



# Data Journalism

## Web Scraping - Selenium

InfoUma 2019-20 *Andrea Marchetti*

# Selenium

---

- Selenium nasce nel 2004 come tool per lo sviluppo di **test automatici per applicazioni web**
- Possibile scrivere script che simulano l'interazione umana con una applicazione web
- Con il tempo si dimostra un ottimo tool per progettare attività di scraping
- [selenium.dev](https://selenium.dev)
- [selenium wikipedia](https://en.wikipedia.org/wiki/Selenium_(software))

# Componenti di Selenium

---

Selenium IDE

Selenium Web Driver

Selenium Grid

# Selenium IDE

---

Implementato come estensione di chrome e firefox

Consente di registrare, editare e riprodurre le azioni eseguite su di una pagina web

# Selenium Web Driver

---

Selenium-WebDriver esegue chiamate dirette al browser usando il supporto nativo contenuto nei browser per l'automazione

Queste chiamate dipendono dal browser che stiamo usando, questo significa che dovremo installare un **Driver** differente per il browser che vogliamo usare e utilizzare una interfaccia differente di **WebDriver**

Webdriver è una [raccomandazione W3C](#) del 2018

# Selenium GRID

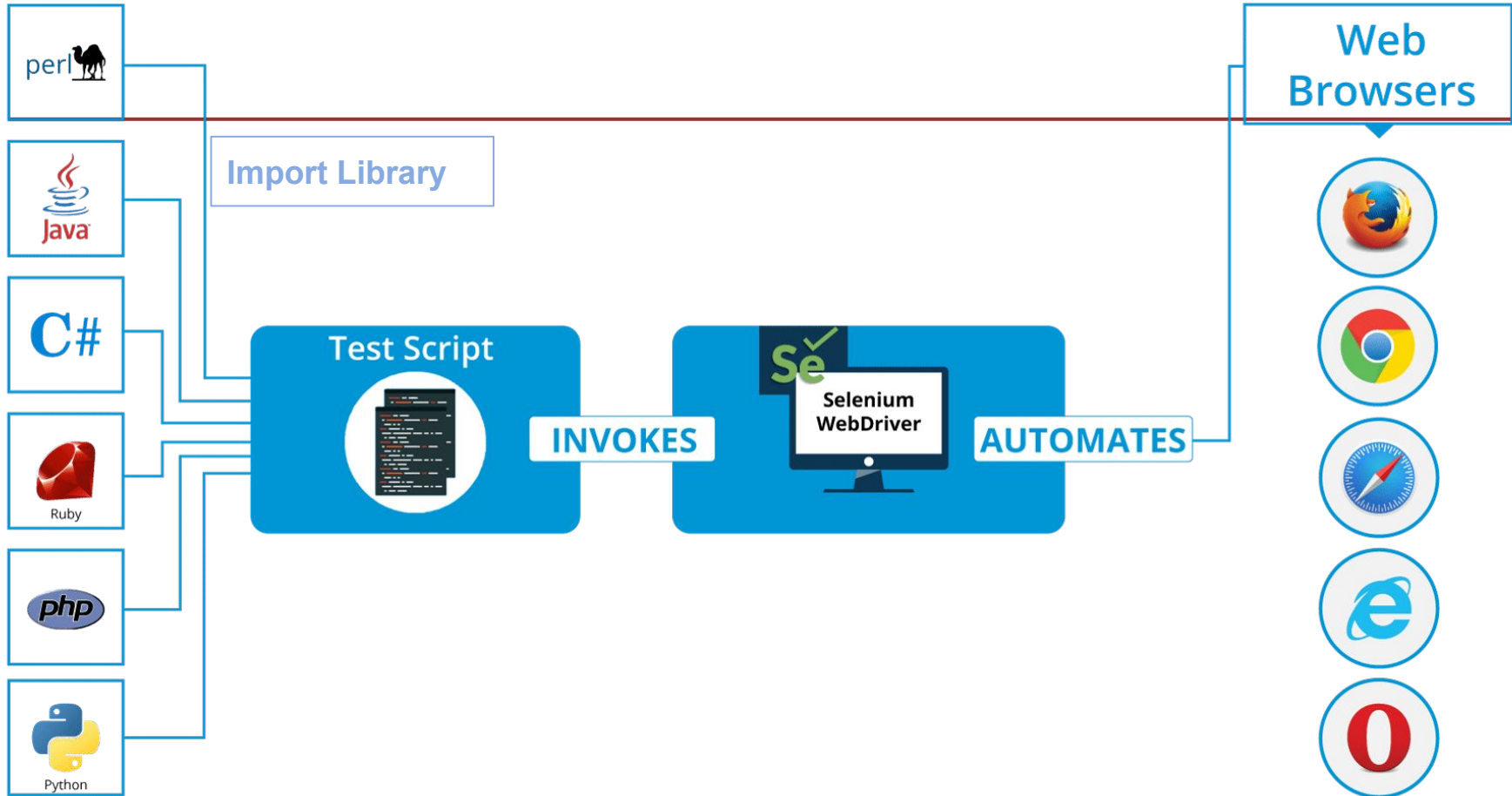
---

Consente di parallelizzare il lavoro svolto con WebDriver

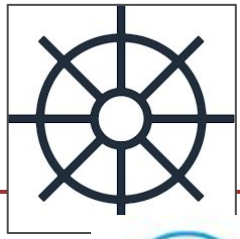
Riduzione del tempo

---

# Selenium WebDriver







Import Selenium  
WebDriver  
Library



SCRIPT

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

driver = webdriver.Chrome()
driver.get("http://www.example.com")

# Find an element by ID
element = driver.find_element_by_id("myForm")

# Click the element
element.click()

# Wait for the page to load
WebDriverWait(driver, 10).until(
    EC.presence_of_element_located((By.ID, "myForm"))
)

# Print the page title
print(driver.title)
```

ChromeDriver



# Driver dei maggiori browser

---

Implementano le specifiche WebDriver W3C

Requisiti per installare e usare i vari driver

- [ChromeDriver](#)
- [GeckoDriver](#) (Firefox)
- Safari
- Edge
- InternetExplorer
- Opera

```
from selenium import webdriver

# selenium ha le interfacce per i maggiori browser
driverChrome = webdriver.Chrome()
driverFirefox = webdriver.FireFox()
driverEdge = webdriver.Edge()
driverSafari = webdriver.Safari()
driverOpera = webdriver.Opera()
driverExplorer = webdriver.Ie()
```

# Lancio del driver di Chrome

---

```
from selenium import webdriver.Chrome

# Lancio del driver di Chrome che fa partire una pagina vuota del browser
driver = Chrome()

# Istruisco il driver di Chrome a caricare una url
driver.get("http://www.booking.com")
```

# Modalità Headless

---

Quando si invoca un driver di un browser questo fa partire una pagina del browser che poi verrà pilotata.

Con Chrome e Firefox si può evitare questo comportamento con l'opzione Headless

Utile per velocizzare lo scraping

```
from selenium.webdriver import Chrome
from selenium.webdriver.chrome.options import Options

chromeOptions = Options()
chromeOptions.add_argument("--headless")

driver = Chrome(options=chromeOptions)
driver.get("http://www.repubblica.it")
```

# Un primo esempio

Lo script, tramite un'istanza di Chrome apre la pagina <http://www.google.com> e attraverso la searchbar cerca il termine **selenium**.

```
from selenium import webdriver.Chrome
from selenium.webdriver.common.keys import Keys
```

```
driver = Chrome()
driver.get("http://www.google.com")
```

```
elem = driver.find_element_by_name("q")
elem.clear()
elem.send_keys("selenium")
elem.send_keys(Keys.RETURN)
```

```
driver.quit()
```



Cerca con Google

Mi sento fortunato

```
▶ <style data-impl="1583918643441">...</style>
  <div class="pR49Ae gsfi" jsname="vdLsw"></div>
...
  <input class="gLfyf gsfi" maxlength="2048" name="q" type="text" jsaction=
    "paste:puy29d" aria-autocomplete="both" aria-haspopup="false"
    autocapitalize="off" autocomplete="off" autocorrect="off" autofocus role=
    "combobox" spellcheck="false" title="Cerca" value aria-label="Cerca" data-
    ved="0ahUKEwjA_dngjJLoAhWKDxQKHfaQB28Q39UDCAY"> == $0
  </div>
▶ <div class="dRYYxd">...</div>
```

# Parsing e simulazione tastiera

---

```
elem = driver.find_element_by_name("q")
```

Il WebDriver offre diversi modi per trovare gli elementi presenti nella pagina. In questo caso localizziamo un elemento input text grazie al suo attributo **name**.

```
elem.clear()  
elem.send_keys("selenium")  
elem.send_keys(Keys.RETURN)
```

Per simulare la digitazione da tastiera utilizziamo il metodo **send\_keys()**. Prima però, ci assicuriamo che non ci sia già del testo nell'input text svuotandolo tramite il **metodo clear()**. Tasti speciali come INVIO possono essere simulati grazie alla classe **Keys**.

# WebDriver API for Python

---

[Selenium With Python: documentazione](#)

# Trovare un elemento nella pagina

---

Il WebDriver fornisce diverse strategie per localizzare un elemento presente nella pagina.

```
driver.find_element_by_id('')
driver.find_element_by_name('')
driver.find_element_by_xpath('')
driver.find_element_by_link_text('')
driver.find_element_by_partial_link_text('')
driver.find_element_by_tag_name('')
driver.find_element_by_class_name('')
driver.find_element_by_css_selector('')
```

Tutti i metodi restituiscono un oggetto di tipo **WebElement**. Nel caso non esista un elemento corrispondente ai criteri della ricerca viene generata l'eccezione **NoSuchElementException**.



# Trovare più elementi nella pagina

---

È possibile localizzare anche più elementi contemporaneamente.

```
driver.find_elements_by_name('')
driver.find_elements_by_xpath('')
driver.find_elements_by_link_text('')
driver.find_elements_by_partial_link_text('')
driver.find_elements_by_tag_name('')
driver.find_elements_by_class_name('')
driver.find_elements_by_css_selector('')
```

Tutti i metodi restituiscono una **lista** di oggetti di tipo **WebElement**.  
Nel caso non esista nemmeno un elemento corrispondente ai criteri della ricerca viene restituita una lista di lunghezza 0.

# WebElement

---

Il WebElement è un'interfaccia che rappresenta un elemento nella pagina. Ci permette di interagire con l'elemento e di estrarne informazioni attraverso vari metodi.

```
webelement.clear()
webelement.click()
webelement.get_attribute('')
webelement.get_property('')
webelement.is_displayed()
webelement.is_enabled()
webelement.is_selected()
webelement.screenshot('')
webelement.send_keys('')
webelement.submit()
webelement.value_of_css_property('')
```

# WebElement

---

È fornito anche di attributi.

```
webElement.id  
webElement.location  
webElement.location_once_scrolled_into_view  
webElement.parent  
webElement.rect  
webElement.screenshot_as_base64  
webElement.screenshot_as_png  
webElement.size  
webElement.tag_name  
webElement.text
```

Inoltre tutti i  
WebElement sono forniti  
degli stessi metodi di  
localizzazione presenti  
nel WebDriver.  
Ricerca un elemento a  
partire da un altro  
elemento

---

**Esempio**

# Femminile.Football.it

<https://femminile.football.it/ricerca.php>

Simulo un click sulla lettera Q che è un link

The screenshot shows a web browser displaying a search page for players. The page has a header with the word "Giocatori" and a search bar with the text "Cerca un Giocatore". Below the search bar is a grid of letters from A to Z. The letter "Q" is highlighted. Below the grid is a search button with the text "Ricerca libera: inserisci le lettere iniziali o il cognome intero". The browser's developer tools are open, showing the DOM structure. The selected element is a link with the href "?Ricerca=giocatori&iniziale=Q" and the title "18 risultati".

Si apre da Chrome "Strumenti per sviluppatori" e si "ispeziona" gli elementi del DOM

```
lettera = driver.find_element_by_link_text("Q").click()
```

text-  
alian

# Femminile.Football.it

**Giocatori**

Cerca un Giocatore

Seleziona l'iniziale del cognome:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Ricerca libera: inserisci le lettere iniziali o il cognome intero

**Risultati ottenuti:**

Clicca su un nome per accedere alla relativa scheda:

- QAFOKU MARTINA (4) [nato a il 28/11/2000 ]
- QUADRELLI SARA (57) [nato a Cattolica il 28/08/1997 ]
- QUAGGIOTTO ELENA (57) [nato a il 30/04/1992 ]
- QUAGLIA FRANCESCA (57) [nato a Padova il 13/04/1996 ]
- QUAGLIOTTO JESSICA (57) [nato a il 16/03/1993 ]
- QUARANTA MARTINA (57) [nato a il 19/05/1995 ]
- QUARESIMA MARA (57) [nato a il 15/04/2002 ]
- QUARTICELLI VALENTINA (57) [nato a Roma il 20/02/1990 ]
- QUARTO FEDERICA (57) [nato a il 13/04/1992 ]
- QUARTULLO FEDERICA (57) [nato a il 20/06/1991 ]
- QUATTROCCHI FRANCESCA (57) [nato a Palermo il 03/08/1988 ]
- QUAZZICO FRANCESCA (57) [nato a Taranto il 07/05/2001 ]
- QUERCIA MARZIA (57) [nato a il 16/09/1980 ]
- QUERZOLA LAURA (57) [nato a il 20/08/1991 ]
- QUIN JENNIFER (57) [nato a il ]
- QUINTO MICHELINA (57) [nato a il 30/03/1988 ]
- QUISTINI VALENTINA (57) [nato a il 30/11/1985 ]
- QUITADAMO SONIA (57) [nato a il 10/10/1985 ]

Elements Console Sources >> 4 2

```
><table width="100%" border="0"
cellspacing="0" cellpadding="2"
bgcolor="#F3f5f8" align="center">...
</table>
...
▼<table width="100%" border="0"
cellspacing="0" cellpadding="0"
height="20" bgcolor="#FFFFFF"> == $0
▼<tbody>
▼<tr>
▼<td height="20" nowrap
bgcolor="#F3f5f8" align="left">
"
"
<a href="/
schedagiocatore.php?
id_giocatore=8316">QAFOKU
MARTINA (4) [nato a il
28/11/2000 ]</a>
</td>
</tr>
```

tbody

Styles Ev

XPATCH della tabella contenente i dati  
`/html/body/table/tbody/tr/td/table/tbody/tr/td[2]`  
`/table/tbody/tr/td[2]/table[2]`

# Scheda player

[https://femminile.football.it/schedagiocatore.php?id\\_giocatore=2082](https://femminile.football.it/schedagiocatore.php?id_giocatore=2082)

Seleziona Scheda :  2018-2019

## MARA ZABBEO

### Scheda anagrafica

Data di nascita	07/08/1981	
luogo di nascita	Camposanpiero ( Padova ) - Italia	
nazione	Italia	
altezza		
peso forma		
ruolo	Difensore	
sito ufficiale		
Nazionalita	 Italiana	

### Presenze

	Presenze	Minuti (dal 0 in poi)	Gol	Squalifiche
<b>Totali</b>	<b>97</b>	<b>9977</b>	<b>1</b>	<b>1</b>
Serie A	0		0	0
Serie A2	43	4646	1	1
Serie B	54	5331	0	0
Regionali	0	0	0	0

### Squadre

Squadra	Campioni x squadra disputati	Serie A	Serie A2	Serie B	Regionali
<b>Totali</b>	<b>8</b>	<b>0</b>	<b>0</b>	<b>5</b>	<b>0</b>
Vicenza	1	0	0	1	0
Mestre	1	0	0	1	0
Schio	2	0	1	1	0
Padova	4	0	2	2	0

/html/body/table/tbody/tr/td/table/tbody/tr/td[2]/table[3]/tbody/tr[1]/td[2]/table//tr

/html/body/table/tbody/tr/td/table/tbody/tr/td[2]/table[5]/tbody/tr[1]/td[2]/table//tr

Cambiando id ottengo i dati di una calciatrice.

`getPlayerData(id)`

---

# Espressioni regolari in Python



# Python string methods

---

Python fornisce delle funzioni per manipolare le stringhe le più usate sono:

- `split('-')` # prende una stringa e restituisce una lista di stringhe
- `strip()` # elimina gli spazi bianchi marginali
- `replace()` # sostituisce una sottostringa con un'altra

Le altre possono essere trovate su [w3schools](https://www.w3schools.com/python/python_string_methods.asp)

# Python Espressioni Regolari

---

```
import re # importare la libreria
```

## **match**

restituisce un'istanza di MatchObject se zero o più caratteri all'inizio della stringa corrispondono al pattern. È comunque possibile specificare una posizione da cui iniziare la ricerca diversa dalla testa della stringa, e anche un limite superiore). Restituisce None se la stringa non corrisponde al pattern.

## **search**

scandisce la stringa cercando una corrispondenza con il pattern e restituisce l'istanza MatchObject corrispondente. Restituisce None se in nessuna posizione la stringa corrisponde al pattern. Anche qui è possibile specificare un punto d'inizio e di termine per la ricerca come per match.

## **split**

in breve divide la stringa secondo le occorrenze del pattern e restituisce le porzioni ottenute come lista.

## **escape**

prende in ingresso una stringa e la restituisce con i caratteri alfanumerici protetti da barre oblique inverse.

## **sub**

restituisce la stringa ottenuta sostituendo alle occorrenze non sovrapposte del pattern l'entità rimpiazzo, che può essere una stringa o una funzione, eventualmente per un numero di volte prefissato.

# Sintassi Espressioni Regolari

---

Character	Description
[]	A set of characters
\	Signals a special sequence (can also be used to escape special characters)
.	Any character (except newline character)
^	Starts with
\$	Ends with
*	Zero or more occurrences
+	One or more occurrences
{}	Exactly the specified number of occurrences
	Either or
()	Capture and group

# Regular Expressions

---

[documentazione](#)

test on regular expression tester [online](#)

# Save a list of dictionary to file

---

```
import json
with open('playersDB.json', 'w') as jsonFile:
    json.dump(playersDB, jsonFile)
```



```
import csv
keys = playersDB[0].keys() # Recupero i nomi dei campi dal primo dictionary
with open('playersDB.csv', 'w', newline='') as csvFile:
    dict_writer = csv.DictWriter(csvFile, keys) # imposto i nomi dei campi
    dict_writer.writeheader()                 # scriva la prima riga
    dict_writer.writerows(playersDB)          # scrivo i dati
```

# Attese in Selenium

## documentazione

- contenuti diversi possono essere caricati sulla pagina **con tempi diversi**
  - la diffusa presenza di tecnologie come AJAX rende frequente questa possibilità
- la ricerca di un elemento **non ancora presente** solleva **un'eccezione**
- la gestione delle attese durante lo scraping può risolvere questi problemi
- inserire attese può prevenire anche possibili **banning** da parte del server

# Selenium WebDriver

---

Fornisce due tipi di waits

Esplicita

Implicita

# WebDriverWait

Selenium WebDriver fornisce due tipi di wait;

- Esplicita
  - Aspetta finché si verifica una certa condizione prima di procedere con la ricerca
- Implicita
  - Ripete la ricerca ad intervalli di tempo regolari (POLL) per un certo periodo di tempo



# WebDriverWait Esplicita

```
from selenium.webdriver.support.ui import WebDriverWait  
  
elem = WebDriverWait(driver, tempo).until(condizione)
```

nel dettaglio, l'esecuzione riprende se:

- si verifica la **condizione** specificata
- si raggiunge un **tempo** massimo per l'attesa

# Expected Conditions

- è possibile specificare la propria condizione andando a definire delle classi Python opportune
- sono già previste una serie di condizioni di uso comune che possono essere usate direttamente:
  - *visibility\_of\_element\_located*
  - *element\_to\_be\_clickable*
  - *text\_to\_be\_present\_in\_element*
  - ...

```
from selenium.webdriver.support import expected_conditions as EC
```

```
elem = WebDriverWait(driver, tempo)
```

# Selezione elementi

- il modo più semplice per selezionare elementi all'interno di una condizione è tramite la cosiddetta **classe By**
- è uno dei metodi previsti in Selenium per la selezione degli elementi
- tramite la **classe By** è possibile specificare numerosi criteri di selezione:
  - *By.ID*
  - *By.XPATH*
  - *By.CLASS\_NAME*
  - *By.CSS\_SELECTOR*
  - ...

# Utilizzo della classe By

```
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By
```

```
driver = webdriver.Firefox()
driver.get('http://www.mypage.com')
```

```
wait = WebDriverWait(driver, 10)
```

```
try:
    element = wait.until(EC.presence_of_element_located(
        (By.CSS_SELECTOR, "span.titolone")))
```

# WebDriverWait Implicita

```
from selenium import webdriver
```

```
driver = webdriver.Firefox()
```

```
driver.implicitly_wait(10) # 10 seconds
```

```
driver.get("http://somedomain/url_that_delays_loading")
```

```
myDynamicElement = driver.find_element_by_id("myDynamicElement")
```