

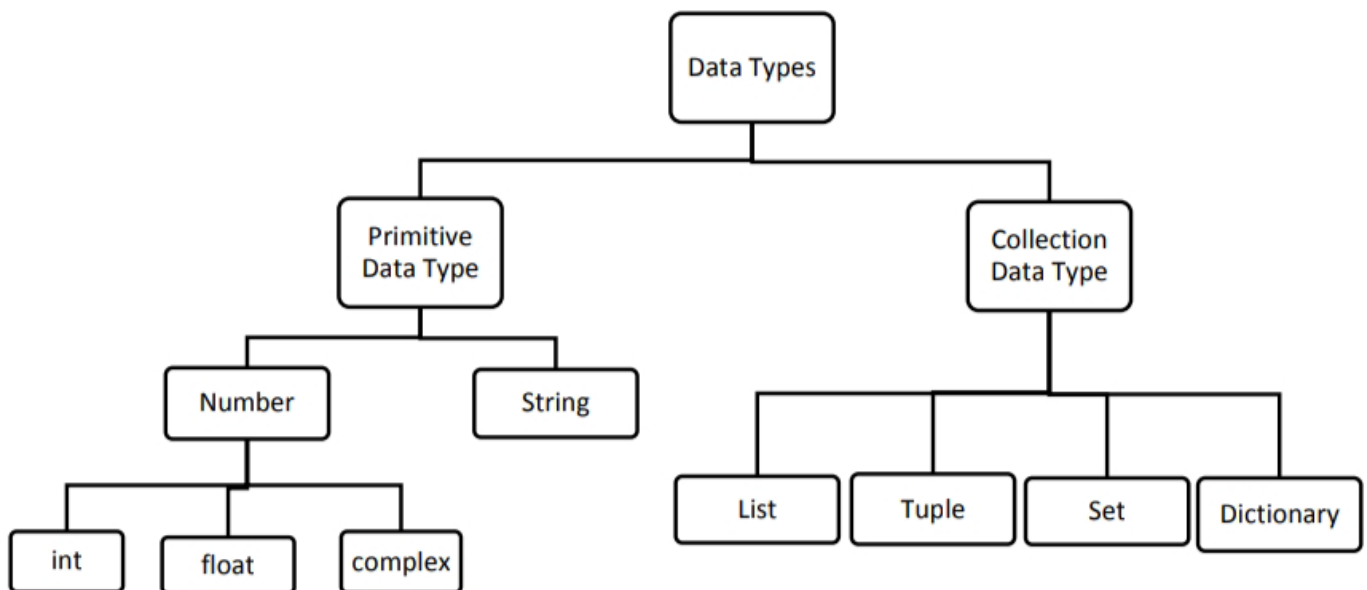
## CHAPTER-3

### DATA HANDLING

#### 3.1 Data Types in Python:

Python has Two data types –

1. Primitive Data Type (Numbers, String)
2. Collection Data Type (List, Tuple, Set, Dictionary)



#### 1. Primitive Data Types:

**a. Numbers:** Number data types store numeric values.

There are three numeric types in Python:

- int
- float
- complex

#### Example:

|         |           |
|---------|-----------|
| w = 1   | # int     |
| y = 2.8 | # float   |
| z = 1j  | # complex |

- **integer** : There are two types of integers in python:

- int
- Boolean

- **int**: int or integer, is a whole number, positive or negative, without decimals.

*Example:*

```
x = 1
y = 35656222554887711
z = -3255522
```

- **Boolean**: It has two values: True and False. **True** has the value **1** and **False** has the value **0**.

*Example:*

```
>>>bool(0)
```

**False**

```
>>>bool(1)
```

**True**

```
>>>bool('')
```

**False**

```
>>>bool(-34)
```

**True**

```
>>>bool(34)
```

**True**

- **float** : float or "floating point number" is a number, positive or negative, containing one or more decimals. Float can also be scientific numbers with an "e" to indicate the power of 10.

*Example:*

```
x = 1.10
y = 1.0
z = -35.59

a = 35e3
b = 12E4
c = -87.7e100
```

- **complex** : Complex numbers are written with a "j" as the imaginary part.

*Example:*

```
>>>x = 3+5j
```

```
>>>y = 2+4j
```

```
>>>z=x+y
```

```
>>>print(z)
```

```
5+9j
```

```
>>>z.real
```

```
5.0
```

```
>>>z.imag
```

```
9.0
```

Real and imaginary part of a number can be accessed through the attributes **real** and **imag**.

**b. String:** Sequence of characters represented in the quotation marks.

- Python allows for either pairs of single or double quotes. Example: 'hello' is the same as "hello" .
- Python does not have a character data type, a single character is simply a string with a length of 1.
- The python string store Unicode characters.
- Each character in a string has its own index.
- String is immutable data type means it can never change its value in place.

## 2. Collection Data Type:

- List
- Tuple
- Set
- Dictionary

### 3.2 MUTABLE & IMMUTABLE Data Type:

#### ➤ Mutable Data Type:

These are changeable. In the same memory address, new value can be stored.

Example: List, Set, Dictionary

#### ➤ Immutable Data Type:

These are unchangeable. In the same memory address new value cannot be stored.

Example: integer, float, Boolean, string and tuple.

### 3.3 Basic Operators in Python:

i. Arithmetic Operators

ii. Relational Operator

iii. Logical Operators

iv. Bitwise operators

v. Assignment Operators

vi. Other Special Operators

- Identity Operators
- Membership operators

i. **Arithmetic Operators:** To perform mathematical operations.

| OPERATOR | NAME             | SYNTAX   | RESULT<br>(X=14, Y=4) |
|----------|------------------|----------|-----------------------|
| +        | Addition         | $x + y$  | 18                    |
| -        | Subtraction      | $x - y$  | 10                    |
| *        | Multiplication   | $x * y$  | 56                    |
| /        | Division (float) | $x / y$  | 3.5                   |
| //       | Division (floor) | $x // y$ | 3                     |
| %        | Modulus          | $x \% y$ | 2                     |
| **       | Exponent         | $x ** y$ | 38416                 |

*Example:*

```
>>>x= -5
>>>x**2
>>> -25
```

- ii. **Relational Operators:** Relational operators compare the values. It either returns **True** or **False** according to the condition.

| OPERATOR | NAME                     | SYNTAX   | RESULT<br>(IF X=16, Y=42) |
|----------|--------------------------|----------|---------------------------|
| >        | Greater than             | $x > y$  | False                     |
| <        | Less than                | $x < y$  | True                      |
| ==       | Equal to                 | $x == y$ | False                     |
| !=       | Not equal to             | $x != y$ | True                      |
| >=       | Greater than or equal to | $x >= y$ | False                     |
| <=       | Less than or equal to    | $x <= y$ | True                      |

- iii. **Logical operators:** Logical operators perform **Logical AND**, **Logical OR** and **Logical NOT** operations.

| OPERATOR | DESCRIPTION                                        | SYNTAX             |
|----------|----------------------------------------------------|--------------------|
| and      | Logical AND: True if both the operands are true    | $x \text{ and } y$ |
| or       | Logical OR: True if either of the operands is true | $x \text{ or } y$  |
| not      | Logical NOT: True if operand is false              | $\text{not } x$    |

**Examples of Logical Operator:**

The **and** operator: The and operator works in two ways:

- Relational expressions as operands
- numbers or strings or lists as operands

**a. Relational expressions as operands:**

| X     | Y     | X and Y |
|-------|-------|---------|
| False | False | False   |
| False | True  | False   |
| True  | False | False   |
| True  | True  | True    |

```
>>> 5>8 and 7>3
False
>>> (4==4) and (7==7)
True
```

**b. numbers or strings or lists as operands:**

In an expression X and Y, if first operand has **false value**, then return first operand X as a result, otherwise returns Y.

```
>>>0 and 0
0
>>>0 and 6
0
>>>'a' and 'n'
'n'
```

| X     | Y     | X and Y |
|-------|-------|---------|
| false | false | X       |
| false | true  | X       |
| true  | false | Y       |
| true  | true  | Y       |

```
>>>6>9 and 'c'+9>5 # and operator will test the second operand only if the first operand
False              # is true, otherwise ignores it, even if the second operand is wrong
```

The **or** operator: The or operator works in two ways:

- Relational expressions as operands
- numbers or strings or lists as operands

**a. Relational expressions as operands:**

| X     | Y     | X or Y |
|-------|-------|--------|
| False | False | False  |
| False | True  | True   |
| True  | False | True   |
| True  | True  | True   |

```
>>> 5>8 or 7>3
True
>>> (4==4) or (7==7)
True
```

***b. numbers or strings or lists as operands:***

In an expression X or Y, if first operand has **true value**, then return first operand X as a result, otherwise returns Y.

| X     | Y     | X or Y |
|-------|-------|--------|
| false | false | Y      |
| false | true  | Y      |
| true  | false | X      |
| true  | true  | X      |

```
>>>0 or 0
```

```
0
```

```
>>>0 or 6
```

```
6
```

```
>>>'a' or 'n'
```

```
'a'
```

```
>>>6<9 or 'c'+9>5 # or operator will test the second operand only if the first operand
```

```
True # is false, otherwise ignores it, even if the second operand is wrong
```

The **not** operator:

```
>>>not 6
```

```
False
```

```
>>>not 0
```

```
True
```

```
>>>not -7
```

```
False
```

Chained Comparison Operators:

```
>>> 4<5>3 is equivalent to >>> 4<5 and 5>3
```

```
True
```

```
True
```

iv. **Bitwise operators:** Bitwise operators acts on bits and performs bit by bit operation.

| OPERATOR | DESCRIPTION | SYNTAX |
|----------|-------------|--------|
| &        | Bitwise AND | x & y  |
|          | Bitwise OR  | x   y  |

|    |                     |       |
|----|---------------------|-------|
| ~  | Bitwise NOT         | ~x    |
| ^  | Bitwise XOR         | x ^ y |
| >> | Bitwise right shift | x>>   |
| << | Bitwise left shift  | x<<   |

### Examples:

Let

a = 10

b = 4

print(a & b)

print(a | b)

print(~a)

print(a ^ b)

print(a >> 2)

print(a << 2)

### Output:

0

14

-11

14

2

40

v. **Assignment operators:** Assignment operators are used to assign values to the variables.

| OPERA<br>TOR | DESCRIPTION                                                                            | SYNTAX         |
|--------------|----------------------------------------------------------------------------------------|----------------|
| =            | Assign value of right side of expression to left side operand                          | x = y + z      |
| +=           | Add AND: Add right side operand with left side operand and then assign to left operand | a+=b<br>a=a+b  |
| -=           | Subtract AND: Subtract right operand from left operand and then assign to left operand | a-=b a=a-<br>b |
| *=           | Multiply AND: Multiply right operand with left operand and then assign to left operand | a*=b<br>a=a*b  |

|     |                                                                                                            |                  |
|-----|------------------------------------------------------------------------------------------------------------|------------------|
| /=  | Divide AND: Divide left operand with right operand and then assign to left operand                         | a/=b<br>a=a/b    |
| %=  | Modulus AND: Takes modulus using left and right operands and assign result to left operand                 | a%=b<br>a=a%b    |
| //= | Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand | a//=b<br>a=a//b  |
| **= | Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand        | a**=b<br>a=a**b  |
| &=  | Performs Bitwise AND on operands and assign value to left operand                                          | a&=b<br>a=a&b    |
| =   | Performs Bitwise OR on operands and assign value to left operand                                           | a =b<br>a=a b    |
| ^=  | Performs Bitwise xOR on operands and assign value to left operand                                          | a^=b<br>a=a^b    |
| >>= | Performs Bitwise right shift on operands and assign value to left operand                                  | a>>=b<br>a=a>>b  |
| <<= | Performs Bitwise left shift on operands and assign value to left operand                                   | a<<=b a=<br>a<<b |

vi. **Other Special operators:** There are some special type of operators like-

- a. **Identity operators-** **is** and **is not** are the identity operators both are used to check if two values are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

**is**            True if the operands are identical  
**is not**      True if the operands are not identical

### Example:

Let

a1 = 3

b1 = 3

a2 = 'PythonProgramming'

b2 = 'PythonProgramming'

a3 = [1,2,3]

b3 = [1,2,3]

print(a1 is not b1)

```
print(a2 is b2)      # Output is False, since lists are mutable.
print(a3 is b3)
```

**Output:**

False

True

False

**Example:**

```
>>>str1= "Hello"
```

```
>>>str2=input("Enter a String :")
```

Enter a String : Hello

```
>>>str1==str2      # compares values of string
```

True

```
>>>str1 is str2     # checks if two address refer to the same memory address
```

False

- b. **Membership operators-** **in** and **not in** are the membership operators; used to test whether a value or variable is in a sequence.

|               |                                            |
|---------------|--------------------------------------------|
| <b>in</b>     | True if value is found in the sequence     |
| <b>not in</b> | True if value is not found in the sequence |

**Example:**

Let

```
x = 'Digital India'
```

```
y = {3:'a',4:'b'}
```

```
print('D' in x)
```

```
print('digital' not in x)
```

```
print('Digital' not in x)
```

```
print(3 in y)
```

```
print('b' in y)
```

## Output:

True

True

False

True

False

## 3.4 Operator Precedence and Associativity:

**Operator Precedence:** It describes the order in which operations are performed when an expression is evaluated. Operators with higher **precedence** perform the operation first.

**Operator Associativity:** whenever two or more operators have the same precedence, then associativity defines the order of operations.

| Operator       | Description                  | Associativity        | Precedence                                                                            |
|----------------|------------------------------|----------------------|---------------------------------------------------------------------------------------|
| ( ), { }       | Parentheses (grouping)       | Left to Right        |  |
| f(args...)     | Function call                | Left to Right        |                                                                                       |
| x[index:index] | Slicing                      | Left to Right        |                                                                                       |
| x[index]       | Subscription                 | Left to Right        |                                                                                       |
| **             | Exponent                     | <b>Right to Left</b> |                                                                                       |
| ~x             | Bitwise not                  | Left to Right        |                                                                                       |
| +x, -x         | Positive, negative           | Left to Right        |                                                                                       |
| *, /, %        | Product, division, remainder | Left to Right        |                                                                                       |
| +, -           | Addition, subtraction        | Left to Right        |                                                                                       |
| <<, >>         | Shifts left/right            | Left to Right        |                                                                                       |
| &              | Bitwise AND                  | Left to Right        |                                                                                       |
| ^              | Bitwise XOR                  | Left to Right        |                                                                                       |
|                | Bitwise OR                   | Left to Right        |                                                                                       |
| <=, <, >, >=   | Comparisons                  | Left to Right        |                                                                                       |
| =, %=, /=, +=  | Assignment                   |                      |                                                                                       |
| is, is not     | Identity                     |                      |                                                                                       |
| in, not in     | Membership                   |                      |                                                                                       |
| not            | Boolean NOT                  | Left to Right        |                                                                                       |
| and            | Boolean AND                  | Left to Right        |                                                                                       |
| or             | Boolean OR                   | Left to Right        |                                                                                       |
| lambda         | Lambda expression            | Left to Right        |                                                                                       |