | It divides the value of the left operand by the value of the right operand and assigns the reminder back to the left operand. For example, if a = 20, b = 10 => a % = b will be equal to a = a % b and therefore, a = 0. |
| \*\*= | a\*\*=b will be equal to a=a\*\*b, for example, if a = 4, b =2, a\*\*=b will assign 4\*\*2 = 16 to a. |
| //= | A//=b will be equal to a = a// b, for example, if a = 4, b = 3, a//=b will assign 4//3 = 1 to a. |

**Program Code:**

Now we give code examples of Assignment operators in Python. The code is given below -

1. a = 32         # Initialize the value of a
2. b = 6          # Initialize the value of b
3. **print**('a=b:', a==b)
4. **print**('a+=b:', a+b)
5. **print**('a-=b:', a-b)
6. **print**('a\*=b:', a\*b)
7. **print**('a%=b:', a%b)
8. **print**('a\*\*=b:', a\*\*b)
9. **print**('a//=b:', a//b)

**Output:**

Now we compile the above code in Python, and after successful compilation, we run it. Then the output is given below -

AD

a=b: False

a+=b: 38

a-=b: 26

a\*=b: 192

a%=b: 2

a\*\*=b: 1073741824

a//=b: 5

Bitwise Operators

The two operands' values are processed bit by bit by the bitwise operators. The examples of Bitwise operators are bitwise OR (|), bitwise AND (&), bitwise XOR (^), negation (~), Left shift (<<), and Right shift (>>). Consider the case below.

**For example,**

1. **if** a = 7
2. b = 6
3. then, binary (a) = 0111
4. binary (b) = 0110
6. hence, a & b = 0011
7. a | b = 0111
8. a ^ b = 0100
9. ~ a = 1000
10. Let, Binary of x = 0101
11. Binary of y = 1000
12. Bitwise OR = 1101
13. 8 4 2 1
14. 1 1 0 1 = 8 + 4 + 1 = 13
16. Bitwise AND = 0000
17. 0000 = 0
19. Bitwise XOR = 1101
20. 8 4 2 1
21. 1 1 0 1 = 8 + 4 + 1 = 13
22. Negation of x = ~x = (-x) - 1 = (-5) - 1 = -6
23. ~x = -6

In the below table, we are explaining the works of the bitwise operators.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| & (binary and) | A 1 is copied to the result if both bits in two operands at the same location are 1. If not, 0 is copied. |
| | (binary or) | The resulting bit will be 0 if both the bits are zero; otherwise, the resulting bit will be 1. |
| ^ (binary xor) | If the two bits are different, the outcome bit will be 1, else it will be 0. |
| ~ (negation) | The operand's bits are calculated as their negations, so if one bit is 0, the next bit will be 1, and vice versa. |
| << (left shift) | The number of bits in the right operand is multiplied by the leftward shift of the value of the left operand. |
| >> (right shift) | The left operand is moved right by the number of bits present in the right operand. |

**Program Code:**

Now we give code examples of Bitwise operators in Python. The code is given below -

AD

1. a = 5          # initialize the value of a
2. b = 6          # initialize the value of b
3. **print**('a&b:', a&b)
4. **print**('a|b:', a|b)
5. **print**('a^b:', a^b)
6. **print**('~a:', ~a)
7. **print**('a<<b:', a<<b)
8. **print**('a>>b:', a>>b)

**Output:**

Now we compile the above code in Python, and after successful compilation, we run it. Then the output is given below -

a&b: 4

a|b: 7

a^b: 3

~a: -6

a<>b: 0

AD

Logical Operators

The assessment of expressions to make decisions typically uses logical operators. The examples of logical operators are and, or, and not. In the case of logical AND, if the first one is 0, it does not depend upon the second one. In the case of logical OR, if the first one is 1, it does not depend on the second one. Python supports the following logical operators. In the below table, we explain the works of the logical operators.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| and | The condition will also be true if the expression is true. If the two expressions a and b are the same, then a and b must both be true. |
| or | The condition will be true if one of the phrases is true. If a and b are the two expressions, then an or b must be true if and is true and b is false. |
| not | If an expression **a** is true, then not (a) will be false and vice versa. |

**Program Code:**

Now we give code examples of arithmetic operators in Python. The code is given below -

1. a = 5          # initialize the value of a
2. **print**(Is this statement true?:',a > 3 **and** a < 5)
3. **print**('Any one statement is true?:',a > 3 **or** a < 5)
4. **print**('Each statement is true then return False and vice-versa:',(**not**(a > 3 **and** a < 5)))

**Output:**

Now we give code examples of Bitwise operators in Python. The code is given below -

Is this statement true?: False

Any one statement is true?: True

Each statement is true then return False and vice-versa: True

AD

Membership Operators

The membership of a value inside a Python data structure can be verified using Python membership operators. The result is true if the value is in the data structure; otherwise, it returns false.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| in | If the first operand cannot be found in the second operand, it is evaluated to be true (list, tuple, or dictionary). |
| not in | If the first operand is not present in the second operand, the evaluation is true (list, tuple, or dictionary). |

**Program Code:**

Now we give code examples of Membership operators in Python. The code is given below -

1. x = ["Rose", "Lotus"]
2. **print**(' Is value Present?', "Rose" **in** x)
3. **print**(' Is value not Present?', "Riya" **not** **in** x)

**Output:**

Now we compile the above code in Python, and after successful compilation, we run it. Then the output is given below -

Is value Present? True

Is value not Present? True

AD

Identity Operators

|  |  |
| --- | --- |
| **Operator** | **Description** |
| is | If the references on both sides point to the same object, it is determined to be true. |
| is not | If the references on both sides do not point at the same object, it is determined to be true. |

**Program Code:**

Now we give code examples of Identity operators in Python. The code is given below -

1. a = ["Rose", "Lotus"]
2. b = ["Rose", "Lotus"]
3. c = a
4. **print**(a **is** c)
5. **print**(a **is** **not** c)
6. **print**(a **is** b)
7. **print**(a **is** **not** b)
8. **print**(a == b)
9. **print**(a != b)

**Output:**

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

True

False

False

True

True

False

Operator Precedence

The order in which the operators are examined is crucial to understand since it tells us which operator needs to be considered first. Below is a list of the Python operators' precedence tables.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| \*\* | Overall other operators employed in the expression, the exponent operator is given precedence. |
| ~ + - | the minus, unary plus, and negation. |
| \* / % // | the division of the floor, the modules, the division, and the multiplication. |
| + - | Binary plus, and minus |
| >> << | Left shift. and right shift |
| & | Binary and. |
| ^ | | Binary xor, and or |
| <= < > >= | Comparison operators (less than, less than equal to, greater than, greater then equal to). |
| <> == != | Equality operators. |
| = %= /= //= -= += \*= \*\*= | Assignment operators |
| is is not | Identity operators |
| in not in | Membership operators |
| not or and | Logical operators |