API and requests

2023-11-07

Agenda

- What is an API?
- The `requests` library
- Examples from English Wikipedia API

What is API (Application Programming Interface)

- A set of routines, tools, and standards that enable software applications to interact with each other
- In simple terms, API is an intermediary between two software applications

What is API (Application Programming Interface)

- It allows developers/researchers to access the functionality of a particular software application (without understanding the underlying structure)
- For many data science project, APIs are used for data collection

HTTP (Hypertext Transfer Protocol)

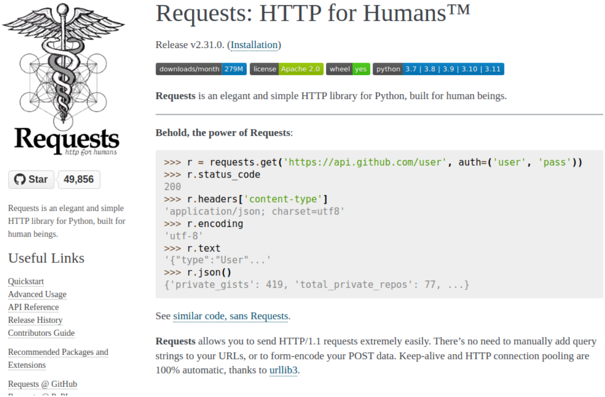
Protocol used to transfer data over the web

- “request”s
  - GET: read data
  - PATCH: update data
  - POST: create data
  - DELETE: delete data
- “response” is the result provided by API (mainly in JSON format)
- “key” is provided to users and used to check whether the request made from an authenticated one
- “endpoint” works as a front door for users and provides an access point to the server (or host)

Why API?

- Lots of data live on the internet
- Collecting data on scale:
    - Manual browsing/copying/pasting infeasible
    - Need to access programmatically

`requests` library



Requests is an elegant and simple HTTP library for Python, built for human beings.

Useful Links

Quickstart
Advanced Usage
API Reference
Release History
Contributors Guide

Recommended Packages and Extensions

Requests @ GitHub

Requests: HTTP for Humans™

Release v2.31.0. (Installation)

Requests is an elegant and simple HTTP library for Python, built for human beings.

Behold, the power of Requests:

```
>>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf8'
>>> r.encoding
'utf-8'
>>> r.text
'{"type":"User"...'
>>> r.json()
{'private_gists': 419, 'total_private_repos': 77, ...}
```

See similar code, sans Requests.

Requests allows you to send HTTP/1.1 requests extremely easily. There's no need to manually add query strings to your URLs, or to form-encode your POST data. Keep-alive and HTTP connection pooling are 100% automatic, thanks to urllib3.

Figure 1: requests library homepage

`requests` library

- One of the most popular libraries in Python
- Allows us to send a “request” to servers

requests library

- We now have a “Response” object

```python
import requests
r = requests.get('https://www.python.org/')
type(r)
```

```
requests.models.Response
```

`requests` library

```
r.status_code
```

```
200
```

- Status code 200 means success

`requests` library

- See Here for an exhaustive list of status codes
- Some of the most common status codes related to “GET”
  - 200 OK: Data retrieved successfully
  - 400 Bad Request: The server didn’t understand the request, possibly due to a missing or incorrect parameter
  - 401 Unauthorized: You might need authentication or your key might be wrong
  - 403 Forbidden: You don’t have permission to access the data
  - 404 Not Found: The endpoint or data you’re trying to access doesn’t exist
  - 429 Too Many Requests: You’ve hit a rate limit and need to slow down your requests

requests library

```
print(r.text) # print the html file
```

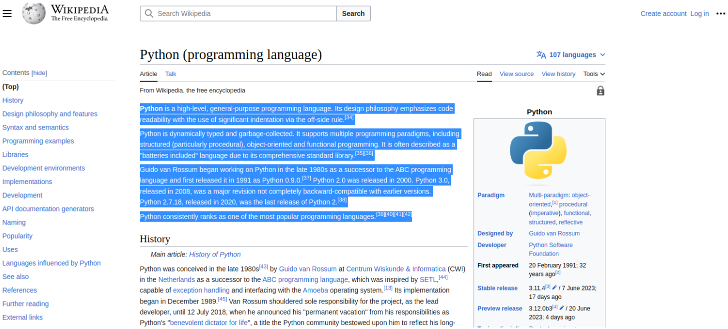
```
<!doctype html>
<!--[if lt IE 7]>   <html class="no-js ie6 lt-ie7 lt-ie8 lt-ie9">   <![endif]-->
<!--[if IE 7]>      <html class="no-js ie7 lt-ie8 lt-ie9">          <![endif]-->
<!--[if IE 8]>      <html class="no-js ie8 lt-ie9">                 <![endif]-->
<!--[if gt IE 8]><!--><html class="no-js" lang="en" dir="ltr">  <!--<![endif]-->

<head>
    <!-- Google tag (gtag.js) -->
    <script async src="https://www.googletagmanager.com/gtag/js?id=G-TF35YF9CVH"></script>
    <script>
      window.dataLayer = window.dataLayer || [];
      function gtag(){dataLayer.push(arguments);}
      gtag('js', new Date());
      gtag('config', 'G-TF35YF9CVH');
    </script>

    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <link rel="prefetch" href="//ajax.googleapis.com/ajax/libs/jquery/1.8.2/jquery.min.js">
    <link rel="prefetch" href="//ajax.googleapis.com/ajax/libs/jqueryui/1.12.1/jquery-ui.min.js">

    <meta name="application-name" content="Python.org">
    <meta name="msapplication-tooltip" content="The official home of the Python Programming Language">
    <meta name="apple-mobile-web-app-title" content="Python.org">
    <meta name="apple-mobile-web-app-capable" content="yes">
    <meta name="apple-mobile-web-app-status-bar-style" content="black">
```

HTML (HyperText Markup Language) documents

- Used for creating and structuring sections, paragraphs, and links on web pages
- Typically they contain a lot of information
- Not all of these are useful for a researcher

Web pages

- We need to extract part of the information that is useful
- E.g., no need for the Wikipedia logo, search bar, links to other languages, “Create Account”, or “Log In”



Python (programming language)

From Wikipedia, the free encyclopedia

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation via the off-side rule.[34]

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.[35][36]

Guido van Rossum began working on Python in the late 1980s as a successor to the ABC programming language and first released it in 1991 as Python 0.9.0.[37] Python 2.0 was released in 2000. Python 3.0, released in 2008, was a major revision not completely backward-compatible with earlier versions. Python 2.7.18, released in 2020, was the last release of Python 2.[38]

Python consistently ranks as one of the most popular programming languages.[39][40][41][42]

Python

Paradigm	Multi-paradigm: object-oriented,[1] procedural (imperative), functional, structured, reflective
Designed by	Guido van Rossum
Developer	Python Software Foundation
First appeared	20 February 1991; 32 years ago[2]
Stable release	3.11.4[3] / 7 June 2023; 17 days ago
Preview release	3.12.0b3[4] / 20 June 2023; 4 days ago

History

Main article: History of Python

Python was conceived in the late 1980s[43] by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC programming language, which was inspired by SETL,[44] capable of exception handling and interfacing with the Amoeba operating system.[13] Its implementation began in December 1989.[45] Van Rossum shouldered sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his "permanent vacation" from his responsibilities as Python's "benevolent dictator for life", a title the Python community bestowed upon him to reflect his long-

Figure 2: Wikipedia page for Python

Parsing HTML

- To parse an HTML document, we need a parser, a software that
  - Recognizes the structure of and HTML document
  - Allows the extraction of certain parts
- The `beautifulSoup` library serves that purpose
- We'll look into that in the next class

`requests` library

- Requests typically start with an `endpoint` defined by the server (or “host” as opposed to “clients”)
- English Wikipedia provides the `https://en.wikipedia.org/w/api.php` endpoint
- YouTube provides many endpoints, depending on what one is working with, e.g.:
  - `https://www.googleapis.com/youtube/v3/commentThreads`
  - `https://www.googleapis.com/youtube/v3/channels`

GET request

Parameters are specified in the following the format (note the `?` in the beginning)

- `?param1=value1&param2=value2&param3=value3...`

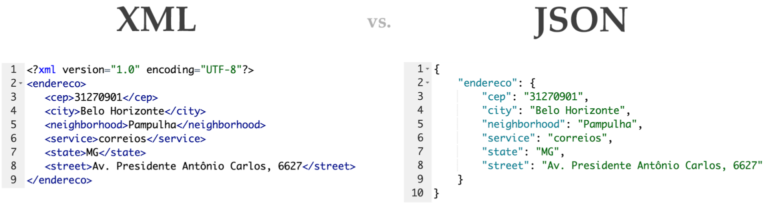
For example:

- `https://www.example.com/api/posts?query=america&sort=newest`
- `https://www.example.com/api/posts` is the endpoint
- Query to search for is “america”
- Results should be sorted from “newest”

GET request

- What the possible parameters are depend on the system
- Need to check documentation
- Many APIs will return data in JSON format, and sometimes XML also used
- Some will allow you to specify `&format=json` or `&format=xml` or something of this sort

JSON vs XML



```xml
<?xml version="1.0" encoding="UTF-8"?>
<endereco>
   <cep>31270901</cep>
   <city>Belo Horizonte</city>
   <neighborhood>Pampulha</neighborhood>
   <service>correios</service>
   <state>MG</state>
   <street>Av. Presidente Antônio Carlos, 6627</street>
</endereco>
```

```json
{
    "endereco": {
        "cep": "31270901",
        "city": "Belo Horizonte",
        "neighborhood": "Pampulha",
        "service": "correios",
        "state": "MG",
        "street": "Av. Presidente Antônio Carlos, 6627"
    }
}
```

Figure 3: JSON vs. XML (Source: Wikimedia Commons)

- Python's `json` and `xml` modules parse these types, but **`json`** much more common and easier

Example: Wikipedia

- English Wikipedia's `https://en.wikipedia.org/w/api.php` API endpoint
- We'll use it for demo purposes
- None of the parameters in this Wikipedia API are necessarily common across APIs
- Again, for any API, you need to check documentation
  - Which parameters are used
  - What are the possible values
  - Wether you need an API key

Example: Wikipedia

- After checking the documentation at `https://en.wikipedia.org/w/api.php`, we'll do the following
- Get information on the Wikipedia page `"Jimmy Carter"`
- Return in JSON format

Example: Wikipedia

- Specifically, get data on other language versions

```
endpoint = 'https://en.wikipedia.org/w/api.php'
req = endpoint + '?action=query&titles=Jimmy Carter&prop=langlinkscount&format=json'
r = requests.get(req)
```

Example: Wikipedia

- Check if request was successful

```
r.status_code
```

```
200
```

```
type(r)
```

```
requests.models.Response
```

Example: Wikipedia

- Returned text is a string in JSON format

```
type(r.text)
print(r.text)
```

```
{"batchcomplete":"","query":{"pages":{"15992":{"pageid":15992,"ns":0,"title":"J
```

Example: Wikipedia

- `json` module can parse this and turn into a dictionary

```
import json
d = json.loads(r.text)
type(d)

import pprint
pp = pprint.PrettyPrinter()
pp.pprint(d)
```

```
{'batchcomplete': '',
 'query': {'pages': {'15992': {'langlinkscount': 152,
                               'ns': 0,
                               'pageid': 15992,
                               'title': 'Jimmy Carter'}}}}
```

Example: Wikipedia

- Instead of creating a giant string to the request, we can pass them with a parameter dictionary

```python
endpoint = 'https://en.wikipedia.org/w/api.php'
parameters = {'action':'query',
              'titles':'Jimmy Carter',
              'prop':'langlinkscount',
              'format': 'json'}

r = requests.get(endpoint, params = parameters)
r.status_code
```

```
200
```

API get requests with the `requests`

- Also, instead of first getting a string and converting to a dictionary, we can simply do the following

```
d = r.json()
pp.pprint(d)
```

```
{'batchcomplete': '',
 'query': {'pages': {'15992': {'langlinkscount': 152,
                               'ns': 0,
                               'pageid': 15992,
                               'title': 'Jimmy Carter'}}}}
```

API get requests with the **requests**

- Many nested dictionaries

```
d['query']['pages']
```

```
{'15992': {'pageid': 15992,
  'ns': 0,
  'title': 'Jimmy Carter',
  'langlinkscount': 152}}
```

```
d['query']['pages']['15992']['langlinkscount']
```

```
152
```

API get requests with the **requests**

- Let's do another query
- Get daily pageviews associated with a given page

```
parameters = {'action':'query',
              'titles':'Jimmy Carter',
              'prop':'pageviews',
              'format': 'json'}

r = requests.get(endpoint, params = parameters)
r.status_code
```

```
200
```

API get requests with the **`requests`**

```
pp.pprint(r.json())
```

```
{'batchcomplete': '',
 'query': {'pages': {'15992': {'ns': 0,
                               'pageid': 15992,
                               'pageviews': {'2023-09-06': 13784,
                                             '2023-09-07': 15146,
                                             '2023-09-08': 15059,
                                             '2023-09-09': 14037,
                                             '2023-09-10': 14482,
                                             '2023-09-11': 14393,
                                             '2023-09-12': 13661,
                                             '2023-09-13': 13612,
                                             '2023-09-14': 18812,
                                             '2023-09-15': 24290,
                                             '2023-09-16': 21575,
                                             '2023-09-17': 18609,
                                             '2023-09-18': 15456,
                                             '2023-09-19': 13511,
                                             '2023-09-20': 14076,
                                             '2023-09-21': 19718,
                                             '2023-09-22': 22643,
                                             '2023-09-23': 23566,
                                             '2023-09-24': 36024,
                                             '2023-09-25': 25309,
                                             '2023-09-26': 18023,
                                             '2023-09-27': 16226,
                                             '2023-09-28': 19178,
                                             '2023-09-29': 23818,
                                             '2023-09-30': 40583,
```

API get requests with the `requests`

- Can work our way through the dict to the specific part we want

```
r.json()['query']['pages']['15992']['pageviews']
```

```
{'2023-09-06': 13784,
 '2023-09-07': 15146,
 '2023-09-08': 15059,
 '2023-09-09': 14037,
 '2023-09-10': 14482,
 '2023-09-11': 14393,
 '2023-09-12': 13661,
 '2023-09-13': 13612,
 '2023-09-14': 18812,
 '2023-09-15': 24290,
 '2023-09-16': 21575,
 '2023-09-17': 18609,
 '2023-09-18': 15456,
 '2023-09-19': 13511,
 '2023-09-20': 14076,
 '2023-09-21': 19718,
 '2023-09-22': 22643,
 '2023-09-23': 23566,
 '2023-09-24': 36024,
 '2023-09-25': 25309,
 '2023-09-26': 18023,
 '2023-09-27': 16226,
 '2023-09-28': 19178,
```

API get requests with the **requests**

- we saw that the Jimmy Carter page is available in 152 languages
- let's get information about what those languages are

```
parameters = {'action':'query',
              'titles':'Jimmy_Carter',
              'prop':'langlinks',
              'format': 'json'}

r = requests.get(endpoint, params = parameters)
r.status_code
```

```
200
```

API get requests with the `requests`

- but there aren't 152 languages here?

```
r.json()
```

```
{'continue': {'llcontinue': '15992|ay', 'continue': '||'},
 'query': {'normalized': [{'from': 'Jimmy_Carter', 'to': 'Jimmy Carter'}],
  'pages': {'15992': {'pageid': 15992,
    'ns': 0,
    'title': 'Jimmy Carter',
    'langlinks': [{'lang': 'ace', '*': 'Jimmy Carter'},
     {'lang': 'af', '*': 'Jimmy Carter'},
     {'lang': 'als', '*': 'Jimmy Carter'},
     {'lang': 'am', '*': '    '},
     {'lang': 'an', '*': 'Jimmy Carter'},
     {'lang': 'ang', '*': 'Iacobus Carter'},
     {'lang': 'ar', '*': '      '{,
     {'lang': 'ary', '*': '      '{,
     {'lang': 'arz', '*': '      '{,
     {'lang': 'ast', '*': 'Jimmy Carter'}]}}}}
```

API get requests with the `requests`

- Turns out, in this case, the API gave us only part of the information
- The documentation tells how to "continue"
- Pagination refers to a technique used in API design and development to retrieve large data sets in a structured and manageable manner
- When an API endpoint returns a large amount of data, pagination allows the data to be divided into smaller, more manageable chunks or pages

API get requests with the `requests`

- The previous output has a “continue” key
- We need to supply the key:value from there to the next query to continue

API get requests with the `requests`

```
parameters = {'action':'query',
              'titles':'Jimmy_Carter',
              'prop':'langlinks',
              'format': 'json',
              'llcontinue': '15992|ay'}

r = requests.get(endpoint, params = parameters)
r.status_code
```

```
200
```

API get requests with the **`requests`**

- We have new results, and more information on how to continue

```python
import pprint
pp = pprint.PrettyPrinter()
pp.pprint(r.json())
```

```
{'continue': {'continue': '||', 'llcontinue': '15992|bi'},
 'query': {'normalized': [{'from': 'Jimmy_Carter', 'to': 'Jimmy Carter'}],
           'pages': {'15992': {'langlinks': [{'*': 'Jimmy Carter',
                                              'lang': 'ay'},
                                             {'*': 'Cimmi Karter',
                                              'lang': 'az'},
                                             {'*': '      ', 'lang': 'azb'},
                                             {'*': 'Jimmy Carter',
                                              'lang': 'ban'},
                                             {'*': 'Jimmy Carter',
                                              'lang': 'bar'},
                                             {'*': 'Jimmy Carter',
                                              'lang': 'bat-smg'},
                                             {'*': 'Jimmy Carter',
                                              'lang': 'bcl'},
                                             {'*': '       ',
                                              'lang': 'be'},
                                             {'*': '          ',
                                              'lang': 'be-x-old'},
                                             {'*': '       ',
                                              'lang': 'bg'}],
                               'ns': 0,
                               'pageid': 15992,
                               'title': 'Jimmy Carter'}}}}
```

API get requests with the **`requests`**

- Many other requests are possible—need to check documentation
- For example, this API allows combining titles with the | sign
- Some others might ask you to combine with ,

```
parameters = {'action':'query',
              'titles':'Jimmy_Carter|George_H._W._Bush',
              'prop':'langlinkscount',
              'format': 'json'}

r = requests.get(endpoint, params = parameters)
r.status_code
```

```
200
```

API get requests with the **`requests`**

```
r.json()
```

```
{'batchcomplete': '',
 'query': {'normalized': [{'from': 'Jimmy_Carter', 'to': 'Jimmy Carter'},
   {'from': 'George_H._W._Bush', 'to': 'George H. W. Bush'}],
  'pages': {'11955': {'pageid': 11955,
    'ns': 0,
    'title': 'George H. W. Bush',
    'langlinkscount': 154},
   '15992': {'pageid': 15992,
    'ns': 0,
    'title': 'Jimmy Carter',
    'langlinkscount': 152}}}}
```

API get requests with the `requests`

- Get the content of an Wikipedia page in its entirety

```
parameters = {'action':'parse',
              'page':'KAIST',
              'format': 'json'}

r = requests.get(endpoint, params = parameters)
r.status_code
```

```
200
```

A wide range of APIs

- Many public APIs you can use, test etc: https://github.com/public-apis/public-apis
- Many, many others!!