1. In the last tutorial we learned [how to leverage the Scrapy framework](https://www.scrapingbee.com/blog/web-scraping-with-scrapy/) to solve common web scraping problems. Today we are going to take a look at Selenium (with Python ❤️ ) in a step-by-step tutorial.

Selenium refers to a number of different open-source projects used for browser automation. It supports bindings for all major programming languages, including our favorite language: Python.

The Selenium API uses the WebDriver protocol to control a web browser, like Chrome, Firefox or Safari. The browser can run either localy or remotely.

At the beginning of the project (almost 20 years ago!) it was mostly used for cross-browser, end-to-end testing (acceptance tests).

Now it is still used for testing, but it is also used as a general browser automation platform. And of course, it us used for web scraping!

Selenium is useful when you have to perform an action on a website such as:

* Clicking on buttons
* Filling forms
* Scrolling
* Taking a screenshot

It is also useful for executing Javascript code. Let's say that you want to scrape a Single Page Application. Plus you haven't found an easy way to directly call the underlying APIs. In this case, Selenium might be what you need.

2. **Web scraping**, **web harvesting**, or **web data extraction** is [data scraping](https://en.wikipedia.org/wiki/Data_scraping) used for [extracting data](https://en.wikipedia.org/wiki/Data_extraction) from [websites](https://en.wikipedia.org/wiki/Website). The web scraping software may directly access the [World Wide Web](https://en.wikipedia.org/wiki/World_Wide_Web) using the [Hypertext Transfer Protocol](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol) or a web browser. While web scraping can be done manually by a software user, the term typically refers to automated processes implemented using a [bot](https://en.wikipedia.org/wiki/Internet_bot" \o "Internet bot) or [web crawler](https://en.wikipedia.org/wiki/Web_crawler). It is a form of copying in which specific data is gathered and copied from the web, typically into a central local [database](https://en.wikipedia.org/wiki/Database) or spreadsheet, for later [retrieval](https://en.wikipedia.org/wiki/Data_retrieval) or [analysis](https://en.wikipedia.org/wiki/Data_analysis).

Web scraping a web page involves fetching it and extracting from it. Fetching is the downloading of a page (which a browser does when a user views a page). Therefore, web crawling is a main component of web scraping, to fetch pages for later processing. Once fetched, then extraction can take place. The content of a page may be [parsed](https://en.wikipedia.org/wiki/Parsing), searched, reformatted, its data copied into a spreadsheet or loaded into a database. Web scrapers typically take something out of a page, to make use of it for another purpose somewhere else. An example would be to find and copy names and telephone numbers, or companies and their URLs, or e-mail addresses to a list (contact scraping).

Web scraping is used for [contact scraping](https://en.wikipedia.org/wiki/Contact_scraping), and as a component of applications used for [web indexing](https://en.wikipedia.org/wiki/Web_indexing), [web mining](https://en.wikipedia.org/wiki/Web_mining) and [data mining](https://en.wikipedia.org/wiki/Data_mining), online price change monitoring and [price comparison](https://en.wikipedia.org/wiki/Comparison_shopping_website), product review scraping (to watch the competition), gathering real estate listings, weather data monitoring, [website change detection](https://en.wikipedia.org/wiki/Change_detection_and_notification), research, tracking online presence and reputation, [web mashup](https://en.wikipedia.org/wiki/Web_mashup), and [web data integration](https://en.wikipedia.org/wiki/Web_data_integration).

[Web pages](https://en.wikipedia.org/wiki/Web_page) are built using text-based mark-up languages ([HTML](https://en.wikipedia.org/wiki/HTML) and [XHTML](https://en.wikipedia.org/wiki/XHTML)), and frequently contain a wealth of useful data in text form. However, most web pages are designed for human [end-users](https://en.wikipedia.org/wiki/End-user_(computer_science)) and not for ease of automated use. As a result, specialized tools and software have been developed to facilitate the scraping of web pages.

Newer forms of web scraping involve monitoring data feeds from web servers. For example, [JSON](https://en.wikipedia.org/wiki/JSON) is commonly used as a transport storage mechanism between the client and the web server.

There are methods that some websites use to prevent web scraping, such as detecting and disallowing bots from crawling (viewing) their pages. In response, there are web scraping systems that rely on using techniques in [DOM](https://en.wikipedia.org/wiki/Document_Object_Model) parsing, [computer vision](https://en.wikipedia.org/wiki/Computer_vision) and [natural language processing](https://en.wikipedia.org/wiki/Natural_language_processing) to simulate human browsing to enable gathering web page content for offline parsing.

3. Indexes support the efficient execution of queries in MongoDB. Without indexes, MongoDB must perform a *collection scan*, i.e. scan every document in a collection, to select those documents that match the query statement. If an appropriate index exists for a query, MongoDB can use the index to limit the number of documents it must inspect.

Indexes are special data structures [[1]](https://www.mongodb.com/docs/manual/indexes/#footnote-b-tree) that store a small portion of the collection's data set in an easy to traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field. The ordering of the index entries supports efficient equality matches and range-based query operations. In addition, MongoDB can return sorted results by using the ordering in the index.

The following diagram illustrates a query that selects and orders the matching documents using an index:

Fundamentally, indexes in MongoDB are similar to indexes in other database systems. MongoDB defines indexes at the [collection](https://www.mongodb.com/docs/manual/reference/glossary/#std-term-collection) level and supports indexes on any field or sub-field of the documents in a MongoDB collection.

Default \_id Index

MongoDB creates a [unique index](https://www.mongodb.com/docs/manual/core/index-unique/#std-label-index-type-unique) on the [\_id](https://www.mongodb.com/docs/manual/core/document/#std-label-document-id-field) field during the creation of a collection. The \_id index prevents clients from inserting two documents with the same value for the \_id field. You cannot drop this index on the \_id field.

**NOTE**

In [sharded clusters](https://www.mongodb.com/docs/manual/reference/glossary/" \l "std-term-sharded-cluster), if you do *not* use the \_id field as the [shard key](https://www.mongodb.com/docs/manual/reference/glossary/#std-term-shard-key), then your application **must** ensure the uniqueness of the values in the \_id field to prevent errors. This is most-often done by using a standard auto-generated [ObjectId](https://www.mongodb.com/docs/manual/reference/glossary/" \l "std-term-ObjectId).

Create an Index

➤ Use the **Select your language** drop-down menu in the upper-right to set the language of the examples on this page.

To create an index in the [Mongo Shell](https://www.mongodb.com/docs/manual/tutorial/getting-started/), use [db.collection.createIndex()](https://www.mongodb.com/docs/manual/reference/method/db.collection.createIndex/" \l "mongodb-method-db.collection.createIndex).

|  |
| --- |
| db.collection.createIndex( <key and index type specification>, <options> ) |

MongoDB Shell

The following example creates a single key descending index on the name field:

|  |
| --- |
| db.collection.createIndex( { name: -1 } ) |

MongoDB Shell

The [db.collection.createIndex()](https://www.mongodb.com/docs/manual/reference/method/db.collection.createIndex/" \l "mongodb-method-db.collection.createIndex) method only creates an index if an index of the same specification does not already exist.

|  |  |
| --- | --- |
| [[1](https://www.mongodb.com/docs/manual/indexes/#ref-b-tree-id2)] | MongoDB indexes use a B-tree data structure. |

**Index Names**

The default name for an index is the concatenation of the indexed keys and each key's direction in the index ( i.e. 1 or -1) using underscores as a separator. For example, an index created on { item : 1, quantity: -1 } has the name item\_1\_quantity\_-1.

You can create indexes with a custom name, such as one that is more human-readable than the default. For example, consider an application that frequently queries the products collection to populate data on existing inventory. The following [createIndex()](https://www.mongodb.com/docs/manual/reference/method/db.collection.createIndex/" \l "mongodb-method-db.collection.createIndex) method creates an index on item and quantity named query for inventory:

|  |
| --- |
| db.products.createIndex( |
| { item: 1, quantity: -1 } , |
| { name: **"query for inventory"** } |
| ) |

You can view index names using the [db.collection.getIndexes()](https://www.mongodb.com/docs/manual/reference/method/db.collection.getIndexes/" \l "mongodb-method-db.collection.getIndexes) method. You cannot rename an index once created. Instead, you must drop and re-create the index with a new name.

Index Types

MongoDB provides a number of different index types to support specific types of data and queries.

**Single Field**

In addition to the MongoDB-defined \_id index, MongoDB supports the creation of user-defined ascending/descending indexes on a [single field of a document](https://www.mongodb.com/docs/manual/core/index-single/).

4. Set expressions are used to define the scope of a calculation. The central part of the set expression is the set modifier that **specifies a selection**. This is used to modify the user selection, or the selection in the set identifier, and the result defines a new scope for the calculation.

5. db.collection.aggregate([

{ $project : { "Tags.\_id" : 1 }},

{ $unwind : "$Tags" },

{ $match: {$or: [{"Tags.\_id":"tag1"},{"Tags.\_id":"tag2"}]}},

{ $group: {

\_id : "$\_id",

count: { $sum:1 }

}},

{$sort: {"count":-1}}

],

{

explain:true

}