Session on Python Lists

Introduction to Python Dictionaries

Python dictionaries are a collection of key-value pairs. Each key is unique and maps to a value. Dictionaries are unordered, mutable, and dynamic, making them highly versatile for various applications.

Creating Dictionaries

1. Empty Dictionary

```
my_dict = {}
```

2. Dictionary with Initial Values

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
```

3. Using the `dict()` Constructor

```
my_dict = dict(a=1, b=2, c=3)
```

4. Dictionary Comprehension

```
squares = {x: x**2 for x in range(1, 6)}  # {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

Dictionary Functions and Methods

1. Adding Items

```
my_dict['d'] = 4  # {'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

2. Removing Items

```
del my_dict['a']  # {'b': 2, 'c': 3, 'd': 4}
```

Session on Python Lists

3. Accessing Values

```
value = my_dict['b']  # 2
```

4. Using `get()` Method

```
value = my_dict.get('b')  # 2
```

5. Checking for Keys

```
if 'b' in my_dict:
    print("Key 'b' is present")
```

6. Iterating Over Dictionary

```
for key in my_dict:
    print(key, my_dict[key])
```

7. Dictionary Methods

```
keys = my_dict.keys()  # dict_keys(['b', 'c', 'd'])
values = my_dict.values()  # dict_values([2, 3, 4])
items = my_dict.items()  # dict_items([('b', 2), ('c', 3), ('d', 4)])
my_dict.clear()  # {}
```

Uniqueness of Python Dictionaries

- Unordered: Items do not have a defined order.

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

- Unique Keys: Each key is unique within a dictionary.
- Dynamic Size: Dictionaries can grow and shrink as needed.

Key Concepts in Detail

Creating a Dictionary

Dictionaries can be created using curly braces `{}` or the `dict()` constructor. Keys must be unique and immutable, such as strings, numbers, or tuples.

Example:

```
my_dict = {'name': 'Alice', 'age': 25, 'city': 'New York'}
```

Accessing Values

Values can be accessed using the key. If the key does not exist, a `KeyError` is raised. The `get()` method is safer as it returns `None` or a default value if the key is not found.

Example:

```
name = my_dict['name']  # 'Alice'
age = my_dict.get('age')  # 25
country = my_dict.get('country', 'Unknown')  # 'Unknown'
```

Adding and Modifying Items

Session on Python Lists

You can add new key-value pairs or update existing ones by assigning a value to a key.

Example:

my_dict['email'] = 'alice@example.com'

my_dict['age'] = 26

Removing Items

Items can be removed using the `del` statement or the `pop()` method, which returns the value of the removed key.

Example:

del my_dict['city']

email = my_dict.pop('email', 'No Email')

Iterating Over a Dictionary

You can iterate over keys, values, or key-value pairs using a `for` loop.

Example:

```
for key in my_dict:
    print(key, my_dict[key])

for key, value in my_dict.items():
    print(key, value)
```

Session on Python Lists

Dictionary Comprehension

Dictionary comprehensions provide a concise way to create dictionaries from iterables.

Example:

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

Memory Management

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

Example:

```
import sys

my_dict = {'a': 1, 'b': 2, 'c': 3}
print(sys.getsizeof(my_dict))  # Returns the memory size of the dictionary

# Copying a dictionary
my_dict_copy = my_dict.copy()  # Creates a new dictionary with the same key-value pairs
print(sys.getsizeof(my_dict_copy))  # Same memory size as the original dictionary
```

Session on Python Lists

Comprehensive Example Incorporating All Concepts

```
# Creating a dictionary with initial values
my_dict = {'name': 'Alice', 'age': 25, 'city': 'New York'}

# Accessing values
name = my_dict['name']  # 'Alice'
age = my_dict.get('age')  # 25

# Adding and modifying items
my_dict['email'] = 'alice@example.com'
my_dict['age'] = 26

# Removing items
del my_dict['city']
email = my_dict.pop('email', 'No Email')

# Iterating over a dictionary
for key in my_dict:
    print(key, my_dict[key])

for key, value in my_dict.items():
    print(key, value)

# Dictionary comprehension
squares = {x: x**2 for x in range(6)}
```

Session on Python Lists

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

Examples of Dictionary Operations

Example 1: Grouping Anagrams (LeetCode Style)

```
def group_anagrams(words):
    anagrams = {}
    for word in words:
        sorted_word = ''.join(sorted(word))
        if sorted_word not in anagrams:
            anagrams[sorted_word] = [word]
        else:
            anagrams[sorted_word].append(word)
    return list(anagrams.values())

# Test
print(group_anagrams(['eat', 'tea', 'tan', 'ate', 'nat', 'bat']))
# Output: [['eat', 'tea', 'ate'], ['tan', 'nat'], ['bat']]
```

Example 2: Top K Frequent Elements (LeetCode Style)

```
from collections import Counter
```

Session on Python Lists

```
def top_k_frequent(nums, k):
    count = Counter(nums)
    return [item for item, _ in count.most_common(k)]


# Test
print(top_k_frequent([1, 1, 1, 2, 2, 3], 2))
# Output: [1, 2]
```