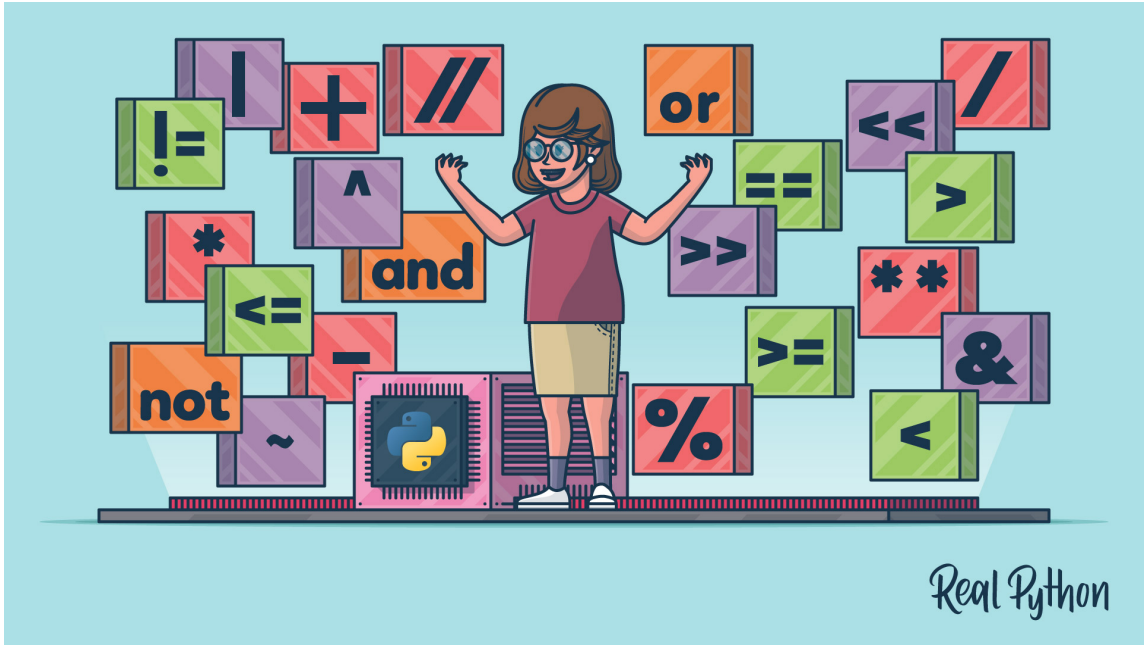
Operators and Expressions in Python

Table of Contents



Arithmetic Operators in Python

Operator	Type	Operation	Sample Expression	Result
`+`	Unary	Positive	`+a`	`a` without any transformation since this is simply a complement to negation
`+`	Binary	Addition	`a + b`	The arithmetic sum of `a` and `b`
`-`	Unary	Negation	`-a`	The value of `a` but with the opposite sign
`-`	Binary	Subtraction	`a - b`	`b` subtracted from `a`
`*`	Binary	Multiplication	`a * b`	The product of `a` and `b`
`/`	Binary	Division	`a / b`	The quotient of `a` divided by `b`, expressed as a float
`%`	Binary	Modulo	`a % b`	The remainder of `a` divided by `b`
`//`	Binary	Floor division or integer division	`a // b`	The quotient of `a` divided by `b`, rounded to the next smallest whole number
`**`	Binary	Exponentiation	`a**b`	`a` raised to the power of `b`

Comparison Operators in Python

Operator	Operation	Sample Expression	Result
`==`	Equal to	`a == b`	• `True` if the value of `a` is equal to the value of `b` • `False` otherwise
`!=`	Not equal to	`a != b`	• `True` if `a` is not equal to `b` • `False` otherwise
`<`	Less than	`a < b`	• `True` if `a` is less than `b` • `False` otherwise
`<=`	Less than or equal to	`a <= b`	• `True` if `a` is less than or equal to `b` • `False` otherwise
`>`	Greater than	`a > b`	• `True` if `a` is greater than `b` • `False` otherwise
`>=`	Greater than or equal to	`a >= b`	• `True` if `a` is greater than or equal to `b` • `False` otherwise

Boolean Operators in Python

Operator	Sample Expression	Result
`and`	`x and y`	• `True` if both `x` and `y` are `True` • `False` otherwise
`or`	`x or y`	• `True` if either `x` or `y` is `True` • `False` otherwise
`not`	`not x`	• `True` if `x` is `False` • `False` if `x` is `True`

If `x` is	`x and y` returns
Truthy	`y`
Falsy	`x`

If `x` is	`x or y` returns
Truthy	`x`
Falsy	`y`

If `x` is	`not x` returns
Truthy	`False`
Falsy	`True`

Identity Operators in Python

Operator	Sample Expression	Result
`is`	`x is y`	• `True` if `x` and `y` hold a reference to the same in-memory object • `False` otherwise
`is not`	`x is not y`	• `True` if `x` points to an object different from the object that `y` points to • `False` otherwise

Membership Operators in Python

Operator	Sample Expression	Result
`in`	`value in collection`	• `True` if `value` *is* present in `collection` • `False` otherwise
`not in`	`value not in collection`	• `True` if `value` *is not* present in `collection` of values • `False` otherwise

Bitwise Operators in Python

Operator	Operation	Sample Expression	Result
`&`	Bitwise AND	`a & b`	• Each bit position in the result is the logical AND of the bits in the corresponding position of the operands. • `1` if both bits are `1`, otherwise `0`.
`|`	Bitwise OR	`a | b`	• Each bit position in the result is the logical **OR** of the bits in the corresponding position of the operands. • `1` if either bit is `1`, otherwise, `0`.
`~`	Bitwise NOT	`~a`	• Each bit position in the result is the logical negation of the bit in the corresponding position of the operand. • `1` if the bit is `0` and `0` if the bit is `1`.
`^`	Bitwise XOR (exclusive OR)	`a ^ b`	• Each bit position in the result is the logical **XOR** of the bits in the corresponding position of the operands. • `1` if the bits in the operands are different, `0` if they're equal.
`>>`	Bitwise right shift	`a >> n`	Each bit is shifted right `n` places.
`<<`	Bitwise left shift	`a << n`	Each bit is shifted left `n` places.

Operator Precedence in Python

Operators	Description
`**`	Exponentiation
`+x`, `-x`, `~x`	Unary positive, unary negation, bitwise negation
`*`, `/`, `//`, `%`	Multiplication, division, floor division, modulo
`+`, `-`	Addition, subtraction
`<<`, `>>`	Bitwise shifts
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`==`, `!=`, `<`, `<=`, `>`, `>=`, `is`, `is not`, `in`, `not in`	Comparisons, identity, and membership
`not`	Boolean NOT
`and`	Boolean AND
`or`	Boolean OR
`:=`	Walrus

Arithmethic Augmented Assignment Operators

Operator	Description	Sample Expression	Equivalent Expression
`+=`	Adds the right operand to the left operand and stores the result in the left operand	`x += y`	`x = x + y`
`-=`	Subtracts the right operand from the left operand and stores the result in the left operand	`x -= y`	`x = x - y`
`*=`	Multiplies the right operand with the left operand and stores the result in the left operand	`x *= y`	`x = x * y`
`/=`	Divides the left operand by the right operand and stores the result in the left operand	`x /= y`	`x = x / y`
`//=`	Performs floor division of the left operand by the right operand and stores the result in the left operand	`x //= y`	`x = x // y`
`%=`	Finds the remainder of dividing the left operand by the right operand and stores the result in the left operand	`x %= y`	`x = x % y`
`**=`	Raises the left operand to the power of the right operand and stores the result in the left operand	`x **= y`	`x = x**y`

Bitwise Augmented Assignment Operators

Operator	Operation	Example	Equivalent
`&=`	Augmented bitwise AND (conjunction)	`x &= y`	`x = x & y`
`|=`	Augmented bitwise OR (disjunction)	`x |= y`	`x = x | y`
`^=`	Augmented bitwise XOR (exclusive disjunction)	`x ^= y`	`x = x ^ y`
`>>=`	Augmented bitwise right shift	`x >>= y`	`x = x >> y`
`<<=`	Augmented bitwise left shift	`x <<= y`	`x = x << y`

Concatenation and Repetition Operators

Operator	Operation	Sample Expression	Result
`+`	Concatenation	`seq_1 + seq_2`	A new sequence containing all the items from both operands
`*`	Repetition	`seq * n`	A new sequence containing the items of `seq` repeated `n` times

Concatenation and Repetition Augmented Assignment Operators

Operator	Description	Example
`+=`	• Runs an augmented concatenation operation on the target sequence. • Mutable sequences are updated in place. • If the sequence is immutable, then a new sequence is created and assigned back to the target name.	`seq_1 += seq_2`
`*=`	• Adds `seq` to itself `n` times. • Mutable sequences are updated in place. • If the sequence is immutable, then a new sequence is created and assigned back to the target name.	`seq *= n`