.NET CORE

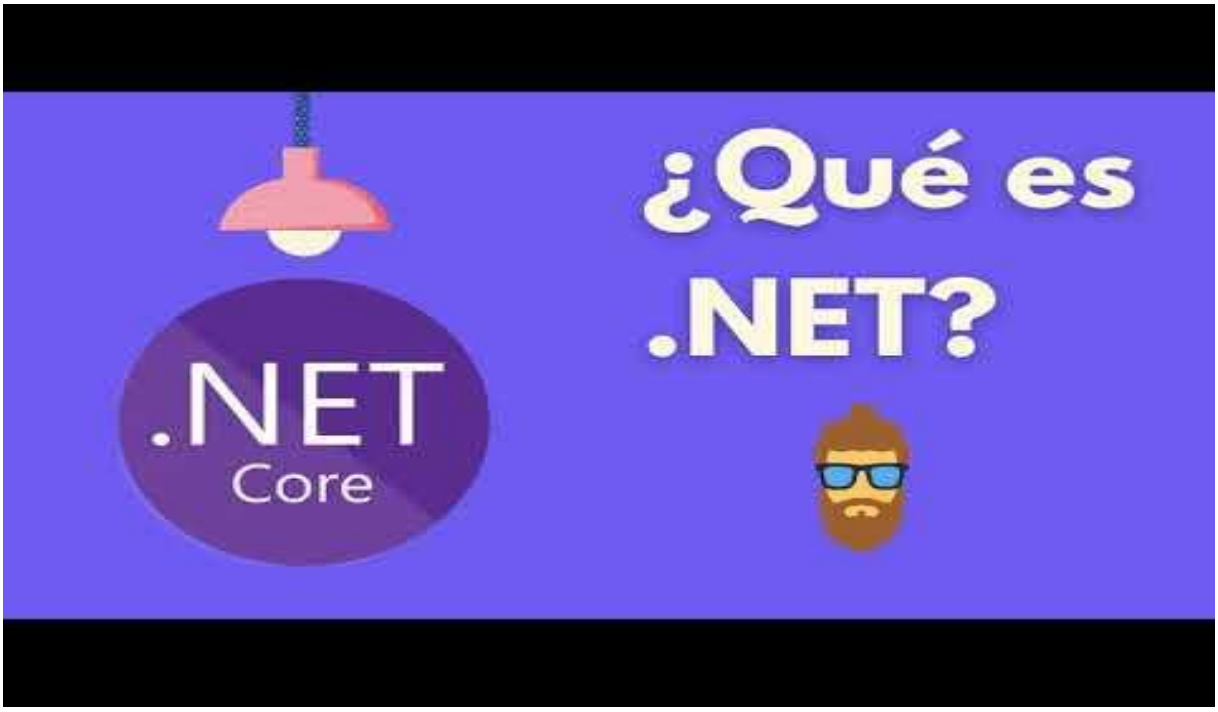
Semana 1

1. Qué es .NET Core?

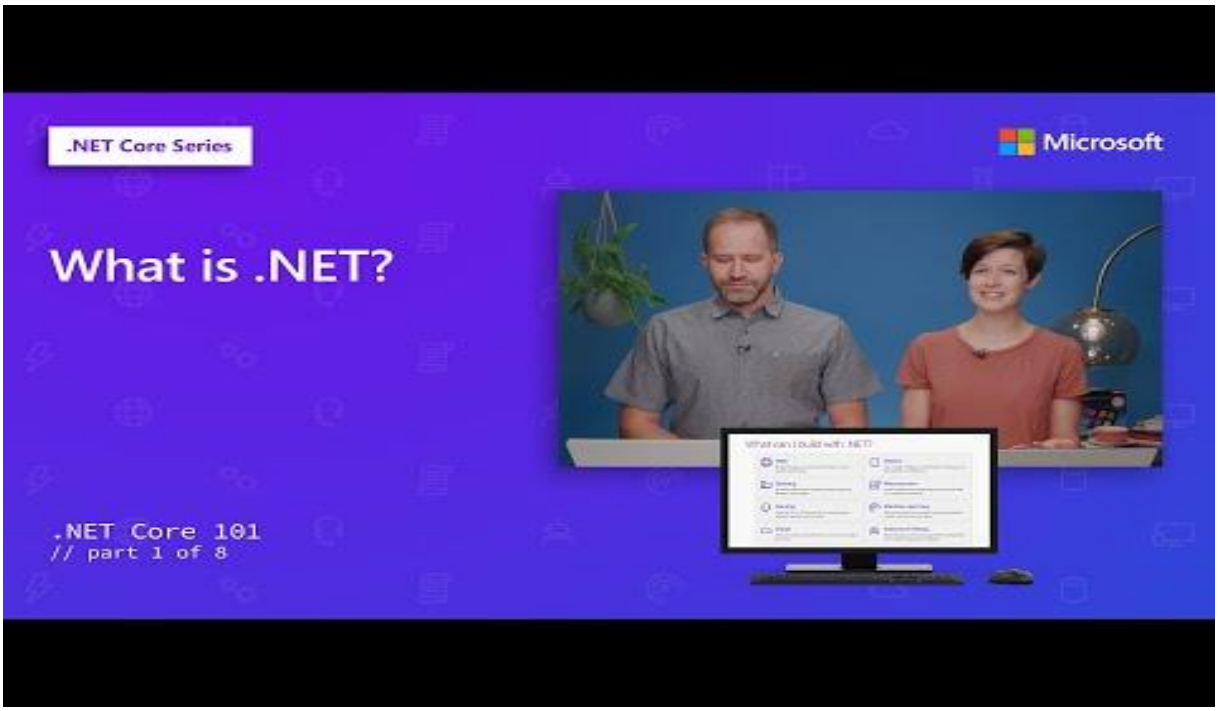
Objetivos:

- Diferenciar el significado de los conceptos: API, Librería, Lenguaje, Framework al describir .NET
- Identificar la diferencia entre .NET Core, Framework, Standard

[¿Qué es NET? ¿Y .NET Core? ¿Para que sirve? - Introducción completa al framework y sus opciones]



En Inglés (solo el primer video de la serie, los demás son opcionales)
[What is .NET? | .NET Core 101 [1 of 8]]



2. Hola Mundo! - C#

Objetivos:

- Instalar el SDK y un IDE
- Familiarizarse con algunos comandos del .NET CLI

Parte 1:
https://dotnet.microsoft.com/learn/dotnet/hello-world-tutorial/install

Parte 2:

Create a .NET console application using Visual Studio Code - .NET | Microsoft Docs

3. Adiós Mundo!

Objetivos:

- Conocer las diferentes versiones del lenguaje C# (Intentar para un par de instrucciones, cuándo aparecieron)
- https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history
- Familiarizarse con las estructuras básicas de control de flujo del lenguaje

- Familiarizarse con pruebas de concepto para entender desempeño y discutir esto por qué es importante en un ambiente Serverless vs Server-based computing.
- Nulos en .NET
- Concepto Clean Code: https://www.toptal.com/abap/clean-code-and-the-art-of-exception-handling y olvidarlo y reemplazarlo por este:
- https://martinfowler.com/articles/replaceThrowWithNotification.html

Crear un método para validar una entrada.

(Los que ya tengan experiencia en .NET Core usen las nuevas instrucciones de C# 9 para métodos simplificados -Top Level Statement-)

Parámetros de entrada: Un string que almacena una identificación.

Parámetros de salida: Una lista con los problemas de validación. Si la identificación es correcta, devolver una lista vacía.

Reglas:

Una identificación válida debe tener mínimo 5 caracteres, máximo 32.

Una identificación válida debe empezar con una letra mayúscula: A-Z

Implementar el método de 3 maneras diferentes:

Método 1: Expresiones regulares

Método 2: If normales.

Método 3: Lanzando excepciones cuando no se cumpla alguna condición.

Crear ciclos de 1’000.000 de interacciones con cada método.
Dentro del ciclo llamar 4 veces el método evaluado: Con una cadena sin problema, con una cadena que no cumple la longitud mínima, con una cadena que no cumple la longitud máxima, con una cadena que no cumple la regla de la mayúscula inicial.

Medir el tiempo de ejecución total

Medir el consumo de memoria y procesador promedio y total por cada método (System.Diagnostics)

Mostrar los tiempos de ejecución e información de memoria/procesador de cada método en consola.

Semana 2

1. Pruebas Unitarias

Una nota respecto a pruebas unitarias: En el pasado hemos cometido el error de preocuparnos primero por lo funcional/técnico y luego montar las pruebas unitarias, y luego estrellarnos con los conceptos y luego darnos cuenta de que igual tenemos bugs… y el esfuerzo de las pruebas unitarias para qué sirvió?

Objetivos:

- Comenzar a explorar el concepto de "cobertura" en pruebas unitarias.
- Preparar el ambiente de desarrollo para que ejecute pruebas unitarias con xUnit.

Lecturas (20 mins)

https://www.codegrip.tech/productivity/everything-you-need-to-know-about-code-coverage/

https://www.stickyminds.com/article/100-percent-unit-test-coverage-not-enough

Instalar el Test Explorer y xUnit:

https://www.codemag.com/Article/2009101/Interactive-Unit-Testing-with-.NET-Core-and-VS-Code

Para uno de los métodos implementados en la semana 1 crear pruebas unitarias.
Utilizar los decoradores [Fact] y [Theory] y explicar la diferencia.

Cómo ver la cobertura en VS Code?
Se puede montar un complemento:
https://marketplace.visualstudio.com/items?itemName=markis.code-coverage

No encontré uno que marque la cobertura…. Pero la forma como lo hacemos realmente en los proyectos es montando un servidor de SonarQube.

La recomendación: Monten un servidor de SonarQube Local.
Esto es opcional (por largo), pero muy recomendado.
https://marketplace.visualstudio.com/items?itemName=markis.code-coverage