Creating A Simple News Article Web Scraper

Large datasets for modern data applications must be generated and compiled before analysis can be done. A growing popularity technique for gathering large sets of data from online sources is web scraping. This refers to using software to gather data over time from the websites a developer is interested in. Depending on the amount of data needed, and the type of application it is needed for, web scraping might need to be done over a long period of time. In the case of rate limiting, some scrapers must also be scheduled at specific times of the day and only run for specific amounts of time. This tutorial will walk through a simple web scraper for news articles, setting up a database to store article data, and setting up a scheduler to run the scraper at specified times. Let's begin.

1. Install Python

https://www.python.org/downloads/

Assuming you have a linux install, with apt:

```
$ sudo apt-get update
$ sudo apt-get install python3.6
```

Will download a python3 install. I am going to move forward assuming that python3.6 is the installation you are using. If you prefer python2, make sure you check the newspaper package for the correct install.

2. Install Postgresql

Postgresql is a database software that is quick to set up to store data. This is how we will save data that the scraper finds.

Install:

```
$ sudo apt-get install postgresql
```

3. Get the python packages

Install the required python packages using pip. Pip is a command line tool built-in with current python releases to make installing python packages easier. The help menu for pip can be accessed using

```
$ pip -H
```

The following packages are needed:

Newspaper3k (or newspaper for python 2), which is a library which will make scraping news data easy:

```
$ pip install newspaper3k
```

SQLAlchemy is a package designed to make it easy to interface with our postgres database.

```
$ pip install SQLAlchemy
```

APScheduler is a package that helps schedule events to happen regularly, such as scraping a website every day.

```
$ pip install apscheduler
```

4. Create Simple Web Scraper

Now that we have installed everything, the next step is to create a python file that can scrape news article data. The newspaper package makes this very easy.

Here is the sample code for scraper.py:

```
import newspaper

url = 'https://news.google.com/'
paper = newspaper.build(url)

for article in paper.articles:
      if article.title is not None:
            print(article.title)
```

First, we define a url that we wish to scrape.

Next, newspaper.build takes in that url and generates a newspaper object. A newspaper object is generated starting at the url we gave, and creates a collection of articles.

The last few lines print out the title of all articles in our paper.articles list. The reason for the if statement that checks for the presence of a title is because scraping can sometimes be messy, and not everything that is scraped has everything.

You can run the program using:

```
$ python scraper.py
```

5. Make postgres table for data

In order to connect to our postgresql server, we need to login as user postgres. Linux is a bit strange with the conventions for doing this, so assuming you have sudo access, you can login as user postgres with the following command:

```
$sudo -u postgres psql
```

Logging into postgres should result in a postgres server command prompt opening up, like so:

```
psql (9.5.8)
Type "help" for help.

postgres=#
```

Once logged in to postgres, the following commands will create the database and table we need:

```
CREATE DATABASE articles;

\c articles;

CREATE TABLE article_text( title varchar, body varchar );
```

First we create our article database.

Next, \c articles will connect us to the database we just made so that we can create a table in that database.

Finally, the create table statement will create a table that has columns for title and body text.

6. Insert from Scraper to Database

Let's edit the previous scraper.py file to make it insert into our new table in the database:

```
import newspaper
import sqlalchemy
from sqlalchemy import create_engine

host = 'localhost'
dbname = 'articles'
user = 'postgres'

def scrape():
        try:
                # create sqlalchemy engine for db connection
                eng = create_engine('postgresql://' + user + '@' + host +
                        '/' + dbname)
                # get meta data describing database
                meta = sqlalchemy.MetaData(bind=eng, reflect=True)
                # build an object representing the table we want to
interact with
                table = meta.tables['article_text']
        except Exception as e:
                print(e)

        url = 'https://news.google.com'
        paper = newspaper.build(url)

        for article in paper.articles:
                if article.title is not None:
                        # download html from tree of article URLS
                        article.download()
                        # parse downloaded html to readable text
                        article.parse()
                        # clear newlines out of body text
                        body = article.text.replace('\n', ' ')
                        # insert into database
                        insert = table.insert().values(title=article.title,
body=body)
                        eng.execute(insert)
```

And then we will create a main.py which calls our scrape function we defined in scraper.py:

```
from scraper import scrape

if __name__ == '__main__':

        scrape()
```

This seems redundant to wrap our scraper up into this other file, but it will make scheduling the scraper much easier when we get the next step.

To run and verify your scraper runs as designed, simply run:

```
$ python main.py
```

It is recommended that you put print statements in the code so that you can easily verify that the scraper runs without crashing and executes as expected. Once you believe your scraper is working, you can verify that inserts are being made to the database by doing a

```
select * from article_text;
```

from postgres, after connecting to the database again. You should be able to see some article titles and corresponding article body text inside your database. If not, verify the connection info and check your python files for typos. Once your scraper correctly scrapes data and pushes it to the database, we are ready to schedule our scraper to run every night.

7. Schedule Scraper

If we change up the structure of main.py to be the following:

```
from scraper import scrape
from apscheduler.schedulers.background import BackgroundScheduler as Scheduler
import datetime

if __name__ == '__main__':

        scrape()
        s = Scheduler()

        # s.add_job(scrape, 'cron', hour=12, minute=30)
        s.add_job(scrape, 'date')
        s.start()
```

This will schedule a job immediately, using the scrape() function from our scaper as the job that will be executed. Above that, in comments, the code to schedule a job that reoccurs every time the date matches the fields specified. So in this example, the scheduler will run the job every time the hour matches 12 and the minute matches 30, every day. Jobs can be customized based on your preferences, for every day, every Monday, etc.

See here for more information about job scheduling:

http://apscheduler.readthedocs.io/en/latest/modules/triggers/cron.html

Once you have specified the details for job scheduling, we can schedule the scraper by running:

```
$ python main.py
```

Scheduling for a cron job will automatically handle scheduling that job more than once. If you plan on scraping for an extended period, the scheduler can handle running scheduled jobs indefinitely, as long as the python process remains alive.