Data Types and Structures Questions

1. What are data structures, and why are they important.?
   - **Arrays**: Store elements in a fixed-size sequential list.
   - **Linked Lists**: Elements (nodes) linked using pointers.
   - **Stacks**: Last-In-First-Out (LIFO) structure.
   - **Queues**: First-In-First-Out (FIFO) structure.
   - **Trees**: Hierarchical structure (e.g., binary trees, BST).
   - **Graphs**: Set of nodes (vertices) connected by edges.
   - **Hash Tables (or Hash Maps)**: Key-value pairs for fast lookup.

Why Data Structures Are Important:

- **Efficiency**: They allow efficient use of memory and processing power, especially for large datasets.
  - Example: Searching in a binary search tree is faster than in a list.
- **Optimization**: The right data structure reduces time and space complexity.
  - Example: Using a hash table for lookups vs. linear search.
- **Real-World Problem Solving**: Many applications (like databases, search engines, and social networks) rely on efficient data structuring.
- **Algorithm Performance**: Algorithms are often designed for specific data structures. Understanding one helps understand the other.
- **Code Maintainability**: Proper use of data structures leads to clearer, more maintainable, and scalable code.

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

- **Mutable Data Types**: Can be **changed after creation**.
  Examples: list, dict, set
  Example:

```
my list = [1, 2]

my_list.append(3)  # Now: [1, 2, 3]
```

- **Immutable Data Types**: **Cannot be changed** after creation.
  Examples: int, float, str, tuple
  Example:

```
my_str = "hello"

my_str += " world"  # New string created: "hello world"
```

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

Feature	**List**	**Tuple**
Mutability	✅ Mutable (can be changed)	❌ Immutable (cannot be changed)
Syntax	[1, 2, 3]	(1, 2, 3)
Performance	Slower	Faster (more memory efficient)
Use Case	When data may change	When data should stay constant
Methods	More built-in methods	Fewer built-in methods

Example:-

```
my_list = [1, 2, 3]
my_tuple = (1, 2, 3)
```

- Use **lists** when you need to modify data; use **tuples** for fixed, unchangeable data.

4.Describe how dictionaries store data.

- Dictionaries store data as **key-value pairs**.
- Internally, they use a **hash table** to map each key to its value.
- Each **key** is hashed to a unique index where the **value** is stored.
- Keys must be **unique** and **immutable** (e.g., strings, numbers, tuples).
- Values can be of **any data type** and **can be duplicated**.

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

- **Need to store unique items** (sets automatically remove duplicates).
- **Want faster membership checks** (sets use a hash table, so x in set is generally faster than x in list).
- **Don't care about order** (sets are unordered collections).

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

Feature	String	List
Definition	Sequence of characters	Sequence of any data types
Mutable?	❌ Immutable (can't be changed)	✅ Mutable (can be changed)
Syntax	'hello' or "hello"	[1, 2, 3] or ['a', 'b']
Element type	Only characters (str)	Can hold mixed types

Strings are for text; lists are for collections of items (can be anything).

7.How do tuples ensure data integrity in Python?

- **Tuples are immutable**, meaning their content **cannot be changed** after creation.
- This immutability **prevents accidental modification** of data.
- Ensures **fixed, reliable data** throughout the program.
- Useful for representing **constant collections** like coordinates, settings, or keys.

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

- A **hash table** is a data structure that stores key-value pairs using a **hash function** to compute an index for each key.
- It allows **fast data retrieval** (average O(1) time) by directly accessing the storage location.
- **Python dictionaries** are implemented using hash tables internally.
- When you use a dictionary, Python hashes the key to quickly find, add, or remove values.

9. Can lists contain different data types in Python..?

- **Yes**, Python lists can hold **elements of different data types** in the same list.
- Example:

```
my_list = [1, "hello", 3.14, True]
```

- This makes lists flexible for storing mixed data.

10. Explain why strings are immutable in Python.?

- Strings are **immutable** to ensure **data integrity** and **security**.
- Immutability allows strings to be **hashable**, enabling their use as dictionary keys and set elements.
- It improves **performance** by allowing **string interning** and safe sharing between parts of a program.

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

- **Fast lookups** by key (average O(1) time) vs. slower list searches (O(n)).

- Store **key-value pairs** for direct access, not just indexed items.
- Keys are **unique**, enabling quick data retrieval without duplicates.
- Better for **mapping** relationships.

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

- Use a **tuple** when you need to store a **fixed, unchangeable group of values**. For example, storing a person's **(lalit, 22, 12/01/2003)** as a tuple is preferable because these values shouldn't change, and tuples are **faster and immutable**, making them ideal for **read-only data**.

13.How do sets handle duplicate values in Python..?

- In Python, sets automatically remove duplicate values. When you add duplicates to a set, only unique elements are kept, ensuring each value appears only once.

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

- In **lists**, the in keyword checks if a **value** exists.
- In **dictionaries**, in checks if a **key** exists — not the value.

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

- No, you **cannot modify elements of a tuple** because tuples are **immutable**. This means once created, their contents **cannot be changed**, ensuring data remains constant.

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

- A **nested dictionary** is a dictionary that contains **another dictionary as a value**. It's useful for organizing **structured data**, like records or settings.
- **Example:**
  Storing employee details

```
employees = {

   "E01": {"name": "John", "role": "Manager"},

   "E02": {"name": "Sara", "role": "Engineer"}

}
```

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

- Accessing elements in a dictionary has average time complexity of O(1) due to hashing, making it very fast. In rare worst cases (e.g., hash collisions), it can be O(n).

18 In what situations are lists preferred over dictionaries.?

- Lists are preferred when you need to store ordered, sequential data and access elements by position (index) rather than by key. They're ideal for iterating in order and managing collections of similar items.

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

- Dictionaries were considered unordered before Python 3.7, meaning they didn't maintain the insertion order. Since Python 3.7+, they do preserve order, but conceptually, retrieval is based on keys, not position—so order still doesn't affect how you access data by key.

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

- A list retrieves data by index (position), while a dictionary retrieves data by key. Lists are best for ordered data; dictionaries are best for key-value pairs.

Practical Questions

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

```
name = "lalit chauhan"

print(name)

lalit Chauhan
```

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

```
print(len("Hello World"))

11
```

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

```
string = "Python Programming"

sliced_string = string[:3]

print(sliced_string)

Pyt
```

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

```
string="hello"

uprString = string.upper()

print(uprString)

hello
```

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

```
String = "I like apple"

repString = String.replace('apple', 'orange')

print(repString)

I like orange
```

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

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

print(numList)


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

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

```
numList=[1, 2, 3, 4]
```

```
numList.append(10)
print(numList)
[1, 2, 3, 4, 10]
```

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

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

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

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

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

```
numList=[10, 20, 30, 40, 50]
print(numList[::-1])
[50, 40, 30, 20, 10]
```

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

```
fisrtTuple = (100,200,300)
print(fisrtTuple)
(100, 200, 300)
```

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

```
colTuple = ('red', 'green', 'blue', 'yellow')
print(colTuple[-2])
blue
```

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

```
numTuple = (10, 20, 5, 15)
print(min(numTuple))
 5
```

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

```
aniTuple = ('dog', 'cat', 'rabbit')
print(aniTuple.index('cat'))
1
```

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

```
fruTuple = ('Apple','mango','kiwi')

ifKiwi= 'kiwi' in fruTuple

print(ifKiwi)

True
```

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

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

print(alphSet)

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

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

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

clearSet = alphSet.clear()

print(clearSet)

None
```

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

```
alphaSet = {1, 2, 3, 4}

remSet= alphaSet.remove(4)

print(alphaSet)

{1, 2, 3}
```

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

```
set1 = {1, 2, 3}

set2 = {3, 4, 5}

setUnion = set1.union(set2)

print(setUnion)


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

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

```
set1 = {1, 2, 3}

set2 = {3, 4, 5}

setIntersect= set1.intersection(set2)

print(setIntersect)
```

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

```
perDict = {'name':'lalit', 'age':'22', 'city':'new delhi'}

print(perDict)


{'name': 'lalit', 'age': '22', 'city': 'new delhi'}
```

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

```
perDict = {'name':'lalit', 'age':'22', 'city':'new delhi'}

perDict['Country']='USA'

print(perDict)

{'name': 'lalit', 'age': '22', 'city': 'new delhi', 'Country': 'USA'}
```

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

```
persDict = {'name': 'Alice', 'age': 30}

val = persDict['name']

print(val)

Alice
```

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

```
bobdict = {'name': 'Bob', 'age': 22, 'city': 'New York'}

remDic = bobdict.pop('age')

print(remDic)

print(bobdict)

22

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

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

```
nameDict = {'name': 'Alice', 'city': 'Paris'}

doExist = 'city' in nameDict

print(doExist)

True
```

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

```
exList=['Jyoti','Anjali','Ruchi']

exTuple=('Komal','Priti','Priyanka')

exDict={'Name':'Komal','Age':'24','Relation':'wife'}

print(exList)

print(exTuple)

print(exDict)

['Jyoti', 'Anjali', 'Ruchi']

('Komal', 'Priti', 'Priyanka')

{'Name': 'Komal', 'Age': '24', 'Relation': 'wife'}
```

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
```

```
# Create a list of 5 random numbers between 1 and 100

random_numbers = random.sample(range(1, 101), 5)


# Sort the list in ascending order

random_numbers.sort()


# Print the result

print(random_numbers)

[63, 73, 89, 95, 97]
```

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

```
strList=['Hema','Rekha','Sushma','jaya','Nirma']

thirdIndex=strList[3]

print(thirdIndex)

jaya
```

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

```
Dict1={'Name':'tannu','Age':'21','Relation':'wife'}

Dict2={'MarriageDate':'23/05/2025'}

combDict={**Dict1,**Dict2}

print(combDict)

{'Name': 'tannu', 'Age': '21', 'Relation': 'wife', 'MarriageDate': '23/05/2025'}
```

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

```
string_list = ["apple", "banana", "cherry", "apple", "banana"]

string_set = set(string_list)

print(string_set)

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