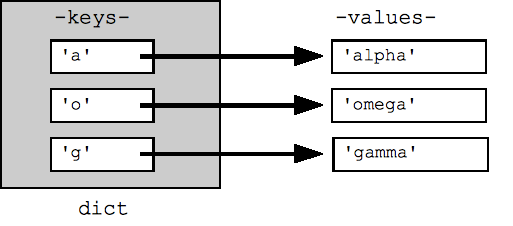
**Dictionaries**

Dictionary is the Python's HashTable implementation and might be one of the most common used data structures in programming. But they are named differently in almost most common languages. In Java they're called HashMap, in .NET KeyValuePair. In the most simplest form, a dictionary is;



**Creating a dictionary**

There are two ways to create dictionaries in Python;

**>>>** michaeljordan **=** **{** "name" **:** "Micheal"**,** "surname" **:** "Jordan"**,** "nick" **:** "AirJordan"**,** "status" **:** "Legend"**,** "age" **:** **50}**

**>>>** michaeljordan\_two **=** dict**(**name**=**"Micheal"**,**surname**=**'Jordan'**,**nick**=**"AirJordan"**,**status**=**"Legend"**,**age**=50)**

"name" is the key, and "Michael" is the value of the key. This is why .NET call them KeyValuePairs, and behind the scenes, dictionary objects' keys are stored as hashed, thus in Java they call them HashMaps.

As you see when you define a dictionary via curly brackets {}, you have to use string declaration syntax for keys as well.

**How to get the value of a key ?**

There are a number of ways to get the value(s) of a key

**>>>** michaeljordan**[**'name'**]**

'Michael'

there is a side effect when using this version; if the given key does not exist, Python will throw a KeyError exception. I typed the following into the Python REPL, and got the following result

**>>>** michaeljordan**[**"python\_articles"**]**

Traceback **(**most recent call last**):**

File "<stdin>"**,** line **1,** **in** **<**module**>**

**KeyError:** 'python\_articles'

Python is a straightforward language, if you want to get a value of a key, use get function.

**>>>** michaeljordan**.**get**(**"python\_articles"**)**

will return None so that you won't see anything in the terminal.

**>>>** michaeljordan**.**get**.**\_\_doc\_\_

'D.get(k[,d]) -> D[k] if k in D, else d. d defaults to None.'

The value of a non-existing key is None, Let's specify a default value <no\_name>

**>>>** michaeljordan**.**get**(**"python\_articles"**,**"<no\_name>"**)**

'<no\_name>'

**Check if a key exist**

Dictionaries have **has*key****method which returns a bool value; (****Note : Python 3 does not have has*key function anymore.**)

**>>>** michaeljordan**.**has\_key**(**"python\_articles"**)**

False

but it's not the preferred way. The better way to do it is;

**>>>** 'python\_articles' **in** michaeljordan

False

**>>>** 'name' **in** michaeljordan

True

**Iterate all the key/values of the dictionary**

**>>>** michaeljordan**.**items**()**

**[(**'status'**,** 'Legend'**),** **(**'nick'**,** 'AirJordan'**),** **(**'age'**,** **50),** **(**'surname'**,** 'Jordan'**),** **(**'name'**,** 'Micheal'**)]**

**>>>** **for** key**,** value **in** michaeljordan**.**items**():**

**...** **print** key**,** value

**...**

status Legend

nick AirJordan

age **50**

surname Jordan

name Micheal

To print key values, sorted by the key, you can use the built in sorted function

**>>>** **for** key**,** value **in** sorted**(**michaeljordan**.**items**()):**

**...** **print** key**,** value

**...**

age **50**

name Micheal

nick AirJordan

status Legend

surname Jordan

**How to get all the keys within a dictionary ?**

keys method helps you to get all the keys of a dictionary. It returns a list of strings

**>>>** michaeljordan**.**keys**()**

**[**'status'**,** 'nick'**,** 'surname'**,** 'name'**]**

**Updating the value of a key**

I made a typo while setting the name of Mr Jordan. Let's fix it.

michaeljordan**[**'name'**]** **=** "Michael"

michaeljordan\_two**[**'name'**]** **=** "Michael"

**Adding a new key value to existing Dictionary**

It's pretty straight forward as well

michaeljordan**[**'team'**]** **=** "Chicago Bulls"

adds the key **team** to the dictionary

**Dictionaries inside dictionaries**

If you are going to work on any API, you have the understand the nested concept of dictionaries which are inside lists, tuples or in other dictionaries.

Let's add Mr. Jordans work history. Starting with Chicago Bulls

**>>>** michaeljordan **=** **{** "name" **:** "Micheal"**,** "surname" **:** "Jordan"**,** "nick" **:** "AirJordan"**,** "status" **:** "Legend"**,** "age" **:** **50** **,** "team" **:** **{** "name" **:** "Chicago Bulls" **,** "start\_year" **:** **1984** **,** "end\_year" **:** **1998** **}** **}**

Lets get the **start*year****and****end*year** from the dictionary.

**>>>** michaeljordan**[**'team'**][**'start\_year'**]**

**1984**

But Mr. Jordan did not play only for Chicago Bulls

**>>>** michaeljordan**[**'teams'**]** **=** **[** **{**"name" **:** "Chicago Bulls" **,** "start\_year" **:** **1984** **,** "end\_year" **:** **1998** **},** **{** "name" **:** "Washington Wizards" **,** "start\_year" **:** **2001** **,** "end\_year" **:** **2003** **}** **]**

Iterate through all the teams that Mr. Jordan played

**>>>** michaeljordan**[**'teams'**]** **=** **[** **{**"name" **:** "Chicago Bulls" **,** "start\_year" **:** **1984** **,** "end\_year" **:** **1998** **},** **{** "name" **:** "Washington Wizards" **,** "start\_year" **:** **2001** **,** "end\_year" **:** **2003** **}** **]**

**>>>** **for** team **in** michaeljordan**[**'teams'**]** **:**

**...** **print** team**[**'name'**],** team**[**'start\_year'**],** team**[**'end\_year'**]**

**...**

Chicago Bulls **1984** **1998**

Washington Wizards **2001** **2003**

Lets combine our [List Comprehension knowledge](http://pythonarticles.com/list_comprehension.html) with dictionaries. Create an array of array which contains the values and key of the teams data.

**>>>** **[**team**.**values**()** **for** team **in** michaeljordan**[**'teams'**]]**

**[[1998,** **1984,** 'Chicago Bulls'**],** **[2003,** **2001,** 'Washington Wizards'**]]**

**>>>** **[**team**.**keys**()** **for** team **in** michaeljordan**[**'teams'**]]**

**[[**'end\_year'**,** 'start\_year'**,** 'name'**],** **[**'end\_year'**,** 'start\_year'**,** 'name'**]]**

**>>>** """Rather than using a list, lets use a tuple"""

**>>>** **[**tuple**(**team**.**values**())** **for** team **in** michaeljordan**[**'teams'**]]**

**[(1998,** **1984,** 'Chicago Bulls'**),** **(2003,** **2001,** 'Washington Wizards'**)]**

**Copy a dictionary into another one**

Our original Mr. Jordan information was stored in a variable called **michaeljordan** and we made some changes, but**michaeljordan\_two** was not sync with the michaeljordan

**>>>** michaeljordan **=** **{** "name" **:** "Micheal"**,** "surname" **:** "Jordan"**,** "nick" **:** "AirJordan"**,** "status" **:** "Legend"**,** "age" **:** **50}**

**>>>** michaeljordan\_two **=** michaeljordan**.**copy**()**

**>>>** id**(**michaeljordan**)**

**4298505024**

**>>>** id**(**michaeljordan\_two**)**

**4298523824**

**>>>** michaeljordan**[**'test'**]** **=** **1**

**>>>** michaeljordan\_two**[**'test'**]**

Traceback **(**most recent call last**):**

File "<stdin>"**,** line **1,** **in** **<**module**>**

**KeyError:** 'test'

**>>>** michaeljordan**.**copy**.**\_\_doc\_\_

'D.copy() -> a shallow copy of D'

As the copy function descriptor explains, method copies all the value of the michaeljordan variable into a new **dictionary instance**.

**>>>** michaeljordan\_three **=** michaeljordan

**>>>** michaeljordan**[**'test2'**]** **=** **2**

**>>>** michaeljordan\_three**[**'test2'**]**

**2**

Whereas assigning michaeljordan*three to michaeljordan made the michaeljordan*three pointing to the value of the michaeljordan. So when working with dictionaries, think carefully how the new variable should act upon the changes in the original variable.

**Delete a key from the dictionary**

Our team key became pointless once we set the teams key. Lets remove it

**>>>** **del** michaeljordan**[**'team'**]**

**>>>** michaeljordan

**{**'status'**:** 'Legend'**,** 'surname'**:** 'Jordan'**,** 'name'**:** 'Micheal'**,** 'age'**:** **50,** 'teams'**:** **[{**'end\_year'**:** **1998,** 'start\_year'**:** **1984,** 'name'**:** 'Chicago Bulls'**},** **{**'end\_year'**:** **2003,** 'start\_year'**:** **2001,** 'name'**:** 'Washington Wizards'**}],** 'nick'**:** 'AirJordan'**}**

**Setdefault**

It's a handy function, like a placeholder. If the key exist in the dictionary returns the value, if not the value which set by thesetdefault call is returned. Here is an example..

**>>>** a **=** **{}**

**>>>** a**.**setdefault**(**"test"**,**"empty"**)**

'empty'

**>>>** a**[**'test'**]**

'empty'

**>>>** a**[**'test2'**]**

Traceback **(**most recent call last**):**

File "<stdin>"**,** line **1,** **in** **<**module**>**

**KeyError:** 'test2'

**>>>** a**[**'test'**]** **=** 'not-empty-anymore'

**>>>** a**[**'test'**]**

'not-empty-anymore'

**Dictionary comprehension**

Have you read the [List Comprehension](http://pythonarticles.com/list_comprehension.html) ?

The following code will only work **Python2.7+**

**>>>** sequence **=** **[(1,2),** **(2,3)** **,** **(3,4)]** *# will work with list of tuples*

**>>>** sequence **=** **((1,2),** **(2,3)** **,** **(3,4))** *# will also work with tuple of tuples*

**>>>** **{**key**:** value **for** **(**key**,** value**)** **in** sequence**}**

**{1:** **2,** **2:** **3,** **3:** **4}**

For earlier versions of python

dict((key, value) for (key, value) in sequence)