**Tipo de artículo: Artículo original**

# Desarrollo de pruebas funcionales con Selenium WebDriver y Python

# *Development of functional tests with Selenium WebDriver and Python*

**Maydalis Hernández Pérez**[1*], https://orcid.org/0000-0002-5598-6212
**Luis Angel Llull Céspedes** [2], https://orcid.org/0000-0002-3553-4593

[1] Departamento de Informática, Facultad 4, Universidad de las Ciencias Informáticas. mhernandezp@uci.cu
[2] Departamento de Matemática, Facultad FTE, Universidad de las Ciencias Informáticas. lallull@uci.cu

*** Autor para correspondencia: mhernandezp@uci.cu**

**Resumen**

Las pruebas manuales de software consumen mucho tiempo y recursos estando sujetas a errores humanos. Para solventar estos problemas existen las pruebas automatizadas. Para aplicar este tipo de prueba existen varias herramientas entre las que se encuentra Selenium WebDriver una herramienta de código abierto que su mayor uso está dado por su utilización en el desarrollo de pruebas funcionales. Esta herramienta brinda una interfaz de programación de aplicaciones para trabajar con el lenguaje de programación Python. Por lo que, en este trabajó se documentó el desarrollo de este tipo de prueba utilizando las tecnologías mencionadas con anterioridad y empleando como buena práctica el patrón Page Object Model. Con este trabajo finalmente se demuestra que las pruebas automatizadas tienen un tiempo de ejecución menor que las pruebas manuales y que estas además se ejecutan de forma precisa, lo que ofrece un resultado más preciso y fiable.

**Palabras clave**: Page Object Model, pruebas automatizadas**,** pruebas funcionales, pruebas manuales, Selenium WebDriver

***Abstract***
*Manual software testing is time consuming and resource intensive and subject to human error. To solve these problems there are automated tests. To apply this type of test, there are several tools, among which is Selenium WebDriver, an open source tool whose greatest use is given by its use in the development of functional tests. This tool provides an application programming interface for working with the Python programming language. Therefore, in this work the development of this type of test was documented using the technologies mentioned above and using the Page Object Model pattern as a good practice. With this work, it is finally shown that automated tests have a shorter execution time than manual tests and that they are also executed accurately, which offers a more precise and reliable result.*

***Keywords:*** *Page Object Model, automated tests, functional tests, manual tests, Selenium WebDriver*

**Recibido: 10/01/2022**
**Aceptado: 28/04/2022**

## Introducción

En el desarrollo de software se cometen errores, para solucionarlos a tiempo y no afecten la calidad del producto es necesario realizar pruebas. Las pruebas según Somerville "intentan demostrar que un programa hace lo que se intenta que haga, así como descubrir defectos en el programa antes de usarlo. Al probar el software, se ejecuta un programa con datos artificiales. Hay que verificar los resultados de la prueba que se opera para buscar errores, anomalías o información de atributos no funcionales del programa" (Sommerville, 2011). Cuando las pruebas son aplicadas adecuadamente pueden garantizar que el software y los procesos en el ciclo de vida se ajustan a las necesidades específicas (Feldman, 2005) por lo que; debe ser un procedimiento involucrado en todas las fases del ciclo de vida (Serna y otros, 2011). Para garantizar su éxito se debe definir una estrategia de prueba en la que se definan actividades de: planeación de las pruebas, diseño de casos de prueba, ejecución y evaluación de las pruebas.

En el caso específico de la ejecución de las pruebas debe iniciar con la creación de los datos de prueba para ejecutar los casos de prueba diseñados. La ejecución de estos casos, puede realizarse de manera manual o automatizada.

En la ejecución de las pruebas manuales todo el proceso requiere un probador que interactúe con cada caso de prueba para analizarlo y reportar los resultados. Estas pruebas son muy utilizadas en los proyectos, pero hay que tener presente que hoy en día los sistemas cada vez son más complejos y tienen mayor cantidad de requisitos. Esto implica que sea mucho mayor el número de casos de prueba a ejecutar, lo que el tiempo planificado para la ejecución debe ser mayor. Otro elemento, de importancia es que cuando se aplica Pruebas de Confirmación y Regresión es necesario volver a repetir pruebas lo que implica tiempo, que se puede utilizar en el desarrollo de otras actividades con mayor complejidad. Además, las pruebas manuales son más propensas a errores y la creación de software preciso y confiable es un problema abierto (Jha y otros, 2022).

Para solventar estas deficiencias existen las pruebas automatizadas. Este método implica el uso de herramientas de software especiales para controlar el desempeño de las pruebas. Entonces los resultados reales de las pruebas se comparan con los previstos. La mayoría de las operaciones se realizan automáticamente, con poca o ninguna intervención por parte del ingeniero de pruebas. Los probadores escriben scripts (los llamados casos de prueba automatizados) que tienen un conjunto de acciones y comprobaciones. Las pruebas escritas automatizadas pueden tener muchos beneficios y pueden ser muy útiles para el proyecto por ejemplo: mejora la calidad del producto, en términos de un menor número de defectos presentes en el ciclo de vida, incrementa y mejora la cobertura de la prueba y del código, hay mayor cantidad de valores de entrada probados, reduce el tiempo de la prueba, es decir, ofrece la capacidad de ejecutar más pruebas en un lapso menor, reutilizar casos de prueba, porque se diseñan con el mantenimiento en mente, reduce el esfuerzo humano, lo que se puede aprovechar en otras actividades y disminuye

costos, porque la automatización no requiere mucha intervención humana (Serna, 2019). Para el desarrollo de estas existen herramientas entre las que se encuentra Selenium WebDriver (Muthukadan, 2022). Selenium WebDriver es la habilidad de prueba más buscada en términos del número de anuncios de empleo en algunos países como Australia (Zhan, 2021).

Es utilizada para automatizar los navegadores web en muchas plataformas *y* permite simular un usuario que usa un navegador web. Al igual que un usuario, puede abrir un sitio web, hacer clic en enlaces, completar formularios y navegar alrededor. También puede examinar la página, mirar elementos en ella y tomar decisiones basado en lo que se ve. Este es utilizado para realizar pruebas automatizadas ya que, permite simular el comportamiento de una página web cuando se realiza una acción determinada. Además, otro de los beneficios que tiene es que WebDriver facilita las pruebas en diferentes sistemas operativos y diferentes configuraciones de navegador. Para su utilización están disponibles diferentes API para su uso con otros lenguajes como Java y Python.

En este trabajo se documenta como se debe desarrollar pruebas funcionales automáticas mediante la API de Selenium WebDriver para Python. Además, se propone como buena práctica la utilización del patrón Page Object Model el cual encapsula el comportamiento de la página web sin tener que lidiar con él código HTML en el desarrollo de los test. El desarrollo de este trabajo permitirá demostrar que realizando pruebas automatizadas el tiempo de ejecución de las pruebas es mucho menor que si se realizaran de forma manual, demostrando así las ventajas mencionadas con anterioridad.

## Materiales y métodos

Se realizó una revisión bibliográfica en artículos científicos y sitios web oficiales para identificar cuáles son las ventajas de realizar pruebas automatizadas en los proyectos. Además, se consultó información oficial de las tecnologías empleadas en el desarrollo de la propuesta. Algunos de los criterios de búsquedas utilizados para el desarrollo de la investigación fueron las siguientes: automatic software tests, manual software testing, software testing, advantages of automatic tests, Selenium WebDriver, Page Object Model y Python unittest library.

### Preparación del entorno de prueba

Para utilizar la API de Selenium WebDriver para Python lo primero que se requiere es tener instalado previamente este lenguaje. Se seleccionó este lenguaje porque a diferencias de los programas realizados en Java, los programas de Python se ejecutan más rápido. Por otro lado, cuenta con la biblioteca unittest que permite escribir casos de prueba y cuenta con funciones que posibilitan comprobar si las funcionalidades de un sistema se ejecutan de forma correcta. Además, en términos generales, se define que Python, en comparación con Java, es más simple y compacto. En este trabajo se utilizó la versión 3.8.1 de Python y como IDE de desarrollo el Pycharm en su versión 2019.3.1.

Hay que tener presente que Selenium no es una biblioteca estándar de Python y por lo tanto debe ser instalada, para esto se debe de utilizar el comando pip install selenium. Otro elemento importante es que hay que definir en qué navegador se va a realizar las pruebas, ya que; dependiendo de esto hay que descargar el controlador para este navegador. El controlador de un navegador web permite el enlace entre este y los test en Selenium. En este trabajo se utiliza Firefox y para descargar su controlador en este caso, geckodriver se puede acceder al enlace: https://github.com/mozilla/geckodriver/releases/. Una vez descargado el archivo se descomprime y el fichero que tiene como nombre geckodriver.exe hay que enviarlo al directorio de instalación de Python. Con estos pasos ya está listo el entorno para realizar las pruebas.

## Resultados y discusión

Para explicar cómo se realizan las pruebas automatizadas utilizando la API de Selenium WebDriver para Python el desarrollo de este trabajo este compuesto por las siguientes secciones:

- ✓ Patrón Page Object Model para el desarrollo de pruebas automatizadas.
- ✓ Estudio del sistema Compute-Database para la definición del caso de prueba automatizar.
- ✓ Desarrollo de un caso de prueba y de una suite de prueba de forma automática.
- ✓ Análisis de los resultados obtenidos.

### Patrón Page Object Model para el desarrollo de pruebas automatizadas

Cuando se diseña una prueba automatizada para un sitio web, se consultan sus elementos para escribir en un campo de texto, hacer clic en enlaces, botones, y determinar qué se muestra a continuación. Hay que tener en cuenta que estos elementos pueden sufrir cambios posteriores en su implementación, lo que trae consigo que si estos se manipulan directamente en los test de prueba estos requerirán un alto grado de mantenimiento. Para solucionar este problema existe un patrón que se denomina Page Object Model (Nishmitha y otros, 2016).

Este patrón es un modelo de diseño para construir la automatización de pruebas de Selenium el cual encapsula el comportamiento de las páginas web, o una parte de ella. Esto permite escribir tests y manipular elementos de las páginas sin tener que lidiar con el HTML. Entre las ventajas de utilización de este patrón se encuentran: fácil de mantener, fácil lectura de los test de pruebas, elimina la redundancia ya que; no existe duplicidad de funciones y se puede reutilizar código (funciones). Para desarrollar la automatización de las pruebas se utilizó este patrón como buena práctica el cual será explicado en este trabajo (Nishmitha y otros, 2016).

### Estudio del sistema Compute-Database para la definición del caso de prueba automatizar

Para desarrollar este trabajo se buscó en Internet páginas web de pruebas que estuvieran disponibles para realizar pruebas funcionales. En una búsqueda, se encontró una página en la que se hace referencia a varios sitios que se

pueden utilizar para realizar pruebas esta página es:https://cl.abstracta.us/blog/mejores-sitios-web-prueba-practicar-tests/. De este sitio se seleccionó la página web que tiene como nombre Computer-Database, un sitio de prueba que cuenta con una base de datos informática, donde hay una lista con varias columnas y un filtro de búsqueda.

Luego de realizar una revisión al sistema se pudo identificar que tiene varios requisitos funcionales los cuales pueden ser probados como: Adicionar computadora, Modificar datos de una Computadora, Listar Computadoras y Eliminar Computadora. Una vez que se estudió el funcionamiento de cada uno de los requisitos, se definió que para este trabajo solo se automatizará un caso de prueba que estará en correspondencia con los requisitos funcionales Adicionar Computadora y Modificar datos de la computadora.

Hay que tener presente que en un caso de prueba se puede comprobar varios escenarios, es decir, puede estar compuesto por varios test de pruebas, por ejemplo: en el caso del Adicionar Computadora se pueden identificar varios escenarios, es decir cuándo:

- ✓ Se introduce y/o selecciona los datos de la computadora y selecciona la opción Create this computer.
- ✓ Se selecciona la opción Cancelar.
- ✓ Existen datos incompletos.
- ✓ Existen datos incorrectos.

Es decir, se realizó un caso de prueba correspondiente al requisito Adicionar Computadora, en el que solo se creó los test para los escenarios 1, 2 y 3 y se decidió incluir otro test de prueba para cuando se Modifique los datos de una computadora y se seleccione la opción de Update this computer estos se modifiquen de forma correcta. Esto se realizó así para que en el trabajo se reflejará la mayor cantidad de elementos de Selenium y no fuera demasiado largo. Sin embargo, se recomienda que se defina un caso de prueba para cuando se Modifiquen los datos de una computadora y en este se reflejen los diferentes test de pruebas que estarían en correspondencia con los diferentes flujos que pueden ocurrir en este requisito.

## Desarrollo del caso de prueba y de una suite de prueba

Primeramente, se deben de crear dos archivos con extensión .py. El primero será una clase donde se define el comportamiento de la página web que será probada, consultando sus componentes para hacer una acción específica con ellos. Hay que tener en cuenta que cada vez que se cree un archivo con esta finalidad el nombre que se le pone debe empezar con Page. Mientras que el otro archivo es de igual forma una clase que representa el caso de prueba y en este se definen los test de pruebas. Es decir, en este se crean instancias del archivo anteriormente creado para poder utilizar las funciones que se definan en los Pages Object. En este caso los archivos se denominan Page_PC.py y Test_PC.py respectivamente.

Cuando se defina un caso de prueba al estar compuesto por varios test, cada test definido va a tener dos funciones principales. Una función va a estar declarada en la clase Page_PC.py y es donde se define el comportamiento de este requisito. Es decir, en esta función se les indica las instrucciones necesarias para que se simule la ejecución de un requisito determinado en el navegador. Mientras que la otra función se declara en la clase Test_PC.py que es la encargada de comprobar si ese escenario se ejecuta de forma correcta o lanza un error. A continuación, se explican cuáles fueron los pasos realizados para poder probar los escenarios siguientes:

- ✓ Introduce y/o selecciona los datos de una computadora y selecciona la opción Create this computer.
- ✓ Se modifican los datos de una computadora y se selecciona la opción Update this computer.

Escenario introduce y/o selecciona los datos de una computadora y selecciona la opción Create this computer

En este caso lo primero es que se importan las siguientes bibliotecas:

```
from selenium import webdriver
from selenium.webdriver.support.ui import Select
from selenium.webdriver.common.by import By
```

**Figura 1**. Bibliotecas utilizadas en la clase Page_PC.py

- ✓ En este caso la biblioteca Select tiene funcionalidades que permiten utilizar funciones que para seleccionar un valor de una lista desplegable.
- ✓ WebDriver es la interfaz básica para simular las interacciones del usuario con cualquier navegador, ya sea Firefox, Chrome, Edge, Safari o Internet Explorer.
- ✓ Mientras que la biblioteca By se utiliza para localizar diferentes elementos de las páginas web como botones, campos de textos entre otros. Esta biblioteca te permite buscar por los siguientes localizadores: ID, NAME, Link Text, CSS Selector, XPath. La elección de la localización depende en gran medida de la página a probar.

En la declaración de la clase se evidencia que en el método inicialización es necesario recibir como parámetro una variable denominada driver, que representa una instancia del navegador que se va a utilizar para desarrollar los test. Posteriormente en el mismo método se declaran y se inicializan los atributos. En este caso los atributos de esta clase están en correspondencia con los elementos que se deben localizar en la página web para poder crear los test de pruebas. Por ejemplo, para este escenario los atributos fueron los siguientes:

```
def __init__(self, driver):
    self.driver = driver
    self.name = 'name'
    self.introduced = 'introduced'
    self.discontinuado = 'discontinued'
    self.select_unidad = 'company'
    self.botton_adicionar='add'
    self.botton_create = '/html/body/section/form/div/input'
```

**Figura 2.** Función de inicialización de la clase Page_PC.py**.**

En el caso del primero se define un atributo driver que será igual al driver (instancia del navegador) que le pasan como parámetro a la función de inicialización. En el caso de los demás, si se abre la página a probar se identifica que para poder adicionar una nueva computadora hay que seleccionar la opción Add a new computer**.** Una vez selecciona la opción se abre un formulario que te permite añadir una nueva computadora como se evidencia a continuación:

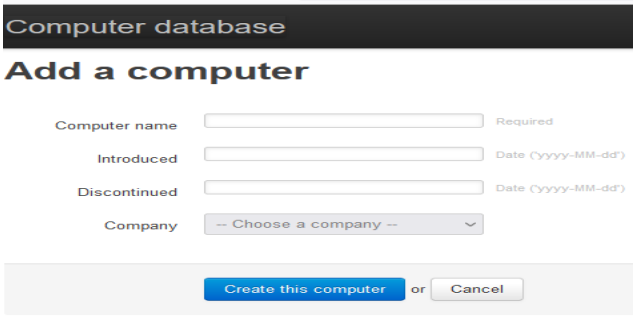


**Figura 3.**Formulario del sistema para Adicionar una nueva computadora.

Por lo tanto, en total se deben de localizar 6 elementos dos botones, un botón que se selecciona para acceder al formulario y el botón que se encuentra en la página del formulario que te permite crea una nueva computadora en el base de datos. Además, tres campos de textos (Computer name, Introduced, Discontinued) y un campo de selección (Company). Por lo tanto, hay 6 atributos que representan esos componentes.

Entonces los valores con los que se inicializa están en correspondencia al localizador que se va a utilizar, por ejemplo: en el caso de los campos Computer name, Introduced, Discontinued, Company y el botón de Add computer van a ser localizados a través de su ID y por tanto el valor inicial del atributo será igual al valor que tiene ese localizador en el código del elemento.

Por ejemplo, si se quiere localizar el campo de texto que corresponde al campo Computer name, se debe dar clic derecho en el componente y seleccionar la opción Inspeccionar. Una vez que se seleccione se abrirá una pestaña donde se podrá ver localizado este componente en el código HTML (por sus siglas en inglés: HyperText Markup Language) como se muestra a continuación:

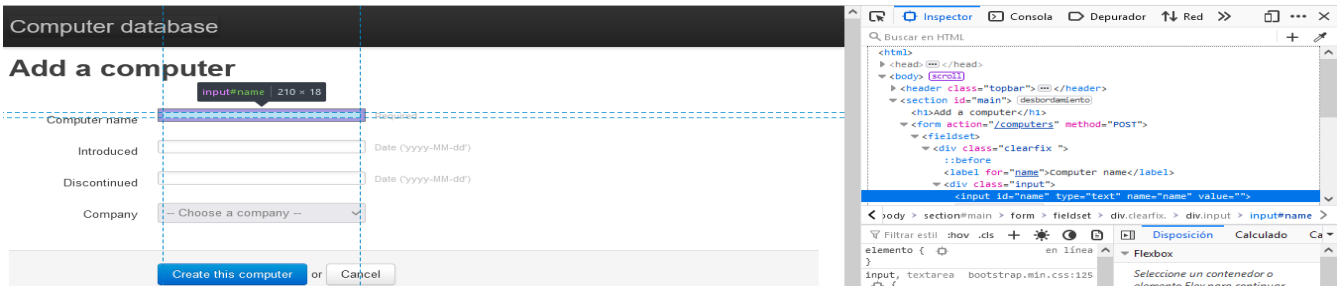


**Figura 4.** Localizando el campo de texto Computer name.

Una vez que se tiene localizado se pueden visualizar algunas propiedades como: el id, name, class entre otros. En este caso cuando se observa el elemento en el HTML y al estar definido como programador que al localizar este elemento será por su ID se selecciona el id que tiene asignado este componente y con ese valor se inicializará el atributo que está en correspondencia con el componente.

Hay que tener en cuenta que en algunas ocasiones no se cuentan con todas estas propiedades o por ejemplo varios componentes tienen un mismo name y hay que acudir a otros tipos de localizadores para poder acceder de forma correcta. Por ejemplo, el localizador XPath permite acceder a casi cualquier elemento, incluso aquellos sin atributos de clase, nombre o ID ya que; es el idioma utilizado para localizar nodos XML (por sus siglas en inglés: Extensible Markup Language). Es decir, cómo se puede pensar en HTML como una implementación de XML, también se puede utilizar XPath en la localización de elementos HTML.

En este caso se utilizó este localizador para localizar el botón de Create this computer**.** Para poder acceder a un XPath se da clic derecho en el componente, se selecciona la opción Inspeccionar y una vez localizado el elemento en el HTML se da clic derecho y se selecciona la opción Copiar y de esta la opción XPath y de esta forma se tendrá el XPath del componente. A continuación, se muestra una imagen con los pasos realizados:


**Figura 5**. Localizando el XPath del botón Create this Computer.

Un elemento importante a tener presente es que por cada una función que se vaya a realizar en las clases PAGE no hay que hacer una función de inicialización, solo que se está exponiendo paso por paso lo realizado para este escenario.

Una vez que se encuentren inicializados los componentes entonces se podrán crear las respectivas funciones que se van a utilizar posteriormente en cada uno de los test de prueba. Enfatizar que este patrón es una forma de mejorar el código, que te permite una mejor comprensión de las pruebas, pero no quiere decir que en los mismos test de pruebas o en las funciones propias de las clases Page Object no se puedan localizar elementos nuevos que no se tuvieron en cuenta en los atributos. A continuación, se define el procedimiento en el que se accede a los componentes y se define el comportamiento de este escenario:

```
def Adicionar(self):
    self.driver.find_element(By.ID, self.botton_adicionar).click()
    self.driver.find_element(By.ID, self.name).send_keys("Asus")
    self.driver.find_element(By.ID, self.introduced).send_keys("2017-07-04")
    self.driver.find_element(By.ID, self.discontinuado).send_keys("2020-07-03")
    company= Select(self.driver.find_element(By.ID, self.select_unidad))
    company.select_by_value("36")
    self.driver.find_element(By.XPATH, self.botton_create).submit()
```

**Figura 6.**Función que define el comportamiento de la página web cuando se Adiciona una computadora.

En este caso se está localizando en primer lugar al botón Add a new Computer. Para esto se utiliza la función fin_element (parámetro uno, parámetro dos) del navegador, recordar que en este caso el atributo driver es una instancia del navegador que se utilizará. Esta función recibe dos parámetros, en el primero se define el tipo de localizador a utilizar para localizar los elementos. Para esto se utiliza la biblioteca By indicando que se buscará por ID y en el segundo parámetro sé indica que identificador va a buscar. Este identificador no, es más, que el valor que tiene asignado el atributo botton_adicionar (es el identificador que tiene ese botón en el HTML)**.** Una vez localizado entonces se puede dar clic en el utilizando la función clik ().

Cuando se da clic automáticamente se abre una página que es el formulario para Adicionar una nueva computadora. Al acceder a ella, se debe localizar los campos Computer name, Introduced y Discontinued, para posteriormente utilizar la función send_keys (texto). La cual te permite escribir en un campo de texto. En este caso por ejemplo en el caso del campo Computer name una vez localizado se va a escribir “Asus”.

En el caso específico del campo Company como es una lista que te permite seleccionar entre varios elementos. Primero se obtuvo ese componente de selección mediante su identificador y una vez seleccionado se puede utilizar la función select_by_value (elemento), donde entre paréntesis se define de todos los elementos cual se desea

seleccionar. Una vez que están todos los campos llenados y seleccionados entonces queda enviar la información. En este caso se localiza el botón de Create this computer utilizando el localizador XPath, recordar que se definió un atributo que tiene almacenado el XPath de este componente. Una vez localizado entonces se puede utilizar el método submit () para poder enviar la información. Si todo sale correcto una vez que se seleccione la opción Create this computer entonces se regresa al listado de computadoras.

Hasta el momento solo se ha modelado el comportamiento de la página cuando se adiciona una computadora de forma correcta. Pero se debe verificar si este comportamiento se realiza de forma correcta y no ocurren fallos. Para esto es donde se crea el otro archivo donde se crean los test de pruebas en este caso el archivo Test_PC.py.

En este archivo se debe de importar las bibliotecas:

```
from selenium import webdriver
import unittest
from Page_PC import Page_PC
```

**Figura 7**. Bibliotecas necesarias en el archivo Test_PC.py

- ✓ unittest es una biblioteca de Python que proporciona una clase base, TestCase, que se puede utilizar para crear un conjunto de pruebas.
- ✓ Por último, importar la clase Page_PC que es la que tiene todas las funciones que tienen indistintamente los comportamientos definidos para cada uno de los test.

Al definir la clase donde se van a definir todos los test esta debe de heredar de la clase TestCase de unnittest. Esta clase está pensada para ser utilizada como clase base, es decir, representa las unidades lógicas de test, es decir tiene implementado métodos que el código de test puede utilizar para chequear y reportar distintos tipos de fallo. Las instancias de TestCase proveen tres grupos de métodos: un grupo empleado para ejecutar el test, otro usado para que la implementación del test chequee condiciones y reporte fallos, y algunos métodos de indagación que permiten recopilar información sobre el test en sí mismo. En este caso se utilizan los del primer y segundo grupo. Primeramente, se define el método setUp (). Este método es utilizado para ejecutar un conjunto fijo de acciones antes de ejecutar los test y no devuelve nada. En este caso las instrucciones definidas fueron las siguientes.

```
class Test_GestionarPC(unittest.TestCase):
    def setUp(self):
        self.driver=webdriver.Firefox()
        self.driver.get('http://computer-database.gatling.io/computers')
        self.driver.implicitly_wait(25)
```

**Figura 8**. Función setUp ().

En este caso primero se debe crear una instancia de un controlador del navegador con el que se quiere trabajar en este caso utilizando la biblioteca webdriver se puede definir el navegador que en este caso es Firefox. Una vez que se crea la instancia de ese navegador entonces se le da la instrucción de abrir una dirección mediante la utilización del método get (parámetro) el cual recibe como parámetro la dirección del sitio que se desea probar. Una vez que se acceda entonces se utiliza el método implicity_wait el cual recibe por parámetro el tiempo que se debe esperar antes de realizar una acción. Esto es una buena práctica porque en ocasiones la página puede tener varios componentes, lo cual puede rentalizar las pruebas y que los diferentes test no se realicen de forma adecuada ya que; puede traer consigo que no se localicen los elementos de forma correcta. En conclusión, este método no valida si hay fallos, sino que prepara el entorno para los test de pruebas. Una vez que se definió esta función entonces ya se pueden crear los test de pruebas en este caso todos los test que se creen su nombre deben de empezar por test. Por ejemplo, en este caso se va a crear el test para cuando se adicione una computadora de forma correcta es decir comprobar que el primer escenario se realice de forma correcta. A continuación, se muestra el código del test:

```
def test_Adicionar_Computadora(self):
    page_adicionar=Page_PC(self.driver)
    page_adicionar.Adicionar()
```

**Figura 9.** Test Introduce y/o selecciona los datos de una computadora y selecciona la opción Create this computer.

Como se evidencia el test está compuesto por dos solas instrucciones en una se crea la instancia de la clase Page_PC en la cual se encuentra definido un método que tiene toda la lógica de comportamiento de cuando se adicione una computadora de forma correcta. En este caso recordar que para crear una instancia de esta clase su función de inicialización recibía como parámetro una variable que hace referencia a la instancia del navegador que se creó. Por tanto, al crear la instancia de esta clase se le pasa como parámetro la instancia del navegador Firefox creado. Una vez creada la instancia de esta clase se pueden acceder a sus funciones que en este caso sería Adicionar.

El test implementado es fácil de comprender ya que; fue utilizado como referencia el patrón Page Object que te permite separar la lógica de comportamiento de la página con el test en sí mismo. Una vez que se ejecuta el test se evidencia de forma automática una simulación de los pasos definidos. De no existir error se mostrará un OK como señal de que la prueba pasa. Cuando se ejecuta los test pueden arrojar los siguientes resultados:

**Tabla 1**.Posibles resultados al ejecutar un test.

| **Resultado** | **Descripción** |
|---|---|
| Ok | La prueba pasa |
| Fail | La prueba pasa y genera una exception AssertionError |
| Error | La prueba genera cualquier excepción que no es un AssertionError |

Finalmente, después de ejecutada este es el resultado.

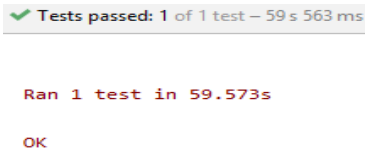


**Figura 10.** Resultado obtenido al ejecutar la prueba.

Se evidencia que no hay error y el test se ejecutó en un tiempo muy pequeño en comparación a si se realiza de forma manual.

Escenario se modifican los datos de una computadora y se selecciona la opción Update this computer.

A continuación, se proporciona otro ejemplo en el que se explica los pasos realizados para definir el comportamiento de cuando se desea Modificar los datos de una computadora de forma correcta.

Primeramente, se crea una función en la que se define el comportamiento de cuando se Modifica los datos de una computadora.

```
def Modificar_Campos(self):
    tab = self.driver.find_element(By.XPATH, self.tabla)
    lista = []
    for d in tab.find_elements(By.TAG_NAME, "a"):
        f = d.get_attribute('href')
        lista.append(f)
    z = lista[4]
    print(z)
    self.driver.execute_script("window.open('');")
    self.driver.switch_to.window(self.driver.window_handles[1])
    self.driver.get(z)
    d=self.driver.find_element(By.ID, self.introduced)
    d.send_keys("2021-03-04")
    self.driver.find_element(By.XPATH, self.botton_create).submit()
    w = self.driver.find_element_by_xpath('/html/body/section/table/tbody/tr[1]')
    lista_da = []
    for busc in w.find_elements_by_tag_name('td'):
        es = busc.text
        lista_da.append(es)
    prueba = lista_da[1]
    return prueba
```

**Figura 11**. Función que define el comportamiento de la página web cuando se Modifica los datos de una computadora.

1. En la misma clase PAGE_PC.py se define una nueva función que en este caso se nombró por Modificar ().

2. En este caso se localiza la tabla por el XPath una vez que se tiene localizada se recorre cada una de las etiquetas <a> que pertenecen a ella. En este caso se utiliza una función nueva que es la de find_elements en este caso es parecido a find_element, es decir te permite buscar todos los elementos que tengan en este caso etiqueta <a>**.**
3. Después, de las etiquetas a se van a obtener todos los links y se van adicionar a una lista. En este caso para obtener el link se utiliza la función get_attribute(‘href’).
4. Una vez que se encuentre la lista llena, se obtiene la que se encuentra en la posición número 5 índice 4 ya que; mediante este link se accede al formulario de la primera computadora que aparece en el listado que en este caso tiene como nombre ACE y es la que se quiere modificar sus datos.
5. Después que se tiene el link entonces, se abre una nueva pestaña mediante el método execute_script(“window.open(‘’) ;) el cual te crea una nueva pestaña en el navegador. Entonces en ese mismo momento estarían abiertas dos pestañas.
6. En la segunda es donde se quiere abrir el link donde se encuentra el formulario de esa computadora para esto se utiliza la función switch_to.window(self.driver.window_handles[1]). Con este método lo que se está haciendo es parándose en esa segunda pestaña. Una vez que se este en la segunda pestaña entonces se le indica que debe acceder al link que se encuentra almacenado en la variable z. Hay que tener presente que si se hubiera puesto entre los corchetes un 0 se estaría parando en la pestaña donde se encuentra la tabla.
7. Una vez que se encuentra en el formulario de esa computadora entonces, se localizará el campo Introduced con el objetivo de poner una fecha de entrada porque antes no tenía nada.
8. Una vez localizado hay que escribir en el campo de texto utilizando el método send_keys (texto) el cual se le pasa por parámetro el texto que se quiere poner que en este caso es la fecha siguiente 2021-03-04.
9. Después que se introduce la fecha se localiza el botón utilizado para enviar los datos del formulario y una vez localizado se utiliza el método submit para enviar la informacion.
10. Cuando se modifica un dato en el formulario se regresa al listado de computadora y supuestamente los datos modificados deben verse reflejados. Es por esto que en este caso cuando se realice submit se regresa al listado para localizar donde se encuentra esa computadora, y de esa fila se localiza todas las etiquetas <tr> para obtener los textos e ir ingresando en la lista. Finalmente, la función implementada devolverá lo que se encuentra en el indicé 1 de la lista, que es donde se encuentra almacenado el texto que se debe encontrar en el campo Introduced de esa computadora.

Una vez realizado esta función en el archivo Page_PC, se realiza el test de prueba en el archivo PC.py. A continuación, se muestra el test creado.

```
def test_Modificar(self):
    page_nueva = Page_PC(self.driver)
    prueba = page_nueva.Modificar_Campos()
    self.assertEqual(prueba, '2021-03-04', 'No paso el test')
```

**Figura 12**.Test de prueba para cuando se modifiquen los datos de una computadora.

En este caso de igual forma se crea una instancia de la clase Page_PC y cuando este creado entonces de esta clase se invoca la función Modificar_Campos. Enfatizar que esta función devuelve un valor que en este caso es un texto que va a representar la fecha de entrada de esa computadora que se decidió modificar su fecha. Después lo que se hace es utilizar un método de afirmación que ofrece la clase TestCase para comprobar y reportar fallos, en este caso assertEqual (primer, segundo, tercer). Este método permite comprobar si el parámetro a es igual al b si es así pasa la prueba. El tercer parámetro si se especifica es un mensaje que se mostrará en caso de que no sean iguales. A continuación, se muestran otros métodos de comprobación que ofrece la clase TestCase

**Tabla 2**. Métodos de comprobación que ofrece la clase TestCase de Python.

| **Método** | **Comprueba que** |
|---|---|
| assertEqual (a, b) | a==b |
| assertNotEqual (a, b) | a! =b |
| assertTrue(x) | bool x= true |
| assertFalse(x) | bool x=false |
| assertIs(a, b) | a is b |
| assertIsNot(a, b) | a is not b |
| assertIsNone(x) | x is none |
| assertIsNotNone(x) | x is not none |
| assertIn (a, b) | a in b |
| assertNotIn (a, b) | a not in b |
| assertIsInstance (a, b) | isInstance (a, b) |
| assertNotIsInstance (a, b) | not isInstance (a, b) |

En este ejemplo los dos elementos que se comparan es el valor que devuelve la función Modificar_Campos (texto que representa lo que se encuentra almacenado en el campo Introduced de la computadora seleccionada) con la fecha que supuestamente debería tener esa computadora que es 2021-03-04). Una vez ejecutado el test se muestran los resultados.

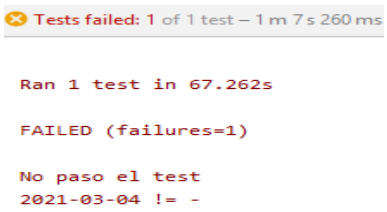


**Figura 13**.Resultado obtenido una vez ejecutado el test.

Como se evidencia el test falló ya que; los dos datos son diferentes. Por lo que; se identifica una no conformidad. Es decir, cuando se Modifica los datos de una computadora estos no se actualizan en el listado de las computadoras. En caso de que se comprará con el carácter (-) al ejecutar el test se evidencia que no existe error y que la prueba pasa. Es decir, se puede identificar que la función creada en la clase Page_PC.py se realiza de forma correcta.

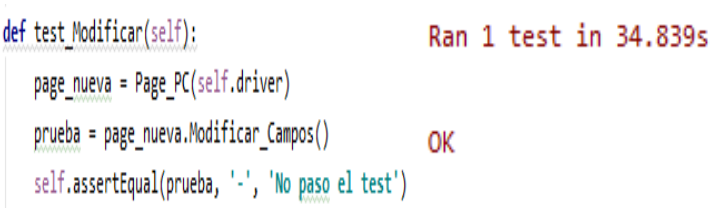


**Figura 14**. Resultados obtenidos una vez modificado el criterio de comparación del test.

Finalmente, en la clase Test_PC.py después de todos los test se define el método tearDown () el cual es llamado inmediatamente después de que se ejecute un test de prueba y se obtenga registrado el resultado. Este este compuesto por una solo instrucción que indica que cuando se termine de ejecutar un test se cierre el navegador.

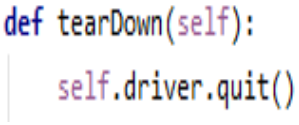


**Figura 15**. Función tearDow.

Finalmente, un proyecto real estará compuesto por varios casos de pruebas. Cuando existen casos de pruebas relacionados entre sí, por ejemplo, que pertenezcan a un módulo determinado del sistema se pueden integrar a una suite de prueba, donde se definirán que casos de pruebas lo integran y con un solo clic se mostrara los resultados

obtenidos una vez ejecutados los casos de prueba. Para crear una suite hay que crear un nuevo archivo .py y seguir los pasos que se encuentran en la Figura 16:

```
import unittest
from Test_PC import Test_GestionarPC
#Cargando los casos de prueba que se van añadir a la suite
testu=unittest.TestLoader().loadTestsFromTestCase(Test_GestionarPC)
#Creando la suite de prueba y añadiendo al listado  los casos de prueba que van a pertenecer
#a esa suite
suite = unittest.TestSuite([testu])
#Para correr la suite de prueba
unittest.TextTestRunner(verbosity=2).run(suite)
```

**Figura 16**.Creación de una suite de prueba

Luego de ejecutarlo se evidencia que hubo un solo error y que el tiempo en ejecutar este caso de prueba con los 4 test incluidos fue de: 135.694s.

```
C:\Users\maydalis\PycharmProjects\untitled6\venv\Scripts\python.exe C:/Users/maydalis/PycharmProjects/untitled6/test_Suite.py
test_Adicionar_Campos_Vacios (Test_PC.Test_GestionarPC) ... ok
test_Adicionar_Computadora (Test_PC.Test_GestionarPC) ... ok
test_Cancelar_Adicionar (Test_PC.Test_GestionarPC) ... 574 computers found
ok
test_Modificar (Test_PC.Test_GestionarPC) ... http://computer-database.gatling.io/computers/381
FAIL
```

**Figura 17**. Resultados obtenidos después de ejecutar la suite de prueba.

### Análisis de los resultados

Para demostrar que las pruebas automáticas son más eficientes que las manuales se realizaron los mismos de test de forma manual y contigua. El tiempo empleado para la ejecución de los mismos fue de 6 minutos demostrándose así que las pruebas automatizadas tienen un tiempo de ejecución menor, aunque su coste inicial de implementación es alto. Sin embargo, el ahorro de tiempo contra restra esta inversión inicial consiguiendo que sean mucho más rentables a nivel económico.

## Conclusiones

El uso principal de Selenium WebDriver radica en el desarrollo de pruebas funcionales porque permite simular el comportamiento de un usuario trabajando en el sistema. Para utilizar las API que ofrece para los lenguajes de Java y en este caso Python es necesario tener conocimiento del lenguaje de programación empleado. La utilización del

patrón Page Object Model es una buena práctica porque permite dividir el comportamiento de la página web con los test en sí, lo que permite a cualquier miembro del equipo comprender los test realizados de forma fácil. Esto facilita que si se realiza alguna modificación posteriormente estos puedan ser modificados de forma sencilla. De forma general se demostró que con el desarrollo de pruebas automatizadas en el proyecto se solventan todos los problemas mencionados con las pruebas manuales y que su principal ventaja radica en una disminución del tiempo empleado para el desarrollo de pruebas funcionales durante el desarrollo del proyecto.

## Conflictos de intereses

Los autores no poseen conflictos de intereses.

## Contribución de los autores

1. Conceptualización: Maydalis Hernández Pérez, Luis Angel Llull Céspedes.
2. Curación de datos: Maydalis Hernández Pérez, Luis Angel Llull Céspedes.
3. Investigación: Maydalis Hernández Pérez, Luis Angel Llull Céspedes.
4. Software: Maydalis Hernández Pérez, Luis Angel Llull Céspedes.
5. Visualización: Maydalis Hernández Pérez, Luis Angel Llull Céspedes.
6. Redacción – borrador original: Maydalis Hernández Pérez, Luis Angel Llull Céspedes.
7. Redacción – revisión y edición: Maydalis Hernández Pérez, Luis Angel Llull Céspedes.

## Financiamiento

El trabajo no requirió financiación

## Referencias

Feldman, Start. Quality Assurance: Much More than Testing. Queve-Quality Assurance,2005, 3(1):26-29.

Jha, N; Pople, R; Chackraborty, S; Kumar, P. Software test Automation using Selenium and Machine Learning.

Muthukadan, Baiju. Selenium Python Bindings, 2022.

Nishmitha, G.C; Dr. Phaneendra, H.D. Implementation of Page Object Model in Selenium Automation. International Journal of Advanced Research in Computer and Communication Engineering, 2006, 5:446-448.

Nyamathulla, S; Ratnababu, P; Sultana, Nazma Shaik; Lakshmi, Bhagya N. A Review on Selenium Web Driver with Python, 2021, 25: 16760-16768.

Serna, M.E; Arango, F. Software testing: More than a stage in the life cycle. Revista de Ingenería, 35:34-40.

Proceedings of First International Conference on Computational Electronics for Wireless Communications. Lecture Notes in Networks and Systems, 2022, 329.

Serna, Edgar M; Martínez, Raquel M; Tamayo, Paula O. Una revisión a la realidad de la automatización de las pruebas del software. Computación y Sistemas, 2019, 23(1): 169–183.

Sommerville, Ian. Ingenería de Software. Ciudad de México, Pearson Educación, 2011. 206.

Zhan, Zhimin. Selenium WebDriver Recipes in Python, 2021.