Python – Collections, functions and Modules

Accessing List

1. Understanding how to create and access elements in a list.

- In Python, a list is a container that holds multiple items in one variable.
- These items can be numbers, strings, or even other lists.
- Lists are changeable, meaning we can add, remove, or update items whenever we want.
- In Python programming, lists are used to store a collection of data in an ordered way. They're useful when we want to group related values together.

❖ How to create a list?

- We create a list by placing items inside square brackets [], separated by commas. Lists can hold numbers, strings, or a mix of both.
- Lists are useful in Python programming when we want to store and manage a collection of data in one variable.

❖ How to access a list?

- We access elements in a list using index numbers.
- Indexing starts from 0, so the first item is at position 0, the second at 1, and so on. We use square brackets [] to get the value at a specific position.
- We can also use negative indexing to access items from the end. For example, -1 gives us the last item in the list.

2. Indexing in lists (positive and negative indexing).

- Indexing is the method of accessing individual elements of the list using their position.
- Python supports two types of indexing:

➢ Positive Indexing:

- Positive indexing means accessing elements from the beginning of the list.

- The index starts at 0 for the first element, and increases by 1 for each subsequent element.
- It is the most commonly used form of indexing.

➢ **Negative Indexing:**

- Negative indexing means accessing elements from the end of the list.
- The index starts at -1 for the last element, and decreases by 1 as we move left.
- It is useful when we want to access elements from the end of the list without knowing its length.

3. Slicing a list: accessing a range of elements.

- Slicing is a technique used to access a group (or sub-part) of elements from a list, rather than a single item.
- It provides a powerful way to extract a range of elements without using loops.
- Syntax: list[start:stop:step]
  - start → The index to begin the slice (inclusive)
  - stop → The index to end the slice (exclusive)
  - step → The interval between elements (optional)
  - If step is not provided, Python uses a default value of 1.

List Operations

1. Common list operations: concatenation, repetition, membership.

❖ **Concatenation Operator**:

- The + operator is used to combine two or more lists into a single list. This operation is known as concatenation.
- When two lists are joined using +, a new list is created that contains all elements of the first list followed by all elements of the second list.
- Syntax: new_list = list1 + list2

- The original lists remain unchanged.
- Both operands must be lists; you cannot concatenate a list and a non-list item directly.

❖ Repetition Operator:

- The * operator is used to repeat the elements of a list multiple times.
- When a list is multiplied by an integer n, it repeats the list n times in the same order.
- Syntax: new_list = list * n
- The list is not modified; a new repeated list is created.
- The repetition factor must be a non-negative integer.

❖ Membership Operator:

- Membership operators are used to check whether an element exists in a list or not.

  ➢ **Types:**
    - in: Returns True if the element exists in the list.
    - not in: Returns True if the element does not exist in the list.

- Syntax: element in list
  element not in list
- Useful in conditional statements.
- Common in loops, filters, and search operations.

2. Understanding list methods like append (), insert (), remove (), pop ().

❖ append () Method:

- The append () method is used to add a single element to the end of a list.
- Syntax: list.append(element)
- The list grows in size by one.
- The element can be of any type (integer, string, list, etc.).
- Appending a list will add it as a single nested list, not merge the lists.

❖ insert () Method:

- The insert () method allows you to add an element at a specific index in the list.
- Syntax: list.insert(index, element)
- The element is inserted before the given index.

- If the index is out of range, it adds the element at the end.
- The original list is modified.

❖ remove () Method:

- The remove () method is used to delete the first occurrence of a specific value in the list.
- Syntax: list.remove(element)
- Only the first matching element is removed.
- If the element does not exist, it raises a ValueError.

❖ pop () Method:

- The pop () method is used to remove and return an element from a list. By default, it removes the last element.
- Syntax: list.pop() # Removes last item
  list.pop(index) # Removes item at given index
- If index is not given, it removes the last item.
- Returns the removed element.
- Raises IndexError if the index is out of range.

Working with Lists

1. Iterating over a list using loops.

- In Python, a list is a built-in data structure used to store a collection of items.
- A list can contain elements of different data types like integers, strings, floats, or even other lists.
- One of the most common operations performed on a list is iteration, which means accessing each element in the list, one by one, usually to perform some action on it.

❖ Why Iterate Over a List?

- Iterating over a list allows us to:
- Access and print each item.
- Perform calculations using list elements.

- Search for a particular value.
- Modify or filter the list. Perform bulk operations like sorting, counting, or transforming values.

2. Sorting and reversing a list using sort (), sorted (), and reverse ().

❖ Sorting a list:

- Sorting means arranging the elements of a list in a specific order — either ascending or descending.

❖ sort () Method:

- It is a list method.
- It modifies the original list (in-place sorting).
- It does not return a new list.

❖ sorted () Function:

- It is a built-in function, not a method of the list.
- It returns a new sorted list, leaving the original list unchanged.

❖ Reversing a List:

- Reversing means flipping the order of elements (last becomes first, and so on). It does not sort, only reverses the current order.

❖ reverse () Method:

- It is used to reverse the original list in-place.
- It does not return a new list.

3. Basic list manipulations: addition, deletion, updating, and slicing.

❖ Addition in a List:

- Adding elements to a list means inserting new items. Python provides various ways to add elements.

➢ append () Method:

- Adds a single element to the end of the list.

➢ insert () Method:

- Inserts an element at a specific position (index).

➢ extend () Method:

- Adds multiple elements from another list or iterable.

❖ Deletion from a List:

- You can remove elements from a list using various methods:

➢ remove () Method:

- Removes the first occurrence of the specified value.

➢ pop () Method:

- Removes the item at a given index. If no index is specified, it removes the last item.

➢ del Statement:

- Deletes a specific element or the entire list.

➢ clear () Method:

- Removes all items from the list.

❖ Updating a List:

- You can update list elements by assigning new values using indexing.

❖ Slicing a List:

- Slicing is used to extract a portion of a list by specifying a start, stop, and step.
  - start: Starting index (inclusive)
  - stop: Ending index (exclusive)
  - step: Interval (default is 1)

Tuple

1. Introduction to tuples, immutability.

❖ Introduction

- In Python, a tuple is a collection data type that can hold multiple items in a single variable, similar to lists.
- However, unlike lists, tuples are immutable, meaning that once a tuple is created, its contents cannot be changed.
- Tuples are used to group related data. They are defined by enclosing elements in parentheses (), separated by commas.

➢ Key Characteristics of Tuples:

- Ordered: Tuples maintain the order of elements.
- Immutable: Elements cannot be added, removed, or changed.
- Can contain mixed data types: integers, strings, booleans, etc.
- Duplicates allowed: Tuples can contain repeated values.

➢ Immutability:

- Immutability means unchangeable. If an object is immutable, its state cannot be modified after it is created.

➢ Why Are Tuples Immutable?

- Tuples do not support operations like. append (), remove (), or item assignment. Once defined, you cannot change, delete, or update any value inside the tuple.

2. Creating and accessing elements in a tuple.

❖ Creating a Tuple:

- In Python, a tuple is created by placing a sequence of values separated by commas inside parentheses ().
- Tuples can store elements of different data types, such as integers, strings, floats, or even other tuples.
- Syntax: my_tuple = (element1, element2, element3, ...)

❖ Accessing Elements in a Tuple:

- You can access individual elements in a tuple using indexing. Indexing in Python starts from 0.
- Syntax: tuple_name[index]

❖ Slicing a Tuple:

- Slicing allows you to access a range of elements from the tuple.
- Syntax: tuple_name[start:stop]

3. Basic operations with tuples: concatenation, repetition, membership.

❖ Concatenation of Tuples:

- Concatenation means combining two or more tuples into one.
- Syntax: tuple1 + tuple2
- Concatenation creates a new tuple.
- Original tuples remain unchanged because tuples are immutable.

❖ Repetition of Tuples:

- Repetition means repeating the elements of a tuple multiple times using the * operator.
- Syntax: tuple * n
- n is the number of times to repeat the tuple.

❖ Membership Operation in Tuples:

- Membership operators in and not in are used to check whether an element is present in the tuple.
- Syntax: element in tuple
  element not in tuple
- Returns True if the element is found.
- Returns False if the element is not found.

Accessing Tuples

1. Accessing tuple elements using positive and negative indexing.

- In Python, tuples are ordered collections of items.
- This means that each element in a tuple has a fixed position, also known as an index, which can be used to access the element.

➢ Python supports two types of indexing:
  - Positive Indexing
  - Negative Indexing

❖ Positive Indexing:

- Positive indexing starts from 0 and moves from left to right. The first element has the index 0, the second has 1, and so on.
- Use positive indexing when you want to count from the start.

❖ Negative Indexing:

- Negative indexing starts from -1 and moves from right to left. The last element is -1, the second last is -2, and so on.
- Negative indexing is especially useful when you want to access elements starting from the end of the tuple, without needing to know its length.
- Use negative indexing when you want to count from the end.

2. Slicing a tuple to access ranges of elements.

- In Python, slicing is a technique used to access a range of elements from a tuple.
- Instead of accessing a single element using indexing, slicing allows you to extract a subsection (subtuple) from the original tuple.
- Syntax: tuple_name[start:stop:step]
  - start: The index to begin the slice (inclusive).
  - stop: The index to end the slice (exclusive).
  - step *(optional)*: The interval between elements (default is 1).

- Slicing in tuples provides an easy way to access multiple elements and subsets from a tuple without using loops.
- It uses the format tuple[start:stop:step], and supports both positive and negative indices.

- Since tuples are immutable, slicing returns a new tuple containing the selected range of elements.

Dictionaries

1. Introduction to dictionaries: key-value pairs.

- In Python, a dictionary is a built-in data structure used to store data in the form of key-value pairs.
- It is one of the most powerful and flexible data types in Python, ideal for representing data that is associated in pairs—like a name with a phone number, or a student's ID with their grades.
- A dictionary in Python is unordered, mutable, and indexed. It is defined by curly braces {}, and each entry in a dictionary consists of a key and its associated value.
- Syntax: dictionary_name = {
  "key1": "value1",
  "key2": "value2",
  "key3": "value3"
  }

➢ **What are Keys and Values?**

- A key is a unique identifier for an item.
- A value is the data associated with the key.

- Each key must be unique and immutable (strings, numbers, or tuples can be keys), while values can be of any data type, including lists or other dictionaries.
- Dictionaries in Python are essential for handling structured data where each value is labelled with a unique key.
- They are widely used in real-world applications like storing user information, counting frequency of items, and building databases in memory.
- Their flexibility and performance make them one of the most important data types in Python.

2. Accessing, adding, updating, and deleting dictionary elements.

- Python dictionaries allow us to store and manipulate data using key-value pairs.
- Once a dictionary is created, we can easily access, add, update, and delete its elements.

➢ Accessing Dictionary Elements:

- To access a value in a dictionary, use its key inside square brackets or the. get () method.

➢ Adding Elements to a Dictionary:

- You can add a new key-value pair simply by assigning a value to a new key.

➢ Updating Dictionary Elements:

- If a key already exists in the dictionary, assigning a new value will update the existing value.

➢ Deleting Elements from a Dictionary:

- Python provides several ways to remove elements from a dictionary.

3. Dictionary methods like keys (), values (), and items ().

❖ keys () Method:

- The keys () method returns a view object that displays a list of all the keys in the dictionary.
- Syntax: dictionary.keys()

❖ values () Method:

- The values () method returns a view object containing all the values in the dictionary.
- Syntax: dictionary.values()

❖ items () Method:

- The items () method returns a view object of the dictionary's key-value pairs as tuples.
- Syntax: dictionary.items()

Working with Dictionaries

1. Iterating over a dictionary using loops.

- Dictionaries in Python store data in the form of key-value pairs.
- To work with all the elements in a dictionary, we often need to iterate over it using loops.
- Python provides simple and efficient ways to loop through dictionaries using the for loop.
- By default, looping over a dictionary will iterate through its keys.
- To access only the values in a dictionary, use the. values () method.
- The most common way to iterate through a dictionary is using the. items() method, which returns both the key and value.

2. Merging two lists into a dictionary using loops or zip ().

- In Python, you can create a dictionary by combining two separate lists — one containing the keys and the other containing the values.
- This process is called merging two lists into a dictionary, and it can be done in multiple ways, such as using a loop or the built-in zip () function.
- You can use a loop to iterate through the indices of the lists and manually assign key-value pairs.
- The zip () function combines two lists into pairs. These pairs can directly be converted into a dictionary using the dict () constructor.
- The zip() function provides a quick and efficient method, while the loop method offers more control and flexibility. Both are essential tools for working with structured data.

3. Counting occurrences of characters in a string using dictionaries.

- In Python, dictionaries are ideal for storing data as key-value pairs, which makes them perfect for counting how many times each character appears in a string.
- Each character becomes a key, and the number of times it appears becomes the value.
- Python's collections module has a class called Counter that does this directly.

Functions

1. Defining functions in Python.

- Functions in Python are reusable blocks of code designed to perform a specific task.
- They allow you to organize your program into smaller, manageable, and modular pieces. Python provides great support for defining and using functions.
- A function is a named section of code that performs a specific task.
- It can take inputs (called parameters) and can return an output (called a return value).
- Why Use Functions?
- To avoid code repetition
- To make code more organized and readable
- To promote modularity and reuse
- To enable easy testing and debugging
  - Syntax: def function_name(parameters):
    """Optional: Docstring describing the function."""
    # block of statements
    return result # Optional

- Functions in Python start with the def keyword.

2. Different types of functions: with/without parameters, with/without return values.

❖ Function with No Parameters and No Return Value:

- Definition: These functions neither take any input (parameters) nor return any value.
- Purpose: Used when a fixed task is to be performed without the need for input or result to be sent back.

❖ Function with Parameters but No Return Value:

- Definition: These functions take input values (parameters) but do not return any output.
- Purpose: Used when the function requires some input to perform its task but doesn't need to send a result back.

❖ Function with No Parameters but with Return Value:

- Definition: These functions do not accept input but return a value after processing.
- Purpose: Useful when the function needs to generate or fetch some data without user input.

❖ Function with Parameters and Return Value:

- Definition: These functions accept input values and also return a result.
- Purpose: Most flexible type; used when both input and output are required.

3. Anonymous functions (lambda functions).

- An anonymous function is a function without a name.
- In Python, these are created using the lambda keyword. That's why they are also called lambda functions.
- They are usually used for short, simple operations and are defined in a single line.

➢ Syntax of a Lambda Function:

- lambda arguments: expression
  - lambda → keyword to define an anonymous function.
  - arguments → input values (like parameters).
  - expression → single expression that is evaluated and returned.

- Lambda functions provide a concise and quick way to create simple functions in Python.
- While powerful, they should be used only when the function logic is simple and readable in a single line.

➢ When to Use Lambda Functions?

- For small, simple operations.
- When using functions like map (), filter (), or sorted ().
- When you don't want to formally define a function with def.

Modules

1. Introduction to Python modules and importing modules.

❖ What is a Module in Python?

- A module in Python is simply a file containing Python code (functions, variables, or classes) that you can reuse in other programs.
- It helps in organizing code into smaller and manageable parts.
- Makes your code modular, reusable, and maintainable.
- A module is just a .py file that contains code you want to reuse.

➢ Types of Modules:

- Built-in Modules
- User-defined Modules
- External Modules

➢ Why Use Modules?

- Avoid code repetition
- Divide large programs into smaller files
- Increase readability
- Share functions across multiple programs

❖ Importing Modules in Python

- To use a module in your Python program, you need to import it.
- Modules are an essential part of Python that allow you to write clean, maintainable, and reusable code.
- Whether using built-in or user-defined modules, importing them properly helps keep your programs organized and efficient.

2. Standard library modules: math, random.

❖ math Module:

- The random module is used to generate random numbers, select random elements, shuffle lists, etc.

- Use math when you need to perform mathematical calculations.

➢ How to Import the math Module?

- import math

➢ Commonly Used math Module Functions:

- math.sqrt(x)
- math.floor(x)
- math.ceil(x)
- math.factorial(x)
- math.pi
- math.e
- math.sin(x)

❖ random Module:

- The random module is used to generate random numbers, select random elements, shuffle lists, etc.
- Use random when you want to add random behaviour to your programs.

➢ How to Import the random Module?

- import random

➢ Commonly Used random Module Functions:

- random.random()
- random.randint(a, b)
- random.choice(seq)
- random.randrange(start, stop)
- random.uniform(a, b)
- random.shuffle(list)

3. Creating custom modules.

❖ Steps to Create and Use a Custom Module:

- Step 1: Create a Python File (Module)
- Step 2: Use the Custom Module in Another Python File

➢ Different Ways to Import Custom Modules:

- Syntax:
  - import my_module
  - from my_module import greet
  - from my_module import *
  - import my_module as mm