Web Scraping:

- Technique to fetch data from websites
- automation of data extraction process from websites.
- Construction of agent to download / parse / organize data From the web in an automated manner.
- technique to collect content and data from the internet.
- done with the help of web scraping softwares (web scrapers) auto-atically load and extract based on user requirements.
- saved in local file analyzed as needed.

We can use : Java / python / Ruby / Node

Some software companies have designed ≠ tools that enabdex other people to use scraping techniques by means of attractive and powerful user interface.

Video / image / text / user-er sentiments.

Some definitions:

→ HTTP: hypertext transfer Protocol
Machine interchange information
Transported over the internet enable data exchange (via www.)
-declarative Document type definition (HTML, XML, JSON..)

→ HTML: hyperlext Markup Language (STandarol)

Parsing

analyzing strings and symbols to reveal the data You need only

Crawling

Moving across/Through a website to gather data From more than 1 URL.

→ JSON: JavaScript Open Notation
Readable Text used to transmit data objects consisting of attribute-value pairs.

→ API: Application programming interface
Set of rules / protocols to build a software application. In the context web.
Used to gather clean data from a website (data that is not wrapped in HTML, Javascript, bound in HTTP....)

Web Scraping Techniques:

Manual Extraction Techniques

Manually copy pasting

Automated Extraction Techniques:

Automatically based on user requirement.

Parsing:

make something understandable by analyzing it part by part
convert information to easier form.

HTML Parsing: Extracting information based on user requirements
using JavaScript

DOM Parsing: Document object Model
defines an interface that enables a user to modify / update
style / structure / content of XML document

Challenges of Web Scraping:

Data Warehousing:

the data warehousing infrastructure should be properly built
before storing / Exporting data.

Website Structure changes:

Every user interface is updated to improve experience.
⟹ requires web scrapers to modify their code.

Anti-Scraping Technologies:

Some websites use anti-scraping technologies.
they use IP blocking mechanism.

Quality of data extracted:

Records that do not meet the quality of information will affect
the integrity of data.

Scraping libraries in Python:

Beautiful Soup: parse data from XML and HTML.
makes navigating and searching through large swathes much easier.

Scrapy: Python-based application framework that crawls and extracts structured data from the web.
It can be used as a general-purpose web crawler, extract data through API.

Pandas: Used for data manipulation / indexing
" to scrape the web in conjunction with BeautifulSoup.
if we use Pandas we can continue to analyse using 1 language.

Parsehub: (not in Python)

Web Scrapers

Step 1: Making HTTP request to a server (Knocking the door)
Step 2: Extracting and parsing the website's code (The bot can read/Extract HTML/XML code)
Step 3: saving relevant data (stored in Excel file such as .csv .xls format, mysql database)

Steps of Scraping:

1) Find URL
2) Inspect the Page
3) Identify the data you want to extract
4) write the necessary code
5) Execute the code (python web_scrap.py)
6) Store the data (.csv)

Python IDLE

Python installation comes with an Integrated Development and Learning Environment. (Edit .py files with ease)

Best place shell to experiment python : Shell (a basic Read-Eval-Print loop)

```
from urllib.request import urlopen
    url = "http://olympus.realpython.org/profiles/aphrodite"
    page = urlopen(url)

    html_bytes = page.read
html_bytes = html_bytes.decode("utf-8")
print(html)          → (see the content of the page)
```

Extract text from HTML 8

```
    title_index = html.find("<title>")
title_index                          ↖ index of the <title>

    start_index = title_index + len("<title>")
start_index                          ↖ index of the title itself.

    end_index = html.find("</title>")
end_index
title = html[start_index:end_index]   } extract the title by slicing HTML string
title
```

Regular Expressions

```
import re
```

Use special characters (metacharacters)

```
re.findall("ab*c", "ac")
```

↓

we can replace it with "Beautiful Soup"

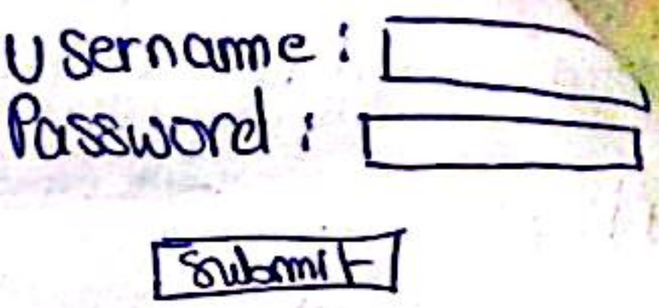
```
from bs4 import BeautifulSoup
from urllib.request import urlopen
                ↖ requesting page content
url = "http://olympus.realpython.org/profiles/dionysus"
page = urlopen(url)
html = page.read().decode("utf-8")
soup = BeautifulSoup(html, "html.parser")
Print (Soup.get_text())
Soup.find_all("img") ← it gives me image Url
image1, image2 = soup.find_all("img")
image1.name
image1["src"]
Soup.title   (when using BeautifulSoup it corrects mistakes)
                                    <TITLE> → <title>
soup.title.string ← string between title tags.
soup.find_all("img", src = "/static/dionysus.jpg")
```

Interact with HTML forms:

MechanicalSoup : web browser with no graphical user interface

```
import mechanicalsoup
browser = mechanicalsoup.browser
url = "http://oly-pus.realpython.org/login"
page = browser.get(url)
page ← <Response [200]>
type(page.soup) → mechanicalSoup uses BeautifulSoup to parse HTML.
page.soup ← view the HTML
```

import mechanicalsoup



#1
```
browser = mechanicalsoup.Browser()
Url = "http://olympus.real.org/login"
Login_page = browser.get(Url)
Login_html = login_page.soup
```

#2
```
form = login_html.select("form")[0]
form.select('input")[0]["value"] = "Zeus"
  //      //        [1]    //     = "Thunder"
```

#3
```
profiles_page = browser.submit(form, login_page.url)
```

profiles_page.url → to confirm that the submission successfully redirected to the profiles_page.

login_html.select("form") returns a list of all <form> elements on the page

browser.submit() → to submit the form.

Obtain the URL for each link on the profiles page.

```
links = profiles_page.soup.select("a")
for link in links:
    address = link["href"]
    text = link.text
    print(F"{text}: {address}")
```

} iterate through links.

storing Scraped data to database

storing data to CSV file:

CSV: comma-seperated values
Supported by microsoft Excel

to write to a CSV file in python:

1- Open csv file in the write mode:

```
import CSV
csvfile = open ('test.csv', 'w+')
```

read and write
'r': read
'w': write

to support writing non-ASCII values to a CSV file:
specify the character encoding:

```
csvfile = open ('test.csv', 'w', encoding = 'UTF8')
```

2- Create a CSV writer object:

```
Writer = csv.writer (csv file)
```

pass the opened file as its arguments.

3- Write data to the CSV file:

```
writer.writerow (('number', 'number plus 2', 'number times 2'))
for i in range (10):
    writer.writerow ((i, i+2, i*2))
```

4- Close the CSVfile

```
csvfile.close()
```