IIT Madras
BSc Degree

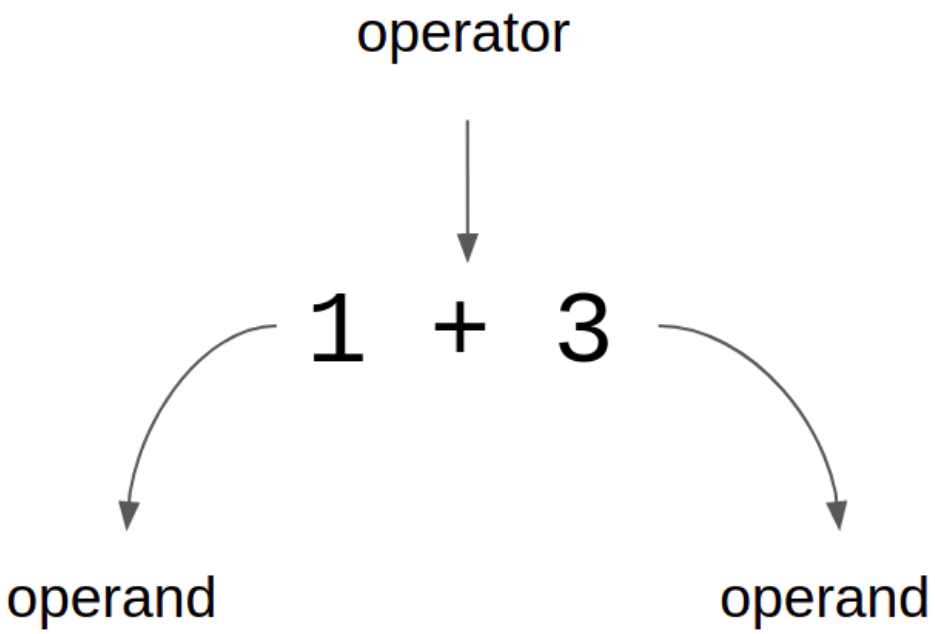
Home Lesson-1.3

Lesson-1.2

Operators

Arithmetic

The anatomy of an operation is given below:



The following table gives the symbols for arithmetic operators and the operations that they correspond to:

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
//	Floor division
%	Modulus
**	Exponentiation

All the operators in the above table are binary, i.e., they operate on two operands. Let us now take a look at each operator:

```
>>> 10 + 5
15
>>> 10 - 5
5
>>> 10 * 5
50
>>> 10 / 5
2.0
>>> 10 // 5
2
>>> 10 % 5
0
>>> 10 ** 5
100000
```

The last three operators might be new. In more familiar terms, these are the mathematical operations that they correspond to:

- `//` is called the floor division operator. `x // y` gives the quotient when `x` is divided by `y`. For example, `8 // 3` is `2`.
- `%` is called the modulus operator. `x % y` gives the remainder when `x` is divided by `y`. For example, `10 % 3` is `1`.
- `**` is called the exponentiation operator. `x ** y` returns x^y.

`/` and `//` are two different operators. `/` gives the complete result of division, while `//` returns the quotient. For example, `5 / 2` results in `2.5` while `5 // 2` gives `2`. There are two more arithmetic operators of interest to us, unary plus and unary minus. These are the `+` and `-` signs. Unlike the operators that we have seen so far, these two are unary operators, i.e., they operate on one operand. For example:

```
>>> - 2
-2
>>> + 2
2
```

It is important to note that the symbols for plus and minus operators are the same as the ones for addition and subtraction. The context determines the nature of the operator:

```
>>> - 1    # unary minus
-1
>>> 1 - 1  # subtraction operator
```

Sometimes both of them could come together in the same expression:

```
>>> 1 - - 1
2
>>> # The minus on the left is subtraction
>>> # The minus on the right is unary minus
```

In all the operations that we have seen so far, the operands have been literals. In general, the operands can also be variables:

```
>>> x = 1
>>> y = x * 5
>>> print(x, y)
1 5
```

Relational

The following table gives the symbols for relational operators and the operations that they correspond to:

Operator	Operation
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
==	double equal to
!=	not equal to

All the operators in the above table are binary. Let us now take a look at each of them:

```
>>> 10 > 5
True
>>> 10 < 5
False
>>> 10 >= 5
True
>>> 10 <= 5
False
>>> 10 == 5
False
>>> 10 != 5
True
```

Relational operators are also called comparison operators. The result of any comparison operation is a boolean value: `True` or `False`. The result of a comparison operation can be assigned to a variable:

```
>>> x = 10
>>> y = 15
>>> z = y > x
>>> print(z)
True
```

The `==` symbol corresponds to the equality operator and should not be confused with `=`, the assignment operator.

Logical

The following table gives the logical operators and the operations that they correspond to:

Operator	Operation
not	negation
and	logical conjunction
or	logical disjunction

`and` and `or` are binary operators; `not` is a unary operator. Let us now take a look at each of them:

```
>>> True and False
False
>>> True or False
True
>>> x = False
>>> y = not x
>>> print(y)
True
```

The use of parenthesis after `not` is optional. For example:

```
>>> x = True
>>> not x
False
>>> x = False
>>> not(x)
True
```

Convention

Consider the following lines of code:

```
>>> print(1 + 2)
3
>>> print(1+2)
3
```

Both lines 1 and 3 give the same output. Line-1 has a space before and after the `+` operator, while line-3 doesn't. Both ways are syntactically correct. In this course, we will be following the first convention: there is always a space separating the operator from the operands. This is also true for the `=` operator.

```
>>> x = 2 # We will follow this
>>> x=2   # We will NOT follow this
# But both conventions are valid
```

Expressions

An expression is some combination of literals, variables and operators. For example, the following are expressions:

- `1 + 4 / 4 ** 0`
- `x / y + z * 2.0`
- `3 > 4 and 1 < 10`
- `not True and False`

Each expression evaluates to some value. This value has a type. In the above examples, the first two expressions result in a `float`, while the next two expressions result in a `bool`. In the next few sections, we shall study two types of expressions:

- Arithmetic: an expression whose type is either `int` or `float`
- Boolean: an expression whose type is `bool`

Type of Expressions

Arithmetic Expressions

Let us now look at the `type` of simple arithmetic operations. In mathematics, the result of adding two integers is another integer. Is this true in the case of Python? First, let us execute the following statement in the interpreter and see what we get:

```
>>> 1 + 2
3
```

The way to check the type of this expression is to use the `type()` function. For example, we have:

```
>>> 1 + 2
3
>>> type(1 + 2)
<class 'int'>
```

So far the interpreter's behaviour conforms to our intuition. Let us now change this code slightly:

```
>>> 1.0 + 2
3.0
>>> type(1.0 + 2)
<class 'float'>
```

We see that the result is `3.0` which is of type `float`. The conclusion is that `float` is more dominant than `int` as far as the addition operation is concerned. What about other operations? Let us check with the help of the following examples:

```
>>> type(7.0 * 5)
<class 'float'>
>>> type(7.0 / 5)
<class 'float'>
>>> type(7.0 // 5)
<class 'float'>
>>> type(7.0 ** 5)
<class 'float'>
>>> type(7.0 % 5)
<class 'float'>
```

All the operations result in a `float`. From this we see that `float` is more dominant than `int`, irrespective of the operator involved.

Boolean Expressions

Expressions that involve a relational operator will result in a `bool`. For example:

```
>>> 2 > 1
True
>>> type(2 > 1)
<class 'bool'>
```

Expressions that involve logical operators will naturally result in a `bool`. For example:

```
>>> True and False
False
>>> type(True and False)
<class 'bool'>
```

One way to analyze the outcome of boolean expressions that involve variables is to exhaustively list down the different combinations of values that variables can take and evaluate the expression for each such combination. For example, assume that `X` and `Y` are two boolean variables. Now, consider the following expression:

```
>>> X or Y
```

We can take the help of a concept called **truth table** to analyze the outcomes:

`X`	`Y`	`X or Y`
`True`	`True`	`True`
`True`	`False`	`True`
`False`	`True`	`True`
`False`	`False`	`False`