**Intermediate Python for Data Science**

**#2 Dictionaries & Pandas**

**Motivation for dictionaries**

To see why dictionaries are useful, have a look at the two lists defined on the right. countries contains the names of some European countries. capitals lists the corresponding names of their capital.

**Instructions**

**100 XP**

* Use the [**index()**](https://docs.python.org/3/library/stdtypes.html#common-sequence-operations) method on countries to find the index of 'germany'. Store this index as ind\_ger.
* Use ind\_ger to access the capital of Germany from the capitals list. Print it out.

Script.py  
1 # Definition of countries and capital

2 countries = ['spain', 'france', 'germany', 'norway']

3 capitals = ['madrid', 'paris', 'berlin', 'oslo']

4

5 # Get index of 'germany': ind\_ger

6 ind\_ger = countries.index("germany")

7

8 # Use ind\_ger to print out capital of Germany

9 print(capitals[ind\_ger])

IPython Shell  
In [1]: # Definition of countries and capital

countries = ['spain', 'france', 'germany', 'norway']

capitals = ['madrid', 'paris', 'berlin', 'oslo']

# Get index of 'germany': ind\_ger

ind\_ger = countries.index("germany")

# Use ind\_ger to print out capital of Germany

print(capitals[ind\_ger])

berlin

<script.py> output:

berlin

In [2]:

**Create dictionary**

The countries and capitals lists are again available in the script. It's your job to convert this data to a dictionary where the country names are the keys and the capitals are the corresponding values. As a refresher, here is a recipe for creating a dictionary:

my\_dict = {

"key1":"value1",

"key2":"value2",

}

In this recipe, both the keys and the values are strings. This will also be the case for this exercise.

**Instructions**

**100 XP**

* With the strings in countries and capitals, create a dictionary called europe with 4 key:value pairs. Beware of capitalization! Make sure you use lowercase characters everywhere.
* Print out europe to see if the result is what you expected.

Script.py  
01 # Definition of countries and capital

02 countries = ['spain', 'france', 'germany', 'norway']

03 capitals = ['madrid', 'paris', 'berlin', 'oslo']

04

05 # From string in countries and capitals, create dictionary europe

06 europe = { 'spain':'madrid', 'france':'paris', 'germany':'berlin', 'norway':'oslo'}

07

08 # Print europe

09 print(europe)

IPython Shell  
In [2]: # Definition of countries and capital

countries = ['spain', 'france', 'germany', 'norway']

capitals = ['madrid', 'paris', 'berlin', 'oslo']

# From string in countries and capitals, create dictionary europe

europe = { 'spain':'madrid', 'france':'paris', 'germany':'berlin', 'norway':'oslo'}

# Print europe

print(europe)

{'france': 'paris', 'norway': 'oslo', 'germany': 'berlin', 'spain': 'madrid'}

<script.py> output:

{'france': 'paris', 'norway': 'oslo', 'germany': 'berlin', 'spain': 'madrid'}

In [3]:

**Access dictionary**

If the keys of a dictionary are chosen wisely, accessing the values in a dictionary is easy and intuitive. For example, to get the capital for France from europe you can use:

europe['france']

Here, 'france' is the key and 'paris' the value is returned.

**Instructions**

**100 XP**

* Check out which keys are in europe by calling the [**keys()**](https://docs.python.org/3/library/stdtypes.html#dict.keys) method on europe. Print out the result.
* Print out the value that belongs to the key 'norway'.

Script.py  
01 # Definition of dictionary

02 europe = {'spain':'madrid', 'france':'paris', 'germany':'berlin', 'norway':'oslo' }

03

04 # Print out the keys in europe

05 print(europe.keys())

06

07 # Print out value that belongs to key 'norway'

08 print(europe['norway'])

IPython Shell  
In [1]: # Definition of dictionary

europe = {'spain':'madrid', 'france':'paris', 'germany':'berlin', 'norway':'oslo' }

# Print out the keys in europe

print(europe.keys())

# Print out value that belongs to key 'norway'

print(europe['norway'])

dict\_keys(['france', 'norway', 'germany', 'spain'])

oslo

<script.py> output:

dict\_keys(['france', 'norway', 'germany', 'spain'])

oslo

In [2]:

**Dictionary Manipulation (1)**

If you know how to access a dictionary, you can also assign a new value to it. To add a new key-value pair to europeyou can use something like this:

europe['iceland'] = 'reykjavik'

**Instructions**

**100 XP**

* Add the key 'italy' with the value 'rome' to europe.
* To assert that 'italy' is now a key in europe, print out 'italy' in europe.
* Add another key:value pair to europe: 'poland' is the key, 'warsaw' is the corresponding value.
* Print out europe.

Script.py  
01 # Definition of dictionary

02 europe = {'spain':'madrid', 'france':'paris', 'germany':'berlin', 'norway':'oslo' }

03

04 # Add italy to europe

05 europe['italy'] = 'rome'

06

07 # Print out italy in europe

08 print('italy' in europe)

09

10 # Add poland to europe

11 europe['poland'] = 'warsaw'

12

13 # Print europe

14 print(europe)

IPython Shell  
In [1]: # Definition of dictionary

europe = {'spain':'madrid', 'france':'paris', 'germany':'berlin', 'norway':'oslo' }

# Add italy to europe

europe['italy'] = 'rome'

# Print out italy in europe

print('italy' in europe)

# Add poland to europe

europe['poland'] = 'warsaw'

# Print europe

print(europe)

True

{'spain': 'madrid', 'germany': 'berlin', 'norway': 'oslo', 'italy': 'rome', 'poland': 'warsaw', 'france': 'paris'}

<script.py> output:

True

{'spain': 'madrid', 'germany': 'berlin', 'norway': 'oslo', 'italy': 'rome', 'poland': 'warsaw', 'france': 'paris'}

In [2]:

**Dictionary Manipulation (2)**

Somebody thought it would be funny to mess with your accurately generated dictionary. An adapted version of the europe dictionary is available in the script on the right.

Can you clean up? Do not do this by adapting the definition of europe, but by adding Python commands to the script to update and remove key:value pairs.

**Instructions**

**100 XP**

* The capital of Germany is not 'bonn'; it's 'berlin'. Update its value.
* Australia is not in Europe, Austria is! Remove the key 'australia' from europe.
* Print out europe to see if your cleaning work paid off.

Sript.py  
01 # Definition of dictionary

02 europe = {'spain':'madrid', 'france':'paris', 'germany':'bonn',

03 'norway':'oslo', 'italy':'rome', 'poland':'warsaw',

04 'australia':'vienna' }

05

06 # Update capital of germany

07 europe['germany'] = 'berlin'

08

09 # Remove australia

10 del(europe['australia'])

11

12 # Print europe

13 print(europe)

IPython Shell  
In [1]: # Definition of dictionary

europe = {'spain':'madrid', 'france':'paris', 'germany':'bonn',

'norway':'oslo', 'italy':'rome', 'poland':'warsaw',

'australia':'vienna' }

# Update capital of germany

europe['germany'] = 'berlin'

# Remove australia

del(europe['australia'])

# Print europe

print(europe)

{'poland': 'warsaw', 'norway': 'oslo', 'france': 'paris', 'spain': 'madrid', 'germany': 'berlin', 'italy': 'rome'}

In [2]:

**Dictionariception**

Remember lists? They could contain anything, even other lists. Well, for dictionaries the same holds. Dictionaries can contain key:value pairs where the values are again dictionaries.

As an example, have a look at the script where another version of europe - the dictionary you've been working with all along - is coded. The keys are still the country names, but the values are dictionaries that contain more information than just the capital.

It's perfectly possible to chain square brackets to select elements. To fetch the population for Spain from europe, for example, you need:

europe['spain']['population']

**Instructions**

**100 XP**

* Use chained square brackets to select and print out the capital of France.
* Create a dictionary, named data, with the keys 'capital' and 'population'. Set them to 'rome' and 59.83, respectively.
* Add a new key-value pair to europe; the key is 'italy' and the value is data, the dictionary you just built.

Script.py  
01 # Dictionary of dictionaries

02 europe = { 'spain': { 'capital':'madrid', 'population':46.77 },

03 'france': { 'capital':'paris', 'population':66.03 },

04 'germany': { 'capital':'berlin', 'population':80.62 },

05 'norway': { 'capital':'oslo', 'population':5.084 } }

06

07 # Print out the capital of France

08 print(europe['france']['capital'])

09

10 # Create sub-dictionary data

11 data = {'capital':'rome', 'population':59.83}

12

13 # Add data to europe under key 'italy'

14 europe['italy'] = data

15

16 # Print europe

17 print(europe)

IPython Shell  
In [1]: # Dictionary of dictionaries

europe = { 'spain': { 'capital':'madrid', 'population':46.77 },

'france': { 'capital':'paris', 'population':66.03 },

'germany': { 'capital':'berlin', 'population':80.62 },

'norway': { 'capital':'oslo', 'population':5.084 } }

# Print out the capital of France

print(europe['france']['capital'])

# Create sub-dictionary data

data = {'capital':'rome', 'population':59.83}

# Add data to europe under key 'italy'

europe['italy'] = data

# Print europe

print(europe)

paris

{'spain': {'population': 46.77, 'capital': 'madrid'}, 'norway': {'population': 5.084, 'capital': 'oslo'}, 'france': {'population': 66.03, 'capital': 'paris'}, 'italy': {'population': 59.83, 'capital': 'rome'}, 'germany': {'population': 80.62, 'capital': 'berlin'}}

<script.py> output:

paris

{'spain': {'population': 46.77, 'capital': 'madrid'}, 'norway': {'population': 5.084, 'capital': 'oslo'}, 'france': {'population': 66.03, 'capital': 'paris'}, 'italy': {'population': 59.83, 'capital': 'rome'}, 'germany': {'population': 80.62, 'capital': 'berlin'}}

In [2]:

**Dictionary to DataFrame (1)**

Pandas is an open source library, providing high-performance, easy-to-use data structures and data analysis tools for Python. Sounds promising!

The DataFrame is one of Pandas' most important data structures. It's basically a way to store tabular data where you can label the rows and the columns. One way to build a DataFrame is from a dictionary.

In the exercises that follow you will be working with vehicle data from different countries. Each observation corresponds to a country and the columns give information about the number of vehicles per capita, whether people drive left or right, and so on.

Three lists are defined in the script:

* names, containing the country names for which data is available.
* dr, a list with booleans that tells whether people drive left or right in the corresponding country.
* cpc, the number of motor vehicles per 1000 people in the corresponding country.

Each dictionary key is a column label and each value is a list which contains the column elements.

**Instructions**

**100 XP**

* Import pandas as pd.
* Use the pre-defined lists to create a dictionary called my\_dict. There should be three key value pairs:
* key 'country' and value names.
* key 'drives\_right' and value dr.
* key 'cars\_per\_cap' and value cpc.
* Use [**pd.DataFrame()**](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html) to turn your dict into a DataFrame called cars.
* Print out cars and see how beautiful it is.

Script.py  
01 # Pre-defined lists

02 names = ['United States', 'Australia', 'Japan', 'India', 'Russia', 'Morocco', 'Egypt']

03 dr = [True, False, False, False, True, True, True]

04 cpc = [809, 731, 588, 18, 200, 70, 45]

05

06 # Import pandas as pd

07 import pandas as pd

08

09 # Create dictionary my\_dict with three key:value pairs: my\_dict

10 my\_dict = { 'country':names, 'drives\_right':dr, 'cars\_per\_cap':cpc }

11

12 # Build a DataFrame cars from my\_dict: cars

13 cars = pd.DataFrame(my\_dict)

14

15 # Print cars

16 print(cars)

IPython Shell  
In [1]: # Pre-defined lists

names = ['United States', 'Australia', 'Japan', 'India', 'Russia', 'Morocco', 'Egypt']

dr = [True, False, False, False, True, True, True]

cpc = [809, 731, 588, 18, 200, 70, 45]

# Import pandas as pd

import pandas as pd

# Create dictionary my\_dict with three key:value pairs: my\_dict

my\_dict = { 'country':names, 'drives\_right':dr, 'cars\_per\_cap':cpc }

# Build a DataFrame cars from my\_dict: cars

cars = pd.DataFrame(my\_dict)

# Print cars

print(cars)

cars\_per\_cap country drives\_right

0 809 United States True

1 731 Australia False

2 588 Japan False

3 18 India False

4 200 Russia True

5 70 Morocco True

6 45 Egypt True

<script.py> output:

cars\_per\_cap country drives\_right

0 809 United States True

1 731 Australia False

2 588 Japan False

3 18 India False

4 200 Russia True

5 70 Morocco True

6 45 Egypt True

In [2]:

**Dictionary to DataFrame (2)**

The Python code that solves the previous exercise is included on the right. Have you noticed that the row labels (i.e. the labels for the different observations) were automatically set to integers from 0 up to 6?

To solve this a list row\_labels has been created. You can use it to specify the row labels of the cars DataFrame. You do this by setting the index attribute of cars, that you can access as cars.index.

**Instructions**

**100 XP**

* Hit *Submit Answer* to see that, indeed, the row labels are not correctly set.
* Specify the row labels by setting cars.index equal to row\_labels.
* Print out cars again and check if the row labels are correct this time.

Script.py  
01 import pandas as pd

02

03 # Build cars DataFrame

04 names = ['United States', 'Australia', 'Japan', 'India', 'Russia', 'Morocco', 'Egypt']

05 dr = [True, False, False, False, True, True, True]

06 cpc = [809, 731, 588, 18, 200, 70, 45]

07 dict = { 'country':names, 'drives\_right':dr, 'cars\_per\_cap':cpc }

08 cars = pd.DataFrame(dict)

09 print(cars)

10

11 # Definition of row\_labels

12 row\_labels = ['US', 'AUS', 'JAP', 'IN', 'RU', 'MOR', 'EG']

13

14 # Specify row labels of cars

15 cars.index = row\_labels

16

17 # Print cars again

18 print(cars)

IPython Shell  
In [1]: import pandas as pd

# Build cars DataFrame

names = ['United States', 'Australia', 'Japan', 'India', 'Russia', 'Morocco', 'Egypt']

dr = [True, False, False, False, True, True, True]

cpc = [809, 731, 588, 18, 200, 70, 45]

dict = { 'country':names, 'drives\_right':dr, 'cars\_per\_cap':cpc }

cars = pd.DataFrame(dict)

print(cars)

# Definition of row\_labels

row\_labels = ['US', 'AUS', 'JAP', 'IN', 'RU', 'MOR', 'EG']

# Specify row labels of cars

cars.index = row\_labels

# Print cars again

print(cars)

cars\_per\_cap country drives\_right

0 809 United States True

1 731 Australia False

2 588 Japan False

3 18 India False

4 200 Russia True

5 70 Morocco True

6 45 Egypt True

cars\_per\_cap country drives\_right

US 809 United States True

AUS 731 Australia False

JAP 588 Japan False

IN 18 India False

RU 200 Russia True

MOR 70 Morocco True

EG 45 Egypt True

In [2]:

**CSV to DataFrame (1)**

Putting data in a dictionary and then building a DataFrame works, but it's not very efficient. What if you're dealing with millions of observations? In those cases, the data is typically available as files with a regular structure. One of those file types is the CSV file, which is short for "comma-separated values".

To import CSV data into Python as a Pandas DataFrame you can use [**read\_csv()**](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html).

Let's explore this function with the same cars data from the previous exercises. This time, however, the data is available in a CSV file, named cars.csv. It is available in your current working directory, so the path to the file is simply 'cars.csv'.

**Instructions**

**100 XP**

* To import CSV files you still need the pandas package: import it as pd.
* Use [**pd.read\_csv()**](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html) to import cars.csv data as a DataFrame. Store this dataframe as cars.
* Print out cars. Does everything look OK?

Script.py  
1 # Import pandas as pd

2 import pandas as pd

3

4 # Import the cars.csv data: cars

5 cars = pd.read\_csv('cars.csv')

6

7 # Print out cars

8 print(cars)

IPython Shell  
In [1]: # Import pandas as pd

import pandas as pd

# Import the cars.csv data: cars

cars = pd.read\_csv('cars.csv')

# Print out cars

print(cars)

Unnamed: 0 cars\_per\_cap country drives\_right

0 US 809 United States True

1 AUS 731 Australia False

2 JAP 588 Japan False

3 IN 18 India False

4 RU 200 Russia True

5 MOR 70 Morocco True

6 EG 45 Egypt True

In [2]:

**CSV to DataFrame (2)**

Your [**read\_csv()**](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html) call to import the CSV data didn't generate an error, but the output is not entirely what we wanted. The row labels were imported as another column without a name.

Remember index\_col, an argument of [**read\_csv()**](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html), that you can use to specify which column in the CSV file should be used as a row label? Well, that's exactly what you need here!

Python code that solves the previous exercise is already included; can you make the appropriate changes to fix the data import?

**Instructions**

**100 XP**

* Run the code with *Submit Answer* and assert that the first column should actually be used as row labels.
* Specify the index\_col argument inside [**pd.read\_csv()**](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html): set it to 0, so that the first column is used as row labels.
* Has the printout of cars improved now?

Script.py  
1 # Import pandas as pd

2 import pandas as pd

3

4 # Fix import by including index\_col

5 cars = pd.read\_csv('cars.csv', index\_col= 0)

6

7 # Print out cars

8 print(cars)

IPython Shell  
In [1]: # Import pandas as pd

import pandas as pd

# Fix import by including index\_col

cars = pd.read\_csv('cars.csv')

# Print out cars

print(cars)

Unnamed: 0 cars\_per\_cap country drives\_right

0 US 809 United States True

1 AUS 731 Australia False

2 JAP 588 Japan False

3 IN 18 India False

4 RU 200 Russia True

5 MOR 70 Morocco True

6 EG 45 Egypt True

<script.py> output:

Unnamed: 0 cars\_per\_cap country drives\_right

0 US 809 United States True

1 AUS 731 Australia False

2 JAP 588 Japan False

3 IN 18 India False

4 RU 200 Russia True

5 MOR 70 Morocco True

6 EG 45 Egypt True

In [2]: # Import pandas as pd

import pandas as pd

# Fix import by including index\_col

cars = pd.read\_csv('cars.csv', index\_col= 0)

# Print out cars

print(cars)

cars\_per\_cap country drives\_right

US 809 United States True

AUS 731 Australia False

JAP 588 Japan False

IN 18 India False

RU 200 Russia True

MOR 70 Morocco True

EG 45 Egypt True

<script.py> output:

cars\_per\_cap country drives\_right

US 809 United States True

AUS 731 Australia False

JAP 588 Japan False

IN 18 India False

RU 200 Russia True

MOR 70 Morocco True

EG 45 Egypt True

In [3]:

**Square Brackets (1)**

In the video, you saw that you can index and select Pandas DataFrames in many different ways. The simplest, but not the most powerful way, is to use square brackets.

In the sample code on the right, the same cars data is imported from a CSV files as a Pandas DataFrame. To select only the cars\_per\_cap column from cars, you can use:

cars['cars\_per\_cap']

cars[['cars\_per\_cap']]

The single bracket version gives a Pandas Series, the double bracket version gives a Pandas DataFrame.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Use single square brackets to print out the countrycolumn of cars as a Pandas Series.
* Use double square brackets to print out the countrycolumn of cars as a Pandas DataFrame.
* Use double square brackets to print out a DataFrame with both the country and drives\_right columns of cars, in this order.

Sciript.py  
01 # Import cars data

02 import pandas as pd

03 cars = pd.read\_csv('cars.csv', index\_col = 0)

04

05 # Print out country column as Pandas Series

06 print(cars['country'])

07

08 # Print out country column as Pandas DataFrame

09 print(cars[['country']])

10

11 # Print out DataFrame with country and drives\_right columns

12 print(cars[['country', 'drives\_right']])

IPython Shell  
In [1]: # Import cars data

import pandas as pd

cars = pd.read\_csv('cars.csv', index\_col = 0)

# Print out country column as Pandas Series

print(cars['country'])

# Print out country column as Pandas DataFrame

print(cars[['country']])

# Print out DataFrame with country and drives\_right columns

print(cars[['country', 'drives\_right']])

US United States

AUS Australia

JAP Japan

IN India

RU Russia

MOR Morocco

EG Egypt

Name: country, dtype: object

country

US United States

AUS Australia

JAP Japan

IN India

RU Russia

MOR Morocco

EG Egypt

country drives\_right

US United States True

AUS Australia False

JAP Japan False

IN India False

RU Russia True

MOR Morocco True

EG Egypt True

In [2]:

**Square Brackets (2)**

Square brackets can do more than just selecting columns. You can also use them to get rows, or observations, from a DataFrame. The following call selects the first five rows from the cars DataFrame:

cars[0:5]

The result is another DataFrame containing only the rows you specified.

Pay attention: You can only select rows using square brackets if you specify a slice, like 0:4. Also, you're using the integer indexes of the rows here, not the row labels!

**Instructions**

**100 XP**

* Select the first 3 observations from cars and print them out.
* Select the fourth, fifth and sixth observation, corresponding to row indexes 3, 4 and 5, and print them out.

Script.py  
1 # Import cars data

2 import pandas as pd

3 cars = pd.read\_csv('cars.csv', index\_col = 0)

4

5 # Print out first 3 observations

6 print(cars[0:3])

7

8 # Print out fourth, fifth and sixth observation

9 print(cars[3:6])

IPython Shell  
In [1]: # Import cars data

import pandas as pd

cars = pd.read\_csv('cars.csv', index\_col = 0)

# Print out first 3 observations

print(cars[0:3])

# Print out fourth, fifth and sixth observation

print(cars[3:6])

cars\_per\_cap country drives\_right

US 809 United States True

AUS 731 Australia False

JAP 588 Japan False

cars\_per\_cap country drives\_right

IN 18 India False

RU 200 Russia True

MOR 70 Morocco True

In [2]:

**loc and iloc (1)**

With [**loc**](http://pandas.pydata.org/pandas-docs/stable/indexing.html#different-choices-for-indexing) and [**iloc**](http://pandas.pydata.org/pandas-docs/stable/indexing.html#different-choices-for-indexing) you can do practically any data selection operation on DataFrames you can think of. [**loc**](http://pandas.pydata.org/pandas-docs/stable/indexing.html#different-choices-for-indexing)is label-based, which means that you have to specify rows and columns based on their row and column labels. [**iloc**](http://pandas.pydata.org/pandas-docs/stable/indexing.html#different-choices-for-indexing)is integer index based, so you have to specify rows and columns by their integer index like you did in the previous exercise.

Try out the following commands in the IPython Shell to experiment with [**loc**](http://pandas.pydata.org/pandas-docs/stable/indexing.html#different-choices-for-indexing) and [**iloc**](http://pandas.pydata.org/pandas-docs/stable/indexing.html#different-choices-for-indexing) to select observations. Each pair of commands here gives the same result.

cars.loc['RU']

cars.iloc[4]

cars.loc[['RU']]

cars.iloc[[4]]

cars.loc[['RU', 'AUS']]

cars.iloc[[4, 1]]

As before, code is included that imports the cars data as a Pandas DataFrame.

**Instructions**

**100 XP**

* Use [**loc**](http://pandas.pydata.org/pandas-docs/stable/indexing.html#different-choices-for-indexing) or [**iloc**](http://pandas.pydata.org/pandas-docs/stable/indexing.html#different-choices-for-indexing) to select the observation corresponding to Japan as a Series. The label of this row is JAP, the index is 2. Make sure to print the resulting Series.
* Use [**loc**](http://pandas.pydata.org/pandas-docs/stable/indexing.html#different-choices-for-indexing) or [**iloc**](http://pandas.pydata.org/pandas-docs/stable/indexing.html#different-choices-for-indexing) to select the observations for Australia and Egypt as a DataFrame. You can find out about the labels/indexes of these rows by inspecting cars in the IPython Shell. Make sure to print the resulting DataFrame.

Script.py  
01 # Import cars data

02 import pandas as pd

03 cars = pd.read\_csv('cars.csv', index\_col = 0)

04

05 # Print out observation for Japan

06 print(cars.loc['JAP'])

07

08 # Print out observations for Australia and Egypt

09 print(cars.loc[['AUS','EG']])

IPython Shell  
In [1]: print(cars)

cars\_per\_cap country drives\_right

US 809 United States True

AUS 731 Australia False

JAP 588 Japan False

IN 18 India False

RU 200 Russia True

MOR 70 Morocco True

EG 45 Egypt True

In [2]: # Import cars data

import pandas as pd

cars = pd.read\_csv('cars.csv', index\_col = 0)

# Print out observation for Japan

print(cars.loc['JAP'])

# Print out observations for Australia and Egypt

print(cars.loc[['AUS','EG']])

cars\_per\_cap 588

country Japan

drives\_right False

Name: JAP, dtype: object

cars\_per\_cap country drives\_right

AUS 731 Australia False

EG 45 Egypt True

<script.py> output:

cars\_per\_cap 588

country Japan

drives\_right False

Name: JAP, dtype: object

cars\_per\_cap country drives\_right

AUS 731 Australia False

EG 45 Egypt True

In [3]:

**loc and iloc (2)**

loc and [**iloc**](http://pandas.pydata.org/pandas-docs/stable/indexing.html#different-choices-for-indexing) also allow you to select both rows and columns from a DataFrame. To experiment, try out the following commands in the IPython Shell. Again, paired commands produce the same result.

cars.loc['IN', 'cars\_per\_cap']

cars.iloc[3, 0]

cars.loc[['IN', 'RU'], 'cars\_per\_cap']

cars.iloc[[3, 4], 0]

cars.loc[['IN', 'RU'], ['cars\_per\_cap', 'country']]

cars.iloc[[3, 4], [0, 1]]

**Instructions**

**100 XP**

* Print out the drives\_right value of the row corresponding to Morocco (its row label is MOR)
* Print out a sub-DataFrame, containing the observations for Russia and Morocco and the columns country and drives\_right.

Script.py  
01 # Import cars data

02 import pandas as pd

03 cars = pd.read\_csv('cars.csv', index\_col = 0)

04

05 # Print out drives\_right value of Morocco

06 print(cars.loc['MOR'],'drives\_right')

07

08 # Print sub-DataFrame

09 print(cars.loc[['RU', 'MOR'], ['country', 'drives\_right']])

IPython Shell  
In [1]: print(cars)

cars\_per\_cap country drives\_right

US 809 United States True

AUS 731 Australia False

JAP 588 Japan False

IN 18 India False

RU 200 Russia True

MOR 70 Morocco True

EG 45 Egypt True

In [2]: # Import cars data

import pandas as pd

cars = pd.read\_csv('cars.csv', index\_col = 0)

# Print out drives\_right value of Morocco

print(cars.loc['MOR'],'drives\_right')

# Print sub-DataFrame

print(cars.loc[['RU', 'MOR'], ['country', 'drives\_right']])

cars\_per\_cap 70

country Morocco

drives\_right True

Name: MOR, dtype: object drives\_right

country drives\_right

RU Russia True

MOR Morocco True

<script.py> output:

cars\_per\_cap 70

country Morocco

drives\_right True

Name: MOR, dtype: object drives\_right

country drives\_right

RU Russia True

MOR Morocco True

In [3]:

# loc and iloc (3)

It's also possible to select only columns with [**loc**](http://pandas.pydata.org/pandas-docs/stable/indexing.html#different-choices-for-indexing) and [**iloc**](http://pandas.pydata.org/pandas-docs/stable/indexing.html#different-choices-for-indexing). In both cases, you simply put a slice going from beginning to end in front of the comma:

cars.loc[:, 'country']

cars.iloc[:, 1]

cars.loc[:, ['country','drives\_right']]

cars.iloc[:, [1, 2]]

**Instructions**

**100 XP**

* Print out the drives\_right column as a Series using [**loc**](http://pandas.pydata.org/pandas-docs/stable/indexing.html#different-choices-for-indexing) or [**iloc**](http://pandas.pydata.org/pandas-docs/stable/indexing.html#different-choices-for-indexing).
* Print out the drives\_right column as a DataFrame using [**loc**](http://pandas.pydata.org/pandas-docs/stable/indexing.html#different-choices-for-indexing) or [**iloc**](http://pandas.pydata.org/pandas-docs/stable/indexing.html#different-choices-for-indexing).
* Print out both the cars\_per\_cap and drives\_rightcolumn as a DataFrame using [**loc**](http://pandas.pydata.org/pandas-docs/stable/indexing.html#different-choices-for-indexing) or [**iloc**](http://pandas.pydata.org/pandas-docs/stable/indexing.html#different-choices-for-indexing).

Script.py  
01 # Import cars data

02 import pandas as pd

03 cars = pd.read\_csv('cars.csv', index\_col = 0)

04

05 # Print out drives\_right column as Series

06 print(cars.iloc[:, 2])

07

08 # Print out drives\_right column as DataFrame

09 print(cars.iloc[:, [2]])

10

11 # Print out cars\_per\_cap and drives\_right as DataFrame

12 print(cars.iloc[:, [0, 2]])

IPython Shell  
In [1]: print(cars)

cars\_per\_cap country drives\_right

US 809 United States True

AUS 731 Australia False

JAP 588 Japan False

IN 18 India False

RU 200 Russia True

MOR 70 Morocco True

EG 45 Egypt True

In [2]: # Import cars data

import pandas as pd

cars = pd.read\_csv('cars.csv', index\_col = 0)

# Print out drives\_right column as Series

print(cars.iloc[:, 2])

# Print out drives\_right column as DataFrame

print(cars.iloc[:, [2]])

# Print out cars\_per\_cap and drives\_right as DataFrame

print(cars.iloc[:, [0, 2]])

US True

AUS False

JAP False

IN False

RU True

MOR True

EG True

Name: drives\_right, dtype: bool

drives\_right

US True

AUS False

JAP False

IN False

RU True

MOR True

EG True

cars\_per\_cap drives\_right

US 809 True

AUS 731 False

JAP 588 False

IN 18 False

RU 200 True

MOR 70 True

EG 45 True

<script.py> output:

US True

AUS False

JAP False

IN False

RU True

MOR True

EG True

Name: drives\_right, dtype: bool

drives\_right

US True

AUS False

JAP False

IN False

RU True

MOR True

EG True

cars\_per\_cap drives\_right

US 809 True

AUS 731 False

JAP 588 False

IN 18 False

RU 200 True

MOR 70 True

EG 45 True

In [3]: