Operators

- Arithmetic operators: `+`, `-`, `*`, `/`, `//`, `%`, `**`
- Assignment operators: `=`, `+=`, `-=`, `*=`, `/=`, `//=`, `%=`, `**=`
- Comparison operators: `==`, `!=`, `>`, `<`, `>=`, `<=`
- Logical operators: `and`, `or`, `not`
- Identity operators: `is`, `is not`
- Membership operators: `in`, `not in`
- ~~Bitwise operators~~: `&`, `|`, `^`, `>>`, `<<`, `~`

Arithmetic Operators

There are a total of **seven** arithmetic operators in Python. Here are the four basic operators: addition `+`, subtraction `-`, multiplication `*`, division `/`. There are three more arithmetic operators which might be unfamiliar to you: modulus `%`, exponentiation `**`, and the floor division `//`.

Modulus Operator

Modulus operator `%` gets the remainder of a division.

$$5 \bmod 2 \equiv 1$$

$$\begin{array}{r} 2 \\ 2\overline{)5} \\ \underline{-4} \\ 1 \end{array} \quad \leftarrow (\text{Remainder is } 1)$$

```
>>> 5 % 2
1
```

Exponentiation Operator

Exponentiation operator `**` gets the n^{th} power of a value.

$$2^2 = 4$$

```
>>> 2**2
4
```

Floor Division

Floor division operator `//` gets the *quotient* of a division. It is a division in which the *remainder* is *discarded*.

$$\left\lfloor \frac{5}{2} \right\rfloor \equiv 2$$

$$\begin{array}{r} \mathbf{2} \\ 2\overline{)5} \\ -\,4 \\ \hline 1 \end{array} \quad \leftarrow (\text{Quotient is } 2)$$

```
>>> 5 // 2
2
```

Assigment Operators

Assignment operators are used to assign values to variables. There are a total of **eight** assignment operators in Python: `=`, `+=`, `-=`, `*=`, `/=`, `//=`, `%=`, `**=`. The single equal sign assignment operator `=` is the basic operator that simply assigns a value.

Single Assignment

```
>>> number = 5
>>> number
5
```

Multiple Assignment

```
>>> first_var, second_var = 0, 1
>>> first_var
0
>>> second_var
1
```

Swapping Values with Multiple Assignment

```
>>> first_var, second_var = 0, 1
>>> first_var, second_var = second_var, first_var
>>> first_var
1
>>> second_var
0
```

Augmented Assignment Operators

An assignment operator combined with arithmetic operators.

```
>>> number = 5
>>> number += 3 # number = 5 + 3
>>> number
8
>>> number -= 3 # number = 8 - 3
>>> number
5
>>> number *= 3 # number = 5 * 3
>>> number
15
>>> number /= 2 # number = 15 / 2
>>> number
7
>>> number %= 2 # number = 7 % 2
>>> number
1
>>> number += 9 # number = 1 + 9
>>> number //= 3 # number = 10 // 3
>>> number
3
>>> number **= 2 # number = 3 ** 2
>>> number
9
```

As you can see from the above example, the assignment operators that are augmented with the arithmetic operators carry out the arithmetic operations implicitly and then overwrites the existing variable with the resulting value.

If we take line #2 for example, the `number` variable initially has the value `5`, but the augmented operator `+=` tells Python to evaluate `number + 3` first and then overwrite the `number` variable with the resulting value of `8`.

Comparison Operators

Comparison operators are used in comparing two values.

There are a total of **six** comparison operators in Python: **equal-to operator:** `==`, **not-equal-to operator:** `!=`, **greater-than operator:** `>`, **less-than operator:** `<`, **greater-than-or-equal-to operator:** `>=`, **less-than-or-equal-to operator:** `<=`.

Logical Operators

Logical operators are used for combining conditional expressions.

There are a total of three logical operators: **and operator:** `and`, **or operator:** `or`, and the **not operator:** `not`.

```
# the `and` operator
>>> first_number = 5
>>> second_number = 10
>>> first_number > 0 and second_number < 20
True

# the `or` operator
>>> first_number > 5 or second_number < 20
True

# the `not` operator
>>> not first_number == second_number
True
```

Line #4's conditional expression `first_number > 5 and second_number < 20` is evaluated to `True` only if **both** sides (left-side and right-side of the `and` operator) of the expression evaluate to `True`.

Line #8's conditional expression `first_number > 5 or second_number < 20` should evaluate to `True`, only if **either** side of the `or` operator evaluates to `True`. In this case, `second_number < 20` evaluated to `True`; therefore, the entire expression on line #8 is evaluated to `True`.

Line #12's conditional expression `first_number == second_number` should evalute to `True`, only if the right-side of the expression `first_number == second_number` evaluates to `False`. In this case, `first_number` is not equal to `second_number`; therefore `first_number == second_number` will evaluate to `False`; therefore the `not` operator will evalute the entire expression at line #12 to `True`. Essentially, the `not` operator reverses the boolean value that is given to it.

Identity Operators

The identify operators are used for checking if two objects are referencing the same memory location on the system. In other words, they're used to compare if two **objects** have the same **identity**. The identity operators are **not** used for comparing whether or not the two objects have the same value.

There are a total of two identity operators: **is operator:** `is` and the **is-not operator:** `is not`.

> Identity operators are used for checking if **two objects** have the same **identity**.

How to Retrieve the Identity of an Object

Given a list containing a various elements, we can retrieve the identity of the list by using a built-in Python function called `id()`.

```
>>> L = ["Hello", 2.5, "Good", 1]
>>> id(L)
57448264L
```

In the above example, we created a list with various elements, and called the `id()` function with the list as its argument. We then get an output of `57448264L`. The identity that gets returned from the `id()` function will be different depending on the system that the Python program is run.

`is` Operator

The `is` identity operator will tell Python to evaluate an identity expression within which it resides to `True` if both sides of the operator have the same identity.

```
>>> L1 = ["Apple", "Banana"]
>>> L2 = ["Apple", "Banana"]
>>> L1 is L2
False
```

`is not` Operator

The `is not` identity operator will tell Python to evaluate an identity expression within which it resides to `True` if both sides of the operator have different identities.

```
>>> L1 = ["Apple", "Banana"]
>>> L2 = ["Apple", "Banana"]
>>> L1 is not L2
True
```

In Contrast to the `==` and `!=` Comparison Operators

```
>>> L1 = ["Apple", "Banana"]
>>> L2 = ["Apple", "Banana"]
>>> L1 is L2
False
>>> L1 is not L2
True
>>> L1 == L2
True
>>> L1 != L2
False
```

As you can see from the above example, the `==` and `!=` comparison operators compares the values within the two lists. `L1` does have the same values as `L2` does, so using `==` for comparison will evalute to `True`, and since `L1` and `L2` have the same values, using `!=` will evalute to `False`.

Membership Operators

The membership operators are used for checking if an object exists in a sequence of objects.

There are a total of **two** membership operators in Python: **in operator:** `in` and the **not-in operator:** `not in`.

```
>>> L = ["Apple", "Banana"]
>>> "Banana" in L
True
>>> "Peach" not in L
True
>>> "Watermelon" in L
False
>>> "Apple" not in L
False
```

As you can see from the example above, the `"Banana"` object (a string literal) exists in the list `L`; therefore, the expression `"Banana" in L` will evaluate to `True`. Also, since the object `"Peach"` (another string literal) does not exist in the list `L`, the expression `"Peach" not in L` will evaluate to `True`.