**DATA STRUCTURES**

1. What are data structures, and why are they important?Data structures are ways to organize and store data efficiently so that operations like access, insertion, and deletion can be performed easily. They are important for writing efficient and clean code.

2. Explain the difference between mutable and immutable data types with examples.  
  
Mutable data types (like list, dict, set) can be changed after creation. Immutable data types (like tuple, str, int) cannot be modified once created.  
Example:

* list1 = [1, 2] → You can do list1.append(3)
* tuple1 = (1, 2) → You cannot do tuple1[0] = 3

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

* Lists are mutable; tuples are immutable.
* Lists use square brackets []; tuples use parentheses ().
* Tuples are faster and use less memory.

4. Describe how dictionaries store data.  
Dictionaries store data as key-value pairs. Each key must be unique, and values are accessed using keys.

5. Why might you use a set instead of a list in Python?  
Use a set when you need to store unique items and perform fast membership tests or set operations like union and intersection.

6. What is a string in Python, and how is it different from a list?  
A string is a sequence of characters and is immutable. A list is a mutable sequence that can contain elements of different types.

7. How do tuples ensure data integrity in Python?  
Tuples are immutable, so once created, their data cannot be altered. This helps prevent accidental modifications.

8. What is a hash table, and how does it relate to dictionaries in Python?  
A hash table is a data structure that maps keys to values using a hash function. Python dictionaries use hash tables internally for fast key-based access.

9. Can lists contain different data types in Python?  
Yes, lists can contain a mix of integers, strings, floats, objects, or even other lists.

10. Explain why strings are immutable in Python.  
Strings are immutable to ensure data consistency and thread safety, and also to enable memory optimization through string interning.

11. What advantages do dictionaries offer over lists for certain tasks?  
Dictionaries provide faster lookups using keys and are ideal for associating related data (e.g., storing user info by user ID).

12. Describe a scenario where using a tuple would be preferable over a list.  
Use a tuple when you need a fixed set of values that shouldn’t change—like coordinates ((x, y)) or days of the week.

13. How do sets handle duplicate values in Python?  
Sets automatically remove duplicate values and store only unique elements.

14. How does the "in" keyword work differently for lists and dictionaries?

* In a list: checks if a value exists.
* In a dictionary: checks if a key exists.

15. Can you modify the elements of a tuple? Explain why or why not.  
No, tuples are immutable. Once created, their elements cannot be changed.

16. What is a nested dictionary, and give an example of its use case?  
A nested dictionary is a dictionary inside another dictionary.  
Example: students = {"John": {"math": 90, "science": 85}}

17. Describe the time complexity of accessing elements in a dictionary.  
Average-case time complexity is O(1) due to hash table implementation.

18. In what situations are lists preferred over dictionaries?  
When the data is ordered and you don't need key-based access—like storing a list of names or numbers.

19. Why are dictionaries considered unordered, and how does that affect data retrieval?  
(Older versions) Dictionaries didn’t preserve insertion order. In Python 3.7+, order is preserved, but keys are still accessed via hashes, not positions.

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

* List: Access by index (position-based).
* Dictionary: Access by key (key-based), which is generally faster and more meaningful.