SELENIUM

By: Julio Berrocal Alvarez

AGENDA

- About Selenium
- Selenium for Data Science
- Installation
- Quick Tutorial
- Example #1 – Selenium for Web Scraping
- Example #2 – Selenium for Web Testing

ABOUT SELENIUM

- **Utility:** Selenium is a powerful open-source tool for web browser automation. It allows users to automate web browsers across many platforms.
- **History:** First developed in 2004 by Jason Huggins as an internal tool at ThoughtWorks, building the Core mode as "JavaScriptTestRunner" for the testing of an internal Time and Expenses application.
- **Cross-Browser Compatibility**: Automate operations across different browsers like Chrome, Firefox, Safari, and Internet Explorer.
- **Language Support**: Write scripts in multiple programming languages including Java, C#, Python, and JavaScript.
- **Selenium WebDriver**: Allows direct calls to the browser using each browser’s native support for automation.

SELENIUM FOR DATA SCIENCE

- **Web Scraping**: Automate data extraction from websites for data analysis, market research, and data visualization projects.
- **Testing and Quality Assurance**: Automate testing of web applications to ensure accuracy and efficiency in data-driven applications.
- **Automating Repetitive Tasks**: Use Selenium for automating repetitive web-based administration and data entry tasks to save time and reduce errors.
- **Data Collection at Scale**: Collect large volumes of data from websites which can be processed and analyzed for insights into various data science applications.

INSTALLATION

- **Install Selenium**: Use pip, the Python package installer. Open your command line or terminal and type

```
pip install selenium
```

- **WebDriver**: Download the WebDriver for your preferred browser(s). Each browser has its own driver, which must be installed separately.

- **Import**: Import all the necessary libraries into your script.

```
# Import necessary libraries
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
```

- **Verify**: Write a short script to verify the installation worked, such as the following:

```
from selenium import webdriver
driver = webdriver.Chrome()
driver.get("http://www.google.com")
print(driver.title)
driver.quit()
```

TUTORIAL USING GOOGLE

- **Accessing a Website**:

```
# Accessing a website

url = 'https://www.google.com/'
driver = webdriver.Chrome()
driver.get(url)
```

- **Printing the Title and Source Code:**

```
# Printing the Website's Title
print(driver.title)

# Printing the page source code
print(driver.page_source)
```

- **Submitting a Query into the Google Search Bar:**

```
# Submitting a query into the Google search bar

# Finding the search bar element
search = driver.find_element(By.NAME, 'q')

# Typing our query and submitting it
search.send_keys('Jupiter')
search.send_keys(Keys.RETURN)

# Retreiving the Current URL to visualize the search
print(driver.current_url)
```

TUTORIAL USING GOOGLE

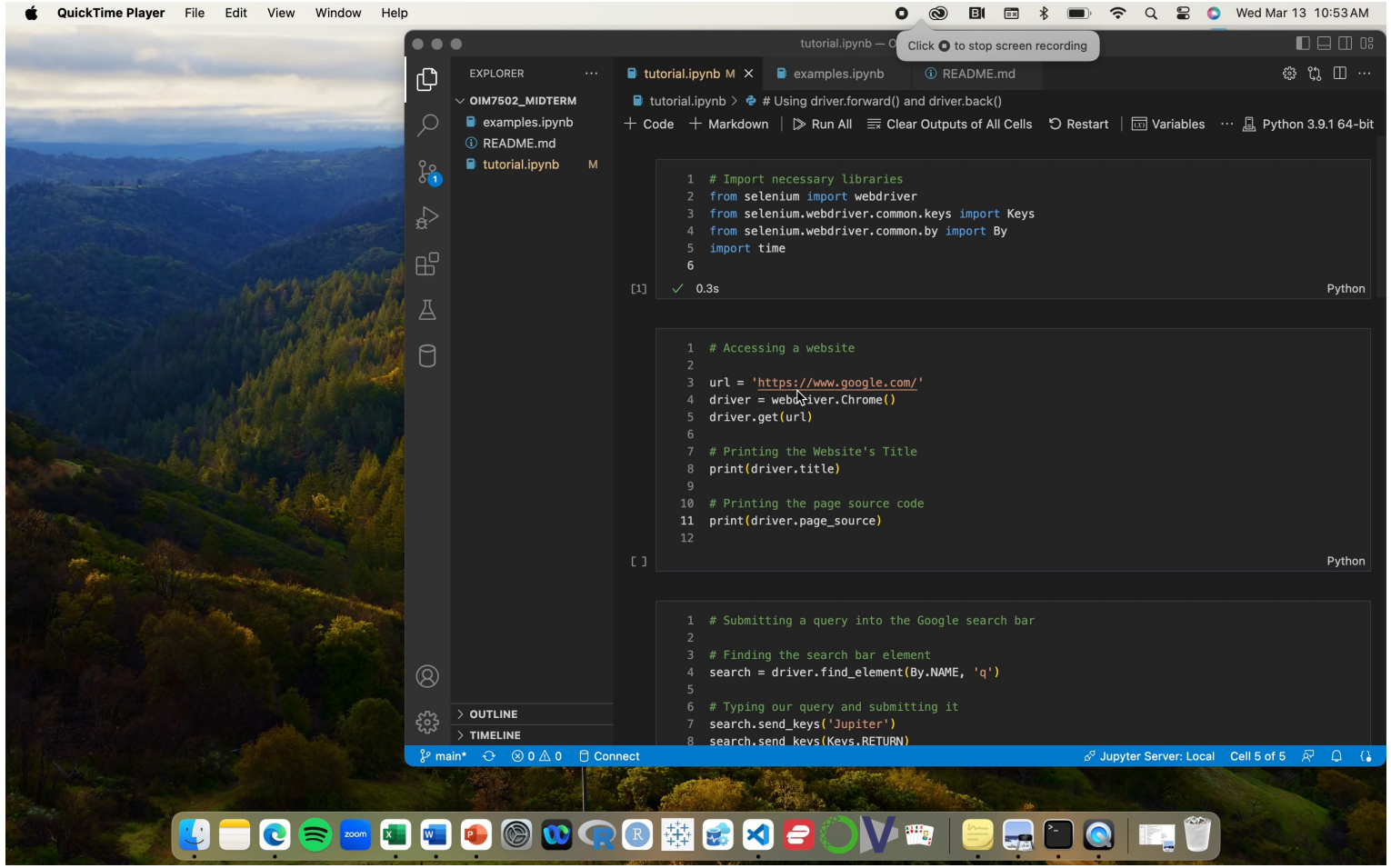
- **Accessing Quick Links in a Website**:

```python
# Accessing Quick Links within website

# Accessing the Images tab on Google
images = driver.find_element(By.LINK_TEXT, 'Images')
images.click()

# Wait for 1.5 seconds
time.sleep(1.5)

# Switching to the books tab on Google
books = driver.find_element(By.LINK_TEXT, 'Books')
books.click()

# Wait for 1.5 seconds
time.sleep(1.5)

# Close the browser
driver.quit()
```

- **Using driver.forward() / driver.back()**

```python
# Using driver.forward() and driver.back()

driver = webdriver.Chrome()

driver.get("http://www.wikipedia.org")
print("Title of the page is:", driver.title)

time.sleep(1.5)

driver.get("http://www.google.com")
print("Title of the page is:", driver.title)

time.sleep(1.5)

# Use the back method to go back to the previous page
driver.back()
print("Title after going back:", driver.title)

time.sleep(1.5)

# Use the forward method to navigate forward in the browser history
driver.forward()
print("Title after going forward:", driver.title)

# Close the browser
driver.quit()
```

PICTURING THE TUTORIAL



SELENIUM FOR WEB SCRAPING

In this example, we are using Selenium to access the Python website, search for a query of our choosing, and then return a list of all the results of the query.

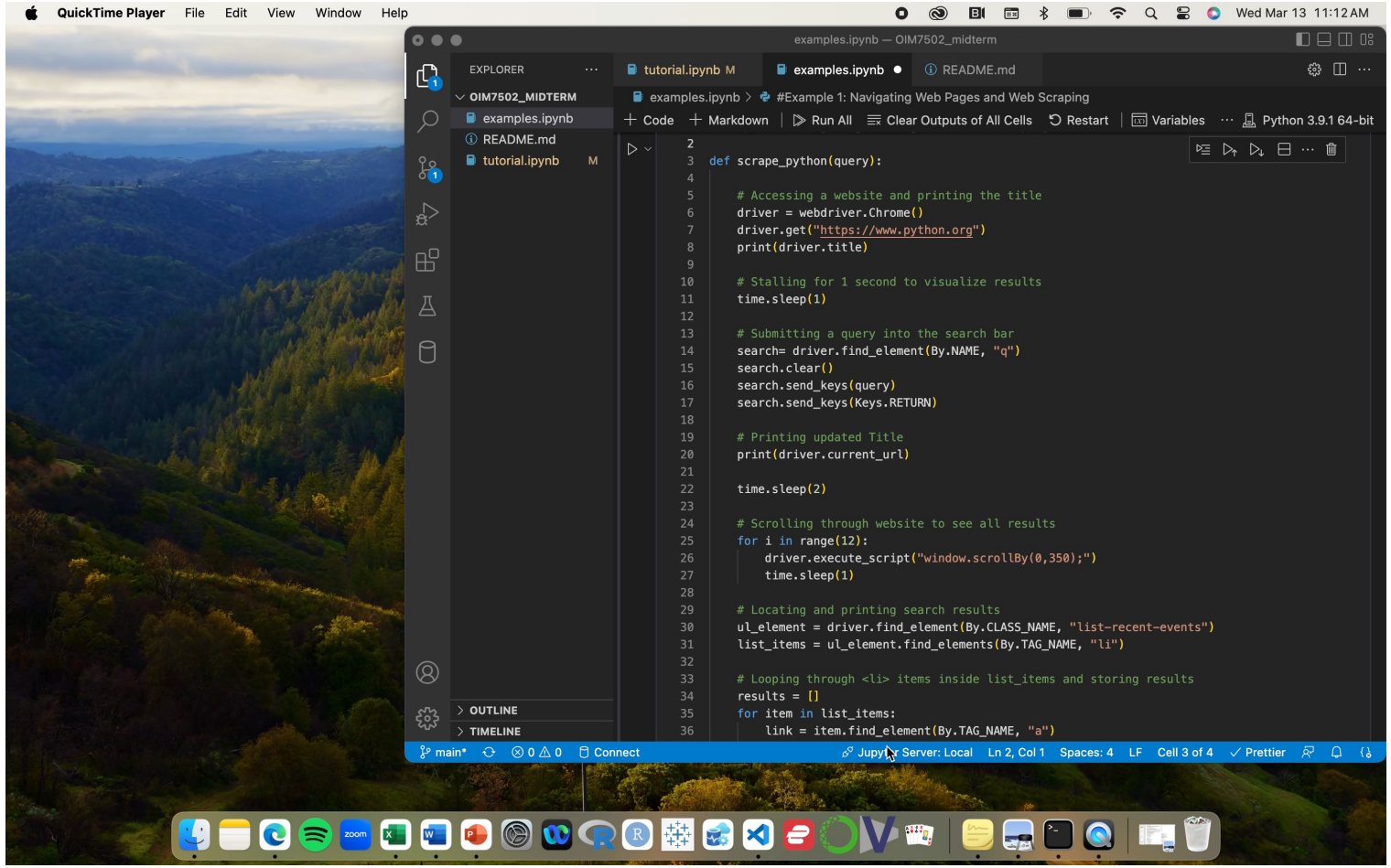
```
['PEP 206 -- Python Advanced Library',
 'Python Software Foundation: Press Release 13-Feb-2003',
 'Python Software Foundation: Press Release 9-Feb-2004',
 'Job - Senior Python Backend Engineer',
 'Mission',
 'Fiscal Sponsorees',
 'PyCon Registration Form',
 'PyCon Registration Form',
 'Python Software Foundation: Press Release 27-Dec-2004',
 'Community Events Manager',
 'Python Software Foundation: Minutes of Board of Directors Meeting (March 11, 2003)',
 'PyCon Home at python.org',
 'Search Python Resources',
 '2021 PSF Annual Report',
 'Applications for Python',
 'Python Software Foundation FAQ',
 'Job - Software Development Engineer',
 'Python Success Stories',
 'Job - Python Sr Dev / TL Dev urgent position',
 'Job - Senior Python Developer']
```

Output for the query 'Advanced Python':

```python
def scrape_python(query):

    # Accessing a website and printing the title
    driver = webdriver.Chrome()
    driver.get("https://www.python.org")
    print(driver.title)

    # Stalling for 1 second to visualize results
    time.sleep(1)

    # Submitting a query into the search bar
    search= driver.find_element(By.NAME, "q")
    search.clear()
    search.send_keys(query)
    search.send_keys(Keys.RETURN)
```

```python
    # Scrolling through website to see all results
    for i in range(12):
        driver.execute_script("window.scrollBy(0,350);")
        time.sleep(1)

    # Locating and printing search results
    ul_element = driver.find_element(By.CLASS_NAME, "list-recent-events")
    list_items = ul_element.find_elements(By.TAG_NAME, "li")

    # Looping through <li> items inside list_items and storing results
    results = []
    for item in list_items:
        link = item.find_element(By.TAG_NAME, "a")
        results.append(link.text)

    time.sleep(2)

    driver.quit()
```

SELENIUM FOR WEB SCRAPING



SELENIUM FOR WEB TESTING

In this example, we are using Selenium and Unittest to test some of the features of the Python website such as:

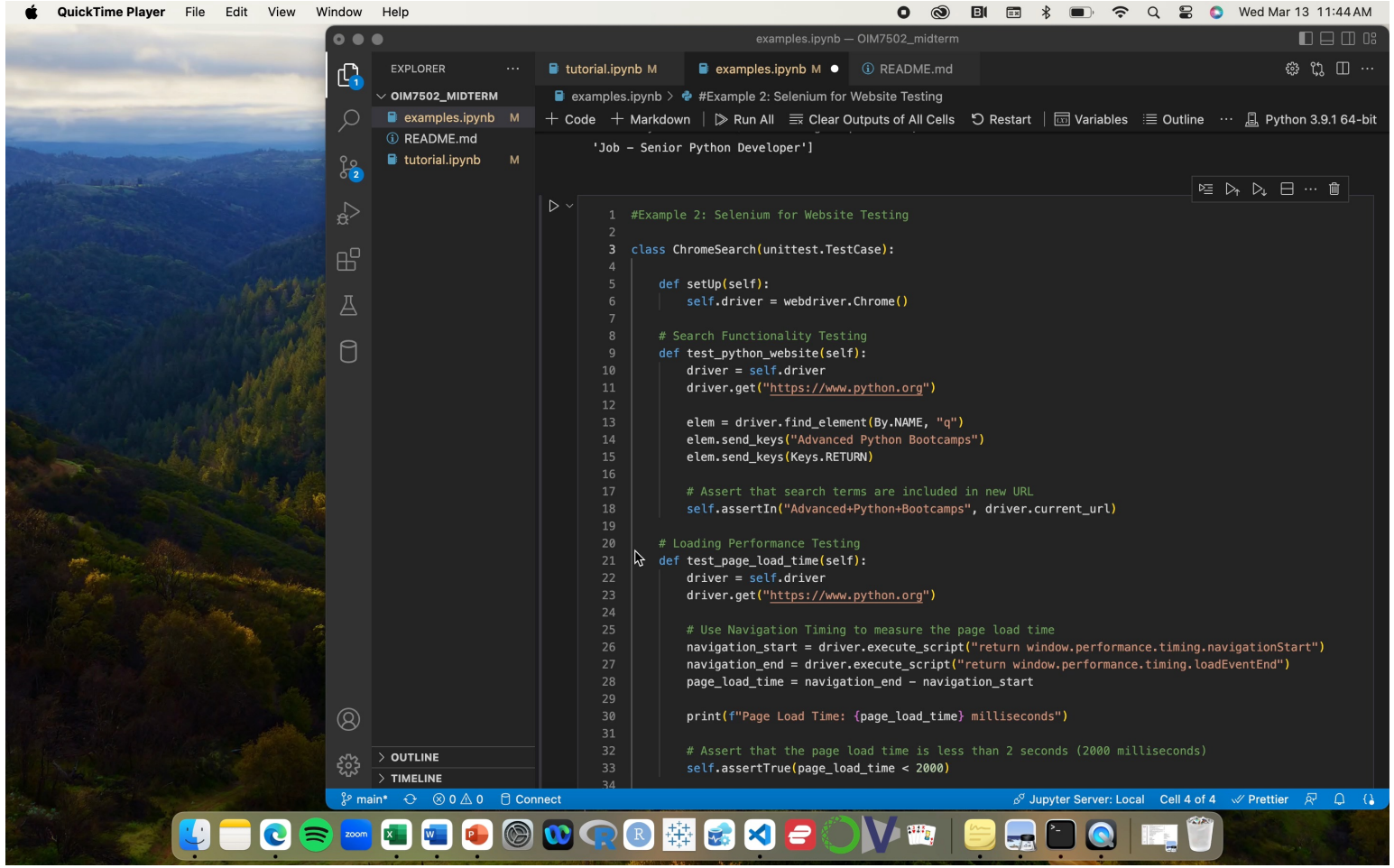
1. Loading Performance Testing
2. Mobile Device Feature Testing.
3. Search Functionality Testing

```python
# Loading Performance Testing
def test_page_load_time(self):
    driver = self.driver
    driver.get("https://www.python.org")

    # Use Navigation Timing to measure the page load time
    navigation_start = driver.execute_script("return window.performance.timing.navigationStart")
    navigation_end = driver.execute_script("return window.performance.timing.loadEventEnd")
    page_load_time = navigation_end - navigation_start

    print(f"Page Load Time: {page_load_time} milliseconds")

    # Assert that the page load time is less than 2 seconds (2000 milliseconds)
    self.assertTrue(page_load_time < 2000)
```

```python
#Design Testing: Mobile Device
def test_menu_visibility_mobile(self):
    driver = self.driver
    driver.get("https://www.python.org")

    # Set window size to simulate a mobile device
    driver.set_window_size(480, 800)

    # Finding mobile menu toggle 'Close' switch
    close_link = driver.find_element(By.ID, "close-python-network")
    close_link.click()

    time.sleep(1)

    # Finding mobile menu toggle 'Open' switch
    open_link = driver.find_element(By.ID, 'python-network')
    open_link.click()

    # Assert both toggle switches are available on the mobile device version
    self.assertTrue(close_link.is_displayed())
    self.assertTrue(open_link.is_displayed())
```

```python
# Search Functionality Testing
def test_python_website(self):
    driver = self.driver
    driver.get("https://www.python.org")

    elem = driver.find_element(By.NAME, "q")
    elem.send_keys("Advanced Python Bootcamps")
    elem.send_keys(Keys.RETURN)

    # Assert that search terms are included in new URL
    self.assertIn("Advanced+Python+Bootcamps", driver.current_url)
```

SELENIUM FOR WEB TESTING



CONCLUSION

- **Versatility**: Selenium's flexibility in languages, browsers, and systems makes it an invaluable tool for web automation and testing.
- **Data Science Applications**: Beyond testing, Selenium is a powerful tool for data collection, automating tasks, and web scraping for data science projects.
- **Community and Resources**: A strong community and a plethora of resources are available to help learn Selenium, from official documentation to community forums and tutorials.
- **Continuous Development**: As an open-source project, Selenium is continuously being improved to meet the evolving needs of web automation and testing.