**Name:** Flores, Christian T.

**Course & Section:** CS3C

**LIST**

* **DEFINING A LIST**
* a list is a versatile and fundamental data structure used to store collections of items. It is an ordered sequence of elements, which can be of any data type, such as integers, floats, strings, or even other lists. Lists are mutable, meaning their elements can be changed after the list is created. Lists are defined by enclosing the elements within square brackets [], separated by commas.
* Lists are often used in scenarios where you need to work with a collection of items that can be modified, reordered, or iterated over easily. They are widely used in programming for tasks ranging from simple data storage to complex algorithm implementations.
* **LIST SYNTAX**
* The syntax for creating a list in Python is quite straightforward. You enclose the elements of the list within square brackets [] and separate them with commas. Here's the basic syntax:

**my\_list = [element1, element2, element3, ...]**

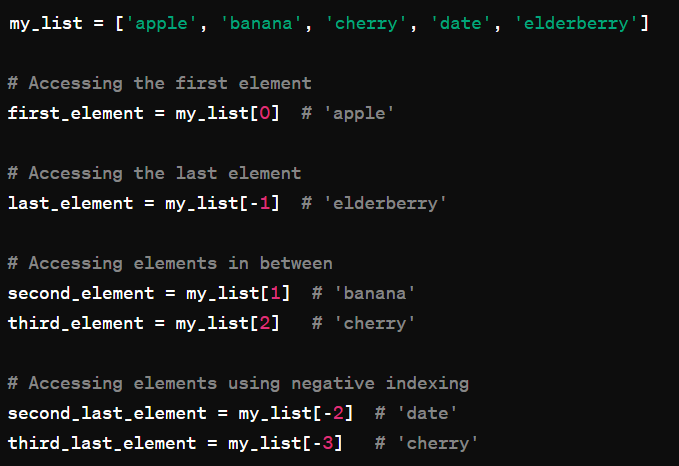
* Each element can be of any data type, and they can be heterogeneous, meaning you can mix different types within the same list. For example:

**my\_list = [1, 2, 3, "hello", True, 5.5]**

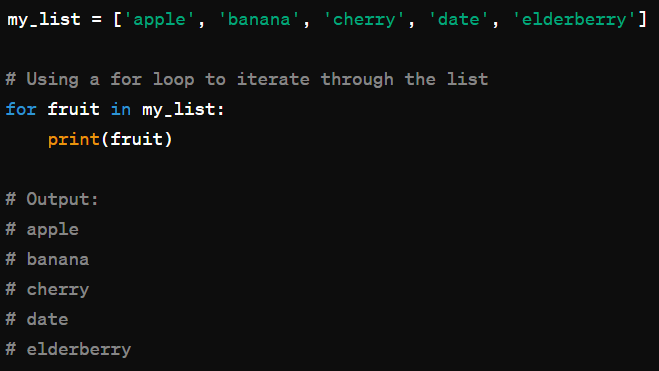
* You can also create an empty list by simply using empty square brackets:

**empty\_list = []**

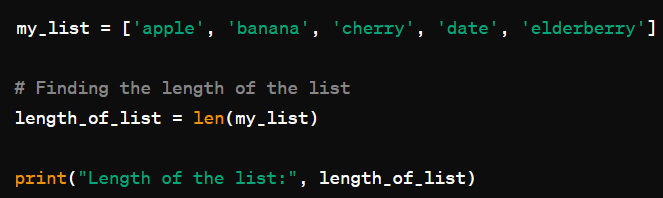
* **ACCESSING LIST ELEMENT**
* In Python, you can access individual elements of a list using indexing. List indexing starts from 0 for the first element and increases sequentially. Additionally, negative indexing allows you to access elements from the end of the list, starting from -1. Here's how you access list elements:



* **LOOPING THROUGH A LIST**
* Looping through a list in Python is commonly done using a for loop. You can iterate over each element in the list and perform operations or actions as needed. Here's how you can loop through a list:



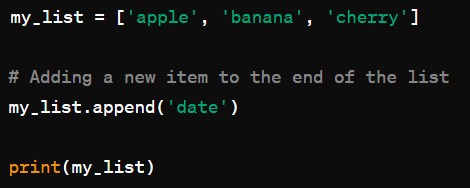
* **LIST LENGTH**
* In Python, you can find the length of a list—the number of elements it contains—using the built-in len() function. Here's how you can use it:





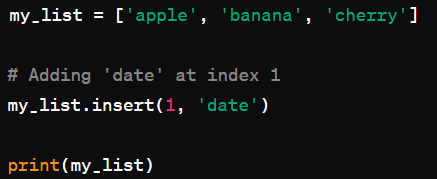
* **ADD ITEMS IN THE LIST**
* You can add items to a list in Python using several methods, such as append(), insert(), or concatenation. Here are examples of each method:

1. Using the append() method:



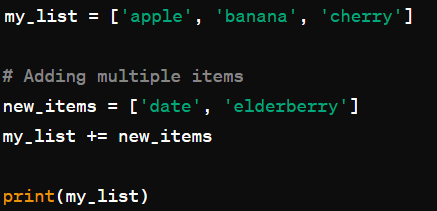


1. Using the insert() method to add an item at a specific position:





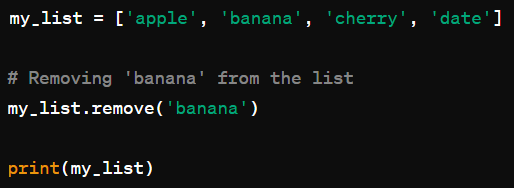
1. Using concatenation to add multiple items (or another list) to the end of the list:





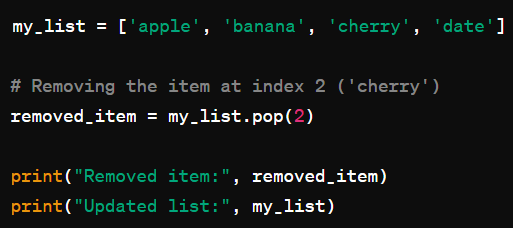
* **REMOVE ITEM FROM A LIST**
* You can remove items from a list in Python using various methods like remove(), pop(), or del. Here are examples of each method:

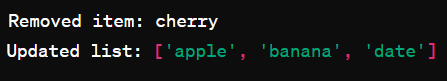
1. Using the remove() method to remove a specific item:



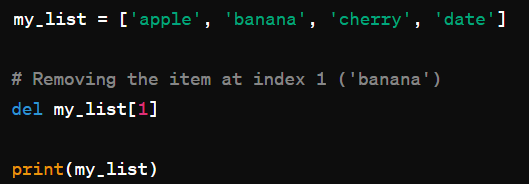


1. Using the pop() method to remove an item at a specific index and return it:





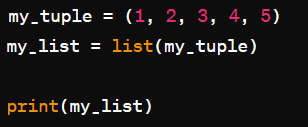
1. Using the del statement to remove an item or a slice of items based on index:





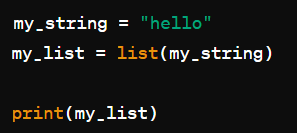
* **THE LIST () CONSTRUCTOR**
* The list() constructor can convert other iterable objects like tuples or strings into a list or create an empty list if no arguments are provided. Here are some examples:

1. Creating a list from a tuple:



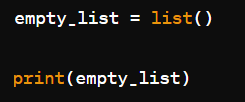


1. Creating a list from a string:





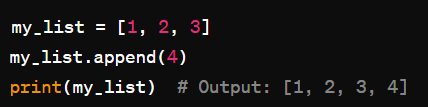
1. Creating an empty list:



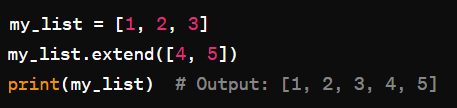


* **LIST METHODS**

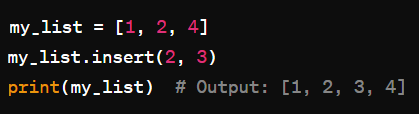
1. append(): Adds an element to the end of the list.



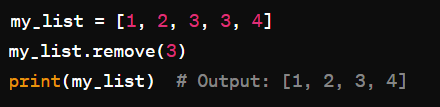
1. extend(): Extends the list by appending elements from the iterable.



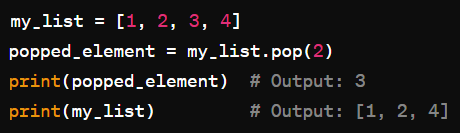
1. insert(): Inserts an element at a specified position.



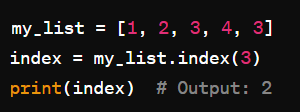
1. remove(): Removes the first occurrence of a specified value.



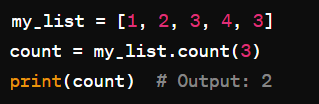
1. pop(): Removes the element at the specified position (or the last element if not specified) and returns it.

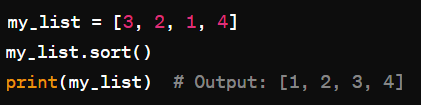


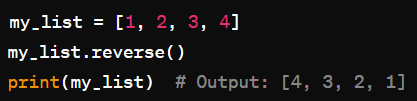
1. index(): Returns the index of the first occurrence of a specified value.



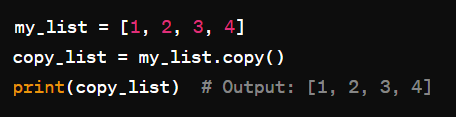
1. count(): Returns the number of occurrences of a specified value.



1. sort(): Sorts the list in ascending order.
2. reverse(): Reverses the elements of the list.



1. copy(): Returns a shallow copy of the list.

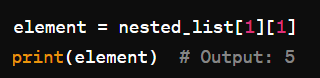


* **NESTES LISTS**
* Nested lists in Python are lists that contain other lists as elements. These inner lists can themselves contain any data type, including other lists, creating a hierarchical structure. Here's an example of a nested list:

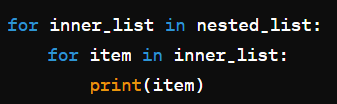


In this example, nested\_list contains three inner lists, each representing a row of a matrix.

* You can access elements of a nested list using multiple index values. For example, to access the element 5:



* You can also loop through nested lists using nested loops:



This would print each element of the nested list on a separate line.

* Nested lists are useful for representing multi-dimensional data structures such as matrices, tables, or hierarchical data where elements have nested relationships. They provide a flexible and powerful way to organize and manipulate data in Python.