Physical Computing with the Pico

Date Created December 2022

Reviewed March 2023

Version 3v0

Last Printed 8 May 2023 10:00

Description

A tutorial to guide teachers and learners through experiments in Physical Computing

This Project is developed with the help of schools in Southern Province, Zambia.

Table of Contents

Physical Computing An Introduction....4
- What is a breadboard and how do we use it?....4
- What is a micro-controller....5
- Raspberry Pi Pico and Pico W Micro-controllers....6
- The Raspberry Pi Pico....7
  - General Purpose Input (GPIO) pins on the Raspberry Pi Pico....8
  - Setting up the Pico on the SB components Pico Breadboard....9
  - Micro-Python....10

Experiment 1....11
- Blinking the Pico onboard LED....11
  - Code for blinking on-board LED....11

Experiment 2....12
- Repeating the on off function from Experiment 1....12

Experiment 3....14
- Testing the Pico Breadboard with Pico installed....14
  - Code for test.py....15

Experiment 4....16
- Using the breadboard and discrete components....16
  - Flashing LED on pin 15....17
  - Using Timer for pin 15....17
- Experiment 5....18
  - Traffic Lights with discrete components....18
- Experiment 6....21
  - Control LED brightness with Pulse Width Modulation or PWM....21
- Experiment 7....23
  - Temperature Measurement....23
    - Programming....26
- Experiment 8....27
  - CSV file of Temperature Measurement Data....27
    - Programming....28
- Experiment 9....30
  - Temperature Changes and Energy....30
- Experiment 10....31
  - Recording Temperature Changes Anywhere....31
    - Programming....32
- Experiment 11....34
  - Using a Proximity Sensor (PIR) as a simple motion detector....34
    - Programming....35
- Experiment 12....36
  - Light level control with LDR and LED....36
    - Programming....36
- Experiment 13....38
  - Temperature control with fan....38
    - Programming....41

Appendix 1 Pico Pins....43

Glossary....44

Python Notes....46

Comments....46
Variables....46
Strings....46
Python Numbers....46
Printing....47
Python Indentation....47
Example Checking a password....48
Conditions....48
Loops....49
while loop....49
for loop....50
Functions....51
Built in functions....52
Sensors....53
The DS18B20 temperature sensor....53
DS18B20 Features and Specifications....53

Physical Computing An Introduction

What is a breadboard and how do we use it?

In our experiments we will be using the SB-Components Pico Breadboard. This board has additional components as well as a breadboard. These components, buttons, LEDs, and a buzzer are pre-wired to the Pico holder. We will be using the Raspberry Pi Pico. The pins of the Pico are also offset for easier connection.

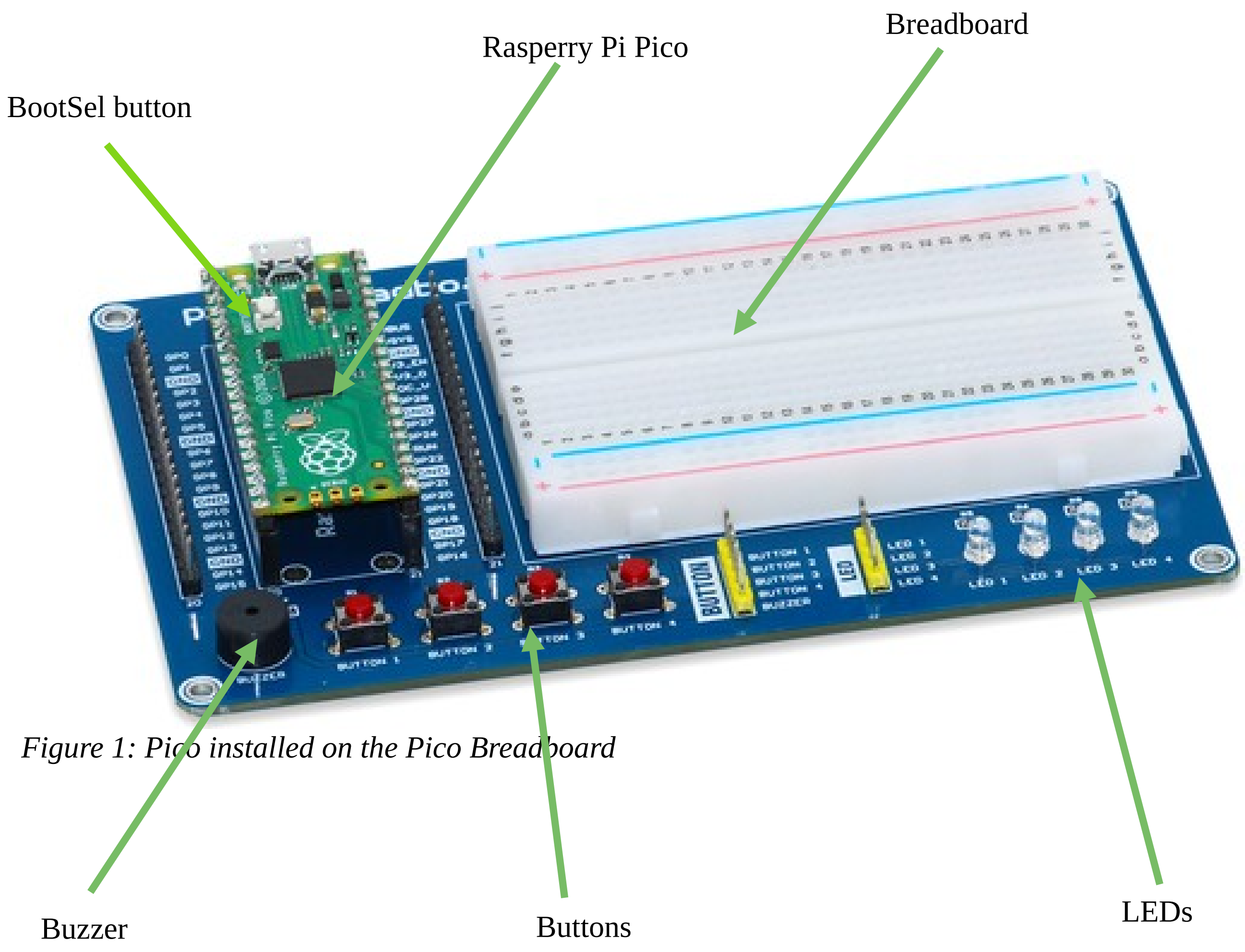


Figure 1: Pico installed on the Pico Breadboard

What is a micro-controller

A micro-controller is single board computer (SBC). Micro-controllers can provide functionality and give intelligence to products and systems. From medical devices to washing machines, many everyday products are controlled by these devices.

A microcontroller is an example of a single board computer (SBC) and is manufactured as an integrated circuit (IC). It can be programmed to perform different processing functions.

Advantages and disadvantages of using microcontrollers

Advantages

The size of a circuit can be significantly reduced. This is because programming replaces physical components.

They can be reprogrammed many times. This allows changes to be made without replacing actual components.

They have pins for connecting several input and output devices, adding to flexibility.

Disadvantages

They often cost more than simple integrated circuits. They are not the best choice for simple systems.

Programming software and hardware is required. This can be expensive for development .

The language of the system must be learned and this adds to development costs.

Raspberry Pi Pico and Pico W Micro-controllers

Although these controllers are based on the same micro-processors, the Pi's RP2040 chip, the Pico W also contains a wireless integrated circuit or IC. Useful for remote working.

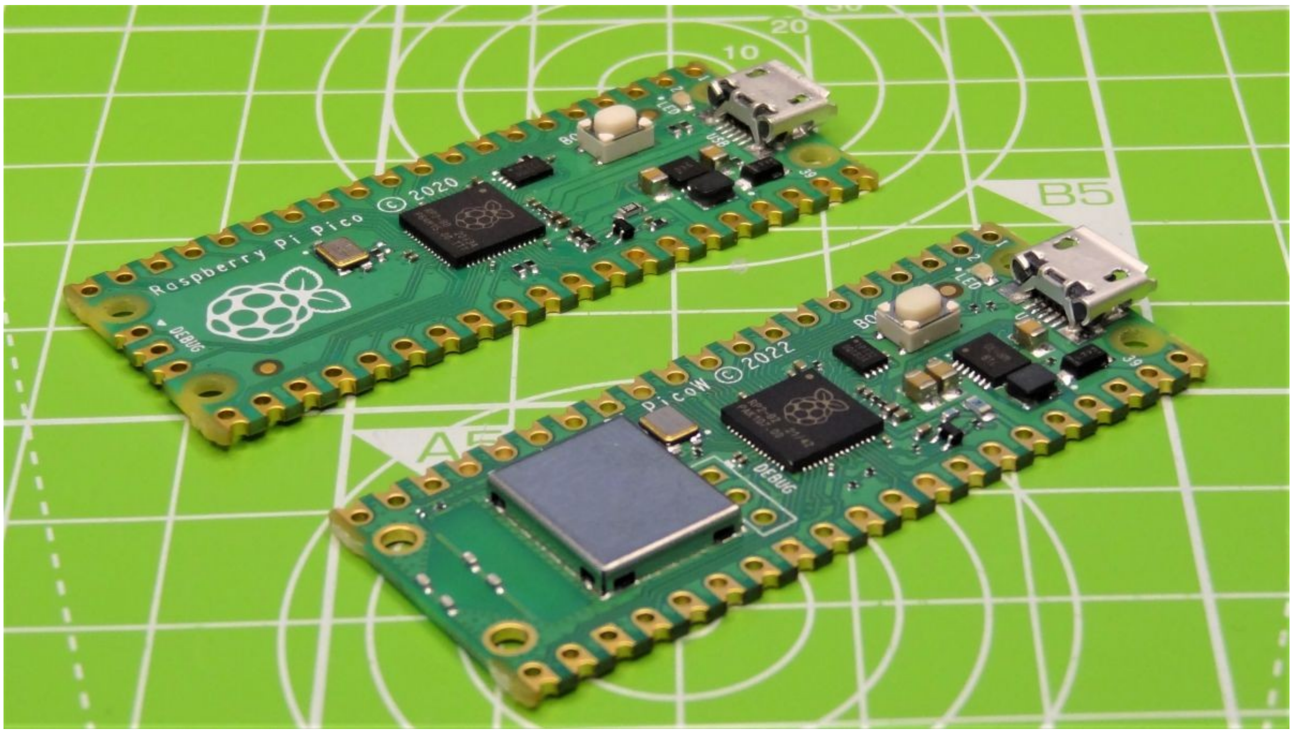


Figure 2: Raspberry Pi Pico and Pico W

The Raspberry Pi Pico

Pico, based on the Pi's RP2040 chip that arrived in 2022.

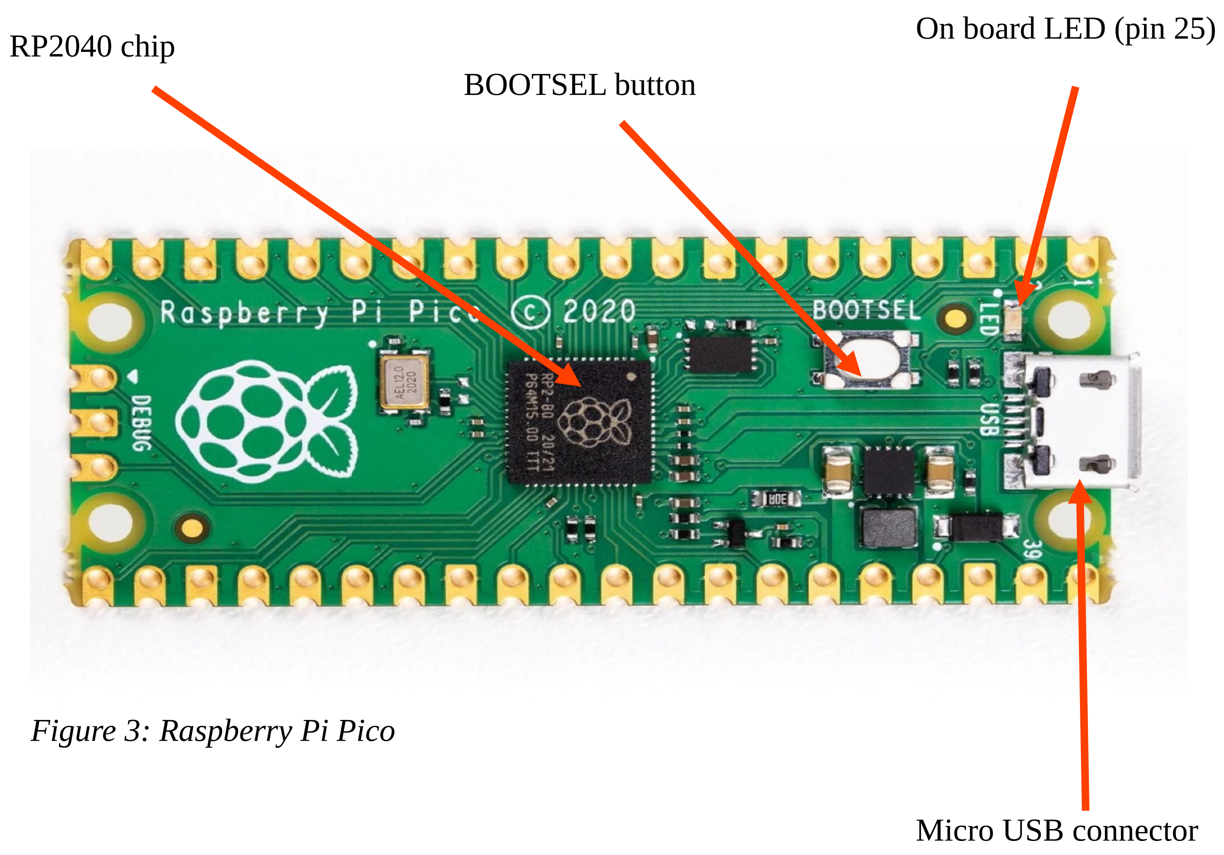


Figure 3: Raspberry Pi Pico

General Purpose Input (GPIO) pins on the Raspberry Pi Pico

Each GPIO pin can be programmed to act in different ways making the Pico a very versatile controller.
GPIO refers to a set of pins on your computer’s or controller’s mainboard. These pins can send or receive electrical signals, but they aren’t designed for any specific purpose. This is why they’re called “general-purpose” IO. This is unlike common port standards such as USB or DVI. With those cables, each pin wired inside the connection has a designated purpose, which is determined by the governing body that created the standard.
GPIO puts you in charge of what each pin actually does. Although there are still different types of pins on the GPIO array.

On the Raspberry Pi and Pico you’ll find a few types of pin:

Pins that provide power at typical voltages such as 3.3V or 5V. This is to power connected devices that don’t have their own power source, such as a simple LED.

Ground pins that do not output power, but are necessary to complete some circuits.

GPIO pins, which can be configured to send or receive electrical signals.

Special purpose pins, which vary based on the specific GPIO in question.

GPIO implementations can vary in the exact details on a per-device basis, but the idea is always to allow users to receive or send an electrical signal to almost anything.

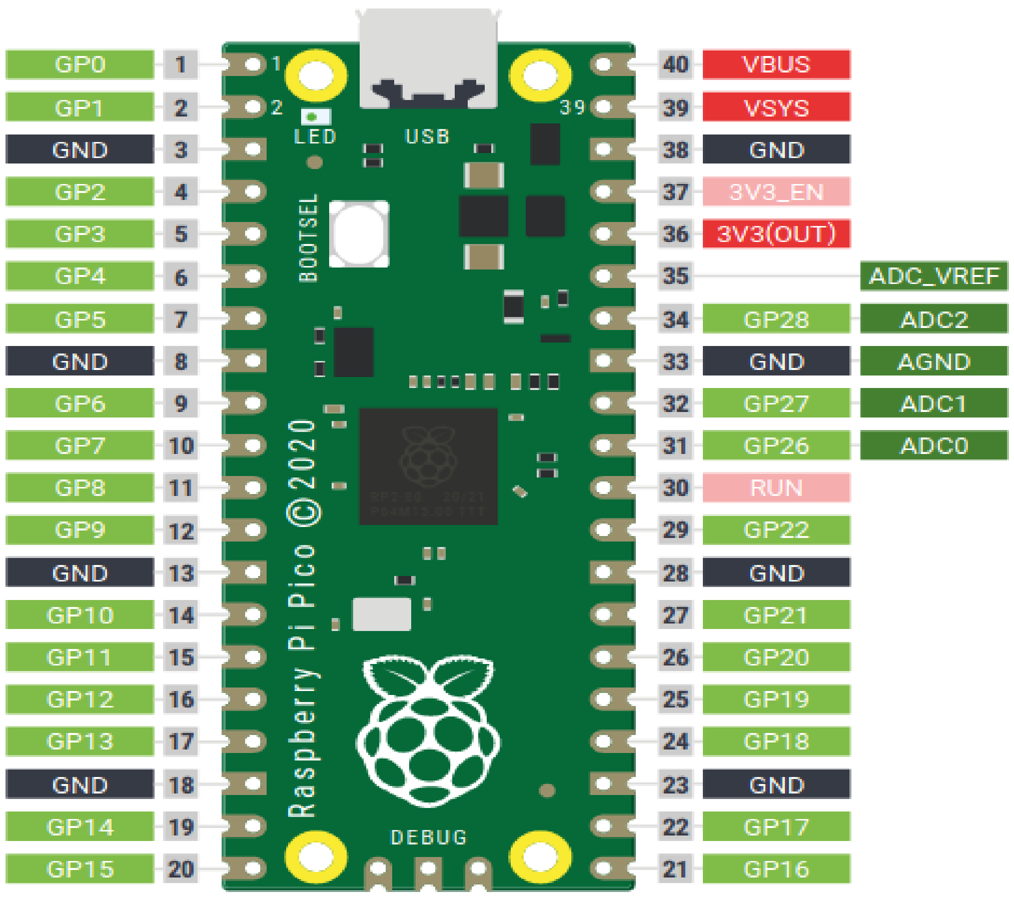


Figure 4: Pico GPIO

Setting up the Pico on the SB components Pico Breadboard

If the Pico hasn't come pre-installed on the SB Pico Breadboard push the Pico pins down into carrier. Make sure the USB connector is at the top of the board.

To install MicroPython on your Pico , you can follow the instructions on the Raspberry Pi website:

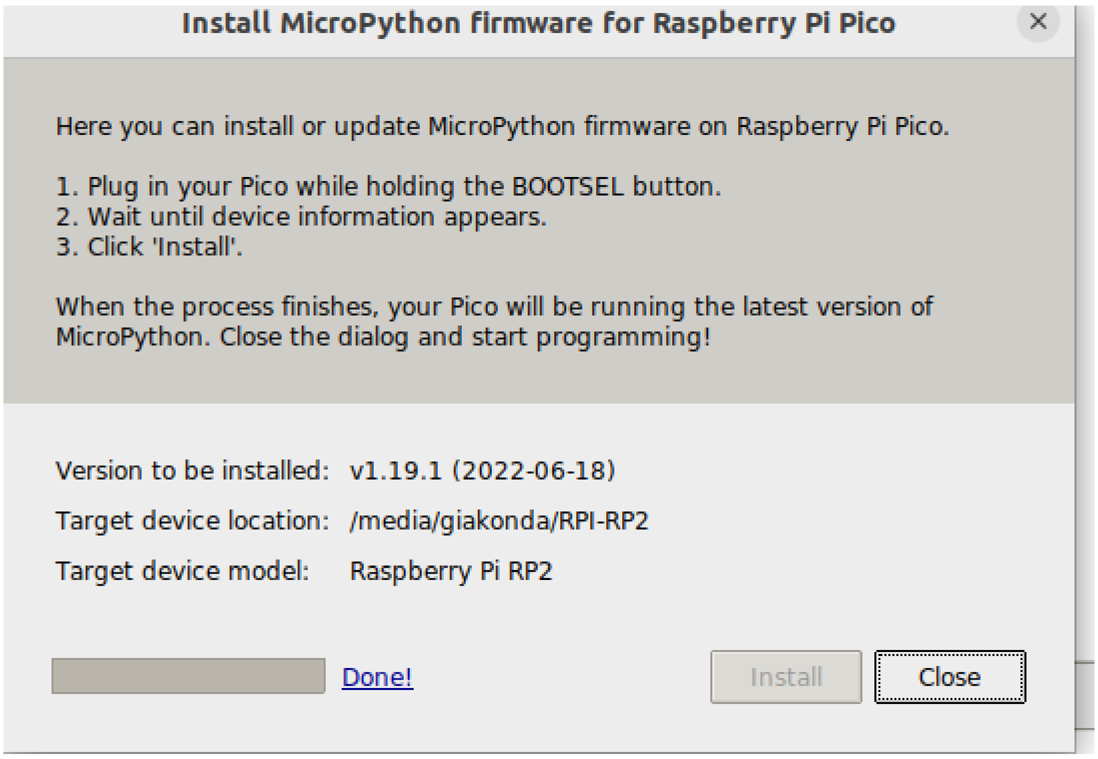
https://micropython.org/download/rp2-pico/rp2-pico-latest.uf2

You can program your Pico by connecting it to a computer via USB, then dragging and dropping a file onto it. The Raspberry Pi team have put together a downloadable UF2 file to let you install MicroPython more easily.

NOTE: be sure to download the latest UF2 file for Pico and not the one for Pico W:

https://micropython.org/download/rp2-pico/rp2-pico-latest.uf2

Hold down the white BOOTSEL button on the Pico then connect your Pico to your Pi. You will be asked to open the file manager, do this. Copy the .uf2 to the RP-RP2. It may take a few seconds for the file to copy. Once done the Pico W will install the uf2 file and reboot.



Install MicroPython firmware for Raspberry Pi Pico

Here you can install or update MicroPython firmware on Raspberry Pi Pico.

1. Plug in your Pico while holding the BOOTSEL button.
2. Wait until device information appears.
3. Click 'Install'.

When the process finishes, your Pico will be running the latest version of MicroPython. Close the dialog and start programming!

Version to be installed: v1.19.1 (2022-06-18)

Target device location: /media/giakonda/RPI-RP2

Target device model: Raspberry Pi RP2

Done!

Micro-Python

MicroPython is a full implementation of the Python 3 programming language that runs directly on embedded hardware like Raspberry Pi Pico. You get an interactive prompt (the REPL) to execute commands immediately via USB Serial port, and a built-in filesystem. The Pico port of MicroPython includes modules for accessing low-level chip-specific hardware.

I have added the necessary pico files (pico-latest.uf2) to the Picos that I have but of course they must be added to every Pico. I have discovered that if you use the Pico and when an Install button appears you click it then everything works as it should. (Things can only get better)

The IDE for MicroPython, Thonny, is pre-installed on Raspberry Pi OS (Operating System)

I have used a Raspberry Pi Zero 2 WH for the computer to programme the Pico and it turns out to be excellent.

An IDE, Integrated Development Environment, is an application that contains a text editor, compiler, debugger and a number of other functions that facilitate programming and testing.

With MicroPython, you can learn how to write code to control the Pico’s GPIO pins, blink LEDs, create games, and even control a greenhouse!

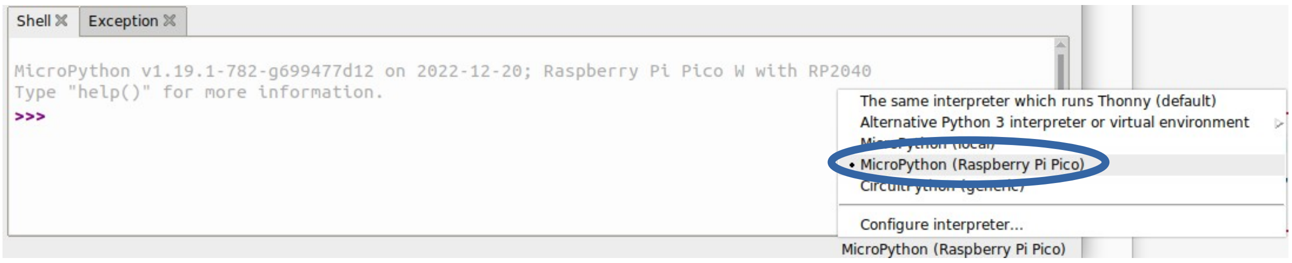
MicroPython quick reference:

https://docs.micropython.org/en/latest/rp2/quickref.html

Experiment 1

Blinking the Pico onboard LED.

Launch Thonny, make sure you select MicroPython (Raspberry Pi Pico) from the bottom right of the Thonny IDE


Shell
Exception
The same interpreter which runs Thonny (default)
Alternative Python 3 interpreter or virtual environment
MicroPython (local)
• MicroPython (Raspberry Pi Pico)
CircuitPython (generic)
Configure interpreter...
MicroPython (Raspberry Pi Pico)

```
MicroPython v1.19.1-782-g699477d12 on 2022-12-20; Raspberry Pi Pico W with RP2040
Type "help()" for more information.
>>>
```

With the Thonny IDE running copy the following code into the editor. When you save the code make sure you save it to the Pico not the local computer. Also don’t forget to add the .py extension to the file name so that is is recognised as a Python program

Code for blinking on-board LED

```
# machine is a python library that gives access to the Pin function
import machine
# time is a python library that gives access to the sleep function
import time
led = machine.Pin(25, machine.Pin.OUT)
led.on()
time.sleep(1)
led.off()
```

Running this program will turn the LED on pause for 1 sec and then turn it off,

Note: If you get a message saying there is an error the chances are you have not used the correct .uf2 file.

Note: Any line starting with a # is ignored when the program is run. It is present to explain something or to make the code more understandable.

Experiment 2

Repeating the on off function from Experiment 1

An upgrade Experiment 1 would be to toggle the LED on and off repeatedly.

This can be done with the use of the Python while statement.

Note the use of the indent to mark the code to be repeated in the while loop. This is a very important part of the Python language.

Also note the colon that marks the while statement.

The while statement expects a condition attached to it which evaluates to either True or False. In this experiment we shall shortcut this process by using True as the result of the evaluation. The result of doing this of course will mean the loop will go on forever, or at least until the program is stopped or the device is turned off

```
# using the Python while loop
# machine is a python library giving access to the Pin() function
import machine
# time is a python library giving access to the sleep() function
import time
led = machine.Pin(25, machine.Pin.OUT)
while True:
    led.on()
    time.sleep(1)
    led.off()
    time.sleep(1)
```

Running this program will turn the LED on if it’s off and off if it’s on. This will continue until the program is stopped.

Note: The number of spaces in the indentation must stay the same. i.e. if you use 4 spaces then all of your indents must have 4 spaces.

Note: External components could be connected with Pico Breadboard kit via the 400 points breadboard or soldered directly

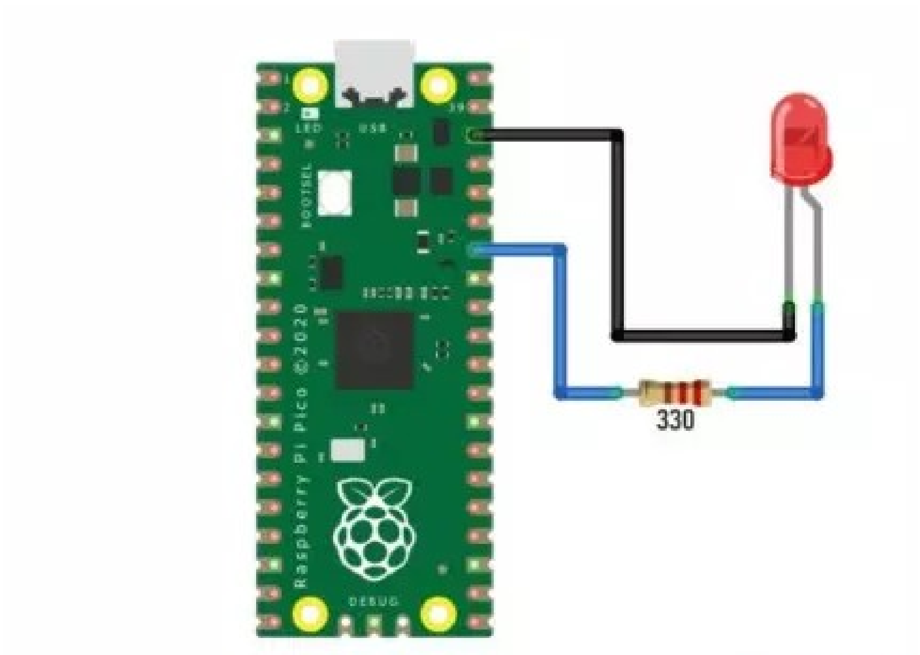


Figure 5: An LED can be connected with a 330 ohms protecting resistor using Pin 28

Experiment 3

Testing the Pico Breadboard with Pico installed

The Basic Logic of the board

Push buttons : Not pressed = Logic 0 , Pressed = Logic 1 (INPUT)

- Led : LED On = 1, LED Off = 0 (OUTPUT) Meeting ID: 219 648 0677

Passcode: Siavonga

-
- Buzzer : Buzzer On = 1, Buzzer Off = 0 (OUTPUT)
- USB Cable to connect Pico to a computer with Thonny installed
- Jumper Cables

Steps :

- Connect Raspberry Pi Pico onto the header of Pico Breadboard Kit.
- Connect the USB cable to the Raspberry Pi Pico USB port.
- Use jumper cables to connect Switches , LED's and Buzzer with Raspberry Pi Pico GPIO headers. (see diagrams following)
- Now use example code "test.py" from pico breadboard kit's GitHub repository in any MicroPython supported IDE (preferred Thonny IDE).
- Copy paste code into the IDE and choose interpreter as MicroPython (Raspberry Pi pico).
- Click on green play button to run the experiments that follow on Pico Breadboard Kit.

Note: External components can also be connected with Pico Breadboard kit via the 400 points breadboard. Just as you would with an ordinary breadboard

The code for the test1.py program

This program tests the Pico breadboard. Lights Buttons and Buzzer. Notice how the indents are used to identify statements in Python.

Note: Double check the wiring. It is very easy to connect the wrong pins.

Code for test.py

```
# A program to show how the breadboard works
# Bring into the workspace the Libraries we shall need
# A library called machine which will give access to the pins on
the pico
from machine import Pin # machine library
# A library called utime which will provide time related functions
import utime # utime/microtime library

button1 = Pin(16, Pin.IN)   #connect Button 1 on GP16

led1 = Pin(18, Pin.OUT)     #connect Led 1 on GP18
led2 = Pin(19, Pin.OUT)     #connect Led 2 on GP19
led3 = Pin(20, Pin.OUT)     #connect Led 3 on GP20
led4 = Pin(21, Pin.OUT)     #connect Led 4 on GP21

buz = Pin(17, Pin.OUT)      #connect Buzzer 4 on GP17

while 1:
    b1 = button1.value()    # Button1 value when pressed = 1

    led1.toggle() # if LED is on turn it off and if off turn it on
    led2.toggle()
    led3.toggle()
    led4.toggle()

    buz.toggle() # if buzzer is on turn it off, if off turn it on

    if b1:  # b1 is 1 if pressed or 0 if not pressed
        print('Button 1 Not pressed!')
        utime.sleep(0.5) # so do nothing for .5 sec
    else:
        print('Button 1 Pressed!')
        utime.sleep(0.5)

    utime.sleep(0.5)
```

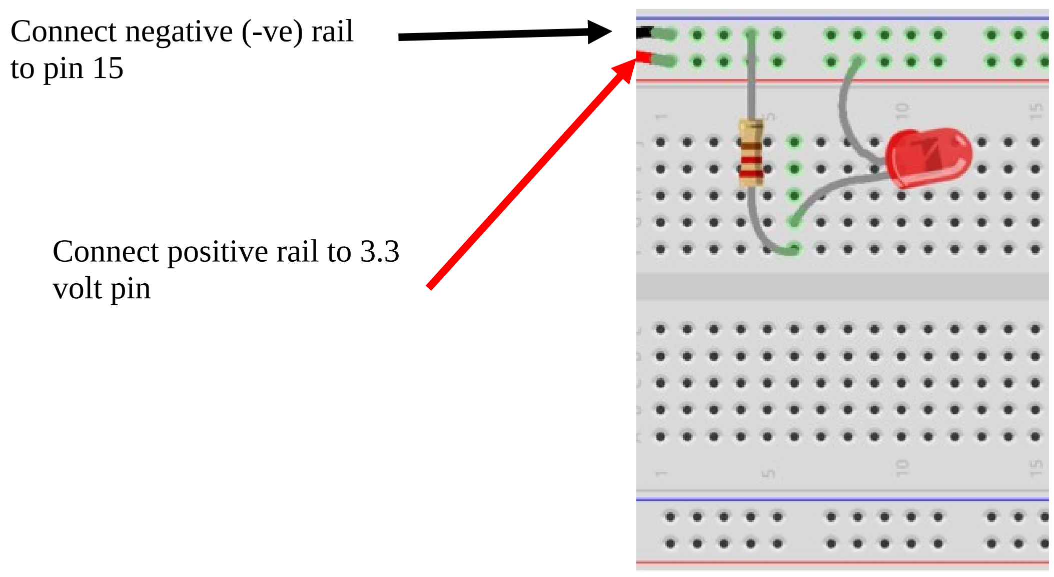
Experiment 4

Using the breadboard and discrete components

In this experiment we are going to blink a red LED. There are several ways to do this, we are going to use two of them.

This requires a resistor between about 50 and 330 ohms, an LED, and a pair of jumper leads to connect up your Raspberry Pi Pico as shown in the image below.

Note: Take great care with placing components on the breadboard it is very easy to use the wrong holes.


Connect negative (-ve) rail to pin 15
Connect positive rail to 3.3 volt pin
1
5
10
15

The red and black cables connect to the Pico. To light the LED simply connect the ground (black) to any of the ground (GND) pins on the Pico. Then connect the red to the 3.3 volt pin on the Pico.

If all we wanted to do was light the LED we could use a battery. However we want to control the LED hence the use of the Pico micro-controller.

The next part of the experiment is to blink the LED. To do this connect the ground line to pin 15 on the Pico

We could now reuse the code in experiment 2 to do this, however we need to change the onboard LED (pin 25) for the LED connected to pin 15

Flashing LED on pin 15

```
# machine is a python library that gives access to the Pin
function

import machine

# time is a python library that gives access to the sleep function

import time

led = machine.Pin(15, machine.Pin.OUT)

while True:

    led.on()

    time.sleep(1)

    led.off()

    time.sleep(1)
```

An alternative way to do this would be to use the **Timer()** function from the machine library.

Using Timer for pin 15

```
from machine import Pin, Timer

led = Pin(15, Pin.OUT)

timer = Timer()

def blink(timer):

    led.toggle()

timer.init(freq=2.5, mode=Timer.PERIODIC, callback=blink)
```

Note: The case of a letter is important in Python.In the above example Timer is different to timer. They have different effects.

Experiment 5

Traffic Lights with discrete components

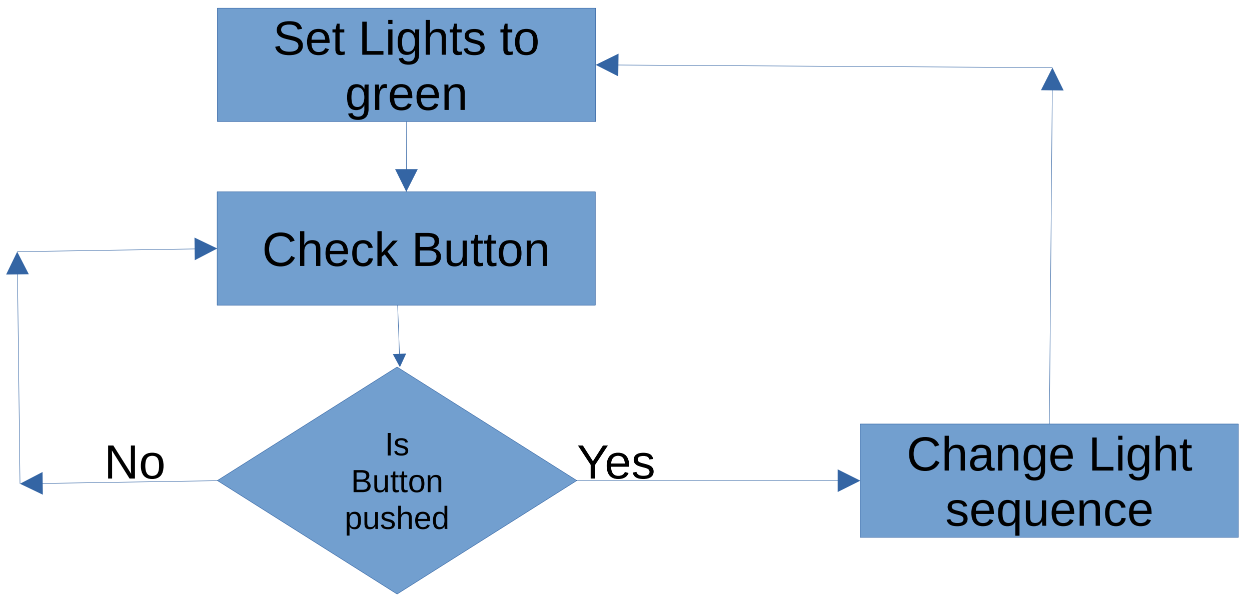
In this experiment we will simulate a Pedestrian controlled road crossing using LEDs and Resistors to mimic the traffic lights and the on board buttons and buzzer to represent the pedestrian control.

The LEDs will be installed on the breadboard.

The sequence we will use is as follows

Set the lights to green.

If the button is pressed set the traffic lights to RED and beep the buzzer for 10 seconds then reset the lights to green



The changing of the lights is done in a particular sequence:

Red

Red

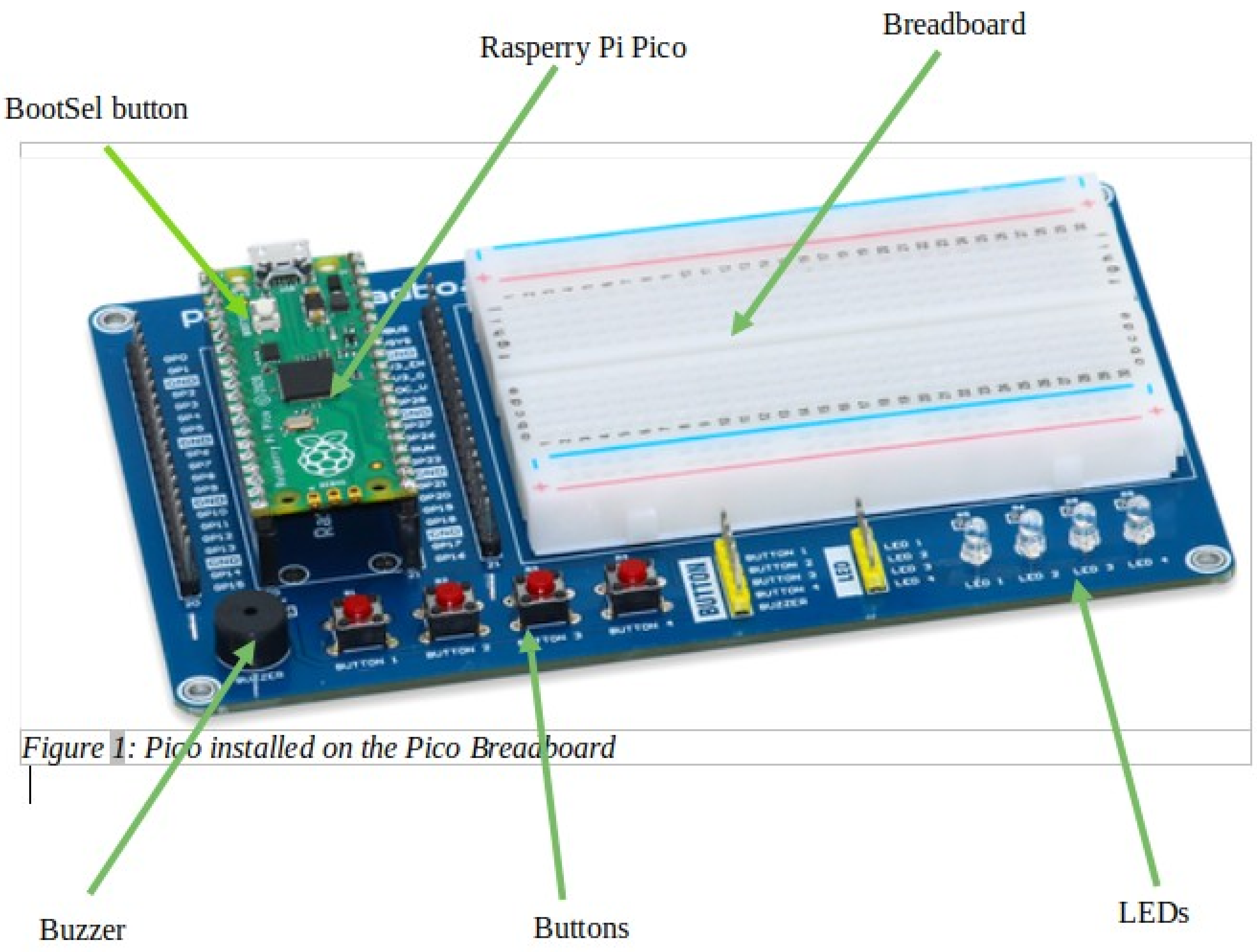


Figure 1: Pico installed on the Pico Breadboard

Amber

Green

Amber

Red

The LEDs will be set up as follows:

- Board GND to breadboard ground rail (using a jumper wire)
- Board GP11 to 220Ω resistor
- Red LED+ to 220Ω resistor (connected to GP10)
- Red LED- to breadboard ground rail (using a jumper wire)
- Board GP14 to 220Ω resistor
- Amber LED+ to 220Ω resistor (connected to GP11)
- Amber LED- to breadboard ground rail (using a jumper wire)
- Board GP13 to 220Ω resistor
- Green LED+ to 220Ω resistor (connected to GP12)
- Green LED- to breadboard ground rail (using a jumper wire)

The Button and Buzzer will be setup as in Experiment 3

Wire the LEDs with 220ohm resistors

Red LED on pin GP10.

Amber LED on pin GP11.

Green LED on pin GP12.

Task: Your task is to write the code to change the lights in the sequence given above when button1 is pressed

Experiment 6

Control LED brightness with Pulse Width Modulation or PWM

Pulse-width modulation (PWM), or pulse-duration modulation (PDM), is a method of reducing the average power delivered by an electrical signal, by effectively chopping it up into discrete parts. The average value of voltage (and current) fed to the load is controlled by turning the switch between supply and load on and off at a fast rate. The longer the switch is on compared to the off periods, the higher the total power supplied to the load. Along with maximum power point tracking (MPPT), it is one of the primary methods of reducing the output of solar panels to that which can be utilized by a battery.
PWM is particularly suited for running inertial loads such as motors, which are not as easily affected by this discrete switching, because their inertia causes them to react slowly. The PWM switching frequency has to be high enough not to affect the load, which is to say that the resultant waveform perceived by the load must be as smooth as possible. This allows you to give analogue behaviours to digital devices, such as LEDs. This means that rather than an LED being simply on or off, you can control its brightness.
The main advantage of PWM is that power loss in the switching devices is very low. When a switch is off there is practically no current, and when it is on and power is being transferred to the load, there is almost no voltage drop across the switch. Power loss, being the product of voltage and current, is thus in both cases close to zero. PWM also works well with digital controls, which, because of their on/off nature, can easily set the needed duty cycle.

For this Experiment, you can connect pin 16 to the LED1 pin on the Pico kit board, not the breadboard this time. There is no resistor to connect up as this is taken care of with the pico kit board itself. Of curse you could still use a discreet LED and resistor on the breadboard if you wish.
Open a new file in Thonny and add the following code.

```
from machine import Pin, PWM
from time import sleep
pwm = PWM(Pin(16))
pwm.freq(1000)
while True:
     for duty in range(65025):
     pwm.duty_u16(duty)
sleep(0.001)
for duty in range(65025, 0, -1):
     pwm.duty_u16(duty)
sleep(0.001)
```

Save and run the file. You should see the LED pulse bright and dim, in a continuous cycle.
The frequency (pwm.freq) tells Raspberry Pi Pico how often to switch the power between on and of for the LED.
The duty cycle tells the LED for how long it should be on each time. For Raspberry Pi Pico in MicroPython, this can range from 0 to 65025. 65025 would be 100% of the time, so the LED would stay bright. A value of around 32512 would indicate that it should be on for half the time.
Have a play with the pwm.freq() values and the pwm.duty_u16 values, as well as the length of time for the sleep, to get a feel for how you can adjust the brightness and pace of the pulsing LED.

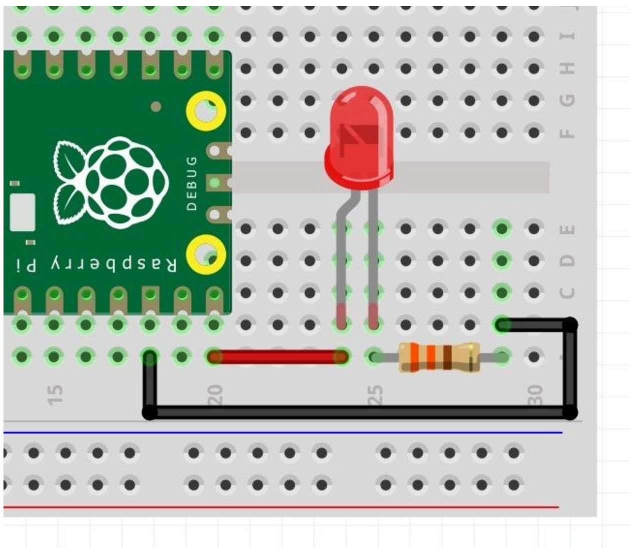


Figure 6: LED wiring for experiment 6

Raspberry Pi Pico	Wire Colour	LED
GPIO15	Red	Anode (Long leg)
GND	Black	Cathode (Short leg)

The LED has a 330 Ohm resistor on the cathode leg, inline with GND. This reduces the amount of current that the LED can consume.

Experiment 7

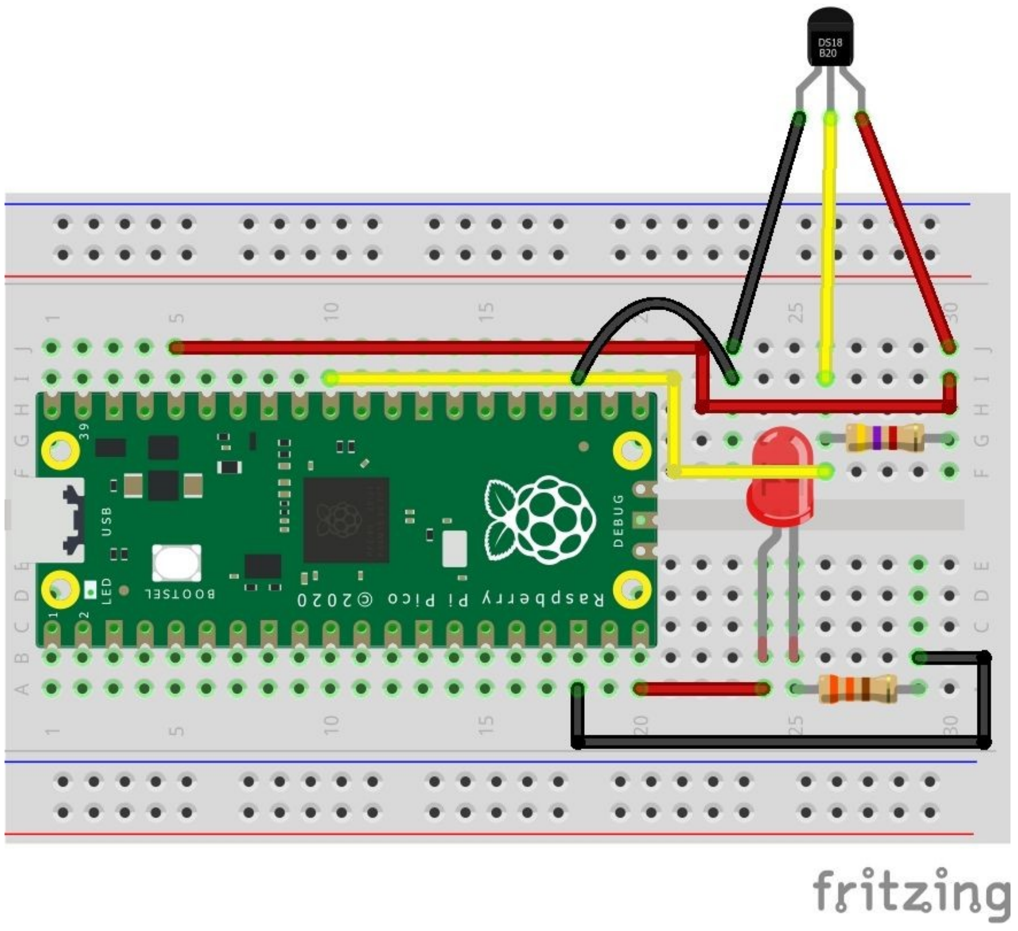
Temperature Measurement

This experiment is taken from Tom’s Hardware

Equipment needed

Raspberry Pi Pico / Pico W, DS18B20 Sensor, LED, 330 Ohm Resistor, 4.7K Ohm Resistor, jumper wires, Breadboard/Pico breadboard kit.

Raspberry Pi Pico	Wire Colour	DS18B20
3V3	Red	VDD
GPIO26	Yellow	Data
GND	Black	GND



All three connections from thermo-probe are made into the Pico breadboard kit, and the wires are used to connect to the Pico. Between the data and 3V3 pins (yellow and red), there is a 4.7K Ohm resistor which is used to pull the data pin high using the 3.3V supplied. This maintains a steady connection between the data pin and Pico. See diagram above.

Program steps are as follows:

```
import onewire, ds18x20, time
```

This line imports three modules of pre-written code. First is onewire, a module that enables the Pico to talk to the one-wire interface of the DS18B20. Next is ds18x20, a module that interprets the sensor data from the DS18B20 and provides us with human readable data. Lastly we import time which is used to pace our project code.

```
from machine import Pin
```

This imports the Pin class from the Machine module. This will enable our code to interact with the components connected to the GPIO.

```
SensorPin = Pin(26, Pin.IN)
alert = Pin(15, Pin.OUT)
```

This creates two objects, SensorPin and alert. SensorPin is the GPIO pin used to connect the data pin from the DS18B20 to the Pico. Alert is the GPIO pin that connects to the anode (long leg) of the LED.

```
roms = sensor.scan()
```

This is necessary in order to find the identity of the probe as the one wire protocol allows for multiple devices to be connected on the circuit. All one-wire devices have a unique registration number stored in ROM that we need to identify before they can be used. This line of code creates an object, roms and uses the sensor object to scan the interface to find our DS18B20 temperature sensor.

```
print(roms)
```

Displays the device ID

```
while True:
     sensor.convert_temp()
     time.sleep(2)
     for rom in roms:
         temperature = round(sensor.read_temp(rom),1)
         if temperature <= 20:
             print("Warning the temperature is",temperature,"C")
             for i in range(10):
                 alert.toggle()
                 time.sleep(0.5)
             else:
                 print(temperature,"C")
```

```
                    time.sleep(5)
```

This block of code uses a while True loop to run the next lines of code in a never ending loop.

Lets look more closely at this never ending loop

```
    sensor.convert_temp()
```

This sets the temperature reading to use Celsius.

Note: that the code within the while True loop is indented to show that it belongs in the loop.

```
    time.sleep(2)
```

This line adds a two second pause to the code. This gives the DS18B20 time to settle before we take a reading.

```
    for rom in roms:
```

Used a for loop to iterate through the returned list of ROMs. As we only have one DS18B20 on the one-wire interface, only one ROM will be stored in the list that we iterate over.

```
        temperature = round(sensor.read_temp(rom),1)
```

This creates an object, temperature, and uses it to store the output of reading the DS18B20 sensor. The output is wrapped in a round() function that will round the returned data to 1 decimal place.

```
        if temperature <= 20:
```

This is a conditional test to check the value stored in the temperature object against a hard coded value. In this example, if the temperature is less than or equal to 20 Degrees Celsius then the condition will evaluate as True, and the next section of indented code will run.

```
            print("Warning the temperature is",temperature,"C")
            for i in range(10):
                alert.toggle()
                time.sleep(0.5)
```

Print a message to the Python Shell that warns the user that the temper is below 20 C and include the temperature value in the sentence and blink the LED 10 times, with a 0.5 second delay.

```
            else:
                print(temperature,"C")
time.sleep(5)
```

The else condition activates if the temperature is greater than 20 C. This condition will print the current temperature and add a five second pause before ending the loop and returning back to the start of the loop.

Save the code to the Raspberry Pi Pico as TemperatureMonitor.py.13.00008

Click on the Run icon to start the code. After a short pause the temperature details will appear in the Python shell. If the temperature is below 20 C, the LED will blink five times to warn us.

Programming

```
# Temperature monitoring
import onewire, ds18x20, time
from machine import Pin
SensorPin = Pin(26, Pin.IN)
alert = Pin(15, Pin.OUT)
sensor = ds18x20.DS18X20(onewire.OneWire(SensorPin))
roms = sensor.scan()
print(roms)
while True:
     sensor.convert_temp()
     time.sleep(2)
     for rom in roms:
         temperature = round(sensor.read_temp(rom),1)
         if temperature <= 20:
             print("Warning the temperature is",temperature,"C")
             for i in range(10):
                 alert.toggle()
                 time.sleep(0.5)
             else:
                 print(temperature,"C")
time.sleep(5)
```

Experiment 8

CSV file of Temperature Measurement Data

A CSV or Comma Separated Variable file is commonly used to transfer data from one device to another or one application to another.

To store any data as a Comma Separated Values file in Python (and its flavours), the following hierarchy is followed.

- Creation of CSV file
- Opening of the file
- Writing data for the first time / Appending data
- Closing of CSV file.

Overview:

```
file=open("data.csv","w")      # creation and opening of a CSV file in Write mode

# Type Program Logic Here

file.write(str(value)+",")     # Writing data in the opened file

# file.flush()                 # Internal buffer is flushed (not necessary if close() function is used)

file.close()                   # The file is closed
```

Detail:

```
Objectname=open("filename.csv","mode")
```

The different modes of opening a file are:

- Write "w"
- Append "a"
- Read "r"
- Read-Write "r+"

Write mode is used for writing data onto a fresh file.

In append mode, data gets added to the existing data.

Using the Read mode, pre-existing data can be accessed but it cannot be modified.

In Read-Write mode, the data can be read and tailored.

Writing Data into the CSV File

Once the CSV file is created and opened in Write mode ("w"), the write() function is used for writing data.

```
Objectname.write(str(value)+",")
```

Using the write function, the value to be stored is given as a string so as to add a separator " , " (Comma) at the end of each value. This process writes and stores the given data in CSV format. In the syntax, the function str() converts the given value into a string. The symbol "+" adds the separator comma (" , ") at the end of each string.

Programming

```
import onewire, ds18x20
import time
import machine
from machine import Pin
rtc=machine.RTC()
SensorPin = Pin(26, Pin.IN)
alert = Pin(15, Pin.OUT)
sensor = ds18x20.DS18X20(onewire.OneWire(SensorPin))
roms = sensor.scan()

print('Probe ID ',roms)
Objectname=open("temp_test.csv","w")
while True:
   timestamp=rtc.datetime()
   sensor.convert_temp()
   time.sleep(2)
   for rom in roms:
      temp = round(sensor.read_temp(rom),1)
      timestring="%04d-%02d-%02d %02d:%02d:%02d"%(timestamp[0:3] + 
timestamp[4:7])

      Objectname.write(timestring + "," + str(temp) + "\n") # 
produces a line of temp need /n

   time.sleep(5)
```

Contents of the file temp_test.csv

Remember this file is on the Pico in CSV (Comma Separated Variable) format.

The format is Date and Time “,” Temperature

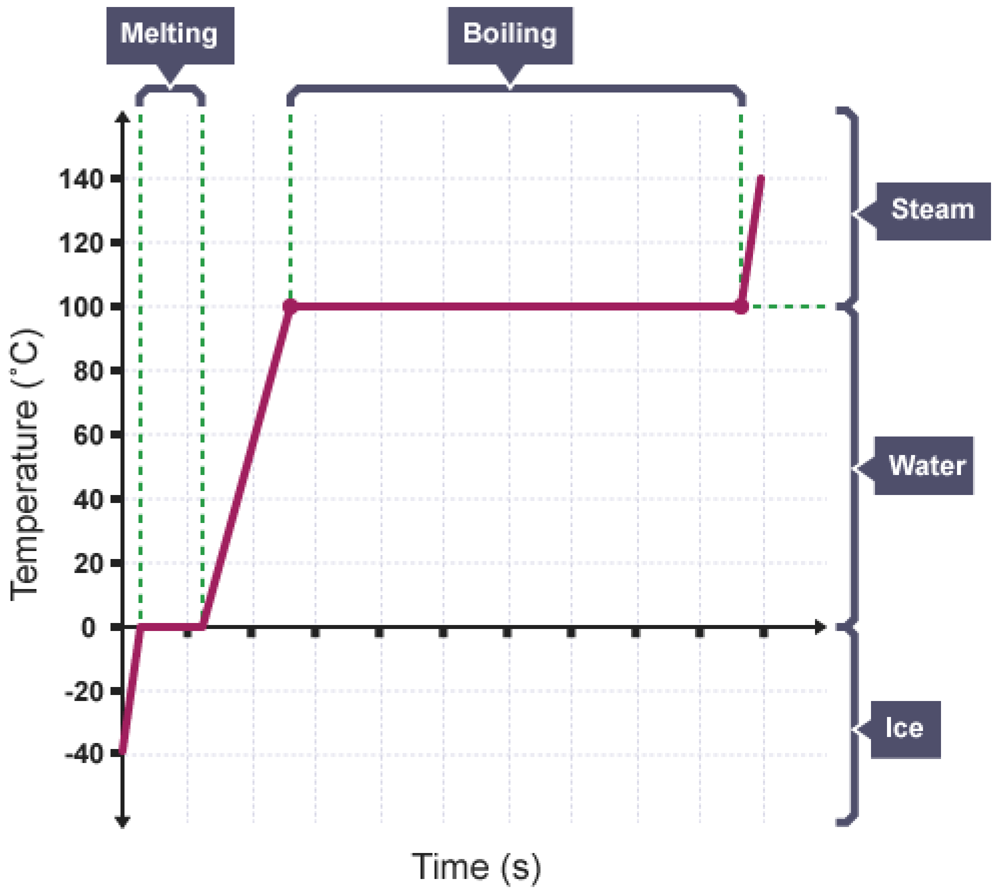
```
2023-02-16 16:27:06,22.0
2023-02-16 16:27:13,22.2
2023-02-16 16:27:20,22.4
2023-02-16 16:27:27,22.5
2023-02-16 16:27:34,22.6
2023-02-16 16:27:41,22.7
2023-02-16 16:27:48,22.8
2023-02-16 16:27:55,22.9
2023-02-16 16:28:02,23.1
2023-02-16 16:28:09,23.1
2023-02-16 16:28:16,23.3
2023-02-16 16:28:24,23.4
2023-02-16 16:28:31,23.4
2023-02-16 16:28:38,23.3
2023-02-16 16:28:45,23.1
2023-02-16 16:28:52,23.1
```

Experiment 9

Temperature Changes and Energy

Place 50 grams of crushed ice straight from the freezer into the calorimeter.

- Place the immersion heater into the central hole at the top of the calorimeter.
- Clamp the thermometer with its bulb in the ice but near the top of the ice.
- Record the temperature of the ice.
- Connect the heater to the power supply and joulemeter, turn it on and record the temperature every 20 seconds.
- Continue until the thermometer bulb is no longer under the level of the water.



Experiment 10

Recording Temperature Changes Anywhere

The Pico has 5 ADC(Analogue-to-Digital Converter) channels. ADC0, ADC1, ADC2, and ADC3 are connected to GP26, GP27, GP28, and GP29 respectively. The fifth ADC channel is connected to an internal temperature sensor. These ADCs measure analogue voltage in the range of 0-3.3 Volts.

We are going to make use of the internal temperature sensor to turn the Pico into a temperature monitoring device.

It requires the Pico, obviously, a power source such as a phone power bank and most importantly the program to run on the Pico to be named main.py. This is important because any program called main.py will automatically run when the Pico is connected to a power supply.

To program the Pico it must be first connected to a laptop running Thonny the Python IDE best suited, in my opinion, to running micro-Python.

The programme/code is on the next page.

This program can measure the temperature of a location ans store the results in a file temps.txt on the pico which can then later be retrieved and investigated.

Note: The program will run automatically once power is applied to the chip provided the program name is main.py

Programming

```
# onboard temp, led and time
#records the temperature and time on the pico
import machine
import utime

sensor_temp = machine.ADC(machine.ADC.CORE_TEMP)
# gets the voltage on the pico from the temperature sensor

conversion_factor = 3.3 / (65535)
# converts the voltage to temperature

led_onboard = machine.Pin(25, machine.Pin.OUT)
led_onboard.value(0)
# sets up the onboard LED

rtc=machine.RTC() # Sets up undocumented real time clock

file = open("temps.txt", "w")
print('Opening temps.txt for writing')
while True:
    reading = sensor_temp.read_u16() * conversion_factor
    timestamp=rtc.datetime()
    temperature = 27 - (reading - 0.706)/0.001721

    timestring="%04d-%02d-%02d %02d:%02d:%02d"%(timestamp[0:3] +
                                            timestamp[4:7])
    file.write(timestring + "," + str(temperature) + "\n")
    file.flush()
    led_onboard.value(1) #turns LED on
    utime.sleep(0.01)
    led_onboard.value(0) #turns LED off

    utime.sleep(10) # wait 10 seconds
```

The temps.txt file will look like this:

```
2023-02-16 15:12:10,14.87265
2023-02-16 15:12:20,14.87265
2023-02-16 15:12:30,14.87265
2023-02-16 15:12:40,15.3408
2023-02-16 15:12:50,15.3408
2023-02-16 15:13:00,15.3408
2023-02-16 15:13:10,15.3408
2023-02-16 15:13:20,14.87265
2023-02-16 15:13:30,15.3408
2023-02-16 15:13:40,15.3408
2023-02-16 15:13:50,15.80894
2023-02-16 15:14:00,14.87265
2023-02-16 15:14:10,14.87265
2023-02-16 15:14:20,15.3408
2023-02-16 15:14:30,15.3408
2023-02-16 15:14:40,15.3408
2023-02-16 15:14:50,15.3408
2023-02-16 15:15:00,14.87265
2023-02-16 15:15:10,14.87265
2023-02-16 15:15:20,15.3408
2023-02-16 15:15:30,15.3408
2023-02-16 15:15:40,15.3408
2023-02-16 15:15:50,14.87265
2023-02-16 15:16:00,14.87265
2023-02-16 15:16:11,14.87265
2023-02-16 15:16:21,15.3408
```

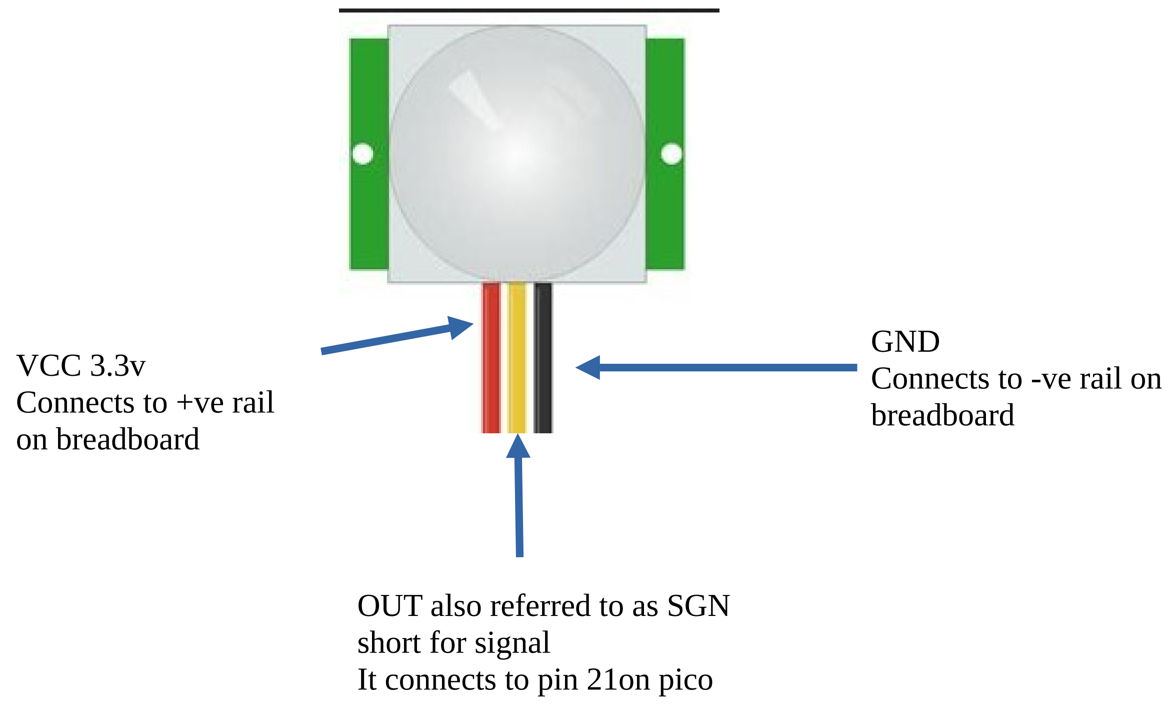
Experiment 11

Using a Proximity Sensor (PIR) as a simple motion detector

In this experiment we will work with a motion sensor and an LED to create a circuit similar to that found in a home intruder alarm.

Equipment

An LED



A 330 Ohm resistor . This is used to ensure the LED is not overloaded

A Passive Infrared (PIR) sensor commonly used in home security systems to detect movement.

Jumper wires

component, a. In this project it will perform that same function, and our code will trigger an LED to turn on when the sensor reports movement.

The PIR sensor has three pins. VCC, OUT and GND. The VCC pin is used to supply 3.3V of power from the Raspberry Pi Pico.

Using a jumper wire, connect VCC from the PIR to the 3.3V pin, (pin 37) which is just next to the resistor.

With another jumper wire connect the OUT pin of the PIR to pin 21 of the Pico.

Connect the GND pin of the PIR to the GND rail of the breadboard.

Programming

The libraries machine and utime enable us to respectively communicate with the GPIO with Pin() and with sleep() to control the pace of the experiment.

```
Complete program

from machine import Pin
import utime
led = Pin(28, Pin.OUT)
pir = Pin(16, Pin.IN, Pin.PULL_UP)
led.low()
utime.sleep(3)
while True:
   print(pir.value())
   if pir.value() == 0:
       print("Movement detected - LED On")
       led.high() #LED turned on
       utime.sleep(5)
   else:
       print("No movement detected")
       led.low() # LED turned off
utime.sleep(0.2)
```

Experiment 12

Light level control with LDR and LED

Connect a light sensor to the Raspberry Pi Pico and write a program that turns on an LED when the light level drops below a certain threshold.

Gather materials: You'll need a Raspberry Pi Pico board, a breadboard, a light-dependent resistor (LDR), a 10K ohm resistor, an LED, a 220 ohm resistor, and some jumper wires.

1. Connect the LDR to the breadboard: Place the LDR on the breadboard and connect one leg to a row of the breadboard using a jumper wire. Then, connect the other leg of the LDR to another row on the breadboard using another jumper wire.
2. Connect the 10K ohm resistor: Connect one leg of the 10K ohm resistor to the same row on the breadboard as the LDR leg connected in step 2. Then, connect the other leg of the resistor to another row on the breadboard.
3. Connect the LED: Place the LED on the breadboard and connect the positive (anode) leg to a row on the breadboard using a jumper wire. Then, connect the negative (cathode) leg to another row on the breadboard using another jumper wire.
4. Connect the 220 ohm resistor: Connect one leg of the 220 ohm resistor to the same row on the breadboard as the negative leg of the LED. Then, connect the other leg of the resistor to another row on the breadboard.
5. Connect the Raspberry Pi Pico: Connect the Raspberry Pi Pico to the breadboard using jumper wires. Connect one wire to pin 34 on the Pico and the other end to the row on the breadboard where the LDR and resistor are connected. Connect another wire to pin 18 on the Pico and the other end to the row on the breadboard where the LED and resistor are connected.
6. Write the code: Open your preferred programming environment and write the following code:

Programming

```
import machine
import time

ldr_pin = machine.ADC(0)
led_pin = machine.Pin(18, machine.Pin.OUT)

while True:
    light_level = ldr_pin.read_u16()
    if light_level < 2000:
        led_pin.on()
    else:
        led_pin.off()
    time.sleep(0.1)
```

This code reads the value of the LDR using the ADC (analog-to-digital converter) on pin 34, and turns on the LED on pin 18 when the light level drops below 2000. The `time.sleep(0.1)` line adds a small delay between each measurement to avoid excessive processing.

8. Upload and run the code: Save the code to a file on your computer, then upload it to the Raspberry Pi Pico using the USB cable. After uploading, the LED should turn on and off depending on the light level detected by the LDR.

That's it! You've successfully connected a light sensor to the Raspberry Pi Pico and written a program to control an LED based on the sensor's readings.

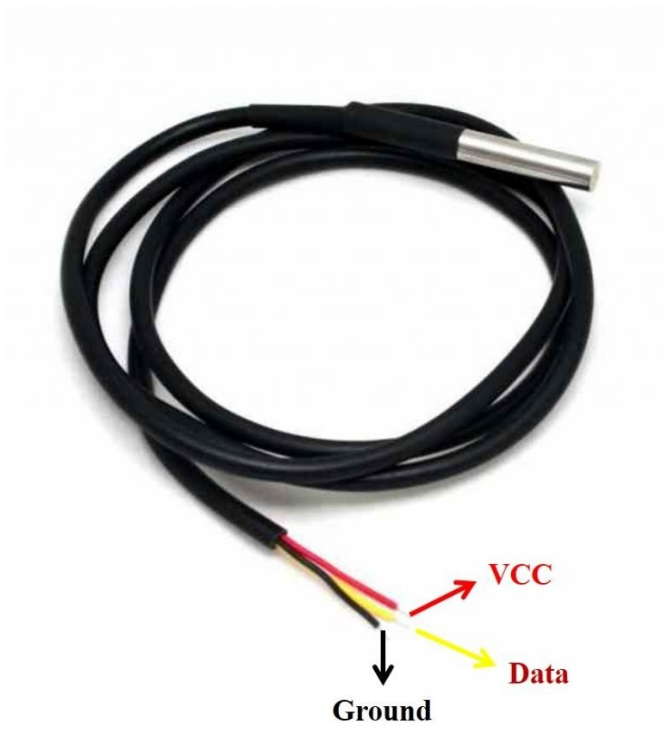
Experiment 13

Temperature control with fan

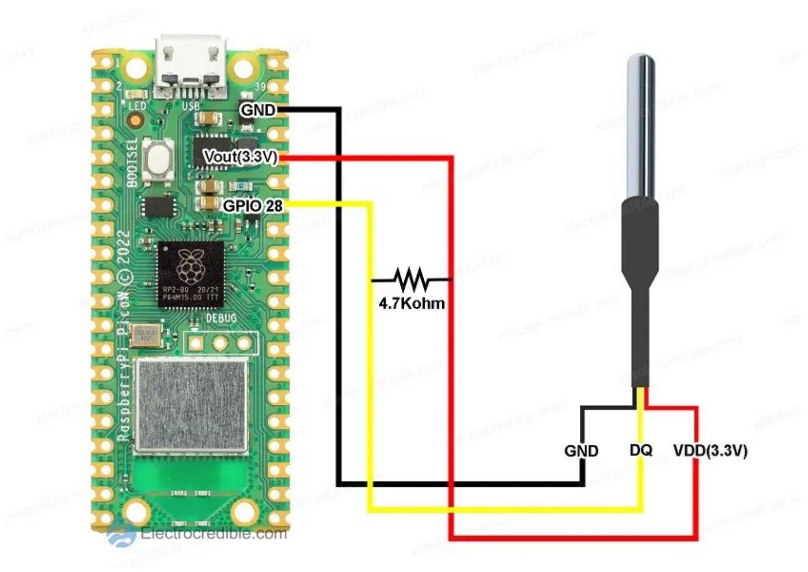
Connect a temperature sensor and a fan, and write a program that turns on the fan when the temperature gets too high.

Gather materials: You'll need a Raspberry Pi Pico board, a breadboard, a temperature sensor (such as a DS18B20), a transistor (such as a TIP120), a DC fan, a 1K ohm resistor, a diode, and some jumper wires.

1. Connect the temperature sensor to the breadboard: Connect the temperature sensor to the breadboard using jumper wires. Connect the red wire to a row on the breadboard, the black wire to another row, and the yellow or white wire to another row.



2. Connect the 1K ohm resistor: Connect one leg of the 1K ohm resistor to the same row on the breadboard as the yellow or white wire of the temperature sensor. Then, connect the other leg of the resistor to another row on the breadboard.



3. Connect the transistor: Place the transistor on the breadboard and connect the collector (middle leg) to a row on the breadboard using a jumper wire. Then, connect the emitter (left leg) to another row on the breadboard using another jumper wire. Finally, connect the base (right leg) to another row on the breadboard using a jumper wire.
4. Connect the diode: Connect the cathode (marked with a stripe) of the diode to the same row on the breadboard as the collector of the transistor. Then, connect the anode of the diode to another row on the breadboard.
5. Connect the fan: Connect the positive wire of the fan to the same row on the breadboard as the anode of the diode. Then, connect the negative wire of the fan to another row on the breadboard.
6. Connect the Raspberry Pi Pico: Connect the Raspberry Pi Pico to the breadboard using jumper wires. Connect one wire to pin 28 on the Pico and the other end to the row on the breadboard where the yellow or white wire of the temperature sensor and the 1K ohm resistor are connected. Connect another wire to pin 12 on the Pico and the other end to the row on the breadboard where the base of the transistor is connected.
7. Write the code: Open your preferred programming environment and write the following code:

Programming

```
import machine
import onewire
import ds18x20
import time

fan_pin = machine.Pin(12, machine.Pin.OUT)
fan_pwm = machine.PWM(fan_pin)
fan_pwm.freq(100)

ds_pin = machine.Pin(28)
ds_sensor = ds18x20.DS18X20(onewire.OneWire(ds_pin))

while True:
    ds_sensor.convert_temp()
    time.sleep_ms(750)
    temp = ds_sensor.read_temp(rom)
    if temp > 25:
        fan_pwm.duty(512)
    else:
        fan_pwm.duty(0)
    time.sleep(1)
```

This code reads the value of the temperature sensor using the `ds18x20` library, and turns on the fan using a PWM signal on pin 12 when the temperature gets above 25 degrees Celsius. The `time.sleep(1)` line adds a 1 second delay between each measurement to avoid excessive processing.

9. Upload and run the code: Save the code to a file on your computer, then upload it to the Raspberry Pi Pico using the USB cable. After uploading, the fan should turn on and off depending on the temperature detected by the temperature sensor.

Experiment 14

To be continued

Appendix 1 Pico Pins

This illustrates the many functions of the Pins

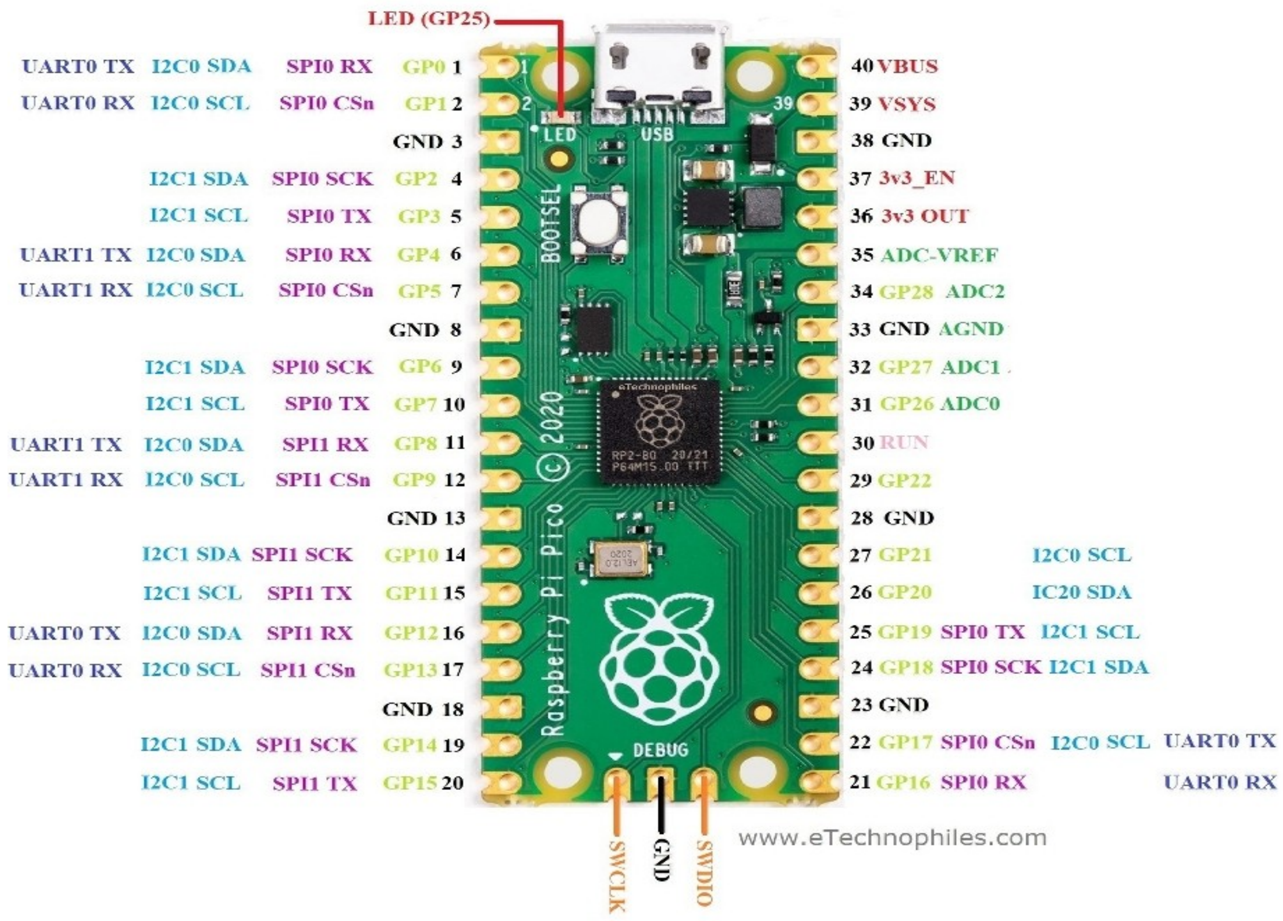


Figure 7: Many of the pins on the Pi Pico board can provide different features and capabilities.

Glossary

Python Notes

Comments

Comments are essential they can add to the clarity of the code and help in understanding the processes going on.

In a Python script you can place a comment anywhere as long as it is preceded by a hash #

```
# This line will be ignored by Python.
```

Variables

Variables are containers for storing data values.

A variable is created the moment you first assign a value to it.

Strings

Simply a string of characters

Strings can be enclosed by single ‘ or double “ quotes.

```
name=”GiaKonda”

phone=’01633 482648’
```

Python Numbers

There are three numeric types in Python:

Float, or "floating point number" is a number, positive or negative, containing decimals.

Float can also be scientific numbers with an "e" to indicate the power of 10. (12e2 is 1200)

Integer or whole number, a whole number, positive or negative, without decimals.

Complex numbers are written with a "j" as the imaginary part: 4+2j

You can convert from one type to another with the int(), float(), and complex() methods:

```
#convert from int to float:

a = float(x)

#convert from float to int:

b = int(y)

#convert from int to complex:

c = complex(x)
```

Printing

Invariably the first thing to be done in Python classes is to print Hello World. This is done with the print statement.

```
print(‘Hello World’) # This displays Hello World on the output device
```

Python Indentation

Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.

Python will give you an error if you skip the indentation:

The number of spaces is up to you as a programmer, the most common use is four, but it has to be at least one.

You have to use the same number of spaces in the same block of code, otherwise Python will give you an error:

It is generally good practice for you not to mix tabs and spaces when coding in Python. Doing this can possibly cause a `TabError`, and your program will crash. Be consistent when you code - choose either to indent using tabs or spaces and follow your chosen convention throughout your program.

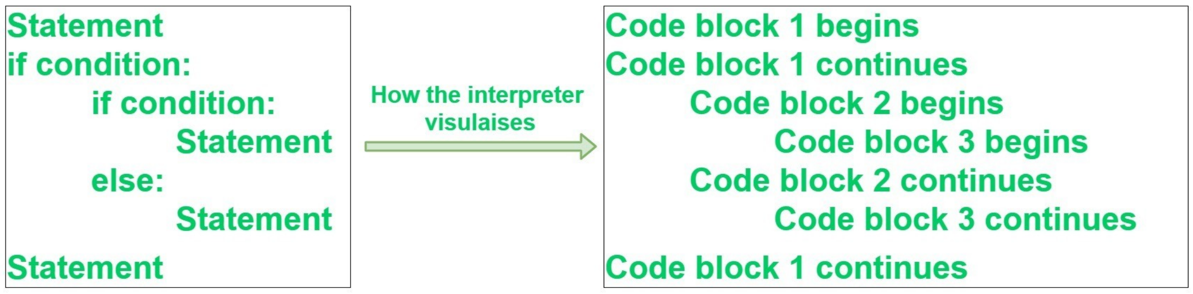


Figure 8: Python indentation

Example Checking a password

```
pwd = 'apple'
if pwd == 'apple':
    print('Logging on ...')
else:
    print('Incorrect password.')

print('All done!')
```

Conditions

a == b # a is equal to b

a != b # a is not equal to

a > b # is greater than b

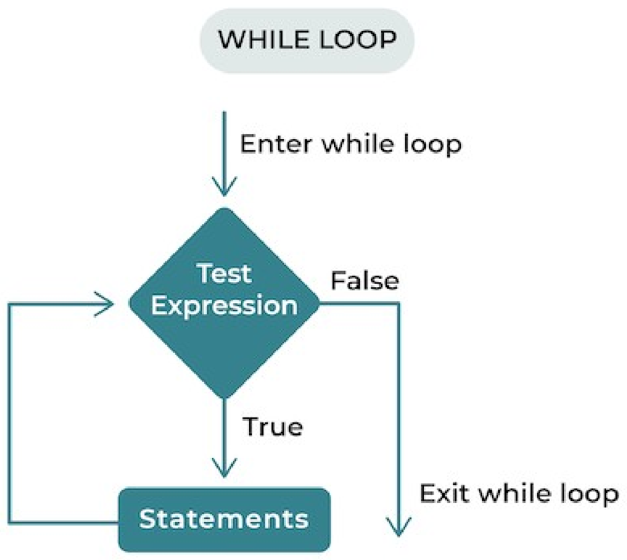
a <= b # a is less than or equal to b

a >= b # a is greater than or equal to b

a = True # a is True (The boolean value True)

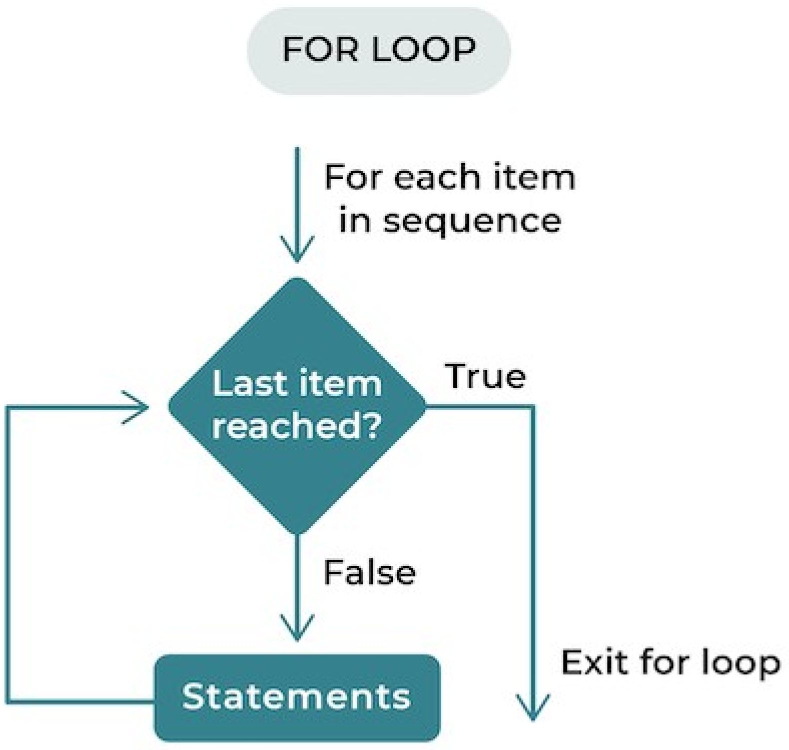
Loops

while loop



```
i = 0
while I<5:
    print(i)
    I += 1
```

for loop



```
# for loop to print a set of characters
letters = [‘a’,’b’,’c’,’d’,’e’]
for char in letters
    print(char)
```

Functions

Define functions as code elements to do a particular task When you call that function, you execute the code it contains. Functions also let you input parameters to run the same code with different values.

There are different types of functions in Python:

1. Built-in functions provided with Python.
2. User-defined functions that programmers (that's you!) create.

You define it using the `def` keyword, the name of the function, parentheses, and a colon `:`.

If the function requires one or more parameters, they go inside the parentheses separated by commas.

```
def say_hi(name):

   print('Hi' +name)

say_hi('GiaKonda')
```

Let's say you want to add two numbers together. Below is a code snippet for an `add()` method that takes in two numbers as the parameters and returns the sum.

```
def add(x,y)
    return x+y
```

Built in functions

len(string) returns the length of a string

format(value,format) sets the format of text or numbers

isinstance(object,type)

str(object)

int(value) returns the integer value of a number

range(min, max, step)

open(filename,mode) opens a file as read only, read write etc.

type(object) returns the type of a variable

exec(code) execute a system command

float(value) returns a number as floating point

min(value) returns the minimum value

max(value) returns the maximum value

round(value) returns a rounded number

Sensors

The DS18B20 temperature sensor

DS18B20 Features and Specifications

DS18B20 has the following electrical characteristics:

- Supply Voltage(VDD): 3V to 5.5V.
- Standby current: 750nA (typical).
- Active current: 1mA (typical).
- Voltage range on any pin relative to ground: -0.5V to +6.0V

Here are the key features of DS18B20

- **Wide temperature range**: The sensor can measure temperature over a wide range, from -55°C to +125°C.
- **High accuracy**: The DS18B20 can measure temperature with an accuracy of ±0.5°C (between -10°C and +85°C).
- **1-wire interface**: With a single wire interface, the sensor makes wiring easier and enables the connection of numerous sensors to a single microcontroller or data logger.
- **Multiple resolutions**: The DS18B20 has several resolution settings, measuring temperatures with resolutions from 9 bits to 12 bits. The measurement accuracy increases with resolution, while conversion times increase.
- **Low power consumption**: The sensor uses only 1 µA when in standby mode and 750 µA when converting temperatures.