Day 10 (5/9/25): Iterating and Copying Collections in Python

1. Introduction

Collections (lists, tuples, sets, dictionaries) are often **iterated** to process elements. Python also provides **ways to copy collections** without affecting the original data.

2. Iterating Collections

A. Using for Loop

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

Output:

```
apple
banana
cherry
```

B. Using while Loop

```
numbers = [1, 2, 3, 4]
i = 0
while i < len(numbers):
    print(numbers[i])
    i += 1
```

Output:

```
1
2
3
4
```

C. Iterating with enumerate()

enumerate() provides **index along with element**.

```
fruits = ["apple", "banana", "cherry"]
for index, fruit in enumerate(fruits):
    print(index, fruit)
```

Output:

```
0 apple
1 banana
2 cherry
```

D. Iterating Dictionaries

```
student = {"name": "Alice", "age": 20}
for key, value in student.items():
    print(key, ":", value)
```

Output:

```
name : Alice
age : 20
```

3. Copying Collections

A. Shallow Copy

- Creates a **new collection** but nested elements refer to **same objects**.
- Methods:
    - list.copy() → List
    - dict.copy() → Dictionary
    - copy.copy() → All collections

```
import copy
original = [1, 2, [3, 4]]
shallow = copy.copy(original)
shallow[2][0] = 99
print("Original:", original)
print("Shallow:", shallow)
```

Output:

Original: [1, 2, [99, 4]]
Shallow: [1, 2, [99, 4]]

Nested list affected due to shallow copy.

B. Deep Copy

- Creates a **completely independent copy**, including nested elements.
- Method: `copy.deepcopy()`

```
import copy
original = [1, 2, [3, 4]]
deep = copy.deepcopy(original)
deep[2][0] = 99
print("Original:", original)
print("Deep:", deep)
```

Output:

```
Original: [1, 2, [3, 4]]
Deep: [1, 2, [99, 4]]
```

4. Summary

- **Iteration**: Use `for`, `while`, `enumerate()`, or `.items()` for dictionaries.
- **Copying**:
    - **Shallow copy**: top-level copy; nested objects are shared.
    - **Deep copy**: complete independent copy.