

# Riferimento rapido Selenium WebDriver

Automazione del browser, interazione con elementi, attese e asserzioni

## Configurazione

### Installazione

```
pip install selenium webdriver-manager
# webdriver-manager auto-downloads browser drivers
```

### Configurazione base del driver

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
driver = webdriver.Chrome(
    service=Service(ChromeDriverManager().install()))
```

### Modalità headless

```
options = webdriver.ChromeOptions()
options.add_argument("--headless=new")
options.add_argument("--no-sandbox")
driver = webdriver.Chrome(options=options)
```

### Browser supportati

<b>webdriver.Chrome()</b>	Google Chrome / Chromium
<b>webdriver.Firefox()</b>	Mozilla Firefox (GeckoDriver)
<b>webdriver.Edge()</b>	Microsoft Edge (Chromium)
<b>webdriver.Safari()</b>	Apple Safari (solo macOS)

## Browser e navigazione

### Navigazione

```
driver.get("https://example.com")
driver.back() # browser back
driver.forward() # browser forward
driver.refresh() # reload page
```

### Proprietà del browser

<b>driver.title</b>	Titolo della pagina corrente
<b>driver.current_url</b>	URL della pagina corrente
<b>driver.page_source</b>	Sorgente HTML completa della pagina
<b>driver.get_cookies()</b>	Elenco di tutti i cookie

### Gestione delle finestre

```
driver.set_window_size(1920, 1080)
driver.maximize_window()
driver.minimize_window()
driver.quit() # close all windows, end session
```

## Ricerca degli elementi

### Strategie di localizzazione

```
from selenium.webdriver.common.by import By
driver.find_element(By.ID, "login-btn")
driver.find_element(By.CLASS_NAME, "nav-item")
driver.find_element(By.CSS_SELECTOR, "div.card > h2")
driver.find_element(By.XPATH, "//input[@name='q']")
```

### Strategie By

<b>By.ID</b>	Corrisponde all'attributo id dell'elemento
<b>By.NAME</b>	Corrisponde all'attributo name dell'elemento
<b>By.CLASS_NAME</b>	Corrisponde alla classe CSS (una sola classe)
<b>By.TAG_NAME</b>	Corrisponde al nome del tag HTML
<b>By.CSS_SELECTOR</b>	Selettore CSS (il più flessibile)
<b>By.XPATH</b>	Espressione XPath
<b>By.LINK_TEXT</b>	Testo esatto del link
<b>By.PARTIAL_LINK_TEXT</b>	Corrispondenza parziale del testo del link

## Ricerca multipla

```
items = driver.find_elements(By.CSS_SELECTOR, "li.item")
for item in items:
    print(item.text)
# Returns empty list if none found (no exception)
```

## Interazione

### Clic e digitazione

```
elem = driver.find_element(By.ID, "search")
elem.clear() # clear existing text
elem.send_keys("selenium python")
elem.submit() # submit parent form
```

### Menu a tendina

```
from selenium.webdriver.support.ui import Select
select = Select(driver.find_element(By.ID, "country"))
select.select_by_visible_text("Canada")
select.select_by_value("ca")
select.select_by_index(2)
```

### Proprietà degli elementi

<b>.text</b>	Contenuto testuale visibile
<b>.get_attribute('href')</b>	Valore dell'attributo HTML
<b>.is_displayed()</b>	True se l'elemento è visibile
<b>.is_enabled()</b>	True se l'elemento è interattivo
<b>.is_selected()</b>	True se la casella/radio è selezionata
<b>.tag_name</b>	Tag HTML (es. 'input', 'div')
<b>.value_of_css_property('color')</b>	Valore calcolato della proprietà CSS

## Attese

### Attesa esplicita

```
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
elem = WebDriverWait(driver, 10).until(
    EC.presence_of_element_located((By.ID, "result")))
```

### Condizioni attese

<b>presence_of_element_located</b>	L'elemento esiste nel DOM
<b>visibility_of_element_located</b>	L'elemento è visibile nella pagina
<b>element_to_be_clickable</b>	L'elemento è visibile e abilitato
<b>text_to_be_present_in_element</b>	L'elemento contiene il testo atteso
<b>alert_is_present</b>	È visualizzato un alert JavaScript
<b>staleness_of</b>	L'elemento non è più nel DOM
<b>title_contains</b>	Il titolo della pagina contiene il testo

### Attesa implicita

```
driver.implicitly_wait(10) # seconds, applies globally
# Explicit waits are preferred - more precise control
```

## Frame e finestre

### Frame

```
driver.switch_to.frame("frame-name") # by name/id
driver.switch_to.frame(0) # by index
driver.switch_to.frame(elem) # by element
driver.switch_to.default_content() # back to main
```

## Finestre e schede

```
original = driver.current_window_handle
driver.switch_to.new_window("tab") # open new tab
driver.switch_to.window(original) # switch back
driver.close() # close current tab
```

## Alert

```
alert = driver.switch_to.alert
print(alert.text)
alert.accept() # click OK
alert.dismiss() # click Cancel
alert.send_keys("input text")
```

## Screenshot

### Cattura screenshot

```
driver.save_screenshot("page.png") # full page
elem = driver.find_element(By.ID, "chart")
elem.screenshot("chart.png") # single element
```

### Screenshot come Base64

```
b64 = driver.get_screenshot_as_base64()
png = driver.get_screenshot_as_png() # bytes
```

## Azioni

### Catene di azioni

```
from selenium.webdriver.common.action_chains import ActionChains
actions = ActionChains(driver)
actions.move_to_element(menu).click().perform()
```

### Azioni da tastiera

```
from selenium.webdriver.common.keys import Keys
elem.send_keys(Keys.ENTER)
elem.send_keys(Keys.CONTROL, "a") # select all
actions.key_down(Keys.SHIFT).click(elem).perform()
```

### Azioni del mouse

<b>.click(elem)</b>	Clic sull'elemento
<b>.double_click(elem)</b>	Doppio clic sull'elemento
<b>.context_click(elem)</b>	Clic destro sull'elemento
<b>.move_to_element(elem)</b>	Passa il mouse sull'elemento
<b>.drag_and_drop(src, dst)</b>	Trascina la sorgente sulla destinazione
<b>.click_and_hold(elem)</b>	Tieni premuto il pulsante del mouse
<b>.release()</b>	Rilascia il pulsante del mouse

## Asserzioni

### Asserzioni comuni (pytest)

```
assert "Dashboard" in driver.title
assert driver.find_element(By.ID, "msg").text == "Done"
assert driver.current_url.endswith("/home")
assert len(driver.find_elements(By.CSS_SELECTOR, "tr")) > 0
```

### Asserzioni basate su attesa

```
WebDriverWait(driver, 5).until( # appears
    EC.visibility_of_element_located((By.ID, "success")))
WebDriverWait(driver, 5).until( # disappears
    EC.invisibility_of_element_located((By.ID, "spinner")))
```

### Esecuzione JavaScript

```
result = driver.execute_script("return document.title")
driver.execute_script(
    "arguments[0].scrollIntoView(true);", elem)
```

# Riferimento rapido Selenium WebDriver

---

## Pattern comuni

### Pattern Page Object

```
class LoginPage:
    URL = "/login"
    user_loc = (By.ID, "username")
    def login(self, drv, user, pwd):
        drv.find_element(*self.user_loc).send_keys(user)
```

### Context manager

```
from selenium import webdriver
with webdriver.Chrome() as driver:
    driver.get("https://example.com")
    print(driver.title)
# driver.quit() called automatically
```

### Ripetizione e pulizia

```
try:
    driver.get("https://example.com")
    WebDriverWait(driver, 10).until(
        EC.element_to_be_clickable((By.ID, "btn")))
finally: driver.quit()
```