Web Scraping

Web Scraping

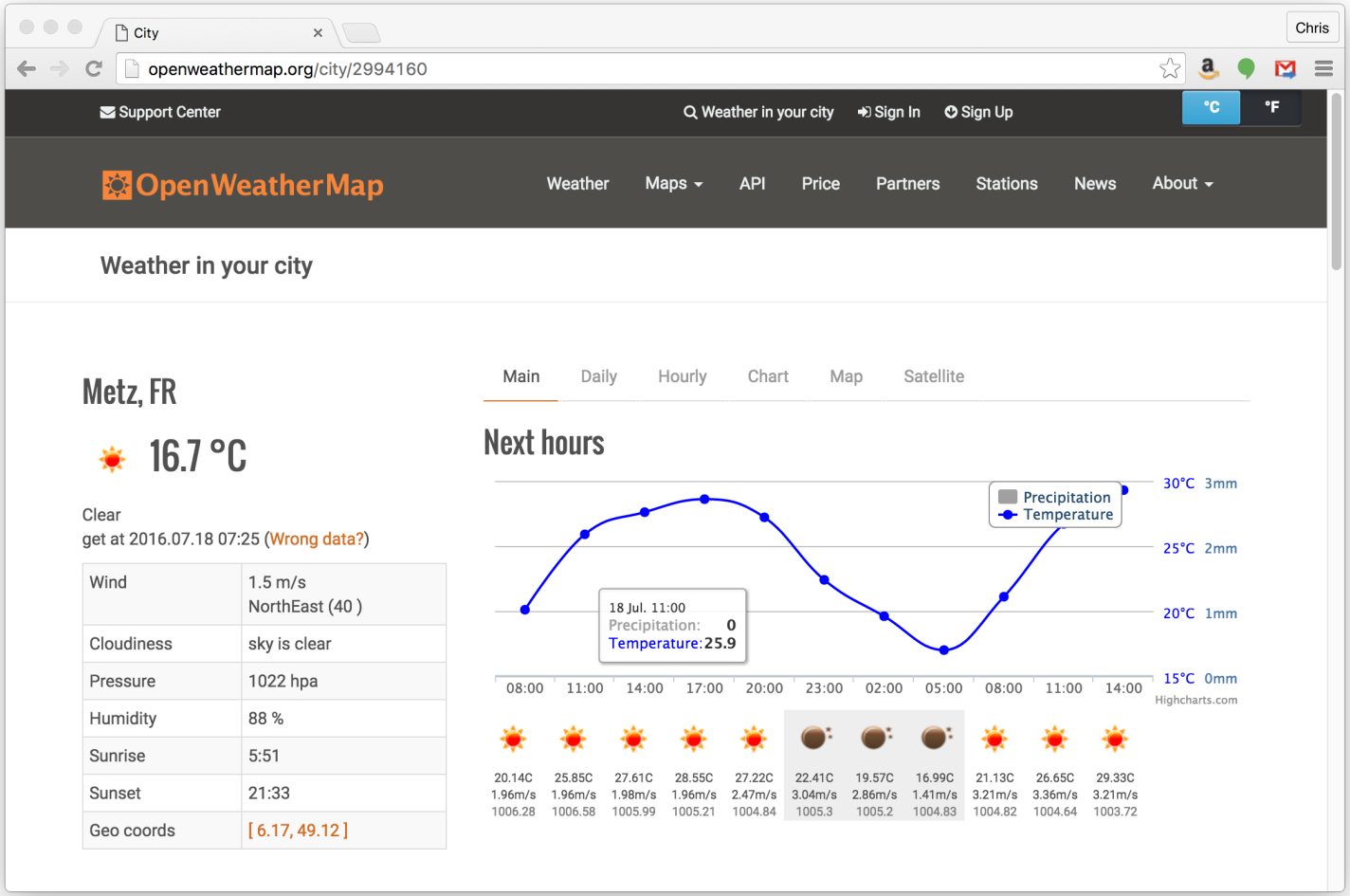
Two ways to mine data from the web

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

We'll see examples of both.

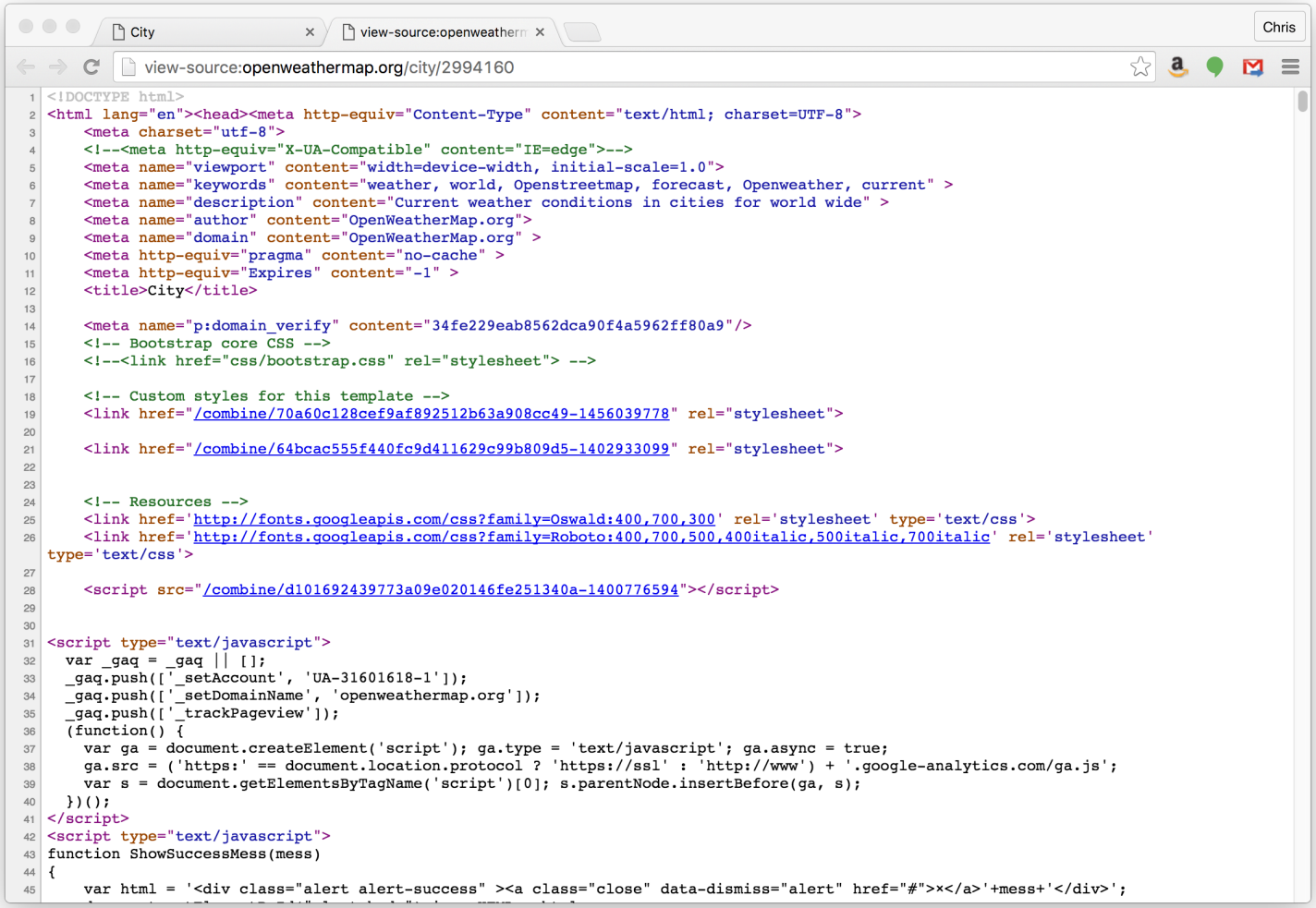
Web Scraping

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?

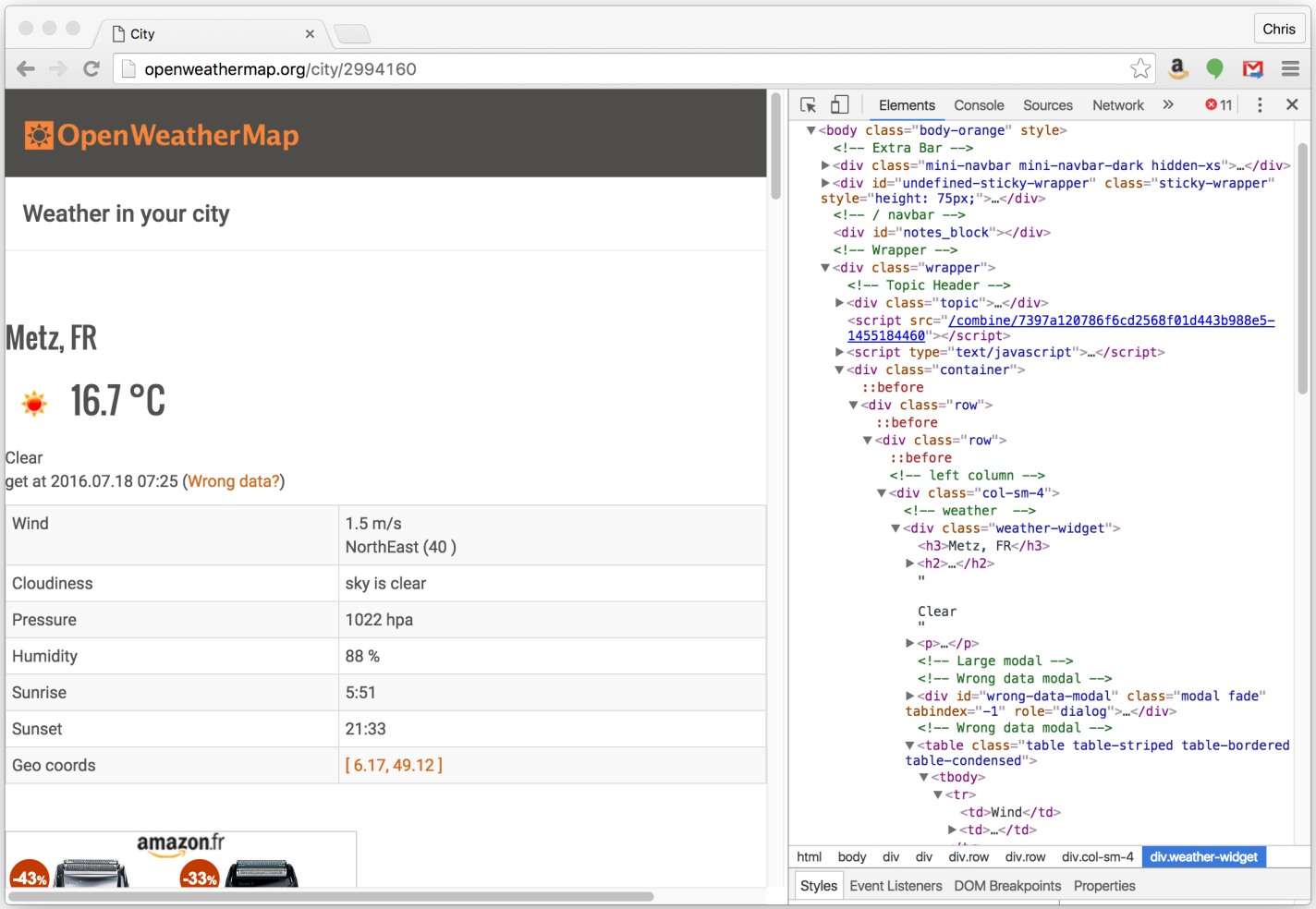
A web page is a chunk of text containing HTML code. The browser "renders" the HTML graphically. So web scraping means analyzing text using Python's text processing features.




```
<!DOCTYPE html>
<html lang="en"><head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <meta charset="utf-8">
    <!--<meta http-equiv="X-UA-Compatible" content="IE=edge">-->
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="keywords" content="weather, world, Openstreetmap, forecast, Openweather, current" >
    <meta name="description" content="Current weather conditions in cities for world wide" >
    <meta name="author" content="OpenWeatherMap.org">
    <meta name="domain" content="OpenWeatherMap.org" >
    <meta http-equiv="pragma" content="no-cache" >
    <meta http-equiv="Expires" content="-1" >
    <title>City</title>

    <meta name="p:domain_verify" content="34fe229eab8562dca90f4a5962ff80a9"/>
    <!-- Bootstrap core CSS -->
    <!--<link href="css/bootstrap.css" rel="stylesheet"> -->

    <!-- Custom styles for this template -->
    <link href="/combine/70a60c128cef9af892512b63a908cc49-1456039778" rel="stylesheet">

    <link href="/combine/64bcac555f440fc9d411629c99b809d5-1402933099" rel="stylesheet">


    <!-- Resources -->
    <link href='http://fonts.googleapis.com/css?family=Oswald:400,700,300' rel='stylesheet' type='text/css'>
    <link href='http://fonts.googleapis.com/css?family=Roboto:400,700,500,400italic,500italic,700italic' rel='stylesheet'
type='text/css'>

    <script src="/combine/d101692439773a09e020146fe251340a-1400776594"></script>


<script type="text/javascript">
  var _gaq = _gaq || [];
  _gaq.push(['_setAccount', 'UA-31601618-1']);
  _gaq.push(['_setDomainName', 'openweathermap.org']);
  _gaq.push(['_trackPageview']);
  (function() {
    var ga = document.createElement('script'); ga.type = 'text/javascript'; ga.async = true;
    ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www') + '.google-analytics.com/ga.js';
    var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(ga, s);
  })();
</script>
<script type="text/javascript">
function ShowSuccessMess(mess)
{
    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:



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
```

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 laguages, 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.