

**Spiders**

* **Definition**: Spiders are custom classes in Scrapy where you define how to scrape websites, which data to collect, and how to follow links.
* **Function**: Spiders generate **requests** to be processed by the Scrapy Engine. These requests specify the URL(s) to be visited.
* **Interaction**: After receiving a **response** from the downloader, spiders extract the relevant data (like product information) or generate new requests to follow additional links.

**2. Scheduler**

* **Definition**: The scheduler is responsible for managing the order in which requests are processed.
* **Function**: It receives requests from the **engine** and enqueues them for later processing. It ensures that Scrapy does not overwhelm websites by controlling the request rate.
* **Interaction**: It sends requests back to the engine when it is time for them to be processed and downloaded.

**3. Engine**

* **Definition**: The engine is the core component of Scrapy, orchestrating the data flow between all other components.
* **Function**: It receives requests from spiders, sends them to the **scheduler**, retrieves them for processing, and passes them to the **downloader**. Once a response is received, it sends it back to the spider or item pipeline.
* **Interaction**: It controls the overall workflow, ensuring requests and responses move between components.

**4. Downloader**

* **Definition**: The downloader is responsible for fetching web pages from the Internet.
* **Function**: It handles the requests sent from the engine and retrieves the actual content (HTML, images, JSON, etc.) from the specified URL.
* **Interaction**: It returns the downloaded **response** back to the engine for further processing.

**5. Downloader Middleware**

* **Definition**: This is a layer between the engine and the downloader.
* **Function**: It allows you to process or modify requests and responses before sending them to the downloader or before the engine processes them.
* **Interaction**: It can be used for tasks like adding user-agent headers, handling cookies, or managing retries.

**6. Spider Middleware**

* **Definition**: Spider middleware sits between the engine and the spider.
* **Function**: It allows you to modify requests and responses before they reach the spider or after the spider generates them.
* **Interaction**: It can be used for tasks such as filtering or preprocessing data before passing it to the spider.

**7. Response**

* **Definition**: A response is the content retrieved by the downloader from the specified URL.
* **Function**: Once the response is received, it is passed back to the **spider** for data extraction.
* **Interaction**: The spider processes this response, extracting relevant data or issuing new requests for further links to follow.

**8. Item Pipelines**

* **Definition**: Item pipelines process the scraped data extracted by the spiders.
* **Function**: Once an item (i.e., scraped data) is generated by a spider, it is sent through the item pipeline for tasks like data cleaning, validation, or storing in a database.
* **Interaction**: The processed items are sent to a storage system (e.g., database, file).

**Data Flow in Scrapy:**

1. **Requests**: Spiders generate initial requests for URLs to scrape.
2. **Scheduler**: The requests are passed to the scheduler, which queues them up for downloading.
3. **Downloader**: The engine sends the queued requests to the downloader, which fetches the content from the web.
4. **Responses**: The downloaded content is passed back as responses to the engine.
5. **Spider Processing**: The engine hands the responses over to the spiders, which extract relevant data or generate new requests.
6. **Item Processing**: Extracted items are sent through the item pipeline for further processing.
7. **New Requests**: If the spider generates new requests, they are sent back to the scheduler, repeating the process.

**Summary:**

* **Spiders** define what data to extract.
* **Scheduler** manages the order of requests.
* **Downloader** fetches the web content.
* **Engine** coordinates everything.
* **Middleware** allows customization at different stages.
* **Item Pipelines** process and store the extracted data.

This architecture ensures an efficient and scalable way of crawling and scraping data from the web.

Sample code for scrapy

#Install Scrapy

**pip install scrapy**

#books\_scraper should be changed it is your project name rest #scrapy (This will create a project folder named books\_scraper.)#

**scrapy startproject books\_scraper**

#Navigate into the spiders’ directory within the project and create a Python file for your spider. Let's name it books\_spider.py

**cd books\_scraper/books\_scraper/spiders**

**touch books\_spider.py**

**import scrapy**

**class BooksSpider(scrapy.Spider):**

**name = 'books'**

**start\_urls = ['http://books.toscrape.com']**

**def parse(self, response):**

**# Extract the titles of books on the current page**

**for book in response.css('article.product\_pod'):**

**yield {**

**'title': book.css('h3 a::attr(title)').get(),**

**}**

**# Follow pagination links to scrape more books**

**next\_page = response.css('li.next a::attr(href)').get()**

**if next\_page:**

**yield response.follow(next\_page, self.parse)**

# To run the spider, navigate back to the project’s main directory and run:

**scrapy crawl books -o books.json**

#This command will run the spider and ( “-o” ) save the scraped book titles in a file called books.json.

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Scrapy** | **BeautifulSoup** |
| **Type of Tool** | Scrapy is a full-fledged web scraping framework. It includes tools for handling requests, parsing, and storing data. | BeautifulSoup is a parsing library that helps with extracting data from HTML or XML documents. It does not handle requests or large-scale scraping on its own. |
| **Usage Scope** | Best suited for large-scale scraping tasks, web crawling across multiple pages, and handling complex scraping workflows. | Best suited for small to medium-sized scraping tasks, especially when only parsing HTML from a single page is needed. |
| **Handling Requests** | Scrapy has a built-in mechanism to handle HTTP requests, manage responses, and follow links automatically (web crawling). | BeautifulSoup requires external libraries like requests or urllib to fetch the HTML content. It does not make HTTP requests by itself. |
| **B NSpeed and Efficiency** | Scrapy is highly optimized for speed and performance, capable of parallel requests and asynchronous processing. | BeautifulSoup is slower because it is primarily a parser. The speed depends on how you handle HTTP requests externally. |
| **Data Export and Storage** | Scrapy natively supports exporting data to formats like JSON, CSV, XML, and databases, making it powerful for data handling. | BeautifulSoup doesn't natively support data export. You need to write custom code to save parsed data in specific formats. |
| **Learning Curve** | Steeper learning curve due to its more advanced features like middleware, pipelines, and asynchronous processing. | Easier to learn for beginners as it focuses primarily on parsing HTML and is more straightforward for simple tasks. |
| **Pagination and Crawling** | Scrapy is designed for web crawling and can easily handle pagination and multi-page scraping out of the box. | BeautifulSoup is not designed for crawling, and you have to manually handle pagination by combining it with other libraries like requests. |
| **Community Support and Documentation** | Scrapy has extensive documentation and a large community, making it suitable for professional and large-scale projects. | BeautifulSoup has good documentation and community support but is generally considered for simpler tasks. |

**Illustrate the importance of crawler, scraper, and parser. List 3 types of parser.**

1. **Crawler**:
   * **Function**: A web crawler (or spider) systematically browses the web, finding and indexing URLs of interest.
   * **Importance**:
     + **Data Discovery**: Crawlers help discover pages and sites that might not be easily accessible. They find all links on a website, collecting and storing URLs for further analysis.
     + **Automation**: Automates the process of visiting websites, ensuring that a wide range of data is accessible for further scraping.
     + **Indexing and Searching**: Essential for search engines to build up-to-date indexes of the web, making information retrieval faster and more efficient.
2. **Scraper**:
   * **Function**: A scraper extracts data from the web content (HTML, JSON, etc.) discovered by the crawler.
   * **Importance**:
     + **Data Extraction**: Scrapers are used to pull specific data from websites (e.g., product details, prices, reviews) for analysis or integration into another system.
     + **Data Collection**: Automates the gathering of structured or unstructured data from the web, reducing manual efforts.
     + **Efficiency**: Helps in the collection of large volumes of data in an organized manner, useful in fields like e-commerce, research, and sentiment analysis.
3. **Parser**:
   * **Function**: A parser processes the scraped data and converts it into a structured format (like JSON, CSV, or XML).
   * **Importance**:
     + **Data Structuring**: Parsers convert raw HTML or XML into structured data that can be stored, queried, and analyzed efficiently.
     + **Accuracy**: Ensures that the right data fields are extracted and understood in the right context (for instance, distinguishing between product name and description).
     + **Data Transformation**: It enables transforming unstructured data into formats suitable for further use in machine learning models, databases, or analytical tools.

**Types of Parsers**

1. **HTML Parser**:
   * Parses HTML documents to extract meaningful data such as headings, paragraphs, and metadata. Tools like **BeautifulSoup** in Python are commonly used for HTML parsing.
2. **XML Parser**:
   * Reads and parses XML documents to extract data. XML parsers are widely used in configurations, data transmission, and documents with a defined structure. Python’s xml.etree.ElementTree module is a popular tool.
3. **JSON Parser**:
   * Handles the parsing of JSON data (JavaScript Object Notation), which is lightweight and widely used for APIs. This parser transforms JSON data into objects or dictionaries for easy access and manipulation.

**Conclusion:**

* **Crawler** helps in discovering data.
* **Scraper** extracts specific information.
* **Parser** organizes the extracted data for further use. Each plays a crucial role in the process of web scraping and data analysis.