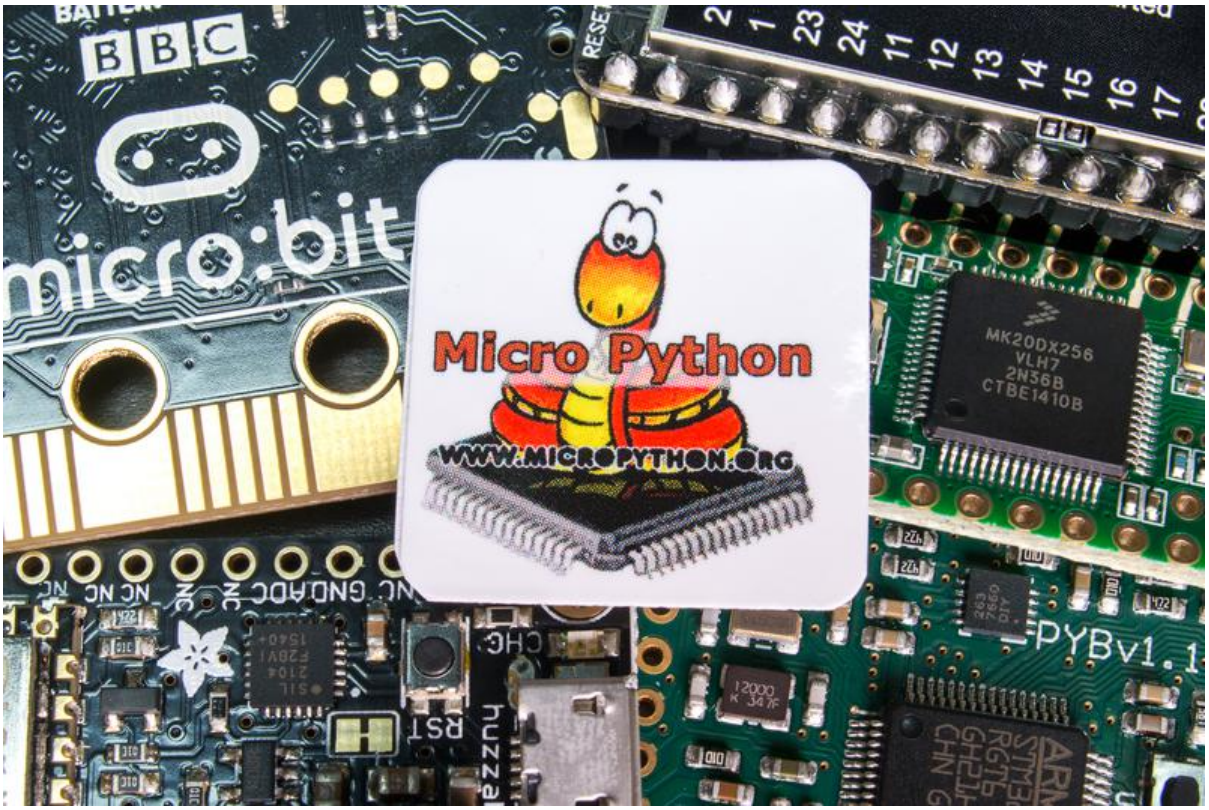
# MicroPython Basics: What is MicroPython?

Created by Tony DiCola




https://learn.adafruit.com/micropython-basics-what-is-micropython

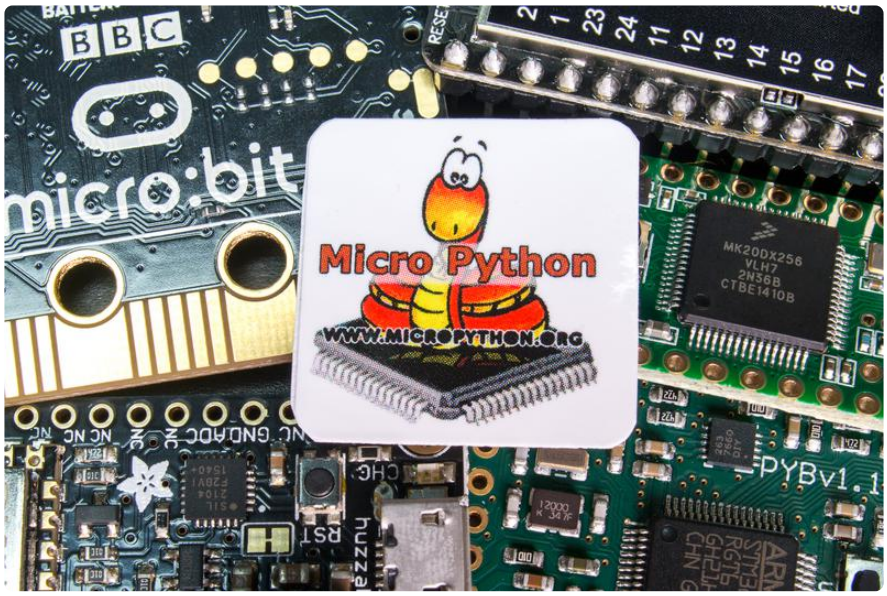
Last updated on 2023-08-29 03:12:17 PM EDT

# Table of Contents

Overview 3

# Overview

The information in this guide is no longer supported and may not work. We are only supporting CircuitPython on our boards. For more information about using CircuitPython, check out Welcome to CircuitPython: https://learn.adafruit.com/welcome-to-circuitpython




This guide explains what is the MicroPython programming language (), why you might want to use it for hardware projects, and where to find more information on MicroPython.  If you're familiar with Arduino () you'll learn how MicroPython compares to it and why you might want to explore using MicroPython in place of Arduino.  Don't worry though if you're totally new to hardware and programming MicroPython is a great place to start learning!

In addition this guide also explores CircuitPython () which is Adafruit's open source derivative of MicroPython with a focus on being simple for beginners to get started with electronics.  Read below to learn more about the differences between CircuitPython and MicroPython.  Both use the same Python programming language and have similar features--almost anything you can do in MicroPython can be done in CircuitPython (and more!).

## What is MicroPython?

MicroPython () is a tiny open source Python programming language () interpretor that runs on small embedded development boards.  With MicroPython you can write clean and simple Python code to control hardware instead of having to use complex low-level languages like C or C++ (what Arduino uses for programming).

The simplicity of the Python programming language makes MicroPython an excellent choice for beginners who are new to programming and hardware. However MicroPython is also quite full-featured and supports most of Python's syntax so even seasoned Python veterans will find MicroPython familiar and fun to use.

Beyond its ease of use MicroPython has some unique features that set it apart from other embedded systems:

- Interactive REPL, or read-evaluate-print loop. This allows you to connect to a board and have it execute code without any need for compiling or uploading--perfect for quickly learning and experimenting with hardware!
- Extensive software library. Like the normal Python programming langauge MicroPython is 'batteries included' and has libraries built in to support many tasks. For example parsing JSON data from a web service, searching text with a regular expression, or even doing network socket programming is easy with built-in libraries for MicroPython.
- Extensibility. For advanced users MicroPython is extensible with low-level C/C++ functions so you can mix expressive high-level MicroPython code with faster low-level code when you need it.

---

## What is CircuitPython? How does it relate to MicroPython?

CircuitPython () is Adafruit's open source derivative of MicroPython. This version of MicroPython was created to add support for easily getting started with electronics using boards like Circuit Playground Express, Trinket M0, Gemma M0, and more. Check out the CircuitPython documentation () for more details about it, including a list of differences between CircuitPython and MicroPython ().

In almost all cases you can do the same things with both CircuitPython and MicroPython. CircuitPython has slightly different and simpler APIs for accessing some hardware components like digital I/O, I2C, SPI, etc. However the core Python language support is the same between CircuitPython and MicroPython. Unless a guide or project notes otherwise you can use the same code between CircuitPython and MicroPython and vice-versa.

---

## What can MicroPython do?

Almost anything you can imagine! Just like an Arduino board MicroPython can control hardware and connected devices. You can control GPIO pins to blink lights, read switches, and more. You can drive PWM outputs for servos, LEDs, etc. or read analog sensors with an analog to digital converter. Talking to I2C or SPI devices is

easy too, and you'll even find network & WiFi support on some boards. MicroPython even has libraries for controling other hardware like NeoPixels and LED strips, tiny OLED displays, and more.

In short, MicroPython can do a lot!

For some examples of what MicroPython can do check out the following:

- Video overview of MicroPython Pyboard ()
- Video overview of MicroPython on ESP8266 ()
- Kickstarter campaign video for MicroPython on ESP8266 ()
- Video demo of dice game in MicroPython on the bbc:microbit ()
- Play with a live MicroPython Pyboard over the internet ()

## What can't MicroPython do?

There are very few limitations with MicroPython, almost anything an Arduino can do can also be done by a MicroPython board. However one thing to realize is that MicroPython code isn't as fast and might use a little more memory compared to similar Arduino or other low-level C/C++-based code. Usually this doesn't matter since the speed and memory differences are small and don't impact most normal uses.

However be aware that code which has tight timing or performance requirements might not work in MicroPython. For example 'bit banging' a fast serial protocol entirely in MicroPython might not be the best idea. However there are ways to mix both MicroPython and low-level C/C++ code so you can have the best of both worlds--your main logic in clean and easy to understand MicroPython code, and performance critical parts written in faster low-level code.

The MicroPython langauge implements most of the core Python 3 language, however MicroPython can't implement the entire Python 3 standard library. Python is known for having an extensive standard library, but trying to squeeze such a big library onto tiny boards with just kilobytes of memory isn't possible. MicroPython instead implements smallers versions of some Python standard libraries to give you a great development experience.

## How does MicroPython/CircuitPython compare to Arduino?

There are a couple important differences between Arduino and MicroPython. The first is that Arduino is an entire 'ecosystem' with the Arduino IDE (i.e. the desktop

application you use to write and upload sketches), the Arduino programming language (based on C/C++), and Arduino hardware like the Arduino Uno R3 board.

MicroPython is only a programming language interpreter and does not include an editor. Some MicroPython boards support a web-based code prompt/editor, but with most MicroPython boards you'll write code in your desired text editor and then use small tools to upload and run the code on a board.

If you're coming to MicroPython or CircuitPython from a language that has a full IDE/editing environment like Arduino you might want some tips on how to configure and use a text editor and other tools for MicroPython/CircuitPython development. See these videos below for a deep dive into setting and using MicroPython/CircuitPython on different platforms:

- CircuitPython on ChromeOS ()
- CircuitPython on Windows ()
- CircuitPython on macOS ()

The second important difference is that the MicroPython language is interpreted instead of being compiled into code the CPU can run directly like with the Arduino programming language. Interpreted means when MicroPython code runs it has to do a little more work to convert from MicroPython code to instructions the CPU understands.

A major advantage of interpreted code is that it can be much cleaner and simpler compared to languages that compile directly to CPU instructions. You can even write and run interpreted code like MicroPython directly on a board without any compiling or uploading--something that's impossible with Arduino!

One disadvantage of interpreted code and MicroPython vs. Arduino is that there's less performance and sometimes more memory usage when interpreting code. A function or sketch written in Arduino will run as fast as possible on a board's CPU whereas similar code in MicroPython will be a little slower because it has to interpret every instruction and convert it to CPU code. In practice this performance hit is rarely an issue for the kinds of projects you might create with Arduino. If you do run into performance or memory problems MicroPython allows you to write code in C/C++ or even the board's native CPU assembly instructions for maximum perfomance.

Ultimately there's no simple answer for the choice between MicroPython and Arduino. Each has strengths and weaknesses that should be considered for your own projects. Don't be afraid to try MicroPython--for some boards like the

ESP8266 you can actually use either MicroPython or Arduino and pick the best one for your needs.

---

## What hardware supports MicroPython?

Be sure to check the MicroPython website () to see the latest information on supported boards. As of August 2016 these boards support MicroPython in various ways:

- pyboard ()
  - This is the first MicroPython board and has very complete support for the language and hardware peripherals. This board comes to you with MicroPython running on it so you can get started using it immediately without any setup. Check out the pyboard documentation () for more details on its capabilities.
- ESP8266 ()
  - MicroPython support for the popular ESP8266 WiFi microcontroller is excellent. With MicroPython on ESP8266 you can access peripherals like GPIO, ADC, PWM, and I2C/SPI devices. In addition WiFi & internet access is available and well supported. There's even a web-based REPL that allows you to run MicroPython code on the ESP8266 through your web browser! Check out the ESP8266 MicroPython documentation () and the MicroPython ESP8266 FAQ forum page () for more information. If you're looking for an inexpensive and easy board to start with MicroPython the ESP8266 is a great option.
- SAMD21-based Boards ()
  - Atmel SAMD21-based boards like the Feather M0 and Arduino Zero can use CircuitPython, Adafruit's open source derivative of MicroPython. See the Metro M0 Express guide () for more information on using CircuitPython with these boards.
- WiPy ()
  - The WiPy is another MicroPython board with WiFi and great support. Pycom () is the company behind the WiPy board and they provide a nice integrated development environment to load and run MicroPython code on their boards. Be sure to see the WiPy page on Pycom's website () for more information about the board's capabilities, in particular note the board currently doesn't support floating point calculations.
- BBC micro:bit ()
  - The BBC micro:bit has great support for MicroPython and a very nice set of tools to write and upload code. With MicroPython on the micro:bit you can access the board's onboard peripherals including its

LEDs, accelerometer, GPIO, radio, and more. Check out the official micro:bit MicroPython documentation () for more details.

- Teensy 3.x ()
  - The Teensy 3.x series of microcontrollers have an early port of MicroPython available (). Be aware you might need to be familiar with building firmware and compiling code to use this port as there aren't pre-made images available. This port of MicroPython is also a bit less mature compared to other boards but can still access basic peripherals like GPIO on the board. If you're looking at using MicroPython on a Teensy you'll want to check out the Teensy forums () to learn more about what's possible in the current port.

See the MicroPython GitHub repository () for more information on other supported boards & platforms.

---

## Do I need to be a Python expert to use MicroPython?

Absolutely not! MicroPython implements most of the Python 3 core language and as such it can be as simple or complex as you need. As a Python beginner you'll feel right at home accessing MicroPython's REPL and experimenting with code. Python is praised for having a clean and easy to learn syntax that's great for beginners.

If you're new to Python it will help to get familiar with the basics of Python programming on your computer. Some good free resources for learning Python are:

- Learning Python resources from the Hitchhiker's Guide to Python ()
- Learn Python The Hard Way ()

If you're more experienced with Python you'll be amazed at just how much of Python's expressive power is in MicroPython. Advanced concepts like functional and dynamic programming are all possible with MicroPython. You can even dive into the MicroPython codebase and start extending it with new functions, libraries, and more.

---

## Where can I learn more about MicroPython?

Check out the following resources to learn more about MicroPython and how to use it with specific boards:

- MicroPython homepage ()
- MicroPython pyboard documentation ()

- MicroPython ESP8266 documentation () and MicroPython ESP8266 FAQ forum page ()
- MicroPython BBC micro:bit documentation ()
- MicroPython WiPy documentation ()
- MicroPython Developer Wiki ()
- Differences between MicroPython and standard desktop Python ()

---

## How do I get help with MicroPython?

There's a growing community of MicroPython users who might be able to help you if you have questions or issues:

- MicroPython forums ()
- MicroPython GitHub home () (be courteous and don't clutter GitHub issues with troubleshooting & tech support that's better done on forums)

Remember MicroPython is primarily a volunteer effort so be respectful of people spending their spare time to help others!

---

## How do I get started with MicroPython?

Check out more guides in the MicroPython Basics series:

- MicroPython Basics: How to Load MicroPython on a Board ()
- MicroPython Basics: Blink a LED ()
- MicroPython Basics: Load Files & Run Code ()
- MicroPython Basics: ESP8266 WebREPL ()
- MicroPython Basics: Loading Modules ()

Also check the MicroPython category on the learning system () for more MicroPython guides!