Operators:

Operators are used to perform operations on variables and values.
Python divides the operators in the following groups:

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

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

2.Assignment Operators:
Assignment operators are used to assign values to variables:

- =(Assign the value of right side of the left side operand)
- += (Add AND:Add right side of the operand with left side operand and then sign to left operand)
- -= (Subtract AND: Subtract right operand from left operand and then assign to left operand
- *= (Multiply AND: Multiply right operand with left operand then assign to left operand
- /= (Divide AND:Divide left operand with right operand and then assign to left operand
- %= (Modulus AND:Take modulus using left and right operand and assign the result to left operand)
- //= (Divide AND:Divide left operand with right operand and then assign the value to left operand)
- **= (Exponent AND:Calculate exponent value using operands and assign value to left operand
- &= (Performs Bitwise AND on operands and assign value to left operand)
- != (Performs Bitwise OR on operands and assign value to left operand)
- ^= (Perform Bitwise xOR operator on operand and assign value to left operand)
- >>= (Perform Bitwise right shift on operands and assign value to left operand)
- <<= (Perform Bitwise left shift on operands and assign value to left operand)

3.Comparison Operators:
Comparison operators are used to compare two values:

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

4.Logical Operators:
Logical operators are used to combine conditional statements:

- and (Return true if both the statements are true)
- or (Return true if one of the statements are true)
- not (Reverse the results , returns False if the result is true)

5.Identity Operators:
Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

- is (Return True if both variables are the same objects)
- is not (Return True if both variables are not the same objects)

6.Membership operators:
Membership operators are used to test if a sequence is presented in an object:

- in (Return True if a sequence with the specified value is present in the objects)
- not in (Return True if a sequence with the specified value is not present in the objects)

7.Bitwise Operators:
Bitwise operators are used to compare (binary) numbers:

- & (Bitwise AND)
- | (Bitwise OR)
- >> (Bitwise right shift)
- << (Bitwise left shift)
- ~ (Bitwise NOT)
- ^ (Bitwise XOR)

8.Operator Precedence:
Operator precedence describes the order in which operations are performed.

Expression, Operators and Operands:

Expression:

An expression is a combination of one or more operands, zero or more operators, and zero or more pairs of parentheses.

There are three kinds of expressions:

- An arithmetic expression evaluates to a single arithmetic value.
- A character expression evaluates to a single value of type character.
- A logical and relational expression evaluates to a single logical value.

Operators:

The operators indicate what action or operation to perform.

Operands:

The operands indicate what items to apply the action to. An operand can be any of the following kinds of data items:

- Constant
- Variable
- Array element
- Function
- Substring
- Structured record field (if it evaluates to a scalar data item)
- An expression

Data Types and Data Structures

S.N	Data Types	Data Structures
1.	Data Type is the kind or form of a variable which is being used throughout the program. It defines that the particular variable will assign the values of the given data type only.	Data Structure is the collection of different kinds of data. That entire data can be represented using an object and can be used throughout the entire program.
2.	Implementation through Data Types is a form of abstract implementation.	Implementation through Data Structures is called concrete implementation.
3.	Can hold values and not data, so it is data less	Can hold different kind and types of data within one single object
4.	Values can directly be assigned to the data type variables.	The data is assigned to the data structure object using some set of algorithms and operations like push, pop and so on.
5.	No problem of time complexity	Time complexity comes into play when working with data structures
6.	Examples: int, float, double	Examples: stacks, queues, tree

Set	Tuple	List	Dictionary
x = {1,2,”test”,”class”}	x = (1,2, ”test”, “class”)	x = [1,2,”test”, ”class”]	x = {“mango”:10, “apple”:100, “banana”:200, “kiwi”:2, “grapes”:14}
Nested set: x = {{1,2,”test”,”class”} ,{3,4,5,”hari”,”shyam”} }	Nested tuple: x = ((1,2,”test”,”class”), (3,4,”hari”,”shyam”))	Nested List: x = [[1,2,”test”,”class”], [3,4,”Hari”,”Shyam”]]	Nested Dictionary: x = {‘fruits’:{“mango”:10, “apple”:100, “banana”:200} ‘drinks’:{‘coke’:14, ‘redbull’:30, ‘lassi’:35}}
Set is a mutable because it can be modified, updated and access the elements.	Tuple is a immutable because it can’t modify, update and access the elements.	List is a mutable because it can be modified, updated and access the elements.	Dictionary is a mutable Because it can be modified, updated and access the elements
A Set is represented by { }	A Tuple is represented by ( )	A List is represented by [ ]	A Dictionary is represented by { }