[Excerpt from "Programming with Python for Social Scientists"]

### 13. 1. Show Me the Code! Doing Detective Work in HTML

Before we dig into any scraping of information from the web, we need to first understand the format in which that information is presented. HTML, or HyperText Markup Language, is the standard language for presenting information on the web via a browser – HTML code is the building blocks out of which a webpage can be made. So, whenever you’re looking at a webpage, you’re looking at the end result of some code written in HTML. However, for our purposes, we don’t need to know too much about how to construct webpages with HTML – after all, this book is about teaching you Python, not other languages! What we *do* need to be aware of is how HTML elements – the individual bits of a webpage – appear in code, so that we can use that knowledge to scrape out the information we want from any given webpage.

The first thing to note is that HTML is a language that looks quite similar to something we’ve already seen; XML (cf. Chapter 12, section 12. 4). Just as with XML, HTML provides a way of structuring information with “tags” to keep various different elements organised and readily accessible. We don’t need to know about all the different features available within HTML, but here’s an example of some really basic HTML code that we can use to understand what sort of thing we’re likely to find behind any given webpage:

<html>

<body>

<h1>This is the main title of the webpage</h1>

<h2>This is a sub-heading</h2>

<p>This is a paragraph of text.</p>

<h2>This is another sub-heading</h2>

<p>This is a paragraph of text with <b>some words</b> in bold.</p>

<img src=“image.jpg” width=“100” height=“100”>

<p>And that just above is an image.</p>

</body>

</html>

This is what a website looks like “behind the scenes” – just various bits of information tagged in different ways. What can we learn from this? From what we learned about XML in Chapter 12, We might already recognise the use of “angle brackets” (e.g. < and >) that sit around each piece of information and define what *type* of information it is. These sets of angle brackets always come in pairs – one set before the information to denote the start of it, and one set after the information to denote where it ends. For instance, we can see there’s a <html> tag right at the beginning of this example which indicates where we’re going to see some HTML code placed, and then a </html> tag right at the bottom to close the tag. Similarly, within that <html></html> pairing we can see that there is also a <body></body> pairing (which denotes where the “body text” of the HTML is going to begin and end). And there are various other elements within that body text too – there are text headings (denoted with the <h1></h1> and <h2></h2> pairings) and text paragraphs (denoted with <p></p> pairings). The one thing in this example that *doesn’t* come as a pair of tags is the image – this comes as a single set of angle brackets in which the image type is declared (“img”) and various paramaters are set (e.g. the source/location of the image we want to see on this mockup webpage, and its dimensions in pixels) – so, clearly sometimes the tags don’t come in pairs, but the information required can instead be contained within the brackets of a single tag.

###### Exploring a real example of HTML

All of this is not intended as instruction in how to create a webpage out of raw HTML code – as I’ve said, this is a book about Python, not HTML. However, the thing we need to pay attention to for the purposes of web scraping in Python is that each piece of information in a HTML script comes “tagged” as a specific type of content – hence, if we want to pull out certain types of information from a webpage, we’ll need to know what type of content it is tagged as. Now we can look at a real webpage and see the code that constructs it to start working our way towards scraping it out with Python.

For this example, the information I’m interested in scraping are the text transcripts of (UK) Prime Minister’s speeches, which are freely available via the UK governments official website – as social scientists, we might use this as material for thinking about the rhetoric that sits around various political ideologies on various topics. I have chosen one particular speech to work with here: a speech from June 2018 where Prime Minister Theresa May was speaking about the UK National Health Service (GOV.UK, 2018). So let’s visit that webpage and do some detective work to see how that speech transcripts looks in HTML:

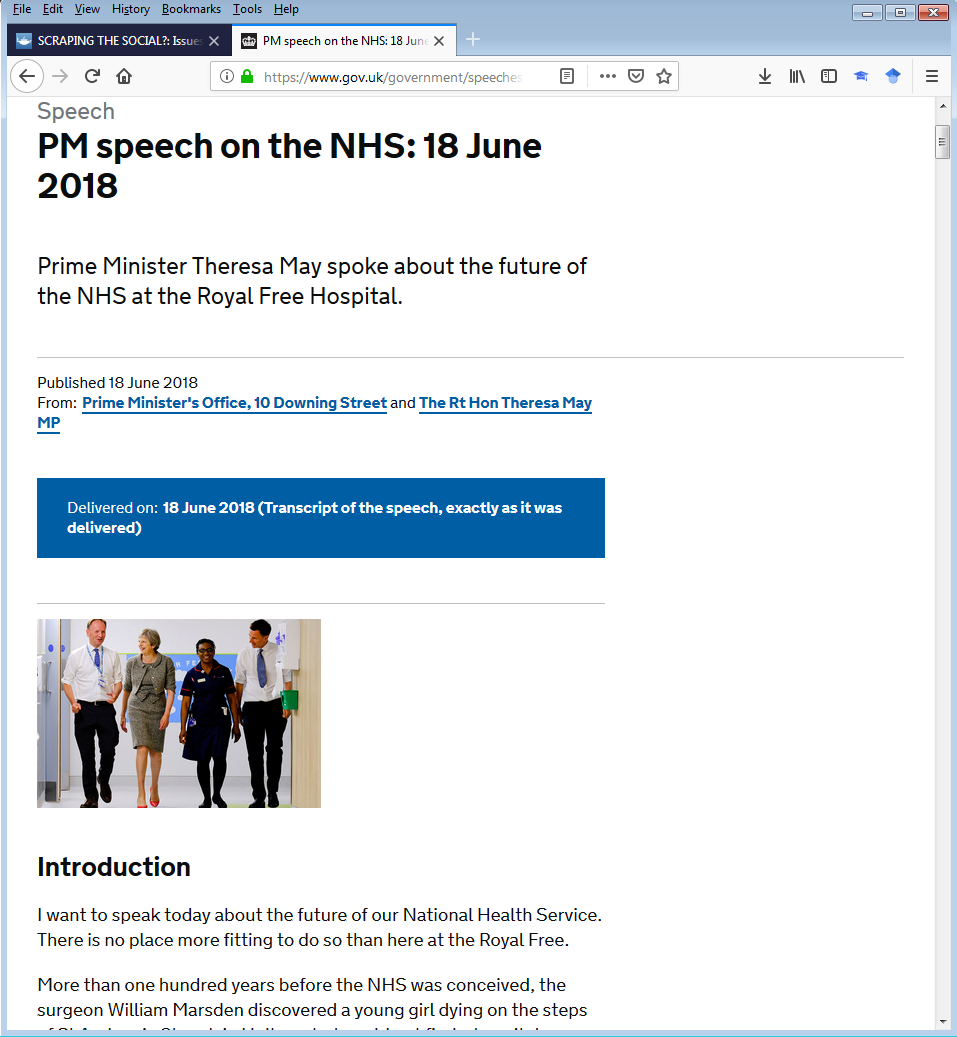


Figure 13. 1. The speech we’re scraping, in its webpage format

So this is what the webpage looks like when we visit it in a web browser. We might already be able to guess what we’re going to find in the HTML code to some extent – alongside various other bits and bobs like links and clickable buttons there are some headings, there’s an image, there are some paragraphs of text, and we’ve seen how they might look in HTML in the example above already. If I’m interested in pulling out the text of the speech (some of which is visible in Figure 13. 1. Above), I’m going to want to see what’s in the paragraphs of body text rather than focus on things like the images and links and so on – this gives me a clue what to look out for in the HTML code.

Now comes the fun bit: let’s get into the HTML! In my particular browser, going to the “Tools” menu and then the “Web Developer” option allows me to select to see the “Page Source” (i.e. the HTML code through which the website is constructed)[[1]](#footnote-1). And if I do that, here’s what I see:

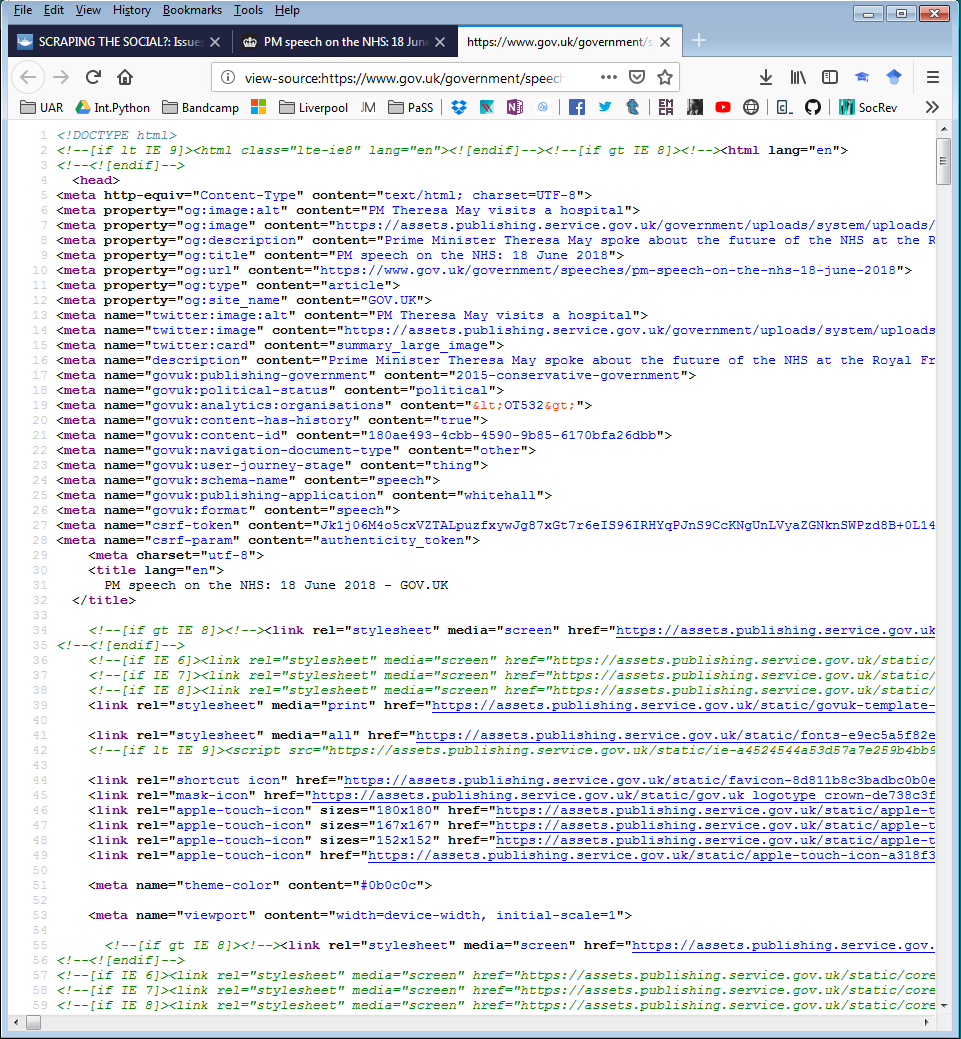


Figure 13. 2. Some of the HTML code of the webpage we’re scraping

Phew, that’s a lot of stuff to look at! As noted already though, we don’t need to understand the ins and outs of this page – all we need to do is find the bits of information we’re interested in extracting from this webpage to see how it has been tagged. And to do that, we can do a bit of simple detective work to sift through the code to locate information relevant to us. Look back up at Figure 13. 1. – we can see that in the opening paragraph of text there’s a line that begins “I want to speak today”. Given we can see that on the webpage, we know that it must also therefore be present in the HTML code somewhere, and therefore, we can search for this text to identify which bits of code to look out for in the whole script[[2]](#footnote-2). So, let’s use this bit of text – “I want to speak today” – to try and identify the lines of HTML we’re interested in and the tags that they come tagged with.

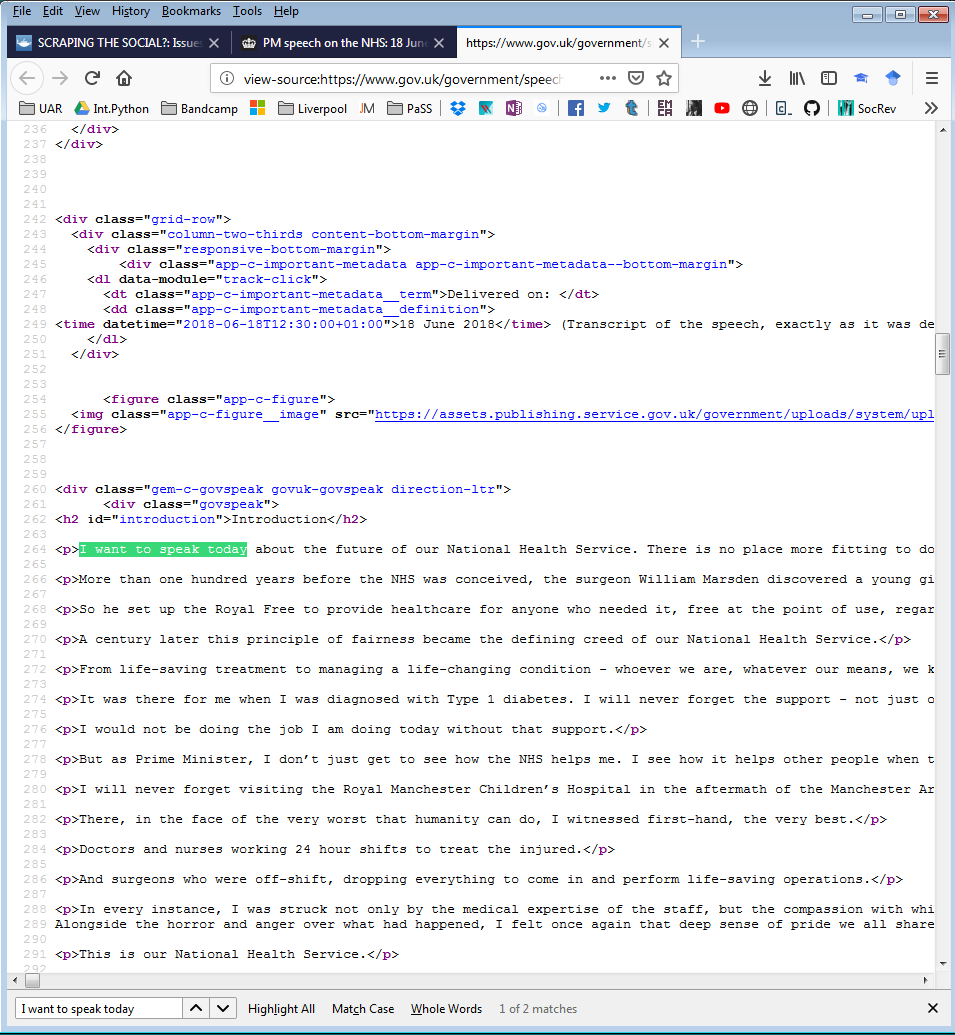


Figure 13. 3. Searching for the text of the speech

Right at the bottom of Figure 13. 3. you can see I’ve opened up a “find” menu and typed in the phrase I want to search for (“I want to speak today”)[[3]](#footnote-3) – because HTML code is stored as text, it’s possible to search through that text using text strings in this way, and we’re taking advantage of this here. You can also see in Figure 13. 3. that I’ve been able to locate the section of code where this opening line of the speech is stored – it looks like every paragraph of the Prime Minister’s speech on this page is tagged as an individual paragraph (i.e. the bits of text I’m interested in are contained within a <p></p> tag pairing). Perfect! Now we have done enough digging around in this webpage to know that all the information we want is stored as paragraphs tagged with <p></p>, and now we just need to tell Python to pull it out!

1. I use Mozilla Firefox over other browsers for various reasons – privacy, customisation, the warm fuzzy feeling of using open source software – but you will be able to see the HTML behind websites using any popular browser of your choice. The commands to enable this might be different though, so be aware of this if you’re not using Firefox yourself. [↑](#footnote-ref-1)
2. It’s worth noting that searching for text is not *always* going to work for every potential case – for instance, what if the text we’re actually interested in wasn’t written into the HTML *as* text, but as an image or something else like that? In that case, the text *wouldn’t* be available in the code to search and we’d have to do some more detective work to try to find it by another means. However, that’s part of the fun of web scraping: figuring out the internal structure of the webpage and whatever it is you’re looking for within it. And in fact, that’s exactly what Marres and Weltevrede (2013) are trying to get us to pay attention to, as noted in the introduction to this chapter. [↑](#footnote-ref-2)
3. On my browser of choice, Mozilla Firefox, you can do this by pressing Ctrl+F – this might apply to your browser of choice also, but if not, you should still be able to find a way to search a page for text information. If you can’t, then I recommend you switch browsers, since web scraping is going to be unnecessarily difficult without these sorts of tools for helping you do detective work on HTML code. [↑](#footnote-ref-3)