Data Journalism

Web Scraping

InfoUma 2018-19 ***Andrea Marchetti***

Cos'è Selenium http://www.seleniumhq.org/

- Selenium nasce nel 2004 come tool per lo sviluppo di **test automatici per applicazioni web**
- Sw che permette di guidare le interazioni con una pagina web
- Selenium ha due componenti
  - Selenium IDE
  - Selenium Web Driver.

Selenium IDE

- Selenium IDE è un ambiente di sviluppo integrato per gli script di Selenium
- È implementato come un estensione di Firefox o di Chrome
- Permette di registrare, editare e riprodurre test automatici sul browser.

http://www.seleniumhq.org/projects/ide/

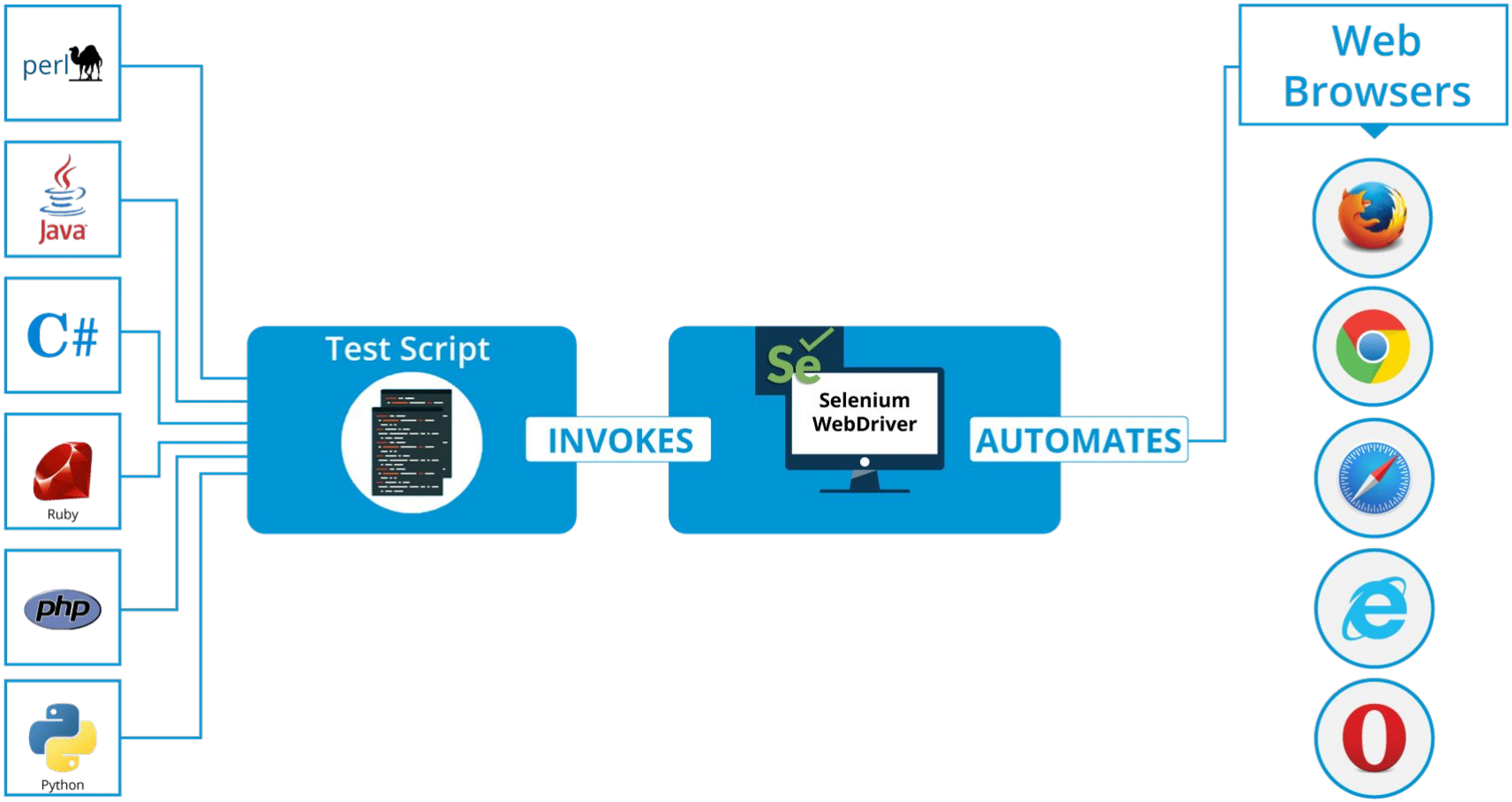
Selenium WebDriver

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

Queste chiamate dipendono dal browser che stiamo usando, questo significa che dovremo installare un **webdriver** differente per il browser che vogliamo usare

Selenium WebDriver

- La libreria Selenium è disponibile in diversi linguaggi: Java, C#, **Python**, Ruby, Perl, PHP, Javascript.
- WebDriver offre al momento sette diverse interfacce, capaci di relazionarsi con diversi browser: Firefox, Internet Explorer, Chrome, Opera, PhantomJS.


perl
Java
C#
Ruby
php
Python
Test Script
INVOKES
Selenium
WebDriver
AUTOMATES
Web
Browsers

Web Scraping

- La possibilità di automatizzare la navigazione lo rende un ottimo strumento per fare web scraping, soprattutto nelle pagine web dove è presente una forte componente di interazione javascript dove gli altri strumenti falliscono.
- Nel prossimo esempio sarà utilizzato come linguaggio di programmazione Python. È possibile trovare una documentazione ai seguenti link:

http://www.seleniumhq.org/docs/03_webdriver.jsp#introducing-the-selenium-webdriver-api-by-example

http://selenium-python.readthedocs.io/

Installazione

https://selenium-python.readthedocs.io/installation.html

Installare l'interfaccia a selenium

```
pip install selenium
```

Scaricarsi il driver relativo al browser usato
Chrome:

https://sites.google.com/a/chromium.org/chromedriver/downloads

Edge:

https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/

Firefox:

https://github.com/mozilla/geckodriver/releases

Safari:

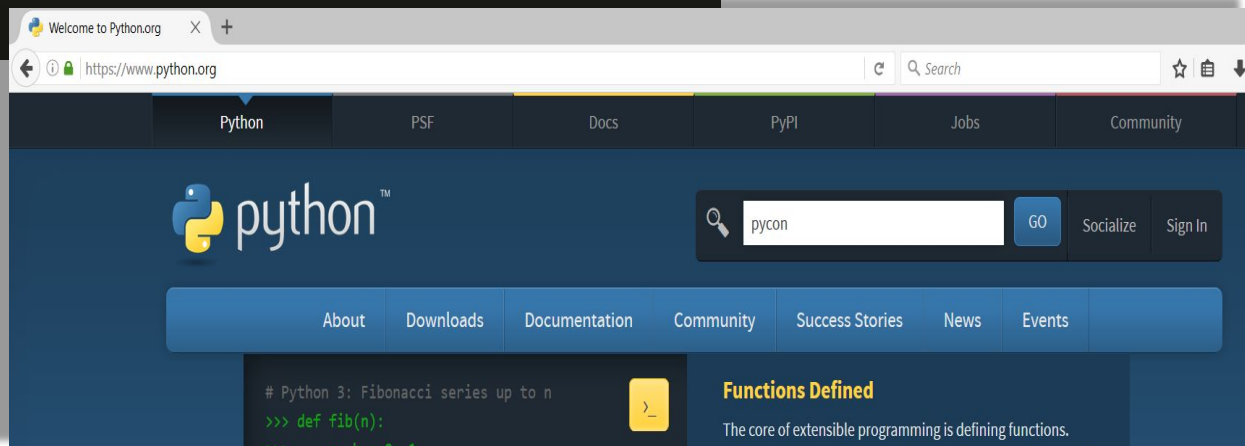
https://webkit.org/blog/6900/webdriver-support-in-safari-10/

Un primo esempio

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

driver = webdriver.Firefox()
driver.get("http://www.python.org")
assert "Python" in driver.title
elem = driver.find_element_by_name("q")
elem.clear()
elem.send_keys("pycon")
elem.send_keys(Keys.RETURN)
assert "No results found." not in driver.page_source
driver.close()
```

Lo script, tramite un'istanza di Firefox apre la pagina http://www.python.org e attraverso la searchbar cerca il termine pycon.



Analizziamo il codice - 1

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

Il modulo selenium.webdriver fornisce il necessario per interfacciarsi con il browser tra cui i nomi dei tasti della keyboard

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

Viene creata un'istanza di Firefox.

```
driver.get("http://www.python.org")
```

Il metodo driver.get() apre la pagina data dall'URL. Il WebDriver attenderà che la pagina sia stata caricata completamente (quando l'evento onload viene lanciato) prima di ridare il controllo allo script. Se la pagina utilizza molto AJAX durante il caricamento, può risultare complicato per il WebDriver sapere quando la pagina è stata caricata al 100%.

Analizziamo il codice - 2

```
assert "Python" in driver.title
```

Asserzione per confermare che nel title della pagina sia presente la parola “Python”.

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

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

```
elem.clear()
elem.send_keys("pycon")
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.

Analizziamo il codice - 3

```
assert "No results found." not in driver.page_source
```

Asserzione per confermare che nel sorgente della pagina non sia presente la stringa “No results found.” e che quindi siano stati trovati dei risultati.

```
driver.close()
```

Chiude l’istanza aperta di Firefox.

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 generate l'eccezione NoSuchElementException.

Trovare più elementi nella pagina

È possibile localizzare più 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 - 1

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 - 2

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.

Usare selenium - 1

Come abbiamo già visto, per navigare su una pagina si usa il comando:

```
driver.get(URL)
```

Mentre per selezionare un elemento si può usare il metodo:

```
element = driver.find_element_by_css_selector(SELETTORE)
```

Se vogliamo invece ottenere una lista di elementi il metodo da utilizzare è:

```
elements = driver.find_elements_by_css_selector(SELETTORE)
```

Questo metodo è molto utile quando dobbiamo stessa classe CSS.

Usare selenium - 2

Una volta ottenuta una lista, la possiamo scorrere grazie al metodo for:

```
for element in elements:
    FAI QUALCOSA
```

Mentre per selezionare un elemento all'interno di quello corrente si può usare il metodo:

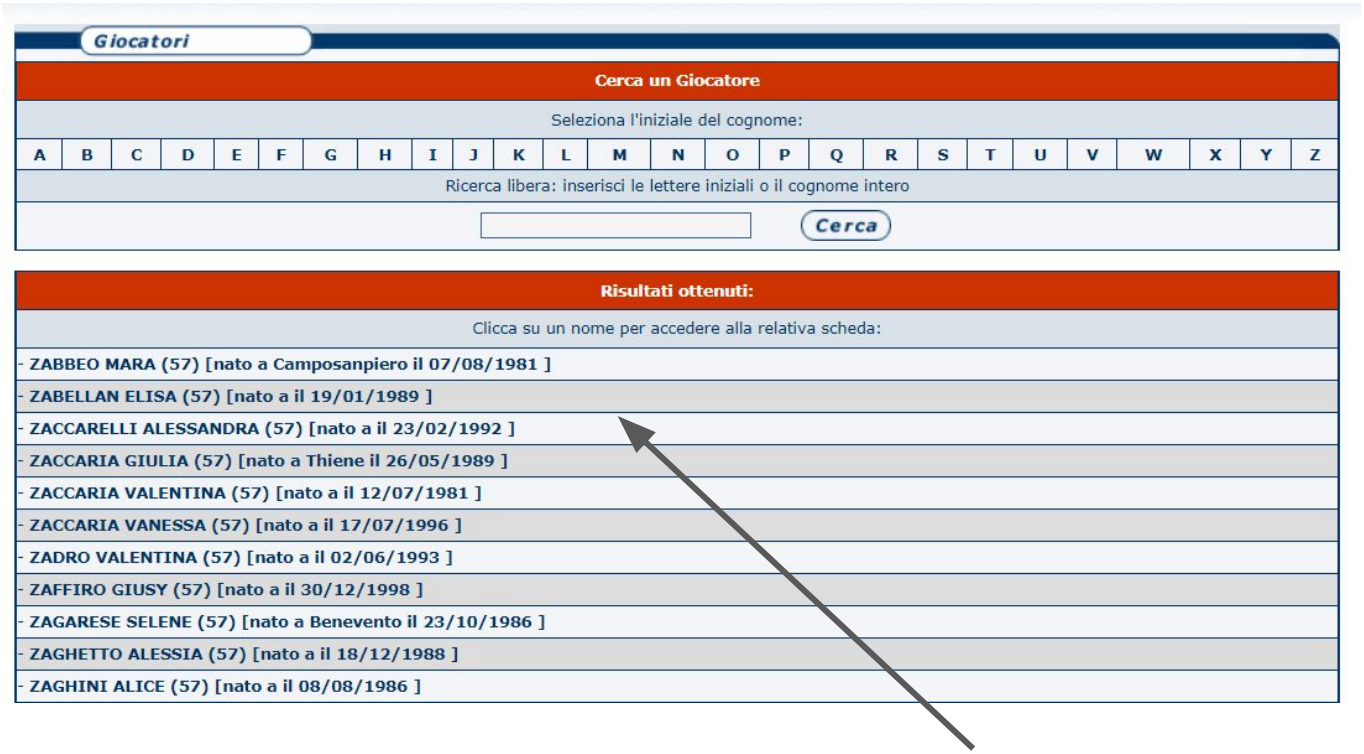
```
element.find_element_by_css_selector(SELETTORE)
```

È importante ricordare che ottenuto un elemento dal driver tramite un metodo di ricerca è possibile effettuare una nuova ricerca all'interno dell'elemento.

Esempio

Scheda players

https://femminile.football.it/ricerca.php?Ricerca=giocatori&iniziale=Z



Cambiando la lettera ottengo tutte le calciatrici.

getPlayers(letter)

XPATH 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

Seleziona Scheda : Riepilogo Generale | 2018-2019 | OK

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
Totali	**97**	**9977**	**1**	**1**
Serie A	0	0	0	0
Serie A2	43	4646	1	1
Serie B	54	5331	0	0
Regionali	0	0	0	0

Squadre

Squadra	Campionati x squadra disputati	Serie A	Serie A2	Serie B	Regionali
Totali	**8**	**0**	**0**	**5**	**0**
Vicenza	1	0	0	1	0
Mestre	1	0	0	1	0
Schio	2	0	1	1	0
Padova	4	0	2	2	0

Cambiando id ottengo i dati di una calciatrice.

getPlayerData(id)

/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

Python string methods

w3schools

`split('-')` # prende una stringa e restituisce una lista di stringhe

`strip()` # elimina gli spazi bianchi

`replace()` # sostituisce una sottostringa con un altra

Regular Expressions

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
{}	Excactly the specified number of occurrences
|	Either or
()	Capture and group

Regular Expressions

([0-9]){2}\/([0-9]){2}\/([0-9]){4}

match 22/33/4444

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)
       .until(EC.element_to_be_clickable(selezione elemento))
```

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")))
except:
    print("something went bad")
finally:
    driver.quit()
```

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")
```