Python Operators:

- The operator can be defined as a symbol which is responsible for a particular operation between two operands.
- Python provides a variety of operators, which are described as follows.

- ✓ Arithmetic operators
- ✓ Comparison operators
- ✓ Assignment Operators
- ✓ Logical Operators
- ✓ Bitwise Operators
- ✓ Membership Operators
- ✓ Identity Operators
- ✓

Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Description
+ (Addition)	It is used to add two operands. For example, if a = 20, b = 10 => a+b = 30
- (Subtraction)	It is used to subtract the second operand from the first operand. If the first operand is less than the second operand, the value results negative. For example, if a = 20, b = 10 => a - b = 10
/ (divide)	It returns the quotient after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a/b = 2.0
*** (Multiplication)**	It is used to multiply one operand with the other. For example, if a = 20, b = 10 => a * b = 200
% (reminder)	It returns the reminder after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a%b = 0
**** (Exponent)**	It is an exponent operator represented as it calculates the first operand power to the second operand.
// (Floor division)	It gives the floor value of the quotient produced by dividing the two operands.

Python Comparison Operators

Comparison operators are used to compare two values:

Operator	Description
==	If the value of two operands is equal, then the condition becomes true.
!=	If the value of two operands is not equal, then the condition becomes true.

<=	If the first operand is less than or equal to the second operand, then the condition becomes true.
>=	If the first operand is greater than or equal to the second operand, then the condition becomes true.
>	If the first operand is greater than the second operand, then the condition becomes true.
<	If the first operand is less than the second operand, then the condition becomes true.

Assignment Operators

The assignment operators are used to assign the value of the right expression to the left operand. The assignment operators are described in the following table.

Operator	Description
=	It assigns the value of the right expression to the left operand.
+=	It increases the value of the left operand by the value of the right operand and assigns the modified value back to left operand. For example, if a = 10, b = 20 => a+ = b will be equal to a = a+ b and therefore, a = 30.
-=	It decreases the value of the left operand by the value of the right operand and assigns the modified value back to left operand. For example, if a = 20, b = 10 => a- = b will be equal to a = a- b and therefore, a = 10.
*=	It multiplies the value of the left operand by the value of the right operand and assigns the modified value back to then the left operand. For example, if a = 10, b = 20 => a* = b will be equal to a = a* b and therefore, a = 200.
%=	It divides the value of the left operand by the value of the right operand and assigns the reminder back to the left operand. For example, if a = 20, b = 10 => a % = b will be equal to a = a % b and therefore, a = 0.
**=	a**=b will be equal to a=a**b, for example, if a = 4, b =2, a**=b will assign 4**2 = 16 to a.
//=	A//=b will be equal to a = a// b, for example, if a = 4, b = 3, a//=b will assign 4//3 = 1 to a.

Python Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Example:

```
if a = 7
  b = 6
then, binary (a) = 0111
   binary (b) = 0110

hence, a & b = 0011
    a | b = 0111
        a ^ b = 0100
    ~ a = 1000
```

Membership Operators

- Python membership operators are used to check the membership of value inside a Python data structure.
- If the value is present in the data structure, then the resulting value is true otherwise it returns false.

Operator	Description
in	It is evaluated to be true if the first operand is found in the second operand (list, tuple, or dictionary).
not in	It is evaluated to be true if the first operand is not found in the second operand (list, tuple, or dictionary).

Identity Operators

The identity operators are used to decide whether an element certain class or type.

Operator	Description
is	It is evaluated to be true if the reference present at both sides point to the same object.
is not	It is evaluated to be true if the reference present at both sides do not point to the same object.

Python Comments

- We might wish to take notes of why a section of script functions, for instance. We leverage the remarks to accomplish this. Formulas, procedures, and sophisticated business logic are typically explained with comments.
- Single-line comments, multi-line comments, and documentation strings are the 3 types of comments in Python.

Below are some of the most common uses for comments:

- Readability of the Code
- Restrict code execution
- Provide an overview of the program or project metadata
- To add resources to the code

Types of Comments in Python

Single-Line Comments

- Single-line remarks in Python have shown to be effective for providing quick descriptions for parameters, function definitions, and expressions.
- A single-line comment of Python is the one that has a hashtag # at the beginning of it and continues until the finish of the line.
- The Python compiler ignores this line.

Code

```
# This code is to show an example of a single-line comment
print( 'This statement does not have a hashtag before it' )
```

Multi Line Comments

- Python does not really have a syntax for multi line comments.
- To add a multiline comment you could insert a # for each line:

```
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

Python Data Types

Built-in Data Types

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`

Sequence Types: `list`, `tuple`, `range`

Mapping Type: `dict`

Set Types: `set`, `frozenset`

Boolean Type: `bool`

Binary Types: `bytes`, `bytearray`, `memoryview`

None Type: `NoneType`

Example	**Data Type**
`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple

`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict
`x = {"apple", "banana", "cherry"}`	set
`x = frozenset({"apple", "banana", "cherry"})`	frozenset
`x = True`	bool
`x = b"Hello"`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview
`x = None`	NoneType

Getting the Data Type

You can get the data type of any object by using the `type()` function:

```
x = 5
print(type(x))
```

Boolean Values

In programming you often need to know if an expression is `True` or `False`.

You can evaluate any expression in Python, and get one of two answers, `True` or `False`.

When you compare two values, the expression is evaluated and Python returns the Boolean answer:

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

Evaluate Values and Variables

The `bool()` function allows you to evaluate any value, and give you `True` or `False` in return,

```
print(bool("Hello"))
print(bool(15))
```