1.What are data structures, and why are they important3?

- **Data structures** are ways to store and organize data so that operations like retrieval, insertion, deletion, and traversal can be done efficiently. They're essential because they help optimize performance in algorithms and manage large datasets effectively.

2. **Difference between mutable and immutable data types (with examples):**

- **Mutable: Can be changed after creation.**
  Example: `list`, `dict`, `set`

```
my_list = [1, 2, 3]
my_list[0] = 100  # allowed
```

Immutable: Cannot be changed after creation.
Example: `int`, `str`, `tuple`

```
my_str = "hello"
```

3. Lists vs Tuples in Python:

-

Feature	List	Tuple
Mutability	Mutable	Immutable
Syntax	`[1, 2, 3]`	`(1, 2, 3)`
Performance	Slower	Faster
Use case	Dynamic data	Static data

4. How dictionaries store data:

- **Dictionaries store key-value pairs using hashing. Each key is hashed to a specific location in memory, making lookup fast (average O(1) time complexity).**

5. Why use a set instead of a list in Python:

- **Sets store unique elements only.**
  They offer faster membership testing (`in` operator) due to internal hashing.
  Useful when duplicate values are not desired.

6. What is a string in Python, and how is it different from a list?

- **A string is a sequence of characters, immutable.**
  A list is a sequence of elements (can be of any type), mutable.

7. How tuples ensure data integrity:

- **Tuples are immutable, so once created, their content cannot be changed. This prevents accidental modification, ensuring data safety and consistency.**

8. What is a hash table, and how does it relate to dictionaries?

- **A hash table is a data structure that maps keys to values using a hash function.**
  Python dictionaries are built using hash tables.

9. Can lists contain different data types in Python?

- **Yes, Python lists can hold heterogeneous data.**

10. Why are strings immutable in Python?

- To ensure hash consistency (strings are used as dictionary keys).
For memory efficiency (shared across programs).
For thread-safety, as changes can't occur mid-operation.

11. Advantages of dictionaries over lists:

- Faster lookup (O(1) vs O(n)).
Key-value mapping is more intuitive for data like configs, records, etc.

12. When to use a tuple over a list:

- When the data:
- Should **not change** (e.g., coordinates, dates).
- Needs to be used as a **dictionary key** (only hashable types allowed)

13. How do sets handle duplicate values?

- **Sets automatically remove duplicates.**

14. How does the “in” keyword differ for lists and dictionaries?

- **List: Checks for value.**

  Dict: Checks for key, not value.

15. Can you modify the elements of a tuple? Why/Why not?

- No, tuples are **immutable**. You cannot change, add, or remove elements once defined.

16. What is a nested dictionary? Use case?

- **A nested dictionary is a dictionary inside another dictionary.**

17. Time complexity of accessing dictionary elements:

- **Average Case: O(1)**
  Worst Case: O(n) (very rare, due to hash collisions)

18. When are lists preferred over dictionaries?

- **When order matters.**
  When storing sequential data.
  When no need for key-value mapping.

19. Why are dictionaries considered unordered, and how does it affect retrieval?

- **Before Python 3.7, dictionaries didn’t preserve insertion order.**
  From Python 3.7+, they do maintain order, but conceptually they're still unordered collections — retrieval is by key, not position.

20. Difference between list and dictionary (data retrieval):

- **List: Retrieve by index.**
  `my_list[0]`
-
- **Dictionary: Retrieve by key.**
  `my_dict['name']`