# Theory Questions of DATA STRUCTURE

---

## Question 1: What are data structures, and why are they important?

Data structures are ways to store and organize data in a computer efficiently so that operations like access, insertion, deletion, and search can be performed quickly. They are essential because they help manage data in an optimized way, making algorithms run faster and use memory efficiently.

### Examples of data structures:

1. **Lists** – Dynamic arrays that allow fast insertions and deletions.
2. **Tuples** – Immutable sequences that store multiple values.
3. **Dictionaries** – Key-value pairs that allow fast lookups.
4. **Sets** – Collections of unique elements that prevent duplicates.
5. **Stacks & Queues** – Used in algorithms like recursion and scheduling.
6. **Trees & Graphs** – Used in complex structures like file systems and social networks.

### Importance of data structures:

- They **increase efficiency** in operations like sorting and searching.
- They **reduce memory usage** by organizing data optimally.
- They **enable complex problem-solving**, like AI and databases.

---

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

A **mutable** object can be changed after it is created, whereas an **immutable** object cannot be changed once created.

### Mutable Objects (Can be changed):

- Lists
- Dictionaries
- Sets

### Immutable Objects (Cannot be changed):

- Strings
- Tuples
- Integers & Floats

### Example of a mutable object (List):

```
my_list = [1, 2, 3]
my_list.append(4)  # Modifies the list
print(my_list)  # Output: [1, 2, 3, 4]
```

### Example of an immutable object (Tuple):

```
my_tuple = (1, 2, 3)
my_tuple[1] = 5  #     Raises TypeError because tuples cannot be modified
```

## Question 11: What advantages do dictionaries offer over lists for certain tasks?

Dictionaries are **more efficient** than lists in certain situations:

### 1. Faster Lookups (`O(1)` vs. `O(n)`)

- **Dictionaries** use a **hash table** to store key-value pairs, so retrieving a value by key takes constant time (`O(1)`).
- **Lists** require scanning (`O(n)`) to find an element.

### Example:

```
# Using a dictionary
student_grades = {"Alice": "A", "Bob": "B"}
print(student_grades["Alice"])  # O(1)

# Using a list (inefficient for lookup)
students = [("Alice", "A"), ("Bob", "B")]
for name, grade in students:
    if name == "Alice":
        print(grade)  # O(n)
```

### 2. Meaningful Data Storage

- Dictionary **keys** act as labels, making data retrieval intuitive.

### 3. No Duplicates in Keys

- Unlike lists, dictionaries prevent duplicate keys.

---

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

Tuples are preferable when **data should not be modified**.

### Scenario 1: Storing Fixed Coordinates

```
location = (40.7128, 74.0060)  # New York City latitude & longitude
```

- Coordinates should **not change**, so a tuple is ideal.

### Scenario 2: Dictionary Keys

Tuples can be used as **keys in dictionaries**, while lists cannot.

```
locations = {
    (40.7128, 74.0060): "New York",
    (34.0522, -118.2437): "Los Angeles"
}
```

- Lists **cannot** be used as dictionary keys since they are mutable.

## Practical Questions

**QUESTION 1: Write a code to create a string with your name and print it**

```
# Creating a string with my name
name = "Bishwajeet Dutta"

# Printing the name
print(name)
```

```
Bishwajeet Dutta
```

**QUESTION 2: Write a code to find the length of the string `"Hello World"`.**

```
# Defining the string
text = "Hello World"

# Finding and printing the length of the string
print(len(text))
```

```
11
```

**QUESTION 3: Write a code to slice the first 3 characters from the string `"Python Programming"`.**

```
# Defining the string
text = "Python Programming"

# Slicing the first 3 characters
sliced_text = text[:3]

# Printing the result
print(sliced_text)
```

```
Pyt
```

**QUESTION 4: Write a code to convert the string `"hello"` to uppercase.**

```
# Defining the string
text = "hello"

# Converting to uppercase
uppercase_text = text.upper()

# Printing the result
print(uppercase_text)
```

```
HELLO
```

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

```
# Defining the string
text = "I like apple"

# Replacing "apple" with "orange"
modified_text = text.replace("apple", "orange")

# Printing the result
print(modified_text)
```

```
I like orange
```

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

```
# Creating a list with numbers 1 to 5
numbers = [1, 2, 3, 4, 5]

# Printing the list
print(numbers)
```

```
[1, 2, 3, 4, 5]
```

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

```
# Defining the list
numbers = [1, 2, 3, 4]

# Appending the number 10
numbers.append(10)

# Printing the updated list
print(numbers)
```

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

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

```
# Defining the list
numbers = [1, 2, 3, 4, 5]

# Removing the number 3
numbers.remove(3)

# Printing the updated list
print(numbers)
```

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

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

```
# Defining the list
letters = ['a', 'b', 'c', 'd']

# Accessing the second element
second_element = letters[1]

# Printing the result
print(second_element)
```

```
b
```

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

```
# Defining the list
numbers = [10, 20, 30, 40, 50]

# Reversing the list
numbers.reverse()

# Printing the reversed list
print(numbers)
```

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

**QUESTION 11: Write a code to create a tuple with the elements `100, 200, 300` and print it.**

```
# Creating a tuple
numbers_tuple = (100, 200, 300)

# Printing the tuple
print(numbers_tuple)
```

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

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

```
# Defining the tuple
colors = ('red', 'green', 'blue', 'yellow')

# Accessing the second-to-last element
second_last_element = colors[-2]

# Printing the result
print(second_last_element)
```

```
blue
```

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

```
# Defining the tuple
numbers = (10, 20, 5, 15)

# Finding the minimum number
min_number = min(numbers)

# Printing the result
print(min_number)
```

```
5
```

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

```
# Defining the tuple
animals = ('dog', 'cat', 'rabbit')

# Finding the index of 'cat'
cat_index = animals.index('cat')

# Printing the result
print(cat_index)
```

```
1
```

**QUESTION 15: Write a code to create a tuple containing three different fruits and check if `"kiwi"` is in it.**

```
# Defining the tuple with three different fruits
fruits = ("apple", "banana", "mango")

# Checking if "kiwi" is in the tuple
is_kiwi_present = "kiwi" in fruits

# Printing the result
print(is_kiwi_present)
```

```
False
```

**QUESTION 16: Write a code to create a set with the elements `'a', 'b', 'c'` and print it.**

```
# Creating a set with elements 'a', 'b', 'c'
my_set = {'a', 'b', 'c'}

# Printing the set
print(my_set)
```

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

**QUESTION 17: Write a code to clear all elements from the set `{1, 2, 3, 4, 5}`.**

```
# Creating a set with elements {1, 2, 3, 4, 5}
my_set = {1, 2, 3, 4, 5}

# Clearing all elements from the set
my_set.clear()

# Printing the set after clearing
print(my_set)
```

```
set()
```

**QUESTION 18: Write a code to remove the element `4` from the set `{1, 2, 3, 4}`.**

```
# Creating a set with elements {1, 2, 3, 4}
my_set = {1, 2, 3, 4}

# Removing the element 4 from the set
my_set.remove(4)

# Printing the set after removal
print(my_set)
```

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

**QUESTION 19: Write a code to find the union of two sets `{1, 2, 3}` and `{3, 4, 5}`.**

```
# Creating two sets
set1 = {1, 2, 3}
set2 = {3, 4, 5}

# Finding the union of the sets
union_set = set1.union(set2)

# Printing the result
print(union_set)
```

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

**QUESTION 20: Write a code to find the intersection of two sets `{1, 2, 3}` and `{2, 3, 4}`.**

```
# Creating two sets
set1 = {1, 2, 3}
set2 = {2, 3, 4}

# Finding the intersection of the sets
intersection_set = set1.intersection(set2)

# Printing the result
print(intersection_set)
```

```
{2, 3}
```

**QUESTION 21: Write a code to create a dictionary with the keys `"name"`, `"age"`, and `"city"`, and print it.**

```
# Creating a dictionary with keys "name", "age", and "city"
person_info = {
    "name": "John",
    "age": 25,
    "city": "New York"
}

# Printing the dictionary
print(person_info)
```

```
{'name': 'John', 'age': 25, 'city': 'New York'}
```

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

```
# Define the dictionary
person = {'name': 'John', 'age': 25}

# Add a new key-value pair
person['country'] = 'USA'

# Print the updated dictionary
print(person)
```

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

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

```
# Define the dictionary
person = {'name': 'Alice', 'age': 30}

# Access the value associated with the key "name"
print(person['name'])
```

```
Alice
```

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

```
# Creating the dictionary
person = {'name': 'Bob', 'age': 22, 'city': 'New York'}

# Removing the key "age"
del person['age']

# Printing the updated dictionary
print(person)
```

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

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

```
# Define the dictionary
my_dict = {'name': 'Alice', 'city': 'Paris'}

# Check if the key "city" exists
if 'city' in my_dict:
    print("Key 'city' exists in the dictionary.")
else:
    print("Key 'city' does not exist in the dictionary.")
```

```
Key 'city' exists in the dictionary.
```

**QUESTION 26: Write a code to create a list, a tuple, and a dictionary, and print them all.**

```
# Creating a list, a tuple, and a dictionary
my_list = [1, 2, 3, 4, 5]
my_tuple = (10, 20, 30, 40, 50)
my_dict = {"name": "Alice", "age": 25, "city": "New York"}

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

```
List: [1, 2, 3, 4, 5]
Tuple: (10, 20, 30, 40, 50)
Dictionary: {'name': 'Alice', 'age': 25, 'city': 'New York'}
```

**QUESTION 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.**

```python
# Importing the random module
import random

# Generating a list of 5 random numbers between 1 and 100
random_numbers = [random.randint(1, 100) for _ in range(5)]

# Sorting the list in ascending order
random_numbers.sort()

# Printing the sorted list
print("Sorted random numbers:", random_numbers)
```

```
Sorted random numbers: [15, 34, 39, 66, 73]
```

**QUESTION 28: Write a code to create a list with strings and print the element at the third index.**

```python
# Creating a list with strings
string_list = ["apple", "banana", "cherry", "date", "elderberry"]

# Printing the element at the third index
print(string_list[3])
```

```
date
```

**QUESTION 29: Write a code to combine two dictionaries into one and print the result.**

```python
# Define two dictionaries
dict1 = {'a': 1, 'b': 2}
dict2 = {'c': 3, 'd': 4}

# Merge dictionaries
combined_dict = {**dict1, **dict2}

# Print result
print(combined_dict)
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

**QUESTION 30: Write a code to convert a list of strings into a set.**

```python
# Define a list of strings
string_list = ["apple", "banana", "cherry", "apple", "banana"]

# Convert the list into a set
string_set = set(string_list)

# Print the result
print(string_set)
```

```
{'cherry', 'apple', 'banana'}
```