XPath Navigation

WEB SCRAPING IN PYTHON

Thomas Laetsch
Data Scientist, NYU

datacamp

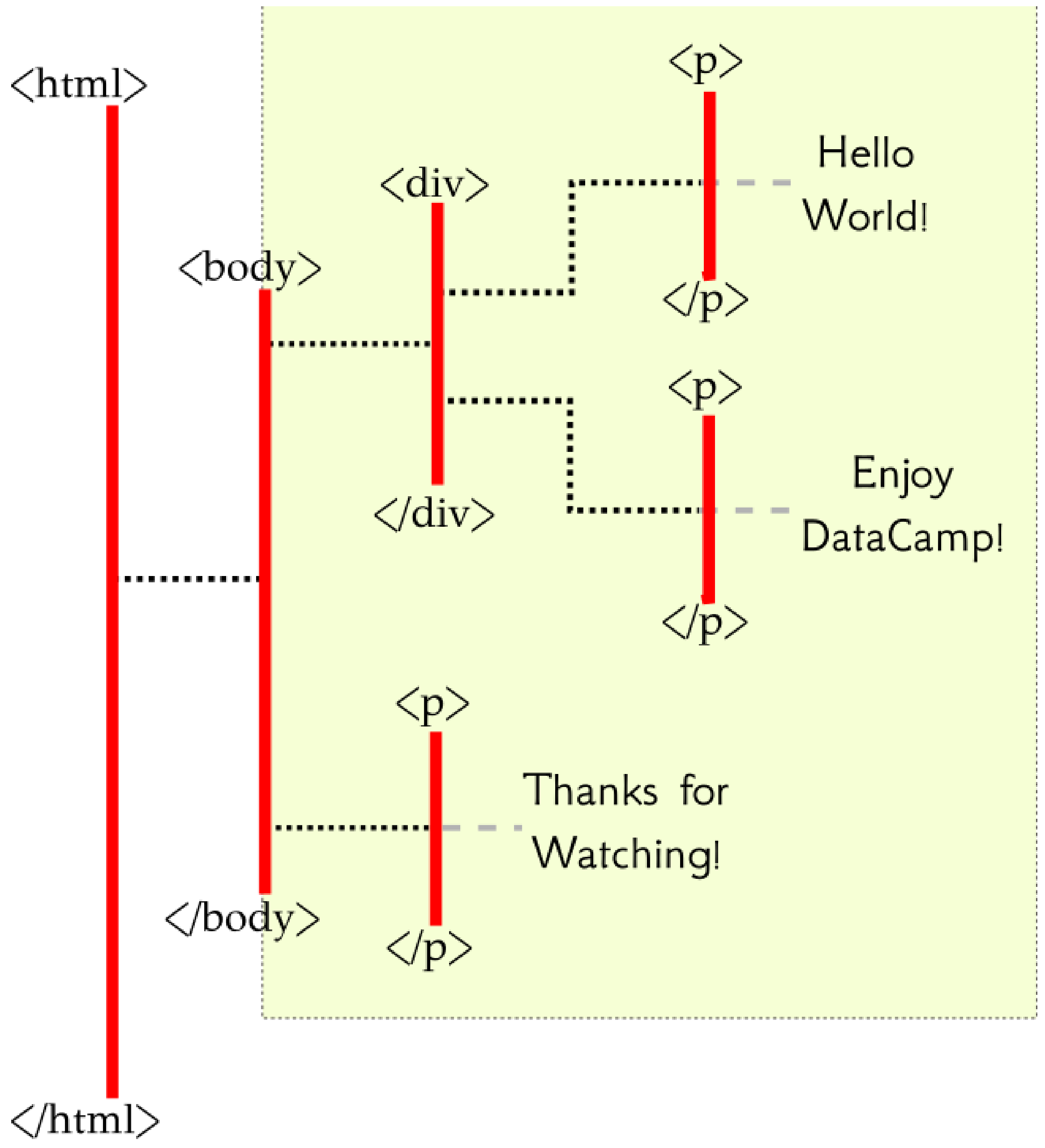
Slashes and Brackets

- Single forward slash / looks forward **one** generation
- Double forward slash // looks forward **all** future generations
- Square brackets [] help narrow in on specific elements

but it turns out these brackets aren't always necessary nor desired.

Note: Since "/html/body/*" selects all elements one generation below the body element without concern of the tag type, it selects all children of the body element. On the other hand, "/html/body//*" selects all elements from all future generations of the body element (that is, all descendants of the body) regardless of tag type.

To Bracket or not to Bracket



```
xpath = '/html/body'
```

```
xpath = '/html[1]/body[1]'
```

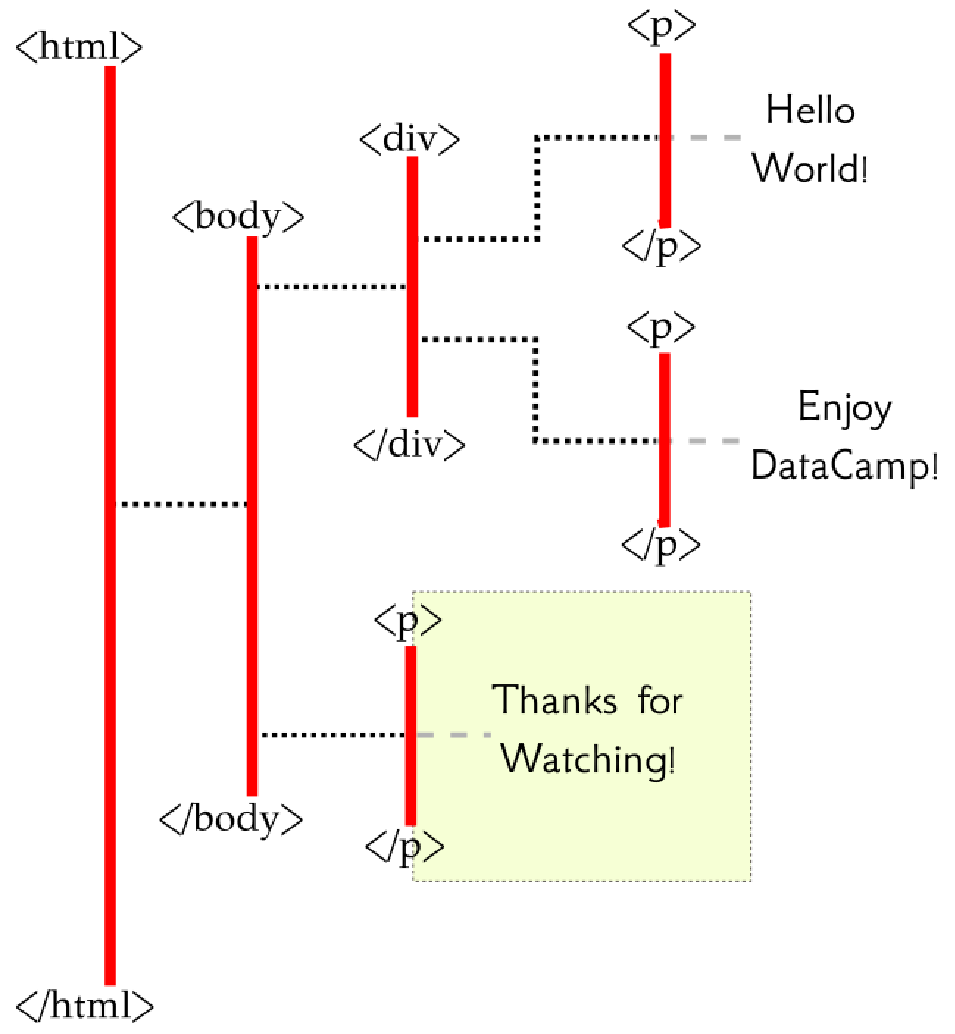
- Give the same selection

3. To Bracket or not to Bracket
For example, the first XPath expression without brackets and the second with brackets lead us to the same element the body element since there is only one html element at the root level, and one body element which is a child of that html element.

A Body of P

```
xpath = '/html/body/p'
```

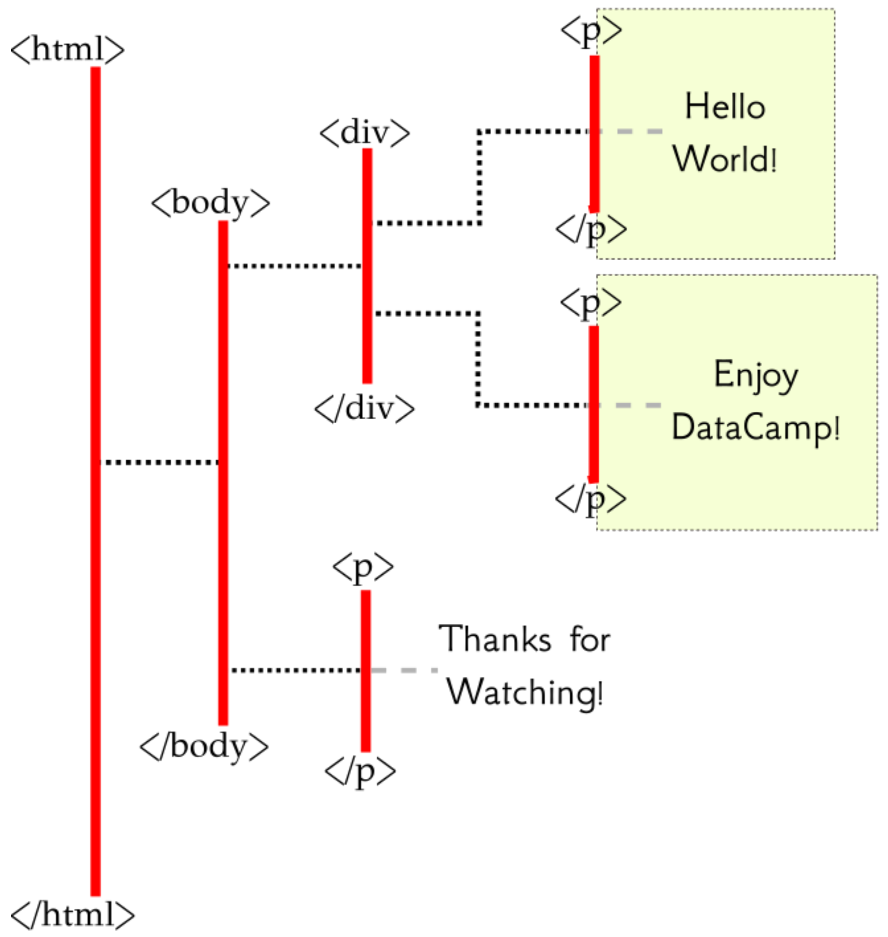
4. A Body of P
As another example, the XPath string here leads us to the single paragraph element which is one generation below the body element. Do you understand why there is only this one element selected? It's because there is only one paragraph element which is a child of the body element!

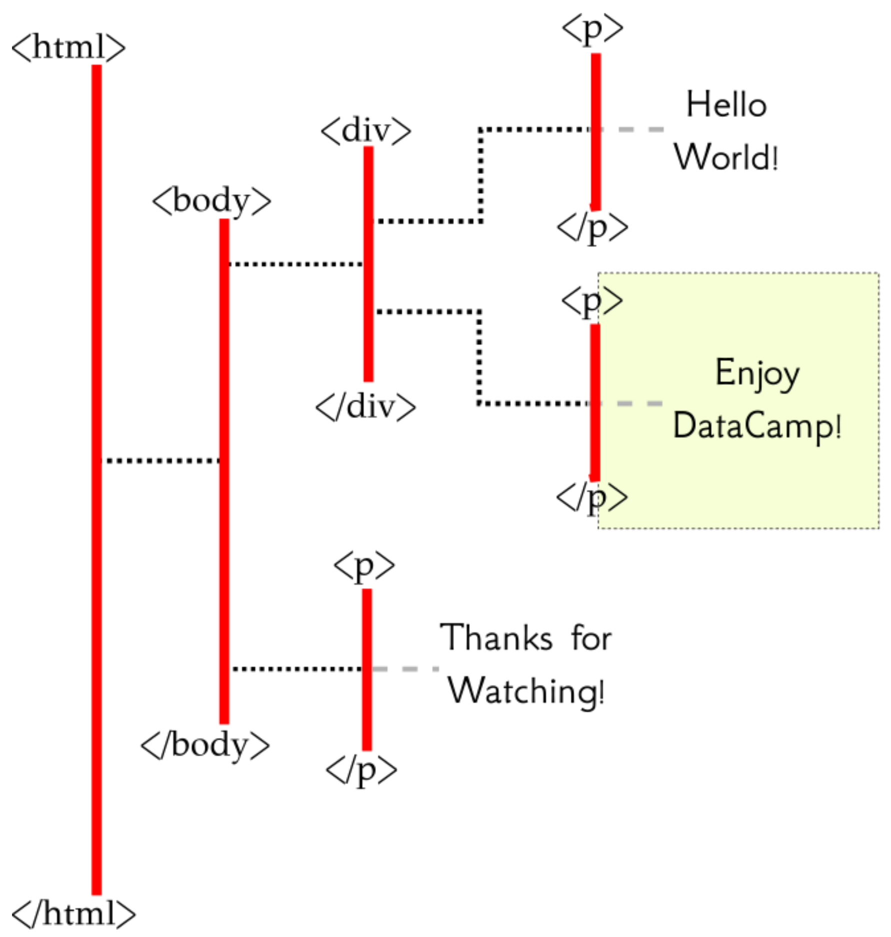


The Birds and the Ps

```
xpath = '/html/body/div/p'
```

```
xpath = '/html/body/div/p[2]'
```



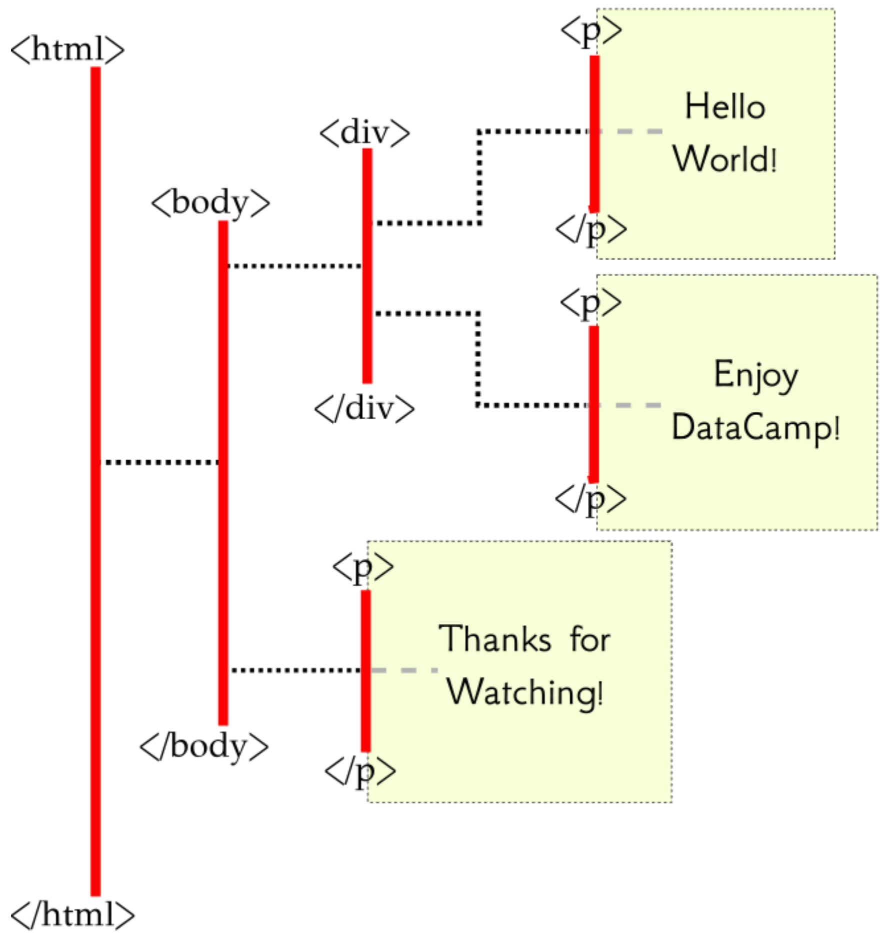


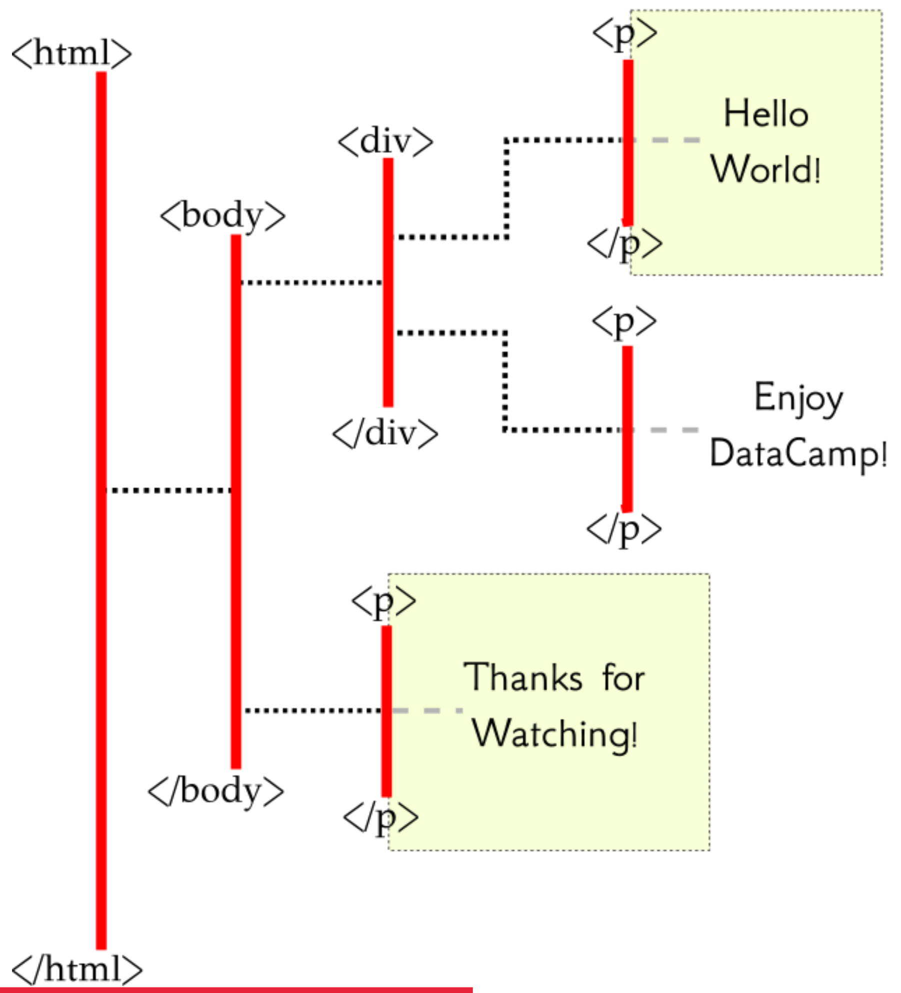
5. The Birds and the Ps
On the other hand, if we consider the XPath string here, this directs us to both paragraph children of the div element. We haven't clarified a specific one. We could, of course, add in brackets to help narrow in on which element we want; in this case, we will be selecting the second paragraph child of the div element.

Double Slashing the Brackets

```
xpath = '//p'
```

```
xpath = '//p[1]'
```



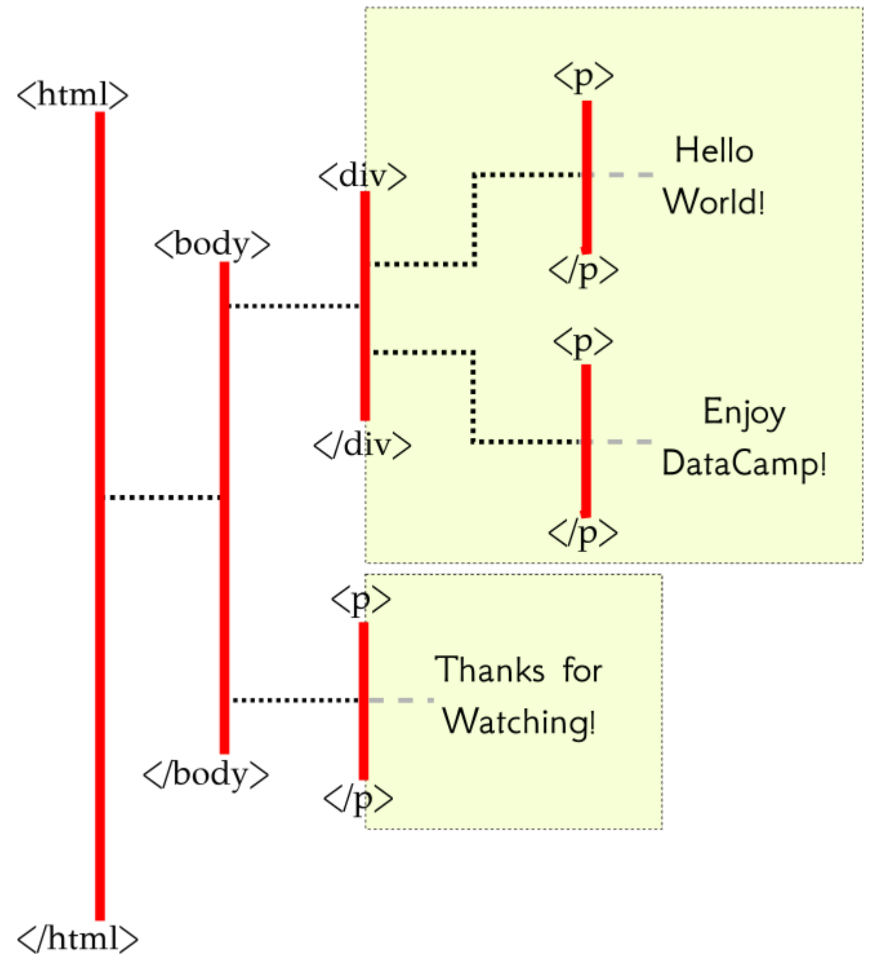


... Honestly, I don't often mix double forward-slashes and brackets filled with numbers. We'll see in later slides there are other, more interesting ways to use brackets to select elements.

The Wildcard

```
xpath = '/html/body/*'
```

- The asterisks * is the "wildcard"



7. The Wildcard
The asterisks indicates we want to ignore tag type. For example, in this expression, we are directed to both children of the body element, regardless that one is a div element and one is a paragraph element.

Xposé

WEB SCRAPING IN PYTHON

8. Xposé
To summarize, we have learned the basics of XPath notation, including the meaning of single and double forward slashes, as well as some uses for the square brackets; and finally you saw the asterisks as the "wildcard" character. You will become more familiar with these ideas in the exercises!

Off the Beaten XPath

WEB SCRAPING IN PYTHON

Thomas Laetsch
Data Scientist, NYU

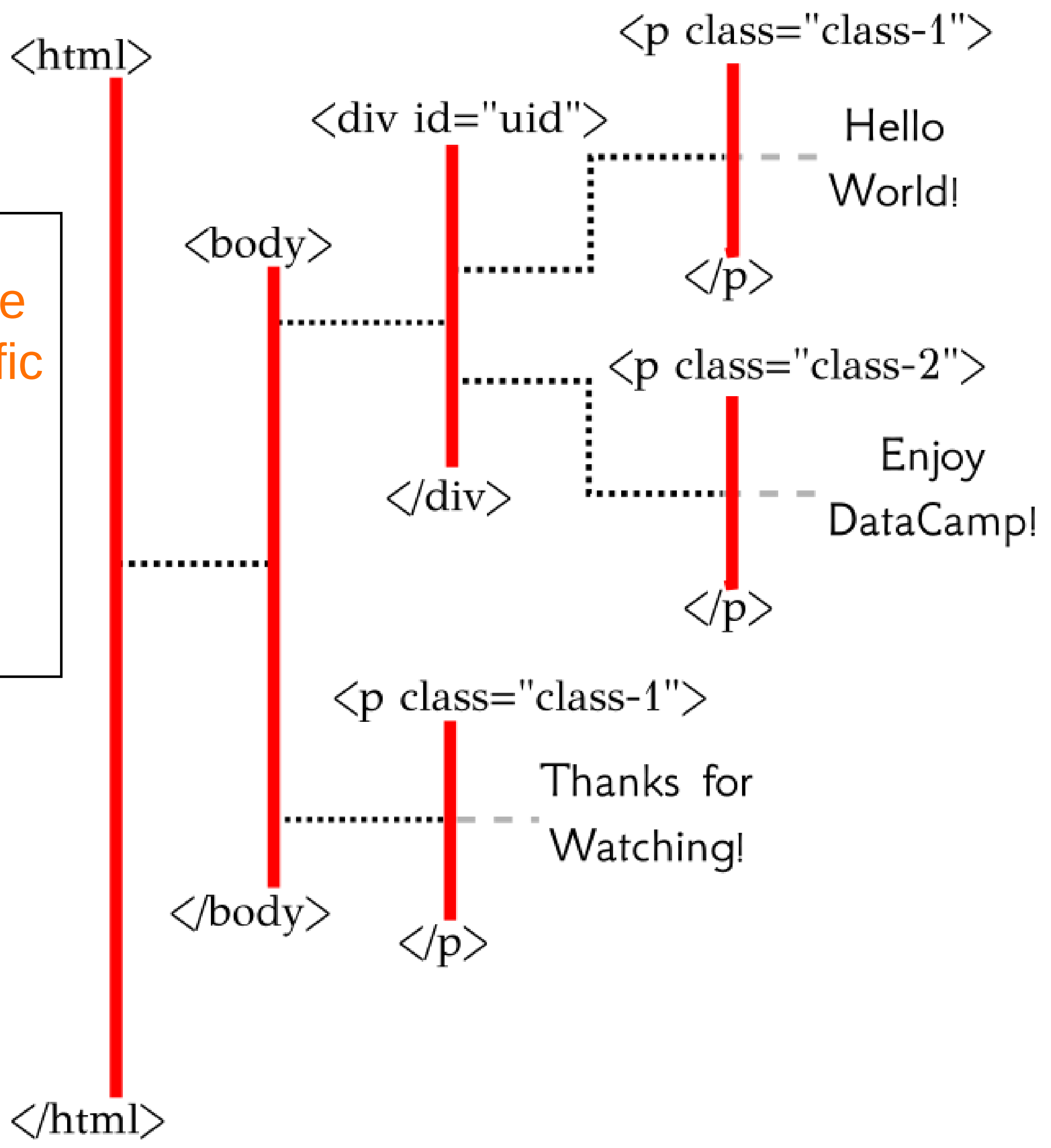
(At)tribute

- `@` represents "attribute"
  - `@class` ←
  - `@id` ←
  - `@href` ←

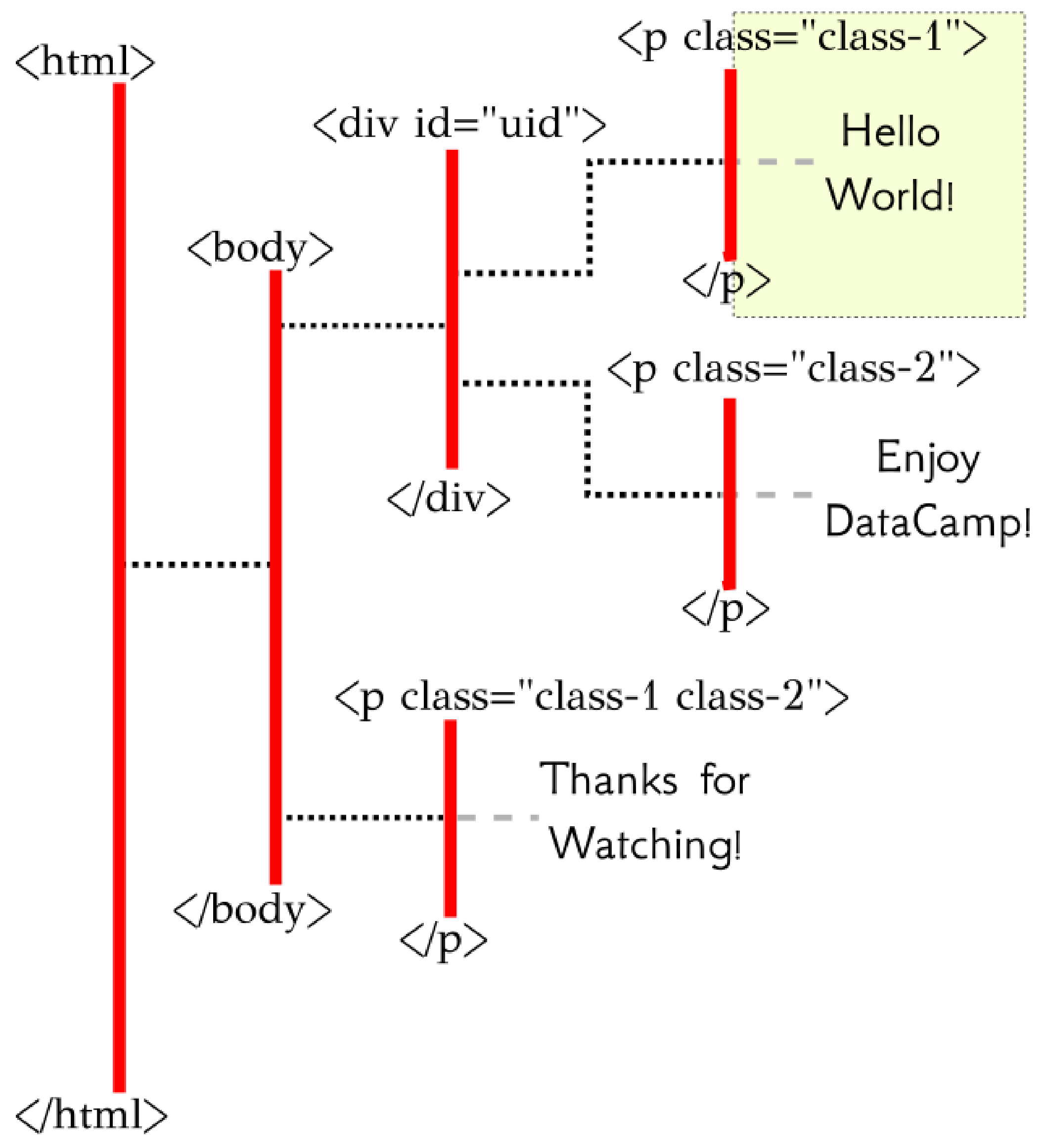
2. (At)tribute
Let's start by first pointing out that in XPath notation, the @ symbol is used to distinguish attributes. For example, if we see @class, @id, or @href, in the XPath expression, then it is referring to a class attribute, id attribute, or href attribute, respectively.

Brackets and Attributes

3. Brackets and Attributes
We saw before that square brackets can be used in xpath syntax to hone in on a specific element or elements based on their order within a given generation. We can also include other information within square brackets to select specific elements.



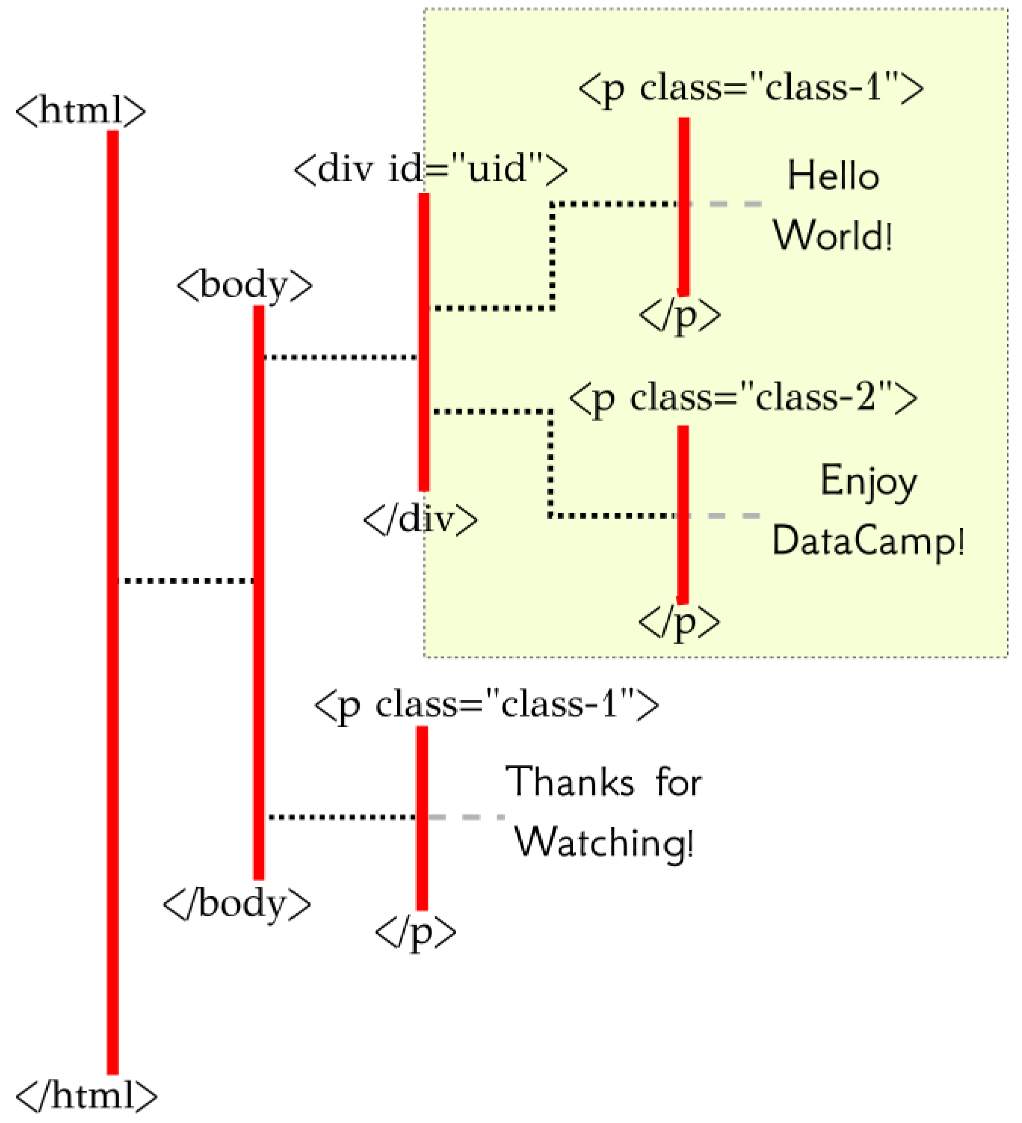
Brackets and Attributes



```
xpath = '//p[@class="class-1"]'
```

4. Brackets and AttributesßFor example, the XPath string here will direct to all paragraph elements from //p, and then reduce down to all those whose class attribute is equal to "class-1". Note that we have the class attribute in quotations. Now, my convention is to use single quotes to define the XPath string, and double quotes as needed within the XPath expression itself.

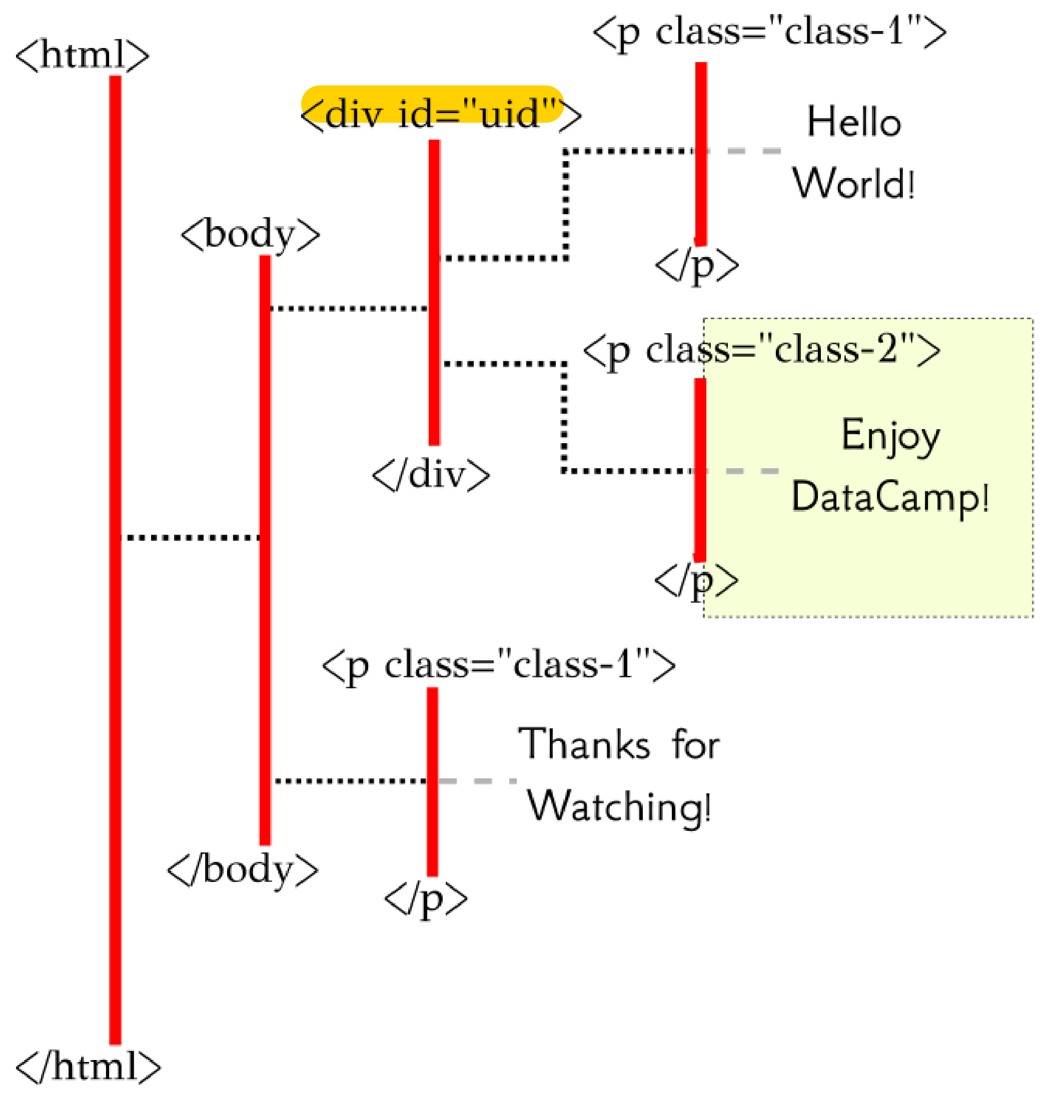
Brackets and Attributes



```
xpath = '//*[@id="uid"]'
```

5. Brackets and Attributes
As another example, we could use the expression here with the wildcard character to first direct to all elements within the HTML document, and reduce down to whichever element has "uid" as its id attribute.

Brackets and Attributes



```
xpath = '//div[@id="uid"]/p[2]'
```

6. Brackets and Attributes
Or, we could combine what we know to first navigate to the div element with id attribute equal to "uid", and then collect the second paragraph child of that div element.

Content with Contains

Xpath Contains Notation:

contains(@attri-name, "string-expr")

7. Content with Contains
A useful tool we can include within our square-bracketed expression is the "contains" function. The format of the "contains" function is given abstractly here, with the left argument containing the attribute name (including the at symbol), and the right argument is the string expression we want to search for within the given attribute. What it does is searches the attributes of that specific attribute name and matches with those where the string expression is a sub-string of the full attribute.

Contain This

```
xpath = '//*[contains(@class,"class-1")]'
```

✓ `<p class="class-1"> ... </p>`

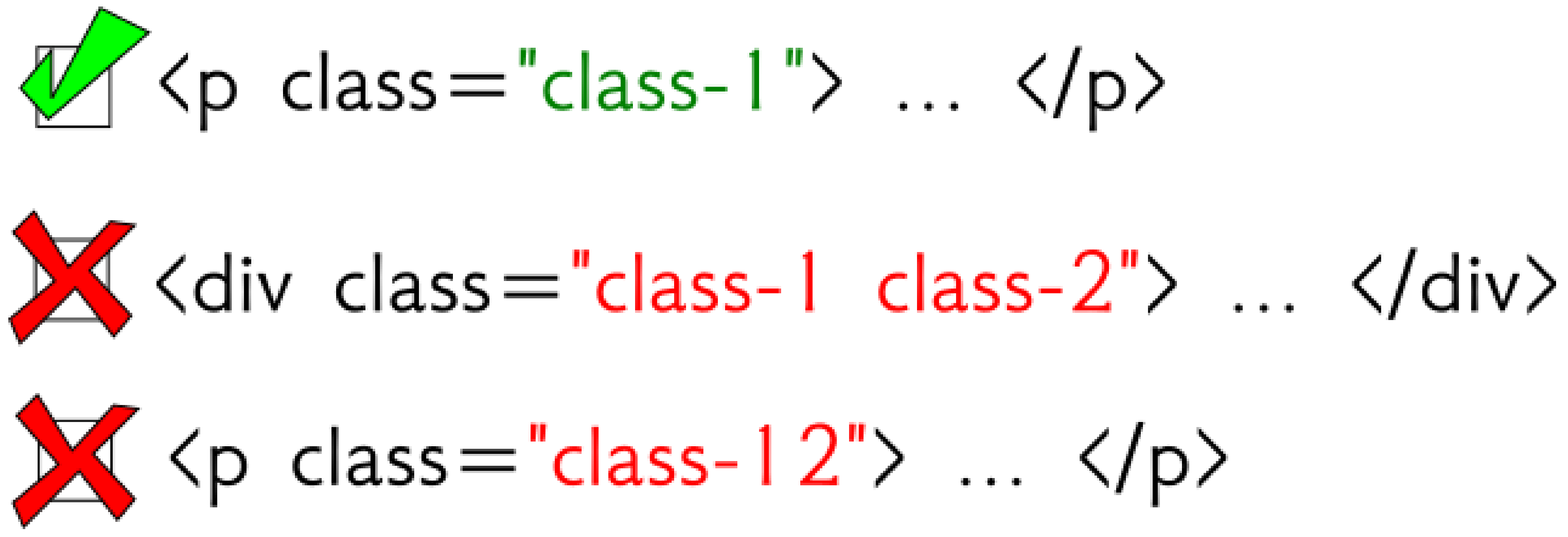
✓ `<div class="class-1 class-2"> ... </div>`

✓ `<p class="class-12"> ... </p>`

8. Contain This
To make this clearer, let's look at an example. The expression here will choose all elements in which the string "class-1" is contained as a substring within the full class attribute; this even includes the third paragraph belonging to class-12, because class-1 is a substring of class-12.

Contain This

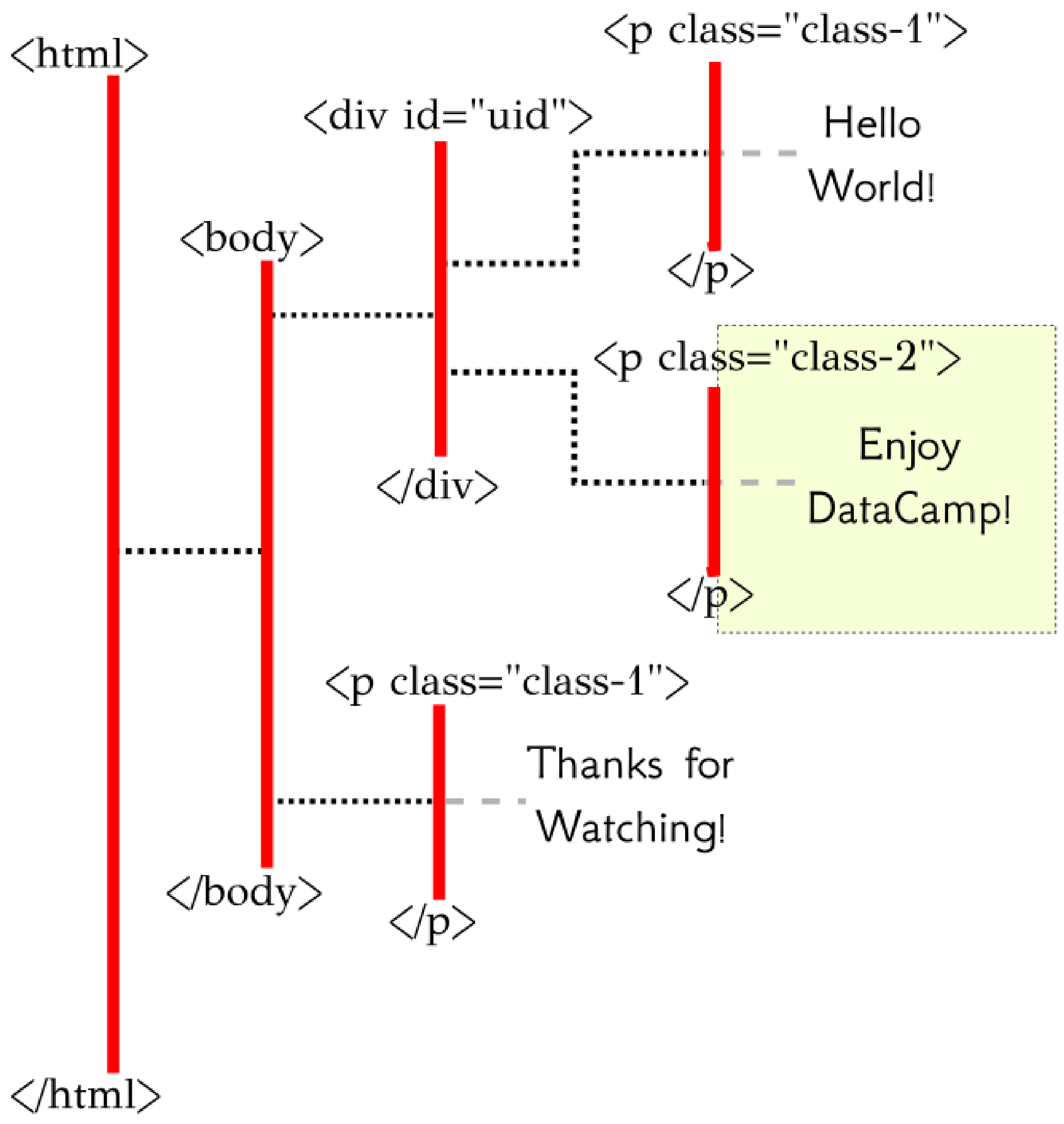
```
xpath = '//*[@class="class-1"]'
```



```
✓ <p class="class-1"> ... </p>
✗ <div class="class-1 class-2"> ... </div>
✗ <p class="class-12"> ... </p>
```

9. Contain This
The last example differs from what we've seen so far since the expression here without the contains function only matches elements whose entire class attribute is equal to "class-1".

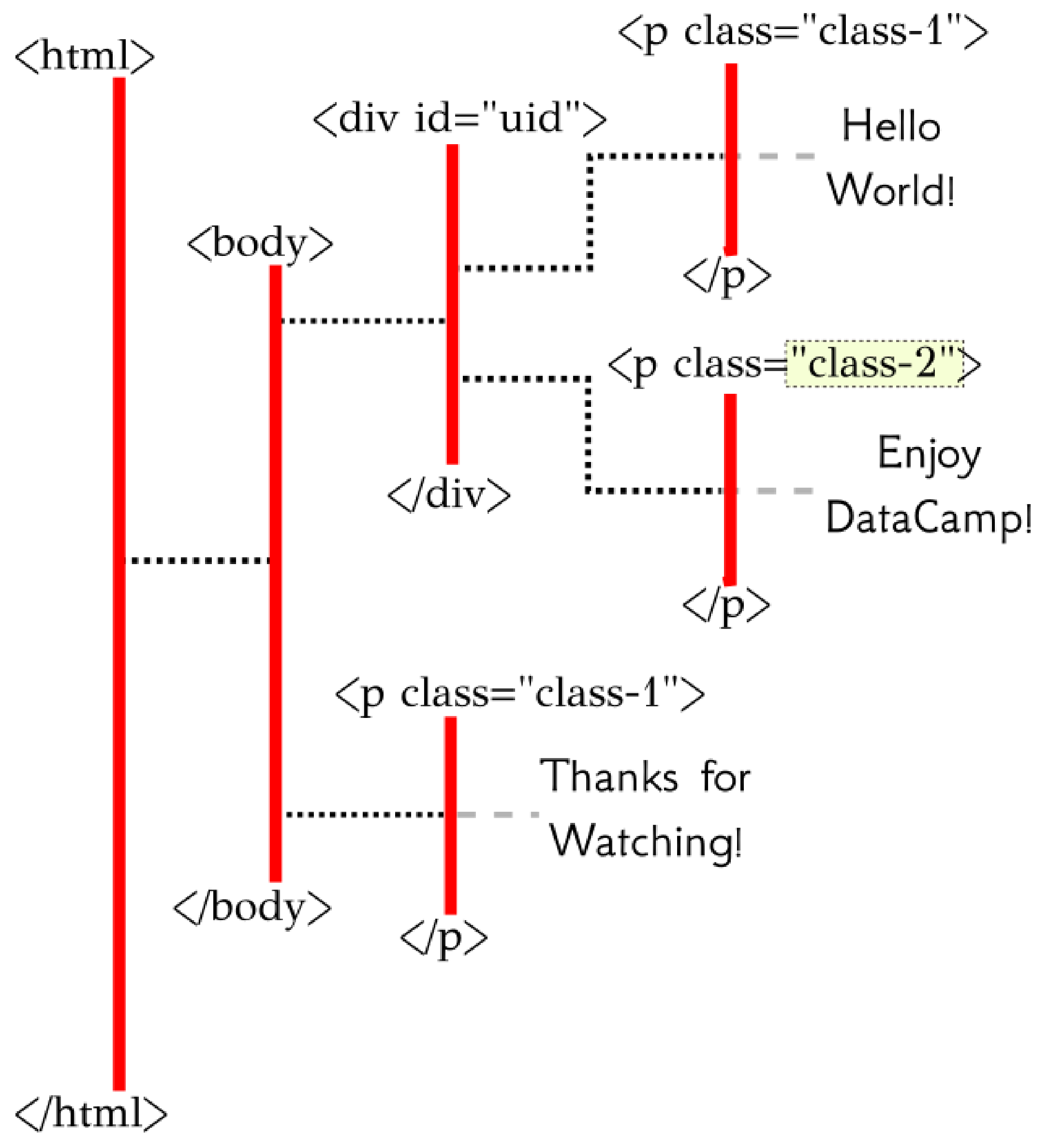
Get Classy



```
xpath = '/html/body/div/p[2]'
```

10. Get Classy
Now, let's consider how to direct to the attribute information itself. To do so, we first create an XPath expression to the element or elements we want to pull out some attribute information from. Say, we would like to direct to the class attribute of this highlighted paragraph element. We already know how to direct to the highlighted area.

Get Classy



```
xpath = '/html/body/div/p[2]/@class'
```

11. Get Classy
To direct to the attribute itself, we take the XPath, follow it by a forward slash, and follow that by the @ symbol connected to the attribute name of interest, in this case, class. As a quick note. If we were instead to use a **double forward slash before the @ symbol** with the attribute name, we would not only direct to the attribute of the elements selected in the XPath, but also all of those attributes in their future generations too.

End of the Path

WEB SCRAPING IN PYTHON

Introduction to the scrapy Selector

WEB SCRAPING IN PYTHON

Thomas Laetsch
Data Scientist, NYU

1. Introduction to the scrapy Selector
In this lesson we will begin to familiarize ourselves with scrapy's Selector object, so named because it is the scrapy object used to select portions of the HTML using XPath (or a so-called CSS Locator -- something we'll learn about later). Some of what we see in this lesson may seem a bit gnarly. But keep in mind that once we master this step, we will have already learned how to use the main parsing tool scrapy offers, letting us actually read in an HTML document and access the inner elements we want.

Setting up a Selector

```
from scrapy import Selector
```

```
html = '''
<html>
  <body>
    <div class="hello datacamp">
      <p>Hello World!</p>
    </div>
    <p>Enjoy DataCamp!</p>
  </body>
</html>
'''
```

2. Setting up a Selector
Through this lesson, we will be using the Selector we set up in this slide as our running example. We will import Selector from scrapy. We have made a string of HTML, which we pass to the Selector as text, creating a selector object "sel", which is the object we'll be learning to use. It will become clearer as we move along, but we can think of the Selector "sel" as having "selected" the entire HTML document. Before moving on, let's note that the html has two paragraph elements, the first saying "Hello World!"; the second saying "Enjoy DataCamp!".

```
sel = Selector( text = html )
```

- Created a scrapy Selector object using a string with the html code
- The selector `sel` has selected the **entire** html document

Selecting Selectors

- We can use the `xpath` call within a `Selector` to create new `Selector` s of specific pieces of the html code
- The return is a `SelectorList` of `Selector` objects

```
sel.xpath("//p")
# outputs the SelectorList:
[<Selector xpath='//p' data='<p>Hello World!</p>'>,
 <Selector xpath='//p' data='<p>Enjoy DataCamp!</p>'>]
```

3. Selecting Selectors
To put to work all the XPath notation we've built-up, we can call the xpath Selector method to create new Selector objects, selecting the pieces of HTML we are interested in. When doing so, the return value is a SelectorList (a list with some "scrapy extras") containing new Selector objects. For example, if we use an xpath to select all paragraph objects from our running example, we will have a SelectorList of two Selector objects, one for each of the paragraphs.

Extracting Data from a SelectorList

4. Extracting Data from a SelectorList
Selectors and SelectorLists are nice, but at the end of the day, we really want to access the data inside the Selector or SelectorList. We can do this easily by using the "extract" method. For a SelectorList, by calling the extract method, we are left with a list of strings, where each of the strings is the data from the Selectors which were originally in the SelectorList. If we only want the data (as a string) from the first element of a SelectorList, we can also call the quick extract_first method.

- Use the `extract()` method

```
>>> sel.xpath("//p")
out: [<Selector xpath='//p' data='<p>Hello World!</p>'>,
      <Selector xpath='//p' data='<p>Enjoy DataCamp!</p>'>]
```

```
>>> sel.xpath("//p").extract()
out: [ '<p>Hello World!</p>',
       '<p>Enjoy DataCamp!</p>' ]
```

- We can use `extract_first()` to get the first element of the list

```
>>> sel.xpath("//p").extract_first()
out: '<p>Hello World!</p>'
```

XPath Chaining
Selector and SelectorList objects allow for chaining when using the xpath method. What this means is that you can apply the xpath method over once you've already applied it. For example, if sel is the name of our Selector, then

Extracting Data from a Selector

```
ps = sel.xpath('//p')
second_p = ps[1]
```

```
second_p.extract()
out: '<p>Enjoy DataCamp!</p>'
```

5. Extracting Data from a Selector
Although extract_first is convenient if you want the first piece of data in a SelectorList, we could grab the data from any other Selector within our SelectorList. To give an example of this, lets say we create a SelectorList named ps, then take the second Selector in the list (remembering that python indexes lists starting at 0, so we use the index 1 rather than 2). Then we apply the extract function to this Selector. Note that a Selector only has one piece of data, so extract leaves us with the string of this data (rather than a list of strings as was the case with SelectorLists).

Select This Course!

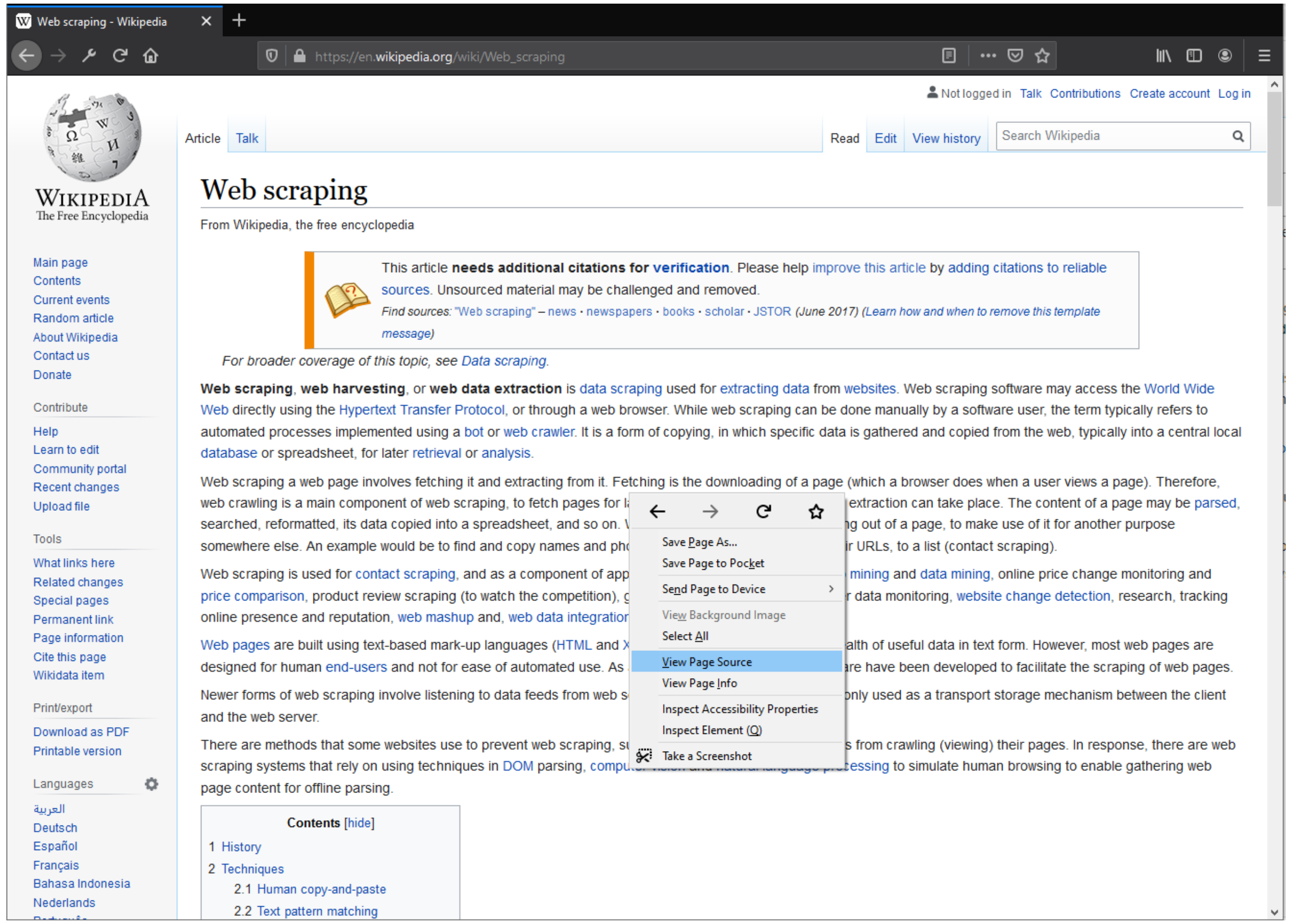
WEB SCRAPING IN PYTHON

"Inspecting the HTML"

WEB SCRAPING IN PYTHON

Thomas Laetsch, PhD
Data Scientist, NYU

"Source" = HTML Code



Web scraping

From Wikipedia, the free encyclopedia

This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed.
Find sources: "Web scraping" – news · newspapers · books · scholar · JSTOR *(June 2017) (Learn how and when to remove this template message)*

For broader coverage of this topic, see Data scraping.

Web scraping, **web harvesting**, or **web data extraction** is data scraping used for extracting data from websites. Web scraping software may access the World Wide Web directly using the Hypertext Transfer Protocol, or through a web browser. While web scraping can be done manually by a software user, the term typically refers to automated processes implemented using a bot or web crawler. It is a form of copying, in which specific data is gathered and copied from the web, typically into a central local database or spreadsheet, for later retrieval or analysis.

Web scraping a web page involves fetching it and extracting from it. Fetching is the downloading of a page (which a browser does when a user views a page). Therefore, web crawling is a main component of web scraping, to fetch pages for l extraction can take place. The content of a page may be parsed, searched, reformatted, its data copied into a spreadsheet, and so on. V ng out of a page, to make use of it for another purpose somewhere else. An example would be to find and copy names and ph ir URLs, to a list (contact scraping).

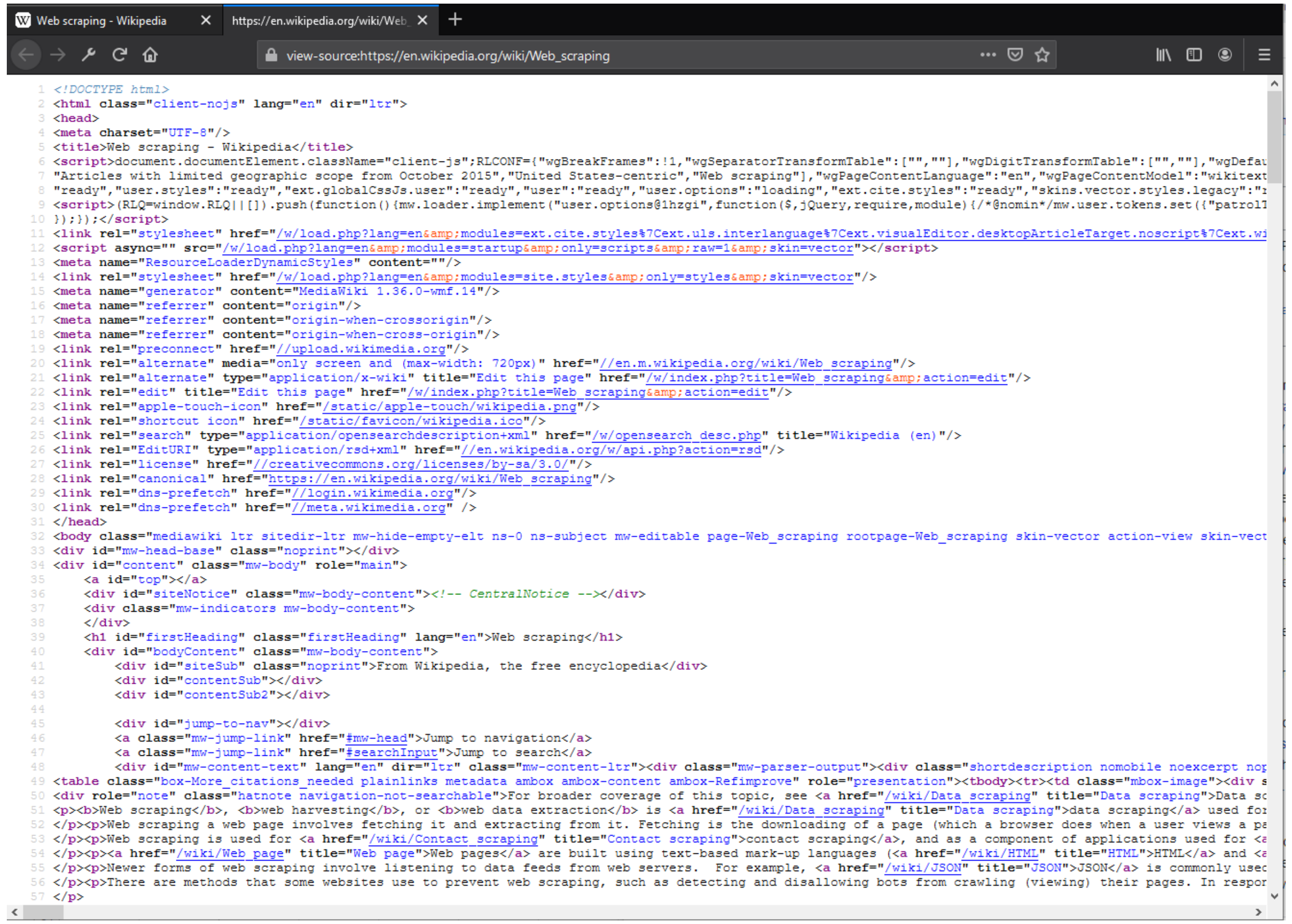
Web scraping is used for contact scraping, and as a component of app mining and data mining, online price change monitoring and price comparison, product review scraping (to watch the competition), g r data monitoring, website change detection, research, tracking online presence and reputation, web mashup and, web data integration

Web pages are built using text-based mark-up languages (HTML and X alth of useful data in text form. However, most web pages are designed for human end-users and not for ease of automated use. As re have been developed to facilitate the scraping of web pages.

Newer forms of web scraping involve listening to data feeds from web s only used as a transport storage mechanism between the client and the web server.

There are methods that some websites use to prevent web scraping, s s from crawling (viewing) their pages. In response, there are web scraping systems that rely on using techniques in DOM parsing, compu essing to simulate human browsing to enable gathering web page content for offline parsing.

Contents [hide]
1 History
2 Techniques
2.1 Human copy-and-paste
2.2 Text pattern matching

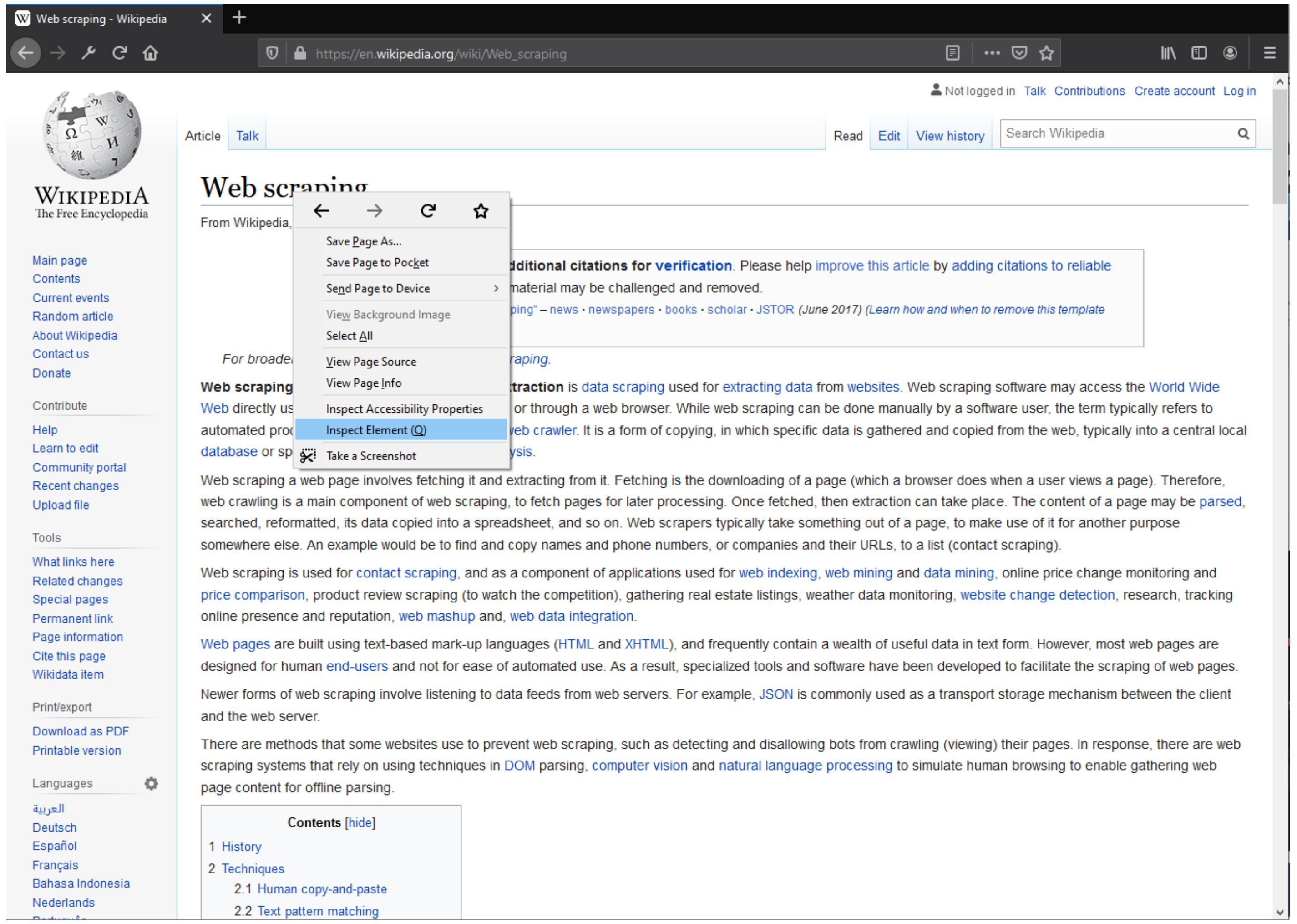


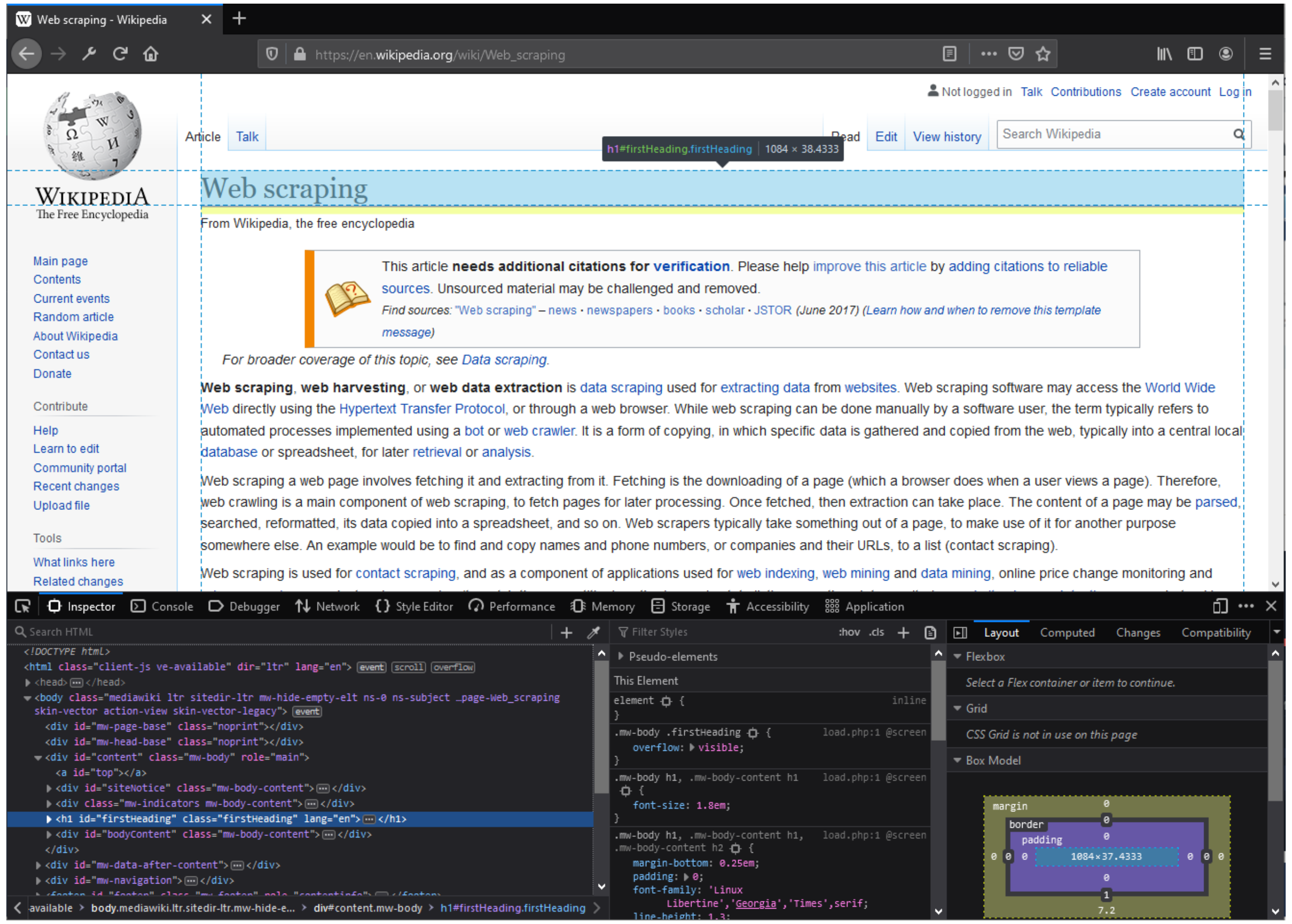
Web scraping - Wikipedia
https://en.wikipedia.org/wiki/Web_
view-source:https://en.wikipedia.org/wiki/Web_scraping

```
<!DOCTYPE html>
<html class="client-nojs" lang="en" dir="ltr">
<head>
<meta charset="UTF-8"/>
<title>Web scraping - Wikipedia</title>
<script>document.documentElement.className="client-js";RLCONF={"wgBreakFrames":!1,"wgSeparatorTransformTable":["",""],"wgDigitTransformTable":["",""],"wgDefau
"Articles with limited geographic scope from October 2015","United States-centric","Web scraping"],"wgPageContentLanguage":"en","wgPageContentModel":"wikitext
"ready","user.styles":"ready","ext.globalCssJs.user":"ready","user":"ready","user.options":"loading","ext.cite.styles":"ready","skins.vector.styles.legacy":"r
<script>(RLQ=window.RLQ||[]).push(function(){mw.loader.implement("user.options@1hzgi",function($,jQuery,require,module){/*@nomin*/mw.user.tokens.set({"patrolT
});});</script>
<link rel="stylesheet" href="/w/load.php?lang=en&amp;modules=ext.cite.styles%7Cext.uls.interlanguage%7Cext.visualEditor.desktopArticleTarget.noscript%7Cext.wi
<script async="" src="/w/load.php?lang=en&amp;modules=startup&amp;only=scripts&amp;raw=1&amp;skin=vector"></script>
<meta name="ResourceLoaderDynamicStyles" content=""/>
<link rel="stylesheet" href="/w/load.php?lang=en&amp;modules=site.styles&amp;only=styles&amp;skin=vector"/>
<meta name="generator" content="MediaWiki 1.36.0-wmf.14"/>
<meta name="referrer" content="origin"/>
<meta name="referrer" content="origin-when-crossorigin"/>
<meta name="referrer" content="origin-when-cross-origin"/>
<link rel="preconnect" href="//upload.wikimedia.org"/>
<link rel="alternate" media="only screen and (max-width: 720px)" href="//en.m.wikipedia.org/wiki/Web_scraping"/>
<link rel="alternate" type="application/x-wiki" title="Edit this page" href="/w/index.php?title=Web_scraping&amp;action=edit"/>
<link rel="edit" title="Edit this page" href="/w/index.php?title=Web_scraping&amp;action=edit"/>
<link rel="apple-touch-icon" href="/static/apple-touch/wikipedia.png"/>
<link rel="shortcut icon" href="/static/favicon/wikipedia.ico"/>
<link rel="search" type="application/opensearchdescription+xml" href="/w/opensearch_desc.php" title="Wikipedia (en)"/>
<link rel="EditURI" type="application/rsd+xml" href="//en.wikipedia.org/w/api.php?action=rsd"/>
<link rel="license" href="//creativecommons.org/licenses/by-sa/3.0/"/>
<link rel="canonical" href="https://en.wikipedia.org/wiki/Web_scraping"/>
<link rel="dns-prefetch" href="//login.wikimedia.org"/>
<link rel="dns-prefetch" href="//meta.wikimedia.org" />
</head>
<body class="mediawiki ltr sitedir-ltr mw-hide-empty-elt ns-0 ns-subject mw-editable page-Web_scraping rootpage-Web_scraping skin-vector action-view skin-vect
<div id="mw-head-base" class="noprint"></div>
<div id="content" class="mw-body" role="main">
    <a id="top"></a>
    <div id="siteNotice" class="mw-body-content"><!-- CentralNotice --></div>
    <div class="mw-indicators mw-body-content">
    </div>
    <h1 id="firstHeading" class="firstHeading" lang="en">Web scraping</h1>
    <div id="bodyContent" class="mw-body-content">
        <div id="siteSub" class="noprint">From Wikipedia, the free encyclopedia</div>
        <div id="contentSub"></div>
        <div id="contentSub2"></div>

        <div id="jump-to-nav"></div>
        <a class="mw-jump-link" href="#mw-head">Jump to navigation</a>
        <a class="mw-jump-link" href="#searchInput">Jump to search</a>
        <div id="mw-content-text" lang="en" dir="ltr" class="mw-content-ltr"><div class="mw-parser-output"><div class="shortdescription nomobile noexcerpt nop
<table class="box-More_citations_needed plainlinks metadata ambox ambox-content ambox-Refimprove" role="presentation"><tbody><tr><td class="mbox-image"><div s
<div role="note" class="hatnote navigation-not-searchable">For broader coverage of this topic, see <a href="/wiki/Data_scraping" title="Data scraping">Data sc
<p><b>Web scraping</b>, <b>web harvesting</b>, or <b>web data extraction</b> is <a href="/wiki/Data_scraping" title="Data scraping">data scraping</a> used for
</p><p>Web scraping a web page involves fetching it and extracting from it. Fetching is the downloading of a page (which a browser does when a user views a pa
</p><p>Web scraping is used for <a href="/wiki/Contact_scraping" title="Contact scraping">contact scraping</a>, and as a component of applications used for <a
</p><p><a href="/wiki/Web_page" title="Web page">Web pages</a> are built using text-based mark-up languages (<a href="/wiki/HTML" title="HTML">HTML</a> and <a
</p><p>Newer forms of web scraping involve listening to data feeds from web servers.  For example, <a href="/wiki/JSON" title="JSON">JSON</a> is commonly usec
</p><p>There are methods that some websites use to prevent web scraping, such as detecting and disallowing bots from crawling (viewing) their pages. In respor
</p>
```

Inspecting Elements





Web scraping

From Wikipedia, the free encyclopedia

This article needs additional citations for verification. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed.
Find sources: "Web scraping" – news · newspapers · books · scholar · JSTOR (June 2017) (Learn how and when to remove this template message)

For broader coverage of this topic, see Data scraping.

Web scraping, **web harvesting**, or **web data extraction** is data scraping used for extracting data from websites. Web scraping software may access the World Wide Web directly using the Hypertext Transfer Protocol, or through a web browser. While web scraping can be done manually by a software user, the term typically refers to automated processes implemented using a bot or web crawler. It is a form of copying, in which specific data is gathered and copied from the web, typically into a central local database or spreadsheet, for later retrieval or analysis.

Web scraping a web page involves fetching it and extracting from it. Fetching is the downloading of a page (which a browser does when a user views a page). Therefore, web crawling is a main component of web scraping, to fetch pages for later processing. Once fetched, then extraction can take place. The content of a page may be parsed, searched, reformatted, its data copied into a spreadsheet, and so on. Web scrapers typically take something out of a page, to make use of it for another purpose somewhere else. An example would be to find and copy names and phone numbers, or companies and their URLs, to a list (contact scraping).

Web scraping is used for contact scraping, and as a component of applications used for web indexing, web mining and data mining, online price change monitoring and

```
<!DOCTYPE html>
<html class="client-js ve-available" dir="ltr" lang="en"> event scroll overflow
  <head> … </head>
  <body class="mediawiki ltr sitedir-ltr mw-hide-empty-elt ns-0 ns-subject …page-Web_scraping
  skin-vector action-view skin-vector-legacy"> event
    <div id="mw-page-base" class="noprint"></div>
    <div id="mw-head-base" class="noprint"></div>
    <div id="content" class="mw-body" role="main">
      <a id="top"></a>
      <div id="siteNotice" class="mw-body-content"> … </div>
      <div class="mw-indicators mw-body-content"> … </div>
      <h1 id="firstHeading" class="firstHeading" lang="en"> … </h1>
      <div id="bodyContent" class="mw-body-content"> … </div>
    </div>
    <div id="mw-data-after-content"> … </div>
    <div id="mw-navigation"> … </div>
```

```
element {                                   inline
}
.mw-body .firstHeading {        load.php:1 @screen
  overflow: visible;
}
.mw-body h1, .mw-body-content h1  load.php:1 @screen
{
  font-size: 1.8em;
}
.mw-body h1, .mw-body-content h1,  load.php:1 @screen
.mw-body-content h2 {
  margin-bottom: 0.25em;
  padding: 0;
  font-family: 'Linux
      Libertine','Georgia','Times',serif;
  line-height: 1.3;
```

3. Inspecting Elements
Another useful tool which Firefox provides (as does Chrome, and I'm sure many other browsers), is the ability to "inspect an element". What this means is that you can select an element on the website and ask to be directed to the actual HTML Code for that specific element. In Firefox, you let your mouse hover over the element you're interested in, then right-click (or Control+Click on a Mac), and select "Inspect Element". This opens a second pane in the browser and highlights the HTML for that specific element. This is extremely useful when trying to figure out characteristics of an element which you might want to use in your scraper.

HTML text to Selector

```python
from scrapy import Selector
```

```python
import requests
url = 'https://en.wikipedia.org/wiki/Web_scraping'
html = requests.get( url ).content
```

```python
sel = Selector( text = html )
```

4. HTML text to Selector

The final piece of this lesson is to give you a quick understanding of how we get the HTML text into a Selector object. The way we do this is using the python requests library. We won't delve into requests more than this short introduction here. In fact, in the last chapter, you will learn how to do everything within scrapy itself, without using requests. But this is still a nice piece of information to carry with you. In this example we will get the HTML from the DataCamp course directory. The requests library makes it easy to get the HTML as a string by first passing to requests.get the url (as a string) of the site you want the HTML from, then looking at the content, as we have done here. You'll notice that we assigned the content to the variable html to pass to the Selector.

You Know Our Secrets

WEB SCRAPING IN PYTHON