**Crear carpeta**

mkdir GIT

cd GIT

mkdir IS10A

cd IS10A

**Tipo consola en .net:\_**

dotnet new console -o src\ConsoleApp

dotnet new classlib -o src/Application

dotnet add src\ConsoleApp reference src\Application

**Tipo web en .net:**

dotnet new web -o src\WebApp

dotnet add src\WebApp reference src\Application

**Abrir el Directorio con vs code**

code .

**Eliminar y crear archivos necesarios**:

rm src\Application\Class1.cs

code src\Application\Triangulo.cs

code src\Application\PrismaTriangular.cs

code src\WebApp\Pages\Index.cshtml

code src\ConsoleApp\Program.cs

code src\WebApp\Program.cs

[**src\Application\Triangulo.cs.**](https://gist.github.com/jlcarrillog/2f55790fa7f32b01d6a2d851e2beffea#file-02-src-application-cuadrado-cs)

namespace Application

{

    public class Triangulo

    {

        // Método para calcular el perímetro de un triángulo

        public static double Perimetro(double a, double b, double c)

        {

            double perimetro = a + b + c;

            return perimetro;

        }

        // Método para calcular el área de un triángulo

        public static double Area(double b, double d)

        {

            double area = (b \* d) / 2;

            return area;

        }

    }

}

**src\Application\PrismaTriangular.cs**

namespace Application

{

    public static class PrismaTriangular

    {

        // Método para calcular el volumen de un prisma triangular

        public static double Volumen(double areaBase, double alturaPrisma)

        {

            return areaBase \* alturaPrisma;

        }

    }

}

**04-src\ConsoleApp\Program.cs**

using Application;

class Program

{

    static void Main(string[] args)

    {

        // Lados del triángulo base

        double ladoA = 5.0; // Primer lado del triángulo

        double ladoB = 6.0; // Base del triángulo

        double ladoC = 7.0; // Tercer lado del triángulo

        // Altura del triángulo correspondiente a la base 'ladoB'

        double alturaTriangulo = 4.0;

        // Altura del prisma

        double alturaPrisma = 10.0;

        // Cálculo del perímetro del triángulo

        double perimetro = Triangulo.Perimetro(ladoA, ladoB, ladoC);

        // Cálculo del área de la base triangular

        double area = Triangulo.Area(ladoB, alturaTriangulo);

        // Cálculo del volumen del prisma triangular

        double volumen = PrismaTriangular.Volumen(area, alturaPrisma);

        // Salida de los resultados

        Console.WriteLine($"El perímetro de la base de un Prisma Triangular con lados de {ladoA}cm, {ladoB}cm y {ladoC}cm es {perimetro}cm");

        Console.WriteLine($"El área de la base de un Prisma Triangular con base de {ladoB}cm y altura de {alturaTriangulo}cm es {area}cm²");

        Console.WriteLine($"El volumen de un Prisma Triangular con área de base de {area}cm² y altura de {alturaPrisma}cm es {volumen}cm³");

    }

}

**src\WebApp\Pages\Index.cshtml**

@page

@{

    // Lados del triángulo

    double ladoA = 5; // Corresponde a 'a'

    double ladoB = 3; // Corresponde a 'b' (base del triángulo)

    double ladoC = 4; // Corresponde a 'c'

    // Altura del triángulo correspondiente a la base 'b'

    double alturaTriangulo = 4;

    // Altura del prisma

    double alturaPrisma = 10;

    // Cálculo del perímetro del triángulo

    double perimetro = Application.Triangulo.Perimetro(ladoA, ladoB, ladoC);

    // Cálculo del área de la base triangular

    double area = Application.Triangulo.Area(ladoB, alturaTriangulo);

    // Cálculo del volumen del prisma triangular (usando el área de la base)

    double volumen = Application.PrismaTriangular.Volumen(area, alturaPrisma);

}

El perímetro de la base de un triángulo con lados de @(ladoA)cm, @(ladoB)cm y @(ladoC)cm es @(perimetro)cm

<br />

El área de la base de un triángulo con base de @(ladoB)cm y altura de @(alturaTriangulo)cm es @(area)cm²

<br />

El volumen de un prisma triangular con un área de base de @(area)cm² y una altura de @(alturaPrisma)cm es @(volumen)cm³

**src\WebApp\Program.cs**

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddRazorPages();

var app = builder.Build();

app.MapRazorPages();

app.Run();

**cmd.ps1**

# Ejecutar el proyecto modificado

dotnet run --project src\ConsoleApp

# Ejecutar el proyecto modificado, se deberá abrir un explorador web y navegar a al dirección https://localhost:5001

dotnet run --project src\WebApp

# Para detener el servidor se debe presionar Ctrl+C

Dotnet new unit -o test/Application.UnitTest

Dotnet add test/Application.UnitTest reference src/Application

**2DA PARTE DEL PROYECTO**

**Pruebas unitarias**

Dotnet new xunit -o test/Application.UnitTest

Dotnet add test/Application.UnitTest reference src/Application

Code src\Application.UnitTest\Triangulo.cs

code src\Application.UnitTest\PrismaTriangular.cs

**Xunit Application Traingulo**

using System;

using Xunit;

using Application;

namespace Application.UnitTest

{

    public class UnitTestTriangulo

    {

        // Pruebas para el Perímetro

        [Theory]

        [InlineData(3, 4, 5, 12)] // Perímetro = a + b + c

        [InlineData(6, 8, 10, 24)]

        [InlineData(5, 5, 5, 15)]

        [InlineData(7, 24, 25, 56)]

        [InlineData(10, 10, 15, 35)]

        public void TestPerimetro(double a, double b, double c, double perimetroEsperado)

        {

            // Act - Realizar

            double resultado = Triangulo.Perimetro(a, b, c);

            // Assert - Verificar

            Assert.Equal(perimetroEsperado, resultado, 5); // Tolerancia de 5 decimales

        }

        // Pruebas para el Área

        [Theory]

        [InlineData(6, 4, 12)] // Área = (base \* altura) / 2

        [InlineData(10, 5, 25)]

        [InlineData(8, 3, 12)]

        [InlineData(15, 10, 75)]

        [InlineData(7, 6, 21)]

        public void TestArea(double baseTriangulo, double altura, double areaEsperada)

        {

            // Act - Realizar

            double resultado = Triangulo.Area(baseTriangulo, altura);

            // Assert - Verificar

            Assert.Equal(areaEsperada, resultado, 5); // Tolerancia de 5 decimales

        }

    }

}

**Xunit Application PrismaTriangular**

using System;

using Xunit;

using Application;

namespace Application.UnitTest

{

    public class UnitTestPrismaTriangular

    {

        // Pruebas para el Volumen

        [Theory]

        [InlineData(6, 10, 60)]   // Volumen = Área \* Altura del prisma

        [InlineData(12, 5, 60)]

        [InlineData(7.5, 8, 60)]

        [InlineData(20, 15, 300)]

        [InlineData(25, 12, 300)]

        public void TestVolumen(double areaBase, double alturaPrisma, double volumenEsperado)

        {

            // Act - Realizar

            double resultado = PrismaTriangular.Volumen(areaBase, alturaPrisma);

            // Assert - Verificar

            Assert.Equal(volumenEsperado, resultado, 5); // Tolerancia de 5 decimales

        }

    }

}

**Pruebas de Integracion Area, perimetro y volumen**

using Xunit;

using Application;

namespace Application.IntegrationTest

{

    public class IntegrationTestApplication

    {

        [Fact]

        public void TestCalculoAreaBase()

        {

            double baseTriangulo = 5.0;

            double alturaTriangulo = 6.0;

            double areaBase = Triangulo.Area(baseTriangulo, alturaTriangulo);

            Assert.Equal(15.0, areaBase, 2);

        }

        [Fact]

        public void TestCalculoVolumen()

        {

            double baseTriangulo = 5.0;

            double alturaTriangulo = 6.0;

            double alturaPrisma = 10.0;

            double areaBase = Triangulo.Area(baseTriangulo, alturaTriangulo);

            double volumen = PrismaTriangular.Volumen(areaBase, alturaPrisma);

            Assert.Equal(150.0, volumen, 2);

        }

        [Fact]

        public void TestPerimetroBaseYAreaBase()

        {

            double lado1 = 3.0;

            double lado2 = 4.0;

            double lado3 = 5.0;

            double baseTriangulo = 5.0;

            double alturaTriangulo = 6.0;

            double perimetro = Triangulo.Perimetro(lado1, lado2, lado3);

            double areaBase = Triangulo.Area(baseTriangulo, alturaTriangulo);

            Assert.True(perimetro > 0 && areaBase > 0);

        }

        [Fact]

        public void TestVolumenNoNegativo()

        {

            double areaBase = 15.0;

            double alturaPrisma = 10.0;

            double volumen = PrismaTriangular.Volumen(areaBase, alturaPrisma);

            Assert.True(volumen > 0);

        }

        [Fact]

        public void TestAreaBaseComparacion()

        {

            double basePequena = 3.0;

            double alturaPequena = 4.0;

            double baseGrande = 5.0;

            double alturaGrande = 6.0;

            double areaPequena = Triangulo.Area(basePequena, alturaPequena);

            double areaGrande = Triangulo.Area(baseGrande, alturaGrande);

            Assert.True(areaGrande > areaPequena);

        }

    }

}

**Pruebas de Integracion WEB**

using System.Net;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc.Testing;

using Xunit;

namespace WebApp.IntegrationTest

{

    public class IntegrationTestWeb : IClassFixture<WebApplicationFactory<Program>>

    {

        private readonly WebApplicationFactory<Program> \_factory;

        public IntegrationTestWeb(WebApplicationFactory<Program> factory)

        {

            \_factory = factory;

        }

        [Fact]

        public async Task TestHomePage()

        {

            var client = \_factory.CreateClient();

            var response = await client.GetAsync("/");

            Assert.Equal(HttpStatusCode.OK, response.StatusCode);

        }

        [Fact]

        public async Task Test404ErrorPage()

        {

            var client = \_factory.CreateClient();

            var response = await client.GetAsync("/nonexistent-page");

            Assert.Equal(HttpStatusCode.NotFound, response.StatusCode);

        }

        [Fact]

        public async Task TestStaticFile404()

        {

            var client = \_factory.CreateClient();

            var response = await client.GetAsync("/css/invalid-file.css");

            Assert.Equal(HttpStatusCode.NotFound, response.StatusCode);

        }

        [Fact]

        public async Task TestInvalidEndpoint404()

        {

            var client = \_factory.CreateClient();

            var response = await client.GetAsync("/api/invalid-endpoint");

            Assert.Equal(HttpStatusCode.NotFound, response.StatusCode);

        }

        [Fact]

        public async Task Test404ForInvalidQueryString()

        {

            var client = \_factory.CreateClient();

            var response = await client.GetAsync("/?invalid=query");

            Assert.Equal(HttpStatusCode.OK, response.StatusCode); // Página existe

        }

    }

}

**Git - Clonar repositorio remoto**

Git clone https://github.com/Danyepro/IS10A.git

**Git**

**Modificar y subir al repositorio remoto**

Git add .

Git commit -m “mensaje”

Git congif –global user.email [user@mail.com](mailto:user@mail.com)

Git config –global user.name “Fname SName”

Git commit -m “mensaje”

Git push