PYTHON – DATA STRUCTURE

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

> Data structures are ways to organize and store data in a computer so it can be accessed and modified efficiently. They help in managing large amounts of data, improving performance, and enabling easy data manipulation.

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

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

```
my_list = [1, 2, 3]
my_list[0] = 10  # Valid
```

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

```
my_str = "hello"
my_str[0] = "H"  #error
```

3. Main differences between lists and tuples in Python:

Feature	List	Tuple
Mutability	Mutable	Immutable
Syntax	`[1, 2, 3]`	`(1, 2, 3)`
Performance	Slower	Faster (read-only)
Use Case	Data that changes	Fixed data (e.g., coordinates)

4. How do dictionaries store data?

> Dictionaries store data as **key-value pairs** using a structure called a **hash table**, allowing fast access by key.
Example:

```
person = {"name": "Alice", "age": 25}
```

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

> Sets are used when:

- You want **unique elements only**.
- You need **fast membership testing**.

PYTHON – DATA STRUCTURE

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

> A **string** is a sequence of characters (immutable), while a **list** is a sequence of any data types (mutable).

```
my_str = "hello"  # Cannot change characters
my_list = ['h', 'e', 'l', 'l', 'o']  # Can be modified
```

7. How do tuples ensure data integrity in Python?

> Tuples are **immutable**, meaning data can't be altered. This makes them ideal for fixed data like dates or configurations.

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 **hashing function**. Python's `dict` is built on hash tables for quick key-based access.

9. Can lists contain different data types in Python?

> Yes. Python lists are flexible and can store **mixed data types**.

```
mixed = [1, "hello", 3.14, True]
```

10. Why are strings immutable in Python?

> Strings are immutable for **performance, security, and thread safety**. Any change creates a new string instead of modifying the original.

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

> Dictionaries offer:

- **Faster lookups** by key (O(1) time).
- **Clear mapping** between keys and values.
- **Better organization** for labeled data.

12. How do sets handle duplicate values in Python?

> Sets **automatically remove duplicates**.

```
s = {1, 2, 2, 3}
print(s  # Output: {1, 2, 3}
```

PYTHON – DATA STRUCTURE

13. Scenario where a tuple is better than a list:

> When storing **coordinates (x, y)** or **database records** that shouldn't change

```
location = (18.5, 79.1)  # Tuple ensures data stays unchanged
```

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

- In **lists**, `in` checks for values.
- In **dictionaries**, `in` checks for **keys**, not values.

```
'a' in ['a', 'b']  # True
'a' in {'a': 1, 'b': 2}  # True (key check)
1 in {'a': 1, 'b': 2}  # False (1 is a value, not a key)
```

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

> No. Tuples are **immutable**, so their elements can't be changed after creation. This ensures consistency and safety.

16. What is a nested dictionary? Give an example.

> A dictionary **inside another dictionary**.
Example:

```
employee = {
  "name": "John",
  "address": {
    "city": "Hyderabad",
    "zip": "500001"
  }
}
```

17. Time complexity of accessing elements in a dictionary:

- Average and best case: **O(1)** (constant time)
- Worst case (rare): **O(n)** (if many key collisions occur)

18. When are lists preferred over dictionaries?

- When you need **ordered data** without key-value mapping.

- When iterating over **sequential data**.

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

> Before Python 3.7, dictionaries didn't maintain insertion order. Now they do, but still, retrieval is done by **key**, not position—so you can't access values by index like lists.

20. Difference between list and dictionary in data retrieval:

- **List:** Access by **index (position)**

```
names[0]  # First item
```

- **Dictionary:** Access by **key (label)**

```
person["name"]  # "saad"
```

For answers to practical questions please refer to the link given below:

Data Structure Asgnm.ipynb - Colab