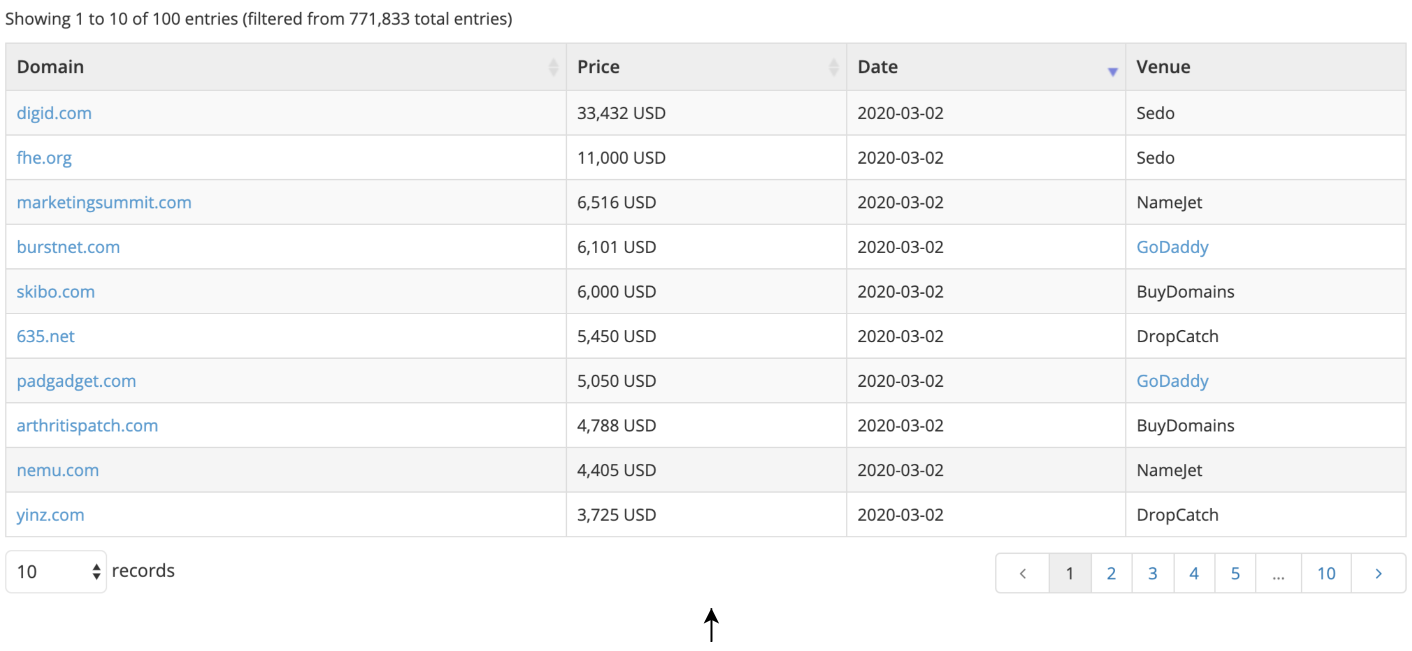
Scraping Domains

In this lesson, you will build a proper automated scraper to store domain name data in a csv file.

We'll cover the following

- Writing the scraper
- Scraper in action

The website namebio.com will be scraped in this lesson. This site contains the domain name information like which domain is sold when, for what price, and where it is parked. In free mode, we can look at *one-hundred* data points with four columns of data which include the **domain name, domain price, date it is sold, and the site where the domain is parked**. The data is displayed and available in the form of a table like in the following image.



Showing 1 to 10 of 100 entries (filtered from 771,833 total entries)

Domain	Price	Date	Venue
digid.com	33,432 USD	2020-03-02	Sedo
fhe.org	11,000 USD	2020-03-02	Sedo
marketingsummit.com	6,516 USD	2020-03-02	NameJet
burstnet.com	6,101 USD	2020-03-02	GoDaddy
skibo.com	6,000 USD	2020-03-02	BuyDomains
635.net	5,450 USD	2020-03-02	DropCatch
padgadget.com	5,050 USD	2020-03-02	GoDaddy
arthritispatch.com	4,788 USD	2020-03-02	BuyDomains
nemu.com	4,405 USD	2020-03-02	NameJet
yinz.com	3,725 USD	2020-03-02	DropCatch

10 records

Writing the scraper

As can be seen in the above image, to get to the next table of domains, the next number button needs to be clicked and this is where the real power of selenium comes into play. Each and every button on the webpage can be clicked simply by finding its `xpath` and clicking the button using the `click()` function. Let's see this in action.

in action.

```
# Import the require packages
from selenium import webdriver
import time
import csv

# Initialize the column names list
col = ['Name','Price','Date','Parked_At']

# Write the column name list in the csv file
with open('Domain.csv', 'w') as csvFile:
        writer = csv.writer(csvFile)
        writer.writerow(col)

chrome_options = webdriver.ChromeOptions()
#chrome_options.add_argument("--headless")
chrome_options.add_argument("--disable-extensions")
chrome_options.add_argument("--disable-gpu")
chrome_options.add_argument("--no-sandbox")

# Initialize the browser object
driver = webdriver.Chrome('<Provide the path of chromedriver>', options=chrome_options)

# URL to go to
url = "https://namebio.com/"
# Maximize the browser window
driver.maximize_window()
# Opens the URL in the browser
driver.get(url)

# Counter to traverse through the 10 tables
i = 0

while i < 10:

  #  Get all elements with the table row tag using the xpath method
  trs = driver.find_elements_by_xpath('//*[@id="search-results"]/tbody/tr')

  # Traverse through the rows
  for tr in trs:

    # Split each row to get four values
    tr = str(tr.text).split(' ')
    del tr[2:3]

    # Write the list of values to the csv file
    with open('Domain.csv', 'a') as csvFile:
        writer = csv.writer(csvFile)
        writer.writerow(tr)

    # Close the csv file
    csvFile.close()

  # When all rows are read then click the next arrow button to move to next table
  driver.find_element_by_xpath('//*[@id="search-results_next"]/a').click()

  # Give a 2 sec delay for the table to get loaded
  time.sleep(2)
```

```
    # Increment the counter
    i+=1


# Close the browser
driver.close()
```

The above code automates the browser to open the link and scrape all the information in all the tables by automatically clicking through the next button. Let's break down the code.

On **line 7**, the names of the columns are defined in a list to store the data in perfect order.

On **lines 10-12**, the list with column names is written to a file named `Domain.csv`. The scraped data also gets stored in this file.

On **line 26**, the `maximize_window()` function maximizes the browser window to cover the entire screen. It's always a good practice to enlarge the screen while scraping.

On **lines 31 & 33**, the loop and the loop counter is defined. This controls how many times the next button should be pressed to scrape the table. The loop counter is marked less than **ten** as we have to navigate through ten tables.

On **line 36**, the `find_elements_by_xpath` function is used to fetch all the elements that have the path given as its parameter. This function is different from the `find_element_by_xpath` function as this one only fetches a single element from its path. The purpose of grabbing multiple elements here is because the information is spread among different `tr` tags. Therefore, once all the `tr` tags are fetched, we can separate them to extract the required information. The `tr` tag is part of the `table` tag in HTML. More information on this can be obtained here.

On **line 39**, a loop is used to traverse through all the `tr` tags.

On **lines 42 & 43**, as the required data is divided into four columns, some preprocessing is done to separate the row data and obtain all four domain data points individually in the form of a list.

On **lines 46-48**, the list with the four data points is written to the `Domain.csv` file in the order of their columns. That's why the row data is converted to the list form.

On **line 51**, the `Domain.csv` file is closed for writing.

On **line 54**, the `find_element_by_xpath` function is used to fetch the element of the **next** button that triggers the new table to appear. Once this element is obtained, the `click()` function of the browser instance is applied to it; This triggers a click event and the next table appears.

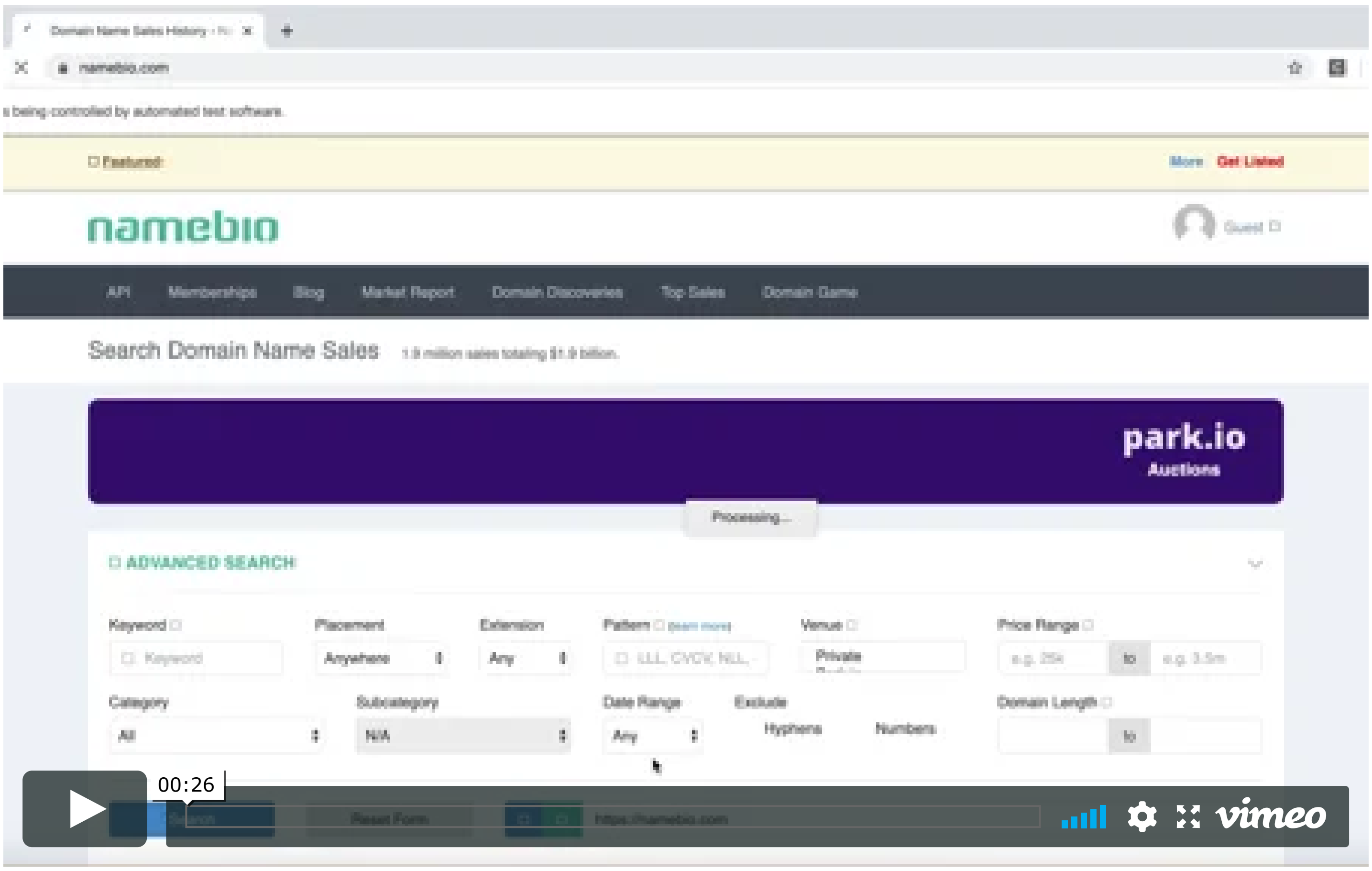
On **line 57**, the `time.sleep()` function is used to provide a **two sec** delay for the table to get loaded. It is good practice to provide a little delay between click events as the event might take some time before occurring.

On **line 60**, the loop counter is incremented.

On **line 63**, the browser is closed after the scraping is done using the `close()` function of the browser instance.

Scraper in action

The following video shows the working of the above code.



In the video, you see that a browser opens and then traverses to each new table automatically and scrapes the data. The `Domain.csv` file is available for download below. It contains all the domain name information from the site. It would have taken hours to extract that information manually, but with *selenium*, we have all the data under **thirty sec**.

Domain.csv

This marks the end of the scraping chapter. Keep exploring the selenium package to discover new ways to scrape different kinds of data.

A quiz awaits in the next lesson!