Instalation

Usefule sources

* https://thepythonscrapyplaybook.com/

Windows:

* pip install scrapy
* virtual env is installed with python3 (pip install virtualenv)
* create virtual environment ( It helps to isolate the third party python libraries and their versions so it doesn’t affect other projects if you upgrade one. )
* python3 -m venv venv // the last venv is the folder name
* venv/Scripts/activate(venv/bin/activate) // activate venv

Other commands

* scrapy crawl nameofspider

save

* scrapy crawl nameofspider -O filename.csv //capital o

Setting up Project

* scrapy startproject nameofproject

folder structure

* spider => what we scrape
* items => a temporary place to store the data
* piplines => store in databases for example
* middlewares // managing cookies, cach
  + downloader middleware
  + spider middleware // adding or removing requests or items, handling different exceptions that crop up if there is an error
* settings // enable obey robot.txt, number of concurrent requests. Enable the middlewares that you create in settings. Enable item-pipleline if you create a new one.
  + Print each product and its price. Also print a description of the price.

Spider folder

* scrapy genspider nameofspider urltoscrape // To create a spider folder. Should be run inside the spider folder.
* name = ‘x’ // this what we use in command scrapy crawl name
* allowed\_domains = [‘url’] // to limit what we want to scrape
* start\_urls = [‘ulr’]

parse function

* will be called when the response comes back. What we want to scrape from the page.
* scrapy shell // to run some commands and use scrapy functions in shell
  + fetch(‘url’) // puts the response in a variable called response
  + response.css(‘selector’)**.get()** // get() gives only the first item. If you want furthur css on the element **you shoud not use get()** as it turns it to string.
  + response.css(‘selector**::text**’).get() // gives the text
  + response.css(‘selector**’**)**.attrib[‘href’]**
  + response.css(’selector**::attr(href)**’).get()
  + books = response.css(‘selector’).getall()
  + response.xpath(“//ul[@class=’x’]/li[@class=’y’]/preceding-sibling::li[1]/a/text()”.get()
    - extract()
    - extract\_first()
    - get()
    - getall() // xpath("//ul[@class='simple']/text()").getall()
* yield // we need to yield items
* response.follow(url) // callback is executed after having the response

yield response.follow(next\_page\_url, callback = self.parse)

items.py

* Define the items we want to return. We can have multiple item classes inheriting from scrapy.Item
* It helps with misspellings
* **X = scrapy.Field()**
* from foldername.items import ItemName // In spider file import the item:

bookitem = BookItem()

book\_item[‘x’] = response.css(‘y’)

yield bookitem

* specify a specific **serializer** on a sepecific field ( for example save pound money character not encoded correctly). Put the sign before the value yourself.

def serialize\_price(value):

return f’$ {str(value)}’

class BookItem(scrapy.Item):

price\_tax = scrapy.Field(**serializer = serialize\_price**)

* If you have a lot of post-processing and cleaning it’s better to use pipleline instaed

Pipelines.py

* Cleaning
* Format
* Converting relative urls
* Validate data
* Store the data – sql
* **ItemAdapter** gives some useful functions we can use to clean the data

Enable pipeline

ITEM\_PIPELINES = {

‘bookscraper.pipelines.BookscrapperPiplines’: 300,

}

* BookscrapperPiplines is the name of the class

from itemadapter import ItemAdapter

class BookscraperPipeline:

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

adapter = ItemAdapter(item)

field\_name = adapter.field\_names()

for field\_name in field\_names:

if field\_name != ‘description’:

value = adapter.get(field\_name)

adapter[field\_name] = value[0].strip()

return item

* **adapter.get(field\_name) may return a tuple (value,). We need to get the first item of the tuple. This may because of the selector that we have chosen.**

Saving data

Command line

* scrapy crawl nameofcrawler -O x.csv // -O overwrites, -o appends
* types: csv, json

Feed setting

* Settings file
* Add this:

FEEDS = {

‘data.json’: {‘format’: ‘json’}

}

* Run the scrapper
* Also can define the FEEDS in the **spider file** // custom\_settings allows to override the settings file

custom\_setting = {

‘FEEDS:’: {‘data.json’: {‘format’: ‘json’, ‘overwrite’: True}}

}

Databases

* In pipeline file
* You can create a new class for saving into the db in pipeline apart from the one used for cleaning.
* Import connector, setup connection and cursor
* Use \_\_init\_\_ to setup connection.
* Use ‘**CREATE TABLE IF NOT EXISTS**’ to avoid errors.
* Self.cursor.execut(‘query’)
* Use the **process\_item** function for insert and commit and return the item.
* **The return item helps with other piplines, if we add another pipeline it will be passed to it.**
* Close the connection, close\_spider is a function that scrapy looks for when the spider is ready to close and is present it will be run.
* **Enable new pipeline in settings.py**
* clean the data first and the save to db. The numbers are the order in which the items in the item pipeline have precedence. The lower the number the higher priority. These numbers can be any numbers.

ITEM\_PIPELINES = {

‘bookscraper.pipelines.BookscraperPipeline’: 300,

‘bookscraper.pipelines.SaveToMySQLPipeline: 400,

}

User Agent and headers

* User agent in header tells who we are to the server
* Useragentstring.com // analyze user agent string
* For large emcommerce websites, anibots may recognize simple headers.
* Ip addresses, cookies, sessions in the request also is used to block the request
* Sometimes by just changing the user agent we can go through simpler websites, but more complex websites may capture, os system, versions, and … to find a pattern even with different user agents. For the latter we need to change all the header and notjust the user agent.

User agent

* In the spider file

//sample user agent string

USER\_AGENT = ‘Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Mobile Safari/537.36’

* Why not put it in the settings.py?
  + Because we don’t want to use it for each request. After a few requests the server can detect and block us.
* Create list of user agents:

user\_agent\_list = [ ‘’, ‘’, ‘’]

* And loop through them

response.follow(url, callback, headers = {“User-Agent”: self.user\_agent\_list[random.randint(0, len(self.user\_agent\_list)-1)])

Middle ware for user agents

* scrapeops.io // What if we have thousands of requests? A limited list would not work. Register, gives you an api key to use their api.

A screenshot of a computer code

Description automatically generated

* In the middlewares.py create a new class
* in settings set the settings you want to use in your middlewares. For example:
  + SCRAPEOPS\_API\_KEY= ‘ key ‘
  + scrapeops\_face\_user\_agents\_enabled = 50
* get the settings
* request the list
* get random user agent
* check if the module is enabled
* process\_request // is what scrapy looks for middlewares and it process what we have defined to happen. Get random agent and put it in the header.

from urllib.parse import urlencode

from random import randint

import requests

class ScrapeOpsFakeUserAgentMiddleware:

@classmethod

## makes sure we have access to crawler settings

def from\_crawler(cls, crawlr):

return cls(crawler.settings)

#set some settings

def \_\_init\_\_(self, settings):

self.scrapeops\_api\_key = settings.get(‘SCRAPEOPS\_API\_KEY’)

self.scrapeops\_endpoint = setting.get(‘Scrapeops\_fake\_user\_agent\_endpoint’, default)

self.scrapeops\_num\_results = settings.get(‘SCRAPEOPS\_NUM\_RESULTS’)

self.scrapeops\_fake\_user\_agents\_active = settings.get(‘SCRAPEOPS\_FAKE\_USER\_AGENT\_ENABLED’, default)

self.headers\_list = []

self.\_get\_user\_agents\_list()

self.\_scrapeops\_face\_user\_agents\_enabled()

def get\_user\_agents\_list():

payload = {‘api\_key’: self.scrapeops\_api\_key}

if self.scrapeops\_num\_results is not None:

payload[‘num\_results’] self.scrapeops\_num\_results

response = requests.get(self.scrapeops\_endpoint, params = urlencode(payload))

json\_response = resposn.json()

self.user\_agents\_list = json\_response.get(‘result’, [])

def \_get\_random\_user\_agent(self):

random\_index = randint(0, len(self.user\_agents\_list) -1)

return self.user\_agents\_lsit[random\_index]

def \_scrapeops\_fake\_use\_agents\_enabled(self):

if self.scrapeops\_api\_key is None or self.scrapeops\_api\_key == ‘’ or

self.scr

self.scrapeops\_fake\_urser\_agents\_active = True

else self.scrapeops\_fake\_user\_agents\_active = True

def process\_request(self, request, spider):

random\_user\_agent = self.\_get\_random\_user\_agent()

request.headers[‘User-Agent’] = random\_user\_agent

* Last step => got to settings.py add the middleware to DOWNLOADER\_MIDDLEWARES. ScrapeOpsFakeUserAgentMIddleware is the name of the class.

DOWNLOADER\_MIDDLEWARES = {

‘bookscraper.middlewares.ScrapeOpsFakeUserAgentMIddleware’: 400,

}

* We can add another middleware to get random full headers as well from this provider.

Proxies

* Change ip addresses
* pip install scrapy-rotating-proxies // it installs a middleware, need to activate it

DOWNLOADER\_MIDDLEWARES = {

‘rotating\_proxies.middlewares.rotatingProxyMiddleware’:610,

‘rotating\_proxies.middlewares.BanDetectionMiddleware:620,

}

* in settings.py:

ROTATING\_PROXY\_LIST = [

‘proxy1.com:8000’,

‘proxy2.com:8031’,

‘proxy3.com:8032’

]

* or if it’s in a file

ROTATING\_PROXY\_LIST\_PATH = ‘/my/path/proxies.txt’

Proxy port

* request the list
* one ip address and a port // a proxy provider will give us this and will handle ip change.
* We can still use change of agent and headers
* Setup your settings and username and pass with the provider and they will give you an endpoint string (for example in smart proxy:)

A close up of a computer screen

Description automatically generated

* We can still use change of agent and headers

Method 1

* Setup your settings and username and pass with the provider and they will give you an endpoint string:
* Disable the middlewares for rotating\_proxy module as smart proxy handles them it self.
* In spider file, in a follow request

yield response.follow(book\_url, callback= self.parse\_book\_page, meta = {‘proxy’: ‘https//user-asdas3a4545:12345678@gate.smartproxy.com:7000’)

Method 2

* Custom middleware.
* This will change the requset, and adds the proxy to it. The proxy user and pass are encoded and added to the header so the traffic can go through.
* In settings.py:

PROXY\_USER = ‘username’

PROXY\_PASSWORD = ‘password’

PROXY\_ENDPOINT = ‘GATE.SMARTPROXY.COM’

PROXY\_PORT = ‘7000’

import base64

class MyProxyMiddleware(object):

@classmethod

def from\_crawler(cls, crawler):

return cls(crawler.settings)

def \_\_init\_\_(self, settings):

self.user = settings.get(‘PROXY-USER’)

self.password = settings.get(‘PROXY\_PASSWORD’)

self.endpoint = settings.get(‘PROXY\_ENDPOINT’)

sel.port = settings.get(‘PROXY\_PORT’)

#this function is what scrapy look for

def process\_request(self, request, spider):

user\_credintials = ‘{user}:{passw}’.format(user=self.user, passw=self.password)

basic\_authentiction = ‘Basic ‘ +

# Need this for proxy authorization header for the request

base64.b64encode(user\_credentials.encode()).decode()

host = ‘http://{endpoint}:{port}’.format(endpoint = self.endpoint, port = self.port)

request.meta[‘proxy’] = host

request.headers[‘Proxy-Authorization’] = basic\_authentication

* Enbale the middleware in downloader middlewares:

‘bookscraper.middlewares.MyProxyMiddleware’: 350

Proxy API endpoint

* Scrapeops.io // If we don’t even want to deal with user agents and headers, or we need headless browser with javascript running, we can do that by using a servic called proxy api and use their endpoints.
* Setup for free for limited usage => go to proxy aggregator page => request builder => api key and endpoint
* In spider file: a new function

def get\_proxy\_url(url):

payload = {‘api\_key’: API\_KEY , ‘url’: url}

proxy\_url = ‘https://proxy.scrapeops.io/v1/?’ + urlencode(payload)

return proxy\_url

* And use it:

yield response.follow(url = get\_proxy\_url(book\_url), callback= self.parse\_book\_page)

* Add the start\_request function to send the first url through proxy end point as well // scrapy looks for this function when you start up the spider, if it’s not given the spider uses the start\_urls.

def start\_request(self) :

yield Scrapy.Request(url = get\_proxy\_url(self.start\_urls[0]), callback= self.parse\_book\_page)

* In the **allowed\_domains** also the domain of proxy api whould be added

Proxy middleware by scrapops

* It makes things easier with proxy apis. With this we don’t have to get\_proxy\_url function everywhere we need it.
* pip intall scrapeops-scrapy-proxy-sdk
* in settings.py:

SCRAPEOPS\_API\_KEY = ‘YOUR\_API\_KEY’

SCRAPEOPS\_PROXY\_ENABLED = True

DOWNLOADER\_MIDDLEWARES = {

‘scrapeops\_scrapy\_proxy\_sdk.scrapeops\_scrapy\_proxy\_sdk.ScrapeOpsScrapyProxySdk’: 725

}

* and run it!
* You can see the request report in their dashboard.