1. What are data structures, and why are they important?

Data structures are special ways of organizing data so we can use it effectively. They help make programs faster, save memory, and solve problems more easily by storing and managing data in the best possible format.

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

Mutable types allow changes after they are created, like adding or removing items. Immutable types cannot be changed once made.

Example:

Mutable: list, dict, set

Immutable: str, int, tuple

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

Lists are changeable (mutable), while tuples are fixed (immutable). Lists use more memory and are slower, while tuples are faster and safer for fixed data.

4. Describe how dictionaries store data.

Dictionaries keep data as key-value pairs. Each key is linked to a value, and keys must be unique. They use hashing to access data quickly.

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

Sets are used when you want only unique items and faster checking of whether a value is present.

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

A string is a sequence of characters and is immutable. A list is mutable and can store different types of items.

7. How do tuples ensure data integrity in Python?

Tuples are immutable, so their values cannot be changed once defined. This prevents accidental changes to the data.

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

A hash table is a structure that maps keys to values for fast access. Python dictionaries use hash

tables internally.

9. Can lists contain different data types in Python?

Yes, lists can contain elements of different data types including numbers, strings, and even other lists.

10. Explain why strings are immutable in Python.

Strings are immutable to improve performance and ensure they remain unchanged during operations.

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

Dictionaries allow fast access using keys and are more efficient when dealing with labeled data.

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

Tuples are preferred for fixed data, such as coordinates or as keys in dictionaries due to their immutability.

13. How do sets handle duplicate values in Python?

Sets automatically remove duplicate values, keeping only unique elements.

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

'in' checks for presence of a value in a list and checks for presence of a key in a dictionary.

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

No, tuples are immutable. Once created, their contents cannot be changed.

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

A nested dictionary is a dictionary inside another dictionary, useful for organizing structured data.

Example: {'name': 'Sara', 'grades': {'math': 90}}

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

Accessing items in a dictionary usually takes constant time, O(1), due to hashing.

18. In what situations are lists preferred over dictionaries?

Lists are better when order matters and you don't need key-value pairs.

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

Dictionaries were unordered before Python 3.7. Now they maintain insertion order but still use keys for access, not index.

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

Lists use index to retrieve items, while dictionaries use keys.

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

```
my_name = "Nidhi"
print(my_name)
```

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

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

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

```
phrase = "Python Programming"
first_three = phrase[:3]
print(first_three)
```

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

```
word = "hello"
print(word.upper())
```

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

```
sentence = "I like apple"
new_sentence = sentence.replace("apple", "orange")
print(new_sentence)
```

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

```
numbers = [1, 2, 3, 4, 5]
print(numbers)
```

27. 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)

print(my_list)
```

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

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

numbers.remove(3)

print(numbers)
```

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

```
letters = ['a', 'b', 'c', 'd']

print(letters[1])
```

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

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

nums.reverse()

print(nums)
```