Data types and structures

1. What are data structures, and why are they important

   -> A **data structure** is a way of organizing, storing, and managing data in a computer so it can be used efficiently.

   Importance of Data Structures :
   Efficiency – They make operations like searching, inserting, deleting, or sorting much faster.
   (Example: Finding a friend in a phone book is easier if it's organized alphabetically — same idea in data structures.)

   Memory Management – They help optimize storage space.
   (Example: Arrays use continuous memory, while linked lists use scattered memory with pointers.)

   Problem Solving – Many real-world problems map naturally to certain data structures.
   (Example: Social networks → graphs, Undo/Redo feature → stack, CPU task scheduling → queue.)

   Foundation of Algorithms – Almost all algorithms are designed with specific data structures in mind. Without them, efficient algorithms wouldn't exist.

2. Explain the difference between mutable and immutable data types with examples

   ->**Mutable Data Types**

- **Definition**: Data that **can be changed/modified after creation**.
- **Behavior**: If you update a mutable object, the same memory location is updated — no new object is created.

Examples in Python:

- `list`
- `dict` (dictionary)
- `set`

Immutable Data Types

- **Definition**: Data that **cannot be changed after creation.**
- **Behavior**: If you "change" it, a **new object** is created at a different memory location.

Examples in Python:

Data types and structures

- int
- float
- string (str)
- tuple
- bool

3. What are the main differences between lists and tuples in Python ?

-> **Lists**

1. **Mutable** → You can add, remove, or change elements after creation.
2. **Slower** than tuples because they allow modifications and need extra memory.
3. **More methods** available (like append(), remove(), sort()).

Tuples

1. **Immutable** → Once created, you cannot change, add, or remove elements.
2. **Faster** than lists because of immutability and lower memory usage.
3. **Fewer methods** available (count(), index() only).

4. Describe how dictionaries store data.

-> **Key–Value Pairs**

- A dictionary stores data as **pairs**: each item has a **key** and a **value**.

Hash Table Under the Hood

- Python dictionaries are implemented using a **hash table**.

Data types and structures

- When you create a key, Python runs it through a **hash function** → converts it into a number (hash value).
- That hash value decides where the key–value pair will be placed in memory (like an index).

Fast Lookup

- Because of hashing, finding values is very quick (average **O(1)** time).

5.Why might you use a set instead of a list in Python ?

-> **Reason to use set instead of a list :**

No Duplicates → A set automatically removes duplicate values, while a list can store duplicates.

Faster Lookup → Checking if an item exists in a set is much faster (`O(1)` average) than in a list (`O(n)`), because sets use hashing.

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

-> A **string** is a sequence of characters enclosed in quotes (`' '` or `" "`).

Difference Between String and List

1. **Data Type Stored**
   - String → stores only characters.
   - List → can store elements of any type (numbers, strings, even other lists).
2. **Mutability**
   - String → **immutable** (cannot be changed after creation).
   - List → **mutable** (can be changed by adding, removing, or updating elements).
3. **Methods/Operations**
   - String → has text-related methods (`upper()`, `lower()`, `replace()` etc.).
   - List → has sequence methods (`append()`, `remove()`, `sort()` etc.).

Data types and structures

7.How do tuples ensure data integrity in Python ?

-> **Immutability** → Once a tuple is created, its elements **cannot be changed, added, or removed**. This prevents accidental modification of important data.

Hashability → Since tuples are immutable, they can be used as **keys in dictionaries** or **elements in sets**, ensuring data remains fixed and reliable.

Predictability in Programs → Because data inside a tuple cannot change, programs can rely on it for **consistent values** (e.g., fixed configuration, constant lookup values).

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

->A **hash table** is a data structure that stores data in **key–value pairs**.
It uses a **hash function** to convert the key into a hash value (an integer), which decides where the value will be stored in memory.
This allows **very fast lookups, insertions, and deletions** (average case → `O(1)` time).

Relation of hash table with dictionaries:

Underlying Implementation → Python dictionaries are **built using hash tables**.

- Keys are hashed → hash value decides the “bucket” (memory slot) where the value is stored.

Fast Access → Because of hashing, accessing a value by its key is much faster than searching through a list

Key Uniqueness → In a dictionary, keys must be unique. If two keys have the same hash (rare case = **collision**), Python uses techniques like **open addressing** to resolve it.

9. Can lists contain different data types in Python ?

->Yes — Python **lists can store different data types** in the same list.

A list is just a sequence of objects, and since Python is dynamically typed, it doesn’t restrict the type of items stored.

10. Explain why strings are immutable in Python ?

-> Reasons Strings are Immutable

1. **Memory & Efficiency**
    - Python often reuses the same string object in memory (string interning).
    - If strings were mutable, changing one string would accidentally change all references to it, wasting memory and causing bugs.

Data types and structures

2. **Hashing & Dictionary Keys**
    - Strings are commonly used as **keys in dictionaries** and **elements in sets**.
    - Keys need to stay constant (hash value shouldn't change).
    - Immutability guarantees that once a string is created, its hash stays the same.
3. **Security & Reliability**
    - Because strings are widely used (e.g., file paths, URLs, identifiers), immutability ensures they cannot be altered unexpectedly, keeping programs more secure and predictable.

11.What advantages do dictionaries offer over lists for certain tasks ?

->**Advantages of Dictionaries Over Lists**

1. **Fast Lookup by Key**
    - In a list, searching for an item takes time (`O(n)`), but in a dictionary, looking up a value by its key is very fast (`O(1)` average) thanks to hashing.
2. **Key–Value Mapping**
    - Lists only store values in order, but dictionaries let you associate each value with a unique key.
    - This makes data more meaningful and easier to access.
3. **No Need for Index Management**
    - In lists, you access data by position (`list[0]`, `list[1]`), which is less intuitive.
    - In dictionaries, you directly use keys (`dict["name"]`), making the code more readable and self-explanatory.

12. Describe a scenario where using a tuple would be preferable over a list ?

-> **Scenario Where Tuple is Preferable**

- When you need to store **fixed, unchangeable data** that should not be modified during the program.
- Example: **Geographic coordinates (latitude, longitude)**. Once defined, these values represent a specific location and should not be altered accidentally.
- Using a tuple ensures **data integrity**, because immutability prevents changes.

13. How do sets handle duplicate values in Python ?

Data types and structures

1. **Uniqueness Rule** → A set automatically stores only **unique elements**.
2. **Duplicate Removal** → If you try to add a duplicate value, Python simply **ignores it**, keeping only one copy.
3. **Reason** → Sets are based on **hash tables**, which cannot hold two identical keys.

14. How does the “in” keyword work differently for lists and dictionaries3

-> **In Lists**

- The `in` keyword checks whether a given **value** exists in the list.
- It performs a sequential search through the elements.
- Time complexity: **O(n)** in the average case.

In Dictionaries

- The `in` keyword checks whether a given **key** exists in the dictionary, **not the value**.
- It uses hashing, so lookups are very fast.
- Time complexity: **O(1)** in the average case.

15. Can you modify the elements of a tuple? Explain why or why not ?

->No — **you cannot modify the elements of a tuple** once it is created.

Reason :

1. **Immutability** → Tuples are designed to be **immutable**, meaning their contents cannot be changed, added, or removed after creation.
2. **Data Integrity** → This immutability ensures that tuples provide a stable, unchanging sequence, which is useful when storing constant or fixed data.
3. **Hashability** → Because tuples are immutable, they can be used as **keys in dictionaries** and **elements in sets** — something mutable structures like lists cannot do.

16.What is a nested dictionary, and give an example of its use case?

->A **nested dictionary** is a dictionary that contains another dictionary as its value.

It allows storing data in a **hierarchical or structured form**, like layers.

Example of use case:

Student Records System: Each student’s information (like age, marks, subjects) can be stored inside their own dictionary, and all students can be grouped in one main dictionary.

Data types and structures

This makes data organized, structured, and easy to access.

17. Describe the time complexity of accessing elements in a dictionary.

-> **Time Complexity of Dictionary Access**

1. **Average Case → O(1)** (constant time)
    - Because dictionaries use a **hash table** internally.
    - The key is hashed, and Python jumps directly to the memory location where the value is stored.
2. **Worst Case → O(n)**
    - This happens only if there are many **hash collisions** (different keys mapping to the same hash bucket).
    - In practice, Python's optimized hashing makes this very rare.

18. In what situations are lists preferred over dictionaries?

->**Situations Where Lists Are Preferred**

1. **Ordered Data** → When the order of elements matters, because lists maintain the sequence of insertion.
2. **Sequential Access** → When you primarily need to iterate over all elements rather than access by key.
3. **Homogeneous Data** → When storing similar items without needing key–value mapping, e.g., a collection of numbers, names, or objects.

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

-> **Reason to considered dictionaries unordered:**

Dictionaries store **key–value pairs using a hash table**, not in a sequential order.

- The position of items in memory depends on the **hash of the key**, not the order of insertion.

Procedure of effecting data retrieval:

Access by Key Only → You cannot rely on index positions like lists; you must use the key to retrieve a value.

No Positional Access → You cannot ask for the "first" or "third" item directly without converting keys or values to a list.

Fast Lookups → Despite being unordered, dictionaries allow very fast **access by key**, which is usually more important than order.

Data types and structures

20. Explain the difference between a list and a dictionary in terms of data retrieval.

-> **List vs Dictionary: Data Retrieval**

1. **Access Method**
    - **List** → You retrieve elements by **index position** (e.g., first, second, third element).
    - **Dictionary** → You retrieve elements by **key**, not by position.
2. **Speed**
    - **List** → Searching for a specific value requires **scanning elements one by one**, so it can be slower (`O(n)` time).
    - **Dictionary** → Access by key is **fast** (`O(1)` average time) due to hashing.
3. **Use Case**
    - **List** → Best for **ordered collections** where sequence matters.
    - **Dictionary** → Best for **key–value mapping** where fast lookup by key is needed.

.