EPICODE

Python
GIORNO 8 - WebScraping

Agenda:

- ❏ Concetti base di HTML e CSS
- ❏ Raccolta automatizzata di informazioni da pagine web

Perché?

- Capita spesso che, durante la definizione di un problema o della sua soluzione, ci si scontri con lo scoglio della mancanza dei dati. Utilizzare tecniche di WebScraping può aiutarci a risolvere questo problema.

Alla fine di questa lezione saprete:

- ❏ Cos'è HTML e come funziona
- ❏ Cos'è CSS e come funziona
- ❏ Come utilizzare l'inspector del vostro browser per analizzare la struttura di una pagina web
- ❏ Effettuare richieste verso il web tramite Python e la libreria Requests
- ❏ Estrapolare informazioni da pagine web strutturate usando la libreria BeautifulSoup

HTML e CSS

Una moderna pagina web è una struttura composta da diversi livelli (layers), tra cui il documento HTML che definisce la struttura e il contenuto della pagina, e i fogli di stile CSS che gestiscono l'aspetto visivo e lo stile di presentazione, facilitando l'interazione tra contenuto e design.

HTML

Un documento HTML è la struttura di base di una pagina web, che definisce il contenuto e la disposizione degli elementi sulla pagina attraverso l'utilizzo di tag e attributi.

Ma cos'è HTML?
HTML, acronimo di HyperText Markup Language, è il linguaggio di markup utilizzato per la creazione e la strutturazione dei documenti su Internet. Consiste in una serie di elementi che permettono di definire la struttura logica e il contenuto di una pagina web, consentendo di organizzare testo, immagini, link e altri elementi multimediali in modo coerente e accessibile.

HTML serve dunque per la *formattazione*, e cioè per definire struttura e aspetto degli elementi della pagina web, e non per la *programmazione*.

HTML

Ecco un esempio di documento HTML, preso dalla prima pagina web della storia:

```
<HEADER>
<TITLE>The World Wide Web project</TITLE>
<NEXTID N="55">
</HEADER>
<BODY>
<H1>World Wide Web</H1>The WorldWideWeb (W3) is a wide-area<A
NAME=0 HREF="WhatIs.html">
hypermedia</A> information retrieval
initiative aiming to give universal
access to a large universe of documents.<P>
Everything there is online about
W3 is linked directly or indirectly
to this document, including an <A
NAME=24 HREF="Summary.html">executive
summary</A> of the project, <A
NAME=29 HREF="Administration/Mailing/Overview.html">Mailing lists</A>
, <A
NAME=30 HREF="Policy.html">Policy</A> , November's  <A
NAME=34 HREF="News/9211.html">W3  news</A> ,
<A
NAME=41 HREF="FAQ/List.html">Frequently Asked Questions</A> .
<DL>
<DT><A
NAME=44 HREF="../DataSources/Top.html">What's out there?</A>
<DD> Pointers to the
world's online information,<A
NAME=45 HREF="../DataSources/bySubject/Overview.html"> subjects</A>
, <A
NAME=z54 HREF="../DataSources/WWW/Servers.html">W3 servers</A>, etc.
<DT><A
NAME=46 HREF="Help.html">Help</A>
<DD> on the browser you are using
<DT><A
NAME=13 HREF="Status.html">Software Products</A>
<DD> A list of W3 project
components and their current state.
(e.g. <A
NAME=27 HREF="LineMode/Browser.html">Line Mode</A> ,X11 <A
NAME=35 HREF="Status.html#35">Viola</A> ,  <A
NAME=26 HREF="NeXT/WorldWideWeb.html">NeXTStep</A>
, <A
NAME=25 HREF="Daemon/Overview.html">Servers</A> , <A
NAME=51 HREF="Tools/Overview.html">Tools</A> ,<A
NAME=53 HREF="MailRobot/Overview.html"> Mail robot</A> ,<A
NAME=52 HREF="Status.html#57">
Library</A> )
<DT><A
NAME=47 HREF="Technical.html">Technical</A>
<DD> Details of protocols, formats,
program internals etc
<DT><A
NAME=40 HREF="Bibliography.html">Bibliography</A>
<DD> Paper documentation
on  W3 and references.
<DT><A
NAME=14 HREF="People.html">People</A>
```

HTML

Vediamo che sembra del codice scritto in un linguaggio di programmazione, e c'è del testo racchiusi tra elementi che presentano la struttura `<elemento>TESTO</elemento>`
Quello che è scritto tra i caratteri `<>` prende il nome di tag: ce ne sono vari e vanno a definire delle caratteristiche fondamentali e dei comportamenti legati al testo che inseriamo al loro interno.

Ad esempio: il tag `<h1>` (o `<H1>`) è il tag che definisce un titolo di primo livello (ovvero il titolo "più grande", a cui può seguire un sottotitolo, sottosottotitolo, eccetera). Se inseriamo del testo dopo un tag `<H1>`, e dopo il testo chiudiamo il tag usando `</H1>`, quando il browser caricherà il documento HTML per visualizzare la pagina web ciò che è scritto tra l'apertura e la chiusura del tag verrà visualizzato come un header di primo livello:

`<h1>Epicode</h1>` diventerà # Epicode

HTML

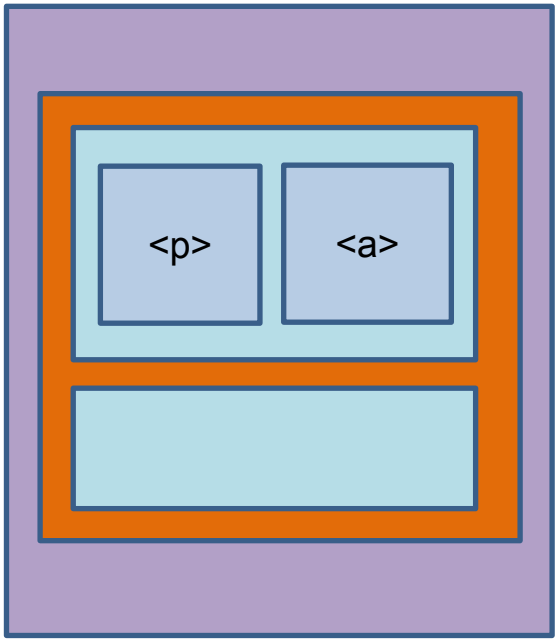
Alcuni dei tag più comuni sono:

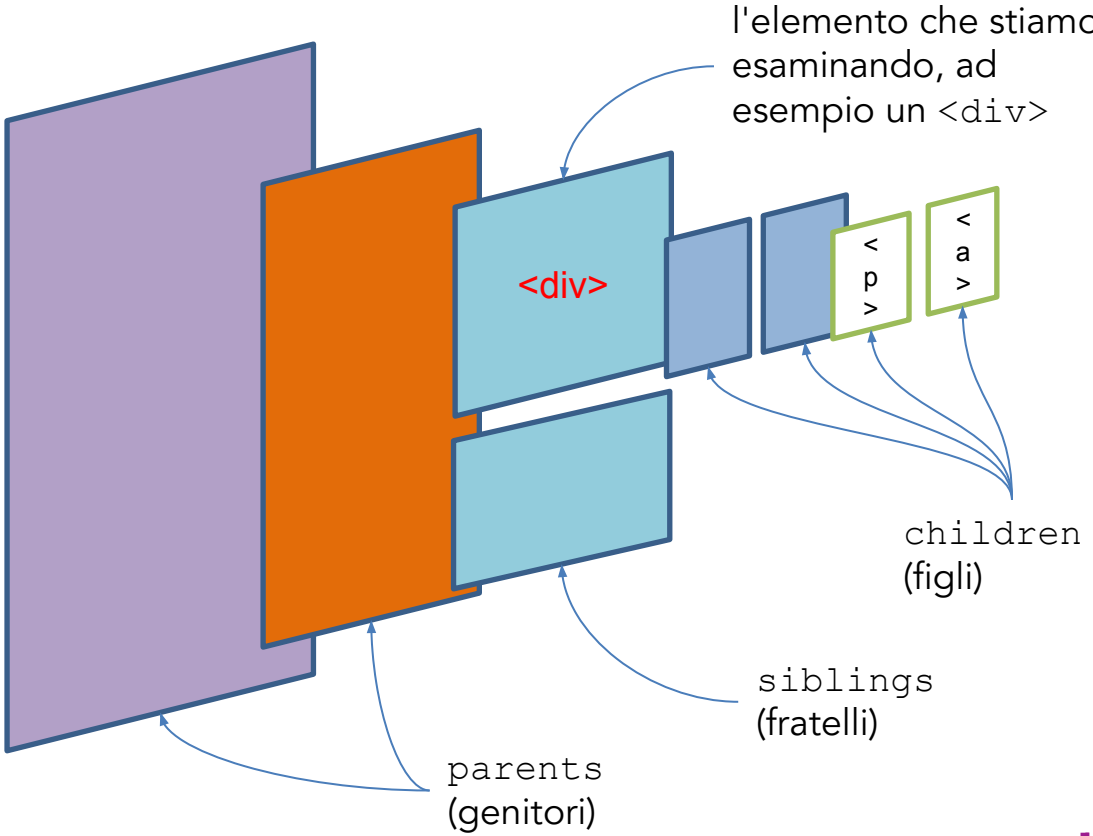
- `<h1>` - `<h6>`: Definiscono i titoli di diverse dimensioni.
- `<p>`: Definisce un paragrafo di testo.
- `<a>`: Definisce un collegamento ipertestuale.
- `<img>`: Incorpora un'immagine nell'HTML.
- `<ul>`: Definisce una lista non ordinata.
- `<ol>`: Definisce una lista ordinata.
- `<li>`: Definisce un elemento di lista.
- `<div>`: Definisce una sezione o una divisione nel documento.
- `<span>`: Definisce un'area di testo inline.
- `<table>`: Definisce una tabella.

Ai tag si possono aggiungere degli attributi, ad esempio `href=` per il tag `<a>`, in cui specifichiamo la destinazione del collegamento, o `src=` per il tag `<img>`, in cui specifichiamo l'origine dell'immagine da caricare.

HTML

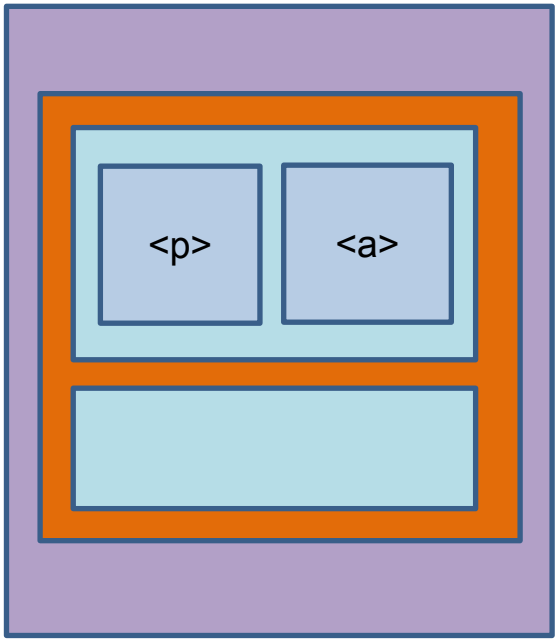
Gli elementi della pagina hanno una struttura a contenitore:

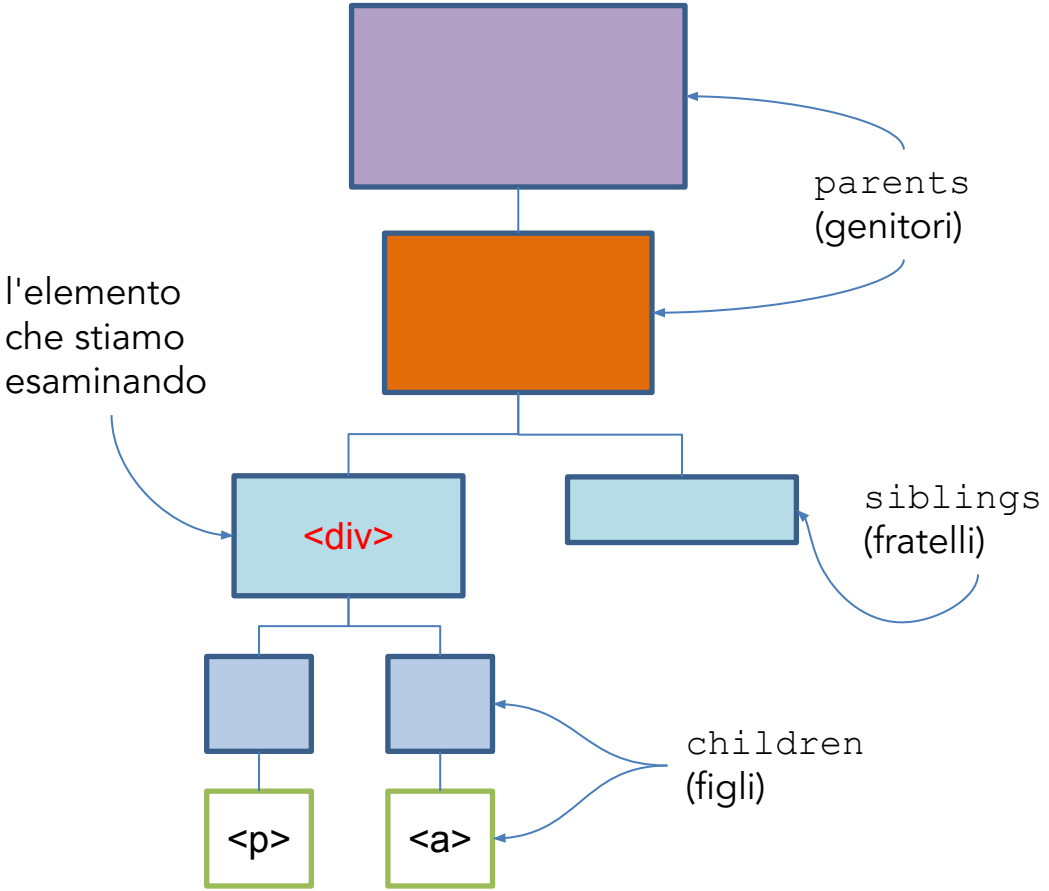




HTML

...che può essere pensata anche come una struttura ad albero:





HTML

Visitiamo la pagina web di cui abbiamo visto un esempio di codice un paio di slides più su:

https://info.cern.ch/hypertext/WWW/TheProject.html

Come vediamo contiene informazioni, testo, titoli, link ed è funzionale, ma manca completamente qualsiasi tipo di design grafico.

Per definire la struttura grafica di una pagina ci vengono in aiuto i fogli di stile CSS.

CSS

Un foglio di stile CSS (Cascading Style Sheets) è un insieme di regole e istruzioni utilizzate per definire l'aspetto visivo e lo stile di presentazione di una pagina web.

Queste regole possono includere definizioni di colori, dimensioni, font, margini, padding e altri attributi che determinano l'aspetto grafico degli elementi HTML presenti sulla pagina.

CSS

Un esempio di foglio di stile CSS:

```css
.indentedBox {
  padding:0px 7px 0px 7px
}
.box td,
.box th {
  padding-left:7px;
  padding-right:7px
}
.box h1,
.box h2 {
  padding:5px 5px;
  margin:0;
  top:-1px;
  background-color:#EBE8D5;
  color:#000000;
  font-size:1.2em
}
.box li {
  margin-left:17px
}
.box .content {
  margin:8px
}
div.content {
  position:relative
}
body>div.content {
  min-height:100%
}
.overflowHidden {
  width:100%;
  overflow:hidden
}
.textRight {
  display:block;
  text-align:right
}
.clearFix:after {
  content:'.';
  display:block;
  clear:both;
  visibility:hidden;
  height:0;
  line-height:0
}
.vcenteredLineOfBlocks {
  display:table
}
.vcenteredLineOfBlocks>* {
  display:table-cell;
  vertical-align:middle;
  padding-right:10px
}
```

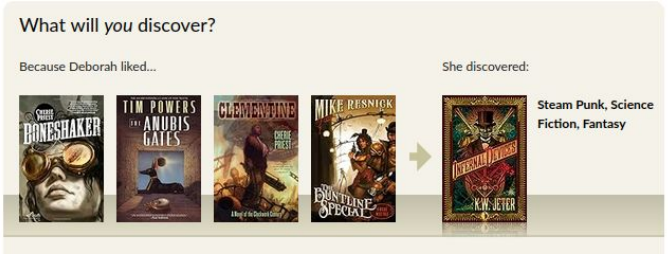
CSS

Ciò che nel foglio di stile è preceduto da " . " viene definito classe: permette di definire degli stili generali che verranno applicati a qualsiasi elemento HTML, facendo riferimento alla classe specifica dentro al tag.
Questa parte di codice HTML contiene dei riferimenti a delle classi CSS (ad esempio il primo tag, evidenziato in blu, ha l'attributo class che fa riferimento ad una classe CSS specifica)

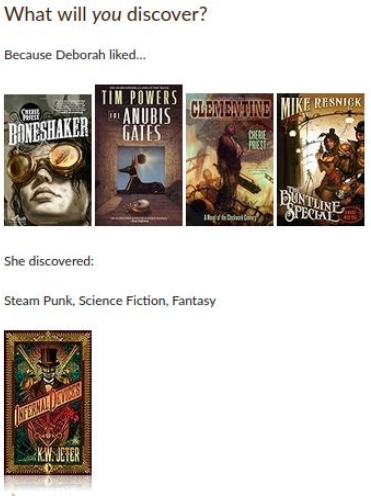
CSS

Vediamo cosa succede a quell'elemento se togliamo il riferimento alla classe CSS:

Con CSS:

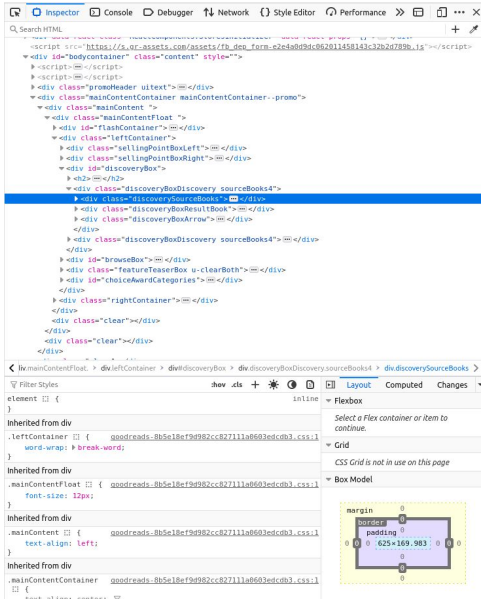


Senza CSS:



Inspector

Per accedere al codice HTML e CSS di qualsiasi pagina web basta che, una volta caricata, usiamo il tasto destro e selezioniamo “Inspect” (o “Ispeziona”, o “Analizza”, dipende dal browser e dalla lingua impostata). Otteniamo lo stesso risultato tramite delle shortcut, ad esempio F12, o CTRL+U (anche qui dipende dal browser). Si aprirà una sezione simile a questa:

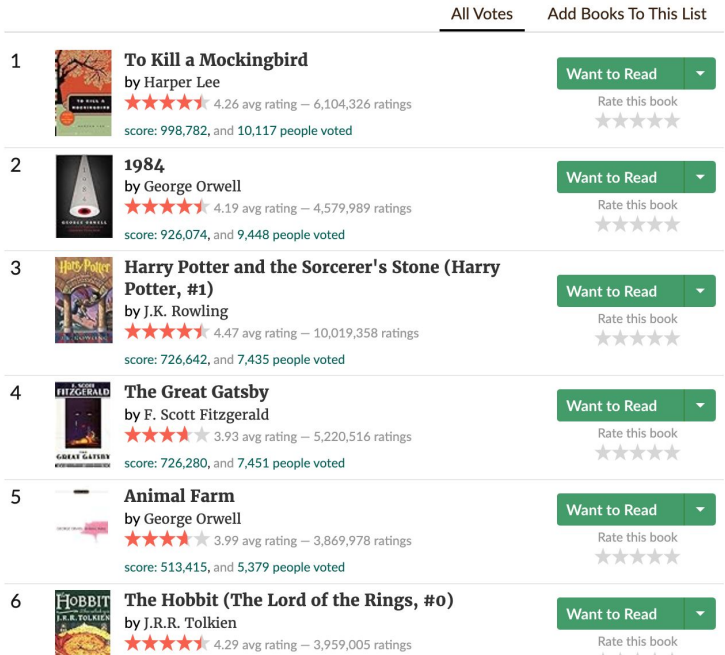


Inspector

Qui è presente tutta la struttura del sito web, tutto il codice HTML, CSS ed eventuali script se presenti. Per andare ad analizzare un singolo elemento, poi, possiamo usare l'icona in alto a sinistra ed andare a selezionarlo con il mouse: vedremo le informazioni sull'elemento in overlay e, se lo selezioniamo, il codice andrà automaticamente alla sezione che lo definisce.

Inspector

Poniamoci un obiettivo: vogliamo creare un `DataFrame` contenenti informazioni riguardo ai migliori libri del XX secolo come elencati sul sito di GoodReads, a questo link: https://www.goodreads.com/list/show/6.Best_Books_of_the_20th_Century



Per ognuno di questi libri ci interessano:

- ❑ Titolo
- ❑ Autore
- ❑ Punteggio medio
- ❑ Numero di review
- ❑ Data di pubblicazione

Inspector

- Titolo
- Autore
- Punteggio medio
- Numero di votazioni ricevute

Inspector

- ❏ Titolo
- ❏ Autore
- ❏ Punteggio medio
- ❏ Numero di votazioni ricevute

```
<a class="bookTitle" itemprop="url" href="/book/show/2657.To_Kill_a_Mockingbird">
  <span itemprop="name" role="heading" aria-level="4">To Kill a Mockingbird</span>
```

```
<a class="authorName" itemprop="url" href="https://www.goodreads.com/author/show/1825.Harper_Lee">
  <span itemprop="name">Harper Lee</span>
```

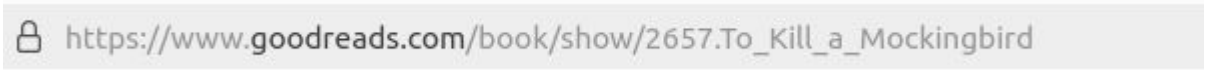
```
<span class="minirating">
  <span class="stars staticStars notranslate">
    <span class="staticStar p10" size="12x12"></span>
    <span class="staticStar p10" size="12x12"></span>
    <span class="staticStar p10" size="12x12"></span>
    <span class="staticStar p10" size="12x12"></span>
    <span class="staticStar p3" size="12x12"></span>
  </span>
  4.26 avg rating — 6,103,063 ratings
</span>
```

Inspector

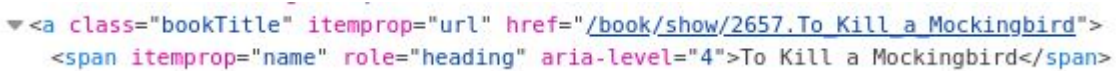
Per avere l'informazione sulla data di pubblicazione dobbiamo aprire il link cliccando sul titolo del libro. Facciamo caso però all'indirizzo della pagina "base", l'indirizzo che si apre cliccando sul titolo del libro e l'indirizzo presente nell'attributo `href=` del tag `<a>` relativo al titolo che abbiamo ottenuto grazie all'Inspector:

Pagina "base":

Pagina del libro:

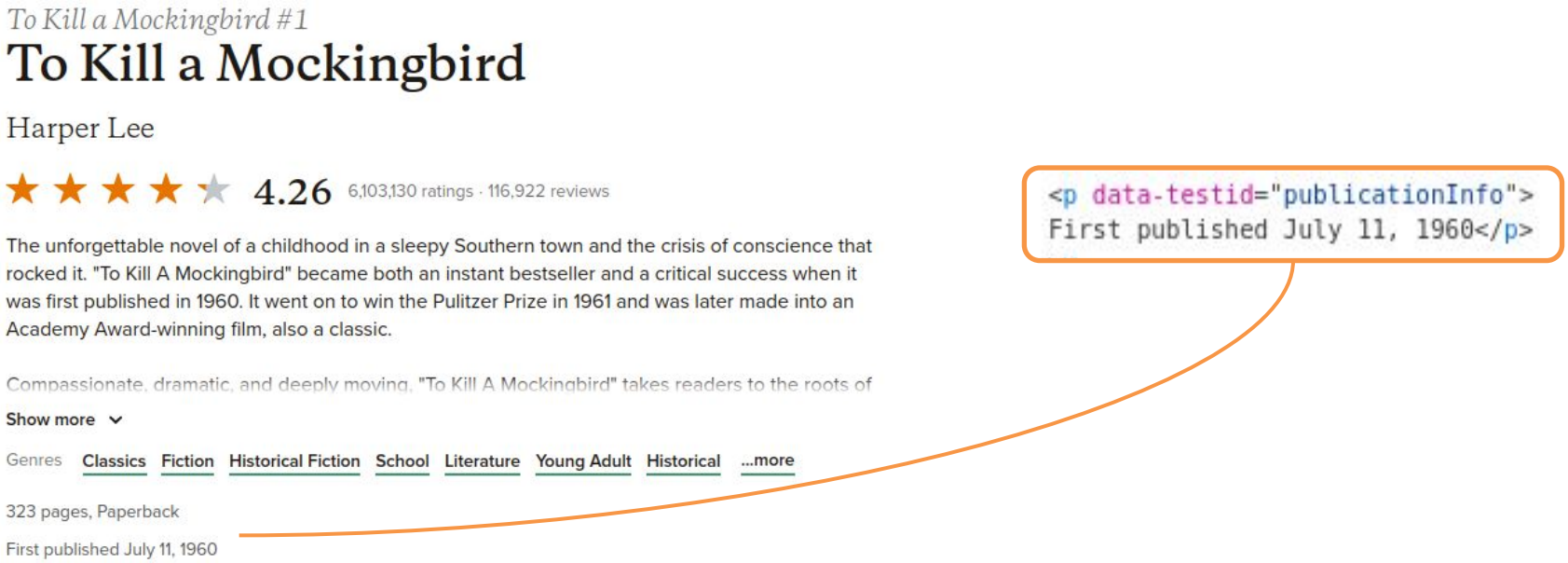


Tag `<a>` del libro nella pagina "base"



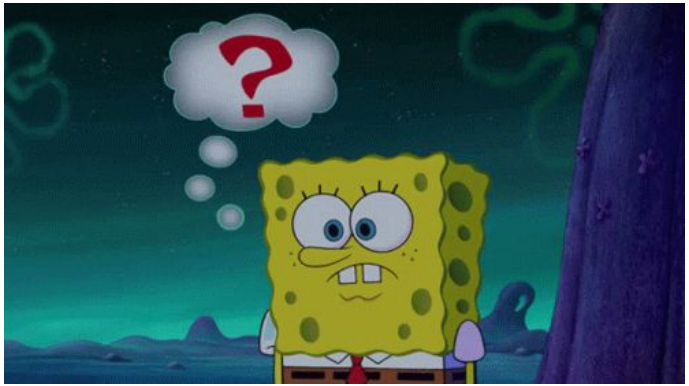
Inspector

Se apriamo la pagina, troviamo la data di pubblicazione



Inspector

QUIZZONE



Come possiamo verificare se tutti i libri seguono la stessa struttura di tag HTML e classi CSS?

Requests

Possiamo accedere al codice HTML di una pagina web anche tramite Python, per la precisione utilizzando la libreria requests, che permette di gestire richieste e chiamate verso internet. Non fa parte della Standard Library ma è una libreria molto diffusa: a seconda del tipo di installazione che abbiamo effettuato può darsi sia già inclusa.
Per mandare una richiesta ad una pagina internet basta passare l'URL alla funzione `.get()`

```
1 import requests
```

```
1 base_url = "https://www.goodreads.com"
2 list_details = "/list/show/6.Best_Books_of_the_20th_Century"
3 list_url = base_url+list_details
4 print(list_url)
```

```
https://www.goodreads.com/list/show/6.Best_Books_of_the_20th_Century
```

```
1 req = requests.get(list_url)
2 print(req)
```

```
<Response [200]>
```

Il `print` ci mostra il codice della risposta: `200` significa che la pagina ha risposto correttamente. Altri codici sono meno piacevoli, come il famoso `404`. Possiamo trovare una lista completa sul portale di sviluppo Mozilla: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Requests

Accedendo all'attributo `.text` abbiamo accesso a tutto il codice della pagina in un'unica stringa:

```
print(req.text)
```

```
<!-- Add query string params -->



<tr itemscope itemtype="http://schema.org/Book">
    <td valign="top" class="number">99</td>
  <td width="5%" valign="top">
    <div id="16299" class="u-anchorTarget"></div>
      <div class="js-tooltipTrigger tooltipTrigger" data-resource-id="16299" data-resource-type="Book">
        <a title="And Then There Were None" href="/book/show/16299.And_Then_There_Were_None">
          <img alt="And Then There Were None" class="bookCover" itemprop="image" src="https://i.gr-assets.com/images/S/compressed.photo.goodreads.com/books/1638425885i/16299._SY75_.jpg" />
</a>        </div>
  </td>
  <td width="100%" valign="top">
    <a class="bookTitle" itemprop="url" href="/book/show/16299.And_Then_There_Were_None">
      <span itemprop='name' role='heading' aria-level='4'>And Then There Were None</span>
</a>      <br/>
      <span class='by'>by</span>
<span itemprop='author' itemscope='' itemtype='http://schema.org/Person'>
<div class='authorName__container'>
<a class="authorName" itemprop="url" href="https://www.goodreads.com/author/show/123715.Agatha_Christie"><span itemprop="name">Agatha Christie</span></a>
</div>
</span>

      <br/>
      <div>
        <span class="greyText smallText uitext">
            <span class="minirating"><span class="stars staticStars notranslate"><span size="12x12" class="staticStar p10"></span><span size="12x12" class="staticStar p10"></span><span size="12x12" cl
        </span>
      </div>


            <div style="margin-top: 5px">
          <span class="smallText uitext">
            <a href="#" onclick="Lightbox.showBoxByID(&#39;score_explanation&#39;, 300); return false;">score: 59,446</a>,
            <span class="greyText">and</span>
            <a id="loading_link_606607" href="#" onclick="new Ajax.Request(&#39;/list/list_book/5069463&#39;, {asynchronous:true, evalScripts:true, onFailure:function(request){Element.hide(&#39;loading_
             

          </span>
```

BeautifulSoup

Dato che il codice HTML segue delle strutture regolari (i tag e i loro attributi), esistono delle librerie che permettono di navigare questi documenti in modo strutturato: una delle più comuni è BeautifulSoup. Anche BeautifulSoup non è inclusa nella Standard Library, quindi va installata a parte tramite uno dei metodi che conosciamo. Una volta installata, possiamo importarla e creare un oggetto di tipo `BeautifulSoup`.

```
1 from bs4 import BeautifulSoup
2 soup = BeautifulSoup(req.text)
3 print(soup)
```

```
-->
<div data-react-class="ReactComponents.LoginInterstitial" data-react-props='{"allowFacebookSignIn":true,"allowAmazonSignIn":true,"overrideSignedOutPageCount":false,"path":{"signInUrl"
<div id="overlay" onclick="Lightbox.hideBox()" style="display:none"></div>
<div id="box" style="display:none">
<div class="xBackground js-closeModalIcon" id="close" onclick="Lightbox.hideBox()" title="Close this window"></div>
<div id="boxContents"></div>
<div id="boxContentsLeftovers" style="display:none"></div>
<div class="clear"></div>
</div>
<div id="fbSigninNotification" style="display:none;">
<p>Welcome back. Just a moment while we sign you in to your Goodreads account.</p>
<img alt="Login animation" src="https://s.gr-assets.com/assets/facebook/login_animation-085464711e6c1ed5ba287a2f40ba3343.gif"/>
</div>
<script>
  //<![CDATA[
    qcdata = {} || qcdata;
      (function(){
        var elem = document.createElement('script');
        elem.src = (document.location.protocol == "https:" ? "https://secure" : "http://pixel") + ".quantserve.com/aquant.js?a=p-0dUe_kJAjvkoY";
        elem.async = true;
        elem.type = "text/javascript";
        var scpt = document.getElementsByTagName('script')[0];
        scpt.parentNode.insertBefore(elem,scpt);
      }());
    var qcdata = {qacct: 'p-0dUe_kJAjvkoY'};
  //]]>
</script>
<noscript>
<img alt="Quantcast" border="0" height="1" src="//pixel.quantserve.com/pixel/p-0dUe_kJAjvkoY.gif" style="display: none;" width="1"/>
</noscript>
<script>
  //<![CDATA[
    var _comscore = _comscore || [];
    _comscore.push({ c1: "2", c2: "6035830", c3: "", c4: "", c5: "", c6: "", c15: ""});
    (function() {
    var s = document.createElement("script"), el = document.getElementsByTagName("script")[0]; s.async = true;
    s.src = (document.location.protocol == "https:" ? "https://sb" : "http://b") + ".scorecardresearch.com/beacon.js";
    el.parentNode.insertBefore(s, el);
    })();
  //]]>
```

BeautifulSoup

Contiene lo stesso codice, come vediamo dal risultato di `print()`, ma c'è una differenza fondamentale: il datatype. Se l'attributo `.text` di requests è di tipo stringa, il nostro nuovo oggetto è un tipo particolare:

```
1 print(type(req.text))
2 print(type(soup))
```

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

Un oggetto di tipo `bs4.BeautifulSoup` ha i propri metodi, tra questi troviamo il metodo `.find_all()`, a cui possiamo passare un tag e una classe: ritornerà una lista con tutti gli elementi HTML che condividono queste caratteristiche.

BeautifulSoup

Ad esempio, avevamo visto grazie all'Inspector che il titolo del libro era contenuto in un elemento di tipo `<a>` la cui classe CSS di riferimento era `"bookTitle"`:

```
<a class="bookTitle" itemprop="url" href="/book/show/2657.To_Kill_a_Mockingbird">
  <span itemprop="name" role="heading" aria-level="4">To Kill a Mockingbird</span>
```

Se noi utilizziamo questi riferimenti come argomenti del metodo `.find_all()`, otterremo una lista con tutti i tag contenenti i titoli dei libri:

```
1 title = soup.find_all("a", class_="bookTitle")
2 print(title[0:5])
```

```
[<a class="bookTitle" href="/book/show/2657.To_Kill_a_Mockingbird" itemprop="url">
<span aria-level="4" itemprop="name" role="heading">To Kill a Mockingbird</span>
</a>, <a class="bookTitle" href="/book/show/61439040-1984" itemprop="url">
<span aria-level="4" itemprop="name" role="heading">1984</span>
</a>, <a class="bookTitle" href="/book/show/3.Harry_Potter_and_the_Sorcerer_s_Stone" itemprop="url">
<span aria-level="4" itemprop="name" role="heading">Harry Potter and the Sorcerer's Stone (Harry Potter, #1)</span>
</a>, <a class="bookTitle" href="/book/show/4671.The_Great_Gatsby" itemprop="url">
<span aria-level="4" itemprop="name" role="heading">The Great Gatsby</span>
</a>, <a class="bookTitle" href="/book/show/170448.Animal_Farm" itemprop="url">
<span aria-level="4" itemprop="name" role="heading">Animal Farm</span>
</a>]
```

BeautifulSoup

Il tag contiene molte cose: in questo momento noi siamo interessati al testo, a cui accediamo attraverso l'attributo `.text`

```
1 title[0].text
```

```
'\nTo Kill a Mockingbird\n'
```

Possiamo dunque fare un loop sulla lista che otteniamo da `.find_all()` per estrarre tutti i titoli.

Molto spesso i dati presi tramite WebScraping richiedono una pulizia abbastanza approfondita: possiamo cominciare già ad applicarla qui, rimuovendo ad esempio newlines o whitespaces:

```
1 titles = []
2 for title in soup.find_all("a", class_="bookTitle"):
3   titles.append(title.text.strip())
4 print(titles)
```

```
['To Kill a Mockingbird', '1984', "Harry Potter and the Sorcerer's Stone (Harry Potter, #1)", 'The Great Gatsby', 'Animal Farm', 'The Hobbit (The Lord of the Rings, #0)'
```

BeautifulSoup

Applichiamo lo stesso concetto per estrarre i nomi degli autori:

```
<a class="authorName" itemprop="url" href="https://www.goodreads.com/author/show/1825.Harper_Lee">
  <span itemprop="name">Harper Lee</span>
```

```
1 authors = []
2 for author in soup.find_all("a", class_="authorName"):
3   authors.append(author.text)
4 print(authors)
```

```
['Harper Lee', 'George Orwell', 'J.K. Rowling', 'F. Scott Fitzgerald', 'George Orwell', 'J.R.R. Tolkien', 'Antoine de Saint-
```

BeautifulSoup

Facciamo la stessa cosa anche per il rating:

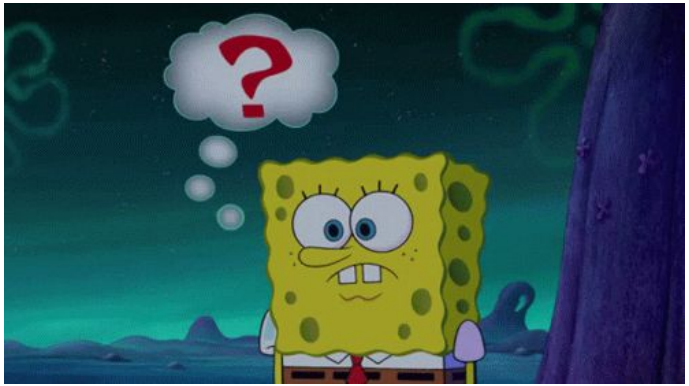
```
▼<span class="minirating">
  ▼<span class="stars staticStars notranslate">
      <span class="staticStar p10" size="12x12"></span>
      <span class="staticStar p10" size="12x12"></span>
      <span class="staticStar p10" size="12x12"></span>
      <span class="staticStar p10" size="12x12"></span>
      <span class="staticStar p3" size="12x12"></span>
    </span>
    4.26 avg rating — 6,103,063 ratings
  </span>
```

```
1 ratings = []
2 for rating in soup.find_all("span", class_="minirating"):
3   ratings.append(rating.text)
4 print(ratings)
```

```
[' 4.26 avg rating — 6,103,193 ratings', ' 4.19 avg rating — 4,579,049 ratings', ' 4.47 avg rating — 10,017,460 ratings', ' 3.
```

List comprehension

QUIZZONE



Come posso riscrivere quel loop in modo da estrarre solo la votazione media? E solo il numero di voti espressi?

BeautifulSoup

Per la data di pubblicazione dobbiamo accedere alla pagina specifica di ogni libro. Abbiamo però il link nel tag `<a>` del titolo del libro, all'interno dell'attributo `href=`:

```
<a class="bookTitle" itemprop="url" href="/book/show/2657.To_Kill_a_Mockingbird">
  <span itemprop="name" role="heading" aria-level="4">To Kill a Mockingbird</span>
```

utilizzando il metodo `.get()`, invece dell'attributo `.text`, possiamo accedere a questa informazione e, combinandola con l'indirizzo "base" della pagina, trovare l'URL di ogni sito:

```
1 links = [base_url+link.get("href") for link in soup.find_all("a", class_="bookTitle")]
2 print(links)
```

```
['https://www.goodreads.com/book/show/2657.To_Kill_a_Mockingbird', 'https://www.goodreads.com/book/show/61439040-1984', 'https://www.go
```

BeautifulSoup

Ora mandiamo una richiesta, tramite requests, al nuovo URL, creiamo un oggetto `BeautifulSoup` e cerchiamo il dato che ci serve:

```
<p data-testid="publicationInfo">
First published July 11, 1960</p>
```

```
1 new_req = requests.get(links[0])
2 new_soup = BeautifulSoup(new_req.text)
3 new_soup.find("p", attrs={"data-testid":"publicationInfo"}).text
```

```
'First published July 11, 1960'
```

BeautifulSoup

Possiamo creare un dizionario in cui mappiamo i titoli dei libri al link della loro pagina, in modo da avere accesso ad ogni data di pubblicazione:

```
title_links = dict()

for k in range(len(titles)):
      title = titles[k]
      link = links[k]
      titles_links[ title ] = link

print(title_links)

>>> {'To Kill a Mockingbird': 'https://www.goodreads.com/book/show/2657.To_Kill_a_Mockingbird',
'1984': 'https://www.goodreads.com/book/show/61439040-1984', …}
```

BeautifulSoup

Unendo tutto ciò che abbiamo costruito fino a qui in un unico loop possiamo gestire lo scraping completo di tutte le informazioni:

```
all_books = list()
num_books = len(titles)

for k in range(num_books):
    print("Scraping book no.", k, "of", num_books)
    title = titles[k]
    author = authors[k]
    rating = ratings[k]
    book_page_link = links[k]
    req = requests.get(book_page_link)
    soup = BeautifulSoup(req.text)
    published_date = soup.find("p", attrs={"data-testid": "publicationInfo"})
    if published_date is not None:  # talvolta non è presente
        published_date = published_date.text
    else:
        published_date == ""
    book = {"title": title,
            "author": author,
            "rating": rating,
            "published_date": published_date}
    all_books.append(book)
```

BeautifulSoup

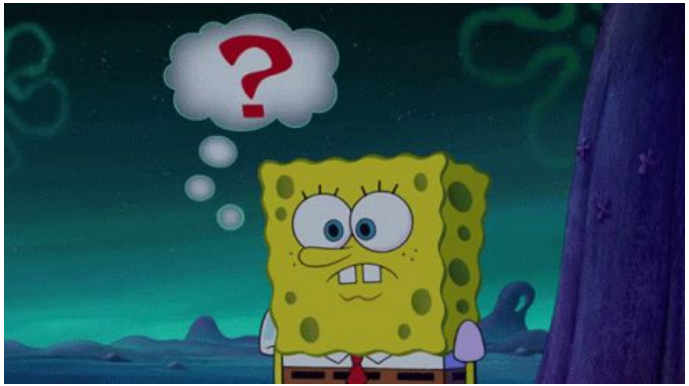
Chiudiamo il tutto caricando il risultato della lista in un DataFrame

```
1 import pandas as pd
2
3 df = pd.DataFrame(all_books)
4 df
```

	title	author	ratings_avg	ratings_count	published_date
0	To Kill a Mockingbird	Harper Lee	4.26 avg rating	6,089,604 ratings	First published July 11, 1960
1	1984	George Orwell	4.19 avg rating	4,566,201 ratings	First published June 8, 1949
2	Harry Potter and the Sorcerer's Stone (Harry P...	J.K. Rowling	4.47 avg rating	9,993,492 ratings	First published June 26, 1997
3	The Great Gatsby	F. Scott Fitzgerald	3.93 avg rating	5,206,480 ratings	First published April 10, 1925
4	Animal Farm	George Orwell	3.99 avg rating	3,859,161 ratings	First published August 17, 1945
...	...	...	...	...	...
95	Cat's Cradle	Kurt Vonnegut Jr.	4.16 avg rating	401,415 ratings	First published January 1, 1963
96	Blindness	José Saramago	4.17 avg rating	287,835 ratings	First published January 1, 1995
97	A Portrait of the Artist as a Young Man	James Joyce	3.64 avg rating	151,329 ratings	First published December 29, 1916
98	And Then There Were None	Agatha Christie	4.28 avg rating	1,345,283 ratings	First published November 6, 1939
99	Winnie-the-Pooh (Winnie-the-Pooh, #1)	A.A. Milne	4.36 avg rating	326,238 ratings	First published October 14, 1926

100 rows × 5 columns

QUIZZONE



Come posso verificare che il mio scraping sia andato a buon fine per tutti i libri?

Se c'è qualcosa che non è andata come ci aspettavamo, come possiamo verificarlo?

Principali funzioni del pacchetto BeautifulSoup

Metodi di navigazione:

- `find()`, `find_all()`: Trova elementi specifici nel documento in base a criteri come il tag, l'attributo, la classe, ecc.
- `select()`: Utilizza i selettori CSS per trovare elementi nel documento.

Metodi di ricerca:

- `find_parent()`, `find_parents()`: Trova il genitore o i genitori di un elemento.
- `find_next_sibling()`, `find_previous_sibling()`: Trova il fratello successivo o precedente di un elemento.

Metodi di estrazione di dati:

- `get_text()`: Estrae il testo contenuto all'interno di un elemento.
- `get()`: Ottiene il valore di un attributo di un elemento.

EPICODE

GRAZIE

Epicode