



Web scraping con Python. Extraer datos de una web. Guía de inicio de BeautifulSoup

- Categoría: [Tutoriales Python](https://j2logo.com/category/blog/tutoriales-python/) (<https://j2logo.com/category/blog/tutoriales-python/>)
- avanzado (<https://j2logo.com/tag/avanzado/>), [beautifulsoup](https://j2logo.com/tag/beautifulsoup/) (<https://j2logo.com/tag/beautifulsoup/>),
python (<https://j2logo.com/tag/python/>), [scraping](https://j2logo.com/tag/scraping/) (<https://j2logo.com/tag/scraping/>)

El web scraping es una técnica que permite extraer datos e información de una web. Este tutorial es una guía de inicio al web scraping con Python, utilizando para ello la librería *Beautiful Soup*.

Antes de comenzar, quiero resaltar que si quieras realizar *web scraping*, debes ser justo y actuar de forma legal y según las políticas que rijan cada una de las páginas de las que pretendas extraer información.

Lejos de lo que te puedas imaginar y de que pienses que el web scraping es cosa de hackers, lo cierto es que el web scraping permite, por ejemplo, llevar a cabo un análisis SEO de una web, comprobar enlaces rotos, generar el sitemap de una página, vigilar a la competencia o estar al tanto de cambios en una web. Esta técnica también puede utilizarse en las primeras fases de un proyecto de Big Data o Machine Learning, en los que datos e información juegan un papel importantísimo.

Así que, ¿estás preparad@ para aprender web scraping con Python?

Índice

- Web scraping con Python
- Qué es BeautifulSoup
- Pasos para hacer web scraping en Python con BeautifulSoup
- Tipos de objetos de BeautifulSoup
- Navegar a través de los elementos de BeautifulSoup
- Hijos y descendientes de un elemento en BeautifulSoup
- Padres de un elemento
- Buscar elementos en BeautifulSoup

Web scraping con Python

¿Por qué web scraping con Python? Fundamentalmente, por ser un lenguaje intuitivo, fácil de usar y por las librerías y herramientas que han surgido en torno al mismo para llevar a cabo esta técnica.

Yo, en función del proyecto y sus características, normalmente sigo una de estas dos alternativas:



- Utilizar el framework **Scrapy**. Este framework, además de hacer **scraping** (extraer información de una página) permite hacer **crawling** fácilmente (descubrir los enlaces de una web y navegar a través de ellos).

Qué es Beautiful Soup

Beautiful Soup es una librería Python que permite extraer información de contenido en formato *HTML* o *XML*. Para usarla, es necesario especificar un *parser*, que es responsable de transformar un documento *HTML* o *XML* en un árbol complejo de objetos Python. Esto permite, por ejemplo, que podamos interactuar con los elementos de una página web como si estuviésemos utilizando las herramientas del desarrollador de un navegador.

A la hora de extraer información de una web, uno de los parsers más utilizado es el parser *HTML* de *lxml*. Precisamente, será el que utilicemos en este tutorial.

A continuación, te muestro cómo instalar tanto la librería *Beautiful Soup* como el parser *lxml* utilizando el gestor de paquetes *pip*.

Para instalar **Beautiful Soup**, ejecuta el siguiente comando:

```
$> pip install beautifulsoup4
```

Para instalar el parser **lxml**, ejecuta el siguiente comando:

```
$> pip install lxml
```

Pasos para hacer web scraping en Python con Beautiful Soup

Antes de entrar en detalle con los ejemplos, te voy a indicar los pasos que debes seguir para extraer información de una web:

1 – Identifica los elementos de la página de los que extraer la información

Las páginas web son documentos estructurados formados por una jerarquía de elementos. El primer paso para extraer información es identificar correctamente el elemento o elementos que contienen la información deseada.

Para ello, lo más fácil es abrir la página en un navegador e inspeccionar el elemento. Esto se consigue haciendo clic con el botón derecho sobre el elemento en cuestión y pulsando sobre la opción *Inspeccionar* o *Inspeccionar elemento* (depende del navegador).

Quédate con toda la información disponible asociada al elemento (como la *etiqueta*, o los atributos *id* y/o *class*) ya que, posteriormente, te hará falta para utilizarla en *Beautiful Soup*.



petaciones `http://`). El contenido de la respuesta, el que contiene la página en `HTML`, será el que pasemos posteriormente a `BeautifulSoup` para generar el árbol de elementos y poder hacer consultas al mismo.

3 – Crear la «sopa»

El contenido de la página obtenido en el paso anterior será el que utilicemos para crear la «sopa», esto es, el árbol de objetos Python que representan al documento `HTML`.

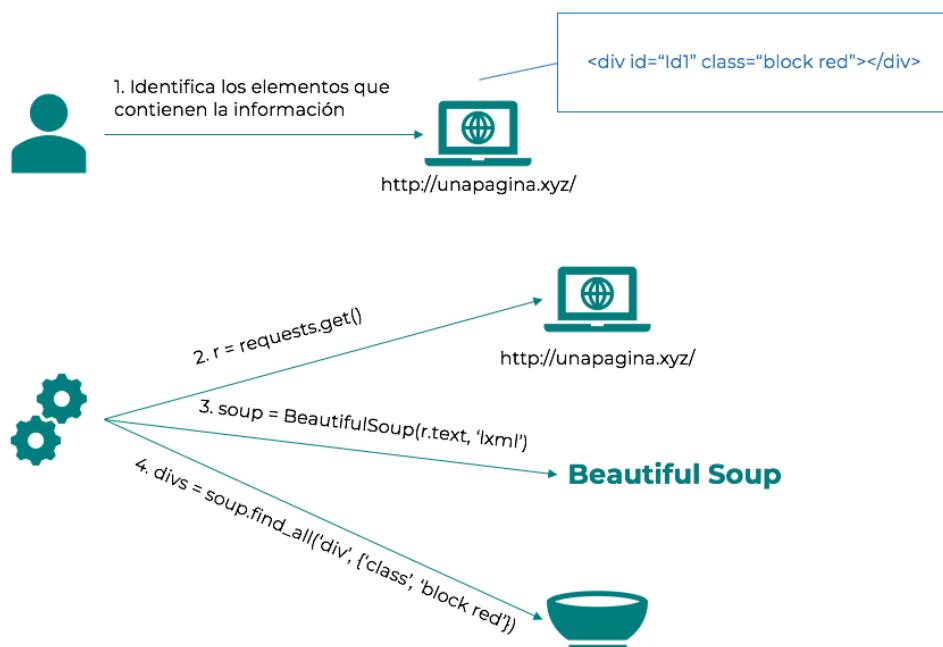
Para ello, hay que crear un objeto de tipo `BeautifulSoup`, al cuál se le pasa el texto en formato `HTML` y el identificador del parser a utilizar:

```

1 import requests
2 from bs4 import BeautifulSoup
3
4 r = requests.get('http://unapagina.xyz')
5 soup = BeautifulSoup(r.text, 'lxml')
```

4 – Busca los elementos en la «sopa» y obtén la información deseada

El último paso es hacer una búsqueda en el árbol y obtener los objetos que contienen la información y datos que necesitamos. Para ello, tienes varias alternativas como veremos en las siguientes secciones.



Tipos de objetos de BeautifulSoup

En la sección anterior hemos visto cómo crear un objeto de tipo `BeautifulSoup`. Este objeto, que representa al árbol de objetos Python resultante de parsear el documento `HTML` de entrada, será el punto de partida para navegar a través de los elementos del árbol, así como para realizar las búsquedas necesarias en el mismo.



información de una web con *Beautiful Soup*.

💡 Imagina que este documento *HTML* se ha obtenido al consultar una página con la librería `requests`.

```
1 <html lang="es">
2 <head>
3     <meta charset="UTF-8">
4     <title>Página de prueba</title>
5 </head>
6 <body>
7     <div id="main" class="full-width">
8         <h1>El título de la página</h1>
9         <p>Este es el primer párrafo</p>
10        <p>Este es el segundo párrafo</p>
11        <div id="innerDiv">
12            <div class="links">
13                <a href="https://pagina1.xyz/">Enlace 1</a>
14                <a href="https://pagina2.xyz/">Enlace 2</a>
15            </div>
16            <div class="right">
17                <div class="links">
18                    <a href="https://pagina3.xyz/">Enlace 3</a>
19                    <a href="https://pagina4.xyz/">Enlace 4</a>
20                </div>
21            </div>
22        </div>
23        <div id="footer">
24            <!-- El footer -->
25            <p>Este párrafo está en el footer</p>
26            <div class="links footer-links">
27                <a href="https://pagina5.xyz/">Enlace 5</a>
28            </div>
29        </div>
30    </div>
31    </body>
32 </html>
```

Bien, teniendo en cuenta este contenido, paso a explicarte los distintos objetos que puedes encontrar en un árbol (o sopa) de *Beautiful Soup*. Pero antes, vamos a crear el objeto `BeautifulSoup` para que podamos *jugar* un poco.



(https://j2logo.com)

```

3 |     contenido =
4 |     4   <html lang="es">
5 |     5   <head>
6 |     6       <meta charset="UTF-8">
7 |     7       <title>Página de prueba</title>
8 |     8   </head>
9 |     9   <body>
10 |    10   <div id="main" class="full-width">
11 |    11       <h1>El título de la página</h1>
12 |    12       <p>Este es el primer párrafo</p>
13 |    13       ...
14 |    14   </div>
15 |    15   </body>
16 |    16   </html>
17 |    17   """
18 |
19 |    19 soup = BeautifulSoup(contenido, 'lxml')
```

Tag

Este objeto se corresponde con una etiqueta *HTML* o *XML*. Por ejemplo, dado el objeto `soup`, podemos acceder al objeto (`tag`) que representa al **título** de la página usando la etiqueta `title`:

```

1 |   >>> soup.title
2 |   <title>Página de prueba</title>
```

Dado un objeto de tipo `Tag`, podemos acceder a sus atributos tratando al objeto como si fuera un diccionario. Además, se puede acceder a ese diccionario por medio del atributo `attrs`:

```

1 |   >>> div_main = soup.div
2 |   >>> div_main['id']
3 |   'main'
4 |   >>> div_main.attrs
5 |   {'id': 'main', 'class': ['full-width']}
```

Como puedes apreciar, hay atributos que son simples, como el `id` y otros, como `class`, que son multivaluados.

NavigableString

Un objeto de este tipo representa a la cadena de texto que hay contenida en una etiqueta. Se accede por medio de la propiedad `string`:

```

1 |   >>> primer_parrago = soup.p
2 |   >>> texto = primer_parrago.string
3 |   >>> texto
4 |   'Este es el primer párrafo'
5 |   >>> type(texto)
6 |   <class 'bs4.element.NavigableString'>
```

Además, la clase `NavigableString` es clase padre de otras, entre las que se encuentran:

- `Comment` : Esta clase representa un comentario *HTML*



Navegar a través de los elementos de BeautifulSoup

En esta sección y las dos siguientes veremos cómo podemos navegar a través de los objetos del árbol de *Beautiful Soup*.

Lo importante en este punto es que entiendas que un objeto de tipo `Tag` puede contener bien otros objetos de tipo `Tag`, bien objetos de tipo `NavigableString`.

En la sección anterior hemos visto que **es posible acceder a los objetos del árbol utilizando los nombres de las etiquetas como atributos**. Esta es la forma más simple de navegar a través del árbol.

A continuación, te muestro un ejemplo para acceder a la etiqueta `meta` y mostrar su atributo `charset`:

```
1 |   >>> soup.meta['charset']
2 |   'UTF-8'
```

Además, usando los nombres de etiquetas se puede navegar hacia los elementos interiores de un objeto.

Si vuelves a examinar el documento *HTML* del ejemplo, verás que el bloque `div` que contiene los enlaces `Enlace 1` y `Enlace 2` está contenido en el bloque `div` con `id="innerDiv"` que a su vez está contenido en el bloque con `id="main"`. Por tanto, podemos acceder al mismo de la siguiente manera:

```
1 |   >>> soup.div.div.div
2 |   <div class="links">
3 |   <a href="https://pagina1.xyz/">Enlace 1</a>
4 |   <a href="https://pagina2.xyz/">Enlace 2</a>
5 |   </div>
```

Sin embargo, esta forma de acceder a los elementos puede que no sea la más adecuada en todas las situaciones. ¿Por qué? Porque **de este modo, BeautifulSoup solo devuelve el primer elemento que se corresponda con la etiqueta**.

Por tanto, así sería imposible acceder, por ejemplo, al bloque `div` con `id="footer"`.

Hijos y descendientes de un elemento en BeautifulSoup

Hay otras formas de acceder a los hijos de un objeto, además de a través de las etiquetas:

- **El atributo** `contents` : Devuelve una lista con todos los hijos de **primer nivel** de un objeto.
- **Usando el generador** `children` : Devuelve un iterador para recorrer los hijos de **primer nivel** de un objeto.



Veamos esto con un ejemplo. Imagina que quieres acceder a todos los hijos del bloque `div` con `id="innerDiv"`.

```

1  >>> inner_div = soup.div.div
2
3  # contents
4  >>> hijos = inner_div.contents
5  >>> type(hijos)
6  <class 'list'>
7  >>> for child in hijos:
8      ...      if child.name: # Ignoramos los saltos de línea
9          ...          print(f'{child.name}')
10 div
11 div
12
13
14 # children
15 >>> hijos = inner_div.children
16 >>> type(hijos)
17 <class 'list_iterator'>
18 >>> for child in hijos:
19     ...      if child.name: # Ignoramos los saltos de línea
20         ...          print(f'{child.name}')
21 div
22 div
23
24
25 # descendants
26 >>> hijos = inner_div.descendants
27 >>> for child in hijos:
28     ...      if child.name:
29         ...          print(f'{child.name}')
30 div
31 a
32 a
33 div
34 div
35 a
36 a

```

! IMPORTANTE: Los objetos de tipo texto no contienen los atributos que hacen referencia a los hijos, solo el atributo `string`.

Padres de un elemento

Además de a los hijos, es posible navegar hacia arriba en el árbol accediendo a los objetos padre de un elemento. Para ello, puedes usar las propiedades `parent` y `parents`:

- `parent` referencia al objeto padre de un elemento (`Tag` o `NavigableString`).
- `parents` es un generador que permite recorrer recursivamente todos los elementos padre de uno dado.



Buscar elementos en BeautifulSoup

Y hemos llegado, quizá, a una de las partes más importantes de esta librería: las búsquedas y filtros.

Beautiful Soup pone a nuestra disposición diferentes métodos para buscar elementos en el árbol. Sin embargo, dos de los principales son `find_all()` y `find()`.

Ambos métodos trabajan de forma similar. Básicamente, buscan entre los descendientes de un objeto de tipo `Tag` y recuperan todos aquellos que cumplan una serie de filtros.

Filtro por nombre de etiqueta

El filtro más básico consiste en pasar el nombre de la etiqueta a buscar como primer argumento de la función (parámetro `name`).

Imagina que quieras recuperar todos los enlaces (etiqueta `<a>`) que hay en el texto HTML del ejemplo. Se podría hacer del siguiente modo:

```

1  >>> enlaces = soup.find_all('a')
2  >>> for enlace in enlaces:
3      ...         print(enlace)
4
5  <a href="https://pagina1.xyz/">Enlace 1</a>
6  <a href="https://pagina2.xyz/">Enlace 2</a>
7  <a href="https://pagina3.xyz/">Enlace 3</a>
8  <a href="https://pagina4.xyz/">Enlace 4</a>
9  <a href="https://pagina5.xyz/">Enlace 5</a>

```

Filtro por atributos

Además del nombre de la etiqueta, puedes especificar parámetros con nombre. Si estos no coinciden con los nombres de los parámetros de la función, serán tratados como atributos de la etiqueta entre los que filtrar.

Por ejemplo, si quisieras encontrar el bloque `div` con `id="footer"`, podrías aplicar el siguiente filtro:

```

1  >>> footer = soup.find_all(id='footer')
2  >>> print(footer)
3  [<div id="footer">
4  <!-- El footer -->
5  <p>Este párrafo está en el footer</p>
6  <div class="links footer-links">
7  <a href="https://pagina5.xyz/">Enlace 5</a>
8  </div>
9  </div>]

```

Filtro por clases CSS

Otro filtro muy común que se suele aplicar es identificar un elemento a partir de sus clases de CSS.



```

1  >>> links_divs = soup.find_all('div', class_="links")
2  >>> print(links_divs)
3  [<div class="links">
4  <a href="https://pagina1.xyz/">Enlace 1</a>
5  <a href="https://pagina2.xyz/">Enlace 2</a>
6  </div>,
7  <div class="links">
8  <a href="https://pagina3.xyz/">Enlace 3</a>
9  <a href="https://pagina4.xyz/">Enlace 4</a>
10 </div>,
11 <div class="links footer-links">
12 <a href="https://pagina5.xyz/">Enlace 5</a>
13 </div>]

```

Si te has fijado, la búsqueda devuelve también el bloque contenido en el bloque del footer, dado que `links` es una de sus clases.

¿Cómo se puede buscar por la cadena exacta del atributo `class_`? Así:

```

1  >>> footer_links = soup.find_all(class_="links footer-links")
2  >>> print(footer_links)
3  [<div class="links footer-links">
4  <a href="https://pagina5.xyz/">Enlace 5</a>
5  </div>]

```

Sin embargo, las variaciones de esta cadena no funcionan:

```

1  >>> footer_links = soup.find_all(class_="footer-links links")
2  >>> print(footer_links)
3  []

```

No obstante, si se quiere buscar un elemento del árbol para el que coincidan dos o más clases CSS, se puede usar un selector CSS con el método `select()`:

```

1  >>> footer_links = soup.select('div.footer-links.links')
2  >>> print(footer_links)
3  [<div class="links footer-links">
4  <a href="https://pagina5.xyz/">Enlace 5</a>
5  </div>]

```

Limitando las búsquedas

Como puedes observar, `find_all()` devuelve una lista de objetos. Internamente, el método tiene que recorrer todos los hijos del objeto invocador en busca de aquellos que cumplan los diferentes filtros. En ocasiones, si el árbol (o documento) es muy grande, esta función puede tardar un poco. Se puede optimizar con los siguientes parámetros:

- `limit` : Indica el número de resultados a devolver por la función. Una vez encontrados, la función deja de buscar.
- `recursive` : Indica si la función debe buscar entre todos los elementos hijos de un objeto o solo aquellos de primer nivel. Por defecto es `True`.



único objeto si lo encuentra.

```
1 | >>> soup.find('title')
2 | <title>Página de prueba</title>
```

El código anterior es equivalente al siguiente:

```
1 | >>> soup.find_all('title', limit=1)
2 | [<title>Página de prueba</title>]
```

Conclusiones

Has llegado al final del tutorial de introducción a *web scraping con Python*. Como en otros campos de la programación, no hay una única forma de hacer las cosas.

No obstante, sí que me gustaría darte unos tips antes de terminar:

- Como te indicaba en la introducción, sé respetuoso con los términos legales de las páginas de las que vayas a extraer información. Si no tienes permiso, no lo hagas.
- Las páginas web cambian continuamente. Es posible que un programa o script que te funcione hoy, mañana ya no lo haga. Esto implica tener que rehacer parte del trabajo. El web scraping es así.
- Ciertos sitios trabajan con frameworks Javascript que renderizan el HTML dinámicamente. Esto quiere decir que, cuando haces una petición al navegador, el contenido que descargas no es un documento HTML, sino un script. En estos casos la cosa se complica pero esto da para otro post.

Si te ha gustado esto, mira lo que tengo a continuación 

Vivir de Python. Sin filtros. Cada semana, lo bueno y lo malo.





no), sin postureo y sin humo.

🎯 **OBJETIVO:** Facturar 10K € al mes.

👉 ¿Tú también quieras monetizar tus proyectos? ¿O simplemente te da curiosidad ver si esto funciona?

☑ **Apúntate y acompáñame en el experimento.**

❤️ He ayudado a miles de programadores/as como tú a ser mejores Pythonistas.

Ya hay +20.000 pythonistas en la newsletter

Nombre

(Requerido) Email

Quiero ser un Pythonista

* Al enviar el formulario confirmas que aceptas la POLÍTICA DE PRIVACIDAD (<https://j2logo.com/politica-de-privacidad/>)

¡Ey়! Esto también te puede interesar



[\(https://j2logo.com/python/listar-directorio-en-python/\)](https://j2logo.com/python/listar-directorio-en-python/)

Listar directorio en Python. Listar ficheros de un directorio
[\(https://j2logo.com/python/listar-directorio-en-python/\)](https://j2logo.com/python/listar-directorio-en-python/)



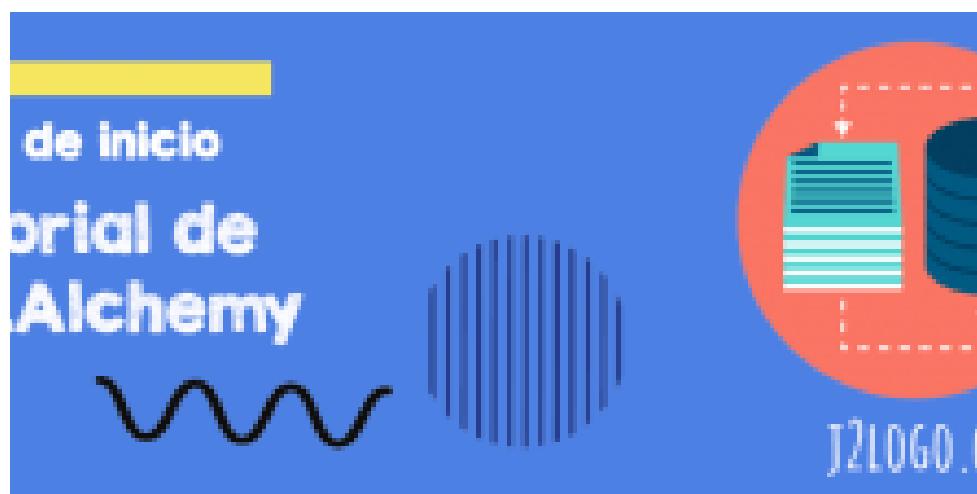
Librería requests Cómo hacer peticiones http en Python



j2logo.com

(https://j2logo.com/python/python-requests-peticiones-http/)

Python requests. La librería para hacer peticiones http en Python
(https://j2logo.com/python/python-requests-peticiones-http/)



(https://j2logo.com/python/sqlalchemy-tutorial-de-python-sqlalchemy-guia-de-inicio/)

SQLAlchemy. Tutorial de Python SQLAlchemy. Guía de inicio
(https://j2logo.com/python/sqlalchemy-tutorial-de-python-sqlalchemy-guia-de-inicio/)



(https://j2logo.com/python/crear-archivo-excel-en-python-con-openpyxl/)

Crear archivo excel en Python con openpyxl (https://j2logo.com/python/crear-archivo-excel-en-python-con-openpyxl/)



(<https://j2logo.com/flask/tutorial-como-crear-api-rest-python-con-flask/>)

Tutorial de cómo crear un API REST en Python con Flask
(<https://j2logo.com/flask/tutorial-como-crear-api-rest-python-con-flask/>)



(<https://j2logo.com/python/split-dividir-un-string-en-python-en-tokens/>)

Python split – Dividir string en Python en tokens (<https://j2logo.com/python/split-dividir-un-string-en-python-en-tokens/>)



👋 Soy Juan José Lozano Gómez, Ingeniero Informático y apasionado del Python práctico: automatizo cosas, scrapeo webs y creo bots que (a veces) generan ingresos.

Estoy intentando monetizar mis proyectos y en este blog teuento todo: ideas, código, errores y algún que otro acierto.

Más sobre mí (<https://j2logo.com/about/>)

(http
s://w
ww.i
nsta
@gra
m.co
m/j2l
ogo/
)
(http
s://w
ww.y
outu
be.c
om/j
2log
o)
(http
s://w
ww.li
nked
in.co
m/in/
j2log
o/)
(mail
to:ju
anjo
@j2l
ogo.
com
)

Mis lecturas recomendadas



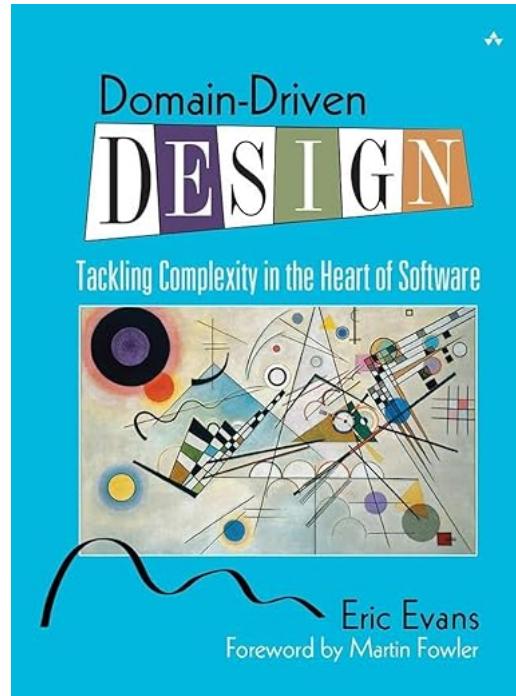
Architecture Patterns with Python

Enabling Test-Driven Development,
Domain-Driven Design, and Event-Driven
Microservices



Harry J.W. Percival
& Bob Gregory

(<https://www.amazon.es/dp/1492052205?tag=j2logo-21>)



(<https://www.amazon.es/dp/0321125215/?tag=j2logo-21>)

J2logo

Sobre mí(<https://j2logo.com/about/>)

Contacto(<https://j2logo.com/contacto/>)

Contratación

Blog



Tutoriales Python

Mis Proyectos

Bots y Automatización

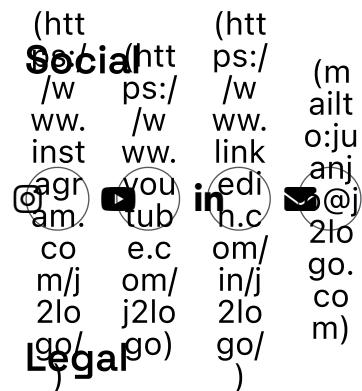
Scraping y Datos

Monetización

Formación

[Tutorial Python](https://j2logo.com/python/tutorial/)(<https://j2logo.com/python/tutorial/>)

[Tutorial Flask](https://j2logo.com/tutorial-flask-espanol/)(<https://j2logo.com/tutorial-flask-espanol/>)



[Aviso legal](https://j2logo.com/aviso-legal/)(<https://j2logo.com/aviso-legal/>)

[Política de privacidad](https://j2logo.com/politica-de-privacidad/)(<https://j2logo.com/politica-de-privacidad/>)

[Política de cookies](https://j2logo.com/politica-de-cookies/)(<https://j2logo.com/politica-de-cookies/>)

Un saludo a tod@s l@s pythonistas 🍀

👉 Nunca dejes que nadie te diga que no puedes hacer algo

Copyright © 2018-2025 Juan José Lozano Gómez