Chapter 1: Getting Started with Pico and MicroPython

Note

Before you start this chapter, please make sure you have learned the basic concepts of computers. It is covered in a chapter called Chapter 0 and is available from our website.

http:www.MicroDigitalEd.com

Section 1.1: Raspberry Pi Pico Features and History

The Raspberry Pi is a commonly used platform for teaching and learning programming. It was started by the Raspberry Pi foundation in the UK which is a non-profit charity organization. The first board called Raspberry Pi Model B came out in 2012 and now it is broadly used by students from elementary schools to colleges. The original Raspberry Pi was built on an Arm chip made by Broadcom Corp. Since then, it has gone through several upgrades. See https://www.raspberrypi.org/products/ for more details. The latest Raspberry Pi offering is called Pico. See Figures 1-1 to 1-3. The Pico board uses an Arm Cortex M0 chip designed by the Pi foundation itself. The Arm CPU microcontroller used in the Pico board is called RP2040. The RP is short for Raspberry Pi and will be explained shortly. The Pico can be programmed in Python or C/C++.

Figure 1-1: RP Pico board https://www.raspberrypi.org/products/

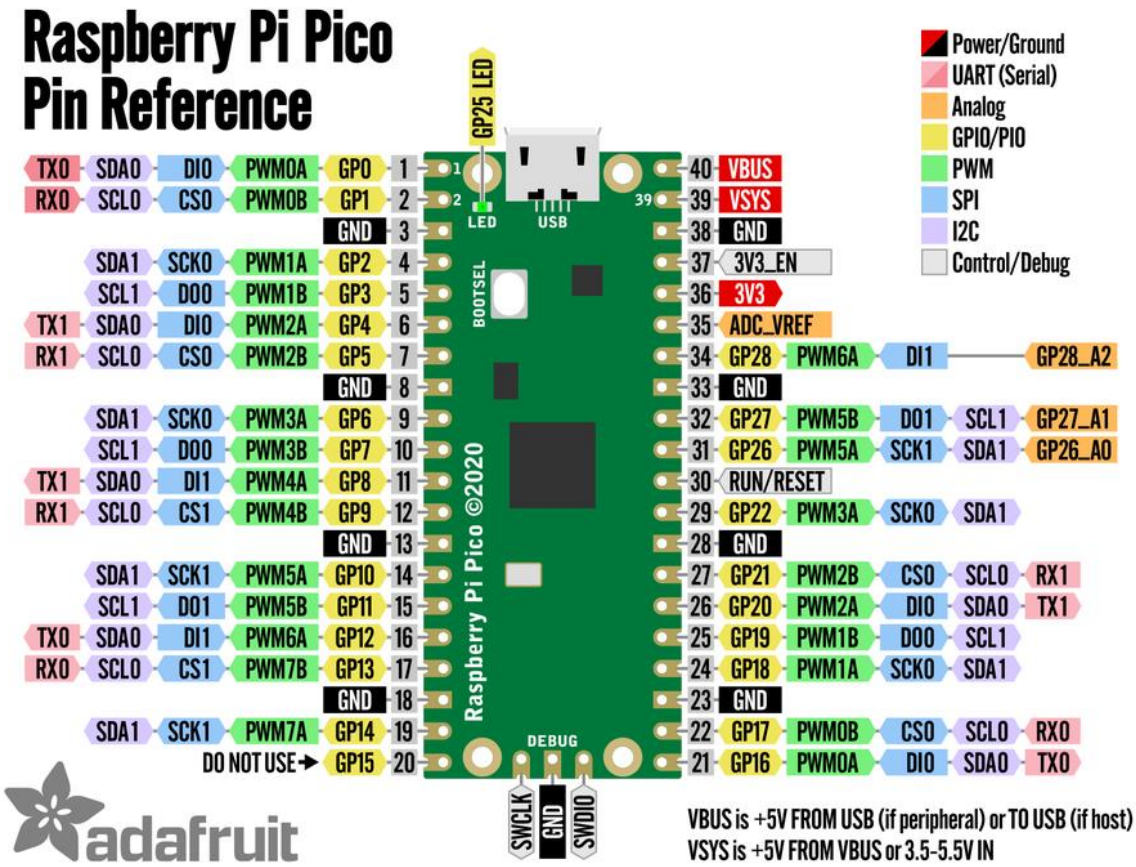


Figure 1-2: Raspberry Pi Pico Pin Reference (https://www.adafruit.com/product/4864)

The RP Pico board comes without a header. You can also purchase them with a soldered header. See the figure below.

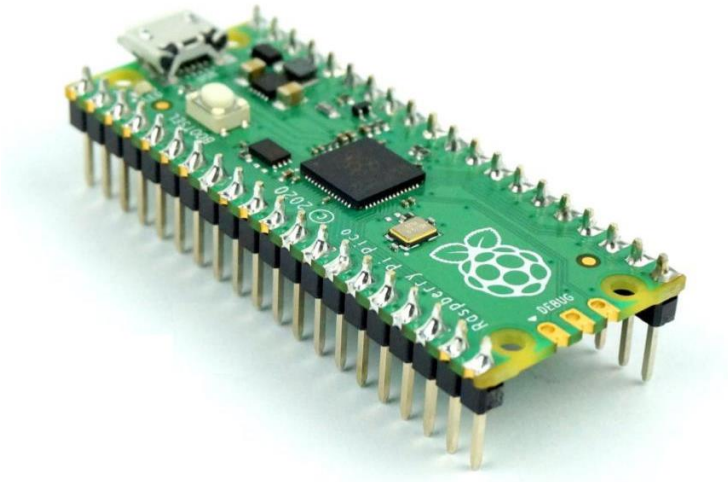


Figure 1-3: RP Pico board with pin headers

Note
For more information about RP Pico see the following websites:
https://www.raspberrypi.org/

RP2040 Microcontroller

As we stated, the Pico board uses the RP2040, a 56-pin microcontroller chip from the Raspberry Pi foundation. This is in contrast to other Raspberry Pi products which use an Arm chip made by Broadcom. See Appendix A for RP2040 microcontroller pins.

The Broadcom BCM2835/36 chips are used in the Raspberry Pi Model A, B, B+, the Compute Module, and the Raspberry Pi Zero board. They are intended as a credit card size (or smaller) single-board computer. With years of enhancements, the latest Pi 4 Model B has a 1.5 GHz quad-core CPU and up to 8G RAM. It can handle dual high-resolution monitors comfortably as a Linux computer.

The Pico is the first board that uses a microcontroller designed in-house by the Raspberry Pi foundation. As a microcontroller, it is targeting a different field of applications. The numbering of the RP2040 microcontroller has the following format:

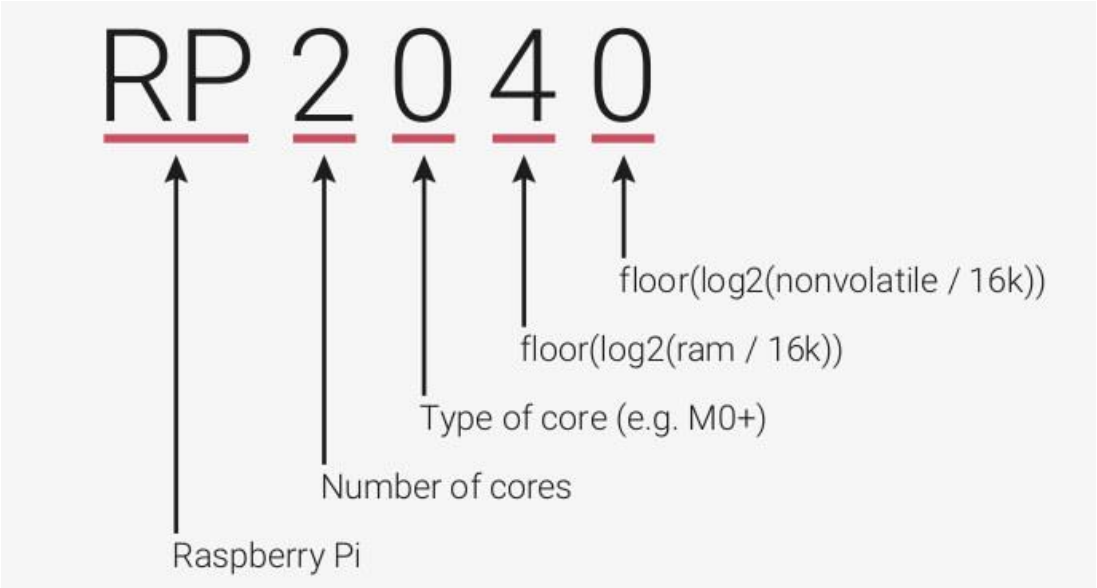


Figure 1-4: RP 2040 naming convention

The first digit of RP2040 shows that the chip has two CPU cores. The second digit specifies the type of Arm Cortex core used. It can be M0, M0+, M3, M4, or M7. The RP Arm chips can come with Flash non-volatile memory and RAMs. In the case of the RP2040, it has 264KB of RAM and zero amount of on-chip Flash memory. However, the RP2040 has 16K of ROM (read-only memory) and can be connected to external Flash memory by the QSPI (quad serial peripheral interface). Like most microcontrollers, it has a large number of on-chip peripherals. See Figure 1-5.

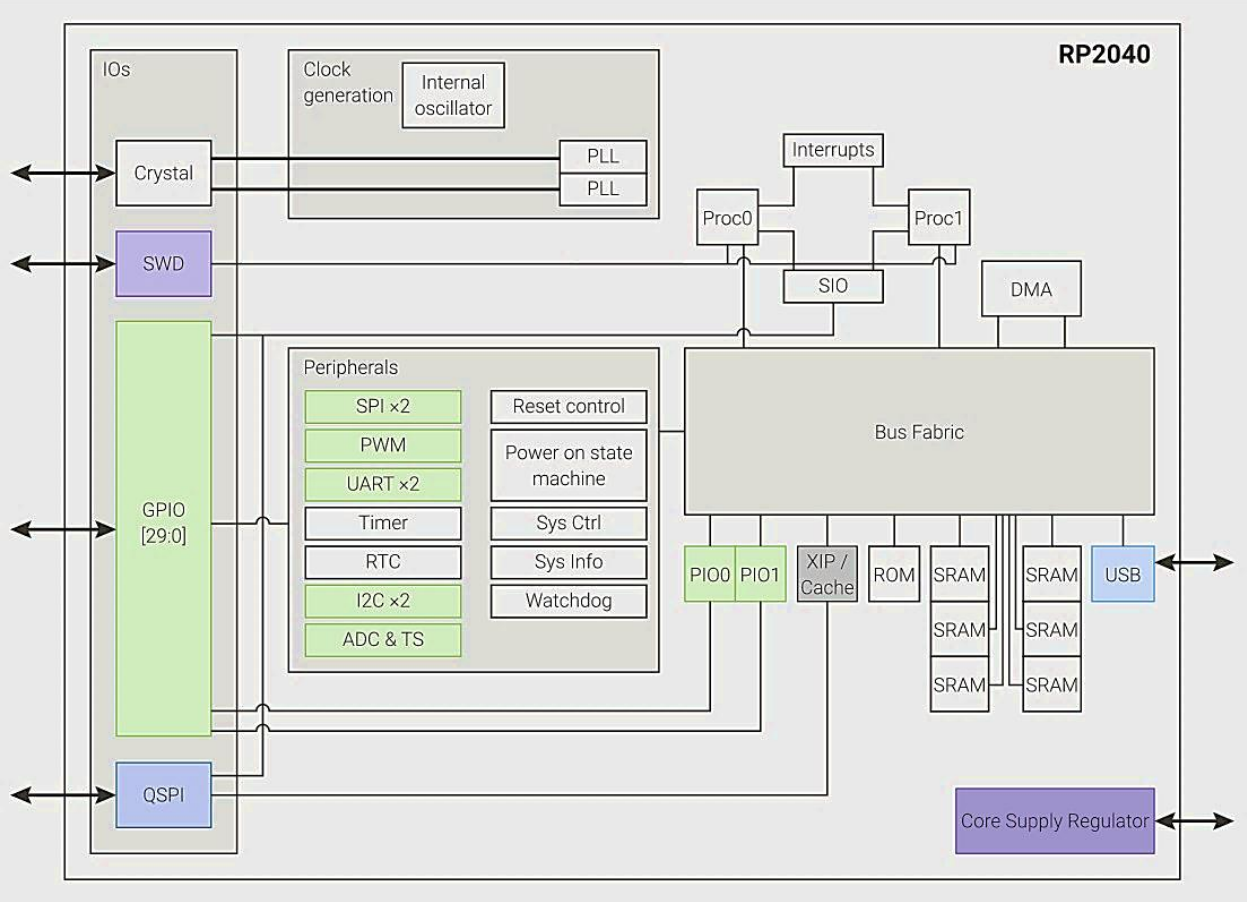


Figure 1-5: RP2040 Microcontroller High-Level Block Diagram

Regarding Figure 1-5, the following points must be noted:

1) The ROM is used for program code and read-only data. The content of the ROM is fixed in the factory and cannot be modified by the user. In the case of the Pico, it is loaded with the bootloader.
2) The SRAM can be used for program code, variables, scratchpad, heap, or stack.
3) The XIP (Execute-In-Place) module translates the QSPI to external flash access as local memory so the CPU can execute code out of external flash memory as if it is in the local memory. To speed up the external flash access, a 16K cache is added to the XIP module.
4) The peripherals such as GPIO, Timers, UART, ADCs, and so on do the work of the microcontrollers. Future chapters will show how they work.

Raspberry Pi Pico Board

The Pico board has an RP2040 microcontroller. It has a 12 MHz crystal and can be used to generate up to 133 MHz precision system clock.

An external 2 MB QSPI flash memory is connected to the microcontroller by the QSPI interface. SPI is a serial peripheral interface, which we will discuss in detail in the future chapter. Quad SPI is an SPI with four data lines and hence has four times the throughput.

A micro USB connector is used to supply 5V power to the board. A switching power regulator on board converts it to 3.3V to be used by the microcontroller and the flash memory. The USB connection can also be used for a serial port over USB as a communication channel between the Pico board and a host computer, which could be a PC, a Mac, or a Raspberry Pi.

The ROM of the RP2040 is loaded with a drag-and-drop bootloader. If the BOOTSEL push-button switch is held down while connecting the USB cable, the bootloader will run and the Pico board appears as a USB mass storage device to the host computer. By dragging a program file in UF2 format into this folder on the host computer, the file will be programmed into the external flash memory.

If the Pico board is connected to the USB without holding down the BOOTSEL switch, the boot code in the ROM will run instead of the bootloader. The boot code sets up the QSPI and XIP modules and then executes the program loaded in the external flash memory.

From the Raspberry Pi website, you may find the C/C++ SDK (Software Development Kit) to generate UF2 files from C or C++ programs. But in this book, we are going to focus on Python programming only. You can download the MicroPython UF2 file for the Pico board from the Raspberry Pi website or the MicroPython website. https://micropython.org

MicroPython for Raspberry Pi Pico

MicroPython is a Python 3 implementation for microcontrollers. When you drag the downloaded MicroPython UF2 file into the Pico bootloader folder, the MicroPython interpreter will be programmed into the external flash of the Pico board. In addition to the Python interpreter, it will also provide a 1.4 MB flash filesystem that can be used by the MicroPython interpreter and your Python programs. In addition to Python, several library modules to support the Pico board hardware are included. We will be using them later in this book.

The MicroPython REPL (Read-Eval-Print Loop, the interactive connection) is on the virtual USB serial connection. A terminal emulator program such as TeraTerm for Windows, screen for macOS, or picocom or minicom for Linux is needed to talk to REPL.

Although Python programming can be done through REPL, it is easier to write the Python scripts (program code) in a text file and load/execute the file using REPL. This way, if you need to make modifications or corrections to the scripts, you do not have to type in the whole thing again. A Python IDE will make this procedure even easier. We will discuss that in the following section.

In the next section, we will describe the details of setting up the MicroPython development environment.

Portability

All the programs in this book were written for and tested on the Raspberry Pi Pico board but these programs will work on some other hardware platforms with little or no modifications. MicroPython was written on the pyboard, which has an STM32F MCU, it has been ported to some STM32F and other Arm-based boards. It has also been ported to many other processors. You may find them on the Download page of the MicroPython website https://micropython.org/download/. Out of these boards, the Arduino Nano RP2040 Connect board is the closest one because it has the same RP2040 MCU on board. The only modifications you need are the pin assignments. For other boards, more changes are needed. For example, the way the I/O pins are named and the pins designated for special functions are different.

Section 1.2: Getting Started with MicroPython

In this book, we assume that you have some experience programming in Python. If Python language is somewhat foreign to you or you feel that a refresh will help you, please check out Appendix C toward the end of the book.

Download MicroPython

There are several places where you can download MicroPython for Raspberry Pi Pico, one being the MicroPython website: https://micropython.org/download/rp2-pico/.

On that webpage are several last stable releases and several latest loads under Nightly build. As a beginner, you might want to stay with the stable releases. Through this book, rp2-pico-20210618-v1.16.uf2 will be used. (By the time this book is published, newer versions of the files will be available and we expect them to work fine with the programs in this book). Click on the file name and the file will be downloaded to your computer.

Figure 1-6: From the MicroPython.org website, choose the latest stable release.

MicroPython is free and open-source. If you browse around the downloads of the MicroPython website, you will also find the GitHub link of the source code. They do provide instructions on how to build the uf2 file for the Pico board or other boards from the source code but that is an advanced topic we are not planning to cover in this book.

The other place to find the file is on the Raspberry Pi website.

Hold the BOOTSEL button (the only button) down on the Pico board while connecting the USB cable to a host computer. A window for the bootloader folder should appear on the host computer.

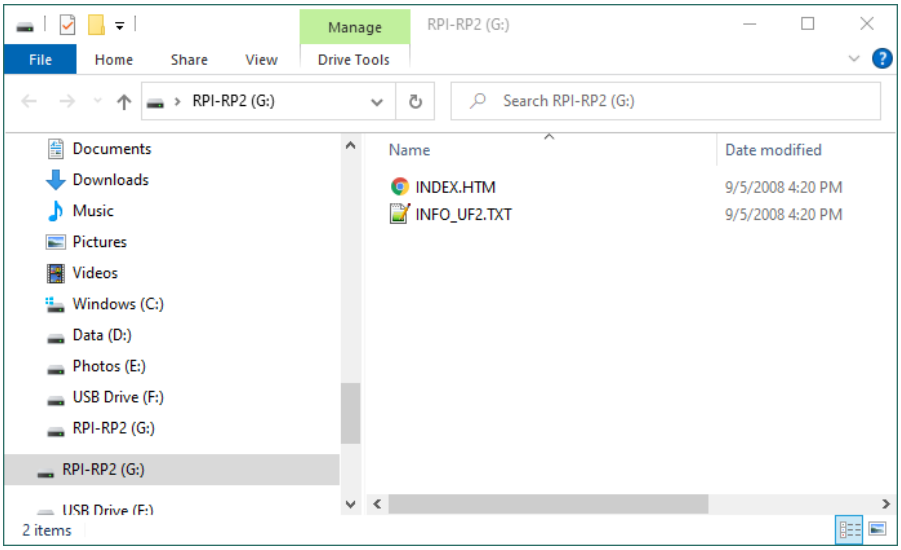


Figure 1-7: Pico board bootloader appears as a folder on the host computer display

Click on INDEX.HTM brings you to the Raspberry Pi Pico website.

On the page, click on the "Getting started MicroPython" link will bring you to step-by-step instructions starting from downloading the file.

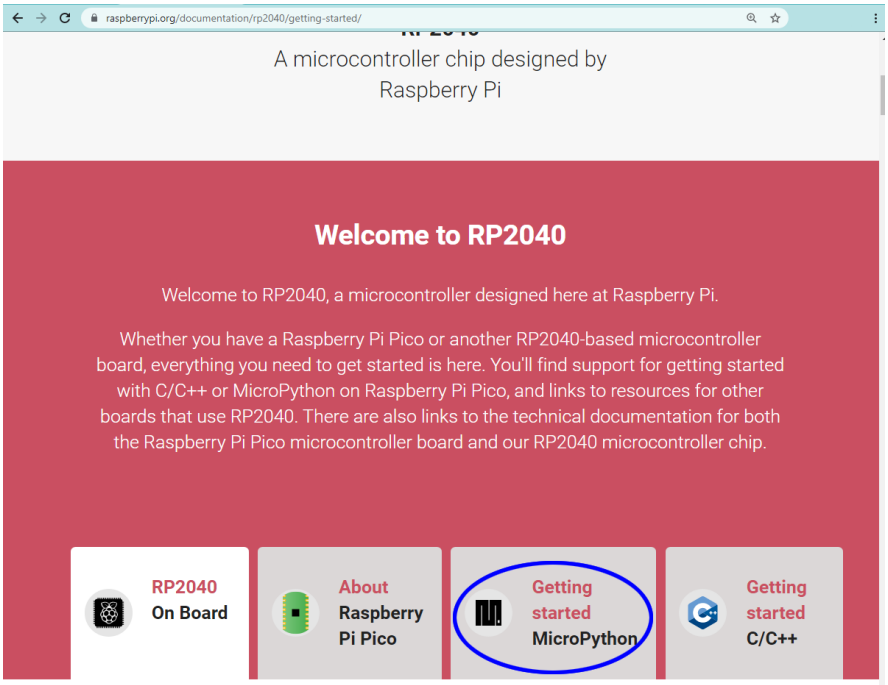


Figure 1-8: Link to step-by-step instructions to download and install MicroPython on Raspberry Pi Pico board

Install MicroPython in Pico Board

Find the .uf2 file in the download folder, drag and drop the file in the bootloader folder in Figure 1-7. The MicroPython will be programmed into the external flash memory of the Pico board and the execution of the Python interpreter started. If the virtual serial connection is established between the Pico board and the host computer, you should be able to see the REPL prompt in the terminal window.

The MicroPython REPL is on the virtual USB serial connection. A terminal emulator program such as TeraTerm for Windows, screen for MacOS, or picocom or minicom for Linux is needed to talk to REPL. You should be able to find the downloads for these programs by searching the Internet. Alternatively, if you jump forward to "Install Thonny IDE," you may use the "Shell" pane of Thonny IDE as the REPL terminal. We will discuss Thonny IDE later in this chapter.

Because the Pico REPL USB connection is pure virtual, no physical serial port is involved, and the Baud rate and other terminal settings have no effect.

REPL Prompt

Python is an interpretive language. When you run Python from the command prompt of a computer, the Python interpreter is launched under the operating system, whether it is Windows, macOS, or Linux. MicroPython does not run under an operating system. It runs standalone or bare-metal as embedded programmers like to call it. When Python is ready to accept input, it displays the REPL prompt which is ">>> " at the left margin of the terminal.

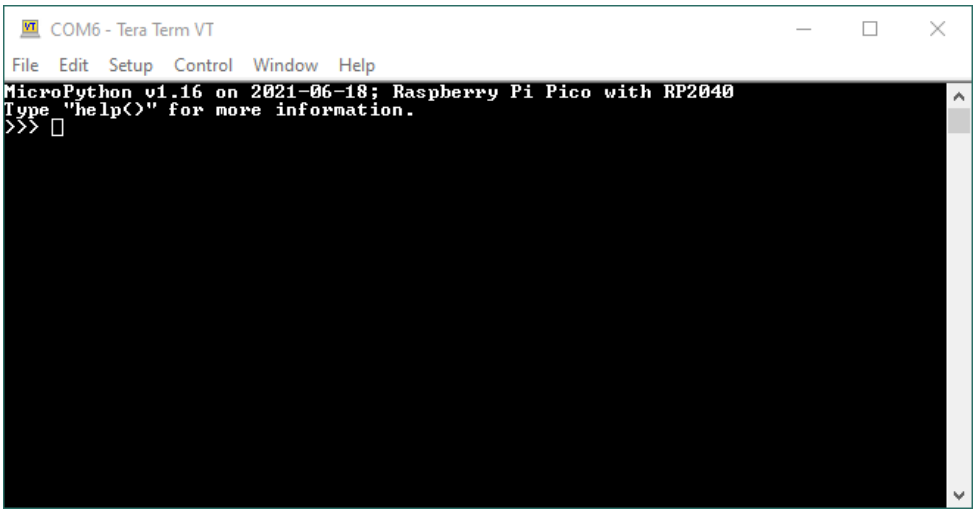


Figure 1-9: MicroPython launch message and REPL prompt from Raspberry Pi Pico

What is typed after the REPL prompt will be **read** by the Python interpreter when you hit the Enter key. The line you typed is **evaluated** and the result is **printed** on the terminal. Then Python interpreter will **loop** back to start over again with another prompt. This is what REPL stands for, Read, Evaluate, Print, and Loop.

Let's give it a try. At the REPL prompt, type "3 * 5" then hit Enter. Python will evaluate the expression and print the result 15. A new REPL prompt is displayed and wait for your next input.

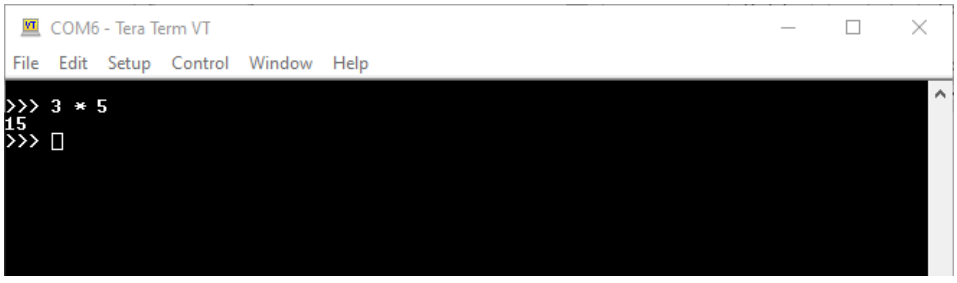


Figure 1-10: MicroPython evaluates 3 * 5 and prints the result 15 followed by a new prompt.

Note that as long as there is no ambiguity, white space is optional. “3 * 5” is the same as “3*5” in Python. But you should sprinkle spaces to make the code easier for the human to read. And better yet, follow the Python Coding Style Guide (PEP 8).

We can start writing Python programs. For starter, the famous hello world, at the prompt type:

```
>>> print('hello world')
```

When you hit Enter key, the phrase “hello world” will be printed on the terminal and a new prompt is displayed. (In Python, a literal string can be enclosed by single quotes or double quotes.)

```
>>> print ('hello world')
hello world
>>>
```

Next, we will write a multiple-line program to calculate the square of five numbers. At the prompt, type

```
>>> a = (2, 3, 4, 5, 6)
>>>
```

to declare a tuple with five integers. A tuple is an array of immutable objects, which means once the values are assigned, they cannot be modified. The next line is:

```
>>> a = (2, 3, 4, 5, 6)
>>> for n in a:
...
```

When you hit Enter, the prompt changes to “...” and the cursor is indented by four spaces. This is the MicroPython REPL auto-indent. When you end a line with ‘:’ you are starting a code block. Unlike some other languages using a pair of braces to enclose a block, Python uses indentation to signify a block.

```
>>> a = (2, 3, 4, 5, 6)
>>> for n in a:
...     print(n * n)
...
```

After the next line, Python REPL assumes you are still entering code for the same block so the cursor is still indented. But our code block for the for loop has only one line. To terminate the block, hit the Backspace key and the cursor will “outdent” to the beginning of the line.

```
>>> a = (2, 3, 4, 5, 6)
>>> for n in a:
...     print(n * n)
...
```

Now, hit Enter and the program will be executed and the outputs are printed on the terminal.

```
>>> a = (2, 3, 4, 5, 6)
>>> for n in a:
...     print(n * n)
...
4
9
16
25
36
>>>
```

Alternatively, if you are at the end of the program, just keep hitting the Enter key and REPL will be convinced that you are done typing and want to execute the program.

So far, we asked Python to do tasks that were finished in a fraction of seconds. If we ran a program that went on and on and we decided to stop its execution, hitting Ctrl-C (hold down Ctrl key and hit C key) will terminate the program, and get the REPL prompt back. In the future chapters, we will talk about running programs in the background in later chapters. Hitting Ctrl-C does not terminate the background program. Hitting Ctrl-D will initiate a soft reboot and stop all running programs. Sometimes, some background tasks triggered by hardware will keep running. In that case, unplug the USB cable and power cycle the Pico board should always bring back the REPL prompt.

```
MPY: soft reboot
MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>
```

Helping Hints from REPL

REPL has several ways of providing hints for programming, they are dir(), help(), and autocompletion. These are not meant for the substitutions for reading books or online tutorials/manuals because of the resource limitations, MicroPython REPL help() provides very limited hints. Nonetheless, they are useful reminders while we write Python programs.

“dir()” function at REPL prompt displays the names of the objects/functions in the namespace:

```
>>> dir(tuple)
['__class__', '__name__', 'count', 'index', '__bases__', '__dict__']
>>>
```

Those with dunder (short for double underscore) before and after the name are called magic object/function. We will not be troubled by them in this book.
In addition to the names, “help()” gives more information:

```
>>> help(tuple)
object <class 'tuple'> is of type type
  count -- <function>
  index -- <function>
>>>
```

Tuple being immutable (the objects in the tuple once initiated cannot be modified), it has only two functions. List is mutable and many functions can be used to modify it.

```
>>> help(list)
object <class 'list'> is of type type
  append -- <function>
  clear -- <function>
  copy -- <function>
  count -- <function>
  extend -- <function>
  index -- <function>
  insert -- <function>
  pop -- <function>
  remove -- <function>
  reverse -- <function>
  sort -- <function>
>>>
```

Now, let's enter an array of even numbers:

```
>>> even = [6, 4, 10, 2, 8]
>>>
```

Is "even" a list or a tuple? We can ask REPL by using the "type()" function:

```
>>> even = [6, 4, 10, 2, 8]
>>> type(even)
<class 'list'>
>>>
```

If I want to find out what I may do to the list "even", I can use auto-completion by typing "even." then hitting the "Tab" key:

```
>>> even.
append          clear           copy            count
extend          index           insert          pop
remove          reverse         sort
>>> even.
```

REPL displays all the available functions and then replays what you entered for you to continue.

```
>>> even.s
```

When you enter "s" and hit the Tab key, REPL will complete the line with the only choice starting with "s":

```
>>> even.sort
```

Because "sort" is a function, we need to complete it with a pair of parentheses then hit Enter key:

```
>>> even.sort()
>>> even
[2, 4, 6, 8, 10]
>>>
```

To see the sorted result, enter "even" and hit the Enter key.

If there are multiple choices, REPL will display all the options for you to choose from:

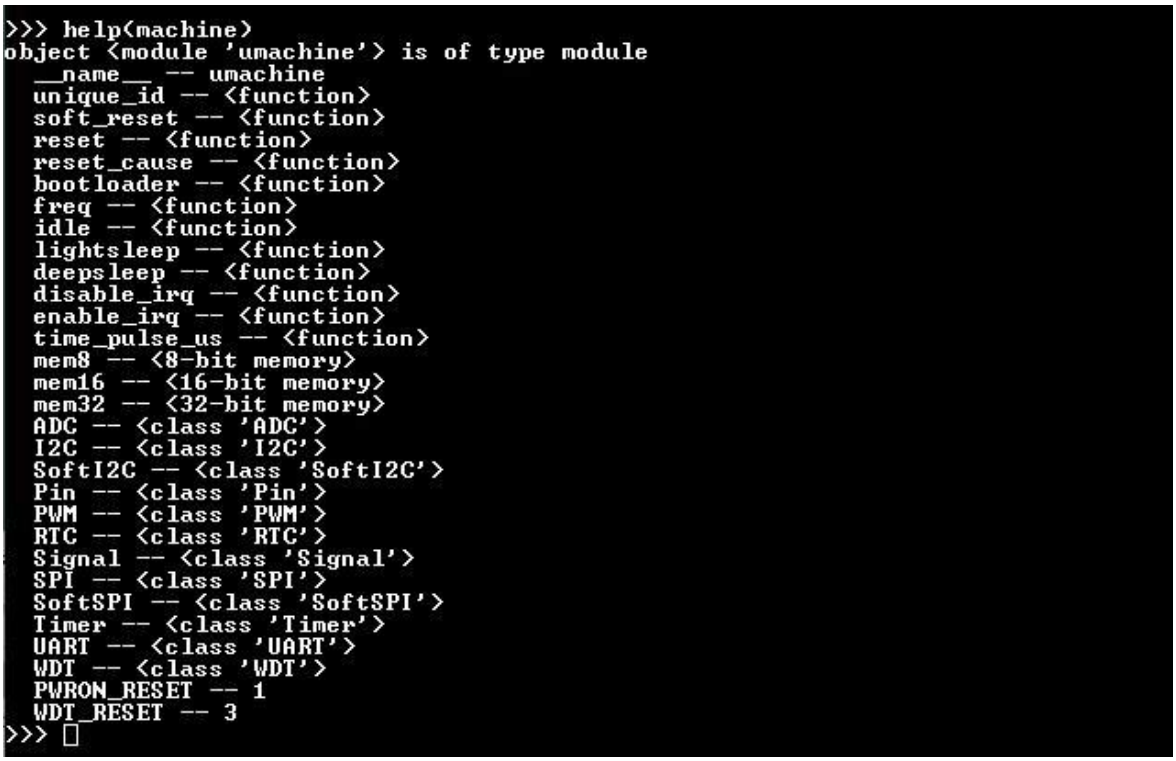
```
>>> even.re
remove          reverse
>>> even.re
```

If nothing can be done by autocompletion and you hit the Tab key, nothing will happen.

"machine", the Pico Specific Module

Unlike a microprocessor, microcontrollers come with a wealth of peripherals. MicroPython is created for microcontrollers, so it comes with a module to handle the peripherals and more. A module is a file containing Python code, which may be classes, functions, and/or variables. This module for the Pico board is named "machine".

We will use the “help()” function of REPL to find what is in the “machine” module:



As you can see, because of the resource limitations, help() provides only the names and types. To find out details of each function or class, you will need to go to the MicroPython website.

Blinky, the Hello World of Microcontroller

In 1978, Brian Kernighan co-authored “The C Programming Language” with the creator of C language, Dennis Ritchie. In this book, he used a three-line code to print the message, “hello world” on the console as the first program example. Because of the popularity of the C programming language and the book, a generation of programmers learned computer programming starting from these three lines of code and it became the standard first program example for many programming language tutorials or books. Somehow the words became capitalized and an exclamation sign was added to the end and it became “Hello World!”.

In microcontroller programming, printing a message on a console may not be an easy task. It involves configuring the serial communication peripheral to match a host computer set up with a console terminal emulator. Many of the microcontroller applications do not even have a serial communication interface. So, blinking an LED connected to a GPIO (general-purpose I/O) pin became the first example program in many introductory documents. Python is different in that it has an interpreter and the REPL. We were able to do “hello world” from the get-go. Now we are going to blink an LED, which turned out to be slightly more complicated.

The Pico board has an onboard LED connected to the GP25 pin, which spared us the effort of building a circuit with an LED. What we need to do is to configure the GP25 pin as an output pin and turn it on and

off. In the "machine" module, there is a class Pin that handles all the GPIO pins of the RP2040 microcontroller.

First, we need to instantiate an LED object using class Pin for GP25:

```
>>> LED = machine.Pin(25)
```

To control the LED, we need to configure the pin as an output pin:

```
>>> LED.init(machine.Pin.OUT)
```

To turn it on and off,

```
>>> LED.on()
```

```
>>> LED.off()
```

In order to turn the LED on and off repetitively, we need to put the last two lines in a loop:

```
>>> while True:
...     LED.on()
...     LED.off()
...
```

Remember after the third line hit the Backspace key and then the Enter key. The program will be running but the LED is on at half the intensity. It is turning on and off so fast that our naked eyes cannot detect it is blinking. If you probe one of the terminals of the LED with an oscilloscope, you should be able to see the signal going between 0 and 2V at about 32 kHz.

To slow down the blinking, we need to insert a delay between turning the LED on and off. The problem with REPL is that we cannot go back to edit the lines we already entered. We have to type in the whole program over again with the new lines inserted[1]. It would be easier if we have an application running on the host computer to manage the files and file editing for us so that we can edit the file on the host computer and reload the new code.

There are several such apps available for MicroPython. In this book, we will be using the Thonny IDE.

Thonny IDE

Thonny was developed as a Python IDE for the beginners at University of Tartu, Estonia. With the participation of the Raspberry Pi Foundation, it supports MicroPython programming on the Raspberry Pi Pico. It is a free download at https://thonny.org/.

[1] Truth be told, REPL does keep the history of all the keyboard entries. You may recall the previous lines use the up arrow key and edit them. But as you will see when the program gets longer, this method gets messier.

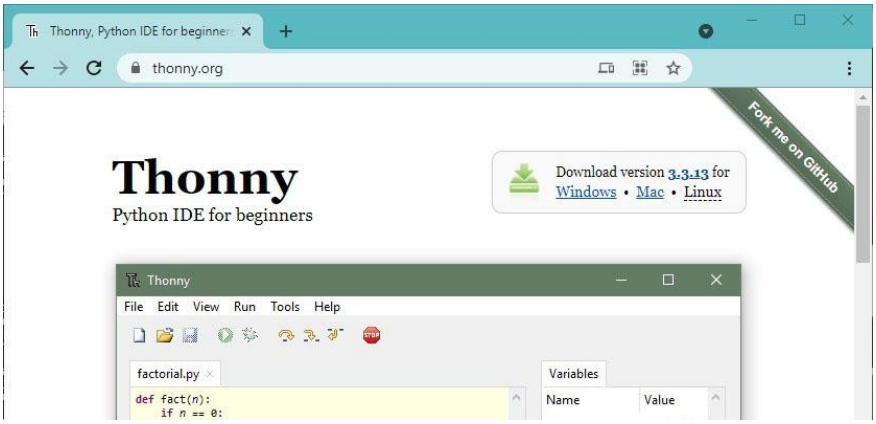


Figure 1-11: Thonny website, where the download links are in the upper-right corner

Once the installation file is downloaded, launch the installation by executing the file. Follow the instructions and Thonny will be installed on your computer.

At the first launch, take the standard initial settings. You will be able to change the settings later. Thonny will open with the bundled Python3, which you can see in the lower right corner of the Thonny window.

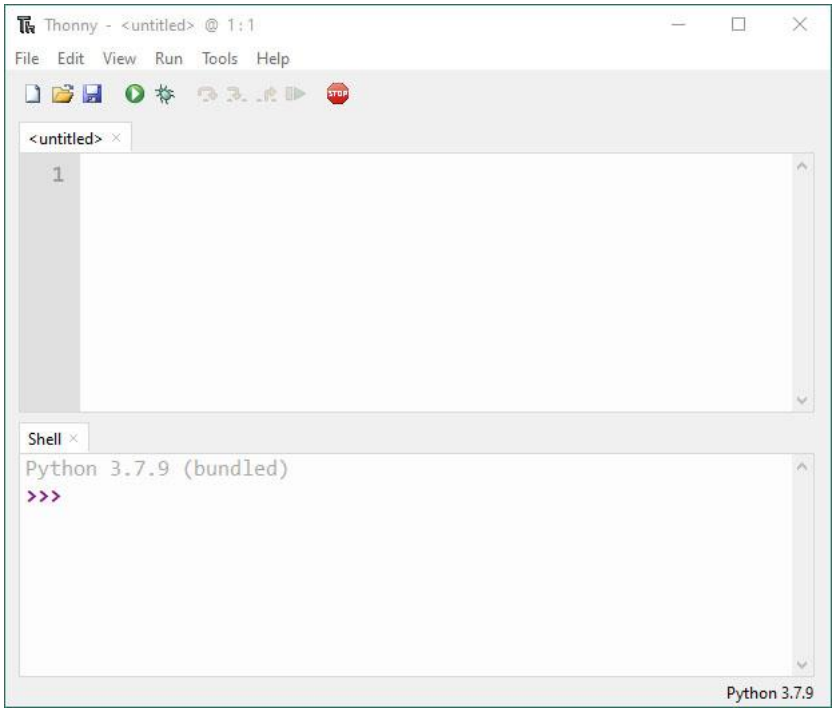


Figure 1-12: The default configuration launch of Thonny

This is the Python running on the host computer. A copy of Python for the host computer is bundled with the Thonny installation. There are two panes in the window, the top is the editor pane and the bottom is the Shell pane (the REPL). If you write Python code this way, the program will be running on the host computer. To run the MicroPython on the Pico board, connect the Pico board to the host computer via a USB cable. If you were using a terminal emulator to talk to Pico before, make sure you close the terminal app first.

When Thonny detects an external connection, clicking on the lower right corner where the Python version is displayed will bring out a pop-up selection.

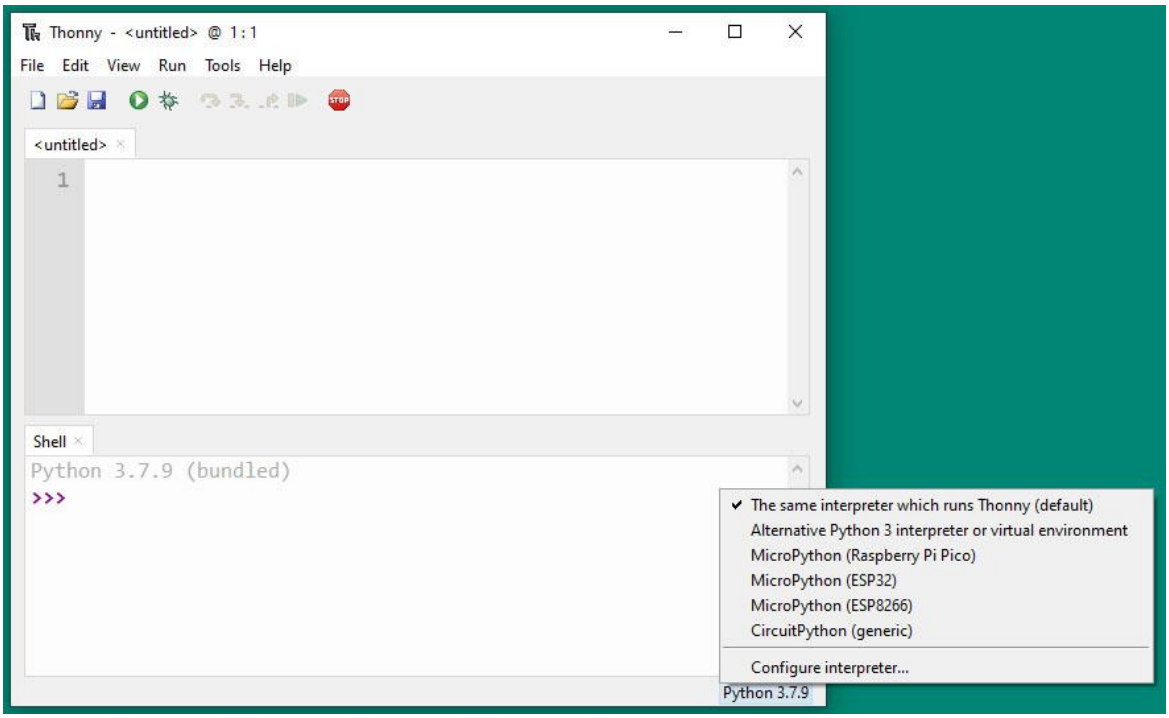


Figure 1-13: When Thonny detects an external Python connection, it displays the possible connections to choose from.

Select “MicroPython (Raspberry Pi Pico)” and the Shell pane should display the REPL prompt from MicroPython.

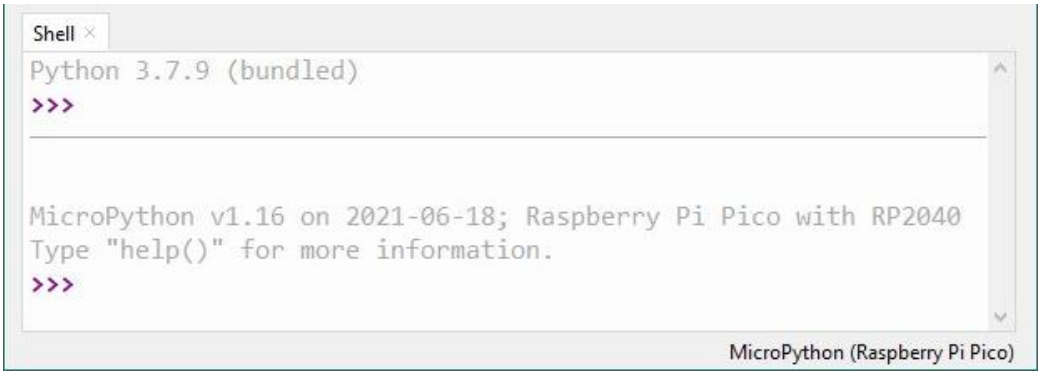


Figure 1-14: When MicroPython (Raspberry Pi Pico) is selected, the Shell makes the connection and displays the REPL prompt.

This REPL is the same as the one we used with the terminal emulator earlier except the output has fancier fonts and colors for different contexts.

Figure 1-15: The REPL prompt in the Shell pane works the same except the outputs have different fonts and colors for different contexts.

Writing Python Program in a File

Previously, we entered a program to blink the onboard LED through REPL. We did not complete the program because it required us to re-enter the program. With Thonny, we will be able to edit the program in a file and reload it much easier. First, let's type up the same code in the editor pane:

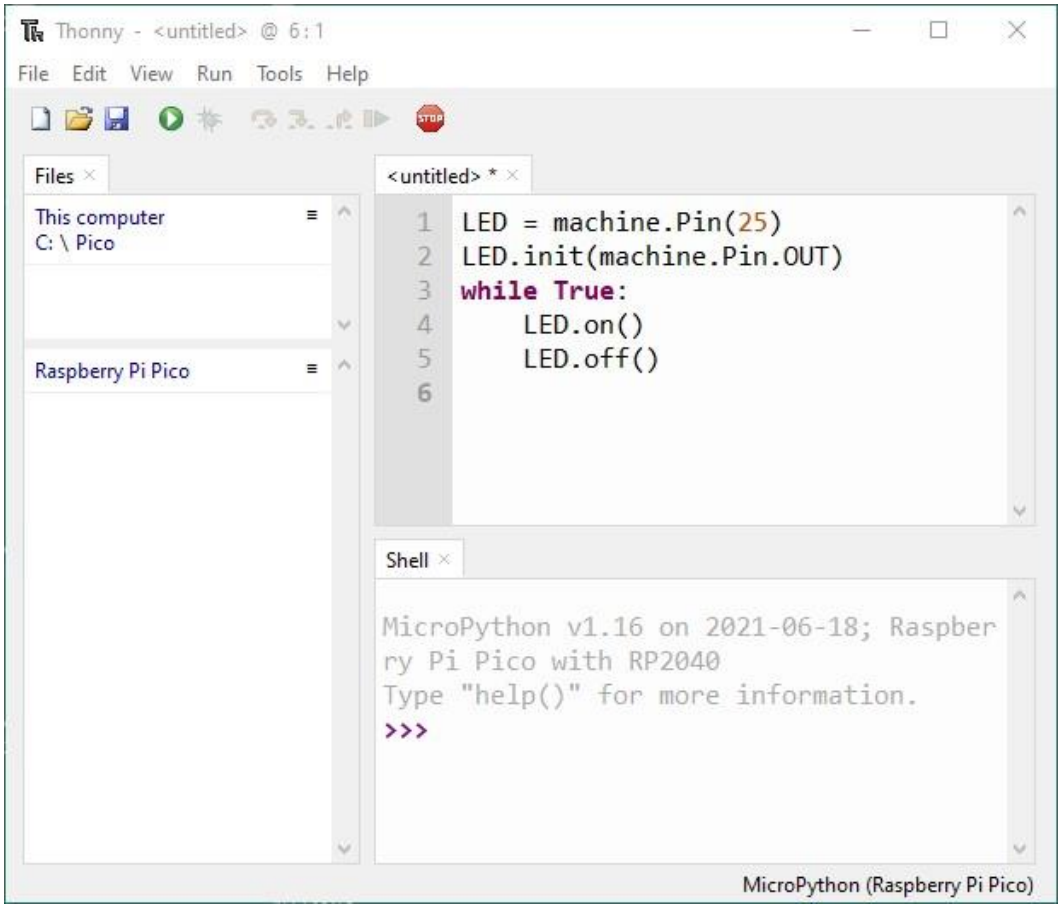


Figure 1-16: Enter the program in the editor pane

When you finished entering the code, you may run the code by clicking the “Run current script” button (the green circle with a white triangle) or hitting the F5 key on the keyboard. Thonny will ask you to save the file first and offers you the choice of saving the file on the host computer or the Pico board.

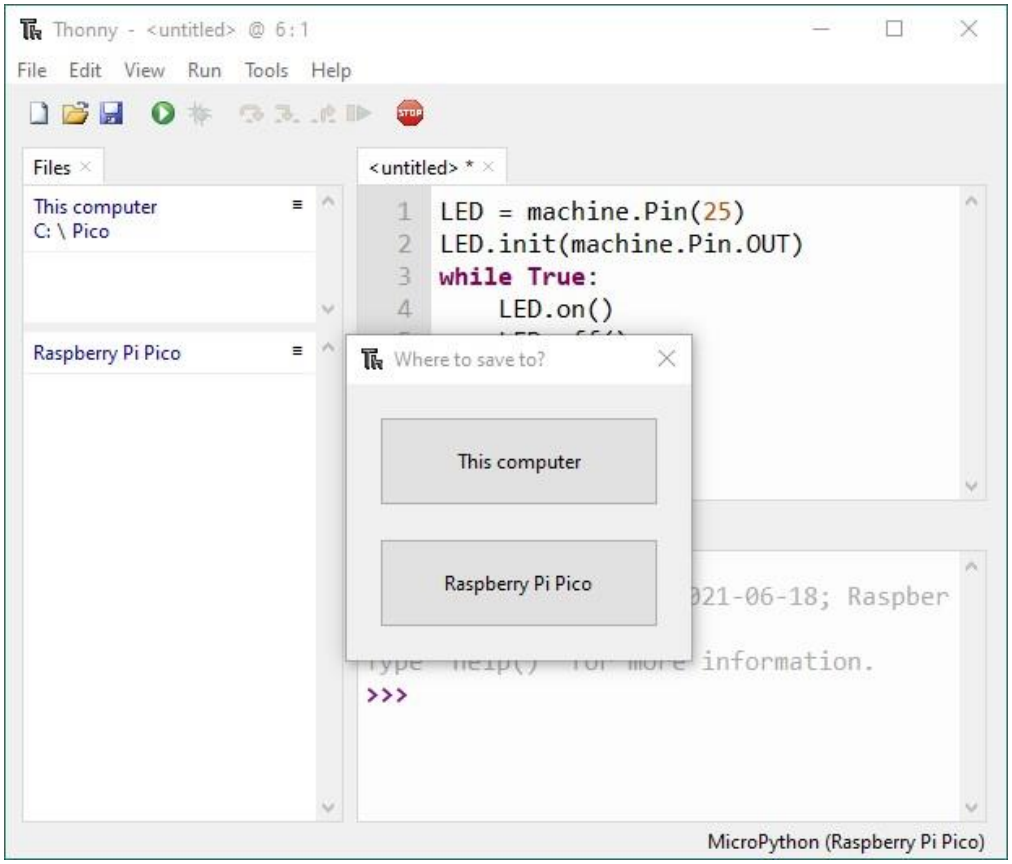


Figure 1-17: Before running the code, Thonny requires you to save it in a file first and offers the choice of saving it at the host computer or the Pico board.

In this example, we will save it on the Pico board so that you can carry the board with the programs with you. The disadvantage of saving the program files in the Pico board is that if you lose the board, you lose your programs. But at any rate, no matter where you save your programs, you should always save a backup copy.

After you chose a destination to save the program to, Thonny will ask you for a file name. The file extension should be .py and you may use any file name except “boot.py” or “main.py”. These two files have special meaning to MicroPython. When MicroPython is rebooted (powered-up reset, Ctrl-D at REPL, or clicking the STOP sign in Thonny), “boot.py” is executed followed by “main.py”. We will talk about this later.

We will name this program “blinky.py”. After you type the file name and click the OK button, the file is saved and the file name should appear in the Files pane on the left. Thonny will issue a REPL command “%Run -c $EDITOR_CONTENT” and the program should be running on the Pico board.

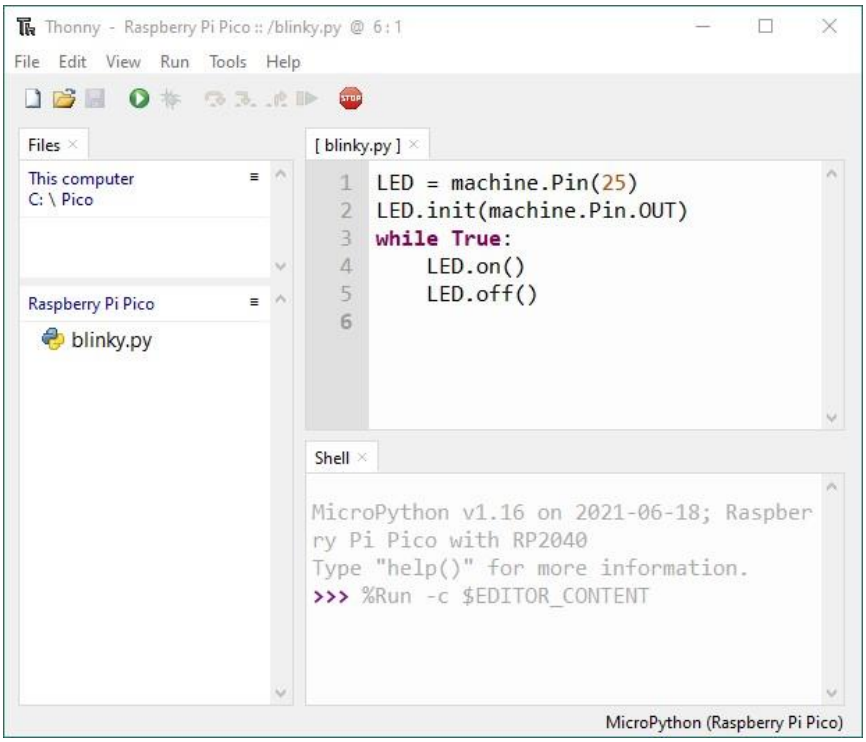


Figure 1-18: After giving a file name, the file should appear in the Files pane, and the program run on the Pico board.

Like before, the LED is blinking so fast that you need an oscilloscope to see the blinking. We will have to insert a delay between turning on and off the LED.

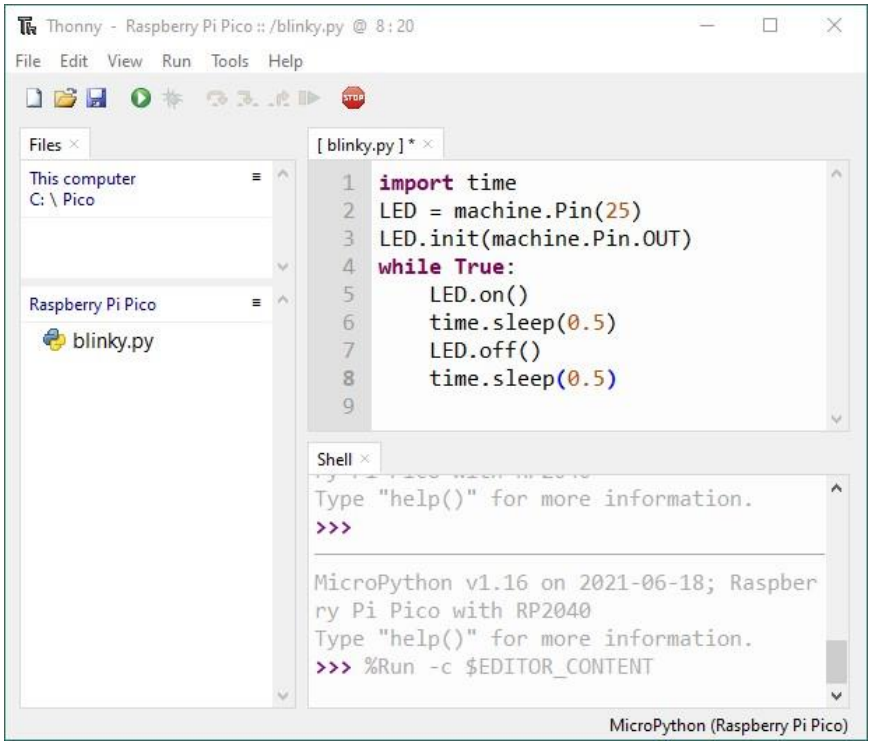


Figure 1-19: Add calls to sleep() function.

MicroPython has a sleep() function in the module “time”. It accepts a floating-point number for the sleep interval with second as the unit. We need to import the “time” module before using the sleep() function.

Before we leave, we want to stop the program execution by clicking at the bottom of the Shell pane and hitting the Ctrl-C key to interrupt the currently running program and get the REPL prompt back.