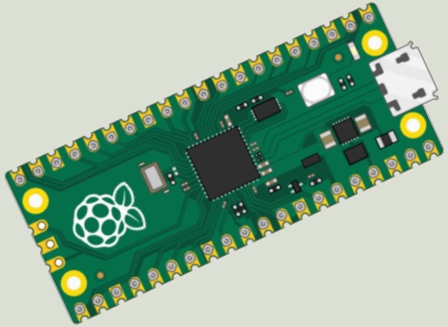
https://www.halvorsen.blog



Raspberry Pi Pico

Hans-Petter Halvorsen

Contents

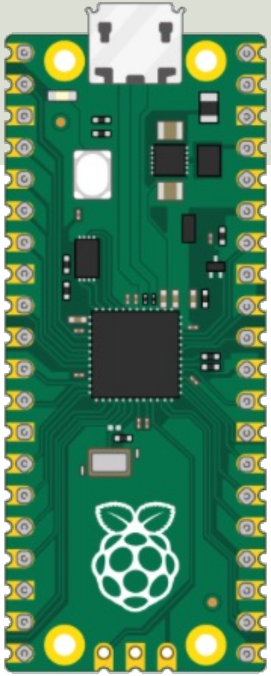
- Introduction
- Raspberry Pi Pico
- Thonny Python Editor
- MicroPython
- Python Examples
  - Blinking onboard LED
  - Blinking external LED
  - Pulse Width Modulation (PWM)
  - Temperature Sensor (TMP36)
- Running Pico without a PC
- PicoZero (Short Introduction)

Introduction

Introduction

- In this Tutorial we are introducing Raspberry Pi Pico
- Raspberry Pi Pico is a “downscaled” version of the original Raspberry Pi and is more comparable with Arduino compared to the original Raspberry Pi
- You also need to use a downscaled version of Python, called MicroPython

Raspberry Pi Pico



- Raspberry Pi Pico is a microcontroller board developed by the Raspberry Pi Foundation
- Raspberry Pi Pico has similar features as Arduino devices
- Raspberry Pi Pico is typically used for Electronics projects, IoT Applications, etc.
- You typically use MicroPython, which is a downscaled version of Python, in order to program it

https://www.raspberrypi.com/products/raspberry-pi-pico/

https://projects.raspberrypi.org/en/projects/getting-started-with-the-pico

What do you need?

- Raspberry Pi Pico
- A Micro-USB cable
- A PC with Thonny Python Editor (or another Python Editor)
- Breadboard
- Electronics Components like LED, Resistors, Jumper wires, etc.

Raspberry Pi Pico

Table of Contents

Raspberry Pi Pico

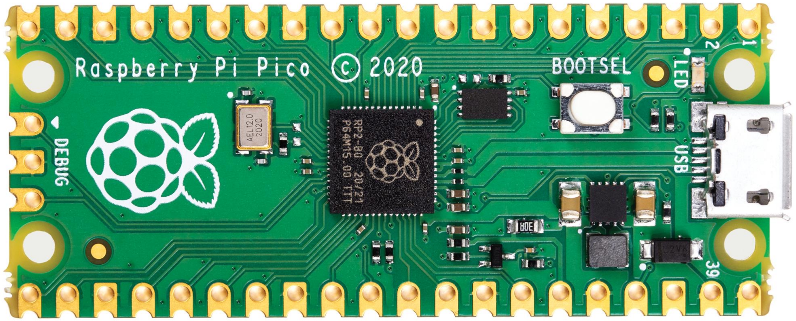
We have 4 different types:

- Raspberry Pi Pico (original)
- Raspberry Pi Pico H - pre-soldered header pins included
- Raspberry Pi Pico W – WiFi included
- Raspberry Pi Pico WH – WiFi and pre-soldered header pins included

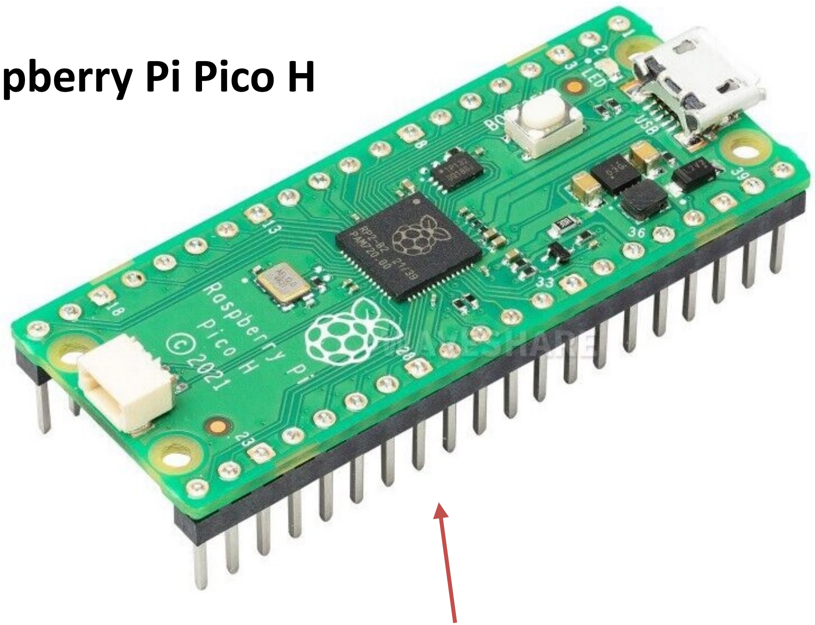
https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html

Raspberry Pi Pico Series

Raspberry Pi Pico (original)

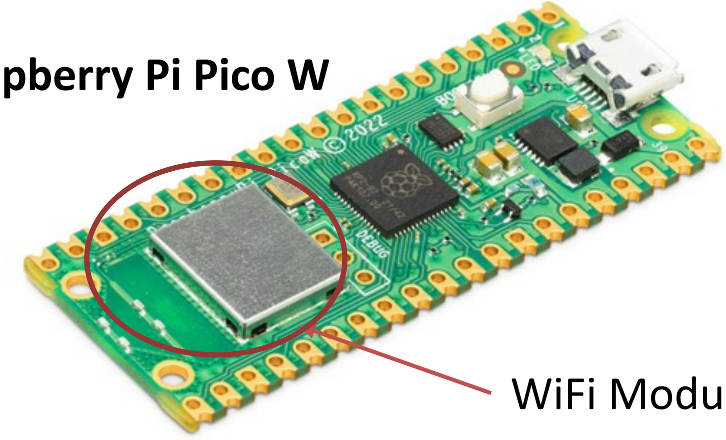


Raspberry Pi Pico H



Pre-soldered header pins included

Raspberry Pi Pico W

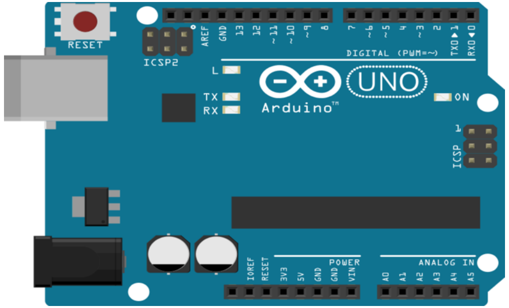


WiFi Module and Antenna

Arduino vs. Raspberry Pi

Arduino Family

Arduino UNO



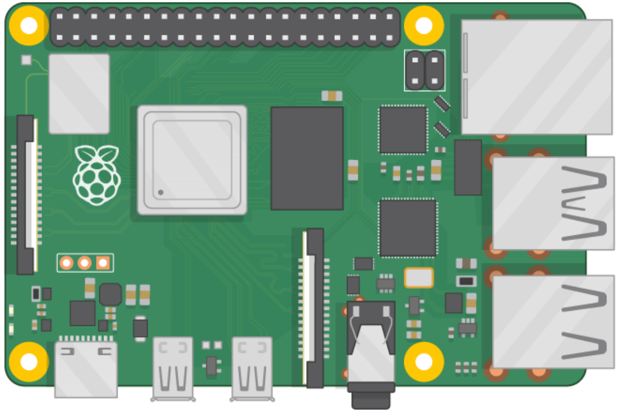
Arduino UNO and similar Arduino boards is a Microcontroller Unit (MCU)

Programming Language: Arduino IDE and C/C++

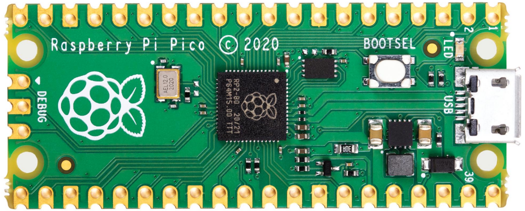
Raspberry Pi
is a Single-Board Computer (SBC), which is a microcontroller unit with CPU, RAM, and external hard disk.

Operating System: Linux
Programming Language: Python + many others

Raspberry Pi



Raspberry Pi Pico



Raspberry Pi Pico is a Microcontroller Unit (MCU)

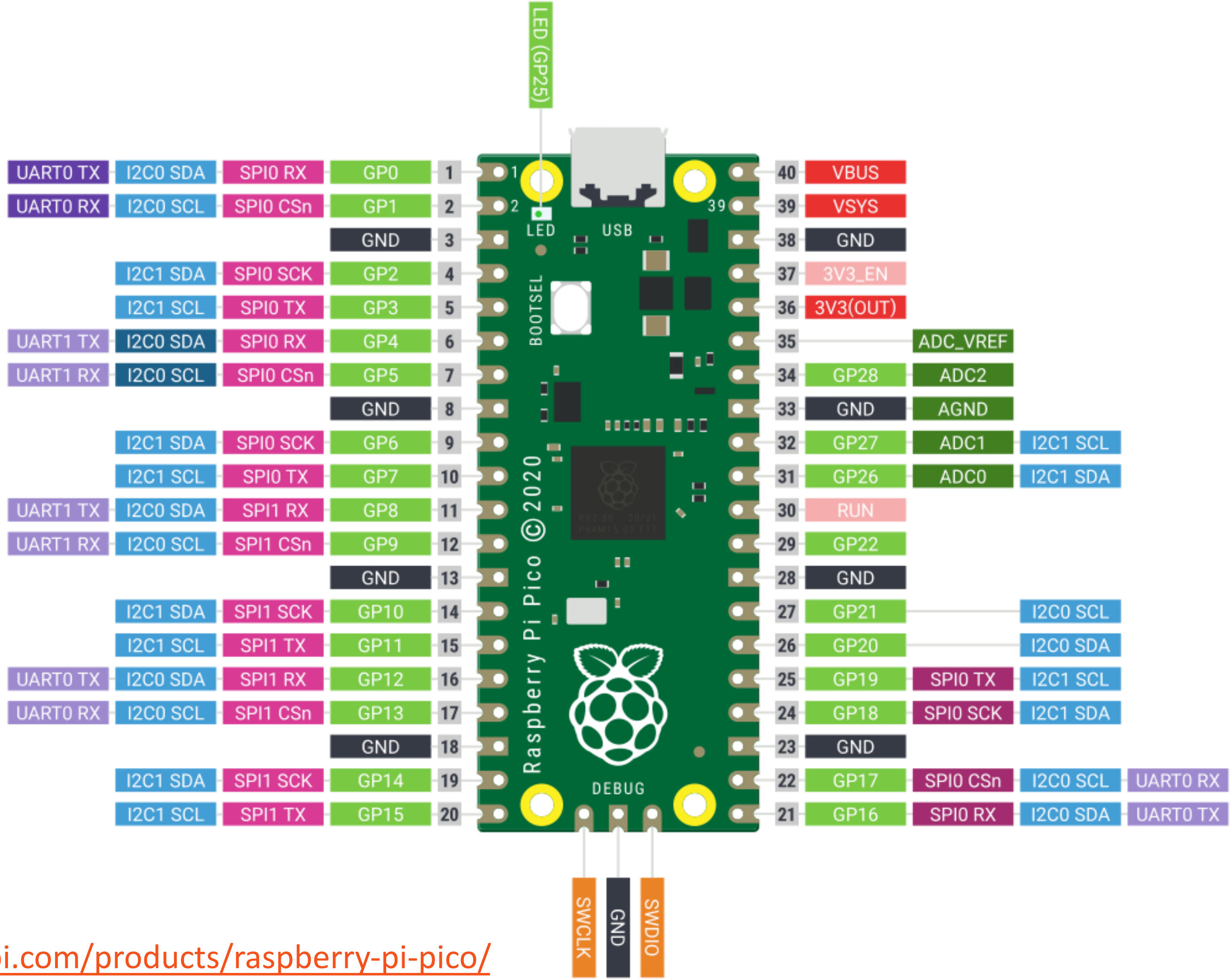
Programming Language: MicroPython or C/C++

Raspberry Pi Pico Specifications

- Size: 21 mm × 51 mm
- **Micro-USB B port** for power and data
- CPU: Dual-core Arm Cortex-M0+ @ 133MHz
- Memory: 264KB on-chip SRAM; 2MB onboard QSPI Flash
- Interface: **26 GPIO pins**, including **3 Analog Inputs (ADC)**
- Peripherals:
  - 2 × UART
  - 2 × **SPI** controllers
  - 2 × **I2C** controllers
  - 16 × **PWM** channels

Pico PinOut

- 🟥 Power
- ⬛ Ground
- ◩ UART / UART (default)
- 🟩 GPIO, PIO, and PWM
- 🟩 ADC
- ◩ SPI / SPI (default)
- ◩ I2C / I2C (default)
- 🟥 System Control
- 🟧 Debugging



UART0 TX	I2C0 SDA	SPI0 RX	GP0	1
UART0 RX	I2C0 SCL	SPI0 CSn	GP1	2
			GND	3
	I2C1 SDA	SPI0 SCK	GP2	4
	I2C1 SCL	SPI0 TX	GP3	5
UART1 TX	I2C0 SDA	SPI0 RX	GP4	6
UART1 RX	I2C0 SCL	SPI0 CSn	GP5	7
			GND	8
	I2C1 SDA	SPI0 SCK	GP6	9
	I2C1 SCL	SPI0 TX	GP7	10
UART1 TX	I2C0 SDA	SPI1 RX	GP8	11
UART1 RX	I2C0 SCL	SPI1 CSn	GP9	12
			GND	13
	I2C1 SDA	SPI1 SCK	GP10	14
	I2C1 SCL	SPI1 TX	GP11	15
UART0 TX	I2C0 SDA	SPI1 RX	GP12	16
UART0 RX	I2C0 SCL	SPI1 CSn	GP13	17
			GND	18
	I2C1 SDA	SPI1 SCK	GP14	19
	I2C1 SCL	SPI1 TX	GP15	20

40	VBUS			
39	VSYS			
38	GND			
37	3V3_EN			
36	3V3(OUT)			
35		ADC_VREF		
34	GP28	ADC2		
33	GND	AGND		
32	GP27	ADC1	I2C1 SCL	
31	GP26	ADC0	I2C1 SDA	
30	RUN			
29	GP22			
28	GND			
27	GP21		I2C0 SCL	
26	GP20		I2C0 SDA	
25	GP19	SPI0 TX	I2C1 SCL	
24	GP18	SPI0 SCK	I2C1 SDA	
23	GND			
22	GP17	SPI0 CSn	I2C0 SCL	UART0 RX
21	GP16	SPI0 RX	I2C0 SDA	UART0 TX

https://www.raspberrypi.com/products/raspberry-pi-pico/

Thonny Python Editor

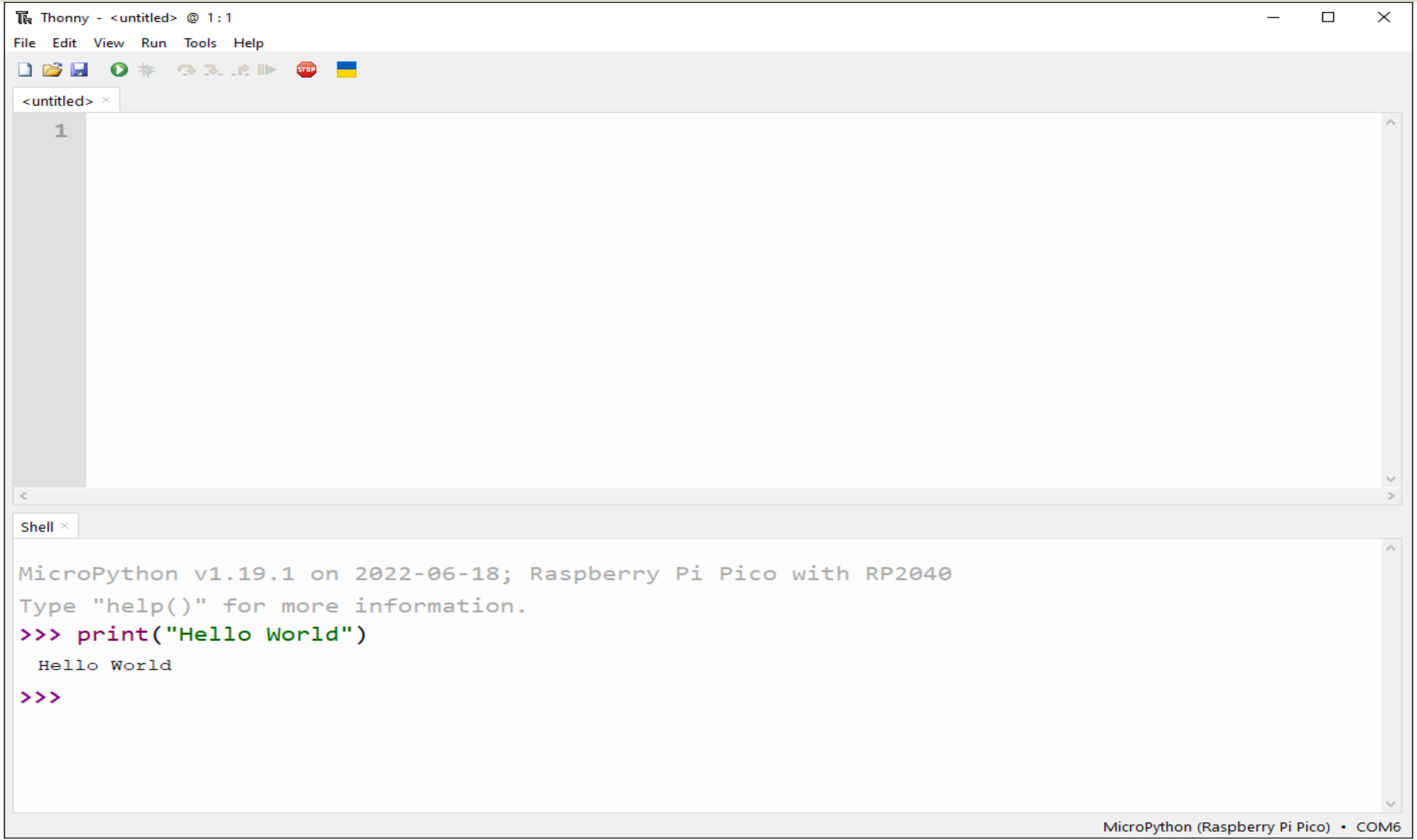
Hans-Petter Halvorsen

Table of Contents

Thonny

- Thonny is a simple and user-friendly Python Editor
- Cross-platform: Windows, macOS and Linux
- Its free
- https://thonny.org

Thonny


Thonny - <untitled> @ 1 : 1
File
Edit
View
Run
Tools
Help
STOP
<untitled>
1
Shell
MicroPython (Raspberry Pi Pico) • COM6

```
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> print("Hello World")
 Hello World
>>>
```

MicroPython

Table of Contents

MicroPython

- MicroPython is a downscaled version of Python
- It is typically used for Microcontrollers and constrained systems

https://docs.micropython.org/en/latest/index.html https://micropython.org

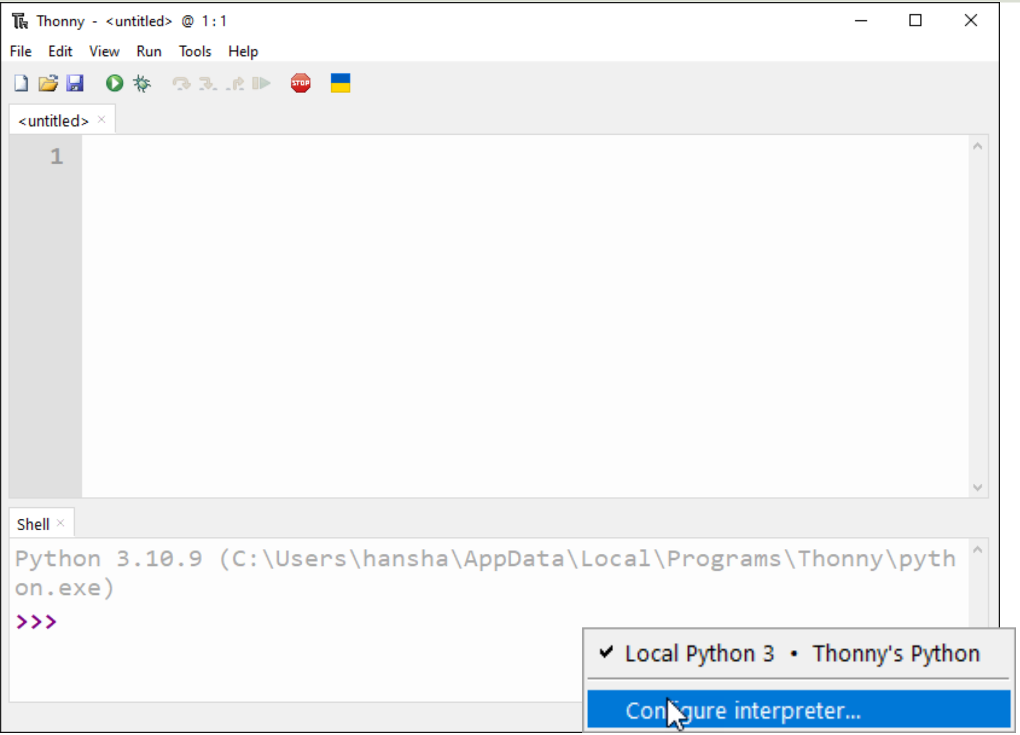
MicroPython Firmware

- The first time you need to install the MicroPython Firmware on your Raspberry Pi Pico
- You can install the MicroPython Firmware manually or you can use the Thonny Editor

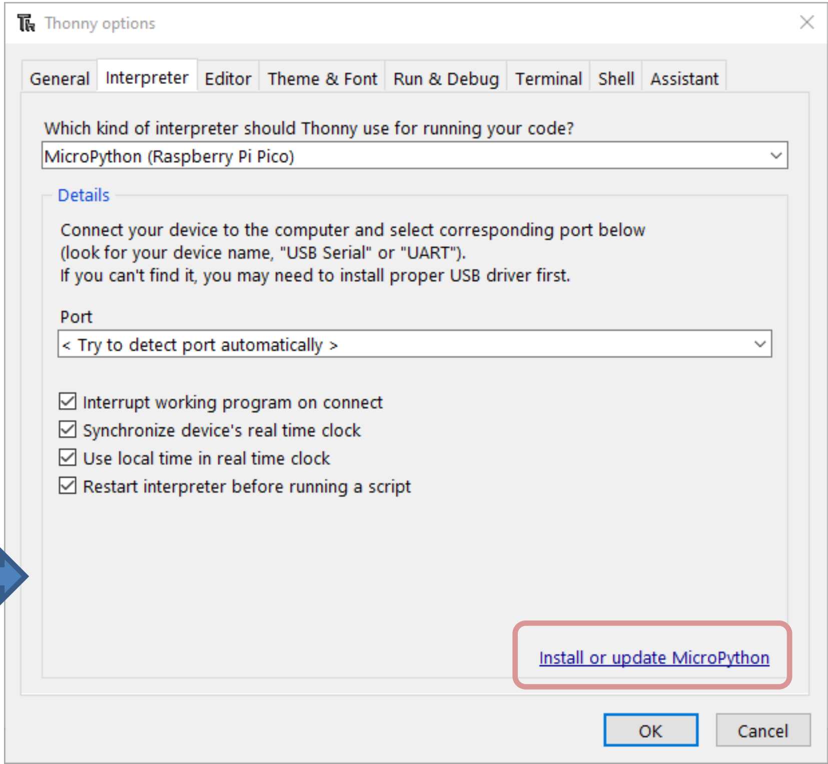
Install MicroPython Firmware Manually

- Download the **MicroPython UF2 File** to your PC https://www.raspberrypi.com/documentation/microcontrollers/micropython.html
- Push and hold the **BOOTSEL button** and plug your Pico into the USB port of your PC. Release the BOOTSEL button after your Pico is connected.
- It will mount as a Mass Storage Device called **RPI-RP2**.
- **Drag and Drop** the MicroPython UF2 File onto the RPI-RP2 volume. Your Pico will reboot.
- You are now running MicroPython

Install MicroPython Firmware using Thonny



```
Python 3.10.9 (C:\Users\hansha\AppData\Local\Programs\Thonny\python.exe)
>>>
```



Which kind of interpreter should Thonny use for running your code?

MicroPython (Raspberry Pi Pico)

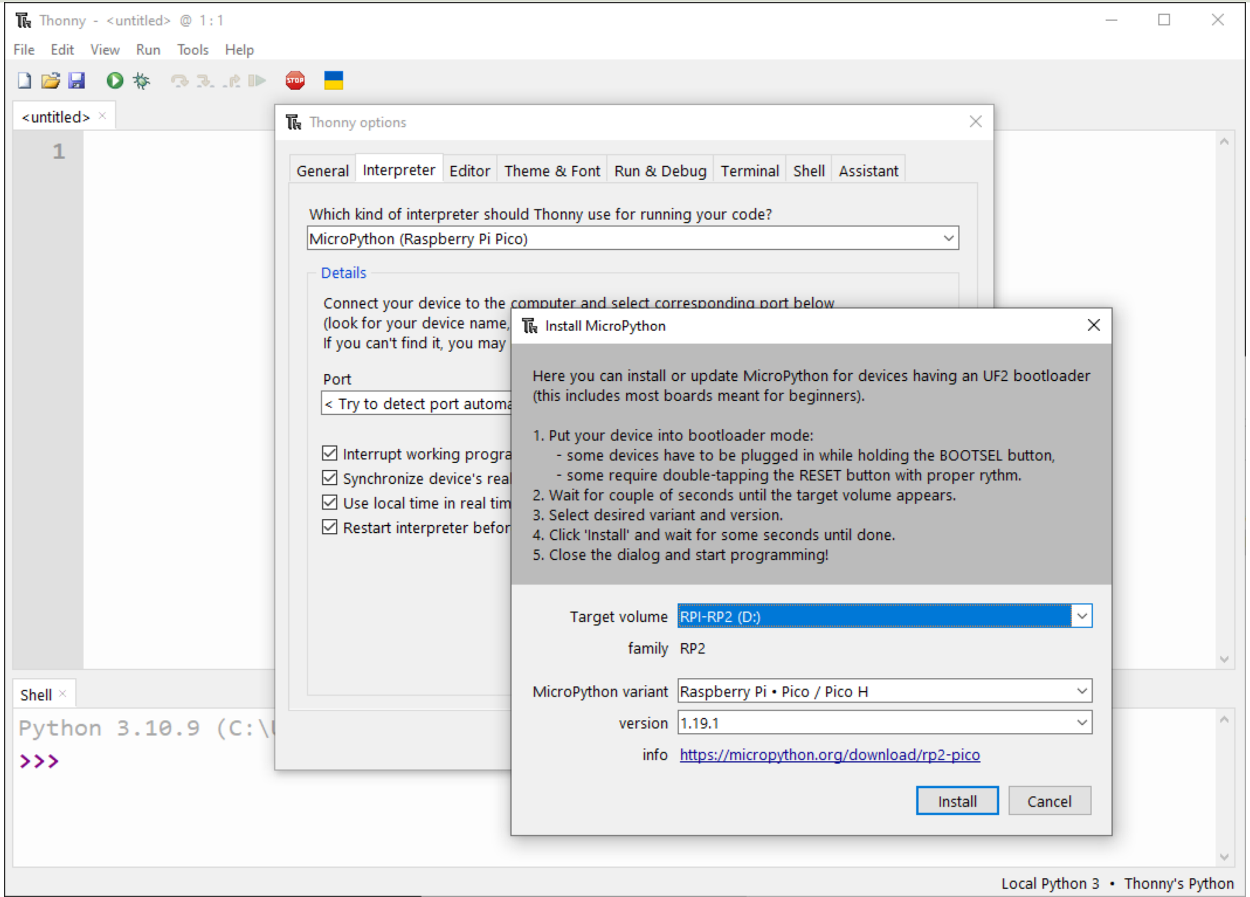
Connect your device to the computer and select corresponding port below
(look for your device name, "USB Serial" or "UART").
If you can't find it, you may need to install proper USB driver first.

Port: < Try to detect port automatically >

☑ Interrupt working program on connect
☑ Synchronize device's real time clock
☑ Use local time in real time clock
☑ Restart interpreter before running a script

https://projects.raspberrypi.org/en/projects/getting-started-with-the-pico/3

Install MicroPython Firmware using Thonny



Thonny - <untitled> @ 1 : 1
File Edit View Run Tools Help
<untitled>
Shell
Python 3.10.9 (C:\
>>>
Thonny options
General Interpreter Editor Theme & Font Run & Debug Terminal Shell Assistant
Details
Local Python 3 • Thonny's Python

Which kind of interpreter should Thonny use for running your code?

MicroPython (Raspberry Pi Pico)

Connect your device to the computer and select corresponding port below
(look for your device name,
If you can't find it, you may

Port: < Try to detect port automa

☑ Interrupt working progra
☑ Synchronize device's real
☑ Use local time in real tim
☑ Restart interpreter befor

Install MicroPython

Here you can install or update MicroPython for devices having an UF2 bootloader (this includes most boards meant for beginners).

1. Put your device into bootloader mode:
   - some devices have to be plugged in while holding the BOOTSEL button,
   - some require double-tapping the RESET button with proper rythm.
2. Wait for couple of seconds until the target volume appears.
3. Select desired variant and version.
4. Click 'Install' and wait for some seconds until done.
5. Close the dialog and start programming!

Target volume: RPI-RP2 (D:)

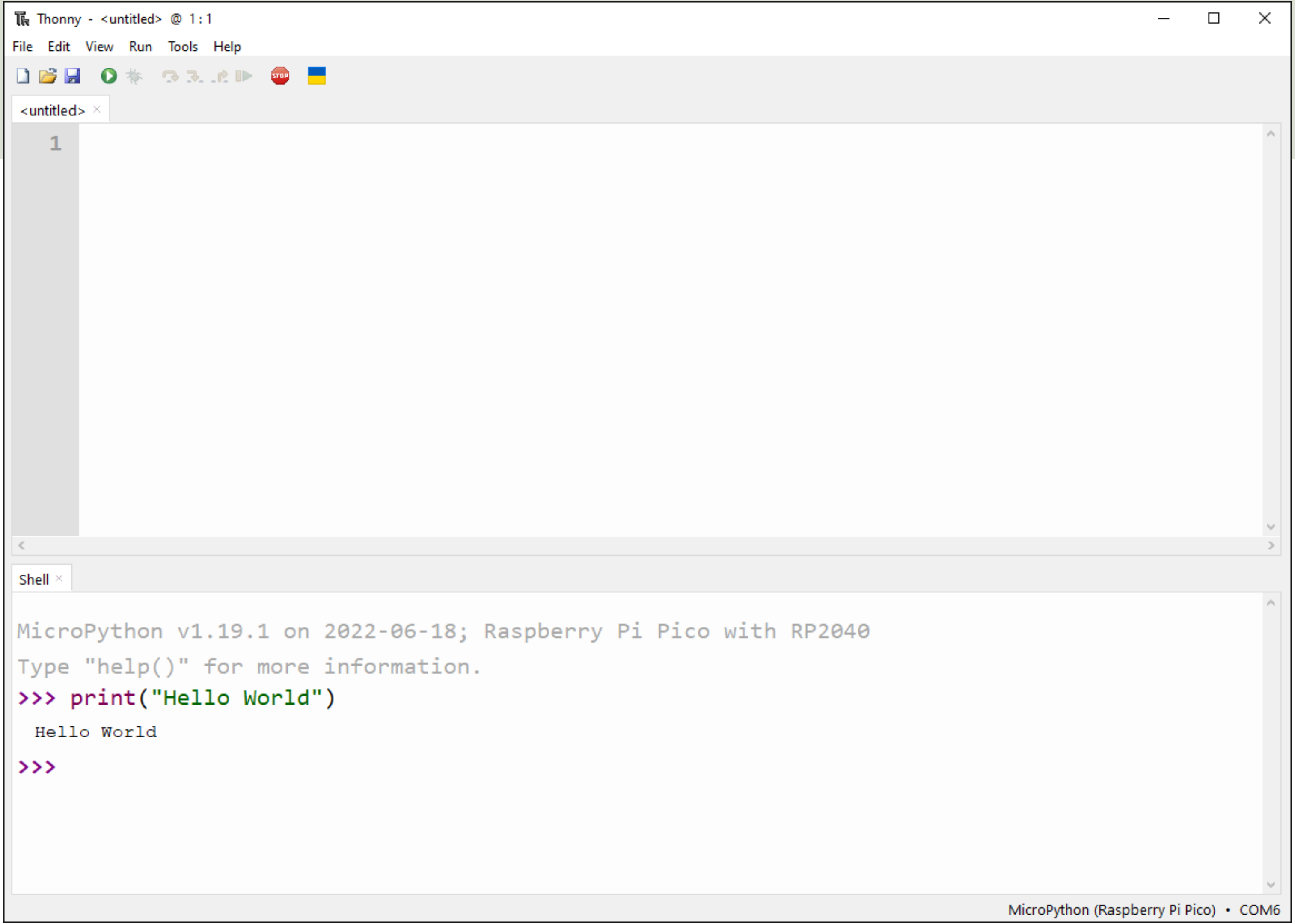
family: RP2

MicroPython variant: Raspberry Pi • Pico / Pico H

version: 1.19.1

info: https://micropython.org/download/rp2-pico

Install   Cancel

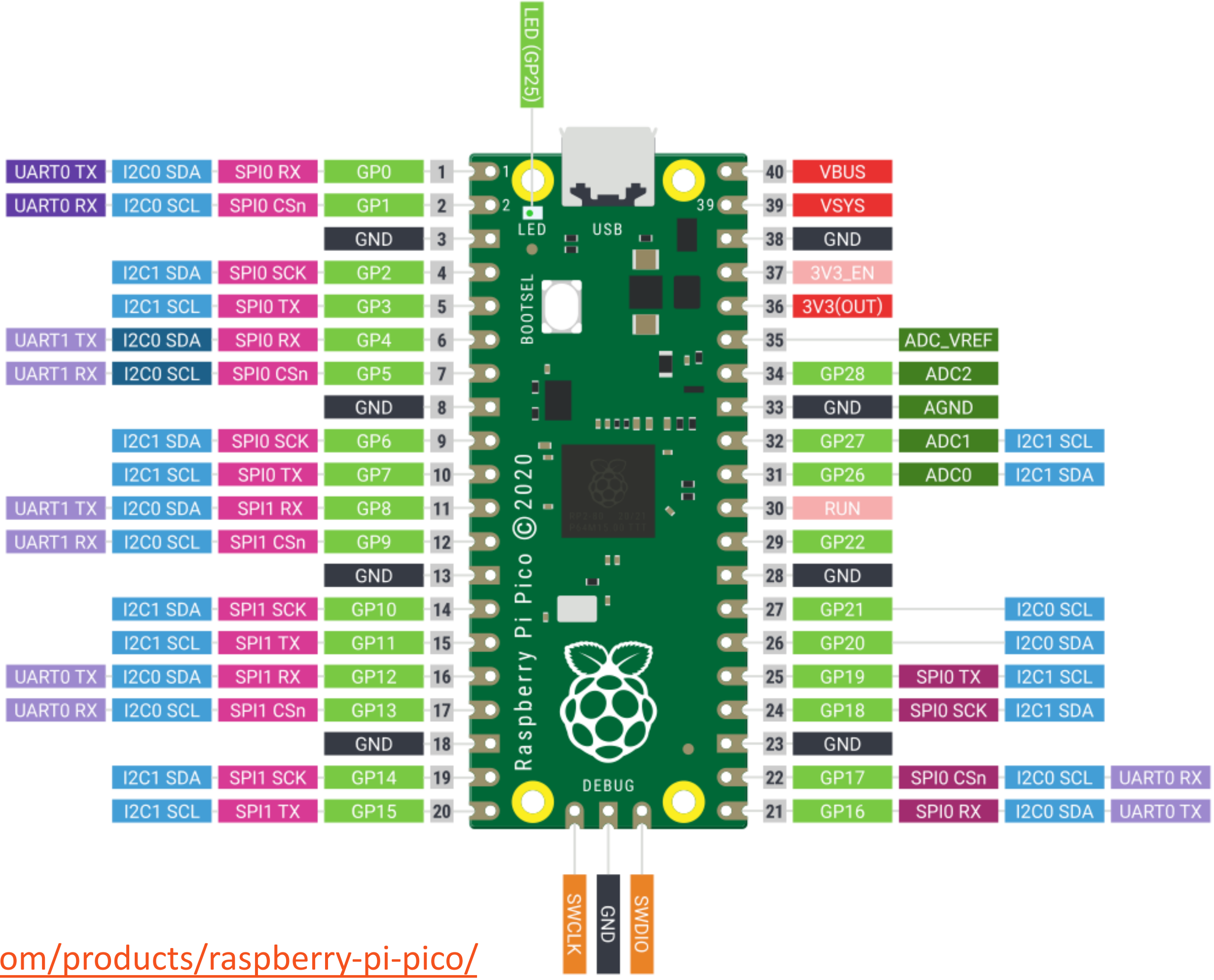

Thonny - <untitled> @ 1 : 1
File
Edit
View
Run
Tools
Help
STOP
<untitled>
1
Shell
MicroPython (Raspberry Pi Pico) • COM6

```
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> print("Hello World")
 Hello World
>>>
```

Python Examples

Pico Pinout

- Power
- Ground
- UART / UART (default)
- GPIO, PIO, and PWM
- ADC
- SPI / SPI (default)
- I2C / I2C (default)
- System Control
- Debugging



UART0 TX	I2C0 SDA	SPI0 RX	GP0	1
UART0 RX	I2C0 SCL	SPI0 CSn	GP1	2
			GND	3
	I2C1 SDA	SPI0 SCK	GP2	4
	I2C1 SCL	SPI0 TX	GP3	5
UART1 TX	I2C0 SDA	SPI0 RX	GP4	6
UART1 RX	I2C0 SCL	SPI0 CSn	GP5	7
			GND	8
	I2C1 SDA	SPI0 SCK	GP6	9
	I2C1 SCL	SPI0 TX	GP7	10
UART1 TX	I2C0 SDA	SPI1 RX	GP8	11
UART1 RX	I2C0 SCL	SPI1 CSn	GP9	12
			GND	13
	I2C1 SDA	SPI1 SCK	GP10	14
	I2C1 SCL	SPI1 TX	GP11	15
UART0 TX	I2C0 SDA	SPI1 RX	GP12	16
UART0 RX	I2C0 SCL	SPI1 CSn	GP13	17
			GND	18
	I2C1 SDA	SPI1 SCK	GP14	19
	I2C1 SCL	SPI1 TX	GP15	20

40	VBUS			
39	VSYS			
38	GND			
37	3V3_EN			
36	3V3(OUT)			
35		ADC_VREF		
34	GP28	ADC2		
33	GND	AGND		
32	GP27	ADC1	I2C1 SCL	
31	GP26	ADC0	I2C1 SDA	
30	RUN			
29	GP22			
28	GND			
27	GP21		I2C0 SCL	
26	GP20		I2C0 SDA	
25	GP19	SPI0 TX	I2C1 SCL	
24	GP18	SPI0 SCK	I2C1 SDA	
23	GND			
22	GP17	SPI0 CSn	I2C0 SCL	UART0 RX
21	GP16	SPI0 RX	I2C0 SDA	UART0 TX

https://www.raspberrypi.com/products/raspberry-pi-pico/

Communicate with the Pins

You need to use the **machine** library in order to communicate with the Pins on the Pico:

```
import machine

.. Your Code
```

The machine library consists of several modules, if you only need the Pin module:

```
from machine import Pin

.. Your Code
```

Communicate Pico Hardware

The **machine** Library within MicroPython has the following Classes/Modules:

- **Pin** – control I/O pins
- Signal – control and sense external I/O devices
- **ADC** – analog to digital conversion
- ADCBlock – control ADC peripherals
- **PWM** – pulse width modulation
- UART – duplex serial communication bus
- **SPI** – a Serial Peripheral Interface bus protocol (controller side)
- **I2C** – a two-wire serial protocol
- I2S – Inter-IC Sound bus protocol
- RTC – real time clock
- **Timer** – control hardware timers
- WDT – watchdog timer
- SD – secure digital memory card (cc3200 port only)
- SDCard – secure digital memory card

https://docs.micropython.org/en/latest/index.html

Blinking onboard LED

Table of Contents

Turn on/off the onboard LED

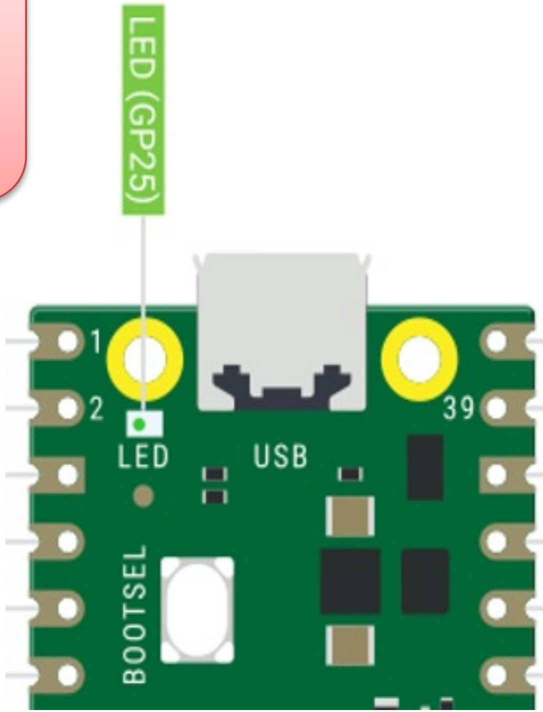
```
import machine

pin = 25
led = machine.Pin(pin, machine.Pin.OUT)
led.value(1)
```

Note! If you are using **Raspberry Pi Pico W** instead of the original Raspberry Pi Pico, you need to do as follows:

```
pin = "LED"
led = machine.Pin(pin, machine.Pin.OUT)
```

Because on the Raspberry Pi Pico W pin 25 is used for internal communication with the WiFi chip.

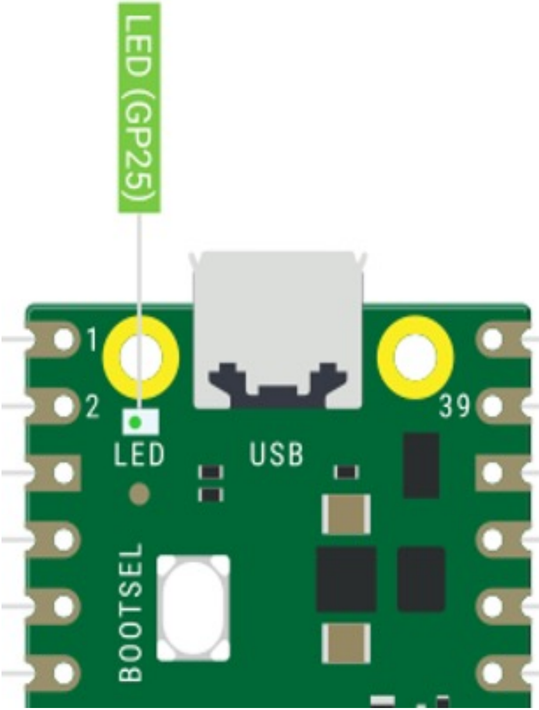
```
import machine

pin = 25
led = machine.Pin(pin, machine.Pin.OUT)
led.value(0)
```



https://projects.raspberrypi.org/en/projects/getting-started-with-the-pico/5

Toggle the onboard LED

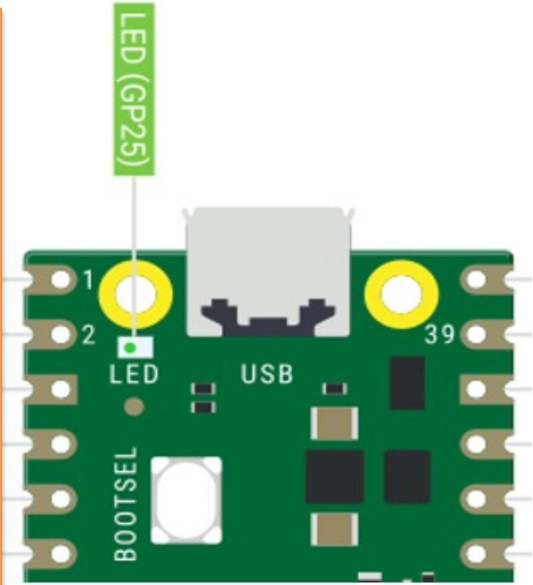
```
import machine

pin = 25

led = machine.Pin(pin, machine.Pin.OUT)

led.toggle()
```



https://projects.raspberrypi.org/en/projects/getting-started-with-the-pico/5

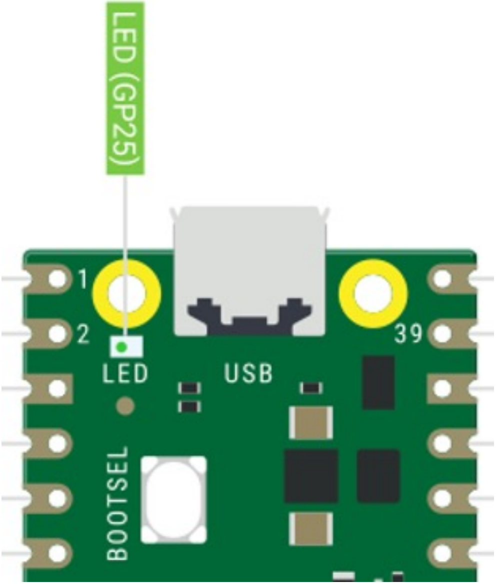
Blink the onboard LED

```
import machine
import time

pin = 25

led = machine.Pin(pin, machine.Pin.OUT)

while True:
    led.value(1)
    time.sleep(2)
    led.value(0)
    time.sleep(2)
```



Blink the onboard LED v2

```
import machine

pin = 25

led = machine.Pin(pin, machine.Pin.OUT)

while True:
    led.value(1)
    machine.lightsleep(1000)
    led.value(0)
    machine.lightsleep(1000)
```



Blink the onboard LED v3

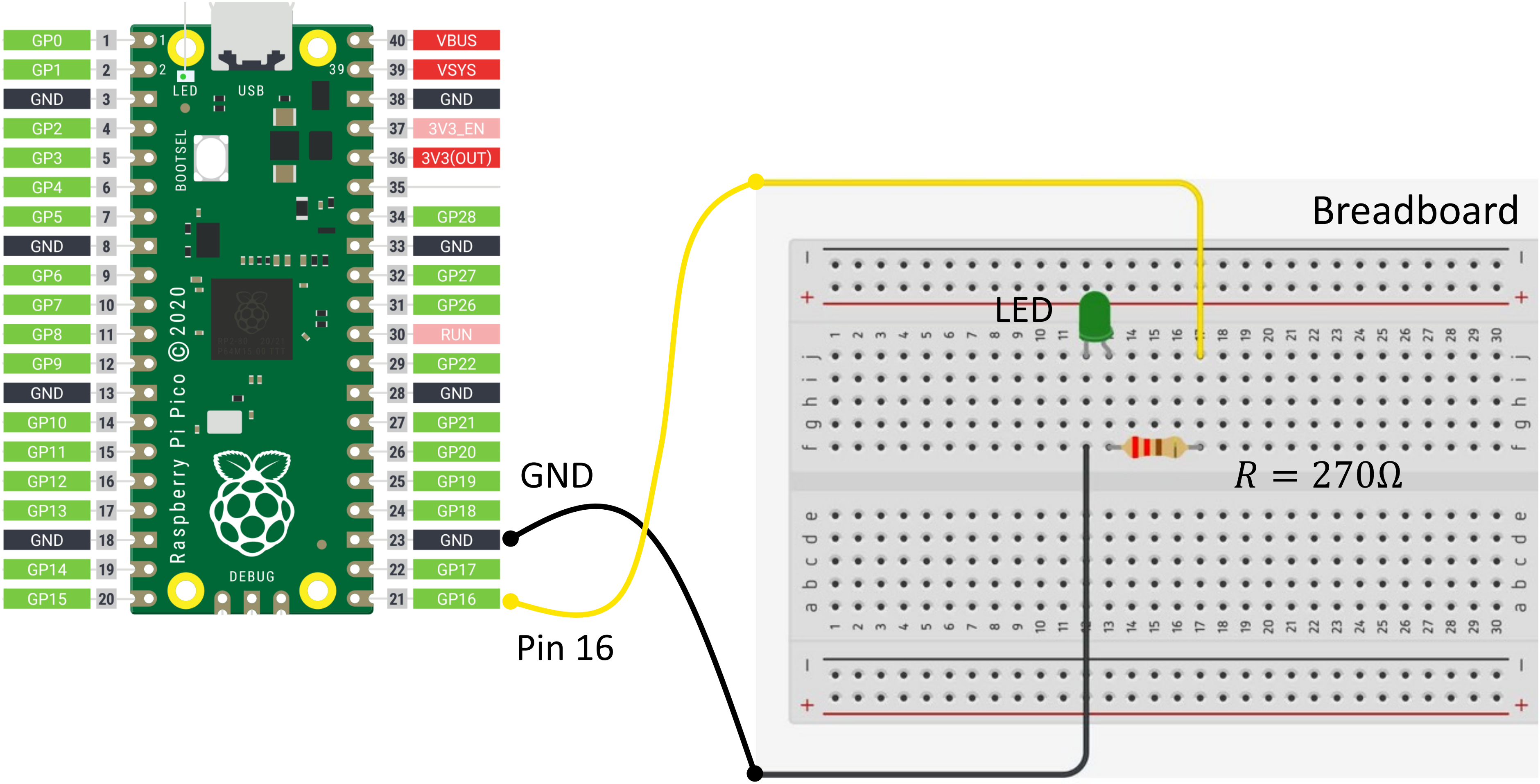
```
from machine import Pin, Timer

pin = 25
led = Pin(pin, Pin.OUT)
timer = Timer()

def blink(timer):
    led.toggle()

timer.init(freq=1, mode=Timer.PERIODIC, callback=blink)
```

Instead of a While Loop you can use the Timer module to set a timer that runs a function at regular intervals.

https://projects.raspberrypi.org/en/projects/getting-started-with-the-pico/5

Blinking external LED

Table of Contents



Breadboard

LED

GND

Pin 16

$R = 270\Omega$

Why do you need a Resistor?

If the current becomes too large, the LED will be destroyed. To prevent this to happen, we will use a Resistor to limit the amount of current in the circuit.

What should be the size of the Resistor?

A LED typically need a current like 20mA (can be found in the LED Datasheet).
We use Ohm’s Law:

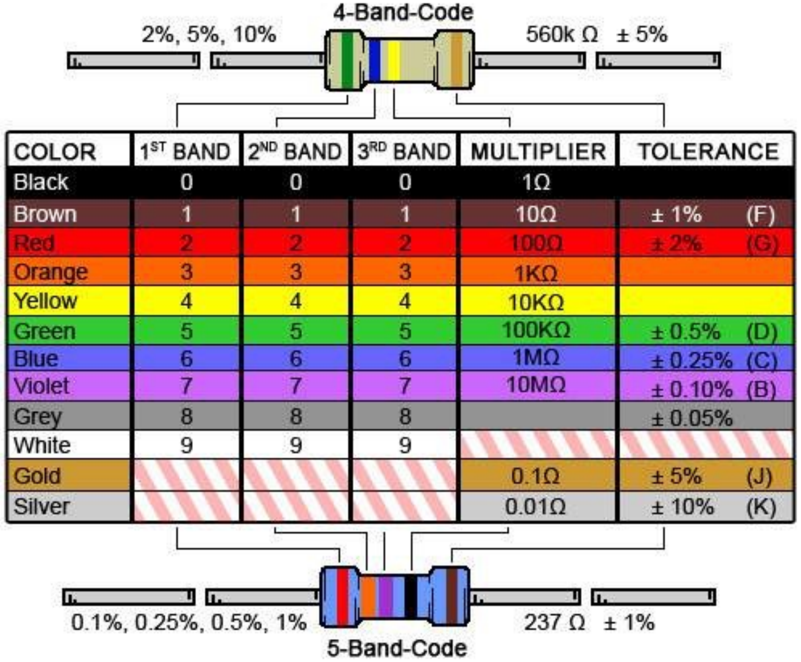
$$U = RI$$

Arduino gives $U = 5V$ and $I = 20mA$. We then get:

$$R = \frac{U}{I}$$

The Resistor needed will be $R = \frac{5V}{0.02A} = 250\Omega$. Resistors with R=250Ω is not so common, so we can use the closest Resistors we have, e.g., 270Ω

Resistor Colors and Size


4-Band-Code
2%, 5%, 10%
560k Ω ± 5%
0.1%, 0.25%, 0.5%, 1%
237 Ω ± 1%
5-Band-Code

COLOR	1ST BAND	2ND BAND	3RD BAND	MULTIPLIER	TOLERANCE
Black	0	0	0	1Ω	
Brown	1	1	1	10Ω	± 1% (F)
Red	2	2	2	100Ω	± 2% (G)
Orange	3	3	3	1KΩ	
Yellow	4	4	4	10KΩ	
Green	5	5	5	100KΩ	± 0.5% (D)
Blue	6	6	6	1MΩ	± 0.25% (C)
Violet	7	7	7	10MΩ	± 0.10% (B)
Grey	8	8	8		± 0.05%
White	9	9	9	▨	▨
Gold	▨	▨	▨	0.1Ω	± 5% (J)
Silver	▨	▨	▨	0.01Ω	± 10% (K)

You can also use a **Multimeter**

Resistor Calculator: http://www.allaboutcircuits.com/tools/resistor-color-code-calculator/

Blinking LED

```
import machine
import time

pin = 16
led = machine.Pin(pin, machine.Pin.OUT)

while True:
    led.value(1)
    time.sleep(2)
    led.value(0)
    time.sleep(2)
```

Pulse Width Modulation (PWM)

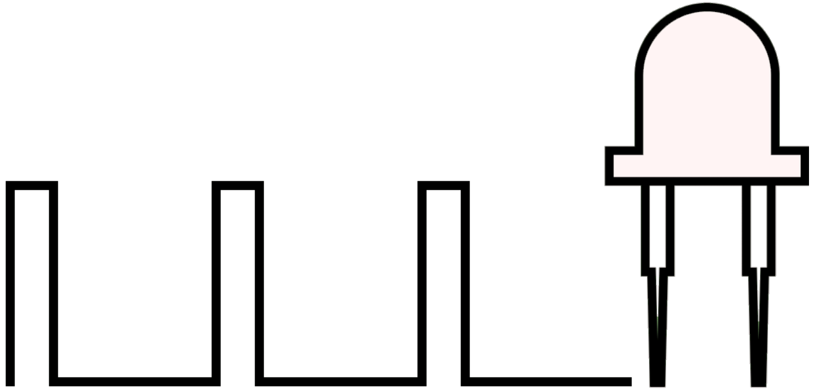
Controlling LED Brightness using PWM

- We've seen how to turn an LED on and off, but how do we control its brightness levels?
- An LED's brightness is determined by controlling the amount of current flowing through it, but that requires a lot more hardware components.
- A simple trick we can do is to flash the LED faster than the eye can see!
- By controlling the amount of time, the LED is on versus off, we can change its perceived brightness.
- This is known as *Pulse Width Modulation* (PWM).

https://learn.sparkfun.com/tutorials/python-programming-tutorial-getting-started-with-the-raspberry-pi/experiment-1-digital-input-and-output

Controlling LED Brightness using PWM

Below we see how we can use PWM to control the brightness of a LED



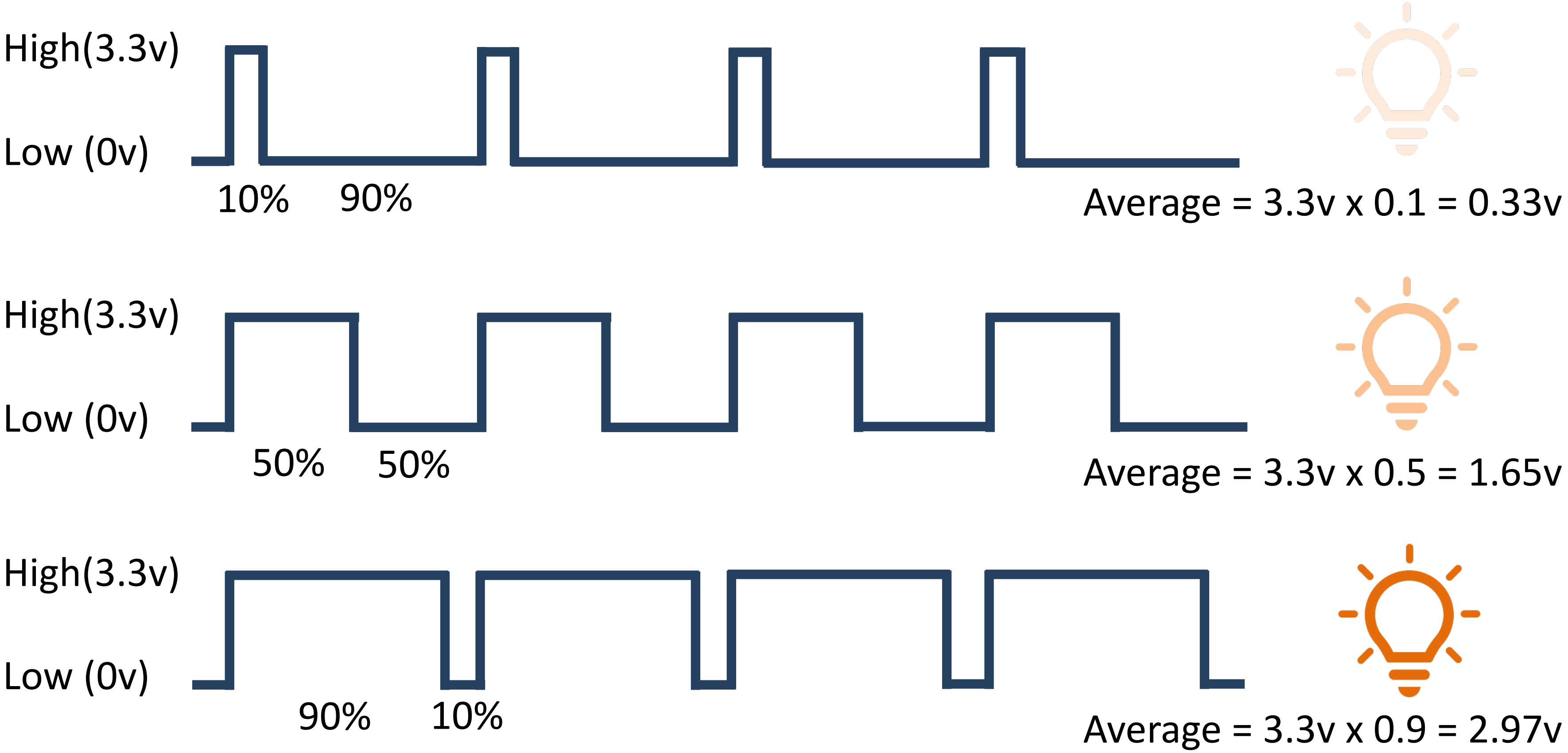
https://www.electronicwings.com/raspberry-pi/raspberry-pi-pwm-generation-using-python-and-c

PWM on Raspberry Pi Pico:

16 bit gives 2^16 = 65536 different levels, i.e., from **0 to 65535**

https://docs.micropython.org/en/latest/library/machine.PWM.html

Pulse Width Modulation (PWM)



High(3.3v)

Low (0v)

10% 90%

Average = 3.3v x 0.1 = 0.33v

High(3.3v)

Low (0v)

50% 50%

Average = 3.3v x 0.5 = 1.65v

High(3.3v)

Low (0v)

90% 10%

Average = 3.3v x 0.9 = 2.97v

PWM Example

```
from machine import Pin, PWM
from time import sleep

pin = 16
pwm = PWM(Pin(pin))
pwm.freq(1000)

N = 65535
for brightness in range(N):
    pwm.duty_u16(brightness)
    sleep(0.0001)

pwm.duty_u16(0) #Turn LED of when finished
```

PWM Example v2

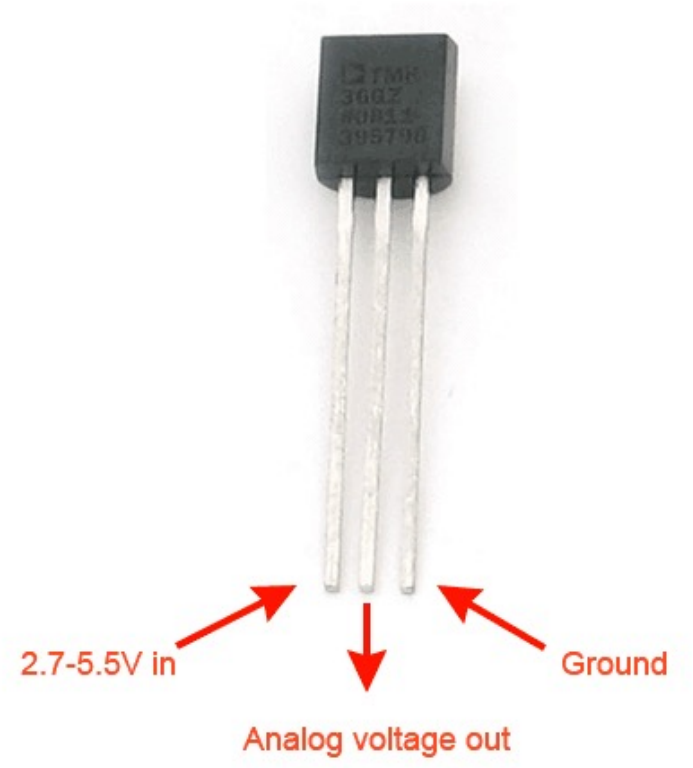
```
from machine import Pin, PWM
from time import sleep

pin = 16
pwm = PWM(Pin(pin))
pwm.freq(1000)

start = 0
step = 100
stop = 65535

for brightness in range(start, stop, step):
    pwm.duty_u16(brightness)
    sleep(0.01)

pwm.duty_u16(0)
```

TMP36 Temperature Sensor

Hans-Petter Halvorsen

Table of Contents

TMP36 Temperature Sensor

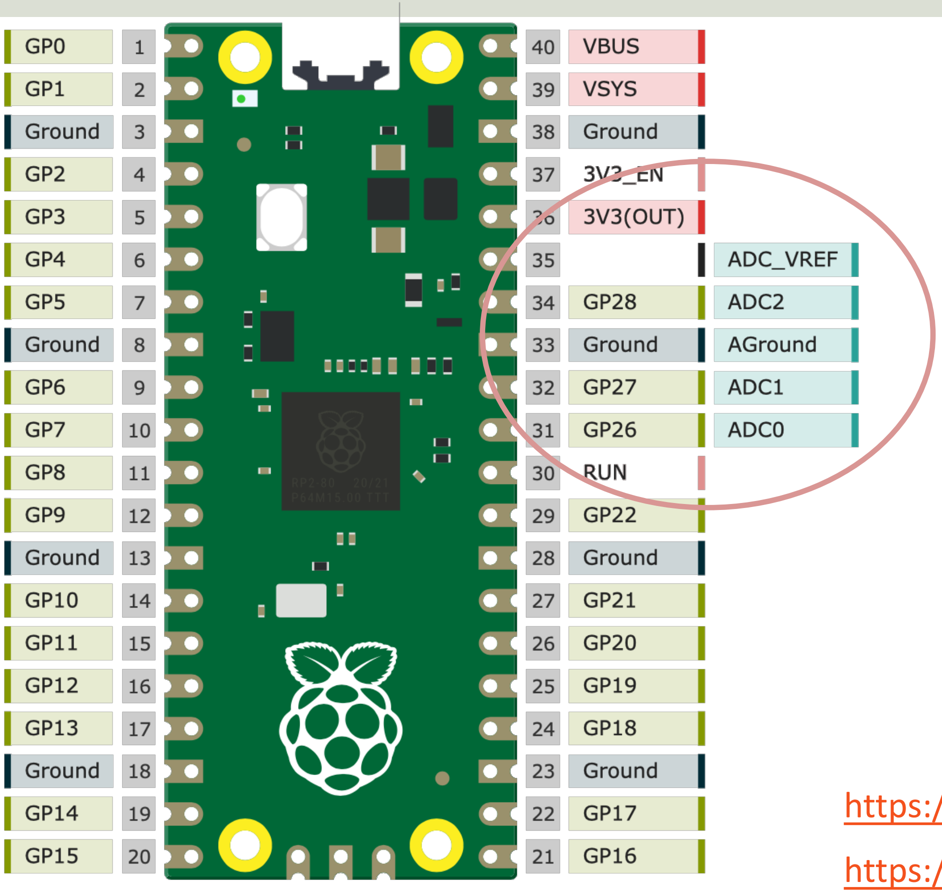


A Temperature sensor like TM36 use a solid-state technique to determine the temperature.

They use the fact as temperature increases, the voltage across a diode increases at a known rate.

https://learn.adafruit.com/tmp36-temperature-sensor

Analog Values with Pico



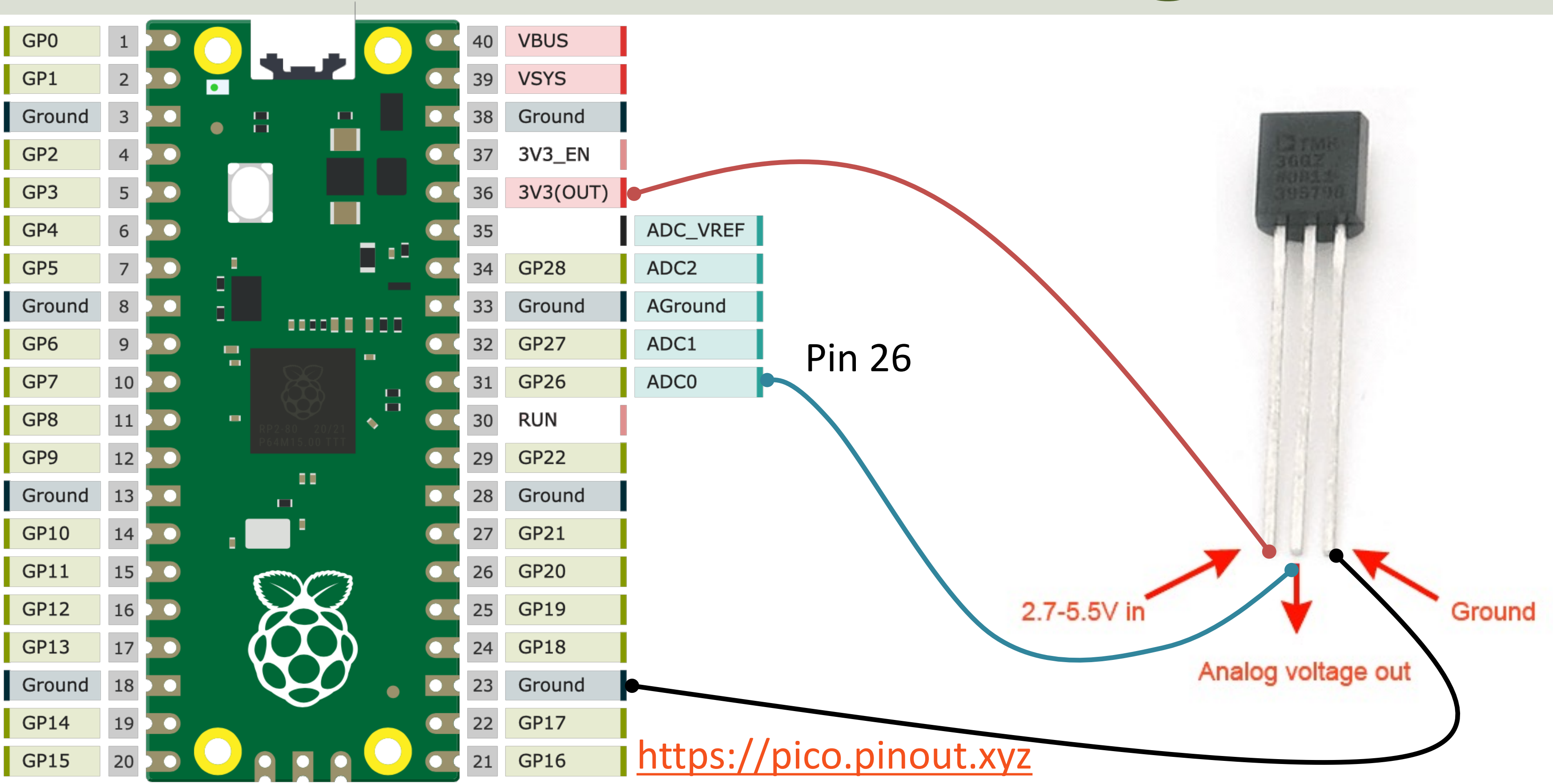
Raspberry Pi Pico has 3 Analog Inputs (ADC)

ADC 0 – Pin 26
ADC 1 – Pin 27
ADC 2 – Pin 28

https://pico.pinout.xyz

https://docs.micropython.org/en/latest/library/machine.ADC.html

TMP36 Wiring



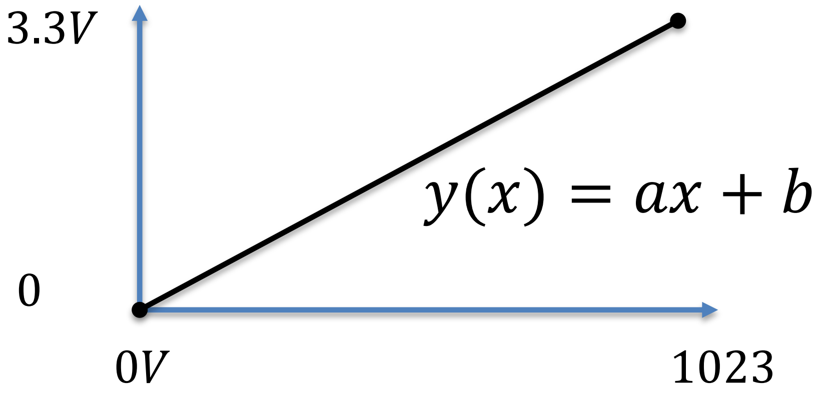
https://pico.pinout.xyz

ADC Value to Voltage Value

Analog Pins: The built-in Analog-to-Digital Converter (ADC) on Pico is 16bit, producing values from 0 to 65535.

The `read_u16()` function gives a value between 0 and 65535. It must be converted to a Voltage Signal 0 - 3.3v

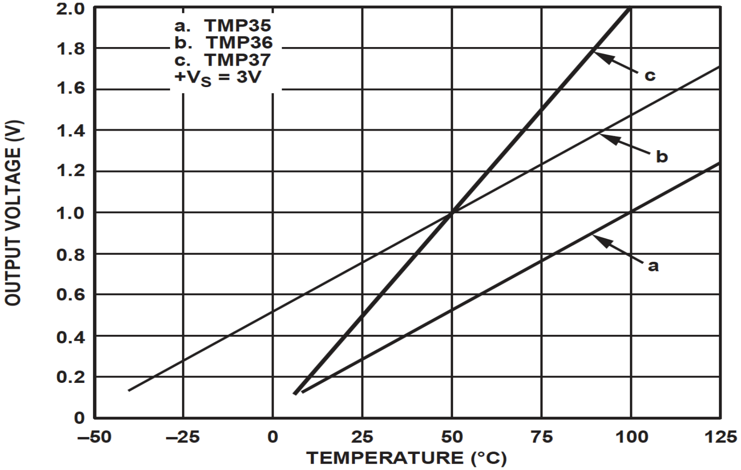
ADC = 0 -> 0v
ADC = 65535 -> 3.3v



This gives the following conversion formula:

$$y(x) = \frac{3.3}{65535}x$$

Voltage to degrees Celsius



Convert from Voltage (V) to degrees Celsius
From the **Datasheet** we have:

$$(x_1, y_1) = (0.75V, 25°C)$$
$$(x_2, y_2) = (1V, 50°C)$$

There is a linear relationship between Voltage and degrees Celsius:

$$y = ax + b$$

We can find a and b using the following known formula:

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1)$$

This gives:

$$y - 25 = \frac{50 - 25}{1 - 0.75}(x - 0.75)$$

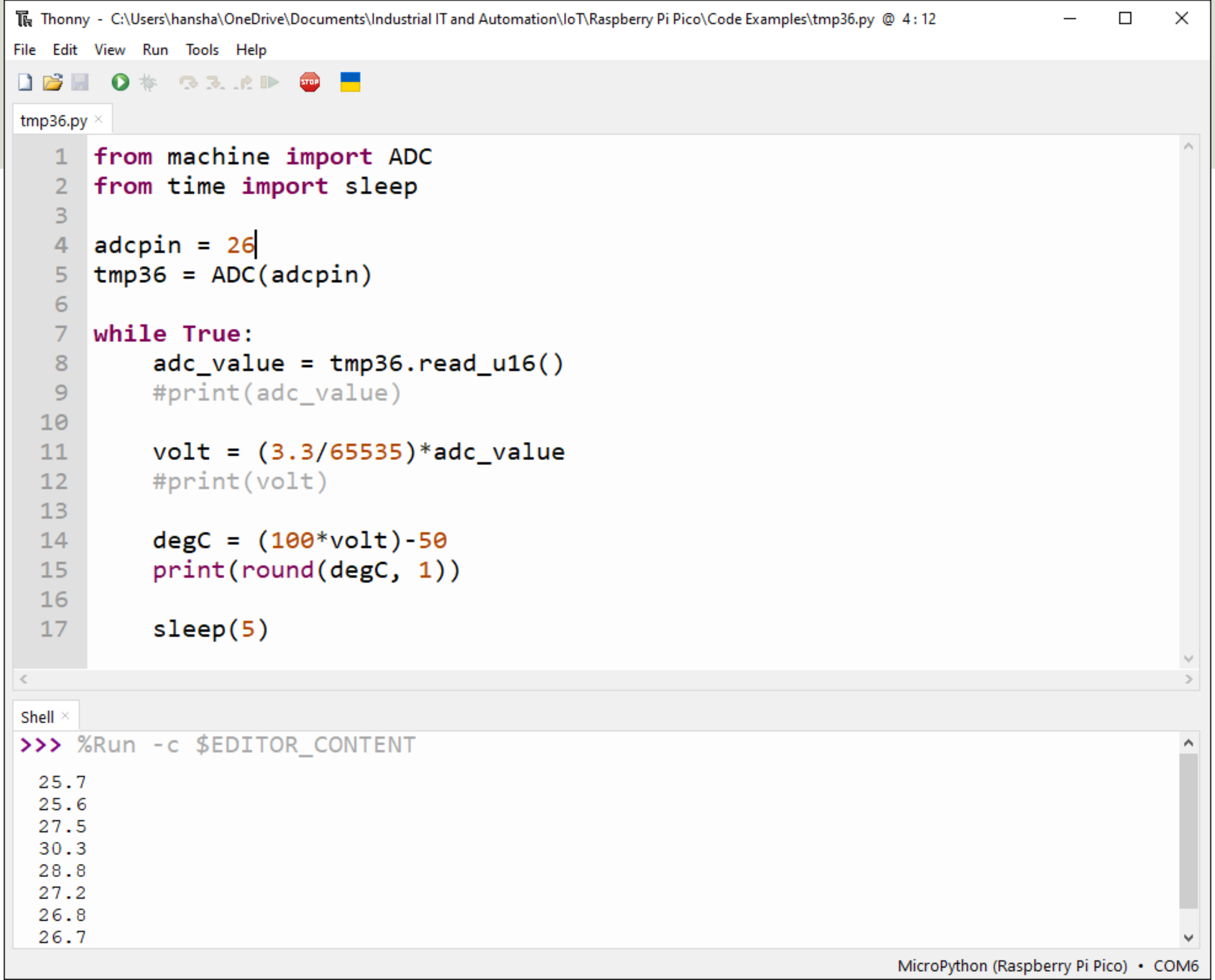
Then we get the following formula:

$$y = 100x - 50$$

Datasheet: https://cdn-learn.adafruit.com/assets/assets/000/010/131/original/TMP35_36_37.pdf

TMP36 Example

```
from machine import ADC
from time import sleep

adcpin = 26
tmp36 = ADC(adcpin)

while True:
    adc_value = tmp36.read_u16()
    volt = (3.3/65535)*adc_value
    degC = (100*volt)-50
    print(round(degC, 1))
    sleep(5)
```


Thonny - C:\Users\hansha\OneDrive\Documents\Industrial IT and Automation\IoT\Raspberry Pi Pico\Code Examples\tmp36.py @ 4 : 12
File
Edit
View
Run
Tools
Help
tmp36.py
Shell
MicroPython (Raspberry Pi Pico) • COM6

```
from machine import ADC
from time import sleep

adcpin = 26
tmp36 = ADC(adcpin)

while True:
    adc_value = tmp36.read_u16()
    #print(adc_value)

    volt = (3.3/65535)*adc_value
    #print(volt)

    degC = (100*volt)-50
    print(round(degC, 1))

    sleep(5)
```

```
>>> %Run -c $EDITOR_CONTENT

  25.7
  25.6
  27.5
  30.3
  28.8
  27.2
  26.8
  26.7
```

https://www.halvorsen.blog

Running Pico without PC

Hans-Petter Halvorsen

Table of Contents

Running Pico without PC

- If you want to run your Raspberry Pi Pico without it being attached to a computer, you can use an external USB Micro Power Supply (between 1.8V and 5.5V)
- To automatically run a MicroPython program, simply save it to the device with the name **main.py**
- Save the main.py file on the Raspberry Pi
- Unplug the connection to your PC, then attach the USB Micro Power Supply
- Then the main.py should automatically run when the Pico is starting

https://projects.raspberrypi.org/en/projects/getting-started-with-the-pico/9

Soft reboot command

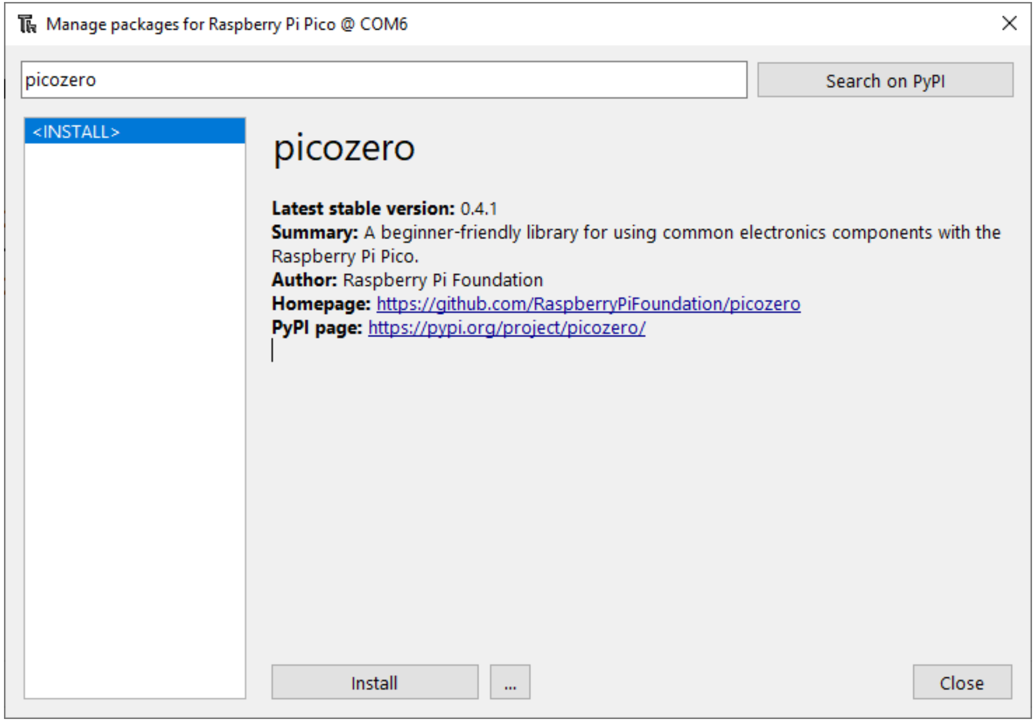
- You can also click Ctrl + D in the Shell inside the Thonny Editor to force a soft reboot command.
- In both cases the "main.py" program should start to run automatically.

https://projects.raspberrypi.org/en/projects/getting-started-with-the-pico/9

PicoZero

PicoZero

- The **picozero** Python Library is intended to be a beginner-friendly library for using common electronics components with the Raspberry Pi Pico
- **It can be used instead of the machine Library in many cases**
- You install it like an ordinary Python Library using “pip install picozero” or from the “Manage Packages” window in the Thonny editor



picozero

Latest stable version: 0.4.1
Summary: A beginner-friendly library for using common electronics components with the Raspberry Pi Pico.
Author: Raspberry Pi Foundation
Homepage: https://github.com/RaspberryPiFoundation/picozero
PyPI page: https://pypi.org/project/picozero/

https://pypi.org/project/picozero/
https://picozero.readthedocs.io
https://github.com/RaspberryPiFoundation/picozero

LED Example

```
from picozero import LED
from time import sleep

pin = 16
led = LED(pin)

led.on()
sleep(1)
led.off()
```

LED Example v2

```
from picozero import LED
from time import sleep

pin = 16
led = LED(pin)

while True:
    led.toggle()
    sleep(1)
```

Raspberry Pi Pico Resources

- Raspberry Pi Pico: https://www.raspberrypi.com/products/raspberry-pi-pico/
- Raspberry Pi Foundation: https://projects.raspberrypi.org/en/projects?hardware[]=pico
- Getting Started with Pico: https://projects.raspberrypi.org/en/projects/getting-started-with-the-pico
- MicroPython: https://docs.micropython.org/en/latest/index.html

Hans-Petter Halvorsen

University of South-Eastern Norway

www.usn.no

E-mail: hans.p.halvorsen@usn.no

Web: https://www.halvorsen.blog