Tesztelés

A tesztelés, mint elméletben láttuk, igen komoly dolog. A segítségével sok kellemetlenséget megelőzhetünk.

Először nézzünk egy rendkívül egyszerű **Unit** tesztet. Persze ehhez először adni kellene valamilyen definíciószerűséget, hogy tudjuk, egyáltalán mi az a **unit** teszt.
Lássuk: A **unit teszt** egy olyan automatizált teszt, amely a program egyetlen, legkisebb önálló egységének (unit) a viselkedését ellenőrzi izolált környezetben. **C#**-ban a **unit** egy **metódus** és a teszttel ennek a viselkedését vizsgáljuk. Azt vizsgáljuk, hogy adott bemenet esetén mindig adott kimenet legyen. A metódus belső logikája vagy implementálása (megvalósítása) lényegtelen a teszt szempontjából.

Unit tesztek lehetőséges teszt frameworkjei:

- **xUnit** (leggyakoribb .NET Core-nál)
- **NUnit**
- **MSTest** (Microsoft saját megoldása)

Mi itt az **xUnit**-ot nézzük (egyszerű matematikai műveleteket tesztelünk)

Létrehozunk egy konzolos projektet **Unit_teszt_01** néven. Nem írunk bele semmit, hisz nem ezt teszteljük majd!

Utána létrehozunk egy osztályt **Szamologep** néven:

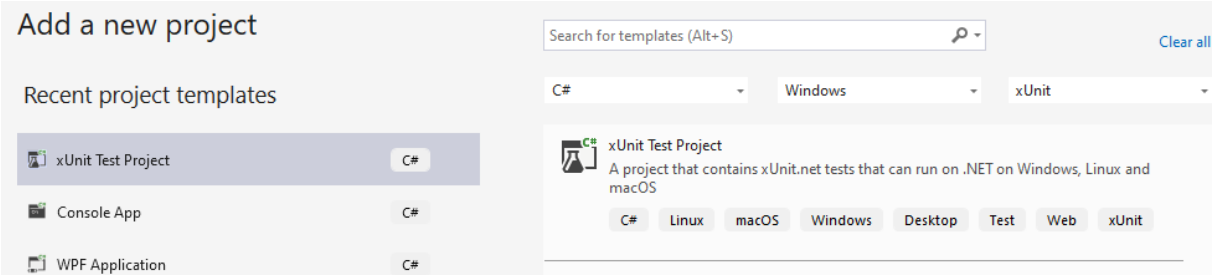
```
public class Szamologep
{
    public int Osszeadas(int a, int b)
    {
        return a + b;
    }

    public int Kivonas(int a, int b)
    {
        return a - b;
    }

    public int Szorzas(int a, int b)
    {
        return a * b;
    }

    public int Osztas(int a, int b)
    {
        if (b == 0)
            throw new DivideByZeroException();

        return a / b;
    }
}
```

Ennek az osztálynak a metódusait fogjuk tesztelni!

Teszt projekt létrehozása

Visual Studio-ban:

1. Solution → **Add** → **New Project**
2. Válaszd: **xUnit Test Project**
3. Add hozzá reference-ként a fő projektet (**Unit_teszt_01)**

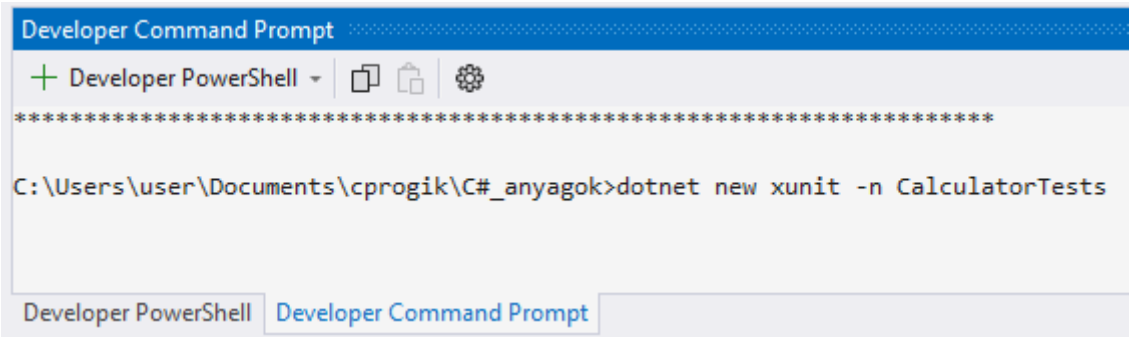


Vagy CLI-ből, bár itt figyelni kell, mert alapból a parancs **NEM** adja hozzá a projektet a Solution-hoz!

```
dotnet new xunit -n CalculatorTests
dotnet add CalculatorTests reference CalculatorProject
```

A Solution-hoz adás:

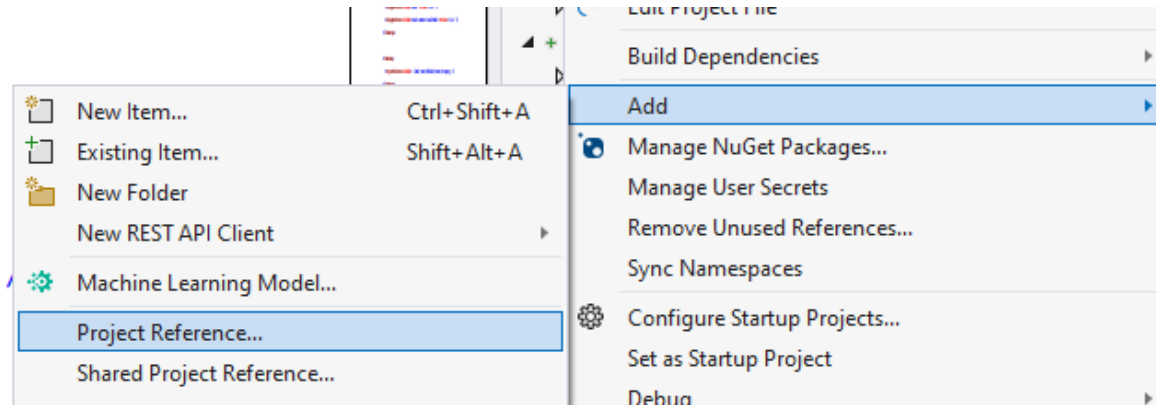
dotnet sln add CalculatorTests/CalculatorTests.csproj

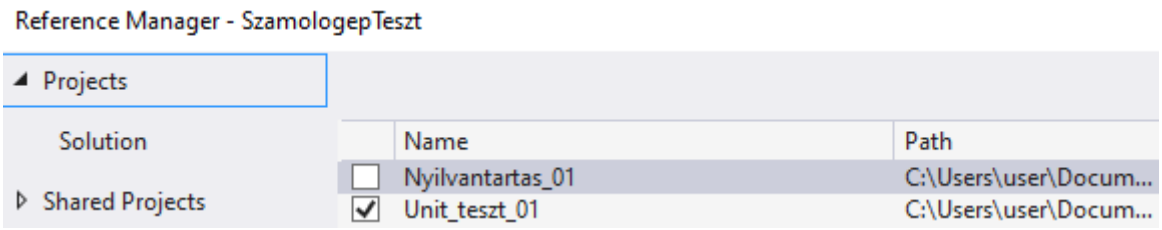


Jobb, ha a grafikus felületnél maradunk! Szerintem.

Kész a tesztelendő osztályunk és kész a teszt projektünk, már csak össze kell kapcsolni őket, hogy tudjanak egymásról! A teszt prokjekthez REFERENCIAKÉNT hozzáadjuk azt a projektet, amely tartalmazza a tesztelendő osztályt! (amelyikben üres a Main; **Unit_teszt_01**)

Jobb fül a teszt projekten, majd a Referencia menüpontnál kiválasztjuk a másik projektet!





És kész a referencia. Kezdhetjük a teszteket.

Nézzük az összeadás tesztelését:

```
using Unit_teszt_01;

namespace SzamologepTeszt
{
    public class UnitTest1
    {
        [Fact]
        public void Test1()
        {
            var sz = new Szamologep();

            var result = sz.Osszeadas(2, 3);

            Assert.Equal(5, result);
        }
```

Vegyük sorról sorra.

A `using Unit_teszt_01;` usingolást az osztályt tartalmazó projektre a referencia hozta létre, nem nekünk kellett beállítani!

A `[Fact]` egy attribútum az **xUnit** teszt frameworkben, ami azt jelenti, hogy az utána következő metódus egy teszt, amit a teszt futtatónak le kell futtatnia. (**NEM** a főprogramot indítjuk majd!!!)

Az első tesztelendő metódus neve: `public void Test1()`

Példányosítunk egy objektumot a **Szamologep** osztályból: `var sz = new Szamologep();`

Meghívjuk a tesztelendő osztály egy metódusát, itt az Osszeadas-t:
`var result = sz.Osszeadas(2, 3);`

Az **Assert** az ellenőrző mechanizmus, amely validálja, hogy a rendszer viselkedése megfelel az elvárt eredménynek, és eltérés esetén hibát jelez. Valójában ellenőrzi, hogy a kapott eredmény megfelel-e az elvárt eredménynek. A leggyakrabban használt **Assert** típusok:

Értékek összehasonlítása:

- *Assert.Equal(expected, actual);*
- *Assert.NotEqual(notExpected, actual);*

Igaz/hamis:

- *Assert.True(condition);*
- *Assert.False(condition);*

Null ellenőrzés:

- *Assert.Null(obj);*
- *Assert.NotNull(obj);*

Kivétel ellenőrzés:

- *Assert.Throws<DivideByZeroException>(() => sz.Osztas(10, 0));*

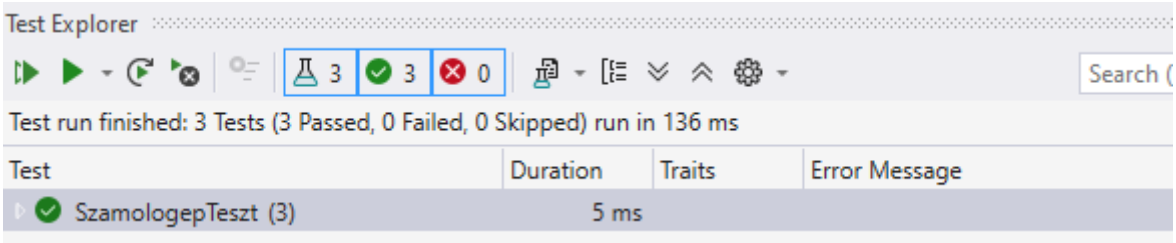
Nézzük meg a szorzás tesztjét:

```
[Fact]
public void Szorzas()
{
    var sz = new Szamologep();

    var result = sz.Szorzas(5, 3);

    Assert.Equal(15, result);
}
```

És végül az osztás tesztjét, figyelembe véve, hogy 0-val nem lehet osztani!

```
[Fact]
 public void Osztas_nullaval()
 {
     var sz = new Szamologep();

     Assert.Throws<DivideByZeroException>(() => sz.Osztas(10, 0));
 }
```

Kész, ennyit akartunk tesztelni. Most futtassuk a tesztet. Felső menü: ***Test → Run All***



A teszt eredménye megjelenik az osztályban is (a kivonásnál **NEM** teszteltünk, ott **NINCS** visszajelzés!):

```
public class Szamologep
{
    1 reference | 1/1 passing
    public int Osszeadas(int a, int b)
    {
        return a + b;
    }

    0 references
    public int Kivonas(int a, int b)
    {
        return a - b;
    }

    1 reference | 1/1 passing
    public int Szorzas(int a, int b)
    {
        return a * b;
    }
```

A tesztelést futtathatjuk paraméteresen is, ebben az esetben egyszerre több értéket is tudunk tesztelni. A szintaxisa:

```
[Theory]
[InlineData(5, 3, 8)]
[InlineData(-11, 11, 0)]
[InlineData(-8, -14, -22)]
public void OsszeadasTobbTesztesettel(int a, int b, int expected)
{
    var sz = new Szamologep();

    var result = sz.Osszeadas(a, b);

    Assert.Equal(expected, result);
}
```

Itt a `[Theory]` azt jelenti, hogy a teszt több különböző bemenettel fog lefutni.

- Az `[InlineData]` egyszerű értékekhez használható
- A `[MemberData]` komplexebb értékekhez való pl.:

```
public static IEnumerable<object[]> TestData =>
     new List<object[]>
     {
         new object[] { 2, 3, 5 },
         new object[] { 4, 6, 10 }
     };

 [Theory]
 [MemberData(nameof(TestData))]
 public void OsszeadasKomplex(int a, int b, int expected)
 {
     var sz = new Szamologep();
     var result = sz.Osszeadas(a, b);
     Assert.Equal(expected, result);
 }
```

- A `[ClassData]` akkor kell, ha külön osztályból jön az adat. (később vesszük)

A `IEnumerable<object[]>` egy olyan foreach-el bejárható gyűjtemény, amely elemei objektumok (az `object[]` bármit tud tárolni, így az xUnit nem fekszik meg, ha itt számot vagy szöveget, netán komplexebb típust kap paraméterül)

Az **xUnit**:

1. Végigmegy az `IEnumerable<object[]>` -en
2. Minden `object[]`-t paraméterként bead a metódusnak

Futásidők minta: