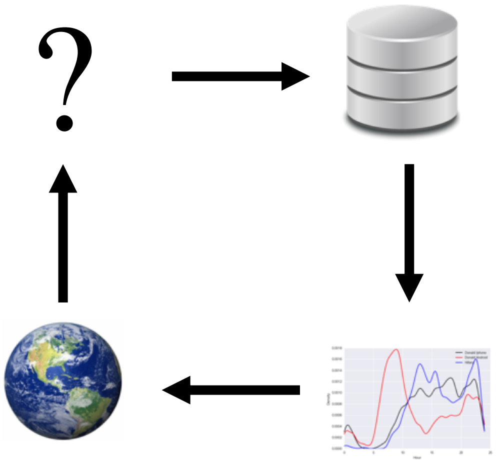
Data 100

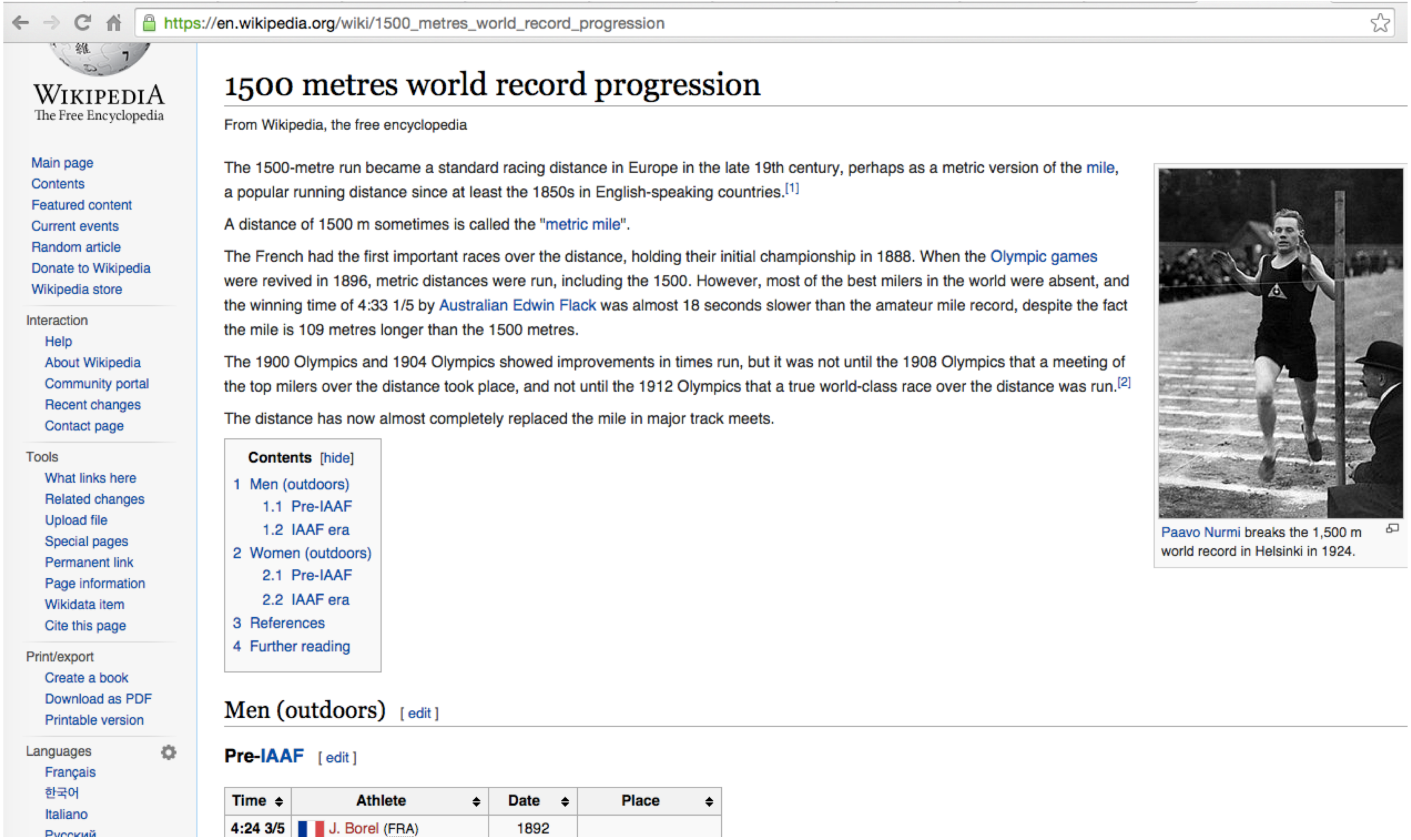
Lecture 23:
Web Scraping Technologies



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

Wikipedia Page



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.[1]

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.[2]

The distance has now almost completely replaced the mile in major track meets.

Paavo Nurmi breaks the 1,500 m world record in Helsinki in 1924.

Contents [hide]

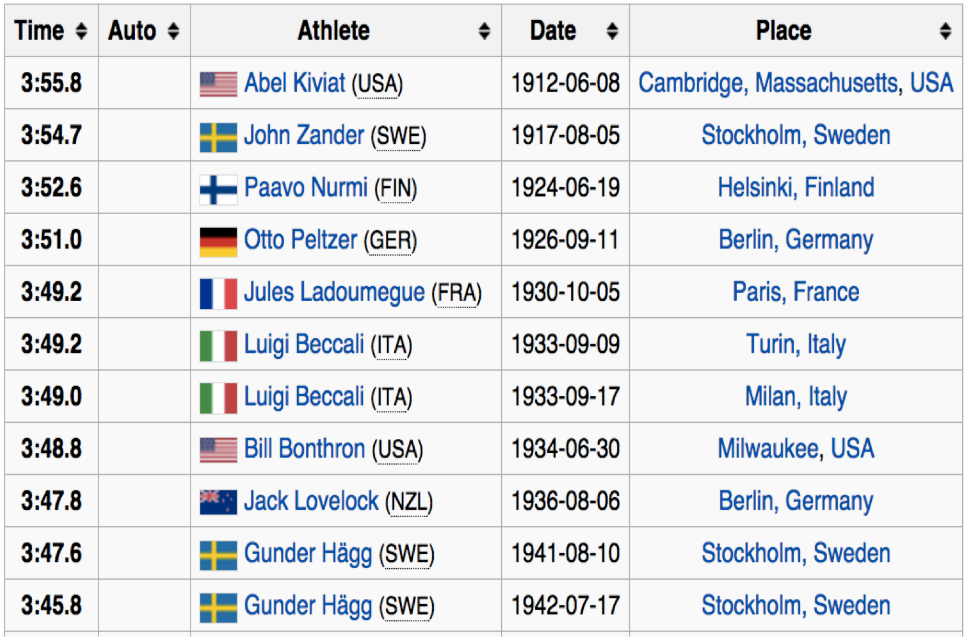
Men (outdoors) [edit]

Pre-IAAF [edit]

Time	Athlete	Date	Place
4:24 3/5	J. Borel (FRA)	1892	

Table of run times and dates

- We want to scrape the times and dates that appear in this table on the Web page



Time	Auto	Athlete	Date	Place
3:55.8		Abel Kiviat (USA)	1912-06-08	Cambridge, Massachusetts, USA
3:54.7		John Zander (SWE)	1917-08-05	Stockholm, Sweden
3:52.6		Paavo Nurmi (FIN)	1924-06-19	Helsinki, Finland
3:51.0		Otto Peltzer (GER)	1926-09-11	Berlin, Germany
3:49.2		Jules Ladoumegue (FRA)	1930-10-05	Paris, France
3:49.2		Luigi Beccali (ITA)	1933-09-09	Turin, Italy
3:49.0		Luigi Beccali (ITA)	1933-09-17	Milan, Italy
3:48.8		Bill Bonthron (USA)	1934-06-30	Milwaukee, USA
3:47.8		Jack Lovelock (NZL)	1936-08-06	Berlin, Germany
3:47.6		Gunder Hägg (SWE)	1941-08-10	Stockholm, Sweden
3:45.8		Gunder Hägg (SWE)	1942-07-17	Stockholm, Sweden

Ex 2. We are interested in gas prices - available from web forms on CA Energy Commission’s site

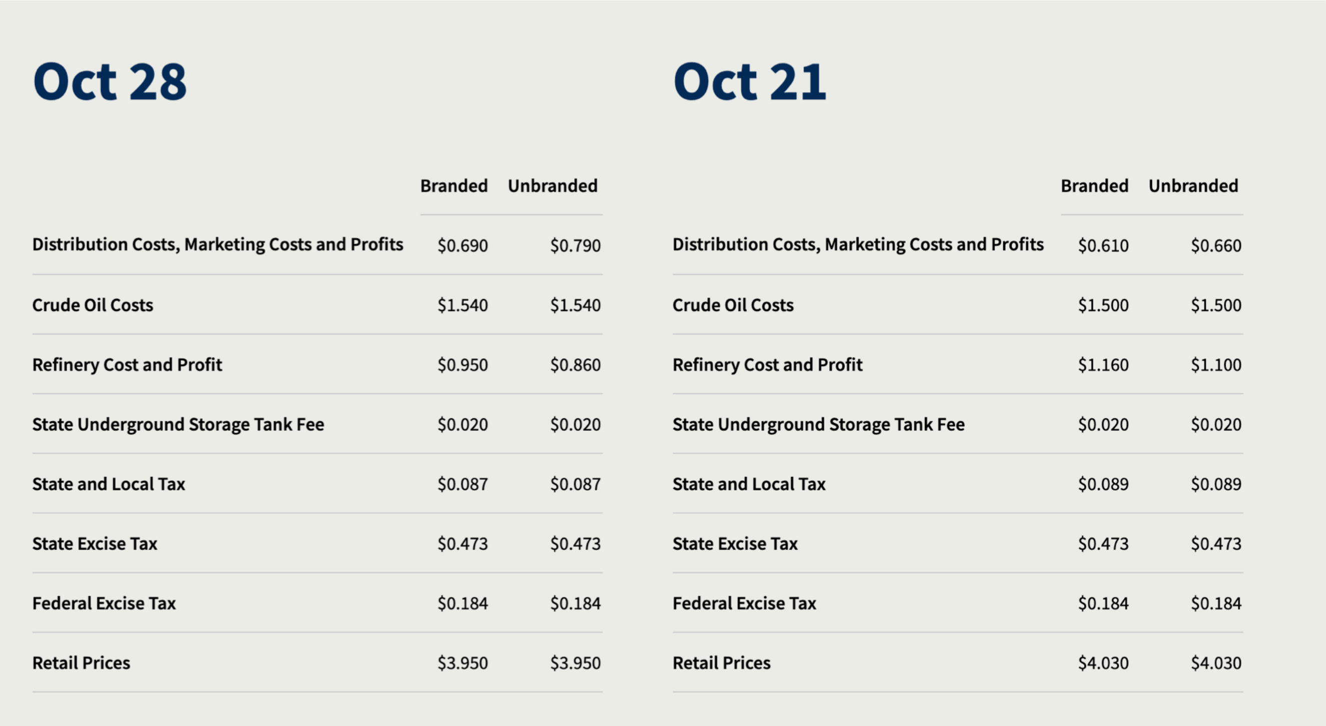
https://ww2.energy.ca.gov/almanac/transportation_data/gasoline/margins/index_cms.php

CA Energy Commission

CA.GOV
Share:
About
Careers
Contact
Ev
CALIFORNIA ENERGY COMMISSION
HOME
PROCEEDINGS
RULES AND REGULATIONS
PROGRAMS AND TOPICS
FUNDING
DATA
Home > Data and Reports > Energy Almanac > Transportation Energy > Estimated Gasoline Price Breakdown and Margins Details

Estimated 2019 Gasoline Price Breakdown and Margins Details

Tables of Weekly Gas Prices



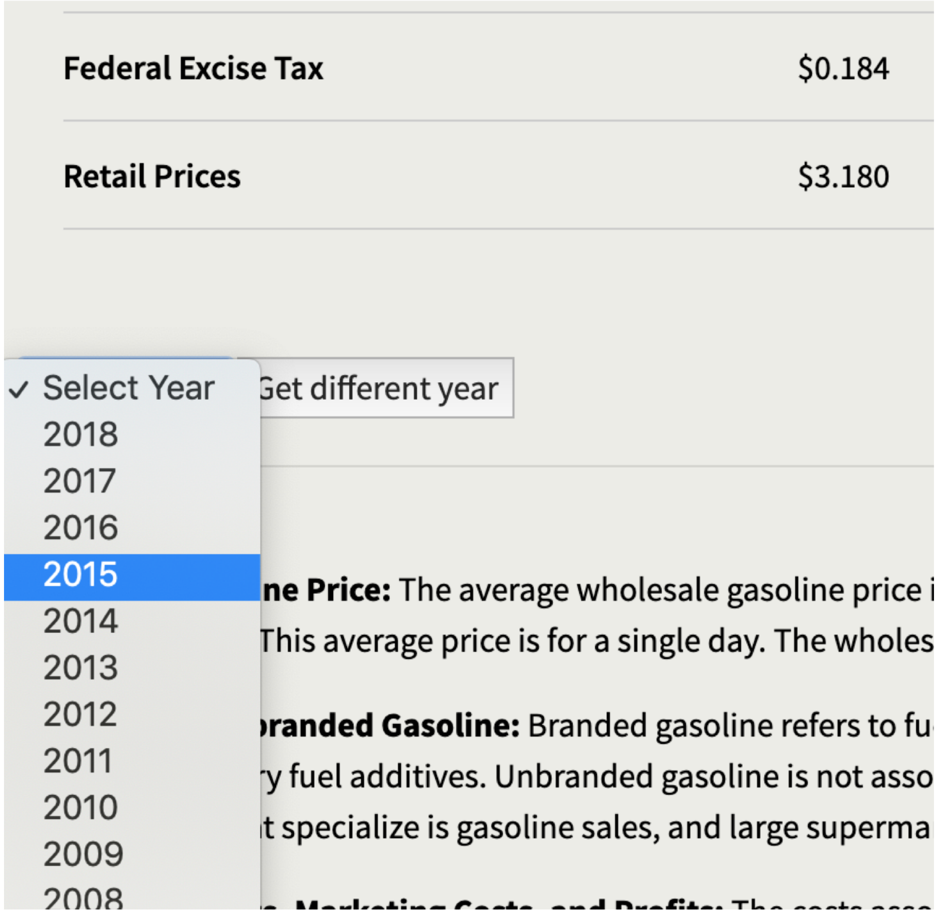
Oct 28

	Branded	Unbranded
Distribution Costs, Marketing Costs and Profits	$0.690	$0.790
Crude Oil Costs	$1.540	$1.540
Refinery Cost and Profit	$0.950	$0.860
State Underground Storage Tank Fee	$0.020	$0.020
State and Local Tax	$0.087	$0.087
State Excise Tax	$0.473	$0.473
Federal Excise Tax	$0.184	$0.184
Retail Prices	$3.950	$3.950

Oct 21

	Branded	Unbranded
Distribution Costs, Marketing Costs and Profits	$0.610	$0.660
Crude Oil Costs	$1.500	$1.500
Refinery Cost and Profit	$1.160	$1.100
State Underground Storage Tank Fee	$0.020	$0.020
State and Local Tax	$0.089	$0.089
State Excise Tax	$0.473	$0.473
Federal Excise Tax	$0.184	$0.184
Retail Prices	$4.030	$4.030

Want Data for Additional Years



Federal Excise Tax	$0.184
Retail Prices	$3.180

ne Price: The average wholesale gasoline price i
This average price is for a single day. The wholes

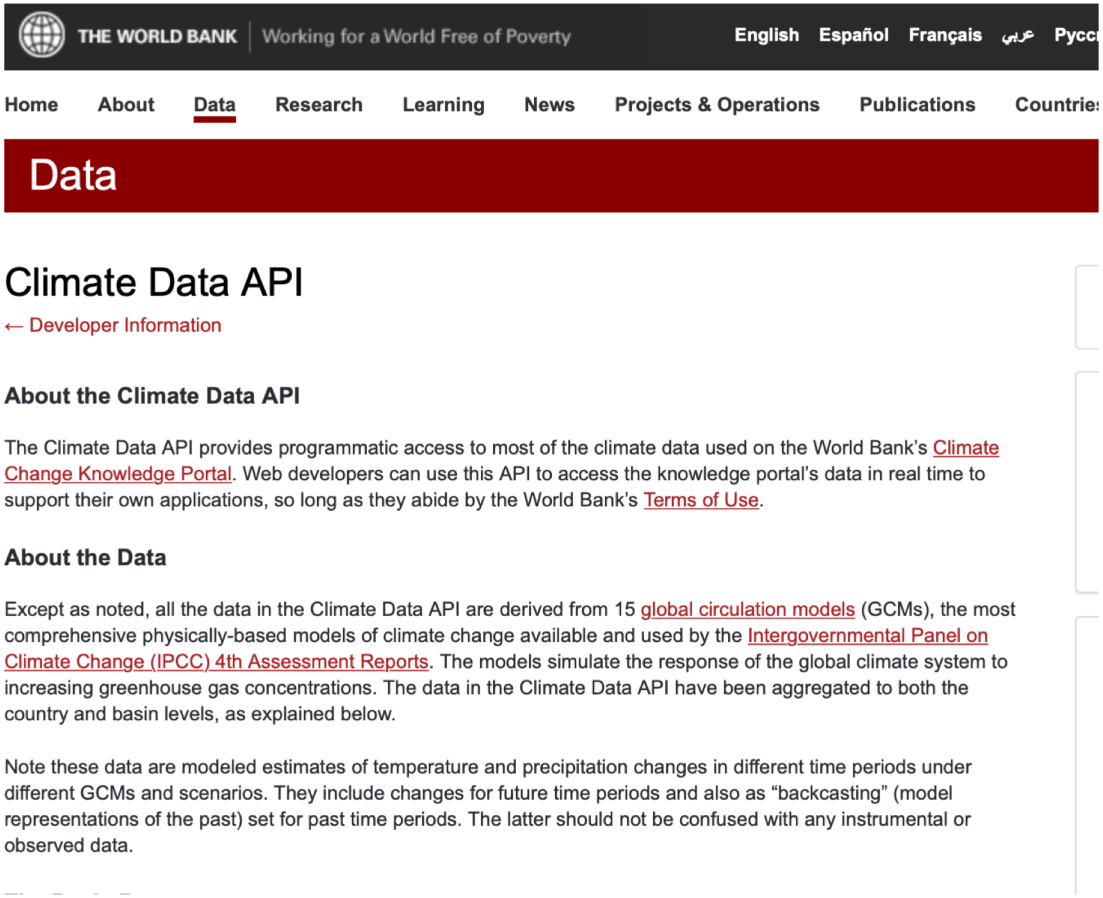
randed Gasoline: Branded gasoline refers to fu
y fuel additives. Unbranded gasoline is not asso
t specialize is gasoline sales, and large superma

, Marketing Costs, and Profits: The costs asso

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



Climate Data API

← Developer Information

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.

Today

Data Scientists retrieve data from the Web programmatically

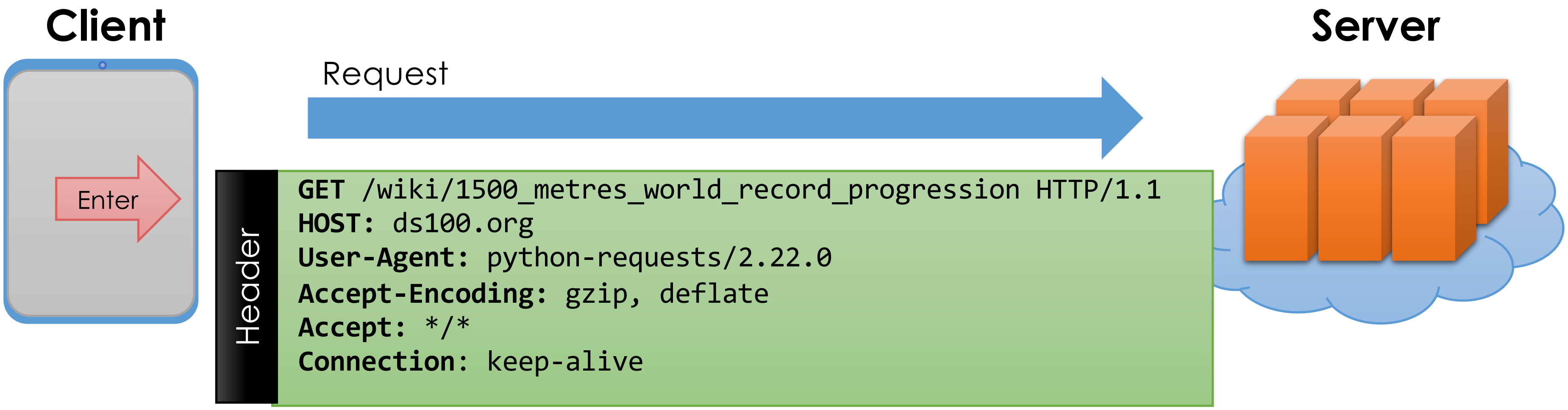
- Pandas, BeautifulSoup, and lxml libraries
- Formats: HTML, XML, and JSON
- Trees: XPath and BeautifulSoup
- HTTP – Get and Post, and REST APIs

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



```
GET /wiki/1500_metres_world_record_progression HTTP/1.1
HOST: ds100.org
User-Agent: python-requests/2.22.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
```

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

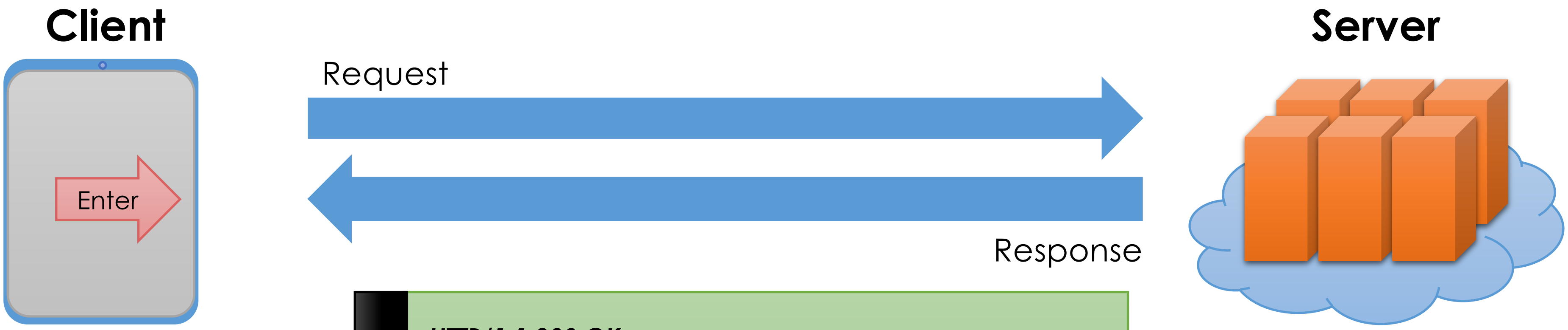
Remaining Header Lines

- Key–value pairs
- Specify a range of attributes

Optional Body

- send extra parameters & data

Request – Response Protocol



Header

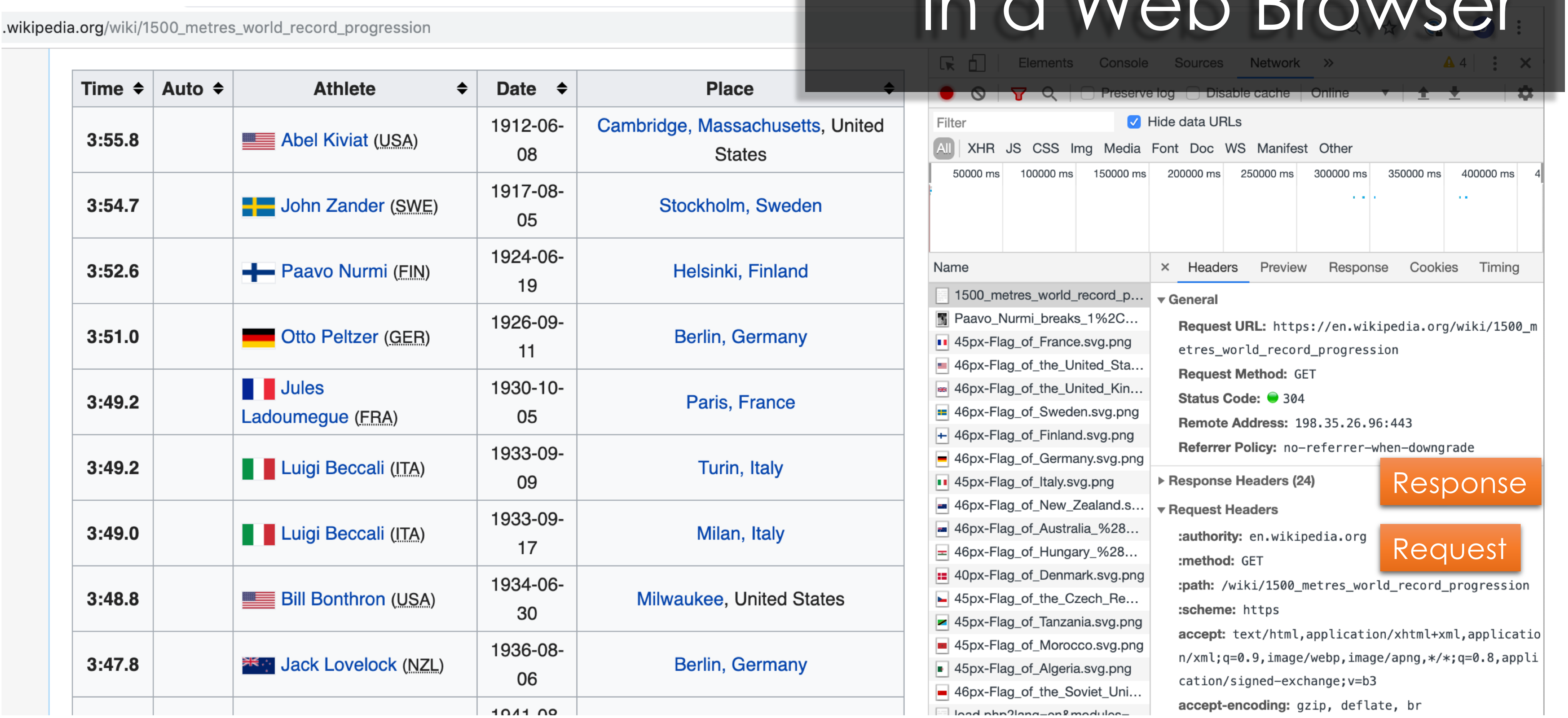
```
HTTP/1.1 200 OK
Date: Mon, 11 Nov 2019 22:20:16 GMT
Content-Type: text/html; charset=UTF-8
Server: mw1266.eqiad.wmnet
Content-Encoding: gzip
Content-Length: 19310
```

Body

```
<!DOCTYPE html>\n<html class="client-nojs" lang="en"
dir="ltr">\n<head>\n<meta charset="UTF-8"/>\n<title>1500 metres
world record progression - Wikipedia</title> …
```

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

In a Web Browser



.wikipedia.org/wiki/1500_metres_world_record_progression

Time	Auto	Athlete	Date	Place
3:55.8		Abel Kiviat (USA)	1912-06-08	Cambridge, Massachusetts, United States
3:54.7		John Zander (SWE)	1917-08-05	Stockholm, Sweden
3:52.6		Paavo Nurmi (FIN)	1924-06-19	Helsinki, Finland
3:51.0		Otto Peltzer (GER)	1926-09-11	Berlin, Germany
3:49.2		Jules Ladoumegue (FRA)	1930-10-05	Paris, France
3:49.2		Luigi Beccali (ITA)	1933-09-09	Turin, Italy
3:49.0		Luigi Beccali (ITA)	1933-09-17	Milan, Italy
3:48.8		Bill Bonthron (USA)	1934-06-30	Milwaukee, United States
3:47.8		Jack Lovelock (NZL)	1936-08-06	Berlin, Germany
			1941-08	

Name:
- 1500_metres_world_record_p…
- Paavo_Nurmi_breaks_1%2C…
- 45px-Flag_of_France.svg.png
- 46px-Flag_of_the_United_Sta…
- 46px-Flag_of_the_United_Kin…
- 46px-Flag_of_Sweden.svg.png
- 46px-Flag_of_Finland.svg.png
- 46px-Flag_of_Germany.svg.png
- 45px-Flag_of_Italy.svg.png
- 46px-Flag_of_New_Zealand.s…
- 46px-Flag_of_Australia_%28…
- 46px-Flag_of_Hungary_%28…
- 40px-Flag_of_Denmark.svg.png
- 45px-Flag_of_the_Czech_Re…
- 45px-Flag_of_Tanzania.svg.png
- 45px-Flag_of_Morocco.svg.png
- 45px-Flag_of_Algeria.svg.png
- 46px-Flag_of_the_Soviet_Uni…
- load.php?lang=en&modules=…

▾ General

Request URL: https://en.wikipedia.org/wiki/1500_metres_world_record_progression
Request Method: GET
Status Code: ● 304
Remote Address: 198.35.26.96:443
Referrer Policy: no-referrer-when-downgrade

▸ Response Headers (24)

▾ Request Headers

:authority: en.wikipedia.org
:method: GET
:path: /wiki/1500_metres_world_record_progression
:scheme: https
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
accept-encoding: gzip, deflate, br

Response

Request

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

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

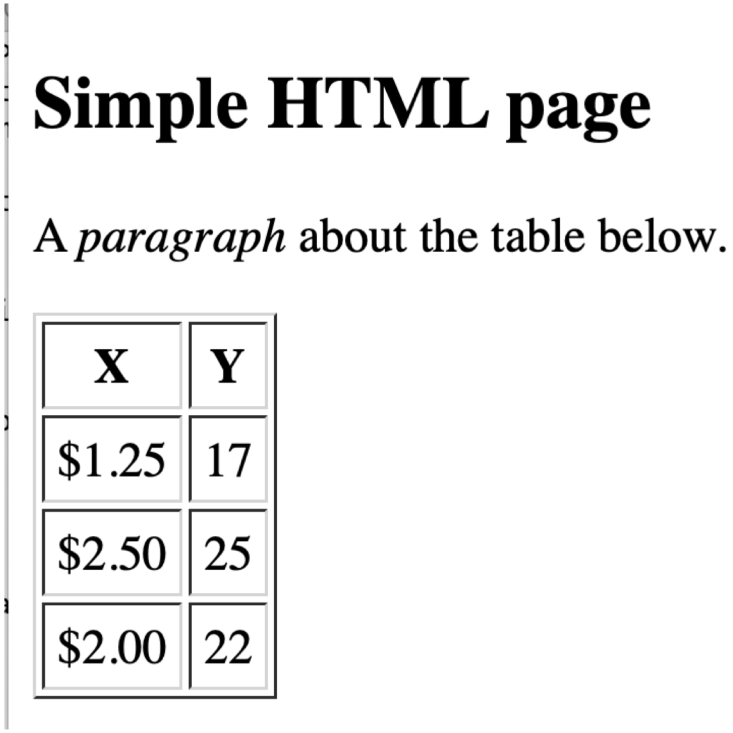
Pandas **read_html**

- 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 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>
```

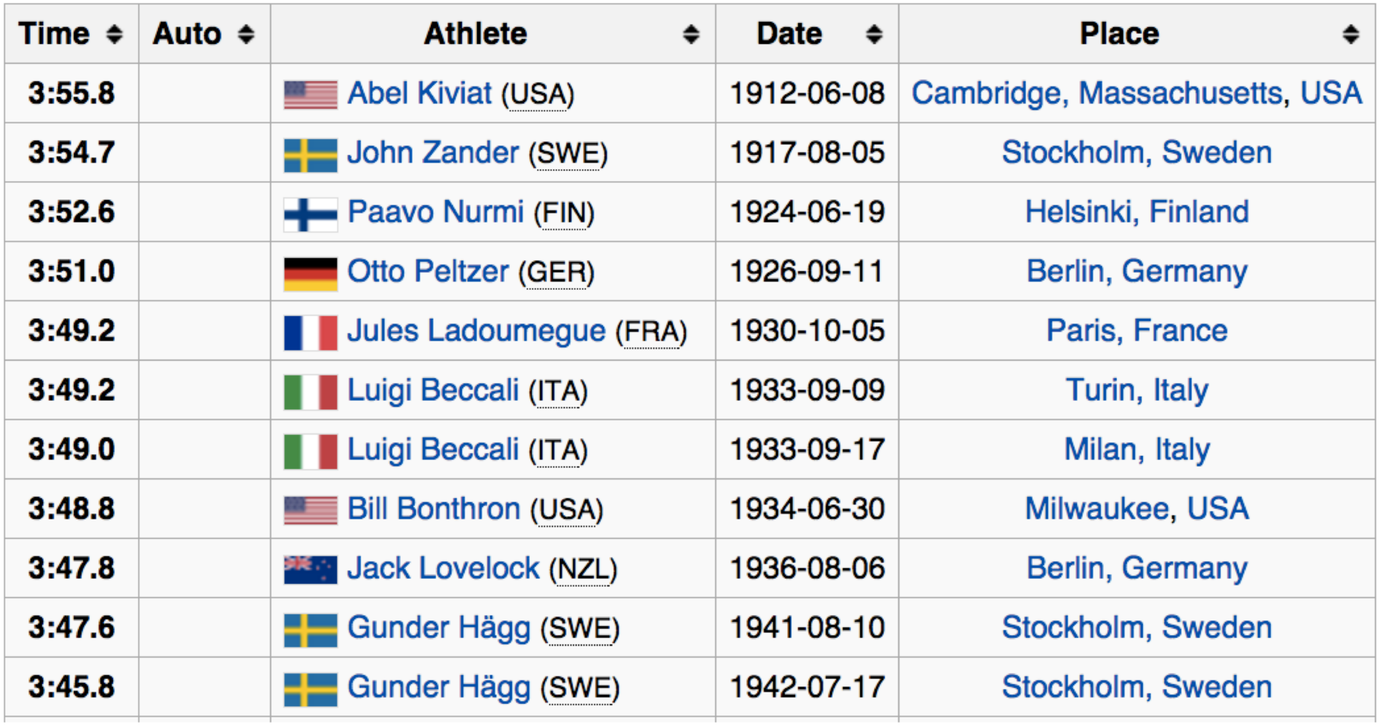


Simple HTML page

A *paragraph* about the table below.

X	Y
$1.25	17
$2.50	25
$2.00	22

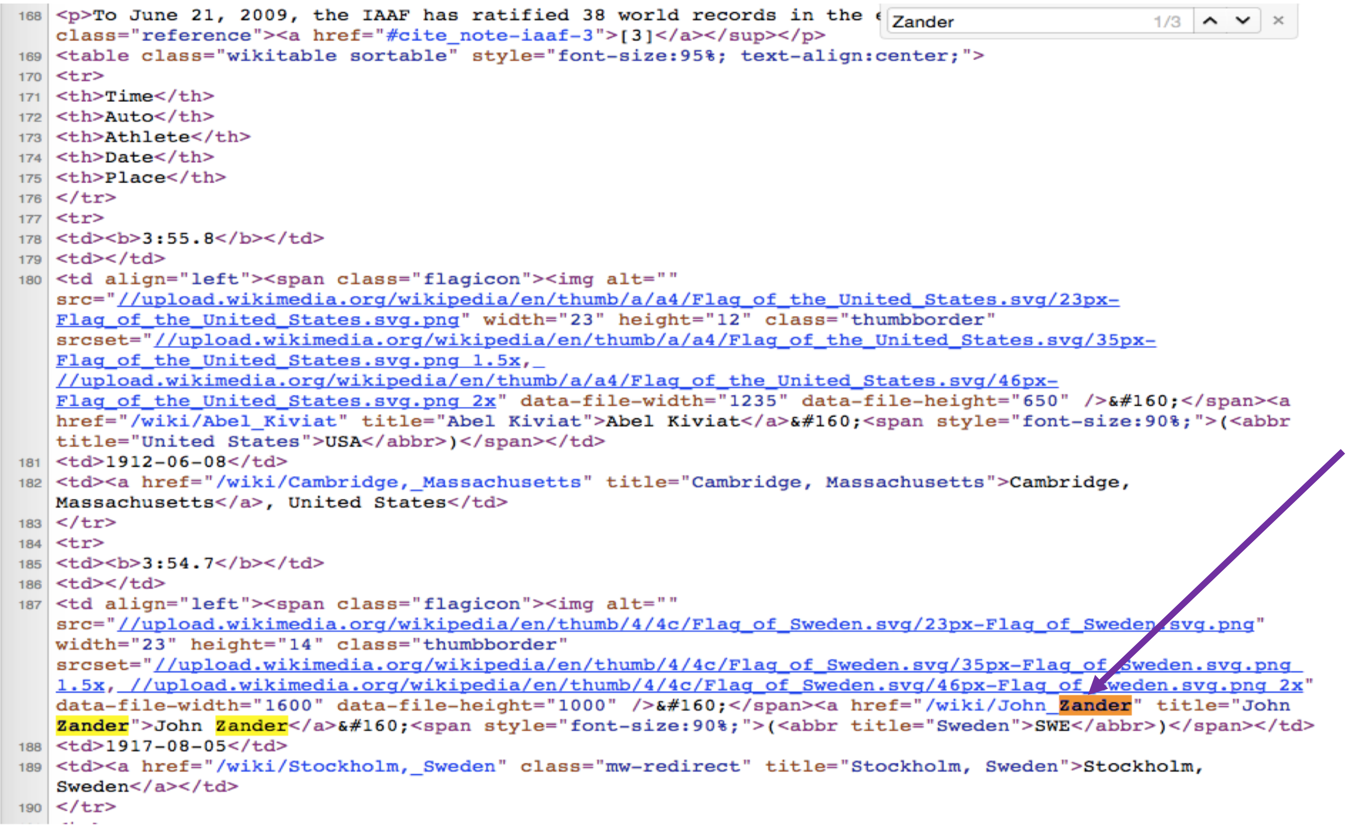
Many Tables on the 1500m page



Time	Auto	Athlete	Date	Place
3:55.8		Abel Kiviat (USA)	1912-06-08	Cambridge, Massachusetts, USA
3:54.7		John Zander (SWE)	1917-08-05	Stockholm, Sweden
3:52.6		Paavo Nurmi (FIN)	1924-06-19	Helsinki, Finland
3:51.0		Otto Peltzer (GER)	1926-09-11	Berlin, Germany
3:49.2		Jules Ladoumegue (FRA)	1930-10-05	Paris, France
3:49.2		Luigi Beccali (ITA)	1933-09-09	Turin, Italy
3:49.0		Luigi Beccali (ITA)	1933-09-17	Milan, Italy
3:48.8		Bill Bonthron (USA)	1934-06-30	Milwaukee, USA
3:47.8		Jack Lovelock (NZL)	1936-08-06	Berlin, Germany
3:47.6		Gunder Hägg (SWE)	1941-08-10	Stockholm, Sweden
3:45.8		Gunder Hägg (SWE)	1942-07-17	Stockholm, Sweden

This is the table we want.

Use Browser to Examine page source

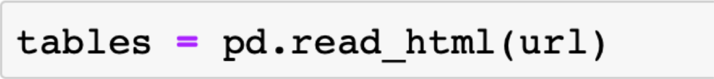


```
<p>To June 21, 2009, the IAAF has ratified 38 world records in the
class="reference"><a href="#cite_note-iaaf-3">[3]</a></sup></p>
<table class="wikitable sortable" style="font-size:95%; text-align:center;">
<tr>
<th>Time</th>
<th>Auto</th>
<th>Athlete</th>
<th>Date</th>
<th>Place</th>
</tr>
<tr>
<td><b>3:55.8</b></td>
<td></td>
<td align="left"><span class="flagicon"><img alt=""
src="//upload.wikimedia.org/wikipedia/en/thumb/a/a4/Flag_of_the_United_States.svg/23px-
Flag_of_the_United_States.svg.png" width="23" height="12" class="thumbborder"
srcset="//upload.wikimedia.org/wikipedia/en/thumb/a/a4/Flag_of_the_United_States.svg/35px-
Flag_of_the_United_States.svg.png 1.5x,
//upload.wikimedia.org/wikipedia/en/thumb/a/a4/Flag_of_the_United_States.svg/46px-
Flag_of_the_United_States.svg.png 2x" data-file-width="1235" data-file-height="650" />&#160;</span><a
href="/wiki/Abel_Kiviat" title="Abel Kiviat">Abel Kiviat</a>&#160;<span style="font-size:90%;">(<abbr
title="United States">USA</abbr>)</span></td>
<td>1912-06-08</td>
<td><a href="/wiki/Cambridge,_Massachusetts" title="Cambridge, Massachusetts">Cambridge,
Massachusetts</a>, United States</td>
</tr>
<tr>
<td><b>3:54.7</b></td>
<td></td>
<td align="left"><span class="flagicon"><img alt=""
src="//upload.wikimedia.org/wikipedia/en/thumb/4/4c/Flag_of_Sweden.svg/23px-Flag_of_Sweden.svg.png"
width="23" height="14" class="thumbborder"
srcset="//upload.wikimedia.org/wikipedia/en/thumb/4/4c/Flag_of_Sweden.svg/35px-Flag_of_Sweden.svg.png
1.5x, //upload.wikimedia.org/wikipedia/en/thumb/4/4c/Flag_of_Sweden.svg/46px-Flag_of_Sweden.svg.png 2x"
data-file-width="1600" data-file-height="1000" />&#160;</span><a href="/wiki/John_Zander" title="John
Zander">John Zander</a>&#160;<span style="font-size:90%;">(<abbr title="Sweden">SWE</abbr>)</span></td>
<td>1917-08-05</td>
<td><a href="/wiki/Stockholm,_Sweden" class="mw-redirect" title="Stockholm, Sweden">Stockholm,
Sweden</a></td>
</tr>
```

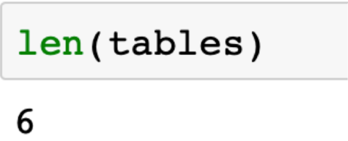
Here’s the HTML for the table we want.

Notice the name Zander

Pandas extracts tables from HTML documents as a list of data frames

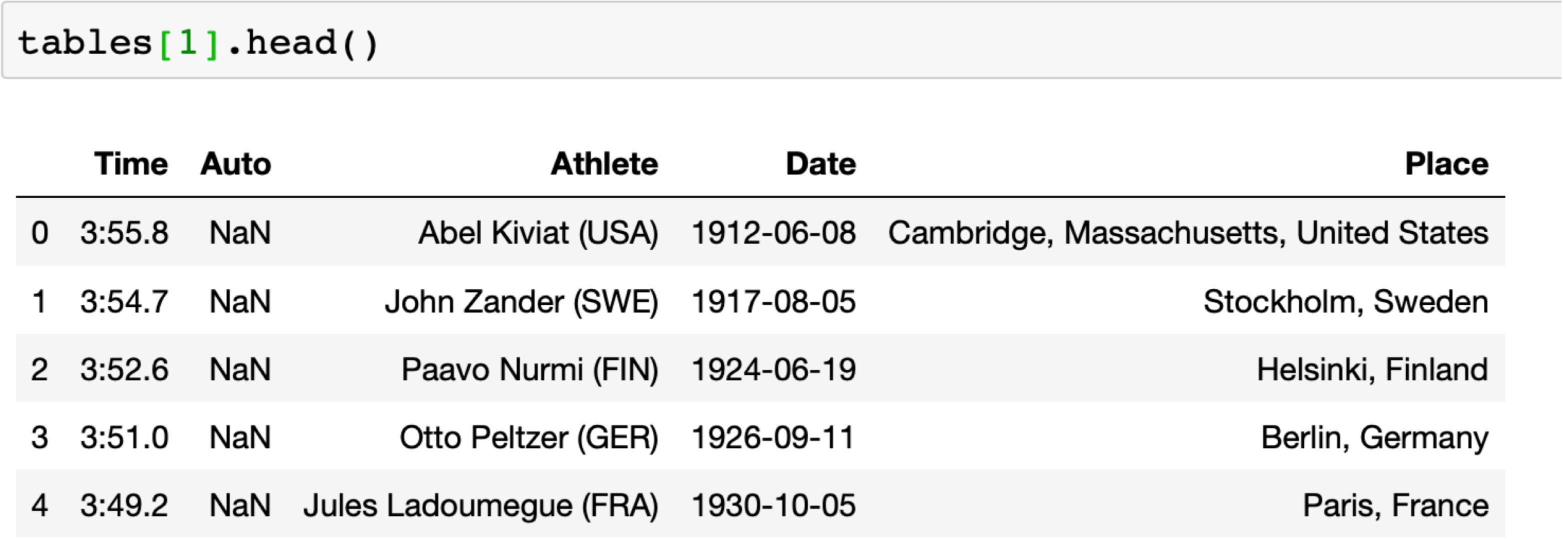


```
tables = pd.read_html(url)
```



```
len(tables)
```

```
6
```

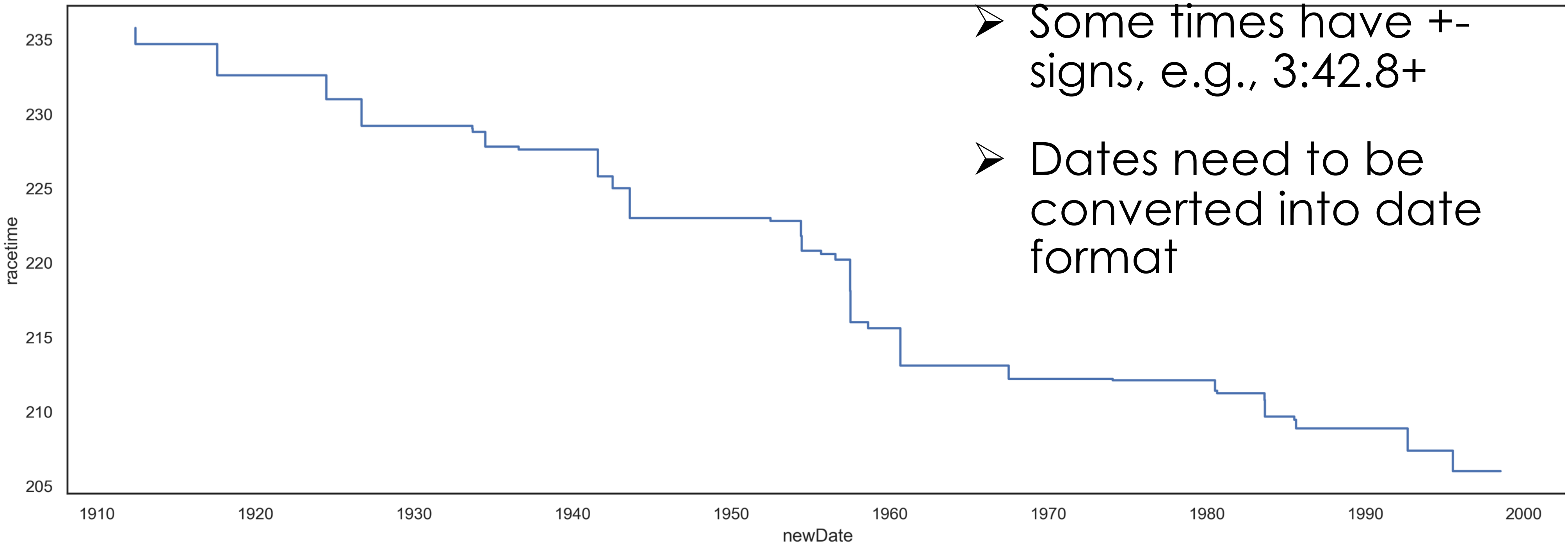


```
tables[1].head()
```

	Time	Auto	Athlete	Date	Place
0	3:55.8	NaN	Abel Kiviat (USA)	1912-06-08	Cambridge, Massachusetts, United States
1	3:54.7	NaN	John Zander (SWE)	1917-08-05	Stockholm, Sweden
2	3:52.6	NaN	Paavo Nurmi (FIN)	1924-06-19	Helsinki, Finland
3	3:51.0	NaN	Otto Peltzer (GER)	1926-09-11	Berlin, Germany
4	3:49.2	NaN	Jules Ladoumegue (FRA)	1930-10-05	Paris, France

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

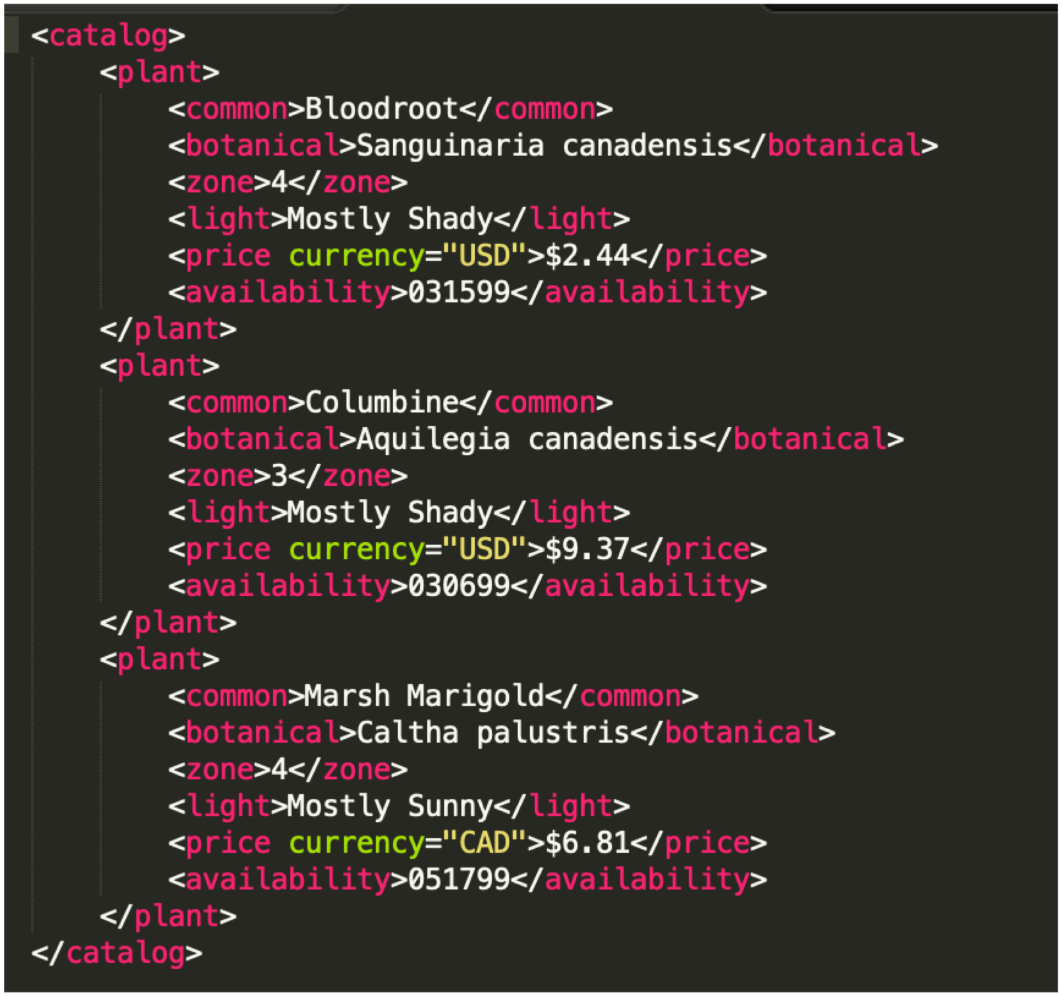


XML

eXtensible Markup Language

HTML/XML/JSON

- 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



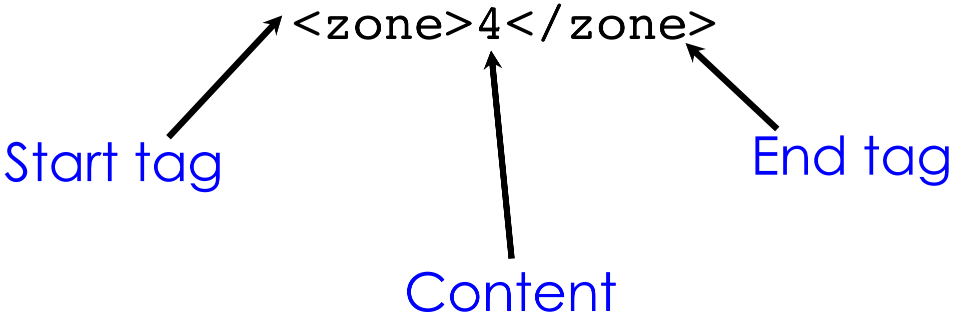
```xml
<catalog>
    <plant>
        <common>Bloodroot</common>
        <botanical>Sanguinaria canadensis</botanical>
        <zone>4</zone>
        <light>Mostly Shady</light>
        <price currency="USD">$2.44</price>
        <availability>031599</availability>
    </plant>
    <plant>
        <common>Columbine</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>
```

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>
```

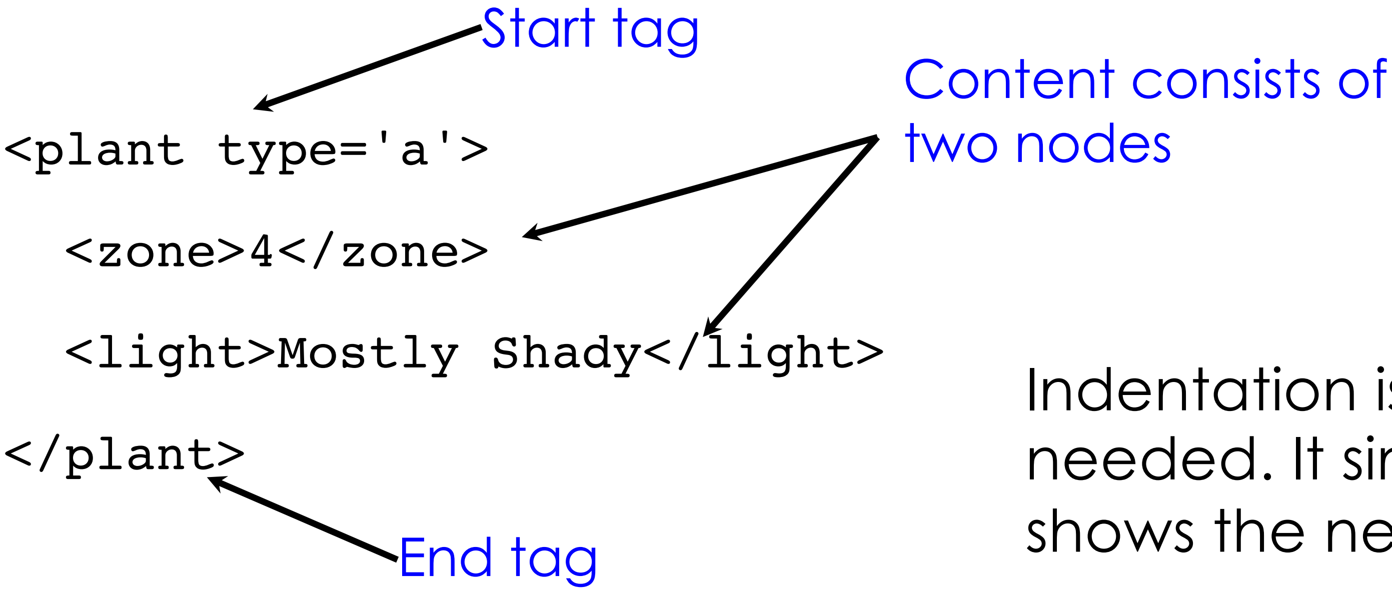
Start tag

Content

End tag

Syntax: Nesting

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



Start tag

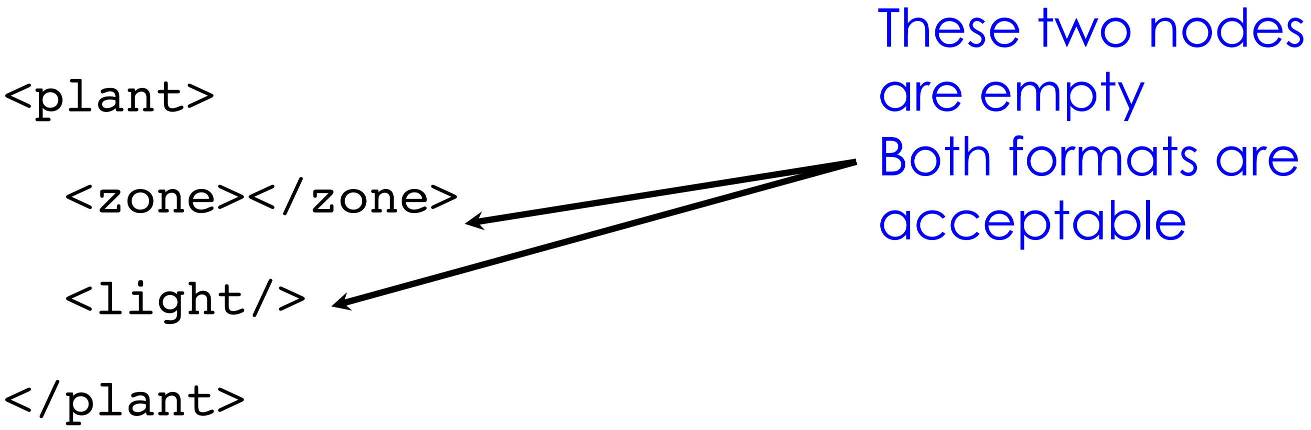
Content consists of two nodes

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

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)

The attribute named type has a value of “a”

This empty node has two attributes: source and class

```
<plant type='a'>

  <zone></zone>

  <light source="2" class="new"/>

</plant>
```

Syntax: Comments

Comments can appear anywhere

Two comments

```
<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.

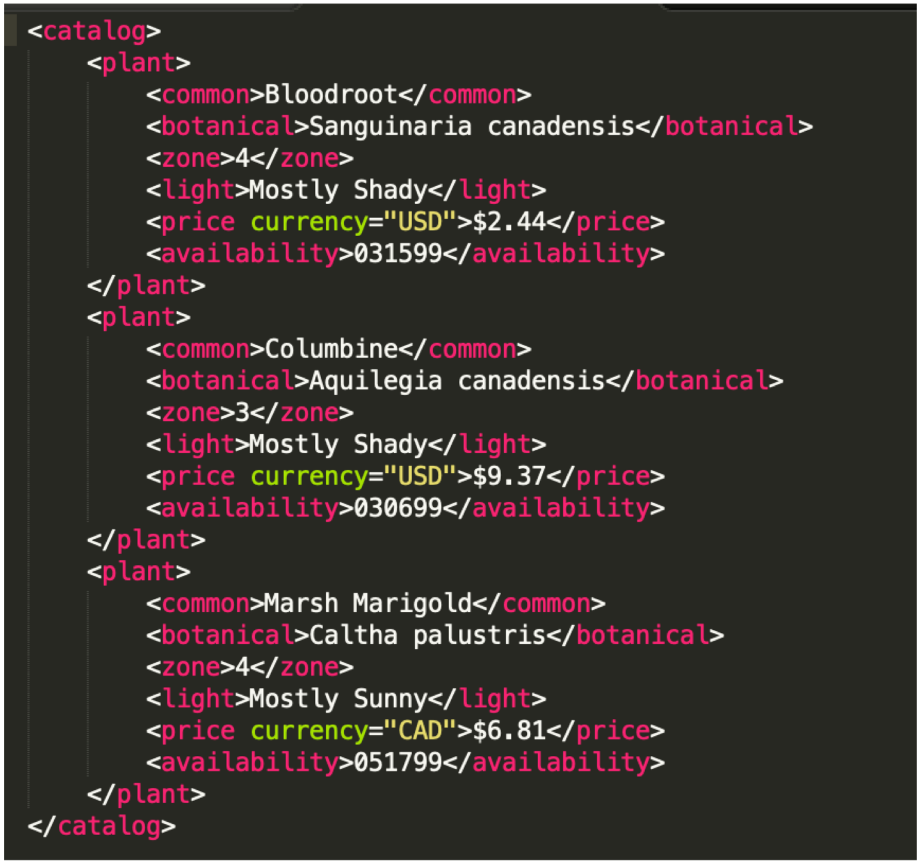
xHTML: Extensible Hypertext Markup Language

- 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

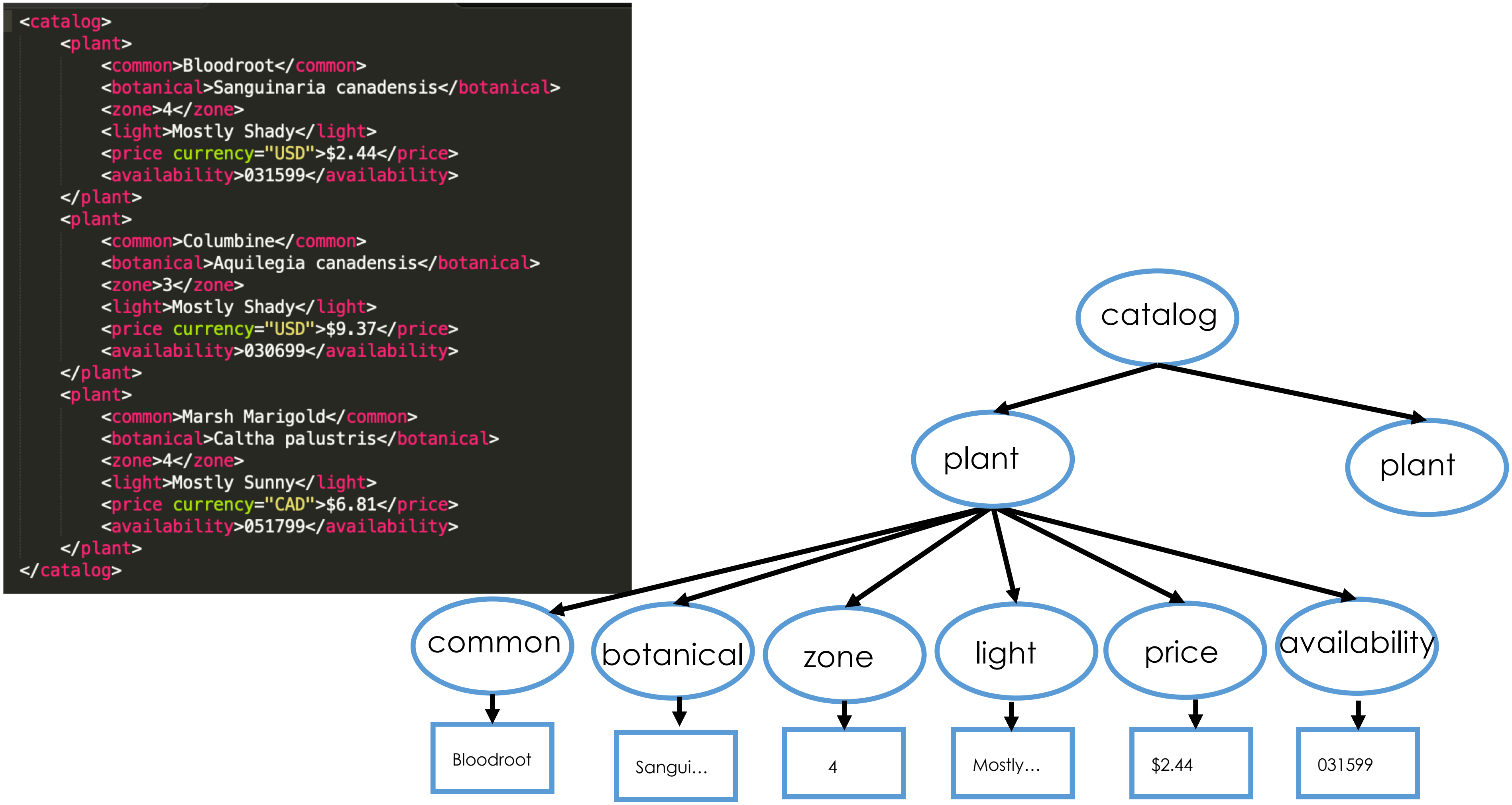


```
<catalog>
    <plant>
        <common>Bloodroot</common>
        <botanical>Sanguinaria canadensis</botanical>
        <zone>4</zone>
        <light>Mostly Shady</light>
        <price currency="USD">$2.44</price>
        <availability>031599</availability>
    </plant>
    <plant>
        <common>Columbine</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>
```

- Treat XML & HTML as a Tree
  - Fits XML and well-formed HTML
- Visual containment → children
- Manipulated dynamically using JavaScript
  - Parsing in Python → 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

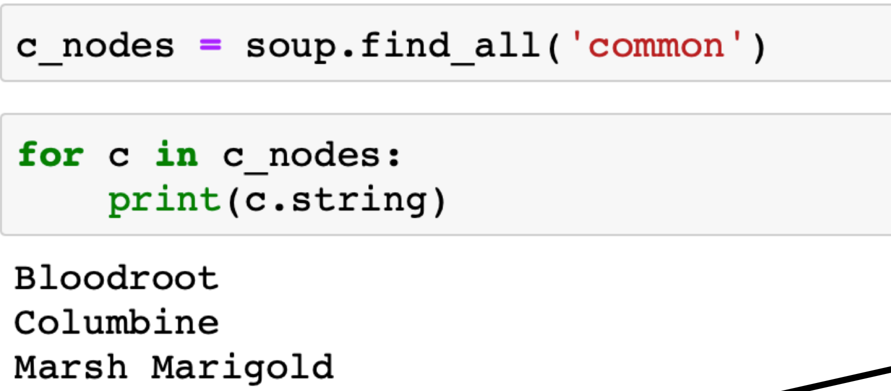
1. Retrieve common names of all plants
2. Retrieve plants that grow in zone 4
3. Retrieve common names of plants that grow in zone 4
4. Retrieve prices of plants whose prices are listed in USD

Beautiful Soup

Locate nodes and content in a well-formed XML document

1. 🟩

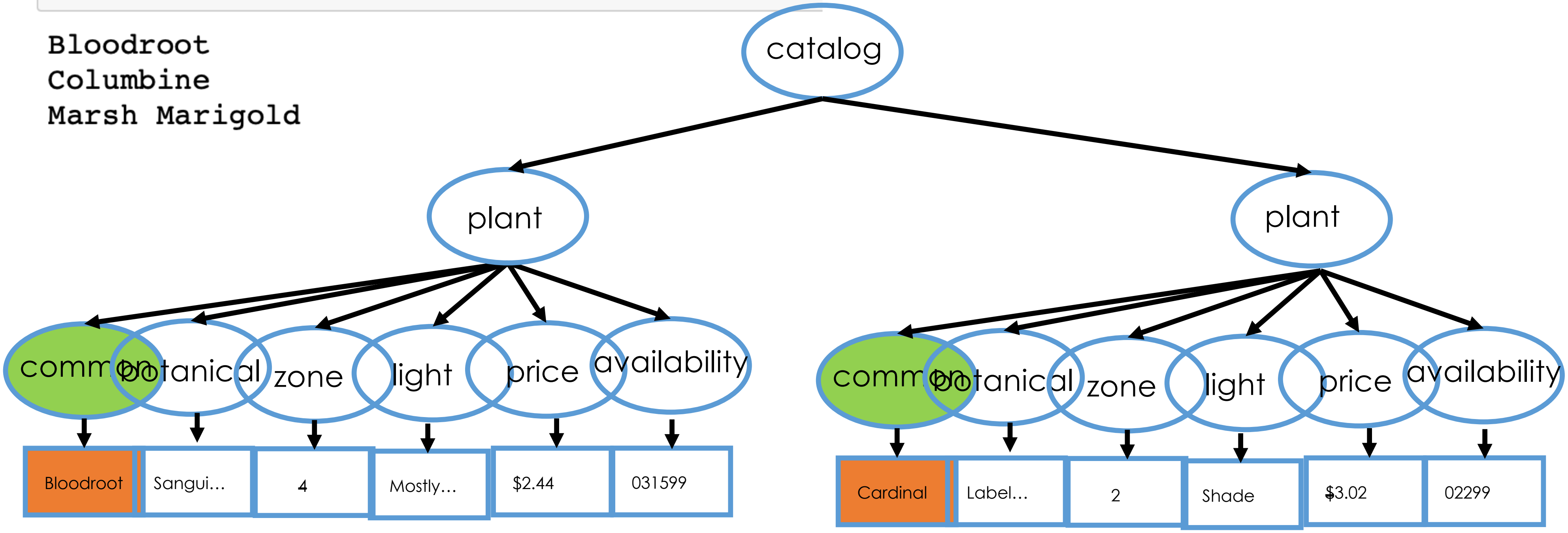
2. 🟧



```
c_nodes = soup.find_all('common')
```

```
for c in c_nodes:
    print(c.string)
```

```
Bloodroot
Columbine
Marsh Marigold
```



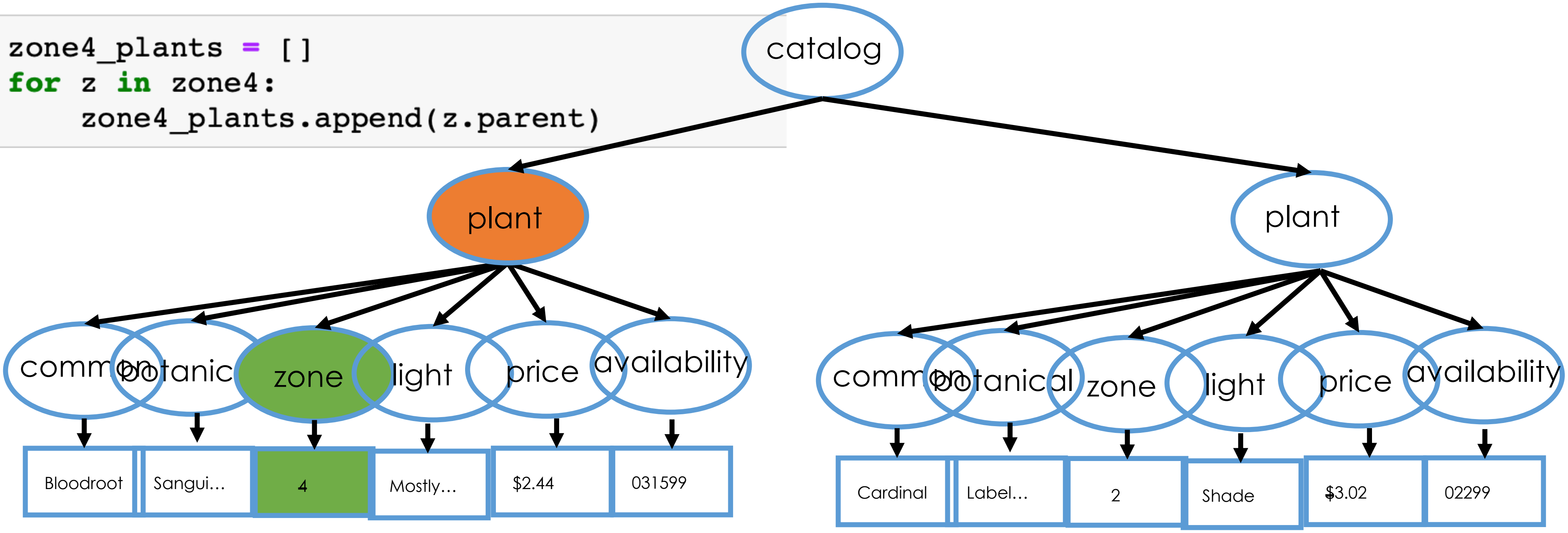
```
zone4 = soup.find_all('zone', text = "4")

zone4
```

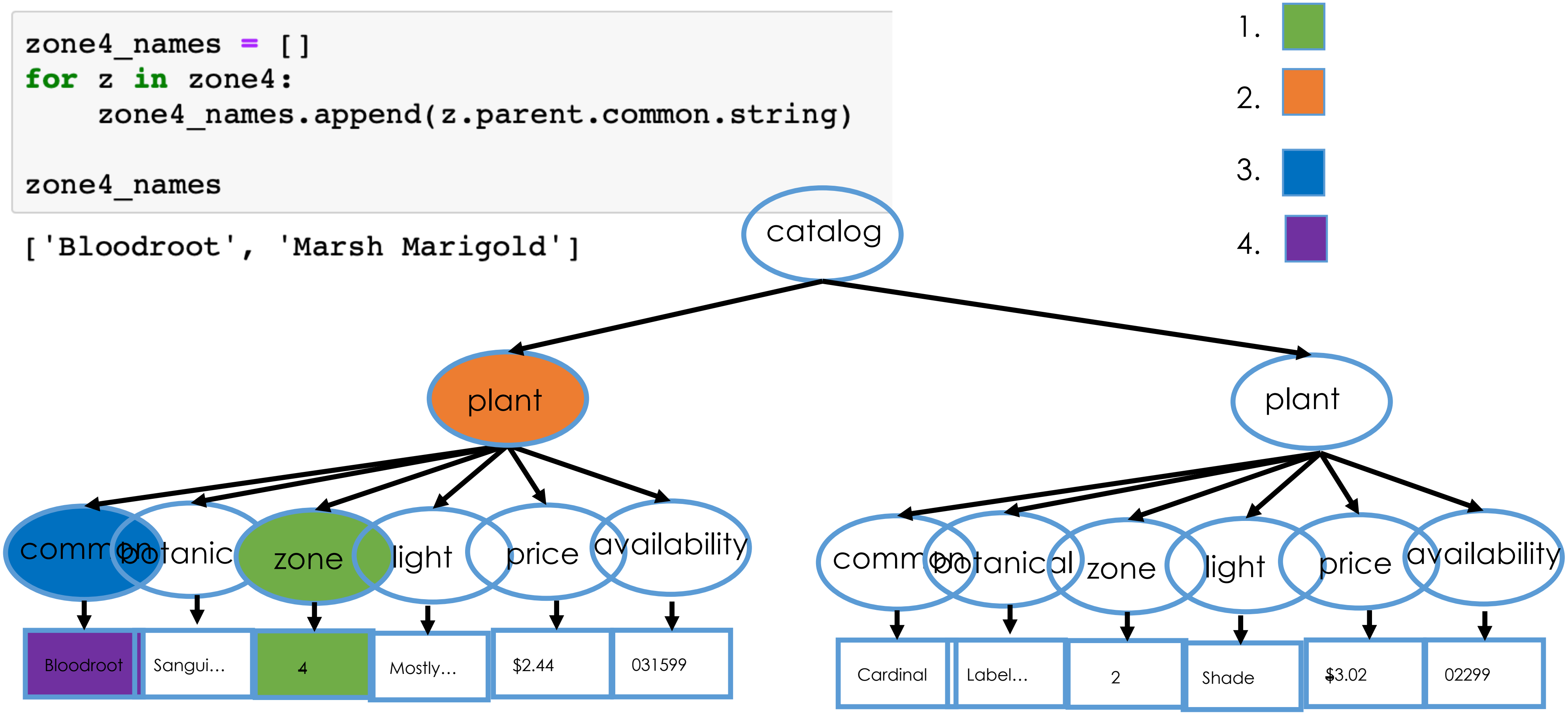
```
[<zone>4</zone>, <zone>4</zone>]
```

1. 🟩

2. 🟧

```
zone4_plants = []
for z in zone4:
    zone4_plants.append(z.parent)
```





```
zone4_names = []
for z in zone4:
    zone4_names.append(z.parent.common.string)

zone4_names
```

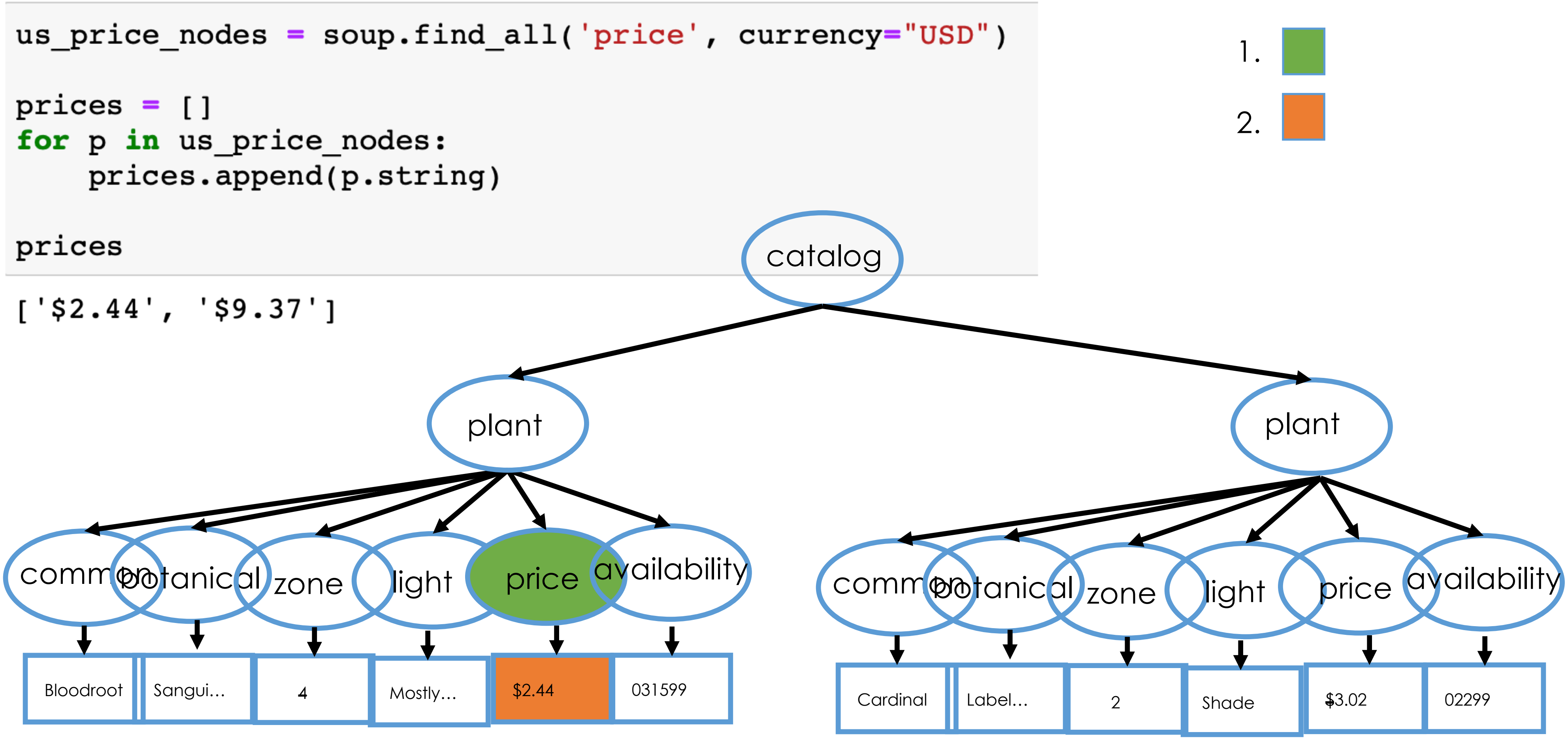
```
['Bloodroot', 'Marsh Marigold']
```

1. 🟩

2. 🟧

3. 🟦

4. 🟪



```
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']
```

1. 🟩

2. 🟧

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

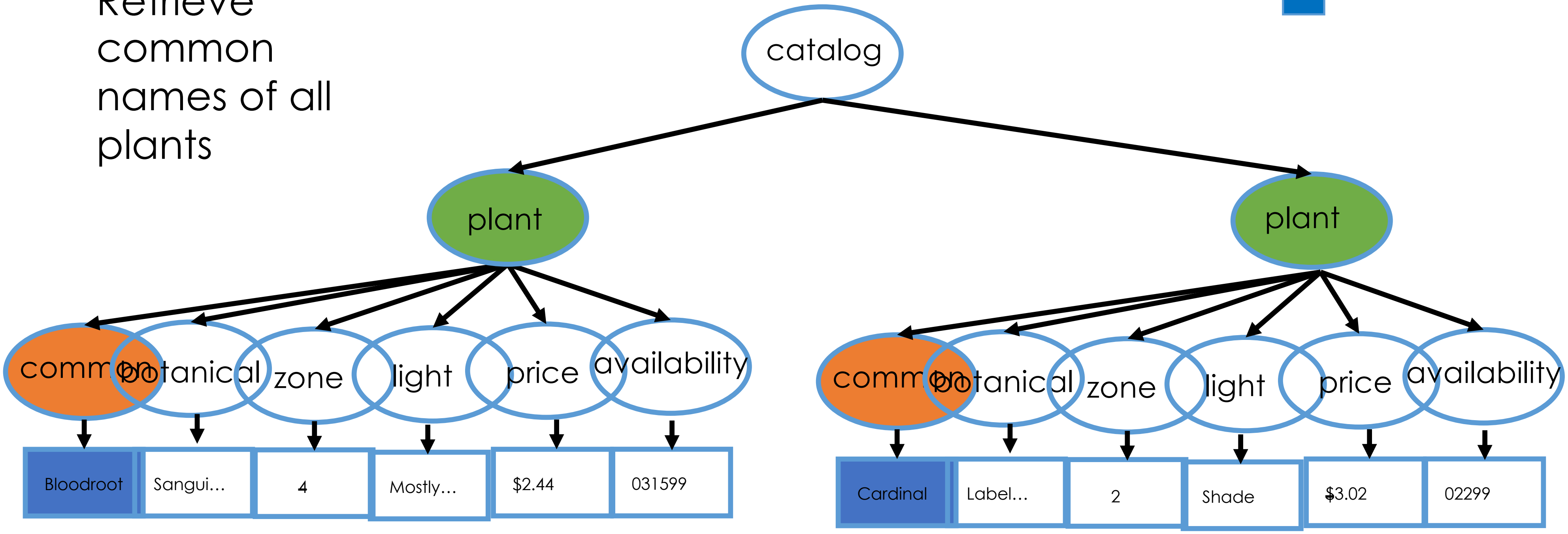
Four Tasks

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

//plant/common/text()

Retrieve common names of all plants

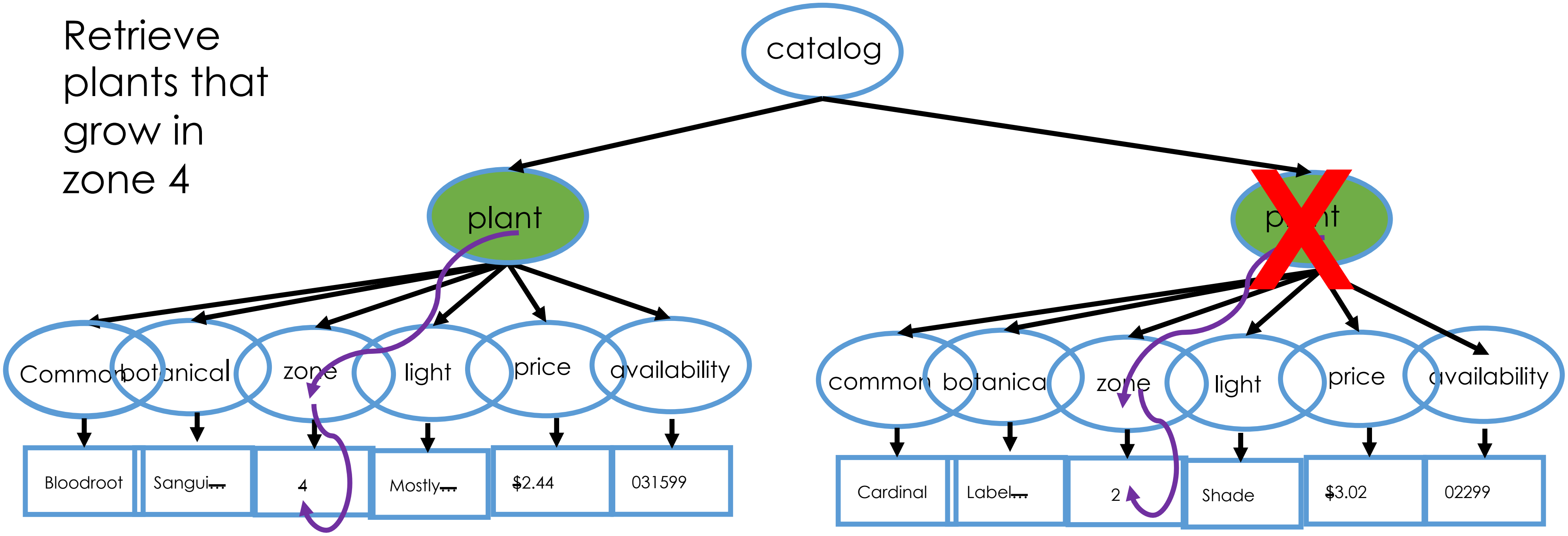
1. 🟩
2. 🟧
3. 🟦



1. 🟩

//plant[zone/text() = ‘4’]

Retrieve plants that grow in zone 4



/catalog/plant/zone/

//zone

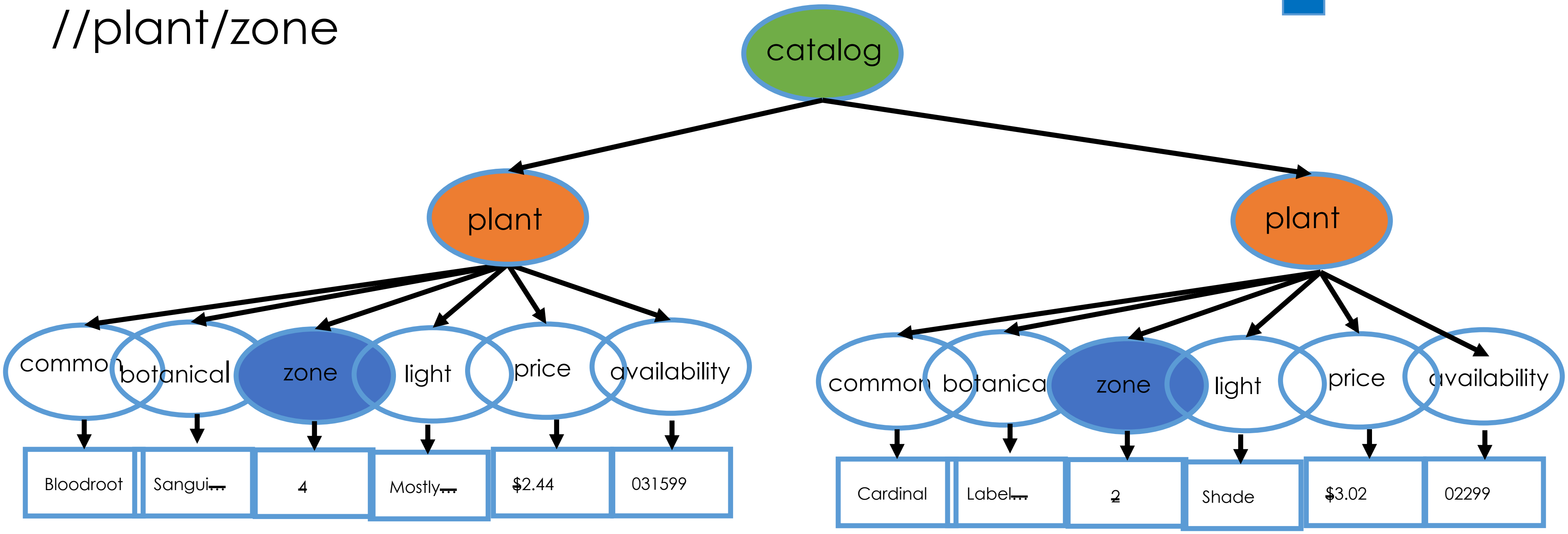
//plant/zone

3 ways to locate zone nodes

1. 🟩

2. 🟧

3. 🟦



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

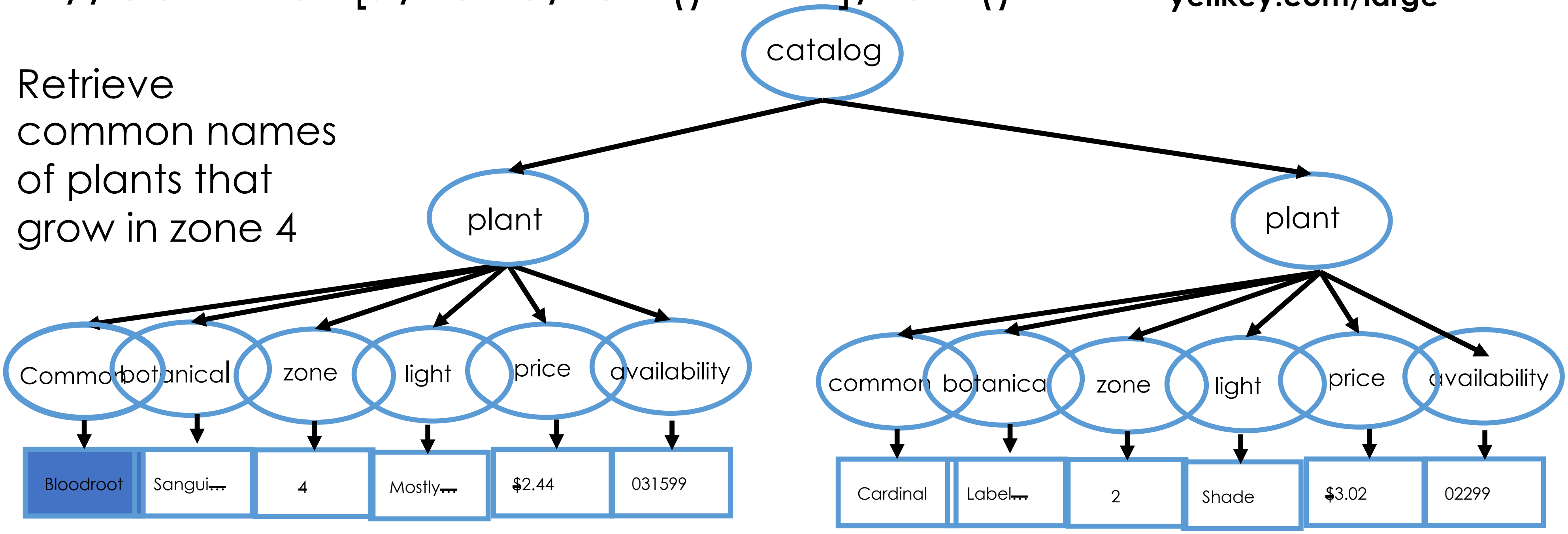
/catalog/plant/zone[text() = ‘4’]/common/text()

//plant[zone/text() = ‘4’]/common/text()

//common[../zone/text() = ‘4’]/text()

yellkey.com/large

Retrieve common names of plants that grow in zone 4



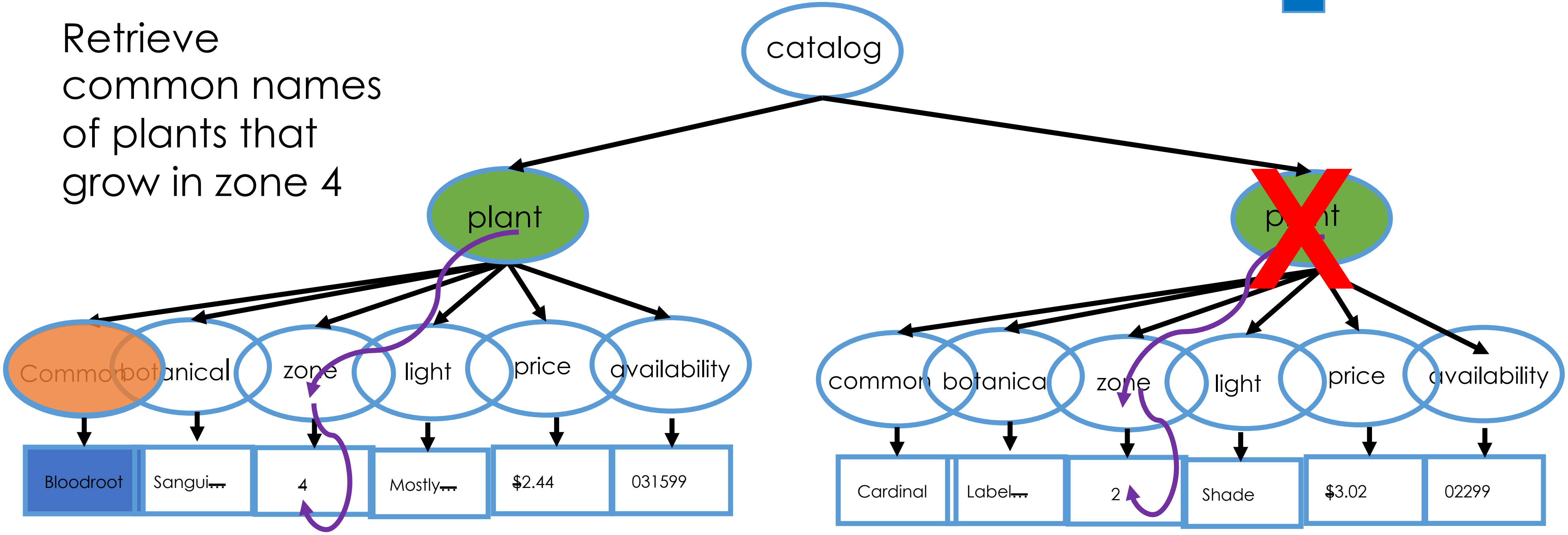
//plant[zone/text() = ‘4’]/common/text()

1. 🟩

2. 🟧

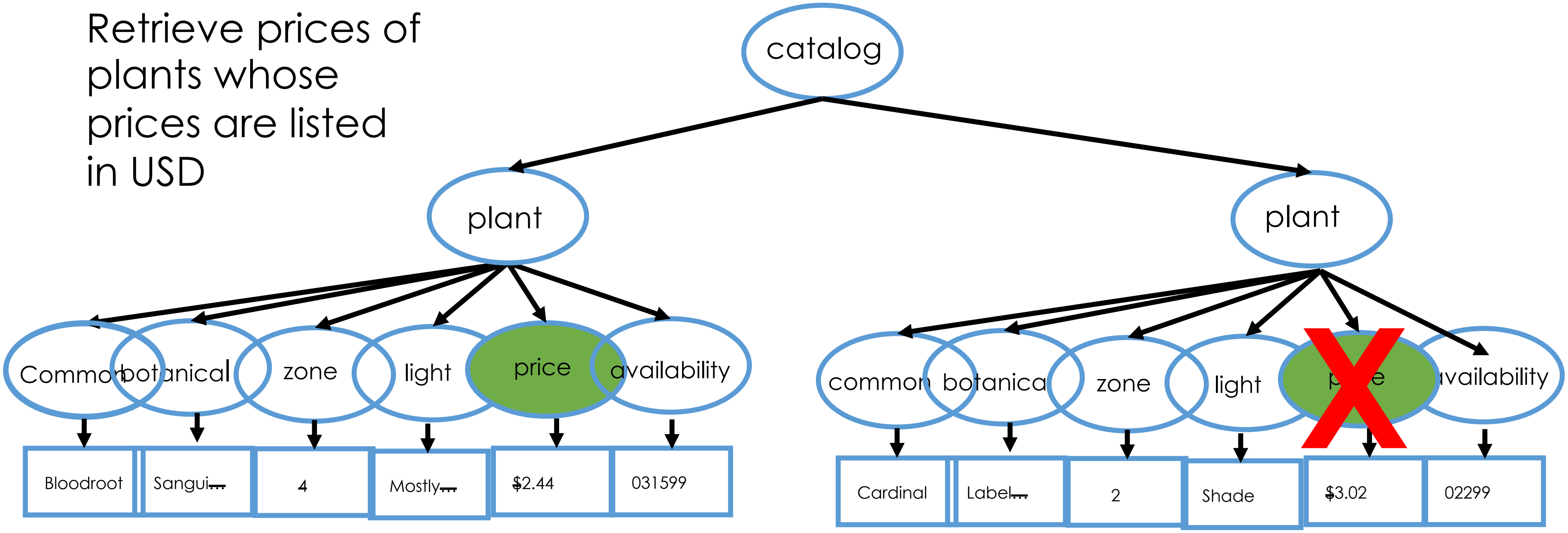
3. 🟦

Retrieve common names of plants that grow in zone 4



//price[@currency= ‘USD’]

Retrieve prices of plants whose prices are listed in 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

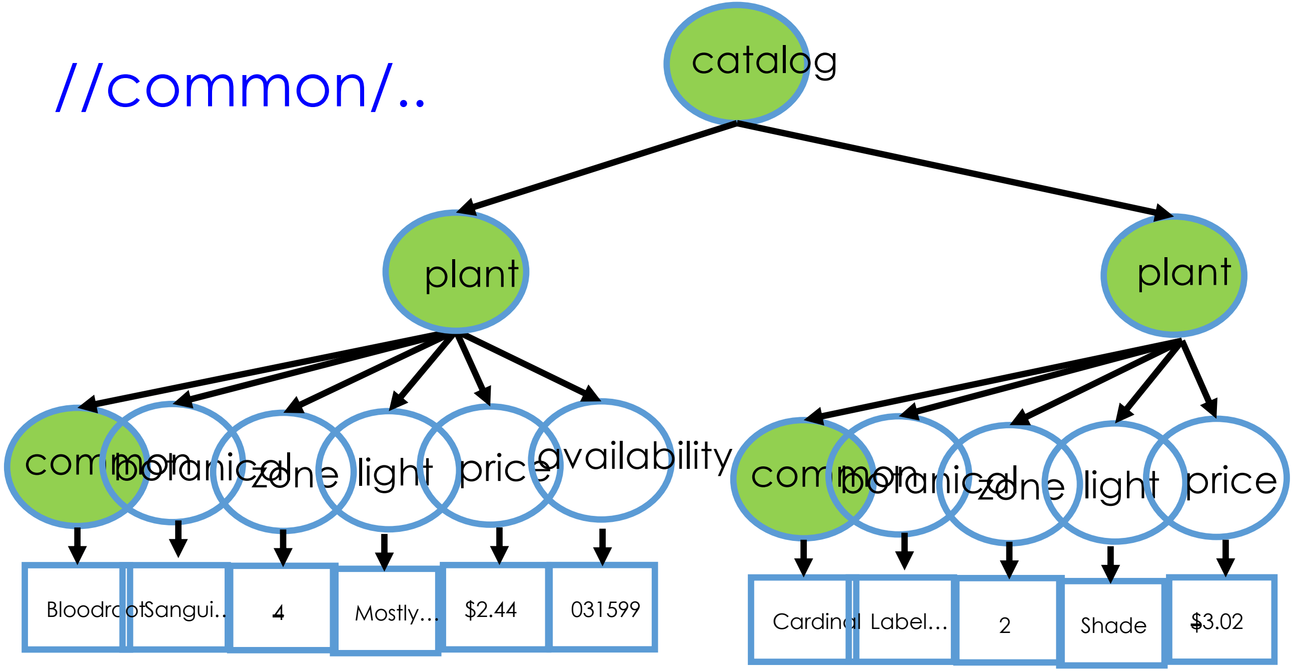
- “child”, which is the default and can be dropped,
- “descendant-or-self”, which looks anywhere down the tree from current node(s) is abbreviated by “//”
- “self” is abbreviated with a .
- “parent” is abbreviated to ..

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

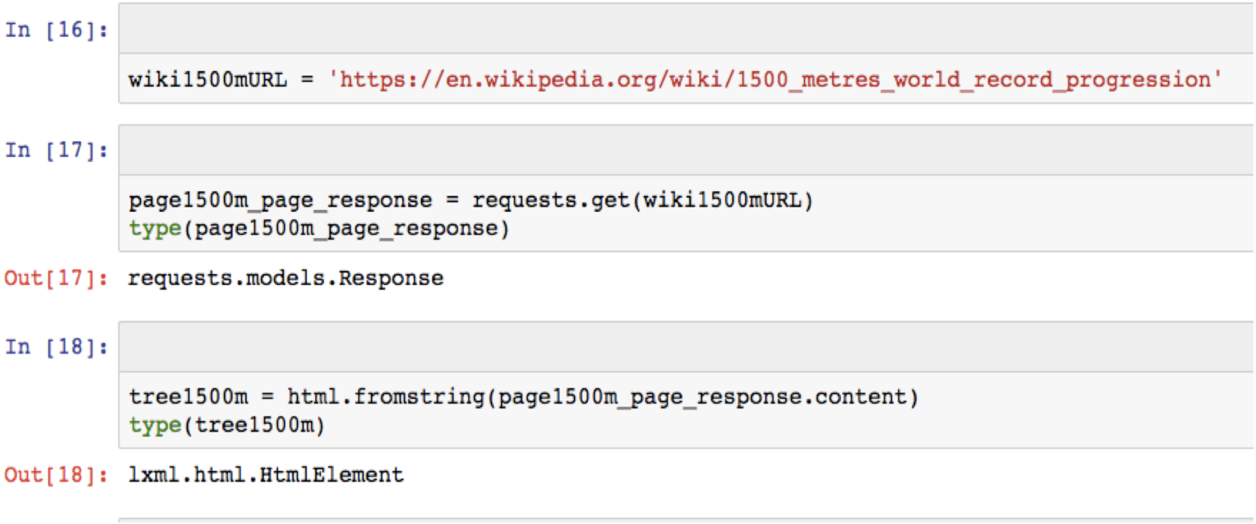
- The *predicate* filters the qualifying nodes, i.e., takes a subset of them.
- The predicate is optional and for our purposes will either be
  - a number, which asks for a specific element, e.g. [2] for the second node
  - an attribute filter, e.g.,

```
//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/



```
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

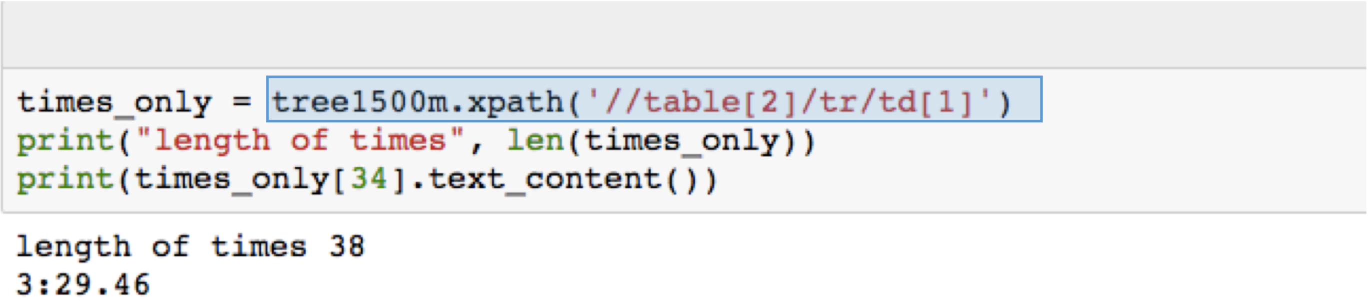
Create an HTML "tree"

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?

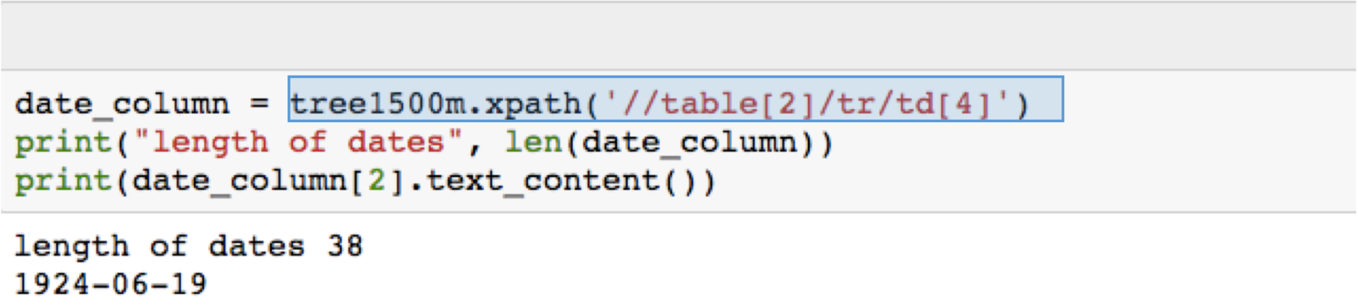
Extract the run times



```
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
```

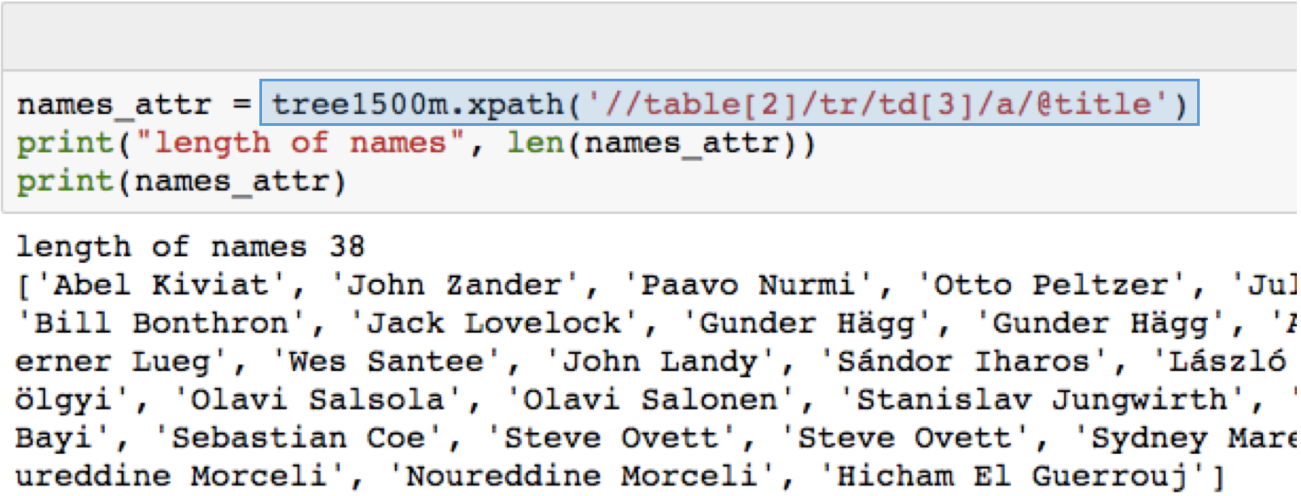
Extract the dates



```
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



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

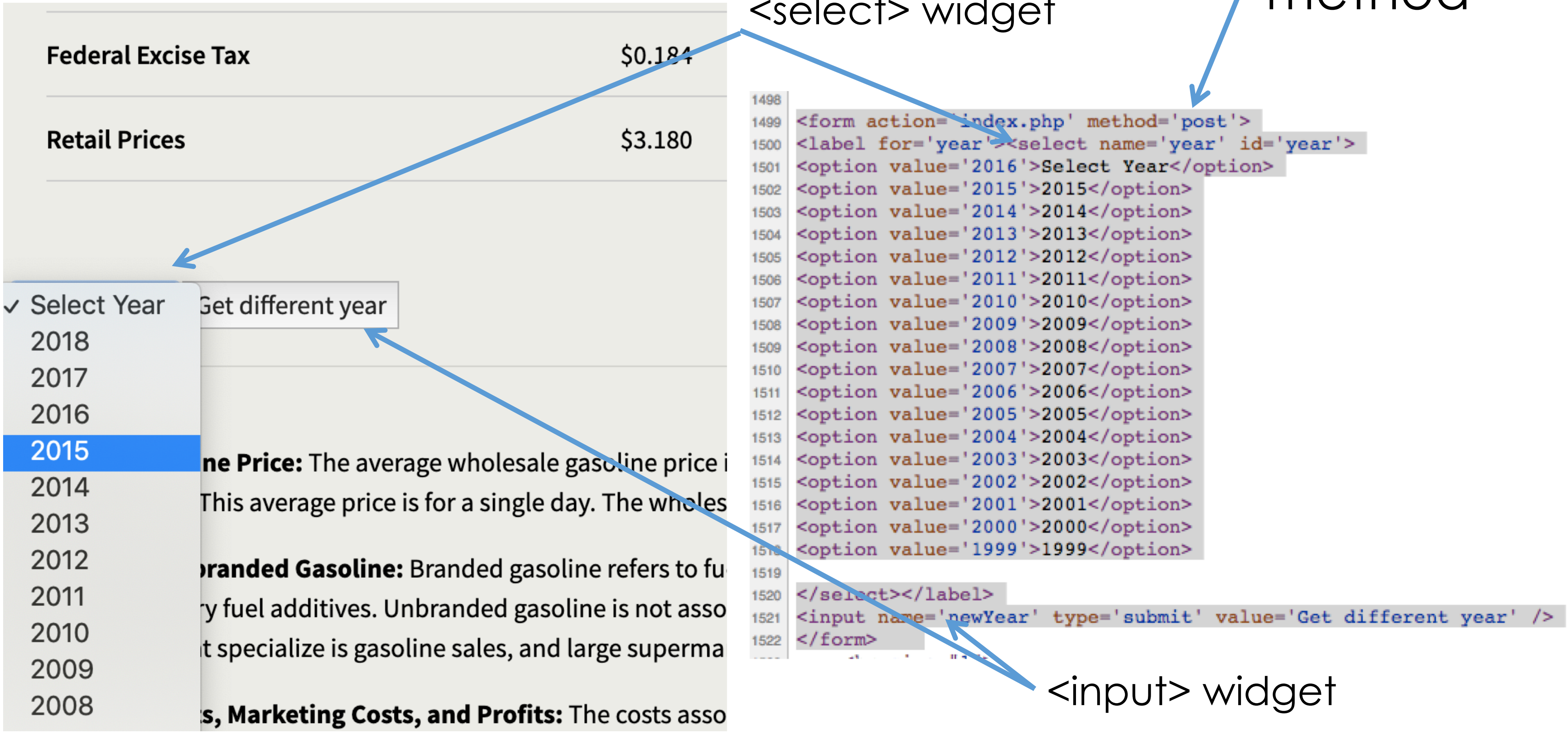
```
length of names 38
['Abel Kiviat', 'John Zander', 'Paavo Nurmi', 'Otto Peltzer', 'Jul
'Bill Bonthron', 'Jack Lovelock', 'Gunder Hägg', 'Gunder Hägg', 'A
erner Lueg', 'Wes Santee', 'John Landy', 'Sándor Iharos', 'László
ölgyi', 'Olavi Salsola', 'Olavi Salonen', 'Stanislav Jungwirth', '
Bayi', 'Sebastian Coe', 'Steve Ovett', 'Steve Ovett', 'Sydney Mare
ureddine Morceli', 'Noureddine Morceli', 'Hicham El Guerrouj']
```

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



Federal Excise Tax	$0.184
Retail Prices	$3.180

ne Price: The average wholesale gasoline price i
This average price is for a single day. The wholes

randed Gasoline: Branded gasoline refers to fu
y fuel additives. Unbranded gasoline is not asso
t specialize is gasoline sales, and large superma

s, Marketing Costs, and Profits: The costs asso

```
<form action='index.php' method='post'>
<label for='year'><select name='year' id='year'>
<option value='2016'>Select Year</option>
<option value='2015'>2015</option>
<option value='2014'>2014</option>
<option value='2013'>2013</option>
<option value='2012'>2012</option>
<option value='2011'>2011</option>
<option value='2010'>2010</option>
<option value='2009'>2009</option>
<option value='2008'>2008</option>
<option value='2007'>2007</option>
<option value='2006'>2006</option>
<option value='2005'>2005</option>
<option value='2004'>2004</option>
<option value='2003'>2003</option>
<option value='2002'>2002</option>
<option value='2001'>2001</option>
<option value='2000'>2000</option>
<option value='1999'>1999</option>

</select></label>
<input name='newYear' type='submit' value='Get different year' />
</form>
```

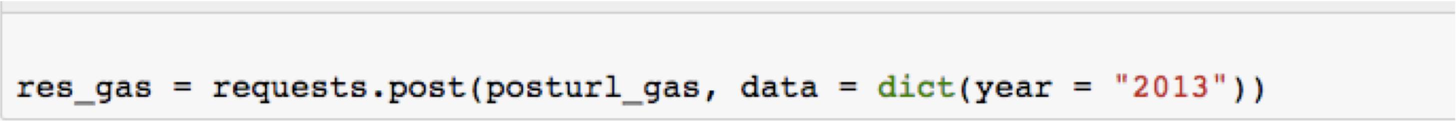
<select> widget

POST method

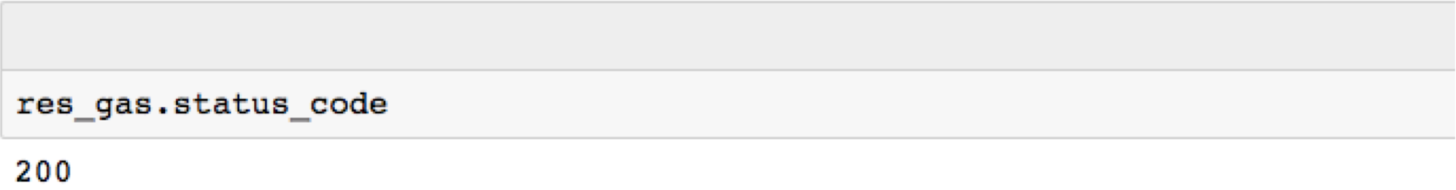
<input> widget

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

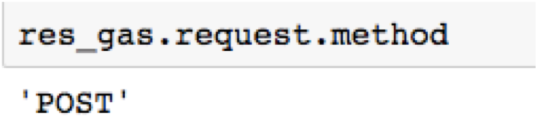


```
res_gas = requests.post(posturl_gas, data = dict(year = "2013"))
```



```
res_gas.status_code
```

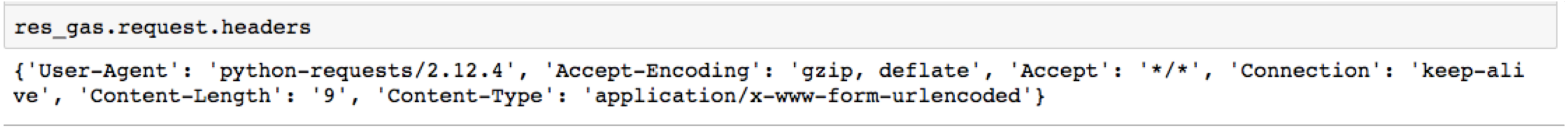
```
200
```



```
res_gas.request.method
```

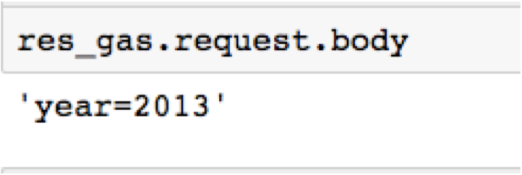
```
'POST'
```

Notice the POST method



```
res_gas.request.headers
```

```
{'User-Agent': 'python-requests/2.12.4', 'Accept-Encoding': 'gzip, deflate', 'Accept': '*/*', 'Connection': 'keep-ali
ve', 'Content-Length': '9', 'Content-Type': 'application/x-www-form-urlencoded'}
```



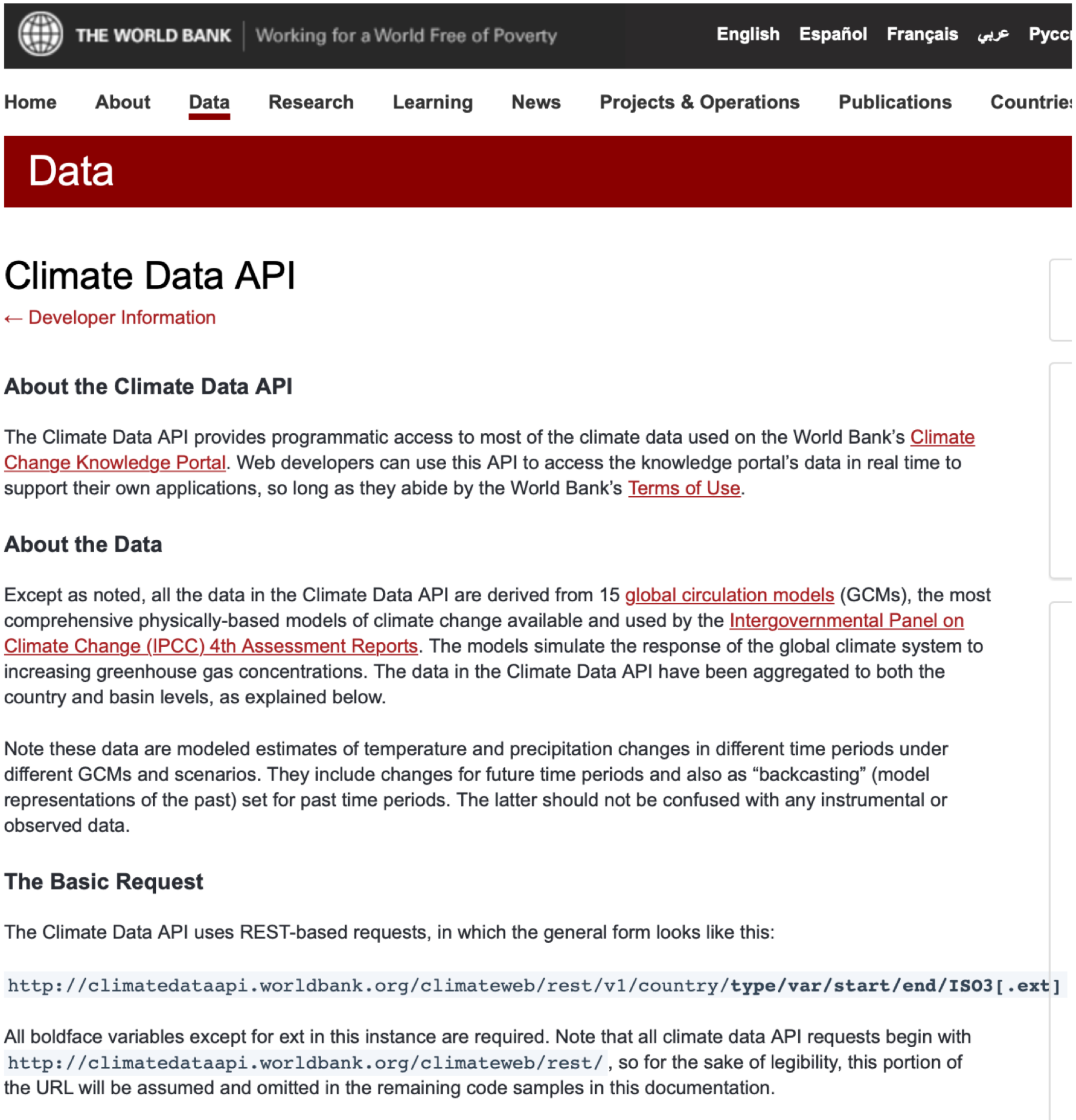
```
res_gas.request.body
```

```
'year=2013'
```

The body of the POST request contains the form information

Ex 3. A REST request for climate simulation data

REST - Representational State Transfer



Data

Climate Data API

← Developer Information

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.

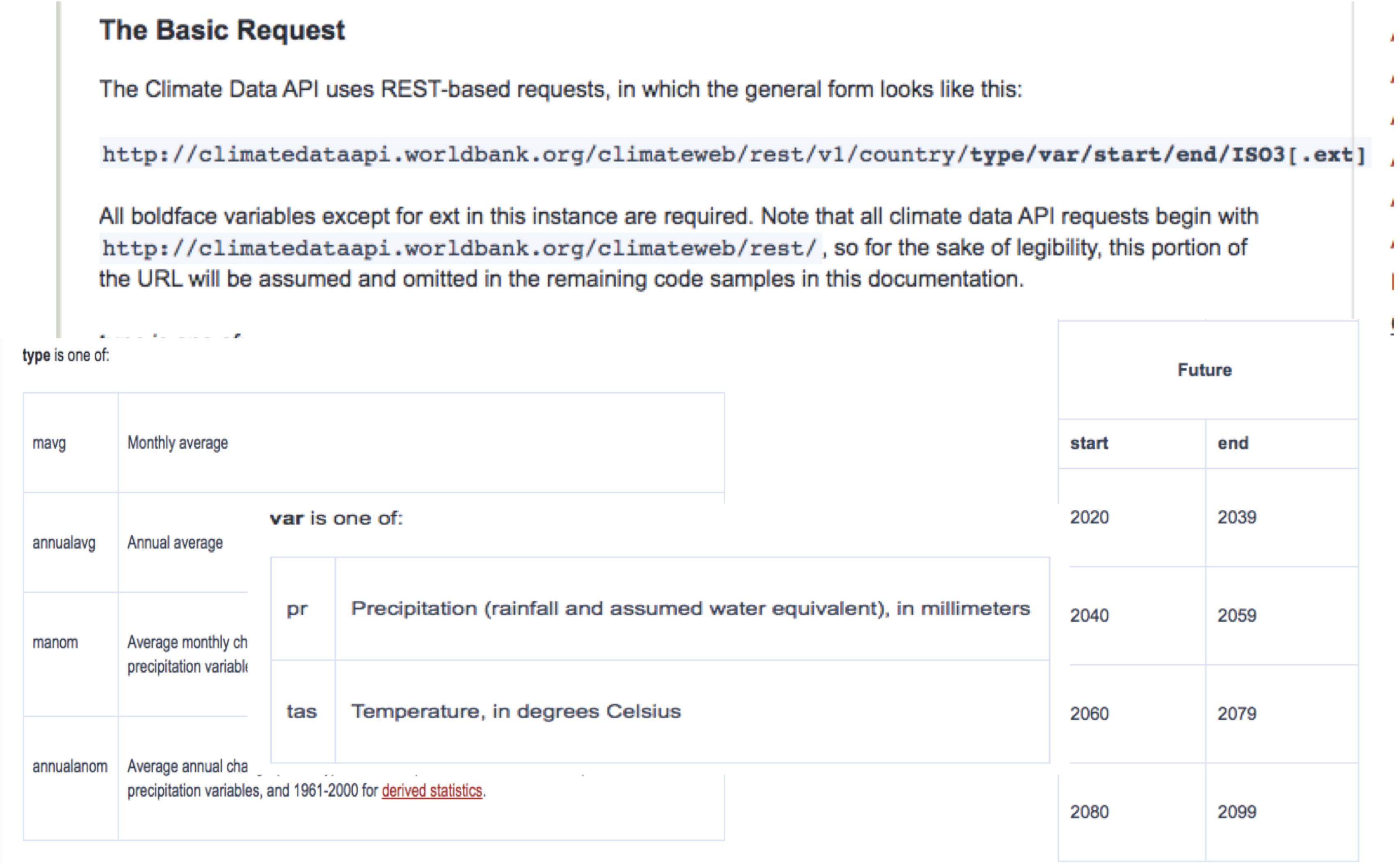
Note these data are modeled estimates of temperature and precipitation changes in different time periods under different GCMs and scenarios. They include changes for future time periods and also as "backcasting" (model representations of the past) set for past time periods. The latter should not be confused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

```
http://climatedataapi.worldbank.org/climateweb/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/climateweb/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/climateweb/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/climateweb/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.

type is one of:

mavg	Monthly average
annualavg	Annual average
manom	Average monthly ch precipitation variabl
annualanom	Average annual cha precipitation variables, and 1961-2000 for derived statistics.

var is one of:

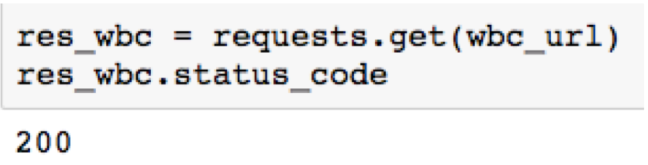
pr	Precipitation (rainfall and assumed water equivalent), in millimeters
tas	Temperature, in degrees Celsius

Future	
start	**end**
2020	2039
2040	2059
2060	2079
2080	2099

World Bank Climate Data REST requests

- From documentation, we need to create requests with URLs like:

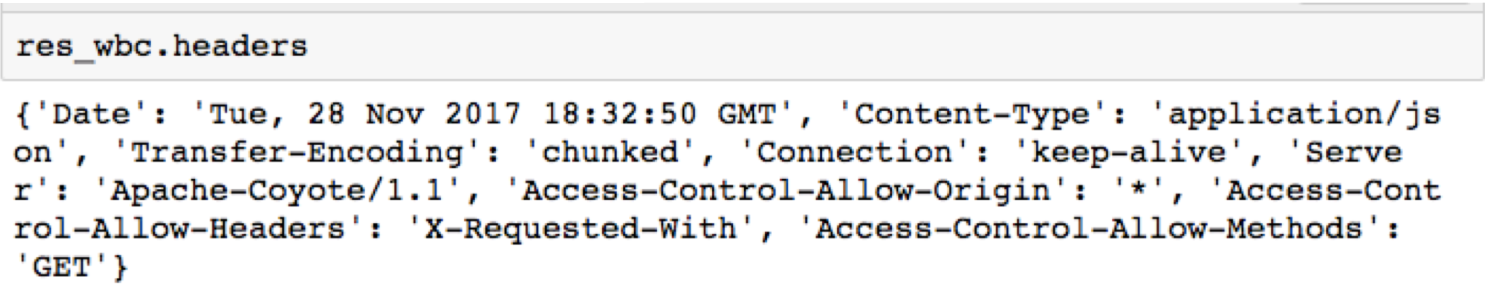
wbc_url = "http://climatedataapi.worldbank.org/climateweb/rest/v1/country/mavg/bccr_bcm2_0/pr/2020/2039/CAN"



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

```
200
```

Our request was successful

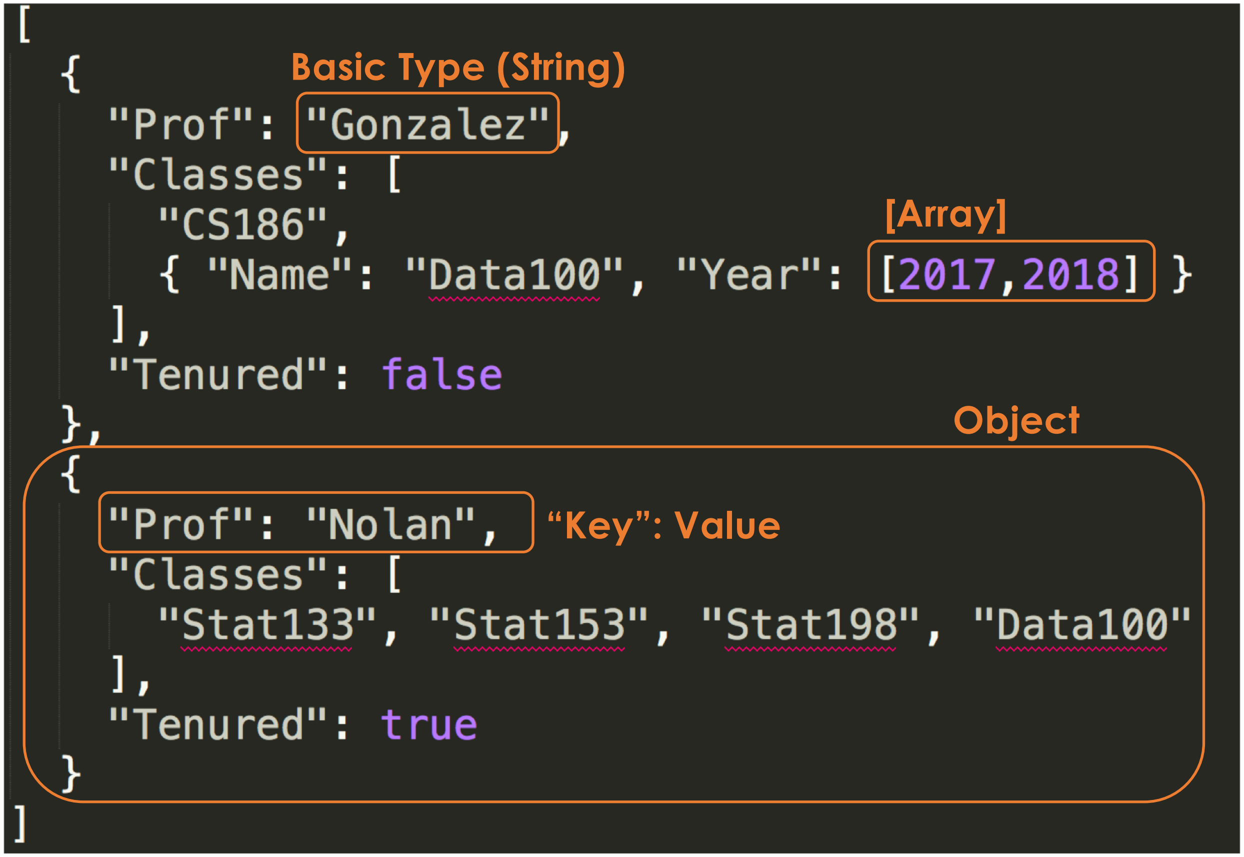


```
res_wbc.headers
```

```
{'Date': 'Tue, 28 Nov 2017 18:32:50 GMT', 'Content-Type': 'application/js
on', 'Transfer-Encoding': 'chunked', 'Connection': 'keep-alive', 'Serve
r': 'Apache-Coyote/1.1', 'Access-Control-Allow-Origin': '*', 'Access-Cont
rol-Allow-Headers': 'X-Requested-With', 'Access-Control-Allow-Methods':
'GET'}
```

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

JSON: JavaScript Object Notation



```
[
  {
    "Prof": "Gonzalez",
    "Classes": [
      "CS186",
      { "Name": "Data100", "Year": [2017,2018] }
    ],
    "Tenured": false
  },
  {
    "Prof": "Nolan",
    "Classes": [
      "Stat133", "Stat153", "Stat198", "Data100"
    ],
    "Tenured": true
  }
]
```

- 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