

# Web Scraping

# Web Scraping

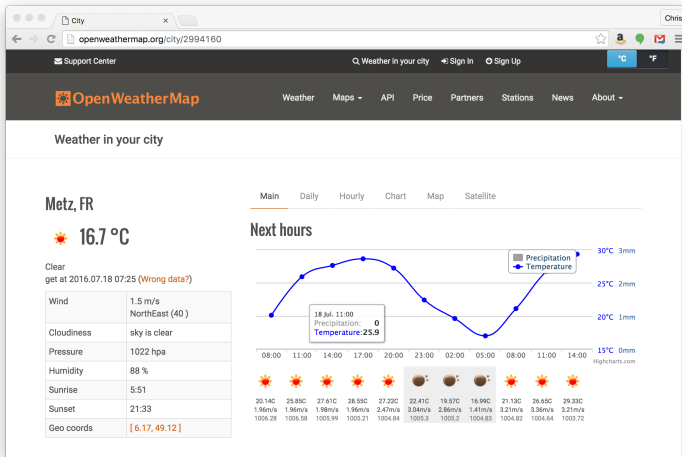
Two ways to mine data from the web

- ▶ The hard way, by web scraping
- ▶ The easy way, using web service APIs

We'll see examples of both.

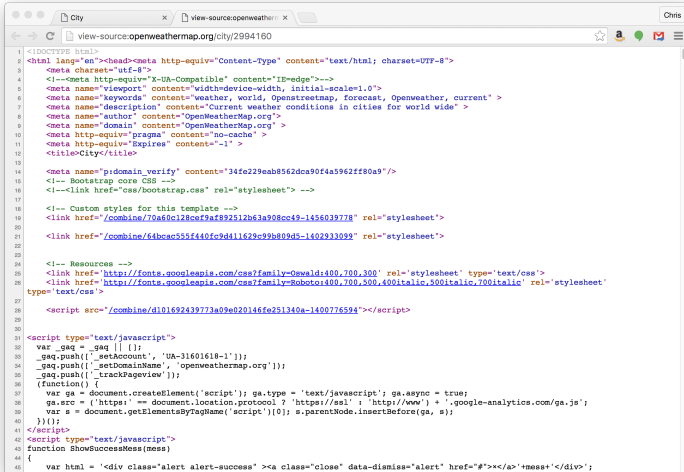
# Web Scraping

Web scraping, a.k.a. screen scraping, means getting data from a web page. Suppose we want to get the current wind data for a city from [Open Weather Map](#).



# What is a Web Page?

A web page is a chunk of text containing HTML code. The browser "renders" the HTML graphically. So web scraping means analyzing text using Python's text processing features.



```
1 <!DOCTYPE html>
2 <html lang="en"><head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
3   <meta charset="utf-8">
4   <!--<meta http-equiv="X-UA-Compatible" content="IE=edge"-->
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <meta name="keywords" content="weather, world, Openstreetmap, forecast, Openweather, current" >
7   <meta name="description" content="Current weather conditions in cities for world wide" >
8   <meta name="author" content="OpenWeatherMap.org">
9   <meta name="domain" content="OpenWeatherMap.org" >
10  <meta http-equiv="pragma" content="no-cache" >
11  <meta http-equiv="Expires" content="-1" >
12  <title>City</title>
13
14  <meta name="p:domain_verify" content="34fe229eab8562dca90f4a5962ff80a9"/>
15  <!-- Bootstrap core CSS -->
16  <!--<link href="css/bootstrap.css" rel="stylesheet" -->
17
18  <!-- Custom styles for this template -->
19  <link href="/combine/70a60c128cef9af892512b63a908cc49-1456039778" rel="stylesheet">
20
21  <link href="/combine/64bracc55f440fc9d411629c99b809d5-1402933099" rel="stylesheet">
22
23
24  <!-- Resources -->
25  <link href='http://fonts.googleapis.com/css?family=Oswald:400,700,300' rel="stylesheet" type="text/css">
26  <link href='http://fonts.googleapis.com/css?family=Roboto:400,700,500,400italic,300italic,700italic' rel="stylesheet"
27  type="text/css">
28
29  <script src="/combine/d101692439773a09e020146fe251340a-1400776594"></script>
30
31  <script type="text/javascript">
32    var _gaq = _gaq || [];
33    _gaq.push(['_setAccount', 'UA-31601618-1']);
34    _gaq.push(['_setDomainName', 'openweathermap.org']);
35    _gaq.push(['_trackPageview']);
36    (function() {
37      var ga = document.createElement('script'); ga.type = 'text/javascript'; ga.async = true;
38      ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www') + '.google-analytics.com/ga.js';
39      var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(ga, s);
40    })();
41  </script>
42  <script type="text/javascript">
43    function ShowSuccessMess(mess)
44    {
45      var html = 'div class="alert alert-success"><a class="close" data-dismiss="alert" href="#"></a>'+mess+'</div>';
```

# Finding The Data On the Page

First you need to find the data within the HTML code for a page so you can construct a regex. Your browser's developer features can help you find the data:

The screenshot shows a web browser displaying the OpenWeatherMap page for Metz, FR. The page shows the current temperature as 16.7°C and a table of weather details. The developer tools are open to the Elements panel, showing the HTML structure of the page. The selected element is a `div` with the class `weather-widget`, which contains the weather details table.

Wind	1.5 m/s NorthEast (40)
Cloudiness	sky is clear
Pressure	1022 hpa
Humidity	88 %
Sunrise	5:51
Sunset	21:33
Geo coords	[6.17, 49.12]

```
HTML: <div class="weather-widget">
  <h3>Metz, FR</h3>
  <div class="clear">
    <p>Clear</p>
  </div>
  <div class="wrong-data-modal">
    <p>Wrong data modal</p>
  </div>
  <table class="table table-striped table-bordered table-condensed">
    <tbody>
      <tr>
        <td>Wind</td>
        <td>1.5 m/s
          NorthEast (40)</td>
      </tr>
      <tr>
        <td>Cloudiness</td>
        <td>sky is clear</td>
      </tr>
      <tr>
        <td>Pressure</td>
        <td>1022 hpa</td>
      </tr>
      <tr>
        <td>Humidity</td>
        <td>88 %</td>
      </tr>
      <tr>
        <td>Sunrise</td>
        <td>5:51</td>
      </tr>
      <tr>
        <td>Sunset</td>
        <td>21:33</td>
      </tr>
      <tr>
        <td>Geo coords</td>
        <td>[6.17, 49.12]</td>
      </tr>
    </tbody>
  </table>
</div>
```

# Getting the Web Page's HTML Code

To get the HTML code of the web page into a Python string variable that you can play with, use Python's `urllib.request` module:

```
import urllib.request
# 2994160 is the city code for Metz, FR
request = urllib.request.Request("http://www.openweathermap.com/city/2994160")
response = urllib.request.urlopen(request)
page_bytes = response.read()
page_text = page_bytes.decode()
# page_text is Python str containing the HTML code
```

or with the `requests` module, which is what we'll use:

```
import requests
resp = requests.get("http://www.openweathermap.com/city/2994160")
resp.text # the text of the web page
```

## Extracting the Data

Looks like the wind data is in the second `<td>` element after the `<div class="weather-widget">` tag, following a `<td>Wind</td>` element. We can play around with the HTML text in the Python REPL. We eventually end up with:

```
wind = re.findall(r'<td>Wind</td><td>(.*?)</td>', page_text.replace("\n",""))[0]
```

Notice that we used a capture group to get the element data.

## Aside: Parsing HTML

HTML is context free language, which roughly means that it supports arbitrary nesting of elements. For example, you could have arbitrarily nested `div` elements with "leaf" elements containing text data, e.g.:

```
<div>
  <div>
    <div>some text</div>
  </div>
</div>
```

By the rules of HTML, you could nest `div` tags as deeply as you want. Regular expressions match regular languages, which don't support arbitrary nesting. So how can we use regexes to "parse" HTML?



# Regex Matching in HTML Code

Parsing means scanning the linear sequence of symbols in a string to determine its structure (usually by putting the symbols in a tree). We don't need to parse HTML to find data on a web page. While the HTML **language** supports arbitrary nesting, a particular web page will be nested to a particular depth, resulting a simple linear sequence of symbols that we can match with a regular expression.