# **Raspberry Pi Pico training for using microPython for building favorite projects**

**The general plan:**

**Saturday September 11 2021 10AM – 11:30 AM**

10AM Meet & Greet, Project Show & Tell

10:30 AM Program: **HANDS ON !** Introduction to the Raspberry Pi Pico controller hardware.

using the BOOTSEL pushbutton, loading the Pico with micropython UF2.

How to turn an output off or on. High Low True False 1 0

**Run Blink25**

**Saturday October 2 2021 10AM – 11:30 AM**  
10AM Meet & Greet, Project Show & Tell  
10:30 AM Program: Raspberry Pi Pico: hardware inputs & coding.  
Dry contacts (pushbuttons & switches),  
what is pull up and pull down ?   
simple hardware outputs & coding them. LEDs, buzzers,

**Saturday November 6 2021 10AM – 11:30 AM**

10AM Meet & Greet, Project Show & Tell  
~ 10:30 AM Hands On Program: Raspberry Pi Pico:  
micropython cmds to read digital and analog inputs, potentiometers,  
read the Pico onboard temperature sensor

intro to PWM, observe PWM with an oscilloscope, LED dimmer

What is digital ? binary ? Base 2 words, bytes, bits,

**Saturday December 4 2021 10AM – 11:30 AM**  
10AM Meet & Greet, Project Show & Tell  
10:30 AM HANDS ON Program: Raspberry Pi Pico: robot sensor hardware outputs & coding them. ping sensors, PWM for driving servos and DC motor H bridge control. RUN/RESET PB input, maybe more ??

Def FUNCTIONs, RANDOM

**By December , the students will have enough hands training to build their own Raspberry Pi Pico BeetleBot**

## Tutorials and online references:

* <https://www.raspberrypi.org/documentation/rp2040/getting-started/>
* <https://www.raspberrypi.org/documentation/microcontrollers/#getting-started-with-micropython>
* <https://hackspace.raspberrypi.org/books/micropython-pico> (free download of the book)
* [**https://dronebotworkshop.com/pi-pico/**](https://dronebotworkshop.com/pi-pico/)
* <https://www.tomshardware.com/how-to/program-raspberry-pi-pico-with-arduino-ide> this link has the help to install the Arduino version needed for Pico programming on a Raspberry Pi
* <https://www.tomshardware.com/news/raspberry-pi-pico-tutorials-pinout-everything-you-need-to-know>
* To clear out Arduino (or micropython) load on the RPi Pico, use flash\_nuke.uf2 , found at :

<https://learn.adafruit.com/raspberry-pi-pico-led-arcade-button-midi-controller-fighter/installing-circuitpython>

* [Servos for Robots](https://www.rs-online.com/designspark/servos-for-robots?s=03) <https://www.rs-online.com/designspark/servos-for-robots?s=03>
* <https://github.com/MechaGary/RaspberryPiPicoFall2021_2022>

<https://youtu.be/7ty-Tb7d824> The BeetleBot from show & tell in action

<https://www.learnpython.org/> <https://www.learnpython.org/en/Loops>

Learn the Basics

[Hello, World!](https://www.learnpython.org/en/Hello%2C_World%21)

[Variables and Types](https://www.learnpython.org/en/Variables_and_Types)

[Lists](https://www.learnpython.org/en/Lists)

[Basic Operators](https://www.learnpython.org/en/Basic_Operators)

[String Formatting](https://www.learnpython.org/en/String_Formatting)

[Basic String Operations](https://www.learnpython.org/en/Basic_String_Operations)

[Conditions](https://www.learnpython.org/en/Conditions)

[Loops](https://www.learnpython.org/en/Loops)

[Functions](https://www.learnpython.org/en/Functions)

[Classes and Objects](https://www.learnpython.org/en/Classes_and_Objects)

[Dictionaries](https://www.learnpython.org/en/Dictionaries)

[Modules and Packages](https://www.learnpython.org/en/Modules_and_Packages)

Data Science Tutorials

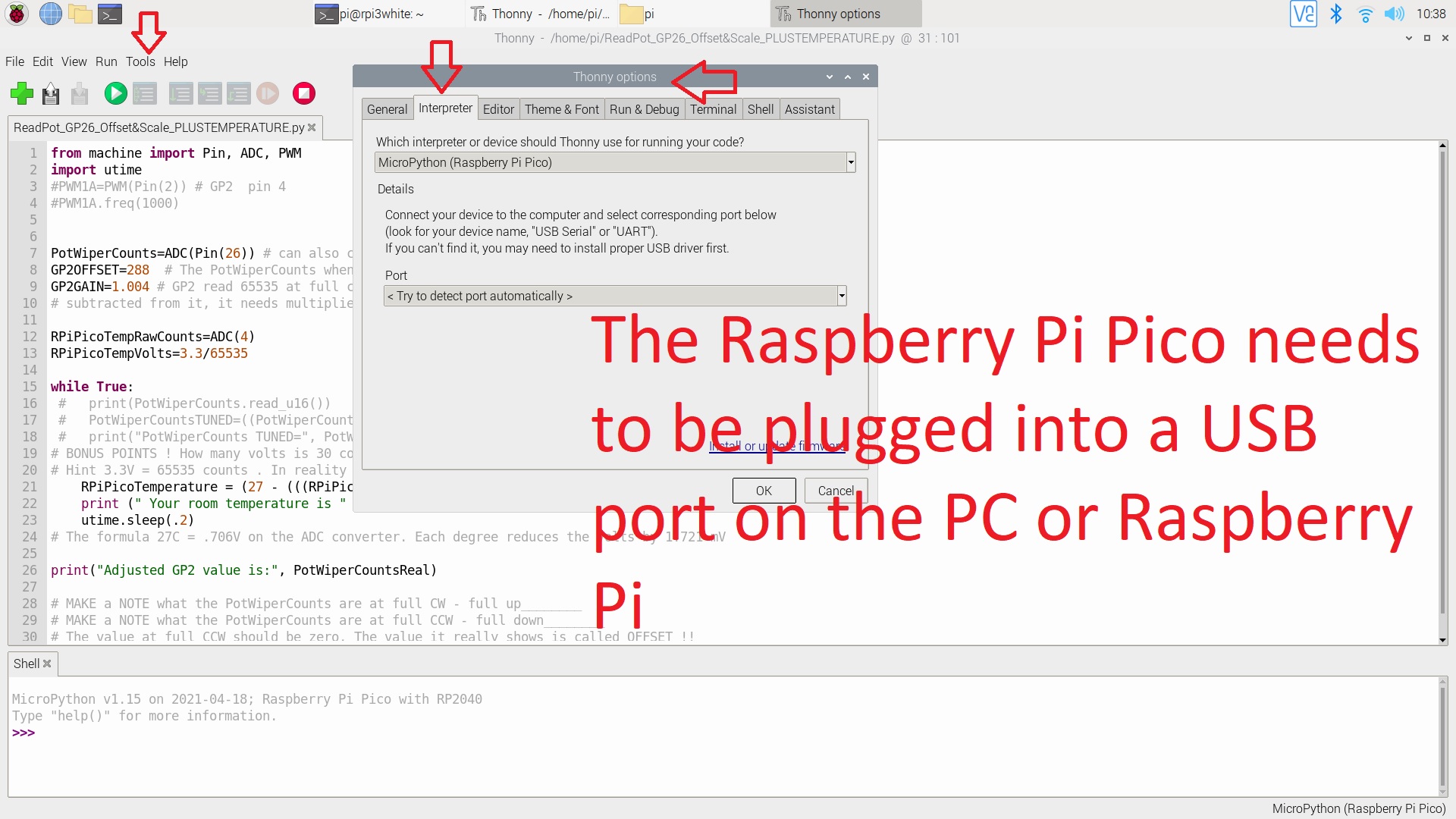
[Numpy Arrays](https://www.learnpython.org/en/Numpy_Arrays)

[Pandas Basics](https://www.learnpython.org/en/Pandas_Basics)

**September 2021 class Overview: About the Raspberry Pi Pico.**

**Getting Started With Raspberry Pi Pico – free pdf Download of the book (or borrow from library?)**[**https://hackspace.raspberrypi.org/books/micropython-pico/pdf/download**](https://hackspace.raspberrypi.org/books/micropython-pico/pdf/download)

1. What is a Raspberry Pi Pico…..????
2. What is a solderless breadboard ?
3. What is a buss ?
4. Strip jumper wires for breadboard
5. Connect the USB cable to the PiTop and a Pico. What happens ?
6. Get UF2 files for microPython. <https://www.raspberrypi.org/documentation/microcontrollers/micropython.html>
7. Load microPython to Pico.
8. Open Thonny, write program Blink25



https://github.com/MechaGary/RaspberryPiPicoFall2021\_2022.git

## The class demo is using Thonny microPython, programmed from a Raspberry Pi 3B to the Pico.

[Getting Started with RP2040 – Raspberry Pi](https://www.raspberrypi.org/documentation/rp2040/getting-started/#getting-started-with-micropython) - https://www.raspberrypi.org/documentation/microcontrollers/#getting-started-with-micropython

download rp2-pico-20210418-v1.15.uf2, drag and drop to the Pico. Pico rebooted into FS mode (took a minute or 2). No LEDS lit (there aren’t any preprogrammed)

Blink or Use GP25 LED as a power on LED or RUNNING LED !

Hold down the ‘BOOTSEL’ button on the top of your Pico; then, while still holding it down, connect the other end of the micro USB cable to one of the USB ports on your Raspberry Pi or other computer. Count to three, then let go of the ‘BOOTSEL’ button. After a few more seconds you should see your Pico appear as a removable drive – as though you’d connected a USB flash drive or external hard drive. On your Raspberry Pi you’ll see a pop-up asking if you’d like to open the drive in the File Manager. In the File Manager window, you’ll see two files on your Pico (Figure 1-16): INDEX.HTM and INFO\_UF2.TXT. The second file holds information about your Pico, such as the version of the bootloader it’s currently running. The first file, INDEX.HTM, is the one you want: point the mouse cursor at it and double-click it to open it in your browser. When the browser opens, you’ll see a welcome page telling you all about your Pico. Read the information on the page, then click on the MicroPython tab to load the MicroPython-specific section of the page. Click on the ‘Download UF2 file’ button (Figure 1-17, overleaf) to download the MicroPython firmware – a small file which contains MicroPython for your Pico.

**FAQs/NOTES:**

* **ALWAYS DISCONNECT THE PICO FROM POWER WHILE WIRING**
* **Raspberry Pi . org instructions and files are used for the classes. Adafruit CircuitPython is close, uses MU , but some differences exist. Refer to:** **https://learn.adafruit.com/getting-started-with-raspberry-pi-pico-circuitpython?view=all&gclid=EAIaIQobChMI1\_CMsf\_q8gIVcQaICR2TVgnZEAAYASAAEgIrCvD\_BwE**
* **Use BOOTSEL pushbutton on Pico to have it behave like a USB stick, then drag & drop UF2 file.**
* Thonny Python IDE Tools\Options\Interpreter MUST be set to Pico. Check everytime when opening Thonny.
* **Libraries are very important for anything to work in Python**
* # is a comment in python
* Ran example to blink GPIO 25 led using Thonny. However, it doesn’t automatically run when USB is plugged in**. To have the file run on powerup, save it to the Pico , naming it main.py !**
* NOTE on Thonny Python, if you make a typing mistake, the code may compile without error. While True: the equivalent of void Loop () in Arduino. Can create logic to bail out of infinite loop- break <https://www.learnpython.org/en/Loops>
* Python uses indentation to define code blocks, instead of brackets. The standard Python indentation is 4 spaces, although tabs and any other space size will work, as long as it is consistent. Notice that code blocks do not need any termination.



Blink25.py copy code into Thonny and load to the Pico, Click RUN !

PLEASE NOTE THE INDENTATIONS used for making loops.

from machine import Pin

import utime

led = Pin(25, Pin.OUT)

led.low()

while True:

led.toggle()

utime.sleep(.1)

When do you need to repower with the BOOTSEL button pressed ?

**CLASS 2**

10:30 AM Program: Raspberry Pi Pico: hardware inputs & coding.  
Dry contacts (pushbuttons & switches),  
what is pull up and pull down ?   
simple hardware outputs & coding them. LEDs, buzzers,

* **REMEMBER if your code turns something ON, the code probably needs to turn it off somewhere/sometime.**
* **In Thonny, the code may throw errors if the hardware connected to the Pico is missing.**
* **The Pico code may not function correctly if hardware is wrong or missing.**
* How do you think the LEDs will turn on in **blink25\_2\_3\_4.py** – before actually running the code ?
* Are these the same ? What is the meaning ? HIGH TRUE 1 ON , LOW FALSE 0 OFF

**blink25\_2\_3\_4.py wire in the “bug” leds , type in the code and run.**

**Hint you can open blink25.py , save as blink25\_2\_3\_4 and type in the remaining code**

from machine import Pin

import utime

led\_GP25= Pin(25, Pin.OUT) #remember this pinout is not accessible, the LED is hardwired on the Pico.

led\_GP2 = Pin(2, Pin.OUT) # GP2 is physical pin 4

led\_GP3 = Pin(3, Pin.OUT) # GP3 is physical pin 5

led\_GP4 = Pin(4, Pin.OUT) # GP3 is physical pin 6

led\_GP25.low() # this line ensures the LED is initilized to the OFF state.

**#Initialzing outputs and values are very important to start a machine in a safe predictable way**.

led\_GP2.low()

led\_GP3.low()

led\_GP4.low()

while True:

led\_GP25.toggle()

utime.sleep(.1)

led\_GP2.toggle()

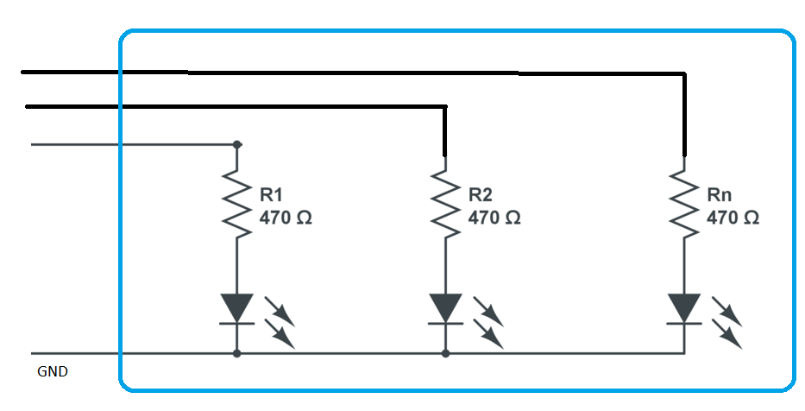
utime.sleep(.1)

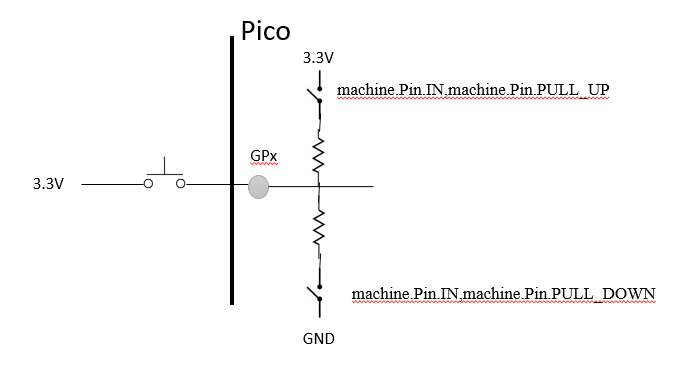
led\_GP3.toggle()

utime.sleep(.1)

led\_GP4.toggle()

utime.sleep(.1)

The “bug”



**LEDonFromPushButton.py**

from machine import Pin

import utime

pushbutton1= Pin(16, machine.Pin.IN,machine.Pin.PULL\_DOWN)#GP16 is pin 21

# the open contacts of switch and pushbutton dont provide voltage to the Pico input

# it jumps around called floating.

# the pull down feature turns on a resistor between the input and GND so the input reads 0 (low) when button not pushed

# the pull up feature turns on a resistor between the input and 3V so the input reads 1 (HIGH) when button not pushed

pushbutton2= Pin(17, machine.Pin.IN,machine.Pin.PULL\_DOWN)#GP17 is pin22

pushbutton1LED= Pin(25, machine.Pin.OUT)#GP25 is the Pico mounted LED

pushbutton2LED= Pin(2, machine.Pin.OUT)#GP2 is Pico pin4

while True:

if pushbutton1.value() ==1:

pushbutton1LED.value(1)

if pushbutton1.value() ==0:

pushbutton1LED.value(0)

if pushbutton2.value() ==1:

pushbutton2LED.value(1)

if pushbutton2.value() ==0:

pushbutton2LED.value(0)

utime.sleep(.15) # this is to help debounce the pushbutton

**LEDonFromPushButtonAndBuzzer.py**

from machine import Pin

import utime

pushbutton1= Pin(16, machine.Pin.IN,machine.Pin.PULL\_DOWN)#GP16 is pin 21

# the open contacts of switch and pushbutton dont provide voltage to the Pico input

# it jumps around called floating.

# the pull down feature turns on a resistor between the input and GND so the input reads 0 (low) when not pushed

# the pull up feature turns on a resistor between the input and 3V so the input reads 1 (HIGH) when not pushed

pushbutton2= Pin(17, machine.Pin.IN,machine.Pin.PULL\_DOWN)#GP17 is pin22

pushbutton1LED= Pin(25, machine.Pin.OUT)#GP25 is the Pico mounted LED

pushbutton2LED= Pin(2, machine.Pin.OUT)#GP2 is Pico pin 4

pushbutton2Buzzer= Pin(14, machine.Pin.OUT)#GP14 is Pico pin19

while True:

if pushbutton1.value() ==1:

pushbutton1LED.value(1)

if pushbutton1.value() ==0:

pushbutton1LED.value(0)

# this is a brute force way to code both states of the pushbutton. Another way is using the if....else statement

if pushbutton2.value() ==1:

pushbutton2LED.value(1)

if pushbutton2.value() ==0:

pushbutton2LED.value(0)

# added in buzzer output

if pushbutton2.value() ==1:

pushbutton2Buzzer.value(1)

if pushbutton2.value() ==0:

pushbutton2Buzzer.value(0)

utime.sleep(.1) # this is to help debounce the pushbutton

**LEDonFromPushButtonAndBuzzerIF\_ELSEpy.py**

from machine import Pin

import utime

pushbutton1= Pin(16, machine.Pin.IN,machine.Pin.PULL\_DOWN)#GP16 is pin 21

# the open contacts of switch and pushbutton dont provide voltage to the Pico input

# it jumps around called floating.

# the pull down feature turns on a resistor between the input and GND so the input reads 0 (low) when not pushed

# the pull up feature turns on a resistor between the input and 3V so the input reads 1 (HIGH) when not pushed

pushbutton2= Pin(17, machine.Pin.IN,machine.Pin.PULL\_DOWN)#GP17 is pin22

pushbutton1LED= Pin(25, machine.Pin.OUT)#GP25 is the Pico mounted LED

pushbutton2LED= Pin(2, machine.Pin.OUT)#GP2 is Pico pin 4

pushbutton2Buzzer= Pin(14, machine.Pin.OUT)#GP14 is Pico pin19

while True:

if pushbutton1.value() ==1:

pushbutton1LED.value(1)

else:

pushbutton1LED.value(0)

# if....else statement is shorter, Make Note where the colons are in the python statements

if pushbutton2.value() ==1:

pushbutton2LED.value(1)

pushbutton2Buzzer.value(1)

else:

pushbutton2LED.value(0)

pushbutton2Buzzer.value(0)

utime.sleep(.1) # this is to help debounce the pushbutton

**Saturday November 6 2021 10AM – 11:30 AM CLASS 3**

10AM Meet & Greet, Project Show & Tell  
~ 10:30 AM Hands On Program: Raspberry Pi Pico:  
micropython cmds to read digital and analog inputs, potentiometers,  
read the Pico onboard temperature sensor

intro to PWM, observe PWM with an oscilloscope, LED dimmer

<https://projects.raspberrypi.org/en/projects/getting-started-with-the-pico/8#:~:text=Your%20Raspberry%20Pi%20Pico%20has,your%20circuit%20with%20a%20potentiometer>.

**ANALOG INPUTS: The Raspberry Pi Pico has three analog inputs, and they all have 12-bits of resolution (4095 counts, but Python rescales the value to 16bits 65535). The three available inputs are:**

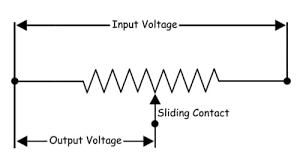
* **GPIO 26 – ADC0 (pin 31)**
* **GPIO 27 – ADC1 (pin 32)**
* **GPIO 28 – ADC2 (pin 34)**

ADC4 is used for the internal temperature sensor.

* 3V input to GPIO = 65535 counts. Divide the counts by 65535 to display the value in volts, or convert to other values.

**All ABOUT POTS , rheostats, potentiometers**

* Potentiometers (pots) are used in joystick game controllers for reading joystick position, and inexpensive servos for position feedback.
* **A potentiometer (pronounced “poe-ten-shee-AH-meh-ter”) is a variable resistor.... The resistance between the two outer terminals is a fixed (constant) resistance, and it is the maximum resistance of the pot. The voltage at the third terminal (wiper) can be adjusted to give any fraction of the voltage across the ends of the resistor.**
* **ALWAYS CHECK THE POT TERMINAL NUMBERS TO MAKE SURE WIRES GO TO CORRECT Pot TERMINALs. INCORRECT WIRING CAN SHORT OUT THE POWER SUPPLY**
* Wire one end of a pot (potentiometer) to GND, the other end to 3V, wire the pot wiper to GPIO13 (physical pin 17). Resistors can be used in the legs of the pot to intentionally add offset(minimum value) or to limit the range of the pot. Anymore this is usually done in software.



Wire V3.3 to the (usually) right hand side so a CW turn makes the voltage value larger

Wire GND to the (usually) left side so a CCW turn makes the voltage value smaller

Math Genius Question: If you needed to make a multiplier value, called ratio, to scale any signal based on where the voltage of the GPIO value, what would the equation look like ? Hint The maximum value of ratio is 1 (representing 100%).

**ReadPot\_GP26\_A0.py** reading analog inputs are simple, type in the program and test for yourself.

Wire the pot terminal 1 to COM, terminal 2 to GP26(physical pin31), terminal 3 to +3V

from machine import Pin, ADC

import utime

PotWiperCounts=ADC(Pin(26))

while True:

print(PotWiperCounts.read\_u16())

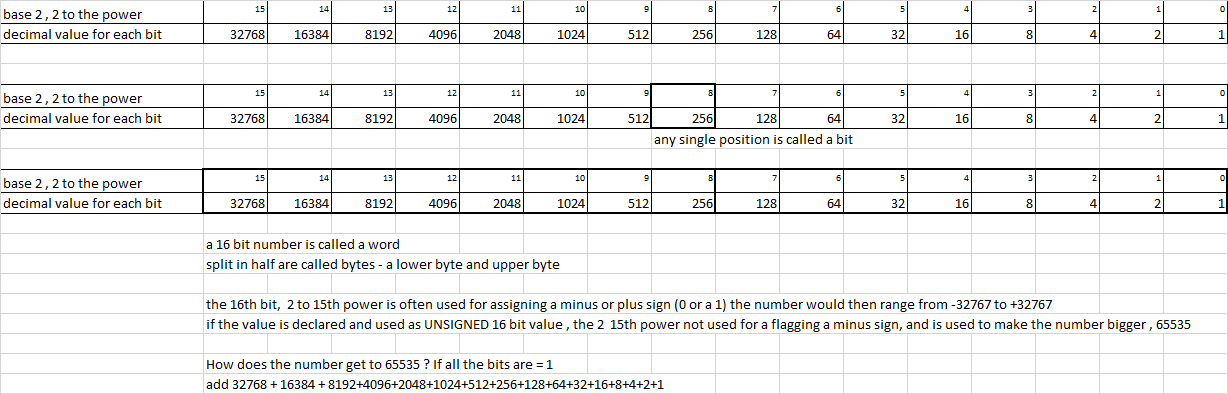
utime.sleep(.5)

------------------- DID YOU NOTICE the read\_u16()? What does that mean ? ---------------------

What does it mean to have a u16 number ?

What value is a 12 bit number ?

What value can you read with 8 bit resolution ?



**ReadPot\_GP26\_Offset&Scale.py** type in the program and test for yourself.

from machine import Pin, ADC, PWM

import utime

#PWM1A=PWM(Pin(2)) # GP2 pin 4

#PWM1A.freq(1000)

PotWiperCounts=ADC(Pin(26)) # can also call machine.ADC(3)

GP2OFFSET=288 # The PotWiperCounts when the pot is full CCW

GP2GAIN=1.004 # GP2 read 65535 at full cw, however once 288 counts are

# subtracted from it, it needs multiplied by the ratio to make it back to 65535 full scale

while True:

print(PotWiperCounts.read\_u16())

PotWiperCountsTUNED=((PotWiperCounts.read\_u16())-GP2OFFSET)\*GP2GAIN

print("PotWiperCounts TUNED=", PotWiperCountsTUNED)

# BONUS POINTS ! How many volts is 30 counts of offset ?

# Hint 3.3V = 65535 counts . In reality the resolution is still 4096 12 bits

utime.sleep(1)

# MAKE a NOTE what the PotWiperCounts are at full CW - full up\_\_\_\_\_\_\_\_

# MAKE a NOTE what the PotWiperCounts are at full CCW - full down\_\_\_\_\_\_\_\_

# The value at full CCW should be zero. The value it really shows is called OFFSET !!

# The next program you will correct the OFFSET and then also the gain error - The value at full CW !

**ReadPot\_GP26\_Offset&Scale\_PLUSTEMPERATURE.py** type in the program and test for yourself.

from machine import Pin, ADC, PWM

import utime

PotWiperCounts=ADC(Pin(26)) # can also call machine.ADC(3)

GP2OFFSET=288 # The PotWiperCounts when the pot is full CCW

GP2GAIN=1.004 # GP2 read 65535 at full cw, however once 288 counts are

# subtracted from it, it needs multiplied by the ratio to make it back to 65535 full scale

RPiPicoTempRawCounts=ADC(4)

RPiPicoTempVolts=3.3/65535

while True:

RPiPicoTemperature = (27 - (((RPiPicoTempRawCounts.read\_u16()\*RPiPicoTempVolts)-.706)/.001721))

print (" Your room temperature is " , RPiPicoTemperature)

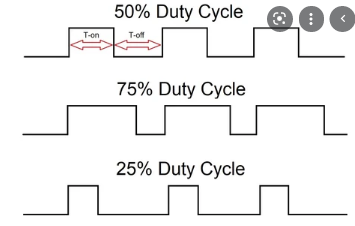
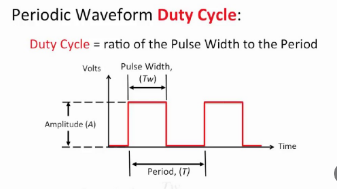
utime.sleep(.2)

# The formula 27C = .706V on the ADC converter. Each degree reduces the volts by 1.721 mV

print("Adjusted GP2 value is:", PotWiperCountsReal)

**What is PWM Pulse Width Modulation ?** It varying an output using digital switching. To the user it looks like an analog signal, sorta like persistence of vision projects.

Did you ever toggle a light switch really fast and see it looked like the light dimmed ? That our next experiment.



**ReadPot\_GP26\_DimLED\_GP2.py** type in the program that dims an LED based on a pot wiper position – a dimmer switch -- and test for yourself.

from machine import Pin, ADC, PWM

import utime

PWM1A=PWM(Pin(2)) # GP2 pin 4

PWM1A.freq(1000)

PotWiperCounts=ADC(Pin(26)) # can also call machine.ADC(3)

while True:

print(PotWiperCounts.read\_u16())

duty=PotWiperCounts.read\_u16()

PWM1A.duty\_u16(duty)

utime.sleep(.05)

Why is this program so short ?

<https://projects.raspberrypi.org/en/projects/getting-started-with-the-pico/7>

from machine import Pin, PWM

from time import sleep

pwm = PWM(Pin(15))

pwm.freq(1000)

while True:

    for duty in range(65025):

                 pwm.duty\_u16(duty)

                 sleep(0.0001)

        for duty in range(65025, 0, -1):

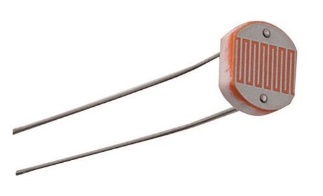
                 pwm.duty\_u16(duty)

                 sleep(0.0001)

65025 = 100%   2\*\*\*\*16 = 65535

Whoa ! much better resolution for angular and speed control (well, kinda)

<https://datasheets.raspberrypi.org/pico/raspberry-pi-pico-python-sdk.pdf>

**** A photoresistor or photocell is a light-controlled variable resistor.

The resistance of a photoresistor decreases with increasing light.

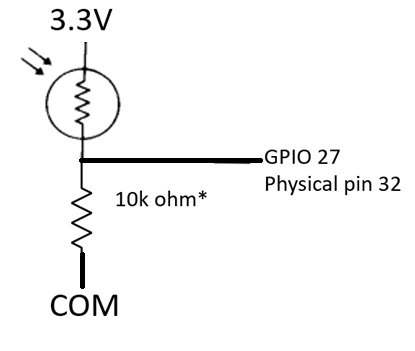
These photoresistors are built to have the same spectral response as human eyes.

The difference between component’s varies a lot, in bright light 100 ohms to 400 ohms.

In the dark, measures anywhere from 200k to 2 Mohm.

Because of the low resistance in bright light, a series resistor is needed to limit the photo resistor current. That series resistor also creates a nice voltage divider to wire into GPIO for brightness detection.

The photoresistor datasheet details about light levels and range the photoresistor can be applied to.



Wired this way, bright light has a higher voltage & numeric value of analog input, darkness takes the GPIO analog input value to zero.

Vcc is 3.3V

Vgpio/ Vcc = 10 k ohm / (Rphotoresistor + 10kohm)

\*this is where we become engineers. 10k was selected as a swag – educated guess of a starting point.

A simulator program could produce tables of different volts VS ohms VS light conditions. We could do the same thing with a spreadsheet to optimize and also experiment with different values.

**CASE 1 (300 ohm)**At extreme bright the photoresistor resistance is near zero. 300 ohms current limit resistor would limit the ckt to 10mA, easy on component & Pico power supply.

Vgpio would be 3.3V \*300 ohms /500 ohms, 2 V. Average light to dark 3.3V\*200,000 ohms/200,300 ohms 3.3V, the analog range is only using 30% of available counts.

**CASE 2 ( use 1kohm resistor)** - bright light Vgpio would be 3.3V \* 1000 ohms /1300 ohms, 2.5 V

Much improved range, average light to dark 3.3V\*1000 ohms/201,000 ohms = .016V

**CASE 3 (10 kohm resistor)** – bright light Vgpio would be 3.3V \* 10000 ohms /10300 ohms, 3.2V. The average light to dark 3.3V\*10,000 ohms/210,000 ohms = .157 V has more range and tolerance in case the photo resistor has a higher real value. If 2Mohm, then 3.3V\*10,000 ohms/2,010,000 ohms= .016V

Even tho python provides a resolution of 1/65535, the real analog resolution is still 1/4095 = .0002 V

So could even try 100kohm as an experiment.

For this example, we stick with the swag at 10kohm.

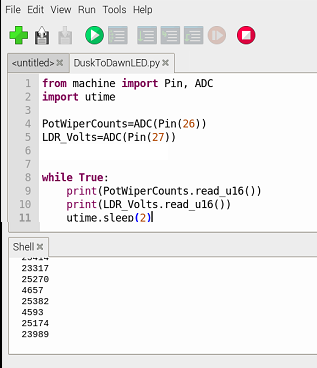
**Open program ReadPot\_GP26\_A0.py, and save as DuskToDawnLED.py**

**So lets first write a program to switch on the LED when the it gets dark outside**.

**Your house may have outside lights with this same circuit, or car may have this exact same control circuit for the headlights! Headlights would have some time delay in case you drive under a shade tree.**

**First add in the code line to read the LDR\_Volts at GPIO 27 (A1), run the code to see the value.**

**Put your finger over the LDR and observe the value change, it should get smaller.**

****

**After proving the analogs are reading, then complete the coding**

**DuskToDawnLED.py**

from machine import Pin, ADC

import utime

PotWiperCounts=ADC(Pin(26))

LDR\_Counts=ADC(Pin(27))

u16\_2\_volts = 3.3/(65535) #create a conversion factor

DuskToDawnLED= Pin(25, machine.Pin.OUT)#GP25 is the Pico mounted LED

DuskToDawnLED.value(0)

LightOnAdjust=1.1 ## in volts at GPIO

while True:

#print(PotWiperCounts.read\_u16())

#print(LDR\_Counts.read\_u16())

LDR\_Volts=(LDR\_Counts.read\_u16())\*u16\_2\_volts

print(LDR\_Volts)

if ((LDR\_Volts) < (LightOnAdjust)):

DuskToDawnLED.value(1)

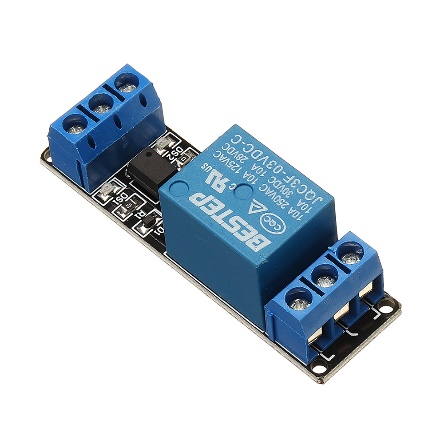
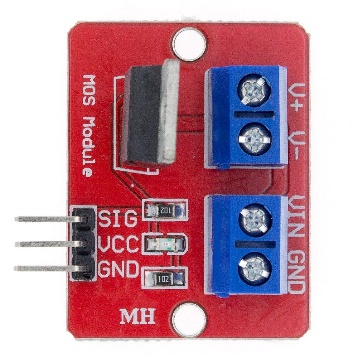
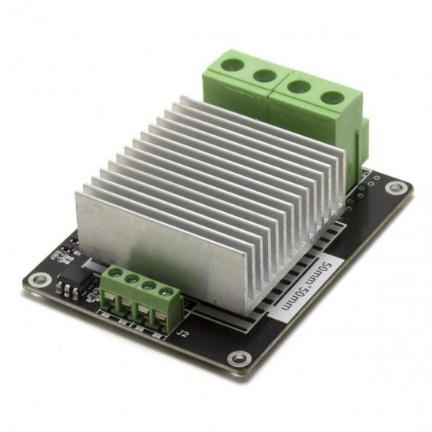
else:

DuskToDawnLED.value(0)

utime.sleep(2)

**hardware:**

If you’d like to turn on a bank of high power LEDS or a motor, you can use this GPIO output to drive a bipolar transistor base or a MOSFET gate to switch on a lot bigger loads, or use a relay module.

**NEXT DuskToDawnLED.py save as DuskToDawnLEDwDimmer.py**

**\*\*\* sorry, GP25 – Rpi Pico mounted LED cannot be PWM ☹ please use GP2 (pin 4 ) for an LED**

**write a program to dim or brighten the LED as the ambient light changes.**

**HINT: Your car may have this exact same control circuit for the dashboard lights !**

**Make the LED its brightest when it turns on, and dims as the LDR\_volts heads towards zero.**

**DuskToDawnLEDwDimmer.py**

**from machine import Pin, ADC, PWM**

**import utime**

**PotWiperCounts=ADC(Pin(26))**

**LDR\_Counts=ADC(Pin(27))**

**u16\_to\_volts = 3.3/(65535)**

**#volts\_2\_U16 = (65535)/3.3**

**# DuskToDawnLED= Pin(25, machine.Pin.OUT)#GP25 is the Pico mounted LED**

**# DuskToDawnLED.value(0)**

**DuskToDawnLED= PWM(Pin(2))# physical pin 4**

**DuskToDawnLED.freq(100)**

**LightOnAdjust=1.5 ## in volts at GPIO**

**# volts\_to\_U16 = (65535)/LightOnAdjust**

**volts\_to\_U16 = (65535)/LightOnAdjust**

**while True:**

**#print(PotWiperCounts.read\_u16())**

**#print(LDR\_Counts.read\_u16())**

**LDR\_Volts=(LDR\_Counts.read\_u16())\*u16\_to\_volts**

**print(LDR\_Volts)**

**if ((LDR\_Volts) < (LightOnAdjust)): ## floating point value of volts**

**DimmerDutyCounts=65535**

**if ((LDR\_Volts) < ((LightOnAdjust)\*.8)): ## floating point value of volts**

**DimmerDutyCounts=52400**

**if ((LDR\_Volts) < ((LightOnAdjust)\*.6)): ## floating point value of volts**

**DimmerDutyCounts=39000**

**if ((LDR\_Volts) < ((LightOnAdjust)\*.4)): ## floating point value of volts**

**DimmerDutyCounts=26200**

**if ((LDR\_Volts) < ((LightOnAdjust)\*.2)): ## floating point value of volts**

**DimmerDutyCounts=13000**

**if ((LDR\_Volts) > ((LightOnAdjust))): ## floating point value of volts**

**DimmerDutyCounts=0**

**DuskToDawnLED.duty\_u16(DimmerDutyCounts)**

**utime.sleep(2)**

**Sidetracked: Persistence of vision** traditionally refers to the optical illusion that occurs when visual perception of an object does not cease for some time after the rays of light proceeding from it have ceased to enter the eye.[[1]](https://en.wikipedia.org/wiki/Persistence_of_vision#cite_note-Cyclop%C3%A6dia-1) The illusion has also been described as "retinal persistence",[[2]](https://en.wikipedia.org/wiki/Persistence_of_vision" \l "cite_note-2) "persistence of impressions",[[3]](https://en.wikipedia.org/wiki/Persistence_of_vision#cite_note-3) simply "persistence" and other variations.

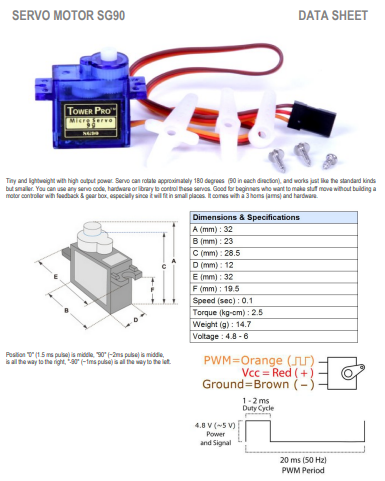
PLAYTIME = change DuskToDawnLED.freq(100) to DuskToDawnLED.freq(10), what do you see ?

PLAYTIME = change DuskToDawnLED.duty\_u16(32767) to change DuskToDawnLED.duty\_u16(xxxxx). How close to 0 can xxxxx before you cant see the LED glow ? what do you see ? Does it matter how bright the room is in seeing the LED glow ?

PLAYTIME change **LightOnAdjust= xx** to different values. What happens ?

Can you change the if statements to a u\_16 variable for dimmer duty cycle ?

**CLASS 4 hobby servos – used in model planes, cars, robots**

****

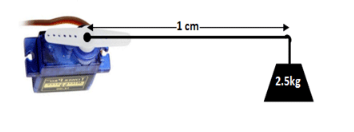
**NEVER NEVER NEVER NEVER**

Power a servo

(or an H bridge or a solenoid )

from the Pico 3.3V power supply.

Pico 3.3V limit is 300mA

****

About 5 lbs to stall out this servo.

A longer servo arm lowers weight lifting capacity.

**Angular servo 0-90-180 can also be called a -90-0-90 servo,**

set PWM freq = 50Hz

**center position =**  7.5% duty  1.5ms/20ms **DutyValue =** **4,876**

**minimum position =**= 5% duty   1ms/20ms **DutyValue = 3,251**

**maximumposition = 10% duty 2ms/20ms DutyValue = 6535**

**Servos can be purchased in many different sizes, torque ratings, voltages, speed**

**The same control scheme can be used to drive AC brushless controllers for drones, cars, planes and boats, but these applications generally don’t operate in reverse, only 0 degrees to 90 degrees**

**continuous servo** (doesn’t stop at any angle)

set PWM freq = 50Hz.

Full speed FWD = 10% duty

Zerospeed == 7.5% duty  1.5/20

Full speed reverse = 5% duty   1/20

**%Duty =   (SpeedRef  \*  .25) + 7.5 ( continuous servo)**

**DutyValue= (%Duty/100)\*65535**

**USE AN OSCILLOSCOPE TO VERIFY The PULSE WIDTHS**

**Also Refer to** [Servos for Robots](https://www.rs-online.com/designspark/servos-for-robots?s=03) <https://www.rs-online.com/designspark/servos-for-robots?s=03>

for additional information.

**ServoExampleClass4 – sweeps a servo arm from endpoints. Copy into Python and prove it works. HINT servo control lead wires to GP10, COM to COM, +5 to VBUS**

from machine import Pin, PWM

# Pin uses the GPIO value , not the actual pin number

import utime

AliveLED = Pin(25,machine.Pin.OUT)

AliveLED.value(0)

servo = PWM(Pin(10))# physical pin14

servo.freq(50) # 50 Hz is 20 msec period

# 100% duty of PWM is 65535 counts

# 2 ms of 20 ms is 10% for full spd FWD 6553

# ( 1.5 ms / 20 ms ) \* 65535 = 4875 for zero degrees

# 1 ms of 20 ms is full neg 3251 counts

#AliveLED.value(1)

count=0

while True:

ServoDutyCounts=3250

AliveLED.toggle()

while True:

AliveLED.toggle()

ServoDutyCounts=ServoDutyCounts+25

servo.duty\_u16(ServoDutyCounts)

utime.sleep(.03) # time to reach position

if (ServoDutyCounts>6553):

break

ServoDutyCounts=6553

while True:

AliveLED.toggle()

ServoDutyCounts=ServoDutyCounts-25

servo.duty\_u16(ServoDutyCounts)

utime.sleep(.03) # time to reach position, remember duty cycle is 20 msec, so if write references faster than than, servo cant keep up

if (ServoDutyCounts<3252):

break

# What happens of duty cycle in code exceeds servo spec

# There are 2 ways of make the servo sweep faster. What are they ?

**BONUS TASK:**

**Write Your OWN Program to read the value of a pot, and have the servo follow it**

[**https://www.raspberrypi.org/blog/how-to-add-a-reset-button-to-your-raspberry-pi-pico/**](https://www.raspberrypi.org/blog/how-to-add-a-reset-button-to-your-raspberry-pi-pico/)

**Defining FUNCTIONS in your code , AND using a Ping module**

A Python function is **a group of code instructions representing a desired behavior or calculation that is used repeatedly**. To run the code in a function, you must call the function. A function can be called from anywhere after the function is defined. Functions can return a value using a return statement.

For example, a function called Tease the Sibling.

1. Call siblings name
2. Did they answer ?

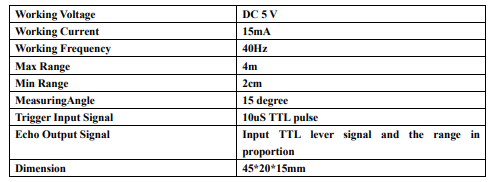
Yes – tell them to come see you

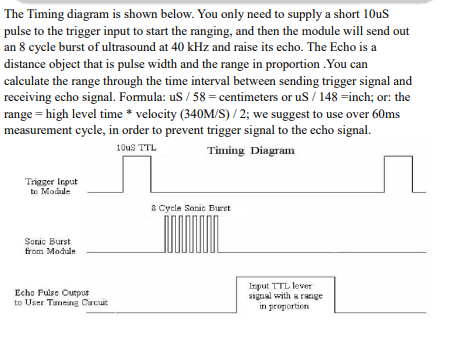
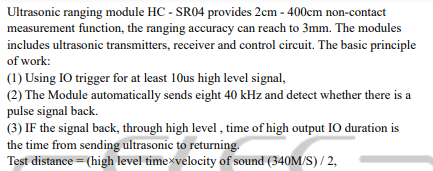
Else NO – call their name again louder

1. Is sibling in front of you ?
   1. Yes, take their toy

Else no – yell their even louder and tell them you have their ice cream

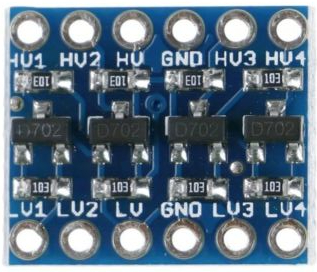
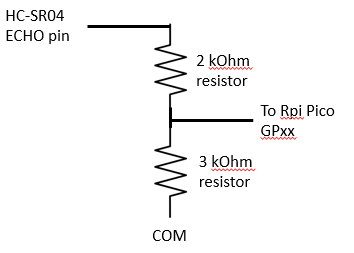
**The Ping Module hardware is an inexpensive way to measure distance on demand or on regular intervals. The code is a good example of a grouping code FUNCTION .**

****

****

**The HC-SR04 operates at 5V. The trigger is OK to output from the Pico at 3V, but the HC-SR04 ECHO needs scaled down from 5V to avoid overvolting the Pico INPUT.**

**use 2 resistors ( 5V \* (3.3k/5.5k)) ~=3V to reduce the ECHO volts or use a level shifter**

****

[**https://datasheets.raspberrypi.org/pico/pico-datasheet.pdf**](https://datasheets.raspberrypi.org/pico/pico-datasheet.pdf)