Operators in python:

Python operators are used to perform operations values and variables. We have 7 types of operators:

1. Arithmetic operators
2. Assignment operators
3. Comparison operators
4. Logical operators
5. Identity operators
6. Membership operators
7. Bitwise operators

**Arithmetic operators:** This are used to perform mathematical calculations. This include addition, subtraction, multiplication, division, floor division, modulus, power.

Addition-Adds two operands----(10+3)or (x+y)

Subtraction- Subtracts two operands-------(10-3) or (x-y)

Multiplication- Multiplies two operands----(10\*3) or (x\*y)

Division- Divides the first operand with second operand---(10/2)

Floor Division- It divides the operands and gives the result with nearest

Integer (5//2) o/p—3

Modulus-It is used to find the remainder of first operand when it is divided by

The second operand. (10/3)—1 o/p

Power- It is used to calculate the power of an operator. (2\*\*2---4)

**Assignment operators**: It is used to assign a value to a variable.

A=3

B=10

C=A+B

In this assignment operators we can perform all arithmetic operations.

**Comparison operators**: This operators are used to compare the values of two operands.

|  |  |  |
| --- | --- | --- |
| < | Less than it represents true if left side operand is less than right side operand | 9<15 o/p True |
| > | Greater than It represents true if left side operand is greater than right side. | 10>5 o/p True |
| == | Equals It represents true if both operands are equal | 10==10 o/p True |
| != | Not Equal It represents true if both the operands are not equal | 10 != 15 o/p True |
| >= | Greater than equal represents true if left operand is greater or equal to right operand | 25 >= 23 o/p True |
| <= | Less than equal represents true if left operand is less than or equal to right operand | 25<=44 o/p True |

**Logical operators:** It is used to combine multiple conditions together and make a single boolean expression as output. Three types of logical operators:

And: Returns true if both the conditions are True.

Or: Returns true if any one condition is true.

Not: Returns reciprocal of the expected output.

**Identity operators:** It is used to compare the objects if both the objects are actually of the same data type and share the same memory location.

IS and IS NOT are the operators.

IS : Returns true if both the variables are of the same object.

x = ["apple", "banana"]

y = ["apple", "banana"]

z = x

o/p True because z, x share same object.

IS NOT: Returns true if both variables are not the same object.

x = ["apple", "banana"]

y = ["apple", "banana"]

z = x

print(x is not y)

o/p True.

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

IN and NOT IN are operators.

IN: Returns True if a sequence with the specified value is present in the object

x = ["apple", "banana"]

print("banana" in x) o/p True

NOT IN: Returns True if a sequence with the specified value is not present in

the object.

x = ["apple", "banana"]

print("pineapple" not in x) o/p True

**Bitwise operators:** This operators are used to perform bitwise calculations on integers.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| & | 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 |

**DATA TYPES:**

Python provides various built-in data types to handle different types of data. These are categorized as follows:

1. Numeric Types

Used to store numerical values. There are 3 numeric types in python:

•int: Integer values (e.g., 10, -5)

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

Example:

x=1

print(type(x)) # int

•float: Floating-point numbers (e.g., 10.5, -3.14)

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

Example:

x=10.5

print(type(x)) # float

•complex: Complex numbers (e.g., 3+4j, -2-5j)

Complex numbers are written with a "j" as the imaginary part:

Example:

x=3+5j

Print(type(x)) # complex

2. Sequence Types

Used to store ordered collections:

•str: String of characters (e.g., 'hello', "Python")

Strings are represented in single quotes or double quotes.

Example:

Print(‘Hello’) # Hello

•list: It is a ordered collection (e.g., [1, 2, 3])

List is Mutable. It means after creation of the list, we can perform any kind of operation on it.

List is represented in square brackets [ ].

Example:

x=[1,2,3]

print(x) # [1,2,3]

•tuple: It is a ordered collection (e.g., (4, 5, 6))

Tuple is Immutable. It means after creation of the list, we cannot perform any kind of operation on it.

Tuple is represented in parenthesis ( ).

Example:

x=(1,2,3)

print(x) # (1,2,3)

3. Mapping Type

Used to store key-value pairs:

•Dict: Dictionary (e.g., {'name': 'John', 'age': 30})

It is a collection of key-value pairs.

It is represented in curly braces { }.

Example:

x={'name': 'John', 'age': 30}

Print(x) # {'name': 'John', 'age': 30}

4. Set Types

Used to store unordered collections of unique elements:

•set:It is an unordered collection of elements(e.g., {1, 2, 3})

It is represented in curly braces { }.

Set does not allow duplicate elements.

Set is Mutable, but the elements in the set are Immutable.

Example:

thisset = {"apple", "banana", "cherry"}

print(thisset) # {‘apple’, ’banana’, ’cherry’)

•frozenset: Immutable version of a set (e.g., frozenset([1, 2, 3]))

It is an unordered and unindexed collection of unique elements.

It also does not allow duplicate elements.

Example:

mylist = ['apple', 'banana', 'cherry']

x = frozenset(mylist) #frozenset({‘apple’, ’banana’, ’cherry’})

5. Boolean Type

Represents truth values:

•bool: Can be 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:

Example:

print(10 > 9) # True

print(10 == 9) # False

print(10 < 9) # True

6. Binary Types

Used to store binary data:

•bytes: Immutable sequence of bytes (e.g., b'hello')

Example:

x=b’Hello’

print(x) # b’Hello’

•bytearray: Mutable sequence of bytes (e.g., bytearray ([65, 66, 67]))

Example:

x=bytearray(5)

print(x) # bytearray(b'\x00\x00\x00\x00\x00')

•memoryview: Provides a view of binary data (e.g., memoryview(b'hello'))

Example:

x = memoryview(bytes(5))

print(x) # <memory at 0x00B08FA0>

7. None Type

Represents the absence of a value:

•NoneType: Only one value, None

These data types help in managing and processing data efficiently in Python.