

Long Tran

DATA SCRAPING APPLICATION WITH SCRAPY

Bachelor's thesis

Bachelor of Engineering

Information Technology

2023



South-Eastern Finland
University of Applied Sciences

Degree title	Bachelor of Engineering
Author(s)	Long Tran
Thesis title	Data scraping application with Scrapy
Commissioned by	SPORTPRO Co. Ltd
Year	2023
Pages	49 pages, 5 pages of appendices
Supervisor(s)	Timo Mynttinen

ABSTRACT

Because of the popularity of data nowadays, many businesses and companies are interested in any means of collecting data. The objective of the thesis project was to research and implement a robust solution for gathering and extracting data from any website.

The thesis utilized Scrapy, a Python web scraping framework, to solve the mentioned problem. To study and research the functionality of Scrapy, this project attempted to scrape Amazon, one of the largest e-commerce websites. The paper also briefly introduced other available frameworks for scraping. Moreover, the scraping policy and other related problems were discussed as well.

The project successfully retrieved various pieces of information from Amazon and exported the data into several formats. In addition, this study has been a great help to understand many different elements of Scrapy. In future projects, the commission could take advantage of this technology to learn more about their customer needs and behaviors.

Keywords: Scrapy, Python, data, website, scraping

CONTENTS

1	INTRODUCTION	1
2	DATA SCRAPING DEFINITION	2
3	DATA SCRAPING FUNDAMENTALS	4
3.1	Hypertext Markup Language	5
3.2	Document Object Model	8
3.3	Document Object Model selector	9
3.3.1	XPath selector	10
3.3.2	CSS selector	12
3.4	Scraping framework	13
3.4.1	Scrapy	13
3.4.2	Selenium and Beautiful Soup	17
3.5	Programming language Python	18
3.6	Legal issues and ethical scraping	19
4	DATA SCRAPING APPLICATION WITH SCRAPY	20
4.1	Prerequisite	21
4.2	Data extraction	24
4.2.1	Navigating through Amazon	24
4.2.2	Extracting product information	30
4.2.3	Scraping multiple products	33
4.3	Data transformation	35
4.4	Data loading	37
4.4.1	Command-line method	38
4.4.2	Programming method	38
4.4.3	Export result	41
4.5	Optional settings	42

5 CONCLUSION.....	43
REFERENCES	45
LIST OF TABLES	47
LIST OF FIGURES	48

1 INTRODUCTION

Nowadays, data has always played an important role in many different fields. For example, patient data assists healthcare providers to reduce errors, lower readmission fees, gain better insight into illness treatment and causes of diseases in healthcare (TEG Analytics, n.d.). There are more varieties of data in the education field such as assessments, attendance, participation, grades, etc. These data help teachers evaluate students more efficiently and deliver suitable teaching methods (Learning A-Z, n.d.). In the retail industry, sales information can be used to research consumer behavior. Therefore, businesses can utilize this to attract more customers and enhance their overall shopping experience. These are only one of the few benefits that data has brought to those subjects.

With the rapid growth in data usage and the rise of big data, companies have attempted to collect data by using several methods or even creating their own procedures to extract data from the desired source. A well-known approach for businesses to gather information from a website is data scraping because there is a significant amount of information people can find on the Internet nowadays. Scrapy is one of the most popular web scraping frameworks on the market, considering its high scalability and flexibility. Therefore, it technically could compile information from any website from small to considerable volumes of data. Moreover, it could emulate an ETL (Extract-Transform-Load) process, which basically extracts and transforms the data into a readable material, then stores the data in the desired format.

The objective of the thesis is to study the functionality of Scrapy by attempting to scrape the Amazon website. This paper will go through the principles of data scraping from understanding the core concept to developing a highly optimized scraper by combining Scrapy with Python programming. This thesis also focuses on experimenting with various settings of Scrapy to learn as many elements of this framework as possible.

2 DATA SCRAPING DEFINITION

Data scraping (also known as web scraping, web harvesting, or even web data extraction) is a process of collecting information that people can view from a website. Another popular term that people usually hear in web scraping is web crawling. Crawling is an action that specifies a web scraper having the ability to navigate through different pages and links on a website. This is an important factor in any data scraping project because the information is often distributed on different pages on the website. The most obvious scraping action is manually copying and pasting the online material into a text file or a spreadsheet such as Microsoft Excel. While the user technically can manually extract the data, data scraping nowadays is usually referred as a bot that executes an ETL (Extract-Transform-Load) process.

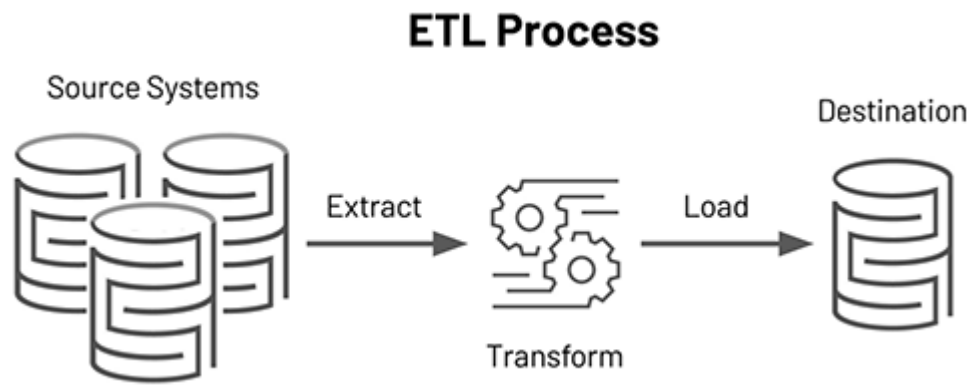


Figure 1. Illustration of ETL process. Extract Transform Load (ETL). Databrick. n.d. Available at <https://www.databricks.com/glossary/extract-transform-load>. Accessed April 2023

Figure 1 illustrates a typical ETL pipeline process. ETL pipeline, or data pipeline, is a crucial term in the data industry. It stands for extract, transform, and load which is a 3-phase data processing process. The details of each phase can be found on Amazon Web Services (Amazon Web Services, n.d.):

- Data extraction (Extract): The first step of the process is pulling raw data from one source or various sources. Data typically comes from SQL or NoSQL servers, CRM (Customer Relationship Management) and ERP (Enterprise Resource Planning), data storing format (XML, JSON, spreadsheet), and from web pages. Correspondingly, the abstracted data will be stored in a staging area. This area is a temporary storage for the transform phase.

- Data transformation (Transform): This phase will transform the raw data from the extraction phase to match the requirement of the user and deliver it to the target format. These are the techniques that have many use cases:
 - Deduplication identifies and removes duplicate records
 - Validation excludes unwanted information from the dataset
 - Format revision converts data into the desired format. Converting temperature from Fahrenheit (°F) to Celsius (°C), currency from euro (€) to pound (£) or even dollars (\$) are some examples
 - Derivation calculates new values from the existing values
 - Joining connects the same data from different data sources to create a relationship between them
 - Splitting separates a column or data attribute into multiple columns in the target source. This will help to categorize data more dedicatedly
 - Summarization improves data quality by reducing a large number of data values into a smaller dataset
 - Encryption protects the collected data by adding encryption before delivering the data to the target format

- Data loading (Load): The final phase of the process is to move the transformed data from the staging area into the target format. The whole process is typically automated from beginning to end.

Web scraping technically functions in the same manner as an ETL pipeline.

Scrapers collect information on the website and then store the data in temporary storage. Finally, the engine will distribute the scraped materials into the target format such as database, JSON, XML, or even a spreadsheet. Therefore, this is a huge advantage for organizations because ETL provides better data management and high customization for their own use cases. However, building pipelines that ensure high data reliability is extremely complex and requires a lot of optimizations from the engineers because of the changeability coming from the data itself, regardless of the source.

Occasionally, the manual copy-paste method can be better than modern scraping technologies because many websites prevent machine automation. Additionally, setting up a scraper engine practically requires some effort from the user.

However, a human-based operation is strictly limited when working with a large amount of data. For instance, the scraper wants to find some questions about data scraping on Stack Overflow, a popular forum for technology content.

The screenshot shows the Stack Overflow search results page for the query 'data scraping'. The page header includes the Stack Overflow logo, navigation links for 'About', 'Products', and 'For Teams', and a search bar containing the text 'data scraping'. On the left sidebar, there are navigation options: 'Home', 'PUBLIC' (with 'Questions' selected), 'Tags', 'Users', 'Companies', 'COLLECTIVES' (with 'Explore Collectives'), and 'TEAMS' (with 'Stack Overflow for Teams'). The main content area is titled 'Search Results' and shows 'Results for data scraping' with 'Search options not deleted'. It indicates '500 results' and provides sorting options: 'Relevance', 'Newest', and 'More'. Two search results are visible:

- Result 1:** 'RegEx match open tags except XHTML self-contained tags' with 3531 votes. The snippet reads: 'If you have a small set of HTML pages that you want to scrape data from and then stuff into a database, regexes might work fine. ...'. It includes tags for 'html', 'regex', and 'xhtml' and is attributed to 'Community wiki Kaitin Duck Sherwood'.
- Result 2:** 'How do I prevent site scraping? [closed]' with 330 votes, 26 answers, and 132k views. The snippet reads: 'I've been noticing other music sites scraping our site's data (I enter dummy Artist names here and then do google searches for them). How can I prevent screen scraping? ...'. It includes tags for 'html', 'web-scraping', 'architecture', and 'piracy-prevention' and is attributed to 'pixel 3,449 asked Jul 1, 2010 at 20:49'.

Figure 2. Stack Overflow example #1

Figure 2 shows the search result for the data scraping topic on Stack Overflow. There are approximately 500 results on this topic and this could take a lot of time if the scraper decides to collect this the traditional way. In contrast, web scraping not only provides us a consistent way to extract the data from the site but also delivers better data quality. Furthermore, when pairing with a scheduler, it is ensured that the information is kept up to date at all times.

In short, data scraping is a process that performs scraping operations on web pages with high flexibility and optimization. The engine can perform complex data transformation and ensure data reliability because scraper operations execute similarly to an ETL pipeline.

3 DATA SCRAPING FUNDAMENTALS

To perform data scraping, scrapers need to understand the basics of HTML which are the building blocks of web pages. They also need to have some programming skills to write code that can navigate through websites and extract the desired data. In addition, it is important to know that web scraping raises ethical and legal concerns, particularly when it involves collecting personal or

sensitive information without consent. All of the above problems will be discussed in this chapter.

3.1 Hypertext Markup Language

Data scraping retrieves data from the Internet by actively interacting with the web pages. Therefore, HTML (Hypertext Markup Language) is a crucial factor when it comes to web scraping. HTML is a markup language used for creating websites alongside CSS (Cascade Style Sheet) and JavaScript. People might mistake HTML as a kind of programming language although it is not. A markup language is a computer language that performs text-encoding on a set of symbols inserted in a text document to control its structure, formatting, or even the relationship between the components rather than performing any action itself (Wikipedia, n.d.). Meanwhile, the programming language is also a computer language that developers use to build programs, scripts, and various sets of instructions for computers to execute.

When scraping, developers have to interact with HTML elements frequently to communicate with the web page itself. Therefore, it is mandatory to know the working principles of HTML. This paper will use the example on the Mozilla blog to explain the basic functionality of HTML.

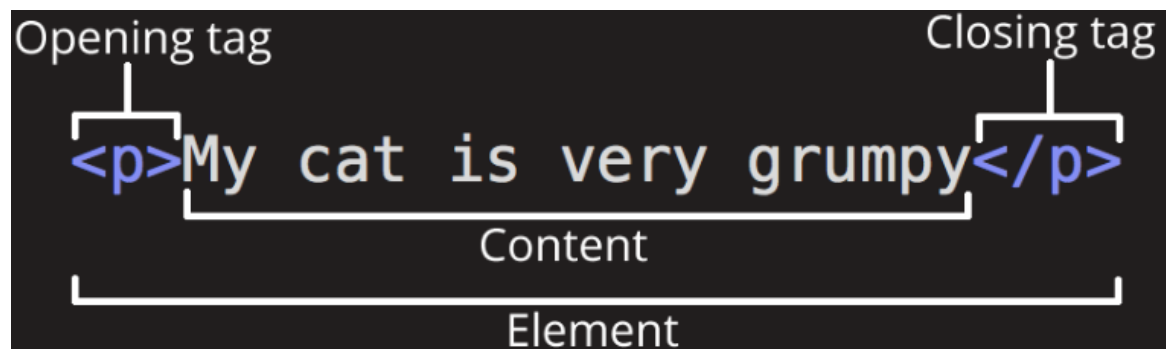


Figure 3. HTML element. HTML basics. Mozilla. n.d. Available at https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics. Accessed April 2023

Figure 3 displays the most basic form of HTML. HTML tags are used to define the elements inside an HTML document. The browser will use these tags to adjust

the web layout the match the settings correspondingly. Both tags on the figure share some similarities:

- The opening tag: The tag contains the name of the element, which is “p” in the figure. This will instruct the web page to implement this component starting from this point. The structure of the opening tag is “<name of the element>”
- The closing tag: The tag also has the name of the element inside closed brackets, which is “p”. However, its structure is “</name of the element>” which is slightly different from the opening tag to distinguish them from each other. The closing tag states the ending of the element
- The content: The content of the element which is normally plain text
- The element: The combination of opening tag, closing tag, and content

It is useful to know some of these regularly seen tags when you are trying to collect data on any website. Programmers can find the full detail of the above tags as well as every other tag of HTML on the W3Schools website:

<https://www.w3schools.com/tags/>

Table 1. Popular tag in scraping

Tag	Type	Functionality
DOCTYPE	meta	Declare the type of the document to the browser
title	meta	Declare the title of the document
p	text	Specify a paragraph
h1-h6	text	Represent 6 levels of section headings (font size)
br	text	Execute a line break
span	style	Container that marks up a part of text
a	link	Define a hyperlink
base	link	Specify the base link or/and target for all relative link
img	image	Embed an image into the page
ul	list	Specify unordered list
ol	list	Specify ordered list
li	list	Specify a list of items; usually inside ul, ol or menu tags
table	table	Specify an HTML table; consists of th, tr, and td tags
th	table	Define table name
tr	table	Define table row in the corresponding table element
td	table	Define table cell in the corresponding table element

All HTML elements can provide additional information about the elements by specifying attributes in the opening tag. Figure 4 shows another example from Mozilla that the writer will use to explain HTML attributes.

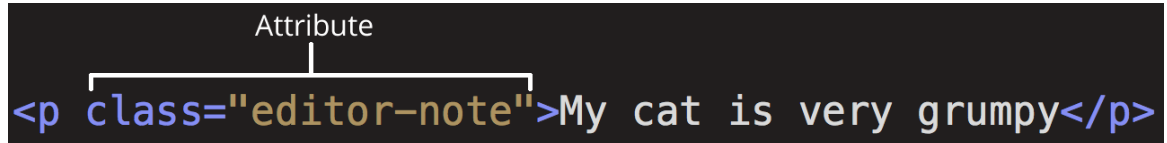


Figure 4. HTML attribute. HTML basics. Mozilla. n.d. Available at https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics. Accessed April 2023

In this element, “class” is the attribute name while “editor-note” is the attribute value. Attributes have various useful functions to support the elements. For instance, the “class” attribute in the figure serves a non-unique identifier that can be targeted alongside with other classes has the same value to customize with style information and other things. Scrapers are going to interact a lot with this “class” attribute. Hence, it is useful to know what it does beforehand. In addition, the “id” attribute is also an essential element for web scraping, which specifies a unique identifier for an HTML element. In HTML, this attribute is used to point to a specific style declaration in a style sheet. While in data scraping, scrapers use “id” as an identifier to extract the information they need.

It is crucial for the scrapers to retrieve the HTML document in order to be able to extract the information from web pages. Thankfully, most of the modern web browsers nowadays is equipped with a developer tool that can view the HTML document of the page with a few clicks: Right click on the web page you want to view the HTML → View page source / Inspect. Figure 5 shows two available options that the developer tool offers:

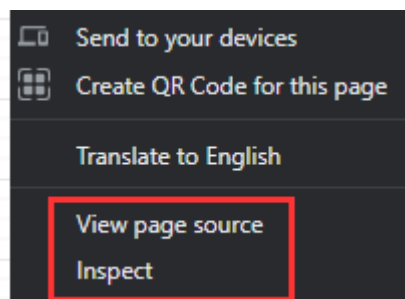


Figure 5. Developer tool example

- View page source: This option basically allows us to view the entire HTML document as it was delivered from the web server to our browser.
- Inspect: The more popular option for web scrapers. This option allows scrapers to choose whichever part on the web page to scan for its HTML element in the form of an HTML DOM (Document Object Model). For instance, if we try to inspect a random element on the page, the developer tool will appear and point to the corresponding element.

3.2 Document Object Model

DOM represents the structure of the HTML page as a tree of nodes and it is created whenever the page is loaded. Each HTML element is presented as a node. This object model acts as a programming interface for HTML because it allows developers to add, update, delete, or even move nodes on the tree. DOM is extremely useful in web scraping because programmers can use these nodes to interact with the HTML page extract the needed information. Figure 6 shows example of a DOM tree

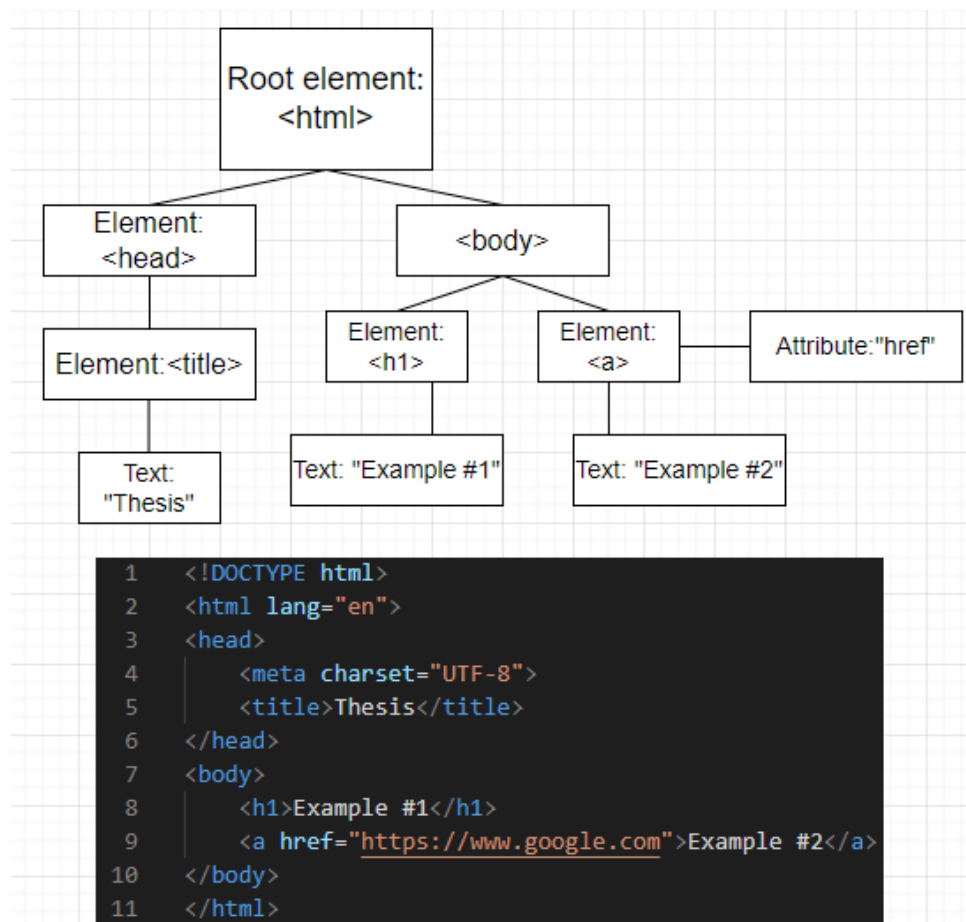


Figure 6. Example of a DOM tree

The aspect that makes DOM more manageable than the original HTML is DOM presents every node of elements or attributes as a hierarchical relationship. The writer will further examine the previous example to understand more about DOM.

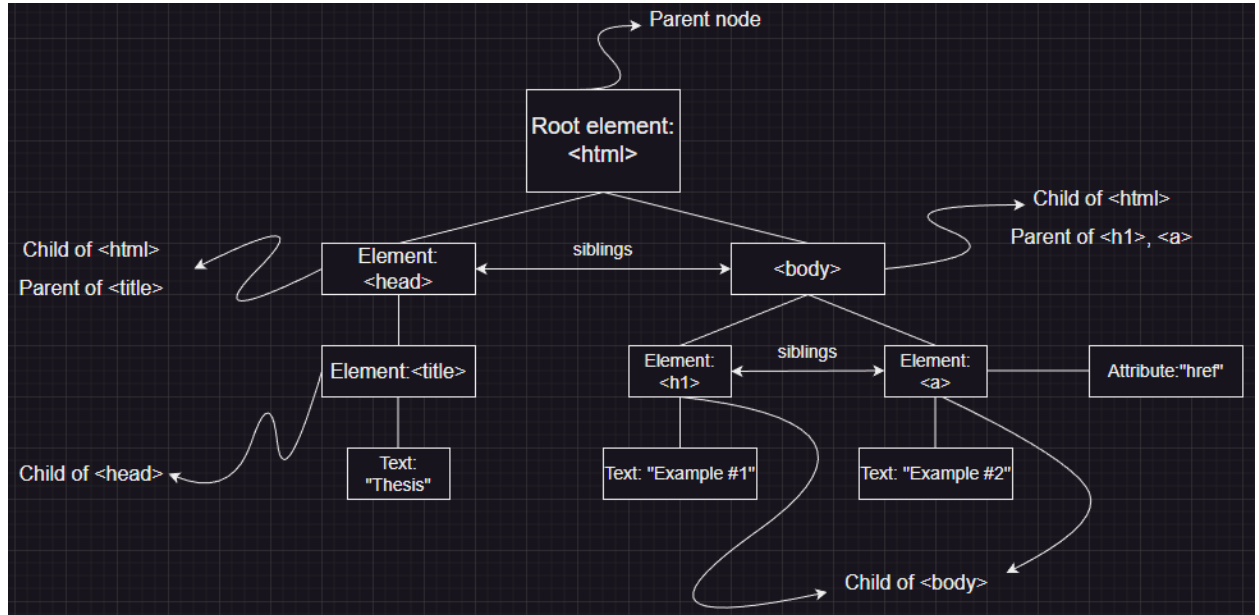


Figure 7. DOM tree characteristics

Figure 7 explains the main characteristics of the nodes:

- <html> is the root parent which consists of a collection of child nodes that represent the elements, attributes, and content of the HTML component
- Each node in the tree has a parent node, except the root element
- Each node can also have multiple child nodes
- Nodes having the same parent are siblings
- Each node represents an HTML element

Programmers take advantage of this to operate the HTML by using a technology called selector, which is frequently used in markup languages.

3.3 Document Object Model selector

As it was discussed in the previous chapter, we can interact with the web page through programming by utilizing the DOM node. In order to extract the information from the HTML elements, developers have to initialize a setup that consists of several steps: search for the needed tags → select the desired attribute → extract the information using regular expression → handle special

cases. That is a great deal of work due to the fact that how messy and complicated modern website's DOM tree is. Therefore, a selector is required to pass the targeted elements for the programming language to work with. A selector can simplify lots of the abstraction process from the traditional way by selecting and processing elements with high precision. The technology has the flexibility to navigate through every layer of tree hierarchy.

3.3.1 XPath selector

XPath means XML (Extensible Markup Language) Path Language. XPath is a selector that utilizes path expressions to select nodes or node sets in an XML document. The reason that web scraping can utilize this selector technique is XML is also a markup language similar to HTML. While XPath is not a programming language itself, this selector can create an expression that can directly point to a specific DOM node containing HTML tags or attributes without manually iterating the entire DOM tree. The below figure shows an example of a typical XPath syntax.

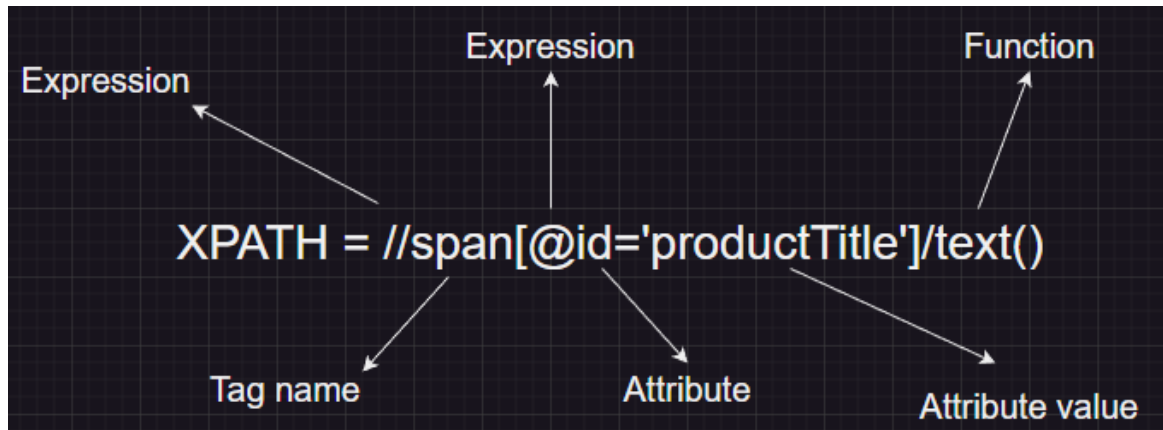


Figure 8. XPath components

The example shows the basic fundamental of an XPath syntax. This expression will retrieve the text value from all of the span tags that have `id='productTitle'`. While the function is optional, some of them are extremely useful to manipulate the result. The `text()` will extract the plain text from the attribute value. The thesis will mainly utilize this selector for the project. Therefore, the author will further

research each component of XPath to understand more about the capabilities of this selector.

Table 2 showcases fundamental XPath expression in scraping:

Table 2. XPath expressions

XPath expression	Functionality
nodename	Select all nodes with nodename
/	Select node from the path starting from root node
//	Select node from the context node
.	Select the current node
..	Select the current node's parent
@	Select attribute

List of the useful functions when extracting data is shown in the following table:

Table 3. XPath functions

XPath function	Usage
text()	Return the plain value from the element
number()	Convert data or node to number type
string()	Convert data or node to string type
substring()	Extract a substring from a string
contains()	String validation using another string
starts-with() ends-with()	Check if the string starts/ends with the given string; return True/False

Axes that are the components that XPath uses to access through the different layer of the DOM tree. List of commonly used axes for web scraping:

Table 4. XPath axes

XPath axes	Usage
following::	Indicate all the nodes that appear after the context node
preceding::	Indicate all the nodes that appear before the context node
parent::	Indicate the parent node of the context node
child::	Indicate the child of the context node (Default option if XPath expression does not specify an axis argument)
self	Indicate the context node itself

These are some XPath expression examples and their usage to showcase the functionality of the above components:

Table 5. XPath examples

Example	Explanation
/html/body/title	Select the title node
//span	Select all the span nodes
//span[@id]	Select all the span nodes that have the id attribute
//span[@id='title']	Select all the span nodes with the following attribute: <i>id='title'</i>
//span[@id='title']/text()	Retrieve the text content of every span element that has <i>id='title'</i>
//h1/span[1]	Select the first child of the element
//*	Select every element in the document
//div[@class='product']//following::span	Select every span element after the div element that has <i>class='product'</i>
//div[@class='product']//parent::span	Select every span parent element within the div element that has <i>class='product'</i>

On that account, the XPath selector is a crucial technology for web scraping. XPath can examine several different layers of a markup document to extract the needed elements. More real-life usage of XPath will be showcased in the later chapter of the thesis because the project is going to utilize XPath to develop a web scraping application.

3.3.2 CSS selector

CSS selector is another viable option to extract the needed information from HTML. In CSS, selectors are used to target the HTML elements on web pages in order to implement CSS settings. Web scraping takes advantage of this functionality to extract the selected elements.

While the project chooses XPath as its main selector, it is difficult to fully explain the CSS option without project implementation. Therefore, the writer will not explain in detail how CSS selector works and only list it as a viable option for web scraping. More detail about this selector can be found on Web Scraper documentation: <https://webscraper.io/documentation/css-selector>

3.4 Scraping framework

Scraping frameworks are designed to automate the process of collecting data from the web, allowing developers to efficiently retrieve large amounts of information from multiple sources. They typically provide a set of tools and libraries that make it easier for developers to extract and process data from web pages, as well as handle common issues such as page navigation, authentication and storage.

3.4.1 Scrapy

Scrapy is an open-source framework for extracting data from websites and it is the thesis's main focus. The framework is currently maintained by Zyte, a web scraping development company, and many other contributors. Scrapy is written in Python and it still frequently gets updated because of the large community support. It is available on three main operating systems which are Windows, Linux and MacOS. However, this framework is only available on one programming language, which is Python at the moment. Scrapy gains many benefits from Python and the author will discuss this in the later section.

Scrapy workflow is constructed by several components which are engine, spiders, scheduler, downloader, item pipeline, downloader middlewares, and spider middlewares. The information about these can be found on the Scrapy documentation page (Scrapy, n.d.):

- The engine controls the data flow between all of the components and triggers events when actions occur
- The scheduler manages requests from the engine
- Downloader fetches web pages into the engine

- Spider parses responses using HTTP requests and extracts the needed items from the response
- Item pipeline is responsible for processing the data once they have been extracted by the spiders. Users interact with this component to create data transformation tasks including cleaning, validation, and persistence
- Downloader middleware is the middle process of the engine and the downloader. The middleware passes information from both sides. However, users can adjust the configurations based on their usage. For instance:
 - change received responses before passing it to a spider
 - send a new request instead of passing the received response
 - pass responses to a spider without fetching a web page
 - drop request
- Spider middleware is the middle process of the engine and the spider. While it is optional, this middleware is capable of:
 - adjust the post-process output of spider callbacks
 - change received responses before passing it to a spider
 - handle spider exceptions

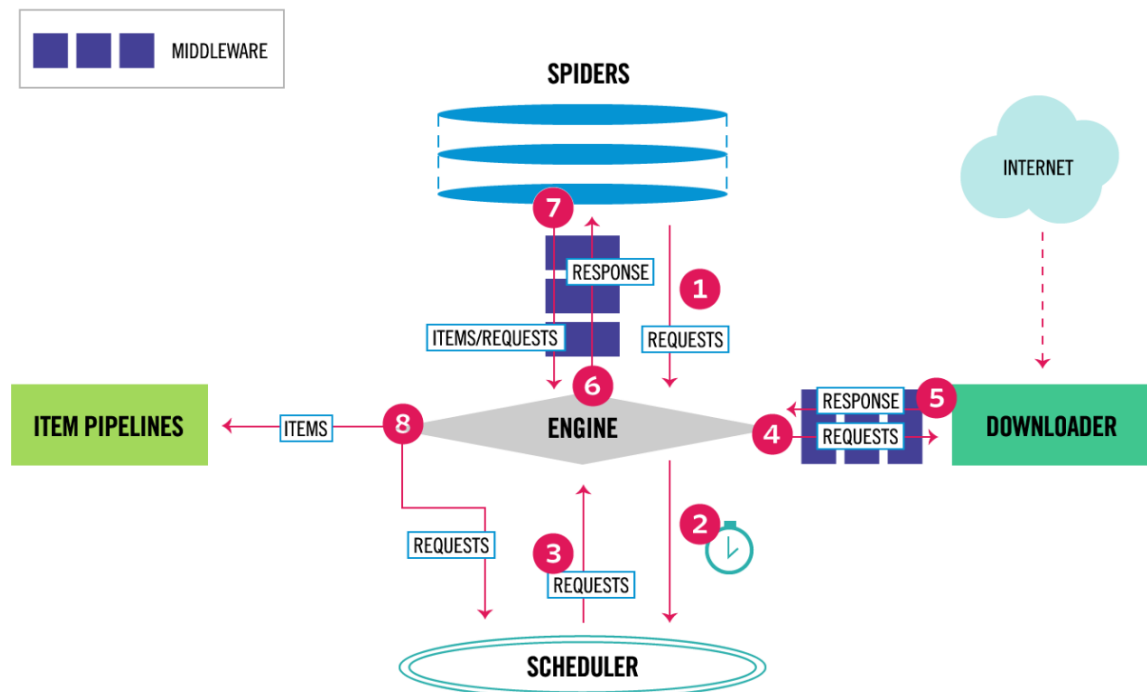


Figure 9. Scrapy dataflow. Architecture overview. Scrapy. n.d. Available at <https://docs.scrapy.org/en/latest/topics/architecture.html>. Accessed April 2023

Figure 9 illustrates how each component interacts with the other:

1. The engine receives the initial requests to crawl from the spider
2. The engine instructs the scheduler to automatically send a new request after the first one from the spider

3. The scheduler returns the next request to the engine. From this point, the process will keep repeating until there are no new requests left
4. The engine sends the request to the downloader
5. The downloader generates a response from the request and then sends it back to the engine
6. The engine receives the response from the downloader and sends it back to the spider for data processing
7. The spider processes the item from the previous step and then returns the processed item to the engine
8. The engine sends the processed item to the item pipelines for data reformatting and exporting

From the workflow, the developer can observe that Scrapy architecture resolves heavily around the core engine. The engine handles every response and request from other components and then passes it to the proper destination. However, while the core engine typically functions behind the scene, users mainly work with Scrapy through spiders and item pipelines.

The combination of spider and item pipelines represents an ETL process. Spider is a custom Python class that allows programmers to customize their coding to parse and gather the data from the response. In this class, users choose what web page to parse and utilize XPath to extract the needed HTML elements from the site, and setups the crawler to navigate through the pages. Therefore, the spider technically serves as the Extract phase from ETL. While the spider specializes in abstraction, item pipelines handle the scraped material and process them into the desired format. These working principles are relatively the same as the Transform phase and the Load phase from ETL. Therefore, this is a significant advantage for Scrapy for having a built-in ETL architecture. As the writer already discussed in the previous chapter, ETL delivers high-quality data and provides better control over the data flow.

Parallel processing is another useful feature that Scrapy supports. It allows the engine to scrape multiple items concurrently. For example, the developer has to collect all questions related to the database on Stack Overflow.

The screenshot shows the Stack Overflow interface for the 'database' tag. At the top, there's a search bar with '[database]' entered. The main heading is 'Questions tagged [database]' with an 'Ask Question' button. Below this is a descriptive paragraph about databases. A filter bar shows '192,405 questions' and various filters like 'Newest', 'Active', 'Bountied 3', 'Unanswered', and 'More'. The first question listed is 'transfer Sqlite database information from one android device to another over bluetooth or wifi-direct' by user 'Sanjay Jaryal', posted 17 minutes ago. The question has 0 votes, 0 answers, and 3 views. Tags for 'android', 'database', 'sqlite', 'bluetooth', and 'wifi-direct' are visible below the question title.

Figure 10. Stack Overflow example #2

Figure 10 shows that there are approximately 192,405 questions for this topic. If the users are using 50 items per page layout, they have to scrape relatively 3,849 pages. Parallel processing allows Scrapy to perform 16 requests at the same time assuming that each request takes one second to collect 50 questions on a single page. Users will be able to generate 16 pages in one second meaning $16 * 50 = 800$ items. Accordingly, this will take about $3,849 / 16 = 241$ seconds in total to gather all the needed data. It is important to keep in mind that the performance is subjective because it heavily relies on several elements such as hardware and internet bandwidth. Scraper can also setup how many requests to execute at a time and the default value is 16. There is no limit to this value, however, setting this value too high might cause heavy traffic to the targeted website. The developer should ask the website owner for permission in this case. The parallel processing is still an outstanding feature that makes Scrapy scales application more effectively than other scraping framework on the market, all things considered.

Despite the speed and efficiency, Scrapy still has several limitations to consider. Scrapy could not handle dynamically-loaded content such as JavaScript by default. The reason is JavaScript elements typically will not be shown on the HTML document or DOM tree. This is a noticeable downside since JavaScript is casually implemented in most modern websites nowadays. Fortunately, there is a

workaround solution for this problem which is Splash. Splash is a Python library that directly supports Scrapy to extract data from JavaScript rendered websites, more information about Splash can be found in the Splash documentation: <https://splash.readthedocs.io/en/stable/faq.html>. While Scrapy provides massive flexibility and highly customizable settings, the framework produces a decent amount of complexity and might be overwhelming for anyone who attempts to learn and use Scrapy.

3.4.2 Selenium and Beautiful Soup

While the thesis mainly focuses on Scrapy framework, this section will briefly introduce other available options for data scraping. Depending on situations, these alternatives may prove to be a better solution than Scrapy.

Selenium is an open-source automated testing framework that specializes in validating web applications across different browsers and platforms. Unlike Scrapy, Selenium is available for several programming language such as Java, C#, Python and many others. Selenium can render all elements of the selected web page including JavaScript into its web engine. After that, the programmer can freely interact with the website elements to collect the information by utilizing several built-in methods. Because the framework was not originally designed for data scraping, Selenium is missing a lot of support features for scraping itself. Moreover, the performance is not comparable to Scrapy because Selenium basically loads the entire web page into its engine to function. In contrast, if speed is not your top priority, Selenium will be a decent option since it can also handle JavaScript effortlessly.

Beautiful Soup is a Python library that directly supports data scraping. While Beautiful Soup is technically not a framework, it is worth mentioned because this library is a powerful tool for scraping data on the Internet. The engine also cannot handle dynamic content because it also parses web data using HTTP request, which is similar to Scrapy. Because Beautiful Soup is a lightweight library, it lacks the customization and the performance to extract large volume of data. However, this characteristic makes BeautifulSoup suitable for small-scale projects.

3.5 Programming language Python

Python is a general-purpose programming language that was originally found by Guido van Rossum in 1991. General-purpose programming language is a term that describes computer language that can be used to build a wide variety of applications serving different usage. Python is available on multiple operating systems including Windows, macOS, Linux/UNIX and more. Python is designed to have great code readability with the use of significant indentation. Therefore, the language is not difficult to learn and start programming with it. Moreover, Python supports several programming paradigms including structured, functional and object-oriented programming (Wikipedia, n.d.).

Python has always been one of the best programming languages for data processing. The simplicity is not the only reason that makes Python extremely popular in the data industry. Users can regularly find solutions and helps on the Internet nowadays because Python have such a huge and active community. Moreover, the language has accessed to considerable number of data-related libraries and frameworks:

- Pandas: A popular library that allows programmers to perform data processing and data visualization
- NumPy: NumPy offers several high-level mathematical functions to perform calculations with the data
- Sklearn: A library specializes in machine learning. Sklearn provides utilities for developing machine learning model
- Matplotlib: A powerful data visualizer for many variants of data
- Scrapy, Selenium, BeautifulSoup: All three popular scraping tools are all available on Python

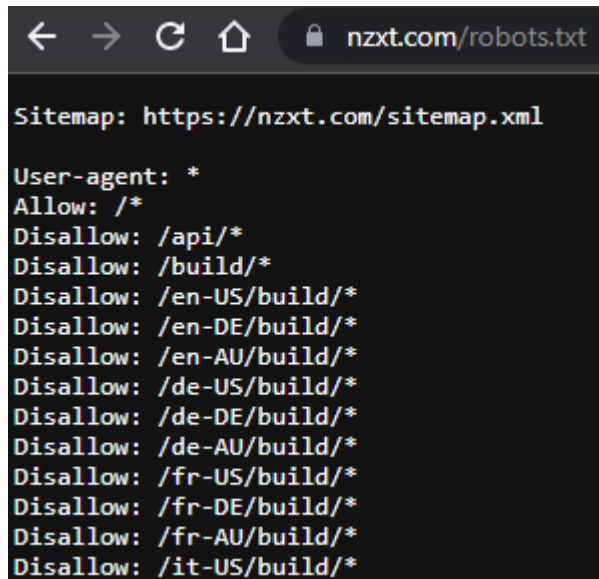
Therefore, Python is a solid option for data scraping project. Scrapy and other scraping framework have accessed to many useful tools that Python has to offer. There are others language that developers can consider for their data scraping project such as C++, Java, JavaScript and many others. However, it is not recommended to use these languages for complex data project because most of them are limited to powerful data processing tools comparing to Python.

3.6 Legal issues and ethical scraping

Many people assume that scraping is an act of stealing data on the Internet. In contrast, data scraping is a technique that collects only visible information on the Internet. Web scraping is technically legal based on the definition. There are no laws or restrictions that prohibit people from scraping websites under normal circumstances. It is important to know that there are also exceptions:

- Scraper uses the collected information for the harmful intention
- Scraper extracts personal information which heavily violates the regulations of many countries such as GDPR (General Protection Regulation) in the EU
- Scraper copies the material that is copyrighted

Another crucial element that helps us to identify what is allowed to scrape is the robots.txt file. This text file is basically a set of instructions for bots that try to navigate through the website. The bots in this case are mostly search engines such as Google and Bing. Web scrapers can also utilize robot.txt in order to decide which section of the page is allowed to scrape.



```
← → ↻ 🏠 🔒 nzxt.com/robots.txt
Sitemap: https://nzxt.com/sitemap.xml
User-agent: *
Allow: /*
Disallow: /api/*
Disallow: /build/*
Disallow: /en-US/build/*
Disallow: /en-DE/build/*
Disallow: /en-AU/build/*
Disallow: /de-US/build/*
Disallow: /de-DE/build/*
Disallow: /de-AU/build/*
Disallow: /fr-US/build/*
Disallow: /fr-DE/build/*
Disallow: /fr-AU/build/*
Disallow: /it-US/build/*
```

Figure 11. robots.txt example

Figure 11 shows how to access the robots.txt of most modern websites nowadays. The file can be accessed by adding /robots.txt to the path of the URL

address. In addition, the content of the text file is also shown and we are going to use this example to show how robots.txt create rules for the website:

Table 6. robots.txt rules

Sitemap	Show the sitemap URL, which is used to display important content that the site recommends the bot to navigate
User-agent: *	Specify that the following rule applies to all bots and users
Allow: /*	Permit bots to crawl the entire website not including the exception from the Disallow rule
Disallow: /api/*	Forbid bots to crawl the context URL path

While scraping is completely legal, many websites recommend that they are not being scraped in general. One major problem that website owner has with scraping is it could potentially cause heavy traffic or even interrupt the website services if scrapers parse numerous requests concurrently over a long period. This could be treated as a DoS (Denial-of-Service) attack due to the consequence that overload scraping can cause. Correspondingly, a few numbers of webpages such as Newegg directly prohibit scraping on their site, which is understandable. Therefore, developers should carefully read the terms of service and policy agreement on the targeted website. Additionally, these are several things that the developer should consider before scraping any website:

- Prioritize API (Application Programming Interface) over scraping if the website provides
- Only scrape the data that you need
- Always check the terms of use and robots.txt of the selected website
- Schedule your scrapers properly to not overload the website infrastructures

4 DATA SCRAPING APPLICATION WITH SCRAPY

This project attempted to scrape Amazon, one of the largest e-commerce websites at the moment. By implementing this, the developer will have a better understanding of the capabilities of Scrapy and data scraping in general. This chapter will showcase how to build a web scraper from installing needed components to optimizing Scrapy performance.

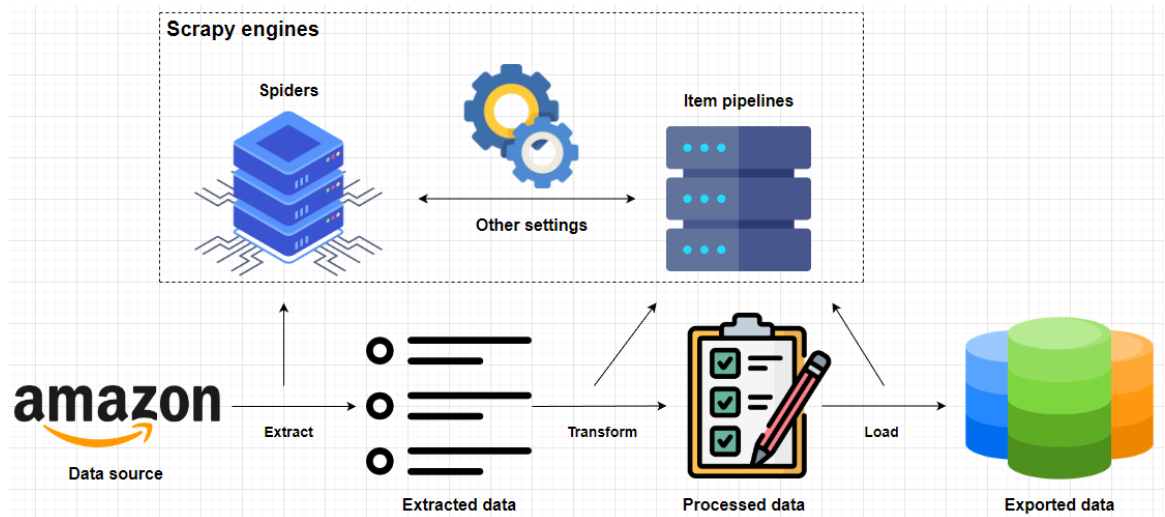


Figure 12. Project architecture

The scraping project architecture is shown in Figure 12. This project will attempt to build a data scraping application that collects any product information on the site including price, name, description, and many others. The main functionalities of this application are specifying the needed product through the search bar, extracting its information, and navigating through different pages to collect all data of the searched product. Moreover, the collected data will be transformed by utilizing the item pipelines of Scrapy. The final step is exporting the data to the desired format which can be a spreadsheet, JSON, XML, or a database table.

4.1 Prerequisite

The first step is always to check if the selected website specifically forbids data scraping. In this case, Amazon does not prohibit scraping activities in their terms of service. Therefore, it is safe to collect the data from Amazon website.

Additionally, developer should also review the robots.txt of Amazon to learn which section of the site is not available to interact with. The robots.txt of Amazon can be accessed by navigating to <https://www.amazon.com/robots.txt>

After researching the site rules, we set up the project environment. The operating system that the writer used for this project is Windows. Two core components of this application are Scrapy and Python:

- Python 3.11 will be used. The installation file for Windows can be found from on the Python website: <https://www.python.org/downloads/>

- Scrapy 2.7.1 will be used. Programmers can install Scrapy through pip, a Python package installer. Pip is automatically installed when the user installs Python. It is recommended to install the framework under a virtual environment that Python has to offer. This will prevent version conflict between the components. Using the following command will install Scrapy:
`pip install scrapy`

After the installation, programmers can start working on the project immediately.

In the initialization step, the user needs to create necessary files for Scrapy by doing it manually or entering Scrapy built-in command into the command prompt:

```
scrapy startproject scrapy_amazon
```

This will create a directory including all the files we need for our project:

- Folder *spiders* contains all the spiders, which are used to extract the data. There can be multiple spiders in a project
- *items.py* defines the data model of the scraped items
- *middlewares.py* controls the mid-process of every component in Scrapy
- *pipelines.py* contains the item pipelines, which perform all of the data processing work of the project
- *settings.py* allows the user to adjust multiple settings including the number of concurrent requests, cookies configuration,...
- *scrapy.cfg* contains the meta information about the current project

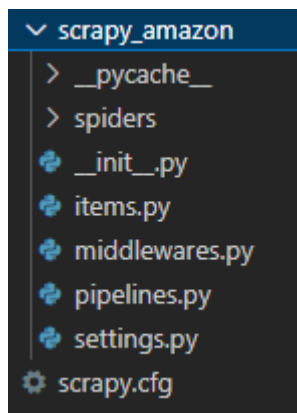


Figure 13. Project folder

A spider can be created by using the following syntax:

```
scrapy genspider Products www.amazon.com
```

The syntax will generate a script named Products.py inside the *spiders* folder.

The command includes our spider name and the targeted website for scraping.

```

1 import scrapy
2 from scrapy.spiders import CrawlSpider, XMLFeedSpider, CSVFeedSpider, SitemapSpider
3
4
5 class ProductSpider(scrapy.Spider):
6     name = 'Product'
7     allowed_domains = ['amazon.com']
8     start_urls = ['http://amazon.com/']
9
10    def parse(self, response):
11        pass
12

```

Figure 14. Initial the spider

Scrapy will generate a template for our spider, a Python class. The class contains metadata for Scrapy to process. Users can create requests to websites by using the parse function. Scrapy also supports many different types of spiders serving different purposes:

Table 7. Spider types

Spider	A generic spider that serves multiple purposes based on how users setup. This spider will be used for the thesis.
CrawlSpider	Focus on navigating through different pages and links
XMLFeedSpider	Specialize in extracting XML content
CSVFeedSpider	Specialize in extracting CSV content
SitemapSpider	A crawl spider that prioritizes the sitemap rules

Finally, before building the spider, the scraper needs to plan how the spider parses the website in advance.

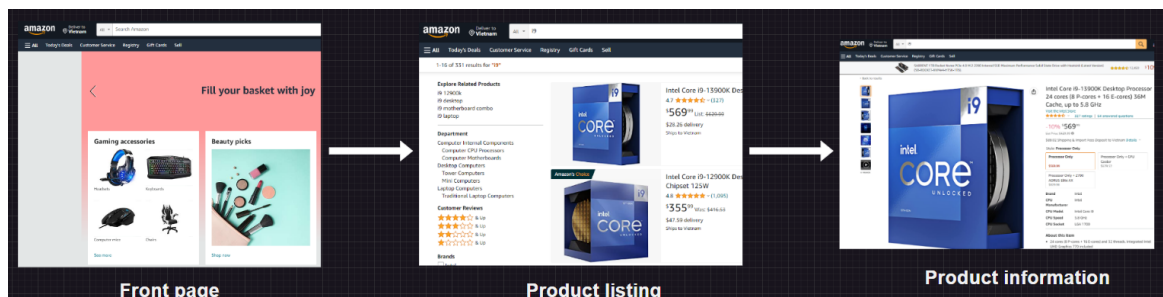


Figure 15. Project flow

Figure 15 shows how the spider navigates in this project: Front page → Product listing → Product information.

4.2 Data extraction

Data extraction is a crucial step in web scraping, which is the process of automatically collecting and extracting data from websites. In this chapter, the author attempted to study the web structure of Amazon and extracted the data.

4.2.1 Navigating through Amazon

In order to be able to collect certain product information, the programmer has to guide the spider from the starting point to the destination. URL and HTML are two important components that help us to navigate around the website. This section of the project will utilize the URL address to parse the product listing into the engine. It is important to know the different components of a web address.

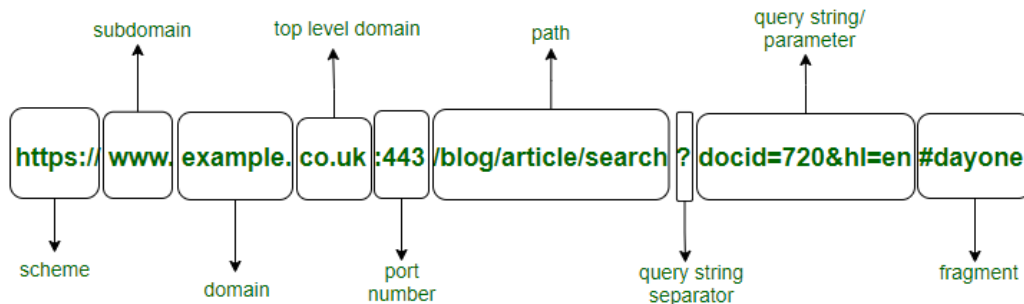


Figure 16. URL components. Components of a URL. GeeksforGeeks. n.d. Available at <https://www.geeksforgeeks.org/components-of-a-url/>. Accessed April 2023

The following diagram from GeeksforGeeks shows how a URL is address typically constructed. From front page to product listing, we are going to take advantage of the search function of Amazon.

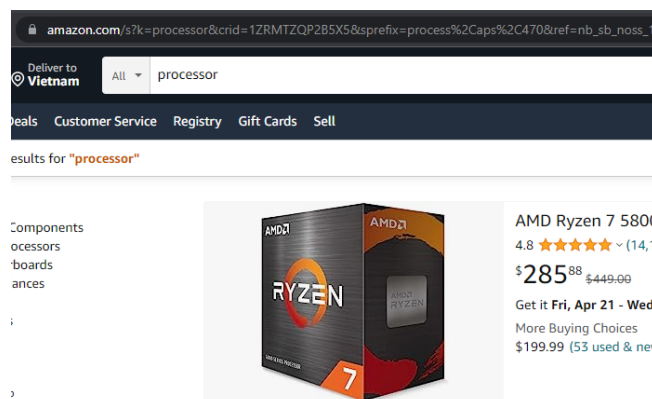


Figure 17. Amazon search #1

From Figure 17, we can see that every time the user searches for a product, the page will redirect the user to the searched item catalog. By searching “processor”, Amazon redirects the user to the “processor” catalog. Our main focus in the figure is the address. The address is overwhelming in this situation. However, spiders do not need the full address to access the web page. Most components in the address are optional arguments. These arguments are mostly query-string which is indicated by a precede “?” symbol or “&” symbol. Scrapers can safely ignore these and focus on the important parts which are the domain name and the path of the address.

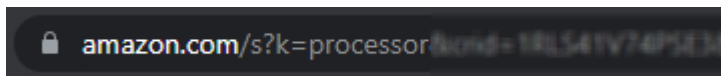


Figure 18. Search URL

Now we can clearly see the URL format of the catalog page on Amazon. Developers can assume that “s?k” presents for “searched keyword” because it has “processor” value, which is our searched item. We need to confirm this by performing a simple test. The author will attempt to change the “s?k” element to another value for this test.

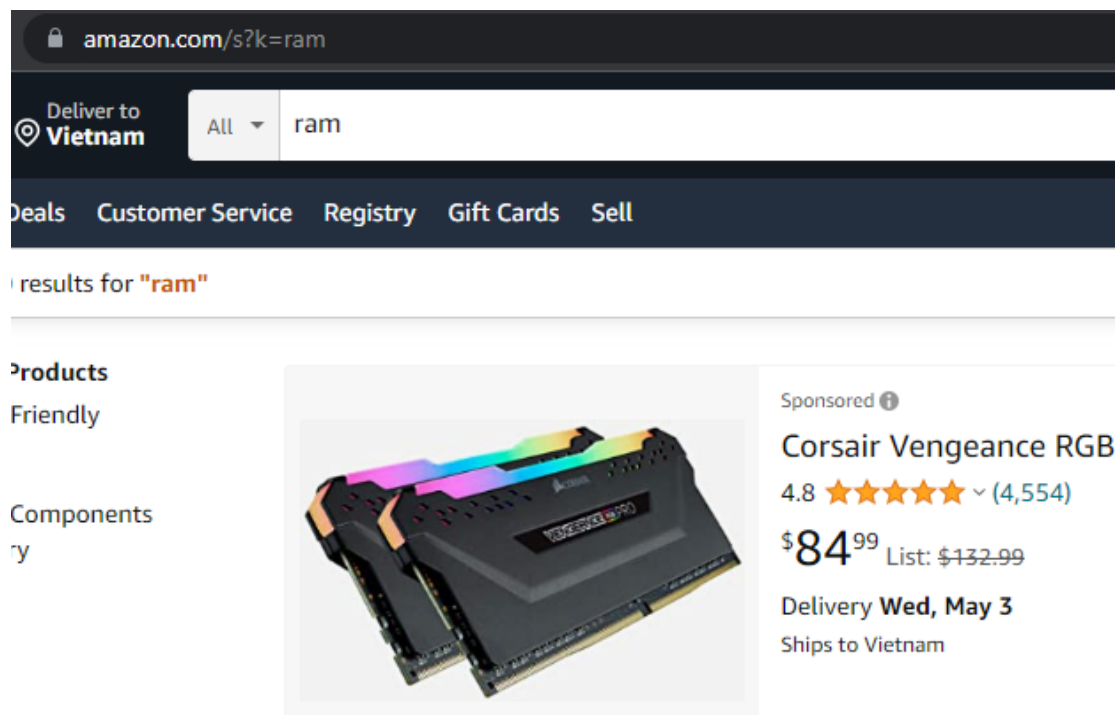


Figure 19. Amazon search #2

Figure 19 proves our theory was correct. By adjusting the “s?k” value, Amazon will redirect us to the corresponding catalog. Therefore, the developer can safely implement this logic into the spider.

```
class ProductSpider(scrapy.Spider):
    name = 'Products'
    search_key = input("Enter an item: ")
    search_item = search_key.replace(' ', '+')
    allowed_domains = ['www.amazon.com']
    start_urls = [f'https://www.amazon.com/s?k={search_item}']
```

Figure 20. Implement search function

- *search_key* receives the inputted item from the user
- *search_item* handles the spacing between items that contain many words such as Raspberry Pi, graphic cards, etc.
- The search function will be implemented at the start of the program because we want this to run as soon as the spider starts

After successfully parsing the listing into the Scrapy engine, users can interact with HTML contents of the page by utilizing Scrapy. Therefore, the writer will analyze the product properties to figure out a reliable way to navigate to the product information page from the catalog. Every product should have the same HTML patterns so the scraper only needs to examine any single item. This is where programmers should utilize the Inspect option from developer tools provided by the web browsers. We are looking for the *href* element which usually contains the product information link. In this case, clicking the image or the product title will move the user to the corresponding product page. Therefore, the image and the title of the product are the two elements that we are going to analyze using the tools. The author will attempt to examine the image element using the Inspect option:

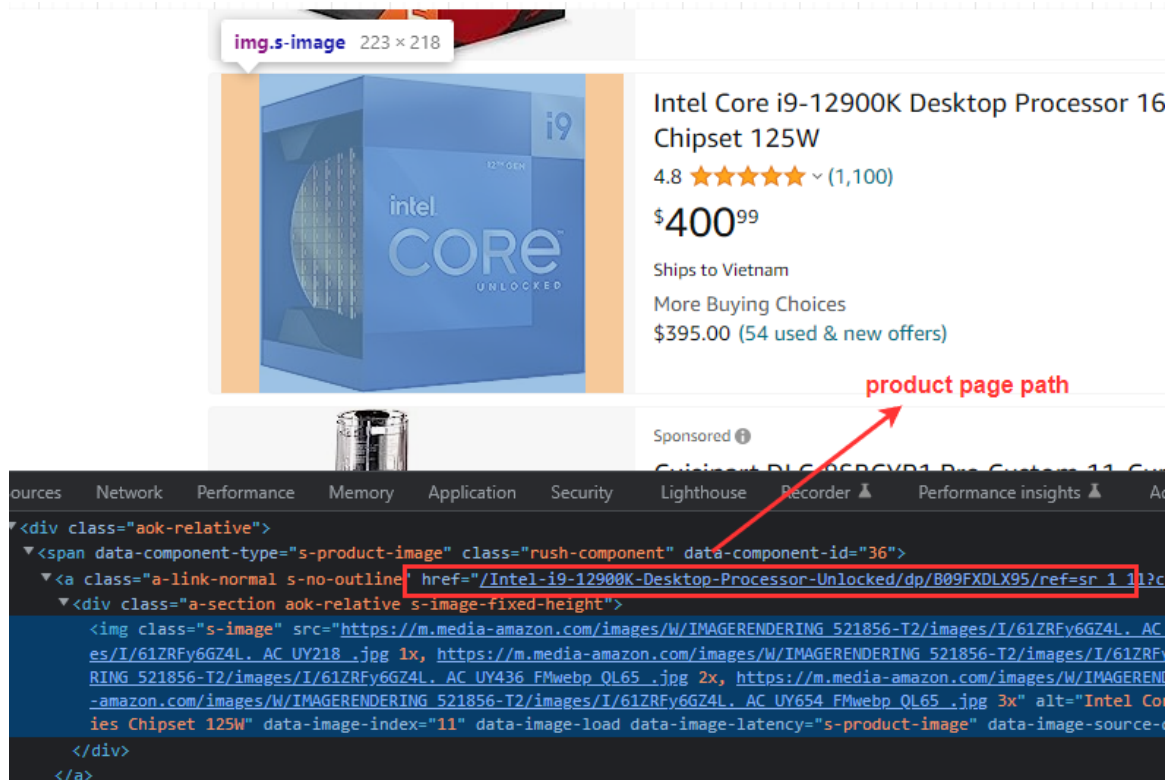


Figure 21. Inspect Amazon product listing #2

The developer tool also points out the property that the user uses the inspect option on. From the figure, we can see the *href* that we are looking for is the parent node of the highlighted node. It is important to know that the tool only highlights the actual image node not including any other nodes. Scraper can also apply the same logic to the process of getting the link from the item title similar to Figure 22.

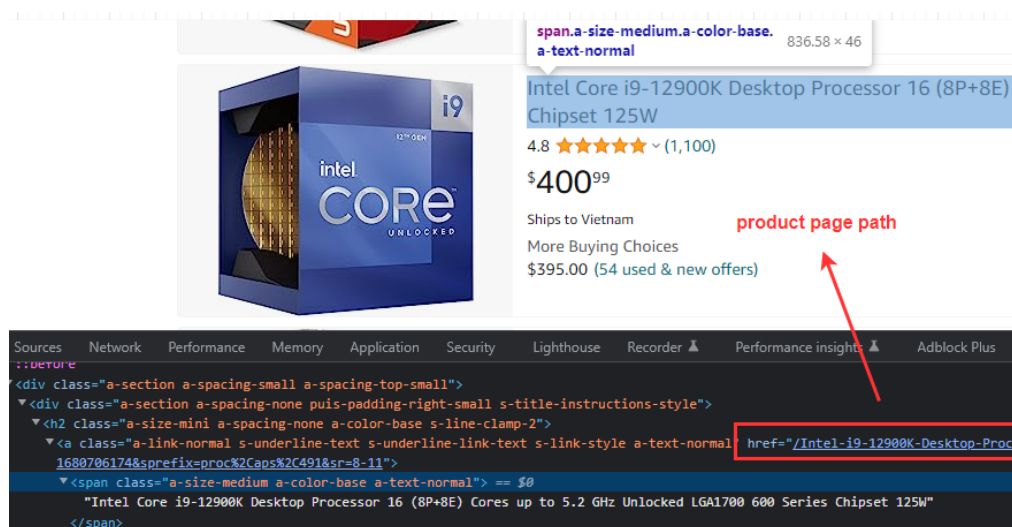
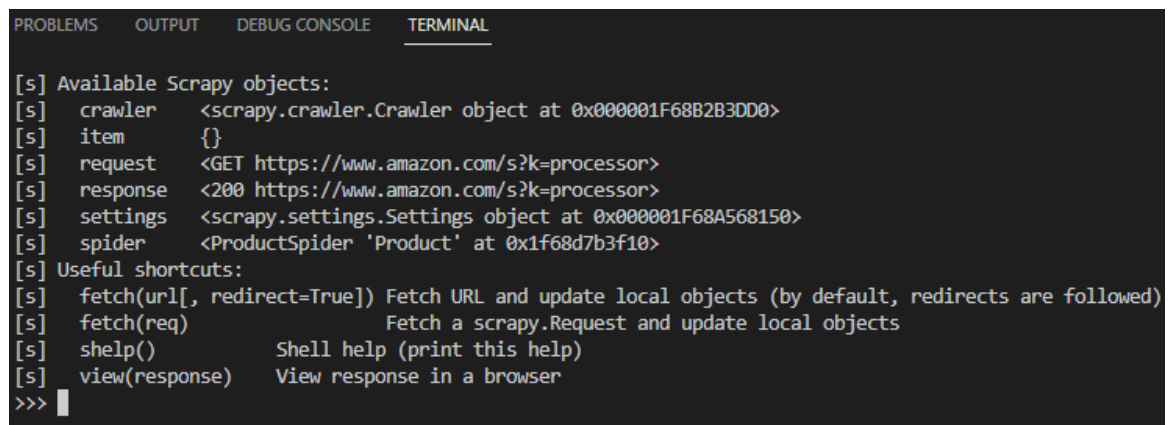


Figure 22. Inspect Amazon product listing #3

Finally, the programmer needs to implement these findings into the spider. When working with DOM and HTML, XPath and Scrapy interactive shell are used to interact with them. Scrapy shell provides a command line environment that allows developers to test their XPath expression. While the whole engine will run when spiders start a request, Scrapy shell only renders the targeted address with needed components for code testing. It is recommended to use the shell for testing or debugging extraction code because of the efficiency. The programmer can start a Scrapy shell section by inputting the following command:

`scrapy shell https://www.amazon.com/s?k=processor`



```

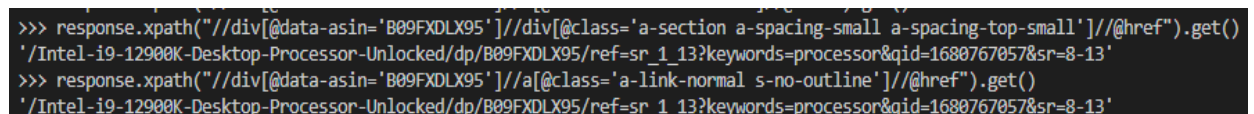
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

[s] Available Scrapy objects:
[s]  crawler  <scrapy.crawler.Crawler object at 0x000001F68B2B3DD0>
[s]  item     {}
[s]  request  <GET https://www.amazon.com/s?k=processor>
[s]  response <200 https://www.amazon.com/s?k=processor>
[s]  settings <scrapy.settings.Settings object at 0x000001F68A568150>
[s]  spider   <ProductSpider 'Product' at 0x1f68d7b3f10>
[s] Useful shortcuts:
[s]  fetch(url[, redirect=True]) Fetch URL and update local objects (by default, redirects are followed)
[s]  fetch(req)                   Fetch a scrapy.Request and update local objects
[s]  shelp()                      Shell help (print this help)
[s]  view(response)              View response in a browser
>>>

```

Figure 23. Scrapy shell interface

The following figure contains the XPath expression for retrieving the address path from both the product image and title. This is known as the traditional way to retrieve a URL address from any webpage. The *data-asin* attribute is from the parent node in both elements which is used to identify the context item.



```

>>> response.xpath("//div[@data-asin='B09FXDLX95']//div[@class='a-section a-spacing-small a-spacing-top-small']//@href").get()
'/Intel-i9-12900K-Desktop-Processor-Unlocked/dp/B09FXDLX95/ref=sr_1_13?keywords=processor&qid=1680767057&sr=8-13'
>>> response.xpath("//div[@data-asin='B09FXDLX95']//a[@class='a-link-normal s-no-outline']//@href").get()
'/Intel-i9-12900K-Desktop-Processor-Unlocked/dp/B09FXDLX95/ref=sr_1_13?keywords=processor&qid=1680767057&sr=8-13'

```

Figure 24. Testing XPath expression with Scrapy shell

Another approach for accessing the product page is using ASIN. ASIN is the acronym for Amazon Standard Identify Number. These identifiers typically have 10-character alphanumeric assigned by Amazon. Each product on Amazon is given a unique ASIN. For example, the B09FXDLX95 presents the i9-12900k

processor. Every product link on Amazon has the same format when ignoring all optional parameters: “https://www.amazon.com/dp/{asin}”.

The writer will choose the ASIN method to implement in the Scrapy spider. The parse function contains most of the extraction code of the spiders. We need to loop the entire product listing page to gather all the available product addresses.

```
def parse(self, response):

    product_listing = response.xpath('//*[@data-asin]')
    for product in product_listing:
        asin = product.xpath('@data-asin').extract_first()
        product_address = f"https://www.amazon.com/dp/{asin}"
        yield response.follow(product_address,
                               #callback=self.parse_product,
                               meta={'asin': asin, 'page_location': response.url})
```

Figure 25. parse() code

The **parse()** function will focus on scraping the product listing:

- *product_listing* extracts all the ASIN available on the catalog
- The loop makes sure that the spider visits every product inside *product_listing*
- *asin* is extracted again for constructing the address
- *product_address* is the product URL address
- *response* is an object that represents an HTTP response which is used to feed the spider for processing, follow method allows the response to access a URL address. These are the parameters of the syntax
 - *urls = product_address*
 - *callback* executes the corresponding function, **parse_product**, when the request is made. The **parse_product** function contains the data extraction code for the product page and the author will demonstrate this in detail in the next section. This option will be disabled for now.
 - *meta* produces the metadata for the corresponding item

Scraper can test the codes by running the following command to start the spider

`scrapy crawl Products`

Although we did not build any code that exports output, we can still check the Scrapy log which is automatically generated whenever the spider is running. The log contains several useful pieces of information for debugging purposes:

- The initialization of the engine
- The shutdown of the engine
- Error codes if an error occurs
- HTTP request status
- Output of extraction code

The debug log shows that we have successfully accessed the product page. The HTTP code 200 indicates that the request has succeeded.

```
[scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.amazon.com/dp/B00018RRRK> (referer: https://www.amazon.com/dp/B09HDPPXNV)
[scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.amazon.com/dp/B071CH3TLT> (referer: https://www.amazon.com/dp/B091D2NR1Q)
[scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.amazon.com/dp/B08LLDLQMY> (referer: https://www.amazon.com/dp/B07CTBHQZK)
[scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.amazon.com/dp/B095BRGSFH> (referer: https://www.amazon.com/dp/B07CTBHQZK)
[scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.amazon.com/dp/B09BB74CLW> (referer: https://www.amazon.com/dp/B00018RRRK)
[scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.amazon.com/dp/B07ZXV5NCL> (referer: https://www.amazon.com/dp/B071CH3TLT)
```

Figure 26. Debugging with Scrapy log #1

As we can see from the figure above, the ASIN method affords a decently cleaner URL address while the link from the traditional method contains too many optional arguments in general. This is useful considering that scrapers usually want their data as clean as possible. However, unlike Amazon, many websites do not provide a unique identifier for their content. Therefore, it is recommended to know the traditional way to extract an address because this method will work most of the time.

4.2.2 Extracting product information

The first step is initializing the Scrapy engine to store the upcoming data. Scraper accomplishes this by creating a data model in `items.py`. In this project, the writer wants to store the following information:

```

items.py x Products.py
scrapy_amazon > items.py > ...
1  import scrapy
2
3  class Product(scrapy.Item):
4      asin = scrapy.Field()
5      title = scrapy.Field()
6      price = scrapy.Field()
7      price_currency = scrapy.Field()
8      review_count = scrapy.Field()
9      rating = scrapy.Field()
10     url = scrapy.Field()
11     search_key = scrapy.Field()
12

```

Figure 27. Initialize Scrapy data model

These are the information that we are going to collect on Amazon. Each piece of information represents a data field in Scrapy. Spiders usually return the extracted data as items, Python objects that specify key-value pairs. Scrapy supports multiple types of items through its built-in *ItemAdapter* library. The scraper also needs to add the data class to the spider `Products.py`:

```
import scrapy
```

```
from scrapy_amazon.items import Product
```

Figure 28 shows the location of each needed information on the product page.

The screenshot shows an Amazon product page for an ASRock X670E Steel Legend Support AMD AM5 RYZEN 7000 Series Processors Motherboard. Red arrows point to the following elements:

- rating**: Points to the star rating and '25 ratings' text.
- title**: Points to the product title.
- price and currency**: Points to the price '\$334⁹¹'.
- number of reviews**: Points to the '25 ratings' text.

Other visible information includes:

- Brand: ASRock
- Amazon's Choice for "asrock x670e talchit"
- Discount: -12% \$334⁹¹ (List Price: \$379.99)
- Shipping: \$86.20 Shipping & Import Fees Deposit to Vietnam
- Delivery: Monday, May 8. Order within 10 hrs 55 mins
- Stock: Only 16 left in stock - order soon
- Buttons: Add to Cart, Buy Now
- Payment: Secure transaction
- Ships from: Amazon.com
- Sold by: Amazon.com

Figure 28. Examine Amazon product information page

Developers should approach this problem in the same manner as the previous section. In summary, the scraper needs to carefully examine the HTML document using developer tools, debug the XPath expression through Scrapy shell, then implement the extraction codes into the spider.

```
def parse_product(self, response):

    product = Product()
    search_key = self.search_item

    product['asin'] = response.meta['asin']
    product['title'] = response.xpath("//span[@id='productTitle']/text()").get().strip()
    product['price'] = response.xpath("//span[@class='a-offscreen']/text()").get()[1:]
    product['price_currency'] = response.xpath("//span[@class='a-price-symbol']/text()").get()
    product['review_count'] = response.xpath("//span[@id='acrCustomerReviewText']/text()").get()
    product['rating'] = response.xpath('//*[id="acrPopover"]/@title').extract_first()
    product['url'] = response.url
    product['search_key'] = search_key

    yield product
```

Figure 29. parse_product() code

We are going to create a new parse function called **parse_product**:

- *product* is the data model that we created earlier for the engine
- *title*, *price*, *price_currency*, *review_count*, and *rating* are the collected data using XPath
- *asin*, *search_key*, and *url* are variables from other parse functions

For the testing, it is recommended to test each function separately before combining them by using the *callback* argument. *asin*, *search_key* will not be in the output because they are from the initial parse function.

```
2023-04-09 10:43:05 [scrapy.core.scrapers] DEBUG: Scraped from <200 https://www.amazon.com/dp/B08GPCH9C3
{'price': '334.91',
 'price_currency': '$',
 'rating': '4.2 out of 5 stars',
 'review_count': '25 ratings',
 'title': 'ASRock X670E Steel Legend Support AMD AM5 RYZEN 7000 Series '
         'Processors Motherboard',
 'url': 'https://www.amazon.com/dp/B08GPCH9C3'}
2023-04-09 10:43:05 [scrapy.core.engine] INFO: Closing spider (finished)
```

Figure 30. Debugging with Scrapy log #2

4.2.3 Scraping multiple products

After the scraper satisfied with the extracted information, enabling `parse_product` function into the callback argument of the initial parse function will allow the spider to parse every single product on the listing.

```
def parse(self, response):
    product_listing = response.xpath('//*[@@data-asin]')
    for product in product_listing:
        asin = product.xpath('@data-asin').extract_first()
        product_address = f"https://www.amazon.com/dp/{asin}"
        yield response.follow(product_address,
                              callback=self.parse_product,
                              meta={'asin': asin, 'page_location': response.url})
```

Figure 31. Adding scraping multiple products feature

The following output is generated from the keyword “processor”. From the engine log in Figure 32, we can see from the log that we have been able to scrape 18 items in total of 23 pages. There are some missing pages because few of them are not actually Amazon product page and they also have `@data-asin` attribute.

```
url: https://www.amazon.com/dp/B08166SLDF
2023-04-09 16:23:43 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.amazon.com/dp/B08166SLDF> (referer: https://www.
2023-04-09 16:23:43 [scrapy.core.scrapers] DEBUG: Scraped from <200 https://www.amazon.com/dp/B08166SLDF>
{'asin': 'B08166SLDF',
 'price': '169.00',
 'price_currency': '$',
 'rating': '4.8 out of 5 stars',
 'review_count': '19,735 ratings',
 'search_key': 'processor',
 'title': 'AMD Ryzen 5 5600X 6-core, 12-Thread Unlocked Desktop Processor with
        Wraith Stealth Cooler',
 'url': 'https://www.amazon.com/dp/B08166SLDF'}
2023-04-09 16:23:46 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.amazon.com/dp/B08164VTWH> (referer: https://www.
2023-04-09 16:23:46 [scrapy.core.scrapers] DEBUG: Scraped from <200 https://www.amazon.com/dp/B08164VTWH>
{'asin': 'B08164VTWH',
 'price': '349.00',
 'price_currency': '$',
 'rating': '4.8 out of 5 stars',
 'review_count': '8,796 ratings',
 'search_key': 'processor',
 'title': 'AMD Ryzen 9 5900X 12-core, 24-Thread Unlocked Desktop Processor',
 'url': 'https://www.amazon.com/dp/B08164VTWH'}
2023-04-09 16:23:46 [scrapy.extensions.logstats] INFO: Crawled 23 pages (at 23 pages/min), scraped 18 items (at 18 items/min)
```

Figure 32. Debugging with Scrapy log #3

Another feature that programmer needs to implement is scraping multiple pages. Scraper usually approaches this problem by examining the “next” button. They setup a callback loop that automatically navigates to the next page if the HTML element of the button is available. The loop will stop if it reaches the final page meaning that the button is not accessible in this situation.

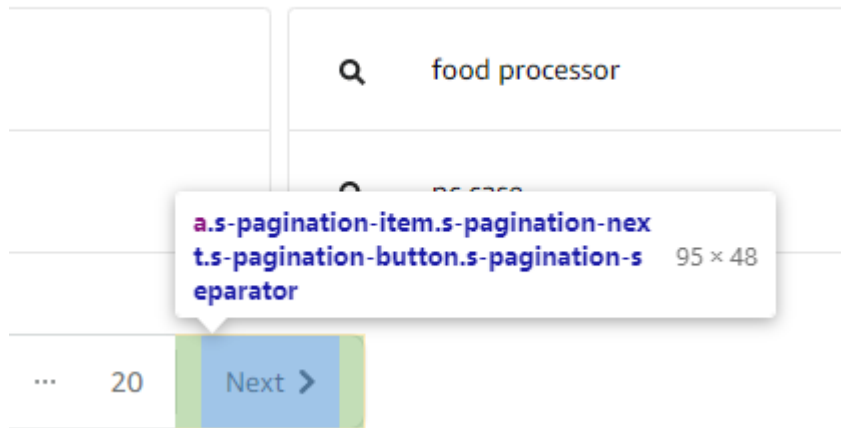


Figure 33. Inspecting "next" button

The author will attempt to add this logic into the **parse()** function because the “next” button is still an element inside the item catalog.

```
def parse(self, response):
    search_item = self.search_item
    page_counter = 1
    next_button = response.xpath("//a[contains(@aria-label, 'Go to next page')]/@href").get()
    if next_button and page_counter < 3:
        page_counter = page_counter + 1
        next_page = f'https://www.amazon.com/s?k={search_item}&page={page_counter}'
        yield response.follow(next_page, callback=self.parse)
```

Figure 34. Adding scraping multiple pages feature

- *search_item* is the class variable that contains the searched item. This will be used to build the URL address
- *page_counter* will be used to track the page for the engine and construct the page address. This variable can also be served as a page limiter
- *next_button* is the availability of the button
- *next_page* contains the URL address format of the pages

After adding the codes, the **parse()** function should look similar to the following.

```
def parse(self, response):

    product_listing = response.xpath('//*[@data-asin]')
    for product in product_listing:
        asin = product.xpath('@data-asin').extract_first()
        product_address = f"https://www.amazon.com/dp/{asin}"
        yield response.follow(product_address,
                              callback=self.parse_product,
                              meta={'asin': asin, 'page_location': response.url})

    search_item = self.search_item
    page_counter = 1
    next_button = response.xpath("//a[contains(@aria-label, 'Go to next page')]/@href").get()
    if next_button and page_counter < 3:
        page_counter = page_counter + 1
        next_page = f"https://www.amazon.com/s?k={search_item}&page={page_counter}'
        yield response.follow(next_page, callback=self.parse)
```

Figure 35. Final version of parse() function

This concludes the extraction process of the project. In summary, starting the spider will prompt the user an input which is the targeted item to collect information on Amazon. After entering the desired item, the spider will navigate to the corresponding product listing to collect all available product URL addresses. These addresses will be visited by the spider to collect all the required data. When there is no item left, the engine starts a new parse operation on the next page. The process will stop when it reaches the destination page.

4.3 Data transformation

One of the main goals of data scraping is to extract structured data from unstructured source which is typically web pages in this case. Data cleaning or data formatting is an important process in scraping. Scrapy item pipelines provide developers a powerful programming interface for the cleaning process. In this section, programmers will interact mainly with the pipelines.py file and control the pipelines through the settings.py.

Figure 36 shows the examples of what programmers can do in the transformation phase of Scrapy:

```

pipelines.py X
scrapy_amazon > pipelines.py > ExportToDatabase > __init__
6  from itemadapter import ItemAdapter
7  from scrapy.exceptions import DropItem
8
9  class DuplicateCheck:
10     def __init__(self):
11         self.names_seen = set()
12     def process_item(self, item, spider):
13         adapter = ItemAdapter(item)
14         if adapter['title'] in self.names_seen:
15             raise DropItem("Duplicate item found: {item!r}")
16         else:
17             self.names_seen.add(adapter['title'])
18         return item
19
20     class ItemValidation:
21         def process_item(self, item, spider):
22             adapter = ItemAdapter(item)
23             if adapter['search_key'].lower() not in adapter['title'].lower():
24                 raise DropItem("Item from advertisement")
25             else:
26                 return item
27
28     class ReviewToInteger:
29         def process_item(self, item, spider):
30             adapter = ItemAdapter(item)
31             adapter['review_count'] = re.sub(r'^0-9', '', adapter['review_count'])
32             return item

```

Figure 36. Creating transform features

In order to enable the process, scrapers have to initialize them in settings.py. The priority of the execution is represented by numbers similar to the figure below. The pipeline process will execute from low to high order and start with the lowest value, which is *DuplicateCheck* in the following configuration.

```

# Configure item pipelines
# See https://docs.scrapy.org/en/latest/topics/item-pipeline.html
ITEM_PIPELINES = {
    'scrapy_amazon.pipelines.DuplicateCheck': 100,
    'scrapy_amazon.pipelines.ItemValidation': 200,
    'scrapy_amazon.pipelines.ReviewToInteger': 300,
}

```

Figure 37. Adding features to pipelines

Scrapy divides the transformation process into multiple operations and these processes are contained in a Python class. The class receives the collected item and performs various actions over them. *ItemAdapter* is the core component of this process which returns the scraped data by spiders into the pipeline engine. The developer can interact with any data by calling adapter syntax. These are the usage of the classes from Figure 45:

- *DuplicateCheck* drops any duplicate item using the title of the product.
- *ItemValidation* drops any unrelated item because there can be an item from Amazon advertisement. If the title does not contain the search keyword, the item will be dropped
- *ReviewToInteger* removes all non-numeric characters from `review_count`.
This enables the user the ability to perform any calculation on the data

The writer did not perform any complicated transformation in this project because the product information on Amazon is fairly organized and well-structured already. The example showcases several uses of the pipelines to customize our data. These includes cleaning the data, validating data, dropping duplicate values into the desirable format. One of the advantages of Scrapy is providing a programming interface for this process. Transform data through programming provides extremely high flexibility and customization in general.

4.4 Data loading

This is the final phase of a data scraping project. In the loading phase, the scraper will deliver the structured data into the desired format. Scrapy provides two solutions for this process either using command-line or programming. The command-line method is performed by adding additional argument into the crawl command while programming method utilizes built-in feature of Scrapy and several libraries to produce an output source. The author is going to examine the capabilities of both methods in detail in this section.

Table 8. Comparison of command-line and programming

Command-line	Programming
Can export to the following format: CSV, JSON, XML	Can export to the following format: CSV, JSON, XML and database
Supports only local environment	Supports both local and cloud platform
No special customization	Customizable with other Python libraries
Does not require any additional setup to function	Requires additional setup to function

4.4.1 Command-line method

The example shows how to create an output using one of these command lines:

```
scrapy crawl Products --output filename.csv
```

```
scrapy crawl Products -o filename.csv
```

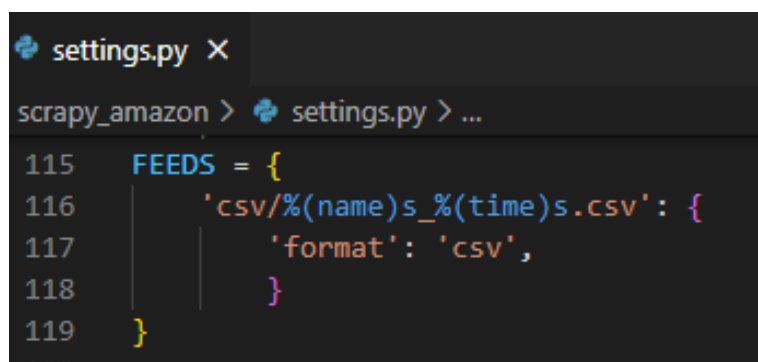
```
scrapy crawl Products -O filename.csv
```

to overwrite the file has the same name

Users can choose to export to another format by replacing the extension of the file. This method is known for its convenience because no additional setup is required. It is most used for debugging purposes during the development stage.

4.4.2 Programming method

The programming method utilizes the feed export features of Scrapy. This feature allows developers to generate feeds with the collected data. Developers need to add the following code into settings.py to enable Scrapy feeds.



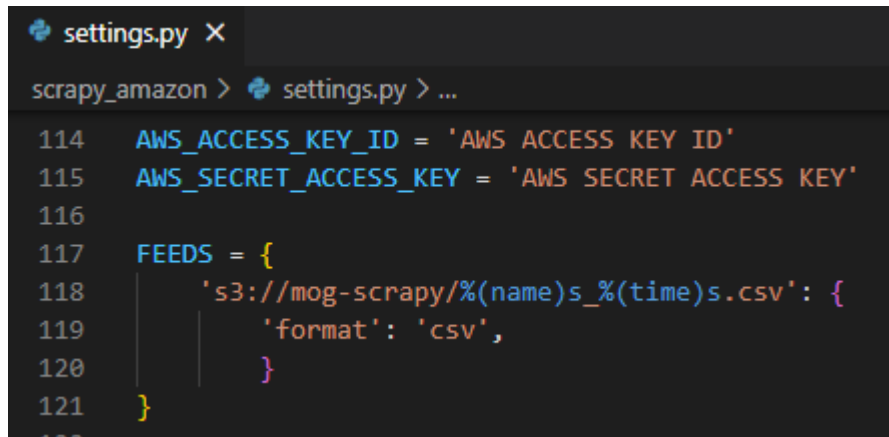
```

settings.py X
scrapy_amazon > settings.py > ...
115 FEEDS = {
116     'csv/%(name)s_%(time)s.csv': {
117         'format': 'csv',
118     }
119 }

```

Figure 38. Scrapy feeds

Feed export can also deliver output to cloud platforms such as AWS (Amazon Web Services) or GCP (Google Cloud Platform). We are going to implement a feature that exports the scraped data to AWS S3, a storage service that stores data as objects within buckets. The initial step is to install `botocore` which provides a low-level interface to most AWS services. Users can install this package by using pip installer: `pip install botocore`



```
settings.py X
scrapy_amazon > settings.py > ...
114 AWS_ACCESS_KEY_ID = 'AWS ACCESS KEY ID'
115 AWS_SECRET_ACCESS_KEY = 'AWS SECRET ACCESS KEY'
116
117 FEEDS = {
118     's3://mog-scrapy/%(name)s_%(time)s.csv': {
119         'format': 'csv',
120     }
121 }
```

Figure 39. Export data to AWS S3 using Scrapy feeds

This project assumes that the scraper had already setup the AWS account and S3 service. Access key ID and secret access key are two components that users need in order to access S3. After that, running the `crawl` command of the spider will automatically send the result to the selected S3 bucket.

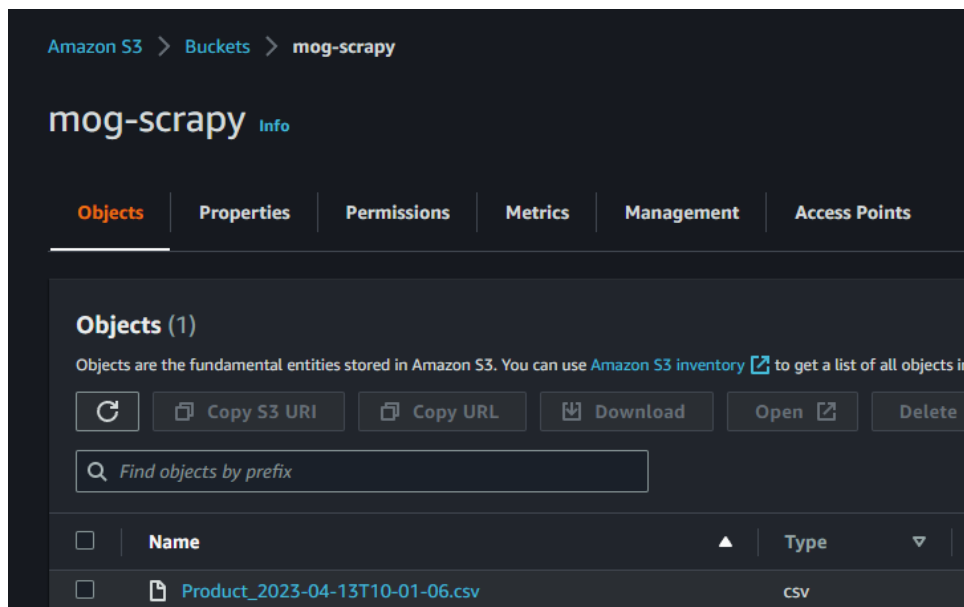
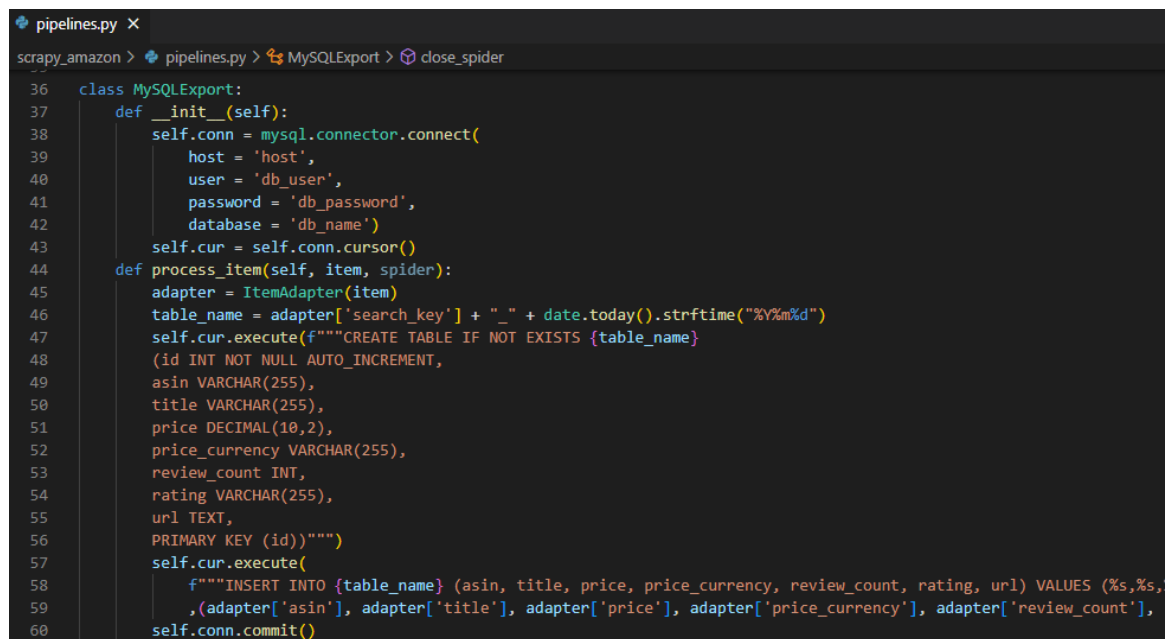


Figure 40. Exported data in S3

Figure 40 displays the extracted CSV on AWS S3. While the programming method can work decently with cloud platforms, the method is also capable of delivering scraped data to the database. Although Scrapy does not have a built-in database export feature, this feature can still be implemented by utilizing the item pipelines and SQL connector libraries. MySQL connector will be used because MySQL is used to store the collected material in this project. The first step is to install the connector and import the library into Scrapy pipelines:

```
pip install mysql-connector-python
```

```
import mysql.connector
```



```

pipelines.py X
scrapy_amazon > pipelines.py > MySQLExport > close_spider

36 class MySQLExport:
37     def __init__(self):
38         self.conn = mysql.connector.connect(
39             host = 'host',
40             user = 'db_user',
41             password = 'db_password',
42             database = 'db_name')
43         self.cur = self.conn.cursor()
44     def process_item(self, item, spider):
45         adapter = ItemAdapter(item)
46         table_name = adapter['search_key'] + "_" + date.today().strftime("%Y%m%d")
47         self.cur.execute(f"CREATE TABLE IF NOT EXISTS {table_name}
48             (id INT NOT NULL AUTO_INCREMENT,
49             asin VARCHAR(255),
50             title VARCHAR(255),
51             price DECIMAL(10,2),
52             price_currency VARCHAR(255),
53             review_count INT,
54             rating VARCHAR(255),
55             url TEXT,
56             PRIMARY KEY (id))")
57         self.cur.execute(
58             f"INSERT INTO {table_name} (asin, title, price, price_currency, review_count, rating, url) VALUES (%s,%s,
59             ,{adapter['asin']}, {adapter['title']}, {adapter['price']}, {adapter['price_currency']}, {adapter['review_count']},
60             self.conn.commit()

```

Figure 41. Adding export to database feature

As we already know, we have to include a new feature to Scrapy pipeline as a Python class. **MySQLExport** function includes a MySQL database authentication and a SQL query to insert the scraped data. ItemAdapter is used to pass the data into the query while the database connector handles the table creation and data delivery. The programmer also needs to include the class in the settings.py pipeline section to activate the SQL export feature.

4.4.3 Export result

The information from the below figure is generated using the search keyword “motherboard” on Amazon.

The screenshot displays the output of a data scraping process for motherboards. It shows three different export formats: JSON, CSV, and XML, along with a table view of the data stored in a database.

JSON Output:

```

1
2 550", "rating": "4.7 out of 5 stars", "url": "https://www.amazon.com/dp/B00213KL
3 nks, Addressable Gen 2 RGB Header and Aura Sync", "price": "169.99", "price_curr
4 L-W321-PL12BL-B,Black", "price": "148.99", "price_currency": "$", "review_count"
5 "review_count": "892", "rating": "4.4 out of 5 stars", "url": "https://www.ama
6 "price_currency": "$", "review_count": "128", "rating": "4.4 out of 5 stars", "U

```

CSV Output:

```

1 asin,price,price_currency,rating,review_count,search_key,title,url
2 B00213KL5I,6.03,$,4.7 out of 5 stars,650,motherboard,StarTech.com 6-32 Bl$,
3 B08F9VP2RC,169.99,$,4.6 out of 5 stars,2049,motherboard,"ASUS ROG Strix le_c
4 B0B2L1Z4XL,148.99,$,4.5 out of 5 stars,1518,motherboard."Thermaltake TOUing

```

XML Output:

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <items>
3 <item><asin>B08T6H14RR</asin><title>ASUS TUF Gaming Z590-Plus W:G Strix Z
4 <item><asin>B09JZGTYXJ</asin><title>GIGABYTE Z690 UD AX DDR4 (L4520M S2H

```

Database Table:

id	asin	title	price	price_currency	review_count	rating
1	B07Q2HPXTV	Thermaltake Pacific C360 De	314.13	\$	347	4.1 out of 5 stars
2	B09S6NQPM6	GIGABYTE H610M S2H DDR4	89.99	\$	125	4.5 out of 5 stars
3	B0876HBFVC	MSI MPG Z490 Gaming Plus	119.99	\$	1429	4.6 out of 5 stars
4	B0B2L1Z4XL	Thermaltake TOUGHLIQUID	148.99	\$	1519	4.5 out of 5 stars
5	B089D1YG11	MSI B550M PRO-VDH WiFi P	119.99	\$	1963	4.4 out of 5 stars
6	B088W7RKVZ	ASUS ROG Strix B550-F Gam	191.99	\$	4431	4.7 out of 5 stars
7	B08F7BHDLY	Gigabyte A520I AC (AMD Ry	109.99	\$	403	4.6 out of 5 stars
8	B09J1RM86X	ASUS ROG Strix Z690-A Gam	308.95	\$	506	4.5 out of 5 stars
9	B08F7HPJ4F	Gigabyte A520M S2H (AMD	74.99	\$	503	4.5 out of 5 stars
10	B0B9G2ZP4K	Intel Core i9 (12th Gen) i9-12	967.92	\$	128	4.4 out of 5 stars
11	B089CWDHFZ	MSI MAG B550 TOMAHAWK	169.99	\$	4616	4.7 out of 5 stars
12	B09LHBXM3Z	Gigabyte Z690 AORUS ULTR	348.88	\$	22	4.1 out of 5 stars
13	B089CZSQB4	MSI B550-A PRO ProSeries N	139.99	\$	2152	4.6 out of 5 stars

Figure 42. Scraped data in JSON, CSV, XML and SQL database

Figure 42 shows the final product of the project. The outputs include several popular data storing formats such as JSON, CSV, XML, and database. The scraped data has a delightful format corresponding to our programming during the transformation process.

4.5 Optional settings

When scrapers try to scrape a particular site, there might be a situation where the spider is blocked or blacklisted by the website owner. In this project case, Amazon website is not an exception. Many websites have a system that automatically blocks the following scenario:

- The website receives too many requests from an IP address
- The website receives too many concurrent requests from an IP address
- The website receives requests from an IP address that has an unknown or suspicious request header
- The website receives a request from a bot which is identified through the address user-agent. User-agent is a string that displays the information about the requesting application and operating system to the server
- The IP address violates the rules from the robots.txt file

Fortunately, Scrapy provides multiple configurations to resolve these problems. All of these settings can be found in the settings.py. The following configurations are essential to many scraping projects:

- `ROBOTSTXT_OBEY`: Enforces the rule from robots.txt to the spiders. This option is enabled by default. Scrapers should keep this on all the time
- `CONCURRENT_REQUESTS`: Controls the number of requests that the Scrapy engine parses concurrently. The default value is 16. Setting this value too high might cause heavy traffic to the scraped website. The user should adjust this value accordingly to the situation.
- `DOWNLOAD_DELAY`: Controls the delay between each request. The default value is 0. If scrapers are getting blocked temporarily, they should try to adjust this value
- `DEFAULT_REQUEST_HEADERS`: Request headers are used to provide information about the HTTP request context. The configuration is empty by default which makes the Scrapy engine typically suspicious to websites. Therefore, developers should fill in the content of this configuration to avoid being blocked by the website owner. There are many instructions or blogs to fill these headers on the Internet. The following figure shows the

headers that the author uses for this project. More information about each component can be found on Mozilla developer blog:

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>

```
# Override the default request headers:
DEFAULT_REQUEST_HEADERS = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:98.0) Gecko/20100101 Firefox/98.0",
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8",
    "Accept-Language": "en-US,en;q=0.5",
    "Accept-Encoding": "gzip, deflate",
    "Connection": "keep-alive",
    "Upgrade-Insecure-Requests": "1",
    "Sec-Fetch-Dest": "document",
    "Sec-Fetch-Mode": "navigate",
    "Sec-Fetch-Site": "none",
    "Sec-Fetch-User": "?1",
    "Cache-Control": "max-age=0",
}
```

Figure 43. DEFAULT_REQUEST_HEADERS setting

- **DOWNLOADER_MIDDLEWARES**: The programmer can activate a downloader middleware by adding the component into this option. Downloader middleware allows scrapers to alternate requests and responses. The middlewares can set up a rotating proxies service for Scrapy. This service provides multiple different IP addresses for scraping instead of the user's IP address. These addresses will be rotate frequently to bypass the IP checking from the website.

5 CONCLUSION

The thesis provided data scraping as a method to gather the information on the Internet. The reason why data scraping gains so much popularity nowadays is the rapid development of the data industry and the Internet. The paper also introduced many aspects of web scraping such as definition, properties, legal issues. Additionally, the author examined in detail several useful frameworks and libraries for gathering information on modern websites.

Another major topic of this thesis is Scrapy, one of the most efficient scraping frameworks on the market at the moment. We have been able to scrape Amazon, a website that has a decent amount of complexity, in order to learn more about

the functionality of Scrapy and how data scraping works in a project environment. This data scraping application will collect the basic product information from the user's selected item. Then the scraped material is transformed into a desirable format for the consumer. We learn how to set up various configurations that Scrapy provides to make the engine works more effectively. Finally, our data is stored in multiple different formats such as CSV, JSON, XML, and SQL database. Moreover, we also configure Scrapy to deliver the data into S3, a popular object storage on Amazon Web Services. This proves the ability to work with cloud platforms of Scrapy.

Overall, data scraping is a reliable process to assemble information from the online world while Scrapy is the most efficient tool for the job. However, a reasonable effort is required to learn and utilize both data scraping and Scrapy properly. In contrast, mastering this craft not only grants you the ability to gather a large amount of information on the Internet but also significant boosts your confidence as a data worker or a developer.

REFERENCES

Amazon Web Services. What is ETL (Extract Transform Load)?. Web page. Available at: <https://aws.amazon.com/what-is/etl/>
[Accessed April 2023]

ComTec. Top 10 Advantages of Using Data Analytics in the Retail Industry. Web page. Available at: <https://www.comtecinfo.com/rpa/top-10-advantages-of-using-data-analytics-in-the-retail-industry/>
[Accessed April 2023]

Databricks. Extract Transform Load (ETL). Web page. Available at: <https://www.databricks.com/glossary/extract-transform-load>
[Accessed April 2023]

GeeksforGeeks. Components of a URL. Web page. Available at: <https://www.geeksforgeeks.org/components-of-a-url/>
[Accessed April 2023]

Learning A-Z. Data in Education. Web page. Available at: <https://www.learninga-z.com/site/resources/breakroom-blog/data-in-education>
[Accessed April 2023]

Mozilla. HTML basics. Web page. Available at: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics
[Accessed April 2023]

Mozilla. HTTP headers. Web page. Available at: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>
[Accessed April 2023]

Mozilla. XPath axes. Web page. Available at: <https://developer.mozilla.org/en-US/docs/Web/XPath/Axes>
[Accessed April 2023]

Scrapy. Architecture overview. Web page. Available at: <https://docs.scrapy.org/en/latest/topics/architecture.html>
[Accessed April 2023]

ScrapingBee. Practical XPath for Web Scraping. Web page. Available at: <https://www.scrapingbee.com/blog/practical-xpath-for-web-scraping/>
[Accessed April 2023]

Splash. Splash documentation. Web page. Available at: <https://splash.readthedocs.io/en/stable/faq.html>
[Accessed April 2023]

TEG Analytics. Top 5 Benefits of Data Analytics in Healthcare. Web page. Available at:
<https://teganalytics.com/top-5-benefits-of-data-analytics-in-healthcare/>
[Accessed April 2023]

W3Schools. HTML tags. Web page. Available at:
<https://www.w3schools.com/tags/>
[Accessed April 2023]

W3Schools. XPath syntax. Web page. Available at:
https://www.w3schools.com/xml/xpath_syntax.asp
[Accessed April 2023]

Web Scraper. CSS selector. Web page. Available at:
<https://webscraper.io/documentation/css-selector>
[Accessed April 2023]

Ryan, M. 2018. Web Scraping with Python: Collecting More Data from the Modern Web. 2nd Edition. O'Reilly Media. Available at:
<https://www.amazon.com/Web-Scraping-Python-Collecting-Modern/dp/1491985577>
[Accessed April 2023]

Wikipedia. Python (programming language). Web page. Available at:
[https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language))
[Accessed April 2023]

Wikipedia. Markup language. Web page. Available at:
https://en.wikipedia.org/wiki/Markup_language
[Accessed April 2023]

Wikipedia. Web scraping. Web page. Available at:
https://en.wikipedia.org/wiki/Web_scraping
[Accessed April 2023]

Wikipedia. Document Object Model. Web page. Available at:
https://en.wikipedia.org/wiki/Document_Object_Model
[Accessed April 2023]

Yodlee. The Power of Retail Analytics. Web page. Available at:
<https://www.yodlee.com/data-analytics/big-data-retail-analytics>
[Accessed April 2023]

LIST OF TABLES

Table 1. Popular tag in scraping	6
Table 2. XPath expressions.....	11
Table 3. XPath functions	11
Table 4. XPath axes	11
Table 5. XPath examples	12
Table 6. robots.txt rules	20
Table 7. Spider types	23
Table 8. Comparison of command-line and programming.....	38

LIST OF FIGURES

Figure 1. Illustration of ETL process. Extract Transform Load (ETL). Databrick. n.d. Available at https://www.databricks.com/glossary/extract-transform-load . Accessed April 2023.....	2
Figure 2. Stack Overflow example #1.....	4
Figure 3. HTML element. HTML basics. Mozilla. n.d. Available at https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics . Accessed April 2023	5
Figure 4. HTML attribute. HTML basics. Mozilla. n.d. Available at https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics . Accessed April 2023	7
Figure 5. Developer tool example.....	7
Figure 6. Example of a DOM tree.....	8
Figure 7. DOM tree characteristics.....	9
Figure 8. XPath components.....	10
Figure 9. Scrapy dataflow. Architecture overview. Scrapy. n.d. Available at https://docs.scrapy.org/en/latest/topics/architecture.html . Accessed April 2023..	14
Figure 10. Stack Overflow example #2.....	16
Figure 11. robots.txt example	19
Figure 12. Project architecture	21
Figure 13. Project folder	22
Figure 14. Initial the spider	23
Figure 15. Project flow.....	23
Figure 16. URL components. Components of a URL. GeeksforGeeks. n.d. Available at https://www.geeksforgeeks.org/components-of-a-url/ . Accessed April 2023	24
Figure 17. Amazon search #1	24
Figure 18. Search URL.....	25
Figure 19. Amazon search #2	25
Figure 20. Implement search function	26
Figure 21. Inspect Amazon product listing #2.....	27

Figure 22. Inspect Amazon product listing #3.....	27
Figure 23. Scrapy shell interface	28
Figure 24. Testing XPath expression with Scrapy shell.....	28
Figure 25. parse() code	29
Figure 26. Debugging with Scrapy log #1	30
Figure 27. Initialize Scrapy data model	31
Figure 28. Examine Amazon product information page.....	31
Figure 29. parse_product() code	32
Figure 30. Debugging with Scrapy log #2.....	32
Figure 31. Adding scraping multiple products feature	33
Figure 32. Debugging with Scrapy log #3.....	33
Figure 33. Inspecting "next" button.....	34
Figure 34. Adding scraping multiple pages feature	34
Figure 35. Final version of parse() function	35
Figure 36. Creating transform features.....	36
Figure 37. Adding features to pipelines	36
Figure 38. Scrapy feeds	38
Figure 39. Export data to AWS S3 using Scrapy feeds	39
Figure 40. Exported data in S3.....	39
Figure 41. Adding export to database feature	40
Figure 42. Scraped data in JSON, CSV, XML and SQL database	41
Figure 43. DEFAULT_REQUEST_HEADERS setting.....	43