Python Operators

Operators are used to perform operations on variables and values.

Python divides the operators in the following groups:

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

Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Example** |
| + | Addition | x + y |
| - | Subtraction | x - y |
| \* | Multiplication | x \* y |
| / | Division | x / y |
| % | Modulus | x % y |
| \*\* | Exponentiation | x \*\* y |
| // | Floor division | x // y |

Python Assignment Operators

Assignment operators are used to assign values to variables:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same As** |
| = | x = 5 | x = 5 |
| += | x += 3 | x = x + 3 |
| -= | x -= 3 | x = x - 3 |
| \*= | x \*= 3 | x = x \* 3 |
| /= | x /= 3 | x = x / 3 |
| %= | x %= 3 | x = x % 3 |
| //= | x //= 3 | x = x // 3 |
| \*\*= | x \*\*= 3 | x = x \*\* 3 |
| &= | x &= 3 | x = x & 3 |
| |= | x |= 3 | x = x | 3 |
| ^= | x ^= 3 | x = x ^ 3 |
| >>= | x >>= 3 | x = x >> 3 |
| <<= | x <<= 3 | x = x << 3 |

## Python Comparison Operators

Comparison operators are used to compare two values:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Example** |
| == | Equal | x == y |
| != | Not equal | x != y |
| > | Greater than | x > y |
| < | Less than | x < y |
| >= | Greater than or equal to | x >= y |
| <= | Less than or equal to | x <= y |

## Python Logical Operators

Logical operators are used to combine conditional statements:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| and | Returns True if both statements are true | x < 5 and  x < 10 |
| or | Returns True if one of the statements is true | x < 5 or x < 4 |
| not | Reverse the result, returns False if the result is true | not(x < 5 and x < 10) |

## Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| is | Returns True if both variables are the same object | x is y |
| is not | Returns True if both variables are not the same object | x is not y |

## Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| in | Returns True if a sequence with the specified value is present in the object | x in y |
| not in | Returns True if a sequence with the specified value is not present in the object | x not in y |

## Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Description** |
| & | AND | Sets each bit to 1 if both bits are 1 |
| | | OR | Sets each bit to 1 if one of two bits is 1 |
| ^ | XOR | Sets each bit to 1 if only one of two bits is 1 |
| ~ | NOT | Inverts all the bits |
| << | Zero fill left shift | Shift left by pushing zeros in from the right and let the leftmost bits fall off |
| >> | Signed right shift | Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off |

# **Python Membership and Identity Operators | in, not in, is, is not**

**Membership Operators**

Membership operators are operators used to validate the membership of a value. It test for membership in a sequence, such as strings, lists, or tuples.

1. **in operator :**The ‘in’ operator is used to check if a value exists in a sequence or not. Evaluates to true if it finds a variable in the specified sequence and false otherwise.

|  |
| --- |
| # Python program to illustrate  # Finding common member in list  # using 'in' operator  list1=[1,2,3,4,5]  list2=[6,7,8,9]  for item in list1:      if item in list2:          print("overlapping")  else:      print("not overlapping") |

not overlapping

**Same example without using in operator:**

|  |
| --- |
| # Python program to illustrate  # Finding common member in list  # without  using 'in' operator    #  Define a function() that takes two lists  def overlapping(list1,list2):        c=0      d=0      for i in list1:          c+=1      for i in list2:          d+=1      for i in range(0,c):          for j in range(0,d):              if(list1[i]==list2[j]):                  return 1       return 0  list1=[1,2,3,4,5]  list2=[6,7,8,9]  if(overlapping(list1,list2)):      print("overlapping")  else:      print("not overlapping") |

Output:

not overlapping

**not in’ operator-** Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.

|  |
| --- |
| # Python program to illustrate  # not 'in' operator  x = 24  y = 20  list = [10, 20, 30, 40, 50 ];    if ( x not in list ):     print("x is NOT present in given list")  else:     print("x is  present in given list")    if ( y in list ):     print("y is present in given list")  else:     print("y is NOT present in given list") |

**Identity operators**

In Python are used to determine whether a value is of a certain class or type. They are usually used to determine the type of data a certain variable contains.  
There are different identity operators such as

1. **‘is’ operator –** Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.

|  |
| --- |
| # Python program to illustrate the use  # of 'is' identity operator  x = 5  if (type(x) is int):      print("true")  else:      print("false") |

Output:

true

**‘is not’ operator –** Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.

|  |
| --- |
| # Python program to illustrate the  # use of 'is not' identity operator  x = 5.2  if (type(x) is not int):      print("true")  else:      print("false") |

Output:

true