เสมือนคำนำ (เรื่องเล่าก่อนเริ่มต้น)

หลายคนอาจเคยมีความฝันที่จะสร้างอุปกรณ์อิเล็กทรอนิกส์อัจฉริยะด้วยตนเอง ไม่ว่าจะเป็นระบบรดน้ำต้นไม้อัตโนมัติ, ไฟส่องสว่างที่เปิดปิดตามความเคลื่อนไหว, หรือสถานีตรวจวัดสภาพอากาศขนาดเล็กที่สามารถส่งข้อมูลขึ้นสู่อินเทอร์เน็ตได้ หากแนวคิดเหล่านี้สร้างความตื่นเต้น แต่ก็ตามมาด้วยความกังวลว่าการเริ่มต้นอาจเป็นเรื่องยากเกินไป หนังสือเล่มนี้ได้มอบแนวทางและคำตอบไว้ให้ทุกท่านแล้ว

ลองจินตนาการถึงการเขียนโปรแกรมควบคุมอุปกรณ์อิเล็กทรอนิกส์ต่าง ๆ ด้วยภาษา Python ซึ่งเป็นภาษาที่ขึ้นชื่อในด้านความเรียบง่าย โครงสร้างชัดเจน อ่านเข้าใจง่าย และได้รับความนิยมอย่างกว้างขวางในหมู่นักพัฒนาทั่วโลก นั่นคือสิ่งที่ MicroPython ได้เข้ามาสานต่อ โดยนำภาษา Python มาปรับแต่งให้สามารถทำงานได้บนไมโครคอนโทรลเลอร์ ซึ่งเป็นหัวใจสำคัญของระบบฝังตัว (Embedded Systems) มากมายในชีวิตประจำวันและเมื่อ MicroPython ถูกนำมาใช้งานร่วมกับ ESP32 ชิปไมโครคอนโทรลเลอร์ยอดนิยม ซึ่งมีทั้งประสิทธิภาพสูง มาพร้อมกับ Wi-Fi และ Bluetooth ที่อยู่ในชิพ ในราคาที่คนทั่วไปเข้าถึงได้ และมีผู้ใช้งานอย่างแพร่หลายทั่วโลก การผสมผสานนี้จึงก่อให้เกิดเป็นคู่หูที่ทรงพลังสำหรับการสร้างสรรค์โครงงานอิเล็กทรอนิกส์และ IoT (Internet of Things) ที่หลากหลาย จากผู้คนทั่วทุกมุมโลก

สำหรับผู้ที่คุ้นเคยกับการเขียนโปรแกรมบน Arduino ด้วยภาษา C/C++ มาก่อน อาจรู้สึกว่าการทดลองพัฒนาโค้ดแต่ละครั้งต้องผ่านกระบวนการคอมไพล์และอัปโหลดโปรแกรมไปยังบอร์ดใหม่ทุกครั้ง ซึ่งทำให้เสียเวลาในการทดสอบมาก โดยเฉพาะเมื่อทำงานกับโค้ดขนาดใหญ่หรือวงจรที่มีความซับซ้อน MicroPython เข้ามาเปลี่ยนประสบการณ์เหล่านี้ ด้วยความยืดหยุ่นของภาษา Python ผู้เรียนสามารถทดลองโค้ดได้ทันทีผ่านระบบ **REPL (Read-Evaluate-Print Loop)** ซึ่งช่วยให้มองเห็นผลลัพธ์ของการสั่งงานกับฮาร์ดแวร์แบบเรียลไทม์โดยไม่ต้องผ่านขั้นตอนการคอมไพล์และอัปโหลดซ้ำในทุกครั้ง โค้ดที่เขียนมีความกระชับ อ่านเข้าใจง่าย ลดภาระความซับซ้อนด้านไวยากรณ์ เหมาะอย่างยิ่งสำหรับผู้ที่ต้องการสำรวจเทคโนโลยีใหม่ ๆ หรือกำลังมองหาเครื่องมือที่ช่วยให้การพัฒนาโครงงานเป็นไปอย่างสนุกสนานและมีประสิทธิภาพมากยิ่งขึ้น

หนังสือเล่มนี้ไม่ได้มุ่งหวังเพียงการสอนวิธีทำให้ไฟ LED กะพริบ (แม้ว่านั่นจะเป็นจุดเริ่มต้นการเรียนรู้ที่ดีก็ตาม) แต่เนื้อหาจะนำพาผู้อ่านเดินทางไปไกลกว่านั้น โดยจะเริ่มปูพื้นฐานตั้งแต่การทำความเข้าใจ MicroPython และ ESP32, การติดตั้งเครื่องมือที่จำเป็น, การเรียนรู้คำสั่ง Python พื้นฐานในบริบทของการควบคุมฮาร์ดแวร์, ไปจนถึงการเชื่อมต่อ ESP32 เข้ากับเซ็นเซอร์ต่างๆ, การแสดงผล, การควบคุมมอเตอร์ และที่สำคัญคือการเชื่อมต่อกับเครือข่าย Wi-Fi เพื่อสร้างโครงงาน IoT ที่สามารถโต้ตอบกับโลกออนไลน์ได้จริง

ตัวอย่าง

เป้าหมายของหนังสือเล่มนี้

หนังสือเล่มนี้ถูกเรียบเรียงขึ้นด้วยความตั้งใจที่จะเป็น "คู่มือ" และ "แหล่งเรียนรู้" สำหรับ

- **ผู้เริ่มต้นใหม่ ที่อาจยังไม่มีประสบการณ์ด้านอิเล็กทรอนิกส์หรือการเขียนโปรแกรม**
- **ผู้ใช้งาน Arduino ที่ต้องการขยายขอบเขตความรู้และทดลองใช้ MicroPython**
- **นักเรียน นักศึกษา และผู้สนใจทั่วไป ที่มีความใฝ่ฝันจะเรียนรู้การสร้างโครงงาน IoT ด้วยตนเอง**
- **ครูอาจารย์ ที่กำลังมองหาแนวทางและสื่อการสอนสำหรับวิชาที่เกี่ยวข้อง**

เนื้อหาในหนังสือมุ่งเน้นการนำเสนอที่ เข้าใจง่าย เป็นขั้นตอน มีตัวอย่างที่ชัดเจน และ รวบรวมความรู้ที่จำเป็นทั้งภาคทฤษฎีและปฏิบัติ เพื่อให้ผู้เรียนสามารถศึกษาและลงมือทำตามได้จริง จนสามารถสร้างสรรค์โครงงานของตนเองได้อย่างมั่นใจ ทั้งหมดนี้ได้ถูกถ่ายทอดเป็นภาษาไทยที่กระชับและเข้าใจง่าย โดยคนไทย เพื่อคนไทย

การเดินทางสู่การสร้างสรรค์โครงงาน IoT ด้วย MicroPython และ ESP32:

ภายในหนังสือเล่มนี้ ผู้อ่านจะได้เรียนรู้ตามลำดับขั้นดังนี้:

1. **การตั้งค่าสภาพแวดล้อม และเครื่องมือที่จำเป็นสำหรับการพัฒนา**
2. **พื้นฐานภาษา Python ที่สำคัญและประยุกต์ใช้บ่อยใน MicroPython**
3. **การควบคุมขา GPIO ของ ESP32 เพื่อสั่งงาน LED, อ่านค่าจากสวิตช์ และอื่นๆ**
4. **การทำงานกับ เซ็นเซอร์ ที่นิยมใช้ เช่น เซ็นเซอร์วัดอุณหภูมิ, ความชื้น, ระยะทาง**
5. **การ เชื่อมต่อ Wi-Fi และพื้นฐานการสื่อสารผ่านเครือข่าย**
6. **โครงงานตัวอย่าง ที่นำความรู้ทั้งหมดมาประยุกต์ใช้ เช่น สถานีตรวจอากาศส่วนบุคคล, ระบบควบคุมไฟอัจฉริยะ...**

ผู้เขียนเชื่อว่าเมื่อผู้อ่านศึกษาหนังสือเล่มนี้จบ จะไม่เพียงแต่เกิดความเข้าใจในหลักการทำงานของ MicroPython และ ESP32 แต่ยังจะได้รับแรงบันดาลใจและความพร้อมที่จะสร้างสรรค์สิ่งประดิษฐ์และนวัตกรรมใหม่ ๆ ออกมาอีกมากมาย ถึงเวลาแล้วที่จะปลดล็อกศักยภาพของไมโครคอนโทรลเลอร์ด้วยความมหัศจรรย์และความเรียบง่ายของ Python มาเริ่มต้นการเดินทางที่น่าตื่นเต้นนี้ไปด้วยกัน

ตัวอย่าง

มานพ ปักษี

ส่วนที่ 1: ก้าวแรกสู่โลก MicroPython กับ ESP32

บทที่ 1: MicroPython และ ESP32 คืออะไร? ทำไมต้องใช้คู่นี้?

1.1 ยินดีต้อนรับสู่โลกแห่งไมโครคอนโทรลเลอร์ยุคใหม่

ในยุคปัจจุบัน เทคโนโลยีได้เข้ามาเป็นส่วนหนึ่งของชีวิตประจำวันอย่างแยกไม่ออก หนึ่งในเทคโนโลยีที่มีบทบาทสำคัญที่อยู่เบื้องหลังของนวัตกรรมและสิ่งอำนวยความสะดวกมากมายก็คือ **ไมโครคอนโทรลเลอร์ (Microcontroller)** หรือที่เราอาจคุ้นเคยในชื่อ **"สมองกลจิ๋ว",”สมองกลฝังตัว”** อุปกรณ์ขนาดเล็กเหล่านี้เปรียบเสมือนหัวใจสำคัญที่ขับเคลื่อนอุปกรณ์อิเล็กทรอนิกส์หลากหลายชนิด ตั้งแต่ของเล่นเด็ก, เครื่องใช้ไฟฟ้าในบ้าน, ระบบควบคุมในรถยนต์, ไปจนถึงอุปกรณ์ทางการแพทย์ที่ซับซ้อน และที่กำลังได้รับความสนใจอย่างกว้างขวางคือการนำไปใช้ในโครงงาน **Internet of Things (IoT)** หรือ "อินเทอร์เน็ตของสรรพสิ่ง" ซึ่งเป็นการเชื่อมโยงอุปกรณ์ต่างๆ เข้ากับเครือข่ายอินเทอร์เน็ต ทำให้สามารถควบคุม สั่งการ หรือเก็บข้อมูลได้จากระยะไกล

การสร้างสรรค์สิ่งประดิษฐ์หรือระบบอัจฉริยะด้วยไมโครคอนโทรลเลอร์จึงไม่ใช่เรื่องไกลตัวอีกต่อไป เปิดโอกาสให้ทุกคน ไม่ว่าจะเป็นนักเรียน นักศึกษา นักประดิษฐ์ หรือผู้ที่สนใจทั่วไป สามารถเรียนรู้และพัฒนาโครงงานของตนเองขึ้นมาได้ อย่างไรก็ตาม ในอดีต การเริ่มต้นใช้งานไมโครคอนโทรลเลอร์อาจมีอุปสรรคอยู่บ้าง ไม่ว่าจะเป็นความซับซ้อนของภาษาโปรแกรมที่ใช้ควบคุม เครื่องมือพัฒนาที่อาจต้องตั้งค่าหลายขั้นตอน หรือข้อจำกัดทางด้านฮาร์ดแวร์ที่อาจไม่ยืดหยุ่นพอสำหรับโครงงานที่ต้องการความสามารถ หลากหลาย และรอบด้าน

แต่ในปัจจุบัน ด้วยความก้าวหน้าของเทคโนโลยี ทั้งด้านซอฟต์แวร์และฮาร์ดแวร์ ได้ทำให้การเข้าถึงและการเรียนรู้ไมโครคอนโทรลเลอร์เป็นเรื่องง่ายและสนุกสนานยิ่งขึ้นอย่างที่ไม่เคยเป็นมาก่อน หนังสือเล่มนี้จะนำพาผู้อ่านทุกท่านเข้าสู่โลกของไมโครคอนโทรลเลอร์ยุคใหม่ ผ่านการทำความรู้จักกับเครื่องมืออันทรงพลังสองสิ่งที่กำลังปฏิวัติวงการนี้ นั่นคือ **MicroPython** และชิป **ESP32** ซึ่งจะช่วยลดข้อจำกัดเดิม ๆ และเปิดประตูสู่ความเป็นไปได้ใหม่ ๆ ในการสร้างสรรค์โครงงานอิเล็กทรอนิกส์และ IoT ได้อย่างง่ายดายและมีประสิทธิภาพ

1.2 MicroPython: Python ในร่างจิ๋วแต่ทรงพลัง

MicroPython คืออะไร? MicroPython คือการนำภาษาโปรแกรม Python ที่ได้รับความนิยมอย่างสูง มาปรับปรุงและย่อส่วนเพื่อให้สามารถทำงานได้อย่างมีประสิทธิภาพบนไมโครคอนโทรลเลอร์ ซึ่งเป็นอุปกรณ์ที่มีทรัพยากรจำกัด เช่น หน่วยความจำ (RAM) และพื้นที่จัดเก็บข้อมูล (Flash memory) น้อยกว่าคอมพิวเตอร์ทั่วไป

มานพ ปักษี

โดยยังคงรักษาจุดเด่นหลักของภาษา Python ไว้ นั่นคือไวยากรณ์ที่อ่านง่าย เข้าใจง่าย คล้ายคลึงกับภาษาอังกฤษ ทำให้ผู้เริ่มต้นสามารถเรียนรู้ได้อย่างรวดเร็ว ผู้สร้าง MicroPython คือ Damien George วิศวกรชาวออสเตรเลีย ผู้เริ่มต้นโครงการนี้ผ่านการระดมทุนบน Kickstarter ในปี 2013 โดยมีเป้าหมายเพื่อสร้างซอฟต์แวร์ที่ช่วยให้การเขียนโปรแกรมควบคุมฮาร์ดแวร์เป็นเรื่องง่ายขึ้น นับตั้งแต่นั้นมา MicroPython ก็ได้รับการพัฒนาอย่างต่อเนื่องและมีชุมชนผู้ใช้งานขยายใหญ่ขึ้นทั่วโลก รองรับไมโครคอนโทรลเลอร์หลากหลายรุ่น

ข้อดีของ MicroPython:

- **เรียนรู้ง่าย เขียนโค้ดสั้นกระชับ:** ด้วยไวยากรณ์ที่เรียบง่ายของ Python ผู้ที่ไม่มีพื้นฐานการเขียนโปรแกรมมาก่อนก็สามารถเริ่มต้นได้ไม่ยาก และสำหรับผู้ที่มีพื้นฐาน Python อยู่แล้ว ก็สามารถนำความรู้เดิมมาปรับใช้ได้ทันที นอกจากนี้ โค้ดที่เขียนด้วย MicroPython มักจะสั้นกว่าโค้ดที่เขียนด้วยภาษาโปรแกรมระดับต่ำอย่าง C/C++ สำหรับงานเดียวกัน ทำให้เข้าใจภาพรวมของโปรแกรมได้ง่ายขึ้น
- **พัฒนาและทดสอบได้รวดเร็ว (Rapid Prototyping):** MicroPython มาพร้อมกับเครื่องมือที่เรียกว่า REPL (Read-Evaluate-Print Loop) ซึ่งเป็น Command Line Interface ที่ช่วยให้ผู้ใช้สามารถพิมพ์คำสั่ง Python และเห็นผลลัพธ์การทำงานได้ทันทีบนตัวไมโครคอนโทรลเลอร์ ทำให้การทดลองโค้ดส่วนเล็กๆ หรือการตรวจสอบการทำงานของฮาร์ดแวร์เป็นไปอย่างรวดเร็ว ไม่จำเป็นต้องคอมไพล์และอัปโหลดโปรแกรมใหม่ทั้งหมดทุกครั้งที่แก้ไขโค้ดเพียงเล็กน้อย
- **มีไลบรารีพื้นฐานพร้อมใช้งาน:** MicroPython มีโมดูล (ไลบรารี) มาตรฐานที่จำเป็นสำหรับการควบคุมฮาร์ดแวร์ เช่น การจัดการขา GPIO (General Purpose Input/Output) การสื่อสารแบบ I2C, SPI, UART รวมถึงการเชื่อมต่อเครือข่าย ทำให้การเริ่มต้นพัฒนาโครงงานเป็นไปได้สะดวก
- **ขยายองค์ความรู้ Python:** การเรียนรู้ MicroPython เท่ากับเป็นการเรียนรู้ภาษา Python ไปในตัว ซึ่งเป็นภาษาที่มีความต้องการสูงในตลาดงานปัจจุบัน สามารถนำไปต่อยอดในการพัฒนาเว็บแอปพลิเคชัน, งานด้านข้อมูล (Data Science), ปัญญาประดิษฐ์ (AI) และอื่นๆ ได้อีกมากมาย

1.3 ESP32: ชิปมหัศจรรย์สำหรับโครงงาน IoT

ESP32 คืออะไร? ESP32 คือชื่อของซีรีส์ชิปไมโครคอนโทรลเลอร์ราคาประหยัดที่พัฒนาโดยบริษัท Espressif Systems จากประเทศจีน ซึ่งได้รับการออกแบบมาให้มีความสามารถรอบด้าน โดยเฉพาะอย่างยิ่งสำหรับ

มหาวิทยาลัยราชภัฏ

งานที่เกี่ยวข้องกับ Internet of Things (IoT) และอุปกรณ์พกพา ESP32 เป็นรุ่นที่พัฒนาต่อยอดมาจาก ESP8266 ซึ่งเคยสร้างความนิยมอย่างสูงในหมู่นักพัฒนาและนักประดิษฐ์ ด้วยการเพิ่มคุณสมบัติและความสามารถที่ทรงพลังยิ่งขึ้น

โดยทั่วไป เมื่อพูดถึง ESP32 มักจะหมายถึงตัวชิป SoC (System on a Chip) แต่ในท้องตลาด บอร์ดพัฒนา (Development Board) ที่ใช้ชิป ESP32 ก็มักถูกเรียกว่า "บอร์ด ESP32" เช่นกัน บอร์ดเหล่านี้จะรวมเอาชิป ESP32 พร้อมวงจรสนับสนุนที่จำเป็น เช่น วงจรแปลงแรงดันไฟฟ้า, พอร์ต USB สำหรับการโปรแกรมและจ่ายไฟ, และขาต่อต่างๆ ทำให้ง่ายต่อการนำไปใช้งาน

จุดเด่นของ ESP32:

- **การเชื่อมต่อไร้สายครบครัน:** ESP32 มาพร้อมกับ Wi-Fi (มาตรฐาน 802.11 b/g/n) และ Bluetooth (ทั้ง Classic และ Bluetooth Low Energy - BLE) ในตัว ทำให้สามารถเชื่อมต่อกับเครือข่ายอินเทอร์เน็ตหรืออุปกรณ์อื่นๆ แบบไร้สายได้อย่างง่ายดาย ซึ่งเป็นหัวใจสำคัญของโครงงาน IoT
- **ประสิทธิภาพสูง:** ส่วนใหญ่มาพร้อมกับโปรเซสเซอร์ Xtensa LX6 แบบ Dual-core (สองแกนประมวลผล) ที่ความเร็วสัญญาณนาฬิกาสูงถึง 240 MHz (ขึ้นอยู่กับรุ่น) ทำให้สามารถประมวลผลงานที่ซับซ้อนได้ดีกว่าไมโครคอนโทรลเลอร์แกนเดียวหลายๆ รุ่น
- **หน่วยความจำเพียงพอ:** มีหน่วยความจำ SRAM และ Flash memory ภายในตัวในปริมาณที่ค่อนข้างมากสำหรับไมโครคอนโทรลเลอร์ (เช่น SRAM หลายร้อย KB และ Flash memory หลาย MB) เพียงพอสำหรับการรัน MicroPython และเก็บโปรแกรมขนาดกลางได้
- **ขาเชื่อมต่อ (GPIO) อเนกประสงค์:** มีขา General Purpose Input/Output (GPIO) จำนวนมาก ซึ่งสามารถกำหนดค่าให้ทำงานได้หลากหลายหน้าที่ เช่น เป็น Digital Input/Output, Analog Input (ADC), Digital to Analog Output (DAC), Pulse Width Modulation (PWM), รวมถึงรองรับโปรโตคอลการสื่อสารมาตรฐาน เช่น I2C, SPI, UART
- **เซ็นเซอร์ในตัว:** บางรุ่นของ ESP32 มีเซ็นเซอร์พื้นฐานในตัว เช่น เซ็นเซอร์สัมผัส (Touch Sensor) และเซ็นเซอร์อุณหภูมิ (Temperature Sensor)

สมาคมไทยอุปกรณ์

- **ความปลอดภัย:** รองรับคุณสมบัติด้านความปลอดภัยของฮาร์ดแวร์ เช่น Secure Boot และ Flash Encryption
- **ราคาคุ้มค่า:** เมื่อเทียบกับคุณสมบัติและความสามารถที่ได้รับ บอร์ดพัฒนา ESP32 ถือว่ามีราคาที่เข้าถึงได้ง่าย ทำให้นักเรียน นักศึกษา และนักประดิษฐ์ทั่วไปสามารถเริ่มต้นได้โดยที่ไม่ต้องลงทุนสูงมากนัก

1.4 MicroPython + ESP32: คู่หูที่ลงตัวที่สุดสำหรับการเริ่มต้น

เมื่อความเรียบง่ายและยืดหยุ่นของ MicroPython ผสานเข้ากับพลังและความสามารถรอบด้านของชิป ESP32 จึงเกิดเป็นแพลตฟอร์มที่ลงตัวอย่างยิ่งสำหรับการเรียนรู้และพัฒนาโครงงานอิเล็กทรอนิกส์และ IoT โดยเฉพาะสำหรับผู้เริ่มต้นและผู้ที่ต้องการสร้างต้นแบบ (Prototype) อย่างรวดเร็ว

เหตุผลที่ MicroPython และ ESP32 เป็นคู่หูที่ยอดเยี่ยม:

- **ลดกำแพงการเริ่มต้น:** ผู้เริ่มต้นไม่จำเป็นต้องกังวลกับความซับซ้อนของภาษา C/C++ หรือการตั้งค่าเครื่องมือที่ยุ่งยาก เพียงแค่ติดตั้งเฟิร์มแวร์ MicroPython ลงบนบอร์ด ESP32 และใช้โปรแกรม Text Editor หรือ IDE ง่ายๆ อย่าง Thonny ก็สามารถเริ่มเขียนโค้ดควบคุมฮาร์ดแวร์ได้ทันที
- **ใช้ประโยชน์จากฮาร์ดแวร์ ESP32 ได้เต็มที่ (อย่างง่ายดาย):** MicroPython สำหรับ ESP32 มีโมดูลที่ช่วยให้การเข้าถึงคุณสมบัติหลักของ ESP32 เช่น Wi-Fi, Bluetooth, GPIO, ADC, PWM เป็นเรื่องง่าย เพียงเขียนโค้ด Python ไม่กี่บรรทัดก็สามารถสั่งงานฟังก์ชันเหล่านี้ได้
- **เหมาะสำหรับโครงงาน IoT:** การเชื่อมต่อ Wi-Fi ของ ESP32 เมื่อรวมกับความสามารถในการจัดการข้อความและข้อมูลของ Python ทำให้การพัฒนาอุปกรณ์ IoT ที่ส่งข้อมูลไปยังเซิร์ฟเวอร์ หรือรับคำสั่งผ่านเครือข่าย ทำได้ไม่ซับซ้อน
- **ชุมชนผู้ใช้งานขนาดใหญ่:** ทั้ง MicroPython และ ESP32 ต่างก็มีชุมชนผู้ใช้งานที่กระตือรือร้นและพร้อมให้ความช่วยเหลือ เมื่อพบปัญหาหรือมีข้อสงสัย สามารถค้นหาข้อมูล แนวทางการแก้ไข หรือตัวอย่างโค้ดได้ไม่ยาก ด้วยการผสมผสานนี้ ผู้อ่านสามารถจินตนาการถึงการสร้างสรรค์โครงงานที่หลากหลายได้อย่างง่ายดาย เช่น:
    - ระบบควบคุมไฟส่องสว่างในบ้านผ่านสมาร์ตโฟน
    - สถานีตรวจวัดสภาพอากาศขนาดเล็กที่ส่งข้อมูลอุณหภูมิและความชื้นขึ้นเว็บไซต์
    - หุ่นยนต์ขนาดเล็กที่ควบคุมการเคลื่อนที่ด้วยคำสั่ง Python
    - ระบบแจ้งเตือนเมื่อมีผู้บุกรุกผ่านเซ็นเซอร์ตรวจจับความเคลื่อนไหว

มกุฎ ปักษี

- อุปกรณ์สวมใส่ที่เก็บข้อมูลสุขภาพเบื้องต้นและส่งไปยังแอปพลิเคชัน

1.5 เปรียบเทียบสำหรับผู้ใช้ Arduino: ก้าวใหม่ที่น่าลอง

สำหรับผู้ที่คุ้นเคยกับแพลตฟอร์ม Arduino และภาษา C/C++ การมาถึงของ MicroPython บน ESP32 อาจเป็นทั้งโอกาสและความท้าทายใหม่ที่น่าสนใจ Arduino ได้สร้างคุณูปการอย่างใหญ่หลวงในการทำให้การเรียนรู้ไมโครคอนโทรลเลอร์เป็นเรื่องง่ายสำหรับคนจำนวนมาก และ MicroPython ก็มีเป้าหมายเดียวกัน แต่มาพร้อมกับแนวทางที่แตกต่างออกไป

ความแตกต่างหลักระหว่าง Arduino (C/C++) และ MicroPython บน ESP32

คุณสมบัติ	Arduino (C/C++)	MicroPython บน ESP32
ภาษาโปรแกรม	C/C++ (Static Typing)	Python (Dynamic Typing)
การทำงาน	คอมไพล์เป็น Machine Code แล้วอัปโหลด	Interpreter ทำงานบนเฟิร์มแวร์ MicroPython
การพัฒนา	เขียนโค้ด -> คอมไพล์ -> อัปโหลด -> ทดสอบ	เขียนโค้ด -> (อัปโหลดไฟล์ .py) -> ทดสอบผ่าน REPL/รันไฟล์
ความเร็วในการทดลอง	ช้ากว่า เนื่องจากต้องคอมไพล์ใหม่ทุกครั้ง	เร็วกว่ามาก สามารถทดลองโค้ดทีละบรรทัดได้
ความซับซ้อนของภาษา	สูงกว่า โดยเฉพาะเรื่อง Pointer, Memory Management	ต่ำกว่า ไวยากรณ์อ่านง่าย จัดการหน่วยความจำอัตโนมัติ
การจัดการทรัพยากร	ผู้เขียนโค้ดต้องจัดการอย่างระมัดระวัง	มี Garbage Collector ช่วยจัดการหน่วยความจำ

ข้อดีที่ผู้ใช้ Arduino อาจได้รับจาก MicroPython:

- **ลดเวลาในการพัฒนา:** วงจรการแก้ไขโค้ดและทดสอบที่สั้นลงอย่างเห็นได้ชัด
- **ภาษาที่ยืดหยุ่นกว่า:** การจัดการข้อความ (String), รายการข้อมูล (List), และโครงสร้างข้อมูลแบบ Dictionary ใน Python ทำได้ง่ายและมีประสิทธิภาพกว่าใน C/C++ พื้นฐาน
- **การเขียนโปรแกรมเชิงวัตถุ (OOP):** แม้ C++ จะรองรับ OOP แต่ Python ก็มีโครงสร้าง OOP ที่เข้าใจง่ายและเป็นธรรมชาติ

- **ลดความกังวลเรื่อง Memory Management พื้นฐาน:** ช่วยให้ผู้เริ่มต้นโฟกัสไปที่ตรรกะของโปรแกรมได้มากขึ้น

อย่างไรก็ตาม ไม่ได้หมายความว่า MicroPython ดีกว่า Arduino C/C++ ในทุกกรณี การเขียนโปรแกรมด้วย C/C++ โดยตรงยังคงให้ประสิทธิภาพการทำงานที่เร็วกว่าและใช้ทรัพยากรน้อยกว่า เหมาะสำหรับงานที่ต้องการความเร็วสูงมากๆ หรือมีข้อจำกัดด้านหน่วยความจำที่เข้มงวดสุดๆ MicroPython จึงเป็นอีก **ทางเลือกหนึ่ง** ที่น่าสนใจสำหรับโครงงานหลายประเภท โดยเฉพาะโครงงาน IoT ที่เน้นการเชื่อมต่อเครือข่ายและการจัดการข้อมูล ซึ่งความเร็วในการพัฒนาอาจมีความสำคัญมากกว่าประสิทธิภาพระดับไมโครวินาที

1.6 MicroPython กับ ESP32 ในมุมมองของครูอาจารย์

สำหรับครูอาจารย์ผู้สอนวิชาวิทยาการคำนวณ, การออกแบบและเทคโนโลยี, หรือวิชาที่เกี่ยวข้องกับ STEM และ IoT นั้น MicroPython และ ESP32 ถือเป็นเครื่องมือการสอนที่มีศักยภาพสูงมาก

- **ลดอุปสรรคการเรียนรู้โค้ดดิ้ง:** ไวยากรณ์ที่ง่ายของ Python ช่วยให้นักเรียนที่ไม่เคยมีประสบการณ์มาก่อนสามารถเข้าใจแนวคิดการเขียนโปรแกรมได้เร็วขึ้น ลดความกลัวและความรู้สึกว่าโค้ดดิ้งเป็นเรื่องยาก
- **ส่งเสริมการเรียนรู้แบบโครงงานเป็นฐาน (Project-Based Learning):** นักเรียนสามารถนำความรู้ไปสร้างสรรค์โครงงานที่จับต้องได้จริงอย่างรวดเร็ว ตั้งแต่การควบคุม LED ง่ายๆ ไปจนถึงการสร้างอุปกรณ์ IoT ที่เชื่อมต่ออินเทอร์เน็ต ซึ่งช่วยเพิ่มแรงจูงใจและความสนุกสนานในการเรียนรู้
- **เชื่อมโยงกับชีวิตจริง:** โครงงาน IoT ที่สร้างจาก ESP32 และ MicroPython สามารถออกแบบให้แก้ปัญหาในชีวิตประจำวันหรือในโรงเรียนได้ เช่น ระบบตรวจวัดคุณภาพอากาศ, ระบบรดน้ำต้นไม้อัตโนมัติสำหรับแปลงผักของโรงเรียน ทำให้นักเรียนเห็นคุณค่าของสิ่งที่เรียน
- **พัฒนาทักษะการแก้ปัญหาและคิดเชิงคำนวณ:** การออกแบบและแก้ไขโปรแกรมสำหรับควบคุมฮาร์ดแวร์เป็นการฝึกฝนทักษะการคิดวิเคราะห์ การวางแผน และการแก้ปัญหาอย่างเป็นระบบ
- **เครื่องมือที่เข้าถึงง่ายและราคาไม่แพง:** บอร์ด ESP32 มีราคาที่ไม่สูงนัก ทำให้โรงเรียนหรือสถาบันการศึกษาสามารถจัดหาให้นักเรียนใช้งานได้อย่างทั่วถึง

การใช้ MicroPython และ ESP32 ในห้องเรียน จึงไม่เพียงแต่สอนทักษะการเขียนโปรแกรม แต่ยังเป็นการเปิดโลกทัศน์ให้นักเรียนเห็นถึงพลังของเทคโนโลยีในการสร้างสรรค์นวัตกรรมและแก้ปัญหาต่างๆ รอบตัว

1.7 บทสรุป: เตรียมพร้อมสำหรับการเดินทาง

มหาวิทยาลัยราชภัฏ

จากที่กล่าวมาทั้งหมด จะเห็นได้ว่า MicroPython และ ESP32 คือการผสมผสานที่ลงตัวระหว่างความง่ายในการใช้งานของซอฟต์แวร์และความสามารถอันทรงพลังของฮาร์ดแวร์ นับเป็นเครื่องมือที่เปิดโอกาสให้ทุกคนสามารถก้าวเข้าสู่โลกแห่งการสร้างสรรค์โครงงานอิเล็กทรอนิกส์และ Internet of Things ได้อย่างมั่นใจ ไม่ว่าจะมีพื้นฐานมาก่อนหรือไม่ก็ตาม

เสน่ห์ของ MicroPython อยู่ที่ความสามารถในการปลดปล่อยจินตนาการของผู้พัฒนาให้เป็นอิสระจากความซับซ้อนของโค้ด ในขณะที่ ESP32 ก็มอบขุมพลังและช่องทางการเชื่อมต่อที่หลากหลายรอให้ถูกนำไปใช้งาน หนังสือเล่มนี้จะเป็นเสมือนแผนที่นำทาง พาผู้อ่านไปสำรวจดินแดนอันน่าตื่นเต้นนี้ ตั้งแต่การทำความรู้จัก การติดตั้งเครื่องมือ จนถึงการลงมือสร้างโครงงานจริง

ในบทต่อไป เราจะเริ่มต้นการเดินทางภาคปฏิบัติ ด้วยการเตรียมความพร้อมเครื่องมือและอุปกรณ์ที่จำเป็น รวมถึงการติดตั้ง MicroPython ลงบนบอร์ด ESP32 เพื่อให้พร้อมสำหรับการเขียนโปรแกรมแรก ขอเชิญผู้อ่านทุกท่านเตรียมตัวให้พร้อม แล้วมาเริ่มต้นสร้างสรรค์สิ่งประดิษฐ์ด้วยกัน!

บทที่ 2: ติดตั้งและเริ่มต้นใช้งาน MicroPython บน ESP32

หลังจากที่เราได้ทำความรู้จักกับ MicroPython และ ESP32 รวมถึงเหตุผลที่ทำให้ทั้งสองเป็นคู่หูที่ยอดเยี่ยมสำหรับการเริ่มต้นในบทที่ 1 แล้ว ในบทนี้ จะเข้าสู่ภาคปฏิบัติกันอย่างเต็มตัว โดยจะแนะนำขั้นตอนที่จำเป็นทั้งหมด ตั้งแต่การเลือกบอร์ด ESP32, การเตรียมอุปกรณ์, การติดตั้งไดรเวอร์, การติดตั้งเฟิร์มแวร์ MicroPython ลงบนบอร์ด ESP32, ไปจนถึงการตั้งค่าโปรแกรม Thonny IDE และการทดลองเขียนโปรแกรมแรกเพื่อควบคุมฮาร์ดแวร์เบื้องต้น การทำตามขั้นตอนในบทนี้อย่างละเอียดจะช่วยให้ผู้อ่านมีพื้นฐานที่มั่นคงและพร้อมสำหรับการเรียนรู้การเขียนโปรแกรม MicroPython เพื่อสร้างสรรค์โครงงานต่างๆ ในบทต่อๆ ไป

2.1 การเลือกบอร์ด ESP32 สำหรับมือใหม่

ชิป ESP32 ถูกนำไปใช้งานในบอร์ดพัฒนา (Development Board) หลากหลายรูปแบบและหลายผู้ผลิต สำหรับผู้เริ่มต้น การเลือกบอร์ดที่เหมาะสมจะช่วยให้การเรียนรู้ง่ายขึ้น บอร์ดที่แนะนำควรมีลักษณะดังนี้:

- **มีพอร์ต Micro-USB หรือ USB-C:** สำหรับการจ่ายไฟและอัปโหลดโปรแกรมได้สะดวก
- **มีวงจร USB to UART ในตัว:** บอร์ดส่วนใหญ่มักใช้ชิป เช่น CP2102, CP2104 หรือ CH340/CH9102F เพื่อทำหน้าที่นี้ ทำให้สามารถเชื่อมต่อกับคอมพิวเตอร์ผ่านสาย USB ได้โดยตรง
- **มีปุ่ม BOOT (หรือ FLASH) และ EN (หรือ RESET/RST):** ปุ่มเหล่านี้จำเป็นสำหรับการเข้าโหมดดาวน์โหลดเฟิร์มแวร์

- **มี LED ออนบอร์ด (Onboard LED):** อย่างน้อยหนึ่งดวง เพื่อใช้ทดสอบการเขียนโปรแกรมเบื้องต้นได้ง่าย
- **ขา GPIO ถูกต่อออกมาให้ใช้งานง่าย:** ควรมี Pinout Diagram (แผนผังขา) ที่ชัดเจน และขาต่างๆ ควรถูกพิมพ์ชื่อกำกับไว้บนบอร์ดหรือหาข้อมูลได้ง่าย

บอร์ด ESP32 ที่เป็นที่นิยมและเหมาะสำหรับผู้เริ่มต้น ได้แก่:

- **ESP32-DevKitC:** เป็นบอร์ดมาตรฐานจาก Espressif เอง มีหลายเวอร์ชันย่อย (เช่น ใช้ชิป ESP32-WROOM-32, ESP32-WROVER) หาซื้อง่าย มีข้อมูลสนับสนุนเยอะ
- **NodeMCU-32S:** เป็นอีกหนึ่งบอร์ดที่ได้รับความนิยมสูง มีลักษณะคล้ายกับ NodeMCU ที่ใช้ ESP8266 ทำให้ผู้ที่เคยใช้ ESP8266 มาก่อนคุ้นเคยได้ง่าย
- **บอร์ดจากผู้ผลิตอื่นๆ:** เช่น Wemos Lolin D32, Adafruit HUZZAH32, SparkFun ESP32 Thing ซึ่งมักจะมีคุณสมบัติเพิ่มเติมหรือการออกแบบที่เป็นเอกลักษณ์

คำแนะนำ: ก่อนซื้อ ควรตรวจสอบข้อมูลจำเพาะของบอร์ด รีวิวจากผู้ใช้งาน และแหล่งข้อมูลสนับสนุน เพื่อให้ได้บอร์ดที่ตรงกับความต้องการและง่ายต่อการเริ่มต้น

2.2 เตรียมความพร้อมก่อนเริ่มต้น

ก่อนจะลงมือติดตั้งและเขียนโปรแกรม มีสิ่งจำเป็นที่ต้องเตรียมให้พร้อมดังนี้:

1. **บอร์ด ESP32:** ตามที่ได้เลือกไว้ในหัวข้อ 2.1
2. **สาย Micro-USB หรือ USB-C Data Cable:**
    - *ข้อควรระวัง:* ตรวจสอบให้แน่ใจว่าเป็นสายข้อมูล (Data Cable) ไม่ใช่สายชาร์จไฟอย่างเดียว (Charge-only Cable) สายข้อมูลจะสามารถทั้งจ่ายไฟและรับส่งข้อมูลได้ สังเกตได้จากเมื่อเสียบกับคอมพิวเตอร์แล้วควรมีอุปกรณ์ใหม่ถูกตรวจพบ (แม้จะยังไม่ได้ติดตั้งไดรเวอร์ก็ตาม)
3. **คอมพิวเตอร์:** ระบบปฏิบัติการ Windows, macOS, หรือ Linux ก็ได้
4. **การเชื่อมต่ออินเทอร์เน็ต:** สำหรับดาวน์โหลดซอฟต์แวร์และไดรเวอร์ที่จำเป็น

2.3 การติดตั้งไดรเวอร์ USB to UART (ถ้าจำเป็น)

บอร์ด ESP32 ส่วนใหญ่จะใช้ชิปแปลงสัญญาณ USB เป็น UART (Universal Asynchronous Receiver/Transmitter) เพื่อให้คอมพิวเตอร์สามารถสื่อสารกับชิป ESP32 ผ่านทางพอร์ต USB ได้ ชิปที่นิยมใช้คือ CP210x (จาก Silicon Labs) หรือ CH340/CH9102F (จาก WCH)

- **สำหรับ Windows:**

สมาพัฒ ปัทมี

- โดยทั่วไป Windows 10/11 อาจรู้จักชิปเหล่านี้และติดตั้งไดรเวอร์ให้โดยอัตโนมัติเมื่อเสียบบอร์ด ESP32 ครั้งแรก
- หากไม่รู้จัก (เช่น ใน Device Manager แสดงเป็น Unknown Device หรือมีเครื่องหมายตกใจสีเหลือง) จำเป็นต้องดาวน์โหลดและติดตั้งไดรเวอร์ด้วยตนเอง:
    - **CP210x:** ดาวน์โหลดจากเว็บไซต์ Silicon Labs (ค้นหา "CP210x USB to UART Bridge VCP Drivers")
    - **CH340/CH9102F:** ดาวน์โหลดจากเว็บไซต์ผู้ผลิต WCH (ค้นหา "CH341SER.EXE" หรือ "CH340 driver")
- หลังติดตั้งไดรเวอร์และเสียบบอร์ด ESP32 ควรจะเห็นพอร์ต COM ใหม่ปรากฏขึ้นใน Device Manager (อยู่ใต้ Ports (COM & LPT)) เช่น Silicon Labs CP210x USB to UART Bridge (COM3) หรือ USB-SERIAL CH340 (COM4) หรือ COM เลขอื่น ๆ แล้วแต่เครื่องนั้น ๆ ให้จดจำหมายเลขพอร์ต COM นี้ไว้

- **สำหรับ macOS:**
    - macOS ส่วนใหญ่มักจะรู้จักชิป CP210x โดยไม่ต้องติดตั้งไดรเวอร์เพิ่มเติม
    - สำหรับชิป CH340/CH9102F อาจจำเป็นต้องติดตั้งไดรเวอร์จากผู้ผลิต WCH (ค้นหา "CH34x_Install_V1.x.pkg" หรือไดรเวอร์สำหรับ Mac รุ่นล่าสุด)
    - หลังจากเสียบบอร์ด ESP32 พอร์ตจะปรากฏในรูปแบบ /dev/cu.usbserial-xxxx หรือ /dev/cu.SLAB_USBtoUART (สำหรับ CP210x) หรือ /dev/cu.wchusbserialxxxx (สำหรับ CH340) สามารถตรวจสอบได้โดยเปิด Terminal แล้วพิมพ์คำสั่ง ls /dev/cu.*
- **สำหรับ Linux:**
    - Linux Kernel ส่วนใหญ่รองรับชิปทั้ง CP210x และ CH340/CH9102F โดยไม่ต้องติดตั้งไดรเวอร์เพิ่มเติม
    - เมื่อเสียบบอร์ด ESP32 พอร์ตมักจะปรากฏในรูปแบบ /dev/ttyUSB0, /dev/ttyUSB1 หรือคล้ายกัน สามารถตรวจสอบได้โดยเปิด Terminal แล้วพิมพ์คำสั่ง ls /dev/ttyUSB* หรือ dmesg | grep tty หลังจากเสียบบอร์ด

สำเนาใช้งานปัญญา

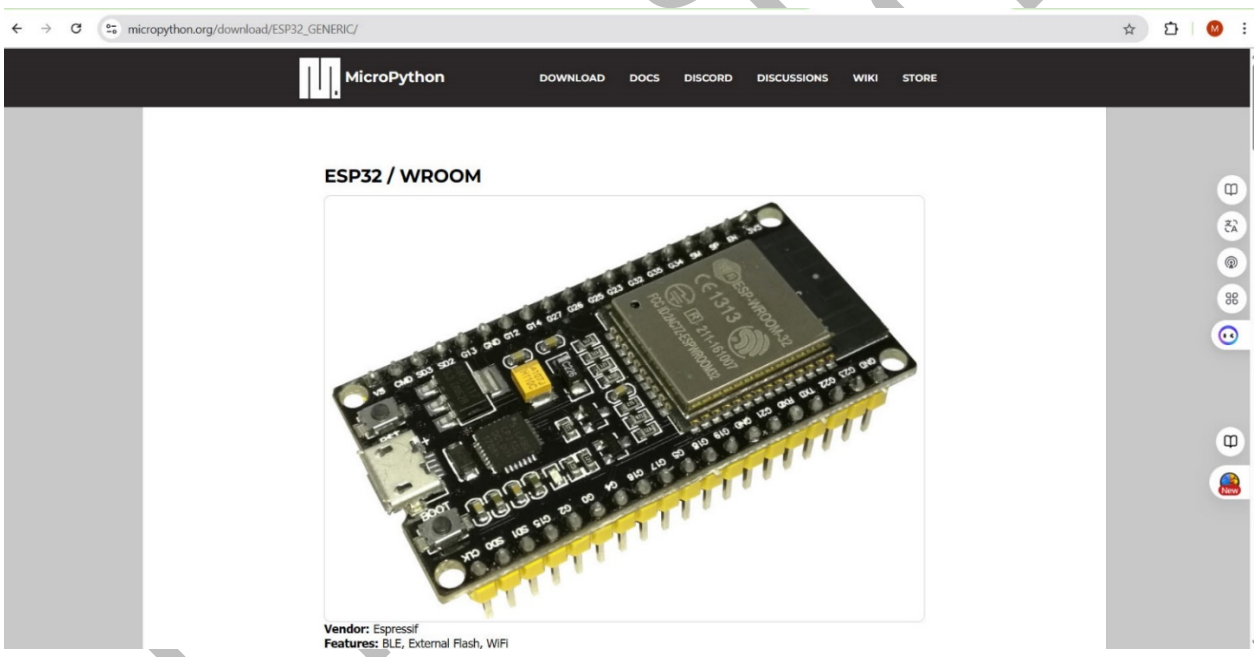
- *ข้อควรทราบสำหรับ Linux:* ผู้ใช้อาจต้องเพิ่มสิทธิ์ในการเข้าถึงพอร์ตอนุกรม โดยการเพิ่ม user เข้าไปในกลุ่ม dialout (หรือ tty ในบาง Distribution) ด้วยคำสั่ง sudo usermod -a -G dialout $USER (แล้ว logout และ login ใหม่ หรือ reboot)

2.4 การติดตั้งเฟิร์มแวร์ MicroPython ลงบน ESP32

เฟิร์มแวร์ (Firmware) คือซอฟต์แวร์ที่ทำงานอยู่บนตัวฮาร์ดแวร์โดยตรง ในที่นี้คือระบบ MicroPython ที่จะทำให้ ESP32 สามารถเข้าใจและรันโค้ด Python ได้ โดยปกติบอร์ด ESP32 ที่ซื้อมาใหม่จะยังไม่มีเฟิร์มแวร์ MicroPython ติดตั้งอยู่ หรืออาจจะมีเฟิร์มแวร์อื่น (เช่น AT Command firmware หรือ Arduino bootloader) ดังนั้น จึงจำเป็นต้องติดตั้งเฟิร์มแวร์ MicroPython ก่อน

2.4.1 ดาวน์โหลดเฟิร์มแวร์ MicroPython

1. ไปที่เว็บไซต์ทางการของ MicroPython: https://micropython.org/download/esp32/



2. มองหาส่วน "Firmware for ESP32 boards"
3. ดาวน์โหลดไฟล์เฟิร์มแวร์เวอร์ชันล่าสุดที่เป็น **Stable** (แนะนำ) หรือ Daily builds (หากต้องการลองฟีเจอร์ใหม่ล่าสุด แต่อาจไม่เสถียร) ไฟล์เฟิร์มแวร์สำหรับ ESP32 ทั่วไปจะมีนามสกุล .bin (เช่น ESP32_GENERIC-YYYYMMDD-vx.xx.x.bin)

คำแนะนำ: สำหรับบอร์ด ESP32 ทั่วไปที่ใช้ชิป ESP32-WROOM หรือ ESP32-SOLO ให้เลือกเฟิร์มแวร์ GENERIC หากใช้บอร์ดที่มีคุณสมบัติพิเศษ เช่น SPIRAM (PSRAM) อาจมีเฟิร์มแวร์เฉพาะ เช่น GENERIC_SPIRAM ควรตรวจสอบสเปกของบอร์ดที่ใช้

มานพ ปัทโย

- On Windows, the port name is usually similar to `COM4`.

Flashing

Then deploy the firmware to the board, starting at address 0x1000:

```
esptool.py --baud 460800 write_flash 0x1000 ESP32_BOARD_NAME-DATE-VE
RSION.bin
```

Replace `ESP32_BOARD_NAME-DATE-VERSION.bin` with the `.bin` file downloaded from this page.

As above, if `esptool.py` can't automatically detect the serial port then you can pass it explicitly on the command line instead. For example:

```
esptool.py --port PORTNAME --baud 460800 write_flash 0x1000 ESP32_BO
ARD_NAME-DATE-VERSION.bin
```

Troubleshooting

If flashing starts and then fails partway through, try removing the `--baud 460800` option to flash at the slower default speed.

If these steps don't work, consult the MicroPython ESP32 Troubleshooting steps and the esptool documentation.

Important: From the options below, download the `.bin` file for your board.

Firmware

Releases

v1.25.0 (2025-04-15) .bin / [.app-bin] / [.elf] / [.map] / [Release notes] (latest)
v1.24.1 (2024-11-29) .bin / [.app-bin] / [.elf] / [.map] / [Release notes]
v1.24.0 (2024-10-25) .bin / [.app-bin] / [.elf] / [.map] / [Release notes]
v1.23.0 (2024-06-02) .bin / [.app-bin] / [.elf] / [.map] / [Release notes]
v1.22.2 (2024-02-22) .bin / [.app-bin] / [.elf] / [.map] / [Release notes]
v1.22.1 (2024-01-05) .bin / [.app-bin] / [.elf] / [.map] / [Release notes]
v1.22.0 (2023-12-27) .bin / [.app-bin] / [.elf] / [.map] / [Release notes]
v1.21.0 (2023-10-05) .bin / [.app-bin] / [.elf] / [.map] / [Release notes]
v1.20.0 (2023-04-26) .bin / [.elf] / [.map] / [Release notes]
v1.19.1 (2022-06-18) .bin / [.elf] / [.map] / [Release notes]
v1.18 (2022-01-17) .bin / [.elf] / [.map] / [Release notes]
v1.17 (2021-09-02) .bin / [.elf] / [.map] / [Release notes]
v1.16 (2021-06-23) .bin / [.elf] / [.map] / [Release notes]
v1.15 (2021-04-18) .bin / [.elf] / [.map] / [Release notes]
v1.14 (2021-02-02) .bin / [.elf] / [.map] / [Release notes]
v1.13 (2020-09-02) .bin / [.elf] / [.map] / [Release notes]
v1.12 (2019-12-20) .bin / [.elf] / [.map] / [Release notes]

Preview builds

v1.26.0-preview.158.g5cfafb73d (2025-05-28) .bin / [.app-bin] / [.elf] / [.map]
v1.26.0-preview.148.g49f81d504 (2025-05-22) .bin / [.app-bin] / [.elf] / [.map]
v1.26.0-preview.146.g7f6fedef2 (2025-05-21) .bin / [.app-bin] / [.elf] / [.map]
v1.26.0-preview.142.ga1ee42cd3 (2025-05-21) .bin / [.app-bin] / [.elf] / [.map]
(These are automatic builds of the development branch for the next release)

Firmware (Support for OTA)

Releases

v1.25.0 (2025-04-15) .bin / [.app-bin] / [.elf] / [.map] / [Release notes] (latest)
v1.24.1 (2024-11-29) .bin / [.app-bin] / [.elf] / [.map] / [Release notes]
v1.24.0 (2024-10-25) .bin / [.app-bin] / [.elf] / [.map] / [Release notes]
v1.23.0 (2024-06-02) .bin / [.app-bin] / [.elf] / [.map] / [Release notes]
v1.22.2 (2024-02-22) .bin / [.app-bin] / [.elf] / [.map] / [Release notes]
v1.22.1 (2024-01-05) .bin / [.app-bin] / [.elf] / [.map] / [Release notes]
v1.22.0 (2023-12-27) .bin / [.app-bin] / [.elf] / [.map] / [Release notes]
v1.21.0 (2023-10-05) .bin / [.app-bin] / [.elf] / [.map] / [Release notes]
v1.20.0 (2023-04-26) .bin / [.elf] / [.map] / [Release notes]
v1.19.1 (2022-06-18) .bin / [.elf] / [.map] / [Release notes]
v1.18 (2022-01-17) .bin / [.elf] / [.map] / [Release notes]

เครื่องมือที่ใช้ในการอัปโหลดเฟิร์มแวร์ มีทั้งแบบพิมพ์คำสั่ง Command-Line และ GUI

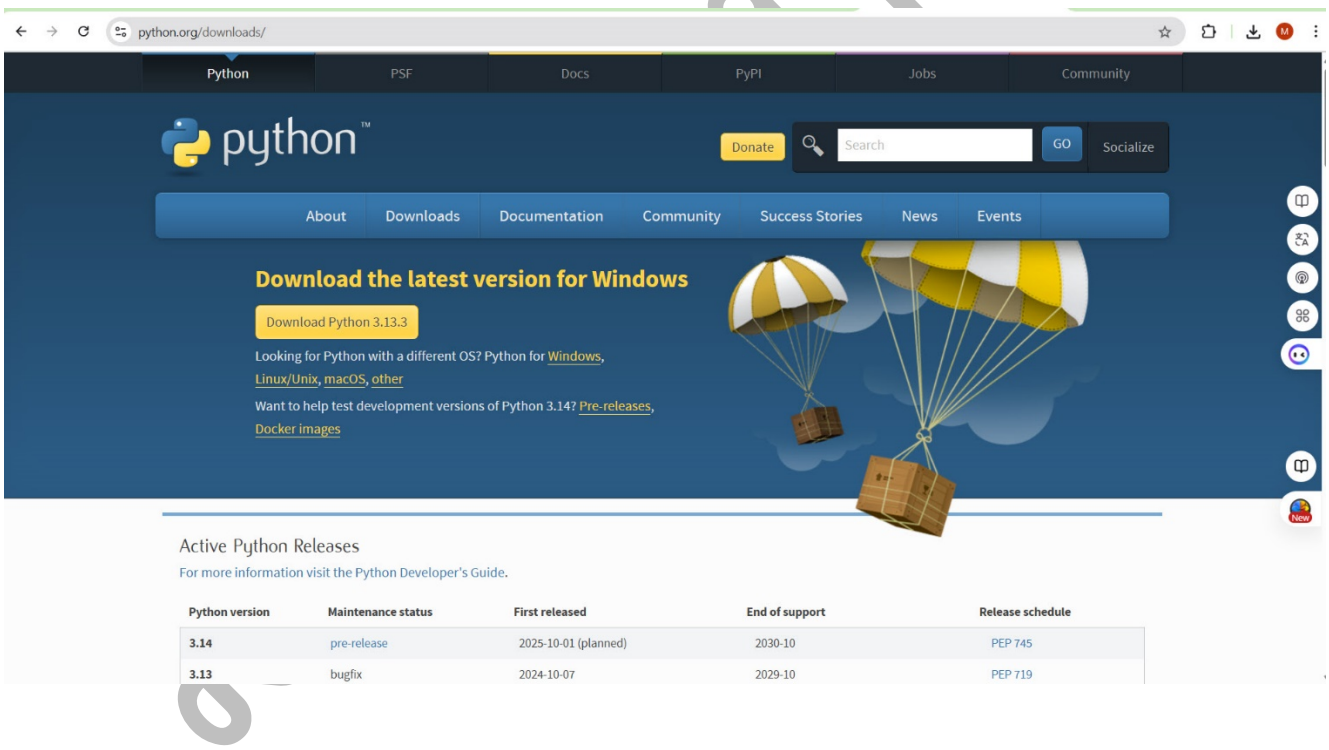
เลือกใช้งานได้ตามถนัดหากถนัดแบบพิมพ์คำสั่งก็ต้องติดตั้ง esptool.py แต่ถ้าหากว่าการพิมพ์คำสั่งมันดูยุ่งยากซับซ้อนเกินไป ก็ให้ข้ามขั้นตอนนี้ไป ดาวน์โหลดโปรแกรมแบบ GUI มาติดตั้งซึ่งใช้งานง่ายกว่า

แต่อย่างไรก็ดี ต้องติดตั้ง python ตามข้อ1 ก่อน ส่วน esptool.py ไม่จำเป็นต้องติดตั้งก็ได้

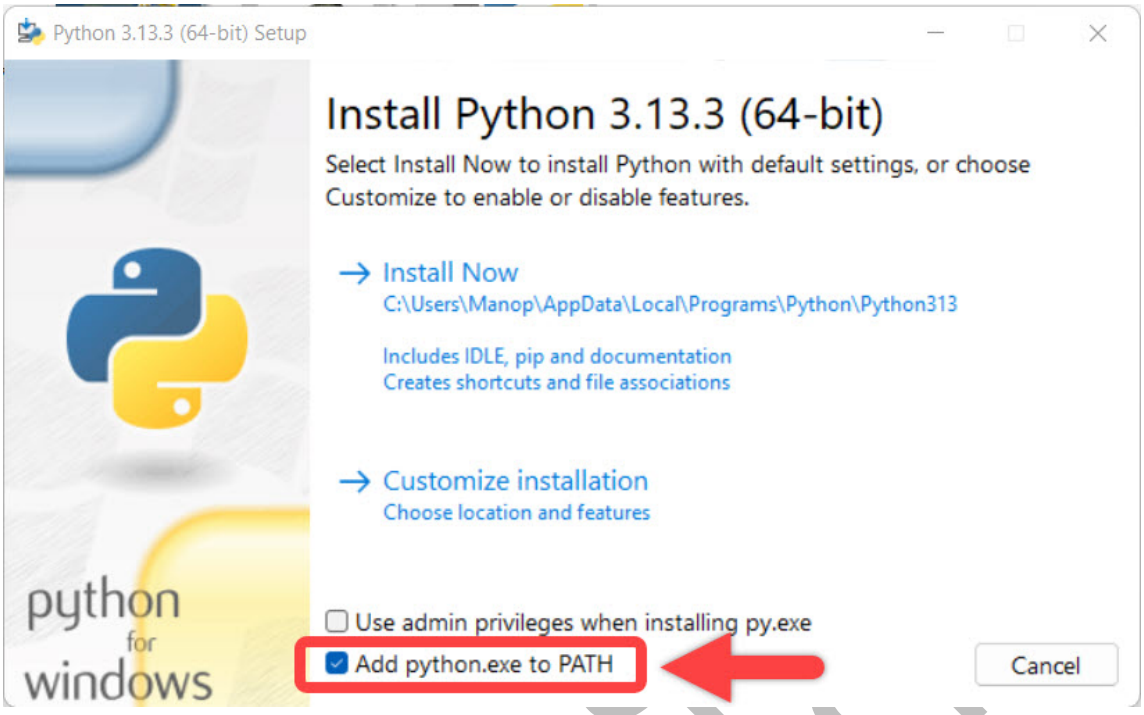
2.4.2 ติดตั้ง Python และ esptool.py

esptool.py เป็นเครื่องมือที่พัฒนาด้วย Python สำหรับสื่อสารกับ Bootloader ของชิป Espressif (ESP8266, ESP32, ESP32-S2, etc.) เพื่ออัปโหลดเฟิร์มแวร์, อ่านข้อมูลจากชิป และอื่นๆ

1. **ต้องมี Python ติดตั้งบนคอมพิวเตอร์:** หากยังไม่มี ให้ดาวน์โหลดและติดตั้ง Python 3.x จาก https://www.python.org/downloads/

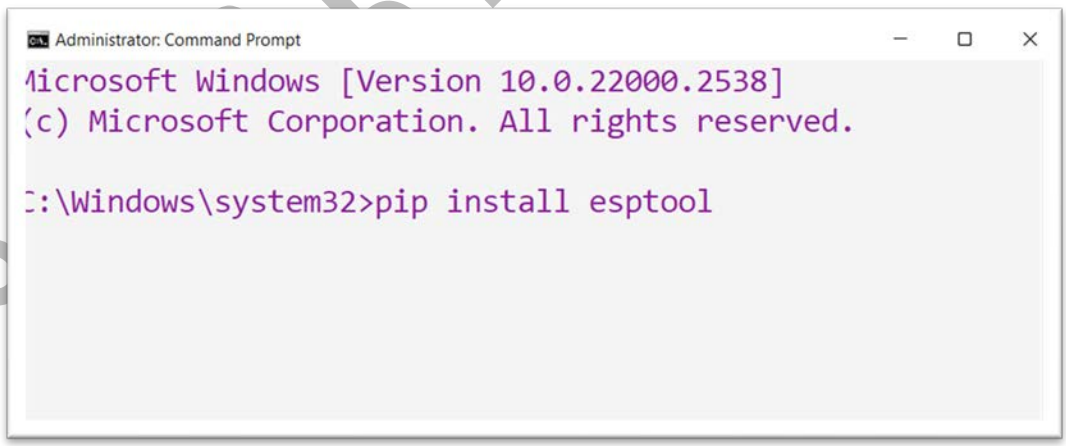


ในระหว่างการติดตั้งบน Windows แนะนำให้เลือก "Add Python to PATH"



2. **ติดตั้ง esptool.py ผ่าน pip:** เปิด Command Prompt (Windows) หรือ Terminal (macOS/Linux) แล้วพิมพ์คำสั่ง:

pip install esptool แล้วกดปุ่ม **Enter**



เมื่อติดตั้งเสร็จแล้วจะมีข้อความแจ้งในหน้าต่าง พร้อมกับแสดงเวอร์ชันที่ติดตั้ง

```
Administrator: Command Prompt
Downloading ecdsa-0.19.1-py2.py3-none-any.whl (150 kB)
Downloading pyserial-3.5-py2.py3-none-any.whl (90 kB)
Downloading PyYAML-6.0.2-cp313-cp313-win_amd64.whl (156 kB)
Downloading reedsolo-1.7.0-py3-none-any.whl (32 kB)
Downloading intelhex-2.3.0-py2.py3-none-any.whl (50 kB)
Downloading bitarray-3.4.2-cp313-cp313-win_amd64.whl (141 kB)
Downloading cffi-1.17.1-cp313-cp313-win_amd64.whl (182 kB)
Downloading six-1.17.0-py2.py3-none-any.whl (11 kB)
Downloading pycparser-2.22-py3-none-any.whl (117 kB)
Building wheels for collected packages: esptool
  Building wheel for esptool (pyproject.toml) ... done
  Created wheel for esptool: filename=esptool-4.8.1-py3-none-any.whl size=530968 sha256=97c6dd769341f7a8129497c58db0f6
d09361ebe5614911ddc4503e1c8bc346a7
  Stored in directory: c:\users\manop\appdata\local\pip\cache\wheels\19\a9\df\aaee748849b28e4b4a0f52bdcf4e16106e74a11e
a9d930d31b
Successfully built esptool
Installing collected packages: reedsolo, pyserial, intelhex, bitarray, six, PyYAML, pycparser, bitstring, ecdsa, cffi,
 cryptography, esptool
Successfully installed PyYAML-6.0.2 bitarray-3.4.2 bitstring-4.3.1 cffi-1.17.1 cryptography-45.0.3 ecdsa-0.19.1 esptoo
l-4.8.1 intelhex-2.3.0 pycparser-2.22 pyserial-3.5 reedsolo-1.7.0 six-1.17.0

[notice] A new release of pip is available: 25.0.1 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Windows\system32>
```

หากต้องการหากต้องการอัปเกรดเป็นเวอร์ชันล่าสุด พิมพ์ **python.exe -m pip install –upgrade pip**

เสร็จแล้วกดปุ่ม Enter อีกครั้งแล้วรอจนกว่าจะมีข้อความแจ้ง

```
Administrator: Command Prompt
Requirement already satisfied: cffi>=1.14 in c:\users\manop\appdata\local\programs\pyth
on\python313\lib\site-packages (from cryptography>=2.1.4->esptool) (1.17.1)
Requirement already satisfied: six>=1.9.0 in c:\users\manop\appdata\local\programs\pyth
on\python313\lib\site-packages (from ecdsa>=0.16.0->esptool) (1.17.0)
Requirement already satisfied: pycparser in c:\users\manop\appdata\local\programs\pytho
n\python313\lib\site-packages (from cffi>=1.14->cryptography>=2.1.4->esptool) (2.22)

[notice] A new release of pip is available: 25.0.1 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Windows\system32>python.exe -m pip install --upgrade pip
Requirement already satisfied: pip in c:\users\manop\appdata\local\programs\python\pyth
on313\lib\site-packages (25.0.1)
Collecting pip
  Downloading pip-25.1.1-py3-none-any.whl.metadata (3.6 kB)
Downloading pip-25.1.1-py3-none-any.whl (1.8 MB)
   ---------------------------------------- 1.8/1.8 MB 23.4 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 25.0.1
    Uninstalling pip-25.0.1:
      Successfully uninstalled pip-25.0.1
Successfully installed pip-25.1.1

C:\Windows\system32>
```

3. **หากต้องการตรวจสอบเวอร์ชันให้พิมพ์ esptool version แล้วกด Enter**

2.4.3 การเชื่อมต่อบอร์ดและค้นหาพอร์ต

1. เสียบบอร์ด ESP32 เข้ากับคอมพิวเตอร์ผ่านสาย USB
2. ค้นหาหมายเลขพอร์ต (Port) ที่บอร์ด ESP32 เชื่อมต่ออยู่ (ตามวิธีในหัวข้อ 2.3)
   - Windows: เช่น COM3, COM4
   - macOS: เช่น /dev/cu.usbserial-xxxx หรือ /dev/cu.SLAB_USBtoUART
   - Linux: เช่น /dev/ttyUSB0

2.4.4 การลบข้อมูลใน Flash (Erasing the flash)

ก่อนติดตั้งเฟิร์มแวร์ใหม่ แนะนำให้ลบข้อมูลทั้งหมดในหน่วยความจำ Flash ของ ESP32 ก่อน เพื่อป้องกันปัญหาที่อาจเกิดจากข้อมูลเก่าที่หลงเหลืออยู่ ในส่วนของการ ลบเฟิร์มแวร์นั้น สามารถทำได้ทั้งวิธี Command Line หรือ ใช้โปรแกรมสำเร็จรูปแบบ GUI ก็ได้เหมือนกัน ส่วนนนี้แล้วแต่คนถนัดการใช้โปรแกรมแบบ GUI ต้องดาวน์โหลดโปรแกรมมาก่อน แต่ถ้าจะใช้แบบ command line ก็เปิด terminal พิมพ์คำสั่งได้เลย แต่ต้องดูหมายเลข port ที่เชื่อมต่อด้วย

1. เปิด Command Prompt หรือ Terminal
2. พิมพ์คำสั่ง esptool.py โดยแทนที่ [PORT] ด้วยหมายเลขพอร์ตที่ถูกต้องด้วย

esptool.py --chip esp32 --port [PORT] erase_flash

ตัวอย่างสำหรับ Windows: esptool.py --chip esp32 --port COM3 erase_flash

ตัวอย่างสำหรับ macOS/Linux: esptool.py --chip esp32 --port /dev/ttyUSB0 erase_flash

นี่คือการแฟลชเฟิร์มแวร์ด้วยการพิมพ์คำสั่ง Command-Line

หากไม่ถนัดการพิมพ์คำสั่งแบบ **command line** ก็มีทางเลือก:

🔧 **ทางเลือกการใช้เครื่องมือแบบ GUI ที่ใช้งานง่ายกว่า**

หากไม่ต้องการพิมพ์คำสั่ง ก็ใช้โปรแกรม GUI เหล่านี้แทนได้เลย จะปลอดภัย และ ใช้งานง่ายกว่า

✅ **1. ESPHome-Flasher**

- มีปุ่ม "Erase Flash" โดยตรง
- ใช้งานง่าย เหมาะกับผู้ไม่ถนัด Command Line

📌 ใช้ได้กับ: ESP32 / ESP8266

🔗 https://github.com/esphome/esphome-flasher/releases

✅ **2. ESP32 Flash Download Tool (Espressif Official)**

- มี checkbox ให้เลือก **“Erase Flash”**
- รองรับพาร์ติชันแบบละเอียด

📌 ใช้ได้กับ: ESP32 โดยตรง

🔗 https://www.espressif.com/en/support/download/other-tools

✅ **3. NodeMCU PyFlasher (สำหรับ ESP8266)**

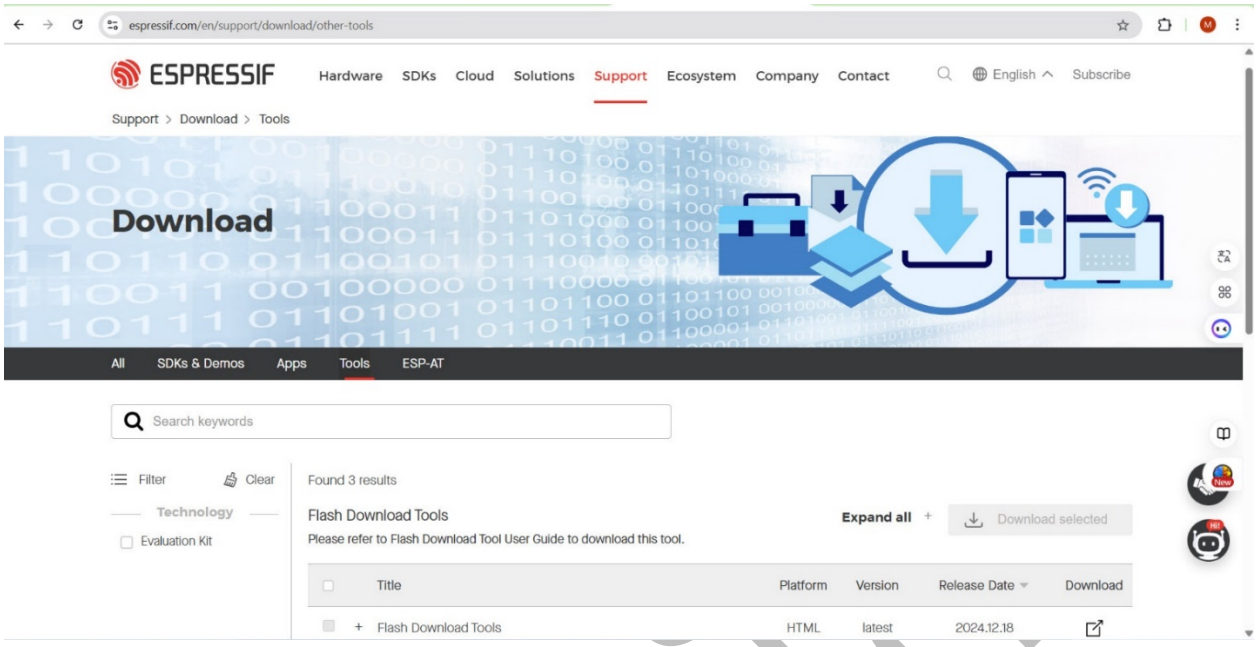
- มีปุ่ม **“Erase Flash before flashing”**
- ใช้งานง่ายมากในสาย MicroPython

แต่ในหนังสือเล่มนี้ผู้เขียนขอเลือกใช้ **ESP32 Flash Download Tool** มาเป็นตัวอย่าง

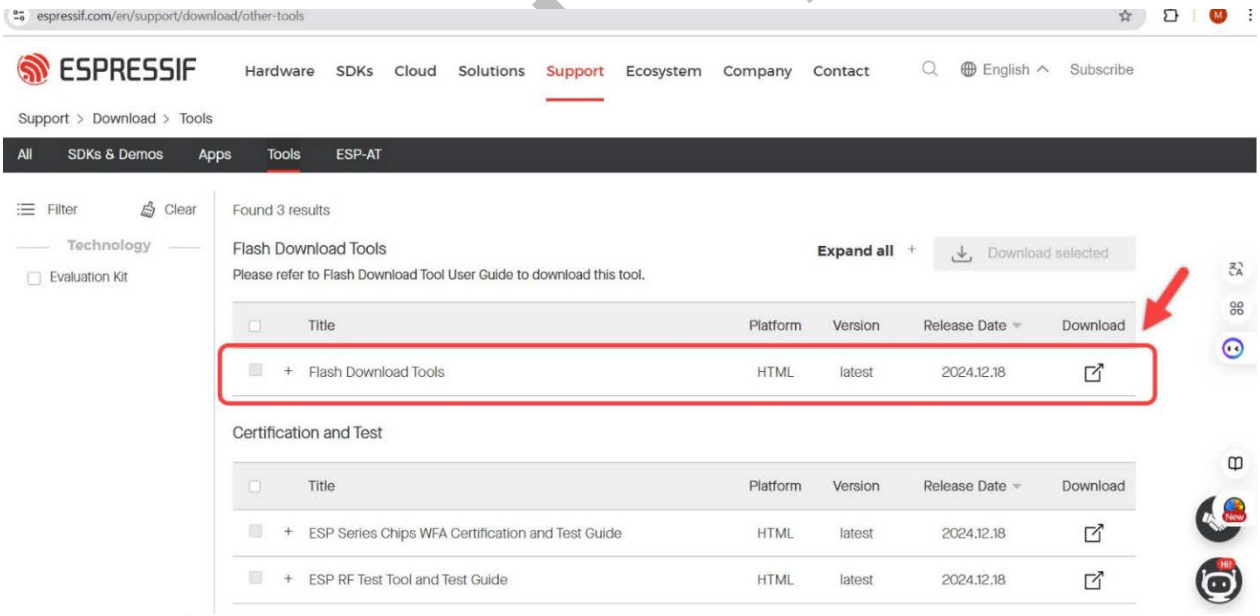
เพราะเป็นโปรแกรมของ **Espressif** เอง

มวลชน ปัญญา

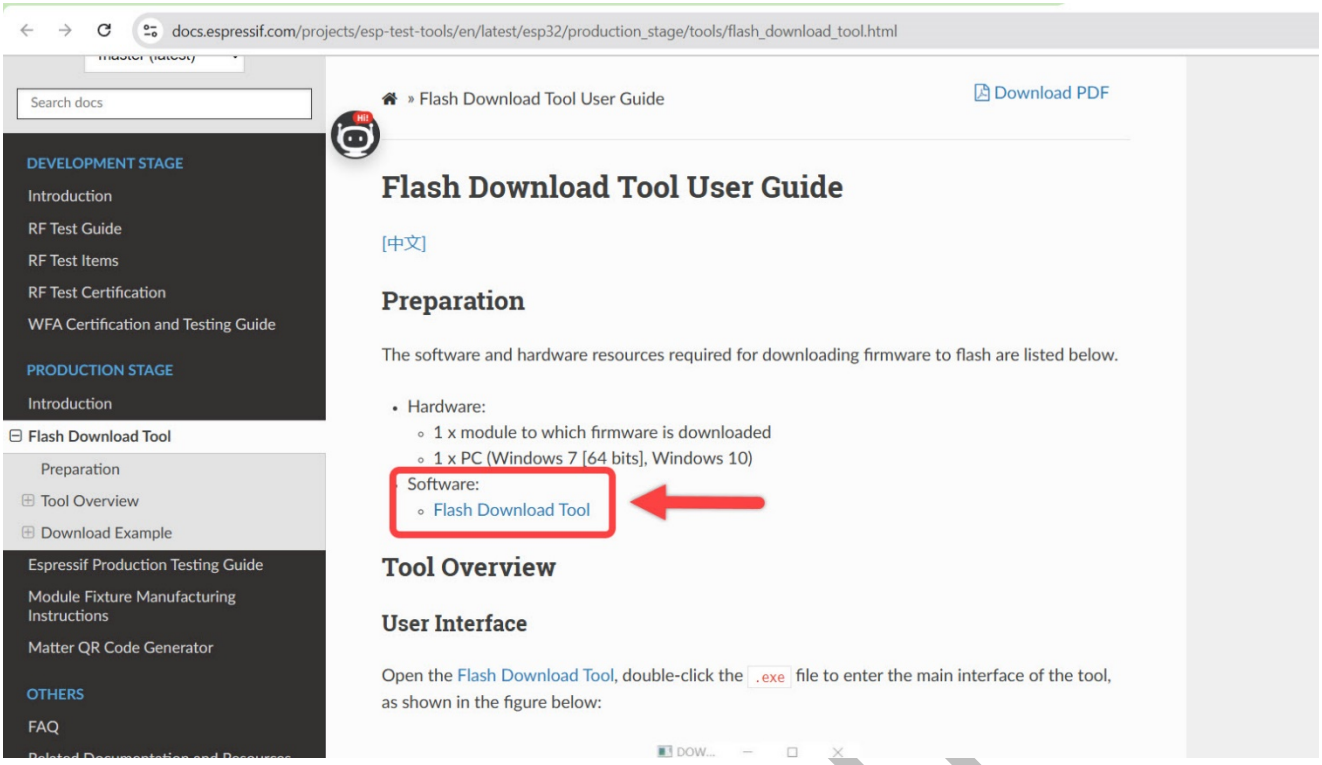
ให้ผู้อ่านเข้าไปในเว็บไซต์ https://www.espressif.com/en/support/download/other-tools



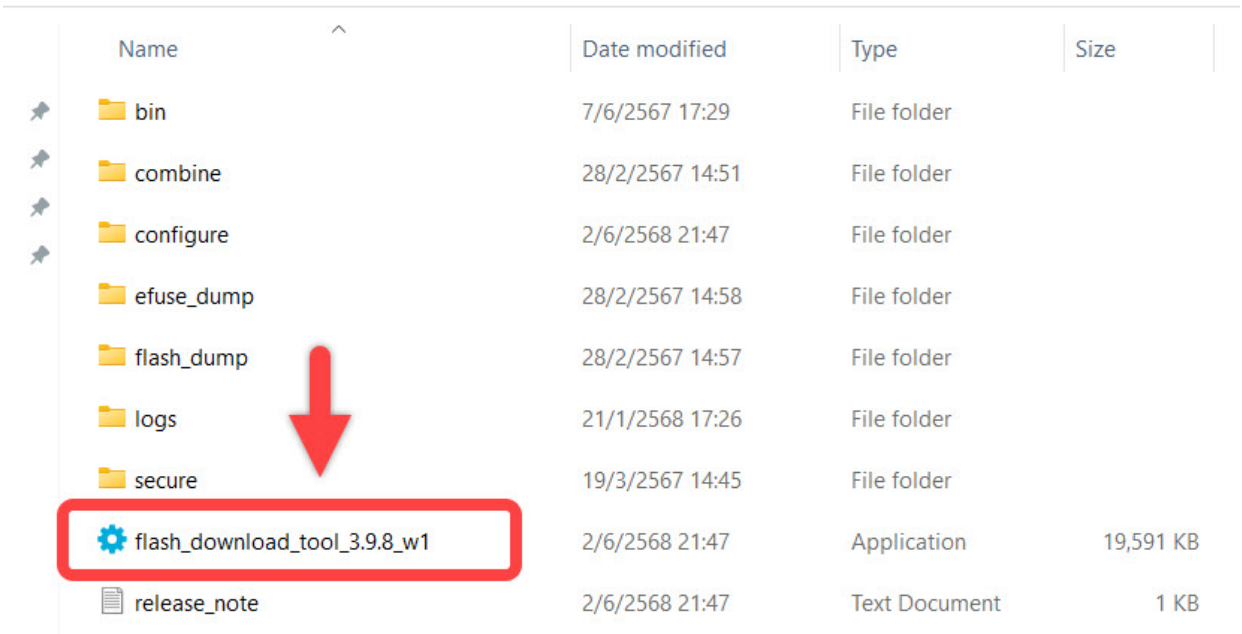
เมื่อเข้าไปที่หน้าเว็บไซต์จะพบกับหน้าตาแบบนี้



ให้คลิกที่ flash Download Tools

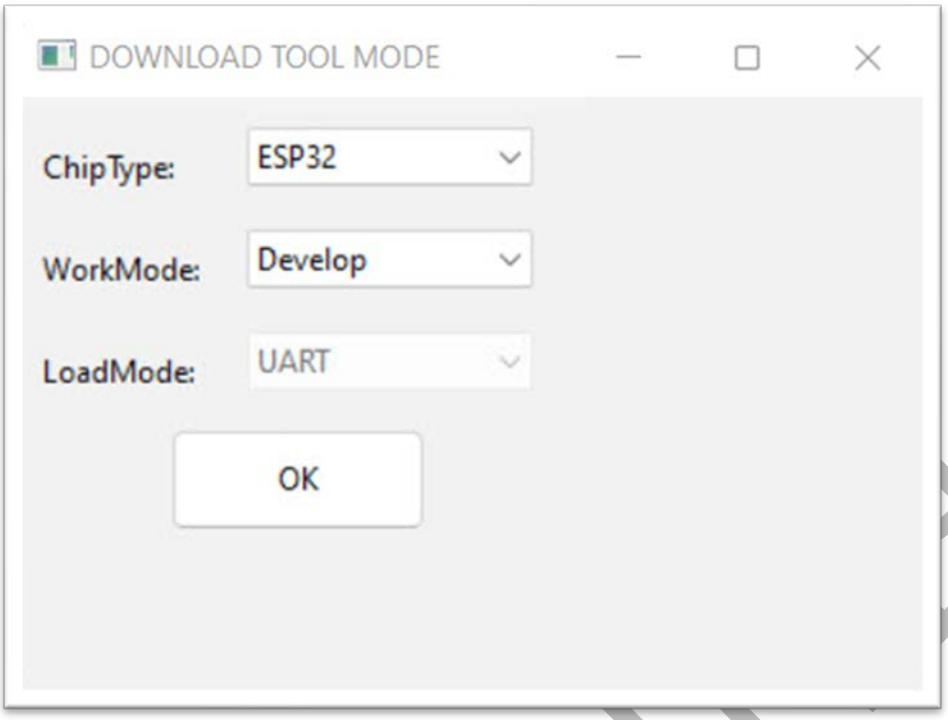


คลิกที่ Flash Download Tool เพื่อดาวน์โหลดโปรแกรม



Name	Date modified	Type	Size
bin	7/6/2567 17:29	File folder	
combine	28/2/2567 14:51	File folder	
configure	2/6/2568 21:47	File folder	
efuse_dump	28/2/2567 14:58	File folder	
flash_dump	28/2/2567 14:57	File folder	
logs	21/1/2568 17:26	File folder	
secure	19/3/2567 14:45	File folder	
flash_download_tool_3.9.8_w1	2/6/2568 21:47	Application	19,591 KB
release_note	2/6/2568 21:47	Text Document	1 KB

เมื่อแตกไฟล์แล้วจะได้ไฟล์ตามภาพ เปิดไฟล์โปรแกรมที่ลูกศรชี้ด้วยการดับเบิ้ลคลิก



หน้าตาโปรแกรมเริ่มแรกจะเป็นแบบนี้ ให้เลือก Chipset ให้ตรงกับที่เราจะใช้งาน

และในส่วนของ **WorkMode** นั้นมี 2 โหมด คือ **Develop** และ **Factory** ขออธิบายเพิ่มเติมว่า

ความแตกต่างระหว่างโหมด Factory และ Develop ในโปรแกรม ESP32 Flash Download Tool

เมื่อเปิดใช้งานโปรแกรม **ESP32 Flash Download Tool** ซึ่งเป็นเครื่องมืออย่างเป็นทางการจากบริษัท **Espressif** ผู้พัฒนาไมโครคอนโทรลเลอร์ตระกูล ESP32 ผู้ใช้จะพบเมนูให้เลือก **Work Mode** ซึ่งประกอบด้วยสองตัวเลือกหลัก คือ **Factory Mode** และ **Develop Mode** โดยแต่ละโหมดมีวัตถุประสงค์ในการใช้งานแตกต่างกันอย่างชัดเจน ดังนี้

1. Develop Mode (โหมดนักพัฒนา)

Develop Mode เป็นโหมดที่ออกแบบมาสำหรับผู้พัฒนาเฟิร์มแวร์ทั่วไปหรือผู้ใช้งาน ESP32 ในระดับบุคคลหรือองค์กรขนาดเล็ก โดยมีคุณสมบัติที่เปิดให้ผู้ใช้สามารถกำหนดรายละเอียดต่าง ๆ ได้อย่างยืดหยุ่น เช่น:

- การเลือกพอร์ต (COM port) และความเร็ว Baud Rate
- การกำหนดไฟล์เฟิร์มแวร์ .bin หลายไฟล์ เช่น bootloader, partition table, firmware
- การเลือกตำแหน่งหน่วยความจำ (offset) ของแต่ละไฟล์
- การลบข้อมูลในหน่วยความจำ Flash ทั้งหมด (Erase Flash) ก่อนแฟลช
- การแสดง Log และสถานะการแฟลชแบบเรียลไทม์

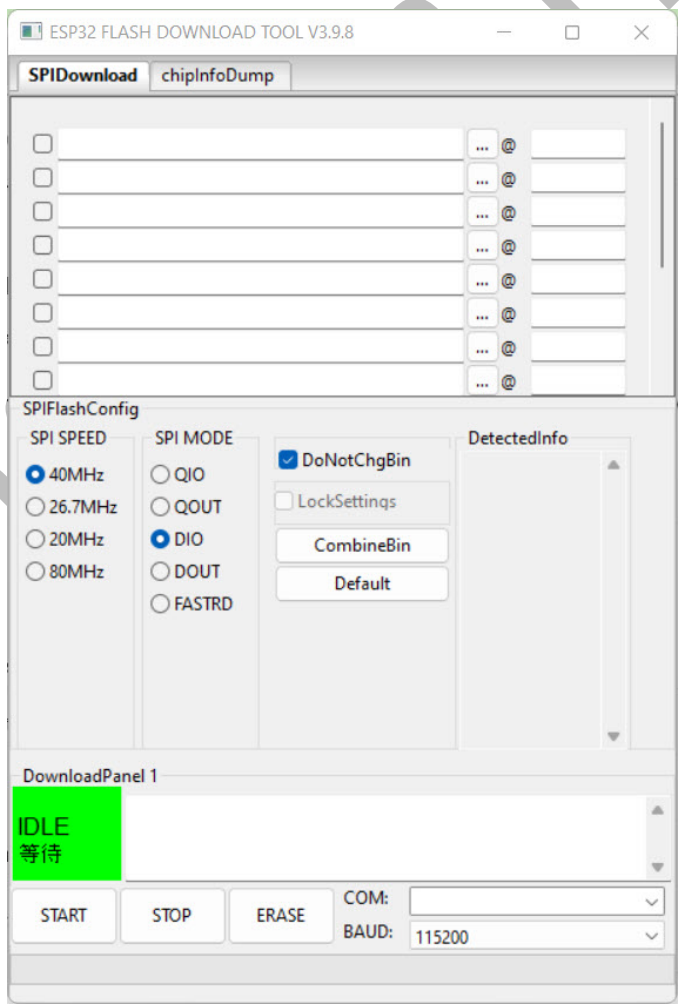
โหมดนี้เหมาะกับงานพัฒนา ทดลอง แก้ไขโค้ด หรือแฟลชเฟิร์มแวร์ด้วยตนเองในสถานการณ์ทั่วไป เช่น การอัปโหลดเฟิร์มแวร์ MicroPython, Arduino หรือเฟิร์มแวร์ที่คอมไพล์เองจาก ESP-IDF

2. Factory Mode (โหมดโรงงาน)

Factory Mode เป็นโหมดเฉพาะทางที่ออกแบบมาสำหรับกระบวนการผลิตในระดับอุตสาหกรรม โดยรองรับการเขียนเฟิร์มแวร์ลงในอุปกรณ์หลายตัวแบบต่อเนื่องอัตโนมัติ (Mass Production) โหมดนี้สามารถเชื่อมต่อกับระบบสายพานการผลิต หรือเครื่องเขียนเฟิร์มแวร์ได้ และมักใช้งานร่วมกับสคริปต์หรือไฟล์ควบคุมเฉพาะ เช่น:

- การกำหนดหมายเลข MAC Address หรือ Serial Number ที่ไม่ซ้ำกันในแต่ละอุปกรณ์
- การบันทึกหรืออ่านข้อมูลเฉพาะทางจาก Flash หลังแฟลชเสร็จ
- การควบคุมการเขียนแฟลชแบบไม่มีการแสดง GUI (Headless)

โหมดนี้เหมาะสำหรับโรงงานหรือผู้ผลิตสินค้าจำนวนมากที่ต้องการกระบวนการผลิตอัตโนมัติ รวดเร็ว และควบคุมคุณภาพอย่างเข้มงวด เมื่อเลือกแล้วกดปุ่ม OK จะได้หน้าต่างใหม่ตามภาพ



การตั้งค่าพารามิเตอร์สำคัญใน ESP32 Flash Download Tool

ในการใช้งานโปรแกรม ESP32 Flash Download Tool ผู้ใช้จำเป็นต้องกำหนดค่าพารามิเตอร์ต่าง ๆ ให้เหมาะสมกับเฟิร์มแวร์และบอร์ด ESP32 ที่ใช้งาน โดยเฉพาะในส่วนของ **SPIFlashConfig** ซึ่งมีผลโดยตรงต่อความเร็วและความเข้ากันได้ของการแฟลชเฟิร์มแวร์

1. SPI SPEED (ความเร็วในการส่งข้อมูลแฟลช)

ตัวเลือกนี้กำหนดความเร็วของสัญญาณ SPI ที่ใช้ในการเขียนข้อมูลไปยัง Flash

ค่า	**ความหมาย**	**ข้อแนะนำ**
20 MHz	ความเร็วต่ำสุด	ใช้ในกรณีที่แฟลชไม่เสถียร
26.7 MHz	ความเร็วกลาง	ใช้งานได้ทั่วไป
40 MHz	ความเร็วมาตรฐาน **(ค่าเริ่มต้น)**	แนะนำสำหรับการใช้งานทั่วไป
80 MHz	ความเร็วสูงสุด	ใช้ในกรณีบอร์ดหรือ Flash รองรับความเร็วสูง

แนะนำ: ให้เลือก **40 MHz** เป็นค่าเริ่มต้น เพราะเหมาะกับบอร์ด ESP32 ทั่วไป และมีความเสถียรในการแฟลช

---

2. SPI MODE (โหมดการสื่อสารของแฟลช)

ตัวเลือกนี้กำหนดรูปแบบการเชื่อมต่อระหว่างชิป ESP32 กับ Flash memory

โหมด	**คำอธิบาย**	**ความเข้ากันได้**
QIO	Quad I/O (4 บิต)	ใช้กับ Flash ที่รองรับ QIO
DIO	Dual I/O (2 บิต)	**ใช้กันทั่วไป**, ปลอดภัย, เสถียร
DOUT	Dual Output	ใช้ในบางกรณี
QOUT	Quad Output	ใช้กับ Flash บางรุ่น
FASTRD	Fast Read	เร็ว แต่ไม่รองรับทุกบอร์ด

แนะนำ: เลือกเป็น **DIO** สำหรับการใช้งานทั่วไป เพราะเป็นโหมดที่ใช้ได้กับบอร์ดส่วนใหญ่ และให้ความเสถียรสูง

---

3. ตัวเลือกอื่น ๆ ที่สำคัญ

- **DoNotChgBin**

  หากติ๊กถูกไว้ (ค่าเริ่มต้น) โปรแกรมจะไม่เปลี่ยนแปลงไฟล์ .bin ที่เลือกไว้

แนะนำให้เปิดไว้เสมอ เพื่อป้องกันการเปลี่ยนแปลงโดยไม่ตั้งใจ

- **CombineBin**

  ใช้รวมไฟล์ .bin หลายไฟล์เป็นไฟล์เดียว (สำหรับ advanced user เท่านั้น)

- **Default**

  ปุ่มนี้จะรีเซ็ตการตั้งค่าทั้งหมดกลับเป็นค่าเริ่มต้นของโปรแกรม

สรุปการตั้งค่าที่แนะนำสำหรับผู้เริ่มต้น

ตัวเลือก	**ค่าที่แนะนำ**
SPI SPEED	40 MHz
SPI MODE	DIO
DoNotChgBin	✓ เปิดไว้
CombineBin	ไม่ต้องใช้
COM Port	ตรวจสอบจาก Device Manager
Baudrate	115200 หรือ 921600 /460800(flash)

การตั้งค่าทั้งหมดนี้สามารถใช้ร่วมกับเฟิร์มแวร์ MicroPython, Arduino, หรือ ESP-IDF ได้อย่างมีเสถียรภาพ โดยไม่ต้องแก้ไขเพิ่มเติมในกรณีทั่วไป หากใช้งานกับบอร์ดที่มีพฤติกรรมพิเศษ หรือแฟลชประเภทพิเศษ อาจต้องปรับค่าบางส่วนตามคู่มือของผู้ผลิตบอร์ดนั้น ๆ

ชาญพ ปักษี

การลบข้อมูลใน Flash ของ ESP32 ก่อนแฟลชเฟิร์มแวร์ MicroPython

ก่อนการติดตั้งเฟิร์มแวร์ MicroPython หรือเฟิร์มแวร์ใด ๆ ลงใน ESP32 แนะนำให้ลบข้อมูลทั้งหมดในหน่วยความจำ Flash ของบอร์ดก่อน เพื่อป้องกันปัญหาที่อาจเกิดจากข้อมูลเก่าหลงเหลืออยู่ เช่น โค้ดจากเฟิร์มแวร์เดิม หรือการจัดวางพาร์ติชันที่ไม่สอดคล้องกัน ซึ่งอาจทำให้เฟิร์มแวร์ใหม่ทำงานไม่สมบูรณ์

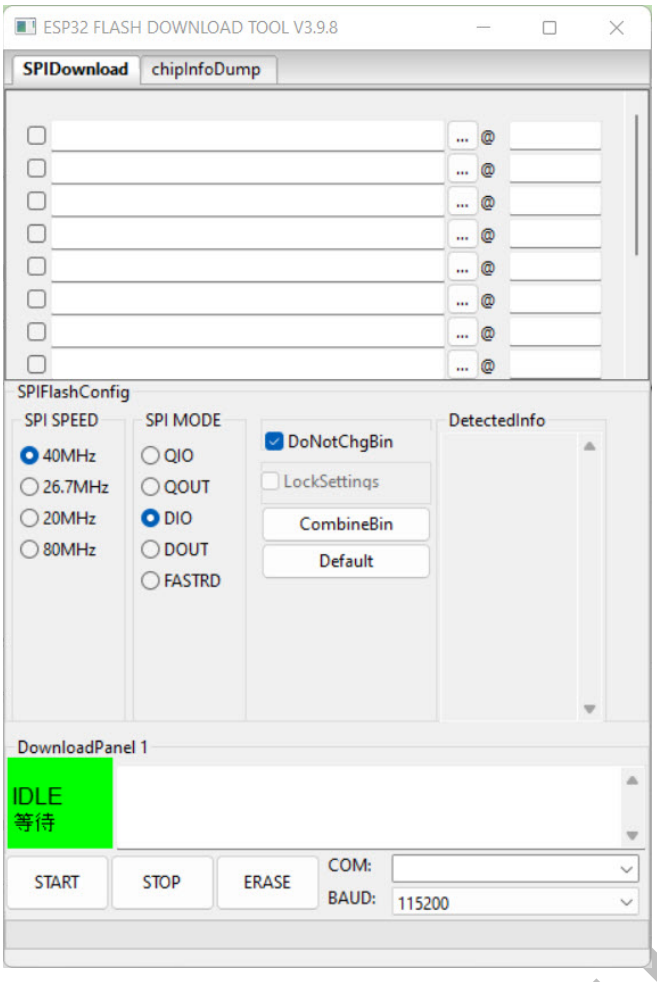
หากไม่ต้องการใช้คำสั่งผ่าน Terminal การลบ Flash สามารถทำได้ผ่านโปรแกรม **ESP32 Flash Download Tool** ซึ่งเป็นซอฟต์แวร์แบบกราฟิกที่พัฒนาโดย Espressif ผู้ผลิตชิป ESP โดยตรง

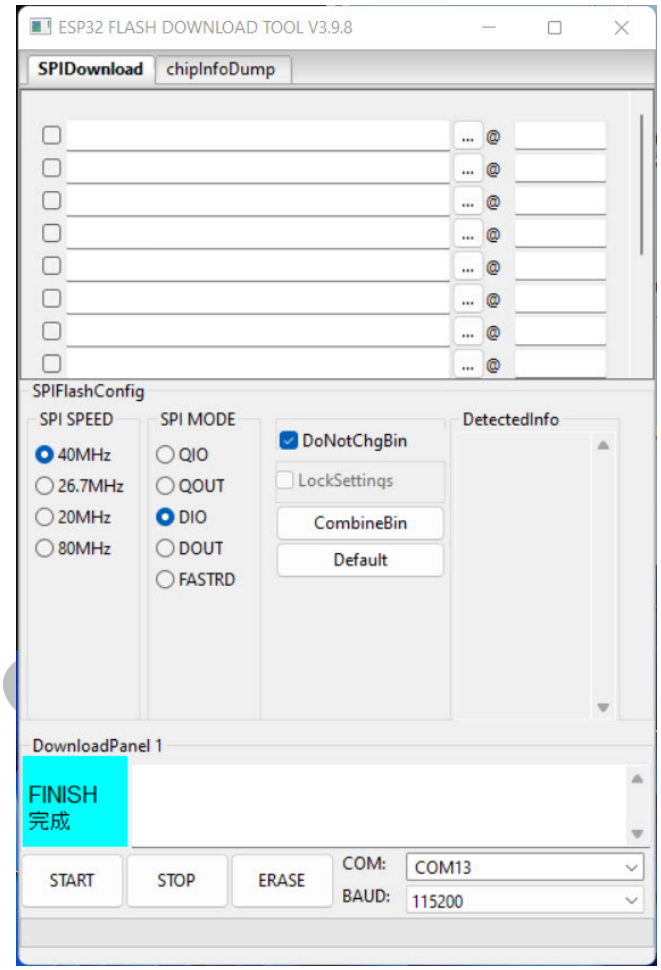
ขั้นตอนการลบ Flash ด้วย ESP32 Flash Download Tool

1. เปิดโปรแกรม **ESP32 Flash Download Tool** และเลือกแท็บ SPIDownload
2. ปล่อยช่องใส่ไฟล์ทั้งหมดว่างไว้ เนื่องจากต้องการลบ Flash เท่านั้น ไม่ได้แฟลชไฟล์ใด
3. กำหนดพอร์ตเชื่อมต่อของบอร์ดในช่อง COM: (เช่น COM3, COM4 เป็นต้น)
4. กำหนดความเร็ว Baudrate ที่ใช้ในการเชื่อมต่อ เช่น 115200
5. คลิกปุ่ม **ERASE** เพื่อเริ่มต้นลบข้อมูลใน Flash
6. รอจนกระทั่งสถานะด้านล่างแสดงข้อความว่า "FINISH" หรือ "ERASE SUCCESS"

เมื่อกระบวนการลบเสร็จสมบูรณ์ หน่วยความจำ Flash ของ ESP32 จะอยู่ในสถานะพร้อมสำหรับการติดตั้งเฟิร์มแวร์ใหม่ได้อย่างปลอดภัย

มาเพ ปัญญา





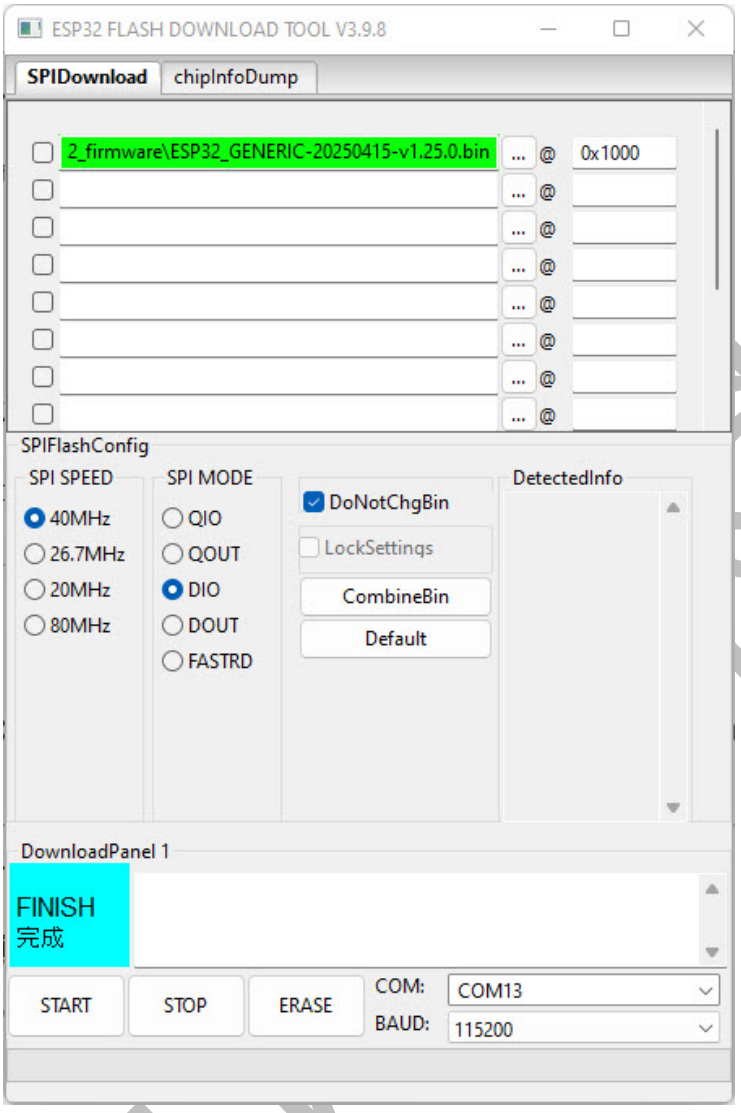
2.4.5 การเขียนเฟิร์มแวร์ (Writing the firmware)

การเตรียมแฟลชเฟิร์มแวร์ MicroPython

หลังจากลบ Flash แล้ว สามารถดำเนินการแฟลชเฟิร์มแวร์ MicroPython ได้โดยใช้เครื่องมือเดียวกัน โดยมีขั้นตอนเพิ่มเติมดังนี้:

1. ใช้เฟิร์มแวร์ .bin ของ MicroPython ที่ดาวน์โหลดมาจากในหัวข้อ 2.4.1

2. ในหน้าหลักของโปรแกรม ESP32 Flash Download Tool:
    - ในบรรทัดแรกของช่องใส่ไฟล์ คลิกปุ่ม ... เพื่อเลือกไฟล์ .bin ที่ดาวน์โหลดมา



o ระบุตำแหน่งการแฟลช (offset) เป็น 0x1000 ซึ่งเป็นตำแหน่งมาตรฐานของเฟิร์มแวร์ MicroPython

ทำไมต้องระบุตำแหน่ง 0x1000 เมื่อแฟลชเฟิร์มแวร์ MicroPython?

การระบุตำแหน่ง 0x1000 ในขั้นตอนการแฟลชเฟิร์มแวร์ MicroPython ลงบนบอร์ด ESP32 ไม่ใช่การกำหนดตำแหน่งแบบสุ่ม แต่เป็นการอ้างอิงถึง **โครงสร้างหน่วยความจำแฟลช (Flash memory map)** ที่ถูกกำหนดมาโดย MicroPython และ ESP-IDF (ESP32 SDK)

◆ ความเข้าใจเบื้องต้นเกี่ยวกับ Flash Memory ของ ESP32

ESP32 มีหน่วยความจำแฟลชภายนอก (external flash) ซึ่งแบ่งออกเป็นหลายส่วน และถูกโหลดเข้าตาม **offset address** ที่เฉพาะเจาะจง เช่น:

- 0x1000 ⟶ Bootloader
- 0x8000 ⟶ Partition Table

- 0x10000 ⟶ Application หรือ Firmware หลัก

แต่สำหรับ MicroPython นั้น ได้จัดเตรียม **ไฟล์ .bin เดียว** ซึ่งรวมทุกส่วนเอาไว้แล้ว ไม่จำเป็นต้องแยกแฟลชไฟล์ bootloader หรือ partition table เหมือนกับบางเฟิร์มแวร์ของ ESP-IDF

◆ **เหตุผลที่แฟลช MicroPython ต้องใช้ตำแหน่ง 0x1000**

แม้ว่าไฟล์ .bin ที่ดาวน์โหลดจาก MicroPython.org จะเป็นแบบรวมไฟล์ (monolithic binary) แต่ก็ออกแบบมาให้ถูกโหลดที่ตำแหน่ง 0x1000 โดยเฉพาะ ซึ่งเป็น **ตำแหน่งมาตรฐานของ Bootloader** ในระบบ ESP32

โดย Bootloader นี้มีหน้าที่เริ่มระบบและโหลดโค้ดหลักเข้า RAM เพื่อนำไปประมวลผล

หากแฟลชไปที่ตำแหน่งผิด เช่น 0x0000 หรือ 0x2000 อาจทำให้บอร์ดไม่สามารถบู๊ตเข้าสู่ MicroPython ได้เลย เพราะ Bootloader จะไม่อยู่ในตำแหน่งที่สามารถเรียกบู๊ตได้

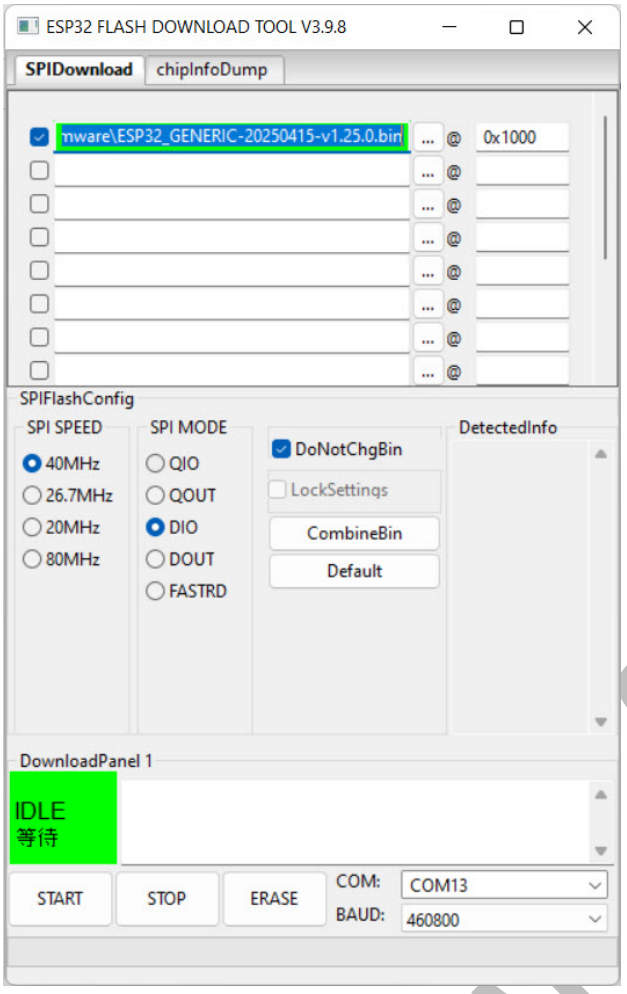
◆ **เปรียบเทียบให้เห็นภาพ**

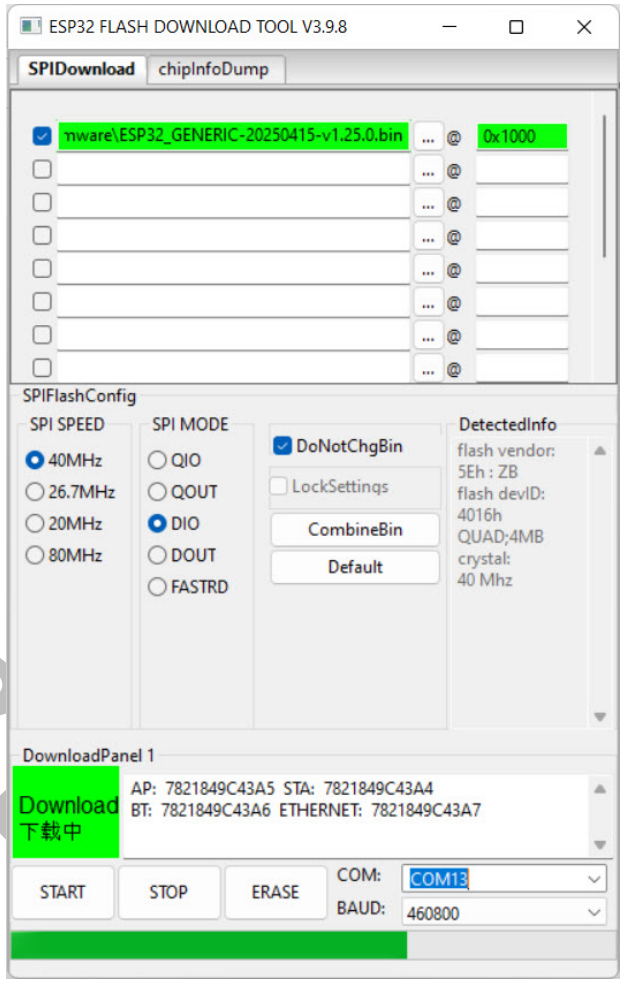
ตำแหน่งในแฟลช (Offset)	เนื้อหาที่ควรอยู่	หมายเหตุ
0x1000	Bootloader (MicroPython รวมไว้แล้ว)	ตำแหน่งที่ต้องระบุ
0x8000	Partition Table	รวมใน .bin แล้ว
0x10000	Firmware หลัก (MicroPython REPL)	รวมใน .bin แล้ว

การระบุตำแหน่งแฟลชที่ 0x1000 เป็นการอ้างอิงตามโครงสร้างที่ MicroPython กำหนดไว้ในเฟิร์มแวร์แบบ monolithic ซึ่งรวม Bootloader, Partition Table และ Firmware ไว้ในไฟล์เดียว และถูกออกแบบมาให้ทำงานถูกต้องเมื่อลงที่ตำแหน่ง 0x1000 หากระบุตำแหน่งไม่ถูกต้อง ESP32 จะไม่สามารถบูตเข้าสู่ MicroPython ได้ เพราะจะหา Bootloader ไม่เจอหรือโหลดผิดที่

3. **ตรวจสอบพอร์ต และ Baudrate ให้ถูกต้อง ในการแฟลชนั้นแนะนำให้ตั้งค่าที่ 460800**
4. คลิกปุ่ม **START** เพื่อเริ่มแฟลชเฟิร์มแวร์ลงบนบอร์ด จะมีแถบแสดงสีเขียวด้านล่าง

มูกเพ็ญ ปุกาย

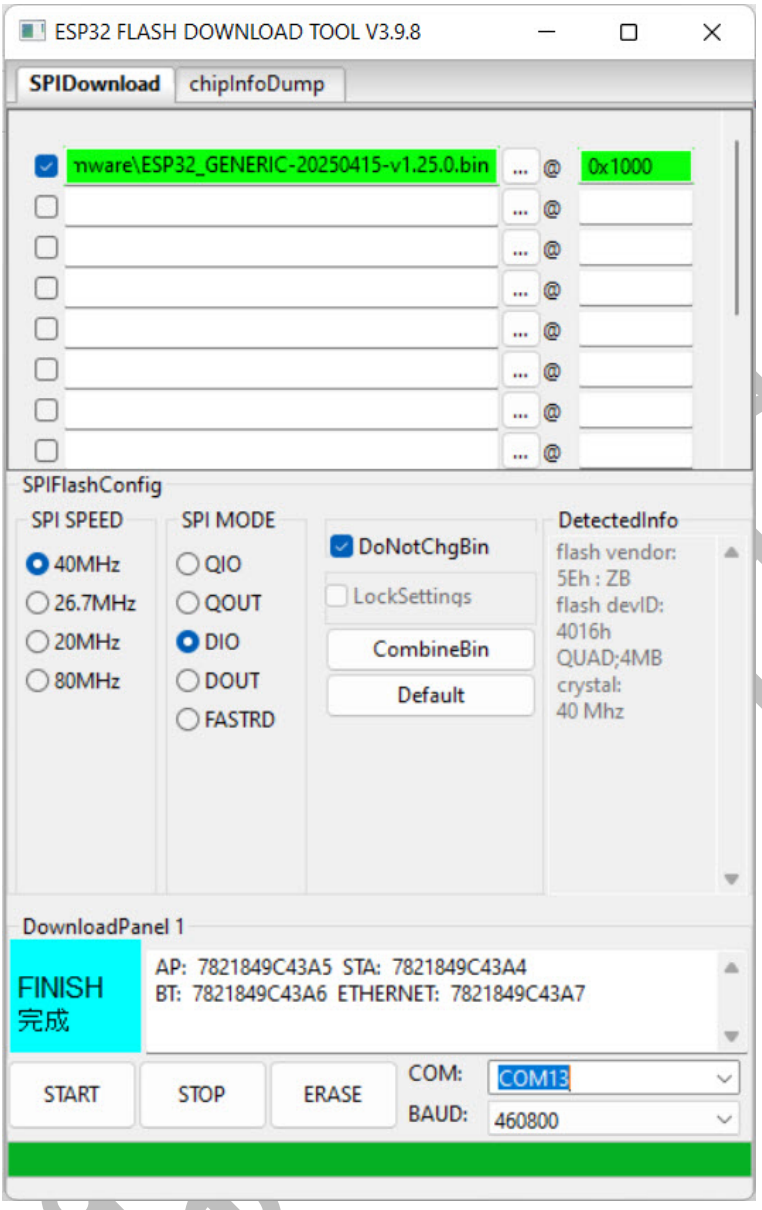




เมื่อแฟลชเสร็จสมบูรณ์ บอร์ด ESP32 จะพร้อมใช้งานกับ MicroPython ทันที โดยสามารถเชื่อมต่อผ่าน Serial Terminal หรือโปรแกรม Thonny IDE เพื่อเริ่มต้นการเขียนโปรแกรมต่อไป

การยืนยันการแฟลชเฟิร์มแวร์ และการแสดงรหัสประจำตัวของบอร์ด ESP32

หลังจากดำเนินการแฟลชเฟิร์มแวร์ MicroPython ลงบนบอร์ด ESP32 โดยใช้โปรแกรม ESP32 Flash Download Tool แล้ว ผู้ใช้จะได้รับข้อความแจ้งเตือน และโปรแกรมจะดึงข้อมูลรหัสประจำตัวของบอร์ดมาแสดงในช่องด้านล่างของหน้าต่าง เมื่อการแฟลชสำเร็จ โปรแกรมจะแสดงข้อความ **"FINISH 完成"** ที่มุมล่างซ้าย ซึ่งเป็นภาษาจีน แปลว่า "เสร็จสมบูรณ์" แสดงว่าเฟิร์มแวร์ถูกเขียนลงในหน่วยความจำของ **ESP32** เรียบร้อยแล้วโดยไม่มีข้อผิดพลาด ขั้นตอนต่อไปคือการรีสตาร์ทบอร์ดด้วยการกดปุ่ม **EN** (หรือ **RESET**) บนบอร์ด หรือถอดสาย USB แล้วเสียบใหม่ เพื่อให้บอร์ดเริ่มต้นทำงานด้วยเฟิร์มแวร์ใหม่นี้



การแสดง MAC Address ของ ESP32

หลังจากแฟลชเสร็จ โปรแกรมจะอ่านข้อมูล MAC Address ของระบบสื่อสารต่าง ๆ ที่อยู่ภายใน ESP32 และแสดงออกมาดังตัวอย่าง แต่ละบอร์ดจะมีค่าที่แตกต่าง และไม่ซ้ำกัน

AP: 7821849C43A5 STA: 7821849C43A4

BT: 7821849C43A6 ETHERNET: 7821849C43A7

MAC Address แต่ละค่าคือหมายเลขประจำตัวเฉพาะของอุปกรณ์เครือข่ายภายในบอร์ด เลขเหล่านี้เปรียบได้กับ "เลขบัตรประชาชนของ ESP32" ที่ใช้ระบุตัวตนในระบบเครือข่าย และที่สำคัญคือ **ไม่สามารถเปลี่ยนแปลงได้**

ความหมายของแต่ละค่า

รหัส	หน่วยงานที่เกี่ยวข้อง	การใช้งาน
AP	Wi-Fi Access Point	ใช้เมื่อบอร์ดเป็นตัวปล่อยสัญญาณ Wi-Fi
STA	Wi-Fi Station	ใช้เมื่อบอร์ดเชื่อมต่อ Wi-Fi ภายนอก
BT	Bluetooth	ใช้ในการเชื่อมต่อ Bluetooth กับอุปกรณ์อื่น
ETHERNET	Ethernet	ใช้เมื่อเชื่อมต่อกับโมดูล LAN PHY ภายนอก

ความสำคัญของ MAC Address

MAC Address ใน ESP32 ถูกฝังถาวรไว้ในหน่วยความจำชนิดพิเศษชื่อ eFuse ซึ่งเป็นหน่วยความจำแบบเขียนได้ครั้งเดียว ดังนั้นค่าที่แสดงออกมาเหล่านี้จะไม่สามารถเปลี่ยนแปลงได้ และแตกต่างกันในแต่ละบอร์ดอย่างแน่นอน 100% แม้ว่าโปรแกรมเมอร์จะสามารถตั้งค่า MAC Address ปลอมได้ในซอฟต์แวร์ (spoofing) แต่เมื่อบอร์ดถูกรีเซตหรือเปิดใหม่ MAC Address จะกลับคืนเป็นค่าต้นฉบับที่ถูกฝังไว้ในโรงงานเสมอ

- การแสดงข้อความ **FINISH 完成** หมายถึงการแฟลชเฟิร์มแวร์สำเร็จแล้ว
- ข้อมูล AP, STA, BT, ETHERNET คือ MAC Address ถาวรที่แตกต่างกันในแต่ละบอร์ด
- MAC Address จะไม่มีทางซ้ำกัน และไม่สามารถเปลี่ยนได้
- ใช้ในการระบุตัวตนของบอร์ดในระบบ Wi-Fi, Bluetooth หรือ Ethernet ได้อย่างแม่นยำ

หากนำ ESP32 หลายตัวมาใช้งานร่วมกันในระบบเดียว เช่น ระบบบ้านอัจฉริยะ MAC Address เหล่านี้จะช่วยให้แยกบอร์ดแต่ละตัวได้อย่างชัดเจนแม้มีหน้าตาเหมือนกันทุกประการ

2.5 Thonny IDE: เพื่อนคู่ใจนักพัฒนา MicroPython

การเริ่มต้นเรียนรู้ MicroPython จะง่ายขึ้นมาก หากมีเครื่องมือที่ออกแบบมาอย่างเข้าใจผู้ใช้งาน โดยเฉพาะมือใหม่ที่อาจยังไม่ถนัดการเขียนโค้ดหรือใช้งานเครื่องมือซับซ้อน Thonny IDE จึงเป็นทางเลือกยอดนิยมที่นักพัฒนาทั่วโลกแนะนำ เพราะมีจุดเด่นคือความเรียบง่าย ใช้งานง่าย แต่ยังคงทรงพลังและรองรับการทำงานกับบอร์ดไมโครคอนโทรลเลอร์ยอดนิยมอย่างครบถ้วน

ตัวอย่าง

Thonny เป็นโปรแกรมประเภท Integrated Development Environment (IDE) ที่พัฒนาขึ้นโดยมหาวิทยาลัย Tartu จากประเทศเอสโตเนีย โดยเน้นการใช้งานสำหรับภาษา Python เป็นหลัก แต่สิ่งที่ทำให้ Thonny แตกต่างจาก IDE ทั่วไปคือความสามารถในการรองรับ MicroPython ได้โดยตรง ซึ่งทำให้ผู้ใช้งานสามารถเชื่อมต่อกับบอร์ดเช่น ESP32, ESP8266 หรือแม้แต่ Raspberry Pi Pico ได้ทันทีโดยไม่ต้องติดตั้งปลั๊กอินเพิ่มเติม

จุดเด่นข้อแรกของ Thonny IDE คือสามารถเชื่อมต่อกับบอร์ด ESP32 หรือ ESP8266 ได้ทันที หลังจากที่ผู้ใช้แฟลชเฟิร์มแวร์ MicroPython ลงในบอร์ดเรียบร้อยแล้ว เพียงเสียบสาย USB เข้ากับคอมพิวเตอร์ Thonny จะตรวจจับพอร์ตอนุกรม (Serial) ได้อัตโนมัติ ช่วยลดขั้นตอนที่ยุ่งยากและเหมาะอย่างยิ่งสำหรับผู้เริ่มต้น
อีกฟีเจอร์หนึ่งที่โดดเด่นคือการมีหน้าต่าง REPL Console ในตัว ผู้ใช้สามารถพิมพ์คำสั่งแบบสด (Interactive) เพื่อทดสอบการทำงานของบอร์ดได้ทันที เช่น การสั่งพิมพ์ข้อความ การอ่านค่าพิน หรือการทดลองฟังก์ชัน Python ต่าง ๆ โดยไม่ต้องเขียนโค้ดยาวแล้วกดรันเสมอไป ซึ่งช่วยให้การเรียนรู้และแก้ไขโค้ดเป็นไปได้อย่างรวดเร็วและแม่นยำ

Thonny ยังมีระบบจัดการไฟล์ที่รองรับการอัปโหลดไฟล์ .py ไปยังบอร์ด MicroPython ได้โดยตรงผ่านหน้าต่างฝั่งซ้ายของโปรแกรม ซึ่งเปรียบเสมือนตัวจัดการไฟล์ในระบบปฏิบัติการ ทำให้สามารถลากวางไฟล์ หรือบันทึกโค้ดลงในบอร์ดได้ทันทีโดยไม่ต้องพึ่งพาเครื่องมือเสริม เช่น ampy หรือ mpremote

อินเทอร์เฟซของ Thonny ได้รับการออกแบบมาให้เหมาะสำหรับผู้เริ่มต้นอย่างแท้จริง โดยแสดงเฉพาะเมนูและปุ่มที่จำเป็น เช่น Run, Save, Stop, และ Debug ไม่รกสายตาเหมือน IDE ระดับสูงทั่วไป ทำให้ผู้ใช้ไม่รู้สึกสับสนและสามารถจดจ่อกับการเขียนโปรแกรมได้เต็มที่

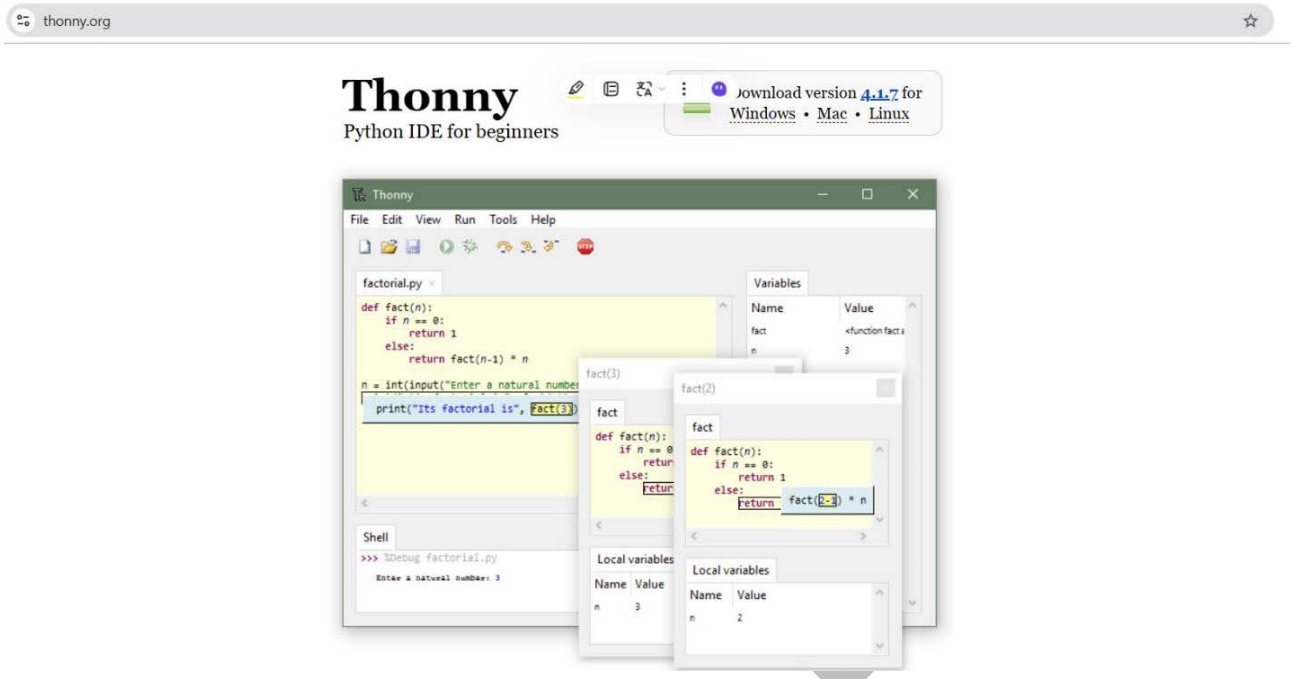
อีกหนึ่งคุณสมบัติที่น่าประทับใจคือความสามารถในการแสดงค่าตัวแปรแบบภาพ (Visual Variable Inspector) ซึ่งช่วยให้ผู้เรียนสามารถติดตามการเปลี่ยนแปลงค่าของตัวแปรในแต่ละช่วงของโปรแกรมได้อย่างเป็นระบบ เหมาะสำหรับการสอนและการเรียนรู้แนวคิดของโค้ดเชิงลึก โดยเฉพาะผู้ที่ยังไม่เข้าใจการ Debug แบบข้อความ

Thonny IDE เหมาะสำหรับนักเรียน นักศึกษา ผู้สอน และผู้เริ่มต้นใช้งาน MicroPython ที่ต้องการเครื่องมือที่ใช้งานง่าย ติดตั้งไม่ยุ่งยาก และสามารถเริ่มต้นได้ทันที เหมาะสำหรับใช้งานทั้งในห้องเรียน เวิร์กช็อป หรือการเรียนรู้ด้วยตนเองที่บ้าน โปรแกรมนี้สามารถใช้งานได้ฟรี และรองรับทั้ง Windows, macOS และ Linux
ด้วยคุณสมบัติทั้งหมดนี้ Thonny IDE จึงถือเป็นเครื่องมือหลักที่ควรติดตั้งเป็นอันดับแรก สำหรับผู้ที่ต้องการเข้าสู่โลกของ MicroPython อย่างมั่นใจและมีประสิทธิภาพ

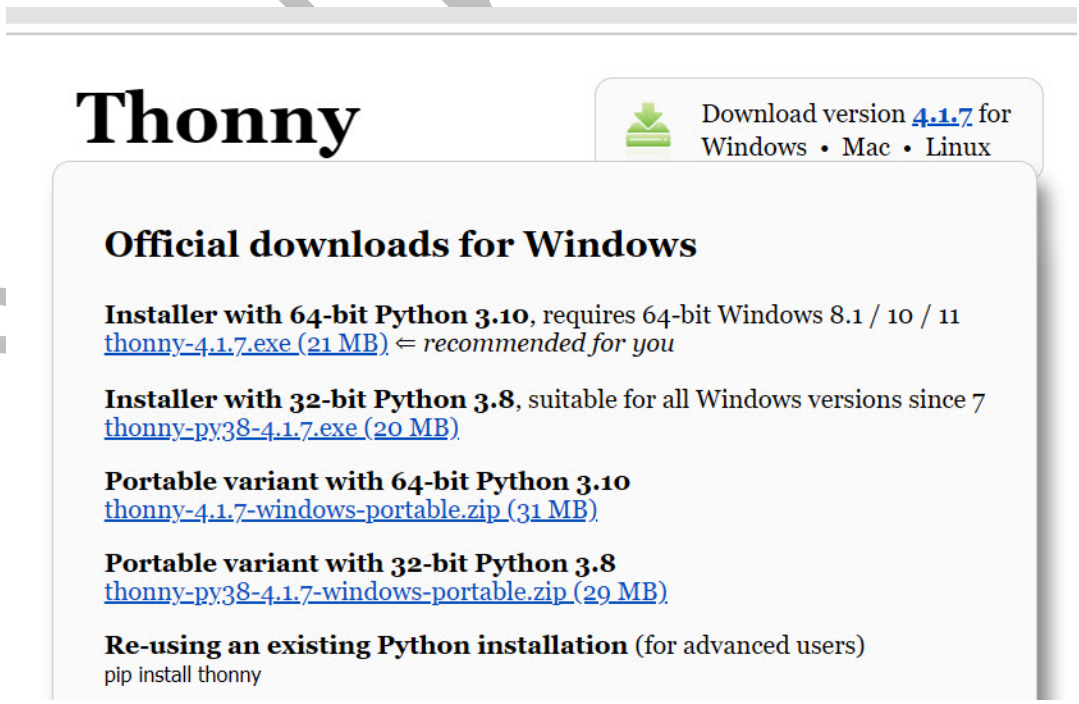
มาใหม่ ปักใจ

2.5.1 การติดตั้ง Thonny IDE

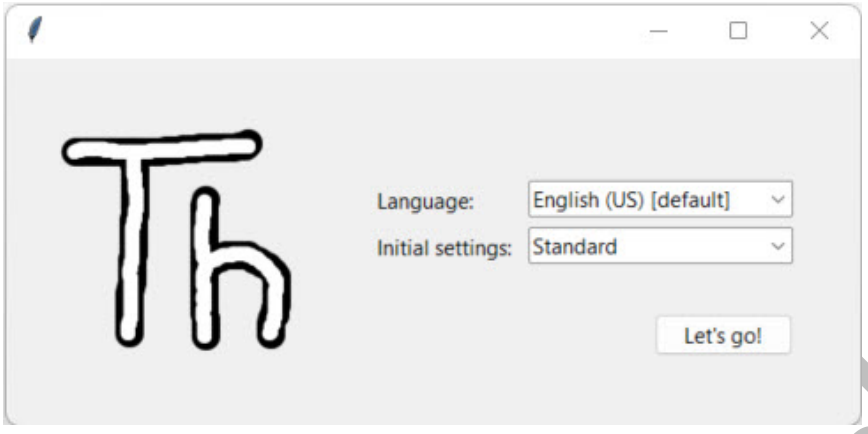
1. ไปที่เว็บไซต์ทางการของ Thonny: https://thonny.org/



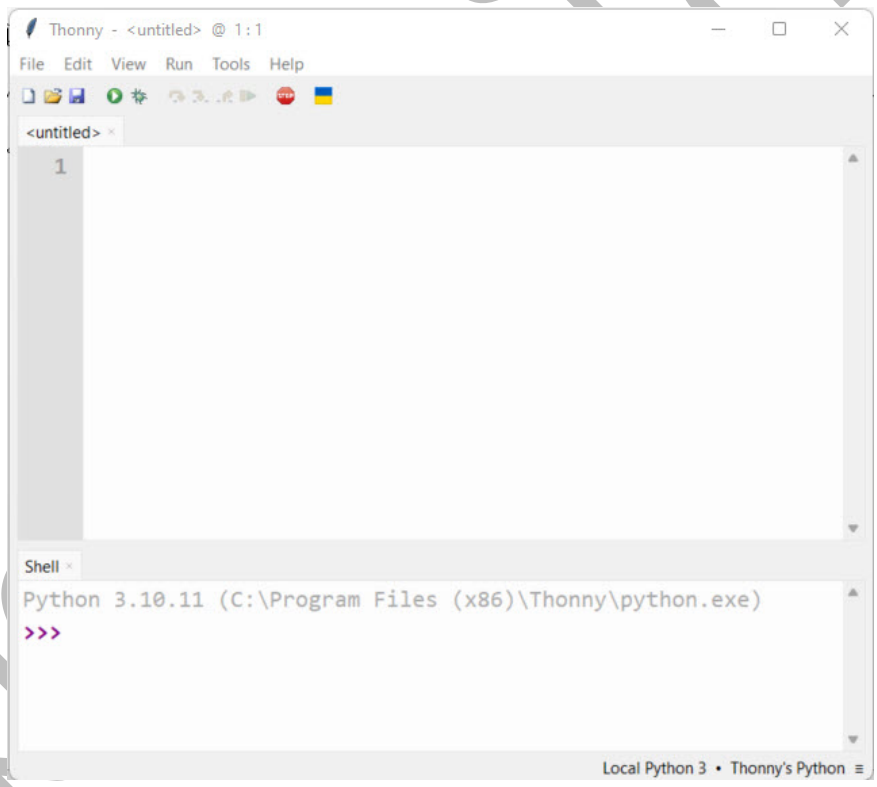
2. ดาวน์โหลดโปรแกรมติดตั้ง (Installer) ที่ตรงกับระบบปฏิบัติการของคอมพิวเตอร์ (Windows, macOS, Linux) แต่สำหรับผู้เขียนขอเลือกเวอร์ชันที่ทำงานบนระบบปฏิบัติการ wondows 8.1/10/11 ให้ผู้อ่านเลือกตามระบบปฏิบัติการที่ท่านใช้งานจริง



3. ทำการติดตั้งตามขั้นตอนปกติของแต่ละระบบปฏิบัติการ เช่นเดียวกับโปรแกรมทั่วไป



เมื่อติดตั้งเสร็จการเปิดใช้งานครั้งแรกโปรแกรมจะถามให้เลือกภาษา แนะนำให้เลือกภาษาอังกฤษ เพื่อความเป็นสากลในการเขียนโปรแกรม และการแนะนำการตั้งค่าต่างๆในโปรแกรมจะได้เป็นไปในทิศทางเดียวกัน หรือใครอยากลองภาษาไทยก็ลองดูครับ



2.5.2 การตั้งค่า Thonny สำหรับ ESP32

1. เปิดโปรแกรม **Thonny IDE**
2. ไปที่เมนู **Tools > Options...**

   (หรือ **Thonny > Preferences...** สำหรับผู้ใช้ macOS)
3. เลือกแท็บ **Interpreter**

4. ในช่อง

   “Which interpreter or device should Thonny use for running your code?”

   ให้เลือกเป็น:

   ⟶ **MicroPython (ESP32)**

5. ในช่อง *“Port or WebREPL”* ให้เลือกพอร์ตที่บอร์ด ESP32 เชื่อมต่ออยู่

   เช่น COM13, /dev/cu.usbserial-xxxx, หรือ /dev/ttyUSB0

   - หากไม่พบพอร์ตที่ถูกต้อง ให้คลิกปุ่ม **Rescan**

     หรือ **ตรวจสอบการเชื่อมต่อบอร์ด** และ **การติดตั้งไดรเวอร์** ว่าถูกต้องหรือไม่
   - หากเพิ่งติดตั้งเฟิร์มแวร์ MicroPython และรีเซ็ตบอร์ดแล้ว

     ESP32 ควรจะพร้อมให้ Thonny เชื่อมต่อทันที

6. คลิกปุ่ม **OK** เพื่อบันทึกการตั้งค่า

7. ที่ด้านล่างของหน้าต่าง Thonny จะมีส่วนที่เรียกว่า **Shell** หรือ **REPL**

   หากการเชื่อมต่อสำเร็จ จะเห็นข้อความต้อนรับจาก MicroPython ปรากฏขึ้น พร้อมเครื่องหมาย >>> เช่น:

```
MicroPython v1.25.0 on 2025-04-15; Generic ESP32 module with
ESP32
Type "help()" for more information.
>>>
```

8. หากไม่มีเครื่องหมาย >>> ปรากฏขึ้น ให้ลองกดปุ่ม

   STOP / Restart backend (ปุ่มสีแดงมีไอคอน Stop)

   หรือกลับไปตรวจสอบการตั้งค่าพอร์ตอีกครั้ง

✅ คำแนะนำเพิ่มเติม

💡 **หมายเหตุ:** การเลือก Interpreter และพอร์ตที่ถูกต้องเป็นสิ่งสำคัญ หากเลือกผิด อาจไม่สามารถเชื่อมต่อกับบอร์ด ESP32 ได้ และจะไม่เห็นข้อความ REPL ที่พร้อมใช้งาน

ตัวอย่าง

2.6 โปรแกรมแรก: "สวัสดี ESP32!" และการควบคุม LED ออนบอร์ด

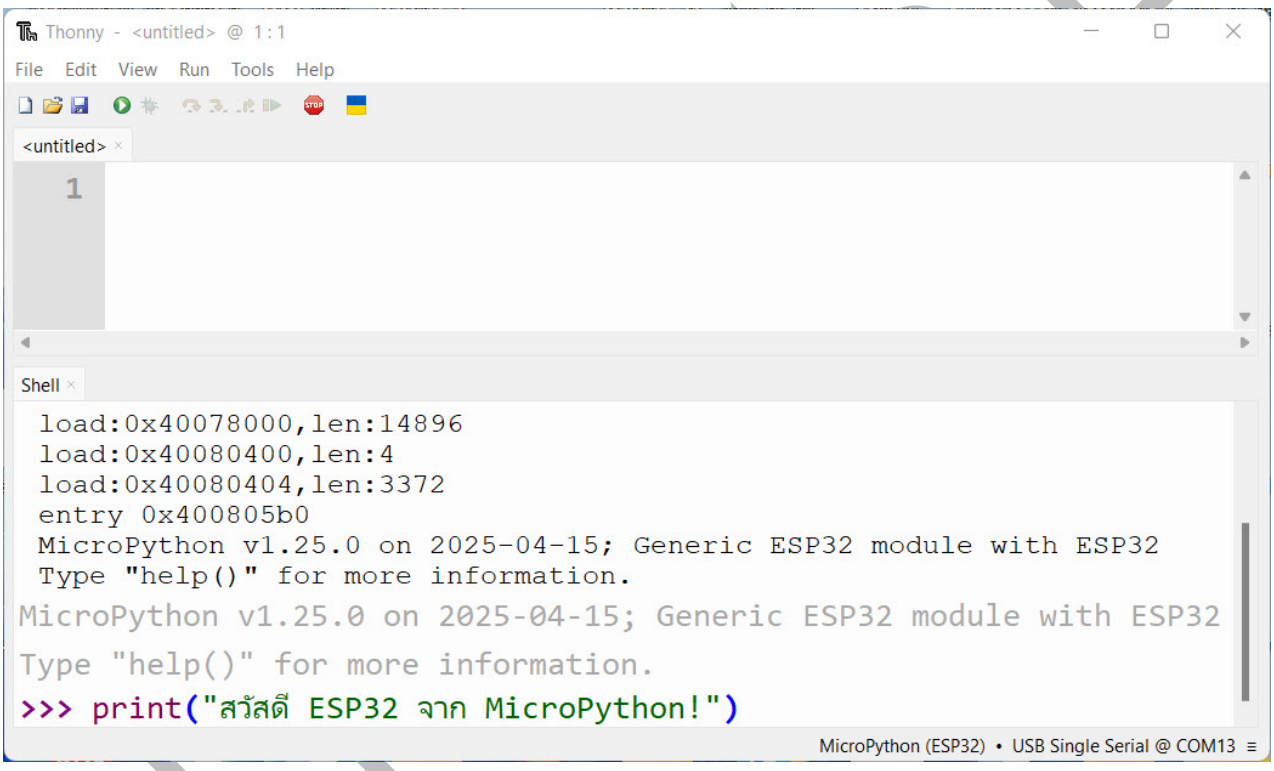
เมื่อ Thonny IDE เชื่อมต่อกับ ESP32 ได้แล้ว ก็ถึงเวลาทดลองเขียนโปรแกรมแรก

2.6.1 การใช้งาน REPL เบื้องต้น

REPL (Read-Evaluate-Print Loop) ช่วยให้สามารถส่งคำสั่ง Python ไปยัง ESP32 และรับผลลัพธ์กลับมาได้ทันที

1. ที่หน้าต่าง Shell ของ Thonny (ตรงที่มี >>>) ลองพิมพ์คำสั่ง Python ง่ายๆ

   print("สวัสดี ESP32 จาก MicroPython!") ตามรูปภาพข้างล่าง แล้วกดปุ่ม Enter



จะแสดงผลตอบกลับมาทันที หรืออาจลองพิมพ์บวกเลขง่าย ๆ เช่น 250+250 จะแสดงผลลัพธ์ออกมาดังภาพ

2. **การควบคุม LED ออนบอร์ดผ่าน REPL:** บอร์ด ESP32 ส่วนใหญ่จะมี LED ติดตั้งอยู่บนบอร์ด (Onboard LED) ซึ่งมักจะเชื่อมต่อกับขา GPIO หมายเลข 2 (GPIO2) แต่บางบอร์ดอาจเป็นขาอื่น ให้ตรวจสอบจาก Pinout Diagram ของบอร์ดที่ใช้

ลองพิมพ์คำสั่งต่อไปนี้ทีละบรรทัดใน REPL เพื่อทำให้ LED ติดและดับ (สมมติว่า LED ต่อกับ GPIO2):

>>> import machine # นำเข้าโมดูล machine สำหรับควบคุมฮาร์ดแวร์

>>> led = machine.Pin(2, machine.Pin.OUT) # กำหนดขา GPIO2 เป็น Output

>>> led.on() # สั่งให้ LED ติด (จ่าย Logic Low, บางบอร์ด LED ติดเมื่อ Logic High)

>>> led.value(1) # หรือใช้ value(1) สำหรับติด (ถ้า on() ไม่ติด ให้ลอง value(0))

สังเกตว่า LED บนบอร์ดควรติดขึ้นมา

>>> led.off() # สั่งให้ LED ดับ (จ่าย Logic High, บางบอร์ด LED ดับเมื่อ Logic Low)

>>> led.value(0) # หรือใช้ value(0) สำหรับดับ (ถ้า off() ไม่ดับ ให้ลอง value(1))

LED บนบอร์ดควรจะดับลง

หมายเหตุ: ลักษณะการทำงานของ led.on(), led.off(), led.value(1), led.value(0) อาจแตกต่างกันไปในแต่ละบอร์ด บางบอร์ด LED จะติดเมื่อได้รับสัญญาณ LOW (Active Low) บางบอร์ดจะติดเมื่อได้รับ HIGH (Active High) หากคำสั่งหนึ่งไม่ทำงานตามคาด ให้ลองใช้คำสั่งตรงข้ามดู

2.6.2 การเขียนและบันทึกไฟล์สคริปต์

หมายเหตุสำคัญ: ควรทำความเข้าใจก่อนว่า คำสั่งใน REPL เป็นเพียงการควบคุมชั่วคราวเท่านั้น เหมาะสำหรับการทดลองสั้นๆ แต่ถ้าต้องการให้โปรแกรมทำงานซ้ำๆ หรือทำงานอัตโนมัติเมื่อเปิดเครื่อง ควรบันทึกโค้ดเป็นไฟล์

ตัวอย่างเช่น เมื่อเราพิมพ์คำสั่งในหน้าต่าง REPL ของ Thonny เหมือนที่เราทำกันไปเมื่อครู่

```
import machine
led = machine.Pin(2, machine.Pin.OUT)
led.on()
```

ตัวอย่าง

บอร์ดจะตอบสนองทันที เช่น ไฟ LED ติด แต่การทำงานเหล่านี้เป็นเพียงการทำงาน ชั่วคราว และเกิดขึ้นใน หน่วยความจำแรม (RAM) เท่านั้น เมื่อรีสตาร์ทบอร์ด, ปิดไฟ, หรือถอดสาย USB – คำสั่งที่พิมพ์ไว้จะหายไปทันที

ถ้าอยากให้บอร์ดจำคำสั่งและทำงานอัตโนมัติทุกครั้งที่เปิดเครื่อง

ให้เขียนคำสั่งเหล่านั้นลงในไฟล์ชื่อว่า main.py

แล้วอัปโหลดไฟล์นี้ไปยังบอร์ด ESP32 ผ่านโปรแกรม Thonny หรือเครื่องมืออื่น ๆ เช่น mpremote

1. ใน Thonny IDE คลิกที่ File > New เพื่อเปิดหน้าต่าง Editor ใหม่สำหรับเขียนโค้ด
2. พิมพ์โค้ดต่อไปนี้ใน Editor (โค้ดไฟกะพริบ):

main.py *

```python
import machine
import time

# กำหนดขา LED (ตรวจสอบหมายเลขขาของบอร์ดที่ใช้)
led_pin = 2
led = machine.Pin(led_pin, machine.Pin.OUT)

print("โปรแกรมไฟกะพริบเริ่มต้นแล้ว!")

while True:
    led.on()  # LED ติด
    time.sleep_ms(500)  # หน่วงเวลา 500 มิลลิวินาที (0.5 วินาที)
    led.off()  # LED ดับ
    time.sleep_ms(500)  # หน่วงเวลา 500 มิลลิวินาที
```

3. คลิกที่ File > Save as...
4. Thonny จะถามว่าจะบันทึกไฟล์ไว้ที่ไหน ให้เลือก MicroPython device
5. ตั้งชื่อไฟล์เป็น main.py แล้วคลิก OK
    - **ความสำคัญของ boot.py และ main.py:**
        - boot.py: เป็นไฟล์สคริปต์ที่จะทำงานเป็นไฟล์แรกสุดเมื่อ ESP32 เปิดเครื่อง (หลังจากรีเซ็ต) มักใช้สำหรับตั้งค่าระบบเบื้องต้น (เช่น การเชื่อมต่อ Wi-Fi)

- main.py: เป็นไฟล์สคริปต์หลักของโปรแกรม จะทำงานหลังจาก boot.py ทำงานเสร็จ (หากมี boot.py)

6. หลังจากบันทึกไฟล์ main.py ลงใน ESP32 แล้ว ให้ลองกดปุ่ม STOP/Restart backend ใน Thonny หรือกดปุ่ม EN/RESET บนบอร์ด ESP32 โปรแกรมไฟกะพริบควรจะเริ่มทำงานโดยอัตโนมัติ และ LED ออนบอร์ดจะกะพริบ สามารถดูข้อความ print ได้ในหน้าต่าง Shell ของ Thonny
7. หากต้องการหยุดโปรแกรมที่กำลังรันอยู่ใน Loop (เช่น while True) ให้กด Ctrl+C ในหน้าต่าง Shell ของ Thonny

2.7 REPL: เครื่องมือทดลองโค้ดอันทรงพลัง

ดังที่ได้ทดลองไปแล้ว REPL (Read-Evaluate-Print Loop) เป็นเครื่องมือที่มีประโยชน์อย่างยิ่งในการพัฒนา MicroPython:

- **ทดสอบแนวคิดอย่างรวดเร็ว:** สามารถทดลองคำสั่งหรือฟังก์ชันเล็กๆ ได้ทันทีโดยไม่ต้องเขียนเป็นไฟล์เต็ม
- **ตรวจสอบสถานะฮาร์ดแวร์:** ใช้ REPL เพื่ออ่านค่าจากเซ็นเซอร์ หรือตรวจสอบสถานะของขา GPIO ได้โดยตรง
- **ดีบักโปรแกรม:** เมื่อโปรแกรมที่รันจากไฟล์เกิดปัญหา สามารถใช้ REPL เพื่อตรวจสอบค่าของตัวแปร หรือทดลองเรียกใช้ฟังก์ชันต่างๆ เพื่อหาสาเหตุ
- **เรียนรู้โมดูลและคำสั่งใหม่ๆ:** สามารถ import โมดูลแล้วใช้คำสั่ง help(module_name) หรือ dir(module_name) เพื่อดูว่ามีฟังก์ชันอะไรให้ใช้งานบ้าง

การฝึกใช้งาน REPL ให้คล่องแคล่วจะช่วยให้กระบวนการเรียนรู้และพัฒนาโครงงาน MicroPython เป็นไปอย่างราบรื่นและมีประสิทธิภาพมากขึ้น

2.8 บทสรุปและสิ่งที่รออยู่ในบทถัดไป

ในบทนี้ ผู้อ่านได้เรียนรู้ขั้นตอนที่สำคัญทั้งหมดในการเตรียมความพร้อมสำหรับการพัฒนา MicroPython บน ESP32 ตั้งแต่การเลือกบอร์ด, การติดตั้งไดรเวอร์, การติดตั้งเฟิร์มแวร์ MicroPython ด้วย esptool.py, ไปจนถึงการตั้งค่า Thonny IDE และการทดลองเขียนโปรแกรมแรกเพื่อควบคุม LED ออนบอร์ด ทั้งผ่าน REPL และการบันทึกเป็นไฟล์ main.py

การที่สามารถทำให้ LED ดวงเล็กๆ บนบอร์ดกะพริบได้ด้วยโค้ด Python ที่เขียนขึ้นเองนั้น ถือเป็นก้าวแรกที่ยิ่งใหญ่ และเป็นพื้นฐานสำคัญสู่การสร้างสรรค์โครงงานที่ซับซ้อนยิ่งขึ้น

สุทธิพร ปัญญี

💡 **ทำไมต้องใช้ชื่อ main.py กับ MicroPython?**

เมื่อใช้งาน MicroPython บนบอร์ด ESP32 หรือ ESP8266 เรามักจะต้องการให้บอร์ด "เริ่มทำงานอัตโนมัติ" ทันทีหลังจากได้รับพลังงาน ไม่ว่าจะเป็นการจ่ายไฟผ่านพอร์ต USB, การเสียบเข้ากับแบตเตอรี่, หรือการใช้แหล่งจ่ายไฟภายนอกอื่น ๆ

✅ กล่าวคือ **หลังจากจ่ายไฟเลี้ยงให้กับบอร์ด** MicroPython จะเริ่มกระบวนการทำงานตามลำดับดังนี้:

1. **รัน boot.py (ถ้ามี):** มักใช้สำหรับตั้งค่าระบบ เช่น Wi-Fi, UART
2. **รัน main.py (ถ้ามี):** เป็นจุดเริ่มต้นของโปรแกรมที่เขียนโดยผู้ใช้งาน

ดังนั้น หากต้องการให้บอร์ด **เริ่มทำงานอัตโนมัติทันทีหลังจากได้รับไฟเลี้ยง** ไม่ว่าจะต่อผ่าน USB หรือจ่ายไฟจากภายนอก — จะต้องมีไฟล์ main.py อยู่ในระบบไฟล์ของบอร์ด

🔸 **คำว่า main.py จึงเปรียบเสมือนกับ int main() ในภาษา C:**

เป็นตำแหน่งที่ระบบจะเริ่ม "รัน" โปรแกรมของเราโดยอัตโนมัติ และเป็นมาตรฐานของ MicroPython ที่ฝังอยู่ในเฟิร์มแวร์อยู่แล้ว หากไม่มีไฟล์ main.py อยู่เลย บอร์ดจะเข้าสู่โหมด REPL (พร้อมพิมพ์คำสั่งแบบสด) แต่จะไม่ทำงานใด ๆ อัตโนมัติ — ซึ่งอาจไม่เหมาะกับโปรเจกต์ที่ต้องติดตั้งจริง เช่น วัดอุณหภูมิ, สั่งเปิดปั๊มน้ำ, หรือควบคุมอุปกรณ์ภาคสนาม

⚠️ **แล้วถ้ามี "หลายโปรเจกต์"... จะเกิดอะไรขึ้น?**

ลองจินตนาการว่า:

- เรามีโปรเจกต์ควบคุมไฟ LED อัตโนมัติ
- มีอีกโปรเจกต์วัดอุณหภูมิ DHT22
- และโปรเจกต์แสดงผลบน OLED

แน่นอนว่า **แต่ละโปรเจกต์ก็ต้องมี main.py ของตัวเอง** แล้วถ้าเราเก็บโปรเจกต์ทั้งหมดนี้ไว้ในคอมพิวเตอร์และทุกโปรเจกต์ดันใช้ชื่อไฟล์ว่า main.py เหมือนกันหมด... สิ่งที่จะเกิดขึ้น คือ: เราจะไม่รู้ว่าไฟล์ main.py ไฟล์ไหนเป็นของโปรเจกต์ไหน และโดยปกติทั่วไปแล้วคอมพิวเตอร์จะไม่อนุญาตให้เราตั้งชื่อไฟล์ซ้ำกันไว้ในที่เดียวกันอยู่แล้ว ดังนั้นการบันทึกไฟล์ควรสร้างโฟล์เดอร์ขึ้นมาแยกเก็บแต่ละโปนเจกต์ให้ชัดเจน เพราะทุกครั้งที่อัปโหลด main.py ลง

สมาพ ยุ่ภาใช้

บอร์ด ถ้าเราเผลอเลือกผิดไฟล์ แฟลชผิดโปรเจกต์ หรือเขียนทับโดยไม่ได้ตั้งใจ — โค้ดเดิมจะหายไปทันที และเมื่อรีสตาร์ทบอร์ดอีกครั้ง มันก็จะรันโค้ดผิดโปรเจกต์แบบไม่รู้ตัว

🧠 วิธีจัดการหลายโปรเจกต์อย่างมืออาชีพ

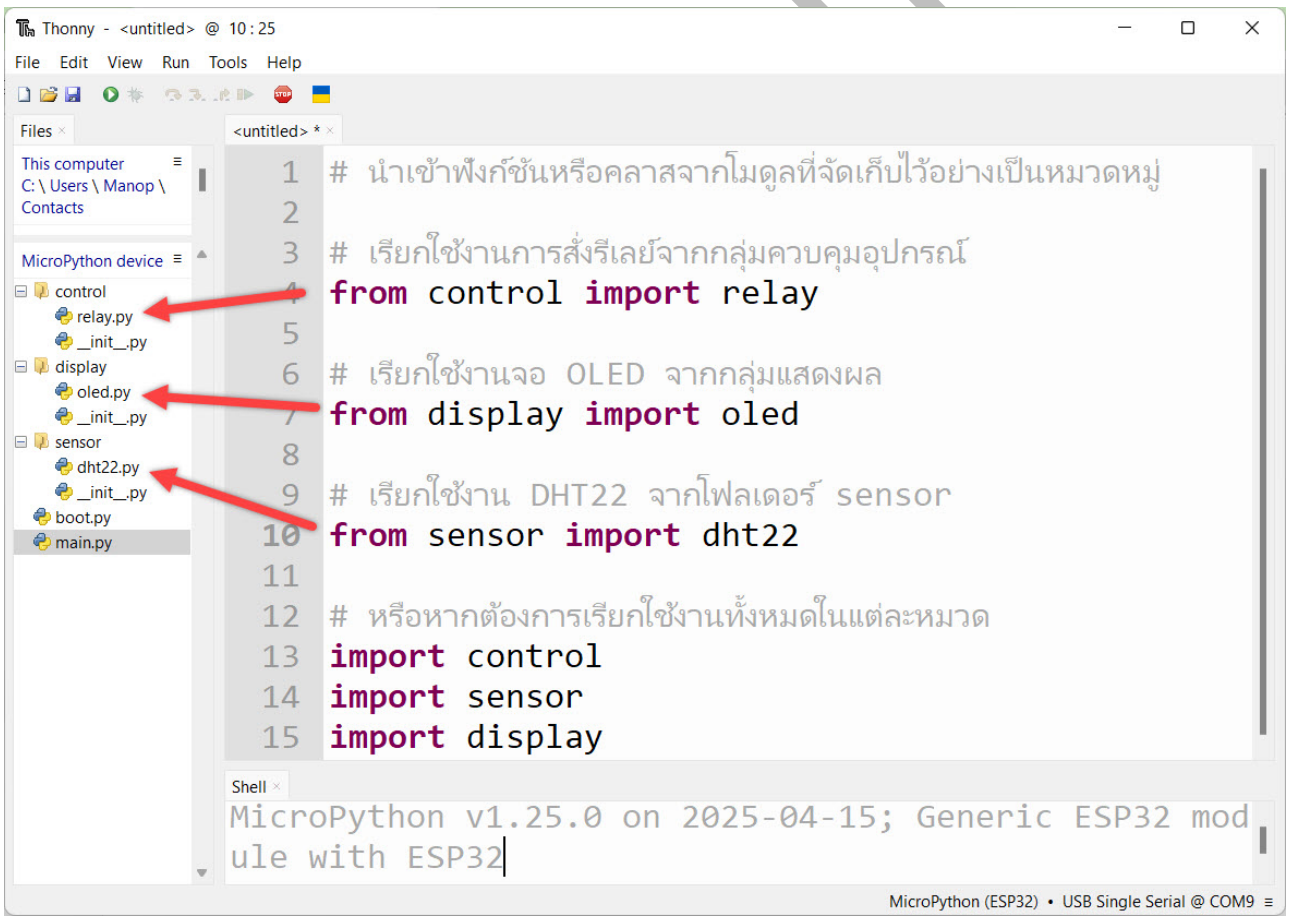
เพื่อหลีกเลี่ยงความโกลาหลของ main.py ที่ซ้ำซ้อน อาจเลือกวิธีใดวิธีหนึ่งต่อไปนี้:

1. **จัดเก็บแยกตามโฟลเดอร์บนคอมพิวเตอร์**

คือการสร้างโฟลเดอร์และตั้งชื่อไว้ให้ชัดเจนแยกเอาไว้ เมื่อจะใช้งานโปรเจกต์ใด ก็ค่อยอัปโหลด main.py จากโฟลเดอร์นั้นไปยังบอร์ด

---

2. **รวมโมดูลการทำงานแต่ละส่วนไว้ในบอร์ด แล้วใช้ main.py เป็นตัวเลือก**



---

3. **ใช้ config.txt เลือกโปรเจกต์แบบยืดหยุ่น**

ตัวอย่าง

วิธีนี้เหมาะสำหรับนักพัฒนาที่ต้องการสลับโปรเจกต์หลายชุดบนบอร์ดเดียวโดยไม่ต้องแฟลชใหม่ทุกครั้ง โดยจะใช้ไฟล์ config.txt เป็นตัวกำหนดชื่อโปรเจกต์ที่ต้องการให้รัน เช่น: เขียนในไฟล์ config.txt ว่า sensor.dht22

แล้วที่ไฟล์ main.py ให้เพิ่มโค้ดเรียกใช้เข้าไป

```
with open("config.txt") as f:
module = f.read().strip()
__import__(module)
```

แต่ผู้เขียน **ไม่แนะนำสำหรับผู้เริ่มต้น** เนื่องจากต้องพิมพ์ชื่อโมดูลให้ถูกต้อง 100% ตามโครงสร้างโฟลเดอร์ (เช่น sensor.dht22, display.oled, control.relay) หากพิมพ์ผิดเพียงตัวเดียว บอร์ดจะไม่สามารถรันโปรเจกต์ได้ และไม่มีข้อความแจ้งเตือน

ดังนั้น วิธีนี้จึงเหมาะกับนักพัฒนาระดับสูงหรือผู้ที่ต้องการระบบที่ปรับแต่งอัตโนมัติโดยเขียนโค้ดควบคุมเพิ่มเติมเอง

📝 สรุปอีกครั้ง

- บอร์ด MicroPython จะรัน main.py ทุกครั้งหลังจากจ่ายไฟเลี้ยง
- ถ้าเรามีหลายโปรเจกต์ ควรแยกโฟลเดอร์ไว้ หรือเขียน main.py ให้เลือกงานเอง
- หลีกเลี่ยงการใช้ชื่อ main.py ซ้ำกันโดยไม่จัดระเบียบ

เพราะการมีหลาย main.py โดยไม่วางแผน จะทำให้เกิดความสับสนหากมีหลายโปรเจกต์

📍 ถึงตอนนี้... เรากำลังทำอะไรอยู่?

ในบทที่ผ่านมา เราได้เริ่มต้นทดลองเขียนคำสั่งควบคุมบอร์ด ESP32 ผ่านภาษา MicroPython ไม่ว่าจะเป็นการเปิด-ปิดไฟ LED, การพิมพ์ข้อความ, หรือการทดลองสั่งงานผ่าน REPL ซึ่งทั้งหมดนี้ถือเป็นจุดเริ่มต้นที่ช่วยเปิดประตูสู่โลกของการเขียนโปรแกรมฝังตัว (Embedded Programming) อย่างเรียบง่ายและน่าตื่นเต้น เพราะทุกคำสั่งที่เราพิมพ์ลงไป บอร์ดก็ตอบสนองกลับมาทันที ไม่ว่าจะเป็นแสงไฟที่กระพริบขึ้น การแสดงผลข้อความ หรือค่าที่ส่งกลับจากเซ็นเซอร์ต่าง ๆ ความสำเร็จในระดับพื้นฐานนี้สร้างแรงจูงใจให้ผู้เรียนรู้สึกว่า “เขียนโปรแกรมได้แล้ว” อย่างเป็นรูปธรรม

ตัวอย่าง

แต่สิ่งสำคัญที่อยากให้ทุกคนตระหนักไว้ตั้งแต่เนิ่น ๆ ก็คือ ✨ สิ่งที่เราทำมาจนถึงตอนนี้ ยังเป็นเพียงแค่ “การทดลองใช้งานเบื้องต้น” เท่านั้น เพราะเราแค่พิมพ์คำสั่งแล้วดูผลลัพธ์ทันที ซึ่งก็คือการสั่งให้บอร์ดทำงานแบบสำเร็จรูปด้วยคำสั่งง่าย ๆ โดยยังไม่ได้สัมผัสกับ "โครงสร้างของภาษา Python" ที่อยู่เบื้องหลังเลย — ทั้งการนิยามตัวแปร, การกำหนดเงื่อนไข, การใช้ลูป, หรือแม้แต่ฟังก์ชันที่ทำให้โค้ดสามารถจัดระเบียบและนำกลับมาใช้ซ้ำได้อย่างเป็นระบบ

ลองจินตนาการว่าเราเดินทางไปยังประเทศที่ใช้ภาษาอังกฤษเป็นหลัก แล้วเริ่มสื่อสารได้จากประโยคพื้นฐานอย่าง “Hello”, “How are you?”, “Thank you.” ฟังดูเหมือนเราพูดภาษาอังกฤษได้แล้วใช่ไหม? แต่ในความเป็นจริง หากเรายังไม่เข้าใจไวยากรณ์, คำกริยา, การเรียงลำดับคำ หรือกฎของภาษา เราก็ยังไม่สามารถสื่อสารในสถานการณ์ที่ซับซ้อนได้อย่างมั่นใจ เช่นเดียวกันกับการเขียน MicroPython — หากเราไม่เข้าใจพื้นฐานของภาษา Python อย่างแท้จริง การพัฒนาโค้ดที่ซับซ้อน หรือการควบคุมฮาร์ดแวร์ที่มีหลายองค์ประกอบพร้อมกัน ก็จะกลายเป็นเรื่องที่ยุ่งยากและสับสนในที่สุด

💡 **แล้ว “พื้นฐานของภาษา Python” คืออะไร?** มันคือเครื่องมือชุดสำคัญที่จะทำให้เราเปลี่ยนจากผู้ใช้คำสั่งเป็นครั้งคราว มาเป็นนักพัฒนาที่ควบคุมตรรกะของโปรแกรมได้อย่างเต็มรูปแบบ ไม่ว่าจะเป็นตัวแปร (Variables), ชนิดข้อมูล (Data Types), ตัวดำเนินการ (Operators), นิพจน์ (Expressions), คำสั่งควบคุมการทำงาน (Control Statements) เช่น if, while, for, ฟังก์ชัน (Functions), โมดูล (Modules), การจัดการข้อผิดพลาด (Error Handling) ตลอดจนแนวคิดเชิงวัตถุ (OOP) ที่เป็นรากฐานของการพัฒนาระบบในระดับมืออาชีพ

🧠 **ทำไมเราจึงต้องเข้าใจสิ่งเหล่านี้?** เพราะการเขียนโปรแกรมควบคุมอุปกรณ์จริง ๆ ไม่ได้จบแค่การทำให้ไฟติดหรือดับ แต่คือการสั่งงานแบบมีเงื่อนไข ควบคุมตามเวลา จัดการหลายอุปกรณ์พร้อมกัน รอรับค่าจากเซ็นเซอร์หลายตัว หรือแม้แต่ทำงานร่วมกับเครือข่ายอินเทอร์เน็ต หากเราไม่มีความเข้าใจในภาษา Python การพัฒนาโปรเจกต์แบบนี้จะกลายเป็นการลองผิดลองถูกที่เต็มไปด้วยความเครียด แต่หากเรารู้จักภาษาอย่างแท้จริง แม้โค้ดจะยากขึ้น เราก็จะสามารถเขียน วิเคราะห์ และแก้ไขได้อย่างเป็นระบบด้วยตนเอง

📘 สรุปก่อนเข้าสู่บทถัดไป สิ่งที่เราได้เรียนรู้คือการใช้งาน MicroPython อย่างง่าย ซึ่งเป็นจุดเริ่มต้นที่ยอดเยี่ยม แต่ยังไม่เพียงพอหากเราต้องการสร้างระบบจริงที่มีตรรกะซับซ้อน หรือสามารถนำไปประยุกต์ใช้ในงานต่าง ๆ ได้อย่างมั่นใจ ต่อจากนี้ เราจะเริ่มต้นเรียนรู้ "ภาษา Python" อย่างจริงจัง เพื่อให้เราสามารถพัฒนาโปรแกรมของตนเองได้อย่างมั่นคง เป็นระบบ และพร้อมต่อยอดสู่โปรเจกต์ที่ใช้งานได้จริงในอนาคต

ตัวอย่างหนังสือ

ในบทถัดไป (บทที่ 3) เราจะทบทวนพื้นฐานภาษา Python ที่จำเป็นและมักใช้งานบ่อยในการเขียนโปรแกรมสำหรับไมโครคอนโทรลเลอร์ เพื่อให้ผู้อ่านมีความเข้าใจในโครงสร้างภาษาและคำสั่งต่างๆ เพียงพอที่จะนำไปประยุกต์ใช้ในการควบคุมฮาร์ดแวร์ได้อย่างมั่นใจ เตรียมพบกับความสนุกในการเรียนรู้ภาษา Python ในบริบทของ MicroPython กันได้เลย!

ส่วนที่ 2: พื้นฐาน MicroPython สำหรับควบคุมฮาร์ดแวร์

บทที่ 3: กำเนิดและโครงสร้างของภาษา Python (สำหรับ MicroPython)

ก่อนที่เราจะเริ่มเรียนรู้การเขียนโปรแกรมเพื่อควบคุมฮาร์ดแวร์ด้วย MicroPython อย่างเป็นระบบ เราควรทำความเข้าใจรากฐานของภาษา Python เสียก่อน เพราะ MicroPython แม้จะเป็นเวอร์ชันย่อของ Python แต่ก็ยังคงใช้โครงสร้างหลักเดียวกันเกือบทั้งหมด หากเข้าใจพื้นฐานของ Python อย่างถ่องแท้แล้ว การต่อยอดไปสู่การควบคุมเซ็นเซอร์ การสื่อสารข้อมูล หรือการจัดการเงื่อนไขซับซ้อนต่าง ๆ ก็จะง่ายขึ้นอย่างมาก

3.1 กำเนิดของภาษา Python

ภาษา Python ถูกออกแบบโดย Guido van Rossum ในช่วงปลายทศวรรษ 1980 และเปิดตัวอย่างเป็นทางการในปี 1991 โดยมีเป้าหมายเพื่อให้เป็นภาษาที่อ่านง่าย เข้าใจง่าย และมีไวยากรณ์ที่ชัดเจน ไม่ซับซ้อนเหมือนภาษา C หรือ Java ซึ่งเป็นที่นิยมในยุคนั้น Python ได้รับแรงบันดาลใจจากภาษาการเขียนโปรแกรม ABC (ที่พัฒนาโดย CWI – ศูนย์วิจัยของเนเธอร์แลนด์) และมีเป้าหมายที่จะใช้งานได้ทั้งเพื่อการเรียนการสอนและพัฒนาแอปพลิเคชันจริงจัง ปัจจุบัน Python ถูกนำไปใช้ทั้งในงานวิทยาศาสตร์ข้อมูล, ปัญญาประดิษฐ์, ระบบเว็บ, งานอัตโนมัติ, รวมถึงงาน Embedded ที่เราใช้กับไมโครคอนโทรลเลอร์เช่น ESP32 ด้วย

3.2 โครงสร้างของภาษา Python

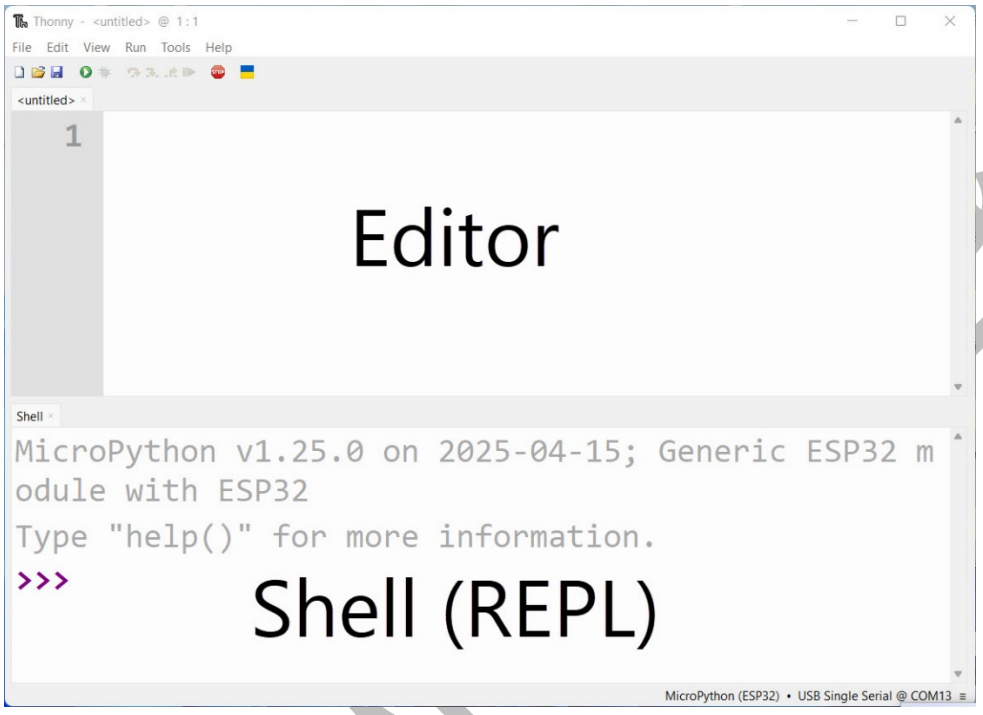
Python เป็นภาษาระดับสูง (High-level Language) ที่ถูกออกแบบมาให้มีลักษณะ:

- **อ่านง่าย**: ใช้การจัดย่อหน้า (Indentation) เพื่อกำหนดขอบเขตของโค้ดแทนการใช้วงเล็บปีกกา
- **ไดนามิก**: ไม่จำเป็นต้องกำหนดชนิดของตัวแปรล่วงหน้า (Dynamic Typing)
- **กระชับ**: เขียนโค้ดให้น้อยที่สุด แต่สามารถสื่อความหมายได้ชัดเจน

สุภาพ ปุกาย

- **เป็นเชิงวัตถุ (OOP)**: สนับสนุนการเขียนแบบวัตถุเป็นพื้นฐาน พร้อมด้วยแนวคิดคลาส (Class) และอ็อบเจกต์ (Object)

ก่อนจะทำความเข้าใจเรื่องของโครงสร้างภาษา เราจะมาดูเครื่องมือกันก่อน เพราะเราจะต้อง ทำตามตัวอย่างในหนังสือเพื่อเพิ่มความเข้าใจจริง ใน Thonny มีอยู่ 2 ส่วนคือ ช่อง Editor ข้างบน กับ ช่อง Shell (REPL) ด้านล่าง



3.2.1 ความแตกต่างระหว่าง Shell (REPL) และ Editor ใน Thonny

เมื่อเริ่มต้นใช้งาน Thonny IDE เพื่อเขียนโปรแกรมด้วย MicroPython บนบอร์ดไมโครคอนโทรลเลอร์ เช่น ESP32 ผู้เรียนจะพบว่าหน้าต่างโปรแกรมมีสองส่วนหลัก ได้แก่ “Shell” (หรือ REPL) และ “Editor” ซึ่งทั้งสองส่วนนี้มีวัตถุประสงค์และลักษณะการใช้งานที่แตกต่างกันอย่างชัดเจน **การใช้งานเมนูต่างๆในโปรแกรมจะอยู่ท้ายเล่ม**

Shell (REPL):

REPL ย่อมาจาก Read-Eval-Print Loop ซึ่งหมายถึงการรับคำสั่งจากผู้ใช้ ประมวลผล แสดงผล และรอรับคำสั่งใหม่วนไปเรื่อย ๆ ส่วนนี้อยู่ที่ด้านล่างของหน้าต่าง Thonny โดยผู้ใช้สามารถพิมพ์คำสั่ง Python ลงไปโดยตรง เช่น print("Hello"), led.on() หรือ 2 + 3 แล้วกด Enter ระบบจะประมวลผลและแสดงผลทันทีในบรรทัดถัดไป

REPL เหมาะสำหรับการทดลองคำสั่งสั้น ๆ การทดสอบแนวคิดเบื้องต้น หรือการตรวจสอบค่าของตัวแปรในขณะทำงาน แต่ข้อจำกัดของ REPL คือ คำสั่งต่าง ๆ ที่พิมพ์ลงไปจะไม่ถูกบันทึกไว้เป็นโปรแกรม และจะหายไปเมื่อปิดบอร์ดหรือปิดโปรแกรม หรือให้เข้าใจง่ายๆ ก็คือโค้ดที่เขียนในช่องนี้จะไม่ได้ถูกบันทึกไว้ในบอร์ดเลย ไม่ใช่การอัปโหลดเหมือนใน Arduino

Editor:

ส่วนของ Editor อยู่ที่ด้านบนของ Thonny เป็นพื้นที่สำหรับเขียนโปรแกรมเต็มรูปแบบ ซึ่งประกอบด้วยคำสั่งหลายบรรทัด ฟังก์ชัน หรือโครงสร้างควบคุมแบบซับซ้อน โปรแกรมที่เขียนใน **Editor** จะต้องสั่งให้ทำงานด้วยการกดปุ่ม “**Run**” หรือกดปุ่มลัด **F5** เพื่อส่งโค้ดทั้งหมดไปยังบอร์ด ESP32 คล้ายกับการการกดอัปโหลดใน Arduino นั่นเอง

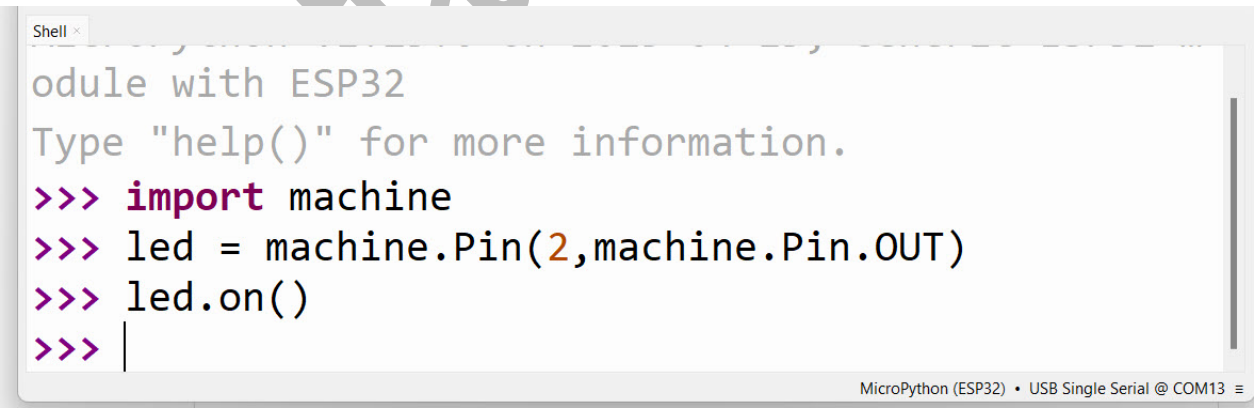
โค้ดใน Editor สามารถบันทึกไว้ในรูปแบบไฟล์ .py และนำกลับมาใช้งานใหม่ได้ในภายหลัง จึงเหมาะสำหรับการเขียนโปรเจกต์จริงที่มีความซับซ้อนหรือโค้ดที่ต้องใช้ซ้ำหลายครั้ง

เปรียบเทียบการใช้งาน:

- ถ้าเปรียบ REPL เป็น “การพิมพ์เขียนโค้ด เพื่อสื่อสารกับบอร์ดแบบสด ๆ”
  Editor ก็เหมือน “เขียนจดหมายยาว ๆ ให้บอร์ดทำงานตามลำดับ”
- REPL ใช้ได้เร็ว เห็นผลทันที
  Editor ต้องสั่งรันก่อนถึงจะเห็นผล
- REPL เหมาะสำหรับการทดลองเฉพาะหน้า
  Editor เหมาะสำหรับเขียนงานจริงที่ต้องการความต่อเนื่อง

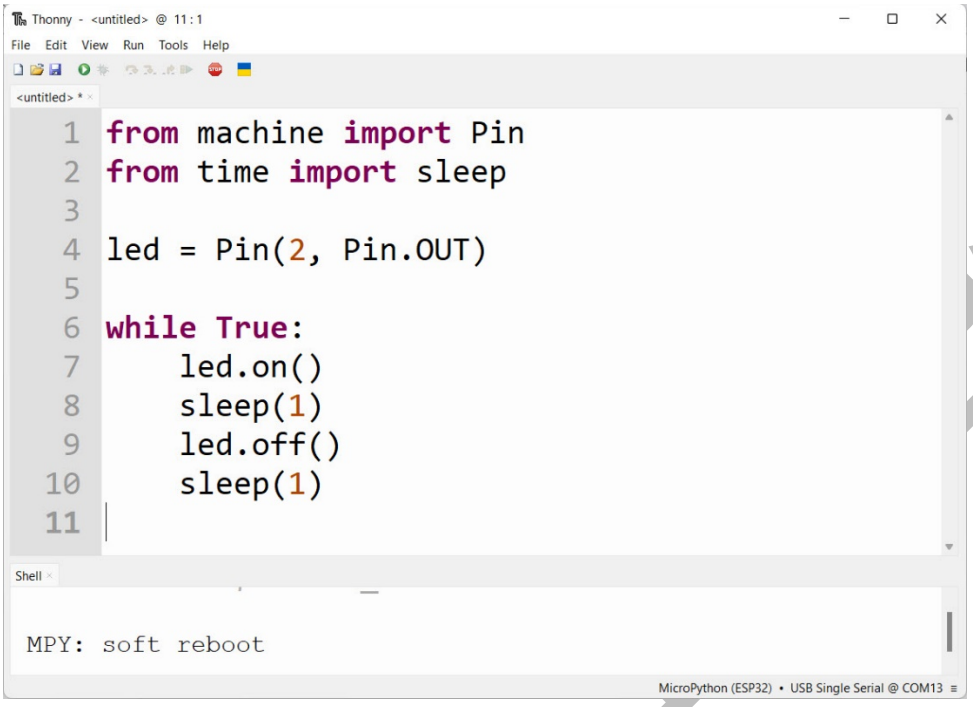
ตัวอย่างการใช้งาน:

ทดลองสั่งให้ LED ติดทันที: แต่หากปิดโปรแกรม led จะไม่ติดอีกต่อไป

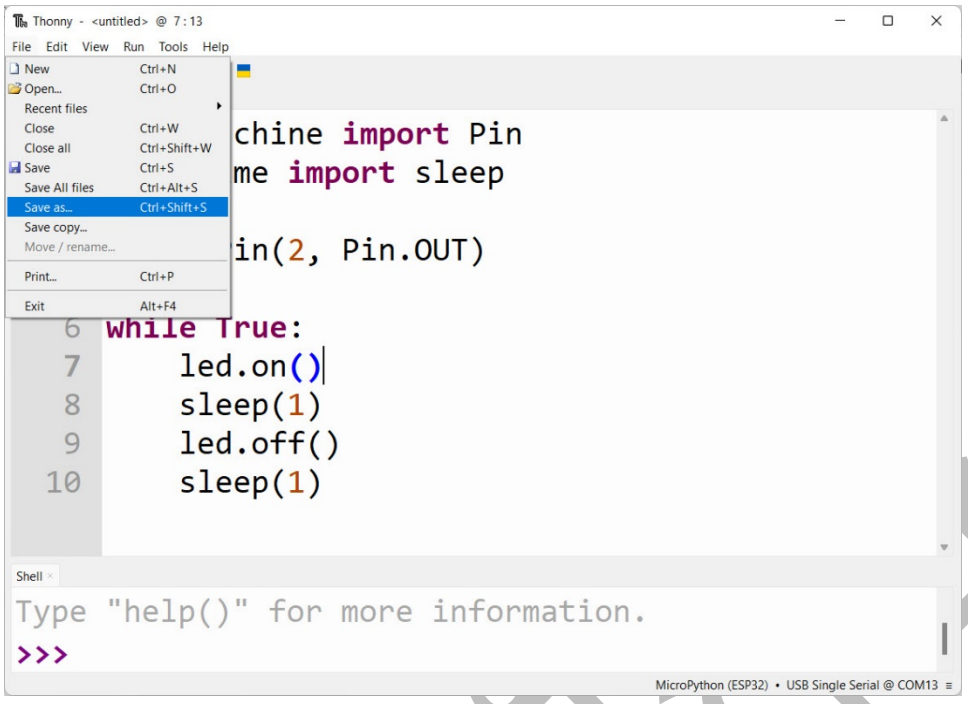


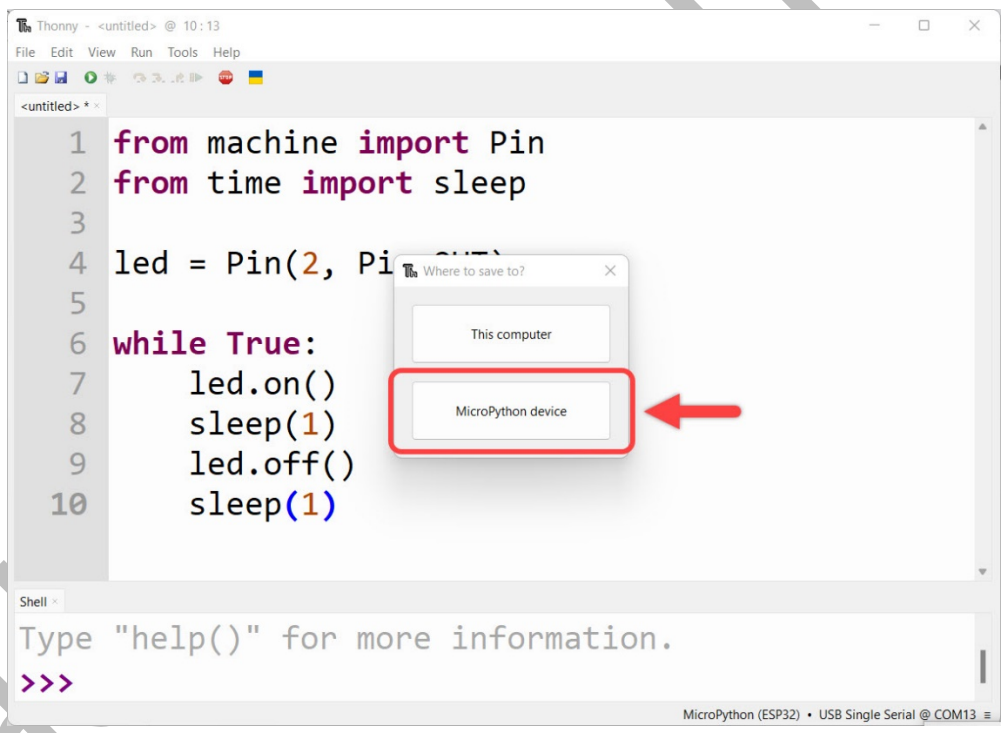
สุพัฒ ปัทมี

เขียนโค้ดกระพริบไฟแบบวนซ้ำ: จากนั้นต้อง **กดปุ่ม Run** เพื่อให้โค้ดเริ่มทำงาน แต่หากปิดโปรแกรมและถอดสายออกแล้วเอามาเสียบใหม่บอร์ดก็จะยังไม่ทำงานเหมือนกัน การจะให้บอร์ดทำงานได้เองจะต้องบันทึกไฟล์เอาไว้ในหน่วยความจำของบอร์ด

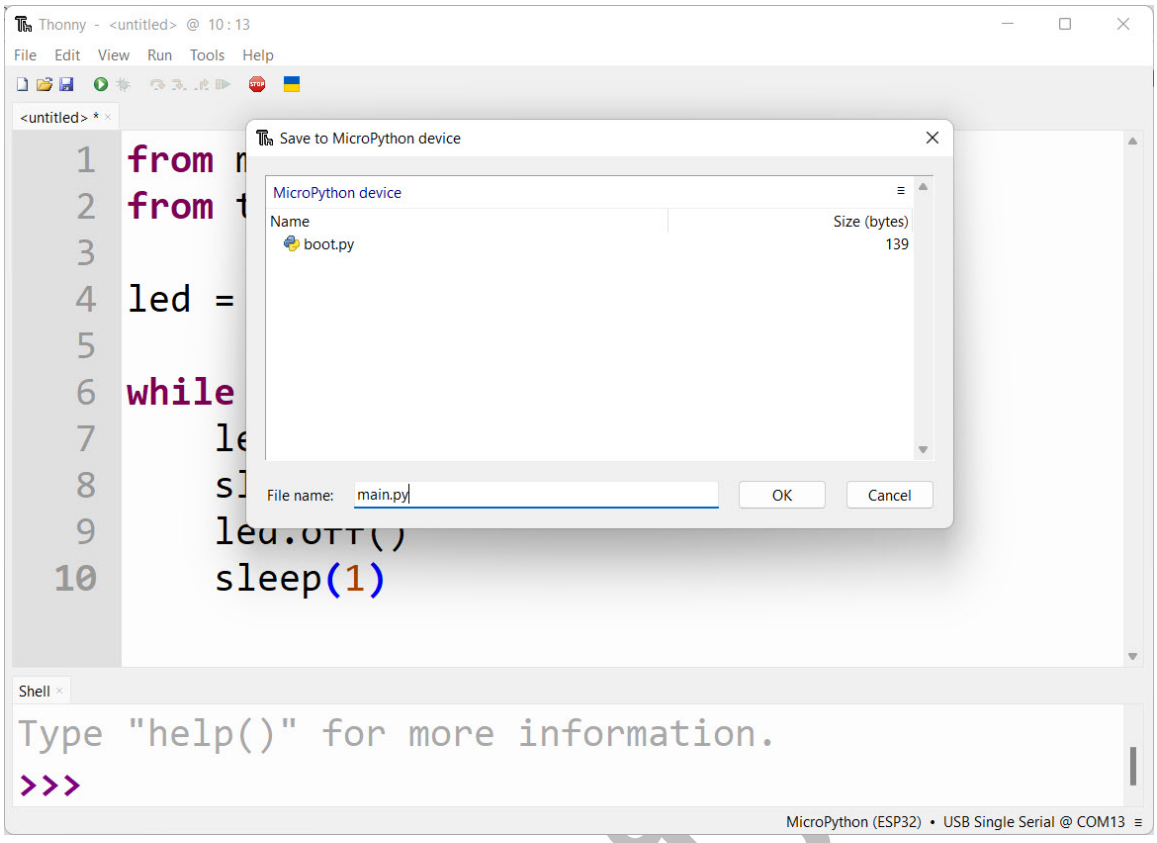


การบันทึกที่เหมือนกับการอัปโหลดโปรแกรมลงบอร์ดให้ไปที่เมนู File เลือก save หรือ save As จะมีหน้าต่างเล็ก ๆ เด้งขึ้นมาให้เราเลือกว่าจะบันทึกไว้ที่ไหน หากต้องการบันทึกไว้ในบอร์ดให้เลือก MicroPython device ดังรูป





และเมื่อกดแล้วจะมีหน้าต่างใหม่ขึ้นมาให้ตั้งชื่อไฟล์ว่า **main.py** แล้วกด OK



หากบันทึกแล้วไฟไม่กระพริบให้กดปุ่ม Reset หรือถอดสาย USB ออกแล้วเสียบใหม่

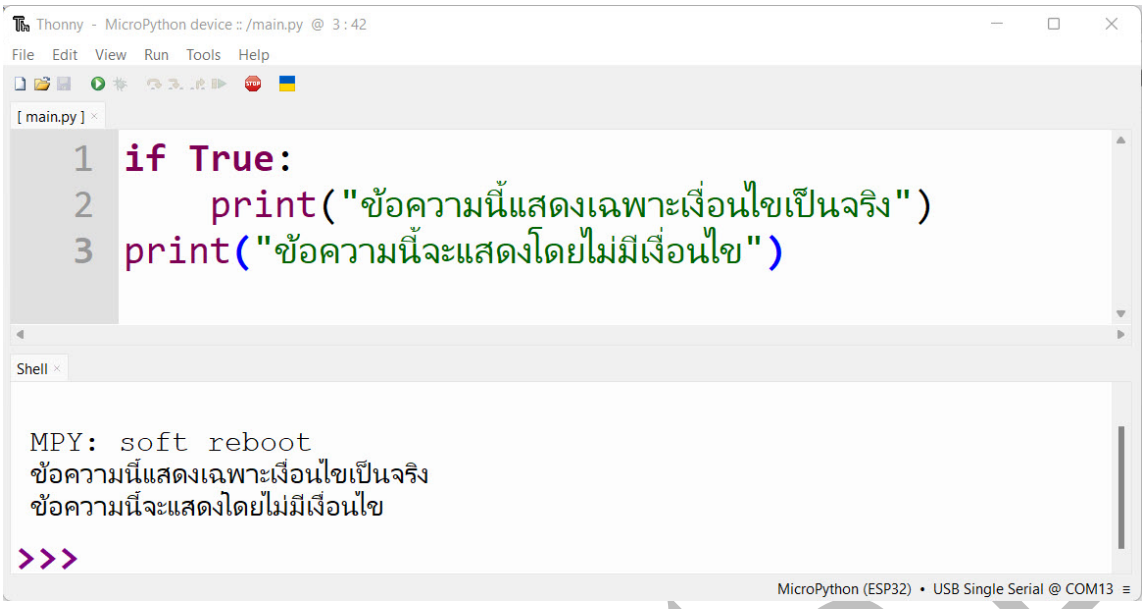
การเข้าใจความแตกต่างระหว่าง REPL และ Editor เป็นพื้นฐานสำคัญที่ผู้เริ่มต้นควรเรียนรู้ เพื่อสามารถเลือกใช้ให้เหมาะสมกับจุดประสงค์แต่ละช่วงในการพัฒนาโปรแกรมด้วย MicroPython บน ESP32 ได้อย่างมีประสิทธิภาพ

3.3 จุดเด่นของภาษา Python ที่ควรเข้าใจ

เพื่อให้เข้าใจและใช้ภาษา Python ได้อย่างมั่นใจในการเขียน MicroPython ควรเข้าใจจุดเด่นสำคัญของภาษา ซึ่งแตกต่างจากภาษาคอมพิวเตอร์อื่น ๆ โดยเฉพาะผู้ที่มีพื้นฐานจากภาษา C หรือ C++ จะพบว่ามีความแตกต่างหลายจุด เช่น:

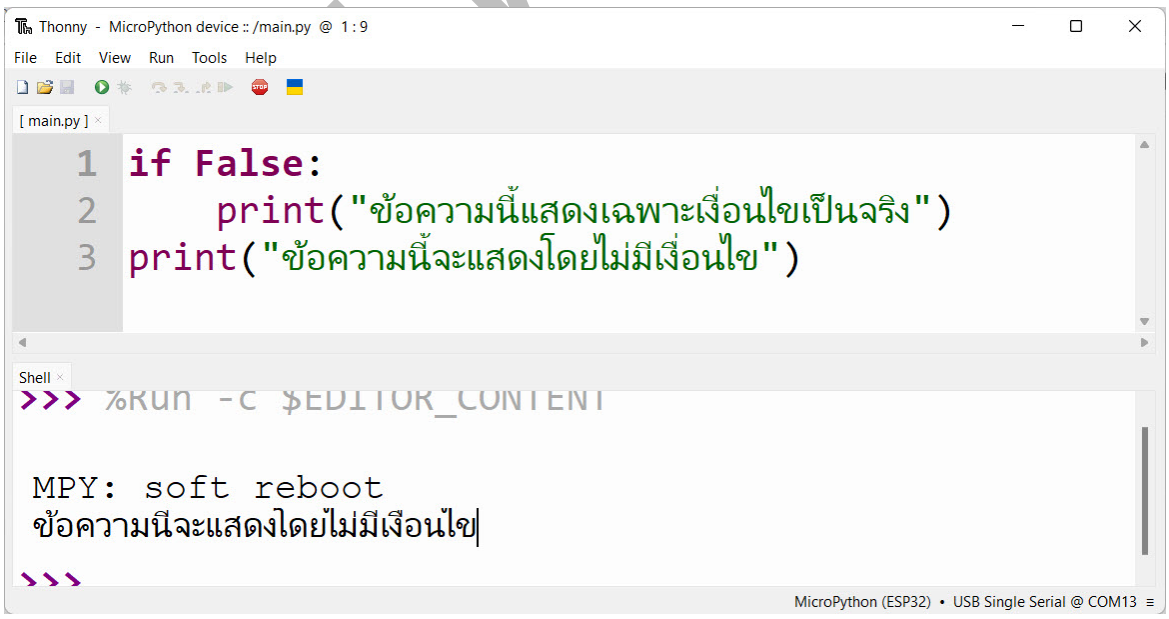
- **อ่านง่าย (Readable)**

Python ใช้ "การจัดย่อหน้า" หรือ **Indentation** เพื่อกำหนดขอบเขตของโค้ด แทนการใช้เครื่องหมายวงเล็บปีกกา {} เหมือนในภาษา C/C++

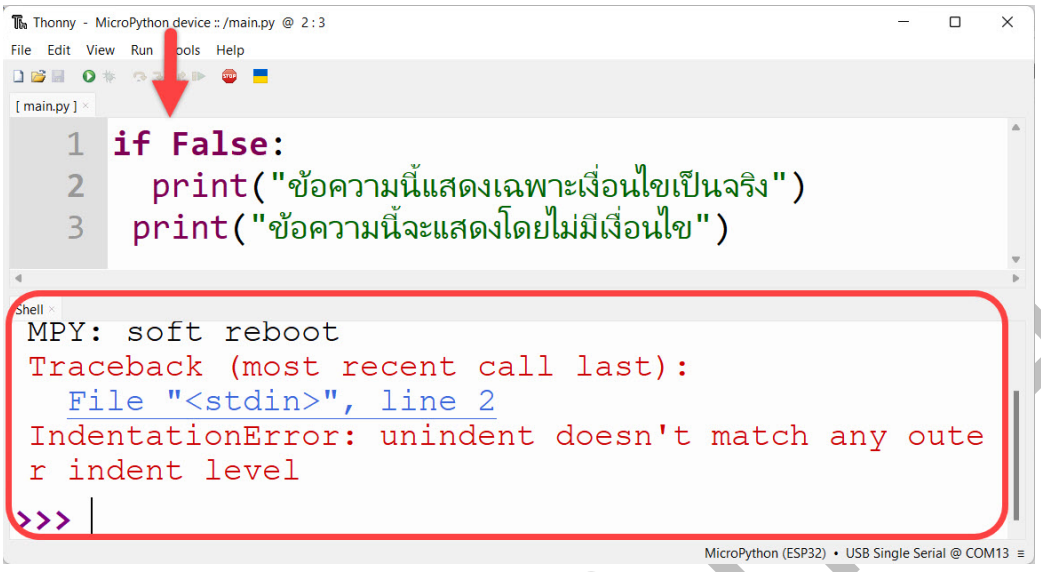


จากภาพจะเห็นว่าบรรทัดที่ 2 ถูกย่อหน้า หากเทียบกับภาษา C คืออยู่ในวงเล็บปีกกานั่นเอง

ในตัวอย่างนี้ print("ข้อความนี้แสดงเฉพาะเงื่อนไขเป็นจริง") จะถูกดำเนินการเมื่อเงื่อนไข **if True:** เป็นจริง เพราะมันอยู่ภายใต้การย่อหน้า **(Tab หรือ Space 4 ช่อง)** ส่วนบรรทัดที่ 3 ก็จะแสดงไม่ว่าเงื่อนไขจะเป็นจริงหรือเท็จ เพราะไม่มีย่อหน้า python จะถือว่าไม่ได้อยู่ในเงื่อนไข ถ้าเปรียบเทียบกับภาษา C ก็คืออยู่นอกวงเล็บปีกกาของ if นั่นเอง และเมื่อเราทดลองเปลี่ยนเงื่อนไขให้เป็นเท็จ หรือ False จะเห็นว่าข้อความบรรทัดที่ 2 จะไม่แสดงออกมา จะมีเพียงข้อความบรรทัดที่ 3 เท่านั้น



และถ้าเราทดลองเขียนแบบไม่มีการย่อหน้า หรือย่อหน้าไม่ถูกต้อง โปรแกรมจะไม่สามารถรันได้เลย จะแสดง error ทันที ถือเป็นจุดเด่นที่ช่วยให้โค้ดดูสะอาด อ่านง่าย และลดโอกาสเกิดข้อผิดพลาดจากการลืมปิดวงเล็บ

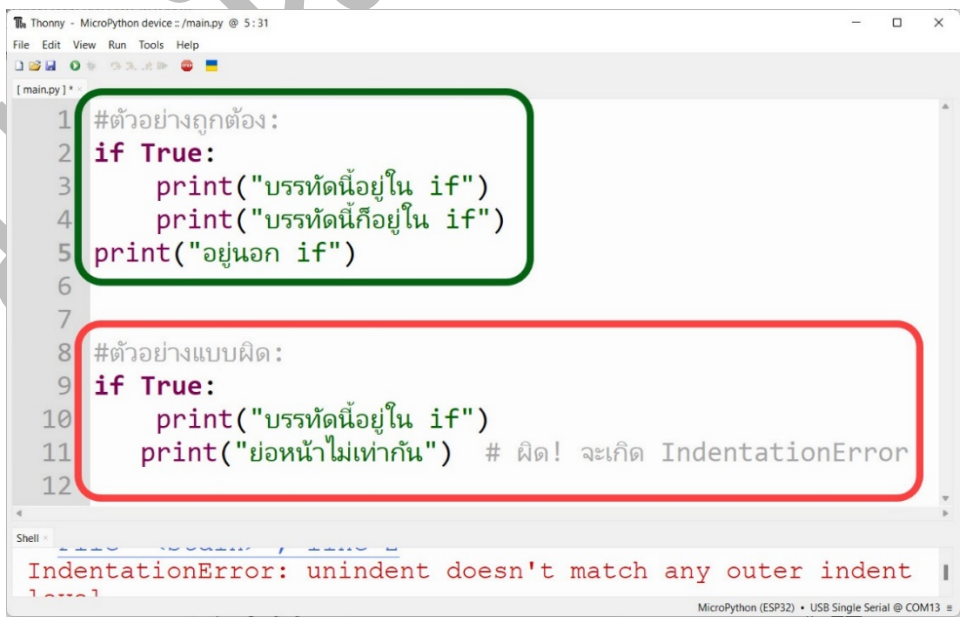


จะรู้ได้อย่างไรว่าต้องย่อหน้า (Indentation) กี่ที?

ทุกครั้งที่เขียนคำสั่งภายใต้ if, for, while, def, class, หรือ try — ต้องย่อหน้าเข้าไป 1 ระดับ และทุกคำสั่งที่อยู่ในกลุ่มเดียวกันต้องย่อหน้าให้อยู่ระดับเดียวกันทั้งหมด

- **ย่อหน้าแบบมาตรฐาน**: ใช้ Space 4 ครั้ง (หรือ 1 Tab หากตั้งค่าตัวแก้ไขให้ใช้แทนกันได้)
- **ห้ามผสม Tab และ Space ในโปรแกรมเดียวกันเด็ดขาด**

ตัวอย่างเช่น



🔧 **เคล็ดลับ**: ใช้โปรแกรม Thonny IDE ซึ่งช่วยย่อหน้าอัตโนมัติเมื่อพิมพ์คำสั่ง เช่น if, for, while แล้วกด Enter โดยเมื่อพิมพ์เสร็จแล้วกดปุ่ม Enter โปรแกรม จะย่อหน้าให้อัตโนมัติอยู่แล้ว แต่ความผิดพลาดอาจเกิดขึ้นได้หากมีการเคาะเพิ่มเติมหรือแก้ไขแทรกโค้ดระหว่างบรรทัดด้วยตัวเอง

- **ไดนามิก (Dynamic Typing)**

  การเขียนโปรแกรมภาษา Python ไม่จำเป็นต้องระบุชนิดของตัวแปรก่อนใช้งาน เช่น ไม่ต้องระบุว่าชนิดของตัวแปรนั้นว่าจะใช้เก็บข้อมูลชนิดไหน เป็น int, float, หรือ string เหมือนภาษา C/C++ หรือ Java

  ในภาษา Python ผู้ใช้เพียงกำหนดค่าลงไป ตัวแปรจะได้รับชนิดตามค่าที่กำหนดให้โดยอัตโนมัติ

🔴 **ข้อควรรู้: ใน Python ถ้ายังไม่กำหนดค่าให้ตัวแปร ก็ไม่สามารถประกาศได้ล่วงหน้าแบบเปล่า ๆ**

มานพ ปัทกิ