Arithmetic Operators

These operators are used to perform arithmetic operations. Some of the operators also work on string, list and other data types. Take a look at the list of operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
//	Floor Division
%	Modulus
**	Exponential

*Note: ** enjoys highest priority and having associativity right to left. /,//,*,% have same priority (lower than **) but their associativity is left to right. +, - has same priority (lower than /) their associativity is also left to right.*

<u>An Example:</u>

```
x = 5
y = 3

print(x + y)     #will result 8
print(x - y)     #will result 2
print(x * y)     #will result 15
print(x / y)     #will result 1.6666666666666667
print(x // y)    #will result 1
print(x % y)     #will result 2
print(x ** y)    #will result 125
```

The + and * operators also works for string & list (discussed later) data type. The + operator work for concatenation in String data type and * operator work for replication. For + operator both values must be of string type and for * operator one operand must be of int data type
.

<u>An Example</u>

```
s1 = 'ABC'
print(s1 + 'DEF')      #will result ABCDEF
#print(s1 + 45)        TypeError: must be str, not int
print(s1 + str(45))    #will result ABC45
print(s1 * 3)          #will result in ABCABCABC
#print(s1 * 'D')       can't multiply sequence by non-int of type 'str'
```

Note: the str() function is used to convert int/float value to string.

Relational operators

Operator	Description
==	Equality/Comparison
!=	Inequality
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

- Output of relational expression is always either True of False value.

<u>An Example</u>

```
x = 5
y = 3
z = 5

print(x == y)          #will result in False
print(x == z)         #will result in True
print(x != y)         #will result in True
print(x > y)          #will result in True
print(x < y)          #will result in False
print(x <= y)         #will result in False
print(x >= y)         #will result in True
```

Using relational operator we can compare String values also, the values are matched character by character on behalf of their ASCII value. To find ASCII value of a character we have ord() function and to find character on ASCII value we have chr() function.

<u>An Example</u>

```
print(ord('A'))       #will result 65
print(chr(97))        #will result a
```

Now let us take another example of comparison

```
s1 = 'Apple'
s2 = 'Apple'
print(s1 == s2)       #will result in True
print(s1 != s2)       #will result in False

s3 = ' Apple'         #mind space at the starting

print(s1 == s3)       #will result in False
print(s1 != s3)       #will result in True
print(s1 > s3)        #will result in True
print(s1 < s3)        #will result in False

s4 = 'Applepie'
print(s3 < s4)        #will result in True
```

Logical operators

Operator	Description
and	True if both operands are true, false otherwise
or	True if any operand is true, false otherwise
not	True if operand is false

The truth table for logical operator is as follow *(for boolean operands)*

X	Y	X and Y	X or Y	not Y
True	True	True	True	False
True	False	False	True	True
False	True	False	True	
False	False	False	False	

These operators are used to join conditions. We will discuss about these operators later

<u>An Example</u>

```
print(5 > 8 or 5 < 2)   #will result in false as both are false
print(5 > 8 or 5 < 10)  #will result in true as second condition is true
print(5 > 2 or 5 < 2)   #will result in true as first condition is true
print(5 > 2 or 5 < 10)  #will result in true as first condition is true
```

Note: One thing to keep in mind is that whenever L.H.S. Condition of or operator is True then next condition is not checked because it does not have any impact on final result so ultimate result is evaluated to True

```
print(5 > 2 and 5 < 10) #will result in true as both condition is true
print(5 > 2 and 5 < 2)  #will result in false as second condition is false
print(5 > 8 and 5 < 10) #will result in true as first condition is false
print(5 > 8 and 5 < 2)   #will result in false as first condition is false
```

Note: One thing to keep in mind is that whenever L.H.S. Condition of and operator is False then next condition is not checked because it does not have any impact on final result so ultimate result is evaluated to False

As we already know that any nonzero numeric value/non empty string or sequence is considered equivalent to True, zero numeric value/ empty string or sequence is considered as false. We can also use non-boolean values as operands for and/or operator but the way of evaluation is different. Take a look at table below

X	**Y**	**X and Y**	**X or Y**	**not Y**
$true_{tval}$	$true_{tval}$	Y	X	False
$true_{tval}$	$false_{tval}$	Y	X	True
$false_{tval}$	$true_{tval}$	X	Y	
$false_{tval}$	$false_{tval}$	X	Y	

Here is the summarization of the above table

- In expression X or Y, if first operand (i.e. expression X) has $false_{tval}$ then return second operand Y as result otherwise return X.
- In expression X and Y, if first operand (i.e. expression X) has $False_{tval}$ then return first operand X as result otherwise return Y.
- The Not/negation operator negates or reverses the truth value of the expression following it i.e. if the expression id True or $true_{tval}$ then it will result false and vice versa.

<u>An Example</u>

```
print(5 or 10)       #will result in 5
print(10 or 0)       #will result in 10
print(0 or 8)        #will result in 8
print(0 or 0)        #will result in 0
print(0 or (5 > 2)) #will result in True

print(5 and 10)      #will result in 10
print(10 and 0)      #will result in 0
print(0 and 10)      #will result in 0
print(0 and 0)       #will result in 0

print(not 5)         #will result in False
print(not 0)         #will result in True
```

One more thing to keep in mind is that in python programming expression

```
     x < y < z
```

is same as

```
     x < y and y < z
```

Membership operators

These operators are used to test for membership in a sequence, such as strings, lists, or tuples. It has only two operators that are as follow

Operator	**Description**
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.

<u>An Example</u>

```
print('m' in 'metier')            #will result in True
print('l' not in 'metier')        #will result in True
print('met' in 'metier')          #will result in True
print('net' not in 'metier')      #will result in True
```

Identity Operators

The operators are used to determine whether a value is of a certain class or type. They are usually used to determine the type of data a certain variable contains. It also has only two operators that are as follow

Operator	Description
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.

<u>An Example:</u>
```
a = 10
print(type(a) is int)
a = 1.5
print(type(a) is float)
a = "RegEx"
print(type(a) is str)
```

The output of above code is

```
True
True
True
```

Difference between == and `is` operator

- The == operator is just used to check for the equality of values of variables means it does not check if two variables refers to the same object or not.
- On the other hand `is` operator checks if two variables refers to the same object or not. As we can say that if variables refer to the same object then their value will be same.
- if `is` operator returns true for two variables then == will also returns true but vice versa does not hold necessarily.

<u>An Example</u>
```
a = 10
b = 10
print("a == b:",a == b)
print("a is b:",a is b)
print("id(a) =",id(a),"id(b) =",id(b))

print("==================================")

p = 1.5
q = float(input("Please enter 1.5 "))
print("p == q:",p == q)
print("p is q:",p is q)
print("id(p) =",id(p),"id(q) =",id(q))
```

The output of above code is
```
a == b: True
a is b: True
id(a) = 1990160128 id(b) = 1990160128
==================================
Please enter 1.5 1.5
p == q: True
p is q: False
id(p) = 353238588008 id(q) = 353238587984
```

Assignment Operators

Operator	Description
=	assignment operator, used to assign value to a variable
+=	Addition assignment operator
-=	Subtraction assignment operator
*=	Multiplication assignment operator
/=	Division assignment operator

%=	Modulus assignment operator
//=	Floor Division assignment operator
**=	Exponential assignment operator
&=	Bitwise and assignment operator
I=	Bitwise or assignment operator
^=	Bitwise xor assignment operator
>>=	Bitwise right shift assignment operator
<<=	Bitwise left shift assignment operator

Ternary operator in python

Ternary operator, also known as conditional expressions are operators that evaluate something based on a condition being true or false. It simply allows to test a condition in a single line replacing the multiline if-else making the code compact. Syntax is as follow

true statement if condition else *false statement*

An Example:

```
a, b = 10, 20
min = a if a < b else b
print(min)

c, d = 20, 15
print ("Both c and d are equal" if c == d else "c is greater than d"
        if c > d else "c is less than d")
```

The above code will produce following output

```
10
c is greater than d
```

Slicing operator

The slicing operator is used to retrieve an element or set of elements from any sequence. The slicing operator has following general form

[start: stop: step]

The default value of start is 0, stop is n (number of elements in a sequence) and step is 1. The slicing operator starts from 0 and traverses up to n – 1 and covers every element that is multiple of step.

An Example

```
s1 = 'Python King'
print(s1)             # Prints complete string
print(s1[0])          # Prints first character of the string
print(s1[2:])         # Prints string starting from 3rd character
print(s1[:5])         #  Prints  string  starting  from  0th  character  to  5th
character
print(s1[2:5])        # Prints characters starting from 3rd to 5th
print(s1[0:11:2])     # Prints every alternative character
print(s1[-11:])       # Prints complete string
print(s1[-1:-12:-2])  # Prints every alternative character in reverse order
```