



UNIVERSITÀ
di **VERONA**

Linguaggio di Programmazione Python

16/05/2022

Giulio Mazzi

giulio.mazzi@univr.it



Web scraping

Un web scraper è un programma che scarica contenuto (in genere testi scritti in linguaggio naturale o in html) da una o più pagine web per estrarne dati.

In genere uno scraper si interfaccia con lo strato "esterno" di un sito web, quindi con le informazioni normalmente disponibili agli utenti.

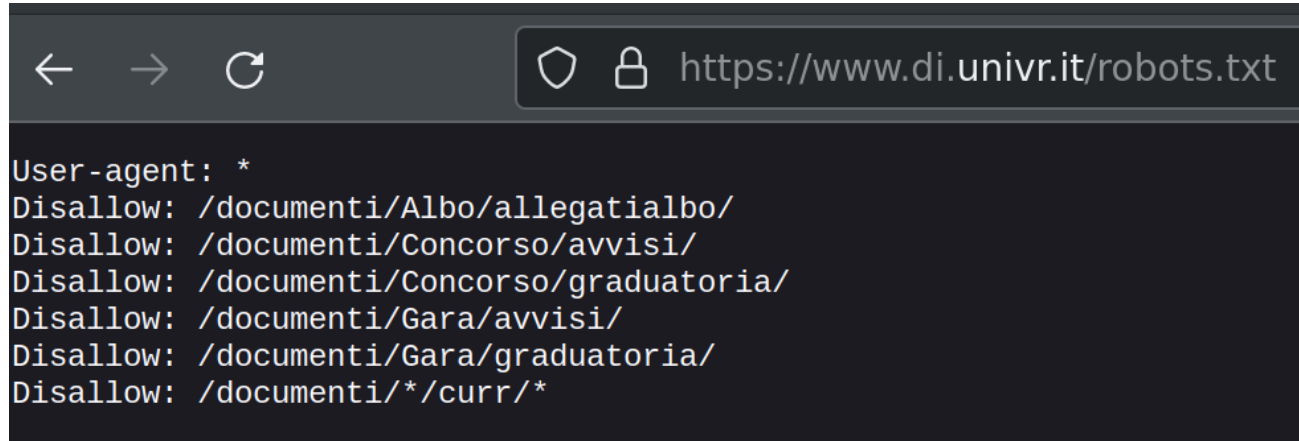
Il web scraping non è illegale, ma alcuni suoi utilizzi possono esserlo! Ad esempio, Il Garante Privacy ne vieta l'uso per la raccolta di dati personali senza consenso.

Web scraping - Limitazioni

Si stima che più del 50% del traffico sull'internet sia composto da bot.

Molti siti web forniscono un file, **robots.txt**, che presenta permessi e limitazioni. Rispettatelo sempre.

Cercate anche di limitare il numero di richieste al secondo (per evitare congestioni) e di avvisare sempre il gestore del sito su cui volete fare scraping.



```
← → ↻ https://www.di.univr.it/robots.txt
User-agent: *
Disallow: /documenti/Albo/allegatialbo/
Disallow: /documenti/Concorso/avvisi/
Disallow: /documenti/Concorso/graduatoria/
Disallow: /documenti/Gara/avvisi/
Disallow: /documenti/Gara/graduatoria/
Disallow: /documenti/*/curr/*
```

Esempio di robots.txt (<https://www.di.univr.it/robots.txt>)

HTML - Cenni

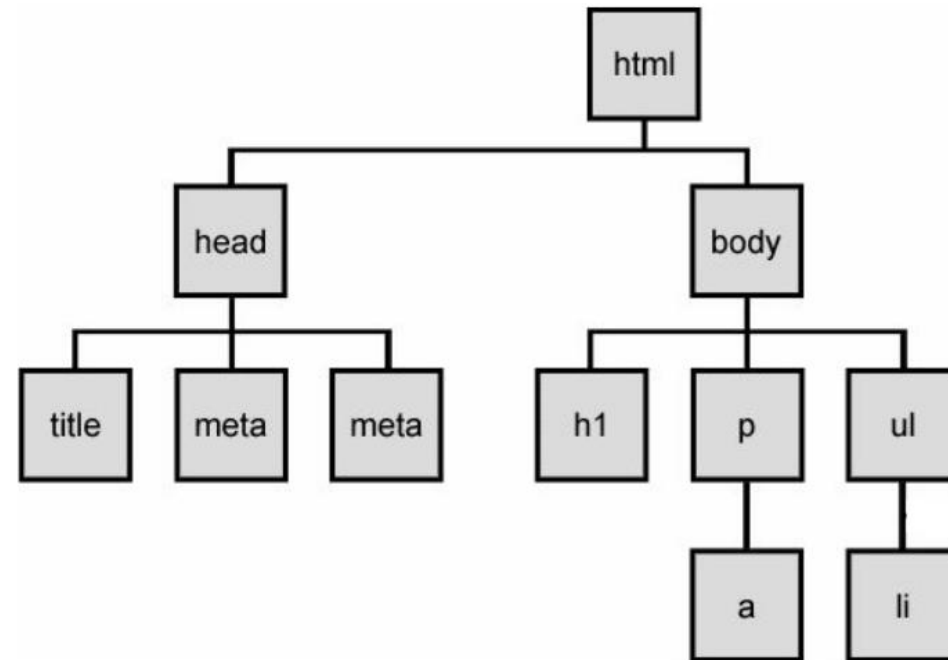
HTML è un linguaggio di markup. Serve a strutturare documenti ipertestuali, in genere pagine web. Potete pensare al codice come ad un albero, in cui ogni nodo è un tag. I tag sono racchiusi fra le parentesi angolari <>, e possono anche avere attributi.

```
<html>

<head>
  <title>My First Web Page</title>
</head>

<body>
  <h1>My First Web Page</h1>
  <p><b>Hello World Wide Web!</b></p>
  <p><i>Hello World Wide Web!</i></p>
  <p><u>Hello World Wide Web!</u></p>
  <p>This is my first web page.</p>
  <p>HTML tags can give <b><i>various</i></b>
  <u>looks and format</u> to the content of this w
</body>

</html>
```



Beautiful soup – Web scraping in python

Beautifulsoup è una libreria molto popolare per fare scraping su documenti HTML (o XML).

Supporta diversi parser e fornisce un interfaccia user friendly per navigare, modificare e fare ricerche nell'albero ottenuto dal parsing del documento.

Documentazione: <https://beautiful-soup-4.readthedocs.io/en/latest/>

NB: usate esplicitamente beautifulsoup 4, (`python3 -m pip install beautifulsoup4`).

Non è l'unico strumento di questo tipo! Per approfondire:

- [Scrappy](#) è una libreria più a basso livello (e potenzialmente più performante!) di beautifulsoup.
- [Selenium](#) è utile per gestire pagine che usano molto codice javascript.

Requests

Beautifulsoup non si occupa di "catturare" pagine web, le analizza e basta.

In python, possiamo usare la libreria requests per scaricare il codice che ci interessa.

```
import requests
```

```
req = requests.get("https://www.ilmeteo.it/meteo/Verona")
```

```
# stampo solo i primi 500 caratteri|  
print(req.text[:500])
```

```
<!-- OP view,9 --><!-- OP view,8 --><!-- OP view,10 --><!-- OP view,6 --><!-- OP view,11 --><!DOCTYPE html PUBLIC "-//  
W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml" lang="it" xml:lang="it">  
<head>  
  <title>Meteo VERONA &#9655; Previsioni fino a 15 giorni &raquo; ILMETEO.it</title>  
  <meta http-equiv="Content-Style-Type" content="text/css" />  
  <meta name="theme-color" content="#007ab9">
```

PS: controllate il robots.txt prima di scaricare!

Da HTML a soup

Potete creare una "zuppa" (il nome deriva da [tag soup](#)) partendo da una stringa (nel nostro caso, la pagina raccolta con requests).

I quattro parser supportati di default sono:

- "html.parser": default. Veloce, ottimo per documenti semplici. Ha più problemi a gestire file mal formattati.
- "html5lib": molto più lento, ma estremamente indulgente. Ottimo per gestire file complessi, a patto che la velocità non sia un problema
- "xml" solo per file XML
- "lxml": supporta XML e HTML, veloce e indulgente con gli errori

```
import bs4 as bs # beautifulsoup

# usa html.parser come parser
soup = bs.BeautifulSoup(req.text, "html.parser")
print(type(soup))
```

```
<class 'bs4.BeautifulSoup'>
```

```
# parser alternativo: html5lib
soup = bs.BeautifulSoup(req.text, "html5lib")
print(type(soup))
```

```
<class 'bs4.BeautifulSoup'>
```

get_text() e attributi

Possiamo accedere all'intero testo del documento con `get_text()`.

Spesso però è preferibile lavorare con sotto elementi del testo.

Beautifulsoup costruisce un attributo per ogni tag dell'albero. Possiamo navigare direttamente all'elemento che ci interessa!

```
soup.title
```

```
<title>Meteo VERONA ▷ Previsioni fino a 15 giorni » ILMETEO.it</title>
```

```
soup.title.name
```

```
'title'
```

```
soup.title.string
```

```
'Meteo VERONA ▷ Previsioni fino a 15 giorni » ILMETEO.it'
```

```
soup.title.parent.name
```

```
'head'
```


L'oggetto Tag

I nodi contenuti in soup sono oggetti di tipo Tag. Essi contengono:

- attributi di default (t.name, t.string, ecc...),
- Un genitore (t.parent)
- Uno o più figli (posso accedere a tutti i figli con t.children, oppure usando gli specifici tag)
- Il tag precedente (t.prev) e successivo (t.next)

```
print(soup.a)
```

```
<a class="tab" href="https://www.ilmeteo.it/" id="tab1" title="Home Page"><span class="color">Home</span></a>
```

```
print(type(soup.a))
```

```
<class 'bs4.element.Tag'>
```

```
print(soup.a.next)
```

```
<span class="color">Home</span>
```

```
print(soup.a.next.next.next.next.next)
```

```
<span class="color">Previsioni</span>
```

Attributi

Possiamo accedere agli attributi di un tag (ovvero, le informazioni contenute come sotto campi del tag) usando una notazione identica a quella dei dizionari python.

Per esempio, un link (tag <a>) avrà sempre un attributo "href" con il link vero e proprio.

```
# stampa il primo link, i suoi attributi sono class, href, id, title
print(soup.a)
```

```
<a class="tab" href="https://www.ilmeteo.it/" id="tab1" title="Home Page"><span class="color">Home</span></a>
```

```
# posso accedere agli attributi del mio link usando le parentesi []
print(soup.a["href"])
print(soup.a["class"])
```

```
https://www.ilmeteo.it/
['tab']
```

Ricerca - find e find_all

Lavorare direttamente con l'albero è comodo e flessibile, ma non sufficiente! Spesso vogliamo accedere ad un elemento specifico (o anche a più di uno) in una posizione a noi ignota.

Beautifulsoup offre i metodi find e find_all per facilitarci la vita. Il metodo find() trova la prima istanza dell'elemento cercato, mentre find_all le trova tutte (possiamo combinarlo ad un ciclo for).

I metodi accettano una stringa, che sarà il nome del tag (NB, non dell'attributo!). Se vogliamo lavorare con gli attributi, li dobbiamo passare come parametri opzionali.

```
# trova il prossimo tag <a>
soup.find('a')
```

```
<a class="tab" href="https://www.ilmeteo.it/" id="ta
```

```
# trova tutti i tag <a>
all_links = soup.find_all('a')
print(all_links[0:2])
```

```
[<a class="tab" href="https://www.ilmeteo.it/" id="ta
ss="tab tab-on" href="https://www.ilmeteo.it/portale/
Europa e tutto il mondo"><span class="color">Previsio
```

```
# ricerca di più elementi
hs = soup.find_all(['h1', 'h2', 'h3'])
print(hs[0:3])
```

```
[<h1>Meteo VERONA ▷ Previsioni fino a 15 giorni</h1>
>]
```

```
# cerca tutti gli elementi con un attributo 'href'
soup.find_all(href=True)
```

```
# possiamo fare ricerche anche sugli attributi
# cerca solo elementi in cui l'attributo id è 'tab2'
soup.find_all(id="tab2")
```