Web Scraping: Data search and retrieval methodology in information science

https://doi.org/10.56238/sevened2023.008-002

Gilnei Machado
Ph.D. Professor, Professor in the Graduate Program in Information Science, State University of Londrina

ABSTRACT
Web scraping is a process of automated data collection on websites through the action of bots or programs. It is important nowadays because databases are getting bigger and bigger and, in general, there is an urgent need for information. The technique presented makes it possible to extract texts, numerical data, images, files and tables, available both on the site's home page and in its various tabs. The aim of this chapter is to present the potential of using Web scraping through Python to collect data from websites and its importance for information retrieval in Information Science. We used command lines written by Lisa Tagliaferri to check whether these command lines work and whether we were able to obtain the desired information. The site used to retrieve the desired information about artists' names and the links to their names was Web archive, which lists all the artists whose works are in the National Gallery of Art in the USA. As a result, we realized that command lines are extremely useful for obtaining information, since they allow us to obtain a large amount of information in a short time. In conclusion, we saw that scraping data from websites is perfectly feasible using Python and its code and that the information retrieved was fully satisfactory.

Keywords: Information retrieval, Data scraping, Programming in Information Science, Python.

1 INTRODUCTION

Web Scraping is a process of automated data collection on internet sites (MITCHELL, 2018) through the action of "*bots*" or programs, for subsequent processing of this data and removal of the desired information.

Data scraping can be done manually, when it comes to a small amount of data, however, in times of "*Big Data*", it is practically impossible to carry out the activity in this way, which is why, in the *Web scraping* Programming is extremely important, mainly because it makes it possible to automate the consultation of the servers where the data are stored, and is well defined as an automatic method of data extraction (BARBOSA; CAVALCANTI, 2020).

The technique, presented here, not only allows the extraction of texts or numerical data from the analyzed sites, but also helps in the extraction of information presented in the form of images, files and tables, made available both on the home page of the site and in its various tabs, that is, in all areas of the site.

The entire *web* scraping process can be carried out through Python or R, especially the steps of programming, scraping and mining the data to extract elements and behaviors relevant to the interests of the research. In Python, the *Beautiful Soup* and *Requests modules are used*.

It should be noted that it is not the purpose of this chapter to teach how to use Python, R or even to teach a technique that allows you to scrape data on any and all web pages. The objective here is to discuss the importance of the *Web Scraping* technique or method, for the extraction of data in an automated way from the internet and the transformation of this data into relevant and useful information (MOOERS, 1951). Thus, the chapter aims to present the potential of the use of *Web scraping* through Python in the collection of data present *in* web *sites* and its importance for the retrieval of information in Information Science.

2 METHODOLOGY

As this is a theoretical-conceptual-practical debate, we used analysis and bibliographic research on the subject, seeking to elucidate the doubts related to all basic conceptual elements.

On the other hand, the practical part demanded the study of the language used in Python and the ways of writing the programming for Web *Scraping* through the modules *Beautiful Soup* and *Requests*.

The Python programming language is widely used in the data science community and therefore has an ecosystem of modules and tools that can be used in various projects. In this chapter we will focus on the *Beautiful Soup* and *Requests modules*.

Beautiful Soup is a library of Python tools that allows for a quick turnaround of results in *web scraping* projects. Currently available as *Beautiful Soup 4* (BS4) and compatible with Python 2.7 and Python 3. *Beautiful Soup* creates a parsing tree from HTML and XML documents. On the other hand, the Requests library is used to make requests and requests in a database.

The scraping of data on the internet is completely ethical and legal, in case of correct use of the information collected and in case of collection on websites that do not express in writing the prohibition of this practice. The legal challenge to the procedure may arise from the misuse of information, collection of private and private data and sale to third parties of the information obtained.

3 RESULTS AND DISCUSSION

3.1 PERFORMING WEB SCRAPING WITH PYTHON

As previously highlighted, the purpose of this chapter is not to teach Python, but it is worth remembering that it was used as a basis for the development of this research and that it greatly facilitates the work of those who want or need to scrape data from the web. Therefore, it should be noted that before any step to be carried *out in web scraping , it is necessary to install and configure* *this software, in addition to importing and installing the modules or libraries* Requests *and* Beautifull Soup*, in case you decide to carry out the scraping process with the help of Python, of course.*

In addition to installing the software and importing the libraries, it is necessary to think about the command lines or the program to be used and, for this stage of the chapter, we use command lines written by Lisa Tagliaferri[1] (2023). The goal here is to check if these command lines work and if we can, in fact, get the desired information.

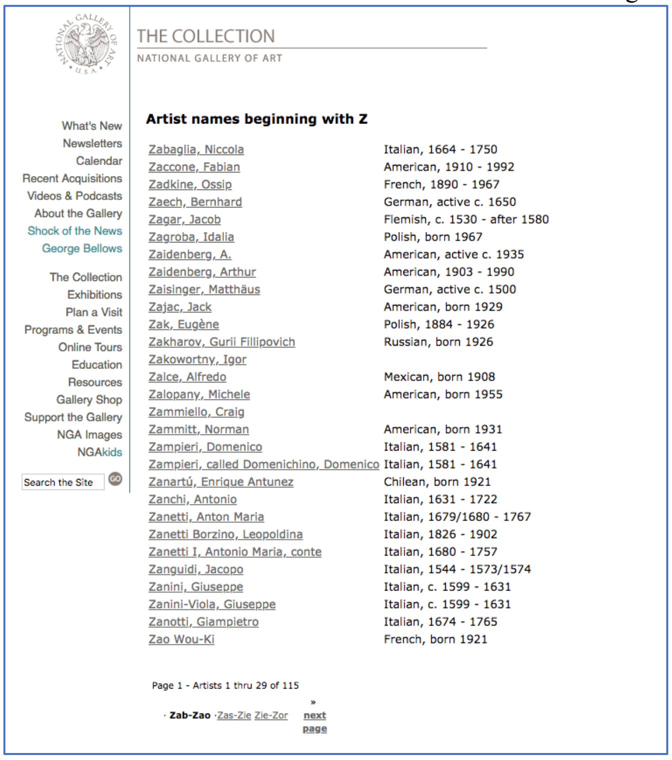
We will collect and analyze the data from a web page, in order to record the information in a text or CSV file. The page to be used is the official website of the *National Gallery of Art of*[2] the United States. The *National Gallery* is an art museum located in the city of Washington, D.C., where more than 120,000 pieces of art by more than 13,000 artists are stored, dating from the Renaissance to the present day.

Despite the large number of works and artists present in the museum, initially we will not need to extract much data from the site, as the objective here is to demonstrate the effectiveness of the tool in extracting this data and not in extracting all the data. With this, we will reduce the scope of research to just one artist and among all the letters of the alphabet, we decided to stick with the artists whose last name begins with the letter "Z".

Of the artists with a surname beginning with "Z", the one who appears in first place in the search carried out on the *Web Archive,* [3]where the works and artists of the museum are registered, is Niccola Zabaglia[4] (Figure 01), with the last artist mentioned being Václav Zykmund[5]. In all, 116 artists were found with surnames beginning with the letter "Z", distributed in four pages of results (e.g., Zab-Zao, Zas-Zie, Zie-Zor) (Figure 01).

[1] https://www.digitalocean.com/community/users/ltagliaferri, accessed November 20, 2023.
[2] https://www.nga.gov/
[3] Index available at: https://web.archive.org/ or https://archive.org/ or in https://web.archive.org/web/20170131230332/https://www.nga.gov/collection/an.shtm of the Internet Archive.
[4] https://web.archive.org/web/20121007172955/http://www.nga.gov/collection/anZ1.htm
[5] https://web.archive.org/web/20121010201041/http://www.nga.gov/collection/anZ4.htm

Figure 01: Index obtained in the search for Artist with Last Name starting with the letter Z



THE COLLECTION
NATIONAL GALLERY OF ART

What's New
Newsletters
Calendar
Recent Acquisitions
Videos & Podcasts
About the Gallery
Shock of the News
George Bellows

The Collection
Exhibitions
Plan a Visit
Programs & Events
Online Tours
Education
Resources
Gallery Shop
Support the Gallery
NGA Images
NGAkids

Search the Site GO

Artist names beginning with Z

Zabaglia, Niccola	Italian, 1664 - 1750
Zaccone, Fabian	American, 1910 - 1992
Zadkine, Ossip	French, 1890 - 1967
Zaech, Bernhard	German, active c. 1650
Zagar, Jacob	Flemish, c. 1530 - after 1580
Zagroba, Idalia	Polish, born 1967
Zaidenberg, A.	American, active c. 1935
Zaidenberg, Arthur	American, 1903 - 1990
Zaisinger, Matthäus	German, active c. 1500
Zajac, Jack	American, born 1929
Zak, Eugène	Polish, 1884 - 1926
Zakharov, Gurii Fillipovich	Russian, born 1926
Zakowortny, Igor	
Zalce, Alfredo	Mexican, born 1908
Zalopany, Michele	American, born 1955
Zammiello, Craig	
Zammitt, Norman	American, born 1931
Zampieri, Domenico	Italian, 1581 - 1641
Zampieri, called Domenichino, Domenico	Italian, 1581 - 1641
Zanartú, Enrique Antunez	Chilean, born 1921
Zanchi, Antonio	Italian, 1631 - 1722
Zanetti, Anton Maria	Italian, 1679/1680 - 1767
Zanetti Borzino, Leopoldina	Italian, 1826 - 1902
Zanetti I, Antonio Maria, conte	Italian, 1680 - 1757
Zanguidi, Jacopo	Italian, 1544 - 1573/1574
Zanini, Giuseppe	Italian, c. 1599 - 1631
Zanini-Viola, Giuseppe	Italian, c. 1599 - 1631
Zanotti, Giampietro	Italian, 1674 - 1765
Zao Wou-Ki	French, born 1921

Page 1 - Artists 1 thru 29 of 115

· Zab-Zao ·Zas-Zle Zle-Zor » next page

Source: Available at: https://web.archive.org/web/20121007172955/http://www.nga.gov/collection/anCOM1.htm, accessed November 20, 2023.

3.1.1 Step 1: Importing the libraries

Once we have decided on which site to scrape the data and most importantly, which data to scrape, it is time to proceed to what we call step 1, which involves the installation and configuration of Python and its libraries.

To install Python, just go to the https://www.python.org/downloads/ page and choose the version of interest (Windows, Linux, Mac), download it and proceed with the installation.

Python uses IDEs to insert command lines. Among the many IDEs that exist, we suggest installing Visual Studio, which is available at https://code.visualstudio.com/download.

Once Python and Visual Studio are installed, it's time to import the Python libraries, the first of which is the *Requests Library* and the second is *BeautifullSoup*. This step must be performed using command lines typed in Visual Studio (Figure 02).

A library is nothing more than a set of modules and commands in Python, organized and made available in advance so that you can import and save time when programming.

Once these two libraries are imported, we will need to identify or assign the URL of the homepage, creating a variable called *page,* using the *requests.get()* method as shown in Figure 02.

Figure 02: Importing Python Libraries.

```
import requests
from bs4 import BeautifulSoup


# Collecting the first page of artists
page = requests.get('https://web.archive.org/web/20121007172955/https://www.nga.gov/collection/anZ1.htm')
```

Source: The Author (2023).

After importing the *beautifullSoup* You must create an object or parse tree. This object must take as an argument the text of the page to be analyzed, i.e., *page.text = html.parser*. In other words, an html file is created, which is analyzed to remove the content of the response from the server (Figure 03).

Figure 03: Creating a BeautifulSoup Object

```
import requests
from bs4 import BeautifulSoup


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

# Creating a BeautifulSoup Object
soup = BeautifulSoup(page.text, 'html.parser')
```

Source: The Author (2023).

With the page of interest collected, it is possible to continue the process and proceed with the extraction of the texts of interest on the web page. It should be noted that this is only possible thanks to the fact that the page has been collected, analyzed, and configured as an object of *BeautifullSoup*.

3.1.2 Step 2: Extracting text from a web page

At this point we are interested in searching for the name of the artists and the most relevant links to the site, however, other data can also be collected, such as the nationality of the artists and the periods of history in which they lived.

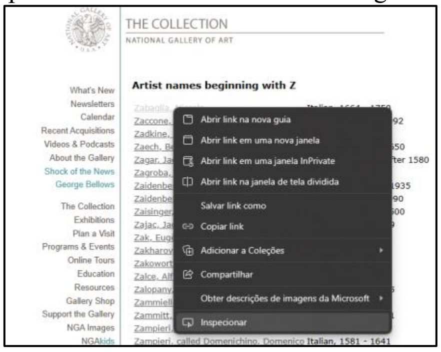
The question at this point for those who are not familiar with the subject may be: How to do this? Where do we find such information? The answer to these questions is quite simple. The information can be found in the DOM of the website or web page.

3.1.2.1 And what is the DOM?

DOM is the acronym for "Document Object Model," which is a programming interface for HTML, XML, and SVG documents that provides a structured representation of the document. This model defines the methods that allow access to the structure, style and content of the site, being formed by nodes and objects with various properties and methods, which provide the connection between web pages to scripts or programming languages.

Access to the DOM of each site is done through the browser, just having the site open and right-clicking on any area of it and choosing the "Inspect" or "Inspect Document" option. In the list of artists' names, the object of analysis in this chapter, we could click on any name to obtain the desired information, as shown in Figure 04.

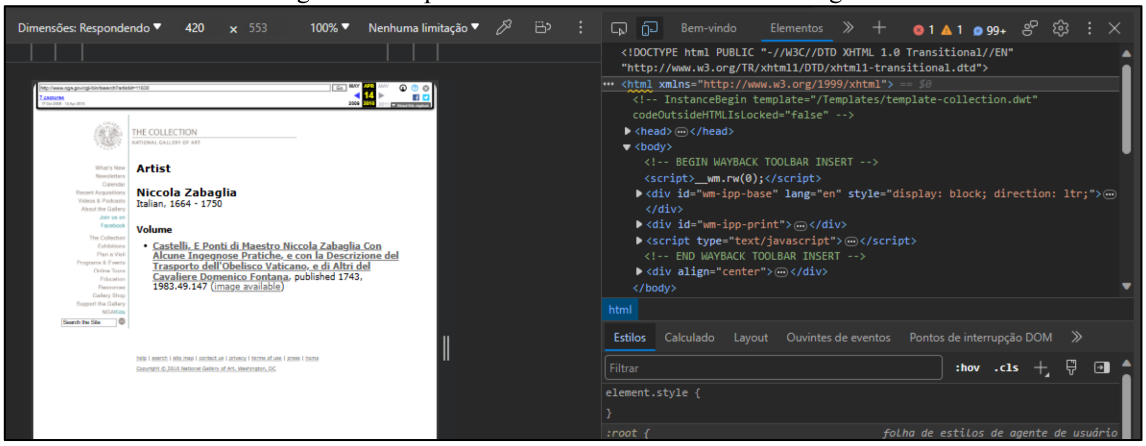
Figure 04: Representation of DOM access through the inspect item



Source: the author (2023).

After clicking on the Inspect item in the menu, the tools of interest to the developers will appear in the browser (Figure 05), which will allow access to the *Tags* linked to the names present in the list of artists and the html of the site.

Figure 05: Inspected item: artist name Niccola Zabaglia



Artist

Niccola Zabaglia
Italian, 1664 - 1750

Volume

- Castelli, E Ponti di Maestro Niccola Zabaglia Con Alcune Ingegnose Pratiche, e con la Descrizione del Trasporto dell'Obelisco Vaticano, e di Altri del Cavaliere Domenico Fontana, published 1743, 1983.49.147 (image available)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"> == $0
  <!-- InstanceBegin template="/Templates/template-collection.dwt"
  codeOutsideHTMLIsLocked="false" -->
  <head>…</head>
  <body>
    <!-- BEGIN WAYBACK TOOLBAR INSERT -->
    <script>__wm.rw(0);</script>
    <div id="wm-ipp-base" lang="en" style="display: block; direction: ltr;">…
    </div>
    <div id="wm-ipp-print">…</div>
    <script type="text/javascript">…</script>
    <!-- END WAYBACK TOOLBAR INSERT -->
    <div align="center">…</div>
  </body>
```

```
element.style {
}
:root {                                  folha de estilos de agente de usuário
```

Source: The Author (2023).

When we inspect from the name of Niccola Zabaglia it is possible to see that the table with the name of the artists present in the letter Z is inside the *<div>* of the *tags* where we have a class called *"BodyText"*. Associated with this name is also a link *tag*, as it has a page that explains who the artist is (Figure 05).

In data scraping, it is important to extract the name of the artists present in the table, described in the *file "BodyText <div>",* for this you must go back to Python and *BeautifullSoup* with the command find() *and* find_all(). The program indicated for this is shown in Figure 06.

Figure 06: Extraction of text from the Site.

```
import requests
from bs4 import BeautifulSoup


# Collect and analyze the first page
page = requests.get('https://web.archive.org/web/20121007172955/https://www.nga.gov/collection/anZ1.htm')
soup = BeautifulSoup(page.text, 'html.parser')

# Extract all text from BodyText div
artist_name_list = soup.find(class_='BodyText')

# Extract text from all instances of <a> tag from within BodyText div
artist_name_list_items = artist_name_list.find_all('a')
```

Source: The Author (2023).

Right after finding the name of the artists present in the list, we will need to perform a *"forloop"*[6] (loop) over these names or variable (artist_name_list_items). Their presentation will be performed using *prettify() (*Figure 07), which is used to transform the *BeautifulSoup parse tree* into a *formatted unicode* string.

Figure 07: Creating a for loop.

```
...
artist_name_list = soup.find(class_='BodyText')
artist_name_list_items = artist_name_list.find_all('a')

# Create loop to print the names of all artists
for artist_name in artist_name_list_items:
    print(artist_name.prettify())
```

Source: The Author (2023).

From this moment on, we can run the program in order to obtain feedback from the pages where the information is stored. This data has already been recorded in the created document (Nome_do_arquivo).

Figure 08: Execution of the File with the information of the four pages

```
1.      python nga_z_artists.py (that's the file name[7])
2.
```

Source: The Author (2023).

Once the procedure has been performed, we will receive the result shown in Figure 09.

What we see in the output at this point is the full text and *tags related*to all the artist names within the *<a>tags* found in the *<div class="BodyText">tag* of the first page, as well as some

[6] A for loop is used to iterate over a sequence (which is a list, a tuple, a dictionary, a set, or a string). This looping will be performed as many times, until the programmer's need is satisfied.
[7] nga_z_artists.py = Python file with the list of artists with surnames beginning with "Z" from the *National Gallery of Art* (nga).

additional link text at the bottom. Since we don't want this extra information, we'll work to remove it in the next section.

So far, we've been able to collect all the link text data in a *<div>* section of the web page. However, it doesn't matter at the moment that links don't reference artist names, so we're going to work to remove that part.

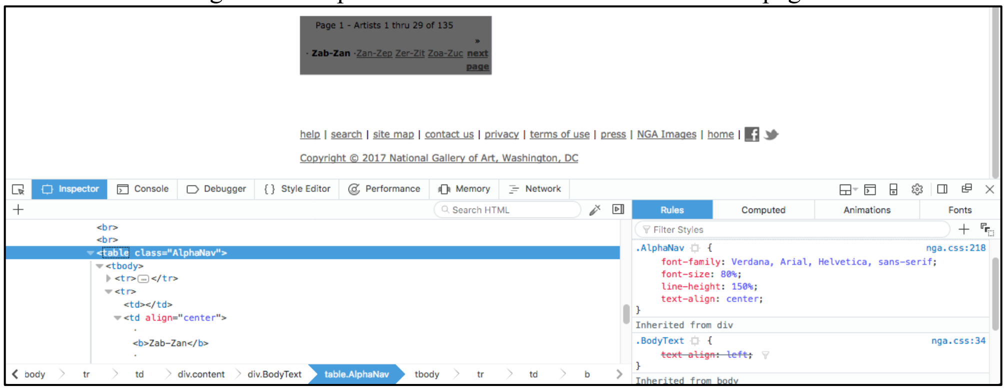
Figure 09: File Execution Result with Information

```
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>
```

Source: The Author (2023).

To remove the links from the page, let's right-click again and inspect the DOM. We'll see that the links at the bottom of the *<div class="BodyText">* section are contained in an HTML table *<table class="AlphaNav">* (Figure 10).

Figure 10: Inspection of the DOM or html of the web page.


Page 1 - Artists 1 thru 29 of 135
· Zab-Zan ·Zan-Zep Zer-Zit Zoa-Zuc » next page
help | search | site map | contact us | privacy | terms of use | press | NGA Images | home |
Copyright © 2017 National Gallery of Art, Washington, DC
Inspector
Console
Debugger
Style Editor
Performance
Memory
Network
Search HTML
Rules
Computed
Animations
Fonts
Filter Styles
body tr td div.content div.BodyText table.AlphaNav tbody tr td b

```
<br>
<br>
<table class="AlphaNav">
 <tbody>
  <tr>…</tr>
  <tr>
   <td></td>
   <td align="center">
    ·
    <b>Zab-Zan</b>
    ·
```

```
.AlphaNav {                                        nga.css:218
    font-family: Verdana, Arial, Helvetica, sans-serif;
    font-size: 80%;
    line-height: 150%;
    text-align: center;
}
Inherited from div
.BodyText {                                        nga.css:34
    ~~text-align: left;~~
}
Inherited from body
```

Source: The Author (2023).

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 delete it along with its contents. We'll use the *last_links* variable to reference these bottom links and add them to the program file (Figure 11):

Figure 11: Removing Link Buttons

```
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())
```

Source: The Author (2023).

Now, when we run the program with Python, we will receive the output exposed in Figure 12.

At this point, we see that the output no longer includes the links at the bottom of the web page and now displays only the links associated with the artists' names. So far, we've directed the links specifically to the names of the artists, but we have extra *tag data* that isn't interesting for analysis and should be eliminated. This is what will be accomplished in the next section of commands.

Figure 12: Output File After Execution

```
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>
```

Source: The Author (2023).

3.1.3 Step 03: Extracting the Content of a Tag

To access only artist names, we'll want to target the content of the <to>tags instead of printing the entire link tag. We can do this with *Beautiful Soup's .contents*, which will allow us to revise forloop so that instead of printing the entire link and its tag, we only print the list with the full names of the artists:

With the commands in Figure 13, we're *iterating* over the list of names by calling the index number of each item. We can run the program with Python to view the list of names and we will get back a list (Figure 14), with all the names of the artists available on the first page of the letter Z.

If we also want to capture the links associated with all these names, we can extract these URLs from <a>tags of a page using get('href'), which is a method of the *Beautiful Soup method* (Figure 15).

From the output of the links, we know that the entire URL is not being captured, so we will concatenate the link string with the front of the *URL* string (in this case https://web.archive.org/) (Figure 15).

Figure 13: Removal of Tag Children – artist names

```
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')

# Using .contents to remove <a> the  children from the tag to artist_name in artist_name_list_items: </a>
   names = artist_name.contents[0]
   print(names)
```

Source: The Author (2023).

Figure 14: Output: List of artists' names with surname in "Z"

```
Output
Zabaglia, Niccola
Zaccone, Fabian
Zadkine, Ossip
...
Zanini-Viola, Giuseppe
Zanotti, Giampietro
Zao Wou-Ki
```

Source: The Author (2023).

Figure 15: Using href

```
...
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)
```

Source: The Author (2023).

When we run the program in Figure 15, we will receive the names of the artists and the URLs of the links that present more information about the artists (Figure 16).

Figure 16: Output after href run: names and links

```
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
```

Source: The Author (2023).

Although we are extracting information from the website, at the moment it is only being presented in our terminal window, however this presentation format does not interest us much, although it is valuable. This chapter of this demonstration is interested in capturing and using this data elsewhere and this will only be possible by recording this information in a text file (CSV).

3.1.4 Step 04: Writing the Data to a CSV File

Collecting and visualizing data that is only presented in a Python IDE terminal window is neither very practical nor useful. On the other hand, comma-separated value (CSV) files allow you to store data in plain text tables, and are a common format for spreadsheets and databases. For this procedure, we first need to import the built-in Python *csv module* (Figure 17).

Figure 17: Importing a CSV file

```
import csv
```

Source: The Author (2023).

Right after importing the CSV, you can create and open a file called *z-artist-names.csv* (or whatever name you want to give it). In the creation of this file, the variable 'f' and the mode 'w' will be used. You should also write the titles of the top lines: *Name and Link*, which will be passed to the *writerow()* method as a list (Figure 18).

Figure 18: Creating a CSV file with Artist Name and Link.

```
f = csv.writer(open('z-artist-names.csv', 'w'))
f.writerow(['Name', 'Link'])
```

Source: The Author (2023).

Finally, inside forloop, we'll write each line with the names of the artists and the links associated with them (Figure 19).

Figure 19: Creating Lines with Artist Name and Link

```
f.writerow([names, links])
```

Source: The Author (2023).

You can see the rows for each of these tasks in Figure 20.

Figure 20: Adding each artist's name and associated link to a row

```
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 record and add header 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 the link associated with a row
    f.writerow([names, links])
```

Source: The Author (2023).

If we run the program at this point, we won't get any output as a response, instead a file will be created in the directory you're working on, called z-artist-names.csv (or whatever name you want to give it).

In the test performed for this chapter, a readable CSV file in Excel was created. It is worth reminding readers that we are performing this scraping on a site where the information of the National Gallery of Arts is registered and that this may cause the presence of non-executable links in the output of the program, as is the case of the link in the output to the artist "Zadkine, Ossip" present in Figure 21.

Figure 21: Names and links of the artists present in the output of the terminal

```
z-artist-names.csv
```

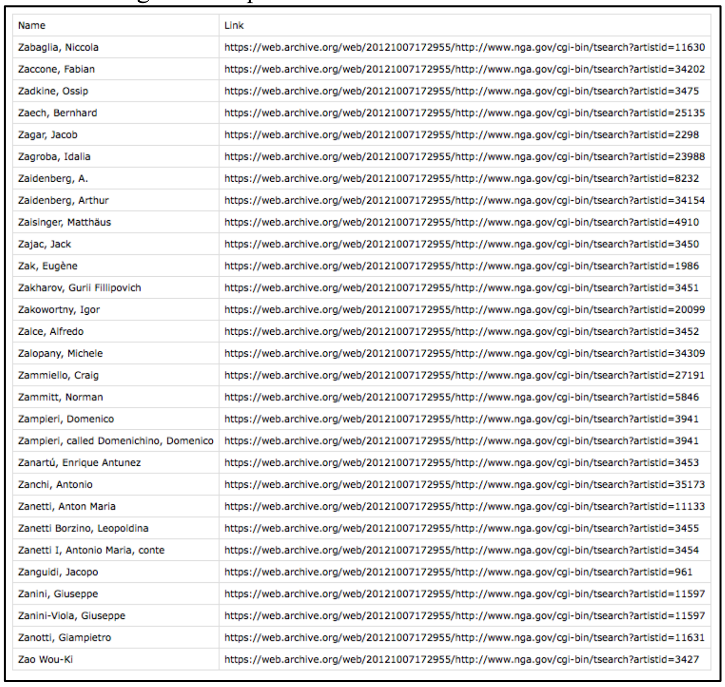
```
Name, Link
"Zabaglia, Niccola",https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=11630
"Zaccone, Fabian",https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=34202
"Hitkine, Ossip",https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=3475w
...
```

Source: The Author (2023).

In the CSV file, this information may look like a spreadsheet (Figure 22).

With the creation of the CSV file, we will be able to use the data in a more meaningful way, since the information collected is now stored on our computer.

Figure 22: Spreadsheet created from CSV file.



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=3475
Zaech, Bernhard	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=25135
Zagar, Jacob	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=2298
Zagroba, Idalia	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=23988
Zaidenberg, A.	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=8232
Zaidenberg, Arthur	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=34154
Zaisinger, Matthäus	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=4910
Zajac, Jack	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=3450
Zak, Eugène	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=1986
Zakharov, Gurii Fillipovich	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=3451
Zakowortny, Igor	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=20099
Zalce, Alfredo	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=3452
Zalopany, Michele	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=34309
Zammiello, Craig	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=27191
Zammitt, Norman	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=5846
Zampieri, Domenico	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=3941
Zampieri, called Domenichino, Domenico	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=3941
Zanartú, Enrique Antunez	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=3453
Zanchi, Antonio	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=35173
Zanetti, Anton Maria	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=11133
Zanetti Borzino, Leopoldina	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=3455
Zanetti I, Antonio Maria, conte	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=3454
Zanguidi, Jacopo	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=961
Zanini, Giuseppe	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=11597
Zanini-Viola, Giuseppe	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=11597
Zanotti, Giampietro	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=11631
Zao Wou-Ki	https://web.archive.org/web/20121007172955/http://www.nga.gov/cgi-bin/tsearch?artistid=3427

Source: The Author (2023).

3.1.5 Step 05: Retrieving Information from Related Pages

We have created a program to extract information from the first page of the list of artists whose surnames begin with the letter **Z**, however, there are 4 pages in total of these artists available on the site. To collect all these pages, we can perform further iterations with *forloops*. This will review most of the code we've written so far, but it will employ similar concepts.

To begin, we must create a list that will contain the pages (Figure 23).

Figure 23: Creation of the list containing the web pages referring to the artists

```
pages = []
```

Source: The Author (2023).

We'll populate this initialized list with a *forloop* (Figure 24).

Figure 24: Scrolling through pages 1 to 5 to make the list of 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)
```

Source: The Author (2023).

Since there are 4 pages for the letter Z, we constructed *the forloop* present in Figure 24 with a range of 1 to 5 so that it runs through each of the 4 pages. For this particular site, the URLs start with the string *https://web.archive.org/web/20121007172955/https://www.nga.gov/collection/anZ* and then are followed by a number for the page (which will be the integer i of the *forloop* that we converted to a *string*) and end with *.htm*.

We'll concatenate these *strings* and then append the result to the *pages* list. In addition to this *loop*, we must use another *loop* that will run through each of the pages. The code in this forloop *is going to look similar to the code that we've created so far in that it's performing the task that we completed for the first page of the letter Z artists, this will be accomplished for each of the four pages and since we put the original program on the second* forloop*, we now have the* original loop as a *nested for loop.* [8]

In the code (Figure 25), you should see that the first *forloop is iterating through* the pages and the second forloop *is collecting data from each of those pages and then adding the names of the artists and links line by line on each line of each page.*

These two *forloops* come below the *import* statements, the creation and writing of the CSV file (with the line to write the file headers), and the initialization of the *pages* variable (assigned to a list). The two *forloops* will look like this (Figure 25):

Figure 25: Execution of the two *forloops* traversing the 4 web pages

```
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')
```

[8] It's the loop that occurs without interruptions. Example: the months of the year. They start in January and go until December, then it starts all over again.

```
        f.writerow([names, links])
```

Source: The Author (2023).

Within the larger context of the programming file, the complete code looks like this (Figure 26):

Figure 26: Full Code

```
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])
```

Source: The Author (2023).

It may take a while to create the CSV file, since the program is long, but once created, the production will be complete, showing the names of the artists and their associated links from Zabaglia, Niccola to Zykmund, Václav.

4 FINAL THOUGHTS

The exercises carried out throughout this chapter, using the lines of code created by Lisa Tagliaferri (2023), allowed us to verify that data scraping on websites is perfectly achievable from Python and its codes.

The results obtained were fully satisfactory, since we were able to obtain, automatically, the information related to artists with surnames beginning with Z and who have works in the *National*

Gallery of Art. This procedure could be carried out for any list of artists, that is, for any letter between A and Z. It is not advisable to carry out the search for isolated artists, due to the work it takes and the ease that it would be to enter the site and type the name of this artist.

The retrieval of the names and links to these artist names allows access to other information about these artists, such as the time in which they lived, works created and place of birth.

Before thinking about carrying out the web scraping procedure, it becomes necessary to take into account the servers from which we will obtain the information. It is essential to check if the website has terms of service or terms of use related to web scraping, that is, if they allow or prohibit it in an explicit (written) way or if the website has an API that allows you to capture data, before extracting it.

It should be remembered that the constant and incessant search for data from a website will make it understand it as a robot or malware that must be blocked and certainly your IP will be blocked. That's why we shouldn't continuously access servers to collect data, we shouldn't overload someone else's servers.

A good practice for web scraping, and to avoid being blocked, is to include a header that contains your name and email so that a website can identify you and contact you if you have any questions. An example header you can use with the *Python Requests library* is shown in Figure 27:

Figure 27: Creating a Header for Personal Identification During Scraping

```
import requests

headers = {
    'User-Agent': 'Your Name, example.com',
    'From': 'email@example.com'
}

url = 'https://example.com'

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

Source: The Author (2023).

At this point, it's worth remembering that using the tools of *BeautifullSoup* and *Requests* is a good way to perform web scraping. And that saving the results in a CSV text file readable in Excel or another application allows for more in-depth analysis of the data and information collected.

REFERENCES

BARBOSA, A. B. G.; CAVALCANTI, A. B. Web Scraping e Análise de Dados. Anais do V CONAPESC, Campina Grande: Realize Editora, 10 dez. 2020. Disponível em: https://www.editorarealize.com.br/index.php/artigo/visualizar/73189. Acesso em: 13 nov. 2023.

MITCHELL, R. E. *web scraping with Python: collecting more data from the modern web*. Second Edition e. Sebastopol, AC: O'Reilly Media, 2018.

MOOERS, C. N. Zatocoding applied to mechanical organization of knowledge. American Documentation, [*s.l.*], v. 2, n. 1, p. 20-32, 1951. Disponível em: https://onlinelibrary.wiley.com/doi/abs/10.1002/asi.5090020107. Acesso em: 25 novembro de 2023.

NGA - NATIONAL GALLERY OF ARTS. Disponível em: https://www.nga.gov/, acesso em 25 de novembro de 2023.

TAGLIAFERRI, L. *How To Scrape Web Pages with Beautiful Soup and Python 3*. https://www.digitalocean.com/community/tutorials/how-to-scrape-web-pages-with-beautiful-soup-and-python-3, Acesso em 20 de novembro de 2023.