Data Collection
Web Scraping

Andrea Marchetti

Web Scraping

Il **web scraping** (detto anche **web harvesting** o **web data extraction**) è una tecnica di estrazione di dati da un sito web per mezzo di programmi software che appartengono alla famiglia dei **bot**.

Un esempio di web scraping è strettamente correlato **all'indicizzazione** dei siti Internet effettuato dai motori di ricerca - crawler

Il web scraping si concentra nell’estrarre **dati non strutturati** presenti nella pagine HTML e salvarli ad es su CSV

Web scraping

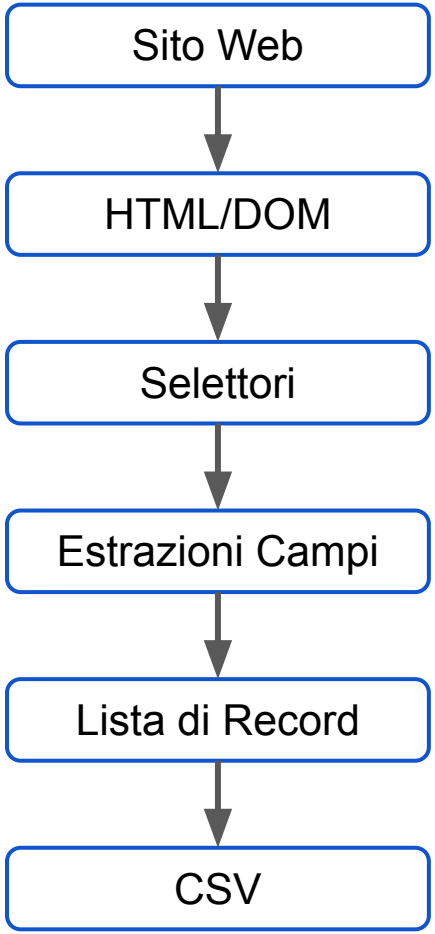
Librerie python Selenium, Playwright, nascono per simulare da programma le possibili interazioni di utente nei confronti di una applicazione Web

Posso scrivere un programma per

- aprire una finestra del browser,
- navigare in una pagina web
- compilare campi di input
- fare click sui pulsanti
- gestire le finestre di dialogo

Oltre che per testare applicazioni web questi prodotti sono usati per fare web scraping

Anatomia di uno scraper



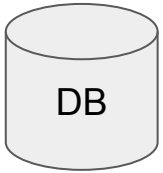
Nome articolo	Prezzo	Stelle
De'Longhi Stilosa EC260.W, Macchina da Caffè per Polvere o Cialde E.S.E., Sistema Latte Manuale per Cappuccino	119,90 €	4,2
De'Longhi Icona Vintage ECOV311.BK Macchina da Caffè Espresso Manuale e Cappuccino	149,90 €	4,0
Macchina Caffè Cialde E.S.E. per Espresso Cremoso – 10 CIALDE COMPRESE, Macchinetta Del Caffè Cialde Compatta	89,98 €	4,1
De'Longhi Dedica Style – Perfetto Macchina da Caffè Espresso, Compatibile con Cialde ESE, Montalatte	244,90 €	3,6
Macchina Caffè CIALDE e CAPSULE 7 in 1 Camry – Compatibile capsule Nespresso, Lavazza A Modo Mio, Dolce…	159,99 €	4,2

Metodi di Web Scraping

- Uso di Web API nascoste
- Dom Parsing
- Text Pattern matching
- Computer Vision + Machine Learning

Web API interne

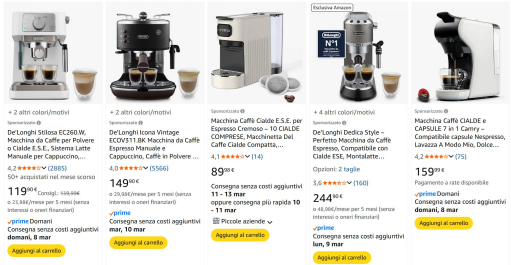
Siti popolati da Web Api interne



Web
API


```
"PostingDate": "2023-04-02",
"Lines": [
  {
    "LineNo": 10000,
    "Type": 2,
    "No": "1996-S",
    "Quantity": 12,
    "UnitPrice": 1397.3
  },
  {
    "LineNo": 20000,
    "Type": 2,
    "No": "1900-S",
    "Quantity": 4,
    "UnitPrice": 192.8
  }
]
}
```


Wrapping
HTML



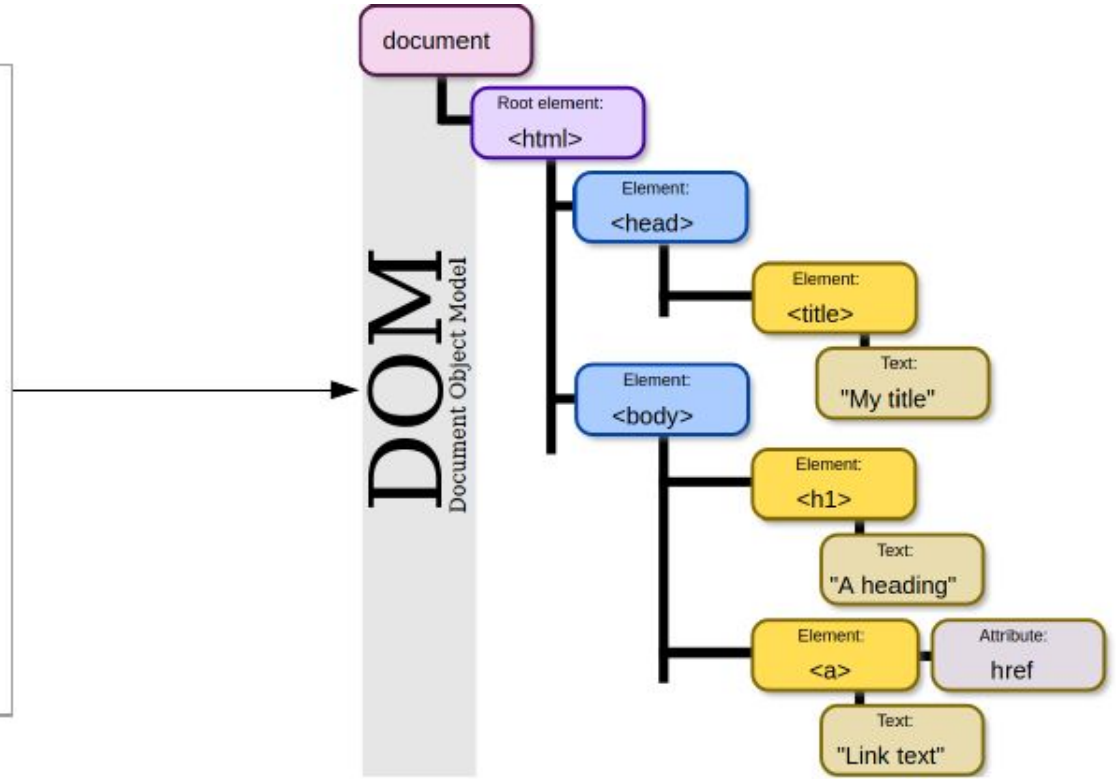
De wrapping

DOM Parsing

DOM Parsing

Raw HTML (text)

```
1  <!DOCTYPE html>
2  <html>
3
4  <head>
5      <title>My title</title>
6  </head>
7
8  <body>
9
10     <h1>A Heading</h1>
11     <a href="#">Link text</a>
12
13 </body>
14
15 </html>
```



Librerie Python per Web Scraping basate su DOM parsing

Beautifulsoap	Parsing HTML semplice e veloce
LXML	Parsing HTML semplice e veloce
Scrapy	Scraping su larga scala
Selenium	Interazione con pagine Javascript dinamiche
Playwright	Interazione con pagine Javascript dinamiche

Playwright

Cos’è Playwright

Sistema di Automazione di browser (**Chrome, Edge, Firefox, Safari**)

Funzionamento **headless** o con **interfaccia grafica**

Perfetto per **web scraping** e **test automatici**

Supporto a **Python**, **JavaScript**, **Java**, e altri linguaggi

Cos’è Playwrigth

Posso scrivere un **programma** che simula il comportamento umano per

- aprire una finestra del browser,
- navigare in una pagina web
- compilare campi di input
- fare click sui pulsanti
- gestire le finestre di dialogo

Storia di Playwright

- **2017**: **Google** sviluppa **Puppeteer**, una libreria per automatizzare esclusivamente Chrome.
- **2020**: **Microsoft** introduce **Playwright**, derivato da Puppeteer, che supporta più browser (Chrome, Edge, Firefox, Safari).
- Rapidamente adottato dalla community grazie alle prestazioni elevate e alla semplicità d’uso.

Playwright Vs Selenium

Prestazioni elevate: È generalmente più veloce e meno pesante di Selenium.

Facilità d'uso: Configurazione più semplice e intuitiva.

Gestione automatica di caricamenti e attese: Riduce la necessità di espliciti wait per elementi.

Multi-browser integrato: Non richiede setup aggiuntivi per browser diversi.

Installazione

Assicurati di avere Python 3.7+ installato

Installa Visual Studio Code da code.visualstudio.com

Installa l’estensione Python in VS Code

Installazione

Da Vs Code apri un terminale

```
python -m venv .venv # Crea un ambiente virtuale chiamato ".venv"

.venv\Scripts\activate # Attiva l’ambiente

pip install playwright # installa Playwright

playwright install chromium # installa Chrome/Edge
```

Usa l'ambiente virtuale in VS Code

- Apri il file .venv/bin/python (Mac/Linux) o venv\Scripts\python.exe (Windows).
- Premi Ctrl + Shift + P e digita "Python: Select Interpreter".
- Scegli l'interprete Python dell'ambiente virtuale (venv).

Installazione

```
pip install playwright
playwright install
```

Il primo comando installa la libreria Python per interagire con Playwright.

Il secondo comando installa i browser necessari al funzionamento di Playwright.

Test installazione

```
from playwright.sync_api import sync_playwright


with sync_playwright() as p:
    browser = p.chromium.launch(headless=True)
    page = browser.new_page()
    page.goto("https://example.com")
    print("Titolo della pagina:", page.title())
    browser.close()
```

Se funziona stampa:
Titolo della pagina: Example Domain

ATTENZIONE non usare NOTEBOOK con PlayWright

Demo

Dataroom di Milena Gabanelli

https://www.corriere.it/dataroom-milena-gabanelli/

Primo Esempio

```python
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()
    page.goto("http://localhost:8000/html/Toscana.html")

    # Seleziona tutti gli elementi <li> all'interno
    # della lista con id "Toscana"
    li_elements = page.locator("li")
    for i in range(li_elements.count()):
        print(li_elements.nth(i).inner_text())

    browser.close()
```

```html
<body>
    <h1>Regione Toscana</h1>
    <ul id="Toscana">
        <li>Arezzo</li>
        <li class="cap">Firenze</li>
        <li>Grosseto</li>
        <li>Livorno</li>
        <li>Lucca</li>
        <li>Massa-Carrara</li>
        <li>Pisa</li>
        <li>Pistoia</li>
        <li>Prato</li>
        <li>Siena</li>
    </ul>
</body>
```

Note

Il metodo `locator('li')` seleziona tutti gli elementi <li> presenti sulla pagina.

Utilizzando il metodo `.count()` ottieni il numero totale degli elementi trovati.

Puoi accedere ad ogni elemento individualmente con `.nth(indice)`.

Modalità Headless

```
browser = p.chromium.launch(headless=False|True)
```

headless = True # mostra la finestra del browser

utile in fase iniziale per testare cosa fa l'applicazione

headless=False # non mostra la finestra del browser

una volta che l'applicazione è pronta

Metadati della pagina

```
page.title()                                # Titolo della pagina

page.url                                    # URL della pagina

page.content()                              # sorgente html della pagina

page.screenshot(path="screenshot.png") # Salva uno screenshot della pagina
```

Tutti i modi per individuare elementi HTML con Playwright

Selezione tramite ruoli ARIA (Accessibilità)

```
# es. Ruolo: button, link, checkbox, radio, listitem ecc.
page.get_by_role('heading', name='Regione Toscana')
```

Selezione tramite ruoli ARIA (Accessibilità)

```
# es. Ruolo: button, link, checkbox, radio, listitem ecc.
page.get_by_role('heading', name='Regione Toscana')
```

Selezione tramite ruoli ARIA (Accessibilità)

```
# es. Ruolo: buttor
page.get_by_role('h
```

```
<h1>Regione Toscana</h1>
<ul id="Toscana">
    <li>Arezzo</li>
    <li class="Capoluogo">Firenze</li>
    <li>Grosseto</li>
    <li>Livorno</li>
   <li>Lucca</li>
   <!- . . . ->
</ul>
```

Ruoli ARIA gestiti da playwright

```
button
link
heading
textbox
searchbox
checkbox
radio
combobox
listbox
option
dialog
alertdialog
```

```
tab
tabpanel
tablist
menuitem
table
row
cell
columnheader
rowheader
list
listitem
img
```

```
progressbar
slider
spinbutton
switch
tooltip
navigation
main
banner
contentinfo
form
region
article
```

S el’elemento ha un ruolo accessibile chiaro, parti quasi sempre da qui.
Usalo per pulsanti, link, input, heading, checkbox, …

Ruoli ARIA gestiti da playwright

```
button
link
heading
textbox
s
c
r
c
l
option
dialog
alertdialog
tab
tabpanel
tablist
menuitem
list
listitem
img
progressbar
slider
spinbutton
switch
form
region
article
```

Se l'elemento ha un ruolo accessibile chiaro, parti quasi sempre da qui. Usalo per pulsanti, link, input, heading, checkbox,

S el’elemento ha un ruolo accessibile chiaro, parti quasi sempre da qui. Usalo per pulsanti, link, input, heading, checkbox, …

Selezione tramite label

per selezionare elementi associati a una label esplicita

```
page.get_by_label('Cerca provincia')
```

Selezione tramite label

per selezionare elemer

```
page.get_by_label
```

```
<h1>Regione Toscana</h1>
<ul id="Toscana">
   <li>Arezzo</li>
   <li class="Capoluogo">Firenze</li>
   <li>Grosseto</li>
   <!- . . . ->
</ul>
<label for="search">Cerca provincia</label>
<input id="search" type="text" >
```

Selezione tramite label

per selezionare elemer

```
page.get_by_label
```

```
<h1>Regione Toscana</h1>
<ul id="Toscana">
   <li>Arezzo</li>
   <li class="Capoluogo">Firenze</li>
   <li>Grosseto</li>
   <!- . . . ->
</ul>
```

Per i **form** è il selettore più leggibile: segue il significato, non il layout.

Selezione per testo

```
page.get_by_text("Regione Toscana")
```

Selezione per testo

```
page.get_by_text("R
```

```
<h1>Regione Toscana</h1>
<ul id="Toscana">
    <li>Arezzo</li>
    <li class="Capoluogo">Firenze</li>
    <li>Grosseto</li>
    <li>Livorno</li>
    <li>Lucca</li>
    <!- . . . ->
</ul>
```

Selezione per testo

```
page.get_by_text("R
```

```
<h1>Regione Toscana</h1>
<ul id="Toscana">
    <li>Arezzo</li>
    <li class="Capoluogo">Firenze</li>
    <li>Grosseto</li>
    <li>Livorno</li>
    <li>Lucca</li>
```

Molto utile quando il testo visibile è davvero univoco e stabile.
Da usare per link, pulsanti, messaggi, heading

Selettore CSS

```
page.locator('ul#Toscana > li.Capoluogo')
```

Selettore CSS

```
page.locator('ul#To
```

```
<h1>Regione Toscana</h1>
<ul id="Toscana">
    <li>Arezzo</li>
    <li class="Capoluogo">Firenze</li>
    <li>Grosseto</li>
    <li>Livorno</li>
    <li>Lucca</li>
    <!- . . . ->
</ul>
```

Selettore XPath

```
page.locator('//ul[@id="Toscana"]/li')

page.locator('//li')
```

Selettore XPath

```
page.locator('//ul
page.locator('//li
```

```
<h1>Regione Toscana</h1>
<ul id="Toscana">
    <li>Arezzo</li>
    <li class="Capoluogo">Firenze</li>
    <li>Grosseto</li>
    <li>Livorno</li>
    <li>Lucca</li>
    <!- . . . ->
</ul>
```

Modi x individuare elementi HTML

5 Selezione per placeholder (per input)

per campi input con placeholder definito

```
<!-- Input email -->
<
r                                                    e.com">
</div>
```


```
page.get_by_placeholder('nome@example.com')
```

Modi x individuare elementi HTML

7 Selezione tramite Alt-text (per immagini)

per elementi immagine <img>

```
page.get_by_alt_text('Logo')
```

Modi x individuare elementi HTML

8 Selezione per attributi personalizzati (data-attributes)

```
<li data-provincia="Firenze">Firenze</li>
```

```
page.locator('li[data-provincia="Firenze"]')
```

Concatenazione

```
page.locator('ul#Toscana').locator(':text("Firenze")')
page.get_by_text('Firenze').locator('xpath=..')  # genitore
```

Selezione del primo, ultimo, n-esimo

```
page.locator('ul#Toscana > li').first
page.locator('ul#Toscana > li').last
page.locator('ul#Toscana > li').nth(3)  # terzo elemento
```

Ordine consigliato dei selettori in Playwright

get_by_role()

get_by_label()

get_by_text()

locator(css): standard, semplice e leggibile.

locator(xpath): potente, ma meno leggibile, per selezioni complesse.

concatenazione: per elementi annidati

Documentazione CSS Selector, XPath

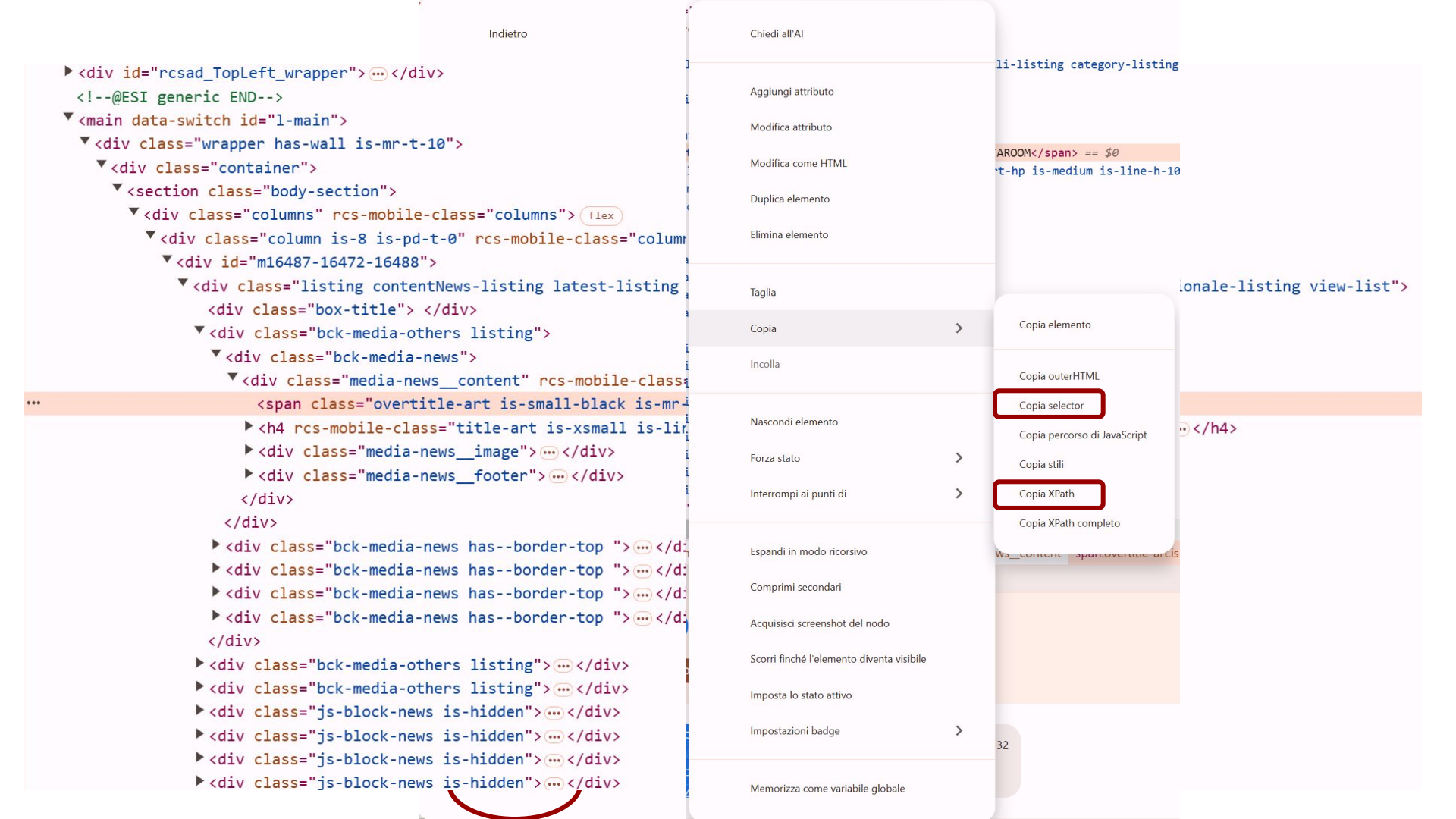
CSS Selectors Reference w3schools

Try CSS Selector Test di espressioni XPath

XPath Syntax w3schools

XPath Test di espressioni XPath

Strumenti per sviluppatori
Developer Tools


Indietro
Chiedi all'AI
Aggiungi attributo
Modifica attributo
Modifica come HTML
Duplica elemento
Elimina elemento
Taglia
Copia
Incolla
Nascondi elemento
Forza stato
Interrompi ai punti di
Espandi in modo ricorsivo
Comprimi secondari
Acquisisci screenshot del nodo
Scorri finché l'elemento diventa visibile
Imposta lo stato attivo
Impostazioni badge
Memorizza come variabile globale
Copia elemento
Copia outerHTML
Copia selector
Copia percorso di JavaScript
Copia stili
Copia XPath
Copia XPath completo
...
<!--@ESI generic END-->
<main data-switch id="l-main">

<section class="body-section">
flex
<div class="column is-8 is-pd-t-0" rcs-mobile-class="columm

<div class="listing contentNews-listing latest-listing

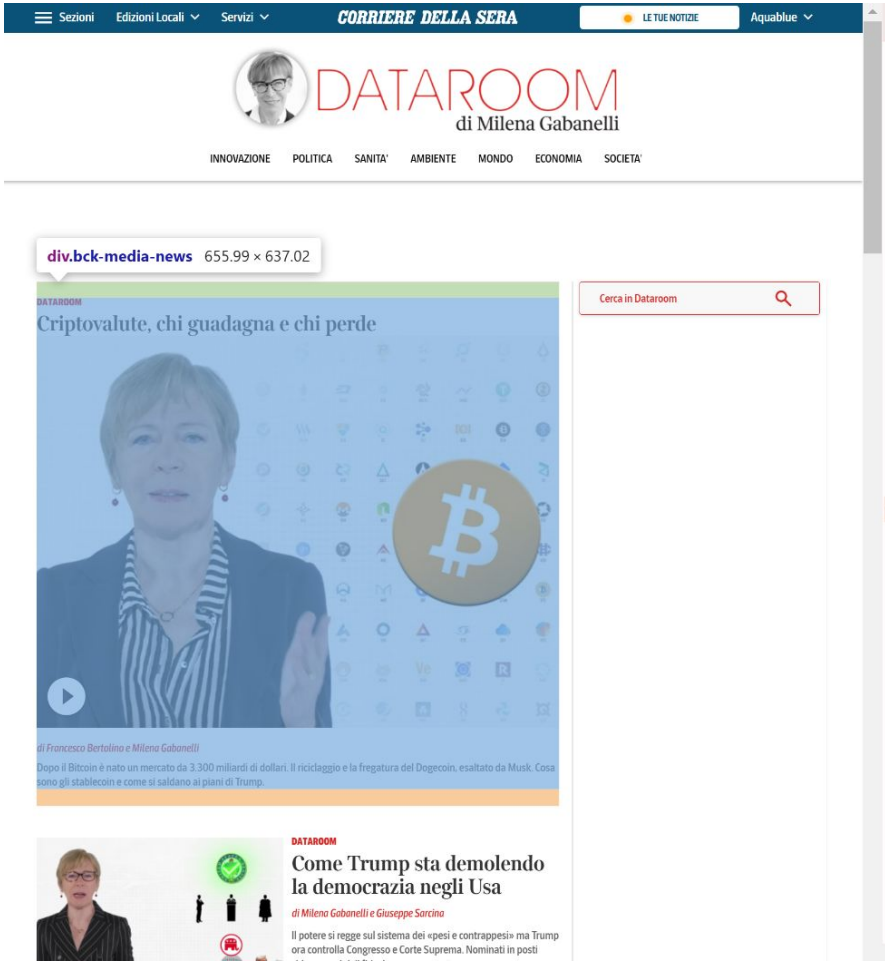
<div class="media-news__content" rcs-mobile-class
<span class="overtitle-art is-small-black is-mr-
<h4 rcs-mobile-class="title-art is-xsmall is-lin
...
...

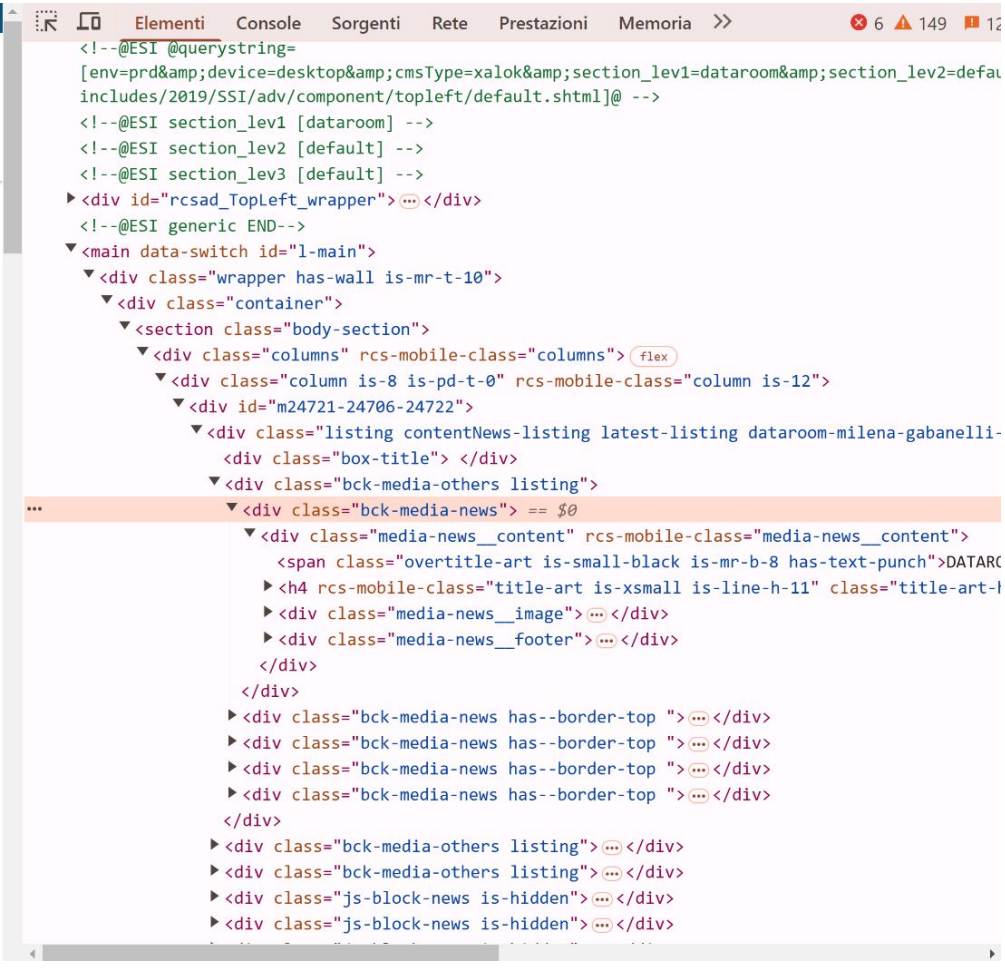
...</di
...</di
...</di
...</di

...
...
...
...
...
...
li-listing category-listing
AROOM == $0
t-hp is-medium is-line-h-10
onale-listing view-list">
</h4>
32

#m24721-24706-24722 > div > div:nth-child(2) > div:nth-child(1) > div > span

//*[@id="m24721-24706-24722"]/div/div[2]/div[1]/div/span


Sezioni
Edizioni Locali
Servizi
CORRIERE DELLA SERA
LE TUE NOTIZIE
Aquablue
DATAROOM
di Milena Gabanelli
INNOVAZIONE
POLITICA
SANITA'
AMBIENTE
MONDO
ECONOMIA
SOCIETA'
div.bck-media-news 655.99 × 637.02
DATAROOM
Criptovalute, chi guadagna e chi perde
di Francesco Bertolino e Milena Gabanelli
Dopo il Bitcoin è nato un mercato da 3.300 miliardi di dollari. Il riciclaggio e la fregatura del Dogecoin, esaltato da Musk. Cosa sono gli stablecoin e come si saldano ai piani di Trump.
Cerca in Dataroom
DATAROOM
Come Trump sta demolendo la democrazia negli Usa
di Milena Gabanelli e Giuseppe Sarcina
Il potere si regge sul sistema dei «pesi e contrappesi» ma Trump ora controlla Congresso e Corte Suprema. Nominati in posti


Elementi
Console
Sorgenti
Rete
Prestazioni
Memoria
6
149
<!--@ESI @querystring=
[env=prd&device=desktop&cmsType=xalok&section_lev1=dataroom&section_lev2=defau
includes/2019/SSI/adv/component/topleft/default.shtml]@ -->
<!--@ESI section_lev1 [dataroom] -->
<!--@ESI section_lev2 [default] -->
<!--@ESI section_lev3 [default] -->
...
<!--@ESI generic END-->
<main data-switch id="1-main">

<section class="body-section">
flex

<div class="listing contentNews-listing latest-listing dataroom-milena-gabanelli-

 == $0

DATARO
<h4 rcs-mobile-class="title-art is-xsmall is-line-h-11" class="title-art-
...
...

...
...
...
...

...
...
...
...

Metodi più importanti di PlayWright

Estrazione di contenuto

```
# Estrazione del contenuto visibile di un elemento.
titolo = page.locator("h1").inner_text()

# Estrazione di tutto il contenuto di un elemento.
paragrafo = page.locator("div.content").text_content()

# Estrazione di più elementi.
paragrafi = page.locator("p").all_inner_texts()

# Estrazione di attributi.
url = page.locator("a").get_attribute("href")
immagine = page.locator("img").get_attribute("src")
```

Interazione con Form elements

```
# Inserisce "Testo di esempio".
page.locator("input#text-input").fill("Testo di esempio")

# Seleziona l'opzione con value "opzione2".
page.locator("select#select").select_option("opzione2")

# Seleziona il radio button con id "radio2".
page.locator("input#radio2").check()

# Clicca sul pulsante di submit per inviare il form.
page.locator("button[type='submit']").click()
```

Spiegazione Demo

Dataroom di Milena Gabanelli

DataRoom

```
from playwright.sync_api import sync_playwright

url = "https://www.corriere.it/dataroom-milena-gabanelli/"

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page() # Apre una nuova pagina
    page.goto(url) # Vai su URL

    # Accetta cookie (se appare popup)
    try:
        page.click('button:text("ACCETTA E CONTINUA")')
    except:
        pass  # ignora se il popup non appare
```

DataRoom

```
# Trova tutti i tag <div> con classe "bck-media-news"
news = page.locator('div.bck-media-news')
# Cicla su news e crea un record per ogni new
results = []
for i in range(news.count()):
  results.append({
   'titolo': news.nth(i).locator('h4'             ).inner_text().strip(),
   'url'   : news.nth(i).locator('h4>a'           ).get_attribute('href'),
   'autori': news.nth(i).locator('span.author-art').inner_text().strip(),
   'desc'  : news.nth(i).locator('p.subtitle-art' ).inner_text().strip()
  })
```

DataRoom

```
    import pandas as pd

    # Salva i risultati su file CSV con pandas
    df = pd.DataFrame(results)
    df.to_csv('notizie.csv', index=False)

    browser.close() # Chiude il browser
```

Limiti dello scraping e contro misure

Limiti dello scraping

Per difendersi da azioni di scraping molti siti (Amazon, Google, Facebook, Instagram, …) adottano delle contromisure quali:

- Cambiare di nascosto il loro HTML
- Inserire dei blocchi anti-bot che controllano ad esempio la frequenza temporale delle richieste provenienti dallo stesso IP
- Verificare se il client è un browser o un bot che simula un browser

Raccomandazioni

identifica chiaramente il client con uno User-Agent descrittivo

limita la frequenza delle richieste

rispetta robots.txt e termini d’uso

attendi il caricamento completo della pagina

aggiungi pause, retry e gestione errori per non stressare il server

salva i risultati localmente invece di ripetere richieste inutili

gestisci cookie banner, login e redirect correttamente

imposta lingua, viewport e timezone coerenti con il test

usa selettori robusti e timeout ragionevoli

Dichiarazione agent

```
agent_chrome = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/145.0.0.0 Safari/537.36"

agent_esplicito = "CorsoDataJournalismBot/1.0 (ricerca didattica; contatto: email@example.com)"

browser = p.chromium.launch(headless=False)  # Cambia a False per vedere il browser
# Imposta un user agent per evitare blocchi
context = browser.new_context(user_agent=agent_chrome)
page = context.new_page()
```

Limitare la frequenza delle richieste al server

```
from playwright.sync_api import sync_playwright

import time
urls = ["https://example.com/page1","https://example.com/page2",
        "https://example.com/page3", ]

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()

    for url in urls:
        page.goto(url)
        time.sleep(2)  # aspetta 2 secondi
        page.wait_for_timeout(2000) # aspetta 2 secondi
        print(f"Visitato {url}")


    browser.close()
```

Limitare la frequenza delle richieste al server

 # introdurre una pausa casuale

```
time.sleep(random.uniform(1.5, 3.5))
```

 # limitare le richieste per minuto ex max 30 al minuto

```
requests_per_minute = 30
delay = 60 / requests_per_minute
time.sleep(delay)
```

Uno scraper responsabile introduce pause tra le richieste per non sovraccaricare il server e ridurre il rischio di blocco.

Attendere il caricamento di una pagina o di un elemento

```
# Attesa esplicita del caricamento completo della pagina
page.goto("https://example.com", wait_until="load")
# Attesa esplicita per un elemento specifico
page.goto("https://example.com")
page.locator("div.article-card").wait_for()
# Attesa esplicita per un elemento locator che mi interessa
page.goto("https://example.com", wait_until="load")
page.wait_for_selector("h1")
# Oppure
page.locator("h1").wait_for()
# Clicca su un link e aspetta che la nuova pagina sia completamente caricata
page.get_by_role("link", name="Notizie").click()
page.wait_for_load_state("load")
```

Attendere il caricamento di una pagina o di un elemento

```
# Attesa esplicita del caricamento completo della pagina
page.goto("https://example.com", wait_until="load")
# Attesa esplicita per un elemento specifico
page.goto("https://example.com")
page.locator("div.article-card").wait_for()
# Attesa esplicita per un elemento locator che mi interessa
page.goto("https://example.com", wait_until="load")
page.wait_for_selector("h1")
# Oppure
page.locator("h1").wait_for()
# Cli
page
page
```

Quando carichi una nuova pagina aspetta l’elemento che contiene i dati da estrarre

Rispettare i file robots.txt

corriere.it/robots.txt

google.com/robots.txt

amazon.it/robots.txt

- robots.txt va controllato rispetto allo User-Agent che dichiari.
- robots.txt dice se una pagina è consentita o no, ma non garantisce da solo uno scraping “corretto”

Un file robots.txt non blocca tecnicamente uno scraper, ma comunica ai crawler quali parti del sito non dovrebbero essere visitate

Gestione errori, pause, retry

```
for url in urls:
    try:
        page.goto(url, wait_until="domcontentloaded", timeout=15000)
    except TimeoutError:
        print("Timeout:", url)
        time.sleep(5)
        continue
    except Exception as e:
        print("Errore:", url, e)
        time.sleep(3)
        continue
    time.sleep(2)
```

Gestione errori, pause, retry

```
for url in urls:
    try:
        page.goto(url, wait_until="domcontentloaded", timeout=15000)
    except TimeoutError:
        print("Timeout:", url)
        time.sleep(5)
        continue
    except Exception as e:
        print("Errore:", url, e)
```

Aggiungere pause, retry e gestione errori significa rendere lo scraper più lento, stabile e rispettoso: meno richieste consecutive, pochi tentativi in caso di errore, comportamento prudente quando il server risponde male

Conclusioni

Risorse utili e documentazione

Documentazione ufficiale: playwright.dev/python

Repository GitHub: github.com/microsoft/playwright-python

Cosa abbiamo imparato

Fare scraping equivale a collezionare dati estratti da pagine web e salvarli (ex. csv)

I dati estratti necessitano di una fase di data cleaning

Le procedure di scraping sono fragili e necessitano di continua manutenzione