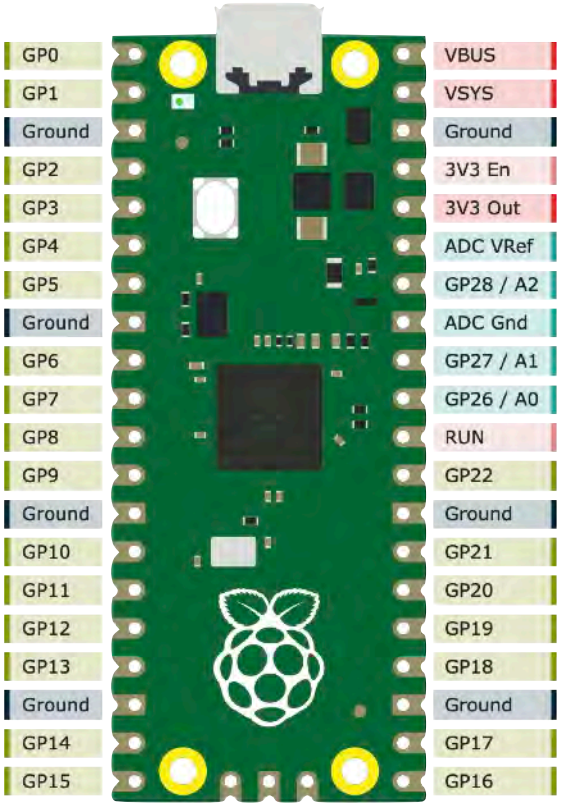
Raspberry Pi Pico

A quick guide for artists and makers



The Raspberry Pi Pico is a **microcontroller**, which means it's a very simple computer. You can connect inputs to it such as light sensors and outputs such as motors and lights.

This schematic shows the 'pinout' of the Pico. It tells you which pins you can use and what their numbers are.[1]

'3.3V' and 'Ground' for instance are the pins that supply power. Think of them as the plus (V for voltage) and minus of a battery.
All pins that have a green label can be used to connect inputs and outputs.

Installing the software

The Thonny editor allows us to program the Pico. Download it at thonny.org: select the right version for your computer by choosing Windows, Mac or Linux in the top right corner. Download the file, install it and then open the program.

With Thonny we can write software in the Python programming language. Type this line in the upper window of the Thonny screen:

```
print('Hello world!')
```

Then click the Run button:  When asked to save the file, save it as 'test.py'. Then the code will run and the result comes up in the lower part of the screen.

---

[1] A more complete diagram of the Pico's pin functions: https://pico.pinout.xyz

Your first Pico program

Thonny can also be used to run code on the Pico board instead of on your computer.

Connect the Pico to your computer via a USB cable. Select 'MicroPython (Raspberry Pi Pico)' in the bottom right-hand corner of the Thonny screen. When you run code now, it will run on the Pico.

If you don't see 'MicroPython (Raspberry Pi Pico)' in the list, then check that your Raspberry Pi Pico is plugged in. If you still don't see it in the list, then you need to install the firmware. See the page 'Adding the MicroPython firmware' at the end of this document.

Copy this code to Thonny and run it: Every time you run it (by clicking on the Run button) you will see the LED light on the board change.[2]

```
from machine import Pin
led = Pin(25, Pin.OUT)
led.toggle()
```

It is also possible to have the Pico flash the LED repeatedly without waiting for you to run the program again. Add the lines to the right to the previous code and run it. **Leave the whitespaces in** (4 spaces or 1 tab), because that matters in Python.

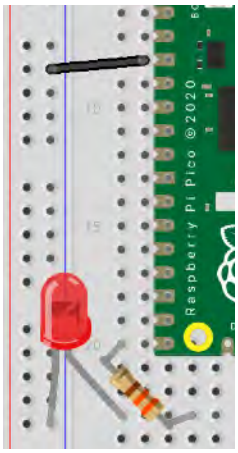
```
from time import sleep
while True:
    led.toggle()
    sleep(1)
```

Exercise

1. Can you make the LED light blink faster?
2. Can you make the LED blink different patterns?

More LEDs using the breadboard

The breadboard allows you to easily connect more LEDs to the Pico (see next page for more explanation).



Take a 330 Ohm **resistor** and put one of its legs in the breadboard next to pin GP15. The other leg can go in one of the rows below it.

Put an LED with its longer leg in the same row as the resistor, like in the schematic. Put the other leg in a hole of the blue minus rail of the breadboard. Use a wire (preferably a black one) to connect the

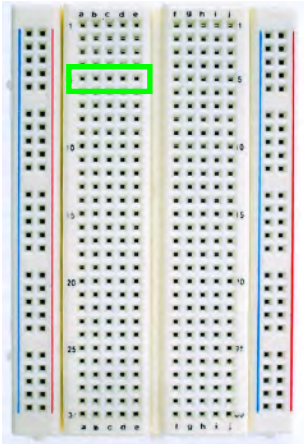
[2] When using a Pico W (it has a silvery square on it instead of the raspberry logo), use '"LED"' instead of 25.

minus rail to a ground pin of the Pico ('GND'). Change the pin number in the script from 25 to 15 and run it. Now the LED on the breadboard should be blinking!

Adding more LEDs is no problem. Each one needs to be linked to a separate GP-pin of the Pico via its own resistor, but all can be connected to the same ground rail. In the code, duplicate both lines that mentions 'led' for each LED that you add, changing the pin number and its name (so 'led' should be something like 'led2').

How a breadboard works 🍞🔌

Connecting sensors and other parts to the Pico is often easiest using a breadboard.



You can plug wires and components into the holes on the breadboard. They are connected to each other via metal strips on the inside.

Holes in the middle are connected in **rows of five** (as indicated in the photo). They are only connected to each other, so not to holes above or below them or to holes opposite of the groove.

The vertical columns, along the red and blue lines, are used to connect multiple components to a power supply such as a battery. Red is for plus and blue (or black) is minus, just like batteries have a plus and a minus. Those holes are connected to all others in the same column and not to any others.

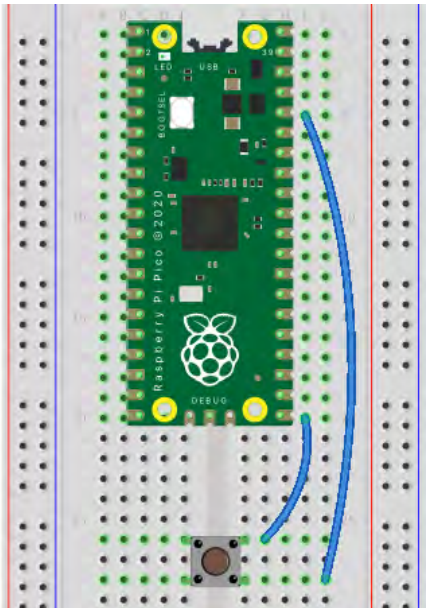
It's pretty hard to break the Pico, but **keep plus and minus separated** or things will break. 🔥

To prevent any mistakes, always stick to the right colours: the red column on the breadboard is for plus and the blue column for minus.
Also, as much as possible, use red wires to connect components to the positive side, and black wires to make connections with the negative side.

Digital sensors: a switch

We can connect sensors to the Pico to make it react to its environment. A switch is the simplest possible sensor, so let's start with that.

Connect a pushbutton with the Pico like in the diagram:



- Put the button in the breadboard, with two legs on either side of the groove in the middle.
- Connect one leg of the button with the 3.3V pin of the Pico (5th on the right hand side).
- Connect the other side of the button with pin GP16 (also see the green labels in the schematic on the first page).

Copy and paste the following code into Thonny:

```
from machine import Pin
import time

led = Pin(25, Pin.OUT) # on a Pico W, write '"LED"' instead of 25
button = Pin(16, Pin.IN, Pin.PULL_DOWN)

while True:
    if button.value():
        led.toggle()
        time.sleep(0.5)
```

 Push the play button in Thonny to run the script.

 Now try pushing the pushbutton on the breadboard!

You should now have your first working digital sensor. When you push the button, the LED on the Pico turns on. So already you are combining an input and output!

Learn how this code works and how you can adapt it on the next two pages.

If-statements

To have Pico do something only when something changes, or when some condition is fulfilled, we can use **if-statements**. This is what they look like in MicroPython:

```
if sensor_value > 30:
    # do something here
```

You can think of if-statements as questions. Sometimes the question is ‘is the button being pressed?’ Or, as in the example above: ‘is the value coming from a sensor larger than 30?’

Only when the answer to such a question is ‘yes, that is true’, then Pico will run the code that is on the next line. It can also be many more lines: all lines that have the same number of spaces in front of them belong to the same if-statement.
The script on the previous page for instance has an if-statement looking like this:

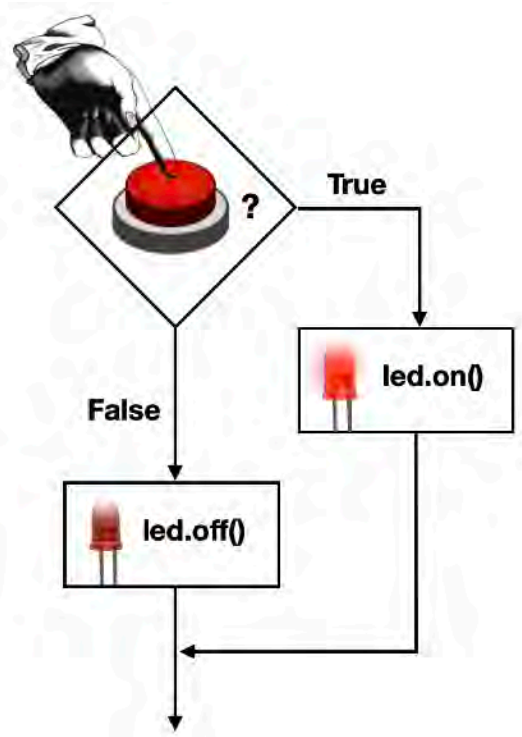
```
if button.value():
    led.toggle()
    time.sleep(0.5)
```

So here the question is: ‘is the button being pressed?’ If the answer is ‘yes’ (‘True’), then the LED is toggled.

You can also change it so that the LED turns on when the button is being pressed and off when it isn’t:

```
if button.value():
    led.on()
else:
    led.off()
```

This is an addition to the if-statement: ‘else’ allows us to add instructions in case the answer to the question is ‘no’ or ‘False’.



Exercise
With the code just mentioned, the LED turns on when you press the button. Can you change it so that the LED *turns off* when you press the button?

Other parts of the script: libraries, comments, loops

The program for the digital sensor first imports a couple of libraries, pre-made pieces of software. In this case we need them to help Pico understand where the button and the LED are connected and what time is:

```
from machine import Pin
import time
```

Then we can tell Pico that we want to use the built-in LED and a button that is connected to pin GP16. We tell it that the LED is an output and the button is an input:

```
led = Pin(25, Pin.OUT)
button = Pin(16, Pin.IN, Pin.PULL_DOWN)
```

The next part has **comments** in it, the text in red with a # in front of it. Those comments are often used by programmers to explain what the code does. The comments are there for you and are ignored by the Pico.

As explained in these comments, the Pico checks if the button is being pressed with an **if-statement** (see previous page). When you push the button that is connected to the Pico, button.value() will return 'True'. When you let go of the button, button.value() will say 'False'.[3]

So if the button is pressed and button.value() returns 'True', the Pico turns the LED on or off and waits half a second before starting over:

```
while True:
    if button.value():   # check if the button is being pressed
        led.toggle()     # switch LED on or off
        time.sleep(0.5)  # wait for half a second
```

All of this is happening inside a loop, which means that the status of the button is checked continuously. This **while-loop** repeats whatever is inside it (indicated by the indentation) for as long as you don't hit Thonny's stop button.

[3] That is why a button is an example of a digital sensor: there are only two possible states. Other examples of digital sensors are motion sensors (like PIR sensors that automatically turn on the light in bathrooms) and 'electric eyes' that make sure elevator doors do not close when someone is between them.

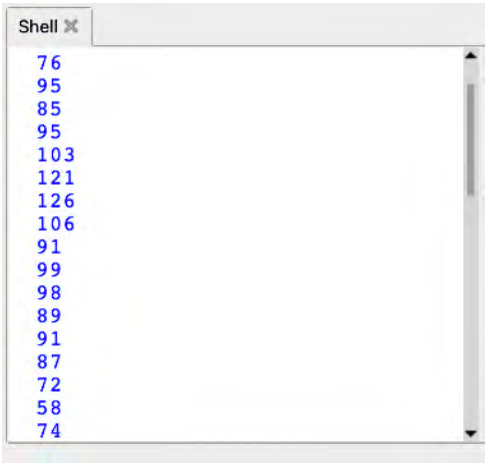
Shell and plotter

Imagine something goes wrong, like the LED not turning on when you expect it to. In cases like that, it is incredibly useful to have the Pico tell us what is happening. That helps us find out what the problem is.

To do that, add the following line to the previous script, directly below the line where it checks the button value. (Make sure it has the exact same number of spaces in front of it as the line before it.)

```
print('This shows up in the shell')
```

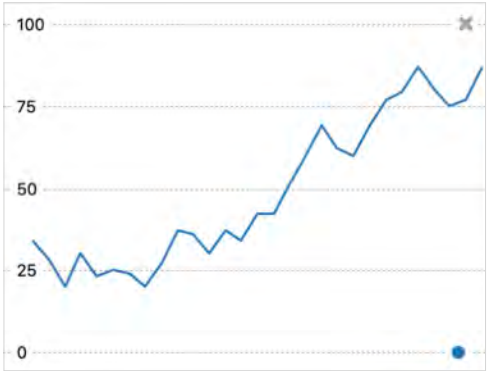
If you run the script now, then the Pico will print a message to the shell window[4] when the button is pressed.



When you don't see the numbers changing, then you know that the button is either not working or not connected correctly. So that is a great help for finding out if the Pico is doing what we were expecting, and if not, where the problem is.[5]

Plotter

To show values in a more visual way, Thonny also has a plotter built into it.



You can try it with the analogue sensors on the next page. In the window of the plotter you should see a graph of the changing values!

If you want to see data from multiple sensors as separate lines in the plotter, then send their values to the computer on one line, like so:

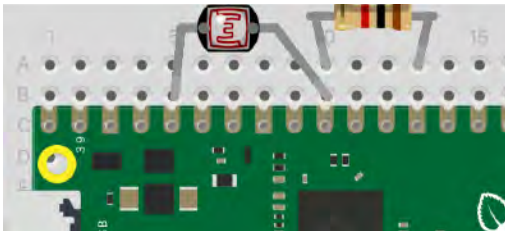
```
print(value1, value2)
```

[4] Don't see the shell window? Make it visible via Thonny's 'View' menu.

[5] We can also use the print-function to send sensor data from the Pico to software on the computer other than Thonny. It allows you to use sensor data to make interactive animations or music for instance.

Analogue sensors

Unlike digital sensors, that only know 1 and 0 (on or off, true or false), analogue sensors can give in-between values.
One example is the light sensor, which can sense if it's dark or very light but also all gradient values in between.[6]

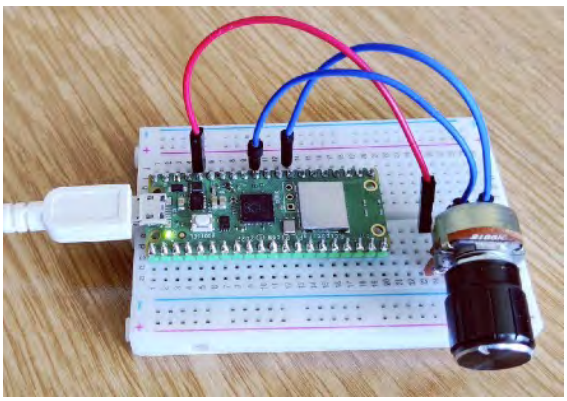


To read those light values from the Pico, connect a light sensor to 3.3V (5th pin from the top on the right-hand side) and to pin GP26 (10th pin).
Then connect a **resistor** with a value between 500 Ohm and 10KiloOhm to GP26 and to GND.

```
import machine, time
ldr = machine.ADC(26)

while True:
    print(ldr.read_u16())
    time.sleep_ms(70)
```

Now run this script.
Look at the values coming from the analogue sensor using the plotter!

Another example of an analogue sensor is the **potentiometer**. It is best known as the volume knob on speakers and amplifiers.



Connect it to the same pins on the Pico as before: middle pin to pin GP26, one of the outer pins to 3.3V and the other to GND.

You can also use different outputs instead of just the plotter. You could for instance make the LED blink faster or slower by turning the knob!

```
import machine, time

potentiometer = machine.ADC(26)
led = machine.Pin(25, machine.Pin.OUT)

while True:
    # Divide potmeter value by all possible
    # values to get a value between 0 and 1.
    value = potentiometer.read_u16()/65535
    print(value)
    led.toggle()
    time.sleep(value)
```

[6] It is actually a light dependant resistor (LDR).

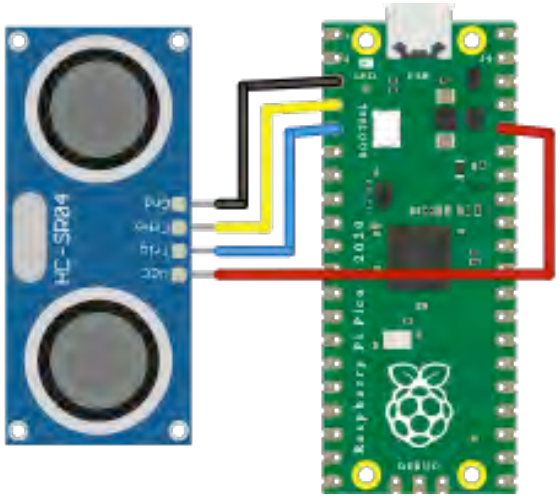
Distance sensors 

If you want to know how far away something or someone is, use an ultrasonic distance sensor. It works by sending an ultrasonic pulse, and then measuring how long the echo takes to come back. By dividing that time by the speed of sound, the Pico can calculate how far away an object is.

Those calculations are not hard, but it's even easier using the picozero library.[7] Install that if necessary (explained in 'Installing libraries' near the end of this guide).

Connect the sensor to the Pico, perhaps via a breadboard, like so:

- 'VCC' pin on the sensor to 3V3 on the Pico[8]
- 'GND' on the sensor to GND on the Pico
- 'Echo' to pin GP2 on the Pico
- 'Trig' to pin GP3 on the Pico.



Run this script on the Pico and the shell and plotter will show the distances the sensor is sensing:

```
from picozero import DistanceSensor
from time import sleep

ds = DistanceSensor(echo=2, trigger=3, max_distance=3)

while True:
    print(ds.distance * 100) # multiply by 100 to get centimeters
    sleep(0.1)
```

The function ds.distance gives the distance in meters. About 4 or 5 meters is the maximum you can get with this sensor.[9]

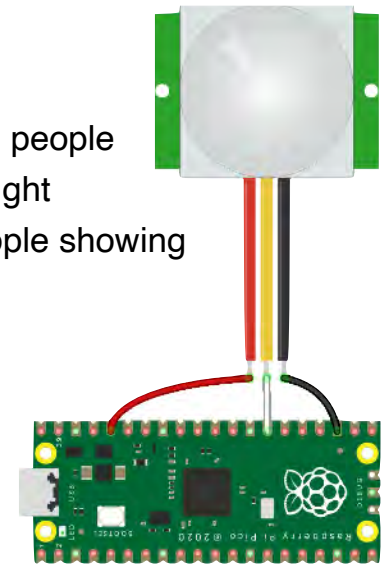
---

[7] More documentation for the PicoZero library can be found here: https://picozero.readthedocs.io

[8] Some distance sensors only work well at 5 Volts. If you see erratic values coming in, try a different model sensor or use the sensor at 5V and use something called a voltage divider on the echo output (Google it :-).

[9] Ultrasonic distance sensors measure distances in a narrow beam of 15 degrees. If you want to scan a wider area, you can combine multiple ultrasonic sensors next to eachother, or have one sensor rotate on a little motor to make a sweeping radar. If you only want to know if a person shows up in the room, then use a PIR sensor (see next page).

Sensing people: motion sensor 👫



Distance sensors are not really good at sensing if there are people around. They only measure distance in a fairly narrow, straight line. If you want to make something that has to react to people showing up in a gallery, then you need a PIR motion sensor.[10]

Make these connections between the sensor and the Pico:
(Lift up the white dome of the sensor to see what each wire is, or see photo below.)

PIR	Pico
VCC	3.3V
OUT	GP28
GND	GND

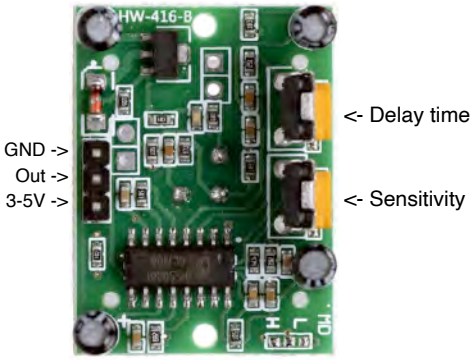
You can then use the following code to see when the sensor detects movement:

```
import machine, utime
pir = machine.Pin(28, machine.Pin.IN, machine.Pin.PULL_DOWN)

def pir_handler(pin):
   utime.sleep_ms(100)
   if pin.value():
      print("Motion detected!")

pir.irq(trigger=machine.Pin.IRQ_RISING, handler=pir_handler)
```

That last line creates an **interrupt**. It prepares the Pico to do something when a certain event occurs—in this case, the signal from the sensor changing. The 'handler' part points to the function 'pir_handler' above it: that will be run when the trigger happens. After setting up an interrupt the code can go on to do other things.

The PIR sensor's **field of view** is about a third of a circle wide. It is often useful to limit that by covering the white dome with tape or putting it inside an empty cup.

Using the screws you can set the delay after a detection (0.3 at full left to 300 seconds at full right) and the sensitivity (3 to 7 meters).[11]



[10] PIR stands for 'passive infrared'. That means the sensor doesn't send out infrared light (like a heart rate sensor) but that it looks for changes in infrared light intensity. People give off infrared light (as body heat).

[11] More on the PIR sensor here: https://osoyoo.com/2017/07/27/pir-motion-sensor/

Variables

Variables are words that we use to make Pico remember things. That's often useful, for instance when we want to keep track of how often something has happened.

A variable is defined like so:

```
count = 100
```

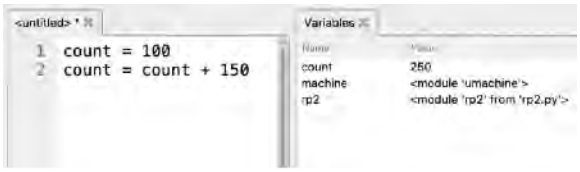
In this case we name the variable 'count', but it could just as well be 'x', or 'sensorValue'. Usually we choose a name that makes really clear what information is stored in that variable.
Then we tell the variable what its initial value is (in this case 100).

Once a variable is defined like that, you can start using it in calculations.
If we do this…

```
count = count + 150
```

… then 'count' now has a value of 250.

If you open the 'Variables' window via the menu 'View', you will see all variables in use in a program.



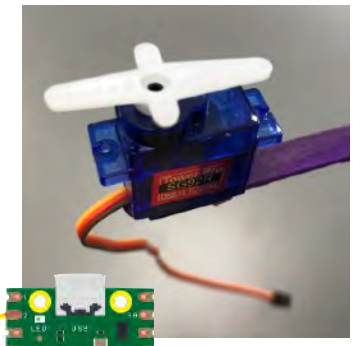
Exercise

Variables allow you to keep track of things, for instance how often a button was pressed. You can adapt the script on the page about digital sensors to do that. First try it yourself, and then read the solution in the footnote.[12]

[12] Add a variable at the top of the script. Add 1 to the variable, right where the script does something when the button is pressed. Add a 'print()' statement to that same spot to write the value of your variable to the shell. Also see what happens when you remove 'sleep(0.5)'.

Making things move with servo motors

The motor seen on the right is called a micro servo. This type of motor is found in lots of toys and robots. They are usually limited to 180 degrees rotation.[13]



Most small servo motors have three wires:

- brown is ground (GND)
- red is plus (so voltage/3.3V)
- yellow is signal, used to tell the servo where to turn. Connect it to pin GP1 on the Pico.

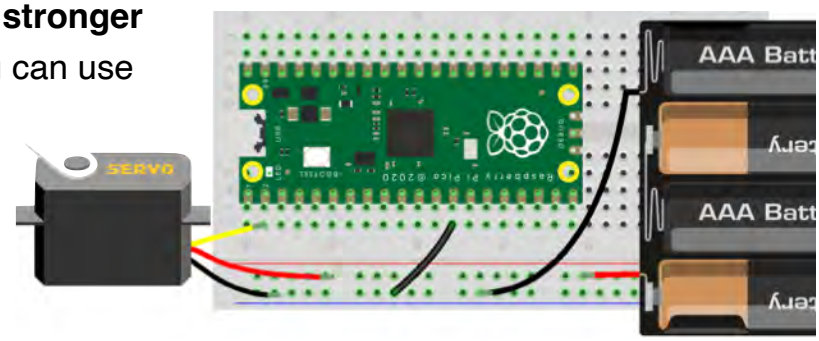
Move the servo between its minimum and maximum position:

```
from picozero import Servo
servo = Servo(1)
servo.pulse()
```

Use a **for-loop** to make the servo turn from its minimum to maximum position in 100 steps:

```
from picozero import Servo
from time import sleep
servo = Servo(1)

for i in range(0, 100):
    servo.value = i / 100
    sleep(0.1)

servo.off()
```

There are **bigger and (much) stronger servos** than micro servos. You can use those with Pico as well, but to give them enough power you need an external power supply, like so:

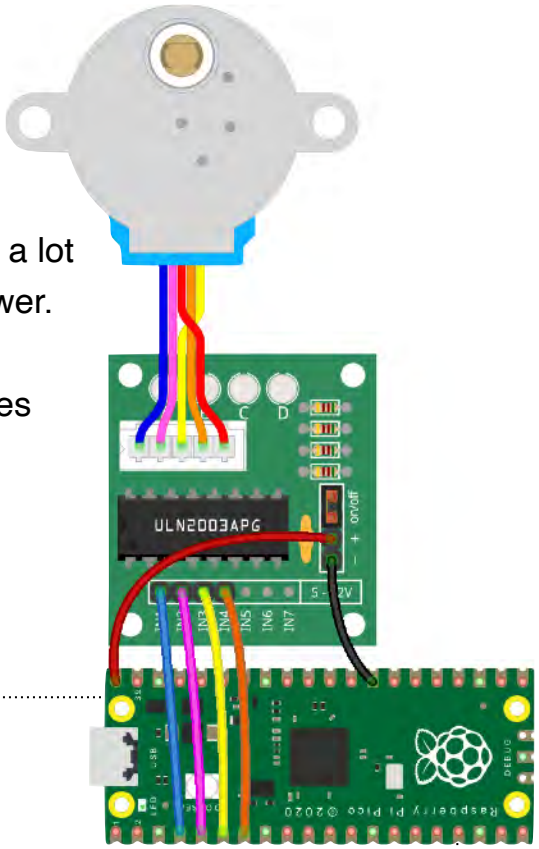


Exercise

1. Move the servo to different positions a couple of times, and have it wait a while in between. You only need a few servo.value()'s and sleep()'s to do it.
2. Also try servo.min(), servo.mid() and servo.max() to move the servo to its minimum, middle and maximum positions.

[13] There are actually also 'continuous' servos; they can keep rotating continually. Continuous servos translate the position you give them into a direction and a speed. '0' for instance is full speed to the left, '90' is standing still and '110' means 'turn slowly to the right'. If you want to keep track of how far a continuous servo has turned, you need to keep track of that using an 'encoder wheel'.

Stepper motor



Stepper motors are not necessarily fast or strong, but they can be much more precise than servos.

The small stepper motor we use here[14] does not draw a lot of current, so you can use the Pico's VBUS pin for power.

The 'moveSteps' function in the following program takes three arguments: number of steps, wait time in milliseconds and direction (clockwise or not).
2048 steps is one complete turn, but you can move it many times more—or just a few tiny steps.

```
from machine import Pin
from time import sleep_ms

IN1 = Pin(2,Pin.OUT)
IN2 = Pin(3,Pin.OUT)
IN3 = Pin(4,Pin.OUT)
IN4 = Pin(5,Pin.OUT)
pins = [IN1, IN2, IN3, IN4]

sequence = [[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]]

def moveSteps(steps, wait, clockwise):
    order = [0,1,2,3] if clockwise else [3,2,1,0]
    for s in range(steps):
        for i in range(4):
            pins[i].value(sequence[order[s%4]][i])
            sleep_ms(wait)

moveSteps(2048, 1, True)  # change these values
```

Exercise

- Experiment a little with running moveSteps a few times with different values.
- Add two buttons to make the motor turn in different directions.
- You could also add a sensor and then attach a needle or pointer to the stepper to indicate for instance the temperature or sound level in the room.

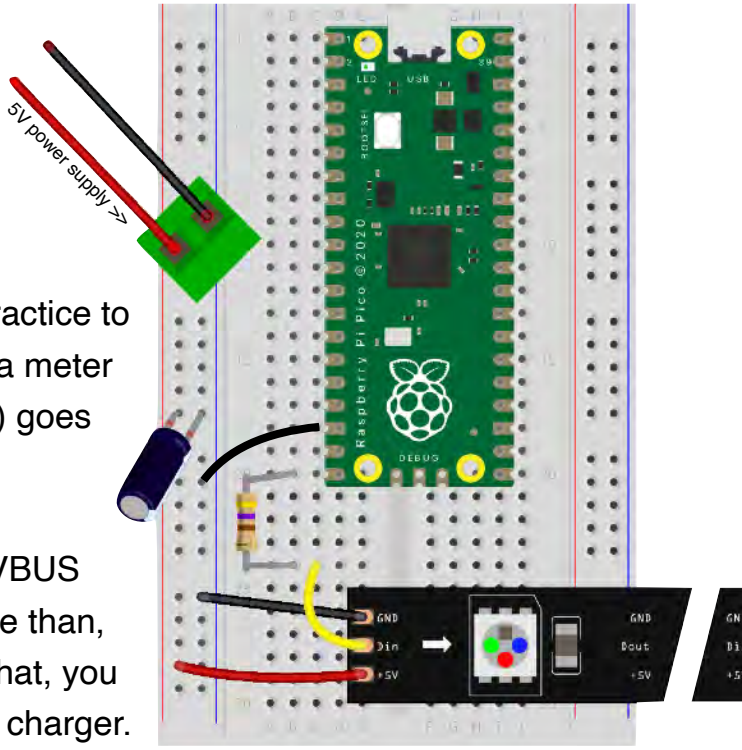
---

[14] It's the 28BYJ-48 stepper motor with the ULN2003 motor driver, a very common combination. This motor can do about 15 rotations per minute (RPM). See https://microcontrollerslab.com/28byj-48-stepper-motor-raspberry-pi-pico-micropython/ for more information on the 28BYJ-48.

Lighting things up with LED strips

LEDs[15] are used in everything from bike lights to televisions. We already used single LEDs with the Pico; here we will connect a longer strip of addressable LEDs.

Wire up the LED strip as in the diagram. 'DIN' stands for 'data in'. That is where the instructions from the Pico are going in,[16] via a resistor (between 300-500 Ohm) to GP15.



The black cilinder is a **capacitor**; it is good practice to have one when using LED strips longer than a meter or two. Its negative side (with the white stripe) goes into the minus rail of the breadboard.

The **power** for the LEDs can come from the VBUS pin of the Pico, but not if you want to use more than, say, 20 LEDs. If the LED strip is longer than that, you have to get power in from a powerbank or 5V charger.

Software

To make individual LEDs each light up with different colours we'll use a library that can help us talk to WS2812b or SK6812 (Neopixel) LED strips. Download the library here: https://github.com/blaz-r/pi_pico_neopixel
Click on the green button 'Code' and then choose 'Download ZIP'.
Unpack the zip-file, open 'neopixel.py' in Thonny and then follow the steps on the page 'Initialization' on that website.

Run the examples to learn how to use the library. You may need to change the number of pixels (here 10), the GP pin number and colour order ('GRB' or 'RGBW' instead of 'RGB').

```
from neopixel import Neopixel

pixels = Neopixel(10, 0, 15, "RGB")
pixels.fill((20, 5, 0))
pixels.show()
```

Also see 'Documentation' for all possible ways to light up the LEDs.

---

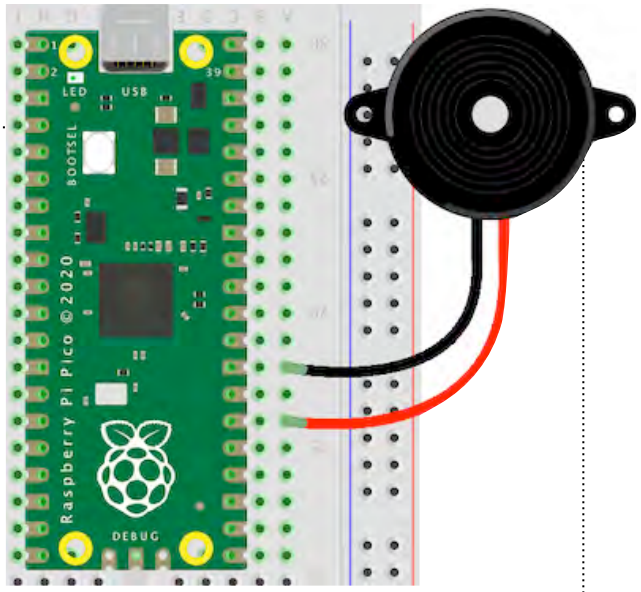
[15] 'LED' stands for light emitting diode. A diode is an electronic part that only allows current to flow in one direction, and LEDs are a type of diode that incredibly usefully produce light when conducting electricity.

[16] The data should go in at the beginning of the strip, as indicated by an arrow on the strip. If you cut the strip into smaller sections and solder new wires onto the strip, make sure you do so at the correct end.

Making beepy noises with buzzers

Buzzers are tiny speakers. They are very commonly used to make beeping noises, in anything from desktop computers to product scanners in supermarkets.[17]

Connect the buzzer to a ground pin and pin 20 of the Pico. Then run this code:




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

up = True
frequency = 88

buzzer = PWM(Pin(20))
buzzer.duty_u16(1000)

try:
    while True:
        buzzer.freq(frequency)
        sleep(0.1)
        if up:
            frequency = int(frequency * 1.1)
        else:
            frequency = int(frequency / 1.4)

        if frequency > 880:
            up = False
        elif frequency < 88:
            up = True
            frequency = 88
except KeyboardInterrupt:
    buzzer.duty_u16(0000)  # turn the buzzer off when the program stops
```

Exercise

Try to play actual tunes. You can find music pieces for buzzer to play with the Pico here: https://github.com/twisst/pico-songs

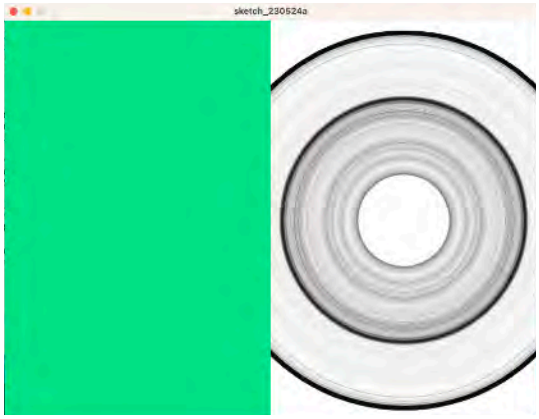
The sound is not going to be loud, because the Pico can only give the buzzer 3.3 Volts. You can use an NPN transistor to be able to use it with the 5 Volts from the VBUS pin[18], or replace the buzzer with an amplified speaker.

---

[17] Buzzers (also called piezo buzzers) actually come in two flavors: active and passive. To play melodies you need the passive kind. Active buzzers are the things that make the 'beep' sound in electronic devices. Those will try to produce the same pitch 'beep' no matter what tone you ask them to play. To make active buzzers beep, just set their pin to ‘HIGH’.

[18] See https://www.coderdojotc.org/micropython/motors/02-transistor/ for a schematic.

Using Pico with Processing



The Pico sends data to your computer via the USB cable, using print(). That data can be used as input for many different programs[19] to make interactive music, games and animations.
Let's try it with a sensor and Processing.

Set up the light sensor from the earlier page on analogue sensors. Save the first script on that page to the Pico as a file called '**main.py**'.[20] That will make it run whenever the Pico starts, without you having to press the run button in Thonny.

Download the Processing-program from Processing.org.
Once installed, use it to run the code to the right—you should see an animation on your screen that reacts to changing values from the light sensor.

If nothing happens, you probably need to change the **port number** in the code. The script prints a list of ports on your computer to the shell window at the bottom. Choose the number of the port the Pico is connected to.[21]

Also make sure Thonny is no longer connected to the Pico. If it is, Processing will show a 'Port busy' error. In that case click 'Disconnect' in Thonny's 'Run' menu.

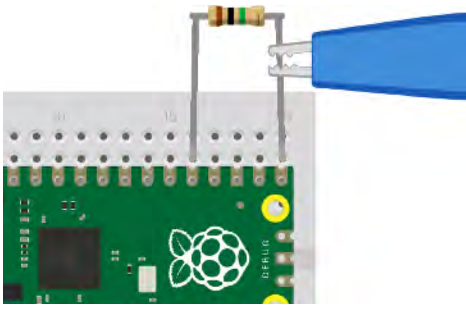
```
import processing.serial.*;
Serial port;
void setup() {
 size(800, 600);
 printArray(Serial.list());

 // change port number here:
 port = new Serial(this, Serial.list()[1]);

 background(255, 10);
 colorMode(HSB, 65535);
}

void draw() {
 while (port.available() > 0) {
 String s = port.readStringUntil(10);
 if (s != null) {
  try {
   float intVal=Integer.parseInt(s.trim());
   println(intVal);
   fill(intVal/2, intVal, intVal);
   noStroke();
   rect(0, 0, width/2, height);
   noFill();
   stroke(0, 3000);
   circle(600, 300, (intVal/100));
  } catch (NumberFormatException npe) {
   // do nothing when incoming data
   // is not a whole number.
  }
 }
 }
}
```

[19] Programs such as TouchDesigner, Max/MSP, Unity, Ableton and Processing.

[20] To do that: open the 'File' menu, choose 'Save as…' and then 'Raspberry Pi Pico'. Write 'main.py' as the file name and click on 'Ok'. Then choose 'Send EOF / Soft reboot' from the 'Run' menu.

[21] It's something like /dev/ttyACM0 or /dev/cu.usbmodem14201 but it changes and depends your OS.

Making things touch sensitive 🖐️📈



The Pico can act as a capacitive touch sensor: it can sense if someone is touching some conductive material (a metal sculpture, a pencil drawing, a glas of water, plants).
Connect GP19 to GP16 via a 1 MegaOhm resistor. Clamp an alligator cable to the resistor leg that is connected to GP16. Clamp the other end of that cable to the thing you want to make touch sensitive.[22]
Run this code and you should see the result via the plotter and Pico's onboard LED:

```
from machine import Pin
from utime import sleep_us, ticks_us, sleep

led = Pin(25, Pin.OUT)
send = Pin(19, Pin.OUT)

samples = 500 # how many samples to take (choose a value between 100 and 600)
threshold = 25000 # a value that is clearly higher than when not touching.

def readSensor(): # returns the time it takes the sensor to receive the pulse

    send.value(0)
    sensor = Pin(16, Pin.IN) # set sensor to input
    start = ticks_us() # start signal
    send.value(1)
    while sensor.value() < 1: # wait for the sensor to read 'true'
        pass
    end = ticks_us() # record how long it took to receive the signal
    send.value(0)

    # reset the sensor pin to get more reliable readings
    sensor = Pin(16,Pin.OUT) # set sensor to output
    sensor.value(1) # pull sensor pin high
    sleep_us(10)
    sensor.value(0) # pull sensor pin low
    sensor = Pin(16,Pin.IN) # set sensor to input again

    return(end-start)

while True:
    total = 0
    for _ in range(samples):
        total = total + readSensor() # add up a series of sensor readings

    if total > threshold: # if sensor value is higher than the threshold...
        led.on() # turn the LED on
    else:
        led.off()

    print(threshold, total)
    sleep(0.02)
```

[22] For best results: connect your laptop to wall power, put the sensitive object on a non-conductive surface.

A stopwatch for Pico programs: Timer 😴 ⏱️

If you want Pico to do two things at the same time, like blinking an LED and meanwhile keeping an eye on if a button is pressed, then it is important *not* to use the sleep() function.

sleep() makes the Pico wait for a while, and in the meantime nothing else can get done.

If we want Pico to do more than one thing at a time, we need MicroPython's **Timer** library. Timer keeps track of how much time has passed to do tasks at regular intervals.

An example script is shown below. One timer turns the on-board LED on or off every 50 microseconds. A second timer prints the time since the script started and does that every second.

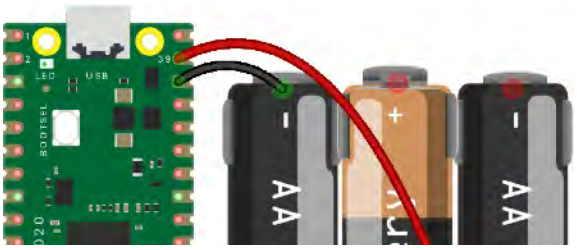
```
from machine import Pin, Timer
import time

led= Pin(25, Pin.OUT)
start = time.time()

def switchLED(self):      # function to toggle LED
    led.toggle()

def runtime(self):        # function to print the time
    print("seconds since started: ", time.time() - start)

timer=Timer(-1)           # initialize a timer
timer.init(period=50, callback=switchLED)   # start the timer

timer2=Timer(-1)          # initialize another timer
timer2.init(period=1000, callback=runtime)   # start timer
```

To have a program do something only when a certain event occurs, like the press of a button, add something called an **interrupt**. See the page on PIR sensors for an example of how that works.

Running Pico without a computer

The Pico can run independently from your computer. If you save your program on the Pico and keep it powered, you don't need to keep your computer around.

If you save your script as **main.py** on the Pico, it automatically runs when power is connected.



One way to **power** the Pico is using an adapter or a powerbank via the micro-USB connector.[23] The other way is to supply between 2 and 5 Volts to the VSYS pin. You can use a battery pack (one with 3x AA batteries will put out 4.5 Volts). There are also add-on boards on the market that will let you connect a LiPo battery to the Pico.[24]

If you want to keep sending data to a computer: with a Pico W you can use **Wi-Fi or Bluetooth** to set up communication.

Installing libraries

You really do not need to program everything yourself. A *lot* of software has already been written by people who love to share their work with you.[25] It often comes as a bundle of a few files, called a module or library. MicroPython has a number of modules built-in, but sometimes you will need to install them yourself.

To install libraries into Thonny:

- go to the menu 'Tools' on the top of your screen
- select 'Manage Packages'
- type the name of the library in the search field and
- click on 'Search on PyPI'.
- Then click on the search result, then 'Install' and 'Close' once it's done.

---

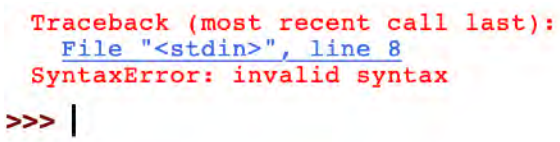
[23] Some powerbanks will shut off after a while if the current draw is too low, so you may want to check that.

[24] This is a variation on the Pico board with a LiPo connector directly on it: https://shop.pimoroni.com/products/pimoroni-pico-lipo

[25] It is called open-source software and it's awesome.

Troubleshooting 🐞 🎯

Error messages are your friends. They will tell you what is going wrong where. When you see a message such as this one, click on the blue link and then that line will be highlighted, showing you where the code needs fixing.



Problems with connecting

1. Select the right interpreter in the bottom right of the Thonny screen. If you want your code to run on the Pico, it should say 'MicroPython Raspberry Pi Pico'. (If it's not there, see the next page on how to install new firmware.)
2. Is the USB cable connected properly?
3. Try connecting the cable to another USB port (yes it's silly and yes it may help).
4. Can you make a connection with the Pico when nothing else connected to it?
5. Try another Pico.

Problems with code

Often, the error messages the Pico gives in the shell are really quite helpful. Try to understand what they mean.

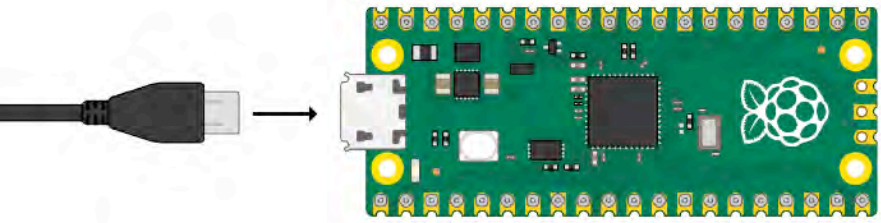
1. Click on the link in the error message to find out where in the code the error is happening. It should highlight the line with the problem.
2. IndentationError: unexpected indent: make sure that lines within an if-statement or while-loop have the same number of spaces in front of them.
3. ModuleNotFoundError: No module named '..': install the required library; see the section 'Installing libraries' in this guide.
4. Other errors: search online for solutions to the particular error messages you are seeing. Search with terms between quotes, like 'Pico "exact error here"'.

No error, but also not working the way you expected?

- Check if all components are connected to the pins they are assigned in the code.
- Use print()'s to find out what Pico is doing (see the page 'Shell and Plotter').
- If you can't figure it out: try to find code that does what you want online.
- Try another one of the component you are using, to find out if the first one might be broken. After that, try different wires or even a different breadboard.

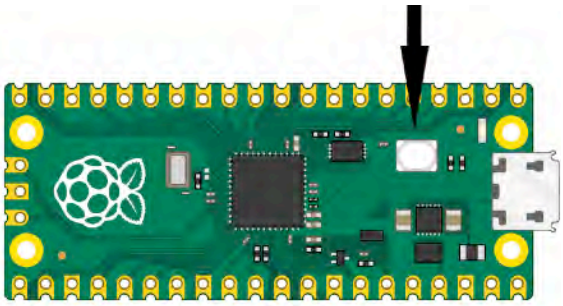
Adding the MicroPython firmware

To be able to ‘speak’ Python, the Pico needs to be prepared. If you are using a brand new Raspberry Pi Pico, you will need to put the MicroPython firmware on it.

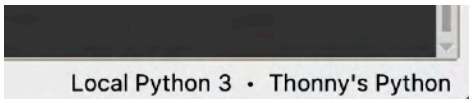


Start by plugging a micro USB cable into the micro USB port on your Pico.

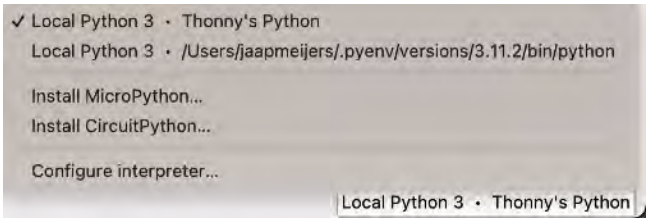
Press the white ‘BOOTSEL’ button. While holding it, connect the other end of the USB cable to a USB port on your computer. After three seconds, let go of the button.



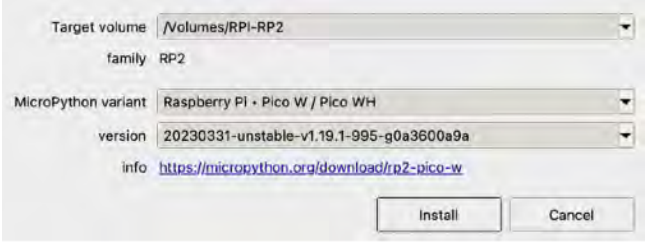
In the bottom right-hand corner of the Thonny window, you will see the current version of Python.



Click on it and choose ‘Install MicroPython’ from the list that appears. If you don’t see this option, check that your Raspberry Pi Pico is plugged in.



In the next screen, it should say ‘RPI-RP2’ as the target volume. Set the MicroPython variant to ‘Pico/Pico H’ (or ‘Pico W/Pico WH’ if you know you have that one) and click ‘Install’.



Wait until it is done and then click ‘Close’. Now you should be able to select ‘MicroPython (Raspberry Pi Pico)’ from the list.

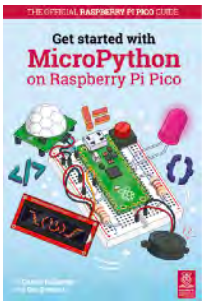
Tips on making things with electronics

There is an infinite number of input and output combinations you can make with the Pico. For instance, having an MP3-player play sounds when a motion sensor detects motion. Or let LED strips change colour based on a heart rate you pick up with a heart rate sensor. (See the next page for more examples.)

A few things that are not covered in this quick guide but that you can definitely do, are using the Pico as an HID (human input device, so it pretends to be a keyboard or mouse),[26] using it as a MIDI controller or sending out basic video. The Pico W also has Bluetooth and WiFi, so you can send data wirelessly.

There are practically no barriers anymore to what you can do with electronics. Parts have become incredibly cheap, and information on how to use them is always available online. **Search online** for something like 'Pico plant moisture sensor' and you will find lots of tutorials that show how to connect such a sensor but also share the necessary code!

A couple of sources are particularly practical and inspirational:



- Official guide from the Raspberry Pi Foundation, available as a free PDF: https://hackspace.raspberrypi.com/books/micropython-pico
- Raspberry Pi Foundation website: https://projects.raspberrypi.org/en/projects/getting-started-with-the-pico/
- Instructables: https://www.instructables.com/search/?q=raspberry%20pi%20pico
- Hackster.io: https://www.hackster.io/Pico

The most important thing to understand is that **everyone can do programming and make things with electronics**. Nobody is an expert right from the start at anything, but practising and not giving up will allow you to make awesome things.

---

[26] See https://hridaybarot.home.blog/2021/01/31/using-raspberry-pi-pico-has-hid-device-to-control-mouse-and-keyboard/ — that actually requires Circuitpython, so a different flavor firmware.

More inputs and outputs

Inputs

Distance sensor
Accelerometer
Gyroscope
Compass
Temperature
Relative humidity
Moisture (water plants automatically!)
Barometer
Light sensor
Colour sensor (sort candy to colour :-)
Control knobs and linear potentiometers (like the ones on audio mixers)
Microphones
Touch
Switches
Keyboard and mouse
Fingerprint
Cameras
Motion sensor
Infrared receiver (controlling your project with a remote control!)
Heartbeat[27]
Smoke
Gas
Radiation
Flame detector
Magnetic fields (Hall effect sensor or reed switch)
Vibration
Bend or stretch sensors
Water current
Brain activity

Outputs

Light:
- lamps
- lasers
- LEDs
- glowing kanthal wire

Sound:
- speakers
- transducers
- MP3 players[28]

Motors:
- servos
- precise stepper motors
- fast DC motors
- strong gear motors
- vibration motors
- linear actuators
- ventilators

Water pump (for instance to make it rain or let a statue ‘cry’)
Solenoids
Elektromagnets
Video (animations, videos, virtual reality/augmented reality)
LCD display (like in vending machines)
Metal that changes shape
Heating/cooling elements
Smoke machine
Website or social media (automatically post info based on sensor data)

This is not a complete list, the possibilities are practically endless.

What combinations of inputs and outputs are you going to try?

---

[27] Some heartrate sensors barely work. This one is a bit more expensive, but it is pretty reliable: https://shop.pimoroni.com/products/max30101-breakout-heart-rate-oximeter-smoke-sensor

[28] This one is cheap and easy to connect: https://wiki.dfrobot.com/DFPlayer_PRO_SKU_DFR0768

Contents

This manual was written by Jaap Meijers, instructor and manager of the Hacklab at the Royal Academy of Art in The Hague.

It was published in June 2023

Image credits

- Pico pinout diagram, images on page ‘Adding the MicroPython firmware’ and official guide book cover: Raspberry Pi Foundation
- All connection diagrams: Fritzing.org
- Breadboard: Victoria.nunez2/ Wikimedia
- Flame icon, icons in diagram about if-statements and icons on distance sensors page: https://freesvg.org
- Push button photo: Kevin Casper/ publicdomainpictures.net
- LDR: Arnau 944/Wikimedia
- Potentiometer: Iainf/Wikimedia
- Cash register: Bruno Neurath-Wilson/Unsplash
- Stopwatch: Matthew/Flickr
- Plants with Pico, moisture sensor and LED ring: https://andywarburton.co.uk/2021/pi-pico-soil-moisture-indicator/

[29] https://creativecommons.org/licenses/by/4.0/