📘 Session 9: Mastering Iteration in Python

Iteration & Practice

---

1. Introduction to Iteration

Iteration means **repeating a process multiple times**.
In Python, iteration is most commonly used to loop through **collections** such as lists, tuples, sets, and dictionaries.

- **Without iteration:** You would write code for each item separately.
- **With iteration:** You can process thousands of elements in a loop.

Real-life Examples:

- Taking attendance roll by roll.
- Checking every item in a shopping cart.
- Sending the same message to multiple contacts.

---

2. Iterating with `dict.items()`

Definition

The `.items()` method of a dictionary returns key-value pairs as tuples.

Why use it?

- Iterating only on the dictionary → gives **keys only**.
- `.items()` → gives **both key and value** at the same time.

Syntax

```
for key, value in dictionary.items():
    print(key, value)
```

Example

```
data = {"apples": 3, "bananas": 5}
for fruit, count in data.items():
    print(f"I have {count} {fruit}.")
```

Output:

```
I have 3 apples.
I have 5 bananas.
```

---

3. Iterating with `enumerate()`

Definition

`enumerate()` adds a **counter (index)** to an iterable. It returns pairs `(index, element)`.

Why use it?

- Sometimes you need both the **position** and the **value**.
- Avoids writing manual counters.

Syntax

```
for index, element in enumerate(iterable, start=0):
    print(index, element)
```

Example

```
fruits = ["apple", "banana", "mango"]
for index, fruit in enumerate(fruits, start=1):
    print(f"{index}. {fruit}")
```

Output:

```
1. apple
```

```
2. banana
3. mango
```

---

4. Iterating with `zip()`

Definition

`zip()` combines two or more iterables element-wise into tuples.

Why use it?

- To loop over **multiple lists together**.
- Useful when pairing items like names with scores.

Syntax

```
for a, b in zip(list1, list2):
    print(a, b)
```

Example

```
students = ["Alice", "Bob", "Charlie"]
marks = [85, 92, 78]

for s, m in zip(students, marks):
    print(f"{s} -> {m}")
```

Output:

```
Alice -> 85
Bob -> 92
Charlie -> 78
```

---

5. String Formatting with f-Strings

Definition

f-strings (formatted string literals) allow embedding variables directly in strings.

Why use it?

- Cleaner than concatenation (`+`)
- Easier than `.format()`
- More readable and faster

Syntax

```
f"Text {variable}"
```

Example

```
name = "Rahul"
score = 95
print(f"{name} scored {score} marks.")
```

Output:

```
Rahul scored 95 marks.
```

---

6. Practice Problems

Problem 1: Word Counter

Task: Count how many times each word appears in a sentence.

Input:

```
"python is fun and python is powerful"
```

Output:

```
python: 2
is: 2
fun: 1
and: 1
powerful: 1
```

Solution:

```
sentence = "python is fun and python is powerful"
words = sentence.split()

counter = {}
for word in words:
    counter[word] = counter.get(word, 0) + 1

for w, c in counter.items():
    print(f"{w}: {c}")
```

Problem 2: Parallel Iteration

Task: Match students with marks using `zip()`.

Input:

```
students = ["Rahul", "Sonia", "Amit"]
marks = [90, 85, 78]
```

Output:

```
Rahul -> 90
Sonia -> 85
Amit -> 78
```

Solution:

```
for s, m in zip(students, marks):
    print(f"{s} -> {m}")
```

Problem 3: Enumerate Challenge

Task: Print list of cities with positions starting from 1.

Input:

```
["Delhi", "Mumbai", "Kolkata", "Chennai"]
```

Output:

```
1. Delhi
2. Mumbai
3. Kolkata
4. Chennai
```

Solution:

```
cities = ["Delhi", "Mumbai", "Kolkata", "Chennai"]
for idx, city in enumerate(cities, start=1):
    print(f"{idx}. {city}")
```

---

7. Summary

- **Iteration**: Repeating actions over collections.
- **dict.items()**: Loop with key-value pairs.
- **enumerate()**: Loop with index + value.
- **zip()**: Loop over multiple lists together.
- **f-strings**: Cleaner string formatting.

✅ These are must-have tools for writing clean, efficient Python code.
✅ Practice them on platforms like **HackerRank / LeetCode**.