Lists

What is List?

- List is a datatype where u can store multiple values.
- List items are ordered, changeable, and allow duplicate values.
- List items are indexed, the first item has index `[0]`, the second item has index `[1]` etc.

List vs Array

- Array is homogenous and list is heterogenous & Homogenous
- Array store values in continuous memory. List doesn't do it.
- Arrays are much faster because of continuous memory
- List are more programmer-friendly

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
# ['apple', 'banana', 'cherry']
```

List - Assigning, 2D

`variable = []` # This is how u make empty list

```
# 2D List

L1 = [1,2,3,[4,5]]

# 3D List

L2 = [[ [1, 2], [3, 4]], [[5, 6], [7, 8] ]]
```

Accessing

```
L1 = [1, 2, 3, [4, 5]]

L1[0] # 1
L1[2] # 3
L1[-1] # [4,5]
L1[-1:-5:-1] # [[4, 5],3,2,1]
L1[-1][0] # 4

L2 = [[ [1, 2], [3, 4]], [[5, 6], [7, 8] ]]

L2[1][1][0] # 7
```

```
# Checking item if exist

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

for element in thislist:
    if isinstance(element, list):  # Check if the element is a list
        if 4 in element:
            print("4 is found within a sublist")
            break
```

`isinstance()`

The `isinstance()` function returns `True` if the specified object is of the specified type, otherwise `False`.

💡 `isinstance(` *`object`* `,` *`type`* `)`

Editing

```
L = [0,1,2,3,4]

L[0] = 100 # [100,1,2,3,4]

L[-1] = 500 # [100,1,2,3,500]

L[1:4] = [200, 300, 400] # [100,200,300,400,500]
```

💡 `Note` : The length of the list will change when the number of items inserted does not match the number of items replaced.

```
thislist = ["apple", "banana", "cherry"]
thislist[1:2] = ["blackcurrant", "watermelon"]
print(thislist) # ["apple", "blackcurrant", "watermelon", "cherry"]
```

💡 If you insert *less* items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly

```
thislist = ["apple", "banana", "cherry"]
thislist[1:3] = ["watermelon"]
print(thislist) # ["apple", "watermelon"]
```

Adding Items

Append

To add an item to the end of the list, use the `append()` method:

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist) # ['apple', 'banana', 'cherry', 'orange']
```

Extend

To append elements from *another list* to the current list, use the `extend()` method.

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
```

```
print(thislist)
# ['apple', 'banana', 'cherry', 'mango', 'pineapple', 'papaya']

thislist.extend(tropical[0:2])
print(thislist)
#['apple', 'banana', 'cherry', 'mango', 'pineapple']
```

```
#Extend try to append multiple items, so this string first convert into
#list then got appended in list L

L = [1,2,3]
L.extend('adi') # [1,2,3,'a','d','i']
```

💡 The `extend()` method does not have to append *lists*, you can add any iterable object (tuples, sets, dictionaries etc.).

```
thislist = ["apple", "banana", "cherry"]
thistuple = ("kiwi", "orange")
thislist.extend(thistuple)
print(thislist) # ['apple', 'banana', 'cherry', 'kiwi', 'orange']
```

Insert

To insert a list item at a specified index, use the `insert()` method.

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist) # ["apple", "orange", "banana", "cherry"]
```

Delete

Del

The `del` keyword delete the list completely.

The `del` keyword also removes the specified index:

```
thislist = ["apple", "banana", "cherry"]
del thislist
#NameError: name 'thislist' is not defined
```

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
#['banana', 'cherry']
```

Remove

The `remove()` method removes the specified item.

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
#['banana', 'cherry']
```

💡 If there are more than one item with the specified value, the `remove()` method removes the first occurrence:

```
thislist = ["apple", "banana", "cherry", "banana", "kiwi"]
thislist.remove("banana")
print(thislist)
# ["apple", "cherry", "banana", "kiwi"]
```

Pop

The `pop()` method removes the specified index.

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
#['apple', 'cherry']
```

💡 If you do not specify the index, the `pop()` method removes the last item.

```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
# ['apple', 'banana']
```

Clear

The `clear()` method empties the list.
The list still remains, but it has no content.

```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
# []
```

Operation

Loop through a List

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)

# apple
# banana
# cherry
```

Loop through the Index Numbers

```
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
  print(thislist[i])

# apple
# banana
# cherry
```

Looping Using List Comprehension

```
thislist = ["apple", "banana", "cherry"]
[print(x) for x in thislist]
```

List Comprehension

The return value is a new list, leaving the old list unchanged.

💡 `newlist = [` `expression` `for` `item` `in` `iterable` `if` `condition` `== True` `]`

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist
```

Expression

```
newlist = [x.upper() for x in fruits]
```

```
newlist = ['hello' for x in fruits]
# ["hello", "hello", "hello", "hello", "hello"]
# "Return the item if it is not banana,
#  if it is banana return orange".
newlist = [x if x != "banana" else "orange" for x in fruits]
```

Sort

List objects have a `sort()` method that will sort the list alphanumerically, ascending, by default

`sort()` function only work with homogenous list.

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]

thislist.sort()

print(thislist)

# ['banana', 'kiwi', 'mango', 'orange', 'pineapple']

thislist = [100, 50, 65, 82, 23]
thislist.sort()
print(thislist)
# [23, 50, 65, 82, 100]
```

Sort Descending

`reverse = True`

```
thislist = [100, 50, 65, 82, 23]
thislist.sort(reverse = True)
print(thislist)
# [100, 82, 65, 50, 23]
```

Case Sensitive Sort

By default the `sort()` method is case sensitive, resulting in all capital letters being sorted before lower case letters

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort()
print(thislist) # ['Kiwi', 'Orange', 'banana', 'cherry']
```

Customize Sort Function

You can also customize your own function by using the keyword argument `key =` *`function`*.

```
def myfunc(n):
  return abs(n - 50)

thislist = [100, 50, 65, 82, 23]
```

```
thislist.sort(key = myfunc)
print(thislist) # [50, 65, 23, 82, 100]
```

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort(key = str.lower)
print(thislist) # ['banana', 'cherry', 'Kiwi', 'Orange']
```

Reverse Order

What if you want to reverse the order of a list, regardless of the alphabet?

The `reverse()` method reverses the current sorting order of the elements.

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.reverse()
print(thislist) # ['cherry', 'Kiwi', 'Orange', 'banana']
```

Copy List

You cannot copy a list simply by typing `list2 = list1`, because: `list2` will only be a *reference* to `list1`, and changes made in `list1` will automatically also be made in `list2`.

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
# ['apple', 'banana', 'cherry']
```

Join values of List

Join

- This method is specifically designed for joining strings in a list.
- It takes a separator as an argument and concatenates all the strings in the list, separated by the specified separator.

```
my_list = ["apple", "banana", "cherry"]
joined_string = " ".join(my_list)
print(joined_string)  # Output: apple banana cherry
```