

Forum:[Hack Forums](#)

Thread: [Cannot find element Requests, Beautifulsoup Python](#)

roosiinfidel09 - May 29, 2020, 9:57 pm

I am trying to move to the next page till the 'next' button exists at this link 'https://www.cbp.gov/contact/find-broker-by-port/4901'. I realized that the requests response doesn't have the button in it hence BeautifulSoup cannot find it. I tried adding headers/user-agent to requests but the element still doesn't appear. As far as I can tell, there is no Javascript generating content on this page. Here is the code. What am I missing? The csv just reads the previous link to reach the page above.

```
Code def second_links(second_links_list=[], page2_num=0):
    try:
        with open('port.csv', 'r') as read_obj:
            csv_reader = reader(read_obj)
            for row in csv_reader:
                row = row[-1]
                page2 = requests.get(row.format(page2_num))
                soup2 = BeautifulSoup(page2.content, 'html')
                results2 = soup2.find(id='region-content')
                table2cells = results2.find_all('td', class_='views-field views-field-title views-align-center')
                for cell in table2cells:
                    cell2link = cell.find('a', href=True)
                    second_links_list.append('https://www.cbp.gov'+cell2link['href'])

                next2_page = results2.find('li', class_='pager-next')
                if next2_page:
                    page2_num += 1
                    second_links(second_links_list, page2_num)
            return second_links_list
    except requests.exceptions.ConnectionError:
        page2.status_code = 'connection refused'
```

vorticalbox - May 30, 2020, 1:29 am

Might that javascript it used to add the button dynamically in which case requests will never see it.

roosiinfidel09 - May 30, 2020, 1:47 am

(May 30, 2020 01:29 AM)vorticalbox Wrote: [CITE](#) Might that javascript it used to add the button dynamically in which case requests will never see it.

I don't think there is any script involved. I scraped the same links from the previous page.

Athr - May 30, 2020, 3:44 am

Try adding a return before calling the function.

From this:

```
Code if next2_page:
    page2_num += 1
    second_links(second_links_list, page2_num)
```

To this:

```
Code if next2_page:
    page2_num += 1
    return second_links(second_links_list, page2_num)
```

roosiinfidel09 - May 30, 2020, 5:21 am

(May 30, 2020 03:44 AM)Athr Wrote: [\[CITE\]](#) Try adding a return before calling the function.

From this:

```
Code if next2_page:
    page2_num += 1
    second_links(second_links_list, page2_num)
```

To this:

```
Code if next2_page:
    page2_num += 1
    return second_links(second_links_list, page2_num)
```

No, Man doesn't work. Doesn't find any element with this. The problem isn't the code..I am guessing it's the page. I have checked this line by line with pdb..I doesn't find the 'next' button. I also have a Chrome extension called "DataMiner"..even that is not able to click on next after selecting it..So, it looks like the issue is with the page.

(May 30, 2020 03:44 AM)Athr Wrote: [\[CITE\]](#) Try adding a return before calling the function.

From this:

```
Code if next2_page:
    page2_num += 1
    second_links(second_links_list, page2_num)
```

To this:

```
Code if next2_page:
    page2_num += 1
    return second_links(second_links_list, page2_num)
```

Maybe you are on the right track..My Indentation seemed off. For now, your modification doesn't give any errors...doesn't scrape as well though..stuck in the same loop..

synchronizing - May 30, 2020, 10:07 am

BS4 is not needed for this specific use. Notice that when you are on the first page, you have the following URL:

Code <https://www.cbp.gov/contact/find-broker-by-port/4901>

When you click "Next", notice how the URL changes to:

Code <https://www.cbp.gov/contact/find-broker-by-port/4901?page=1>

If you change the "page=1" to "page=0", you will get the 1st page:

Code <https://www.cbp.gov/contact/find-broker-by-port/4901?page=0>

Notice, however, if you go out of bounds you receive empty results:

Code <https://www.cbp.gov/contact/find-broker-by-port/4901?page=3>

What you are better off doing is using BS4 to check if the page is empty after reaching it via a "counter". Something as simple as:

```
Code from bs4 import BeautifulSoup
import requests
```

```
def get(page=0):
    url = f'https://www.cbp.gov/contact/find-broker-by-port/4901?page={page}'
    return requests.get(url).text

def get_info(data):
    broker_name = data.find("td", {"class": "views-field views-field-title views-align-center"})
    broker_filer_code = data.find("td", {"class": "views-field views-field-field-broker-filer-code
views-align-center"})
    broker_link = data.find('a', href=True)['href']

    return broker_name.text.strip(), broker_filer_code.text.strip(), broker_link
```

```
counter = 0
info = []
while True:
    soup = BeautifulSoup(get(page=counter), 'html.parser')
    odds = soup.find_all('tr', {"class": "odd"})
    evens = soup.find_all('tr', {"class": "even"})

    if not odds and not evens:
        break
    else:
        counter += 1

    for odd, even in zip(odds, evens):
        info.append(get_info(odd))
        info.append(get_info(even))
```

print(info)
The above will output:

```
Code [('AXIOM TRADE INC', 'BTL', '/contact/find-broker-by-port/4901/BTL'),
('DE LA CRUZ CUSTOMS BROKER INC', 'ENM', '/contact/find-broker-by-port/4901/ENM'),
('ECI CUSTOMS BROKERAGE INC', 'BGZ', '/contact/find-broker-by-port/4901/BGZ'),
('EDWIN SEDA PEREZ', '9JD', '/contact/find-broker-by-port/4901/9JD'),
('EXPEDITORS INT'L (PUERTO RICO) INC', 'ES9', '/contact/find-broker-by-port/4901/ES9'),
('GRISEL PADILLA', 'MU8', '/contact/find-broker-by-port/4901/MU8'),
('INTEGRITY CUSTOMS BROKERAGE LLC', '9QB', '/contact/find-broker-by-port/4901/9QB'),
('INTER-WORLD CUSTOMS BROKERS INC', 'N35', '/contact/find-broker-by-port/4901/N35'),
('JAIME MADURO SANTANA', 'ALA', '/contact/find-broker-by-port/4901/ALA'),
('JOSE G FLORES', '256', '/contact/find-broker-by-port/4901/256'),
('JOSE M RAMOS GARCIA', '97Q', '/contact/find-broker-by-port/4901/97Q'),
('JOSE R BERMUDEZ', '9HD', '/contact/find-broker-by-port/4901/9HD'),
('JUAN GARCIA', '9ST', '/contact/find-broker-by-port/4901/9ST'),
('JULIO CACERES DBA TRADEWORKS INC', '97D', '/contact/find-broker-by-port/4901/97D'),
('JULIO RODRIGUEZ USCB CORP', 'EWV', '/contact/find-broker-by-port/4901/EWV'),
('MANUEL A RAMOS', 'G68', '/contact/find-broker-by-port/4901/G68'),
('MANUEL RAMOS-GANDIA INC', 'CDX', '/contact/find-broker-by-port/4901/CDX'),
('NESTOR REYES INC', '508', '/contact/find-broker-by-port/4901/508'),
('NORBERTO DAVID COLON', 'BLC', '/contact/find-broker-by-port/4901/BLC'),
('P R INTERNATIONAL CUSTOMS BROKERS', 'D05', '/contact/find-broker-by-port/4901/D05'),
('PANALPINA INC', '554', '/contact/find-broker-by-port/4901/554'),
('PEDRO L CARMONA INC', 'BWV', '/contact/find-broker-by-port/4901/BWV'),
```

```
('PEDRO L SITIRICHE-TORRES', 'E9T', '/contact/find-broker-by-port/4901/E9T'),  
( 'RADIX GROUP INTERNATIONAL INC DBA DHL GLOBAL FORWARDING', '336',  
'/contact/find-broker-by-port/4901/336'),  
( 'RANK SHIPPING OF PUERTO RICO INC', 'D84', '/contact/find-broker-by-port/4901/D84'),  
( 'RENE ORTIZ-VILLAFANE INC', '438', '/contact/find-broker-by-port/4901/438'),  
( 'ROSA MARINA FLORES-ALVAREZ', 'NZ5', '/contact/find-broker-by-port/4901/NZ5'),  
( 'UPS SUPPLY CHAIN SOLUTIONS INC', 'UPS', '/contact/find-broker-by-port/4901/UPS')]
```

roosiinfidel09 - May 30, 2020, 10:49 am

I guess I should have put the whole thing in context. There are 220 links before the page in above code. Each one leads to a page like this. This page could have 3 pages like this one, 1 or 25 as well. Then I need to click each of these then scrape the addresses for each. Now, Here is the code which will give you an idea why your solution might be a little problematic when the need comes to click on each of the links scraped to get their addresses.

```
Code import pdb  
from csv import reader  
import requests  
from bs4 import BeautifulSoup  
page_num = 0  
  
url_cbp =  
r"https://www.cbp.gov/contact/find-broker-by-port?field_port_location_tid=All&field_port_code_value  
=&page={}"  
  
def get_links(links_total=[], page_num=0):  
    page = requests.get(url_cbp.format(page_num))  
    soup = BeautifulSoup(page.content, 'html.parser')  
    results = soup.find(id='region-content')  
  
    table_cells = results.find_all('td', class_='views-field')  
    for cell in table_cells:  
        # print(cell )  
        # print('\n\n')  
        cell_link = cell.find('a', href=True)  
        links_total.append('https://www.cbp.gov'+cell_link['href'])  
  
    next_page = results.find('li', class_='pager-next')  
  
    if next_page:  
        page_num += 1  
        get_links(links_total, page_num)  
  
    return links_total  
  
all_links = get_links()  
with open('Port.csv', 'w') as f:  
    for link in all_links:  
        f.write(link)  
        f.write('\n')
```

```

def second_links(second_links_list=[], page2_num=0):
    try:
        with open('port.csv', 'r') as read_obj:
            csv_reader = reader(read_obj)
            for row in csv_reader:
                row = row[-1]
                page2 = requests.get(row.format(page2_num))
                soup2 = BeautifulSoup(page2.content, 'html')
                results2 = soup2.find(id='region-content')
                table2cells = results2.find_all('td', class_='views-field views-field-title views-align-center')
                for cell in table2cells:
                    cell2link = cell.find('a', href=True)
                    second_links_list.append('https://www.cbp.gov'+cell2link['href'])

                next2_page = results2.find('li', class_='pager-next')
                if next2_page:
                    page2_num += 1
                    second_links(second_links_list, page2_num)
            return second_links_list
    except requests.exceptions.ConnectionError:
        page2.status_code = 'connection refused'

answer = second_links()
with open('Agents.csv', 'w') as fp:
    for ans in answer:
        fp.write(link)
        f.pwrite('\n')

```

synchronizing - May 30, 2020, 2:33 pm

(May 30, 2020 10:49 AM)roosiinfidel09 Wrote: [CITE](#) I guess I should have put the whole thing in context. There are 220 links before the page in above code. Each one leads to a page like this. This page could have 3 pages like this one, 1 or 25 as well. Then I need to click each of these then scrape the addresses for each. Now, Here is the code which will give you an idea why your solution might be a little problematic when the need comes to click on each of the links scraped to get their addresses.

...

I see. In such case, here are some good guidelines to follow when you scrape:
Be ambiguous, and don't write repetitive code. In other words, generalize as best as possible. We know the website generalizes the table content via the "class" variable (odd and even).

We also know that "?page=" works in any context within the website, so use that to your advantage.

If the code becomes >25 lines, then use OOP. Create objects as needed.

In this Covid afternoon I decided to take the time to show you what I mean. So check the code out. Notice how our starting point is the object "Main", so start there if you are trying to understand what is going on.

```
Code from bs4 import BeautifulSoup
import requests
```

```
class utils:
```

```
    @staticmethod
```

```
    def cbp_url(path):
```

```
        """ Combines path with cbp.gov url. """
```

```
        return f"https://www.cbp.gov{path}"
```

```
    @staticmethod
```

```
    def yield_info(path):
```

```
        """ Yields each individual row of the column inside the passed path. """
```

```
        url = utils.cbp_url(path)
```

```
        page = 0
```

```
        while True:
```

```
            page_url = f"{url}?page={page}"
```

```
            req = requests.get(page_url)
```

```
            soup = BeautifulSoup(req.text)
```

```
            odds = soup.find_all("tr", {"class": "odd"})
```

```
            evens = soup.find_all("tr", {"class": "even"})
```

```
            if not odds and not evens:
```

```
                raise StopIteration()
```

```
            column_info = []
```

```
            for odd, even in zip(odds, evens):
```

```
                column_info.append(odd)
```

```
                column_info.append(even)
```

```
            for info in column_info:
```

```
                yield info
```

```
class Broker:
```

```
    """ Class containing broker information. """
```

```
    def __init__(self, soup):
```

```
        self.soup = soup
```

```
        self.path = self.soup.find("a", href=True)["href"]
```

```
        self.url = utils.cbp_url(self.path)
```

```
        self.name = soup.find("td", {"class": "views-field-title"}).text.strip()
```

```
        self.address, self.phone = self.get_info()
```

```
        # Gathers code. Done this way for readability sake.
```

```
        code = soup.find("td", {"class": "views-field-field-broker-filer-code"})
```

```
        self.code = code.text.strip()
```

```

def get_info(self):
    """ Gathers further info about specific broker. """

    req = requests.get(self.url)
    soup = BeautifulSoup(req.text)

    address = soup.find("td", "views-field-field-location-address")
    phone = soup.find("td", "views-field-field-phone").text.strip()

    return address.text.strip().replace("\n", ", "), phone

```

```

def compile(self):
    return {
        "Name": self.name,
        "Code": self.code,
        "Address": self.address,
        "Phone": self.phone,
        "URL": self.url,
    }

```

```

def __repr__(self):
    return str(self)

```

```

def __str__(self):
    return f'{self.name}'

```

```

class Location:
    """ Class containing Location information. """

```

```

def __init__(self, soup):
    self.soup = soup

    self.path = self.soup.find("a", href=True)["href"]
    self.url = utils.cbp_url(self.path)
    self.name, id = self.soup.text.strip().split(" - ")

```

```

    if id.isdigit():
        self.id = int(id)
    else:
        self.id = id

```

```

    self.brokers = []
    for broker in utils.yield_info(self.path):
        self.brokers.append(Broker(broker))
    break

```

```

def compile(self):
    location = {"Name": self.name, "ID": self.id, "URL": self.url, "Brokers": {}}

    for broker in self.brokers:
        location["Brokers"][broker.name] = broker.compile()

```

```

    return location

def __repr__(self):
    return str(self)

def __str__(self):
    return f'{self.name}. {len(self.brokers)} brokers available.'

class Main:
    """ Main class. """

    def __init__(self):
        self.locations = []

        for location in utils.yield_info("/contact/find-broker-by-port"):
            self.locations.append(Location(location))
            break

    def compile(self):
        data = {}
        for location in self.locations:
            data[location.name] = location.compile()

        return data

    def __repr__(self):
        return str(self)

    def __str__(self):
        return f'Found {len(self.locations)} locations.'

```

With the code above, if you do:

```

Code main = Main()
print(main.compile())

```

It will output:

```

Code {
  "Aguadilla, Puerto Rico": {
    "Name": "Aguadilla, Puerto Rico",
    "ID": 4901,
    "URL": "https://www.cbp.gov/contact/find-broker-by-port/4901",
    "Brokers": {
      "AXIOM TRADE INC": {
        "Name": "AXIOM TRADE INC",
        "Code": "BTL",
        "Address": "MAI CENTER SUITE 210, SAN MIGUEL INDUSTRIAL PARK, 771 CALLE 1,
SUITE 210, SAN JUAN, PR 00920, United States",
        "Phone": "+1 (787) 706-0473",
        "URL": "https://www.cbp.gov/contact/find-broker-by-port/4901/BTL",
      }
    },
  },
}

```

```
}  
}
```

To make it so it scrapes every single page, then in the following two locations:

```
Code self.brokers = []  
for broker in utils.yield_info(self.path):  
    self.brokers.append(Broker(broker))  
    break  
and
```

```
Code for location in utils.yield_info("/contact/find-broker-by-port"):  
    self.locations.append(Location(location))  
    break
```

Make sure to remove the "break" statement. I purposely put those there so that I did not flood the website with requests. I want to make a few things obvious, so come back here when you've understood the code:

Notice how we split up the data via a pipeline that goes from Main -> Location -> Broker. This is important to keep in mind because when you scrape, you want to attempt to replicate the information flow as though it was you physically clicking the website. This will make your life 1000 times easier, since you can then "match" what is going on inside each page, with what is going on in your code.

Notice that by splitting up the data we have more control of the pipeline. By splitting up the data we can very easily edit the code above and say "get me the first 'n' results of 'y' location".

Pay attention how we generalized the data gathering. By creating a "utils" class, we've ensured that a lot of repetitive code was taken care of.

Notice the `__repr__` and `__str__` functions. If you don't know, read up on them - very easy tools to debug your code.

roosiinfidel09 - May 31, 2020, 2:31 am

Thanks a ton for taking out the time. I am not familiar with some of the concepts as you correctly guessed. Will make sure to read up on them. Thanks a lot again.

synchronizing - May 31, 2020, 8:58 am

(May 31, 2020 02:31 AM)roosiinfidel09 Wrote: [\[CITE\]](#) Thanks a ton for taking out the time. I am not familiar with some of the concepts as you correctly guessed. Will make sure to read up on them. Thanks a lot again.

Let me know what you need help understanding - would be happy to help.