Session on Python Sets

Introduction to Python Tuples

Python tuples are an ordered collection of elements that are immutable. Once created, the elements of a tuple cannot be changed, making them useful for fixed collections of items.

Creating Tuples

1. Empty Tuple

my_tuple = ()

2. Tuple with Initial Values

my_tuple = (1, 2, 3, 4, 5)

3. Without Parentheses

my_tuple = 1, 2, 3, 4, 5

4. Single Element Tuple

my_tuple = (1,)

5. Using the `tuple()` Constructor

my_tuple = tuple([1, 2, 3, 4, 5])

Tuple Functions and Methods

1. Accessing Elements

first_element = my_tuple[0] # 1

Session on Python Sets

2. Slicing Tuples

sub_tuple = my_tuple[1:3] # (2, 3)

3. Tuple Unpacking

a, b, c, d, e = my_tuple

4. Using `count()` Method

count_of_2 = my_tuple.count(2) # 1

5. Using `index()` Method

index_of_3 = my_tuple.index(3) # 2

Uniqueness of Python Tuples

- Ordered: Items have a defined order.
- Immutable: Once created, the elements cannot be changed.
- Allows Duplicate Elements: Tuples can have items with the same value.
- Fixed Size: The size of the tuple is fixed once created.

Key Concepts in Detail

Creating a Tuple

Tuples can be created using parentheses `()` or without parentheses by separating values with commas. Single element tuples require a trailing comma.

Session on Python Sets

Example:

```
my_tuple = (1, 2, 3, 4, 5)
```

Accessing Elements

Elements in a tuple can be accessed using indexing. Negative indexing is also supported.

Example:

```
first_element = my_tuple[0]  # 1
last_element = my_tuple[-1]  # 5
```

Slicing Tuples

Slicing allows you to access a subset of a tuple by specifying the start and end indices.

Example:

```
sub_tuple = my_tuple[1:4]  # (2, 3, 4)
```

Tuple Unpacking

Tuple unpacking allows you to assign each element of a tuple to a variable.

Example:

```
a, b, c, d, e = my_tuple
```

```
print(a, b, c, d, e)  # 1 2 3 4 5
```

Tuple Methods

Tuples have two built-in methods: `count()` and `index()`.

Example:

```
count_of_2 = my_tuple.count(2)  # 1
index_of_3 = my_tuple.index(3)  # 2
```

Iterating Over a Tuple

You can iterate over a tuple using a `for` loop.

Example:

```
for item in my_tuple:
    print(item)
```

Tuple Comprehension

Tuples do not support comprehension directly, but you can create a tuple from a generator expression.

Example:

```
squares = tuple(x**2 for x in range(6))  # (0, 1, 4, 9, 16, 25)
```

Session on Python Sets

Memory Management

Tuples are more memory efficient than lists due to their immutability and fixed size.

Example:

```
import sys

my_tuple = (1, 2, 3, 4, 5)
print(sys.getsizeof(my_tuple))  # Returns the memory size of the tuple

# Copying a tuple
my_tuple_copy = my_tuple  # Tuples are immutable, so copy is the same as original
print(sys.getsizeof(my_tuple_copy))  # Same memory size as the original tuple
```

Comprehensive Example Incorporating All Concepts

```
# Creating a tuple with initial values
my_tuple = (1, 2, 3, 4, 5)

# Accessing elements
first_element = my_tuple[0]  # 1
last_element = my_tuple[-1]  # 5

# Slicing
```

```
sub_tuple = my_tuple[1:4]  # (2, 3, 4)


# Unpacking
a, b, c, d, e = my_tuple
print(a, b, c, d, e)  # 1 2 3 4 5


# Methods
count_of_2 = my_tuple.count(2)  # 1
index_of_3 = my_tuple.index(3)  # 2


# Iterating over a tuple
for item in my_tuple:
    print(item)


# Tuple comprehension (via generator expression)
squares = tuple(x**2 for x in range(6))  # (0, 1, 4, 9, 16, 25)


# Memory management
import sys
print("Original tuple memory size:", sys.getsizeof(my_tuple))
print("Squares tuple memory size:", sys.getsizeof(squares))
```

Examples of Tuple Operations

Example 1: Finding Maximum and Minimum Elements (LeetCode Style)

Session on Python Sets

```
def find_max_min(nums):
    return max(nums), min(nums)


# Test
print(find_max_min((1, 2, 3, 4, 5)))
# Output: (5, 1)
```

Example 2: Merging Two Tuples (LeetCode Style)

```
def merge_tuples(tuple1, tuple2):
    return tuple1 + tuple2


# Test
print(merge_tuples((1, 2), (3, 4, 5)))
# Output: (1, 2, 3, 4, 5)
```