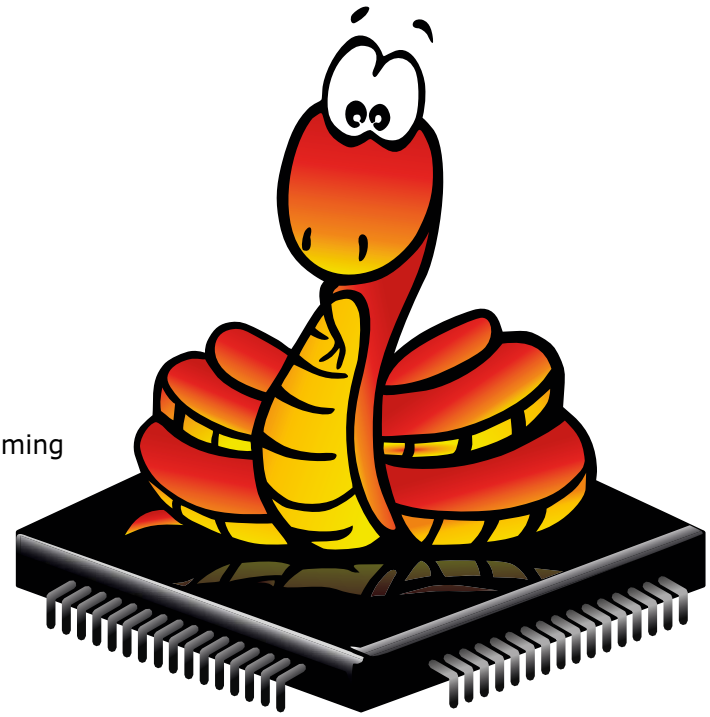
MicroPython

Python for small systems

By **Dogan Ibrahim** (UK)



MicroPython is a highly efficient and powerful programming language derived from Python, and having inherited a small collection of libraries. MicroPython fits in a mere 256 KB of code space and 16 KB of RAM, permitting it to be used on microcontrollers and other embedded systems with limited resources.

Python is used in many universities and technical colleges around the world as the initial programming language for students. The large collection of powerful libraries and the ease of use of Python make it an ideal language for everyone new to programming. MicroPython, originally targeted at 32-bit ARM microcontrollers, is compatible with Python and, thanks to its small size, makes an excellent choice for use on embedded processors. With MicroPython complex and manageable code can be developed to control embedded systems that would otherwise require languages such as C or C++. MicroPython allows experienced as well as novice Python users to program small embedded systems.

The main reasons for MicroPython to not fully support Python 3 are the lack of sufficient memory and missing hardware and software features (such as multitasking and multiprocessing) of embedded processors. The differences between Python 3 and MicroPython can be found at [1].

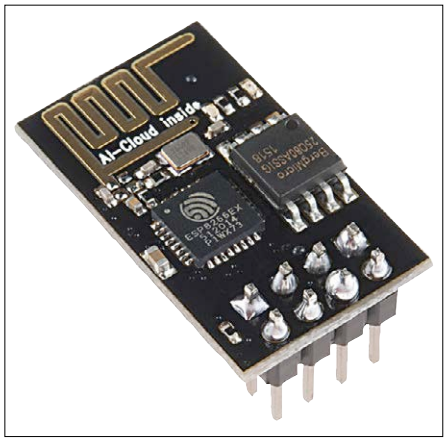


Figure 1. The ESP01, a highly practical and popular ESP8266-based module.

Figure 2. The BBC micro:bit can be programed online in MicroPython.

Why MicroPython?

MicroPython and Python offer some unique features compared to other programming languages. In a nutshell:

- (Micro)Python is interactive meaning the program is not compiled and uploaded into the target processor, but rather interpreted and acted upon at runtime. Although this makes programs run somewhat slower, it has the advantage of the user being able to easily experiment with his/her code. For example, we can simply do interactive calculations as if we are using a calculator, or experiment with parts of the program until we get the desired results. This feature, also known as Read-Evaluate-Print-Loop or REPL, is not available in compiled languages such as C or C++.
- In addition to the large number of built-in functions, an extensive set of function libraries is available that can be included in (or imported into) a program. For example, there are libraries for random number generation, trigonometry, making music, networking, string & file processing, graphics & gaming, and much more.
- MicroPython can be mixed with other programming languages such as C or C++. This gives additional power and flexibility since parts of the code that require high speed can be developed using languages better suited for such tasks.
- Exceptions and error handling are supported, which is especially important in real-time programming. Without proper error handling a crashed program may stop the processor in an unknown state which can have highly undesirable effects.
- It is open source meaning that the latest release can be downloaded [2] and run at no cost. The source code of MicroPython can be modified and ported to specific processors of interest.
- Finally, the language is human-readable and its syntax is easy to learn and understand.

What MicroPython can and can't do

MicroPython can, in general, do every-

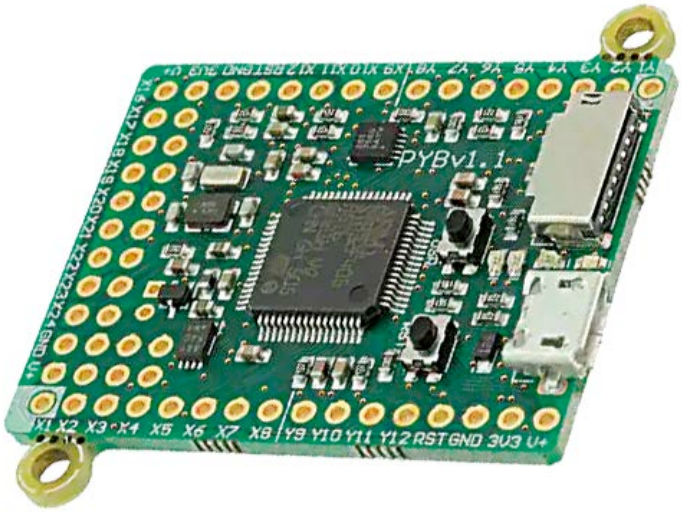


Figure 3. So you know, the pyboard is the official MicroPython microcontroller board.

Figure 4. The WiPy is similar to the ESP8266-based NodeMCU board, but with a CC3200 ARM Cortex M4 instead.

thing other programming languages can do too, like controlling hardware devices such as LEDs & displays, switches & buttons, sensors, motors and so on. Communication busses such as RS-232, CAN, I²C, SPI, and others can easily be used thanks to built-in and external libraries. Network and Wi-Fi-based programs can be written to communicate with other devices on a network, or to develop IoT systems.

Because MicroPython is an interpreted language it is slower compared to other embedded programming languages, consequently it is not a good choice for fast digital signal processing or real-time applications where high execution speeds are critical. In addition, although less important nowadays, MicroPython uses slightly more memory than most other embedded languages. Since MicroPython is a subset of Python that does not support all the Python libraries, a program developed in Python may not work on an embedded system running MicroPython.

Boards supported by MicroPython

The number of development boards supported by MicroPython is increasing along with its popularity. Let's list a few.

ESP8266

Boards fitted with this popular Wi-Fi-capable MCU (**Figure 1**) with its built-in TCP/IP stack and USB interface can be programed in MicroPython. The MCU is based on a 32-bit RISC CPU and MicroPython offers support for GPIO, SPI, I²C, UART, ADC and I²S. This may well be one of the cheapest ways to start out with if you wish to experiment with MicroPython.

BBC micro:bit

This credit card sized board (**Figure 2**) with its many built-in features such as 25 LEDs, two pushbuttons, an accelerometer, a compass, GPIO, I²C, UART, and ADC is supported by MicroPython. Programs can be created online which removes the need to do any setup or configuration. The BBC micro:bit is highly recommended for people new to programming.

pyboard

Based on the STM32F series Cortex M4 processor this development board (**Figure 3**) comes preloaded with MicroPython. The pyboard can be connected to a PC through its USB port. The board features a real-time clock, an accelerometer, GPIO, an ADC, four LEDs, and a microSD card slot.

WiPy

Like the pyboard this board (**Figure 4**) too has built-in MicroPython support. Its pins are suitable for plugging it on a breadboard. The board is based on the CC3200 Cortex M4 processor running at 80 MHz and includes UART, SPI, I²S, ADCs, Wi-Fi, GPIO, timers, and hash and encryption engines.

Some other embedded development boards of interest that come with MicroPython support include: Teensy 3.x, SAMD21, LoPy, STM32F4-Discovery, Raspberry Pi and BeagleBone (both run the full Python 3 code).

MicroPython program example

Here is a simple MicroPython program that runs on the BBC micro:bit. It's a thermostat where the CPU temperature is read continuously and an appropriate message is displayed on the on-board LED matrix. When the temperature is equal to or higher than 25 ºC the message 'HIGH' is displayed. If the temperature is between 20 ºC and 25 ºC 'MEDIUM' is shown. Otherwise, 'LOW' is displayed.

```
#Simple CPU thermostat program
from microbit import *

while True:
  temp = temperature()
  If temp >= 25:
    display.scroll("HIGH")
  elif temp >= 20 and temp < 25:
    display.scroll("MEDIUM")
  else:
    display.scroll("LOW")
```

(160315)

Web Links

[1] https://github.com/micropython/micropython/wiki/Differences
[2] https://github.com/micropython/
[3] http://micropython.org/