

Guía de ejercicios

Evaluación de Sistema QA
Semana 2



Nombre: Herramientas de Automatización de Prueba

Desarrollo de la guía

Instrucciones

Estimado(a) estudiante:

Es un placer presentarte una guía para ayudarte a familiarizarte con Selenium, una poderosa herramienta de automatización de pruebas de sistemas.

El propósito de esta guía es ofrecer una visión detallada y práctica del proceso de automatización de pruebas utilizando Selenium.

A lo largo de este documento, nos centraremos en un ejercicio completo (a nivel básico) que abarca la preparación del ambiente de prueba, la ejecución del programa de pruebas y el análisis de los resultados obtenidos.

Es fundamental destacar que este recurso no constituye una evaluación, sino más bien un material adicional destinado a brindarte la oportunidad de practicar lo estudiado a nivel teórico y reforzar tus habilidades.

¡Anímate a abordar estos desafíos y descubre cómo tu habilidad para automatizar pruebas puede crecer con cada ejercicio analizado, comprendido y completado!

Requisitos previos:

El ejercicio que se detalla a continuación incluye el código fuente necesario. Simplemente sigue los pasos descritos en el documento. No te preocupes, no necesitas ser un experto en programación, ya que nos enfocaremos exclusivamente en las pruebas de un software que ya ha sido construido en Python. Sin embargo, es importante contar con un entendimiento básico de lógica de programación y un lenguaje de desarrollo.

Además, para trabajar de manera efectiva, necesitarás tener instalado el Entorno de Desarrollo Integrado (IDE) Visual Studio Code y haber configurado el entorno Python. Esta configuración te permitirá ejecutar las pruebas de manera fluida y eficiente.

Ejercicio:

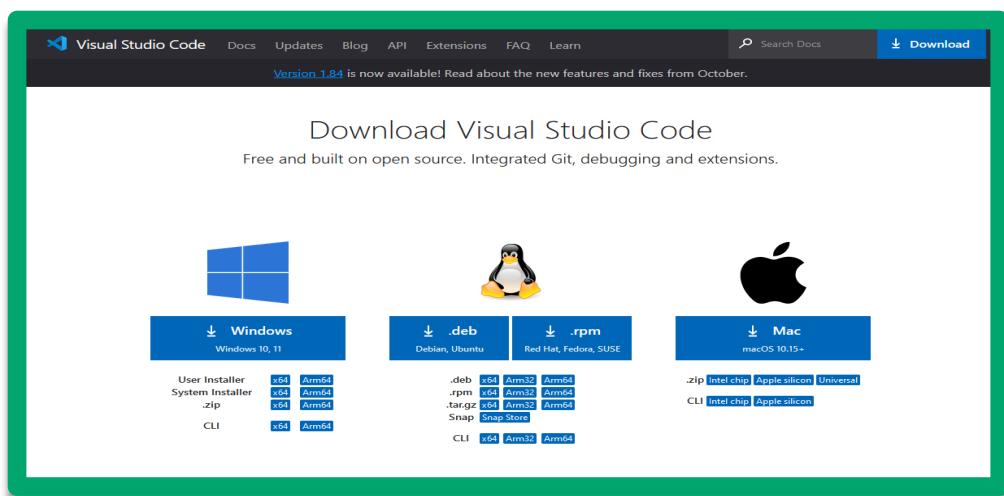
Desarrollar pruebas automatizadas utilizando Selenium para las operaciones principales de una calculadora sencilla implementada en Python.

Para alcanzar lo anterior, se construyó la Función [probar_operacion\(driver, num1, num2, operador, resultado esperado\)](#). Se encarga de ejecutar la prueba con los datos respectivos.

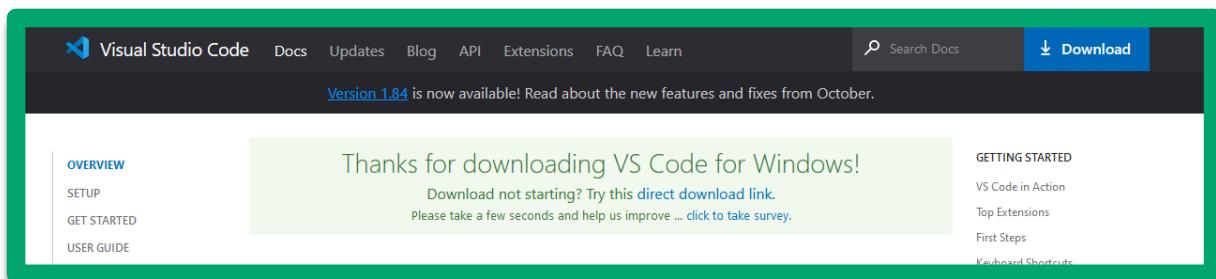
IMPORTANTE: Esta sección brinda al equipo encargado la flexibilidad para realizar modificaciones continuas, incluyendo la adición de nuevos datos de prueba. Aquí se especifican los parámetros a probar y los resultados esperados.

Fase I: Selenium**1.Preparación del ambiente****Instalación de Visual Studio Code desde su sitio oficial**

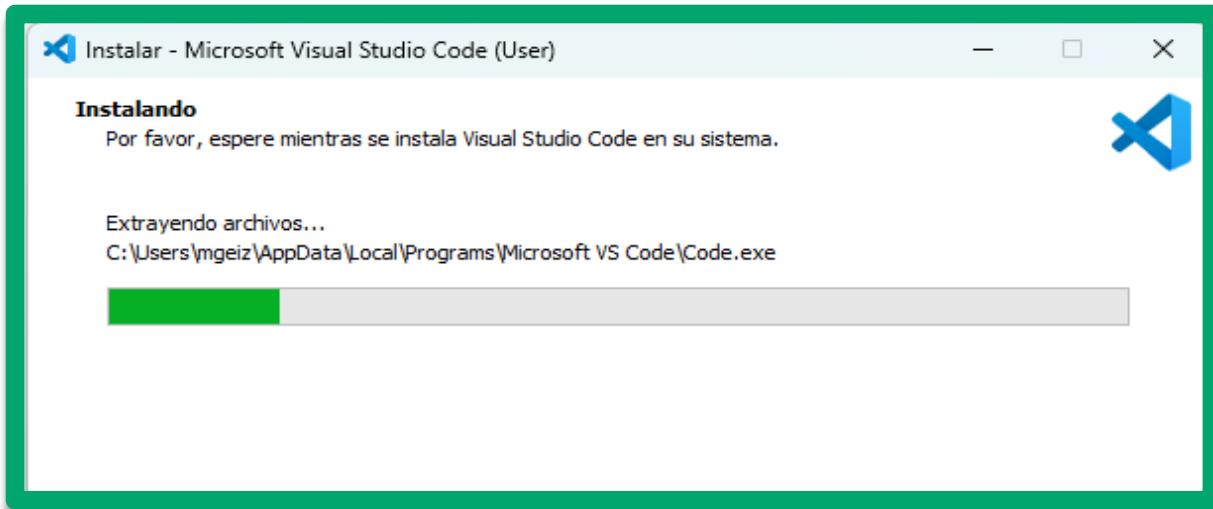
- a. Descargarlo del siguiente URL: <https://code.visualstudio.com/download>



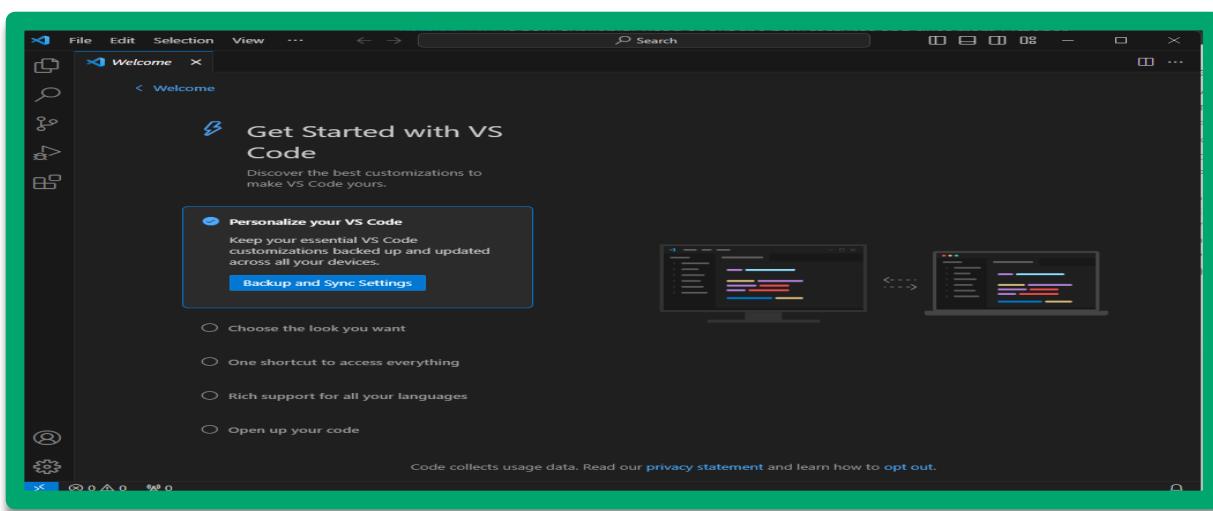
- b. Seleccionar la opción de nuestra preferencia según las características del Sistema Operativo. Para este ejemplo, descargaremos la versión para Windows 11.



- c. Abrir el archivo ejecutable y procedemos con la instalación



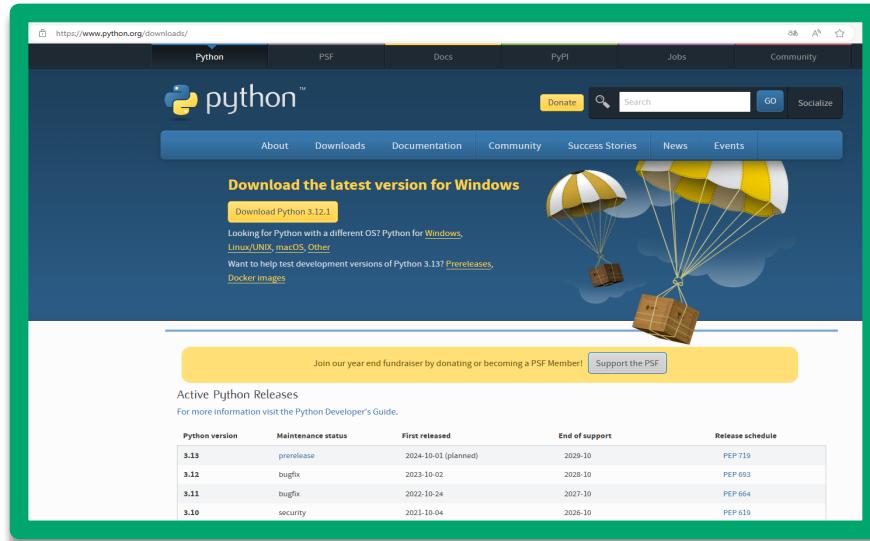
d. Abrir Visual Studio Code



IMPORTANTE: Dejarlo abierto para utilizarlo más adelante.

2. Instalación de Python:

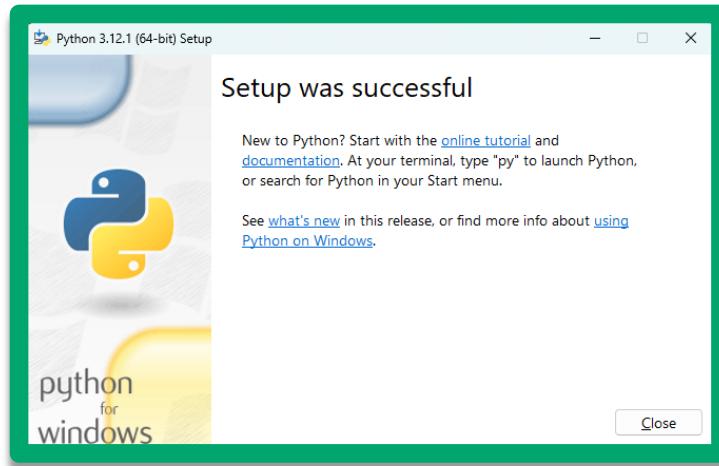
- Descargarlo desde la página oficial de Python e instalar la versión adecuada para tu sistema operativo.



- Presionar download Python 3.12.1
- Ejecutar el instalador

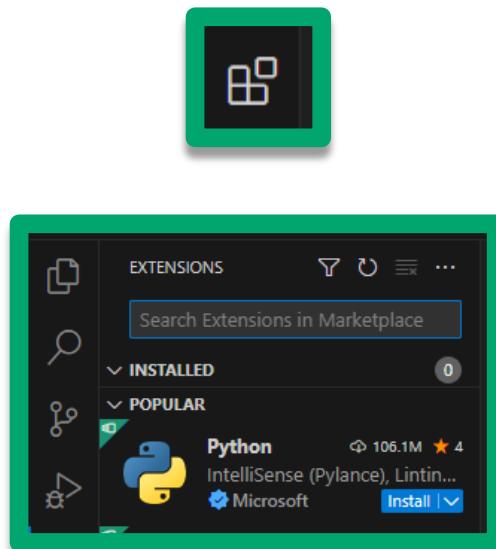


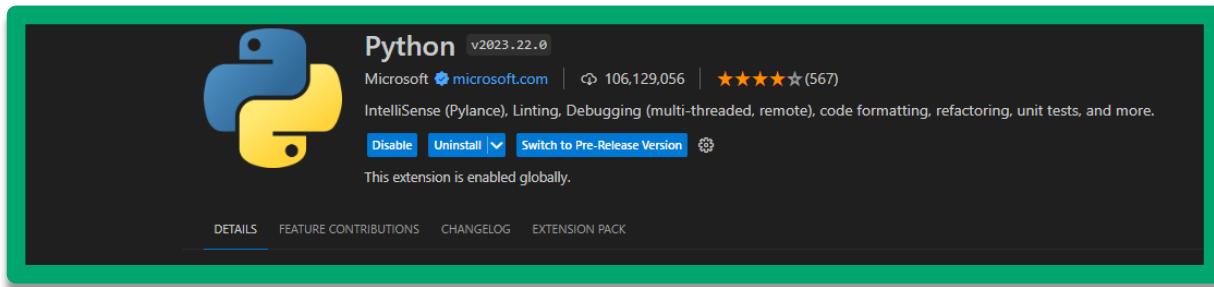
- Seleccionamos las dos opciones (Use admin/Add Python) y presionamos Install now



2.1. Instalar la extensión de Python para Visual Studio Code:

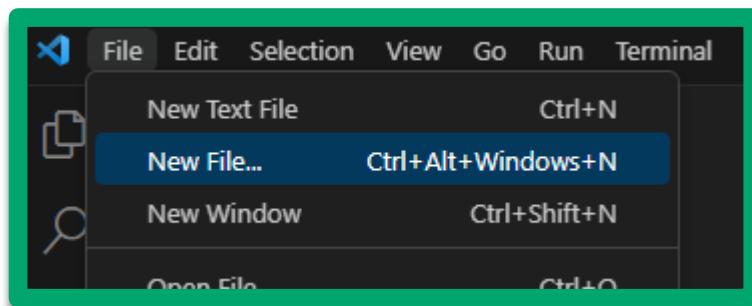
Abrir Visual Studio Code y ve a la pestaña de extensiones (lado izquierdo de la interfaz o usando Ctrl+Shift+X). Busca "Python" en la barra de búsqueda de extensiones. Selecciona la extensión oficial de Python (creada por Microsoft) y haz clic en "Install" para instalarla.



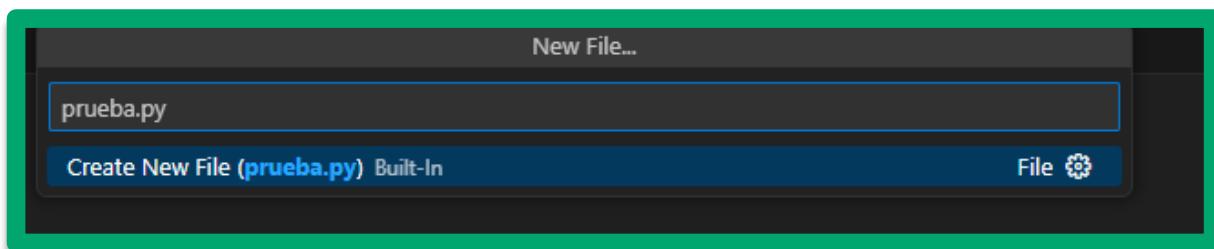


a. Crear o abre un archivo Python:

Crea un nuevo archivo .py o abre uno existente para comenzar a escribir código en Python (esto es solo para probar instalación, porque al final el código del ejercicio de la guía está más adelante). Pueden obviar este tipo de prueba.



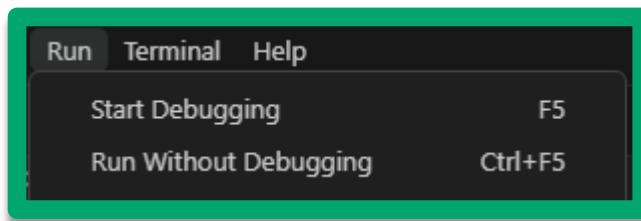
b. Si vas a crear un nuevo archivo para Python, asegúrate de que la extensión termine en .py



c. Luego, copia un código Python

Ejecuta tu código:

Para ejecutar tu código, puedes ir a la opción Run – Run Without Debugging, o presionar directamente Control-F5



3. Instalar Flask

Flask es un popular *framework web* ligero y flexible para Python. Se utiliza para crear aplicaciones *web* rápidas y eficientes con Python de una manera sencilla y modular. Está diseñado para ser fácil de aprender y utilizar, lo que lo hace ideal para desarrolladores que están comenzando con el desarrollo *web* o que desean crear aplicaciones *web* simples y rápidas.

Para instalar Flask en Visual Studio Code, debes seguir los pasos a continuación:

1. Abre Visual Studio Code.
2. Abre la terminal integrada de Visual Studio Code. Puedes hacerlo seleccionando “Terminal” en el menú superior y luego “Nueva Terminal”.
3. En la terminal ejecuta el siguiente comando para instalar Flask usando pip:

pip install flask

Esto instalará Flask en el entorno virtual de Visual Studio Code.

Después de instalar Flask, puedes:

- Crear una aplicación Flask en Visual Studio Code y comenzar a trabajar.
- Debes crear un archivo tipo nombre.py con el código de la aplicación y una carpeta templates con archivos HTML como index.html y result.html.

IMPORTANTE:

Para ejecutar tu aplicación Flask desde Visual Studio Code, simplemente ejecuta python nombre.py en la terminal integrada, asegurándote de que estés en el directorio correcto donde se encuentra tu archivo nombre.py. Luego, podrás acceder a tu aplicación Flask desde tu navegador *web* en la dirección **http://localhost:5000**.

4. Instalar Selenium

Instalar Selenium ejecutando el siguiente comando en la línea de comandos de la terminal de Visual Studio Code:

pip install selenium

Este comando instalará Selenium y todas las dependencias necesarias en tu entorno de Python.

Después de seguir estos pasos, Selenium estará instalado en tu sistema y podrás usarlo en los proyectos de Python, incluidos aquellos que estás desarrollando en Visual Studio Code.

4.1. Instalar controlador Selenium para navegador

Selenium requiere que tengas instalado el navegador Google Chrome, al cual se le instalará un controlador que permitirá a Selenium tomar control de la página web y poder desplazarse a través de los diferentes elementos que contenga.

Para instalar el controlador, debes ir a la página web

<https://chromedriver.chromium.org/downloads>

Y descargar la última versión para el sistema operativo que dispongas.

Una vez que hallas descargado el archivo, deberás ejecutarlo y el controlador quedará cargado en memoria.

Nombre	Tipo	Tamaño comprimido	Protegido ...	Tamaño
chromedriver.exe	Aplicación	6.404 KB	No	11.986 KB
LICENSE.chromedriver	Archivo CHROMEDRIVER	46 KB	No	243 KB

Al ejecutar el archivo, se abrirá una ventana que no deberás cerrar mientras dure la sesión de pruebas con Selenium:

```
C:\Users\lfzene\Downloads\chromedriver_win32\chromedriver.exe
Starting ChromeDriver 114.0.5735.90 (386bc09e8f4f2e025eddae123f36f6263096ae49-refs/branch-heads/5735@{#1052}) on port 95
15
Only local connections are allowed.
Please see https://chromedriver.chromium.org/security-considerations for suggestions on keeping ChromeDriver safe.
ChromeDriver was started successfully.
```

Fase II: Ejecución de pruebas

1. Código del *software* o sistema a probar

A continuación, se muestra el código completo del *software* a probar, además de la plantilla HTML. Se desarrolló una calculadora sencilla en una página *web*, incluyendo las cuatro operaciones básicas.

Código de aplicación de calculadora sencilla en Python

```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/calculate', methods=['POST'])
def calculate():
    num1 = float(request.form['num1'])
    num2 = float(request.form['num2'])
    operator = request.form['operator']

    if operator == 'sumar':
        result = num1 + num2 + 1
    elif operator == 'restar':
        result = num1 - num2
    elif operator == 'multiplicar':
        result = num1 * num2
    elif operator == 'dividir':
        if num2 != 0:
            result = num1 / num2
        else:
            result = 'Error: División por cero'
    else:
        result = 'Error: Operador inválido'

    return render_template('result.html', result=result)
```

```
    return render_template('index.html', result=result)

if __name__ == '__main__':
    app.run(debug=True)
```

- Crea una carpeta denominada “pruebas” y guarda este código en un archivo denominado “calculadora.py” dentro de esa carpeta.

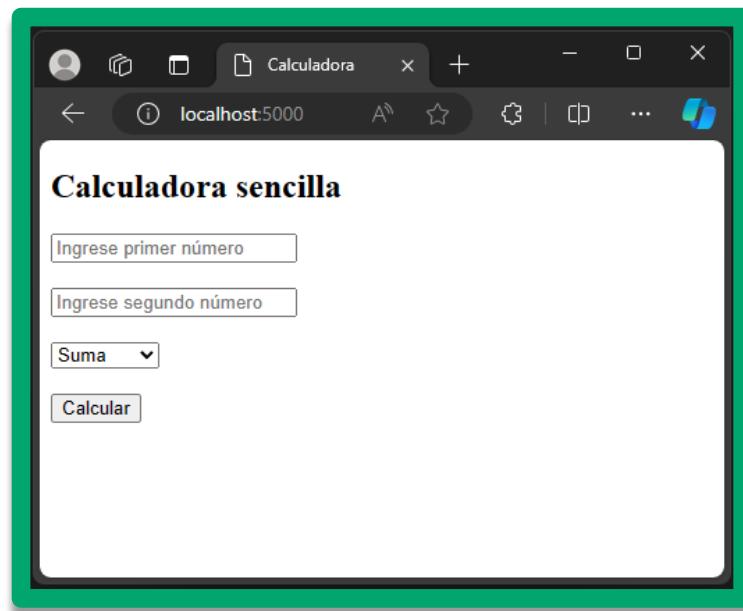
Plantilla HTML de la calculadora

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Calculadora</title>
</head>
<body>
    <h2>Calculadora sencilla</h2>
    <form action="/calculate" method="post">
        <input type="text" name="num1" placeholder="Ingrese primer número"
required><br><br>
        <input type="text" name="num2" placeholder="Ingrese segundo número"
required><br><br>
        <select name="operator">
            <option value="sumar">Suma</option>
            <option value="restar">Resta</option>
            <option value="multiplicar">Multiplica</option>
            <option value="dividir">Divide</option>
        </select><br><br>
        <input type="submit" value="Calcular">
    </form>
    {% if result %}
        <h2>Resultado:</h2>
        <p>{{ result }}</p>
    {% endif %}
</body>
</html>
```

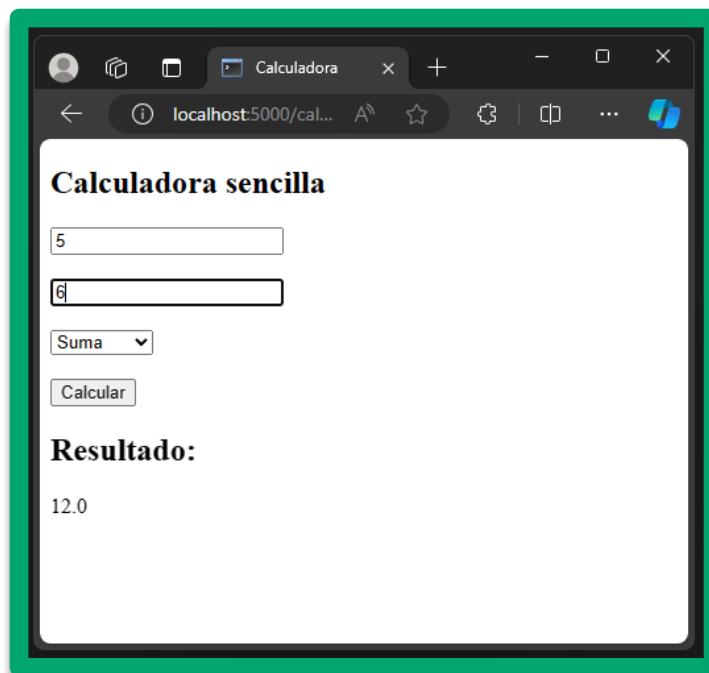
- Crea una carpeta dentro de la carpeta “pruebas” llamada “templates” y guarda este código en un archivo denominado “index.html”

2. Ejecución del programa:

- Para ejecutar el programa, en Visual Studio Code, presiona Control-F5, selecciona Python (esto solo se hará la primera vez), y luego ubica la terminal de Visual Studio Code, y presiona el enlace directo de la URL en donde se ha instanciado la plantilla, o escribe directamente en el navegador la siguiente dirección: localhost:5000.



2.1. Prueba la aplicación, ingresando los números, seleccionando el operador y haciendo clic en el botón “Calcular”:



- Observa que deliberadamente, se ha dejado un defecto en la aplicación en torno a la operación de la adición, ya que el resultado esperado es 11.0, pero está sumando un número de más. Esto se ha dejado así con el propósito de convertirlo en un hallazgo cuando se realicen las pruebas automatizadas.

3. Prueba automatizada con Selenium

Se ha desarrollado un programa en Python utilizando Selenium, con el objetivo de realizar pruebas automatizadas del programa de la calculadora sencilla. A continuación el código:

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

def iniciar_navegador():
    driver = webdriver.Chrome()
    return driver

def abrir_pagina(driver, url):
    driver.get(url)

def realizar_operacion(driver, num1, num2, operador):
    num1_input = driver.find_element("name", "num1")
    num2_input = driver.find_element("name", "num2")
    operator_select = driver.find_element("name", "operator")

    submit_button = driver.find_element("xpath", "//input[@type='submit']")

    num1_input.clear()
    num2_input.clear()

    num1_input.send_keys(str(num1))
    num2_input.send_keys(str(num2))

        #0 adición
    if operador == 1:
        operator_select.send_keys(Keys.DOWN)           #sustracción
    elif operador == 2:
        operator_select.send_keys(Keys.DOWN)           #multiplicación
        operator_select.send_keys(Keys.DOWN)
    elif operador == 3:
        operator_select.send_keys(Keys.DOWN)           #división
        operator_select.send_keys(Keys.DOWN)
        operator_select.send_keys(Keys.DOWN)
```

```
submit_button.click()

time.sleep(2) # Esperar a que se muestre el resultado

def obtener_resultado(driver):
    result_element = driver.find_element("xpath","//p")
    return result_element.text

def cerrar_navegador(driver):
    driver.quit()

def probar_operacion(driver, num1, num2, operador, resultado_esperado):

    realizar_operacion(driver, num1, num2, operador)
    resultado_obtenido = obtener_resultado(driver)

    if operador == 0:
        accion = "más"
    elif operador == 1:
        accion = "menos"
    elif operador == 2:
        accion = "por"
    elif operador == 3:
        accion = "entre"

    print(f"Operación: {num1} {accion} {num2}")
    print("Resultado esperado:", resultado_esperado)
    print("Resultado obtenido:", resultado_obtenido)
    if str(resultado_esperado) == resultado_obtenido:
        print("¡Prueba exitosa!")
    else:
        print("¡Prueba fallida!")

def ejecutar_pruebas():
    driver = iniciar_navegador()
    abrir_pagina(driver, "http://localhost:5000")

    #0 down suma
    #1 down resta
    #2 down multiplica
    #3 down divide

    # Pruebas de adición
    probar_operacion(driver, 10, 5, 0, '15.0')
    probar_operacion(driver, 20, 30, 0, '50.0')

    # Pruebas de sustracción
```

```
probar_operacion(driver, 10, 5, 1, '5.0')
probar_operacion(driver, 50, 30, 1, '20.0')

# Pruebas de multiplicación
probar_operacion(driver, 5, 5, 2, '25.0')
probar_operacion(driver, 6, 7, 2, '42.0')

# Pruebas de división
probar_operacion(driver, 100, 5, 3, '20.0')
probar_operacion(driver, 50, 2, 3, '25.0')

cerrar_navegador(driver)

# Ejecutar las pruebas
ejecutar_pruebas()
```

Explicación del código:

Función iniciar_navegador():

Esta función inicializa un navegador *web* controlado por Selenium. En este caso, se utiliza el controlador de Chrome (`webdriver.Chrome()`) para abrir una instancia del navegador Chrome.

Función abrir_pagina(driver, url):

Abre una página *web* en el navegador controlado por Selenium. Recibe como parámetros el objeto `driver` que representa el navegador y la URL de la página que se desea abrir.

Función realizar_operacion(driver, num1, num2, operador):

Esta función realiza una operación en la calculadora *web*. Encuentra los elementos de entrada (números y operador), ingresa los números y selecciona la operación deseada. Luego, hace clic en el botón de calcular y espera a que se muestre el resultado.

La selección del operador se hace simulando la presión de la tecla “DOWN” (flecha abajo), aplicada al objeto de la página *web* que despliega una lista de opciones.

Cuando se desea seleccionar el operador adición, no se aplica la tecla,

Si el operador a seleccionar es sustracción, se aplica la tecla una sola vez, si el operador es multiplicación, se aplica dos veces y si se desea seleccionar división, se aplica tres veces.

Función obtener_resultado(driver):

Obtiene el resultado de la operación realizada en la calculadora *web*. Busca el elemento `<p>` que contiene el resultado y devuelve su texto.

Función cerrar_navegador(driver):

Esta función cierra el navegador controlado por Selenium.

Función ejecutar_pruebas():

Esta función ejecuta una serie de pruebas en la calculadora *web*. Primero, inicializa el navegador, luego abre la página *web* de la calculadora. Luego, ejecuta una serie de pruebas para diferentes operaciones (adición, sustracción, multiplicación y división) con diferentes conjuntos de números. Finalmente, cierra el navegador.

Ejecución de las pruebas:

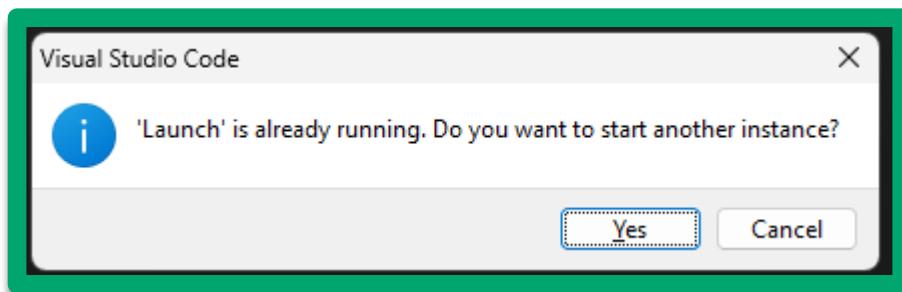
Se llama a la función ejecutar_pruebas() para ejecutar todas las pruebas definidas en el *script*. El *script* contiene dos (2) pruebas definidas para cada operador, en total, ocho (8) pruebas.

Función probar_operacion(driver, num1, num2, operador, resultado Esperado):

- Es la función que se encarga de ejecutar la prueba, y que contiene los datos a probar.
- Es la sección que el equipo encargado, podrá modificar constantemente agregando nuevos datos de prueba, se especifican los parámetros a probar y los resultados esperados.
- Esta función realiza una prueba de una operación en la calculadora *web* y compara el resultado obtenido con el resultado esperado. Primero realiza la operación llamando a realizar_operacion(), luego obtiene el resultado y lo compara con el resultado esperado. Finalmente, imprime el resultado de la prueba.

3.1. Ejecución del programa de pruebas

- Para poder ejecutar el programa de pruebas, se debe tener ejecutando el programa de la calculadora. Al iniciar la ejecución con Visual Studio Code, aparecerá el siguiente mensaje:



- Al cual debemos responder "Si" (Yes).
- Inmediatamente, se abrirá de manera automática, la página index.html con el navegador Google Chrome, y de manera muy rápida podremos observar cómo va ejecutando los diferentes sets de pruebas.

- Finalmente, en el termina, podremos ver el resultado de las pruebas, en donde se hace la comparación del resultado esperado versus el resultado obtenido:

```
Operación: 10 más 5
Resultado esperado: 15.0
Resultado obtenido: 16.0
¡Prueba fallida!
Operación: 20 más 30
Resultado esperado: 50.0
Resultado obtenido: 51.0
¡Prueba fallida!
Operación: 10 menos 5
Resultado esperado: 5.0
Resultado obtenido: 5.0
¡Prueba exitosa!
Operación: 50 menos 30
Resultado esperado: 20.0
Resultado obtenido: 20.0
¡Prueba exitosa!
Operación: 5 por 5
Resultado esperado: 25.0
Resultado obtenido: 25.0
¡Prueba exitosa!
Operación: 6 por 7
Resultado esperado: 42.0
Resultado obtenido: 42.0
¡Prueba exitosa!
Operación: 100 entre 5
Resultado esperado: 20.0
Resultado obtenido: 20.0
¡Prueba exitosa!
Operación: 50 entre 2
Resultado esperado: 25.0
Resultado obtenido: 25.0
¡Prueba exitosa!
```

Fase III: Análisis de los Resultados

1. Análisis de lo obtenido en las pruebas

- Se evidencia una falla en la operación de la suma, el comportamiento que se observa, es que la operación está sumando un 1 adicional. Se recomienda la corrección inmediata.

```
Operación: 10 más 5
Resultado esperado: 15.0
Resultado obtenido: 16.0
```

```
¡Prueba fallida!
Operación: 20 más 30
Resultado esperado: 50.0
Resultado obtenido: 51.0
¡Prueba fallida!
```

1.1. Corrección del defecto, repetición de las pruebas y análisis de resultados

Se corrige el defecto en la línea correspondiente a la suma, en el programa calculadora.py

Antes:

```
if operator == 'sumar':
    result = num1 + num2 + 1
```

Ahora:

```
if operator == 'sumar':
    result = num1 + num2
```

1.2. Se realiza un nuevo plan de pruebas:

```
# Pruebas de adición
probar_operacion(driver, 12, 5, 0, '17.0')
probar_operacion(driver, 200, 30, 0, '230.0')

# Pruebas de sustracción
probar_operacion(driver, 12, 5, 1, '7.0')
probar_operacion(driver, 40, 30, 1, '10.0')

# Pruebas de multiplicación
probar_operacion(driver, 5, 6, 2, '30.0')
probar_operacion(driver, 7, 7, 2, '49.0')

# Pruebas de división
probar_operacion(driver, 100, 2, 3, '50.0')
probar_operacion(driver, 40, 2, 3, '20.0')
```

Y se obtiene el siguiente resultado:

```
Operación: 12 más 5
Resultado esperado: 17.0
Resultado obtenido: 17.0
¡Prueba exitosa!
Operación: 200 más 30
Resultado esperado: 230.0
```

```
Resultado obtenido: 230.0
¡Prueba exitosa!
Operación: 12 menos 5
Resultado esperado: 7.0
Resultado obtenido: 7.0
¡Prueba exitosa!
Operación: 40 menos 30
Resultado esperado: 10.0
Resultado obtenido: 10.0
¡Prueba exitosa!
Operación: 5 por 6
Resultado esperado: 30.0
Resultado obtenido: 30.0
¡Prueba exitosa!
Operación: 7 por 7
Resultado esperado: 49.0
Resultado obtenido: 49.0
¡Prueba exitosa!
Operación: 100 entre 2
Resultado esperado: 50.0
Resultado obtenido: 50.0
¡Prueba exitosa!
Operación: 40 entre 2
Resultado esperado: 20.0
Resultado obtenido: 20.0
¡Prueba exitosa!
```

Resultado Final:

Se concluye que la aplicación no tiene errores, y puede estar lista para salir a producción.