How To Web Scrape Wikipedia Using Python, Urllib, Beautiful Soup and Pandas

In this tutorial we will use a technique called web scraping to extract data from a website.

We'll be using **Python 3.7** through a **Jupyter Notebook** on **Anaconda** and the **Python** libraries **urllib**, **BeautifulSoup** and **Pandas**.

(If you don't have Anaconda or Jupyter Notebook installed on your Windows machine, check out our tutorial How Do I Install Anaconda On Windows (https://lowmess.com/how-install-anaconda-windows/)? before getting started. If you're on Linux or Mac OS X you'll have to Google it. Bill Gates fanboy in the house....)

What is Web Scraping?

Web scraping (also known as screen scraping, data scraping, web harvesting, web data extraction and a multitude of other aliases) is a method for extracting data from web pages.

I've done a quick primer on WTF Is…Web Scraping (https://lowmess.com/wtf-is-web-scraping/) to get you up to speed on what it is and why we might use it. Have a quick read and re-join the tour group as soon as possible.

Step By Step Tutorial

OK. Now we know what web scraping is and why we might have to use it to get data out of a website.

How exactly do we get started scraping and harvesting all of that delicious data for our future perusal and use?

Standing on the shoulders of giants.

When I first tried screen scraping with Python I used an earlier version of it and worked through Sunil Ray's Beginner's Guide on the Analytics Vidhya blog (https://www.analyticsvidhya.com/blog/2015/10/beginner-guide-web-scraping-beautiful-soup-python/).

Working with **Python 3.7** now I had to change some libraries plus do a few further corrective steps for the data I'm looking to get hence not just pointing you straight to that article.

Which Python libraries will we be using for web scraping?

Urllib.request

As we are using Python 3.7, we will use urllib.request to fetch the HTML from the URL we specify that we want to scrape.

Learn more about urllib.request. (https://docs.python.org/3.0/library/urllib.request.html)

BeautifulSoup

Once urllib.request has pulled in the content from the URL, we use the power of BeautifulSoup to extract and work with the data within it. BeautifulSoup4 has a multitude of functions at it's disposal to make this incredibly easy for us.

Learn more about Beautiful Soup (https://www.crummy.com/software/BeautifulSoup/bs4/doc/).

Anything else we need to know before we kick this off?

Are you familiar with HTML?

HTML (Hypertext Markup Language) is the standard markup langauge for creating web pages. It consists of a collection of tags which represent **HTML** elements. These elements combined tell your web browser what the structure of the web page looks like. In this tutorial we will mostly be concerned with the **HTML** table tags as our data is contained in a table. For more reading on HTML, check out W3Schools Introduction to HTML (https://www.w3schools.com/html/html_intro.asp).

Right, let's get into it.

1. Open a new Jupyter notebook.

You do have it installed, don't you? You didn't just skip the advice at the top, did you? If so, go back and get that done then come back to this point.

2. Choosing our target Wikipedia page.

Like our friend Sunil, **we are going to scrape some data from a Wikipedia page**. While he was interested in state capitals in India, I've decided to pick a season at random from the English Premier League, namely the 1999/2000 season (https://en.wikipedia.org/wiki/1999%E2%80%932000_FA_Premier_League).

When I went and looked at the page I instantly regretted picking this season (fellow Tottenham Hotspur fans will understand why when they see the manager and captain at the end but I'll stick with it as some kind of sado-masochism if nothing else).

3. Import urllib library.

Firstly, we need to import the library we will be using to connect to the Wikipedia page and fetch the contents of that page:

```
# import the library we use to open URLs
import urllib.request
```

Next we specify the **URL** of the Wikipedia page we are looking to scrape:

```
# specify which URL/web page we are going to be scraping
url = "https://en.wikipedia.org/wiki/1999%E2%80%932000_FA_Premier_League"
```

Using the **urllib.request** library, we want to query the page and put the HTML data into a variable (which we have called 'url'):

```
# open the url using urllib.request and put the HTML into the page variable
page = urllib.request.urlopen(url)
```

4. Import BeautifulSoup library.

Next we want to import the functions from **Beautiful Soup** which will let us parse and work with the HTML we fetched from our Wiki page:

```
# import the BeautifulSoup library so we can parse HTML and XML documents
from bs4 import BeautifulSoup
```

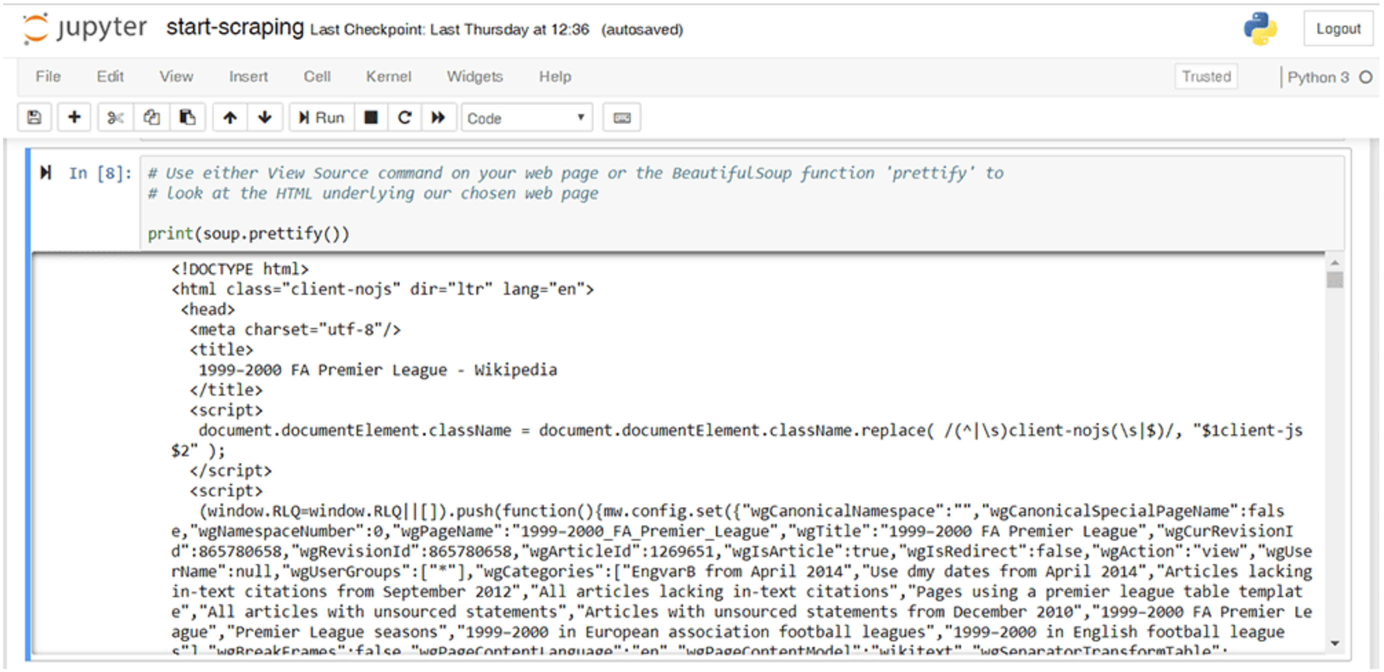
Then we use **Beautiful Soup** to parse the HTML data we stored in our 'url' variable and store it in a new variable called 'soup' in the **Beautiful Soup** format. Jupyter Notebook prefers we specify a parser format so we use the “lxml” library option:

```
# parse the HTML from our URL into the BeautifulSoup parse tree format
soup = BeautifulSoup(page, "lxml")
```

5. Take a look at our underlying HTML code.

To get an idea of the structure of the underlying HTML in our web page, we can view the code in two ways: a) right click on the web page itself and click View Source ***or*** b) use **Beautiful Soup’s prettify function** and check it out right there in our Jupyter Notebook.

Let’s see what prettify() gives us:


jupyter start-scraping Last Checkpoint: Last Thursday at 12:36 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

In [8]:
```
# Use either View Source command on your web page or the BeautifulSoup function 'prettify' to
# look at the HTML underlying our chosen web page

print(soup.prettify())
```

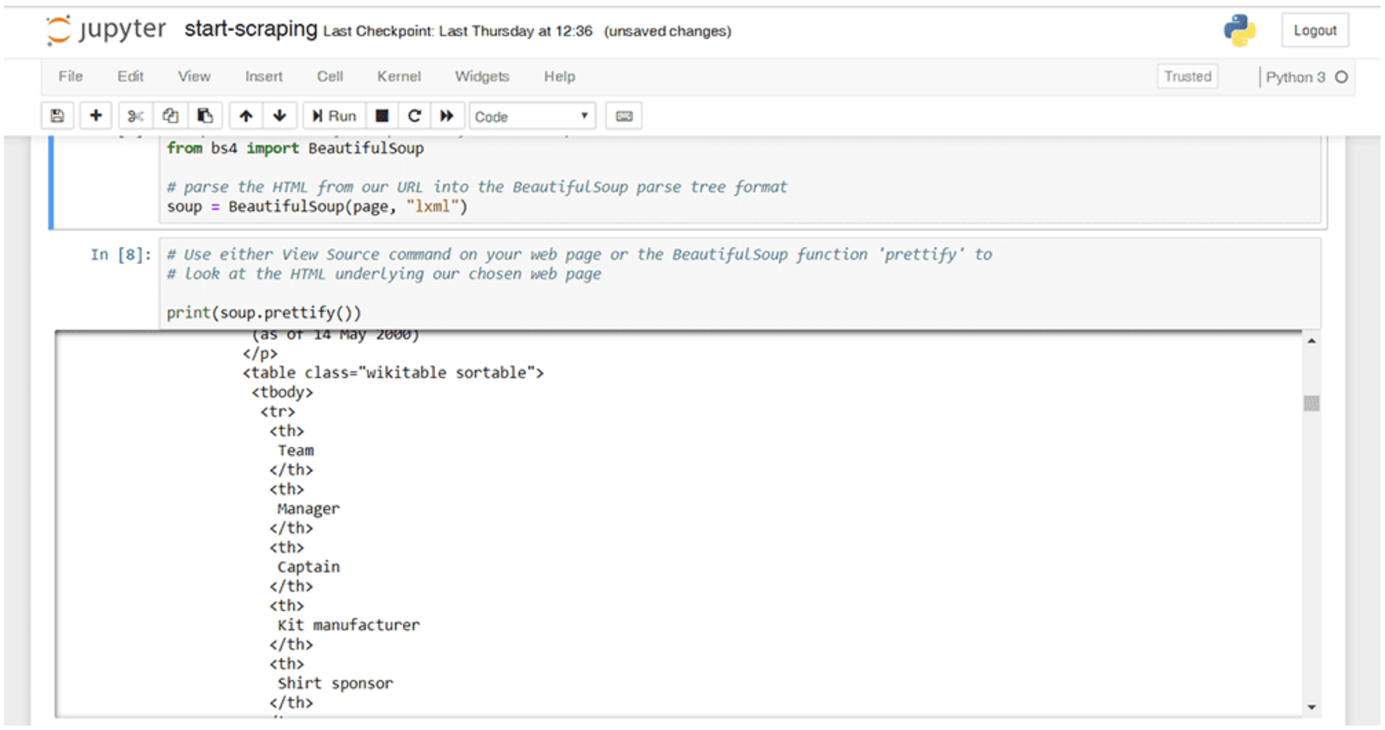
```
<!DOCTYPE html>
<html class="client-nojs" dir="ltr" lang="en">
 <head>
  <meta charset="utf-8"/>
  <title>
   1999-2000 FA Premier League - Wikipedia
  </title>
  <script>
   document.documentElement.className = document.documentElement.className.replace( /(^|\s)client-nojs(\s|$)/, "$1client-js
$2" );
  </script>
  <script>
   (window.RLQ=window.RLQ||[]).push(function(){mw.config.set({"wgCanonicalNamespace":"","wgCanonicalSpecialPageName":fals
e,"wgNamespaceNumber":0,"wgPageName":"1999-2000_FA_Premier_League","wgTitle":"1999-2000 FA Premier League","wgCurRevisionI
d":865780658,"wgRevisionId":865780658,"wgArticleId":1269651,"wgIsArticle":true,"wgIsRedirect":false,"wgAction":"view","wgUse
rName":null,"wgUserGroups":["*"],"wgCategories":["EngvarB from April 2014","Use dmy dates from April 2014","Articles lacking
in-text citations from September 2012","All articles lacking in-text citations","Pages using a premier league table templat
e","All articles with unsourced statements","Articles with unsourced statements from December 2010","1999-2000 FA Premier Le
ague","Premier League seasons","1999-2000 in European association football leagues","1999-2000 in English football league
s"],"wgBreakFrames":false,"wgPageContentLanguage":"en","wgPageContentModel":"wikitext","wgSeparatorTransformTable":
```

6. Find the table we want.

By looking at our Wikipedia page for the 1999/2000 Premier League season, we can see there is a ***LOT*** of information in there. From a written synopsis of the season to specific managerial changes, we have a veritable treasure trove of data to mine.

What we are going to go for though is the table which shows the **personnel and kits** for each Premier League club. It's already set up in nice rows and columns which should make our job a little easier as beginner web scrapers.

Let's have a look for it in our prettifyed HTML:


jupyter start-scraping Last Checkpoint: Last Thursday at 12:36 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
from bs4 import BeautifulSoup

# parse the HTML from our URL into the BeautifulSoup parse tree format
soup = BeautifulSoup(page, "lxml")
```

```
In [8]: # Use either View Source command on your web page or the BeautifulSoup function 'prettify' to
        # look at the HTML underlying our chosen web page

        print(soup.prettify())
```

```
         (as of 14 May 2000)
        </p>
        <table class="wikitable sortable">
         <tbody>
          <tr>
           <th>
            Team
           </th>
           <th>
            Manager
           </th>
           <th>
            Captain
           </th>
           <th>
            Kit manufacturer
           </th>
           <th>
            Shirt sponsor
           </th>
```

And there it is.

Starting with an **HTML <table> tag** with a class identifier of "**wikitable sortable**". We'll make a note of that for further use later.

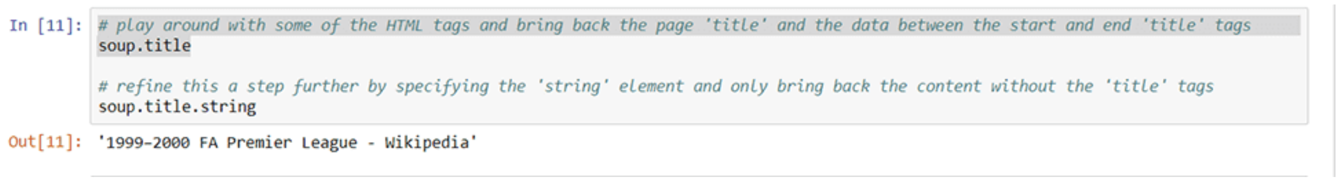
Scroll down a little to see how the table is made up and you'll see the rows start and end with **<tr>** and **</tr>** tags.

The top row of headers has **<th>** tags while the data rows beneath for each club has **<td>** tags. It's in these <td> tags that we will tell Python to extract our data from.

7. Some fun with BeautifulSoup functions.

Before we get to that, let's try out a few **Beautiful Soup** functions to illustrate how it captures and is able to return data to us from the HTML web page.

If we use the title function, Beautiful Soup will return the HTML tags for the title and the content between them. Specify the string element of '**title**' and it gives us just the content string between the tags:



```
In [11]: # play around with some of the HTML tags and bring back the page 'title' and the data between the start and end 'title' tags
         soup.title

         # refine this a step further by specifying the 'string' element and only bring back the content without the 'title' tags
         soup.title.string
Out[11]: '1999-2000 FA Premier League - Wikipedia'
```

8. Bring back ALL of the tables.

We can use this knowledge to start planning our attack on the HTML and homing in only on the table of personnel and kit information that we want to work with on the page.

We know the data resides within an **HTML** table so firstly we send **Beautiful Soup** off to retrieve all instances of the <table> tag within the page and add them to an array called all_tables:

```
# use the 'find_all' function to bring back all instances of the 'table' tag in t
all_tables=soup.find_all("table")
all_tables
```

Looking through the output of "**all_tables**" we can again see that the class id of our chosen table is "**wikitable sortable**". We can use this to get BS to only bring back the table data for this particular table and keep that in a variable called "**right_table**":

```
right_table=soup.find('table', class_='wikitable sortable')
right_table
```

9. Ignore the headers, find the rows.

Now it starts to get a little more technical. We know that the table is set up in rows (starting with <tr> tags) with the data sitting within <td> tags in each row. We aren't too worried about the header row with the <th> elements as we know what each of the columns represent by looking at the table.

To step things up a notch we could have set BeautifulSoup to find the <th> tags and assigned the contents of each to a variable for future use.

We've got enough to get getting on with getting the actual data though so let's crack on.

10. Loop through the rows.

We know we have to **start looping through the rows** to get the data for every club in the table. The table is well structured with each club having it's own defined row. This makes things somewhat easier.

There are five columns in our table that we want to scrape the data from so we will set up five empty lists (A, B, C, D and E) to store our data in.

To start with, **we want to use the Beautiful Soup 'find_all' function again** and set it to look for the string 'tr'. We will then set up a **FOR** loop for each row within that array and set Python to loop through the rows, one by one.

Within the loop we are going to use **find_all** again to search each row for <td> tags with the 'td' string. We will add all of these to a variable called 'cells' and then check to make sure that there are 5 items in our 'cells' array (i.e. one for each column).

If there are then we use the **find(text=True))** option to extract the content string from within each <td> element in that row and add them to the A-E lists we created at the start of this step. Let's have a look at the code:

```
A=[]
B=[]
C=[]
D=[]
E=[]

for row in right_table.findAll('tr'):
    cells=row.findAll('td')
    if len(cells)==5:
        A.append(cells[0].find(text=True))
        B.append(cells[1].find(text=True))
        C.append(cells[2].find(text=True))
        D.append(cells[3].find(text=True))
        E.append(cells[4].find(text=True))
```

Still with me? Good. This all should work perfectly, shouldn't it?

We're looping through each row, picking out the <td> tags and plucking the contents from each into a list.

Bingo. This is an absolute gift. Makes you wonder why people make such a fuss about it, doesn't it?

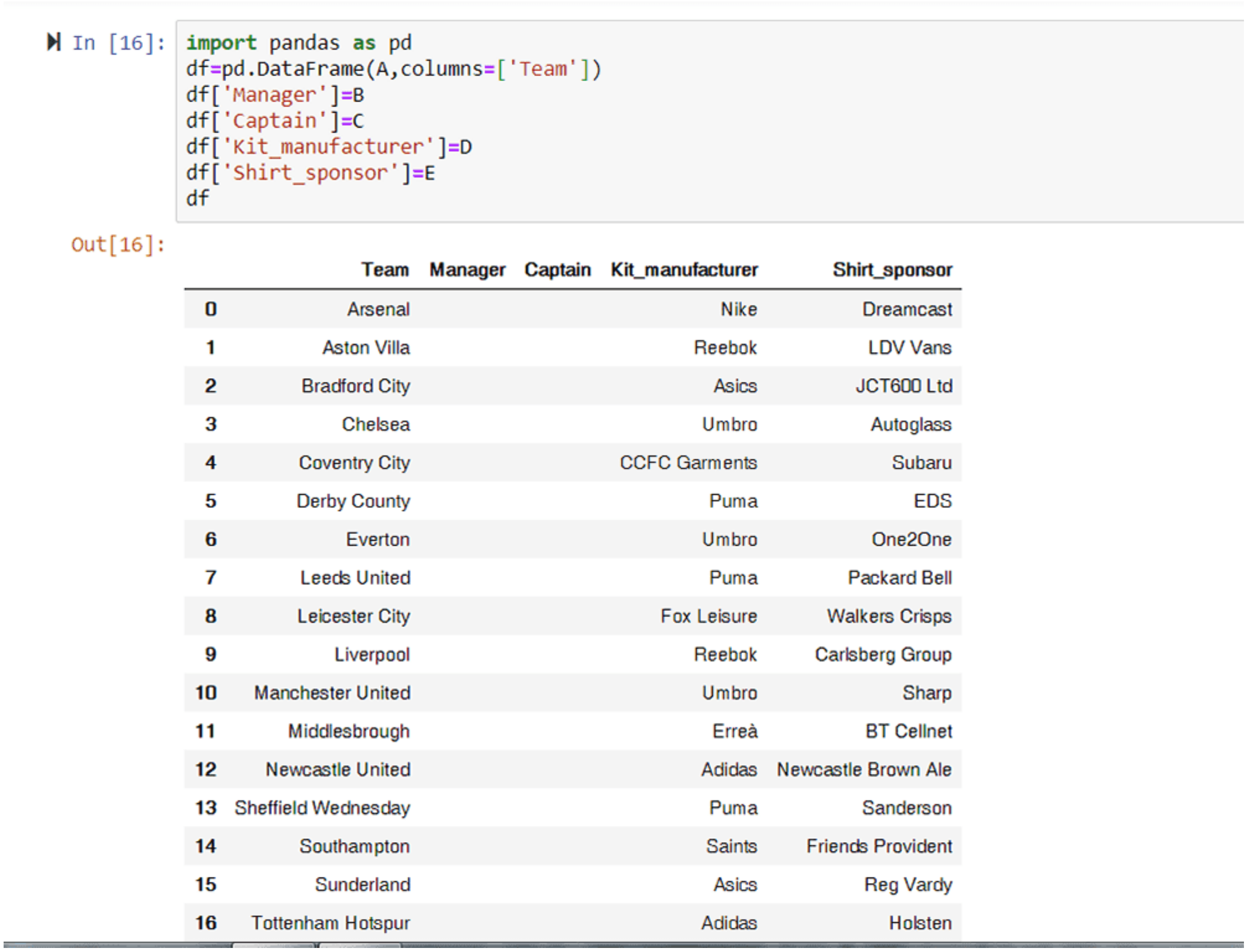
11. Introducing pandas and dataframes.

To see what our loop through the Personnel and Kits table has brought us back, we need to bring in another big hitter of the Python library family – **Pandas**. Pandas lets us convert lists into **dataframes** which are 2 dimensional data structures with rows and columns, very much like spreadsheets or SQL tables.

We'll import pandas and create a **dataframe** with it, assigning each of the lists A-E into a column with the name of our source table columns i.e. Team, Manager, Captain, Kit_Manufacturer and Shirt_Sponsor.

```
import pandas as pd
df=pd.DataFrame(A,columns=['Team'])
df['Manager']=B
df['Captain']=C
df['Kit_manufacturer']=D
df['Shirt_sponsor']=E
df
```

Let's run the Pandas code and see what our table looks like:



```
In [16]: import pandas as pd
         df=pd.DataFrame(A,columns=['Team'])
         df['Manager']=B
         df['Captain']=C
         df['Kit_manufacturer']=D
         df['Shirt_sponsor']=E
         df
Out[16]:
```

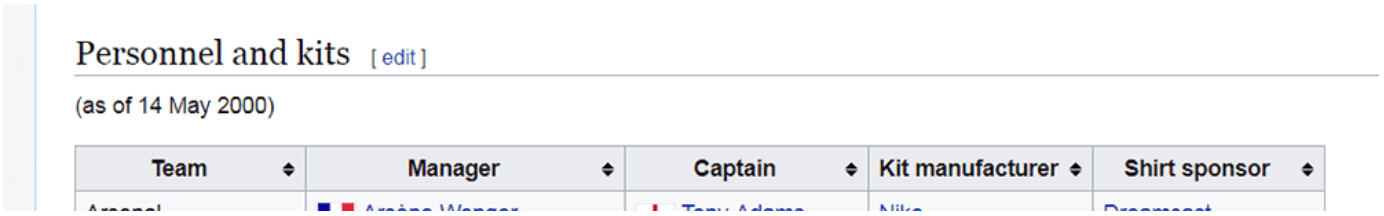
	Team	Manager	Captain	Kit_manufacturer	Shirt_sponsor
0	Arsenal			Nike	Dreamcast
1	Aston Villa			Reebok	LDV Vans
2	Bradford City			Asics	JCT600 Ltd
3	Chelsea			Umbro	Autoglass
4	Coventry City			CCFC Garments	Subaru
5	Derby County			Puma	EDS
6	Everton			Umbro	One2One
7	Leeds United			Puma	Packard Bell
8	Leicester City			Fox Leisure	Walkers Crisps
9	Liverpool			Reebok	Carlsberg Group
10	Manchester United			Umbro	Sharp
11	Middlesbrough			Erreà	BT Cellnet
12	Newcastle United			Adidas	Newcastle Brown Ale
13	Sheffield Wednesday			Puma	Sanderson
14	Southampton			Saints	Friends Provident
15	Sunderland			Asics	Reg Vardy
16	Tottenham Hotspur			Adidas	Holsten

Hmmm. That's not what we wanted. Where's the Manager and Captain data?

Clearly something went wrong in those cells so we need to go back to our **HTML** to see what the problem is.

12. Searching for the problem.

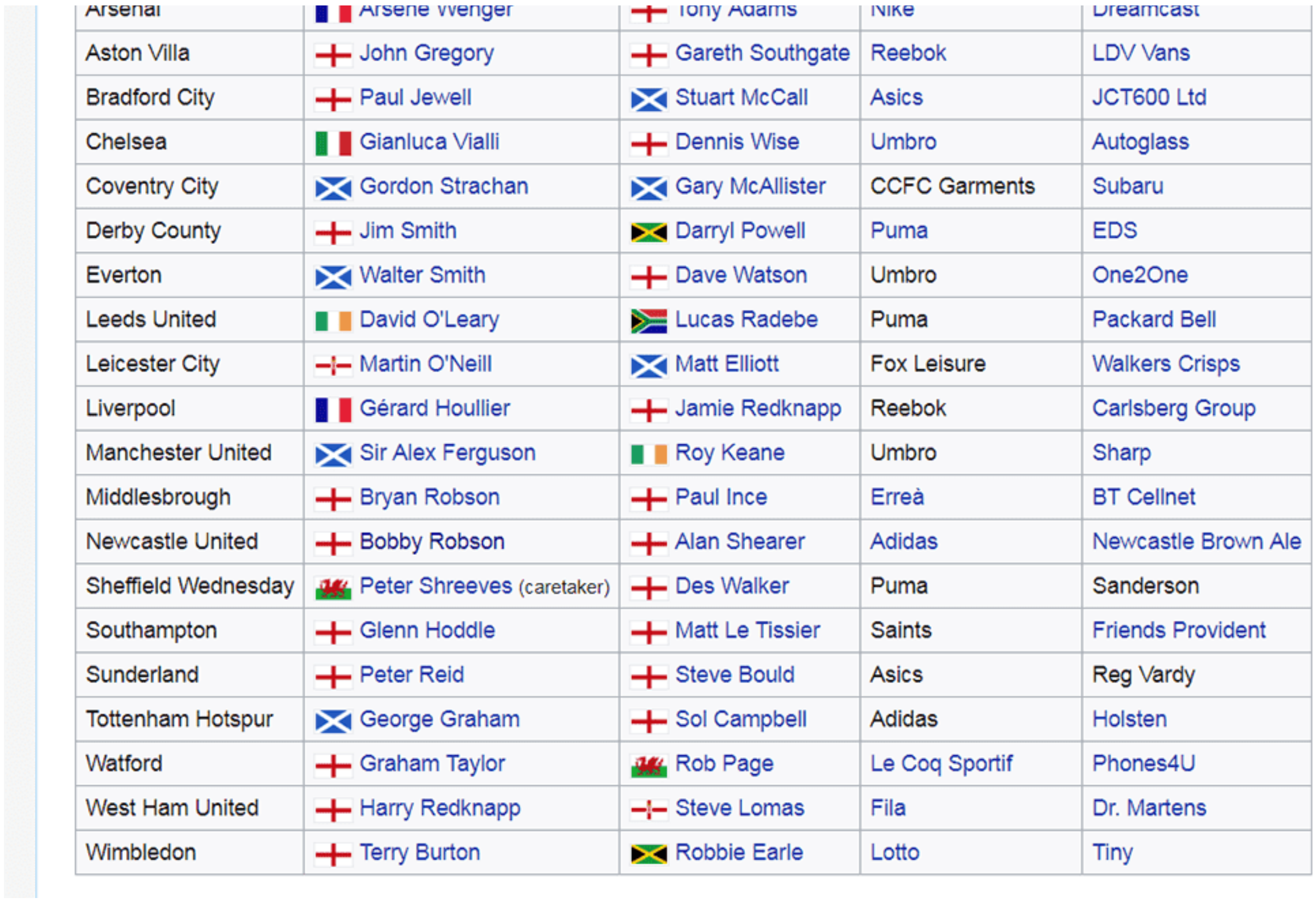
Looking at our **HTML**, there does indeed seem to be something a little different about the Manager and Captain data within the <td> tags. Wikipedia has (very helpfully/unhelpfully) added a little flag within <span> tags to help display the nationality of the Managers and Captains in question.



Personnel and kits [edit]

(as of 14 May 2000)

Team	Manager	Captain	Kit manufacturer	Shirt sponsor



Arsenal	Arsène Wenger	Tony Adams	Nike	Dreamcast
Aston Villa	John Gregory	Gareth Southgate	Reebok	LDV Vans
Bradford City	Paul Jewell	Stuart McCall	Asics	JCT600 Ltd
Chelsea	Gianluca Vialli	Dennis Wise	Umbro	Autoglass
Coventry City	Gordon Strachan	Gary McAllister	CCFC Garments	Subaru
Derby County	Jim Smith	Darryl Powell	Puma	EDS
Everton	Walter Smith	Dave Watson	Umbro	One2One
Leeds United	David O'Leary	Lucas Radebe	Puma	Packard Bell
Leicester City	Martin O'Neill	Matt Elliott	Fox Leisure	Walkers Crisps
Liverpool	Gérard Houllier	Jamie Redknapp	Reebok	Carlsberg Group
Manchester United	Sir Alex Ferguson	Roy Keane	Umbro	Sharp
Middlesbrough	Bryan Robson	Paul Ince	Erreà	BT Cellnet
Newcastle United	Bobby Robson	Alan Shearer	Adidas	Newcastle Brown Ale
Sheffield Wednesday	Peter Shreeves (caretaker)	Des Walker	Puma	Sanderson
Southampton	Glenn Hoddle	Matt Le Tissier	Saints	Friends Provident
Sunderland	Peter Reid	Steve Bould	Asics	Reg Vardy
Tottenham Hotspur	George Graham	Sol Campbell	Adidas	Holsten
Watford	Graham Taylor	Rob Page	Le Coq Sportif	Phones4U
West Ham United	Harry Redknapp	Steve Lomas	Fila	Dr. Martens
Wimbledon	Terry Burton	Robbie Earle	Lotto	Tiny

It sure looks nice on the Wiki page but it's messing up my screen-scraping tutorial so I'm somewhat less than happy to have it in there.

Using the knowledge we've gained above, is there a simple way to workaround this problem and just lift out the Manager and Captain names as we planned?

This is how I decided to do it.

Looking at the HTML code, I can see that there are two sets of <a> tags i.e. hyperlinks within each cell for both the Manager and Captain data. The first is a link over the flag's <img> tag and the second is a link on the Manager/Captain's name.

If we can get the content string between the <a> and </a> tags on the ***SECOND*** of those, we have got the data we need.

I did a 'find_all' within the individual cells to look for the <a> tags and assign that to a variable (mlnk for Managers, clnk for Captains). I knew it was the second <a> tag's content string that I needed to get the name of the Manager and the Captain so I appended the content of the second element in the mlnk/clnk array I had created to the specific list (list B for Managers, list C for Captains).

As so:

```
A=[]
B=[]
C=[]
D=[]
E=[]

for row in right_table.findAll('tr'):
    cells=row.findAll('td')
    if len(cells)==5:
        A.append(cells[0].find(text=True))
        mlnk=cells[1].findAll('a')
        B.append(mlnk[1].contents[0])
        clnk=cells[2].findAll('a')
        C.append(clnk[1].contents[0])
        D.append(cells[3].find(text=True))
        E.append(cells[4].find(text=True))
```

Now run that and re-run our pandas code from before and “hopefully” we’ll fill in those blanks from the previous output:

Hurrah!

We now have 20 rows for the 20 clubs with columns for Team Name, Manager, Captain, Kit Manufacturer and Shirt Sponsor. **Just like we always wanted.**

(I'll ignore the names in the Manager and Captain columns for Tottenham, must research my examples better before getting started…)

Wrapping Up

That successful note brings us to the end of our **Getting Started Web Scraping with Python** tutorial. Hopefully it gives you enough to get working on to try some scraping out for yourself. We've introduced **urllib.request** to fetch the URL and HTML data, **Beautiful Soup** to parse the HTML and **Pandas** to transform the data into a dataframe for presentation.

We also saw that things don't always work out just as easily as we hope for when working with web pages but it's best to roll with the punches and come up with a plan to workaround it as simply as possible,

If you have any questions, please send me a mail (alan AT alanhylands DOT com). Happy scraping but if you get caught…we never met!