Type Conversion

- Type Conversion is the process of converting a data type into another data type.
- There can be two types of Conversion:

A] Implicit Type conversion

- It is **performed by Python interpreter only.**
- Interpreter **automatically converts one data type to another data type without any user involvement.**
- **Type promotion** that allows performing operations by **converting data into a wider-sized data type without any loss of information.**

```
#Implicit Type Conversion
x = 123
y = 1.23
z = x + y

print("Datatype of x:", type(x))
print("Datatype of y:", type(y))
print("Value of z:", z)
print("Datatype of z:",type(z))

Output:
Datatype of x: <class 'int'>
Datatype of y: <class 'float'>
Value of z: 124.23
Datatype of z: <class 'float'>
```

B] Explicit Type conversion

- It is **performed manually by the user by using the type conversion functions.**
- This Explicit Type Conversion is known as **Type Casting**.
- In Type Casting, loss of data may occur as we enforce the object to a specific data type.

Type conversion functions

- **int(x, base)**: Converts **any data type to integer**. 'Base' specifies that the base **x** is a string.
- **float(x)**: Converts x to a floating-point number.
- **str(x):** Converts object x to a string representation.
- **tuple(x) : Convert X to a tuple.**
- **set(x) : Convert X to a to set.**
- **list(x) : Convert X to a list type**.
- **complex(real,imag) : Converts real numbers to complex number.**
- **dict(d): Convert a tuple of order (key,value) into a dictionary**.

```
# Python code to demonstrate Type conversion
# Using int(), float()
# Initializing string
s = "10010"

# printing string converting to int base 2
a = int(s,2)
print ("After converting to integer base 2 : ", end="")
print (a)

# printing string converting to float
b = float(s)
print ("After converting to float : ", end="")
print (b)
```

Output:

```
After converting to integer base 2 : 18
After converting to float : 10010.0
```

```
# Python code to demonstrate Type conversion
# Using  tuple(), set(), list()
# Initializing string
s = 'machine'

# printing string converting to tuple
c = tuple(s)
print ("After converting string to tuple : ",end="")
print (c)

# printing string converting to set
c = set(s)
print ("After converting string to set : ",end="")
print (c)

# printing string converting to list
c = list(s)
print ("After converting string to list : ",end="")
print (c)
```

Output:

```
After converting string to tuple: ('m', 'a', 'c', 'h', 'i', 'n', 'e')
After converting string to set : {'a', 'm', 'c', 'i', 'h', 'n', 'e'}
After converting string to list : ['m', 'a', 'c', 'h', 'i', 'n', 'e']
```

```
# Python code to demonstrate Type conversion
# using  dict(), complex(), str()
# initializing integers
a = 1
b = 2

# initializing tuple
tup = (('a', 1) ,('f', 2), ('g', 3))

# printing integer converting to complex number
c = complex(1,2)
print ("After converting integer to complex number : ",end="")
print (c)

# printing integer converting to string
c = str(a)
print ("After converting integer to string : ",end="")
print (c)

# printing tuple converting to expression dictionary
c = dict(tup)
print ("After converting tuple to dictionary : ",end="")
print (c)

Output:
After converting integer to complex number : (1+2j)
After converting integer to string: 1
After converting tuple to dictionary : {'a': 1, 'f': 2, 'g': 3}
```

Operators

- Operators are used to perform operations on variables and values.
- The values that the operators work on are called operands.
  - OPERATORS: These are the special symbols. Example -, + , * , /, etc.
  - OPERANDS: These are the value on which the operator is applied.

Types of Operators

In Python, Operators are categorised into following types:

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

Arithmetic Operators

- Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication, and division.

Operator	Description	Syntax
+	**Addition:** adds two operands	x + y
–	**Subtraction:** subtracts two operands	x – y
*	**Multiplication:** multiplies two operands	x * y
/	**Division (float):** divides the first operand by the second	x / y
//	**Division (floor):** divides the first operand by the second	x // y
%	**Modulus:** returns the remainder when the first operand is divided by the second	x % y
**	**Power:** Returns first raised to power second	x ** y

Comparison Operators

Comparison operators are used to compare two values. It either returns **True** or **False** according to the condition.

Operator	Description	Syntax
>	**Greater than:** True if the left operand is greater than the right	x > y
<	**Less than:** True if the left operand is less than the right	x < y
==	**Equal to:** True if both operands are equal	x == y
!=	**Not equal to:** True if operands are not equal	x != y
>=	**Greater than or equal:** to True if the left operand is greater than or equal to the right	x >= y
<=	**Less than or equal:** to True if the left operand is less than or equal to the right	x <= y

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
<<	Left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Logical Operators

Logical operators perform **Logical AND**, **Logical OR**, and **Logical NOT** operations and 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 either 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)

Identity Operators

Identity operators are used to check if two values are located on the same part of the memory. Two variables that are equal do not imply that they are identical.

Operator	Description	Example
is	True if the operands are identical	x is y
is not	True if the operands are not identical	x is not y

Membership Operators

Membership operators are used to test test whether a value or variable is in a Sequence:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Operator Precedence

- Operator precedence determine the priorities of the operator.
  - ✓ Evaluation of the expression is based on precedence of operators.
  - ✓ When an expression contains different kinds of operators, precedence determines which operator should be applied first.
  - ✓ Higher precedence operator is evaluated before the lower precedence operator.
  - ✓ The unary operators need only one operand, and they have a higher precedence than the binary operators.
- The following table lists precedence of all operators from highest to lowest.

Order of Precedence	Operators	Description
1	**	Exponentiation (raise to the power)
2	~ , + , -	Complement, unary plus and unary minus
3	* , / , % , //	Multiply, divide, modulo and floor division
4	+ , -	Addition and subtraction
5	<= , < , > , >=, == , !=	Relational and Comparison operators
6	=, %=, /=, //=, -=, +=, *=, **=	Assignment operators
7	is, is not	Identity operators
8	in, not in	Membership operators
9	not	Logical operators
10	and	
11	or	

Note:

a) Parenthesis can be used to override the precedence of operators i.e. the expression within () is evaluated first.

b) For operators with equal precedence, the expression is evaluated from left to right.