Data 100
Lecture 23:
Web Scraping Technologies
Ex 1. We are interested in Men’s 1500m world records – found in a Wikipedia table

https://en.wikipedia.org/wiki/1500_metres_world_record_progression
1500 metres world record progression

From Wikipedia, the free encyclopedia

The 1500-metre run became a standard racing distance in Europe in the late 19th century, perhaps as a metric version of the mile, a popular running distance since at least the 1850s in English-speaking countries. A distance of 1500 m sometimes is called the "metric mile".

The French had the first important races over the distance, holding their initial championship in 1888. When the Olympic games were revived in 1896, metric distances were run, including the 1500. However, most of the best milers in the world were absent, and the winning time of 4:33 1/5 by Australian Edwin Flack was almost 18 seconds slower than the amateur mile record, despite the fact the mile is 109 metres longer than the 1500 metres.

The 1900 Olympics and 1904 Olympics showed improvements in times run, but it was not until the 1908 Olympics that a meeting of the top milers over the distance took place, and not until the 1912 Olympics that a true world-class race over the distance was run. The distance has now almost completely replaced the mile in major track meets.

Men (outdoors)

<table>
<thead>
<tr>
<th>Pre-IAAF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
</tr>
<tr>
<td>4:24 3/5</td>
</tr>
</tbody>
</table>
We want to scrape the times and dates that appear in this table on the Web page.

<table>
<thead>
<tr>
<th>Time</th>
<th>Auto</th>
<th>Athlete</th>
<th>Date</th>
<th>Place</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:55.8</td>
<td></td>
<td>Abel Kiviat (USA)</td>
<td>1912-06-08</td>
<td>Cambridge, Massachusetts, USA</td>
</tr>
<tr>
<td>3:54.7</td>
<td></td>
<td>John Zander (SWE)</td>
<td>1917-08-05</td>
<td>Stockholm, Sweden</td>
</tr>
<tr>
<td>3:52.6</td>
<td></td>
<td>Paavo Nurmi (FIN)</td>
<td>1924-06-19</td>
<td>Helsinki, Finland</td>
</tr>
<tr>
<td>3:51.0</td>
<td></td>
<td>Otto Peltzer (GER)</td>
<td>1926-09-11</td>
<td>Berlin, Germany</td>
</tr>
<tr>
<td>3:49.2</td>
<td></td>
<td>Jules Ladoumegue (FRA)</td>
<td>1930-10-05</td>
<td>Paris, France</td>
</tr>
<tr>
<td>3:49.2</td>
<td></td>
<td>Luigi Beccali (ITA)</td>
<td>1933-09-09</td>
<td>Turin, Italy</td>
</tr>
<tr>
<td>3:49.0</td>
<td></td>
<td>Luigi Beccali (ITA)</td>
<td>1933-09-17</td>
<td>Milan, Italy</td>
</tr>
<tr>
<td>3:48.8</td>
<td></td>
<td>Bill Bonthron (USA)</td>
<td>1934-06-30</td>
<td>Milwaukee, USA</td>
</tr>
<tr>
<td>3:47.8</td>
<td></td>
<td>Jack Lovelock (NZL)</td>
<td>1936-08-06</td>
<td>Berlin, Germany</td>
</tr>
<tr>
<td>3:47.6</td>
<td></td>
<td>Gunner Hägg (SWE)</td>
<td>1941-08-10</td>
<td>Stockholm, Sweden</td>
</tr>
<tr>
<td>3:45.8</td>
<td></td>
<td>Gunner Hägg (SWE)</td>
<td>1942-07-17</td>
<td>Stockholm, Sweden</td>
</tr>
</tbody>
</table>
Ex 2. We are interested in gas prices - available from web forms on CA Energy Commission’s site

https://ww2.energy.ca.gov/almanac/transportation_data/gasoline/margins/index_cms.php
Estimated 2019 Gasoline Price Breakdown and Margins Details
## Tables of Weekly Gas Prices

<table>
<thead>
<tr>
<th>Oct 28</th>
<th></th>
<th>Oct 21</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribution Costs, Marketing Costs and Profits</td>
<td>$0.690</td>
<td>Distribution Costs, Marketing Costs and Profits</td>
<td>$0.610</td>
</tr>
<tr>
<td>Crude Oil Costs</td>
<td>$1.540</td>
<td>Crude Oil Costs</td>
<td>$1.500</td>
</tr>
<tr>
<td>Refinery Cost and Profit</td>
<td>$0.950</td>
<td>Refinery Cost and Profit</td>
<td>$1.160</td>
</tr>
<tr>
<td>State Underground Storage Tank Fee</td>
<td>$0.020</td>
<td>State Underground Storage Tank Fee</td>
<td>$0.020</td>
</tr>
<tr>
<td>State and Local Tax</td>
<td>$0.087</td>
<td>State and Local Tax</td>
<td>$0.089</td>
</tr>
<tr>
<td>State Excise Tax</td>
<td>$0.473</td>
<td>State Excise Tax</td>
<td>$0.473</td>
</tr>
<tr>
<td>Federal Excise Tax</td>
<td>$0.184</td>
<td>Federal Excise Tax</td>
<td>$0.184</td>
</tr>
<tr>
<td>Retail Prices</td>
<td>$3.950</td>
<td>Retail Prices</td>
<td>$4.030</td>
</tr>
</tbody>
</table>
Want Data for Additional Years

<table>
<thead>
<tr>
<th>Federal Excise Tax</th>
<th>$0.184</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retail Prices</td>
<td>$3.180</td>
</tr>
</tbody>
</table>

Select Year

- 2018
- 2017
- 2016
- 2015
- 2014
- 2013
- 2012
- 2011
- 2010
- 2009
- 2008

Unbranded Gasoline: Branded gasoline refers to fuel that has been attracted to a gasoline station with a brand name. Unbranded gasoline is not associated with any fuel additives. Unbranded gasoline is not associated with any fuel additives. Unbranded gasoline is not associated with any fuel additives. Unbranded gasoline is not associated with any fuel additives.
Ex 3. We want to study global climate models - available from World Bank
World Bank REST API

Instructions for how to retrieve data their data files.
Today

Data Scientists retrieve data from the Web programmatically

- Pandas, BeautifulSoup, and lxml libraries
- Formats: HTML, XML, and JSON
- Trees: XPath and BeautifulSoup
- HTTP – Get and Post, and REST APIs
HTTP – Hypertext Transfer Protocol
HTTP
Hypertext Transfer Protocol

- Created at CERN by Tim Berners-Lee in 1989 as part of the World Wide Web
- Started as a simple request-response protocol used by web servers and browsers to access hypertext
- Widely used exchange data and provides services:
  - Access webpage & submit forms
  - Common API to data and services across the internet
- Foundation of modern REST APIs
Request – Response Protocol

First line contains:

- **GET** /wiki/1500...progression HTTP/1.1
- a method, e.g., GET or POST
- a URL or path to the document
- the protocol and its version

Remaining Header Lines

- Key–value pairs
- Specify a range of attributes

Optional Body

- send extra parameters & data
Request – Response Protocol

- First line contains status code
- Key-Value Pair Lines
- Data properties
- Body
- Returned data
- HTML/JSON/Bytes

**HTTP/1.1 200 OK**
Date: Mon, 11 Nov 2019 22:20:16 GMT
Content-Type: text/html; charset=UTF-8
Server: mw1266.eqiad.wmnet
Content-Encoding: gzip
Content-Length: 19310

<!-- HTML content -->
<table>
<thead>
<tr>
<th>Time</th>
<th>Auto</th>
<th>Athlete</th>
<th>Date</th>
<th>Place</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:55.8</td>
<td></td>
<td>Abel Kiviät (USA)</td>
<td>1912-06-08</td>
<td>Cambridge, Massachusetts, United States</td>
</tr>
<tr>
<td>3:54.7</td>
<td></td>
<td>John Zander (SWE)</td>
<td>1917-08-05</td>
<td>Stockholm, Sweden</td>
</tr>
<tr>
<td>3:52.6</td>
<td></td>
<td>Paavo Nurmi (FIN)</td>
<td>1924-06-19</td>
<td>Helsinki, Finland</td>
</tr>
<tr>
<td>3:51.0</td>
<td></td>
<td>Otto Peltzer (GER)</td>
<td>1926-09-11</td>
<td>Berlin, Germany</td>
</tr>
<tr>
<td>3:49.2</td>
<td></td>
<td>Jules Ladoumégue (FRA)</td>
<td>1930-10-05</td>
<td>Paris, France</td>
</tr>
<tr>
<td>3:49.2</td>
<td></td>
<td>Luigi Beccali (ITA)</td>
<td>1933-09-09</td>
<td>Turin, Italy</td>
</tr>
<tr>
<td>3:49.0</td>
<td></td>
<td>Luigi Beccali (ITA)</td>
<td>1933-09-17</td>
<td>Milan, Italy</td>
</tr>
<tr>
<td>3:48.8</td>
<td></td>
<td>Bill Bonthon (USA)</td>
<td>1934-06-30</td>
<td>Milwaukee, United States</td>
</tr>
<tr>
<td>3:47.8</td>
<td></td>
<td>Jack Lovelock (NZL)</td>
<td>1936-08-06</td>
<td>Berlin, Germany</td>
</tr>
</tbody>
</table>
Request Types (Main Types)

- **GET** – *get information*
  - Parameters passed in URI (limited to ~2000 characters)
    - `/app/user_info.json?username=mejoeyg&version=now`
    - Request body is typically ignored
  - Should not have side-effects (e.g., update user info)
  - Can be cached in on server, network, or in browser (bookmarks)

- **POST** – *send information*
  - Parameters passed in URI and BODY
  - May and typically will have side-effects
  - Often used with web forms.
  - Related requests: PUT, DELETE
Response Status Codes

- **100s Informational** – Communication continuing, more input expected from client or server
- **200 Success** - e.g., 200 - general success;
- **300s Redirection or Conditional Action** – requested URL is located somewhere else.
- **400s Client Error**
  - 404 indicates the document was not found
  - 403 indicates that the server understood the request but refuses to authorize it
- **500s Internal Server Error or Broken Request** – error on the server side
Managing Requests: requests Library

```python
res = requests.get(url)  # GET Method

Access the request status with `res.status_code`

Access the request method with `res.request.method`
Access the request header with `res.request.headers`

Access the response header with `res.headers`
Access the response body (content) with `res.content`
```
Getting data from tables on the Web
Starting Simple with Pandas
Pandas `read_html`

- Loads tables from web pages
  - Looks for `<table></table>` tags
  - Table needs to be well formatted
  - Returns a list of DataFrames
- Can load directly from URL
  - Careful! Data changes. Save a copy on the Web page contents with your analysis
- You will often need to do additional transformations to prepare the data
HTML – HyperText Markup Language
Simple HTML Document

```html
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Example</title>
  </head>
  <body>
    <h2>Simple HTML page</h2>
    <p>A <i>paragraph</i> about the table below.</p>
    <table id="mydata" border="1" cellpadding="4">
      <tr><th>X</th><th>Y</th></tr>
      <tr><td>$1.25</td><td>17</td></tr>
      <tr><td>$2.50</td><td>25</td></tr>
      <tr><td>$2.00</td><td>22</td></tr>
    </table>
  </body>
</html>
```
Many Tables on the 1500m page

This is the table we want.

<table>
<thead>
<tr>
<th>Time</th>
<th>Auto</th>
<th>Athlete</th>
<th>Date</th>
<th>Place</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:55.8</td>
<td>🇺🇸</td>
<td>Abel Kiviat (USA)</td>
<td>1912-06-08</td>
<td>Cambridge, Massachusetts, USA</td>
</tr>
<tr>
<td>3:54.7</td>
<td>🇸🇪</td>
<td>John Zander (SWE)</td>
<td>1917-08-05</td>
<td>Stockholm, Sweden</td>
</tr>
<tr>
<td>3:52.6</td>
<td>🇫🇮</td>
<td>Paavo Nurmi (FIN)</td>
<td>1924-06-19</td>
<td>Helsinki, Finland</td>
</tr>
<tr>
<td>3:51.0</td>
<td>🇩🇪</td>
<td>Otto Peltzer (GER)</td>
<td>1926-09-11</td>
<td>Berlin, Germany</td>
</tr>
<tr>
<td>3:49.2</td>
<td>🇫🇷</td>
<td>Jules Ladoumegue (FRA)</td>
<td>1930-10-05</td>
<td>Paris, France</td>
</tr>
<tr>
<td>3:49.2</td>
<td>🇮🇹</td>
<td>Luigi Beccali (ITA)</td>
<td>1933-09-09</td>
<td>Turin, Italy</td>
</tr>
<tr>
<td>3:49.0</td>
<td>🇮🇹</td>
<td>Luigi Beccali (ITA)</td>
<td>1933-09-17</td>
<td>Milan, Italy</td>
</tr>
<tr>
<td>3:48.8</td>
<td>🇺🇸</td>
<td>Bill Bonthron (USA)</td>
<td>1934-06-30</td>
<td>Milwaukee, USA</td>
</tr>
<tr>
<td>3:47.8</td>
<td>🇳🇿</td>
<td>Jack Lovelock (NZL)</td>
<td>1936-08-06</td>
<td>Berlin, Germany</td>
</tr>
<tr>
<td>3:47.6</td>
<td>🇸🇪</td>
<td>Gunder Hägg (SWE)</td>
<td>1941-08-10</td>
<td>Stockholm, Sweden</td>
</tr>
<tr>
<td>3:45.8</td>
<td>🇸🇪</td>
<td>Gunder Hägg (SWE)</td>
<td>1942-07-17</td>
<td>Stockholm, Sweden</td>
</tr>
</tbody>
</table>
Use Browser to Examine page source

Here’s the HTML for the table we want.

Notice the name Zander
Pandas extracts tables from HTML documents as a list of data frames

```python
tables = pd.read_html(url)
len(tables)  # 6
```

<table>
<thead>
<tr>
<th>Time</th>
<th>Auto</th>
<th>Athlete</th>
<th>Date</th>
<th>Place</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:55.8</td>
<td>NaN</td>
<td>Abel Kiviat (USA)</td>
<td>1912-06-08</td>
<td>Cambridge, Massachusetts, United States</td>
</tr>
<tr>
<td>3:54.7</td>
<td>NaN</td>
<td>John Zander (SWE)</td>
<td>1917-08-05</td>
<td>Stockholm, Sweden</td>
</tr>
<tr>
<td>3:52.6</td>
<td>NaN</td>
<td>Paavo Nurmi (FIN)</td>
<td>1924-06-19</td>
<td>Helsinki, Finland</td>
</tr>
<tr>
<td>3:51.0</td>
<td>NaN</td>
<td>Otto Peltzer (GER)</td>
<td>1926-09-11</td>
<td>Berlin, Germany</td>
</tr>
<tr>
<td>3:49.2</td>
<td>NaN</td>
<td>Jules Ladoumégue (FRA)</td>
<td>1930-10-05</td>
<td>Paris, France</td>
</tr>
</tbody>
</table>
Clean and Transform Data

- Need times in seconds
- Some times have + signs, e.g., 3:42.8+
- Dates need to be converted into date format
XML

eXtensible Markup Language
Most services will exchange data in XML and/or JSON

Why?

- Descriptive
  - Can maintain meta-data
- Extensible
  - Organization can change and maintain compatibility
- Human readable
  - Useful for debugging and provides a common interface
- Machine readable
  - A wide range of technologies for parsing
XML is a standard for semantic, hierarchical representation of data.
Syntax

The basic unit of XML code is called an “element” or “node”

Each Node has a start tag and end tag

```
<zone>4</zone>
```

Start tag

Content

End tag
Syntax: Nesting

A node may contain other nodes (children) in addition to plain text content.

```xml
<plant type='a'>
  <zone>4</zone>
  <light>Mostly Shady</light>
</plant>
```

Start tag

Content consists of two nodes

End tag

Indentation is not needed. It simply shows the nesting.
Syntax: Empty Nodes

<plant>
  <zone></zone>
  <light/>
</plant>

These two nodes are empty. Both formats are acceptable.
Syntax: Attributes

Nodes may have attributes (and attribute values)

```xml
<plant type='a'>
  <zone/>
  <light source="2" class="new"/>
</plant>
```

The attribute named type has a value of “a”

This empty node has two attributes: source and class
Syntax: Comments

Comments can appear anywhere

<plant>

<!-- elem with content -->

<zone>4 <!-- a second comment --></zone>

<light>Mostly Shady</light>

</plant>
Well-formed XML

- An element must have both an **open** and **closing** tag. However, if it is empty, then it can be of the form `<tagname/>`.
- Tags must **nest properly**.
  - Bad!: `<plant><kind></plant></kind>`
- Tag names are case-sensitive; start and end tags must match exactly.
- No spaces are allowed between `<` and `tag name`.
- Tag names must begin with a letter and contain only alphanumeric characters.
Well-formed XML:

- All **attributes** must appear in quotes:
  
  ```
  name = "value"
  ```

- Isolated markup characters must be specified via entity references. `<` is specified by `&lt;` and `>` is specified by `&gt;`.

- All XML documents must have one **root node** that contains all the other nodes.
xHTML: Extensible Hypertext Markup Language

- HTML is an XML-"like" structure → Pre-dated XML
  - HTML is often not well-formed, which makes it difficult to parse and locate content,
  - Special parsers “fix” the HTML to make it well-formed
    - Results in even worse HTML

- xHTML was introduced to bridge HTML and XML
  - Adopted by many webpages
  - Can be easily parsed and queried by XML tools
DOM – Document Object Model
A tree representation
DOM: Document Object Model

- Treat XML & HTML as a Tree
- Fits XML and well-formed HTML
- Visual containment \(\rightarrow\) children
- Manipulated dynamically using JavaScript
- Parsing in Python \(\rightarrow\) Selenium + Headless Chrome ... (out of scope)
Tree terminology

- There is only one root (AKA document node) in the tree, and all other nodes are contained within it.
- We think of these other nodes as descendants of the root node.
- We use the language of a family tree to refer to relationships between nodes.
  - parents, children, siblings, ancestors, descendants
- The terminal nodes in a tree are also known as leaf nodes. Text content always falls in a leaf node.
<catalog>
  <plant>
    <common>Bloodroot</common>
    <botanical>Sanguinaria canadensis</botanical>
    <zone>4</zone>
    <light>Mostly Shady</light>
    <price currency="USD">$2.44</price>
    <availability>031599</availability>
  </plant>
  <plant>
    <common>Columbine</common>
    <botanical>Aquilegia canadensis</botanical>
    <zone>3</zone>
    <light>Mostly Shady</light>
    <price currency="USD">$9.37</price>
    <availability>030699</availability>
  </plant>
  <plant>
    <common>Marsh Marigold</common>
    <botanical>Caltha palustris</botanical>
    <zone>4</zone>
    <light>Mostly Sunny</light>
    <price currency="CAD">$6.81</price>
    <availability>051799</availability>
  </plant>
</catalog>
Four Tasks

1. Retrieve common names of all plants
2. Retrieve plants that grow in zone 4
3. Retrieve common names of plants that grow in zone 4
4. Retrieve prices of plants whose prices are listed in USD
Beautiful Soup
Locate nodes and content in a well-formed XML document
Bloodroot
Columbine
Marsh Marigold
Bloodroot
Sangu... 4 Mostly... $2.44 031599

Cardinal
Label... 2 Shade $3.02 02299
zone4_names = []
for z in zone4:
    zone4_names.append(z.parent.common.string)

zone4_names
['Bloodroot', 'Marsh Marigold']
us_price_nodes = soup.find_all('price', currency="USD")

prices = []
for p in us_price_nodes:
    prices.append(p.string)

prices

['$2.44', '$9.37']
XPath
Locate nodes and content in a well-formed XML document
What is XPath?

- Extraction tool designed for locating content in an XML/HTML file
- Uses the DOM hierarchy of a well-formed XML document to specify the desired chunks to extract
- An XPath expression is a location path that is made up of location steps separated by forward slash /
- Syntax is similar to but more powerful than the way files are located in a hierarchy of directories in a computer file system
Four Tasks

1. Retrieve common names of all plants
2. Retrieve plants that grow in zone 4
3. Retrieve common names of plants that grow in zone 4
4. Retrieve prices of plants whose prices are listed in USD
Retrieve common names of all plants.
//plant[zone/text() = '4']

Retrieve plants that grow in zone 4
availability

catalog

common

Bloodroot

botanica

Sangui…

zone

4

Mostly...

$2.44

031599

plant

3 ways to locate zone nodes

1. 

2. 

3. 

catalog

plant

common

botanical

zone

light

price

availability

Cardinal

Label...

2

Shade

$3.02

02299

//catalog/plant/zone/

//zone

//plant/zone
What’s the difference between these 3 XPath expressions?

//catalog/plant/zone – Any zone node that is a child of plant and grandchild of catalog

//zone – Any zone node anywhere in document are located

//plant/zone – Any zone node that is a child of a plant node anywhere in document

For this document these XPath expressions are equivalent
Retrieve common names of plants that grow in zone 4
Retrieve common names of plants that grow in zone 4

//plant[zone/text() = '4']/common/text()
Retrieve prices of plants whose prices are listed in USD

//price[@currency='USD']
XPath syntax

- Each step has three parts:
  - Axis (direction)
  - Nodetest, and
  - Predicate (optional)
XPath syntax – The axis

The axis is the direction to look (from the current location):

- up the tree one level to the parent,
- up the tree to all ancestors,
- across to older siblings (to the left),
- across to younger siblings (to the right),
- down the tree to child nodes,
- down the tree to any descendant
Simple XPath axes have shortcuts

- “child”, which is the default and can be dropped,
- “descendant-or-self”, which looks anywhere down the tree from current node(s) is abbreviated by “//”
- “self” is abbreviated with a .
- “parent” is abbreviated to ..
Axis shortcuts

Child axis  /catalog/plant/common

Descendant or self  //common

Parent of common  //common/..
XPath syntax – The nodetest

- The *nodetest* is typically a node name that you wish to locate

- For our purposes, the *nodetest* will always be a node name or `text()` for the text content or `@attributename` for that value of an attribute
XPath expressions – The predicate

- The predicate filters the qualifying nodes, i.e., takes a subset of them.

- The predicate is optional and for our purposes will either be
  - a number, which asks for a specific element, e.g. [2] for the second node
  - an attribute filter, e.g.,

```
//plant[zone = "4" or light = "Shade"]
```
Ex 1. Wikipedia Tables
We use the *requests* library to access the web page.

http://docs.python-requests.org/en/master/

```python
In [16]:
wiki1500mURL = 'https://en.wikipedia.org/wiki/1500_metres_world_record_progression'

In [17]:
page1500m_page_response = requests.get(wiki1500mURL)
type(page1500m_page_response)

Out[17]: requests.models.Response

In [18]:
tree1500m = html.fromstring(page1500m_page_response.content)
type(tree1500m)

Out[18]: lxml.html.HtmlElement
```

We “get” the page

Create an HTML “tree”

We use the *lxml* library to create a “tree” consisting of page contents.

http://lxml.de/tutorial.html/
Where in the page are the data?
Extract the run times

```python
times_only = tree1500m.xpath('//table[2]/tr/td[1]')
print("length of times", len(times_only))
print(times_only[34].text_content())
```

length of times 38
3:29.46

Extract the dates

```python
date_column = tree1500m.xpath('//table[2]/tr/td[4]')
print("length of dates", len(date_column))
print(date_column[2].text_content())
```

length of dates 38
1924-06-19
Extract the names

```python
names_attr = tree1500m.xpath('//table[2]/tr/td[3]/a/@title')
print("length of names", len(names_attr))
print(names_attr)
```

length of names 38

HTTP & XPath

- We used HTTP to access the Wikipedia page
- We used XPath to extract the text content of interest from the page
- We can also use Beautiful Soup (see notebook)
- Pandas can extract the table too (see notebook).
- When the data are not in a table then knowing XPath (and Beautiful Soup) can be valuable.
Ex. 2: Acquiring Data from Web forms
View Source

<select> widget

POST method

<input> widget
POST Method

- Requests the server to accept the entity enclosed in the body of the request
- For example, the information in a web form to a data handling process
```python
res_gas = requests.post(posturl_gas, data = dict(year = "2013"))
```

<table>
<thead>
<tr>
<th>res_gas.status_code</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>res_gas.request.method</th>
</tr>
</thead>
<tbody>
<tr>
<td>'POST'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>res_gas.request.headers</th>
</tr>
</thead>
<tbody>
<tr>
<td>{'User-Agent': 'python-requests/2.12.4', 'Accept-Encoding': 'gzip, deflate', 'Accept': '<em>/</em>', 'Connection': 'keep-alive', 'Content-Length': '9', 'Content-Type': 'application/x-www-form-urlencoded'}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>res_gas.request.body</th>
</tr>
</thead>
<tbody>
<tr>
<td>'year=2013'</td>
</tr>
</tbody>
</table>

Notice the POST method

The body of the POST request contains the form information
Ex 3. A REST request for climate simulation data

REST - Representational State Transfer

Climate Data API

About the Climate Data API

The Climate Data API provides programmatic access to most of the climate data used on the World Bank’s Climate Change Knowledge Portal. Web developers can use this API to access the knowledge portal’s data in real time to support their own applications, so long as they abide by the World Bank’s Terms of Use.

About the Data

Except as noted, all the data in the Climate Data API are derived from 15 global circulation models (GCMs), the most comprehensive physically-based models of climate change available and used by the Intergovernmental Panel on Climate Change (IPCC) 4th Assessment Reports. The models simulate the response of the global climate system to increasing greenhouse gas concentrations. The data in the Climate Data API have been aggregated to both the country and basin levels, as explained below.

Note these data are modeled estimates of temperature and precipitation changes in different time periods under different GCMs and scenarios. They include changes for future time periods and also as “backcasting” (model representations of the past) set for past time periods. The latter should not be confused with any instrumental or observed data.

The Basic Request

The Climate Data API uses REST-based requests, in which the general form looks like this:


All boldface variables except for ext in this instance are required. Note that all climate data API requests begin with http://climatedataapi.worldbank.org/climatetext/rest/, so for the sake of legibility, this portion of the URL will be assumed and omitted in the remaining code samples in this documentation.
The Basic Request

The Climate Data API uses REST-based requests, in which the general form looks like this:

http://climatedataapi.worldbank.org/climatetweb/rest/v1/country/type/var/start/end/ISO3[.ext]

All boldface variables except for ext in this instance are required. Note that all climate data API requests begin with http://climatedataapi.worldbank.org/climatetweb/rest/, so for the sake of legibility, this portion of the URL will be assumed and omitted in the remaining code samples in this documentation.

<table>
<thead>
<tr>
<th>type is one of:</th>
<th>Future</th>
</tr>
</thead>
<tbody>
<tr>
<td>mavg</td>
<td>Monthly average</td>
</tr>
<tr>
<td>annualavg</td>
<td>Annual average</td>
</tr>
<tr>
<td>manom</td>
<td>Average monthly change in precipitation variable</td>
</tr>
<tr>
<td>annualanom</td>
<td>Average annual change in precipitation variables, and 1961-2000 for derived statistics</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>var is one of:</th>
<th>start</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr Precipitation (rainfall and assumed water equivalent), in millimeters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tas Temperature, in degrees Celsius</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Future</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
</tr>
<tr>
<td>2020</td>
</tr>
<tr>
<td>2040</td>
</tr>
<tr>
<td>2060</td>
</tr>
<tr>
<td>2080</td>
</tr>
</tbody>
</table>
World Bank Climate Data REST requests

- From documentation, we need to create requests with URLs like:
  
  \[
  \text{wbc\_url} = "http://climatedataapi.worldbank.org/climateweb/rest/v1/country/mavg/bccr BCM2_0/pr/2020/2039/CAN"
  \]

```python
res_wbc = requests.get(wbc_url)
res_wbc.status_code
```

200

Our request was successful

The header tells us that the body of the request is JSON formatted

```json
```
JSON: JavaScript Object Notation

- Recursive datatype
- Data inside of data

- **Value** is a:
  - A **basic type**:
    - String
    - Number
    - true/false
    - Null
  - **Array of Values**
  - A **dictionary of key:Value** pairs
Scraping Etiquette
Before you scrape:

- Check to see if CSV, JSON, or XML version of an HTML page are available – better to use those
- Check to see if there is a Python library that provides structured access (e.g., tweetPy)
- Check that you have permission to scrape
If you do scrape:

- Be careful to not overburden the site with your requests
- Test code on small requests
- Save the results of each request so you don’t have to repeat the request unnecessarily