Introduction to Python Programming
Data Structures Quiz – Answer Key

1. **(C) Tuple.** Tuples are immutable in Python, meaning their elements cannot be changed after creation. Lists, dictionaries, and sets are all mutable.

2. **(C) 3.** When duplicate keys exist in a dictionary literal, the last value overwrites earlier ones. The key `'a'` is assigned 3 in the final assignment.

3. **(C) `append()`.** The `append()` method adds a single element to the end of a list. `insert()` adds at a specific index, `extend()` adds multiple elements, and `add()` is for sets.

4. **(B) 3.** Sets automatically remove duplicate elements. The set `{1, 2, 3, 2, 1}` reduces to `{1, 2, 3}`, which has length 3.

5. **(B) `d = {}`.** Empty curly braces create an empty dictionary. `[]` creates a list, `set()` creates an empty set, and `dict[]` is invalid syntax.

6. **True.** Python lists are heterogeneous and can store elements of any data type, including mixed types like `[1, "hello", 3.14, True]`.

7. **True.** Dictionary keys must be unique and hashable. If a duplicate key is assigned, the new value overwrites the old. Values have no uniqueness constraint.

8. **False.** Sets are unordered collections, so `pop()` removes and returns an arbitrary element, not necessarily the last one added. The removal order is not guaranteed.

9. **Mutability:** Lists are mutable (can be modified after creation), while tuples are immutable (cannot be changed once created).

   Syntax: Lists use square brackets `[1, 2, 3]`, tuples use parentheses `(1, 2, 3)`.

   Use cases: Lists are ideal for collections that need modification (e.g., maintaining a shopping cart). Tuples are suitable for fixed data (e.g., coordinates, database records) and can serve as dictionary keys.

   Performance: Tuples have slightly faster iteration and lower memory overhead due to their immutability.

   Example:

   ```
   # List modification (allowed)
   my_list = [1, 2, 3]
   my_list[0] = 10  # Works

   # Tuple modification (error)
   my_tuple = (1, 2, 3)
   my_tuple[0] = 10  # TypeError
   ```

10. **Hash table mechanism:** Python dictionaries use hash tables for O(1) average-case lookup. When a key is added, Python computes its hash value to determine the storage location.

    Valid keys: Keys must be hashable (immutable types like strings, numbers, tuples). Lists and sets cannot be keys because they are mutable.

    Operations:

    ```
    # Creating and adding
    student = {'name': 'Alice', 'age': 20}
    student['grade'] = 'A'  # Add new key-value

    # Updating
    student['age'] = 21  # Update existing

    # Deleting
    del student['grade']  # Remove key-value pair
    removed = student.pop('age')  # Remove and return value
    ```

    Key uniqueness: Duplicate keys overwrite previous values; `{'a': 1, 'a': 2}` results in `{'a': 2}`.