Chapter 20: LIST – Python Data Structures

Introduction

In Python, a **list** is one of the most versatile and widely used **data structures**. It allows you to store multiple items in a single variable and work with them easily. Lists are **ordered**, **changeable (mutable)**, and **allow duplicate values**. This chapter will help you understand how to create, access, manipulate, and perform operations on lists—essential for developing AI programs where datasets and groupings of information are handled frequently.

20.1 What is a List?

A **list** is a collection of items enclosed in **square brackets [ ]**, separated by **commas**.

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

Key features of lists:

- Lists can store **different data types**: integers, floats, strings, even other lists.
- Lists are **indexed**, starting from `0`.
- Lists are **mutable** (can be changed after creation).

20.2 Creating a List

```
# Creating lists
numbers = [1, 2, 3, 4, 5]
fruits = ["apple", "banana", "mango"]
mixed = [1, "apple", 3.14, True]
empty = []  # empty list
```

20.3 Accessing Elements from a List

```
fruits = ["apple", "banana", "mango"]
print(fruits[0])  # Output: apple
print(fruits[2])  # Output: mango
```

Negative Indexing:

```
print(fruits[-1])  # Output: mango
print(fruits[-2])  # Output: banana
```

20.4 Slicing a List

Slicing allows you to get a **subset** of the list.

```
fruits = ["apple", "banana", "mango", "orange", "grapes"]
print(fruits[1:4])  # Output: ['banana', 'mango', 'orange']
print(fruits[:3])   # Output: ['apple', 'banana', 'mango']
print(fruits[2:])   # Output: ['mango', 'orange', 'grapes']
```

20.5 Modifying List Elements

```
fruits = ["apple", "banana", "mango"]
fruits[1] = "kiwi"
print(fruits)  # Output: ['apple', 'kiwi', 'mango']
```

20.6 Adding Elements to a List

Using `append()`

Adds an element at the end.

```
fruits.append("orange")
```

Using `insert()`

Adds an element at a specific index.

```
fruits.insert(1, "grapes")
```

20.7 Removing Elements from a List

Using `remove()`

Removes the first occurrence of the value.

```
fruits.remove("banana")
```

Using `pop()`

Removes and returns the element at the specified index.

```
fruits.pop(2)
```

Using `del`

Deletes the element by index.

```
del fruits[0]
```

20.8 Traversing a List (Looping Through a List)

```
for fruit in fruits:
    print(fruit)

# Using index
for i in range(len(fruits)):
    print(fruits[i])
```

20.9 List Functions and Methods

Function/Method	Description
`len(list)`	Returns number of elements in list
`min(list)`	Returns smallest item
`max(list)`	Returns largest item
`sum(list)`	Returns sum of all numeric values
`list.sort()`	Sorts the list (ascending)
`list.reverse()`	Reverses the order of the list
`list.index(value)`	Returns index of first occurrence of value
`list.count(value)`	Returns count of a value
`list.copy()`	Returns a shallow copy of the list
`list.clear()`	Removes all elements

20.10 Nested Lists

A list within a list.

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

print(matrix[0][1])  # Output: 2
```

20.11 List Comprehension (Advanced)

A concise way to create lists.

```
squares = [x*x for x in range(1, 6)]
print(squares)  # Output: [1, 4, 9, 16, 25]
```

20.12 Applications of List in AI

- Storing **datasets**.
- Managing **input-output values** in machine learning models.
- Creating **feature vectors**.
- Holding data from **sensors**, **user inputs**, or **text processing**.
- Managing results of **image pixels**, **audio samples**, or **word embeddings**.

Summary

- A **list** in Python is an ordered, mutable collection of elements.
- It can store **different data types** and allows **indexing and slicing**.
- Python provides multiple **methods** like `append()`, `insert()`, `remove()`, and `pop()` to manipulate lists.
- **List comprehension** offers a quick way to generate or modify lists.
- Lists are heavily used in **data science** and **AI**, especially in handling and processing structured data.
