GOBIERNO DE ESPAÑA
MINISTERIO DE HACIENDA Y FUNCIÓN PÚBLICA
INAP
INSTITUTO NACIONAL DE ADMINISTRACIÓN PÚBLICA

Introducción a la programación en Python

Tema 13. Interactuar con webs

Autor: Antonio Villena Martín

Contenido

1. Introducción

HTML, siglas en inglés de *HiperText Markup Language* (lenguaje de marcado de hipertexto). Permite indicar la estructura de un documento mediante etiquetas. Es un lenguaje usado para la elaboración de páginas web. Al igual que XML, consiste en una serie de elementos con etiquetas que se estructuran de manera anidada.

Web Scraping es una técnica utilizada para la extracción de información dentro de una web, simulando la navegación humana. Con esta información se puede obtener el contenido de las etiquetas de un documento HTML o XML con el fin de recopilar información sobre el contenido de una página web, como los valores y contenidos de sus etiquetas, metadatos, enlaces dentro de etiquetas de tipo <a>, etc. En resumen, y haciendo una traducción literal, se trata de "escarbar una web ".

Con esta técnica se puede obtener mucha información haciendo uso de búsquedas en páginas web, que junto con expresiones regulares permite extraer los datos deseados.

Hay una gran variedad de soluciones de procesado XML y HTML disponibles para Python.

En este curso se va a usar una librería de Python llamada ***BeautifulSoup***, que es muy conocida y está bastante extendida.

`BeautifulSoup` se usa para extraer datos de archivos HTML y XML. Proporciona métodos para iterar, buscar y modificar el árbol de la estructura.

Para instalar la librería `BeautifulSoup` hay que instalar el paquete "**`bs4`**". En caso de tener que hacerla de manera manual, habría que ejecutar lo siguiente:

```
pip install bs4
```

Además, si se quiere trabajar con XML, se ha de tener instalado un ***parser*** de html o xml. Para instalarlo de manera manual, habría que ejecutar lo siguiente:

```
pip install lxml
```

Para usarlo, se importa con la siguiente sentencia:

```
from bs4 import BeautifulSoup
```

2. Pasos para hacer web scraping en Python con BeautifulSoup

Antes de entrar en materia, los pasos básicos que hay que seguir para extraer información de una web son:

1. **Identificar los elementos de la página web de los que se quiere extraer información.**

   Una web está estructurada en una serie de elementos ordenados según una jerarquía. El primer paso para extraer información útil es identificar el elemento o elementos que contienen la información que interesa.

   Lo más sencillo, es abrir la página en cualquier navegador e inspeccionar el elemento. Esto se hace mediante el botón derecho del ratón sobre el elemento en cuestión y pulsando sobre la opción «inspeccionar» o «inspeccionar elemento».

   Se tiene que retener toda la información disponible asociada al elemento, como la etiqueta, los atributos, las clases, los ids, etc. para posteriormente usarlos en `BeautifulSoup`.

2. **Descargar el contenido de la página web.**

   Se suele usar la librería **`request`** de Python. El contenido de la respuesta contiene toda la página en HTML, que será el que se pase a `BeautifulSoup` para generar el árbol de elementos y poder hacer consultas.

3. **Crear el objeto «sopa» instanciado de BeautifulSoup.**

   El contenido obtenido en el anterior paso será el que se use para crear la «sopa», es decir, el árbol de objetos Python que representan el documento HTML.

   Se crea un objeto `BeautifulSoup` al que se le pasa el texto en formato HTML y el identificador del parser a utilizar (`html.parser`, `lxml`, `xml`, etc.). Por ejemplo:

```
from bs4 import BeautifulSoup
import requests
req = requests.get('https://www.ign.es/web/ign/portal')
soup = BeautifulSoup(req.content, "lxml")
```

4. **Buscar los elementos en el objeto «sopa» y obtener la información deseada.**

   El último paso es hacer la búsqueda en el árbol y extraer los elementos que contienen la información y datos de interés. Existen varias alternativas para ello.

2.1. Tipos de objetos BeautifulSoup

El objeto `BeautifulSoup` que se cree, representa al árbol de objetos Python resultante de parsear el documento HTML de entrada, que será el punto de partida para navegar a través de los elementos del árbol, así como para realizar las búsquedas necesarias para el mismo.

2.1.1. Tag

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

```
from bs4 import BeautifulSoup
import requests
req = requests.get('https://www.ign.es/web/ign/portal')
soup = BeautifulSoup(req.content, "lxml")

soup.title


#imprime:

<title>Instituto Geográfico Nacional</title>
```

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

```
>>> div_main = soup.div
>>> div_main['id']
'wrapper'

>>> div_main.attrs
{'class': ['container-fluid'], 'id': 'wrapper'}
```

Como se puede ver, hay atributos simples, como `id` y otros multivaluados como `class`, que son almacenados en una lista.

Otra manera de acceder a los atributos como la clase y el id es mediante `tag['class']` y `tag['id']` respectivamente.

También se puede acceder al nombre de la etiqueta utilizando `tag.name`.

No solamente se puede extraer información de un HTML, sino que se puede agregar, quitar o modificar atributos de etiqueta dentro de un árbol `soup`.

2.1.2. NavigableString

Este objeto representa una cadena de texto que hay contenida en una etiqueta. Se accede por medio de la propiedad `string`.

```
>>> primer_span = soup.span
>>> primer_span
<span class="lang-show taglib-language-list-text" lang="es-ES">ES </span>
>>> texto = primer_span.string
>>> texto
ES

>>> type(texto)
<class 'bs4.element.NavigableString'>
```

La clase `NavigableString` es clase padre de otras, como, por ejemplo:

- `Comment`: Esta clase representa un comentario HTML
- `Stylesheet`: Esta clase representa un código CSS embebido
- `Script`: Esta clase representa un código Javascript embebido

Hay unos cuantos métodos útiles como `replace_with("string")` para reemplazar texto dentro de una etiqueta.

2.2. Navegar a través de los elementos de BeautifulSoup

Un objeto de tipo `Tag` puede contener tanto otros objetos de tipo `Tag`, como objetos de tipo `NavigableString`.

Se puede navegar a través del árbol DOM haciendo uso de los nombres de las etiquetas. Se puede obtener el primer elemento enlace de una página mediante `soup.a`, y todos los enlaces de la página a través de `soup.find_all('a')`.

Como ya se ha visto, es posible acceder a los objetos del árbol utilizando los nombres de las etiquetas como atributos. En el siguiente ejemplo, se muestra cómo se acceder a la etiqueta `meta` y se muestra el atributo `content`.

```
>>> soup.meta['content']
'max-age=3600'
```

Además, usando los nombres de etiquetas se puede navegar hacia los elementos interiores de un objeto. En el siguiente ejemplo con el que se está trabajando, se puede apreciar que el elemento `div` con `id="lupaBusqueda"`, a su vez está contenido en el `div` con `id="mobileLangContainer"`, que a su vez está contenido en el `div` con la `class="containe clearfix"`, que está contenida en `header` con `id="banner"`.

Se puede acceder al elemento `<div id="lupaBusqueda">` de la siguiente manera:

```
>>> soup.header.div.div.div

<div id="lupaBusqueda">
 <img alt="Búsqueda movil" src="/IGNCNIG/Imagenes/Utils/searchIconWhite.png"/>
</div>
```

Esta manera de acceder a los elementos puede que no sea la más eficiente en algunas situaciones, ya que solamente devuelve el primer elemento que se corresponde con la etiqueta.

2.3. Hijos y descendientes de un elemento en BeautifulSoup

Una forma alternativa de acceder a los elementos es a través de sus 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.
- Por medio del generador **`descendants`**: este atributo devuelve un iterador que permite recorrer todos los hijos de un objeto. No importa el nivel de anidamiento.

Ejemplo:

Se tiene el siguiente código HTML

```
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Página de prueba</title>
</head>
<body>
<div id="main" class="full-width">
    <h1>El título de la página</h1>
    <p>Este es el primer párrafo</p>
    <p>Este es el segundo párrafo</p>
    <div id="innerDiv">
        <div class="links">
            <a href="https://pagina1.xyz/">Enlace 1</a>
            <a href="https://pagina2.xyz/">Enlace 2</a>
        </div>
        <div class="right">
            <div class="links">
                <a href="https://pagina3.xyz/">Enlace 3</a>
                <a href="https://pagina4.xyz/">Enlace 4</a>
            </div>
        </div>
    </div>
    <div id="footer">
        <!-- El footer -->
        <p>Este párrafo está en el footer</p>
        <div class="links footer-links">
            <a href="https://pagina5.xyz/">Enlace 5</a>
        </div>
    </div>
</div>
</body>
</html>
```

Se quiere acceder a todos los hijos del bloque `div` con `id="innerDiv"`. Se va a hacer uso de las tres alternativas vistas:

```
>>> inner_div = soup.div.div

# contents
>>> hijos = inner_div.contents
```

```
>>> type(hijos)
<class 'list'>
>>> for child in hijos:
...     if child.name:  # Ignoramos los saltos de línea
...         print(f'{child.name}')
div
div


# children
>>> hijos = inner_div.children
>>> type(hijos)
<class 'list_iterator'>
>>> for child in hijos:
...     if child.name:  # Ignoramos los saltos de línea
...         print(f'{child.name}')
div
div


# descendants
>>> hijos = inner_div.descendants
>>> type(hijos)
>>> generator
>>> for child in hijos:
...     if child.name:
...         print(f'{child.name}')
div
a
a
div
div
a
a
```

2.4. Padres de un elemento

Además de a los hijos, se puede navegar hacia arriba en el árbol accediendo a los objetos padre de un elemento. Para ello se hace uso de 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.

2.5. Buscar elementos en BeautifulSoup

BeautifulSoup pone a disposición diferentes métodos para buscar elementos en el árbol. Los dos principales son `find_all()` y `find()`. Ambos métodos trabajan de manera similar, buscan entre los descendientes de un objeto de tipo `Tag` y recuperan todos aquellos que cumplan una serie de filtros.

2.5.1. 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`).

Si se quiere recuperar todos los enlaces (etiqueta `<a>`) que hay en el texto HTML de la web `https://www.ign.es/web/ign/portal`, se podría hacer del siguiente modo:

```
from bs4 import BeautifulSoup
import requests
req = requests.get('https://www.ign.es/web/ign/portal')
soup = BeautifulSoup(req.content, "lxml")

enlaces = soup.find_all('a')
for enlace in enlaces:
    print(enlace)
```

```
#Imprime:
<a class="taglib-language-list-text"
href="https://www.ign.es/web/ign/portal/inicio?p_p_id=82&amp;p_p_lifecycle=1&amp;
p_p_state=normal&amp;p_p_mode=view&amp;_82_struts_action=%2Flanguage%2Fview&amp;_
82_redirect=%2Fweb%2Fign%2Fportal&amp;_82_languageId=ca_ES" lang="ca-ES">CA</a>
<a class="taglib-language-list-text"
href="https://www.ign.es/web/ign/portal/inicio?p_p_id=82&amp;p_p_lifecycle=1&amp;
p_p_state=normal&amp;p_p_mode=view&amp;_82_struts_action=%2Flanguage%2Fview&amp;_
82_redirect=%2Fweb%2Fign%2Fportal&amp;_82_languageId=eu_ES" lang="eu-ES">EU</a>
……
```

2.5.2. Filtro por atributos

Además del nombre de la etiqueta, se puede 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.

Si se quisiera encontrar el bloque `div` con `id="heading"`, se podría aplicar el siguiente filtro siguiendo con el ejemplo del punto 2.5.1:

```
heading = soup.find_all(id='heading')
print(heading)

#Imprime algo similar a:
[<div class="pull-left" id="heading"> <div class="w100 mauto cb"> <div
id="logoFomento"> <!-- Logo temporal <style type="text/css">
                #heading #logoFomento img {width: 350px; max-width: 350px;}
                @media (max-width: 1300px)
        href="https://www.youtube.com/user/IGNSpain" style="color:lightgray"
target="_blank" title="Youtube"> </a> <a class="tooltip-trigger flaticon-social-
rss-circle-internet" data-placement="bottom" data-toggle="tooltip"
href="/web/ign/portal/social-rss" style="color:lightgray" target="_self"
title="rss"> </a></div> </div> </div> </div>]
```

2.5.3. Filtro por clases CSS

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

Si se quiere buscar en el documento todos los bloques `a` que tengan la clase `"taglib-language-list-text"`, se puede hacer de la siguiente manera siguiendo con el ejemplo del punto 2.5.1:

```
links_a = soup.find_all('a', class_="taglib-language-list-text")
print(links_a)

#Imprime:
[<a class="taglib-language-list-text"
href="https://www.ign.es/web/ign/portal/inicio?p_p_id=82&amp;p_p_lifecycle=1&amp;
p_p_state=normal&amp;p_p_mode=view&amp;_82_struts_action=%2Flanguage%2Fview&amp;_
82_redirect=%2Fweb%2Fign%2Fportal&amp;_82_languageId=ca_ES" lang="ca-ES">CA</a>,
…
<a class="taglib-language-list-text last"
href="https://www.ign.es/web/ign/portal/inicio?p_p_id=82&amp;p_p_lifecycle=1&amp;
p_p_state=normal&amp;p_p_mode=view&amp;_82_struts_action=%2Flanguage%2Fview&amp;_
82_redirect=%2Fweb%2Fign%2Fportal&amp;_82_languageId=en_GB" lang="en-GB">English
</a>]
```

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()`:

```
div_classes = soup.select('div.w100.cb.tal')
print(div_classes)

#Imprime:
[<div class="w100 cb tal" style="padding-top: 30px;"> <h2>Congresos</h2> <div
class="row-fluid"> <!--<div class="span6"><img alt="Congresos"
src="https://www.ign.es/resources/docs/IGNCnig/Imagenes_web/HOME/congresos.png"
/> <a class="geoportal" href="https://congreso.ign.es/web/10asambleahpgg/inicio"
```

```
target="_blank" title="10ª Asamblea Hispano Portuguesa de Geodesia y
Geofísica">10ª Asamblea Hispano Portuguesa de Geodesia y Geofísica</a></div> <div
class="span6"><img alt="Congresos"
src="https://www.ign.es/resources/docs/IGNCnig/Imagenes_web/HOME/congresos.svg"
/> <a class="geoportal" href="http://www.ibercarto.ign.es/" target="_blank"
title="IX Encuentro del Grupo de Trabajo de Cartotecas (IBERCARTO)">IX Encuentro
del Grupo de Trabajo de Cartotecas (IBERCARTO)</a></div>--> <div class="span6">
<div><img alt="Congresos" ,…..]
```

2.5.4. Limitando las búsquedas

Como se puede 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`.

Teniendo esto en cuenta, si se tiene la seguridad de que el documento HTML solo tiene un objeto específico, se puede usar la función `find()` ya que está más optimizada. Esta función se comporta de manera similar a `find_all()`, solo que devuelve un único objeto si lo encuentra.

```
>>> soup.find('title')
<title>Instituto Geográfico Nacional</title>
```

De manera similar, se obtiene una lista con todos los elementos, pero al tener un único elemento `title`, es equivalente al anterior:

```
>>> soup.find_all('title')
[<title>Instituto Geográfico Nacional</title>]
```

2.6. Conclusiones del web scraping

Las páginas web cambian continuamente. Es posible que un programa o script que funciona hoy, mañana ya no lo haga. Esto implica tener que rehacer parte del trabajo. Con lo que hay que estar revisando el código constantemente.

Ciertos sitios trabajan con frameworks Javascript que renderizan el HTML dinámicamente. Esto quiere decir que, cuando se hace una petición al navegador, el contenido que se descarga no es un documento HTML, sino un script. En estos casos la cosa se complica.

3. Ejercicios propuestos

3.1. Ejercicio 1

Escribe un programa de Python para crear un árbol de análisis BeautifulSoup en una cadena Unicode bien formateada, con una línea separada para cada etiqueta y cadena XML.

El código que se ha de formatear es el siguiente:

```
<Catalog>
   <Book id="bk101">
      <Author>Garghentini, Davide</Author>
      <Title>XML Developer's Guide</Title>
      <Genre>Computer</Genre>
      <Price>44.95</Price>
      <PublishDate>2000-10-01</PublishDate>
      <Description>An in-depth look at creating applications
      with XML.</Description>
   </Book>
   <Book id="bk102">
      <Author>Garcia, Debra</Author>
      <Title>Midnight Rain</Title>
      <Genre>Fantasy</Genre>
      <Price>5.95</Price>
      <PublishDate>2000-12-16</PublishDate>
      <Description>A former architect battles corporate zombies,
      an evil sorceress, and her own childhood to become queen
      of the world.</Description>
   </Book>
```

```
</Catalog>
```

3.2. Ejercicio 2

Crea un documento HTML que incluya cada uno de los siguientes usuarios. Cada usuario ha de ser un elemento independiente llamado `persona` en el documento y cada campo se corresponde con una etiqueta.

Todos los elementos `persona` han de estar incluidos en un elemento llamado `grupo_personas`.

nombre	movil	edad	genero
Manuel	555255222	21	Hombre
Juani	569654125	45	Mujer
Luis	896545214	33	Hombre

El documento final ha de quedar de la siguiente manera:

```
<?xml version="1.0" encoding="utf-8"?>
<grupo_personas>
        <persona>
                <nombre>Manuel</nombre>
                <movil>555255222</movil>
                <edad>21</edad>
                <genero>Hombre</genero>
        </persona>
</grupo_personas>
        <persona>
          ..................
```

La estructura del script podría ser la siguiente:

```
from bs4 import BeautifulSoup
```

```
soup = BeautifulSoup("<grupo_personas></grupo_personas>", "xml")


# Listas donde se encuentran los datos a introducir como elementos XML
encabezados = ["nombre", "movil", "edad", "genero"]
personas = [["Manuel", "555255222", "21", "Hombre"],
            ["Juani", "569654125", "45", "Mujer"],
            ["Luis", "896545214", "33", "Hombre"]]


# TODO: Recorre la lista de personas y va creando elementos dentro del
elemento "persona"



# TODO: Imprime el resultado por consola



# TODO: Exporta el resultado a un fichero xml
```

3.3. Ejercicio 3

En este ejercicio vamos a abrir el fichero XML generado en el ejercicio anterior en modo solo lectura. Una vez hayamos creado un objeto BeautifulSoup con el contenido del xml, vamos a buscar el elemento "`nombre`" que contenga la cadena de texto "`Juani`" y la vamos a sustituir por "`Almudena`".

Como pasos finales, vamos a imprimir por consola el resultado del XML y vamos a guardarlo en un fichero llamado "`salida2.xml`".

La estructura del script podría ser la siguiente:

```
from bs4 import BeautifulSoup


# TODO: Abre el fichero xml y lo almacena en una variable
```

```
# Crea el objeto BeautifulSoup
soup = BeautifulSoup(directorio, "xml")

# TODO: Recorre todos los elementos "nombre" para encontrar aquel que
contenga el string 'Manuel'



# TODO: Imprime por consola
print(soup.prettify())

# TODO: Exporta el resultado a un fichero xml
with open("salida2.xml", "w") as f:
    f.write(str(soup))
```

3.4. Ejercicio 4

Genera un objeto `BeautifulSoup` con el código indicado a continuación.

```
<libros>
  <libro>
    <titulo>The Cat in the Hat</titulo>
    <autor>Dr. Seuss</autor>
    <precio>6.99</precio>
  </libro>
  <libro>
    <titulo>Ender's Game</titulo>
    <autor>Orson Scott Card</autor>
    <precio>8.99</precio>
  </libro>
```

```
  <libro>
    <titulo>Prey</titulo>
    <autor>Michael Crichton</autor>
    <precio>9.35</precio>
  </libro>
</libros>
```

Posteriormente extrae e imprime por consola el título, el autor y el precio con el siguiente formato:

```
_________________
Titulo: The Cat in the Hat
Autor: Dr. Seuss
Precio: 6.99
_________________
```

4. Solución ejercicios propuestos

4.1. Solución ejercicio 1

```
from bs4 import BeautifulSoup


str1 = """
<Catalog>
   <Book id="bk101">
      <Author>Garghentini, Davide</Author>
      <Title>XML Developer's Guide</Title>
      <Genre>Computer</Genre>
      <Price>44.95</Price>
      <PublishDate>2000-10-01</PublishDate>
      <Description>An in-depth look at creating applications
      with XML.</Description>
   </Book>
   <Book id="bk102">
```

```
      <Author>Garcia, Debra</Author>
      <Title>Midnight Rain</Title>
      <Genre>Fantasy</Genre>
      <Price>5.95</Price>
      <PublishDate>2000-12-16</PublishDate>
      <Description>A former architect battles corporate zombies,
      an evil sorceress, and her own childhood to become queen
      of the world.</Description>
   </Book>
</Catalog>
"""

# Imprime cadena original
print("Cadena original:")
print(str1)

# Crea el objeto BeautifulSoup con el texto del código
soup = BeautifulSoup(str1, "xml")

# Imprime el código formateado como Unicode
print("\nCadena formateada Unicode:")
print(soup.prettify())
```

4.2. Solución ejercicio 2

```
from bs4 import BeautifulSoup

soup = BeautifulSoup("<grupo_personas></grupo_personas>", "xml")
```

```
# Listas donde se encuentran los datos a introducir como elementos XML

encabezados = ["nombre", "movil", "edad", "genero"]

personas = [["Manuel", "555255222", "21", "Hombre"],

            ["Juani", "569654125", "45", "Mujer"],

            ["Luis", "896545214", "33", "Hombre"]]


# Recorre la lista de personas y va creando elementos dentro del elemento
"persona"

for element in personas:

    persona = soup.new_tag("persona")

    soup.grupo_personas.append(persona)


    nombre = persona.append(soup.new_tag(encabezados[0]))

    persona.nombre.string = element[0]


    movil = persona.append(soup.new_tag(encabezados[1]))

    persona.movil.string = element[1]


    edad = persona.append(soup.new_tag(encabezados[2]))

    persona.edad.string = element[2]


    genero = persona.append(soup.new_tag(encabezados[3]))

    persona.genero.string = element[3]


# Imprime el resultado por consola

print(soup.prettify())


# Exporta el resultado a un fichero xml

with open("salida.xml", "w") as f:
```

```
    f.write(str(soup))
```

4.3. Solución ejercicio 3

```
from bs4 import BeautifulSoup


# Abre el fichero xml y lo almacena en una variable

with open("./salida.xml", "r") as file:

    directorio = file.read()


# Crea el objeto BeautifulSoup

soup = BeautifulSoup(directorio, "xml")


# Recorre todos los elementos "nombre" para encontrar aquel que contenga
el string 'Manuel'

for elem in soup.find_all("nombre"):

    if (elem.string == "Juani"):

        elem.string = "Almudena"


# Imprime por consola

print(soup.prettify())


# Exporta el resultado a un fichero xml

with open("salida2.xml", "w") as f:

    f.write(str(soup))
```

4.4. Solución ejercicio 4

```
from bs4 import BeautifulSoup


str1 = """
```

```
<libros>
  <libro>
    <titulo>The Cat in the Hat</titulo>
    <autor>Dr. Seuss</autor>
    <precio>6.99</precio>
  </libro>
  <libro>
    <titulo>Ender's Game</titulo>
    <autor>Orson Scott Card</autor>
    <precio>8.99</precio>
  </libro>
  <libro>
    <titulo>Prey</titulo>
    <autor>Michael Crichton</autor>
    <precio>9.35</precio>
  </libro>
</libros>
"""


# Crea el objeto BeautifulSoup
soup = BeautifulSoup(str1, 'xml')


# Busca todos los elementos 'libro'
libros = soup.find_all('libro')


# Bucle recorriendo la lista de libros e imprimiendo la información
for libro in libros:
    print('_____________________________________________')
    print('Título: ' + libro.find('titulo').get_text())
```

```
    print('Autor: ' + libro.find('autor').get_text())

    print('Precio: ' + libro.find('precio').get_text())

    print('_____________________________________\n')
```

5. Recursos

https://j2logo.com/python/web-scraping-con-python-guia-inicio-beautifulsoup/

https://code.tutsplus.com/es/tutorials/scraping-webpages-in-python-with-beautiful-soup-the-basics--cms-28211