Robert Dempsey

- 15+ years in tech with 8 years programming
- Built 3 businesses
- Currently specializing in data wrangling
- Learning more of the data science

Robert Dempsey

robertwdempsey.com

rdempsey

robertwdempsey

rdempsey

Course Overview

Data Acquisition & Wrangling

- Part One: pulling and combining data from APIs
- **Part Two: web scraping**

Pre-Requisites

Pre-Requisites

- Some experience with Python
- Python 2.7.9 installed (Anaconda suggested)
- A text editor
  - Sublime Text 2:

Goal

Extract data from websites that have no API for market research purposes

Real-World Applications

- Gathering additional information to combine with existing databases
  - Are certain people commenting on certain websites?

Introduction

Why This Is Useful

- Not all data is available via an API
- APIs don’t always give you everything you need

Use Case: AirBnB

- Scraped Craigslist to target apartment postings and build their own customer base.
- Built bots to automatically post submissions.

What You'll Learn

- Writing a web scraper using the Scrapy framework
- Identifying the Xpath to elements you want to extract

Web Scraping Concepts

Web Scraping Concepts

- Web Scraper
- Web Spider
- Document Object Model (DOM)
- CSS Selectors
- XPath

Web Scraper

- Computer software that extracts information from websites; could be full web pages or part of a web page

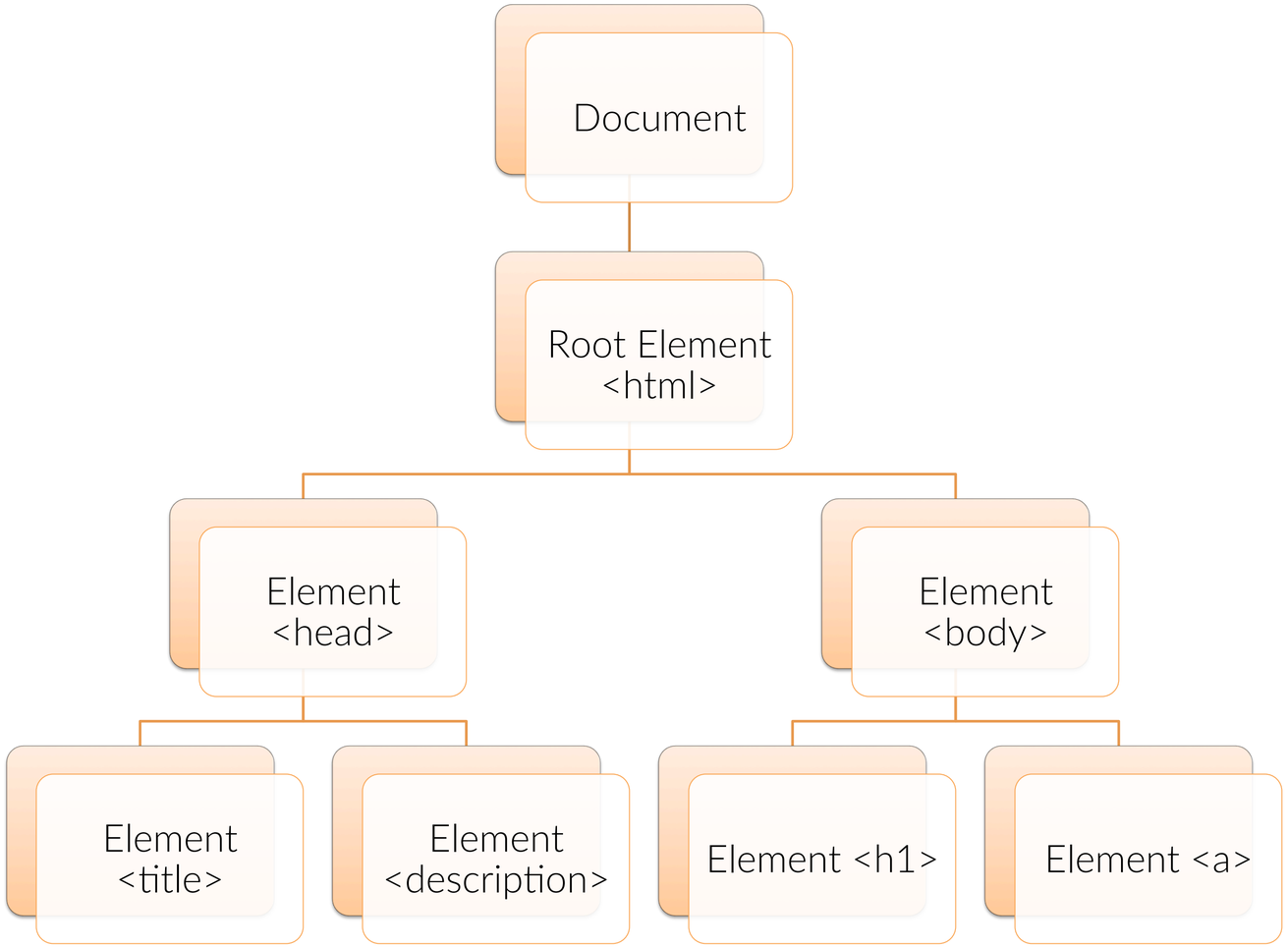
Web Spider

- An Internet bot which systematically browses the World Wide Web, typically for the purpose of Web indexing.
- Also known as a “web crawler”

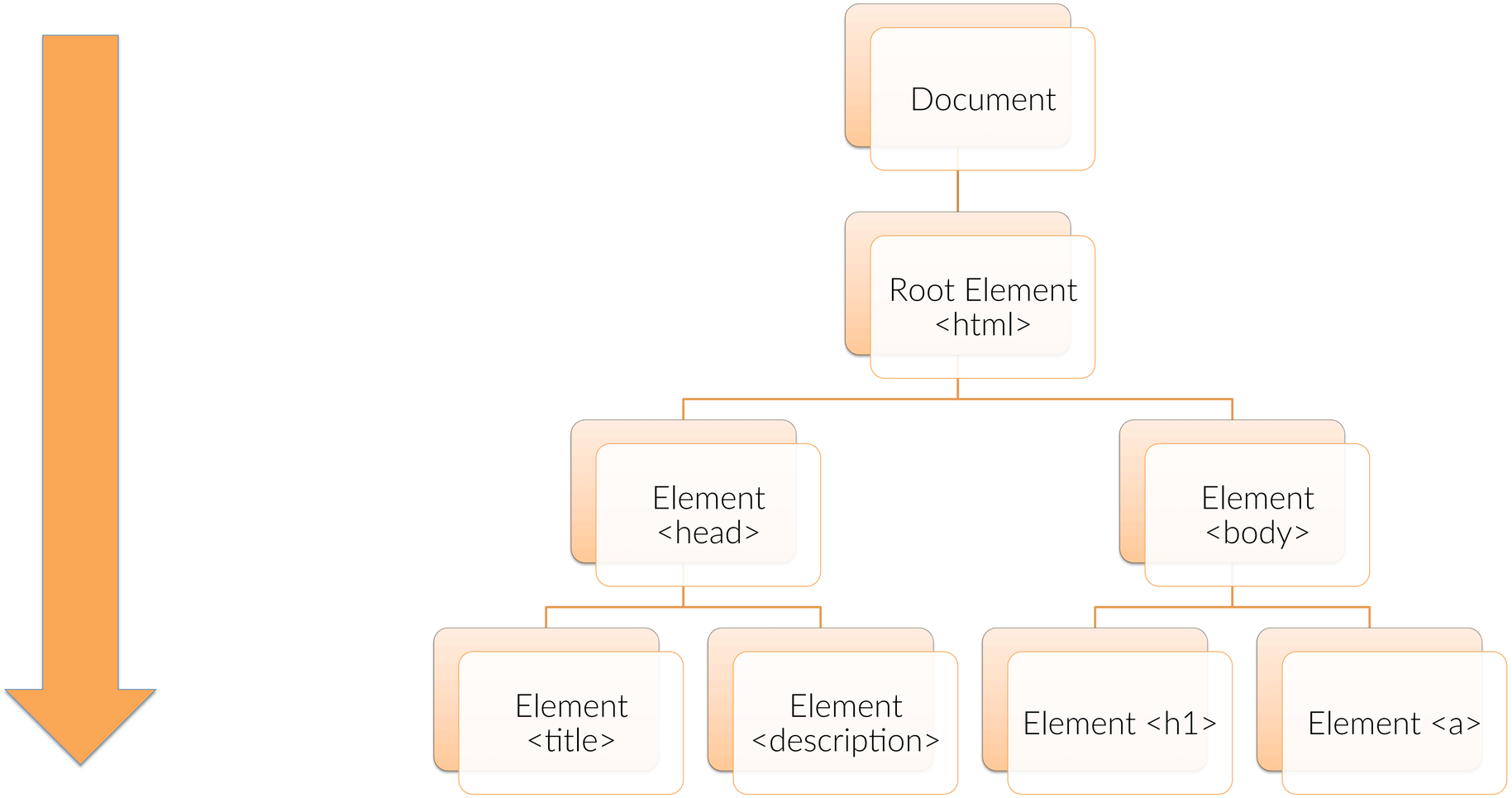
Document Object Model (DOM)

- A cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML, and XML documents.
- The nodes of every document are organized in a tree structure, called the DOM tree.
- Objects in the DOM tree may be addressed and manipulated by using methods on the objects.

DOM


Document
Root Element
<html>
Element
<head>
Element
<body>
Element
<title>
Element
<description>
Element <h1>
Element <a>

Parsing the DOM



CSS Selectors

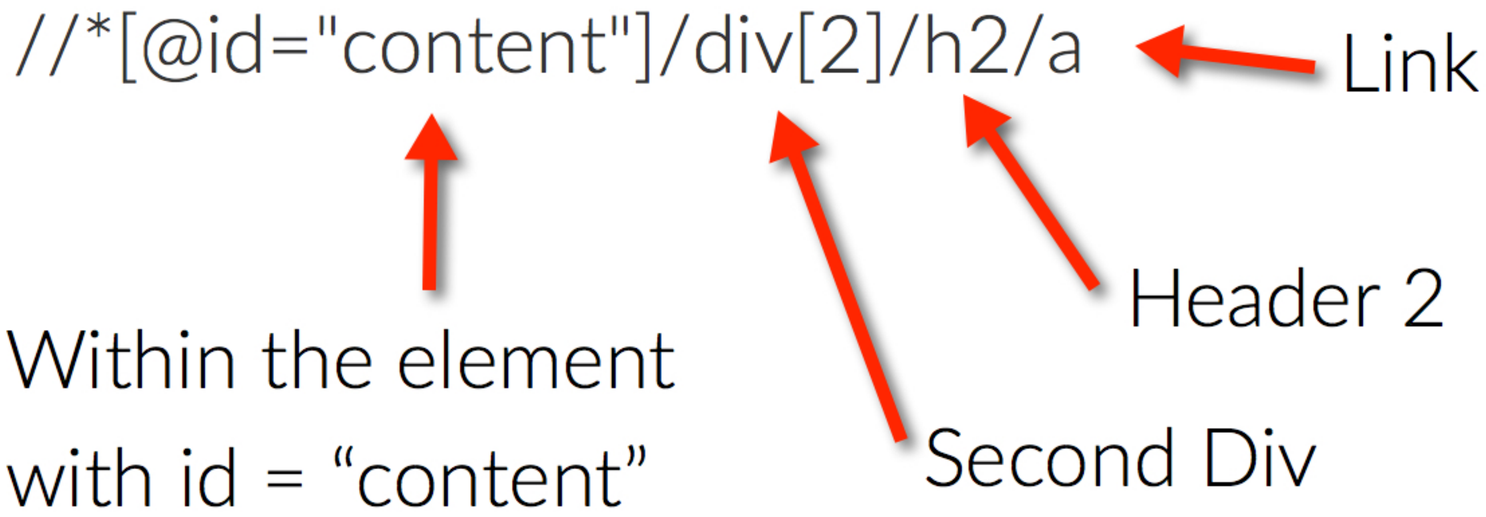
Patterns used to select the element(s) you want to style

Selector	Example	Example Description
.class	.entry-title	Selects all elements with class="entry-title"
#id	#content-sidebar-wrapper	Selects all elements with id = "content-sidebar-wrapper"
element	p	Selects all <p> elements (paragraph)
[attribute]	[target]	Selects all elements with a target attribute (links)

XPath

- Used to navigate through elements and attributes in a document
  - A syntax for defining parts of a document
  - Uses path expressions to navigate documents
  - Contains a library of standard functions
    - String values
    - Numeric values
    - Date and time comparison
    - Node manipulation
    - Sequence manipulation
    - Boolean values
    - Much more

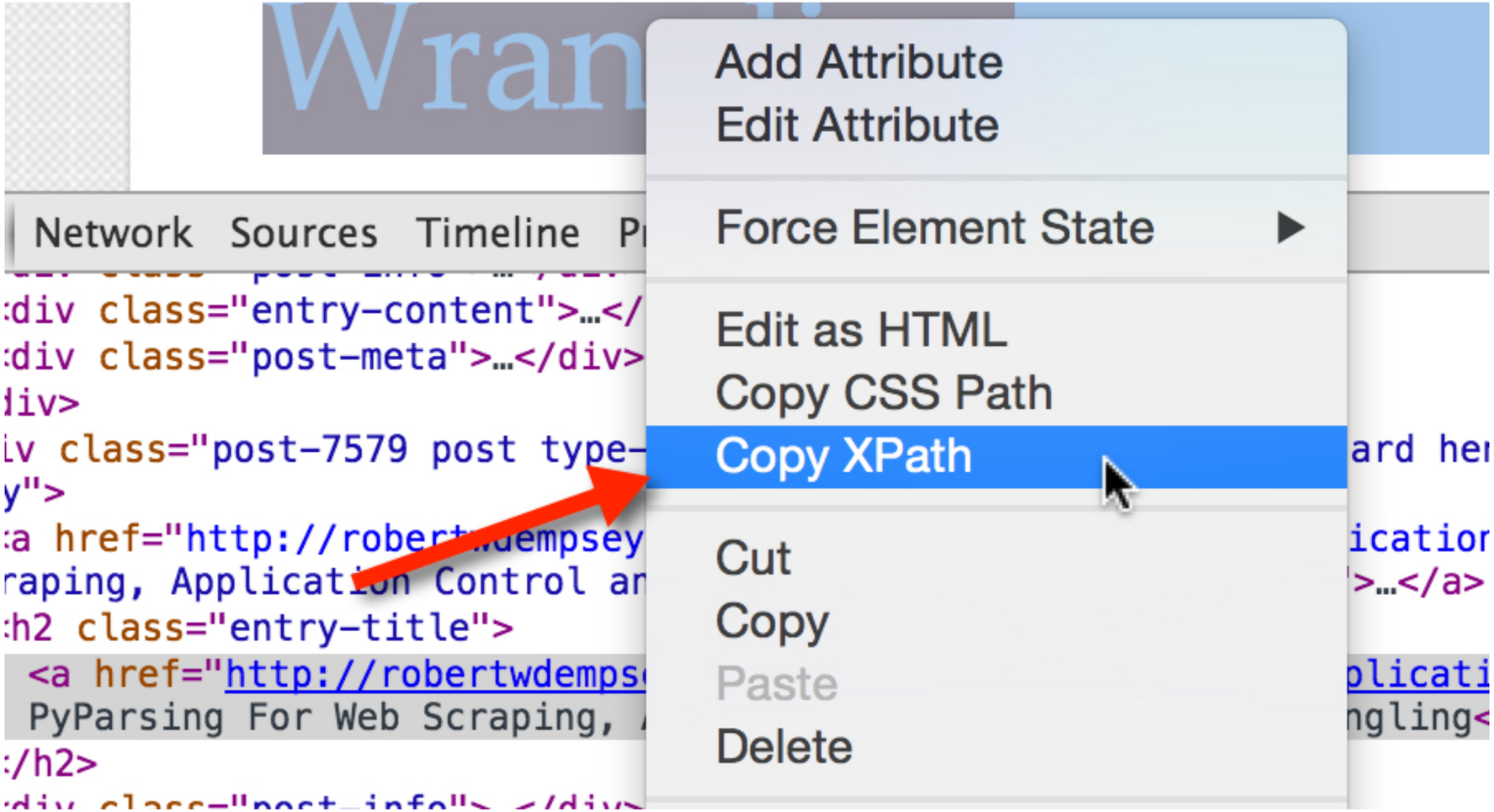
XPath Example



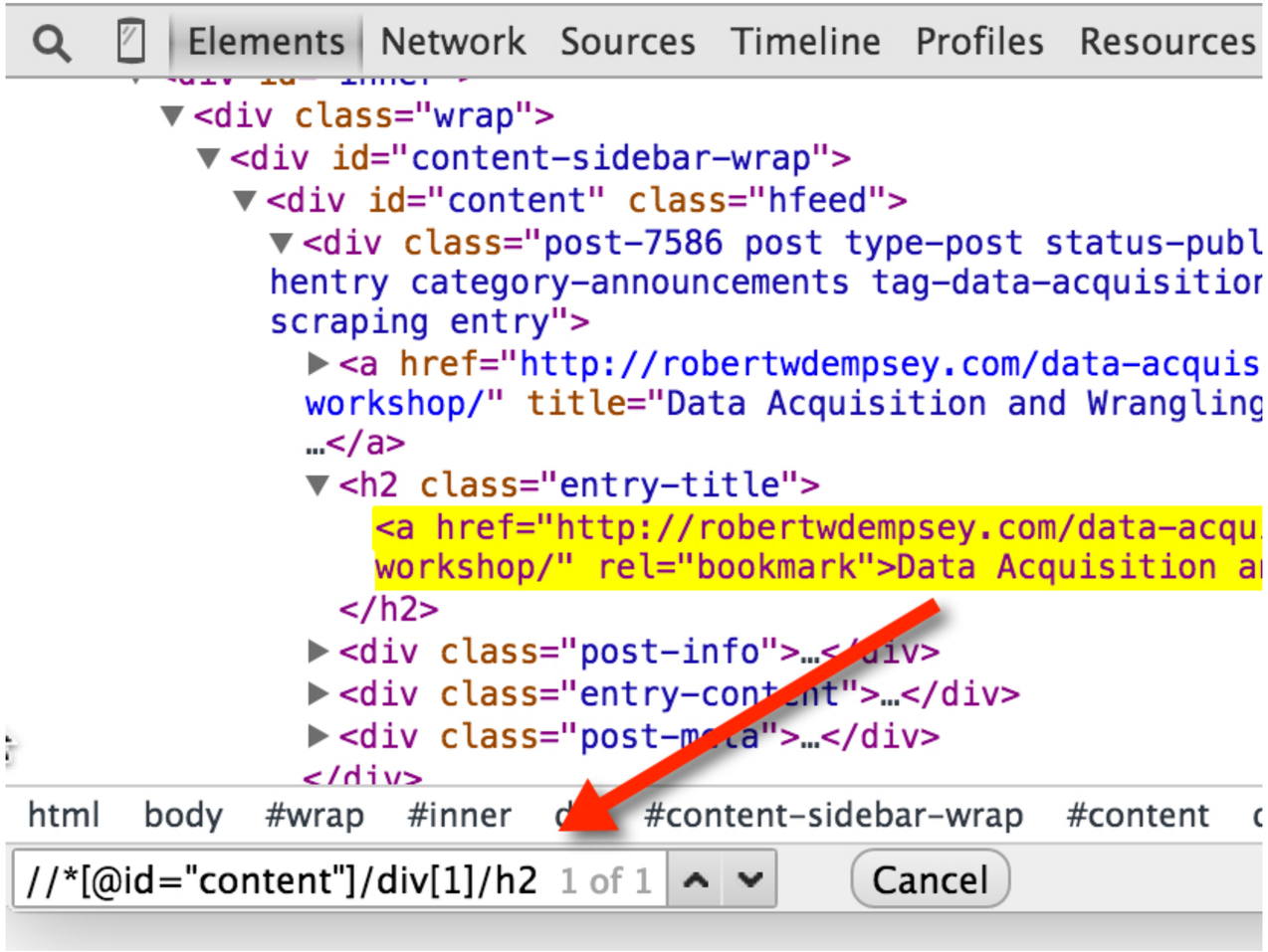
Identify & Test the XPath - Chrome

- Identify
  - Highlight the item you want
  - Right-click and select **Inspect Element**
  - In the console that appears right-click the item and select **Copy XPath**
  - Paste the XPath into your code
- Test
  - Press **F12** if the Chrome DevTools aren't already open
  - Press **Command + F** to enable the DOM search panel
  - Type in the XPath to evaluate
  - Matched elements will be highlighted in the DOM

Finding the XPath



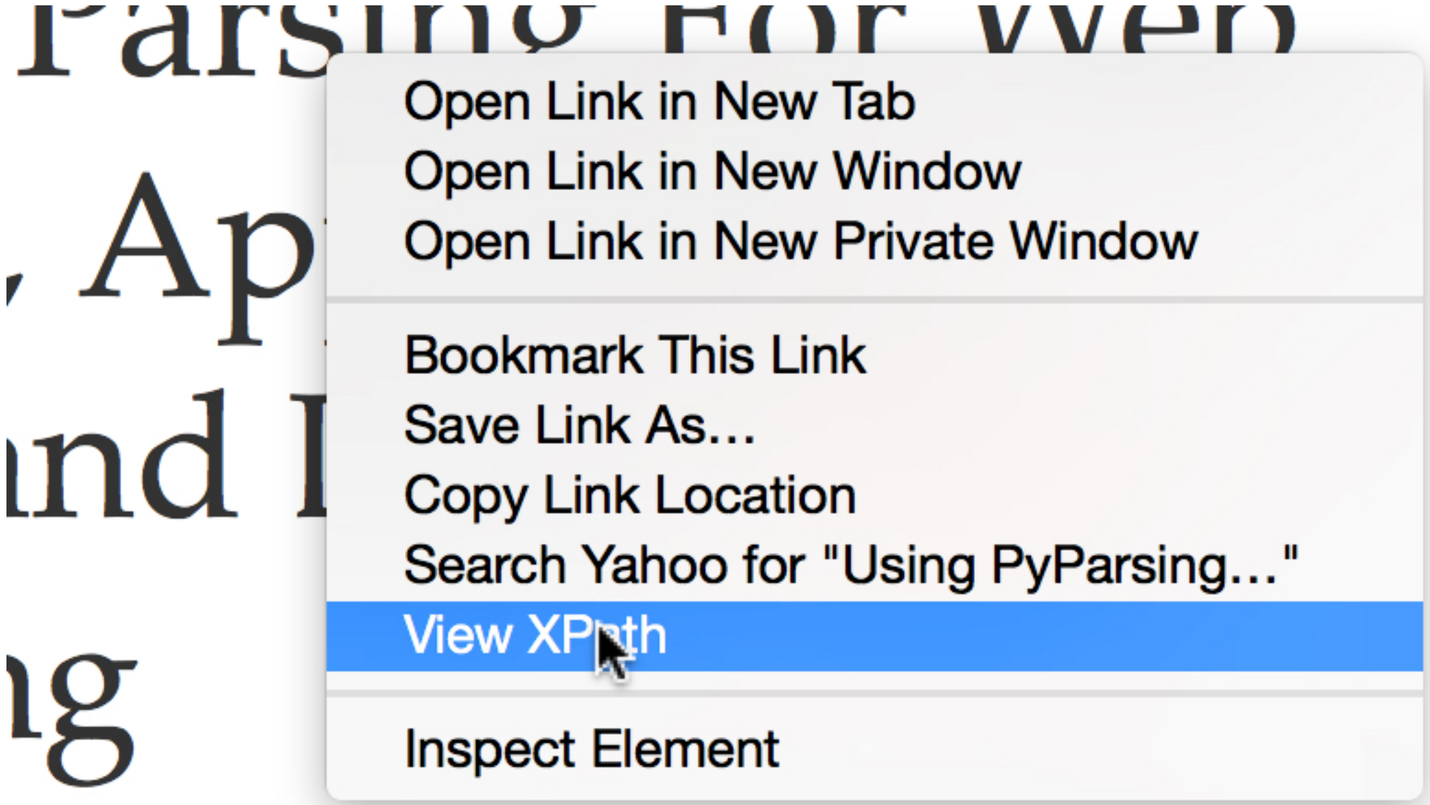
Testing In The DOM Search Panel



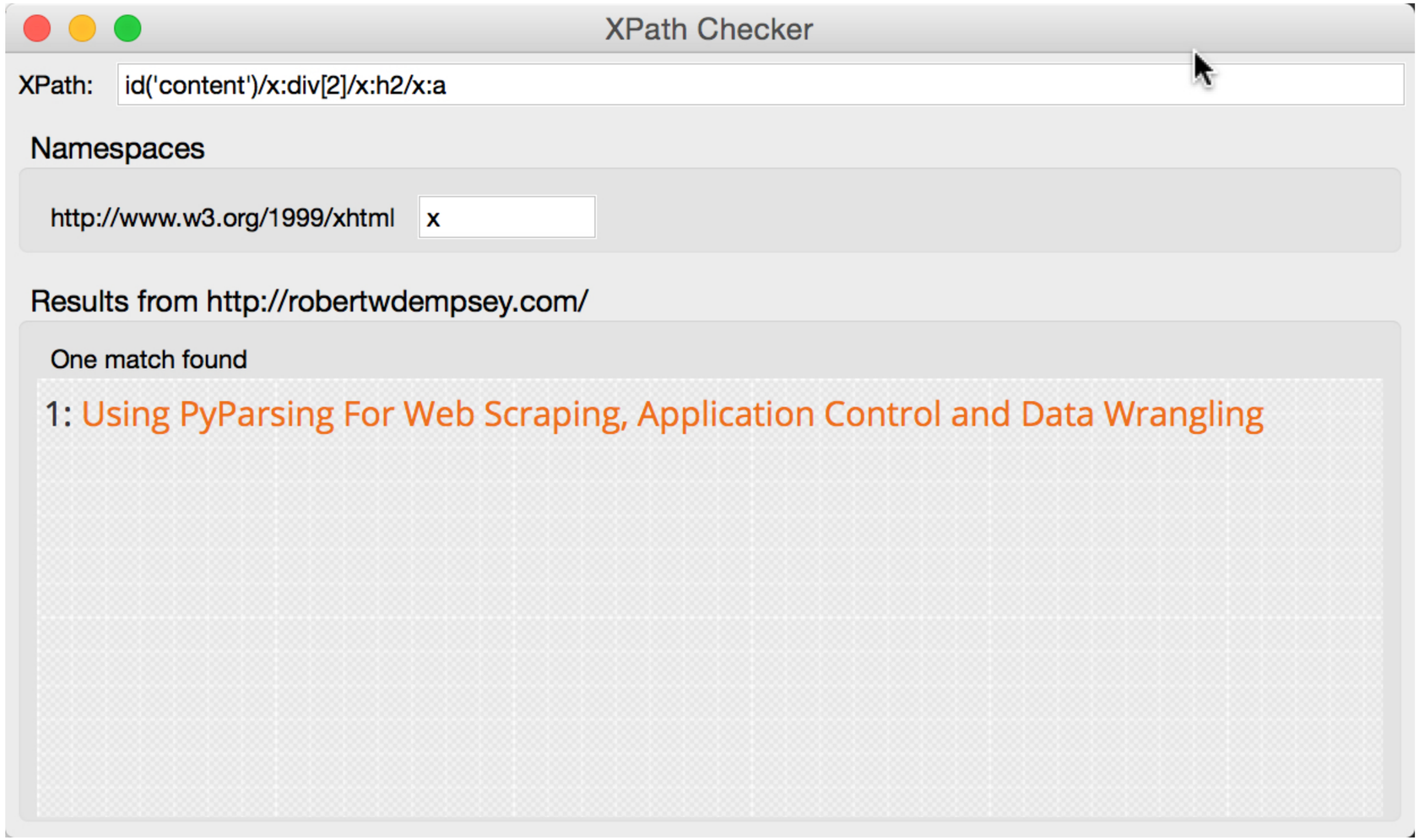
Identify & Test the XPath - FFDE

- Identify
  - Install the XPath Checker extension and restart Firefox
    - https://addons.mozilla.org/en-US/firefox/addon/xpath-checker/
  - Open the web page you want
  - Right-click the element you are looking for and select **View XPath**
  - Paste the result into your code
- Test
  - Right click an elements and select **View XPath**
  - Enter the XPath you want to test in the XPath Checker window
  - The results will automatically update

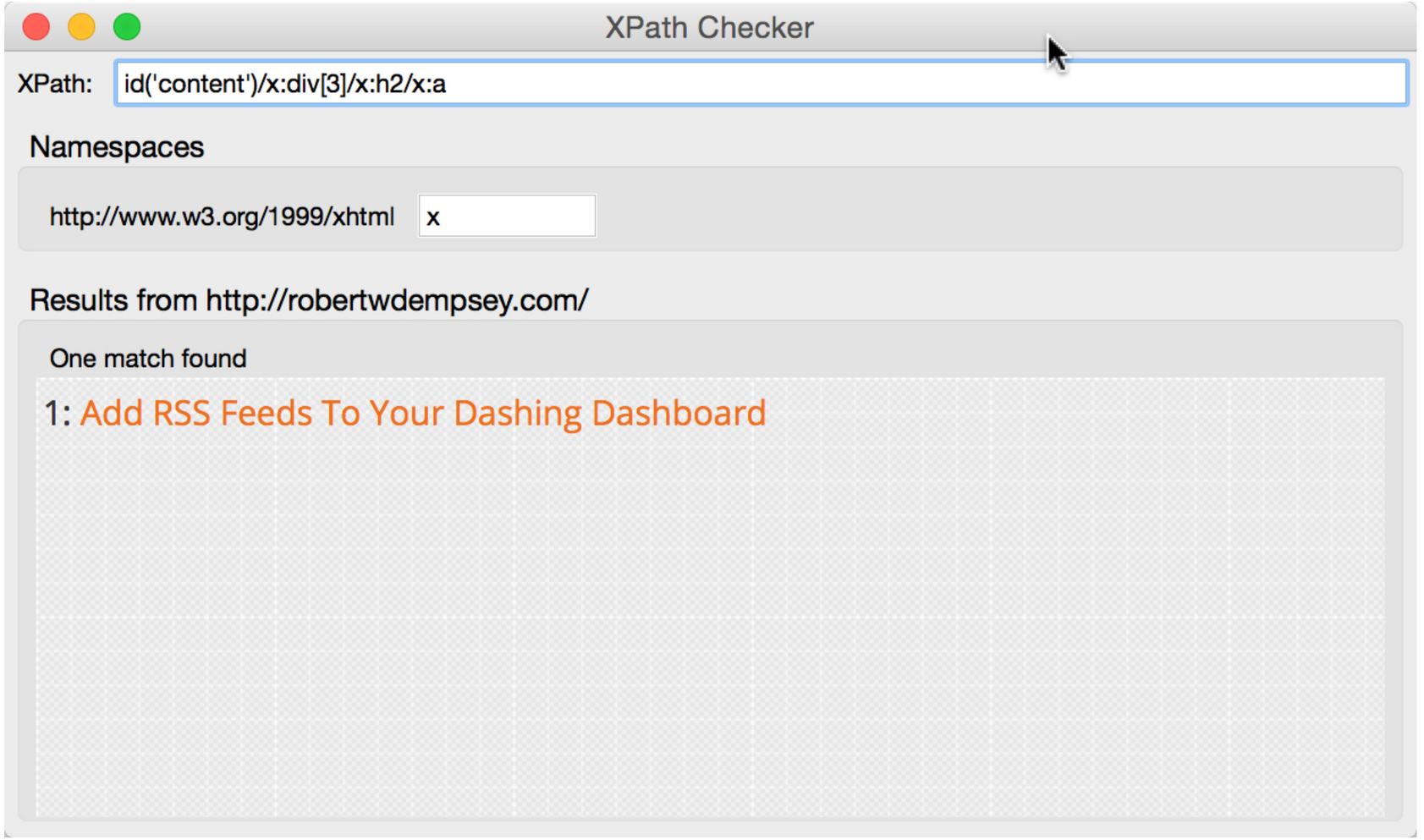
Identifying the XPath - FFDE



Identifying the XPath - FFDE



Testing the XPath - FFDE

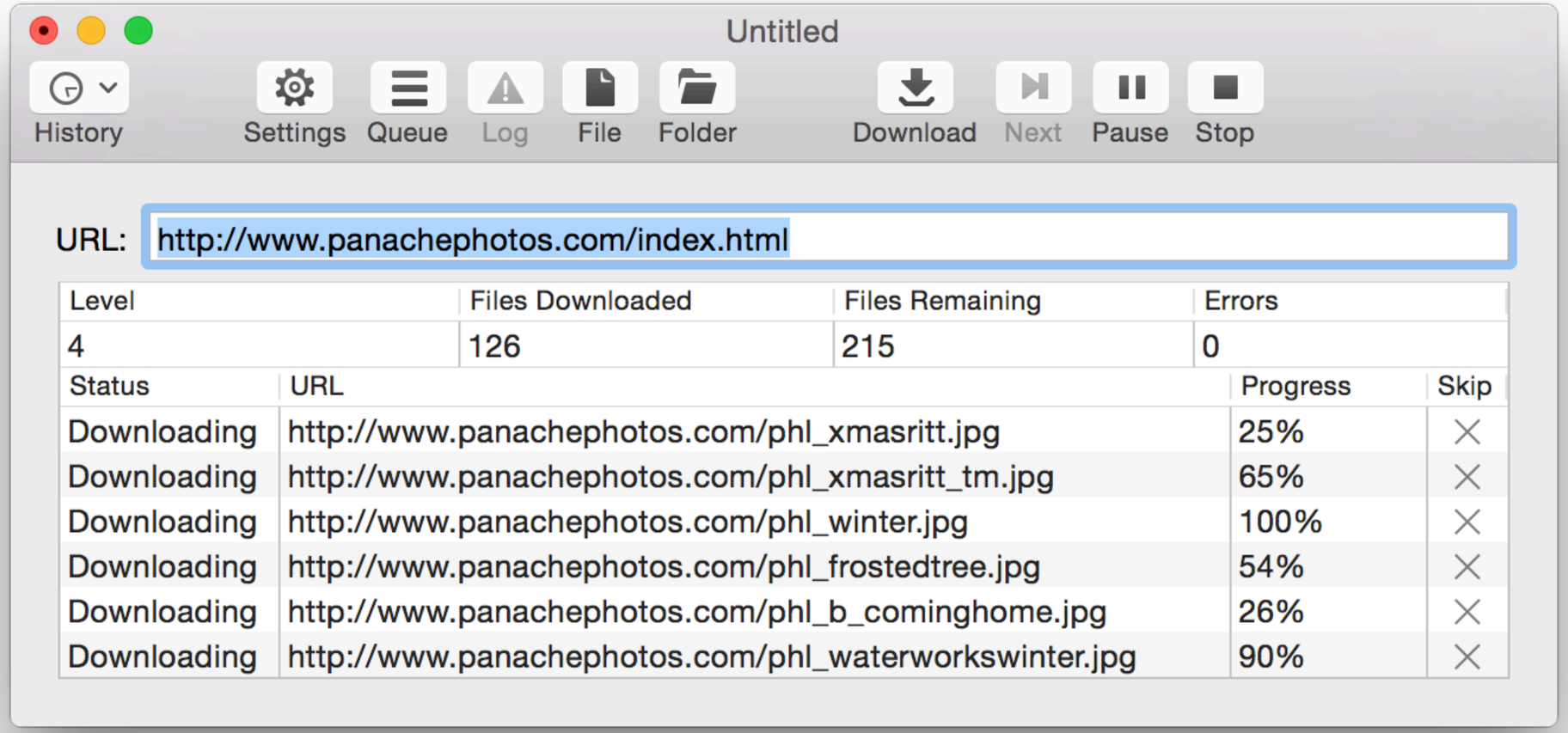


Build vs. Buy

Build vs. Buy

- Buy
  - Need something fast and simple
- Build
  - Need something truly custom
  - Web pages use crappy markup and it's harder to fully automate

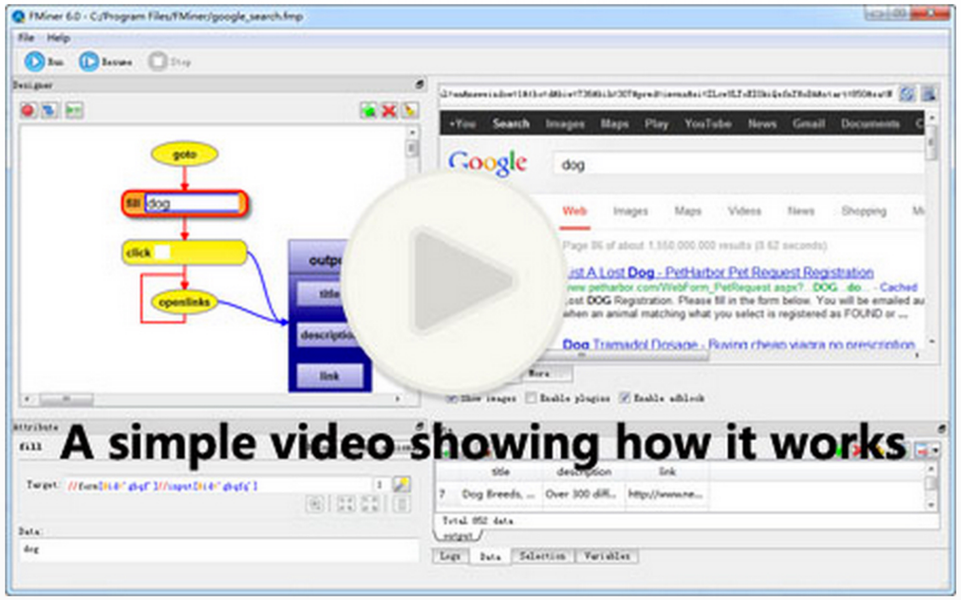
SiteSucker


Untitled
History
Settings
Queue
Log
File
Folder
Download
Next
Pause
Stop
URL: http://www.panachephotos.com/index.html
Level
Files Downloaded
Files Remaining
Errors
4
126
215
0
Status
URL
Progress
Skip
Downloading http://www.panachephotos.com/phl_xmasritt.jpg 25%
Downloading http://www.panachephotos.com/phl_xmasritt_tm.jpg 65%
Downloading http://www.panachephotos.com/phl_winter.jpg 100%
Downloading http://www.panachephotos.com/phl_frostedtree.jpg 54%
Downloading http://www.panachephotos.com/phl_b_cominghome.jpg 26%
Downloading http://www.panachephotos.com/phl_waterworkswinter.jpg 90%

FMiner

FMiner visual web scraping software with macro recorder and diagram designer

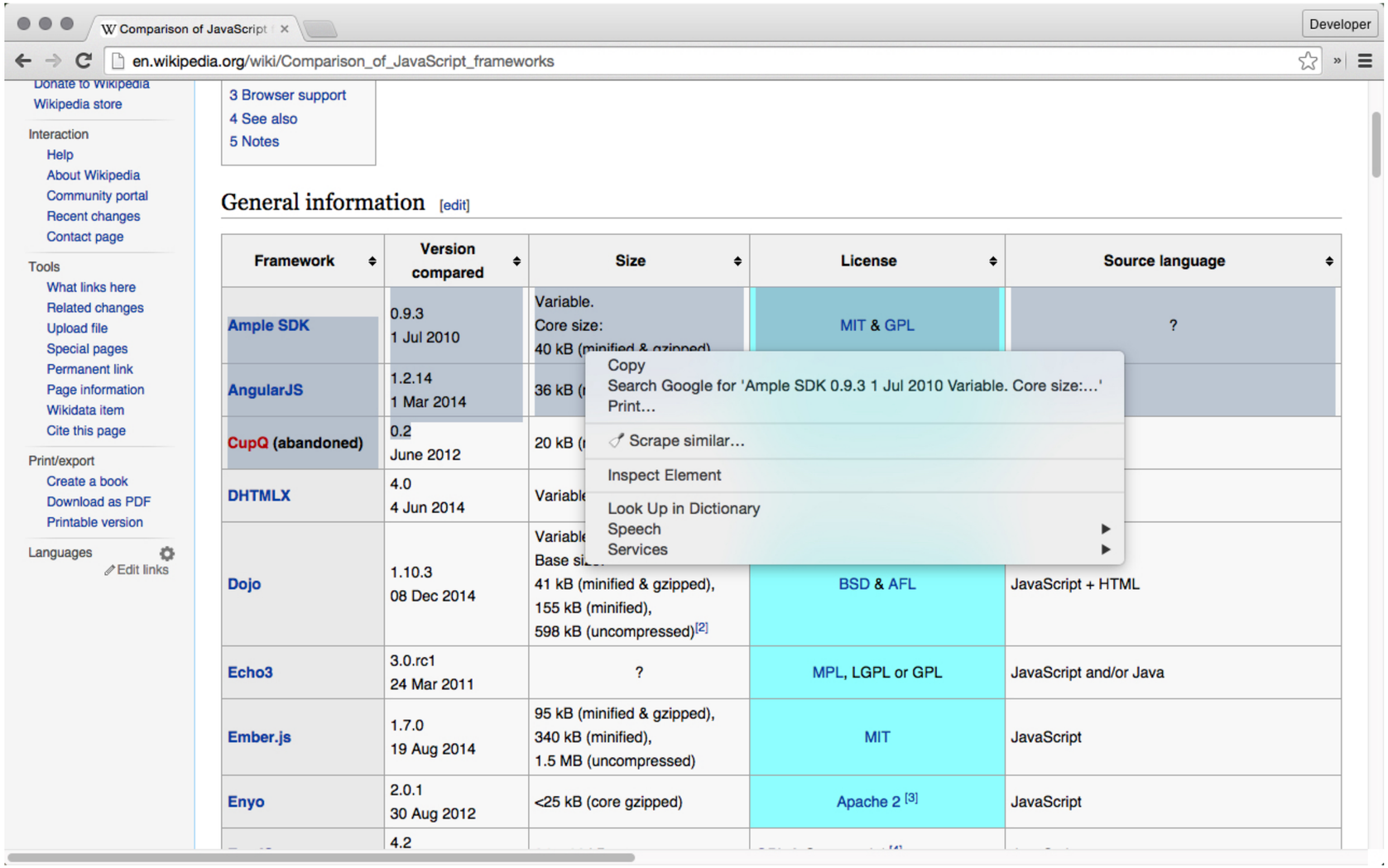
HOME | DOWNLOAD | PURCHASE | TUTORIALS | DOCUMENTATION | FORUM | Contact us



Visual Web Scraping Software with
Macro Recorder and Diagram Designer

Video Tutorials or Download FMiner

Scraper – Chrome Extension



Scrapy Overview

- An open source and collaborative framework for extracting the data you need from websites in a fast, simple, yet extensible way.
- Fast and powerful
  - Write the rules to extract the data. Scrapy does the rest.
- Easily extensible
  - Add new functionality without having to touch the core.
- Portable, Python
  - Runs everywhere Python lives (Windows, Mac, Linux)
- Works with Python 2.7.9

Scrapy is Powerful

- Ran on a Raspberry Pi 2
- Less than 350 lines of code
- Written in less than a day
- Collected 50k profiles in three days

Rules of the Road

Things to Look Out For

- CSS Sprites
  - Image files compressed into a single file
- Honeypot
  - A trap set to detect, deflect, or, in some manner, counteract attempts at unauthorized use of information systems
- IP blocking
  - Blocking web traffic from a specific IP address
- Pages generated by JavaScript
- Captchas
- Logins
- Ad pop-ups

JavaScript Example: Inc 5000

```
<h2 class="sectionhead">Inc. 5000 2014: The Fastest-Growing Private U.S. Companies, At a Glance</h2>
<script src="https://code.angularjs.org/1.2.22/angular.min.js"></script>
<script src="https://code.angularjs.org/1.2.22/angular-animate.min.js"></script>
<script src="http://stage.inc.com/js/Inc5000ListApp.js?UPDATE1"></script>
<script type="text/javascript">
            inc.inc5000listapp = Inc.Apps.Inc5000ListApp();
        </script>
<!-- INC 5000 LIST -->
▼<div id="inc5000app" ng-app="Inc5000List" ng-controller="Inc5000ListCtrl" style="display: inline-block; border-bottom: #000000 7px solid;" class="ng-scope">
  ▶<div id="col-l">…</div>
  ▼<div id="col-r">
    ▶<div id="pagination-bar">…</div>
     <!-- DATA TABLE -->
    ▼<table class="data-container">
       <!-- HEADER ROW -->
      ▼<tbody>
        ▶<tr> </tr>
```

Rules of the Scraping Road

- Always observe a website's terms of service (TOS)
- If indexing a site be sure to read the robots.txt file
  - Indicates which areas of the site should not be processed or scanned
- Throttle your Spider so you don't take down the website
  - Scrapy has auto-throttling
  - Build a timer that waits a random amount of seconds between page requests
- Use VPN software
  - Hide My Ass
- Cycle your user agents
  - A self-identification of a web brower to a website when making a request

Building a Web Scraper with Scrapy

Part One

Build-out Overview

- Specify the data (Item)
- Create the spider (Spider)
- Find the path to the data elements (XPath Selectors)
- Extract the data
- Process the data
- Export to JSON

What We'll Scrape

- Food and nutrition benefits from the US Government: Benefits.gov
  - http://www.benefits.gov/benefits/browse-by-category/category/FOO
- List of 265 food and nutrition programs
- Two types of pages
  - A single list page
  - A detail page for each program

Why This Site?

- It's government data, therefore it's publicly available
- No API ☹
- Could be useful to a non-profit, another government agency or an NGO
- The HTML is well structured
- No gotchas in our way

The Data We'll Save

- List page
  - Program name
  - Link to program details
  - Short description
- Detail page
  - Program title
  - Managing agency and location (state)
  - Program description
  - General program requirements
  - Your next steps
    - Application process
    - Program contact information

Path To The Data

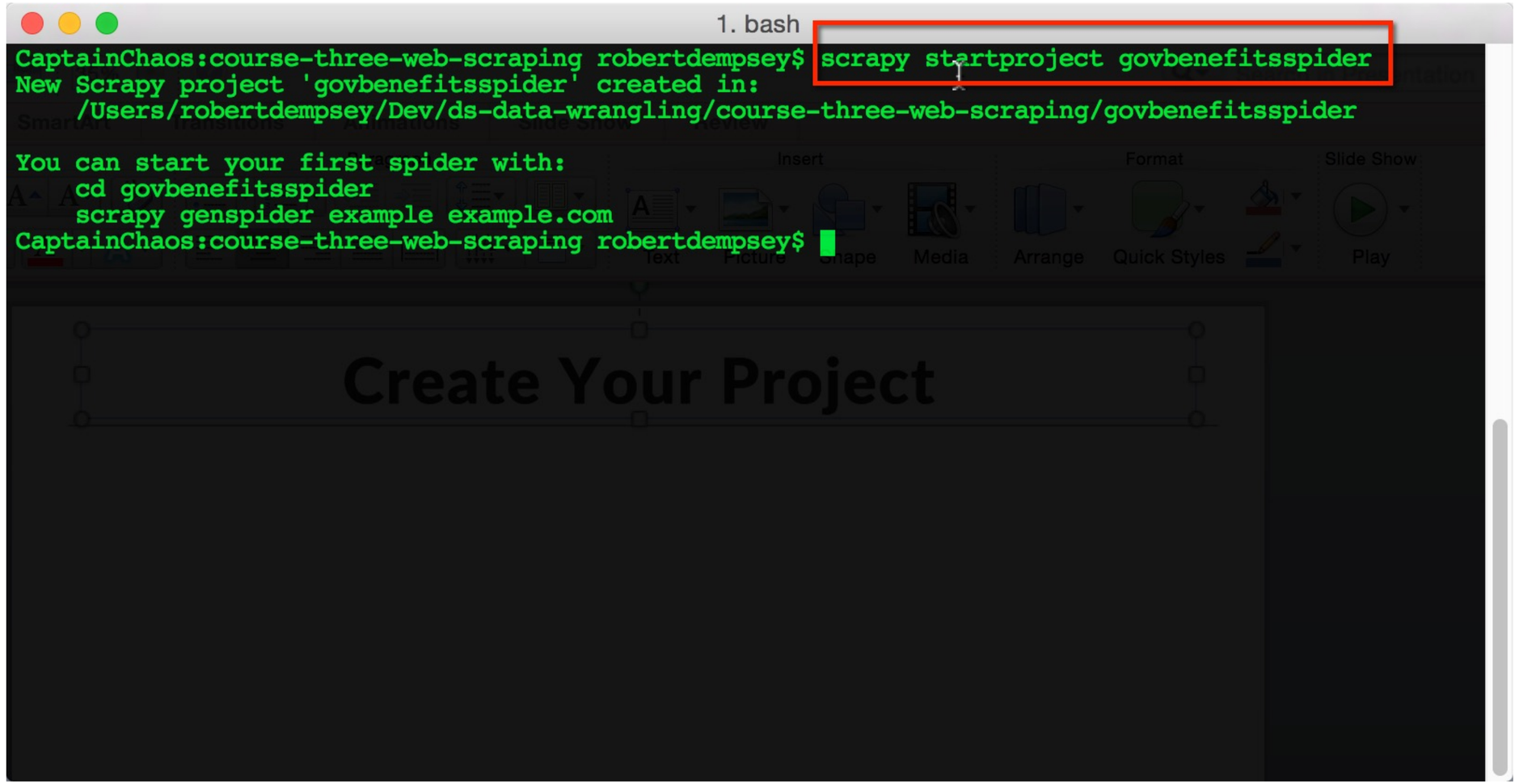
- Write a spider to create a CSV file of programs from the list page
- Write an additional spider to loop through the program pages and extract the data to a CSV

Install Scrapy & Dependencies

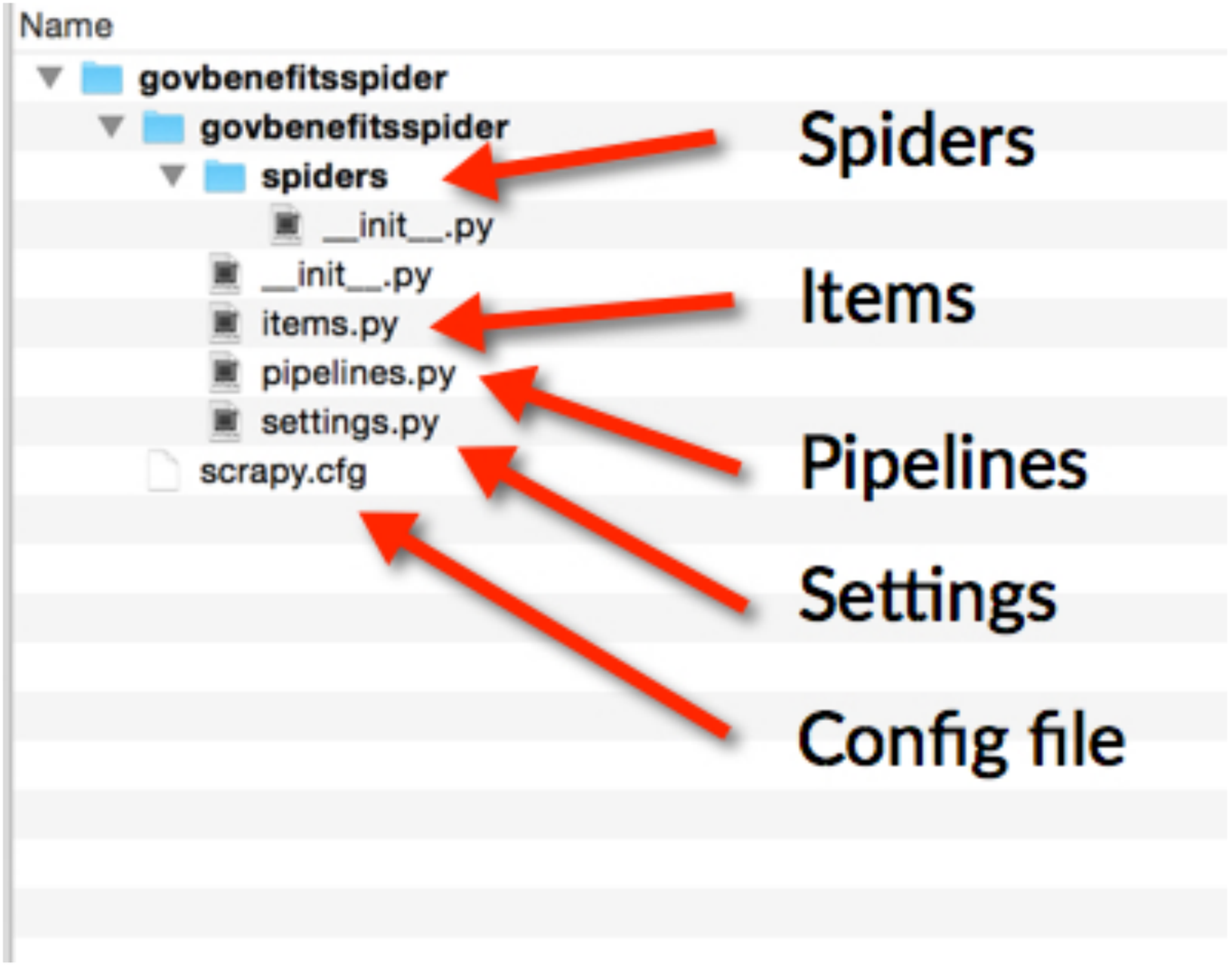
pip install scrapy

pip install service_identity

Create Your Project



Scrapy Project Structure



Define Our Item

```
import scrapy


class BenefitProgramItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    program_title = scrapy.Field()
    program_description = scrapy.Field()
    program_details_link = scrapy.Field()
```

Write Our Spider

```
1  import scrapy
2
3  class BenefitProgramSpider(scrapy.Spider):
4    name = "benefitprograms"
5    allowed_domains = ["benefits.gov"]
6    start_urls = [
7          "http://www.benefits.gov/benefits/browse-by-category/category/F00"
8      ]
9
10   def parse(self, response):
11     filename = response.url.split("/")[-2]
12     with open(filename, 'wb') as f:
13       f.write(response.body)
```

Name the spider

Set the allowed domains

Where should we start?

Save the results to a file named after the URL

Crawl!

scrapy crawl benefitprograms

What Happened

- Scrapy creates scrapy.Request objects for each URL in the start_urls attribute of the Spider, and assigns them the parse method of the spider as their callback function.
- These Requests are scheduled, then executed, and scrapy.http.Response objects are returned and then fed back to the spider, through the parse() method.
- The response (the entire web page in this case) is saved to a file.

Using The Scrapy Shell

scrapy shell
http://www.benefits.gov/benefits/browse-by-category/category/FOO

Commands to try

- response.body
- response.headers
- response.selector

Testing XPath

In [5]: response.xpath("/html/body/div[2]/div[2]/div[2]/div/div[2]/div[4]/div[1]/div/div[1]/span[1]/a")

Out[5]: [<Selector xpath='/html/body/div[2]/div[2]/div[2]/div/div[2]/div[4]/div[1]/div/div[1]/span[1]/a' data=u'<a href="/benefits/benefit-details/1578"'>]

Better

In [6]: response.xpath("/html/body/div[2]/div[2]/div[2]/div/div[2]/div[4]/div[1]/div/div[1]/span[1]/a")**.extract()**

Out[6]: [u'Alabama Food Assistance Program (SNAP)']

Best

In [7]: response.xpath("/html/body/div[2]/div[2]/div[2]/div/div[2]/div[4]/div[1]/div/div[1]/span[1]/a/**text()").extract()**

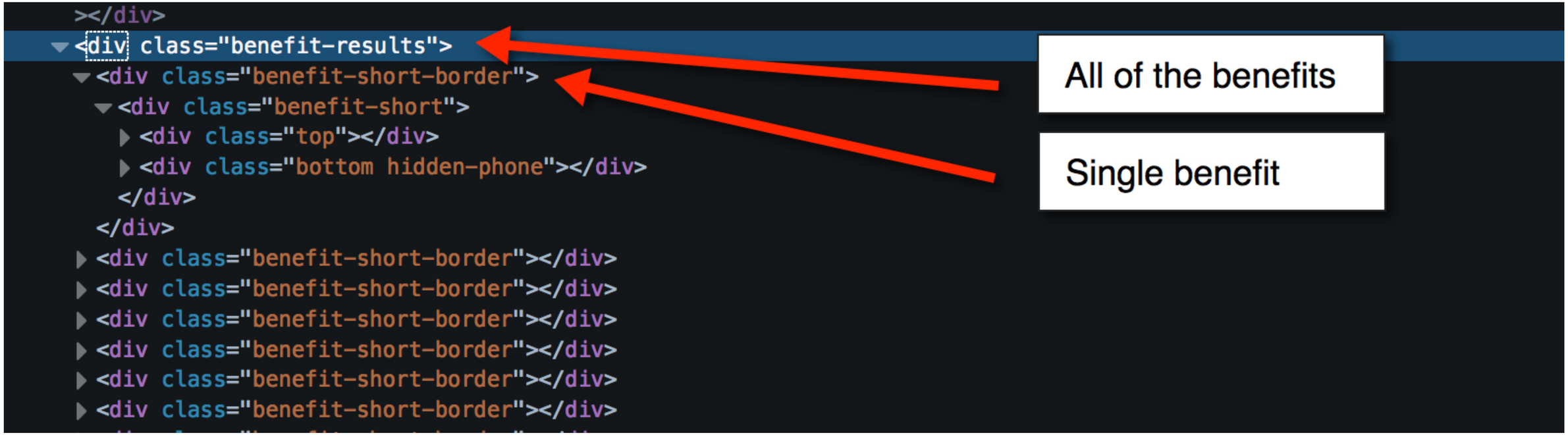
Out[7]: [u'Alabama Food Assistance Program (SNAP)']

Even Better

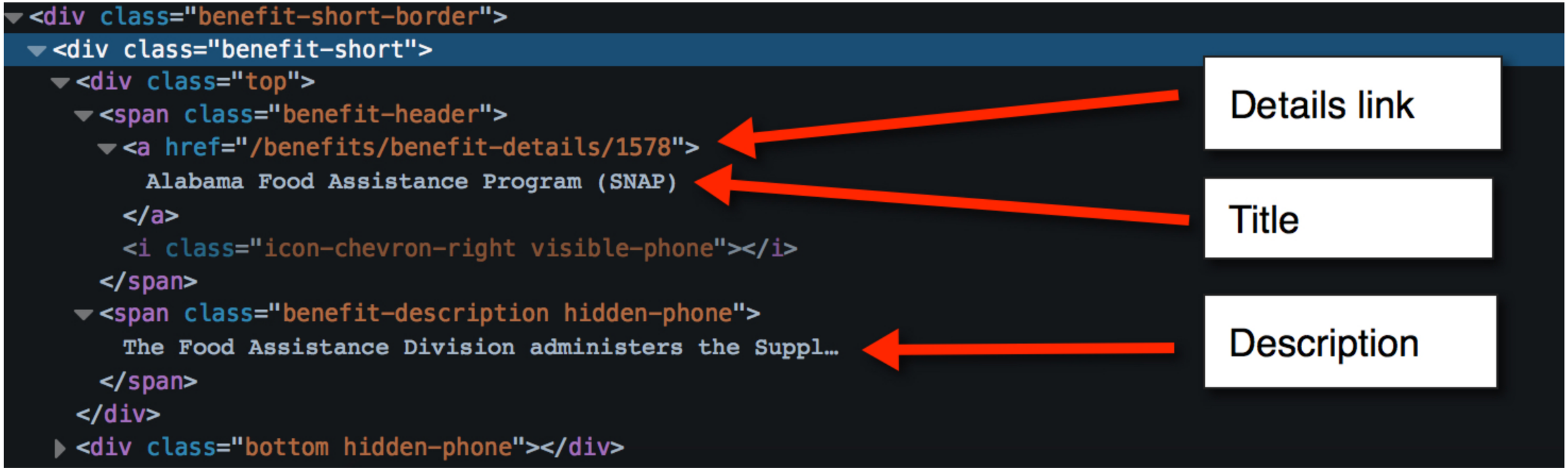
```
In[8]: response.xpath('//span[@class="benefit-header"]/a/text()').extract()

Out[8]:
[u'Alabama Food Assistance Program (SNAP)',
 u'Alabama School Breakfast and Lunch Program',
...
u'Wyoming Supplemental Nutrition Assistance Program (SNAP)']
```

Extracting the Data



Extracting the Data



One-Line Export to JSON

```
scrapy crawl benefitprograms –o program_summaries.json
```

IMPORTANT

This Works

```
programs = sel.xpath('//div[@class="top"]')
items = []

for program in programs:
  item = BenefitProgramItem()
  item['program_title'] = program.xpath(
        'span[@class="benefit-header"]/a/text()').extract()
```

This Doesn’t

```
programs = sel.xpath('//div[@class="top"]')
items = []

for program in programs:
  item = BenefitProgramItem()
  item['program_title'] = program.xpath(
        '//span[@class="benefit-header"]/a/text()').extract()
```

Result


```
[
  {
    "program_title": [
      "Alabama Food Assistance Program (SNAP)"
    ],
    "program_description": [
      "The Food Assistance Division administers the Supplemental Nutrition Assistance Program (SNAP) in Alabama. The Food Assistance Program's purpose is to end hunger and improve nutrition by providing monthly benefits to eligible low income households to ..."
    ],
    "program_details_link": [
      "/benefits/benefit-details/1578"
    ]
  },
...
```

Building a Web Scraper with Scrapy

Part Two

Where We Are

- Built a web scraper to pull the program name, details link, and short description of US Government Food/Nutrition programs.
- Saved the programs to a JSON file.

What's Next

- Parse our existing JSON file.
- Loop through the program links and capture the specifics on each program.
- Save it all to another JSON file.

Build-out Redux

- Specify the data (Item)
- Create the spider (Spider)
- Find the path to the data elements (XPath Selectors)
- Extract the data
- Process the data
- Export to JSON

Following The Rules

Rules We'll Observe

- Throttling your spiders
- Rotate your user agents

Throttling Your Spider

```
# Be nice to the sites we're crawling
AUTOTHROTTLE_ENABLED = True
AUTOTHROTTLE_START_DELAY = 5.0
AUTOTHROTTLE_MAX_DELAY = 60.0
DOWNLOAD_DELAY = 5

# Disable cookies
COOKIES_ENABLED = False
```

Delay between 5 and 60 seconds

Disable cookies

Rotating User Agents

- Pip install fake-useragent
- Add the configuration to your settings.py

Settings.py Additions

```
# Use the fake_useragent python library
# Requires you to: pip install fake-useragent
DOWNLOADER_MIDDLEWARES = {
    'scrapy.contrib.downloadermiddleware.useragent.UserAgentMiddleware': None,
    'govbenefitsspider.middlewares.RandomUserAgentMiddleware': 400,
}
```

Middlewares.py

```
1  import os
2  import random
3  from scrapy.conf import settings
4  from fake_useragent import UserAgent
5
6  class RandomUserAgentMiddleware(object):
7      def __init__(self):
8          super(RandomUserAgentMiddleware, self).__init__()
9          self.ua = UserAgent()
10
11     def process_request(self, request, spider):
12         request.headers.setdefault('User-Agent', self.ua.random)
```

Robert Dempsey

robertwdempsey.com

rdempsey

robertwdempsey

rdempsey