1.What are data structures, and why are they important
Ans. Data structures are ways to organize and store data in a computer so that it can be efficiently accessed and modified. They are important because they enable efficient data management, improve code performance, and simplify complex problems.

Common data structures include:

1. Arrays
2. Linked Lists
3. Stacks
4. Queues
5. Trees
6. Graphs
7. Hash Tables

2. Explain the difference between mutable and immutable data types with examples.
Ans- Mutable Data Type Can be changed after creation.

Examples:
1. Lists (`[1, 2, 3]`)
2. Dictionaries (`{"name": "John"}`)
3. Sets (`{1, 2, 3}`)

Immutable Data Type Cannot be changed after creation.

Examples:
1. Tuples (`(1, 2, 3)`)
2. Strings (`"hello"`)
3. Integers (`5`)

In short, mutable data types can be modified, while immutable data types cannot be modified once created.

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

Ans - In Python, lists and tuples both store collections of data, but differ in mutability, performance and memory usage. Lists are mutable, allowing modifications, while tuples are immutable. Choosing between them depends on whether you need to modify the data or prioritize performance and memory efficiency.

4. Describe how dictionaries store data?
Ans- *Dictionaries* store data in key-value pairs, where:

1. *Keys* are unique identifiers (strings, integers, etc.)
2. *Values* are the data associated with each key

Dictionary Structure:
```
```

```
my_dict = {"name": "John", "age": 30}
```
In this example:

- `"name"` and `"age"` are keys
- `"John"` and `30` are values

How Dictionaries Store Data:
1. Keys are hashed to generate a unique index
2. Values are stored at the corresponding index

Benefits:
1. Fast lookups (O(1) average time complexity)
2. Efficient data retrieval and manipulation

Use Cases:
1. Data caching
2. Configuration files
3. Data serialization

Dictionaries are a powerful data structure in Python, allowing for efficient and flexible data storage and retrieval.

5. Why might you use a set instead of a list in Python?
Ans. You might use a *set* instead of a *list* in Python when:

1. *Uniqueness matters*: Sets automatically eliminate duplicates, ensuring all elements are unique.
2. *Fast membership testing*: Sets provide fast lookup times (O(1) average) to check if an element exists.
3. *Order doesn't matter*: Sets are unordered, which can be beneficial when order is irrelevant.

Use Cases:
1. Removing duplicates from a collection
2. Fast membership testing
3. Set operations (union, intersection, difference)

Example:
```
my_set = {1, 2, 3, 2, 1}  # {1, 2, 3}
print(2 in my_set)  # True
```
6. What is a string in Python, and how is it different from a list?
Ans- *String* in Python:

1. A sequence of characters (letters, numbers, symbols)
2. Immutable (cannot be changed after creation)

3. Defined using quotes (`"""` or `'"`)

Example:
```
my_string = "hello"
```

List in Python:

1. A collection of items (can be of different data types)
2. Mutable (can be changed after creation)
3. Defined using square brackets `[]`

Example:
```
my_list = ["h", "e", "l", "l", "o"]
```

Key differences:
1. *Immutability*: Strings are immutable, while lists are mutable.
2. *Data type*: Strings are sequences of characters, while lists can contain various data types.

Use Cases:
1. Strings for text manipulation and processing.
2. Lists for storing and manipulating collections of items.

7.How do tuples ensure data integrity in Python?
Ans. Tuples ensure data integrity in Python by being *immutable*, meaning their contents cannot be modified after creation. This prevents accidental changes or modifications, ensuring data consistency and reliability.

Benefits:
1. Data integrity
2. Code safety
3. Predictable behavior

Example:
my_tuple = (1, 2, 3)
my_tuple[0] = 10 # Error: 'tuple' object does not support item assignment.

8. What is a hash table, and how does it relate to dictionaries in Python?
Ans- Hash Table: A data structure that stores key-value pairs using a hash function to map keys to indices.

*Dictionaries in Python: Implemented using hash tables, allowing for:

1. Fast lookups (O(1) average)

2. Efficient insertion and deletion

How it works:
1. Keys are hashed to generate indices
2. Values are stored at corresponding indices

Benefits:
1. Fast data retrieval
2. Efficient data storage

In Python, dictionaries utilize hash tables to provide fast and efficient data access.

9. Can lists contain different data types in Python?
Ans- “Yes”, lists in Python can contain different data types, such as:

1. Integers
2. Strings
3. Floats
4. Booleans
5. Other lists
6. Dictionaries

Example:
my_list = [1, "hello", 3.14, True, [1, 2, 3], {"name": "John"}]

This flexibility allows lists to store and manipulate diverse data in Python.

10. Explain why strings are immutable in Python?
Ans- Strings are immutable in Python* because:

1. Security: Immutable strings prevent unintended changes, ensuring data integrity.
2. Performance: Immutable strings enable efficient caching and reuse.
3. Thread Safety: Immutable strings are safe to share between threads without risking concurrent modification.

Benefits:
1. Predictable behavior
2. Code reliability
3. Improved performance

This design choice ensures strings remain consistent and reliable throughout their lifecycle.

11. What advantages do dictionaries offer over lists for certain tasks?
Ans- Dictionaries offer several advantages over lists:

1. Fast lookups: Dictionaries provide fast lookups by key (O(1) average), while lists require linear search (O(n)).
2. Efficient data retrieval: Dictionaries allow direct access to values by key.

3. Flexible data structure: Dictionaries can store complex data with meaningful keys.

Use cases:
1. Data caching
2. Configuration files
3. Data serialization
4. Fast lookups and retrieval

Example:

```
person = {"name": "John", "age": 30}
print(person["name"])  # John
```

Dictionaries are ideal when you need fast lookups, efficient data retrieval, or flexible data structures.