LIST

DEFINING A LIST

- simply write Python code within a Python script or a Jupyter Notebook. You can execute this code in any Python environment, including Anaconda, by either writing it in a script file **(e.g., `my_script.py`)** and running it with Python, or by typing it into a Jupyter Notebook cell and executing the cell.

  Here's an example of defining a list:
  my_list = [1, 2, 3, 4, 5]

LIST SYNTAX

- enclose the elements of the list within square brackets `[]`, separated by commas.

  Here's an example of list syntax:

  my_list = [element1, element2, element3, ...]

- Where `element1`, `element2`, `element3`, etc., are the elements you want to include in the list. These elements can be of any data type (e.g., integers, strings, floats, other lists, etc.).

  Here's an example:

- my_list = [1, 2, 3, 'a', 'b', 'c', [4, 5, 6]]

  This creates a list with integers, strings, and another nested list as elements.

ACCESSING LIST ELEMENTS

- accessing elements in a list involves specifying the index or range of indices within square brackets to retrieve the desired elements. Python's indexing and slicing mechanisms provide flexibility in accessing elements based on position within the list.

Here's an example of accessing elements in a list:

```
my_list = [10, 20, 30, 40, 50]

# Accessing individual elements
print(my_list[0])  # Output: 10 (first element)
print(my_list[2])  # Output: 30 (third element)

# Accessing elements using negative indices (counting from
the end)
print(my_list[-1])  # Output: 50 (last element)
print(my_list[-3])  # Output: 30 (third element from the
end)

# Slicing to access multiple elements
print(my_list[1:4])  # Output: [20, 30, 40] (elements from
index 1 to 3)
print(my_list[:3])   # Output: [10, 20, 30] (first three
elements)
print(my_list[2:])   # Output: [30, 40, 50] (elements from
index 2 to the end)
```

LOOP THROUGH A LIST

- commonly done using a `for` loop. This code will print each element of the list `my_list` on a new line. You can perform any operation inside the loop for each element of the list.

Here's an example of how to loop through a list:

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

# Using a for loop to iterate over each element in the list
for item in my_list:
    print(item)
```

LIST LENGTH

- To get the length of a list in Python, you can use the `len()` function.

Here's how you can use it:

```
my_list = [1, 2, 3, 4, 5]
length = len(my_list)
print("Length of the list:", length)
```

This will output:

```
Length of the list: 5
```

- The `len()` function returns the number of elements in the list. In this case, the list `my_list` contains 5 elements, so the length is 5.

ADD ITEMS IN THE LIST

- To add items to a list in Python, you have a few options depending on where you want to add the items:

● To add an item to the end of the list, you can use the **`append()` method.**

```
my_list = [1, 2, 3]
my_list.append(4)
```

After this, `my_list` will become `[1, 2, 3, 4]`.

● To add multiple items (another list, tuple, etc.) to the end of the list, you can use the **`extend()` method.**

```
my_list = [1, 2, 3]
my_list.extend([4, 5])
```

After this, `my_list` will become `[1, 2, 3, 4, 5]`.

● To add an item at a specific index in the list, you can use the **`insert()` method.**

```
my_list = [1, 2, 3]
my_list.insert(1, 4)  # Inserting 4 at index
```

After this, `my_list` will become `[1, 4, 2, 3]`.

REMOVE ITEM FROM A LIST

- To remove an item from a list in Python, you have a few options depending on what you want to achieve:

- If the value of the item you want to remove, you can use the **`remove()` method.** This method removes the first occurrence of the specified value.

  ```
  my_list = [1, 2, 3, 4, 5]
  my_list.remove(3)
  # Removes the first occurrence of 3
  ```

  After this, `my_list` will become `[1, 2, 4, 5]`.

- If the index of the item you want to remove, you can use the **`pop()` method.** This method removes the item at the specified index and returns its value.

  ```
  my_list = [1, 2, 3, 4, 5]
  removed_item = my_list.pop(2)
  # Removes the item at index 2 (value: 3)
  ```

  After this, `my_list` will become `[1, 2, 4, 5]`, and `removed_item` will be `3`.

- If the index of the item you want to remove and don't need the removed item, you can use the `del` statement.

  ```
  my_list = [1, 2, 3, 4, 5]
  del my_list[2]
  # Removes the item at index 2 (value: 3)
  ```

  After this, `my_list` will become `[1, 2, 4, 5]`.

THE LIST () CONSTRUCTOR

- It's used to create a new list. It can accept an iterable object (like another list, tuple, string, etc.) as an argument and creates a new list containing the elements of that iterable object and provides a convenient way to create a list from different types of iterable objects or to create an empty list.

Here are a few examples of using the `list()` constructor:

1. Creating a list from a tuple:

```
my_tuple = (1, 2, 3)
my_list = list(my_tuple)
print(my_list)  # Output: [1, 2, 3]
```

2. Creating a list from a string:

```
my_string = "hello"
my_list = list(my_string)
print(my_list)  # Output: ['h', 'e', 'l', 'l', 'o']
```

3. Creating an empty list:

```
empty_list = list()
print(empty_list)  # Output: []
```

4. Creating a list with specified elements:

```
my_list = list([1, 2, 3])
print(my_list)  # Output: [1, 2, 3]
```

LIST METHODS

- some commonly used methods for lists in Python. These are just a few of the many methods available for lists. They provide various functionalities to manipulate and work with lists efficiently.

1.) **append()**: Adds an element to the end of the list.

```
my_list = [1, 2, 3]
my_list.append(4)
```

After this, `my_list` will become `[1, 2, 3, 4]`.

2.) **extend():** Extends the list by appending elements from another iterable.

```
my_list = [1, 2, 3]
my_list.extend([4, 5])
```

After this, `my_list` will become `[1, 2, 3, 4, 5]`.

3.) **insert():** Inserts an element at a specified position.

```
my_list = [1, 2, 3]
my_list.insert(1, 4)  # Inserting 4 at index 1
```

After this, `my_list` will become `[1, 4, 2, 3]`.

4.) **remove():** Removes the first occurrence of a specified value.

```
my_list = [1, 2, 3, 4, 3]
my_list.remove(3)  # Removes the first occurrence of 3
```

After this, `my_list` will become `[1, 2, 4, 3]`.

5.) **pop():** Removes and returns the element at a specified position (default is the last element).

```
my_list = [1, 2, 3, 4]
popped_item = my_list.pop(1)  # Removes the element at
index 1 (value: 2)
```

After this, `my_list` will become `[1, 3, 4]`, and `popped_item` will be `2`.

6.) **index():** Returns the index of the first occurrence of a specified value.

```
my_list = [1, 2, 3, 4]
index = my_list.index(3)  # Returns the index of 3
```

Here, `index` will be `2`.

7.) **count():** Returns the number of occurrences of a specified value.

```
my_list = [1, 2, 2, 3, 2]
count = my_list.count(2)  # Returns the number of
occurrences of 2
```

Here, `count` will be `3`.

8.) **sort():** Sorts the list in ascending order.

```
my_list = [3, 1, 4, 2]
my_list.sort()
```

After this, `my_list` will become `[1, 2, 3, 4]`.

9.) **reverse():** Reverses the order of the elements in the list.

```
my_list = [1, 2, 3, 4]
my_list.reverse()
```

After this, `my_list` will become `[4, 3, 2, 1]`.

NESTED LIST

- contains other lists as its elements. This allows you to create a list of lists, or a list of any other data type.

  Here's an example of a nested list:

  ```
  nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
  ```

  In this example, `nested_list` contains **three lists as its elements:**

  1. `[1, 2, 3]`
  2. `[4, 5, 6]`
  3. `[7, 8, 9]`

You can access elements of a nested list using multiple indices. For example, to access the element `5`, you would use the indices `[1][1]`, because it is the second element of the second list (`[4, 5, 6]`) within `nested_list`.

```
element = nested_list[1][1]  # Accessing the element 5
print(element)  # Output: 5
```

You can also iterate over nested lists using nested loops:

```
nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

for sublist in nested_list:
    for element in sublist:
        print(element)
```

This will print each element of the nested list on a new line.