Operator: Operator are special symbols in python or any language which can manipulate the value of operands

- The value that the operator operates on is called the operand
- For example: here 2 + 3 = 5. Here, is the operator that performs addition and 2 and 3 represent the operands.

Types of operator

- Arithmetic operators
- Comparison operators or Relational operators
- Logical operators
- Assignment operators
- Identity operators
- Membership operators
- bitwise operators

Arithmetic operator

Arithmetic operators are used with numeric values to perform commom mathematical operations

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Modulus (%)
- Exponentiation (**)
- Floor division (//)

In [1]:
```
a = 10
b = 20
print(a + b)            #Addition
print(a - b)            #Subtraction
print(a * b)            #Multiplication
print(a / b)            #Division
print(a % b)            #Modulus
print(a ** b)           #Exponentiation
print(a // b)           #Floor division
```

```
30
-10
200
0.5
10
100000000000000000000
0
```

Relational Operators

- Greater than (>)
- less than (<)
- Greater Equal equal to (>=)
- Less than equal to (<=)
- Equal to (==)
- Not Equal to

In [2]:
```
a = 40
b = 40
print( a > b)           #Greater than
print( a < b)           # less than
print( a >= b)          #Greater Equal to
print( a <= b)          #Less than Equal
print( a == b)          #Equal to
print( a != b)          #Not Equal to
```

```
False
False
True
True
True
False
```

How to find Unicode value

In [5]:
```
a ="a"
ord(a)
```

Out[5]:
```
97
```

Logical operator

- And (&)
- or (|)
- not (!)

In [7]:
```
a = 20
b = 50
print ( a == 20 and b == 50)
print ( a == 20 or b == 50)
print ( a != 20)
```

```
True
True
False
```

Assignment operator: Assignment operator is nothing it is equal to opperator

In [8]:
```
a=20
a
```

Out[8]:
```
20
```

In [9]:
```
x = y = z = 28
```

In [10]:
```
print(x)
print(y)
print(z)
```

```
28
28
28
```

compound assignment operator

- +=
- -=
- *=
- /=
- %=
- //=
- **=
- &=
- !=
- ^=
- =
- <<=

In [14]:
```
x
```

Out[14]:
```
-28
```

Identity operator

```
There are two types of the identity operator
1. is
2. is not
```

The identity operator basically compares the id of the variables if 'id' is the same then the 'is' operator gives true otherwise false

In [8]:
```
a = 4
b = 5
a is not b
```

Out[8]:
```
True
```

In [24]:
```
id(a)
```

Out[24]:
```
140727932746264
```

```
In [25]: id(b)

Out[25]: 140727932746296
```

Since a and b are pointing to same objects, so the operator is returns True

```
In [16]: a = 5
         b = 5
         a is b

Out[16]: True

In [19]: id(a)

Out[19]: 140727932746296

In [20]: id(b)

Out[20]: 140727932746296
```

Membership Operators

Membership operators are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary)

There are 2 Membership operators

- in
- not in

```
In [17]: 's' in 'sanjan'

Out[17]: True

In [18]: 's' not in 'sanjan'

Out[18]: False

In [5]: import datetime
        age = eval(input("Enter your age "))
        print(f"your age is {age} in {datetime.datetime.now().year}")
        print(f"you will be 100 year old in {datetime.datetime.now().year +(100 - age)}"

       your age is 7 in 2024
       you will be 100 year old in 2117

In [7]: fname = input("Enter you name")
        lname = input("Enter your last name")
        print(fname[::-1]+" "+lname[::-1])

         najnas tidnap
```

Bitwise operator

1. Bitwise AND (&)
2. Bitwise OR (|)
3. Bitwise XOR (^)
4. Bitwise NOT (~)
5. Left Shift (<<)
6. Right Shift (>>)

In [10]:
```
# Example 1: Bitwise AND
a = 12  # 1100 in binary
b = 5   # 0101 in binary
result = a & b  # 0100 in binary (4 in decimal)
print(result)  # Output: 4

# Example 2: Bitwise OR
a = 12  # 1100 in binary
b = 5   # 0101 in binary
result = a | b  # 1101 in binary (13 in decimal)
print(result)  # Output: 13

# Example 3: Bitwise XOR
a = 12  # 1100 in binary
b = 5   # 0101 in binary
result = a ^ b  # 1001 in binary (9 in decimal)
print(result)  # Output: 9

# Example 4: Bitwise NOT
a = 12  # 1100 in binary
result = ~a  # 0011 in binary (-13 in decimal, two's complement)
print(result)  # Output: -13

# Example 5: Left Shift
a = 12  # 1100 in binary
result = a << 2  # 110000 in binary (48 in decimal)
print(result)  # Output: 48

# Example 6: Right Shift
a = 12  # 1100 in binary
result = a >> 2  # 11 in binary (3 in decimal)
print(result)  # Output: 3
```

```
4
13
9
-13
48
3
```