List:-

List are just like the arrays declared in other languages . in other languages like c++, arrays are homogenous(homogenous means containing only one type of data, like integer, float Boolean etc.

List need not to be homogenous always which make it a most powerful tool in python , means in list we can store heterogenous datatypes .

List is a collection of things, enclosed in [] and separated by commas.

***List a mutable data type because we can change element of a list in place. In other words , the memory address of a list will not change even after in its values.***

For example:-

L=[1,2,3,”abc”,[1,2,3]]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Creating a list:-**

L=[] # used to create empty list

Or

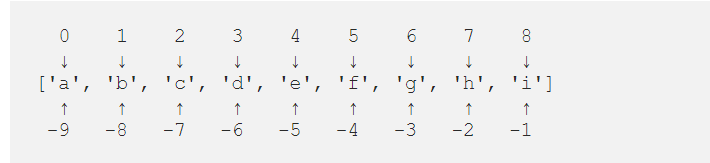
L=list()

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Indexing in list:-**

Lists are “zero indexed”, so [0] returns the zero-th (i.e. the left-most) item in the list, and [1] returns the one-th item (i.e. one item to the right of the zero-th item). Since there are 9 elements in our list ([0] through [8]), attempting to access my\_list[9] throws an IndexError: list index out of range, since it is actually trying to get the tenth element, and there isn’t one.

Python also allows you to index from the end of the list using a negative number, where [-1] returns the last element. This is super-useful since it means you don’t have to programmatically find out the length of the iterable in order to work with elements at the end of it. The indices and reverse indices of my\_list are as follows:



**Note: In negative Indexing the last element is represented by -1 and not -0.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Accessing a list :-**

**We can use the the index operator to access an item in a list . index start from 0 , a list having 5 elements will have index from 0 to 4 .**

**If you trying to access an element other that will raise an Indexerror . the index must be an integer. We can’t use float or other types , this will result into Type Error.**

**For ex:-**

**L=[1,2,3,4,5,”abc”]**

**You want to access 4th item from list:-**

**print(l[3])**

**o/p:- 4**

**Note:- let suppose there is a nested list and you want to access the list item .**

**L= [1,2,3, [1,2,3]]**

**Print(l[3][1])**

**o/p: -2**

**Note :-**

**You can also access the list through negative index.**

**L=[1,2,3,4]**

**print(l[-1])**

**o/p:-4**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**slicing in a list:-**

**slicing is similar to indexing but you can retrieve multiple item in list by separating item numbers with a colon.**

***Syntax:-***

**L[start:stop:step]**

**Start: -start position**

**Stop: - end position**

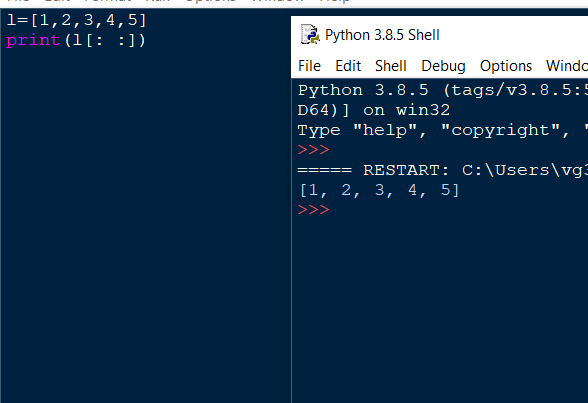
**Step: - increment**

**We will understand the concept of slicing by using a simple example:-**

**L=[1,2,3,4,5]**

**print(L[ : : ]**

**o/p:-[1,2,3,4,5]**

****

**# what happened here how it will print whole the list?**

**By default start index is 0,end is length of list , step is 1**

**That’s why it will print whole the string.**

**Example:-**

**L=[1,2,3,4,5]**

**Print(L[2: :])**

**o/p:-3,4,5**

**Example:-**

**L=[1,2,3,4,5]**

**Print(L [1:3:])**

**o/p:-2,3,4**

**Example:-**

**What happened if you take “end value”more than list size?**

**It will print the items of a list from starting index to end of the index.**

**Example:-**

**L=[1,2,3,4,5,6]**

**Print(l[:50:])**

**o/p:-[1,2,3,4,5]**

**Text

Description automatically generated**

**Example:-**

**L=[1,2,4,4,5]**

**Print(L[: :-1]**

**o/p:-[5,4,4,2,1]**

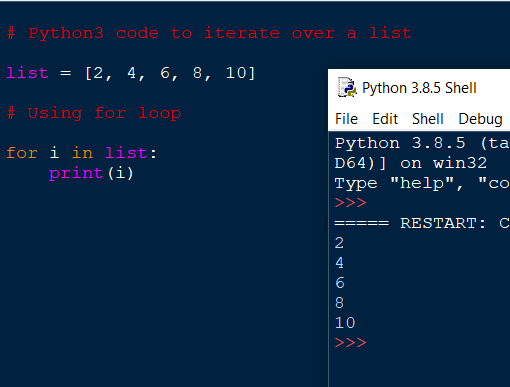
**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Traversing in a list:-**

**Method of Traversing Lists :-**

1. **Using for loop**
2. **Using for loop and range()**
3. **Using while loop**
4. **Using list comprehension**
5. **Using enumerate**

**Using for loop :-**

****

**Using for loop and range():**

**Now the question arise in our mind that what is range function?**

Python range() function generates the **immutable sequence of numbers** starting from the given start integer to the stop integer. The range ()is a built-in function that returns a range object that consists series of integer numbers, which we can iterate using a for loop.

In Python, Using for loop with range(), we can repeat an action a specific number of times.

***Syntax:***

***range(start,stop,step)***

***start* :- Optional. An integer number specifying at which position to start. Default is 0**

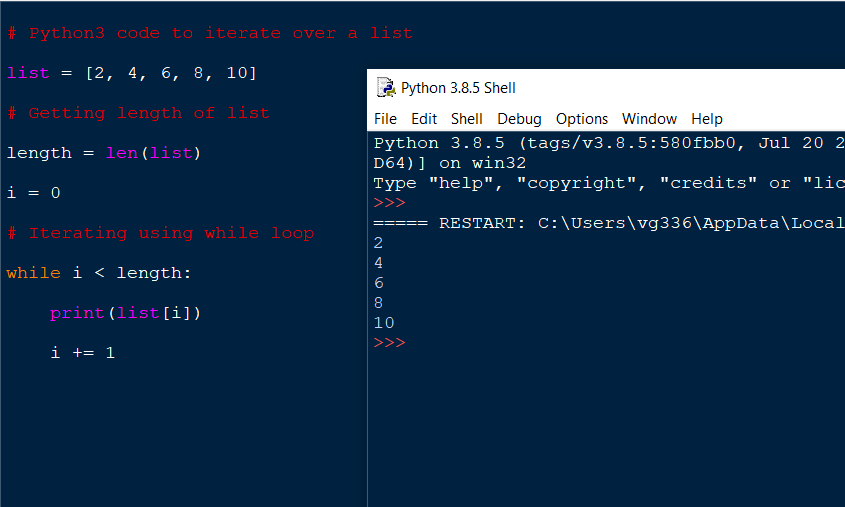
***stop* :- Required. An integer number specifying at which position to stop (not included).**

***Step*:- Optional. An integer number specifying the incrementation. Default is 1**

**Text

Description automatically generated**

**Using while Loop:-**

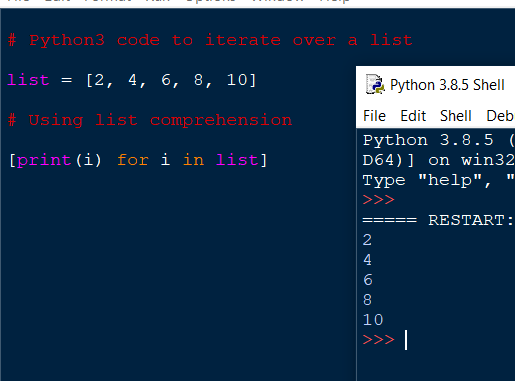
****

**Using List comprehension:-**

**Syntax:-**

***List [expression for variable in sequence]***

Where, the expression is evaluated once, for every item in the sequence. List comprehensions help programmers to create lists in a concise way. This is mainly beneficial to make new lists where each element is the obtained by applying some operations to each member of another sequence or iterable. List comprehension is also used to create a subsequence of those elements that satisfy a certain condition.

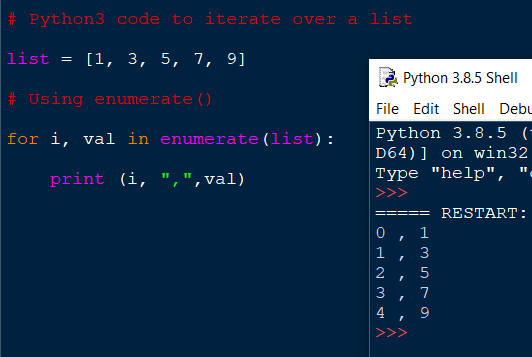


**Using enumerate():-**

If you wish to convert the list into an iterable list of tuples (or get the index on the basis of a condition check, for instance, in linear search, one might want to save the index of minimum element), one can use the enumerate () function.

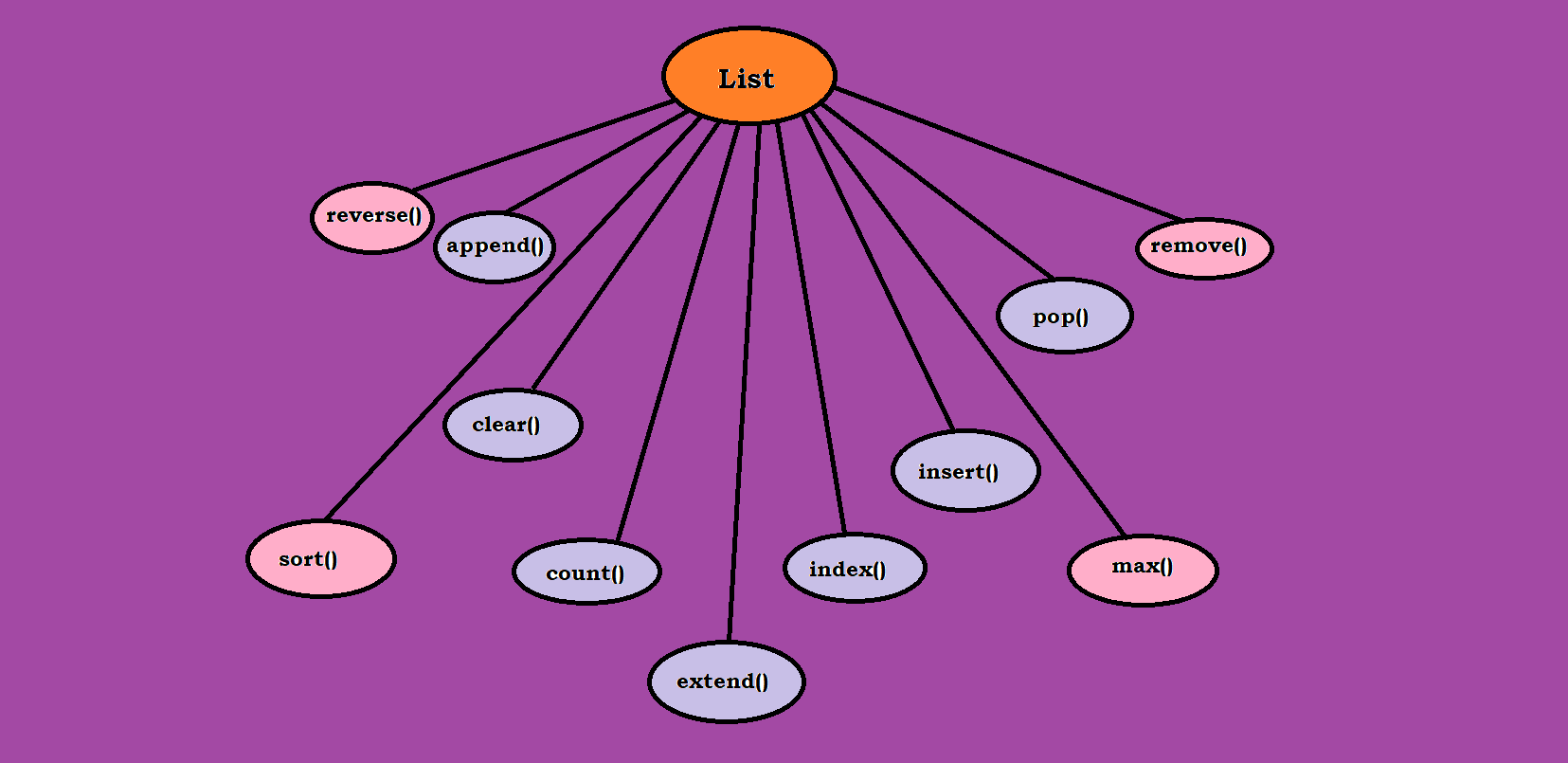
***Syntax:-***

**enumerate(iterable, start=0)**



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**List operation :-**

****

1. **append()**

Used for appending and adding elements to List. It is used to add elements to the last position of the List in Python.

***Syntax:-***

List.append(element)

L=[1,2,3,4]

l.append(5)

print(l)

o/p:- [1,2,3,4,5]

1. **insert()**

Inserts an element at the specified position.

***Syntax:***

**List.insert(<position,element)**

Note: Position mentioned should be within the range of List, as in this case between 0 and 4, elsewise would throw IndexError.

L=[1,2,3,4]

L.insert(2,7)

Print(List)

o/p:-[1,2,7,3,4]

1. **extend()**

Adds contents to List2 to the end of List1.

***Syntax:-***

List1.extend(List2)

L1=[1,2,3]

L2=[2,3,4,5]

L1.extend(L2)

Print(List1)

o/p:-

[1,2,3,2,3,4,5]

1. **pop()**

the index is not necessary parameter , if not mentioned takes the last index.

***Syntax:-***

List.pop([index])

Note:-index must be in range in list , elsewise indexErrors occurs.

L=[1,2,3,4,5,6]

Print(l.pop(4))

o/p:-4

1. **Del()**

Element to be deleted is mentioned using list name and index.

***Syntax:-***

del list.[index]

l=[1,2,3,4,5]

del l[0]

print(l)

o/p:- [2,3,4,5]

1. **Remove()**

Element to be deleted is mentioned using list name and element

***Syntax:-***

List.remove(element)

l=[1,2,3,4,5,6]

l.remove(3)

print(l)

o/p:- [1,2,4,5,6]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Filter():-**

The filter() function constructs a list from those elements of the list for which a function retums True.

***Syntax:-***

**filter(function, sequence)**

As per the syntax, the filter() function returns a sequence that contains items from the sequence for which the function is True. If sequence is a string, Unicode, or a tuple, then the result will be of the same type: otherwise, it is always a list.

Text

Description automatically generated

**Programming tip:-**

Do not add or remove element from the list during iteration.

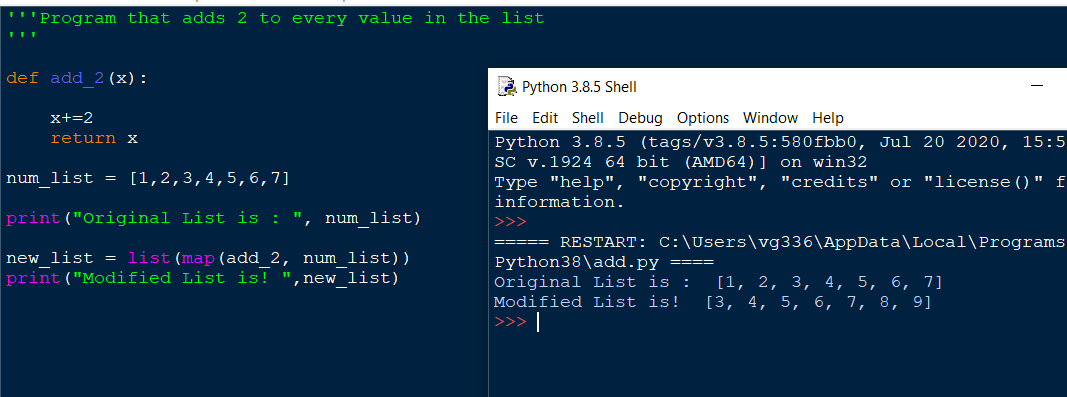
**map():-**

The map() function applies a particular function to every element of a list .its syntax is same as the filter function.

***syntax:-***

**map(func,iter)**

After applying the specified function on the sequence , the map() function returns the modified list. The map() function call function (item) for each item in the sequence and returns a list of the returns values.



Note that in the above code, the map() function calls add\_2() which adds 2 to every value in the list. You can even pass more than one sequence in the map() function. But in this case, remember two things.

1.First, the function must have as many arguments as there are sequences.

2.Second, each argument is called with the corresponding item from each sequence (or None if one sequence is shorter than another).

**Redunce() function :-**

It returns a single value generated by called the function on the first two items of the sequence , then on the result and next item, and so on.

Syntax:-

Reduce(function,sequence)

