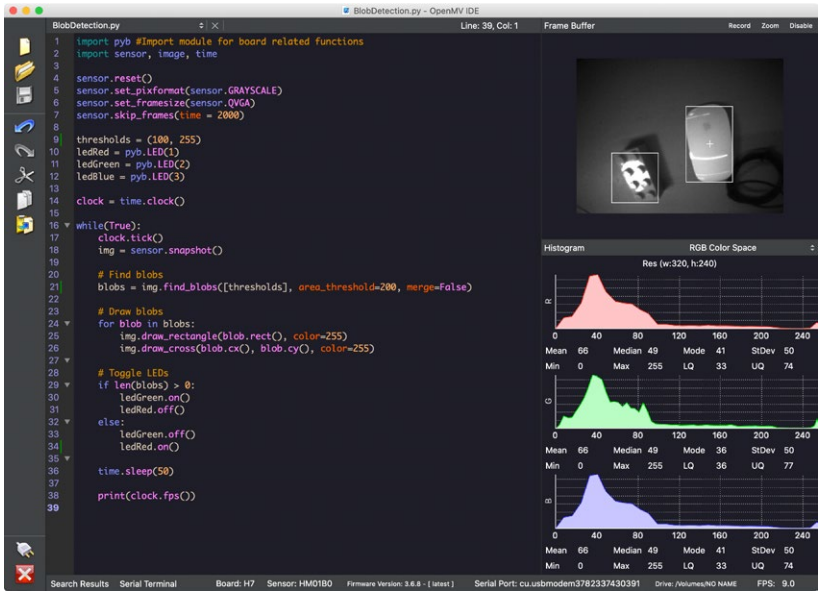
MicroPython Enters the World of Arduino

By Stuart Cording (Elektor)

MicroPython has made it to the world of Arduino, providing the first significant alternative to programming in C and C++. So, what's all the fuss, how easy is it to use, and who can benefit from programming in this, for microcontrollers, relatively new language? Elektor spoke to Sebastian Romero (Head of Content, Arduino) to find out more.

C and C++ have been the staple of Arduino software development since Arduino's inception in the early 2000s. Thanks to a predefined program structure with a setup() and loop() function, beginners to the world of embedded software development have been guided through setting up their board and executing their application in a loop. Now there is a new language available. MicroPython is a lightly stripped-back version of Python, an interpreted, general-purpose programming language that targets microcontrollers. The question is, why use an interpreted language on hardware used for real-time applications?



Blob detection running on Arduino Portenta H7 in OpenMV.

About Sebastian Romero (Head of Content @Arduino)

Sebastian Romero, head of content at Arduino, is an interaction designer, educator and creative technologist with a weakness for humans. With his team, he is responsible for crafting enthralling learning experiences to help millions of engineers, designers, artists, hobbyists and students to innovate.

"Because MicroPython's simplicity makes it well suited for beginners, educators were one of the first to ask us about it," explains Sebastian Romero, Head of Content at Arduino.

C makes interaction with microcontroller registers easier than with assembler, and object-oriented programming (OOP) in C++ makes for more concise code with fewer mistakes. However, parsing strings is challenging, and there is no native support in the language for handling today's web data formats, such as HTTP, JSON[1], or RegEx[2] (regular expression). With today's education revolving around interaction with the Internet and web services, C/C++ has been sidelined in favor of languages such as Python, which make coding such applications more straightforward.

"As a result, if you're a tutor teaching Python, you prefer to stick with Python when the topic of microcontrollers comes up," says Sebastian.

Of course, it's not just educators. Makers have had a range of MicroPython-capable boards on offer from

other sources, such as the ESP32, Raspberry Pi Pico, and pyboard, and the industry is increasingly looking to MicroPython as well. The rapid growth in machine learning (ML) is, in part, thanks to the existence of libraries available for Python. With teams of engineers competent in Python, few want to switch to C/C++ when they transfer their ML model and application to a microcontroller, preferring to stick with one development stack. The other issue is the workforce – it is increasingly challenging to find C/C++ programmers, while academia is churning out plenty of Python-competent engineers.

MicroPython Vs. Python: What's the Difference?

Python started its life back in the late 1980s, designed by Guido van Rossum[3]. Designed to be fun to use, it also aims to be explicit rather than implicit, simple, and result in readable code. In 2013, Damien George successfully launched a Kickstarter [4] campaign to deliver a version designed from the ground up for microcontrollers along with the pyboard hardware to run it. Micro Python, as it was named at the time, promised a scripting language that would "allow you to effortlessly blink LEDs, read voltages," and more. USB-enabled microcontrollers would appear as a USB flash drive onto which code could be uploaded. Alternatively, the device could appear as a serial device, offering a command line known as REPL (read, evaluate, print, loop).

Unlike C/C++ sketches which must be compiled and downloaded, MicroPython can be executed immediately after every change.

As might be expected, MicroPython requires a reasonable amount of memory to run in order to support uploaded code and its interpretation during execution. While 128 KB flash and 8 KB SRAM is enough, the feature set would be so limited as to make for a poor experience. Hence, most MicroPython boards settle for a microcontroller with at least 256 KB flash and 16 KB SRAM. This also has the side effect of selecting a device with a relatively powerful processor operating at around 50 MHz or more that offers a respectable range of peripherals.

"It constantly surprises me how much you can achieve with just 16 KB of SRAM," shares Sebastian - the quote is from Jim Mussared, an Embedded Engineer at micropython.org. "Most students, at least with introductory-level projects, don't need lots of heap. Their projects typically grow in complexity due to more code."

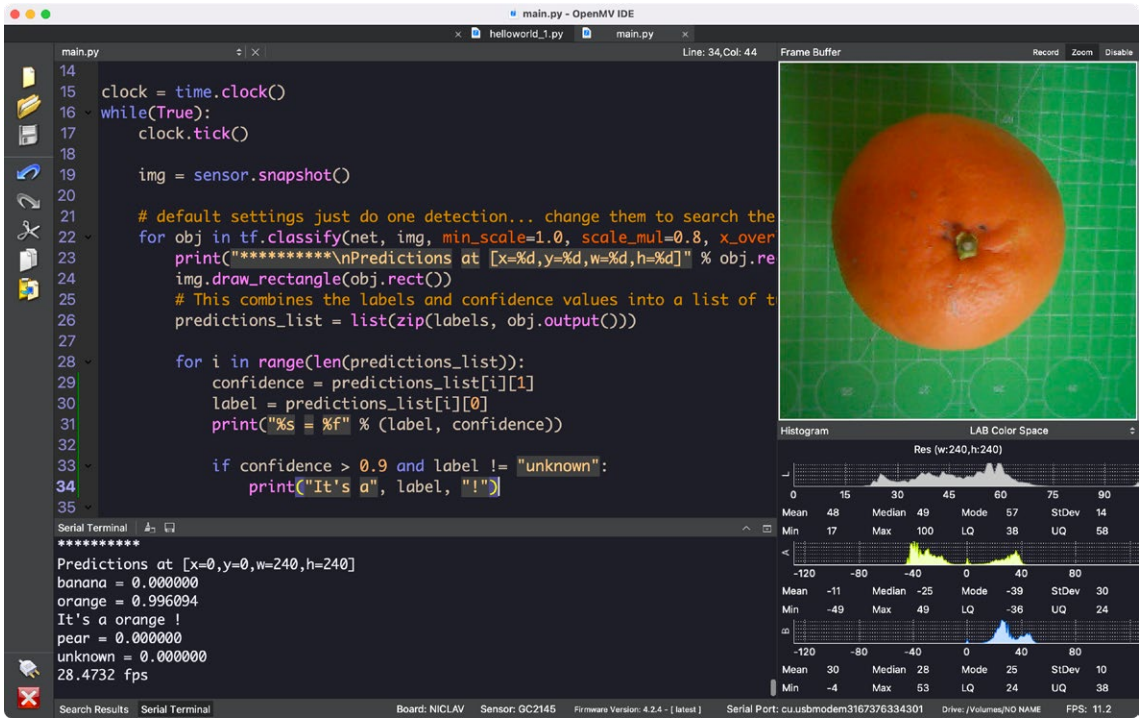
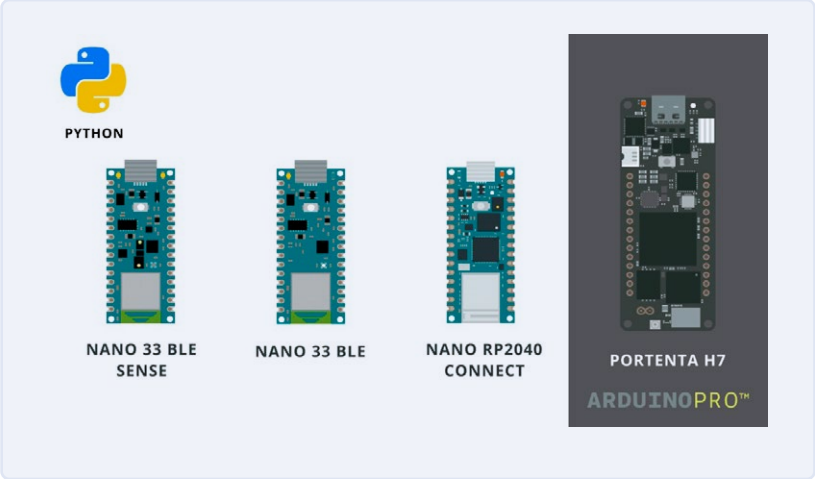


Image classification running on Arduino Nicla Vision in OpenMV.



Such MicroPython projects mainly implement state machines and evaluate sensor data.

But, the SRAM is not only used to store variables. It also stores compiled bytecode, such as imported modules, for execution by the MicroPython virtual machine (VM). That can cause issues when handling large data, such as strings, or creating and destroying many objects that leave insufficient SRAM to run the compiler. However, once the code has reached a mature state, bytecode can be precompiled and stored in flash (in the filesystem) or implemented as frozen bytecode to save even more SRAM.[5]

MicroPython is also fussier than Python when inputting code, demanding the correct use of spacing. Users also quickly learn that some default features, such as a full implementation of the standard library, are unavailable due to limited hardware capabilities.

Development Environment for Arduino MicroPython

The Arduino team selected the version of MicroPython maintained by OpenMV, as Arduino's new camera-equipped devices benefit from the Machine Vision features and the built-in support for Tensor Flow Lite provided by OpenMV [6]. As a result, users have a well-supported, mature platform and development environment. OpenMV was created to support machine vision applications on microcontrollers, which aligns well with many users' desire to create image based ML applications.

OpenMV IDE (Integrated Development Environment) provides the working area for MicroPython coders rather than the traditional Arduino IDE. It offers the core features that developers expect, such as a code development window and a serial terminal. On top, there is support for machine vision applications, such as visualization of a frame buffer and a histogram tool to visually analyse color and brightness ranges. Furthermore it has a built-in tool to upload camera pictures directly to Edge Impulse Studio to easily train a machine learning model.

But, perhaps the greatest change lies with how code is developed and deployed.

"Unlike C/C++ sketches that must be compiled and uploaded, MicroPython can be executed immediately after every change. That speeds up development significantly and brings the coding experience closer to that of Python," said Sebastian.

Another great feature is REPL, enabling short scripts to be executed or individual functions to be tested directly on the target controller.

Arduino Hardware Support for MicroPython

In total, five Arduino boards currently support MicroPython: the Nano 33 BLE and Nano 33 BLE Sense, the Nano RP2040 Connect, the Portenta H7 and Nicla Vision. Most of the boards require a firmware update to upload the MicroPython runtime into flash before getting started. As we've come to expect, not only is this process simple, but it is also well-documented[7]. Boards such as the Nano 33 have a preparation step that uses the Arduino IDE, while the others are immediately recognized by OpenMV and programmed with the necessary firmware.

The application is written as a Python script in OpenMV that is uploaded to the target board. A single click on the Play button is all that stands between the programmer and code execution.

What's Next?

What is the future for Arduino now that MicroPython is here? It is natural to worry that, with the introduction of MicroPython, the traditional C/C++ sketch may become a historical artifact. But this is neither desired nor the plan. In scenarios where real time execution is a requirement, C/C++ will still be the go to weapon.

"We are seeing growth in demand for Python support on microcontrollers, especially in industry pioneers working to develop ML applications who already use a Python stack," says Sebastian.

In fact, MicroPython support extends the available options for Arduino users rather than replacing them. Long term, C/C++ sketch developers should find suitable boards in the same form factor that they're using, that also support MicroPython.

„For many of the classic Arduino boards, a MicroPython implementation would have a very limited feature set and hence isn't a reasonable option," Sebastian adds.

Users can also continue contributing to the success of Arduino [8] with MicroPython as they have done in the past. There are 15 years of contributed C/C++ code that is still being maintained and used as drivers in combination with MicroPython. Bindings are then used to link MicroPython with this base code. For those wishing to get involved with MicroPython [10] development, OpenMV is hosted on GitHub [9], a project to which the Arduino team also contributes.

MicroPython can be seen as an addition to the current Arduino ecosystem, with its adoption driven by the success of Python as the language of preference for ML applications and interacting with cloud services. With microcontrollers increasingly hitting hundreds of megahertz and offering heaps of memory, the move to an interpreted language will be seen as an irrelevance in many cases. Of course, there are exceptions where real-time accuracy and precision are a must, and C/C++ will always be there for those who require it. For now, though, educators and students benefit significantly, allowing knowledge of Python to be transferred to microcontrollers (additionally they benefit from the simplified syntax and readability of the code). At the same time, industry developers can stick with a single language for application development.

220415-01

About the Author

Stuart Cording is an engineer and journalist with more than 25 years of experience in the electronics industry. You can find many of his recent Elektor articles at www.elektormagazine.com/cording. In addition to writing for Elektor, he hosts the monthly livestream, *Elektor Engineering Insights* (www.elektormagazine.com/eei), and he teaches Elektor Academy courses (www.elektormagazine.com/elektor-academy).

Questions or Comments?

If you have technical questions, feel free to e-mail the author at stuart.cording@elektor.com or the Elektor editorial team at editor@elektor.com.

WEB LINKS

[1] JSON, Wikipedia: https://en.wikipedia.org/wiki/JSON
[2] Regular Expression, Wikipedia: https://en.wikipedia.org/wiki/Regular_expression
[3] Python, Wikipedia: https://en.wikipedia.org/wiki/Python_(programming_language)
[4] D. George, "Micro Python: Python for microcontrollers," Kickstarter, 2016: https://www.kickstarter.com/projects/214379695/micro-python-python-for-microcontrollers
[5] "MicroPython on Microcontrollers": https://docs.micropython.org/en/latest/reference/constrained.html
[6] OpenMV: https://openmv.io/
[7] K. Soderby, "Python with Arduino Boards," Arduino, 2022: https://docs.arduino.cc/learn/programming/arduino-and-python
[8] Arduino, GitHub: https://github.com/arduino
[9] OpenMV, GitHub: https://github.com/openmv
[10] Official uPython Repo: https://github.com/micropython/micropython