Instalation

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,

}