1: What are data structures, and why are they important?
Answer: Data structures are ways of organizing and storing data in a computer so it can be used efficiently. Examples include lists, tuples, dictionaries, sets, stacks, queues, and trees.
They are important because they help in:
1.Efficient data access
2.Reducing time complexity
3.Managing large data easily
4.Enabling easier implementation of algorithms

2: Difference between mutable and immutable data types with examples:
Answer:
>> Mutable: Can be changed after creation (e.g., list, dictionary, set)
>> Immutable: Cannot be changed after creation (e.g., tuple, string, int)
Example :

```
# Mutable
my_list = [1, 2, 3]
my_list.append(4)  # List changes

# Immutable
my_string = "hello"
# my_string[0] = 'H'  # Error: strings are immutable
```

3: What are the main differences between lists and tuples in Python3
Answer:

Lists:	Tuples:
Mutable	Immutable
Slower performance	Faster performance
Defined by []	Defined by ()
Can be changed	Cannot be changed

4: Describe how dictionaries store data
Answer:
Dictionaries store data in key-value pairs using a hash table. Each key is unique and points to its associated value:
Example:

```
person = {'name': 'Alice', 'age': 30}
```

5: Why might you use a set instead of a list in Python?
Answer:
1.To store unique values automatically
2.Faster membership testing
3.Useful in removing duplicate
Example:

```
my_set = set([1, 2, 2, 3])  # Result: {1, 2, 3}
```

6: What is a string in Python, and how is it different from a list?
Answer:
>> A string is a sequence of characters (immutable).
>> A list is a sequence of elements (mutable, can hold any data types).
Example:

```
my_string = "hello"
my_list = ['h', 'e', 'l', 'l', 'o']
```

7: How do tuples ensure data integrity in Python?
Answer:
>> Tuples are immutable, meaning once created, their data cannot be changed.
>> This ensures that the data stored remains constant and protected.

---
8. What is a hash table, and how does it relate to dictionaries in Python?
Answer:
>> A hash table stores data as key-value pairs using a hash function.
>> Python’s dict is built on hash tables, allowing O(1) average time complexity for lookups
---
9. Can lists contain different data types in Python?
Answer:
Yes! Lists are heterogeneous and can store any data types together:
Example:
my_list = [1, "Hello", 3.14, True]

10.Why are strings immutable in Python?
Answer:
>> Strings are immutable to improve performance and memory optimization.
>> This also makes strings hashable, allowing them to be used as dictionary keys.
---
11. What advantages do dictionaries offer over lists for certain tasks
Answer:
>> Faster lookups (O(1) average case)
>> Easy mapping between unique keys and values
>> Useful for associative data like records or JSON-like structures
---
12. Describe a scenario where using a tuple would be preferable over a list
Answer:
>> When you want to store data that should not change like coordinates (x, y)
>> Use as dictionary keys or elements in a set (since lists are unhashable)
>> Other Real-life Examples:
*RGB color values (255, 0, 0)
*Days of the week ('Mon', 'Tue', 'Wed')
*Database records fetched as tuples
*Storing configuration settings
---
13. How do sets handle duplicate values in Python?
Answer:
>> Sets automatically remove duplicates
Example:
my_set = {1, 2, 2, 3}
print(my_set) # Output: {1, 2, 3}
---
14. How does the in keyword work differently for lists and dictionaries?
Answer:
>> For lists: in checks if a value exists in the list.
>> For dictionaries: in checks if the key exists, not the value.
Example:
print(2 in [1, 2, 3]) # True (value)
print('name' in {'name': 'John'}) # True (key)
---
15. Can you modify the elements of a tuple? Why or why not?
Answer:
No, tuples are immutable. You cannot change, add, or remove elements after creation.
---

16. What is a nested dictionary? Example use case:
Answer:
A dictionary inside another dictionary
Example:
```
students = {
    'student1': {'name': 'Alice', 'age': 20},
    'student2': {'name': 'Bob', 'age': 22}
}
```

Use case: Storing structured data like records.
---
17. Describe the time complexity of accessing elements in a dictionaryP
Answer:
Time Complexity of Accessing Elements in a Dictionary (Python):
🔎 Average Case:
>>O(1) — Constant time
>>Accessing a value by key is very fast because Python dictionaries are implemented using hash tables.
>>The key is hashed, and the hash points directly to the value's memory location.
Example:
```
my_dict = {'name': 'Alice', 'age': 30}
print(my_dict['name'])  # O(1)
```
⚠️ Worst Case:
>> O(n) — Rare but possible when there are many hash collisions.
>> Python handles collisions using techniques like open addressing, but in extreme cases (bad hash function or malicious input), lookup time can degrade.

18.When are lists preferred over dictionaries?
Answer:
>> When the data is ordered
>> When index-based access is needed
>> When you care about the order of insertion

19.Why are dictionaries considered unordered (until Python 3.6)?
Answer:
>> Dictionaries were unordered because they use a hash table for storage.
>> Since Python 3.7, dictionaries maintain insertion order but are still fast because of hashing.
---
20. Explain the difference between a list and a dictionary in terms of data retrieval.
Answer:

List	Dictionary
Access by index	Access by key
O(1) for index access	O(1) average case for key lookup
Sequential data	Key-value mapping