Dictionaries in Python

A dictionary is a data structure in Python that allows you to store key-value pairs. Each key in a dictionary is unique, and the associated value can be of any data type, including another dictionary.

Dictionaries are a powerful tool in Python and can be used in a variety of applications, such as building databases, managing configuration files, and storing large amounts of data. Understanding how to use dictionaries effectively can be a valuable skill for any Python programmer.

Creating a dictionary

You can create a dictionary in Python by enclosing a comma-separated list of key-value pairs in curly braces {}. For example:

```
In [1]: my_dict = {'key1': 'value1', 'key2': 2, 'key3': [1, 2, 3]}
```

Accessing dictionary values

You can access the values in a dictionary by using the key as the index. For example:

```
In [2]: print(my_dict['key1'])  # output: 'value1'
```

```
value1
```

Modifying dictionary values

You can modify the values in a dictionary by assigning a new value to the key. For example:

```
In [3]: my_dict['key2'] = 3
        print(my_dict['key2'])  # output: 3
```

```
3
```

Adding new key-value pairs

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

```
In [4]: my_dict['key4'] = 'new value'
```

```
In [5]: print(my_dict)
```

```
{'key1': 'value1', 'key2': 3, 'key3': [1, 2, 3], 'key4': 'new value'}
```

Removing key-value pairs

You can remove a key-value pair from a dictionary using the del keyword. For example:

```
In [6]: del my_dict['key3']
```

```
In [7]: print(my_dict)
```

```
{'key1': 'value1', 'key2': 3, 'key4': 'new value'}
```

Dictionary methods

There are several built-in methods in Python that you can use to manipulate dictionaries. Some of the most common ones are:

keys(): Returns a list of all the keys in the dictionary. Example:

```
In [8]: # Creating a dictionary
        my_dict = {'apple': 5, 'banana': 10, 'orange': 15}

        keys = my_dict.keys()
        print(keys) # Output: dict_keys(['apple', 'banana', 'orange'])
```

```
dict_keys(['apple', 'banana', 'orange'])
```

values(): Returns a list of all the values in the dictionary. Example:

```
In [9]: values = my_dict.values()
        print(values) # Output: dict_values([5, 10, 15])
```

```
dict_values([5, 10, 15])
```

items(): Returns a list of all the key-value pairs in the dictionary as tuples. Example:

```
In [10]: items = my_dict.items()
         print(items) # Output: dict_items([('apple', 5), ('banana', 10), ('orange', 15)])
```

```
dict_items([('apple', 5), ('banana', 10), ('orange', 15)])
```

copy(): Creates a copy of the dictionary. Example:

```
In [11]: my_dict_copy = my_dict.copy()
         print(my_dict_copy) # Output: {'apple': 5, 'banana': 10, 'orange': 15}
```

```
{'apple': 5, 'banana': 10, 'orange': 15}
```

clear(): Removes all items from the dictionary. Example:

```
In [12]: my_dict.clear()
         print(my_dict) # Output: {}
         print(my_dict_copy) # Output: {'apple': 5, 'banana': 10, 'orange': 15}
```

```
{}
{'apple': 5, 'banana': 10, 'orange': 15}
```

Conclusion

To summarize, dictionaries are a built-in data structure in Python that allow you to store collections of key-value pairs. They are implemented as hash tables, which means that they provide very fast lookup times for values based on their keys.

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

Sets in Python

A set is another data structure in Python that is similar to a list or a dictionary, but with some unique properties. A set is an unordered collection of unique elements.

Sets are useful in a variety of applications, such as removing duplicates from a list or checking for common elements between two sets. Understanding how to use sets effectively can be a valuable skill for any Python programmer.

Creating a set

You can create a set in Python by enclosing a comma-separated list of elements in curly braces {}. For example:

```
In [13]: my_set = {1, 2, 3, 4, 5}
```

Adding and removing elements

You can add an element to a set using the add() method, and remove an element using the remove() method. For example:

In [14]:
```
my_set.add(6)
my_set.remove(3)
```

In [15]:
```
print(my_set)
```

```
{1, 2, 4, 5, 6}
```

Set operations

There are several built-in set operations in Python that you can use to manipulate sets. Some of the most common ones are:

union(): Returns a new set containing all the elements from two or more sets. Example:

In [16]:
```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
union_set = set1.union(set2)
print(union_set)  # Output: {1, 2, 3, 4, 5}
```

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

intersection(): Returns a new set containing only the elements that are common to two or more sets. Example:

In [17]:
```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
intersection_set = set1.intersection(set2)
print(intersection_set)  # Output: {3}
```

```
{3}
```

difference(): Returns a new set containing the elements that are in one set but not in another. Example:

In [18]:
```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
difference_set = set1.difference(set2)
difference_set2 = set2.difference(set1)
print(difference_set)  # Output: {1, 2}
print(difference_set2)
```

```
{1, 2}
{4, 5}
```

symmetric_difference(): Returns a new set containing the elements that are in either of the two sets, but not in both. Example:

```
In [19]: set1 = {1, 2, 3}
         set2 = {3, 4, 5}
         symmetric_difference_set = set1.symmetric_difference(set2)
         print(symmetric_difference_set)  # Output: {1, 2, 4, 5}
```

```
{1, 2, 4, 5}
```

Set Methods

There are several built-in methods in Python that you can use to manipulate sets. Some of the most common ones are:

len(): Returns the number of elements in the set. Example:

```
In [20]: my_set = {1, 2, 3, 4, 5}
         print(len(my_set))  # Output: 5
```

```
5
```

add(): Adds a single element to the set. Example:

```
In [21]: my_set = {1, 2, 3}
         my_set.add(4)
         print(my_set)  # Output: {1, 2, 3, 4}
```

```
{1, 2, 3, 4}
```

remove(): Removes a specific element from the set. Example:

```
In [22]: my_set = {1, 2, 3}
         my_set.remove(3)
         print(my_set)  # Output: {1, 2}
```

```
{1, 2}
```

clear(): Removes all the elements from the set. Example:

```
In [23]: my_set = {1, 2, 3}
         my_set.clear()
         print(my_set)  # Output: set()
```

```
set()
```

copy(): Returns a new set that is a copy of the original set. Example:

```
In [24]: my_set = {1, 2, 3}
         new_set = my_set.copy()
         print(new_set)  # Output: {1, 2, 3}
```

```
{1, 2, 3}
```

Conclusion

In conclusion, sets are a built-in data structure in Python that allow you to store collections of unique elements. They are implemented as hash tables, which means that they provide very fast lookup times for elements.