🕵 ¿Cómo sacar información de una página sin API?

(Web Scraping con Python)

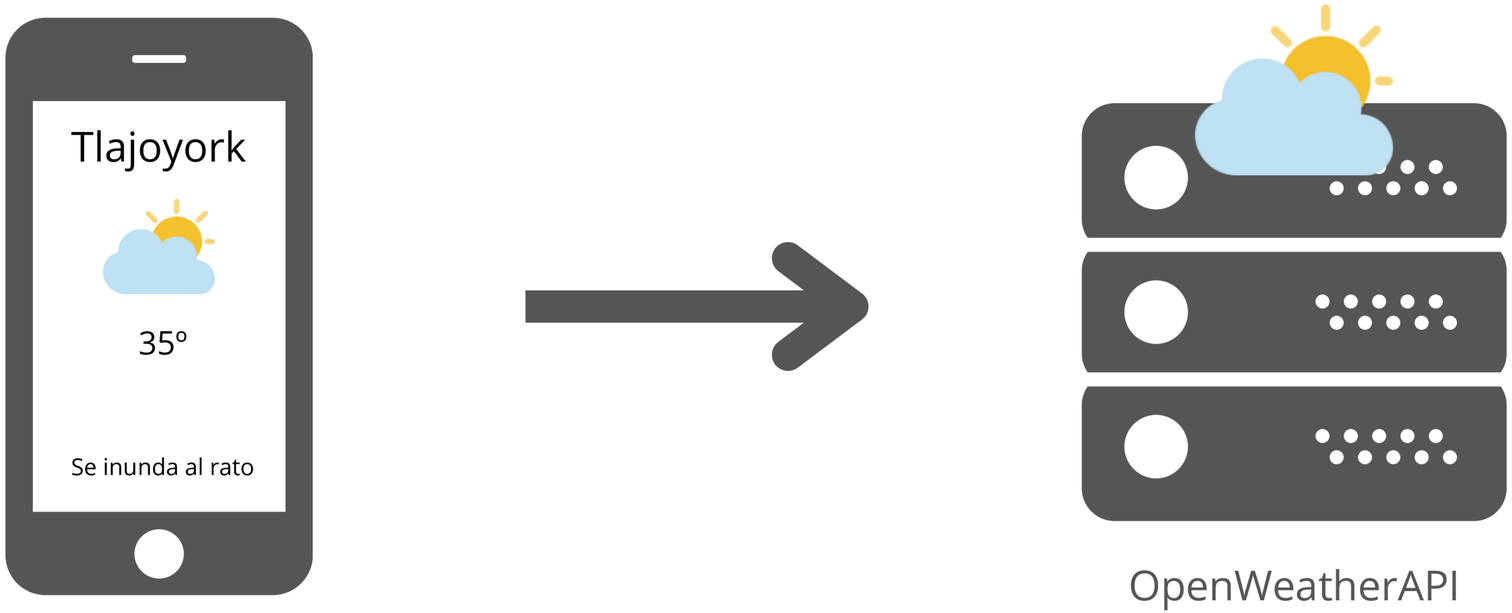
👨‍💻 ¿Quién soy yo?

Daniel Hernández Muñoz
(Lord Friky)

- Jr Software Engineer en WM Asesoria 💼
- Estudiante de Ingeniería Informática en UdG CUCEI 🦁
- Desarrollador y contribuidor de proyectos de código abierto desde los 16 💻
- Pythonista desde los 18 🐍

lordfriky daniel-hdezm

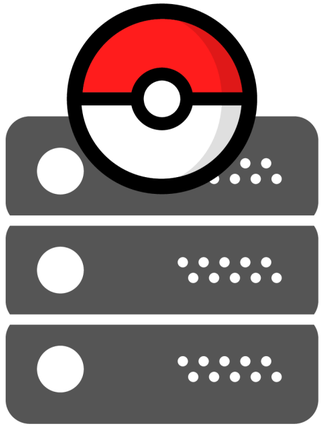
Las API



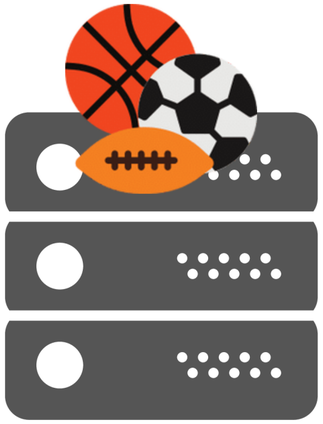
¿Qué hacemos si queremos hacer una aplicación,
que consulte información actualizada regularmente?...

...Usamos una **API**!

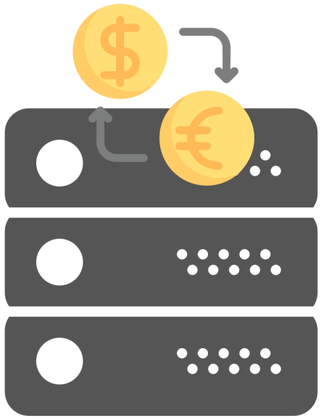
Tenemos APIs públicas para casi todo:



PokéAPI



TheSportsDB



CoinGecko

Reddit

Más en https://github.com/public-apis/public-apis

¿Pero qué pasa con las páginas que no nos ofrecen esta facilidad para consultar su información de manera automática?

Un poco de backstory...

El registro de materias en la Universidad de Guadalajara siempre ha sido estresante

- La oferta de clases casi nunca suele estar completa al momento del registro
- No todas las clases tienen cupos asignados
- No todas las clases tienen la información completa (ej. les faltan horarios, aulas, profesores, etc)
- Suelen agregar materias/cupos durante el registro

Yo me harté de eso, así que decidí hacer un bot que tanto:

- Me avisara cuando hubiera un cambio en la oferta (incluso cuando se liberara un cupo de una clase llena que me interesaba)
- Agendara por mi (**esto no lo vamos a tocar en la plática**)

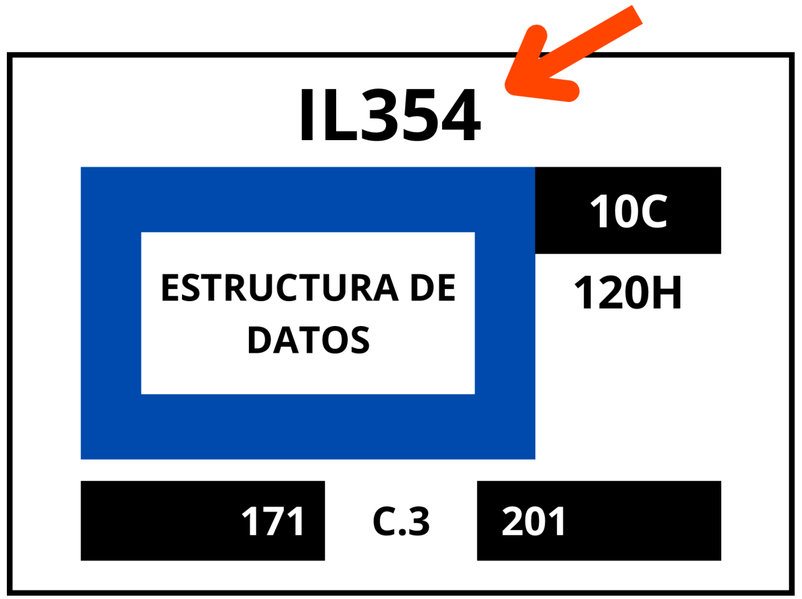
Esa fue mi motivación para aprender Web Scraping

A todo esto, ¿cómo se ve la oferta de SIIAU?

In-depth: Formulario de consulta

- Es requerido **mínimo** el ciclo y el centro
- Si incluimos algo en los demás campos se busca por **match exacto**
- Generalmente los alumnos buscamos algo agregando la **clave de la carrera** y/o la **clave de la materia**

¿De donde sacamos las claves de las materias?



¡De nuestra malla de estudios!

(Un poco irrelevante pero ayuda al contexto)

In-depth: Resultado de la consulta

Consulta de Oferta Académica del ciclo 202310														
NRC	**Clave**	**Materia**	**Sec**	**CR**	**CUP**	**DIS**	**Ses/Hora/Días/Edif/Aula/Periodo**						**Ses/Profesor**	
193981	IL345	MATEMATICAS DISCRETAS	D02	8	0	0	01	1100-1255	L	DEDZ2	A013	16/01/23 - 31/05/23	01	GARCIA NAVA, DALMIRO
							01	1100-1255	. . I . . .	DEDP	A016	16/01/23 - 31/05/23		
193982	IL345	MATEMATICAS DISCRETAS	D03	8	0	0	01	1500-1655	. M	DEDU	A021	16/01/23 - 31/05/23	01	VILLALOBOS JIMENEZ, SARA EDITH
							01	1500-1655	. . . J . .	DEDU	A005	16/01/23 - 31/05/23		

Cada clase tiene:

- NRC y Sección

(Estos identifican a la clase en específico)

- Clave, nombre y créditos

(Estos se comparten entre todas las clases de dicha materia)

- ¿Sesiones?

(Estas incluyen el profesor que imparte cada sesión, en las horas que lo hace, los dias, en qué salón y el periodo)

¿SIIAU no tenía un captcha para prevenir bots?

¡No!
Su captcha no sirve para nada :)

(sólo se valida localmente)

Además, puedes encontrar un enlace directo a la oferta en Google, para evitar la autenticación!

http://consulta.siiau.udg.mx > sspseca.forma_consulta

Forma de consulta de Oferta Académica

Para consultar la **Oferta Académica** introduce el patrón de búsqueda que te servirá para filtrar los cursos. Recuerda...

Bien... ¿Pero ahora cómo consultamos páginas web en Python?

Requests +
BeautifulSoup

Selenium

Scrapy

Requests + BeautifulSoup

Selenium

No usaremos Scrapy hoy, pero puede ser una buena alternativa para investigar!

Scrapy

Requests + BeautifulSoup

- Sólo carga HTML estático 📄
- Funciona rápidamente 🚀
- Consume pocos recursos 🔋
- No puede hacer clic, scroll ni esperar eventos ❌

¡Mejor para páginas simples y estáticas!

Selenium

- Soporta JavaScript, si es que hay contenido dinamico 📊
- Es más lento, al requerir renderizar todo 🐢
- Requiere de más recursos al usar un navegador completo 🔥
- Puede simular la interacción completa del usuario 🤳

¡Mejor para contenido cargado dinámicamente!

requests-beautifulsoup.py

```python
import requests
from bs4 import BeautifulSoup

url = "https://example.com"
response = requests.get(url)

soup = BeautifulSoup(response.text, "html.parser")
primer_p = soup.find("p")

# Siempre es buena idea revisar primero
# si un elemento existe
if primer_p:
    print(primer_p.get_text())
```

selenium.py

```python
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from webdriver_manager.chrome import ChromeDriverManager

options = webdriver.ChromeOptions()
options.add_argument("--headless")  # Ejecuta sin abrir ventana

driver = webdriver.Chrome(
  service=Service(ChromeDriverManager().install()),
  options=options
)
driver.get("https://example.com")

primer_p = driver.find_element(By.TAG_NAME, "p")

if primer_p:
    print(primer_p.text)

driver.quit()
```

Para efectos de esta plática usaremos Requests + BeautifulSoup

¿Cómo encontramos elementos?

Tanto BeautifulSoup como Selenium nos dan métodos para buscar tanto el primer elemento con ciertas características como todas sus instancias (aunque Selenium puede esperar a que un elemento aparezca).

BeautifulSoup	Selenium
`soup.find()`	`driver.find_element()`
`soup.find_all()`	`driver.find_elements()`

Características por las que podemos buscar un elemento:

- Por su etiqueta

```
soup.find("p")
```

- Por sus clases

```
soup.find(class_="a b")
```

- Por su texto exacto

```
soup.find(string="texto")
```

- Por su ID

```
soup.find(id="mi-id")
```

- Por sus atributos

```
soup.find("div", {"atr": "val"})
```

Etcétera...
(revisar la documentación de cada uno para casos más específicos)

Manejando elementos

Una vez que hayamos encontrado el elemento que buscábamos, podemos usar los siguientes métodos/propiedades para trabajar con él:

Los elementos de **BeautifulSoup** los denotaremos con *tag*, y los de **Selenium** con *element*

- Obtener su texto

```
tag.get_text()
element.text
```

- Obtener algun atributo

```
tag[“atr”] ó tag.get(“atr”)
element.get_attribute(“atr”)
```

- Obtener su etiqueta

```
tag.name
element.tag_name
```

- Navegar por sus hijos (de la misma manera en que los encontramos)

```
tag.find(...)                element.find_element(...)
tag.find_all(...)            element.find_elements(...)
```

Ahora, ¿Cómo se ve el HTML del SIIAU?

(Spoiler: **¡Horrible!**)

formulario_consulta.html

```
<SELECT NAME="ciclop" SIZE="1" id=cicloID>
<OPTION value='2025D'>2025D - Promocion 2025D
<OPTION value='2025C'>2025C - Promocion 2025C
<OPTION value='202580'>202580 - Ciclo de Verano 2025
<OPTION value='202520'>202520 - Calendario 25 B
<OPTION value='202510'>202510 - Calendario 25 A
... El resto de los ciclos ...
</SELECT>

<SELECT NAME="cup" SIZE="1">
<OPTION VALUE="3">3 - C. U. DE TLAJOMULCO
<OPTION VALUE="4">4 - C. U. DE GUADALAJARA
<OPTION VALUE="5">5 - C.U. DE TLAQUEPAQUE
<OPTION VALUE="6">6 - C.U. DE CHAPALA
<OPTION VALUE="A">A - C.U. DE ARTE, ARQ. Y DISEÑO
<OPTION VALUE="B">B - C.U. DE CS. BIOLOGICO Y AGR.
... El resto de los centros ...
</SELECT>
```

Nunca se cierran los <select> de las opciones que nos importan

Las tablas no incluyen el <thead> ni el <tbody>

resultados_consulta.html

```
<TABLE  BORDER=1 cellspacing=0 cellpadding="0">
<TR bgcolor=navy>
<TH COLSPAN="11">Consulta de Oferta Acad&eacute;mica del ciclo 202510</TH>
</TR>
<TR bgcolor=A9C0DB>
<TH>NRC</TH>
<TH>Clave</TH>
<TH>Materia</TH>
<TH>Sec</TH>
<TH>CR</TH>
<TH>CUP</TH>
<TH>DIS</TH>
<TH>Ses/Hora/D&iacute;as/Edif/Aula/Periodo</TH>
<TH>Ses/Profesor</TH>
</TR>
<TR style="background-color:#e5e5e5;">
<TD class=tddatos>210900</TD>
<TD class=tddatos><A HREF="*irrelevante-y-largo*">V0731</A></TD>
<TD class=tddatos><A HREF="*irrelevante-y-largo*">ESTRUCTURAS DE DATOS</A></TD>
<TD class=tddatos>D01</TD>
... Resto de la información ...
```

Solución: Usar un parser más o menos flexible! (según tus necesidades)

diferente_parser.py

```
# Parser por defecto
#   Te da la información de la manera más acercada
#   a como viene en el documento

soup1 = BeautifulSoup(documento_html, "html.parser")


# Parser más flexible
#   Te da la información de la manera más cercana a
#   a como esperarías en el navegador

# pip install html5lib
soup2 = BeautifulSoup(documento_html, 'html5lib')
```

Nota: Al trabajar con Selenium será más similar como con el parser más flexible!

Plan de trabajo 📝

Para el **formulario** de consulta:

- Extraer las opciones de ciclo
- Extraer las opciones de centros
- Averiguar los parámetros necesarios para realizar la consulta

Para los **resultados** de la consulta:

- Determinar cómo vamos a presentar la información
- Parsearla acordemente

Ahora sí

¡A trabajar!

extraer_ciclos_y_centros.py

```python
import requests
from bs4 import BeautifulSoup

url_formulario = "http://consulta.siiau.udg.mx/wco/sspseca.forma_consulta"
formulario_consulta = requests.get(url_formulario)
soup = BeautifulSoup(formulario_consulta.text, 'html5lib')



ciclos = [] # Tuplas de (clave, descripcion)
ciclos_select = soup.find('select', {'name': 'ciclop'})

for option in ciclos_select.find_all('option'):
    clave = option['value']
    texto = option.get_text(strip=True)
    descripcion = texto.split(" - ", 1)[-1]
    ciclos.append((clave, descripcion))



centros = [] # Lo mismo que ciclos
centros_select = soup.find('select', {'name': 'cup'})

for option in centros_select.find_all('option'):
    clave = option['value']
    texto = option.get_text(strip=True)
    descripcion = texto.split(" - ", 1)[-1]
    centros.append((clave, descripcion))
```

← Hacemos el request para obtener el formulario

← Extraemos los ciclos por medio de su argumento *ciclop*

← Extraemos los centros por medio de su argumento *cup*

Revisando el HTML del formulario podemos darnos cuenta que los siguientes son la **dirección** y sus **argumentos** para realizar la consulta de la oferta

payload_para_consulta.py

```python
url_consulta = "http://consulta.siiau.udg.mx/wco/sspseca.consulta_oferta"

payload = {
    'ciclop': '', # Clave de ciclo - requerido
    'cup': '', # Clave de centro - requerido

    'majrp': '', # Clave de carrera en mayúsculas
    'crsep': '', # Clave de materia en mayúsculas
    'materiap': '', # Descripción de materia en mayúsculas

    'horaip': '', # Hora de inicio
    'horafp': '', # Hora de fin

    'lup': '', # M si filtramos por lunes
    'map': '', # T si filtramos por martes
    'mip': '', # W si filtramos por miercoles
    'jup': '', # R si filtramos por jueves
    'vip': '', # F si filtramos por viernes
    'sap': '', # S si filtramos por sabado

    'edifp': '', # Clave de edificio
    'aulap': '', # Clave de aula

    'dispp': '', # D si queremos sólo las clases con cupos disponibles
    'ordenp': '1', # Orden: 1 = materia, 2 = clave, 3 = nrc

    'mostrarp': 999_999_999_999_999, # Cantidad de resultados a mostrar
    'p_start': 0 # Offset para paginación, podemos dejarlo en 0
}
```

clase.py

```python
propuesta_formato_clase = {
    # Datos generales
    'clave': '',
    'nombre': '',
    'creditos': 0,

    # Datos específicos
    'nrc': '',
    'seccion': '',
    'cupo': 0,
    'disponibles': 0,

    'sesiones': [
        {
            'id': 0,
            'profesor': 'Juan Perez',
            'horarios': [
                {
                    'hora_inicio': '08:00',
                    'hora_fin': '09:00',
                    'dias': ['L', 'M', 'I', 'J', 'V', 'S'],

                    'edificio': '',
                    'aula': '',
                    'periodo': ''
                }
            ]
        }
    ]
}
```

Debido a las observaciones hechas en la consulta de resultados, considero que este es un buen formato para presentar la información de cada clase.

Ahora sólo parseamos la información como decidimos presentarla...

consulta_oferta.py

```
respuesta_consulta = requests.post(url_consulta, data=payload)
soup = BeautifulSoup(formulario_consulta.text, 'html5lib')

clases = []
# Recordamos que html5lib agrega el tbody
clases_tabla = soup.find('tbody')

# Buscamos sólo los hijos inmediatos y evitamos el header
for fila_clase in clases_tabla.find_all('tr', recursive=False)[2:]:
    # Buscamos de nuevo los hijos inmediatos por las tablas de sesiones
    celdas = fila_clase.find_all('td', recursive=False)

    # Usaremos funciones de caja negra para simplificar el ejemplo
    clase = {
        'clave': extraer_texto_de_link(celdas[1]),
        'nombre': extraer_texto_de_link(celdas[2]),
        'creditos': int(celdas[4].get_text(strip=True)),

        'nrc': celdas[0].get_text(strip=True),
        'seccion': celdas[3].get_text(strip=True),
        'cupo': int(celdas[5].get_text(strip=True)),
        'disponibles': int(celdas[6].get_text(strip=True)),
        'sesiones': vincular_sesiones(
            horarios = celdas[7],
            profesores = celdas[8]
        )
    }

    clases.append(clase)
```

Por último, creamos una interfaz bonita para trabajar con lo que hemos hecho :)

SiiauOferta.py

```python
from siiau import SiiauOferta

ciclos = SiiauOferta.obtener_ciclos()
centros = SiiauOferta.obtener_centros()

clases = SiiauOferta.consultar_oferta(payload)
```

¡Tiempo de Demostración!

¿Alguna duda?

¡Muchas gracias por asistir a mi plática! 🫂

lordfriky daniel-hdezm