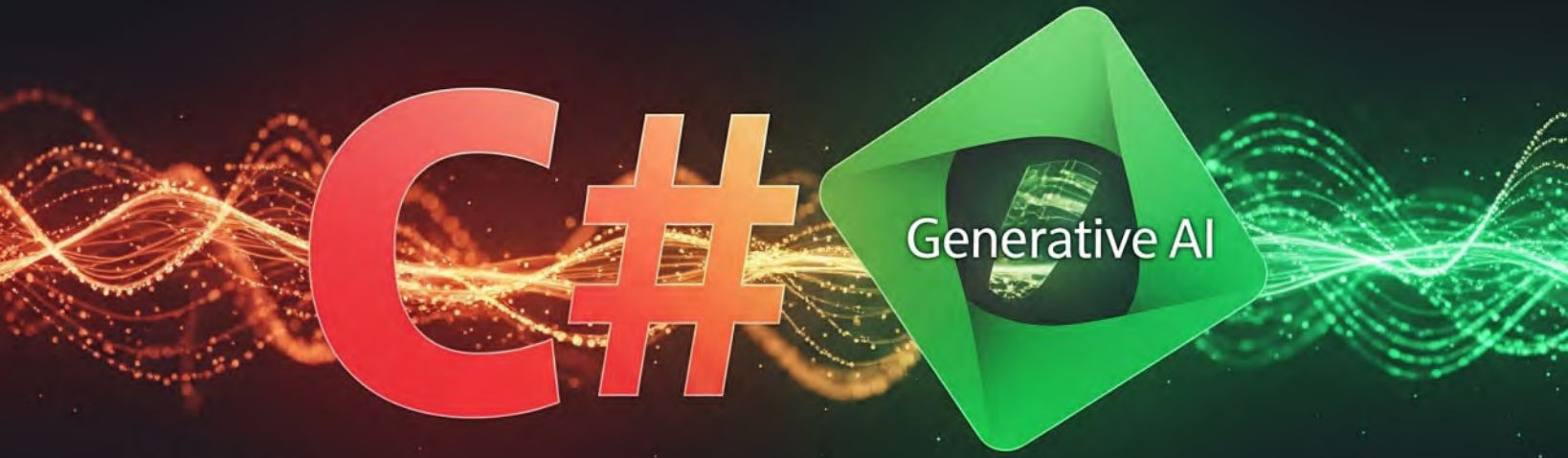
C# Programming: Beginner

(Integrative-Generative AI Edition)



Contents

.NET

Student Price Book Center

ai

คำนำ

ในยุคดิจิทัลที่เทคโนโลยีสารสนเทศเข้ามามีบทบาทสำคัญในทุกมิติของชีวิต การพัฒนาแอปพลิเคชันและระบบซอฟต์แวร์กลายเป็นทักษะที่สำคัญอย่างยิ่งสำหรับนักศึกษา นักพัฒนา และผู้สนใจทั่วไป ภาษา C# (อ่านว่า “ซีชาร์ป”) คือหนึ่งในภาษาการเขียนโปรแกรมที่ได้รับความนิยมสูงอย่างต่อเนื่อง ด้วยคุณลักษณะที่ใช้งานง่าย ทรงพลัง และรองรับการพัฒนาแอปพลิเคชันหลากหลายแพลตฟอร์มผ่าน .NET ecosystem ทั้งบน Windows, Linux, macOS, เว็บ, โมบาย และคลาวด์

หนังสือเล่มนี้จัดทำขึ้นภายใต้เป้าหมายสำคัญคือ **การปูพื้นฐานความรู้ด้านการเขียนโปรแกรมภาษา C# อย่างเป็นระบบ จากระดับเริ่มต้นไปจนถึงสามารถประยุกต์ใช้แนวคิดต่าง ๆ ได้อย่างมั่นใจ** โดยเนื้อหาถูกออกแบบตามลำดับขั้นเพื่อให้ผู้อ่านที่ไม่มีพื้นฐานมาก่อนสามารถเข้าใจและฝึกฝนได้ด้วยตนเอง โดยเริ่มจากความเข้าใจในภาษา C# และแพลตฟอร์ม .NET ก่อนเข้าสู่แนวคิดพื้นฐานของภาษา การใช้งานเมธอด อาร์เรย์และคอลเลกชัน จนถึงหลักการเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming)

บทที่ 1 ว่าด้วยการแนะนำภาษา C# และแพลตฟอร์ม .NET โดยอธิบายตั้งแต่ประวัติความเป็นมา พัฒนาการของเทคโนโลยี .NET ตั้งแต่ .NET Framework สู่ .NET Core และ .NET 5+ ผู้อ่านจะได้เข้าใจความแตกต่างของแต่ละเวอร์ชัน ข้อดี ข้อจำกัด และแนวทางการเลือกใช้งานให้เหมาะกับบริบทการพัฒนา พร้อมทั้งฝึกติดตั้งเครื่องมือพัฒนาอย่าง .NET SDK, Visual Studio และ VS Code รวมถึงได้ลงมือสร้างโปรเจกต์ Console Application แรกด้วยตนเองอย่างเป็นขั้นตอน

บทที่ 2 มุ่งเน้นที่โครงสร้างพื้นฐานของภาษา C# ตั้งแต่การประกาศตัวแปร ชนิดข้อมูลพื้นฐาน เช่น int, float, string, และ bool การใช้คำสั่งเงื่อนไข (if, else, switch) และการควบคุมการทำงานแบบวนลูป (for, while, do-while, foreach) ผู้อ่านจะได้เรียนรู้การรับข้อมูลจากผู้ใช้และแสดงผลทางหน้าจอด้วย Console.ReadLine() และ Console.WriteLine() ซึ่งเป็นจุดเริ่มต้นสำคัญของการติดต่อกับผู้ใช้งาน

บทที่ 3 กล่าวถึงการใช้งานเมธอด (Methods) ซึ่งเป็นกลไกสำคัญในการแยกส่วนการทำงานของโปรแกรมให้เป็นโมดูลเล็ก ๆ ที่สามารถนำกลับมาใช้ซ้ำได้อย่างมีประสิทธิภาพ ผู้อ่านจะได้เรียนรู้วิธีสร้างเมธอด การส่งพารามิเตอร์เข้าเมธอด การรับค่าคืนจากเมธอด และการใช้คีย์เวิร์ดพิเศษ เช่น ref, out, และ params ซึ่งช่วยให้การรับส่งค่าข้อมูลระหว่างเมธอดมีความยืดหยุ่นยิ่งขึ้น

บทที่ 4 พาผู้อ่านเข้าสู่โลกของการจัดการข้อมูลจำนวนมากด้วย **อาร์เรย์ (Array)** และ **คอลเลกชัน (Collection)** ซึ่งเป็นโครงสร้างข้อมูลที่ใช้เก็บชุดของข้อมูลชนิดเดียวกันและชนิดต่างกันอย่างมีระบบ โดยเนื้อหาครอบคลุมตั้งแต่อาร์เรย์ 1 มิติ และ 2 มิติ ไปจนถึงคอลเลกชันพื้นฐานอย่าง List<T> และ Dictionary<TKey, TValue> พร้อมตัวอย่างการใช้งานในสถานการณ์จริง ซึ่งจะช่วยให้ผู้อ่านสามารถประมวลผลข้อมูลได้อย่างมีประสิทธิภาพ

บทที่ 5 เป็นการแนะนำแนวคิดการเขียนโปรแกรมเชิงวัตถุ (OOP) ซึ่งเป็นรูปแบบการเขียนโปรแกรมที่เน้นการจำลองสิ่งของหรือสถานการณ์ในโลกจริงให้กลายเป็นวัตถุ (Object) ในภาษา C# ผู้อ่านจะได้รู้จักกับการสร้าง Class และ Object, การใช้งาน Fields และ Properties, การออกแบบ

Constructor สำหรับกำหนดค่าเริ่มต้นของวัตถุ และการใช้ this keyword เพื่ออ้างอิงสมาชิกภายในวัตถุเอง ซึ่งเป็นรากฐานของการพัฒนาโปรแกรมที่ซับซ้อนในลักษณะเชิงโครงสร้าง

ตลอดทั้งเล่ม ผู้อ่านจะได้พบกับ **ตัวอย่างโปรแกรมบูรณาการ** ในแต่ละบท ซึ่งไม่เพียงแต่ช่วยเสริมความเข้าใจเชิงแนวคิด แต่ยังเปิดโอกาสให้ฝึกคิด ฝึกแก้ปัญหา และฝึกลงมือพัฒนาโปรแกรมอย่างเป็นระบบ นอกจากนี้ยังมีคำอธิบายเชิงลึกในหัวข้อที่สำคัญเพื่อขยายความเข้าใจและเชื่อมโยงกับแนวคิดระดับสูงต่อไปในอนาคต

ผู้เขียนเชื่อมั่นว่า หนังสือ *C# Programming: Beginner* เล่มนี้จะเป็นคู่มือสำคัญสำหรับผู้ที่เริ่มต้นเข้าสู่โลกของการพัฒนาโปรแกรมด้วยภาษา C# ไม่ว่าจะเป็นนักเรียน นักศึกษา ครู อาจารย์ หรือผู้ที่ต้องการเสริมทักษะด้านโปรแกรมมิ่ง เพื่อเตรียมพร้อมสู่การพัฒนาแอปพลิเคชันระดับมืออาชีพในอนาคต

ขอขอบคุณผู้อ่านทุกท่านที่เลือกหนังสือเล่มนี้เป็นแนวทางในการเรียนรู้ ขอให้ทุกท่านได้รับประโยชน์สูงสุดจากการศึกษาเนื้อหาภายในเล่ม และสามารถต่อยอดความรู้เพื่อสร้างสรรค์นวัตกรรมที่มีคุณค่าในเส้นทางสายวิชาชีพด้านเทคโนโลยีสารสนเทศต่อไป

ด้วยรักและปรารถนาดี

ศูนย์หนังสือราคานักเรียน

สารบัญ

บทที่ 1

แนะนำ C# และ .NET

(Introduction to C# and .NET)

เนื้อหา

- แนะนำ C# และ .NET
- แนะนำ C# และ .NET (ฉบับละเอียดเชิงลึก)
- ประวัติของ C# และ .NET (แบบละเอียดและลึก)
- .NET Framework vs .NET Core vs .NET 5+ (แบบละเอียด เชิงลึก)
- การติดตั้ง .NET SDK และ Visual Studio / VS Code (เชิงลึก)
- สร้างโปรเจกต์ Console Application แรก (ด้วย .NET SDK + Visual Studio / VS Code)

บทนำบทที่ 1: แนะนำ C# และ .NET

ในโลกของการพัฒนาแอปพลิเคชันที่มีความซับซ้อนและเปลี่ยนแปลงอย่างรวดเร็ว ภาษาโปรแกรมที่มีประสิทธิภาพ มีความยืดหยุ่น และสามารถนำไปใช้งานได้หลากหลายแพลตฟอร์ม กลายเป็นสิ่งจำเป็น สำหรับนักพัฒนา C# (อ่านว่า “ซีชาร์ป”) คือหนึ่งในภาษาที่ตอบโจทย์เหล่านี้ได้อย่างสมบูรณ์แบบ ด้วยการออกแบบโดย Microsoft ให้เป็นภาษาระดับสูงที่ใช้แนวคิดเชิงวัตถุ (Object-Oriented Programming) อย่างเต็มรูปแบบ และมีความกลมกลืนอย่างใกล้ชิดกับแพลตฟอร์ม .NET ซึ่งเป็นโครงสร้างพื้นฐานสำหรับการพัฒนาแอปพลิเคชันในหลายลักษณะ ทั้งแบบเดสก์ท็อป เว็บ และโมบาย

C# เริ่มต้นพัฒนาในช่วงต้นทศวรรษ 2000 โดยมี Anders Hejlsberg ผู้ออกแบบภาษาชื่อดังเป็นผู้นำโครงการ จุดมุ่งหมายหลักของ C# คือการสร้างภาษาที่มีพลังแบบ C++ แต่มีความปลอดภัยและใช้งานง่ายเหมือน Java โดยการผนวกเข้ากับ .NET Framework ซึ่งเป็นแพลตฟอร์มที่รวมเอาเครื่องมือ ไลบรารี และคอมไพเลอร์ไว้ครบถ้วน ทำให้นักพัฒนาสามารถสร้างแอปพลิเคชันได้อย่างมีประสิทธิภาพสูงสุด

ในช่วงหลายปีที่ผ่านมา .NET ได้พัฒนาตัวเองจาก .NET Framework ซึ่งรองรับเฉพาะ Windows ไปสู่ .NET Core ที่มีความสามารถแบบข้ามแพลตฟอร์ม และในที่สุดคือ .NET 5+ ซึ่งรวมศักยภาพของทั้งสองโลกเข้าด้วยกัน ภายใต้แนวคิด "One .NET" ที่มุ่งสู่การสร้างแพลตฟอร์มแบบรวมศูนย์ ที่สามารถนำไปใช้ได้ทั้งบน Windows, macOS, Linux, Cloud, IoT และแม้แต่บนอุปกรณ์มือถือ

ความเข้าใจความแตกต่างระหว่าง .NET Framework, .NET Core และ .NET 5+ เป็นสิ่งสำคัญสำหรับผู้เริ่มต้น เนื่องจากแต่ละเวอร์ชันมีคุณสมบัติ ข้อจำกัด และการนำไปใช้งานที่แตกต่างกัน .NET Framework ยังใช้กับแอปพลิเคชันเดิมบน Windows ในขณะที่ .NET Core และ .NET 5+ สนับสนุนการ

พัฒนาแอปที่มีประสิทธิภาพและยืดหยุ่นมากขึ้น โดยเน้นการทำงานแบบโอเพนซอร์สและครอบคลุมหลายระบบปฏิบัติการ

สำหรับนักพัฒนาที่เริ่มต้นเรียนรู้ C# ในยุคปัจจุบัน เครื่องมือที่สำคัญคือ .NET SDK ซึ่งเป็นชุดเครื่องมือที่ใช้สำหรับคอมไพล์และรันโค้ด C# ในทุกระบบปฏิบัติการ รวมถึง Visual Studio และ Visual Studio Code ซึ่งเป็นสภาพแวดล้อมสำหรับการพัฒนา (IDE/Editor) ที่ได้รับความนิยมสูง โดย Visual Studio เหมาะสำหรับผู้ใช้ Windows ที่ต้องการฟีเจอร์ครบถ้วน ส่วน VS Code เป็นเครื่องมือเบา รองรับหลายภาษา และสามารถทำงานได้ทั้งบน Windows, macOS และ Linux

ขั้นตอนการติดตั้งเครื่องมือเหล่านี้ไม่ซับซ้อน และสามารถใช้งานได้ฟรีจากเว็บไซต์ทางการของ Microsoft ผู้เรียนจะได้เรียนรู้การติดตั้ง .NET SDK และ Visual Studio/VS Code อย่างเป็นระบบ พร้อมเข้าใจการกำหนดค่าพื้นฐานเพื่อเริ่มต้นพัฒนาโปรเจกต์ C# อย่างมั่นใจ

เมื่อเครื่องมือพร้อมใช้งานแล้ว สิ่งสำคัญถัดมาคือการสร้างโปรเจกต์แรกของตนเอง ผ่านรูปแบบ Console Application ซึ่งเป็นโปรแกรมพื้นฐานที่รับอินพุตและแสดงผลทางหน้าจอคอนโซล จุดประสงค์ของโปรเจกต์นี้ไม่ใช่เพียงแค่การแสดงผลข้อความเท่านั้น แต่ยังเป็นการเรียนรู้โครงสร้างของโปรเจกต์ C# การทำงานของไฟล์ Program.cs และคำสั่งเบื้องต้น เช่น Console.WriteLine() อย่างเข้าใจ

การทดลองเขียนโปรแกรมแรกไม่เพียงช่วยเสริมสร้างความมั่นใจให้กับผู้เรียนเท่านั้น แต่ยังเป็นการวางรากฐานเชิงโครงสร้างของโปรแกรมแบบ C# ที่จะสามารถต่อยอดไปสู่การพัฒนาแอปในระดับที่ซับซ้อนยิ่งขึ้น ไม่ว่าจะเป็นเว็บแอป, แอปพลิเคชันแบบ GUI, เกม หรือแม้แต่ระบบที่ใช้เทคโนโลยี AI และ Cloud

บทที่ 1 นี้จึงมีเป้าหมายสำคัญในการปูพื้นฐานด้านภาษาและเครื่องมือ เพื่อให้ผู้อ่านมีความเข้าใจทั้งเชิงทฤษฎีและการปฏิบัติ ไม่เพียงแต่รู้จักภาษา C# และแพลตฟอร์ม .NET เท่านั้น แต่ยังสามารถลงมือสร้างและรันแอปพลิเคชันด้วยตนเองได้ในเวลาอันสั้น

การเริ่มต้นเรียนรู้ภาษาโปรแกรมอาจดูท้าทาย แต่ด้วยแนวทางที่ชัดเจน เครื่องมือที่มีประสิทธิภาพ และตัวอย่างที่เข้าใจง่าย บทนี้จะช่วยให้ผู้อ่านก้าวแรกได้อย่างมั่นใจ พร้อมเปิดประตูสู่โลกแห่งการพัฒนาแอปพลิเคชันด้วย C# อย่างเต็มรูปแบบในบทถัดไป

แนะนำ C# และ .NET

- ประวัติของ C# และ .NET
- .NET Framework vs .NET Core vs .NET 5+
- การติดตั้ง .NET SDK และ Visual Studio / VS Code
- สร้างโปรเจกต์ Console Application แรก

☐ **1. ประวัติของ C# และ .NET**

☐ **C# คืออะไร?**

- **C# (อ่านว่า "ซีชาร์ป")** เป็นภาษาโปรแกรมเชิงวัตถุ (Object-Oriented Programming: OOP)

- พัฒนาโดย **Microsoft** ภายใต้การนำของ **Anders Hejlsberg**
- เปิดตัวครั้งแรกในปี **2000** พร้อมกับ **.NET Framework**
- ได้รับแรงบันดาลใจจากภาษา C, C++, Java
- ถูกออกแบบให้ปลอดภัย, มีประสิทธิภาพ และรองรับการพัฒนาแอปพลิเคชันทุกประเภท เช่น:
    - Desktop
    - Web
    - Mobile
    - Game
    - Cloud
    - IoT

จุดเด่นของ C#

- มี **Garbage Collector** คอยจัดการหน่วยความจำอัตโนมัติ
- สนับสนุนทั้ง **OOP** และ **Functional Programming**
- รองรับ **Asynchronous Programming (async/await)**
- ทำงานร่วมกับ **.NET Runtime** ได้อย่างสมบูรณ์แบบ

---

2. .NET Framework vs .NET Core vs .NET 5+

.NET คืออะไร?

- **.NET** คือ **แพลตฟอร์มในการพัฒนาแอปพลิเคชันของ Microsoft**
- ประกอบด้วย:
    - Runtime (เช่น CLR – Common Language Runtime)
    - Class Libraries
    - Language Support (C#, VB.NET, F#)
    - Toolchain (Compiler, Debugger, NuGet)

⚖ **ความแตกต่างของแต่ละเวอร์ชัน:**

รายการ	.NET Framework	.NET Core	.NET 5+ (รวม .NET 6, 7, 8...)
เปิดตัวปี	2002	2016	2020 เป็นต้นไป
รองรับระบบ	Windows เท่านั้น	Cross-platform (Windows, Linux, macOS)	Cross-platform
ประสิทธิภาพ	ต่ำกว่า .NET Core	เร็วขึ้น, น้ำหนักเบา	รวมข้อดีของทั้ง 2
ใช้งาน	ยังใช้กับ legacy app	แนะนำสำหรับแอปใหม่	ทางเลือกหลักในปัจจุบัน

รายการ	.NET Framework	.NET Core	.NET 5+ (รวม .NET 6, 7, 8...)
สถานะ	ไม่พัฒนาเพิ่ม	ยังใช้งานได้	พัฒนาอย่างต่อเนื่อง

□ สรุป

Microsoft แนะนำให้ใช้ **.NET 6 ขึ้นไป** สำหรับการพัฒนาแอปใหม่ทั้งหมด

□ 3. การติดตั้ง .NET SDK และ Visual Studio / VS Code

□ ติดตั้ง .NET SDK

1. ไปที่เว็บไซต์: https://dotnet.microsoft.com/en-us/download
2. ดาวน์โหลด .NET SDK ล่าสุด (แนะนำเป็น LTS เช่น .NET 8 LTS)
3. เลือกตามระบบปฏิบัติการ: Windows, macOS, Linux
4. เมื่อติดตั้งเสร็จ ให้เปิด Terminal / CMD แล้วพิมพ์:

```
dotnet --version
```

จะแสดงเวอร์ชัน เช่น 8.0.301 แปลว่าพร้อมใช้งานแล้ว □

□ ติดตั้ง Visual Studio (หรือ Visual Studio Code)

ทางเลือก 1: Visual Studio (เหมาะกับมือใหม่)

- ดาวน์โหลดได้จาก https://visualstudio.microsoft.com/
- แนะนำใช้ **Visual Studio Community Edition** (ฟรี)
- ตอนติดตั้งให้เลือก workload:
  - □ **.NET desktop development**

ทางเลือก 2: Visual Studio Code (VS Code)

- ดาวน์โหลดได้จาก https://code.visualstudio.com/
- ติดตั้ง Extension:
  - C# โดย Microsoft (ผ่าน Marketplace)
  - C# Dev Kit (เสริมประสบการณ์การพัฒนา)
- ใช้ร่วมกับ Terminal และ dotnet CLI ได้อย่างยืดหยุ่น

□ 4. สร้างโปรเจกต์ Console Application แรก

□ วิธีที่ 1: ใช้ Visual Studio

1. เปิด Visual Studio > เลือก "Create a new project"
2. เลือก: **Console App (.NET Core / .NET 6/7/8)**
3. ตั้งชื่อโปรเจกต์ เช่น HelloCSharp

4. กด Create

□ Visual Studio จะสร้างไฟล์ให้คุณอัตโนมัติ เช่น:

- Program.cs
- .csproj

□ โค้ดตัวอย่างใน Program.cs:

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine("Hello, C#!");
    }
}
```

5. กดปุ่ม ▶□ “Start” เพื่อรันโปรแกรม
   ผลลัพธ์: Hello, C#!

□ วิธีที่ 2: ใช้ Command Line (dotnet CLI)

```
dotnet new console -n HelloCSharp
cd HelloCSharp
dotnet run
```

□ Output:

```
Hello, C#!
```

□ สรุปบทที่ 1

หัวข้อ	สาระสำคัญ
C# คือภาษา OOP จาก Microsoft ที่ใช้ใน .NET Platform	
.NET 5+ คือเวอร์ชันปัจจุบันที่รวม .NET Core และ Framework	
สามารถติดตั้ง .NET SDK และ IDE ได้ฟรี	
การสร้าง Console App เป็นจุดเริ่มต้นของการฝึกเขียนโปรแกรม	

แนะนำ C# และ .NET (ฉบับละเอียดเชิงลึก)

☐ 1. ประวัติของ C# และ .NET

☐ จุดเริ่มต้นของ C# และ .NET

- ปี **2000** Microsoft เปิดตัว **.NET Framework** รุ่นแรก พร้อมกับภาษา **C#** โดยมีจุดประสงค์เพื่อแข่งขันกับ Java และเป็นทางเลือกใหม่ในการพัฒนา Windows Application
- C# พัฒนาโดย **Anders Hejlsberg** (ผู้สร้างภาษา Turbo Pascal และ Delphi มาก่อน)
- C# เป็นภาษา **Multi-paradigm** คือ:
  - รองรับ **Object-Oriented Programming (OOP)**
  - รองรับ **Component-Oriented Programming**
  - รองรับ **Functional Programming** (ตั้งแต่ C# 3.0 เป็นต้นไป)

☐ จุดประสงค์หลักของ .NET:

1. สร้างสภาพแวดล้อมที่ปลอดภัยและมีประสิทธิภาพ
2. ลดปัญหา **DLL Hell** (เวอร์ชันของไลบรารีที่ขัดแย้งกัน)
3. พัฒนาแอปแบบ **managed code** ด้วย CLR (Common Language Runtime)
4. เขียนหลายภาษา (C#, VB.NET, F#) แต่ทำงานร่วมกันได้ใน CLR เดียวกัน

☐ 2. .NET Framework vs .NET Core vs .NET 5+

☐ สถาปัตยกรรมพื้นฐานของ .NET

```
+-----------------------+
| Application Layer     |
| (Console, Web, Desktop)|
+-----------------------+
| Base Class Libraries  |
+-----------------------+
| Common Language Runtime|
| - JIT Compiler        |
| - Garbage Collector   |
| - Security/Interop    |
+-----------------------+
| Operating System      |
+-----------------------+
```

☐ .NET Framework (2002–2022)

- ใช้ได้เฉพาะ **Windows**
- ทำงานกับ Windows Forms, ASP.NET, WPF
- ยังถูกใช้ในองค์กรที่มีระบบ legacy

- ไม่รองรับ cross-platform

.NET Core (2016–2020)

- **เปิดโอเพ่นซอร์ส (Open Source)**
- ทำงานได้หลายระบบปฏิบัติการ (Windows, Linux, macOS)
- มี performance ที่ดีกว่า .NET Framework
- รองรับ **Docker**, Microservices

.NET 5+ (2020 – ปัจจุบัน)

- เรียกว่า **Unified Platform** (รวม .NET Core, Xamarin, Mono เข้าด้วยกัน)
- รองรับทุกแพลตฟอร์มจาก codebase เดียว:
  - Desktop (Windows Forms, WPF)
  - Web (ASP.NET Core, Blazor)
  - Mobile (Xamarin, .NET MAUI)
  - Cloud
  - IoT
- ใช้แนวคิด **“One .NET”**
- อัปเดตปีเวอร์ชัน เช่น .NET 6 (LTS), .NET 7, .NET 8 (LTS), .NET 9 ฯลฯ

ข้อเปรียบเทียบเชิงลึก

คุณสมบัติ	.NET Framework	.NET Core / .NET 5+
ระบบปฏิบัติการ	Windows	Cross-platform
Open-source		
Performance	ปานกลาง	สูง
Modular (NuGet)	จำกัด	เต็มรูปแบบ (Slim Runtime)
Mobile Support		(ผ่าน MAUI, Xamarin)
Blazor / WebAssembly		
Web API Modern Tools	ASP.NET (เดิม)	ASP.NET Core

---

3. การติดตั้ง .NET SDK และ Visual Studio / VS Code

.NET SDK คืออะไร?

- SDK = Software Development Kit
- รวม:
  - Compiler (csc.exe สำหรับ C#)
  - dotnet CLI

- Runtime
- Template engine
- Debugger integration

☐ SDK = ใช้ "พัฒนา" ได้

☐ Runtime = ใช้ "รัน" ได้อย่างเดียว

☐ **วิธีติดตั้ง .NET SDK (Windows/Mac/Linux)**

1. เข้าเว็บไซต์ https://dotnet.microsoft.com/download
2. ดาวน์โหลดเวอร์ชัน **Latest LTS** เช่น .NET 8.0.301 (LTS)
3. เลือกระบบปฏิบัติการ: Windows / Mac / Linux
4. รันตัวติดตั้งและทำตามขั้นตอน

☐ **ตรวจสอบหลังติดตั้ง:**

```
dotnet --info
dotnet --version
```

☐ ถ้าแสดงเวอร์ชัน = พร้อมใช้งานแล้ว เช่น:

```
.NET SDK (reflecting any global.json):
 Version:   8.0.301
```

☐ **Visual Studio vs Visual Studio Code**

☐ **Visual Studio (Full IDE)**

- เหมาะสำหรับ:
  - มือใหม่
  - การพัฒนาแบบ GUI
  - Debug ขั้นสูง
- ฟีเจอร์เด่น:
  - IntelliSense
  - Live debugging
  - Designer tools
- ข้อเสีย: กิน RAM เยอะ

☐ **Visual Studio Code (Editor + CLI)**

- น้ำหนักเบา
- ติดตั้ง Extension สำหรับ C#:
  - C# for Visual Studio Code (powered by OmniSharp)
  - C# Dev Kit (optional)
- ทำงานร่วมกับ dotnet CLI ได้ดีมาก

☐ 4. สร้างโปรเจกต์ Console Application แรก

☐ โครงสร้างพื้นฐานของ Console App

เมื่อสร้างโปรเจกต์ dotnet new console จะได้โครงสร้างดังนี้:

```
HelloCSharp/
│
├── HelloCSharp.csproj    <-- Project file (XML)
└── Program.cs            <-- Entry point
```

☐ เขียนโปรแกรม “Hello, World!”

☐ ไฟล์: Program.cs

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine("Hello, C#!");
    }
}
```

☐ อธิบายโค้ด:

- using System; ⟶ import namespace System ซึ่งมีคลาส Console
- class Program ⟶ สร้างคลาสหลัก
- static void Main() ⟶ Entry point ของแอป (เริ่มรันที่นี่)
- Console.WriteLine(...) ⟶ แสดงข้อความไปยัง Console

☐ สร้างโปรเจกต์ด้วย dotnet CLI

ขั้นตอน:

```
dotnet new console -n HelloCSharp
cd HelloCSharp
dotnet run
```

☐ Output:

```
Hello, C#!
```

☐ ไฟล์ .csproj (Project File)

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net8.0</TargetFramework>
  </PropertyGroup>
</Project>
```

☐ คำอธิบาย:

- OutputType = Exe หมายถึงเป็น executable app
- TargetFramework = ระบุว่าใช้ .NET version อะไร (net8.0 = .NET 8)

☐ **สรุปสิ่งที่คุณควรรู้จากบทที่ 1**

หัวข้อ	สิ่งที่ได้เรียนรู้
C# เป็นภาษา OOP ที่พัฒนาโดย Microsoft	
.NET Framework vs Core vs 5+ มีความแตกต่างชัดเจน	
SDK คือเครื่องมือหลักสำหรับพัฒนา C#	
Visual Studio เหมาะกับมือใหม่, VS Code เหมาะกับ CLI	
Console App คือจุดเริ่มต้นของการเรียนรู้ทุกระดับ	

☐ **แบบฝึกหัดแนะนำ**

1. ติดตั้ง .NET SDK และ Visual Studio หรือ VS Code
2. สร้าง Console App ด้วยคำสั่ง dotnet new console -n MyApp
3. เปลี่ยนข้อความใน Program.cs เป็น:

```
Console.WriteLine("Welcome to C# Learning!");
```

4. รันโปรแกรมด้วย dotnet run

ประวัติของ C# และ .NET (แบบละเอียดและลึก)

☐ **จุดเริ่มต้นของ .NET**

☐ **ปัญหาที่ Microsoft ต้องการแก้:**

ในช่วงปลายทศวรรษ 1990:

- Microsoft ประสบปัญหากับการพัฒนาแอป Windows ด้วย C++/COM ซึ่งซับซ้อนและจัดการหน่วยความจำยาก
- เกิด **“DLL Hell”** – คือการจัดการเวอร์ชันของไลบรารี .dll ที่ทำให้โปรแกรม crash ง่าย

- Java ซึ่งเป็นภาษาจาก Sun Microsystems กำลังมาแรง เพราะ:
  - ทำงานแบบ Cross-platform
  - มี Virtual Machine
  - มี Garbage Collector
- Microsoft ต้องการสร้าง “แพลตฟอร์มใหม่” ที่ดีกว่า Java, ปลอดภัยกว่า C++ และง่ายกว่าสำหรับนักพัฒนา

ทางออก: สร้างแพลตฟอร์มใหม่ ⟶ .NET Framework

- เปิดตัวปี **2002**
- ใช้หลักการ **Managed Code** คือ โค้ดที่รันบน Runtime ที่จัดการทุกอย่างให้อัตโนมัติ (ผ่าน **CLR – Common Language Runtime**)
- รองรับหลายภาษา เช่น C#, VB.NET, JScript.NET
- ใช้ **Base Class Library (BCL)** ซึ่งรวมคลาสมาตรฐานสำหรับทุกภาษา

การกำเนิดของภาษา C#

ผู้ออกแบบ: Anders Hejlsberg

- เขาเป็นคนเดียวกับผู้สร้าง **Turbo Pascal** และ **Delphi**
- ถูก Microsoft จ้างมาสร้างภาษายุคใหม่ที่ “ทันสมัย, ปลอดภัย, เชิงวัตถุ และมีประสิทธิภาพ”

ชื่อภาษา

- เดิมเรียกว่า “Cool” (C-like Object Oriented Language)
- ต่อมาถูกเปลี่ยนชื่อเป็น **C#** (อ่านว่า “ซีชาร์ป”)
  - ได้แรงบันดาลใจจากโน้ตดนตรี (Sharp = ยกระดับ)
  - ยังคล้ายกับชื่อ “C++” แต่แสดงถึงความ “ทันสมัยยิ่งกว่า”

C# และ .NET รุ่นแรก

.NET Framework 1.0 + C# 1.0 (2002)

- เป้าหมายหลัก:
  - สนับสนุน Windows Forms
  - พัฒนาเว็บด้วย ASP.NET Web Forms
  - มี ADO.NET สำหรับเชื่อมต่อฐานข้อมูล
  - สร้างแอปเดสก์ท็อปแบบ OOP ได้ง่าย
- จุดเด่น:
  - Garbage Collector
  - Type Safety
  - Exception Handling

- Reflection

☐ **วิวัฒนาการของ C# และ .NET**

☐ **C# Evolution (ฟีเจอร์ที่พัฒนาในแต่ละเวอร์ชัน)**

เวอร์ชัน	ปี	ฟีเจอร์เด่น
C# 1.0	2002	Class, OOP, Exception
C# 2.0	2005	Generics, Nullable Types
C# 3.0	2007	LINQ, Lambda, Anonymous Types
C# 4.0	2010	Dynamic Type, Optional Parameters
C# 5.0	2012	async/await
C# 6.0	2015	String interpolation, Expression-bodied members
C# 7.0	2017	Tuples, Pattern Matching
C# 8.0	2019	Nullable Reference Types, Switch Expressions
C# 9.0	2020	Records, Init-only setters
C# 10.0	2021	Global usings, File-scoped namespace
C# 11.0	2022	Raw string literals, Required members
C# 12.0	2023	Collection expressions, Primary constructors

☐ แต่ละเวอร์ชันของ C# พัฒนาไปพร้อมกับ .NET

☐ **.NET Evolution**

เวอร์ชัน	ปี	หมายเหตุ
.NET Framework 1.0	2002	เริ่มต้นเฉพาะ Windows
.NET Framework 2.0–4.8	2005–2019	รองรับ WinForms, WPF, ASP.NET
.NET Core 1.0	2016	เริ่มเปิดเป็น Cross-platform
.NET Core 2.0–3.1	2017–2019	เพิ่มความเสถียร
.NET 5.0	2020	รวม .NET Core + Mono
.NET 6.0 (LTS)	2021	รองรับ MAUI, Blazor
.NET 7.0	2022	Performance-focused
.NET 8.0 (LTS)	2023	Native AOT, Cloud optimized

เวอร์ชัน	ปี	หมายเหตุ
.NET 9.0	2024	[กำลังพัฒนา]

หลักการสำคัญของ .NET

CLR (Common Language Runtime)

- เปรียบเสมือน “Virtual Machine” ของ .NET
- ทำหน้าที่:
  - Compile CIL (Intermediate Language) ⟶ Machine Code
  - จัดการ Memory และ Garbage Collection
  - จัดการ Exception, Security, Threading
  - สนับสนุนภาษา .NET ทุกภาษา

BCL (Base Class Library)

- เป็นชุดคลาสที่ใช้ร่วมกัน เช่น:
  - System, System.IO, System.Collections, System.Net
- เขียนโค้ดครั้งเดียว ใช้ได้หลายภาษา

⚙ Managed vs Unmanaged Code

ประเภท	รายละเอียด
Managed	โค้ดที่รันใน CLR เช่น C#, VB.NET
Unmanaged	โค้ดระดับล่าง เช่น C/C++, Win32 API

ความสำคัญในยุคปัจจุบัน

- ปัจจุบัน .NET กลายเป็น **แพลตฟอร์ม unified ที่แข็งแกร่ง**
- ใช้ในองค์กรระดับโลก เช่น Microsoft, StackOverflow, Alibaba, Siemens
- รองรับทั้ง Web, Desktop, Mobile, Cloud, Game, IoT
- มีเครื่องมือพัฒนาที่ทรงพลัง เช่น:
  - Visual Studio
  - Visual Studio Code
  - JetBrains Rider

สรุปประวัติ C# และ .NET

หัวข้อ	รายละเอียด
ผู้สร้าง	Microsoft (C#: Anders Hejlsberg)

หัวข้อ	รายละเอียด
ปีเริ่มต้น	2000 (เปิดตัวจริงในปี 2002)
จุดประสงค์	แก้ปัญหาการพัฒนาแอปที่ซับซ้อนของ Windows, แข่งกับ Java
จุดเด่น	OOP, Garbage Collector, Multiplatform, Interoperability
การเติบโต	พัฒนาอย่างต่อเนื่อง และกลายเป็นแพลตฟอร์มระดับโลก

.NET Framework vs .NET Core vs .NET 5+ (แบบละเอียด เชิงลึก)

□ บริบทโดยรวม

.NET คือ **แพลตฟอร์มพัฒนาแอปพลิเคชันแบบ Multi-language และ Multi-platform** ที่ให้คุณสร้างแอป:

- Web
- Desktop
- Mobile
- Cloud
- Game
- IoT

  โดยใช้ภาษาอย่างเช่น **C#, F#, VB.NET**

แต่ในอดีต .NET ถูกแยกออกเป็น **หลายเวอร์ชัน/สาขา** ที่มีข้อจำกัดต่างกัน เช่น:

- .NET Framework ⟶ ใช้ได้แค่ Windows
- .NET Core ⟶ Cross-platform
- .NET 5+ ⟶ รวมทุกอย่างไว้ในแพลตฟอร์มเดียว

□ .NET Framework

□ **ประวัติ:**

- เปิดตัวปี 2002
- ใช้พัฒนา Windows Apps และ Web Apps (ASP.NET Web Forms, MVC)

□ **สถาปัตยกรรม:**

- ทำงานเฉพาะบน **Windows OS**
- มีเครื่องมืออย่าง **WPF, Windows Forms, ASP.NET**
- ทำงานบน **CLR (Common Language Runtime)**

⚠□ **ข้อจำกัด:**

- **ไม่รองรับ Cross-platform**
- ไม่เปิด Open Source
- ระบบ Legacy เยอะ (หนัก)
- ไม่รองรับเทคโนโลยีสมัยใหม่ เช่น WebAssembly, Docker

☐ **สถานะปัจจุบัน:**

- **หยุดพัฒนา**แล้วที่เวอร์ชัน **.NET Framework 4.8**
- ยังมีการอัปเดตเล็กน้อยเพื่อสนับสนุนระบบเก่าในองค์กร

☐ **.NET Core**

☐ **จุดเริ่มต้น:**

- เปิดตัวปี 2016 เพื่อแก้ปัญหาของ .NET Framework
- สร้างใหม่จากศูนย์ ให้เป็น **Open Source** และ **Cross-platform**

☐ **จุดเด่น:**

ด้าน	รายละเอียด
Cross-platform	ทำงานได้บน Windows, Linux, macOS
Performance	เร็วขึ้นและเบากว่า Framework
Side-by-side	ติดตั้งหลายเวอร์ชันในเครื่องเดียวกันได้
Container-friendly	ทำงานกับ Docker ได้ดี
Open Source	มี Community ช่วยพัฒนา (ผ่าน GitHub)
Modern API	สนับสนุนเทคโนโลยีใหม่ เช่น gRPC, Web API, Blazor

☐ **จุดอ่อน:**

- ยังไม่รองรับบางเทคโนโลยีเก่าของ Windows เช่น WCF, WebForms
- ต้องอาศัย Visual Studio 2017+ หรือ CLI

☐ **.NET 5+ (Unified Platform)**

☐ **จุดเริ่มต้นของการรวม:**

- Microsoft ประกาศในปี 2019 ว่า:

“จะรวม .NET Framework, .NET Core และ Mono/Xamarin เข้าด้วยกัน เป็นแพลตฟอร์มเดียวที่เรียกว่า **.NET** (เริ่มจากเวอร์ชัน 5 ขึ้นไป)”

☐ **เป้าหมายของ .NET 5+:**

1. ใช้ **Base Class Library เดียวกัน** ทุกแพลตฟอร์ม
2. มี **Runtime เดียว (CoreCLR)**

3. รองรับทุกแพลตฟอร์ม: Desktop, Web, Mobile, Cloud, IoT
4. สนับสนุน Tools ใหม่ๆ เช่น:
    - MAUI (Mobile/Desktop)
    - Blazor (WebAssembly)
    - Native AOT
    - Cloud-optimized Runtime

□ เปรียบเทียบโดยละเอียด

คุณสมบัติ	.NET Framework	.NET Core	.NET 5+
รองรับระบบปฏิบัติการ	Windows เท่านั้น	Windows, Linux, macOS	Windows, Linux, macOS
Cross-platform	□	□	□
Open-source	□	□	□
Performance	ปานกลาง	ดี	ดียิ่งขึ้น
Deployment	Global-only	Side-by-side	Side-by-side
Language Support	C#, VB.NET, F#	C#, F# (VB บางส่วน)	C#, F# (VB บางส่วน)
GUI Framework	WinForms, WPF	WinForms, WPF (Windows only)	WinForms, WPF, MAUI
Web Framework	ASP.NET	ASP.NET Core	ASP.NET Core, Blazor
Mobile Support	□	Xamarin (แยก)	□ (ผ่าน .NET MAUI)
Game Development	□	Unity (บางส่วน)	Unity / Mono-based
Target Audience	Enterprise legacy	Modern Dev / Cloud	Unified for All
สถานะ	Legacy / คงที่	ยังใช้งานได้	**แนะนำสำหรับอนาคต**

□ Roadmap ของ .NET (อย่างเป็นทางการ)

เวอร์ชัน	ปี	ประเภท	หมายเหตุ
.NET Core 3.1	2019	LTS	เสถียร, รองรับ WPF
.NET 5	2020	Current	เริ่ม unified platform

เวอร์ชัน	ปี	ประเภท	หมายเหตุ
.NET 6	2021	**LTS**	แนะนำสำหรับโปรดักชัน
.NET 7	2022	Current	Performance-focused
.NET 8	2023	**LTS**	MAUI, AOT, Cloud
.NET 9	2024	Current	[พัฒนาอยู่]
.NET 10	2025	**LTS**	[แผนล่วงหน้า]

ℹ **LTS (Long-Term Support)** = สนับสนุน 3 ปี

ควรใช้เวอร์ชัน LTS สำหรับระบบในองค์กร/การใช้งานจริง

สรุปแนวทางการเลือกใช้งาน

คุณต้องการ	แนะนำใช้
พัฒนาแอป Windows แบบเก่า (Legacy)	.NET Framework (เฉพาะจำเป็น)
พัฒนาแอป Web หรือ Desktop แบบทันสมัย	.NET 6+
พัฒนา Web API, Blazor, MAUI, Cloud	.NET 6+ / .NET 8
ทำงานบน Linux / Docker / macOS	.NET Core / .NET 5+
โปรเจกต์ใหม่ทั้งหมด	**.NET 6 LTS หรือใหม่กว่า**

ข้อคิดเชิงกลยุทธ์

- .NET Framework คืออดีตที่ยิ่งใหญ่ แต่จำกัดบน Windows
- .NET Core คือการเปลี่ยนผ่านสู่ยุค Cross-platform
- .NET 5+ คืออนาคตที่รวมทุกสิ่งไว้ในแพลตฟอร์มเดียว

หากคุณจะเริ่มเรียน C# และ .NET วันนี้ — **ใช้ .NET 6 หรือ .NET 8** เท่านั้น

เพราะนั่นคือสิ่งที่ Microsoft และชุมชนทั่วโลกกำลังพัฒนาและสนับสนุน

สรุปสุดท้าย

หัวข้อ	ข้อมูลสำคัญ
.NET Framework	Legacy, Windows-only, ไม่ขยายอีกแล้ว
.NET Core	Open-source, Cross-platform, เร็ว
.NET 5+	Unified, รองรับทุกแพลตฟอร์ม, พร้อมอนาคต

การติดตั้ง .NET SDK และ Visual Studio / VS Code (เชิงลึก)

☐ **สิ่งที่ต้องรู้ก่อนติดตั้ง**

☐ **คำศัพท์พื้นฐาน:**

คำศัพท์	ความหมาย
.NET SDK	ชุดเครื่องมือพัฒนาสำหรับเขียนและคอมไพล์ C# (Compiler, CLI, Libraries)
.NET Runtime	ส่วนที่ใช้รันโปรแกรมที่เขียนด้วย .NET (ไม่สามารถเขียนหรือ build ได้)
Visual Studio (VS)	IDE (Integrated Development Environment) ที่สมบูรณ์แบบจาก Microsoft
Visual Studio Code (VS Code)	Text Editor ที่เบา รองรับหลายภาษา ติดตั้ง Extension เพื่อใช้กับ .NET ได้

☐ **ส่วนที่ 1: ติดตั้ง .NET SDK**

☐ **ดาวน์โหลด .NET SDK**

1. เข้าลิงก์:
   - ☐ https://dotnet.microsoft.com/download
2. เลือกเวอร์ชันที่แนะนำ:
   - ☐ ใช้เวอร์ชัน **.NET 8 (LTS)** หรือ **.NET 6 (LTS)**

   (ถ้าไม่แน่ใจ – ใช้ .NET 8 ไปเลย)
3. ดาวน์โหลดตามระบบปฏิบัติการของคุณ:
   - **Windows** ⟶ .exe installer
   - **macOS** ⟶ .pkg
   - **Linux** ⟶ ตาม distro (apt/yum/zypper)

⚙☐ **ตรวจสอบหลังติดตั้ง**

เปิด **Command Prompt** หรือ **Terminal** แล้วพิมพ์:

```
dotnet --version
```

☐ หากขึ้นเวอร์ชัน เช่น 8.0.100 แปลว่าติดตั้งเรียบร้อย

☐ หากขึ้น dotnet ไม่พบ ⟶ ต้องรีสตาร์ทเครื่อง หรือเช็ก PATH

☐ **ส่วนที่ 2: ติดตั้ง Visual Studio (VS) [เหมาะสำหรับผู้เริ่มต้น]**

☐ **ดาวน์โหลด Visual Studio**

1. เข้าเว็บ:
   - ☐ https://visualstudio.microsoft.com
2. เลือกเวอร์ชัน:
   - ☐ **Community Edition (ฟรี)**
   - สำหรับผู้เริ่มต้น – เหมาะที่สุด

☐ **ติดตั้ง Workload ที่จำเป็น**

1. หลังเปิดตัวติดตั้ง ให้เลือก:
   - ☐ .NET desktop development
   - ☐ .NET and web development *(ถ้าจะเขียน Web)*
   - ☐ (ใช้พื้นที่ ~3-6GB)
2. คลิก Install แล้วรอ

☐ **ความสามารถของ Visual Studio:**

ความสามารถ	รายละเอียด
IntelliSense	ช่วยเติมโค้ดอัตโนมัติ
Debugger	ไล่บั๊กแบบ real-time
Designer UI	ใช้ออกแบบหน้าจอ (WinForms, WPF)
Git Built-in	ใช้งานร่วมกับ Git ได้เลย
Template	สร้างโปรเจกต์หลากหลายแบบ

☐ **ส่วนที่ 3: ติดตั้ง Visual Studio Code (VS Code) [เหมาะสำหรับผู้ที่ต้องการเบา/เร็ว]**

☐ **ดาวน์โหลด VS Code**

1. เข้าเว็บ:
   - ☐ https://code.visualstudio.com
2. ดาวน์โหลดและติดตั้งตามระบบปฏิบัติการ

☐ **ติดตั้ง Extension ที่จำเป็น**

เปิด VS Code ⟶ ไปที่ Extensions (ไอคอนสี่เหลี่ยมด้านซ้าย) แล้วติดตั้ง:

Extension Name	คำอธิบาย
☐ **C#** (โดย Microsoft)	รองรับ C#, IntelliSense, Debug
☐ **.NET Install Tool**	ตรวจสอบ/ติดตั้ง .NET Runtime
☐ **Code Runner** *(ไม่บังคับ)*	รันไฟล์เดี่ยวได้รวดเร็ว

⚙□ ทดสอบเครื่องมือ

1. เปิด Terminal หรือ Command Prompt

2. สร้างโปรเจกต์ใหม่:

```
dotnet new console -n HelloWorldApp
cd HelloWorldApp
dotnet run
```

3. หากแสดงผล:

```
Hello, World!
```

□ แสดงว่าทุกอย่างพร้อมแล้ว!

□ ทางเลือกเครื่องมือ (เพิ่มเติม)

เครื่องมือ	ใช้เมื่อ
Rider (JetBrains)	ต้องการ IDE แบบมืออาชีพ (เสียเงิน)
MonoDevelop	สำหรับ Linux ที่ต้องการ GUI IDE
Command-line only	ใช้ dotnet CLI ทั้งหมด (นักพัฒนา DevOps, Server)

□ สรุป

สิ่งที่ต้องติดตั้ง	เหมาะกับใคร	หมายเหตุ
□ .NET SDK	ทุกคน	จำเป็นพื้นฐาน
□ Visual Studio	ผู้เริ่มต้น, Windows	ฟีเจอร์ครบ
□ VS Code	สายเบา, Linux, macOS	เร็ว, ยืดหยุ่น
□ Extensions	C# (VS Code)	ให้รองรับภาษาและ IntelliSense

สร้างโปรเจกต์ Console Application แรก (ด้วย .NET SDK + Visual Studio / VS Code)

□ เป้าหมาย

- รู้จักคำสั่ง dotnet new, dotnet run, dotnet build
- เข้าใจโครงสร้างของโปรเจกต์ C#
- แก้ไขและรันโปรแกรมได้ด้วยตนเอง

☐ **วิธีที่ 1: ใช้ Command Line (CLI) + VS Code**

☐ **ขั้นตอน:**

1. **เปิด Command Prompt / Terminal**
2. สร้างโปรเจกต์:

```
dotnet new console -n HelloWorldApp
cd HelloWorldApp
```

คำสั่งนี้จะ:

- สร้างโฟลเดอร์ชื่อ HelloWorldApp
- สร้างไฟล์ C# พื้นฐานให้ครบถ้วน

3. **เปิดโฟลเดอร์ใน VS Code:**

```
code .
```

4. แก้ไขไฟล์ Program.cs:

```
using System;

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("สวัสดีชาวโลกจาก C#!");
    }
}
```

5. **รันโปรแกรม:**

```
dotnet run
```

☐ ผลลัพธ์:

```
สวัสดีชาวโลกจาก C#!
```

☐ **วิเคราะห์โค้ด:**

```
using System;                // เรียกใช้ namespace ที่มี Console
class Program                // ประกาศคลาสหลัก
{
    static void Main(string[] args)  // เมธอดหลักของโปรแกรม (entry point)
    {
        Console.WriteLine("...");    // แสดงข้อความบนหน้าจอ
    }
```

}

□ โครงสร้างไฟล์ที่ได้จาก dotnet new console

HelloWorldApp/

├── HelloWorldApp.csproj ← ไฟล์โปรเจกต์ (ตั้งค่าการ build)

├── Program.cs ← ไฟล์โค้ดหลัก

□ คำสั่งพื้นฐานอื่นที่ควรรู้:

คำสั่ง	ความหมาย
dotnet build	คอมไพล์โปรเจกต์ (ไม่รัน)
dotnet run	คอมไพล์ + รัน
dotnet clean	ล้างไฟล์ที่ build แล้ว
dotnet publish	เตรียมไฟล์สำหรับการ deploy
dotnet new --list	ดู template ทั้งหมด

□ วิธีที่ 2: ใช้ Visual Studio GUI (สำหรับมือใหม่ที่ใช้ Windows)

1. เปิด Visual Studio ⟶ กด “Create a new project”
2. เลือก “Console App (.NET Core)” หรือ “Console App (.NET 6/7/8)”
3. กำหนดชื่อเช่น HelloWorldApp
4. คลิก “Create”
5. Visual Studio จะเปิดไฟล์ Program.cs ให้พร้อม

□ เปลี่ยนโค้ด:

```
Console.WriteLine("Hello from Visual Studio!");
```

6. กดปุ่ม ▶□ “Start” หรือ F5 ⟶ โปรแกรมจะรัน

□ เคล็ดลับเพิ่มเติม

เปลี่ยนรูปแบบการเขียนจาก top-level ⟶ explicit method:

.NET 6/7/8 ใช้รูปแบบใหม่ที่ไม่มี Main() ให้เห็น:

```
Console.WriteLine("Hello");
```

ถ้าคุณต้องการใช้โครงสร้างแบบดั้งเดิมเพื่อเรียนรู้ชัดเจนขึ้น:

```
using System;
```

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello");
    }
}
```

☐ **สรุป**

สิ่งที่คุณได้เรียนรู้	รายละเอียด
การสร้างโปรเจกต์ C#	dotnet new console -n <ชื่อโปรเจกต์>
การรันโปรแกรม	dotnet run
โครงสร้างไฟล์	Program.cs, .csproj
จุดเริ่มต้นของ C#	static void Main(string[] args)
ใช้เครื่องมือใดได้บ้าง	CLI / Visual Studio / VS Code

ต่อไปนี้ผมจะจัดตัวอย่างโปรแกรม **Console Application ด้วยภาษา C#** โดยแบ่งเป็น:

☐ **สารบัญ**

☐ **โปรแกรมพื้นฐาน (3 ตัวอย่าง)**

1. โปรแกรมคำนวณพื้นที่สี่เหลี่ยม
2. โปรแกรมแปลงอุณหภูมิ
3. โปรแกรมหาผลรวมและค่าเฉลี่ยของตัวเลข

☐ **โปรแกรมแนวประยุกต์ (3 ตัวอย่าง)**

4. โปรแกรมจัดการรายการสินค้าในตะกร้า (List + Menu)
5. โปรแกรมตรวจสอบรหัสผ่าน (String + Logic)
6. โปรแกรมสุ่มเลขล็อตเตอรี่ (Random + เงื่อนไข)

☐ **โปรแกรมบูรณาการ (3 ตัวอย่าง)**

7. โปรแกรมจองที่นั่งโรงหนัง (Array + Menu + Logic)
8. โปรแกรมระบบลงทะเบียนนักเรียน (List + Class + Validation)
9. โปรแกรมคำนวณเงินเดือนพนักงาน (OOP + Switch + Input validation)

☐ **ตัวอย่างที่ 1: โปรแกรมคำนวณพื้นที่สี่เหลี่ยม**

□ โครงสร้างโปรเจกต์

```
RectangleArea/
├── RectangleArea.csproj
└── Program.cs
```

□ Program.cs

```
using System;

class Program
{
    static void Main()
    {
        Console.Write("กรอกความกว้าง: ");
        double width = Convert.ToDouble(Console.ReadLine());

        Console.Write("กรอกความยาว: ");
        double height = Convert.ToDouble(Console.ReadLine());

        double area = width * height;

        Console.WriteLine($"พื้นที่ของสี่เหลี่ยมคือ {area} ตารางหน่วย");
    }
}
```

□ คำอธิบายโค้ด

- รับค่ากว้างและยาวจากผู้ใช้ผ่าน Console.ReadLine()
- แปลงค่าจาก string เป็น double
- คำนวณพื้นที่ด้วยสูตร area = width * height
- แสดงผลผ่าน Console.WriteLine()

▶□ ผลการรัน

```
กรอกความกว้าง: 5
กรอกความยาว: 10
พื้นที่ของสี่เหลี่ยมคือ 50 ตารางหน่วย
```

□ ตัวอย่างที่ 2: โปรแกรมแปลงอุณหภูมิ (Celsius ⟶ Fahrenheit)

```
using System;
```

```
class Program
{
    static void Main()
    {
        Console.Write("กรอกอุณหภูมิ (องศาเซลเซียส): ");
        double celsius = Convert.ToDouble(Console.ReadLine());

        double fahrenheit = (celsius * 9 / 5) + 32;

        Console.WriteLine($"เท่ากับ {fahrenheit} °F");
    }
}
```

▶ **ผลการรัน**

```
กรอกอุณหภูมิ (องศาเซลเซียส): 30
เท่ากับ 86 °F
```

ตัวอย่างที่ 3: โปรแกรมหาผลรวมและค่าเฉลี่ย

```
using System;

class Program
{
    static void Main()
    {
        Console.Write("ใส่จำนวนตัวเลข: ");
        int count = Convert.ToInt32(Console.ReadLine());

        double sum = 0;

        for (int i = 1; i <= count; i++)
        {
            Console.Write($"ตัวเลขที่ {i}: ");
            double num = Convert.ToDouble(Console.ReadLine());
            sum += num;
```

```
        }

        double avg = sum / count;
        Console.WriteLine($"ผลรวม: {sum}");
        Console.WriteLine($"ค่าเฉลี่ย: {avg}");
    }
}
```

▶ **ผลการรัน**

```
ใส่จำนวนตัวเลข: 3
ตัวเลขที่ 1: 10
ตัวเลขที่ 2: 20
ตัวเลขที่ 3: 30
ผลรวม: 60
ค่าเฉลี่ย: 20
```

ตัวอย่างที่ 4: โปรแกรมจัดการรายการสินค้าในตะกร้า (List + Menu)

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
        List<string> cart = new List<string>();
        string input;

        do
        {
            Console.WriteLine("\n1. เพิ่มสินค้า");
            Console.WriteLine("2. แสดงสินค้า");
            Console.WriteLine("3. ออกจากโปรแกรม");
            Console.Write("เลือกเมนู: ");
            input = Console.ReadLine();
```

```
            switch (input)
            {
                case "1":
                    Console.Write("ชื่อสินค้า: ");
                    string item = Console.ReadLine();
                    cart.Add(item);
                    Console.WriteLine("□ เพิ่มสินค้าเรียบร้อย");
                    break;

                case "2":
                    Console.WriteLine("□ รายการสินค้าในตะกร้า:");
                    foreach (string p in cart)
                        Console.WriteLine($"- {p}");
                    break;
            }

        } while (input != "3");
    }
}
```

▶□ ผลการรัน

```
1. เพิ่มสินค้า
2. แสดงสินค้า
3. ออกจากโปรแกรม
เลือกเมนู: 1
ชื่อสินค้า: นม
□ เพิ่มสินค้าเรียบร้อย

เลือกเมนู: 2
□ รายการสินค้าในตะกร้า:
- นม
```

□ ตัวอย่างที่ 5: โปรแกรมตรวจสอบรหัสผ่าน

```
using System;
```