1. Crawler Manager

It takes a set of URL's from Link Extractor and then sends the Next URL to the DNS resolver to obtain its corresponding IP address. This saves a lot of time because spiders do not have to send requests to DNS every time they want to download a page.

2. Robots.txt file — used to avoid web pages & check listed for web pages

Using this web authors express their wish as to which pages they want the crawlers to avoid.

3. Spider

They download robots.txt file and other pages that are requested by the crawler manager and permitted by web authors. The robots.txt files are sent to crawler manager for processing and extracting the URLs. It also sends other downloaded files to a central indexer.

4. Link Extractor

It look through the pages downloaded by the spiders, extracts URLs from the links in those pages and then sends the URLs to the crawler manager for downloading afterwards.

5.8.3 Common Uses of Web Crawler

1. Used by anyone looking for to collect information about what is available on public web pages.
2. To collect data so that when Internet surfers enter a search term on their site, they can quickly provide the surfer with relevant web sites.
3. Used to perform a textual analysis i.e. they may explore the Internet to determine what words are commonly used today.
4. Used for market researchers to determine and assess trends in a given market.
5. Used for the search engines and other users to regularly ensure that their databases are up-to-date.
6. Used to create a copy of all the visited pages for later processing by a search engine that will index the downloaded pages to provide fast searches.
7. Used for automating maintenance tasks on a web site, such as checking links, or validating HTML code.
8. Used to gather specific types of information from Web pages, such as harvesting e-mail addresses usually for spam.

5.9 Meta-crawler

University Question

Q. What are Meta Crawlers ? Explain with suitable example. **SPPU : Dec. 12, May 16, 8 Marks**

- Meta-crawler is one of the first meta search engines that leverages many of the web's best search engines simultaneously and return the most efficient search result.
- It is originally developed in 1994 at the University of Washington and was first released to the web in June 1995.
- Provides options to the user to search for images, video, news, yellow and white pages.

- Queries many of the web's top search engines like About, Ask Jeeves, FindWhat, LookSmart, Overture and so on simultaneously and retrieve the best search results across the internet and organize them in a uniform format. It also ranks the result according to the relevance.
- Most important feature of meta-crawler is the efficiency it provides over the other engines.
- Users are able to receive the relevant result from more than a dozen leading search engine. And the time required is same as the time required to search one engine independently.
- Meta-crawler is more likely than a regular search engine to obtain accurate result and to find relevant information.
- Meta-crawler combines and normalizes confidence scores in order to rank each result in a voted ordering and then present that result to the user.
- Meta-crawler has extended its meta search capabilities to include auctions, audio/MP3, image domain name and origin, newsgroups and directories.
- Meta-crawler offers 'power search' options associated with power search.
- The power search include the ability to select specific engines to search, the option to adjust the amount of time to wait for results, quantity of results per page, and the number of results per search source.
- Has capability to customize searches to produce more timely and more relevant results. The first level of customization consists of the "any" words and as a "phrase" option.
- In addition to the previous search specification, meta-crawler recognizes special search syntax allowing the searcher to return even more specific results.

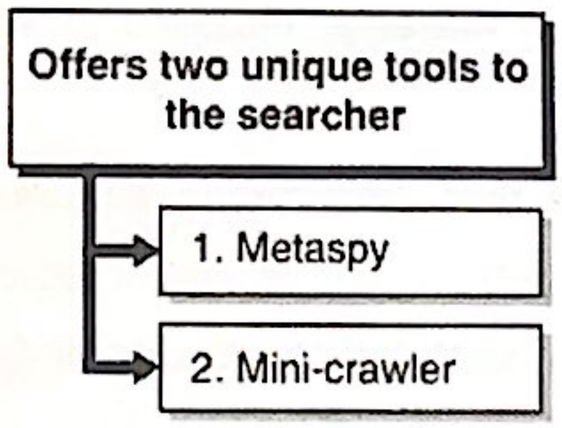


Fig. 5.9.1 : Tools to the searcher

1. **Meta-spy**

   Gives users a glimpse of what others are searching for at that very moment.

2. **Mini-crawler**

   It is valuable for those who are trying to manage space on their desktop. It is mini version of meta-crawler that gives user the constant access to the web's top meta search services.

5.10 Trends and Research Issues

Below are the trends and research issues in web searching :

- Distributed data which spans over many computers and platforms also the available bandwidth and reliability on the network interconnections varies widely
- High percentage of volatile data as new computers/sites/pages can be added and removed easily and we also have dangling links etc. when domain or file names change or disappear

- Scaling issues difficult to cope with due to large volume of data
- No conceptual organization of unstructured and redundant data, HTML pages are only semi-structured and repetition of data
- Special IR models tailored for the web are needed as web user queries are different.
- There is no editorial process and data can be false, invalid, poorly written, with many typos
- Also heterogeneous data like multiple media types, multiple formats, different languages and different alphabets
- More research work is needed to combine the structure and content in the queries and visual metaphors to visualize the answers.
- As on demand large number of web pages are created current techniques are not suitable to search dynamic pages.
- More implementation details must be improved for indexing.
- Improved mechanisms are needed to detect and eliminate repeated web pages.
- More research is required for searching non-textual multimedia objects.
- Better user interface is required for extraction of the main content of a page or the formulation of content-based queries.

While search engines have evolved continuously over the last 15 years, they still face many challenges, particularly with infrequent and detailed queries. A user that is seeking a phone number of their doctor will frequently be frustrated with the answers produced by the search engine. To cope with queries of this nature, search engines need to evolve further. They need to evolve to incorporate knowledge encoded in some form that it can be useful for ranking purposes.

Also important, it is frequently the case that determining the most relevant document for a query requires interpreting the content of the document and determining its central topics. That is, document understanding is an area little understood and which needs much more research, if search is to be further improved.

5.11 Introduction to Web Scraping

- Web scraping is commonly known as screen scraping, data mining, web harvesting, or similar variations.
- Web scraping is the practice of gathering unstructured data through any means except the program interacting with an API and store that into structured form
- Web scraping is accomplished by writing an automated program that queries a web server, requests data in the form of HTML and other files that compose web pages, and then parses that data to extract needed information
- Web scraping encompasses a wide variety of programming techniques and technologies, such as data analysis, natural language parsing, and information security etc.
- Below Fig. 5.11.1 illustrate the web scrapping

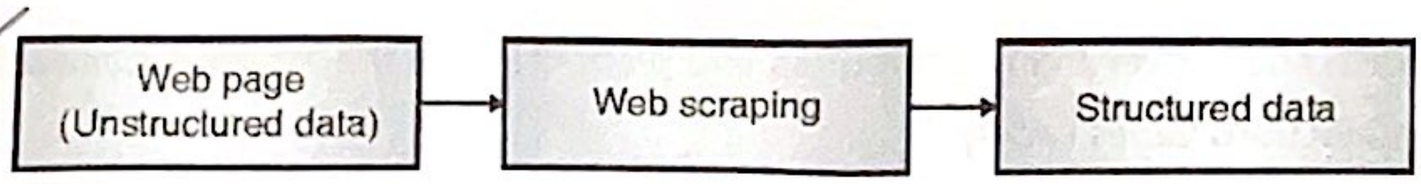


Fig. 5.11.1 : web scraping

5.12 Python for Web Scraping

Because of the following features Python is good for web scraping

- Python is easy to use and simple to code as no need to use semicolons, curly braces etc.
- Python has large collection of modules e.g Numpy, Matplotlib, Pandas, Sciklit which provides method and services for all the applications. And because of that it is suitable for scraping and manipulation of extracted data.
- Python is dynamically typed i.e. we don't need to define the data types of the variable and we can use directly whenever required. Because of dynamic type it saves time and make the job faster.
- Python has easily understandable syntax. Python syntax is expressive and easily readable, and the indentation used in Python also helps the user to differentiate between different scope/blocks in the code.
- In python small code is required for the larger task and it saves the time. And as web scraping is used to save the time python is the best choice for the same..

5.13 Requests

- Requests module is used for making HTTP requests to a specific URL and returns the response.
- Python requests module provide inbuilt functionalities for managing both the request and response.

5.13.1 Installation

Requests module is installed by using the command-

Pip install requests

Installation depends on the type of operating system, the basic command above is anywhere to open the command terminal and run it.

5.13.2 Making a Request

- Python requests module has several built-in methods to make HTTP requests to specified URI using GEt, POST, PUT, PATCH, HEAD requests.
- HTTP requests is either to retrieve data from specified URI or to push data to server.
- It works as a request-response protocol between a client and a server.
- GET method is used to retrieve information from the given server using a given URI. GET method sends the encoded user information appended to the page requests.

Example : Python requests making GET request

```
import requests

# Making a GET request
req = requests.get('https://realpython.com/python-requests/')

# check status code for response received
# success code - 200
print(req)

# print content of request
print(req.content)
```

Output-

<Response [200]>
b'\n<!doctype html>\n<html lang="en">\n<head>\n<link href="https://cdn.realpython.com"
rel="preconnect">\n<link href="https://files.realpython.com"
rel="preconnect">\n<title>Python\xe2\x80\x99s Requests Library (Guide) \xe2\x80\x93 Real
Python</title>\n<meta name="author" content="Real Python">\n<meta name="description" content="In this
tutorial on Python's "requests" library, you'll see some of the most useful features that
requests has to offer as well as how to customize and optimize those features. You'll learn how to use
requests efficiently and stop requests to external services from slowing down your application.">\n<meta
name="keywords" content="">\n<meta charset="utf-8">\n<meta name="viewport" content="width=device-
width, initial-scale=1, shrink-to-fit=no, viewport-fit=cover">\n<link rel="stylesheet"
href="https://cdn.realpython.com/static/realpython.min.6f017403b980.css">\n<link rel="stylesheet"
href="https://cdn.realpython.com/static/gfonts/font.5ac42994de49.css">\n<link rel="preload"
href="https://cdn.realpython.com/static/glightbox.min.f69035b3cab2.css" as="style"
onload="this.onload=null;this.rel=\'stylesheet\'"><noscript><link rel="stylesheet"
href="https://cdn.realpython.com/static/glightbox.min.f69035b3cab2.css"></noscript>\n<link
rel="canonical" href="https://realpython.com/python-requests/">\n<meta name="twitter:card"
content="summary_large_image">\n<meta name="twitter:image"
content="https://files.realpython.com/media/Python-Requests-Library-
Tutorial_Watermarked.3c9dfdc7b014.jpg">\n<meta property="og:image"
content="https://files.realpython.com/media/Python-Requests-Library-
Tutorial_Watermarked.3c9dfdc7b014.jpg">\n<meta name="twitter:creator" content="@realpython">\n<meta
name="twitter:site" content="@realpython">\n<meta property="og:title" content="Python\xe2\x80\x99s
Requests Library (Guide) \xe2\x80\x93 Real Python">\n<meta property="og:type" content="article">\n<meta
property="og:url" -cap" aria-hidden="true"> Python Tutorials \xe2\x86\x92
<small class="text-
secondary">In-depth articles and video courses</small>\n<a class="dropdown-item" href="/learning-
paths/" style="color: #ffc873; line-height: 110%;">
Learning Paths.....

5.13.3 Response Object

- Requests to a URI returns a response and response object in terms of python is returned by requests.method(), method being get, post, put etc.
- Response object helps in data normalization and creating ideal portions of code.
- One can check if the request was processed successfully or not by using the response.status_code which returns the status code from the header itself.

Example : Python requests Response Object

```
import requests

# Making a GET request
requests = requests.get('https://realpython.com/python-requests/#getting-started-with-requests/')

# print request object
print(requests.url)
```

```
# print status code
print(requests.status_code)
```

Output:

```
https://www.geeksforgeeks.org/python-programming-language/
200
```

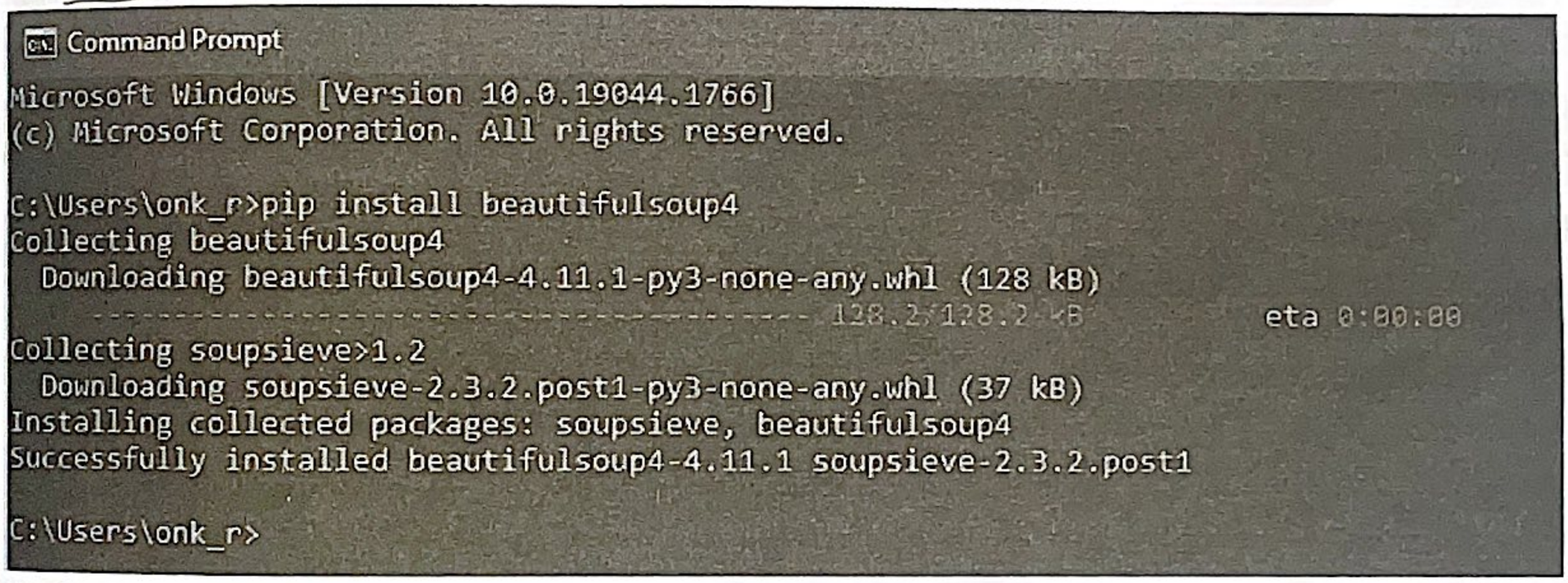
5.14 Beautiful Soup

- Beautiful Soup module is used to extract information from the HTML and XML files.
- It provides a parse tree and the functions to navigate, search or modify this parse tree.
- Beautiful Soup library is built on top of the HTML parsing libraries like html5lib, lxml, html.parser, etc. So Beautiful Soup object and specify the parser library can be created at the same time.

5.14.1 Installation

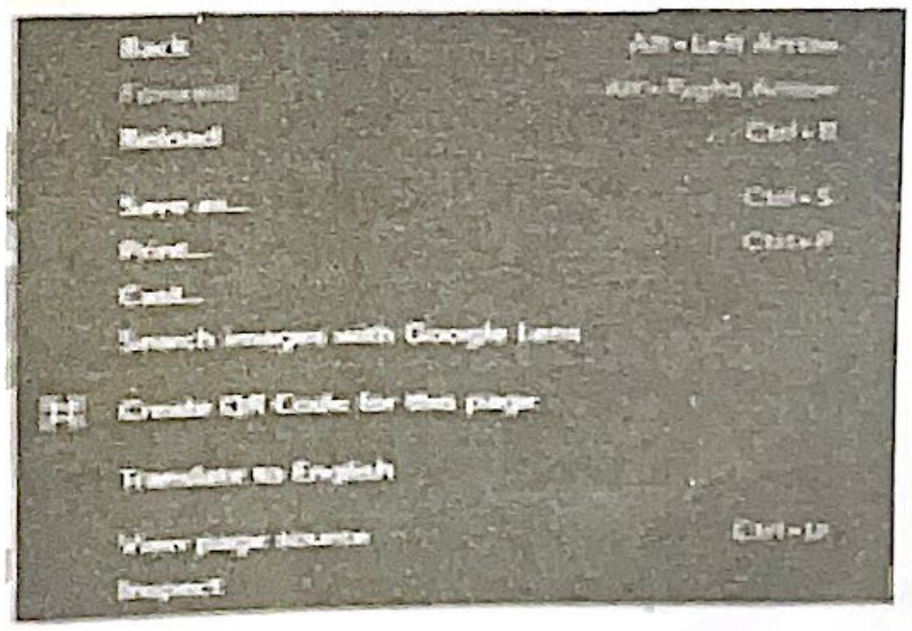
Beautiful Soup is installed by running the following command in the terminal

pip install beautifulsoup4

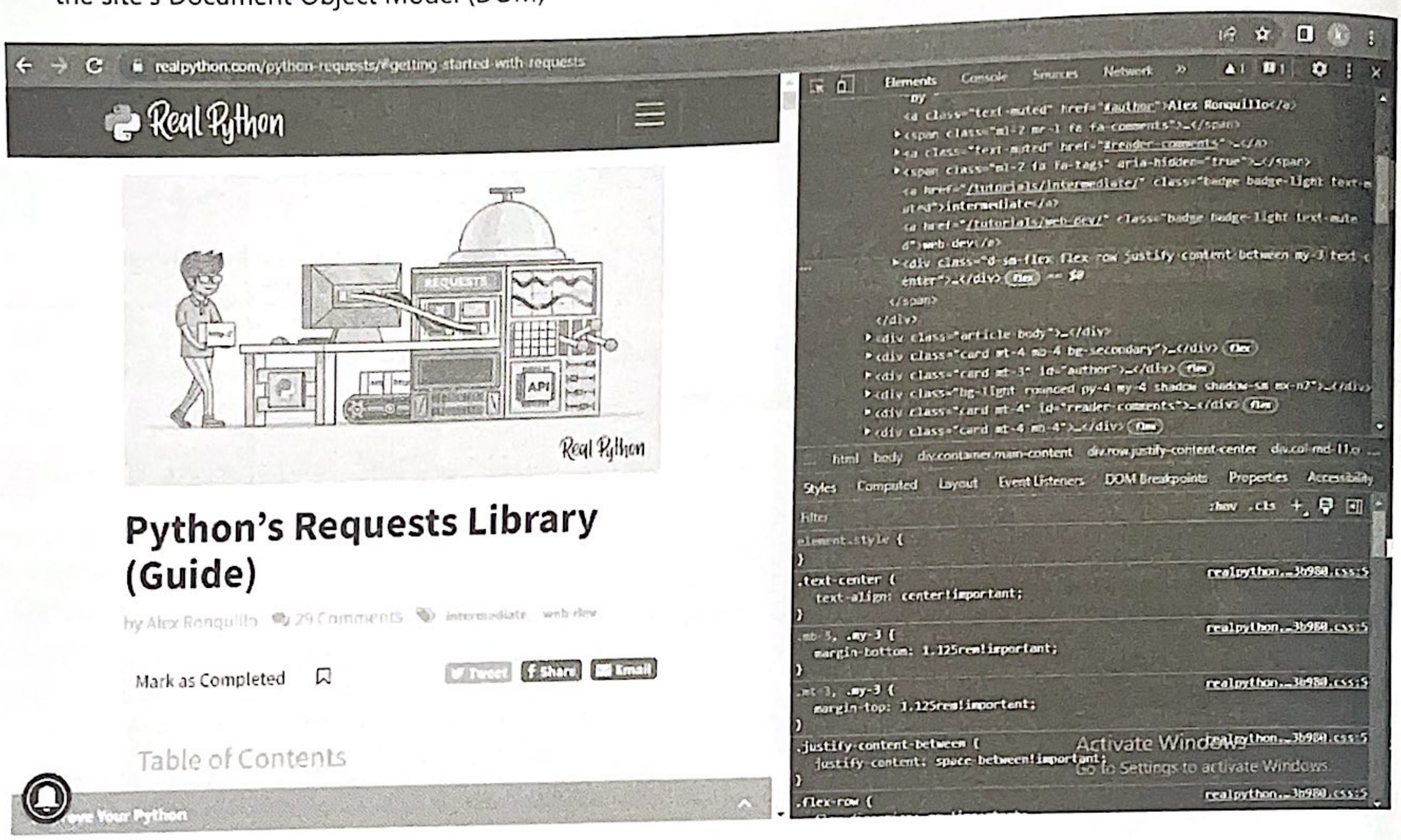


5.14.2 Inspecting Website

- In order to select the desired data from the entire page we must understand the structure of the page
- We can understand the structure by right-clicking on the page that we want to scrape and select the inspect element



- After clicking the inspect button the Developer Tools of the browser gets open. The developer's tools allow seeing the site's Document Object Model (DOM)



5.15 HTML Parsing

- Once we get the HTML of the page we can parse this raw HTML code into some useful information.
- First we will create a BeautifulSoup object by specifying the parser we want to use.

Example : Python BeautifulSoup Parsing HTML

```
import requests
from bs4 import BeautifulSoup


# Making a GET request
req = requests.get('https://realpython.com/python-requests/#getting-started-with-requests')

# check status code for response received
# success code - 200
print(req)


# Parsing the HTML
soup = BeautifulSoup(req.content, 'html.parser')
print(soup.prettify())
```

Output :

```
<Response [200]>
<!DOCTYPE doctype html>
<html lang="en">
<head>
<link href="https://cdn.realpython.com" rel="preconnect"/>
<link href="https://files.realpython.com" rel="preconnect"/>
<title>
Python's Requests Library (Guide) - Real Python
</title>
<meta content="Real Python" name="author"/>
<meta content="In this tutorial on Python's &quot;requests&quot; library, you'll see some of the most useful features that requests has to of
<meta content="" name="keywords"/>
<meta charset="utf-8"/>
<meta content="width=device-width, initial-scale=1, shrink-to-fit=no, viewport-fit=cover" name="viewport"/>
<link href="https://cdn.realpython.com/static/realpython.min.6f817403b080.css" rel="stylesheet"/>
<link href="https://cdn.realpython.com/static/gfonts/font.5ac42994de49.css" rel="stylesheet"/>
<link as="style" href="https://cdn.realpython.com/static/glightbox.min.f69035b3cab2.css" onload="this.onload=null;this.rel='stylesheet'" rel=
<noscript>
<link href="https://cdn.realpython.com/static/glightbox.min.f69035b3cab2.css" rel="stylesheet"/>
</noscript>
<link href="https://realpython.com/python-requests/" rel="canonical"/>
<meta content="summary_large_image" name="twitter:card"/>
<meta content="https://files.realpython.com/media/Python-Requests-Library-Tutorial_Watermarked.3c9dfdc7b014.jpg" name="twitter:image"/>
<meta content="https://files.realpython.com/media/Python-Requests-Library-Tutorial_Watermarked.3c9dfdc7b014.jpg" property="og:image"/>
```

Example : To extract the title of the page

```
import requests
from bs4 import BeautifulSoup

# Making a GET request
r = requests.get('https://realpython.com/python-requests/#getting-started-with-requests')

# Parsing the HTML
soup = BeautifulSoup(r.content, 'html.parser')

# Getting the title tag
print(soup.title)

# Getting the name of the tag
print(soup.title.name)

# Getting the name of parent tag
print(soup.title.parent.name)

# use the child attribute to get
# the name of the child tag
```

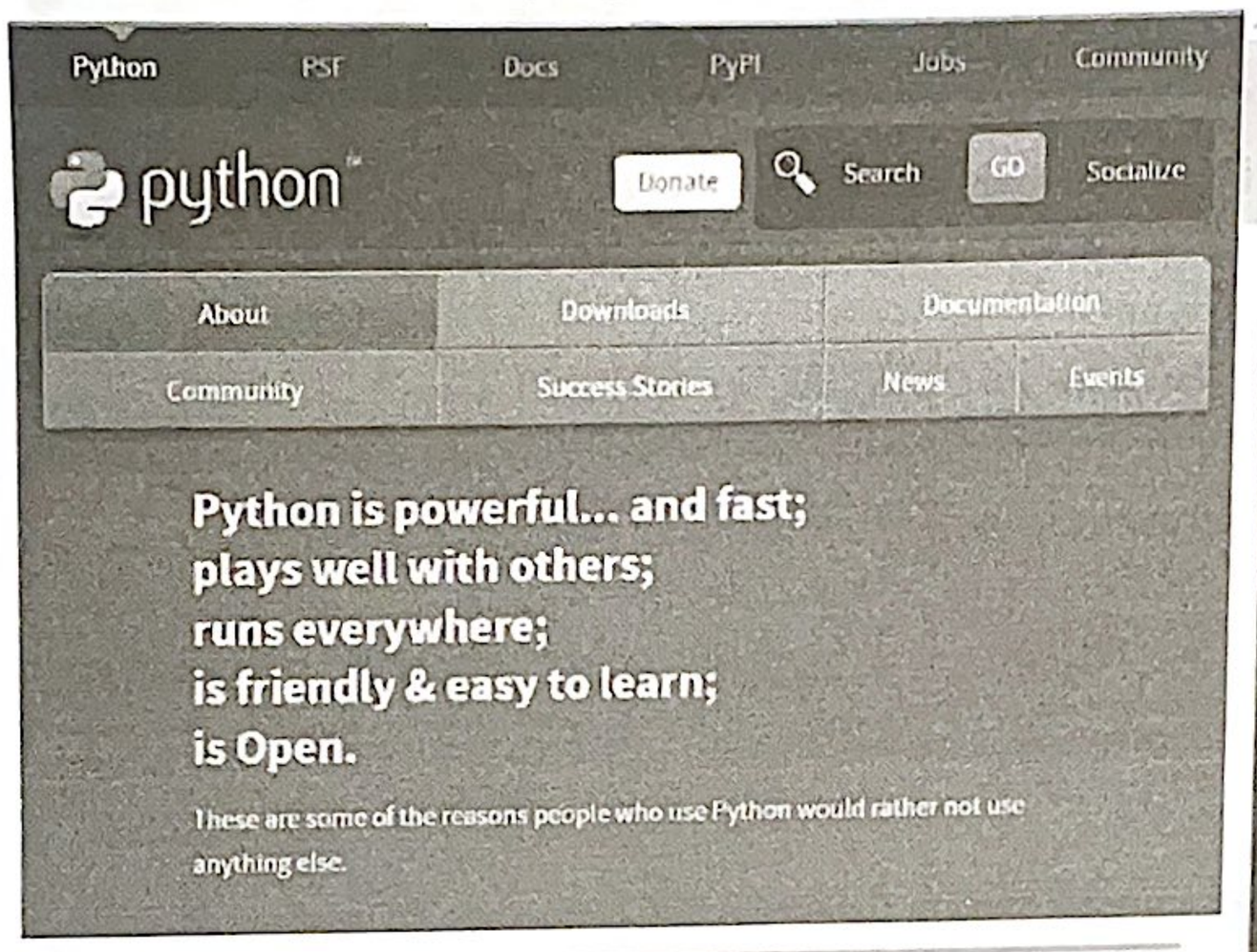
Output :

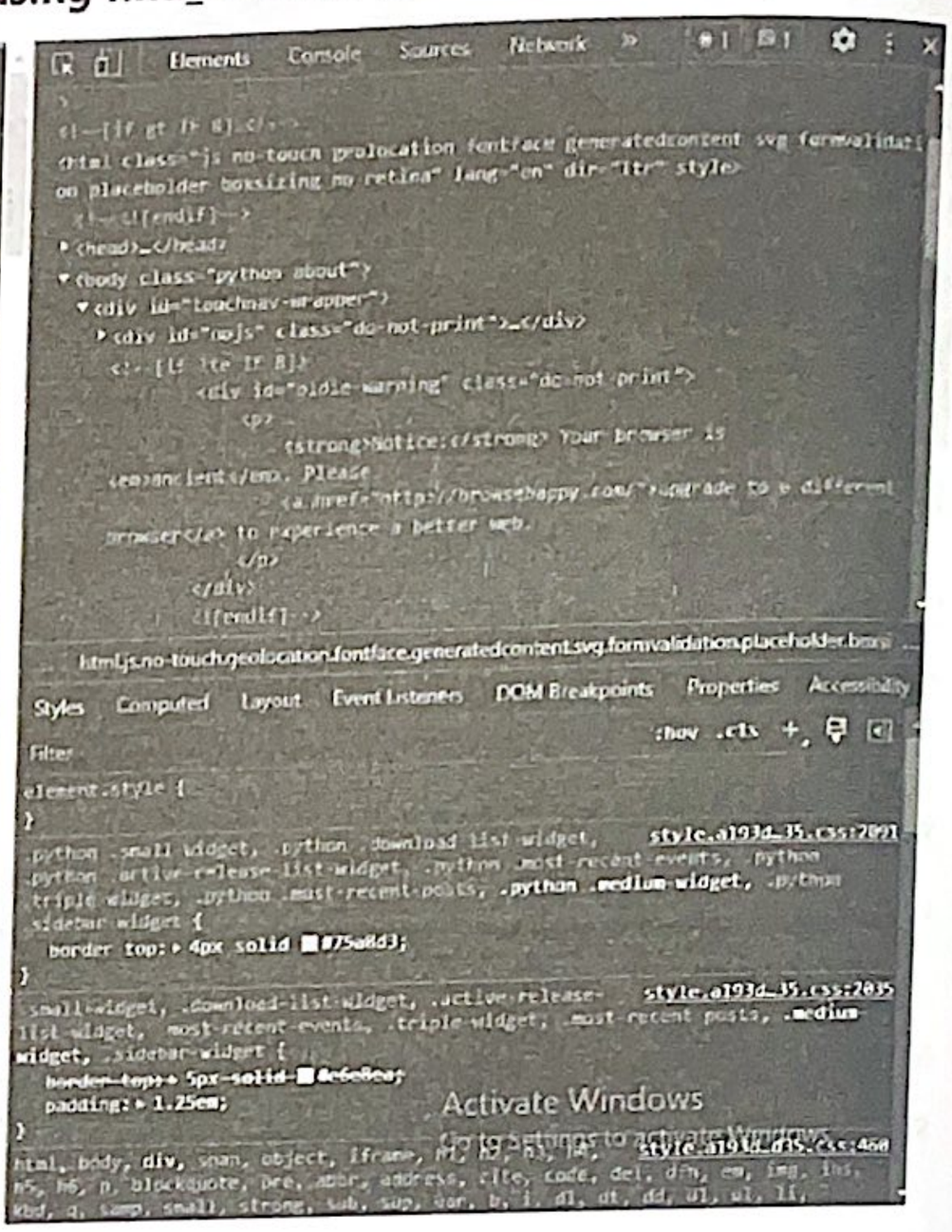
```
<title>Python's Requests Library (Guide) – Real Python</title>
title
head
```

5.15.1 Finding Elements by Class

- The soup object contains all the data in the nested structure which could be programmatically extracted.
- The website contains a lot of text so now let's scrape all those content.
- All the content of the page is under the div with class entry-content.
- The find class will find the given tag with the given attribute.

Example : To find all the p tags present the following class using find_all class of the BeautifulSoup.






```
import requests
from bs4 import BeautifulSoup

# Making a GET request
req = requests.get('https://www.python.org/about/')

# Parsing the HTML
soup = BeautifulSoup(req.content, 'html.parser')

s = soup.find('div', class_='do-not-print')
content = s.find_all('p')

print(content)
```

Output :

[Notice: While JavaScript is not essential for this website, your interaction with the content will be limited. Please turn JavaScript on for the full experience.]

Example : Extracting text from the tags

```
import requests
from bs4 import BeautifulSoup

# Making a GET request
req = requests.get('https://www.python.org/about/')

# Parsing the HTML
soup = BeautifulSoup(req.content, 'html.parser')

s = soup.find('div', class_='do-not-print')
content = s.find_all('p')

for line in content:
  print(line.text)
```

Output :

Notice : While JavaScript is not essential for this website, your interaction with the content will be limited. Please turn JavaScript on for the full experience.

Reviewed Questions

Q. 1 Explain distributed architecture of a search engine.

Q. 2 Explain distributed architecture of a search engine.

Q. 3 Explain distributed architecture of a search engine.

Q. 4 Discuss challenges involved in web searching.

Q. 5 Explain crawler indexer architecture in details.

Q. 6 Write a note on characterizing the web.

Q. 7 What are meta crawlers ? Explain with suitable example.

Q. 8 Write note on Beautiful Soup

Q. 9 Explain HTML parsing

❑❑❑