Theoretical Questions and Answers

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

Data structures are ways to organize and store data efficiently to perform operations like access, addition, and deletion. They are important because they allow programs to manage data efficiently, optimize performance, and solve problems effectively.

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

- Mutable: Data types that can be changed after creation.
  Example: Lists -> my_list = [1, 2]; my_list.append(3) (List changes).
- Immutable: Data types that cannot be modified after creation.
  Example: Tuples -> my_tuple = (1, 2); my_tuple[0] = 3 (Throws an error).

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

- Lists: Mutable, dynamic, slower. Example: [1, 2, 3]
- Tuples: Immutable, static, faster. Example: (1, 2, 3)

4. Describe how dictionaries store data.

Dictionaries store data as key-value pairs in a hash table, where each key is hashed to a specific memory location for efficient retrieval.

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

Sets ensure no duplicate values and provide faster membership testing (in operation) compared to lists.

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

A string is an immutable sequence of characters. Unlike lists, strings cannot be modified directly, and each element of a string is a character, whereas list elements can be of any type.

7. How do tuples ensure data integrity in Python?

Tuples are immutable, meaning their elements cannot be changed. This ensures data integrity, making tuples useful for fixed or constant data.

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 to allow O(1) time complexity for accessing elements.

9. Can lists contain different data types in Python?

Yes, lists can contain elements of mixed data types, such as integers, strings, and even other lists.

Example: [1, "hello", 3.14, [2, 3]].

10. Explain why strings are immutable in Python.

Strings are immutable to ensure safety, efficiency, and hashability (so they can be used as keys in dictionaries).

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

Dictionaries provide faster lookups (O(1) time complexity) for key-based access compared to lists, which require searching through elements (O(n) time complexity).

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

When storing fixed data that should not change, such as coordinates (x, y) or a database record.

13. How do sets handle duplicate values in Python?

Sets automatically discard duplicate values when elements are added.

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

- Lists: Checks for presence of a value.
- Dictionaries: Checks for the presence of a key.

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 contains other dictionaries as values.

Example: {"student1": {"name": "John", "age": 20}}.

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

Accessing an element in a dictionary has an average time complexity of O(1).

18. In what situations are lists preferred over dictionaries?

When the order of elements matters or when only values (not keys) are needed.

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

Dictionaries prior to Python 3.7 were unordered, meaning the order of elements was not guaranteed. However, in Python 3.7+, dictionaries maintain insertion order, but retrieval is still based on keys, not index positions.

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

- List: Access elements using an index. Example: my_list[0].
- Dictionary: Access elements using keys. Example: my_dict["key"].

Practical Questions and Code Solutions

1. Write a code to create a string with your name and print it.

```python
name = "John Doe"
print(name)
```

2. Write a code to find the length of the string "Hello World".

```python
string = "Hello World"
print(len(string))
```

3. Write a code to slice the first 3 characters from the string "Python Programming".

```python
string = "Python Programming"
print(string[:3])
```

4. Write a code to convert the string "hello" to uppercase.

```python
string = "hello"
print(string.upper())
```

5. Write a code to replace the word "apple" with "orange" in the string "I like apple".

```python
string = "I like apple"
print(string.replace("apple", "orange"))
```

6. Write a code to create a list with numbers 1 to 5 and print it.

```python
numbers = [1, 2, 3, 4, 5]
print(numbers)
```

7. Write a code to append the number 10 to the list [1, 2, 3, 4].

```python
numbers = [1, 2, 3, 4]
numbers.append(10)
print(numbers)
```

8. Write a code to remove the number 3 from the list [1, 2, 3, 4, 5].

```python
numbers = [1, 2, 3, 4, 5]
numbers.remove(3)
print(numbers)
```

9. Write a code to access the second element in the list ['a', 'b', 'c', 'd'].

```python
letters = ['a', 'b', 'c', 'd']
print(letters[1])
```

10. Write a code to reverse the list [10, 20, 30, 40, 50].

```python
numbers = [10, 20, 30, 40, 50]
numbers.reverse()
print(numbers)
```