AKADEMIA
LEONA KOŹMIŃSKIEGO

Janusz Garścia
38653

Practical applications of Web Scraping on the example of e-commerce.

Bachelor's dissertation
prepared under supervision of

Warsaw 2021

European Funds Knowledge Education Development
Republic of Poland
European Union European Social Fund

Abstract

Web Scraping, dating almost back to the creation of the World Wide Web, has first seen the light of day in 1993. Scraping aims to harvest data from websites in an automated and quick way in order to gain data. The paper focuses on the technology as a tool, examines and explores the theoretical terminology, as well as practical applications in the business world. The project takes form of a problem-solving approach and plans to show the use and the process of building and using said tool, while explaining what is happening behind the scenes and phenomena connected to the technology of Web Scraping.

The paper uses the means of Python and R programming languages. Jupyter Notebook framework, Scrapy and BeautifulSoup4 libraries to achieve desired results.

Keywords

Web Scraping, World Wide Web, Scraper, Crawler, Business, Technology, E-commerce, Practical, Tool, Problem Solving, Python, R

Table of Contents

Introduction

The e-commerce market has been on a rise ever since the World Wide Web was introduced and the COVID-19 epidemic has only helped it grow further. The market grew by $60 billion in a year due to the outbreak ("U.S. E-Commerce Sales 1999-2021", 2021). Eurostat claimed that 73% of internet users have used e-commerce services to purchase items over the internet ("E-commerce statistics for individuals", 2021). There is no doubt that e-commerce is a booming business nowadays. This is important for Web Scraping as the technology is used the most in the market of online sales as it is the easiest to implement.

The paper explores the technology of scraping and how it can be used for any benefit. It examines how useful scraping is as a versatile tool on the example of gathering listing from OLX and Allegro – leading e-commerce websites in Poland. It aims to show strategies and the structure of real scraping projects as well as the building process. The project endorses a problem-solving approach due to the nature of programming. The purpose of the paper is to create something original and expose the reader to the process of solving said problems and to expose them to the thoughts behind the building process. This is an academic paper as a practical project and aims to prove a practical applicability of a self-developed program in the real world by using technical and entrepreneurial skills acquired during courses at Kozminski University.

The paper is divided into three parts. The first part explains the theoretical terminologies connected to the World Wide Web, Web Scraping and the technology behind the IT ecosystems. Explains what scrapers can be used for and under what circumstances, touches on many different basic technologies that may be crucial in understanding how Web Scraping works. The second part is the source code breakdown, which explains how I have written the programs and explains how they work and what the problems were. Readers are exposed to parts of code and a comprehensible explanation of what the part of code does and for what reason. The third part focuses on the application of scraping in business and explains how I have used the program to my benefit. Dives into what can be done to improve the program and my future plans connected to the general theme of the paper.

I. Web Scraping Theory Explained

1.1. What is web scraping?

In order to create extraordinary artificial intelligence models, that perform tasks that years ago were considered unspoken of or simply impossible, you need data – a lot of it, especially in a world where data is quickly becoming yet another currency format. Data can be used for feeding machine learning systems, analyzing data patterns, predicting the future, going a step ahead of competitors or even just the customer – naturally so, there is a higher need for this currency nowadays (Moulik, 2020, p. 1). Web scraping, web crawling, crawlers, spiders, and scrapers are all just one of many tools used to gather said data. The only difference between spiders and scrapers is that spiders are designed to continuously fetch URLs which they continue to extract data from, and web scrapers are specialized in extracting data from only a certain website (Sirisuriya, 2015, p. 135).

Web scraping itself is the action of obtaining data from any website. Usually, the process is automated, but the definition is not limited to that. You could technically visit a certain website and gather the data manually by coping and pasting the data. However, this is highly inefficient on a bigger scale, thus very rare to encounter. The process is automated in order to gather great quantities of data which later can be analyzed or used for different purposes. Typically, the data is stored in offline databases. Scrapers can be used in different spheres of the World Wide Web (or WWW, or W3) (T. Berners-Lee et al., 1944, p. 1) and provide us with different outcomes, which can be as following:

- Contact scraping – used for obtaining information about people for marketing purposes i.e.: a database consisting of names, telephone numbers and emails, which later can be used to analyze their activity, predict choices, and likes – something like personal ads (Sirisuriya, 2015, p. 136).
- Real estate scraping – gathering listing to compare, observe price changes and analyze the data in order to take informed decisions and find the best possible offer (Manuel and Santos, 2018, p. 41).
- Product scraping (reviews and prices), Market Analysis – scraping reviews for either sentiment analysis or to watch competitors and gain an edge. Scraping products can also work for monitoring certain markets and notifying of new available products.

Price scraping can also be used for competitor analysis, but also for monitoring price changes, analyzing the prices i.e.: distribution, correlations. This project heavily endorses this approach (Gode et al., 2018, p. 365).

- Other minor usage like weather scraping, researching, tracking some online entity i.e.: through Instagram posts, tweets etc., one can also store them in case they get deleted.
- Other more complex examples when it comes to scrapers as a tool in a larger scheme of things can be: data and web mining, web indexing and monitoring changes on the web and its history. This works a bit differently than the other mentioned usage and is irrelevant to the topic.

As illustrated by the examples, web scrapers can be applied in many cases. All the examples above also share a similar goal which is obtaining data. Simply, whenever and wherever one would like to obtain data from the World Wide Web and store it, use it for something – a web scraper can be used.

The complexity of writing a web scraper or web scrapers in general may vary based on how hard the information that is needed is to access, how much of that information is required, how many places (be it websites) it must access, how well a website is protected, and how well a website's source code is suited for information extraction. Fact is, websites are not written with this in mind, they are solely designed for human exploration, user readability and user experience by implementing intuitive visual cues that robots will completely miss. However, there are efforts of creating machine learning models that employ computer vision in order to 'see' the webpage as a human would, which ironically is going back to where it all started as screen scraping (obviously, a very basic form of it) was the origin of the scraping that we know today (Oostenrijk, 2004).

1.2. Basic terminology

Going more in depth, let us see how web scrapers work from more of a technical side. In order to understand how a web scraper connects to a website and obtains the data it is necessary to understand how the internet that everyone uses on the daily basis works. Put it simply, users need to somehow connect to the sites in order to view the content. World Wide Web is exactly about that communication between the web clients and the web servers. The communication between those services is made possible by Hypertext Transfer Protocol (HTTP) requests (Gettys, Ed., 1999, p. 35-41). Therefore, HTTP is the fundament of the whole information system data communication for the World Wide Web (WWW). In order to complete the first required step in web scraping which is connecting to a website we need to request the connection using said HTTP requests. HTTP works as following, first the web client i.e., a browser connects to the web by sending an HTTP request, which then is received and analyzed by the web server. After the server processes the request or returns an HTTP response to the client. The client receives the response and can view the output – e.g., a google website in a Firefox browser (Bishop, Ed., 2021, p. 21). After all the processes going on in the background an HTML page is shown to the user in the browser.

Hypertext Markup Language (HTML) (Raggett et al., 1998) is a standard markup language for the output or content to be displayed in browsers, specifically designed for human users. HTML is assisted by JavaScript (JS) and Cascading Style Sheets (CSS) ("HTML & CSS - W3C", n.d.) and the latter will be crucial for extracting elements from HTML documents in this project. HTML consists of many elements either with attributes or with meta elements containing information about the document. The elements in the source code consist of start and end <> tags with arrows. For example, in the source code this would be a *<h1>Heading</h1>* and this would be a *<p>Paragraph</p>*. Websites are built out of example elements as provided and this is what lies beneath the visual rendered representation that the end user sees on their screens in the browser (Harris, 2018, p. 10).

Web scrapers can connect to the websites using HTTP requests without running a web client i.e., browser, with exceptions that will be discussed in the next subheading. This is advantageous in a way that the scrapers cut the time needed for opening a browser and navigating – instead, they send a request and download the source code immediately and then select elements of interest, impersonating a browser by passing a header user-agent or not.

After acquiring the source code, the program is free to do whatever it was designed to with the code, as it is now stored locally (most commonly in a variable) and no other request are going to be sent. The source code needs to be parsed in order to be able to select elements from it. Parsing means converting a certain string (text) into some structured data form. Among the most popular parsing types are HTML/XML parsing and JSON parsing, a

CSV file can also be parsed. The way it works is that the parser, so the program which parses the source text into structured data, has a sort of a prescription. It knows where the data that it is looking for is, because the format of the source code is always the same. In case it is not the parses returns an error.

Now that the parsed source code is in the program’s possession it must run selectors in order to extract the selected, required data from targeted elements of interest. Among the most popular selectors are CSS, XPath (Grasso et al., 2013) and some other package built-in selectors, like BeautifulSoup4’s find selector. So as to understand the premise better, it is essential to be exposed to the actual code itself. As an example, let us take a look at this very basic block of code that I have prepared for the purpose of explaining the usage of element selectors in web scrapers.

```html
<html>
  <head>
    <title>This is a title</title>
  </head>
  <body>
    <h1>This is a header</h1>
    <p>
      <a href="https://www.google.com/">This is the link</a> that directs you to google.
    </p>
  </body>
</html>
```

Figure 1. Basic sample HTML code snippet

As it is, there are three options available to extract different parts of this html code. First let us extract the simplest – the title.

Table 1. Selectors comparison by Python input and output; grabbing title.

	Input	**Output**
CSS	`html.select('title')`	`[<title>This is a title</title>]`
Xpath	`response.xpath('//title/text()')`	`['This is a title']`
BS4	`html.find('title')`	`<title>This is a title</title>`

As illustrated above, the *CSS* way outputs the title with the tags and the brackets in the beginning and the end indicating that it is a list, in order to get rid of the brackets we could simply put a *[0]* to indicate that we want to select the first most element of the list – in this case there is only one element, but the program parses it into a list anyway. The XPath way parsed without tags but in a list, The BS4 way parsed without a list, but with tags.

HREF elements are elements that hold links, which hold useful information as in many cases scrapers need to follow links or extract links in order to store it in a database for later use. In this project we will be scraping hyperlinks in order for easier navigation for users from the database if a good offer is found. Take a look at how href elements are extracted using different methods:

Table 2. Selectors comparison by Python input and output; grabbing link.

	Input	**Output**
CSS	`html.select('a:link')`	`[<a href="https://www.google.com/">This is the link</a>]`
Xpath	`response.xpath('//p/a/@href')`	`['https://www.google.com/']`
BS4	`html.find('a')['href']`	`'https://www.google.com/'`

CSS, XPATH and the BS4 package included selector have their own benefits and downsides. In projects for certain actions, it is good to use all of the approaches and mix them together for the fastest execution and smallest, optimized amount of code. In this project all of the approaches listed are used in order to achieve set goals. Some approaches are easier to use in certain situations than others and it is the developer's decision to use what fits them the most and seems the easiest to use at a certain stage of the project.

1.3. Web Scraping Libraries

When it comes to web scraping there are many ways of tackling the issue. Python has an immense number of packages/libraries to use for certain different tasks. From math, arrays, data wrangling to building artificial intelligence models. In general, whatever task needs to be done there is a good chance that there is a library for that.

Web scraping has three main go-to libraries: Scrapy, BeautifulSoup4 and Selenium. Each of them has their pros and cons, different use cases and basically are a bit different environment to work with, but at their core they remain as a tool used to scrape the web. In order to fully grasp which tools are available for what purpose, it is crucial to understand what each of the libraries offer and how they differ in developing a project. In this project all of the three libraries have been used.

1.3.1. Scrapy

"An open source and collaborative framework for extracting the data you need from websites. In a fast, simple, yet extensible way" [1]

Scrapy is the most popular web scraping tool, and it is exactly because it is not just a library that aids you in the process but a library that is an ecosystem in which you work and develop the program. On top of that, Scrapy is the fastest and beats everything else in terms of performance too – it is due to the fact that Scrapy is built on Twisted which lets Scrapy run asynchronously, meaning that it can perform tasks simultaneously without waiting for each one to finish ("Scrapy 2.5 documentation — Scrapy 2.5.0 documentation", 2021). Scrapy offers an amazingly complex and robust ecosystem when creating a project. In the project there are a lot of different components responsible for different things stored in different files in the project. For example, the project can consist of a spider, items, item loader, item pipeline, exporters and more. However, these features provide benefit you are not forced to use them, sometimes the complexity of this ecosystem gets in the way when writing something simpler or when we are experimenting.

Scrapy has a built in XPath and CSS data extraction support, as well as a "Scrapy Shell" which can be used for testing the selectors and parts of code inside of said shell. It is a wonderful idea, because yet again Scrapy manages to include a "built-in way" of handling the project without having to resort to third party software or other means of achieving your goals. The shell lets the user fetch a website by inputting a URL and try out some of the selectors using either CSS or XPath. Once the testing is done and it succeeded you can grab the code to the main file of the project. The shell looks exactly like a regular command prompt would look like . This library also offers building scrapers with more of a spider/crawler

[1] Retrieved May 03, 2021, from https://scrapy.org/

approach. Which is a common goal in many projects, because spiders have the ability to constantly crawl websites (e.g.: crawl websites everyday), follow links and append the data to a database.

Due to the performance and extensibility of the library it is the best one to use in big, scalable projects. A big project could consist of Scrapy (crawler framework), MongoDB (database) and Django (interface, websites). All of these tools can be used with each other to create a fully-fledged operational system. This is also one of the hardest and most complex way to approach a project.

1.3.2. BeautifulSoup4

“Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work.”[2]

Beautiful Soup, also among the most popular libraries considering scraping, is considered to be the “smaller brother of Scrapy”. Beautiful Soup needs a bit more to work “out-of-the-box”, since it does not come with a parser or a request system included. Beautiful Soup is more of a library that is on top of a parser and its job is to make the process of working with the content easier; like selecting, viewing, and finding data. In order to use BS4 it is first necessary to download and install a parser and a request module. Among the most popular choices are *lxml* and *html.parser* (for parsing), and *requests* or one of the *urlib* modules. With that said, Beautiful Soup is not so self-dependent, but with additional modules it fulfills its role as it was meant to. This library also supports CSS but does not support XPath (though there is a workaround). It has its own functions for finding elements like find(), find_all() or making the output pretty in order to make it readable and easier to navigate through for the user. Having its own functions for finding elements is a huge advantage as it can be very useful to use in certain situations. You can find a lot of stuff and extract it from a webpage by using these functions in one or two lines of code.

Beautiful Soup is also the fastest way for creating a minimum viable product (MVP) from scratch. It is easy to learn and pretty intuitive at how it handles problems. Beautiful

[2] Retrieved May 03, 2021, from https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Soup combined with requests, a parser and Jupyter Notebook is probably the best way to experiment with scraping, when the experiments begin to be too complex to be handled comfortably in scrapy shell. Even when they are not too complex Jupyter Notebook provides a better structure of running code blocks and saving progress either way.

Beautiful Soup is a lot easier to implement into a frontend due to the fact that it is down to the user to build the structure of the project, while in scrapy the whole framework can get into a developer's way and throw errors a lot. Beautiful Soup projects are more connected to Python's syntax, the language itself and object-oriented-programing (OOP), which makes handling the project easier and connecting the project to an interface also easier, since you already know everything about the project's structure, naturally so by building it by yourself. Unfortunately, BS4 is not as fast as Scrapy and does not come close to it in terms of items that it offers as a whole. However, it still remains one of the most used libraries due to it being so user friendly and sometimes extremely powerful frameworks are just not needed to achieve goals in simpler tasks.

1.3.3. Selenium

"Selenium automates browsers. That's it! What you do with that power is entirely up to you. Primarily it is for automating web applications for testing purposes but is certainly not limited to just that. Boring web-based administration tasks can (and should) also be automated as well."[3]

Selenium is a bit different because as stated on their website it was created with automating the process of testing if a certain website's user interface is working properly without having to click everything manually. The main difference is that in previous examples the libraries do not "paint" the design of the site, because it is not needed only the extracted data is. This enables a whole lot of new activities, but it also makes the process a lot slower ("Selenium documentation", 2021)[4]. Selenium is a great tool when it comes to more complex sites, dynamic sites, e.g., automating Instagram's site requires you to scroll due to the infinite scroll feature in their design, that will not load the content – meaning that it will not be available for download unless you scroll.

[3] Retrieved May 03, 2021, from https://www.selenium.dev
[4] https://www.selenium.dev/documentation/

Selenium is also great in tricking websites into thinking that your program in fact is not a robot by implementing features that impersonate humans. Those features include but are not limited to mouse movement instead of mouse “teleporting/snapping”, time-waits, operating in a browser instead of just sending requests, cookie manipulation and more.

1.3.4. Concluding

As presented, the packages have their pros and cons, but these libraries also propose something unique and are to be used for different things. With that said, there is no best module, there is only a module that fits the situation and your requirements best. They can also be used to together with each other or alternately, meaning it is not necessary to choose only one module. When it comes to the language responsible for selecting and extracting data from html webpages, this is mostly the same regardless of which package you decide to use. Scrapy excels because it has the shell and built-in CSS and XPath. BS4 does not come short here because it too can have XPath and CSS selectors and on top of that it has its own selector functions. When it comes to performance Scrapy is better, but for beginners it will be faster to build a scraper from scratch using BS4.

1.4. Preventing Web Scraping

As the phenomenon of web scraping grows, so does the concern of businesses that own websites with information. Website hosts have to come up with ways to combat web scraping for whatever reason. Due to the nature of web scrapings sending a lot of requests quickly, poorly developed scrapers can be mistaken as DoS or DDoS attacks (Denial of Service) (Sourav and Mishra, 2012). Attacks like those are meant to exhaust network bandwidth by sending a lot of requests at once continuously, ultimately resulting in a certain website going offline due to excess traffic. This definitely is not the goal of web scrapers, but websites have to prepare for such attack regardless, sometimes tagging web scrapers as threats too (Marques et al., 2018).

It is important to understand what type of measures are taken in order to prevent scraping and keep websites safe in order to deploy a scraper that successfully scrapes without harming the host of the website. From simplest to more complex here is a list of measures taken to prevent scraping.

1.4.1. Traffic, IP, requests

As already noted, the first and foremost way of blocking scrapers is to limit how many requests users can send in a certain timeframe from the same IP address or client. If a single IP or client is flooding the server there is a very high chance that this is an automated activity. This measure only takes care of basic scraping and DoS attacks. It is important to carefully set up a system that distinguishes ‘good’ packets from ‘bad’ packets, as to not harm the regular users (Jing, et al., 2006).

1.4.2. User Agent, Headers

Websites can distinguish who is connecting to them by the client that they are using, e.g., which browser is used, by checking the user agent. As an example, a user agent of my default browser is as follows: “Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.128 Safari/537.36 OPR/75.0.3969.243” After using any parser it is easy to deduct what client I am using on what system. On the courtesy of whatismybrowser[5] let us investigate into what client I am using.

Table 3. Extracting information from user agent.

Software	OS	Software Full	OS Full	Simple String
Opera 75	Windows 10	75.0.3969.243	NT 10.0	Opera 75 on Windows 10

Headers are also checked along with the user agent and can also be helpful in determining which user to blacklist. HTTP headers are used for passing information between the client and web server – informing it which type of algorithms can be used, which language, encoding and referer. Taking a look at the most common examples (Kirjazovas, 2021) of said headers.

Table 4. Example headers, from: https://oxylabs.io/blog/5-key-http-headers-for-web-scraping

Header	Example value
HTTP header Accept-Language	en-US

[5] Retrieved May 07, 2021, from https://developers.whatismybrowser.com/useragents/parse/

HTTP header Accept-Encoding	gzip, deflate
HTTP headers Accept	text/html
HTTP header Referer	http://www.google.com/

Clients should be blacklisted based on the uniqueness of the client judging by the user agent and the headers. If user agent is empty, headless, or using only the most common scraper user agents, or if the headers are among the most common ones (JonasCz., 2021).

1.4.3. Website structure and repetitive behavior

Most of these web scrapers are based off the website's DOM structure (Gupta, et al., 2003), by extracting items from the HTML source (Rahman, et al., 2001) – as we have seen in the example of selectors by HTML differences. Changing the code of the website, the structure or even rotating the class names randomly will render web scrapers useless. Changing the code manually requires so much intervention that it might be hard to accomplish, as it is very tedious. Randomly generating the code, or names of classes and elements might be better. Adding another layer of markup text in the code will disable scrapers until they adjust the scraper's code for the site.

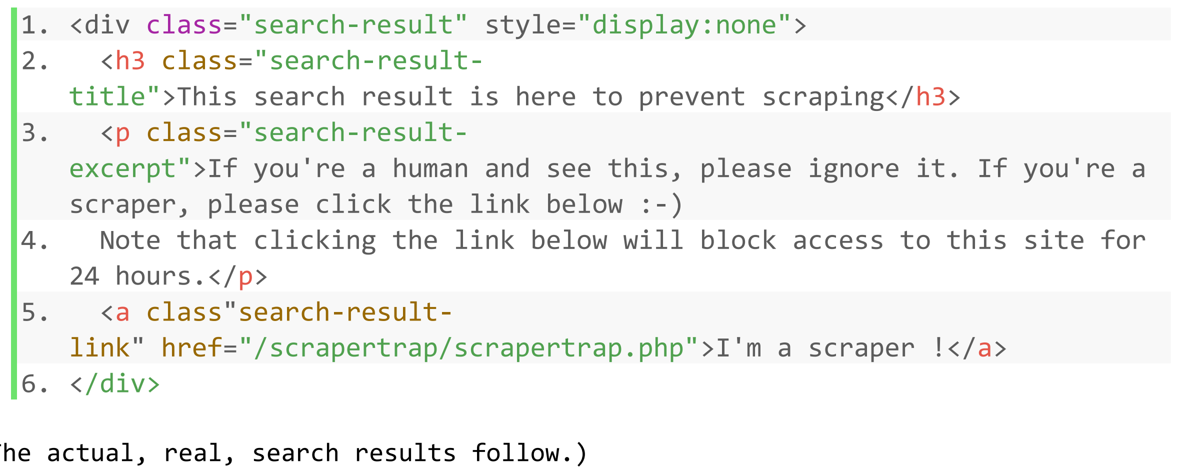
If the key structure is known, it is easier to predict which points in the structure are points of interest to potential web scrapers. Web scrapers will scrape according, anchored to those points, e.g., a division in code named "price" is easily targetable by bots. It is easy to track bots that access only 3 elements one by one, repetitively. The same points of interests can be used to introduce another layer, or frequent classes name changes to throw off scrapers.

1.4.4. Honeypots

Honeypots (Mokube and Adams, 2007) are traps for robots which are usually not visible to the typical user but are hidden in the source of the page. Hidden in a way that scrapers that follow all results will find the honeypot and attempt to enter it. Once a scraper has fallen in such a trap it can be completely banned. It is easy to distinguish this form a user because users have no way of finding the honeypot in the visual representation of the code. After catching a scraper in a honeypot, it can be analyzed where it came from, what are its weaknesses and analyze its attack patterns. It is worth to note that honeypots are designed not only for scrapers but for malicious attackers in mind too – like request rare limits (Spitzner, 2003).

Honeypots can be implemented when updating the source of the webpage. When updating the last points of interest should be left in the code, but instead should redirect the scraper into an invisible honeypot. This measure also needs to be updated frequently as scrapers may adjust (JonasCz., 2021).

Sample code of a honeypot may look as follows[6]:



```
<div class="search-result" style="display:none">
  <h3 class="search-result-title">This search result is here to prevent scraping</h3>
  <p class="search-result-excerpt">If you're a human and see this, please ignore it. If you're a scraper, please click the link below :-)
  Note that clicking the link below will block access to this site for 24 hours.</p>
  <a class"search-result-link" href="/scrapertrap/scrapertrap.php">I'm a scraper !</a>
</div>
```

```
(The actual, real, search results follow.)
```

Figure 2. Sample honeypot code, from: https://github.com/JonasCz/How-To-Prevent-Scraping

1.4.5. reCAPTCHA

CAPTCHA or Completely Automated Public Turing Computer and Humans Apart ("The reCAPTCHA Project - Carnegie Mellon University CyLab", 2017) is nowadays a popular tool in combating automated responses throughout the internet. CAPTCHA is a system that requires a user to complete a challenge test which is used to determine whether the client is a regular user or a robot. Whenever a correct answer is returned the program classifies the user as a human, otherwise as a bot. The aim of this will always be to remain as a gatekeeper program in order to stall or ridicule bots from a website. It is important for the program to be easily solvable by humans yet hard-to-solve by robots. It is not a flawless system, but a prominent one (von Ahn, Blum and Langford, 2004).

[6] Retrieved May 07, 2021, from https://github.com/JonasCz/How-To-Prevent-Scraping#screw-with-the-scraper-insert-fake-invisible-honeypot-data-into-your-page

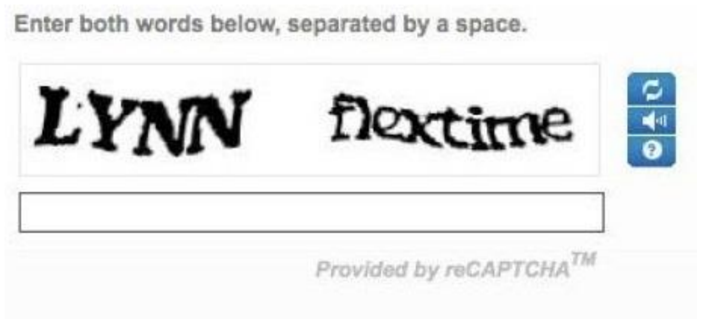


Figure 3. Sample captcha word challenge, from: https://www.huffpost.com/entry/captcha-defeated-by-computer-software_n_4168784

Implementing CAPTCHA is not limited to putting the test as a foremost thing on the website. For example, the CAPTCHA test can be shown after a client is deemed suspicious. As already explained how to target suspicious activity – CAPTCHA can be shown as a tool to verify if the client was indeed a suspicious activity or not. If the bot stops on CAPTCHA and cannot solve it, it cannot continue to scrape. The same principle applies to CPTCHAs in honeypots, registration services, old HTML links, agent, and headers identification and such.

There are many variations of CAPTCHA, but all of them serve the same purpose of filtering out the robots. Among the most important ones are (Singh, Pal, 2014):

- Text based CAPTCHA – simple, requires dictionaries / text banks, easy to solve.
- Image based CAPTCHA – visual puzzles, image / pattern recognition, harder to solve.
- Audio based CAPTCHA – contains audio clips and requires the user to write it down.
- Puzzle based CAPTCHA – contains puzzles, parts of images, or literal puzzles.

Among the newer solutions are reCAPTCHA v2 Checkbox which requires only to click a box, will either pass a user further or require to complete the challenge. Invisible reCAPTCHA v2 which works in the background and requires only the suspicious users to complete it. ReCAPTCHA v3 which is an API written in JS, scores users in the background and does not require additional action, throttles suspicious users when needed.[7] CAPTCHA can be harmful towards user experience as many robot-preventing measures can but does not

[7] Retrieved May 07, 2021, from https://developers.google.com/recaptcha/docs/versions

have to be. On top of that CAPTCHAs that work in the background are seen as less secure (Tanthavech and Nimkoompai, 2019).

1.4.6. Provide APIs

API ("Application Programming Interface", 2021) is a list of operations that are aiming to reduce the complexity of the code and the amount of code that developers would have needed to write otherwise. This system enables a fluent interaction between two separate applications. Operations available in APIs are supposed to provide a solution for certain actions that a developer would like to use, meaning that the developer also does not need to know how the operation works and what the code looks like, although reading the code always helps with understanding what is happening in the background. The point of this system is to establish a connection between different applications and to make the work easier for developers (Hoffman, 2018).

In the example of scrapers, sites may provide an API that would help them in the extraction of information that is needed. For example, auction sites may have an API that helps in gathering offers, uploading offers, editing, or removing offers. The problem with APIs is that they sometimes do not work as expected. Sometimes APIs can cause more trouble than help the developer, further straying away from the initial goal of this system (Henning, 2007). APIs are also prone to errors and outages (obviously, depends on the API) (Lu, et al., 2013), therefore users may still choose to scrape websites even if an API is present. APIs may be slower than well-written scrapers and may have impact on the credibility of the data gathered. It is not always better to use an API and APIs do not exclude presence of web scrapers (Dongo, et al., 2020). APIs are not always free.

1.4.7. Less technical measures and sidenotes

Among the less technical are “tricks” that make scraping harder. First of all, the information that would be forbidden from scraping could be embedded into media objects like JPEGs, PNGs etc. This makes the content, i.e., text “untargetable” meaning that users nor scrapers cannot copy what is displayed inside of said object. This adds another annoying step for scrapers as it is not something that cannot be dealt with using a workaround (OCR – parsing text from images). Another step could be reconsidering what information to display. There is a high chance that anything that is put on the web can be scraped no matter what anti-scraping systems are in use. If there is sensitive information that cannot be seen by scrapers and bots, it might be beneficial to not display the information if it is so important. If displaying said data is in the interest of some party, another step to take is to hide the data behind a login session. Users would need to create an account and login using provided credentials to access important information. CAPTCHA can be added in the registration process as well as the login process, as well as two factor authentication, email verify. This is tedious for scrapers and might discourage just enough of them from scraping a website.

Last is to seek assistance from a lawyer and include in the Terms of Use / Terms of Service that it is indeed illegal to scrape the website at hand. This measure is only as effective as people make it to be, since part of people will stop scraping and part will not. Legalities of web scraping still remain in the grey area, yet wrongfully executed scrapers can suffer from a lot of legal backlashes ("Instamotor Agrees To Pay Craigslist $31M, Stop Scraping - Law360", 2017).

Many of these measures also hinder the usability of the site for the regular user. Therefore, it is necessary to introduce optimal measure with reasoning as there might be more harm introducing those if it is done wrong. Web scrapers most of the time have a way to bypass security measures. If someone is very motivated to scrape the site – they probably will. It may be beneficial to work with scrapers instead of banning them continuously. Fighting the scrapers may bring more harm than good at some points – if the scrapers are assisted and sites are written with them in mind, they would not have to attack sites with so many requests, and if web scrapers are written with the host in mind too, then both parties can enjoy the best of both worlds.

1.5. Avoiding being blocked

When it comes to being blocked and avoiding it, it all comes down to complexity of the scraper and amount of information that it scrapes – how many requests it is going to send. Then it depends on the site’s implemented measures and how hard the host is trying to get rid of scrapers.

1.5.1. Robots.txt

Rule of thumb is to behave nicely when scraping, respect the site and respect the robots.txt. Robots.txt is a file appended to the URL of any website, e.g., example.com/robots.txt. In the file there are instruction considering which part of the website we can scrape, which we cannot, which user-agents are banned (‘*User-agent:* *’ means that settings specified are for any user agent), and additional information directly affecting scrapers like request frequency and such. If a scraper acts accordingly to those instructions the chances are smaller that the scraper will be blocked – but that does not mean that it will not be blocked at all. Most of the time it is still needed to figure out the site’s anti-scraper system by trial and error. Here is a sample robots.txt file taken from OLX:

```
# sitecode:olxpl-desktop
Host: https://www.olx.pl
Sitemap: https://www.olx.pl/sitemap.xml
User-agent: *
Disallow: */ajax/
Disallow: /adminpanel/
Disallow: /api/
Disallow: */facebook/
Disallow: */rss/
Disallow: */konto/
Disallow: */mojolx/
Disallow: /drukuj/
Disallow: /oferta/ulotka/
Disallow: /oferta/kontakt/
Disallow: /platnosci/
Disallow: /nowe-ogloszenie/confirm/
Disallow: /nowe-ogloszenie/confirmpage/
Disallow: /i2/oferta/abuse/
Disallow: /m/oferta/abuse/
Disallow: /i2/oferta/kontakt/*
Allow: /
```

Figure 4. robots.txt that can be found on http://olx.pl/robots.txt.

If a scraper behaves and acts accordingly towards the robot.txt instructions AND scrapes slowly (with waits, without flooding requests) there is a good chance that it will not be blocked, but again it depends on the sites policies.

1.5.2. Rotating proxies, IP

If a client sends an abnormal amount of request to the host, the client has a very high chance of getting banned. The number of requests is calculated by looking at unique IPs and how many requests they have sent. Due to this fact, in many cases, it is necessary to implement a solution that masks the IP of our robot and rotates it between requests. Though, this is not always necessary, because if there is a small scraper it may not be blocked at all. In order to rotate IPs, it is needed to find ‘proxies’ through which our robot can be routed to make it seem like the connection is coming from different places on earth. It is needed to somehow obtain those proxies through different means. There are places on the internet in which you can find free proxies, and there are services that provide proxies or even provide an entire API for requesting an HTML page, e.g., Scrapingdog[8]. Usually, it is better to resort to the paid options, because free proxies in many cases are flagged as suspicious from the get-go, or they are just unusable, meaning that they are incredibly slow, choke on requests and may not return the source at all.

This is a basic measure of avoiding being blocked (in medium to large projects) due to the fact that every site has anti DoS measures which are working on the basis of numbers of requests from certain IP.

1.5.3. Rotating Headers, User Agent

This section works similarly to the one above, i.e., counts number of requests, but this time instead of IP it takes the client – which is the robot's browser and information about it. As I have already presented, information that headers include are: OS, what browser, which version, where it came from (referer), and what it accepts.

A single user from a single user agent will not send many requests as when a regular user browses the web, do not open 25 requests in a matter of seconds. Therefore, it is important to rotate the user agent and headers accordingly in order to bypass this measure. For example, instead of sending 25 requests from one set of user agent and headers, it can be divided by 5 requests. Meaning that every 5 requests the program would randomly pick a set of headers which is no longer suspicious. It is worth to note that part of this solution also includes obtaining the headers as they need to be headers that are commonly used and up to date. This section as well as the one above focuses on spoofing user behavior into our robot, by dividing the requests into a list of IPs and headers – always done randomly.

Once the list is obtained it is not hard to implement such a feature into the code. Usually, a sufficient way to implement it is to create a python loop that uses a range of numbers and picks headers assigned to a number that was randomly picked. On every iteration a random

[8] Retrieved May 09, 2021, from https://www.scrapingdog.com

header will be picked. A library by the name of 'random', responsible for random choices, may also be used to make the process easier.

1.5.4. CAPTCHA

Avoiding CAPTCHA can be very hard, tedious and a time-consuming process. It is best to approach scraping with a strategy that would focus on avoiding being banned by being flagged as suspicious and with that also avoiding being flagged for CAPTCHA. Sites sometimes instead of outright banning users, instead redirect them to a CAPTCHA page – but that does not make the situation much better for scrapers. The robot testing system was literally made to stop robots, naturally it is not easy to crack this measure. However, it is not impossible. Artificial Intelligence models can use OCR to resolve text-based CAPTCHAs, image recognition and voice recognition for image and voice based CAPTHCAs, respectively. Puzzles can be a bit harder, but it is also possible. There are also paid services that solve CAPTCHAs for you[9], or browser addons that work as solvers. The addon can be used with selenium to resolve CAPTHCAs, although it slows down the process a lot. CAPTCHAs can also be tricked by loading cookies, but it is a pretty outdated method as of right now, since a lot of sites reset those cookies pretty frequently.

1.5.5. Others

In reality there might be more measures taken to combat robots. Some sites are easier to scrape, and some sites are very hard. It depends on the host and data available there (e.g., how sensitive it is). The basis of avoiding being blocked revolves around trying to look like a regular user. Some other measures worth to keep in mind are:

- Repetitive patterns – if the robot constantly performs the same operations, over and over again, there is a high chance that it will be blocked based off just that. It can be beneficial in some situations to also rotate the way data is extracted, and which part goes first.
- JS (JavaScript) / AJAX (Asynchronous JavaScript And XML)[10] execution – Website can also execute JS in order to check if you are a real browser, a lot of the times when requesting a raw HTML site, we may not retrieve because a JS found out that it is not

[9] Retrieved May 11, 2021, from https://solvecaptcha.net
[10] Retrieved May 11, 2021, from https://www.w3schools.com/whatis/whatis_ajax.asp

a website (because it cannot open a JS). AJAX sites work in a way that the content loads only when the browser triggers an event that requests the object, which is then fetched to the site, and the content of the site updated. There are ways of bypassing both of these measures. Either by using headless browsers / web drivers (like selenium), or by intercepting AJAX calls and reproducing them, implementing a feature in python that lets the robot run those requests and scripts[11].

- TLS fingerprinting / Browser fingerprinting [12] – Is a security protocol made to uniquely identify users based on this system. It is more of a complex way of keeping track of users, thus may not often be implemented to combat scrapers. It is hard to spoof or bypass and is beyond the scope of this paper.

II. Building the project

2.1. Preparation

2.1.1. Approach

Building projects in any programming language requires a certain attitude and mindset to go about the issue properly. Building such projects basically comes down to problem solving. When writing a program that is connected to the businesses, we are practically connecting the project with the real world – and real world usually has more problems than class exercises or sample projects. Writing a scraper usually means many, many problems and after going and solving one after another the program starts to work. The process looks a lot like trial and error, especially when writing a scraper for the first time, because it requires to learn the basics and structure of HTML, the selector languages and learn libraries responsible for scraping. In order to learn how these all work together it is needed to try out possibilities, test how things work and try to solve easy problems by trial and error. When writing a scraper for the first time it usually takes a lot of time – more than expected and can be a tedious task when trying to come up with an educated guess about the solution to current problem by trialing.

[11] Retrieved May 11, 2021, from https://stackoverflow.com/questions/8049520/web-scraping-javascript-page-with-python
[12] Retrieved May 11, 2021, from https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/

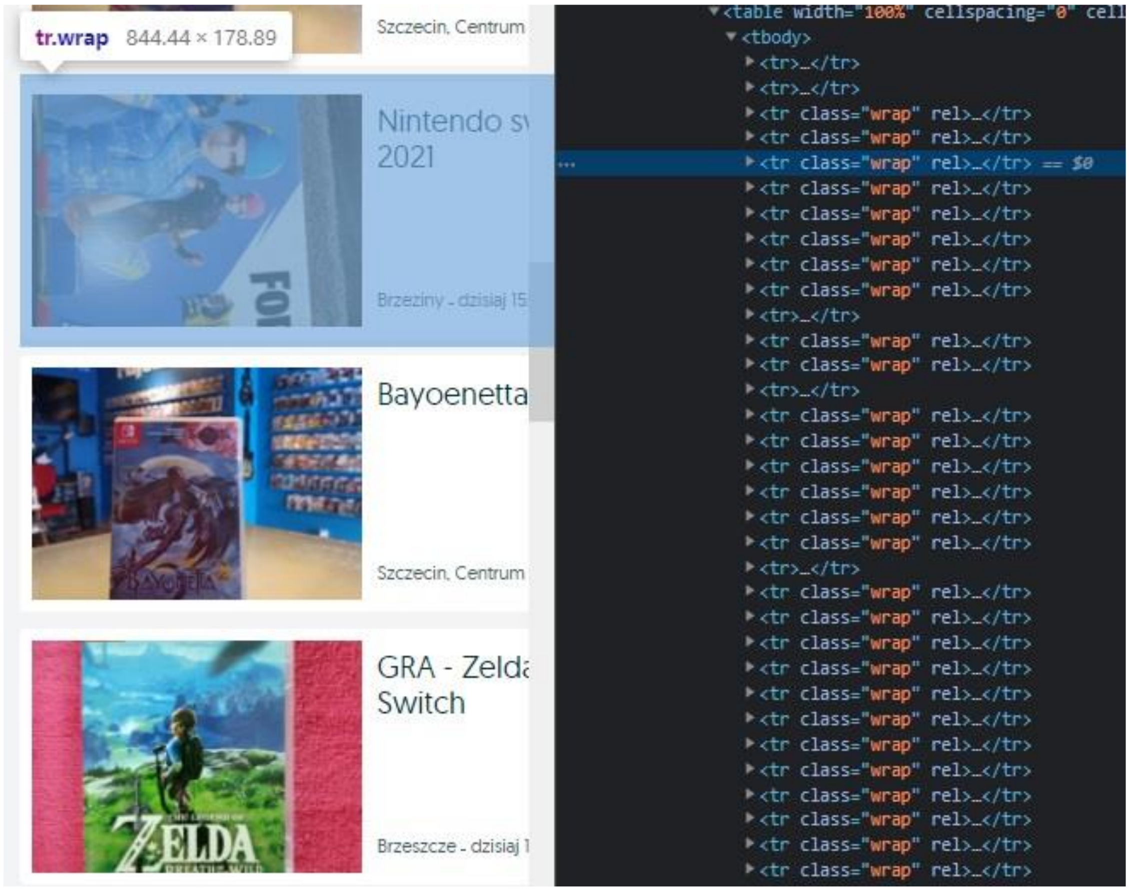
The process of developing the scraper follows the next two parts in which I began with testing the selectors in order to grasp how the robot would work. However, initially I began testing only in Scrapy’s shell and wrote my first scraper for Allegro[13]. I moved to BS4 much later and only due to the fact that BS4 is a lot easier to connect to a frontend, because it does not have such a complicated structure.

2.1.2. Website language and selectors

In order to correctly extract information and select elements from websites it is crucial to understand how the website is built and how the selectors look at the code. Most of the time, the testing begins with visiting the site in an actual browser, pressing “Inspect element” on some element of interest on the site, and looking through the code to find out how we can anchor the scraper to always extract the right part of the page.

Taking OLX as an example we inspect the source code of a random Nintendo Switch offer’s price. The source code of the site will open, and we will be met with the structure which will point to the exact place where the price is located inside of the code. Unfortunately, selecting elements in a scraper is not as simple as that, because we need to provide sufficient information in order for the scraper to extract only the information that interests us and not something additional that would be scraped by a mistake and ruin the data.

[13] https://www.allegro.pl/



```
▼<table width="100%" cellspacing="0" cell
  ▼<tbody>
    ►<tr>…</tr>
    ►<tr>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr class="wrap" rel>…</tr> == $0
    ►<tr class="wrap" rel>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr class="wrap" rel>…</tr>
    ►<tr class="wrap" rel>…</tr>
```

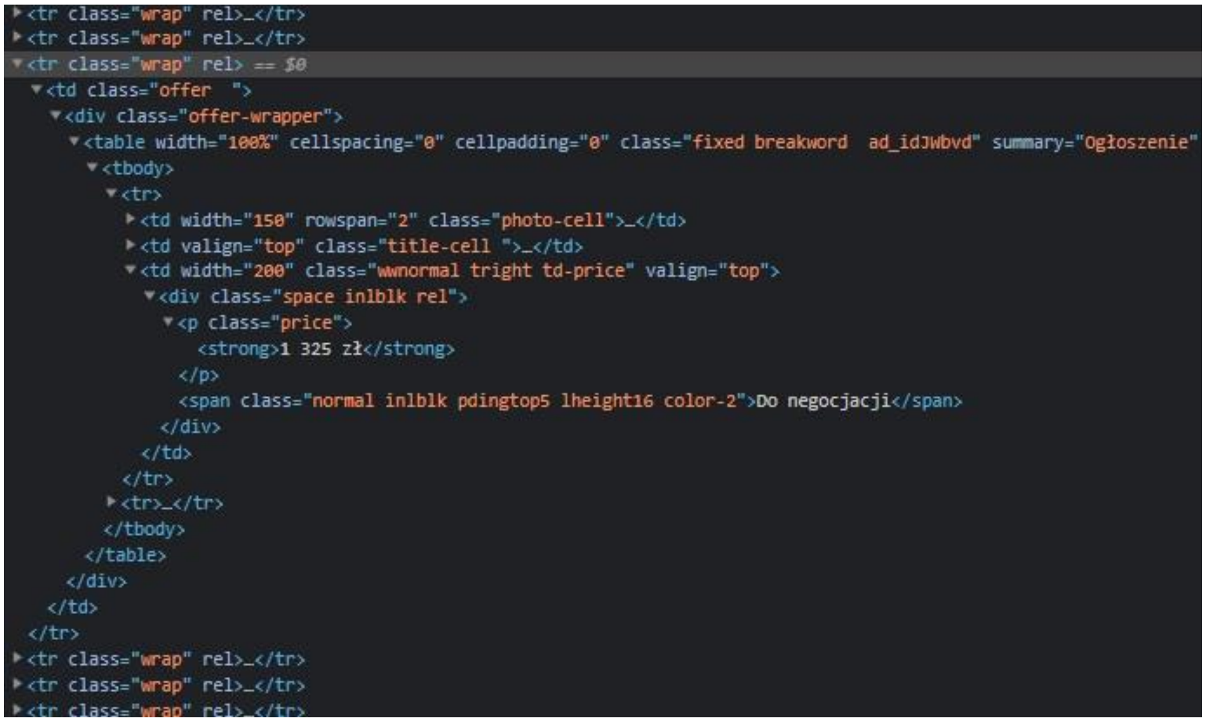
Figure 5. Side by side comparison of the site and the code, respectively. Sorted by offer; every offer has its own group of elements.

After inspecting the source code, we can notice that every offer is wrapped inside of group of elements inside of a <*tr*> tag with a *class* named "wrap". The <*tr*> tag stands for a row in any HTML table. Conveniently, the data (meaning offers) is displayed in a data frame / table like manner where each offer is a different row, hence the source code tags. We can select the offer by passing in exactly those parameters. Having said that, we need to pass information to the scraper so that it looks for every element that has the <*tr*> tag and then that has a *class* named "wrap", this can be done in the following way(CSS and XPath, respectively):

```
response.css("tr[class='wrap']")
response.xpath("//tr[contains(@class, 'wrap')]")
```

Figure 6. CSS & XPath selectors for 'wrap' class element.

After guiding the robot how to find the offers we need to pass another parameter that will tell it where to find the price. After unraveling the code, we can see that there is a paragraph (indicated by the <*p*> tag) with a *class* named “price”.



```
▸<tr class="wrap" rel>…</tr>
▸<tr class="wrap" rel>…</tr>
▾<tr class="wrap" rel> == $0
  ▾<td class="offer  ">
    ▾<div class="offer-wrapper">
      ▾<table width="100%" cellspacing="0" cellpadding="0" class="fixed breakword  ad_idJWbvd" summary="Ogłoszenie"
        ▾<tbody>
          ▾<tr>
            ▸<td width="150" rowspan="2" class="photo-cell">…</td>
            ▸<td valign="top" class="title-cell ">…</td>
            ▾<td width="200" class="wwnormal tright td-price" valign="top">
              ▾<div class="space inlblk rel">
                ▾<p class="price">
                    <strong>1 325 zł</strong>
                  </p>
                  <span class="normal inlblk pdingtop5 lheight16 color-2">Do negocjacji</span>
                </div>
              </td>
            </tr>
          ▸<tr>…</tr>
          </tbody>
        </table>
      </div>
    </td>
  </tr>
▸<tr class="wrap" rel>…</tr>
▸<tr class="wrap" rel>…</tr>
▸<tr class="wrap" rel>…</tr>
```

Figure 7. Unraveling the offer’s collection of elements in order to find where the price is located, so that we can pass it to the robot later.

Similarly as before, we need to guide the robot to find the paragraph with the price name, though this time we need to append the selector with the previous one, which can be done as following:

```
response.css("tr[class='wrap'] p[class='price']")
response.xpath("//tr[contains(@class, 'wrap')]").xpath("//p[contains(@class, 'price')]")
```

Figure 8. CSS & XPath selectors for price class element.

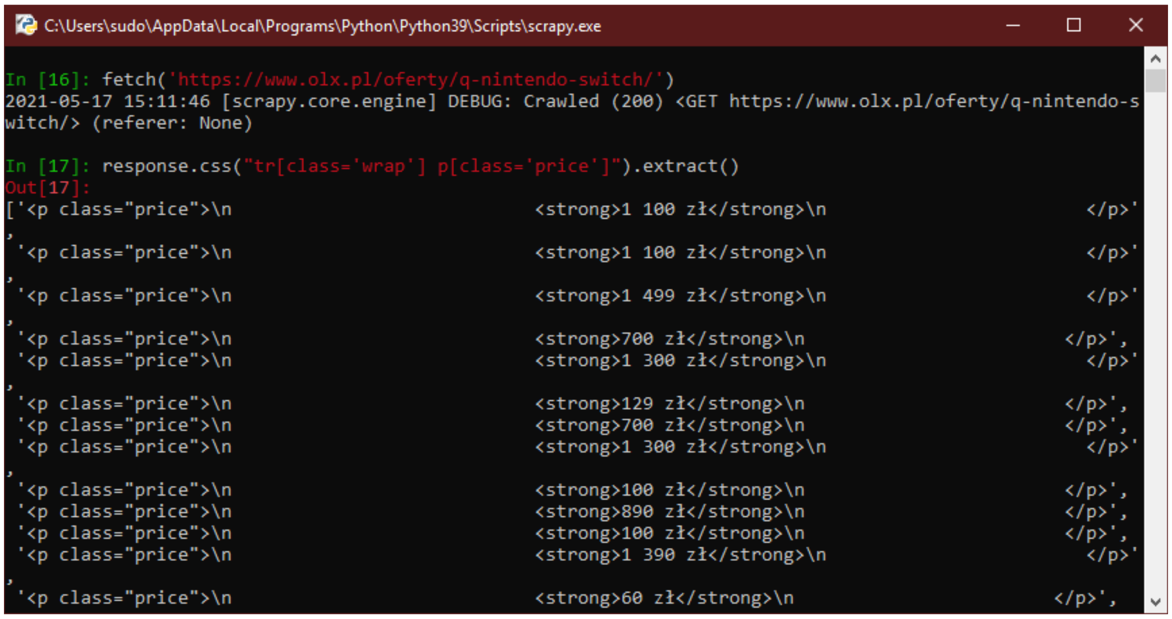
Here is a simplified process of what goes behind the scenes when passing a selector like the one above. The robot first looks for all elements with a <*tr*> tag, then from that lists it picks only those <*tr*> elements which also have a *class* named “wrap”. After that, from that list again it takes all of paragraphs (<*p*> elements) that also have a *class* named “price”.

Basically, you have to pass instructions of how the robot can navigate the page step-by-step in order to get to the desired information.

One could argue that it would be easier to just pass the *p[class='price']* selector and it would work in a lesser amount of code. While this is true in some cases, it is usually better to anchor the robot around couple of elements in order to stop it from scraping elements which are named the same but are in a different place. After all, the <*p class="price"*> element could also appear in the code outside of the <*tr class="wrap" rel*> element which is a holder for offers and we are not interested in anything that appears outside of the offers. On the other hand, it is important not to go overboard with how many elements we are anchoring the robot to, since every slight change in the source code of the site can ruin the scraper if it is anchored to the part of the website that got changed.

2.1.3. Scrapy Shell

When testing a site and the general language of extracting elements and information from said site the Scrapy shell or Jupyter notebook is something extremely useful. It is due to the fact that you can instantly see the output of the selectors and check if they provide the desired result. Initially, you need to provide the URL with the *fetch('example.com')* command, then you can run selector commands on the response by using either *response.css()* or *response.xpath()*. In order to illustrate how it works I will extract Nintendo Switch prices from OLX using Scrapy's shell environment.


C:\Users\sudo\AppData\Local\Programs\Python\Python39\Scripts\scrapy.exe

```
In [16]: fetch('https://www.olx.pl/oferty/q-nintendo-switch/')
2021-05-17 15:11:46 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.olx.pl/oferty/q-nintendo-s
witch/> (referer: None)

In [17]: response.css("tr[class='wrap'] p[class='price']").extract()
Out[17]:
['<p class="price">\n                           <strong>1 100 zł</strong>\n                        </p>'
,
 '<p class="price">\n                           <strong>1 100 zł</strong>\n                        </p>'
,
 '<p class="price">\n                           <strong>1 499 zł</strong>\n                        </p>'
,
 '<p class="price">\n                           <strong>700 zł</strong>\n                        </p>',
 '<p class="price">\n                           <strong>1 300 zł</strong>\n                        </p>'
,
 '<p class="price">\n                           <strong>129 zł</strong>\n                        </p>',
 '<p class="price">\n                           <strong>700 zł</strong>\n                        </p>',
 '<p class="price">\n                           <strong>1 300 zł</strong>\n                        </p>'
,
 '<p class="price">\n                           <strong>100 zł</strong>\n                        </p>',
 '<p class="price">\n                           <strong>890 zł</strong>\n                        </p>',
 '<p class="price">\n                           <strong>100 zł</strong>\n                        </p>',
 '<p class="price">\n                           <strong>1 390 zł</strong>\n                        </p>'
,
 '<p class="price">\n                           <strong>60 zł</strong>\n                       </p>',
```

Figure 9. Scrapy shell in use, how an URL can be fetched and extracted from inside of the shell, on example of Nintendo switch prices on OLX.

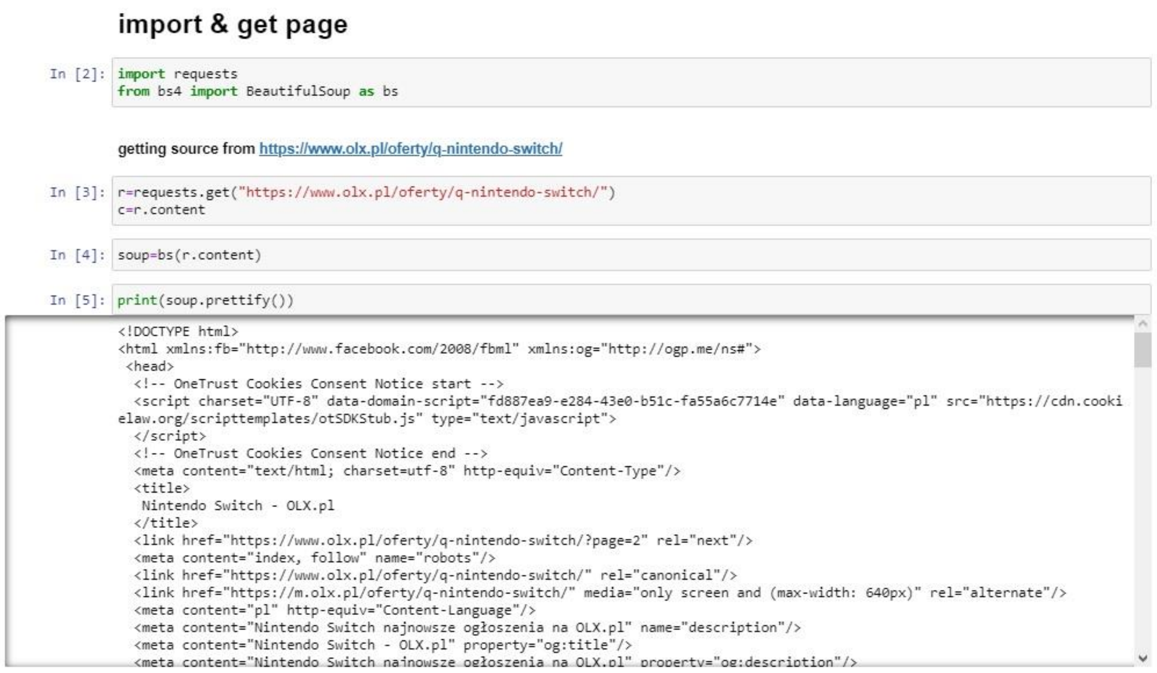
In this example we have used CSS in order to extract the prices, although we could have also used XPath which would look as following:

It is possible to import libraries and perform regular python expressions and functions inside of the shell, e.g., *replace()*. You can also write loops in there and save data to variables, which provides the developer with additional debugging and testing possibilities.

2.1.4. Jupyter Notebook

Jupyter Notebook is an excellent tool for learning a library, testing programs, or even building AI models. It is so prominent, because it lets you run certain parts of the code and see the output immediately – without running the whole application over and over again (you can even run line-by-line). Let us investigate into how Jupyter can be used when developing a scraper.

First, we need to import the required libraries – i.e., *requests* and *BeautifulSoup*. Then we need to request the site (OLX), parse the content, and print it using *prettify()* to check if it worked correctly. *Prettify()* is a function that just prints the content in a 'pretty' manner (tabulators, new lines, and spaces), hence the name of the function.


import & get page

```
In [2]: import requests
        from bs4 import BeautifulSoup as bs
```

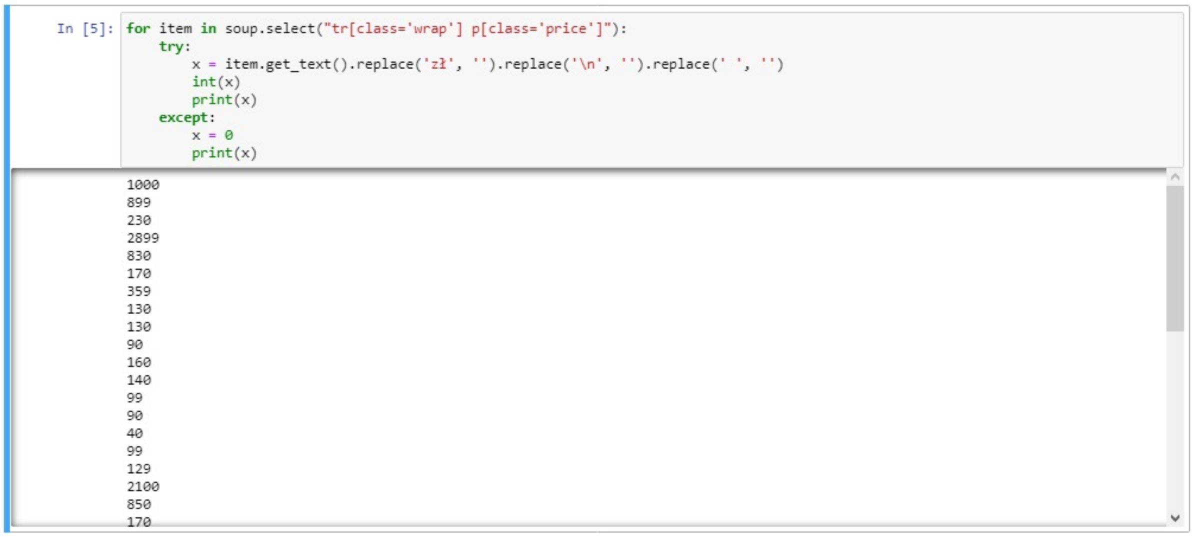
getting source from https://www.olx.pl/oferty/q-nintendo-switch/

```
In [3]: r=requests.get("https://www.olx.pl/oferty/q-nintendo-switch/")
        c=r.content

In [4]: soup=bs(r.content)

In [5]: print(soup.prettify())
```

```
<!DOCTYPE html>
<html xmlns:fb="http://www.facebook.com/2008/fbml" xmlns:og="http://ogp.me/ns#">
 <head>
  <!-- OneTrust Cookies Consent Notice start -->
  <script charset="UTF-8" data-domain-script="fd887ea9-e284-43e0-b51c-fa55a6c7714e" data-language="pl" src="https://cdn.cooki
elaw.org/scripttemplates/otSDKStub.js" type="text/javascript">
  </script>
  <!-- OneTrust Cookies Consent Notice end -->
  <meta content="text/html; charset=utf-8" http-equiv="Content-Type"/>
  <title>
   Nintendo Switch - OLX.pl
  </title>
  <link href="https://www.olx.pl/oferty/q-nintendo-switch/?page=2" rel="next"/>
  <meta content="index, follow" name="robots"/>
  <link href="https://www.olx.pl/oferty/q-nintendo-switch/" rel="canonical"/>
  <link href="https://m.olx.pl/oferty/q-nintendo-switch/" media="only screen and (max-width: 640px)" rel="alternate"/>
  <meta content="pl" http-equiv="Content-Language"/>
  <meta content="Nintendo Switch najnowsze ogłoszenia na OLX.pl" name="description"/>
  <meta content="Nintendo Switch - OLX.pl" property="og:title"/>
  <meta content="Nintendo Switch najnowsze ogłoszenia na OLX.pl" property="og:description"/>
```

Figure 10. Jupyter Notebook in use, importing the site and parsing it inside of Jupyter Notebook framework.

After the site is successfully fetched without any errors (e.g., 403 forbidden – meaning that the bot was blocked), we can again extract the price, but this time removing all of the unnecessary characters and changing the type of the object to *integer*. If the program fails to change the value to *int()* it will then save it as 0. We are using a for loop here in order to perform this action on every item in the list.



```
In [5]: for item in soup.select("tr[class='wrap'] p[class='price']"):
            try:
                x = item.get_text().replace('zł', '').replace('\n', '').replace(' ', '')
                int(x)
                print(x)
            except:
                x = 0
                print(x)
```

```
1000
899
230
2899
830
170
359
130
130
90
160
140
99
90
40
99
129
2100
850
170
```

Figure 11. A for loop that selects the prices form the html and transforms them into an integer or a 0.

Jupyter Notebook provides a perfect environment for testing the code and then perfecting it, improving it until it becomes the final product. However, it does not mean that it in itself cannot hold a finished product too. I will continue to present the code in Jupyter, due to means of easy representation of which code block does what.

2.2. Source code

2.2.1. Allegro Web Scraper Breakdown (Scrapy)

Initially, when I began scraping in February of 2021, allegro did not have such strict measures as it has right now. Nevertheless, it was not easy back then. Today, allegro has a good system of determining whether the connection comes from a real user or if it is from a bot. Usually, you will be met with a 403 error when trying to scrape allegro without headers and proxies, or maybe even with. Allegro also had some interesting class names and website structure which required a sort of reverse-engineering approach. I could not just simply anchor by class name as in the examples I provided above, because Allegro uses something that seems like randomized class names. Therefore, I needed to first somehow guide the scraper to find the names of the classes so that it can later iterate on those. I will elaborate on that later, for now let us take a look at the script responsible for running the spider and extracting the title, price, and URLs to all of the offers connected to a certain keyword query. I will break down the code into a step-by-step explanation in a bit simplified and concise manner.

```python
import scrapy
from scrapy.crawler import CrawlerProcess

class allegro_spidegro(scrapy.Spider):
    name = 'spidegro'

    start_urls = [
        f"https://allegro.pl/listing?string=nintendo%20switch"
    ]
```

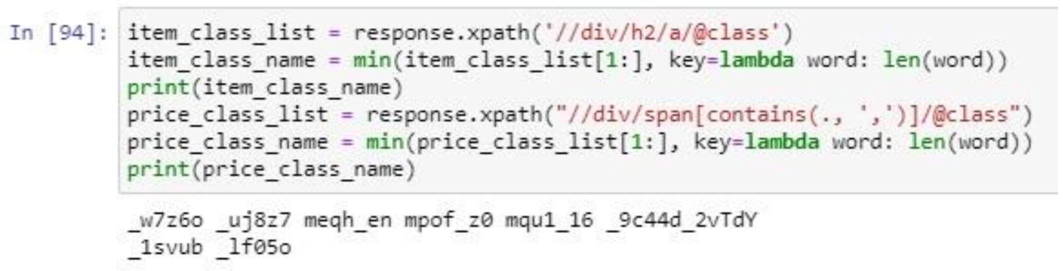
Figure 12. Creating spider class and specifying URLs in Scrapy.

As we know, first we need to import libraries into our file and connect to the website which in Scrapy is all done inside of the framework's base spider class *scrapy.Spider()*. We need to create our own new class and inherit from said class. It is necessary to provide the name of the spider (important for managing and starting spiders) and start URLs which are self-explanatory.

```python
    def parse(self, response):
        item_class_list = response.xpath('//div/h2/a/@class').getall()
        item_class_name = min(item_class_list[1:], key=lambda word: len(word))
        price_class_name = response.xpath("//div/span[contains(., ',')]/@class").get()
```

Figure 13. Beginning of parse() function.

After that, we have to define our *parse()* function. This function is responsible for the process that happens on the website and consists of finding the elements, extracting them, and parsing into whatever format we may need. Usually, the parse function begins with a for loop responsible for said action. In this case, the classes on Allegro are named in some randomly generated manner, or so it seems.



```
In [94]: item_class_list = response.xpath('//div/h2/a/@class')
         item_class_name = min(item_class_list[1:], key=lambda word: len(word))
         print(item_class_name)
         price_class_list = response.xpath("//div/span[contains(., ',')]/@class")
         price_class_name = min(price_class_list[1:], key=lambda word: len(word))
         print(price_class_name)

         _w7z6o _uj8z7 meqh_en mpof_z0 mqu1_16 _9c44d_2vTdY
         _1svub _1f05o
```

Figure 14. Reverse engineered class names from Allegro.

The way it works is that by looking at the site I try to pinpoint some pattern that I can follow and will follow if it is not critically changed. For example, that can be the website’s structure and some element that never changes. Here I used the structure and the fact that in that structure there always appears a comma. I created a list of all the class names and choose the shortest one from the list, because conveniently so – that is what works. At first, I have used regex (regular expressions) in order to grab the names, but it was very unstable, slow and took a lot more code. I only found this solution later into the development and it was about 3 or 4 times shorter than the initial version.

```python
        for item, price in zip(
            response.xpath(f'//div/h2/a[contains(@class, "{item_class_name}")]'),
            response.css(f'span[class="{price_class_name}"]')
        ):
            yield {
                'title': item.xpath('text()').get(),
                'link': item.xpath('@href').get(),
                'price': int(price.xpath('text()').get().replace(' ', '').replace(',', ''))
            }
```

Figure 15. For loop & yield, scraping and saving elements.

Onto the for loop. We have two responses in lists; thus it is needed to use *zip()* function in order to be able to iterate on both using *item*, *price* parameters. Inside of the zipped responses we pass the names of the classes that we have previously retrieved. After that we pass how we want to save our extracted data using yield. Using the get() function that is responsible for extracting, we extract title from raw text, link from the href element and price from the other list of classes. The price is also cleared from spaces and commas and changed to an integer value – using *replace()* and *int()*. With what we have now we can scrape every listing on an entire allegro page, but we also need to navigate through every page, it would be pointless to only scrape the first page.

```python
        next_page = response.css("a[rel='next']").xpath('@href').get()
        if next_page is not None:
            next_page = response.urljoin(next_page)
            yield scrapy.Request(next_page, callback=self.parse)
```

Figure 16. Instructing the scraper to the next page.

On the website, I have found an element responsible for navigating to the next page, inside of that there is an href that holds the URL. In order to navigate through pages, we need another loop. This one works in such a way that if the href element is not empty then it takes the URL and extracts information on that next page in the same way that we have provided in the previous step. However, if the element is empty the scraper will stop (which is a good thing, because there are no more pages to scrape).

```
process = CrawlerProcess(settings={
    "FEEDS": {
        "items.json": {"format": "json"},
    },
})

process.crawl(allegro_spidegro)
process.start()
```

Figure 17. Start the crawling process.

This part is only responsible for choosing the name and the format of the file that the scraper saves extracted information to. It is also one of the ways to start a scraper by running the script instead of using command prompt to run the spider. In case we want to do so, we can start it by writing *scrapy crawl spidegro -o item.json* inside of the project directory. The -*o* stands for output and is followed by name and format.

After a long time of trialing and testing we are able to scrape allegro in only 37 lines of code in an optimal way. However, the problem in scraping usually lies not in extracting the element but in gaining access to the website, which can take substantial amounts of time.

2.2.3. OLX Web Scraper Breakdown (BeautifulSoup)

Scraping OLX is the best and the most crucial of this project, because this is where the actual usability and value of the project comes into play. This part has helped me a lot in real life situations, due to the fact that I operate on OLX frequently when it comes to reselling and more.

```
import requests
from bs4 import BeautifulSoup as bs
import pandas as pd

r=requests.get('https://www.olx.pl/oferty/q-nintendo-switch/')
c=r.content
soup=bs(c, 'html.parser')

base_url = 'https://www.olx.pl/oferty/q-nintendo-switch/?page='
last_page = int(soup.select("a[data-cy='page-link-last']")[-1].get_text().replace('\n',''))
```

Figure 18. Imports, requests and initial preparation for BS4 scraping.

First, as usual, we have to import required libraries that we are going to use and fetch the page by using requests and pass it into the BS4 handler, using an *html.parser*. In this example we will be approaching the way scraper jumps from page 1 to page 2 to page 3 etc. in a different manner. Earlier we instructed it to follow the ‘next page’ button, this time we are going to pass a range into the scraper. This is why we need a *base_url* with unspecified page number and a *last_page* variable that is just a number extracted from the initial page by targeting an element that is responsible for sending users to the last page.

```python
l = []
for page in range(1,last_page+1,1):
    print('scraped: '+base_url+str(page))
    r=requests.get(base_url+str(page), headers={'User-agent': 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:61.0) Gecko/20100101 Firefox/61.0'})
    c=r.content
    soup=bs(c, 'html.parser')
```

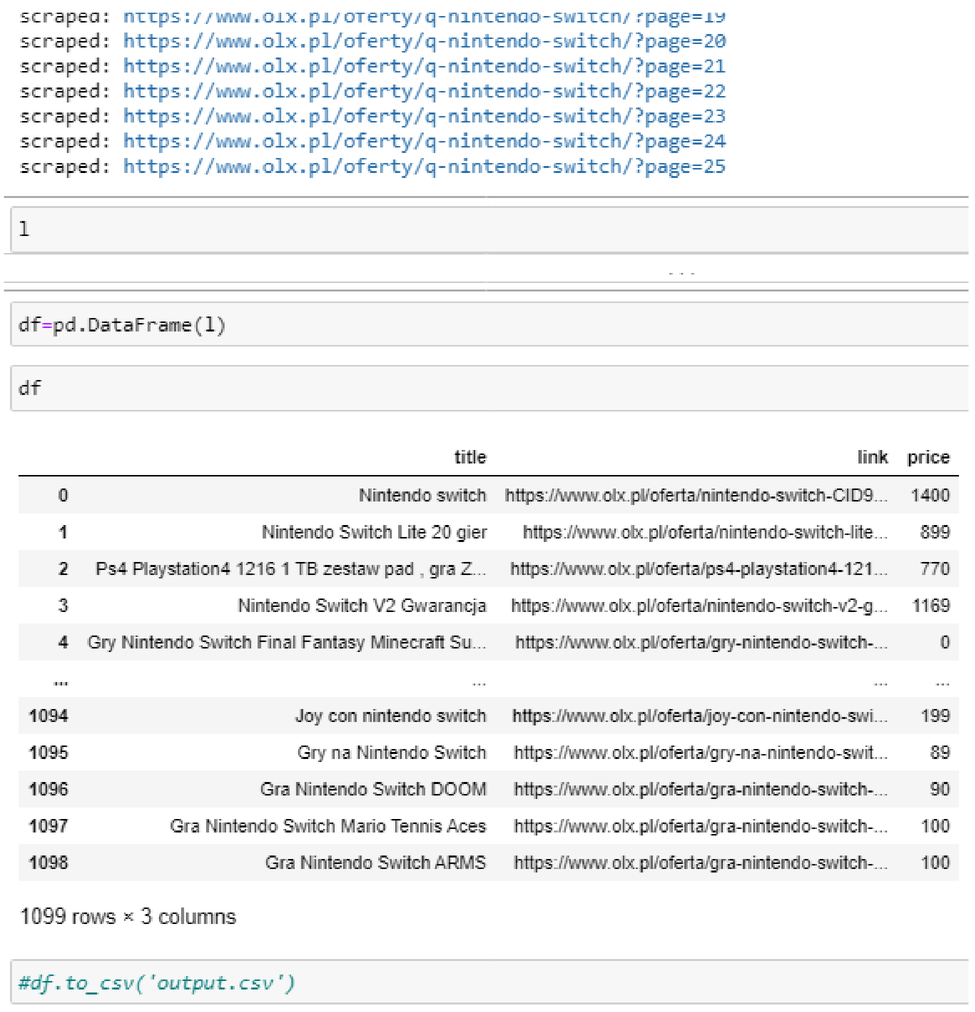
Figure 19. For loop responsible for crawling through all pages.

This is the beginning of the entire scraper loop. First, we need to create an empty list to which the program appends entries. We are passing a range that tells the program to start from 1 and end on *last_page+1* with a step of 1 – in this case we are adding 1 to *last_page*, because of how Python handles where range ends, similarly like the first element in Python is [0] not [1] when, for example, accessing elements of lists. Step means by how many ‘jumps’ the range is supposed to be carried out. If step were 2, we would go from 1 to 3 instead of from 1 to 2. Then we print just to see if the program works which is completely optional, then pass the URLs to the request function and then to the Beautiful Soup parser.

```python
    for item, price in zip(soup.select("tr[class='wrap'] td[class='title-cell'] a"), soup.select("tr[class='wrap'] p[class='price']")):
        d = {}
        d["title"] = item.get_text().replace('\n','')
        d["link"] = item.get('href')
        try:
            d['price'] = int(price.get_text().replace('zł', '').replace('\n', '').replace(' ', ''))
        except:
            d['price'] = 0
        l.append(d)
```

Figure 20. For loop responsible for the scraping action, then appending to list.

This is where the actual scraping takes place. As in the previous example we create a loop where we specify which elements to access, create an empty dictionary and append the content to the corresponding element names. When it comes to price, I had to implement some sort of solution for OLX auctions which do not have a price listed. Sometimes the auctioneers are specifying that they want to exchange their item for a different one (in this case the price equals to a string “Zamienie”), or they want to give it away for free (in this case the price equals to a string “Za darmo”). I used a try and except approach where the program is always supposed to try performing the first action where it converts the output to and integer containing the price. However, if the price is a string containing letters it cannot be changed to and integer, the program will raise an error. Here comes the exception where the price is set to 0. This means that in our dataset every row that has a price of 0 is an auction that is either for free or an exchange auction. Afterwards it will append the dictionary to the previously created list.



```
scraped: https://www.olx.pl/oferty/q-nintendo-switch/?page=19
scraped: https://www.olx.pl/oferty/q-nintendo-switch/?page=20
scraped: https://www.olx.pl/oferty/q-nintendo-switch/?page=21
scraped: https://www.olx.pl/oferty/q-nintendo-switch/?page=22
scraped: https://www.olx.pl/oferty/q-nintendo-switch/?page=23
scraped: https://www.olx.pl/oferty/q-nintendo-switch/?page=24
scraped: https://www.olx.pl/oferty/q-nintendo-switch/?page=25
```

```
l
```

```
df=pd.DataFrame(l)
```

```
df
```

	title	link	price
0	Nintendo switch	https://www.olx.pl/oferta/nintendo-switch-CID9...	1400
1	Nintendo Switch Lite 20 gier	https://www.olx.pl/oferta/nintendo-switch-lite...	899
2	Ps4 Playstation4 1216 1 TB zestaw pad , gra Z...	https://www.olx.pl/oferta/ps4-playstation4-121...	770
3	Nintendo Switch V2 Gwarancja	https://www.olx.pl/oferta/nintendo-switch-v2-g...	1169
4	Gry Nintendo Switch Final Fantasy Minecraft Su...	https://www.olx.pl/oferta/gry-nintendo-switch-...	0
...	...	...	...
1094	Joy con nintendo switch	https://www.olx.pl/oferta/joy-con-nintendo-swi...	199
1095	Gry na Nintendo Switch	https://www.olx.pl/oferta/gry-na-nintendo-swit...	89
1096	Gra Nintendo Switch DOOM	https://www.olx.pl/oferta/gra-nintendo-switch-...	90
1097	Gra Nintendo Switch Mario Tennis Aces	https://www.olx.pl/oferta/gra-nintendo-switch-...	100
1098	Gra Nintendo Switch ARMS	https://www.olx.pl/oferta/gra-nintendo-switch-...	100

1099 rows × 3 columns

```
#df.to_csv('output.csv')
```

Figure 21. Output from Jupyter Notebook, same code.

This is what the output looks like where it prints which page the program is currently scraping. Below that, using Pandas library I converted the list into a data frame and displayed it. After that it can be saved to a *csv* format file, or you can conduct further analysis in Pandas.

2.2.4. CAPTCHA Bypass (Selenium)

Here is an attempt at bypassing captcha that used to work, which is why I will not delve deep into the matter and describe the code in detail, because this approach is depreciated. The bypass was a simple cookie dump and load using Selenium, pickle, and time packages. The way it worked is that once I performed the captcha check I dumped my cookies and saved them to a file. Later I could open the site in Selenium and when prompted with a captcha it would load cookies and refresh, then grab the site code and quit. The problem with this is how often cookies expire and that it is very slow. Generally, it is best to avoid Selenium when scraping, because of the graphical interface that takes times to load. The output and the code are available in Appendixes (1) and (2), respectively.

III. Usability of scraping in business

3.1. Scraping as a tool

The power of scraping usually comes down to how you decide to use it and what for. Scraping can be something extremely helpful or something completely useless, depending on the case and the user. At the end of the day, web scraping is mostly a tool used for obtaining data and the benefit you get from acquiring said data mostly comes down to how you use it, unless you decide to set up a business around scraping data for other businesses.

When it comes to what you want to scrape the sky is the limit – or the internet is. Scraping can get a mishmash of everything and help you in different sectors, as various needs will require various methods. Scraping can prove useful in analyzing markets and researching, for example, when commercial databases and censuses focus on big apartments only, they create a gap in the data. This was addresses by scraping eleven million Craigslist listings, which was later useful in further analysis (Boeing & Waddell, 2017). There are a lot of places for scraping to take place in research and academics and AI. This can be, but is not limited to, as an example studies that focused on finding key drivers on how passengers decide whether to recommend the airlines or not. A prediction model was built to extract those characteristics using logistical regression (Tansitpong, 2020). All of this powered by web scraping. On top of that, web scraping can be used to see patterns across the internet. Like scraping GDPR consent pop-ups and analyzing them. It showed that only 11.8% of sites actually met the requirements of EU laws and that removing opt-out buttons increased consent by 22-23 percent points (Nouwens, et al., 2020).

While this is all interesting this does not really have anything to do with this project, or this paper – nevertheless, it shows one important thing, and it is the flexibility of scraping. Web Scraping can be tweaked in many various ways to achieve desired results, thus a perfect tool for business.

3.2. From tool to benefit

Because of the fact that nowadays data is an asset for every type of business, in the ever-growing era of digitalization, scraping provides a source of a sustainable competitive advantage. In business scraping is becoming a constant flow of data, which can help in gaining more intel to make better informed decisions and gain an edge above others. There is a lot of data on the internet which is waiting to be scraped and organized. Today scraping is not talked about a lot in business, most likely due to its grey area nature. There are many places to establish a new business or to improve something that already exists based on scraping. Among the most popular today are ("The Ultimate Guide to Web Scraping for Business - Economalytics", 2019):

- Competitor Analysis – If you want to excel as a company you need to analyze your competitors, news about them and their prices. Find their strategies and adjust your own to one up the competitor. Extract prices regularly from multiple sources. Helps if you want to enter a market anew as well as when already on the market to undercut the competition. Provides valuable information and can examine their brand presence and status in the market.
- Price & Product Intelligence – Discovering the perfect price and product to sell, its hard to keep track of all prices when they change rapidly. Enrich your models with data and discover which price is the best and which product to sell;
- Lead Generation – Getting fresh leads can be troublesome at times, scraping can help by visiting various places on the internet and gathering contact data, you can also automate the process and the program can contact leads on its own too;
- Dynamic Pricing – Extract data to gain knowledge about how to price and sell better deals to prospects. This can be done instantly and right away. Informing the prospect of the value received on investment. Judged by demand, availability and competitors. Returns and estimate or a forecast[14];
- PR & Brand presence – Gather all posts and places where your company is mentioned and produce an analysis based on the sentiment in real-time. Identify the state of your brand and prevent damage by fraudulent, untruthful or hateful messages;

[14] Retrieved 01 July 2021, from https://limeproxies.netlify.app/blog/web-scraping-use-cases#web

- Business Intelligence – fill in the gaps of missing data, feed the models by newly acquired big data forums;

and more ("25 Ways to Grow Your Business with Web Scraping", 2021).

As presented, there are many ways of improving businesses, this is not talked about extensively yet and the subject lacks real web scraping use cases, regardless of that the tool can find many uses among different kind of businesses, though the most value it brings is to e-commerce and marketing. E-commerce is a sector that is affected by scraping the most, due to the data that can be scraped. In E-commerce businesses always need to be wary of what the other players in the market are doing, what are their prices, they need to be alerted of changes, strategies and monitor their competitors. From simple price undercutting on e-commerce platforms to complicated analysis of multiple steps. Being a step ahead is important and scraping is something that can bring an final competitive edge to the e-commerce market.

Another way of profiting off of the new technology is to create a B2B business that takes care of the processes mentioned for another business. A lot of the sites that provide information for this paper are sites that are providing scraping solutions. In Poland there are no big services (if any) like that which makes it a good place to start with for developing a startup based around this technology.

3.3. My real-life use case

3.3.1. Origin

This project and paper have pretty much originated from this very life application where it was needed. I buy stuff from OLX pretty frequently and usually it is electronics. Most of the time when I buy electronics from OLX, I use them for a while, and sell them for either the same price or even a higher price. I did this couple of times with Logitech wheels, consoles, and other computer accessories. It seemed to me that people did not care for a slight increase in price. With that reasoning I thought that maybe if I negotiated for a lower price and then sold it for a bit more than usual, I could get higher margin, a bigger difference between those prices.

I came up with an idea to start 'flipping' consoles on a used market for profit. The problem was finding the best offers, keeping track of everything and analyzing/finding the market. The purpose of this is to create a program that can aid in such actions and keep trach of everything in one place – what I have prepared is just a beginning or a part of a bigger project.

At some point I decided to go with Nintendo Switch as my product to resell, because of its stable price – a conclusion that I have made after some time, after browsing through the site for a couple of months. This is why in the code explanation part I was scraping Nintendo Switches from OLX, because this is what I used the scraper for.

3.3.2. The market & analysis

After couple successful tries at reselling this I decided to dig deeper and write a scraper in order to analyze the market a bit more. The most important part was to analyze the market and figure out why it is as it is and how to take advantage of it. I scraped a dataset from OLX (Figure 21) and plotted a distribution of price in R (Source code in Appendix 3).

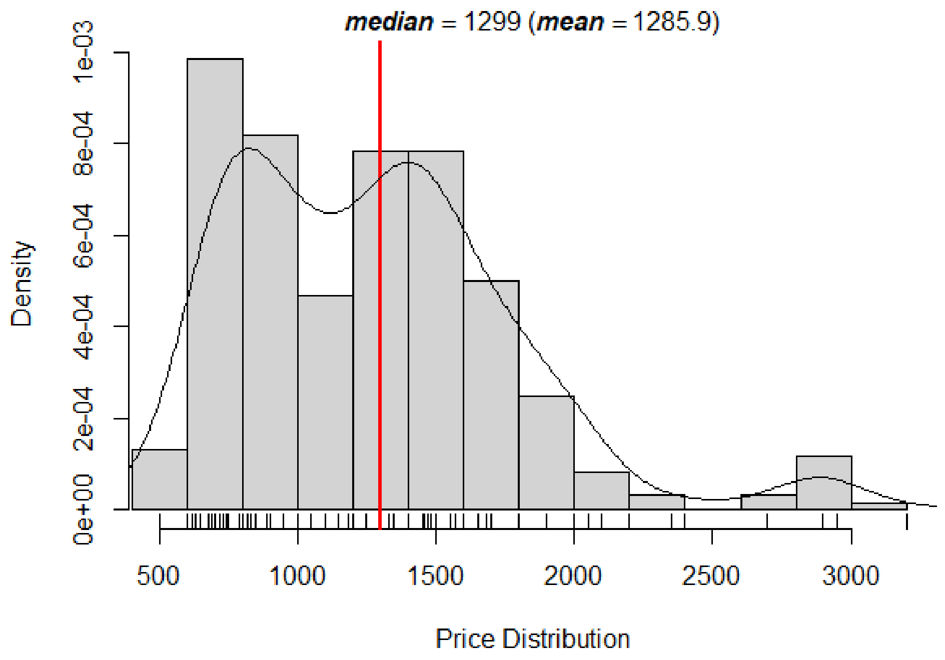


Figure 22. Price distribution of Nintendo Switch consoles on OLX.

Years after release the console still holds its price at around 1300. There are a lot of consoles priced at 900 and some at 800 too. Those consoles are usually just a raw console without any accessories and the ones going for 1200 or more are full sets without games. Consoles that are pricing above this range are sets with all accessories and some games. There are cases where a console is some limited edition, and it may be more expensive. However, in my time doing this I have already seen all versions in every price range it is just a matter of finding the offer.

There is also a distinction between console versions:

- V1 – older, worse battery-life; can be modified (using CFW - custom firmware) to play games for free. In simpler words, allows to pirate games without altering the hardware, only using a software backdoor that exists in recovery mode (free, complicated but possible).
- V2 – newer, better battery-life; the backdoor is fixed in this version, meaning that in order to pirate you need to install a hardware chip (expensive, hard).

This distinction creates a new market among this Nintendo console market, since V1 consoles with the backdoor are not produced anymore – they are only losing in numbers. In basic economic terms, the supply is decreasing and the demand either stays the same or increases, due to extreme game prices. While there is no matter if demand is the same or if it increases, the latter signifies an even faster price growth of those consoles.

Obviously, this market exists because people want to save money (by illegal pirating). Nintendo Switch games cost around 160 to 200 PLN, a V1 console can be modified for 200 to 400 PLN. A V2 console can be modified for 600 to 800 PLN. Either way the cost of modifying the console equals to just a couple of games. The V1s here are ‘the good’ product due to the fact that modifying them does not cost anything in reality, it only requires technical knowledge and time, on top of that they are becoming rarer and rarer.

3.3.3. The Benefit

The benefit in this case comes from the ease of reselling. Even though, there are clear reasons for the two groups in the price distribution to exist, it still allows cycling items through both groups and gaining benefit from that resell. Let us dig deeper in how I tackled the reselling processes.

- Phase 1 – 11-25% R.O.I.:

Initially, when I began reselling, I bought consoles for around 900 PLN and sold them for 1000-1100 PLN. A lot of this varied in terms of what accessories offers had, sometimes I had to buy some accessories to sell for a higher price. This way could provide from 100 to 225 PLN of net profit. This is a very basic way, and it mostly focuses on the ability of finding a good offer, negotiating and selling for more. The scraper helps in finding a range of products by doing some data wrangling on a scraped dataset.

- Phase 2 – 20-50% R.O.I.:

This phase focused on buying V1s for cheap for people that were unaware of this whole phenomenon and selling them to people who wanted to use them for installing CFW. This allowed me to buy consoles for cheaper (accessories not needed) and sell them for a bit more or the same price. This is a more optimal way, because it lets you buy for cheaper and requires less hassle over the accessories since the point of interest is a modifiable console.

- Phase 3 – 30-50% R.O.I.:

This part can achieve such a high return on investment, because in this case I actually install the CFW myself and sell the console to people who want to buy an already modified and ready-to-use console. My highest profit I had from this was by striking a deal and buying a console with all accessories and an additional pad, but missing joy cons (those are attachable pads to the console which are essential for playing) for 750 PLN. I bought the joy cons for 220 (shipping incl.), installed CFW and sold a full set for 1500, giving me exact 50% on return.

This phase has a little bit of cheating in it, because it requires installing CFW which is not strictly connected to reselling or the phenomenon found by scraping, yet I decided to include it too, because the whole analysis has led me up to this point.

- Sidenote:

All of these prices will vary based on shipping costs. Installing CFWs is not illegal, only installing pirated copies of games is.

3.4. Future plans & improvements

This is a basic version of a project that can be extended into something much more, much bigger. There are many ways in which I would like to improve the project and build it into an actual ready-to-use real project. Projects like, later on, can easily be tweaked towards different needs and pitched to customers that require those needs, because of the versatility and a template. Once a project like this is done, we have a template that only requires a change of variables and a bit of writing to make a new different project. My future plans for this are to make a ready product and try to offer a business a solution for their company. As to what can be improved, I will go over in detail from most important (meaning, what has to be done first) to more complicated, fancy solutions. I am also planning on doing a sentiment analysis on some company solely powered by scraping (either social media posts or reviews), but that comes after this project is finished.

3.4.1. Database & automation

First thing to do, which also is not too complicated is to extend the data that is scraped. Obviously, what data is needed varies from project to project, but here we know what we want. For this project, we could extend the dataset simply by automating the scraper to run every day at certain hours (perhaps at different hours to make it less detectable). This will give us historical data with time & date, although just date would probably be sufficient. When the scraper is automated and runs every day, when it saves the output, it can index the current date and append into either a data frame or a json database. You can make it run every day using schedule library and hosting the script on AWS Amazon, or any service that runs 24/7. For example, at my home I have set up a NAS QNAP server which runs all the time (even when the power goes out) and the scraper can be hosted there on small resource-free Linux virtual machine. This can be done on both BeautifulSoup4[15] and Scrapy[16] as it is an external solution.

[15] Retrieved 11 July 2021, from https://coderspacket.com/quote-scraper-in-python-using-beautifulsoup
[16] Retrieved 11 July 2021, from https://stackoverflow.com/questions/44228851/scrapy-on-a-schedule

On top of that there also needs to be a database that the outputs are constantly saved to and appended. My solution of choice is to make a database in MongoDB and combine it with Scrapy[17] as this has already been covered on the internet quite a lot. As for Beautiful Soup the best way is to probably go for SQLite and create a database there using sqlite3 library from Python, although any database package that exists in Python as a library should work.

3.4.2. Frontend development

This one is a bit tricky due to the nature of scraping. I always wonder whether it is actually beneficial to develop a frontend for users or to just provide them with a database filled with their desired data. A frontend makes sense if there are multiple choices to make for the scraper, or just to have a fancy UI to flash to customers.

There is situation where a frontend is very useful and provides a system for people that have not acquired technical skills in Python or in R to move around with datasets freely. That situation is when the data scraped is formed in such a way that you can provide a real-time frontend which after a button click scrapes the data and provides the user with interactive plots and diagrams. A frontend like this can be done in Seaborn and Streamlit[18] and is an excellent tool that saves time for the developer as well as the hassle for users. The UI can be tweaked in order to fit the desires of users or like in my case – provide a real-time monitoring system of prices and listings scraped (sort of like stock market graphs).

I believe that this is a great feature to include in such a project, but it can also turn out to be something of a cool gadget that in the end does not provide any significance to the bigger scheme of things. It all depends on what type of data is being scraped and what it will be used for. In my case it would be a cool feature, but it is not something extremely important as new listings are registered to the database and the price distribution does not change dynamically, thus there is no bigger need for an UI, nevertheless a nice addition.

[17] Retrieved 11 July 2021, from https://realpython.com/web-scraping-with-scrapy-and-mongodb/
[18] Retrieved 11 July 2021, from https://python.plainenglish.io/how-to-build-a-streamlit-app-to-scrape-github-profiles-f36d41fb98c
and from https://github.com/ravigoel08/Streamlit-MBScraper

3.4.3. Recommendation & Alert system

The ultimate goal of a project of this scope would be to implement some sort of AI powered recommendation system that would cut down on the time needed to look through the database or completely automate the process and provide the user with best offers. The problem with a recommendation system that could be used in my case is that there are no available models that could do that. A model can only work if it is well populated, fed data and has enough features to make good decisions. In case of my data that could only be pictures of consoles, their price, location and title. In hindsight, this will never be enough to have a model that has an accuracy that could be considered 'enough'. For me enough would be 75% and above accuracy, optimal is around 90%. The model will not have enough features because recommendation models usually are run by content[19] or collaborative[20] filtering(Afkhamizadeh et al., 2021).

Unfortunately, the recommendation system cannot exist in my case although in different cases there are many algorithms to choose from to perform different actions on datasets that may be useful. The world does not end here. In order to substitute this system, I decided to go for an alert system that will notify me of any new listing in a certain price range (800-1200 PLN). Alert systems like these can work by notifying by slack, email, messenger or even SMS services. However, I chose to use a Discord[21] solution, because it lets me create an entire environment with different channels for different notifications[22]. This solution is not perfect as it requires to go into every new offer whenever it is published and look at it. In this case it is not much of a problem due to low frequency of added listings but in different scenarios that could be impossible.

3.4.4. Auto message

Connected to the previous feature which alerts of every new listing, this feature could be quite useful when it comes to finding the correct product. In my case finding a V1 Nintendo Switch is crucial. Distinguishing a console between a hackable V1 and a new V2

[19] Focuses on item's properties. Measured by similarity in those properties.
[20] Focuses on relationship between users and their ratings.
[21] Retrieved 11 July 2021, from https://discord.com/
[22] Retrieved 11 July 2021, from https://medium.com/@maxwellflitton/free-python-alert-systems-with-discord-6b04d314455c

is as simple as writing the serial number of the console into a site[23] and then scraping the result. After inputting the serial number on the site, the checker gives us a result. If the switch is patched there is a red alert box, if it is not there is a green one.

The auto messaging bot would be quite useful when it comes to being first at asking people for the serial number. After the person behind the offer responds with the key, the scraping and messaging bot can input the serial number and check for the result – if the box is green alert me on Discord, if the box is red do nothing. After that I can proceed to negotiations and purchasing of the console. As we know, reselling V1 consoles provides the highest profit, therefore this addition might become crucial and be the first one to implement.

Final Conclusions

Web Scraping is the next big tool in our digital society, the fuel for Artificial Intelligence models and a gap-filler for big data institutions. As it grows it is optimal to grow with the technology and scale. The tool is flexible and can be used for many different things according to the likes of the developer. However, the sector that benefits the most is e-commerce, and this is where scraping usually starts (monitoring products, prices and changes). The technology is not popularized as much as it should be, and it seems that it is being overlooked for the time being (due to lack of resources about the topic). As more businesses gravitate towards this technology this should be subject to change. Scraping is also in a pretty weird spot when it comes to the law. Recently, Artificial Intelligence has seen some first regulations, because of the fact that it is a powerful tool in a grey area. Scraping is in a similar spot, except that it has not been popularized yet.

Web Scraping has many applications in business, academics, analytics and as a tool for harvesting data. The paper has shown examples of the technology in use through academic papers and through my own examples. I believe that it successfully shows technical skills and an ability to use them in the real world in a practical way. On top of that, it provides a multitude of technical explanations for people that would like to take the journey and start learning about web scraping and scrapers.

[23] https://ismyswitchpatched.com

The project is far from over, but it let me achieve sufficient results when it comes to benefit and to what I have learned during the development. I am aware of the project's shortcomings, its upsides and future possibilities. This project has let me put my foot in the door when it comes to the world of web scraping. It opened up many possibilities and many roads to choose from. I have a pretty good understanding of what is going on behind the scenes of web scraping, but I am also aware that what I did are only first steps. The technology proved useful, and I believe that it has many applications in business. My ultimate goal is to try and find a customer for any of the scraping services mentioned in the paper, starting from e-commerce services like Allegro and OLX. If I manage to achieve something I will create a company providing IT solutions under the it.waw.pl domain which I own (currently offline). I would like to start from polish customers and base in Warsaw. If anything is to succeed a perfect plan of action would be to extend services and solutions over time that it would provide.

Bibliography

1. U.S. E-Commerce Sales 1999-2021. Marketplace Pulse. (2021). Retrieved 12 July 2021, from https://www.marketplacepulse.com/stats/us-ecommerce/us-e-commerce-sales-22.
2. E-commerce statistics for individuals. Ec.europa.eu. (2021). Retrieved 12 July 2021, from https://ec.europa.eu/eurostat/statistics-explained/index.php?title=E-commerce_statistics_for_individuals.
3. Fielding, Roy T.; Gettys, James; Mogul, Jeffrey C.; Nielsen, Henrik Frystyk; Masinter, Larry; Leach, Paul J.; Berners-Lee, Tim (June 1999). Hypertext Transfer Protocol – HTTP/1.1 Retrieved 2021-05-07
4. Bishop, Mike (February 2, 2021). "Hypertext Transfer Protocol Version 3 (HTTP/3)". tools.ietf.org. Retrieved 2021-05-07
5. Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, and Arthur Secret. 1994. The World-Wide Web. *Commun. ACM* 37, 8 (Aug. 1994), 76–82. DOI:https://doi.org/10.1145/179606.179671
6. Scrapy | A Fast and Powerful Scraping and Web Crawling Framework. Scrapy.org. (2021). Retrieved 3 May 2021, from https://scrapy.org/.
7. Supratik Moulik. January 01, 2020. Data as the New Currency—How Open Source Toolkits Have Made Labeled Data the Core Value in the AI Marketplace. DOI: https://doi.org/10.1016/j.acra.2019.09.016 Retrieved 2021-05-07
8. Sirisuriya, S. (2015). *A Comparative Study on Web Scraping*. Ir.kdu.ac.lk. Retrieved 7 May 2021, from http://ir.kdu.ac.lk/bitstream/handle/345/1051/com-059.pdf?sequence=1&isAllowed=y.

9. Manuel, J., & Santos, A. (2018). 41. Repositorio-aberto.up.pt. Retrieved 7 May 2021, from https://repositorio-aberto.up.pt/bitstream/10216/116510/2/296684.pdf.
10. Gode, S., Saurkar, A., & Pathare, K. (2018). An Overview On Web Scraping Techniques And Tools. Ijfrcsce.org. Retrieved 7 May 2021, from http://www.ijfrcsce.org/index.php/ijfrcsce/article/view/1529.
11. van Oostenrijk, A. (2004). Screen scraping web services. *Nijmegen, The Netherlands: Radboud University of Nijmegen, Department of Computer Science*
12. Raggett, D., Le Hors, A., & Jacobs, I. (1998). HTML 4.0 Specification. Immagic.com. Retrieved 7 May 2021, from https://www.immagic.com/eLibrary/ARCHIVES/SUPRSDED/W3C/W980424S.pdf.
13. *HTML & CSS - W3C*. W3.org. Retrieved 7 May 2021, from https://www.w3.org/standards/webdesign/htmlcss#whatcss.
14. Harris, P. (2018). What is HTML code?. Rosen Pub. Group, page 10.
15. Giovanni Grasso, Tim Furche, and Christian Schallhart. 2013. Effective web scraping with OXPath. In Proceedings of the 22nd International Conference on World Wide Web (WWW '13 Companion).
16. Association for Computing Machinery, New York, NY, USA, 23–26. DOI:https://doi.org/10.1145/2487788.2487796
17. Scrapy 2.5 documentation — Scrapy 2.5.0 documentation. docs.scrapy.org. (2021). Retrieved 3 May 2021, from https://docs.scrapy.org/en/latest/index.html.
18. Kumar Sourav and Debi Prasad Mishra. 2012. DDoS detection and defense: client termination approach. In Proceedings of the CUBE International Information Technology Conference (CUBE '12). Association for Computing Machinery, New York, NY, USA, 749–752. DOI:https://doi.org/10.1145/2381716.2381859
19. Marques P. et al., "Detecting Malicious Web Scraping Activity: A Study with Diverse Detectors," 2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC), 2018, pp. 269-278, doi: 10.1109/PRDC.2018.00049.
20. Jing Y., X. Wang, X. Xiao and G. Zhang, "NIS04-5: Defending Against Meek DDoS Attacks By IP Traceback-based Rate Limiting," IEEE Globecom 2006, 2006, pp. 1-5, doi: 10.1109/GLOCOM.2006.283.
21. Kirjazovas, V. (2021). Most Common HTTP Headers. Oxylabs.io. Retrieved 7 May 2021, from https://oxylabs.io/blog/5-key-http-headers-for-web-scraping.
22. Application Programming Interface. HubSpire. (2021). Retrieved 9 May 2021, from https://www.hubspire.com/resources/general/application-programming-interface/.
23. Hoffman, C. (2018). What Is an API?. How-To Geek. Retrieved 9 May 2021, from https://www.howtogeek.com/343877/what-is-an-api/.
24. Cz., J. (2021). JonasCz/How-To-Prevent-Scraping. GitHub. Retrieved 7 May 2021, from https://github.com/JonasCz/How-To-Prevent-Scraping.

25. Suhit Gupta, Gail Kaiser, David Neistadt, and Peter Grimm. 2003. DOM-based content extraction of HTML documents. In Proceedings of the 12th international conference on World Wide Web (WWW '03). Association for Computing Machinery, New York, NY, USA, 207–214. DOI:https://doi.org/10.1145/775152.775182
26. Instamotor Agrees To Pay Craigslist $31M, Stop Scraping - Law360. Law360.com. (2017). Retrieved 9 May 2021, from https://www.law360.com/articles/951532/instamotor-agrees-to-pay-craigslist-31m-stop-scraping.
27. Rahman, A. F. R., Alam, H., & Hartono, R. (2001, September). Content extraction from html documents. In 1st Int. Workshop on Web Document Analysis (WDA2001) (pp. 1-4).
28. Iyatiti Mokube and Michele Adams. 2007. Honeypots: concepts, approaches, and challenges. In Proceedings of the 45th annual southeast regional conference (ACM-SE 45). Association for Computing Machinery, New York, NY, USA, 321–326. DOI:https://doi.org/10.1145/1233341.1233399
29. L. Spitzner, "Honeypots: catching the insider threat," *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, 2003, pp. 170-179, doi: 10.1109/CSAC.2003.1254322.
30. "The reCAPTCHA Project – Carnegie Mellon University CyLab". www.cylab.cmu.edu. Archived from the original on 2017-10-27. Retrieved 7 May 2021.
31. L. von Ahn, M. Blum and J. Langford, "Telling Humans and Computers Apart Automatically", Communications of the ACM, vol. 47, no. 2, pp. 56-60, 2004.
32. Singh, V. P., & Pal, P. (2014). Survey of different types of CAPTCHA. International Journal of Computer Science and Information Technologies, 5(2), 2242-2245.
33. Nitirat Tanthavech and Apichaya Nimkoompai. 2019. CAPTCHA: Impact of Website Security on User Experience. In Proceedings of the 2019 4th International Conference on Intelligent Information Technology (ICIIT '19). Association for Computing Machinery, New York, NY, USA, 37–41. DOI:https://doi.org/10.1145/3321454.3321459
34. Michi Henning. 2007. API: Design Matters: Why changing APIs might become a criminal offense. Queue 5, 4 (May-June 2007), 24–36. DOI:https://doi.org/10.1145/1255421.1255422
35. Qinghua Lu, Liming Zhu, Len Bass, Xiwei Xu, Zhanwen Li, and Hiroshi Wada. 2013. Cloud API issues: an empirical study and impact. In Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures (QoSA '13). Association for Computing Machinery, New York, NY, USA, 23–32. DOI:https://doi.org/10.1145/2465478.2465481
36. Irvin Dongo, Yudith Cadinale, Ana Aguilera, Fabiola Martínez, Yuni Quintero, and Sergio Barrios. 2020. Web Scraping versus Twitter API: A Comparison for a Credibility Analysis. In Proceedings of the 22nd International Conference on Information Integration and Web-based Applications & Services (iiWAS '20). Association for Computing Machinery, New York, NY, USA, 263–273. DOI:https://doi.org/10.1145/3428757.3429104

37. Boeing, G., & Waddell, P. (2017). New Insights into Rental Housing Markets across the United States: Web Scraping and Analyzing Craigslist Rental Listings. Journal of Planning Education and Research, 37(4), 457–476. https://doi.org/10.1177/0739456X16664789
38. Praowpan Tansitpong. 2020. Identifying key drivers in airline recommendations using logistic regression from web scraping. In Proceedings of the 2020 the 3rd International Conference on Computers in Management and Business (ICCMB 2020). Association for Computing Machinery, New York, NY, USA, 112–116. DOI:https://doi.org/10.1145/3383845.3383870
39. Midas Nouwens, Ilaria Liccardi, Michael Veale, David Karger, and Lalana Kagal. 2020. Dark Patterns after the GDPR: Scraping Consent Pop-ups and Demonstrating their Influence. Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems. Association for Computing Machinery, New York, NY, USA, 1–13. DOI:https://doi.org/10.1145/3313831.3376321
40. Midas Nouwens, Ilaria Liccardi, Michael Veale, David Karger, and Lalana Kagal. 2020. Dark Patterns after the GDPR: Scraping Consent Pop-ups and Demonstrating their Influence. Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems. Association for Computing Machinery, New York, NY, USA, 1–13. DOI:https://doi.org/10.1145/3313831.3376321
41. The Ultimate Guide to Web Scraping for Business - Economalytics. Economalytics. (2019). Retrieved 1 July 2021, from http://economalytics.com/the-ultimate-guide-to-web-scraping-for-business/.
42. 25 Ways to Grow Your Business with Web Scraping. Octoparse.com. (2021). Retrieved 1 July 2021, from https://www.octoparse.com/blog/web-scraping-for-businesses#.
43. Afkhamizadeh, M., Avakov, A., & Takapoui, R. (2021). Automated Recommendation Systems. web.stanford.edu. Retrieved 11 July 2021, from https://web.stanford.edu/~takapoui/linear_bandits.pdf.

Internet sources:

1. https://www.crummy.com/software/BeautifulSoup/bs4/doc/
2. https://www.selenium.dev
3. https://www.selenium.dev/documentation/en/webdriver/
4. https://whatismybrowser.com
5. https://developers.google.com/recaptcha/docs/versions
6. https://www.w3schools.com/whatis/whatis_ajax.asp
7. https://www.scrapingdog.com
8. https://solvecaptcha.net
9. https://stackoverflow.com/questions/8049520/web-scraping-javascript-page-with-python
10. https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/
11. https://www.octoparse.com/blog/web-scraping-for-businesses#
12. https://medium.com/@maxwellflitton/free-python-alert-systems-with-discord-6b04d314455c
13. http://infolab.stanford.edu/~ullman/mmds/ch9.pdf
14. https://limeproxies.netlify.app/blog/web-scraping-use-cases#web

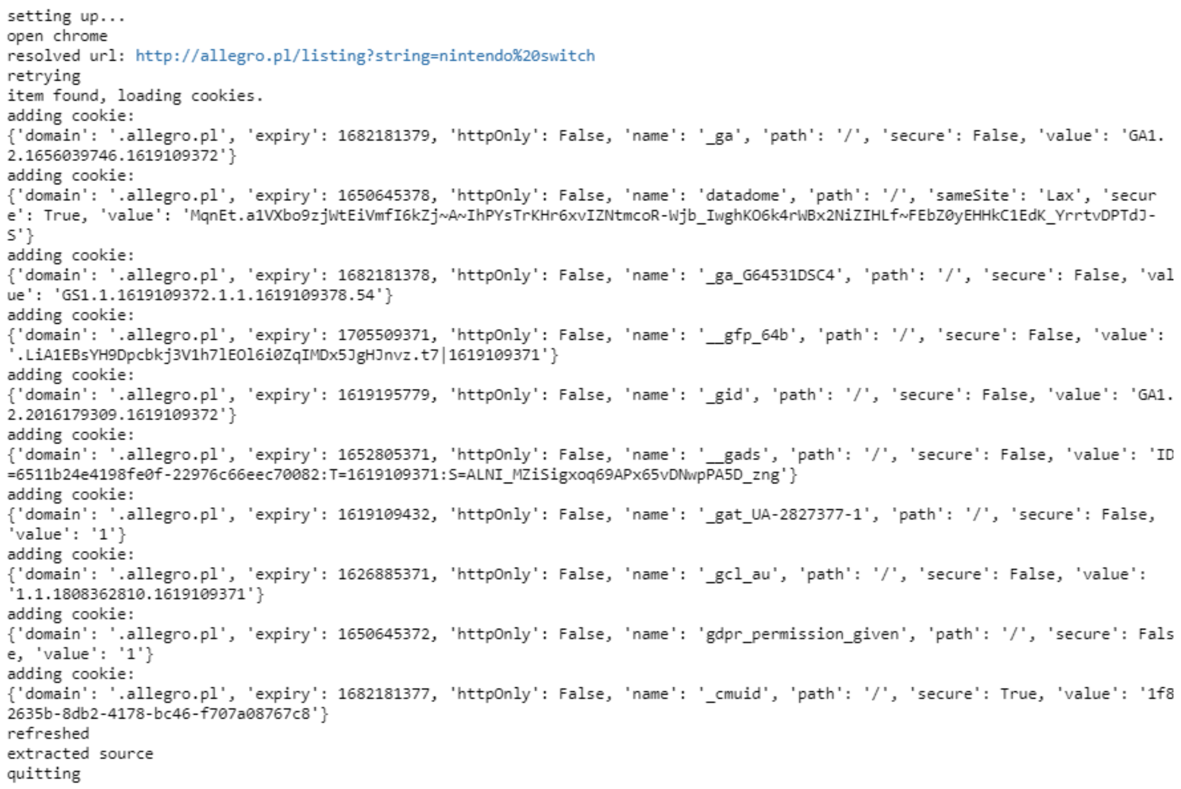
15. https://coderspacket.com/quote-scraper-in-python-using-beautifulsoup
16. https://stackoverflow.com/questions/44228851/scrapy-on-a-schedule
17. https://realpython.com/web-scraping-with-scrapy-and-mongodb/
18. https://python.plainenglish.io/how-to-build-a-streamlit-app-to-scrape-github-profiles-f36d41fb98c
19. https://github.com/ravigoel08/Streamlit-MBScraper
20. https://discord.com/
21. https://medium.com/@maxwellflitton/free-python-alert-systems-with-discord-6b04d314455c
22. https://ismyswitchpatched.com

List of tables

Table of figures

Appendixes

Appendix 1



```
setting up...
open chrome
resolved url: http://allegro.pl/listing?string=nintendo%20switch
retrying
item found, loading cookies.
adding cookie:
{'domain': '.allegro.pl', 'expiry': 1682181379, 'httpOnly': False, 'name': '_ga', 'path': '/', 'secure': False, 'value': 'GA1.
2.1656039746.1619109372'}
adding cookie:
{'domain': '.allegro.pl', 'expiry': 1650645378, 'httpOnly': False, 'name': 'datadome', 'path': '/', 'sameSite': 'Lax', 'secur
e': True, 'value': 'MqnEt.a1VXbo9zjWtEiVmfI6kZj~A~IhPYsTrKHr6xvIZNtmcoR-Wjb_IwghKO6k4rWBx2NiZIHLf~FEbZ0yEHHkC1EdK_YrrtvDPTdJ-
S'}
adding cookie:
{'domain': '.allegro.pl', 'expiry': 1682181378, 'httpOnly': False, 'name': '_ga_G64531DSC4', 'path': '/', 'secure': False, 'val
ue': 'GS1.1.1619109372.1.1.1619109378.54'}
adding cookie:
{'domain': '.allegro.pl', 'expiry': 1705509371, 'httpOnly': False, 'name': '__gfp_64b', 'path': '/', 'secure': False, 'value':
'.LiA1EBsYH9Dpcbkj3V1h7lEOl6i0ZqIMDx5JgHJnvz.t7|1619109371'}
adding cookie:
{'domain': '.allegro.pl', 'expiry': 1619195779, 'httpOnly': False, 'name': '_gid', 'path': '/', 'secure': False, 'value': 'GA1.
2.2016179309.1619109372'}
adding cookie:
{'domain': '.allegro.pl', 'expiry': 1652805371, 'httpOnly': False, 'name': '__gads', 'path': '/', 'secure': False, 'value': 'ID
=6511b24e4198fe0f-22976c66eec70082:T=1619109371:S=ALNI_MZiSigxoq69APx65vDNwpPA5D_zng'}
adding cookie:
{'domain': '.allegro.pl', 'expiry': 1619109432, 'httpOnly': False, 'name': '_gat_UA-2827377-1', 'path': '/', 'secure': False,
'value': '1'}
adding cookie:
{'domain': '.allegro.pl', 'expiry': 1626885371, 'httpOnly': False, 'name': '_gcl_au', 'path': '/', 'secure': False, 'value':
'1.1.1808362810.1619109371'}
adding cookie:
{'domain': '.allegro.pl', 'expiry': 1650645372, 'httpOnly': False, 'name': 'gdpr_permission_given', 'path': '/', 'secure': Fals
e, 'value': '1'}
adding cookie:
{'domain': '.allegro.pl', 'expiry': 1682181377, 'httpOnly': False, 'name': '_cmuid', 'path': '/', 'secure': True, 'value': '1f8
2635b-8db2-4178-bc46-f707a08767c8'}
refreshed
extracted source
quitting
```

Appendix 2

```python
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import pickle
import time
print('setting up...')
DRIVER_PATH = './chromedriver'
URL = 'http://allegro.pl/listing?string=nintendo%20switch'
options = Options()
options.headless = False
options.add_argument("--window-size=1280,720")
options.add_argument("--enable-javascript")
print('open chrome')
driver = webdriver.Chrome(options=options, executable_path=DRIVER_PATH)
driver.get(URL)
print('resolved url: '+URL)
try:
    print('retrying...')
    element = WebDriverWait(driver, 15).until(
        EC.visibility_of_element_located((By.ID, "captcha-container"))
    )
except Exception:
    pass
finally:
    print('item found, loading cookies.')
    cookies = pickle.load(open("cookies.pkl", "rb"))
    for cookie in cookies:
        driver.add_cookie(cookie)
        print(f'adding cookie: \n{cookie}')
    time.sleep(6)
    driver.refresh()
    print('refreshed')
    time.sleep(5)
    html_page = driver.page_source
    print('extracted source')
    time.sleep(2)
    print('quitting')
    driver.quit()
```

Appendix 3

```r
library(readr)
library(tidyverse)
library(psych)

df <- read_csv('output.csv')
df2 <- filter(df, price>=500)

x = df2$price
hist(x, main = mtext(bquote(
  paste(bolditalic("median") == .(round(median(x),1)), " (",
        bolditalic("mean") == .(round(mean(x),1)), ")")
)), xlab = "Price Distribution", xlim=range(x), prob=TRUE)
rug(x)
abline(v = median(x), col = "red", lwd=2)
lines(density(x))

summary(x)
describe(x)
```

Student’s statement of authorship

informing that the student has developed their bachelor’s dissertation unassisted

Warsaw, ...

Name and surname: ....

Student ID no.: ....

STATEMENT OF AUTHORSHIP

I hereby declare that the bachelor’s dissertation entitled

....

....

....

submitted at Kozminski University, has been written by me alone, unassisted, and has never served as the basis for any official procedure involving taking steps leading to obtaining a higher education diploma confirming the conferment of an academic degree.

I also declare that my master’s dissertation does not infringe any copyright within the meaning of the act of 4 February 1994 on copyright and related rights (Journal of Laws of the Republic of Poland of 2019, item 1231) or moral rights protected under the law.

At the same time, I acknowledge that the master’s dissertation will be checked using a plagiarism detection system integrated with a National repository of written theses and dissertations.

/Student’s signature/

Supervisor’s declaration

Warsaw, ...

Supervisor’s name and surname: ....

DECLARATION

I hereby declare that the bachelor’s dissertation of student ....
(student ID no.) .... , entitled
....
....
....,

has been developed under my supervision.

At the same time, I acknowledge that the bachelor’s dissertation will be checked using a plagiarism detection system integrated with a National repository of written theses and dissertations.

/Supervisor’s signature/