**Data Collection**

The term data collection refers to the process of acquiring data, collating that data, and then loading the data into your work environment of choice (such as Jupyter Notebook). In this subunit, you’ll learn how to collect data from two main channels — the internet and files — and will get practice importing data from a CSV file, calling API to request data from websites, and scraping data from websites.

1

**Importing Data in Python, Part 1**

Saved



3 - 5 Hours

73 Points

As a data scientist, you’ll need to clean, wrangle and munge, visualize, build predictive models, and interpret data. Before you can do any of these things, however, you’ll need to know how to get data into Python. In this DataCamp resource, you'll learn three ways to import data into Python:

* From flat files such as .txts and .csvs
* From files native to other software such as Excel spreadsheets, Stata, SAS, and MATLAB files
* From relational databases such as SQLite and PostgreSQL

1. 1

#### Introduction and flat files

0%

In this chapter, you'll learn how to import data into Python from all types of flat files, which are a simple and prevalent form of data storage. You've previously learned how to use NumPy and pandas—you will learn how to use these packages to import flat files and customize your imports.

[View Chapter Details](https://www.datacamp.com/courses/importing-data-in-python-part-1?embedded=true)

  2

#### Importing data from other file types

0%

You've learned how to import flat files, but there are many other file types you will potentially have to work with as a data scientist. In this chapter, you'll learn how to import data into Python from a wide array of important file types. These include pickled files, Excel spreadsheets, SAS and Stata files, HDF5 files, a file type for storing large quantities of numerical data, and MATLAB files.

[View Chapter Details](https://www.datacamp.com/courses/importing-data-in-python-part-1?embedded=true)

  3

#### Working with relational databases in Python

0%

In this chapter, you'll learn how to extract meaningful data from relational databases, an essential skill for any data scientist. You will learn about relational models, how to create SQL queries, how to filter and order your SQL records, and how to perform advanced queries by joining database tables.

[View Chapter Details](https://www.datacamp.com/courses/importing-data-in-python-part-1?embedded=true)

This course is part of these tracks:

* [Data Scientist with Python](https://www.datacamp.com/tracks/data-scientist-with-python)
* [Importing & Cleaning Data with Python](https://www.datacamp.com/tracks/importing-cleaning-data-with-python)

##### Datasets

* [Chinook (SQLite)](https://assets.datacamp.com/production/repositories/487/datasets/ec8aa8bc9ffea6b4e2729e1a0a2d4aea2f300b3a/Chinook.sqlite)
* [LIGO (HDF5)](https://assets.datacamp.com/production/repositories/487/datasets/ab9107b749b832daada36bfaa718d9a591a0d69c/L-L1_LOSC_4_V1-1126259446-32.hdf5)
* [Battledeath (XLSX)](https://assets.datacamp.com/production/repositories/487/datasets/5e8897e4624f8577ed0d33aeafbe7bd88bfc424b/battledeath.xlsx)
* [Extent of infectious diseases (DTA)](https://assets.datacamp.com/production/repositories/487/datasets/c4129edae533cf2683d8995f6dcdbcf5f41520ba/disarea.dta)
* [Gene expressions (MATLAB)](https://assets.datacamp.com/production/repositories/487/datasets/2fc0beea2d8cc7c93d79e79344a6e9e66f65d1fe/ja_data2.mat)
* [MNIST](https://assets.datacamp.com/production/repositories/487/datasets/d6d1b84ef06151ff913b4173e2eca8e6d5fa959b/mnist_kaggle_some_rows.csv)
* [Sales (SAS7BDAT)](https://assets.datacamp.com/production/repositories/487/datasets/0300d44b3ac77accc4b9706af86e33037bda6861/sales.sas7bdat)
* [Seaslugs](https://assets.datacamp.com/production/repositories/487/datasets/07cd090cb965782011a76af72c16b400a5ca5cc0/seaslug.txt)
* [Titanic](https://assets.datacamp.com/production/repositories/487/datasets/be79810c4288801167cfb31dbedd396559816ade/titanic_sub.csv)
* **Course Description**



* As a data scientist, you will need to clean data, wrangle and munge it, visualize it, build predictive models, and interpret these models. Before you can do so, however, you will need to know how to get data into Python. In this course, you'll learn the many ways to import data into Python: from flat files such as .txt and .csv; from files native to other software such as Excel spreadsheets, Stata, SAS, and MATLAB files; and from relational databases such as SQLite and PostgreSQL.

#### Introduction and flat files

0%

In this chapter, you'll learn how to import data into Python from all types of flat files, which are a simple and prevalent form of data storage. You've previously learned how to use NumPy and pandas—you will learn how to use these packages to import flat files and customize your imports.

**Daily XP0**

# Welcome to the course!

**50 XP**

## 1. Welcome to the course!

Welcome to the first course on Importing Data in Python! My name is Hugo Bowne-Anderson and I am a Data Scientist at DataCamp.

## 2. Import data

In this course, you'll learn how to import data from a large variety of import data sources, for example, (i) flat files such as dot txts and dot csvs; (ii) files native to other software such as Excel spreadsheets, Stata, SAS and MATLAB files;

## 3. Import data

(iii) relational databases such as SQLite & PostgreSQL. We’ll cover all of these topics in this course.

## 4. Plain text files

First off, we're going to learn how to import basic text files, which we can broadly classify into 2 types of files - those containing plain text, such as the opening of Mark Twain's novel The Adventures of Huckleberry Finn, which you can see here,

## 5. Table data

and those containing records, that is, table data, such as titanic dot csv, in which each

1. 1 Source: Kaggle

## 6. Table data

row is a unique passenger onboard and each

## 7. Table data

column is a characteristic or feature, such as gender, cabin and 'survived or not'. The latter is known as a flat file and we'll come back to these in a minute.

## 8. Reading a text file

In this section, we'll figure out how to read lines from a plain text file: So let's do it! To check out any plain text file, you can use Python’s basic open function to open a connection to the file. To do so, you assign the filename to a variable as a string, pass the filename to the function open and also pass it the argument mode equals 'r', which makes sure that we can only read it (we wouldn't want to accidentally write to it!), assign the text from the file to a variable text by applying the method read to the connection to the file. After you do this, make sure that you close the connection to the file using the command file dot close. It’s always best practice to clean while cooking!

## 9. Printing a text file

You can then print the file to console and check it out using the command print(text). A brief side note:

## 10. Writing to a file

if you wanted to open a file in order to write to it, you would pass it the argument mode equals 'w'. We won't use that in this course as this is course on Importing Data but it is good to know. You can avoid having to close the connection to the file by

## 11. Context manager with

using a with statement. This allows you to create a context in which you can execute commands with the file open. Once out of this clause/context, the file is no longer open and, for this reason, with is called a Context Manager. What you're doing here is called 'binding' a variable in the context manager construct; while still within this construct, the variable file will be bound to open(filename, 'r'). It is best practice to use the with statement as you never have to concern yourself with closing the files again.

## 12. In the exercises, you’ll:

In the following interactive coding sessions, you’ll figure out how to print files to console. You’ll also learn to print specific lines, which can be very useful for large files. Then we’ll be back to discuss flat files and then I'll show you how to use the Python package NumPy to make our job of importing flat files & numerical data a far easier beast to tame.

## 13. Let's practice!

Enjoy!

# Exploring your working directory

In order to import data into Python, you should first have an idea of what files are in your working directory.

IPython, which is running on DataCamp's servers, has a bunch of cool commands, including its [magic commands](https://ipython.readthedocs.io/en/stable/overview.html). For example, starting a line with ! gives you complete system shell access. This means that the IPython magic command ! ls will display the contents of your current directory. Your task is to use the IPython magic command ! ls to check out the contents of your current directory and answer the following question: which of the following files is in your working directory?

##### Instructions

**50 XP**

##### Possible Answers



huck\_finn.txt



titanic.csv



**moby\_dick.txt This is the correct answer**

# Importing entire text files

In this exercise, you'll be working with the file moby\_dick.txt. It is a text file that contains the opening sentences of Moby Dick, one of the great American novels! Here you'll get experience opening a text file, printing its contents to the shell and, finally, closing it.

##### Instructions

**100 XP**

* Open the file moby\_dick.txt as read-only and store it in the variable file. Make sure to pass the filename enclosed in quotation marks ''.
* Print the contents of the file to the shell using the print() function. As Hugo showed in the video, you'll need to apply the method read() to the object file.
* Check whether the file is closed by executing print(file.closed).
* Close the file using the close() method.
* Check again that the file is closed as you did above.
* # Open a file: file
* file = open('moby\_dick.txt', mode='r')
* # Print it
* print(file.read())
* # Check whether file is closed
* print(file.closed)
* # Close file
* file.close()
* # Check whether file is closed
* print(file)

Call me Ishmael. Some years ago--never mind how long precisely--having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen and regulating the circulation. Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself involuntarily pausing before coffin warehouses, and bringing up the rear of every funeral I meet; and especially whenever my hypos get such an upper hand of me, that it requires a strong moral principle to prevent me from deliberately stepping into the street, and methodically knocking people's hats off--then, I account it high time to get to sea as soon as I can. This is my substitute for pistol and ball. With a philosophical flourish Cato throws himself upon his sword; I quietly take to the ship. There is nothing surprising in this. If they but knew it, almost all men in their degree, some time or other, cherish very nearly the same feelings towards the ocean with me.

False True

**Daily XP20**

# Importing text files line by line

For large files, we may not want to print all of their content to the shell: you may wish to print only the first few lines. Enter the readline() method, which allows you to do this. When a file called file is open, you can print out the first line by executing file.readline(). If you execute the same command again, the second line will print, and so on.

In the introductory video, Hugo also introduced the concept of a **context manager**. He showed that you can bind a variable file by using a context manager construct:

with open('huck\_finn.txt') as file:

While still within this construct, the variable file will be bound to open('huck\_finn.txt'); thus, to print the file to the shell, all the code you need to execute is:

with open('huck\_finn.txt') as file:

print(file.readline())

You'll now use these tools to print the first few lines of moby\_dick.txt!

##### Instructions

**100 XP**

* Open moby\_dick.txt using the with context manager and the variable file.
* Print the first three lines of the file to the shell by using readline() three times within the context manager.

# Open a file: file file = open('moby\_dick.txt', mode='r') # Print it print(file.read()) # Check whether file is closed print(file.closed) # Close file file.close() # Check whether file is closed print(file)

# Read & print the first 3 lines

with open('moby\_dick.txt') as \_\_\_\_:

    print(\_\_\_\_)

    print(\_\_\_\_)

    print(\_\_\_\_)

 Read & print the first 3 lines

with open('moby\_dick.txt') as file:

    print(file.readline())

    print(file.readline())

    print(file.readline())

# Read & print the first 3 lines

with open('moby\_dick.txt') as file:

print(file.readline())

print(file.readline())

print(file.readline())

CHAPTER 1. Loomings.

Call me Ishmael. Some years ago--never mind how long precisely--having

**Daily XP300**

# The importance of flat files in data science

**50 XP**

## 1. The importance of flat files in data science

Now you know how to import plain text files,

## 2. Flat files

we're going to look at flat files, such as 'titanic dot csv',

## 3. Flat files

in which each

## 4. Flat files

row is a unique passenger onboard and each

## 5. Flat files

column is a feature of attribute, such as gender, cabin and 'survived or not'. It is essential for any budding data scientist to know precisely what the term flat file means.

## 6. Flat files

Flat files are basic text files containing records, that is, table data, without structured relationships. This is in contrast to a relational database, for example, in which columns of distinct tables can be related. We'll get to these later. To be even more precise, flat files consist of records, where by a record we mean a row of fields or attributes, each of which contains at most one item of information. In the flat file 'titanic dot csv', each

## 7. Flat files

row or record is a unique passenger onboard and each column is a feature or attribute, such as

## 8. Flat files

name, gender and cabin.

## 9. Header

It is also essential to note that a flat file can have a header, such as in 'titanic dot csv', which is a

## 10. Header

row that occurs as the first row and describes the contents of the data columns or states what the corresponding attributes or features in each column are. It will be important to know whether or not your file has a header as it may alter your data import. The reason that flat files are so important in data science is that we data scientists really honestly like to think in records or rows of attributes.

## 11. File extension

Now you may have noticed that the file extension was dot csv. You may be wondering what this is? Well, CSV is an acronym for comma separated value and it means exactly what it says. The values in each row are separated by commas. Another common extension for a flat file is dot txt, which means a text file. Values in flat files can be separated by characters or sequences of characters other than commas, such as a tab, and the character or characters in question is called a delimiter.

## 12. Tab-delimited file

See here an example of a tab-delimited file. The data consists of the famous MNIST digit recognition images, where

## 13. Tab-delimited file

each row contains the pixel values of a given image. Note that all fields in the MNIST data are numeric, while the 'titanic dot csv' also contained strings.

## 14. How do you import flat files?

How do we import such files? If they consist entirely of numbers and we want to store them as a numpy array, we could use numpy. If, instead, we want to store the data in a dataframe, we could use pandas. Most of the time, you will use one of these options. In the rest of this Chapter, you'll learn how to import flat files that contain only numerical data, such as the MNIST data, and import flat files that contain both numerical data and strings, such as 'titanic dot csv'.

## 15. Let's practice!

But first, lets get you to do a couple of quick multiple choice questions to test your knowledge of flat files.

# Pop quiz: examples of flat files

You're now well-versed in importing text files and you're about to become a wiz at importing flat files. But can you remember exactly what a flat file is? Test your knowledge by answering the following question: which of these file types below is NOT an example of a flat file?

##### Answer the question

**50XP**

#### Possible Answers



A .csv file.

press1



A tab-delimited .txt.

press2



A relational database (e.g. PostgreSQL). **This is the answer**

press3

# Pop quiz: what exactly are flat files?

Which of the following statements about flat files is incorrect?

##### Answer the question

**50XP**

#### Possible Answers



Flat files consist of rows and each row is called a record.

press1



Flat files consist of multiple tables with structured relationships between the tables.

press2 **this is the answer**



A record in a flat file is composed of fields or attributes, each of which contains at most one item of information.

press3



Flat files are pervasive in data science.

# Why we like flat files and the Zen of Python

In PythonLand, there are currently hundreds of Python Enhancement Proposals, commonly referred to as PEPs. [PEP8](https://www.python.org/dev/peps/pep-0008/), for example, is a standard style guide for Python, written by our sensei Guido van Rossum himself. It is the basis for how we here at DataCamp ask our instructors to style their code. Another one of my favorites is [PEP20](https://www.python.org/dev/peps/pep-0020/), commonly called the Zen of Python. Its abstract is as follows:

Long time Pythoneer Tim Peters succinctly channels the BDFL's guiding principles for Python's design into 20 aphorisms, only 19 of which have been written down.

If you don't know what the acronym BDFL stands for, I suggest that you look [here](https://docs.python.org/3.3/glossary.html#term-bdfl). You can print the Zen of Python in your shell by typing import this into it! You're going to do this now and the 5th aphorism (line) will say something of particular interest.

The question you need to answer is: **what is the 5th aphorism of the Zen of Python?**

##### Instructions

**50 XP**

##### Possible Answers



**Flat is better than nested. This is the answer**



Flat files are essential for data science.



The world is representable as a flat file.



Flatness is in the eye of the beholder.

In [1]:

import this

The Zen of Python, by Tim Peters

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

**Flat is better than nested.**

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.

There should be one-- and preferably only one --obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

Although never is often better than \*right\* now.

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea -- let's do more of those!

**Daily XP500**

# Importing flat files using NumPy

**50 XP**

## 1. Importing flat files using NumPy

Okay so you now know how to use Python’s built-in open function to open text files. What if you now want to import a flat file and assign it to a variable? If all the data are numerical, you can use the package numpy to import the data as a numpy array. Why would we want to do this?

## 2. Why NumPy?

First off, numpy arrays are the Python standard for storing numerical data. They are efficient, fast and clean.

## 3. Why NumPy?

Secondly, numpy arrays are often essential for other packages, such as scikit-learn, a popular Machine Learning package for Python. Numpy itself has a number of built-in functions that make it far easier and more efficient for us to import data as arrays. Enter the NumPy functions loadtxt and genfromtxt.

## 4. Importing flat files using NumPy

To use either of these we first need to import NumPy. We then call loadtxt and pass it the filename as the first argument, along with the delimiter as the 2nd argument. Note that the default delimiter is any white space so we’ll usually need to specify it explicitly.

## 5. Customizing your NumPy import

There are a number of additional arguments you may wish to specify. If, for example, your data consists of numerics and your header has strings in it, such as in the MNIST digits data, you will want to skip the first row by calling loadtxt with the argument skiprows equals 1; if you want only the 1st and 3rd columns of the data,

## 6. Customizing your NumPy import

you’ll want to set usecols equals the list containing ints 0 and 2.

## 7. Customizing your NumPy import

You can also import different datatypes into NumPy arrays: for example, setting the argument dtype equals 'str' will ensure that all entries are imported as strings. loadtxt is great for basic cases, but tends to break down when we have

## 8. Mixed datatypes

mixed datatypes, for example,

1. 1 Source: Kaggle

## 9. Mixed datatypes

columns consisting of floats AND columns consisting of strings, such as we saw in the Titanic dataset.

1. 1 Source: Kaggle

## 10. Let's practice!

Now it's your turn to have fun with loadtxt. You'll also gain hands-on experience with other functions that can handle mixed datatypes. In the next video we’ll see that, although NumPy arrays can handle data of mixed types, the natural place for such data really is the dataframe.

**Daily XP550**

##### Exercise

##### Exercise

# Using NumPy to import flat files

In this exercise, you're now going to load the MNIST digit recognition dataset using the numpy function loadtxt() and see just how easy it can be:

* The first argument will be the filename.
* The second will be the delimiter which, in this case, is a comma.

You can find more information about the MNIST dataset [here](http://yann.lecun.com/exdb/mnist/) on the webpage of Yann LeCun, who is currently Director of AI Research at Facebook and Founding Director of the NYU Center for Data Science, among many other things.

##### Instructions

**100 XP**

* Fill in the arguments of np.loadtxt() by passing file and a comma ',' for the delimiter.
* Fill in the argument of print() to print the type of the object digits. Use the function type().
* Execute the rest of the code to visualize one of the rows of the data.
* # Import package
* import numpy as np
* # Assign filename to variable: file
* file = 'digits.csv'
* # Load file as array: digits
* digits = np.loadtxt(\_\_\_\_, delimiter=\_\_\_\_)
* # Print datatype of digits
* print(\_\_\_\_)
* # Select and reshape a row
* im = digits[21, 1:]
* im\_sq = np.reshape(im, (28, 28))
* # Plot reshaped data (matplotlib.pyplot already loaded as plt)
* plt.imshow(im\_sq, cmap='Greys', interpolation='nearest')
* plt.show()

# Read & print the first 3 lines with open('moby\_dick.txt') as file: print(file.readline()) print(file.readline()) print(file.readline())

# Import package

import numpy as np

# Assign filename to variable: file

file = 'digits.csv'

# Load file as array: digits

digits = np.loadtxt(file, delimiter=',')

# Print datatype of digits

print(type(digits))

# Select and reshape a row

im = digits[21, 1:]

im\_sq = np.reshape(im, (28, 28))

# Plot reshaped data (matplotlib.pyplot already loaded as plt)

plt.imshow(im\_sq, cmap='Greys', interpolation='nearest')

plt.show()

# Import package

import numpy as np

# Assign filename to variable: file

file = 'digits.csv'

# Load file as array: digits

digits = np.loadtxt(file, delimiter=',')

# Print datatype of digits

print(type(digits))

# Select and reshape a row

im = digits[21, 1:]

im\_sq = np.reshape(im, (28, 28))

# Plot reshaped data (matplotlib.pyplot already loaded as plt)

plt.imshow(im\_sq, cmap='Greys', interpolation='nearest')

plt.show()

<class 'numpy.ndarray'>

**Daily XP650**

##### Exercise

##### Exercise

# Customizing your NumPy import

What if there are rows, such as a header, that you don't want to import? What if your file has a delimiter other than a comma? What if you only wish to import particular columns?

There are a number of arguments that np.loadtxt() takes that you'll find useful:

* delimiter changes the delimiter that loadtxt() is expecting.
  + You can use ',' for comma-delimited.
  + You can use '\t' for tab-delimited.
* skiprows allows you to specify how many rows (not indices) you wish to skip
* usecols takes a list of the indices of the columns you wish to keep.

The file that you'll be importing, digits\_header.txt, has a header and is tab-delimited.

##### Instructions

**100 XP**

* Complete the arguments of np.loadtxt(): the file you're importing is tab-delimited, you want to skip the first row and you only want to import the first and third columns.
* Complete the argument of the print() call in order to print the entire array that you just imported.
* # Import numpy
* import numpy as np
* # Assign the filename: file
* file = 'digits\_header.txt'
* # Load the data: data
* data = np.loadtxt(\_\_\_\_, delimiter=\_\_\_\_, skiprows=\_\_\_\_, usecols=\_\_\_\_)
* # Print data
* print(\_\_\_\_)

# Import package import numpy as np # Assign filename to variable: file file = 'digits.csv' # Load file as array: digits digits = np.loadtxt(file, delimiter=',') # Print datatype of digits print(type(digits)) # Select and reshape a row im = digits[21, 1:] im\_sq = np.reshape(im, (28, 28)) # Plot reshaped data (matplotlib.pyplot already loaded as plt) plt.imshow(im\_sq, cmap='Greys', interpolation='nearest') plt.show()

# Import numpy

import numpy as np

# Assign the filename: file

file = 'digits\_header.txt'

# Load the data: data

data = np.loadtxt(file, delimiter='\t', skiprows=1, usecols=[0,2])

# Print data

print(data)

# Import numpy

import numpy as np

# Assign the filename: file

file = 'digits\_header.txt'

# Load the data: data

data = np.loadtxt(file, delimiter='\t', skiprows=1, usecols=[0,2])

# Print data

print(data)

[[1. 0.]

[0. 0.]

[1. 0.]

[4. 0.]

[0. 0.]

[0. 0.]

[7. 0.]

[3. 0.]

[5. 0.]

[3. 0.]

[8. 0.]

[9. 0.]

[1. 0.]

[3. 0.]

[3. 0.]

[1. 0.]

[2. 0.]

[0. 0.]

[7. 0.]

[5. 0.]

[8. 0.]

[6. 0.]

[2. 0.]

[0. 0.]

[2. 0.]

[3. 0.]

[6. 0.]

[9. 0.]

[9. 0.]

[7. 0.]

[8. 0.]

[9. 0.]

[4. 0.]

[9. 0.]

[2. 0.]

[1. 0.]

[3. 0.]

[1. 0.]

[1. 0.]

[4. 0.]

[9. 0.]

[1. 0.]

[4. 0.]

[4. 0.]

[2. 0.]

[6. 0.]

[3. 0.]

[7. 0.]

[7. 0.]

[4. 0.]

[7. 0.]

[5. 0.]

[1. 0.]

[9. 0.]

[0. 0.]

[2. 0.]

[2. 0.]

[3. 0.]

[9. 0.]

[1. 0.]

[1. 0.]

[1. 0.]

[5. 0.]

[0. 0.]

[6. 0.]

[3. 0.]

[4. 0.]

[8. 0.]

[1. 0.]

[0. 0.]

[3. 0.]

[9. 0.]

[6. 0.]

[2. 0.]

[6. 0.]

[4. 0.]

[7. 0.]

[1. 0.]

[4. 0.]

[1. 0.]

[5. 0.]

[4. 0.]

[8. 0.]

[9. 0.]

[2. 0.]

[9. 0.]

[9. 0.]

[8. 0.]

[9. 0.]

[6. 0.]

[3. 0.]

[6. 0.]

[4. 0.]

[6. 0.]

[2. 0.]

[9. 0.]

[1. 0.]

[2. 0.]

[0. 0.]

[5. 0.]]

**Daily XP750**

##### Exercise

##### Exercise

# Importing different datatypes

The file seaslug.txt

* has a text header, consisting of strings
* is tab-delimited.

These data consists of percentage of sea slug larvae that had metamorphosed in a given time period. Read more [here](http://www.stat.ucla.edu/~rgould/datasets/aboutseaslugs.html).

Due to the header, if you tried to import it as-is using np.loadtxt(), Python would throw you a ValueError and tell you that it could not convert string to float. There are two ways to deal with this: firstly, you can set the data type argument dtype equal to str (for string).

Alternatively, you can skip the first row as we have seen before, using the skiprows argument.

##### Instructions

**100 XP**

* Complete the first call to np.loadtxt() by passing file as the first argument.
* Execute print(data[0]) to print the first element of data.
* Complete the second call to np.loadtxt(). The file you're importing is tab-delimited, the datatype is float, and you want to skip the first row.
* Print the 10th element of data\_float by completing the print() command. Be guided by the previous print() call.
* Execute the rest of the code to visualize the data.
* # Assign filename: file
* file = 'seaslug.txt'
* # Import file: data
* data = np.loadtxt(\_\_\_\_, delimiter='\t', dtype=str)
* # Print the first element of data
* print(data[0])
* # Import data as floats and skip the first row: data\_float
* data\_float = np.loadtxt(\_\_\_\_, delimiter=\_\_\_\_, dtype=\_\_\_\_, skiprows=\_\_\_\_)
* # Print the 10th element of data\_float
* print(\_\_\_\_)
* # Plot a scatterplot of the data
* plt.scatter(data\_float[:, 0], data\_float[:, 1])
* plt.xlabel('time (min.)')
* plt.ylabel('percentage of larvae')
* plt.show()

# Import numpy import numpy as np # Assign the filename: file file = 'digits\_header.txt' # Load the data: data data = np.loadtxt(file, delimiter='\t', skiprows=1, usecols=[0,2]) # Print data print(data)

# Assign filename: file

file = 'seaslug.txt'

# Import file: data

data = np.loadtxt(file, delimiter='\t', dtype=str)

# Print the first element of data

print(data[0])

# Import data as floats and skip the first row: data\_float

data\_float = np.loadtxt(file, delimiter='\t', dtype=float, skiprows=1)

# Print the 10th element of data\_float

print(data\_float[9])

# Plot a scatterplot of the data

plt.scatter(data\_float[:, 0], data\_float[:, 1])

plt.xlabel('time (min.)')

plt.ylabel('percentage of larvae')

plt.show()

# Assign filename: file

file = 'seaslug.txt'

# Import file: data

data = np.loadtxt(file, delimiter='\t', dtype=str)

# Print the first element of data

print(data[0])

# Import data as floats and skip the first row: data\_float

data\_float = np.loadtxt(file, delimiter='\t', dtype=float, skiprows=1)

# Print the 10th element of data\_float

print(data\_float[9])

# Plot a scatterplot of the data

plt.scatter(data\_float[:, 0], data\_float[:, 1])

plt.xlabel('time (min.)')

plt.ylabel('percentage of larvae')

plt.show()

['Time' 'Percent']

[0. 0.357]

**Daily XP850**

##### Exercise

##### Exercise

# Working with mixed datatypes (1)

Much of the time you will need to import datasets which have different datatypes in different columns; one column may contain strings and another floats, for example. The function np.loadtxt() will freak at this. There is another function, np.genfromtxt(), which can handle such structures. If we pass dtype=None to it, it will figure out what types each column should be.

Import 'titanic.csv' using the function np.genfromtxt() as follows:

data = np.genfromtxt('titanic.csv', delimiter=',', names=True, dtype=None)

Here, the first argument is the filename, the second specifies the delimiter , and the third argument names tells us there is a header. Because the data are of different types, data is an object called a [structured array](http://docs.scipy.org/doc/numpy/user/basics.rec.html). Because numpy arrays have to contain elements that are all the same type, the structured array solves this by being a 1D array, where each element of the array is a row of the flat file imported. You can test this by checking out the array's shape in the shell by executing np.shape(data).

Accessing rows and columns of structured arrays is super-intuitive: to get the ith row, merely execute data[i] and to get the column with name 'Fare', execute data['Fare'].

After importing the Titanic data as a structured array (as per the instructions above), print the entire column with the name Survived to the shell. What are the last 4 values of this column?

##### Instructions

**50 XP**

##### Possible Answers



1,0,0,1.



1,2,0,0.



1,0,1,0. **This is the answer**



0,1,1,1.

# Assign filename: file file = 'seaslug.txt' # Import file: data data = np.loadtxt(file, delimiter='\t', dtype=str) # Print the first element of data print(data[0]) # Import data as floats and skip the first row: data\_float data\_float = np.loadtxt(file, delimiter='\t', dtype=float, skiprows=1) # Print the 10th element of data\_float print(data\_float[9]) # Plot a scatterplot of the data plt.scatter(data\_float[:, 0], data\_float[:, 1]) plt.xlabel('time (min.)') plt.ylabel('percentage of larvae') plt.show()

[2]:

data = np.genfromtxt('titanic.csv', delimiter=',', names=True, dtype=None)

In [3]:

np.shape(data)

Out[3]:

(891,)

In [4]:

print(data['Survived'])

##### Exercise

# Working with mixed datatypes (2)

You have just used np.genfromtxt() to import data containing mixed datatypes. There is also another function np.recfromcsv() that behaves similarly to np.genfromtxt(), except that its default dtype is None. In this exercise, you'll practice using this to achieve the same result.

##### Instructions

**100 XP**

* Import titanic.csv using the function np.recfromcsv() and assign it to the variable, d. You'll only need to pass file to it because it has the defaults delimiter=',' and names=True in addition to dtype=None!
* Run the remaining code to print the first three entries of the resulting array d.

 Assign the filename: file

file = 'titanic.csv'

# Import file using np.recfromcsv: d

# Print out first three entries of d

print(d[:3])

# Assign the filename: file

file = 'titanic.csv'

# Import file using np.recfromcsv: d

d = np.recfromcsv(file)

# Print out first three entries of d

print(d[:3])

# Assign the filename: file

file = 'titanic.csv'

# Import file using np.recfromcsv: d

d = np.recfromcsv(file)

# Print out first three entries of d

print(d[:3])

[(1, 0, 3, b'male', 22., 1, 0, b'A/5 21171', 7.25 , b'', b'S')

(2, 1, 1, b'female', 38., 1, 0, b'PC 17599', 71.2833, b'C85', b'C')

(3, 1, 3, b'female', 26., 0, 0, b'STON/O2. 3101282', 7.925 , b'', b'S')]

**Daily XP1050**

##### Exercise

##### Exercise

# Using pandas to import flat files as DataFrames (1)

In the last exercise, you were able to import flat files containing columns with different datatypes as numpy arrays. However, the DataFrame object in pandas is a more appropriate structure in which to store such data and, thankfully, we can easily import files of mixed data types as DataFrames using the pandas functions read\_csv() and read\_table().

##### Instructions

**100 XP**

* Import the pandas package using the alias pd.
* Read titanic.csv into a DataFrame called df. The file name is already stored in the file object.
* In a print() call, view the head of the DataFrame.

# Import pandas as pd

# Assign the filename: file

file = 'titanic.csv'

# Read the file into a DataFrame: df

df = pd.read\_csv(\_\_\_\_)

# View the head of the DataFrame

# Assign the filename: file file = 'titanic.csv' # Import file using np.recfromcsv: d d = np.recfromcsv(file) # Print out first three entries of d print(d[:3])

# Import pandas as pd

import pandas as pd

# Assign the filename: file

file = 'titanic.csv'

# Read the file into a DataFrame: df

df = pd.read\_csv(file)

# View the head of the DataFrame

print(df.head())

# Import pandas as pd

import pandas as pd

# Assign the filename: file

file = 'titanic.csv'

# Read the file into a DataFrame: df

df = pd.read\_csv(file)

# View the head of the DataFrame

print(df.head())

PassengerId Survived Pclass Sex Age ... Parch Ticket Fare Cabin Embarked

0 1 0 3 male 22.0 ... 0 A/5 21171 7.250 NaN S

1 2 1 1 female 38.0 ... 0 PC 17599 71.283 C85 C

2 3 1 3 female 26.0 ... 0 STON/O2. 3101282 7.925 NaN S

3 4 1 1 female 35.0 ... 0 113803 53.100 C123 S

4 5 0 3 male 35.0 ... 0 373450 8.050 NaN S

[5 rows x 11 columns]

**Daily XP1150**

##### Exercise

##### Exercise

# Using pandas to import flat files as DataFrames (2)

In the last exercise, you were able to import flat files into a pandas DataFrame. As a bonus, it is then straightforward to retrieve the corresponding numpy array using the attribute values. You'll now have a chance to do this using the MNIST dataset, which is available as digits.csv.

##### Instructions

**100 XP**

* Import the first 5 rows of the file into a DataFrame using the function pd.read\_csv() and assign the result to data. You'll need to use the arguments nrows and header (there is no header in this file).
* Build a numpy array from the resulting DataFrame in data and assign to data\_array.
* Execute print(type(data\_array)) to print the datatype of data\_array.
* # Assign the filename: file
* file = 'digits.csv'
* # Read the first 5 rows of the file into a DataFrame: data
* # Build a numpy array from the DataFrame: data\_array
* # Print the datatype of data\_array to the shell
* print(type(data\_array))

# Import pandas as pd import pandas as pd # Assign the filename: file file = 'titanic.csv' # Read the file into a DataFrame: df df = pd.read\_csv(file) # View the head of the DataFrame print(df.head())

# Assign the filename: file

file = 'digits.csv'

# Read the first 5 rows of the file into a DataFrame: data

data = pd.read\_csv(file, nrows=5, header=None)

# Build a numpy array from the DataFrame: data\_array

data\_array = data.values

# Print the datatype of data\_array to the shell

print(type(data\_array))

# Assign the filename: file

file = 'digits.csv'

# Read the first 5 rows of the file into a DataFrame: data

data = pd.read\_csv(file, nrows=5, header=None)

# Build a numpy array from the DataFrame: data\_array

data\_array = data.values

# Print the datatype of data\_array to the shell

print(data\_array)

[[1. 0. 0. ... 0. 0. 0.]

[0. 0. 0. ... 0. 0. 0.]

[1. 0. 0. ... 0. 0. 0.]

[4. 0. 0. ... 0. 0. 0.]

[0. 0. 0. ... 0. 0. 0.]]

# Assign the filename: file

file = 'digits.csv'

# Read the first 5 rows of the file into a DataFrame: data

data = pd.read\_csv(file, nrows=5, header=None)

# Build a numpy array from the DataFrame: data\_array

data\_array = data.values

# Print the datatype of data\_array to the shell

print(type(data\_array))

<class 'numpy.ndarray'>

##### Exercise

# Customizing your pandas import

The pandas package is also great at dealing with many of the issues you will encounter when importing data as a data scientist, such as comments occurring in flat files, empty lines and missing values. Note that missing values are also commonly referred to as NA or NaN. To wrap up this chapter, you're now going to import a slightly corrupted copy of the Titanic dataset titanic\_corrupt.txt, which

* contains comments after the character '#'
* is tab-delimited.

##### Instructions

**100 XP**

* Complete the sep (the pandas version of delim), comment and na\_values arguments of pd.read\_csv(). comment takes characters that comments occur after in the file, which in this case is '#'. na\_values takes a list of strings to recognize as NA/NaN, in this case the string 'Nothing'.
* Execute the rest of the code to print the head of the resulting DataFrame and plot the histogram of the 'Age' of passengers aboard the Titanic
* # Import matplotlib.pyplot as plt
* import matplotlib.pyplot as plt
* # Assign filename: file
* file = 'titanic\_corrupt.txt'
* # Import file: data
* data = pd.read\_csv(file, sep=\_\_\_\_, comment=\_\_\_\_, na\_values=\_\_\_\_)
* # Print the head of the DataFrame
* print(data.head())
* # Plot 'Age' variable in a histogram
* pd.DataFrame.hist(data[['Age']])
* plt.xlabel('Age (years)')
* plt.ylabel('count')
* plt.show()

# Assign the filename: file file = 'digits.csv' # Read the first 5 rows of the file into a DataFrame: data data = pd.read\_csv(file, nrows=5, header=None) # Build a numpy array from the DataFrame: data\_array data\_array = data.values # Print the datatype of data\_array to the shell print(type(data\_array))

# Import matplotlib.pyplot as plt

import matplotlib.pyplot as plt

# Assign filename: file

file = 'titanic\_corrupt.txt'

# Import file: data

data = pd.read\_csv(file, sep='\t', comment='#', na\_values='Nothing')

# Print the head of the DataFrame

print(data.head())

# Plot 'Age' variable in a histogram

pd.DataFrame.hist(data[['Age']])

plt.xlabel('Age (years)')

plt.ylabel('count')

plt.show()

# Import matplotlib.pyplot as plt

import matplotlib.pyplot as plt

# Assign filename: file

file = 'titanic\_corrupt.txt'

# Import file: data

data = pd.read\_csv(file, sep='\t', comment='#', na\_values='Nothing')

# Print the head of the DataFrame

print(data.head())

# Plot 'Age' variable in a histogram

pd.DataFrame.hist(data[['Age']])

plt.xlabel('Age (years)')

plt.ylabel('count')

plt.show()

PassengerId Survived Pclass Sex Age ... Parch Ticket Fare Cabin Embarked

0 1 0 3 male 22.0 ... 0 A/5 21171 7.250 NaN S

1 2 1 1 female 38.0 ... 0 PC 17599 NaN NaN NaN

2 3 1 3 female 26.0 ... 0 STON/O2. 3101282 7.925 NaN S

3 4 1 1 female 35.0 ... 0 113803 53.100 C123 S

4 5 0 3 male 35.0 ... 0 373450 8.050 NaN S

[5 rows x 11 columns]

## 1. Final thoughts on data import

We have seen a number of ways to read, print and import flat files. As a data scientist, you will most often wish to use pandas, however it was important to check out all the possible ways to import because you never know when they will be useful.

## 2. Next chapters:

In the next chapter, we'll see just how useful pandas can be when attempting to import a variety of other file types, such as Excel spreadsheets, along with native SAS & Stata files. It is also important to remember that, due to the active development community in open source softwares, there is constant activity in file formats and ways to import data: for example, on March 29, 2016, Wes McKinney, the creator of pandas, and Hadley Wickham, of R development fame, announced a new and fast on-disk format for dataframes for R and Python, called feather . As dataframes are one the most important data structures for data scientists, let's definitely keep our eyes on feather. After learning to import many other file types in the next chapter, you'll learn how to interact with relational databases in Python.

## 3. Next course:

Then, in the sequel to this course, you'll learn how to tear all types of data down from the web and how to interact with APIs to fulfil your big data fix. These are all essential techniques for the modern day Data Scientist to master, and the upcoming Chapters will place you in good stead to becoming

## 4. Let's practice!

an Importing Data ninja Pythonista.

**Daily XP1450**

# Introduction to other file types

**50 XP**

## 1. Introduction to other file types

Now that you have mastered the art of importing flat files in Python, it is time to check out a number of other file types that you will find yourself needing to work with as a data scientist.

## 2. Other file types

In this chapter, you will learn how to import Excel spreadsheets, which professionals from all disciplines use to store their data. You will also gain familiarity with importing MATLAB, SAS and Stata files, which are commonplace. You will also learn how to import HDF5 files and you'll actually import an HDF5 file containing data from the Laser Interferometer Gravitational-Wave Observatory project that provided empirical support for Einstein's Theory of Gravitational Waves in 2016. HDF5 files are becoming a more prevalent way to store large datasets, as demonstrated by the fact that the LIGO researchers use it to store their data.

## 3. Pickled files

Another file type you'll learn about in this Chapter is that of a 'pickled' file. This is a file type native to Python. The concept of pickling a file is motivated by the following: while it may be easy to save a numpy array or a pandas dataframe to a flat file, there are many other datatypes, such as dictionaries and lists, for which it isn't obvious how to store them. 'Pickle' to the rescue! If you want your files to be human readable, you may want to save them as text files in a clever manner (JSONs, which you will see in a later chapter, are appropriate for Python dictionaries). If, however, you merely want to be able to import them into Python, you can serialize them. All this means is converting the object into a sequence of bytes, or bytestream. As this is a course in Importing Data in Python,

## 4. Pickled files

you'll learn how to import files that have already been pickled. As you have done before, when opening such a file, you'll want to specify that it is read only; you'll also want to specify that it is a binary file, meaning that it is computer-readable and not human-readable. To specify both read only and binary, you'll want pass the string 'rb' as the second argument of open.

## 5. Importing Excel spreadsheets

You'll then dive head-first into Excel spreadsheets, the use of which is so widespread that they need next to no introduction at all. An Excel file generally consists of a number of sheets. There are many ways to import Excel files and you'll use pandas to do so because it produces dataframes natively, which is great for your practice as a Data Scientist. As you can see in this example, you can use the functionExcelfile to assign an Excel file to a variable data. As an Excel file consists of sheets, the first thing to do is figure out what the sheets are. This is straightforward with the command 'data dot sheet\_names'. To then load a particular sheet as a dataframe, you need only apply the method parse to the object data with a single argument, which is either the name as a string or the index as a float of the sheet that you wish to load: pandas is clever enough to know if you're telling it the sheet name or the index!

## 6. You’ll learn:

You'll also learn how to customize your spreadsheet import in order to skip rows, import only certain columns and to change the column names. That's enough from me,

## 7. Let's practice!

it's now time to get your hands dirty with pickled files and Excel spreadsheets. Enjoy!

##### Exercise

# Not so flat any more

In Chapter 1, you learned how to use the IPython magic command ! ls to explore your current working directory. You can also do this natively in Python using the [library os](https://docs.python.org/2/library/os.html), which consists of miscellaneous operating system interfaces.

The first line of the following code imports the library os, the second line stores the name of the current directory in a string called wd and the third outputs the contents of the directory in a list to the shell.

import os

wd = os.getcwd()

os.listdir(wd)

Run this code in the IPython shell and answer the following questions. Ignore the files that begin with ..

Check out the contents of your current directory and answer the following questions: (1) which file is in your directory and NOT an example of a flat file; (2) why is it not a flat file?

##### Instructions

**50 XP**

##### Possible Answers



database.db is not a flat file because relational databases contain structured relationships and flat files do not.



battledeath.xlsx is not a flat because it is a spreadsheet consisting of many sheets, not a single table**. This is the answer**



titanic.txt is not a flat file because it is a .txt, not a .csv.

# Import matplotlib.pyplot as plt import matplotlib.pyplot as plt # Assign filename: file file = 'titanic\_corrupt.txt' # Import file: data data = pd.read\_csv(file, sep='\t', comment='#', na\_values='Nothing') # Print the head of the DataFrame print(data.head()) # Plot 'Age' variable in a histogram pd.DataFrame.hist(data[['Age']]) plt.xlabel('Age (years)') plt.ylabel('count') plt.show()

**Daily XP50**

##### Exercise

##### Exercise

# Loading a pickled file

There are a number of datatypes that cannot be saved easily to flat files, such as lists and dictionaries. If you want your files to be human readable, you may want to save them as text files in a clever manner. JSONs, which you will see in a later chapter, are appropriate for Python dictionaries.

However, if you merely want to be able to import them into Python, you can [serialize](https://en.wikipedia.org/wiki/Serialization) them. All this means is converting the object into a sequence of bytes, or a bytestream.

In this exercise, you'll import the pickle package, open a previously pickled data structure from a file and load it.

##### Instructions

**100 XP**

* Import the pickle package.
* Complete the second argument of open() so that it is read only for a binary file. This argument will be a string of two letters, one signifying 'read only', the other 'binary'.
* Pass the correct argument to pickle.load(); it should use the variable that is bound to open.
* Print the data, d.
* Print the datatype of d; take your mind back to your previous use of the function type().
* # Import pickle package
* # Open pickle file and load data: d
* with open('data.pkl', \_\_\_\_) as file:
* d = pickle.load(\_\_\_\_)
* # Print d
* print(\_\_\_\_)
* # Print datatype of d
* print(\_\_\_\_)

# Import pickle package

import pickle

# Open pickle file and load data: d

with open('data.pkl', 'rb') as file:

    d = pickle.load(file)

# Print d

print(d)

# Print datatype of d

print(type(d))

# Import pickle package

import pickle

# Open pickle file and load data: d

with open('data.pkl', 'rb') as file:

d = pickle.load(file)

# Print d

print(d)

# Print datatype of d

print(type(d))

{'June': '69.4', 'Aug': '85', 'Airline': '8', 'Mar': '84.4'}

<class 'dict'>

**Daily XP150**

##### Exercise

##### Exercise

# Listing sheets in Excel files

Whether you like it or not, any working data scientist will need to deal with Excel spreadsheets at some point in time. You won't always want to do so in Excel, however!

Here, you'll learn how to use pandas to import Excel spreadsheets and how to list the names of the sheets in any loaded .xlsx file.

Recall from the video that, given an Excel file imported into a variable spreadsheet, you can retrieve a list of the sheet names using the attribute spreadsheet.sheet\_names.

Specifically, you'll be loading and checking out the spreadsheet 'battledeath.xlsx', modified from the Peace Research Institute Oslo's (PRIO) [dataset](https://www.prio.org/Data/Armed-Conflict/Battle-Deaths/The-Battle-Deaths-Dataset-version-30/). This data contains age-adjusted mortality rates due to war in various countries over several years.

##### Instructions

**100 XP**

* Assign the spreadsheet filename (provided above) to the variable file.
* Pass the correct argument to pd.ExcelFile() to load the file using pandas, assigning the result to the variable xls.
* Print the sheetnames of the Excel spreadsheet by passing the necessary argument to the print() function.

# Import pickle package import pickle # Open pickle file and load data: d with open('data.pkl', 'rb') as file: d = pickle.load(file) # Print d print(d) # Print datatype of d print(type(d))

# Import pandas

import pandas as pd

# Assign spreadsheet filename: file

file = \_\_\_\_

# Load spreadsheet: xls

xls = pd.ExcelFile(\_\_\_\_)

# Print sheet names

print(\_\_\_\_)

# Import pandas

import pandas as pd

# Assign spreadsheet filename: file

file = 'battledeath.xlsx'

# Load spreadsheet: xls

xls = pd.ExcelFile(file)

# Print sheet names

print(xls.sheet\_names)

# Import pandas

import pandas as pd

# Assign spreadsheet filename: file

file = 'battledeath.xlsx'

# Load spreadsheet: xls

xls = pd.ExcelFile(file)

# Print sheet names

print(xls.sheet\_names)

['2002', '2004']

##### Exercise

# Importing sheets from Excel files

In the previous exercises, you saw that the Excel file contains two sheets, '2002' and '2004'. The next step is to import these.

In this exercise, you'll learn how to import any given sheet of your loaded .xlsx file as a DataFrame. You'll be able to do so by specifying either the sheet's name or its index.

The spreadsheet 'battledeath.xlsx' is already loaded as xls.

##### Instructions

**100 XP**

* Load the sheet '2004' into the DataFrame df1 using its name as a string.
* Print the head of df1 to the shell.
* Load the sheet 2002 into the DataFrame df2 using its index (0).
* Print the head of df2 to the shell.
* # Load a sheet into a DataFrame by name: df1
* df1 = xls.parse(\_\_\_\_)
* # Print the head of the DataFrame df1
* print(\_\_\_\_)
* # Load a sheet into a DataFrame by index: df2
* # Print the head of the DataFrame df2
* print(\_\_\_\_)

# Import pandas import pandas as pd # Assign spreadsheet filename: file file = 'battledeath.xlsx' # Load spreadsheet: xls xls = pd.ExcelFile(file) # Print sheet names print(xls.sheet\_names)

# Load a sheet into a DataFrame by name: df1

df1 = xls.parse('2004')

# Print the head of the DataFrame df1

print(df1.head())

# Load a sheet into a DataFrame by index: df2

df2 = xls.parse(0)

# Print the head of the DataFrame df2

print(df2.head())

# Import pandas

import pandas as pd

# Assign spreadsheet filename: file

file = 'battledeath.xlsx'

# Load spreadsheet: xls

xls = pd.ExcelFile(file)

# Print sheet names

print(xls.sheet\_names)

# Load a sheet into a DataFrame by name: df1

df1 = xls.parse('2004')

# Print the head of the DataFrame df1

print(df1.head())

# Load a sheet into a DataFrame by index: df2

df2 = xls.parse(0)

# Print the head of the DataFrame df2

print(df2.head())

War(country) 2004

0 Afghanistan 9.451

1 Albania 0.130

2 Algeria 3.407

3 Andorra 0.000

4 Angola 2.598

War, age-adjusted mortality due to 2002

0 Afghanistan 36.084

1 Albania 0.129

2 Algeria 18.314

3 Andorra 0.000

4 Angola 18.965

Awesome! You'll typically find yourself referring to the Excel sheet by name, but it's good to know you can also use indexes.

**Daily XP350**

##### Exercise

##### Exercise

# Customizing your spreadsheet import

Here, you'll parse your spreadsheets and use additional arguments to skip rows, rename columns and select only particular columns.

The spreadsheet 'battledeath.xlsx' is already loaded as xls.

As before, you'll use the method parse(). This time, however, you'll add the additional arguments skiprows, names and usecols. These skip rows, name the columns and designate which columns to parse, respectively. All these arguments can be assigned to lists containing the specific row numbers, strings and column numbers, as appropriate.

##### Instructions

**100 XP**

* Parse the first sheet by index. In doing so, skip the first row of data and name the columns 'Country' and 'AAM due to War (2002)' using the argument names. The values passed to skiprows and names all need to be of type list.
* Parse the second sheet by index. In doing so, parse only the first column with the usecols parameter, skip the first row and rename the column 'Country'. The argument passed to usecols also needs to be of type list.

# Load a sheet into a DataFrame by name: df1 df1 = xls.parse('2004') # Print the head of the DataFrame df1 print(df1.head()) # Load a sheet into a DataFrame by index: df2 df2 = xls.parse(0) # Print the head of the DataFrame df2 print(df2.head())

# Parse the first sheet and rename the columns: df1

df1 = xls.parse(\_\_\_\_, skiprows=\_\_\_\_, names=\_\_\_\_)

# Print the head of the DataFrame df1

print(df1.head())

# Parse the first column of the second sheet and rename the column: df2

df2 = xls.parse(\_\_\_\_, usecols=\_\_\_\_, skiprows=\_\_\_\_, names=\_\_\_\_)

# Print the head of the DataFrame df2

print(df2.head())

# Parse the first sheet and rename the columns: df1

df1 = xls.parse(0, skiprows=[0], names=['Country','AAM due to War (2002)'])

# Print the head of the DataFrame df1

print(df1.head())

# Parse the first column of the second sheet and rename the column: df2

df2 = xls.parse(1, usecols=[0], skiprows=[0], names=['Country'])

# Print the head of the DataFrame df2

print(df2.head())

# Parse the first sheet and rename the columns: df1

df1 = xls.parse(0, skiprows=[0], names=['Country','AAM due to War (2002)'])

# Print the head of the DataFrame df1

print(df1.head())

# Parse the first column of the second sheet and rename the column: df2

df2 = xls.parse(1, usecols=[0], skiprows=[0], names=['Country'])

# Print the head of the DataFrame df2

print(df2.head())

ERROR! Session/line number was not unique in database. History logging moved to new session 22

Country AAM due to War (2002)

0 Albania 0.129

1 Algeria 18.314

2 Andorra 0.000

3 Angola 18.965

4 Antigua and Barbuda 0.000

Country

0 Albania

1 Algeria

2 Andorra

3 Angola

4 Antigua and Barbuda

## 1. Importing SAS/Stata files using pandas

There are many statistical software packages out there and, although you may not need to do so all the time, it will be important for you, as a working Data Scientist, to be able to import these files into your Python environment.

## 2. SAS and Stata files

The most common examples are SAS, which is an acronym for 'Statistical Analysis System', and Stata, which is a contraction of 'Statistics' and 'Data'. The former is used a great deal in business analytics and biostatistics, while the latter is popular in academic social sciences research, such as economics and epidemiology.

## 3. SAS files

SAS files are important because SAS is a software suite that performs advanced analytics, multivariate analyses, business intelligence, data management, predictive analytics and is a standard for statisticians to do computational analysis.

## 4. Importing SAS files

The most common SAS files have the extension dot sas7bdat and dot sas7bcat, which are dataset files and catalog files respectively. You'll learn how to import the former as dataframes using the function SAS7BDAT (upper case) from the package sas7bdat (lower case). In this case, you can bind the variable file to a connection to the file 'urbanpop dot sas7bdat' in a context manager. Within this context, you can assign to a variable df\_sas the result of applying method to\_data\_frame to file.

## 5. Importing Stata files

Stata files have extension dot dta and we can import them using pandas. We don't even need to initialize a context manager in this case! We merely pass the filename to the function read\_stata and assign it to a variable, just like this. In the following exercises, you'll gain invaluable experience at importing these important file formats in Python as pandas dataframes and then seeing what was inside them.

## 6. Let's practice!

Now it's your turn, happy importing!

# How to import SAS7BDAT

How do you correctly import the function SAS7BDAT() from the package sas7bdat?

##### Answer the question

**50XP**

#### Possible Answers



import SAS7BDAT from sas7bdat

press1



from SAS7BDAT import sas7bdat

press2



import sas7bdat from SAS7BDAT

press3



**from sas7bdat import SAS7BDAT this is the answer**

press4

##### e

##### Exercise

# Importing SAS files

In this exercise, you'll figure out how to import a SAS file as a DataFrame using SAS7BDAT and pandas. The file 'sales.sas7bdat' is already in your working directory and both pandas and matplotlib.pyplot have already been imported as follows:

import pandas as pd

import matplotlib.pyplot as plt

The data are adapted from the website of the undergraduate text book [Principles of Econometrics](http://www.principlesofeconometrics.com/) by Hill, Griffiths and Lim.

##### Instructions

**100 XP**

* Import the module SAS7BDAT from the library sas7bdat.
* In the context of the file 'sales.sas7bdat', load its contents to a DataFrame df\_sas, using the method to\_data\_frame() on the object file.
* Print the head of the DataFrame df\_sas.
* Execute your entire script to produce a histogram plot!

# Parse the first sheet and rename the columns: df1 df1 = xls.parse(0, skiprows=[0], names=['Country','AAM due to War (2002)']) # Print the head of the DataFrame df1 print(df1.head()) # Parse the first column of the second sheet and rename the column: df2 df2 = xls.parse(1, usecols=[0], skiprows=[0], names=['Country']) # Print the head of the DataFrame df2 print(df2.head())

# Import sas7bdat package

from \_\_\_\_ import \_\_\_\_

# Save file to a DataFrame: df\_sas

with SAS7BDAT('sales.sas7bdat') as file:

    \_\_\_\_

# Print head of DataFrame

# Plot histogram of DataFrame features (pandas and pyplot already imported)

pd.DataFrame.hist(df\_sas[['P']])

plt.ylabel('count')

plt.show()

# Import sas7bdat package

from sas7bdat import SAS7BDAT

# Save file to a DataFrame: df\_sas

with SAS7BDAT('sales.sas7bdat') as file:

    df\_sas = file.to\_data\_frame()

# Print head of DataFrame

print(df\_sas.head())

# Plot histogram of DataFrame features (pandas and pyplot already imported)

pd.DataFrame.hist(df\_sas[['P']])

plt.ylabel('count')

plt.show()

# Import sas7bdat package

from sas7bdat import SAS7BDAT

# Save file to a DataFrame: df\_sas

with SAS7BDAT('sales.sas7bdat') as file:

df\_sas = file.to\_data\_frame()

# Print head of DataFrame

print(df\_sas.head())

# Plot histogram of DataFrame features (pandas and pyplot already imported)

pd.DataFrame.hist(df\_sas[['P']])

plt.ylabel('count')

plt.show()

YEAR P S

0 1950.0 12.9 181.9

1 1951.0 11.9 245.0

2 1952.0 10.7 250.2

3 1953.0 11.3 265.9

4 1954.0 11.2 248.5

**Daily XP650**

##### Exercise

##### Exercise

# Using read\_stata to import Stata files

The pandas package has been imported in the environment as pd and the file disarea.dta is in your working directory. The data consist of disease extents for several diseases in various countries (more information can be found [here](http://www.cid.harvard.edu/ciddata/geog/readme_disarea.html)).

What is the correct way of using the read\_stata() function to import disarea.dta into the object df?

##### Instructions

**50 XP**

##### Possible Answers



df = 'disarea.dta'



df = read\_stata.pd('disarea.dta')



df = pd.read\_stata('disarea.dta')



df = pd.read\_stata(disarea.dta)

# Import sas7bdat package from sas7bdat import SAS7BDAT # Save file to a DataFrame: df\_sas with SAS7BDAT('sales.sas7bdat') as file: df\_sas = file.to\_data\_frame() # Print head of DataFrame print(df\_sas.head()) # Plot histogram of DataFrame features (pandas and pyplot already imported) pd.DataFrame.hist(df\_sas[['P']]) plt.ylabel('count') plt.show()

**Daily XP700**

##### Exercise

##### Exercise

# Importing Stata files

Here, you'll gain expertise in importing Stata files as DataFrames using the pd.read\_stata() function from pandas. The last exercise's file, 'disarea.dta', is still in your working directory.

##### Instructions

**100 XP**

* Use pd.read\_stata() to load the file 'disarea.dta' into the DataFrame df.
* Print the head of the DataFrame df.

Visualize your results by plotting a histogram of the column disa10. We’ve already provided this # Import pandas

import pandas as pd

# Load Stata file into a pandas DataFrame: df

# Print the head of the DataFrame df

# Plot histogram of one column of the DataFrame

pd.DataFrame.hist(df[['disa10']])

plt.xlabel('Extent of disease')

plt.ylabel('Number of countries')

plt.show()

* code for you, so just run it!
* # Import pandas
* import pandas as pd
* # Load Stata file into a pandas DataFrame: df
* df = pd.read\_stata('disarea.dta')
* # Print the head of the DataFrame df
* print(df.head())
* # Plot histogram of one column of the DataFrame
* pd.DataFrame.hist(df[['disa10']])
* plt.xlabel('Extent of disease')
* plt.ylabel('Number of countries')
* plt.show()

# Import pandas

import pandas as pd

# Load Stata file into a pandas DataFrame: df

df = pd.read\_stata('disarea.dta')

# Print the head of the DataFrame df

print(df.head())

# Plot histogram of one column of the DataFrame

pd.DataFrame.hist(df[['disa10']])

plt.xlabel('Extent of disease')

plt.ylabel('Number of countries')

plt.show()

wbcode country disa1 disa2 disa3 ... disa21 disa22 disa23 disa24 disa25

0 AFG Afghanistan 0.00 0.00 0.76 ... 0.0 0.00 0.02 0.00 0.00

1 AGO Angola 0.32 0.02 0.56 ... 0.0 0.99 0.98 0.61 0.00

2 ALB Albania 0.00 0.00 0.02 ... 0.0 0.00 0.00 0.00 0.16

3 ARE United Arab Emirates 0.00 0.00 0.00 ... 0.0 0.00 0.00 0.00 0.00

4 ARG Argentina 0.00 0.24 0.24 ... 0.0 0.00 0.01 0.00 0.11

[5 rows x 27 columns]

**Daily XP800**

# Importing HDF5 files

**50 XP**

## 1. Importing HDF5 files

According to the 2013 O'Reilly book Python and HDF5 by Andrew Collette,

## 2. HDF5 files

"In the Python world, consensus is rapidly converging on Hierarchical Data Format version 5, or 'HDF5,' as the standard mechanism for storing large quantities of numerical data." How large are we talking here? According to Collette, "It’s now relatively common to deal with datasets hundreds of gigabytes or even terabytes in size; HDF5 itself can scale up to exabytes." Let's explore with a concrete example from LIGO, the Laser Interferometer Gravitational-Wave Observatory project.

## 3. Importing HDF5 files

You first import the package h5py and then import the file using 'h5py dot File', remembering to use 'r' in order to specify read only. Printing the datatype to the shell reveals that we are dealing with an h5py file.

## 4. The structure of HDF5 files

But what is the structure of this file? You can explore it's hierarchical structure as you would that of a Python dictionary using the method keys. You see that there are three keys, meta, quality and strain. Each of these is an HDF group. You can think of these groups as directories. The LIGO documentation tells us that 'meta' contains meta-data for the file, 'quality' contains information about data quality and 'strain' contains 'strain data from the interferometer', the main measurement performed by LIGO, the data of interest. If you knew what data and metadata should be in each group, you could access it straightforwardly. However, if not, due to the hierarchical nature of the file structure, it is easy to explore. For example,

## 5. The structure of HDF5 files

let's say you wanted to find out what type of metadata there is, you could easily print out the keys. Now you know the keys, you can access any metadata of interest. If you're interested in 'Description' and 'Detector', you can pass these keys to the numpy-dot-array function to convert the values to a NumPy array. You see that the data in the file is 'Strain data time series from LIGO' and that the detector used was 'H1'. Next perhaps you would like to check out the actual data? Great idea and that's precisely what you're going to do in the upcoming exercises!

## 6. The HDF Project

Before you do so, it is also worth noting that the HDF project is actively maintained by the HDF group, based in Champaign, Illinois and formerly part of the University of Illinois Urbana-Champaign.

## 7. Let's practice!

Now it's time for you to import and visualize some of the data that led to the validation of Einstein's Theory of Gravitational Waves, enjoy!

**Daily XP850**

##### Exercise

# Using File to import HDF5 files

The h5py package has been imported in the environment and the file LIGO\_data.hdf5 is loaded in the object h5py\_file.

What is the correct way of using the h5py function, File(), to import the file in h5py\_file into an object, h5py\_data, for reading only?

##### Instructions

**50 XP**

##### Possible Answers



h5py\_data = File(h5py\_file, 'r')



**h5py\_data = h5py.File(h5py\_file, 'r') This the correct answer**



h5py\_data = h5py.File(h5py\_file, read)



h5py\_data = h5py.File(h5py\_file, 'read')

# Import pandas import pandas as pd # Load Stata file into a pandas DataFrame: df df = pd.read\_stata('disarea.dta') # Print the head of the DataFrame df print(df.head()) # Plot histogram of one column of the DataFrame pd.DataFrame.hist(df[['disa10']]) plt.xlabel('Extent of disease') plt.ylabel('Number of countries') plt.show()

##### Exercise

# Using h5py to import HDF5 files

The file 'LIGO\_data.hdf5' is already in your working directory. In this exercise, you'll import it using the h5py library. You'll also print out its datatype to confirm you have imported it correctly. You'll then study the structure of the file in order to see precisely what HDF groups it contains.

You can find the LIGO data plus loads of documentation and tutorials [here](https://losc.ligo.org/events/GW150914/). There is also a great tutorial on Signal Processing with the data [here](https://www.gw-openscience.org/GW150914data/LOSC_Event_tutorial_GW150914.html).

##### Instructions

**100 XP**

* Import the package h5py.
* Assign the name of the file to the variable file.
* Load the file as read only into the variable data.
* Print the datatype of data.
* Print the names of the groups in the HDF5 file 'LIGO\_data.hdf5'.
* # Import packages
* import numpy as np
* import \_\_\_\_
* # Assign filename: file
* # Load file: data
* data = h5py.File(\_\_\_\_, \_\_\_\_)
* # Print the datatype of the loaded file
* # Print the keys of the file
* for key in \_\_\_\_:
* print(\_\_\_\_)

# Import packages

import numpy as np

import h5py

# Assign filename: file

file = 'LIGO\_data.hdf5'

# Load file: data

data = h5py.File(file, 'r')

# Print the datatype of the loaded file

print(type(data))

# Print the keys of the file

for key in data.keys():

    print(key)

# Import packages

import numpy as np

import h5py

# Assign filename: file

file = 'LIGO\_data.hdf5'

# Load file: data

data = h5py.File(file, 'r')

# Print the datatype of the loaded file

print(type(data))

# Print the keys of the file

for key in data.keys():

print(key)

<class 'h5py.\_hl.files.File'>

meta

quality

strain

<script.py> output:

<class 'h5py.\_hl.files.File'>

meta

quality

strain

# Import packages import numpy as np import h5py # Assign filename: file file = 'LIGO\_data.hdf5' # Load file: data data = h5py.File(file, 'r') # Print the datatype of the loaded file print(type(data)) # Print the keys of the file for key in data.keys(): print(key)

**Daily XP1000**

##### Exercise

# Extracting data from your HDF5 file

In this exercise, you'll extract some of the LIGO experiment's actual data from the HDF5 file and you'll visualize it.

To do so, you'll need to first explore the HDF5 group 'strain'.

##### Instructions

**100 XP**

* Assign the HDF5 group data['strain'] to group.
* In the for loop, print out the keys of the HDF5 group in group.
* Assign the time series data data['strain']['Strain'] to a NumPy array called strain.
* Set num\_samples equal to 10000, the number of time points we wish to sample.
* Execute the rest of the code to produce a plot of the time series data in LIGO\_data.hdf5.
* # Get the HDF5 group: group
* # Check out keys of group
* for key in \_\_\_\_:
* print(\_\_\_\_)
* # Set variable equal to time series data: strain
* # Set number of time points to sample: num\_samples
* # Set time vector
* time = np.arange(0, 1, 1/num\_samples)
* # Plot data
* plt.plot(time, strain[:num\_samples])
* plt.xlabel('GPS Time (s)')
* plt.ylabel('strain')
* plt.show()

# Get the HDF5 group: group

group = data['strain']

# Check out keys of group

for key in group.keys():

    print(key)

# Set variable equal to time series data: strain

strain = np.array(data['strain']['Strain'])

# Set number of time points to sample: num\_samples

num\_samples = 10000

# Set time vector

time = np.arange(0, 1, 1/num\_samples)

# Plot data

plt.plot(time, strain[:num\_samples])

plt.xlabel('GPS Time (s)')

plt.ylabel('strain')

plt.show()

# Get the HDF5 group: group

group = data['strain']

# Check out keys of group

for key in group.keys():

print(key)

# Set variable equal to time series data: strain

strain = np.array(data['strain']['Strain'])

# Set number of time points to sample: num\_samples

num\_samples = 10000

# Set time vector

time = np.arange(0, 1, 1/num\_samples)

# Plot data

plt.plot(time, strain[:num\_samples])

plt.xlabel('GPS Time (s)')

plt.ylabel('strain')

plt.show()

Strain

ERROR! Session/line number was not unique in database. History logging moved to new session 11

<script.py> output:

Strain

MATLAB, which is short

## 2. MATLAB

for Matrix Laboratory, is a numerical computing environment that is an industry standard in the disciplines of engineering and science. This is due in part to its powerful linear algebra and matrix capabilities, in part to its proprietary nature and in part to how difficult the academic world finds it to shake off old habits. Regardless of the reasons for its widespread use, the fact of the matter is that a lot of people use MATLAB and save their data as 'dot mat' files, the file format native to MATLAB. How can you import these into Python?

## 3. SciPy to the rescue!

Luckily for us Python afficionados, the standard library scipy has functions loadmat and savemat, which allow us to read and write dot mat files, respectively.

## 4. What is a .mat file?

"What exactly is in a dot mat file?" you may ask. To answer this, lets look at the MATLAB IDE. In particular,

## 5. What is a .mat file?

check out the MATLAB workspace where all your variables are stored. This workspace can contain strings, floats, vectors and arrays, among many other objects. A dot mat file is simply a collection of such objects.

## 6. Importing a .mat file

Now this means when importing a dot mat file in Python, we should expect to see a number of different variables and objects. In this code, I first import scipy-dot-io and then load the file 'workspace dot mat'. Checking out what type of object results tells me that it's a dictionary. How this dictionary relates to a MATLAB workspace is straightforward: the keys of the Python dictionary are the MATLAB variable names and the values of the Python dictionary are the objects that are assigned to the variables. In the example above, mat['x'] is a numpy corresponding to the MATLAB array x in your MATLAB workspace. It's that easy.

## 7. Let's practice!

Now it's your turn to import a MATLAB workspace and check out what it contains, happy exploring!

**Daily XP1150**

##### Exercise

##### Exercise

# Loading .mat files

In this exercise, you'll figure out how to load a MATLAB file using scipy.io.loadmat() and you'll discover what Python datatype it yields.

The file 'albeck\_gene\_expression.mat' is in your working directory. This file contains [gene expression data](https://www.mcb.ucdavis.edu/faculty-labs/albeck/workshop.htm) from the Albeck Lab at UC Davis. You can find the data and some great documentation [here](https://www.mcb.ucdavis.edu/faculty-labs/albeck/workshop.htm).

##### Instructions

**100 XP**

* Import the package scipy.io.
* Load the file 'albeck\_gene\_expression.mat' into the variable mat; do so using the function scipy.io.loadmat().
* Use the function type() to print the datatype of mat to the IPython shell.
* Import package
* # Load MATLAB file: mat
* # Print the datatype type of mat
* print(\_\_\_\_)

# Get the HDF5 group: group group = data['strain'] # Check out keys of group for key in group.keys(): print(key) # Set variable equal to time series data: strain strain = np.array(data['strain']['Strain']) # Set number of time points to sample: num\_samples num\_samples = 10000 # Set time vector time = np.arange(0, 1, 1/num\_samples) # Plot data plt.plot(time, strain[:num\_samples]) plt.xlabel('GPS Time (s)') plt.ylabel('strain') plt.show()

# Import package

import scipy.io

# Load MATLAB file: mat

mat = scipy.io.loadmat('albeck\_gene\_expression.mat')

# Print the datatype type of mat

print(type(mat))

# Import package

import scipy.io

# Load MATLAB file: mat

mat = scipy.io.loadmat('albeck\_gene\_expression.mat')

# Print the datatype type of mat

print(type(mat))

<class 'dict'>

**Daily XP1250**

##### Exercise

# The structure of .mat in Python

Here, you'll discover what is in the MATLAB dictionary that you loaded in the previous exercise.

The file 'albeck\_gene\_expression.mat' is already loaded into the variable mat. The following libraries have already been imported as follows:

import scipy.io

import matplotlib.pyplot as plt

import numpy as np

Once again, this file contains [gene expression data](https://www.mcb.ucdavis.edu/faculty-labs/albeck/workshop.htm) from the Albeck Lab at UCDavis. You can find the data and some great documentation [here](https://www.mcb.ucdavis.edu/faculty-labs/albeck/workshop.htm).

##### Instructions

**100 XP**

* Use the method .keys() on the dictionary mat to print the keys. Most of these keys (in fact the ones that do NOT begin and end with '\_\_') are variables from the corresponding MATLAB environment.
* Print the type of the value corresponding to the key 'CYratioCyt' in mat. Recall that mat['CYratioCyt'] accesses the value.
* Print the shape of the value corresponding to the key 'CYratioCyt' using the numpy function shape().
* Execute the entire script to see some oscillatory gene expression data!
* # Print the keys of the MATLAB dictionary
* print(\_\_\_\_)
* # Print the type of the value corresponding to the key 'CYratioCyt'
* # Print the shape of the value corresponding to the key 'CYratioCyt'
* # Subset the array and plot it
* data = mat['CYratioCyt'][25, 5:]
* fig = plt.figure()
* plt.plot(data)
* plt.xlabel('time (min.)')
* plt.ylabel('normalized fluorescence (measure of expression)')
* plt.show()

# Import package import scipy.io # Load MATLAB file: mat mat = scipy.io.loadmat('albeck\_gene\_expression.mat') # Print the datatype type of mat print(type(mat))

# Print the keys of the MATLAB dictionary

print(mat.keys())

# Print the type of the value corresponding to the key 'CYratioCyt'

print(type(mat['CYratioCyt']))

# Print the shape of the value corresponding to the key 'CYratioCyt'

print(np.shape(mat['CYratioCyt']))

# Subset the array and plot it

data = mat['CYratioCyt'][25, 5:]

fig = plt.figure()

plt.plot(data)

plt.xlabel('time (min.)')

plt.ylabel('normalized fluorescence (measure of expression)')

plt.show()

# Print the keys of the MATLAB dictionary

print(mat.keys())

# Print the type of the value corresponding to the key 'CYratioCyt'

print(type(mat['CYratioCyt']))

# Print the shape of the value corresponding to the key 'CYratioCyt'

print(np.shape(mat['CYratioCyt']))

# Subset the array and plot it

data = mat['CYratioCyt'][25, 5:]

fig = plt.figure()

plt.plot(data)

plt.xlabel('time (min.)')

plt.ylabel('normalized fluorescence (measure of expression)')

plt.show()

dict\_keys(['\_\_header\_\_', '\_\_version\_\_', '\_\_globals\_\_', 'rfpCyt', 'rfpNuc', 'cfpNuc', 'cfpCyt', 'yfpNuc', 'yfpCyt', 'CYratioCyt'])

<class 'numpy.ndarray'>

(200, 137)

# Introduction to relational databases

**50 XP**

## 1. Introduction to relational databases

You have already mastered the art of importing all types of single files in Python: congratulations! However, to earn your daily bread and butter as a Data Scientist, you'll be required to interact with more complex data structures, such as relational databases.

## 2. What is a relational database?

What is a relational database? It's a type of database that is based upon the Relational model of data, first described by Ted Codd in the late 1960s. Before getting too theoretical, however, let's check out at an illuminating example,

## 3. Example: Northwind database

the Northwind Traders database, a synthetic database that contains sales data for a fictitious company. Firstly, a database consists of tables. Here you can see 3 tables from the Northwind database:

## 4. Example: Northwind database

'Orders',

## 5. Example: Northwind database

'Customers' and

## 6. Example: Northwind database

'Employees'. So what's a table? A table generally represents one entity type,

## 7. The Orders table

such as 'Order' . Notice that this table looks a great deal like a dataframe. That's the point. In a relational database table,

## 8. The Orders table

each row or record represents an instance of the entity type: in this case, each row is an Order.

## 9. The Orders table

Each column represents an attribute of each instance, such as 'OrderDate' in the case of 'Orders'. In this sense, a table is entirely analogous to a dataframe. It is essential that each row contain a unique identifier, known as a primary key, that we can use to explicitly access the row in question. In our 'Orders' table, you can see that

## 10. The Orders table

the key is 'OrderID' the first column. But recall that a database consists of many tables! The really cool thing about relational databases is not merely that you have a bunch of tables,

## 11. Tables are linked

but that the tables are linked. How this linking works is ultra-intuitive: see that the 'Orders' table has

## 12. Tables are linked

both a column called 'CustomerID' and one called 'EmployeeID'. These columns correspond precisely to the primary keys in the

## 13. Tables are linked

'Customers' and 'Employees' tables, respectively. So, given an Order, you can immediately look up the details of the relevant Customer or Employee. This is cool because it means that you don't need to store all the Customer details, such as first name, last name, company with every order that they place: you merely need to look it up in the 'Customers' table. This saves an incredible amount of space!

## 14. Relational model

As stated earlier, the relational database model was originally proposed by "Ted" Codd and has been widely adopted. There is a great deal to theory but it is most neatly summarized in Codd's 12 Rules, also known as Codd's 12 Commandments, which he developed in the early 1980s to combat what he viewed as a dilution of his original relational database vision. Codd's 12 Rules actually consist of 13 rules but they are zero-indexed, that is, the first rule is zero-indexed. If that makes you laugh, you're definitely a geek like me! These 13 rules were defined to describe what a Relational Database Management System should adhere to in order to be considered relational.

## 15. Relational Database Management Systems

Among the most popular of such systems are PostreSQL (commonly called Postgres),

## 16. Relational Database Management Systems

MySQL and

## 17. Relational Database Management Systems

SQLite, all of which use the SQL query language. In fact, SQL itself is actually an acronym

## 18. Relational Database Management Systems

for Structured Query Language, which describes how you communicate with a database in order to both access and update the information it contains. The term "querying" is really just a fancy way of saying getting data out from the database. Next up, you'll learn how to connect to databases but before that,

## 19. Let's practice!

let's make sure that you have a solid conceptual grip on the relational model.

# Pop quiz: The relational model

Which of the following is not part of the relational model?

##### Answer the question

**50XP**

#### Possible Answers



Each row or record in a table represents an instance of an entity type.

press1



Each column in a table represents an attribute or feature of an instance.

press2



Every table contains a primary key column, which has a unique entry for each row.

press3



**A database consists of at least 3 tables. This is the answer this is False**

press4



There are relations between tables.

press5

**Daily XP1450**

# Creating a database engine in Python

**50 XP**

## 1. Creating a database engine in Python

All right, we're back! What we really want to do is to get data out of our databases using SQL, or the Structured Query Language. But before we get to that, we're going to need to figure out how to connect to a database.

## 2. Creating a database engine

We'll use an SQLite database as an example because SQLite is fast and simple while still containing enough functionality to introduce you to all the necessary concepts of querying a database. There are times when you would prefer to use PostgreSQL or MySQL, but for our purposes here, an Introduction to Interacting with Relational Databases in Python, a SQLite database is perfect. We'll once again look at the Northwind database. There are many packages we could use to access an SQLite database such as sqlite3 and SQLAlchemy. We'll use SQLAlchemy as it works with many other Relational Database Management Systems, such as Postgres and MySQL. So without further ado, to connect to 'Northwind dot sqlite', we need to import the relevant funtion create\_engine from the package SQLAlchemy. We then use the function create\_engine to fire up an SQL engine that will communicate our queries to the database. The only required argument of create\_engine is a string that indicates the type of database you're connecting to and the name of the database. Next, In order to query the database, we need to connect to the engine to do so.

## 3. Getting table names

But before we do this, we would like to know the names of the tables it contains. To do this, apply the method table\_names to the object engine. This will return a list of the table names that you can then print the console. Now it's your turn to do the same: fire up the database engine and print the table names!

## 4. Let's practice!

After this, I'll be back to show you how to connect to the engine, query your DBs and then you'll get loads of practice writing your own queries to import data from relational databases!

**Daily XP1500**

##### Exercise

##### Exercise

# Creating a database engine

Here, you're going to fire up your very first SQL engine. You'll create an engine to connect to the SQLite database 'Chinook.sqlite', which is in your working directory. Remember that to create an engine to connect to 'Northwind.sqlite', Hugo executed the command

engine = create\_engine('sqlite:///Northwind.sqlite')

Here, 'sqlite:///Northwind.sqlite' is called the connection string to the SQLite database Northwind.sqlite. A little bit of background on the [Chinook database](https://github.com/lerocha/chinook-database): the Chinook database contains information about a semi-fictional digital media store in which media data is real and customer, employee and sales data has been manually created.

Why the name Chinook, you ask? According to their [website](https://github.com/lerocha/chinook-database),

The name of this sample database was based on the Northwind database. Chinooks are winds in the interior West of North America, where the Canadian Prairies and Great Plains meet various mountain ranges. Chinooks are most prevalent over southern Alberta in Canada. Chinook is a good name choice for a database that intends to be an alternative to Northwind.

##### Instructions

**100 XP**

* Import the function create\_engine from the module sqlalchemy.
* Create an engine to connect to the SQLite database 'Chinook.sqlite' and assign it to engine.
* # Import necessary module
* from \_\_\_\_ import \_\_\_\_
* # Create engine: engine

# Import necessary module

from sqlalchemy import create\_engine

# Create engine: engine

engine = create\_engine('sqlite:///Chinook.sqlite')

# Import necessary module from sqlalchemy import create\_engine # Create engine: engine engine = create\_engine('sqlite:///Chinook.sqlite')

**Daily XP1600**

##### Exercise

##### Exercise

# What are the tables in the database?

In this exercise, you'll once again create an engine to connect to 'Chinook.sqlite'. Before you can get any data out of the database, however, you'll need to know what tables it contains!

To this end, you'll save the table names to a list using the method table\_names() on the engine and then you will print the list.

##### Instructions

**100 XP**

* Import the function create\_engine from the module sqlalchemy.
* Create an engine to connect to the SQLite database 'Chinook.sqlite' and assign it to engine.
* Using the method table\_names() on the engine engine, assign the table names of 'Chinook.sqlite' to the variable table\_names.
* Print the object table\_names to the shell.
* # Import necessary module
* # Create engine: engine
* # Save the table names to a list: table\_names
* # Print the table names to the shell
* print(\_\_\_\_)

# Import necessary module from sqlalchemy import create\_engine # Create engine: engine engine = create\_engine('sqlite:///Chinook.sqlite')

# Import necessary module

from sqlalchemy import create\_engine

# Create engine: engine

engine = create\_engine('sqlite:///Chinook.sqlite')

# Save the table names to a list: table\_names

table\_names = engine.table\_names()

# Print the table names to the shell

print(table\_names)

# Import necessary module

from sqlalchemy import create\_engine

# Create engine: engine

engine = create\_engine('sqlite:///Chinook.sqlite')

# Save the table names to a list: table\_names

table\_names = engine.table\_names()

# Print the table names to the shell

print(table\_names)

['Album', 'Artist', 'Customer', 'Employee', 'Genre', 'Invoice', 'InvoiceLine', 'MediaType', 'Playlist', 'PlaylistTrack', 'Track']

**Daily XP1700**

# Querying relational databases in Python

**50 XP**

This course is also available on the mobile app

## 1. Querying relational databases in Python

Now that you have figured out how to create a database engine and to list the tables of the database in question, it's time to connect to the engine and query the database. Once again, the term "querying" is just a fancy way of saying getting data out from the database.

## 2. Basic SQL query

THe HELLO WORLD of SQL queries is 'SELECT \* FROM Table\_Name', where 'Table\_name' is the name of any of the tables in the database. This query returns all columns of all rows of the Table of interest. For example, I could query the Northwind database with 'SELECT \* FROM Orders' and this would return all columns of all rows of the 'Orders' table. The star after SELECT means 'all columns'. Straightforward, right? Well, nearly! This is an SQL query and we need to figure out how to make such a query using python, SQLAlchemy and, in fact, we'll also use pandas to store the results of our queries.

## 3. Workflow of SQL querying

The workflow will be as follows. You'll import the required packages and functions, create the engine, connect to it, query the database save the results of the query to a dataframe, and close the connection.

## 4. Your first SQL query

Let's now check out how to do each of these steps! Create the engine using the function create\_engine. To connect to the database after creating the engine, you create a connection object con by applying the method connect to the engine. To query the DB, apply the method execute to the connection con and pass it a single argument, the relevant SQL query; This creates a SQLAlchemy results object which we assign to the variable rs. To turn the results object rs into a dataframe, we apply the method fetchall to rs and save it as a dataframe using the pandas function DataFrame. fetchall fetches all rows, as you would expect. To close the connection, execute 'con dot close'. Don't forget to do this!

## 5. Printing your query results

You can then print the head of the dataframe, as we have done before, as a sanity check: all the rows look good but the column names aren't correct.

## 6. Set the DataFrame column names

To fix this, before closing the connection, you can set the dataframe's column names by executing 'df dot columns equals rs dot keys'.

## 7. Set the data frame column names

One last note: analogous to what you saw in chapter 1 when opening plain text files, you can use

## 8. Using the context manager

the context manager construct to open a connection, which will save you the trouble of closing the connection later, or save you the trouble of forgetting to close it! There are two other differences that you may have notice between this and the previous code: firstly, I no longer have 'SELECT \*' in the SQL query; I now have column names of the table 'Orders'; all this does is it imports those particular columns and no others whereas 'SELECT \*' imports all columns; secondly, instead of applying the method fetchall to the results rs, I apply the method fetchmany with the argument size equals 5; this imports 5 rows instead of all rows. You'll become better acquainted with these functions and arguments very soon.

## 9. Let's practice!

That's enough out of me! It's time for you to practice writing your own SQL queries to import data from your database, enjoy!

**Daily XP1750**

##### Exercise

##### Exercise

# The Hello World of SQL Queries!

Now, it's time for liftoff! In this exercise, you'll perform the Hello World of SQL queries, SELECT, in order to retrieve all columns of the table Album in the Chinook database. Recall that the query SELECT \* selects all columns.

##### Instructions

**100 XP**

* Open the engine connection as con using the method connect() on the engine.
* Execute the query that **selects** ALL columns **from** the Album table. Store the results in rs.
* Store all of your query results in the DataFrame df by applying the fetchall() method to the results rs.
* Close the connection!

# Import necessary module from sqlalchemy import create\_engine # Create engine: engine engine = create\_engine('sqlite:///Chinook.sqlite') # Save the table names to a list: table\_names table\_names = engine.table\_names() # Print the table names to the shell print(table\_names)

# Import packages

from sqlalchemy import create\_engine

import pandas as pd

# Create engine: engine

engine = create\_engine('sqlite:///Chinook.sqlite')

# Open engine connection: con

# Perform query: rs

rs = con.execute(\_\_\_\_)

# Save results of the query to DataFrame: df

df = pd.DataFrame(\_\_\_\_)

# Close connection

# Print head of DataFrame df

print(df.head())

# Import packages

from sqlalchemy import create\_engine

import pandas as pd

# Create engine: engine

engine = create\_engine('sqlite:///Chinook.sqlite')

# Open engine connection: con

con = engine.connect()

# Perform query: rs

rs = con.execute("SELECT \* FROM Album")

# Save results of the query to DataFrame: df

df = pd.DataFrame(rs.fetchall())

# Close connection

con.close()

# Print head of DataFrame df

print(df.head())

# Import packages

from sqlalchemy import create\_engine

import pandas as pd

# Create engine: engine

engine = create\_engine('sqlite:///Chinook.sqlite')

# Open engine connection: con

con = engine.connect()

# Perform query: rs

rs = con.execute("SELECT \* FROM Album")

# Save results of the query to DataFrame: df

df = pd.DataFrame(rs.fetchall())

# Close connection

con.close()

# Print head of DataFrame df

print(df.head())

0 1 2

0 1 For Those About To Rock We Salute You 1

1 2 Balls to the Wall 2

2 3 Restless and Wild 2

3 4 Let There Be Rock 1

4 5 Big Ones 3

**Daily XP1850**

##### Exercise

##### Exercise

# Customizing the Hello World of SQL Queries

Congratulations on executing your first SQL query! Now you're going to figure out how to customize your query in order to:

* Select specified columns from a table;
* Select a specified number of rows;
* Import column names from the database table.

Recall that Hugo performed a very similar query customization in the video:

engine = create\_engine('sqlite:///Northwind.sqlite')

with engine.connect() as con:

rs = con.execute("SELECT OrderID, OrderDate, ShipName FROM Orders")

df = pd.DataFrame(rs.fetchmany(size=5))

df.columns = rs.keys()

Packages have already been imported as follows:

from sqlalchemy import create\_engine

import pandas as pd

The engine has also already been created:

engine = create\_engine('sqlite:///Chinook.sqlite')

The engine connection is already open with the statement

with engine.connect() as con:

All the code you need to complete is within this context.

##### Instructions

**100 XP**

* Execute the SQL query that **selects** the columns LastName and Title **from** the Employee table. Store the results in the variable rs.
* Apply the method fetchmany() to rs in order to retrieve 3 of the records. Store them in the DataFrame df.
* Using the rs object, set the DataFrame's column names to the corresponding names of the table columns.
* # Open engine in context manager
* # Perform query and save results to DataFrame: df
* with engine.connect() as con:
* rs = \_\_\_\_
* df = pd.DataFrame(\_\_\_\_)
* df.columns = \_\_\_\_
* # Print the length of the DataFrame df
* print(len(df))
* # Print the head of the DataFrame df
* print(df.head())

# Import packages from sqlalchemy import create\_engine import pandas as pd # Create engine: engine engine = create\_engine('sqlite:///Chinook.sqlite') # Open engine connection: con con = engine.connect() # Perform query: rs rs = con.execute("SELECT \* FROM Album") # Save results of the query to DataFrame: df df = pd.DataFrame(rs.fetchall()) # Close connection con.close() # Print head of DataFrame df print(df.head())

# Open engine in context manager

# Perform query and save results to DataFrame: df

with engine.connect() as con:

    rs = con.execute("SELECT LastName, Title FROM Employee")

    df = pd.DataFrame(rs.fetchmany(size=3))

    df.columns = rs.keys()

# Print the length of the DataFrame df

print(len(df))

# Print the head of the DataFrame df

print(df.head())

Open engine in context manager

# Perform query and save results to DataFrame: df

with engine.connect() as con:

rs = con.execute("SELECT LastName, Title FROM Employee")

df = pd.DataFrame(rs.fetchmany(size=3))

df.columns = rs.keys()

# Print the length of the DataFrame df

print(len(df))

# Print the head of the DataFrame df

print(df.head())

3

LastName Title

0 Adams General Manager

1 Edwards Sales Manager

2 Peacock Sales Support Agent

**Daily XP1950**

##### Exercise

##### Exercise

# Filtering your database records using SQL's WHERE

You can now execute a basic SQL query to select records from any table in your database and you can also perform simple query customizations to select particular columns and numbers of rows.

There are a couple more standard SQL query chops that will aid you in your journey to becoming an SQL ninja.

Let's say, for example that you wanted to get all records from the Customer table of the Chinook database for which the Country is 'Canada'. You can do this very easily in SQL using a SELECT statement followed by a WHERE clause as follows:

SELECT \* FROM Customer WHERE Country = 'Canada'

In fact, you can filter any SELECT statement by any condition using a WHERE clause. This is called filtering your records.

In this interactive exercise, you'll select all records of the Employee table for which 'EmployeeId' is greater than or equal to 6.

Packages are already imported as follows:

import pandas as pd

from sqlalchemy import create\_engine

Query away!

##### Instructions

**100 XP**

* Complete the argument of create\_engine() so that the engine for the SQLite database 'Chinook.sqlite' is created.
* Execute the query that **selects** all records **from** the Employee table **where** 'EmployeeId' is greater than or equal to 6. Use the >= operator and assign the results to rs.
* Apply the method fetchall() to rs in order to fetch all records in rs. Store them in the DataFrame df.
* Using the rs object, set the DataFrame's column names to the corresponding names of the table columns.
* # Create engine: engine
* engine = create\_engine(\_\_\_\_)
* # Open engine in context manager
* # Perform query and save results to DataFrame: df
* with engine.connect() as con:
* rs = con.execute(\_\_\_\_)
* df = pd.DataFrame(\_\_\_\_)
* df.columns = \_\_\_\_
* # Print the head of the DataFrame df
* print(df.head())

# Open engine in context manager # Perform query and save results to DataFrame: df with engine.connect() as con: rs = con.execute("SELECT LastName, Title FROM Employee") df = pd.DataFrame(rs.fetchmany(size=3)) df.columns = rs.keys() # Print the length of the DataFrame df print(len(df)) # Print the head of the DataFrame df print(df.head())

# Create engine: engine

engine = create\_engine('sqlite:///Chinook.sqlite')

# Open engine in context manager

# Perform query and save results to DataFrame: df

with engine.connect() as con:

    rs = con.execute("SELECT \* FROM Employee WHERE EmployeeId >= 6")

    df = pd.DataFrame(rs.fetchall())

    df.columns = rs.keys()

# Print the head of the DataFrame df

print(df.head())

# Create engine: engine

engine = create\_engine('sqlite:///Chinook.sqlite')

# Open engine in context manager

# Perform query and save results to DataFrame: df

with engine.connect() as con:

rs = con.execute("SELECT \* FROM Employee WHERE EmployeeId >= 6")

df = pd.DataFrame(rs.fetchall())

df.columns = rs.keys()

# Print the head of the DataFrame df

print(df.head())

EmployeeId LastName FirstName Title ReportsTo ... Country PostalCode Phone Fax Email

0 6 Mitchell Michael IT Manager 1 ... Canada T3B 0C5 +1 (403) 246-9887 +1 (403) 246-9899 michael@chinookcorp.com

1 7 King Robert IT Staff 6 ... Canada T1K 5N8 +1 (403) 456-9986 +1 (403) 456-8485 robert@chinookcorp.com

2 8 Callahan Laura IT Staff 6 ... Canada T1H 1Y8 +1 (403) 467-3351 +1 (403) 467-8772 laura@chinookcorp.com

[3 rows x 15 columns]

**Daily XP2050**

##### Exercise

##### Exercise

# Ordering your SQL records with ORDER BY

You can also order your SQL query results. For example, if you wanted to get all records from the Customer table of the Chinook database and order them in increasing order by the column SupportRepId, you could do so with the following query:

"SELECT \* FROM Customer ORDER BY SupportRepId"

In fact, you can order any SELECT statement by any column.

In this interactive exercise, you'll select all records of the Employee table and order them in increasing order by the column BirthDate.

Packages are already imported as follows:

import pandas as pd

from sqlalchemy import create\_engine

Get querying!

##### Instructions

**100 XP**

* Using the function create\_engine(), create an engine for the SQLite database Chinook.sqlite and assign it to the variable engine.
* In the context manager, execute the query that **selects** all records **from** the Employee table and **orders** them in increasing order **by** the column BirthDate. Assign the result to rs.
* In a call to pd.DataFrame(), apply the method fetchall() to rs in order to fetch all records in rs. Store them in the DataFrame df.
* Set the DataFrame's column names to the corresponding names of the table columns.

# Create engine: engine engine = create\_engine('sqlite:///Chinook.sqlite') # Open engine in context manager # Perform query and save results to DataFrame: df with engine.connect() as con: rs = con.execute("SELECT \* FROM Employee WHERE EmployeeId >= 6") df = pd.DataFrame(rs.fetchall()) df.columns = rs.keys() # Print the head of the DataFrame df print(df.head())

# Create engine: engine

# Open engine in context manager

with engine.connect() as con:

    rs = \_\_\_\_

    df = \_\_\_\_

    # Set the DataFrame's column names

# Print head of DataFrame

print(df.head())

# Create engine: engine

engine = create\_engine('sqlite:///Chinook.sqlite')

# Open engine in context manager

with engine.connect() as con:

    rs = con.execute("SELECT \* FROM Employee ORDER BY BirthDate")

    df = pd.DataFrame(rs.fetchall())

 # Set the DataFrame's column names

df.columns = rs.keys()

# Print head of DataFrame

print(df.head())

# Create engine: engine

engine = create\_engine('sqlite:///Chinook.sqlite')

# Open engine in context manager

with engine.connect() as con:

rs = con.execute("SELECT \* FROM Employee ORDER BY BirthDate")

df = pd.DataFrame(rs.fetchall())

# Set the DataFrame's column names

df.columns = rs.keys()

# Print head of DataFrame

print(df.head())

EmployeeId LastName FirstName Title ReportsTo ... Country PostalCode Phone Fax Email

0 4 Park Margaret Sales Support Agent 2.0 ... Canada T2P 5G3 +1 (403) 263-4423 +1 (403) 263-4289 margaret@chinookcorp.com

1 2 Edwards Nancy Sales Manager 1.0 ... Canada T2P 2T3 +1 (403) 262-3443 +1 (403) 262-3322 nancy@chinookcorp.com

2 1 Adams Andrew General Manager NaN ... Canada T5K 2N1 +1 (780) 428-9482 +1 (780) 428-3457 andrew@chinookcorp.com

3 5 Johnson Steve Sales Support Agent 2.0 ... Canada T3B 1Y7 1 (780) 836-9987 1 (780) 836-9543 steve@chinookcorp.com

4 8 Callahan Laura IT Staff 6.0 ... Canada T1H 1Y8 +1 (403) 467-3351 +1 (403) 467-8772 laura@chinookcorp.com

[5 rows x 15 columns]

# Querying relational databases directly with pandas

**50 XP**

## 1. Querying relational databases directly with pandas

You have seen that,

## 2. The pandas way to query

after creating a database engine, you can get the results of any particular line using 4 lines of code: connecting, executing a query, passing the results to a dataframe and naming the columns: 4 lines of code is pretty good but you can do better! You can actually do it in 1 line, utilizing the pandas function read\_sql\_query and passing it 2 arguments. The first argument will be the query you wish to make, the 2nd argument the engine you want to connect to. And thus you can achieve the same as this code by executing this single line. The power of pandas! In the following exercises, you'll gain more expertise in writing SQL queries and using pandas to execute them.

## 3. Let's practice!

What are you waiting for? The end of this video? Let's do it!

# Import packages

from sqlalchemy import create\_engine

import pandas as pd

# Create engine: engine

engine = create\_engine('sqlite:///Chinook.sqlite')

# Execute query and store records in DataFrame: df

df = pd.read\_sql\_query("SELECT \* FROM Album", engine)

# Print head of DataFrame

print(df.head())

# Open engine in context manager and store query result in df1

with engine.connect() as con:

    rs = con.execute("SELECT \* FROM Album")

    df1 = pd.DataFrame(rs.fetchall())

    df1.columns = rs.keys()

# Confirm that both methods yield the same result

print(df.equals(df1))

# Import packages

from sqlalchemy import create\_engine

import pandas as pd

# Create engine: engine

engine = create\_engine('sqlite:///Chinook.sqlite')

# Execute query and store records in DataFrame: df

df = pd.read\_sql\_query("SELECT \* FROM Album", engine)

# Print head of DataFrame

print(df.head())

# Open engine in context manager and store query result in df1

with engine.connect() as con:

rs = con.execute("SELECT \* FROM Album")

df1 = pd.DataFrame(rs.fetchall())

df1.columns = rs.keys()

# Confirm that both methods yield the same result

print(df.equals(df1))

AlbumId Title ArtistId

0 1 For Those About To Rock We Salute You 1

1 2 Balls to the Wall 2

2 3 Restless and Wild 2

3 4 Let There Be Rock 1

4 5 Big Ones 3

True

**Daily XP2300**

##### Exercise

##### Exercise

# Pandas for more complex querying

Here, you'll become more familiar with the pandas function read\_sql\_query() by using it to execute a more complex query: a SELECT statement followed by both a WHERE clause AND an ORDER BY clause.

You'll build a DataFrame that contains the rows of the Employee table for which the EmployeeId is greater than or equal to 6 and you'll order these entries by BirthDate.

##### Instructions

**100 XP**

* Using the function create\_engine(), create an engine for the SQLite database Chinook.sqlite and assign it to the variable engine.
* Use the pandas function read\_sql\_query() to assign to the variable df the DataFrame of results from the following query: **select** all records **from** the Employee table **where** the EmployeeId is greater than or equal to 6 and **ordered by** BirthDate (make sure to use WHERE and ORDER BY in this precise order).
* # Import packages
* from sqlalchemy import create\_engine
* import pandas as pd
* # Create engine: engine
* # Execute query and store records in DataFrame: df
* # Print head of DataFrame
* print(df.head())

# Create engine: engine engine = create\_engine('sqlite:///Chinook.sqlite') # Execute query and store records in DataFrame: df df = pd.read\_sql\_query("SELECT \* FROM Album", engine) # Print head of DataFrame print(df.head()) # Open engine in context manager and store query result in df1 with engine.connect() as con: rs = con.execute("SELECT \* FROM Album") df1 = pd.DataFrame(rs.fetchall()) df1.columns = rs.keys() # Confirm that both methods yield the same result print(df.equals(df1))

# Import packages

from sqlalchemy import create\_engine

import pandas as pd

# Create engine: engine

engine = create\_engine('sqlite:///Chinook.sqlite')

# Execute query and store records in DataFrame: df

df = pd.read\_sql\_query("SELECT \* FROM Employee WHERE EmployeeId >=6 ORDER BY BirthDate", engine)

# Print head of DataFrame

print(df.head())

# Import packages

from sqlalchemy import create\_engine

import pandas as pd

# Create engine: engine

engine = create\_engine('sqlite:///Chinook.sqlite')

# Execute query and store records in DataFrame: df

df = pd.read\_sql\_query("SELECT \* FROM Employee WHERE EmployeeId >=6 ORDER BY BirthDate", engine)

# Print head of DataFrame

print(df.head())

EmployeeId LastName FirstName Title ReportsTo ... Country PostalCode Phone Fax Email

0 8 Callahan Laura IT Staff 6 ... Canada T1H 1Y8 +1 (403) 467-3351 +1 (403) 467-8772 laura@chinookcorp.com

1 7 King Robert IT Staff 6 ... Canada T1K 5N8 +1 (403) 456-9986 +1 (403) 456-8485 robert@chinookcorp.com

2 6 Mitchell Michael IT Manager 1 ... Canada T3B 0C5 +1 (403) 246-9887 +1 (403) 246-9899 michael@chinookcorp.com

[3 rows x 15 columns]

**Daily XP100**

# Advanced querying: exploiting table relationships

**50 XP**

## 1. Advanced querying: exploiting table relationships

By now, you have become familiar with querying individual tables of databases, such as the 'Orders' table of the Northwind database. You will remember, however, that much of the power of relational databases stems from the fact that they can capture relationships between tables: the tables are linked!

## 2. Tables are linked

For example, as we saw earlier, the 'Orders' table of the the Northwind Traders database has both a column called

## 3. Tables are linked

'CustomerID' and one called 'EmployeeID', columns which correspond precisely to the

## 4. Tables are linked

primary keys in the 'Customers' and 'Employees' tables, respectively. This means that, given an Order, you can immediately look up the details of the relevant Customer or Employee in the appropriate table.

## 5. JOINing tables

Now what if you want to incorporate such information into your query? For example, if you want to query the 'Orders' table and include, for each Order, information about the corresponding Customer from the 'Customers' table? A specific illustrative example will go a long way here: let's say that we wanted, for each Order, to get the

## 6. JOINing tables

OrderID and the

## 7. JOINing tables

CompanyName of the Customer. The OrderID lives in the 'Orders' table while the CompanyName lives in the 'Customers' table. SQL has a really clever way of doing this: it's called a JOIN because what you're really doing is joining two tables together,

## 8. JOINing tables

in this case, the 'Orders' and

## 9. JOINing tables

'Customers' tables. Specifically

## 10. INNER JOIN in Python (pandas)

it's an INNER JOIN. There are other types of JOINs which we won't cover here. As it's the CustomerID columns of the 'Orders' and 'Customers' tables that correspond to each, you'll want to JOIN the tables ON these columns and that is precisely what I have done in this code. The notation of dot followed by Column name is merely selecting a column of a table. The table that we are selecting from is "Orders INNER JOIN Customers on Orders dot CustomerID equals Customers dot CustomerID" and I am selecting the OrderID column and the CompanyName column of this new table. I know that that's a bit to take in, but after you get your hands dirty performing some INNER JOINs, you'll be far more comfortable with this advanced querying technique.

## 11. Let's practice!

It's your turn to query!

**Daily XP150**

##### Exercise

##### Exercise

# The power of SQL lies in relationships between tables: INNER JOIN

Here, you'll perform your first INNER JOIN! You'll be working with your favourite SQLite database, Chinook.sqlite. For each record in the Album table, you'll extract the Title along with the Name of the Artist. The latter will come from the Artist table and so you will need to INNER JOIN these two tables on the ArtistID column of both.

Recall that to INNER JOIN the Orders and Customers tables from the Northwind database, Hugo executed the following SQL query:

"SELECT OrderID, CompanyName FROM Orders INNER JOIN Customers on Orders.CustomerID = Customers.CustomerID"

The following code has already been executed to import the necessary packages and to create the engine:

import pandas as pd

from sqlalchemy import create\_engine

engine = create\_engine('sqlite:///Chinook.sqlite')

##### Instructions

**100 XP**

* Assign to rs the results from the following query: **select** all the records, extracting the Title of the record and Name of the artist of each record **from** the Album table and the Artist table, respectively. To do so, INNER JOIN these two tables on the ArtistID column of both.
* In a call to pd.DataFrame(), apply the method fetchall() to rs in order to fetch all records in rs. Store them in the DataFrame df.
* Set the DataFrame's column names to the corresponding names of the table columns.

# Open engine in context manager

# Perform query and save results to DataFrame: df

with engine.connect() as con:

    \_\_\_\_

    \_\_\_\_

    \_\_\_\_

# Print head of DataFrame df

print(df.head())

# Import packages from sqlalchemy import create\_engine import pandas as pd # Create engine: engine engine = create\_engine('sqlite:///Chinook.sqlite') # Execute query and store records in DataFrame: df df = pd.read\_sql\_query("SELECT \* FROM Employee WHERE EmployeeId >=6 ORDER BY BirthDate", engine) # Print head of DataFrame print(df.head())

# Open engine in context manager

# Perform query and save results to DataFrame: df

with engine.connect() as con:

    rs = con.execute("SELECT Title, Name FROM Album INNER JOIN Artist on Album.ArtistID = Artist.ArtistID")

    df = pd.DataFrame(rs.fetchall())

    df.columns = rs.keys()

# Print head of DataFrame df

print(df.head())

rint(df.head())

# Open engine in context manager

# Perform query and save results to DataFrame: df

with engine.connect() as con:

rs = con.execute("SELECT Title, Name FROM Album INNER JOIN Artist on Album.ArtistID = Artist.ArtistID")

df = pd.DataFrame(rs.fetchall())

df.columns = rs.keys()

# Print head of DataFrame df

print(df.head())

Title Name

0 For Those About To Rock We Salute You AC/DC

1 Balls to the Wall Accept

2 Restless and Wild Accept

3 Let There Be Rock AC/DC

4 Big Ones Aerosmith

**Daily XP250**

##### Exercise

##### Exercise

# Filtering your INNER JOIN

Congrats on performing your first INNER JOIN! You're now going to finish this chapter with one final exercise in which you perform an INNER JOIN and filter the result using a WHERE clause.

Recall that to INNER JOIN the Orders and Customers tables from the Northwind database, Hugo executed the following SQL query:

"SELECT OrderID, CompanyName FROM Orders INNER JOIN Customers on Orders.CustomerID = Customers.CustomerID"

The following code has already been executed to import the necessary packages and to create the engine:

import pandas as pd

from sqlalchemy import create\_engine

engine = create\_engine('sqlite:///Chinook.sqlite')

##### Instructions

**100 XP**

* Use the pandas function read\_sql\_query() to assign to the variable df the DataFrame of results from the following query: **select** all records **from** PlaylistTrack INNER JOIN Track on PlaylistTrack.TrackId = Track.TrackId that satisfy the condition Milliseconds < 250000.
* # Execute query and store records in DataFrame: df
* # Print head of DataFrame
* print(df.head())

# Open engine in context manager # Perform query and save results to DataFrame: df with engine.connect() as con: rs = con.execute("SELECT Title, Name FROM Album INNER JOIN Artist on Album.ArtistID = Artist.ArtistID") df = pd.DataFrame(rs.fetchall()) df.columns = rs.keys() # Print head of DataFrame df print(df.head())

# Execute query and store records in DataFrame: df

df = pd.read\_sql\_query("SELECT \* FROM PlaylistTrack INNER JOIN Track on PlaylistTrack.TrackId = Track.TrackId WHERE Milliseconds < 250000", engine)

# Print head of DataFrame

print(df.head())

# Execute query and store records in DataFrame: df

df = pd.read\_sql\_query("SELECT \* FROM PlaylistTrack INNER JOIN Track on PlaylistTrack.TrackId = Track.TrackId WHERE Milliseconds < 250000", engine)

# Print head of DataFrame

print(df.head())

PlaylistId TrackId TrackId Name AlbumId ... GenreId Composer Milliseconds Bytes UnitPrice

0 1 3390 3390 One and the Same 271 ... 23 None 217732 3559040 0.99

1 1 3392 3392 Until We Fall 271 ... 23 None 230758 3766605 0.99

2 1 3393 3393 Original Fire 271 ... 23 None 218916 3577821 0.99

3 1 3394 3394 Broken City 271 ... 23 None 228366 3728955 0.99

4 1 3395 3395 Somedays 271 ... 23 None 213831 3497176 0.99

[5 rows x 11 columns]

This course is also available on the mobile app

## 1. Final Thoughts

Congratulations on doing so well in a

## 2. What you’ve learned:

crash course on Relational Databases. Whether you have seen them before or not, you're now able to create engines and connect to them in Python, perform simple SELECT queries, filter you results using WHERE, perform more complex queries such as JOIN and store all your querying results in pandas dataframes, among many other things. These skills will get you a long way in using Python to import data from relational databases. In fact, having made it through this course, you're now well-equipped to import all types of files in Python, from plain text to flat files, Excel to Matlab, among many others. Well done! But that doesn't mean there's not more to be learnt. What about importing data from the web, where so much of it is??

## 3. Next course:

In the sequel to this, you'll learn how to import web-data. It's a short course and will equip you to import data from the world wide web and pull data from Application Programming Interfaces, also known as APIs. You'll take a deep dive into the Twitter streaming API, which allows us to stream real-time tweets. I'm sure that you're itching to get your hands dirty with web data;

## 4. Let's practice!

so I'll see you in the next course.

# Intermediate Importing Data in Python

* 2 hours
* 7 Videos
* 28 Exercises
* 150,137 Participants
* 2,300 XP

### Course Description



As a data scientist, you will need to clean data, wrangle and munge it, visualize it, build predictive models and interpret these models. Before you can do so, however, you will need to know how to get data into Python. In the prequel to this course, you learned many ways to import data into Python: from flat files such as .txt and .csv; from files native to other software such as Excel spreadsheets, Stata, SAS, and MATLAB files; and from relational databases such as SQLite and PostgreSQL. In this course, you'll extend this knowledge base by learning to import data from the web and by pulling data from Application Programming Interfaces— APIs—such as the Twitter streaming API, which allows us to stream real-time tweets.

Read More

1. 1

#### Importing data from the Internet

0%

The web is a rich source of data from which you can extract various types of insights and findings. In this chapter, you will learn how to get data from the web, whether it is stored in files or in HTML. You'll also learn the basics of scraping and parsing web data.

##### Importing flat files from the web

50 xp

##### Importing flat files from the web: your turn!

100 xp

##### Opening and reading flat files from the web

100 xp

##### Importing non-flat files from the web

100 xp

##### HTTP requests to import files from the web

50 xp

##### Performing HTTP requests in Python using urllib

100 xp

##### Printing HTTP request results in Python using urllib

100 xp

##### Performing HTTP requests in Python using requests

100 xp

##### Scraping the web in Python

50 xp

##### Parsing HTML with BeautifulSoup

100 xp

##### Turning a webpage into data using BeautifulSoup: getting the text

100 xp

##### Turning a webpage into data using BeautifulSoup: getting the hyperlinks

100 xp

[Hide Chapter Details](https://www.datacamp.com/courses/importing-data-in-python-part-2?embedded=true)

  2

#### Interacting with APIs to import data from the web

0%

In this chapter, you will gain a deeper understanding of how to import data from the web. You will learn the basics of extracting data from APIs, gain insight on the importance of APIs, and practice extracting data by diving into the OMDB and Library of Congress APIs.

##### Introduction to APIs and JSONs

50 xp

##### Pop quiz: What exactly is a JSON?

50 xp

##### Loading and exploring a JSON

100 xp

##### Pop quiz: Exploring your JSON

50 xp

##### APIs and interacting with the world wide web

50 xp

##### Pop quiz: What's an API?

50 xp

##### API requests

100 xp

##### JSON–from the web to Python

100 xp

##### Checking out the Wikipedia API

100 xp

[Hide Chapter Details](https://www.datacamp.com/courses/importing-data-in-python-part-2?embedded=true)

  3

#### Diving deep into the Twitter API

0%

In this chapter, you will consolidate your knowledge of interacting with APIs in a deep dive into the Twitter streaming API. You'll learn how to stream real-time Twitter data, and how to analyze and visualize it.

##### The Twitter API and Authentication

50 xp

##### Streaming tweets

100 xp

##### Load and explore your Twitter data

100 xp

##### Twitter data to DataFrame

100 xp

##### A little bit of Twitter text analysis

100 xp

##### Plotting your Twitter data

100 xp

##### Final Thoughts

50 xp

[Hide Chapter Details](https://www.datacamp.com/courses/importing-data-in-python-part-2?embedded=true)

**Daily XP450**

##### Exercise

##### Exercise

# Importing flat files from the web: your turn!

You are about to import your first file from the web! The flat file you will import will be 'winequality-red.csv' from the University of California, Irvine's [Machine Learning repository](https://archive.ics.uci.edu/ml/index.php). The flat file contains tabular data of physiochemical properties of red wine, such as pH, alcohol content and citric acid content, along with wine quality rating.

The URL of the file is

'https://assets.datacamp.com/production/course\_1606/datasets/winequality-red.csv'

After you import it, you'll check your working directory to confirm that it is there and then you'll load it into a pandas DataFrame.

##### Instructions

**100 XP**

* Import the function urlretrieve from the subpackage urllib.request.
* Assign the URL of the file to the variable url.
* Use the function urlretrieve() to save the file locally as 'winequality-red.csv'.
* Execute the remaining code to load 'winequality-red.csv' in a pandas DataFrame and to print its head to the shell.

# Import package

from \_\_\_\_ import \_\_\_\_

# Import pandas

import pandas as pd

# Assign url of file: url

# Save file locally

# Read file into a DataFrame and print its head

df = pd.read\_csv('winequality-red.csv', sep=';')

print(df.head())

# Import package

from urllib.request import urlretrieve

# Import pandas

import pandas as pd

# Assign url of file: url

url = 'https://assets.datacamp.com/production/course\_1606/datasets/winequality-red.csv'

# Save file locally

urlretrieve(url, 'winequality-red.csv')

# Read file into a DataFrame and print its head

df = pd.read\_csv('winequality-red.csv', sep=';')

print(df.head())

# Import package

from urllib.request import urlretrieve

# Import pandas

import pandas as pd

# Assign url of file: url

url = 'https://assets.datacamp.com/production/course\_1606/datasets/winequality-red.csv'

# Save file locally

urlretrieve(url, 'winequality-red.csv')

# Read file into a DataFrame and print its head

df = pd.read\_csv('winequality-red.csv', sep=';')

print(df.head())

fixed acidity volatile acidity citric acid residual sugar chlorides ... density pH sulphates alcohol quality

0 7.4 0.70 0.00 1.9 0.076 ... 0.998 3.51 0.56 9.4 5

1 7.8 0.88 0.00 2.6 0.098 ... 0.997 3.20 0.68 9.8 5

2 7.8 0.76 0.04 2.3 0.092 ... 0.997 3.26 0.65 9.8 5

3 11.2 0.28 0.56 1.9 0.075 ... 0.998 3.16 0.58 9.8 6

4 7.4 0.70 0.00 1.9 0.076 ... 0.998 3.51 0.56 9.4 5

[5 rows x 12 columns]

**Daily XP550**

##### Exercise

##### Exercise

# Opening and reading flat files from the web

You have just imported a file from the web, saved it locally and loaded it into a DataFrame. If you just wanted to load a file from the web into a DataFrame without first saving it locally, you can do that easily using pandas. In particular, you can use the function pd.read\_csv() with the URL as the first argument and the separator sep as the second argument.

The URL of the file, once again, is

'https://assets.datacamp.com/production/course\_1606/datasets/winequality-red.csv'

##### Instructions

**100 XP**

* Assign the URL of the file to the variable url.
* Read file into a DataFrame df using pd.read\_csv(), recalling that the separator in the file is ';'.
* Print the head of the DataFrame df.
* Execute the rest of the code to plot histogram of the first feature in the DataFrame df.
* # Import packages
* import matplotlib.pyplot as plt
* import pandas as pd
* # Assign url of file: url
* # Read file into a DataFrame: df
* # Print the head of the DataFrame
* print(\_\_\_\_)
* # Plot first column of df
* df.iloc[:, 0].hist()
* plt.xlabel('fixed acidity (g(tartaric acid)/dm$^3$)')
* plt.ylabel('count')
* plt.show()

# Import package from urllib.request import urlretrieve # Import pandas import pandas as pd # Assign url of file: url url = 'https://assets.datacamp.com/production/course\_1606/datasets/winequality-red.csv' # Save file locally urlretrieve(url, 'winequality-red.csv') # Read file into a DataFrame and print its head df = pd.read\_csv('winequality-red.csv', sep=';') print(df.head())

# Import packages

import matplotlib.pyplot as plt

import pandas as pd

# Assign url of file: url

url = 'https://assets.datacamp.com/production/course\_1606/datasets/winequality-red.csv'

# Read file into a DataFrame: df

df = pd.read\_csv(url, sep= ';')

# Print the head of the DataFrame

print(df.head())

# Plot first column of df

df.iloc[:, 0].hist()

plt.xlabel('fixed acidity (g(tartaric acid)/dm$^3$)')

plt.ylabel('count')

plt.show()

# Import packages

import matplotlib.pyplot as plt

import pandas as pd

# Assign url of file: url

url = 'https://assets.datacamp.com/production/course\_1606/datasets/winequality-red.csv'

# Read file into a DataFrame: df

df = pd.read\_csv(url, sep= ';')

# Print the head of the DataFrame

print(df.head())

# Plot first column of df

df.iloc[:, 0].hist()

plt.xlabel('fixed acidity (g(tartaric acid)/dm$^3$)')

plt.ylabel('count')

plt.show()

fixed acidity volatile acidity citric acid residual sugar chlorides ... density pH sulphates alcohol quality

0 7.4 0.70 0.00 1.9 0.076 ... 0.998 3.51 0.56 9.4 5

1 7.8 0.88 0.00 2.6 0.098 ... 0.997 3.20 0.68 9.8 5

2 7.8 0.76 0.04 2.3 0.092 ... 0.997 3.26 0.65 9.8 5

3 11.2 0.28 0.56 1.9 0.075 ... 0.998 3.16 0.58 9.8 6

4 7.4 0.70 0.00 1.9 0.076 ... 0.998 3.51 0.56 9.4 5

[5 rows x 12 columns]

**Daily XP650**

##### Exercise

##### Exercise

# Importing non-flat files from the web

Congrats! You've just loaded a flat file from the web into a DataFrame without first saving it locally using the pandas function pd.read\_csv(). This function is super cool because it has close relatives that allow you to load all types of files, not only flat ones. In this interactive exercise, you'll use pd.read\_excel() to import an Excel spreadsheet.

The URL of the spreadsheet is

'https://assets.datacamp.com/course/importing\_data\_into\_r/latitude.xls'

Your job is to use pd.read\_excel() to read in all of its sheets, print the sheet names and then print the head of the first sheet using its name, not its index.

Note that the output of pd.read\_excel() is a Python dictionary with sheet names as keys and corresponding DataFrames as corresponding values.

##### Instructions

**100 XP**

* Assign the URL of the file to the variable url.
* Read the file in url into a dictionary xls using pd.read\_excel() recalling that, in order to import all sheets you need to pass None to the argument sheet\_name.
* Print the names of the sheets in the Excel spreadsheet; these will be the keys of the dictionary xls.
* Print the head of the first sheet using the sheet name, not the index of the sheet! The sheet name is '1700'
* # Import package
* import pandas as pd
* # Assign url of file: url
* # Read in all sheets of Excel file: xls
* # Print the sheetnames to the shell
* # Print the head of the first sheet (using its name, NOT its index)

# Import packages import matplotlib.pyplot as plt import pandas as pd # Assign url of file: url url = 'https://assets.datacamp.com/production/course\_1606/datasets/winequality-red.csv' # Read file into a DataFrame: df df = pd.read\_csv(url, sep= ';') # Print the head of the DataFrame print(df.head()) # Plot first column of df df.iloc[:, 0].hist() plt.xlabel('fixed acidity (g(tartaric acid)/dm$^3$)') plt.ylabel('count') plt.show()

# Import package

import pandas as pd

# Assign url of file: url

url = 'https://assets.datacamp.com/course/importing\_data\_into\_r/latitude.xls'

# Read in all sheets of Excel file: xls

xls = pd.read\_excel(url, sheet\_name=None)

# Print the sheetnames to the shell

print(xls.keys())

# Print the head of the first sheet (using its name, NOT its index)

print(xls['1700'].head())

# Import package

import pandas as pd

# Assign url of file: url

url = 'https://assets.datacamp.com/course/importing\_data\_into\_r/latitude.xls'

# Read in all sheets of Excel file: xls

xls = pd.read\_excel(url, sheet\_name=None)

# Print the sheetnames to the shell

print(xls.keys())

# Print the head of the first sheet (using its name, NOT its index)

print(xls['1700'].head())

dict\_keys(['1700', '1900'])

country 1700

0 Afghanistan 34.565

1 Akrotiri and Dhekelia 34.617

2 Albania 41.312

3 Algeria 36.720

4 American Samoa -14.307

**Daily XP750**

# HTTP requests to import files from the web

**50 XP**

## 1. HTTP requests to import files from the web

Congrats on importing your first web data! In order to import files from the web,

## 2. URL

we used the urlretrieve function from urllib dot requests. Lets now unpack this a bit and, in the process, understand a few things about how the internet works. URL stands for Uniform or Universal Resource Locator and all they really are are references to web resources. The vast majority of URLs are web addresses, but they can refer to a few other things, such as file transfer protocols (FTP) and database access. We'll currently focus on those URLs that are web addresses OR the locations of websites. Such a URL consists of 2 parts, a protocol identifier http or https and a resource name such as datacamp dot com. The combination of protocol identifier and resource name uniquely specifies the web address! To explain URLs, I have introduced yet another acronym

## 3. HTTP

http, which itself stands for HyperText Transfer Protocol. Wikipedia provides a great description of HTTP. "The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web." Note that HTTPS is a more secure form of HTTP. Each time you go to a website, you are actually sending an HTTP request to a server. This request is known as a GET request, by far the most common type of HTTP request. We are actually performing a GET request when using the function urlretrieve. The ingenuity of urlretrieve also lies in fact that it not only makes a GET request but also saves the relevant data locally. In the following, you'll learn how to make more GET requests to store web data in your environment. In particular, you'll figure out how to get the HTML data from a webpage. HTML stands for Hypertext Markup Language and is the standard markup language for the web.

## 4. GET requests using urllib

To extract the html from the wikipedia home page, you import the necessary functions, specify the URL, package the GET request using the function Request, send the request and catch the response using the function urlopen. This returns an HTTPResponse object, which has an associated read method. You then apply this read method to the response, which returns the HTML as a string, which you store in the variable html. You remember to be polite and close the response!

## 5. GET requests using requests

Now we are going to do the same, however here we'll use the requests package, which provides a wonderful API for making requests. According to the requests package website. "Requests allows you to send organic, grass-fed HTTP/1 dot 1 requests, without the need for manual labor." and the following organizations claim to use requests internally: "Her Majesty's Government, Amazon, Google, Twilio, NPR, Obama for America, Twitter, Sony, and Federal U.S. Institutions that prefer to be unnamed."

## 6. GET requests using requests

Moreover, "Requests is one of the most downloaded Python packages of all time, pulling in over 7,000,000 downloads every month. All the cool kids are doing it!" Lets now see requests at work. Here, you import the package requests, specify the URL, package the request, send the request and catch the response with a single function requests dot get; apply the text method to the response which returns the HTML as a string.

## 7. Let's practice!

That's enough out of me for the time being. Let's get you hacking away at pulling down some HTML from the web using GET requests! GET coding!

**Daily XP800**

##### Exercise

##### Exercise

# Performing HTTP requests in Python using urllib

Now that you know the basics behind HTTP GET requests, it's time to perform some of your own. In this interactive exercise, you will ping our very own DataCamp servers to perform a GET request to extract information from the first coding exercise of this course, "https://campus.datacamp.com/courses/1606/4135?ex=2".

In the next exercise, you'll extract the HTML itself. Right now, however, you are going to package and send the request and then catch the response.

##### Instructions

**100 XP**

* Import the functions urlopen and Request from the subpackage urllib.request.
* Package the request to the url "https://campus.datacamp.com/courses/1606/4135?ex=2" using the function Request() and assign it to request.
* Send the request and catch the response in the variable response with the function urlopen().
* Run the rest of the code to see the datatype of response and to close the connection!
* # Import packages
* # Specify the url
* url = "https://campus.datacamp.com/courses/1606/4135?ex=2"
* # This packages the request: request
* # Sends the request and catches the response: response
* # Print the datatype of response
* print(type(response))
* # Be polite and close the response!
* response.close()

# Import package import pandas as pd # Assign url of file: url url = 'https://assets.datacamp.com/course/importing\_data\_into\_r/latitude.xls' # Read in all sheets of Excel file: xls xls = pd.read\_excel(url, sheet\_name=None) # Print the sheetnames to the shell print(xls.keys()) # Print the head of the first sheet (using its name, NOT its index) print(xls['1700'].head())

# Import packages

from urllib.request import urlopen, Request

# Specify the url

url = "https://campus.datacamp.com/courses/1606/4135?ex=2"

# This packages the request: request

request = Request(url)

# Sends the request and catches the response: response

response = urlopen(request)

# Print the datatype of response

print(type(response))

# Be polite and close the response!

response.close()

# Import packages

from urllib.request import urlopen, Request

# Specify the url

url = "https://campus.datacamp.com/courses/1606/4135?ex=2"

# This packages the request: request

request = Request(url)

# Sends the request and catches the response: response

response = urlopen(request)

# Print the datatype of response

**print(type(response))**

**html = response.read()**

**print(response)**

**# Be polite and close the response!**

**response.close()**

<class 'http.client.HTTPResponse'>

<http.client.HTTPResponse object at 0x7fbdb7762100>

# Import packages

from urllib.request import urlopen, Request

# Specify the url

url = "https://campus.datacamp.com/courses/1606/4135?ex=2"

# This packages the request: request

request = Request(url)

# Sends the request and catches the response: response

response = urlopen(request)

# Print the datatype of response

print(type(response))

# Be polite and close the response!

response.close()

<class 'http.client.HTTPResponse'>

**Daily XP900**

##### Exercise

##### Exercise

# Printing HTTP request results in Python using urllib

You have just packaged and sent a GET request to "https://campus.datacamp.com/courses/1606/4135?ex=2" and then caught the response. You saw that such a response is a http.client.HTTPResponse object. The question remains: what can you do with this response?

Well, as it came from an HTML page, you could read it to extract the HTML and, in fact, such a http.client.HTTPResponse object has an associated read() method. In this exercise, you'll build on your previous great work to extract the response and print the HTML.

##### Instructions

**100 XP**

* Send the request and catch the response in the variable response with the function urlopen(), as in the previous exercise.
* Extract the response using the read() method and store the result in the variable html.
* Print the string html.
* Hit submit to perform all of the above and to close the response: be tidy!

# Import packages

from urllib.request import urlopen, Request

# Specify the url

url = "https://campus.datacamp.com/courses/1606/4135?ex=2"

# This packages the request

request = Request(url)

# Sends the request and catches the response: response

# Extract the response: html

# Print the html

# Be polite and close the response!

response.close()

# Import packages from urllib.request import urlopen, Request # Specify the url url = "https://campus.datacamp.com/courses/1606/4135?ex=2" # This packages the request: request request = Request(url) # Sends the request and catches the response: response response = urlopen(request) # Print the datatype of response print(type(response)) # Be polite and close the response! response.close()

# Import packages

from urllib.request import urlopen, Request

# Specify the url

url = "https://campus.datacamp.com/courses/1606/4135?ex=2"

# This packages the request

request = Request(url)

# Sends the request and catches the response: response

response = urlopen(request)

# Extract the response: html

html = response.read()

# Print the html

print(html)

# Be polite and close the response!

response.close()

# Import packages

from urllib.request import urlopen, Request

# Specify the url

url = "https://campus.datacamp.com/courses/1606/4135?ex=2"

# This packages the request

request = Request(url)

# Sends the request and catches the response: response

response = urlopen(request)

# Extract the response: html

html = response.read()

# Print the html

print(html)

# Be polite and close the response!

response.close()

b'<!doctype html><html lang="en"><head><link rel="apple-touch-icon-precomposed" sizes="57x57" href="/campus/apple-touch-icon-57x57.png"><link rel="apple-touch-icon-

##### Exercise

# Performing HTTP requests in Python using requests

Now that you've got your head and hands around making HTTP requests using the urllib package, you're going to figure out how to do the same using the higher-level requests library. You'll once again be pinging DataCamp servers for their "http://www.datacamp.com/teach/documentation" page.

Note that unlike in the previous exercises using urllib, you don't have to close the connection when using requests!

##### Instructions

**100 XP**

* Import the package requests.
* Assign the URL of interest to the variable url.
* Package the request to the URL, send the request and catch the response with a single function requests.get(), assigning the response to the variable r.
* Use the text attribute of the object r to return the HTML of the webpage as a string; store the result in a variable text.
* Hit submit to print the HTML of the webpage.

# Import package

# Specify the url: url

# Packages the request, send the request and catch the response: r

# Extract the response: text

# Print the html

print(text)

# Import packages from urllib.request import urlopen, Request # Specify the url url = "https://campus.datacamp.com/courses/1606/4135?ex=2" # This packages the request request = Request(url) # Sends the request and catches the response: response response = urlopen(request) # Extract the response: html html = response.read() # Print the html print(html) # Be polite and close the response! response.close()

# Import package

import requests

# Specify the url: url

url = "http://www.datacamp.com/teach/documentation"

# Packages the request, send the request and catch the response: r

r = requests.get(url)

# Extract the response: text

text = r.text

# Print the html

print(text)

# Import packages

from urllib.request import urlopen, Request

# Specify the url

url = "https://campus.datacamp.com/courses/1606/4135?ex=2"

# This packages the request

request = Request(url)

# Sends the request and catches the response: response

response = urlopen(request)

# Extract the response: html

html = response.read()

# Print the html

print(html)

# Be polite and close the response!

response.close()

**# Import package**

**import requests**

**# Specify the url: url**

**url = "http://www.datacamp.com/teach/documentation"**

**# Packages the request, send the request and catch the response: r**

**r = requests.get(url)**

**# Extract the response: text**

**text = r.text**

**# Print the html**

**print(text)**

<!doctype html>

<html lang="en" data-direction="ltr">

<head>

**Daily XP1100**

# Scraping the web in Python

**50 XP**

## 1. Scraping the web in Python

Wow! you have just scraped HTML data from the web and you've done so using two different packages, urllib and requests. You also saw that requests provided a higher-level interface in that you needed to write less lines of to retrieve the relevant HTML as a string.

## 2. HTML

You've got the HTML of your page of interest but, generally HTML is a humble-jumble mix of both unstructured and structured data. A word on these terms: Structured data is data that has a pre-defined data model or that is organized in a defined manner. Unstructured data is data that does not possess either of these properties. HTML is interesting because, although much of it is unstructured text, it does contain tags that determine where, for examples, headings can be found, and hyperlinks.

## 3. BeautifulSoup

In general, to turn HTML that you have scraped from the world wide web into useful data, you'll need to parse it and extract structured data from it. In this video and the next few interactive exercises, we'll provide a brief introduction to how you can perform such tasks using the Python package BeautifulSoup. Lets check out the package's website. The first words at the top are: "You didn't write that awful page. You're just trying to get some data out of it. Beautiful Soup is here to help. Since 2004, it's been saving programmers hours or days of work on quick-turnaround screen scraping projects." Firstly, a word on the name of the package: BeautifulSoup? In web development, the term "tag soup" refers to structurally or syntactically incorrect HTML code written for a web page. What Beautiful Soup does best is to make tag soup beautiful again and to extract information from it with ease! In fact, the main object created and queried when using this package is called BeautifulSoup and it has a very important associated method called prettify! Lets now see BeautifulSoup in Beautiful Action!

## 4. BeautifulSoup

Once again, you use requests to scrape the HTML from the web. Then you create a BeautifulSoup object from the resulting HTML and prettify.

## 5. Prettified Soup

Printing the prettified Soup and the original HTML, you can see that for, example, the prettified Soup is indented in the way you would expect properly written HTML to be.

## 6. Exploring BeautifulSoup

You'll explore a few of the methods that you can apply to your soupified HTML in the following exercises, such as title and get\_text, which extract the title and text, respectively.

## 7. Exploring BeautifulSoup

You'll also work with the Soupy method find\_all in order to extract the URLs of all of the hyperlinks in the HTML. These are merely a few of many methods existing in BeautifulSoup to extract data from HTML. If, after completing these exercises, you find yourself thirsting for more BeautifulSoup, there are plenty of great resources on their website.

## 8. Let's practice!

Okay, now it's your turn to jump into the deep end of the proverbial soup bowl! Happy hacking!

**Daily XP1150**

##### Exercise

##### Exercise

# Parsing HTML with BeautifulSoup

In this interactive exercise, you'll learn how to use the BeautifulSoup package to parse, prettify and extract information from HTML. You'll scrape the data from the webpage of Guido van Rossum, Python's very own [Benevolent Dictator for Life](https://en.wikipedia.org/wiki/Benevolent_dictator_for_life). In the following exercises, you'll prettify the HTML and then extract the text and the hyperlinks.

The URL of interest is url = 'https://www.python.org/~guido/'.

##### Instructions

**100 XP**

* Import the function BeautifulSoup from the package bs4.
* Assign the URL of interest to the variable url.
* Package the request to the URL, send the request and catch the response with a single function requests.get(), assigning the response to the variable r.
* Use the text attribute of the object r to return the HTML of the webpage as a string; store the result in a variable html\_doc.
* Create a BeautifulSoup object soup from the resulting HTML using the function BeautifulSoup().
* Use the method prettify() on soup and assign the result to pretty\_soup.
* Hit submit to print to prettified HTML to your shell!

# Import packages

import requests

from \_\_\_\_ import \_\_\_\_

# Specify url: url

# Package the request, send the request and catch the response: r

# Extracts the response as html: html\_doc

# Create a BeautifulSoup object from the HTML: soup

# Prettify the BeautifulSoup object: pretty\_soup

# Print the response

print(pretty\_soup)

# Import package import requests # Specify the url: url url = "http://www.datacamp.com/teach/documentation" # Packages the request, send the request and catch the response: r r = requests.get(url) # Extract the response: text text = r.text # Print the html print(text)

# Import packages

import requests

from bs4 import BeautifulSoup

# Specify url: url

url = 'https://www.python.org/~guido/'

# Package the request, send the request and catch the response: r

r = requests.get(url)

# Extracts the response as html: html\_doc

html\_doc = r.text

# Create a BeautifulSoup object from the HTML: soup

soup = BeautifulSoup(html\_doc)

# Prettify the BeautifulSoup object: pretty\_soup

pretty\_soup = soup.prettify()

# Print the response

print(pretty\_soup)

# Import package

import requests

# Specify the url: url

url = "http://www.datacamp.com/teach/documentation"

# Packages the request, send the request and catch the response: r

r = requests.get(url)

# Extract the response: text

text = r.text

# Print the html

print(text)

# Import packages

**import requests**

**from bs4 import BeautifulSoup**

**# Specify url: url**

**url = 'https://www.python.org/~guido/'**

**# Package the request, send the request and catch the response: r**

**r = requests.get(url)**

**# Extracts the response as html: html\_doc**

**html\_doc = r.text**

**# Create a BeautifulSoup object from the HTML: soup**

**soup = BeautifulSoup(html\_doc)**

**# Prettify the BeautifulSoup object: pretty\_soup**

**pretty\_soup = soup.prettify()**

**# Print the response**

**print(pretty\_soup)**

<html>

<head>

<title>

Guido's Personal Home Page

</title>

</head>

<body bgcolor="#FFFFFF" text="#000000">

<!-- Built from main -->

<h1>

<a href="pics.html">

<img border="0" src="images/IMG\_2192.jpg"/>

</a>

Guido van Rossum - Personal Home Page

<a href="pics.html">

<img border="0" height="216" src="images/guido-headshot-2019.jpg" width="270"/>

</a>

</h1>

<p>

<a href="http://www.washingtonpost.com/wp-srv/business/longterm/microsoft/stories/1998/raymond120398.htm">

<i>

"Gawky and proud of it."

</i>

</a>

<h3>

<a href="images/df20000406.jpg">

Who I Am

</a>

</h3>

<p>

Read

my

<a href="http://neopythonic.blogspot.com/2016/04/kings-day-speech.html">

"King's

Day Speech"

</a>

for some inspiration.

<p>

I am the author of the

<a href="http://www.python.org">

Python

</a>

programming language. See also my

<a href="Resume.html">

resume

</a>

**Daily XP1250**

##### Exercise

##### Exercise

# Turning a webpage into data using BeautifulSoup: getting the text

As promised, in the following exercises, you'll learn the basics of extracting information from HTML soup. In this exercise, you'll figure out how to extract the text from the BDFL's webpage, along with printing the webpage's title.

##### Instructions

**100 XP**

* In the sample code, the HTML response object html\_doc has already been created: your first task is to Soupify it using the function BeautifulSoup() and to assign the resulting soup to the variable soup.
* Extract the title from the HTML soup soup using the attribute title and assign the result to guido\_title.
* Print the title of Guido's webpage to the shell using the print() function.
* Extract the text from the HTML soup soup using the method get\_text() and assign to guido\_text.
* Hit submit to print the text from Guido's webpage to the shell.
* # Import packages
* import requests
* from bs4 import BeautifulSoup
* # Specify url: url
* url = 'https://www.python.org/~guido/'
* # Package the request, send the request and catch the response: r
* r = requests.get(url)
* # Extract the response as html: html\_doc
* html\_doc = r.text
* # Create a BeautifulSoup object from the HTML: soup
* # Get the title of Guido's webpage: guido\_title
* # Print the title of Guido's webpage to the shell
* # Get Guido's text: guido\_text
* # Print Guido's text to the shell
* print(guido\_text)

# Import packages import requests from bs4 import BeautifulSoup # Specify url: url url = 'https://www.python.org/~guido/' # Package the request, send the request and catch the response: r r = requests.get(url) # Extracts the response as html: html\_doc html\_doc = r.text # Create a BeautifulSoup object from the HTML: soup soup = BeautifulSoup(html\_doc) # Prettify the BeautifulSoup object: pretty\_soup pretty\_soup = soup.prettify() # Print the response print(pretty\_soup)

# Import packages

import requests

from bs4 import BeautifulSoup

# Specify url: url

url = 'https://www.python.org/~guido/'

# Package the request, send the request and catch the response: r

r = requests.get(url)

# Extract the response as html: html\_doc

html\_doc = r.text

# Create a BeautifulSoup object from the HTML: soup

soup = BeautifulSoup(html\_doc)

# Get the title of Guido's webpage: guido\_title

guido\_title = soup.title

# Print the title of Guido's webpage to the shell

print(guido\_title)

# Get Guido's text: guido\_text

guido\_text = soup.get\_text()

# Print Guido's text to the shell

print(guido\_text)

# Import packages

import requests

from bs4 import BeautifulSoup

# Specify url: url

url = 'https://www.python.org/~guido/'

# Package the request, send the request and catch the response: r

r = requests.get(url)

# Extracts the response as html: html\_doc

html\_doc = r.text

# Create a BeautifulSoup object from the HTML: soup

soup = BeautifulSoup(html\_doc)

# Prettify the BeautifulSoup object: pretty\_soup

pretty\_soup = soup.prettify()

# Print the response

print(pretty\_soup)

# Import packages

import requests

from bs4 import BeautifulSoup

# Specify url: url

url = 'https://www.python.org/~guido/'

# Package the request, send the request and catch the response: r

r = requests.get(url)

# Extract the response as html: html\_doc

html\_doc = r.text

# Create a BeautifulSoup object from the HTML: soup

soup = BeautifulSoup(html\_doc)

# Get the title of Guido's webpage: guido\_title

guido\_title = soup.title

# Print the title of Guido's webpage to the shell

print(guido\_title)

# Get Guido's text: guido\_text

guido\_text = soup.get\_text()

# Print Guido's text to the shell

print(guido\_text)

<title>Guido's Personal Home Page</title>

Guido's Personal Home Page

Guido van Rossum - Personal Home Page

"Gawky and proud of it."

Who I Am

Read

my "King's

Day Speech" for some inspiration.

I am the author of the Python

programming language. See also my resume

and my publications list, a brief bio, assorted writings, presentations and interviews (all about Python), some

pictures of me,

my new blog, and

my old

blog on Artima.com. I am

@gvanrossum on Twitter.

I am currently a Distinguished Engineer at Microsoft.

I have worked for Dropbox, Google, Elemental Security, Zope

Corporation, BeOpen.com, CNRI, CWI, and SARA. (See

my resume.) I created Python while at CWI.

How to Reach Me

You can send email for me to guido (at) python.org.

I read everything sent there, but I receive too much email to respond

to everything.

My Name

My name often poses difficulties for Americans.

Pronunciation: in Dutch, the "G" in Guido is a hard G,

pronounced roughly like the "ch" in Scottish "loch". (Listen to the

sound clip.) However, if you're

American, you may also pronounce it as the Italian "Guido". I'm not

too worried about the associations with mob assassins that some people

have. :-)

Spelling: my last name is two words, and I'd like to keep it

that way, the spelling on some of my credit cards notwithstanding.

Dutch spelling rules dictate that when used in combination with my

first name, "van" is not capitalized: "Guido van Rossum". But when my

last name is used alone to refer to me, it is capitalized, for

example: "As usual, Van Rossum was right."

Alphabetization: in America, I show up in the alphabet under

"V". But in Europe, I show up under "R". And some of my friends put

me under "G" in their address book...

More Hyperlinks

Here's a collection of essays relating to Python

that I've written, including the foreword I wrote for Mark Lutz' book

"Programming Python".

I own the official

Python license.

The Audio File Formats FAQ

I was the original creator and maintainer of the Audio File Formats

FAQ. It is now maintained by Chris Bagwell

at http://www.cnpbagwell.com/audio-faq. And here is a link to

SOX, to which I contributed

some early code.

"On the Internet, nobody knows you're

a dog."

##### Exercise

##### Exercise

# Turning a webpage into data using BeautifulSoup: getting the hyperlinks

In this exercise, you'll figure out how to extract the URLs of the hyperlinks from the BDFL's webpage. In the process, you'll become close friends with the soup method find\_all().

##### Instructions

**100 XP**

* Use the method find\_all() to find all hyperlinks in soup, remembering that hyperlinks are defined by the HTML tag <a> but passed to find\_all() without angle brackets; store the result in the variable a\_tags.
* The variable a\_tags is a results set: your job now is to enumerate over it, using a for loop and to print the actual URLs of the hyperlinks; to do this, for every element link in a\_tags, you want to print() link.get('href').
* # Import packages
* import requests
* from bs4 import BeautifulSoup
* # Specify url
* url = 'https://www.python.org/~guido/'
* # Package the request, send the request and catch the response: r
* r = requests.get(url)
* # Extracts the response as html: html\_doc
* html\_doc = r.text
* # create a BeautifulSoup object from the HTML: soup
* soup = BeautifulSoup(html\_doc)
* # Print the title of Guido's webpage
* print(soup.title)
* # Find all 'a' tags (which define hyperlinks): a\_tags
* # Print the URLs to the shell
* for \_\_\_\_ in \_\_\_\_:
* \_\_\_\_

# Import packages import requests from bs4 import BeautifulSoup # Specify url: url url = 'https://www.python.org/~guido/' # Package the request, send the request and catch the response: r r = requests.get(url) # Extract the response as html: html\_doc html\_doc = r.text # Create a BeautifulSoup object from the HTML: soup soup = BeautifulSoup(html\_doc) # Get the title of Guido's webpage: guido\_title guido\_title = soup.title # Print the title of Guido's webpage to the shell print(guido\_title) # Get Guido's text: guido\_text guido\_text = soup.get\_text() # Print Guido's text to the shell print(guido\_text)

# Import packages

import requests

from bs4 import BeautifulSoup

# Specify url

url = 'https://www.python.org/~guido/'

# Package the request, send the request and catch the response: r

r = requests.get(url)

# Extracts the response as html: html\_doc

html\_doc = r.text

# create a BeautifulSoup object from the HTML: soup

soup = BeautifulSoup(html\_doc)

# Print the title of Guido's webpage

print(soup.title)

# Find all 'a' tags (which define hyperlinks): a\_tags

a\_tags = soup.find\_all('a')

# Print the URLs to the shell

for link in a\_tags:

    print(link.get('href'))

# Import packages

import requests

from bs4 import BeautifulSoup

# Specify url

url = 'https://www.python.org/~guido/'

# Package the request, send the request and catch the response: r

r = requests.get(url)

# Extracts the response as html: html\_doc

html\_doc = r.text

# create a BeautifulSoup object from the HTML: soup

soup = BeautifulSoup(html\_doc)

# Print the title of Guido's webpage

print(soup.title)

# Find all 'a' tags (which define hyperlinks): a\_tags

a\_tags = soup.find\_all('a\_tags')

# Print the URLs to the shell

for link in a\_tags:

print(link.get('href'))

<title>Guido's Personal Home Page</title>

<script.py> output:

<title>Guido's Personal Home Page</title>

# Import packages

import requests

from bs4 import BeautifulSoup

# Specify url

url = 'https://www.python.org/~guido/'

# Package the request, send the request and catch the response: r

r = requests.get(url)

# Extracts the response as html: html\_doc

html\_doc = r.text

# create a BeautifulSoup object from the HTML: soup

soup = BeautifulSoup(html\_doc)

# Print the title of Guido's webpage

print(soup.title)

# Find all 'a' tags (which define hyperlinks): a\_tags

a\_tags = soup.find\_all('a')

# Print the URLs to the shell

for link in a\_tags:

print(link.get('href'))

<title>Guido's Personal Home Page</title>

pics.html

pics.html

http://www.washingtonpost.com/wp-srv/business/longterm/microsoft/stories/1998/raymond120398.htm

images/df20000406.jpg

http://neopythonic.blogspot.com/2016/04/kings-day-speech.html

http://www.python.org

Resume.html

Publications.html

bio.html

http://legacy.python.org/doc/essays/

http://legacy.python.org/doc/essays/ppt/

interviews.html

pics.html

http://neopythonic.blogspot.com

http://www.artima.com/weblogs/index.jsp?blogger=12088

https://twitter.com/gvanrossum

Resume.html

guido.au

http://legacy.python.org/doc/essays/

images/license.jpg

http://www.cnpbagwell.com/audio-faq

http://sox.sourceforge.net/

images/internetdog.gif

**Daily XP1450**

# Introduction to APIs and JSONs

**50 XP**

## 1. Introduction to APIs and JSONs

In this chapter, you'll explore pulling data from the web even further by

## 2. APIs

learning how to interact with APIs, or Application Programming Interfaces. An API is a set of protocols and routines for building and interacting with software applications. In particular, you'll learn how to use the Open Movie Database API and, in the next chapter,

## 3. APIs

the Twitter API to pull data from both applications, while learning about API interaction best practices. A standard form for transferring data through APIs is

## 4. JSONs

the JSON file format, so in this video, we'll focus our attention squarely on these. Then we'll move onto actually getting data from APIs. JSON is an acronym that is short for JavaScript Object Notation. It is a file format that arose out of a growing need for real-time server-to-browser communication that wouldn't necessarily rely on Flash or Java and was first specified and also popularized by Douglas Crockford, an American programmer and entrepreneur. One of the cool things about JSONs is that they're human readable, that is, they can naturally be read by humans unlike, for example, pickled files, as we saw in the previous course. As they're human readable, let's check one out!

## 5. JSONs

Here you see a JSON from the OMDB OR Open Movie Database API. In particular, this is JSON containing information about the movie Snakes on a Plane. First notice that the JSON consists of name-value pairs separated by commas. This will remind you of the key-value pairs in a Python dictionary! We'll see in a minute that, for this reason, when loading JSONs into Python, it is natural to store them in a dict. The keys in JSONs will always be strings enclosed in quotation marks. The values can be strings, integers, arrays or even objects. Such an object can even be a JSON and then you have nested JSONs but we won't go further into these here. In this case of the Snakes on a Plane JSON, all the values are strings and we can see this from the quotation marks.

## 6. JSONs

The value corresponding to the key 'Title' is the title of the movie as a string: Snakes on a Plane.

## 7. JSONs

The value corresponding to the key 'Year' is the year of release as a string: 2006 and so on. There's the rating, the runtime, director, writers, plot, language and much more! You'll soon learn how to use the OMDB API and Python to automate retrieval of such data, but first you'll figure out how to load JSONs from a local directory.

## 8. Loading JSONs in Python

Lets say that I had the JSON stored in my working directory as 'snakes dot json'. To load the JSON into my Python environment, I would first import the package json and then open a connection to the file and use the function json dot load to load the JSON. If I then check the datatype of json\_data by executing type(json\_data), I see that Python cleverly imported the JSON as a dictionary!

## 9. Exploring JSONs in Python

To print the key-value pairs to the console, I can then iterate over the key-value pairs using a for loop. Now it's your turn to test your JSON skills

## 10. Let's practice!

so get coding!

# Pop quiz: What exactly is a JSON?

Which of the following is **NOT** true of the JSON file format?

##### Answer the question

**50XP**

#### Possible Answers



JSONs consist of key-value pairs.

press1



JSONs are human-readable.

press2



The JSON file format arose out of a growing need for real-time server-to-browser communication.

press3



T**he function json.load() will load the JSON into Python as a list. This the answer**

press4



The function json.load() will load the JSON into Python as a dictionary.

press5

**Daily XP1550**

##### Exercise

##### Exercise

# Loading and exploring a JSON

Now that you know what a JSON is, you'll load one into your Python environment and explore it yourself. Here, you'll load the JSON 'a\_movie.json' into the variable json\_data, which will be a dictionary. You'll then explore the JSON contents by printing the key-value pairs of json\_data to the shell.

##### Instructions

**100 XP**

* Load the JSON 'a\_movie.json' into the variable json\_data within the context provided by the with statement. To do so, use the function json.load() within the context manager.
* Use a for loop to print all key-value pairs in the dictionary json\_data. Recall that you can access a value in a dictionary using the syntax: dictionary[key].
* # Load JSON: json\_data
* with open("a\_movie.json") as json\_file:
* \_\_\_\_
* # Print each key-value pair in json\_data
* for k in json\_data.keys():
* print(k + ': ', \_\_\_\_)

# Import packages import requests from bs4 import BeautifulSoup # Specify url url = 'https://www.python.org/~guido/' # Package the request, send the request and catch the response: r r = requests.get(url) # Extracts the response as html: html\_doc html\_doc = r.text # create a BeautifulSoup object from the HTML: soup soup = BeautifulSoup(html\_doc) # Print the title of Guido's webpage print(soup.title) # Find all 'a' tags (which define hyperlinks): a\_tags a\_tags = soup.find\_all('a') # Print the URLs to the shell for link in a\_tags: print(link.get('href'))

# Load JSON: json\_data

with open("a\_movie.json") as json\_file:

    json\_data = json.load(json\_file)

# Print each key-value pair in json\_data

for k in json\_data.keys():

    print(k + ': ', json\_data[k])

# Load JSON: json\_data

with open("a\_movie.json") as json\_file:

json\_data = json.load(json\_file)

# Print each key-value pair in json\_data

for k in json\_data.keys():

print(k + ': ', json\_data[k])

Title: The Social Network

Year: 2010

Rated: PG-13

Released: 01 Oct 2010

Runtime: 120 min

Genre: Biography, Drama

Director: David Fincher

Writer: Aaron Sorkin, Ben Mezrich

Actors: Jesse Eisenberg, Andrew Garfield, Justin Timberlake

Plot: As Harvard student Mark Zuckerberg creates the social networking site that would become known as Facebook, he is sued by the twins who claimed he stole their idea, and by the co-founder who was later squeezed out of the business.

Language: English, French

Country: United States

Awards: Won 3 Oscars. 173 wins & 186 nominations total

Poster: https://m.media-amazon.com/images/M/MV5BOGUyZDUxZjEtMmIzMC00MzlmLTg4MGItZWJmMzBhZjE0Mjc1XkEyXkFqcGdeQXVyMTMxODk2OTU@.\_V1\_SX300.jpg

Ratings: [{'Source': 'Internet Movie Database', 'Value': '7.8/10'}, {'Source': 'Rotten Tomatoes', 'Value': '96%'}, {'Source': 'Metacritic', 'Value': '95/100'}]

Metascore: 95

imdbRating: 7.8

imdbVotes: 710,400

imdbID: tt1285016

Type: movie

DVD: 11 Jan 2011

BoxOffice: $96,962,694

Production: N/A

Website: N/A

Response: True

**Daily XP1650**

##### Exercise

##### Exercise

# Pop quiz: Exploring your JSON

Load the JSON 'a\_movie.json' into a variable, which will be a dictionary. Do so by copying, pasting and executing the following code in the IPython Shell:

import json

with open("a\_movie.json") as json\_file:

json\_data = json.load(json\_file)

Print the values corresponding to the keys 'Title' and 'Year' and answer the following question about the movie that the JSON describes:

Which of the following statements is true of the movie in question?

##### Instructions

**50 XP**

##### Possible Answers



The title is 'Kung Fu Panda' and the year is 2010.



The title is 'Kung Fu Panda' and the year is 2008.



**The title is 'The Social Network' and the year is 2010. This is the correct answer**



The title is 'The Social Network' and the year is 2008.

# Load JSON: json\_data with open("a\_movie.json") as json\_file: json\_data = json.load(json\_file) # Print each key-value pair in json\_data for k in json\_data.keys(): print(k + ': ', json\_data[k])

In [3]:

for key, value in json\_data.items(): print(key + ':', value)

Title: The Social Network

In [6]:

for key in json\_data.keys(): print(key + ':', json\_data[key])

Title: The Social Network

Year: 2010

Rated: PG-13

**Daily XP1700**

# APIs and interacting with the world wide web

**50 XP**

## 1. APIs and interacting with the world wide web

Congrats on making it through your crash course in JSONs! JSONs are everywhere and one of the main motivating reasons for getting to know how to work with them as a Data Scientist is that much of the data that you'll get from APIs are packaged as JSONs.

## 2. Herein, you’ll learn

In this video, you'll learn what APIs are, why they are so important, and see a number of illustrative examples. In the subsequent interactive exercises, you'll gain valuable practice connecting to a variety of APIs, pulling and parsing data from them.

## 3. What is an API?

So what is an API and why are they so important? Simply put, an API is a set of protocols and routines for building and interacting with software applications. Another way to think of it is that an API is a bunch of code that allows two software programs to communicate with each other. For example, if you wanted to stream twitter data by writing some Python code, you would use the Twitter API. If you wanted to automate pulling and processing information

## 4. What is an API?

from Wikipedia in your programming language of choice, you could do so using the Wikipedia API.

## 5. APIs are everywhere

Using such APIs have now become standard ways of interacting with such applications. Twitter has an API that is used by marketing companies and social scientists engaged in research concerning social networks.

## 6. APIs are everywhere

Uber,

## 7. APIs are everywhere

Facebook and

## 8. APIs are everywhere

Instagram all have APIs. Now let's figure out how to connect to an API and how to pull data from it.

## 9. Connecting to an API in Python

In this example, we'll pull movie data from the Open Movie Database, or OMDB, API. Once again, you'll use the ever-elegant requests library. You import requests and assign the URL of interest to the variable url. You then package and send the request to the URL, which describes your API query, and catch the response in one line of code. Thanks again, requests package! Another really cool aspect of the requests package is that the Response objects, such as r, have an associate method json, which is a built-in JSON decoder for when we're dealing with JSON data. This returns a dictionary and we can then print all the key-value pairs to check out what we pulled from the OMBD API!

## 10. What was that URL?

Now the last thing to discuss is how the URL we used actually pulled data from the API. To do so, lets break it up into chunks. The http signifies that we're making an HTTP request, the 'www dot omdb dot api' that we're querying the OMDB API, then there's the "?t equals hackers" which is the really interesting part and something we haven't discussed yet in this course. This string that begins with a question mark is called a Query String. Query Strings are parts of URLs that do not necessarily fit into conventional a hierarchical path structure. What follows the question mark in the query string is the query we are making to the OMBD API. The query we just made was simple : querying 't equals hackers' asked the API to return the data about the movie with the title Hackers. The 't' in the query stood for title.

## 11. OMDb API

We knew that this was how to perform such a query from the documentation on the OMDB API's homepage. Under "Usage" here, they state explicitly that 'Send all data requests to: http:// www dot omdbapi dot com /?'.

## 12. OMDb API

They also have a query string parameters table that shows how to query a particular title or a particular movie ID.

## 13. It’s a regular URL!

It is also worth mentioning that there is nothing special about this URL and so you can also navigate to it in your browser of choice. It will generally look like this. I like to use a Chrome extension called JSON formatter to make it a bit prettier. Alright. Now you know all about APIs and have a basic practical understanding of how to query them,

## 14. Let's practice!

lets get you writing some Python to extract some data from a number of APIs! Happy coding!

# Pop quiz: What's an API?

Which of the following statements about APIs is NOT true?

##### Answer the question

**50XP**

#### Possible Answers



An API is a set of protocols and routines for building and interacting with software applications.

press1



API is an acronym and is short for Application Program interface.

press2



It is common to pull data from APIs in the JSON file format.

press3



**All APIs transmit data only in the JSON file format. This is the correct answer**

press4



An API is a bunch of code that allows two software programs to communicate with each other.

press5

**Daily XP1800**

##### Exercise

##### Exercise

# API requests

Now it's your turn to pull some movie data down from the Open Movie Database (OMDB) using their API. The movie you'll query the API about is The Social Network. Recall that, in the video, to query the API about the movie Hackers, Hugo's query string was 'http://www.omdbapi.com/?t=hackers' and had a single argument t=hackers.

Note: recently, OMDB has changed their API: you now also have to specify an API key. This means you'll have to add another argument to the URL: apikey=72bc447a.

##### Instructions

**100 XP**

* Import the requests package.
* Assign to the variable url the URL of interest in order to query 'http://www.omdbapi.com' for the data corresponding to the movie The Social Network. The query string should have two arguments: apikey=72bc447a and t=the+social+network. You can combine them as follows: apikey=72bc447a&t=the+social+network.
* Print the text of the response object r by using its text attribute and passing the result to the print() function.
* # Import requests package
* # Assign URL to variable: url
* # Package the request, send the request and catch the response: r
* r = requests.get(url)
* # Print the text of the response

# Load JSON: json\_data with open("a\_movie.json") as json\_file: json\_data = json.load(json\_file) # Print each key-value pair in json\_data for k in json\_data.keys(): print(k + ': ', json\_data[k])

# Import requests package

import requests

# Assign URL to variable: url

url = 'http://www.omdbapi.com/?apikey=72bc447a&t=the+social+network'

# Package the request, send the request and catch the response: r

r = requests.get(url)

# Print the text of the response

print(r.text)

# Import requests package

import requests

# Assign URL to variable: url

url = 'http://www.omdbapi.com/?apikey=72bc447a&t=the+social+network'

# Package the request, send the request and catch the response: r

r = requests.get(url)

# Print the text of the response

print(r.text)

{"Title":"The Social Network","Year":"2010","Rated":"PG-13","Released":"01 Oct 2010","Runtime":"120 min","Genre":"Biography, Drama","Director":"David Fincher","Writer":"Aaron Sorkin, Ben Mezrich","Actors":"Jesse Eisenberg, Andrew Garfield, Justin Timberlake","Plot":"As Harvard student Mark Zuckerberg creates the social networking site that would become known as Facebook, he is sued by the twins who claimed he stole their idea, and by the co-founder who was later squeezed out of the business.","Language":"English, French","Country":"United States","Awards":"Won 3 Oscars. 173 wins & 186 nominations total","Poster":"https://m.media-amazon.com/images/M/MV5BOGUyZDUxZjEtMmIzMC00MzlmLTg4MGItZWJmMzBhZjE0Mjc1XkEyXkFqcGdeQXVyMTMxODk2OTU@.\_V1\_SX300.jpg","Ratings":[{"Source":"Internet Movie Database","Value":"7.8/10"},{"Source":"Rotten Tomatoes","Value":"96%"},{"Source":"Metacritic","Value":"95/100"}],"Metascore":"95","imdbRating":"7.8","imdbVotes":"710,400","imdbID":"tt1285016","Type":"movie","DVD":"11 Jan 2011","BoxOffice":"$96,962,694","Production":"N/A","Website":"N/A","Response":"True"}

**Daily XP1900**

##### Exercise

##### Exercise

# JSON–from the web to Python

Wow, congrats! You've just queried your first API programmatically in Python and printed the text of the response to the shell. However, as you know, your response is actually a JSON, so you can do one step better and decode the JSON. You can then print the key-value pairs of the resulting dictionary. That's what you're going to do now!

##### Instructions

**100 XP**

* Pass the variable url to the requests.get() function in order to send the relevant request and catch the response, assigning the resultant response message to the variable r.
* Apply the json() method to the response object r and store the resulting dictionary in the variable json\_data.
* Hit submit to print the key-value pairs of the dictionary json\_data to the shell.
* # Import package
* import requests
* # Assign URL to variable: url
* url = 'http://www.omdbapi.com/?apikey=72bc447a&t=social+network'
* # Package the request, send the request and catch the response: r
* # Decode the JSON data into a dictionary: json\_data
* # Print each key-value pair in json\_data
* for k in json\_data.keys():
* print(k + ': ', json\_data[k])

# Import requests package import requests # Assign URL to variable: url url = 'http://www.omdbapi.com/?apikey=72bc447a&t=the+social+network' # Package the request, send the request and catch the response: r r = requests.get(url) # Print the text of the response print(r.text)

# Import package

import requests

# Assign URL to variable: url

url = 'http://www.omdbapi.com/?apikey=72bc447a&t=social+network'

# Package the request, send the request and catch the response: r

r = requests.get(url)

# Decode the JSON data into a dictionary: json\_data

json\_data = r.json()

# Print each key-value pair in json\_data

for k in json\_data.keys():

    print(k + ': ', json\_data[k])

# Import package

import requests

# Assign URL to variable: url

url = 'http://www.omdbapi.com/?apikey=72bc447a&t=social+network'

# Package the request, send the request and catch the response: r

r = requests.get(url)

# Decode the JSON data into a dictionary: json\_data

json\_data = r.json()

# Print each key-value pair in json\_data

for k in json\_data.keys():

print(k + ': ', json\_data[k])

Title: The Social Network

Year: 2010

Rated: PG-13

Released: 01 Oct 2010

Runtime: 120 min

Genre: Biography, Drama

Director: David Fincher

Writer: Aaron Sorkin, Ben Mezrich

Actors: Jesse Eisenberg, Andrew Garfield, Justin Timberlake

Plot: As Harvard student Mark Zuckerberg creates the social networking site that would become known as Facebook, he is sued by the twins who claimed he stole their idea, and by the co-founder who was later squeezed out of the business.

Language: English, French

Country: United States

Awards: Won 3 Oscars. 173 wins & 186 nominations total

Poster: https://m.media-amazon.com/images/M/MV5BOGUyZDUxZjEtMmIzMC00MzlmLTg4MGItZWJmMzBhZjE0Mjc1XkEyXkFqcGdeQXVyMTMxODk2OTU@.\_V1\_SX300.jpg

Ratings: [{'Source': 'Internet Movie Database', 'Value': '7.8/10'}, {'Source': 'Rotten Tomatoes', 'Value': '96%'}, {'Source': 'Metacritic', 'Value': '95/100'}]

Metascore: 95

imdbRating: 7.8

imdbVotes: 710,400

imdbID: tt1285016

Type: movie

DVD: 11 Jan 2011

BoxOffice: $96,962,694

Production: N/A

Website: N/A

Response: True

**Daily XP100**

##### Exercise

##### Exercise

# Checking out the Wikipedia API

You're doing so well and having so much fun that we're going to throw one more API at you: the Wikipedia API (documented [here](https://www.mediawiki.org/wiki/API:Main_page)). You'll figure out how to find and extract information from the Wikipedia page for Pizza. What gets a bit wild here is that your query will return nested JSONs, that is, JSONs with JSONs, but Python can handle that because it will translate them into dictionaries within dictionaries.

The URL that requests the relevant query from the Wikipedia API is

https://en.wikipedia.org/w/api.php?action=query&prop=extracts&format=json&exintro=&titles=pizza

##### Instructions

**100 XP**

* Assign the relevant URL to the variable url.
* Apply the json() method to the response object r and store the resulting dictionary in the variable json\_data.
* The variable pizza\_extract holds the HTML of an extract from Wikipedia's Pizza page as a string; use the function print() to print this string to the shell.
* # Import package
* import requests
* # Assign URL to variable: url
* # Package the request, send the request and catch the response: r
* r = requests.get(url)
* # Decode the JSON data into a dictionary: json\_data
* # Print the Wikipedia page extract
* pizza\_extract = json\_data['query']['pages']['24768']['extract']
* \_\_\_\_

# Import package import requests # Assign URL to variable: url url = 'http://www.omdbapi.com/?apikey=72bc447a&t=social+network' # Package the request, send the request and catch the response: r r = requests.get(url) # Decode the JSON data into a dictionary: json\_data json\_data = r.json() # Print each key-value pair in json\_data for k in json\_data.keys(): print(k + ': ', json\_data[k])

# Import package

import requests

# Assign URL to variable: url

url = 'https://en.wikipedia.org/w/api.php?action=query&prop=extracts&format=json&exintro=&titles=pizza'

# Package the request, send the request and catch the response: r

r = requests.get(url)

# Decode the JSON data into a dictionary: json\_data

json\_data = r.json()

# Print the Wikipedia page extract

pizza\_extract = json\_data['query']['pages']['24768']['extract']

print(pizza\_extract)

# Import package

import requests

# Assign URL to variable: url

url = 'https://en.wikipedia.org/w/api.php?action=query&prop=extracts&format=json&exintro=&titles=pizza'

# Package the request, send the request and catch the response: r

r = requests.get(url)

# Decode the JSON data into a dictionary: json\_data

json\_data = r.json()

# Print the Wikipedia page extract

pizza\_extract = json\_data['query']['pages']['24768']['extract']

print(pizza\_extract)

<link rel="mw-deduplicated-inline-style" href="mw-data:TemplateStyles:r1033289096">

<p class="mw-empty-elt">

</p>

<p><b>Pizza</b> (<span><small>English: </small></span> <i title="English pronunciation respelling"><span>PEET</span>-sə</i>, <small>Italian: </small><span title="Representation in the International Phonetic Alphabet (IPA)" lang="it-Latn-fonipa">[ˈpittsa]</span>, <small>Neapolitan: </small><span title="Representation in the International Phonetic Alphabet (IPA)" lang="nap-Latn-fonipa">[ˈpittsə]</span>) is a dish of Italian origin consisting of a usually round, flat base of leavened wheat-based dough topped with tomatoes, cheese, and often various other ingredients (such as various types of sausage, anchovies, mushrooms, onions, olives, vegetables, meat, ham, etc.), which is then baked at a high temperature, traditionally in a wood-fired oven. A small pizza is sometimes called a pizzetta. A person who makes pizza is known as a <b>pizzaiolo</b>.

</p><p>In Italy, pizza served in a restaurant is presented unsliced, and is eaten with the use of a knife and fork. In casual settings, however, it is cut into wedges to be eaten while held in the hand.

</p><p>The term <i>pizza</i> was first recorded in the 10th century in a Latin manuscript from the Southern Italian town of Gaeta in Lazio, on the border with Campania. Modern pizza was invented in Naples, and the dish and its variants have since become popular in many countries. It has become one of the most popular foods in the world and a common fast food item in Europe, the Americas and Australasia; available at pizzerias (restaurants specializing in pizza), restaurants offering Mediterranean cuisine, via pizza delivery, and as street food. Various food companies sell ready-baked pizzas, which may be frozen, in grocery stores, to be reheated in a home oven.

</p><p>In 2017, the wor

# nd Authentication

**50 XP**

## 1. The Twitter API and Authentication

Congratulations on interacting with your very first APIs and getting data from them! You're on the home stretch now.

## 2. Herein, you’ll learn

As a final deep dive, you're going to stream data from the Twitter API. You'll learn how to filter incoming tweets for keywords, you'll learn about the principles of API authentication and OAuth. You'll also learn the basics of the package

## 3. Herein, you’ll learn

tweepy, which many people in PythonLand use to interact with the Twitter API.

## 4. Access the Twitter API

One of the first major differences between the Twitter API and all the APIs you have seen so far is that you were able to access all the others anonymously and Twitter requires that you have an account. In order gain access to the Twitter API, one needs to create a twitter account if you don't already have one,

## 5. Access the Twitter API

log into Twitter Apps and click "Create a New App" - you'll need to agree to a variety of terms and conditions here,

## 6. Access the Twitter API

then , go to your "Keys and Access Tokens" tab and Copy your API key, your API secret,

## 7. Access the Twitter API

your Access Token and your Access Token secret. These are the Authentication credentials that will allow you to access the Twitter API from Python. In the following interactive exercises, we won't require that you create your own Twitter account and App: we'll do a mock run-through of how you would stream data and analyze it as if you had done so.

## 8. Twitter has a number of APIs

It is now important to mention that Twitter has a number of APIs. Firstly, they have a REST API; we won't go into the gory details of REST APIs here but I'll say two things - one: REST is short for Representational State Transfer; two: Twitter's REST API allows the user to "read and write Twitter data". In order to "monitor or process Tweets in real-time",

## 9. Twitter has a number of APIs

that is, to stream Twitter data, however, we'll want to use Twitter's Streaming API. In particular,

## 10. Twitter has a number of APIs

we'll use the public stream, which Twitter's API documentation defines as "Streams of the public data flowing through Twitter." The Public Stream itself contains a number of options. As we want to read and process tweets,

## 11. Twitter has a number of APIs

we'll want to use the GET statuses/sample API, which "Returns a small random sample of all public statuses."

## 12. Twitter has a number of APIs

If you wanted to access absolutely "All public statuses", you would need to use Twitter's Firehose API, which is not publicly available and would most likely cost you a pretty penny.

## 13. Tweets are returned as JSONs

One last point to note before we begin streaming tweets: tweets are returned to us as JSONs and they contain numerous possible fields. Check out the Twitter tweet field guide here. You can get tweet text, user, language, time of tweet,

## 14. Tweets are returned as JSONs

among many other fields. Let's now see how to access and stream data from the Twitter API! For first-time Python tweet-streamers, I usually recommend the package tweepy,

## 15. Using Tweepy: Authentication

which has a nice balance of usability and capability. Let's now use it to stream some tweets! First off, we need to create variables to store our access token and secret, plus our consumer key and secret.

## 16. Using Tweepy: stream tweets!!

To stream, we create an instance of tweepy's Stream class, passing our consumer key, consumer secret, access token, and access token secret. You can then stream tweets that contain keywords of choice by applying the filter method to the object stream! In the following exercises, you'll practice writing Python code to stream tweets and then you'll do some basic analysis of these tweets to see how often particular keywords are mentioned.

## 17. Let's practice!

Enjoy!

**Daily XP250**

##### Exercise

##### Exercise

# Streaming tweets

It's time to stream some tweets! Your task is to create the Streamobject and to filter tweets according to particular keywords. tweepy has been imported for you.

##### Instructions

**100 XP**

* Create your Stream object with the credentials given.
* Filter your Stream variable for the keywords "clinton", "trump", "sanders", and "cruz".
* # Store credentials in relevant variables
* consumer\_key = "nZ6EA0FxZ293SxGNg8g8aP0HM"
* consumer\_secret = "fJGEodwe3KiKUnsYJC3VRndj7jevVvXbK2D5EiJ2nehafRgA6i"
* access\_token = "1092294848-aHN7DcRP9B4VMTQIhwqOYiB14YkW92fFO8k8EPy"
* access\_token\_secret = "X4dHmhPfaksHcQ7SCbmZa2oYBBVSD2g8uIHXsp5CTaksx"
* # Create your Stream object with credentials
* stream = tweepy.Stream(\_\_\_\_, \_\_\_\_, \_\_\_\_, \_\_\_\_)
* # Filter your Stream variable
* stream.filter(\_\_\_\_)

# Import package import requests # Assign URL to variable: url url = 'https://en.wikipedia.org/w/api.php?action=query&prop=extracts&format=json&exintro=&titles=pizza' # Package the request, send the request and catch the response: r r = requests.get(url) # Decode the JSON data into a dictionary: json\_data json\_data = r.json() # Print the Wikipedia page extract pizza\_extract = json\_data['query']['pages']['24768']['extract'] print(pizza\_extract)

**Daily XP350**

##### Exercise

##### Exercise

# Load and explore your Twitter data

Now that you've got your Twitter data sitting locally in a text file, it's time to explore it! This is what you'll do in the next few interactive exercises. In this exercise, you'll read the Twitter data into a list: tweets\_data.

Be aware that this is real data from Twitter and as such there is always a risk that it may contain profanity or other offensive content (in this exercise, and any following exercises that also use real Twitter data).

##### Instructions

**100 XP**

* Assign the filename 'tweets.txt' to the variable tweets\_data\_path.
* Initialize tweets\_data as an empty list to store the tweets in.
* Within the for loop initiated by for line in tweets\_file:, load each tweet into a variable, tweet, using json.loads(), then append tweet to tweets\_data using the append() method.
* Hit submit and check out the keys of the first tweet dictionary printed to the shell.
* # Import package
* import json
* # String of path to file: tweets\_data\_path
* # Initialize empty list to store tweets: tweets\_data
* # Open connection to file
* tweets\_file = open(tweets\_data\_path, "r")
* # Read in tweets and store in list: tweets\_data
* for line in tweets\_file:
* \_\_\_\_
* \_\_\_\_
* # Close connection to file
* tweets\_file.close()
* # Print the keys of the first tweet dict
* print(tweets\_data[0].keys())

# Store credentials in relevant variables consumer\_key = "nZ6EA0FxZ293SxGNg8g8aP0HM" consumer\_secret = "fJGEodwe3KiKUnsYJC3VRndj7jevVvXbK2D5EiJ2nehafRgA6i" access\_token = "1092294848-aHN7DcRP9B4VMTQIhwqOYiB14YkW92fFO8k8EPy" access\_token\_secret = "X4dHmhPfaksHcQ7SCbmZa2oYBBVSD2g8uIHXsp5CTaksx" # Create your Stream object with credentials stream = tweepy.Stream(consumer\_key, consumer\_secret, access\_token, access\_token\_secret) # Filter your Stream variable stream.filter(["clinton", "trump", "sanders", "cruz"])

# Import package

import json

# String of path to file: tweets\_data\_path

tweets\_data\_path = 'tweets.txt'

# Initialize empty list to store tweets: tweets\_data

tweets\_data = []

# Open connection to file

tweets\_file = open(tweets\_data\_path, "r")

# Read in tweets and store in list: tweets\_data

for line in tweets\_file:

    tweet =json.loads(line)

    tweets\_data.append(tweet)

# Close connection to file

tweets\_file.close()

# Print the keys of the first tweet dict

print(tweets\_data[0].keys())

# Import package

import json

# String of path to file: tweets\_data\_path

tweets\_data\_path = 'tweets.txt'

# Initialize empty list to store tweets: tweets\_data

tweets\_data = []

# Open connection to file

tweets\_file = open(tweets\_data\_path, "r")

# Read in tweets and store in list: tweets\_data

for line in tweets\_file:

tweet =json.loads(line)

tweets\_data.append(tweet)

# Close connection to file

tweets\_file.close()

# Print the keys of the first tweet dict

print(tweets\_data[0].keys())

dict\_keys(['in\_reply\_to\_user\_id', 'created\_at', 'filter\_level', 'truncated', 'possibly\_sensitive', 'timestamp\_ms', 'user', 'text', 'extended\_entities', 'in\_reply\_to\_status\_id', 'entities', 'favorited', 'retweeted', 'is\_quote\_status', 'id', 'favorite\_count', 'retweeted\_status', 'in\_reply\_to\_status\_id\_str', 'in\_reply\_to\_user\_id\_str', 'id\_str', 'in\_reply\_to\_screen\_name', 'coordinates', 'lang', 'place', 'contributors', 'geo', 'retweet\_count', 'source'])

**Daily XP450**

##### Exercise

##### Exercise

# Twitter data to DataFrame

Now you have the Twitter data in a list of dictionaries, tweets\_data, where each dictionary corresponds to a single tweet. Next, you're going to extract the text and language of each tweet. The text in a tweet, t1, is stored as the value t1['text']; similarly, the language is stored in t1['lang']. Your task is to build a DataFrame in which each row is a tweet and the columns are 'text' and 'lang'.

##### Instructions

**100 XP**

* Use pd.DataFrame() to construct a DataFrame of tweet texts and languages; to do so, the first argument should be tweets\_data, a list of dictionaries. The second argument to pd.DataFrame() is a list of the keys you wish to have as columns. Assign the result of the pd.DataFrame() call to df.
* Print the head of the DataFrame.

# Import package

import pandas as pd

# Build DataFrame of tweet texts and languages

df = pd.DataFrame(tweets\_data, columns=('text','lang'))

# Print head of DataFrame

print(df.head())

# Import package

import pandas as pd

# Build DataFrame of tweet texts and languages

df = pd.DataFrame(tweets\_data, columns=('text','lang'))

# Print head of DataFrame

print(df.head())

text lang

0 b"RT @bpolitics: .@krollbondrating's Christoph... en

1 b'RT @HeidiAlpine: @dmartosko Cruz video found... en

2 b'Njihuni me Zonj\\xebn Trump !!! | Ekskluzive... et

3 b"Your an idiot she shouldn't have tried to gr... en

4 b'RT @AlanLohner: The anti-American D.C. elite... en

**Daily XP550**

##### Exercise

##### Exercise

# A little bit of Twitter text analysis

Now that you have your DataFrame of tweets set up, you're going to do a bit of text analysis to count how many tweets contain the words 'clinton', 'trump', 'sanders' and 'cruz'. In the pre-exercise code, we have defined the following function word\_in\_text(), which will tell you whether the first argument (a word) occurs within the 2nd argument (a tweet).

import re

def word\_in\_text(word, text):

word = word.lower()

text = text.lower()

match = re.search(word, text)

if match:

return True

return False

You're going to iterate over the rows of the DataFrame and calculate how many tweets contain each of our keywords! The list of objects for each candidate has been initialized to 0.

##### Instructions

**100 XP**

* Within the for loop for index, row in df.iterrows():, the code currently increases the value of clinton by 1 each time a tweet (text row) mentioning 'Clinton' is encountered; complete the code so that the same happens for trump, sanders and cruz.

# Initialize list to store tweet counts

[clinton, trump, sanders, cruz] = [0, 0, 0, 0]

# Iterate through df, counting the number of tweets in which

# each candidate is mentioned

for index, row in df.iterrows():

    clinton += word\_in\_text('clinton', row['text'])

    trump += word\_in\_text(\_\_\_\_, \_\_\_\_)

    sanders += word\_in\_text(\_\_\_\_, \_\_\_\_)

    cruz += word\_in\_text(\_\_\_\_, \_\_\_\_)

# Import package import pandas as pd # Build DataFrame of tweet texts and languages df = pd.DataFrame(tweets\_data, columns=('text','lang')) # Print head of DataFrame print(df.head())

# Initialize list to store tweet counts

[clinton, trump, sanders, cruz] = [0, 0, 0, 0]

# Iterate through df, counting the number of tweets in which

# each candidate is mentioned

for index, row in df.iterrows():

    clinton += word\_in\_text('clinton', row['text'])

    trump += word\_in\_text('trump', row['text'])

    sanders += word\_in\_text('sanders', row['text'])

    cruz += word\_in\_text('cruz', row['text'])

# Initialize list to store tweet counts [clinton, trump, sanders, cruz] = [0, 0, 0, 0] # Iterate through df, counting the number of tweets in which # each candidate is mentioned for index, row in df.iterrows(): clinton += word\_in\_text('clinton', row['text']) trump += word\_in\_text('trump', row['text']) sanders += word\_in\_text('sanders', row['text']) cruz += word\_in\_text('cruz', row['text'])

**Daily XP650**

##### Exercise

##### Exercise

# Plotting your Twitter data

Now that you have the number of tweets that each candidate was mentioned in, you can plot a bar chart of this data. You'll use the statistical data visualization library [seaborn](https://stanford.edu/~mwaskom/software/seaborn/), which you may not have seen before, but we'll guide you through. You'll first import seaborn as sns. You'll then construct a barplot of the data using sns.barplot, passing it two arguments:

1. a list of labels and
2. a list containing the variables you wish to plot (clinton, trump and so on.)

Hopefully, you'll see that Trump was unreasonably represented! We have already run the previous exercise solutions in your environment.

##### Instructions

**100 XP**

* Import both matplotlib.pyplot and seaborn using the aliases plt and sns, respectively.
* Complete the arguments of sns.barplot:
  + The first argument should be the list of labels to appear on the x-axis (created in the previous step).
  + The second argument should be a list of the variables you wish to plot, as produced in the previous exercise (i.e. a list containing clinton, trump, etc).

# Initialize list to store tweet counts [clinton, trump, sanders, cruz] = [0, 0, 0, 0] # Iterate through df, counting the number of tweets in which # each candidate is mentioned for index, row in df.iterrows(): clinton += word\_in\_text('clinton', row['text']) trump += word\_in\_text('trump', row['text']) sanders += word\_in\_text('sanders', row['text']) cruz += word\_in\_text('cruz', row['text'])

# Import packages

import matplotlib.pyplot as plt

import seaborn as sns

# Set seaborn style

sns.set(color\_codes=True)

# Create a list of labels:cd

cd = ['clinton', 'trump', 'sanders', 'cruz']

# Plot the bar chart

ax = sns.barplot(\_\_\_\_, \_\_\_\_)

ax.set(ylabel="count")

plt.show()

# Import packages

import matplotlib.pyplot as plt

import seaborn as sns

# Set seaborn style

sns.set(color\_codes=True)

# Create a list of labels:cd

cd = ['clinton', 'trump', 'sanders', 'cruz']

# Plot the bar chart

ax = sns.barplot(cd, [clinton, trump, sanders, cruz])

ax.set(ylabel="count")

plt.show()

# Import packages

import matplotlib.pyplot as plt

import seaborn as sns

# Set seaborn style

sns.set(color\_codes=True)

# Create a list of labels:cd

cd = ['clinton', 'trump', 'sanders', 'cruz']

# Plot the bar chart

ax = sns.barplot(cd, [clinton, trump, sanders, cruz])

ax.set(ylabel="count")

plt.show()

# Import packages import matplotlib.pyplot as plt import seaborn as sns # Set seaborn style sns.set(color\_codes=True) # Create a list of labels:cd cd = ['clinton', 'trump', 'sanders', 'cruz'] # Plot the bar chart ax = sns.barplot(cd, [clinton, trump, sanders, cruz]) ax.set(ylabel="count") plt.show()

## 1. Final Thoughts

Wowee congratulations! You've just completed your deep dive into the Twitter API, in which you streamed tweets, processed them and visualized your results. Amazing! If you've made it this far,

## 2. What you’ve learned:

you're now be able to import data in Python from a wide array of sources. To recap the skill-set and data importing chops that you have gained in this course AND its prequel, you're now adept at importing basic text files and flat files, the basic bread and butter of any working Data Scientist's professional life, local files in other formats, such as Excel spreadsheets, SAS, Stata and MATLAB files, pickled and HDF5 files. These skills will make you an even better collaborator for many working professionals out there. You've built up your basic skills at writing SQL queries and can now get all types of data out of relational databases. You're able to pull data from the web: not only can you import basic web files, but you know a number of ways to issue GET requests and can even do some basic web scraping and HTML parsing! On top of all of this, you've learnt how to pull data from APIs and have had hands-on experience doing so with dives into several examples. You've learned a great deal and

## 3. Let's practice!

successfully completed these two courses on Importing Data in Python. Happy importing!