Department of Computer Science and Engineering
PES University, Bangalore, India

Lecture Notes
Python for Computational Problem Solving
UE23CS151A

Lecture #19

Precedence and Associativity of operators

By,
Prof. Sindhu R Pai,
Anchor, PCPS - 2023
Assistant Professor
Dept. of CSE, PESU

Verified by,
PCPS Team - 2023

Many Thanks to
Dr. Shylaja S S (Director, CCBD and CDSAML Research Centers, Former Chairperson, CSE, PES University)
Prof. Chitra G M (Asst. Prof, Dept. of CSE, PCPS Anchor – 2022)

Precedence and Associativity of Operators

Precedence of Operators:

When we have an expression that contains more than one operator and are different operators, we need to look at the **precedence table of operators to decide in which order the evaluation of an expression** must happen. Operator precedence guarantees a **consistent interpretation of expressions.**

Let us consider the following examples of Arithmetic Expression -

- **Evaluation of 4 + 3 * 5**

There are two possible ways in which it can be evaluated 4 + 3 * 5 → 4 + 15 → 19 Or 4 + 3 * 5 → 7 * 5 → 35. To ensure that the results will be the same at all times, we use the rules of operator precedence.

- **Evaluation of 4 + 2 ** 5 // 10**

4 + 2 ** 5 // 10 → 4 + 32 // 10 → 4 + 3 → 7

Note: Operator precedence table for Arithmetic Operators has ** at the top. Means it has highest priority. Binary + and – has the lowest priority.

Priority	Operators List
1	**
2	Unary + and -
3	*, /, //, %
4	Binary + and -

Likewise every operator in an expression has its precedence set as shown in the table on Page #5.

It is a good programming practice to use parentheses to know about your precedence of operators in an expression.

Consider 4 + 2 ** 2 // 10. If we want, 4+2 to happen first and then power to 2, please use parentheses as shown here => ((4 + 2) ** 2)//10

Observe the below outputs from python interpreter.

```
>>> 4 + 2 ** 2 // 10
4
>>> (4 + 2) ** 2 // 10
3
>>> ((4 + 2) ** 2) // 10
3
```

Associativity of Operators:

If **more than one operator with the same level of precedence exists**, rules of Associativity indicate the order in which an expression is evaluated.

Let us consider the following examples of Arithmetic Expression –

- **Evaluation of 2 + 3 + 4**

  There are two possible ways in which it can be evaluated. (2 + 3) + 4 or 2 + (3 + 4) ➝ 9.

Addition operator follows the Associative Law. Hence the order of evaluation doesn't matter.

- **Evaluation of 8 - 4 – 2**

  This can be evaluated as (8 - 4) – 2 -> 4 – 2 -> 2 or 8 – (4 - 2) -> 8 – 2 -> 6.

Subtraction operator doesn't follow the Associative Law. Hence the Associativity table makes sense. As the subtraction is Left to right, the result of above expression is 2.

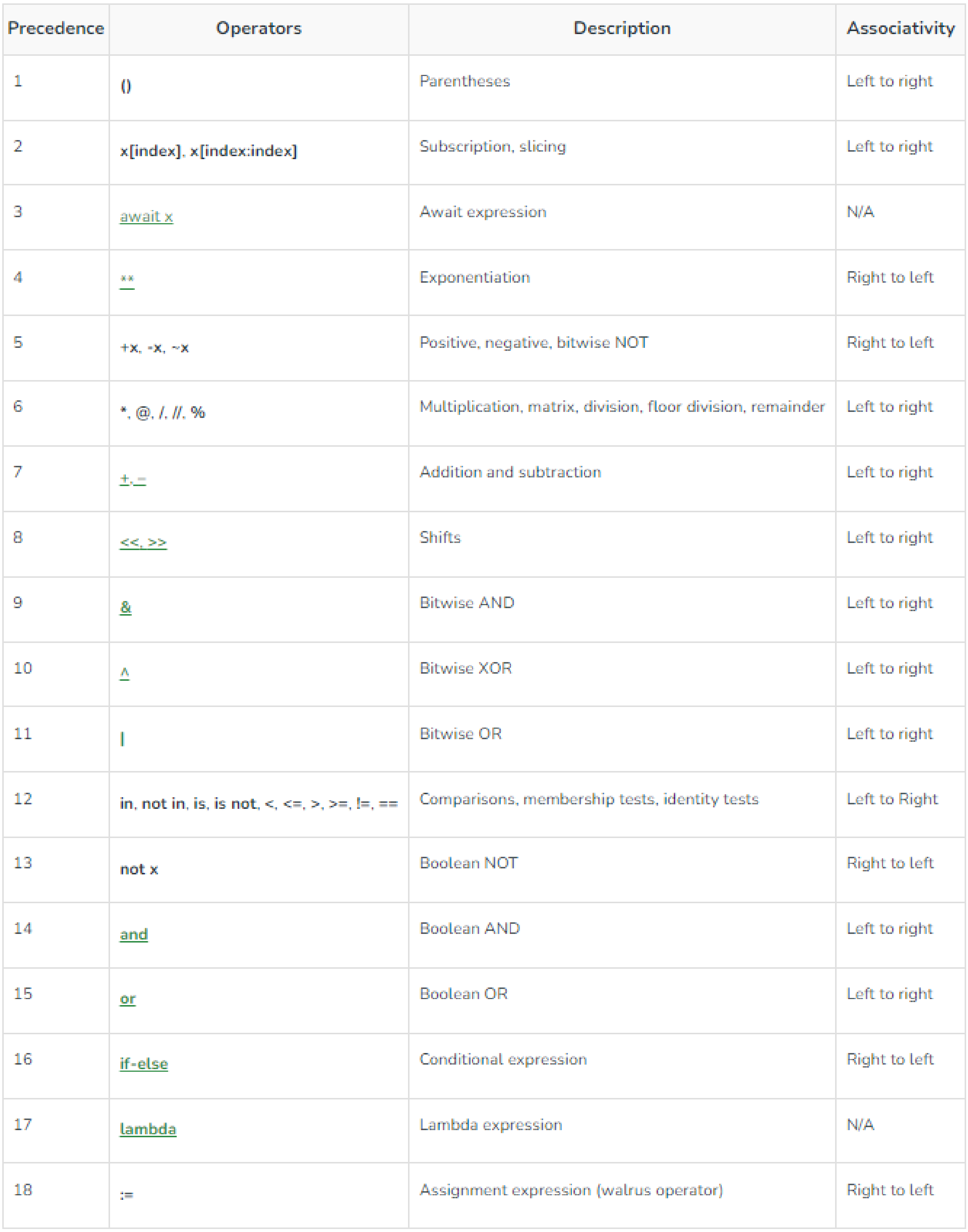
Priority	Operators List	Associativity
1	**	Right to Left
2	Unary + and -	Left to Right
3	*, /, //, %	Left to Right
4	Binary + and -	Left to Right

Division and Power operator also do not follow the Associative law. Try it!!

```
8 / 4 / 2       #result is 1.0         (8 / 4) / 2     #result is 4.0
2**3**2         #result is 512         (2**3)** 2      #result is 64
```

We can use parentheses – (and) to prioritize the operators in an expression as per our wish. But Associativity within the parenthesized expression remains the same.

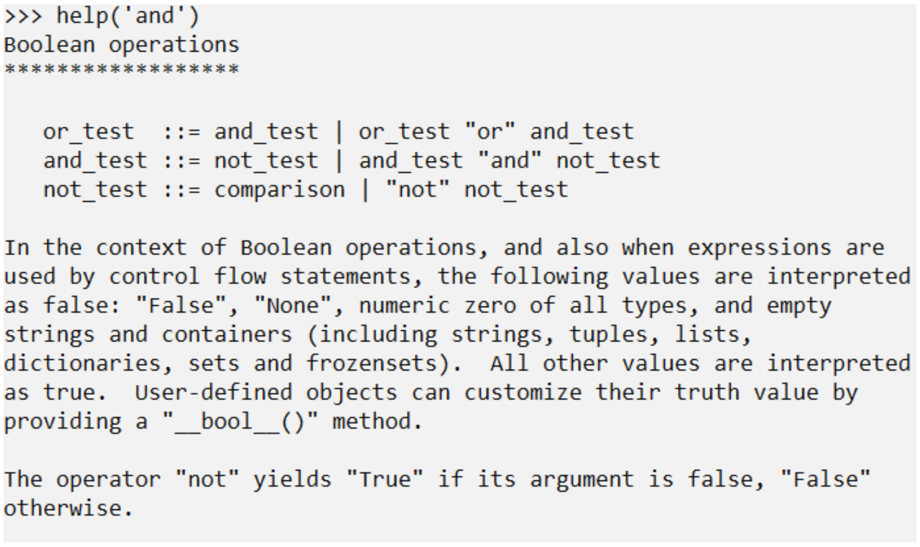
Precedence and Associativity table for all operators in python



Precedence	Operators	Description	Associativity
1	()	Parentheses	Left to right
2	x[index], x[index:index]	Subscription, slicing	Left to right
3	await x	Await expression	N/A
4	**	Exponentiation	Right to left
5	+x, -x, ~x	Positive, negative, bitwise NOT	Right to left
6	*, @, /, //, %	Multiplication, matrix, division, floor division, remainder	Left to right
7	+, -	Addition and subtraction	Left to right
8	<<, >>	Shifts	Left to right
9	&	Bitwise AND	Left to right
10	^	Bitwise XOR	Left to right
11	|	Bitwise OR	Left to right
12	in, not in, is, is not, <, <=, >, >=, !=, ==	Comparisons, membership tests, identity tests	Left to Right
13	not x	Boolean NOT	Right to left
14	and	Boolean AND	Left to right
15	or	Boolean OR	Left to right
16	if-else	Conditional expression	Right to left
17	lambda	Lambda expression	N/A
18	:=	Assignment expression (walrus operator)	Right to left

Ref: [Precedence and Associativity of Operators in Python - GeeksforGeeks]

Note: There are so many operators and each of them is specific for its operation. If we have to know in detail about any operator, please use help() on the operator within quotes. Screenshot attached.

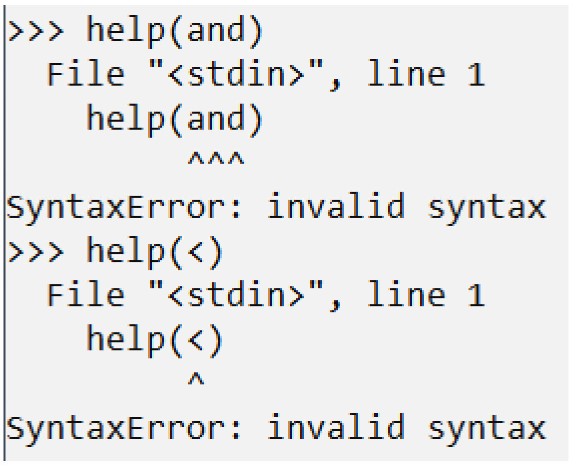


```
>>> help('and')
Boolean operations
******************

   or_test  ::= and_test | or_test "or" and_test
   and_test ::= not_test | and_test "and" not_test
   not_test ::= comparison | "not" not_test

In the context of Boolean operations, and also when expressions are
used by control flow statements, the following values are interpreted
as false: "False", "None", numeric zero of all types, and empty
strings and containers (including strings, tuples, lists,
dictionaries, sets and frozensets).  All other values are interpreted
as true.  User-defined objects can customize their truth value by
providing a "__bool__()" method.

The operator "not" yields "True" if its argument is false, "False"
otherwise.
```



```
>>> help(and)
  File "<stdin>", line 1
    help(and)
         ^^^
SyntaxError: invalid syntax
>>> help(<)
  File "<stdin>", line 1
    help(<)
         ^
SyntaxError: invalid syntax
```

- END -