Lecture #9: JSON and HTML

Presented by Jamal Bouajjaj

2023–10–02

For University of New Haven's Fall 2023 CSCIxx51 Course

Copyright (C) 2023 Jamal Bouajjaj under GPLv3

JSON

JSON File

JSON is an open format designed for computer data interchange, but also is human readable. It's very populate on the Web to exchange data, and Javascript easily eats it up.

JSON Example

Here is an example of a JSON file/data set:

```
{
  "first_name": "John",
  "last_name": "Smith",
  "is_alive": true,
  "age": 27,
  "address": {
    "street_address": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postal_code": "10021-3100"
  },
  "phone_numbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [
    "Catherine",
    "Thomas",
    "Trevor"
  ],
  "spouse": null
}
```

Wait...key value?

Wait a second, did you just say key-value pairs?

Wait...key value?

Wait a second, did you just say key-value pairs?

Isn't that same same as a Python dictionary?

Wait...key value?

Wait a second, did you just say key-value pairs?

Isn't that same same as a Python dictionary?

To that I say...YES! A JSON file can be easily imported and save from/to a Python dictionary, making them quite easy to play with.

JSON Module

The Python standard library has a nice module called *json* that nicely loads and saves JSON files.

JSON Module Example

See lecture9-json.py

HTTP

HTTP

HTTP is a an abstraction layer protocol (highest level on OSI level) for distributed, collaborative, hypertext information systems[1].

Hypertext is simply data that can refer to another data. For example, an HTML document that refers to another CSS file for styling.

This protocol is what the World-Wide-Web is built upon.

HTTP is a request-response protocol with a client-server model. The protocol is also stateless.[2].

[1] RFC9110, `https://datatracker.ietf.org/doc/html/rfc9110`
[2] `https://en.wikipedia.org/wiki/HTTP`

Request Methods

HTTP has many possible request methods a client can make.

HTTP has many request methods, but two are the most common:

- **GET**: Request an info from the server. This should have no other effect other than getting the data.
- **POST**: Mostly for sending some information to a server, for example to post something.

Return Code

Per transaction, the server returns a return code as a 3 digit number. Here are what the 3rd digit of the code means, and some common example

- **1xx**: Information Response
- **2xx**: Sucess
  - **200**: OK
- **3xx**: Redirection
- **4xx**: Client Error
  - **404**: Not Found
- **5xx**: Server Error

HTTP Message

An HTTP message body has the following information[3]:

- **Request Line** (GET /logo.gif HTTP/1.1) or **Status Line** (HTTP/1.1 200 OK)
- **Headers**
- **Empty Line**
- **Message Body Data**

[3] https://en.wikipedia.org/wiki/HTTP_message_body

SECURITY

HTTP is **NOT SECURE!!!!!!!!**. DO NOT TRY and send sensitive data across HTTP alone.I'll be demonstrating this in Wireshark

SECURITY

HTTP is **NOT SECURE!!!!!!!!**. DO NOT TRY and send sensitive data across HTTP alone.I'll be demonstrating this in Wireshark

If you want secure content, use HTTPS, which encrypts your data over TLS. This is shown in your browser by the padlock icon, and almost all websites have HTTPS implemented.

SECURITY

HTTP is **NOT SECURE!!!!!!!!**. DO NOT TRY and send sensitive data across HTTP alone.I'll be demonstrating this in Wireshark

If you want secure content, use HTTPS, which encrypts your data over TLS. This is shown in your browser by the padlock icon, and almost all websites have HTTPS implemented.

Side note: To all of the VPN ads that state they will "encrypt your data to prevent hackers", this isn't correct for the most part. Your data send to and from the server is already encrypted with "military grade encryption" (i.e AES)

HTTP example

Let's try to get an HTTP request. I will be using the tool *curl*, which is available on Linux, FreeBSD, and MacOS. On Windows you will have to get it (good luck without a package manager!)

API

An *API* is simply an interface designed for computers to talk to each other. It is simply a specification.

A *Web API* is an API that uses HTTP as the transfer protocol. It is the most common usage of the term API.

The most popular type of return content type for an API is JSON or XML.

API

An *API* is simply an interface designed for computers to talk to each other. It is simply a specification.

A *Web API* is an API that uses HTTP as the transfer protocol. It is the most common usage of the term API.

The most popular type of return content type for an API is JSON or XML.JSON, so...

API

An *API* is simply an interface designed for computers to talk to each other. It is simply a specification.

A *Web API* is an API that uses HTTP as the transfer protocol. It is the most common usage of the term API.

The most popular type of return content type for an API is JSON or XML.JSON, so...YES, Python makes interfacing with web APIs and parsing thru the data quite easily. XML too, but I don't cover that today as it's the lesser popular.

HTTP in Python

There are two main modules used to simplify requesting HTTP content in Python:

- urllib.request (Standard Library)
- requests

Both handle re-directs for you (from a 301 status code for example). They also handle HTTPS for you!

URL Lib Example

```
import urllib.request
with urllib.request.urlopen("https://semver.org") as u:
  print("->", u.status)
  print("->", u.reason)
  print("->", u.url)
  print("->", u.getheaders())
  print("->", u.read())
```

GET with Data

```
url =
 ↪  "https://api.weather.gov/gridpoints/OKX/65,67/forecast"
u = urllib.request.urlopen(url)
print("->", u.status)
print("->", u.reason)
print("->", u.url)
print("->", u.getheaders())
print("->", u.read())
```

GET with Data

```
url = "https://opentdb.com/api.php"
data = {'amount': 10}
url = url + '?' + urllib.parse.urlencode(data)
u = urllib.request.urlopen(url)
print("->", u.status)
print("->", u.reason)
print("->", u.url)
print("->", u.getheaders())
print("->", u.read())
```

POST with Data

```
url = "https://reqres.in/api/users"
data = {'name': 'morpheus', 'job': 'leader'}
header = {'User-Agent' : 'Mozilla/5.0'} # due to it
↪  rejecting otherwise
data = urllib.parse.urlencode(data).encode()
url = urllib.request.Request(url, data, header)
u = urllib.request.urlopen(url)
print("->", u.status)
print("->", u.reason)
print("->", u.url)
print("->", u.getheaders())
#print("->", u.read())
data = json.load(u)
print(data)
```

requests

requests is an non-standard Python module (so must be installed) that makes HTTP communication a little easier.

requests example

```
import requests
u = requests.get("https://semver.org")
print("->", u.status_code)
print("->", u.reason)
print("->", u.url)
print("->", u.headers)
print("->", u.text)
```

GET with Data

```
url = "https://opentdb.com/api.php"
data = {'amount': 10}
u = requests.get(url, data)
print("->", u.status_code)
print("->", u.reason)
print("->", u.url)
print("->", u.headers)
# print("->", u.text)
print("->", u.json())
```

POST with Data

```
url = "https://reqres.in/api/users"
data = {'name': 'morpheus', 'job': 'leader'}
header = {'User-Agent' : 'Mozilla/5.0'} # due to it
↪  rejecting otherwise
u = requests.post(url, data, headers=header)
print("->", u.status_code)
print("->", u.reason)
print("->", u.url)
print("->", u.headers)
# print("->", u.text)
print("->", u.json())
```

End

The end