

ASSIGNMENT 2 DATA TYPES AND STRUCTURES QUESTIONS

Double-click (or enter) to edit

1.What are data structures, and why are they important? -A data structure is a specialized format for organizing, processing, retrieving and storing data. There are several basic and advanced types of data structures. Data structures bring together the data elements in a logical way and facilitate the effective use, persistence and sharing of data. They provide a formal model that describes the way the data elements are organized. Data structures are the building blocks for more sophisticated applications.

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

- Mutable data types are those that can be changed after their creation, while immutable data types cannot be altered once created. For example, lists are mutable, meaning you can add, remove, or change elements after the list is created.

3.What are the main differences between lists and tuples in Python? -Tuples are immutable objects and lists are mutable objects. Once defined, tuples have a fixed length and lists have a dynamic length. Tuples use less memory and are faster to access than to lists. Tuple syntax uses round brackets or parenthesis, and list syntax uses square brackets.

4.Describe how dictionaries store data? -Dictionaries are used to store data values in key:value pairs. A dictionary is a collection which is ordered*, changeable and do not allow duplicates. As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered.

5.Why might you use a set instead of a list in Python? -One of the main advantages of using sets in Python is that they are highly optimized for membership tests. For example, sets do membership tests a lot more efficiently than lists.

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

- string is a sequence of characters between single or double quotes. A list is a sequence of items, where each item could be anything (an integer, a float, a string, etc).

7.How do tuples ensure data integrity in Python? -Tuples in Python ensure data integrity through their immutability. Once a tuple is created, its contents cannot be modified, which means elements cannot be added, removed, or changed. This characteristic guarantees that the data within a tuple remains constant throughout its lifecycle. Immutability is useful in scenarios where data integrity is crucial, such as representing fixed records, storing configuration settings, or ensuring that data passed between functions remains unchanged. Because of their immutability, tuples can also be used as keys in dictionaries and elements in sets.

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, where each key is unique and mapped to a specific location (index) in an array using a hash function. This allows for efficient retrieval, insertion, and deletion of data. In Python, dictionaries are implemented using hash tables. When a key-value pair is added to a dictionary, Python calculates the hash value of the key and uses it to determine the index where the value will be stored. When retrieving a value, Python calculates the hash value of the key again and uses it to locate the corresponding value in the hash table. Hash tables handle collisions, which occur when different keys have the same hash value, typically through techniques like chaining (using linked lists) or open addressing. Python automatically manages the size of the hash table, resizing it as needed to maintain performance.

9.Can lists contain different data types in Python? -Yes, lists in Python can contain elements of different data types. Python lists are versatile and allow for storing a mix of integers, strings, floats, booleans, or even other lists and data structures within the same list. This flexibility is a key feature of Python.

10.Explain why strings are immutable in Python? -Python strings are "immutable" which means they cannot be changed after they are created (Java strings also use this immutable style). Since strings can't be changed, we construct *new* strings as we go to represent computed values.

11.What advantages do dictionaries offer over lists for certain tasks? -List Dictionary It can contain duplicate elements. It cannot contain duplicate keys. Slicing can be done. Slicing cannot be done. It is preferred when the contents are not fixed. It is preferred when working with large amounts of data.

12.Describe a scenario where using a tuple would be preferable over a list? -Tuples are immutable. Hence, they are primarily used to store data that doesn't change frequently. Any operation can store data in a tuple when you don't want it to change. Tuples are great to use if you want the data in your collection to be read-only, never to change, and always remain the same and constant.

13.How do sets handle duplicate values in Python? -Sets in Python are designed to store only unique elements. When an attempt is made to add a duplicate value to a set, Python automatically discards the duplicate, ensuring that the set retains only distinct elements. This behavior is a fundamental characteristic of sets, making them useful for tasks such as removing duplicates from a list or checking for the presence of an element.

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

✓ Expected output: {1, 2, 3, 4, 5}

14.How does the "in" keyword work differently for lists and dictionaries? -The in and not in operator works on dictionaries; it tells you whether something appears or not as a key in the dictionary. It

returns two values True or False. The in operator uses different algorithms for lists and dictionaries. For lists, it uses a search algorithm.

15.Can you modify the elements of a tuple? Explain why or why not? -No, you cannot directly modify the elements of a tuple in Python after it's been created. Tuples are immutable, meaning their contents cannot be changed once they are defined. Trying to modify a tuple will result in a `TypeError`

16.What is a nested dictionary, and give an example of its use case? -A nested dictionary in Python is a dictionary where the values are themselves dictionaries. This allows for the creation of a hierarchical structure where data can be organized into multiple levels. A common use case for nested dictionaries is representing complex, multi-layered data, such as employee information with nested address and contact details.

17.Describe the time complexity of accessing elements in a dictionary? -Accessing an element in a dictionary using its key typically has a time complexity of $O(1)$ on average, meaning it takes constant time, regardless of the dictionary's size. However, in the worst-case scenario, it can degrade to $O(n)$, where n is the number of key-value pairs, if there are many hash collisions.

18.In what situations are lists preferred over dictionaries? -For quick data look-ups, configurations, or caches, favor dictionaries. For ordered collections and sequence operations, such as maintaining a stack or queue, lists are more suitable. Lists are faster for index-based access, as retrieving an element by index takes $O(1)$, whereas dictionary lookups require computing a hash. Dictionaries require more memory, as they store keys along with values, unlike lists.

19.Why are dictionaries considered unordered, and how does that affect data retrieval? -A dictionary is termed an unordered collection of objects because dictionaries do not maintain any inherent order of the items based on when they were added. In older versions of Python (before 3.7), dictionaries did not preserve insertion order at all.

20.Explain the difference between a list and a dictionary in terms of data retrieval. In terms of data retrieval, a dictionary excels at quickly accessing values using unique keys, while a list retrieves values based on their position (index) in a sequence. Lists require searching through elements, while dictionaries use hashing for near-instantaneous lookup, making them significantly faster for accessing data by a specific identifier.

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

```
name=input("Enter your name")
```



Enter your nameAvni

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

```
string = "hello world"
length = len(string)
print(length)
```

⇒ 11

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

```
string = "Python Programming"
sliced_string = string[0:3]
print(sliced_string)
```

⇒ Pyt

Start coding or [generate](#) with AI.

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

```
s = "learn python with gfg!"
res = s.upper()
print(res)
```

⇒ LEARN PYTHON WITH GFG!

Start coding or [generate](#) with AI.

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


```
word_mapping = {
    'apple': 'orange',
    'love': 'like'
}
```

Start coding or [generate](#) with AI.

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

```
r1 = 0
r2 = 5


li = list(range(r1, r2))
print(li)
```

 [0, 1, 2, 3, 4]

Start coding or [generate](#) with AI.

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)
print(my_list) # Output: [1, 2, 3, 4, 10]
```

 [1, 2, 3, 4, 10]

Start coding or [generate](#) with AI.

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)
print(my_list)
```

 [1, 2, 4, 5]

Start coding or [generate](#) with AI.

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


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

 b

Start coding or [generate](#) with AI.

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

```
my_list = [10, 20, 30, 40, 50]
my_list.reverse()
print(my_list)
```

 [50, 40, 30, 20, 10]

Start coding or [generate](#) with AI.

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

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

➞ (100, 200, 300)

Start coding or [generate](#) with AI.

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

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

➞ blue

Start coding or [generate](#) with AI.

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

```
my_tuple = (10, 20, 5, 15)
min_number = min(my_tuple)
print(min_number)
```

➞ 5

Start coding or [generate](#) with AI.

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

```
my_tuple = ('dog', 'cat', 'rabbit')
index_of_cat = my_tuple.index('cat')
print(index_of_cat)
```

➞ 1

Start coding or [generate](#) with AI.

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

```
thistuple = ("apple", "kiwi", "cherry")  
print(thistuple[1])
```

⇒ kiwi

Start coding or [generate](#) with AI.

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'}

Start coding or [generate](#) with AI.

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()

Start coding or [generate](#) with AI.

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}.

```
set1 = {1, 2, 3}  
set2 = {3, 4, 5}
```

```
union_set = set1.union(set2)
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}.

```
set1 = {1, 2, 3}
set2 = {2, 3, 4}

intersection_set = set1.intersection(set2)

print(intersection_set)
```

```
➞ {2, 3}
```

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

```
my_dict = {
    "name": "Avni",
    "age": 23,
    "city": "Delhi"
}
print(my_dict)
```

```
➞ {'name': 'Avni', 'age': 23, 'city': 'Delhi'}
```

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,
    "city": "USA"
}

print(my_dict)
```

```
➞ {'name': 'John', 'age': 25, 'city': 'USA'}
```

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(name_value)
```

⇒ Alice

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

```
my_dict = {"name": "Bob", "age": 22}
del my_dict["age"]
print(my_dict)
```

⇒ {'name': 'Bob'}

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

```
my_dict = {
    "name": "Alice",

    "city": "New York"
}
```

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

```
# Creating a list
my_list = [1, 2, 3, "apple", "banana"]

# Creating a tuple
my_tuple = (4, 5, 6, "grape", "orange")

# Creating a dictionary
my_dictionary = {"name": "John", "age": 30, "city": "New York"}

# Printing the list
print("List:", my_list)

# Printing the tuple
print("Tuple:", my_tuple)

# Printing the dictionary
print("Dictionary:", my_dictionary)
```

⇒ List: [1, 2, 3, 'apple', 'banana']
Tuple: (4, 5, 6, 'grape', 'orange')
Dictionary: {'name': 'John', 'age': 30, '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.(replaced)

```
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(random_numbers)
```

⇒ [5, 32, 55, 76, 95]

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

```
a = [10, 20, "GfG", 40, True]

print(a)

# Accessing elements using indexing
print(a[0]) # 10
print(a[1]) # 20
print(a[2]) # "GfG"
print(a[3]) # 40
print(a[4]) # True

# Checking types of elements
print(type(a[2])) # str
print(type(a[4])) # bool
```

⇒ [10, 20, 'GfG', 40, True]
 10
 20
 GfG
 40
 True
 <class 'str'>
 <class 'bool'>

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

```
dict1 = {"name": "Avni", "age": 23}
dict2 = {"city": "Delhi", "country": "India"}
```

```
# Method 1: Using the update() method (modifies the first dictionary)
dict1.update(dict2)
print("Merged dictionary (Method 1):", dict1)
```

```
# Method 2: Using the ** operator (creates a new dictionary)
dict3 = {**dict1, **dict2}
print("Merged dictionary (Method 2):", dict3)
```

```
# Method 3: Using the | operator (Python 3.9+) (creates a new dictionary)
dict4 = dict1 | dict2
print("Merged dictionary (Method 3):", dict4)
```

```
⇒ Merged dictionary (Method 1): {'name': 'Avni', 'age': 23, 'city': 'Delhi', 'country': 'I'}
Merged dictionary (Method 2): {'name': 'Avni', 'age': 23, 'city': 'Delhi', 'country': 'I'}
Merged dictionary (Method 3): {'name': 'Avni', 'age': 23, 'city': 'Delhi', 'country': 'I'}
```

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

```
def convert_list_to_set(string_list):
    string_set = set(string_list)
    return string_set

my_list = ["apple", "banana", "cherry", "apple"]
my_set = convert_list_to_set(my_list)
print(my_set)
# Expected output: {'apple', 'banana', 'cherry'}
```

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

Start coding or [generate](#) with AI.