# Programming Fundamentals with Python. HTTP 2

Pepe García

2020-04-20

# Programming Fundamentals with Python. HTTP 2

# Plan for today

- Learn the very basics of HTML
- How to do web scraping
- Build a Wikipedia Crawler!

# HTML

HTML is the **markup language** that powers the web.

It's used to declare the structure and content of web pages.

# HTML

HTML documents are created using HTML tags, enclosed between **angle brackets.**

The first word that appears inside a tag is the **tag name** and it identifies the type of element

```
<img>
<br>
```

# HTML

There are tags that also declare **attributes** inside them

```
<img src="logo.png">
```

# HTML

There are also tags that need **opening** and **closing** tag. They can contain anything, from text directly, to other tags.

```
<p>
this is text inside a paragraph
</p>

<p>
this paragraph has a
<a href="http://google.com">link to google</a> inside.
</p>
```

# some HTML elements

```
<a href="url">links</a>

<img src="url">

<p>paragraphs</p>

<h1>titles</h1>

<h2>more titles</h2>
```

# Practice

Let's inspect the wiki page of the Platypus and identify different HTML tags.

https://en.wikipedia.org/wiki/Platypus

Change some part of the HTML to see how it changes.

# Two special attributes

Most attributes in HTML are specific to tags. For example **href** is only for **a** tags.

There are two attributes though, that are very useful for crawling: **class** and **id**.

# Attributes: **id**

```
<a id="logo">logo</a>

<div id="footer"></div>
```

Id is used to mark specific parts of the web page with a **unique** identifier. The developer is free to use whatever identifier they want.

# Attributes: **class**

```
<p>
  <a href="http://dundermifflin.com/about-us" class="menu-item">
    About us
  </a>
  <a href="http://dundermifflin.com/careers" class="menu-item">
    Careers
  </a>
  <a href="http://dundermifflin.com/sitemap" class="menu-item">
    Sitemap
  </a>
</p>
```

**class** is used to mark sets of HTML tags that will have a similar meaning.

# Practice

Let's inspect the wiki page of the Platypus and see the use of different **classes** and **ids.**

https://en.wikipedia.org/wiki/Platypus

# Web scrapping

Scrapping is used for extracting data from websites. The most famous **web crawlers** are search engine bots, such as **GoogleBot.**

The idea behind web scrapping is to retrieve the web pages using a **HTTP client**, and then get the information we need from the HTML.

# HTTP Client

As HTTP client we will use **requests**

```
requests.get('https://wikipedia.com/wiki/Echidna')
```

# Dealing with HTML

For dealing with HTML we will use the **BeautifulSoup** library

```
from bs4 import BeautifulSoup
```

# Dealing with HTML

We will start by passing the HTML to our **BeautifulSoup** class:

```
import requests
from bs4 import BeautifulSoup

response = requests.get('https://wikipedia.com')

soup = BeautifulSoup(response.text)
```

# Extracting data from HTML

We will use the **select()** method on **BeautifulSoup** to extract data from the document.

```
import requests
from bs4 import BeautifulSoup

response = requests.get('https://wikipedia.com')

soup = BeautifulSoup(response.text)

links = soup.select('a')
```

We can, for example, get all links in it:

# Extracting data from HTML

We can get all HTML elements **containing a class** using a **dot** before the classname

```
import requests
from bs4 import BeautifulSoup

response = requests.get('https://wikipedia.com')

soup = BeautifulSoup(response.text)

links = soup.select('.menu-item')
```

# Extracting data from HTML

We can select the HTML tag **containing an id** using a **hash** sign before the id

```
import requests
from bs4 import BeautifulSoup

response = requests.get('https://wikipedia.com')

soup = BeautifulSoup(response.text)

links = soup.select('#menu')
```

# Practice

Crawl a Wiki page and see how to get:

- elements via **classname**
- elements via their **id**
- elements via their **HTML tag**

# Exercise time!

In this exercise we'll build a small crawler for Wikipedia.

1. Create a function **get_page** that receives a URL from wikipedia and returns the **parsed** HTML

2. Create a function **crawl** that crawls links from the previous Wikipedia page. (limit it to 5 pages)

# Exercise 1

```
def get_page(page):
    print("crawling " + page)
    response = requests.get(base_url + page)
    soup = BeautifulSoup(response.text, "html.parser")
    return soup
```

# Exercise 2

```
index = {}

def crawl():
    queue = ["/wiki/Echidna"]

    while queue and len(index) < 10:
        current = queue.pop(0)
        soup = get_page(current)
        index[current] = len(soup.select('#bodyContent')[0].get_text().spli
        links = soup.select('a')

        for link in links:
            if 'href' in link.attrs and link['href'].startswith('/wiki'):
                queue.append(link['href'])
```

# HTTP Methods

Depending on the intention of the request, HTTP describes different methods:

| method | **intention** |
|---|---|
| **GET** | access to a resource |
| **POST** | update a resource |
| **PUT** | create a resource |
| **DELETE** | delete a resource |

# HTTP Methods

Methods are part of the **request**. We have already seen how to use the **GET** method with **requests**:

```
import requests


requests.get("http://resource")
```

# HTTP Methods

The other methods can be used in the same way with **requests**:

```
import requests

requests.get("http://resource")
requests.put("http://resource")
requests.post("http://resource")
requests.delete("http://resource")
```

# HTTP servers

HTTP servers answer to requests from clients. We will be using the **flask** library for creating HTTP servers in Python.

HTTP servers handle **routes** in different ways.

# HTTP servers

In order to run a flask web server we'll just need to instantiate the **Flask** class and then call the **run()** method on it:

```
from flask import Flask

server = Flask('my first server')

server.run()
```

# HTTP servers

Trying our server out!

Open a web browser and go to **http://localhost:5000**

You should see a **Not Found** message

# HTTP servers

The reason why we're getting a **Not Found** is because we're not adding any **routes** to our server.

# HTTP servers

```
from flask import Flask

app = Flask("simplest server")

@app.route("/")
def hello():
    return "hello from the web!"

app.run()
```

# HTTP servers

Trying our server out!

Open a web browser and go to **http://localhost:5000**

# Connecting to local HTTP servers

We can use the requests library to connect to local HTTP servers!

```
import requests

response = requests.get("http://localhost:8080/hello")

print(response.text)
```

# Connecting to local HTTP servers

Connecting to a local HTTP server using **requests**

# HTTP routes

Our flask server can handle different routes by adding more handlers to it:

```
@app.route("/hello")
def hello():
    return "hi!"

@app.route("/goodbye")
def hello():
    return "bye!"
```

# HTTP routes

We can also capture part of the path as a variable:

```
@app.route("/hello/<name>")
def hello(name):
    return "hello " + name
```

# HTTP routes

Implement a server that receives name and last name from the URL and greets the user.

# Exercise

Create a HTTP server that can do simple arithmetic tasks.

- **GET /sum/1/2**

returns 3

- **GET /multiply/3/4**

returns 12

# HTTP methods

One can specify which methods the function handles in the **methods** parameter

```
@app.route("/hello", methods=["GET"])
def hello():
    return "hi!"

@app.route("/goodbye", methods=["POST"])
def hello():
    return "bye!"
```

# HTTP methods

Create a web server that stores the mood you're in right now.

It can start with happy :)

# Returning JSON

Flask has a **jsonify** function that we can use to convert the data we want to JSON:

```
from flask import Flask, jsonify

app = Flask("simplest server")

@app.route("/hello/<name>")
def hello(name):
    return jsonify({"message": "hello", "name": name})
```

# Returning JSON

HTTP server to count the number of requests it receives

# Exercise 1

Create a web server that maintains a list of the books you've read.

You should be able to add and delete individual books, and list all the books you've read.