Session on Python Sets

Introduction to Python Sets

Python sets are an unordered collection of unique elements. They are mutable, allowing you to add or remove elements, and they provide high performance for membership tests due to their underlying hash table implementation.

Creating Sets

1. Empty Set

```
my_set = set()
```

2. Set with Initial Values

```
my_set = {1, 2, 3, 4, 5}
```

3. Using the `set()` Constructor

```
my_set = set([1, 2, 3, 4, 5])
```

4. Set Comprehension

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

Set Functions and Methods

1. Adding Items

```
my_set.add(6)  # {1, 2, 3, 4, 5, 6}
```

2. Removing Items

```
my_set.remove(3)  # {1, 2, 4, 5, 6}
```

Session on Python Sets

3. Discarding Items (No Error if Not Found)

```
my_set.discard(7)  # {1, 2, 4, 5, 6}
```

4. Popping Items

```
item = my_set.pop()  # Removes and returns an arbitrary element
```

5. Checking for Membership

```
if 2 in my_set:
    print("2 is in the set")
```

6. Iterating Over a Set

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

7. Set Operations

```
union_set = my_set.union({7, 8})  # {1, 2, 4, 5, 6, 7, 8}

intersection_set = my_set.intersection({4, 5, 6, 7})  # {4, 5, 6}

difference_set = my_set.difference({4, 5})  # {1, 2, 6}

symmetric_difference_set = my_set.symmetric_difference({4, 5, 6, 7})  # {1, 2, 7}
```

Uniqueness of Python Sets

- Unordered: Items do not have a defined order.

- Mutable: You can change the content of a set after it has been created.

- Unique Elements: Each element is unique within a set.

- Dynamic Size: Sets can grow and shrink as needed.

Key Concepts in Detail

Creating a Set

Sets can be created using curly braces `{}` or the `set()` constructor. They can hold elements of different data types but only store unique elements.

Example:

```
my_set = {1, 2, 3, 4, 5}
```

Adding and Removing Items

You can add new elements using the `add()` method and remove elements using the `remove()` or `discard()` method. The `pop()` method removes and returns an arbitrary element.

Example:

```
my_set.add(6)
my_set.remove(3)
my_set.discard(7)  # No error if 7 is not found
```

Set Operations

Session on Python Sets

Sets support several mathematical operations such as union, intersection, difference, and symmetric difference.

Example:

```
set_a = {1, 2, 3}

set_b = {3, 4, 5}


# Union

union_set = set_a.union(set_b)  # {1, 2, 3, 4, 5}


# Intersection

intersection_set = set_a.intersection(set_b)  # {3}


# Difference

difference_set = set_a.difference(set_b)  # {1, 2}


# Symmetric Difference

symmetric_difference_set = set_a.symmetric_difference(set_b)  # {1, 2, 4, 5}
```

Iterating Over a Set

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

Example:

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

Set Comprehension

Set comprehensions provide a concise way to create sets from iterables.

Example:

```
squares = {x**2 for x in range(6)}
```

Memory Management

Understanding memory management is crucial when working with large sets. Sets are implemented as hash tables, providing average O(1) time complexity for lookups, insertions, and deletions.

Example:

```
import sys

my_set = {1, 2, 3, 4, 5}
print(sys.getsizeof(my_set))  # Returns the memory size of the set

# Copying a set
my_set_copy = my_set.copy()  # Creates a new set with the same elements
print(sys.getsizeof(my_set_copy))  # Same memory size as the original set
```

Session on Python Sets

Comprehensive Example Incorporating All Concepts

```
# Creating a set with initial values
my_set = {1, 2, 3, 4, 5}

# Adding and removing items
my_set.add(6)
my_set.remove(3)
my_set.discard(7)  # No error if 7 is not found

# Set operations
set_a = {1, 2, 3}
set_b = {3, 4, 5}

# Union
union_set = set_a.union(set_b)  # {1, 2, 3, 4, 5}

# Intersection
intersection_set = set_a.intersection(set_b)  # {3}

# Difference
difference_set = set_a.difference(set_b)  # {1, 2}

# Symmetric Difference
symmetric_difference_set = set_a.symmetric_difference(set_b)  # {1, 2, 4, 5}
```

Session on Python Sets

```
# Iterating over a set
for item in my_set:
    print(item)


# Set comprehension
squares = {x**2 for x in range(6)}


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

Examples of Set Operations

Example 1: Finding Duplicate Elements (LeetCode Style)

```
def find_duplicates(nums):
    seen = set()
    duplicates = set()
    for num in nums:
        if num in seen:
            duplicates.add(num)
        else:
            seen.add(num)
    return list(duplicates)
```

```
# Test

print(find_duplicates([1, 2, 3, 1, 2, 4]))

# Output: [1, 2]
```

Example 2: Checking for Disjoint Sets (LeetCode Style)

```
def are_disjoint(set1, set2):

    return set1.isdisjoint(set2)


# Test

print(are_disjoint({1, 2, 3}, {4, 5, 6}))  # True

print(are_disjoint({1, 2, 3}, {3, 4, 5}))  # False
```