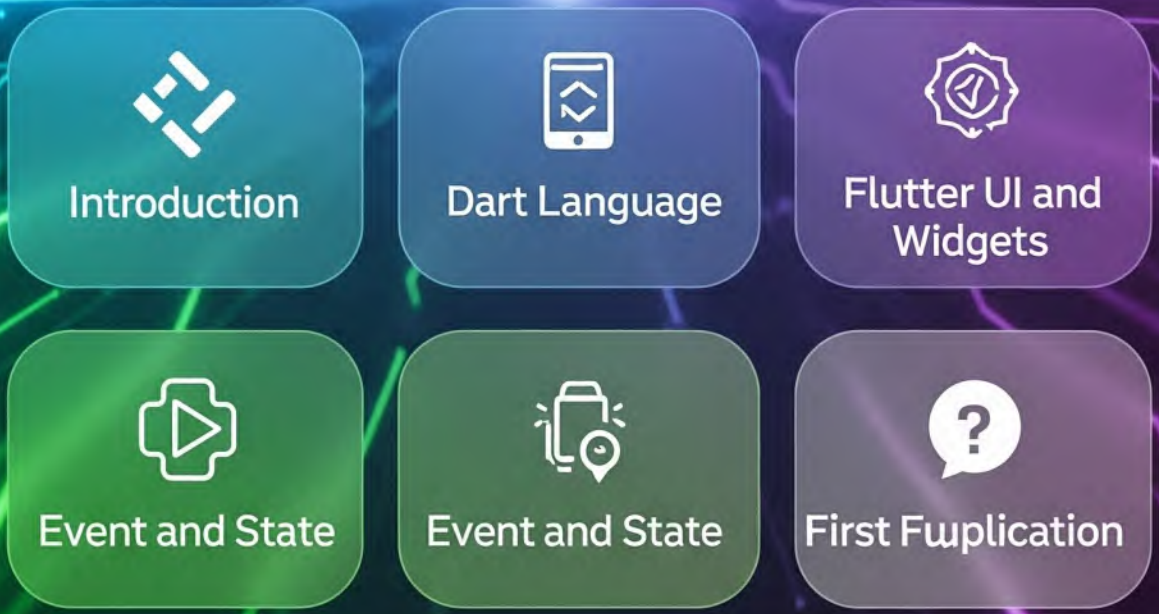
Flutter Mobile Programming: Beginner

(Integrative-Generative AI Edition)



Student Price Book Center

ai

คำนำ

ในยุคปัจจุบันที่เทคโนโลยีมือถือก้าวหน้าอย่างรวดเร็ว การพัฒนาแอปพลิเคชันที่รองรับหลากหลายแพลตฟอร์มกลายเป็นเรื่องจำเป็นสำหรับนักพัฒนาและองค์กรต่าง ๆ เพื่อให้สามารถเข้าถึงผู้ใช้งานได้มากขึ้นโดยไม่ต้องเขียนโค้ดซ้ำซ้อนหลายครั้ง Flutter ซึ่งเป็นเฟรมเวิร์กพัฒนาแอปมือถือที่ Google พัฒนาขึ้นมา จึงได้รับความสนใจอย่างกว้างขวาง ด้วยจุดเด่นในการสร้างแอปข้ามแพลตฟอร์มที่รวดเร็ว มีประสิทธิภาพ และมี UI ที่สวยงาม อีกทั้งยังสามารถเขียนโค้ดด้วยภาษา Dart ที่ออกแบบมาให้เรียนรู้ง่ายและรองรับการพัฒนาแอปที่ทันสมัย

หนังสือเล่มนี้ “Flutter Mobile Programming: Beginner” ถูกออกแบบมาเพื่อเป็นคู่มือสำหรับผู้ที่เริ่มต้นศึกษาการพัฒนาแอปด้วย Flutter โดยเฉพาะอย่างยิ่งผู้ที่ไม่มีพื้นฐานมาก่อน หรือเพิ่งเคยเขียนโปรแกรมในภาษาอื่น ๆ มาก่อน เราจะพาคุณไปรู้จักกับพื้นฐานของ Flutter ตั้งแต่การติดตั้งเครื่องมือและตั้งค่าสิ่งแวดล้อมการพัฒนา ตลอดจนการเรียนรู้ภาษา Dart ซึ่งเป็นภาษาหลักที่ใช้ในการพัฒนาแอปพลิเคชันบน Flutter

เนื้อหาของหนังสือถูกแบ่งออกเป็น 5 บทหลัก เริ่มตั้งแต่บทแรกที่แนะนำ Flutter และขั้นตอนการติดตั้งอย่างละเอียด ตั้งแต่การติดตั้ง Flutter SDK บนระบบปฏิบัติการยอดนิยมอย่าง Windows, macOS และ Linux ไปจนถึงการตั้งค่า IDE ที่เหมาะสม เช่น Android Studio และ Visual Studio Code พร้อมกับการตั้งค่า Emulator และการเชื่อมต่อกับอุปกรณ์จริง เพื่อให้ผู้เรียนสามารถรันแอปแรกได้อย่างรวดเร็วและถูกต้อง

บทที่สองจะนำเสนอพื้นฐานภาษา Dart อย่างครบถ้วน ตั้งแต่ชนิดข้อมูลและตัวแปร การควบคุมการทำงานของโปรแกรมด้วย control flow การเขียนฟังก์ชันและโครงสร้างข้อมูลเชิงวัตถุ รวมไปถึงแนวคิดสำคัญเชิงลึกอย่าง Null Safety, Exception Handling และการเขียนโปรแกรมแบบ Asynchronous ที่จำเป็นสำหรับการพัฒนาแอปพลิเคชันในโลกยุคปัจจุบัน

ในบทที่สาม เราจะเจาะลึกพื้นฐานการสร้าง UI ด้วย Flutter Widgets ซึ่งเป็นหัวใจของการพัฒนาแอปบน Flutter โดยอธิบายการใช้งาน Widgets ประเภทต่าง ๆ เช่น StatelessWidget และ StatefulWidget รวมทั้ง Layout Widgets อย่าง Column, Row, Container และ Stack พร้อมทั้งองค์ประกอบพื้นฐานอย่าง Text, Image, Icon และการตกแต่งต่าง ๆ ที่ช่วยให้แอปของคุณดูสวยงามและใช้งานได้ดี นอกจากนี้ยังอธิบายการใช้งาน MaterialApp, Scaffold, AppBar และ FloatingActionButton รวมถึงการใช้ Theme และ ColorScheme เพื่อสร้างสไตล์ที่สอดคล้องและน่าสนใจ

บทที่สี่เน้นการจัดการ Event และ State ซึ่งเป็นส่วนสำคัญที่ทำให้แอปของคุณตอบสนองต่อการโต้ตอบของผู้ใช้อย่างมีประสิทธิภาพ เราจะเรียนรู้การจัดการ event ต่าง ๆ ผ่าน onPressed, GestureDetector และการอัปเดต UI ด้วย setState รวมถึงวิธีรับข้อมูลจากผู้ใช้ผ่าน TextField, Form และ Button และวิธีการแสดงผลแจ้งเตือนด้วย SnackBar และ AlertDialog

สุดท้ายในบทที่ห้า คุณจะได้ลงมือสร้างแอปแรกของคุณเอง ด้วยการสร้างแอป Counter แบบง่าย ซึ่งจะสอนการแยกแยะโค้ด UI และ Logic อย่างเป็นระบบ พร้อมทั้งแนะนำวิธีการรันแอปบน Android Emulator และ iOS Simulator เพื่อให้มั่นใจว่าแอปของคุณทำงานได้บนแพลตฟอร์มต่าง ๆ อย่างครบถ้วน

หนังสือเล่มนี้มุ่งเน้นให้ผู้อ่านได้รับความรู้ที่ครบถ้วนและสามารถนำไปประยุกต์ใช้ได้จริง มีการอธิบายที่เข้าใจง่ายและเน้นการปฏิบัติ พร้อมด้วยตัวอย่างโค้ดและคำอธิบายที่ชัดเจน เพื่อให้การเรียนรู้ Flutter เป็นไปอย่างมีประสิทธิภาพและสนุกสนาน

หวังเป็นอย่างยิ่งว่าหนังสือ “Flutter Mobile Programming: Beginner” เล่มนี้จะเป็นเครื่องมือที่ดีสำหรับทุกท่าน ไม่ว่าจะเป็นนักเรียน นักศึกษา หรือผู้ที่สนใจเข้าสู่วงการพัฒนาแอปมือถือ และช่วยให้คุณก้าวสู่การเป็นนักพัฒนาแอป Flutter ที่มีทักษะและความมั่นใจ พร้อมสร้างสรรค์แอปพลิเคชันที่มีคุณภาพในอนาคต

ด้วยความตั้งใจและความมุ่งมั่นในการถ่ายทอดความรู้ เราขอขอบคุณผู้อ่านทุกท่านที่ไว้วางใจเลือกหนังสือเล่มนี้เป็นคู่มือการเรียนรู้ ขอให้ทุกท่านสนุกกับการเรียนรู้และประสบความสำเร็จในการพัฒนาแอปด้วย Flutter อย่างเต็มที่

ด้วยรักและปรารถนาดี

ศูนย์หนังสือราคานักเรียน

สารบัญ

หน้า

บทที่ 1

แนะนำ Flutter และการติดตั้ง

(Introduction to Flutter and Installation)

เนื้อหา

- แนะนำ Flutter และการติดตั้ง
- แนะนำ Flutter และการติดตั้งเพิ่มเติม
- Flutter คืออะไร? ความแตกต่างกับ Native และ React Native
- ประวัติและวิวัฒนาการของ Flutter
- การติดตั้ง Flutter SDK บน Windows, macOS, และ Linux แบบเชิงลึก
- การติดตั้งและตั้งค่า Android Studio กับ VS Code สำหรับ Flutter
- การตั้งค่า Emulator และเชื่อมต่อ Device จริง (Flutter)
- โครงสร้างโปรเจ็กต์ Flutter และ ขั้นตอนการรันแอปแรก

บทนำ

ในยุคที่เทคโนโลยีมือถือเติบโตอย่างรวดเร็ว การพัฒนาแอปพลิเคชันที่รองรับหลายแพลตฟอร์มกลายเป็นสิ่งจำเป็น Flutter เป็นหนึ่งในเครื่องมือที่ได้รับความนิยมอย่างสูงในวงการนี้ ด้วยความสามารถในการสร้างแอปที่ทำงานได้ทั้งบน Android และ iOS จากฐานโค้ดเดียวกัน ช่วยลดเวลาพัฒนาและค่าใช้จ่ายได้อย่างมาก

Flutter แตกต่างจาก Native App และ React Native อย่างชัดเจน โดยใช้ภาษา Dart ในการเขียนโค้ด และมีเครื่องมือพร้อมใช้งานที่ช่วยให้สามารถออกแบบ UI ได้อย่างรวดเร็วและสวยงาม อีกทั้งยังมีประสิทธิภาพสูงและการตอบสนองที่ดี ทำให้ผู้พัฒนาสามารถสร้างแอปที่มีประสบการณ์การใช้งานที่ยอดเยี่ยม

ในบทแรกนี้ เราจะเริ่มต้นด้วยการติดตั้ง Flutter SDK ซึ่งรองรับทั้ง Windows, macOS และ Linux เพื่อให้คุณสามารถพัฒนาแอปได้บนระบบปฏิบัติการที่หลากหลาย พร้อมกันนี้ยังแนะนำการติดตั้งเครื่องมือพัฒนาอย่าง Android Studio และ Visual Studio Code ซึ่งเป็น IDE ยอดนิยมสำหรับการเขียนโปรแกรม Flutter

นอกจากนี้ยังครอบคลุมวิธีการตั้งค่า Emulator เพื่อจำลองอุปกรณ์มือถือ รวมถึงการเชื่อมต่อกับอุปกรณ์จริง เพื่อทดสอบแอปพลิเคชันอย่างมีประสิทธิภาพและใกล้เคียงกับการใช้งานจริงมากที่สุด ซึ่งเป็นขั้นตอนสำคัญที่จะช่วยให้การพัฒนาของคุณราบรื่นและมีประสิทธิผล

ส่วนท้ายของบทจะพาคุณไปรู้จักกับโครงสร้างโปรเจ็กต์ Flutter อย่างละเอียด เพื่อให้เข้าใจถึงองค์ประกอบและรูปแบบการจัดการไฟล์ต่าง ๆ ในโปรเจ็กต์ อีกทั้งยังมีขั้นตอนการรันแอปพลิเคชันแรกของคุณอย่างง่ายดาย เพื่อให้คุณได้เห็นผลลัพธ์และเริ่มต้นเส้นทางการพัฒนาแอปด้วย Flutter อย่างมั่นใจ

หนังสือเล่มนี้จึงออกแบบมาเพื่อช่วยให้ผู้ที่สนใจพัฒนาแอปมือถือด้วย Flutter ไม่ว่าจะเป็นผู้เริ่มต้นหรือผู้ที่มีประสบการณ์ สามารถเรียนรู้และปฏิบัติตามได้อย่างเป็นขั้นตอน ครอบคลุมตั้งแต่การติดตั้งจนถึงการเข้าใจโครงสร้างโปรเจ็กต์ พร้อมทั้งสร้างพื้นฐานที่มั่นคงสำหรับการพัฒนาแอปขั้นสูงในบทต่อ ๆ ไป

หวังว่าบทแรกนี้จะเป็นจุดเริ่มต้นที่ดีในการเปิดประตูสู่โลกของ Flutter และช่วยให้คุณมีความรู้และความเข้าใจที่พร้อมจะพัฒนาทักษะจนกลายเป็นนักพัฒนาแอปพลิเคชันที่มีความสามารถในอนาคต.

แนะนำ Flutter และการติดตั้ง

- Flutter คืออะไร? ความแตกต่างกับ Native, React Native
- ติดตั้ง Flutter SDK บน Windows, macOS, Linux
- ติดตั้ง Android Studio หรือ VS Code
- ตั้งค่า Emulator และเชื่อมต่อ Device จริง
- โครงสร้างโปรเจ็กต์ Flutter และการรันแอปแรก

---

1) Flutter คืออะไร — ภาพรวมสั้น ๆ (แต่ครบ)

- **Flutter** เป็น UI toolkit ของ Google สำหรับสร้างแอปแบบ cross-platform (มือถือ, เว็บ, เดสก์ท็อป, embedded) จาก **โค้ดฐานเดียว (single codebase)** โดยใช้ภาษา **Dart**. Flutter ให้ชุด widget และ rendering engine (ใช้ Skia) ที่วาด UI เอง ทำให้ได้ UI ที่เหมือนกันบนทุกแพลตฟอร์มและควบคุมการเรนเดอร์ได้ละเอียด. (flutter.dev, docs.flutter.dev)
- คุณสมบัติเด่นสั้น ๆ:
    - **Hot reload** (แก้ UI/logic แล้วเห็นผลทันทีในโหมด debug) ⟶ เพิ่ม productivity.
    - **AOT compilation** สำหรับ release (ประสิทธิภาพสูงบนมือถือ) และ **JIT** ในโหมด debug (เพื่อทำ hot reload).
    - **Multi-platform targets**: Android, iOS, Web, Windows, macOS, Linux (และอื่น ๆ). (flutter.dev, docs.flutter.dev)

---

2) ความต่างระหว่าง Flutter / Native / React Native (สรุปแบบใช้งานจริง)

- **Native (Android / iOS)**

- Android: Kotlin/Java + Android SDK ⟶ ใช้ UI component ของระบบ (native widgets).
- iOS: Swift/Obj-C + UIKit/SwiftUI ⟶ native UI และเข้าถึง Platform API ได้ตรงที่สุด (ดีที่สุดเรื่องการปรับจูนเฉพาะแพลตฟอร์ม).
- ข้อดี: ประสิทธิภาพ/การเข้าถึงแอพระบบเต็มที่ ข้อเสีย: พัฒนาแยกสองโค้ดฐาน (งานซ้ำ).

- **Flutter**
  - วาด UI ด้วย Skia (ไม่ใช้ native UI controls) ⟶ ได้หน้าตา/พฤติกรรมสม่ำเสมอระหว่างแพลตฟอร์ม, ประสิทธิภาพดีเพราะไม่ต้องผ่าน JS bridge.
  - เหมาะกับงานที่ต้องการ UI/animation ซับซ้อนและต้องการโค้ดร่วมกันมาก ๆ. (docs.flutter.dev)
- **React Native**
  - ใช้ JavaScript/React; โดยปกติจะเรียกใช้ native UI components ผ่าน bridge (ปัจจุบันพัฒนาเป็น JSI / New Architecture เพื่อลด overhead).
  - ข้อดี: ถ้าทีมเชี่ยวชาญ JS/React จะ ramp-up ได้เร็ว; ข้อจำกัดเดิมคือ bridge อาจเป็นคอขวดในบางงานที่ต้องแลกข้อมูลบ่อย ๆ (react native ก็กำลังพัฒนาสถาปัตยกรรมใหม่เพื่อลดข้อจำกัดนี้). (reactnative.dev)
- **สรุปแนวคิดเลือกใช้**:
  - ต้องการ native-look + ทีม native ⟶ Native;
  - ต้องการโค้ดร่วมกับทีม JS/React ⟶ React Native;
  - ต้องการ UI ที่คอนโทรลได้สูง, animation ราบรื่น, โค้ดฐานเดียวข้ามทุกแพลตฟอร์ม ⟶ Flutter. (แต่ขึ้นกับทีมและ ecosystem ของแพ็กเกจที่ต้องใช้). (Droids On Roids, reactnative.dev)

---

3) ติดตั้ง Flutter SDK — ขั้นตอนโดยย่อตาม OS

ก่อนเริ่ม: ดาวน์โหลด SDK จากหน้าทางการ และทำตามขั้นตอนแต่ละ OS อย่างเคร่งครัด — หลังติดตั้งให้รัน flutter doctor เพื่อตรวจสภาพแวดล้อมและแก้ไข dependency ที่ขาด. (docs.flutter.dev)

Windows (สรุป)

1. ดาวน์โหลด ZIP ของ Flutter SDK (Windows) จากหน้า Flutter docs/Install.
2. แตกไฟล์ไปที่โฟลเดอร์ตัวอย่างแนะนำ เช่น C:\src\flutter (อย่าใส่ในโฟลเดอร์ที่ต้องสิทธิพิเศษ เช่น Program Files).
3. **เพิ่ม PATH**: เพิ่ม C:\src\flutter\bin เข้า Environment Variables (User PATH). (หรือใช้ setx PATH "%PATH%;C:\src\flutter\bin" ใน PowerShell/Command Prompt)
4. ติดตั้ง Git for Windows (ถ้ายังไม่มี) — Flutter ใช้ git สำหรับการอัพเดต SDK.

5. เปิด PowerShell/Command Prompt แล้วรัน:
6. flutter doctor
7. flutter doctor --android-licenses
   และแก้ไขตามคำแนะนำของ flutter doctor. (docs.flutter.dev)

macOS (สรุป)

1. ดาวน์โหลด .zip ของ Flutter สำหรับ macOS แล้วแตกไปที่ ~/development/flutter (หรือที่ที่ต้องการ).
2. เปิดเทอร์มินัลแล้วเพิ่ม PATH (ตัวอย่างถ้าใช้ zsh):
3. export PATH="$PATH:$HOME/development/flutter/bin"
   ใส่บรรทัดนี้ใน ~/.zshrc หรือ ~/.bash_profile แล้ว source ~/.zshrc.
4. ถ้าจะ build สำหรับ iOS: ติดตั้ง **Xcode** จาก App Store, รัน sudo xcode-select --switch /Applications/Xcode.app/Contents/Developer และยอมรับ license (sudo xcodebuild -runFirstLaunch).
5. รัน flutter doctor และทำตามคำแนะนำ. (docs.flutter.dev)

Linux (สรุป — ตัวอย่าง Debian/Ubuntu)

1. ดาวน์โหลด .tar.xz ของ Flutter แล้วแตก (เช่น ~/development/flutter).
2. เพิ่ม PATH เช่น:
3. export PATH="$PATH:$HOME/development/flutter/bin"
4. ติดตั้ง dependency พื้นฐาน (ขึ้นกับ distro) — ตัวอย่าง (Ubuntu/Debian) อาจต้องติดตั้ง git, curl, unzip, xz-utils, libglu1-mesa ฯลฯ — ตรวจสอบหน้าติดตั้ง Linux เพื่อคำสั่งที่แม่นยำของ distro คุณ.
5. รัน flutter doctor และแก้ไขตามคำแนะนำ (เช่นติดตั้ง Android SDK, รับ android-licenses ฯลฯ). (docs.flutter.dev)

---

4) ติดตั้ง IDE: Android Studio หรือ VS Code (อธิบายเลือก + ขั้นตอนสั้น ๆ)

- **Android Studio** — เหมาะถ้าคุณต้องการ integrated tools (AVD Manager, SDK Manager, device manager, profiler) ในที่เดียว. ให้ติดตั้งแล้วใน Android Studio -> Plugins ค้นหาและติดตั้ง **Flutter** (จะติดตั้ง Dart plugin ให้อัตโนมัติ). หลังติดตั้ง ให้ตั้งค่า SDK (SDK Manager) และ AVD (Device Manager). (docs.flutter.dev, Android Developers)
- **VS Code** — เบาและเร็ว เหมาะถ้าชอบ editor แบบเบา ๆ ติดตั้ง **Flutter** extension (ซึ่งจะติดตั้ง Dart extension ด้วย). ใน VS Code จะมี command palette สำหรับ Flutter: New Project, Flutter: Run Flutter Doctor และปุ่ม hot-reload. แต่คุณยังต้องติดตั้ง Android SDK/command-line tools (มาจาก Android Studio หรือ SDK tools) เพื่อรัน emulator / build. (docs.flutter.dev)

- **แนะนำการเลือก**: ถ้าคุณเริ่มต้นและต้องการทุกอย่างครบในตัว ⟶ Android Studio; ถ้าคุณชอบ workflow ที่เร็วและ configure เองได้ ⟶ VS Code.

---

5) ตั้งค่า Emulator และเชื่อมต่อ Device จริง

สร้าง Android emulator (AVD)

1. เปิด Android Studio ⟶ Tools > Device Manager (หรือ AVD Manager).
2. สร้าง Virtual Device เลือก System Image (แนะนำใช้ Google APIs หรือ Android 11/12/13/ ล่าสุดที่ต้องการ).
3. เปิด emulator แล้วมันจะปรากฏใน flutter devices หรือใน IDE. (ถ้ามีปัญหาเรื่อง virtualization ให้เปิด HAXM/Hyper-V/Windows Hypervisor หรือตั้งค่า Android Emulator ให้ใช้ ARM images บน Mac M1). (Android Developers, docs.flutter.dev)

เชื่อมต่อ Android real device

1. เปิด **Developer options** บนมือถือ ⟶ เปิด **USB debugging** (หรือ Wireless debugging ถาต้องการ).
2. (Windows) ติดตั้ง OEM USB driver ถ้าจำเป็น.
3. เสียบสาย USB แล้วอนุญาตการเข้าถึงบนมือถือ.
4. รัน flutter devices ⟶ อุปกรณ์จะปรากฏเป็น device id.
5. รัน flutter run -d <deviceId> เพื่อรันแอป. (docs.flutter.dev, Android Developers)

iOS device / Simulator (เฉพาะ macOS)

- Simulator: ใช้ Xcode ⟶ Open Developer Tool > Simulator.
- อุปกรณ์จริง: ต้องมี Xcode ติดตั้ง, ลงทะเบียน device ใน Xcode และจัดการ provisioning/profile (สำหรับการทดสอบบน device บางกรณีต้องสมัคร Apple Developer ถ้าจะ deploy และใช้ capabilities บางอย่าง). (docs.flutter.dev)

---

6) โครงสร้างโปรเจ็กต์ Flutter (ภาพรวมไฟล์/โฟลเดอร์สำคัญ)

เมื่อรัน flutter create my_app จะได้โครงสร้างพื้นฐาน เช่น:

```
my_app/
├── android/          # Android native project (Gradle)
├── ios/              # iOS native project (Xcode)
├── lib/              # โค้ด Dart ของแอป (main.dart เป็นต้น)
│   └── main.dart
├── test/             # unit/widget tests
├── pubspec.yaml      # รายการ dependencies, assets, metadata ของโปรเจกต์
```

```
├── .gitignore
├── README.md
```

- lib/ เป็นที่ตั้งของโค้ดหลัก — โดยทั่วไป lib/main.dart จะมี void main() => runApp(MyApp());. pubspec.yaml ควบคุม dependencies, assets, fonts, และต้อง flutter pub get เมื่อเพิ่ม package. (docs.flutter.dev)

---

7) สร้างและรันแอปแรก — คำสั่งตัวอย่าง

สร้างโปรเจ็กต์ใหม่:

```
flutter create my_app
cd my_app
```

รันบน device/emulator:

```
# ดู device ที่ต่ออยู่
flutter devices

# รันบน default device (IDE จะเลือกให้)
flutter run

# หรือรันบน emulator/physical device เฉพาะ
flutter run -d emulator-5554
flutter run -d <your-device-id>

# สร้าง APK (release)
flutter build apk --release
```

Hot reload / Hot restart:

- ใน terminal ขณะ flutter run ⟶ กด r = hot reload (เฉพาะ debug mode), R = hot restart.
- ใน IDE (Android Studio / VS Code) จะมีปุ่ม ⚡ สำหรับ hot reload. (docs.flutter.dev)

---

8) ตัวอย่างไฟล์ lib/main.dart (แอปตัวอย่างแบบเต็มไฟล์ — Counter แบบสั้น)

```
import 'package:flutter/material.dart';

void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
  const MyApp({super.key});
```

```
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'First Flutter App',
      theme: ThemeData(primarySwatch: Colors.blue),
      home: const HomePage(),
    );
  }
}

class HomePage extends StatefulWidget {
  const HomePage({super.key});
  @override
  State<HomePage> createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  int _counter = 0;

  void _increment() => setState(() => _counter++);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Counter Example')),
      body: Center(
        child: Text('You pressed the button $_counter times',
            style: const TextStyle(fontSize: 20)),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: _increment,
        child: const Icon(Icons.add),
      ),
    );
```

```
  }
}
```

- บันทึกไฟล์นี้ที่ lib/main.dart แล้วรัน flutter run — คุณจะเห็นแอปและสามารถใช้ hot reload ขณะแก้โค้ดได้ทันที. (docs.flutter.dev)

---

9) คำสั่งตรวจเช็ค / แก้ปัญหาเบื้องต้น

- flutter doctor -v ⟶ ตรวจเช็คสถานะทั้งหมด (SDK, IDE, device, licenses). หากมีช่องเหลือง/แดง ทำตามคำแนะนำที่ flutter doctor บอก. (docs.flutter.dev)
- หากเจอปัญหา Android SDK / licenses: flutter doctor --android-licenses แล้วตอบ y ทุกข้อ. (Codecademy)

---

10) สรุปแหล่งอ้างอิงสำคัญ (อ่านต่อ / official)

- หน้าเริ่มต้น Flutter — ภาพรวมและเป้าหมาย. (flutter.dev)
- สถาปัตยกรรม Flutter (architectural overview). (docs.flutter.dev)
- ติดตั้งบน Windows / macOS / Linux (หน้า Install แบบ OS-specific). (docs.flutter.dev)
- ตั้งค่า Android Studio / AVD (Device Manager). (docs.flutter.dev, Android Developers)
- ตั้งค่า VS Code (Flutter extension / workflow). (docs.flutter.dev)
- คำสั่ง CLI (flutter create, flutter run, flutter doctor). (docs.flutter.dev)

แนะนำ Flutter และการติดตั้งเพิ่มเติม

แนะนำ Flutter และการติดตั้ง แบบ **รายละเอียดเชิงลึกจริง ๆ**
ผมจะแจกแจงทั้ง **แนวคิด, สถาปัตยกรรม, ขั้นตอนการติดตั้งแต่ละ OS, การตั้งค่า IDE, การเชื่อมต่ออุปกรณ์, โครงสร้างโปรเจ็กต์, การรันแอปแรก** พร้อมเทคนิคตรวจสอบและแก้ปัญหาที่พบบ่อย

---

1. Flutter คืออะไร? (ภาพใหญ่ + ลึก)

Flutter เป็น **UI SDK (Software Development Kit)** ที่ Google พัฒนาเพื่อสร้างแอป **multi-platform** จาก **single codebase** โดยใช้ภาษา **Dart**
จุดเด่นเชิงเทคนิค:

- **Rendering Engine (Skia)** — Flutter ไม่ใช้ native UI components แต่เรนเดอร์ทุก pixel เอง ทำให้ UI มีหน้าตาและพฤติกรรมเหมือนกันทุกแพลตฟอร์ม
- **Widget System** — ทุกอย่างเป็น widget (Text, Button, Layout, App เองก็เป็น widget) ⟶ composable & customizable

- **Reactive UI Update** — เมื่อ state เปลี่ยน ⟶ UI re-render เฉพาะส่วนที่เปลี่ยน (คล้าย React)
- **Hot Reload** — ใช้ JIT (Just-In-Time) compiler ในโหมดพัฒนา ทำให้แก้โค้ดแล้วเห็นผลในไม่กี่มิลลิวินาที
- **AOT Compilation** — ใช้ Ahead-of-Time compiler สำหรับ release ⟶ ประสิทธิภาพสูง
- **Platform Channels** — กลไกเชื่อม Flutter กับ native code (Java/Kotlin, Swift/Obj-C) เพื่อเรียก API ของระบบ

---

2. เปรียบเทียบ Flutter vs Native vs React Native (เชิงสถาปัตยกรรม)

คุณสมบัติ	Flutter	Native (Android/iOS)	React Native
UI Rendering	Skia engine (วาด pixel เอง)	Native controls	Native controls
Performance	ใกล้เคียง native (ไม่มี JS bridge)	สูงสุด	ขึ้นกับ JS bridge (ใหม่ใช้ JSI)
Language	Dart	Kotlin/Java (Android), Swift/Obj-C (iOS)	JavaScript / TypeScript
Code Sharing	สูง (ทุก platform ใช้โค้ดเดียวกัน)	ต่ำ	สูง
Ecosystem Plugin	ครอบคลุมมาก (pub.dev)	ครบ (native SDK)	ครอบคลุมมาก (npm)
Animation	ลื่นมาก (control ได้ pixel-level)	ลื่น (native APIs)	อาจมี delay ถ้า bridge หนัก

ข้อสรุปสั้น

- เน้น UI/Animation ซับซ้อน ⟶ Flutter
- เน้น integration ลึกกับระบบ ⟶ Native
- เน้นแชร์โค้ดกับ Web/React ⟶ React Native

---

3. การติดตั้ง Flutter SDK (เชิงเทคนิค)

Windows

1. ดาวน์โหลด Flutter SDK .zip จาก flutter.dev
2. แตกไฟล์ไปที่ C:\src\flutter (หลีกเลี่ยงโฟลเดอร์ที่ต้องใช้สิทธิ์ admin เช่น Program Files)
3. ตั้งค่า PATH:

- กด Win + S ⟶ Search **Environment Variables**
- เพิ่ม C:\src\flutter\bin ใน User PATH

4. ติดตั้ง Git for Windows (จำเป็นต่อการ update SDK)
5. ติดตั้ง Android Studio (สำหรับ Android SDK, Emulator)
6. เปิด cmd ⟶ รัน
7. flutter doctor
8. flutter doctor --android-licenses

macOS

1. ดาวน์โหลด .zip แล้วแตกไปที่ ~/development/flutter
2. ตั้ง PATH ใน ~/.zshrc หรือ ~/.bash_profile
3. export PATH="$PATH:$HOME/development/flutter/bin"
4. ติดตั้ง Xcode (จำเป็นสำหรับ iOS)
5. sudo xcode-select --switch /Applications/Xcode.app/Contents/Developer
6. sudo xcodebuild -runFirstLaunch
7. รัน flutter doctor และแก้ไขตามคำแนะนำ

Linux (Ubuntu/Debian ตัวอย่าง)

1. ดาวน์โหลด .tar.xz และแตกไปที่ ~/development/flutter
2. เพิ่ม PATH เหมือน macOS
3. ติดตั้ง dependencies
4. sudo apt-get install git curl unzip xz-utils zip libglu1-mesa
5. รัน flutter doctor

---

4. ติดตั้ง IDE

Android Studio

- ติดตั้ง Plugin: Flutter + Dart
- ใช้ **Device Manager** สร้าง emulator
- มี Android SDK Manager ในตัว

VS Code

- ติดตั้ง Extensions: Flutter + Dart
- เบา เร็ว แต่ต้องติดตั้ง Android SDK/Emulator แยก

---

5. ตั้งค่า Emulator และ Device จริง

Emulator

- เปิด Android Studio ⟶ Tools ⟶ Device Manager ⟶ Create Virtual Device

- เลือก System Image (API 31+, Google Play/Google APIs)
- กด Run ⟶ emulator จะปรากฏใน flutter devices

Device จริง

- เปิด Developer Options + USB Debugging บนมือถือ
- เสียบสาย USB ⟶ อนุญาตการเชื่อมต่อ
- ตรวจสอบ
- flutter devices

---

6. โครงสร้างโปรเจ็กต์ Flutter

เมื่อรัน flutter create my_app:

```
my_app/
├── android/       # Native Android code (Gradle)
├── ios/           # Native iOS code (Xcode)
├── lib/           # Dart source code
│   └── main.dart
├── test/          # Unit tests
└── pubspec.yaml   # Metadata + dependencies
```

ไฟล์สำคัญ:

- **lib/main.dart** ⟶ จุดเริ่มของแอป
- **pubspec.yaml** ⟶ รายการ dependencies, assets
- **android/ios/** ⟶ native code (ปรับแต่งได้เมื่อจำเป็น)

---

7. สร้างและรันแอปแรก

```
flutter create my_app
cd my_app
flutter run
```

โค้ดตัวอย่าง main.dart

```
import 'package:flutter/material.dart';

void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
  const MyApp({super.key});
```

```
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'My First Flutter App',
      home: Scaffold(
        appBar: AppBar(title: const Text('Hello Flutter')),
        body: const Center(child: Text('สวัสดี Flutter!')),
      ),
    );
  }
}
```

8. เทคนิคตรวจสอบและแก้ปัญหา

- flutter doctor -v ⟶ ตรวจสอบทุกอย่าง
- ถ้าเจอ PATH ไม่ถูก ⟶ ตรวจ PATH ด้วย echo $PATH หรือ echo %PATH%
- ถ้า Emulator ไม่โผล่ ⟶ ตรวจสอบ HAXM/Hyper-V (Windows) หรือ ARM image (Mac M1)
- ถ้า Android SDK ไม่เจอ ⟶ ติดตั้งผ่าน Android Studio > SDK Manager

Flutter คืออะไร? ความแตกต่างกับ Native และ React Native

โดยอธิบายทั้งในมุม **สถาปัตยกรรม**, **เทคนิคการทำงาน**, และ **การเลือกใช้งานจริง**

1. Flutter คืออะไร?

Flutter คือ **UI SDK (Software Development Kit)** ที่ Google พัฒนา สำหรับสร้างแอป **cross-platform** จาก **โค้ดเดียว (Single Codebase)** เพื่อรันบน:

- **Mobile** ⟶ Android, iOS
- **Web** ⟶ Flutter Web
- **Desktop** ⟶ Windows, macOS, Linux
- **Embedded** ⟶ เช่น อุปกรณ์ IoT

ภาษาโปรแกรมหลักคือ **Dart**

- ออกแบบมาให้ **compile ได้ทั้ง JIT (Just-In-Time)** และ **AOT (Ahead-of-Time)**
- JIT ⟶ ใช้ตอนพัฒนา (hot reload)
- AOT ⟶ ใช้ตอน release (ความเร็วใกล้ native)

2. จุดเด่นเชิงเทคนิคของ Flutter

- **Own Rendering Engine (Skia)** ⟶ Flutter ไม่พึ่งพา UI ของระบบปฏิบัติการ แต่เรนเดอร์ทุก pixel เอง ทำให้ UI เหมือนกันทุกแพลตฟอร์ม
- **Widget System** ⟶ ทุกอย่างคือ widget (Button, Text, Layout, Page)
- **Reactive UI** ⟶ เมื่อ state เปลี่ยน UI จะ re-render เฉพาะส่วนที่จำเป็น
- **Hot Reload** ⟶ แก้โค้ดแล้วเห็นผลทันทีในไม่กี่มิลลิวินาที
- **Platform Channels** ⟶ ติดต่อกับ native code (Kotlin/Java/Swift/Obj-C) เพื่อใช้ API ของระบบ

3. ความแตกต่าง Flutter vs Native vs React Native

คุณสมบัติ	Flutter	Native (Android/iOS)	React Native
ภาษา	Dart	Kotlin/Java (Android), Swift/Obj-C (iOS)	JavaScript / TypeScript
UI Rendering	ใช้ Skia engine วาด pixel เอง	ใช้ native UI controls ของระบบ	ใช้ native UI controls ผ่าน JS bridge
Performance	ใกล้ native (ไม่มี JS bridge)	สูงสุด	ดี แต่ขึ้นกับประสิทธิภาพ JS bridge
Hot Reload	มี (รวดเร็วมาก)	มีแต่ช้ากว่า (Android Studio / Xcode)	มี (Fast Refresh)
Code Sharing	สูง (1 codebase ครอบคลุม mobile, web, desktop)	ต่ำ (ต้องเขียนแยก)	สูง (Android/iOS ใช้โค้ดร่วมกันได้)
Plugin Ecosystem	pub.dev ครอบคลุมมาก	ครบสมบูรณ์ (native SDK)	npm ครอบคลุมมาก
การเข้าถึง Hardware	ผ่าน Platform Channel	ตรงสุด (native APIs)	ผ่าน Native Module
UI Consistency	เหมือนกันทุกแพลตฟอร์ม	ตามระบบ	ตามระบบ

4. การเลือกใช้

- **เลือก Flutter ถ้า**
  - ต้องการ UI/Animation ลื่นและสวยเหมือนกันทุกแพลตฟอร์ม
  - ต้องการรันหลายแพลตฟอร์มจากโค้ดเดียว

    - เน้น productivity (hot reload, ecosystem widget สำเร็จรูป)
- **เลือก Native ถ้า**
    - ต้องการ performance สูงสุดและเข้าถึง hardware ทุกอย่างทันที
    - แอปต้องใช้ฟีเจอร์เฉพาะของระบบมาก ๆ
    - ทีมมีพื้นฐาน native อยู่แล้ว
- **เลือก React Native ถ้า**
    - ทีมถนัด JavaScript/React
    - ต้องการ reuse code ระหว่างเว็บกับแอป
    - แอป UI ไม่ซับซ้อนมาก และยอมรับข้อจำกัด bridge ได้

ประวัติและวิวัฒนาการของ Flutter

- **จุดเริ่มต้น (ราว 2015–2017):**
  Flutter เริ่มต้นในชื่อโค้ดว่า *Sky* ถูกเผยตัวใน Dart Summit ปี 2015 โดยมีเป้าหมายให้รันที่ 120 fps บน Android ตามด้วยการปล่อยเวอร์ชัน alpha (v0.0.6) ในเดือนพฤษภาคม 2017 (วิกิพีเดีย)
- **เวอร์ชัน 1.0 (ธันวาคม 2018):**
  เปิดตัว Flutter 1.0 อย่างเป็นทางการที่ Flutter Live ‘18 (วิกิพีเดีย)
- **Flutter 2.0 (มีนาคม 2021):**
  เปิดตัวในงาน Flutter Engage พ่วงด้วยเว็บรองรับแบบ stable, renderer แบบ Canvas, และ null safety ใน Dart (วิกิพีเดีย)
- **Flutter 3.0 (พฤษภาคม 2022):**
  รองรับเดสก์ท็อปแบบ stable ทั้ง Windows, macOS, Linux รวมถึงการรองรับ null safety เต็มรูปแบบ (วิกิพีเดีย)

สถิติการใช้งาน & แนวโน้ม

ผู้ใช้และโครงการ

- นักพัฒนาใช้งาน Flutter ประมาณ **2 ล้านคน** และเติบโตอย่างต่อเนื่อง (~10% ต่อเดือนหลัง มี.ค. 2024) (goodfirms.co)
- มี **500,000 แอป Flutter บน Google Play** ในปี 2023 และเติบโต ~50% ต่อปี (goodfirms.co)
- GitHub repository ของ Flutter มี **ประมาณ 145,000 stars** และใน pub.dev มี **กว่า 30,000 packages** (ตอนปี 2023) (goodfirms.co, myteams.co)

ความนิยม:

- Flutter เป็น **framework ที่นักพัฒนารักที่สุดอันดับที่ 3** (มีนักพัฒนาถึง 68.8% อยากใช้ต่อ) (goodfirms.co)
- ในเชิง cross-platform, Flutter แซงหน้า React Native แล้ว และมีสัดส่วนกว่า **46%** (เทียบกับ React Native ~38%) (DOIT)

สัดส่วนใน iOS ใหม่:

- ตาม Reddit อ้างถึงข้อมูลจาก Apptopia – ปี 2024 Flutter อยู่ใน **~30% ของแอป iOS ใหม่** (Reddit)

การเติบโตในระดับองค์กร:

- มีการเพิ่มขึ้นของนักพัฒนา **1 ล้านคน** ใน 4 ปี (ScaleupAlly)
- **68% ของนักพัฒนา** ใช้ Flutter บนอุปกรณ์หลายแพลตฟอร์ม (web, desktop, embedded) (ScaleupAlly)
- ธนาคารและภาคการเงินใช้ Flutter เพิ่มขึ้น **217%** ตั้งแต่ปี 2021 (ScaleupAlly)
- ภูมิภาค Asia-Pacific ตอนนี้มีนักพัฒนา Flutter ถึง **41%** และ India เป็นประเทศที่มีผู้ใช้สูงที่สุด (ScaleupAlly)
- Flutter มี **ความพึงพอใจสูงถึง 92%** (ScaleupAlly)

องค์กรและแบรนด์ที่ใช้ Flutter

- **Google** — ใช้ในแอปเช่น Google Pay และ Google Earth (วิกิพีเดีย)
- **Alibaba** — แอป Xianyu รองรับผู้ใช้ไม่ต่ำกว่า 50 ล้านคน (goodfirms.co, invotech.co)
- **Tencent / WeChat** — แอปที่มีผู้ใช้มากกว่า 1 พันล้านคน รองรับ UX ลื่นจาก Flutter (shivlab.com, codigee.com, sourcebae.com)
- **BMW** — ใช้ Flutter ใน My BMW app และระบบ infotainment ในรถยนต์ (shivlab.com, Reddit)
- **eBay Motors** — แอปเฉพาะของ eBay ที่สร้างด้วย Flutter มีผู้ดาวน์โหลดล้านครั้ง (shivlab.com, Reddit)

อื่น ๆ (จาก Slack/Reddit):

“Zerodha, Cred, ClickUp, Google Pay, Google Earth”

“Nubank” – ธนาคารดิจิทัลที่ใหญ่ที่สุดในบราซิล

“Grab, Costco International (เว้น US/CA), YouTube Create” (Reddit)

“Toyota infotainment”, “ByteDance มีทีม Flutter กว่า 800 คน”, “Supernova.io” (Reddit)

- รายชื่อเพิ่มเติมจาก 6sense: **Charoen Pokphand Foods PCL** (ไทย), **IBM**, **Purdue University**, **University of Maryland**, **Avianca** ฯลฯ (รวมกว่า 15,000 บริษัทที่เริ่มใช้ Flutter ปี 2025) (6sense)

แนวโน้มอนาคต (2025+)

- **แผนของ Google** — Flutter ถูกวางไว้เป็นแกนหลักของกลยุทธ์ ambient computing และ consumer products ของ Google ภายในปี 2026 (ScaleupAlly)
- **ทิศทางเทคโนโลยี** — กำลังถูกพัฒนาในแนวทาง WebAssembly, AI-assisted development, server-driven UI (ScaleupAlly)

สรุปอย่างรวดเร็ว:

1. **ยุคเริ่ม**: ก่อตั้งราว 2015, เวอร์ชัน 1.0 เปิด 2018, ขยายไปถึง Web & Desktop ใน 2021–2022
2. **ผู้ใช้ & การเติบโต**: 2 ล้าน dev, 500k แอป Play Store, community ecosystem แข็งแรง, เป็นที่นิยมระดับสูง
3. **องค์กรใช้จริง**: Alibaba, Tencent, BMW, eBay, Google, Nubank ฯลฯ — ครอบคลุมหลายอุตสาหกรรม
4. **อนาคตสดใส**: ขยายสู่ AI, webassembly, การพัฒนา UI แบบ server-driven พร้อมกลยุทธ์ของ Google รองรับแพลตฟอร์มทุกชนิด

การติดตั้ง Flutter SDK บน Windows, macOS, และ Linux แบบเชิงลึก

---

1. ภาพรวมการติดตั้ง Flutter SDK

Flutter SDK คือชุดเครื่องมือหลักในการพัฒนาแอป Flutter ซึ่งรวมถึง:

- **Dart SDK** (ภาษาที่ใช้เขียน Flutter)
- **Flutter Framework**
- **เครื่องมือ Command Line** (flutter CLI)
- **Engine** สำหรับ Render UI และเชื่อมต่อกับ Native API

การติดตั้งจะแบ่งออกเป็น:

1. ดาวน์โหลดและแตกไฟล์ SDK
2. ตั้งค่า PATH
3. ตรวจสอบการติดตั้งด้วย flutter doctor
4. ติดตั้งเครื่องมือเสริม เช่น Android SDK, Xcode, หรือ Linux tools

---

2. ติดตั้งบน Windows

2.1 ความต้องการระบบ

- Windows 10 หรือ 11 (64-bit)
- พื้นที่ว่างอย่างน้อย 2.5 GB (ไม่รวม Android Studio)
- Git (สำหรับดาวน์โหลดแพ็กเกจและ plugin)

2.2 ขั้นตอนติดตั้ง

1. **ดาวน์โหลด Flutter SDK**
   - ไปที่ https://flutter.dev/docs/get-started/install/windows
   - ดาวน์โหลดไฟล์ .zip (เช่น flutter_windows_3.x.x-stable.zip)
2. **แตกไฟล์**
   - แตกไปยังโฟลเดอร์ที่ไม่มีช่องว่างในชื่อ เช่น C:\src\flutter
   - ไม่ควรติดตั้งใน C:\Program Files\ เพราะต้องใช้สิทธิ์ Admin
3. **ตั้งค่า PATH**
   - เปิด *Start Menu* ⟶ พิมพ์ Environment Variables
   - แก้ไข Path และเพิ่ม:
   - C:\src\flutter\bin
   - รีสตาร์ท Command Prompt หรือ PowerShell
4. **ตรวจสอบการติดตั้ง**
5. flutter doctor
   - คำสั่งนี้จะแสดงสิ่งที่ติดตั้งแล้ว และสิ่งที่ต้องติดตั้งเพิ่ม

---

3. ติดตั้งบน macOS

3.1 ความต้องการระบบ

- macOS Monterey หรือใหม่กว่า
- พื้นที่ว่าง 2.8 GB+
- Git
- สำหรับ iOS: Xcode 14+ (ต้องใช้ macOS)

3.2 ขั้นตอนติดตั้ง

1. **ดาวน์โหลด Flutter SDK**
2. cd ~/development
3. git clone https://github.com/flutter/flutter.git -b stable
4. **ตั้งค่า PATH**
   - แก้ไขไฟล์ ~/.zshrc (หรือ ~/.bashrc ถ้าใช้ bash)
   - export PATH="$PATH:$HOME/development/flutter/bin"
   - รัน:
   - source ~/.zshrc

5. **ตรวจสอบการติดตั้ง**
6. flutter doctor
7. **ติดตั้ง Xcode (สำหรับ iOS)**
8. sudo xcode-select --switch /Applications/Xcode.app/Contents/Developer
9. sudo xcodebuild -runFirstLaunch
    - เปิด Xcode ⟶ Preferences ⟶ Components ⟶ ดาวน์โหลด iOS Simulator

---

4. ติดตั้งบน Linux

4.1 ความต้องการระบบ

- Ubuntu 20.04+ หรือ Linux distro อื่นที่รองรับ
- Git
- Pkg: curl, unzip, xz-utils

4.2 ขั้นตอนติดตั้ง

1. **ติดตั้ง Dependencies**
2. sudo apt-get update
3. sudo apt-get install git curl unzip xz-utils libglu1-mesa
4. **ดาวน์โหลด Flutter SDK**
5. cd ~/development
6. git clone https://github.com/flutter/flutter.git -b stable
7. **ตั้งค่า PATH**
8. echo 'export PATH="$PATH:$HOME/development/flutter/bin"' >> ~/.bashrc
9. source ~/.bashrc
10. **ตรวจสอบการติดตั้ง**
11. flutter doctor

---

5. เคล็ดลับจากผู้เชี่ยวชาญ

- ใช้ flutter upgrade เพื่ออัปเดต SDK เป็นเวอร์ชันล่าสุด
- ใช้ flutter channel เพื่อสลับระหว่าง **stable**, **beta**, **dev**
- ติดตั้ง SDK ไว้ในโฟลเดอร์ ~/development หรือ C:\src เพื่อลดปัญหาสิทธิ์ไฟล์
- สำรองโฟลเดอร์ .pub-cache เพื่อย่นเวลาติดตั้งแพ็กเกจในเครื่องใหม่

การติดตั้งและตั้งค่า Android Studio กับ VS Code สำหรับ Flutter

---

1. ติดตั้ง Android Studio พร้อมตั้งค่าสำหรับ Flutter

1.1 ดาวน์โหลดและติดตั้ง

- เข้าเว็บ https://developer.android.com/studio
- ดาวน์โหลด Android Studio เวอร์ชันล่าสุด (รองรับ Windows/macOS/Linux)
- ติดตั้งตามขั้นตอน (Next ⟶ Next)
- แนะนำให้เลือกติดตั้ง Android SDK, Android SDK Platform-Tools, และ Android SDK Build-Tools ด้วย

1.2 ติดตั้ง Flutter & Dart Plugin

- เปิด Android Studio
- ไปที่ File > Settings > Plugins (บน macOS คือ Android Studio > Preferences > Plugins)
- ค้นหา **Flutter** ⟶ ติดตั้ง plugin นี้ (ระบบจะติดตั้ง Dart Plugin ให้ด้วยอัตโนมัติ)
- รีสตาร์ท Android Studio

1.3 ตั้งค่า Android SDK

- เปิด File > Settings > Appearance & Behavior > System Settings > Android SDK
- เลือก SDK Platforms ⟶ ติ๊กเลือก Android API Level ที่ต้องการ (แนะนำ API 31 หรือสูงกว่า)
- SDK Tools ⟶ ให้แน่ใจว่า **Android SDK Build-Tools**, **Android Emulator**, และ **Android SDK Platform-Tools** ถูกติดตั้ง
- Apply แล้วรอให้ดาวน์โหลดเสร็จ

1.4 สร้างและใช้งาน Emulator

- ไปที่ Tools > Device Manager (หรือ AVD Manager)
- กด “Create Virtual Device”
- เลือกรูปแบบอุปกรณ์ (เช่น Pixel 5)
- เลือกระบบปฏิบัติการ (System Image) ที่ต้องการ (แนะนำ Google APIs, Android 12/13)
- ตั้งชื่อ emulator ⟶ Finish
- กดปุ่ม Run เพื่อเปิด Emulator

1.5 ตรวจสอบว่า Android Studio พร้อมกับ Flutter

- เปิด Terminal (ใน Android Studio หรือแยก)
- รันคำสั่ง
- flutter doctor
- ตรวจสอบสถานะทุกหัวข้อเป็น ✓ (รวมถึง Android toolchain และ connected device)

2. ติดตั้งและตั้งค่า Visual Studio Code (VS Code) สำหรับ Flutter

2.1 ดาวน์โหลดและติดตั้ง VS Code

- ไปที่ https://code.visualstudio.com/
- ดาวน์โหลดเวอร์ชันล่าสุดสำหรับ Windows/macOS/Linux
- ติดตั้งตามขั้นตอน

2.2 ติดตั้ง Extension สำหรับ Flutter และ Dart

- เปิด VS Code
- ไปที่ Extensions (Ctrl+Shift+X หรือคลิกไอคอน Extensions)
- ค้นหา **Flutter** ⟶ ติดตั้ง (ระบบจะติดตั้ง Dart ให้ด้วยอัตโนมัติ)
- รีสตาร์ท VS Code

2.3 ตั้งค่า Android SDK และ Emulator แยก

- ต้องติดตั้ง Android SDK แยก (โดยปกติติดตั้งผ่าน Android Studio หรือดาวน์โหลด command line tools)
- ตั้งค่า PATH ให้ adb และ emulator สามารถเรียกใช้งานได้จาก terminal (เช่นใน .bashrc หรือ Environment Variables)
- สร้างและรัน emulator ผ่าน command line หรือ Android Studio Device Manager
    - ตัวอย่างคำสั่งสร้าง emulator:
    - avdmanager create avd -n my_avd -k "system-images;android-31;google_apis;x86_64"
    - รัน emulator:
    - emulator -avd my_avd

2.4 ทดสอบการเชื่อมต่อกับ Flutter

- เปิด Terminal ใน VS Code หรือแยกออกมา
- รัน
- flutter doctor
- ตรวจสอบ device ที่เชื่อมต่อ (emulator หรือ device จริง)
- รันคำสั่ง
- flutter devices
- หากขึ้นรายชื่อ emulator หรือ device แสดงว่าเชื่อมต่อเรียบร้อย

---

3. เคล็ดลับสำคัญในการใช้งาน IDE กับ Flutter

ประเด็น	คำแนะนำ
Hot Reload	ใช้ได้ทั้ง Android Studio และ VS Code (กดปุ่ม r หรือ Shift + R)

ประเด็น	คำแนะนำ
Build & Run	ใช้ปุ่ม Run ใน IDE หรือคำสั่ง flutter run ใน terminal
Debugging	ใช้ Debugger ใน IDE ตั้ง breakpoint และ inspect variables
Auto Complete	ติดตั้ง Flutter/Dart plugin เพื่อช่วยเรื่องโค้ดสมบูรณ์
Emulator Performance	เปิด Hardware Acceleration (HAXM ใน Windows, Hypervisor ใน macOS)
Flutter DevTools	ใช้ DevTools ใน browser เพื่อตรวจสอบ Performance และ Widget Tree

การตั้งค่า Emulator และเชื่อมต่อ Device จริง (Flutter)

1. ตั้งค่า Emulator (Android Virtual Device - AVD)

1.1 สร้าง Emulator ผ่าน Android Studio

1. เปิด Android Studio ⟶ ไปที่ **Tools > Device Manager** (หรือ AVD Manager)
2. กด **Create Virtual Device**
3. เลือก Device Template ที่ต้องการ (เช่น Pixel 5) ⟶ กด Next
4. เลือก **System Image** ที่ต้องการ (แนะนำ Google APIs, API Level 31 หรือใหม่กว่า)
5. กด **Download** (ถ้ายังไม่มี System Image) ⟶ รอโหลดเสร็จ
6. ตั้งชื่อ AVD แล้วกด Finish

1.2 รัน Emulator

- ใน Device Manager กดปุ่ม Run (รูป Play) เพื่อเปิด Emulator
- รอ Emulator โหลดเสร็จ จะเห็นหน้าจอ Android

1.3 เชื่อมต่อกับ Flutter

- เปิด Terminal แล้วรัน
- flutter devices
- ถ้าแสดงชื่อ emulator และสถานะ “device” ⟶ พร้อมใช้งาน

2. ตั้งค่า Emulator แบบ Command Line (สำหรับ VS Code / Linux)

2.1 สร้าง Emulator

avdmanager create avd -n my_avd -k "system-images;android-31;google_apis;x86_64"

- -n my_avd คือชื่อ emulator
- -k คือ path ของ system image ที่ต้องการ (ใช้คำสั่ง sdkmanager --list ดูชื่อได้)

2.2 รัน Emulator

emulator -avd my_avd

- Emulator จะเปิดขึ้นมาเหมือนเดิม

---

3. เชื่อมต่อ Device จริง (Android)

3.1 เปิด Developer Options และ USB Debugging

- เข้า **Settings > About Phone**
- กด **Build Number** 7 ครั้ง (จะขึ้นว่า “You are now a developer!”)
- กลับไปที่ **Settings > Developer Options**
- เปิด **USB Debugging**

3.2 เชื่อมต่อกับคอมพิวเตอร์

- เสียบสาย USB จากมือถือไป PC
- ถ้ามี popup “Allow USB Debugging?” ให้เลือก “Allow”

3.3 ตรวจสอบ device

- เปิด Terminal
- flutter devices
- ควรเห็นชื่อ device ขึ้นมา เช่น
- 1234567890 device model_name
- หากไม่เจอ ให้ลองรัน
- adb devices
  เพื่อเช็คว่า ADB เจอ device หรือไม่

3.4 ปัญหาที่พบบ่อยและแก้ไข

ปัญหา	วิธีแก้ไข
Device ไม่แสดงใน flutter devices	ติดตั้ง driver มือถือ (Windows) หรือยืนยันอนุญาต USB Debugging
Emulator ช้า/ค้าง	เปิด Hardware Acceleration (HAXM บน Windows, Hypervisor บน Mac)
ADB ไม่ทำงาน	รีสตาร์ท adb ด้วยคำสั่ง adb kill-server และ adb start-server
อุปกรณ์ iOS ไม่ขึ้น	ต้องใช้ Mac และเปิด Developer Mode, เชื่อมต่อผ่าน Xcode

---

4. เชื่อมต่อ Device จริง (iOS)

4.1 เตรียมอุปกรณ์ iOS

- ใช้ Mac เท่านั้น

- เปิด **Settings > Developer** ⟶ เปิด Developer Mode
- เสียบ iPhone ผ่านสาย USB หรือเชื่อมต่อผ่าน Network (Wi-Fi debugging)
- ยืนยันเชื่อมต่อบน iPhone (“Trust This Computer?”)

4.2 ตรวจสอบ device ด้วย Flutter

- รัน
- flutter devices
- จะแสดงชื่อ iPhone หากเชื่อมต่อสำเร็จ

---

5. รันแอปบน Emulator หรือ Device จริง

- รันคำสั่ง
- flutter run
- เลือก device ที่ต้องการ (ถ้ามีหลาย device)
- รอ build และแอปจะรันบน emulator หรือ device จริง

โครงสร้างโปรเจ็กต์ Flutter และ ขั้นตอนการรันแอปแรก

โครงสร้างโปรเจ็กต์ Flutter

เวลารันคำสั่ง

flutter create my_app

Flutter จะสร้างโปรเจ็กต์โครงสร้างพื้นฐานดังนี้:

```
my_app/
├── android/          # โค้ด native ของ Android (Java/Kotlin, Gradle build system)
├── ios/              # โค้ด native ของ iOS (Swift/Obj-C, Xcode project)
├── lib/              # โค้ด Dart หลักของแอป
│   └── main.dart     # จุดเริ่มต้นของแอป
├── test/             # โค้ด unit test
├── build/            # โฟลเดอร์ที่ Flutter สร้างขึ้นอัตโนมัติ (ไม่ต้องแก้ไข)
├── .dart_tool/       # โฟลเดอร์สำหรับเครื่องมือ Dart (auto generated)
├── .idea/            # การตั้งค่า IDE (Android Studio)
├── .vscode/          # การตั้งค่า VS Code
├── pubspec.yaml      # ไฟล์ config รายละเอียดโปรเจ็กต์, dependencies, assets
├── README.md         # คำอธิบายโปรเจ็กต์ (ทั่วไป)
├── analysis_options.yaml # กฎ linting และการตรวจสอบโค้ด Dart
```

└── .gitignore # ไฟล์บอก Git ว่าจะไม่ติดตามไฟล์ไหน

คำอธิบายโฟลเดอร์และไฟล์หลัก

1. android/

- โค้ด native และไฟล์ตั้งค่าของ Android เช่น AndroidManifest.xml, build.gradle
- สามารถแก้ไขสำหรับ custom native integration หรือแก้ไข permission

2. ios/

- โค้ด native iOS, ไฟล์ Xcode project เช่น Info.plist
- แก้ไขสำหรับการเพิ่ม native module หรือสิทธิ์

3. lib/

- โฟลเดอร์หลักสำหรับเขียน Dart code ของแอป
- **main.dart** คือจุดเริ่มต้น (entry point) ของแอป Flutter

4. test/

- โค้ดทดสอบ (unit test) ของแอป

5. pubspec.yaml

- กำหนด metadata เช่น ชื่อแอป, author
- กำหนด dependencies เช่น Flutter SDK, package เสริม
- กำหนด assets (รูป, fonts) ที่ต้องการใช้ในแอป

การรันแอป Flutter ครั้งแรก

ขั้นตอนรันแอปง่าย ๆ

1. สร้างโปรเจ็กต์ใหม่ (ถ้ายังไม่มี)
2. flutter create my_app
3. cd my_app
4. เปิด editor (VS Code หรือ Android Studio) เพื่อดูและแก้ไขไฟล์ lib/main.dart
5. เชื่อมต่อ device หรือ emulator พร้อมตรวจสอบว่าเจอ device
6. flutter devices
7. รันแอป
8. flutter run

ตัวอย่างโค้ดใน lib/main.dart

```
import 'package:flutter/material.dart';
```

```
void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'My First Flutter App',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Hello Flutter'),
        ),
        body: const Center(
          child: Text('สวัสดี Flutter!'),
        ),
      ),
    );
  }
}
```

อธิบายโค้ด

- main() คือจุดเริ่มต้น เรียก runApp() เพื่อเริ่มแอป
- MyApp เป็น StatelessWidget ที่แสดง MaterialApp
- MaterialApp กำหนดชื่อแอป และหน้าหลักเป็น Scaffold (มี AppBar และ Body)
- Body มี Center widget ที่จัดกึ่งกลางข้อความ "สวัสดี Flutter!"

ผลการรัน

แอปจะเปิดหน้าจอแอปที่มีแถบ AppBar ด้านบนชื่อ "Hello Flutter"
และมีข้อความ "สวัสดี Flutter!" กึ่งกลางหน้าจอ

ตัวอย่างโปรแกรม Flutter 3 ตัวอย่างแบบเต็มไฟล์ พร้อมโครงสร้างโฟลเดอร์, คำอธิบายโค้ด

ตัวอย่างโปรแกรม 1: แอปแสดงข้อความง่าย ๆ (Hello World)

โครงสร้างโปรเจ็กต์

```
hello_world/
├── lib/
│   └── main.dart
├── pubspec.yaml
└── (อื่น ๆ ตามมาตรฐาน Flutter)
```

ไฟล์ lib/main.dart

```
import 'package:flutter/material.dart';

void main() {
  runApp(const HelloWorldApp());
}

class HelloWorldApp extends StatelessWidget {
  const HelloWorldApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Hello World Demo',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Hello World'),
        ),
        body: const Center(
          child: Text(
            'Hello, Flutter!',
            style: TextStyle(fontSize: 24),
          ),
        ),
      ),
    );
  }
```

```
}
```

คำอธิบายโค้ด

- main() คือจุดเริ่มต้นของโปรแกรม
- runApp() เรียก Widget หลัก คือ HelloWorldApp
- HelloWorldApp สร้าง MaterialApp มีชื่อเรื่อง และหน้าหลักคือ Scaffold
- Scaffold ประกอบด้วย AppBar และ Body
- Body คือ Center widget ที่จัดกึ่งกลางข้อความ 'Hello, Flutter!'
- ข้อความถูกตั้งขนาด font เป็น 24

ผลการรัน

- หน้าจอแอปมีแถบบนชื่อว่า "Hello World"
- ตรงกลางหน้าจอมีข้อความ "Hello, Flutter!" ตัวใหญ่ชัดเจน

ตัวอย่างโปรแกรม 2: แอปนับจำนวนกดปุ่ม (Counter App)

โครงสร้างโปรเจ็กต์

```
counter_app/
├── lib/
│   └── main.dart
├── pubspec.yaml
└── (อื่น ๆ ตามมาตรฐาน Flutter)
```

ไฟล์ lib/main.dart

```
import 'package:flutter/material.dart';

void main() {
  runApp(const CounterApp());
}

class CounterApp extends StatefulWidget {
  const CounterApp({super.key});

  @override
  State<CounterApp> createState() => _CounterAppState();
}

class _CounterAppState extends State<CounterApp> {
```

```
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Counter Demo',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Counter App'),
        ),
        body: Center(
          child: Text(
            'Count: $_counter',
            style: const TextStyle(fontSize: 32),
          ),
        ),
        floatingActionButton: FloatingActionButton(
          onPressed: _incrementCounter,
          tooltip: 'Increment',
          child: const Icon(Icons.add),
        ),
      ),
    );
  }
}
```

คำอธิบายโค้ด

- ใช้ StatefulWidget เพื่อเก็บ state ตัวแปร _counter
- ฟังก์ชัน _incrementCounter เพิ่มค่าตัวแปรและเรียก setState() เพื่อ update UI

- build() สร้าง UI ที่แสดงจำนวนกดปุ่ม และปุ่ม FloatingActionButton
- เมื่อกดปุ่มจำนวนจะเพิ่มขึ้นและข้อความอัพเดต

ผลการรัน

- หน้าจอแอปมีแถบชื่อ "Counter App"
- ตรงกลางแสดงจำนวนเริ่มต้น 0
- มีปุ่ม + ที่มุมล่างขวา กดแล้วจำนวนเพิ่มขึ้นทันที

ตัวอย่างโปรแกรม 3: แอปแสดงรายการแบบ ListView พร้อมการกดเลือก

โครงสร้างโปรเจ็กต์

```
listview_app/
├── lib/
│   └── main.dart
├── pubspec.yaml
└── (อื่น ๆ ตามมาตรฐาน Flutter)
```

ไฟล์ lib/main.dart

```
import 'package:flutter/material.dart';

void main() {
  runApp(const ListViewApp());
}

class ListViewApp extends StatelessWidget {
  const ListViewApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'ListView Demo',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Simple ListView'),
        ),
        body: const ItemList(),
      ),
```

```
    );
  }
}

class ItemList extends StatefulWidget {
  const ItemList({super.key});

  @override
  State<ItemList> createState() => _ItemListState();
}

class _ItemListState extends State<ItemList> {
  final List<String> items = List.generate(20, (index) => 'Item ${index + 1}');
  String selectedItem = '';

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        if (selectedItem.isNotEmpty)
          Padding(
            padding: const EdgeInsets.all(16.0),
            child: Text(
              'Selected: $selectedItem',
              style: const TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
            ),
          ),
        Expanded(
          child: ListView.builder(
            itemCount: items.length,
            itemBuilder: (context, index) {
              return ListTile(
                title: Text(items[index]),
                onTap: () {
```

```
                setState(() {
                  selectedItem = items[index];
                });
              },
            );
          },
        ),
      ),
    ],
  );
 }
}
```

คำอธิบายโค้ด

- สร้าง List ของ String 20 รายการ
- ใช้ ListView.builder เพื่อสร้างรายการแบบ dynamic
- เมื่อแตะ ListTile จะเปลี่ยนค่าตัวแปร selectedItem และแสดงชื่อรายการด้านบน
- ใช้ Column เพื่อวางข้อความเลือกกับ ListView ในแนวตั้ง

ผลการรัน

- หน้าจอมีแถบชื่อ "Simple ListView"
- แสดงรายการ Item 1 ถึง Item 20 เป็น list scroll ได้
- แตะรายการใด รายการนั้นจะขึ้นแสดงคำว่า “Selected: Item X” ด้านบน

สรุป

บทนำนี้สรุปว่า Flutter เป็นเครื่องมือพัฒนาแอปมือถือข้ามแพลตฟอร์มที่โดดเด่นด้วยประสิทธิภาพและความรวดเร็วในการสร้าง UI โดยใช้ภาษา Dart บทแรกจะพาผู้อ่านติดตั้ง Flutter SDK บนระบบปฏิบัติการต่าง ๆ พร้อมแนะนำการติดตั้ง Android Studio หรือ VS Code เพื่อใช้เป็น IDE ตั้งค่า Emulator และเชื่อมต่อกับอุปกรณ์จริง รวมถึงทำความเข้าใจโครงสร้างโปรเจ็กต์และการรันแอปแรก เพื่อเตรียมพื้นฐานที่มั่นคงสำหรับการพัฒนาแอปด้วย Flutter ต่อไป.

บทที่ 2
พื้นฐานภาษา Dart
(Basic of Dart Language)

เนื้อหา

- พื้นฐานภาษา Dart
- พื้นฐานภาษา Dart (เชิงลึก)
- ตัวแปรและชนิดข้อมูลใน Dart
- Null Safety ใน Dart
- การใช้ List, Set, Map ใน Dart
- Control Flow ใน Dart
- ฟังก์ชัน (Functions) ใน Dart
- การเขียน Class และ Constructor ใน Dart
- Inheritance, Mixins, Abstract class, Interface ใน Dart
- Exception Handling (try-catch-finally) ใน Dart
- Asynchronous Programming ใน Dart (Future, async/await, Stream)

บทนำ

ภาษา Dart เป็นหัวใจหลักของการพัฒนาแอปด้วย Flutter ซึ่งทำหน้าที่เป็นภาษาหลักในการเขียนโค้ดทั้งหมดของแอปพลิเคชัน Dart ถูกออกแบบมาให้มีความเรียบง่ายและทรงพลัง เพื่อรองรับการสร้างแอปที่รวดเร็ว มีประสิทธิภาพ และสามารถทำงานได้หลากหลายแพลตฟอร์ม ด้วยการเรียนรู้พื้นฐานของภาษา Dart อย่างถูกต้อง จะช่วยให้ผู้พัฒนาสามารถเขียนโค้ดได้อย่างมั่นใจและมีประสิทธิผล

ในบทนี้ เราจะเริ่มต้นทำความรู้จักกับตัวแปรและชนิดข้อมูลพื้นฐาน เช่น String, int, double, bool และ dynamic เพื่อเข้าใจวิธีการจัดเก็บและจัดการข้อมูลในโปรแกรมอย่างถูกต้องและมีประสิทธิภาพ นอกจากนี้ยังเน้นเรื่อง Null Safety ซึ่งเป็นคุณสมบัติสำคัญที่ช่วยลดข้อผิดพลาดที่เกิดจากการใช้ค่าที่ว่างเปล่าในภาษา Dart

การใช้งานคอลเลกชันพื้นฐาน เช่น List, Set และ Map เป็นสิ่งจำเป็นสำหรับการจัดการข้อมูลกลุ่มในแอปพลิเคชัน บทนี้จะอธิบายถึงลักษณะและวิธีการใช้งานของแต่ละประเภทอย่างละเอียด เพื่อให้ผู้พัฒนาสามารถเลือกใช้งานได้อย่างเหมาะสมตามบริบทของโปรแกรม

Control flow คือการควบคุมการทำงานของโปรแกรมผ่านคำสั่ง if, switch และ loop ซึ่งช่วยให้โปรแกรมตอบสนองต่อเงื่อนไขต่าง ๆ ได้อย่างยืดหยุ่นและชาญฉลาด บทนี้จะให้ตัวอย่างการใช้งานและแนวทางที่ถูกต้องเพื่อให้เข้าใจพื้นฐานของการไหลของโปรแกรม

ฟังก์ชันใน Dart มีบทบาทสำคัญในการแบ่งแยกและจัดการโค้ดให้เป็นระเบียบ รวมถึงการใช้ optional parameters และ named parameters ที่ช่วยให้การเรียกใช้งานฟังก์ชันมีความยืดหยุ่นและอ่านง่ายมากขึ้น ผู้เรียนจะได้เรียนรู้วิธีการสร้างและใช้งานฟังก์ชันอย่างครบถ้วน

ในส่วนของการเขียน Class และ Constructor จะอธิบายวิธีการสร้างโครงสร้างข้อมูลที่มีความซับซ้อนและเป็นระเบียบมากขึ้น ซึ่งเป็นพื้นฐานของการเขียนโปรแกรมเชิงวัตถุ (OOP) พร้อมกับการใช้งาน inheritance, mixins, abstract class และ interface ที่ช่วยเพิ่มความสามารถและความยืดหยุ่นในการออกแบบระบบซอฟต์แวร์

การจัดการข้อผิดพลาดหรือ exception handling ด้วย try-catch-finally เป็นเทคนิคสำคัญที่ช่วยให้โปรแกรมสามารถตอบสนองต่อสถานการณ์ที่ไม่คาดคิดได้อย่างเหมาะสม บทนี้จะสอนวิธีการจับข้อผิดพลาดและจัดการอย่างมืออาชีพ เพื่อให้แอปพลิเคชันมีความเสถียรและน่าเชื่อถือ

ในยุคของการเชื่อมต่ออินเทอร์เน็ตและแอปพลิเคชันที่ต้องตอบสนองเร็ว การเขียนโปรแกรมแบบ asynchronous จึงเป็นเรื่องจำเป็น Dart มีเครื่องมือสำหรับจัดการ asynchronous programming เช่น Future, async/await และ Stream ซึ่งจะช่วยให้การทำงานกับงานที่ใช้เวลานานหรือรอข้อมูลจากภายนอกเป็นไปอย่างราบรื่นและมีประสิทธิภาพ

การเข้าใจพื้นฐานภาษา Dart อย่างถ่องแท้จะเป็นรากฐานสำคัญสำหรับการพัฒนาแอปด้วย Flutter ที่มีความซับซ้อนมากขึ้นในบทต่อไป ผู้เรียนจะได้รับทักษะและความรู้ที่จำเป็นเพื่อก้าวไปสู่การพัฒนาแอปที่ครบวงจรและมีคุณภาพสูง

บทนี้จึงเป็นก้าวแรกที่มั่นคงและจำเป็นสำหรับผู้ที่ต้องการเป็นนักพัฒนา Flutter มืออาชีพ โดยครอบคลุมเนื้อหาพื้นฐานอย่างครบถ้วนและชัดเจน เพื่อให้คุณพร้อมที่จะสร้างสรรค์แอปพลิเคชันที่ตอบโจทย์ความต้องการได้อย่างสมบูรณ์และมีประสิทธิภาพสูงสุด.

พื้นฐานภาษา Dart

- ตัวแปรและชนิดข้อมูล (String, int, double, bool, dynamic)
- Null Safety
- การใช้ List, Set, Map
- Control flow (if, switch, loop)
- ฟังก์ชัน, optional parameters, named parameters
- การเขียน Class และ Constructor
- Inheritance, Mixins, Abstract class, Interface
- Exception handling (try-catch-finally)
- Asynchronous programming (Future, async/await, Stream)

1. ตัวแปรและชนิดข้อมูล (String, int, double, bool, dynamic)

```
void main() {
  String name = "Flutter";         // ข้อความ
  int age = 5;                     // จำนวนเต็ม
  double height = 170.5;           // จำนวนทศนิยม
  bool isCool = true;              // ค่าความจริง
  dynamic anything = 10;           // ชนิดข้อมูลยืดหยุ่น
  anything = "Now a string";       // เปลี่ยนชนิดข้อมูลได้
}
```

- dynamic คือชนิดตัวแปรที่สามารถเก็บข้อมูลได้ทุกชนิด (แต่ควรใช้อย่างระมัดระวัง)

---

2. Null Safety

- Dart ป้องกันไม่ให้ตัวแปรที่ไม่ควรมีค่าเป็น null มีค่าเป็น null
- ใช้ ? เพื่ออนุญาตให้ตัวแปรเป็น null ได้

```
String? nullableString;    // อนุญาต null ได้
String nonNullableString = "Hello";  // ต้องไม่เป็น null

void main() {
  nullableString = null;
  print(nullableString?.length);  // ถ้า null จะไม่เกิด error แต่คืนค่า null
}
```

---

3. การใช้ List, Set, Map

```
void main() {
  // List (ลำดับที่เก็บค่าได้ซ้ำ)
  List<String> fruits = ['Apple', 'Banana', 'Orange'];

  // Set (เก็บค่าไม่ซ้ำ และไม่มีลำดับ)
  Set<int> numbers = {1, 2, 3, 3};  // จะเก็บเป็น {1,2,3}

  // Map (key-value)
  Map<String, int> scores = {
    'Alice': 90,
```

```
    'Bob': 85,
  };

  print(fruits[1]);       // Banana
  print(numbers.contains(2));  // true
  print(scores['Alice']);  // 90
}
```

---

4. Control flow (if, switch, loop)

```
void main() {
  int number = 3;

  if (number > 0) {
    print("Positive");
  } else {
    print("Non-positive");
  }

  switch (number) {
    case 1:
      print("One");
      break;
    case 2:
      print("Two");
      break;
    default:
      print("Other");
  }

  // for loop
  for (int i = 0; i < 3; i++) {
    print(i);
  }
```

```
  // while loop
  int j = 0;
  while (j < 3) {
    print(j);
    j++;
  }
}
```

---

5. ฟังก์ชัน, optional parameters, named parameters

```
// ฟังก์ชันปกติ
int add(int a, int b) {
  return a + b;
}

// optional positional parameters
String greet(String name, [String? title]) {
  return title != null ? 'Hello $title $name' : 'Hello $name';
}

// named parameters
void printInfo({required String name, int? age}) {
  print('Name: $name, Age: ${age ?? 'unknown'}');
}

void main() {
  print(add(3, 4));                // 7
  print(greet("John"));            // Hello John
  print(greet("John", "Mr."));     // Hello Mr. John
  printInfo(name: "Alice", age: 30);
  printInfo(name: "Bob");
}
```

---

6. การเขียน Class และ Constructor

```
class Person {
```