

1. Operators

Operators are used to perform operations on variables and values. In the example below, we use the + operator to add together two values:

```
print(10 + 5)
```

Although the + operator is often used to add together two values, like in the example above, it can also be used to add together a variable and a value, or two variables:

```
sum1 = 100 + 50    # 150 (100 + 50)
sum2 = sum1 + 250   # 400 (150 + 250)
sum3 = sum2 + sum2  # 800 (400 + 400)
```

2. Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

You might have noticed that there are two division operators:

- / - Division (returns a float)
- // - Floor division (returns an integer) - It rounds DOWN to the nearest integer:

3. Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
=	x = 3	x = x 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3
:=	print(x := 3)	x = 3; print(x)

4. Comparison Operators

Comparison operators are used to compare values with other values:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

The result of the comparison operators returns true / false based on the comparison.

Python also allows to chain the comparison operators like below:

X > 50 and X < 100

5. Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	<code>x < 5 and x < 10</code>
or	Returns True if one of the statements is true	<code>x < 5 or x < 4</code>
not	Reverse the result, returns False if the result is true	<code>not(x < 5 and x < 10)</code>

Example:

* Test if a number is less than 5 **or** greater than 10 → `x < 5 or x > 10`

* Reverse the result with **not** → `not(x > 3 and x < 10)`

6. Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	<code>x is y</code>
is not	Returns True if both variables are not the same object	<code>x is not y</code>

Example:

- The **is** operator returns **True** if both variables point to the same object

```
x = ["ByteUp", "Academy"]
y = ["ByteUp", "Academy"]
z = x
print(x is z)
print(x is y)
```

- The **is not** operator returns **True** if both variables do not point to the same object:

```
x = ["ByteUp", "Academy"]
y = ["ByteUp", "Academy"]
print(x is not y)
```

So, what's the difference between `is` and `==` ?

- **`is`** → Checks if both variables point to the same object in memory
- **`==`** → Checks if the values of both variables are equal

Example:

```
x = [1, 2, 3]
y = [1, 2, 3]
print(x == y)
print(x is y)
```

7. Membership Operators

Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
<code>in</code>	Returns True if a sequence with the specified value is present in the object	<code>x in y</code>
<code>not in</code>	Returns True if a sequence with the specified value is not present in the object	<code>x not in y</code>

Example:

```
academy = ["learn", "with", "byteup"]
print("learn" in academy)
```

Note that this Membership operators works a little different way in strings like below:

```
text = "ByteUp Academy"
print("B" in text)
print("Up" in text)
print("x" not in text)
```

7. Bitwise Operators

Bitwise operators are used to compare binary numbers:

Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	x & y
	OR	Sets each bit to 1 if one of two bits is 1	x y
^	XOR	Sets each bit to 1 if only one of two bits is 1	x ^ y
~	NOT	Inverts all the bits	~x
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off	x << 2
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	x >> 2

Example: The & operator compares each bit and set it to 1 if both are 1, otherwise it is set to 0.

```
print(6 & 3)
```

The binary representation of 6 is 0110

The binary representation of 3 is 0011

Then the & operator compares the bits and returns 0010, which is 2 in decimal.

7. Operator Precedence

Operator precedence describes the order in which operations are performed in Python. Below table described the precedence order from top.

Order	Operator	Description
1	()	Parentheses
2	**	Exponentiation
3	+x -x ~x	Unary plus, unary minus, and bitwise NOT
4	* / // %	Multiplication, division, floor division, and modulus

5	+ -	Addition and subtraction
6	<< >>	Bitwise left and right shifts
7	&	Bitwise AND
8	^	Bitwise XOR
9	 	Bitwise OR
10	== != > >= < <= is is not in not in	Comparisons, identity, and membership operators
11	not	Logical NOT
12	and	AND
13	or	OR

- If two operators have the same precedence, the expression is evaluated from left to right.

Some key examples:

1.

```
print(5 + 4 - 7 + 3)
```

Addition + and subtraction - has the same precedence, and therefore we evaluate the expression from left to right.

2.

```
print((6 + 3) - (6 + 3))
```

Parentheses has the highest precedence, meaning that expressions inside parentheses must be evaluated first.

3.

```
print(100 + 5 * 3)
```

Multiplication * has higher precedence than addition +, and therefore multiplications are evaluated before additions.
