Day 12- Trainer FG
(Set)

Index

(1.50 hrs) ILT

I. Recap

In our last session we learned:

- **What is a Tuple?:** A tuple is an ordered and immutable collection or sequence of elements in Python. Immutable means that once a tuple is created, its elements cannot be modified, added, or removed. Tuples are typically enclosed in parentheses (), although the parentheses are optional in many cases.
- **Features of Tuples:** Tuples in Python have several features that make them a useful data structure in various programming scenarios.
- **Creating Tuples:** Tuples in Python can be created in several ways,Using Parentheses,Using the tuple() Constructor,Creating a Singleton Tuple.
- **Accessing Values in Tuples:** You can access values in tuples by using indexing, slicing, or iterating through the tuple.
- **Various Operations of Tuples:** Tuples in Python support various operations and methods that allow you to work with them effectively.
- **Tuple Functions:** Tuples in Python don't have many built-in methods since they are immutable, but they support a few useful functions and methods for various operations.
- **Indexing and Slicing in Tuple:** In Python, you can access elements in a tuple using indexing and slicing, just like you would with lists and other sequences.

Try this:
Predict the output

```
tuple1 = (10, 20, 30, 40, 50)
tuple1 = tuple1[::-1]
print(tuple1)
```

In this session we are going to understand Set, features of Set, creating sets, and various operations of sets.

II. What is a Set?

A set in Python is an unordered collection of unique elements. It is a built-in data type that is used to store a collection of values, with the key characteristic being that each element in a set must be unique. In other words, a set does not allow duplicate values.

Features of Set

Sets in Python have several distinctive features that make them useful for specific tasks. Here are the key features of sets:

- **Unordered Collection:** Sets are unordered collections of elements. Unlike lists or tuples, the elements in a set have no specific order, and you cannot rely on their position.

- **Unique Elements:** Sets only allow unique elements. If you attempt to add a duplicate element to a set, it will not be stored, ensuring that each element in a set is unique.

- **Mutable:** Sets are mutable, meaning you can add and remove elements from a set after it is created.

- **No Duplicate Values:** Sets automatically remove duplicate values, making them useful for removing duplicates from other collections, such as lists.

- **Mathematical Set Operations:** Sets support various mathematical operations such as union, intersection, difference, and symmetric difference. These operations can be performed efficiently using set methods.

- **Defined Using Curly Braces or the set() Constructor:** You can create a set using curly braces {} or by using the set() constructor function.

- **No Indexing:** Sets do not support indexing, so you cannot access elements by their position in the set. However, you can check for membership and iterate through the elements.

- **Hashable Elements:** Elements in a set must be hashable. This means that they should have a hash value that remains constant during their lifetime. Immutable data types like numbers, strings, and tuples are hashable and can be used in sets.

III. Creating sets

Sets in Python can be created in several ways. Here are some common methods for creating sets:

Using Curly Braces {}: You can create a set by enclosing a comma-separated sequence of values in curly braces {}. Duplicate values will be automatically removed.

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

Using the set() Constructor: You can create a set using the set() constructor and passing an iterable (e.g., a list or a tuple) as an argument. This is useful when you want to convert an existing iterable into a set.

```
my_list = [1, 2, 2, 3, 4]
my_set = set(my_list)
```

Creating an Empty Set: To create an empty set, you can use curly braces {} or the set() constructor with no arguments.

```
empty_set1 = set()
empty_set2 = {}
```

Converting Strings to Sets: You can convert a string into a set of its characters using the set() constructor or a set comprehension.

```
my_string = "hello"
char_set = set(my_string)  # Creates a set of characters: {'h', 'e', 'l', 'o'}
```

Remember that sets are unordered collections of unique elements, so the order of elements in a set is not guaranteed, and duplicates are automatically removed. Sets are often used when you need to work with unique values, remove duplicates, or perform set-related operations.

IV. Various Operations of sets

Sets in Python support various operations and methods that allow you to work with them effectively. Here are some common operations and methods associated with sets:

Adding Elements: You can add elements to a set using the add() method. This method adds a single element to the set.

```
my_set = {1, 2, 3}
my_set.add(4)  # Adds 4 to the set
```

Removing Elements: You can remove elements from a set using the remove() method. If the element is not in the set, it raises a KeyError. To avoid raising an error if the element is not present, you can use the discard() method.

```
my_set = {1, 2, 3}
my_set.remove(2)  # Removes 2 from the set
my_set.discard(4)  # Removes 4 if it exists, does nothing if not
```

Clearing a Set: You can clear all elements from a set using the clear() method.

```
my_set = {1, 2, 3}
my_set.clear()  # Clears all elements, resulting in an empty set
```

Union: You can perform the union of two sets using the union() method or the | operator. The union of sets contains all unique elements from both sets.

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

```
union_result = set1.union(set2)  # or set1 | set2
```

Intersection: You can perform the intersection of two sets using the intersection() method or the & operator. The intersection contains elements that are common to both sets.

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
intersection_result = set1.intersection(set2)  # or set1 & set2
```

Difference: You can find the difference between two sets using the difference() method or the - operator. The difference contains elements that are in the first set but not in the second.

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
difference_result = set1.difference(set2)  # or set1 - set2
```

Symmetric Difference: You can find the symmetric difference between two sets using the symmetric_difference() method or the ^ operator. The symmetric difference contains elements that are in either of the sets but not in both.

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
symmetric_difference_result = set1.symmetric_difference(set2)  # or set1 ^ set2
```

Subset and Superset: You can check if one set is a subset or superset of another using the issubset() and issuperset() methods.

```
set1 = {1, 2, 3}
set2 = {1, 2}
is_subset = set2.issubset(set1)  # True, set2 is a subset of set1
is_superset = set1.issuperset(set2)  # True, set1 is a superset of set2
```

These are some of the common operations and methods that you can perform with sets in Python. Sets are useful for tasks involving unique elements and set-related operations.

Common Use Cases

Sets are commonly used in scenarios where only unique values are allowed. t, checking for membership, performing set operations, and implementing algorithms that require uniqueness.

Here's an example that demonstrates some of these features:

```
# Creating a List with duplicate values
my_list = [1, 2, 3, 3, 4,1,2,1,3]
print("List Values Are")
print(my_list)
print("After Converting set  Values Are")
print(set(my_list))  # Output: {1, 2, 3, 4} by removing duplicate

#Creating a set
my_set={1, 2, 3, 3, 4}# Duplicate elements are automatically removed
print("Before Adding Any Elements")
print(my_set)
# Adding elements to a set
my_set.add(5)  # Adding a new element
print(my_set)  # Output: {1, 2, 3, 4, 5}
print("After  Adding Elements")
print(my_set)

print("Before Removing  Elements")
print(my_set)
# Removing elements from a set
my_set.remove(3)  # Removing an element
print(my_set)  # Output: {1, 2, 4, 5}
print("After  Removing  Elements")
print(my_set)

# Set operations
set1 = {1, 2, 3}
set2 = {3, 4, 5}
union_result = set1.union(set2)  # Union  display all values by merging both set
print("After Union")
```

```
print("Set 1=",set1)
print("Set 2=",set2)
print("Display All unique  values From both set", union_result)  # Output: {1, 2, 3, 4, 5}

intersection_result = set1.intersection(set2) # Intersection display common  values
print("After Intersection")
print("Set 1=",set1)
print("Set 2=",set2)
print("Display  only common  values From both set",intersection_result)  # Output: {3}

difference_result = set1.difference(set2)  # Difference display  elements from the first set that are not present in the second set
print("After Difference")
print("Set 1=",set1)
print("Set 2=",set2)
print("elements from the first set that are not present in the second set",difference_result)  # Output: {1, 2}
```

Output:

```
List Values Are
[1, 2, 3, 3, 4, 1, 2, 1, 3]
After Converting set  Values Are
{1, 2, 3, 4}
Before Adding Any Elements
{1, 2, 3, 4}
{1, 2, 3, 4, 5}
After  Adding Elements
{1, 2, 3, 4, 5}
Before Removing  Elements
{1, 2, 3, 4, 5}
{1, 2, 4, 5}
After  Removing  Elements
{1, 2, 4, 5}
After Union
Set 1= {1, 2, 3}
Set 2= {3, 4, 5}
Display All unique  values Form both set {1, 2, 3, 4, 5}
After Intersection
Set 1= {1, 2, 3}
Set 2= {3, 4, 5}
Display  only common  values Form both set {3}
After Difference
Set 1= {1, 2, 3}
Set 2= {3, 4, 5}
elements from the first set that are not present in the second set {1, 2}
```

These features make sets a valuable data structure in Python for various tasks that involve working with unique elements and performing set-related operations.

Exercise

Use GPT to write a program:

1.Hi! Can you create a program Finding mutual friends between two sets of friends?

After the ILT, the student has to do a 30 min live lab. Please refer to the Day 12 Lab doc in the LMS.