**7. Working with Dictionaries**

1. **Iterating over a dictionary using loops**

Dictionary Fundamentals :

* A Python dictionary is implemented as a hash table (O(1) average time complexity for lookup/insertion/deletion)
* Internally, CPython maintains two structures: a sparse hash table and a dense entry array for keys/values—allowing efficient memory layout and ordered iteration since Python 3.6+ .

What Does Iterating Over a Dictionary Mean?

* Iterating directly (for key in mydict:) is shorthand for iterating over keys using the iterator protocol. This yields keys in insertion order .
* Methods on a dict:
  + .keys() – returns a dynamic view of keys.
  + .values() – returns a dynamic view of values.
  + .items() – returns key–value tuple pairs.

These are implemented efficiently in C, providing fast iteration without constructing new lists

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

**Conceptual Mechanics**

1. zip(keys, values)
   * zip() creates a lazy iterator that pairs items index‑by‑index, stopping at the shortest list
   * Being lazy, it doesn't allocate memory for all pairs at once—making it memory-efficient for large lists .
2. dict(zip(...))
   * Takes the zip iterator and builds a dictionary in one pass.
   * Internally, each tuple (k, v) is inserted into the hash-based dict in *amortized* O(1) per insertion.

Example :

keys = ['name', 'age', 'city']

values = ['Alice', 30, 'New York']

my\_dict = dict(zip(keys, values))

1. **Counting occurrences of characters in a string using dictionaries.**

* Hash table mechanics  
  Counting characters uses a dictionary (hash table) where each character is a key. Insertion and lookup operations execute in O(1) average time due to hashing, making single-pass counting linear: O(n) for a string of length *n*
* Manual loop approach

freq = {}

for c in s:

freq[c] = freq.get(c, 0) + 1

* Performs one dictionary lookup and update per character → O(n) total.
* Uses O(u) space, where *u* = number of unique characters .
* collections.Counter class

from collections import Counter

freq = Counter(s)

* Behaves like a dict, linear-time in most cases: O(n) for counting
* Worst-case can degrade (due to hash collision), but that's rare; average is still O(n)

**Example :**

**s = "hello world"**

**freq = {}**

**for c in s:**

**if c in freq:**

**freq[c] += 1**

**else:**

**freq[c] = 1**