Web Scraping: For Fun and Profit

Many students have wanted to do something related to web scraping for their final project. As a result, I've decided that it's time to compile a short introduction for everyone. Before we write a web scraper, its best to make sure that someone hasn't already done the thing you want to do. There are countless APIs on the internet that perform a wide variety of purposes (my personal favorite is one that returns a dad joke). Pulling from one of these will generally be a lot faster than writing a brand new web scraper.

There's no API for What I want to do. Help!

Or maybe the API just costs money. There are a few instances where we have no choice but to write a web scraper. So, let's write a web scraper. I'll assume you're doing this in python as there are several libraries that will make our task much easier. The libraries we're going to use are:

```
from urllib.request import urlopen
import requests
from bs4 import BeautifulSoup
```

I probably buy more super glue than the average person, so I'm going to be taking the price of super glue from Amazon.

First I need the URL (https://www.amazon.com/Gorilla-Ultimate-Fast-Setting-Cyanoacrylate-Anti-Clog/dp/B0D2RP4B82/ref=sr_1_3?sr=8-3)

In order to make the website think that I'm a person, we'll be sending in some headers. I found these headers online, so I'll be using them.

headers={"accept-language": "en-US,en;q=0.9","accept-encoding": "gzip, deflate, br","User-Agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.0.0 Safari/537.36","accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7"}

We then send a "get" request to get the raw html for the page.

```
resp = requests.get(url, headers=headers)
```

This is actually unreadable, so we need Beautiful Soup to make things look more readable.

```
soup=BeautifulSoup(resp.text,'html.parser')
```

Okay, we have a bunch of HTML. How is this useful?

It depends on what you're trying to get from the website. Let's say I'm trying to get the product name from amazon. I'm going to press ctrl + U to view the source code. We're going to look for the price. We also need to find the id. After sorting through a lot of html, I realize that I need this line:

<div id="titleSection" class="a-section a-spacing-none"> <h1 id="title" class="a-size-large a-spacing-none"> <span id="productTitle" class="a-size-large product-title-word-break"> Gorilla Super Glue Ultimate, Fast-Setting Cyanoacrylate Adhesive for Quick Fixes &amp; Repairs, 15g

Bottle with Anti-Clog Cap, Clear (Pack of 1) </span> </h1> <div id="expandTitleToggle" class="a-section a-spacing-none expand aok-hidden"></div> </div> </div>

From this line, the information I actually need is h1, and id="title"

This turns into this line of code:

```
o={}

try:
    o["title"]=soup.find('h1',{'id':'title'}).text.strip()
except:
    o["title"]=None
```

This will extract the information I need from the webpage and put it in my object.

I can repeat this process with the price

```
o["price"]=soup.find("span",{"class":"a-price"}).find("span").text
```

And these IDs are consistent across (almost) all products on Amazon. I could programmatically get the price of every product on Amazon if I really wanted to.