

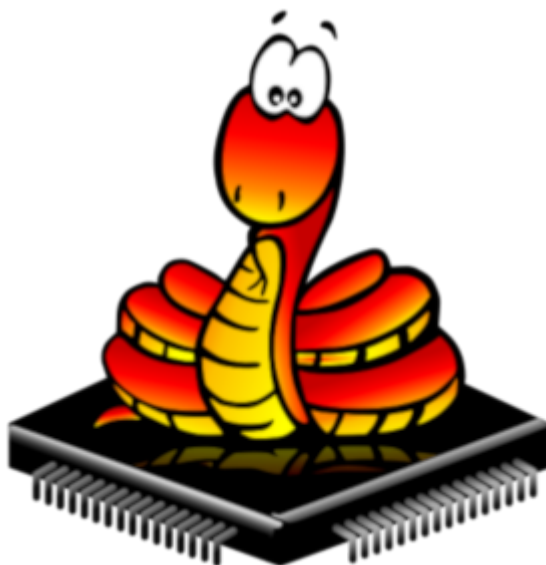
Hardware Programming with MicroPython

แนะนำการใช้งาน MicroPython สำหรับบอร์ดไมโครคอนโทรลเลอร์ประเภทต่าง ๆ

ไมโครไพธอน (MicroPython)

เรียบเรียงโดย เรวัต ศิริโกศาภิรมย์

โดยทั่วไปแล้ว สำหรับการเขียนโปรแกรมสำหรับไมโครคอนโทรลเลอร์ เราจะใช้ภาษาคอมพิวเตอร์ เช่น C/C++ แต่ถ้าจะใช้ภาษาอื่นในประเภท **Scripting Language** อย่างเช่น **Python 3** จะเป็นไปได้หรือไม่ ?



MicroPython Logo (Image Source: Wiki)

ไมโครไพธอนเป็นซอฟต์แวร์ประเภท **Open Source** ทำหน้าที่เป็นเฟิร์มแวร์ (**Firmware**) สำหรับไมโครคอนโทรลเลอร์ที่ช่วยให้เราสามารถเขียนและรันโค้ดไพธอนได้

สำหรับผู้ที่มีพื้นฐานภาษาไพธอนมาบ้างแล้ว การเลือกใช้ไมโครไพธอน อาจช่วยให้การเรียนรู้เกี่ยวกับการเขียนโปรแกรมและใช้งานไมโครคอนโทรลเลอร์สำหรับผู้เริ่มต้น ทำได้ง่ายขึ้น เมื่อเปรียบเทียบกับการใช้งานภาษา C/C++

ประวัติความเป็นมาของ **MicroPython** นั้นเริ่มต้นโดย **Damien P. George** ซึ่งเป็นนักฟิสิกส์ชาวออสเตรเลีย มีความคิดที่จะเขียนโปรแกรมไมโครคอนโทรลเลอร์ โดยใช้ภาษา **Python** ขณะที่ทำงานเป็น **Post-Doctoral Fellow** และทำวิจัยด้านอนุภาคพลังงานสูง ในมหาวิทยาลัย **Cambridge** ประเทศอังกฤษ (UK) ตอนเรียนระดับป.ตรี เขามีโอกาสได้ร่วมทีม **RobotCup** แข่งขันหุ่นยนต์เตะฟุตบอล (**Robot Soccer League**) จึงมีประสบการณ์ด้านฮาร์ดแวร์ อิเล็กทรอนิกส์ และเขียนโปรแกรมด้านสมองกลฝังตัว

เขาได้เลือกใช้ **Python 3** และพัฒนาโปรแกรมหรือเฟิร์มแวร์ เช่น **Python Run-Time Interpreter** สำหรับไมโครคอนโทรลเลอร์ 32 บิต และสร้างบอร์ด **PyBoard** และระดมทุนใน **Kickstarter** ในช่วงปี ค.ศ. **2013-2014** ได้ผู้มาสนับสนุนราว 2,000 ราย (**1,931 backers**) และได้ระดมทุนเงินสูงเกือบ £100,000 ซึ่งเป็นจุดเริ่มต้นของ **MicroPython** และเผยแพร่ซอฟต์แวร์ โดยใช้ลิขสิทธิ์ **MIT license**

ต่อมาในปีค.ศ. **2015** องค์กรที่มีชื่อว่า **European Space Agency (ESA)** ของสหภาพยุโรป ได้ให้การสนับสนุนโครงการ **MicroPython** อีกด้วย และในช่วงเวลาใกล้เคียงกัน **British Broadcasting Corporation (BBC)** ในประเทศอังกฤษ ได้พัฒนาบอร์ด **Micro:bit** และ ไมโครไพธอนได้ถูกนำมาปรับให้ใช้งานได้สำหรับบอร์ดดังกล่าว

ความสำเร็จของไมโครไพธอน ได้ดึงดูดความสนใจของ **Limor "Ladyada"** ผู้ก่อตั้งบริษัท **Adafruit Industries** ในสหรัฐอเมริกา ทางบริษัทได้พัฒนาไลบรารีสำหรับอุปกรณ์หรือโมดูลฮาร์ดแวร์เสริมหลายรูปแบบ (โดยส่วนใหญ่ก็เป็นฮาร์ดแวร์ต่าง ๆ ที่ทางบริษัทได้ผลิตและจำหน่าย) เพื่อรองรับการใช้งานไมโครไพธอน รวมถึงการพัฒนาบอร์ดไมโครคอนโทรลเลอร์ที่นำมาใช้ได้กับไมโครไพธอน เช่น บอร์ด **Circuit Playground Express (SAMD21)** และต่อได้มาพัฒนา **CircuitPython** ต่อยอดมาจากไมโครไพธอน

บอร์ดที่ใช้ชิปหรือโมดูล **ESP8266** และ **ESP32** ของบริษัท **Espressif Systems** ซึ่งมีความสามารถในการเชื่อมต่อ **Wi-Fi** มีราคาไม่แพง และนำไปใช้งานด้าน **IoT** ก็สามารถนำมาใช้งานร่วมกับไมโครไพธอนได้เช่นกัน ในปัจจุบันขณะที่เขียนบทความนี้ จะเห็นได้ว่า มีฮาร์ดแวร์หลายรูปแบบที่สามารถนำมาใช้ร่วมกับไมโครไพธอน

ข้อสังเกตเกี่ยวกับ MicroPython

- เขียนโค้ด **Python 3** เพื่อเข้าถึงและใช้งานฮาร์ดแวร์ภายในไมโครคอนโทรลเลอร์ได้ง่าย เช่น **GPIO, Timer, SPI, I2C, UART, ADC, DAC, Wi-Fi** เป็นต้น
- เนื่องจากมีข้อจำกัดเรื่องหน่วยความจำ จึงมีการนำไลบรารีที่ใช้ได้กับ **Python 3** จำนวนหนึ่งมาใช้งาน และมีการตั้งชื่อให้เริ่มต้นด้วยตัวอักษร 'u' เช่น **utime, urandom** เป็นต้น สามารถดูรายการของไลบรารีที่ใช้ได้จาก **Micropython-Lib**
- สามารถสื่อสารกับไมโครไพธอนได้แบบ **interactive** หรือที่เรียกว่า **REPL (Read, Eval, Print, Loop)** ผ่านพอร์ต **USB-to-Serial** หรือ **USB-CDC**

- ในกรณีที่ใช้ชิป Wi-Fi SoC เช่น ESP8266 หรือ ESP32 ก็สามารถใช้ REPL (WebREPL) แบบไร้สายผ่าน Wi-Fi ได้

การเลือกใช้บอร์ดไมโครคอนโทรลเลอร์สำหรับ MicroPython

สำหรับผู้ที่จะลองใช้หรือเขียนโค้ดไมโครไพธอน ร่วมกับบอร์ดไมโครคอนโทรลเลอร์ ในปัจจุบัน มีไมโครคอนโทรลเลอร์ให้เลือกใช้ได้หลายตระกูล (32 บิต) อาจลองพิจารณาตัวเลือกเหล่านี้

- บอร์ดไมโครคอนโทรลเลอร์ STM32 ซึ่งมีหลายรุ่น แต่แนะนำให้ใช้ STM32F4 (ARM Cortex-M4F), STM32L4, STM32F7 อาจเป็นบอร์ดที่ทางผู้พัฒนา MicroPython ผลิตขาย หรืออาจเป็นบอร์ดอื่น เช่น STM32 Nucleo เป็นต้น
 - บอร์ด **PyBoard** ใช้ชิป STM32F405 (ARM Cortex-M4F, 168 MHz, 1,024 KB Flash ROM, 192 KB SRAM) ซึ่งถือว่าเป็นชิปไมโครคอนโทรลเลอร์ 32 บิต ที่มีประสิทธิภาพสูง บอร์ดมีขนาดเล็ก แต่รูปแบบของบอร์ดไม่เหมาะกับการใช้งานบนเบรตบอร์ด
 - บอร์ด **PyBoard D-Series** เป็นรุ่นถัดจาก PyBoard และใช้ตัวประมวลผล STM32F7 ซึ่งมีความเร็วในการประมวลผลในระดับที่สูงกว่า STM32F4
- บอร์ดไมโครคอนโทรลเลอร์ ESP8266 หรือ ESP32 ของบริษัท Espressif ซึ่งมีข้อดีคือ สามารถเชื่อมต่อผ่าน Wi-Fi ได้ ไปยังอินเทอร์เน็ต จึงเหมาะสำหรับการประยุกต์ใช้งานด้าน IoT ในปัจจุบันก็มีบอร์ดหลายรูปแบบให้เลือกใช้งานและราคาแตกต่างกันไป
 - **NodeMCU DevKit**
 - **Adafruit Feather HUZZAH ESP8266**
 - **Sparkfun ESP32 Thing**
 - **WeMos D1 mini**
 - **WeMos Lolin32**
 - **WeMos LOLIN D32**
 - **WROOM-32 DEVKIT V4**
 - **PyCom WiPy**
- บอร์ด **BBC Micro:bit** ซึ่งใช้ตัวประมวลผล nRF51822 (ARM Cortex-M0, 16MHz, 256 KB Flash, 16KB SRAM) และสามารถสื่อสารไร้สายด้วย Bluetooth 4.0 ได้ และอาจจะเหมาะสำหรับผู้เรียนที่มีประสบการณ์หรือเริ่มต้นใช้งานบอร์ดนี้มาก่อน เช่น การเขียนโปรแกรมโดยการต่อบล็อกและใช้ **Microsoft MakeCode for Micro:bit** เป็นต้น
- บอร์ด **Adafruit's Circuit Playground Express** มีลักษณะเป็นทรงกลม มีขา I/O pins อยู่รอบ ใช้ตัวประมวลผล **ATSAMD21G18** (ARM Cortex-M0, 48 MHz, 256KB Flash, 32 KB SRAM, 2MB external SPI Flash)

ข้อสังเกต: ในปัจจุบันก็มีบอร์ดหรือโมดูล ESP32 ให้เลือกใช้ได้หลายรูปแบบ แต่ก็สามารถจำแนกได้เป็นสองกลุ่ม (เรียกว่า **non-psRAM** กับ **psRAM-enabled**) ซึ่งขึ้นอยู่กับว่า มีการเพิ่มชิป **psRAM (pseudo-static RAM)** ภายนอกหรือไม่ มีความจุตั้งแต่ **4MB** หรือ **8MB** เป็นต้น บอร์ด ESP32 ที่มี **psRAM** ก็มักจะมียาราคาสูงกว่าบอร์ดที่ไม่มี แต่ก็เหมาะสำหรับนำมาใช้กับไมโครไพธอน ทำให้มีหน่วยความจำเพิ่มขึ้นสำหรับสร้าง **Stack/Heap** ได้ขนาดใหญ่ขึ้น เช่น การเก็บข้อมูลในอาร์เรย์ได้มากขึ้น เป็นต้น

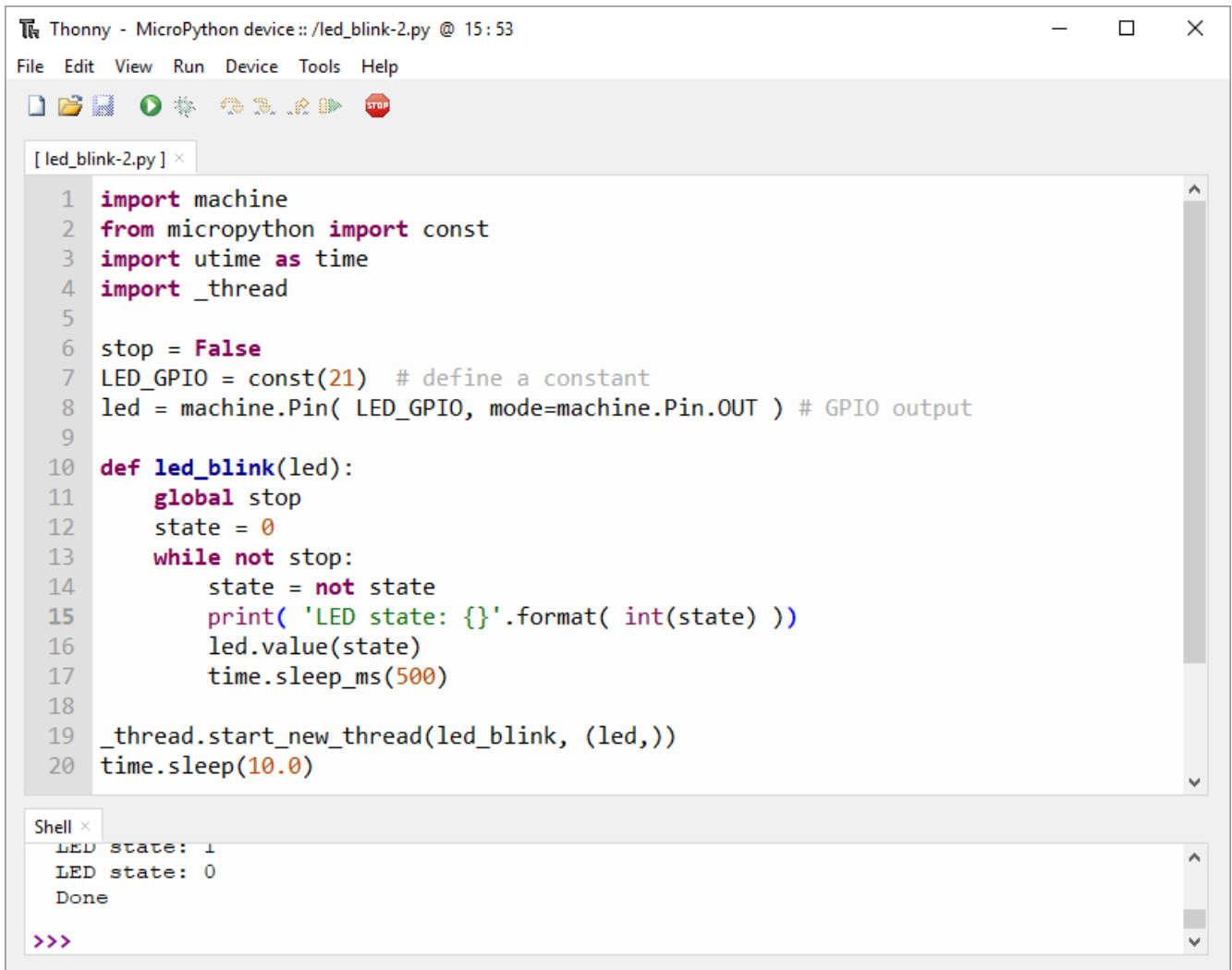
เมื่อเลือกบอร์ดไมโครคอนโทรลเลอร์แล้ว จะต้องเลือกไฟล์ **.bin / .hex** ที่เป็นเฟิร์มแวร์สำหรับไมโครไพธอน ให้ตรงกับบอร์ดที่เลือกใช้ (ให้ไปยังหน้าเว็บสำหรับดาวน์โหลดไฟล์ **MicroPython Firmware** เช่น สำหรับ **ESP32, ESP8266, STM32**) และจะต้องทำขั้นตอนติดตั้งลงในหน่วยความจำ **Flash** ของไมโครคอนโทรลเลอร์ (ซึ่งมีขั้นตอนและวิธีการที่แตกต่างกันไป ขึ้นอยู่กับฮาร์ดแวร์ที่นำมาใช้งาน) หรือเราอาจจะดาวน์โหลด **Source Code** มาจาก **Github** แล้วทำขั้นตอน **Build** เองก็ได้เช่นกัน

การเลือกใช้ Editor สำหรับ MicroPython

ตัวอย่างซอฟต์แวร์ประเภท **Open Source** ที่เราสามารถนำมาใช้เขียนโค้ดไมโครไพธอน และเชื่อมต่อกับบอร์ดไมโครคอนโทรลเลอร์ที่ได้ติดตั้งเฟิร์มแวร์แล้ว ได้แก่

- **Thonny IDE**
- **Mu Editor** (Windows, Mac OS X, Linux, Raspbian OS)
- **uPyCraft IDE**
- **VSCode + Extensions for MicroPython (e.g. PyMakr)**
- ...

ถ้าเปิดพอร์ต **Serial (Virtual COM port)** เชื่อมต่อกับพอร์ต **USB** ของบอร์ดไมโครคอนโทรลเลอร์ที่ได้ติดตั้งเฟิร์มแวร์สำหรับไมโครไพธอนแล้ว จะเชื่อมต่อกับส่วนที่เรียกว่า **REPL (เหมือน Python Interactive Shell)** รอให้ผู้พิมพ์คำสั่งสำหรับไมโครไพธอน



```
1 import machine
2 from micropython import const
3 import utime as time
4 import _thread
5
6 stop = False
7 LED_GPIO = const(21) # define a constant
8 led = machine.Pin( LED_GPIO, mode=machine.Pin.OUT ) # GPIO output
9
10 def led_blink(led):
11     global stop
12     state = 0
13     while not stop:
14         state = not state
15         print( 'LED state: {}'.format( int(state) ))
16         led.value(state)
17         time.sleep_ms(500)
18
19 _thread.start_new_thread(led_blink, (led,))
20 time.sleep(10.0)
```

```
Shell x
LED state: 1
LED state: 0
Done
>>>
```

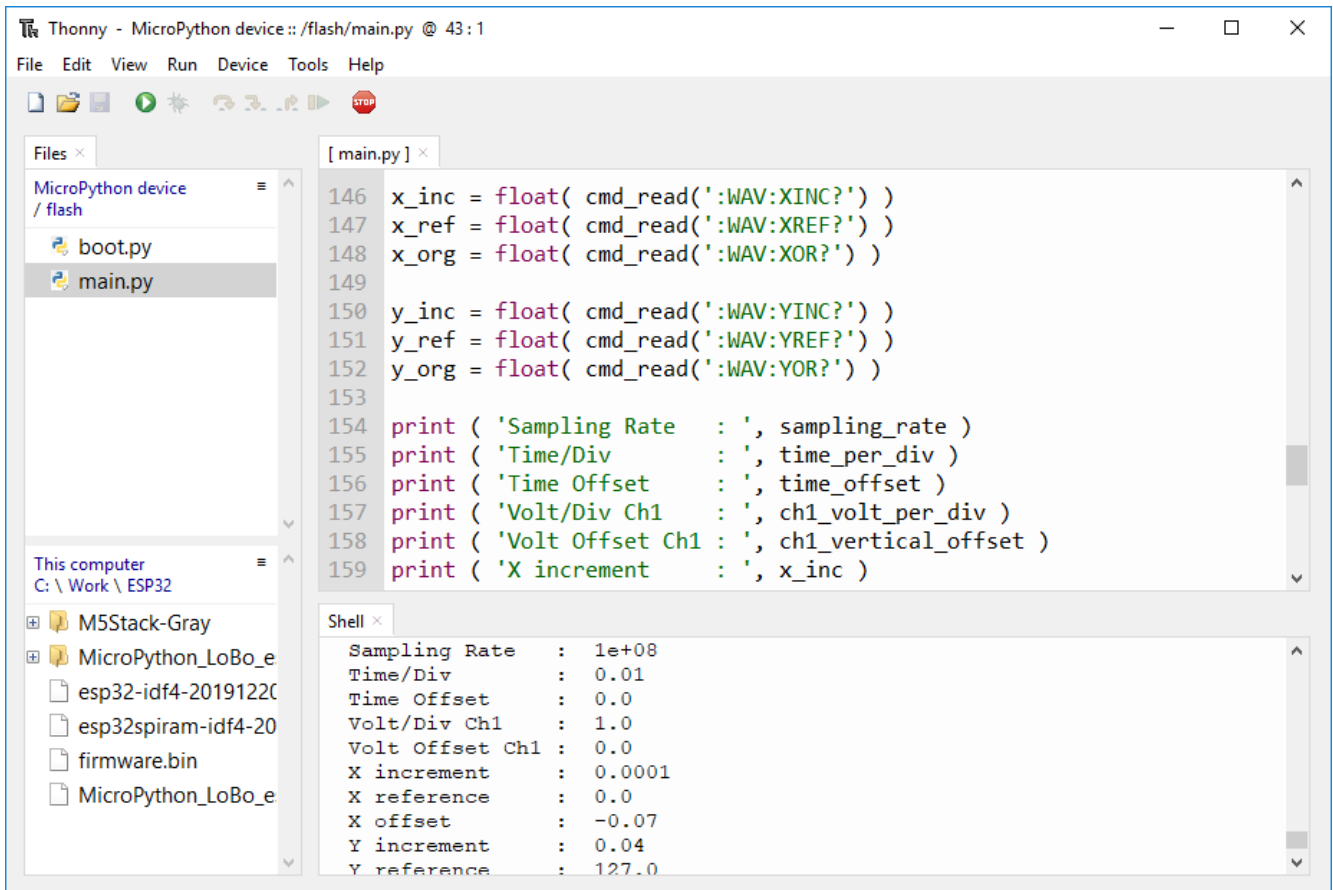
Thonny IDE + MicroPython for ESP32 Demo

หลักการทำงานของ **REPL (Read-Eval-Print Loop)** มีดังนี้

1. Read the user input (Python code)
2. Evaluate Python code
3. Print any results (e.g., output texts or error messages)
4. Loop back to step 1

Python IDE ที่รองรับการใช้งานสำหรับไมโครไพธอน นอกจากมีส่วนที่เชื่อมต่อกับ **REPL** ได้แล้ว ยังสามารถทำคำสั่ง เพื่อรันหรืออัปโหลดโค้ด **.py** และบันทึกเป็นไฟล์ลงใน **Flash Storage** ของ **MicroPython Device** ได้ด้วย

ถ้าจะใช้โปรแกรมแบบ **Command Line** ก็มีตัวเลือก เช่น **micropy-cli** ซึ่งใช้ภาษา **Python 3** ในการพัฒนา และสามารถใช้ร่วมกับ **VS Code** ได้ด้วย



```
Thonny - MicroPython device :: /flash/main.py @ 43 : 1
File Edit View Run Device Tools Help

Files x
MicroPython device
/ flash
boot.py
main.py

This computer
C:\Work\ESP32
M5Stack-Gray
MicroPython_LoBo_e
esp32-idf4-20191220
esp32spiram-idf4-20
firmware.bin
MicroPython_LoBo_e

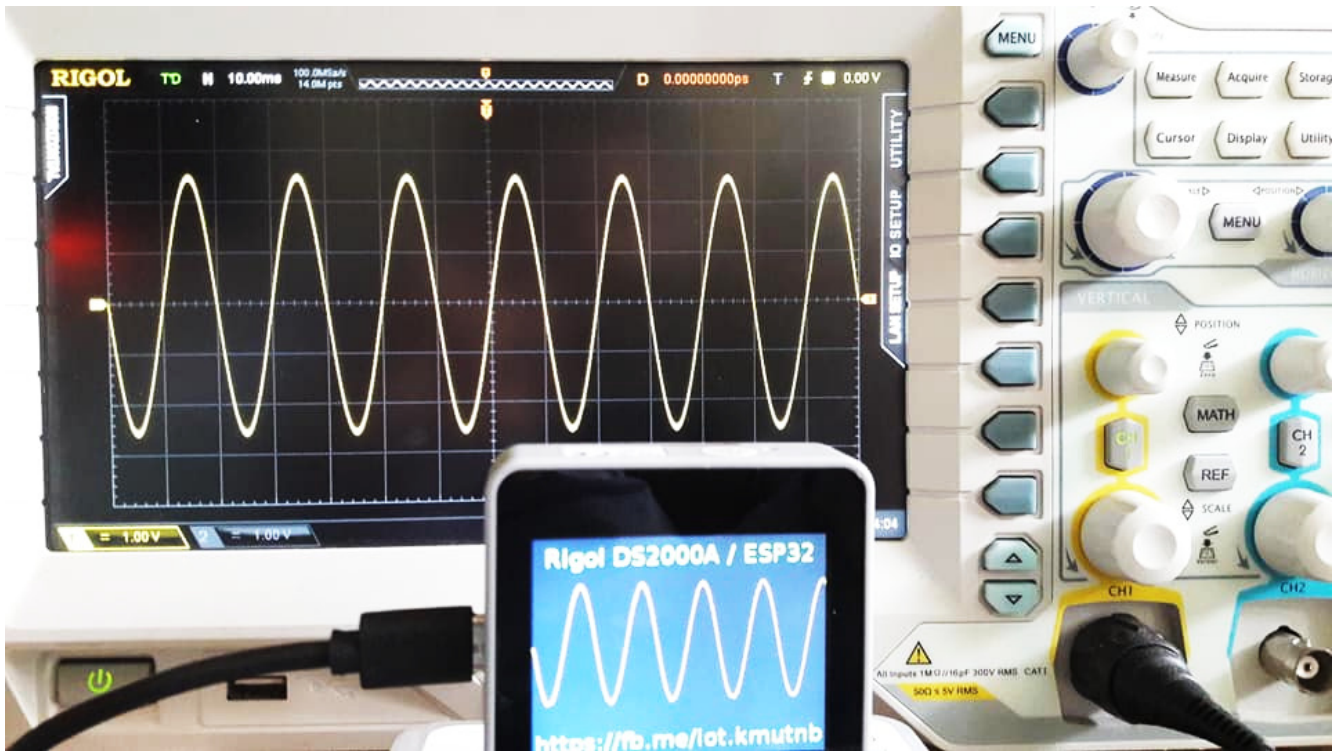
[ main.py ] x
146 x_inc = float( cmd_read(':WAV:XINC?') )
147 x_ref = float( cmd_read(':WAV:XREF?') )
148 x_org = float( cmd_read(':WAV:XOR?') )
149
150 y_inc = float( cmd_read(':WAV:YINC?') )
151 y_ref = float( cmd_read(':WAV:YREF?') )
152 y_org = float( cmd_read(':WAV:YOR?') )
153
154 print ( 'Sampling Rate : ', sampling_rate )
155 print ( 'Time/Div : ', time_per_div )
156 print ( 'Time Offset : ', time_offset )
157 print ( 'Volt/Div Ch1 : ', ch1_volt_per_div )
158 print ( 'Volt Offset Ch1 : ', ch1_vertical_offset )
159 print ( 'X increment : ', x_inc )

Shell x
Sampling Rate : 1e+08
Time/Div : 0.01
Time Offset : 0.0
Volt/Div Ch1 : 1.0
Volt Offset Ch1 : 0.0
X increment : 0.0001
X reference : 0.0
X offset : -0.07
Y increment : 0.04
Y reference : 127.0
```

MicroPython-ESP32 Demo for Interfacing M5Stack with Rigol Oscilloscope

ผู้เขียนได้เคยลองใช้งาน **MicroPython** ร่วมกับ **ESP32** (โมดูล **M5Stack Core**) เช่น การเขียนโปรแกรมเพื่อเชื่อมต่อกับอุปกรณ์เครื่องมือวัด **Rigol Digital Oscilloscope** รุ่น **DS2000A Series** ที่เชื่อมต่อผ่าน **LAN (Ethernet Port)** ในขณะที่ **ESP32** เชื่อมต่อผ่านทาง **Wi-Fi** ในระบบเครือข่ายเดียวกัน

ในกรณีตัวอย่างนี้ แสดงให้เห็นว่า เราสามารถเขียนโค้ดไมโครไพธอน เพื่อส่งคำสั่งไปยังอุปกรณ์เครื่องมือวัดที่อยู่ห่างออกไปในระบบเครือข่าย (เช่น ใช้คำสั่งเกี่ยวกับ **TCP Socket**) เพื่อตั้งค่าการใช้งานอุปกรณ์ และอ่านข้อมูลที่ได้จากการวัดสัญญาณ (เช่น หนึ่งช่องสัญญาณ) แล้วส่งกลับมายัง **ESP32** และแสดงผลเป็นรูปภาพสัญญาณ บนจอภาพ **LCD** ของโมดูล **M5Stack Core** เป็นต้น

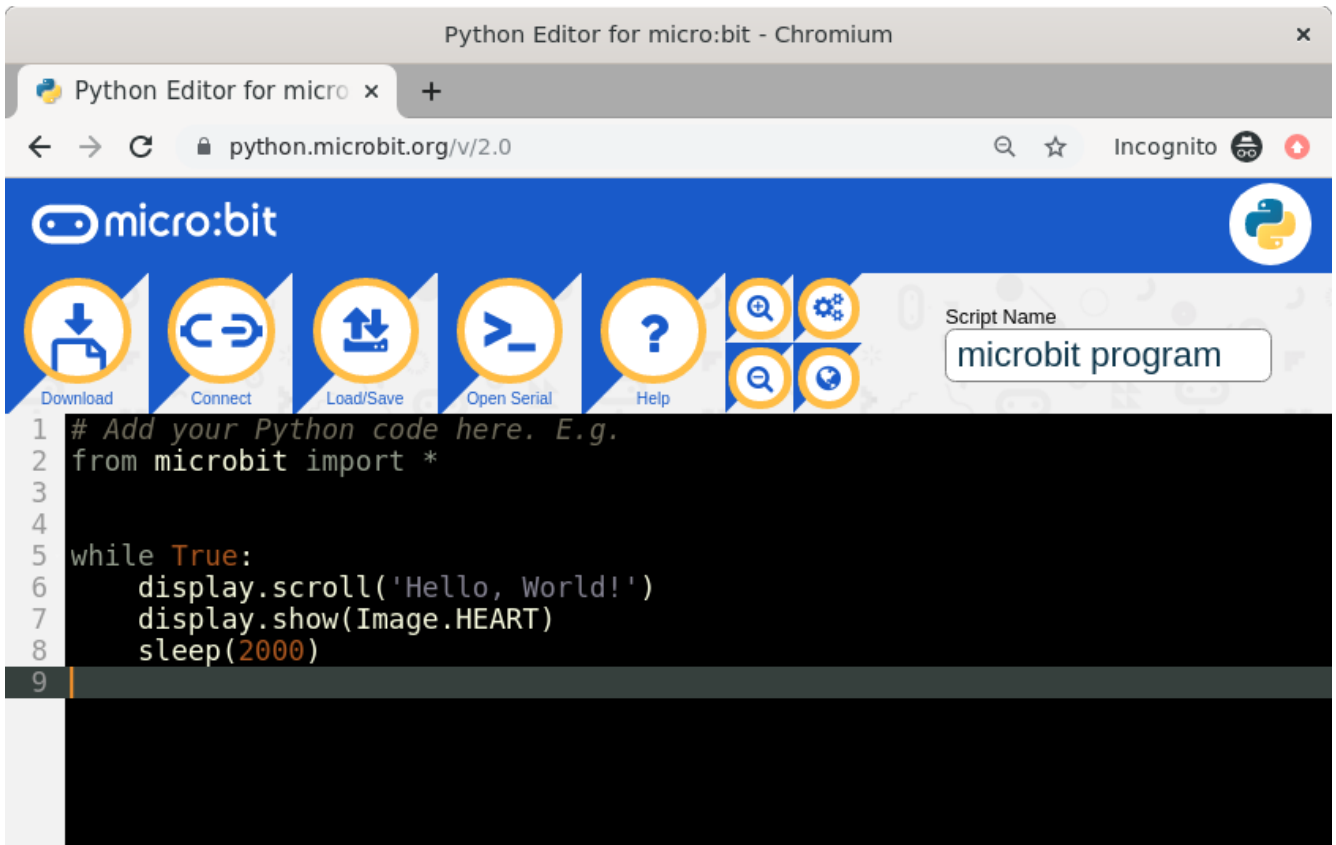


MicroPython-ESP32: Interfacing M5Stack with Rigol Oscilloscope

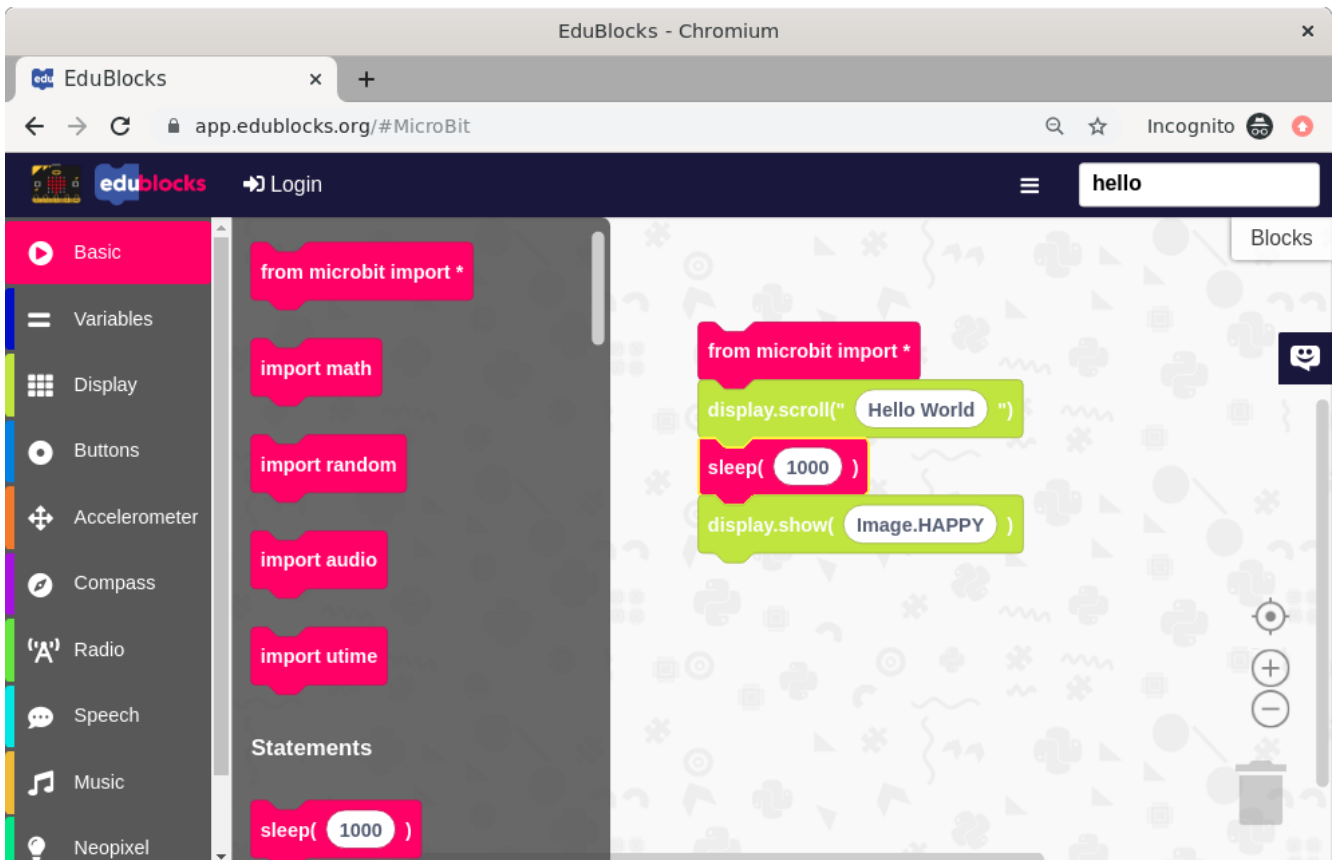
การเขียนโค้ดไมโครไพธอนสำหรับไมโครบิต

ในกรณีที่ใช้บอร์ดไมโครบิต ก็มี **Python Code Editor** แบบออนไลน์ ไว้ให้ใช้งานได้ฟรี และทำงานบนหน้าเว็บเบราว์เซอร์ ถ้าเชื่อมต่อกับบอร์ดผ่านทางพอร์ต **USB (WebUSB)** ก็สามารถรันโค้ดโดยใช้บอร์ดไมโครบิตได้จริง หรือจะใช้ **EduBlocks Editor** แบบออนไลน์ได้เช่นกัน

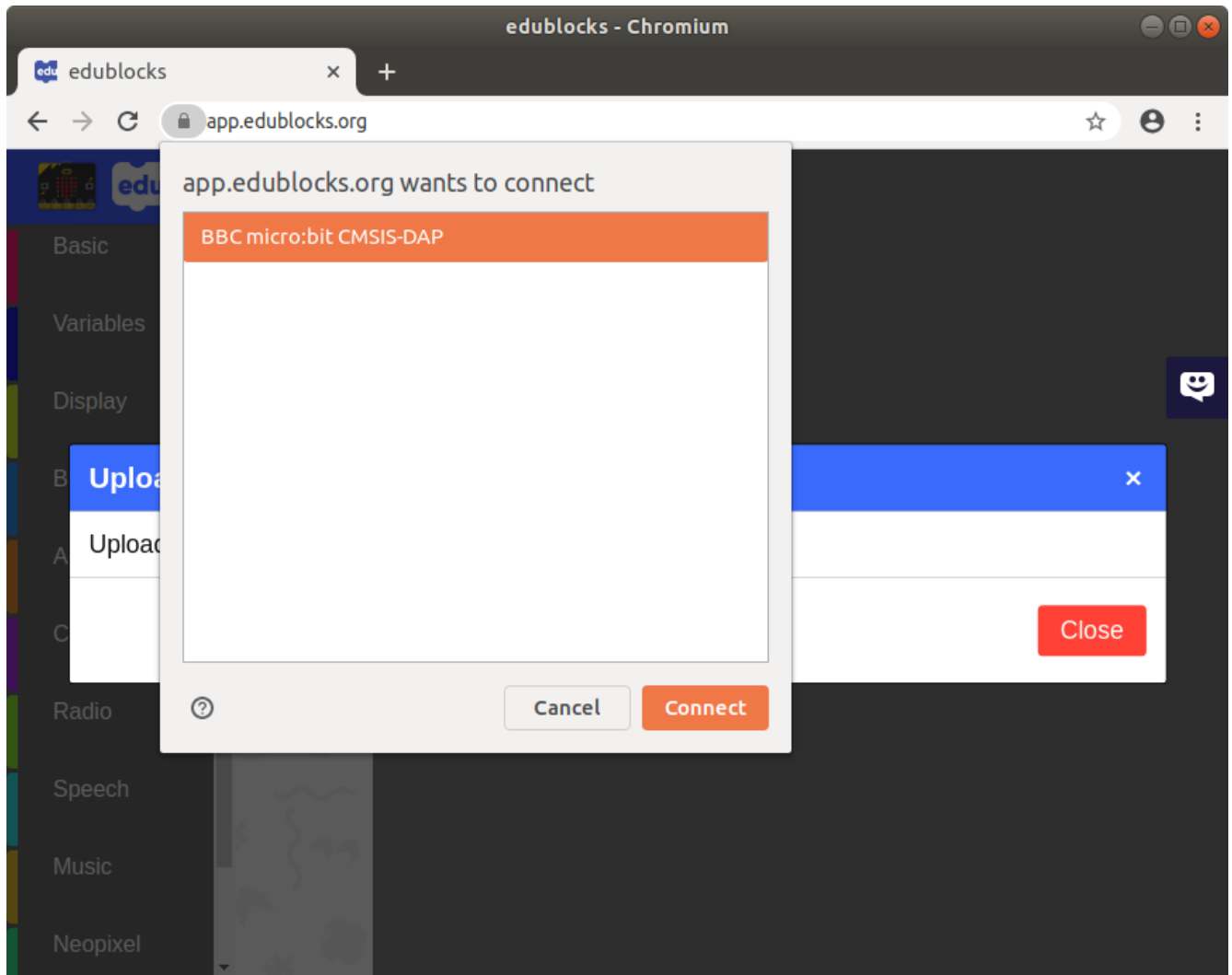
ผู้ที่สนใจสามารถศึกษาชุดคำสั่งหรือ **API** ได้จาก "**Micro:bit MicroPython Documentation**" แต่มีข้อสังเกตอยู่ว่า ไมโครไพธอนสำหรับแต่ละไมโครคอนโทรลเลอร์ที่เป็นเป้าหมาย (**Target Device**) อาจมีคำสั่งหรือ **API** ไม่เหมือนกันในรายละเอียด โดยเฉพาะอย่างยิ่งที่เกี่ยวข้องกับฮาร์ดแวร์ส่วนต่าง ๆ ทั้งภายในไมโครคอนโทรลเลอร์และภายนอก



Online Python Editor for Micro:bit



EduBlocks Editor for BBC Micro:bit (Blocks View)



EduBlocks Editor for BBC Micro:bit (Device Pairing)

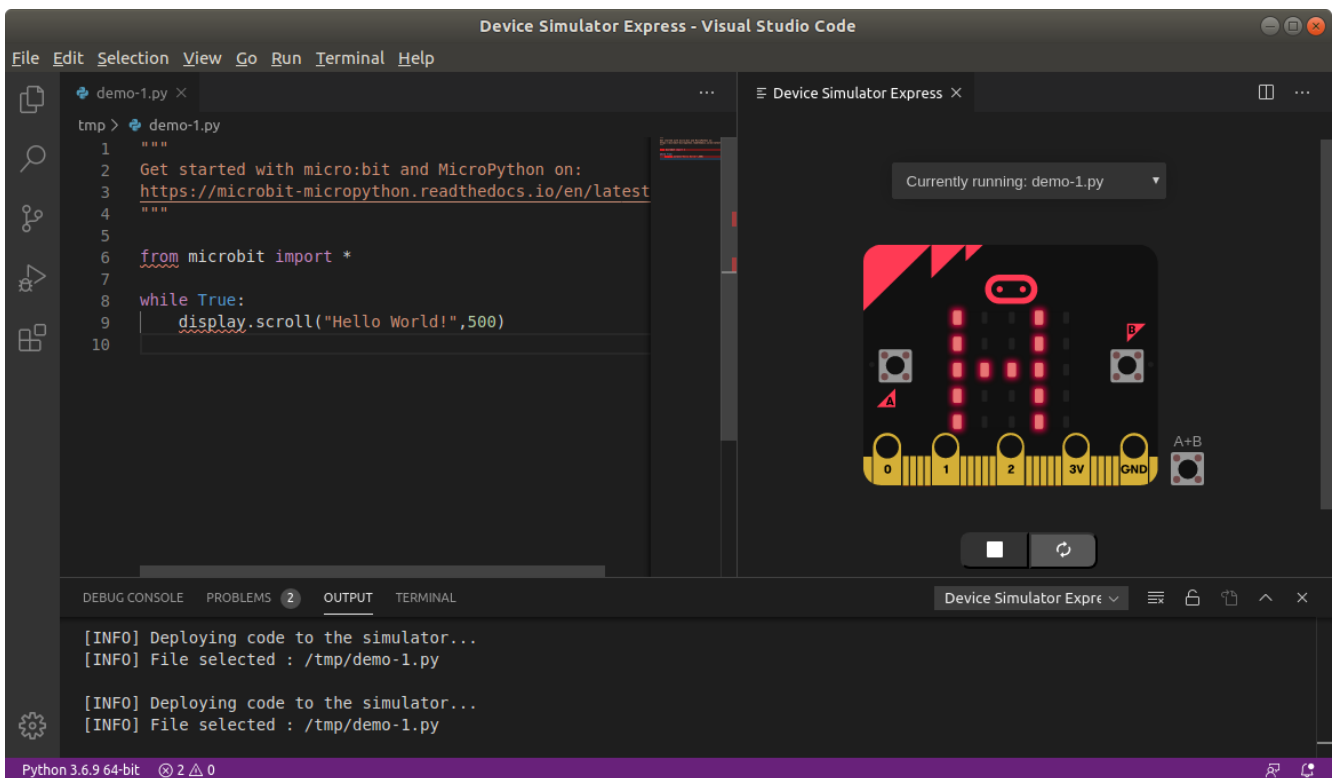
ในกรณีที่ใช้ **VS Code** เป็น IDE และถ้าได้ติดตั้ง **Extension** อย่างเช่น **Device Simulator Express (Source Code Resposity)** ก็สามารถจำลองการทำงานของโค้ดและตรวจสอบความถูกต้องในการทำงานสำหรับบอร์ดไมโครบิตได้

Device Simulator Express เกิดจากกิจกรรมของผู้เข้าร่วม **Microsoft Garage project** ในช่วงปีค.ศ. 2019 และช่วยในการเขียนโค้ด **CircuitPython** สามารถจำลองการทำงานของโค้ดเสมือนจริงได้ แม้ว่าอาจมีข้อจำกัดอยู่บ้าง รองรับบอร์ด **Adafruit Circuit Express** และ **Adafruit CLUE** แต่ก็สามารถใช้กับ **BBC Micro:bit** ได้เช่นกัน สามารถอัปโหลดโค้ดไปยังบอร์ดทดลองได้ด้วย



Installation of Device Simulator Express Extension in VS Code IDE

จุดเด่นของ **Device Simulator Express** คือ ความสามารถในการรันโค้ดไมโครไพธอนโดยใช้ **Built-in Simulator** ได้ ดังนั้นจึงทดสอบโค้ดได้ โดยยังไม่จำเป็นต้องมีบอร์ดไมโครบิต



MicroPython Simulator for BBC Micro:bit

โดยสรุป

การเรียนรู้และใช้งานไมโครไพธอนสำหรับไมโครคอนโทรลเลอร์ เป็นตัวเลือกที่น่าสนใจ อาจช่วยให้ง่ายขึ้นในการเริ่มต้นการเขียนโปรแกรมไมโครคอนโทรลเลอร์สำหรับผู้สนใจ โดยใช้ภาษา **Python** แทนการใช้ภาษา **C/C++** และช่วยให้การพัฒนาระบบสมองกลฝังตัว ทำได้เร็วและง่ายขึ้น

อัปเดต: วันที่ 28 กันยายน 2563