Company: Brāv Conflict Management
Project: Web Scraping Module

Developer: Jack Layfield

Overview and General Information

Environment: Virtual Python Environment, Eclipse IDE
File Type: Python executable (.py)

Purpose: To parse relevant websites looking for email addressed to send advertisements and/or information to potential customers.

All in one module: Module is responsible for setting up a client, performing searches, parsing and compiling emails, and finally sending emails to the proper addresses.

Functionality:

This module works through user inputted keywords. These keywords are words related to the kinds of markets you want to target. For example ‘conflict management’, ‘disputes’, etc.

The module then performs a search with an artificial client using Google search. The search is appended into the search expression for Google. The results page is expanded to the maximum 100 search results, and the html is fetched.

Once the Google page html is fetched, the module begins parsing that html looking for the html tags which contain the URLs for each website. Those URLs are then directly parsed. There are many errors that can occur with this parsing due to privacy and characters, but through many tests, all potential errors have been addressed.

Each website is requested for, and with sufficient permission, parsed, gathering any emails from the website by fetching each website's html and using expressions to properly search for emails within the html. These emails are stored in a set.

Finally, the email list is cleaned, by parsing out any “problem” emails such as one’s that contain the word “example” or “info” or “no-reply” that may be of no use to us.

Libraries

```
from bs4 import BeautifulSoup
import urllib
from urllib.parse import urlsplit
import requests
import sys
import re
import smtplib
from email.mime.multipart import MIMEMultipart
```

```
from email.mime.text import MIMEText
from email.mime.image import MIMEImage
from email.mime.base import MIMEBase
from email import encoders
```

These are all the libraries used. You can read more about them on their respective pages. Essentially, the three main libraries used are:

Beautiful Soup: For processing html and websites
Requests: For URL requests to specific websites
MIME: For email construction and sending

Creating an Executable:

In order to allow people to run the program without loading it up in an IDE, we need to create a general executable file.

To do this we use cx_freeze and the command window. It is very simple. Pip install cx_freeze through your command window. Navigate to the directory with your python file. Type this command on your .py still using the cmd window.

```
cxfreeze filename.py --target-dir dist
```

You should now have an executable in your directory that you can run. Run it like you would any other software.

Here is more direction and information on cx_freeze if you need it:
https://cx-freeze.readthedocs.io/en/latest/script.html

Future Implementations:

Eventually, the process shold be more automated, more accurate, and store data.

In terms of automation here is what should be done:

Machine learning to pick up on popular words among sites that contain useful emails. These keywords should be reintroduced to the program automatically, and the program should continue to run indefinitely.

In terms of accuracy:

Use machine learning principles to better filter websites and remember and learn what websites yield the best emails and most effective results.

Data storage:

Store emails and implement a blast feature to send multiple emails to recipients over certain periods of time.

Email Message:

In order to send a new message, or alter the message being sent you will need to edit the html present in the code starting at around line 160. This string html message will be interpreted and sent to users. Feel free to implement any html features within your message, images and videos included. Header information as well as subject and recipient is present above these lines of code. See important note 4 below for an important piece of information on using messages.

Important notes:

- Google can only display 100 results per page. If you want to visit results 101-infinity, you will need to set the start index higher, but always keep num<=100.

  URL = 'https://google.com/search?q={}&num={}**&start={}'**.format(search, iterations, start)

- Don’t mess with the exceptions or fetch_html function.
- Don’t mess with the UTF-8 encoding handling

- Look at around line 146 you should see this:

  with open('**/Users/Jack/Desktop/img1.png**', 'rb') as f:

  In order to send an image you must have the image on your computer and you must put your unique path to the image in place of mine which is bolded above. I’m sure this can be done by bundling files but this makes cx_freeze hard to use.