```
# "pip install requests" pour pouvoir l'utiliser
import requests

response = requests.get('https://fr.wikipedia.org/plop')
print(response)   -------  <Response [404]>

print(response.status_code)    ----404

page="https://www.beerwulf.com/fr-fr/p/bieres/brasserie-de-sutter-brin-de-folie.33"
content = requests.get(page).text
print(content[:100])     ----

before_price = '<span class="price">'
idx = content.index(before_price)
price = content[idx+len(before_price):idx+100]  ----<span class="price">€2,29</span>\r\n\r\n</div>\r\n\r\n\r\n                    </div>\r\n
           <di'
price = price.split('<')[0]    ----?
```

Moralité

C'est un peu le bordel d'extraire des infos du document html en manipulant le document comme une bête str.

Here comes beautifulsoup:

Beautiful Soup 是一个可以从HTML或XML文件中提取数据的Python库.它能够通过你喜欢的转换器实现惯用的文档导航,查找,修改文档的方式.Beautiful Soup会帮你节省数小时甚至数天的工作时间.

```
from bs4 import BeautifulSoup
```

https://beautifulsoup.readthedocs.io/zh_CN/v4.4.0/

```
html = """
<html>
   <head>
       <style>
       h1 { font-size: 50px; }
       body { font-family: Verdana; }
       li { color: red; }
       ul ul li { color: green; }
       .highlighted { font-weight: bold; }
       .italic { font-style: italic; }
       .highlighted.italic { }
       </style>
   </head>
   <body>
       <h1>Mon titre</h1>
       <p class="highlighted">
           Some text with a<br>
           <a href="https://google.com">link to google</a>
           <img src="https://picsum.photos/200/300">
       </p>
       <p>Some list:</p>
       <ul>
           <li>some item</li>
           <li class="highlighted italic">some item</li>
           <li class="italic">some item</li>
           <ul>
               <li>some other item 1</li>
               <li>some other item 2</li>
           </ul>
           <li>some item</li>
       </ul>
```

```
    </body>
</html>
"""

from bs4 import BeautifulSoup
soup = BeautifulSoup(html)

titre = soup.find('h1')    ----- <h1>Mon titre</h1>
type(titre)    ------ bs4.element.Tag
titre.text    ---- 'Mon titre'
titre.name    ----'h1'

link = soup.find('a')
link.attrs    ----------- {'href': 'https://google.com'}

paragraph = soup.find('p')------<p class="highlighted">
                                Some text with a<br/>
                          <a href="https://google.com">link to google</a>
                          <img src="https://picsum.photos/200/300"/>
                          </p>
paragraph.find('img')

soup.find_all('li', class_="italic")
# La même chose qu'au dessus, mais à l'aide d'un sélecteur css:
soup.select('li.italic')
# Récupérer les li de 2e niveau (qui sont dans un ul lui-même dans un ul)
soup.find('ul').find('ul').find_all('li')
# Même chose avec un sélecteur css:
soup.select('ul ul li')    ---- return all li in ul ul
myli = soup.select('ul ul li')[0]    ----  return first li
myli.find_next_sibling()
myli.parent  ----------- <ul>
                                <li>some other item 1</li>
                                <li>some other item 2</li>
                                </ul>
myli.parent.contents  ----return in a array?
['\n', <li>some other item 1</li>, '\n', <li>some other item 2</li>, '\n']
li.parent.find_all()  # même chose que .contents mais renvoie uniquement
les Tag [<li>some other item 1</li>, <li>some other item 2</li>]
```

快速开始

下面的一段HTML代码将作为例子被多次用到.这是 *爱丽丝梦游仙境的* 的一段内容(以后内容中简称为 *爱丽丝* 的文档):

```
html_doc = """
<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title"><b>The Dormouse's story</b></p>

<p class="story">Once upon a time there were three little sisters; and their names
were
<a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>

<p class="story">...</p>
"""
```

使用BeautifulSoup解析这段代码,能够得到一个 `BeautifulSoup` 的对象,并能按照标准的缩进格式的结构输出:

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html_doc, 'html.parser')
```

```
print(soup.prettify())
# <html>
#  <head>
#   <title>
#    The Dormouse's story
#   </title>
#  </head>
#  <body>
#   <p class="title">
#    <b>
#     The Dormouse's story
#    </b>
#   </p>
#   <p class="story">
#    Once upon a time there were three little sisters; and their names were
#    <a class="sister" href="http://example.com/elsie" id="link1">
#     Elsie
#    </a>
#    ,
#    <a class="sister" href="http://example.com/lacie" id="link2">
#     Lacie
#    </a>
#    and
#    <a class="sister" href="http://example.com/tillie" id="link2">
#     Tillie
#    </a>
#    ; and they lived at the bottom of a well.
#   </p>
#   <p class="story">
#    ...
#   </p>
#  </body>
# </html>
```

几个简单的浏览结构化数据的方法:

```
soup.title
# <title>The Dormouse's story</title>

soup.title.name
# u'title'

soup.title.string
# u'The Dormouse's story'

soup.title.parent.name
# u'head'

soup.p
# <p class="title"><b>The Dormouse's story</b></p>

soup.p['class']
# u'title'

soup.a
# <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>

soup.find_all('a')
# [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
#  <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
#  <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]

soup.find(id="link3")
# <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>
```

从文档中找到所有<a>标签的链接:

```
for link in soup.find_all('a'):
   print(link.get('href'))
   # http://example.com/elsie
   # http://example.com/lacie
   # http://example.com/tillie
```

从文档中获取所有文字内容:

```
print(soup.get_text())
# The Dormouse's story
#
# The Dormouse's story
#
# Once upon a time there were three little sisters; and their names were
# Elsie,
# Lacie and
# Tillie;
# and they lived at the bottom of a well.
#
# ...
```

这是你想要的吗?别着急,还有更好用的

```
tag=soup.dt
tag.string
```

对象的种类

Beautiful Soup将复杂HTML文档转换成一个复杂的树形结构,每个节点都是Python对象,所有对象可以归纳为4种: `Tag` , `NavigableString` , `BeautifulSoup` , `Comment` .

Tag有很多方法和属性,在 遍历文档树 和 搜索文档树 中有详细解释.现在介绍一下tag中最重要的属性: name和attributes
一个tag可能有很多个属性. tag `<b class="boldest">` 有一个 “class” 的属性,值为 “boldest” . tag的属性的操作方法与字典相同:

```
tag['class']
# u'boldest'
```

也可以直接”点”取属性, 比如: `.attrs` :

```
tag.attrs
# {u'class': u'boldest'}
```

tag的名字

操作文档树最简单的方法就是告诉它你想获取的tag的name.如果想获取 <head> 标签,只要用 `soup.head` :

这是个获取tag的小窍门,可以在文档树的tag中多次调用这个方法.下面的代码可以获取<body>标签中的第一个<b>标签:

```
soup.body.b
# <b>The Dormouse's story</b>
```

通过点取属性的方式只能获得当前名字的第一个tag:如果想要得到所有的<a>标签,或是通过名字得到比一个tag更多的内容的时候,就需要用到 Searching the tree 中描述的方法,比如: find_all()

.contents 和 .children

tag的 `.contents` 属性可以将tag的子节点以列表的方式输出:

.descendants

.descendants

`.contents` 和 `.children` 属性仅包含tag的直接子节点.例如,<head>标签只有一个直接子节点<title>

.parent

通过 `.parent` 属性来获取某个元素的父节点.在例子“爱丽丝”的文档中,<head>标签是<title>标签的父节点:

.next_sibling 和 .previous_sibling

在文档树中,使用 `.next_sibling` 和 `.previous_sibling` 属性来查询兄弟节点:

.string

如果tag只有一个 `NavigableString` 类型子节点,那么这个tag可以使用 `.string` 得到子节点:

.strings 和 stripped_strings

如果tag中包含多个字符串 [2] ,可以使用 `.strings` 来循环获取:输出的字符串中可能包含了很多空格或空行,使用 `.stripped_strings` 可以去除多余空白内容:

.next_element 和 .previous_element

`.next_element` 属性指向解析过程中下一个被解析的对象(字符串或tag),结果可能与 `.next_sibling` 相同,但通常是不一样的.

BeautifulSoup

`BeautifulSoup` 对象表示的是一个文档的全部内容.大部分时候,可以把它当作 `Tag` 对象,它支持 遍历文档树 和 搜索文档树 中描述的大部分的方法.

因为 `BeautifulSoup` 对象并不是真正的HTML或XML的tag,所以它没有name和attribute属性.但有时查看它的 `.name` 属性是很方便的,所以 `BeautifulSoup` 对象包含了一个值为“[document]” 的特殊属性 `.name`

##

Inconvénients du scraping:

1) le contenu généré dynamiquement en javascript n'est pas présent initialement sur la page, ce qui fait qu'on peut avoir un contenu différent entre ce qu'on trouve dans l'inspecteur (sur chrome: F12 > Elements)

2) inconvénient du scraping: on est tributaire de l'architecture du HTML. Si les développeurs du site web changent le design, il y a de fortes chances que le programme beautifulsoup doive être réécrit.

Solution: **préférer une API**

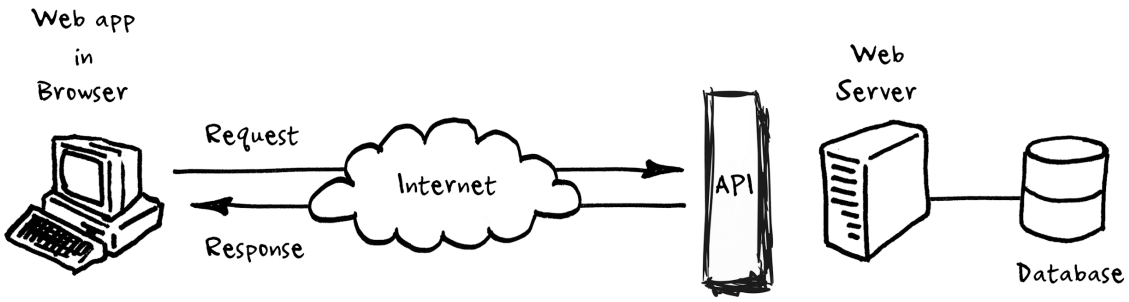
What is an "API" ?

Le terme "API" est très générique et peut désigner bien des choses, mais dans le jargon on l'utilise souvent pour désigner un service web qui renvoie non pas:

> des pages web au format HTML (destinées à être lues par un humain dans son navigateur)

mais:

> des données au format JSON (destinées à être traitées par un programme)



Exemple: deezer (site web) VS deezer API

Récupérer le nombre de fans d'un artiste:

```
1.# Site web
response = requests.get("https://www.deezer.com/fr/artist/939")
soup = BeautifulSoup(response.text)
nb_fans = int(soup.find('div', id='naboo_artist_social_small').span.text)

2.# API JSON
import json
response = requests.get("https://api.deezer.com/artist/939")
data = json.loads(response.text)
nb_fans = data['nb_fan']
```

3.Encore mieux: il y a un module sur pypi pour accéder à l'api de deezer encore + facilement:
https://pypi.org/project/deezer-python/

```
import deezer
c = deezer.Client()
nb_fans = c.get_artist(939).nb_fan
```

Basic Auth

{'Server': 'nginx/1.10.3', 'Date': 'Wed, 02 Oct 2019 21:50:28 GMT', 'Content-Type': 'text/html', 'Content-Length': '195', 'Connection': 'keep-alive', 'WWW-Authenticate': '**Basic realm**="Restricted"'}
Basic Auth (wikipedia) : il faut passer un header Authorization avec la valeur Basic XXX en remplaçant XXX par les credentials username:password encodés en **base64**:

```
res = requests.get('https://kim.fspot.org/private', auth=('admin', 'secret'))  ----- headers = {'Authorization': 'Basic YWRtaW46c2VjcmV0'}
```

Auth par token

https://swoopnow.com/token-based-authentication/
JSON Web Token（缩写 JWT）是目前最流行的跨域认证解决方案，
http://www.ruanyifeng.com/blog/2018/07/json_web_token-tutorial.html

传统的session认证

我们知道，http协议本身是一种无状态的协议，而这就意味着如果用户向我们的应用提供了用户名和密码来进行用户认证，那么下一次请求时，用户还要再一次进行用户认证才行，因为根据http协议，我们并不能知道是哪个用户发出的请求，所以为了让我们的应用能识别是哪个用户发出的请求，我们只能在服务器存储一份用户登录的信息，这份登录信息会在响应时传递给浏览器，告诉其保存为cookie,以便

下次请求时发送给我们的应用，这样我们的应用就能识别请求来自哪个用户了,这就是传统的基于session认证。
但是这种基于session的认证使应用本身很难得到扩展，随着不同客户端用户的增加，独立的服务器已无法承载更多的用户，而这时候基于session认证应用的问题就会暴露出来.

基于session认证所显露的问题

Session: 每个用户经过我们的应用认证之后，我们的应用都要在服务端做一次记录，以方便用户下次请求的鉴别，通常而言session都是保存在内存中，而随着认证用户的增多，服务端的开销会明显增大。
扩展性: 用户认证之后，服务端做认证记录，如果认证的记录被保存在内存中的话，这意味着用户下次请求还必须要请求在这台服务器上,这样才能拿到授权的资源，这样在分布式的应用上，相应的限制了负载均衡器的能力。这也意味着限制了应用的扩展能力。
CSRF: 因为是基于cookie来进行用户识别的, cookie如果被截获，用户就会很容易受到跨站请求伪造的攻击。

基于token的鉴权机制

基于token的鉴权机制类似于http协议也是无状态的，它不需要在服务端去保留用户的认证信息或者会话信息。这就意味着基于token认证机制的应用不需要去考虑用户在哪一台服务器登录了，这就为应用的扩展提供了便利。
流程上是这样的：
用户使用用户名密码来请求服务器
服务器进行验证用户的信息
服务器通过验证发送给用户一个token
客户端存储token，并在每次请求时附送上这个token值
服务端验证token值，并返回数据
这个token必须要在每次请求时传递给服务端，它应该保存在请求头里，另外，服务端要支持CORS(跨来源资源共享)策略，一般我们在服务端这么做就可以了Access-Control-Allow-Origin: *。

JWT长什么样?

JWT是由三段信息构成的，将这三段信息文本用.链接一起就构成了Jwt字符串。就像这样:第一部分我们称它为头部（header),第二部分我们称其为载荷（payload, 类似于飞机上承载的物品)，第三部分是签证（signature).

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

优点

- 因为json的通用性，所以JWT是可以进行跨语言支持的，像JAVA,JavaScript,NodeJS,PHP等很多语言都可以使用。
- 因为有了payload部分，所以JWT可以在自身存储一些其他业务逻辑所必要的非敏感信息。
- 便于传输，jwt的构成非常简单，字节占用很小，所以它是非常便于传输的。
- 它不需要在服务端保存会话信息, 所以它易于应用的扩展

安全相关

- 不应该在jwt的payload部分存放敏感信息，因为该部分是客户端可解密的部分。
- 保护好secret私钥，该私钥非常重要。
- 如果可以，请使用https协议

OAuth est un standard très répandu pour gérer l'authentification car il permet le workflow ci-dessus (autoriser une app à accéder à une portion d'un service où vous êtes inscrit, e.g facebook) dans le browser. Par

conséquent la plupart des API des gros services (twitter, facebook, google, etc.) ont une authentification basée sur OAuth.

Mais il est beaucoup moins simple que du Basic Auth ou bien qu'un simple token dans l'url: cf. cet exemple de requête http authentifiée sur l'api twitter: https://developer.twitter.com/en/docs/basics/authentication/guides/authorizing-a-request

En général on utilise donc des modules python qui abstraient les requêtes http en charge de l'authentification via oauth.

简单说，**OAuth** 就是一种授权机制。数据的所有者告诉系统，同意授权第三方应用进入系统，获取这些数据。系统从而产生一个短期的进入令牌（**token**），用来代替密码，供第三方应用使用。

请求意见稿（英语：Request for Comments，缩写：RFC）是由互联网工程任务组（IETF）发布的一系列备忘录。文件收集了有关互联网相关信息，以及UNIX和互联网社群的软件文件，以编号排定。当前RFC文件是由互联网协会（ISOC）赞助发行。

RFC始于1969年，由当时就读加州大学洛杉矶分校（UCLA）的斯蒂芬·克罗克（Stephen D. Crocker）用来记录有关ARPANET开发的非正式文档，他是第一份RFC文档的撰写者。最终演变为用来记录互联网规范、协议、过程等的标准文件。基本的互联网通信协议都有在RFC文件内详细说明。RFC文件还额外加入许多的论题在标准内，例如对于互联网新开发的协议及发展中所有的记录。