How to create Web Scraping/Crawling with Python

Web scraping deals with extracting or scraping the information from the website. Web scraping is also sometimes referred to as web harvesting or web data extraction. Copying text from a website and pasting it to your local system is also web scraping. However, it is a manual task. Generally, web scraping deals with extracting data automatically with the help of web crawlers. Web crawlers are scripts that connect to the world wide web using the HTTP protocol and allow you to fetch data in an automated manner.

Whether you are a data scientist, engineer, or anybody who analyzes vast amounts of datasets, the ability to scrape data from the web is a useful skill to have. Let's say you find data from the web, and there is no direct way to download it, web scraping using Python is a skill you can use to extract the data into a useful form that can then be imported and used in various ways.

Some of the practical applications of web scraping could be:

- Gathering resume of candidates with a specific skill,
- Extracting tweets from twitter with specific hashtags,
- Lead generation in marketing,
- Scraping product details and reviews from e-commerce websites.

Apart from the above use-cases, web scraping is widely used in natural language processing for extracting text from the websites for training a deep learning model.

While web scraping refers to the actual gathering of web-based data, web crawling refers to the navigation of a program between webpages. Web crawling allows a program to gather related data from multiple web pages and websites

The process of web scraping include:

1. Visual inspection: Figure out what to extract(you need to Inspect the Site Using Developer Tools(web console))
2. Make an HTTP request to the webpage
3. Parse the HTTP response
4. Persist/Utilize the relevant data

How to do that with python?

First, you'll want to get the site's HTML code into your Python script so that you can interact with it. For this task, you'll use Python's requests library. Type the following in your terminal to install it:

$ pip3 install requests

Then open up a new file in your favorite text editor. All you need to retrieve the HTML are a few lines of code:

```
import requests


URL = 'https://www.monster.com/jobs/search/?q=Software-Developer&where=Australia'

page = requests.get(URL)
```

This code performs an HTTP request to the given URL. It retrieves the HTML data that the server sends back and stores that data in a Python object.

Your next step depends of what you need to scraping -Static(server that hosts the site sends back HTML documents that already contain all the data you'll get to see as a user) or Dynamic Site(the HTML change depends of user actions)

For Static website we can use Beautiful Soap

$pip install beautifulsoup4

So the python script should be

```
from bs4 import BeautifulSoup as bs

import urllib

import pandas as pd


    # Using requests module for downloading webpage content

    response = requests.get(url)

    # Parsing html data using BeautifulSoup

    soup = bs(response.content, 'html.parser')

#this will print the Html as HTML in the console

print(soup.prettify())


 body = soup.find('body')
```

To search in the html we will use beautiful soap functions:

//return first found

-search by id:

answer= body.find('div', {"id":"comments-link-46397882"})

-search by classname

answer= body.find('div', {"class":"answer"})

-search by html tag only

answer= body.find('div')

//return array of all possibilities

-same but used find_all instead

//CSS selector -replace nth-child with nth-of-type

answer=body.select('#answer-31675177 > div:nth-of-type(1) > div:nth-of-type(2) > div:nth-of-type(1)')

//Xpath -Beautiful soap can`t handle xpath

For Dynamic website we can use Selenium Web Driver

(Asynchronously Loaded Content and User Interaction)

elenium allows a program to open a web browser and interact with it in the same way that a human user would, including clicking and typing. It also hasBeautifulSoup-esque tools for searching the HTML source of the current page.

Selenium requires an executable driver file for each kind of browser. The following examples use Google Chrome, but Selenium supports Firefox, Internet Explorer, Safari, Opera, andPhantomJS (a special browser without a user interface). To use Selenium, start up a browser using one of the drivers in `selenium.webdriver`. The browser has a `get()` method for going to different web pages, a `page_source`attribute containing the HTML source of the current page, and a `close()` method to exit the browser.

```
from selenium import webdriver

# Start up a browser and go to example.com.

browser = webdriver.Chrome()

browser.get("url")
```

```
# Feed the HTML source code for the page into BeautifulSoup for
processing.

soup = BeautifulSoup(browser.page_source, "html.parser")

browser.close()
```

. Selenium can deliver the HTML page source to BeautifulSoup, but it also has its own tools for finding tags in the HTML.

Method	Returns
`find_element_by_tag_name()`	The first tag with the given name
`find_element_by_name()`	The first tag with the specified `name` attribute
`find_element_by_class_name()`	The first tag with the given `class` attribute
`find_element_by_id()`	The first tag with the given `id` attribute
`find_element_by_link_text()`	The first tag with a matching `href` attribute
`find_element_by_partial_link_text()`	The first tag with a partially matching `href` attribute

Table 6.1: Methods of the `selenium.webdriver.Chrome` class.

Each of the `find_element_by_*()` methods returns a single object representing a web element(of type `selenium.webdriver.remote.webelement.WebElement`) If no such element can be found, a Selenium `NoSuchElementException` is raised. Each webdriver also has several `find_elements_by_*()` methods (elements, plural) that return a list of all matching elements, or an empty list if there are no matches.Web element objects have methods that allow the program to interact with them like `click()` -sends a click,`send_keys()` -enters in text, and `clear()which` deletes existing text(more you can see https://www.selenium.dev/selenium/docs/api/py/webdriver_remote/selenium.webdriver.remote.webelement.html).

Useful Resources:

1.https://zenscrape.com/6-best-web-scraping-tips-for-advanced-python-web-scraping/

2.http://www.acme.byu.edu/wp-content/uploads/2017/08/WebScraping2.pdf