

WebScraping

Instalación de Python	2
Instalación de Git	2
Descarga del WebDrive, Chromedrive	2
Área de trabajo	4
Los requerimientos del archivo	9
Inicio del proyecto	9
Código Base	10
Análisis de la página web	11
Manera de pensar	12
Realización del Web Scraping	13
Tiempo de espera	18
Guardado de datos	19
Guardar proyecto en Github	20
Clonar un repositorio	24
Proyectos futuros	25
Referencias	26

Pasos para realizar webscraping de manera rápida con Python y Selenium, en este caso utilizaremos Google Chrome para el scrapeo.

Instalación de Python

[Download Python | Python.org](#) descargar desde la versión 3.9 - 3.11 son seguras para su uso.

Comando en “cmd” comando python --version para ver la versión de Python instalada.

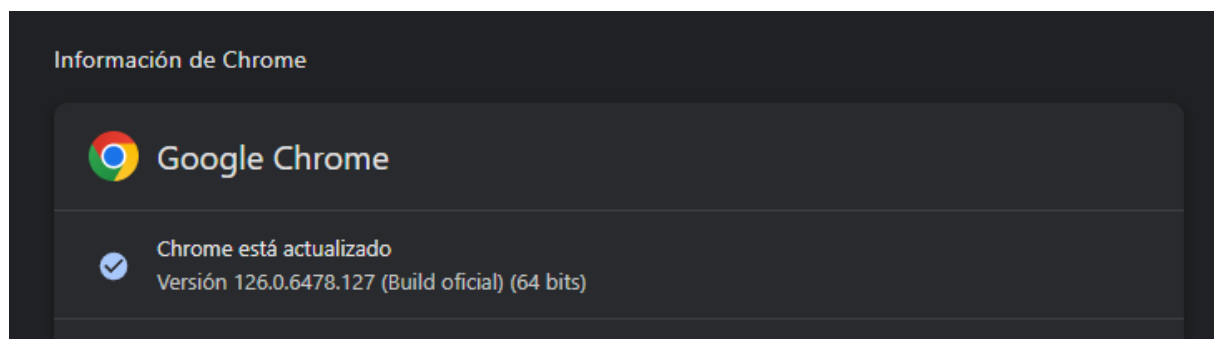
```
C:\Users\kevin>python --version
Python 3.11.5
```

Instalación de Git

Instalaremos Git para el control de versiones de este proyecto. [Git - Downloads \(git-scm.com\)](#) instalarlo dependiendo de la versión de sistema operativo que tengas en tu computadora.

Descarga del WebDrive, Chromedrive

1. Vamos a la página de [Chrome for Testing availability \(googlechromelabs.github.io\)](#) para buscar el web drive dependiendo de la versión de Google Chrome que tengamos. Para ello tenemos que ir a configuración de Google Chrome, luego a información de Chrome, ahí encontraras la versión de Chrome que tienes actualizada.



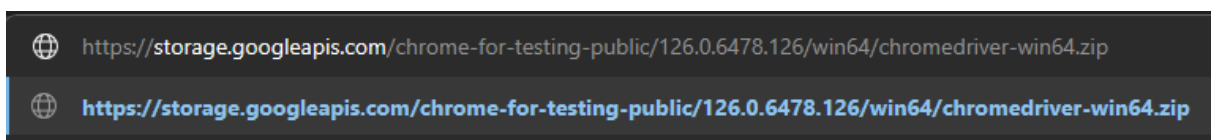
- En mi caso tengo la Versión 126.0.6478.127 (Build oficial) (64 bits), en la página descargamos el ChromeDriver con la versión más cercana a la nuestra.

Stable

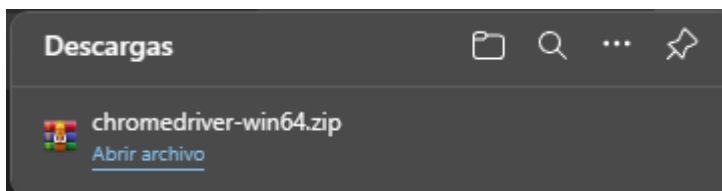
Version: 126.0.6478.126 (r1300313)

Binary	Platform	URL
chrome	linux64	https://storage.googleapis.com/chrome-for-testing-public/126.0.6478.126/linux64/chrome-linux64.zip
chrome	mac-arm64	https://storage.googleapis.com/chrome-for-testing-public/126.0.6478.126/mac-arm64/chrome-mac-arm64.zip
chrome	mac-x64	https://storage.googleapis.com/chrome-for-testing-public/126.0.6478.126/mac-x64/chrome-mac-x64.zip
chrome	win32	https://storage.googleapis.com/chrome-for-testing-public/126.0.6478.126/win32/chrome-win32.zip
chrome	win64	https://storage.googleapis.com/chrome-for-testing-public/126.0.6478.126/win64/chrome-win64.zip
chromedriver	linux64	https://storage.googleapis.com/chrome-for-testing-public/126.0.6478.126/linux64/chromedriver-linux64.zip
chromedriver	mac-arm64	https://storage.googleapis.com/chrome-for-testing-public/126.0.6478.126/mac-arm64/chromedriver-mac-arm64.zip
chromedriver	mac-x64	https://storage.googleapis.com/chrome-for-testing-public/126.0.6478.126/mac-x64/chromedriver-mac-x64.zip
chromedriver	win32	https://storage.googleapis.com/chrome-for-testing-public/126.0.6478.126/win32/chromedriver-win32.zip
chromedriver	win64	https://storage.googleapis.com/chrome-for-testing-public/126.0.6478.126/win64/chromedriver-win64.zip
chrome-headless-shell	linux64	https://storage.googleapis.com/chrome-for-testing-public/126.0.6478.126/linux64/chrome-headless-shell-linux64.zip

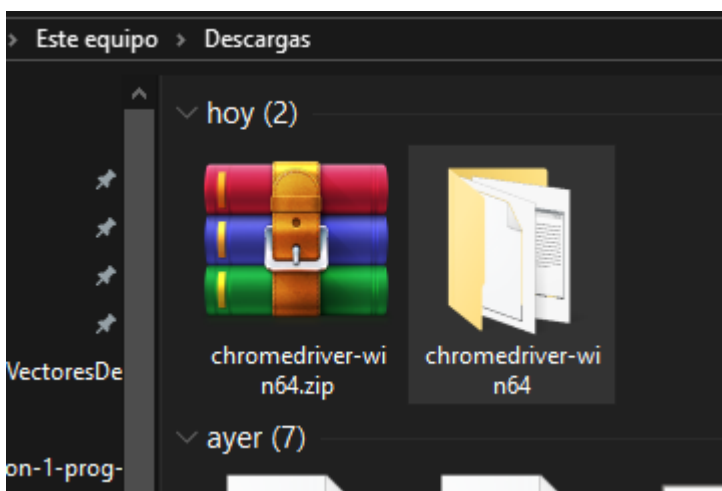
- Para descargar solo tenemos que copiar el link y pegarlo en el buscador de Google.



- Damos enter y comenzará a descargar el archivo .zip con el web driver



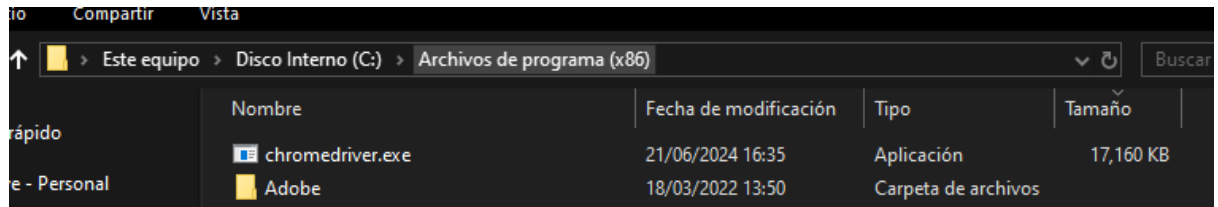
- Extraemos la carpeta



- Entramos y veremos el “chromedriver.exe”

Nombre	Fecha de modificación	Tipo	Tamaño
chromedriver-win64	17/05/2024 22:36	Carpeta de archivos	
chromedriver.exe	21/06/2024 16:35	Aplicación	17,160 KB
LICENSE.chromedriver	21/06/2024 16:35	Archivo CHROME...	405 KB

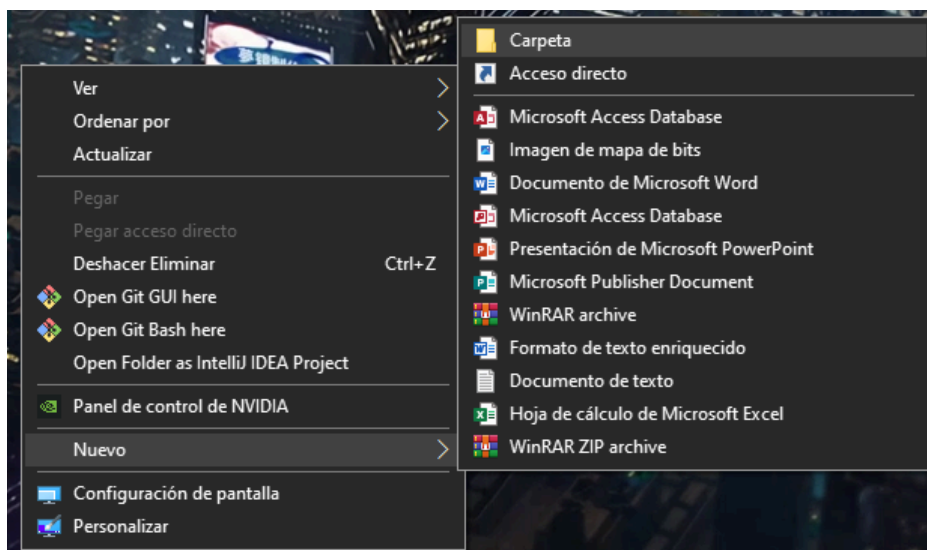
- Copiaremos ese archivo y lo llevaremos al Disco interno C, en la carpeta “Archivos de programa (x86)” y ahí lo pegaremos.



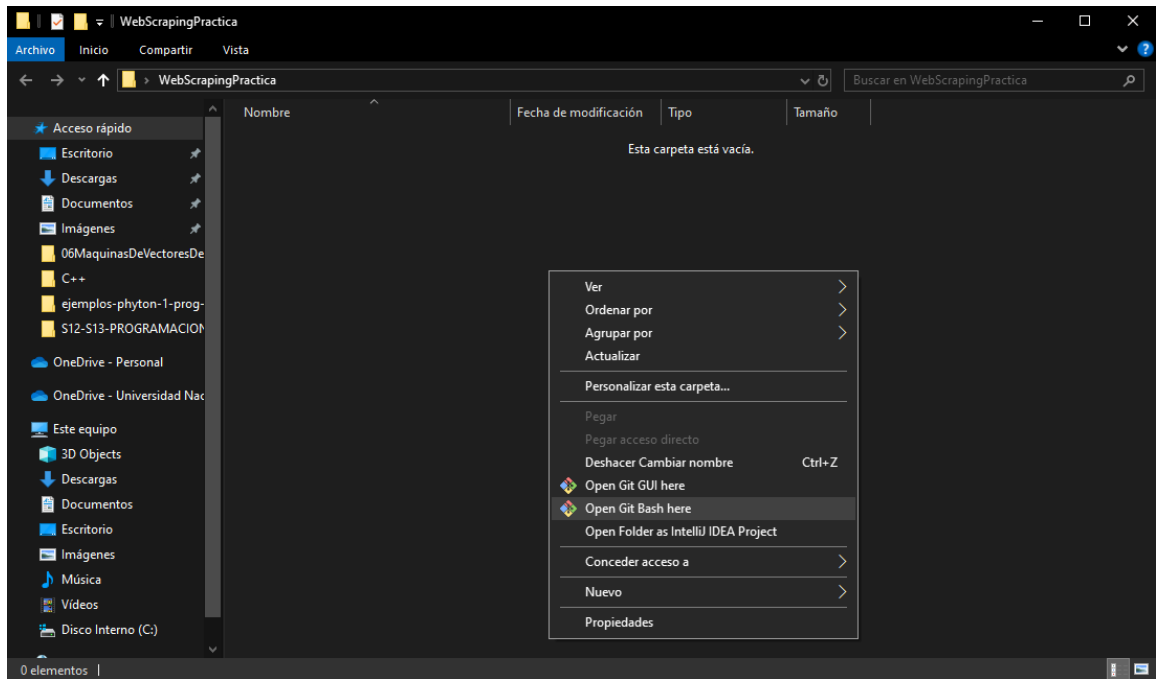
- Recordemos que la ruta donde está ese archivo es “C:\Program Files (x86)\chromedriver.exe”, ya que luego será importante esta ruta para nuestro código.

Área de trabajo

- Creación de Carpeta, le pondremos el nombre que gustemos. En este caso le puse el nombre “WebScrapingPractica”.



- Ingresamos a Visual Studio Code mediante Git Bash, click derecho dentro de la carpeta y hacemos click en “Open Git Bash here”.



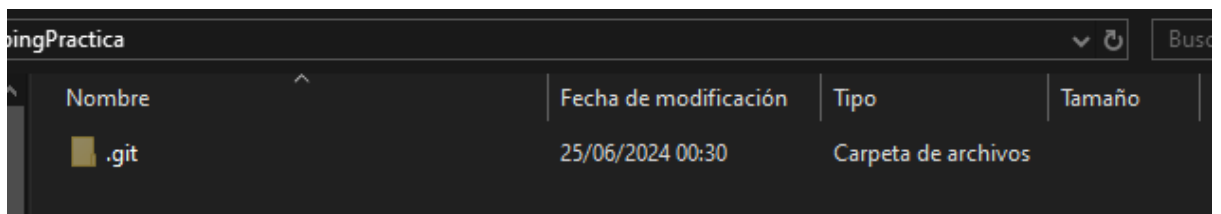
3. Iniciamos el control de versiones con el siguiente comando en la terminal “git init”.

```

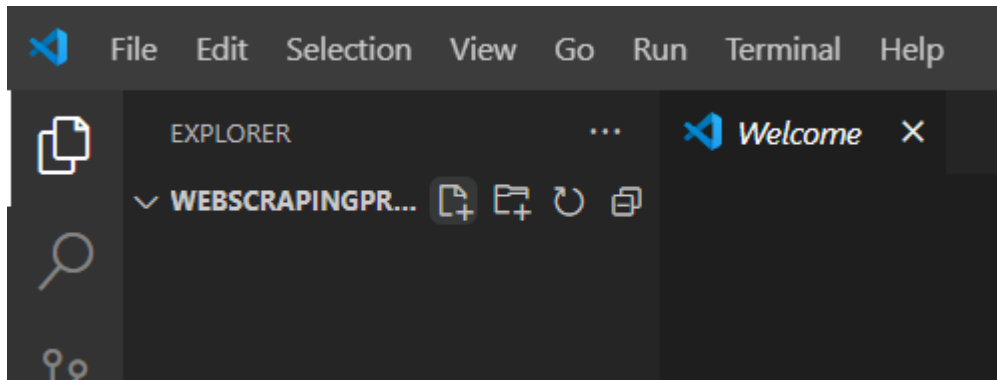
MINGW64/c:/Users/kevin/Desktop/WebScrapingPractica
Kevin Jonathan@DESKTOP-SA54AJI MINGW64 ~/Desktop/WebScrapingPractica
$ git init
Initialized empty Git repository in C:/Users/Beatriz/Desktop/WebScrapingPractica/.git/
Kevin Jonathan@DESKTOP-SA54AJI MINGW64 ~/Desktop/WebScrapingPractica (main)

```

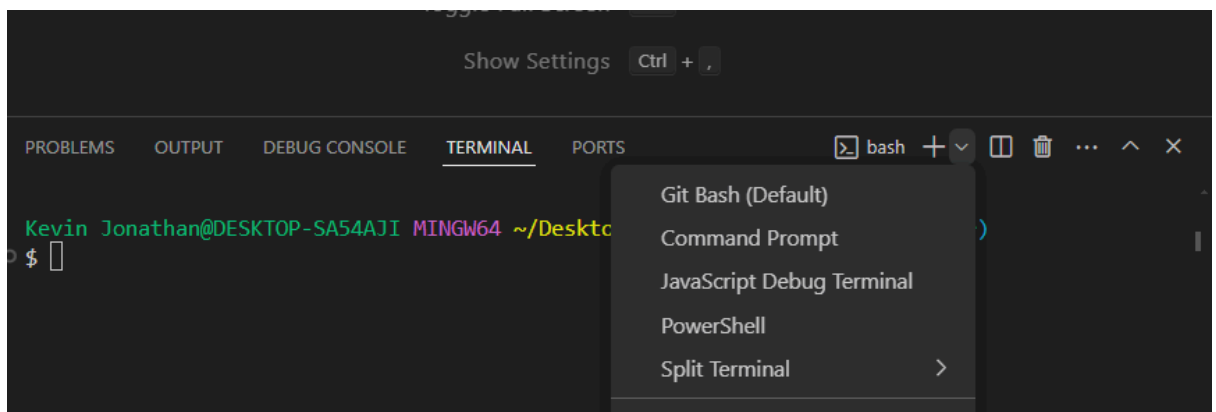
4. Veremos que hay una carpeta “.git” dentro de nuestra carpeta “WebScrapingPractica” el cual será nuestra carpeta de controlador de git.



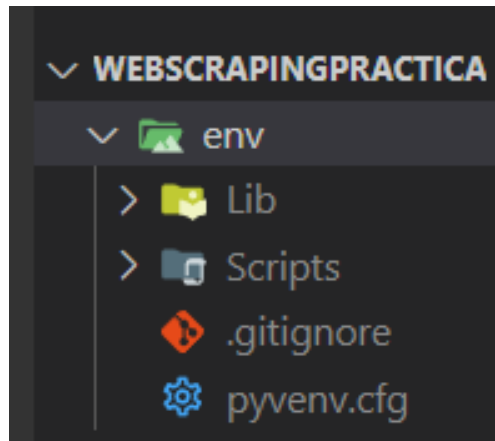
5. En la misma terminal escribimos el comando “code .” el cual abrirá la carpeta en Visual Studio Code.



6. Pulsamos `ctrl + ñ` para abrir la terminal, utilizaremos la terminal de Git Bash, pero ya en Visual Studio Code. Para habilitarlo solo tendremos que hacer click en la flecha hacia abajo y desplegará diferentes opciones de terminal, en este caso utilizaremos Git Bash.



7. Iniciamos el entorno virtual para que nada nos interfiera en la ejecución o que exista cruces de versiones. Utilizaremos la herramienta de “virtualenv”, para ello si no lo tenemos instalado ponemos el comando “`pip install virtualenv`” en la terminal.
 - a. Para ejecutarlo ponemos “`virtualenv -p python3 env`” lo cual nos creara una carpeta de nombre env, env es el nombre que le asignamos por defecto, pero puede utilizar otros nombres. Luego de ejecutar el comando, se nos creara en nuestra área de trabajo una carpeta llamada env, el cual contendrá:
 - i. Carpeta Lib: Lugar donde se almacenaran todas las librerías que instalemos.
 - ii. Carpeta Scripts: Lugar donde estarán los comandos para activar, desactivar y otras opciones del entorno virtual.
 - iii. .gitignore: Son las excepciones, ahí dentro se pone lo que queremos ignorar para no subirlo a nuestro Git.
 - iv. pyvenv.cfg: Configuraciones del entorno.



- b. También observaremos en la terminal la confirmación de la creación del entorno virtual con éxito.

```
Kevin Jonathan@DESKTOP-SA54AJI MINGW64 ~/Desktop/WebScrapingPractica (main)
● $ virtualenv -p python3 env
created virtual environment CPython3.11.5.final.0-64 in 4396ms
  creator CPython3Windows(dest=C:\Users\Beatriz\Desktop\WebScrapingPractica\env,
  clear=False, no_vcs_ignore=False, global=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle
, via=copy, app_data_dir=C:\Users\Beatriz\AppData\Local\pypa\virtualenv)
  added seed packages: pip==24.0, setuptools==70.0.0, wheel==0.43.0
  activators BashActivator,BatchActivator,FishActivator,NushellActivator,PowerSh
ellActivator,PythonActivator
```

- c. Por último, tendremos que activar el entorno virtual una vez habiéndolo creado, mediante el comando “source env/Scripts/activate”

```
Kevin Jonathan@DESKTOP-SA54AJI MINGW64 ~
● $ source env/Scripts/activate
(env)
```

- d. Para desactivarlo sería solo poner el comando “deactivate”, pero esto sería cuando ya no quisiéramos utilizar el entorno, mayormente cuando terminemos la realización del trabajo. Por el momento nos quedaremos en el punto C.

```
Kevin Jonathan@DESKTOP-SA54AJI MINGW64 ~
● $ deactivate
```

Instalación de Selenium:

En la terminal ponemos “pip install selenium”, el cual nos instalara en nuestro entorno virtual la biblioteca Selenium.

```
Kevin Jonathan@DESKTOP-SA54AJI MINGW64 ~/Desktop/WebScrapingPractica (main)
● $ pip install selenium
Collecting selenium
  Using cached selenium-4.22.0-py3-none-any.whl.metadata (7.0 kB)
Collecting urllib3<3,>=1.26 (from urllib3[socks]<3,>=1.26->selenium)
  Using cached urllib3-2.2.2-py3-none-any.whl.metadata (6.4 kB)
Collecting trio~0.17 (from selenium)
  Using cached trio-0.25.1-py3-none-any.whl.metadata (8.7 kB)
Collecting trio-websocket~0.9 (from selenium)
  Using cached trio_websocket-0.11.1-py3-none-any.whl.metadata (4.7 kB)
Collecting certifi>=2021.10.8 (from selenium)
```

Agregado: Podemos poner “pip list” para ver los paquetes instalados en nuestro entorno virtual.

```
Kevin Jonathan@DESKTOP-SA54AJI
● $ pip list
Package                Version
-----
attrs                  23.2.0
certifi                 2024.6.2
cffi                   1.16.0
h11                     0.14.0
idna                    3.7
outcome                1.3.0.post0
pip                    24.0
pycparser               2.22
PySocks                 1.7.1
selenium                4.22.0
setuptools              70.0.0
sniffio                 1.3.1
sortedcontainers        2.4.0
trio                    0.25.1
trio-websocket          0.11.1
typing_extensions       4.12.2
urllib3                 2.2.2
websocket-client        1.8.0
wheel                   0.43.0
wsproto                 1.2.0
```

En este caso vemos Selenium, pero selenium no vino solo, sino que se instaló con otros paquetes que necesita para su funcionamiento.

Los requerimientos del archivo

Con el comando “pip freeze > requirements.txt” crearemos un archivo txt donde guardaremos los paquetes que requiere nuestro proyecto.

```
Kevin Jonathan@DESKTOP-SA54AJI MINGW64 ~
● $ pip freeze > requirements.txt
(env)
```

Todo esto debido a que si en un futuro un nuevo usuario quiera ejecutar nuestro proyecto, al tener su entorno virtual activado, lo único que necesitara es ejecutar de mediante comando “pip install -r requirements.txt” y se le instalará todos los paquetes necesarios para ejecutar nuestro proyecto.

```
Kevin Jonathan@DESKTOP-SA54AJI MINGW64 ~/Desktop/WebScrapingPractica (main)
● $ pip install -r requirements.txt
Requirement already satisfied: attrs==23.2.0 in c:\users\beatriz\desktop\web
Requirement already satisfied: certifi==2024.6.2 in c:\users\beatriz\desktop\
4.6.2)
Requirement already satisfied: cffi==1.16.0 in c:\users\beatriz\desktop\websc
Requirement already satisfied: h11==0.14.0 in c:\users\beatriz\desktop\websc
Requirement already satisfied: idna==3.7 in c:\users\beatriz\desktop\webscrap
Requirement already satisfied: outcome==1.3.0.post0 in c:\users\beatriz\desk
1.3.0.post0)
Requirement already satisfied: pycparser==2.22 in c:\users\beatriz\desktop\w
Requirement already satisfied: PySocks==1.7.1 in c:\users\beatriz\desktop\wel
Requirement already satisfied: selenium==4.22.0 in c:\users\beatriz\desktop\w
.0)
Requirement already satisfied: sniffio==1.3.1 in c:\users\beatriz\desktop\wel
)
```

Apreciamos la instalación de los paquetes con su respectiva versión, este ejemplo lo hice en otro archivo para evidencia que funciona el comando “pip install -r requirements.txt”.

Inicio del proyecto

Creamos la carpeta src para tener nuestros archivos y demás ejecutables, para ello escribimos en la terminal “mkdir src”, entramos a la carpeta con el comando “cd src”, luego creamos el archivo py llamado scraping.py mediante el comando en terminal “touch scraping.py”.

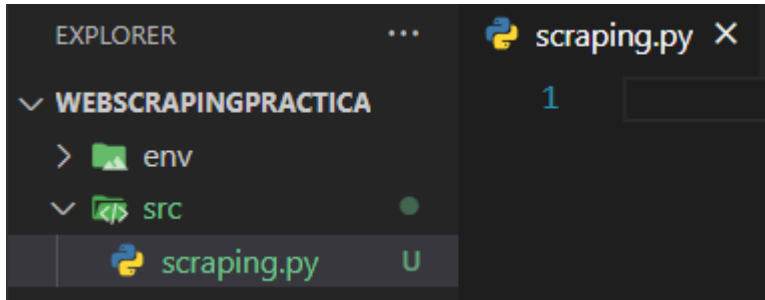
```
Kevin Jonathan@DESKTOP-SA54AJI MINGW64 ~/Desktop/WebScrapingPractica (main)
● $ mkdir src
(env)
Kevin Jonathan@DESKTOP-SA54AJI MINGW64 ~/Desktop/WebScrapingPractica (main)
● $ cd src
x07(env)
Kevin Jonathan@DESKTOP-SA54AJI MINGW64 ~/Desktop/WebScrapingPractica/src (main)
● $ touch scraping.py
x07(env)
```

Con “code scraping.py” abrimos el archivo, aunque desde el área de trabajo también podemos hacerlo.

Manera 1:

```
Kevin Jonathan@DESKTOP-SA54AJI MINGW64 ~/Desk
$ code scraping.py
x0a$ \x1b]633\x3bB\x07\x1b]633\x3bB\x07(env)
```

Manera 2:



Codigo Base

1. Importamos las librerías de Selenium y otras librerías que nos ayudaran en la realización del scrapeo.

```
scraping.py
1 from selenium import webdriver
2 from selenium.webdriver.chrome.service import Service
3 from selenium.webdriver.common.by import By
4 import time
5 import csv
```

2. Creamos las variables “web_side” que guardara el URL del lugar donde queramos hacer el WebScraping y también creamos la variable “path” que guardara la ruta donde está nuestro ChromeDiver

```
6
7 web_side = "http://websecgen.unmsm.edu.pe/carne/carne.aspx"
8 path = "C:\\Program Files (x86)\\chromedriver.exe"
```

3. Creamos la variable “driver” que llamara al Chromedriver.exe mediante la función de selenium “.webdriver.chrome.service” importando Service de la biblioteca. Luego iniciamos la página mediante el método “get()”.

```
10 driver = webdriver.Chrome(service=Service(path))
11 driver.get(web_side)
```

Llegado a este punto podremos realizar webscraping a cualquier página, ahora la lógica para:

1. Navegación por Páginas Web
2. Descarga de Contenido
3. Extracción de Datos
4. Almacenamiento de Datos

Teniendo en cuenta que:

1. Respeto a las Políticas del Sitio Web: Muchos sitios web tienen términos de servicio que prohíben el web scraping. Es importante leer y respetar estas políticas.
2. Frecuencia de Solicitudes: Realizar scraping excesivamente rápido puede sobrecargar los servidores del sitio web objetivo y resultar en bloqueos de IP o en medidas legales.
3. Uso de Datos: Asegúrate de que el uso de los datos extraídos cumpla con las leyes de privacidad y derechos de autor aplicables.

Análisis de la página web

Ya habiendo explicado los que podemos realizar y consideraciones éticas y legales. Vamos a hacer web scraping a la página de carné universitario de la UNMSM [Carné \(unmsm.edu.pe\)](http://unmsm.edu.pe). Utilicen con cuidado esta información.

Este complemento no se admite.

Inicio

Requisitos

Ayuda

República del Perú

Secretaría General de Educación Superior Universitaria

Universidad Nacional José María Arguedas

Nombre

MEDELLA

RUC

Apellido

ELIANA MARILYN

Facultad

INGENIERIA

EPS

CARNE UNIVERSITARIO

2015

Inicio

Requisitos

Ayuda

Código de Matrícula: - (8 caracteres)

Consultar

Facultad:

Programa:

Alumno:

Sólo se muestra información para los alumnos matriculados.
Para más detalles consulte a la ayuda.

SECRETARÍA GENERAL

SIN

IMAGEN

UNMSM

ÁREA DE INFORMÁTICA

© 2008-2017 Secretaría General, Unidad de Informática

e-mail: informatica.secgen@unmsm.edu.pe

Teléfono: 619-7000 anexo 7349

1. Primero entramos a la página y tenemos que analizar los componentes que tiene, que es lo que podemos automatizar, para eso nos basamos en nuestra experiencia en la página y decimos que tiene:

- a. Un lugar para ingresar el código del estudiante, osea un **input**

Código de Matrícula: - (8 caracteres)

- b. Un botón para consultar el carnet.

Consultar

Con esta información ya tenemos los componentes que necesitaremos para la automatización.

Manera de pensar

¿Qué puedo hacer con ese input y botón?

- Puedo ingresar el código de un estudiante y luego consultar, pero si lo hago para un estudiante no me sirve crear tanto para tan poco.
- Si pudiera hacer un bucle para consultar dentro de un rango de estudiantes porque sé que el código de estudiantes es sucesivo, 21200001 hasta 20200305 son los códigos de estudiantes de la FISI de la base 21.
- Ahora donde guardo esa información, tengo que tener un lugar para guardar los datos de esos estudiantes. Si creo un archivo CSV para guardar los datos.

Realización del Web Scraping

1. Creamos el lugar donde se guardaran nuestros datos.

```
14 datos_totales = []
15 nombre_archivo = "FISI_prueba.csv"
```

Creamos estas 2 variables, la primera una lista para guardar los datos y el segundo una cadena para el nombre del archivo.

2. Realizar un bucle for para iterar en cada estudiante

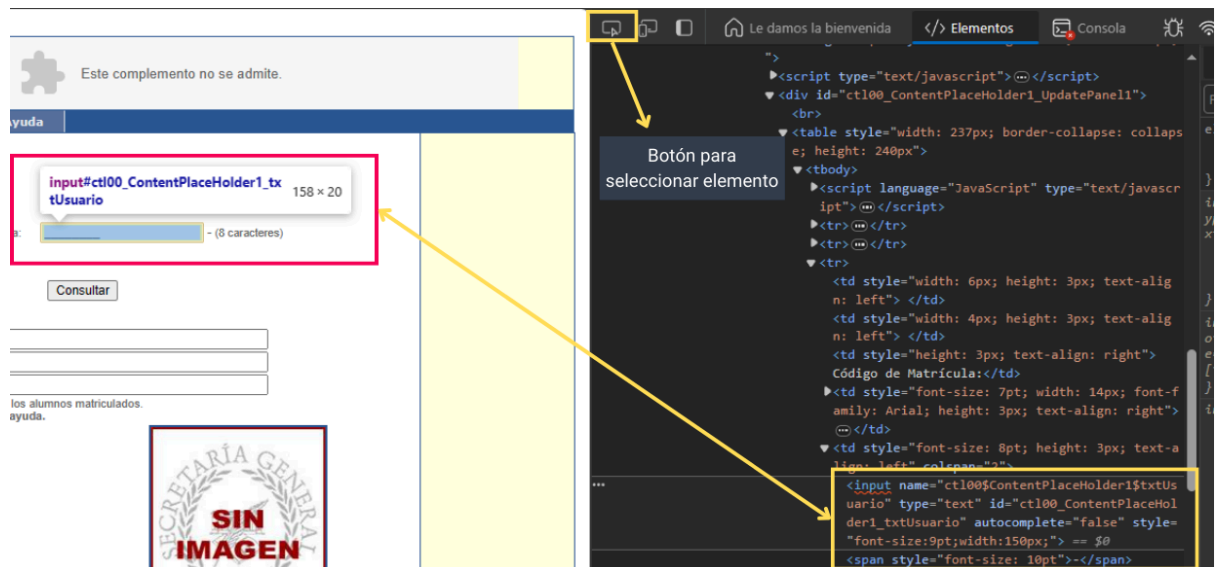
```
17 for codigo in range(codigo_menor, codigo_mayor):
18     pass
```

Cada colocamos un rango para que itere dentro de los códigos, en este caso estoy poniendo código menor y código mayor, para no utilizar el código de algún estudiante de la universidad, ustedes tendrán que poner en vez de esas 2 variables los códigos en el rango que quieran que iteren.

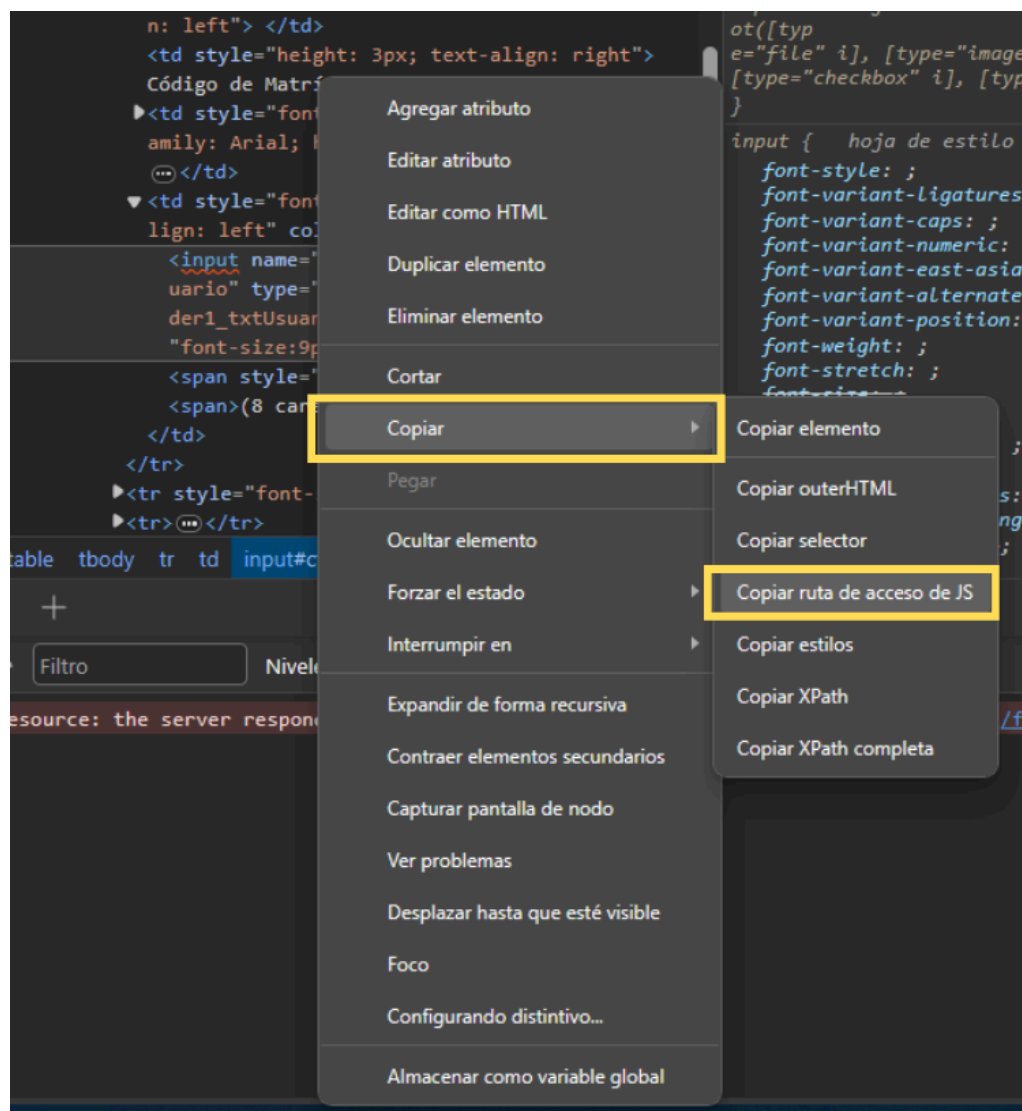
3. Ahora tenemos que acceder al input de la página y para ingresar el código de cada estudiante ahí, para ello utilizamos el método “execute_script” pero a la variable “drive” que era el webdriver

```
17 for codigo in range(21200000, 21200305):
18     driver.execute_script()
```

este método nos permite realizar acciones dentro de la página web, dentro de esos paréntesis ponemos al elemento le queremos realizar la acción. Para eso vamos a la página web y vemos:



Pulsamos el botón para seleccionar elemento y apuntamos al cuadro del input, esto nos direccionará al apartado de HTML donde se encuentra ese input.



Damos anticlick → Copiar → Copiar ruta de acceso de JS para copiar el “document.querySelector("#ctl00_ContentPlaceholder1_txtUsuario")” que será el elemento con el cual interactuemos.

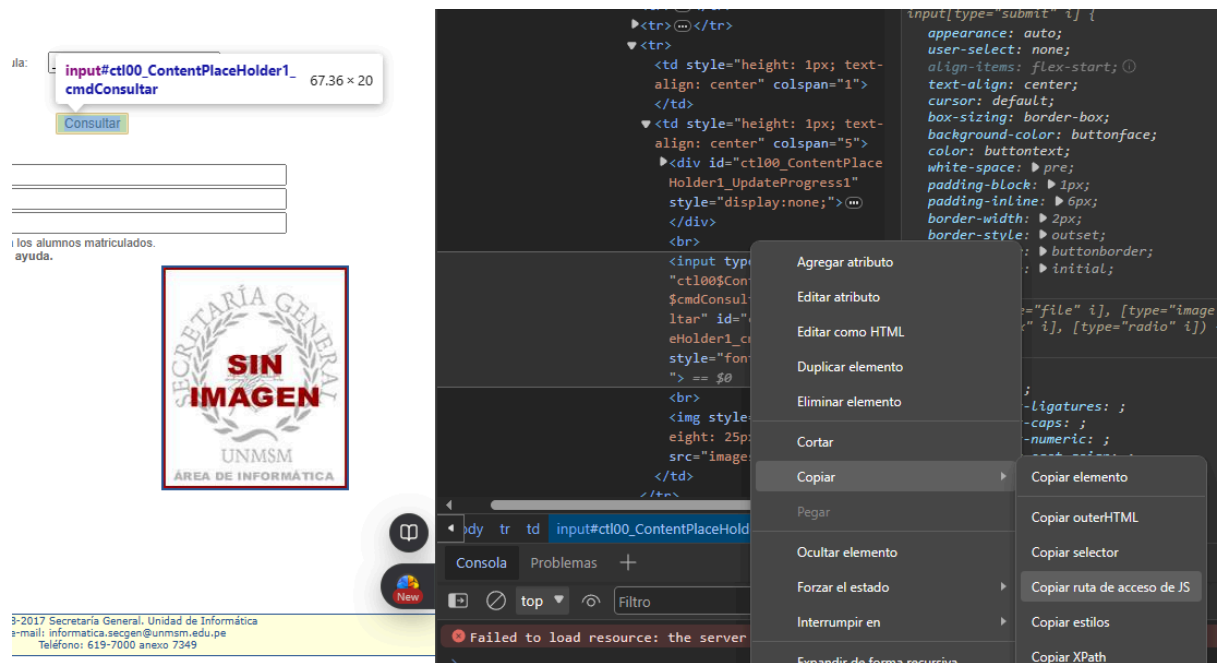
```
for codigo in range(21200000,21200305):  
    driver.execute_script('document.querySelector  
        ("#ctl00_ContentPlaceholder1_txtUsuario")')
```

Insertamos entre comillas simples dentro de los parentesis, ahora para ingresar un valor utilizaremos el “.value = ''” que nos permitirá ingresar valores dentro del input, para este caso queremos ingresar la variable “código”.

```
17 for codigo in range(21200000,21200305):  
18     driver.execute_script(f"document.querySelector  
        ('#ctl00_ContentPlaceholder1_txtUsuario').value = '{codigo}')
```

Ponemos “codigo” entre llaves mediante f que nos permita ingresar variables dentro de una cadena.

Ahora para presionar el boton consultar sería lo mismo.



Copiamos su ruta de acceso JS y lo pegamos en el codigo dentro del método “execute_script”

```

17 for codigo in range(21200000,21200305):
18     driver.execute_script(f"document.querySelector
        ('#ctl00_ContentPlaceHolder1_txtUsuario').value = '{codigo}')"
19
20     driver.execute_script("document.querySelector
        ('#ctl00_ContentPlaceHolder1_cmdConsultar').click();")

```

Pero ahora para hacer click al botón ponemos “.click();” al final de la ruta JS. Esto nos permitirá presionar el botón para la consulta en la página web.

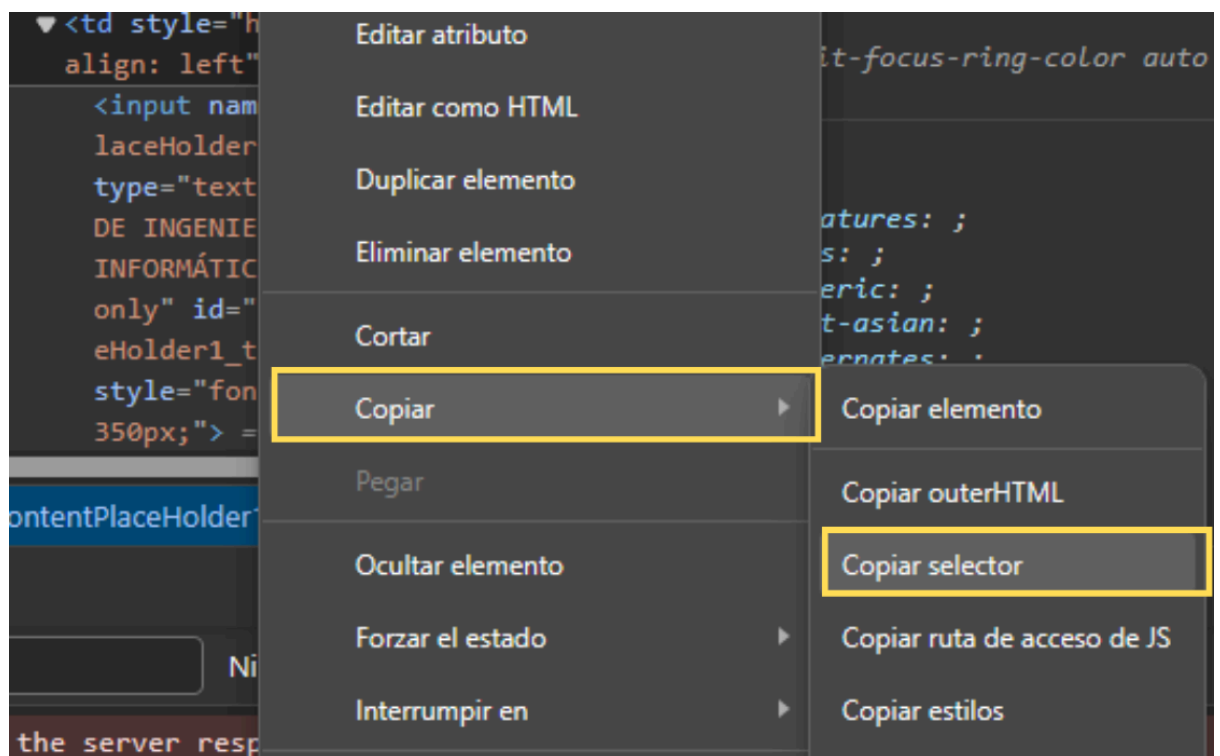
Al hacer click ya habiendo imputado el código se mostrará en la pantalla la acción subsiguiente que es mostrar la información del estudiante de esta manera.

Facultad:

Programa:

Alumno:

Sólo se muestra información para los alumnos matriculados.
Para más detalles consulte a la ayuda.



Importante:

Esta vez ya no copiaremos la ruta de acceso de JS, sino que solo copiaremos el selector, ya que solo vamos a extraer información.

```

21
22 facultad = driver.find_element(By.ID,
        "ctl00_ContentPlaceHolder1_txtFacultad").get_attribute("value")
23

```


1. Guardamos el texto en la variable facultad.
2. driver: Es la instancia del navegador web que se está controlando con Selenium.
3. find_element(By.ID, "ctl00_ContentPlaceHolder1_txtFacultad"): Es un método que busca el primer elemento en la página web que tenga el atributo id igual a "ctl00_ContentPlaceHolder1_txtFacultad".
 - a. By.ID: Es una estrategia de localización que indica que estamos buscando un elemento por su id.
 - b. "ctl00_ContentPlaceHolder1_txtFacultad": Es el valor del id del elemento que estamos buscando.
4. .get_attribute("value"): Una vez que se encuentra el elemento, este método obtiene el valor del atributo especificado del elemento. En este caso, obtiene el valor del atributo "value".

Pero que pasaría si no hubiera datos de un estudiante

Código de Matrícula: - (8 caracteres)

Facultad:

Programa:

Alumno:

Sólo se muestra información para los alumnos matriculados.
Para más detalles consulte a la ayuda.

Carné no tramitado.



Para este caso ponemos una condición de que si al ingresar el código y darle click, el valor de la facultad es diferente a vacío, que continúe con la lógica, caso contrario que el bucle itere otra vez con el siguiente valor del “código”.

```

24     if(facultad != ""):
25         escuela = driver.find_element(By.ID,
26                                     "ctl00_ContentPlaceHolder1_txtPrograma").get_attribute("value")
27         nombre = driver.find_element(By.ID,
28                                     "ctl00_ContentPlaceHolder1_txtAlumno").get_attribute("value")
29     else:
30         continue

```

1. Condición If:

- a. if(facultad != ""):: Comprueba si la variable facultad no está vacía.

2. Extracción de Datos:

- a. escuela = driver.find_element(By.ID, "ctl00_ContentPlaceHolder1_txtPrograma").get_attribute("value"): Si facultad no está vacía, encuentra el campo de entrada con el id ctl00_ContentPlaceHolder1_txtPrograma y obtiene su valor.
- b. nombre = driver.find_element(By.ID, "ctl00_ContentPlaceHolder1_txtAlumno").get_attribute("value"): Luego, encuentra el campo de entrada con el id ctl00_ContentPlaceHolder1_txtAlumno y obtiene su valor.
Con esto terminamos de extraer todos los datos importantes de la página.

3. Continuar en Caso Contrario:

- a. else: continue: Si facultad está vacía, continúa con la siguiente iteración del bucle.

Tiempo de espera

A veces las páginas web tardan en cargar o al realizar una petición tarda un tiempo en responder, para ello tenemos que poner tiempos de espera, para nuestro caso tenemos que poner antes de cada petición o validación

```

17 for codigo in range(21200000,21200305):
18     time.sleep(0.5)
19     driver.execute_script(f"document.querySelector
20         ('#ctl00_ContentPlaceholder1_txtUsuario').value =
21     driver.execute_script(f"document.querySelector
22         ('#ctl00_ContentPlaceholder1_txtFacultad').click(
23     facultad = driver.find_element(By.ID,
24         "ctl00_ContentPlaceholder1_txtFacultad").get_attri
25         ("value")
26     time.sleep(0.5)
27     if(facultad != ""):
28         escuela = driver.find_element(By.ID,

```

Tiempo de espera para las peticiones

Guardado de datos

Ahora tenemos que guardar los datos, para eso

```

26     nombre = driver.find_element(By.ID,
27         "ctl00_ContentPlaceholder1_txtAlumno").get_attribute("value")
28     datos_totales.append([codigo, escuela, nombre])
29 else:
30     continue

```

Guardamos en la lista que creamos al principio una lista con los datos de cada alumno por iteración, mediante el método “append”.

Una vez habiendo iterado por todos los códigos de los estudiantes, guardamos los datos extraídos en nuestra carpeta para poder utilizarla luego como data.

```

32 with open(nombre_archivo, mode='w', newline='', encoding='utf-8') as
33     archivo_csv:
34         escritor_csv = csv.writer(archivo_csv)
35         for datos_iteracion in datos_totales:
36             escritor_csv.writerow(datos_iteracion)

```

- Apertura del Archivo:

- `with open(nombre_archivo, mode='w', newline="", encoding='utf-8') as archivo_csv:`
 - `nombre_archivo`: Es el nombre del archivo CSV donde se escribirán los datos.
 - `mode='w'`: Abre el archivo en modo escritura. Si el archivo ya existe, se sobrescribirá.
 - `newline=""`: Evita la adición de líneas en blanco adicionales en algunos sistemas operativos.
 - `encoding='utf-8'`: Asegura que el archivo se escribe en formato UTF-8, lo que es útil para manejar caracteres especiales y acentos.
- Creación del Escritor CSV:


```
escritor_csv = csv.writer(archivo_csv)
```

 - `csv.writer(archivo_csv)`: Crea un objeto escritor que se utilizará para escribir filas en el archivo CSV.
- Escritura de Datos:


```
for datos_iteracion in datos_totales:
```

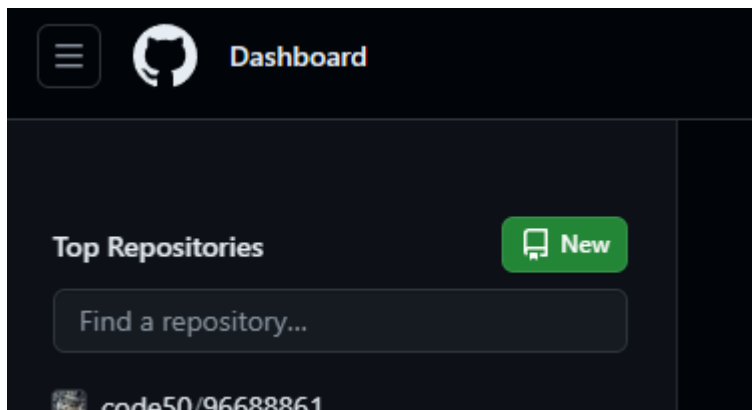
 - Itera sobre `datos_totales`, que es una colección (por ejemplo, una lista de listas) donde cada elemento representa una fila de datos a escribir en el archivo.

```
escritor_csv.writerow(datos_iteracion):
```

 - Escribe cada `datos_iteracion` como una fila en el archivo CSV.

Guardar proyecto en Github

1. Creamos un repositorio en github




Click en el botón verde “New”

2. Ponemos el nombre del repositorio, lo ponemos en público o privado y agregamos un Readme. Presionamos el boton Create repository

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk ().*


Owner *
 Jonathan101120

Repository name *
WebScrapingPractica


✔ WebScrapingPractica is available.

Great repository names are short and memorable. Need inspiration? How about [didactic-octo-tribble](#) ?

Description (optional)

☒  **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

Initialize this repository with:

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore


.gitignore template: **None**


Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: **None**

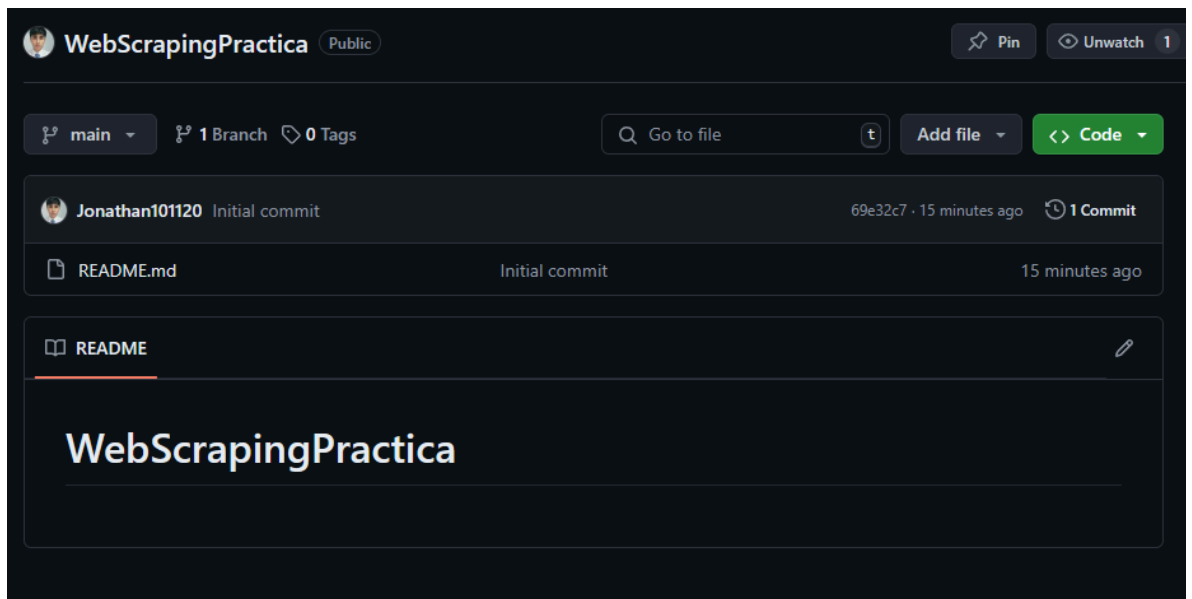
A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set  **main** as the default branch. Change the default name in your [settings](#).

 You are creating a public repository in your personal account.

Create repository

3. Dentro del repositorio creado buscaremos la ruta de nuestro repositorio, en este caso copiaremos el HTTPS el cual se encuentra en el botón verde de nombre Code.



4. Nos dirigimos al nuestro entorno de trabajo para subir nuestro proyecto al repositorio.
 - a. Primero agregamos nuestro archivo scraping.py al área para confirmar que se subirá al repositorio, mediante el comando “git add” + la ubicación del archivo, solo nombre del archivo o todo en general con solo escribir . (punto) que agrega todo lo que está en la carpeta, si es que nos encontramos en la misma carpeta.

```
Kevin Jonathan@DESKTOP-SA54AJI MINGW64 ~/Desktop/WebS
● $ git add .
(env)
```

- b. Con “git status” veremos los archivos que estan para agregar a nuestro repositorio.

```
Kevin Jonathan@DESKTOP-SA54AJI MINGW64 ~/Desktop/WebS
● $ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   requirements.txt
    modified:   src/scraping.py

(env)
```

- c. Segundo agregamos un comentario para que sepan que es lo que estamos agregando a nuestro repositorio, mediante el comando “git commit -m “comentario””. Comentario sería el comentario que harías.

```
Kevin Jonathan@DESKTOP-SA54AJI MINGW64 ~/Desktop/WebSc
● $ git commit -m "Archivo mas sus requerimientos"
[master 592b411] Archivo mas sus requerimientos
 2 files changed, 17 insertions(+), 1 deletion(-)
 create mode 100644 requirements.txt
(env)
```

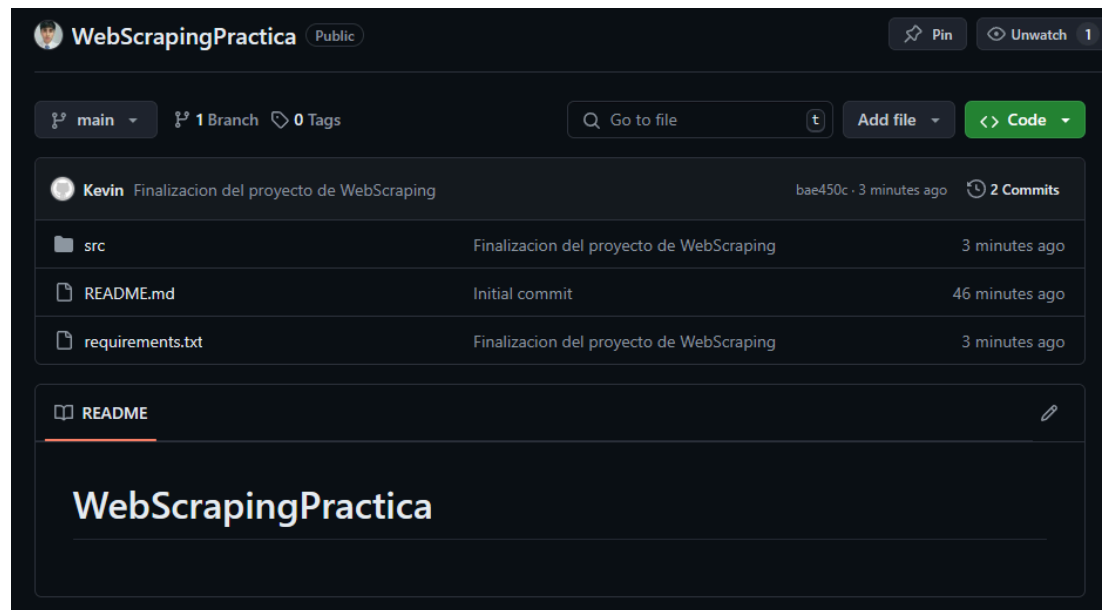
- d. Tercero, le brindamos a git el lugar donde se subirá el archivo, mediante el comando “git remote add origin <https://github.com/Jonathan101120/WebScrapingPractica.git>”.

```
Kevin Jonathan@DESKTOP-SA54AJI MINGW64 ~/Desktop/WebScrapingPractica (main)
$ git remote add origin https://github.com/Jonathan101120/WebScrapingPractica.git
(env)
```

- e. Cuarto, damos la confirmación de subir el archivo, mediante el comando “git push -u origin main”, pero antes ponemos “git pull origin main --rebase” si el repositorio remoto fue creado con algún archivo (por ejemplo, un README.md o .gitignore), y tú no tienes esos cambios en tu repositorio local, para evitar conflictos al momento de subir nuestro proyecto.

```
Kevin Jonathan@DESKTOP-SA54AJI MINGW64 ~/Desktop/WebScrapingPractica (main)
$ git pull origin main --rebase
From https://github.com/Jonathan101120/WebScrapingPractica
 * branch          main          -> FETCH_HEAD
Successfully rebased and updated refs/heads/main.
(env)
Kevin Jonathan@DESKTOP-SA54AJI MINGW64 ~/Desktop/WebScrapingPractica (main)
$ git push -u origin main
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 12 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 1.15 KiB | 1.15 MiB/s, done.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Jonathan101120/WebScrapingPractica.git
 69e32c7..bae450c  main -> main
branch 'main' set up to track 'origin/main'.
(env)
```

- f. Iremos a nuestro repositorio en GitHub, luego de actualizar la página, veremos que se ha subido nuestra carpeta src y dentro de ella estará nuestro archivo “scraping.py”

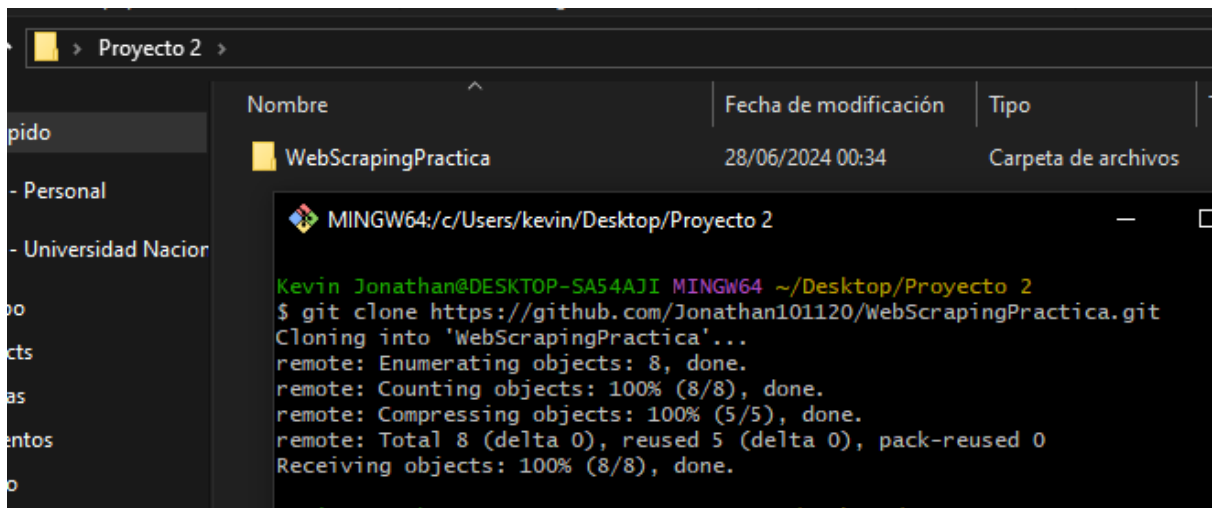


- g. Hasta este punto ya habremos terminado todo el proceso de crear un proyecto, aunque es pequeño, se explicó lo general para poder crear más proyectos a futuro y subirlos a un repositorio en GitHub

Link: [Jonathan101120/WebScrapingPractica \(github.com\)](https://github.com/Jonathan101120/WebScrapingPractica)

Clonar un repositorio

1. Crea una Carpeta: Crea una carpeta donde quieras clonar tu repositorio. Esto no es obligatorio, pero ayuda a mantener tus proyectos organizados.
2. Abrir la Terminal de Git Bash: Navega a la carpeta creada en el paso anterior y abre Git Bash allí. Puedes hacer esto haciendo clic derecho dentro de la carpeta y seleccionando "Git Bash Here" (en Windows) o simplemente navegando a la carpeta usando la terminal.
3. Clonar el Repositorio: Usa el comando `git clone` seguido de la URL de tu repositorio. Esto descargará una copia del repositorio en tu máquina local.



4. Entramos al entorno de trabajo y creamos el entorno virtual para ejecutar el proyecto, luego lo activamos y al final instalamos los paquetes que teníamos en el archivo txt de “requirements.txt”.

```
Kevin Jonathan@DESKTOP-SA54AJI MINGW64 ~/Desktop/Proyecto 2/WebScrapingPractica (main)
$ virtualenv -p python3 env
created virtual environment CPython3.11.5.final.0-64 in 481ms
  creator CPython3Windows(dest=C:\Users\Beatriz\Desktop\Proyecto 2\WebScrapingPractica\env,
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy)
  added seed packages: pip==24.1, setuptools==70.1.0, wheel==0.43.0
  activators BashActivator,BatchActivator,FishActivator,NushellActivator,PowerShellActivator,Python
  \x1b]633\x3bB\x07(env)
Kevin Jonathan@DESKTOP-SA54AJI MINGW64 ~/Desktop/Proyecto 2/WebScrapingPractica (main)
$ source env/Scripts/activate
\x1b]633\x3bB\x07(env)
Kevin Jonathan@DESKTOP-SA54AJI MINGW64 ~/Desktop/Proyecto 2/WebScrapingPractica (main)
$ pip install -r requirements.txt
Collecting attrs==23.2.0 (from -r requirements.txt (line 1))
  Using cached attrs-23.2.0-py3-none-any.whl.metadata (9.5 kB)
Collecting certifi==2024.6.2 (from -r requirements.txt (line 2))
  Using cached certifi-2024.6.2-py3-none-any.whl.metadata (2.2 kB)
Collecting cffi==1.16.0 (from -r requirements.txt (line 3))
  Using cached cffi-1.16.0-cp311-cp311-win_amd64.whl.metadata (1.5 kB)
Collecting h11==0.14.0 (from -r requirements.txt (line 4))
  Using cached h11-0.14.0-py3-none-any.whl.metadata (8.2 kB)
Collecting idna==3.7 (from -r requirements.txt (line 5))
  Using cached idna-3.7-py3-none-any.whl.metadata (9.9 kB)
```

5. Ejecutamos el proyecto y veremos que funciona con normalidad.

Proyectos futuros

1. Análisis de precios entre un grupo de productos.

Links:

- Plaza Vea: <https://www.plazavea.com.pe/>
- Metro: <https://www.metro.pe/>

- Wong: <https://www.wong.pe/?sc=1&spChangeSC=1>
 - Tottus: <https://tottus.falabella.com.pe/tottus-pe>
2. Tomas satelitales:
 - a. [Google Earth](#)
 - b. [SENAMHI - Satelites Goes16](#)
 3. Extracción de datos para cierto estudio:
 - a. [Población Mundial: nan Billones de Personas \(2024\) - Worldometer \(worldometers.info\)](#)
 - b. [Mercados Financieros - Investing.com](#)

Referencias

1. gto76/python-cheatsheet: Comprehensive Python Cheatsheet. (2024). GitHub.
<https://github.com/gto76/python-cheatsheet>
2. LinkedIn. (2024). LinkedIn.com.
[https://www.linkedin.com/pulse/c%C3%B3mo-localizo-elementos-para-los-test-con-selenium-gnocchi-/](https://www.linkedin.com/pulse/c%C3%B3mo-localizo-elementos-para-los-test-con-selenium-gnocchi/)
3. Automate the Boring Stuff with Python. (2024). Automatetheboringstuff.com.
<https://automatetheboringstuff.com/2e/chapter12/>