Data Types and Structures Questions

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

Ans > Data structures are ways of organizing, storing, and managing data in a computer so it can be used efficiently.

1. **Stored in memory**
2. **Accessed**
3. **Modified**

Data Structures in Python

a. String
b. List
c. Tuples
d. Sets
e. Dictionary

Importance

1. Efficiency

- The right data structure improves **speed and memory usage**.
- Example: Using a set instead of a list for membership tests is much faster.

2. Organization

- Helps in organizing complex data logically.
- Example: A tree structure can model folder hierarchies.

3. Solving Real-World Problems

- Algorithms depend on data structures (e.g., searching, sorting, graph traversal).
- Example: Social networks use graphs to model relationships.

4. Scalability

- Good data structures handle **larger datasets** better.

5. Maintainability

- Cleaner, well-structured code is easier to read, debug, and extend.

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

Ans > **Mutability**: Objects/Container Whose state or value can be change after they are created are called as Mutable Objects or Containers.

Examples: List, Set, Dictionary.

1.

```
lst = ['Ankur','Aman','Rphit']
lst[0] = 'Damini'
lst
['Damini', 'Aman', 'Rphit']
```

2.

```
s ={'ankur' ,'aman' ,'rohit'}
s.add('damini')
s
```

Immutable: : Objects/Container Whose state or value can ‘t be change after they are created are called as Immutable Objects or Containers.

Examples: Int, Float, Boolean , String , Tuples .

1.

```
name = 'Ankur'
 name[0] = 'a'
 name
```

Output:

TypeError : 'str' object does not support item assignment

2.

```
t =('ankur','aman','rohit')
 t[0] = 'Damini'
 t
```

TypeError : 'tuple' object does not support item assignment

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

Ans>

List	Tuples
1.list is mutable	1.Tuples is immutable
2.Lists denoted by '[]'	2.Tuples denoted by '()'
3.We can modify the data in list.	3. we can not modify the data in Tuples.
4.when data may change like phone, email etc.	4.when data are fix like Adhar, PAN etc.

4.Describe how dictionaries store data?

Ans>A dictionary in Python is a built-in data structure that stores key-value pairs. It's very useful when you want to associate pieces of data — for example, names and ages, or product IDs and prices.

d ={"name":"Ankur","email":"AV@gami.com","contact":"1234"}

type(d) -> dict

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

Ans> **Sets:**

1.To store only unique values — sets automatically remove duplicates.

2.No need to maintain order — sets are unordered, so use them when order doesn't matter.

3.Set operations are built-in — like union, intersection, and difference.

4.Cleaner code for de-duplicating data — converting a list to a set quickly removes repeats.

```
s ={1,1,2,3,1,3,1,3,1,2,'Ankur','Ankur','ankur','ankur'}
s
output : {1, 2, 3, 'Ankur', 'ankur'}
```

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

Ans> **String :** it is a sequence of character representation and manipulated textual data.

String are represented using single quote 'String' ,double quote "String" and triple quote '''String'''.

str = "PWSkills"

Type(str) -> Str

String	List
Immutable	Mutable
Represented with quote ‘Hello’ single double and triple.	Represented with [‘A’,’B’,’C’]
Modification not allowed	You can add, remove, or change items
Can store Text or character data	Can store anything heterogenous
Str = ”Hello” Str[0] = ‘h’ -> Error: strings are immutable	l = ['h', 'e', 'l', 'l', 'o'] l[0] = 'H' print(l) # ['H', 'e', 'l', 'l', 'o']

7. How do tuples ensure data integrity in Python?

Ans> A tuple in Python is an immutable sequence — once it's created, it cannot be changed. This immutability is what helps protect data integrity.

You **cannot add**, **remove**, or **modify** elements in a tuple.

This means the data stays **exactly as it was defined**.

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

Ans> A dict in Python is a **built-in implementation of a hash table**.

It stores key-value pairs using **hashing**.

That’s why:

1. Keys must be hashable (like strings, numbers, or tuples).
2. You get fast lookups, inserts, and deletes — all in about O(1) time.
3. You can look up a value by key almost instantly.
4. No need to loop through every item (like in a list).

9. Can lists contain different data types in Python?

Ans> Yes, lists in Python can contain different data types.

In list we can store .

Integers, Strings, Floats ,Booleans.

Other lists, tuples, sets, or even functions and objects

```
my_list = [42, "hello", 3.14, True, [1, 2], {"a": 1}, (9, 8)]

print(my_list)  # [42, 'hello', 3.14, True, [1, 2], {'a': 1}, (9, 8)]
```

10. Explain why strings are immutable in Python?

Ans> In Python, strings are immutable, meaning once a string is created, it cannot be changed.

```
s = "hello"

s[0] = "H"  # Error: strings don't support item assignment
```

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

Ans>

1. **Faster Lookups:**

   Accessing values by key is very fast (average O(1) time).

   In lists, you often need to search or know the index (O(n) time).

2. **Key-Value Pairs:**

   Dictionaries store data in labelled pairs (key: value), making them ideal for structured data.

3. **Improved Readability:**

   Code is easier to understand when you use keys like "name" instead of index numbers like [0].

4. **No Need to Remember Positions:**

   You don't need to remember the order of data — just use the key.

5. **Unique Keys Ensure Data Integrity:**

   Dictionaries don't allow duplicate keys, reducing mistakes in data entry or updates.

6. **Flexible Data Representation:**

   You can represent complex records (like user profiles, settings, etc.) more naturally.

7. **Easy Data Updates:**

   You can directly update a value using its key, without worrying about index shifts.

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

Ans> Storing Aadhaar Information

Suppose you're building a system to store people records, and each people has:

Aadhaar number (unique and fixed)

Name

```
# Tuple used to store Aadhaar data

people = ("1234-5678-9012", "Ankur")

# Dictionary where the tuple is the key

people_db = {

  ("1234-5678-9012", "Raj Kumar"): {"address": "New Delhi", "status": "Active"},

  ("5678-1234-9012", "Anita Singh"): {"address": "Mumbai", "status": "Inactive"}

}
```

13. How do sets handle duplicate values in Python?

Ans> Python sets are implemented using a hash table, like dictionaries. Each element is processed by a hash function to generate a unique hash value. This hash value determines the internal index where the element will be stored. When you add a new element, Python computes its hash and checks if an element with the same hash already exists in the set. If such an element exists and is equal to the new one, the new element is not added, effectively discarding the duplicate. Otherwise, the element is stored in the appropriate slot in the hash table.

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

Ans> The in keyword is used to check for membership — it tells you whether a value exists inside a collection or sequence.

x in y checks if x is a member of y

The in keyword works by calling the __contains__() method on an object (if available), or by looping through its elements.
In dictionaries, it uses fast hash-based lookup for keys. In sequences like lists or strings, it performs a linear search.

```
text = "Python programming"
```

```
print("Python" in text)  # True

print("Java" in text) #False
```

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

Ans> You cannot modify a tuple's elements because tuples are immutable by design — they protect data from being accidentally changed.

1. A tuple is an immutable data type in Python.
2. Once it's created, you cannot change, add, or remove its elements.

```
my_tuple = (10, 20, 30)

my_tuple[1] = 99 # TypeError: 'tuple' object does not support item assignment
```

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

Ans> A nested dictionary is a dictionary inside another dictionary.
It lets you organize complex, structured data in a hierarchical format.

```
employees = {

   "Emp1": {"name": "Ankur","Dept": "QA","Salary": 100},

   "Emp2": {"name": "Priya", "Dept": "HR", "Salary": 120},

   "Emp3": {"name": "Ravi","Dept": "Engineering","Salary": 150},

   "Emp4": { "name": "Neha", "Dept": "Finance", "Salary": 130},

   "Emp5": { "name": "Amit",  "Dept": "IT", "Salary": 140}

}
```

Storing Structured Records → For example, student, employee, or customer data with multiple fields.

Grouped Configurations → Application settings grouped by module, like {"database": {"host": ..., "port": ...}}.

Inventory Systems → Organizing items by category, with each item holding details like price and quantity.

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

Ans> Accessing an element by key in a dictionary is on average O(1) — constant time.

This means it takes roughly the same amount of time no matter how big the dictionary is.

The reason: dictionaries use a hash table internally, allowing direct lookup via the key's hash.

In rare cases, due to hash collisions, access time can degrade to O(n), where n is the number of items, but Python's hashing and resizing strategies make this very uncommon.

18. In what situations are lists preferred over dictionaries?

Ans> Lists are preferred over dictionaries when the order of elements matters, when you need to access items by their position (index), when the data doesn't require labels or keys, when you want to allow duplicate values, and when you're working with simple, sequential collections like names, numbers, or tasks.

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

Ans> dictionaries were considered unordered because they stored key-value pairs based on hashing, not in the order you added them. So, if you inserted items in a certain order, you couldn't rely on them being in the same order when retrieved.

Dictionaries preserve insertion order by default.

This means items are retrieved in the same order they were added.

However, dictionaries are still technically considered unordered mappings in the language specification, because their main purpose is fast access by key, not maintaining order.

It affects data retrieval by allowing fast access to values using keys, but traditionally without guaranteeing the order of items; although modern Python versions preserve insertion order, dictionaries are still primarily optimized for quick key lookups, not for ordered data access.

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

Ans>

List	Dictionary
List fetch the data based on index value, you can access element based on the position . lst =[‘a’,’b’,’c’] lst.get[0] = a	Dictionary fetch the data based on key value, you access values using unique identifiers (keys), not positions. Dic = {"name":"Ankur","email":"AV@gami.com","contact":"1234"} Dic[‘name’] = Ankur