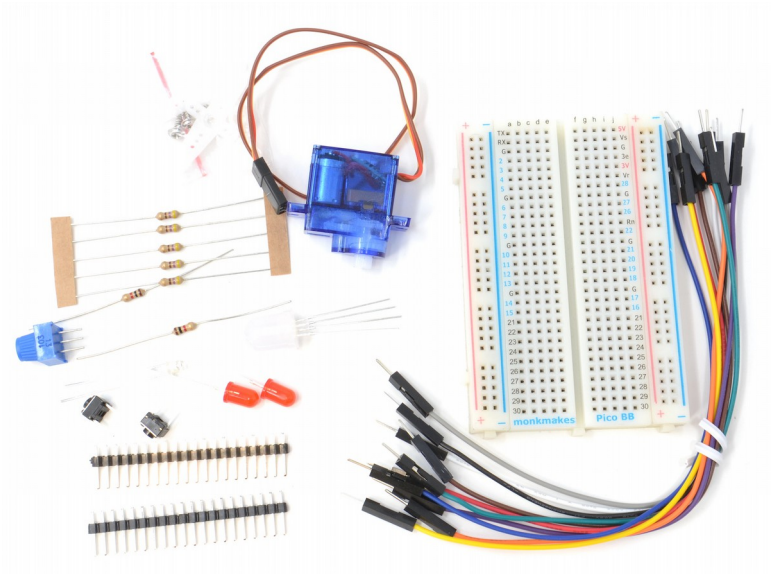
Instructions: Electronics Kit 1 for Pico

Learn Coding and Electronics with the Raspberry Pi Pico



another bright idea!
MONK MAKES PRESS

Dr. Simon Monk

Technical reviewer: Ian Huntley

v1e

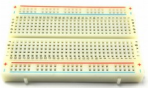
Table of Contents

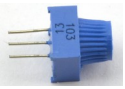
Acknowledgements

The breadboard diagrams in this booklet were created using Fritzing. See https://fritzing.org/

Parts

Before you start making your projects, please make sure that you have all the parts listed below.

Solderless breadboard.	
Jumper wires - Use to connect things together on the breadboard.	
Set of header pins, that can be soldered to your Raspberry Pi Pico. See the section “Setting Up” for instructions. If your Pico already has pins attached, then you don't need these.	
Servomotor and bag of servomotor arms and screws.	
5 x 470Ω resistor (yellow, violet, brown stripes). *	
2 x 1kΩ resistor (brown, black, red). *	
4.7MΩ resistor (yellow, violet and green stripes). *	
2 x Red LED – the longer lead is the + (positive) lead.	
RGB LED – the longest lead is the – (negative) lead.	

2 x Tactile push switches.	
Phototransistor. The longer lead is the negative lead.	
Buzzer.	
Pot (variable resistor).	

* See Appendix C for more help identifying resistors.

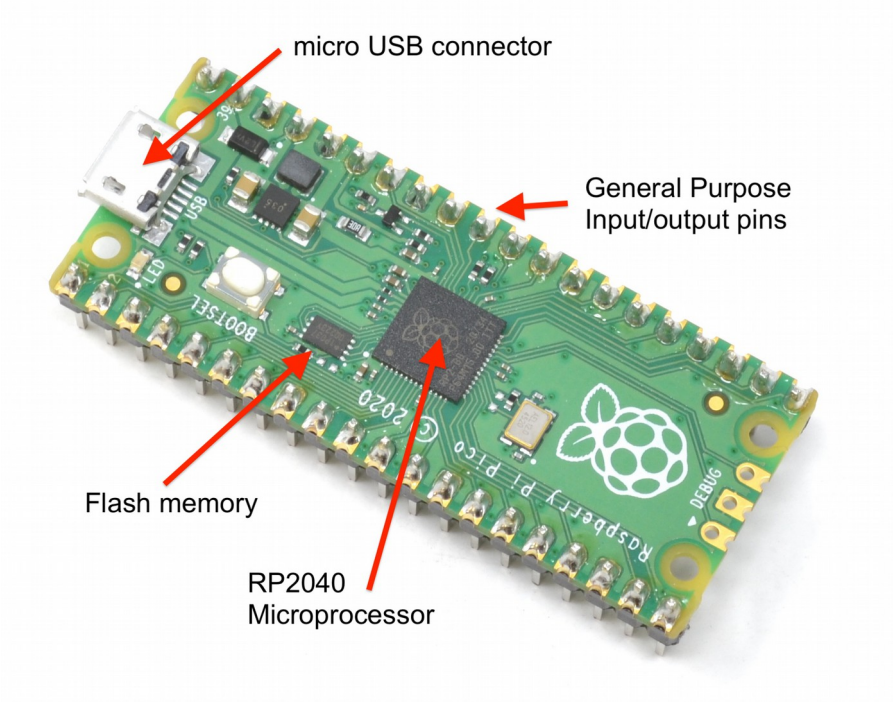
You will also need a Raspberry Pi Pico and a micro-USB lead to connect it to your computer.

If your Pico was supplied without header pins soldered on, then you will need a soldering iron and solder, to attach the supplied header pins onto the Pico.

Some scrap cardboard, scissors, paper glue and a craft knife will also be required if you want to box Projects 5 to 8.

The Raspberry Pi Pico

The Raspberry Pi Pico is a small circuit board containing a microcontroller chip and a few supporting components plus a whole load of connections that you can use to control electronics. This is quite different from other models of Raspberry Pi that are proper computers running an operating system. The pico can only run one program at a time.



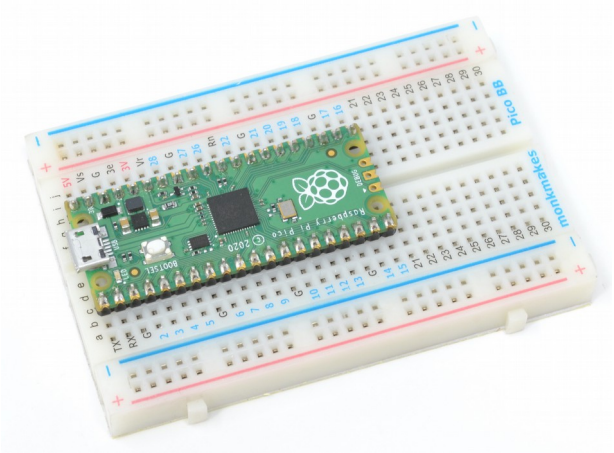
The Pico (as we shall refer to it) needs a computer (Windows, Mac, Linux or a regular Raspberry Pi) in order for you to write programs for it and then send those programs to it using its built-in USB interface. The USB connector also provides power to the Pico.

The connections all around the edge of the Pico are called General Purpose Input Output (GPIO) pins and allow you to connect electronics to the Pico such as LEDs, servomotors and sensors of different types.

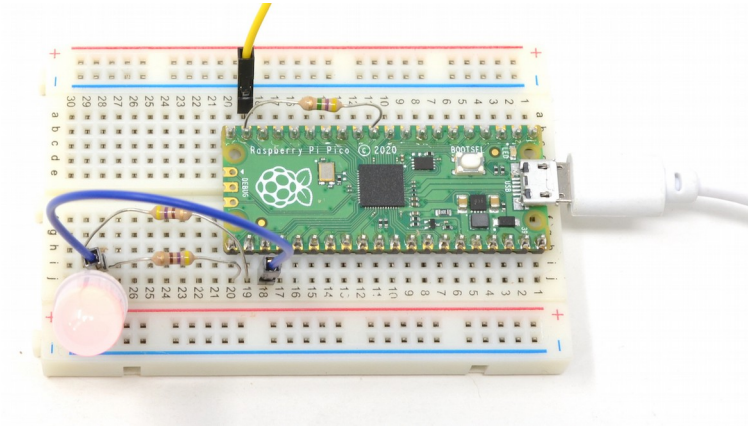
The Pico has its own custom microcontroller chip (the RP2040). This will run programs stored in the Pico's flash memory chip, or even run an interactive Python shell.

Although you could solder things directly to the GPIO pins, that would make it hard to take the project apart again to use the Pico for something else, so most people solder header pins onto the Pico and then plug it into solderless breadboard as shown below.

This breadboard has labels for the Raspberry Pi Pico on it. Some of the photos in this booklet show standard breadboard which has numbered rows rather than Pico pin names.



This way, we get all the soldering over at once, and we can just push components and jumper wires onto the breadboard when we are making a project, as shown below.



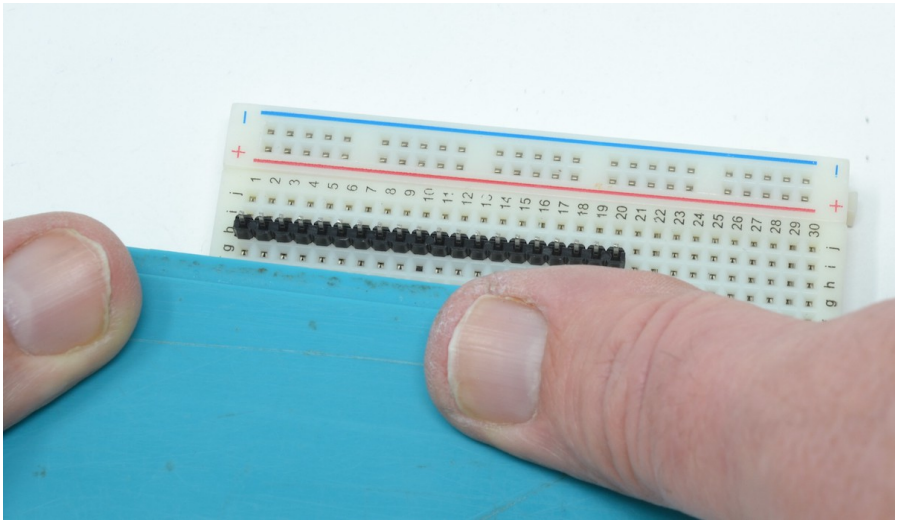
Setting up

Before getting going on the projects, there are a few steps that you need to take to prepare your Pico and the computer that you will need in order to program it.

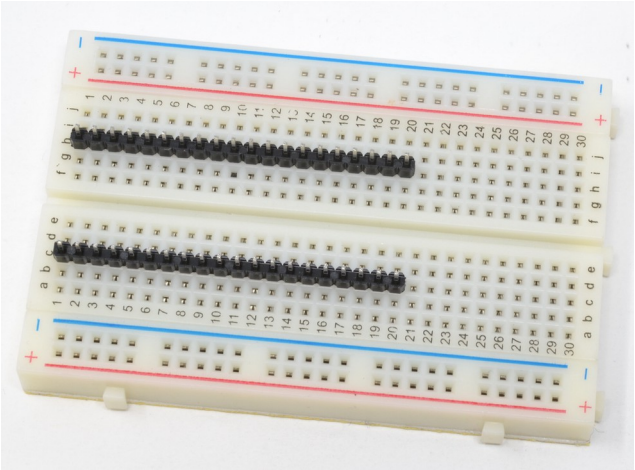
Soldering header pins

If your Pico does not have header pins attached, you will have to solder on the pins supplied with this kit. To do this, you will need a soldering iron and some solder wire. Don't worry if you are new to soldering its not so hard.

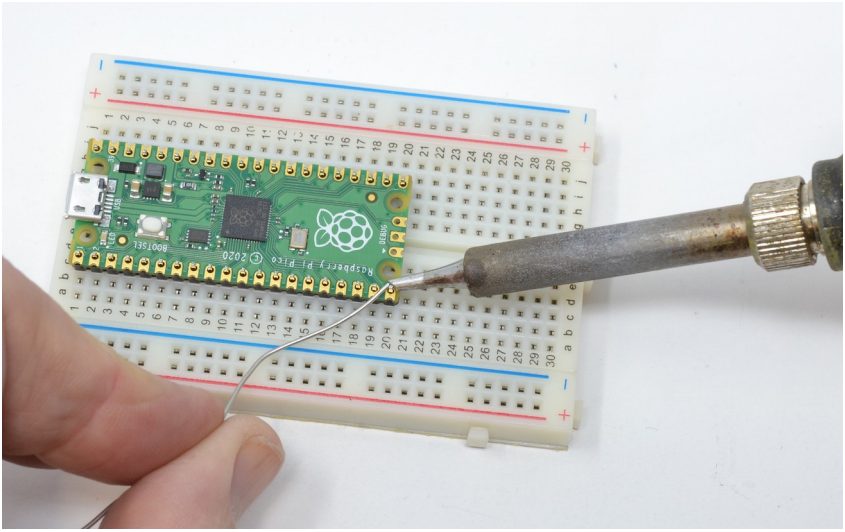
To keep the pins straight, it is a good idea to push them into the solderless breadboard long ends first. This has the added advantage that once soldered, the Pico will be in the right place, and won't need to be relocated. It's a good idea to use a ruler or other flat object to push the rows of header pins onto the breadboard, as they can be sharp on your fingers.



The pins need to both start at row 1 (marked on the breadboard) and run down columns **c** and **h**.



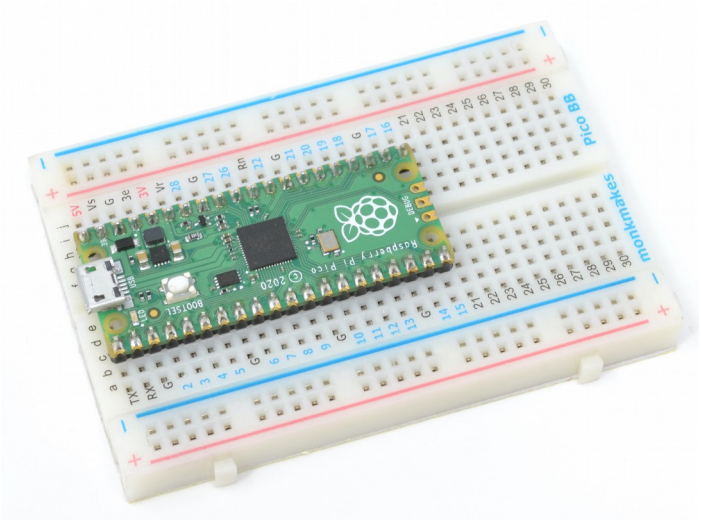
Once the pins are in the right place on the breadboard, the Pico should fit easily over the pins with the USB connector towards the edge of the breadboard.



If its your first attempt at soldering, watch a few tutorials before you start to get an idea of what you are trying to do. Here's a good one:

https://www.youtube.com/watch?v=37mW1i_oEpA

Once all the pins are soldered, your breadboard should look like this:



Thonny

Thonny is a Python editor, that allows you to edit MicroPython programs using your computer before running them on your Pico. Thonny is available for most operating systems including Raspberry Pi OS, so, if you have a Raspberry Pi 4 or 400, you can use it to program your Pico.

If you are using a regular Raspberry Pi to program your Pico, then you are in luck, because Thonny is pre-installed. If you are using a Mac, Windows of Linux computer, then you will need to install it following the instructions here:

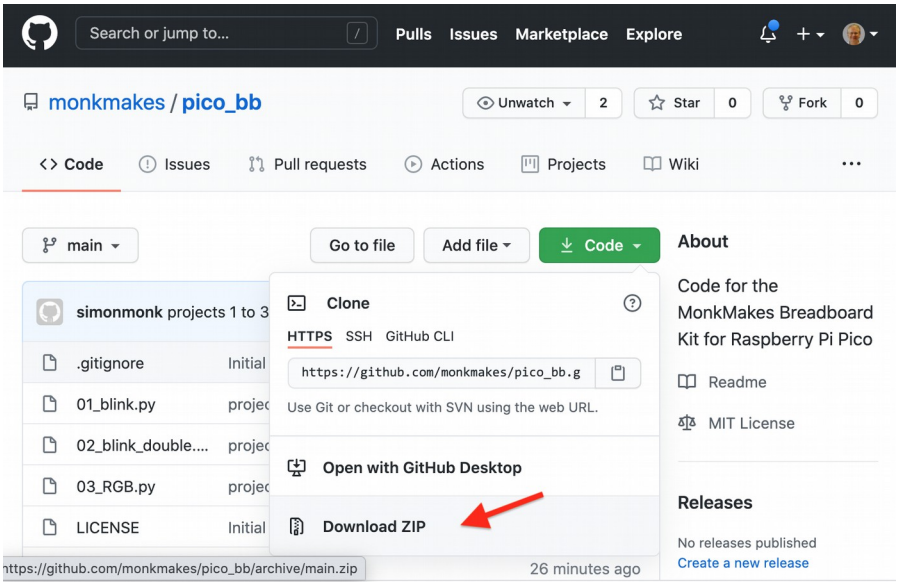
https://thonny.org

Downloading the Project Software

All the project code for this kit is available for download from GitHub here:

https://github.com/monkmakes/pico_kit1

To download the programs onto your computer, type the address above into your computer's browser, click on the Code button, and then select **Download ZIP** from the menu.

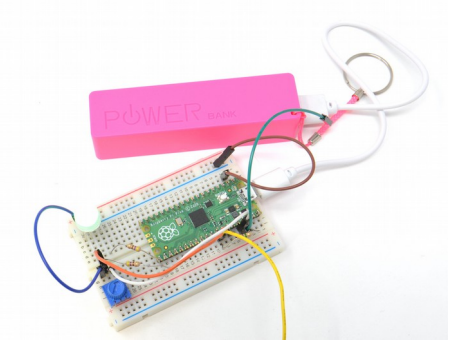


Once the ZIP file has downloaded, extract the contents of the ZIP file. The process will be different for different operating systems, but, generally if you right-click on the file there will be an option to extract the contents.

To open one of the downloaded programs in Thonny, you should be able to double-click it. Alternatively, start Thonny and then click **Open** and browse to the file.

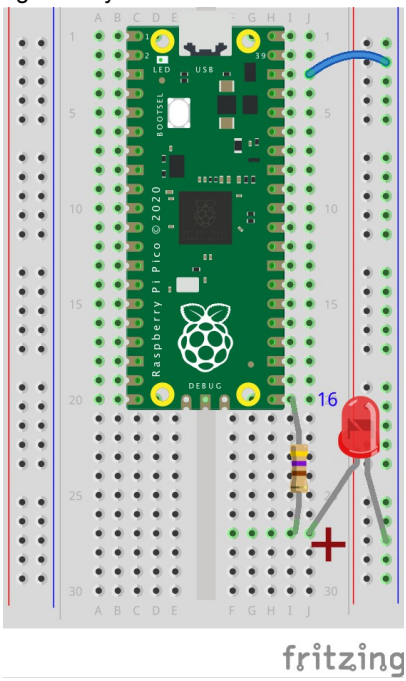
Powering the Projects

Most if the time, your Pico is going to be powered by your computer using the USB lead. However, if you want to make your project independent, a great way of powering it is by using a rechargeable USB battery like the one shown here.



Project 1. Make an LED Blink

To make this first project, you are going to wire up the breadboard as shown below, and then run the program for this project using Thonny.



Step 1. Find the parts you need

For this project, you will need the following items:

- 470Ω Resistor (yellow, violet and brown stripes)
- Red LED
- A jumper wire

Use the table of parts on pages 2 and 3 to identify the parts you need. If you are interested in what the resistor color stripes mean, see Appendix C.

Step 2. Build the breadboard.

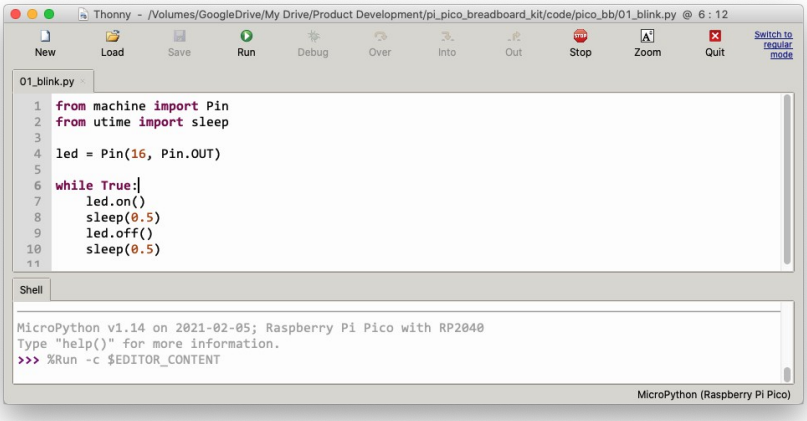
Let's start by placing the components onto the breadboard. Using the diagram to the right, push the component legs through the holes in the breadboard at the positions shown.

If you haven't come across breadboards in the past, Appendix B provides an explanation of how they work.

Be careful with the orientation of the LED - the long leg is the positive lead (row 27), and the shorter leg is the negative lead (the rightmost column marked with a blue line and known as the *negative bus*). It does not matter which way around the resistor goes.

Step 3. Run the Program

To run the program, first open the file 01_blink.py in Thonny and connect your Pico via USB lead to your computer.



To start the program, click on the **Run** button (play icon). If everything is set up correctly, the LED should be blinking on and off.

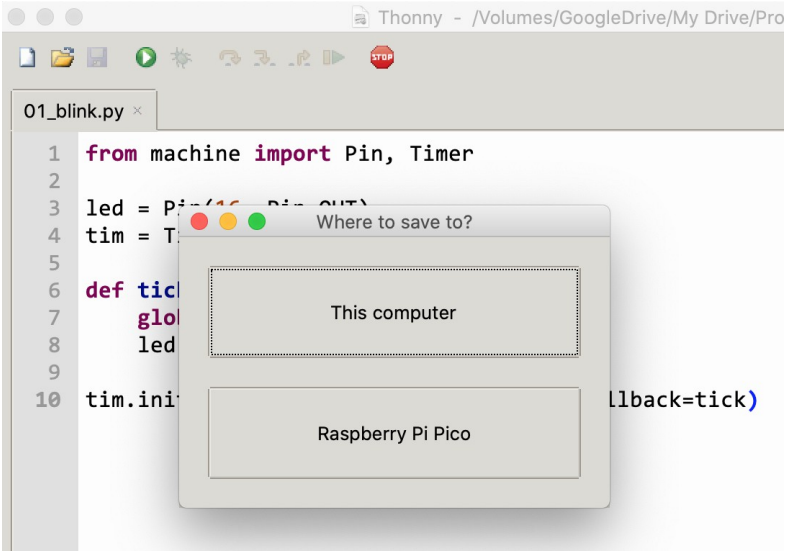
When you want to exit the program, click on the **Stop** *button*.

If the LED doesn't blink on and off, check that it's the right way around and that everything is wired up as shown in the diagram.

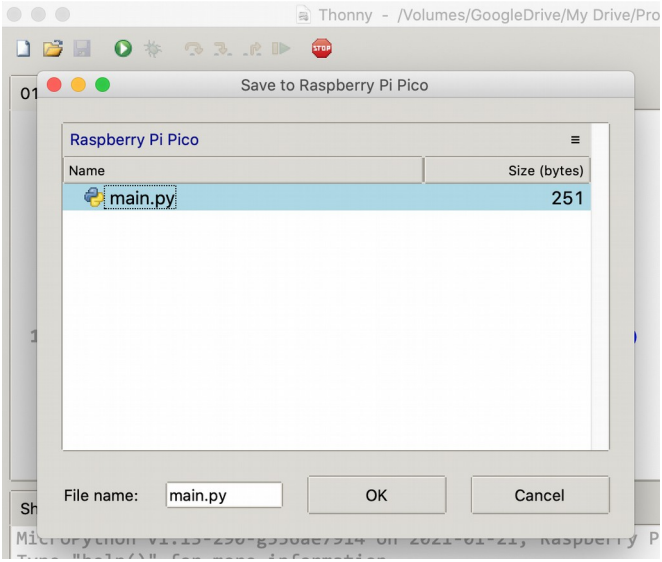
What's happening here is that Thonny is running the program on your Pico without fully installing it on the Pico. By this, I mean that the program will not run on the Pico without the Pico being attached to your computer and if you were to detach your Pico and power it from a separate USB power adapter, then it would not run the program.

To fully install the program, so that it will start automatically when the Pico is powered up, click on the **Stop** button and then switch Thonny to *Regular mode* rather than *beginner* mode by clicking on the link shown above.

Once in regular mode, there are a few new options available on the menu including **Save Copy**. When you click on this, you will be given the option to save to either **This Computer** or your **Raspberry Pi Pico**.



Click the **Raspberry Pi Pico** button, make sure the file has the name **main.py,** and then click OK.



Restart the Pico by unplugging it and plugging it back in again; the LED should now blink, whether or not Thonny is running the program.

The Code for Project 1

If you want to understand how the code works, take a look at the code in Thonny.

Any lines beginning in **#** are called *comment* lines. These are not actually program code, but rather explain something that's going on in the code.

```
from machine import Pin
from utime import sleep

led = Pin(16, Pin.OUT)

while True:
    led.on()
    sleep(0.5) # pause
    led.off()
    sleep(0.5)
```

The *import* commands specify the library files that the code uses. In this case, the *machine* library gives our code access to *Pin* and the *utime* library gives us the *sleep* function.

When defining the variable *led*, the number 16 refers to the pin that the LED is connected to. *Pin.OUT* indicates that the pin is to be used as an output.

The *while* loop will just continue forever, turning the LED on, pausing for half a second and then turning it off again, pausing for another half a second before getting back around the loop again.

This approach is fine for blinking an LED as long as you don't want to do anything else at the same time. In this case, and alternative method of blinking the LED is possible because the Pico's microcontroller chip contains a hardware timer that allows us to set things to happen at regular intervals. This is just what we need to make our LED blink. So you could replace the *while* loop with the following code:

```
from machine import Pin, Timer

led = Pin(16, Pin.OUT)

def tick(timer):
    led.toggle()

Timer().init(freq=2, callback=tick) # call tick twice a sec
```

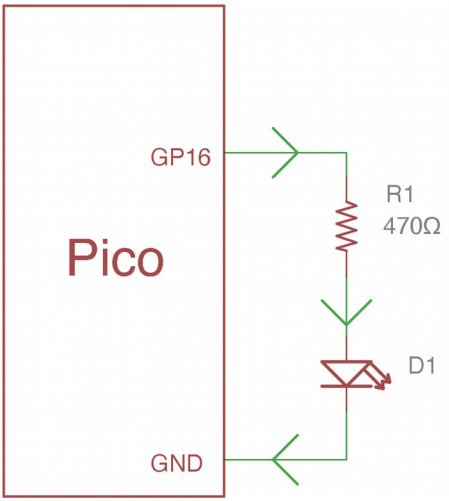
The parameter *freq=2* specifies that the timer should trigger something at 2 Hertz – that is 2 times per second. The callback parameter specifies the name of the function to be called. In this case that is the function called *tick*. This *tick* function tells the pin to toggle. That is, if its on, to turn off and if its off to turn it on.

Things to Try

To make the LED blink faster, try changing the values of the *freq* parameter.

The Electronics

Here is a schematic diagram of how the LED is connected to the Pico.



When the pin GP16 of the Pico has a value of 0, its output voltage is also 0, and when its value is set to 1, its output voltage will be 3.3V. Current will only flow through the LED and resistor, making the LED light if there is a difference in voltage across them. If the output pin is at 0V there is no voltage difference so the LED will not light. However, when the pin is at 3.3V current will flow, lighting the LED. The resistor is needed to restrict the current flowing through the LED. If too much current flows through the LED it could break or damage the output pin of the Pico, so always use a resistor.

The resistor's resistance value is 470Ω. This determines how much the resistor impedes the flow of current and this resistor will limit the flow of current through the LED to about 3mA.

Project 2. Make 2 LEDs Blink

Don't dismantle Project 1 just yet, for Project 2, you are going to add another LED and make the two LEDs blink alternately.

BEFORE you start changing what's on the breadboard, unplug the USB lead from your Pico. An accidental short-circuit could damage or break your Pico or your computer.

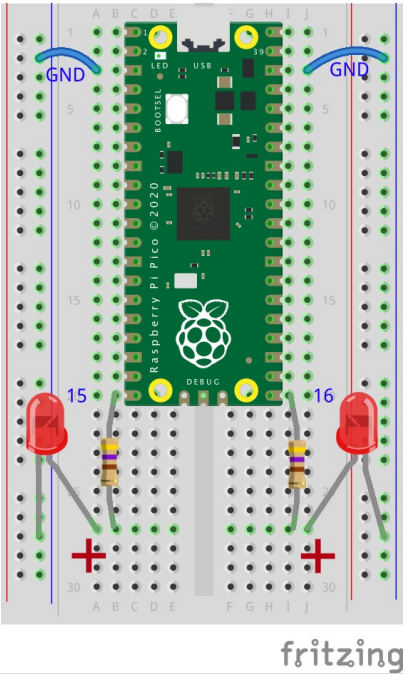
Step 1. Find the parts you need

For this project, you will need the following items:

- Two 470Ω Resistors (yellow, violet, brown stripes)
- Two Red LEDs
- Two jumper wires

Step 2. Build the breadboard.

Wire up the breadboard using the diagram to the right as a guide. Remember to put the LEDs longer positive leads in the position marked with a + below.



Step 3. Run the Program

Load and run the program called 02_blink_double.py in Thonny. The LEDs should now blink in turn.

The Code for Project 2

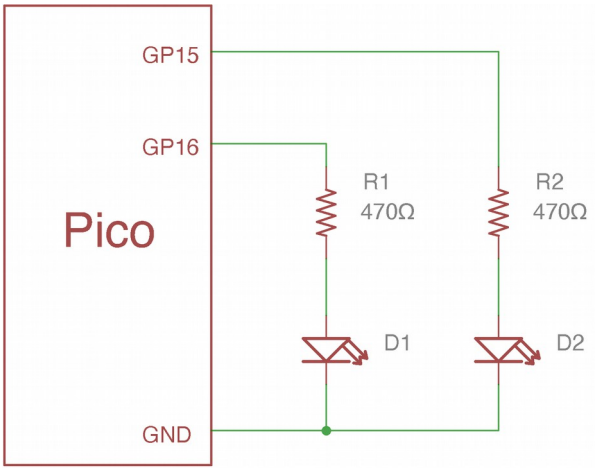
The code is very similar to the more advance blink code using a timer, except that now we have 2 LEDs to toggle.

```
from machine import Pin, Timer

led1 = Pin(16, Pin.OUT)
led2 = Pin(15, Pin.OUT)
led1.value(0)
led2.value(1)

def tick(timer):
    led1.toggle()
    led2.toggle()

Timer().init(freq=2, callback=tick)
```

One LED is turned on and the other off before the toggling starts, so that the LEDs alternate.

The Electronics

Please refer to the description for project 1. This project is just the same, but repeated for a second LED.



Project 3. RGB Color Display

This project will demonstrate how to control an RGB (multicolor) LED with your Pico, cycling through different colors when you press the button.

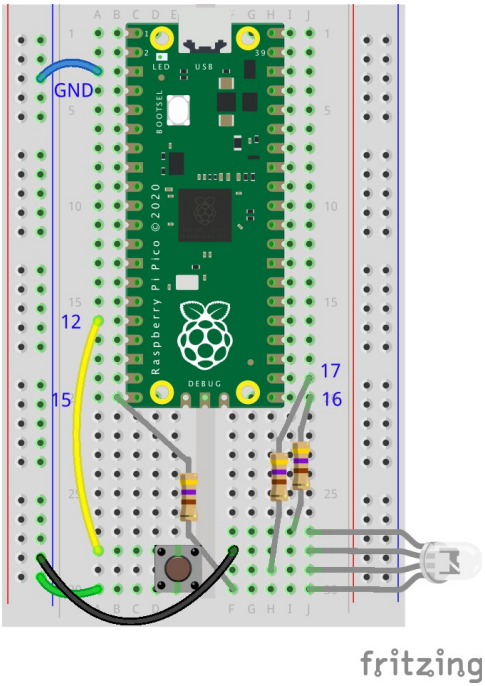
Step 1. Find the parts you need

For this project, you will need the following items:

- Three 470Ω Resistors (yellow, violet, brown stripes)
- An RGB LED
- A push switch
- Four jumper wires

Step 2. Build the breadboard.

Wire up the breadboard using the diagram to the right as a guide.



All three of the resistors used in this project are the same value of 470Ω. You should find that one of the RGB LED's leads is longer than the rest. Place the RGB LED such that the long lead is in row 28 of the breadboard and connected to the GND (black) wire. Make sure that the legs of the resistors are not touching each other.

The push switch should be inserted with its legs on rows 28 and 30.

Step 3. Run the Program

Load and run the program called 03_rgb.py in Thonny. Pressing the switch should cycle through a series of colors on the LED.

The Code for Project 3

Here's the code for Project 3.

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

red_ch = PWM(Pin(16))
green_ch = PWM(Pin(17))
blue_ch = PWM(Pin(15))

red_ch.freq(1000)
green_ch.freq(1000)
blue_ch.freq(1000)

button = Pin(12, Pin.IN, Pin.PULL_UP)
colors = [[255, 0, 0], [127, 127, 0],[0, 255, 0],
          [0, 127,   127], [0, 0, 255], [127, 0, 127]]

def set_color(rgb):
    red_ch.duty_u16(rgb[0] * 256)
    green_ch.duty_u16(rgb[1] * 256)
    blue_ch.duty_u16(rgb[2] * 256)

index = 0
set_color(colors[index])
while True:
    if button.value() == 0:
        index +=1
        if index >= len(colors):
            index = 0
        sleep(0.2)
    set_color(colors[index])
```

Whereas projects 1 and 2 just turned LEDs on and off, in this project, different colors are mixed on the RGB LED by separately controlling the brightness of the red, green and blue channels of the LED.

To do this, something called PWM (Pulse Width Modulation) is used. You can read more about this in Appendix D.

To use PWM with a pin, you need to wrap the *Pin* declaration in *PWM* like this:

```
red_ch = PWM(Pin(16))
```

This specifies that PWM is to be used on GP16. There are similar lines of code for

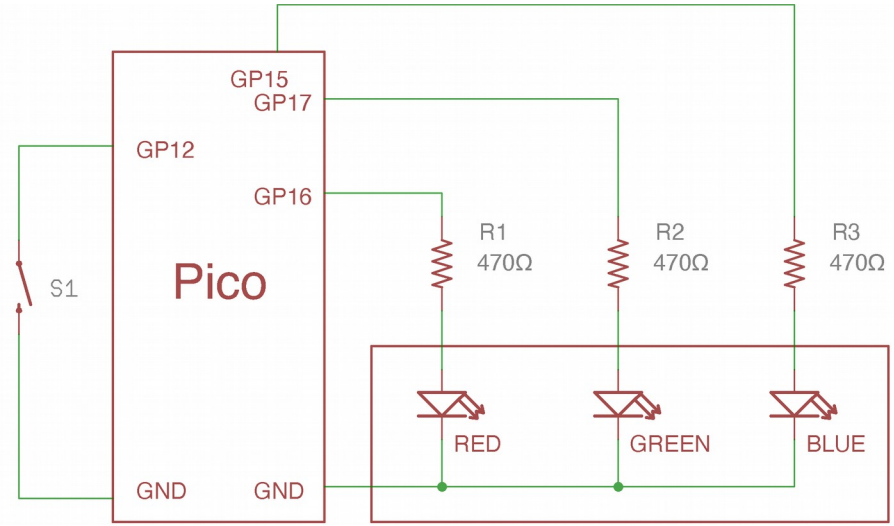
the green and blue channels.

The *colors* list is a list of lists each of three values between 0 and 255. The arrays of three elements represent a single colour in terms of its relative amount of red, green and blue light. So a 0 means that color channel is off, a value of 127 indicates half brightness and a value of 255 full brightness.

The main loop waits for a button press and then adds one to the color *index*. As long as that is within the bounds of the *colors* list, *set_color* is called to change the brightness of the three LED channels.

The Electronics

An RGB LED is actually three LEDs in one package – one red, one green and one blue. Given that LEDs have two legs each you might expect the package to have six legs, but actually it just has 4. That's because all the negative ends of the LEDs are connected to the same pin on the package (the longest leg).

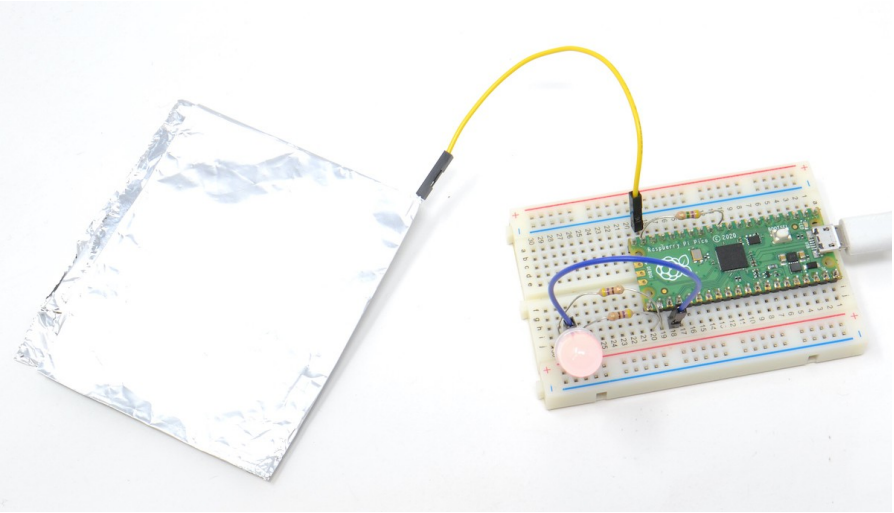


Each LED channel must have its own resistor to limit the current.

The push button connects Pico pin 12 to GND (0V) when the button is pressed. This is what is detected in the Python code. By default, the pin is 'pulled-up' to 3.3V, and so reads as a value of 1 unless the switch is pressed to force the pin to 0V.

Project 4. Touch Switch

This project builds on Project 3, but instead of switching between colors it demonstrates the principal of a touch switch, turning the LED green when the sensor is touched.



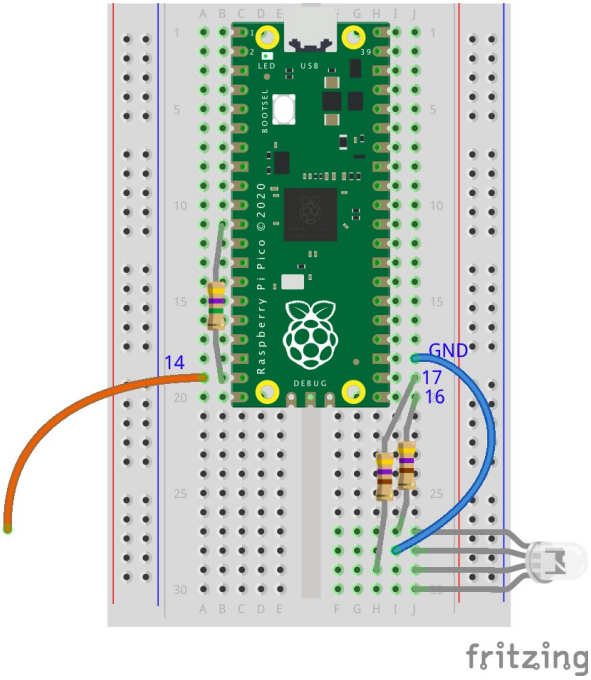
Step 1. Find the things you need

For this project, you will need the following items:

- Two 470Ω resistors (yellow, violet, brown stripes)
- One 4.7MΩ resistor (yellow, violet, **green** stripes)
- An RGB LED
- Two jumper wires
- A small area of metal cooking foil (optional)

Step 2. Build the breadboard

To adapt the breadboard for this project after building Project 3, first remove the push switch and the leftmost resistor (we don't need the blue LED channel) and the jumper wires from the left-hand side of the breadboard.



Next add in the 4.7MΩ resistor and wires as shown here. Note that the RGB LED's GND connection now comes from a different pin on the Pico. All of the GND pins on the Pico are actually connected together and can be used interchangeably.

One end of the orange jumper wire is connected to row 19 on the left (GP14) and the other end will eventually be connected to our metal foil sensor.

Step 3. Run the Program

Load and run the program called 04_touch.py. As you touch the bare end of the sensing wire (jumper wire from row 19), the LED should change color from red to green. If you watch the Thonny Shell, you will see a stream of readings. Try touching the sensing wire and you should see the numbers increase. You may need to change the value of the variable *sense_threshold* from 100 to something between the values you see when not touching versus touching.

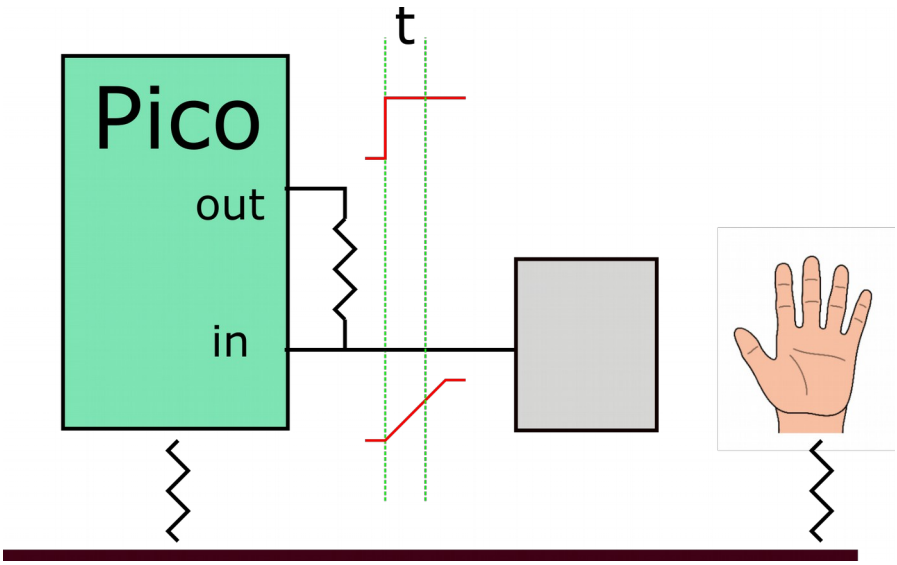
Step 4. Make the Sensor Pad

You can make the touch sensor more sensitive by increasing the area. If you have some cooking foil, try folding it in half twice and trapping the loose end of the sensing lead in a fold in the foil.

You should find that if you do this, you can even hide the foil behind paper, card or plastic and the sensor will still detect touches.

CapSense

This project uses a technique called capsense, that relies on the fact that when you turn on a digital output that is connected to a digital input by a high value resistor (in our case 4.7MΩ – 4.7 million ohms) there is a delay before the input follows the output crossing the threshold where it counts as a digital high.



The time of this lag (t in the diagram) is influenced by the presence of a conductive element (such as a finger or hand) with a very high resistance path back to the Pico. So if you touch or even hold your hand very close to the sensor pad, the delay will increase. By measuring this delay, you can detect when the pad is being touched.

The Code for Project 4

Here is the code for Project 4.

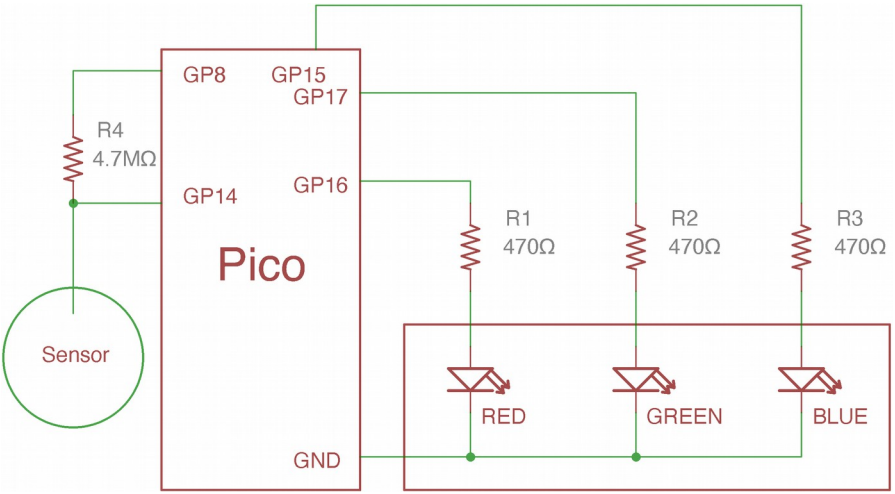
```
from machine import Pin
from utime import sleep, ticks_us

red_ch = Pin(16, Pin.OUT)
green_ch = Pin(17, Pin.OUT)

out_pin = Pin(8, Pin.OUT)
in_pin = Pin(14, Pin.IN)

sense_threshold = 100


def cap_sense_value():
    total = 0
    for i in range(20):
        out_pin.high()
        t0 = ticks_us()
        while in_pin.value() == 0:
            pass
        t1 = ticks_us()
        t = t1 - t0
        out_pin.low()
        sleep(0.01) # allow plenty of time for
                    #in_pin to go to 0V
        total += t
    return total / 20

while True:
    value = cap_sense_value()
    print(value)
    if value >= sense_threshold:
        red_ch.low()
        green_ch.high()
    else:
        red_ch.high()
        green_ch.low()
```

Most of the work takes place in the function *cap_sense_value*. This function actually takes the average of 20 capsense readings for better reliability. To take each of these readings, the output pin is set high and the current time, as a value in microseconds, recorded in the variable *t0*. The while loop then waits until the input pin catches up with the output pin and becomes high. At this point, the time is recorded again (into the variable *t1*) and the delay calculated as the difference

between *t1* and *t0*.

The Electronics

The schematic diagram for this project shows how the 4.7MΩ resistor is connected between the output and input pins (GP8 and GP14).



Project 5. Servomotor Control

This is the first of the projects to use the servomotor supplied with the kit. In this project you will be able to move the arm of the servomotor left or right, using the two small push buttons.

Step 1. Find the parts you need

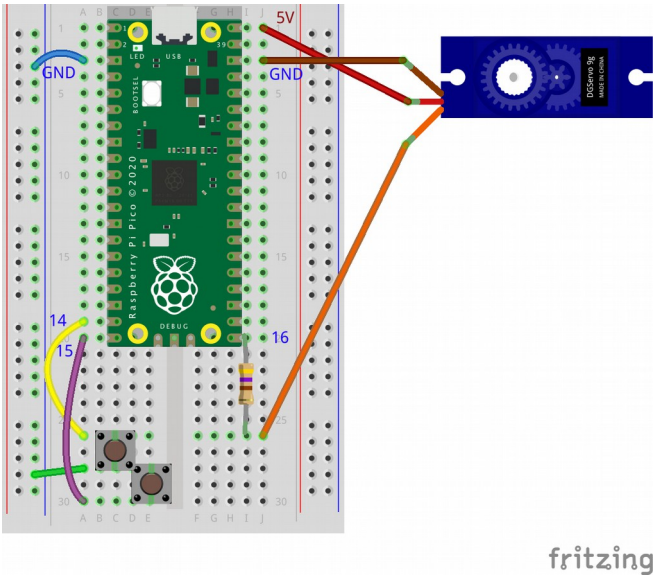
For this project, you will need the following items:

- Servomotor
- An arm for the servomotor
- A 470Ω resistor
- Two push switches
- Seven jumper wires

Step 2. Build the breadboard.

Wire up the breadboard and connect it to your Pico using the diagram below as a guide.

To connect the servomotor, use jumper wires. The red lead of the servomotor should be connected to USB 5V on the Pico, the brown lead of the servomotor to GND and the orange lead to row 26 as shown.



Insert the switches carefully – note that they share one row of connectors on the breadboard.

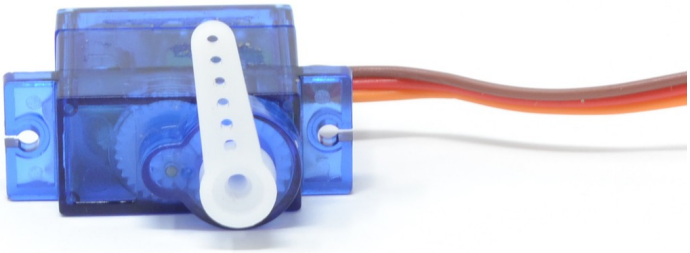
Step 3. Attach a servomotor arm

The servomotor is supplied with a little bag of arms that fit over the cogged drive shaft of the servo motor. Select the one shown below and push it onto the drive shaft. It doesn't matter at what angle you attach it, as you are going to be adjusting it later anyway.

Step 4. Running the Program

Open the file 05_servo_test.py in Thonny and run it. You should find that, when you press one button, the servo arm rotates one way; and when you press the other button it should rotate the other. You will also see the current angle printed out in the Shell.

If the servo arm is not free to move throughout its range you can adjust the position of the arm by taking it off and repositioning it. To get it at the straight up 90 degree position, Stop the program and then run it again. This will start the servo arm at the middle 90 degree position. So, without pressing either button, carefully take the arm off and reposition it so it is pointing straight up as shown below. Note that because of the cogged drive shaft, you probably won't be able to get the arm at exactly the 90 degree position.



Use the up and down buttons and make a note of good values for *min_angle* and *max_angle* so that the left and right end positions are evenly either side of the central position. A little counter-intuitively, the position of the arm is such that an angle of 0 is over to the right and 180 to the left.

The Code for Project 5

Servomotors are controlled by a series of pulses. We are going to generate those pulses using PWM (see Appendix D). For more details of how servomotors work, see Appendix E.

You will find the code for this project in 05_servo_test.py

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

button_up = Pin(14, Pin.IN, Pin.PULL_UP)
button_down = Pin(15, Pin.IN, Pin.PULL_UP)

servo = PWM(Pin(16))
servo.freq(50) # pulse every 20ms

def set_angle(angle, min_pulse_us=500, max_pulse_us=2500):
    us_per_degree = (max_pulse_us - min_pulse_us) / 180
    pulse_us = us_per_degree * angle + min_pulse_us
    # duty 0 to 1023. At 50Hz,
    # each duty_point is 20000/65535 = 0.305 µs/duty_point
    duty = int(pulse_us / 0.305)
    print(angle)
    servo.duty_u16(duty)

angle = 90
set_angle(90)
min_angle = 10
```

```
max_angle = 160

while True:
    if button_up.value() == 0 and angle <= max_angle:
        angle += 1
        set_angle(angle)
        sleep(0.01)
    elif button_down.value() == 0 and angle > min_angle:
        angle -= 1
        set_angle(angle)
        sleep(0.01)
```

The PWM frequency is set to 50Hz for this project, as that is the frequency of pulses that the servomotor expects.

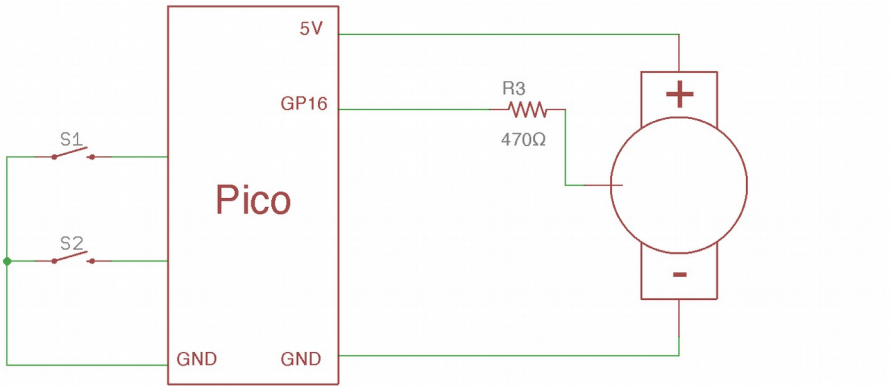
Most of the work in generating the pulses to control the servomotor is contained in the function *set_angle*. This means that, if you want to create your own projects using servomotors, you can just copy this function.

The most important parameter for *set_angle* is the angle that you want to set the servomotor's arm to, which should be between 0 and 180. The other parameters set the minimum and maximum pulse width, that you should not need to change these unless you have an unusual servomotor. The function starts by working out the number of microseconds of pulse which will be required for each degree of angle. It then calculates the total pulse length in microseconds for the angle required. Finally, it calculates the duty (the PWM value) between 0 and 65535 and uses *servo.duty_u16* to set that pulse width on the servomotor's control pin.

The main *while* loop waits for button presses and changes the angle accordingly. Since many servomotors can't manage the full 180 degrees, the range of angle set is limited to between 10 and 170. If you find that the servomotor judders badly at one end of its travel, you may need to adjust *min_angle* or *max_angle*.

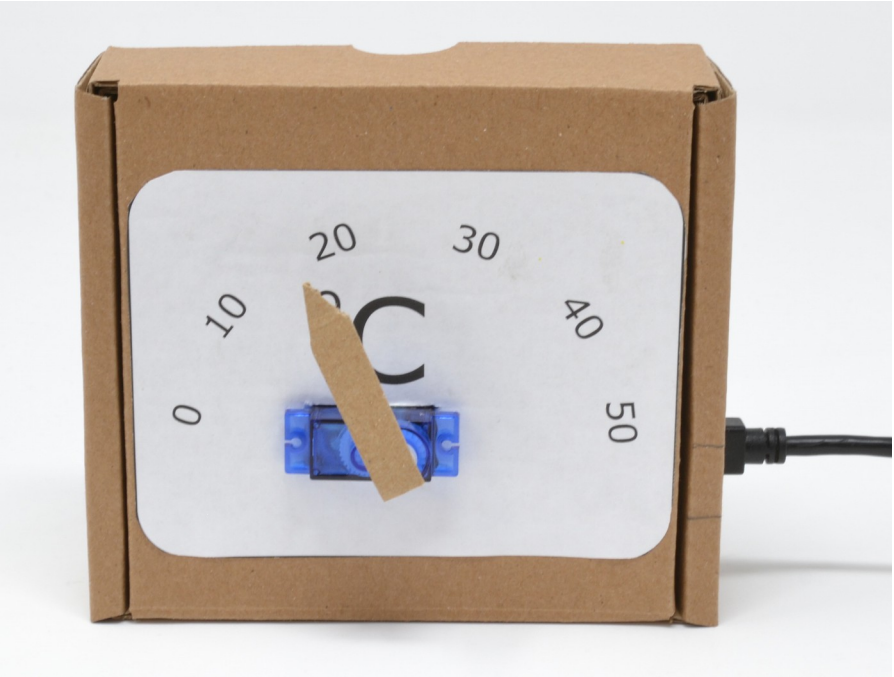
The Electronics

The new part of this project is the servomotor.



Although the servomotor included in the kit will work at 3V, more power is available for it if it is powered from the 5V available over the USB lead. The control connection to the servo is connected via a 470Ω resistor. While not strictly necessary this does mean that if the servomotor tries to draw too much current from the GPIO pin, the Pico will not be damaged.

Project 6. Thermometer



This project uses a servomotor to indicate the temperature, as measured by the temperature sensor built into the Pico's processor. You don't have to put it in a box like this but, if you do, it looks quite smart.

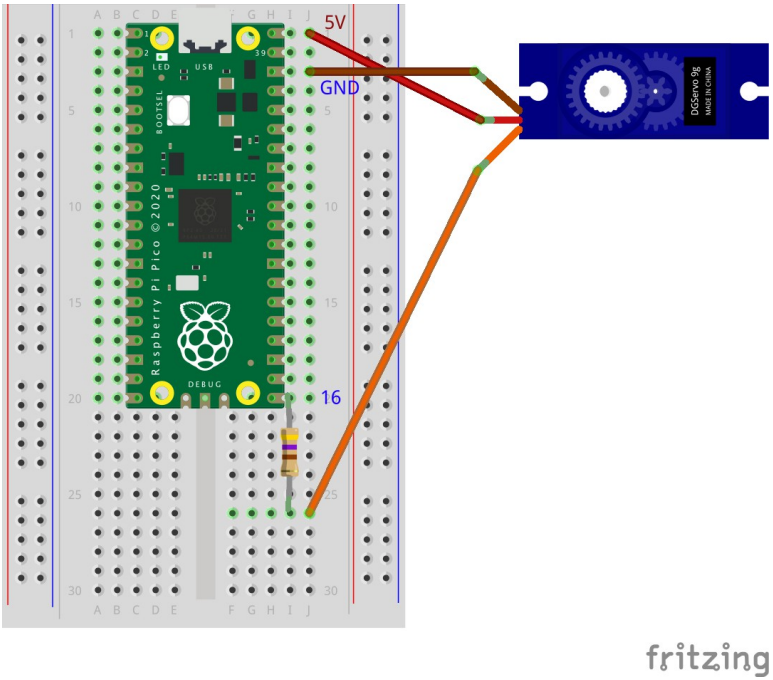
Step 1. Find the parts you need

For this project, you will need the following items:

- Servomotor
- An arm for the servomotor
- A 470Ω resistor (yellow, violet and brown stripes)
- Three jumper wires
- Small brown box (like the one this kit came in)
- Scissors and paper glue
- Craft knife

Step 2. Build the breadboard.

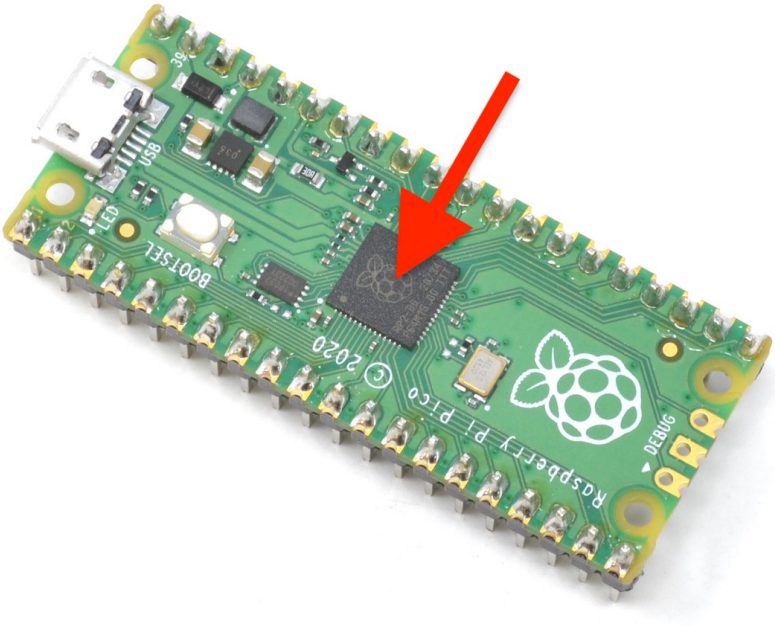
Wire up the breadboard and connect it to your Raspberry Pi using the diagram below as a guide.



If you have just made Project 5, then all you need to do is pull out the switches and jumper wires from the left of the board, leaving just the servomotor and resistor.

Step 3. Running the Program

Before making an enclosure for this project, it's worth making sure everything works. So, run the program 06_thermometer.py and the servo arm should jump to some position. Try putting your finger on the Pico's processor -- the servo arm should move to a new position as the Pico warms up.

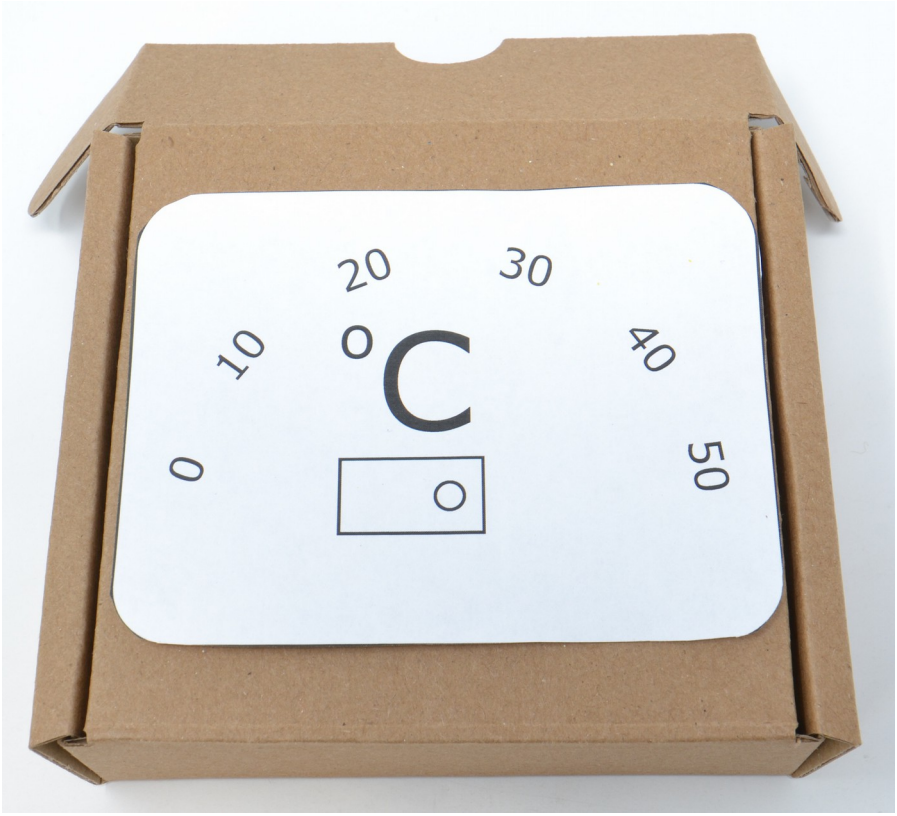


When you take your finger off the Pico's processor chip, you should see the servo arm move back towards its original position.

Step 4. Making an Enclosure

We can see that our thermometer is changing as it warms and cools but, without some kind of scale, we can't use it to tell us the temperature.

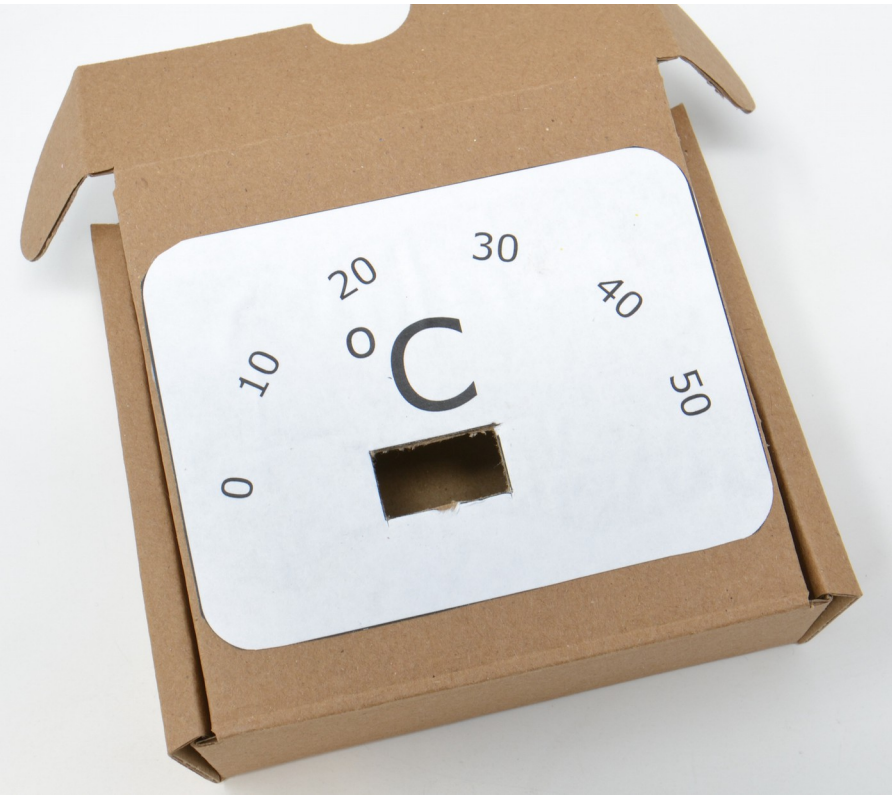
We can make an enclosure using the cardboard box that this kit comes in, or any other smallish box you might have. If you have access to a printer, you can print out one of the scales provided to glue to the box. The file for a Celsius scale is called scale_temp_c.pdf and there is also a Fahrenheit version called scale_temp_f.pdf. You will also find these scales in Appendix F.



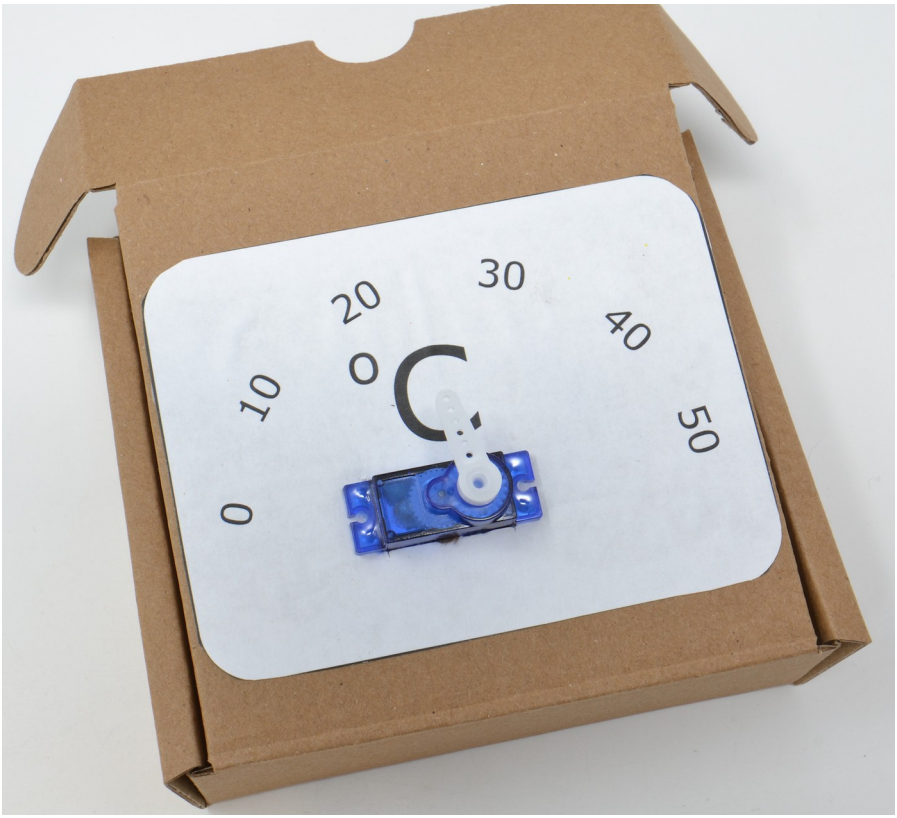
Start by printing out the scale you want to use and cutting out the outline of the temperature scale with scissors. Then stick it onto the card with paper glue (such as PVA).

Allow the glue on the paper scale to completely dry and then using a craft knife, carefully cut out the rectangle where the servo motor is going to go.

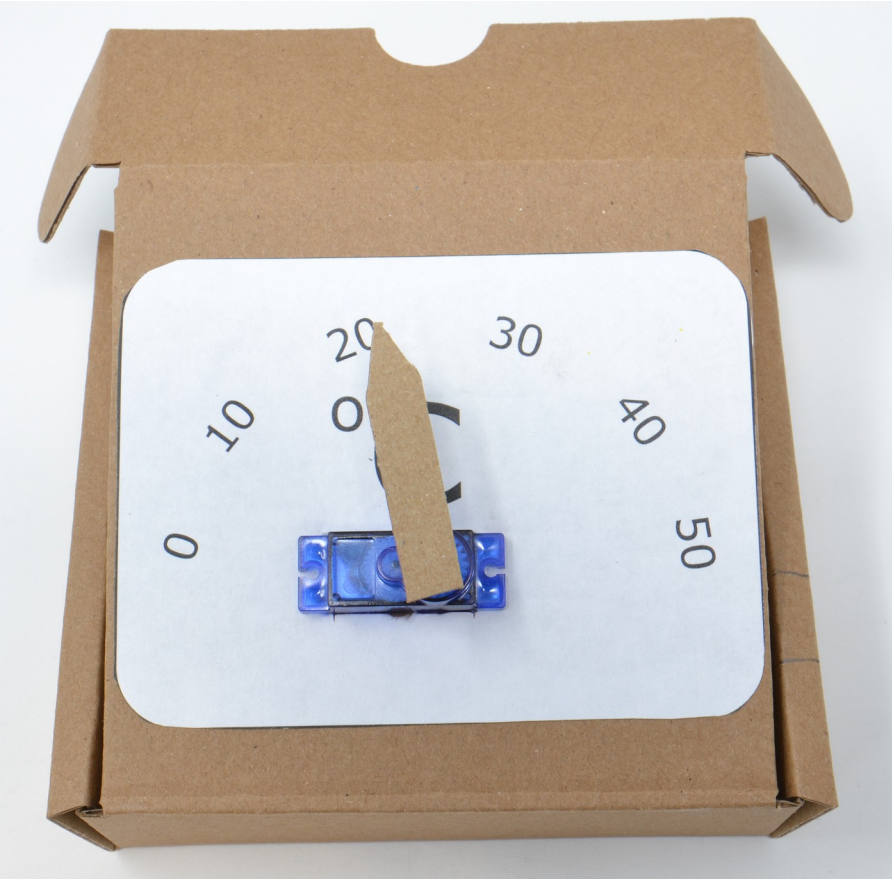
WARNING: Do NOT use a craft knife without adult supervision. If you are an adult, that's fine, you don't need to find another adult, but do be careful, they are nasty sharp things!



We are going to mount the servomotor on the outside of the box, so thread the servomotor's lead through the hole from the front and push the servomotor into place with the shaft to the right. The servomotor will fit quite tightly in the slot, holding it in position.

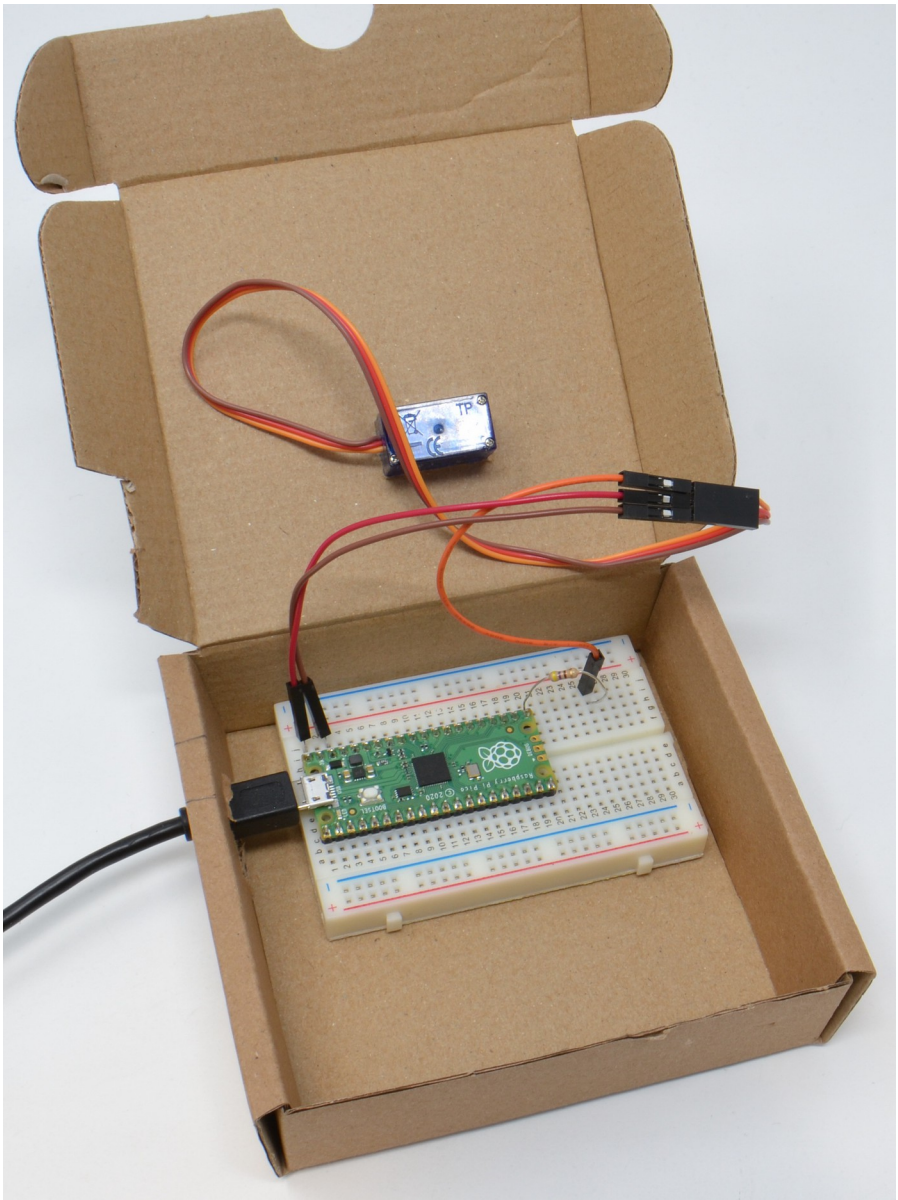


If you want to, find a waste piece of cardboard or other material and make a pointer to glue to the servomotor arm as shown in the photograph on the next page.



Put the breadboard inside the body of the box and reconnect the servomotor. Then make some marks on the side of the box where you will need to make a hole for the USB lead to enter the box.

Cut out a hole for the USB lead and trim the flap on the side of the lid, so that it will shut when the USB lead is through the hole.



Connect the lead to your computer or a USB power supply and your thermometer

should swing into life. Remember, if you are not running the project code from Thonny, you will need to save it onto the Pico as main.py as described on Page 12.

The Code for Project 6

This project uses the same code to control the servomotor as project 5, but in this case, the angle set is calculated from the temperature rather than bing controlled by two buttons.

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

servo = PWM(Pin(16))
servo.freq(50) # pulse every 20ms

temp_sensor = ADC(4)
points_per_volt = 3.3 / 65535

def read_temp_c():
    reading = temp_sensor.read_u16() * points_per_volt
    temp_c = 27 - (reading - 0.706)/0.001721
    return temp_c

def set_angle(angle, min_pulse_us=500, max_pulse_us=2500):
    us_per_degree = (max_pulse_us - min_pulse_us) / 180
    pulse_us = us_per_degree * angle + min_pulse_us
    # duty 0 to 1023. At 50Hz,
    #each duty_point is 20000/65535 = 0.305 µs/duty_point
    duty = int(pulse_us / 0.305)
    servo.duty_u16(duty)

angle = 90
set_angle(90)
min_angle = 10
max_angle = 160
min_temp = 0
max_temp = 50
angle_per_degree_c = (max_angle - min_angle) /
                          (max_temp - min_temp)

while True:
    temp_c = read_temp_c()
    print(temp_c)
    angle = min_angle + (temp_c - min_temp)
              * angle_per_degree_c
    if angle >= min_angle and angle <= max_angle:
        set_angle(max_angle - angle)
```

```
    sleep(0.5)
```

The Pico's internal temperature sensor is connected to ADC 4, an internal pin that is not connected to any of the Pico's header pins. The analog reading that we take from this pin is first converted to a voltage (held in the variable *reading*) and then that voltage is converted to a temperature in degrees C using the formula:

```
temp_c = 27 - (reading – 0.706)/0.001721
```

The numbers in this formula are rather complicated, but he relationship between temperature and voltage of the Pico's internal sensor is something that is defined by the the Pico's designers.

The *while* loop of the program is mostly concerned with converting the temperature value into an angle to which the servomotor should be set, and then setting the servomotor to that angle so long as it is within the range of allowed angles.

Project 7. Thermometer Plus

This project expands on the thermometer built in Project 6 to include a buzzer that will sound if the temperature reaches a certain threshold (25 °C by default). If you boxed Project 6, then you will need to open up the box and add a few components to the breadboard.

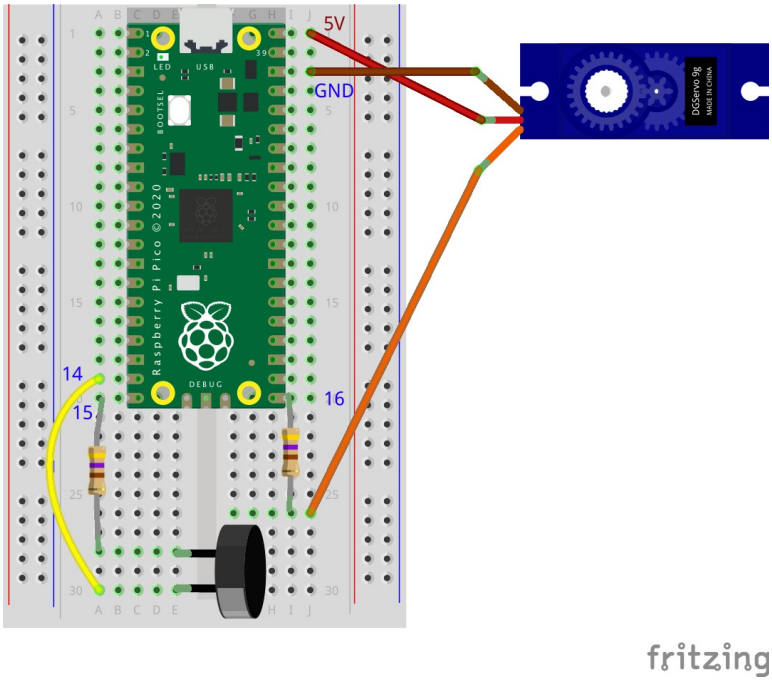
Step 1. Find the parts you need

For this project, you will need the following items in addition to those you used to make project 6.

- A buzzer
- A second 470Ω resistor (yellow, violet and brown stripes)
- An extra jumper wire

Step 2. Build the breadboard.

If you haven't done so, follow the instructions for building Project 6 and then return here.



Add the buzzer and extra resistor as shown. The orientation of the buzzer does not matter as it can be placed either way around.

Step 3. Running the Program

Run the program 07_thermometer_plus.py. Warm the processor up by putting your finger on it. Once the temperature gets above 25 °C, the buzzer should sound. If you can't get the temperature above the default value, you may need to edit the program and change the *set_temp* variable to something lower (see the Code section below if you are unsure how to do this).

The Code for Project 7

The code for this project is similar to Project 6, as the thermometer part of it is unchanged. For an explanation of how the thermometer works, see Project 6.

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

set_temp = 25

buzzerA = Pin(15, Pin.OUT)
buzzerA.high()
buzzerB = Pin(14, Pin.OUT)
buzzerB.low()
buzz = False

servo = PWM(Pin(16))
servo.freq(50) # pulse every 20ms

temp_sensor = ADC(4)
points_per_volt = 3.3 / 65535

def tick(timer):
    if buzz:
        buzzerA.toggle()
        buzzerB.toggle()

def read_temp_c():
    reading = temp_sensor.read_u16() * points_per_volt
    temp_c = 27 - (reading - 0.706)/0.001721
    return temp_c

def set_angle(angle, min_pulse_us=500, max_pulse_us=2500):
    us_per_degree = (max_pulse_us - min_pulse_us) / 180
```

```
    pulse_us = us_per_degree * angle + min_pulse_us
    duty = int(pulse_us / 0.305)
    servo.duty_u16(duty)

angle = 90
set_angle(90)
min_angle = 10
max_angle = 160
min_temp = 0
max_temp = 50
angle_per_degree_c = (max_angle - min_angle) / (max_temp -
min_temp)

Timer().init(freq=1200, callback=tick)

while True:
    temp_c = read_temp_c()
    angle = min_angle + (temp_c - min_temp) *
angle_per_degree_c
    if angle >= min_angle and angle <= max_angle:
        set_angle(max_angle - angle)
    buzz = (temp_c > set_temp)
    print(temp_c)
    sleep(0.5)
```

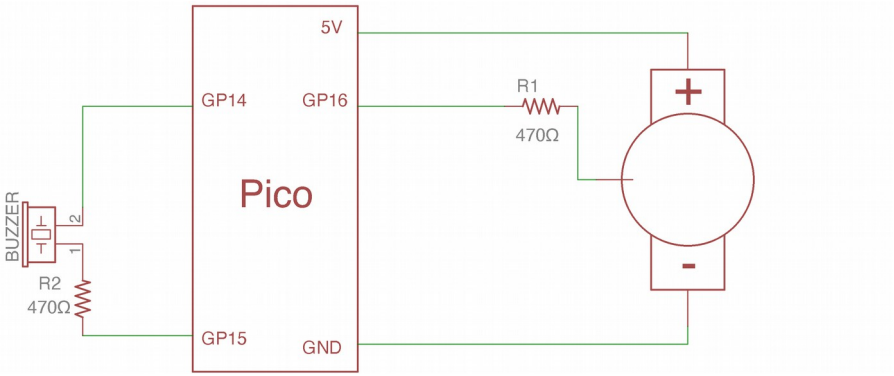
The variable at the beginning of the program, *set_temp,* defines the threshold temperature above which the buzzer will sound. This is the variable to change if you can't get the thermistor up to 25 °C or if you live somewhere warm.

The main difference between this program and the one from Project 6 is the code to control the buzzer. The buzzing is accomplished using two pins (buzzerA and buzzerB) to drive the piezo buzzer. When one pin is high, the other is low and vice-versa.

The function *tick* is called by a timer at 1200Hz (1200 times a second). The function *tick* will only actually flip the pins if the value of the variable buzz is *True*. Something, which is set in the loop as a result of comparing *temp_c* with *set_temp*.

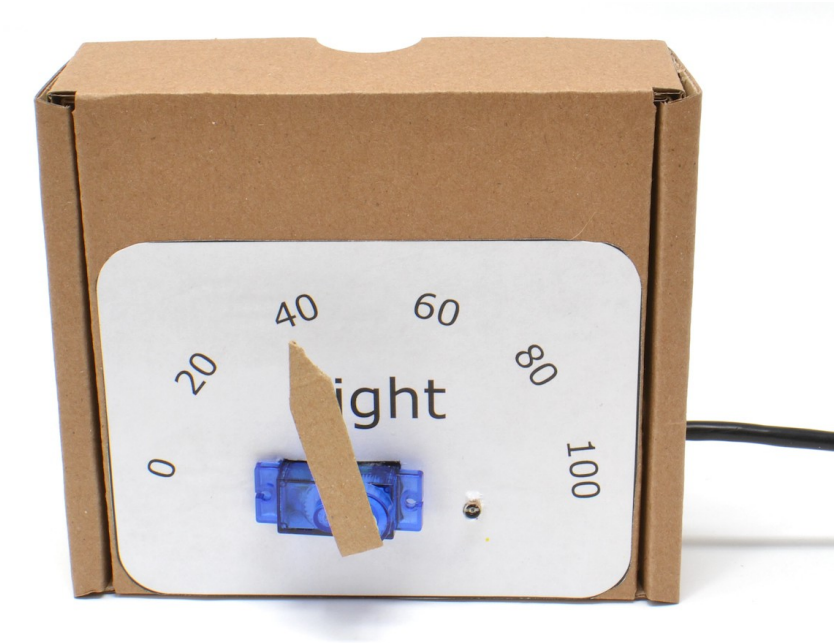
The Electronics

You could drive the buzzer by connecting one of its pins to ground and make the other pin oscillate, but this would be much quieter than the arrangement here, where the two pins drive the buzzer in opposite directions through the whole of the cycle.


5V
GP14
GP16
Pico
R1
470Ω
+
-
BUZZER
2
1
R2
470Ω
GP15
GND

Project 8. Light Meter

In this project, a special type of transistor called a phototransistor is used to measure the light level. The level is indicated using the servomotor, just like in the thermometer project. There is a scale supplied for the light level from 0 to 100, that can be printed out from the file scale_light.pdf if you want to box the project.

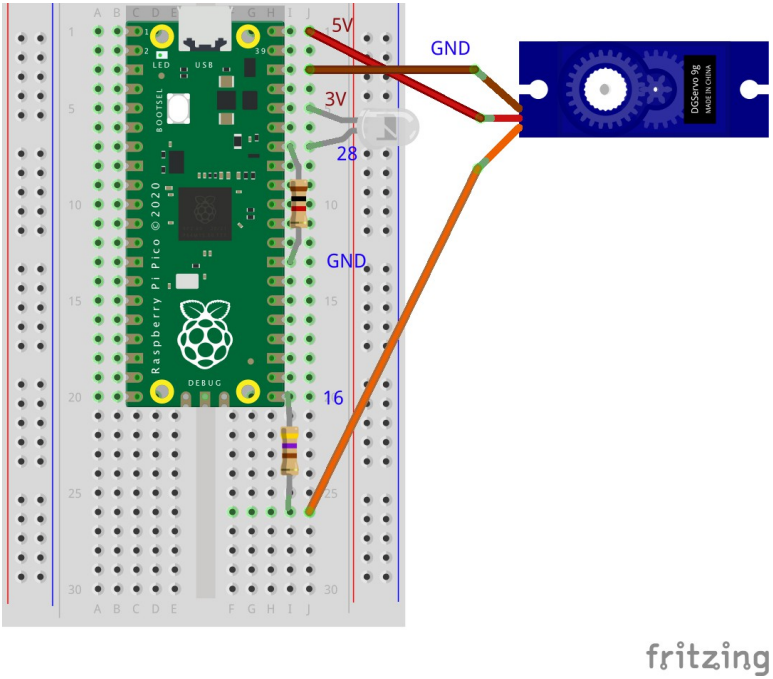


Step 1. Find the parts you need

For this project, you will need the following items in addition to those you used to make project 6.

- A phototransistor
- A 1kΩ resistor (brown, black and red stripes)

Step 2. Build the breadboard.



Follow the instructions for project 6 and then add the phototransistor and 1kΩ resistor. Note, the phototransistor must be placed with the longer lead to row 7 of the breadboard.

Step 3. Running the Program

Unlike the thermometer, which has well defined units of temperature, the scale used for the light level is just a number between 0 and 100.

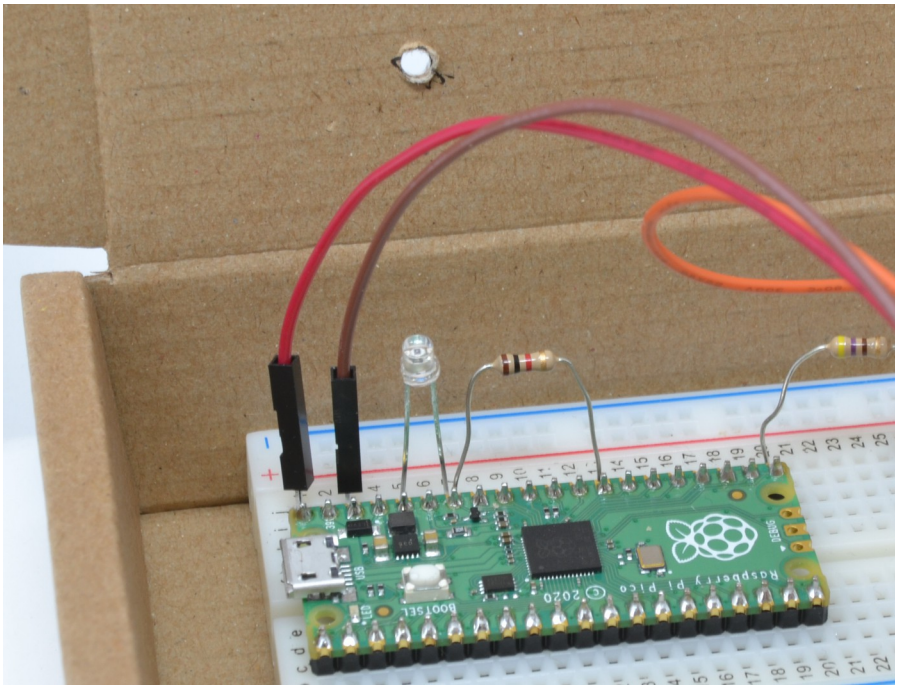
Load the program 08_light_meter.py into Thonny and run it.

You should find that the light level changes when you place your hand over the phototransistor or when you shine a torch on it.

Step 4. Boxing the Project

If you boxed Project 6, then you can follow the same process here, the only differences being that you will need to use the light scale from scale_light.pdf (or Appendix F) and also make a hole in the box for the phototransistor to poke through.

To make the hole in the right place, you may need to fix the breadboard in place and then as you close up the lid, make a pencil mark where the phototransistor meets the lid and make a hole there, so that as the lid closes the phototransistor pushes through the hole.



You can make the hole using a screwdriver or other implement about the same diameter as the phototransistor.

The Code for Project 8

Here is the code for Project 8.

```
from machine import ADC, PWM, Pin
from utime import sleep
from math import sqrt

servo = PWM(Pin(16))
servo.freq(50) # pulse every 20ms

light_sensor = ADC(28)
```

```
dark_reading = 200
scale_factor = 2.5

def set_angle(angle, min_pulse_us=500, max_pulse_us=2500):
    us_per_degree = (max_pulse_us - min_pulse_us) / 180
    pulse_us = us_per_degree * angle + min_pulse_us
    # duty 0 to 1023. At 50Hz, each duty_point
    # is 20000/65535 = 0.305 µs/duty_point
    duty = int(pulse_us / 0.305)
    servo.duty_u16(duty)

def read_light():
    reading = light_sensor.read_u16()
    # print(reading)
    percent = int(sqrt(reading - dark_reading) /
                                          scale_factor)
    if percent < 0:
        percent = 0
    elif percent > 100:
        percent = 100
    return (percent)

min_angle = 10
max_angle = 160
min_light = 0
max_light = 100
angle_per_percent = (max_angle - min_angle) /
                    (max_light - min_light)

while True:
    light_level = read_light()
    print(light_level)
    angle = min_angle + (light_level - min_light) *
            angle_per_percent
    set_angle(max_angle - angle)
    sleep(0.2)
```

In a similar way to Projects 6 and 7, ADC is used to define GP28 as an analog input. Note that only the external pins GP26, 27 and 28 can be used this way, along with the internal pin 4 that is connected to the Pico's built-in temperature sensor.

The variable *dark_reading* should be set to the reading you get when the phototransistor is completely covered. You should not need to change this. However, if you want to check it, uncomment the print statement in the *read_light* function and make a note of the values being displayed in the Shell.

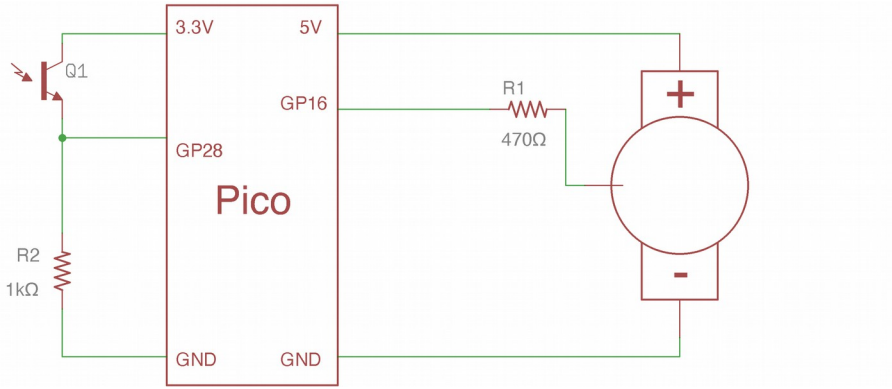
A second variable (*scale_factor*) determines the sensitivity of the meter. Increase

this if you want to make the meter more sensitive.

Reading the light level takes place in the function *read_light*. The analog value from GP28 is read into the variable *reading*. Human perception of light level is not linear -- a bright sunny day is not simply a bit brighter than a normally lit room (as it seems to us) but is actually hundreds of times brighter. Our eyes are extremely good at adjusting to a very wide range of light levels. To compensate for this effect, so that our light meter is not heavily biased to bright light, the square root of the reading is taken after subtracting the value of *dark_reading*. Finally, the function makes sure that this arithmetic has not taken the light reading level outside of the range 0 to 100.

The Electronics

A phototransistor allows more current to flow through as more light falls on it. By using it with a fixed resistor (R2), this current results in a different voltage at GP28, which can be measured and used to provide an indication of the light level.



Project 9. Fader

This project uses a component called a variable resistor (or potentiometer or just pot) to control the brightness of the blue channel of the RGB LED. You will find variable resistors like this often used, say as a volume control on a radio or amplifier.

Step 1. Find the parts you need

To make this project, you will need need:

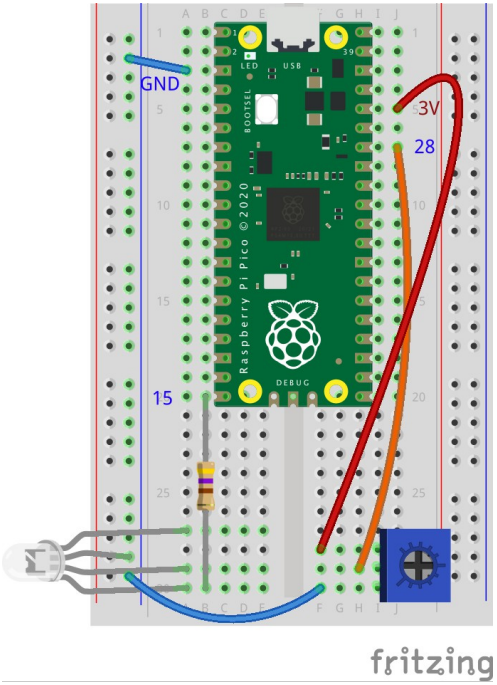
- A 470Ω resistor (yellow, violet and brown) stripes)
- The RGB LED
- The pot
- Four jumper wires

Step 2. Build the breadboard

When assembling the circuit, make sure that the longest leg of the RGB LED goes to the breadboard column marked with a blue line (the negative bus).

The pot is positioned with the pins aligned vertically on the breadboard from rows 28 to 30. It does not matter which way up the pot goes.

Remember that the breadboard holes on the side of the RGB LED are not connected to those near the pot.



Step 3. Running the Program

Open and run the program 09_fader.py in Thonny. When you turn the knob on the pot you will see the brightness of the LED change from off to full brightness.

The Code for Project 9

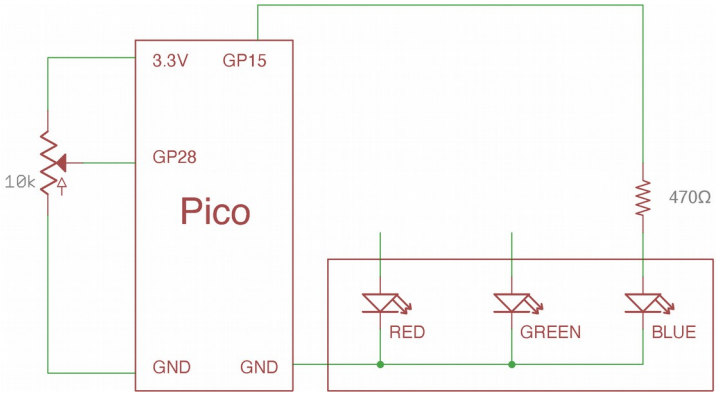
The code for this is very straightforward. An analog reading is made from pin 28 which results in a number between 0 and 65535 depending on the position of the pot's knob. After a delay of 1/10 of a second, which allows the ADC time to recover, the duty of the blue_ch PWM is set to that same value.

The value is also printed into the Python Shell as a way to show how the readings change when you change the position of the pot.

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

blue_ch = PWM(Pin(15))
pot = ADC(28)
blue_ch.freq(1000)

while True:
    reading = pot.read_u16()
    sleep(0.1) # recovery time for ADC
    print(reading)
    blue_ch.duty_u16(reading)
```

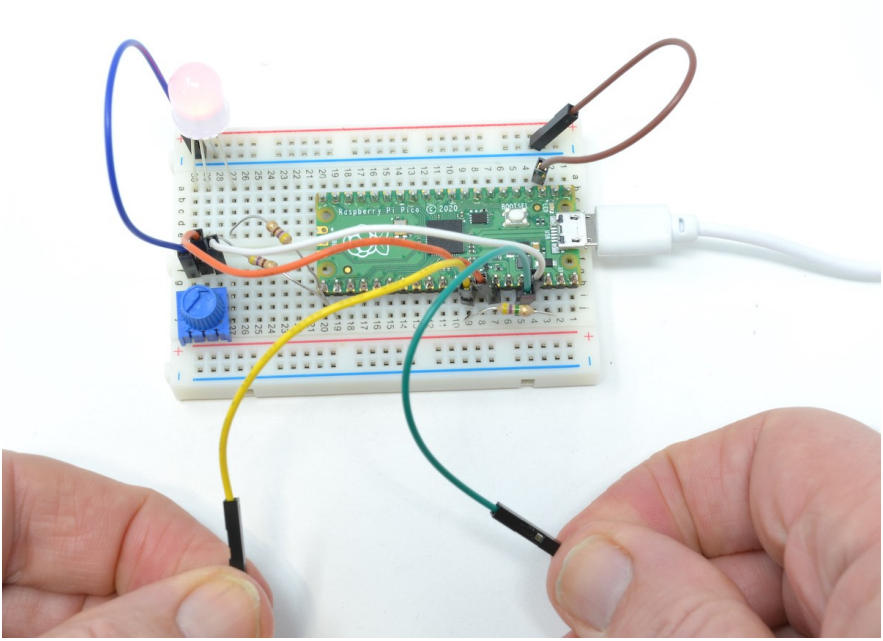
The Electronics

The pot (or variable resistor, if you prefer) has three legs. Two legs are effectively at the end of a resistor with a resistance of 10kΩ, and the third leg is attached to a slider that moves across the variable resistor. As the slider moves towards the 3.3V end of the pot, its voltage will get closer to 3.3V; when the slider is close to the GND end it will be closer to 0V and, guess what, in its center position it will be half way between 0 and 3.3V at about 1.65V.



Project 10. Lie Detector

This is another project that makes use of the pot and RGB LED, but this time just the red and green channels of the LED. When your test subject is holding the metal ends of the green and yellow jumper wires, the LED will indicate whether they are lying or not (well kind of). The pot is used to adjust the lie detector.

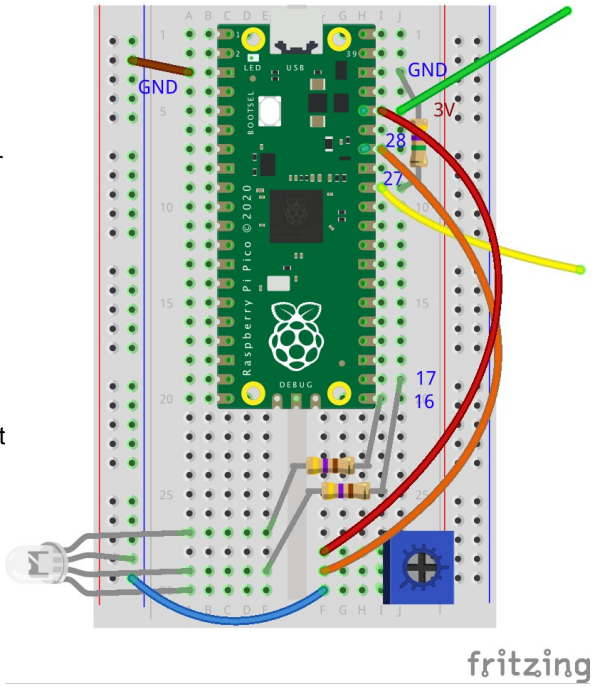


Step 1. Find the parts you need

This project builds on Project 9. The only extra components you need is a second 470Ω resistor and a 4.7MΩ resistor (stripes yellow, violet and green).

Step 2. Build the breadboard

If you have just made Project 9, then you can leave the RGB LED and pot in the same place, but most of the jumper wires and components are in different places. Note that the resistor just to the right of the Pico is the 4.7MΩ resistor.



Step 3. Running the Program

Open and run the program 10_lie_detector.py in Thonny. This is a project that you really need two people for – an interrogator and a subject.

With the subject tightly pinching the metal ends of the green and yellow jumper wires, the interrogator should adjust the pot until the LED goes red and then dial it back a tiny amount until the LED is just green. The interrogator should then ask the subject questions and, if the LED goes red, then this might indicate that the subject is telling lies.

The project uses something called GSR (Galvanic Skin Response). This is just a fancy way of saying the electrical resistance of your skin changes. When you sweat, your skin resistance decreases, and it is this resistance that the lie detector measures.

This method is not at all reliable and you will probably find that you can influence the lie detector in both directions just by how hard you pinch the metal ends of the jumper wires. It's fun though!

The Code for Project 10

Here is the code for Project 10.

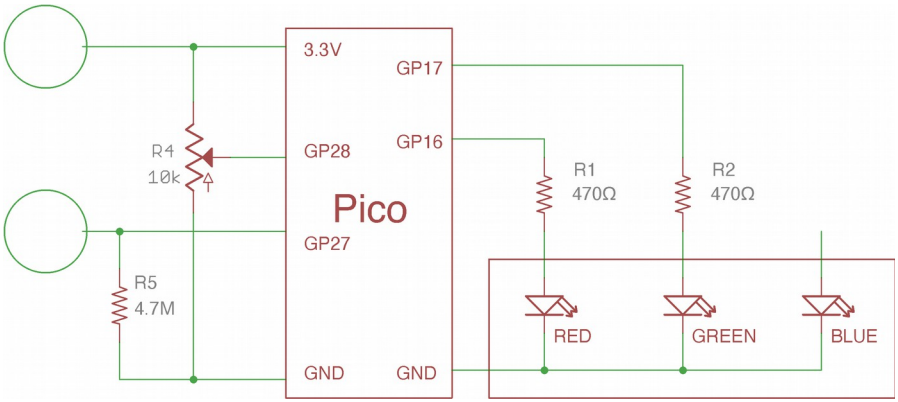
```
from machine import Pin, ADC
from utime import sleep
```

```
red_ch = Pin(16, Pin.OUT)
green_ch = Pin(17, Pin.OUT)
pot = ADC(28)
subject = ADC(27)

while True:
    threshold = pot.read_u16()
    sleep(0.1) # recovery time for ADC
    gsr = subject.read_u16()
    sleep(0.1) # recovery time for ADC
    print("threshold=" + str(threshold) + " gsr=" + str(gsr))
    if gsr > threshold:
        red_ch.high()
        green_ch.low()
    else:
        red_ch.low()
        green_ch.high()
```

In the *while* loop, the position of the pot's knob is read and stored in the variable *threshold,* and then the voltage at the skin resistance sensor is read and stored in the variable *gsr*. These two values are compared and the RGB LED set to green if the skin resistance of the subject is greater than the threshold, otherwise the LED is set to red.

The Electronics

The circuit is very similar to Project 9, except that as well as the variable resistor R4 we have R5, and the variable resistance of the test subjects skin acting as a pot, providing a voltage to be measured at pin GP27.


3.3V
GP17
GP16
GP28
Pico
GP27
GND
GND
R4
10k
R5
4.7M
R1
470Ω
R2
470Ω
RED
GREEN
BLUE

Appendix A. Troubleshooting

Problem: An LED is not working

Solution: Check that the LED is the right way around. The long leg is the positive end and the short leg is the negative end. Make sure that the negative (shorter) leg is connected to GND.

Problem: The project is not working.

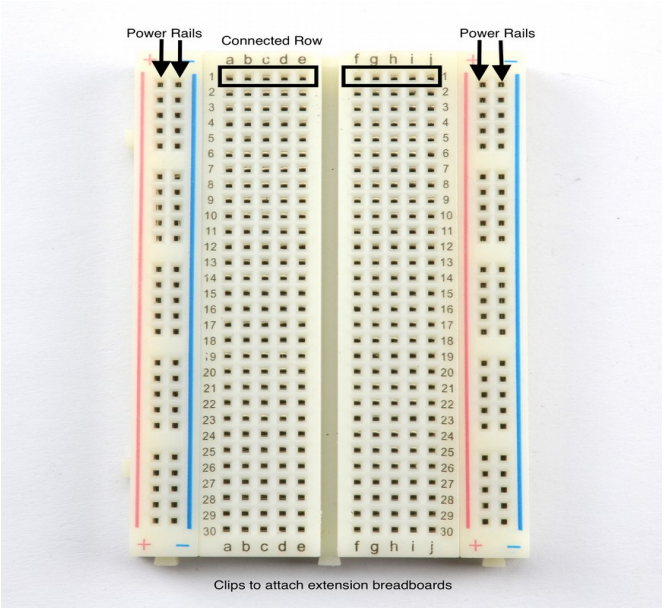
Solution: Carefully check that the legs of each components are not touching each other. Check whether all of the component legs are properly pushed into the breadboard, since sometimes it can be difficult to seat the wire into the breadboard hole properly. Make sure that all of the leads are going to the correct pins on the Pico and to the correct holes on the board. Make sure that all LEDs are the correct way around. Double check that all resistors have the correct value (color stripes). Make sure that you are running the correct program for the project you have set up.

Support

For the most up-to-date help on this kit, see http://monkmakes.com/pico_bb or contact MonkMakes support at support@monkmakes.com

Appendix B. How a Breadboard Works

All the projects in this kit are built on the breadboard into which the Pico and electronic components are fitted.



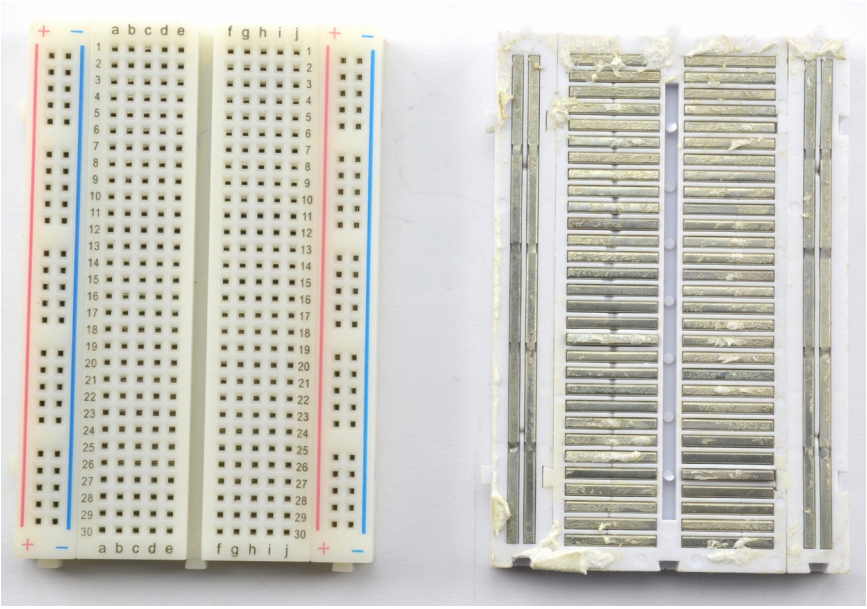
If you connect the projects in exactly the same way as the diagrams then they should work fine. However, it can be quite tedious to use exactly the same holes on the breadboard, so it is worth understanding how the holes are all connected.

The power rails (or busses) on the board are labelled around the outside - two sets of outer rails labelled + and –, and the center holes labelled using numbers 1-30 for rows and a-j for columns.

The bulk of the board in the center is connected **horizontally**. Row 1 is not connected to row 2, but holes 1a, 1b, 1c, 1d and 1e are all connected. The break in the very center of the board also breaks the connections between rows. For example, hole 1e is not connected to 1f. Each row follows the same pattern.

Again, the rows are on the right hand side of the board (1f-1j) are connected together.

If you were to take your breadboard apart, here's what it would look like from the back. You can see the metal clips connecting each section together inside the board.



To use the board, firmly push the metal leads of the components or wires into the holes of the board. Make sure that the leads are in the center of the holes as they can often get stuck on the sides.

The component leads often need to be bent to fit in the right holes, but be careful not to bend the leads too far (especially on the components with shorter leads such as the buzzer) as they can snap off.

Appendix C. The Resistor Color Code

Resistors have little stripes on them that tell you their value. Here's how to read them.

Each color has a value.

There will generally be three colored bands together starting at one end of the resistor, a gap, and then a single band at one end of the resistor. The single band at the far side indicates the accuracy of the resistor value.

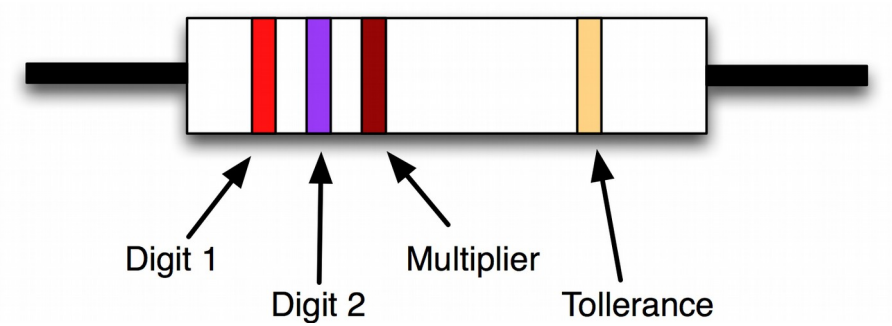
The first band is the first digit, the second the second digit and the third 'multiplier' band is how many zeros to put after the first two digits.

The Gold and Silver stripes at the far end of the resistor are used to indicate how accurate the resistor is, so Gold is +-5% and Silver is +-10%. In other words a Gold (5%) 1000Ω (1kΩ) resistor could have an actual resistance between 950Ω and 1050Ω.

5% is plenty accurate enough for the projects in this kit.

Black	0
Brown	1
Red	2
Orange	3
Yellow	4
Green	5
Blue	6
Violet	7
Gray	8
White	9
Gold	5%
Silver	10%

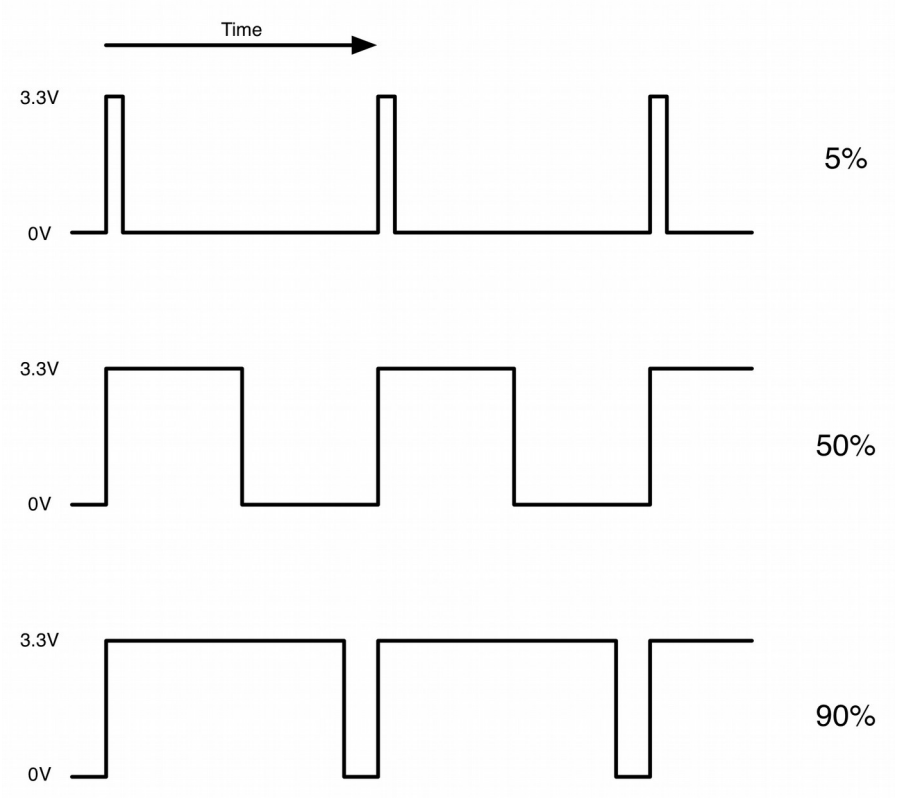
Example:



Red = 2, violet = 7 and brown = 1, so the value is 27 followed by 1 zero or 470Ω and gold indicates 5% tollerance.

Appendix D. Pulse Width Modulation

Pulse Width Modulation (PWM) is a technique that allows you to control the brightness of an LED, or the speed of a motor, by using pulses. In PWM, the pulses always arrive with the same frequency, but the duration of each pulse is varied.

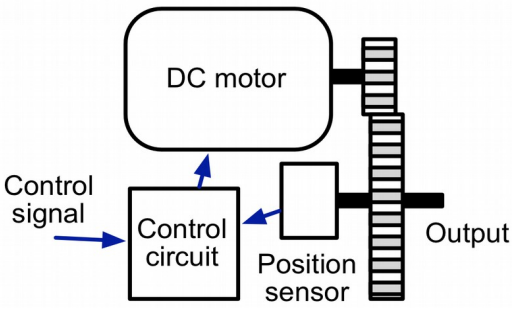


Using the example of an LED, a short pulse will result in the LED being dim. As the *on time* of the pulse increases, so does the brightness of the LED until, when the pulse lasts all the way to the time the next pulse is due, the LED will be on all the time.

In the case of using PWM with an LED, the LED can turn on and off very quickly, and so the LED will actually be flashing but at a frequency too high for you to be able to see it.

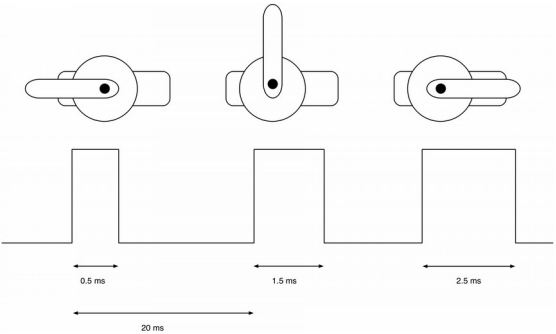
Appendix E. Servomotors

Despite their small size, servomotors are surprisingly complex little machines, controlled using signal pulses. Here's a diagram of how a servo motor works.



A DC motor (normal motor that can rotate continuously) is attached to a gear box, which drives the servo arm, but is also coupled to a position sensor (a pot) that provides feedback to an electronic control circuit that keeps the servo arm at the correct angle.

Control of the position uses a series of pulses arriving every 20 milliseconds.

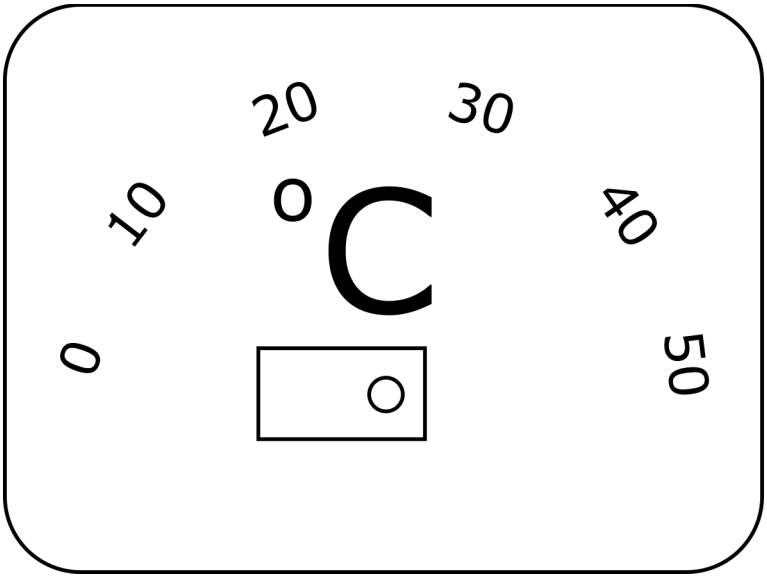


The length of each of these pulses will determine the position of the servo arm. So a short pulse of just 0.5 milliseconds will put the arm at one end of its travel. A pulse of 1.5 milliseconds will put the arm at its center position and a maximum pulse length of 2.5 milliseconds will put it at the other end of its travel.

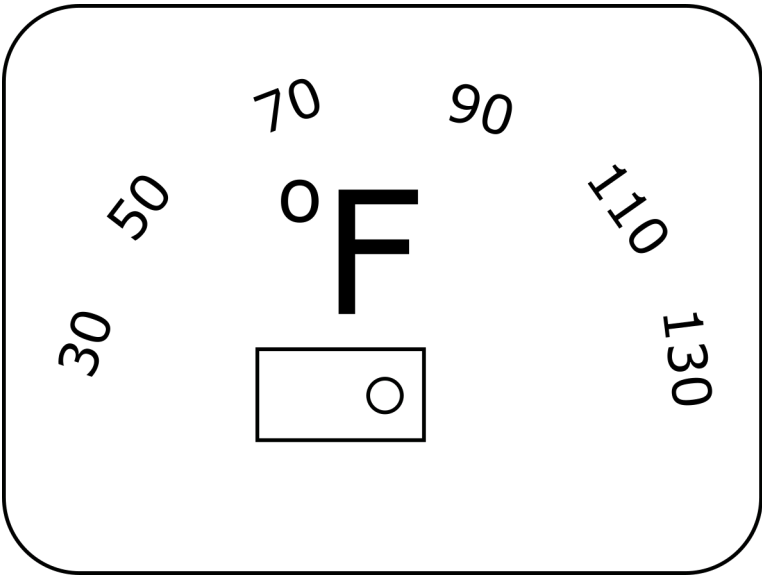
Appendix F. Scales

Feel free to cut out these scales for use on Projects 6 to 8. You will also find these in the code downloads, where the files are called scale_temp_c.pdf temp_scale_f.pdf and scale_light.pdf. There are also vector graphics versions of the files there as well, if you want to customise them.

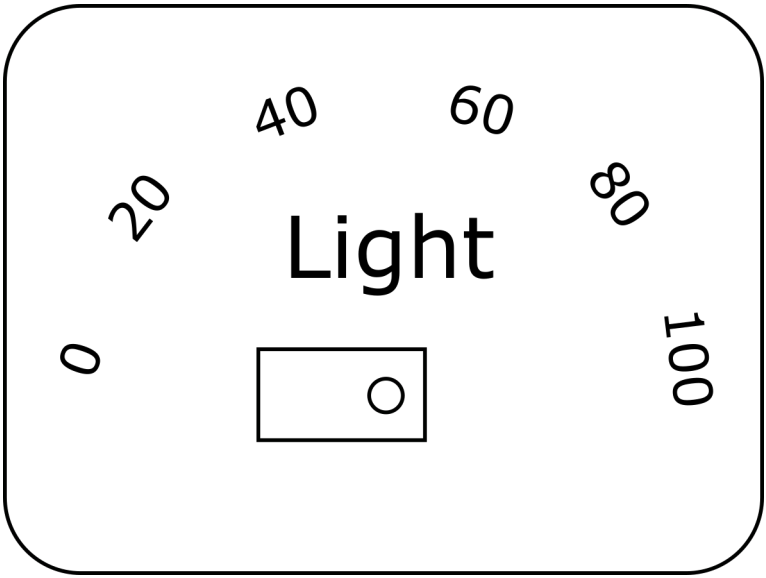
Temperature (degrees C)



Temperature (degrees F)

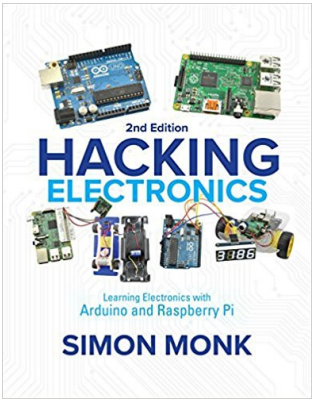


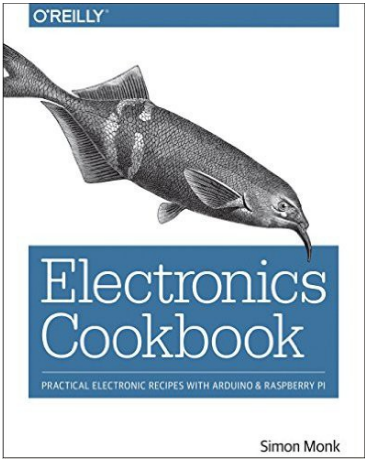
Light

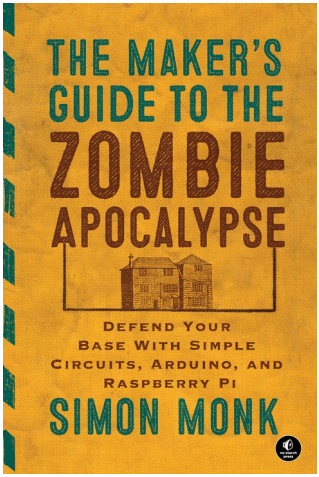


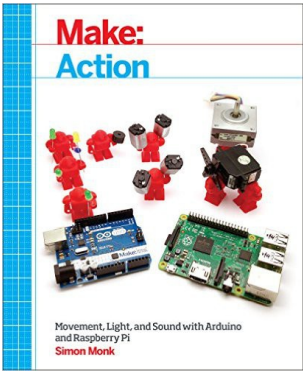
Books

This kit gives you a good set of parts to go off and start developing your own projects. You may find that you want to learn more about electronics. These books were written by the designer of this kit.





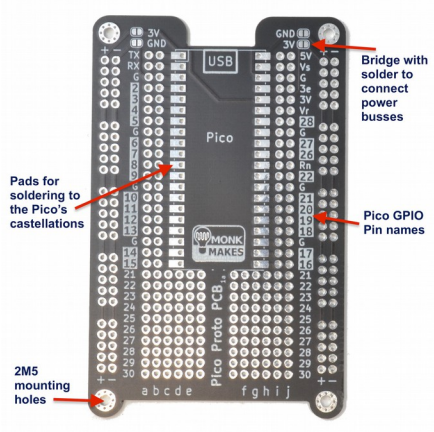


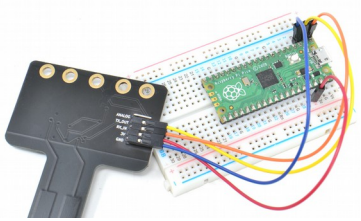


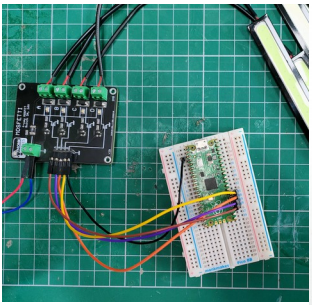
Other Kits

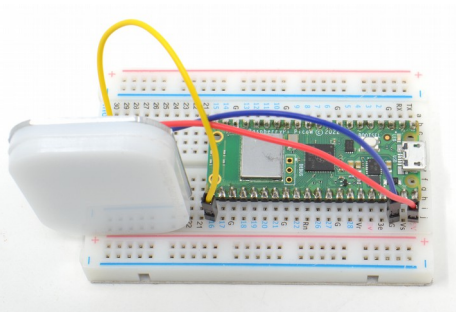
As well as this kit, MonkMakes makes all sorts of kits and gadgets to help with your projects.

Find out more, as well as where to buy here: https://monkmakes.com you can also follow MonkMakes on Twitter @monkmakes.









https://monkmakes.com/pico_proto

https://monkmakes.com/pmon

https://monkmakes.com/mosfetti

https://monkmakes.com/illuminata

Raspberry Pi Pico Pinout

Pins marked with a dot are used in this kit.