Branch: master ▾ Find file Copy path

AM-Automated-Oppurtinity-Capture / README.md

wlhunter00 ok

794d437 12 minutes ago

1 contributor

Raw Blame History

435 lines (367 sloc) 28.3 KB

AM Automated Opportunity Capture

Introduction

This is the Automated Oppourtunity Hunter for Alvarez and Marsal's Forensic Technology Services team. The tool will go out to sites requested by the team, and will scrape information about various RFPs, RFIs, networking events, and various other opportunities that could result in more projects for the team. After this information is scraped, it is stored in a SQL database, and then a daily report on the tools findings will be emailed to the team.

Process

1. Scrape all RFP sites using the Python library BeautifulSoup.
2. Insert this data into a raw SQL table.
3. Scrape all event sites using their API through python.
4. Insert this data into a raw SQL table.
5. Clean the invidual tables using SQL queries.
6. Insert all indivdual sites into master table and current table.
7. Using Pandas take SQL queries and insert the data in dataframe.
8. Export the dataframes into excel sheets.
9. Attach the excel sheet and email the team the results using Yagmail and HMTL.

Table of Contents

Project Status

Version 2.2

Stable build is currently ready. Automated reports sent out daily. Functionality is currently being expanded on.

Recently Added

Created this README as a guide to the application.

To Do:

- [ ] Correctly spell Opportunity everywhere.
- [x] Create this README.
- [ ] Create Presentation.
- [ ] Look to implement dictionaries.
- [ ] Make sure there aren't uneeded loops in main script.
- [ ] Scrape 10Times.com using infinite scrolling.
- [ ] Use FBO.gov's API to scrape their RFPs.
- [ ] Decentralize the PATHs.
- [ ] Rename the job tables in SQL so that it is more clear that they are jobs (current and master).

Technologies

Project was created with:

- Python Version: 3.6
  - BeautifulSoup
  - Pyodbc
  - Pandas
  - Yagmail
  - Requests
- Microsoft SQL Server
- HTML

Installation

For this to fully work, the application needs to be run on a **MAGNUS** computer, as the inserts into the SQL database need to be accessed on the remote desktop.

First, you need to install Python 3.7.1 (you need to have a python version 3.0+). I use ATOM as my text editor, by any works.

Then you are going to set your PATH and `python -m pip install` in the command prompt the following libraries:

1. datetime
2. bs4
3. Pyodbc
4. Yagmail
5. Pandas
6. glob
7. os
8. requests

(Optionally) you can install Microsoft SQL Server to view the data yourself and to debug.

The most important files to have on your computer are the Python Master Function, Master SQL Query, Clean Raw SQL Query, and the SQL to Python Keywords files. Finally you will need the **Email Information** text file. This file is not found in the Github, as it contains sensative information.

Walkthroughs

Files Walkthrough:

- The first folders house the files used to develop the scraping algorithms for the indivdual sites (10Times, DASNY, GOVUK, NYSCR, RFPDB).
- The Excel Sheets folder houses the export sheets from this application.
- The Presentation Photos were simply used for screenshots.
- The Weekly Reports house the powerpoint presentations I have created throughout the internship.
- The folder Python Code houses all of the python scripts, the most important one being the Python Master Function.
- The folder SQL scripts houses all of the SQL server queries, the most important ones being the Master SQL Query, and the Clean Raw SQL Query.

Python Master Function Walkthrough:

- Starts with importing the libraries mentioned in Installation.

- Connects to SQL server.

- Creates lists to store information.
- `def mainFunction():` The main function of the script that will call all of the important functions and complete the process.
    - Parameters: None.
    - Returns: Nothing.
    - Called by: None

Scraping Main Functions

- `def searchAndUpload(container, labelHTML, resultHTML, titleHTML, labelDef, resultDef, titleDef, databaseName, jobNumber, pageNumber, site):` Will scrape all of the relevant information from a job, and then upload this information to an SQL raw table.
    - Walkthrough: Very case based. First you scrap the jobs information using the HTML and class definitions (or hard code scrape it). Then you loop through the whole list and insert this information into SQL. Then based on the sites and the cases you insert specific information into the SQL table.
    - Parameters: Takes in a job (container), the HTML and class of the label, result, and title information, the name of the table, the job number and website.
    - Returns: Nothing.
    - Called by `scrapeSite`
- `def scrapeSite(site, labelHTML, resultHMTL, labelDef, resultDef, containerHTML, containerDef, titleHTML, titleDef, numberOfPages, jobsPerPage):` Main function used for web scraping.
    - Walkthrough: First we get the needed information such as the database names, the starting number, and array of page numbers. Then for each page we get a list of jobs. Then for each job we scrape and upload.
    - Parameters: The website, HTML and class for the label, result, container, and the number of pages and jobs per page.
    - Returns: Nothing.
    - Called by `mainFunction`
- `def scrapeEventbrite():` Scrapes information from Eventbrites API and uploads it to SQL.
    - Walkthrough: First give the categories to scrape. Then for each category find the number of pages of events there are. Then loop

through each event and insert the given information into the table.
    - Parameters: Nothing.
    - Returns: Nothing.
    - Called by: `mainFunction`
- `def executeScriptsFromFile(filename):` Executes SQL query from a file. Used with `cleanRawSQL` and `masterSQLFunction`.
    - Walkthrough: Opens the SQL file, splits each command by find the ';', and then runs each command.
    - Parameters: The location of the SQL query we want to run.
    - Returns: Nothing.
    - Called by: `mainFunction`

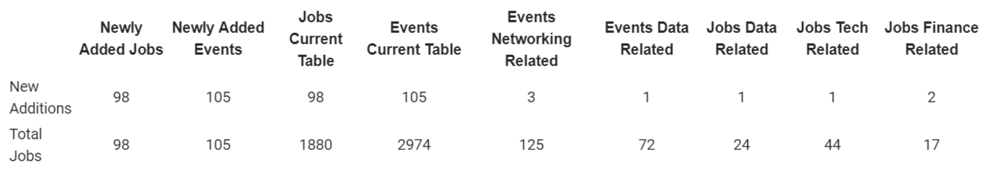
Scraping Helping Functions

- `def removeEscape(text):` removes the escape character for SQL inserts.
    - Parameters: Takes in string to parse.
    - Returns: Changed string.
    - Called by: `insertIntoSQL`, `scrapeEventbrite`
- `def parseASCII(text):` parses out non-ascii characters.
    - Parameters: Takes in string to parse.
    - Returns parsed string.
    - Called by: `insertIntoSQL`, `scrapeEventbrite`
- `def calculatePageNumber(numberOfPages, jobsPerPage, site):` Creates an array of page numbers for the url.
    - Parameters: Takes in the number of pages and the jobs per page, and the site.
    - Returns: An array of page numbers to parsed.
    - Called by: `scrapeSite`
- `def findLastJob(tableName):` Finds last job in tables.
    - Parameters: Table to look at.
    - Returns: ID of the last jobs.
    - Called by: `scrapeSite`
- `def getURL(site, startingNumber, category):` Gives the URL to scrape from. Has to be hardcoded.
    - Parameters: Site to scrape, the page number, and the category we are looking at.
    - Returns: URL to scrape.

        - Called by: `getContainers`
    - `def getContainers(site, startingNumber, HTMLobject, className, category):` Gives array of indivdual jobs to scrape from the given page
        - Parameters: All the information needed to retrieve the URL, and the class HTML and ID of the job object. See Adding a Site.
        - Returns: List of jobs to be scraped individually.
        - Called by: `scrapeSite`
    - `def getDatabase(site):` Gives the database names based on the sites.
        - Parameters: Name of the website.
        - Returns: List of the Database names, the raw and piviot tables.
        - Called by: `scrapeSite`
    - `def getScrapingCase(site):` Gives the scraping case for each website. Has to be hardcoded.
        - Parameters: Name of the site.
        - Returns: Name of the scraping case.
        - Called by: `searchAndUpload` , `getContainers`
    - `def getURLCase(site):` Gives the URL case for each website. Has to be hardcoded.
        - Parameters: Name of the site.
        - Returns: Name of the URL case.
        - Called by: `searchAndUpload`
    - `def listScrape(container, site, type):` The hardcoded method to scrape a site, when the information we are looking for is super specific. So far only used for RFPDB. Usually has to be called twice, once to return a list of labels and another time to return a list of results.
        - Parameters: The job we are looking at, the site, and if we are looking for labels or results
        - Returns: A list containing the given information to be stored.
        - Called by: `searchAndUpload`
    - `def insertIntoSQL(databaseName, jobNumber, label, result, site):` Inserts given information into SQL table.
        - Parameters: Table to insert into, job number, label of information, the information, and the website.
        - Returns: Nothing.
        - Called by: `searchAndUpload`

Exporting Main Functions

- `def queryToExcelSheet():` Will create an excel file from SQL queries.
  - Walkthrough: First we grab the SQL queries we want to use from the text file, then we load the data from the SQL tables into a dataframe, then we take this data and put it into two different excel spreadsheets, one is updating the master sheet and one is the daily excel file.
  - Parameters: None.
  - Returns: Nothing.
  - Called by: `mainFunction`
- `def sendEmail():` Sends email to the team using a premade gmail account. Attached is the excel file that was created.
  - Walkthrough: First we open up the email login file, then we find the latest created excel file, then we create the body of the email, the HTML variable is the table that is created, as shown below. This is all then loaded on an yagmail object and sent to all the email addresses we want.

	Newly Added Jobs	Newly Added Events	Jobs Current Table	Events Current Table	Events Networking Related	Events Data Related	Jobs Data Related	Jobs Tech Related	Jobs Finance Related
New Additions	98	105	98	105	3	1	1	1	2
Total Jobs	98	105	1880	2974	125	72	24	44	17

  - Parameters: None.
  - Returns: Nothing.

Exporting Helping Functions

- `def splitKeyWordFile():` Opens up the keyword file and fills the queries and sheets list.
  - Parameters: None
  - Returns: Technically nothing, but fills both the queries and sheets lists. The queries list specifies what data will be inserted into the excel sheet and the sheets list is the name of that sheet.
  - Called by: `queryToExcelSheet`
- `def loadDataFrames():` Loads list of dataframes that is made up of the list of queries
  - Parameters: Technically nothing, but uses list of queries from `splitKeyWordFile`
  - Returns: Technically nothing, but fills dataFrames list with the result from the select query.
  - Called by: ```queryToExcelSheet``

- `def loadCountingFrames():` For the keyword queries, loads list of ints for how many new records were added
  - Parameters: None.
  - Returns: Technically nothing but fills array with how many new records have been inserted.
  - Called by: `mainFunction`
- `def writeToExcel(writer):` Will write a dataframe into an excel spreadsheet. Loops through all of the queries.
  - Parameters: Takes in a writer, which is basically an excel document.
  - Returns: Nothing.
  - Called by: `queryToExcelSheet`

SQL Walkthrough:

There are two main SQL files, the master function which is what does most of the work in this process, and the cleaning query that is used to fix weird anomalies in the scraped data.

Clean Raw Queries

Very basic queries, this is just done as a measure to fix things that will upset the system. For instance when scraping `NYSCR` sometimes the Due Date is called `Due Date` but it is also sometimes called `End Date` so we want to change that to be uniform. We also sometimes will need to cut strings short or in half depending on the situation.

Master Function Query

The first thing done is to truncate each sites respective pivotTables, and then inserting the raw data into the pivot table like so:

```
insert into NYSCR_pvt
SELECT jobID, Website, [Title:], [Company:], [Category:], [dateInserted:],
[Due Date:], [Issue Date:], [Location:], [URL:], [Ad Type:]
FROM    (   SELECT A.jobID, resultText,  labelText, Website
            FROM NYSCR_raw A
        ) AS P
        PIVOT
        (   MAX(resultText)
            FOR labeltext in ([Title:], [Company:], [Category:],
[dateInserted:], [Due Date:], [Issue Date:], [Location:], [URL:], [Ad
Type:])
        ) AS  PVT
```

What is in the [ ] is the only thing different for each site, and this is what the labels are from each site.

We then insert this pivoted data into the master table with all of the site like so:

```
insert into master_table(JobDescription, Website, DueDate, IssueDate,
InsertDate, RequestType, JobURL, JobLocation, Category, Expired)
select [Title:], [Website], cast([Due Date:] as datetime),
cast(replace(replace( [Issue Date:], char(10), ''), char(13), '') as
date),cast([dateInserted:] as datetime), [Type:], [URL:], [Location:],
[Classification:], 'No'
from DASNY_pvt
WHERE DASNY_pvt.[Title:] not in(select JobDescription from master_table);
```

All we need to do here is align the labels from each sites pivot table to the master tables labels. We want to avoid inserting information already in the master table.

We then deal with the master and current table. We mark jobs as expired if they are done, we delete expired jobs from the current table, delete duplicates from the master table, and then set all of the jobs in the current table to old. We also cut off any job description that is too long so that our tables don't crash.

We then insert all of the current jobs into the current table using:

```
INSERT INTO current_table (JobDescription, Website, Company, DueDate,
IssueDate, InsertDate, RequestType, JobURL, JobLocation, Category, Status,
masterJobId)
SELECT distinct JobDescription, Website, Company, DueDate, IssueDate,
InsertDate, RequestType, JobURL, JobLocation, Category, 'New',
master_table.JobID
FROM master_table
WHERE master_table.JobDescription not in(select JobDescription from
current_table) and master_table.DueDate > GETDATE();
```

After this is done, we truncate the raw tables, and repeate the same process with the events.

Adding a Site

In order to sucessfully add a site to be scraped you need to:

1. Look at the site you want to scrape and figure out how it is formatted.

2. Identifiy all elements you want to scrape from this site.
3. Adapt the `Python Master Function` if need be so it can accommodate this new site.
4. Add a function in the `mainFunction` to scrape this site.
5. Create a raw SQL table in Microsoft SQL server for the site, make sure it has the same format as the others.
6. Create a pivot table for this site.
7. Merge the sites pivot table into the master table.

Looking at HTML

When looking at the site you want to scrape, first pull up the search page with as many results as possible. If you would like, you can add filters on the site, as long as these filters effect the url. In general most sites' HTML is formatted in a similar way. Each oppourtunity will be in its own element, and within that element there will be a class for the label, and then a class for the information to be displayed. You want to look for these elements in the site when scraping:

1. URL.
2. What defines an individual listing.
3. What defines the title.
4. What defines the label text.
5. What defines the result text.
6. Any other information you can scrape.
7. What makes this site different.
8. How many pages you will be scraping.
9. How many opportunities are on each page.

URL

This one is the easiest to understand. The more specific the url, the better.

`https://www.dasny.org/opportunities/rfps-bids?field_solicitation_classificatio_target_id=All&field_solicitation_type_target_id=All&field_goals_target_id=All&field_set_aside_target_id=All&query=&page=1`

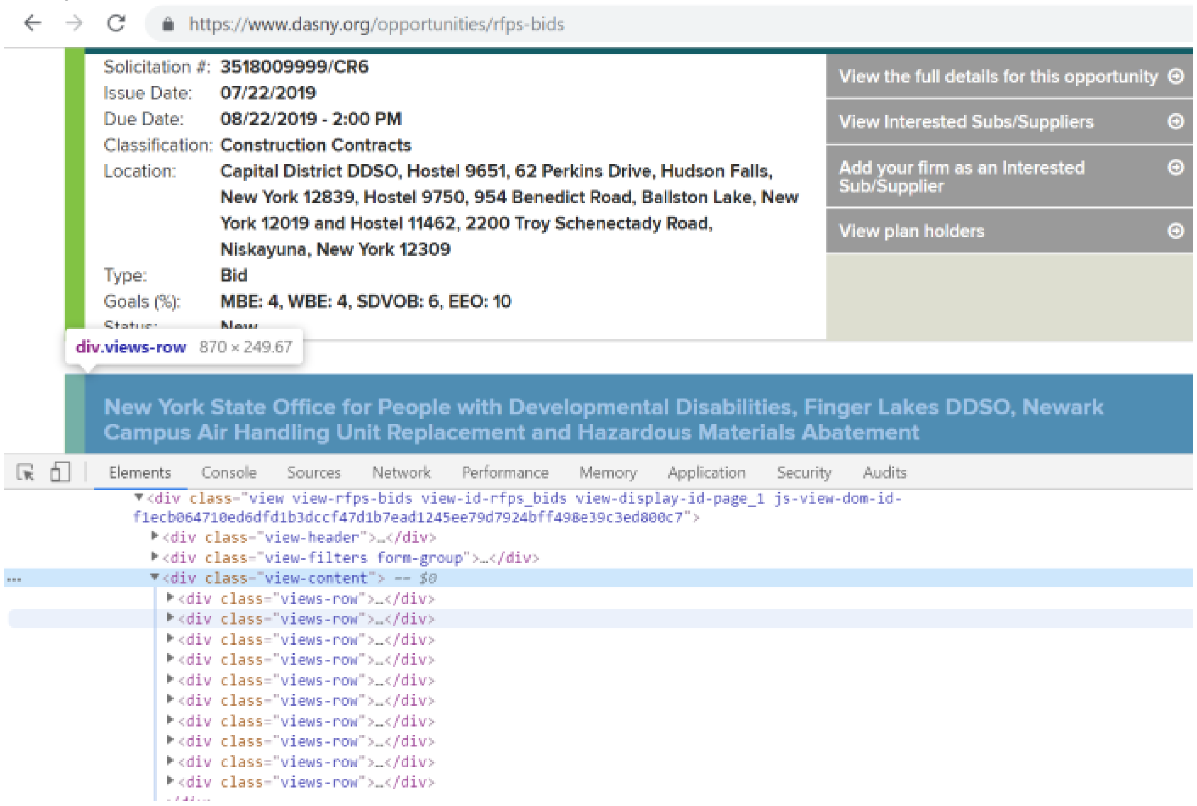
is much better than

`https://www.dasny.org/opportunities/rfps-bids` .

Also notice where there are variables that we want to manipulate. Usually this is pageNumber and category. Since the only thing we are going to manipulate is page number, we need to keep in mind where it is (`page=1` at the end).

Individual Listing

Each specific event or job is in what is called a 'container'. The best way to find this is to open up the website and inspect element in Google Chrome (`F12`). The job container is the farthest out object that encompasses all of the information for that specific job. See below for an example. Note the HTML object name (in this case `<div>`) and the class name (in this case `"views-row"`). This will be the same for every site.


https://www.dasny.org/opportunities/rfps-bids
View the full details for this opportunity
View Interested Subs/Suppliers
Add your firm as an Interested Sub/Supplier
View plan holders
div.views-row 870 × 249.67
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Audits

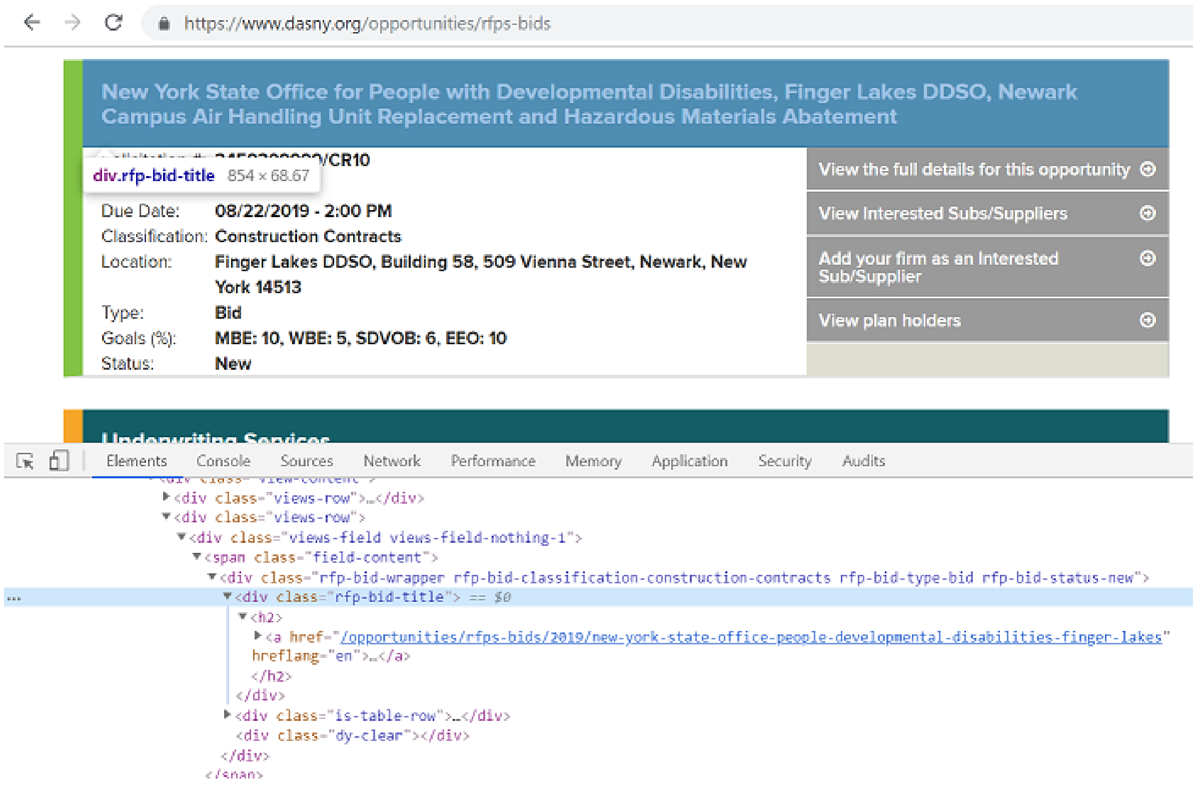
Solicitation #:	3518009999/CR6
Issue Date:	07/22/2019
Due Date:	08/22/2019 - 2:00 PM
Classification:	Construction Contracts
Location:	Capital District DDSO, Hostel 9651, 62 Perkins Drive, Hudson Falls, New York 12839, Hostel 9750, 954 Benedict Road, Ballston Lake, New York 12019 and Hostel 11462, 2200 Troy Schenectady Road, Niskayuna, New York 12309
Type:	Bid
Goals (%):	MBE: 4, WBE: 4, SDVOB: 6, EEO: 10
Status:	New

New York State Office for People with Developmental Disabilities, Finger Lakes DDSO, Newark Campus Air Handling Unit Replacement and Hazardous Materials Abatement

```
▼<div class="view view-rfps-bids view-id-rfps_bids view-display-id-page_1 js-view-dom-id-
f1ecb064710ed6dfd1b3dccf47d1b7ead1245ee79d7924bff498e39c3ed800c7">
  ▶<div class="view-header">…</div>
  ▶<div class="view-filters form-group">…</div>
  ▼<div class="view-content"> == $0
    ▶<div class="views-row">…</div>
    ▶<div class="views-row">…</div>
    ▶<div class="views-row">…</div>
    ▶<div class="views-row">…</div>
    ▶<div class="views-row">…</div>
    ▶<div class="views-row">…</div>
    ▶<div class="views-row">…</div>
    ▶<div class="views-row">…</div>
    ▶<div class="views-row">…</div>
    ▶<div class="views-row">…</div>
```

Titles

More often than not the title will have its own HTML element in the job container. Sometimes there will be even more information than just the title, such as the URL of the specific opportunity, so you can be crafty with how you scrape (like looking for a `<a>` tag). In this specific situation we want to note the HTML tag of the title (`<div>`), the class of the title (`rfp-bid-title`), and any other information (the url has a tag of `<a>`).


https://www.dasny.org/opportunities/rfps-bids
div.rfp-bid-title 854 × 68.67
View the full details for this opportunity
View Interested Subs/Suppliers
Add your firm as an Interested Sub/Supplier
View plan holders
Underwriting Services
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Audits

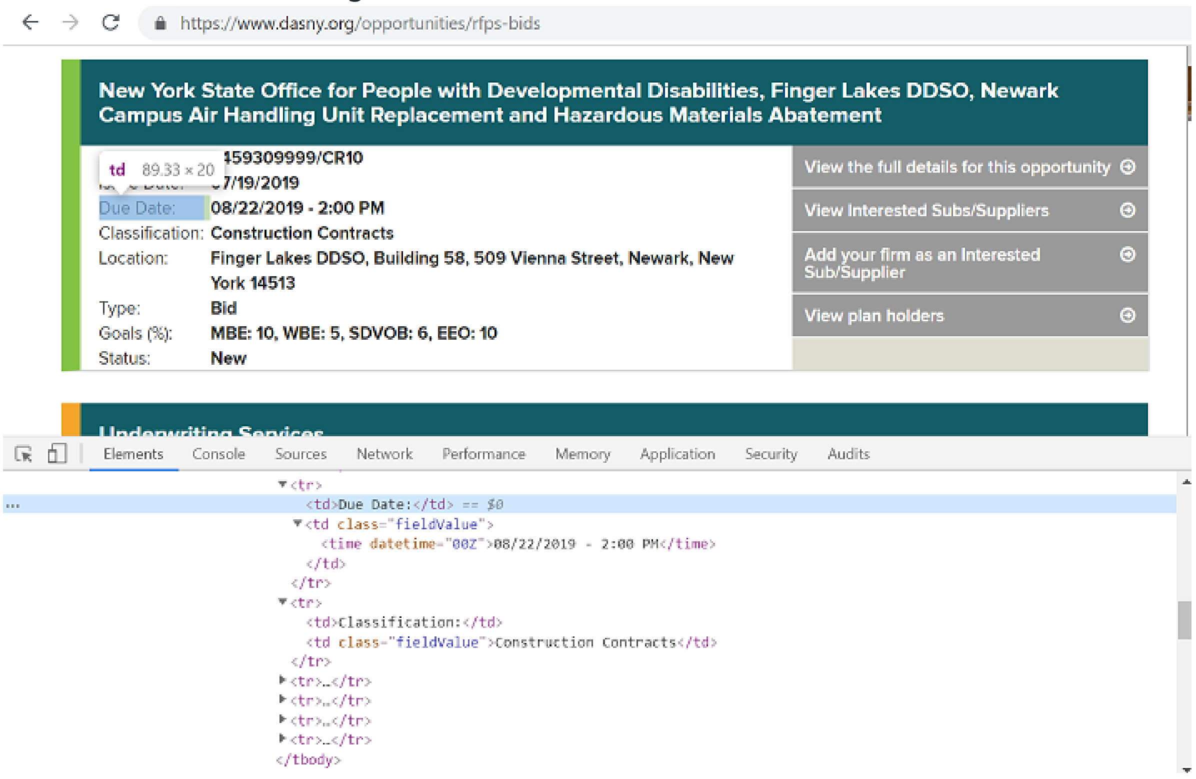
New York State Office for People with Developmental Disabilities, Finger Lakes DDSO, Newark Campus Air Handling Unit Replacement and Hazardous Materials Abatement

Due Date:	08/22/2019 - 2:00 PM
Classification:	Construction Contracts
Location:	Finger Lakes DDSO, Building 58, 509 Vienna Street, Newark, New York 14513
Type:	Bid
Goals (%):	MBE: 10, WBE: 5, SDVOB: 6, EEO: 10
Status:	New

```
▶<div class="views-row">…</div>
▼<div class="views-row">
  ▼<div class="views-field views-field-nothing-1">
    ▼<span class="field-content">
      ▼<div class="rfp-bid-wrapper rfp-bid-classification-construction-contracts rfp-bid-type-bid rfp-bid-status-new">
...     ▼<div class="rfp-bid-title"> == $0
          ▼<h2>
            ▶<a href="/opportunities/rfps-bids/2019/new-york-state-office-people-developmental-disabilities-finger-lakes"
            hreflang="en">…</a>
            </h2>
          </div>
        ▶<div class="is-table-row">…</div>
         <div class="dy-clear"></div>
        </div>
      </span>
```

Labels

Almost every site also has labels. I define labels as the text next to the important information. We want to scrape this so that we can use pivot tables later. The way we upload to SQL is by inputing the label and the text side by side, so its important that for every label we scrape it has a corresponding result text. Some example labels are **Due Date:**, **Status**, **Location**, and **Category**. Take note of every type of label you are scraping for later, as well as the HTML tag of the label (`<td>`) and the class of the label (nothing in this case).



https://www.dasny.org/opportunities/rfps-bids

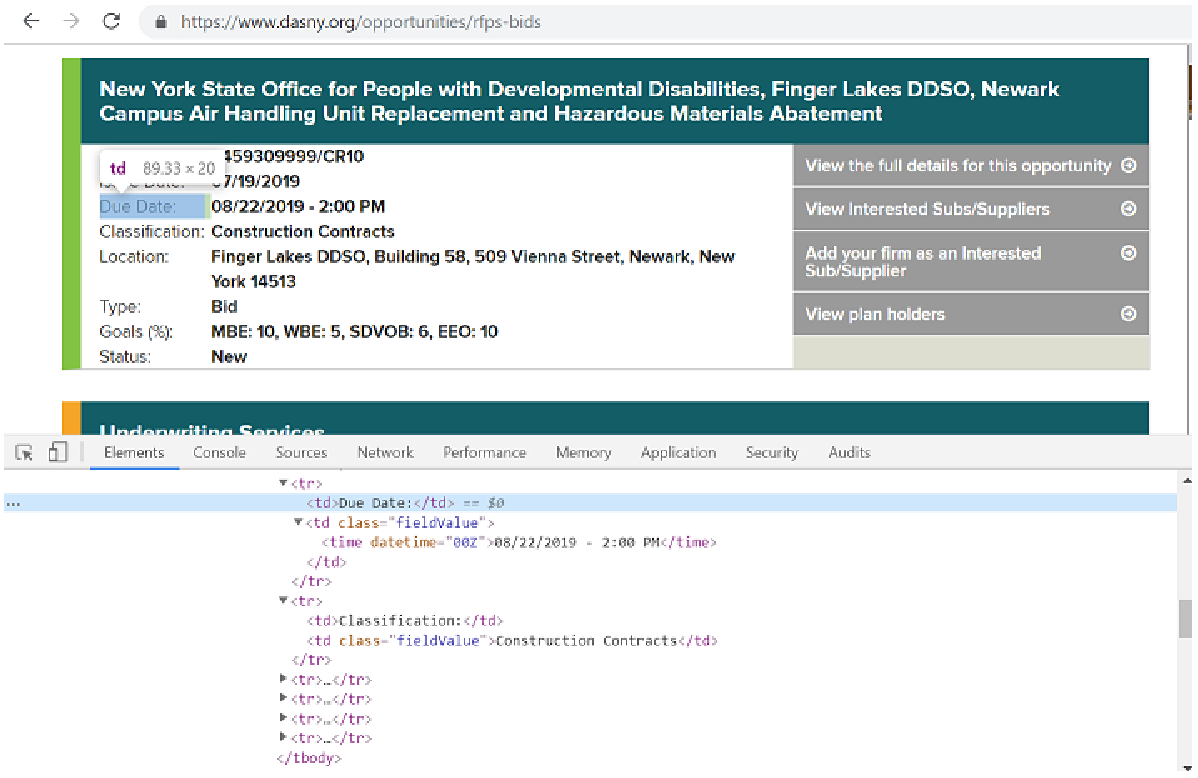
New York State Office for People with Developmental Disabilities, Finger Lakes DDSO, Newark Campus Air Handling Unit Replacement and Hazardous Materials Abatement

	459309999/CR10
	07/19/2019
Due Date:	08/22/2019 - 2:00 PM
Classification:	Construction Contracts
Location:	Finger Lakes DDSO, Building 58, 509 Vienna Street, Newark, New York 14513
Type:	Bid
Goals (%):	MBE: 10, WBE: 5, SDVOB: 6, EEO: 10
Status:	New

```
▼<tr>
   <td>Due Date:</td> == $0
  ▼<td class="fieldValue">
     <time datetime="00Z">08/22/2019 - 2:00 PM</time>
   </td>
 </tr>
▼<tr>
   <td>Classification:</td>
   <td class="fieldValue">Construction Contracts</td>
 </tr>
▶<tr>…</tr>
▶<tr>…</tr>
▶<tr>…</tr>
▶<tr>…</tr>
</tbody>
```

Results

Result text is what I call the actual information we are scrapping. Sometimes this information can be tied to the label which requires a little bit of hard coding. Some example results are **10/23/19**, **Active**, **New York**, and **RFP**. It is ok if the label and the result use the same HTML tag, just make sure you use classes. Its also important not to be too specific for the label text, because with BeautifulSoup you are able to grab just the text. For instance in the example I am about to provide, notice how I am not using the `<time>` tag. Take note of the HTML tag of the result (`<td>`) and the class of the result (`fieldValue`).


https://www.dasny.org/opportunities/rfps-bids
td 89.33 × 20
View the full details for this opportunity
View Interested Subs/Suppliers
Add your firm as an Interested Sub/Supplier
View plan holders
Underwriting Services
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Audits

New York State Office for People with Developmental Disabilities, Finger Lakes DDSO, Newark Campus Air Handling Unit Replacement and Hazardous Materials Abatement

	459309999/CR10
	07/19/2019
Due Date:	08/22/2019 - 2:00 PM
Classification:	Construction Contracts
Location:	Finger Lakes DDSO, Building 58, 509 Vienna Street, Newark, New York 14513
Type:	Bid
Goals (%):	MBE: 10, WBE: 5, SDVOB: 6, EEO: 10
Status:	New

```
▼<tr>
    <td>Due Date:</td> == $0
  ▼<td class="fieldValue">
      <time datetime="00Z">08/22/2019 - 2:00 PM</time>
    </td>
  </tr>
▼<tr>
    <td>Classification:</td>
    <td class="fieldValue">Construction Contracts</td>
  </tr>
►<tr>…</tr>
►<tr>…</tr>
►<tr>…</tr>
►<tr>…</tr>
</tbody>
```

Differences in Sites

Every site is different so it is important to note of differences from the norm. Here are a few differences in sites that I have run across, but every site is different

- Scraping their API is easier.
- Individual opportunities don't have a url.
- Labels are tied together with results.
- There are no labels.
- There is a parallax effect (infinite scrolling).
- You have to login to access important information.
- The way the page numbers are done in the URL is different.

Python Adjustments

The first thing you you want to do is add another function call in `mainFunction` . If there are multiple categories from that site you want to scrape, do something like I did with `RFPDB` . Here is the basis of the function call.

```
def scrapeSite(site, labelHTML, resultHMTL, labelDef, resultDef,
               containerHTML, containerDef, titleHTML, titleDef,
numberOfPages,
               jobsPerPage):
```

You want to insert the information that we found while Looking at the HTML into this. For more information, check out the walkthrough at Scraping Main Functions.

You are next going to want to put the URL of the site in `def getURL(site, startingNumber, category):` . The format should be something similar to this:

```
elif(site == 'RFPDB'):
    urlFromFunction =
'http://www.rfpdb.com/view/category/name/{0}/page/{1}'.format(category,
startingNumber)
```

You will then need to update `def getScrapingCase(site):` and `def getURLCase(site):` accordingly.

In a perfect world, you are now done with the Python part. Realistically, the site will have its own issues, and you will have to change this function based on what you need:

```
def searchAndUpload(container, labelHTML, resultHTML, titleHTML, labelDef,
                    resultDef, titleDef, databaseName, jobNumber,
pageNumber,
                    site):
```

Specifically, you will have to create a line regarding the URL similar to this:

```
elif(site == 'DASNY'):
    link = 'https://www.dasny.org{0}'.format(title.find('a')['href'])
```

More information about the side functions can be found here.

SQL Adjustments

First, you will need to create a Raw and Pivot table for the site. These should be called `site_raw` and `site_pvt` respectively. The create table query should look this so:

```
CREATE TABLE [DASNY_raw](
        [jobID] [int] NULL,
        [labelText] [nvarchar](max) NULL,
        [resultText] [nvarchar](max) NULL,
        [Website] [nvarchar](max) NULL
);
```

We are then going to create the pivot table. jobID, website, dateInserted, and URL: are constants for all sites, but the other columns will be in the `resultText` column from the raw table. An example is below:

```
CREATE TABLE [DASNY_pvt](
        [jobID] [int] NULL,
        [Website] [nvarchar](max) NULL,
        [Title:] [nvarchar](max) NULL,
        [Solicitation #:] [nvarchar](max) NULL,
        [Issue Date:] [nvarchar](max) NULL,
        [Due Date:] [nvarchar](max) NULL,
        [Classification:] [nvarchar](max) NULL,
        [Location:] [nvarchar](max) NULL,
        [Type:] [nvarchar](max) NULL,
        [Goals (%):] [nvarchar](max) NULL,
        [Status:] [nvarchar](max) NULL,
        [dateInserted:] [nvarchar](max) NULL,
        [URL:] [nvarchar](max) NULL
);
```

Then make any adjustments you need to `cleanRawSQL` if there is something wrong with the data or results.

After this you move on to `Master Function Query` . Add these two snipits to the code, and replace what is inside the [ ] with the column names you set before when you created the pivot table.

```
truncate table DASNY_pvt;



insert into DASNY_pvt
SELECT jobID, Website, [Title:], [Solicitation #:], [Issue Date:], [Due
Date:], [Classification:], [Location:], [Type:], [Goals (%):], [Status:],
[dateInserted:], [URL:]
FROM    (   SELECT A.jobID, resultText,  labelText, Website
            FROM DASNY_raw A
        ) AS P
        PIVOT
        (   MAX(resultText)
            FOR labeltext in ([Title:], [Solicitation #:], [Issue Date:],
[Due Date:], [Classification:], [Location:], [Type:], [Goals (%):],
[Status:], [dateInserted:], [URL:])
        ) AS  PVT;
```

And then at the end of the script, truncate the raw table like so:

```
truncate table DASNY_raw;
```

And thats it! If everything goes right these are all the steps you need to add a site to the process. Usually things will go wrong though, generally with the data that is being put into the pivot tables. Make sure you clean the data as much as possible.

Adding an Email

To add an email to recieve the daily report go to `Email Information.txt` file that is stored in the documents folder on the local computer (for security) and add an address to the list directly under the last email address.

Typical Errors

The easiest thing to do when debugging is look at the `mainFunction` in the `Python Master Function.py` . The program was designed to print outputs as it goes through the process, so you will be able to tell where exactly in the process it stopped working. I have divided the main function into four blocks, scraping, SQL, exporting, and sending the email. I reccomend commenting out all of the blocks but one, so that you can run them one at a time and and figure out what really broke. Generally, the most volatile chunk is the exporting to Excel.

Failures Adding a Site

Many things can go wrong when adding a site. It is very important that you analyze the website well. A good rule of thumb is to `select *` the raw table and see how things went. You may have to do a lot of cleaning of the data to put it in the `master_table`. A big thing that can mess with the program is dates, as many sites do these differently, so you may have to use substrings to rearrange the dates. I reccomend branching the `Python Master Function.py` when you add a new site, because more often than note things will go wrong. Usually logic will be broken in `searchAndUpload`.

No Email Sent

There are many reasons why an email wouldn't have been sent, but the best thing thing to do is to run the script yourself and see where it crashes. If it doesn't crash, then there is something wrong with the email or exporting to Excel chunks most likely.

Person Not Receiving Emails

If one person in particular is not receiving emails, there are two possible issues. First, check to see if they were correctly added to the mailing list (see Adding an Email).

Second, it could be an issue with A&M's blocker. Go to Mimecast, login to your A&M account, go to your mailbox and to the advanced tab, and then look for bounced or rejected emails. Alternatively, you will recieve an email from Mimecast saying this email was blocked from amopportunityhunter@gmail.com, and you will want to permit all emails coming from this sender.

No Jobs Added

If an email was sent, but there were no jobs added, then the exporting to Excel and emailing chunks were most likely ok. You are going to want to look at the scraping and the SQL functions. Usually if something is really wrong with the scraping function the script won't even run and that's how you know. If the script is running but no jobs are being shown, then something is wrong with the SQL queries, in particular with the interactions between the main and current tables most likely. Check to see if there is a weird insertion that is ruining things.