List:

```
In Python, a list is a data structure used to store a collection of items. It is a mutable, ordered sequence of elements. Lists are defined using square brackets [], and elements within the list are separated by commas.
Here's a simple example of defining a list in Python:
```

In [6]:
```
L=[1,2,3,4,5]
```

In [7]:
```
print(L)
```

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

```
In this example, "L" is a list containing integers from 1 to 5.
Lists can contain elements of different types, including integers, floats, strings, or even other lists. For example:
```

In [31]:
```
L2=[1, 2, 3, 4, 5, 12.4, 'Zubair', True] #Mixed List
L2
```

Out[31]:
```
[1, 2, 3, 4, 5, 12.4, 'Zubair', True]
```

```
List items are ordered, changeable, and allow duplicate values.
```

In [30]:
```
l=[4,2,6,9,"zub",9,6,6,6,True]
l
```

Out[30]:
```
[4, 2, 6, 9, 'zub', 9, 6, 6, 6, True]
```

```
When we say that lists are ordered, it means that the items have a defined order, and that order will not change.
If you add new items to a list, the new items will be placed at the end of the list.
```

```
Note: There are some list methods that will change the order, but in general: the order of the items will not change.
The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.
```

Nested List:

```
A nested list refers to a list that contains other lists as its elements. These inner lists can themselves contain any data type, including integers, strings, floats, or even other nested lists. Essentially, it's a list within a list.
```

```
Nested lists are a useful feature in Python and are commonly used in situations where you need to organize data hierarchically. They allow for the representation of more complex data structures, such as matrices, tables, or trees.
```

In [10]:
```
L3 = [[1, 2, 3], ["a", "b", "c"], [True, False]] #nested List
```

In [11]:
```
L3
```

Out[11]:
```
[[1, 2, 3], ['a', 'b', 'c'], [True, False]]
```

In [5]:
```
l2=[1,2,3,4,56,6,8,9]
l2
```

Out[5]:
```
[1, 2, 3, 4, 56, 6, 8, 9]
```

In [17]:
```
#if you want to check the datatype of list then simply type
print(type(L))
len(L)
```

```
<class 'list'>
```

Out[17]:
```
(5, 8, 3)
```

In [21]:
```
#if you want to check the length of list
print(len(L)) # or simply type len(L)

```

```
5
```

In [23]:
```
L  #you can call any variable anywhere ones it is declared
```

Out[23]:
```
[1, 2, 3, 4, 5]
```

Indexing:

```
You can access elements within a list using indexing. Indexing starts from 0 for the first element, 1 for the second, and so on. Negative indices can be used to access elements from the end of the list.
Indexing in Python lists allows you to access individual elements within the list. Here's how indexing works:
Positive Indexing: Positive indexing starts from 0 for the first element, 1 for the second element, and so on. You can access elements from the beginning of the list using positive indices.
Negative Indexing: Negative indexing starts from -1 for the last element, -2 for the second-to-last element, and so on. You can access elements from the end of the list using negative indices.
Here's an example of indexing in a list:
```

```
my_list = ['a', 'b', 'c', 'd', 'e']

# Positive indexing
print(my_list[0])  # Output: 'a'
print(my_list[2])  # Output: 'c'
print(my_list[4])  # Output: 'e'
```

```
# Negative indexing
print(my_list[-1]) # Output: 'e'
print(my_list[-3]) # Output: 'c'
print(my_list[-5]) # Output: 'a'
```

```
You can also use indexing to access elements within nested lists:
```

```
nested_list = [[1, 2, 3], ['a', 'b', 'c'], [True, False]]

print(nested_list[0][1])  # Output: 2
print(nested_list[1][2])  # Output: 'c'
print(nested_list[-1][-1]) # Output: False
```

SLICING:

```
Slicing in Python lists allows you to extract a portion of the list.
It's done by specifying a range of indices indicating the start, stop,
and step of the slice. The syntax for slicing is
list[start:stop:step]. Here's what each part means:

start: The index where the slice begins (inclusive).
stop: The index where the slice ends (exclusive).
step: The increment between each index. If omitted, it defaults to 1.
```

In [35]:
```
l=[1,4,3,2,6,7,"Zubair",20.1]
l[1:5] #Get elements from index 1 to index 5 (exclusive)
```

Out[35]: [4, 3, 2, 6]

In [36]:
```
l[0:100] #Get elements from index 1 to index n-1 i:e (last element)
```

Out[36]: [1, 4, 3, 2, 6, 7, 'Zubair', 20.1]

In [37]:
```
l[2:5] # Get elements from index 2 to index 5 (exclusive)
```

Out[37]: [3, 2, 6]

In [38]:
```
l[0:6:2] # Get elements from index 0 to index 6 (exclusive) with a s
```

Out[38]: [1, 3, 6]

In [ ]:
```
l[-3:] # Get the last three elements of the list
```

In [39]:
```
l[2:] # Get all elements except the first two
```

Out[39]: [3, 2, 6, 7, 'Zubair', 20.1]

In [40]:
```
l[::-1] # Reverse the list
```

Out[40]: [20.1, 'Zubair', 7, 6, 2, 3, 4, 1]

List Methods:

```
Lists support various operations such as appending, extending, removing, slicing, and more, making them versatile data structures in Python.
```

```
1. append()  #Adds an element at the end of the list
```

In [42]:
```
l.append("Fayaz")
l
```

Out[42]: [1, 4, 3, 2, 6, 7, 'Zubair', 20.1, 'Fayaz', 'Fayaz']

```
2. clear() #Removes all the elements from the list
```

In [45]:
```
l.clear()
l
```

Out[45]: []

In [48]:
```
l=[1, 4, 3, 2, 6, 7, 'Zubair', 20.1, 'Fayaz', 'Fayaz']
```

In [49]:
```
l.copy() #Returns a copy of the list
```

Out[49]: [1, 4, 3, 2, 6, 7, 'Zubair', 20.1, 'Fayaz', 'Fayaz']

In [50]:
```
l.count(6) #Returns the number of elements with the specified value
```

Out[50]: 1

In [53]:
```
# count() Returns the number of elements with the specified value
l1=[2,1,3,2,3,2,"apple"]
l2=[3,2,5,4,6,7,"banana"]
l1.extend(l2)
l1
```

Out[53]: [2, 1, 3, 2, 3, 2, 'apple', 3, 2, 5, 4, 6, 7, 'banana']

In [55]:
```
# index()Returns the index of the first element with the specified v
l1.index(3)
```

Out[55]: 2

In [57]:
```
# insert() Adds an element at the specified position
l1.insert(3,"orange")
l1
```

Out[57]: [2, 1, 3, 'orange', 'orange', 2, 3, 2, 'apple', 3, 2, 5, 4, 6, 7, 'bana
na']

In [58]:
```
#pop() Removes the element at the specified position
l1.pop()
```

Out[58]:
```
'banana'
```

In [64]:
```
#remove() Removes the first item with the specified value
l1.remove("orange")
l1
```

Out[64]:
```
[2, 1, 3, 2, 3, 2, 'apple', 3, 2, 5, 6, 7]
```

In [66]:
```
# reverse() Reverses the order of the list
l1.reverse()
l1
```

Out[66]:
```
[2, 1, 3, 2, 3, 2, 'apple', 3, 2, 5, 6, 7]
```

In [72]:
```
l=[7,3,4,2,9,0,1,2]
l.sort()
print(l)
```

```
[0, 1, 2, 2, 3, 4, 7, 9]
```