NYC DATA SCIENCE ACADEMY

Jump Start Session: Scrapy

Data Science Bootcamp

Outline

- Introduction to Scrapy
- An Example
- Getting Started
- items.py
- wiki_spider.py
- pipelines.py
- settings.py
- The Full Example

Outline

- Introduction to Scrapy
  - An Example
  - Getting Started
  - items.py
  - wiki_spider.py
  - pipelines.py
  - settings.py
  - The Full Example

Introduction to Scrapy

Introduction to Scrapy

- A powerful application framework for extracting structured data from web pages
- You write rules to extract data, and Scrapy does the rest
- Offers the capability to easily post-process and persist your scraped data
- Has a healthy community:
  - https://twitter.com/ScrapyProject
  - http://stackoverflow.com/tags/scrapy/info
  - https://github.com/scrapy/scrapy

Introduction to Scrapy

- You define items -- containers that are used to collect scraped data
- You define spiders -- Python classes which define how a certain site (or group of sites) will be scraped
- You define item pipelines -- Python classes that sequentially process data

Then you deploy your spider!

Outline

- Introduction to Scrapy
- **An Example**
- Getting Started
- items.py
- wiki_spider.py
- pipelines.py
- settings.py
- The Full Example

An Example

An Example

- To introduce Scrapy, we'll go through the following tasks:
  - Creating a new Scrapy project
  - Defining items to extract
  - Writing a spider
  - Writing an item pipeline
- For this example, we'll scrape data from https://en.wikipedia.org/wiki/List_of_Academy_Award-winning_films

An Example

List of films [edit]

If a film won the Academy Award for Best Picture, its entry is listed in a shaded background with a **boldface** title. Any column in this list may be sorted by clicking the arrow sym desired column heading.

Note: *Competitive awards are separated from honorary awards; as such, any films that were awarded the latter will be shown in brackets next to the number of competitive wir*

Film	Year	Awards	Nominations
Spotlight	2015	2	6
Mad Max: Fury Road	2015	6	10
The Revenant	2015	3	12
Bridge of Spies	2015	1	6
The Big Short	2015	1	5
The Danish Girl	2015	1	4
Room	2015	1	4
Ex Machina	2015	1	2
The Hateful Eight	2015	1	2
Inside Out	2015	1	2
Amy	2015	1	1
Bear Story	2015	1	1
A Girl in the River: The Price of Forgiveness	2015	1	1
Son of Saul	2015	1	1
Spectre	2015	1	1

Outline

- Introduction to scrapy
- An Example
- **Getting Started**
- items.py
- wiki_spider.py
- pipelines.py
- settings.py
- The Full Example

Getting Started

Getting Started

- First, we'll need to obtain the package.
  - Make sure `pip` is installed, and execute the following command:

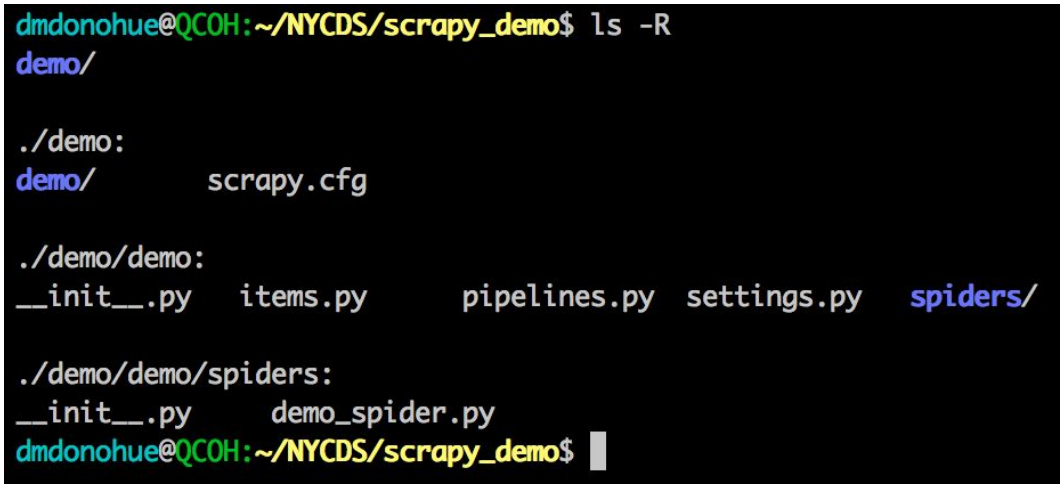
```
pip install scrapy
```

Getting Started

- Next, create a directory where you'll store your project, and run:

```
scrapy startproject demo
```

- This creates a directory with a number of files; we'll walk through each of them.



Outline

- Introduction to scrapy
- An Example
- Getting Started
- **items.py**
- demo_spider.py
- pipelines.py
- settings.py
- The Full Example

items.py

items.py

- Defines the containers that will store the scraped data
- You define the structure of the data you intend to scrape
- Once items are filled, they essentially act as Python dictionaries

items.py

```
1  # -*- coding: utf-8 -*-
2
3  # Define here the models for your scraped items
4  #
5  # See documentation in:
6  # http://doc.scrapy.org/en/latest/topics/items.html
7
8  import scrapy
9
10
11 class DemoItem(scrapy.Item):
12     # define the fields for your item here like:
13     # name = scrapy.Field()
14     pass
15
```

items.py

- Add the names of the fields we intend to scrape.
- Items are declared using a simple class definition syntax, and scrapy `Field` objects, e.g.,

```
from scrapy import Item, Field

class DemoItem(Item):
    film = Field()
    year = Field()
    awards = Field()
    nominations = Field()
```

- That's it!

Outline

- Introduction to scrapy
- An Example
- Getting Started
- items.py
- **demo_spider.py**
- pipelines.py
- settings.py
- The Full Example

demo_spider.py

demo_spider.py

- Create a file called `demo_spider.py` and place it within the `demo/spiders` directory
- A spider is a Python class that defines how the site will be scraped
  - How to perform the crawl (i.e., how to follow links, if necessary)
  - How to parse the contents of the page to extract `Items`

demo_spider.py

- A spider requires a few things (and must subclass `spider.Spider`):
  - `name`: an attribute specifying a unique name to identify the spider
  - `start_urls`: an attribute listing the URLs the spider will start from
  - `parse()`: a method of the spider responsible for processing a `Response` object downloaded from the URL and returning scraped data (as well as more URLs to follow, if necessary)

demo_spider.py

- First, we will need to import `scrapy.Spider` (from which our spider will inherit), and the item class we defined in `items.py`:

```
from scrapy import Spider
from demo.items import DemoItem
```

demo_spider.py

- Next, we define our spider class, and give it its required attributes:

```
from scrapy import Spider
from demo.items import DemoItem


class DemoSpider(Spider):
    name = 'demo'
    allowed_urls = ['en.wikipedia.org']
    start_urls = ['''https://en.wikipedia.org/wiki/
                  List_of_Academy_Award-winning_films''']
```

demo_spider.py

- Now, we need to define the Spider's `parse` method.
- The `parse` method is responsible for:
  - Parsing the responses (web pages) generated by `Request` objects (for each of the URLs in the `start_urls` list)
  - Returning `Item` objects, more `Request` objects (if your spider is following links), or iterables of these
  - Yielding scraped data for processing
- Our `parse` method will simply extract the film title, year, awards, and nominations.

demo_spider.py

- In Scrapy, you can parse a page's content and extract data from the HTML source using the built-in `Selector` class.
  - These are called selectors because they "select" certain parts of the HTML document specified by either XPath or CSS expressions
  - However, you can also use BeautifulSoup, lxml, or any other mechanism you prefer
- We'll fill our items using the built-in `Selector` along with XPath expressions
  - For more information about selectors and other extraction mechanisms, see the Selectors documentation

demo_spider.py

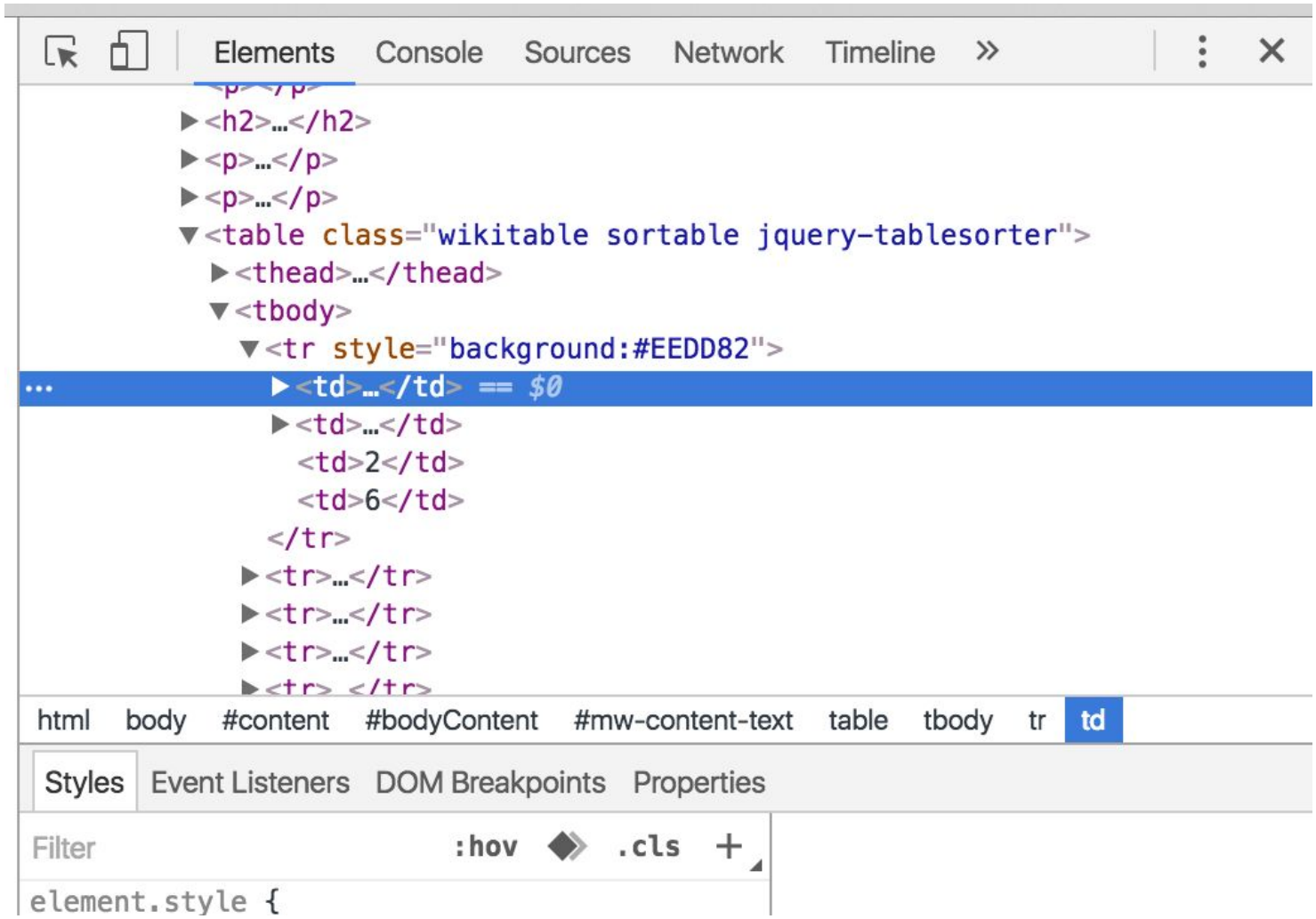
But what is XPath, and how do we use it?

- XPath is a language for selecting nodes in XML or HTML documents. For example:
  - `/html/head/title` selects the `<title>` element inside a `<head>` element of an HTML document
  - `//td` selects all the `<td>` elements
  - `//div[@class="mine"]` selects all the `<div>` elements containing an attribute `class="mine"`
- We'll need to construct the appropriate XPath expression to extract the data from the table on the Wikipedia page.

demo_spider.py

- Luckily, Chrome (and other browsers) have a developer's tool for inspecting the structure of web pages.

demo_spider.py

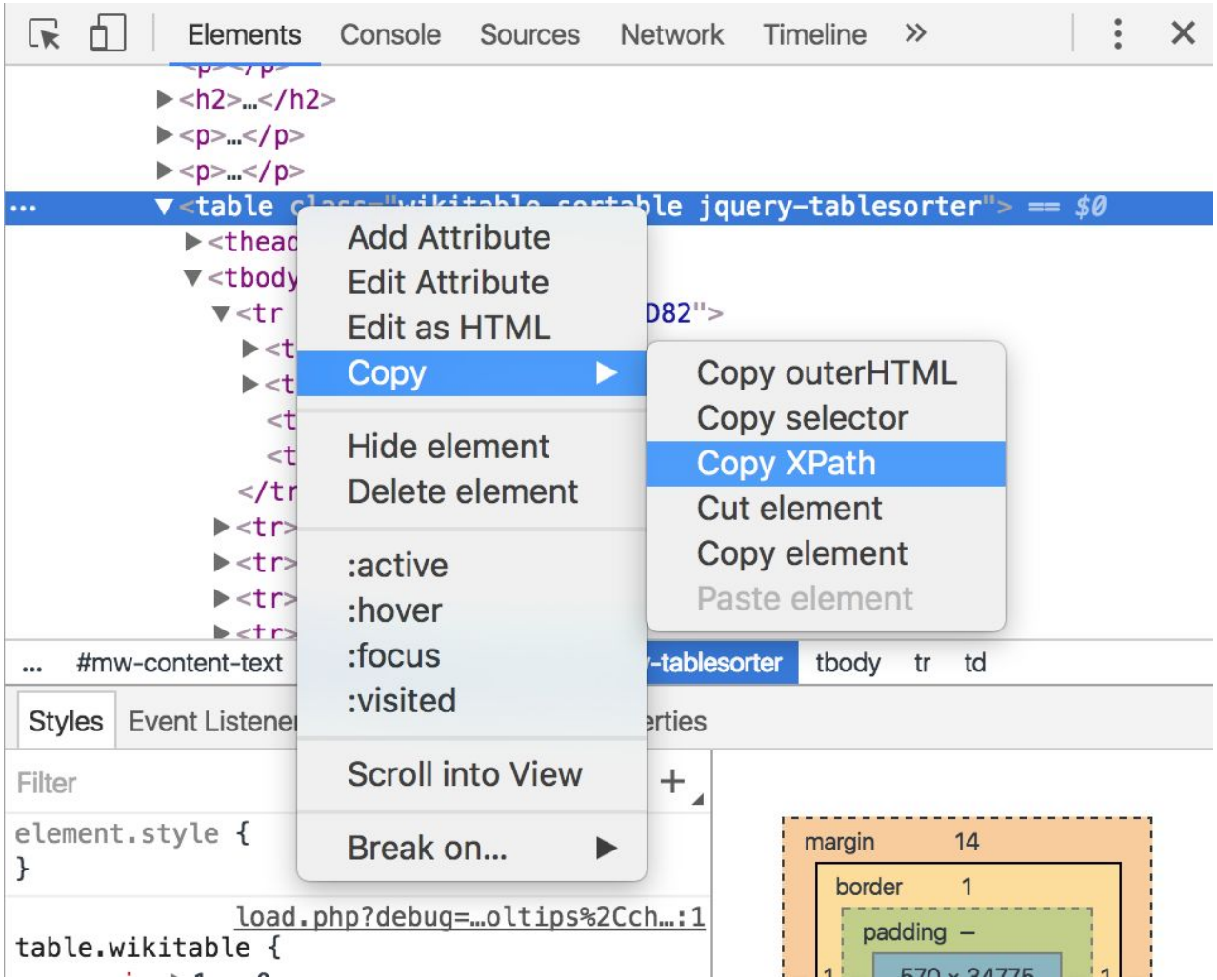


demo_spider.py

- It appears that each row of this table constitutes a `<tr>` element inside of the `<table class=...>` element
- We can extract all of these `<tr>` elements with the XPath expression `//path/to/tr`
- So, how can we determine the parents of these `<tr>` elements?

Again, Chrome developer tools to the rescue!

demo_spider.py



demo_spider.py

- The path to the table of Academy Award winning films is:
  - `//div[@id="mw-content-text"]/table[1]`
  - This means “select the first table inside the `div` element with `id="mw-content-text"`”
- We now want to extract all the `<tr>` elements starting from this path
  - The relevant XPath is, therefore,
    - `//div[@id="mw-content-text"]/table[1]/tr`

demo_spider.py

- To extract the data from the web page using this XPath expression, type:

```
response.xpath('//div[@id="mw-content-text"]/table[1]/tr')
```

- This returns a `Selector` object. To get a Python list of the selected data, use `extract()`:

```
response.xpath('//div[@id="mw-content-text"]/table[1]/tr')\
    .extract()
```

demo_spider.py

- To check that this will work (and for general debugging), enter the following command in your console:

```
scrapy shell <url_of_the_page_you're_scraping>
```

- This will start a Python shell with the Scrapy package imported, along with a `response` object from the request to the supplied URL.

demo_spider.py

```
In [19]: response.xpath('//div[@id="mw-content-text"]/table[1]/tr').extract()[:5]
Out[19]:
[u'<tr>\n<th>Film</th>\n<th>Year</th>\n<th data-sort-type="number">Awards</th>\n<th data-
sort-type="number">Nominations</th>\n</tr>',
 u'<tr style="background:#EEDD82">\n<td><i><b><a href="/wiki/Spotlight_(film)" title="Spo
tlight (film)">Spotlight</a></b></i></td>\n<td><a href="/wiki/2015_in_film" title="2015 i
n film">2015</a></td>\n<td>2</td>\n<td>6</td>\n</tr>',
 u'<tr>\n<td><i><a href="/wiki/Mad_Max:_Fury_Road" title="Mad Max: Fury Road">Mad Max: Fu
ry Road</a></i></td>\n<td><a href="/wiki/2015_in_film" title="2015 in film">2015</a></td>
\n<td>6</td>\n<td>10</td>\n</tr>',
 u'<tr>\n<td><i><a href="/wiki/The_Revenant_(2015_film)" title="The Revenant (2015 film)"
>The Revenant</a></i></td>\n<td><a href="/wiki/2015_in_film" title="2015 in film">2015</a
></td>\n<td>3</td>\n<td>12</td>\n</tr>',
 u'<tr>\n<td><i><a href="/wiki/Bridge_of_Spies_(film)" title="Bridge of Spies (film)">Bri
dge of Spies</a></i></td>\n<td><a href="/wiki/2015_in_film" title="2015 in film">2015</a>
</td>\n<td>1</td>\n<td>6</td>\n</tr>']

In [20]:
```

demo_spider.py

- Now we have a Python list, each list element being the HTML that comprises a row of the table.
- The next task is to iterate over this list, and extract the fields to fill our items.
- We can use Scrapy selectors and more XPath to extract these data.

demo_spider.py

- An example of an element from this list:

```
In [17]: response.xpath('//*[@id="mw-content-text"]/table[1]/tr').extract()[2]
Out[17]: u'<tr>\n<td><i><a href="/wiki/Mad_Max:_Fury_Road" title="Mad Max: Fury Road">Mad Max: Fury R
oad</a></i></td>\n<td><a href="/wiki/2015_in_film" title="2015 in film">2015</a></td>\n<td>6</td>\n<t
d>10</td>\n</tr>'
```

- For each of these elements, we need to extract:
  - The text from the `<a>` tags within the first and second `<td>` tags (these are the film title and year, respectively)
  - The text from the third and fourth `<td>` tags (these are awards and nominations, respectively)
- Import `Selector` from `scrapy.selector`, and then obtain these with:

demo_spider.py

```
# Get the movie title
film = Selector(text=row).xpath('//td[1]/i/a/text()')\
                .extract()

# Get the release year
year = Selector(text=row).xpath('//td[2]/a/text()')\
                .extract()

# Get the number of awards
awards = Selector(text=row).xpath('//td[3]/text()')\
                .extract()

# Get the number of nominations
nominations = Selector(text=row).xpath('//td[4]/text()')\
                .extract()
```

demo_spider.py

- Then you fill an item with:

```
item = DemoItem()
item['film'] = film
item['year'] = year
item['awards'] = awards
item['nominations'] = nominations
```

- Finally, `yield` this item to the item pipeline:

```
yield item
```

demo_spider.py

items.py | demo_spider.py | pipelines.py | settings.py

```
1   from scrapy import Spider
2   from scrapy.selector import Selector
3   from demo.items import DemoItem
4
5   class DemoSpider(Spider):
6       name = 'demo'
7       allowed_urls = ['en.wikipedia.org']
8       start_urls = ['https://en.wikipedia.org/wiki/List_of_Academy_Award-winning_films']
9
10      def parse(self, response):
11          rows = response.xpath('//*[@id="mw-content-text"]/table[1]/tr').extract()
12
13          for row in rows:
14              film = Selector(text=row).xpath('//td[1]/i/a/text()').extract()
15              year = Selector(text=row).xpath('//td[2]/a/text()').extract()
16              awards = Selector(text=row).xpath('//td[3]/text()').extract()
17              nominations = Selector(text=row).xpath('//td[4]/text()').extract()
18
19              item = DemoItem()
20              item['film'] = film
21              item['year'] = year
22              item['awards'] = awards
23              item['nominations'] = nominations
24
25              yield item
```

Outline

- Introduction to scrapy
- An Example
- Getting Started
- items.py
- wiki_spider.py
- **pipelines.py**
- settings.py
- The Full Example

pipelines.py

pipelines.py

- Once an item has been scraped, it is sent to the item pipeline
- Typical uses of an item pipeline are:
  - Cleansing HTML data
  - Validating scraped data
  - Checking for duplicates (and dropping them)
  - storing the scraped items in a file or a database

pipelines.py

- We're going to simply write each row to a text file.
- Each item pipeline component is a Python class that must implement a method with signature `process_item(self, item, spider)`.
- For instance, our item pipeline to write to a text file might look like the following:

```
12  class WriteItemPipeline(object):
13      def __init__(self):
14          self.file = open('AcademyAwards.txt', 'w')
15
16      def process_item(self, item, spider):
17          line = str(item['film'][0]) + '\t' + str(item['year'][0])\
18                  + '\t' + str(item['awards'][0]) + '\t'\
19                  + str(item['nominations'][0]) + '\n'
20          self.file.write(line)
21          return item
```

pipelines.py

- You can see more examples of item pipelines here (including an example of writing to a MongoDB collection).
- Also, check out feed exports, which allow you to serialize the scraped data in a number of formats and export to a number of storage backends.

Outline

- Introduction to scrapy
- An Example
- Getting Started
- items.py
- wiki_spider.py
- pipelines.py
- **settings.py**
- The Full Example

settings.py

settings.py

- The last file to edit is settings.py
- This is where you designate the behavior of all scrapy components (including item pipelines, spiders themselves, and more general global settings)
- In our case, we need only add:

```
DOWNLOAD_DELAY = 3

ITEM_PIPELINES = {'demo.pipelines.WriteItemPipeline': 100, }
```

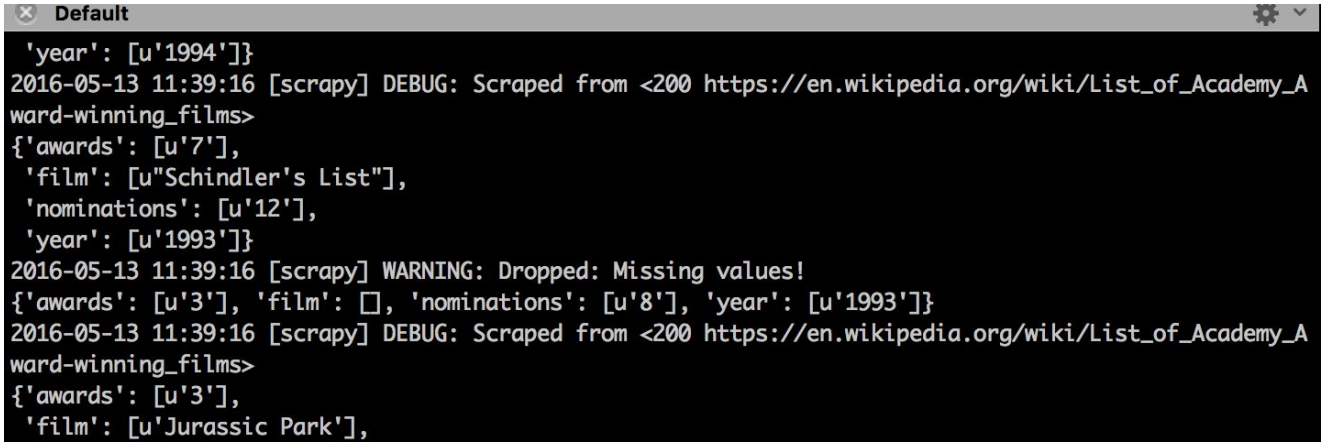
- These (respectively) avoid making too many requests in too short a time, and schedule the order of our item pipelines

settings.py

- Now we're ready to deploy our spider!
- This is done with the following syntax (assuming you're in the top level of the project's directory):

```
scrapy crawl demo
```

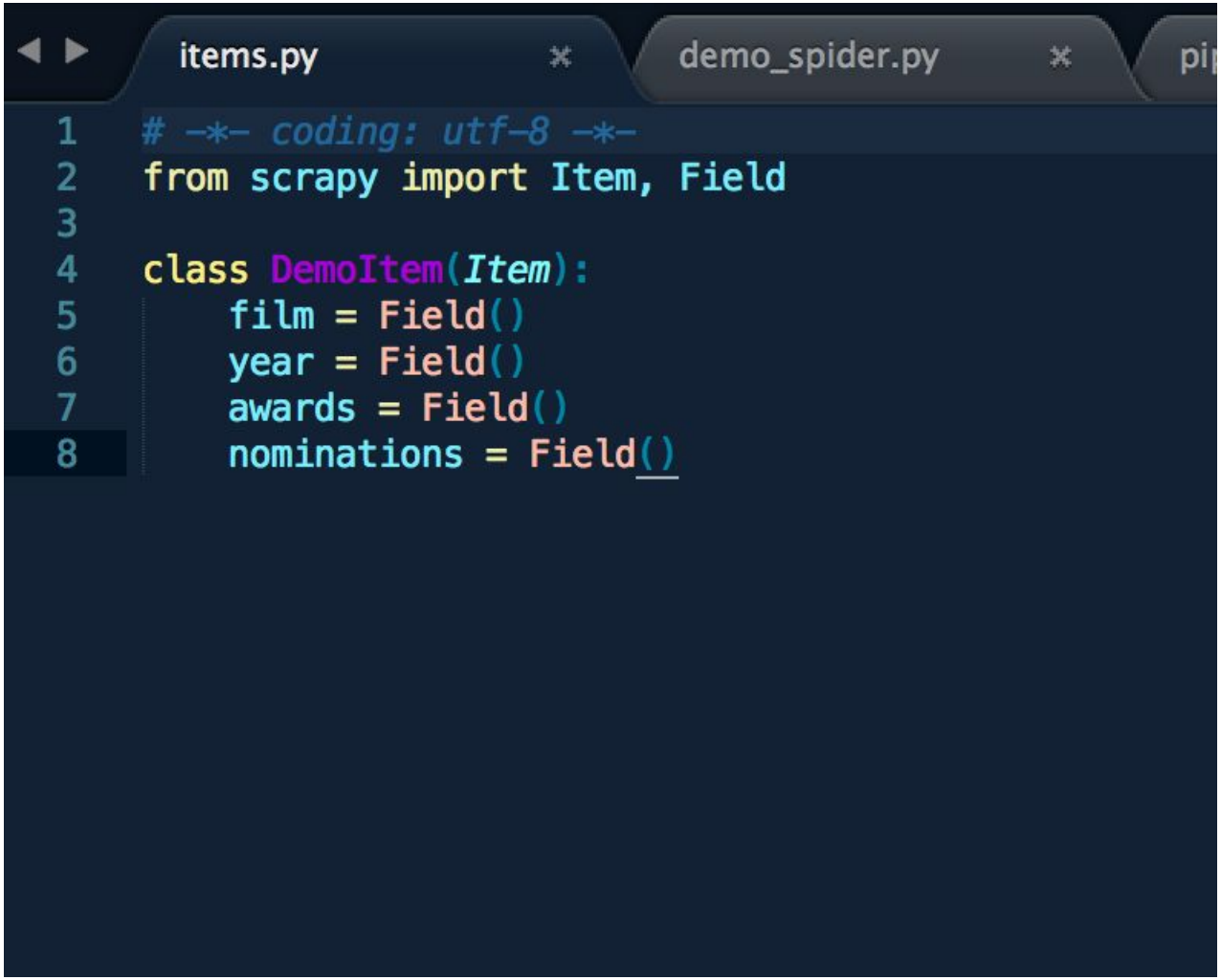
- You should see stuff like this:



Outline

- Introduction to scrapy
- An Example
- Getting Started
- items.py
- wiki_spider.py
- pipelines.py
- settings.py
- **The Full Example**

items.py


items.py
demo_spider.py
1 # -*- coding: utf-8 -*-
2 from scrapy import Item, Field
3
4 class DemoItem(Item):
5 film = Field()
6 year = Field()
7 awards = Field()
8 nominations = Field()

demo_spider.py

items.py | demo_spider.py | pipelines.py | settings.py

```
# -*- coding: utf-8 -*-
from scrapy import Spider
from scrapy.selector import Selector
from demo.items import DemoItem

class DemoSpider(Spider):
    name = 'demo'
    allowed_urls = ['en.wikipedia.org']
    start_urls = ['https://en.wikipedia.org/wiki/List_of_Academy_Award-winning_films']

    def parse(self, response):
        rows = response.xpath('//*[@id="mw-content-text"]/table[1]/tr').extract()

        for row in rows:
            film = Selector(text=row).xpath('//td[1]/i/a/text()').extract()
            year = Selector(text=row).xpath('//td[2]/a/text()').extract()
            awards = Selector(text=row).xpath('//td[3]/text()').extract()
            nominations = Selector(text=row).xpath('//td[4]/text()').extract()

            if not film:
                film = Selector(text=row).xpath('//td[1]/i/b/a/text()').extract()

            item = DemoItem()
            item['film'] = film
            item['year'] = year
            item['awards'] = awards
            item['nominations'] = nominations

            yield item
```

pipelines.py

items.py | demo_spider.py | pipelines.py

```
# -*- coding: utf-8 -*-
from scrapy.exceptions import DropItem

class ValidateItemPipeline(object):
    def process_item(self, item, spider):
        if not all(item.values()):
            raise DropItem("Missing values!")
        else:
            return item


class WriteItemPipeline(object):
    def __init__(self):
        self.file = open('AcademyAwards.txt', 'w')

    def process_item(self, item, spider):
        line = str(item['film'][0]) + '\t' + str(item['year'][0])\
               + '\t' + str(item['awards'][0]) + '\t'\
               + str(item['nominations'][0]) + '\n'
        self.file.write(line)
        return item
```

settings.py

items.py | demo_spider.py | pipelines.py | settings.py

```
1  # -*- coding: utf-8 -*-
2  BOT_NAME = 'demo'
3
4  SPIDER_MODULES = ['demo.spiders']
5  NEWSPIDER_MODULE = 'demo.spiders'
6
7  DOWNLOAD_DELAY = 3
8
9  ITEM_PIPELINES = {'demo.pipelines.ValidateItemPipeline': 100,
10                   'demo.pipelines.WriteItemPipeline': 200}
```