Apprenticeship Project: Webscraper

Webscraper project in Python by William O'Connell.

TOC

Overview

In this project I will be building a webscraper that rips the text from a website, stores the text as an object and then emails the text out to an email address list..

Problems to solve

1. Building the ‘scraper’ in python that rips the text from a site. Utilizing the correct python modules in order to achieve this

2. Utilizing the correct syntax to format the text into a more palatable, accessible and readable format.

3. Setting up an email account that allows me to access it from my python program. Scheduling the program to run at a given time.

4. Making the project in line with CodeNation guidelines to use as a project for the level 3 curriculum.

Project objective

My objective in this project is to build a webscraper that connects a local server to an email account and sends messages automatically.


Webscraping

Webscraping

01

I want to build a program that strips the text from a website. To do this I have researched ways to rip text from websites using Python. I came across a module named Beautifulsoup. https://www.tutorialspoint.com/beautiful_soup/index.htm

Beautiful Soup:
Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work..

Webscraping

02

Beautiful Soup takes the html or xml data froma website and converts it into a python object.

Here we set up Beautiful Soup by importing it at the top of the file along with ‘urllib’.

The website I have used as a tester is the Just For Today website which offers daily pragmatic and spiritual advice. The text changes every day, so it is a good website to use for a test to see if it works each day.

Webscraping

03

As you can see, when we print the soup object, it returns all the html in our terminal in string format.

It includes all the html tags and metadata, which is not very accessible so we will strip those out in our next steps, but it is good to the .read() function and the BeautifulSoup class working.

```
 8   with urllib.request.urlopen(url) as response:
 9       html = response.read()
10       soup = BeautifulSoup(html)
11
12   print(soup)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
, so I'm using the best available HTML parser for this system ("html5lib"). This usually isn't a problem, but if you run th
is code on another system, or in a different virtual environment, it may use a different parser and behave differently.

The code that caused this warning is on line 10 of the file c:\Users\willo\Documents\Coding\just_for_today\webscraper.py. T
o get rid of this warning, pass the additional argument 'features="html5lib"' to the BeautifulSoup constructor.

  soup = BeautifulSoup(html)
<html><head><meta content="text/html; charset=utf-8" http-equiv="Content-Type"/>

    <title>Just for Today Meditation</title>
    <meta content="width=device-width, initial-scale=1,user-scalable=yes" name="viewport"/>
    <meta content="-1" http-equiv="expires"/>
    <meta content="no-cache" http-equiv="Pragma"/>
    <meta content="no-cache" http-equiv="Cache-Control"/>
    <link href="smartphone.css" rel="stylesheet" type="text/css"/>
</head>
<body>
<table align="center">
    <tbody><tr>
        <td align="left"><h2>October 13, 2021</h2></td>
    </tr>
    <tr>
        <td align="center"><h1>Making a difference</h1></td>
    </tr>
    <tr>
        <td align="center">Page 299<br/><br/></td>
```

Webscraping

04

The next part I had to look up in the Beautiful Soup documentation - how to strip out the tags from the object itself. I discovered two methods:

1. The extract() method, which removes the tags and the data inside them.
2. The unwrap() method, which just removes the tags from the data, leaving the data behind.

```
13  for script in soup(["script", "style", "a", "<div id=\"bottom\" >", "title", "meta", "htmlhead", "link", "head"]):
14      script.extract()    # This rips out the tags and data inside the tags from the object
15
16  for tags in soup(["td", "h2", "table", "tr", "body", "tr", "h1", "br", "i", "b", "tbody", "html"]):
17      # This removes the tags from around the data inside, leaving just the text without tags.
18      tags.unwrap()
19
20  text = soup
21  print(text)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
        Sometimes it seems as though there is so much wrong with the world that we might as well forget trying to make a di
fference. "After all," we think, "what in the world can I do? I'm just one person." Whether our concerns are so broad that
we desire global peace or so personal that we simply want recovery made available to every addict who wants it, the task se
ems overwhelming. "So much work to do, so little time," we sigh, sometimes wondering how we'll ever do any good. Amazingly
enough, the smallest contributions can make the biggest difference. To gain more from life than an ordinary, plodding exist
ence requires very little effort on our parts. We ourselves are transformed by the deep satisfaction we experience when we
lift the spirits of just one person. When we smile at someone who is frowning, when we let someone in front of us on the fr
eeway, when we call a newcomer just to say we care, we enter the realm of the extraordinary. Want to change the world? Star
t with the addict sitting next to you tonight, and then imagine your act of kindness multiplied. One person at a time, each
 one of us makes a difference.


        Just for Today: An act of kindness costs me nothing, but is priceless to the recipient. I will be kind to someone t
oday.
```

Formatting the text

In this step I used some syntax in python to make the text object more readable, tripped away some of the whitespace, indentations and made sure things fit on lines correctly. I had to use the str() method to transform the object into a malleable string.

```
text = str(soup)

# # break into lines and remove leading and trailing space on each
lines = (line.strip() for line in text.splitlines())
# # break multi-headlines into a line each
chunks = (phrase.strip() for line in lines for phrase in line.split("  "))
# # drop blank lines
text = '\n'.join(chunk for chunk in chunks if chunk)
```

Formatting the text

After that the only thing left to was to refactor the code into a function so that it may be called and at a later time, I will implement a schedule so that it rips the text daily, creating a new text object from the data anew each day. Now I will be able to import the scrape function in the next part of the program.

```
9   def scrape():
10      with urllib.request.urlopen(url) as response:···
13
14      for script in soup(["script", "style", "a", "<div id=\"bottom\" >", "title", "meta", "htmlhead", "link", "head"])
16
17      for tags in soup(["td", "h2", "table", "tr", "body", "tr", "h1", "br", "i", "b", "tbody", "html"]):···
20
21      text = str(soup)
22
23      # # break into lines and remove leading and trailing space on each
24      lines = (line.strip() for line in text.splitlines())
25      # # break multi-headlines into a line each
26      chunks = (phrase.strip() for line in lines for phrase in line.split("  "))
27      # # drop blank lines
28      text = '\n'.join(chunk for chunk in chunks if chunk)
29
30      print("Scraping...")
31      return text
```

Tech used in Webscaper

The tech we have used in this first part of the program is quite simple. Aside from the new module Beautiful Soup, the other code is fairly simple straightforward for loops and dot notation methods.

01 | Beautiful Soup python module

02 | Urllib module

03 | For loops

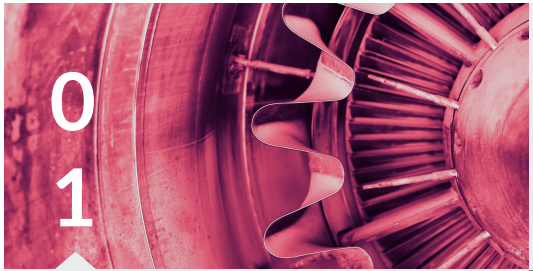
04 | Dot notation, join() method, strip() method etc

05 | Functions in python

Setting up the email functionality.

In the next step of the program I will detail how to set up an email account that allows access from a python program. We will use Gmail for this example.

Email Set Up



Keep a record of the login details

Also, I found that the inbox of my testing account rapidly filled up with test emails, which is reason enough to set up a new Gmail account for development.

Create a new Gmail account

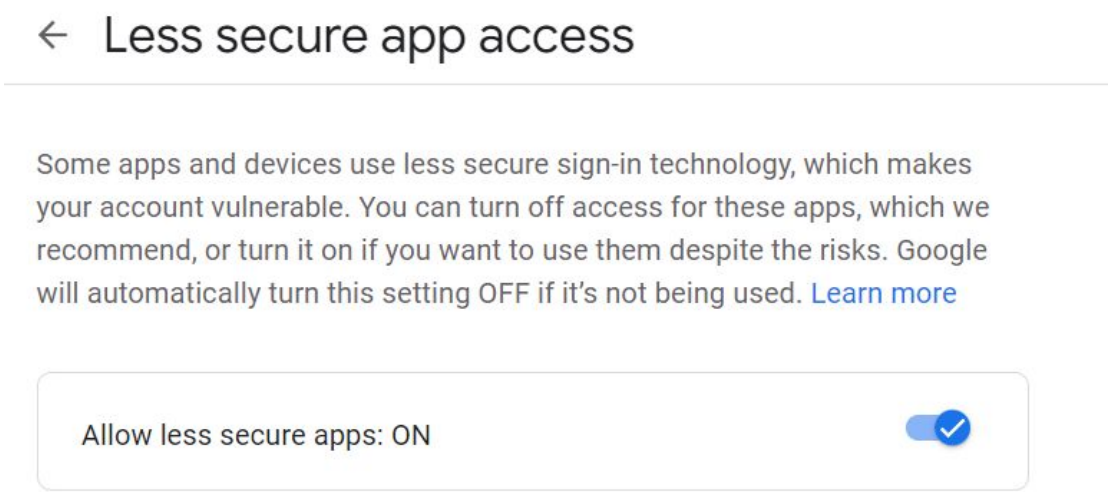
As you'll have to adjust your Gmail account's security settings to allow access from your Python code, create a throwaway account because there's a chance you might accidentally expose your login details.

Turn Allow less secure apps to ON

At : myaccount.google.com/lesssecureapps If you don't want to lower the security settings of your Gmail account, check out Google's documentation on how to gain access credentials for your Python script, using the OAuth2 authorization framework.

Less Secure Apps - ON

This is the google settings page for allowing access to less secure apps to your new Gmail account, we will need this option to be switched on in order to access it from our python program.



Setting up the program email functionality.

Now that our email account is set up and ready for access, let’s dive into the program and set up our email handler.

Emailer

01

We'll be using two libraries for this: email, and smtplib, as well as the MIMEMultipart object. This object has multiple subclasses; these subclasses will be used to build our email message.

1. MIMEText: It consists of simple text. This will be the body of the email.
2. MIMEImage: This would allow us to add images to our emails.
3. MIMEAudio: If we wish to add audio files, we may do it easily with the help of this subclass.
4. MIMEApplication: This can be used to add anything or any other attachments.

Emailer

02

We'll begin by creating a new file, and importing the following libraries/modules.

1. time and schedule for the future scheduling of our program.
2. email.mime… & smtplib for constructing the email itself.
3. Most importantly, we need the scrape() function from our webscrapaer file!

```
1  import schedule
2  import time
3
4  from email.mime.text import MIMEText                    # for text in email
5  from email.mime.image import MIMEImage                  # for images in email
6  from email.mime.multipart import MIMEMultipart
7  import smtplib
8
9  from webscraper import scrape
```

Emailer
03

This is what the entire mail() function should end up looking like.

Notice I have added a print("Sending") on line 25 in order to notify my when the function is running as per my scheduler.

```
12  def mail():
13
14      smtp = smtplib.SMTP('smtp.gmail.com', 587)
15      smtp.ehlo()
16      smtp.starttls()
17      smtp.login('justfortodaytester@gmail.com', '!!Password123')
18
19      msg = MIMEMultipart()
20      msg['Subject'] = "Just For Today"
21      msg.attach(MIMEText(scrape()))
22      to = ["williamjamesoc@gmail.com"]
23      smtp.sendmail(from_addr="justfortodaytester@gmail.com",
24                    to_addrs=to, msg=msg.as_string())
25      print("Sending...")
26      smtp.quit()
```

Emailer
04

```
smtp = smtplib.SMTP('smtp.gmail.com', 587)
smtp.ehlo()
smtp.starttls()
smtp.login('justfortodaytester@gmail.com', '!!Password123')
```

In the first half of the mail() function we set up a connection to our email server. We need to provide the server address and port number to initiate our SMTP connection.

Then we'll use smtp.ehlo to send an EHLO (Extended Hello) command. Now, we'll use smtp.starttls to enable transport layer security (TLS) encryption.

Emailer

05

Now, we simply build the message content.

We assign the MIMEMultipart object to the msg variable after initializing it.

The MIMEText function will be used to attach text. In our instance, we are using the imported scrape() function as our message content!

```
msg = MIMEMultipart()
msg['Subject'] = "Just For Today"
msg.attach(MIMEText(scrape()))
to = ["williamjamesoc@gmail.com"]
smtp.sendmail(from_addr="justfortodaytester@gmail.com",
              to_addrs=to, msg=msg.as_string())
print("Sending...")
smtp.quit()
```

Setting up the program email schedule.

Great, now we have a program that when run, will strip the html from a site, remove the unwanted markup, store the text as data, then email that data out. Next, let’s add automation.

Scheduling

01

Remember that we imported time and schedule at the beginning of this file.

The reason was for our scheduler, so that our program will run on it's own, ripping the text from the website fresh each day and emailing out that information each day as well.

We schedule scrape() before mail() in order to get fresh text, and the program will continue to run due to our while loop.

```
29  schedule.every().day.at("09:20").do(scrape)
30  schedule.every().day.at("09:20").do(mail)
31
32
33  while True:
34      schedule.run_pending()
35      time.sleep(5)
```

Tech used in Emailer

The tech we have used in this second part of the program is a little more complex. Beginning with time and schedule which are straightforward libraries, we then implement more complex libraries in email, smtp and MIME

01 | time, schedule python libraries

02 | smtplib module, ehlo, starttls and login methods

03 | MIME text and MIME multipart to construct the email

04 | while loop to run the program/automate scrape/send

05 | Functions, methods, and dot notation in python

Development errors and debugging.

In this next section I will show some development issues that surfaced, some common errors, and how I went about solving them.

Debugging

01

The first major bug I discovered was that whilst my webscraper worked, it was only ripping the text from the website the one time, then scheduling out the mail() function once a day. Originally I was importing the text variable (the data from the site) from webscraper.py and only scheduling the mail() send out.

It was on the second day, when I received the email through from the emailer section, but with yesterday's scrape I realised my error. I would have to put all of my webscraper code inside a function and import that function into the mail() function in emailer.py. This was a logical bug.

Debugging
02

Here you can see that on line 9 I import the scrape() function from webscraper.py, I think call the function on line 21 inside the MIMEText object parameter.

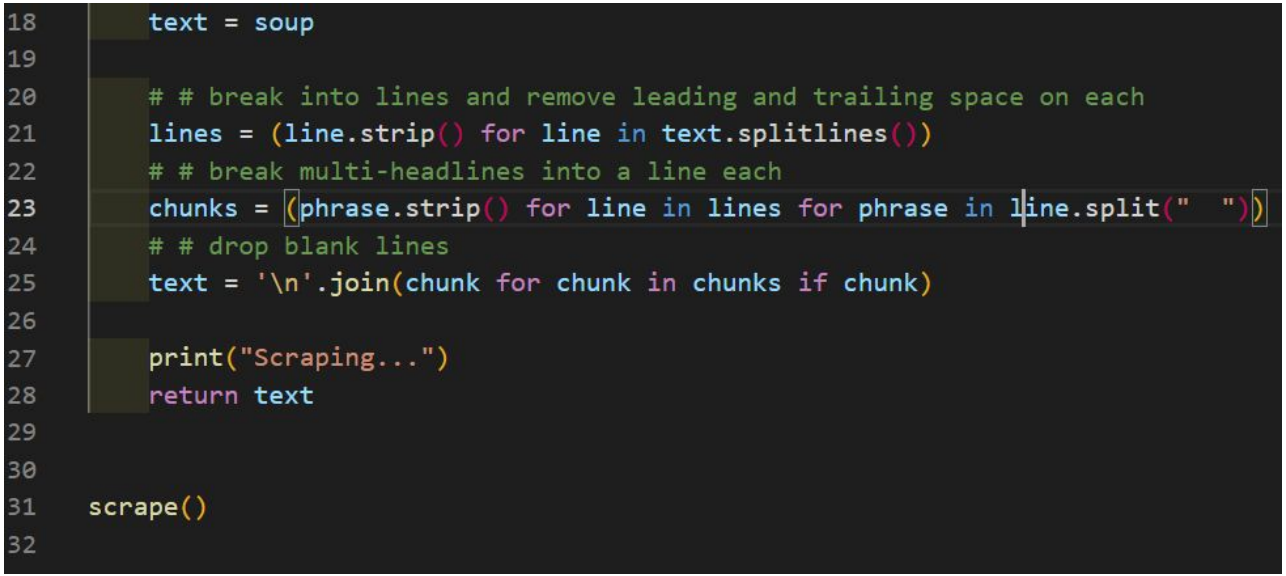
On line 29 you can see that I have scheduled the scrape() to happen just before the mail() schedule so that it rips a new text markup every day.

Bug fixed.

```
9   from webscraper import scrape
10
11
12  def mail():
13
14      smtp = smtplib.SMTP('smtp.gmail.com', 587)
15      smtp.ehlo()
16      smtp.starttls()
17      smtp.login('justfortodaytester@gmail.com', '!!Password123')
18
19      msg = MIMEMultipart()
20      msg['Subject'] = "Just For Today"
21      msg.attach(MIMEText(scrape()))
22      to = ["williamjamesoc@gmail.com"]
23      smtp.sendmail(from_addr="justfortodaytester@gmail.com",
24                    to_addrs=to, msg=msg.as_string())
25      print("Sending...")
26      smtp.quit()
27
28
29  schedule.every().day.at("09:20").do(scrape)
30  schedule.every().day.at("09:20").do(mail)
```

Debugging
03

Another development hurdle for me was the error i received from trying to use the strip method on this object. On line 18 you can see that I have attributed text to equal soup, but when I call the strip methods and join method on this object I received an error.



Debugging

04

This is the error I received when trying to call join and strip methods on the object. The object is not callable. To work around this issue I simply wrapped the ` text = soup` line in the str method, thereby transforming it into a malleable string

```
18      text = str(soup)
```

```
  File "c:\Users\willo\Documents\Coding\just_for_today\webscraper.py", line 31, in <module>
    scrape()
  File "c:\Users\willo\Documents\Coding\just_for_today\webscraper.py", line 21, in scrape
    lines = (line.strip() for line in text.splitlines())
TypeError: 'NoneType' object is not callable
PS C:\Users\willo\Documents\Coding\just_for_today>
```

Clarifying with CodeNation as a teaching tool.

The program runs perfectly without bugs. So now I will detail how this is to be used in python level 3 teaching.

CodeNation Teaching

01

This is the brief that I received from **Liam** at **CodeNation** for the project:

"Python Project for **Innovate: Coding**

A Python project suitable for level 3 learners, building on Python fundamentals and incorporating new theories that learners can build on.

NEW PROJECT - this is an opportunity to develop a new concept that we have not used before so innovative practices should be considered. Topics could include data science, machine learning, artificial intelligence etc.

Time frame: the first draft needs to be produced and tested before 04/11/21"

CodeNation Teaching

02

The specifications were broad enough to allow me to interpret them creatively, so I thought it would be a great idea to create a webscraper project in order to illustrate to students to scope and ability for python programs to reach, with under 100 lines of code.

Our learners will go through the project much in the same way I have detailed here. I will ask them to find a website which changes every day, and I will offer up some examples should they find this difficult. There are many websites that offer a thought for the day or poem for the day or something similar.

CodeNation Teaching

03

On the level 3 python course, the assumption is that they have already completed the Develop: Coding course, and therefore have the necessary skills in python for us to advance further. The fundamentals that we build upon in this project are:

- Functions
- Dot Notation
- Libraries and modules
- While loops and for loops
- Importing and exporting python files

CodeNation Teaching

04

As part of this project, the more advanced theory and concepts we are working with:

- SMTP library and relevant methods, such as transport layer security
- Third party libraries such as Beautiful Soup and the relevant methods
- Urllib response
- Schedule, using schedule and time libraries
- Connection using ports and server addresses

CodeNation Teaching

05

Some websites that offer fresh text each day:

- https://poems.com/todays-poem/
- https://www.allhappyquotes.com/thought-of-the-day/
- https://www.keepinspiring.me/inspirational-thoughts-for-the-day/
- https://www.wow4u.com/page12.html
- https://www.wow4u.com/quote-of-the-day/

Curriculum Adherence

01

As per the project brief, and in adherence to the level 3 python curriculum, this project will be used as a prime example not only to advance the understanding and build upon key foundations of python and programming itself, but also to look at coding standards, and best practices in line with the Unit 2 learning outcomes.

Unit 02 Understand coding design (A/618/0978)

Unit summary	In this unit, the learner will understand the purpose and use of coding standards. They will also learn about good coding principles and practices, and this will help them to understand the coding design process.
Guided learning hours	40
Level	3
Mandatory/optional	Mandatory

Learning outcome 1

The learner will:

1 Understand the use of coding standards

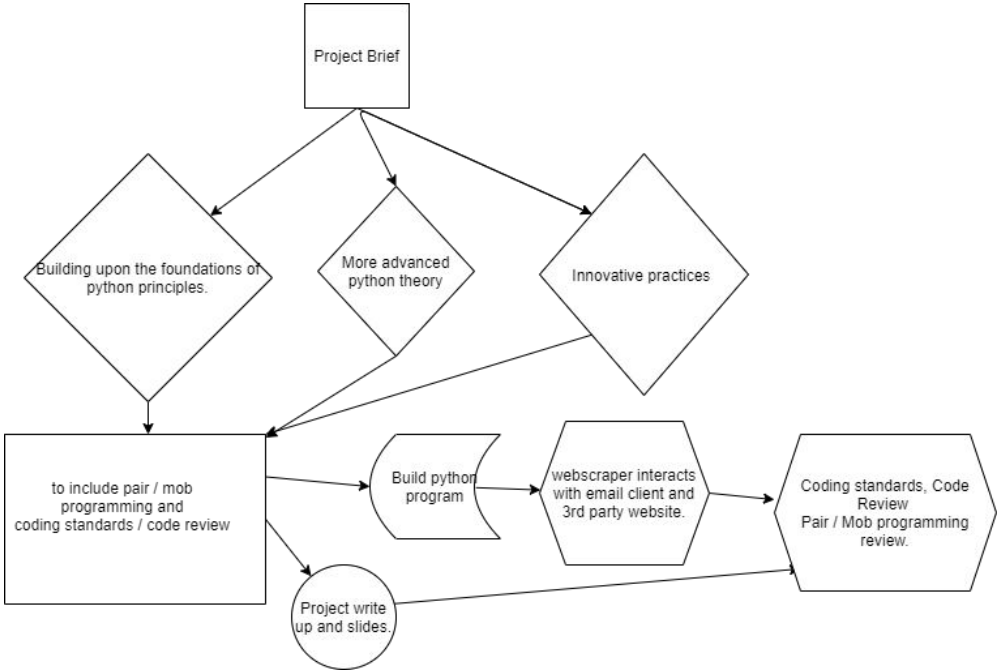
The learner can:

1.1 Identify the importance of coding standards
1.2 Explain the consequences of not following coding standards
1.3 Explain the processes and tools that could be used to promote coding standards

Curriculum Adherence

02

To the right I have included my flowchart which helped me visualise the project overview, not necessarily the coding project itself, but adhering to the brief, and to the curriculum for the level 3 documentation.



Thank you.