**Python**

List methods :

A list in Python represents a list of comma-separated values of any data type between square brackets:

var\_name = [element1, element2, ...]

These elements can be of different data types.

**Indexing**

The position of every element placed in the list starts from the 0th position and step by step it ends at length-1 position. List is ordered, indexed, mutable, and the most flexible and dynamic collection of elements in Python.

**Empty List**

This method allows you to create an empty list:

my\_list = []

index()**Method**

Returns the index of the first element with the specified value:

list.index(element)

append()**Method**

Adds an element at the end of the list:

list.append(element)

extend()**Method**

Add the elements of a given list (or any iterable) to the end of the current list:

list.extend(iterable)

insert()**Method**

Adds an element at the specified position:

list.insert(position, element)

pop()**Method**

Removes the element at the specified position and returns it:

list.pop(position)

remove()**Method**

The remove() method removes the first occurrence of a given item from the list:

list.remove(element)

clear()**Method**

Removes all the elements from the list:

list.clear()

count()**Method**

Returns the number of elements with the specified value:

list.count(value)

reverse()**Method**

Reverses the order of the list:

list.reverse()

sort()**Method**

Sorts the list:

list.sort(reverse=True|False)

Tuples : -

**Tuples**

Tuples are represented as comma-separated values of any data type within parentheses.

**Tuple Creation**

variable\_name = (element1, element2, ...)

These elements can be of different data types.

**Indexing**

The position of every element placed in the tuple starts from the 0th position and step by step it ends at length-1 position. Tuples are ordered, indexed, immutable, and the most secure collection of elements.

Let's talk about some of the tuple methods:

count()**Method**

It returns the number of times a specified value occurs in a tuple:

tuple.count(value)

index()**Method**

It searches the tuple for a specified value and returns the position:

tuple.index(value)

**Sets : -**

**Sets**

A set is a collection of multiple values which is both unordered and unindexed. It is written in curly brackets.

**Set Creation: Way 1**

var\_name = {element1, element2, ...}

**Set Creation: Way 2**

var\_name = set([element1, element2, ...])

Set is an unordered, immutable, non-indexed type of collection. Duplicate elements are not allowed in sets.

**Set Methods**

Let's talk about some of the methods of sets:

add()**Method**

Adds an element to a set:

set.add(element)

clear()**Method**

Remove all elements from a set:

set.clear()

discard()**Method**

Removes the specified item from the set:

set.discard(value)

intersection()**Method**

Returns the intersection of two or more sets:

set.intersection(set1, set2 ... etc)

issubset()**Method**

Checks if a set is a subset of another set:

set.issubset(set)

pop()**Method**

Removes an element from the set:

set.pop()

remove()**Method**

Removes the specified element from the set:

set.remove(item)

union()**Method**

Returns the union of two or more sets:

set.union(set1, set2...)

Dictionary: -

**Dictionaries**

The dictionary is an unordered set of comma-separated key:value pairs, within {}, with the requirement that within a dictionary, no two keys can be the same.

**Dictionary**

<dictionary-name> = {<key>: value, <key>: value ...}

Dictionary is an ordered and mutable collection of elements. Dictionary allows duplicate values but not duplicate keys.

**Empty Dictionary**

By putting two curly braces, you can create a blank dictionary:

mydict = {}

**Adding Element to a Dictionary**

By this method, one can add new elements to the dictionary:

<dictionary>[<key>] = <value>

**Updating Element in a Dictionary**

If a specified key already exists, then its value will get updated:

<dictionary>[<key>] = <value>

**Deleting an Element from a Dictionary**

del keyword is used to delete a specified key:value pair from the dictionary as follows:

del <dictionary>[<key>]

**Dictionary Functions & Methods**

Below are some of the methods of dictionaries:

len()**Method**

It returns the length of the dictionary, i.e., the count of elements (key: value pairs) in the dictionary:

len(dictionary)

clear()**Method**

Removes all the elements from the dictionary:

dictionary.clear()

get()**Method**

Returns the value of the specified key:

dictionary.get(keyname)

items()**Method**

Returns a list containing a tuple for each key-value pair:

dictionary.items()

keys()**Method**

Returns a list containing the dictionary's keys:

dictionary.keys()

values()**Method**

Returns a list of all the values in the dictionary:

dictionary.values()

update()**Method**

Updates the dictionary with the specified key-value pairs:

dictionary.update(iterable)

**File**

**File Handling**

File handling refers to reading or writing data from files. Python provides some functions that allow us to manipulate data in the files.

open()**Function**

var\_name = open("file name", "mode")

**Modes**

1. **r** - to read the content from file
2. **w** - to write the content into file
3. **a** - to append the existing content into file
4. **r+** - To read and write data into the file. The previous data in the file will be overridden.
5. **w+** - To write and read data. It will override existing data.
6. **a+** - To append and read data from the file. It won’t override existing data.

close()**Function**

var\_name.close()

read()**Function**

The read functions contain different methods: read(), readline(), and readlines().

read() # return one big string

It returns a list of lines:

readlines() # returns a list

It returns one line at a time:

readline() # returns one line at a time

write()**Function**

This function writes a sequence of strings to the file.

write() # Used to write a fixed sequence of characters to a file

It is used to write a list of strings:

writelines()

**OOPs**

**Object Oriented Programming (OOPS)**

It is a programming approach that primarily focuses on using objects and classes. The objects can be any real-world entities.

class

The syntax for writing a class in Python:

class class\_name: pass # statements

**Creating an Object**

Instantiating an object can be done as follows:

<object-name> =

In Python, \_\_init\_\_ is a **special method** used to **initialize a newly created object**. It’s often called the **constructor**, although technically the object is created before \_\_init\_\_ is called.

class MyClass:

def \_\_init\_\_(self, param1, param2):

self.attr1 = param1

self.attr2 = param2

**Numpy**