Data Types and Structures Assignment

Anoop Verma (anoopmuz@gmail.com)

Batch - Data Sc. with Gen AI (May 2025)

Q (1) What are data structures, and why are they important?

Ans:- Data structures are specialized formats for organizing, storing, and managing data in a computer so that it can be accessed and modified efficiently. They define the relationship between data elements and the operations that can be performed on them.

Common Types of Data Structures

- Integers (int)
- Floating-point numbers (float)
- Characters (char)/ Strings
- Booleans (bool)
- Arrays
- Linked Lists
- Stacks (LIFO)
- Queues (FIFO)
- Trees (e.g., Binary Trees, Heaps)
- Graphs
- Hash-based: Fast key-value lookups.
- Hash Tables (Dictionaries in Python)
- Sets: Unordered collections of unique elements.

Importantance of Data Structures

- Efficiency:
- Memory Management
- Real-World Problem Solving
- Databases use B-trees for indexing.
- Networking relies on queues for packet scheduling.
- AI/ML uses graphs for neural networks and decision trees.
- Code Readability & Maintainability
- Well-chosen data structures make code more logical and easier to debug.

Q (2) Explain the difference between mutable and immutable data types with examples?

Ans:-

Mutable

- Can be modified after creation
- Same object can change
- e.g. List, Dictionary, Set

Immutable

- Cannot be modified after creation
- A new object is created after modification
- e.g. Integer, Strings, Tuple, Frozensets

Mutable objects save memory, but immutable objects are faster to access.

Mutable objects are paased by reference to a fumction, while immutable objects are passed by value to a function.

Q (3) What are the main differences between lists and tuples in Python?

Ans:-

- List is mutable while Tuple is iimutable.
- List is created using [ ], while Tuple is created using ( ).
- List runs slower, while Tuple runs fatser.
- List consumes more memory, while Tuple is more memory-efficient.
- List has more built-in methodsm while Tuple has less methods.

Q (4) Describe how dictionaries store data.

Ans:- Dictionaries (dict) in Python are hash tables that store data as key-value pairs.

- Keys must be immutable (e.g., int, str, tuple).
- Values can be any type (mutable or immutable).

Q (5) Why might you use a set instead of a list in Python?

Ans:- Sets (set) and lists (list) serve different purposes in Python.

- When no duplicates are needed, a set is used, as it automatically ermoves the duplicates.
- Mathematical set operations like union, intersection, join, differences are available with sets, while list requires manual loop to be written for these kind of operations.

Q (6) What is a string in Python, and how is it different from a list?

Ans:- In Python, a string is a sequence of characters enclosed in single (' '), double (" "), or triple (''' ''' or """ """) quotes. Strings are immutable, meaning their contents cannot be changed after creation.

- List is mutable and is created using square brackets [ ], string is immutable.
- List can hold any type of data, while string can hold only character type data.
- Strings have string-specific funtions, like split(), strip() etc., while lists have list-specific functions, like append(), pop() etc.
- Strings are used for text manupulation while lists are used to store different types of data.

Q (7) How do tuples ensure data integrity in Python?

Ans:- Tuples ensure data integrity by being immutable, meaning once created, their elements cannot be modified, added, or removed. This immutability provides safety to data where data should remain constant.

- Prevents accidental modifications
- Used as dictionary keys
- Helps in returning multiple values from a function
- Tuples are faster than lists, because it is static in nature

Q (8) What is a hash table, and how does it relate to dictionaries in Python?

Ans:- A hash table (or hash map) is a data structure that stores key-value pairs by using a hash function to compute an index (hash) where the value should be placed.

It allows for better time complexity for addition, deletion and modification, and is able to handle situations where two keys hash to the ame index.

Pyhton dictionaries are built using hash-tables. Dictionaries have fast access to its data due to hash (index) and the uniqueness of the keys.

Q (9) Can lists contain different data types in Python?

Ans:- Yes, Python lists can contain different data types.

Q (10) Explain why strings are immutable in Python.

Ans:- Python strings are immutable by its design. Due to its immutable nature, strings have many advantages like,

- Security & Data integrity
- Immutable strings can be used as keys in dictionaries
- Memory effeciency
- Thread safety
- String operations return new strings, instead of manipulating the original
- Immutability ensures functions cannot unexpectedly modify string arguments

Q (11) What advantages do dictionaries offer over lists for certain tasks?

Ans:- Dictionaries (dict) and lists (list) serve different purposes, but dictionaries are more suitable in some specific situations due to their hash table-based implementation, like

- Fast search using keys
- No duplicate keys
- Efficient membership testing
- Data stored in a better way
- Memory efficiency

List is more useful in situations where the data is to be manipulated using sequential methods, ordering of data is important.

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

Ans:- Tuples (tuple) and lists (list) are both sequence types in Python, but tuples have unique advantages in certain scenarios due to their immutability, memory efficiency, and hashability.

The following are the situations where tuples are more useful:

- Immutable Data (Safety & Integrity)
- Hashability (Keys in Dictionaries or Elements in Sets)

- Memory Efficiency & Performance (Tuples are smaller and faster to create than lists because they're stored more compactly)
- Function Arguments & Return Values (Tuples are often used to return multiple values from functions or as fixed-format arguments)
- Thread Safety (Since tuples are immutable, they're inherently thread-safe)

Q (13) How do sets handle duplicate values in Python?

Ans:- In Python, a set is an unordered collection of unique elements. When we add duplicate values to a set, they are automatically removed because sets are designed to enforce uniqueness.

Q (14) How does the "in" keyword work differently for lists and dictionaries.

Ans:- The "in" keyword in Python checks for membership, but its behavior differs significantly between lists and dictionaries due to their underlying data structures.

in with Lists (list)

- Checks for the presence of a value (not a key).
- Time Complexity: O(n) (linear search) - slower for large lists because it is sequential in nature.

in with Dictionaries (dict)

- Checks for the presence of a key (not a value).
- Time Complexity: O(1) (constant-time lookup) - fast due to hashing.

Q (15) Can you modify the elements of a tuple? Explain why or why not.

Ans:- No, tuples are immutable—you cannot modify their elements after creation.

If we try to change a value in Tuple, it gives a "TypeError".

Q (16) What is a nested dictionary, and give an example of its use case?

Ans:- A nested dictionary is a dictionary that contains other dictionaries as values. This allows us to store hierarchical or structured data in a key-value format, where values themselves can be dictionaries (or even lists, tuples, etc.).

Example where it can be used:

- Representing complex records (e.g., e-commerce products with categories).
- Configuration Files (Multi-level settings (e.g., app configurations with sections).
- School/University Data (Organizing student information with nested structures.)

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

Ans:- In Python, dictionaries (dict) are implemented using hash tables, which provide highly efficient element access.

Time complexity for common operations are as follows:

- Accessing a Value by Key - Average Case: O(1) and Worst Case: O(n)
- Checking if a Key Exists (in Operator) - Average Case: O(1) and Worst Case: O(n)
- Accessing Keys, Values, or Items - Iteration is O(n)

Q (18) In what situations are lists preferred over dictionaries?

Ans:- While dictionaries (dict) are used for fast key-based searches, lists (list) are used in scenarios requiring ordered, index-based, or sequential data operations.

Q (19) Why are dictionaries considered unordered, and how does that affect data retrieval.

Ans:- Dictionaries (dict) were officially unordered, meaning:

- Key-value pairs could appear in any order when iterating or printing.
- The order was not guaranteed to match insertion order due to hash table implementation.

It affects the data retrieval in following ways:-

- Iteration Order Was Unpredictable
- Inconsistent dict.keys(), dict.values(), dict.items()
- Reproducibility Issues

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

Ans:- The primary differences between lists and dictionaries in Python in terms of data retrieval are:

- How data is accessed (index vs. key).
- Speed of retrieval (time complexity).
- Use cases (ordered sequences vs. key-value lookups).

```
# (1) Write a code to create a string with your name and print it.

name = "Anoop Verma"
print(name)
```

```
Anoop Verma
```

```
# (2) Write a code to find the length of the string "Hello World".

str = "Hello World"
length = len(str)
print("The length of the string is:", length)
```

```
The length of the string is: 11
```

```
# (3) Write a code to slice the first 3 characters from the string "Python
# Programming".

str = "Python Programming"
first_three_char = str[:3]  # Slice from start (index 0) up to (but not including) index 3
print("First 3 characters:", first_three_char)
```

```
First 3 characters: Pyt
```

```
# (4) Write a code to convert the string "hello" to uppercase.

str = "hello"
ucase_str = str.upper()
print(ucase_str)
```

```
HELLO
```

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

str = "I like apple"
new_str = str.replace("apple", "orange")
print(new_str)
```

```
I like orange
```

```
# (6) Write a code to create a list with numbers 1 to 5 and print it.

num = [10, 20, 30, 40, 50]
print(num)
```

```
[10, 20, 30, 40, 50]
```

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

my_list = [1, 2, 3, 4]
my_list.append(10)  # Adds 10 to the end of the list
print(my_list)
```

```
[1, 2, 3, 4, 10]
```

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

my_list = [1, 2, 3, 4, 5]
my_list.remove(3)  # Removes the first occurrence of 3
print(my_list)
```

```
[1, 2, 4, 5]
```

```
# (9) Write a code to access the second element in the list ['a', 'b', 'c', 'd']

my_list = ['a', 'b', 'c', 'd']
sec_ele = my_list[1]  # Index 1 refers to the second element
print(sec_ele)
```

```
b
```

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

my_list = [10, 20, 30, 40, 50]
my_list.reverse()  # Reverses the list in-place
print(my_list)
```

```
[50, 40, 30, 20, 10]
```

```
# (11) Write a code to create a tuple with the elements 100, 200, 300 and print
# it.

my_tuple = (100, 200, 300)  # Parentheses define a tuple
print(my_tuple)
```

```
(100, 200, 300)
```

```
# (12) Write a code to access the second-to-last element of the tuple ('red',
# 'green', 'blue', 'yellow').

color = ('red', 'green', 'blue', 'yellow')
second_last = color[-2]  # -2 refers to the second-to-last element
print(second_last)
```

```
blue
```

```
# (13) Write a code to find the minimum number in the tuple (10, 20, 5, 15).

num = (10, 20, 5, 15)
min_num = min(num)  # Finds the smallest element
print("The minimum number is:", min_num)
```

```
The minimum number is: 5
```

```
# (14) Write a code to find the index of the element "cat" in the tuple ('dog',
# 'cat', 'rabbit').

animals = ('dog', 'cat', 'rabbit')
index = animals.index('cat')  # Returns the index of 'cat'
print("Index of 'cat':", index)
```

```
Index of 'cat': 1
```

```
# (15) Write a code to create a tuple containing three different fruits and
# check if "kiwi" is in it.

fruits = ('apple', 'banana', 'orange')

if 'kiwi' in fruits:
    print("Yes, 'kiwi' is in the tuple!")
else:
    print("No, 'kiwi' is not in the tuple.")
```

```
No, 'kiwi' is not in the tuple.
```

```
# (16) Write a code to create a set with the elements 'a', 'b', 'c' and print
# it.

my_set = {'a', 'b', 'c'}
print(my_set)
```

```
{'b', 'a', 'c'}
```

```
# (17)  Write a code to clear all elements from the set {1, 2, 3, 4, 5}.

my_set = {1, 2, 3, 4, 5}
my_set.clear()
print(my_set)
```

```
set()
```

```
# (18)  Write a code to remove the element 4 from the set {1, 2, 3, 4}.

my_set = {1, 2, 3, 4}
my_set.remove(4)
print(my_set)
```

```
{1, 2, 3}
```

```
# (19) Write a code to find the union of two sets {1, 2, 3} and {3, 4, 5}.

set_a = {1, 2, 3}
set_b = {3, 4, 5}
union_set = set_a.union(set_b)  # Combines both sets, removes duplicates
print(union_set)
```

```
{1, 2, 3, 4, 5}
```

```
# (20) Write a code to find the intersection of two sets {1, 2, 3} and
# {2, 3, 4}.

set_a = {1, 2, 3}
set_b = {2, 3, 4}
intersection_set = set_a.intersection(set_b)  # Finds common elements
print(intersection_set)
```

```
{2, 3}
```

```
# (21) Write a code to create a dictionary with the keys "name", "age", and
# "city", and print it.

person = {
    "name": "Anoop",
    "age": 59,
    "city": "Muzaffarpur"
}
print(person)
```

```
{'name': 'Anoop', 'age': 59, 'city': 'Muzaffarpur'}
```

```
# (22) Write a code to add a new key-value pair "country": "USA" to the
# dictionary {'name': 'John', 'age': 25}.

person = {'name': 'John', 'age': 25}
person['country'] = 'USA'  # Adds the new key-value pair
print(person)
```

```
{'name': 'John', 'age': 25, 'country': 'USA'}
```

```
# (23) Write a code to access the value associated with the key "name" in the
# dictionary {'name': 'Alice', 'age': 30}.

person = {'name': 'Alice', 'age': 30}
name_value = person['name']
print(name_value)
```

```
Alice
```

```
# (24) Write a code to remove the key "age" from the dictionary
# {'name': 'Bob', 'age': 22, 'city': 'New York'}.

person = {'name': 'Bob', 'age': 22, 'city': 'New York'}
del person['age']
print(person)
```

```
{'name': 'Bob', 'city': 'New York'}
```

```
# (25) Write a code to check if the key "city" exists in the dictionary
# {'name': 'Alice', 'city': 'Paris'}.

person = {'name': 'Alice', 'city': 'Paris'}
if 'city' in person:
    print("Key 'city' exists!")
else:
    print("Key 'city' does not exist.")
```

```
Key 'city' exists!
```

```
# (26) Write a code to create a list, a tuple, and a dictionary, and print them
# all.

# Create a list
my_list = [10, 20, 30, 40]

# Create a tuple
my_tuple = ('apple', 'banana', 'cherry')

# Create a dictionary
my_dict = {'name': 'Anoop', 'age': 59, 'city': 'Muzaffarpur'}

# Print all three
print("List:", my_list)
print("Tuple:", my_tuple)
print("Dictionary:", my_dict)
```

```
List: [10, 20, 30, 40]
Tuple: ('apple', 'banana', 'cherry')
Dictionary: {'name': 'Anoop', 'age': 59, 'city': 'Muzaffarpur'}
```

```
# (27) Write a code to create a list of 5 random numbers between 1 and 100,
# sort it in ascending order, and print the result.(replaced)

import random

# Generate 5 random numbers between 1 and 100
random_num = [random.randint(1, 100) for _ in range(5)]

# Sort the list in ascending order
random_num.sort()

# Print the result
print("Sorted random numbers:", random_num)
```

```
Sorted random numbers: [1, 42, 43, 44, 95]
```

```
# (28) Write a code to create a list with strings and print the element at the
# third index.

my_list = ["apple", "banana", "cherry", "date", "elderberry"]

# Print the element at the third index
print("The element at the third index is:", my_list[3])
```

```
The element at the third index is: date
```

```
# (29)  Write a code to combine two dictionaries into one and print the result.

dict1 = {'a': 1, 'b': 2}
dict2 = {'c': 3, 'd': 4}

combined_dict = dict1.copy()  # Create a copy of dict1 to preserve the original
combined_dict.update(dict2)   # Update with dict2's key-value pairs

print("Combined dictionary (using update()):", combined_dict)
```

```
Combined dictionary (using update()): {'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

```
# (30) Write a code to convert a list of strings into a set.

# Original list of strings
string_list = ["apple", "banana", "cherry", "apple", "banana"]

string_set = set(string_list)

# Print the results
print("Original list:", string_list)
print("Converted set:", string_set)
```

```
Original list: ['apple', 'banana', 'cherry', 'apple', 'banana']
Converted set: {'apple', 'banana', 'cherry'}
```