CHAPTER 7

LISTS

LEARNING OBJECTIVES

After completing this chapter, the reader will comprehend the context and its associated questions:

- Usage of List
- Learn to create, access, and use lists in Python
- Use built-in methods and functions to manipulate lists and list indices

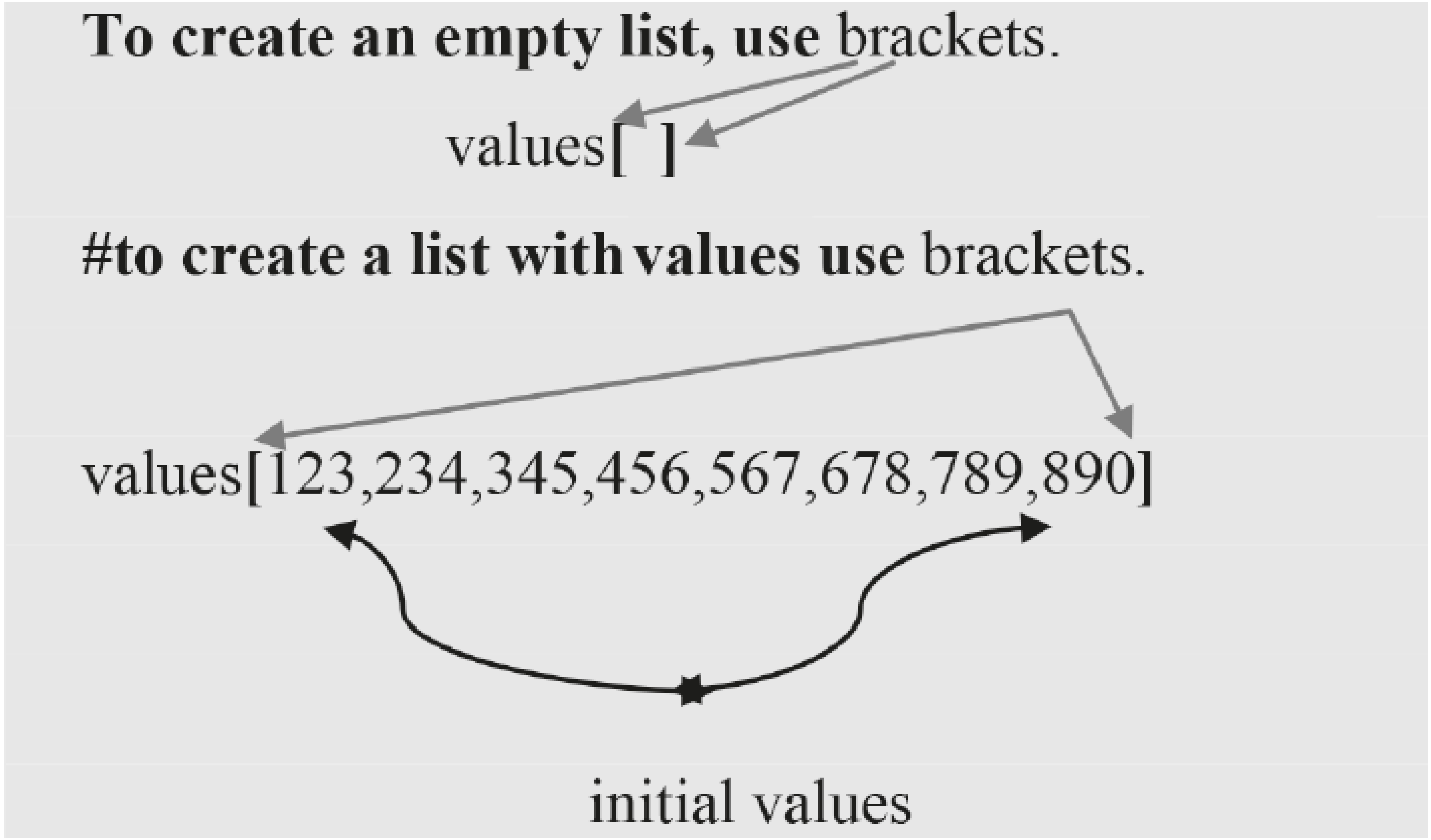
- Know the typical operations performed on lists
- Perform traversals of lists to process items in it.
- Handle nested lists
- Iterative control with Python's "loop" statement and how to make the most of it.

7.2 List

Primary Properties of Lists

- The essential features of Python Lists are as follows:
- A list is mutable, meaning the list's contents may be altered.
- Lists in Python use zero-based indexing.
- Objects of any kind can be found amongst them.

- An index provides quick and easy access to list items.
- There is no cap on the number of sub-lists that can be encapsulated in a single list.
- A list can be of a wide range of sizes.



To create an empty list, use brackets.
values[ ]
#to create a list with values use brackets.
values[123,234,345,456,567,678,789,890]
initial values

Fig. 7.1: Creation of List

Indexing in a LIST

Python Syntax

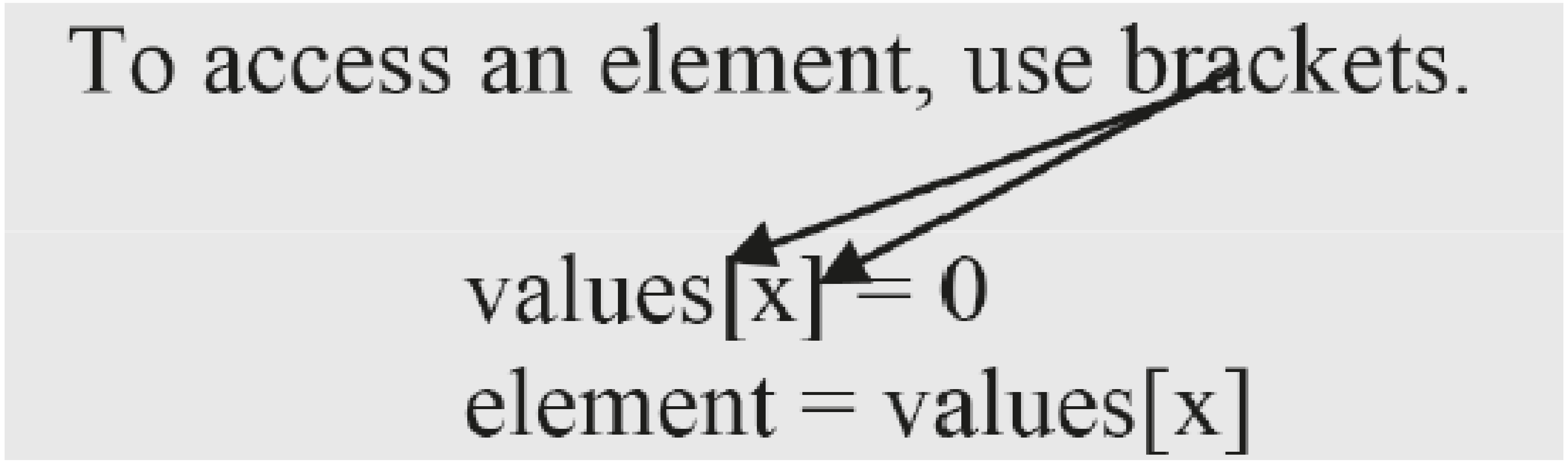
```
[<element>, <element>, ... <element>]
```

Python Syntax

```
name_of_the_list [index of the item to be accessed]
```

I	N	D	E	X	I	N	G
0	1	2	3	4	5	6	7
-8	-7	-6	-5	-4	-3	-2	-1

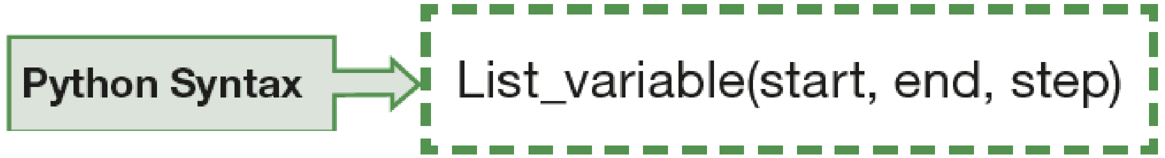
Fig. 7.2: Indexing of List


To access an element, use brackets.

```
values[x] = 0
element = values[x]
```

Fig. 7.3: Accessing values of elements in a list

7.2.3 List Slicing



Two or three parameters can be used in the slicing syntax ([start:end] or [start:end:step])

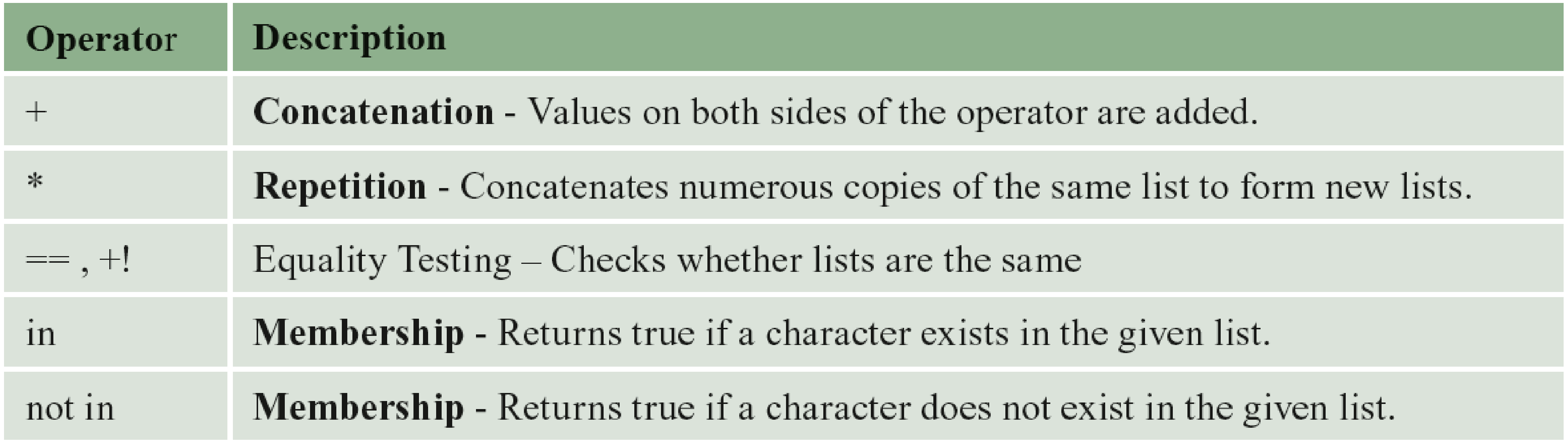
start: The slice's beginning index. The default value is Zero.

end: The slice's final index. The default value is Zero.

step: The items are selected based on the "step" intervals between the beginning and the finish. In this case, it's set to 1.

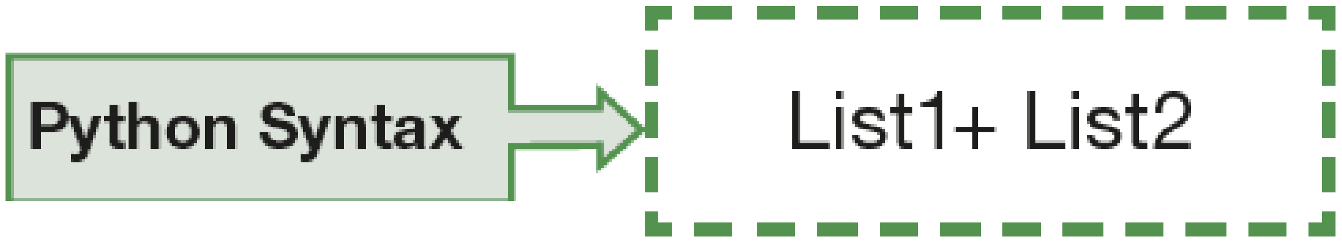
7.3 Operations in the List

Table 7.2: Operators in Python



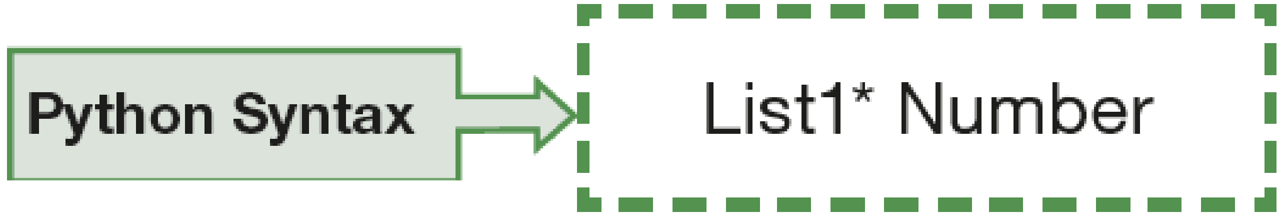
Operator	Description
+	**Concatenation** - Values on both sides of the operator are added.
*	**Repetition** - Concatenates numerous copies of the same list to form new lists.
== , +!	Equality Testing – Checks whether lists are the same
in	**Membership** - Returns true if a character exists in the given list.
not in	**Membership** - Returns true if a character does not exist in the given list.

Concatenation (+) Operator-It is a **binary infix + (plus) operator** regularly overloaded to connote list concatenation. Concatenation of lists means joining the two operands by linking the lists end-to-end. The **syntax for the concatenation (+) operator** is given as :



- If two lists are to be concatenated with a plus (+) operator, then a new list that contains the elements of the first list, followed by the elements of the second list, will be generated.
- Example, to join (concatenate) several lists like the last name, middle name, and first name, together concatenation operators help to do so.

- **Replicate and Repetition** – When dealing with numeric values, the "*" operator has a different purpose than the multiplication operator. With one list and one integer, "*" replicates a list as many times as the integer allows. To put it another way, repetition - generates new lists by appending several instances of the same list. Given below is an expression for the replication (*) operator between two lists:



Membership Operators

- **(i) "in" operator -** This operator determines whether or not an element is contained within a list. If an element is present in the list, true is returned; otherwise, false is returned.
- **(ii) "not in" operator -** This operator determines whether an element is or is not present in the list. If the element is not present in the list, true is returned; otherwise, false is returned.

Table 7.4: Built-in functions in list

Built-In Functions	Description	Examples

min(list_variable)	The string values are equated lexicographically to get the precise result. On the inside, the ASCII values of the characters are equated.	>>> germanSilver_items1= ["copper","zinc"]
		>>> min(germanSilver_items1)
		'copper'

sum(list_variable)	Returns the sum of all the elements in the list.	>>> list1= [999,111,333,888,555,777]
		>>> sum(list1)
		3663

max(list_variable)	Returns the maximum number in the given list.	>>>germanSilver_items1= ["copper","zinc"] >>> max(germanSilver_items1) 'zinc'

len(list_variable)	Returns the finding the length of the given list.	>>> len(germanSilver_items1) 2

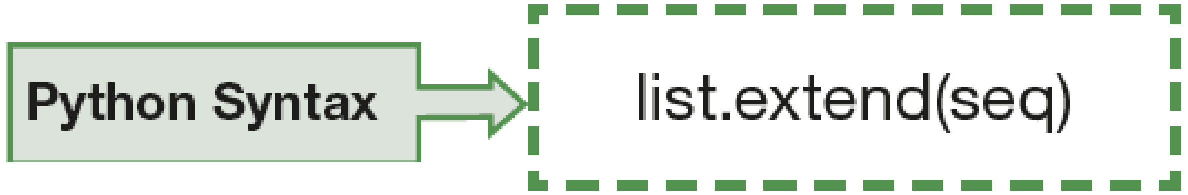
reversed(list_variable)	Returns a list iterator in reverse order of the elements of the given list. list() function is used to convert the iterator as list.	>>> l1 = [4,5,6,8,9,10,11]
		>>> l2_iter = reversed(l1)
		>>> l2 = list(l2_iter)
		>>> l2
		[11, 10, 9, 8, 6, 5, 4]
		>>>

Table 7.3: Built-In Methods in the List

S.No	Built-in Method	Description
1.	list.sort(key)	sort method rearrange the elements in the ascending order (with no prameters or reverse = False) or descending order (with parameter reverse = True)
2.	extend(seq)	This method ends the list with the contents of seq. as it adds multiple items to a list.
3.	insert(index, obj)	This function places an item obj into a list in a given position by adding the element.
4	list.reverse()	all of the items in the list are rearranged, and the iterator's direction is reversed.
5.	count()	The number of entries with the specified value is returned (or) returns the number of times an object appears in the list.

6.	append(obj)	Adds one element to a list with the append() method. (or) object is appended to the list.
7.	clear()	All elements in the list are removed.
8.	del[a : b]	When called, this method removes all elements in the range spanning from index a through index b, as specified by the argument values passed in during execution.
9.	remove(obj item)	Removes the item from the list.
10.	pop(obj=list[-1])	The pop method removes and returns the last element.
11.	index(ele, beg, end)	This method returns the first instance of the given value, or the index at which object is found in the list returned. After the start and before the end, this function returns the index of the first occurrence of an element.

extend(seq)-The list.extend() method in Python outspreads the initial list with all the elements added from a second list. The Python list method extend() keeps adding each element to the list and outspreads the list. The extended list's length will be augmented with the magnitude of the iterable conceded to the extend().

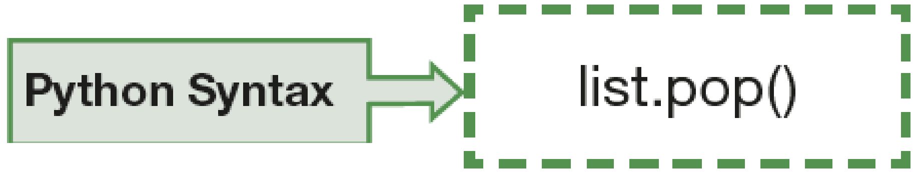


```
list.extend(seq)
```

Deleting an element in a list -Certain application situations demand the deletion of some aspects in a list.

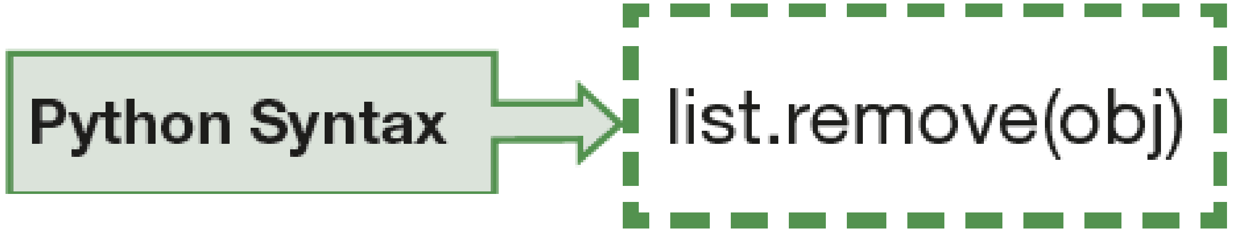
Any one of the following ways can delete the items or elements in a list.

(i) Remove an element within a list using the pop method. If the index is known, it modifies the element in the specified index or removes and returns the last element. The pop() function returns the value of an element at a particular index after it has been removed.

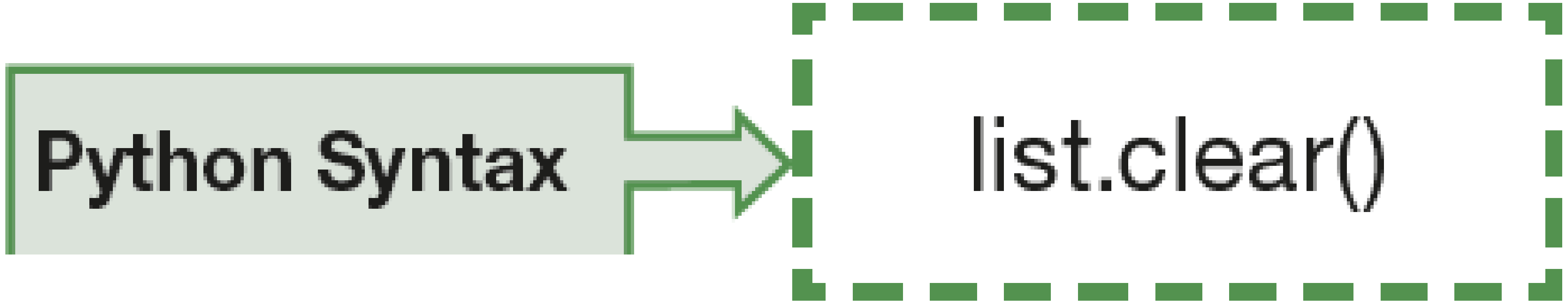


(ii) Eliminate an element within a list using the remove method.

Remove() is a function that removes the first instance of a value in a list.



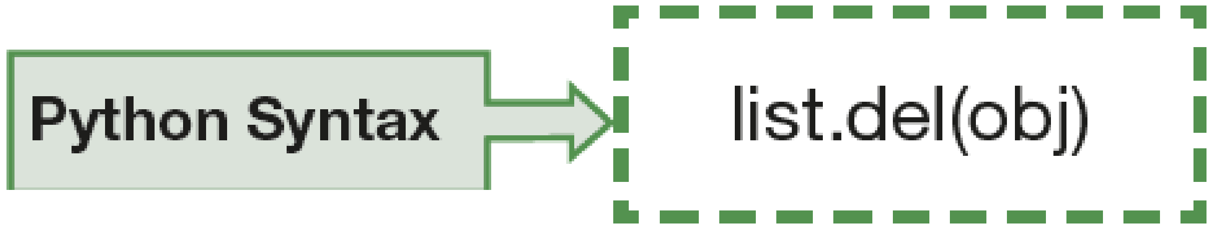
iii) **D**ispels all the elements in the list by clearing all the items from the collection iterable, using the clear method.



```
list.clear()
```

Deleting a single index, a slice of a list, or the entire list is possible using del().To remove more than one element, del with a slice index is better.

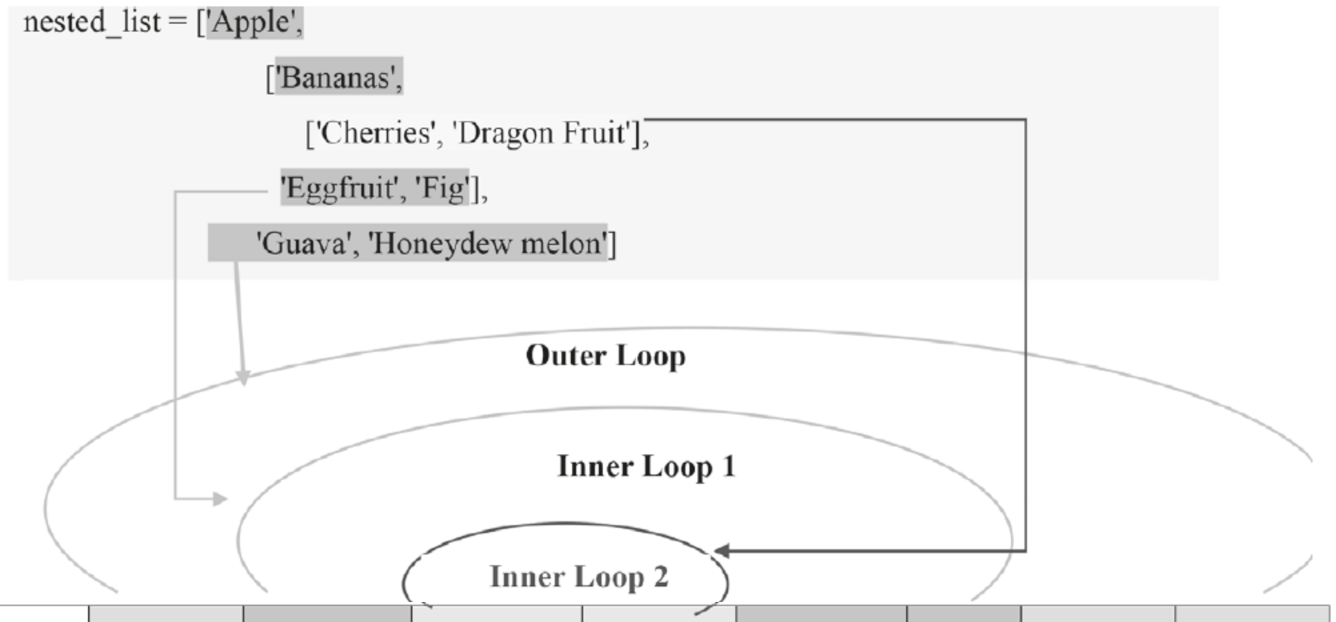
- **del[a:b]** – The list.del() method in Python deletes the elements specified within the (), which mentions the start and end points to be deleted from the iterable.



7.5 Nested List

To Create a Nested List- To create a nested list is to establish a list and then layer one or more other lists inside it. Then, like with ordinary list components, build another nested list by layering two non-empty lists within another list. Nested lists are formed by nesting sub-lists within an additional nested list, which is signalled by using commas to separate the sub-lists within the parent nested list.

- **To Access the Nested List Items by Index -**Finding specific items within a Nested List may be accomplished using its indexes. In a nested list (Refer to Fig. 7.4), to retrieve specific items within each list, use a variety of indexes.



```
nested_list = ['Apple',
                    ['Bananas',
                        ['Cherries', 'Dragon Fruit'],
                      'Eggfruit', 'Fig'],
                  'Guava', 'Honeydew melon']
```

Elements in the list	Apple	Bananas	Cherries	Dragon Fruit	Eggfruit	Fig	Guava	Honey dew Melon
Positive position of the element	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Negative position of the element	**-8**	**-7**	**-6**	**-5**	**-4**	**-3**	**-2**	**-1**

7.6 Mutability of List

Copying a list. Python's copy() function can be used to create a copy of a list identical to the original in terms of content and sorting. The copied list points to a different memory location than the original list. As a result, updating a single list does not affect any other lists.

Python Syntax

```
list.copy(start, end, step)
```

If a new list is created with the = operator, the copy() method makes a shallow copy of the original list, protecting the original from change. The actual list is not mentioned anywhere. A new list reference is also compiled. Use either of the two references to access the most recent version of the list.

Shallow and Deep Copy -A deep copy is used for nested lists.

The duplicated variable has its own distinct memory location in a *Deep Copy*. It is safe to make edits to the deeply copied variable because they will not propagate back to the original.

A *Shallow Copy* is a carbon copy of the original variable. If the new variable is altered, the previous one is also modified.

Shallow And Deep Copies

- Lists can have shallow copies that are precise replicas of the original list's entries. The original list will, thus, reflect any changes made to the entries in the duplicated new list, and vice versa. This makes it easier to maintain both lists. The primary list does not update deep copies; it only updates shallow copies.

- The list is duplicated in deep copies.

The copy module's copy operations include copy() and deepcopy().

Either the = operator or the copy() method can create shallow copies.

Users of the list need to be aware of this behaviour to prevent data modifications

7.7. Looping with lists

Lists are effectively comparable to arrays in other programming languages. They do, however, come with the extra advantage of being size variable. In Python's data structures, the list is of an ampoule type. The method of accessing, one-by-one, the elements of a list is called **a list traversal.**

Looping with lists

It is used to store enormous amounts of data unceasingly.

Python lists are ordered and have a fixed number of elements.

Python has several list traversal methods.

The following methods are available in Python for traversing lists or visiting the elements of a list:

(i) By using a for-loop

(ii) By using a while loop

(iii) Python range () method

(iv) Python enumerate () method

7.8 List as a Stack

Stacks are helpful to

- inverse a string,
- evaluate convert expressions,
- syntax and parenthesis checking,
- Recursive parsing,

7.8 List as a Stack

Stacks are helpful to

- Backtracking,
- histograms,
- tree traversals,
- Tower of Hanoi, and
- topological sorting.

- **Stack performs many operations** like accumulating or appending element; erasure or elimination of elements, and navigating or displaying elements.
- **Accumulating or appending or adding of element:** The insertion of a new element is denoted as "adding" or "appending" an element.

Say, for instance, to add stack instance 5, add it over 4.

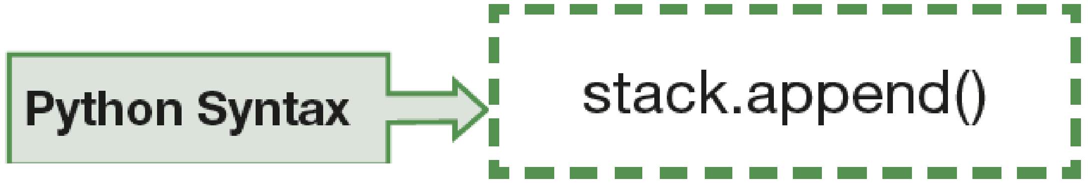
Erasure or elimination of element:

- When the stack is vacant, this does not remove any items
- When the stack has elements, the top element will be eradicated upon encountering this operation. By doing so, the size of the stack can be reduced.
- First, delete stack five and then erase stack 4.

Navigating, displaying or iterating over the stack, each element is displayed on the screen one by one.

Characteristics of stack

- In the operation, the order of inserts is to be kept consistent.
- Everything in stack may be duplicated as often as possible without restriction.
- The same information may be stored in many locations
- This is a complete must-have for anyone who works with data.



7.9 List as Queue

Operations that can be performed on the queue are:

- **Dequeue -** The element to delete in a queue. If the queue is empty, then it is said to be an underflow condition. It removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, it is said to be an Underflow condition.

- **Enqueue -** The element to add to a queue. If the queue is full, then it is said to be an overflow condition. It adds an item to the queue. If the queue is full, it is said to be an overflow condition.
- **Front -** Get the front item from the queue. It gives the front item from the line.

- **IsEmpty -** Check if the queue is empty or not.
- **Peek -** Retrieve the element from the front end of the queue without removing it from the queue.
- **Rear -** Get the last item from the queue. It gives the last item from the queue.
- **Size -** Return the total number of elements present in the queue