Name: Atharva Kesarkar
Class: AI&DS Batch – 1
Roll no: 17

Experiment 3

AIM: Demonstrate Basics of Python Data Structures (Tuple, Dictionary, and Set).

Tools: Anaconda Navigator (Jupyter notebook)

Objective:

- The objective of this experiment is to:

1. Understand the structure, properties, and uses of three core Python data structures: tuple, dictionary, and set.
2. Learn how to declare, access, update (where possible), and perform operations on these data structures.
3. Explore the differences between these data types in terms of mutability, ordering, and usage scenarios.

Theory:

- Python, being a high-level programming language, provides built-in data structures to handle various data types efficiently.
- Three of the most important and commonly used data structures are:

1. Tuple

➤ Definition:

- A tuple is an ordered, immutable collection of elements.
- It is similar to a list but cannot be modified (i.e., no item assignment, insertion, or deletion after creation).

➤ Properties:

- Elements are enclosed in parentheses ().
- It can contain heterogeneous data types (e.g., strings, numbers, booleans).
- Indexing and slicing are allowed.
- Faster than lists due to immutability.

➤ Example:

```
my_tuple = (10, 20, 'Python', 3.14)
print(my_tuple[2])  # Accessing an element
```

2. Dictionary:

➤ Definition:

- A dictionary is an unordered, mutable collection that stores data as key-value pairs.
- It is similar to maps in other languages.

➤ Properties:

- Defined using curly braces {}.
- Each entry is of the form key: value.
- Keys must be unique and immutable.
- Values can be of any type and can be duplicated.

➤ Use Cases:

- Representing real-world entities (e.g., students, products).
- Efficient data lookup using keys.
- JSON-like structures in APIs.

➤ Example:

```
student = {'name': 'Alice', 'age': 21, 'course': 'AI'}

print(student['name'])  # Accessing value using key

student['age'] = 22     # Updating a value
```

3. Set

➤ Definition:

- A set is an unordered collection of unique elements.
- It is useful for storing non-duplicate items and performing set operations.

➤ Properties:

- Defined using curly braces {} or the set() constructor.
- No duplicates allowed.
- No indexing or slicing because sets are unordered.
- Supports mathematical operations like union, intersection, difference.

➤ Use Cases:

- Removing duplicates from a list.
- Membership testing.
- Performing mathematical set operations.

➤ Example:

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

print(numbers)  # Output will be {1, 2, 3, 4}

numbers.add(5)  # Adding an element

numbers.remove(2)  # Removing an element
```

4. Frozenset:

➤ Definition:

- A frozenset is an immutable version of a set.
- Once created, its elements cannot be added, removed, or changed.

➤ Key Features:

- Created using the frozenset() function.
- Supports all read-only set operations (union, intersection, difference).
- Cannot be updated or modified.

➤ Example:

normal_set = {1, 2, 3, 4}

frozen = frozenset(normal_set)

print("Original Frozenset:", frozen)

Code:

```
# Tuple demonstration
print("=== Tuple Demo ===")

# round brackets
mytuple = (1,2,3,4,2,3)
print(mytuple)

# with one item
mytuple = ("Geeks",)
print(type(mytuple))

#not a tuple
mytuple = ("Geeks")
print(type(mytuple))

# tuple constructor
tuple_constructor = (("dsa","development","deep learnig"))
print(tuple_constructor)
```

Output:

```
# Tuple demonstration
print("=== Tuple Demo ===")
# round brackets
mytuple = (1,2,3,4,2,3)
print(mytuple)
# with one item
mytuple = ("Geeks",)
print(type(mytuple))
#not a tuple
mytuple = ("Geeks")
print(type(mytuple))
# tuple constructor
tuple_constructor = (("dsa","development","deep learnig"))
print(tuple_constructor)
```

```
=== Tuple Demo ===
(1, 2, 3, 4, 2, 3)
<class 'tuple'>
<class 'str'>
('dsa', 'development', 'deep learnig')
```

# Dictionary with Integer Keys
Dict = {1: 'Geeks', 2: 'For', 3: 'Geeks'}
print("\nDictionary with the use of Integer Keys: ")
print(Dict)
# Dictionary with Mixed Keys
Dict = {'Name': 'Geeks', 1: [1, 2, 3, 4]}
print("\nDictionary with the use of Mixed Keys: ")
print(Dict)

Output:

```
# Dictionary with Integer Keys
Dict = {1: 'Geeks', 2: 'For', 3: 'Geeks'}
print("\nDictionary with the use of Integer Keys: ")
print(Dict)

# Dictionary with Mixed Keys
Dict = {'Name': 'Geeks', 1: [1, 2, 3, 4]}
print("\nDictionary with the use of Mixed Keys: ")
print(Dict)
```

```
Dictionary with the use of Integer Keys:
{1: 'Geeks', 2: 'For', 3: 'Geeks'}

Dictionary with the use of Mixed Keys:
{'Name': 'Geeks', 1: [1, 2, 3, 4]}
```

```
# Set demonstration
# Typecasting list to set (duplicates will be removed)
myset = set(["a", "b", "b", "c"])
print(myset)  # Output: {'a', 'b', 'c'}

# Adding an element to the set
myset.add("d")
print(myset)  # Output: {'a', 'b', 'c', 'd'}

# Creating a set with some duplicate values (only unique values are stored)
fruits = {"Apple", "Banana", "Cherry", "Apple", "Kiwi"}
print('Unique elements:', fruits)  # Duplicates like "Apple" will appear only once

# Add new fruit to the set
fruits.add("Orange")
print('After adding new element:', fruits)

# Size of the set
print('Size of the set:', len(fruits))

# Check if the element is present in the set
print('Apple is present in the set:', "Apple" in fruits)
print('Mango is present in the set:', "Mango" in fruits)

# Safely remove the element from the set using discard() to avoid KeyError
fruits.discard("Mango")  # Will not raise an error even if "Mango" is not in the set
print('After discarding Mango (safe remove):', fruits)

# Optional: If you want to use remove(), check before removing
if "Mango" in fruits:
    fruits.remove("Mango")  # Safe way to use remove()
    print('After removing Mango:', fruits)
else:
    print("Mango not found, so not removed.")
```

Output:

```
# Typecasting list to set (duplicates will be removed)
myset = set(["a", "b", "b", "c"])
print(myset)  # Output: {'a', 'b', 'c'}

# Adding an element to the set
myset.add("d")
print(myset)  # Output: {'a', 'b', 'c', 'd'}

# Creating a set with some duplicate values (only unique values are stored)
fruits = {"Apple", "Banana", "Cherry", "Apple", "Kiwi"}
print('Unique elements:', fruits)  # Duplicates like "Apple" will appear only once

# Add new fruit to the set
fruits.add("Orange")
print('After adding new element:', fruits)

# Size of the set
print('Size of the set:', len(fruits))

# Check if the element is present in the set
print('Apple is present in the set:', "Apple" in fruits)
print('Mango is present in the set:', "Mango" in fruits)

# Safely remove the element from the set using discard() to avoid KeyError
fruits.discard("Mango")  # Will not raise an error even if "Mango" is not in the set
print('After discarding Mango (safe remove):', fruits)

# Optional: If you want to use remove(), check before removing
if "Mango" in fruits:
    fruits.remove("Mango")  # Safe way to use remove()
    print('After removing Mango:', fruits)
else:
    print("Mango not found, so not removed.")
```

```
{'a', 'c', 'b'}
{'d', 'a', 'c', 'b'}
Unique elements: {'Apple', 'Cherry', 'Kiwi', 'Banana'}
After adding new element: {'Apple', 'Cherry', 'Kiwi', 'Banana', 'Orange'}
Size of the set: 5
Apple is present in the set: True
Mango is present in the set: False
After discarding Mango (safe remove): {'Apple', 'Cherry', 'Kiwi', 'Banana', 'Orange'}
Mango not found, so not removed.
```

```
# Frozenset demonstration
print("\n=== Frozenset Demo ===")
# Unique Elements
fruits = {"Apple","Banana","Cherry","Apple","Kiwi"}
basket = frozenset(fruits)
print("Unique Elements: ",basket)
# keys
student = {"Name": "John", "Age": "25", "Gender": "Male"}
key = frozenset(student)
print("The keys are:", key)
#intialise A and B
A = frozenset([1,2,3,4])
```

```
B = frozenset([3,4,5,6])
# copy()
C = A.copy()
print(C)
# union()
union_set = A.union(B)
print(union_set)
#intersection()
intersection_set = A.intersection(B)
print(intersection_set)
#difference()
difference_set = A.difference(B)
print(difference_set)
# symmetric_difference()
symmetric_difference_set = A.symmetric_difference(B)
print(symmetric_difference_set)
```

Output:

```
# Frozenset demonstration
print("\n=== Frozenset Demo ===")
# Unique Elements
fruits = {"Apple","Banana","Cherry","Apple","Kiwi"}
basket = frozenset(fruits)
print("Unique Elements: ",basket)
# keys
student = {"Name": "John", "Age": "25", "Gender": "Male"}
key = frozenset(student)
print("The keys are:", key)
#intialise A and B
A = frozenset([1,2,3,4])
B = frozenset([3,4,5,6])
# copy()
C = A.copy()
print(C)
# union()
union_set = A.union(B)
print(union_set)
#intersection()
intersection_set = A.intersection(B)
print(intersection_set)
#difference()
difference_set = A.difference(B)
print(difference_set)
# symmetric_difference()
symmetric_difference_set = A.symmetric_difference(B)
print(symmetric_difference_set)
```

```
=== Frozenset Demo ===
Unique Elements:  frozenset({'Apple', 'Cherry', 'Kiwi', 'Banana'})
The keys are: frozenset({'Name', 'Age', 'Gender'})
frozenset({1, 2, 3, 4})
frozenset({1, 2, 3, 4, 5, 6})
frozenset({3, 4})
frozenset({1, 2})
frozenset({1, 2, 5, 6})
```

Conclusion:

We explored Python's core data structures: tuples (immutable, ordered), dictionaries (key-value, mutable), sets (unique, unordered), and frozensets (immutable sets). These structures are essential for efficient data handling and logical program design.

For Faculty use only:

Correction Parameters	Formative Assessment [40%]	Timely completion of practical [40%]	Attendance/ Learning Attitude [20%]	
Marks Obtained				