Prerequisites

Before working on this tutorial, you should have a [local](https://www.digitalocean.com/community/tutorial_series/how-to-install-and-set-up-a-local-programming-environment-for-python-3) or [server-based](https://www.digitalocean.com/community/tutorials/how-to-install-python-3-and-set-up-a-programming-environment-on-an-ubuntu-16-04-server) Python programming environment set up on your machine.

You should have the Requests and Beautiful Soup modules **installed**, which you can achieve by following our tutorial “[How To Work with Web Data Using Requests and Beautiful Soup with Python 3](https://www.digitalocean.com/community/tutorials/how-to-work-with-web-data-using-requests-and-beautiful-soup-with-python-3).” It would also be useful to have a working familiarity with these modules.

Additionally, since we will be working with data scraped from the web, you should be comfortable with HTML structure and tagging.

Understanding the Data

In this tutorial, we’ll be working with data from the official website of the [National Gallery of Art](https://www.nga.gov/) in the United States. The National Gallery is an art museum located on the National Mall in Washington, D.C. It holds over 120,000 pieces dated from the Renaissance to the present day done by more than 13,000 artists.

We would like to search the Index of Artists, which, at the time of updating this tutorial, is available via the [Internet Archive](https://archive.org/)’s [Wayback Machine](https://web.archive.org/) at the following URL:

[**https://web.archive.org/web/20170131230332/https://www.nga.gov/collection/an.shtm**](https://web.archive.org/web/20170131230332/https:/www.nga.gov/collection/an.shtm)

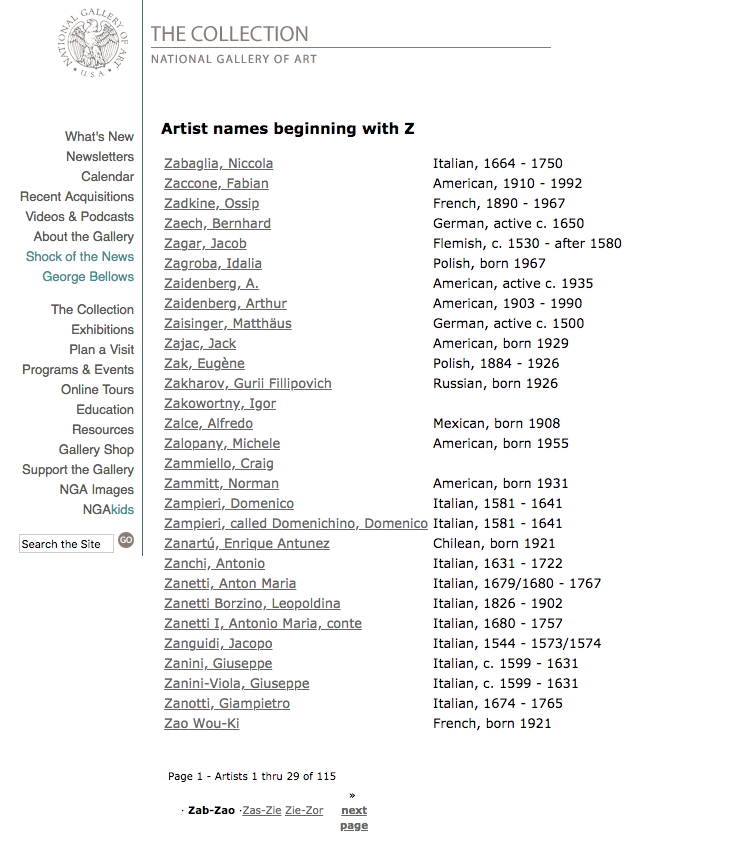
**Note**: The long URL above is due to this website having been archived by the Internet Archive.

The Internet Archive is a non-profit digital library that provides free access to internet sites and other digital media. This organization takes snapshots of websites to preserve sites’ histories, and we can currently access an older version of the National Gallery’s site that was available when this tutorial was first written. The Internet Archive is a good tool to keep in mind when doing any kind of historical data scraping, including comparing across iterations of the same site and available data.

Beneath the Internet Archive’s header, you’ll see a page that looks like this:



Since we’ll be doing this project in order to learn about web scraping with Beautiful Soup, we don’t need to pull too much data from the site, so let’s limit the scope of the artist data we are looking to scrape. Let’s therefore choose one letter — in our example we’ll choose the letter **Z** — and we’ll see a page that looks like this:



In the page above, we see that the first artist listed at the time of writing is **Zabaglia, Niccola**, which is a good thing to note for when we start pulling data. We’ll start by working with this first page, with the following URL for the letter **Z**:

[**https://web.archive.org/web/20121007172955/http://www.nga.gov/collection/anZ1.htm**](https://web.archive.org/web/20121007172955/http:/www.nga.gov/collection/anZ1.htm)

It is important to note for later how many pages total there are for the letter you are choosing to list, which you can discover by clicking through to the last page of artists. In this case, there are 4 pages total, and the last artist listed at the time of writing is **Zykmund, Václav**. The last page of **Z** artists has the following URL:

[**https://web.archive.org/web/20121010201041/http://www.nga.gov/collection/anZ4.htm**](https://web.archive.org/web/20121010201041/http:/www.nga.gov/collection/anZ4.htm)

**However**, you can also access the above page by using the same Internet Archive numeric string of the first page:

[**https://web.archive.org/web/20121007172955/http://www.nga.gov/collection/anZ4.htm**](https://web.archive.org/web/20121007172955/http:/www.nga.gov/collection/anZ4.htm)

This is important to note because we’ll be iterating through these pages later in this tutorial.

To begin to familiarize yourself with how this web page is set up, you can take a look at its [DOM](https://www.digitalocean.com/community/tutorials/introduction-to-the-dom), which will help you understand how the HTML is structured. In order to inspect the DOM, you can open your browser’s [Developer Tools](https://www.digitalocean.com/community/tutorials/how-to-use-the-javascript-developer-console#understanding-other-development-tools).

Importing the Libraries

To begin our coding project, let’s activate our Python 3 programming environment. Make sure you’re in the directory where your environment is located, and run the following command:

* . my\_env/bin/activate

With our programming environment activated, we’ll create a new file, with nano for instance. You can name your file whatever you would like, we’ll call it nga\_z\_artists.py in this tutorial.

* nano nga\_z\_artists.py

Within this file, we can begin to import the libraries we’ll be using — [Requests](http://docs.python-requests.org/en/master/) and Beautiful Soup.

The Requests library allows you to make use of HTTP within your Python programs in a human readable way, and the Beautiful Soup module is designed to get web scraping done quickly.

We will import both Requests and Beautiful Soup with the [import statement](https://www.digitalocean.com/community/tutorials/how-to-import-modules-in-python-3). For Beautiful Soup, we’ll be importing it from bs4, the package in which Beautiful Soup 4 is found.

nga\_z\_artists.py

# Import libraries

**import** requests

**from** bs4 **import** BeautifulSoup

With both the Requests and Beautiful Soup modules imported, we can move on to working to first collect a page and then parse it.

Collecting and Parsing a Web Page

The next step we will need to do is collect the URL of the first web page with Requests. We’ll assign the URL for the first page to the [variable](https://www.digitalocean.com/community/tutorials/how-to-use-variables-in-python-3) page by using the [method requests.get()](http://docs.python-requests.org/en/master/user/quickstart/#make-a-request).

nga\_z\_artists.py

**import** requests

**from** bs4 **import** BeautifulSoup

# Collect first page of artists’ list

page = requests.get('https://web.archive.org/web/20121007172955/https://www.nga.gov/collection/anZ1.htm')

**Note**: Because the URL is lengthy, the code above and throughout this tutorial will not pass [PEP 8 E501](https://www.python.org/dev/peps/pep-0008/#maximum-line-length)which flags lines longer than 79 characters. You may want to assign the URL to a variable to make the code more readable in final versions. The code in this tutorial is for demonstration purposes and will allow you to swap out shorter URLs as part of your own projects.

We’ll now create a BeautifulSoup object, or a parse tree. This object takes as its arguments the page.text document from Requests (the content of the server’s response) and then parses it from Python’s built-in [html.parser](https://docs.python.org/3/library/html.parser.html).

nga\_z\_artists.py

**import** requests

**from** bs4 **import** BeautifulSoup

page = requests.get('https://web.archive.org/web/20121007172955/https://www.nga.gov/collection/anZ1.htm')

# Create a BeautifulSoup object

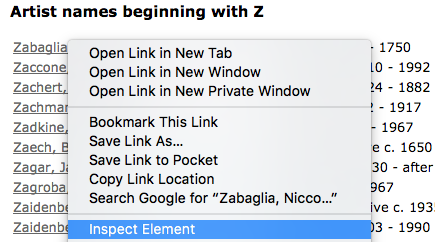
soup = BeautifulSoup(page.text, 'html.parser')

With our page collected, parsed, and set up as a BeautifulSoup object, we can move on to collecting the data that we would like.

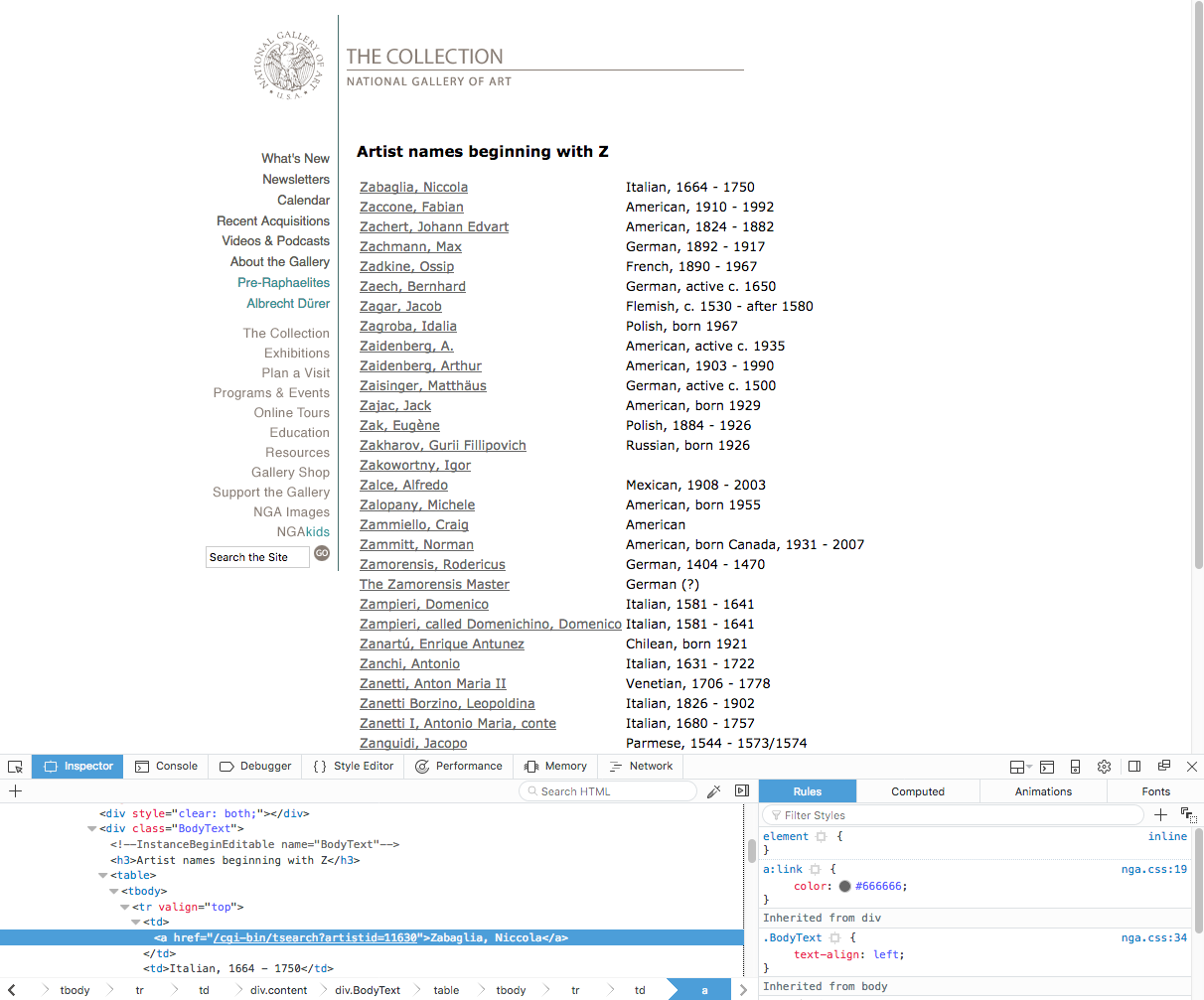
Pulling Text From a Web Page

For this project, we’ll collect artists’ names and the relevant links available on the website. You may want to collect different data, such as the artists’ nationality and dates. Whatever data you would like to collect, you need to find out how it is described by the DOM of the web page.

To do this, in your web browser, right-click — or CTRL + click on macOS — on the first artist’s name, **Zabaglia, Niccola**. Within the context menu that pops up, you should see a menu item similar to **Inspect Element** (Firefox) or **Inspect** (Chrome).



Once you click on the relevant **Inspect** menu item, the tools for web developers should appear within your browser. We want to look for the class and tags associated with the artists’ names in this list.



We’ll see first that the table of names is within <div> tags where class="BodyText". This is important to note so that we only search for text within this section of the web page. We also notice that the name **Zabaglia, Niccola** is in a link tag, since the name references a web page that describes the artist. So we will want to reference the <a> tag for links. Each artist’s name is a reference to a link.

To do this, we’ll use Beautiful Soup’s find() and find\_all() methods in order to pull the text of the artists’ names from the BodyText <div>.

nga\_z\_artists.py

**import** requests

**from** bs4 **import** BeautifulSoup

# Collect and parse first page

page = requests.get('https://web.archive.org/web/20121007172955/https://www.nga.gov/collection/anZ1.htm')

soup = BeautifulSoup(page.text, 'html.parser')

# Pull all text from the BodyText div

artist\_name\_list = soup.find(class\_='BodyText')

# Pull text from all instances of <a> tag within BodyText div

artist\_name\_list\_items = artist\_name\_list.find\_all('a')

Next, at the bottom of our program file, we will want to create a [for loop](https://www.digitalocean.com/community/tutorials/how-to-construct-for-loops-in-python-3) in order to iterate over all the artist names that we just put into the artist\_name\_list\_items variable.

We’ll print these names out with the prettify() method in order to turn the Beautiful Soup parse tree into a nicely formatted Unicode string.

nga\_z\_artists.py

...

artist\_name\_list = soup.find(class\_='BodyText')

artist\_name\_list\_items = artist\_name\_list.find\_all('a')

# Create for loop to print out all artists' names

**for** artist\_name **in** artist\_name\_list\_items:

print(artist\_name.prettify())

Let’s run the program as we have it so far:

* python nga\_z\_artists.py

Once we do so, we’ll receive the following output:

Output

<a href="/web/20121007172955/https://www.nga.gov/cgi-bin/tsearch?artistid=11630">

Zabaglia, Niccola

</a>

...

<a href="/web/20121007172955/https://www.nga.gov/cgi-bin/tsearch?artistid=3427">

Zao Wou-Ki

</a>

<a href="/web/20121007172955/https://www.nga.gov/collection/anZ2.htm">

Zas-Zie

</a>

<a href="/web/20121007172955/https://www.nga.gov/collection/anZ3.htm">

Zie-Zor

</a>

<a href="/web/20121007172955/https://www.nga.gov/collection/anZ4.htm">

<strong>

next

<br/>

page

</strong>

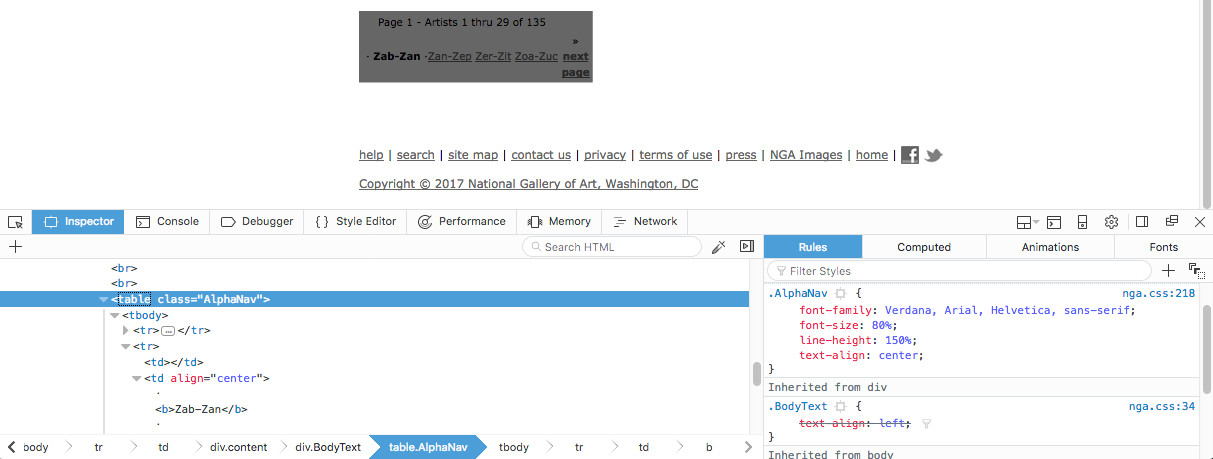
</a>

What we see in the output at this point is the full text and tags related to all of the artists’ names within the <a> tags found in the <div class="BodyText"> tag on the first page, as well as some additional link text at the bottom. Since we don’t want this extra information, let’s work on removing this in the next section.

Removing Superfluous Data

So far, we have been able to collect all the link text data within one <div> section of our web page. However, we don’t want to have the bottom links that don’t reference artists’ names, so let’s work to remove that part.

In order to remove the bottom links of the page, let’s again right-click and **Inspect** the DOM. We’ll see that the links on the bottom of the <div class="BodyText"> section are contained in an HTML table: <table class="AlphaNav">:



We can therefore use Beautiful Soup to find the AlphaNav class and use the decompose() method to remove a tag from the parse tree and then destroy it along with its contents.

We’ll use the variable last\_links to reference these bottom links and add them to the program file:

nga\_z\_artists.py

**import** requests

**from** bs4 **import** BeautifulSoup

page = requests.get('https://web.archive.org/web/20121007172955/https://www.nga.gov/collection/anZ1.htm')

soup = BeautifulSoup(page.text, 'html.parser')

# Remove bottom links

last\_links = soup.find(class\_='AlphaNav')

last\_links.decompose()

artist\_name\_list = soup.find(class\_='BodyText')

artist\_name\_list\_items = artist\_name\_list.find\_all('a')

**for** artist\_name **in** artist\_name\_list\_items:

print(artist\_name.prettify())

Now, when we run the program with the python nga\_z\_artist.py command, we’ll receive the following output:

Output

<a href="/web/20121007172955/https://www.nga.gov/cgi-bin/tsearch?artistid=11630">

Zabaglia, Niccola

</a>

<a href="/web/20121007172955/https://www.nga.gov/cgi-bin/tsearch?artistid=34202">

Zaccone, Fabian

</a>

...

<a href="/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=11631">

Zanotti, Giampietro

</a>

<a href="/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=3427">

Zao Wou-Ki

</a>

At this point, we see that the output no longer includes the links at the bottom of the web page, and now only displays the links associated with artists’ names.

Until now, we have targeted the links with the artists’ names specifically, but we have the extra tag data that we don’t really want. Let’s remove that in the next section.

Pulling the Contents from a Tag

In order to access only the actual artists’ names, we’ll want to target the contents of the <a> tags rather than print out the entire link tag.

We can do this with Beautiful Soup’s .contents, which will return the tag’s children as a Python [list data type](https://www.digitalocean.com/community/tutorials/understanding-lists-in-python-3).

Let’s revise the for loop so that instead of printing the entire link and its tag, we’ll print the list of children (i.e. the artists’ full names):

nga\_z\_artists.py

**import** requests

**from** bs4 **import** BeautifulSoup

page = requests.get('https://web.archive.org/web/20121007172955/https://www.nga.gov/collection/anZ1.htm')

soup = BeautifulSoup(page.text, 'html.parser')

last\_links = soup.find(class\_='AlphaNav')

last\_links.decompose()

artist\_name\_list = soup.find(class\_='BodyText')

artist\_name\_list\_items = artist\_name\_list.find\_all('a')

# Use .contents to pull out the <a> tag’s children

**for** artist\_name **in** artist\_name\_list\_items:

names = artist\_name.contents[0]

print(names)

Note that we are iterating over the list above by calling on the [index number](https://www.digitalocean.com/community/tutorials/understanding-lists-in-python-3#indexing-lists) of each item.

We can run the program with the python command to view the following output:

Output

Zabaglia, Niccola

Zaccone, Fabian

Zadkine, Ossip

...

Zanini-Viola, Giuseppe

Zanotti, Giampietro

Zao Wou-Ki

We have received back a list of all the artists’ names available on the first page of the letter **Z**.

However, what if we want to also capture the URLs associated with those artists? We can extract URLs found within a page’s <a> tags by using Beautiful Soup’s get('href') method.

From the output of the links above, we know that the entire URL is not being captured, so we will [concatenate](https://www.digitalocean.com/community/tutorials/an-introduction-to-working-with-strings-in-python-3#string-concatenation) the link string with the front of the URL string (in this case https://web.archive.org/).

These lines we’ll also add to the for loop:

nga\_z\_artists.py

...

**for** artist\_name **in** artist\_name\_list\_items:

names = artist\_name.contents[0]

links = 'https://web.archive.org' + artist\_name.get('href')

print(names)

print(links)

When we run the program above, we’ll receive **both** the artists’ names and the URLs to the links that tell us more about the artists:

Output

Zabaglia, Niccola

https://web.archive.org/web/20121007172955/https://www.nga.gov/cgi-bin/tsearch?artistid=11630

Zaccone, Fabian

https://web.archive.org/web/20121007172955/https://www.nga.gov/cgi-bin/tsearch?artistid=34202

...

Zanotti, Giampietro

https://web.archive.org/web/20121007172955/https://www.nga.gov/cgi-bin/tsearch?artistid=11631

Zao Wou-Ki

https://web.archive.org/web/20121007172955/https://www.nga.gov/cgi-bin/tsearch?artistid=3427

Although we are now getting information from the website, it is currently just printing to our terminal window. Let’s instead capture this data so that we can use it elsewhere by writing it to a file.

Writing the Data to a CSV File

Collecting data that only lives in a terminal window is not very useful. Comma-separated values (CSV) files allow us to store tabular data in plain text, and is a common format for spreadsheets and databases. Before beginning with this section, you should familiarize yourself with [how to handle plain text files in Python](https://www.digitalocean.com/community/tutorials/how-to-handle-plain-text-files-in-python-3).

First, we need to import Python’s built-in csv module along with the other modules at the top of the Python programming file:

**import** csv

Next, we’ll create and open a file called z-artist-names.csv for us to [write to](https://www.digitalocean.com/community/tutorials/how-to-handle-plain-text-files-in-python-3#step-4-%E2%80%94-writing-a-file) (we’ll use the variable ffor file here) by using the 'w' mode. We’ll also write the top row headings: Name and Link which we’ll pass to the writerow() method as a list:

f = csv.writer(open('z-artist-names.csv', 'w'))

f.writerow(['Name', 'Link'])

Finally, within our for loop, we’ll write each row with the artists’ names and their associated links:

f.writerow([names, links])

You can see the lines for each of these tasks in the file below:

nga\_z\_artists.py

**import** requests

**import** csv

**from** bs4 **import** BeautifulSoup

page = requests.get('https://web.archive.org/web/20121007172955/http://www.nga.gov/collection/anZ1.htm')

soup = BeautifulSoup(page.text, 'html.parser')

last\_links = soup.find(class\_='AlphaNav')

last\_links.decompose()

# Create a file to write to, add headers row

f = csv.writer(open('z-artist-names.csv', 'w'))

f.writerow(['Name', 'Link'])

artist\_name\_list = soup.find(class\_='BodyText')

artist\_name\_list\_items = artist\_name\_list.find\_all('a')

**for** artist\_name **in** artist\_name\_list\_items:

names = artist\_name.contents[0]

links = 'https://web.archive.org' + artist\_name.get('href')

# Add each artist’s name and associated link to a row

f.writerow([names, links])

When you run the program now with the python command, no output will be returned to your terminal window. Instead, a file will be created in the directory you are working in called z-artist-names.csv.

Depending on what you use to open it, it may look something like this:

z-artist-names.csv

Name,Link

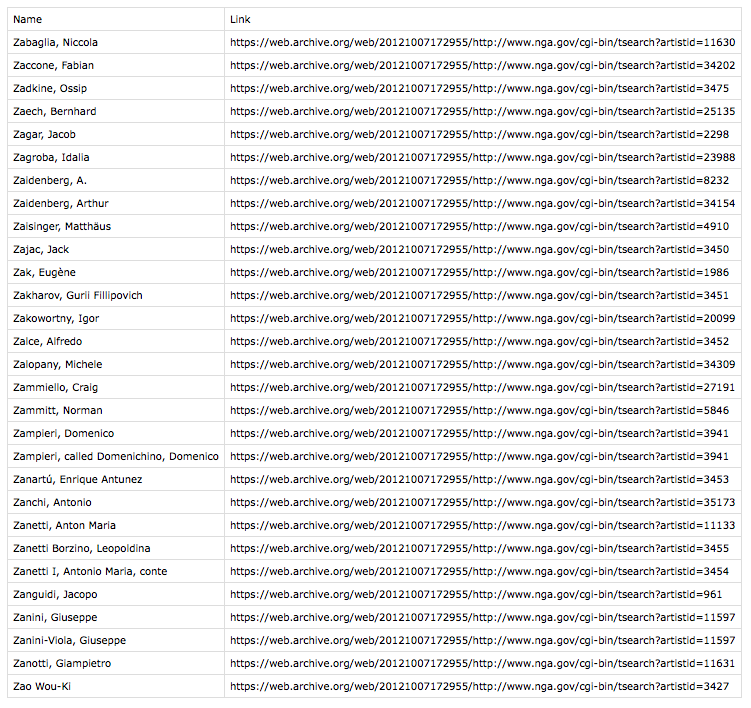
"Zabaglia, Niccola",https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=11630

"Zaccone, Fabian",https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=34202

"Zadkine, Ossip",https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=3475w

...

Or, it may look more like a spreadsheet:



In either case, you can now use this file to work with the data in more meaningful ways since the information you have collected is now stored in your computer’s memory.

Retrieving Related Pages

We have created a program that will pull data from the first page of the list of artists whose last names start with the letter **Z**. However, there are 4 pages in total of these artists available on the website.

In order to collect all of these pages, we can perform more iterations with for loops. This will revise most of the code we have written so far, but will employ similar concepts.

To start, we’ll want to initialize a list to hold the pages:

pages = []

We will populate this initialized list with the following for loop:

**for** i **in** range(1, 5):

url = 'https://web.archive.org/web/20121007172955/https://www.nga.gov/collection/anZ' + str(i) + '.htm'

pages.append(url)

[Earlier in this tutorial](https://www.digitalocean.com/community/tutorials/how-to-scrape-web-pages-with-beautiful-soup-and-python-3#understanding-the-data), we noted that we should pay attention to the total number of pages there are that contain artists’ names starting with the letter **Z** (or whatever letter we’re using). Since there are 4 pages for the letter **Z**, we constructed the for loop above with a range of 1 to 5 so that it will iterate through each of the 4 pages.

For this specific web site, the URLs begin with the string https://web.archive.org/web/20121007172955/https://www.nga.gov/collection/anZ and then are followed with a number for the page (which will be the integer i from the for loop that we [convert to a string](https://www.digitalocean.com/community/tutorials/how-to-convert-data-types-in-python-3)) and end with .htm. We will concatenate these strings together and then append the result to the pages list.

In addition to this loop, we’ll have a second loop that will go through each of the pages above. The code in this for loop will look similar to the code we have created so far, as it is doing the task we completed for the first page of the letter **Z** artists for each of the 4 pages total. Note that because we have put the original program into the second for loop, we now have the original loop as a [nested for loop](https://www.digitalocean.com/community/tutorials/how-to-construct-for-loops-in-python-3#nested-for-loops)contained in it.

The two for loops will look like this:

pages = []

**for** i **in** range(1, 5):

url = 'https://web.archive.org/web/20121007172955/https://www.nga.gov/collection/anZ' + str(i) + '.htm'

pages.append(url)

**for** item **in** pages:

page = requests.get(item)

soup = BeautifulSoup(page.text, 'html.parser')

last\_links = soup.find(class\_='AlphaNav')

last\_links.decompose()

artist\_name\_list = soup.find(class\_='BodyText')

artist\_name\_list\_items = artist\_name\_list.find\_all('a')

**for** artist\_name **in** artist\_name\_list\_items:

names = artist\_name.contents[0]

links = 'https://web.archive.org' + artist\_name.get('href')

f.writerow([names, links])

In the code above, you should see that the first for loop is iterating over the pages and the second forloop is scraping data from each of those pages and then is adding the artists’ names and links line by line through each row of each page.

These two for loops come below the import statements, the CSV file creation and writer (with the line for writing the headers of the file), and the initialization of the pages variable (assigned to a list).

Within the greater context of the programming file, the complete code looks like this:

nga\_z\_artists.py

**import** requests

**import** csv

**from** bs4 **import** BeautifulSoup

f = csv.writer(open('z-artist-names.csv', 'w'))

f.writerow(['Name', 'Link'])

pages = []

**for** i **in** range(1, 5):

url = 'https://web.archive.org/web/20121007172955/https://www.nga.gov/collection/anZ' + str(i) + '.htm'

pages.append(url)

**for** item **in** pages:

page = requests.get(item)

soup = BeautifulSoup(page.text, 'html.parser')

last\_links = soup.find(class\_='AlphaNav')

last\_links.decompose()

artist\_name\_list = soup.find(class\_='BodyText')

artist\_name\_list\_items = artist\_name\_list.find\_all('a')

**for** artist\_name **in** artist\_name\_list\_items:

names = artist\_name.contents[0]

links = 'https://web.archive.org' + artist\_name.get('href')

f.writerow([names, links])

Since this program is doing a bit of work, it will take a little while to create the CSV file. Once it is done, the output will be complete, showing the artists’ names and their associated links from **Zabaglia, Niccola** to **Zykmund, Václav**.

Being Considerate

When scraping web pages, it is important to remain considerate of the servers you are grabbing information from.

Check to see if a site has terms of service or terms of use that pertains to web scraping. Also, check to see if a site has an API that allows you to grab data before scraping it yourself.

Be sure to not continuously hit servers to gather data. Once you have collected what you need from a site, run scripts that will go over the data locally rather than burden someone else’s servers.

Additionally, it is a good idea to scrape with a header that has your name and email so that a website can identify you and follow up if they have any questions. An example of a header you can use with the Python Requests library is as follows:

**import** requests

headers = {

'User-Agent': 'Your Name, example.com',

'From': 'email@example.com'

}

url = 'https://example.com'

page = requests.get(url, headers = headers)

Using headers with identifiable information ensures that the people who go over a server’s logs can reach out to you.