
Breadboards, Raspberry Pi-Pico, & Python

ECE 401 Senior Design I

Week #3

Please visit [Bison Academy](#) for corresponding lecture notes,
homework sets, and videos

Introduction

In Senior Design I, you can

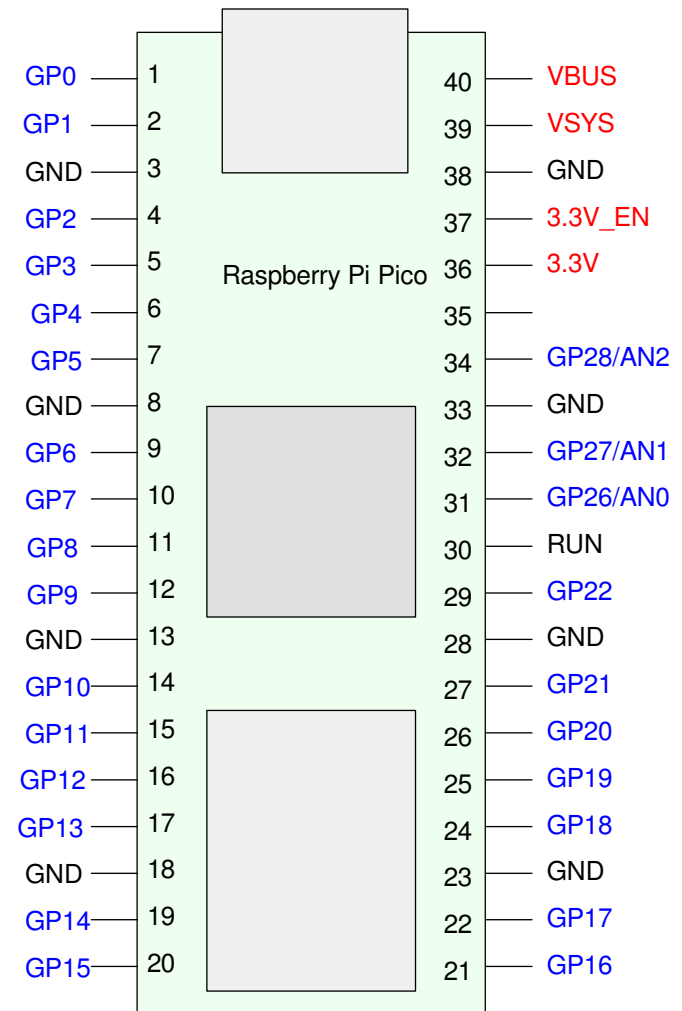
- Use a microcontroller, or
- Not use a microcontroller

Microcontrollers can simplify many designs

- They provide a great deal of flexibility
- They make changing your design as simple as downloading a new program

It's your choice

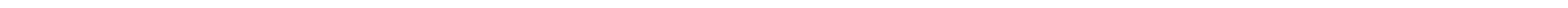
- If you *do* use a microcontroller, use a Raspberry Pi-Pico
- \$4 and we have them in stock



Topics:

In this lecture, we going to cover

- Breadboards
 - What they are
 - Do's and Don'ts of breadboard design
- Hardware:
 - How to wire up a Raspberry Pi-Pico
 - How to connect a push button (binary input)
 - How to connect an LED (binary output)
- Software:
 - Writing a program using Python
 - Setting up a Pi-Pico to execute that program on power-on

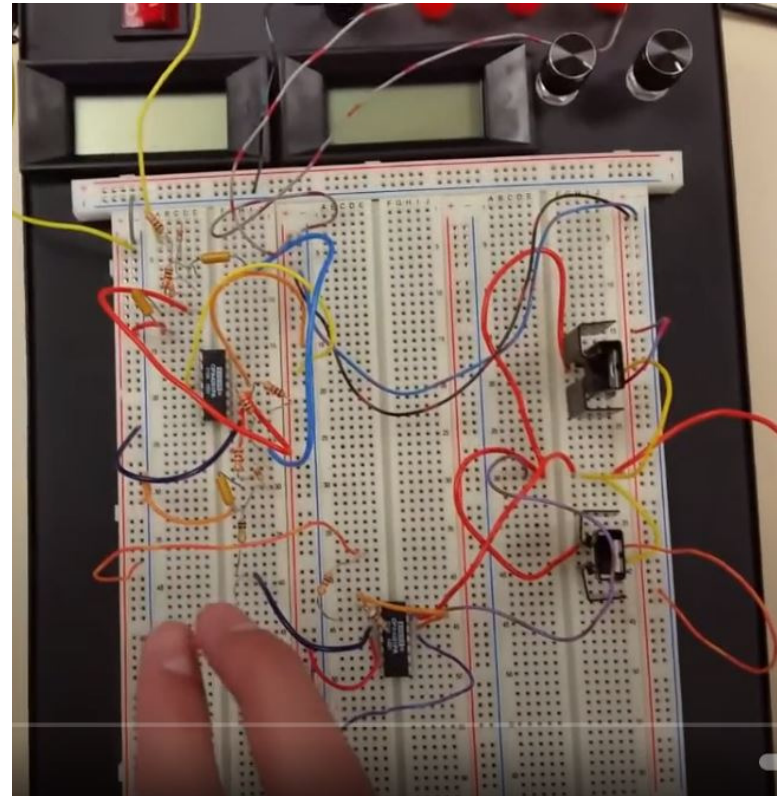


Breadboards:

Once you have your circuit

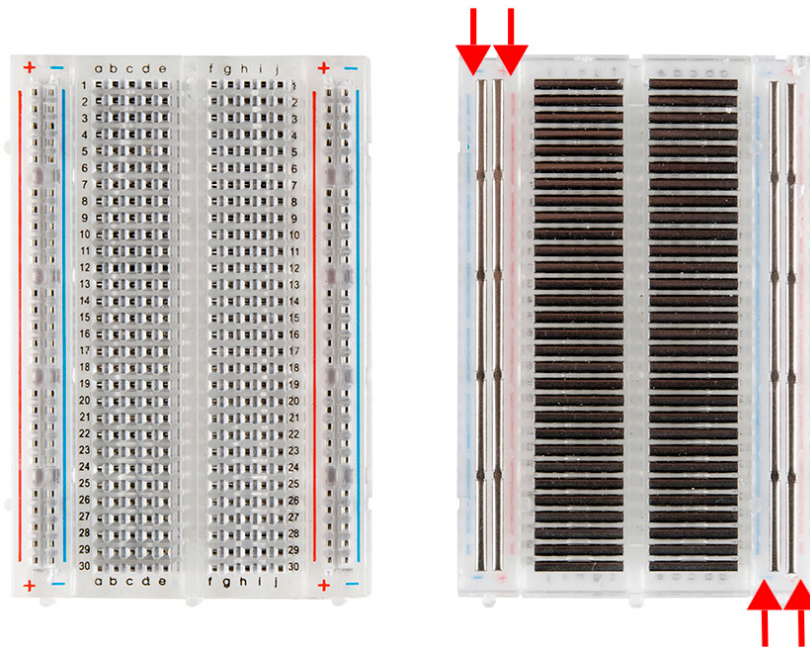
- Designed on paper, and
- Tested in simulation (CircuitLab)

you're ready to test your design in hardware. Breadboards are an easy way to build your circuit and test your design. They're also easy to modify and change: components can be easily added and removed from a breadboard.



Most of the breadboard used in ECE are 830 tie breadboards. These have

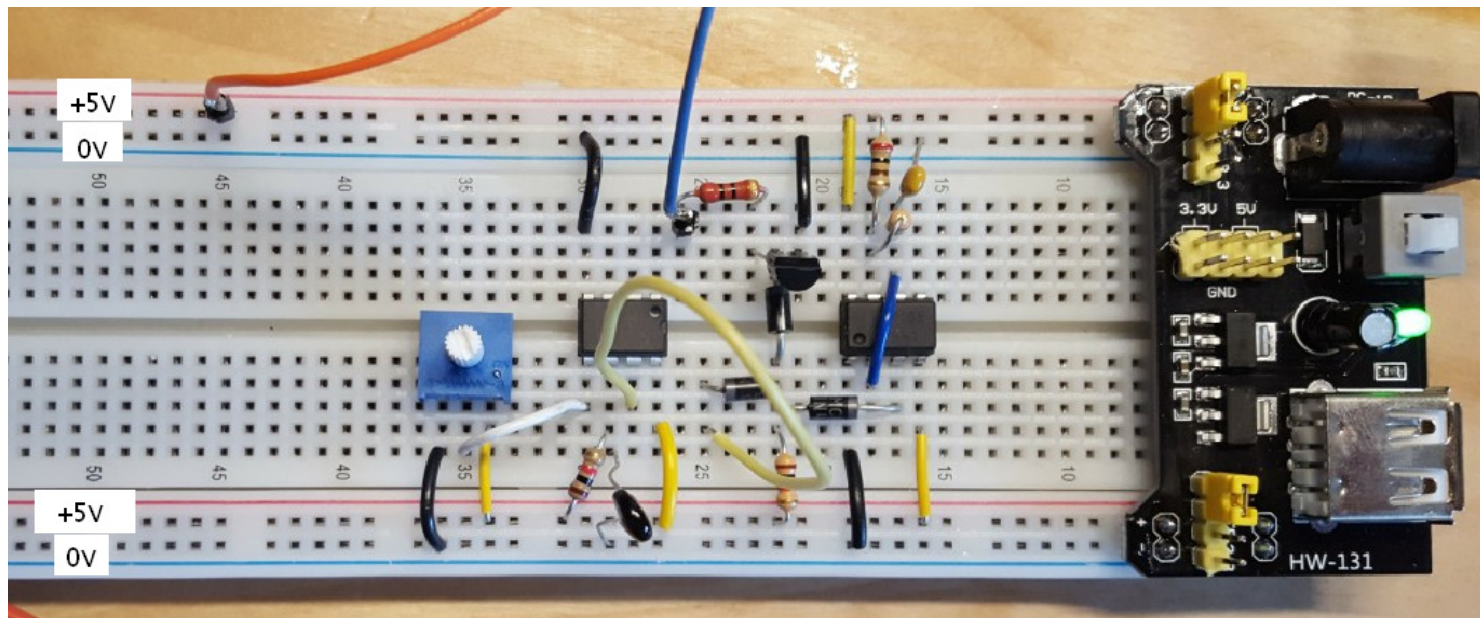
- Four edge connectors that are shorted along the length of the breadboard two left side, two right side. Usually used for power and ground
- Two sets of connectors in the middle one left and one right of the center bar.
- Across the middle is an insulator this separates the middle connectors by 300 mils: the width of a typical IC,



For example, the following breadboard circuit uses

- The red trace along the top and bottom as +5V
- The blue trace along the top and bottom as 0V
- Two IC's go across the middle divider.

The four pins above and below each IC then allow you to connect to that pin



Purpose of Breadboarding:

- Verify your design works in practice

Step 1) Paper Design

- In theory it works...

Step 2) Simulation (CircuitLab)

- Check your design with nonlinear models

Step 3) Breadboard

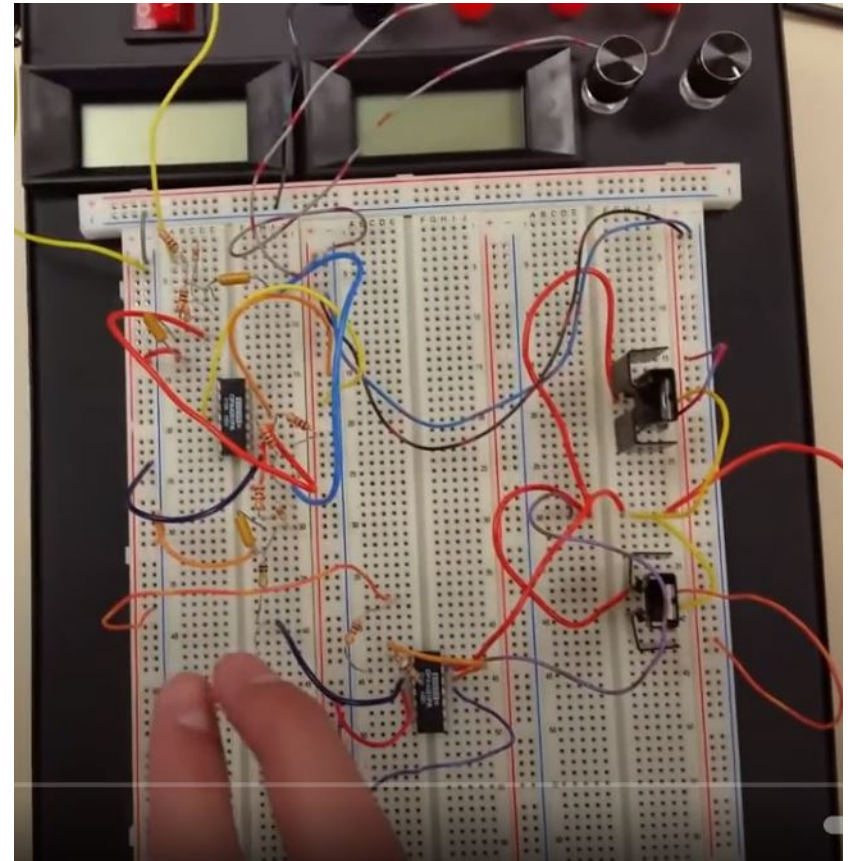
- Check your actual design

Step 4) PCB

- More permanent & abuse tolerant
- Less noise sensitive
- Smaller, easier to package

Notes: With a breadboard

- Changes are fairly easy to implement
 - Components & values can be changed pretty easily
-



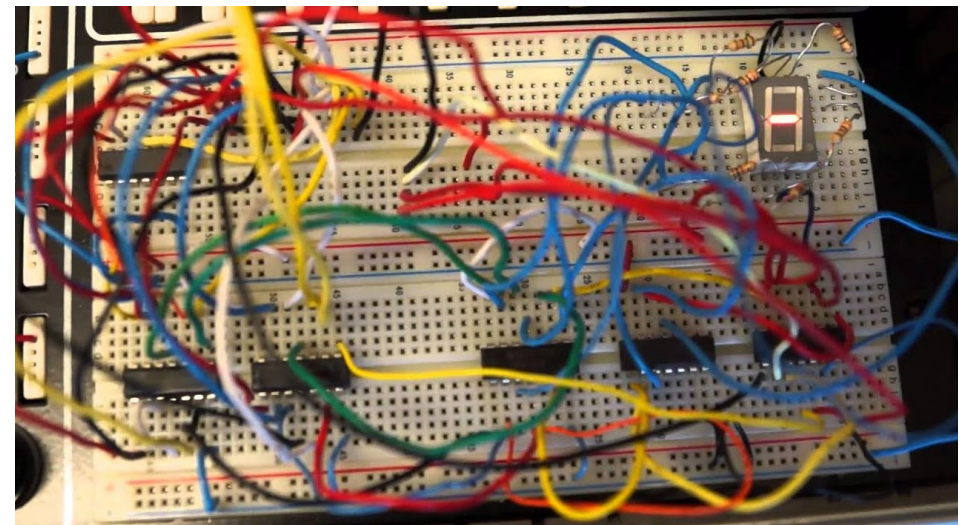
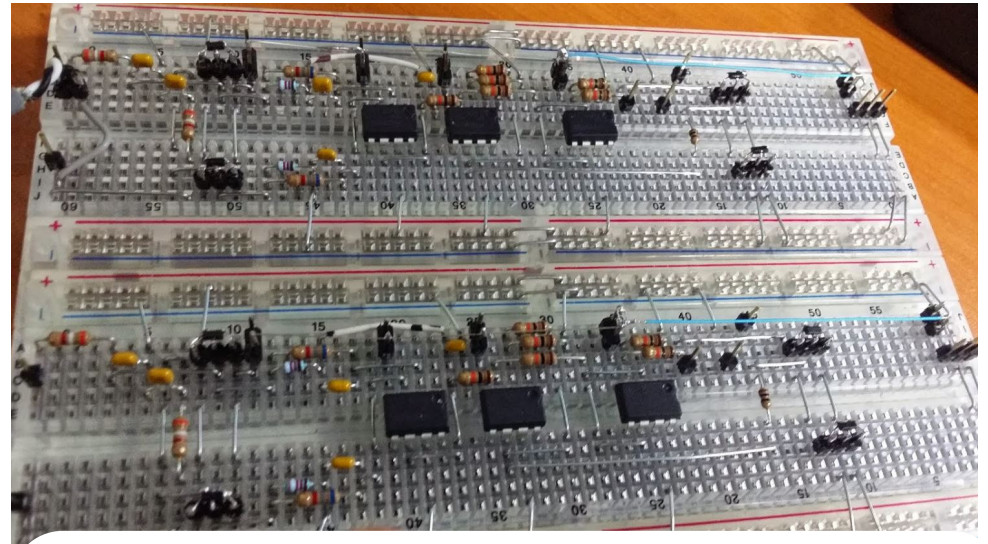
Tricks of Breadboarding

1) Keep Your Circuit Neat

- Use short wires
- Use short component leads
- Organize your breadboard into sections

Keeping your wires short

- Reduces the noise picked up by your wires
- Reduces the chance of a wire falling out
- Helps you see the wiring in your board
- Helps when you need to modify your breadboard circuit.

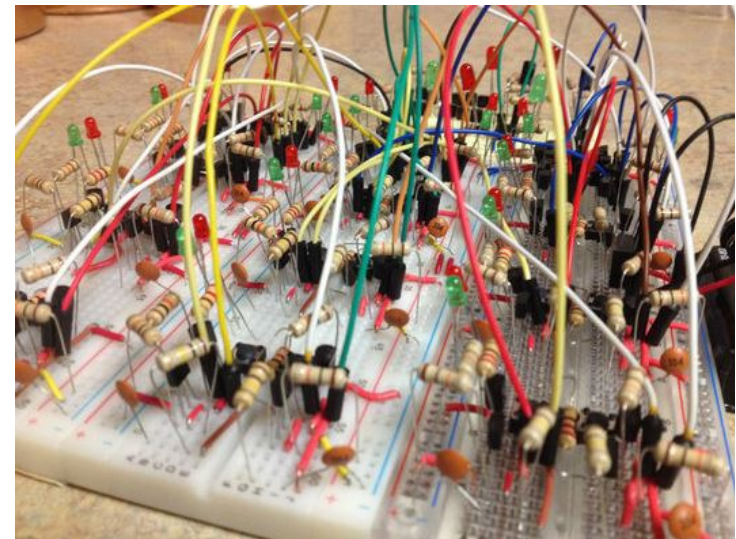
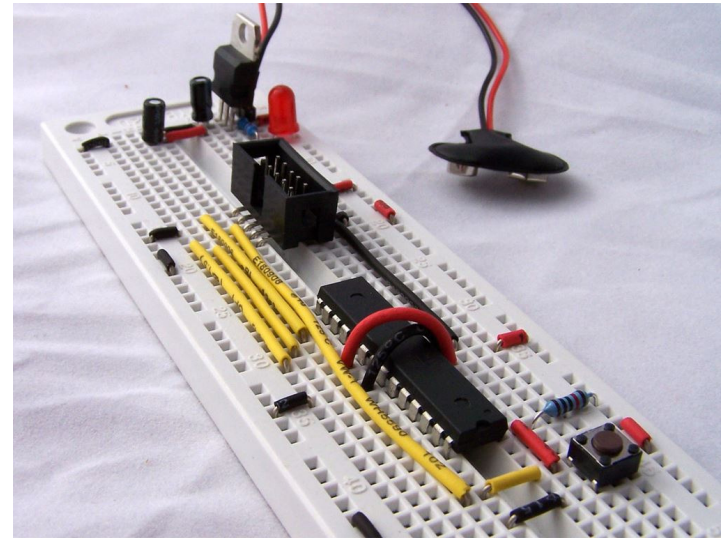


2. Color Code your Wires

- Use red wires for +5V
- Use black wires for ground
- Use different colors for different types of signals.

By color coding your wires,

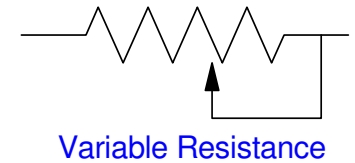
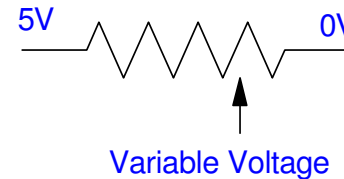
- You can quickly spot if a chip is missing power and/or ground.
- You can quickly see if a signal wire is missing between two ICs



3. Use Potentiometers (2 max)

Potentiometers allow you to

- Adjust voltages (0..5V)
- Adjust resistors (0% to 100%)

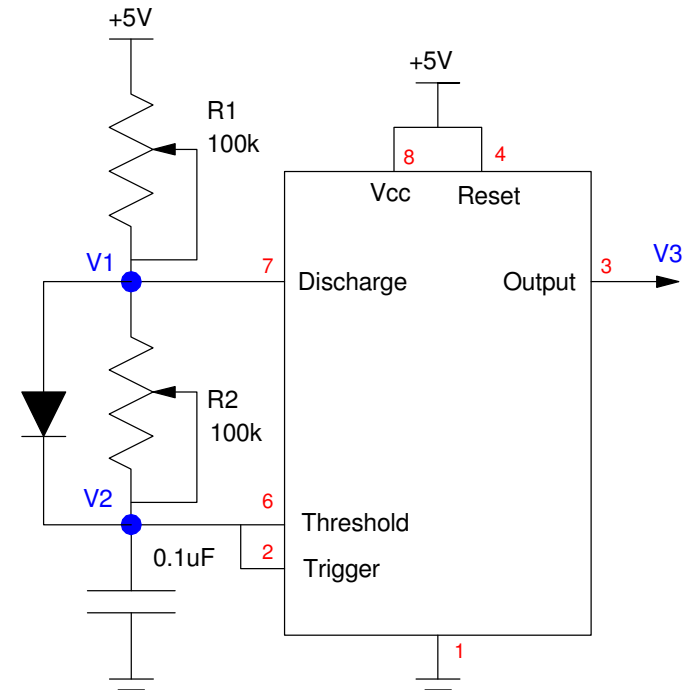


Replacing a resistor with a potentiometer allows you to tune your circuit without having to replace components

- Really useful when you get to PCB's

But...

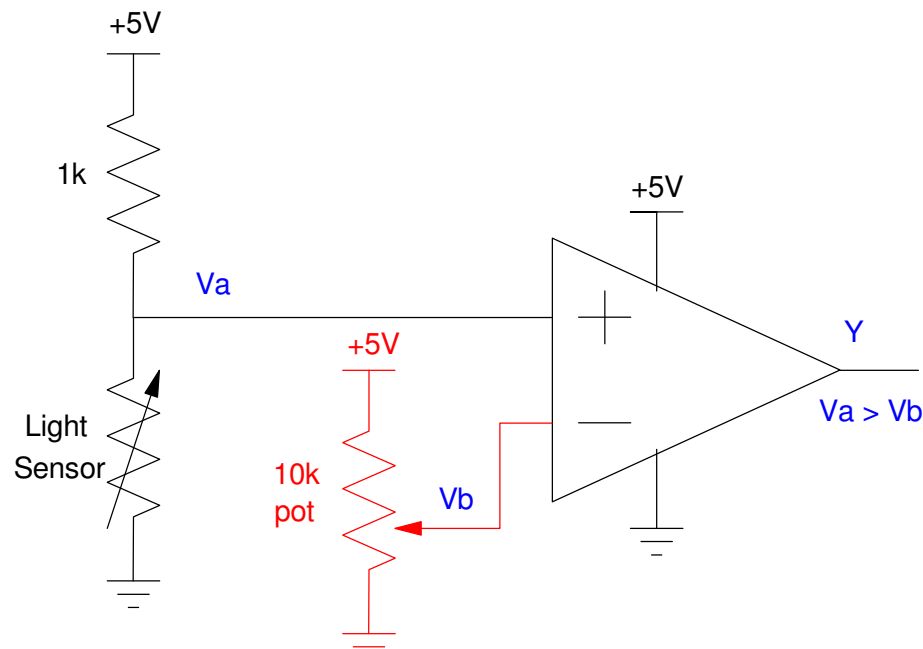
- A resistor costs \$0.02
- Potentiometers cost \$1.55



Use Potentiometers (cont'd)

Example: Variable voltage

- Allows you to adjust the voltage the comparitor switches
- Allows you to adjust the light level where the you turn on



Pots allow you to adjust a voltage

4. Breadboards & Test Points

Some things to think about when using a breadboard are:

- How do you test your circuit?
- What signal do you look at?
- What should the signals look like?
- What procedures do you use?

Note what signals you look at and record what you read. This affects your upcoming PCB layout:

- These same signals should be measured in simulation and on your PCB
 - Test points should be added to your PCB so that you have access to these signals.
-

Test Point Example: Schmitt Trigger Circuit

TP1: 5V

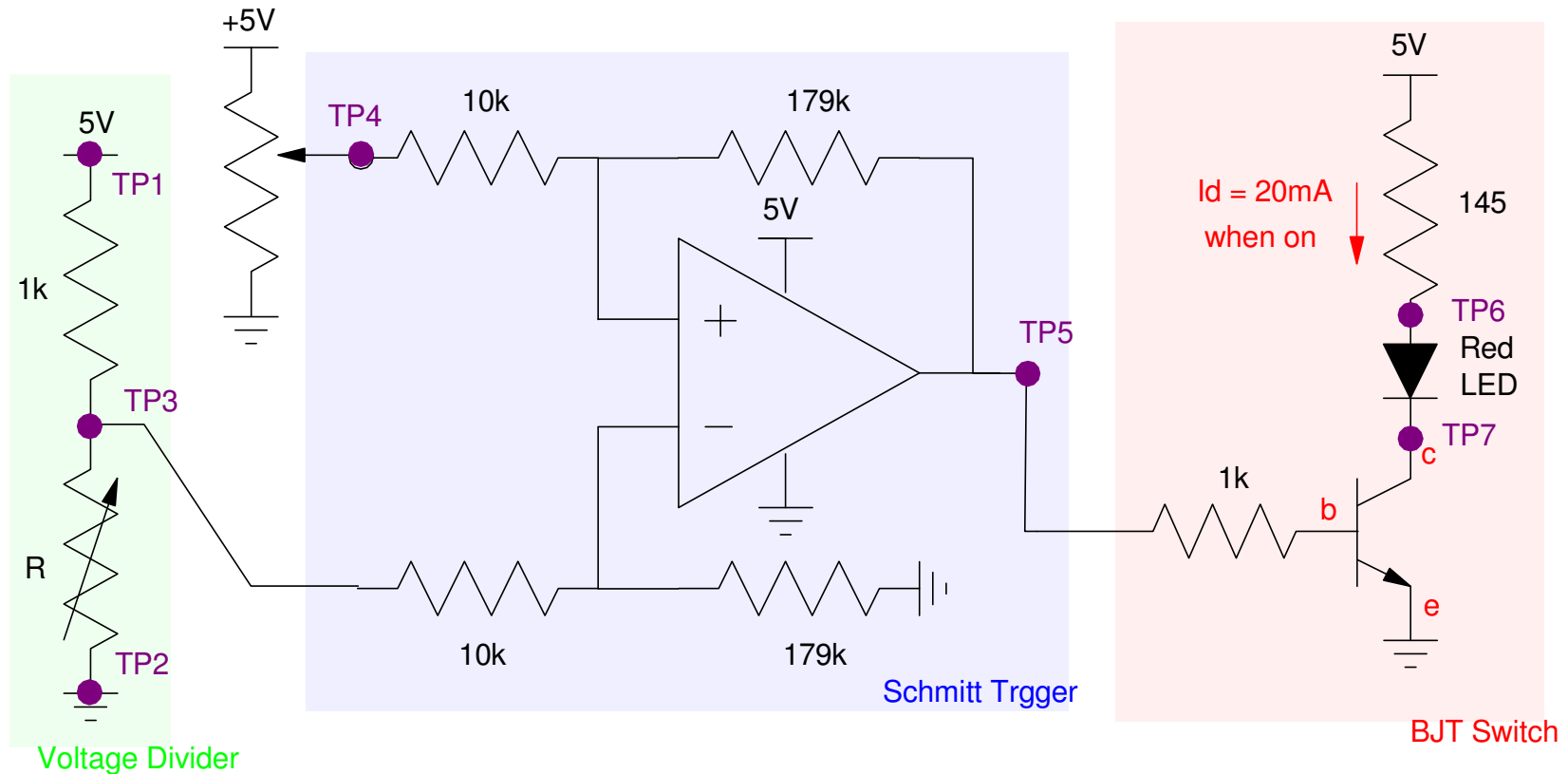
TP4: V(on)

TP7: 0.2V means saturated

TP2: 0V

TP5: 0V=off, 5V=on

TP3: V(sensor) TP6: $I_d = (5V - V_6)/145$



5. Keep Your Breadboard

When done testing your breadboard, keep it together, intact (i.e. don't cannibalize it for parts). If your PCB doesn't work properly, your (working) breadboard circuit will be helpful in debugging what part of your PCB works (and has similar signals), and which part does not work.

This means you'll need two of every part in ECE 401

- One for your breadboard circuit, and
- One for your PCB.
- That's OK.

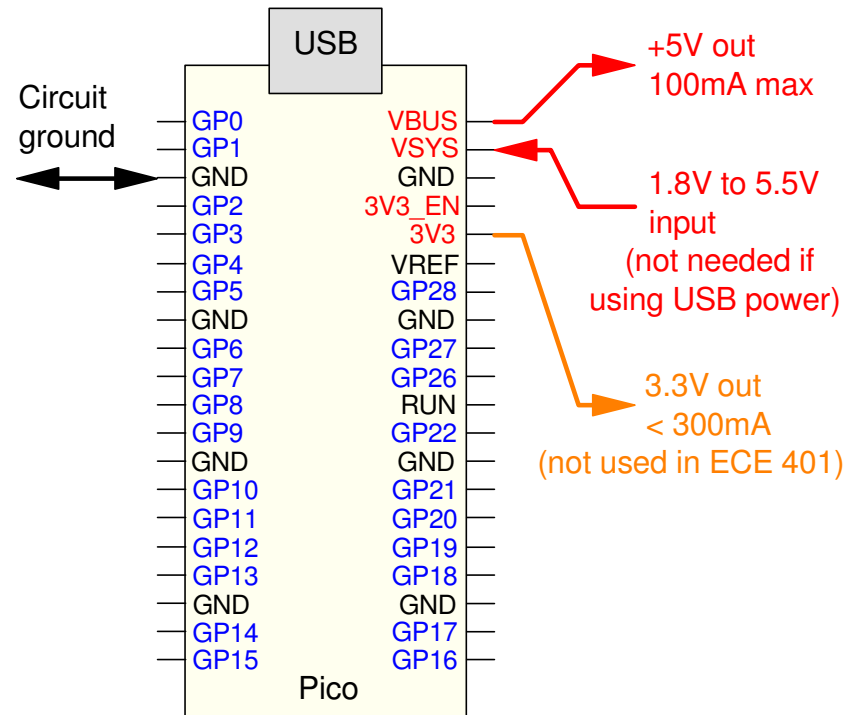
Breadboards with a Raspberry Pi Pico

Power can be provided three ways:

- USB
 - VBUS is powered by the USB
 - Output is 5.0V
 - Capable of 2A (USB limit)
- VBUS (pin 40):
 - Apply 5.0V to VBUS
 - Bypass the USB
- VSYS (pin 39):
 - Provide 1.8V to 5.5V to VSYS.

Either way, the Pico generates 3.3V

- 3.3V (pin 36) Outputs 3.3V
- Capable of up to 300mA



Binary I/O:

- GP0 to GP28 can be binary input or output

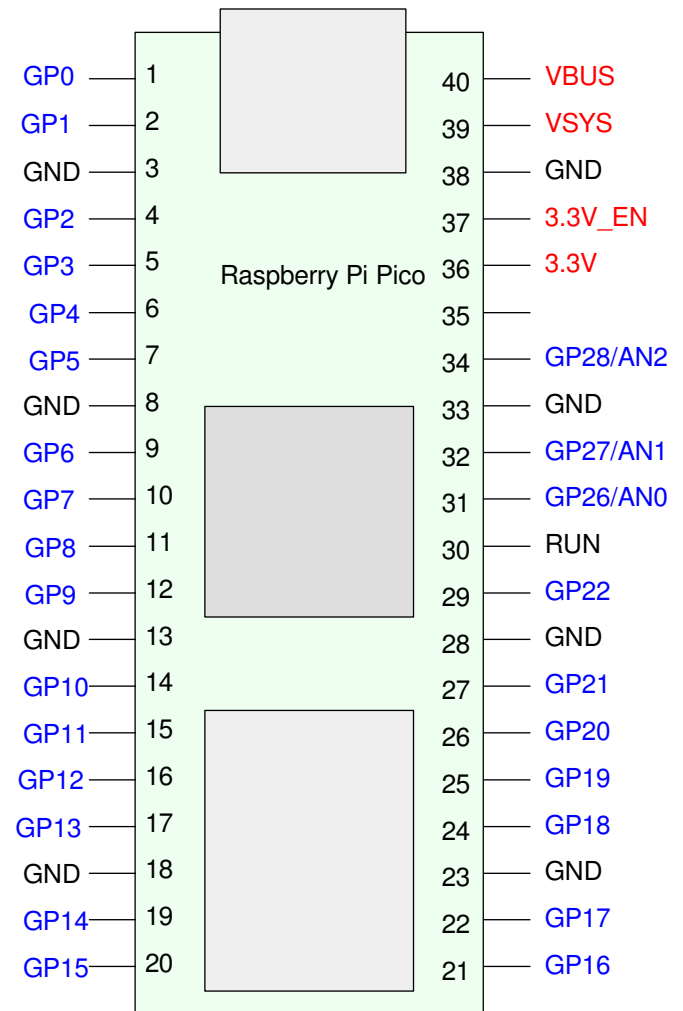
Binary Outputs

- 0V is logic 0
- 3.3V is logic 1
- Can source or sink up to 12mA

Binary Inputs:

- (0.0V - 0.8V) is logic 0
- (2.0V to 3.5V) is logic 1

Do not connect 5.0V to the Pi-Pico's input pins. This may damage the Pi-Pico.



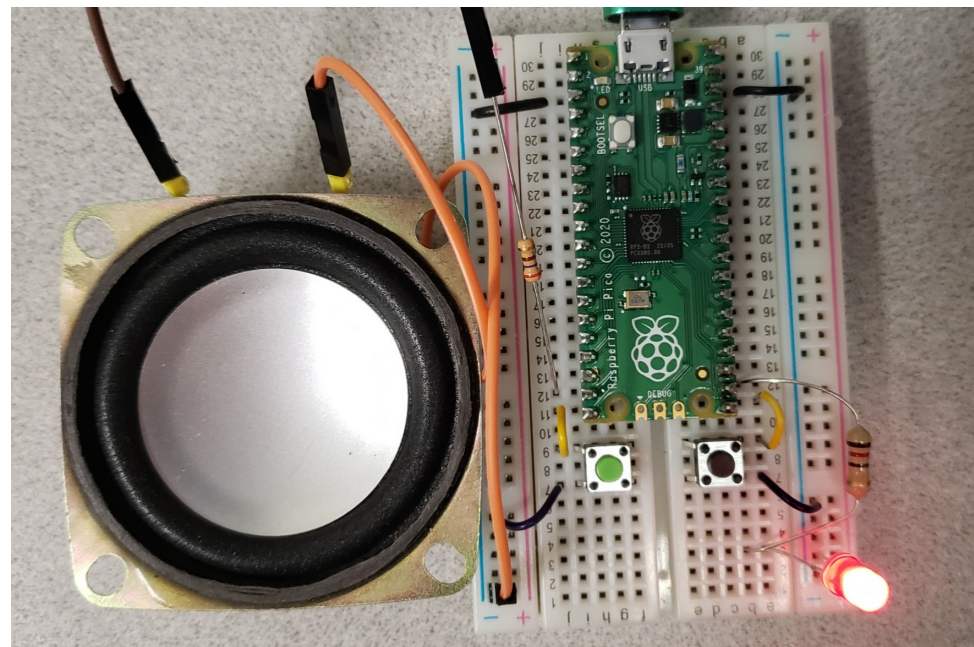
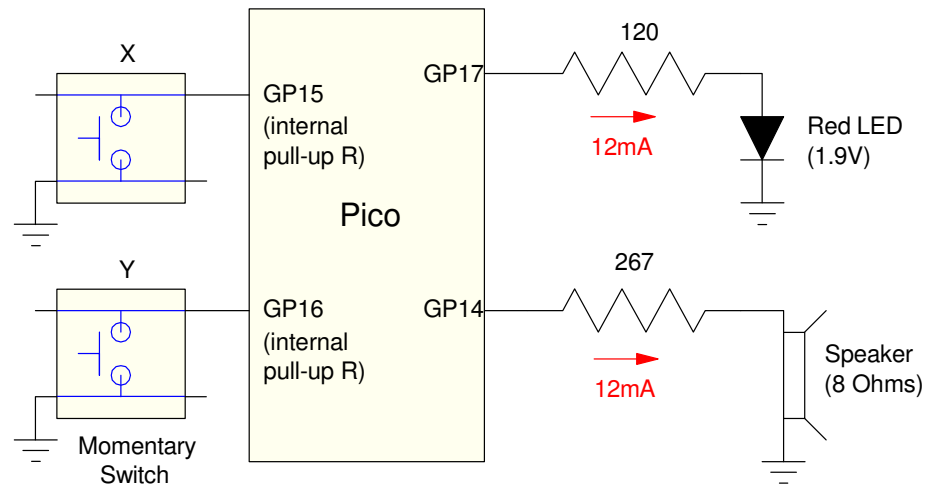
Pi-Pico Example:

Connect your Pico to

- Two push buttons (inputs),
- A speaker (output), and
- A red LED (output)

Schematics is shown on top

Corresponding breadboard shown below



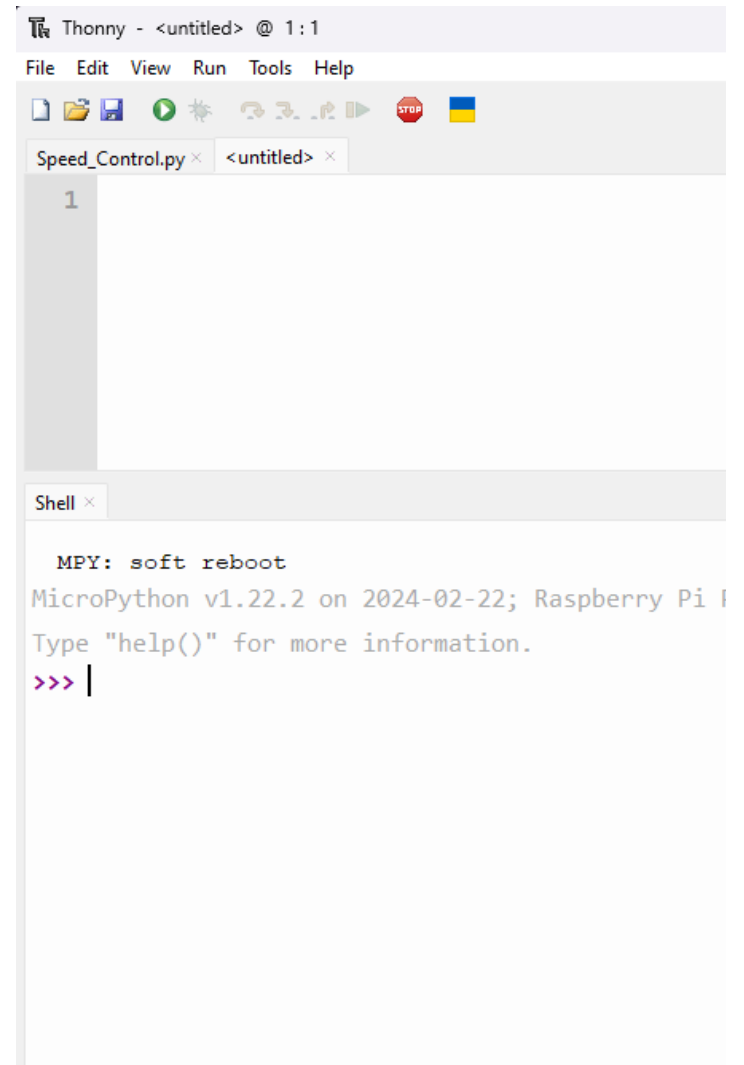
Thonny and MicroPython

Several programming languages are available

- Assembler
- C
- Python (MicroPython for a Pi-Pico)

among others.

In ECE 401, focus on Python



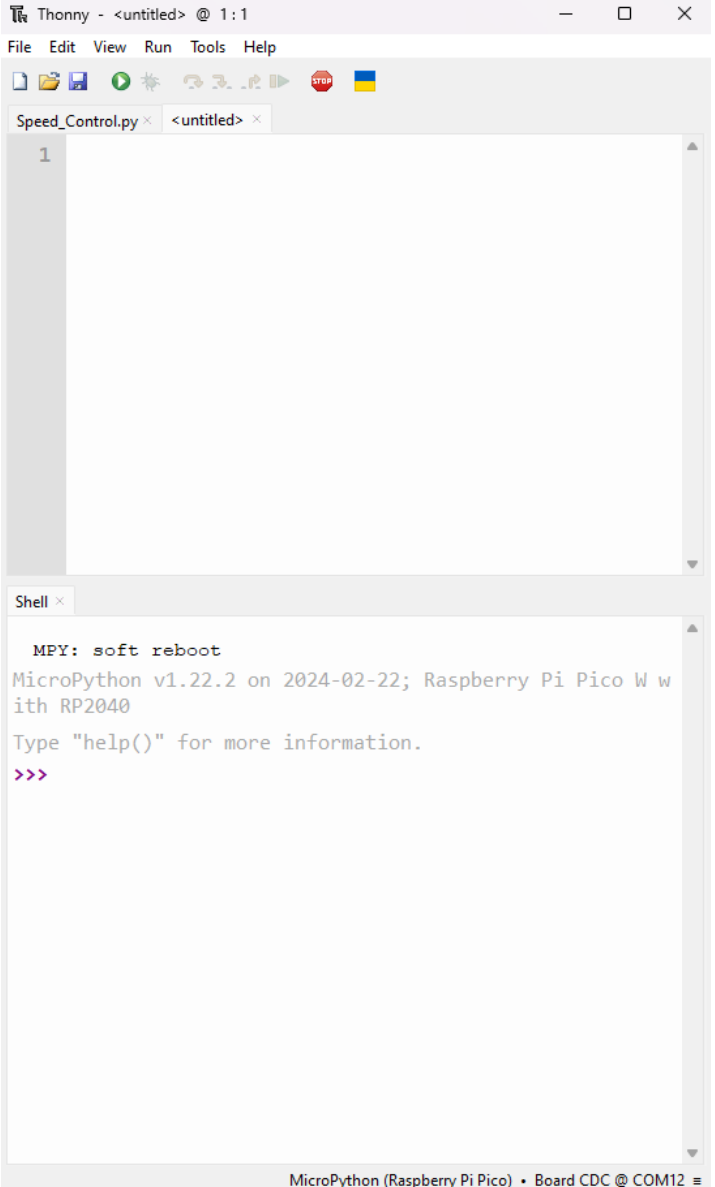
The screenshot shows the Thonny IDE interface. The title bar reads "Thonny - <untitled> @ 1:1". The menu bar includes "File", "Edit", "View", "Run", "Tools", and "Help". Below the menu bar is a toolbar with icons for file operations and execution. The main editor area shows a file named "Speed_Control.py" with a single line of code: "1". Below the editor is a "Shell" window containing the following text: "MPY: soft reboot", "MicroPython v1.22.2 on 2024-02-22; Raspberry Pi", "Type 'help()' for more information.", and a prompt ">>> |".

Installing Thonny

- Locate Thonny 4.1.4
- Download to PC
- Connect to Pico board
- Install Micropython

Click on the lower-right corner

- Select your Pi-Pico chip
- It will prompt you to install MicroPython if this is the first time using your chip



The screenshot shows the Thonny IDE window titled "Thonny - <untitled> @ 1:1". The main editor area contains a file named "Speed_Control.py" with a single line of code on line 1. Below the editor is a "Shell" window displaying the following text:

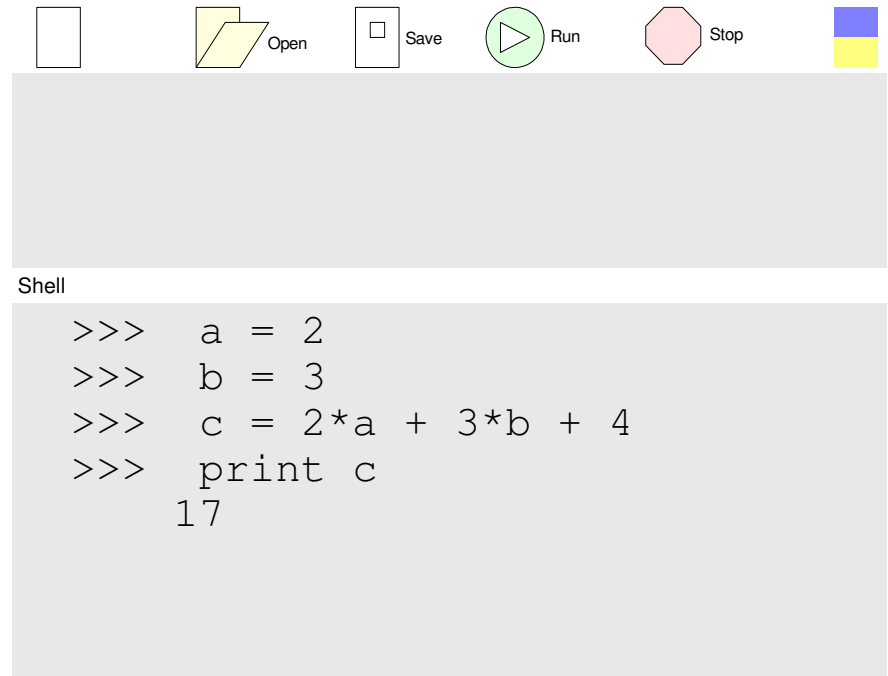
```
MPY: soft reboot
MicroPython v1.22.2 on 2024-02-22; Raspberry Pi Pico W with RP2040
Type "help()" for more information.
>>>
```

The status bar at the bottom of the shell window indicates "MicroPython (Raspberry Pi Pico) • Board CDC @ COM12".

Thonny: Command Window

Python is similar to Matlab:

- The shell window is similar to Matlab's command window
- You can type commands directly in the shell window.
- Python like a calculator:

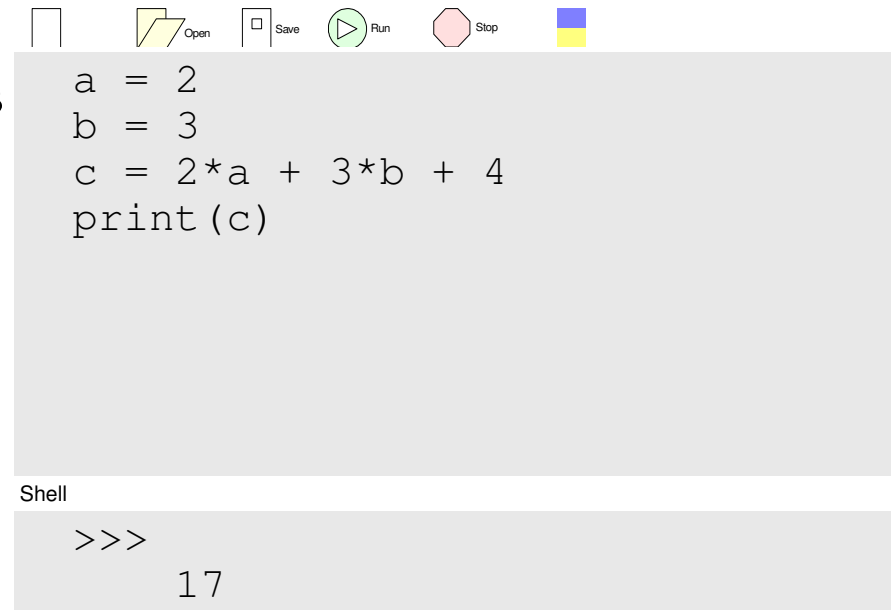


Thonny: Script Window

The top window behaves like Matlab's script window

- Place code to execute here
- Run this code by clicking on *Run*

Results show up in the shell window



Binary I/O with Python

Python is a little different than Matlab.

- Functions are included using the *import* command.
- This makes that library available for use in your program.

Machine Library

- Routines specific to the microcontroller you're using
- Setting I/O pins to input
- Output a square wave

```
import machine

# Output
Button = machine.Pin(0, Pin.OUT)

# Inputs
LED0 = machine.Pin(6, Pin.IN)
LED1 = machine.Pin(7, Pin.IN, Pin.PULL_UP)
LED2 = machine.Pin(8, Pin.IN, Pin.PULL_DOWN)
```

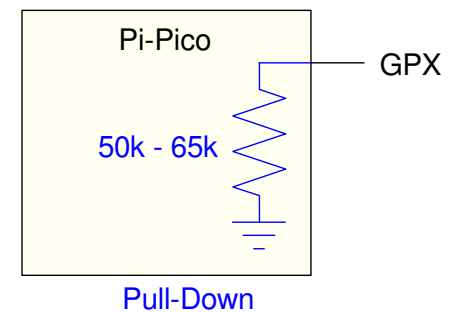
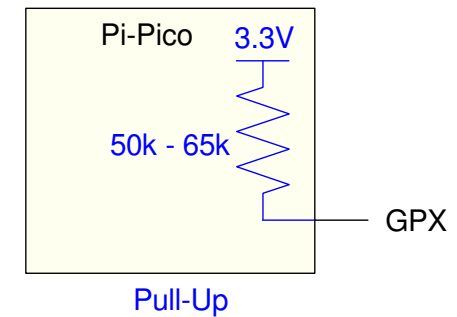
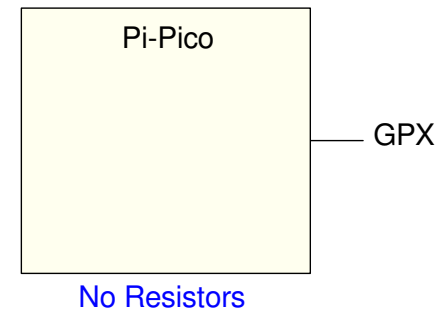
Float / Pull-Up / Pull-Down

Each input pin can have an internal resistor

- Default: no resistor
- Pull-Up: Pulled up to 3.3V with R
- Pull-Down: Tied to ground with R

Pull-up is preferred for push buttons

- Pressing the button ties input GPx to ground
- Safer: No confusion on what logic 0 means
 - ground



Binary Outputs

Each output pin can be set to

- Logic 1 (3.3V) or
- Logic 0 (0V)

Set, Clear, Toggle commands are available

```
# read
Y = Button.value()

# write
LED0.toggle()           # toggle LED0 on/off
LED0.value(1)           # set LED0
LED0.value(0)           # clear LED0
LED0.low()              # clear LED0
LED0.high()             # set LED0
```

Time Library

- *Shows better in the video*

The *time* library contains wait routines:

- `sleep(x)`: pause `x` seconds. `x` can be a floating-point number
- `sleep_ms(x)`: pause `x` milliseconds. `x` must be an integer
- `sleep_us(x)`: pause `x` microseconds. `x` must be an integer.

Example: Every 500ms

- Read the button values
- Display their value in the shell window



```
from machine import Pin
from time import sleep_ms

B0 = Pin(15, Pin.IN, Pin.PULL_UP)
B1 = Pin(16, Pin.IN, Pin.PULL_UP)

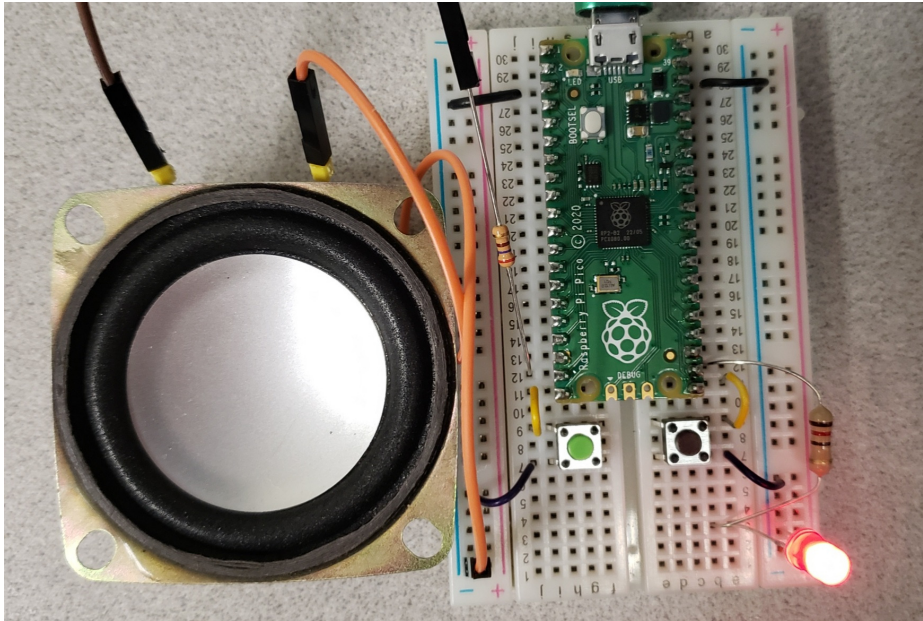
while(1):
    X = B0.value()
    Y = B1.value()
    print(X, Y)
    sleep_ms(500)
```

shell

```
MPY: soft reboot
1 1
0 1
1 1
1 0
1 0
```

Time Library (cont'd)

Blink and LED on and off every 2 seconds



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

```
LED = Pin(17, Pin.OUT)
```

```
LED.value(1)
print('LED On')
sleep(2)
LED.value(0)
print('LED Off')
```

Shell

```
>>>
LED On
LED Off
```

If and While Statements

for-loops, while-loops, and if-statements are really useful

Note: Python does not use end-statements

- Indentation indicated which lines are within a loop
- Empty loops are not allowed
- *pass* behaves like a *nop* statement (do nothing)

Carriage returns and indentations have meaning

- unlike C
- 4-spaces are standard per level
- Anything is allowed - just be consistent

```
for i in range(0, 6):
    d1 = i
    for j in
range(0, 6):
        d2 = j
        y = d1 + d2
```

```
t = 0
dt = 0.01
while(t < 5):
    y = sin(t)
    t += dt
```

```
if(x < 3):
    y = 2*x + 4
elif(x < 5):
    y = 3 - 2*x
else:
    y = 0
```

Example: Turn on and off a light

GP15:

- Input variable B0
- Turn on the light
- Logic 0 when pressed

GP14

- Input variable B1
- Turn off the light
- Logic 0 when pressed

GP17

- Output variable
- Connected to an LED through R
- (limit the current to <12mA)



```
from machine import Pin
from time import sleep_ms

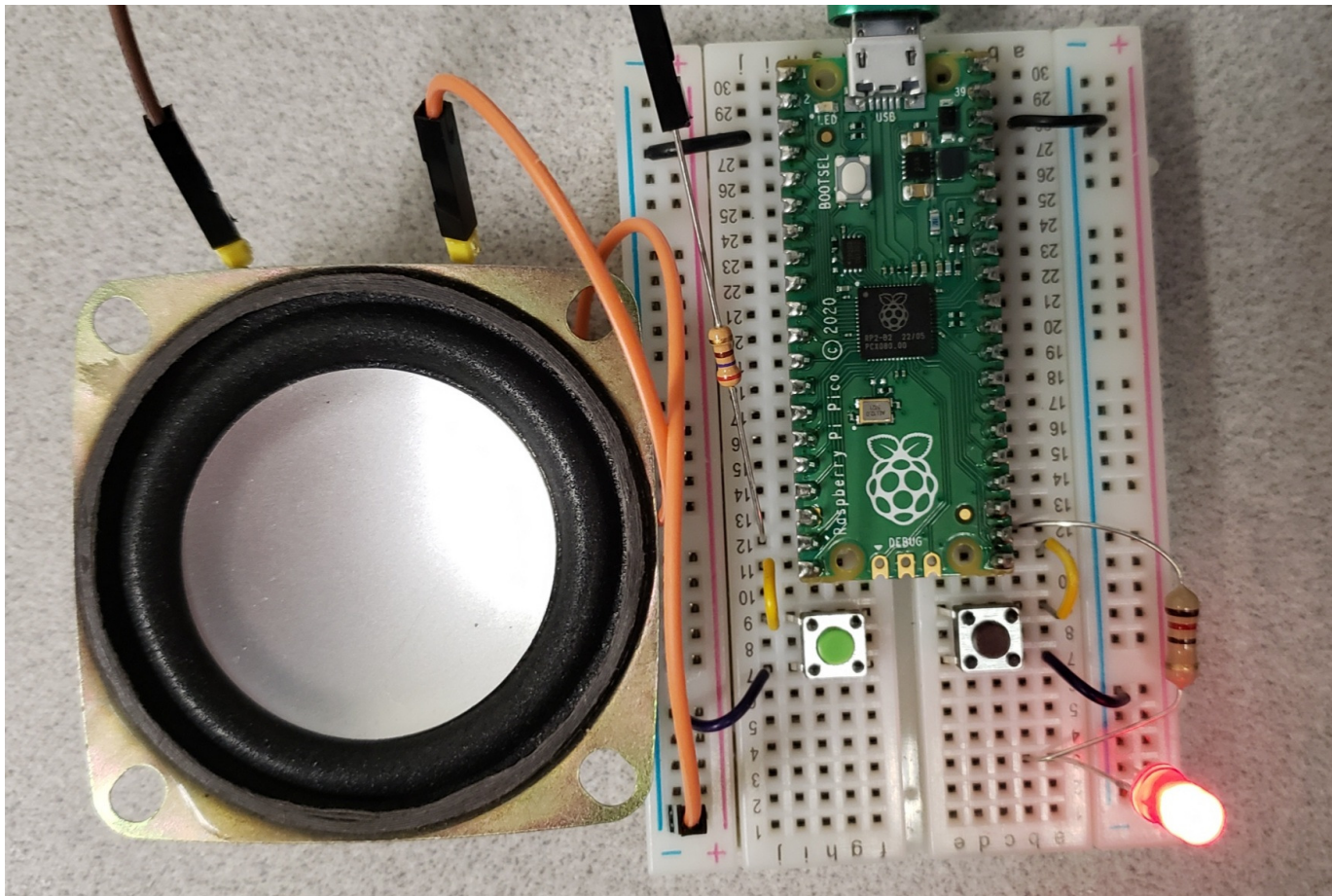
B0 = Pin(15, Pin.IN, Pin.PULL_UP)
B1 = Pin(16, Pin.IN, Pin.PULL_UP)
LED = Pin(17, Pin.OUT)
Spkr = Pin(14, Pin.OUT)

while(1):
    X = B0.value()
    Y = B1.value()
    if(X == 0):
        LED.value(1)
    if(Y == 0):
        LED.value(0)

print(X, Y, LED.value())
sleep_ms(100)
```

Example: Turn on and off a light

Breadboard Results



Two-Key Piano:

PWM allows you to

- Output a square wave
- At a given frequency
 - units Hz
- At a given duty cycle
 - `duty_u16(0) = 0%`
 - `duty_u16(0xFFFF) = 100%`
 - `duty_ns(x) = x us on-time`

This program plays

- 220Hz when B0 is pressed
- 260Hz when B1 is pressed



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

```
B0 = Pin(15, Pin.IN, Pin.PULL_UP)
B1 = Pin(16, Pin.IN, Pin.PULL_UP)
LED = Pin(17, Pin.OUT)
```

```
Spkr = PWM(Pin(14))
Spkr.freq(220)
Spkr.duty_u16(0)
```

```
while(1):
    X = B0.value()
    Y = B1.value()
    if(X == 0):
        Spkr.freq(220)
        Spkr.duty_u16(0x8000)
    elif(Y == 0):
        Spkr.freq(260)
        Spkr.duty_u16(0x8000)
    else:
        Spkr.duty_u16(0)
    sleep_ms(10)
```

For Loops

Similar to Matlab

- Indentation indicates statements within the for-loop

range(0,5)

- Start at 0
- Increment by one each loop
- Continue while <5
 - Slightly different than Matlab

range(0,5,2)

- Step size = 2

[1,3,5,7,11]

- Step through the array



```
for i in range(0,5):  
    print(i, i*i)  
  
for i in range(0,5, 2):  
    print(i, i*i)  
  
for i in [1,3,5,7]:  
    print(i, i*i)
```

Shell

```
>>>  
0      0  
1      1  
2      4  
3      9  
4      16  
  
0      0  
2      4  
4      16  
  
1      1  
3      9  
5      25  
7      49
```

Counter in Python (take 1)

Using the previous hardware,
count how many times button
GP15 is pressed



```
from machine import Pin
from time import sleep

Button = Pin(15, Pin.IN, Pin.PULL_UP)

N = 0

while(1):
    while(Button.value() == 0):
        pass
    while(Button.value() == 1):
        pass
    N = N + 1
    print(N)
```

Shell

```
>>>
1
2
3
4
5
```

Subroutines in Python

In MicroPython, subroutines are defined by the keyword *def*, sort for *define*. The simplest example would be a routine which

- is passed nothing,
- returns nothing, and
- simply prints 'hello' when called:

When you press the *run* command

- Python installs the subroutine defined as *SayHello*
- It then runs the main routine (instruction following all of the definitions)



```
def SayHello():
    print('hello')

# Start of main routine
SayHello()
```

shell

```
>>>
hello
```

Passing parameters to a subroutine

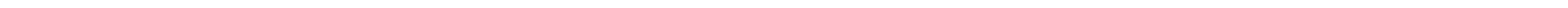
You can pass multiple parameters by simply including them in the definition



```
def Multiply(A, B):  
    C = A * B  
    print(A, ' * ', B, ' = ', C)  
  
# Start of main routine  
Multiply(4, 6)
```

shell

```
>>>  
4 * 6 = 24  
  
>>> Multiply(8, 7)  
8 * 7 = 56
```



Returning Numbers

You can return one number

You can return multiple numbers

- Received as an array, or
- Received as four separate variables



```
# Example of Returning four Numbers
def Operate(A, B):
    C0 = A + B
    C1 = A - B
    C2 = A * B
    C3 = A / B
    return([C0, C1, C2, C3])

# Start of main routine
X = Operate(4,6)
print(X)
```

shell

```
>>>
[10, -2, 24, 0.666667]

>>> C = Operate(8,7)
>>> print(C)
[15, 1, 56, 1.4142857]

>>> [a, b, c, d] = Operate(8,7)
>>>> print(a, b, c, d)
15, 1, 56, 1.4142857
```

Program Execution on Startup

Make your Pi-Pico blink three times at 2Hz on power-up

- On for 100ms
- Off for 400ms
- repeat 3x

First, create a program (assume GP16 has an LED attached)



```
from machine import Pin
from time import sleep

LED = Pin(16, Pin.OUT)

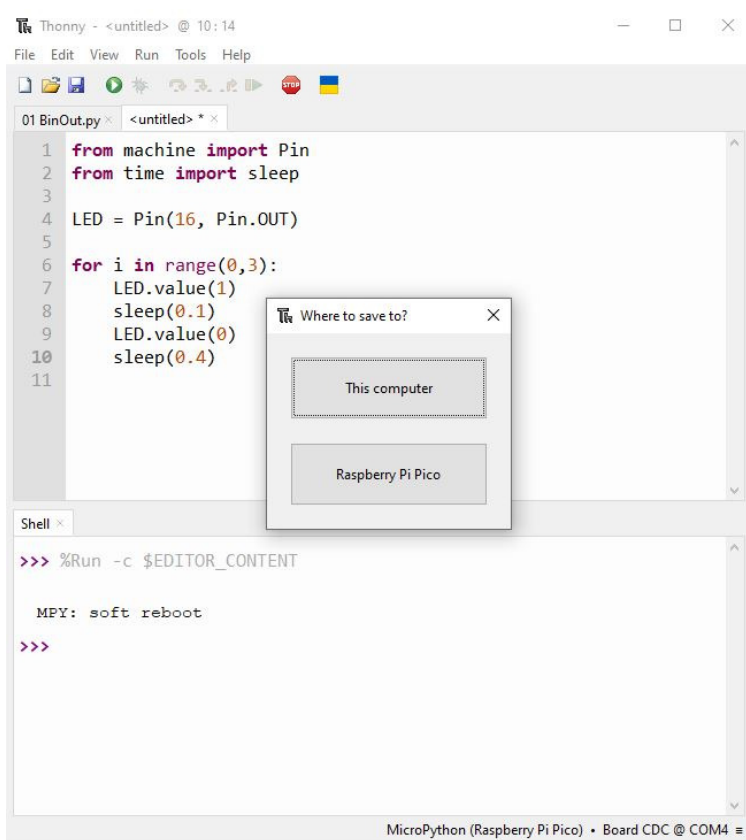
for i in range(0,3):
    LED.value(1)
    sleep(0.1)
    LED.value(0)
    sleep(0.4)
```

Shell

Once this runs,

- Go to File Save As
- Select save to Raspberry Pi Pico
- Save as *main.py*

On power up, this program will execute.



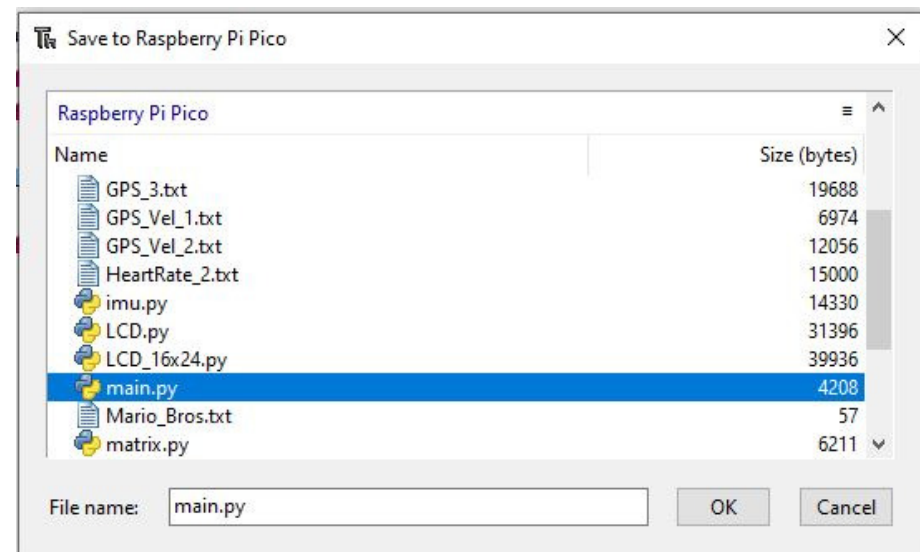
The screenshot shows the Thonny IDE interface. The main editor window displays a Python script named '01 BinOut.py' with the following code:

```
1 from machine import Pin
2 from time import sleep
3
4 LED = Pin(16, Pin.OUT)
5
6 for i in range(0,3):
7     LED.value(1)
8     sleep(0.1)
9     LED.value(0)
10    sleep(0.4)
11
```

A 'Where to save to?' dialog box is open, showing two options: 'This computer' and 'Raspberry Pi Pico'. The 'Raspberry Pi Pico' option is selected. Below the editor is a shell window with the following output:

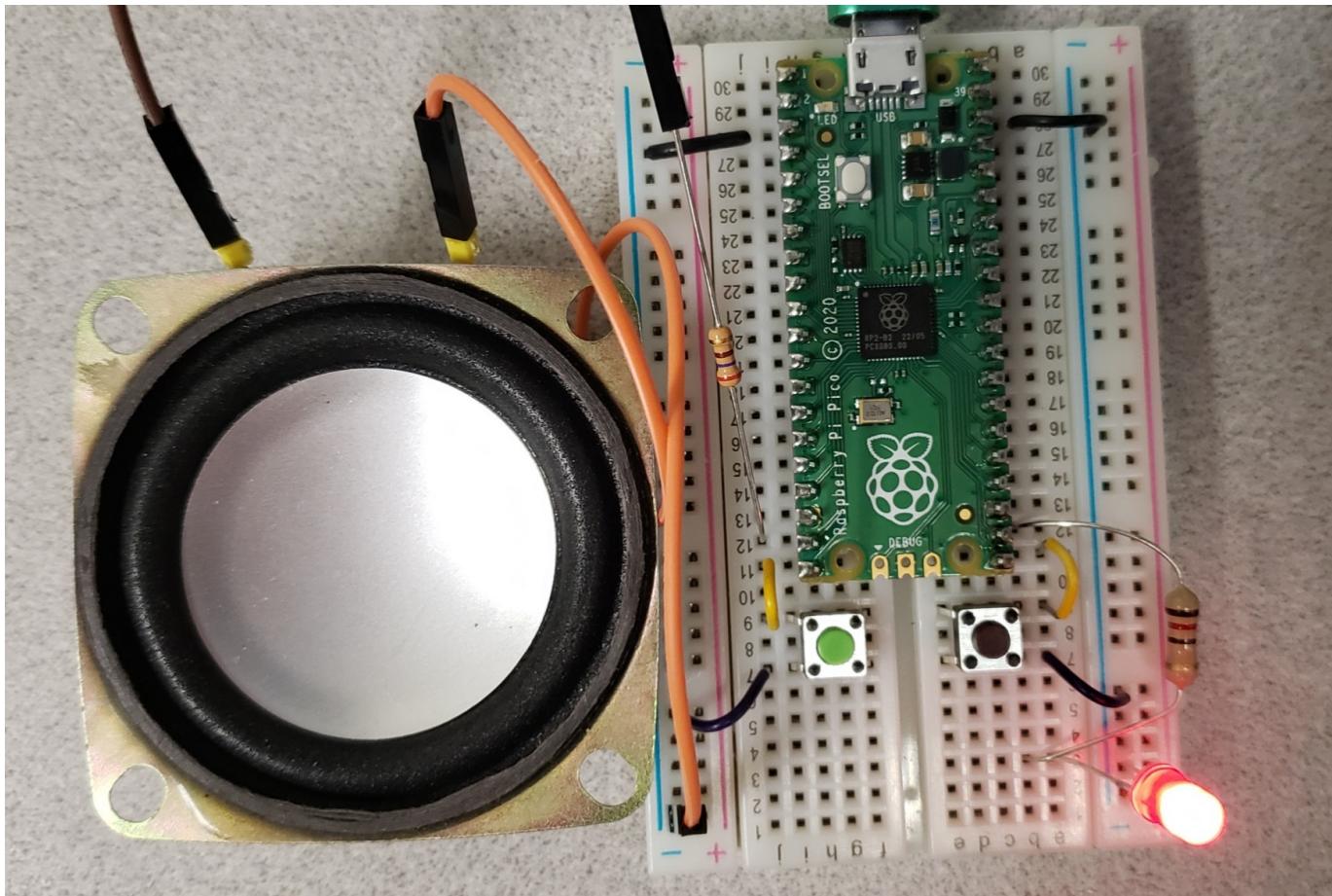
```
>>> %Run -c $EDITOR_CONTENT
MPY: soft reboot
>>>
```

The status bar at the bottom indicates 'MicroPython (Raspberry Pi Pico) • Board CDC @ COM4'.



Result

On power up, the Pico's LED (top-left corner) blinks three times



For more information

ECE 476/676 Advanced Embedded Systems

- <https://www.BisonAcademy.com/ECE476/index.html>

Also also

- <https://www.w3schools.com/python/default.asp>
 - <https://docs.micropython.org/en/latest/pyboard/tutorial/index.html>
 - <https://docs.micropython.org/en/latest/library/index.html>
 - <https://www.fredscave.com/02-about.html>
-

Homework #3:

Fill In Section #2: Requirements

- Engineering Requirements
- Gantt Chart

Engineering Requirements (partial list):

- Must operate off of 5VDC
- Must include at least one integrated circuit
- Must include at least one LED with $I_d = 20\text{mA} \pm 5\text{mA}$
- Must include at least one NPN and one PNP transistor

Power supply = 9V battery (mark +/- polarity)

- use a LM7805 regulator to drop 9V to 5V
- Must have a reverse-polarity protection diode
- Must have a 1/4 Watt 1-Ohm resistor in series with the power supply

(continued next page)

Update Section #3: Paper Design in your OneNote document

Include:

- Your circuit schematics
- Calculations for R's and C's
- Calculations for voltages you expect to see.

Note: If you're using a microprocessor, assume the output pins are either 0V or 3.3V.
