Module 4: Advanced Dictionary Operations

Introduction

Dictionaries in Python are powerful data structures that allow for efficient storage and retrieval of key-value pairs. In this module, we will explore advanced dictionary operations and techniques to manipulate dictionaries effectively.

Dictionary Comprehensions

Dictionary comprehensions provide a concise way to create dictionaries in Python. They follow a similar syntax to list comprehensions but produce dictionaries instead.

```
# Example 1: Creating a dictionary of squares
squares = {x: x**2 for x in range(1, 6)}
print(squares)

# Output: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

Conditional expressions can also be used in dictionary comprehensions to filter elements.

```
# Example 2: Creating a dictionary of even squares
even_squares = {x: x**2 for x in range(1, 11) if x % 2 == 0}
print(even_squares)

# Output: {2: 4, 4: 16, 6: 36, 8: 64, 10: 100}
```

Dictionary Methods

Python provides several built-in methods for manipulating dictionaries:

- `keys()`: Returns a view object containing the keys of the dictionary.
- `values()`: Returns a view object containing the values of the dictionary.
- `items()`: Returns a view object containing the key-value pairs of the dictionary.

- `get()`: Returns the value associated with a specified key. Allows specifying a default value if the key is not found.
- `pop()`: Removes and returns the value associated with a specified key.
- `update()`: Updates the dictionary with key-value pairs from another dictionary or iterable.

```
# Example 3: Using dictionary methods
my_dict = {"a": 1, "b": 2, "c": 3}

# Using keys(), values(), and items()
print(my_dict.keys()) # Output: dict_keys(['a', 'b', 'c'])
print(my_dict.values()) # Output: dict_values([1, 2, 3])
print(my_dict.items()) # Output: dict_items([('a', 1), ('b', 2), ('c', 3)])

# Using get() and pop()
print(my_dict.get("b")) # Output: 2
print(my_dict.pop("c")) # Output: 3
print(my_dict) # Output: {'a': 1, 'b': 2}

# Using update()
my_dict.update({"d": 4})
print(my_dict) # Output: {'a': 1, 'b': 2, 'd': 4}
```

Merging Dictionaries

There are multiple ways to merge dictionaries in Python, including using the `update()` method, unpacking (`**`), and dictionary comprehension.

```
# Example 4: Merging dictionaries
dict1 = {"a": 1, "b": 2}
dict2 = {"b": 3, "c": 4}

# Using update()
dict1.update(dict2)
print(dict1) # Output: {'a': 1, 'b': 3, 'c': 4}

# Using unpacking (**)
merged_dict = {**dict1, **dict2}
print(merged_dict) # Output: {'a': 1, 'b': 3, 'c': 4}

# Using dictionary comprehension
merged_dict = {key: value for d in [dict1, dict2] for key, value in d.items()}
print(merged_dict) # Output: {'a': 1, 'b': 3, 'c': 4}
```

Nested Dictionaries

Dictionaries can be nested within each other to represent structured data. This is useful for organizing hierarchical data.

```
# Example 5: Nested dictionaries
student = {
 "name": "Alice",
 "age": 20,
 "grades": {"math": 90, "science": 85}
}

# Accessing nested values
print(student["name"]) # Output: Alice
print(student["grades"]["math"]) # Output: 90
```

Default Values

The `get()` method can be used to retrieve the value associated with a key, with an option to specify a default value if the key is not found.

```
# Example 6: Using default values with get()
my_dict = {"a": 1, "b": 2}

print(my_dict.get("a", "Key not found")) # Output: 1
print(my_dict.get("c", "Key not found")) # Output: Key not found
```

Alternatively, the `defaultdict` class from the `collections` module can be used to create dictionaries with default values for missing keys.

```
from collections import defaultdict

# Example 7: Using defaultdict
my_dict = defaultdict(int) # Default value for missing keys is 0
my_dict["a"] = 1

print(my_dict["a"]) # Output: 1
print(my_dict["b"]) # Output: 0 (default value)
```

Conclusion

Advanced dictionary operations in Python provide powerful tools for working with dictionaries efficiently. By mastering these techniques, you can manipulate and manage dictionaries effectively, making them a valuable asset in various programming tasks.

Exercises & Answers for Advanced Dictionary Operations

Exercise 1: Dictionary Comprehensions

Create a dictionary `square_dict` that contains the squares of numbers from 1 to 10 as key-value pairs using dictionary comprehension.

Answer 1:

```
square_dict = {x: x**2 for x in range(1, 11)}
print(square_dict)
```

Exercise 2: Dictionary Methods

Given the dictionary `student_scores`, use dictionary methods to:

- Print the keys of the dictionary.
- Print the values of the dictionary.
- Print the key-value pairs of the dictionary.

```
student_scores = {"Alice": 85, "Bob": 90, "Charlie": 75}

# Print the keys
print("Keys:", student_scores.keys())

# Print the values
print("Values:", student_scores.values())

# Print the key-value pairs
print("Key-Value Pairs:", student_scores.items())
```

Exercise 3: Merging Dictionaries

Merge the dictionaries `dict1` and `dict2` into a new dictionary `merged_dict` using dictionary comprehension.

```
dict1 = {"a": 1, "b": 2}
dict2 = {"b": 3, "c": 4}

merged_dict = {key: value for d in [dict1, dict2] for key, value in d.items()}
print(merged_dict)
```

Exercise 4: Nested Dictionaries

Create a nested dictionary `student_info` to store information about a student. Include keys for `"name"`, `"age"`, and `"grades"`. The `"grades"` key should contain another dictionary with subjects as keys and corresponding grades as values.

Answer 4:

```
student_info = {
 "name": "Alice",
 "age": 20,
 "grades": {"math": 90, "science": 85}
}
print(student_info)
```

Exercise 5: Default Values

Given the dictionary `fruit_counts`, retrieve the count of `"apple"`. If the key is not present, return 0 as the default value.

```
from collections import defaultdict

fruit_counts = {"apple": 10, "banana": 5, "orange": 8}

apple_count = fruit_counts.get("apple", 0)
print("Apple Count:", apple_count)
```

These exercises cover various advanced dictionary operations in Python and provide opportunities to practice using dictionary comprehensions, methods, merging dictionaries, working with nested dictionaries, and handling default values.