Web Scraping the SEC Query Page

We've seen in other tutorials how to web scrape the EDGAR archives, but in this tutorial, we will explore a different part of the EDGAR database. The company search page allows us to make a specific query for a single company and their filings, and this page will then return all the documents related to that company. From here, we can filter all of their documents to the ones that meet our criteria.

This includes being able to filter by a specific date or even a particular type of form. Once, we've filtered the results we can go directly to the document or if we want we can go to the filing folder containing that document. One thing to keep in mind is the scope of your search. If you search for a company name, you can get back more than one company back.

This usually doesn't present a problem, but it does mean you may have to look through multiple companies to find the documents you want. It might make more sense to search by the CIK number, to get to the company you want.

Link to the company search page: https://www.sec.gov/edgar/searchedgar/companysearch.html (https://www.sec.gov/edgar/searchedgar/companysearch.html)

In [59]:
```
# import our libraries
import requests
import pandas as pd
from bs4 import BeautifulSoup
```

Section One: Define the Parameters of the Search

To create a search we need to "build" a URL that takes us to a valid results query, this requires taking our base endpoint and attaching on different parameters to help narrow down our search. I'll do my best to explain how each of these parameters works, but unfortunately, there is no formal documentation on this.

Endpoint The endpoint for our EDGAR query is https://www.sec.gov/cgi-bin/browse-edgar (https://www.sec.gov/cgi-bin/browse-edgar) if you go to this link without any additional parameters it will be an invalid request.

Parameters

- **action**: (required) By default should be set to `getcompany` .
- **CIK**: (required) Is the CIK number of the company you are searching.
- **type**: (optional) Allows filtering the type of form. For example, if set to 10-k only the 10-K filings are returned.
- **dateb**: (optional) Will only return the filings before a given date. The format is as follows YYYYMMDD
- **owner**: (required) Is set to `exclude` by default and specifies ownership. You may also set it to `include` and `only` .
- **start**: (optional) Is the starting index of the results. For example, if I have 100 results but want to start at 45 of 100, I would pass 45.
- **state**: (optional) The company's state.
- **filenum**: (optional) The filing number.
- **sic**: (optional) The company's SIC (Standard Industry Classification) identifier
- **output**: (optional) Defines returned data structure as either xml (`atom`) or normal html.
- **count**: (optional) The number of results you want to see with your request, the max is 100 and if not set it will default to 40.

Now that we understand all the parameters let's make a request by defining our endpoint, and then a dictionary of our parameters. Where the key of the dictionary is the parameter name, and the value is the value we want to set for that parameter. Once we've defined these two components we can make our request and parse the response using BeautifulSoup.

In [61]:
```
# base URL for the SEC EDGAR browser
endpoint = r"https://www.sec.gov/cgi-bin/browse-edgar"

# define our parameters dictionary
param_dict = {'action':'getcompany',
              'CIK':'1265107',
              'type':'10-k',
              'dateb':'20190101',
              'owner':'exclude',
              'start':'',
              'output':'',
              'count':'100'}

# request the url, and then parse the response.
response = requests.get(url = endpoint, params = param_dict)
soup = BeautifulSoup(response.content, 'html.parser')

# Let the user know it was successful.
print('Request Successful')
print(response.url)
```

```
Request Successful
https://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&CIK=1265107&type=1
0-k&dateb=20190101&owner=exclude&start=&output=&count=100
```

Side Note: Doing a Company Name Search

In the search defined up above, I assumed we wanted to search by a CIK number. If this is not the case, we will do a different search, a company search. A company search is a more broad search but is simpler because it requires fewer parameters. The only new parameter we have to pass through is the `company` parameter which has the company name as it's value.

In [5]:
```
# base URL for the SEC EDGAR browser
endpoint = r"https://www.sec.gov/cgi-bin/browse-edgar"

# define our parameters dictionary
param_dict = {'action':'getcompany',
              'owner':'exclude',
              'company':'Goldman Sachs'}

# request the url, and then parse the response.
response = requests.get(url = endpoint, params = param_dict)
soup = BeautifulSoup(response.content, 'html.parser')

# Let the user know it was successful.
print('Request Successful')
print(response.url)
```

```
Request Successful
https://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&owner=exclude&comp
any=Goldman+Sachs
```

Section Two: Parse the Response for the Document Details

Once we have our response code, we need to parse it. Our first goal is to find the `table` object that has a `class` attribute of `tableFile2` as this table contains the data related to the documents and the links. Once we grabbed the table, we will iterate through each row in the table parsing the columns. Unfortunately, things can get a little tricky as some documents can contain interactive data and others don't this means that if parse them for links it will return an error because no link exists.

To fix this, I wrap each link in an `if` statement that will only parse the `href` if one exists. The main links I'm looking for are the links to the document itself, the interactive data, and a link to filing folder containing that document. Keep in mind when I say filing folder what I mean is another EDGAR query but with an additional parameter called `filenum` which contains the filing number. Once, we parsed the necessary info, we store it in a dictionary and then store that dictionary in the `master_list` , this way we have a list of all the file names.

In [63]:
```
# find the document table with our data
doc_table = soup.find_all('table', class_='tableFile2')

# define a base url that will be used for link building.
base_url_sec = r"https://www.sec.gov"

master_list = []

# loop through each row in the table.
for row in doc_table[0].find_all('tr')[0:3]:

    # find all the columns
    cols = row.find_all('td')

    # if there are no columns move on to the next row.
    if len(cols) != 0:

        # grab the text
        filing_type = cols[0].text.strip()
        filing_date = cols[3].text.strip()
        filing_numb = cols[4].text.strip()

        # find the links
        filing_doc_href = cols[1].find('a', {'href':True, 'id':'documentsbutto
n'})
        filing_int_href = cols[1].find('a', {'href':True, 'id':'interactiveDat
aBtn'})
        filing_num_href = cols[4].find('a')

        # grab the the first href
        if filing_doc_href != None:
            filing_doc_link = base_url_sec + filing_doc_href['href']
        else:
            filing_doc_link = 'no link'

        # grab the second href
        if filing_int_href != None:
            filing_int_link = base_url_sec + filing_int_href['href']
        else:
            filing_int_link = 'no link'

        # grab the third href
        if filing_num_href != None:
            filing_num_link = base_url_sec + filing_num_href['href']
        else:
            filing_num_link = 'no link'

        # create and store data in the dictionary
        file_dict = {}
        file_dict['file_type'] = filing_type
        file_dict['file_number'] = filing_numb
        file_dict['file_date'] = filing_date
        file_dict['links'] = {}
        file_dict['links']['documents'] = filing_doc_link
        file_dict['links']['interactive_data'] = filing_int_link
        file_dict['links']['filing_number'] = filing_num_link
```

```
        # let the user know it's working
        print('-'*100)
        print("Filing Type: " + filing_type)
        print("Filing Date: " + filing_date)
        print("Filing Number: " + filing_numb)
        print("Document Link: " + filing_doc_link)
        print("Filing Number Link: " + filing_num_link)
        print("Interactive Data Link: " + filing_int_link)

        # append dictionary to master list
        master_list.append(file_dict)
```

```
----------------------------------------------------------------------------------------
----------------------
Filing Type: 10-K
Filing Date: 2018-03-07
Filing Number: 333-11002518671437
Document Link: https://www.sec.gov/Archives/edgar/data/1265107/000126510718000013/0001265107-18-000013-index.htm
Filing Number Link: https://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&filenum=333-110025&owner=exclude&count=100
Interactive Data Link: https://www.sec.gov/cgi-bin/viewer?action=view&cik=1265107&accession_number=0001265107-18-000013&xbrl_type=v
----------------------------------------------------------------------------------------
----------------------
Filing Type: 10-K
Filing Date: 2017-03-13
Filing Number: 333-11002517683575
Document Link: https://www.sec.gov/Archives/edgar/data/1265107/000126510717000007/0001265107-17-000007-index.htm
Filing Number Link: https://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&filenum=333-110025&owner=exclude&count=100
Interactive Data Link: https://www.sec.gov/cgi-bin/viewer?action=view&cik=1265107&accession_number=0001265107-17-000007&xbrl_type=v
```

Section Three: Parsing the Master List

We the `master_list` now populated we can iterate through the dictionary in the list and grab the values we want from each dictionary by passing the keys corresponding to that value. In the example below, I want all the links from a given dictionary, so I parse the links dictionary.

In [64]:
```
# loop through to get the links from the dictionary
for report in master_list[0:2]:

    print('-'*100)
    print(report['links']['documents'])
    print(report['links']['filing_number'])
    print(report['links']['interactive_data'])
```

```
----------------------------------------------------------------------------------------------------
https://www.sec.gov/Archives/edgar/data/1265107/000126510718000013/0001265107-18-000013-index.htm
https://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&filenum=333-110025&owner=exclude&count=100
https://www.sec.gov/cgi-bin/viewer?action=view&cik=1265107&accession_number=0001265107-18-000013&xbrl_type=v
----------------------------------------------------------------------------------------------------
https://www.sec.gov/Archives/edgar/data/1265107/000126510717000007/0001265107-17-000007-index.htm
https://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&filenum=333-110025&owner=exclude&count=100
https://www.sec.gov/cgi-bin/viewer?action=view&cik=1265107&accession_number=0001265107-17-000007&xbrl_type=v
```

Section Four: Parsing the XML version

We saw up above that if we set the `output` parameter to `atom`, that we get back an XML version of the same data, so let's explore how to request and parse the XML output. When we are defining the output parameter, we are accessing the RSS Feed that is linked with EDGAR. While the above example does work relatively easily, it probably makes more sense to use the RSS Feed as the data returned to us is more structured and therefore easier to parse.

The request will be identical except for the fact that we will change the `output` parameter to `atom` and change the parser to `lxml` .

In [7]:
```
# base URL for the SEC EDGAR browser
endpoint = r"https://www.sec.gov/cgi-bin/browse-edgar"

# define our parameters dictionary
param_dict = {'action':'getcompany',
              'CIK':'1265107',
              'type':'10-k',
              'dateb':'20190101',
              'owner':'exclude',
              'start':'',
              'output':'atom',
              'count':'100'}

# request the url, and then parse the response.
response = requests.get(url = endpoint, params = param_dict)
soup = BeautifulSoup(response.content, 'lxml')

# Let the user know it was successful.
print('Request Successful')
print(response.url)
```

```
Request Successful
https://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&CIK=1265107&type=1
0-k&dateb=20190101&owner=exclude&start=&output=atom&count=100
```

Once we have the content, we need to a search for all the `entry` tags as these tags contain the info related to the filings. Each entry tag has the following structure:

`

`

Please keep in mind that I have removed the actual info for readability.

In [56]:
```
# find all the entry tags
entries = soup.find_all('entry')

# initalize our list for storage
master_list_xml = []

# loop through each found entry, remember this is only the first two
for entry in entries[0:2]:

    # grab the accession number so we can create a key value
    accession_num = entry.find('accession-nunber').text

    # create a new dictionary
    entry_dict = {}
    entry_dict[accession_num] = {}

    # store the category info
    category_info = entry.find('category')
    entry_dict[accession_num]['category'] = {}
    entry_dict[accession_num]['category']['label'] = category_info['label']
    entry_dict[accession_num]['category']['scheme'] = category_info['scheme']
    entry_dict[accession_num]['category']['term'] =  category_info['term']

    # store the file info
    entry_dict[accession_num]['file_info'] = {}
    entry_dict[accession_num]['file_info']['act'] = entry.find('act').text
    entry_dict[accession_num]['file_info']['file_number'] = entry.find('file-n
umber').text
    entry_dict[accession_num]['file_info']['file_number_href'] = entry.find('f
ile-number-href').text
    entry_dict[accession_num]['file_info']['filing_date'] =  entry.find('filin
g-date').text
    entry_dict[accession_num]['file_info']['filing_href'] = entry.find('filing
-href').text
    entry_dict[accession_num]['file_info']['filing_type'] =  entry.find('filin
g-type').text
    entry_dict[accession_num]['file_info']['form_number'] =  entry.find('film-
number').text
    entry_dict[accession_num]['file_info']['form_name'] =  entry.find('form-na
me').text
    entry_dict[accession_num]['file_info']['file_size'] =  entry.find('size').
text

    # store extra info
    entry_dict[accession_num]['request_info'] = {}
    entry_dict[accession_num]['request_info']['link'] =  entry.find('link')['h
ref']
    entry_dict[accession_num]['request_info']['title'] =  entry.find('title').
text
    entry_dict[accession_num]['request_info']['last_updated'] =  entry.find('u
pdated').text

    # store in the master list
    master_list_xml.append(entry_dict)

    print('-'*100)
```

```
    print(entry.find('form-name').text)
    print(entry.find('file-number').text)
    print(entry.find('file-number-href').text)
    print(entry.find('link')['href'])
```

```
------------------------------------------------------------------------------
----------------------
Current report
333-110025
https://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&filenum=333-110025
&owner=exclude&count=100
https://www.sec.gov/Archives/edgar/data/1265107/000110465913087899/0001104659
-13-087899-index.htm
------------------------------------------------------------------------------
----------------------
Current report
333-110025
https://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&filenum=333-110025
&owner=exclude&count=100
https://www.sec.gov/Archives/edgar/data/1265107/000110465913085423/0001104659
-13-085423-index.htm
```

Now that we have all the entries stored in our dictionary let's grab the first item and see what the output looks like for the category section.

In [50]:
```
import pprint
pprint.pprint(master_list_xml[0]['0001265107-18-000013']['category'])
```

```
{'label': 'form type', 'scheme': 'https://www.sec.gov/', 'term': '10-K'}
```

Parsing the Next Page

In the example above our results were limited because we did such a narrow search, but it's not uncommon for more broad searches to return over 100 different entries. In these situations, we can leverage the XML output to find the link that takes us to the additional results. This process is easy; we merely find the `link` tag that has a `rel` attribute set to `next`. To demonstrate this, I've added a new URL that will return over 100 results.

In [55]:
```
# base URL for the SEC EDGAR browser
endpoint = r"https://www.sec.gov/cgi-bin/browse-edgar"

# define our parameters dictionary
param_dict = {'action':'getcompany',
              'CIK':'1265107',
              'dateb':'20190101',
              'owner':'exclude',
              'start':'',
              'output':'atom',
              'count':'100'}

# request the url, and then parse the response.
response = requests.get(url = endpoint, params = param_dict)
soup = BeautifulSoup(response.content, 'lxml')

# find the link that will take us to the next page
links = soup.find_all('link',{'rel':'next'})

# while there is still a next page
while soup.find_all('link',{'rel':'next'}) != []:

    # grab the link
    next_page_link = links[0]['href']

    print('-'*100)
    print(next_page_link)

    # request the next page
    response = requests.get(url = next_page_link)
    soup = BeautifulSoup(response.content, 'lxml')

    # see if there is a next link
    links = soup.find_all('link',{'rel':'next'})
```

```
----------------------------------------------------------------------------
----------------------
https://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&CIK=0001265107&typ
e=&datea=&dateb=20190101&owner=exclude&count=100&output=atom&start=100
```

Closing Remarks

The EDGAR query system allows us to quickly filter the companies we want to grab filings for and makes the process of finding the forms we need intuitive. With our knowledge of Python and the request system that EDGAR uses we can gain access to a tremendous amount of financial data that is free for public use.