แนวทางในการจัดกิจกรรมภาคปฏิบัติ

หัวข้อการเรียนรู้ :
เริ่มต้นใช้ Xcode เพื่อเป็นเครื่องมือในการพัฒนา iOS Application ด้วย SwiftUI

ระยะเวลาในการทำกิจกรรม: 360 นาที

วัตถุประสงค์ในการทำกิจกรรม :

1. การพัฒนาความรู้และความเข้าใจในการใช้ Xcode ในฐานะเครื่องมือเพื่อการพัฒนาแอปสำหรับอุปกรณ์ของ Apple
2. การใช้ Xcode Playground เพื่อการเขียนโปรแกรมภาษา Swift และการออกแบบอัลกอริทึม
3. การใช้ Xcode ในกระบวนการพัฒนาแอป ตั้งแต่การสร้างโปรเจกต์ไปจนถึงการทดสอบแอปบนอุปกรณ์เสมือน (Simulator) และอุปกรณ์จริง (Physical Device)
4. การใช้คุณสมบัติสำคัญของ Xcode เพื่อช่วยสนับสนุนการเขียน แก้ไข และทดสอบการทำงานของแอปอย่างเหมาะสม

ความคิดรวบยอด:
กิจกรรมการเรียนรู้ชุดนี้ มุ่งพัฒนาความรู้และทักษะพื้นฐานของผู้เรียนในการใช้ Xcode ในฐานะเครื่องมือสำหรับการเขียนโปรแกรมภาษา Swift และการพัฒนาแอปพลิเคชันสำหรับอุปกรณ์ของ Apple ด้วย SwiftUI โดยเน้นให้ผู้เรียนได้สำรวจและทดลองเพื่อฝึกใช้คุณสมบัติและเครื่องมือต่าง ๆ ที่มีอยู่ใน Xcode ตลอดกระบวนการออกแบบ พัฒนา ทดสอบ และปรับปรุงแอปพลิเคชันให้สามารถทำงานได้อย่างมีประสิทธิภาพ นอกจากนี้ ยังส่งเสริมการประยุกต์ใช้เทคโนโลยีปัญญาประดิษฐ์เพื่อสนับสนุนการเรียนรู้และการพัฒนาแอปในลักษณะของ AI-Assisted Development ผ่านคุณสมบัติ Coding Intelligence

บทความสำหรับอ่านประกอบการทำกิจกรรม
https://ajthiti.gitbook.io/develop-in-swift/meet-xcode

แนวทางในการจัดกิจกรรมการเรียนรู้

ขั้นนำเข้าสู่บทเรียน (Warm-up)

ในช่วงเริ่มต้นของบทเรียน ผู้สอนอาจเริ่มจาก**การตั้งคำถามเพื่อให้ผู้เรียนเริ่มคิดถึง “ที่มาของแอป”** ที่ใช้อยู่เป็นประจำ เช่น “ในโทรศัพท์ หรือ iPad ของเรา มีแอปใดที่เราใช้งานบ่อยที่สุด?” และ "คิดว่าแอปเหล่านี้ถูกสร้างขึ้นมาอย่างไร?” หรือ “ถ้าเราอยากสร้างแอปของตัวเองขึ้นมา เราต้องเริ่มจากอะไร ?” หลังจากนั้น ผู้สอนอาจอธิบายเชื่อมโยงต่อไปว่า **“แอปซึ่งทำงานบน iPhone, iPad และ Mac ถูกนักพัฒนาสร้างขึ้น โดยใช้เครื่องมือเฉพาะที่เรียกว่า “Xcode”**

ผู้สอนเปิดโปรแกรม Xcode และอธิบายว่า **Xcode เป็นศูนย์รวมของเครื่องมือสำหรับการเขียนโค้ด การออกแบบหน้าจอ การทดสอบการทำงานของแอป รวมทั้ง ยังมีความสามารถในการช่วยนักพัฒนาในการเขียนคำสั่ง การให้คำแนะนำเพื่อแก้ไขข้อผิดพลาดที่มีอยู่ในโปรแกรม ซึ่งช่วยให้การพัฒนาเป็นไปอย่างรวดเร็วและมีประสิทธิภาพ** บทเรียนนี้จะพาผู้เรียนไปรู้จัก Xcode ตั้งแต่การใช้เครื่องมือพื้นฐานไปจนถึงการใช้เทคโนโลยีปัญญาประดิษฐ์เพื่อช่วยในการพัฒนาแอป

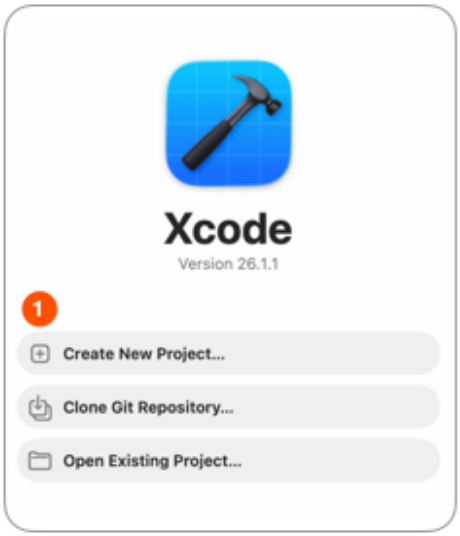
ขั้นการเรียนรู้ (Learn)

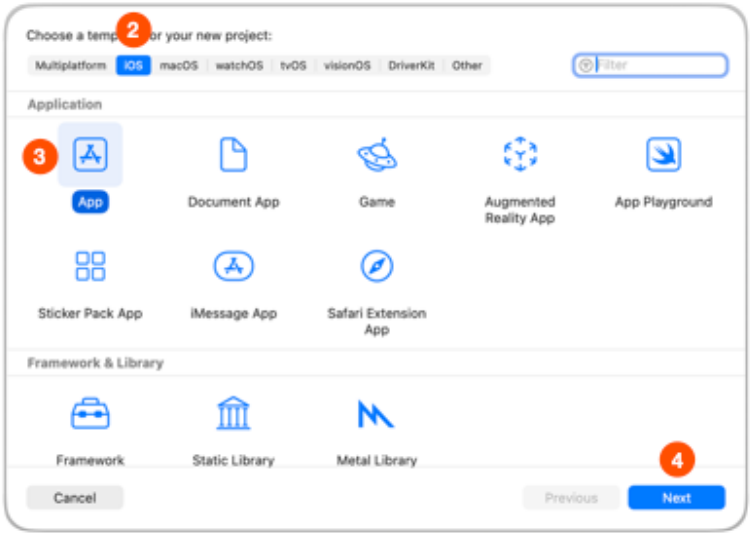
หลังจากผู้เรียนได้รับการกระตุ้นความสนใจในขั้นนำเข้าสู่บทเรียนแล้ว ในขั้นการจัดการเรียนรู้ (Learn) จะมุ่งเน้นการพัฒนาความรู้และทักษะของผู้เรียนผ่านกิจกรรมการเรียนรู้เชิงปฏิบัติอย่างเป็นลำดับ โดยการใช้ Xcode เป็นศูนย์กลางของการเรียนรู้ และค่อยๆ พาผู้เรียนเปลี่ยนบทบาทจาก “ผู้ใช้แอป” ไปสู่ “ผู้พัฒนาแอป”

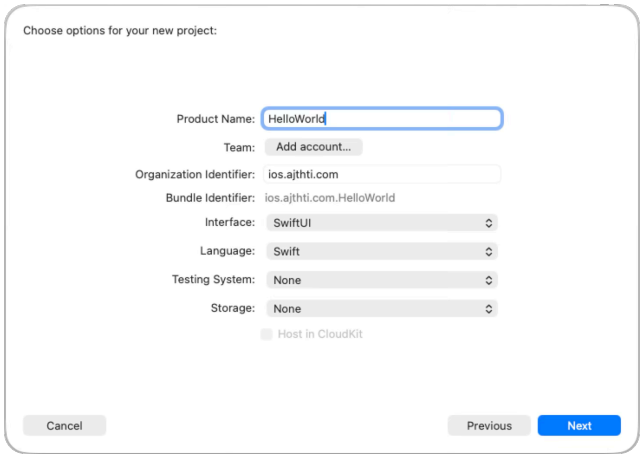
กิจกรรมที่ 1 การเรียนรู้เกี่ยวกับส่วนประกอบต่าง ๆ ของ Xcode

กิจกรรมแรกมีเป้าหมายเพื่อให้ผู้เรียนเข้าใจโครงสร้างและบทบาทของ Xcode ในฐานะเครื่องมือสำหรับการพัฒนาแอป หรือ **Integrated Development Environment (IDE)** ผู้เรียนจะได้สำรวจส่วนประกอบต่างๆ ของ Xcode เช่น Navigator Area, Editor Area, Inspector, Canvas และ Debug Area ซึ่งจะช่วยให้ผู้เรียนเห็นว่า Xcode ไม่ใช่เพียงโปรแกรมสำหรับพิมพ์โค้ด แต่เป็นสภาพแวดล้อมสำหรับการพัฒนาแอปแบบครบวงจร

ผู้สอนอาจเริ่มต้นกิจกรรมโดยแนะนำการใช้ macOS (สำหรับผู้เรียนที่ไม่เคยใช้ macOS มาก่อน) แล้วจึงแนะนำให้ผู้เรียนเปิด Xcode เพิ่มเริ่มต้นสร้างโปรเจกต์แรกโดยเลือก Template เป็น **iOS > App**



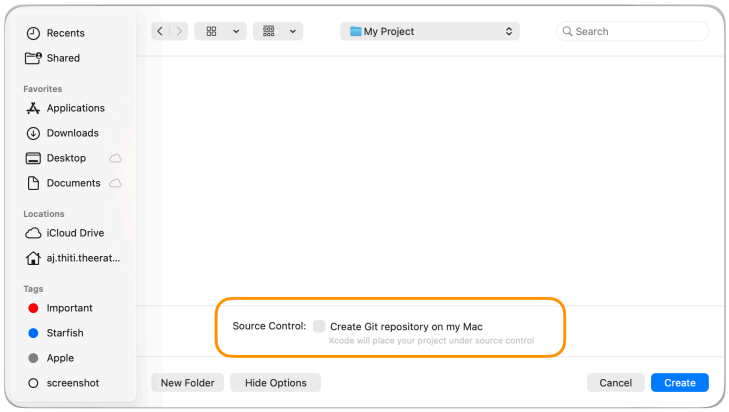




ตั้งชื่อ Project เป็น **HelloWorld** และกำหนดค่าต่างๆ ดังนี้

- กำหนด Organization Identifier เพื่อสร้าง Bundle Identifier ตามที่ต้องการ
- กำหนดค่า Interface เป็น **SwiftUI**
- กำหนดค่า Language เป็น **Swift**
- กำหนดค่า Testing System เป็น **None**
- กำหนดค่า Storage เป็น **None**

ผู้สอนอธิบายถึงความสำคัญของ **Bundle Identifier** และวิธีในการกำหนดค่า **Organization Identifier** ด้วยการใช้ **รูปแบบโดเมนย้อนกลับ (Reverse-DNS)** เช่น com.ajthiti.ios เพื่อให้มั่นใจได้ว่า Bundle Identifier ของแอปที่ถูกสร้างขึ้นจะมีความเป็นเอกลักษณ์ ไม่ซ้ำกัน และเป็นไปตามมาตรฐานสำหรับการเผยแพร่แอปบน App Store

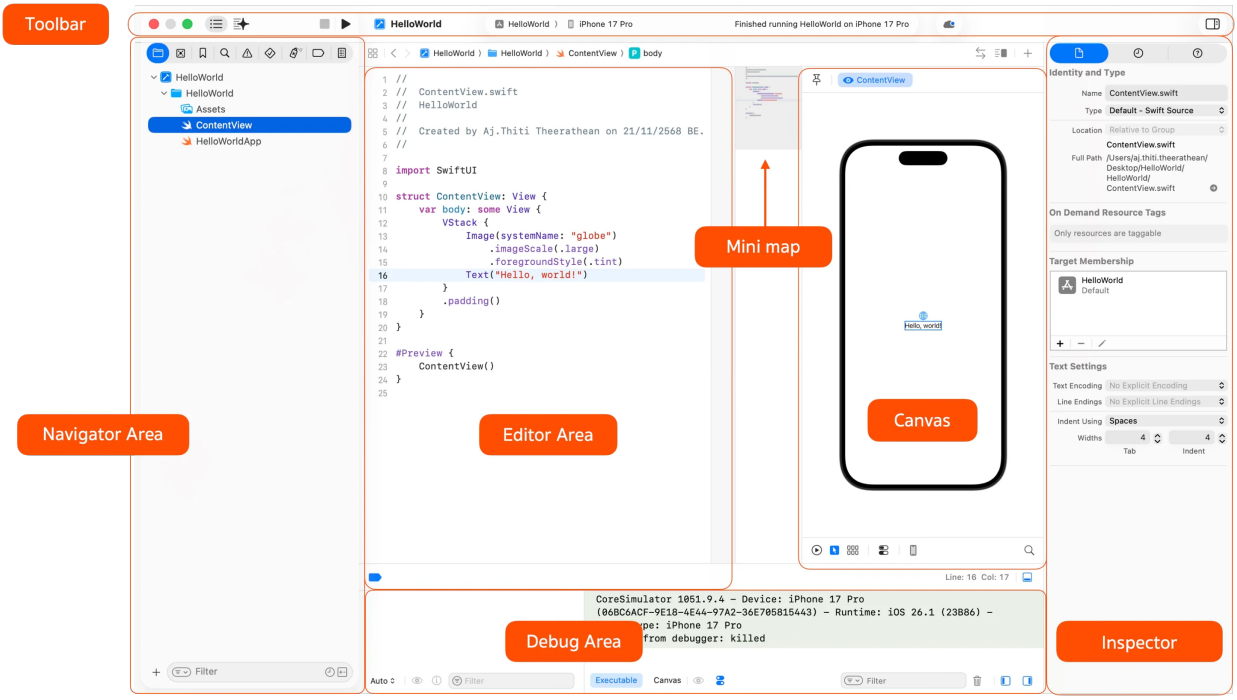


กำหนดตำแหน่งในการจัดเก็บโปรเจกต์

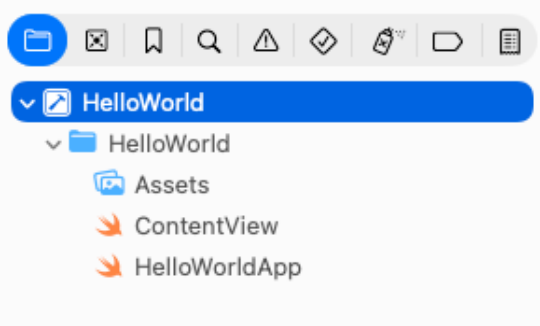
คลิกเพื่อเอาเครื่องหมายถูกที่ช่อง **Source Control** ออกเพื่อยกเลิกการใช้ระบบจัดการเวอร์ชัน (Version Control)

คลิกที่ปุ่ม Create เพื่อสร้างโปรเจกต์

เมื่อสร้างโปรเจกต์ใน Xcode เรียบร้อยแล้ว **ให้คลิกที่ไฟล์ ContentView.swift** เพื่อเปิดส่วนประกอบหลักที่จะถูกใช้ร่วมกันเพื่อสนับสนุนการพัฒนาแอปทั้งด้านการเขียนโค้ด การออกแบบส่วนติดต่อกับผู้ใช้ (UI) และการทดสอบโปรแกรม โดยผู้สอนอธิบายถึงส่วนประกอบต่างๆ บนหน้าจอของ Xcode ดังนี้



- **Toolbar :** แถบเครื่องมือซึ่งอยู่ด้านบนสุดของหน้าต่าง Xcode ประกอบด้วยปุ่มและเมนูเพื่อเรียกผู้ช่วย AI และส่วน Run Distination เพื่อกำหนดเป้าหมายในการรันแอปเพื่อทดสอบการทำงาน
- **Navigator Area :** พื้นที่ด้านซ้ายของ Xcode ใช้สำหรับเลือกและจัดการไฟล์ที่อยู่ในโปรเจกต์
- **Editor Area :** พื้นที่สำหรับการเขียนโค้ด
- **Code mini map :** ภาพรวมของโค้ด (Code Overview) ที่อยู่ด้านขวาของ Editor Area ซึ่งช่วยให้นักพัฒนามองเห็นโครงสร้างโค้ดทั้งหมดและเพิ่มประสิทธิภาพในการนำทางไปยังส่วนต่างๆ ภายในไฟล์
- **Canvas (SwiftUI Preview) :** พื้นที่แสดงผลลัพธ์ของโค้ดแบบเรียลไทม์ (Live Preview)
- **Debug Area :** พื้นที่ด้านล่างใช้สำหรับแสดง Log การทำงานของแอป ซึ่งจะปรากฏขึ้นมาเมื่อรันโปรแกรม โดยนักพัฒนาสามารใช้เครื่องมือในส่วนนี้เพื่อตรวจสอบและวินิจฉัยข้อผิดพลาดในการทำงานของโปรแกรม
- **Inspector :** พื้นที่ด้านขวาสุด ใช้เพื่อตั้งค่าคุณสมบัติของไฟล์ หรือ UI Components โดยหากต้องการซ่อนเพื่อเพิ่มพื้นที่ในการเขียนโค้ด ให้คลิกปุ่ม “Hide or show the Inspectors” ที่มุมขวาบนของ Toolbar

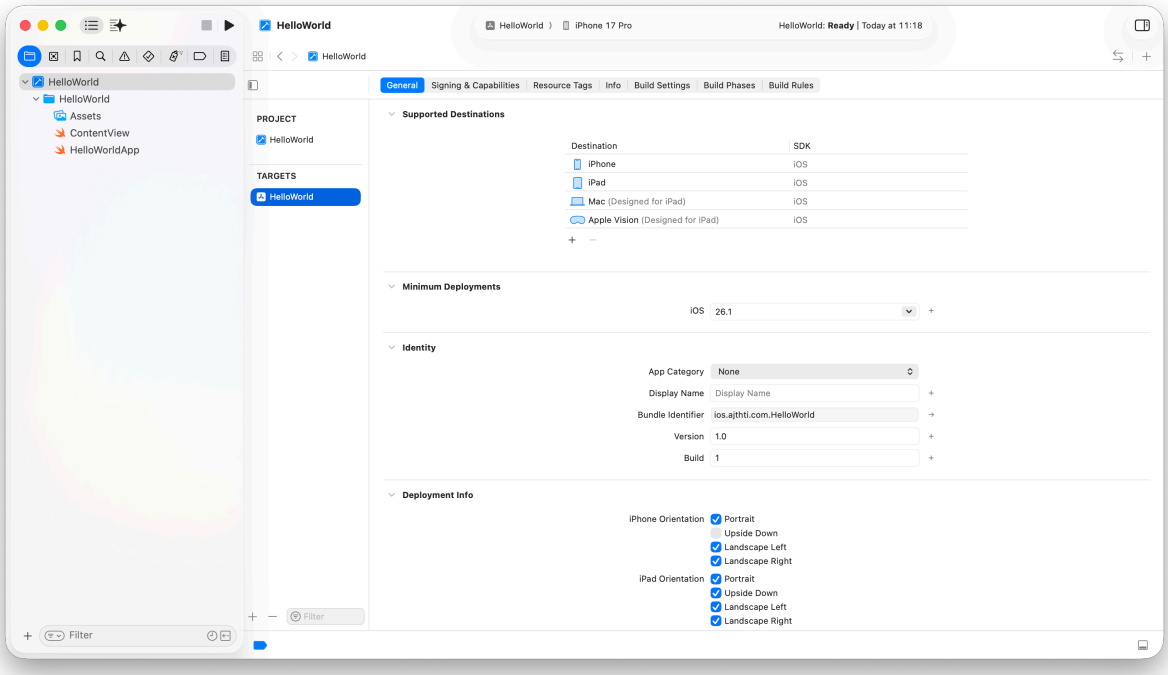


กิจกรรมที่ 2 การสำรวจไฟล์ต่างๆ ในโปรเจกต์

เริ่มการสำรวจไฟล์ต่างๆ ที่อยู่ในโปรเจกต์ เช่น Project File, Swift File และ Assets Cataloge เพื่อทำความเข้าใจเกี่ยวกับโครงสร้างของแอป

การตั้งค่าของแอปที่ Project File

Project File เป็นส่วนที่ใช้เพื่อแสดงหน้าต่างการตั้งค่าคุณสมบัติของแอปที่ถูกกำหนดไว้ตอนสร้างโปรเจกต์ รวมทั้งการกำหนดคุณสมบัติเพิ่มเติมซึ่งเป็นส่วนสำคัญที่ใช้ในการ Build และเผยแพร่แอปไปยัง App Store



เพื่อกำหนดค่าในการ Build ผลลัพธ์ ให้ผู้เรียนคลิกที่ **ชื่อแอป (HelloWorld)** ในส่วน **Targets** และเลือกที่ **แถบ Ganeral** เพื่อกำหนดรายละเอียดของค่าต่างๆ ในการระบุว่า แอปที่สร้างขึ้นจะสามารถทำงานบนอุปกรณ์ประเภทใดได้บ้าง การกำหนดเวอร์ชันขั้นต่ำของระบบปฏิบัติการที่รองรับ และการกำหนดประเภท หมายเลข Version และ Build ของแอปสำหรับการเผยแพร่บน App Store รวมทั้งการกำหนดรายละเอียดของการแสดงผลและพฤติกรรมของแอปบนอุปกรณ์ปลายทาง

โครงสร้างของแอปที่สร้างด้วย SwiftUI

เมื่อสร้างโปรเจกต์ด้วย SwiftUI ผู้เรียนจะพบว่า Xcode ได้สร้างโพลเดอร์สำหรับจัดเก็บไฟล์และโครงสร้างพื้นฐานไว้ให้แล้ว ซึ่งสามารถแบ่งโครงสร้างพื้นฐานของแอปออกได้เป็น 3 ส่วนหลัก คือ

ส่วนเริ่มต้นของแอป -> ไฟล์ HelloWorldApp.swift

ไฟล์ HelloWorldApp.swift ทำหน้าที่เป็น จุดเริ่มต้นของแอปพลิเคชัน (Application Entry Point) ในสถาปัตยกรรมของ SwiftUI โดยเป็นไฟล์ที่กำหนดโครงสร้างระดับแอป (Application Structure) และระบุฉากการแสดงผล (Scene) ที่แอปจะใช้เมื่อเริ่มทำงาน ตัวอย่างโค้ดพื้นฐานมีดังนี้

```swift
import SwiftUI

@main
struct HelloWorldApp: App {
    var body: some Scene {
        WindowGroup {
            ContentView()
        }
    }
}
```

คีย์เวิร์ด **@main** เป็น Attribute ที่ใช้ระบุว่า โครงสร้าง (struct) นี้คือ **จุดเริ่มต้นของโปรแกรม** เมื่อแอปถูกเรียกใช้งาน ระบบจะเริ่มต้นการทำงานจากโค้ดในไฟล์นี้ทันที

คำสั่ง **struct** HelloWorldApp: App เป็นการประกาศว่า HelloWorldApp มีความสอดคล้องกับ App protocol ซึ่งเป็นโปรโตคอลหลักของ SwiftUI สำหรับกำหนดโครงสร้างระดับแอปพลิเคชัน App protocol มีหน้าที่สำคัญดังนี้

- กำหนด Lifecycle ของแอป
- สร้าง Environment เริ่มต้นของแอป
- ระบุ Scene ที่ใช้สำหรับแสดงผล

เมื่อแอปเริ่มทำงาน SwiftUI จะอ่านค่าที่กำหนดไว้ภายใน **var** body: **some** Scene เพื่อสร้างสภาพแวดล้อมของแอป (App Environment) และกำหนดฉากการแสดงผล (Scene) ที่แอปจะต้องใช้ขึ้นมา

ฉากการแสดงผล (Scene) ทำหน้าที่เป็น**ตัวแทนของพื้นที่แสดงผล (Presentation Space)** ซึ่งกำหนดลักษณะของสภาพแวดล้อมที่ใช้แสดง View สำหรับการโต้ตอบกับผู้ใช้ โดยหนึ่งแอปสามารถประกอบด้วย Scene ได้มากกว่าหนึ่งรูปแบบ ทั้งนี้แต่ละ Scene จะถูกออกแบบมาเพื่อรองรับบริบทการใช้งานที่แตกต่างกัน ใน SwiftUI มีรูปแบบของฉากการแสดงผล หรือ Scene ให้เลือกใช้งานหลายประเภท เช่น

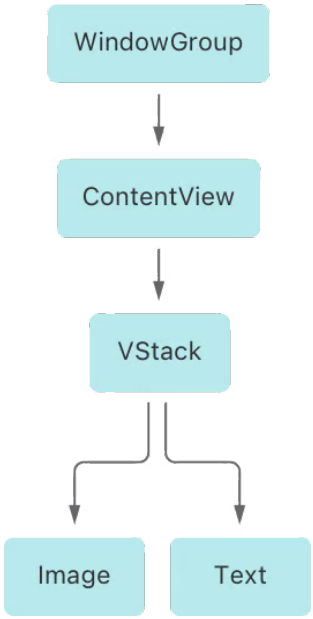
- **WindowGroup** สำหรับจัดการหน้าต่างหรือหน้าจอหลักของแอป
- **DocumentGroup** สำหรับแอปที่ทำงานกับเอกสารหลายไฟล์
- **Settings** สำหรับสร้างหน้าการตั้งค่าของแอป
- **ImmersiveSpace** สำหรับแอปเชิงประสบการณ์เสมือน หรือ Spatial Computing

ภายใน WindowGroup ของฉากการแสดงผล นักพัฒนาจะกำหนด **View หลัก (Root View)** ของแอป ซึ่งมักเป็นไฟล์ที่ชื่อว่า ContentView โดย ContentView จะทำหน้าที่เป็นจุดเริ่มต้นของโครงสร้างหน้าจอทั้งหมดในแอป เมื่อหน้าต่างถูกสร้างขึ้น SwiftUI จะนำ ContentView มาแสดงเป็นหน้าจอแรก

ส่วนการออกแบบส่วนติดต่อผู้ใช้ -> ไฟล์ ContentView.swift
ไฟล์ ContentView.swift เป็นไฟล์หลักที่ใช้เพื่อเริ่มต้นการสร้างส่วนติดต่อกับผู้ใช้ (UI) ของแอป โดยเป็นโครงสร้างที่มีความสอดคล้องกับ**แนวคิด Declarative UI** ซึ่งนักพัฒนาจะอธิบายว่า **“หน้าจอควรมีลักษณะอย่างไร”** มากกว่าการสั่งขั้นตอนการวาด UI ทีละลำดับ

SwiftUI จะอ่านโครงสร้างหน้าจอ (View hierarchy) จาก body และทำหน้าที่วาด UI รวมถึงจัดการการปรับปรุงหน้าจอให้โดยอัตโนมัติ โดยทั่วไปโค้ดพื้นฐานของ ContentView มักมีรูปแบบ ดังนี้

```swift
import SwiftUI

struct ContentView: View {

    var body: some View {

        VStack {

            Image(systemName: "globe")
                .imageScale(.large)
                .foregroundStyle(.tint)

            Text("Hello, world!")

        }
        .padding()

    }
}
```

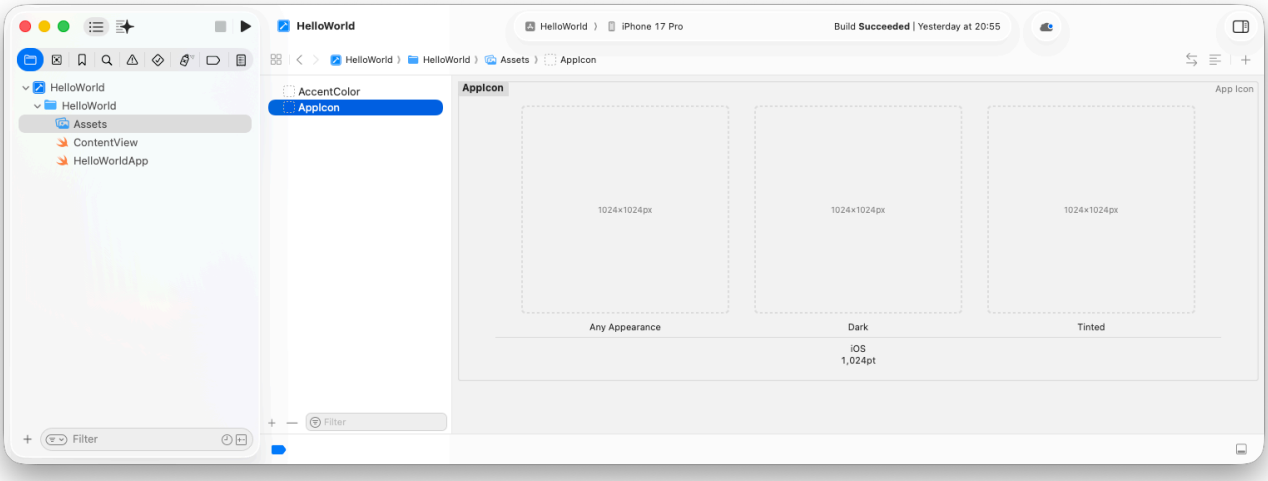


ContentView จะมีโครงสร้างที่กำหนดให้สอดคล้องกับ View protocol ซึ่งเป็นหน่วยพื้นฐานที่สุดในการสร้างส่วนติดต่อกับผู้ใช้ด้วย SwiftUI ซึ่งจะถูกใช้เพื่ออธิบายว่า View ที่จะแสดงขึ้นมาจะประกอบไปด้วยสิ่งใดบ้างและจะต้องมีการจัดวางในตำแหน่งต่างๆ บนจอภาพอย่างไร โดยการนิยามโครงสร้างของหน้าจอในลักษณะ **Tree Structure** หรือที่เรียกว่า View Hierarchy ลงในส่วน body โดย SwiftUI จะอ่านโครงสร้างนี้จากบนลงล่าง และแปลงเป็นส่วนติดต่อกับผู้ใช้จริงบนหน้าจอ

ในการกำหนดตำแหน่งและลักษณะของการจัดวาง UI Components บนจอภาพ เราจะใช้ **Layout Containers** เช่น VStack, HStack หรือ ZStack เพื่อทำหน้าที่จัดเรียงองค์ประกอบตามที่กำหนด โดย Layout เหล่านี้ไม่ใช่ส่วนที่แสดงผลโดยตรง แต่เป็นตัวกำหนดโครงสร้างและตำแหน่งขององค์ประกอบบนหน้าจอ

การแสดงผลในระดับลึกที่สุดของ View Hierarchy คือ **Leaf Views** ซึ่งเป็นกำหนดองค์ประกอบที่ผู้ใช้มองเห็นจริงบนหน้าจอ เช่น Text, Image, Button ซึ่ง View เหล่านี้จะถูกจัดวางอยู่ภายใน Layout Containers ตามที่กำหนดไว้ (ผู้เรียนจะได้ศึกษาเกี่ยวกับวิธีในการสร้างส่วนติดต่อกับผู้ใช้แบบ Declarative ด้วย SwiftUI โดยละเอียดในบทเรียนเรื่อง View และ Modifier)

ส่วนการจัดการทรัพยากร -> ไฟล์ Assets.xcassets

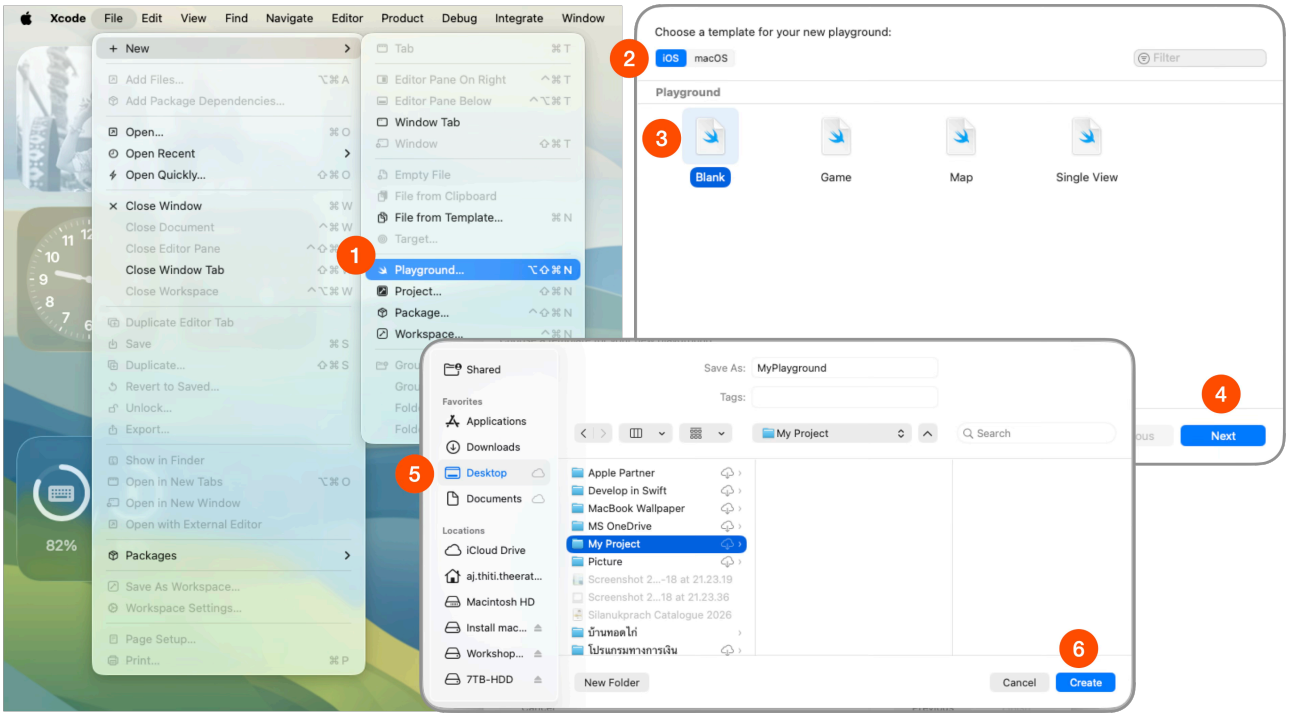


ไฟล์ Asset Catalog เป็นคลังสื่อหรือพื้นที่จัดเก็บ “ทรัพยากร” (Resources) ของแอปในรูปแบบที่ Xcode และระบบปฏิบัติการของ Apple สามารถจัดการให้เหมาะสมได้โดยอัตโนมัติ โดยทรัพยากรที่จะถูกเก็บใน Asset Catalog ได้แก่ ไอคอนของแอป รูปภาพ และชุดสี

กิจกรรมที่ 3 การใช้ Playground เพื่อเรียนรู้การเขียนโปรแกรมภาษา Swift
กิจกรรมในส่วนนี้ มีเป้าหมายเพื่อให้ผู้เรียนสามารถใช้ Playground ซึ่งอยู่ใน Xcode เพื่อทดลองเขียนโปรแกรมภาษา Swift ในลักษณะของการเรียนรู้แบบทดลอง (Experimentation) ผู้เรียนจะได้ฝึกสร้างคำสั่งและอัลกอริทึม เช่น การเก็บค่าในตัวแปร การใช้คำสั่งแบบกำหนดเงื่อนไขและการวนซ้ำ การสร้างขั้นตอนวิธีเพื่อการแก้ไขปัญหา พร้อมทั้งเห็นผลลัพธ์ของโค้ดแบบทันที กิจกรรมนี้ทำหน้าที่เป็นสะพานเชื่อมระหว่างแนวคิดการเขียนโปรแกรมกับการพัฒนาแอปจริง ช่วยให้ผู้เรียนเกิดความมั่นใจก่อนการสร้างโปรเจกต์เพื่อการพัฒนาแอปด้วย SwiftUI

Playground คือ สภาพแวดล้อมสำหรับทดลองเขียนและรันโค้ดภาษา Swift แบบโต้ตอบ ซึ่งนักพัฒนาสามารถใช้ในการสร้างคำสั่งและการทดสอบโค้ดโดยไม่จำเป็นต้องสร้างโปรเจกต์แบบเต็มรูป ในการจัดการเรียนการสอนสำหรับผู้ที่ยังไม่เคยมีพื้นฐานการเขียนโปรแกรมภาษา Swift มาก่อน ผู้สอนอาจให้ผู้เรียนใช้ Playground ร่วมกับการศึกษาเนื้อหาเรื่อง Swift Programming โดยให้ผู้เรียนศึกษาและทดลองเขียนคำสั่งต่างๆ ตามเนื้อหาซึ่งเผยแพร่ในรูปแบบเอกสารออนไลน์ที่เข้าถึงได้จาก https://ajthiti.gitbook.io/swift

เริ่มต้นสร้างไฟล์ Playground เพื่อใช้ในการทดลองเขียนโปรแกรมภาษา Swift

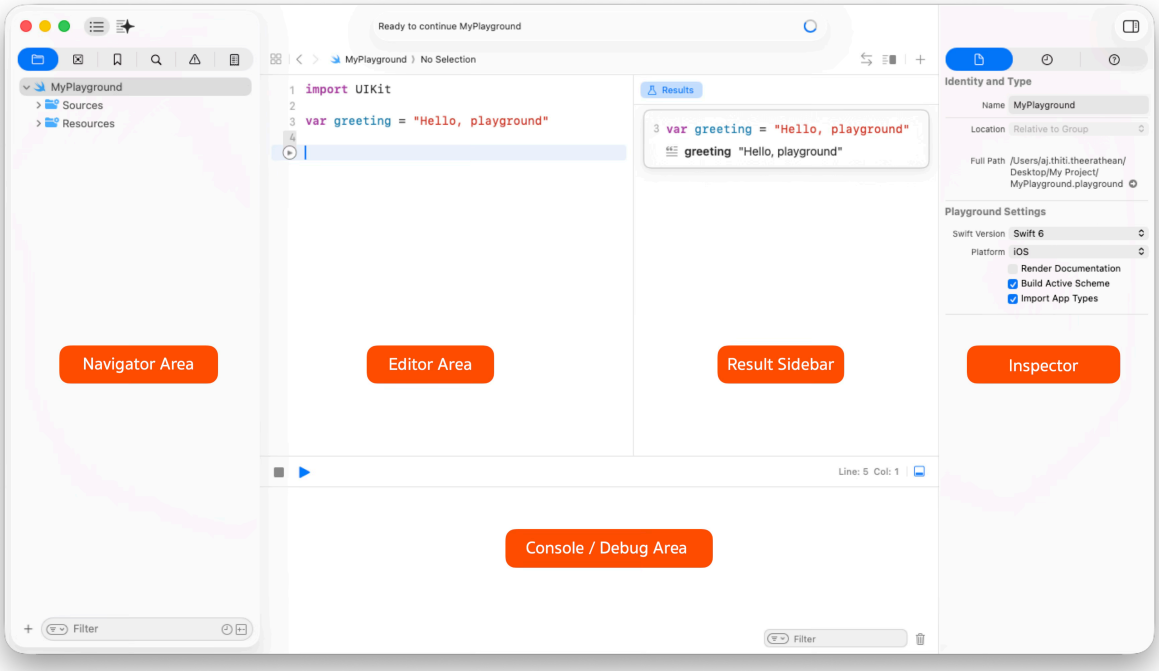


1. เปิดโปรแกรม Xcode เลือกเมนู **File -> New -> Playground**
2. เลือกแพลตฟอร์มเป็น **iOS**
3. เลือกรูปแบบ Playground (Template) เป็น **Blank**
4. คลิกที่ **ปุ่ม Next**
5. ตั้งชื่อไฟล์และเลือกตำแหน่งจัดเก็บ เช่น Desktop > My Project
6. คลิกที่ **ปุ่ม Create**

เมื่อสร้าง Playground สำเร็จ จะพบโครงสร้างของไฟล์ที่เรียบง่าย โดยทั่วไปจะประกอบด้วยโค้ดเริ่มต้น ดังนี้

```
import UIKit
var greeting = "Hello, playground"
```

และผู้เรียนจะเห็นส่วนประกอบหน้าจอสำหรับการทำงานของ Playground ซึ่งแบ่งออกเป็นส่วนๆ ดังนี้



Navigator Area : ทำหน้าที่แสดงโครงสร้างของไฟล์ Playground โดยในภาพจะเห็นองค์ประกอบหลัก ได้แก่

- Playground File (MyPlayground) เป็นไฟล์หลักสำหรับเขียนและทดลองโค้ดภาษา Swift
- Sources ใช้สำหรับเก็บโค้ดที่ต้องการนำมาใช้ซ้ำ
- Resources ใช้สำหรับจัดเก็บไฟล์ทรัพยากร เช่น รูปภาพ ไฟล์ข้อมูล หรือสื่ออื่นๆ

Editor Area : เป็นพื้นที่หลักสำหรับเขียนคำสั่งภาษา Swift
Result Sidebar : เป็นส่วนที่แสดงผลลัพธ์ของโค้ดที่เขียนใน Playground แบบทันที (Immediate Feedback)
Inspector Area : ใช้สำหรับกำหนดคุณสมบัติและการตั้งค่าของ Playground
Console/Debug Area : ใช้แสดงผลลัพธ์หรือสถานะการทำงานของ Playground ระหว่างการรันโค้ด

เพื่อศึกษาการเขียนโปรแกรมด้วยภาษา Swift ผู้สอนอาจให้ผู้เรียนใช้ Playgound เพื่อเรียนรู้วิธีการเขียนโปรแกรมด้วยภาษา Swift ในหัวข้อต่างๆ ดังต่อไปนี้

- ตัวแปร ค่าคงที่ และประเภทของข้อมูล
- การจัดการกับตัวแปรที่ยังไม่ได้กำหนดค่าด้วย Optional
- ตัวดำเนินการประเภทต่างๆ
- การจัดเก็บข้อมูลแบบ Collection
- การใช้คำสั่งแบบเงื่อนไขและคำสั่งทำซ้ำ
- การสร้างฟังก์ชันและการใช้ Closure
- การเขียนโปรแกรมเชิงวัตถุ

โดยผู้เรียนสามารถศึกษาเนื้อหาเพิ่มเติมด้วยตนเองได้จากเว็บไซต์ https://ajthiti.gitbook.io/swift

หลังจากนั้นอาจให้ผู้เรียนทดลองเขียนคำสั่งในภาษา Swift เพื่อออกแบบขั้นตอนวิธี (Algorithm Design) ในการแก้ไขปัญหา เช่น การจัดเรียนข้อมูลในอาร์เรย์ และสังเกตผลลัพธ์ที่เกิดขึ้นในส่วน Result Sidebar

```
import Foundation

var numbers = [42, 18, 30, 15, 50]

for i in 0..<numbers.count {
    for j in 0..<numbers.count - i - 1 {
        if numbers[j] > numbers[j + 1] {
            let temp = numbers[j]
            numbers[j] = numbers[j + 1]
            numbers[j + 1] = temp
        }
    }
}

print(numbers)
```

กิจกรรมที่ 4 คุณสมบัติสำคัญของ Xcode เพื่อสนับสนุนการเขียนและแก้ไขโค้ด
กิจกรรมในส่วนนี้ มีเป้าหมายเพื่อส่งเสริมให้ผู้เรียนเรียนรู้เกี่ยวกับการใช้คุณสมบัติสำคัญของ Xcode เช่น Syntax Highlighting, Code Completion รวมถึงการวิเคราะห์ Errors และ Warnings เพื่อฝึกแก้ไขข้อผิดพลาดและเรียนรู้ว่า ข้อผิดพลาดเป็นส่วนหนึ่งของกระบวนการพัฒนาแอป

คุณสมบัติที่ 1: การมองเห็นโครงสร้างของโค้ดด้วย Syntax Highlighting
หนึ่งในคุณสมบัติที่มีบทบาทสำคัญต่อคุณภาพในการพัฒนาโปรแกรมใน Xcode คือ Syntax Highlighting ซึ่งไม่ใช่แค่การตกแต่งโค้ดให้มีสีสันสวยงาม แต่เป็นกลไกเชิงโครงสร้างที่ช่วยจำแนกองค์ประกอบของภาษาออกจากกันอย่างชัดเจน ส่งผลให้ผู้พัฒนาสามารถมองเห็นทั้ง ความหมาย (Semantics) และ ความสัมพันธ์ของสิ่งต่างๆ ในโค้ดได้ทันที

ผู้สอนอาจเริ่มต้นกิจกรรมโดยใช้โค้ดการจัดเรียนข้อมูลในอาร์เรย์ และสอบถามผู้เรียนว่า “ผู้เรียนสามารถบอกได้หรือไม่ว่า สีของคำที่แสดงในโค้ด มีประโยชน์อย่างไร ? ”

```
import Foundation

var numbers = [42, 18, 30, 15, 50]

for i in 0..<numbers.count {
    for j in 0..<numbers.count - i - 1 {
        if numbers[j] > numbers[j + 1] {
            let temp = numbers[j]
            numbers[j] = numbers[j + 1]
            numbers[j + 1] = temp
        }
    }
}

print(numbers)
```

หลังจากนั้น ผู้สอนจึงเริ่มอธิบายและชี้ให้ผู้เรียนเห็นถึงประโยชน์ของคุณสมบัติ Syntax Highlighting ซึ่งเป็นกลไกที่เปลี่ยนโค้ดจาก “ข้อความธรรมดา” ให้กลายเป็น **โครงสร้างที่มองเห็นได้** โดยให้ผู้เรียนฝึกการจำแนกองค์ประกอบของภาษา เช่น คำสงวน ตัวแปร และค่าของข้อมูล โดยสังเกตจากรูปแบบและสีที่ถูกแสดงขึ้นมาในโค้ด

สำหรับนักพัฒนา คุณสมบัติ Syntax Highlighting ยังมีบทบาทสำคัญต่อการอ่านและทำความเข้าใจโครงสร้างของคำสั่งในภาษาที่ไม่คุ้นเคย ตัวอย่างเช่น

```
func calculateGrade(score: Int) -> String {
    switch score {
    case 90...100:
        return "A"
    case 80..<90:
        return "B"
    case 70..<80:
        return "C"
    default:
        return "F"
    }
}
```

การเน้นสีด้วย Syntax Highlighting ช่วยให้ นักพัฒนาแยกได้ทันทีว่า

- ส่วนใดคือ โครงสร้างควบคุม (switch–case)
- ส่วนใดคือ ช่วงข้อมูล (Range)
- ส่วนใดคือ ค่าที่ส่งกลับ (Return Value)

คุณสมบัติที่ 2: การช่วยเขียนโค้ดอย่างชาญฉลาดด้วย Code Completion
นอกจากการมองเห็นโครงสร้างของโค้ดด้วย Syntax Highlighting แล้ว อีกหนึ่งคุณสมบัติสำคัญของ Xcode ที่มีผลโดยตรงต่อประสิทธิภาพและคุณภาพของการพัฒนาโปรแกรม คือ Code Completion ซึ่งทำหน้าที่เป็น “ผู้ช่วยอัจฉริยะ (Intelligent Assistant)” ที่สนับสนุนการเขียนโค้ดของนักพัฒนาแบบเรียลไทม์

เพื่อให้ผู้เรียนเข้าใจบทบาทของ Code Completion อย่างเป็นรูปธรรม ผู้สอนสามารถเริ่มต้นด้วยการกำหนด **โจทย์เชิงสถานการณ์** ที่ใกล้ตัวและเข้าใจง่ายเพื่อให้ผู้เรียนออกแบบขั้นตอนวิธี เช่น การคำนวณ **ค่าดัชนีมวลกาย (Body Mass Index: BMI)** และให้นำขั้นตอนวิธีดังกล่าวมาทดลองเขียนคำสั่งใน **Xcode Playground** โดยระหว่างการพิมพ์คำสั่ง ผู้สอนควรเน้นย้ำเพื่อให้ผู้เรียนสังเกตอย่างตั้งใจว่า Xcode แสดงคำแนะนำใดขึ้นมาในแต่ละช่วงของการพิมพ์คำสั่ง

ตัวอย่างของโปรแกรมสำหรับการคำนวณ **ค่าดัชนีมวลกาย (Body Mass Index: BMI)**

```
import Foundation

let h: Int = 179
let w: Int = 80

func bmi(height: Int, weight: Int) -> Double {
    return Double(weight) / pow(Double(height) / 100, 2)
}

print(bmi(height: h, weight: w))
```

ผู้สอนอธิบายและชี้ให้ผู้เรียนเห็นว่า Code Completion ไม่ได้เป็นเพียงการเติมคำอัตโนมัติ (Auto-complete) แต่เป็นกลไกที่อาศัยการวิเคราะห์บริบทของโค้ด (Context-Aware Analysis) เพื่อแนะนำคำสั่งที่ถูกต้อง เหมาะสม และสอดคล้องกับสถานการณ์ รวมทั้งยังเป็นหัวใจสำคัญของการพัฒนาโปรแกรมใน Xcode ที่เปลี่ยนกระบวนการเขียนโค้ดจาก **“การพิมพ์คำสั่งตามความจำ”** เป็น **“การออกแบบโค้ดโดยอาศัยความเข้าใจเชิงโครงสร้างและบริบท”**

คุณสมบัติที่ 3: ระบบการตรวจสอบข้อผิดพลาดในโค้ดด้วย Compiler Errors & Warnings
โดยทั่วไปตัวแปรภาษาแบบคอมไพเลอร์ (Compiler) ใน Xcode จะทำหน้าที่แปลโค้ดจากภาษาที่มนุษย์เขียน (Swift) ให้อยู่ในรูปแบบที่ระบบสามารถประมวลผลได้ โดยระหว่างกระบวนการดังกล่าวคอมไพเลอร์จะทำ Static Analysis เพื่อตรวจสอบว่า

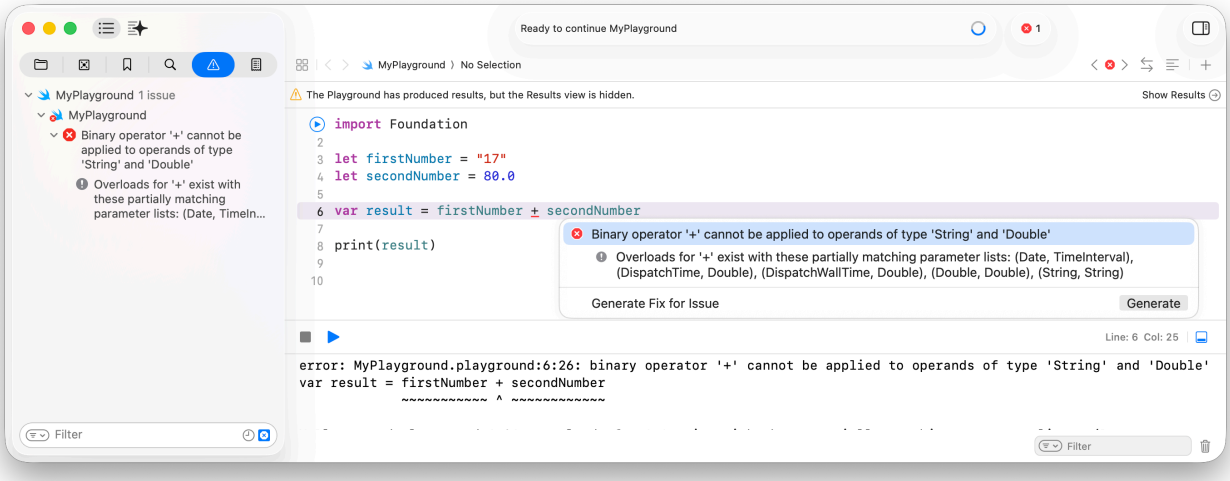
- โค้ดถูกต้องตามไวยากรณ์หรือไม่ (Syntax)
- ชนิดข้อมูลสอดคล้องกันหรือไม่ (Type Safety)
- มีโค้ดที่เสี่ยงต่อความผิดพลาดหรือคุณภาพต่ำหรือไม่

โดยผลลัพธ์ของการตรวจสอบจะถูกแสดงออกมาในรูปแบบของ**ข้อผิดพลาด (Error)** และ**คำเตือน (Warning)** จากคอมไพเลอร์ ซึ่งเป็นกลไกการสะท้อนกลับ (Feedback) ซึ่งช่วยให้นักพัฒนามองเห็นข้อบกพร่องของโค้ด รวมทั้งช่วยให้คำแนะนำในการแก้ไขให้โค้ดมีความถูกต้อง (Correctness) มีคุณภาพ (Quality) และง่ายต่อการดูแลรักษา (Maintainability) ในอนาคต

เพื่อทดลองใช้ระบบการตรวจสอบข้อผิดพลาดในโค้ด ให้ผู้เรียนพิมพ์โค้ดต่อไปนี้

```swift
import Foundation

let firstNumber = "17"
let secondNumber = 80.0

var result = firstNumber + secondNumber

print(result)
```

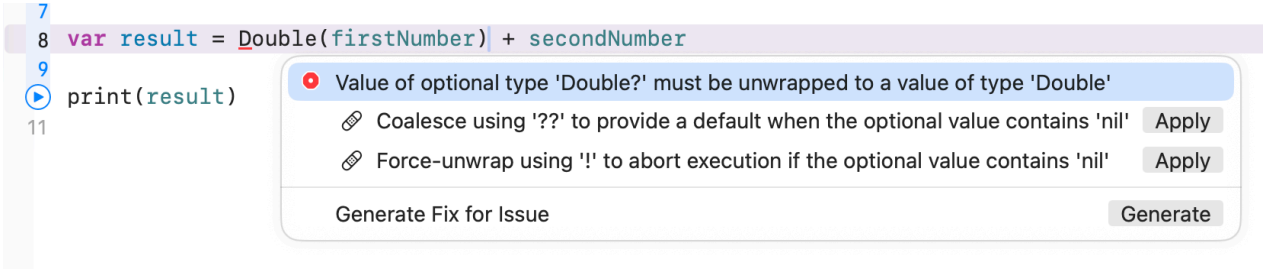
คลิกที่ข้อความ error เพื่อแสดงคำอธิบายจากคอมไพเลอร์



เพื่อขยายความเข้าใจ ผู้สอนอธิบายเกี่ยวกับคุณสมบัติ Type Safety ในภาษา Swift และปัญหา Type Mismatch Error ที่ปรากฏขึ้นมา และเสนอแนะให้ผู้เรียนปรับปรุงโค้ดในบรรทัดที่ 8 เพื่อแก้ปัญหา Type Mismatch Error ด้วยการแปลงชนิดข้อมูล (Type casting) ดังนี้

```
var result = Double(firstNumber) + secondNumber
```

อย่างไรก็ตาม เมื่อผู้เรียนทำการแปลงชนิดของข้อมูลเรียบร้อยแล้ว แต่คอมไพเลอร์ยังคงตรวจพบและแสดงปัญหาขึ้นมาอยู่ ปัญหาดังกล่าว คือ Optional Handling Error ซึ่งเกิดจากการพยายามนำค่าแบบ Optional ไปใช้งานในบริบทที่ต้องการค่าแบบ Non-optional โดยยัง ไม่ได้จัดการ (unwrap) ค่า Optional ให้เรียบร้อย



ในกรณีนี้ ระบบจะไม่เพียงแจ้งว่า “ตรวจพบข้อผิดพลาด” แต่จะ **เสนอแนวทางแก้ไขที่เหมาะสมตามบริบท** ผ่านกลไกที่เรียกว่า **Fix-It** หรือ **Quick Fix** ซึ่งทำงานแบบโต้ตอบกับนักพัฒนาอย่างเป็นขั้นตอน จากตัวอย่าง คอมไพเลอร์เสนอแนวทางในการแก้ไข จำนวน 2 แนวทาง คือ

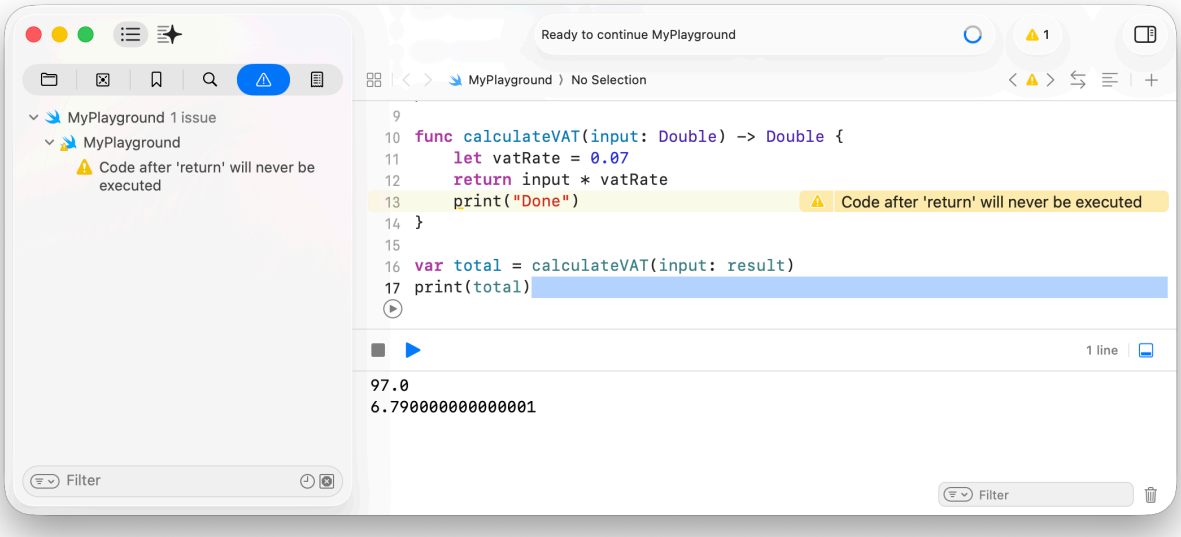
(1) การใช้ Nil-coalescing ด้วย ?? เพื่อการกำหนดค่าเริ่มต้น หากไม่สามารถแปลงค่าได้สำเร็จ

(2) การใช้ Force-unwrap ด้วย ! เพื่อยืนยันว่า ค่าดังกล่าวจะไม่เป็นค่า nil โดยโปรแกรมจะหยุดการทำงานทันทีถ้าผลการทำงานเป็น nil

กำหนดให้ผู้เรียนพิจารณาทางเลือกในการแก้ไขปัญหา พร้อมอธิบายเหตุผล หลังจากนั้นให้เขียนโค้ดเพิ่มดังนี้

```
func calculateVAT(input: Double) -> Double {
    let vatRate = 0.07
    return input * vatRate
    print("Done")
}

var total = calculateVAT(input: result)
print(total)
```

คลิกที่ข้อความ Warning เพื่อแสดงคำอธิบายจากคอมไพเลอร์



ผู้สอนอธิบายเพิ่มเติมเพื่อให้ผู้เรียนเกิดความเข้าใจว่า **ฟังก์ชัน calculateVAT** มีหน้าที่คำนวณภาษีมูลค่าเพิ่มจากค่าที่รับเข้ามาและจะคืนค่าเป็นข้อมูลแบบ Double ตามที่ประกาศไว้ อย่างไรก็ตาม เมื่อฟังก์ชันทำงานมาถึงคำสั่ง return การทำงานของฟังก์ชันจะสิ้นสุดลงทันที ส่งผลให้คำสั่ง **print("Done")** ที่อยู่ถัดมา ไม่มีโอกาสถูกเรียกใช้งานในทุกกรณี คอมไพเลอร์จึงแสดง Warning เพื่อแจ้งให้ผู้พัฒนาทราบว่า โค้ดในส่วนนี้ไม่มีประโยชน์ในเชิงการทำงานของโปรแกรม

ประเด็นสำคัญ คือ Warning ประเภทนี้ **ไม่ใช่ข้อผิดพลาดทางไวยากรณ์ (Syntax Error)** และ **ไม่ใช่ข้อผิดพลาดเชิงตรรกะที่ทำให้ผลลัพธ์ผิดพลาดโดยตรง** เนื่องจากฟังก์ชันยังสามารถคืนค่าและถูกเรียกใช้งานได้ตามปกติ ดังจะเห็นได้จากโค้ดที่เรียกใช้ฟังก์ชัน calculateVAT

```
var total = calculateVAT(input: result)
print(total)
```

โค้ดดังกล่าวยังทำงานและได้ผลลัพธ์ที่ถูกต้อง แสดงให้เห็นว่า Warning ไม่ได้ขัดขวางการทำงานของโปรแกรม แต่ทำหน้าที่เป็น**สัญญาณเตือนด้านคุณภาพของโค้ด (Code Quality)**

ผู้สอนและผู้เรียนสรุปรวมกันเกี่ยวกับประโยชน์ในการนำคุณสมบัติสำคัญของ Xcode เช่น Syntax Highlighting, Code Completion และ Compiler Errors & Warnings มาใช้ในการพัฒนาแอป ดังนี้

- **Syntax Highlighting** ทำหน้าที่เป็นเครื่องมือช่วยในการมองเห็นโครงสร้างของโค้ดอย่างเป็นระบบ การแยกสีช่วยให้นักพัฒนาสามารถรับรู้บทบาทขององค์ประกอบต่างๆ ในภาษาได้ทันที และช่วยให้เข้าใจโครงสร้างเชิงลำดับชั้นของอัลกอริทึมได้ดีขึ้น การเปลี่ยนโค้ดจากข้อความธรรมดาให้กลายเป็นโครงสร้างที่สามารถ “อ่าน วิเคราะห์ และตรวจสอบ” ได้ง่ายเป็นพื้นฐานสำคัญของการออกแบบโปรแกรมที่มีคุณภาพ
- **Code Completion** ทำหน้าที่เป็นผู้ช่วยเชิงปัญญาในระหว่างการเขียนโค้ด โดยอาศัยการวิเคราะห์บริบทของภาษาเพื่อแนะนำคำสั่ง โครงสร้าง และฟังก์ชันเพื่อใช้ในการทำงานที่มีความเหมาะสมในขณะนั้น คุณสมบัตินี้ช่วยลดการพึ่งพาการจดจำไวยากรณ์หรือชื่อคำสั่งทั้งหมด ทำให้นักพัฒนาสามารถมุ่งความสนใจไปที่การออกแบบตรรกะและโครงสร้างของโปรแกรมได้มากขึ้น
- **Compiler Errors & Warnings** ทำหน้าที่เป็นกลไกสะท้อนคุณภาพของโค้ดในเชิงลึก โดย Error ช่วยป้องกันความผิดพลาดที่ทำให้โปรแกรมไม่สามารถทำงานได้ ขณะที่ Warning ทำหน้าที่เตือนถึงปัญหาเชิงโครงสร้าง ความสมเหตุสมผล และแนวปฏิบัติที่ควรปรับปรุง

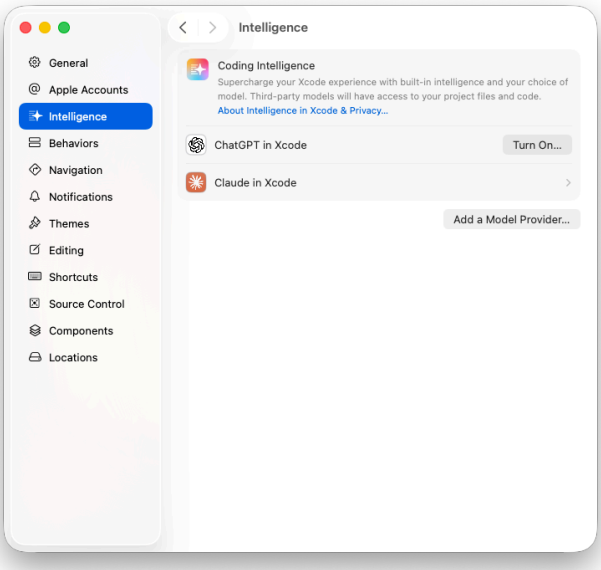
กิจกรรมที่ 5 การใช้เทคโนโลยีปัญญาประดิษฐ์และคุณสมบัติ Coding Intelligence
กิจกรรมต่อยอดจากการใช้เครื่องมือพื้นฐาน ไปสู่การใช้เทคโนโลยีปัญญาประดิษฐ์ผ่านคุณสมบัติ Coding Intelligence ใน Xcode 26 ผู้เรียนจะได้ทดลองใช้ AI เพื่อช่วยแนะนำ แก้ไข และอธิบายโค้ด SwiftUI พร้อมทั้งเรียนรู้การใช้ AI ในฐานะเครื่องมือสนับสนุนการเรียนรู้ (AI-Assisted Development)

Xcode 26 และ ศัยกภาพของชิป Apple Silicon ได้ยกระดับประสบการณ์การพัฒนาแอปไปอีกขั้นด้วยคุณสมบัติ Coding Intelligence ซึ่งเป็นการผสานเทคโนโลยีปัญญาประดิษฐ์เข้ากับ IDE อย่างเป็นระบบ ทำให้สามารถให้คำแนะนำได้อย่างชาญฉลาด เมื่อพิจารณาเปรียบเทียบกับคุณสมบัติ Code Completion หรือ Autocomplete แบบเดิม โดยจะเห็นได้อย่างชัดเจนว่า Autocomplete ในยุคก่อนมุ่งเน้นที่ระดับไวยากรณ์ของภาษาและสามารถตอบคำถามพื้นฐานได้ดีว่า “ควรพิมพ์อะไรต่อ” แต่ไม่สามารถอธิบายได้ว่า “โค้ดนี้ทำงานอย่างไร” หรือ “เหตุใดจึงควรเขียนในลักษณะนี้”

อย่างไรก็ตาม Coding Intelligence ไม่ได้ถูกออกแบบมาเพื่อทำหน้าที่เขียนโค้ดแทนนักพัฒนา แต่ทำหน้าที่เป็น “ผู้ช่วยทางความคิด” ที่ช่วยอธิบาย แนะนำ และสนับสนุนในกระบวนการพัฒนาแอป โดยเฉพาะอย่างยิ่งสำหรับผู้เรียนและนักพัฒนาระดับเริ่มต้น ดังนั้น ในการเรียนรู้และพัฒนาแอปด้วย SwiftUI ผู้เรียนสามารถใช้ AI เพื่อทำหน้าที่ในการอธิบายโค้ด (Code Explainer) ช่วยวิเคราะห์ปัญหาและข้อผิดพลาด (Problem Solver) และสนับสนุนการปรับปรุงโค้ด (Refactoring Assistant)

ก่อนเปิดใช้งาน Coding Intelligence ผู้เรียนควรตรวจสอบเงื่อนไขพื้นฐานดังนี้

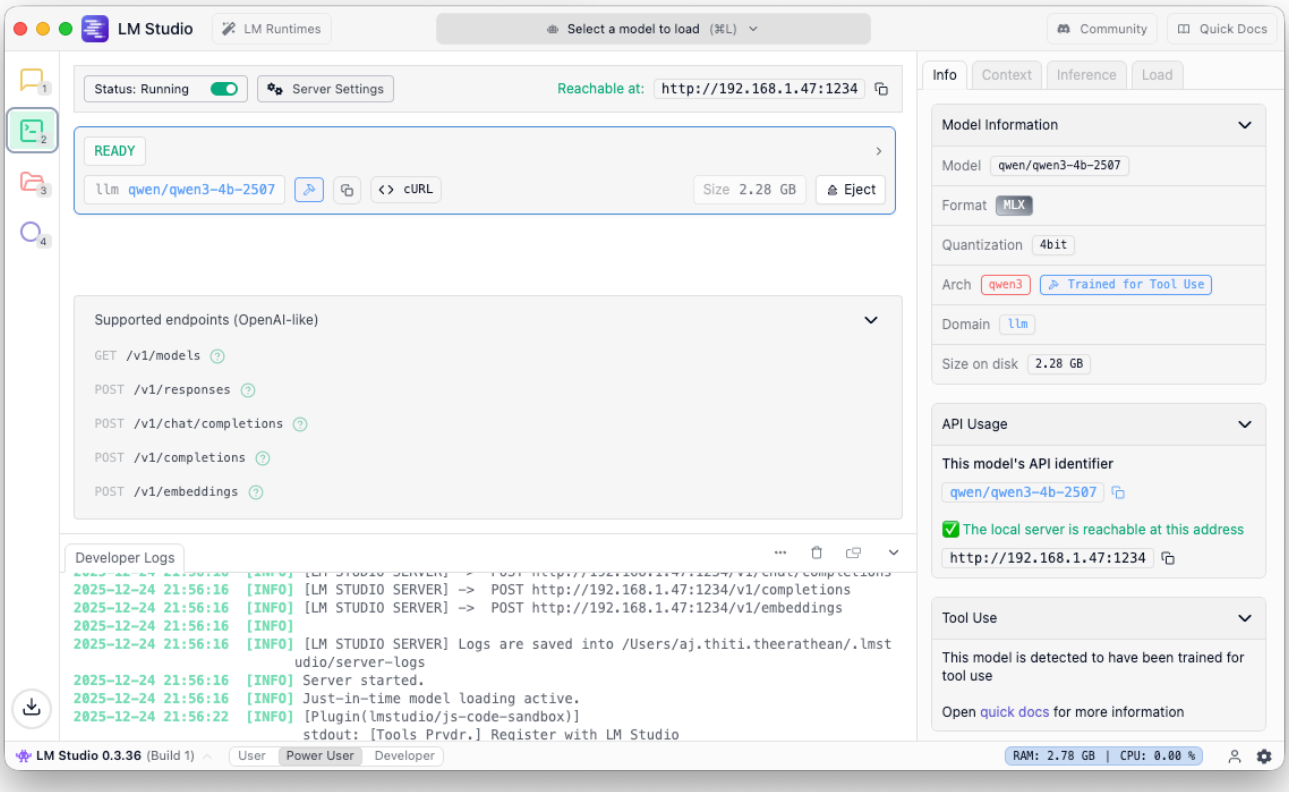
- เครื่อง Mac ใช้ชิป Apple Silicon เช่น M1 หรือ รุ่นที่ใหม่กว่า
- ติดตั้งระบบปฏิบัติการ macOS เวอร์ชัน 26 หรือ รุ่นที่ใหม่กว่า
- ติดตั้ง Xcode 26 เวอร์ชัน 26 หรือ รุ่นที่ใหม่กว่า
- บัญชีผู้ใช้งาน ChatGPT หรือ Claude ai



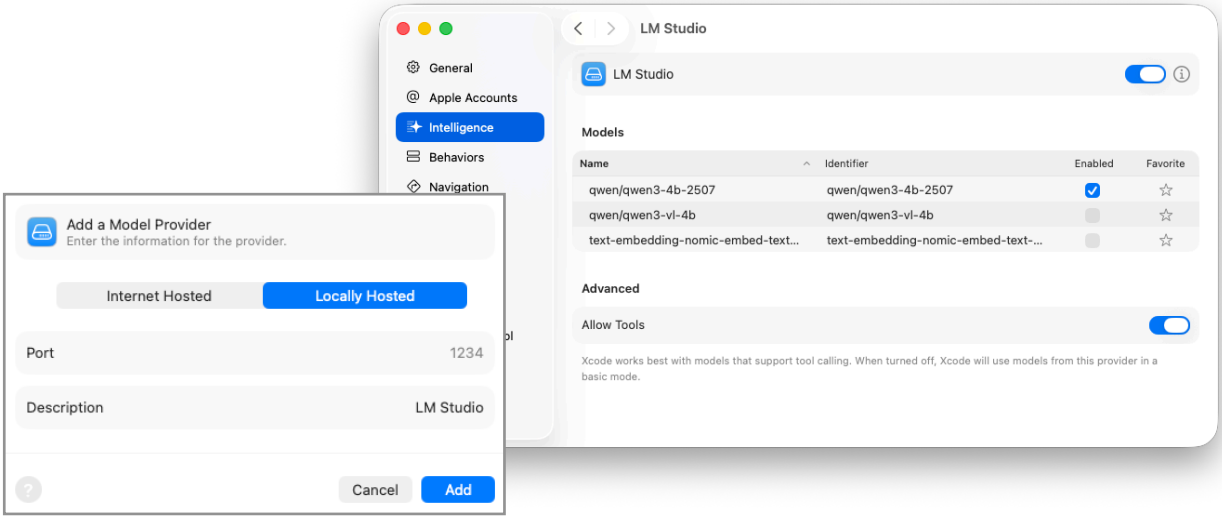
การเปิดใช้คุณสมบัติ Coding Intelligence ใน Xcode

1. เลือก Xcode > Settings...
2. เลือกแท็บ Intelligence
3. เลือก Login ด้วยบัญชีผู้ใช้งาน ChatGPT หรือ Claude ai

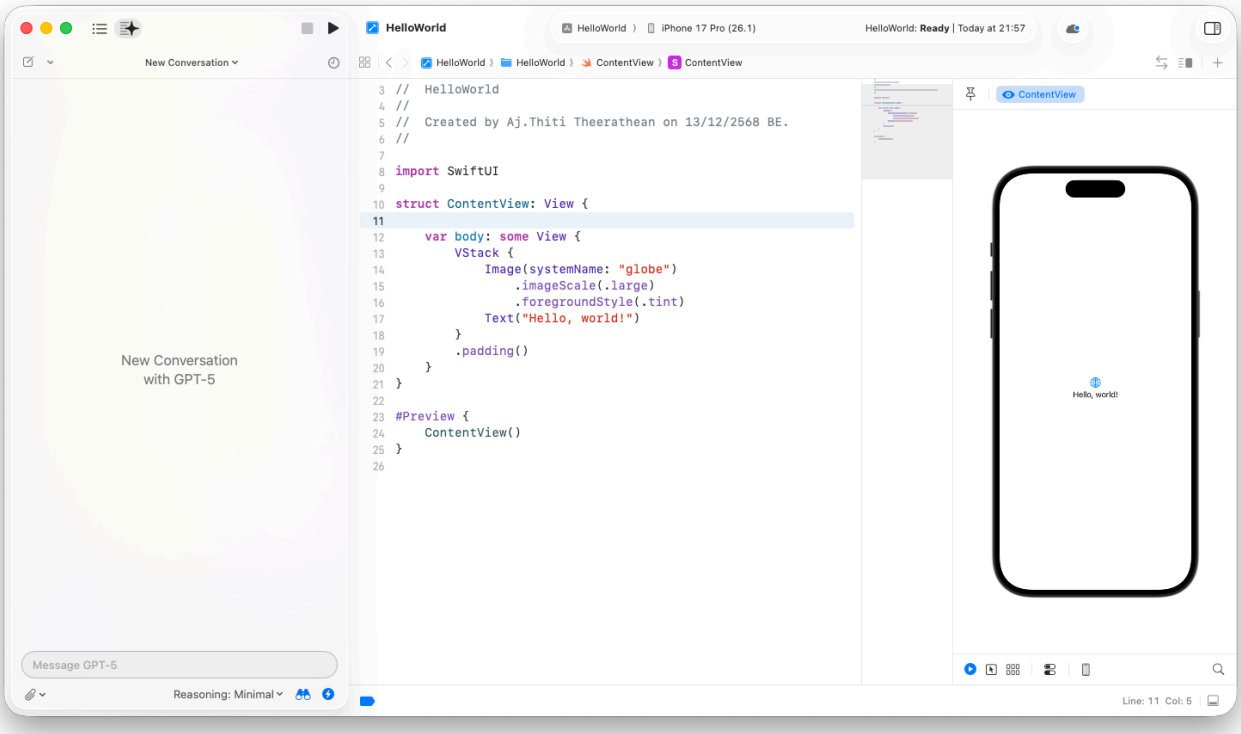
นอกจาก ChatGPT หรือ Claude ai แล้ว เรายังสามารถใช้ **Add a Model Provider..** เพื่อเชื่อมต่อ Xcode กับ LLMs จากผู้ให้บริการรายอื่น หรือการใช้ LM Studio หรือ Ollama เพื่อติดตั้งโมเดลภาษาบนเครื่องของเราเอง และใช้งานในรูปแบบ On-device AI (Local LLMs)ได้อีกด้วย



ตัวอย่างในการกำหนดค่าการใช้งานในรูปแบบ On-device AI โดยติดตั้งโมเดลภาษา Qwen3 ที่มีพารามิเตอร์ 4 พันล้านตัว บนเครื่องของเราเองผ่านโปรแกรม LM Studio



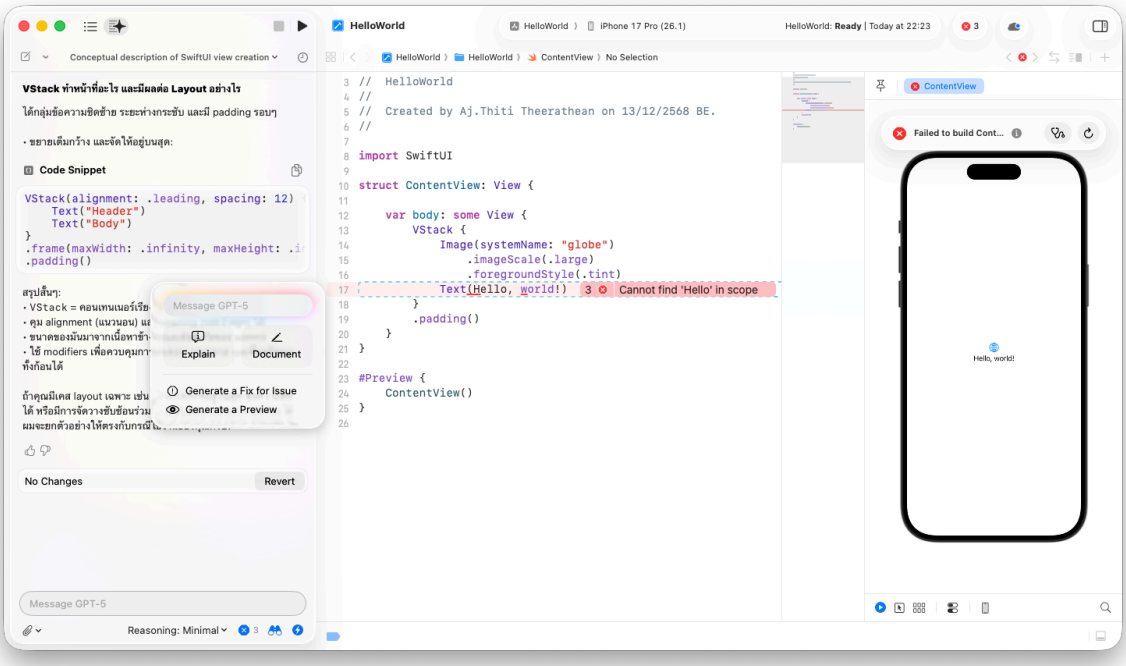
เมื่อติดตั้งโมเดลภาษาเรียบร้อยแล้ว ให้เปิดโปรเจกต์ HelloWorld แล้วคลิกที่ปุ่ม Coding Intelligence



การใช้ Coding Intelligence เพื่ออธิบายโค้ด SwiftUI

1. เลือกที่ไฟล์ ContentView.swift
2. ให้ผู้เรียนตั้งคำถาม โดยการพิมพ์ Prompt ลงในช่อง Message เช่น
   - Prompt - “อธิบายโครงสร้างของ SwiftUI ในไฟล์ ConTentView ในเชิงแนวคิด”
   - Prompt - “VStack ทำหน้าที่อะไร และมีผลต่อ Layout อย่างไร”
3. ผู้เรียนอ่านคำอธิบายจาก AI และสรุปความเข้าใจด้วยภาษาของตนเอง (Reflection สั้น)

การใช้ Coding Intelligence เพื่อวิเคราะห์ปัญหาและข้อผิดพลาด (Problem Solver)



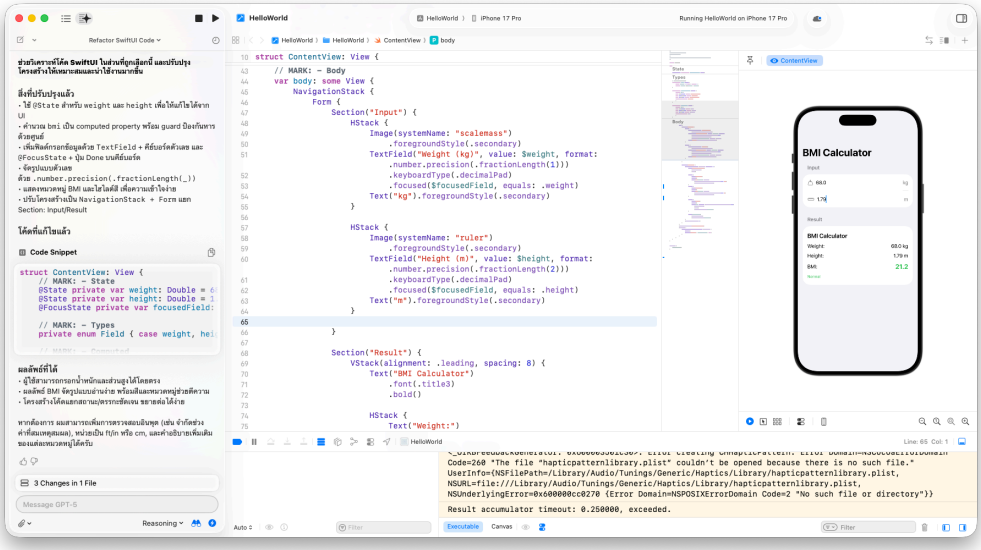
1. แก้โค้ดเพื่อให้เกิด Error เช่น Text(Hello, world!)
2. สังเกต Compiler Error ที่ Xcode แสดงขึ้นมา
3. ดับเบิลคลิกเพื่อเลือกบรรทัดที่เกิด Error
4. คลิกที่สัญลักษณ์ Coding Intelligence และพิมพ์ Prompt เพื่อให้ AI ช่วยอธิบาย Error ที่เกิดขึ้น พร้อมบอกแนวทางแก้ไขปัญหา เช่น
   Prompt - “อธิบายว่า Error นี้เกิดจากอะไร และควรแก้ไขอย่างไร เพราะเหตุใด”
5. ผู้เรียนอ่านคำอธิบายจาก AI และสรุปความเข้าใจด้วยภาษาของตนเอง (Reflection สั้น)
6. แก้ไขโค้ดตามคำแนะนำของ AI

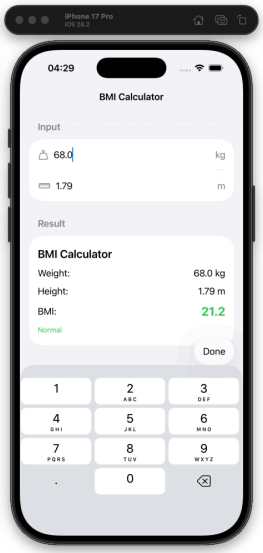
การใช้ Coding Intelligence เพื่อการปรับปรุงโค้ด (Refactoring Assistant)

กำหนดให้ผู้เรียนปรับปรุงโค้ดในไฟล์ ContentView.swift ดังนี้

```
struct ContentView: View {
    var body: some View {

        VStack(spacing: 16) {

            let weight = 68.0
            let height = 1.70
            let bmi = weight / (height * height)

            Text("BMI Calculator")
                .font(.title)

            Text("Weight: \(weight) kg")
            Text("Height: \(height) m")
            Text("BMI: \(bmi)")

        }
        .padding()
    }
}
```

ให้ผู้เรียนเลือกโค้ดทั้งหมดใน ContentView จากนั้นให้เรียกใช้ Coding Intelligence เพื่อให้ช่วยในการปรับปรุงโค้ดด้วย Prompt อย่างเช่น “ช่วยวิเคราะห์โค้ด SwiftUI ในส่วนที่ถูกเลือกนี้ และปรับปรุงโครงสร้างให้เหมาะสมและน่าใช้งานมากขึ้น” หลังจากนั้น ให้ผู้เรียนอ่านคำอธิบายจาก AI และสรุปความเข้าใจด้วยภาษาของตนเอง พร้อมแก้ไขโค้ดตามคำแนะนำของ AI





กิจกรรมที่ 6 : การทดสอบแอปบนอุปกรณ์เสมือนและอุปกรณ์จริง

กิจกรรมสุดท้าย มุ่งให้ผู้เรียนเกิดความเข้าใจเกี่ยวกับการทดสอบแอปที่ผู้เรียนพัฒนาขึ้น โดยการใช้ Simulator และอุปกรณ์จริง (หากมี) ผู้เรียนจะได้เรียนรู้ขั้นตอนการตรวจสอบการทำงานของแอปก่อนนำไปใช้งานและเห็นผลลัพธ์ของการพัฒนาแอปในสภาพแวดล้อมที่ใกล้เคียงการใช้งานจริง กิจกรรมนี้จะทำหน้าที่ปิดวงจรการเรียนรู้ ตั้งแต่การออกแบบ พัฒนา ไปจนถึงการทดสอบ และช่วยให้ผู้เรียนเห็นภาพรวมของกระบวนการพัฒนาแอปอย่างครบถ้วน

การทดสอบการทำงานบนอุปกรณ์เสมือน (Simulator)

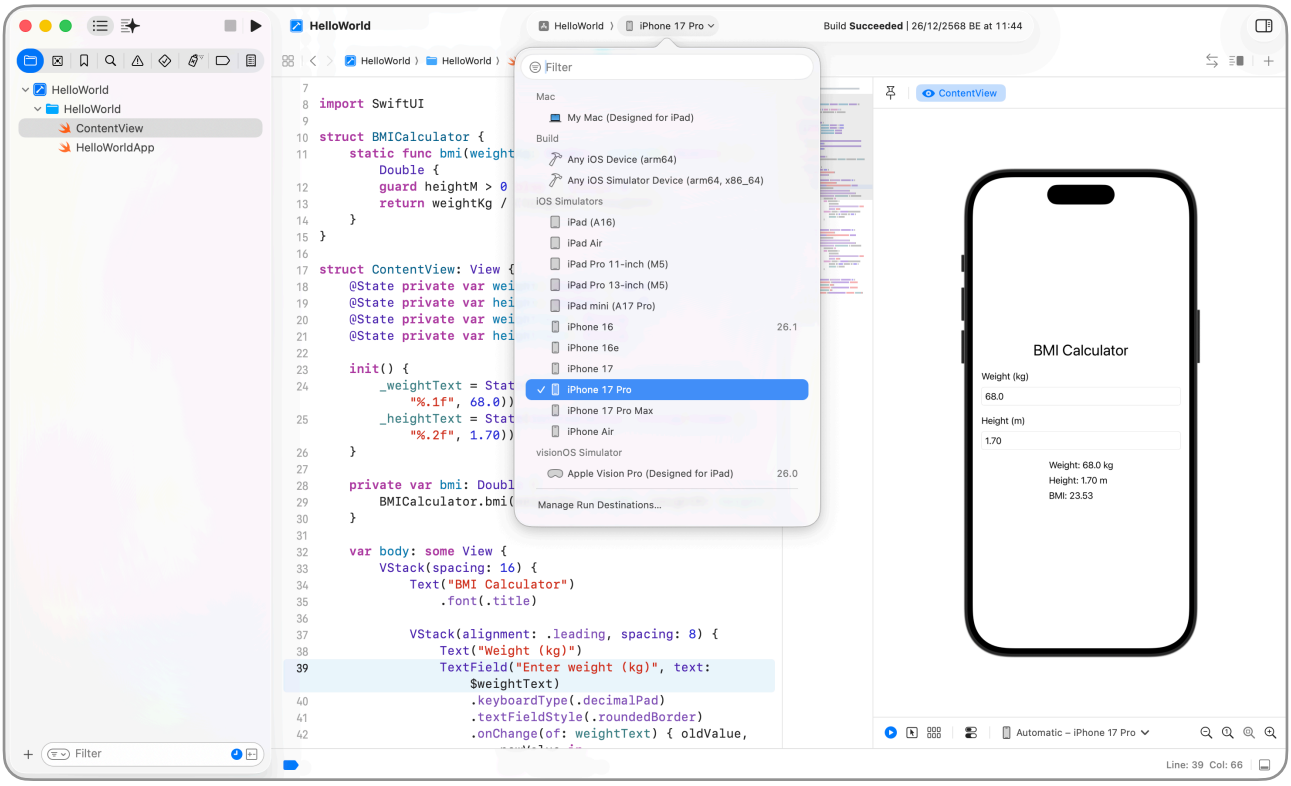
อุปกรณ์เสมือน หรือ Simulator คือ เครื่องมือที่ Xcode จัดเตรียมไว้เพื่อจำลองสภาพแวดล้อมของอุปกรณ์ Apple บนเครื่อง Mac โดยสามารถจำลอง iPhone, iPad หรืออุปกรณ์รุ่นต่างๆ พร้อมระบบปฏิบัติการเวอร์ชันที่หลากหลาย การทดสอบผ่าน Simulator มีข้อได้เปรียบที่สำคัญในด้าน ความสะดวกและรวดเร็วในการพัฒนา

Simulator ช่วยให้นักพัฒนาสามารถเริ่มต้นทดสอบแอปได้ทันทีโดยไม่ต้องมีอุปกรณ์จริง โดยสามารถสั่งรันแอปเพื่อดูผลลัพธ์ได้อย่างรวดเร็ว ทำให้เกิดวงจรการเรียนรู้แบบ เขียน–รัน–ปรับปรุง (Iterative Development) ซึ่งสอดคล้องกับแนวคิดการพัฒนาแบบ Agile

นอกจากนี้ Simulator ยังเหมาะสำหรับการทดสอบด้าน User Interface และ Layout เช่น การปรับขนาดหน้าจอ การเปลี่ยน Orientation

แต่อย่างไรก็ตาม Simulator ก็ยังคงเป็นเพียงการ “จำลอง” จึงไม่สามารถสะท้อนพฤติกรรมของฮาร์ดแวร์ในอุปกรณ์จริงได้ทั้งหมด เช่น ประสิทธิภาพของเซนเซอร์ กล้อง ความแม่นยำของ GPS หรือการจัดการพลังงานของแบตเตอรี่

ขั้นตอนในการทดสอบแอปบนอุปกรณ์เสมือน



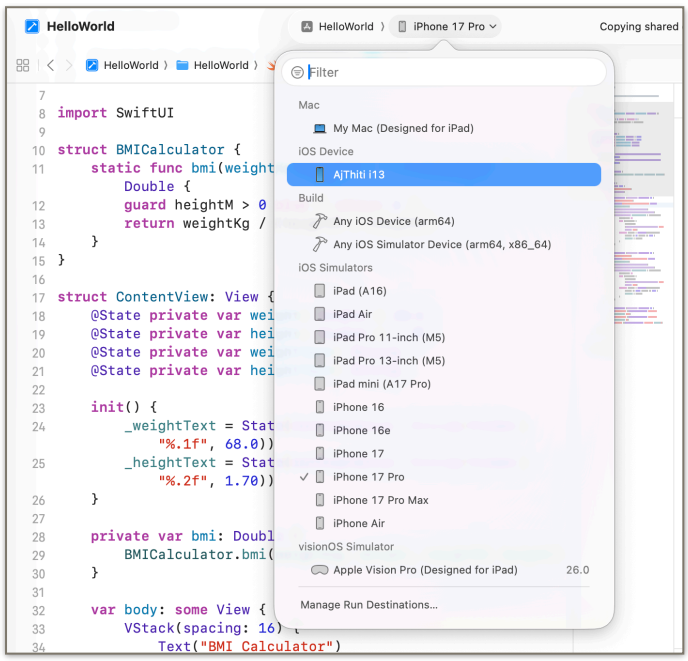
1. เปิดโปรเจกต์ SwiftUI ใน Xcode บริเวณ Toolbar ด้านบน เลือกเมนู **Run Destination** และเลือกอุปกรณ์เสมือนที่ต้องการ เช่น iPhone 17 Pro, iPhone SE หรือ iPad (รุ่นต่าง ๆ)
2. คลิก **ปุ่ม Run** หรือ **กด Command + R**

ผู้สอนสามารถใช้ Simulator เป็นฐานในการตั้งคำถามเพื่อให้ผู้เรียนสะท้อนคิด เช่น
- หากเปลี่ยนอุปกรณ์จาก iPhone เป็น iPad การแสดง UI จะมีการเปลี่ยนไปอย่างไร ?
- Layout ที่ออกแบบไว้รองรับการหมุนหน้าจอหรือไม่ ?

หมายเหตุ : หากต้องการเพิ่มอุปกรณ์สำหรับการทดสอบ สามารถทำได้ที่ Manage Run Distinations...

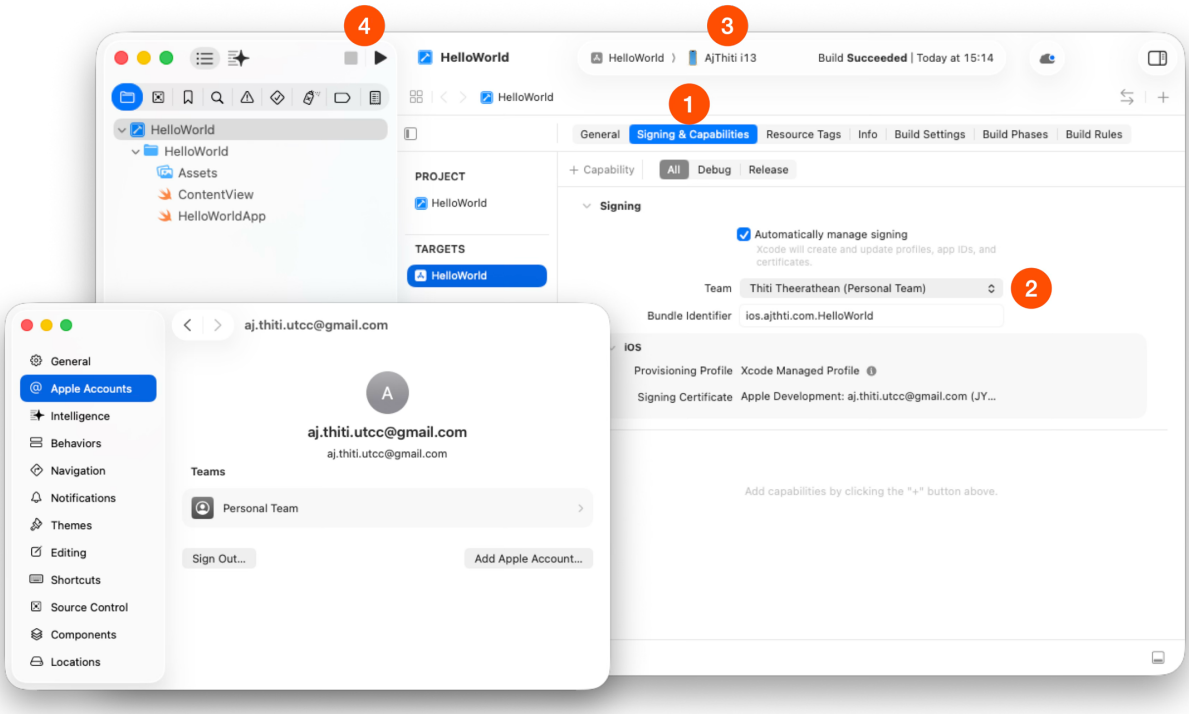
ขั้นตอนในการทดสอบแอปบนอุปกรณ์จริง

ในการสาธิตการทดสอบการทำงานของแอปบนอุปกรณ์จริง ผู้สอนจะต้องดำเนินการสมัครบัญชี Apple ID สำหรับนักพัฒนา ซึ่งสามารถดำเนินการสมัครได้ผ่านเว็บไซต์ https://developer.apple.com/ และเปิดคุณสมบัติ Developer Mode บนอุปกรณ์ที่ต้องการใช้ในการทดสอบ โดยสามารถเปิดใช้คุณสมบัติดังกล่าวได้ที่ **Setting > Privacy & Security > Developer Mode** และกำหนดค่าให้เป็น **On**

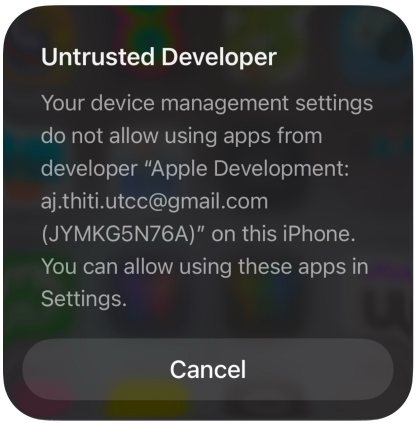


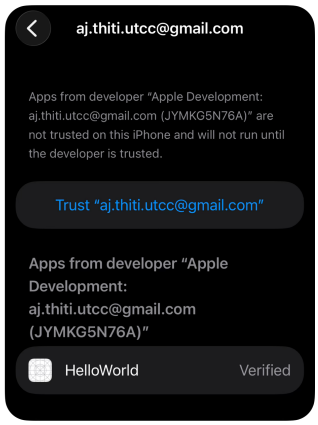
เมื่อเชื่อมต่ออุปกรณ์เข้ากับเครื่อง Mac ผ่านสาย USB แล้ว อุปกรณ์ดังกล่าวจะถูกแสดงขึ้นมาให้เลือกในส่วน **Run Destination** โดยอัตโนมัติ

เนื่องจากแอปที่ติดตั้งบนอุปกรณ์จริง จำเป็นจะต้องผ่านขั้นตอนการยืนยันแหล่งที่มา ซึ่งเป็นกระบวนการทางความปลอดภัยที่เรียกว่า **Code Signing** ซึ่งถูกใช้เพื่อการระบุว่าแอปถูกสร้างโดยนักพัฒนาที่เชื่อถือได้และโค้ดของแอปไม่ได้ถูกแก้ไขหรือดัดแปลงหลังจากผ่านกระบวนการสร้าง (Build) แล้ว สำหรับบัญชีผู้พัฒนาแบบ Free ให้ดำเนินการดังนี้



1. เปิดการตั้งค่าโปรเจกต์ใน Xcode และไปที่ **แท็บ Signing & Capabilities**
2. คลิกที่ **ปุ่ม Team** เพื่อเชื่อมโยงกับบัญชี Apple ID สำหรับการพัฒนา โดยตรวจสอบให้แน่ใจว่า Bundle Identifier ของแอปไม่ซ้ำกับแอปอื่นที่เคยสร้างไว้ และ **เปิดใช้งาน Automatic Signing**
3. กำหนดเป้าหมายที่ Toolbar ของ Xcode โดยเลือกอุปกรณ์จริงที่เชื่อมต่ออยู่เป็น Run Destination
4. คลิกที่ **ปุ่ม Run**





หลังจากติดตั้งแอปบนอุปกรณ์เสร็จแล้ว การเรียกแอปขึ้นมาทำงานครั้งแรกจำเป็นต้องมีการยืนยันสิทธิ์ของนักพัฒนา (Trust Developer) บนอุปกรณ์เป็นขั้นตอนสุดท้าย โดยสามารถดำเนินการได้ที่ **Setting > General > VPN & Device Management**

ภายใต้หัวข้อ **Developer App** จะปรากฏชื่อ Apple ID หรือ Team ที่ใช้พัฒนาแอป ให้แตะเลือกที่ชื่อดังกล่าวเพื่อทำการยืนยันสิทธิ์นักพัฒนาบนอุปกรณ์

ขั้นสรุปบทเรียน (Conclusion)

ภายหลังจากผู้เรียนได้ทำกิจกรรมการเรียนรู้ครบทั้ง 6 กิจกรรม ผู้สอนควรชวนผู้เรียนสรุปร่วมกันว่า บทเรียนนี้ไม่ได้มุ่งเพียงการเรียนรู้วิธีใช้ Xcode ในเชิงเทคนิค แต่เป็นการทำความเข้าใจกระบวนการพัฒนาแอปพลิเคชันบนแพลตฟอร์มของ Apple อย่างเป็นระบบ ตั้งแต่การเตรียมสภาพแวดล้อม การเขียนโค้ด การออกแบบส่วนติดต่อผู้ใช้ ไปจนถึงการทดสอบการทำงานของแอป

ผู้สอนควรเน้นให้ผู้เรียนเห็นว่า Xcode ทำหน้าที่เป็น Integrated Development Environment (IDE) ที่สนับสนุนการทำงานของนักพัฒนาในทุกขั้นตอน ไม่ใช่เพียงพื้นที่สำหรับพิมพ์โค้ด การเข้าใจบทบาทของส่วนประกอบต่าง ๆ เช่น Navigator, Editor, Canvas และ Debug Area ช่วยให้ผู้เรียนสามารถเลือกใช้เครื่องมือได้อย่างเหมาะสมและมีประสิทธิภาพ

ในส่วนของการพัฒนาแอปด้วย SwiftUI ผู้เรียนได้เรียนรู้แนวคิด Declarative UI และโครงสร้างพื้นฐานของแอป ตั้งแต่ระดับ App และ Scene ไปจนถึง View Hierarchy ซึ่งช่วยเปลี่ยนวิธีคิดจากการกำหนดขั้นตอนการทำงานของ UI ไปสู่การอธิบายลักษณะของ UI ที่ต้องการให้เกิดขึ้น
การใช้ Playground ทำให้ผู้เรียนเห็นว่าการทดลองเขียนโปรแกรมและอัลกอริทึมเป็นกระบวนการเรียนรู้ที่สำคัญ ผู้เรียนสามารถทดลองแนวคิด เห็นผลลัพธ์ทันที และเรียนรู้จากความผิดพลาดก่อนนำไปใช้กับโปรเจกต์จริง
ผู้สอนควรสรุปร่วมกับผู้เรียนว่า คุณสมบัติสำคัญของ Xcode เช่น Syntax Highlighting, Code Completion และ Compiler Errors & Warnings เป็นเครื่องมือที่ช่วยยกระดับคุณภาพของโค้ด และส่งเสริมการคิดเชิงโครงสร้าง โดยข้อผิดพลาดและคำเตือนเป็นข้อมูลสะท้อนกลับที่ช่วยพัฒนาความเข้าใจ ไม่ใช่อุปสรรคในการเรียนรู้

ในกิจกรรม Coding Intelligence ผู้เรียนได้เรียนรู้การใช้ AI ในฐานะผู้ช่วยในการอธิบาย วิเคราะห์ และปรับปรุงโค้ด ภายใต้แนวคิด AI-Assisted Development ซึ่งยังคงให้มนุษย์เป็นผู้ตัดสินใจหลักสุดท้าย ผู้สอนควรเชื่อมโยงไปสู่การทดสอบแอปบน Simulator และอุปกรณ์จริง เพื่อให้ผู้เรียนเห็นว่าการพัฒนาแอปไม่สิ้นสุดที่การเขียนโค้ด แต่ต้องผ่านการทดสอบในบริบทที่หลากหลาย รวมทั้งตระหนักถึงประเด็นด้านความปลอดภัยและ Code Signing

คำถามเพื่อการสะท้อนผลการเรียนรู้จากการทำกิจกรรม

1. จงอธิบายว่า Xcode เป็นเครื่องมือที่มีความสำคัญต่อการพัฒนาแอปตามแนวคิดแบบ Agile อย่างไร ?
2. นักเรียนคิดว่า ส่วนประกอบใดของ Xcode ที่มีผลต่อประสิทธิภาพในการพัฒนาแอปมากที่สุด เพราะเหตุใด ?
3. การใช้ Playground มีประโยชน์ต่อการเรียนรู้ภาษา Swift และการพัฒนาแอปอย่างไร ?
4. Syntax Highlighting และ Code Completion ช่วยเพิ่มประสิทธิภาพในการเขียนโค้ดได้หรือไม่ อย่างไร ?
5. การทำกิจกรรม Compiler Errors และ Warnings มีบทบาทอย่างไรต่อการเรียนรู้ และการปรับปรุงคุณภาพของโค้ดอย่างไร ?
6. Coding Intelligence ช่วยสนับสนุนการเรียนรู้ของเราได้อย่างไร และมีข้อควรระวังในการใช้อย่างไร ?
7. อธิบายความแตกต่างระหว่างการทดสอบแอปบน Simulator และอุปกรณ์จริง และเหตุใดการทดสอบทั้งสองรูปแบบจึงมีความสำคัญต่อการพัฒนาแอป

คำศัพท์สำคัญ

Integrated Development Environment (IDE)
สภาพแวดล้อมซอฟต์แวร์แบบบูรณาการที่ออกแบบมาเพื่อสนับสนุนกระบวนการพัฒนาโปรแกรมอย่างครบวงจร ตั้งแต่การเขียนโค้ด การตรวจสอบความถูกต้อง การคอมไพล์/รัน การทดสอบ ไปจนถึงการดีบัก โดยรวมเครื่องมือที่จำเป็นไว้ในระบบเดียว เพื่อเพิ่มประสิทธิภาพ ความถูกต้อง และความเป็นระบบในการพัฒนาซอฟต์แวร์

Bundle Identifier
ตัวระบุเอกลักษณ์ (Unique Identifier) ของแอปพลิเคชันบนแพลตฟอร์มของ Apple (iOS, iPadOS, macOS, watchOS, tvOS) ใช้เพื่อบอกระบบปฏิบัติการและบริการของ Apple ว่า “แอปนี้คือแอปใด” ในเชิงเทคนิค Bundle Identifier เป็น สตริงแบบ Reverse-DNS ที่ต้องไม่ซ้ำกับแอปอื่นในระบบเดียวกัน

ระบบจัดการเวอร์ชัน (Version Control System: VCS)
ระบบซอฟต์แวร์ที่ใช้ควบคุม ติดตาม และจัดการการเปลี่ยนแปลงของไฟล์หรือซอร์สโค้ดตามลำดับเวลา เพื่อให้นักพัฒนาสามารถย้อนกลับ ตรวจสอบ เปรียบเทียบ และทำงานร่วมกันได้อย่างเป็นระบบ โดยเฉพาะในงานพัฒนาซอฟต์แวร์ที่มีการปรับปรุงเอกสารหรือโค้ดอย่างต่อเนื่อง

Playground
สภาพแวดล้อมสำหรับทดลองเขียนและรันโค้ดภาษา Swift แบบโต้ตอบ (Interactive Environment) ที่ออกแบบมาเพื่อสนับสนุนการเรียนรู้ การทดลองแนวคิด และการทดสอบโค้ด โดยไม่จำเป็นต้องสร้างโปรเจกต์แอปเต็มรูปแบบ

Compiler
คอมไพเลอร์ คือ ซอฟต์แวร์ที่ทำหน้าที่แปลโปรแกรมจากภาษาระดับสูงที่มนุษย์เขียน (เช่น Swift) ให้เป็นรูปแบบที่คอมพิวเตอร์สามารถประมวลผลได้ (เช่น Machine Code หรือ Intermediate Representation) โดยกระบวนการแปลนี้จะดำเนินไปอย่างเป็นระบบ พร้อมทั้ง ตรวจสอบความถูกต้องของโค้ด ตามกฎของภาษา

Static Analysis
กระบวนการตรวจสอบ วิเคราะห์ และประเมินคุณภาพของโค้ดโดยไม่ต้องรันโปรแกรม (analyzing code without execution) เพื่อค้นหาข้อผิดพลาดเชิงโครงสร้าง ปัญหาด้านชนิดข้อมูล ความเสี่ยงเชิงตรรกะ และประเด็นด้านคุณภาพของโค้ด จัดเป็นส่วนหนึ่งของกระบวนการตรวจสอบโค้ดอัตโนมัติ (Automated Code Review) ที่มุ่งเน้นการป้องกันปัญหาก่อนเกิดจริง (Preventive Approach)

Code Signing
กระบวนการรับรองความถูกต้องและความน่าเชื่อถือของซอฟต์แวร์ โดยการใช้ **ลายเซ็นดิจิทัล (Digital Signature)** เพื่อยืนยันว่า ซอร์สโค้ดหรือแอปถูกสร้างโดยนักพัฒนาที่ระบุตัวตนได้จริง และโค้ดไม่ถูกแก้ไขหรือดัดแปลงหลังจากถูกเซ็นรับรองแล้ว

ในระบบนิเวศของ Apple (iOS, iPadOS, macOS) Code Signing ถือเป็น กลไกด้านความมั่นคงปลอดภัย (Security Mechanism) ที่สำคัญมาก และเป็นเงื่อนไขบังคับในการรันและเผยแพร่แอป

แนวคิดการพัฒนาซอฟต์แวร์แบบ Agile
แนวคิดและกรอบการทำงานในการพัฒนาซอฟต์แวร์ที่มุ่งเน้น ความยืดหยุ่น (Flexibility) การพัฒนาแบบเป็นรอบสั้น ๆ (Iterative & Incremental) และ การตอบสนองต่อการเปลี่ยนแปลงอย่างรวดเร็ว โดยให้ความสำคัญกับคุณค่าในการใช้งานจริงมากกว่าการทำตามแผนที่ตายตัว

Agile มีรากฐานมาจาก Agile Manifesto (ค.ศ. 2001) ซึ่งเสนอกรอบความคิดที่แตกต่างจากการพัฒนาแบบดั้งเดิม (เช่น Waterfall) โดยสรุปสาระสำคัญได้ดังนี้

1. การพัฒนาแบบวนซ้ำ (Iterative Development) โดยงานจะถูกแบ่งออกเป็นรอบสั้น ๆ (Iteration หรือ Sprint) ในแต่ละรอบจะมีการออกแบบ พัฒนา ทดสอบ และประเมินผลอย่างครบถ้วน ทำให้ทีมสามารถเรียนรู้จากผลลัพธ์และปรับปรุงงานในรอบถัดไปได้ทันที
2. การส่งมอบงานเป็นชิ้นย่อยที่ใช้งานได้จริง (Incremental Delivery) แทนที่จะรอให้โครงการเสร็จสมบูรณ์ทั้งหมดก่อนใช้งาน Agile จะมุ่งส่งมอบผลงานย่อยที่มีคุณค่าและใช้งานได้จริงอย่างต่อเนื่อง
3. การตอบสนองต่อการเปลี่ยนแปลง (Responding to Change) ซึ่ง Agile ยอมรับว่า ความต้องการของผู้ใช้และบริบทการใช้งานอาจเปลี่ยนแปลงได้ตลอดเวลา จึงออกแบบกระบวนการให้สามารถปรับแผนและปรับทิศทางได้โดยไม่กระทบโครงสร้างหลักของงาน
4. การมีส่วนร่วมของผู้มีส่วนได้ส่วนเสีย (Stakeholder Collaboration) ผู้ใช้ ลูกค้า หรือผู้มีส่วนได้ส่วนเสียมีบทบาทสำคัญในการให้ข้อมูลย้อนกลับ (Feedback) อย่างสม่ำเสมอ เพื่อให้ผลงานที่พัฒนาขึ้นสอดคล้องกับความต้องการจริง
5. การให้ความสำคัญกับทีมและการสื่อสาร โดย Agile เชื่อว่าทีมที่มีการสื่อสารที่ดีและมีอิสระในการตัดสินใจ จะสามารถสร้างผลงานที่มีคุณภาพได้ดีกว่าการทำงานตามคำสั่งหรือเอกสารเพียงอย่างเดียว

รายละเอียดสำหรับการอ้างอิงเอกสารชุดนี้
- **ผู้เขียน** ธิติ ธีระเธียร
- **วันที่เผยแพร่** วันที่ 1 มกราคม 2569
- **เข้าถึงได้จาก** https://ajthiti.gitbook.io/develop-in-swift/meet-xcode
- **เงื่อนไขในการใช้งาน** This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.