Operators

Operators are special symbols or keywords that used to perform operations on variables and values. There are so many types of operators:

1. Arithmetic operators

2. Relational Operators

3. Logical Operators

4. Assignment Operators

5. Membership Operators

6. Identity Operators

7. Bitwise wise operator

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 example. Let's assume, a = 20 and b = 12.

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,

- If values are `int` type, the result is int type.
- 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(a//b)

# Output
32
8
240
```

```
1.6666666666666667
8
1
```

Example: Floor division

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

# Output
2
2.0
```

Relational Operators

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.

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)

# Output
True
True
False
False
False
True
```

Logical Operators

(`and`, `or`, `not`)

In python, there are three categories 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

`and` Operator:

and: If both the arguments are True then only the result is True
‘A and B’ returns A if A is False
‘A and B’ returns B if A is not False

```
a = True
b = False
print(a and b)

# Output
False
```

`or` Operator in Python:

or: If at least one argument is True then the result is True
‘A or B’ returns A if A is True
‘A or B’ returns B if A is not True

```
a = True
b = False
print(a or b)

# Output
True
```

`not` Operator in Python:

not: the complement of the boolean value
not A returns False if A is True
not B returns True if A is False

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

# Output
False
True
```

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

Example: Assignment Operators in Python

Operator	Description	Example	Result
`+=`	Adds right-hand operand to left-hand	`x += 3`	`x = x + 3`
`-=`	Subtracts right-hand operand from left-hand	`x -= 4`	`x = x - 4`
`*=`	Multiplies left-hand variable by right-hand operand	`x *= 5`	`x = x * 5`
`/=`	Divides left-hand variable by right-hand operand	`x /= 4`	`x = x / 4`
`%=`	Modulo (remainder of division)	`x %= 3`	`x = x % 3`
`**=`	Exponentiation (raises to power)	`x **= 3`	`x = x ** 3`
`//=`	Floor division (divides and rounds down)	`x //= 2`	`x = x // 2`

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.

`in` operator: The in operators returns True if element is found in the collection of sequences. returns False if not found

`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 Cybrom Technology"
print("Welcome" in text)
print("welcome" in text)
print("nireekshan" in text)
print("Hari" not in text)

# Output
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)

# Output
True
False
True
```

Identity Operators

(`is`,`is not`)

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. Identity operators in Python are used to compare the memory location of two objects. They are:

`is`: Returns True if both variables refer to the same object in memory.
`is not`: Returns True if both variables do not refer to the same object in memory

Example: Identity Operators

```
x = 10
y = 10
print(x is y)
print(x is not y)

# Output
True
False
```

```
a = [1, 2, 3]
b = [1, 2, 3]
print(a is b)
print(a is not b)

# Output
False
True
```

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

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

```
x = 10
y = 20
print(x & y)
                 32      16      8       4       2       1
        x = 10           0       1       0       1       0
                         &       &       &       &       &
        y = 20           1       0       1       0       0
        ------------------------------------------------
        o/p=0            0       0       0       0       0
```

Bit-wise or(|):-----

```
x = 10
y = 20
print(x | y)
                    32  16      8       4       2       1
        x = 10           0       1       0       1       0
                         |       |       |       |       |
        y = 20           1       0       1       0       0
        ------------------------------------------------
        o/p=30           1       1       1       1       0
```

Left Shift :--

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

Right Shift :--

```
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
```