Lists

Lists are collection of data enclosed within [ ] and separated by commas.

The items in a list have a defined order, and that order will not change.

In [34]:
```
# let's create a "python list" fruits

fruits = ["apple", "banana", "strawberry"]
# list containing strings
```

In [35]:
```
# len() function allows to count the number of items present in the list
length_of_fruits = len(fruits)
print(f"there are {length_of_fruits} items in the above list.")
```

```
there are 3 items in the above list.
```

In [36]:
```
# executing a list
print(fruits)
```

```
['apple', 'banana', 'strawberry']
```

In [37]:
```
# can also execute list using "for" loop
for i in fruits:
    print(i, end=" ")

# above we found 3 items
```

```
apple banana strawberry
```

List Indexing

accessing elements in the lists can be achieved using **index number**

indexing follows whole number system i.e. 0, 1, 2, 3...

it can be achieved by using **index operator** [ ]

In [38]:
```
fruits
```

Out[38]:
```
['apple', 'banana', 'strawberry']
```

In [39]:
```
# index
# using fruits[index_value]

# len(fruits) = 3 (say 'n')
# which means the max index value will be n - 1 = 2

print(fruits[0], fruits[1], fruits[2])
```

```
apple banana strawberry
```

In [40]:
```
# What if we try to find 4th item (3rd index) in the list (fruits)
print(fruits[4])

# we'll have a error "IndexError" as the index value (=4) is not in range.
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[40], line 2
      1 # What if we try to find 4th item (3rd index) in the list (fruits)
----> 2 print(fruits[4])

IndexError: list index out of range
```

Lists are mutable.

It means after the creation of a list, we can **change, add, and remove** items.

In [41]:
```
fruits
```

Out[41]:
```
['apple', 'banana', 'strawberry']
```

In [42]:
```
# if we want to change the value of index:0
fruits[0] = "Orange"

# using indexing, we assigned a value to the 0th index as "Orange" which overrid

print(fruits)
```

```
['Orange', 'banana', 'strawberry']
```

Data Types

List items can be of any data type like **Integer, String, Float as well as objects** because of which we can say **Lists are heterogenous**

In [43]:
```
new_list = [1, 2, "veggies", 3.14, True, 7 + 6j]
```

In [44]:
```
# we can determine each type of data stored in the list by
for i in new_list:
    print(type(i), end = " ")
```

```
<class 'int'> <class 'int'> <class 'str'> <class 'float'> <class 'bool'> <class
'complex'>
```

In [45]:
```
# better we can use a built-in function: enumerate()
for x, y in enumerate(new_list):
    print(f"{new_list[x]} = {type(y)}")
```

```
1 = <class 'int'>
2 = <class 'int'>
veggies = <class 'str'>
3.14 = <class 'float'>
True = <class 'bool'>
(7+6j) = <class 'complex'>
```

Nested List

list defined inside another list

In [46]:
```
# nested list
# let's define two list as List_01 and List_02
```

```
List_01 = [1, 2, 3, 4]
print(List_01)

List_02 = [2, 4, 6, 8]

# and now as we know lists are mutable which means we can add or remove or chang
# we can use this functionality to achieve nesting lists as shown below

List_01 = [1, 2, 3, 4, List_02]
print(List_01)
```

```
[1, 2, 3, 4]
[1, 2, 3, 4, [2, 4, 6, 8]]
```

In [47]:
```
# now there's another way of nesting lists using a built-in method of lists i.e.
List_01.append(List_02)
```

In [48]:
```
print(List_01)
```

```
[1, 2, 3, 4, [2, 4, 6, 8], [2, 4, 6, 8]]
```

Let's explore built-in methods of List

- **append():** Adds an element to the end of list
- **extend():** Extend the list by adding elements from another iterable
- **remove():** removes the first occurence of a specified position in the list
- **pop():** removes and returns the specified index or else by default **the last element if no index is specified**
- **clear()**: removes all elements from the list
- **index()**: returns the index of the first occurence of a specified value
- **count()**: returns the number of occurences of a specified value
- **sort()**: sort the list in ascending order (or based on a custom key function such as reverse=False)
- **reverse()**: reverses the order of elements in a list
- **insert()**: inserts an element at a specified position in the list.
- **copy()**: returns a copy of list

In [49]:
```
# append()
fruits.append("kiwi")

print(fruits)
```

```
['Orange', 'banana', 'strawberry', 'kiwi']
```

In [50]:
```
# extend()
fruits.extend(["Lemon", "Raspberry", "Guava"]) # the argument passed inside exte
print(fruits)
```

```
['Orange', 'banana', 'strawberry', 'kiwi', 'Lemon', 'Raspberry', 'Guava']
```

In [51]:
```
# remove()
fruits.remove("banana") # banana will be removed from the list
print(fruits)
```

```
['Orange', 'strawberry', 'kiwi', 'Lemon', 'Raspberry', 'Guava']
```

In [53]:
```
# pop()
fruits.pop()
print(f"The element removed using .pop() is {fruits.pop()}")

print(fruits)
```

```
['Orange', 'strawberry', 'kiwi']
The element removed using .pop() is kiwi
['Orange', 'strawberry']
```

In [54]:
```
# clear()
fruits.clear() # removes every element in the list: fruits
print(fruits)
```

```
[]
```

In [55]:
```
# index() and count()
# let's add some fruit items to our list in order to see capabilities of other m
fruits.extend(["Pineapple", "Grape", "Pear", "Apple", "Watermelon"])
```

In [56]:
```
print(fruits)
```

```
['Pineapple', 'Grape', 'Pear', 'Apple', 'Watermelon']
```

In [57]:
```
# before using index() let's add some duplicate value to the list
fruits.extend(["Apple", "Grape", "Pear"])
```

In [59]:
```
print(fruits)

print(fruits.index("Apple"))
```

```
['Pineapple', 'Grape', 'Pear', 'Apple', 'Watermelon', 'Apple', 'Grape', 'Pear']
3
```

In [62]:
```
# in-order to visualize the funcionality in detail

for index, fruit_index in enumerate(fruits):
    """
    index: int (index value)
    fruit: str
    """
    print(f"{fruit_index} = {index}")
```

```
Pineapple = 0
Grape = 1
Pear = 2
Apple = 3
Watermelon = 4
Apple = 5
Grape = 6
Pear = 7
```

- Here we can see, after using index() functions the index values of some of the items are same as the first occurence.
- e.g. ***Apple*** first occurred in index: **3** and then at index: **5**
- index() will return the first occurence of the item in the list which is **3** (for *Apple*)

In [73]:
```
# count()
print(f"Apple has occured {fruits.count('Apple')} times in the list.")
```

```
Apple has occured 2 times in the list.
```

In [86]:
```python
# sort() and reverse()

fruits.sort() # ascending order
print(fruits)

print("================================================================================")

print("reversed list: ->>")
fruits.sort(reverse=True) # descending order
print(fruits)
```

```
['Apple', 'Apple', 'Grape', 'Grape', 'Pear', 'Pear', 'Pineapple', 'Watermelon']
================================================================================
reversed list: ->>
['Watermelon', 'Pineapple', 'Pear', 'Pear', 'Grape', 'Grape', 'Apple', 'Apple']
```

In [88]:
```python
# reverse()
fruits.sort()

fruits.reverse()
print(fruits)
```

```
['Watermelon', 'Pineapple', 'Pear', 'Pear', 'Grape', 'Grape', 'Apple', 'Apple']
```

In [91]:
```python
# insert()
fruits.insert(3, "Pomegranate")

# subsequnetly the duplicates will also be changed.
print(fruits)
```

```
['Watermelon', 'Pineapple', 'Pear', 'Pomegranate', 'Pomegranate', 'Pomegranate',
'Pear', 'Grape', 'Grape', 'Apple', 'Apple']
```

In [93]:
```python
# copy()
fruits_copy = fruits.copy()
print(fruits_copy)

# lets check
print(fruits_copy == fruits)
```

```
['Watermelon', 'Pineapple', 'Pear', 'Pomegranate', 'Pomegranate', 'Pomegranate',
'Pear', 'Grape', 'Grape', 'Apple', 'Apple']
True
```

List Slicing

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

In [96]:
```python
# let's try with examples
# generate a list of even integers upto 50.

even_numbers = [x for x in range(1, 51) if x % 2 == 0]
print(even_numbers)
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42,
44, 46, 48, 50]
```

In [99]:
```python
# let's try to make slices of the (even_numbers) list
```

```
even_numbers[::]
```

Out[99]:
```
[2,
 4,
 6,
 8,
 10,
 12,
 14,
 16,
 18,
 20,
 22,
 24,
 26,
 28,
 30,
 32,
 34,
 36,
 38,
 40,
 42,
 44,
 46,
 48,
 50]
```

In [100…
```
even_numbers[0:10]
```

Out[100…
```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

In [101…
```
len(even_numbers)
```

Out[101…
```
25
```

In [105…
```
print(even_numbers)
even_numbers[4:20:3]
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42,
44, 46, 48, 50]
```

Out[105…
```
[10, 16, 22, 28, 34, 40]
```

Soumesh Khuntia