Escuela Superior de Ingeniería y Tecnología
CURSO DE ADAPTACIÓN AL GRADO DE INFORMÁTICA

Actividad 3 Grupal

Diseño de Pruebas de Software

Cristina Cacho Martín
Patricia Casas Vázquez
Irene Dacosta Guisado
José Ignacio Bravo Vicente

Índice

Asignatura	Datos del alumno	Fecha
Procesos en Ingeniería del Software	**Apellidos**:	15/01/2025
	Nombre:	

Introducción

Este documento describe el **desarrollo y diseño de pruebas de una calculadora básica** implementada en Python. El objetivo principal de este proyecto es aprender y aplicar conceptos relacionados con las pruebas unitarias, asegurando la calidad del software mediante la validación de cada funcionalidad implementada.

La calculadora incluye las cuatro operaciones matemáticas básicas: suma, resta, multiplicación y división. Las pruebas unitarias han sido diseñadas utilizando la librería *unittest*. Estas pruebas cubren diversos casos, incluyendo números positivos, negativos, cero o errores como la división por cero.

Asignatura	Datos del alumno	Fecha
Procesos en Ingeniería del Software	**Apellidos**:	15/01/2025
	Nombre:	

Especificación del Proyecto

La práctica consiste en el desarrollo de una calculadora básica en Python y la implementación de pruebas unitarias para validar su correcto funcionamiento. A continuación, se describen las características principales del proyecto:

FUNCIONES IMPLEMENTADAS

La calculadora incluye las siguientes operaciones matemáticas básicas:

- **Suma:** Toma dos números y devuelve su suma.
- **Resta:** Toma dos números y devuelve su diferencia.
- **Multiplicación:** Toma dos números y devuelve su producto.
- **División:** Toma dos números y devuelve su cociente, manejando el caso especial de la división por cero para evitar errores.

LIBRERÍA DE PRUEBAS

Se utiliza la librería estándar de Python *unittest* para definir pruebas unitarias que validen cada operación de la calculadora y cubran casos de prueba variados, incluyendo:

- Números positivos.
- Números negativos.
- Uso del cero como operandos.
- Manejo de errores, como la división por cero.

REQUISITOS

- Implementar la calculadora en una clase o como funciones independientes.

Asignatura	Datos del alumno	Fecha
Procesos en Ingeniería del Software	**Apellidos**:	15/01/2025
	Nombre:	

- Diseñar un mínimo de tres casos de prueba por cada operación matemática, asegurando que contemplan entradas diversas y casos límite.
- Escribir comentarios en el código para describir la funcionalidad de cada operación y los casos de prueba diseñados.

Asignatura	Datos del alumno	Fecha
Procesos en Ingeniería del Software	**Apellidos**:	15/01/2025
	Nombre:	

Estructura del Código

El proyecto está organizado en una estructura sencilla que separa el código fuente de las pruebas unitarias. A continuación, se detallan sus principales componentes:

- `calculadora_basica.py`: archivo principal que gestiona la interacción con el usuario y permite realizar las operaciones de suma, resta, multiplicación y división.
- `operaciones.py`: módulo que define las funciones para cada operación matemática: `sumar(a, b)`, `restar(a, b)`, `multiplicar(a, b)` y `dividir(a, b)`.
- `test_operaciones.py`: contiene las pruebas unitarias para las funciones definidas en `operaciones.py`, utilizando la biblioteca `unittest` de Python.
- `README.md`: archivo que proporciona una descripción general del proyecto, incluyendo su propósito, estructura y cómo ejecutarlo.

Asignatura	Datos del alumno	Fecha
Procesos en Ingeniería del Software	**Apellidos**:	15/01/2025
	Nombre:	

Desarrollo

El desarrollo de este proyecto incluye la implementación de una calculadora básica en Python y el diseño de pruebas unitarias con `unittest` para garantizar su funcionamiento.

IMPLEMENTACIÓN DE LA CALCULADORA

La implementación de la calculadora se realizó en el lenguaje de programación Python, siguiendo una arquitectura modular que separa la lógica de las operaciones matemáticas, la interacción con el usuario y las pruebas unitarias. A continuación, se detalla cada aspecto de la implementación:

Módulo de Operaciones

El módulo `operaciones.py` contiene las funciones principales que realizan las operaciones matemáticas básicas: suma, resta, multiplicación y división. Estas funciones están diseñadas para recibir dos argumentos y devolver el resultado correspondiente. Se incluyó un manejo de excepciones en la función `dividir()` para evitar errores en caso de división por cero.

Ejemplo de implementación de las funciones:

```
def sumar(a, b):
    """
    Calcula la suma de dos números.
    """
    return a + b


def dividir(a, b):
    """
    Calcula el cociente de dos números.
    Maneja la división por cero lanzando una excepción.
    """
    if b == 0:
```

```
        raise ValueError("El divisor no puede ser cero.")

    return a / b
…
```

Aplicación Principal

El archivo `calculadora.py` es el punto de entrada de la aplicación. Este módulo utiliza las funciones definidas en `operaciones.py` para realizar los cálculos. La interacción con el usuario se gestiona a través de un menú que permite seleccionar la operación deseada e ingresar los números de entrada.

Flujo principal de la aplicación:

1. Se muestra un banner introductorio al inicio de la ejecución.
2. El usuario selecciona una operación del menú.
3. Se solicitan los números para la operación seleccionada.
4. Se ejecuta la operación correspondiente
5. Se muestra el resultado al usuario.

Ejemplo de código del flujo principal:

```
def main():
    print("Bienvenido a la Calculadora Básica")
    print("Seleccione una operación:")

    print("1. Sumar")
    print("2. Restar")
    print("3. Multiplicar")
    print("4. Dividir")

    try:
        opcion = int(input("Ingrese el número de la operación: "))

        num1 = float(input("Ingrese el primer número: "))
        num2 = float(input("Ingrese el segundo número: "))

        if opcion == 1:
            print(f"Resultado: {sumar(num1, num2)}")
        elif opcion == 2:
            print(f"Resultado: {restar(num1, num2)}")
        elif opcion == 3:
```

```
            print(f"Resultado: {multiplicar(num1, num2)}")
        elif opcion == 4:
            print(f"Resultado: {dividir(num1, num2)}")
        else:
            print("Opción no válida.")

    except ValueError:
        print("Por favor, introduce valores numéricos válidos.")

    except Exception as e:
        print(f"Error: {e}")


if __name__ == "__main__":
    main()
```

DISEÑO DE LAS PRUEBAS UNITARIAS

Para garantizar que las funciones de la calculadora se ejecutan correctamente, se diseñaron varias pruebas unitarias utilizando el módulo `unittest`. Estas pruebas validan tanto los casos normales como los límites, como la división por cero.

Metodología

El diseño de las pruebas siguió los siguientes pasos:

1. **Identificación de casos de prueba:** Para cada función matemática, se identificaron los siguientes tipos de caso:
   - Casos normales (números positivos, negativos y cero).
   - Casos límite o excepcionales (como la división por cero).
2. **Definición de resultados esperados:** Se determinó el resultado correcto para cada entrada proporcionada.
3. **Automatización de pruebas:** Se definieron pruebas automatizadas en una clase test llamada `TestOperaciones`, que hereda de `unittest.TestCase`.

Organización de las Pruebas

Las pruebas se organizaron en el archivo `test_operaciones.py` que incluye:

- Un método `setUp()` donde se definen los datos iniciales y casos de prueba.
- Un método `test_operaciones()` que valida todas las funciones matemáticas utilizando subtests para cada caso específico.

Ejemplo del método `setUp()`:

```
def setUp(self):
    """
    Configura los datos iniciales para las pruebas.
    Define los casos normales y los casos que generan errores.
    """
    self.operaciones = {
        "sumar": {
            "func": sumar,
            "tests": [
                {"args": (10, 4), "expected": 14},
                {"args": (-5, 7), "expected": 2},
                {"args": (0, 0), "expected": 0},
            ],
        },

        "dividir": {
            "func": dividir,
            "tests": [
                {"args": (10, 2), "expected": 5},
            ],
            "errors": [
                {"args": (5, 0), "exception": ValueError},
            ],
        },
    }
…
```

Casos de Prueba Normales

Para cada operación, se probaron escenarios representativos con entradas comunes:

- Suma, resta y multiplicación: Pruebas con números positivos, negativos y ceros.
- División: Pruebas con números enteros y decimales.

Ejemplo de prueba para la función sumar:

```
def test_sumar(self):
    for test in self.operaciones["sumar"]["tests"]:
        with self.subTest(args=test["args"]):
            self.assertEqual(sumar(*test["args"]), test["expected"])
```

Casos de Prueba Excepcionales

Se validaron errores previsibles, como la división por cero, para asegurarnos de que las funciones manejan correctamente las excepciones esperadas.

Ejemplo de prueba para la división por cero:

```
def test_dividir_error(self):
    for error_test in self.operaciones["dividir"]["errors"]:
        with self.subTest(args=error_test["args"]):
            with self.assertRaises(error_test["exception"]):
                dividir(*error_test["args"])
```

Ejecución de las Pruebas

Las pruebas unitarias se ejecutan utilizando el siguiente comando:

```
$ python -m unittest test_operaciones.py
```

Al ejecutarlas, se genera un reporte que indica qué pruebas pasaron correctamente y cuáles fallaron, proporcionando detalles específicos para facilitar la depuración.

Asignatura	Datos del alumno	Fecha
Procesos en Ingeniería del Software	**Apellidos**:	15/01/2025
	Nombre:	

Resultados

La implementación de la calculadora y el diseño de las pruebas unitarias produjeron resultados satisfactorios que demuestran la funcionalidad y la fiabilidad del sistema desarrollado. Se detallan los principales resultados obtenidos:

Ejecución de la Calculadora

A continuación, un ejemplo del flujo interactivo de la calculadora al ejecutarla:

```
$ python calculadora_basica.py

==========================================
            CALCULADORA BÁSICA
==========================================

Universidad Internacional de La Rioja

Curso: Adaptación al Grado de Informática
Asignatura: Procesos en Ingeniería del Software
Práctica: Diseño de pruebas de software

------------------------------------------

Seleccione una operación:

1. Sumar
2. Restar
3. Multiplicar
4. Dividir

Opción: 1

Ingrese el primer número: 10
Ingrese el segundo número: 5

Resultado: 15
```

Resultados de las Pruebas Unitarias

Se ejecutaron las pruebas unitarias definidas en `test_operaciones.py` para validar las funciones del módulo `operaciones.py`. Todas las pruebas fueron exitosas, tanto para los casos normales como para los excepcionales.

El comando para ejecutar las pruebas es el siguiente:

```
$ python -m unittest test_operaciones.py
```

Y la salida obtenida:

```
Ran 10 tests in 0.003s
OK
```

El reporte confirma que todas las funciones de la calculadora operan conforme a lo esperado, incluso en escenarios límite como la división por cero.

Asignatura	Datos del alumno	Fecha
Procesos en Ingeniería del Software	**Apellidos**:	15/01/2025
	Nombre:	

Conclusiones

El desarrollo de esta práctica ayuda a comprender la importancia de las pruebas unitarias en el proceso de desarrollo de software. La implementación de la calculadora y sus respectivas pruebas mejora las siguientes capacidades:

- **Desarrollo de código robusto:** La utilización de pruebas unitarias ha demostrado ser esencial para detectar y corregir errores antes de que el software sea desplegado, como el manejo adecuado de casos especiales como la división por cero.
- **Calidad del Software:** Diseñar pruebas exhaustivas asegura que las funciones implementadas cumplen con los requisitos y se comportan de la manera esperada en los diferentes escenarios.
- **Trabajo Colaborativo:** La práctica grupal ha fomentado habilidades de comunicación y colaboración, lo que resulta fundamental para abordar proyectos de manera eficiente en un entorno profesional.
- **Mejora Continua:** Este ejercicio ha reforzado la importancia de adoptar buenas prácticas de desarrollo tales como la separación del código en módulos o el uso de herramientas estándar para pruebas.

En resumen, creemos que este proyecto no solo ha servido como un ejercicio práctico para aprender sobre pruebas unitarias, también ayudará a los integrantes del equipo a enfrentar retos similares en futuros desarrollos de software.

Asignatura	Datos del alumno	Fecha
Procesos en Ingeniería del Software	**Apellidos**:	15/01/2025
	Nombre:	

Valoración Individual

Tabla de valoración individual cumplimentada:

	Sí	**No**	**A veces**
Todos los miembros se han integrado al trabajo del grupo	x		
Todos los miembros participan activamente	x		
Todos los miembros respetan otras ideas aportadas	x		
Todos los miembros participan en la elaboración del informe	x		
Me he preocupado por realizar un trabajo cooperativo con mis compañeros	x		
Señala si consideras que algún aspecto del trabajo en grupo no ha sido adecuado		x	

Asignatura	Datos del alumno	Fecha
Procesos en Ingeniería del Software	**Apellidos**:	15/01/2025
	Nombre:	

Referencias

Python Software Foundation. (n.d.). *unittest — Unit testing framework*. Retrieved from https://docs.python.org/3/library/unittest.html

Documentación oficial de Python sobre el marco de pruebas unittest.

Asignatura	Datos del alumno	Fecha
Procesos en Ingeniería del Software	**Apellidos**:	15/01/2025
	Nombre:	

ANEXO I. Código fuente

El código de la aplicación es accesible a través del repositorio público:

https://github.com/joseignacio-bravo-unir/calculadora

ANEXO II. Documentación en línea

Para acceder a la documentación en línea (`docstrings`) de la librería podemos usar el comando `pydoc`. Por ejemplo:

```
$ pydoc operaciones
Help on module operaciones:

NAME
    operaciones - operaciones.py

DESCRIPTION
    Universidad Internacional de La Rioja

    Curso: Adaptación al Grado de Informática
    Asignatura: Procesos en Ingeniería del Software
    Práctica: Diseño de pruebas de software, pruebas unitarias

    Descripción:
        Este módulo contiene las funciones necesarias para realizar
        operaciones matemáticas básicas: suma, resta, multiplicación
        y división.

    Funciones:
        - sumar(a, b): Calcula la suma de dos números.
        - restar(a, b): Calcula la diferencia entre dos números.
        - multiplicar(a, b): Calcula el producto de dos números.
        - dividir(a, b): Calcula el cociente de dos números, manejando
          errores como la división por cero.

    Uso:
        Importa este módulo en otros scripts o proyectos para utilizar las
        funciones matemáticas básicas.

    Ejemplo:
        >>> from operaciones import sumar, restar, multiplicar, dividir
        >>> sumar(2, 3)
        5
        >>> dividir(10, 2)
        5.0
…
```

Asignatura	Datos del alumno	Fecha
Procesos en Ingeniería del Software	**Apellidos**:	15/01/2025
	Nombre:	

ANEXO III. Uso de Github

Para facilitar el trabajo en equipo, usaremos GitHub como repositorio de código y para control de versiones. Partiremos del código de ejemplo que el profesor de la asignatura ha compartido a través de su repositorio personal:

https://github.com/jarturomora/bootcamp-demo

Haremos una copia (fork) de trabajo del repositorio. La llamaremos 'calculadora'. Seguiremos los siguientes pasos:

1. Creamos una cuenta GitHub: '*nombre-apellido*-unir'
2. Creación de un token de acceso. Para ello, vamos a la sección de configuración 'tokens' (https://github.com/settings/tokens), creamos un nuevo token y le asignamos permisos para 'repo'.
3. Como siguiente paso, creamos un fork del proyecto "bootcamp-demo" en la URL https://github.com/jarturomora/bootcamp-demo. Le asignamos el nombre 'calculadora' y la descripción 'Actividad grupal para diseño de pruebas de software'.
4. Descargamos (clonamos) el proyecto a nuestro equipo:

```
$ git clone https://github.com/joseignacio-bravo-unir/calculadora.git
```

5. Para verificar que todo es correcto, editamos el fichero README.md y cambiamos la descripción a:

```
# Calculadora con Pruebas Unitarias
…
```

6. Desde el directorio 'calculadora', aplicamos los cambios y subimos al repositorio:

```
$ cd calculadora
$ git status
$ git add .
$ git commit -m "Actualización de README.md para verificar repositorio"
$ git push origin main
```

7. Nos pedirá el usuario y contraseña (usaremos el token creado).

Usuario: `nombre-apellido-unir`
Contraseña: `ghp_xxxx…`

8. Verificamos que la subida es correcta:

```
$ git remote -v
```

9. Accedemos a Github y observamos que los cambios se han aplicado. A partir de aquí, comenzamos a trabajar en la práctica.
10. Para unificar los cambios, podemos crear una rama individual por usuario sobre la que trabajar:

```
$ git checkout -b mi-rama
$ git add .
$ git commit -m "Descripción de los cambios"
$ git push origin mi-rama
```

11. Posteriormente podemos hacer un '*pull request*' a la rama principal para consolidar todos los cambios.

Asignatura	Datos del alumno	Fecha
Procesos en Ingeniería del Software	**Apellidos**:	15/01/2025
	Nombre:	

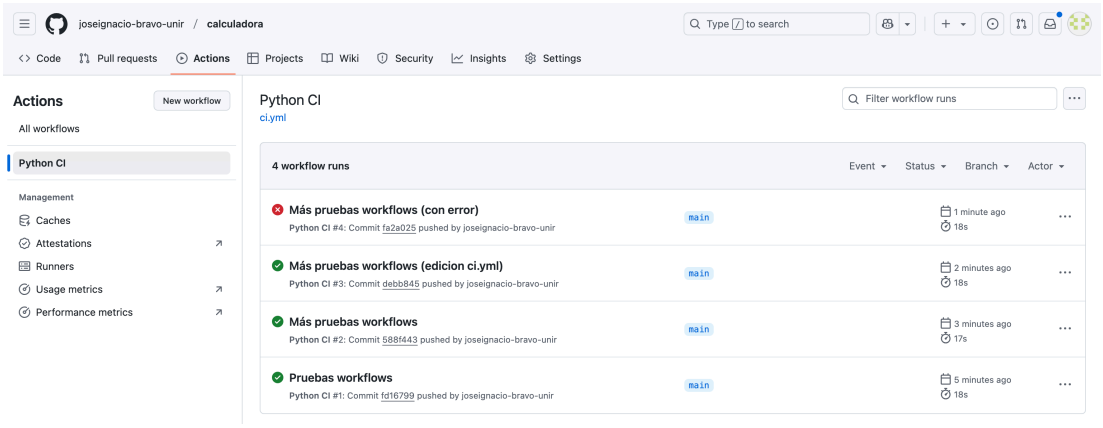
ANEXO IV. Pruebas de Integración Continua (CI)

La *Integración Continua* (CI) es una práctica de desarrollo de software en la que los cambios realizados en el código se integran frecuentemente en un repositorio compartido. Cada integración es verificada automáticamente mediante pruebas para detectar errores lo antes posible. Sus principales características son:

- Automatiza la ejecución de tests cada vez que se realizan cambios en el código (*push* o *pull request*).
- Asegura que el código nuevo no rompa funcionalidades existentes.
- Proporciona retroalimentación rápida a los desarrolladores sobre el estado del código.

GitHub Actions es una solución integrada que permite configurar *workflows* o flujos de trabajo para automatizar tareas tales como la ejecución de pruebas unitarias. El código original de la calculadora incluye un fichero *ci.yaml* de configuración que permite habilitar la integración continua en nuestro repositorio *calculadora*:

***Ilustración 1**. Pruebas de integración continua (CI) con Github Actions*



Fuente: elaboración propia.