Python - Comparison Operators

Python Comparison Operators

Comparison operators in Python are very important in Python's conditional statements (**if, else** and **elif**) and looping statements (while and for loops). The **comparison operators** also called **relational operators**. Some of the well known operators are "<" stands for less than, and ">" stands for greater than operator.

Python uses two more operators, combining "=" symbol with these two. The "<=" symbol is for less than or equal to operator and the ">=" symbol is for greater than or equal to operator.

Different Comparison Operators in Python

Python has two more comparison operators in the form of "==" and "!=". They are for **is equal to** and **is not equal to** operators. Hence, there are six comparison operators in Python and they are listed below in this table:

<	Less than	a<b
>	Greater than	a>b
<=	Less than or equal to	a<=b
>=	Greater than or equal to	a>=b
==	Is equal to	a==b
!=	Is not equal to	a!=b

Comparison operators are binary in nature, requiring two operands. An expression involving a comparison operator is called a Boolean expression, and always returns either True or False.

Example

</> Open Compiler

```
a=5
b=7
```

```
print (a>b)
print (a<b)
```

It will produce the following **output** –

```
False
True
```

Both the operands may be Python literals, variables or expressions. Since Python supports mixed arithmetic, you can have any number type operands.

Example

The following code demonstrates the use of Python's **comparison operators with integer numbers** –

</> Open Compiler

```
print ("Both operands are integer")
a=5
b=7
print ("a=",a, "b=",b, "a>b is", a>b)
print ("a=",a, "b=",b,"a<b is",a<b)
print ("a=",a, "b=",b,"a==b is",a==b)
print ("a=",a, "b=",b,"a!=b is",a!=b)
```

It will produce the following **output** –

```
Both operands are integer
a= 5 b= 7 a>b is False
a= 5 b= 7 a<b is True
a= 5 b= 7 a==b is False
a= 5 b= 7 a!=b is True
```

Learn **Python** in-depth with real-world projects through our **Python certification course**. Enroll and become a certified expert to boost your career.

Comparison of Float Number

In the following example, an integer and a float operand are compared.

Example

</> Open Compiler

```
print ("comparison of int and float")
a=10
b=10.0
print ("a=",a, "b=",b, "a>b is", a>b)
print ("a=",a, "b=",b,"a<b is",a<b)
print ("a=",a, "b=",b,"a==b is",a==b)
print ("a=",a, "b=",b,"a!=b is",a!=b)
```

It will produce the following **output** –

```
comparison of int and float
a= 10 b= 10.0 a>b is False
a= 10 b= 10.0 a<b is False
a= 10 b= 10.0 a==b is True
a= 10 b= 10.0 a!=b is False
```

Comparison of Complex umbers

Although complex object is a number data type in Python, its behavior is different from others. Python doesn't support < and > operators, however it does support equality (==) and inequality (!=) operators.

Example

</> Open Compiler

```
print ("comparison of complex numbers")
a=10+1j
b=10.-1j
print ("a=",a, "b=",b,"a==b is",a==b)
print ("a=",a, "b=",b,"a!=b is",a!=b)
```

It will produce the following **output** –

```
comparison of complex numbers
a= (10+1j) b= (10-1j) a==b is False
a= (10+1j) b= (10-1j) a!=b is True
```

You get a TypeError with less than or greater than operators.

Example

```
print ("comparison of complex numbers")
a=10+1j
b=10.-1j
print ("a=",a, "b=",b,"a<b is",a<b)
print ("a=",a, "b=",b,"a>b is",a>b)
```

It will produce the following **output** –

```
comparison of complex numbers
Traceback (most recent call last):
  File "C:\Users\mlath\examples\example.py", line 5, in <module>
    print ("a=",a, "b=",b,"a<b is",a<b)
                                ^^^
TypeError: '<' not supported between instances of 'complex' and
'complex
```

Comparison of Booleans

Boolean objects in Python are really integers: True is 1 and False is 0. In fact, Python treats any non-zero number as True. In Python, comparison of Boolean objects is possible. "False < True" is True!

Example

```
print ("comparison of Booleans")
a=True
```

```
b=False
print ("a=",a, "b=",b,"a<b is",a<b)
print ("a=",a, "b=",b,"a>b is",a>b)
print ("a=",a, "b=",b,"a==b is",a==b)
print ("a=",a, "b=",b,"a!=b is",a!=b)
```

It will produce the following **output** –

```
comparison of Booleans
a= True b= False a<b is False
a= True b= False a>b is True
a= True b= False a==b is False
a= True b= False a!=b is True
```

Comparison of Sequence Types

In Python, comparison of only similar sequence objects can be performed. A string object is comparable with another string only. A list cannot be compared with a tuple, even if both have same items.

Example

</> Open Compiler

```
print ("comparison of different sequence types")
a=(1,2,3)
b=[1,2,3]
print ("a=",a, "b=",b,"a<b is",a<b)
```

It will produce the following **output** –

```
comparison of different sequence types
Traceback (most recent call last):
  File "C:\Users\mlath\examples\example.py", line 5, in <module>
    print ("a=",a, "b=",b,"a<b is",a<b)
                                ^^^
TypeError: '<' not supported between instances of 'tuple' and 'list'
```

Sequence objects are compared by lexicographical ordering mechanism. The comparison starts from item at 0th index. If they are equal, comparison moves to next index till the items at certain index happen to be not equal, or one of the sequences is exhausted. If one sequence is an initial sub-sequence of the other, the shorter sequence is the smaller (lesser) one.

Which of the operands is greater depends on the difference in values of items at the index where they are unequal. For example, 'BAT'>'BAR' is True, as T comes after R in Unicode order.

If all items of two sequences compare equal, the sequences are considered equal.

Example

</> Open Compiler

```
print ("comparison of strings")
a='BAT'
b='BALL'
print ("a=",a, "b=",b,"a<b is",a<b)
print ("a=",a, "b=",b,"a>b is",a>b)
print ("a=",a, "b=",b,"a==b is",a==b)
print ("a=",a, "b=",b,"a!=b is",a!=b)
```

It will produce the following **output** –

```
comparison of strings
a= BAT b= BALL a<b is False
a= BAT b= BALL a>b is True
a= BAT b= BALL a==b is False
a= BAT b= BALL a!=b is True
```

In the following example, two tuple objects are compared –

Example

</> Open Compiler

```
print ("comparison of tuples")
a=(1,2,4)
b=(1,2,3)
```

```
print ("a=",a, "b=",b,"a<b is",a<b)
print ("a=",a, "b=",b,"a>b is",a>b)
print ("a=",a, "b=",b,"a==b is",a==b)
print ("a=",a, "b=",b,"a!=b is",a!=b)
```

It will produce the following **output** –

```
a= (1, 2, 4) b= (1, 2, 3) a<b is False
a= (1, 2, 4) b= (1, 2, 3) a>b is True
a= (1, 2, 4) b= (1, 2, 3) a==b is False
a= (1, 2, 4) b= (1, 2, 3) a!=b is True
```

Comparison of Dictionary Objects

The use of "<" and ">" operators for Python's dictionary is not defined. In case of these operands, TypeError: '<' not supported between instances of 'dict' and 'dict' is reported.

Equality comparison checks if the length of both the dict items is same. Length of dictionary is the number of key-value pairs in it.

Python dictionaries are simply compared by length. The dictionary with fewer elements is considered less than a dictionary with more elements.

Example

</> Open Compiler

```
print ("comparison of dictionary objects")
a={1:1,2:2}
b={2:2, 1:1, 3:3}
print ("a=",a, "b=",b,"a==b is",a==b)
print ("a=",a, "b=",b,"a!=b is",a!=b)
```

It will produce the following **output** –

```
comparison of dictionary objects
a= {1: 1, 2: 2} b= {2: 2, 1: 1, 3: 3} a==b is False
a= {1: 1, 2: 2} b= {2: 2, 1: 1, 3: 3} a!=b is True
```