Assignment 3: Data Structure

Q1. What are data structures, and why are they important?
Ans: Data structures are organized ways to store, manage, and access data efficiently. They are important because they optimize data processing, enhance performance, and provide a structured approach to data management, especially in complex applications.

Q2. Explain the difference between mutable and immutable data types with examples.
Ans: Mutable data types can be changed after creation (e.g., lists), while immutable data types cannot (e.g., tuples). For instance, a list can have its elements altered, whereas a tuple's contents remain constant.

Q3. What are the main differences between lists and tuples in Python?
Ans: Lists are mutable, allowing changes after creation, while tuples are immutable. Lists consume more memory and have slower access times compared to tuples, but they offer more flexibility for modification.

Q4. Describe how dictionaries store data.
Ans: Dictionaries store data as key-value pairs, where each unique key maps to a specific value. They use hash functions for quick lookups, making data retrieval efficient.

Q5. Why might you use a set instead of a list in Python?
Ans: Sets are used when the uniqueness of elements is important, as they automatically eliminate duplicates. They are also faster for membership testing compared to lists.

Q6. What is a string in Python, and how is it different from a list?
Ans: A string is an immutable sequence of characters, whereas a list is a mutable sequence of elements. Strings cannot be changed after creation, while lists can be modified.

Q7. How do tuples ensure data integrity in Python?
Ans: Tuples are immutable, meaning once created, their elements cannot be changed. This immutability preserves data integrity, especially when consistent, unaltered data is needed.

Q8. What is a hash table, and how does it relate to dictionaries in Python?
Ans: A hash table stores data as key-value pairs, where a hash function computes an index for quick data retrieval. Python dictionaries internally use hash tables for efficient lookups.

Q9. Can lists contain different data types in Python?
Ans: Yes, lists in Python can contain elements of different data types, including integers, strings, and even other lists or objects.

Q10. Explain why strings are immutable in Python.
Ans: Strings are immutable to ensure data consistency and security, especially when used as keys in dictionaries. This immutability also allows for better memory optimization through string interning.

Q11. What advantages do dictionaries offer over lists for certain tasks?
Ans: Dictionaries offer faster lookups (O(1) average time complexity) for key-based data retrieval compared to lists, which require linear searches (O(n)). They are ideal for situations where data is associated with unique identifiers.

Q12. How do sets handle duplicate values in Python?
Ans: Sets automatically eliminate duplicate values, ensuring that each element is unique. Adding a duplicate to a set has no effect, maintaining a collection of distinct elements.

Q13. Describe a scenario where using a tuple would be preferable over a list.
Ans: Tuples are preferable when data should remain constant, such as storing coordinates (x, y) that shouldn't change or returning fixed sets of values from functions.

Q14. How does the “in” keyword work differently for lists and dictionaries?
Ans: In lists, the "in" keyword checks for the presence of a value, while in dictionaries, it checks for the presence of a key, not the value itself.

Q15. Can you modify the elements of a tuple? Explain why or why not.
Ans: No, tuples are immutable, so their elements cannot be changed after creation. This immutability ensures that once set, the data remains consistent and unaltered.

Q16. What is a nested dictionary, and give an example of its use case?
Ans: A nested dictionary is a dictionary within another dictionary. It is useful for representing hierarchical data, such as a database of employees where each department has its own sub-dictionary of members.

Q17. Describe the time complexity of accessing elements in a dictionary.
Ans: The average time complexity for accessing an element in a dictionary is O(1) due to its hash table implementation, but it can degrade to O(n) in cases of hash collisions.

Q18. In what situations are lists preferred over dictionaries?
Ans: Lists are preferred when the order of elements is important, or when performing sequential operations, such as iterating through a collection or performing operations on indexed elements.

Q19. Why are dictionaries considered unordered, and how does that affect data retrieval?
Ans: Dictionaries in Python are unordered because they are based on hash tables. This means that the order of key-value pairs is not guaranteed, which affects scenarios where maintaining sequence is necessary.

Q20. Explain the difference between a list and a dictionary in terms of data retrieval.
Ans: Lists retrieve elements by index, which is efficient for ordered data. Dictionaries retrieve elements by key, providing faster access for key-value mappings, but without preserving element order.