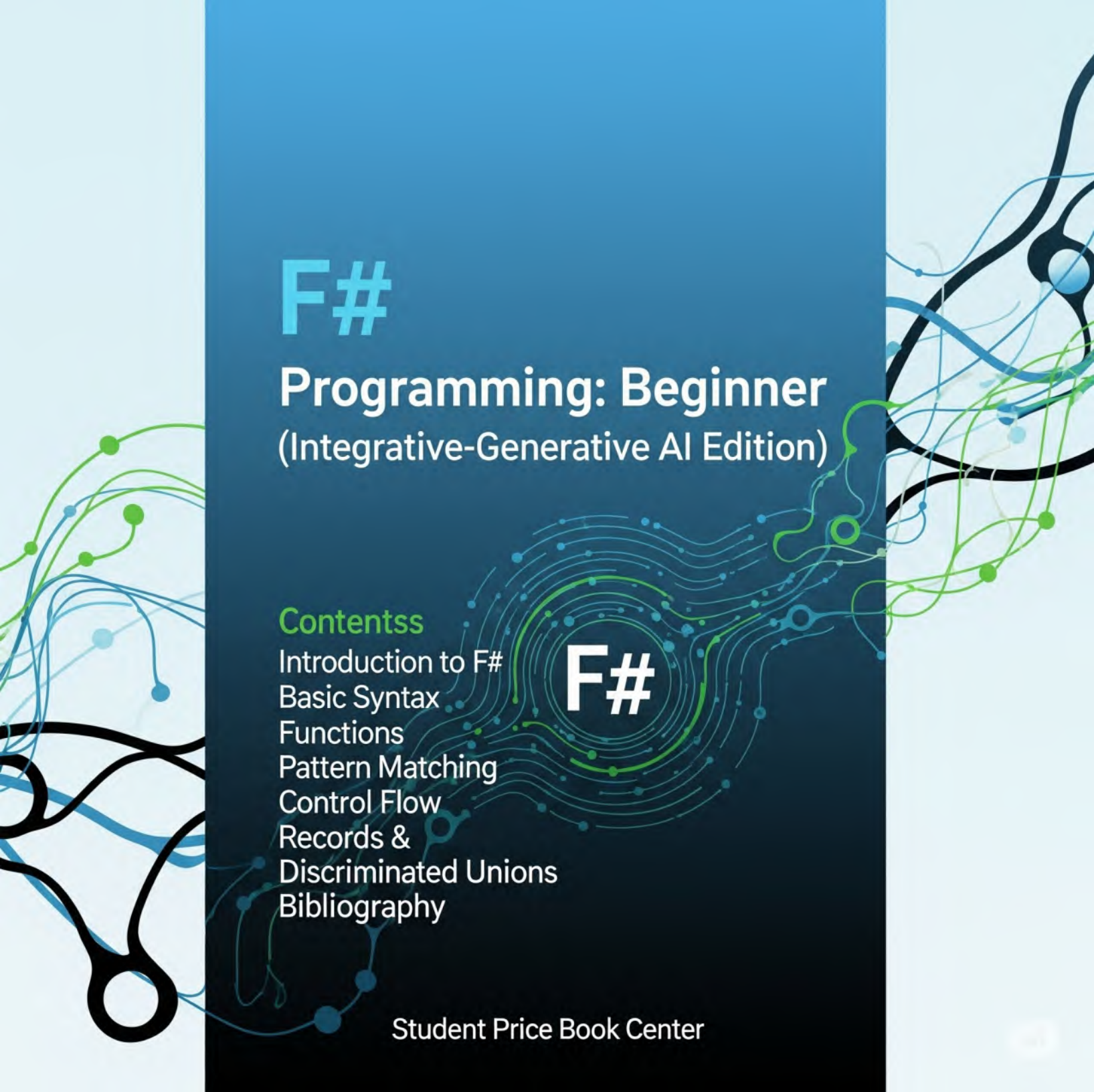
F#

Programming: Beginner

(Integrative-Generative AI Edition)

Contentss

Introduction to F#
Basic Syntax
Functions
Pattern Matching
Control Flow
Records &
Discriminated Unions
Bibliography



Student Price Book Center

คำนำ

การเขียนโปรแกรมสมัยใหม่ไม่ได้จำกัดอยู่เพียงแค่การสั่งให้คอมพิวเตอร์ทำงานตามลำดับคำสั่งแบบดั้งเดิมอีกต่อไป โลกของซอฟต์แวร์ยุคปัจจุบันต้องการความยืดหยุ่น ความสามารถในการจัดการข้อมูลที่ซับซ้อน และการออกแบบโค้ดที่ปลอดภัยต่อความผิดพลาด หนึ่งในภาษาที่ตอบโจทย์ความต้องการเหล่านี้ได้อย่างชัดเจนคือ **F#** ซึ่งเป็นภาษา functional-first บนแพลตฟอร์ม .NET โดย F# ไม่เพียงแต่สนับสนุนการเขียนโปรแกรมเชิงฟังก์ชันเท่านั้น แต่ยังสามารถประยุกต์ใช้แนวคิดเชิงวัตถุ (OOP) และ imperative programming ได้ตามความเหมาะสม

หนังสือเล่มนี้จัดทำขึ้นเพื่อเป็นแนวทางสำหรับผู้เริ่มต้นที่สนใจเรียนรู้ F# ตั้งแต่พื้นฐานจนถึงการประยุกต์ใช้ในโปรแกรมจริง โดยเริ่มจาก **บทที่ 1: Introduction to F#** ซึ่งครอบคลุมประวัติ จุดเด่น ความแตกต่างระหว่าง F# กับ C# และ VB.NET ในบริบทของการพัฒนาเชิงฟังก์ชัน ตลอดจนการติดตั้ง .NET SDK และการสร้างโปรเจ็กต์ F# Console App ขั้นตอนเหล่านี้จะช่วยให้ผู้เรียนมีสภาพแวดล้อมพร้อมสำหรับการทดลองและพัฒนาโปรแกรมของตนเอง

ต่อมา **บทที่ 2: Basic Syntax** จะเน้นการทำความเข้าใจโครงสร้างไวยากรณ์พื้นฐานของ F# เช่น การประกาศตัวแปรด้วย let ที่ immutable ตามค่าเริ่มต้น การใช้ printfn เพื่อแสดงผล การทำงานกับชนิดข้อมูลพื้นฐาน Tuple, List และ Array รวมถึงการใช้ Pipe Operator (|>) ซึ่งเป็นเครื่องมือสำคัญในการจัดการข้อมูลแบบ functional การฝึกทำ Mini Project ขนาดเล็กในบทนี้จะช่วยให้ผู้เรียนสามารถนำแนวคิดไปใช้จริงได้ทันที

บทที่ 3: Functions อธิบายการสร้างและใช้งานฟังก์ชัน ซึ่งเป็นหัวใจของการเขียนโปรแกรมเชิงฟังก์ชัน F# ผู้เรียนจะได้เรียนรู้การประกาศฟังก์ชันธรรมดา การสร้าง **recursive functions** เพื่อแก้ปัญหาที่ซ้ำซ้อน การใช้ **higher-order functions** (HOF) และการประยุกต์ **lambda expressions** ในการสร้างฟังก์ชันแบบไม่ระบุชื่อ ความเข้าใจในบทนี้จะช่วยให้สามารถสร้างฟังก์ชันที่ยืดหยุ่น นำกลับมาใช้ซ้ำได้ และสอดคล้องกับแนวคิด functional-first

ใน **บทที่ 4: Pattern Matching** ผู้อ่านจะได้เรียนรู้เทคนิคการจับคู่รูปแบบของข้อมูลอย่างมีประสิทธิภาพ เช่น การใช้ match ... with การใช้ wildcards _ เพื่อระบุค่าที่ไม่สนใจ การจับคู่กับ Tuples และ Lists และการใช้ **guards (when)** เพื่อกรองเงื่อนไขเฉพาะ การฝึกทำตัวอย่างบูรณาการจะช่วยให้ผู้เรียนสามารถจัดการข้อมูลซับซ้อนและเขียนโค้ดที่อ่านง่าย กระชับ และปลอดภัย

บทที่ 5: Control Flow จะอธิบายการควบคุมลำดับการทำงานของโปรแกรม F# ตั้งแต่การตัดสินใจเชิงเงื่อนไขด้วย if / else การทำซ้ำด้วย loops แบบ for และ while ไปจนถึงการใช้ **recursive loop** ซึ่งเป็นแนวทาง functional สำหรับการทำซ้ำ เทคนิคเหล่านี้ช่วยให้ผู้เรียนสามารถออกแบบโปรแกรมที่มีตรรกะชัดเจนและยืดหยุ่น

ในที่สุด **บทที่ 6: Records & Discriminated Unions** จะเน้นการสร้างโครงสร้างข้อมูลที่ปลอดภัยและอ่านง่าย ผ่าน **record types** สำหรับเก็บข้อมูลเชิงโครงสร้าง **discriminated unions**

(DU) สำหรับข้อมูลหลายรูปแบบ และ **option type** สำหรับค่าที่อาจไม่มี การใช้โครงสร้างข้อมูลเหล่านี้ร่วมกับฟีเจอร์ functional ของ F# จะช่วยให้โค้ดมีความน่าเชื่อถือและสามารถต่อยอดไปสู่โปรแกรมที่ซับซ้อนยิ่งขึ้น

ตลอดเล่มนี้ ผู้เรียนจะได้ทดลองทำ **Mini Project** และตัวอย่างบูรณาการที่สอดคล้องกับแต่ละบท เพื่อให้สามารถนำความรู้ไปประยุกต์ใช้ในโปรแกรมจริง ทั้งนี้ หนังสือเล่มนี้ตั้งใจให้ผู้อ่านได้เรียนรู้ F# อย่างเป็นขั้นตอน ตั้งแต่พื้นฐาน จนถึงการสร้างโค้ดเชิงฟังก์ชันที่มีคุณภาพและสามารถนำไปต่อยอดในงานพัฒนาโปรแกรมเชิงอุตสาหกรรม

ด้วยโครงสร้างการเรียนรู้แบบเป็นขั้นตอน และตัวอย่างการเขียนโค้ดจริง หนังสือเล่มนี้จะเป็นเพื่อนคู่มือที่ช่วยให้ผู้เริ่มต้นเข้าใจและใช้งาน F# ได้อย่างมั่นใจ พร้อมทั้งเป็นรากฐานที่แข็งแรงสำหรับการเรียนรู้แนวคิดการเขียนโปรแกรมเชิงฟังก์ชันและการพัฒนาซอฟต์แวร์ในระดับสูงต่อไป

ด้วยรักและปรารถนาดี

ศูนย์หนังสือราคานักเรียน

สารบัญ

หน้า

บทที่ 1 Introduction to F# (Introduction to F#)..... 1

- Introduction to F#
- Introduction to F# ให้เป็น รายละเอียดเชิงลึก (Deep Dive)
- ประวัติและจุดเด่นของ F#
- ความแตกต่างจาก C# / VB.NET ในบริบทของ F#
- Functional-first, แต่ใช้ OOP และ Imperative ได้
- การติดตั้ง .NET SDK และ F#
- การสร้างโปรเจ็กต์ F# Console App

บทที่ 2 Basic Syntax (Basic Syntax) 40

- Basic Syntax
- รายละเอียดเชิงลึกของบทที่ 2: Basic Syntax
- การเขียน printfn ใน F#
- ตัวแปร let (Immutable by default)
- ชนิดข้อมูลพื้นฐาน (Primitive Types) ใน F#
- Tuple, List และ Array ใน F#
- Pipe Operator (|>) ใน F#
- Mini Project F# – Basic Syntax Examples

บทที่ 3 Functions (Functions) 90

- Functions in F#
- Functions ใน F# เชิงลึก
- การประกาศฟังก์ชันใน F#
- Recursive Functions
- Higher-Order Functions (HOF) ใน F#)
- Lambda Expressions ใน F#
- ตัวอย่าง Mini Project บูรณาการ

บทที่ 4 Pattern Matching (Pattern Matching).....129

- Pattern Matching
- Pattern Matching ใน F# แบบเชิงลึก
- การใช้ match ... with
- Wildcards _ ใน F#
- Matching with Tuples และ Lists ใน F#
- Guards (when) ใน F#
- ตัวอย่างบูรณาการ

บทที่ 5 Control Flow (Control Flow).....172

- Control Flow
- รายละเอียดเชิงลึกของบทที่ 5: Control Flow ใน F#
- If / Else ใน F#
- Loops ใน F# ทั้ง for และ while
- Recursive Loop ใน F#
- ตัวอย่างบูรณาการ

บทที่ 6 Records & Discriminated Unions (Records & Discriminated Unions) 207

- Records & Discriminated Unions
- รายละเอียดเชิงลึกของบทที่ 6: Records & Discriminated Unions + Option Types ใน F#
- รายละเอียดเชิงลึกของ Record Types ใน F#
- รายละเอียดเชิงลึกของ Discriminated Unions (DU) ใน F#
- รายละเอียดเชิงลึกของ Option Type ใน F#

บรรณานุกรม 246

บทที่ 1

Introduction to F#

(Introduction to F#)

เนื้อหา

- Introduction to F#
- Introduction to F# ให้เป็น รายละเอียดเชิงลึก (Deep Dive)
- ประวัติและจุดเด่นของ F#
- ความแตกต่างจาก C# / VB.NET ในบริบทของ F#
- Functional-first, แต่ใช้ OOP และ Imperative ได้
- การติดตั้ง .NET SDK และ F#
- การสร้างโปรเจ็กต์ F# Console App

บทที่ 1: Introduction to F#

การพัฒนาโปรแกรมด้วยภาษา F# ได้รับความสนใจมากขึ้นเรื่อย ๆ ในแวดวงวิศวกรรมซอฟต์แวร์ โดยเฉพาะอย่างยิ่งในกลุ่มนักพัฒนาที่ต้องการความกระชับในการเขียนโค้ดและความสามารถในการจัดการกับปัญหาที่ซับซ้อนในเชิงคณิตศาสตร์และการประมวลผลข้อมูล ภาษานี้ถูกออกแบบมาเพื่อตอบโจทย์งานในลักษณะดังกล่าว ทั้งยังมีรากฐานมาจากตระกูลภาษา ML (Meta Language) และได้ถูกนำมาใช้อย่างจริงจังใน .NET Ecosystem ทำให้สามารถเข้าถึงเครื่องมือ ไลบรารี และโครงสร้างพื้นฐานเดียวกันกับภาษา C# และ VB.NET

จุดเด่นของ F# ที่ทำให้แตกต่างจากภาษาอื่นใน .NET Framework คือความสามารถในการผสมผสานแนวคิดเชิงฟังก์ชัน (functional programming) เข้ากับความยืดหยุ่นของการเขียนแบบเชิงวัตถุ (object-oriented programming) และเชิงคำสั่ง (imperative programming) ได้อย่างลงตัว นักพัฒนาจึงสามารถเลือกใช้แนวทางที่เหมาะสมกับปัญหาได้ ไม่ว่าจะเป็นการจัดการข้อมูลขนาดใหญ่ การสร้างอัลกอริทึมที่ซับซ้อน หรือแม้แต่การพัฒนาแอปพลิเคชันทั่วไป

เมื่อเปรียบเทียบกับภาษา C# และ VB.NET แล้ว F# มีลักษณะของไวยากรณ์ที่กระชับและลดความซ้ำซ้อนในโค้ด การเขียนโปรแกรมด้วย F# มักใช้รูปแบบ declarative ซึ่งเน้นการบอก “สิ่งที่ต้องการให้เกิดขึ้น” มากกว่าการระบุทีละขั้นตอนว่าจะทำอย่างไร ความแตกต่างนี้ทำให้โค้ดที่พัฒนาด้วย F# มีความชัดเจนและเข้าใจง่ายขึ้น ในขณะเดียวกันก็ยังคงสามารถเชื่อมโยงกับไลบรารีหรือระบบงานที่ถูกพัฒนาด้วยภาษา C# หรือ VB.NET ได้โดยตรง

F# ถูกออกแบบให้เป็นภาษา functional-first กล่าวคือ โครงสร้างและคุณลักษณะต่าง ๆ ของภาษาเน้นไปที่การเขียนโปรแกรมเชิงฟังก์ชันเป็นหลัก เช่น การใช้ immutable data และการส่งฟังก์ชันเป็นค่า แต่ก็ไม่ได้ปิดกั้นนักพัฒนาจากการใช้แนวทาง OOP หรือ imperative หากจำเป็น สิ่งนี้ทำให้ F# เป็นภาษาที่มีความยืดหยุ่นสูงและตอบโจทย์การพัฒนาหลากหลายประเภท ตั้งแต่การวิจัยเชิงวิชาการไปจนถึงการใช้งานในภาคธุรกิจ

เพื่อเริ่มต้นใช้งาน F# นักพัฒนาจำเป็นต้องติดตั้ง .NET SDK ซึ่งเป็นเครื่องมือพื้นฐานที่ Microsoft พัฒนาและเผยแพร่ การติดตั้ง .NET SDK จะทำให้ผู้ใช้สามารถเข้าถึงคอมไพเลอร์ เครื่องมือจัดการแพ็กเกจ และคำสั่งที่ใช้ในการสร้างโปรเจกต์ใหม่ การตั้งค่าสภาพแวดล้อมเช่นนี้ช่วยให้การเริ่มต้นเรียนรู้ F# เป็นไปอย่างราบรื่น ไม่ว่าจะทำงานบน Windows, macOS หรือ Linux

หลังจากติดตั้ง .NET SDK แล้ว ขั้นตอนต่อไปคือการสร้างโปรเจกต์ F# Console Application ซึ่งเป็นรูปแบบเบื้องต้นที่ใช้สำหรับเรียนรู้โครงสร้างภาษาและฝึกเขียนโค้ด คำสั่งที่ใช้คือ dotnet new console -lang F# ซึ่งจะสร้างไฟล์และโครงสร้างโปรเจกต์มาตรฐาน นักพัฒนาสามารถเปิดโค้ดดังกล่าวในโปรแกรมแก้ไข เช่น Visual Studio Code หรือ Visual Studio เพื่อทำการแก้ไขและรันโปรแกรมได้ทันที

สรุปได้ว่า บทแรกนี้ได้วางรากฐานความเข้าใจเกี่ยวกับภาษา F# โดยเริ่มตั้งแต่ประวัติ ความแตกต่างจากภาษาอื่นในตระกูล .NET ลักษณะของ functional-first ที่ยังคงรองรับ OOP และ imperative ไปจนถึงขั้นตอนการติดตั้งและการสร้างโปรเจกต์เบื้องต้น เนื้อหาเหล่านี้จะเป็นพื้นฐานสำคัญสำหรับการทำความเข้าใจบทต่อไป ที่จะเจาะลึกไปในเชิงไวยากรณ์ ฟีเจอร์ และรูปแบบการเขียนโปรแกรมด้วย F# อย่างเป็นระบบ

Introduction to F#

- ประวัติและจุดเด่นของ F#
- ความแตกต่างจาก C# / VB.NET
- Functional-first, แต่ใช้ OOP และ Imperative ได้
- การติดตั้ง .NET SDK และ F#
- การสร้างโปรเจ็กต์ F# Console App (dotnet new console -lang F#)

บทที่ 1: Introduction to F#

จุดประสงค์ของบท: เข้าใจประวัติ แนวคิดหลัก จุดเด่น/ข้อแตกต่างจาก C#/VB.NET และทำตามได้จริงตั้งแต่ติดตั้ง .NET SDK จนถึงสร้างและรันโปรเจ็กต์ F# แบบ Console

---

1) ประวัติและจุดเด่นของ F#

F# (เอฟชาร์ป) เป็นภาษาโปรแกรมเชิง *functional-first* ที่รันบน **.NET** พัฒนาโดยทีม Microsoft Research (ริเริ่มโดย Don Syme) และต่อมาถูกเปิดเป็นโอเพนซอร์สภายใต้ .NET Foundation ทำงานร่วมกับ C# และ VB.NET ได้เต็มรูปแบบ

จุดเด่นสำคัญ

- **Functional-first**: แนวคิด immutability, pure functions, higher-order functions, composition และ pattern matching ทำให้โค้ดกระชับและลดบั๊กเชิงสถานะ (state bugs)
- **Concise & Expressive**: syntax กะทัดรัด อ่านไหลลื่น (boilerplate น้อยกว่าภาษา OO ทั่วไป)
- **Strong typing + Type inference**: บังคับใช้ความถูกต้องตั้งแต่คอมไพล์ไทม์ แต่ยังเดา type ให้อัตโนมัติ (ไม่ต้องพิมพ์ type ซ้ำซ้อน)
- **Interoperability (.NET)**: ใช้ไลบรารี NuGet และโค้ด C#/VB.NET ที่มีอยู่เดิมได้ทันที
- **โมเดลข้อมูลทรงพลัง**: *Records* และ *Discriminated Unions (DU)* ทำให้ modeling domain ง่าย ชัด และปลอดภัยกว่า class hierarchy หลายชั้น
- **REPL (F# Interactive)**: ทดลองไอเดียได้รวดเร็วในสไตล์ data science และการคำนวณ

ตัวอย่างกรณีใช้งาน: quantitative finance, data engineering/science, web APIs (Giraffe/Saturn/Suave), งานที่ต้องการ domain modeling ชัดเจน, งานทั่วไปแทน C# ก็ได้

---

2) ความแตกต่างจาก C# / VB.NET (ภาพรวม)

หัวข้อ	F#	C# / VB.NET
กระบวนทัศน์	Functional-first (รองรับ OO/Imperative)	OO-first (รองรับ functional features)
ความกระชับ	โค้ดสั้นกว่า (type inference, pipe, pattern matching)	โครงสร้างชัดแต่ยาวกว่า
การจับคู่รูปแบบ	match + DU/Active Patterns ทรงพลัง	switch/pattern matching พัฒนาเร็วขึ้น แต่ DU แบบเดียวกันไม่มีโดยตรง
การจัดการ null	ใช้ option<'T> แบบ type-safe	มี null และ nullable types
โมเดลข้อมูล	record, DU ใช้แทน class hierarchy ได้ดี	เน้น class/struct/enum, มี record (C#) แต่ไม่มี DU ในตัว
async	Async workflows (async {} + Async.*)	async/await บน Task
การจัดเรียงโฟลว์	Pipelining `>และ composition>>`	

สั้น ๆ ด้วยโค้ดเทียบ

F#:

```
let square x = x * x
let sumOfSquares = [1..5] |> List.map square |> List.sum
printfn "%d" sumOfSquares
```

C#:

```
var sumOfSquares = Enumerable.Range(1, 5).Select(x => x * x).Sum();
Console.WriteLine(sumOfSquares);
```

---

3) Functional-first แต่ใช้ OOP และ Imperative ได้

F# เน้น functional แต่ **ไม่ห้าม** OO/Imperative คุณจึงผสมแนวทางให้เหมาะงานได้

3.1 Functional core: immutability, DU, pattern matching, pipelines

```
type Shape =
  | Circle of radius: float
  | Rectangle of width: float * height: float

let area shape =
  match shape with
  | Circle r -> System.Math.PI * r * r
  | Rectangle (w, h) -> w * h

[ Circle 5.0; Rectangle (4.0, 6.0) ]
|> List.map area
|> List.iter (printfn "Area = %f")
```

3.2 OOP interop: class, interface, members

```
type IGreeter =
  abstract member Greet : string -> string

// class ที่มี constructor และสมาชิก
 type Greeter(prefix: string) =
  interface IGreeter with
    member _.Greet name = $"{prefix}, {name}!"

let g = Greeter("Hello") :> IGreeter
printfn "%s" (g.Greet "F#")
```

3.3 Imperative: mutable/loops (ใช้เท่าที่จำเป็น)

```
let mutable total = 0
for i in 1 .. 5 do
  total <- total + i
printfn "Total = %d" total
```

แนวปฏิบัติ: ใช้ functional เป็นค่าเริ่มต้น แล้วหันไปใช้ mutable/loops เฉพาะจุดที่ให้ประโยชน์ด้านประสิทธิภาพหรือความชัดเจนจริง ๆ

---

4) การติดตั้ง .NET SDK และ F# (ทุกแพลตฟอร์ม)

แนะนำ: ติดตั้งจากเว็บไซต์ทางการของ .NET เพื่อได้รุ่นล่าสุด (รวม F# compiler/interactive มาให้ในSDK) แล้วตรวจสอบด้วย dotnet --version

4.1 ตรวจสอบก่อนว่ามีหรือยัง

```
dotnet --info
```

หากขึ้นข้อมูล SDK/Runtime แสดงว่ามีพร้อมใช้งานแล้ว

4.2 Windows (ทางเลือกยอดนิยม)

- ดาวน์โหลดตัวติดตั้ง .NET SDK จากเว็บทางการ แล้วกดติดตั้งตามขั้นตอน
- หรือใช้ **winget**:

```
winget install --id Microsoft.DotNet.SDK.8 -e
```

เปลี่ยนเลขเวอร์ชันให้เหมาะกับรุ่นล่าสุดในเวลาที่คุณติดตั้ง

4.3 macOS

- ดาวน์โหลดตัวติดตั้ง .pkg จากเว็บทางการ และติดตั้งตามขั้นตอน
- หรือใช้ **Homebrew (cask)**:

```
brew install --cask dotnet-sdk
```

หลังติดตั้ง อาจต้องเปิดแอป Terminal ใหม่แล้วรัน dotnet --version

4.4 Linux (ตัวอย่าง Ubuntu/Debian)

- วิธีที่เสถียรคือเพิ่ม Microsoft package repository แล้วติดตั้ง dotnet-sdk ด้วย apt
- โดยทั่วไปจะมีขั้นตอนประมาณ: เพิ่ม repo ⟶ sudo apt-get update ⟶ sudo apt-get install dotnet-sdk-<major>.<minor>
- ตรวจสอบด้วย dotnet --info

4.5 ติดตั้งเครื่องมือพัฒนา

- **VS Code + ส่วนขยาย Ionide (F#)** สำหรับ IntelliSense, F# Interactive, Debugger
- หรือ **Visual Studio** (Windows) เลือก workload .NET Desktop ที่มี F#

4.6 F# Interactive (REPL)

ทดลองโค้ดแบบโต้ตอบได้ทันที:

```
# บางแพลตฟอร์มใช้แบบนี้
```

dotnet fsi

หรือ

fsi

ใน REPL พิมพ์:

> let add a b = a + b;;

val add : a:int -> b:int -> int

> add 2 3;;

val it : int = 5

พิมพ์ #quit;; เพื่อออกจาก REPL

---

5) สร้างโปรเจ็กต์ F# Console App

คำสั่งพื้นฐาน: dotnet new console -lang "F#" -o HelloFs

5.1 สร้างโปรเจ็กต์

หมายเหตุ: ต้องใส่เครื่องหมายอัญประกาศรอบ F# เพราะสัญลักษณ์ # คือคอมเมนต์บน shell

จะใช้บน Windows PowerShell, cmd, macOS, Linux ได้เหมือนกัน

dotnet new console -lang "F#" -o HelloFs

cd HelloFs

โครงสร้างที่ได้ (โดยประมาณ):

HelloFs/

├── HelloFs.fsproj

└── Program.fs

5.2 เข้าใจไฟล์ F# ที่สำคัญ

- ***.fsproj**: ไฟล์โปรเจ็กต์ (.NET SDK style). **ลำดับไฟล์ F# สำคัญ**: ตัวคอมไพล์จะอ่านตามลำดับที่ระบุใน .fsproj
- **Program.fs**: จุดเริ่มโปรแกรม (มี main หรือ top-level statements)

ตัวอย่าง Program.fs ที่ชัดเจน:

open System

let square x = x * x

let sumOfSquares xs = xs |> List.map square |> List.sum

[<EntryPoint>]

let main argv =

```
let xs = [1..5]
printfn "Sum of squares = %d" (sumOfSquares xs)
0 // รหัสสถานะโปรเซส
```

5.3 รันและบิลด์

```
dotnet run
# หรือคอมไพล์เฉย ๆ
dotnet build
```

ตัวอย่างผลลัพธ์:

```
Sum of squares = 55
```

5.4 เพิ่มแพ็กเกจ NuGet (ตัวอย่าง)

```
dotnet add package FSharp.Data
```

จากนั้นในโค้ดสามารถอ้างอิงไลบรารีที่ติดตั้งได้ตามปกติ

---

6) เคล็ดลับเริ่มต้นให้ถูกทิศ

- **ยึด functional เป็นค่าเริ่มต้น**: ใช้ immutable และ pure functions ก่อนเสมอ หากต้องการประสิทธิภาพค่อยพิจารณา mutable
- **ใช้ DU/record แทน class hierarchy ที่ซับซ้อน** เพื่อให้ domain model ชัดเจนและ pattern matching ง่ายขึ้น
- **เรียบเรียงโค้ดด้วย pipe (|>) และ composition (>>)** เพื่อให้อ่านเหมือน data-flow
- **ทดสอบแนวคิดด้วย REPL (dotnet fsi)** แล้วค่อยย้ายเข้าโปรเจ็กต์
- **จัดลำดับไฟล์ใน .fsproj ให้ถูก** โดยวางไฟล์ที่เป็น dependency ไว้ก่อนไฟล์ที่ใช้งานมัน

---

7) แบบฝึกหัด (Hands-on)

1. ติดตั้ง .NET SDK และเปิด dotnet fsi ลองนิยามฟังก์ชัน factorial

```
let rec factorial n = if n <= 1 then 1 else n * factorial (n-1)
[0..6] |> List.map factorial
```

2. สร้างโปรเจ็กต์ dotnet new console -lang "F#" -o IntroFs แล้วแก้ Program.fs ให้:

- นิยาม type Temperature = C of float | F of float
- เขียนฟังก์ชัน toCelsius ด้วย match
- แสดงผลลัพธ์ผ่าน printfn

ตัวอย่างเฉลยสั้น ๆ:

```
type Temperature = C of float | F of float
let toCelsius t =
  match t with
```

```
  | C c -> c
  | F f -> (f - 32.0) * 5.0 / 9.0

[ C 30.0; F 86.0 ]
|> List.iter (fun t -> printfn "%A -> %.2f C" t (toCelsius t))
```

---

8) สรุปบท

- F# คือภาษา functional-first บน .NET ที่กระชับ ปลอดภัย และทำงานร่วมกับ ecosystem .NET ได้เต็มที่
- แตกต่างจาก C#/VB.NET ตรงที่มี DU, pattern matching, pipe/composition, และการจัดการ option แทน null
- ติดตั้ง .NET SDK ครั้งเดียวพร้อมใช้ F# ได้ทันที: dotnet --version, dotnet fsi
- สร้างโปรเจ็กต์ใหม่ด้วย dotnet new console -lang "F#" แล้วรันด้วย dotnet run

Introduction to F# ให้เป็น รายละเอียดเชิงลึก (Deep Dive)

บทที่ 1: Introduction to F# ให้เป็น **รายละเอียดเชิงลึก (Deep Dive)** โดยจะเพิ่ม **มิติด้านประวัติศาสตร์, technical design, การเปรียบเทียบเชิงลึกกับ C#/VB.NET, เบื้องหลัง functional-first + runtime model, และกลไกการ build/run ของ F# บน .NET**

☐ **Deep Dive — บทที่ 1: Introduction to F#**

1) ประวัติและวิวัฒนาการของ F#

- **ต้นกำเนิด (2002–2005):** Don Syme (นักวิจัย Microsoft Research Cambridge) พัฒนาจากแนวคิดของ ML/OCaml บน .NET CLR ⟶ ได้ภาษา functional-first ที่ยัง interop กับ .NET ได้
- **F# 1.0 (2005–2007):** เผยแพร่เวอร์ชันแรก เน้น community adoption
- **F# 2.0 (2010):** เพิ่ม async workflows (มาก่อน async/await ของ C# หลายปี), computation expressions
- **F# 3.0 (2012):** เพิ่ม Type Providers (พลิกเกมด้านข้อมูล ⟶ query ข้อมูล strongly-typed โดยไม่ต้องเขียน boilerplate)
- **F# 4.0–4.7 (2015–2019):** พัฒนาความเข้ากันได้กับ .NET Core/Standard, เพิ่ม syntax features
- **F# 5.0–7.0 (2020–2023):** บูรณาการกับ .NET 5/6/7+, ปรับปรุง performance, สนับสนุน interop กับ C# 9+/10+ ดีขึ้น

- **ปัจจุบัน:** อยู่ใน .NET 8+, ภายใต้ .NET Foundation, community-driven

เอกลักษณ์สำคัญ: F# คือการผสม **"functional purity + pragmatic interoperability"** — แตกต่างจาก Haskell ที่เน้น pure อย่างเข้มงวด แต่ก็ไม่ OO-first แบบ C# ⟶ ทำให้ F# ยืดหยุ่นสูงและ practical สำหรับ production .NET ecosystem

---

2) จุดเด่นเชิงลึก

- **Immutability by default:** ทุก binding (let) immutable ถ้าไม่ระบุ mutable ⟶ ช่วย reasoning, parallelism, และ concurrency
- **Discriminated Unions (DU):** เป็น "sum types" ที่ C# ไม่มีตรงตัว ⟶ ใช้ model domain ได้ชัด เช่น:
- type Payment = Cash of decimal | Card of string * decimal
- **Type Providers:** (F# 3.0+) ⟶ สร้าง type อัตโนมัติจาก schema จริง (เช่น JSON, XML, DB) แบบ compile-time safe ⟶ Productivity tool ที่ไม่พบใน C#
- **Computation Expressions:** Syntax sugar powerful เช่น async { }, seq { }, task { }, monadic workflows ⟶ ยืดหยุ่นเหมือน Haskell monads แต่ใช้งานง่ายกว่า
- **Interop:** ใช้ไลบรารี .NET ทั้งหมด (เช่น Entity Framework, ASP.NET Core) ได้ทันที โดยไม่ต้องเขียน wrapper

---

3) ความแตกต่างจาก C# / VB.NET (เชิงลึก)

3.1 Paradigm

- **C#/VB.NET:** Object-Oriented-first, functional features แทรกมาทีหลัง (LINQ, lambdas, pattern matching เบื้องต้น)
- **F#:** Functional-first, OO/imperative เป็น secondary

3.2 Null Safety

- F#: ไม่มี null ในภาษาหลัก (ต้องใช้ option<'T>), ทำให้ compiler ตรวจสอบได้ ⟶ ลด NullReferenceException
- C#: มี nullable reference types แต่ต้อง opt-in

3.3 Type System

- F#: **Structural Equality** โดย default (records, DU เทียบค่าได้ทันที)
- C#: ต้อง override Equals/GetHashCode

3.4 Syntax & Boilerplate

- F#: โค้ด functional pipeline, concise
- C#: ใช้ verbose syntax (แต่ช่วยให้มือใหม่เข้าใจโครงสร้างชัด)

□ ตัวอย่าง: **Model Payment**

```
// F#
type Payment =
  | Cash of decimal
  | Card of string * decimal

let pay p =
  match p with
  | Cash amount -> printfn "Cash %.2f" amount
  | Card (no, amount) -> printfn "Card %s paid %.2f" no amount
// C#
public abstract record Payment;
public record Cash(decimal Amount) : Payment;
public record Card(string Number, decimal Amount) : Payment;

void Pay(Payment p) => p switch
{
    Cash c => Console.WriteLine($"Cash {c.Amount}"),
    Card c => Console.WriteLine($"Card {c.Number} paid {c.Amount}"),
    _ => throw new ArgumentException()
};
```

➡□ C# 9+ มี record และ switch pattern แต่ยัง verbose กว่า F#

---

4) Functional-first, OOP และ Imperative (Deep Dive)

- **Functional-first:** Immutability, DU, pipelines, recursion
- **OOP:** class, interface, inheritance ⟶ ใช้ interop หรือ integrate กับ framework
- **Imperative:** mutable state, loops, exceptions ⟶ ใช้ได้แต่ควรเลี่ยงเป็นค่า default

เบื้องหลัง runtime model

- ทุกโค้ด F# คอมไพล์เป็น **IL (Intermediate Language)** บน CLR ⟶ ทำงานเหมือน C#
- F# types (records, DU) ⟶ คอมไพล์เป็น .NET classes/structs ⟶ ใช้จาก C#/VB.NET ได้ทันที
- Async workflows ⟶ คอมไพล์เป็น state machines บน Task/IAsyncResult

---

5) การติดตั้ง .NET SDK และ F# (เชิงลึก)

- **.NET SDK** รวม:
  - C# compiler (csc)
  - F# compiler (fsc)
  - VB.NET compiler (vbc)
  - dotnet CLI (dotnet)
- F# Interactive (fsi.exe) ⟶ REPL ที่สามารถโหลด assembly, ใช้ NuGet package แบบ interactive
- **Ionide (VS Code extension)**: เพิ่ม IntelliSense, project explorer, และ integration กับ F# Interactive

ตรวจสอบเวอร์ชัน:

```
dotnet --list-sdks
```

ใช้ fsi:

```
dotnet fsi
```

---

6) การสร้างโปรเจ็กต์ F# Console App (เชิงลึก)

```
dotnet new console -lang "F#" -o HelloFs
cd HelloFs
dotnet run
```

โครงสร้าง

```
HelloFs/
├── HelloFs.fsproj   # โปรเจ็กต์ (.NET SDK style)
└── Program.fs       # โค้ดหลัก
```

จุดสำคัญ:

- **ลำดับไฟล์ใน fsproj มีผล** ⟶ compiler จะอ่านไฟล์ตามลำดับ ⟶ ต้องจัดเรียง dependency ให้ถูก
- **Entry point:** main function หรือ top-level statements
- **Build artifacts:**
  - /bin/Debug/net8.0/HelloFs.dll ⟶ library
  - /bin/Debug/net8.0/HelloFs.exe ⟶ console app (บน Windows)

---

7) ตัวอย่างการ Debug แบบเชิงลึก

- ใช้ **dotnet run --project HelloFs** เพื่อรันโปรเจ็กต์ที่เจาะจง

- ใช้ **dotnet build -v diag** เพื่อดูรายละเอียดการ build pipeline
- ใช้ **dotnet fsi** โหลดไฟล์ทีละบรรทัดเพื่อตรวจสอบ logic

☐ **สรุปเชิงลึก**

- F# มีพื้นฐานจาก OCaml แต่ pragmatic กว่าเพราะ interop .NET ได้เต็ม
- จุดแข็งคือ **functional-first + .NET ecosystem** (ได้ทั้ง safety + practicality)
- แตกต่างจาก C#/VB.NET ที่ OO-first โดยตรง
- ติดตั้ง .NET SDK ครั้งเดียวใช้งาน F# ได้ครบทั้ง compiler, runtime, และ REPL
- การสร้างโปรเจ็กต์ dotnet new console -lang "F#" เป็นจุดเริ่มที่ดีที่สุด และต้องเข้าใจ **ไฟล์ .fsproj + order ของไฟล์**

ประวัติและจุดเด่นของ F#

☐ **ประวัติของ F#**

- **ต้นกำเนิด (2002–2005):**

  F# ถูกพัฒนาโดย **Don Syme** นักวิจัยจาก Microsoft Research Cambridge

  จุดเริ่มคือการทดลองนำแนวคิดของภาษา **OCaml/ML** มารวมกับ .NET CLR (Common Language Runtime) เพื่อสร้างภาษา functional ที่สามารถทำงานร่วมกับ ecosystem ของ .NET ได้
- **F# 1.0 (2005–2007):**

  เปิดตัวในฐานะภาษา functional-first บน .NET และมี community เริ่มทดลองใช้
- **F# 2.0 (2010):**

  ถูกบรรจุใน **Visual Studio 2010** อย่างเป็นทางการ

  จุดเด่นที่ถูกเพิ่มเข้ามา ⟶ **Asynchronous Workflows** (มาก่อน async/await ของ C# หลายปี)
- **F# 3.0 (2012):**

  เปิดตัว **Type Providers** ซึ่งเป็นนวัตกรรมสำคัญด้านข้อมูล ⟶ สามารถเชื่อมต่อกับข้อมูล (เช่น Database, JSON, XML, Web API) และสร้าง type ที่ strongly-typed โดยอัตโนมัติ
- **F# 4.x (2015–2019):**

  เพิ่มคุณสมบัติเพื่อความเข้ากันได้กับ **.NET Core**

  ปรับปรุง syntax และ performance ให้ทันสมัยขึ้น
- **F# 5.0–7.0 (2020–ปัจจุบัน):**

  ทำงานร่วมกับ .NET 5/6/7/8+

เน้น integration กับ **C# 9+ features**, performance improvements และ tooling (เช่น VS Code + Ionide, Jupyter with F#)

- **ปัจจุบัน:**
  F# เป็น **โอเพนซอร์สภายใต้ .NET Foundation**
  ถูกใช้งานจริงในวงการ **Finance, Data Science, Machine Learning, Web/API Development** และงานที่ต้องการ domain modeling ที่ปลอดภัย

☐ **จุดเด่นของ F#**

1. **Functional-first**
   - ใช้แนวทาง immutability (ค่าเปลี่ยนไม่ได้), pure functions, higher-order functions
   - ลด side-effect และ bug เชิงสถานะ (state bugs)
2. **Concise & Expressive**
   - Syntax กระชับกว่าภาษา OO-first อย่าง C#
   - ใช้ pipeline (|>) และ composition (>>) ทำให้โค้ดอ่านเหมือน flow ของข้อมูล
3. **Strong Type System + Type Inference**
   - ตรวจสอบความถูกต้องตั้งแต่ compile-time
   - ไม่ต้องประกาศ type ทุกครั้ง ⟶ compiler เดา type ได้ (เช่น Haskell, Scala)
4. **Pattern Matching & Discriminated Unions (DU)**
   - ทำให้โค้ดจัดการกับกรณีต่าง ๆ ได้ชัดเจนกว่า switch/if-else
   - Domain modeling ง่ายและปลอดภัย
5. **Type Providers (เอกลักษณ์ของ F#)**
   - ช่วยเชื่อมต่อกับ data source ภายนอกแล้วสร้าง type ให้โดยอัตโนมัติ
   - Productivity สูงมากในงาน **data science, finance, BI**
6. **Interop กับ .NET Ecosystem**
   - ใช้ได้ทุกไลบรารี NuGet, ASP.NET Core, Entity Framework, ML.NET
   - สามารถเรียกใช้โค้ด C#/VB.NET และให้ภาษาเหล่านั้นเรียกใช้ F# ได้ด้วย
7. **REPL (F# Interactive)**
   - รันโค้ดทีละบรรทัดเพื่อทดลองไอเดีย (คล้าย Python, Scala REPL)
   - ดีมากสำหรับงาน data science และการ prototyping
8. **Pragmatic Functional Programming**
   - F# ไม่ได้บังคับ strict functional แบบ Haskell
   - คุณสามารถผสม OOP และ imperative ได้เมื่อเหมาะสม

☐ ตัวอย่างโค้ดแสดงความกระชับของ F#

```
let numbers = [1..5]
let result = numbers |> List.map (fun x -> x * x) |> List.sum
printfn "Sum of squares = %d" result
```

☐ เทียบกับ C#

```
var result = Enumerable.Range(1, 5).Select(x => x * x).Sum();
Console.WriteLine($"Sum of squares = {result}");
```

☐ สรุป:

F# มีจุดแข็งตรงที่ **functional-first + practical interop กับ .NET** ทำให้เหมาะทั้งงาน **domain modeling, data-intensive applications, finance, web APIs** และยังเป็นทางเลือกที่ดีสำหรับนักพัฒนาที่ต้องการโค้ดกระชับ ปลอดภัย และ maintainable

---

ความแตกต่างจาก C# / VB.NET ในบริบทของ F#

☐ **ความแตกต่างระหว่าง F# กับ C# / VB.NET**

แม้ทั้งสามภาษา (**F#, C#, VB.NET**) จะทำงานบน **.NET runtime** และใช้ **.NET Base Class Library (BCL)** ร่วมกันได้ แต่แนวคิดและรูปแบบการออกแบบภาษาแตกต่างกันอย่างชัดเจน

1. Paradigm (แนวคิดการเขียนโปรแกรม)

- **F# ⟶ Functional-first**
  - สนับสนุน immutability, pattern matching, higher-order functions, algebraic data types (Records, Discriminated Unions)
  - ใช้โครงสร้าง **expression-based** (ทุกสิ่งคือ expression ที่คืนค่าได้)
  - ยังรองรับ OOP และ imperative code แต่ไม่ได้ถูกออกแบบมาเป็นหลัก
- **C# / VB.NET ⟶ OOP-first**
  - การเขียนโปรแกรมเชิงวัตถุ (Classes, Inheritance, Interfaces) เป็นศูนย์กลาง
  - มีการสนับสนุน functional programming บางส่วน (เช่น LINQ, lambda expressions, async/await) แต่ไม่ใช่แกนหลักของภาษา

---

2. Syntax และความกระชับ

- **F#**
  - โค้ดกระชับ อ่านง่าย คล้ายภาษา functional อื่น ๆ เช่น Haskell, OCaml
  - ใช้ **indentation** แทน {} หรือ begin/end

- มี type inference ที่ฉลาด ทำให้ไม่ต้องระบุชนิดข้อมูลทุกครั้ง

- **C# / VB.NET**
  - Syntax ค่อนข้าง verbose (โดยเฉพาะ VB.NET)
  - ต้องใช้เครื่องหมาย {} (C#) หรือ End If / End Class (VB.NET)
  - Type inference มี (var ใน C#, Dim ใน VB.NET) แต่ไม่ฉลาดเท่าของ F#

ตัวอย่างเปรียบเทียบ – การหาผลรวมของ list

☐ F#

```
let numbers = [1; 2; 3; 4; 5]
let sum = List.sum numbers
printfn "Sum = %d" sum
```

☐ C#

```
var numbers = new List<int> {1, 2, 3, 4, 5};
var sum = numbers.Sum();
Console.WriteLine($"Sum = {sum}");
```

☐ VB.NET

```
Dim numbers As New List(Of Integer) From {1, 2, 3, 4, 5}
Dim sum = numbers.Sum()
Console.WriteLine("Sum = " & sum)
```

☐ จะเห็นว่า F# ใช้โค้ดสั้นกว่าและใกล้เคียงกับการเขียนสูตรคณิตศาสตร์

---

3. Immutability vs Mutability

- **F#**
  - ค่าเริ่มต้นเป็น **immutable** (เปลี่ยนค่าไม่ได้ เว้นแต่จะประกาศ mutable)
  - ลด bug จากการแก้ค่าตัวแปรระหว่างการทำงาน
- **C# / VB.NET**
  - ค่าเริ่มต้นเป็น **mutable** (เปลี่ยนค่าได้เสมอ)
  - ต้องใช้ readonly หรือ const เพื่อบังคับ immutability

---

4. การทำงานกับ Data Structures

- **F#**
  - มี **Discriminated Unions (DU)**, **Records**, **Pattern Matching** ทำให้การจัดการข้อมูลซับซ้อนทำได้ง่ายและปลอดภัย
  - เขียนโค้ด declarative มากกว่าการเขียน logic เชิง imperative
- **C# / VB.NET**

- ใช้ class และ struct เป็นหลัก
- ไม่มี DU โดยตรง (แม้ C# 9+ จะมี record และ pattern matching แต่ยังไม่ทรงพลังเท่า F#)

---

5. Use Cases ที่นิยม

- **F#**
  - งานด้าน **data science, financial modeling, machine learning, AI, scientific computing**
  - งานที่ต้องการ **parallelism / concurrency** สูง
  - Prototyping ที่ต้องการโค้ดกระชับ
- **C# / VB.NET**
  - งาน **enterprise applications, desktop apps, ASP.NET web apps, game development (Unity)**
  - Ecosystem กว้างขวางกว่า

☐ **สรุปสั้น ๆ**

- **F#** ⟶ เน้น **Functional-first**, โค้ดสั้น, ปลอดภัยจาก bug, เหมาะกับงาน data-heavy / scientific / financial
- **C# / VB.NET** ⟶ เน้น **OOP-first**, ecosystem ใหญ่, เหมาะกับงานระบบธุรกิจและทั่วไป

---

3 โปรแกรมแนวประยุกต์ โดยแต่ละตัวอย่างจะมี

- ☐ โครงสร้างไฟล์โปรเจกต์
- ☐ โค้ดเต็มไฟล์ (ทั้ง F# และ C#/VB.NET)
- ☐ คำอธิบายโค้ด
- ☐ ผลการรัน (Output)

☐ **ตัวอย่างโปรแกรมพื้นฐาน (Basic Examples)**

---

1) Hello World

☐ **โครงสร้าง**

```
HelloWorld/
├── Program.fs      (F#)
├── Program.cs      (C#)
```

└─ Program.vb (VB.NET)

F# (Program.fs)

```
// F# Hello World
open System

[<EntryPoint>]
let main argv =
    printfn "Hello, World from F#!"
    0
```

C# (Program.cs)

```
// C# Hello World
using System;

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

VB.NET (Program.vb)

```
' VB.NET Hello World
Imports System

Module Program
    Sub Main(args As String())
        Console.WriteLine("Hello, World from VB.NET!")
    End Sub
End Module
```

อธิบาย:

- F# ใช้ printfn แทน Console.WriteLine และ entrypoint ใช้ [<EntryPoint>]
- C#/VB.NET ต้องมี Main method ที่เป็น static

ผลลัพธ์:

```
Hello, World from F#!
Hello, World from C#!
Hello, World from VB.NET!
```

2) Factorial (Recursive vs Loop)

F# (Program.fs)

```
// Factorial in F# (Recursive)
let rec factorial n =
    if n = 0 then 1
    else n * factorial (n - 1)

[<EntryPoint>]
let main argv =
    printfn "Factorial 5 = %d" (factorial 5)
    0
```

C# (Program.cs)

```
using System;

class Program {
    static int Factorial(int n) {
        if (n == 0) return 1;
        return n * Factorial(n - 1);
    }
    static void Main(string[] args) {
        Console.WriteLine("Factorial 5 = " + Factorial(5));
    }
}
```

VB.NET (Program.vb)

```
Imports System

Module Program
    Function Factorial(n As Integer) As Integer
        If n = 0 Then
            Return 1
        End If
        Return n * Factorial(n - 1)
    End Function
```

```
    Sub Main(args As String())
        Console.WriteLine("Factorial 5 = " & Factorial(5))
    End Sub
End Module
```

อธิบาย:

- F# ใช้ recursive ธรรมชาติ เพราะเป็น functional-first
- C#/VB.NET ก็ทำ recursive ได้ แต่ต้องเขียน method ชัดเจน

ผลลัพธ์:

```
Factorial 5 = 120
```

---

3) Collection: Square numbers

F# (Program.fs)

```
// F# List Processing
let numbers = [1..5]
let squares = List.map (fun x -> x * x) numbers

[<EntryPoint>]
let main argv =
    printfn "Squares: %A" squares
    0
```

C# (Program.cs)

```
using System;
using System.Linq;

class Program {
    static void Main(string[] args) {
        var numbers = Enumerable.Range(1, 5);
        var squares = numbers.Select(x => x * x);
        Console.WriteLine("Squares: " + string.Join(", ", squares));
    }
}
```

VB.NET (Program.vb)

```
Imports System
```

```
Imports System.Linq

Module Program
    Sub Main(args As String())
        Dim numbers = Enumerable.Range(1, 5)
        Dim squares = numbers.Select(Function(x) x * x)
        Console.WriteLine("Squares: " & String.Join(", ", squares))
    End Sub
End Module
```

☐ **อธิบาย:**

- F# ใช้ List.map สั้น กระชับ
- C#/VB.NET ใช้ LINQ (Select)

☐ **ผลลัพธ์:**

```
Squares: [1; 4; 9; 16; 25]
Squares: 1, 4, 9, 16, 25
Squares: 1, 4, 9, 16, 25
```

☐ **ตัวอย่างโปรแกรมแนวประยุกต์ (Applied Examples)**

---

4) Word Count (นับคำในสตริง)

☐ **F# (Program.fs)**

```
let text = "F# is functional first but supports OOP"
let words = text.Split ' '
let count = words |> Array.length

[<EntryPoint>]
let main argv =
    printfn "Text: %s" text
    printfn "Word Count = %d" count
    0
```

☐ **C# (Program.cs)**

```
using System;

class Program {
```