Introduction:-

The list is the most versatile python data structure and stores an ordered sequence of elements just like your shopping list or to-do-list.In Python, lists are mutable, meaning that the elements can be altered unlike tuples or even strings. These elements of a list are called items and can be any data type.

Creating List:-

Creating lists in python is quite simple, use square brackets [] and separate items in the list with commas.
A list can be empty or contain any number of items with different data types such as integers, float, strings, etc.
Example:-

```
mylist = [ ]  #empty list
mylist2 = [1,2]  # list containing 2 items with integers
mylist3 = [1, 2, 'hello']  # list with 3 items,mixed data types,integers and a string
print(mylist)
print(mylist2)
print(mylist3)
```

```
[]
[1, 2]
[1, 2, 'hello']
```

Note: a list can also hold another list as an item, these are called nested lists.
Example:-

```
mylist4 = [['hello','world'],[1,2,3]]  # a nested list
print(mylist4)
```

```
[['hello', 'world'], [1, 2, 3]]
```

Accessing items in a list:-

We can access items in a list by

1. Indexing:-
   We use the index operator [ ]. Each item in a list has an assigned index value. It is important to **Note** that the index starts from 0 in python and must be an integer.
   Note: The first item of any non-empty list is [0].
   Example:-

   ```
   Languages = ['java', 'python', 'c', 'c++']  # define a list
   print(Languages[0]) # Access the first item of a list at index 0
   ```

   ```
   java
   ```

2. Negative Indexing:-

Python also supports negative indexing. The negative indexing is useful when you want to get the last item in a list because it starts accessing a list from the end.
Note: The last item of any non-empty list is [-1].
Example:-
Items:- java, python, c, c++
Indexes:- -4 -3 -2 -1

```
Languages = ['java', 'python', 'c' ,'c++']  # define a list
print(Languages[-1]) #Access the last item of a list
```

```
C++
```

Note: Trying to access indexes other than these will raise an IndexError. The index must be an integer. We can't use float or other types, this will result in TypeError.

How to change or add elements to a list:-

Lists are mutable, meaning their elements can be changed, unlike string or tuple.
We can use the assignment operator (=) to change an item or a range of items.

```
# Correcting mistake values in a list
odd = [2, 4, 6, 8]

# change the 1st item
odd[0] = 1

print(odd)

# change 2nd to 4th items
odd[1:4] = [3, 5, 7]

print(odd)
```

```
[1, 4, 6, 8]
[1, 3, 5, 7]
```

We can add one item to a list using the append() method or add several items using the extend() method.

```
# Appending and Extending lists in Python
odd = [1, 3, 5]

odd.append(7)

print(odd)

odd.extend([9, 11, 13])

print(odd)
```

```
[1, 3, 5, 7]
[1, 3, 5, 7, 9, 11, 13]
```

We can also use the '+' operator to combine two lists. This is also called concatenation.
The '*' operator repeats a list for the given number of times.

```
# Concatenating and repeating lists
odd = [1, 3, 5]

print(odd + [9, 7, 5])

print(["Yes"] * 3)
```

```
[1, 3, 5, 9, 7, 5]
['Yes', 'Yes', 'Yes']
```

Furthermore, we can insert one item at a desired location by using the method insert() or insert multiple items by squeezing it into an empty slice of a list.

```
# Demonstration of list insert() method
odd = [1, 9]
odd.insert(1,3) #for inserting 1 item

print(odd)

odd[2:2] = [5, 7] #for inserting multiple items

print(odd)
```

```
[1, 3, 9]
[1, 3, 5, 7, 9]
```

How to delete or remove elements from a list:-

We can delete one or more items from a list using the keyword ‘del’. It can even delete the list entirely.

```
# Deleting list items
my_list = [1, 2, 3, 4, 'hello', 6.5, 6.78]

# delete one item
del my_list[2]

print(my_list)

# delete multiple items
del my_list[1:5]

print(my_list)
```

```
[1, 2, 4, 'hello', 6.5, 6.78]
[1, 6.78]
```

We can use remove() method to remove the given item or pop() method to remove an item at the given index.

The pop() method removes and returns the last item if the index is not provided. This helps us implement lists as stacks (first in, last out data structure).

We can also use the clear() method to empty a list.

Length of list:-

Python has useful in-built functions that work with list. len() function helps us in returning the total number of elements in a list.

```
Languages = ['java', 'python', 'c' ,'c++']   # define a list
print(len(Languages)) # print the length of the list
```

```
4
```

For more in-built functions you can refer to the following link:-
https://docs.python.org/3/tutorial/datastructures.html

Slicing of a List:-

Slicing is good for getting a subset of items in a list. It uses the slicing operator : (colon) to extract part of the sequence.

[start : stop : steps]

which means that slicing will start from the index **start** will go up to **stop**(excluding stop) in the step of **steps**. The default value of start is 0, stop is the last index of the list and for step, it is 1

Examples:-

```
Languages = ['java', 'python', 'c','c++', 'ruby' ] # define a list
# [:3] slicing everything up to but not including index 3
print(Languages[:3])
```

```
['java', 'python', 'c']
```

```
Languages = ['java', 'python', 'c','c++', 'ruby' ] # define a list
# [3:] slicing everything from index 3 to the last item
print(Languages[3:])
```

```
['c++', 'ruby']
```

```
Languages = ['java', 'python', 'c','c++', 'ruby' ] # define a list
#elements from beginning to end
print(Languages[:])
```

```
['java', 'python', 'c', 'c++', 'ruby']
```

```
Languages = ['java', 'python', 'c','c++', 'ruby' ] # define a list
# elements from 1st to 3rd
print(Languages[0:3])
```

```
['java', 'python', 'c']
```

```
Languages = ['java', 'python', 'c','c++', 'ruby' ] # define a list
# elements in reverse order
print(Languages[::-1])
```

```
['c++', 'c', 'python', 'java']
```

```
Languages = ['java', 'python', 'c','c++', 'ruby' ] # define a list
# elements from -1 to -5 in reverse order with step 2.
print(Languages[-1:-5:-2])
# Output is ['ruby', 'c'] as it excludes the stop index that is -5
```

```
['ruby', 'c']
```

List Comprehension:-

List comprehension is generating a list based on an existing list. It provides an elegant way to define and create new lists based on existing lists.
Example:-

```
Languages = ['java', 'python', 'c','c++', 'ruby' ]
print(Languages)
Languages1 = [x.upper() for x in Languages]
print(Languages1)
```

```
['java', 'python', 'c', 'c++', 'ruby']
['JAVA', 'PYTHON', 'C', 'C++', 'RUBY']
```

For more information about slicing of list and list comprehension, you can refer to the following links:-

1. https://www.geeksforgeeks.org/python-list-comprehension-and-slicing/

2. https://www.tutorialspoint.com/python-list-comprehension-and-slicing
3. https://www.programiz.com/python-programming/list-comprehension

Conclusion

Now you know what lists are and how to manipulate them. For further understanding on the list, you can refer to the following links:-

1. https://docs.python.org/3/tutorial/datastructures.html

Video Link:-

1. https://www.youtube.com/watch?v=Eaz5e6M8tL4