Scraping data 101

Simple scrapers in Google Spreadsheets

In Google Drive, select Create>Spreadsheet, and type in any cell:

=ImportHTML("ENTER THE URL HERE", "table", 1)

This formula will go to the URL you specify, look for a table, and pull the first one into your spreadsheet.

Source: Paul Bradshaw's Scraping for Journalists

Replace "ENTER THE URL HERE" with “https://en.wikipedia.org/wiki/David_Bowie_discography".

Try it and see what happens. It should look like this:

=ImportHTML("https://en.wikipedia.org/wiki/David_Bowie_discography", "table", 1)

After a moment, the spreadsheet should start to pull in data from the first table on that webpage

Breaking it down:

=ImportHTML("https://en.wikipedia.org/wiki/David_Bowie_discography", "table", 1)

The scraping formula above has two core ingredients: a function, and parameters:

- importHTML is the function.

Functions do things.

This one “Imports data from a table or list within an HTML page”

All the items within the parentheses (brackets) are the parameters.

("https://en.wikipedia.org/wiki/David_Bowie_discography", "table", 1)

Parameters are the ingredients that the function needs in order to work.

In this case, there are three: a URL, the word "table", and a number 1.

You can use different functions in scraping to tackle different problems, or achieve different results.

Google Drive, for example, also has functions called importXML, importFeed and importData

You can also write scrapers with programming languages like Python, Ruby, R or PHP

You can create your own functions that extract particular pieces of data from a page or PDF

Back to the formula:
=ImportHTML("https://en.wikipedia.org/wiki/David_Bowie_discography", "table", 1)

In addition to the function and parameters, other things you should notice:

• First, the = sign at the start. This tells Google Drive that this is a formula, rather than a simple number or text entry

• Second, notice that two of the three parameters use straight quotation marks: the URL, and “table".

This is because they are strings: strings are basically words, phrases or any other collection (i.e. string) of characters. The computer treats these differently from other types of information, such as numbers, dates, or cell references

• The third parameter does not use quotation marks, because it is a number. In this case it’s a number with a particular meaning: an index, which represents the position of the table we’re looking for (first, second, third, etc)

Knowing these things helps you avoid mistakes

For example, if you omit a quotation mark or use curly quotation marks—these sometimes occur when you cut and paste text into formulas—and in adapting a scraper.

Knowing these things helps you avoid mistakes when you create your formulas.

Replace the number 1 in your formula with a number 2. This should now scrape the second table (in Google Drive an index starts from 1):
=ImportHTML("https://en.wikipedia.org/wiki/David_Bowie_discography", "table", 2)

You should now see the second table from that URL (if there is no second table on the web page, you will get an #N/A error).

You can also use "list" instead of "table" if you want to grab a list from the webpage.

First, make sure the webpage has a list.

=ImportHTML("https://en.wikipedia.org/wiki/David_Bowie_discography", "list", 1)

You can also try replacing either string with a cell reference. For example:
=ImportHTML(A2, "list", 1)

In this case it will look in cell A2 for the URL instead. So in cell A2 type or paste:
https://en.wikipedia.org/wiki/David_Bowie_discography

Notice that you don’t need quotation marks around the URL if it’s in another cell.

Using cell references like this makes it easier to change your formula: instead of having to edit the whole formula you only have to change the value of the cell that it’s drawing from.

When we say “table” or “list” we are specifically asking the scraper to look for an HTML tag in the code of the webpage.

To look at the raw HTML of your webpage, right click on the webpage and select View Page Source, or use the shortcuts CTRL+U (Windows) and CMD+U (Mac) in Firefox.

You can also view it by selecting Tools > Web Developer > Page Source in Firefox or View > Developer > View Source in Chrome.

For viewing source HTML, Firefox and Chrome are generally better set up than other browsers.

You’ll now see the HTML source code.

Use Edit>Find on your browser (or CTRL+F) to search for <table (don’t search for <table> with the > character because sometimes table tags have extra information before that, like <table width=“100">).

When =importHTML looks for a table, this tag is what it looks for.

It will grab everything between <table ...> and </table> (which marks the end of the table)

When "list" is specified, =importHTML will look for the tags

(for unordered list - normally displayed as bullet lists)

or <ol> (ordered list - normally displayed as numbered lists).

The end of each list is indicated by either or .

Both tables and lists will include other tags, such as <li> (list item), <tr> (table row) and <td> (table data,
i.e. a table cell) which add further structure.

That’s what Google Drive uses to decide how to organize that data across rows and columns, but you don’t need to worry about them.

Choosing the right index number: the role of trial and error

How do you know what index number to use?

There are two ways:

1. You can look at the raw HTML and count how many tables there are and choose the one you need.

2. You can use trial and error, beginning with 1, and going up until it grabs the table you want. That's normally quicker.

Trial and error is a common way of learning in scraping. It's pretty common not to get things
right the first time.

Don't expect to know everything there is to know about programming: half the fun is solving the inevitable problems that arise, and half the skill is in the techniques you use to solve them.

When you come across a function (pretty much any word that comes after the = sign) it’s always a good idea to Google it.

Google Drive has extensive help pages - documentation - that explain what the function does, as well as discussion around particular questions.

RECAP:

• Functions do things. They are like one-word references to recipes that someone has already written for you

• Functions need ingredients to do this, supplied in parameters

• There are different kinds of parameters: strings, for example, are collections of characters, indicated by quotation marks…

• …and an index is a position indicated by a number, such as first (1), second (2) and so on.

• The strings "table" and "list" in this formula refer to particular HTML tags in the code underlying a page

• You can store parameters elsewhere - for example in another cell - and use a cell reference to bring them in to your formula.

This makes it easier to tweak your scraper without having to go into the formula every time

Scraper #2

What happens when the data isn’t in a table?

You’ll need a more powerful scraper

We’ll explore the concept of structure, why it’s central in scraping and how to find it.

We’ll use the function importXML

importXML allows us to scrape XML pages. XML is a heavily structured format - much more structured than HTML

Go to the sample XML file at http://www.w3schools.com/xml/cd_catalog.xml

In a Google Drive spreadsheet, scrape that XML file by inserting in a cell, then hitting enter:

=IMPORTXML("http://www.w3schools.com/xml/cd_catalog.xml","CATALOG/CD")

After a few moments you should see the sheet fill with details of CDs.

This function has similar parameters to importHTML in the previous scraper - but only two parameters:

a URL, and a query ("CATALOG/CD").

To see what it’s looking for, open th3 URL in a browser that can handle XML well.

Chrome is particularly good with XML or, failing that, Firefox.

The page has a very clear structure. Starting with <CATALOG>, which branches into a series of tags called <CD>, each of which in turn contains a series of tags: <TITLE>, <ARTIST>, and so on.

You can tell that a tag is contained by another tag, because it is indented after it.

And you can collapse the contents of a tag by clicking on the triangular arrow next to it.

```
<CATALOG>
<CD>
<TITLE>Empire Burlesque</TITLE>
<ARTIST>Bob Dylan</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Columbia</COMPANY>
<PRICE>10.90</PRICE>
<YEAR>1985</YEAR>
</CD>
<CD>
<TITLE>Hide your heart</TITLE>
<ARTIST>Bonnie Tyler</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>CBS Records</COMPANY>
<PRICE>9.90</PRICE>
<YEAR>1988</YEAR>
</CD>
```

https://www.w3schools.com/xml/cd_catalog.xml

So the query in our formula,

=IMPORTXML(“http://www.w3schools.com/xml/cd_catalog.xml","CATALOG/CD")

…looks for each <CD> tag within the <CATALOG> tag, and brings back the contents - each <CD> in its own row.

And because <CD> has a number of tags within it, each one of those is given its own column.

=IMPORTXML(“http://www.w3schools.com/xml/cd_catalog.xml","CATALOG/CD/ARTIST")

To show how you might customize this, try changing it to be more specific as follows:
=IMPORTXML(“http://www.w3schools.com/xml/cd_catalog.xml","CATALOG/CD/ARTIST")

Now it’s looking for the contents of <CATALOG><CD><ARTIST> - so you’ll have a single column of just the artists.

=IMPORTXML(“http://www.w3schools.com/xml/cd_catalog.xml","CATALOG")

…again, because <CATALOG> contains a number of <CD> tags, each is put in its own column.

Finally, try adding an index to the end of your formula. Remember, an index indicates the position of something.

So in our first scraper we used the index 1 to grab the first table. In the importXML formula the index is added in square brackets, like so:

=IMPORTXML(“http://www.w3schools.com/xml/cd_catalog.xml","CATALOG/CD[1]")

Try the formula above - then try using different numbers to see which <CD> tag it grabs.

If you use ImportXML with an XML file and it doesn’t work, it may be because the XML is not properly formatted.

You can test this with the W3School’s XML validator, use Open Refine.

Use a conversion program to change the XML to CSV.

Scraper #3: Looking for structure in HTML

Scraping “help wanted” ads online

=importXML("http://
www.gorkanajobs.co.uk/jobs/journalist/",
“//div[@class='jobWrap']")

IMPORTANT NOTE:

Make sure to check that the quotation marks and inverted commas are straight, not curly.

Better still, type it out yourself.

You can see there’s some complicated code in this formula:

=importXML(“http://www.gorkanajobs.co.uk/jobs/journalist/",
“//div[@class=‘jobWrap’]")

Particularly this: (//div[@class=‘jobWrap']).
It’s known as the “query”

We’ll take a deeper dive into HTML

HTML (HyperText Markup Language) describes content on webpages.

HTML tells a browser whether a particular piece of content is a header, a list, a link, a table, emphasized, etc.

It tells you if something is an image, and what that image represents. It can even say whether a section of content is a piece of navigation, a header or footer, an article, product, advertisement and so on.

HTML is written in tags contained within triangular brackets, also known as chevrons, like this: < >.

Examples:
• The <p> tag indicates a new paragraph.

• <h1> indicates a ‘first level’ header - or the most important header of all.

• <h2> indicates a header which is slightly less important, and an <h3> header is less important, and so on, down to <h6>.

• indicates that a word is emphasized. indicates a strong emphasis (bold).

• <img tells the browser to display an image, which can be found at a location indicated by src=“IMAGE URL HERE”>

And so on. If you see a tag that you don’t recognize, Google it.

Think of a tag as pressing a button. When you see the tag , it is like pressing the ‘bold’ button on a Word document.

The tag (note the slash at the start) turns the formatting ‘off’.

The first tag is called an ‘opening’ tag, and the second a ‘closing’ tag.

When you’re scraping, you are often grabbing everything between an opening and closing tag.

Because tags can contain other tags, HTML is supposed to follow the LIFO rule: Last In First Out.

In other words, if you have more than one tag turned ‘on’ (open), you should turn the last one off (close) first, like so:

```
<html>
<head>
</head>
<body>
<p>Words in
<strong>bold
</strong>
</p>
</body>
</html>
```

In the above example, the tag is contained (‘nested’) within the <p> tag, which is nested in the <body> tag, which is nested in the <html> tag. Each has to be closed in reverse order.
 was the last one in, so it should be the first out, then <p>, <body> and finally <html>.

Oh, and they’re not always conveniently indented as above.

Attributes and values

As well as the tag itself, we can find more information about a particular piece of content by a tag's attributes and values, like so:

In this case:

- <a> is the tag
- href is the attribute
- "http://onlinejournalismblog.com" is the value.

Values are normally contained within quotation marks.

You'll notice that only the tag is turned off - with - which is an easy way to identify it.

Here’s similar code for an image:

The <img> tag tells us this is an image, but where does it come from?

The src attribute directs us to the source, and the ‘value’ of that source is “http://onlinejournalismblog.com/logo.png”.

<img is one of the few tags which are opened and closed within the same tag: the slash at the end of the image code above closes it, so you don’t need a second </img> tag.

Other examples include the line break tag,
 and the horizontal rule tag <hr />)

A single tag can have multiple attributes and values. An image, for example, is likely to not only have a source, but also a title, alternative description (“alt”) and other values, like so:

Classifying sections of content: div, span, classes and ids

The use of attributes and values is particularly important when it comes to the use of the tag to divide content into different sections and the 'id=' and 'class='attributes to classify them.

These sections are often what we want to scrape.

The home page at https://www.cbs.com/all-access/live-tv/, for example, uses the following HTML tags to separate different types of content.

To see these right-click on the page and View Source, then search for “<div” or another tag or attribute

…in fact, there are around 150 different tags on that single page, which makes the class and id attributes particularly useful, as they help us identify the specific piece of content we want to scrape.

And class and id attributes are not just used for tags -

<li class="first">
<ul class="recruiterDetails">

<form class="contrastBg block box-innerSmall"
<label class="hideme"
<li id="job10598" class="regular">
<p class="apply">

<a class=“page"

If data we want to scrape is contained in one of these tags on a page, it makes it much easier to specify.

Back to Scraper #3— Scraping a in an HTML webpage

Let’s re-examine the elements in the query of our importXML scraper:

=importXML("https://www.cbs.com/all-access/live-tv/", "//div[@class='messageContainer']")

You can see that it contains the words ‘div’ and ‘class’.

And, spotting that, you might also search the HTML of that page for ‘messageContainer’ (try it). If you did, you would find this:

This is the tag containing the content that the formula scrapes (until you get to the next tag).

And once you know that, you can customize the scraper to scrape the contents of any div class without needing to understand the slashes, brackets and @ signs that surround it.

This is often how coding operates: you find a piece of code that already works, and adapt it to your own needs. As you become more ambitious, or hit problems, you try to find out solutions - but it’s a process of trial and error rather than necessarily trying to learn everything you might need to know, all at once.

So, how can we adapt this code? Here it is again

=importXML("https://www.cbs.com/all-access/live-tv/" “//div[@class='messageContainer']")

- look for the key words div, class and messageContainer:

Now try to guess how you would change that code to scrape the contents of this tag:

The answer is that we just need to change the ‘messageContainer’ bit of the importXML scraper to reflect the different div class, like so:
=importXML("https://www.cbs.com/all-access/live-tv/", “//div[@class=‘showList']")

You can further adapt the scraper by using ‘id’ instead of ‘class’ if that’s what your HTML uses, and replacing div with whatever tag contains the information you want to scrape.

Data scraping tools

https://dataviz.tools/category/data-scraping/https://dataviz.tools/category/data-scraping/

Data Cleaning in Open Refine

https://github.com/OpenRefine/OpenRefine/releases/

http://miriamposner.com/classes/dh101f17/tutorials-guides/data-manipulation/get-started-with-openrefine/