Logical Operators:

There are 3 types of Logical Operators in Python, they are

- **And:** Return True if both the values are True, else return False.

Value 1	Value 2	Result of AND
True	True	True
True	False	False
False	True	False
False	False	False

- **Or :** Return True if any one value is True , else return False.

Value 1	Value 2	Result of OR
True	True	True
True	False	True
False	True	True
False	False	False

- **Not :** Return the opposite of the given value.

Value	Result of NOT
True	False
False	True

Comparision Operators:

The main Comparision Operators in python include:

- **is** - Returns True if both the a and b point to the same memory location, else Return False. Usually, non mutable Data Types with the same values in them return True.

```
# The below code will return False because lst1 and lst2 were saved in a different location.
ip: lst1=[1,2,3,4,5]
    lst2=[1,2,3,4,5]
    lst1 is lst2
op: False


# The below code will return True because a and b were saved in the same location.
ip: a=9
```

```
    b=9
    a is b
op: True
```

- **is not** - It is the opposite of the is operator

```
# The below code will return True because lst1 and lst2 were saved in different locations.
ip: lst1=[1,2,3,4,5]
    lst2=[1,2,3,4,5]
    lst1 is not lst2
op: True


# The below code will return False because a and b were saved in the same location.
ip: a=9
    b=9
    a is not b
op: False
```

- **==** - it will return True if the value assigned to **a** is equal to value assigned to **b**, else False.

```
# The below code will return True because lst1 and lst2 have the same values in them.
ip: lst1=[1,2,3,4,5]
    lst2=[1,2,3,4,5]
    lst1 == lst2
op: True


# The below code will return False because a and b have different values.
ip: a=9
    b=90
    a == b
op: False
```

- **! =** - It is exactly to the opposite of **==** Operator.

```
# The below code will return False because lst1 and lst2 have the same values in them.
ip: lst1=[1,2,3,4,5]
    lst2=[1,2,3,4,5]
    lst1 != lst2
op: False


# The below code will return True because a and b have different values.
ip: a=9
    b=90
    a != b
op: True
```

Arithmetic Operators:

- **+** - addition
- **-** - subtraction
- ***** - multiplication
- **/** - division
- **//** - division without decimal values
- **%** **-** remainder

Number 1	Number 2	Operator	Result
100	9	+	109
100	9	-	91
100	9	*	900
100	9	/	11.1111
100	9	//	11
100	9	%	1

/

Strings:

In Python, a string is a sequence of characters represented in quotes (single or double).

Slicing is a way to extract a portion of a string, list, or any sequence. It's done by specifying the start and end index of the portion, separated by a colon (:). For example:

```
ip: my_string = "Hello World!"
    my_string[0:5]
op: Hello
```

In this example, the slicing **my_string[0:5]** returns a new string that starts at index 0 and ends at index 5 (not including index 5).

If the start index is not specified, it defaults to the start of the sequence. If the end index is not specified, it defaults to the end of the sequence. For example:

```
ip: my_string[:5]
op: Hello
ip: my_string[6:]
op: World!
```

Negative indices can also be used to slice from the end of the sequence. For example:

```
ip: my_string[-6:]
op: World!
```

Reversing a string in Python can be done using slicing with the **[::-1]** syntax. The **[::-1]** syntax means to start from the end of the string (**-1**), and go to the beginning with a step of **-1**. Here's an example:

```
ip: my_string = "Hello World!"
    reversed_string = my_string[::-1]
    reversed_string
op: !dlroW olleH
```

String Functions:

There are several inbuilt functions in python, some of the widely used functions are

len() returns the length of the string.

find() returns the index of the first occurrence of the **substring** in the **string**. If the **substring** is not found, it returns **-1**.

count() returns the number of occurrences of the **substring** in the **string**.

partition() returns a tuple that contains the part of the **string** before the **substring**, the **substring** itself, and the part of the **string** after the **substring**.

upper() returns a new string with all uppercase characters.

lower() returns a new string with all lowercase characters.

swapcase() returns a new string with all uppercase characters converted to lowercase and vice versa.

title() returns a new string with the first character of each word capitalized and the rest of the characters in lowercase.

```
# Create a sample string
my_string = "Hello World!"

# Length of the string
len_string = len(my_string) # len_string = 12

# Find the first occurrence of a substring in the string
index = my_string.find("World") # index = 6

# Count the number of occurrences of a substring in the string
count = my_string.count("l") # count = 3

# Partition the string around a substring
before, sep, after = my_string.partition("World") # before = "Hello ", sep = "World", after = "!"

# Convert the string to uppercase
uppercase_string = my_string.upper() # uppercase_string = "HELLO WORLD!"

# Convert the string to lowercase
lowercase_string = my_string.lower() # lowercase_string = "hello world!"

# Swap the case of the characters in the string
swapped_string = my_string.swapcase() # swapped_string = "hELLO wORLD!"

# Titlecase the string (capitalize first letter of each word)
title_string = my_string.title() # title_string = "Hello World!"
```