

CoderGals Hackathon

Scraping FX exchange rates from the Reuters website

18-20 August 2017, Prizren, Kosovo

Group members:

1. Anita Veseli
2. Argeta Morina
3. Dhurata Kamberi
4. Edona Xhemajli
5. Egëzonë Haziri

# 1. Problem statement

Scraping FX exchange rates from public websites and storing the data into a database

# 2. Proposed solution

For this challenge we decided to use a framework from Python called Scrapy that grabs the FX exchange rates the currency, last time when were updated, the highest deal and the lowest. Scraped data should then be stored in MongoDB, for this procedure we will use PyMongo which is a package from python for connecting MongoDB with Python.

# 3. Installation

We need the [Scrapy](http://doc.scrapy.org/en/1.0/) library along with [PyMongo](http://api.mongodb.org/python/3.0.3/) for storing the data in MongoDB.

### 3.1 Scrapy

If you’re running OSX or a flavor of Linux, install Scrapy with pip (with your virtualenv activated):

$ pip install Scrapy

Once Scrapy is setup, verify your installation by running this command in the Python shell:

>>> import scrapy

>>>

If you don’t get an error then you are good to go!

## 3.2 PyMongo

Next, install PyMongo with pip:

$ pip install pymongo

# 4. Scrapy Project

Start a new Scrapy project by typing in command line:

$ scrapy startproject name\_of\_project

This creates a number of files and folders that includes a basic boilerplate for you to get started quickly:

|-- scrapy.cfg

|\_\_ stack

|--\_\_init\_\_.py

|-- items.py

|-- pipelines.py

|-- settings.py

|\_\_ spiders

## |--\_\_init\_\_.py

## 4.1 Specifying the data

The items.py file is used to define storage “containers” for the data that we plan to scrape.

The StackItem() class inherits from Item ([docs](http://doc.scrapy.org/en/1.0/topics/items.html)), which basically has a number of pre-defined objects that Scrapy has already built for us:

import scrapy

class StackItem(scrapy.Item):

# define the fields for your item here like:

# name = scrapy.Field()

Pass

Let’s add some items that we actually want to collect. For each exchange rate the client needs the currency, last rate, the highest rate and the lowest rate.

from scrapy.item import Item, Field

class StackItem(Item):

currency = Field()

last = Field()

dayhigh = Field()

daylow = Field()

scrapped\_at = Field()

## 4.2 Creating the Spider

We will create a file called stack\_spider.py in the “spiders” directory. In this file we tell Scrapy how to find the exact data we’re looking for. This is specific to each individual web page that we wish to scrape.

We start by defining a class that inherits form Scrapy’s Spider and then adding attributes as needed:

from scrapy import Spider

class StackSpider(Spider):

name = "stack"

allowed\_domains = ["reuters.com"]

start\_urls = [ "https://www.reuters.com/finance/currencies/quote?srcAmt=1&srcCurr=USD&destAmt=&destCurr=USD",

]

The first few variables are self-explanatory:

* name defines the name of the Spider.
* allowed\_domains contains the base-URLs for the allowed domains for the spider to crawl.
* start\_urls is a list of URLs for the spider to start crawling from. All subsequent URLs will start from the data that the spider downloads from the URLS in start\_urls.

## 4.3 XPath Selectors

Scrapy uses XPath selectors to extract data from a website. In other words, we can select certain parts of the HTML data based on a given XPath. As stated in Scrapy’s [documentation](http://doc.scrapy.org/en/1.0/topics/selectors.html), “XPath is a language for selecting nodes in XML documents, which can also be used with HTML.”

Again, we basically tell Scrapy where to start looking for information based on a defined XPath. We navigate to the Reuters’ site in Chrome and find XPath selectors.

We select the table:

table = Selector(response).xpath('//\*[@id="currPairs"]')

And then navigate through the table rows and table cells:

row.xpath('tbody/tr/td/a/text()')

4.4 Extract the Data

We parse the data and scrape like so:

for row in table[:1]:

items = {}

items['currency'] = row.xpath('tbody/tr/td/a/text()').extract()

items['last'] = row.xpath('tbody/tr/td[2]/text()').extract()

items['dayhigh'] = row.xpath('tbody/tr/td[3]/text()').extract()

items['daylow'] = row.xpath('tbody/tr/td[4]/text()').extract()

5. Store the data in MongoDB

The initial step is to create the database that we plan to use to save all of our crawled data. Open settings.py and specify the [pipeline](http://doc.scrapy.org/en/1.0/topics/item-pipeline.html) and add the database settings:

ITEM\_PIPELINES = {'stack.pipelines.MongoPipeline': 100 }

MONGODB\_SERVER = "localhost"

MONGODB\_PORT = 27017

MONGODB\_DB = "scrapy"

MONGODB\_COLLECTION = "currencies"

Connect to Database:

import pymongo

class MongoPipeline(object):

collection\_name = 'scrapy\_items'

def \_\_init\_\_(self, mongo\_uri, mongo\_db):

self.mongo\_uri = mongo\_uri

self.mongo\_db = mongo\_db

@classmethod

def from\_crawler(cls, crawler):

return cls(

mongo\_uri=crawler.settings.get('MONGO\_URI'),

mongo\_db=crawler.settings.get('MONGO\_DATABASE', 'items')

)

def open\_spider(self, spider):

self.client = pymongo.MongoClient(self.mongo\_uri)

self.db = self.client[self.mongo\_db]

def close\_spider(self, spider):

self.client.close()

def process\_item(self, item, spider):

self.db[self.collection\_name].insert(dict(item))

return item

6. Test

Again, run the following command within the “stack” directory:

$scrapy crawl stack

7. Scheduled Scrapy

We run the command to make the requests and store the data in database every 60 seconds:

$watch -n 60 scrapy crawl stack

8. Conclusions and future work

Forex (FX) is the market in which currencies are traded. The data collected from the websites using scrapy could be used for different data analysis that are required by any person, business, firm or country for their financial needs. This data could be used for different visualizations using software applications that could be built specifically for this purpose.