**Assignment 2: Data Types and Structures**

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

> Data structures are organized ways to store, manage, and retrieve data efficiently in a computer.
>
> Importance of Data Structure :
>
> > Efficient Data Management :Help organize large amounts of data so it's easy to access and modify.
> >
> > Faster Searching & Sorting: Certain data structures allow faster operations.
> >
> > Memory Optimization: Help reduce memory usage by using optimal storage formats.
> >
> > Foundation for Algorithms: Most algorithms rely on data structures to work correctly and efficiently.
> >
> > Real-world Problem Solving: Useful in tasks like file systems, databases, social networks, etc.
> >
> > Helps in Decision Making: Choosing the right data structure can improve performance drastically.
>
> Examples: List,tuple,sets

+ Code + Text

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

> Mutable:
>
> > Objects that can be changed after they are created.
> >
> > No new memory location is created
> >
> > Example :list, dict, set

```
# Example of Mutable
my_list = [1, 2, 3]
my_list[0] = 10
print(my_list)
```

> Immutable:
>
> > Objects that cannot be changed after they are created.
> >
> > Creation of new memory location at any changes
> >
> > Examples: int, float, str, tuple

```
# Example of immutable
t = (1, 2, 3)
# t[0] = 10  #This will raise a TypeError
```

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

> Difference bwtween List and tuple
>
> 1. Mutability:**Lists** are mutable whereas **tuples** are immutable.
>
> > This means you can change, add, or remove elements in a list, but you cannot modify a tuple after it's created.
>
> 2. Syntax: **Lists** use square brackets [ ], whereas **tuples** use parentheses ( ).
>
> > This means you define a list like [1, 2, 3] and a tuple like (1, 2, 3).
>
> 3. Methods Available: **Lists** have many built-in methods, whereas **tuples* *have very few.
>
> > This means you can modify a list using methods like append(), pop(), or sort(), but tuples only support count() and index().
>
> 4. Performance: **Tuples** are faster in performance than** lists**.

> This is because tuples are immutable and require less overhead, making them more efficient in fixed-size data operations.

5. Memory Usage: **Tuples** use less memory than **lists.**

> Since tuples are fixed in size, they consume less memory than dynamic lists.

6. Use Case: **Lists** are used when data can change, whereas **tuples** are used for fixed data.

> This means use lists for dynamic operations, and tuples when you want to protect data from modification.

4. Describe how dictionaries store data .

> A dictionary in Python is a collection of key-value pairs, where each key is unique and is used to access its corresponding value.

> Dictionaries store data:

> > A dictionary stores data as key-value pairs.

> > The key is used to identify the data.

> > The value is the data associated with that key.

> > It allows for quick access, update, and deletion of data using keys.

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

> Reasons to use set instead of list :

> 1. No duplicate elements : A set automatically removes duplicates, while a list can store them.
> 2. Faster lookups : Checking if an element exists in a set (in operation) is generally O(1), compared to O(n) for a list.
> 3. Unordered collection : If you don't care about the order of elements, sets are more efficient.
> 4. Mathematical operations : Sets support union, intersection, and difference operations directly.
> 5. Better for uniqueness checks : When you need only unique items, sets avoid manual duplicate removal.

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

> String: A string is a sequence of characters. Python treats anything inside quotes as a string. This includes letters, numbers, and symbols. Python has no character data type so single character is a string of length 1.

> String different from list:

> > 1. Data Type : A string stores only characters, while a list can store elements of any data type (numbers, strings, objects, etc.).
> > 2. Mutability : Strings are immutable, whereas lists are mutable .
> > 3. Element Type : Strings can contain only characters, but lists can have mixed data types.
> > 4. Modification : You cannot directly change a character in a string, but you can change, add, or remove elements in a list.
> > 5. Operations : Strings have text-specific methods (upper(), replace(), split()), while lists have collection-specific methods (append(), remove(), sort()).
> > 6. Representation : Strings are written inside quotes (' ', " "), while lists are written inside square brackets [ ].
> > 7. Usage : Strings are used for text data, while lists are used for storing a sequence of items.

7. How do tuples ensure data integrity in Python ?

> Tuples ensure data integrity in Python mainly because they are immutable.

> Other reasons tuples ensure data integrity:

> > 1. Prevents accidental changes : Since tuples can't be modified, your data stays exactly as it was when created.

2. Reliable as constants : Tuples can store fixed data (like configuration values) that must remain the same throughout the program.
3. Safe for keys in dictionaries : Their immutability allows them to be used as dictionary keys or stored in sets without risk of being altered.
4. Predictable behavior : No unexpected changes in tuple data make debugging and tracking logic easier.
5. Thread-safety : In multi-threaded programs, immutable data structures like tuples reduce the chance of data corruption from concurrent modifications.

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 and uses a hash function to map each key to a specific index (bucket) in memory.

Relation of Hashtable to dictionaries :

1. In Python, a dictionary (dict) is implemented using a hash table.
2. Keys in a dictionary are hashed internally, and their hash values decide where the corresponding values are stored in memory.
3. Only immutable and hashable objects can be used as dictionary keys.
4. Hash tables let dictionaries perform very fast lookups, even with large amounts of data.

```
my_dict = {"name": "Sakshi", "age": 25}
# "name" is hashed internally → value 25 stored at a memory bucket
print(my_dict["name"])  # Output: Alice
```

9. Can lists contain different data types in Python ?

Yes, lists in Python can contain different data types in the same list.

A list can hold integers, floats, strings, booleans, objects, even other lists — all mixed together.

This is possible because Python lists are dynamic and store references to objects, not fixed-type elements.

```
my_list = [10, "Hello", 3.14, True, [1, 2, 3]]
print(my_list)
```

10. Explain why strings are immutable in Python .

Reasons String are immutable:

1. Memory efficiency : Immutability allows Python to reuse identical string objects (string interning), saving memory.
2. Hash consistency : Strings must stay unchanged so their hash value remains constant when used as dictionary or set keys.
3. Thread-safety : Because strings can't be modified, they can be shared between threads without risk of corruption.
4. Predictable behavior : Fixed, unchangeable strings prevent accidental changes, making code more reliable.
5. Performance optimization : Immutable strings can be stored and accessed in a way that speeds up certain operations.

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

Advantages do dictionaries offer over lists for certain tasks are:

1. Faster lookups : Dictionaries provide average O(1) time complexity for accessing values by key, while lists take O(n) for searches.
2. Key-based access : You can access data using meaningful keys instead of relying on numeric indexes.
3. No need for manual searching : Values are retrieved directly by key without iterating through the entire collection.
4. Unordered but organized : Data is stored in a way that's easy to manage when you care about associations rather than order.

> 5. Flexible keys and values : Keys can be any immutable type (string, number, tuple), and values can be any data type.

> 6. Clear mapping of data : Perfect for storing real-world data with a clear “label → value” relationship.

```
# List of [name, phone] pairs
contacts_list = [["Alice", "1234"], ["Bob", "5678"], ["Charlie", "9101"]]

# Find Bob's number
for name, phone in contacts_list:
    if name == "Bob":
        print(phone)  # Output: 5678
        break


# Dictionary of name: phone pairs
contacts_dict = {"Alice": "1234", "Bob": "5678", "Charlie": "9101"}

# Find Bob's number
print(contacts_dict["Bob"])  # Output: 5678
```

> Difference : Time Complexity of Dictionary is O(1) where as list is O(n).

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

> Scenario:
>
> > Storing fixed GPS coordinates of a city in a program.

Example:

```
# Latitude and Longitude of New York
new_york_coordinates = (40.7128, -74.0060)
```

> Why tuple is preferable:
>
> 1. The coordinates should never change during the program’s execution → tuples are immutable, ensuring data integrity.
>
> 2. Tuples can be used as dictionary keys if you need to store location-based data (lists cannot be keys).
>
> 3. Slightly faster than lists for access, making them efficient for fixed datasets.

13. How do sets handle duplicate values in Python ?

> Sets automatically remove duplicate values because they only store unique elements.
>
> When you try to add a duplicate to a set, Python ignores it silently.
>
> This is possible because sets use a hash table internally, where each element’s hash value must be unique.

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

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

> The in keyword behaves differently for lists and dictionaries in Python:
>
> 1.For lists "in" checks if a value exists anywhere in the list.

```
numbers = [1, 2, 3, 4]
print(3 in numbers)   #(checks values)
print(5 in numbers)
```

> 2.For dictionaries – in checks if a key exists in the dictionary (not the value).

```
person = {"name": "Alice", "age": 25}
print("name" in person)   #  checks keys
print("Alice" in person)  # does not check values
```

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

> No we cannot modify the elements of a tuple in Python.
>
> Reason:
>
> 1. Tuples are immutable, meaning once they are created, their elements cannot be changed, added, or removed.
> 2. This immutability ensures data integrity, allows tuples to be used as dictionary keys or stored in sets, and makes them more memory-efficient and thread-safe.

```
t = (1, 2, 3)
t[0] = 10 # TypeError: 'tuple' object does not support item assignment
```

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

> Nested Dictionary:
>
> 1. A nested dictionary is a dictionary that contains another dictionary as one or more of its values.
> 2. It's used to store hierarchical or structured data in a compact, organized way.

```
# Example of usecase
students = {
    "Alice": {"age": 20, "marks": {"Math": 90, "Science": 85}},
    "Bob": {"age": 22, "marks": {"Math": 78, "Science": 88}}
}

# Access Bob's Science marks
print(students["Bob"]["marks"]["Science"])
```

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

> Accessing elements in a dictionary in Python has:
>
> 1. Average case: O(1) : Constant time, because dictionaries use a hash table for key lookups.
> 2. Worst case: O(n) : Happens rarely, only when many keys have the same hash value (hash collision) and Python has to search through them linearly.

```
person = {"name": "Alice", "age": 25}
print(person["age"])  # O(1) average time
```

18. In what situations are lists preferred over dictionaries ?

> Situations where lists are preferred over dictionaries:
>
> 1. Order matters : When you need to preserve and work with the sequence of elements.
> 2. No key-value mapping needed : When you only care about storing values, not associating them with unique keys.
> 3. Allow duplicates : When you need to store the same value multiple times.
> 4. Index-based access : When elements are best accessed by their position (index) rather than by a key.
> 5. Small datasets : For small collections where the speed benefit of a dictionary is negligible.
> 6. Iteration over values : When you want to loop through elements directly without worrying about keys.

```
fruits = ["apple", "banana", "orange", "banana"]
print(fruits[1])
```

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

> Dictionaries are considered unordered:
>
> 1. Before Python 3.7, dictionaries did not guarantee that elements would be stored or iterated in the order they were inserted.

2. This is because dictionaries are implemented using a hash table, where the storage location of each key-value pair depends on the key's hash value, not its position.

Effect on data retrieval

1. Key-based retrieval is unaffected we can always access a value directly using its key in O(1) time, regardless of order.
2. However, iteration order was unpredictable in older versions of Python, meaning looping over a dictionary could produce elements in any sequence.
3. From Python 3.7+, insertion order is preserved as an implementation detail but retrieval by key is still independent of order.

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

Difference between a list and a dictionary in terms of data retrieval:

1. List : Retrieval is index-based

We access elements using their position (0, 1, 2, …).

Time complexity: O(1) for direct index access, but O(n) if you search for a value.

```
fruits = ["apple", "banana", "cherry"]
print(fruits[1])
```

2. Dictionary : Retrieval is key-based

We access values using unique keys, not positions.

Time complexity: O(1) on average because of hashing.

```
person = {"name": "Alice", "age": 25}
print(person["age"])
```

**Practical Questions**

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

name=input("Enter your name")
print(f"My name is {name}")
```

```
Enter your namesakshi
My name is sakshi
```

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

string="Hello World"
length=len(string)
print("Length of string is :",length)
```

```
Length of string is : 11
```

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

string="Python Programming"
print("String after slicing first 3 character :",string[3:])
```

```
String after slicing first 3 character : hon Programming
```

```
#4 Write a code to convert the string "hello" to uppercase

string="hello"
print("String in uppercase",string.upper())
```

```
String in uppercase HELLO
```

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

string="I like apple "

new_string=string.replace("apple","orange") #replacing the word apple --->orange
```

```
print("String before replacement :",string)
print("String after replacemnt: ",new_string)
```

```
String before replacement : I like apple
String after replacemnt:  I like orange
```

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

list=[] #empty list
for i in range(1,6):
  list.append(i) #appending elements
print("List with number 1 to 5",list)
```

```
List with number 1 to 5 [1, 2, 3, 4, 5]
```

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

list=[1,2,3,4]
print("List before appending 10 : ",list)

list.append(10) #adding 10 to list

print("List after appending 10 : " ,list)
```

```
List before appending 10 :  [1, 2, 3, 4]
List after appending 10 :  [1, 2, 3, 4, 10]
```

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

list=[1,2,3,4,5]
print("List before removing 3 : ",list)

list.remove(3) #remove 3 from list

print("List after removing 3 : " ,list)
```

```
List before removing :  [1, 2, 3, 4, 5]
List after removing 3 :  [1, 2, 4, 5]
```

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

list=['a', 'b', 'c', 'd']

'''acessing second element of list the indexing of list starts with 0 . Therefore element at index 1 is second elemnt of list '''
second_element=list[1]

print("The second element in the list :",second_element)
```

```
The second element in the list : b
```

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

list=[10, 20, 30, 40, 50]
print("Orginal List",list)

print()

#revrse inbuiltfunction
list.reverse()
print("Reversed list by reverse function ",list)
print()

orginal_list=[10, 20, 30, 40, 50]

print("Reversed list by slicing",orginal_list[::-1])
```

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

Reversed list by reverse function  [50, 40, 30, 20, 10]

Reversed list by slicing [50, 40, 30, 20, 10]
```

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

tuple=(100, 200, 300 )

print("Tuple :",tuple)
```

```
Tuple : (100, 200, 300)
```

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

tuple=('red', 'green', 'blue', 'yellow')

'''acessing second-to-last element of tuple the indexing of list starts with -1 from last . Therefore element at index -2 access the se
print("Accessing second-to-last element of tuple : ",tuple[-2])
```

```
Accessing second-to-last element of tuple :  blue
```

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

tuple=(10, 20, 5, 15)
print("Minimum value in the tuple :",min(tuple)) # min inbuilt function
```

```
Minimum value in the tuple : 5
```

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

tuple=('dog', 'cat', 'rabbit')
print("index of the cat :",tuple.index("cat")) #index fetch the index of elements
```

```
index of the cat : 1
```

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

tuple1 = ("banana" , "peaches", "apple")
tuple2= ("buleberry" , "kiwi", "grapes" )

result1= "kiwi" in tuple1
result2= "kiwi" in tuple2

print("Tuple1 contain kiwi :",result1)
print("Tuple2 contain kiwi :",result2)
```

```
Tuple1 contain kiwi : False
Tuple2 contain kiwi : True
```

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

set={"a","b","c"}

print("Set",set)
```

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

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

set={1, 2, 3, 4, 5}
print("Before clearing the set: " ,set)

#clear it empty the set
set=set.clear()
print("After clearing the set: ",set)
```

```
Before clearing the set:  {1, 2, 3, 4, 5}
After clearing the set:  None
```

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

my_set = {1, 2, 3, 4}
print("Before removing 4 from the set:", my_set)

# remove the element
my_set.remove(4)

print("After removing 4 from the set:", my_set)
```

```
Before removing 4 from the set: {1, 2, 3, 4}
After removing 4 from the 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}

#using union function  returns unquie element of set
```

```
print("Union of set with union fuction :",set1.union(set2))

# union with "| "
print("Union of set with '|':",set1|set2)
```

```
Union of set with union fuction : {1, 2, 3, 4, 5}
Union of set with '|': {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}

#using intersection function  returns common element of set
print("Intersection of set with intersection function :",set1.intersection(set2))

# Intersection with "& "
print("Intersection of set with '&' :",set1&set2)
```

```
Intersection of set with intersection function : {2, 3}
Intersection of set with '&' : {2, 3}
```

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

my_dictionary={"name":"Sakshi","age":"21","city":"nagpur"}
print("Dictionary :",my_dictionary)
```

```
Dictionary : {'name': 'Sakshi', 'age': '21', 'city': 'nagpur'}
```

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

my_dictionary={'name': 'John', 'age': 25}
print("Dictionary before adding new key-value pair :",my_dictionary)
my_dictionary["country"] = "USA" #adding element
print("Dictionary after adding new key-value pair :",my_dictionary)
```

```
Dictionary before adding new key-value pair : {'name': 'John', 'age': 25}
Dictionary after adding new key-value pair : {'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_dictionary={'name': 'Alice', 'age': 30}
print("The value associated with keyword name :",my_dictionary.get("name")) #accessing the value associated with the key "name"
```

```
The value associated with keyword name : Alice
```

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

my_dict={'name': 'Bob', 'age': 22, 'city': 'New York'}
print("Before removing age :",my_dict)

# removing age
my_dict.pop("age")
print("After removing age :",my_dict)
```

```
Before removing age : {'name': 'Bob', 'age': 22, 'city': 'New York'}
After removing age : {'name': 'Bob', 'city': 'New York'}
```

```
#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.keys():
  print("Dictionary contain city")
else:
  print("False")
```

```
Dictionary contain city
```

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

my_list=[1,2,4,"hello",False,True]
my_tuple=(5,3,6)
my_dict={'name': 'Alice', 'city': 'Paris'}

print(f"my_list :{my_list} and type is {type(my_list)}")
print()
print(f"my_tuple :{my_tuple} and type is {type(my_tuple)}")
```

```
print()
print(f"my dict :{my dict} and type is {type(my dict)}")
```

```
my_list :[1, 2, 4, 'hello', False, True] and type is <class 'list'>

my_tuple :(5, 3, 6) and type is <class 'tuple'>

my_dict :{'name': 'Alice', 'city': 'Paris'} and type is <class 'dict'>
```

```
#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

# List with random 5 numbers
random_numbers = random.sample(range(1, 101), 5)
print("list before sorting :",random_numbers)

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


print("Sorted random numbers:", random_numbers)
```

```
list before sorting : [66, 74, 9, 55, 7]
Sorted random numbers: [7, 9, 55, 66, 74]
```

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

my_list = ["apple", "banana", "cherry", "date", "blueberry"]
print("Element at index 3:", my_list[3])
```

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

dict1={'name': 'Alice', 'city': 'Paris'}
dict2 ={'age':24,'gender':'Male'}

print("Dict 1 : ",dict1)
print("Dict 2 : ",dict2)
print()
# update function extend or combine the list
dict1.update(dict2)

print("Dictionary 1 after combing both dictionary :" , dict1)
```

```
Dict 1 :  {'name': 'Alice', 'city': 'Paris'}
Dict 2 :  {'age': 24, 'gender': 'Male'}

Dictionary 1 after combing both dictionary : {'name': 'Alice', 'city': 'Paris', 'age': 24, 'gender': 'Male'}
```

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

my_list=["Ajay","Samiksha","David","name" ,"name","Samiksha"]
s=set(my_list)

print("List :",my_list)
print()
print("List after converted into set",s)
```

```
List : ['Ajay', 'Samiksha', 'David', 'name', 'name', 'Samiksha']

List after converted into set {'Ajay', 'David', 'Samiksha', 'name'}
```