Dictionaries

Dictionaries are one of the most used data structures in Python. Let's learn about dictionaries in python

WE'LL COVER THE FOLLOWING

Python dictionary is basically a **hash table** or a hash mapping. In some languages, they might be referred to as **associative memories** or **associative arrays**. They are indexed with keys, which can be any immutable type. For example, a string or number can be a key. You need to be aware that a dictionary is an unordered set of key:value pairs and the keys must be unique. You can get a list of keys by calling a dictionary instance's **keys** method. To check if a dictionary has a key, you can use Python's **in** keyword. In some of the older versions of Python (2.3 and older to be specific), you will see the **has_key** keyword used for testing if a key is in a dictionary. This keyword is deprecated in Python 2.x and removed entirely from Python 3.x.

Let's take a moment to see how we create a dictionary.

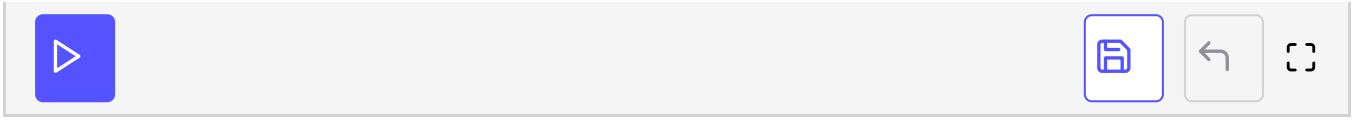
```
my_dict = {}
another_dict = dict()
my_other_dict = {"one":1, "two":2, "three":3}

print(my_other_dict) # {'three': 3, 'two': 2, 'one': 1}
```

The first two examples show how to create an empty dictionary. All dictionaries are enclosed with curly braces. The last line is printed out so you can see how unordered a dictionary is. Now it's time to find out how to access a value in a dictionary.

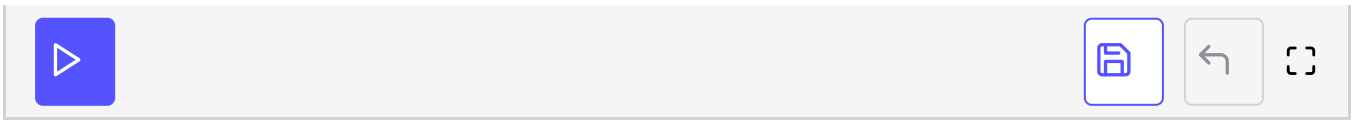
```
my_other_dict = {"one":1, "two":2, "three":3}
print(my_other_dict["one"]) # 1

my_dict = {"name":"Mike", "address":"123 Happy Way"}
print(my_dict["name"]) # 'Mike'
```



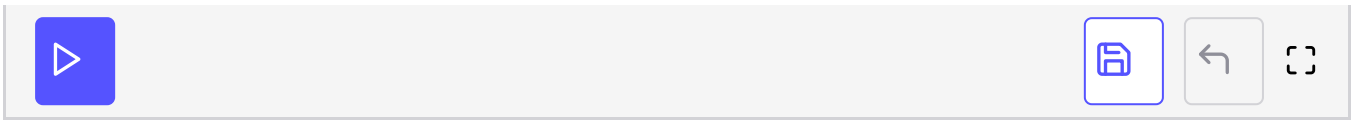
In the first example, we use the dictionary from the previous example and pull out the value associated with the key called “one”. The second example shows how to acquire the value for the “name” key. Now let’s see how to tell if a key is in a dictionary or not:

```
my_dict = {"name":"Mike", "address":"123 Happy Way"}
print("name" in my_dict) # True
print ("state" in my_dict) # False
```



So, if the key is in the dictionary, Python returns a **Boolean True**. Otherwise it returns a Boolean **False**. If you need to get a listing of all the keys in a dictionary, then you do this:

```
my_dict = {"name":"Mike", "address":"123 Happy Way"}
print(my_dict.keys()) # dict_keys(['name', 'address'])
```



In Python 2, the **keys** method returns a list. But in Python 3, it returns a *view object*. This gives the developer the ability to update the dictionary and the view will automatically update too. Also note that when using the **in** keyword for dictionary membership testing, it is better to do it against the dictionary instead of the list returned from the **keys** method. See below:

```
"name" in my_dict          # this is good
"name" in my_dict.keys()   # this works too, but is slower
```

While this probably won’t matter much to you right now, in a real job situation, seconds matter. When you have thousands of files to process, these little tricks can save you a lot of time in the long run!

little tricks can save you a lot of time in the long run!

Wrapping Up

In this chapter you just learned how to construct a Python list, tuple and dictionary. Make sure you understand everything in this section before moving on. These concepts will assist you in designing your programs. You will be building complex data structures using these building blocks every day if you choose to pursue employment as a Python programmer. Each of these data types can be nested inside the others. For example, you can have a nested dictionary, a dictionary of tuples, a tuple made up of several dictionaries, and on and on.

When you are ready to move on, we will learn about Python's support for conditional statements.