Precedence and Associativity of Operators in Python

Precedence of Python Operators

The combination of values, variables, operators, and function calls is termed as an expression. The Python interpreter can evaluate a valid expression.

For example:

```
>>> 5 - 7
-2
```

Here `5 - 7` is an expression. There can be more than one operator in an expression.

To evaluate these types of expressions there is a rule of precedence in Python. It guides the order in which these operations are carried out.

For example, multiplication has higher precedence than subtraction.

```
# Multiplication has higher precedence
# than subtraction
>>> 10 - 4 * 2
2
```

But we can change this order using parentheses `()` as it has higher precedence than multiplication.

```
# Parentheses () has higher precedence
>>> (10 - 4) * 2
12
```

The operator precedence in Python is listed in the following table. It is in descending order (upper group has higher precedence than the lower ones).

Operators	Meaning
`()`	Parentheses
`**`	Exponent
`+x, -x, ~x`	Unary plus, Unary minus, Bitwise NOT
`*, /, //, %`	Multiplication, Division, Floor division, Modulus
`+, -`	Addition, Subtraction
`<<, >>`	Bitwise shift operators
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`==, !=, >, >=, <, <=, is, is not, in, not in`	Comparisons, Identity, Membership operators
`not`	Logical NOT
`and`	Logical AND
`or`	Logical OR

Let's look at some examples:

Suppose we're constructing an if...else block which runs `if` when *lunch* is either `fruit` or `sandwich` and only if *money* is more than or equal to `2`.

```
# Precedence of or & and
meal = "fruit"

money = 0

if meal == "fruit" or meal == "sandwich" and money >= 2:
    print("Lunch being delivered")
else:
    print("Can't deliver lunch")
```

Output

```
Lunch being delivered
```

This program runs `if` block even when *money* is `0`. It does not give us the desired output since the precedence of `and` is higher than `or`.

We can get the desired output by using parenthesis `()` in the following way:

```
# Precedence of or & and
meal = "fruit"

money = 0

if (meal == "fruit" or meal == "sandwich") and money >= 2:
    print("Lunch being delivered")
else:
    print("Can't deliver lunch")
```

Output

```
Can't deliver lunch
```

Associativity of Python Operators

We can see in the above table that more than one operator exists in the same group. These operators have the same precedence.

When two operators have the same precedence, associativity helps to determine the order of operations.

Associativity is the order in which an expression is evaluated that has multiple operators of the same precedence. Almost all the operators have left-to-right associativity.

For example, multiplication and floor division have the same precedence. Hence, if both of them are present in an expression, the left one is evaluated first.

```
# Left-right associativity
# Output: 3
print(5 * 2 // 3)

# Shows left-right associativity
# Output: 0
print(5 * (2 // 3))
```

Output

```
3
0
```

Note: Exponent operator `**` has right-to-left associativity in Python.

```
# Shows the right-left associativity of **
# Output: 512, Since 2**(3**2) = 2**9
print(2 ** 3 ** 2)

# If 2 needs to be exponated fisrt, need to use ()
# Output: 64
print((2 ** 3) ** 2)
```

We can see that `2 ** 3 ** 2` is equivalent to `2 ** (3 ** 2)`.

Non associative operators

Some operators like assignment operators and comparison operators do not have associativity in Python. There are separate rules for sequences of this kind of operator and cannot be expressed as associativity.

For example, `x < y < z` neither means `(x < y) < z` nor `x < (y < z)`. `x < y < z` is equivalent to `x < y and y < z`, and is evaluated from left-to-right.

Furthermore, while chaining of assignments like `x = y = z = 1` is perfectly valid, `x = y = z+= 2` will result in error.

```
# Initialize x, y, z
x = y = z = 1

# Expression is invalid
# (Non-associative operators)
# SyntaxError: invalid syntax

x = y = z+= 2
```

Output

```
  File "<string>", line 8
    x = y = z+= 2
              ^
SyntaxError: invalid syntax
```

Table of Contents