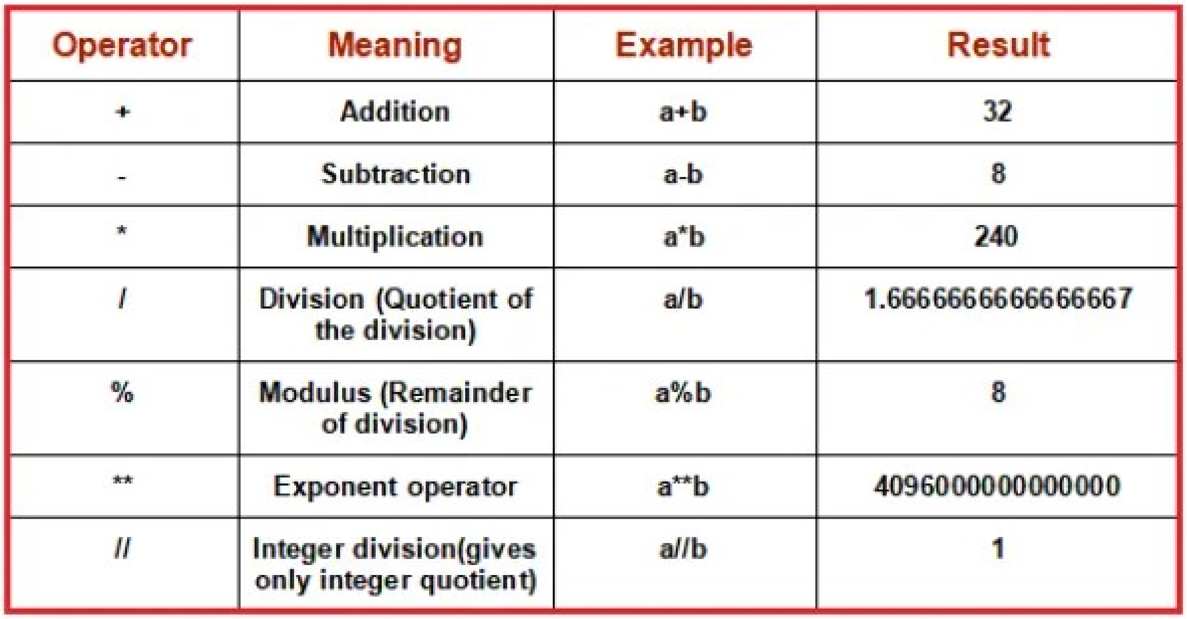
## Operators:

**ARITHMETIC OPERATORs:**
As stated above, these are used to perform that basic mathematical stuff as done in every programming language. Let’s understand them with some examples. Let’s assume, a = 20 and b = 12



| Operator | Meaning | Example | Result |
|---|---|---|---|
| + | Addition | a+b | 32 |
| - | Subtraction | a-b | 8 |
| * | Multiplication | a*b | 240 |
| / | Division (Quotient of the division) | a/b | 1.6666666666666667 |
| % | Modulus (Remainder of division) | a%b | 8 |
| ** | Exponent operator | a**b | 4096000000000000 |
| // | Integer division(gives only integer quotient) | a//b | 1 |

**Note:** Division operator / always performs floating-point arithmetic, so it returns a float value. Floor division (//) can perform both floating-point and integral as well,

1. If values are int type, the result is int type.
2. If at least one value is float type, then the result is of float type.

**Example: Arithmetic Operators in Python:**

```
a = 20
b = 12
print(a+b)
print(a-b)
print(a*b)
print(a/b)
print(a%b)
print(ab)
print(a//b)
```

```

O/P:-
32
8
240
1.6666666666666667
8
4096000000000000
1
```

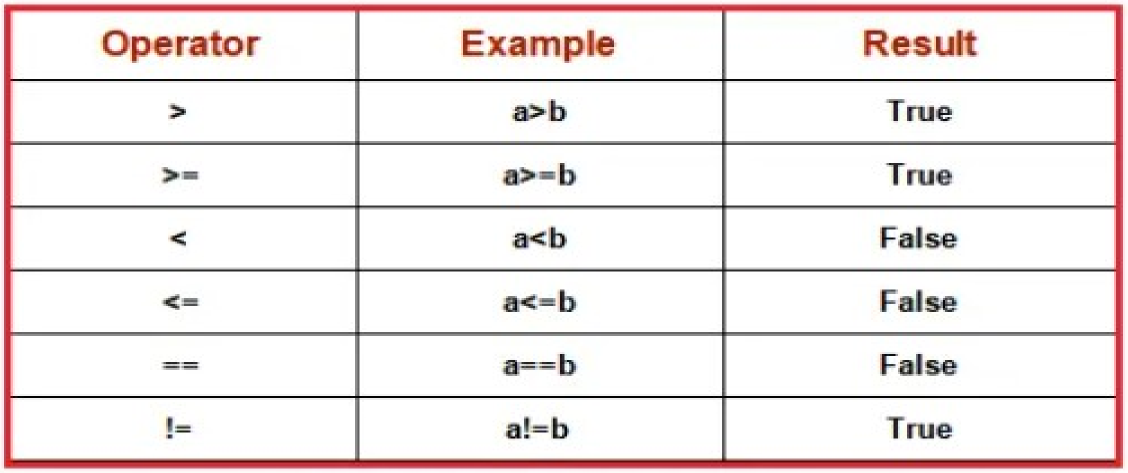
**Example: Floor division**

```
print(12//5)
print(12.0//5)

O/P:-
2
2.0
```

**Relational Operators in Python:-**

These are used to compare two values for some relation and return True or False depending on the relation. Let's assume, a = 13 and b = 5.



| Operator | Example | Result |
|---|---|---|
| > | a>b | True |
| >= | a>=b | True |
| < | a<b | False |
| <= | a<=b | False |
| == | a==b | False |
| != | a!=b | True |

**Example: Relational Operators in Python**

```
a = 13
b = 5
print(a>b)
print(a>=b)
print(a<b)
print(a<=b)
print(a==b)
print(a!=b)

O/P:-
True
True
False
False
False
True
```

**LOGICAL OPERATORS:-**

In python, there are three types of logical operators. They are and, or, not. These operators are used to construct compound conditions, combinations of more than one simple condition. Each simple condition gives a boolean value which is evaluated, to return the final boolean value.

**Note:** In logical operators, False indicates 0(zero) and True indicates non-zero value. Logical operators on boolean types

1. **and**: If both the arguments are True then only the result is True
2. **or**: If at least one argument is True then the result is True
3. **not**: the complement of the boolean value

**Example: Logical operators on boolean types in Python**

```
a = True
b = False
print(a and b)
print(a or b)
print(not a)
print(a and a)
```

```
O/P:-
False
True
False
True
```

**and operator:**

‘A and B’ returns A if A is False
‘A and B’ returns B if A is not False

**Or Operator in Python:**

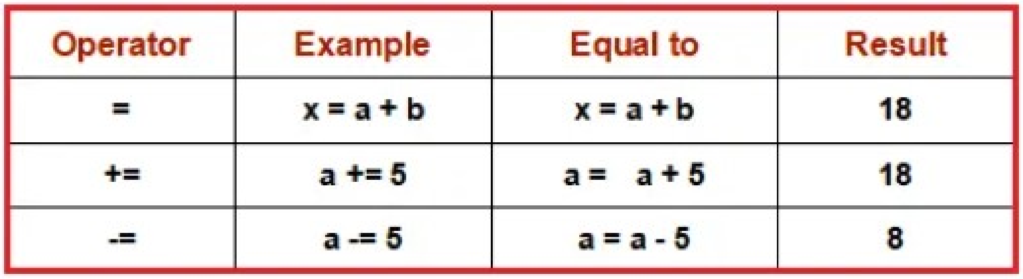
‘A or B’ returns A if A is True
‘A or B’ returns B if A is not True

**Not Operator in Python:**

not A returns False if A is True
not B returns True if A is False

**ASSIGNMENT OPERATORS:**

By using these operators, we can assign values to variables. ‘=’ is the assignment operator used in python. There are some compound operators which are the combination of some arithmetic and assignment operators (+=, -=, *=, /=, %=, **=, //= ). Assume that, a = 13 and b = 5



| Operator | Example | Equal to | Result |
|---|---|---|---|
| = | x = a + b | x = a + b | 18 |
| += | a += 5 | a = a + 5 | 18 |
| -= | a -= 5 | a = a - 5 | 8 |

**Example: Assignment Operators in Python**

```
a=13
print(a)
a+=5
print(a)

O/P:-
13
18
```

**MEMBERSHIP OPERATORS:----**

Membership operators are used to checking whether an element is present in a sequence of elements are not. Here, the sequence means strings, list, tuple, dictionaries, etc which will be discussed in later chapters. There are two membership operators available in python i.e. in and not in.

1. **in operator:** The in operators returns True if element is found in the collection of sequences. returns False if not found
2. **not in operator:** The not-in operator returns True if the element is not found in the collection of sequence. returns False in found

**Example: Membership Operators**

```
text = "Welcome to python programming"
print("Welcome" in text)
print("welcome" in text)
print("nireekshan" in text)
print("Hari" not in text)

O/P:-
True
False
False
True
```

**Example: Membership Operators**

```
names = ["Ramesh", "Nireekshan", "Arjun", "Prasad"]
print("Nireekshan" in names)
print("Hari" in names)
print("Hema" not in names)

O/P:-
True
False
True
```

**IDENTITY OPERATORS:--**

This operator compares the memory location( address) to two elements or variables or objects. With these operators, we will be able to know whether the two objects are pointing to the same location or not. The memory location of the object can be seen using the id() function.

**Example: Identity Operators**

```
a = 25
b = 25
print(id(a))
print(id(b))

O/P:-
1487788114928
1487788114928
```

**Types of Identity Operators in Python:**

There are two identity operators in python, is and is not.

**is:**

1. A is B returns True, if both A and B are pointing to the same address.
2. A is B returns False, if both A and B are not pointing to the same address.

**is not:**

1. A is not B returns True, if both A and B are not pointing to the same object.
2. A is not B returns False, if both A and B are pointing to the same object.

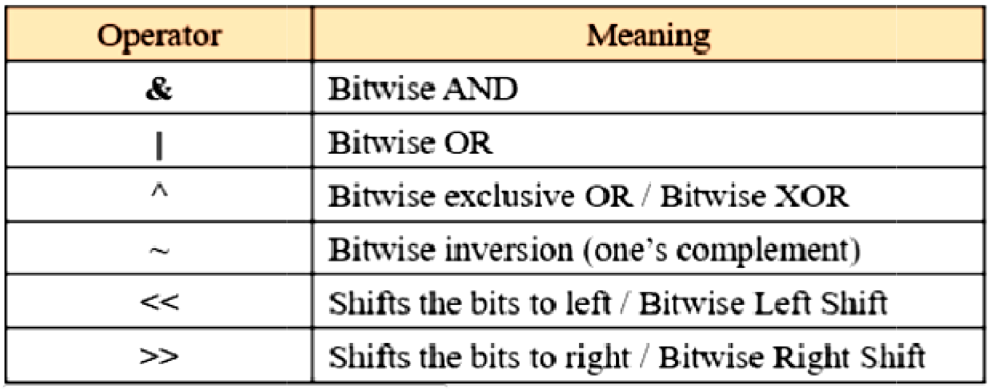
**Example: Identity Operators in Python**

```
a = 25
b = 25
print(a is b)
print(id(a))
print(id(b))

O/P:-
True
2873693373424
2873693373424
```

**Note:** The 'is' and 'is not' operators are not comparing the values of the objects. They compare the memory locations (address) of the objects. If we want to compare the value of the objects. we should use the relational operator '=='.

**Bitwise wise operator :-**



| Operator | Meaning |
|---|---|
| & | Bitwise AND |
| \| | Bitwise OR |
| ^ | Bitwise exclusive OR / Bitwise XOR |
| ~ | Bitwise inversion (one's complement) |
| << | Shifts the bits to left / Bitwise Left Shift |
| >> | Shifts the bits to right / Bitwise Right Shift |

| | | | | | | |
|---|---|---|---|---|---|---|
| ------------- | 4 | 3 | 2 | 1 | 0 | **Bit position** |
| | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | **Bit weight** |
| | | | **Binary number** | | | |
| **Decimal number** | 16 | 8 | 4 | 2 | 1 | |
| 0 | 0 | 0 | 0 | 0 | 0 | |
| 1 | 0 | 0 | 0 | 0 | 1 | |
| 2 | 0 | 0 | 0 | 1 | 0 | |
| 3 | 0 | 0 | 0 | 1 | 1 | |
| 4 | 0 | 0 | 1 | 0 | 0 | |
| 5 | 0 | 0 | 1 | 0 | 1 | |
| 6 | 0 | 0 | 1 | 1 | 0 | |
| 7 | 0 | 0 | 1 | 1 | 1 | |
| 8 | 0 | 1 | 0 | 0 | 0 | |
| 9 | 0 | 1 | 0 | 0 | 1 | |
| A | 0 | 1 | 0 | 1 | 0 | |
| B | 0 | 1 | 0 | 1 | 1 | |
| C | 0 | 1 | 1 | 0 | 0 | |
| D | 0 | 1 | 1 | 0 | 1 | |
| E | 0 | 1 | 1 | 1 | 0 | |
| F | 0 | 1 | 1 | 1 | 1 | |

**Bit-wise and(&):-----**

```
x = 10
y = 20
print(x & y)
```

```
          x = 10      0     1     0     1     0
                      &     &     &     &     &
          y = 20      1     0     1     0     0
          -------------------------
          o/p=0       0     0     0     0     0
```

**<u>Bit-wise or( | ):-----</u>**

**x = 10**

**y = 20**

**print(x | y)**

```
          x = 10      0     1     0     1     0
                      |     |     |     |     |
          y = 20      1     0     1     0     0
        ---------------------------------------------
          o/p=30      1     1     1     1     0
```

**<u>Left Shift :</u>--**

```
x =10
Print( x<<2)          32    16    8     4     2     1
     x=10                         1     0     1     0
     40               1     0     1     0     0     0
```

**<u>Right Shift :</u>--**

```
x =10

Print( x>>2)          8     4     2     1    .(1/2) (1/4)

           x=10       1     0     1     0

           o/p= 2                 1     0     .     1      0
```