with Python!

Barbara Stempień Follow
Jul 24 · 8 min read

Have you ever spent hours extracting data from the web? Got very bored copy-pasting hundreds of values into a spreadsheet? Have you dreamed about copy-paste superhero? Someone who could save you from doing such tedious tasks? Someone who would truly enjoy doing it?

If yes, then here it is! Our copy-paste superhero:

A copy-paste superhero!

Before we talk about our superhero, let's first talk about the task itself—extracting data from the web.

There are several ways to extract large amounts of data in **an automated way.** The most popular are:

- **APIs**—large websites, like Google, Youtube, Facebook, Twitter, LinkedIn, Pinterest, StackOverflow etc. provide APIs to access their data in a structured way;
- **Web scraping**—if there is no API, you may need to scrape the web-site using a web crawler

APIs are great, but not always available. In this article, we will focus on the **web scraping**.

Web scraping, what does it actually mean?

A quick search in the Wikipedia will give us the following result:

> *Web scraping, web harvesting, or web data extraction is data scraping used for extracting data from websites. Web scraping software may access the World Wide Web directly using the Hypertext Transfer Protocol, or through a web browser. While web scraping can be done manually by a software user, the term typically refers to automated processes implemented using a bot or web crawler. It is a form of copying, in which specific data is gathered and copied from the web, typically into a central local database or spreadsheet, for later retrieval or analysis.*

Alright, now that we know what it means, the next question is…

How to build a web scraping program?

There are many ways to build a web scraping program, but we will focus on building it in **Python**.

Our goal is to scrape data from the 2014 FIFA World Cup Brazil page, which holds statistics for the group's stage.

2014 FIFA World Cup Brazil — group statistics page

Are we allowed to scrap the data?

When scraping data from a web, we should first check if we are allowed to do it. *Privacy policy* and *Terms and Conditions* might contain some information on web scraping, however the best source is ***robots.txt***.

What is *robots.txt*?

It is a file that indicates which parts of the website you should not scrap. The information contained in the file is not legally binding. However, you should respect the will of the website owner and adapt to the information contained therein.

How to find and read *robots.txt*?

The file is stored at the root of the website:

```
https://www.fifa.com/robots.txt
```

It is written in a plain text editor, and looks like this:

robots.txt from fifa.com

Let's quickly go through the most common rules.

```
User-agent: *
```

The above means that this section applies to all robots. While the below applies to a specific crawler.

```
User-agent: MyBot
```

Disallow followed by a slash, means that scraping from the entire website is not allowed:

```
Disallow: /
```

While an empty *disallow*, means that all pages can be scraped:

```
Disallow:
```

In most of the cases, only some parts of the website are excluded:

```
Disallow: /infoserv/
Disallow: /thirdparties/
Disallow: /preview/
Disallow: /preview/*
```

We might also see something like this:

```
Crawl-delay: 60
```

Crawl-delay specifies how long a crawler has to wait after a crawl action. In the above example, the crawler is instructed to wait 60 seconds before accessing the website again.

At the very end of the file, you can usually see a link to the sitemap:

```
Sitemap: http://www.fifa.com/sitemap_index.xml
```

fifa.com/robots.txt allows scraping from the group statistics page with no crawl-delay. We can then move on to Python!

. . .

Building program in Python

Installing packages and importing libraries

Before we start building our program we have to add some packages to Python. Packages we need are:

- BeautifulSoup
- Requests
- Pandas

Beautiful Soup is a Python library for parsing HTML and XML documents. It creates a parse tree for parsed pages that can be used to extract data from HTML, which is useful for web scraping.

Requests is a Python HTTP library for making HTTP requests in a simple way.

Pandas is a Python library for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

We can install these packages from the command line, by executing the following commands:

```
python -m pip install requests
python -m pip install beautifulsoup4
python -m pip install pandas
```

The above code assumes we already have Python installed on our machine.

For more information on how to install packages using *pip*, please see Python's documentation. For information on how to install packages using *conda*, please see Anaconda's documentation.

Once we have all packages installed, we can create Python file and import libraries:

```
1  import requests
2  from bs4 import BeautifulSoup
3  import pandas as pd
```

imports.py hosted with ❤ by GitHub view raw

```
1  import requests
2  from bs4 import BeautifulSoup
```

Requesting page content

The next step is to request page content. First, we set the *url* variable to a website URL. Then we use a *try-except* block to check if the page is accessible.

```
1   url = 'https://www.fifa.com/worldcup/archive/brazil20
2
3   try:
4       page_response = requests.get(url, timeout=5)
5
6       if page_response.status_code == 200:
7           page_content = BeautifulSoup(page_response.co
```

But what exactly is happening inside the *try-except* block?

We query the website using *requests.get* with timeout set to 5 seconds, and we store HTTP status code (response) in *page_response*.

- If the page is not accessible (*else* statement) or if a timeout occurred (*except* statement) we print notification to the console.
- If the page is accessible (200 is a standard response for successful HTTP requests), we assign the content of the page to the *page_content* variable using Beautiful Soup format.

Extracting the data

Now, that we have the entire HTML content stored in the python's variable, we can write a code that will extract data out of it. However, before we can do that, we have to understand where exactly the data is stored and how to access it.

HTML, CSS and Developer Tools

To analyze the HTML structure, we can open the FIFA website and use browser's **Developer Tools** (keyboard shortcut—F12). In the elements tab, we can see the entire HTML structure, and we can use the selection tool to find an element that stores data for all of the groups.

Exploring group statistics page with Developer Tools

After exploring for a while, we can see that element which contains data for all of the groups is a *div* element with a *class="inner"*. However, we will not use that element in our program. Instead, we will take its parent element, so a *div* with a *class="module"* and *id="standings"*.

Groups — HTML structure

When selecting elements, we have to be aware that **ids are unique**, but **classes are not**. Therefore, it is possible that there is another element on the page, which also has a *class="inner"*, while there is only one element with *id="standings"*.

Reading HTML and extracting data

Now we know which element contains all statistics, so we can use Beautiful Soup's *find()* method to locate it in the HTML structure (which we previously saved to a *page_content* variable) and save it in the *standings_table* variable.

```
url = 'https://www.fifa.com/worldcup/archive/brazil20

try:
    page_response = requests.get(url, timeout=5)
    if page_response.status_code == 200:

        page_content = BeautifulSoup(page_response.co
```

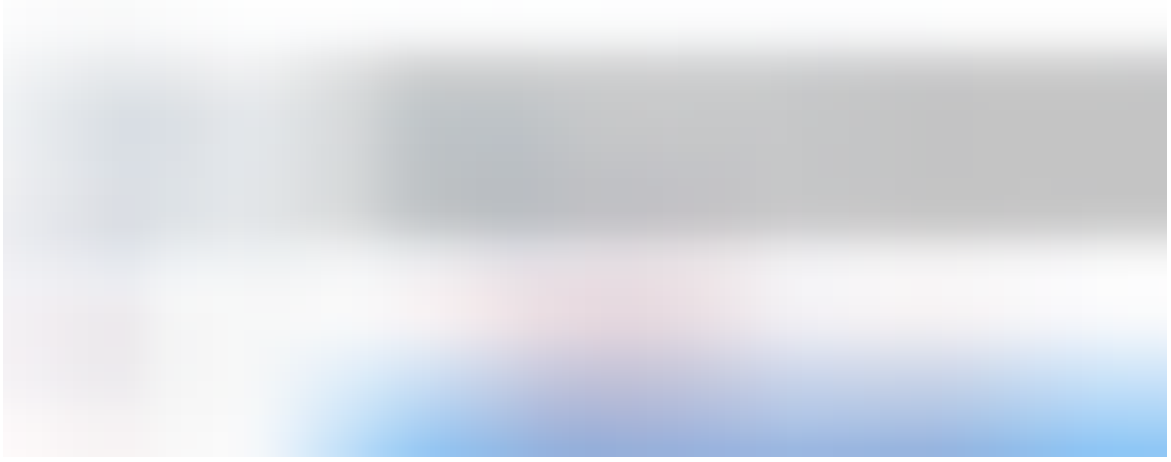
The next step is to locate specific statistics and save them in a python's data structure.

We start with the group letters:

Group letters

We have eight group letters in total, and we want to save all of them in one list. We also want to repeat each letter four times, so that each team is assigned to the group.

We start by analyzing the page structure with **Developer Tools.** We can see that data for each group is stored in the *div* element with *class="group-wrap"* and group letter is stored in a *caption* element with *class="caption-nolink"*.



Group letters — HTML structure

To select all elements that have *class="caption-nolink",* and are descendants of the element with *class="group-wrap",* we can use the following pattern:

```
.group-wrap .caption-nolink
```

We pass this pattern to the Beautiful Soup's *select()* method to find all matching elements. Once we have all of them, we use a list comprehension to get inner text of those elements (group letter) and save the result in python's list.

Then, we use list comprehension again, but this time to repeat each letter in the list four times.

```
1   url = 'https://www.fifa.com/worldcup/archive/brazil20
2
3   try:
4       page_response = requests.get(url, timeout=5)
5
6       if page_response.status_code == 200:
7           page_content = BeautifulSoup(page_response.co
```

The next statistic we want to scrap is the number of the match played by each team.

Match played

A quick look into the HTML structure with **Developer Tools** and we know that numbers of the match played is stored in the *span* element with *class=”text”*. This *span* element is stored in the *td* element with *class=”tbl-matchplayed”*. Since *‘text’* class is too general, and there is a high probability that other elements on the page also have it, we create the following pattern:

```
.group-wrap .tbl-matchplayed span.text
```

This pattern selects all *span* elements with class *text*, which are descendants of the element with class *tbl-matchplayed*, and the element with class *tbl-matchplayed* must be descendant of the element with class *group-wrap*.

Hard to understand? Let’s again look at the HTML tree:

Match played — HTML structure

We pass this pattern to the Beautiful Soup's *select()* method to find all matching elements. Once we have all of them, we use a list comprehension to get inner text of those elements (number of the match played) and save the result in python's list.

```
1   url = 'https://www.fifa.com/worldcup/archive/brazil20
2
3   try:
4       page_response = requests.get(url, timeout=5)
5
6       if page_response.status_code == 200:
7           page_content = BeautifulSoup(page_response.co
```

We repeat the same operation for all other statistics on the page. As a result, we get ten python's lists:

```
1   url = 'https://www.fifa.com/worldcup/archive/brazil20
2
3   try:
4       page_response = requests.get(url, timeout=5)
5
6       if page_response.status_code == 200:
7           page_content = BeautifulSoup(page_response.co
```

As a final step, we save all statistics to pandas dataframe:

```
url = 'https://www.fifa.com/worldcup/archive/brazil20

try:
    page_response = requests.get(url, timeout=5)

    if page_response.status_code == 200:
        page_content = BeautifulSoup(page_response.co
```

Final result (top 20 records):

Thank you for reading!