Dart
Programming Language:
(Integrative-Generative AI Edition)

- Introduction to Dart and Installation
- Dart Program Structure
- Variables and Data Types
- Operators
- Control Flow
- Basic Functions
- Bibliography

Student Price Book Center

คำนำ

ในยุคที่เทคโนโลยีด้านการพัฒนาแอปพลิเคชันก้าวหน้าอย่างรวดเร็ว นักพัฒนาไม่เพียงแต่ต้องตามให้ทันเท่านั้น แต่ยังต้องเลือกเครื่องมือและภาษาโปรแกรมที่เหมาะสมต่อการสร้างสรรค์ผลงานอย่างมีประสิทธิภาพ หนึ่งในภาษาที่ได้รับความสนใจอย่างกว้างขวางในช่วงไม่กี่ปีมานี้คือ **Dart** ซึ่งถูกออกแบบและพัฒนาโดย Google เพื่อรองรับการสร้างซอฟต์แวร์ที่มีประสิทธิภาพสูง ทั้งในฝั่งเว็บ แอปมือถือ และเดสก์ท็อป โดยเฉพาะเมื่อทำงานร่วมกับเฟรมเวิร์ก **Flutter** ทำให้ Dart กลายเป็นหัวใจหลักในการพัฒนาแอปพลิเคชันข้ามแพลตฟอร์ม (Cross-Platform) ที่ทั้งสวยงาม รวดเร็ว และมีความยืดหยุ่น

หนังสือ **Dart Programming Language: Beginner** เล่มนี้ถูกเขียนขึ้นเพื่อเป็นคู่มือสำหรับผู้เริ่มต้นที่ต้องการปูพื้นฐานอย่างมั่นคง พร้อมขยายความเข้าใจไปสู่ระดับที่สามารถประยุกต์ใช้ Dart ในการพัฒนาโปรแกรมได้จริง โครงสร้างเนื้อหาถูกออกแบบให้ค่อย ๆ พาผู้อ่านจากพื้นฐานที่สำคัญที่สุดไล่เรียงไปจนถึงการสร้างโค้ดที่ซับซ้อนและเป็นระบบมากขึ้น โดยแต่ละบทประกอบด้วยคำอธิบายอย่างละเอียด ตัวอย่างโค้ดที่ใช้งานได้จริง และแนวคิดเชิงลึกเพื่อให้ผู้อ่านเข้าใจกลไกการทำงานของภาษาอย่างแท้จริง

บทที่ 1 จะพาผู้อ่านทำความรู้จักกับ Dart อย่างครบถ้วน ทั้งประวัติความเป็นมา เหตุผลที่ Dart เหมาะกับการใช้ร่วมกับ Flutter วิธีการติดตั้งทั้งแบบ Standalone และการติดตั้งผ่าน Flutter SDK รวมถึงการใช้ **DartPad** เพื่อทดลองเขียนโค้ดออนไลน์ และการตั้งค่าสภาพแวดล้อมการพัฒนา (IDE) อย่าง Visual Studio Code หรือ IntelliJ เพื่อเริ่มต้นรันโปรแกรมแรกของคุณ

บทที่ 2 อธิบายโครงสร้างของโปรแกรม Dart ตั้งแต่การเขียนคอมเมนต์ การใช้ฟังก์ชัน print() และการทำงานของฟังก์ชัน main() ในเชิงลึก พร้อมตัวอย่างโปรแกรมที่ช่วยให้ผู้อ่านเข้าใจโครงสร้างของโค้ดได้อย่างชัดเจน

บทที่ 3 จะลงรายละเอียดเรื่อง **ตัวแปรและชนิดข้อมูล** ซึ่งเป็นหัวใจสำคัญของการเขียนโปรแกรม Dart โดยอธิบายการประกาศตัวแปรแบบ var, final, และ const การใช้ชนิดข้อมูลพื้นฐาน (int, double, String, bool) รวมถึงการแปลงชนิดข้อมูล (Type Conversion) ผ่านตัวอย่างจริงที่สามารถนำไปประยุกต์ใช้ในงานพัฒนาได้

บทที่ 4 พาผู้อ่านเข้าสู่โลกของ **ตัวดำเนินการ (Operators)** ทั้งการคำนวณทางคณิตศาสตร์ การกำหนดค่า การเปรียบเทียบ การใช้ตรรกะ รวมถึงตัวดำเนินการตรวจสอบชนิดข้อมูล ซึ่งจะช่วยให้คุณสามารถเขียนเงื่อนไขและการคำนวณได้อย่างแม่นยำ

บทที่ 5 เน้นเรื่อง **Control Flow** หรือการควบคุมการไหลของโปรแกรม ตั้งแต่คำสั่ง if-else, switch-case, ลูป for, while, do-while ไปจนถึงการใช้ break เพื่อควบคุมการทำงานของโปรแกรม โครงสร้างเหล่านี้เป็นสิ่งที่ขาดไม่ได้ในการเขียนโปรแกรมที่ตอบสนองตามเงื่อนไขหรือทำงานซ้ำ

บทที่ 6 กล่าวถึง **ฟังก์ชันพื้นฐาน (Functions)** ทั้งการประกาศและเรียกใช้ การใช้พารามิเตอร์แบบปกติและแบบกำหนดค่าเริ่มต้น (Default Parameter Values) การส่งค่ากลับจากฟังก์ชัน รวมถึงตัวอย่างการประยุกต์ใช้ในโปรแกรมจริง เพื่อสร้างโค้ดที่เป็นโมดูลาร์และดูแลรักษาง่าย

จุดเด่นของหนังสือเล่มนี้อยู่ที่การนำเสนอเนื้อหาอย่างเป็นขั้นตอน พร้อมทั้งมี **ตัวอย่างโค้ดและโปรแกรมประยุกต์** ในทุกบท ซึ่งผู้อ่านสามารถคัดลอกไปทดลองได้ทันที ไม่ว่าจะผ่าน DartPad หรือสภาพแวดล้อมการพัฒนาที่ติดตั้งไว้แล้ว นอกจากนี้ยังมี **ตัวอย่างบูรณาการ** ที่รวมหลายแนวคิดเข้าด้วยกัน เพื่อให้ผู้อ่านเห็นภาพการใช้งาน Dart ในสถานการณ์จริง

ผู้เขียนหวังว่าหนังสือเล่มนี้จะช่วยให้ผู้อ่านมีพื้นฐาน Dart ที่แข็งแรง เข้าใจทั้งแนวคิดและการใช้งานจริง พร้อมที่จะก้าวสู่การพัฒนาแอปพลิเคชันขั้นสูง ไม่ว่าจะเป็นการสร้างแอปด้วย Flutter การทำงานกับ API หรือการพัฒนาเว็บแอปพลิเคชันด้วย Dart ในอนาคต

ท้ายที่สุดนี้ การเรียนรู้ภาษาโปรแกรมไม่ใช่เพียงการจดจำคำสั่ง แต่คือการฝึกคิดอย่างเป็นระบบและการแก้ปัญหาอย่างสร้างสรรค์ ผู้เขียนจึงอยากชวนให้ผู้อ่านเปิดใจ ทดลอง ปรับแก้ และสร้างผลงานของตัวเองระหว่างอ่านหนังสือเล่มนี้ เพราะนั่นคือเส้นทางสู่การเป็นนักพัฒนาที่มั่นใจในความสามารถของตนเอง

ด้วยรักและปรารถนาดี

ศูนย์หนังสือราคานักเรียน

สารบัญ

หน้า

บทที่ 1

แนะนำ Dart และการติดตั้ง

(Introduction to Dart and Installation)

เนื้อหา

- แนะนำ Dart และการติดตั้ง
- เจาะลึก Dart: รายละเอียดเชิงลึก
- Dart คืออะไร (ฉบับเจาะลึก)
- ทำไม Dart ถึงใช้กับ Flutter (ฉบับเจาะลึก)
- วิธีที่ 1: การติดตั้ง Dart พร้อมกับ Flutter SDK (แนะนำ)
- การติดตั้ง Dart SDK โดยละเอียด
- DartPad (Online Playground) คืออะไร?
- การตั้งค่า Integrated Development Environment (IDE) อย่าง VS Code หรือ IntelliJ
- การรันโปรแกรม Dart (ฉบับเจาะลึก)
- ตัวอย่างโปรแกรมภาษา Dart

บทนำ

Dart เป็นภาษาการเขียนโปรแกรมที่ถูกออกแบบมาโดย Google เพื่อรองรับการพัฒนาแอปพลิเคชันที่ทำงานได้ทั้งบนเว็บ เซิร์ฟเวอร์ และโดยเฉพาะบนแพลตฟอร์มมือถือผ่าน Flutter ความโดดเด่นของ Dart คือความสามารถในการคอมไพล์ได้ทั้งแบบ Just-In-Time (JIT) และ Ahead-Of-Time (AOT) ทำให้สามารถพัฒนาได้อย่างรวดเร็วในระหว่างการทดสอบ และทำงานได้อย่างมีประสิทธิภาพเมื่อปล่อยใช้งานจริง อีกทั้งโครงสร้างของภาษายังเข้าใจง่าย มีความใกล้เคียงกับภาษา C-style ซึ่งทำให้นักพัฒนาที่เคยใช้ภาษาอื่นมาก่อนสามารถปรับตัวได้รวดเร็ว

เหตุผลสำคัญที่ Dart ถูกเลือกใช้กับ Flutter คือการผสานการทำงานที่ลงตัวกับสถาปัตยกรรมของ Flutter ซึ่งต้องการภาษาที่สามารถเรนเดอร์ UI ได้แบบ Native Performance และควบคุมการวาดภาพทุกพิกเซลบนหน้าจอ Dart สามารถจัดการกระบวนการนี้ได้โดยไม่ต้องพึ่งพา bridge กับ native code มากนัก ส่งผลให้การทำงานลื่นไหลและตอบสนองได้อย่างทันที อีกทั้งยังรองรับการใช้ “Hot Reload” ซึ่งช่วยให้พัฒนาและปรับแต่ง UI ได้รวดเร็วโดยไม่ต้องเริ่มรันโปรแกรมใหม่

การเริ่มต้นใช้งาน Dart จำเป็นต้องติดตั้ง Dart SDK ซึ่งเป็นชุดเครื่องมือหลักที่ใช้ในการคอมไพล์และรันโปรแกรม SDK นี้มีทั้งตัวคอมไพเลอร์ ไลบรารีมาตรฐาน และเครื่องมือบรรทัดคำสั่ง

(CLI) โดยสามารถติดตั้งได้บนหลายระบบปฏิบัติการ เช่น Windows, macOS และ Linux การติดตั้งที่ถูกต้องจะช่วยให้นักพัฒนาสามารถเริ่มเขียนโปรแกรม Dart ได้อย่างมีประสิทธิภาพและไม่พบปัญหาระหว่างการใช้งาน

สำหรับผู้ที่ต้องการทดลองเขียน Dart อย่างรวดเร็วโดยไม่ต้องติดตั้งเครื่องมือใดๆ สามารถใช้ **DartPad** ซึ่งเป็น Online Playground ที่พัฒนาโดย Google โดย DartPad ให้ผู้ใช้เขียนโค้ด ทดสอบและรันโปรแกรม Dart ได้ผ่านเว็บเบราว์เซอร์ พร้อมแสดงผลลัพธ์แบบทันที เหมาะสำหรับการทดลองฟังก์ชันของภาษา การเรียนการสอน หรือการแบ่งปันโค้ดตัวอย่าง

นอกจาก Dart SDK แล้ว เครื่องมือพัฒนา (IDE) ก็มีบทบาทสำคัญต่อความสะดวกในการเขียนโค้ด ตัวเลือกยอดนิยมคือ **Visual Studio Code (VS Code)** และ **IntelliJ IDEA** ซึ่งสามารถติดตั้ง Dart Plugin เพื่อเพิ่มความสามารถในการทำงาน เช่น การแนะนำโค้ดอัตโนมัติ (Code Completion) การจัดรูปแบบโค้ด (Formatting) และการดีบักโปรแกรม การตั้งค่าเครื่องมือเหล่านี้อย่างถูกต้องจะช่วยให้การพัฒนามีประสิทธิภาพสูงขึ้น

การรันโปรแกรม Dart สามารถทำได้หลายวิธี ทั้งการใช้บรรทัดคำสั่งผ่าน Terminal/Command Prompt และการรันผ่าน IDE การใช้บรรทัดคำสั่งเหมาะสำหรับงานที่ต้องการความรวดเร็วและควบคุมได้อย่างละเอียด ส่วนการรันผ่าน IDE จะเหมาะกับผู้ที่ต้องการความสะดวก เช่น การตั้งค่า Breakpoint เพื่อดีบัก หรือการใช้ Extension และ Plugin เสริม

โดยสรุป บทนี้จะพาผู้อ่านทำความเข้าใจพื้นฐานของภาษา Dart และเหตุผลที่ภาษา Dart เหมาะสมกับ Flutter พร้อมทั้งแนะนำการติดตั้งเครื่องมือและสภาพแวดล้อมที่จำเป็น ทั้ง Dart SDK, DartPad และ IDE ยอดนิยม ตลอดจนการรันโปรแกรม Dart ในรูปแบบต่างๆ เพื่อให้ผู้อ่านสามารถเตรียมความพร้อมก่อนเข้าสู่การเรียนรู้การเขียนโปรแกรมด้วย Dart อย่างเต็มรูปแบบในบทถัดไป

แนะนำ Dart และการติดตั้ง

- Dart คืออะไร และทำไมใช้กับ Flutter
- วิธีติดตั้ง Dart SDK
- การใช้ DartPad (Online Playground)
- การติดตั้ง VS Code / IntelliJ + Dart Plugin
- การรันโปรแกรม Dart (Command line & IDE)

Dart คืออะไร และทำไมใช้กับ Flutter?

Dart คือภาษาโปรแกรมมิ่งแบบ **Object-Oriented** ที่พัฒนาโดย **Google** มีจุดเด่นคือถูกออกแบบมาเพื่อสร้างแอปพลิเคชันที่มีประสิทธิภาพสูงและรันได้บนหลายแพลตฟอร์มพร้อมกัน (Multi-platform)
เหตุผลหลักที่ Dart ถูกใช้กับ **Flutter** คือ:

1. **ประสิทธิภาพสูง (High Performance):** Dart สามารถคอมไพล์เป็น **Native Code** ได้ ทำให้แอปพลิเคชันทำงานได้เร็วเหมือนแอปที่เขียนด้วยภาษาเฉพาะของแพลตฟอร์มนั้น ๆ (เช่น Java/Kotlin บน Android หรือ Swift/Objective-C บน iOS)

2. **JIT และ AOT Compilation:** ในระหว่างการพัฒนา Dart ใช้ **JIT (Just-in-Time) Compilation** ทำให้มีฟีเจอร์ **Hot Reload** ที่ช่วยให้คุณเห็นการเปลี่ยนแปลงของโค้ดบน Emulator หรืออุปกรณ์จริงได้ทันที ส่วนเมื่อจะเผยแพร่แอป จะใช้ **AOT (Ahead-of-Time) Compilation** เพื่อให้แอปทำงานได้เร็วที่สุด
3. **ใช้งานง่าย (Easy to Learn):** Dart มีไวยากรณ์ (Syntax) ที่คล้ายคลึงกับภาษาที่นิยมอยู่แล้ว อย่าง C#, Java และ JavaScript ทำให้ผู้พัฒนาคุ้นเคยได้ง่าย
4. **รองรับ Null Safety:** Dart มีระบบ **Null Safety** ที่แข็งแกร่ง ช่วยป้องกันข้อผิดพลาดที่เกิดจากค่า null ได้ตั้งแต่ขั้นตอนการเขียนโค้ด ทำให้โค้ดมีความน่าเชื่อถือและปลอดภัยมากขึ้น

วิธีติดตั้ง Dart SDK

ถ้าคุณต้องการพัฒนา Flutter คุณไม่จำเป็นต้องติดตั้ง Dart SDK แยกต่างหาก เพราะ **Flutter SDK จะมาพร้อมกับ Dart SDK อยู่แล้ว**

แต่ถ้าต้องการใช้ Dart เดี่ยว ๆ โดยไม่ผ่าน Flutter สามารถทำตามขั้นตอนเหล่านี้:

1. **ไปที่เว็บไซต์:** ไปที่ dart.dev/get-dart
2. **เลือกแพลตฟอร์ม:** เลือกวิธีการติดตั้งที่เหมาะกับระบบปฏิบัติการของคุณ (Windows, macOS, Linux)
3. **ดาวน์โหลดและติดตั้ง:** ทำตามคำแนะนำบนหน้าเว็บสำหรับแพลตฟอร์มนั้น ๆ
4. **ตรวจสอบการติดตั้ง:** เปิด Terminal (หรือ Command Prompt บน Windows) แล้วพิมพ์คำสั่ง dart --version ถ้าติดตั้งสำเร็จจะแสดงเวอร์ชันของ Dart ออกมา

การใช้ DartPad (Online Playground)

DartPad คือเครื่องมือที่ง่ายและเร็วที่สุดในการเริ่มต้นเขียน Dart โดยไม่ต้องติดตั้งอะไรเลย เหมาะสำหรับฝึกฝนหรือทดลองเขียนโค้ดสั้น ๆ

วิธีใช้:

1. ไปที่ dartpad.dev
2. คุณจะเห็นพื้นที่สำหรับเขียนโค้ดและพื้นที่สำหรับดูผลลัพธ์
3. ลองเขียนโค้ด Dart อย่างง่าย ๆ เช่น:

Dart

```dart
void main() {
  print('Hello, Dart!');
}
```

4. กดปุ่ม **Run** เพื่อดูผลลัพธ์

การติดตั้ง VS Code / IntelliJ + Dart Plugin

การใช้ **Integrated Development Environment (IDE)** จะทำให้การเขียนโค้ดง่ายและมีประสิทธิภาพมากขึ้น IDE ยอดนิยมสำหรับ Dart/Flutter มี 2 ตัวหลักคือ **VS Code** และ **IntelliJ IDEA**

สำหรับ VS Code:

1. ติดตั้ง Visual Studio Code
2. เปิด VS Code
3. ไปที่ส่วน **Extensions** (ไอคอนสี่เหลี่ยม) หรือกด Ctrl+Shift+X
4. ค้นหา **"Dart"** และติดตั้ง **Dart Extension** ของ Dart-Code
5. เมื่อติดตั้งเสร็จแล้ว VS Code ก็พร้อมสำหรับการเขียนโค้ด Dart และ Flutter

สำหรับ IntelliJ IDEA:

1. ติดตั้ง IntelliJ IDEA Community Edition
2. เปิด IntelliJ
3. ไปที่ **Settings / Preferences -> Plugins**
4. ค้นหา **"Dart"** และติดตั้ง Dart Plugin
5. ติดตั้งเสร็จแล้ว ให้เริ่มโปรเจกต์ใหม่และเลือก Dart SDK ที่ติดตั้งไว้

การรันโปรแกรม Dart (Command line & IDE)

เมื่อติดตั้งทุกอย่างพร้อมแล้ว คุณสามารถรันโปรแกรม Dart ได้หลายวิธี

1. การรันผ่าน Command Line

1. สร้างไฟล์ Dart ชื่อ hello.dart โดยมีโค้ดดังนี้:

```dart
void main() {
  print('Hello, Dart from Command Line!');
}
```

2. เปิด Terminal หรือ Command Prompt
3. ไปยังไดเรกทอรีที่บันทึกไฟล์ hello.dart ไว้
4. พิมพ์คำสั่ง: dart run hello.dart
5. โปรแกรมจะรันและแสดงผลลัพธ์ใน Terminal

2. การรันผ่าน VS Code / IntelliJ

1. เปิดไฟล์ .dart ใน IDE ของคุณ
2. **VS Code:**
   - ถ้ามีฟังก์ชัน main() อยู่ในไฟล์ จะมีปุ่ม **Run** หรือ **Debug** ปรากฏขึ้นมาเหนือฟังก์ชัน
   - คุณสามารถกดปุ่ม **Run** เพื่อรันโปรแกรมได้เลย
   - หรือกด F5 เพื่อเข้าโหมด Debugging
3. **IntelliJ:**

- คลิกขวาที่ไฟล์ .dart แล้วเลือก **Run 'ชื่อไฟล์.dart'**
- จะมีหน้าต่าง **Run** ปรากฏขึ้นมาพร้อมผลลัพธ์

เจาะลึก Dart: รายละเอียดเชิงลึก

Dart ไม่ได้เป็นแค่ภาษาที่ใช้เขียน Flutter แต่มีแนวคิดและโครงสร้างที่น่าสนใจหลายอย่าง ซึ่งเป็นรากฐานที่ทำให้ Flutter มีประสิทธิภาพสูง

1. Dart Virtual Machine (VM)

Dart มี VM ของตัวเองที่แตกต่างกันไปตามโหมดการทำงาน:

- **Dart JIT (Just-In-Time) VM:** ใช้ในระหว่างการ **พัฒนา** (flutter run, flutter hot-reload)
  - **Hot Reload:** ฟีเจอร์นี้คือจุดเด่นที่ทำให้การพัฒนา Flutter รวดเร็วมาก เมื่อคุณแก้ไขโค้ด Dart ในไฟล์ lib/ JIT VM จะส่งการเปลี่ยนแปลงเฉพาะส่วนนั้น ๆ ไปยังแอปที่กำลังทำงานอยู่ ทำให้แอปอัปเดต UI ทันทีโดยไม่ต้องเริ่มต้นใหม่ทั้งหมด
  - **JIT Compiler:** คอมไพล์โค้ดในระหว่างที่แอปทำงาน ช่วยให้การแก้ไขและทดสอบโค้ดทำได้รวดเร็ว
- **Dart AOT (Ahead-Of-Time) Compiler:** ใช้ในระหว่างการ **คอมไพล์เพื่อเผยแพร่** (flutter build)
  - **Native Code:** AOT Compiler จะคอมไพล์โค้ด Dart ทั้งหมดให้เป็น **Native ARM Code** ที่ทำงานบนอุปกรณ์ iOS และ Android โดยตรง ทำให้ได้ประสิทธิภาพสูงสุด
  - **ขนาดแอปเล็ก:** เนื่องจากไม่มีการบรรจุ JIT Compiler หรือ VM ลงไปในแอปที่เผยแพร่จริง ทำให้ขนาดของแอปเล็กลง และทำงานได้เร็วขึ้น

2. Everything is an Object

ใน Dart ทุกสิ่งคือ **Object** ไม่ว่าจะเป็นตัวเลข (int, double), ข้อความ (String), หรือแม้กระทั่งฟังก์ชันเองก็ตาม ทุก Object จะสืบทอดมาจากคลาสแม่ชื่อ Object

- **Null Safety:** ระบบ Null Safety ของ Dart ใช้แนวคิดที่ว่าตัวแปรจะไม่สามารถมีค่า null ได้เลยถ้าไม่ได้ระบุชัดเจนด้วยเครื่องหมาย ?
  - String name = 'Alice'; // **Non-nullable:** name จะไม่สามารถเป็น null ได้
  - String? greeting = null; // **Nullable:** greeting สามารถเป็น null ได้
  - การใช้ Null Safety ช่วยลดข้อผิดพลาด Runtime Error ที่เกิดจาก null ได้อย่างมีประสิทธิภาพ

3. Asynchronous Programming (Async/Await)

Dart ถูกออกแบบมาให้รองรับการทำงานแบบ Asynchronous ได้อย่างดีเยี่ยม โดยใช้คำสั่ง async และ await เพื่อจัดการกับงานที่ใช้เวลา เช่น การเรียก API, การอ่านไฟล์ หรือการทำงานกับฐานข้อมูล

- **Future:** ใน Dart ข้อมูลที่จะได้จากการทำงานแบบ Asynchronous จะถูกห่อหุ้มไว้ในอ็อบเจกต์ **Future**
- **async และ await:**
  - ใช้คำสั่ง **async** ที่หน้าฟังก์ชันเพื่อระบุว่าฟังก์ชันนี้จะมีการทำงานแบบ Asynchronous
  - ใช้คำสั่ง **await** เพื่อรอให้ Future ดำเนินการเสร็จสิ้นก่อนที่จะทำงานในบรรทัดถัดไป

- โค้ดที่ใช้ async/await จะอ่านง่ายเหมือนโค้ด Synchronous ทั่วไป แต่เบื้องหลังยังคงทำงานแบบ Asynchronous เพื่อไม่ให้แอปค้าง (Block)

ตัวอย่าง:

```dart
Future<String> fetchUserData() async {
  // สมมติเป็นการเรียก API
  await Future.delayed(Duration(seconds: 2)); // รอ 2 วินาที
  return 'User data loaded!';
}

void main() async {
  print('Fetching data...');
  String data = await fetchUserData();
  print(data); // จะพิมพ์ 'User data loaded!' หลังจากรอ 2 วินาที
}
```

4. Isolates

ในขณะที่ภาษาอื่นใช้ Multi-threading, Dart ใช้แนวคิดที่เรียกว่า **Isolates** ซึ่งเป็นเหมือนหน่วยการทำงานอิสระที่แยกจากกันโดยสิ้นเชิง แต่ละ Isolate มีหน่วยความจำ (Memory) เป็นของตัวเอง ทำให้ไม่สามารถแชร์ข้อมูลกันได้โดยตรง

- **ป้องกัน Deadlock:** การใช้ Isolates ช่วยป้องกันปัญหา Deadlock (ภาวะที่โปรแกรมค้าง) ที่มักเกิดขึ้นในการทำงานแบบ Multi-threading
- **ทำงานแบบ Parallel:** Isolates ทำให้ Dart สามารถทำงานแบบขนาน (Parallel) ได้อย่างปลอดภัย เหมาะสำหรับงานที่ต้องใช้การประมวลผลสูง เช่น การคำนวณที่ซับซ้อน หรือการประมวลผลภาพ

Dart คืออะไร (ฉบับเจาะลึก)

Dart คืออะไร (ฉบับเจาะลึก)

Dart คือภาษาโปรแกรมมิ่งแบบ **Object-Oriented** ที่พัฒนาโดย **Google** ในปี 2011 แต่เดิมถูกออกแบบมาเพื่อเป็นคู่แข่งของ JavaScript บนเบราว์เซอร์ แต่ต่อมาได้ปรับเปลี่ยนจุดยืนเพื่อมุ่งเน้นการสร้างแอปพลิเคชันที่ทำงานได้บนหลายแพลตฟอร์ม ซึ่งเป็นรากฐานสำคัญของเฟรมเวิร์ก **Flutter**

Key Characteristics (คุณสมบัติสำคัญ)

1. **Object-Oriented (เน้นวัตถุ):** ใน Dart **ทุกสิ่งคือ Object** ไม่เว้นแม้แต่ตัวเลข, บูลีน, หรือแม้กระทั่งฟังก์ชัน ทุก Object เป็น instance ของคลาส และทุกคลาสสืบทอดมาจากคลาสแม่ชื่อ Object สิ่งนี้ทำให้ Dart มีโครงสร้างที่สม่ำเสมอและจัดการได้ง่าย

2. **Statically Typed แต่ก็ยืดหยุ่น:** Dart เป็นภาษาแบบ **Statically Typed** ซึ่งหมายถึงการตรวจสอบประเภทของตัวแปรในระหว่างการเขียนโค้ด ทำให้สามารถตรวจจับข้อผิดพลาดได้ตั้งแต่เนิ่น ๆ แต่ก็ยังคงความยืดหยุ่นด้วยการใช้คีย์เวิร์ด var และ dynamic ซึ่งช่วยให้การเขียนโค้ดเป็นไปอย่างรวดเร็ว
3. **Ahead-of-Time (AOT) และ Just-in-Time (JIT) Compilation:** นี่คือหัวใจสำคัญที่ทำให้ Dart เหนือกว่าภาษาอื่น ๆ ในบางแง่มุม
   - **JIT (Just-in-Time) Compilation:** ใช้ในระหว่างการ **พัฒนา** โค้ดจะถูกคอมไพล์และรันใน Dart Virtual Machine (VM) ทำให้มีฟีเจอร์ที่เรียกว่า **Hot Reload** ซึ่งช่วยให้คุณเห็นการเปลี่ยนแปลงของ UI ได้ในเสี้ยววินาที ทำให้การพัฒนาเป็นไปอย่างรวดเร็ว
   - **AOT (Ahead-of-Time) Compilation:** ใช้ในระหว่างการ **Build เพื่อ Production** โค้ดจะถูกคอมไพล์เป็น **Native Code** (เช่น ARM code สำหรับอุปกรณ์มือถือ) โดยตรง ทำให้แอปพลิเคชันทำงานได้รวดเร็วและมีประสิทธิภาพสูงสุดโดยไม่ต้องใช้ VM ในระหว่างรัน
4. **Isolates (หน่วยการทำงานแบบแยกส่วน):** แทนที่จะใช้ Multi-threading แบบดั้งเดิมที่อาจก่อให้เกิดปัญหา deadlock, Dart ใช้ **Isolates** ซึ่งเป็นหน่วยการทำงานอิสระที่มีหน่วยความจำของตัวเอง ทำให้แต่ละส่วนทำงานแบบขนาน (Parallel) ได้อย่างปลอดภัยและมีประสิทธิภาพ
5. **Null Safety (ความปลอดภัยจากค่า Null):** Dart เป็นหนึ่งในภาษาแรก ๆ ที่นำระบบ **Sound Null Safety** มาใช้อย่างจริงจัง ทำให้มั่นใจได้ว่าตัวแปรที่คุณไม่ได้กำหนดให้เป็น nullable จะไม่มีทางเป็น null ได้เลย ช่วยลดข้อผิดพลาดที่พบได้บ่อยในภาษาอื่น ๆ

ทำไม Dart ถึงใช้กับ Flutter (ฉบับเจาะลึก)

ความสัมพันธ์ระหว่าง Dart และ Flutter ไม่ใช่เรื่องบังเอิญ แต่เป็นการออกแบบมาอย่างพิถีพิถันจาก Google เพื่อให้ทำงานร่วมกันได้อย่างสมบูรณ์แบบ

1. การสร้าง UI ได้อย่างรวดเร็ว (Hot Reload)

Flutter อาศัย JIT Compilation ของ Dart ในระหว่างการพัฒนา เมื่อคุณแก้ไขโค้ดที่เกี่ยวข้องกับ UI, Dart VM จะอัปเดตเฉพาะส่วนที่เปลี่ยนไปบนหน้าจอแอปโดยไม่ต้องรีสตาร์ทแอปใหม่ ทำให้คุณสามารถทดลองและปรับปรุงดีไซน์ได้แบบเรียลไทม์ ซึ่งช่วยประหยัดเวลาได้อย่างมหาศาล

2. ประสิทธิภาพสูงสุดเมื่อใช้งานจริง (AOT Compilation)

เมื่อถึงเวลาเผยแพร่แอป (Production build), AOT Compiler ของ Dart จะแปลงโค้ด Dart และ Widgets ของ Flutter ให้กลายเป็นโค้ดภาษาเครื่อง (Machine Code) โดยตรง ทำให้แอปที่สร้างด้วย Flutter มีประสิทธิภาพใกล้เคียงกับแอปที่เขียนด้วยภาษาเนทีฟอย่าง Swift หรือ Kotlin

3. สถาปัตยกรรมที่ยืดหยุ่นและมีประสิทธิภาพ

Dart ถูกออกแบบมาให้สามารถจัดการกับ UI ได้อย่างมีประสิทธิภาพในแบบที่ภาษาอื่นทำได้ยาก Flutter ใช้แนวคิดของ **Widgets** ซึ่งเป็นเหมือนบล็อกการสร้าง UI ทั้งหมด ตั้งแต่ปุ่มไปจนถึงหน้าจอ

ทั้งหมด และ Dart มีไวยากรณ์ที่เหมาะสมกับการจัดการและสร้าง Widget Tree ที่ซับซ้อนได้อย่างเป็นระเบียบ

4. Asynchronous Programming (การทำงานแบบไม่บล็อก)

การทำงานกับ API, ฐานข้อมูล, หรือการโหลดไฟล์เป็นเรื่องปกติในการพัฒนาแอป Dart มีระบบจัดการ Asynchronous ที่ยอดเยี่ยมด้วย async และ await ทำให้แอปสามารถทำงานเหล่านี้ในพื้นหลังได้อย่างราบรื่นโดยไม่ทำให้ UI ค้าง (ไม่เกิดอาการ Blocked UI) ซึ่งเป็นสิ่งสำคัญสำหรับแอปพลิเคชันที่มีการโต้ตอบกับผู้ใช้อย่างต่อเนื่อง

กล่าวโดยสรุป, Dart ไม่ได้เป็นแค่ภาษาที่ Flutter เลือกใช้ แต่เป็น **เครื่องมือที่ถูกสร้างมาเพื่อ Flutter โดยเฉพาะ** คุณสมบัติเฉพาะตัวของ Dart ถูกนำมาใช้ประโยชน์อย่างเต็มที่ในเฟรมเวิร์ก Flutter เพื่อมอบประสบการณ์การพัฒนาที่รวดเร็วและประสิทธิภาพของแอปที่ยอดเยี่ยมที่สุดครับ

ขอแสดงความยินดีครับ! ในฐานะผู้เชี่ยวชาญ Dart ผมจะสรุปประวัติ วิวัฒนาการ และข้อมูลที่สำคัญทั้งหมดให้คุณอย่างละเอียดครับ

ประวัติและวิวัฒนาการ (History & Evolution)

- **จุดเริ่มต้น (2011):** Dart ถูกเปิดตัวครั้งแรกโดย Google ในการประชุม GOTO Conference ที่เมือง Aarhus ประเทศเดนมาร์ก โดย Lars Bak และ Kasper Lund ซึ่งมีเป้าหมายเริ่มต้นคือเพื่อเป็นภาษาที่ใช้แทน JavaScript ในการพัฒนาเว็บ โดยวางแผนที่จะรวม Dart VM เข้าไปในเบราว์เซอร์ Chrome โดยตรง อย่างไรก็ตาม แนวคิดนี้ก็ไม่ได้ถูกนำไปใช้ในที่สุด
- **Dart 1.x (2013):** Dart 1.0 ถูกปล่อยออกมาอย่างเป็นทางการ โดยมีจุดเด่นเรื่องความเป็น Static Analysis ที่สามารถตรวจสอบประเภทข้อมูลได้ตั้งแต่ตอนเขียนโค้ด (Optional Type) และมีการเปิดตัว dart2js compiler ที่แปลงโค้ด Dart เป็น JavaScript เพื่อให้รันบนเว็บได้ทุกเบราว์เซอร์
- **Dart 2.0 (2018):** นี่คือจุดเปลี่ยนที่สำคัญที่สุดของ Dart โดยมีการประกาศอย่างชัดเจนว่าจะมุ่งเน้นไปที่การสร้างแอปพลิเคชันแบบ **Client-side** (ฝั่งผู้ใช้) มีการปรับปรุงระบบประเภทข้อมูลให้เป็น **Strongly Typed** มากขึ้น และมีการเพิ่มฟีเจอร์ใหม่ ๆ ที่ทำให้การเขียนโค้ดง่ายขึ้นและมีประสิทธิภาพมากขึ้น
- **Null Safety (2021):** ใน Dart 2.12 มีการนำเสนอระบบ **Sound Null Safety** ที่เป็นฟีเจอร์สำคัญ ช่วยป้องกันข้อผิดพลาดที่เกิดจาก null ได้อย่างมีประสิทธิภาพ ทำให้โค้ดมีความเสถียรและปลอดภัยยิ่งขึ้น
- **Dart 3.0 (2023):** Dart 3.0 ได้นำเสนอคุณสมบัติใหม่ที่น่าสนใจ เช่น **Records**, **Patterns** และ **Sealed Classes** ซึ่งทำให้การเขียนโค้ดสั้นลง อ่านง่ายขึ้น และมีประสิทธิภาพมากขึ้น รวมถึงการรองรับการคอมไพล์โค้ดเป็น **WebAssembly (WASM)** ที่ทำให้ประสิทธิภาพของแอปพลิเคชันบนเว็บดีขึ้นอย่างเห็นได้ชัด

แนวโน้มและสถิติการใช้งาน

- **ความนิยมที่เพิ่มขึ้น:** จากรายงานของ Stack Overflow Developer Survey (2023) ภาษา Dart ติดอันดับต้น ๆ ในหมวด "Most Loved Languages" และ "Most Wanted Languages" ซึ่งสะท้อนให้เห็นว่านักพัฒนาที่ใช้งาน Dart มีความพึงพอใจสูงและผู้ที่ไม่เคยใช้ก็ต้องการที่จะเรียนรู้

แนวโน้มการใช้งาน Dart เติบโตอย่างต่อเนื่องและผูกพันอย่างแยกไม่ออกกับความนิยมของ **Flutter**

- **การใช้งานกับ Flutter:** Dart กลายเป็นภาษาหลักสำหรับการพัฒนาแอปพลิเคชัน Cross-platform ด้วย Flutter ทำให้สามารถสร้างแอปที่ทำงานบน iOS, Android, Web, Desktop และ Embedded System ได้จาก codebase เดียวกัน ซึ่งเป็นเหตุผลหลักที่ทำให้ Dart เป็นที่ต้องการในตลาดแรงงาน
- **ขยายสู่ Backend:** แม้ว่า Flutter จะเป็นที่นิยมที่สุด แต่ Dart ก็สามารถใช้ในการพัฒนา Backend หรือ Server-side ได้เช่นกัน แม้จะยังไม่แพร่หลายเท่าภาษาอื่น ๆ แต่ก็มีไลบรารีและเฟรมเวิร์กอย่าง **Dart Frog** ที่ช่วยให้การพัฒนาเป็นเรื่องง่าย และเป็นตัวเลือกที่ดีสำหรับทีมที่ต้องการใช้ภาษาเดียวทั้ง Full-stack

องค์กรที่ใช้งาน Dart

บริษัทและองค์กรระดับโลกหลายแห่งหันมาใช้ Flutter และ Dart ในการพัฒนาแอปพลิเคชันของตนเอง ซึ่งรวมถึง:

- **Google:** ผู้พัฒนา Dart และ Flutter เอง โดยมีการใช้งานในหลาย ๆ โปรเจกต์ เช่น Google Assistant และ Google Pay
- **Alibaba:** ใช้ Flutter ในแอปพลิเคชันค้าปลีก
- **eBay:** ใช้ Flutter ในส่วนของแอป eBay Motors
- **BMW:** ใช้ Flutter ในการสร้างแอปสำหรับรถยนต์ไฟฟ้า
- **Tencent:** บริษัทเทคโนโลยียักษ์ใหญ่จากจีนก็ใช้ Flutter ในบางโปรเจกต์
- **Skooldio:** องค์กรในประเทศไทยที่ใช้ Flutter ในการพัฒนาแอปพลิเคชันของตนเอง

นอกจากนี้ ยังมีองค์กรขนาดเล็กและสตาร์ทอัพอีกมากมายที่เลือกใช้ Flutter และ Dart เนื่องจากช่วยให้ประหยัดเวลาและค่าใช้จ่ายในการพัฒนาแอปพลิเคชันบนหลายแพลตฟอร์มพร้อมกันครับ

คุณสามารถดูข้อมูลเพิ่มเติมเกี่ยวกับภาษา Dart และ Flutter จากคลิปนี้ได้ รู้จัก ภาษา Dart ที่ใช้พัฒนาแอปผ่าน Flutter ใน 1 นาที | Skooldio Support Highlights.

ในการติดตั้ง Dart SDK มีหลายวิธีขึ้นอยู่กับระบบปฏิบัติการของคุณครับ ผมจะอธิบายแต่ละวิธีอย่างละเอียดให้คุณเลือกใช้ตามความเหมาะสม

วิธีที่ 1: การติดตั้ง Dart พร้อมกับ Flutter SDK (แนะนำ)

นี่เป็นวิธีที่ง่ายและได้รับความนิยมที่สุด เพราะหากคุณต้องการใช้ Dart กับ Flutter คุณไม่จำเป็นต้องติดตั้ง Dart แยกต่างหาก เนื่องจาก **Flutter SDK ได้รวม Dart SDK มาให้แล้ว**

ขั้นตอน:

1. ไปที่เว็บไซต์ทางการของ Flutter: https://flutter.dev/docs/get-started/install
2. เลือกแพลตฟอร์มของคุณ (Windows, macOS, Linux)
3. ทำตามคำแนะนำในการดาวน์โหลดและติดตั้ง Flutter SDK
4. หลังจากติดตั้งเสร็จสิ้น คุณสามารถตรวจสอบการทำงานของ Dart ได้ด้วยการเปิด Terminal (หรือ Command Prompt บน Windows) แล้วพิมพ์คำสั่ง:

Bash

dart --version

ถ้าทุกอย่างถูกต้อง ระบบจะแสดงเวอร์ชันของ Dart SDK ที่ถูกติดตั้งพร้อมกับ Flutter ออกมา

วิธีที่ 2: การติดตั้ง Dart SDK เดี่ยว ๆ (สำหรับผู้ที่ต้องการใช้ Dart โดยไม่ผ่าน Flutter)

ถ้าคุณต้องการใช้ Dart สำหรับโปรเจกต์ที่ไม่มี Flutter เช่น การเขียนสคริปต์, เว็บแอปพลิเคชันฝั่ง Server (Server-side) หรือ Command-line App สามารถติดตั้งได้ตามระบบปฏิบัติการของคุณ

สำหรับ macOS

- **ใช้ Homebrew:** นี่เป็นวิธีที่ง่ายที่สุด

  Bash

  brew tap dart-lang/dart

  brew install dart

  หลังจากติดตั้งเสร็จสิ้น คุณสามารถตรวจสอบเวอร์ชันได้ด้วย dart --version

สำหรับ Windows

1. **ใช้ Chocolatey (แนะนำ):**

   PowerShell

   choco install dart-sdk

2. **ดาวน์โหลดและติดตั้งด้วยตนเอง:**

- ไปที่เว็บไซต์ทางการของ Dart: https://dart.dev/get-dart
- เลือกแพลตฟอร์ม Windows และดาวน์โหลดไฟล์ .zip
- แตกไฟล์ .zip ไปยังตำแหน่งที่คุณต้องการ เช่น C:\tools\dart-sdk
- ตั้งค่า **PATH Environment Variable** เพื่อให้ระบบสามารถเรียกใช้คำสั่ง dart ได้จากทุกที่
  - ค้นหา "Edit the system environment variables" ใน Start Menu
  - คลิก "Environment Variables..."
  - ในส่วน "System variables" ให้หา Path แล้วกด "Edit..."
  - เพิ่มพาธไปยังโฟลเดอร์ bin ของ Dart SDK เช่น C:\tools\dart-sdk\bin
  - ปิดและเปิด Command Prompt ใหม่ แล้วลองพิมพ์ dart --version

สำหรับ Linux

- **ใช้ Package Manager (แนะนำ):**
- **สำหรับ Ubuntu / Debian:**

  Bash

  sudo apt-get update

  sudo apt-get install dart

- **สำหรับ Fedora:**

```
Bash
sudo dnf install dart
```

- **ดาวน์โหลดและติดตั้งด้วยตนเอง:**
  - ไปที่เว็บไซต์ทางการของ Dart: https://dart.dev/get-dart
  - ทำตามคำแนะนำในการเพิ่ม repository และติดตั้ง Dart SDK

สรุป

- **ถ้าคุณจะใช้ Flutter:** ให้ติดตั้ง Flutter SDK อย่างเดียวก็เพียงพอ
- **ถ้าคุณต้องการใช้ Dart เดี่ยว ๆ:** เลือกวิธีการติดตั้งตามระบบปฏิบัติการของคุณ (Homebrew สำหรับ macOS, Chocolatey สำหรับ Windows หรือ Package Manager สำหรับ Linux)

หลังจากติดตั้งเรียบร้อยแล้ว คุณสามารถใช้ DartPad เพื่อทดลองเขียนโค้ดได้ทันที หรือถ้าต้องการเขียนโปรแกรมแบบเต็มรูปแบบ แนะนำให้ใช้ Visual Studio Code หรือ IntelliJ IDEA และติดตั้ง Dart Plugin เพิ่มเติมครับ

การติดตั้ง Dart SDK โดยละเอียด

การติดตั้ง Dart มีสองแนวทางหลักๆ คือการติดตั้งผ่านตัวจัดการแพ็กเกจ (Package Manager) ซึ่งเป็นวิธีที่ง่ายที่สุด หรือการติดตั้งแบบ Manual โดยการดาวน์โหลดไฟล์โดยตรง

1. ติดตั้งบน Windows

มีสองวิธีที่แนะนำสำหรับผู้ใช้ Windows:

วิธีที่ 1: ใช้ Chocolatey (แนะนำ)

Chocolatey คือตัวจัดการแพ็กเกจสำหรับ Windows ที่ช่วยให้การติดตั้งโปรแกรมต่างๆ เป็นเรื่องง่าย หากคุณยังไม่มี Chocolatey ให้ติดตั้งก่อน:

1. เปิด **PowerShell** ในฐานะผู้ดูแลระบบ (Run as Administrator)
2. วางคำสั่งด้านล่างนี้แล้วกด Enter:

```
PowerShell
Set-ExecutionPolicy Bypass -Scope Process -Force;
[System.Net.ServicePointManager]::SecurityProtocol =
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object
System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))
```

3. เมื่อติดตั้ง Chocolatey เสร็จแล้ว ให้ปิดและเปิด PowerShell ใหม่
4. ติดตั้ง Dart SDK ด้วยคำสั่ง:

```
PowerShell
choco install dart-sdk
```

5. ตรวจสอบเวอร์ชันด้วยคำสั่ง:

PowerShell

```
dart --version
```

วิธีที่ 2: ติดตั้งแบบ Manual

1. ไปที่ เว็บไซต์ทางการของ Dart
2. ดาวน์โหลดไฟล์ .zip สำหรับ Windows
3. แตกไฟล์ .zip ไปยังตำแหน่งที่คุณต้องการ เช่น C:\tools\dart-sdk
4. **ตั้งค่า PATH Environment Variable:** เพื่อให้คุณสามารถเรียกใช้คำสั่ง dart จากที่ไหนก็ได้ใน Command Prompt
   - ค้นหา "Edit the system environment variables" ใน Start Menu
   - คลิก "Environment Variables..."
   - ในส่วน **System variables** ให้หา Path แล้วกด "Edit..."
   - คลิก "New" และเพิ่มพาธไปยังโฟลเดอร์ bin ของ Dart SDK เช่น C:\tools\dart-sdk\bin
   - กด OK เพื่อบันทึกการเปลี่ยนแปลงทั้งหมด
5. ปิด Command Prompt หรือ PowerShell ทั้งหมดที่มีอยู่ แล้วเปิดใหม่
6. ลองพิมพ์ dart --version เพื่อตรวจสอบ

---

2. ติดตั้งบน macOS

วิธีที่ 1: ใช้ Homebrew (แนะนำ)

Homebrew เป็นตัวจัดการแพ็กเกจยอดนิยมสำหรับ macOS ที่ทำให้การติดตั้งซอฟต์แวร์ต่างๆ ง่ายมาก

1. เปิด Terminal
2. ติดตั้ง Homebrew (หากยังไม่มี) ด้วยคำสั่ง:

   Bash

   ```
   /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
   ```
3. ติดตั้ง Dart SDK ด้วยคำสั่ง:

   Bash

   ```
   brew tap dart-lang/dart
   brew install dart
   ```
4. ตรวจสอบเวอร์ชันด้วยคำสั่ง:

   Bash

   ```
   dart --version
   ```

วิธีที่ 2: ติดตั้งแบบ Manual

1. ไปที่ เว็บไซต์ทางการของ Dart
2. ดาวน์โหลดไฟล์ .zip สำหรับ macOS
3. แตกไฟล์ .zip ไปยังตำแหน่งที่คุณต้องการ เช่น /usr/local/dart

4. ตั้งค่า PATH เพื่อให้สามารถเรียกใช้คำสั่ง dart ได้ (สามารถทำได้ในไฟล์ .zshrc หรือ .bash_profile ของคุณ)

```
Bash
export PATH="$PATH:/usr/local/dart/bin"
```

5. รีสตาร์ท Terminal หรือใช้คำสั่ง source ~/.zshrc
6. ลองพิมพ์ dart --version เพื่อตรวจสอบ

---

3. ติดตั้งบน Linux

วิธีที่ 1: ใช้ apt (สำหรับ Debian/Ubuntu)

1. เพิ่ม Dart repository ไปยังระบบของคุณ:[1]

```
Bash
sudo apt-get update
sudo apt-get install apt-transport-https
sudo sh -c 'wget -qO- https://dl-ssl.google.com/linux/linux_signing_key.pub | apt-key add -'
sudo sh -c 'wget -qO-
https://storage.googleapis.com/download.dartlang.org/linux/debian/dart_stable.list >
/etc/apt/sources.list.d/dart_stable.list'
```

2. ติดตั้ง Dart SDK:

```
Bash
sudo apt-get update
sudo apt-get install dart
```

3. ตรวจสอบเวอร์ชันด้วยคำสั่ง:

```
Bash
dart --version
```

วิธีที่ 2: ใช้ Snapcraft

1. ติดตั้ง Dart Snap โดยใช้คำสั่ง:

```
Bash
sudo snap install dart --classic
```

2. ตรวจสอบเวอร์ชันด้วยคำสั่ง:

```
Bash
dart --version
```

ไม่ว่าจะเลือกวิธีไหน สิ่งสำคัญคือการตรวจสอบว่าคำสั่ง dart --version ทำงานได้อย่างถูกต้อง หลังจากติดตั้งเสร็จสิ้น ก็พร้อมที่จะเริ่มเขียนโค้ด Dart ได้ทันที

DartPad (Online Playground) คืออะไร?

DartPad คือเครื่องมือออนไลน์ที่ถูกพัฒนาโดย Google เพื่อให้นักพัฒนาสามารถเขียน, รัน และทดสอบโค้ดภาษา Dart และ Flutter ได้โดยตรงจากเบราว์เซอร์ **โดยไม่ต้องติดตั้งซอฟต์แวร์ใด ๆ บนเครื่องคอมพิวเตอร์ของคุณเลย** มันเปรียบเสมือนสนามเด็กเล่นเสมือนจริงที่เหมาะสำหรับ:

- ผู้เริ่มต้นที่ต้องการเรียนรู้ภาษา Dart
- นักพัฒนาที่ต้องการทดลองฟีเจอร์ใหม่ ๆ ของภาษาอย่างรวดเร็ว
- การแบ่งปันโค้ดตัวอย่างกับผู้อื่น
- การสาธิตโค้ดในระหว่างการนำเสนอ

โครงสร้างและส่วนประกอบหลักของ DartPad

เมื่อคุณเข้าไปที่ dartpad.dev คุณจะเห็นหน้าจอที่แบ่งออกเป็น 3 ส่วนหลัก ๆ:

1. **Code Editor (ฝั่งซ้าย):**
   - นี่คือพื้นที่ที่คุณจะเขียนโค้ด Dart หรือ Flutter
   - มีคุณสมบัติพื้นฐานของ Editor ทั่วไป เช่น การเน้นไวยากรณ์ (Syntax Highlighting) และการเติมโค้ดอัตโนมัติ (Code Completion)
   - มีหน้าต่างแสดงผลลัพธ์ของข้อผิดพลาด (Errors), คำเตือน (Warnings) หรือข้อความแนะนำ (Hints) ที่ช่วยในการแก้ไขโค้ด
2. **Output Console (ฝั่งขวาบน):**
   - พื้นที่นี้จะแสดงผลลัพธ์ของโปรแกรมที่คุณรัน
   - หากคุณใช้คำสั่ง print() ผลลัพธ์ก็จะปรากฏที่นี่
   - เหมาะสำหรับการดูผลลัพธ์ของโปรแกรม Command-line อย่างง่าย ๆ
3. **UI Canvas / Documentation (ฝั่งขวาล่าง):**
   - ส่วนนี้จะถูกใช้เมื่อคุณเลือกโหมด **Flutter**
   - จะแสดงผลลัพธ์ของ UI ที่คุณสร้างขึ้นมาใน Editor เช่น Widget ต่าง ๆ ที่แสดงผลบนหน้าจอเสมือนจริง
   - หากคุณเลือกโหมด **Dart** ส่วนนี้จะเปลี่ยนเป็นแท็บ Docs ซึ่งจะแสดงเอกสารประกอบของฟังก์ชัน, คลาส หรือคีย์เวิร์ดที่คุณกำลังเลือกอยู่ เป็นฟีเจอร์ที่ช่วยในการเรียนรู้ได้เป็นอย่างดี

วิธีการใช้งาน DartPad อย่างละเอียด

1. **การเลือกโหมด (Dart vs. Flutter):**
   - ที่มุมบนขวาของ DartPad จะมีเมนูสำหรับเลือกโหมด
   - **"Dart"**: สำหรับการเขียนโค้ด Dart แบบธรรมดา, Command-line scripts หรือโค้ดที่มีการคำนวณ
   - **"Flutter"**: สำหรับการสร้างแอปพลิเคชัน UI ด้วยเฟรมเวิร์ก Flutter
   - **"HTML"**: สำหรับการสร้างเว็บเพจที่ใช้ Dart เป็น Logic (ใช้ได้ทั้ง Dart และ Flutter)
2. **การเขียนและรันโค้ด:**

- เริ่มเขียนโค้ดใน Editor เช่น โค้ด Dart พื้นฐาน:

```
Dart
void main() {
  print('Hello, DartPad!');
}
```

- กดปุ่ม **"Run"** ที่อยู่ด้านบนของหน้าจอ
- ผลลัพธ์จะปรากฏใน **Output Console**

3. **การใช้งานกับ Flutter:**

- เลือกโหมด "Flutter" ที่มุมบนขวา
- โค้ดตัวอย่าง Flutter จะปรากฏขึ้นมา
- ลองเปลี่ยนข้อความในโค้ด เช่น เปลี่ยนจาก Hello World เป็น Hello DartPad!
- กดปุ่ม **"Run"**
- คุณจะเห็น UI บน **UI Canvas** เปลี่ยนแปลงตามที่คุณแก้ไข

4. **การใช้คุณสมบัติอื่น ๆ:**

- **Format:** ปุ่ม Format ที่อยู่ด้านบนจะช่วยจัดระเบียบโค้ดของคุณให้เป็นไปตามมาตรฐาน dartfmt ทำให้โค้ดอ่านง่ายและเป็นระเบียบ
- **Samples:** มีตัวอย่างโค้ดให้เลือกดูมากมายในเมนู Samples เหมาะสำหรับผู้เริ่มต้นที่ต้องการดูโค้ดตัวอย่าง
- **Sharing:** คุณสามารถบันทึกโค้ดของคุณไว้บน URL เฉพาะ แล้วแชร์ลิงก์นั้นให้ผู้อื่นได้ โดยการกดปุ่ม Share
- **Documentation:** เมื่อคุณวางเคอร์เซอร์บนคีย์เวิร์ด, ฟังก์ชัน หรือคลาสในโหมด Dart หรือ Flutter เอกสารประกอบของสิ่งนั้น ๆ จะแสดงขึ้นมาในแท็บ Docs ซึ่งเป็นแหล่งเรียนรู้ที่ยอดเยี่ยม

ข้อจำกัดของ DartPad

แม้ว่า DartPad จะเป็นเครื่องมือที่ทรงพลัง แต่ก็มีข้อจำกัดบางประการ:

- ไม่สามารถใช้งานไลบรารีภายนอก (External Packages) ได้ทั้งหมด
- ไม่เหมาะสำหรับการพัฒนาโปรเจกต์ขนาดใหญ่
- การ Debugging อาจทำได้ไม่ละเอียดเท่ากับใน IDE เช่น VS Code หรือ IntelliJ

โดยสรุปแล้ว DartPad เป็นเครื่องมือชั้นเยี่ยมสำหรับการทดลองและเรียนรู้ภาษา Dart และ Flutter อย่างรวดเร็วและง่ายดาย เหมาะอย่างยิ่งสำหรับผู้ที่กำลังเริ่มต้นหรือต้องการทดลองไอเดียเล็ก ๆ น้อย ๆ

การตั้งค่า Integrated Development Environment (IDE) อย่าง VS Code หรือ IntelliJ

การตั้งค่า Integrated Development Environment (IDE) อย่าง VS Code หรือ IntelliJ ให้พร้อมใช้งาน Dart เป็นขั้นตอนสำคัญในการพัฒนา อธิบายอย่างละเอียดในแต่ละโปรแกรม

1. การติดตั้ง Dart Plugin ใน VS Code (Visual Studio Code)

VS Code เป็น IDE ที่ได้รับความนิยมอย่างมากเนื่องจากมีน้ำหนักเบาและมีความยืดหยุ่นสูง การติดตั้ง Dart plugin นั้นง่ายมาก

ขั้นตอน:

1. **ติดตั้ง Visual Studio Code:** หากยังไม่มี ให้ดาวน์โหลดและติดตั้งจาก https://code.visualstudio.com/
2. **เปิด VS Code:** เมื่อติดตั้งเสร็จแล้ว ให้เปิดโปรแกรมขึ้นมา
3. **ไปที่ส่วน Extensions:** ที่แถบด้านซ้ายมือ ให้คลิกที่ไอคอน Extensions (ไอคอนสี่เหลี่ยมสี่อัน) หรือกด Ctrl + Shift + X บน Windows/Linux หรือ Cmd + Shift + X บน macOS
4. **ค้นหา "Dart":** ในช่องค้นหา ให้พิมพ์คำว่า "Dart"
5. **ติดตั้ง Extension:** คุณจะเห็น Extension ชื่อ **"Dart"** โดยผู้พัฒนา "Dart Code" ซึ่งเป็น Extension อย่างเป็นทางการ ให้คลิกที่ปุ่ม Install
6. **ติดตั้ง Flutter (ถ้าจำเป็น):** หากคุณต้องการพัฒนา Flutter ด้วย VS Code การติดตั้ง **"Flutter"** Extension (โดยผู้พัฒนา "Dart Code") ก็จะติดตั้ง Dart Extension ให้โดยอัตโนมัติเช่นกัน
7. **ตรวจสอบการตั้งค่า:** หลังจากติดตั้งเสร็จสิ้น VS Code จะตรวจหา Dart SDK ที่คุณติดตั้งไว้ก่อนหน้านี้โดยอัตโนมัติ หากไม่พบ คุณอาจต้องกำหนดเส้นทาง (Path) ไปยังโฟลเดอร์ SDK ด้วยตนเอง (ใน settings.json) แต่ส่วนใหญ่แล้วจะทำงานได้ทันที

2. การติดตั้ง Dart Plugin ใน IntelliJ IDEA

IntelliJ IDEA เป็น IDE ที่มีประสิทธิภาพสูงและครบถ้วน เหมาะสำหรับโปรเจกต์ขนาดใหญ่ การติดตั้ง Dart plugin ก็ทำได้ไม่ยาก

ขั้นตอน:

1. **ติดตั้ง IntelliJ IDEA:** ดาวน์โหลดและติดตั้ง IntelliJ IDEA Community Edition ซึ่งเป็นเวอร์ชันฟรีจาก https://www.jetbrains.com/idea/download/
2. **เปิด IntelliJ IDEA:** เมื่อติดตั้งเสร็จแล้ว ให้เปิดโปรแกรมขึ้นมา
3. **ไปที่ส่วน Plugins:**
   - จากหน้าจอ Welcome ให้คลิกที่ **"Plugins"**
   - หรือหากอยู่ในโปรเจกต์อยู่แล้ว ให้ไปที่ File -> Settings... (หรือ IntelliJ IDEA -> Preferences... บน macOS) แล้วเลือก **"Plugins"**
4. **ค้นหา "Dart":** ในหน้าต่าง Plugins ให้เลือกแท็บ **"Marketplace"** และพิมพ์คำว่า "Dart" ในช่องค้นหา

5. **ติดตั้ง Plugin:** คุณจะเห็น Plugin ชื่อ **"Dart"** โดยผู้พัฒนา "JetBrains s.r.o." ซึ่งเป็นผู้พัฒนา IDE นี้เอง ให้คลิกที่ปุ่ม Install
6. **รีสตาร์ท IDE:** หลังจากติดตั้งเสร็จสิ้น โปรแกรมจะแจ้งให้คุณรีสตาร์ท IDE เพื่อให้การตั้งค่ามีผล
7. **ตั้งค่า Dart SDK Path:** เมื่อรีสตาร์ทแล้ว ในการสร้างโปรเจกต์ใหม่ (หรือใน Settings/Preferences -> Languages & Frameworks -> Dart) คุณอาจจะต้องกำหนดเส้นทางไปยัง Dart SDK ที่ติดตั้งไว้บนเครื่อง เพื่อให้ IntelliJ รู้ว่าต้องใช้ SDK ตัวไหนในการคอมไพล์โค้ด

การติดตั้งและตั้งค่าใน IDE ที่กล่าวมานี้จะช่วยให้คุณได้รับประโยชน์จากคุณสมบัติของ IDE อย่างเต็มที่ เช่น การเติมโค้ดอัตโนมัติ, การ Debugging, และการตรวจสอบโค้ดแบบเรียลไทม์

การรันโปรแกรม Dart (ฉบับเจาะลึก)

การรันโปรแกรม Dart หมายถึงการสั่งให้ **Dart Virtual Machine (VM)** หรือ **Dart Compiler** ทำงานกับโค้ด เพื่อประมวลผลและแสดงผลลัพธ์ออกมา มีสองวิธีหลักๆ ที่ต้องรู้:

1. การรันโปรแกรม Dart ผ่าน Command Line (Terminal)

นี่เป็นวิธีพื้นฐานที่สุดและเหมาะสำหรับโปรเจกต์ขนาดเล็ก, การเขียนสคริปต์, หรือการทดสอบโค้ดอย่างรวดเร็ว

หลักการทำงาน:

เมื่อคุณรันโปรแกรม Dart ผ่าน Terminal โดยใช้คำสั่ง dart run หรือ dart ตัว Dart VM จะทำงานในโหมด JIT (Just-In-Time) ซึ่งหมายความว่ามันจะคอมไพล์โค้ดในระหว่างการรัน ทำให้โค้ดทำงานได้ทันทีโดยไม่ต้องผ่านขั้นตอนการ build ที่ซับซ้อน

ขั้นตอนโดยละเอียด:

1. **สร้างไฟล์โปรแกรม Dart:**
   - สร้างไฟล์ชื่อ hello.dart และวางโค้ดต่อไปนี้ลงไป:

```Dart
void main() {
  print('Hello, Dart from Command Line!');
}
```

2. **เปิด Terminal / Command Prompt:**
   - บน Windows: เปิด Command Prompt หรือ PowerShell
   - บน macOS / Linux: เปิด Terminal
3. **ไปยังไดเรกทอรีของไฟล์:**
   - ใช้คำสั่ง cd (Change Directory) เพื่อเข้าไปยังโฟลเดอร์ที่คุณบันทึกไฟล์ hello.dart ไว้
   - เช่น: cd C:\Users\YourName\Documents\dart_projects

4. **รันโปรแกรม:**
   - พิมพ์คำสั่ง:

     Bash

     dart run hello.dart
   - หรือคำสั่งที่ง่ายกว่า (ถ้าไฟล์นั้นอยู่ในแพ็กเกจ):

     Bash

     dart hello.dart
5. **ดูผลลัพธ์:**
   - ผลลัพธ์ Hello, Dart from Command Line! จะแสดงขึ้นมาใน Terminal ทันที

ข้อดีของวิธีนี้:

- **ง่ายและรวดเร็ว:** ไม่ต้องเปิด IDE ที่ใช้ทรัพยากรสูง
- **เหมาะสำหรับ Automation:** ใช้ในสคริปต์อัตโนมัติ (Automated scripts) ได้ง่าย

---

2. การรันโปรแกรม Dart ผ่าน IDE (VS Code / IntelliJ)

การรันโปรแกรมใน IDE จะทำให้คุณได้รับประโยชน์จากฟีเจอร์ต่างๆ เช่น การ Debugging, การเติมโค้ดอัตโนมัติ (Code Completion) และการแสดงผลลัพธ์ในหน้าต่างเดียว

หลักการทำงาน:

เมื่อคุณกดปุ่ม Run ใน IDE ตัว IDE จะส่งคำสั่งไปยัง Dart VM ให้ทำงานกับไฟล์ที่คุณกำลังเปิดอยู่ และจะแสดงผลลัพธ์ในหน้าต่าง Debug Console หรือ Output ของ IDE นั้นๆ นอกจากนี้ IDE ยังรองรับโหมด Debugging ซึ่งช่วยให้คุณสามารถตรวจสอบค่าของตัวแปรในขณะที่โปรแกรมทำงานได้

ขั้นตอนโดยละเอียด (สำหรับ VS Code):

1. **เปิดไฟล์ใน VS Code:**
   - เปิด VS Code และเปิดโฟลเดอร์ที่มีไฟล์ hello.dart
2. **ไปที่ไฟล์ hello.dart:**
   - เปิดไฟล์ที่คุณต้องการรัน
3. **ใช้ปุ่ม Run หรือ Debug:**
   - ที่ด้านบนของฟังก์ชัน main() จะมีปุ่ม **Run** และ **Debug** ปรากฏขึ้นมา
   - คลิกที่ปุ่ม **Run**
4. **ดูผลลัพธ์ใน Output:**
   - VS Code จะเปิดหน้าต่าง **"Debug Console"** หรือ **"Output"** ขึ้นมาโดยอัตโนมัติ และแสดงผลลัพธ์ของโปรแกรม

ขั้นตอนโดยละเอียด (สำหรับ IntelliJ IDEA):

1. **เปิดไฟล์ใน IntelliJ:**
   - เปิดโปรเจกต์ของคุณและเปิดไฟล์ hello.dart

2. **ใช้ปุ่ม Run หรือ Run Configuration:**
   - **วิธีที่ 1 (เร็วที่สุด):** คลิกขวาที่ไฟล์ hello.dart ใน Project View แล้วเลือก **Run 'hello.dart'**
   - **วิธีที่ 2 (แบบมีตัวเลือก):** คลิกที่ปุ่ม **Run** (ไอคอนสามเหลี่ยมสีเขียว) ที่อยู่บนแถบเครื่องมือหลัก หรือที่ด้านบนของฟังก์ชัน main()
3. **ดูผลลัพธ์ใน Run Tool Window:**
   - IntelliJ จะเปิดหน้าต่าง **"Run"** ขึ้นมาที่ด้านล่างของหน้าจอ และแสดงผลลัพธ์การทำงานของโปรแกรม

ข้อดีของวิธีนี้:

- **Debug ได้ง่าย:** สามารถตั้ง **Breakpoint** เพื่อหยุดการทำงานของโปรแกรมและตรวจสอบค่าของตัวแปรได้
- **ครบถ้วนและสะดวก:** IDE จัดการการรันให้ทั้งหมด และมีเครื่องมือช่วยพัฒนาอื่นๆ ที่มีประสิทธิภาพ

โดยสรุปแล้ว การรันผ่าน **Command Line** เหมาะสำหรับการรันอย่างรวดเร็ว ส่วนการรันผ่าน **IDE** นั้นเหมาะสำหรับขั้นตอนการพัฒนาที่จริงจัง ซึ่งต้องการการดีบักและเครื่องมือช่วยอื่นๆ เพื่อให้งานสำเร็จลุล่วงครับ

ตัวอย่างโปรแกรมภาษา Dart

ตัวอย่างโปรแกรมภาษา Dart แบบเต็มไฟล์ให้ 3 โปรแกรม พร้อมคำอธิบายที่ครบถ้วนทั้งโครงสร้าง, การทำงานของโค้ด, และผลลัพธ์ที่ได้

โปรแกรมที่ 1: การจัดการข้อมูลพื้นฐาน

โปรแกรมนี้จะสาธิตการใช้ตัวแปร (Variables), ฟังก์ชัน (Functions) และการแสดงผลข้อมูลแบบพื้นฐานที่สุด

โครงสร้างไฟล์

- **ไฟล์:** user_info.dart

โค้ด

Dart

```dart
// user_info.dart

// ฟังก์ชันหลักที่โปรแกรมจะเริ่มทำงาน
void main() {
  // 1. ประกาศตัวแปรเพื่อเก็บข้อมูลผู้ใช้
  String name = 'Alice';
  int age = 30;
  double height = 165.5;
```

```
  bool isStudent = false;

  // 2. เรียกใช้ฟังก์ชันเพื่อพิมพ์ข้อมูล
  printUserInfo(name, age, height, isStudent);

  // 3. เรียกใช้ฟังก์ชันเพื่อคำนวณปีเกิด
  int birthYear = calculateBirthYear(age);
  print('Your estimated birth year is: $birthYear');
}

// ฟังก์ชันสำหรับพิมพ์ข้อมูลผู้ใช้
void printUserInfo(String name, int age, double height, bool isStudent) {
  print('--- User Information ---');
  print('Name: $name');
  print('Age: $age years old');
  print('Height: $height cm');
  print('Student Status: ${isStudent ? "Yes" : "No"}'); // ใช้ ternary operator
}

// ฟังก์ชันสำหรับคำนวณปีเกิด
int calculateBirthYear(int age) {
  int currentYear = DateTime.now().year;
  return currentYear - age;
}
```

คำอธิบายโค้ด

- main(): เป็นจุดเริ่มต้นของโปรแกรม
- **การประกาศตัวแปร:** เราใช้ String, int, double, และ bool เพื่อกำหนดชนิดข้อมูลให้ตัวแปรต่าง ๆ
- printUserInfo(): ฟังก์ชันนี้รับพารามิเตอร์ 4 ตัว แล้วนำมาพิมพ์ผลลัพธ์ในรูปแบบที่อ่านง่าย
- calculateBirthYear(): ฟังก์ชันนี้คำนวณปีเกิดโดยนำปีปัจจุบัน (DateTime.now().year) ลบด้วยอายุ
- $variable: นี่คือการใช้ **String Interpolation** เพื่อแทรกค่าของตัวแปรลงในข้อความได้อย่างง่ายดาย

ผลการรัน

```
--- User Information ---
Name: Alice
Age: 30 years old
```

Height: 165.5 cm

Student Status: No

Your estimated birth year is: 1995

(หมายเหตุ: ผลลัพธ์ปีเกิดอาจแตกต่างกันไปขึ้นอยู่กับปีที่รันโปรแกรม)

โปรแกรมที่ 2: การใช้งาน Class และ Object

โปรแกรมนี้จะสาธิตการสร้าง **Class** เพื่อเป็นพิมพ์เขียวสำหรับสร้าง **Object** ซึ่งเป็นหัวใจสำคัญของ Object-Oriented Programming

โครงสร้างไฟล์

- **ไฟล์:** car_model.dart

โค้ด

Dart

```dart
// car_model.dart

// Class ที่เป็นพิมพ์เขียวสำหรับ Object "Car"
class Car {
  // Properties หรือ Attributes ของ Object
  String brand;
  String model;
  int year;

  // Constructor (ตัวสร้าง Object)
  Car(this.brand, this.model, this.year);

  // Method (พฤติกรรมของ Object)
  void displayInfo() {
    print('--- Car Details ---');
    print('Brand: $brand');
    print('Model: $model');
    print('Year: $year');
  }

  // Method ที่มีค่าคืนกลับ (return)
  int getCarAge() {
```

```
    int currentYear = DateTime.now().year;
    return currentYear - year;
  }
}

// ฟังก์ชันหลัก
void main() {
  // 1. สร้าง Object จาก Class Car
  Car myCar = Car('Honda', 'Civic', 2020);

  // 2. เรียกใช้ Method ของ Object
  myCar.displayInfo();

  // 3. เรียกใช้ Method เพื่อรับค่า
  int carAge = myCar.getCarAge();
  print('This car is $carAge years old.');
}
```

คำอธิบายโค้ด

- class Car: เราสร้างคลาสชื่อ Car ซึ่งมีคุณสมบัติ (brand, model, year)
- Car(this.brand, this.model, this.year): นี่คือ **Constructor** ที่ใช้ในการสร้าง Object ของคลาส Car
- displayInfo(): คือ **Method** ที่กำหนดพฤติกรรมให้ Object
- myCar = Car(...): คือการสร้าง **Object** หรือ **Instance** ของคลาส Car
- myCar.displayInfo(): คือการเรียกใช้ Method จาก Object myCar

ผลการรัน

--- Car Details ---

Brand: Honda

Model: Civic

Year: 2020

This car is 5 years old.

(หมายเหตุ: ผลลัพธ์อายุรถอาจแตกต่างกันไปขึ้นอยู่กับปีที่รันโปรแกรม)

โปรแกรมที่ 3: การจัดการข้อมูลแบบ Asynchronous

โปรแกรมนี้จะสาธิตวิธีการทำงานแบบ **Asynchronous** ด้วย async และ await เพื่อจัดการกับงานที่ใช้เวลา เช่น การเรียกข้อมูลจากภายนอก

โครงสร้างไฟล์

- **ไฟล์:** async_example.dart

โค้ด

Dart

```
// async_example.dart

import 'dart:async'; // import ไลบรารีสำหรับ Future

// ฟังก์ชันที่จำลองการเรียกข้อมูลจาก API (ใช้เวลา 2 วินาที)
Future<String> fetchData() async {
  print('Start fetching data...');
  // await จะรอให้การทำงานใน Future.delayed เสร็จสิ้นก่อน
  await Future.delayed(Duration(seconds: 2));
  return 'Data fetched successfully!';
}

// ฟังก์ชันหลัก ที่ใช้ await
void main() async {
  print('Program started.');

  // 1. เรียกใช้ฟังก์ชัน fetchData() และรอผลลัพธ์
  // await จะทำให้โค้ดบรรทัดถัดไปทำงานก็ต่อเมื่อ fetchData เสร็จสิ้น
  String data = await fetchData();

  // 2. พิมพ์ผลลัพธ์ที่ได้หลังจากรอ 2 วินาที
  print(data);

  print('Program finished.');
}
```

คำอธิบายโค้ด

- Future<String> fetchData() async: ฟังก์ชันนี้เป็นแบบ **Asynchronous** ที่จะคืนค่าเป็น Future ซึ่งจะเก็บค่า String ในอนาคต
- await Future.delayed(...): คำสั่ง await จะบอกให้ Dart หยุดรอการทำงานของ Future.delayed (ที่จำลองการรอ 2 วินาที) ก่อนที่จะทำงานในบรรทัดถัดไป