Index

[**¿Qué es Selenium?** 1](#_Toc54942653)

[**Historia** 2](#_Toc54942654)

[**Origen del nombre** 2](#_Toc54942655)

[**Selenium IDE** 2](#_Toc54942656)

[**Pros** 3](#_Toc54942657)

[**Contras** 3](#_Toc54942658)

[**Selenium RC** 3](#_Toc54942659)

[**Pros** 3](#_Toc54942660)

[**Contras** 3](#_Toc54942661)

[**Selenium Web Driven** 4](#_Toc54942662)

[**Pros** 4](#_Toc54942663)

[**Contra** 4](#_Toc54942664)

[**Selenium Grid** 4](#_Toc54942665)

[**Características** 4](#_Toc54942666)

[**Otras herramientas de testing y automatización** 5](#_Toc54942667)

[**Puppeteer** 5](#_Toc54942668)

[**PROS** 5](#_Toc54942669)

[**CONTRAS** 5](#_Toc54942670)

[**Cypress.io** 5](#_Toc54942671)

[**PROS** 5](#_Toc54942672)

[**CONTRAS** 5](#_Toc54942673)

[**Configurar entorno de trabajo** 6](#_Toc54942674)

[**Unittest (PyTest)** 6](#_Toc54942675)

[**Encontrar elementos con find\_element** 6](#_Toc54942676)

[**¿Como llegar a cada elemento HTML?** 7](#_Toc54942677)

[**Preparar assertions y test suites** 8](#_Toc54942678)

[**assertions** 8](#_Toc54942679)

[**Test suites** 8](#_Toc54942680)

[**Entender las clases WebDriver y WebElement** 9](#_Toc54942681)

[**Clase WebDriver** 9](#_Toc54942682)

[**Propiedades de la clase WebDriver** 9](#_Toc54942683)

[**Clase WebElement** 10](#_Toc54942684)

[**Propiedades más comunes de la clase WebElement** 10](#_Toc54942685)

[**Métodos más comunes de la clase WebElement** 10](#_Toc54942686)

[**Demora implícita y explícita** 11](#_Toc54942687)

[**Data Driven Testing (DDT)** 11](#_Toc54942688)

[**¿TDD o DDT?** 11](#_Toc54942689)

[**Page Object Model (POM)** 12](#_Toc54942690)

[**Beneficios** 12](#_Toc54942691)

# **¿Qué es Selenium?**

Es una **SUIT de herramientas** para la automatización de navegadores Web.

El objetivo de Selenium NO fue para el Testing ni para el Web Scraping (aunque se puede usar para eso), por lo tanto, no es el más óptimo para estas actividades.

**Protocolo**: WebDriver, herramienta que se conecta a un API.

Selenium WebDriver es la herramienta que utilizaremos en el curso.

* Selenium **NO** es un Software, **ES** una **SUIT de Softwares**.
* DDT: **D**ata **D**rive **T**esting: Ingresar datos para que realice varias pruebas (sin intervención humana).



## **Historia**

Todo comienza en 2004, **Jason Huggins** buscaba automatizar pruebas manuales, creando así una aplicación llamada “JavaScriptTestRunner” que posteriormente fue llamada **Selenium Core**.

* **Paul Hammant** vio el demo y busco una solución a la “Same Origin Policy” y se creó **Selenium RC**.

**Más fácil y mejor**

* **Shinya Katasani** en Japón envolvió el código de Selenium convirtiéndolo en un plugins para Firefox.
* **Selenium IDE** es capaz de grabar, repetir, importar y exportar automatización.

**Un cliente para todos**

* **Simon Stewart** trabajo en varias herramientas llamadas **WebDriver**. Reemplazando JS por un cliente para cada navegador y una API de alto nivel.
* **Selenium RC** se fusiono a este proyecto, dando lugar a **Selenium WebDriver**.

## **Origen del nombre**

En su momento, la compañía de desarrollo que creo Selenium originalmente, tenían competidores, uno de ellos era Mercurial. Un dato curioso es que cuando una persona padece de mercurio debe consumir suplementos a base de selenio para poderse curar, entonces el equipo de desarrollo considero que era una buena idea utilizar el nombre de Selenium porque ellos ofrecían una “cura” para lo que hacían u ofrecían sus competidores. Esto era muy cierto ya que Selenium era un producto con capacidades superiores a lo que hacia la competencia.

## **Selenium IDE**

* No es un lenguaje, sólo funciona con clics.
* Ofrece reportes de las pruebas.

## **Pros**

* Excelente para iniciar en el mundo del testing.
* No requiere saber programar.
* Exporta scripts para Selenium RC y Selenium WebDriver.
* Genera reportes.

## **Contras**

* Disponible para Google Chrome y Firefox.
* No soporta DDT. No permite colocar datos para múltiples pruebas.

## **Selenium RC**

### **Pros**

* Soporte para.
  + Varias plataformas, navegadores y lenguajes.
  + Operaciones lógicas y condicionales.
  + DDT.
* Posee una API madura.

### **Contras**

* Complejo de instalar.
* Necesita de un servidor corriendo.
* Comandos redundantes en una API.
* Navegación no tan realista.

## **Selenium Web Driven**

### **Pros**

* Soporte para múltiples lenguajes.
* Fácil de instalar.
* Comunicación directa con el navegador.
* Interacción más realista.

### **Contra**

* No soporta nuevos navegadores tan rápido.
* No genera reportes o resultados de pruebas.
* Requiere de saber programar.

## **Selenium Grid**

### **Características**

* Se utiliza junto a Selenium RC.
* Permite correr pruebas en paralelo.
* Conveniente para ahorrar tiempo.

# **Otras herramientas de testing y automatización**

## [**Puppeteer**](https://pptr.dev/)

### **PROS**

* Soporte por parte de Google, te brinda datos del Performance Analysis de Chrome y un mayor control de este navegador. No requiere archivos externos como lo hace Selenium con WebDriver.

### **CONTRAS**

* Solo funciona para Google Chrome con JavaScript, tiene una comunidad pequeña así que el apoyo será poco.

## [**Cypress.io**](https://www.cypress.io/)

### **PROS**

* Tiene una comunidad emergente y va creciendo a pasos acelerados, tiene muy buena documentación para implementar Cypress en los proyectos. Es muy ágil en pruebas E2E, está orientado a desarrolladores y tiene un excelente manejo del asincronismo, logrando que las esperas sean dinámicas y también se puedan manejar fácilmente.

### **CONTRAS**

* Solo funciona en Google Chrome con JavaScript, se pueden realizar pruebas en paralelo únicamente en la versión de pago.

# **Configurar entorno de trabajo**

1. Descargamos Python desde el [sitio oficial](https://www.python.org/downloads/).
2. Se necesita tener python 3.6 en adelante.
3. Instalamos Selenium con el comando **pip3 install selenium** en el command prompt (admin) de windows.
4. Instalamos la librería pyunitreport que nos permitirá obtener reportes en HTML, el comando es **pip3 install pyunitreport**

# **Unittest (PyTest)**

* **Text Fixture**: Es todo lo que va ocurrir **antes** y **después** de la automatización.
* **Text Case**: Unidad de código que le indicaremos a Selenium que es lo que queremos que haga.
* **Test Suite**: Colección de **Tests Cases** en un sólo archivo para que podamos ejecutar todos de forma secuencial.
* **Test Runner**: Orquestador de la ejecución. Es decir, es la parte encargada de que se va a ejecutar, en que orden, como y los resultados que nos arrojará.
* **Test Report**: Resumen de resultados.

# **Encontrar elementos con find\_element**

En ocasiones algunos sitios pueden tener bloqueos regionales o no estar disponibles por la alta cantidad de solicitudes que llegan a tener.

Si el sitio de práctica no abre, puedes intentar ingresando a [OneStepCheckout Responsive Demo](http://demo.onestepcheckout.com/).



## **¿Como llegar a cada elemento HTML?**

Podremos llegar a cada uno de ellos a través de los Selectores.

* **ID**
* **Nombre del atributo**.
* **Nombre de la clase**.
* **Nombre de la etiqueta**.
* **XPath**, es una ruta de nodos en el XML que nos indica la ubicación exacta de donde se encuentra un elemento.
* **Selector de CSS**.
* **Texto del** **link**.
* **Texto parcial del link**

# **Preparar** **assertions y test suites**

## **assertions**

Métodos que permiten validar un valor esperado en la ejecución del test. Si el resultado es verdadero, el test continua, en caso contrario “falla” y termina.

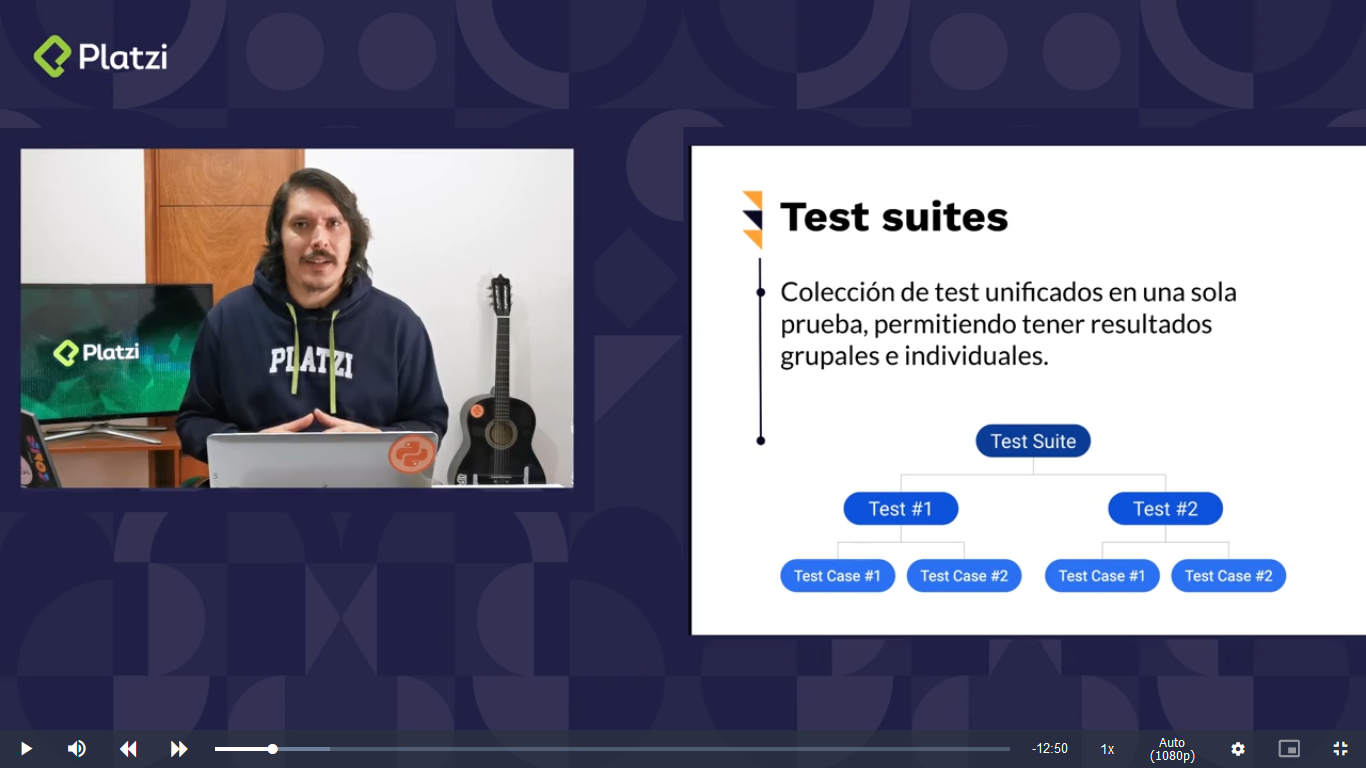
**Ejemplo**

* assertEqual(Price.text, “300”)

Lo que estamos validando en el ejemplo anterior es que **el texto del elemento que se llama ‘price’ sea igual a 300 como un string, no como un número**.

## **Test suites**

Son una colección de test unificados en una sola prueba, permitiendo tener resultados grupales e individuales.



# **Entender las clases WebDriver y WebElement**

Como viste en clases anteriores, un sitio web se construye por código HTML en forma de árbol, conteniendo distintos elementos con los que podemos interactuar según estén presentes o no en nuestra interfaz gráfica.

**Selenium WebDriver** nos brinda la posibilidad de poder referirnos a estos elementos y ejecutar métodos específicos para realizar las mismas acciones que un humano haría sobre los mismos, gracias a las clases **WebDriver** y **WebElement**.

## **Clase WebDriver**

Cuenta con una serie de propiedades y métodos para interactuar directamente con la ventana del navegador y sus elementos relacionados, como son pop-ups o alerts. Por ahora nos centraremos a las más utilizadas.

## **Propiedades de la clase WebDriver**

Estas son las más comunes para acceder al navegador.

|  |  |  |
| --- | --- | --- |
| Propiedad – Atributos | Descripción | Ejemplo |
| current\_url | Obtiene la URL del sitio en la que se encuentra el navegador. | driver.get\_url |
| current\_window\_handle | Obtiene la referencia que identifica a la ventana activa en ese momento. | driver.current\_window\_handle |
| name | Obtiene el nombre del navegador subyacente para la instancia activa. | driver.name |
| orientation | Obtiene la orientación actual del dispositivo móvil. | driver.orientation |
| page\_source | Obtiene el código fuente de disponible del sitio web. | driver.page\_source |
| title | Obtiene el valor de la etiqueta <**title**> del sitio web. | driver.title |

## **Clase WebElement**

Esta clase nos permite interactuar específicamente con elementos de los sitios web como textbox, text area, button, radio button, checkbox, etc.

## **Propiedades más comunes de la clase WebElement**

|  |  |  |
| --- | --- | --- |
| Propiedad – Atributo | Descripción | Ejemplo |
| size | Obtiene el tamaño del elemento. | login.size |
| tag\_name | Obtiene el nombre de la etiqueta HTML del elemento. | login.tag\_name |
| text | Obtiene el texto del elemento. | login.text |

## **Métodos más comunes de la clase WebElement**

|  |  |  |
| --- | --- | --- |
| Método/Atributo | Descripción | Ejemplo |
| clear() | Limpia el contenido de un textarea. | first\_name.clear() |
| click() | Hace clic en el elemento. | send\_button.click() |
| get\_attribute(name) | Obtiene el valor del atributo de un elemento. | submit\_button.get\_attribute(‘value’) last\_name.get\_attribute(max\_length) |
| is\_displayed() | Verifica si el elemento está a la vista al usuario. | banner.is\_displayed() |
| is\_enabled() | Verifica si el elemento está habilitado. | radio\_button.is\_enabled() |
| is\_selected() | Verifica si el elemento está seleccionado, para el caso de checkbox o radio button. | checkbox.is\_selected() |
| send\_keys(value) | Simula escribir o presionar teclas en un elemento. | email\_field.send\_keys(‘team@platzi.com’) |
| submit() | Envía un formulario o confirmación en un text area. | search\_field.submit() |
| value\_of\_css\_property(property\_name) | Obtiene el valor de una propiedad CSS del elemento. | header.value\_of\_css\_property(‘background-color’) |

# **Demora implícita y explícita**

* **Implícita**: Busca una o varios elementos en el DOM si no se encuentran disponibles por la cantidad de tiempo asignado.
* **Explicita**: Utiliza condiciones de espera determinadas y continúan hasta que se cumplan.

# **Data Driven Testing (DDT)**

## **¿TDD o DDT?**

* **Test Driven Development**: Desarrollar código en base a pruebas para que puedan cumplirlas.
* **Data Driven Testing**: Desarrollar pruebas en base a código ya existente para validar en que escenarios pasan o fallan.



# **Page Object Model (POM)**

Es un patrón de diseño utilizado en testing, la forma en que funciona es que, en lugar de tener las pruebas en un solo archivo, las tendremos en archivos independientes los cuales se llamaran Pages haciendo referencia al sitio adonde se aplican.

## **Beneficios**

* Crea un alto nivel de abstracción para minimizar cambios en las pruebas si los desarrolladores modifican el sitio.
* Crea código reutilizable que se puede utilizar en multiples pruebas.
* Las pruebas son más legibles, flexibles y vigentes.

