**Crear carpeta**

cd Documents

mkdir Documentes IS

cd Documents/ISc

mkdir IS10A

cd IS10A

cd Documents/IS/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\Circulo.cs

code src\Application\PrismaCircular.cs

code src\WebApp\Pages\Index.cshtml

code src\ConsoleApp\Program.cs

code src\WebApp\Program.cs

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

namespace Application;

public class Circulo

{

public static double Perimetro(double radio)

{

double perimetro = 2 \* Math.PI \* radio;

return perimetro;

}

public static double Area(double radio)

{

double area = Math.PI \* radio \* radio;

return area;

}

}

**src\Application\PrismaCircular.cs**

namespace Application;

public class PrismaCircular

{

public static double Volumen(double area, double altura)

{

double volumen = area \* altura;

return volumen;

}

}

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

using Application;

class Program

{

static void Main(string[] args)

{

double radio = 5.0;

double altura = 10.0;

double perimetro = Circulo.Perimetro(radio);

double area = Circulo.Area(radio);

double volumen = PrismaCircular.Volumen(area, altura);

Console.WriteLine($"El perímetro de la base de un Prisma Circular con radio de {radio}cm es {perimetro}cm");

Console.WriteLine($"El área de la base de un Prisma Circular con radio de {radio}cm es {area}cm²");

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

}

}

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

@page

@{

    double radio = 5.0;

    double altura = 10.0;

    double perimetro = Application.Circulo.Perimetro(radio);

    double area = Application.Circulo.Area(radio);

    double volumen = Application.PrismaCircular.Volumen(area, altura);

}

<h1>Cálculo de Prisma Circular</h1>

<p>El perímetro de la base de un Prisma Circular con radio de @(radio)cm es @(perimetro)cm</p>

<p>El área de la base de un Prisma Circular con radio de @(radio)cm es @(area)cm²</p>

<p>El volumen de un Prisma Circular con un área de @(area)cm² y una altura de @(altura)cm es @(volumen)cm³</p>

**src\WebApp\Program.cs**

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddRazorPages();

var app = builder.Build();

app.MapRazorPages();

app.Run();

public partial class Program { }

**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\Circulo.cs

code src\Application.UnitTest\PrismaCircular.cs

**Xunit Application - Circulo**

using System;

using Xunit;

using Application;

namespace Application.UnitTest

{

public class UnitTestCirculo

{

// Pruebas para el Perímetro

[Theory]

[InlineData(3, 18.84956)] // Perímetro = 2 \* π \* radio

[InlineData(4, 25.13272)]

[InlineData(5, 31.41593)]

[InlineData(6, 37.69911)]

[InlineData(7, 43.98230)]

public void TestPerimetro(double radio, double perimetroEsperado)

{

// Act - Realizar

double resultado = Circulo.Perimetro(radio);

// Assert - Verificar

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

}

// Pruebas para el Área

[Theory]

[InlineData(3, 28.27433)] // Área = π \* radio^2

[InlineData(4, 50.26544)]

[InlineData(5, 78.53975)]

[InlineData(6, 113.0976)]

[InlineData(7, 153.93804)]

public void TestArea(double radio, double areaEsperada)

{

// Act - Realizar

double resultado = Circulo.Area(radio);

// Assert - Verificar

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

}

}

}

**Xunit Application – PrismaCircular**

using System;

using Xunit;

using Application;

namespace Application.UnitTest

{

public class UnitTestPrismaCircular

{

// Pruebas para el Volumen

[Theory]

[InlineData(3.14159, 4, 50.26544)] // Volumen = Área \* Altura

[InlineData(3.14159, 6, 113.0976)]

[InlineData(3.14159, 10, 314.159)]

[InlineData(3.14159, 12, 452.38934)]

[InlineData(3.14159, 15, 706.8583)]

public void TestVolumen(double pi, double radio, double volumenEsperado)

{

// Act - Realizar

double resultado = PrismaCircular.Volumen(areaEsperada, radio);

// Assert - Verificar

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

}

}

}

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

dotnet new xunit -o test/Application.IntegrationTest dotnet add test/Application.IntegrationTest reference src/Application

dotnet new xunit -o test/WebApp.IntegrationTest dotnet add test/WebApp.IntegrationTest reference src/Application

dotnet add test/WebApp.IntegrationTest reference src/WebApp

code test/Application.IntegrationTest/IntegrationTestApplication.cs

using Xunit;

using Application;

namespace Application.IntegrationTest

{

public class IntegrationTestApplication

{

[Fact]

public void TestCalculoArea()

{

double radio = 5.0;

double area = Circulo.Area(radio);

Assert.Equal(78.539749999999998, area, 2);

}

[Fact]

public void TestCalculoVolumen()

{

double radio = 5.0;

double altura = 10.0;

double area = Circulo.Area(radio);

double volumen = PrismaCircular.Volumen(area, altura);

Assert.Equal(785.39750000000004, volumen, 2);

}

[Fact]

public void TestPerimetroArea()

{

double radio = 5.0;

double perimetro = Circulo.Perimetro(radio);

double area = Circulo.Area(radio);

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

}

[Fact]

public void TestVolumenNoNegativo()

{

double area = 78.53975;

double altura = 10.0;

double volumen = PrismaCircular.Volumen(area, altura);

Assert.True(volumen > 0);

}

[Fact]

public void TestAreaComparacion()

{

double radioPequeno = 3.0;

double radioGrande = 5.0;

double areaPequena = Circulo.Area(radioPequeno);

double areaGrande = Circulo.Area(radioGrande);

Assert.True(areaGrande > areaPequena);

}

}

}

**Pruebas de Integracion WEB**

code test/WebApp.IntegrationTest/IntegrationTestWeb.cs

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

}

}

}

dotnet test test/Application.IntegrationTest

dotnet test test/WebApp.IntegrationTest

**Git**

**Clonar repositorio remoto**

Git clone <https://dev.azure.com/unitec-is/Project/_git/Project>

Cd Project

Dotnet new console

Code.

**Git**

**Modificar y subir al repositorio remoto**

Git add .

Git commit -a -m “Init”

Imagen que contiene Texto

Descripción generada automáticamente

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

Git config –global user.name “Fname SName”

Git commit -a -m “Iniit”

Git push

**Git**

**Bajar los cambios del repositorio remoto**

Git pull