Resolución de ejercicio 1

Objetivo: Aprender a realizar **pruebas unitarias básicas** en varios lenguajes de programación.

- **Lenguajes:** JavaScript, Python, C#, Java.

Ejemplo: Pruebas de división correcta y división por cero.

Realizaremos este laboratorio con la ayuda de **ChatGPT.**

JavaScript

Configuración del entorno

1. Verificación de Instalación:

```bash
node --version
npm --version
```

2. Instalación de Jest:

```bash
npm install --save-dev jest
```

Si el comando **node** o **npm** fallan, instalar Node.js.

JS

Creación de pruebas con Jest

javascript Copiar código

```javascript
// division.test.js
const { test, expect } = require('@jest/globals');

test('Correct division', () => {
  expect(10 / 2).toBe(5);
});

test('Division by zero', () => {
  expect(() => { 10 / 0 }).toThrow();
});
```

Copiar este código en un archivo con extensión ***js***.

Ejecución de pruebas con Jest

```bash
npx jest
```

Ventajas de Jest:

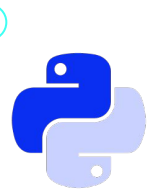
- **Simple configuración y uso.**
- **Flexibilidad y potencia:** Soporta *mocks*, aserciones avanzadas y más.
- **Integración con proyectos:** Ampliamente utilizado en proyectos modernos de JavaScript.

Python

Creación de pruebas

python Copiar código

```python
# test.py
import unittest

class TestDivision(unittest.TestCase):
    def test_correct_division(self):
        self.assertEqual(10 / 2, 5)

    def test_division_by_zero(self):
        with self.assertRaises(ZeroDivisionError):
            10 / 0

if __name__ == '__main__':
    unittest.main()
```



Ejecución de pruebas

bash Copiar código

```
python test.py
```

En Python, para realizar **pruebas unitarias básicas con el módulo `unittest`, no es necesario instalar librerías adicionales.** El módulo `unittest` viene incluido en la biblioteca estándar de Python, por lo que se puede comenzar a escribir y ejecutar pruebas unitarias sin necesidad de instalar nada adicional.

C#

Creación de pruebas

Verificación de Instalación de .NET Core (si este comando falla instalar dotnet).

bash Copiar código

```
dotnet --version
```

Creación de proyecto de pruebas unitarias:

bash Copiar código

```
dotnet new xunit -n MyUnitTestProject
cd MyUnitTestProject
```

csharp Copiar código

```csharp
// DivisionTests.cs
using NUnit.Framework;

[TestFixture]
public class DivisionTests
{
    [Test]
    public void CorrectDivision()
    {
        Assert.AreEqual(10 / 2, 5);
    }

    [Test]
    public void DivisionByZero()
    {
        Assert.Throws<System.DivideByZeroException>(() => { var result = 10 / 0; });
    }
}
```

Ejecución de pruebas

```bash
dotnet test
```

Este comando **`dotnet test` ejecutará todas las pruebas,** en el proyecto actual, utilizando el *framework* de pruebas especificado (en este caso, `xUnit.NET`).

Con estos pasos, se puede asegurar que el entorno esté correctamente configurado y listo para ejecutar pruebas unitarias en C# usando .NET Core.

Java

Creación de pruebas

java Copiar código

```java
// DivisionTests.java
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class DivisionTests {
    @Test
    public void correctDivision() {
        assertEquals(5, 10 / 2);
    }

    @Test(expected = ArithmeticException.class)
    public void divisionByZero() {
        int result = 10 / 0;
    }
}
```

Ejecución de pruebas

```bash
javac -cp junit-4.XX.jar DivisionTests.java
java -cp .:junit-4.XX.jar org.junit.runner.JUnitCore DivisionTests
```

Para poder ejecutar ese comando, se debe tener **Java instalado y el archivo *JAR* de JUnit,** que se puede descargar de: GitHub.com.

Se debe renombrar el ***junit-4.xx.jar*** con el nombre del ***jar*** que se descargó, que debe estar en el mismo directorio que el archivo ***.java***

Resolución de ejercicio 2

Utilizar **ChatGPT** para generar las clases solicitadas siguiendo el enunciado.

Copiar el enunciado del laboratorio y pedirle a ChatGPT, o al LLM que se prefiera, que **genere la clase `Cliente` en los distintos lenguajes.**

Respuestas de ChatGPT

Implementación en JavaScript

javascript Copiar código

```javascript
class Cliente {
    constructor(nombre, apellido, fechaNacimiento, email) {
        this._nombre = nombre;
        this._apellido = apellido;
        this._fechaNacimiento = new Date(fechaNacimiento);
        this._email = email;
    }

    get nombre() {
        return this._nombre;
    }
```

Implementación en Python

python Copiar código

```python
from datetime import date

class Cliente:
    def __init__(self, nombre, apellido, fecha_nacimiento, email):
        self._nombre = nombre
        self._apellido = apellido
        self._fecha_nacimiento = fecha_nacimiento
        self._email = email
```

Implementación en C#

csharp Copiar código

```csharp
using System;

public class Cliente {
    private string nombre;
    private string apellido;
    private DateTime fechaNacimiento;
    private string email;

    public Cliente(string nombre, string apellido, DateTime fechaNacimiento, string email)
        this.nombre = nombre;
```

Implementación en Java

java Copiar código

```java
import java.time.LocalDate;
import java.time.Period;

public class Cliente {
    private String nombre;
    private String apellido;
    private LocalDate fechaNacimiento;
    private String email;

    public Cliente(String nombre, String apellido, LocalDate fechaNacimiento, String email
        this.nombre = nombre;
```

Asegurar la consistencia en los constructores

Se deben **incluir las validaciones, en los constructores,** para asegurar que los datos sean consistentes.

Si ChatGPT no lo hace, se le debe indicar que lo corrija.

Veamos el código del siguiente *slide*.

JAVA

```java
public class Cliente {
   private String nombre;
   private String apellido;
   private LocalDate fechaNacimiento;
   private String email;

   public Cliente(String nombre, String apellido, LocalDate fechaNacimiento, String email) {
       if (nombre.isEmpty() || apellido.isEmpty()) {
           throw new IllegalArgumentException("Nombre y apellido mal.");
       }

       if (fechaNacimiento.isAfter(LocalDate.now())) {
           throw new IllegalArgumentException("La fecha de nacimiento mal");
       }

       if (!isValidEmail(email)) {
           throw new IllegalArgumentException("El email mal,");
       }
```

Asegurar la integridad en los *getters* y *setters*

Se puede hacer que las **propiedades** que se inicializan en el constructor sean **de solo lectura y solo se puedan acceder mediante los *getters*** para asegurar la consistencia de los objetos.

Si el valor que se recibe no coincide con lo esperado, se puede lanzar una excepción.

C#

```
public class Cliente {
    private string _email;

    private bool IsValidEmail(string email) {
        string pattern = @"^[^@\s]+@[^@\s]+\.[^@\s]+$";
        return Regex.IsMatch(email, pattern);
    }

    public string Email {
        get { return _email; }
        set {
            if (!IsValidEmail(value)) {
                throw new ArgumentException("Email incorrecto.");
            }
            _email = value;
        }
    }
}
```

Resolución del ejercicio 3

Se verificará el comportamiento con *Unit Test.*

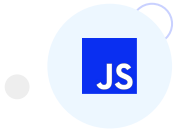
1. Utilizar **ChatGPT** para escribir y ejecutar las pruebas unitarias.
2. Luego, copiar el código en **Visual Studio Code** para verificar cada caso.

1. Unit Test

JavaScript

Generar las pruebas unitarias en JavaScript con ChatGPT.

Guardar tanto la clase como las pruebas unitarias y ejecutarlas.



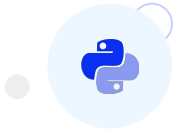
javascript — Copiar código

```javascript
const assert = require('assert');
const Cliente = require('./cliente'); // Asegúrate de que el nombre del archivo sea client

describe('Cliente', function () {
    let cliente;

    beforeEach(function () {
        cliente = new Cliente('Juan', 'Pérez', '1990-01-01', 'juan@empresa.com');
    });

    it('calcula la edad correctamente', function () {
        assert.strictEqual(cliente.calcularEdad(), new Date().getFullYear() - 1990);
    });

    it('determina la mayoría de edad correctamente', function () {
        assert.strictEqual(cliente.esMayorDeEdad(), true);
    });

    it('verifica el email empresarial correctamente', function () {
        assert.strictEqual(cliente.poseeMailEmpresarial(), true);
    });
});
```

Python

Generar las pruebas unitarias en Python con ChatGPT.

Guardar tanto la clase como las pruebas unitarias y ejecutarlas.



python Copiar código

```python
import unittest
from datetime import date
from cliente import Cliente  # Asegúrate de que el nombre del archivo sea cliente.py

class TestCliente(unittest.TestCase):
    def setUp(self):
        self.cliente = Cliente("Juan", "Pérez", date(1990, 1, 1), "juan@empresa.com")

    def test_calcularEdad(self):
        self.assertEqual(self.cliente.calcularEdad(), date.today().year - 1990)

    def test_esMayorDeEdad(self):
        self.assertTrue(self.cliente.esMayorDeEdad())

    def test_poseeMailEmpresarial(self):
        self.assertTrue(self.cliente.poseeMailEmpresarial())

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

C#

Generar las pruebas unitarias en C# con ChatGPT.

Guardar tanto la clase como las pruebas unitarias y ejecutarlas.

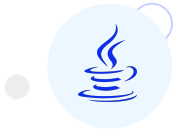
csharp Copiar código

```csharp
using NUnit.Framework;
using System;
using cliente; // Asegúrate de que el espacio de nombres y archivo sean correctos

[TestFixture]
public class ClienteTest {
    private Cliente cliente;

    [SetUp]
    public void SetUp() {
        cliente = new Cliente("Juan", "Pérez", new DateTime(1990, 1, 1), "juan@empresa.com
    }

    [Test]
    public void TestCalcularEdad() {
        Assert.AreEqual(DateTime.Today.Year - 1990, cliente.CalcularEdad());
    }

    [Test]
    public void TestEsMayorDeEdad() {
        Assert.IsTrue(cliente.EsMayorDeEdad());
    }

    [Test]
    public void TestPoseeMailEmpresarial() {
        Assert.IsTrue(cliente.PoseeMailEmpresarial());
    }
}
```

Java

Generar las pruebas unitarias en Java con ChatGPT.

Guardar tanto la clase como las pruebas unitarias y ejecutarlas.



java — Copiar código

```java
import org.junit.jupiter.api.Test;
import java.time.LocalDate;
import static org.junit.jupiter.api.Assertions.*;
import cliente.Cliente;  // Asegúrate de que el nombre del paquete y archivo sea correcto

public class ClienteTest {

    @Test
    public void testCalcularEdad() {
        Cliente cliente = new Cliente("Juan", "Pérez", LocalDate.of(1990, 1, 1), "juan@emp
        assertEquals(LocalDate.now().getYear() - 1990, cliente.calcularEdad());
    }

    @Test
    public void testEsMayorDeEdad() {
        Cliente cliente = new Cliente("Juan", "Pérez", LocalDate.of(1990, 1, 1), "juan@emp
        assertTrue(cliente.esMayorDeEdad());
    }

    @Test
    public void testPoseeMailEmpresarial() {
        Cliente cliente = new Cliente("Juan", "Pérez", LocalDate.of(1990, 1, 1), "juan@emp
        assertTrue(cliente.poseeMailEmpresarial());
    }
}
```

2. Se debe ejecutar el código localmente

Grabar los archivos en **Visual Studio Code,** verificar que compile y ejecutar las pruebas para verificar el **correcto funcionamiento de todo el código fuente.**