# Advanced Scripting Project Documentation

## Kevin O’ Sullivan

## APIs Used

* Scrapy 1.3.3
* Matplotlib 2.0.0

## Development and Test Environment

OS: Windows 10 x64

IDE: PyCharm Community Editon 2017.1.1

Python Interpreter: Python 3.6.0 (installed with Anaconda3 installation)

For the bash scripts in this program to work, Cygwin needs to be installed also, and the ‘C:\cygwin64\bin’ directory needs to be added to the Windows Path environment variable.

## Program Function

This program will scrape Euro exchange rates from the x-rates.com website and store them in .csv files. It will scrape this page 6 times with 1 minute intervals between each scraping attempt. The Euro-US Dollar and Euro-British Pound exchange rates will then be read from each .csv file and plotted onto 2 individual line charts. This will visually demonstrate the changing of the 2 exchange rates over a period of time. Realistically, monitoring the variance of exchange rates over a period as short as 5 minutes isn’t very useful, but is necessary to demonstrate the program’s functionality. It would be more useful to monitor over a period of say, 5 days.

The page being scraped is located here: <http://x-rates.com/table/?from=EUR&amount=1>

## How to run the program

The ‘currency\_scraper.py’ file has to be executed to run the program.

## Important Files

To create a Scrapy project, the command ‘scrapy startproject *projectname’* has to be entered (for this project ‘exchange\_rates’ was entered as the project name). When I entered this command it created the directories and files shown below:

exchange\_rates/

scrapy.cfg

exchange\_rates/

\_\_init\_\_.py

items.py

middlewares.py

pipelines.py

settings.py

spiders/

\_\_init\_\_.py

I did not use all of these files for this project. The files I did use and the files I created from scratch are highlighted below:

exchange\_rates/

scrapy.cfg

currency\_scraper.py

feed\_export\_script.sh

cleanup.sh

scraped\_data\_files/

adding\_directory.txt

exchange\_rates/

\_\_init\_\_.py

items.py

middlewares.py

pipelines.py

settings.py

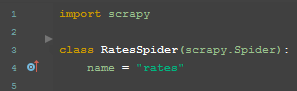
spiders/

\_\_init\_\_.py

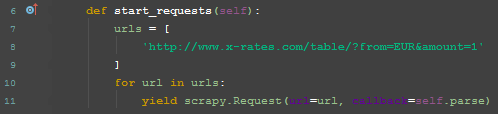
rates\_spider.py

The function of the code in each of these files will be explained below.

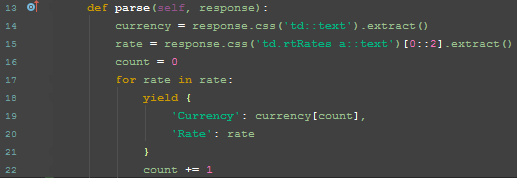
## rate\_spider.py



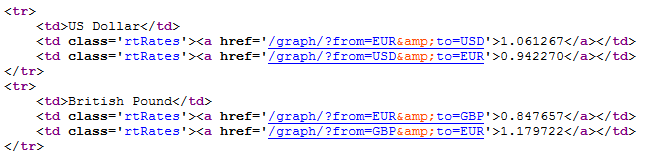
rate\_spider.py contains the scrapy spider that will scrape x-rates.com for the exchange\_rates. On line 3, we can see the spider is a subclass of scrapy.Spider called RatesSpider, which contains two methods, start\_requests() and parse(). On line 4 the name attribute is given a unique value of ‘rates’. This attribute is referenced in ‘feed\_export\_script.sh’ when running the spider.



The start\_requests() method returns an iterable of requests which the spider crawls from. The url of the web page that will be crawled is in a list called urls (lines 7, 8, 9). Lines 10 and 11 contain a for loop which loops through the list of urls (in this case only containing 1 url) and uses the yield keyword to return a html response from each url which can be parsed using the parse() method.

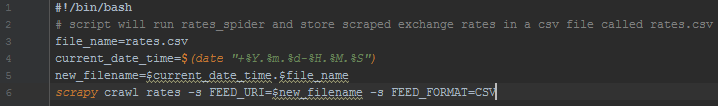


In the parse() method, on line 14, the text stored inside <td> tags is extracted and stored in the currency list. On line 15, every second piece of text stored inside the <a> tags, inside the <td> tags in the class ‘rtRates’ is extracted and stored in the rates list. So as shown in the page source below, ‘US Dollar’ and ‘British Pound’ would be stored in the currency list, while ‘1.061267’ and ‘0.847657’ would be stored in the rates list.



On lines 16 to 22, the for loop returns the scraped currencies and their exchange rates.

## feed\_export\_script.sh



feed\_export\_script.sh is a bash script which gets run by the ‘currency\_scraper.py’ file. It runs the spider which scrapes the data from the webpage. Lines 3 to 5 are used to construct the ‘new\_filename’ string variable which contains the current time and date and ‘rates.csv’. Line 6 then runs the RatesSpider scrapy spider (which is called using the name attribute ‘rates’). Line 6 also contains a feed export, which stores the scraped data in a .csv file named using the value of the ‘new\_filename’ variable.

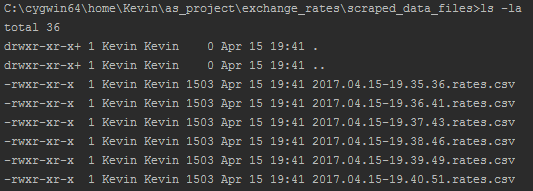
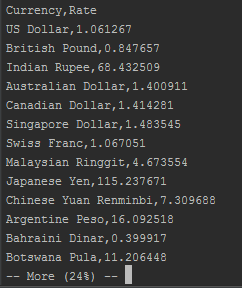
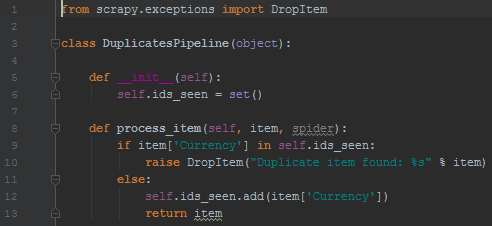


Image above shows some .csv files containing scraped data.



This image shows the contents of one of the .csv files.

## pipelines.py



pipelines.py contains an item pipeline called DuplicatesPipeline. In scrapy an item pipeline is a class that has scraped data sent to it and performs an action on it. In this case, the method process\_item checks if the data it is processing has already been scraped and drops it if it has, which means duplicate items are dropped.

## settings.py

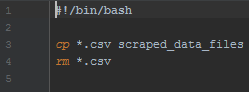


The code above was added to the settings.py file to ensure that the DuplicatesPipeline would be used.

## scraped\_data\_files

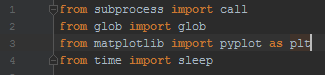
The scraped\_data\_files directory was created to store the .csv files containing scraped data. The ‘adding\_directory.txt’ file inside it has no function and can be ignored.

## cleanup.sh

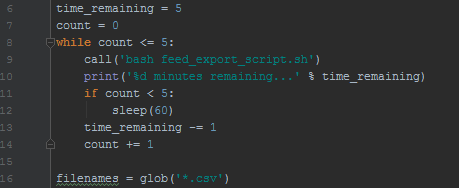


This bash script is called at the end of ‘currency\_scraper.py’. It copies the .csv files containing scraped data into the scraped\_data\_files directory and then deletes them from the exchange\_rates directory. If those files were not removed then every time ‘currency\_scraper.py’ was run there would be more exchange rates plotted on the line charts returned at the end of the program each time.

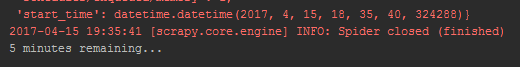
## currency\_scraper.py



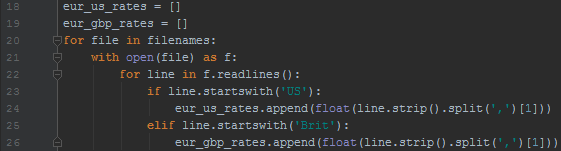
This is the main file of the program and the one that the user needs to run. The first four lines import the modules that are needed.



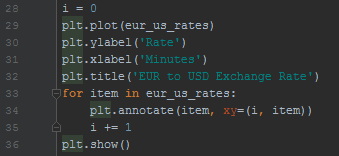
The code in the while loop here repeats until the count variable reaches a value of 6. The call() function, which was imported from the subprocess package, runs ‘feed\_export\_script.sh’. The print statement uses the ‘time\_remaining’ variable to tell the user how many minutes are remaining until the program is finished running. The sleep() function, which was imported from the time package, tells the program to wait for 60 seconds before continuing with the loop. This is encased in an if statement which ensures that the program will not wait for another 60 seconds after ‘feed\_export\_script.sh’ has been run 6 times. The glob function, imported from the glob package, stores any files with the extension ‘.csv’ in the ‘filenames’ list. This means the files containing scraped data will be stored in the filenames list.



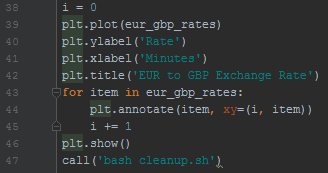
The image above shows the countdown displayed while the program is running.



This for loop goes through the filenames, opens each file, and stores the exchange rate for US Dollar and British Pound in the ‘eur\_us\_rates’ and ‘eur\_gbp\_rates’ lists respectively.



Finally, the pyplot module, which was imported from the matplotlib package under the alias ‘plt’, was used to plot the exchange rates for Euro-US Dollar on a line chart. The plot() function on line 29 uses the list of exchange rates (eur\_us\_rates) for the values it plots on the line chart. ylabel() and xlabel() label the y axis and x axis of the chart ‘Rate’ and ‘Minutes’ respectively. title() titles the chart ‘Eur to USD Exchange Rate’. The for loop uses the annotate() function to label each plot on the chart with the appropriate exchange rate value from the eur\_us\_rates list. The show() function then displays the line chart to the user. The user can then choose to save the chart if they wish.



Another line chart is then created for the Euro-British Pound exchange rates and displayed to the user. The ‘cleanup.sh’ script is then run on line 47.

The two images below show charts output by the program.

