Operators In Python

By Bhadra Reddy

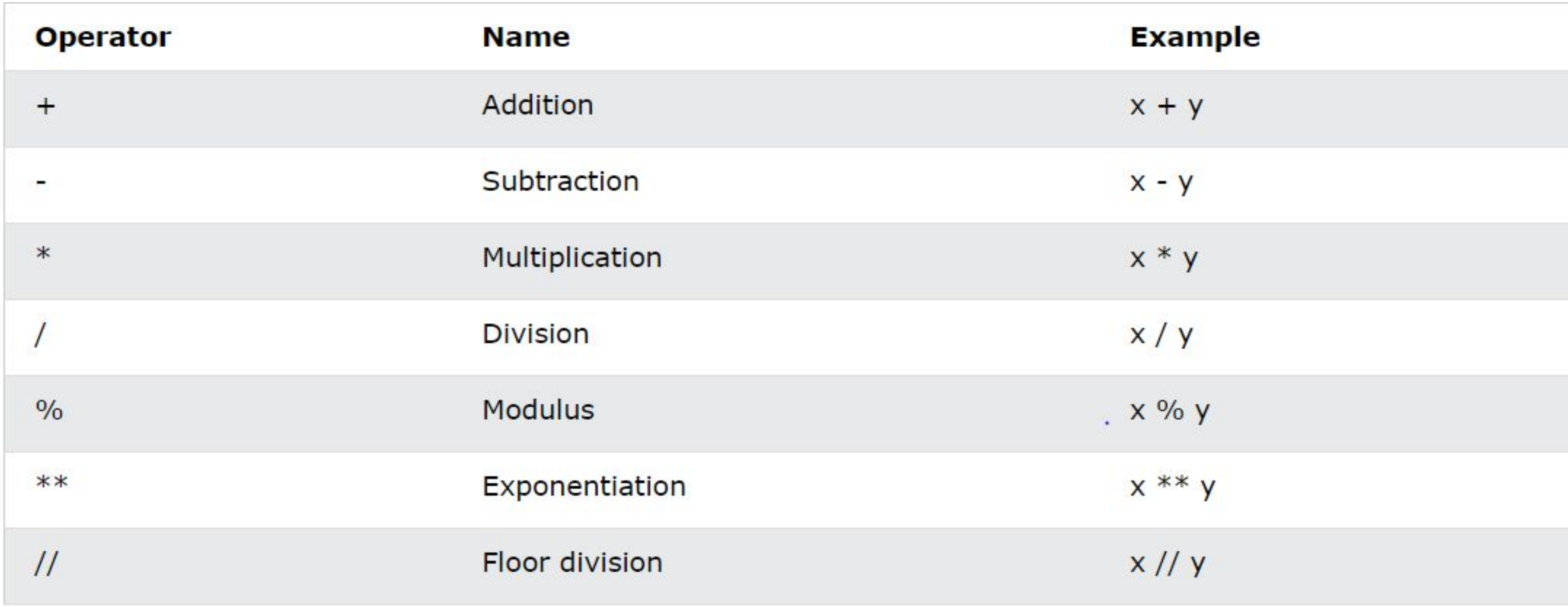
Operator in python

- Operators allow us to perform specific operations on variables.
- The operator can be defined as a symbol which is responsible for a particular operation between two operands.
- There are different types of operators in Python such as

1) Arithmetic operators
2) Assignment operators
3) Comparison operators
4) Logical operators
5) Identity operators
6) Membership operators
7) Bitwise operators

1) Arithmetic Operators (+, - , *, %)

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

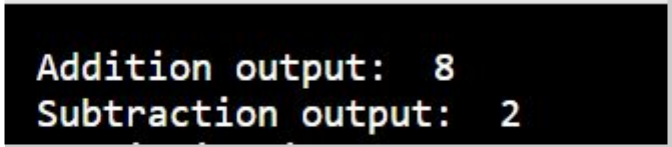


Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Examples

```
# Addition operation(+)

x = 5

y = 3

print("Addition output: ", x + y)

# Subtraction operation(-)

x = 5

y = 3

print("Subtraction output: ", x - y)
```

Output:

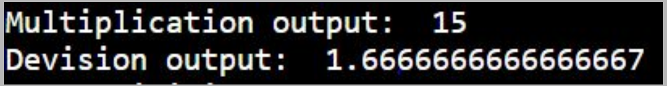


```
Addition output:  8
Subtraction output:  2
```

Example

```
# Multiplication Operation(*)

x = 5

y = 3

print("Multiplication output: ", x * y)

# Division Operation(/)

x = 5

y = 3

print("Division output: ", x / y)
```

Output:

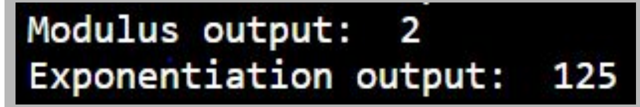


```
Multiplication output:  15
Devision output:  1.6666666666666667
```

Example

```
# Modulus Operation(%)

x = 5

y = 3

print("Modulus output: ", x % y)

# Exponentiation Operation(**)

x = 5

y = 3

print("Exponentiation output: ", x ** y)
```

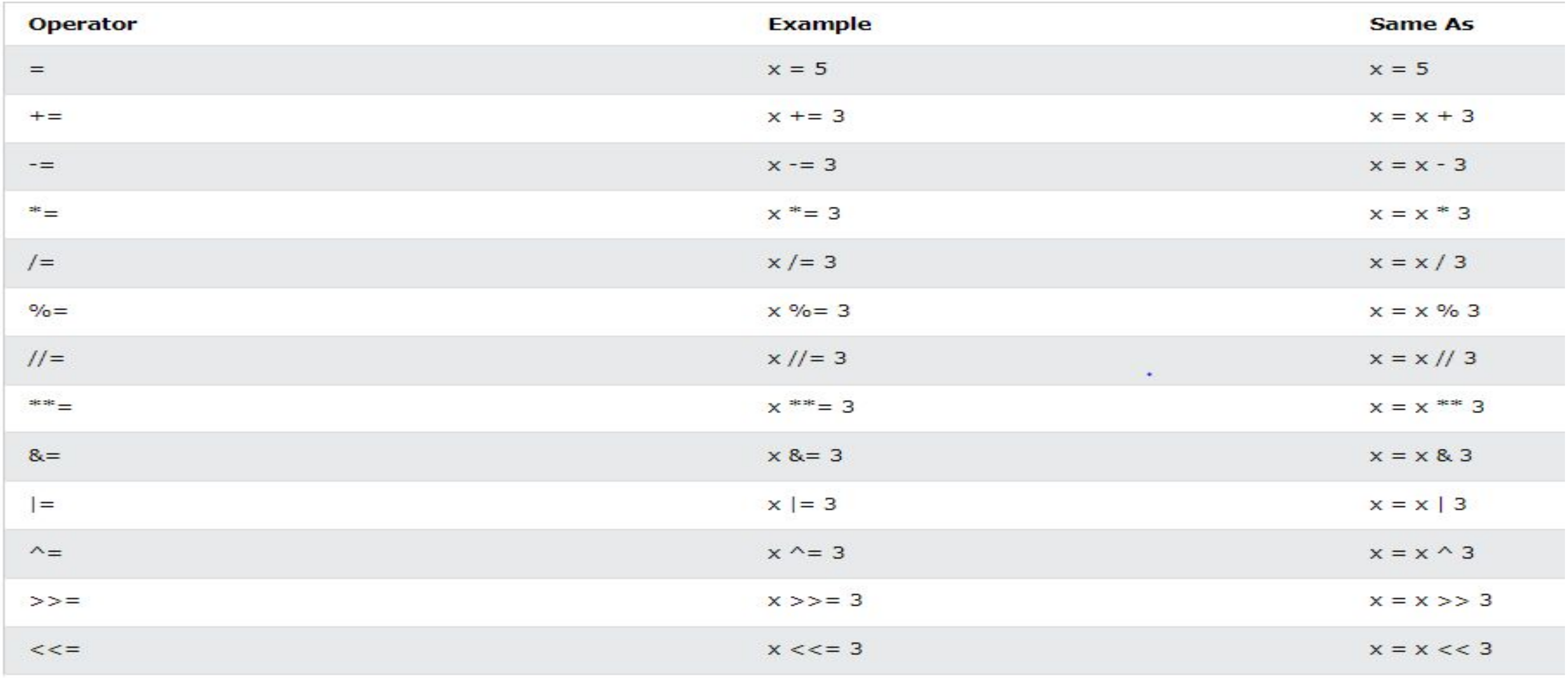
Output:



```
Modulus output:  2
Exponentiation output:  125
```

2) 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

1) Assign(=):

This operator is used to assign the value of the right side of the expression to the left side operand.

Syntax:

```
Variable_name = value
```

```
# Example Program
# Assigning values using
# Assignment Operator
a = 3
b = 5
c = a + b
print(c)
```

Output:

```
8
```

2) add and assign (+=)

This operator is used to add the right side operand with the left side operand and then assigning the result the left operand.

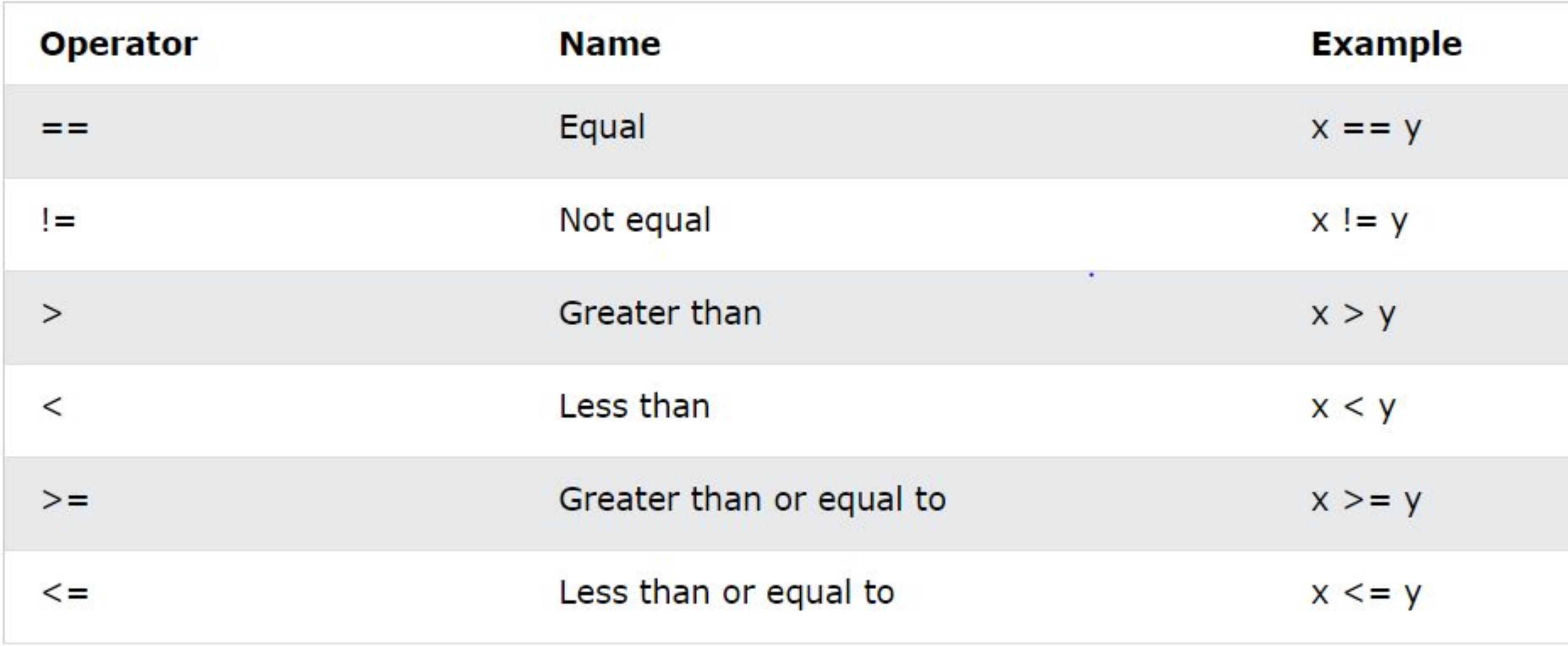
Syntax:

x += y   # Note x and y are operands

```
Example:
a = 3
b = 5

# a = a + b
a += b

# Output
print(a)
```

3) Comparison Operators in Python

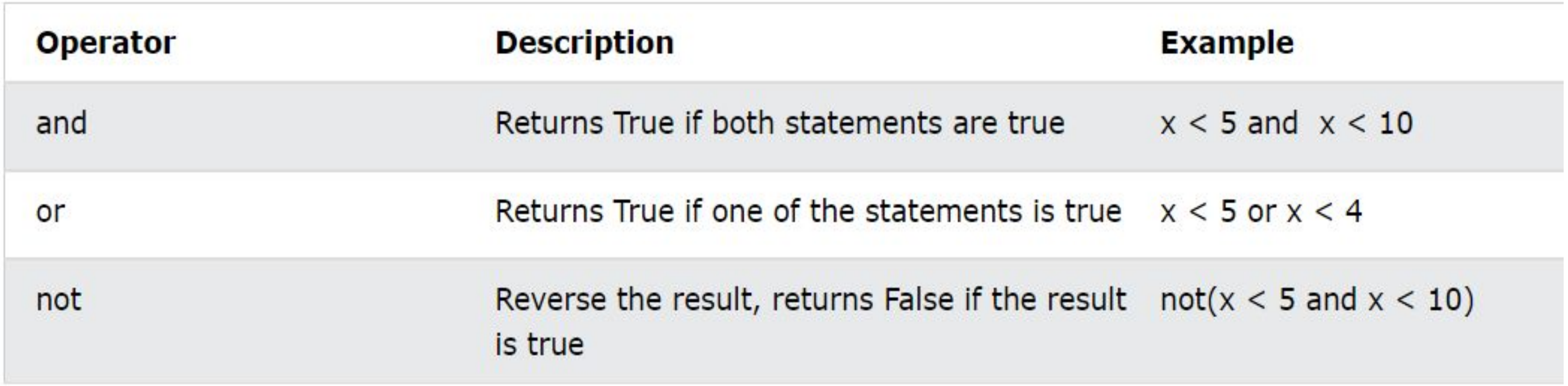
Comparison operators are used to compare two values:



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

4) Logical operators in Python

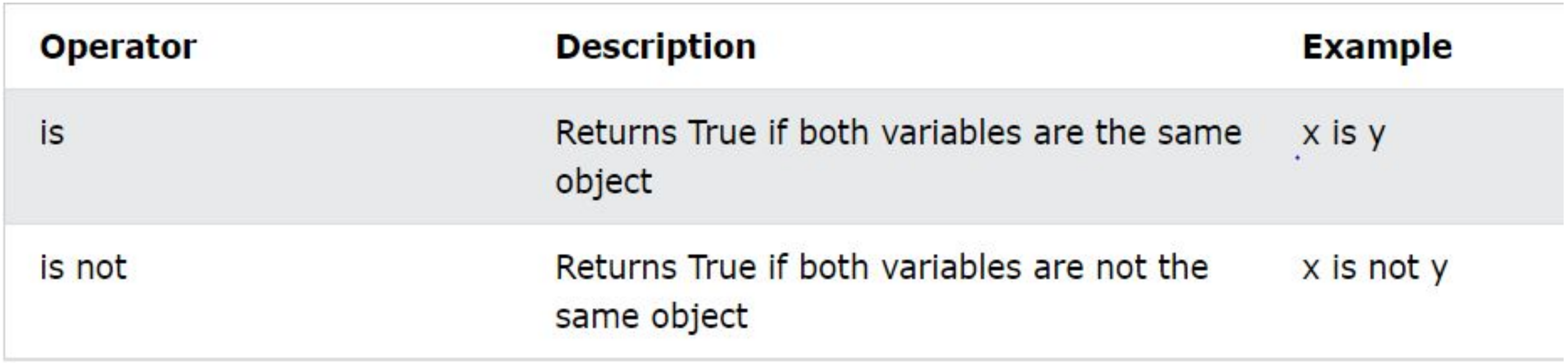
Logical operators are used to combine conditional statements



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

5) Identity Operators in python

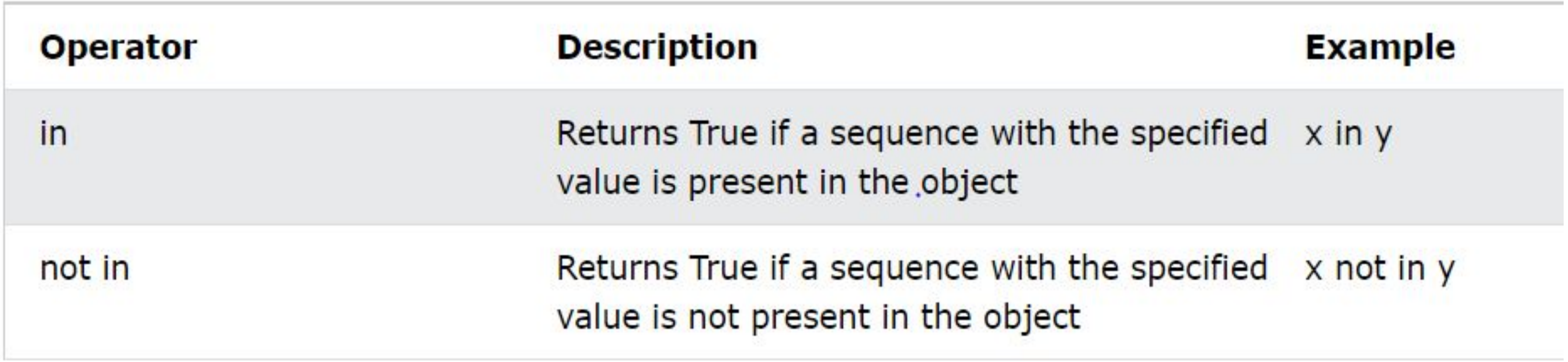
Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:



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

6) Membership Operators in Python

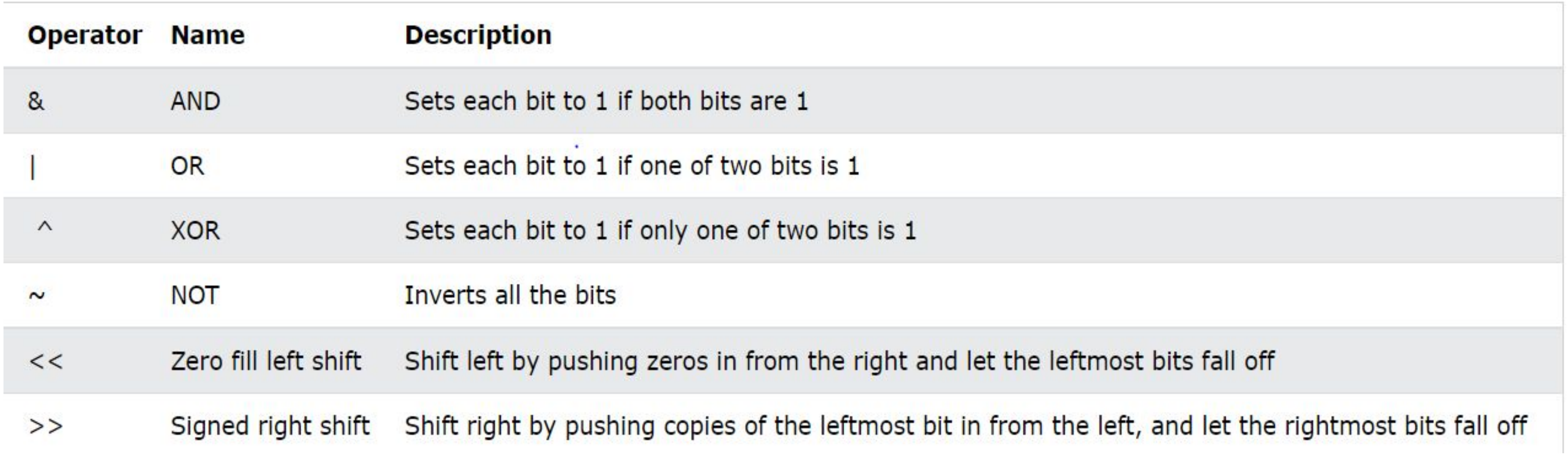
Membership operators are used to test if a sequence is presented in an object



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

7) Bitwise Operators in python

Bitwise operators are used to compare (binary) numbers



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