Python Web-Scraping

Topics

- Web basics
- Making web requests
- Inspecting web sites
- Retrieving JSON data
- Using Xpaths to retrieve `html` content
- Parsing `html` content
- Cleaning and storing text from `html`

Setup

Software and Materials

Follow the Python Installation instructions and ensure that you can successfully start JupyterLab.

Class Structure

Informal - Ask questions at any time. Really!

Collaboration is encouraged - please spend a minute introducing yourself to your neighbors!

Prerequisites

This is an intermediate / advanced Python course:

- Assumes knowledge of Python, including:
  - lists
  - dictionaries
  - logical indexing
  - iteration with for-loops
- Assumes basic knowledge of web page structure
- Relatively fast-paced

If you need an introduction to Python or a refresher, we recommend our Python Introduction.

Goals

This workshop is organized into two main parts:

1. Retrive information in JSON format
2. Parse HTML files

Note that this workshop will not teach you everything you need to know in order to retrieve data from any web service you might wish to scrape.

Web scraping background

What is web scraping?

Web scraping is the activity of automating retrieval of information from a web service designed for human interaction.

Is web scraping legal? Is it ethical?

It depends. If you have legal questions seek legal counsel. You can mitigate some ethical issues by building delays and restrictions into your web scraping program so as to avoid impacting the availability of the web service for other users or the cost of hosting the service for the service provider.

Web scraping approaches

No two websites are identical — websites are built for different purposes by different people and so have different underlying structures. Because they are heterogeneous, there is no single way to scrape a website. **The scraping approach therefore has to be tailored to each individual site.** Here are some commonly used approaches:

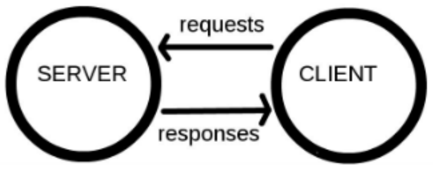
1. Use requests to extract information from structured JSON files
2. Use requests to extract information from HTML
3. Automate a browser to retrieve information from HTML

Bear in mind that even once you've decided upon the best approach for a particular site, it will be necessary to modify that approach to suit your particular use-case.

How does the web work?

Components

Computers connected to the web are called **clients** and **servers**. A simplified diagram of how they interact might look like this:



- **Clients** are the typical web user's internet-connected devices (for example, your computer connected to your Wi-Fi) and web-accessing software available on those devices (usually a web browser like Firefox or Chrome).
- **Servers** are computers that store webpages, sites, or apps. When a client device wants to access a webpage, a copy of the webpage is downloaded from the server onto the client machine to be displayed in the user's web browser.
- **HTTP** is a language for clients and servers to speak to each other.

So what happens?

When you type a web address into your browser:

1. The browser finds the address of the server that the website lives on.

2. The browser sends an **HTTP request message** to the server, asking it to send a copy of the website to the client.
3. If the server approves the client's request, the server sends the client a `200 OK` message, and then starts displaying the website in the browser.

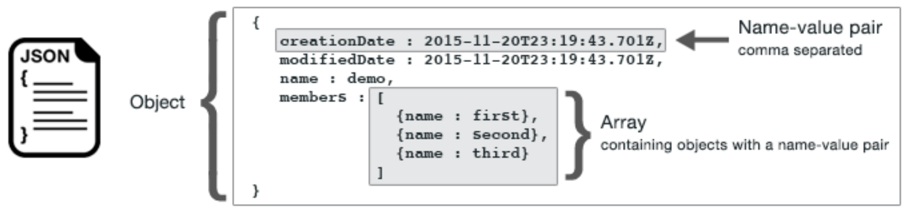
Retrieve data in JSON format if you can

GOAL: To retrieve information in JSON format and organize it into a spreadsheet.

1. Inspect the website to check if the content is stored in JSON format
2. Make a request to the website server to retrieve the JSON file
3. Convert from JSON format into a Python dictionary
4. Extract the data from the dictionary and store in a .csv file

We wish to extract information from https://www.harvardartmuseums.org/collections. Like most modern web pages, a lot goes on behind the scenes to produce the page we see in our browser. Our goal is to pull back the curtain to see what the website does when we interact with it. Once we see how the website works we can start retrieving data from it.

If we are lucky we'll find a resource that returns the data we're looking for in a structured format like JSON.



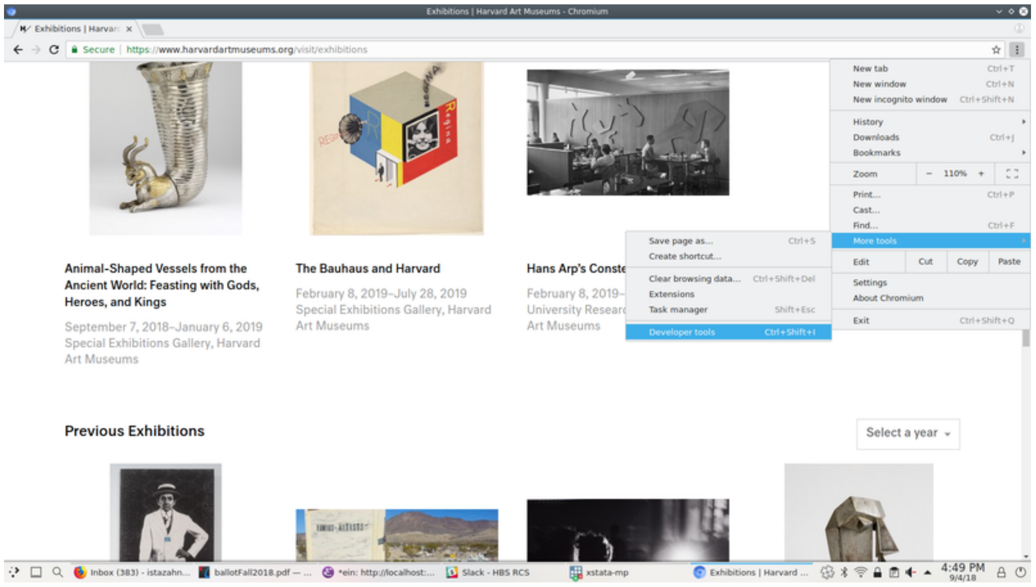
```
{
    creationDate : 2015-11-20T23:19:43.701Z,
    modifiedDate : 2015-11-20T23:19:43.701Z,
    name : demo,
    members : [
               {name : first},
               {name : second},
               {name : third}
              ]
}
```

This is useful because it is very easy to convert data from JSON into a spreadsheet type format — like a csv or Excel file.

Examine the website's structure

The basic strategy is pretty much the same for most scraping projects. We will use our web browser (Chrome or Firefox recommended) to examine the page you wish to retrieve data from, and copy/paste information from your web browser into your scraping program.

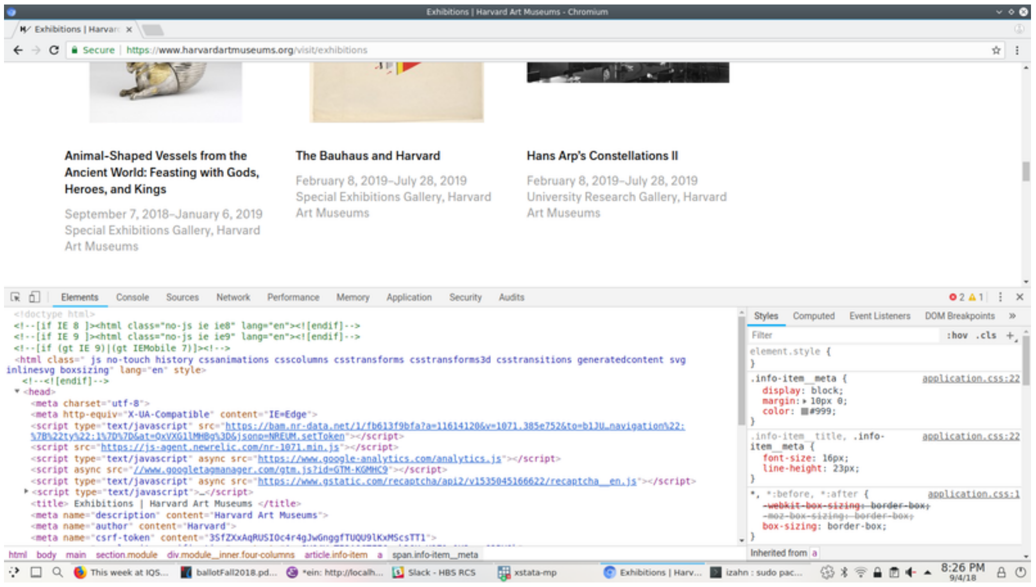
We start by opening the collections web page in a web browser and inspecting it.



Animal-Shaped Vessels from the Ancient World: Feasting with Gods, Heroes, and Kings
September 7, 2018–January 6, 2019
Special Exhibitions Gallery, Harvard Art Museums

The Bauhaus and Harvard
February 8, 2019–July 28, 2019
Special Exhibitions Gallery, Harvard Art Museums

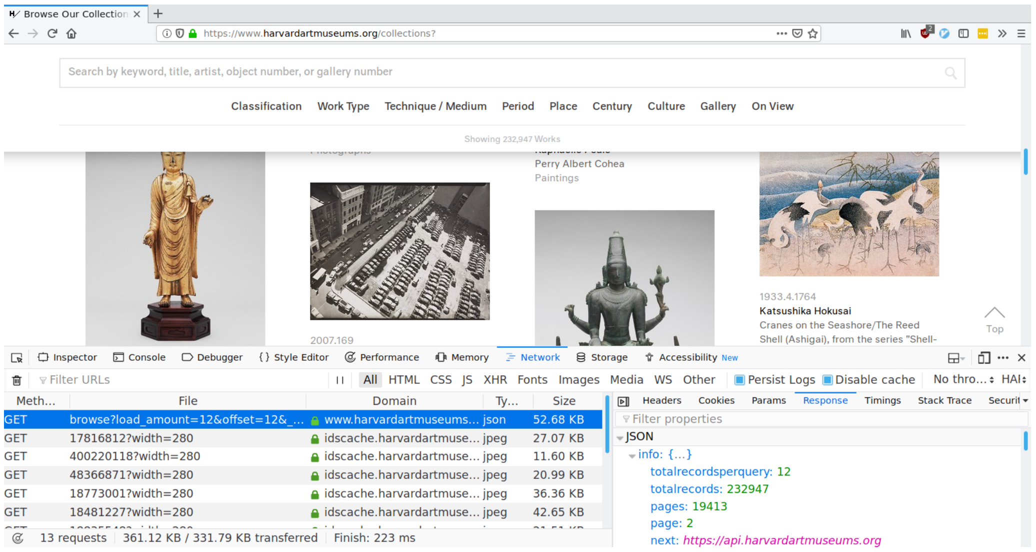
Hans Arp's Constellations II
February 8, 2019–July 28, 2019
University Research Gallery, Harvard Art Museums


Exhibitions | Harvard Art Museums - Chromium
https://www.harvardartmuseums.org/visit/exhibitions
Elements Console Sources Network Performance Memory Application Security Audits
Styles Computed Event Listeners DOM Breakpoints
html body main section.module div.module__inner.four-columns article.info-item a span.info-item__meta

```
<!doctype html>
<!--[if IE 8 ]><html class="no-js ie ie8" lang="en"><![endif]-->
<!--[if IE 9 ]><html class="no-js ie ie9" lang="en"><![endif]-->
<!--[if (gt IE 9)|(gt IEMobile 7)]><!-->
<html class=" js no-touch history cssanimations csscolumns csstransforms csstransforms3d csstransitions generatedcontent svg
inlinesvg boxsizing" lang="en" style>
  <!--<![endif]-->
 ▾ <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=Edge">
    <script type="text/javascript" src="https://bam.nr-data.net/1/fb613f9bfa?a=11614120&v=1071.385e752&to=b1JU.navigation%22:
    %7B%22ty%22:1%7D%7D&at=QxVXG1lMHBg%3D&jsonp=NREUM.setToken"></script>
    <script src="https://js-agent.newrelic.com/nr-1071.min.js"></script>
    <script type="text/javascript" async src="https://www.google-analytics.com/analytics.js"></script>
    <script async src="//www.googletagmanager.com/gtm.js?id=GTM-KGMHC9"></script>
    <script type="text/javascript" async src="https://www.gstatic.com/recaptcha/api2/v1535045166622/recaptcha__en.js"></script>
  ▸ <script type="text/javascript">…</script>
    <title> Exhibitions | Harvard Art Museums </title>
    <meta name="description" content="Harvard Art Museums">
    <meta name="author" content="Harvard">
    <meta name="csrf-token" content="3SfZXxAqRUSI0c4r4gJwGnggfTUQU9lKxMScsTT1">
```

```
element.style {
}
.info-item__meta {                    application.css:22
  display: block;
  margin: ▸ 10px 0;
  color: ■#999;
}
.info-item__title, .info-             application.css:22
item__meta {
  font-size: 16px;
  line-height: 23px;
}
*, *:before, *:after {                application.css:1
  ~~-webkit-box-sizing: border-box;~~
  ~~-moz-box-sizing: border-box;~~
  box-sizing: border-box;
}
```

If we scroll down to the bottom of the Collections page, we'll see a button that says "Load More". Let's see what happens when we click on that button. To do so, click on "Network" in the developer tools window, then click the "Load More Collections" button. You should see a list of requests that were made as a result of clicking that button, as shown below.


Browse Our Collection
https://www.harvardartmuseums.org/collections?
Search by keyword, title, artist, object number, or gallery number
Classification Work Type Technique / Medium Period Place Century Culture Gallery On View
Showing 232,947 Works
Perry Albert Cohea
Paintings
2007.169
1933.4.1764
Katsushika Hokusai
Cranes on the Seashore/The Reed Shell (Ashigai), from the series "Shell-
Top
Inspector Console Debugger Style Editor Performance Memory Network Storage Accessibility New
Filter URLs
All HTML CSS JS XHR Fonts Images Media WS Other
Persist Logs Disable cache No thro... HAI
Headers Cookies Params Response Timings Stack Trace Securit
Filter properties

Meth...	File	Domain	Ty...	Size
GET	browse?load_amount=12&offset=12&_...	www.harvardartmuseums...	json	52.68 KB
GET	17816812?width=280	idscache.harvardartmuse...	jpeg	27.07 KB
GET	400220118?width=280	idscache.harvardartmuse...	jpeg	11.60 KB
GET	48366871?width=280	idscache.harvardartmuse...	jpeg	20.99 KB
GET	18773001?width=280	idscache.harvardartmuse...	jpeg	36.36 KB
GET	18481227?width=280	idscache.harvardartmuse...	jpeg	42.65 KB

13 requests | 361.12 KB / 331.79 KB transferred | Finish: 223 ms

```
▾ JSON
  ▾ info: {…}
      totalrecordsperquery: 12
      totalrecords: 232947
      pages: 19413
      page: 2
      next: https://api.harvardartmuseums.org
```

If we look at that second request, the one to a script named `browse`, we'll see that it returns all the information we need, in a convenient format called `JSON`. All we need to retrieve collection data is to make `GET` requests to https://www.harvardartmuseums.org/browse with the correct parameters.

Launch JupyterLab

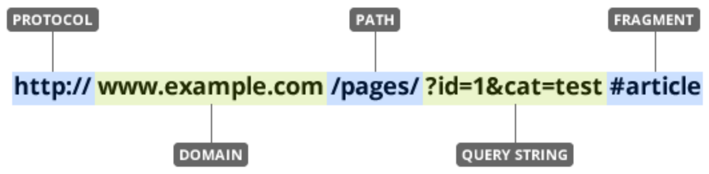
1. Start the `Anaconda Navigator` program
2. Click the `Launch` button under `Jupyter Lab`
3. A browser window will open with your computer's files listed on the left hand side of the page. Navigate to the folder called `PythonWebScrape` that you downloaded to your desktop and double-click on the folder
4. Within the `PythonWebScrape` folder, double-click on the file with the word "BLANK" in the name (`PythonWebScrape_BLANK.ipynb`). A pop-up window will ask you to `Select Kernal` — you should select the Python 3 kernal. The Jupyter Notebook should now open on the right hand side of the page

A Jupyter Notebook contains one or more *cells* containing notes or code. To insert a new cell click the `+` button in the upper left. To execute a cell, select it and press `Control+Enter` or click the `Run` button at the top.

Making requests

To retrieve information from the website (i.e., make a request), we need to know the location of the information we want to collect. The Uniform Resource Locator (URL) — commonly know as a "web address", specifies the location of a resource (such as a web page) on the internet.

A URL is usually composed of 5 parts:



The 4th part, the "query string", contains one or more **parameters**. The 5th part, the "fragment", is an internal page reference and may not be present.

For example, the URL we want to retrieve data from has the following structure:

```
protocol                    domain    path  parameters
   https www.harvardartmuseums.org  browse  load_amount=10&offset=0
```

It is often convenient to create variables containing the domain(s) and path(s) you'll be working with, as this allows you to swap out paths and parameters as needed. Note that the path is separated from the domain with `/` and the parameters are separated from the path with `?`. If there are multiple parameters they are separated from each other with a `&`.

For example, we can define the domain and path of the collections URL as follows:

```
museum_domain = 'https://www.harvardartmuseums.org'
collection_path = 'browse'

collection_url = (museum_domain
                  + "/"
                  + collection_path)

print(collection_url)
```

```
## 'https://www.harvardartmuseums.org/browse'
```

Note that we omit the parameters here because it is usually easier to pass them as a `dict` when using the `requests` library in Python. This will become clearer shortly.

Now that we've constructed the URL we wish to interact with, we're ready to make our first request in Python.

```
import requests

collections1 = requests.get(
    collection_url,
    params = {'load_amount': 10,
                  'offset': 0}
)
```

Note that the parameters `load_amount` and `offset` are essentially another way of setting page numbers — they refer to the amount of information retrieved at one time and the starting position, respectively.

Parsing JSON data

We already know from inspecting network traffic in our web browser that this URL returns JSON, but we can use Python to verify this assumption.

```
collections1.headers['Content-Type']
```

```
## 'application/json'
```

Since JSON is a structured data format, parsing it into Python data structures is easy. In fact, there's a method for that!

```
collections1 = collections1.json()
# print(collections1)
```

That's it. Really, we are done here. Everyone go home!

OK not really, there is still more we can learn. But you have to admit that was pretty easy. If you can identify a service that returns the data you want in structured from, web scraping becomes a pretty trivial enterprise. We'll discuss several other scenarios and topics, but for some web scraping tasks this is really all you need to know.

Organizing & saving the data

The records we retrieved from https://www.harvardartmuseums.org/browse are arranged as a list of dictionaries. We can easily select the fields of arrange these data into a pandas `DataFrame` to facilitate subsequent analysis.

```
import pandas as pd


records1 = pd.DataFrame.from_records(collections1['records'])


print(records1)


##                         copyright  contextualtextcount  ...                        colors
## 0                            None                    0  ...                           NaN
## 1                            None                    0  ...  [{'color': '#af7d4b', 'spe...  [{'r
## 2  © William Kentridge. Court...                    0  ...  [{'color': '#fae1e1', 'spe...  [{'r
## 3  © Jess - The Jess Collins ...                    0  ...  [{'color': '#967d64', 'spe...  [{'r
## 4  © Artists Rights Society (...                    0  ...  [{'color': '#e1e1e1', 'spe...  [{'r
## 5              © Richard Misrach                    0  ...                           NaN  [{'r
## 6                            None                    0  ...  [{'color': '#4b3232', 'spe...  [{'r
## 7                            None                    0  ...  [{'color': '#c8c8af', 'spe...  [{'r
## 8                            None                    0  ...  [{'color': '#fae1c8', 'spe...  [{'r
## 9                            None                    0  ...  [{'color': '#af7d4b', 'spe...  [{'r
##
## [10 rows x 62 columns]
```

and write the data to a file.

```
records1.to_csv("records1.csv")
```

Iterating to retrieve all the data

Of course we don't want just the first page of collections. How can we retrieve all of them?

Now that we know the web service works, and how to make requests in Python, we can iterate in the usual way.

```
records = []
for offset in range(0, 50, 10):
    param_values = {'load_amount': 10, 'offset': offset}
    current_request = requests.get(collection_url, params = param_values)
    records.extend(current_request.json()['records'])


## convert list of dicts to a `DataFrame`
records_final = pd.DataFrame.from_records(records)


# write the data to a file.
records_final.to_csv("records_final.csv")
```

```
print(records_final)
```

```
##                         copyright  contextualtextcount  ...                       people
## 0                            None                    0  ...                            NaN
## 1                            None                    0  ...  [{'role': 'Artist', 'birth...
## 2   © William Kentridge. Court...                    0  ...  [{'role': 'Artist', 'birth...
## 3   © Jess - The Jess Collins ...                    0  ...  [{'role': 'Artist', 'birth...
## 4   © Artists Rights Society (...                    0  ...  [{'role': 'Artist', 'birth...
## 5               © Richard Misrach                    0  ...  [{'role': 'Artist', 'birth...
## 6                            None                    0  ...  [{'role': 'Artist', 'birth...
## 7                            None                    0  ...  [{'role': 'Artist', 'birth...
## 8                            None                    0  ...  [{'role': 'Artist', 'birth...
## 9                            None                    0  ...  [{'role': 'Artist', 'birth...
## 10                           None                    0  ...                            NaN
## 11                           None                    0  ...  [{'role': 'Artist', 'birth...
## 12                           None                    0  ...  [{'role': 'Artist', 'birth...
## 13                           None                    0  ...  [{'role': 'Artist', 'birth...
## 14                           None                    0  ...  [{'role': 'Artist', 'birth...
## 15                © Cai Guo-Qiang                    0  ...  [{'role': 'Artist', 'birth...
## 16                           None                    0  ...  [{'role': 'Artist', 'birth...
## 17                           None                    0  ...  [{'role': 'Artist', 'birth...
## 18                           None                    0  ...  [{'role': 'Artist', 'birth...
## 19                           None                    0  ...  [{'role': 'Artist', 'birth...
## 20                           None                    0  ...  [{'role': 'Artist', 'birth...
## 21                           None                    1  ...                            NaN  {'t
## 22                           None                    0  ...  [{'role': 'Artist', 'birth...
## 23                           None                    0  ...  [{'role': 'Artist', 'birth...
## 24                           None                    0  ...                            NaN
## 25          © 2020 Allan McCollum                    0  ...  [{'role': 'Artist', 'birth...
## 26                           None                    0  ...                            NaN  {'c
## 27                           None                    0  ...  [{'role': 'Artist', 'birth...
## 28                           None                    0  ...  [{'role': 'Artist', 'birth...
## 29                           None                    1  ...                            NaN  {'t
## 30  Art © Nancy Graves Foundat...                    0  ...  [{'role': 'Artist', 'birth...
## 31  © Jenny Holzer / Artists R...                    0  ...  [{'role': 'Artist', 'birth...
## 32  © Estate of Alfred Stiegli...                    0  ...  [{'role': 'Artist', 'birth...
## 33                           None                    0  ...  [{'role': 'Artist', 'birth...
## 34                           None                    0  ...  [{'role': 'Artist', 'birth...
## 35                           None                    0  ...  [{'role': 'Artist', 'birth...
## 36                           None                    0  ...  [{'role': 'Artist', 'birth...
## 37                           None                    0  ...  [{'role': 'Artist', 'birth...
## 38                           None                    0  ...  [{'role': 'Artist', 'birth...
## 39               © Georg Baselitz                    0  ...  [{'role': 'Artist', 'birth...
## 40                           None                    1  ...                            NaN  {'t
## 41                           None                    0  ...  [{'role': 'Artist', 'birth...
## 42  © Sunil Gupta, courtesy se...                    0  ...  [{'role': 'Artist', 'birth...
## 43   © Vik Muniz / Artist Righ...                    0  ...  [{'role': 'Artist', 'birth...
## 44  © Estate of Margaret Bourk...                    0  ...  [{'role': 'Artist', 'birth...
## 45                           None                    0  ...                            NaN
## 46                           None                    0  ...  [{'role': 'Artist', 'birth...
## 47                           None                    0  ...  [{'role': 'Artist', 'birth...
## 48                           None                    0  ...                            NaN
## 49                           None                    0  ...  [{'role': 'Artist', 'birth...
```

```
##
## [50 rows x 63 columns]
```

Exercise 0

Retrieve exhibits data

In this exercise you will retrieve information about the art exhibitions at Harvard Art Museums from https://www.harvardartmuseums.org/exhibitions

1. Using a web browser (Firefox or Chrome recommended) inspect the page at https://www.harvardartmuseums.org/exhibitions. Examine the network traffic as you interact with the page. Try to find where the data displayed on that page comes from.

```
##
```

2. Make a `get` request in Python to retrieve the data from the URL identified in step1.

```
##
```

3. Write a *loop* or *list comprehension* in Python to retrieve data for the first 5 pages of exhibitions data.

```
##
```

4. Bonus (optional): Convert the data you retrieved into a pandas `DataFrame` and save it to a `.csv` file.

```
##
```

Click for Exercise 0 Solution

Parsing HTML if you have to

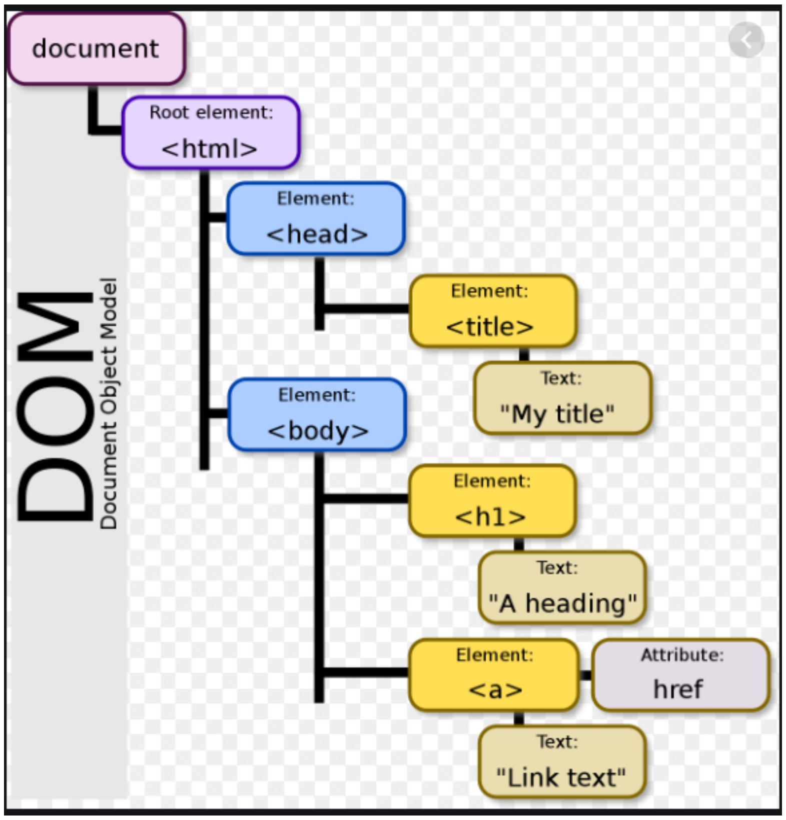
GOAL: To retrieve information in HTML format and organize it into a spreadsheet.

1. Make a request to the website server to retrieve the HTML
2. Inspect the HTML to determine the XPATHs that point to the data we want
3. Extract the information from the location the XPATHs point to and store in a dictionary
4. Convert from a dictionary to a .csv file

As we've seen, you can often inspect network traffic or other sources to locate the source of the data you are interested in and the API used to retrieve it. You should always start by looking for these shortcuts and using them where possible. If you are really lucky, you'll find a shortcut that returns the data as JSON. If you are not quite so lucky, you will have to parse HTML to retrieve the information you need.

Document Object Model (DOM)

To parse HTML, we need to have a nice tree structure that contains the whole HTML file through which we can locate the information. This tree-like structure is the **Document Object Model (DOM)**. DOM is a cross-platform and language-independent interface that treats an XML or HTML document as a tree structure wherein each node is an object representing a part of the document. The DOM represents a document with a logical tree. *Each branch of the tree ends in a node, and each node contains objects*. DOM methods allow programmatic access to the tree; with them one can change the structure, style or content of a document. The following is an example of DOM hierarchy in an HTML document:



Retrieving HTML

When I inspect the network traffic while interacting with https://www.harvardartmuseums.org/calendar I don't see any requests that return JSON data. The best we can do appears to be to return HTML.

To retrieve data on the events listed in the calender, the first step is the same as before: we make a `get` request.

```
calendar_path = 'calendar'

calendar_url = (museum_domain # recall that we defined museum_domain earlier
                + "/"
                + calendar_path)

print(calendar_url)
```

```
## 'https://www.harvardartmuseums.org/calendar'
```

```
events = requests.get(calendar_url)
```

As before, we can check the headers to see what type of content we received in response to our request.

```
events.headers['Content-Type']
```

```
## 'text/html; charset=UTF-8'
```

Parsing HTML using the lxml library

Like JSON, HTML is structured; unlike JSON, it is designed to be rendered into a human-readable page rather than simply to store and exchange data in a computer-readable format. Consequently, parsing HTML and extracting information from it is somewhat more difficult than parsing JSON.

While JSON parsing is built into the Python `requests` library, parsing HTML requires a separate library. I recommend using the HTML parser from the `lxml` library; others prefer an alternative called `beautifulsoup4`.

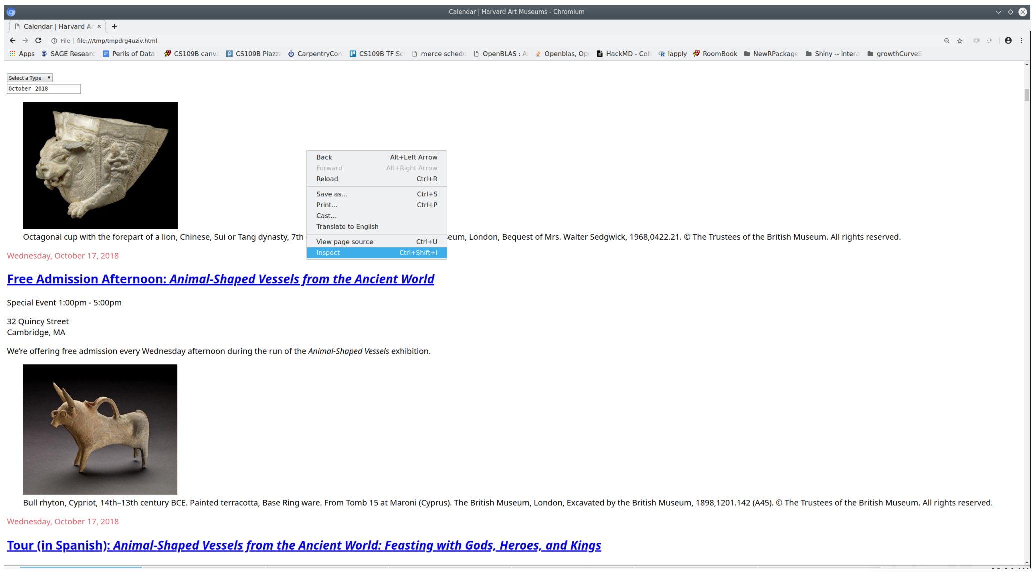
```
from lxml import html

# convert a html text representation (`events.text`) into
# a tree-structure (DOM) html representation (`events_html`)
events_html = html.fromstring(events.text)
```

Using XPath to extract content from HTML

`XPath` is a tool for identifying particular elements within a HTML document. The developer tools built into modern web browsers make it easy to generate `XPath` s that can be used to identify the elements of a web page that we wish to extract.

We can open the HTML document we retrieved and inspect it using our web browser.

```
html.open_in_browser(events_html, encoding = 'UTF-8')
```


Calendar | Harvard Art Museums - Chromium
Calendar | Harvard Ar ×
File | file:///tmp/tmpdrg4uziv.html
Apps SAGE Researc Perils of Data CS109B canva CS109B Piazza CarpentryCon CS109B TF Sc merce schedu OpenBLAS : A Openblas, Op HackMD - Coll lapply RoomBook NewRPackage Shiny -- intera growthCurve
Select a Type
October 2018
Back Alt+Left Arrow
Forward Alt+Right Arrow
Reload Ctrl+R
Save as... Ctrl+S
Print... Ctrl+P
Cast...
Translate to English
View page source Ctrl+U
Inspect Ctrl+Shift+I

Octagonal cup with the forepart of a lion, Chinese, Sui or Tang dynasty, 7th eum, London, Bequest of Mrs. Walter Sedgwick, 1968,0422.21. © The Trustees of the British Museum. All rights reserved.

Wednesday, October 17, 2018

Free Admission Afternoon: *Animal-Shaped Vessels from the Ancient World*

Special Event 1:00pm - 5:00pm

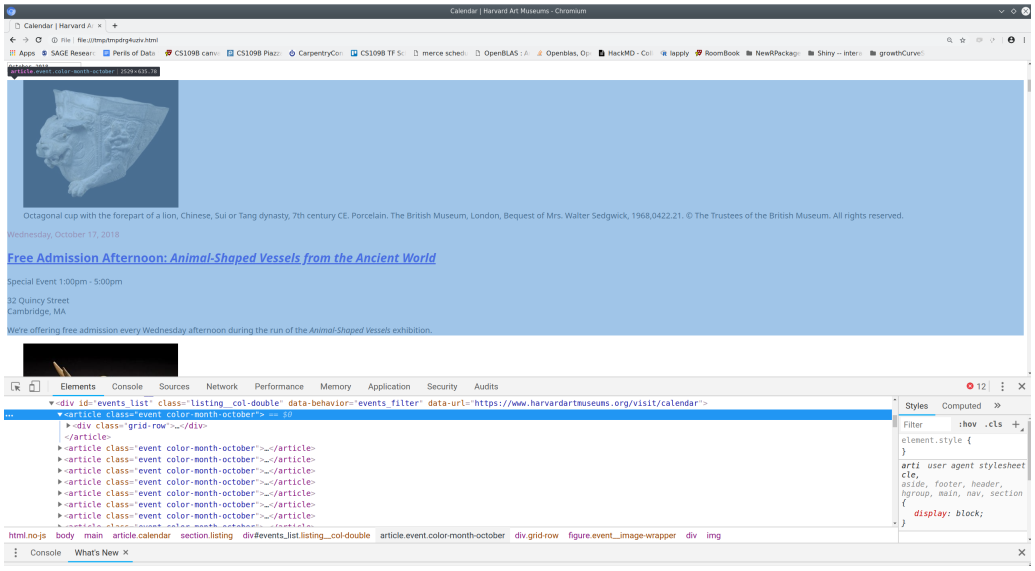
32 Quincy Street
Cambridge, MA

We're offering free admission every Wednesday afternoon during the run of the *Animal-Shaped Vessels* exhibition.

Bull rhyton, Cypriot, 14th–13th century BCE. Painted terracotta, Base Ring ware. From Tomb 15 at Maroni (Cyprus). The British Museum, London, Excavated by the British Museum, 1898,1201.142 (A45). © The Trustees of the British Museum. All rights reserved.

Wednesday, October 17, 2018

Tour (in Spanish): *Animal-Shaped Vessels from the Ancient World: Feasting with Gods, Heroes, and Kings*



Octagonal cup with the forepart of a lion, Chinese, Sui or Tang dynasty, 7th century CE. Porcelain. The British Museum, London, Bequest of Mrs. Walter Sedgwick, 1968,0422.21. © The Trustees of the British Museum. All rights reserved.

Wednesday, October 17, 2018

Free Admission Afternoon: *Animal-Shaped Vessels from the Ancient World*

Special Event 1:00pm - 5:00pm

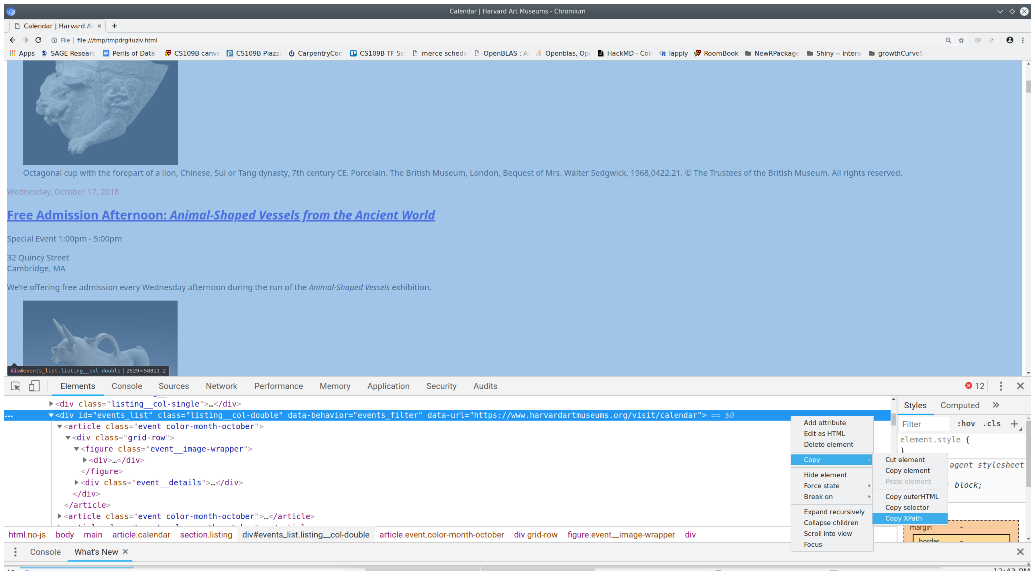
32 Quincy Street
Cambridge, MA

We're offering free admission every Wednesday afternoon during the run of the *Animal-Shaped Vessels* exhibition.

```
▼<div id="events_list" class="listing__col-double" data-behavior="events_filter" data-url="https://www.harvardartmuseums.org/visit/calendar">
  ▼<article class="event color-month-october"> == $0
    ▶<div class="grid-row">…</div>
   </article>
  ▶<article class="event color-month-october">…</article>
  ▶<article class="event color-month-october">…</article>
  ▶<article class="event color-month-october">…</article>
  ▶<article class="event color-month-october">…</article>
  ▶<article class="event color-month-october">…</article>
  ▶<article class="event color-month-october">…</article>
  ▶<article class="event color-month-october">…</article>
```

html.no-js body main article.calendar section.listing div#events_list.listing__col-double article.event.color-month-october div.grid-row figure.event__image-wrapper div img

Once we identify the element containing the information of interest we can use our web browser to copy the `XPath` that uniquely identifies that element.



Octagonal cup with the forepart of a lion, Chinese, Sui or Tang dynasty, 7th century CE. Porcelain. The British Museum, London, Bequest of Mrs. Walter Sedgwick, 1968,0422.21. © The Trustees of the British Museum. All rights reserved.

Wednesday, October 17, 2018

Free Admission Afternoon: *Animal-Shaped Vessels from the Ancient World*

Special Event 1:00pm - 5:00pm

32 Quincy Street
Cambridge, MA

We're offering free admission every Wednesday afternoon during the run of the *Animal-Shaped Vessels* exhibition.

```
▶<div class="listing__col-single">…</div>
▼<div id="events_list" class="listing__col-double" data-behavior="events_filter" data-url="https://www.harvardartmuseums.org/visit/calendar"> == $0
  ▼<article class="event color-month-october">
    ▼<div class="grid-row">
      ▼<figure class="event__image-wrapper">
        ▶<div>…</div>
       </figure>
      ▶<div class="event__details">…</div>
     </div>
   </article>
  ▶<article class="event color-month-october">…</article>
```

html.no-js body main article.calendar section.listing div#events_list.listing__col-double article.event.color-month-october div.grid-row figure.event__image-wrapper div

Next we can use Python to extract the element of interest:

```
events_list_html = events_html.xpath('//*[@id="events_list"]/article')
```

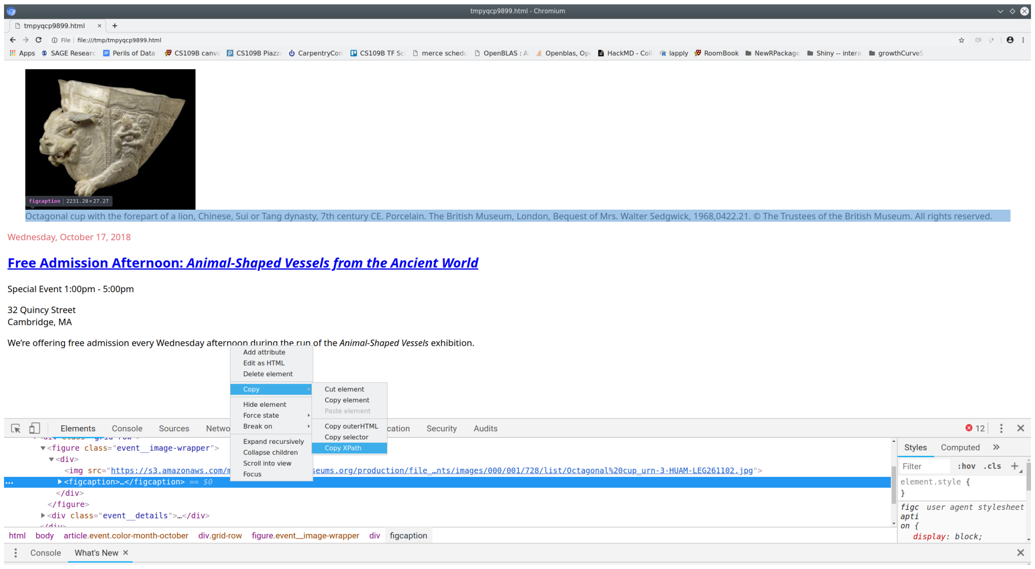
Let's just extract the second element in our events list.

```
second_event_html = events_list_html[1]
```

Once again, we can use a web browser to inspect the HTML we're currently working with - from the second event - and to figure out what we want to extract from it.

```
html.open_in_browser(second_event_html, encoding = 'UTF-8')
```

As before, we can use our browser to find the xpath of the elements we want.


Octagonal cup with the forepart of a lion, Chinese, Sui or Tang dynasty, 7th century CE. Porcelain. The British Museum, London, Bequest of Mrs. Walter Sedgwick, 1968,0422.21. © The Trustees of the British Museum. All rights reserved.

Wednesday, October 17, 2018

Free Admission Afternoon: *Animal-Shaped Vessels from the Ancient World*

Special Event 1:00pm - 5:00pm

32 Quincy Street
Cambridge, MA

We're offering free admission every Wednesday afternoon during the run of the *Animal-Shaped Vessels* exhibition.

```
▼<figure class="event__image-wrapper">
  ▼<div>
     <img src="https://s3.amazonaws.com/m…seums.org/production/file_…nts/images/000/001/728/list/Octagonal%20cup_urn-3-HUAM-LEG261102.jpg">
   ▶<figcaption>…</figcaption> == $0
   </div>
 </figure>
▶<div class="event__details">…</div>
```

```
element.style {
}
figcaption  user agent stylesheet
{
  display: block;
```

(Note that the `html.open_in_browser` function adds enclosing `html` and `body` tags in order to create a complete web page for viewing. This requires that we adjust the `xpath` accordingly.)

By repeating this process for each element we want, we can build a list of the xpaths to those elements.

```
elements_we_want = {'figcaption': 'div/figure/div/figcaption',
                    'date': 'div/div/header/time',
                    'title': 'div/div/header/h2/a',
                    'time': 'div/div/div/p[1]/time',
                    'description': 'div/div/div/p[3]'
                    }
```

Finally, we can iterate over the elements we want and extract them.

```
second_event_values = {}
for key in elements_we_want.keys():
    element = second_event_html.xpath(elements_we_want[key])[0]
    second_event_values[key] = element.text_content().strip()

print(second_event_values)
```

```
## {'date': 'Tuesday, February 9, 2021',
##  'description': 'Curator Rachel Saunders discusses the film Edo Avant-Garde '
##                 'with director Linda Hoaglund.',
##  'figcaption': 'Still from Edo Avant-Garde (2019). Master of the I’nen Seal '
##                '(1600–1630), Sōtatsu school, Trees, Japanese, Edo period, '
##                'mid-17th century. Pair of six-panel folding screens; ink, '
##                'colors, and gold on paper. Freer Gallery of Art, Washington, '
##                'D.C., Charles Lang Freer Endowment, F1962.30. Image courtesy '
##                'of Linda Hoaglund.',
##  'time': '7:00pm - 8:00pm',
##  'title': 'Edo Avant-Garde'}
```

Iterating to retrieve content from a list of HTML elements

So far we've retrieved information only for the second event. To retrieve data for all the events listed on the page we need to iterate over the events. If we are very lucky, each event will have exactly the same information structured in exactly the same way and we can simply extend the code we wrote above to iterate over the events list.

Unfortunately, not all these elements are available for every event, so we need to take care to handle the case where one or more of these elements is not available. We can do that by **defining a function** that tries to retrieve a value and returns an empty string if it fails.

If you're not familiar with Python functions, here's the basic syntax:

```
# anatomy of a function

def name_of_function(arg1, arg2, ...argn):  # define the function name and arguments
    <body of function>   # specify calculations
    return <result>      # output result of calculations
```

Here's an example of a simple function:

```
def square_fun(x):
    y = x**2 # exponentiation
    return y

square_fun(4)
```

```
## 16
```

Here's a function to perform our actual task:

```
def get_event_info(event, path):
    try:
        info = event.xpath(path)[0].text_content().strip()
    except:
        info = ''
    return info
```

Armed with this function, we can iterate over the list of events and extract the available information for each one.

```
all_event_values = {}
for key in elements_we_want.keys():
    key_values = []
    for event in events_list_html:
        key_values.append(get_event_info(event, elements_we_want[key]))
    all_event_values[key] = key_values
```

For convenience we can arrange these values in a pandas `DataFrame` and save them as .csv files, just as we did with our exhibitions data earlier.

```
all_event_values = pd.DataFrame.from_dict(all_event_values)

all_event_values.to_csv("all_event_values.csv")

print(all_event_values)
```

Exercise 1

parsing HTML

In this exercise you will retrieve information about the physical layout of the Harvard Art Museums. The web page at https://www.harvardartmuseums.org/visit/floor-plan contains this information in HTML from.

1. Using a web browser (Firefox or Chrome recommended) inspect the page at https://www.harvardartmuseums.org/visit/floor-plan. Copy the `XPath` to the element containing the list of level information. (HINT: the element if interest is a `ul`, i.e., `unordered list`.)
2. Make a `get` request in Python to retrieve the web page at https://www.harvardartmuseums.org/visit/floor-plan. Extract the content from your request object and parse it using `html.fromstring` from the `lxml` library.

```
##
```

3. Use your web browser to find the `XPath` s to the facilities housed on level one. Use Python to extract the text from those `Xpath` s.

```
##
```

4. Bonus (optional): Write a *for loop* or *list comprehension* in Python to retrieve data for all the levels.

```
##
```

Click for Exercise 1 Solution

`Scrapy` : for large / complex projects

Scraping websites using the `requests` library to make GET and POST requests, and the `lxml` library to process HTML is a good way to learn basic web scraping techniques. It is a good choice for small to medium size projects. For very large or complicated scraping tasks the `scrapy` library offers a number of conveniences, including asynchronous retrieval, session management, convenient methods for extracting and storing values, and more. More information about `scrapy` can be found at https://doc.scrapy.org.

Browser drivers: a last resort

It is sometimes necessary (or sometimes just easier) to use a web browser as an intermediary rather than communicate directly with a web service. This method of using a “browser driver” has the advantage of being able to use the javascript engine and session management features of a web browser; the main disadvantage is that it is slower and tends to be more fragile than using `requests` or `scrapy` to make requests directly from Python. For small scraping projects involving complicated sites with CAPTHAs or lots of complicated javascript using a browser driver can be a good option. More information is available at https://www.seleniumhq.org/docs/03_webdriver.jsp.

Wrap-up

Feedback

These workshops are a work in progress, please provide any feedback to: help@iq.harvard.edu

Resources

- IQSS
  - Workshops: https://www.iq.harvard.edu/data-science-services/workshop-materials
  - Data Science Services: https://www.iq.harvard.edu/data-science-services
  - Research Computing Environment: https://iqss.github.io/dss-rce/
- HBS
  - Research Computing Services workshops: https://training.rcs.hbs.org/workshops
  - Other HBS RCS resources: https://training.rcs.hbs.org/workshop-materials
  - RCS consulting email: mailto:research@hbs.edu