Dictionaries

A dictionary is a data structure in Python that is used to store a collection of key-value pairs. The key in the dictionary is used to index the value, which can be any Python object. Dictionaries are mutable, which means you can modify their contents. They are also unordered, so the order in which items are added to a dictionary is not preserved.

Creating a Dictionary

In Python, dictionaries are defined using curly braces (`{}`). Each key-value pair is separated by a comma (`,`) and the key and value are separated by a colon (`:`). For example:

```
# create an empty dictionary
my_dict = {}

# create a dictionary with some key-value pairs
my_dict = {'apple': 1, 'banana': 2, 'orange': 3}
```

In the above example, the keys are strings (`'apple'`, `'banana'`, `'orange'`) and the respective values are integers (`1`, `2`, `3`). The keys in a dictionary must be unique. If you try to add a key that already exists, the value will be overwritten.

Accessing Values in a Dictionary

Values in a dictionary can be accessed using the key. For example:

```
# access a value using a key
my_dict = {'apple': 1, 'banana': 2, 'orange': 3}
print(my_dict['apple'])
```

If the key does not exist in the dictionary, a `KeyError` will be raised. To avoid this, you can use the `get()` method, which returns `None` if the key is not found in the dictionary:

```
# access a value using the get method
my_dict = {'apple': 1, 'banana': 2, 'orange': 3}
print(my_dict.get('apple'))
print(my_dict.get('grape'))
```

Adding and Modifying Key-Value Pairs in a Dictionary

You can add new key-value pairs to a dictionary by simply assigning a value to a new key. For example:

```
# add a new key-value pair
my_dict = {'apple': 1, 'banana': 2, 'orange': 3}
my_dict['grape'] = 4
print(my_dict)
```

You can also modify the value of an existing key by assigning a new value to it. For example:

```
# modify an existing key-value pair
my_dict = {'apple': 1, 'banana': 2, 'orange': 3}
my_dict['apple'] = 5
print(my_dict)
```

Removing Key-Value Pairs from a Dictionary

To remove a key-value pair from a dictionary, you can use the `del` keyword or the `pop()` method. For example:

```
# delete a key-value pair
my_dict = {'apple': 1, 'banana': 2, 'orange': 3}
del my_dict['apple']
print(my_dict)
```

```
# remove a key-value pair using the pop method
my_dict = {'apple': 1, 'banana': 2, 'orange': 3}
popped_value = my_dict.pop('apple')
print(my_dict)
print(popped_value)
```

Iterating Through a Dictionary

You can iterate through a dictionary using a `for` loop. For example:

```
# iterate through a dictionary
my_dict = {'apple': 1, 'banana': 2, 'orange': 3}
```

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

Alternatively, you can use the `items()` method to iterate through the key-value pairs in a dictionary. For example:

```
# iterate through a dictionary
my_dict = {'apple': 1, 'banana': 2, 'orange': 3}
for key, value in my_dict.items():
    print(key, value)
```

Checking if a Key Exists in a Dictionary

To check whether a key exists in the dictionary, you can use the `in` keyword. For example:

```
# checking if a key exists
my_dict = {'apple': 1, 'banana': 2, 'orange': 3}

print('apple' in my_dict)
```

You can also use the `in` keyword in conjunction with an `if` statement to check if a key exists in a dictionary. For example:

```
# checking if a key exists
my_dict = {'apple': 1, 'banana': 2, 'orange': 3}

if 'apple' in my_dict:
    print('apple exists')
elif 'grape' in my_dict:
    print('grape exists')
```

As mentioned earlier, the `get()` method returns `None` if the key is not found in the dictionary. You can use this to check if a key exists in a dictionary.

Dictionary Comprehensions

Like lists, dictionaries can also be created using comprehensions. It provides a concise way to create a new dictionary from an iterable using a single line of code. The syntax for dictionary comprehension is as follows:

```
{key:value for (key, value) in iterable if condition}
```

where `key` is the key in the resulting dictionary, `value` is the value in the resulting dictionary, and `iterable` is the iterable you are using to create the dictionary. For example:

```
numbers = [1, 2, 3, 4, 5]
squares = {x: x**2 for x in numbers}
print(squares)
```

You can also filter the elements of the iterable using an `if` statement in the dictionary comprehension. For example:

```
numbers = [1, 2, 3, 4, 5]
even_squares = {x: x**2 for x in numbers if x % 2 == 0}
print(even_squares)
```

Exercises

1. Create a dictionary with the following key-value pairs: `"name": "John", "age": 30, "city": "New York"`. Print the dictionary.
2. Add a new key-value pair to the dictionary from exercise 1: `"job": "programmer"`. Print the updated dictionary.
3. Modify the `age` value in the dictionary from exercise 1 to `35`. Print the updated dictionary.
4. Create a dictionary where the keys are the numbers from `1` to `5`, and the values are the squares of those numbers.
5. Write a program that returns the sum of all the values in the dictionary.
6. Write a program that takes two dictionaries and returns a new dictionary that contains all the key-value pairs from both dictionaries.
7. Write a program that reads a file and counts the number of occurrences of each word in the file, using a dictionary to store the counts.
8. Write a program that asks the user for their name, age, and favorite color, and stores this information in a dictionary. Print the dictionary.
9. Write a program that creates a dictionary where the keys are the names of fruits, and the values are the prices of those fruits. Ask the user to enter a fruit name, and then print the price of that fruit.

10. Write a program that creates a dictionary where the keys are the names of students, and the values are lists of their test scores. Ask the user to enter a student name, and then print the average score for that student.

Project: Phonebook

Create a Python program that simulates a phonebook. The program should use a dictionary to store phone numbers and names, and allow the user to add, remove, and look up phone numbers.

Project Requirements

1. The program should allow the user to add phone numbers and names, remove phone numbers, and look up phone numbers.
2. The program should use a dictionary to store phone numbers and names.
3. The program should handle errors appropriately, for example, when the user enters non-numeric inputs.

Example Output

```
Phonebook

Choose an option:
1. Add phone number
2. Remove phone number
3. Look up phone number

Enter your choice (1/2/3): 1

Enter name: John
Enter phone number: 1234567890
Added John with phone number 1234567890 to phonebook.
```

Further Reading

Check out the following resources for more information on dictionaries:

- W3Schools
  - Python Dictionaries

  - Python Dictionary Methods

- Real Python
  - Dictionaries in Python
  - Python Dictionary Methods

- Python Documentation
  - Dictionaries
  - Dictionary Methods
  - Dictionary Views