1. What is data structures, and why are they important?
ANS- Data structures are ways of storing and organizing data in a computer so it can be used efficiently.
   # Why are They Important?

Efficiency: Makes searching, sorting, and updating data faster.

Memory management: Stores data in an organized and compact way.

Reusability: Same structure can be used in many programs.

Scalability: Handles large and complex data easily.

Problem-solving: Essential for algorithms in real-world applications (e.g., Google Maps uses graphs).

2. Expalin the difference between the mutable and immutable data types with examples.
ANS- 1. Mutable Data Types

Definition: Data types whose values can be changed/modified after creation.

You can add, remove, or update elements without creating a new object.

Examples in Python:

List → [1, 2, 3]

Dictionary → {"name": "Raj", "age": 21}

Set → {10, 20, 30}

2. Immutable Data Types

Definition: Data types whose values cannot be changed after creation.

If you try to modify, a new object is created instead of changing the old one.

Examples in Python:

Tuple → (1, 2, 3)

String → "hello"

Integer → 10

Float → 3.14

3. What are the main differnce between lists and tuples in python?

```
ANS- 1. Mutability

List → Mutable (you can change, add, remove elements)

Tuple → Immutable (cannot change once created)

 2. Syntax

List → Created with square brackets [ ]

Tuple → Created with parentheses ( )

4. Describe how dictionaries strore data.
ANS - #A dictionary in Python is a collection of key–value pairs.

Each key must be unique and immutable (like a string, number, or
tuple).

Each value can be of any data type (list, string, int, another dict,
etc.).
    #Collisions

Sometimes two different keys may produce the same hash value → this is
called a collision.

Python handles collisions using open addressing / probing (tries the
next available slot).

5. Why might you use a set instead of a list in Python?
ANS - #Uniqueness of Elements

Set automatically removes duplicate values.

List allows duplicates.
    #Mathematical Operations

Set supports operations like union, intersection, difference (like in
math).

List does not directly support these.

6. What is a string in Python, and how is it different from a list?
ANS-  A string is a sequence of characters enclosed in quotes (' ' or
" ").
      Strings are used to store text.
    In string sequence of Character , in  list sequence of elements.
    String is immutable but list are mutable.
    String enclosed with ' ',“ ”, but list enclosed with [ ].

7. How do tuples ensure data integrity in Python?
ANS - 1. Immutability
```

A tuple is immutable → once created, its elements cannot be changed, added, or removed.

This immutability protects the data from accidental modification.
     2. Reliability in Fixed Data

Tuples are often used when you want to represent constant collections of values (like coordinates, RGB colors, or database records).

Since values cannot change, the data remains stable and consistent throughout the program.

8. What is a hash table , and how does it relate to dictionaries in Python?
ANS - A hash table is a data structure that stores data in key–value pairs.

It uses a hash function to convert a key into an integer (called a hash value).

This hash value determines the index (or bucket) where the value is stored.

Because of hashing, lookup, insertion, and deletion operations are usually very fast (O(1) on average).

#Relation to Dictionaries in Python

In Python, a dictionary (dict) is implemented using a hash table.

Each dictionary entry is stored as a key–value pair inside a hash table.

The key is hashed to find where the value should go.

When you search for a key, Python recomputes its hash → goes straight to the location → fetches the value.

9. Can lists contain different data types in Python?
ANS- Yes, Python lists are heterogeneous, which means a single list can store elements of different data types.

10. Expalin why string are immutable in python.
ANS- *#Reasons for String Immutability*

1.Memory Efficiency

Immutable objects can be shared safely.

Python can store only one copy of a string used in multiple places →

saves memory.

2.Hashing & Dictionary Keys

Strings are often used as keys in dictionaries or elements in sets.

If strings were mutable, their hash value could change → would break dictionary/set lookups.

11. What advantage do dictionaries offer over lists for certain tasks?
ANS - *#Advantages of Dictionaries over Lists*

Dictionaries in Python are key–value pairs, while lists are ordered sequences. This makes dictionaries better than lists for certain tasks:

1. Fast Lookup

Dictionaries use a hash table, so searching for a key is O(1) on average.

Lists require scanning all elements → O(n) time for searching.
2. Suitable for Complex Data

Dictionaries are ideal when you have structured data, like JSON or database records.

Lists are better for sequences or collections, not key-based access.

12. Describe a scenario where using a tuple would be preferable over a list.
ANS- Reason: Tuples are immutable (cannot be changed), while lists are mutable.

This makes tuples safer for fixed data that shouldn’t be modified accidentally.

13. How do sets handle duplicate values in Python?
ANS- *#Definition of a Set*

A set is an unordered collection of unique elements.

Sets automatically remove duplicate values.

```
    # Creating a set with duplicates
my_set = {1, 2, 2, 3, 4, 4, 5}
print(my_set)
```
  OUTPUT- {1,2,3,4,5}

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

```
ANS - The in Keyword in Python

The in keyword is used to check membership (whether an item exists in
a collection).
However, it works differently for lists and dictionaries.

1. Lists

Lists are ordered sequences.

in checks whether the value exists anywhere in the list.

Search time: O(n) → Python may need to check each element.

my_list = [10, 20, 30, 40]
print(20 in my_list)  # True
print(50 in my_list)  # False

2. Dictionaries

Dictionaries are key–value pairs.

in only checks keys, not values.

Search time: O(1) average → very fast, because dictionaries use hash
tables.

my_dict = {"name": "Rajvardhan", "age": 25}

print("name" in my_dict)   # True
print("Rajvardhan" in my_dict)  # False (value is not checked)

15. Can you modify the elements of a tuple? Expalin why are why not?
ANS - You cannot modify tuple elements directly because tuples are
immutable.

Immutability ensures data integrity, performance, and safe usage as
dictionary keys.

Mutable objects inside a tuple can still be changed, but the tuple's
overall structure remains fixed.

16. What is nested dictionary, and give an exampke of its use case?
ANS - A nested dictionary is a dictionary inside another dictionary.

Each key in the outer dictionary can map to a value that is itself a
dictionary.

Useful for storing structured or hierarchical data.

student = {
```

```
    "101": {"name": "ANKIT", "age": 20, "marks": 85},
    "102": {"name": "RAJVARDHAN", "age": 21, "marks": 90}
}

# Accessing nested data
print(student["101"]["name"])   # ANKIT
print(student["102"]["marks"])  # 90
```

17. Describe the time complexity of accessing elements in a dictionary.
ANS - A Python dictionary is implemented using a hash table, which makes element access very fast.

1. Access by Key

```
my_dict = {"name": "Alice", "age": 25, "city": "Delhi"}
print(my_dict["age"])  # 25
```

Python calculates the hash of the key "age".

The hash points directly to the location in memory where the value is stored.

This means accessing a value by its key is O(1) on average → constant time, no matter how large the dictionary is.

18. In what situations are lists preferred over dictionaries?
ANS - 1. Ordered Collections

Lists maintain order of elements.

Use lists when the sequence matters, like storing steps in a process or items in a queue.

2. Index-Based Access

Lists allow direct access by index (list[0], list[1], etc.).

Use lists when you need position-based access or iteration by index.

19. Why are dictionaries considered unordered , and how does that affect data retrival?
ANS - *#Definition of Unordered*

Traditionally, a Python dictionary is unordered, meaning the order in which you add key–value pairs is not guaranteed to be preserved.

Internally, dictionaries are implemented using a hash table, which stores keys based on their hash values, not the order of insertion.

#Effect of Hashing

Each key is hashed to determine where its value will be stored in memory.

Because the hash function decides the storage location, the order of keys may appear random when iterating over the dictionary.

```
    #How This Affects Data Retrieval
```

Access by Key Is Still Fast

Even though the dictionary is unordered, retrieving a value by its key is O(1) average time, because the hash function points directly to the value.

```
print(my_dict["b"])  # 2 → very fast lookup
```

Iteration Order Is Not Guaranteed (Before 3.7)

If your code relies on the order of keys, using a dictionary in older Python versions may lead to unexpected order.

20. Explain the differnece between a list and a dictionary in terms of data retrieval.
ANS - List: Retrieval is index-based, slower for searching by value, order matters.

Dictionary: Retrieval is key-based, very fast (O(1)), no order needed.

```
# PRACTICAL QUESTIONS
```

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

```
name = "Rajvardhan verma"
print(name)
```

```
Rajvardhan verma
```

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

```
x = "Hello world"
print(len(x))
```

```
11
```

3. write a code to slice the first 3 character from the string "python programming".

```
x = "python programming"
slice_x = x[:3]
print(slice_x)
```

```
pyt
```

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

```
x = "hello"
upper_x = x.upper()
print(upper_x)
```

```
HELLO
```

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

```
x = "I like apple"
y = x.replace("apple" , "orange")
print(y)
```

```
I like orange
```

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

```
x = [1,2,3,4,5,6,7]
print(x)
```

```
[1, 2, 3, 4, 5, 6, 7]
```

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

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

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

8. 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)
```

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

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

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

```
b
```

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

```
numbers = [10,20,30,40]
numbers.reverse()
print(numbers)
```

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

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

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

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

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

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

```
blue
```

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

```
numbers = (10,20,5,15)
min_num = min(numbers)
print(min_num)
```

```
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')
print(index)
```

```
1
```

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

```
fruits = ('apple','kiwi','orange')
if "kiwi" in fruits:
     print("kiwi is present in the tuple")
```

```
else:
     print("kiwi is not present in the tuple")
```

```
kiwi is present in the tuple
```

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

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

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

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

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

```
set()
```

18. Write a code to remove 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 {2,3,4}.

```
x = {1,2,3}
y = {2,3,4}
z = x.union(y)
print(z)
```

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

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

```
x = {1,2,3}
y = {2,3,4}
z = x.intersection(y)
print(z)
```

```
{2, 3}
```

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

```
my_dict = {
    "name": "Rajvardhan Verma",
    "age": "24",
```

```
    "city": "Varanasi"
}
print(my_dict)
```

```
{'name': 'Rajvardhan Verma', 'age': '24', 'city': 'Varanasi'}
```

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

```
my_dict = {'name':'john','age':25}
my_dict["country"] = "USA"
print(my_dict)
```

```
{'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'}

```
my_dict = {'name':'Alice','age':'30'}
name_value = my_dict['name']
print("value of the 'name':" ,name_value)
```

```
value of the 'name': Alice
```

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

```
my_dict = {'name':'Bob','age':22,'city':'newyork'}
my_dict.pop('age')
print(my_dict)
```

```
{'name': 'Bob', 'city': 'newyork'}
```

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

```
my_dict = {'name':'Alice','city':'paris'}
if "city" in my_dict:
    print("true")
else:
    print("false")
```

```
true
```

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

```
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'}
```

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.

```
import random

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

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

# Print the sorted list
print("Sorted list:", random_numbers)
```

```
Sorted list: [5, 15, 18, 31, 71]
```

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

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

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

```
Element at index 3: date
```

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

```
dict1 = {"name": "Alice", "age": 25}
dict2 = {"city": "New York", "country": "USA"}

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

# Print the combined dictionary
print("Combined Dictionary:", combined_dict)
```

```
Combined Dictionary: {'name': 'Alice', 'age': 25, 'city': 'New York',
'country': 'USA'}
```

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

```
my_list = ["apple", "banana", "cherry", "apple", "banana"]
my_set = set(my_list)
print("Set:", my_set)
```

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