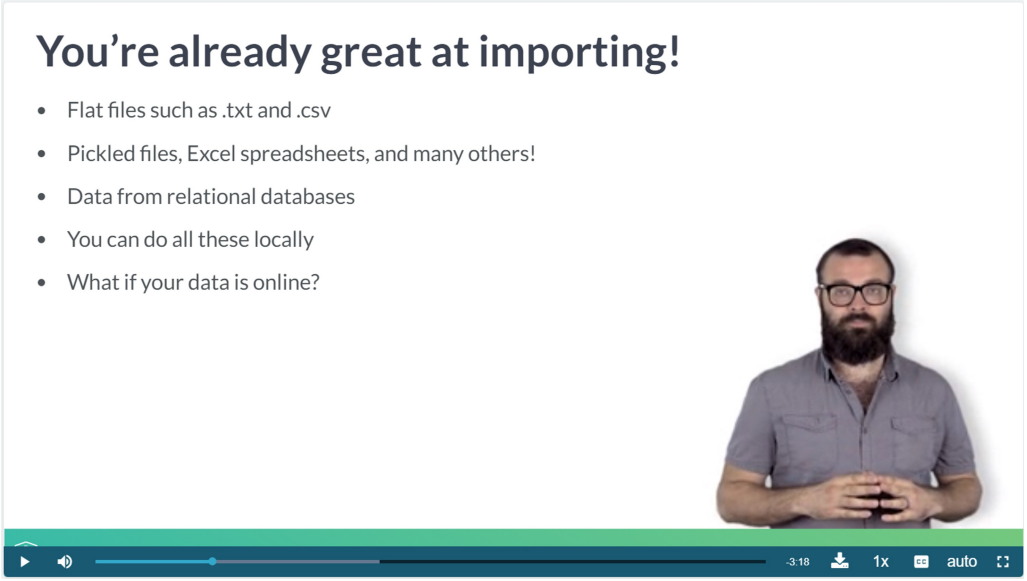
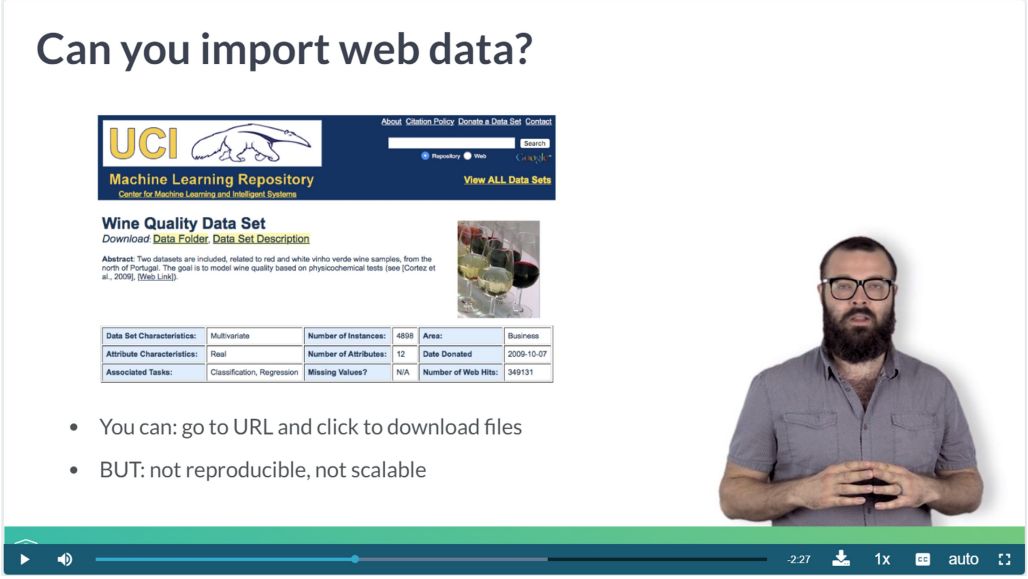
06. Intermediate Importing Data in Python

19 April 2020 13:08

1. Importing data from the Internet

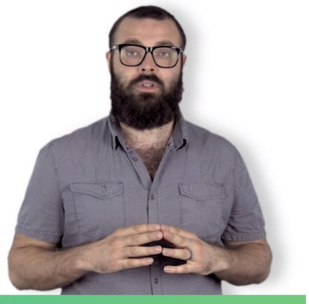
1.1 Importing flat files from the web





You'll learn how to...

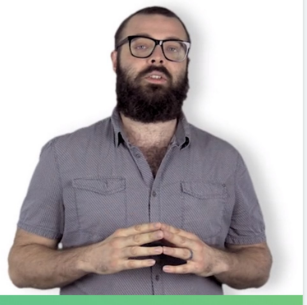
- Import and locally save datasets from the web
- Load datasets into pandas DataFrames
- Make HTTP requests (GET requests)
- Scrape web data such as HTML
- Parse HTML into useful data (BeautifulSoup)
- Use the urllib and requests packages



-1:23 1x auto

The urllib package

- Provides interface for fetching data across the web
- `urlopen()` - accepts URLs instead of file names

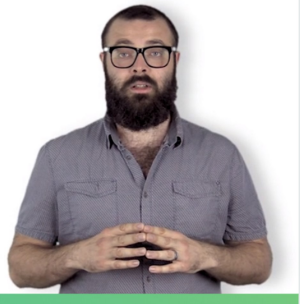


-0:46 1x auto

How to automate file download in Python

```
from urllib.request import urlretrieve
url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/
winequality-white.csv'
urlretrieve(url, 'winequality-white.csv')
```

```
('winequality-white.csv', <http.client.HTTPMessage at 0x103cf1128>)
```



-0:21 1x auto

1.2 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**. 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://s3.amazonaws.com/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:

- 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 urllib.request import urlretrieve

# Import pandas
import pandas as pd

# Assign url of file: url
url = 'https://s3.amazonaws.com/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())
```

1.3 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://s3.amazonaws.com/assets.datacamp.com/production/course_1606/datasets/winequality-red.csv'

Instructions:

- 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
url = 'https://s3.amazonaws.com/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
pd.DataFrame.hist(df.ix[:, 0:1])
plt.xlabel('fixed acidity (g(tartaric acid)/dm$^3$)')
plt.ylabel('count')
plt.show()
```

1.4 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

'http://s3.amazonaws.com/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:

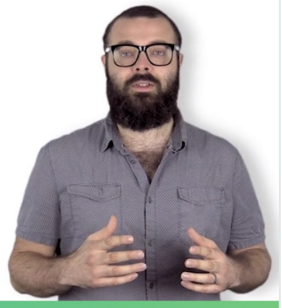
- Assign the URL of the file to the variable url.
- Read the file in url into a dictionary xls using pd.read_excel() recalling that, in order to import all sheets you need to pass None to the argument sheet_name.
- Print the names of the sheets in the Excel spreadsheet; these will be the keys of the 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
url = 'http://s3.amazonaws.com/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())
```

1.5 HTTP requests to import files from the web

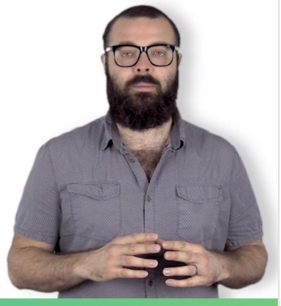
URL

- Uniform/Universal Resource Locator
- References to web resources
- Focus: web addresses
- Ingredients:
  - Protocol identifier - http:
  - Resource name - datacamp.com
- These specify web addresses uniquely



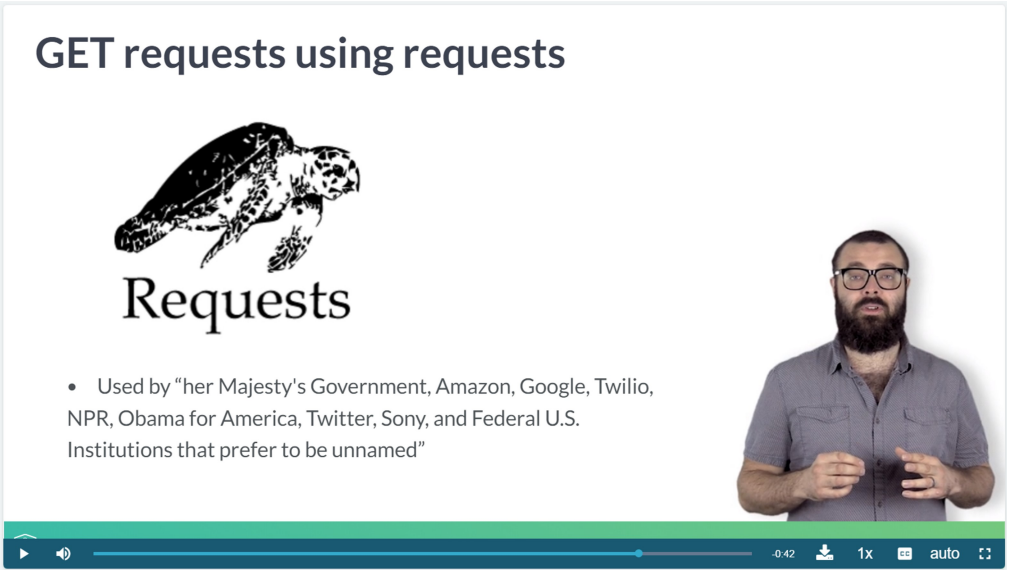
HTTP

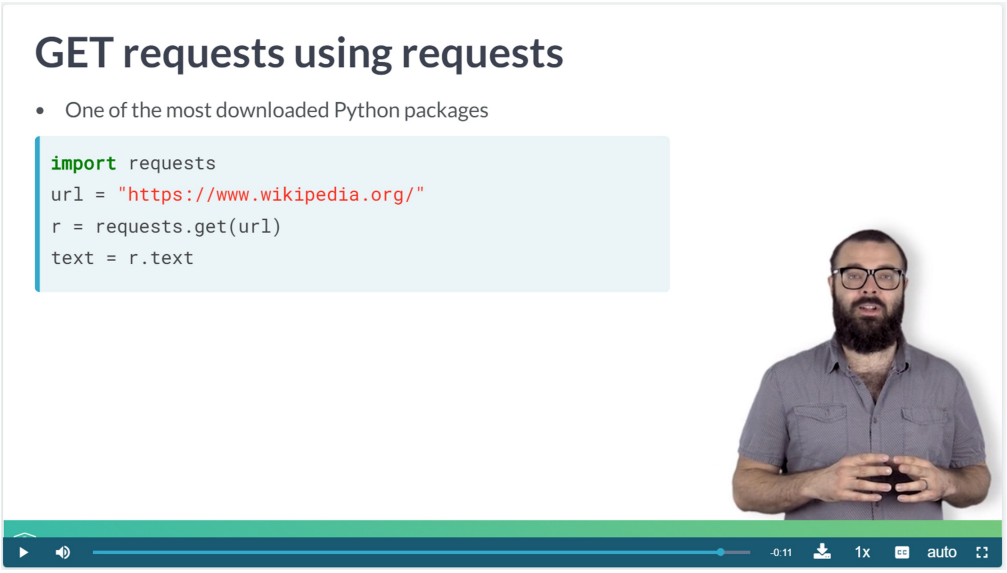
- HyperText Transfer Protocol
- Foundation of data communication for the web
- HTTPS - more secure form of HTTP
- Going to a website = sending HTTP request
  - GET request
- `urlretrieve()` performs a GET request
- HTML - HyperText Markup Language



GET requests using urllib

```
from urllib.request import urlopen, Request
url = "https://www.wikipedia.org/"
request = Request(url)
response = urlopen(request)
html = response.read()
response.close()
```





1.6 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 our teach page,

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

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:

- Import the functions urlopen and Request from the subpackage urllib.request.
- Package the request to the url "http://www.datacamp.com/teach/documentation" 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
from urllib.request import urlopen, Request

# Specify the url
```

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

# 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()
```

1.7 Printing HTTP request results in Python using urllib

You have just packaged and sent a GET request to "http://www.datacamp.com/teach/documentation" 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:

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

# 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()
```

1.8 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!

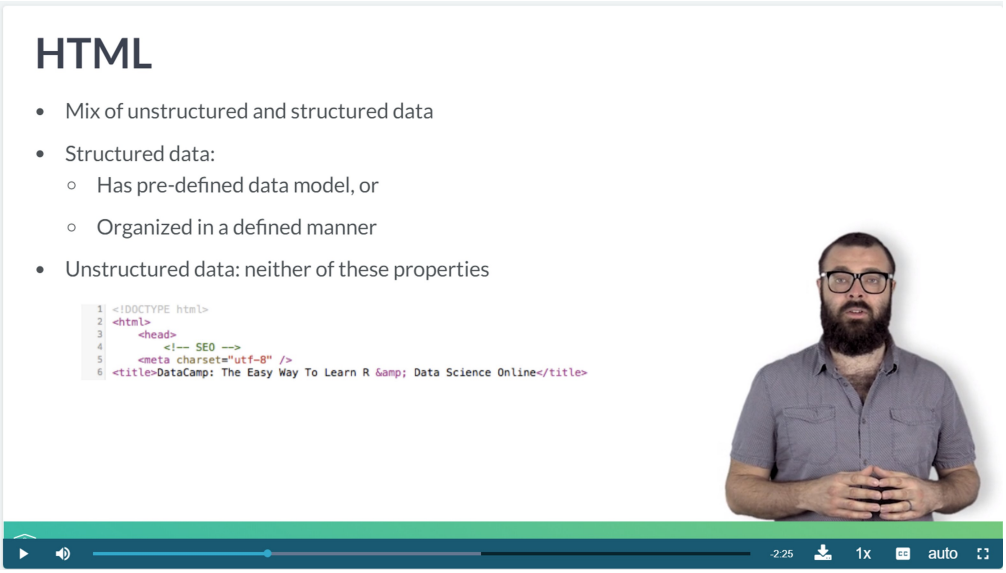
Instructions:

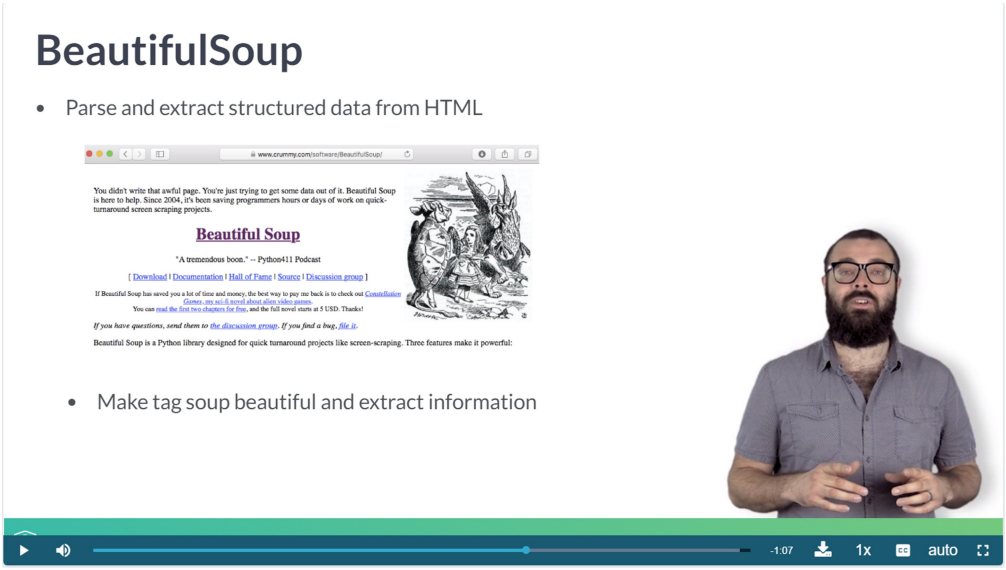
- Import the package requests.
- Assign the URL of interest to the variable url.

- 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
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)
```

1.9 Scraping the web in Python





BeautifulSoup

```
from bs4 import BeautifulSoup
import requests
url = 'https://www.crummy.com/software/BeautifulSoup/'
r = requests.get(url)
html_doc = r.text
soup = BeautifulSoup(html_doc)
```

-1:02 1x auto

Prettified Soup

```
print(soup.prettify())
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" "http://www.w3.org/TR/REC-html40/transitional.dtd">
<html>
 <head>
  <meta content="text/html; charset=utf-8" http-equiv="Content-Type"/>
  <title>
   Beautiful Soup: We called him Tortoise because he taught us.
  </title>
  <link href="mailto:leonardr@segfault.org" rev="made"/>
  <link href="/nb/themes/Default/nb.css" rel="stylesheet" type="text/css"/>
  <meta content="Beautiful Soup: a library designed for screen-scraping HTML and XML." name="Description"/>
  <meta content="Markov Approximation 1.4 (module: leonardr)" name="generator"/>
  <meta content="Leonard Richardson" name="author"/>
 </head>
 <body alink="red" bgcolor="white" link="blue" text="black" vlink="660066">
  <img align="right" src="10.1.jpg" width="250"/>
  <br/>
  <p>
```

-0:45 1x auto

Exploring BeautifulSoup

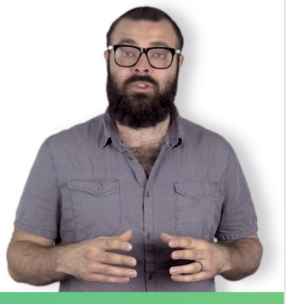
- Many methods such as:

```
print(soup.title)
```

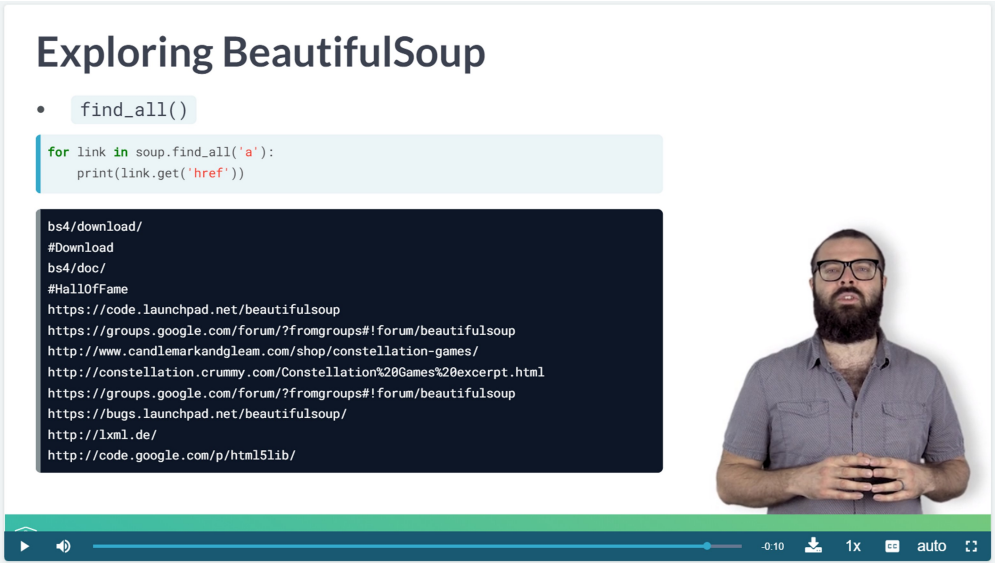
```
<title>Beautiful Soup: We called him Tortoise because he taught us.</title>
```

```
print(soup.get_text())
```

```
Beautiful Soup: We called him Tortoise because he taught us.
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.
```



-0:34 1x auto



1.10 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**. 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:

- 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 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)
```

1.11 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:

- 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
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.text

# Print Guido's text to the shell
print(guido_text)
```

1.12 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:

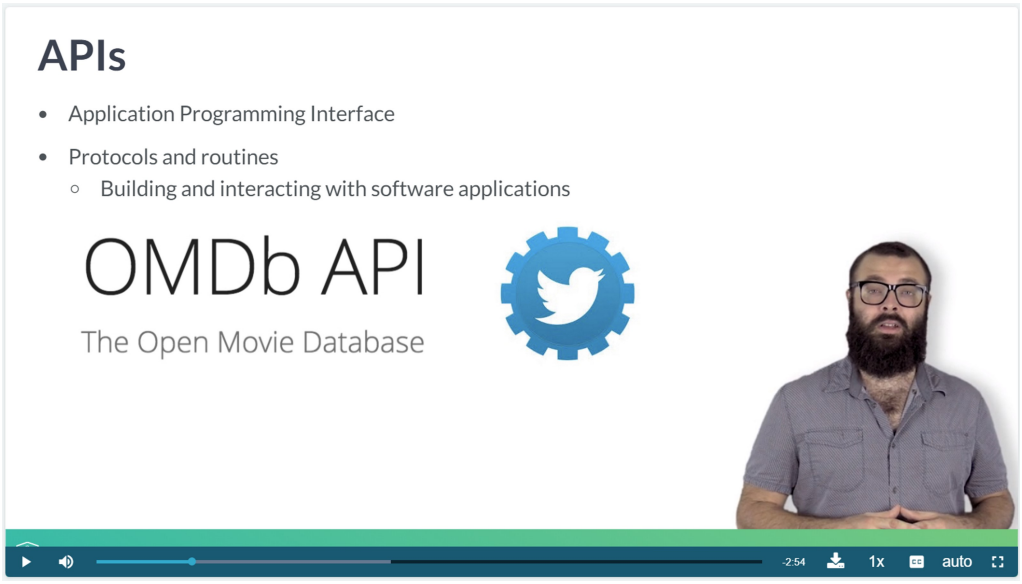
- Use the method find_all() to find all hyperlinks in soup, remembering that hyperlinks are defined by the HTML tag <a> but passed to find_all() without angle brackets; store the result in the variable a_tags.
- The variable a_tags is a results set: your job now is to enumerate over it, using a for loop 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').

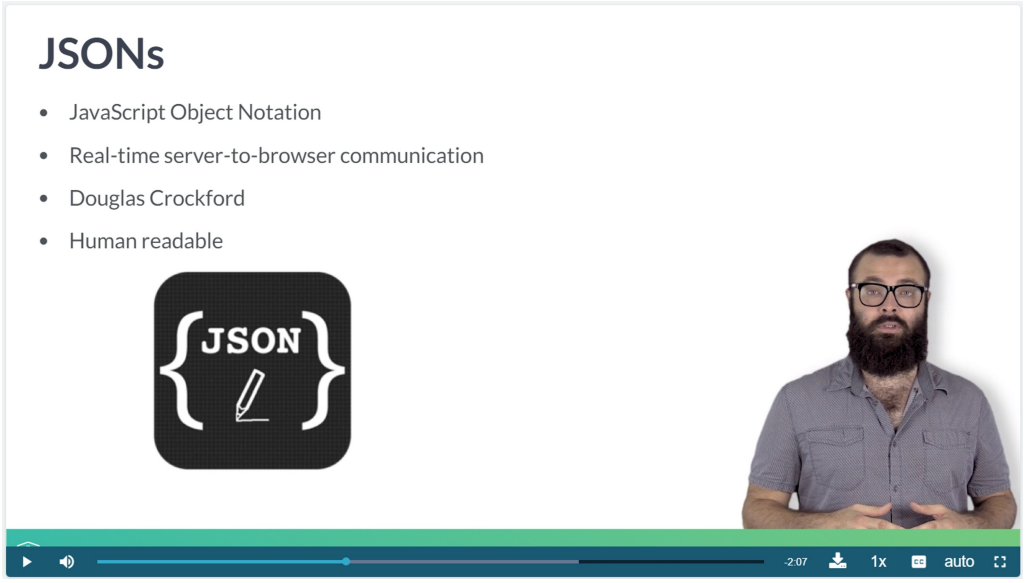
```
# Import packages
import requests
from bs4 import BeautifulSoup

# 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
a_tags = soup.find_all('a')

# Print the URLs to the shell
for link in a_tags:
    print(link.get('href'))
```

2. Interacting with APIs to import data from the web

2.1 Introduction to APIs and JSONs





JSONs

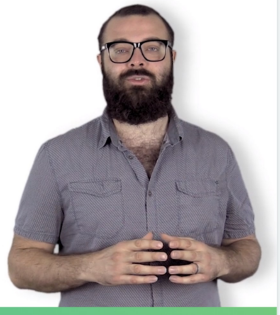
```
{'Actors': 'Samuel L. Jackson, Julianna Margulies, Nathan
Phillips, Rachel Blanchard',
 'Awards': '3 wins & 7 nominations.',
 'Country': 'Germany, USA, Canada',
 'Director': 'David R. Ellis',
 'Genre': 'Action, Adventure, Crime',
 'Language': 'English',
 'Rated': 'R',
 'Released': '18 Aug 2006',
 'Runtime': '105 min',
 'Title': 'Snakes on a Plane',
 'Type': 'movie',
 'Writer': 'John Heffernan (screenplay), Sebastian Gutierrez
(screenplay), David Dalessandro (story), John Heffernan (story)',
 'Year': '2006',
 'imdbID': 'tt0417148',
 'imdbRating': '5.6',
 'imdbVotes': '114,668'}
```

Loading JSONs in Python

```
import json
with open('snakes.json', 'r') as json_file:
        json_data = json.load(json_file)
```

```
type(json_data)
```

```
dict
```



Exploring JSONs in Python

```
for key, value in json_data.items():
    print(key + ':', value)
```

```
Title: Snakes on a Plane
Country: Germany, USA, Canada
Response: True
Language: English
Awards: 3 wins & 7 nominations.
Year: 2006
Actors: Samuel L. Jackson, Julianna Margulies
Runtime: 105 min
Genre: Action, Adventure, Crime
imdbID: tt0417148
Director: David R. Ellis
imdbRating: 5.6
Rated: R
Released: 18 Aug 2006
```

2.2 Pop quiz: What exactly is a JSON?

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

Possible Answers

- [ ] JSONs consist of key-value pairs.
- [ ] JSONs are human-readable.
- [ ] The JSON file format arose out of a growing need for real-time server-to-browser communication.
- [x] The function json.load() will load the JSON into Python as a list.
- [ ] The function json.load() will load the JSON into Python as a dictionary.

2.3 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:

- 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:
    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])
```

2.4 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?

Possible Answers

- [ ] The title is 'Kung Fu Panda' and the year is 2010.
- [ ] The title is 'Kung Fu Panda' and the year is 2008.
- [x] The title is 'The Social Network' and the year is 2010.
- [ ] The title is 'The Social Network' and the year is 2008.

2.5 APIs and interacting with the world wide web

Herein, you'll learn

- What APIs are
- Why APIs are important
- In the exercises:
  - Connecting to APIs
  - Pulling data from APIs
  - Parsing data from APIs

-3:50 1x auto

What is an API?

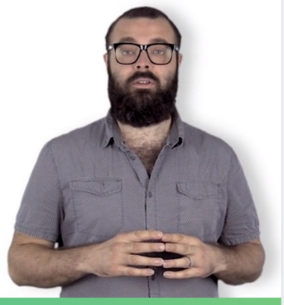
- Set of protocols and routines
- Bunch of code
  - Allows two software programs to communicate with each other

-3:12 1x auto

APIs are everywhere

UBER Developers

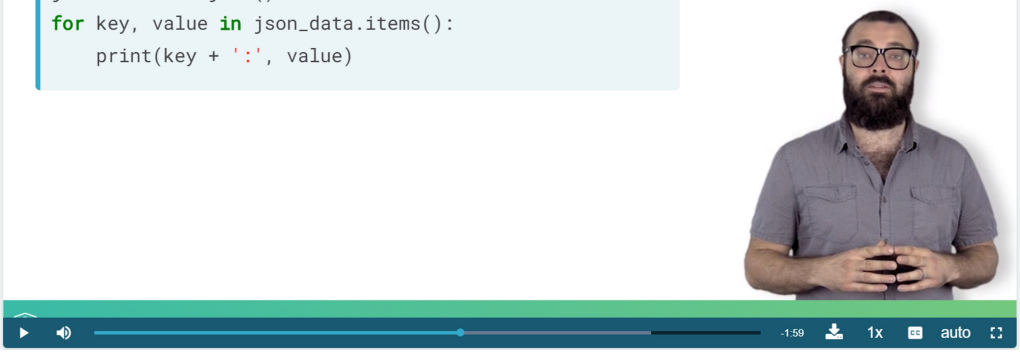
Instagram API



-2:49 1x auto

Connecting to an API in Python

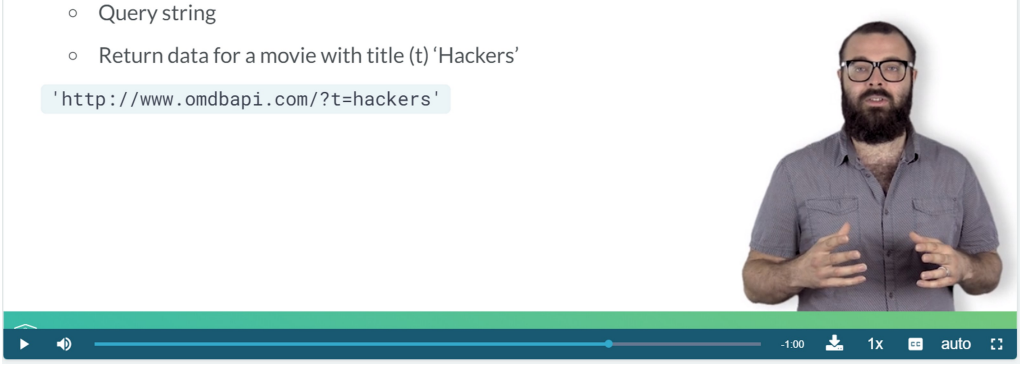
```
import requests
url = 'http://www.omdbapi.com/?t=hackers'
r = requests.get(url)
json_data = r.json()
for key, value in json_data.items():
    print(key + ':', value)
```



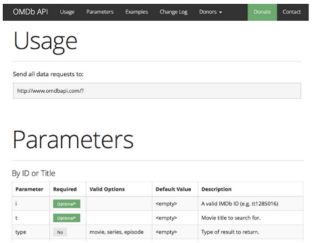
What was that URL?

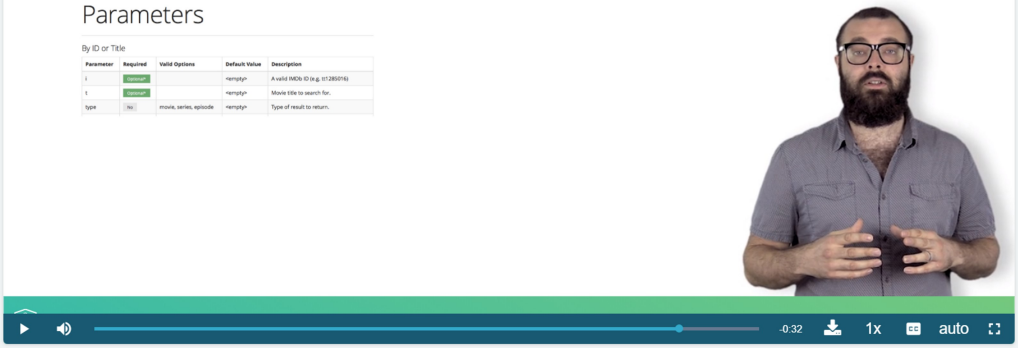
- http - making an HTTP request
- www.omdbapi.com - querying the OMDB API
- `?t=hackers`
  - Query string
  - Return data for a movie with title (t) 'Hackers'

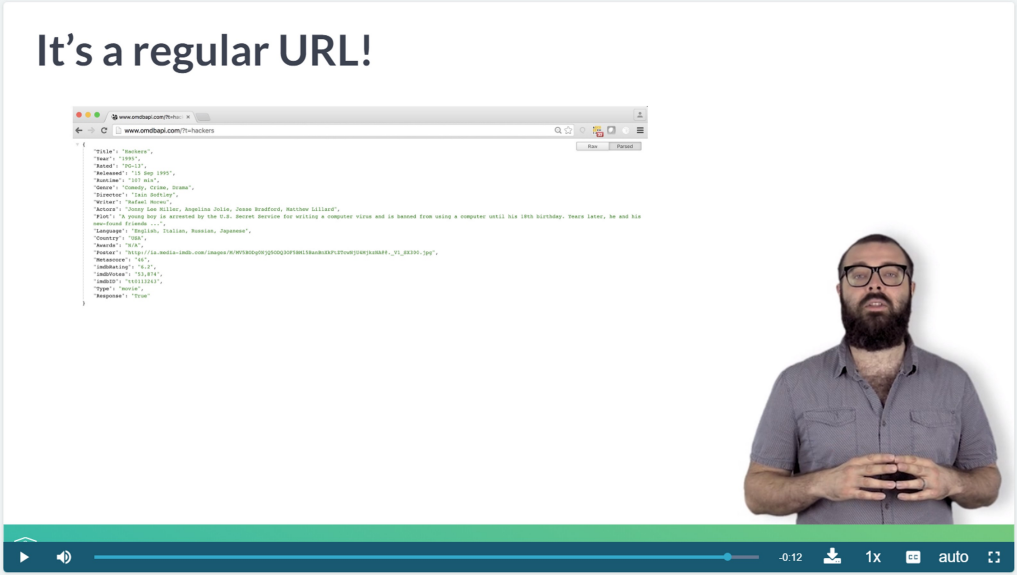
`'http://www.omdbapi.com/?t=hackers'`



OMDb API







2.6 Pop quiz: What's an API?

Which of the following statements about APIs is NOT true?

Possible Answers

- [ ] An API is a set of protocols and routines for building and interacting with software applications.
- [ ] API is an acronym and is short for Application Program interface.
- [ ] It is common to pull data from APIs in the JSON file format.
- [x] All APIs transmit data only in the JSON file format.
- [ ] An API is a bunch of code that allows two software programs to communicate with each other.

press

2.7 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:

- 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
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)
```

2.8 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:

- 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 Answer 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
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])
```

2.9 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**). 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

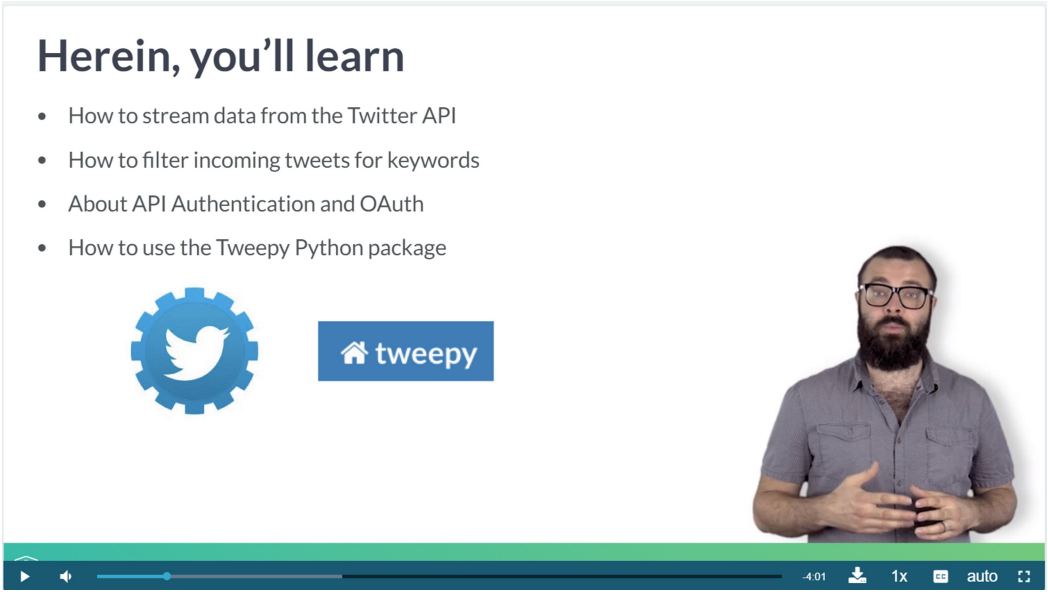
Instructions:

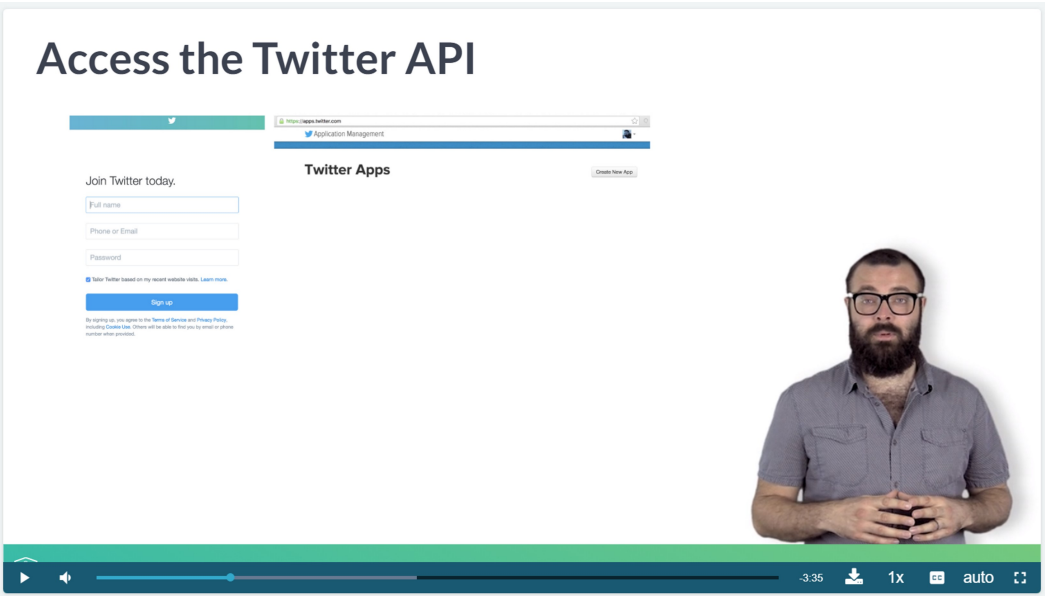
- Assign the relevant URL to the variable url.
- Apply the json() method to the response object r and store the resulting dictionary in the variable json_data.
- The variable pizza_extract holds the HTML of an extract from Wikipedia's *Pizza* page as a string; use the function print() to print this string to the shell.

```
# Import package
import requests

# Assign URL to variable: url
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)
```

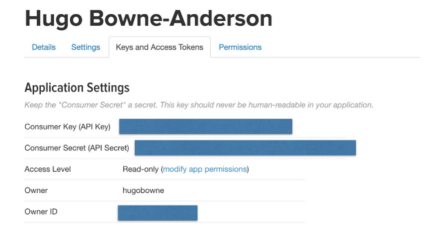
3. Diving deep into the Twitter API

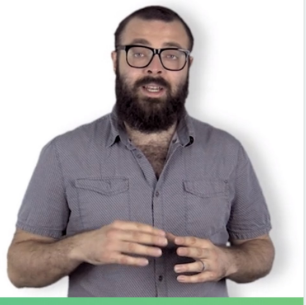
3.1 The Twitter API and Authentication





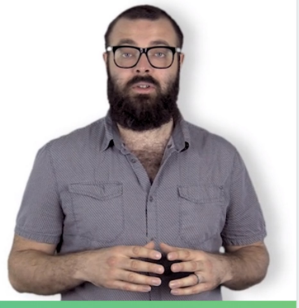
Access the Twitter API





-3:30 1x auto

Access the Twitter API



-3:02 1x auto

Twitter has a number of APIs

REST APIs

The REST APIs provide programmatic access to read and write Twitter data. Author a new Tweet, read author profile and follower data, and more. The REST API identifies Twitter applications and users using OAuth; responses are available in JSON.

If your intention is to monitor or process Tweets in real-time, consider using the Streaming API instead.

-2:38 1x auto

Twitter has a number of APIs

The Streaming APIs

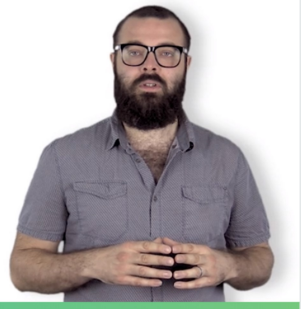
Overview

The Streaming APIs give developers low latency access to Twitter's global stream of Tweet data. A proper implementation of a streaming client will be pushed messages indicating Tweets and other events have occurred, without any of the overhead associated with polling a REST endpoint.

If your intention is to conduct singular searches, read user profile information, or post Tweets, consider using the REST APIs instead.

Twitter offers several streaming endpoints, each customized to certain use cases.

Public streams	Streams of the public data flowing through Twitter. Suitable for following specific users or topics, and data mining.
User streams	Single-user streams, containing roughly all of the data corresponding with a single user's view of Twitter.
Site streams	The multi-user version of user streams. Site streams are intended for servers which must connect to Twitter on behalf of many users.



-2:15 1x auto

Twitter has a number of APIs

GET statuses/sample

Returns a small random sample of all public statuses. The Tweets returned by the default access level are the same, so if two different clients connect to this endpoint, they will see the same Tweets.

Resource URL

`https://stream.twitter.com/1.1/statuses/sample.json`

-2:05 1x auto

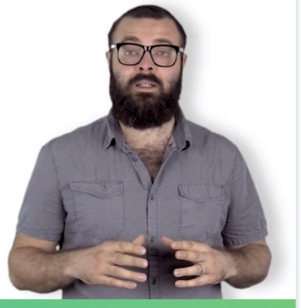
Twitter has a number of APIs

Firehose

API Reference Documents

Streaming

GET statuses/firehose	**This endpoint requires special permission to access.** Returns all public statuses. Few applications require this level of access. Creative use of a combination of other resources and various access levels can satisfy nearly every application use case.



-1:54 1x auto

Tweets are returned as JSONs

https://dev.twitter.com/overview/api/tweets

Field Guide

text	String	The actual UTF-8 text of the status update. See twitter-text for details on what is currently considered valid characters. Example: `"text":"Tweet Button, Follow Button, and Web Intents javascript now support SSL. http:\/\/t.co\/9fbAOoYy ^TS"`

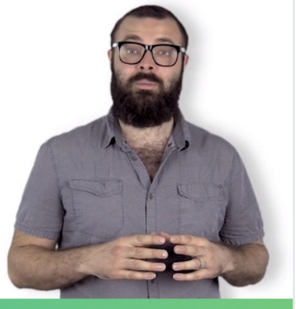
-1:35 1x auto

Tweets are returned as JSONs

https://dev.twitter.com/overview/api/tweets

Field Guide

lang	String	*Nullable.* When present, indicates a BCP 47 language identifier corresponding to the machine-detected language of the Tweet text, or "und" if no language could be detected. Example: `"lang": "en"`

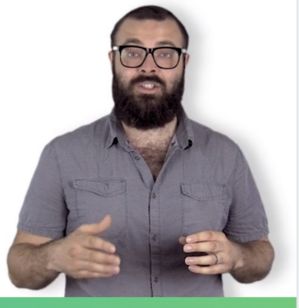


-1:26 1x auto

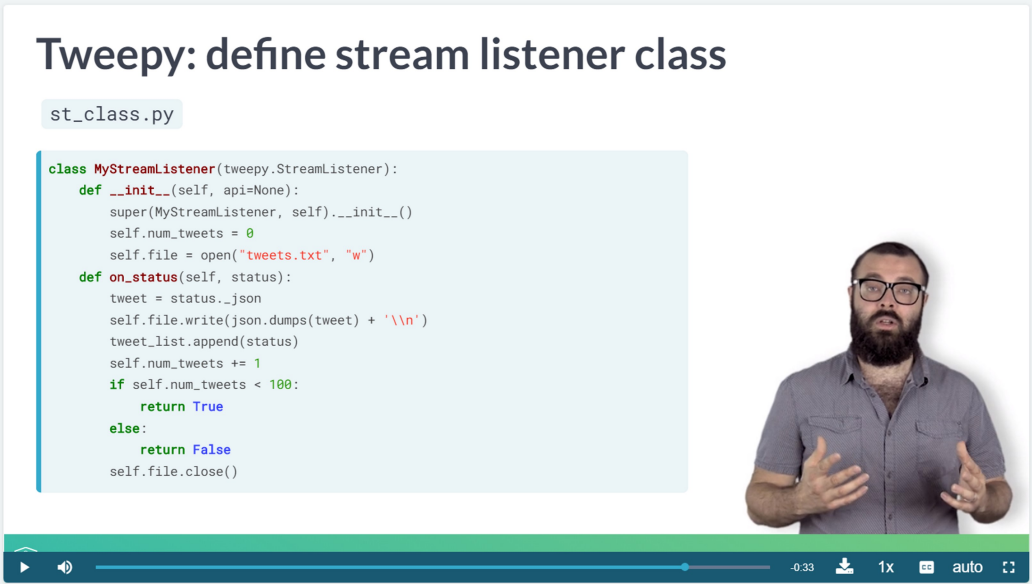
Using Tweepy: Authentication handler

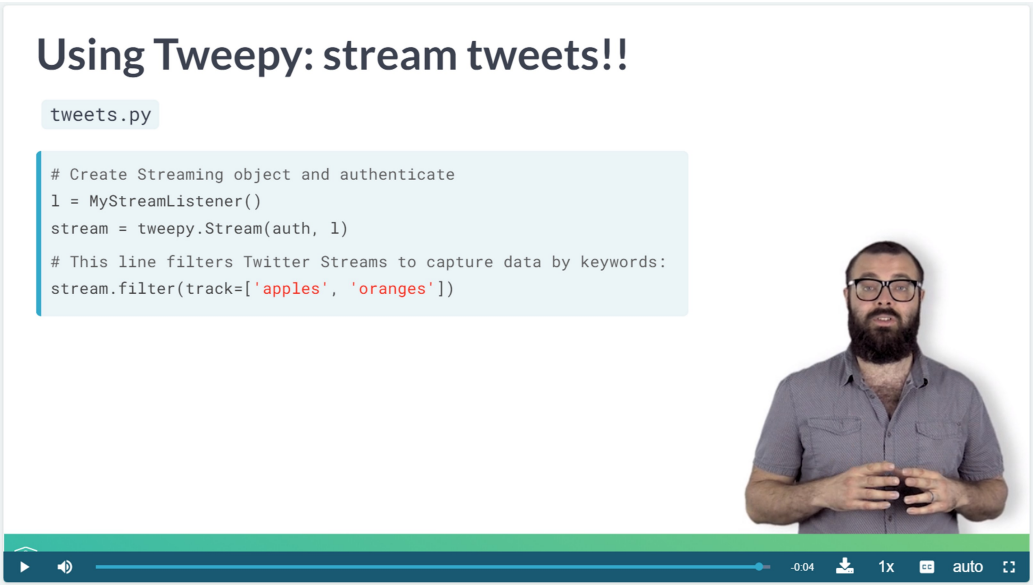
`tw_auth.py`

```
import tweepy, json

access_token = "..."
access_token_secret = "..."
consumer_key = "..."
consumer_secret = "..."

auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
```



-1:02 1x auto





3.2 API Authentication

The package tweepy is great at handling all the Twitter API OAuth Authentication details for you. All you need to do is pass it your authentication credentials. In this interactive exercise, we have created some mock authentication credentials (if you wanted to replicate this at home, you would need to create a **Twitter App** as Hugo detailed in the video). Your task is to pass these credentials to tweepy's OAuth handler.

Instructions:

- Import the package tweepy.
- Pass the parameters consumer_key and consumer_secret to the function tweepy.OAuthHandler().
- Complete the passing of OAuth credentials to the OAuth handler auth by applying to it the method set_access_token(), along with arguments access_token and access_token_secret.

```
# Import package
import tweepy

# Store OAuth authentication credentials in relevant variables
access_token = "1092294848-aHN7DcRP9B4VMTQIhwqOYiB14YkW92fFO8k8EPy"
access_token_secret = "X4dHmhPfaksHcQ7SCbmZa2oYBBVSD2g8uIHXsp5CTaksx"
consumer_key = "nZ6EA0FxZ293SxGNg8g8aP0HM"
consumer_secret = "fJGEodwe3KiKUnsYJC3VRndj7jevVvXbK2D5EiJ2nehafRgA6i"
```

```
# Pass OAuth details to tweepy's OAuth handler
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
```

3.3 Streaming tweets

Now that you have set up your authentication credentials, it is time to stream some tweets! We have already defined the tweet stream listener class, `MyStreamListener`, just as Hugo did in the introductory video. You can find the code for the tweet stream listener class **here**.

Your task is to create the `Stream`object and to filter tweets according to particular keywords.

Instructions:

- Create your `Stream` object with authentication by passing `tweepy.Stream()` the authentication handler `auth` and the Stream listener `l`;
- To filter Twitter streams, pass to the `track` argument in `stream.filter()` a list containing the desired keywords `'clinton'`, `'trump'`, `'sanders'`, and `'cruz'`.

```
# Initialize Stream listener
l = MyStreamListener()

# Create your Stream object with authentication
stream = tweepy.Stream(auth, l)

# Filter Twitter Streams to capture data by the keywords:
stream.filter(track = ['clinton', 'trump', 'sanders', 'cruz'])
```

3.4 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:

- 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
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())
```

3.5 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:

- 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())
```

3.6 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:

- 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('trump', row['text'])
    sanders += word_in_text('sanders', row['text'])
    cruz += word_in_text('cruz', row['text'])
```

3.7 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**, 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:

- 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).

```
# 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()
```

3.8 Final Thoughts