Module 14: Python – Collections, Functions, and Modules

Accessing List

Creating and Accessing Elements: You create a list by enclosing comma-separated items within square brackets [] . You can access individual elements using their position (index).

Positive Indexing: Starts from 0 for the first element, 1 for the second, and so on. Negative Indexing: Starts from -1 for the last element, -2 for the second to last, and so on. This is handy for accessing elements from the end of the list without knowing its exact length.

Slicing a List: This allows you to extract a portion of a list, creating a new list. You specify a start index (inclusive), an end index (exclusive), and an optional step (how many items to skip). The syntax is my\_list[start:end:step]

1. List Operations

Concatenation: Joining two or more lists using the + operator. Repetition: Repeating a list multiple times using the operator. Membership: Checking if an item exists in a list using the in or not in operators.

append() : Adds a single element to the end of the list. insert() : Adds an element at a specified index. remove() : Removes the first occurrence of a specified value. pop() : Removes and returns the element at a specified index

1. Working with Lists

Iterating over a List: You can easily loop through each element in a list using a for loop, which is essential for processing data within the list.

Sorting and Reversing: sort() : A list method that sorts the list in place (modifies the original list). sorted() : A builtin function that returns a new sorted list without modifying the original. reverse() : A list method that reverses the order of elements in place.

Basic List Manipulations: This encompasses the common tasks of adding new elements, deleting existing ones, updating values at specific positions, and using slicing to extract sub-lists

1. Tuple

Introduction to Tuples, Immutability: Tuples are ordered collections of items, just like lists, but once a tuple is created, you cannot change its elements

Creating and Accessing Elements: You create a tuple by enclosing comma-separated items within parentheses. Accessing elements is identical to lists, using positive and negative indexing

Basic Operations with Tuples: Like lists, tuples support concatenation ( + ), repetition ( ), and membership checking

1. Accessing Tuples

Positive and Negative Indexing: You can retrieve individual elements using their position, starting from 0 for the first element (positive) or -1 for the last (negative).

Slicing a Tuple: You can extract a portion of a tuple using slicing [start:end:step] , which returns a new tuple.

1. Dictionaries

Introduction to Dictionaries: Key-Value Pairs: Imagine a real-world dictionary where each word (the "key") has a definition (the "value"). Python dictionaries work similarly. Each item in a dictionary consists of a key and its associated value. Keys must be unique and immutable (like strings or numbers), while values can be of any data type. Dictionaries are defined using curly braces {} .

Accessing: You access values by referring to their corresponding key (e.g., my\_dict['name'] ). Adding: You can add new key-value pairs by assigning a value to a new key. Updating: You can change the value associated with an existing key by assigning a new value to it. Deleting: You can remove key-value pairs using the del keyword or the pop() method.

Dictionary Methods: keys() : Returns a view object that displays a list of all the keys in the dictionary. values() : Returns a view object that displays a list of all the values in the dictionary. items() : Returns a view object that displays a list of a dictionary's key-value tuple pairs.

1. Working with Dictionaries

Iterating over a Dictionary: You can loop through a dictionary's keys, values, or key-value pairs using a for loop, often in combination with the keys() , values() , or items() methods.

Merging Two Lists into a Dictionary: This is a common task where you might have one list for keys and another for values. You can achieve this using a loop or, more efficiently, using the built-in zip() function, which pairs up elements from multiple iterables.

Counting Occurrences: Dictionaries are perfect for counting the frequency of items (like characters in a string or words in a sentence) because each unique item can be a key, and its count can be the value.

1. Functions

Defining Functions: You define a function using the def keyword, followed by the function name, parentheses (which can contain parameters), and a colon. The function's code block is indented

Types of Functions:

With/Without Parameters: Functions can accept input values (parameters) or operate without any input. With/Without Return Values: Functions can send back a result using the return statement, or they can simply perform an action without explicitly returning anything.

Anonymous Functions (Lambda Functions): These are small, single-expression functions that don't have a name. They are defined using the lambda keyword and are often used for short, throwaway functions, especially with higher-order functions like map() , filter() , and sorted() .

1. Modules

Introduction to Python Modules and Importing Modules: A module is simply a Python file ( .py ) containing Python definitions and statements. When you want to use code from another file, you "import" the module. This allows you to access its functions, classes, and variables

Standard Library Modules: Python comes with a vast "standard library" – a collection of pre-installed modules that provide a wide range of functionalities

Creating Custom Modules: You can easily create your own modules by saving your Python code in a .py file. This allows you to organize your projects, separate concerns, and reuse your own code across different programs