Designs using Python & a Raspberry Pi-Pico

ECE 401 Senior Design I

Week #5

Please visit Bison Academy for corresponding lecture notes, homework sets, and videos

Introduction

In Senior Design I, you can

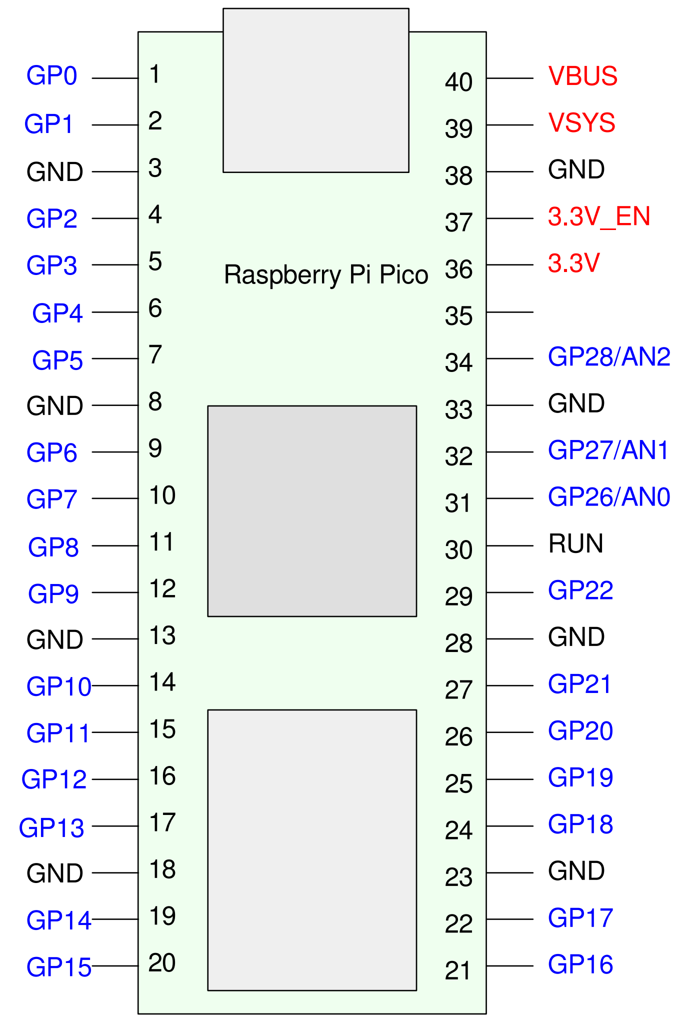
- Use a microcontroller, or
- Not use a microcontroller

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

- 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

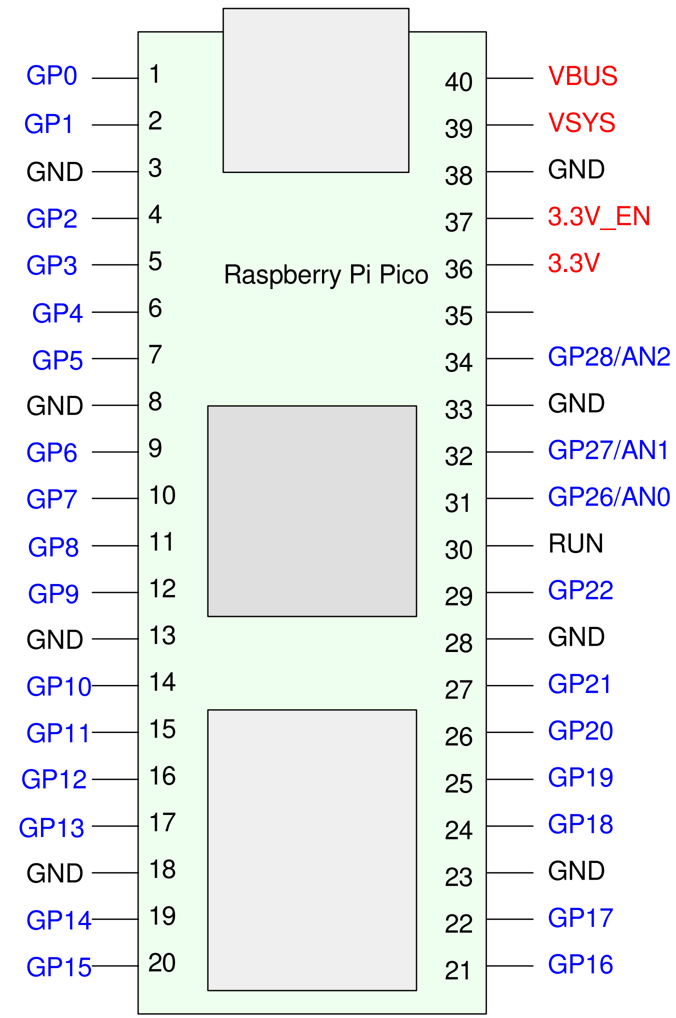
Power:

Power can be provided through:

- USB:
  - The USB cable provides 5V to the Pi-Pico
- VSYS (pin 39):
  - Provide 1.8V to 5.5V to VSYS.

Either way, the Pico geneates two outputs:

- VBUS (pin 40): Outputs 5V
- 3.3V (pin 36) Outputs 3.3V



Binary I/O:

- GP0 to GP28 can be binary input or output

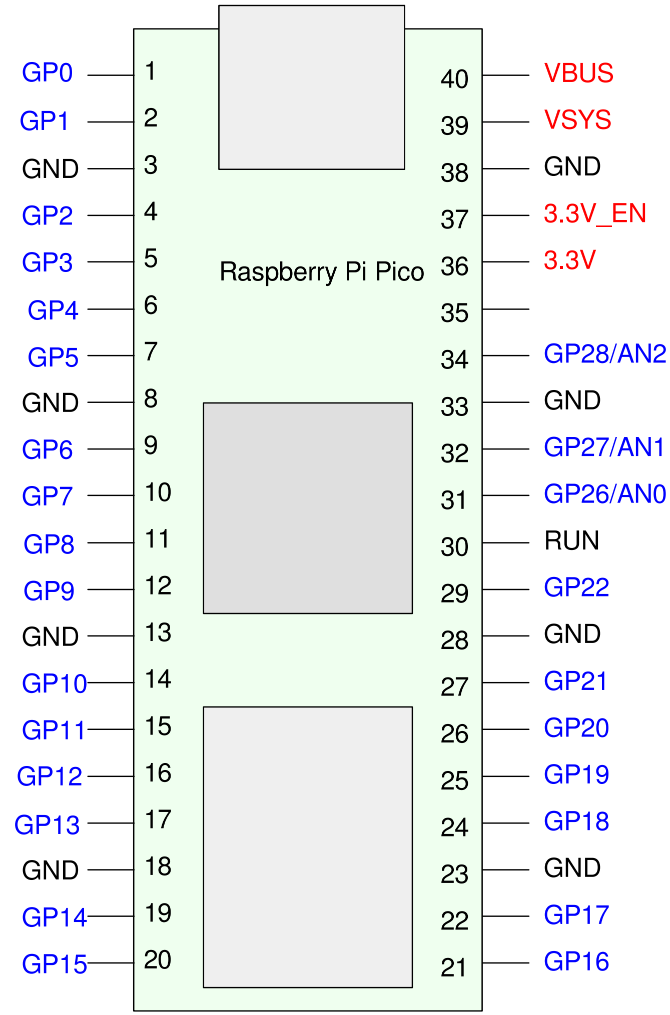
Binary Outputs

- 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.



Internal Pull-Up / Pull-Down Resistors

If set as input, you can include

- An internal pull-up resistor, or
- A pull-down resistor.

Pull-Up:

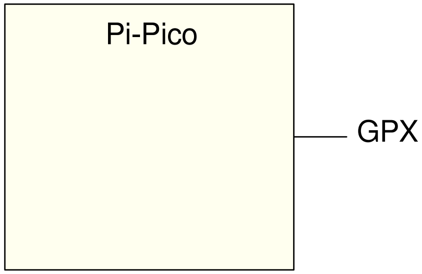
- Floating lead is read as logic 1
- Ground pin to read as logic 0

Pull-Down

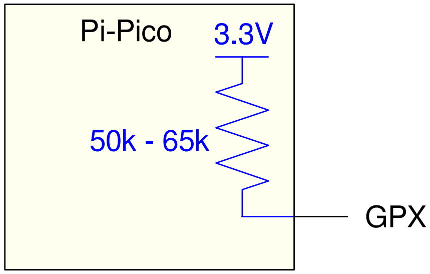
- Floating lead is read as logic 0
- Tie to +3.3V to read as logic 1

Pull-up is preferred for push-buttons

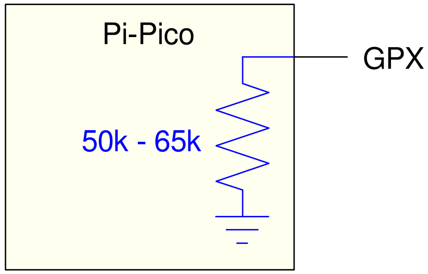
- Safer
- No confusion about what logic 0 means



No Resistors



Pull-Up

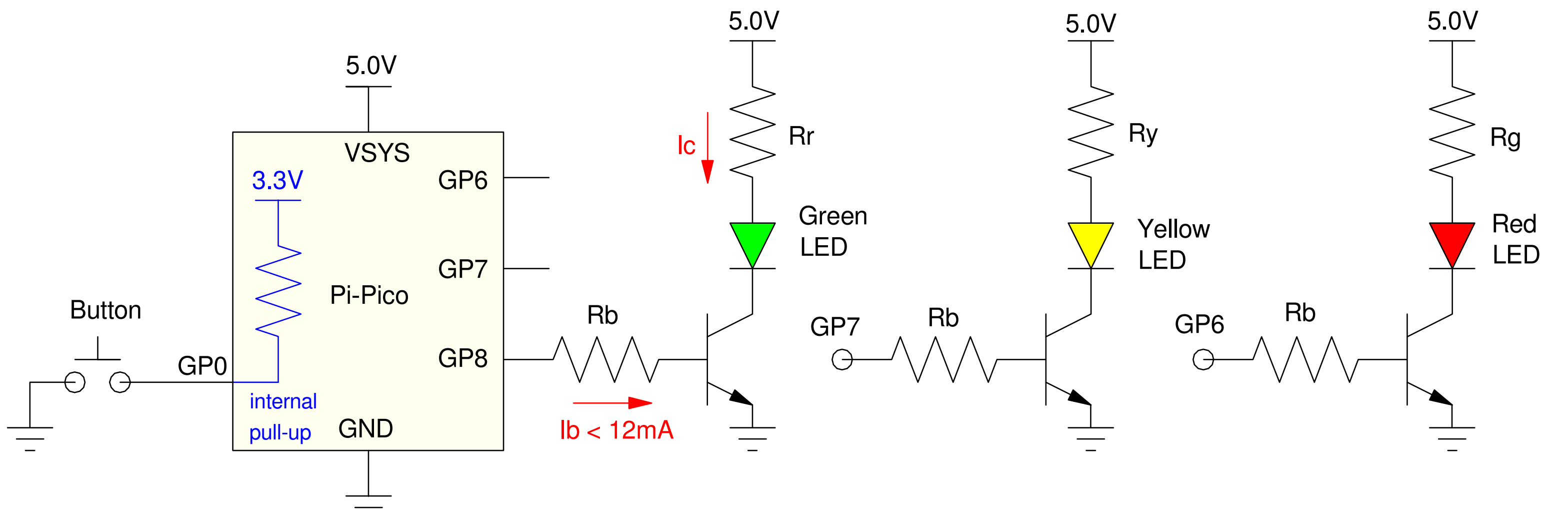


Pull-Down

Hardware Example:

- Power: from the USB cable
- Inputs: Push button connected to GP0
- Outputs: Three LEDs connected to GP6, GP7, and GP8

The hardware could be:

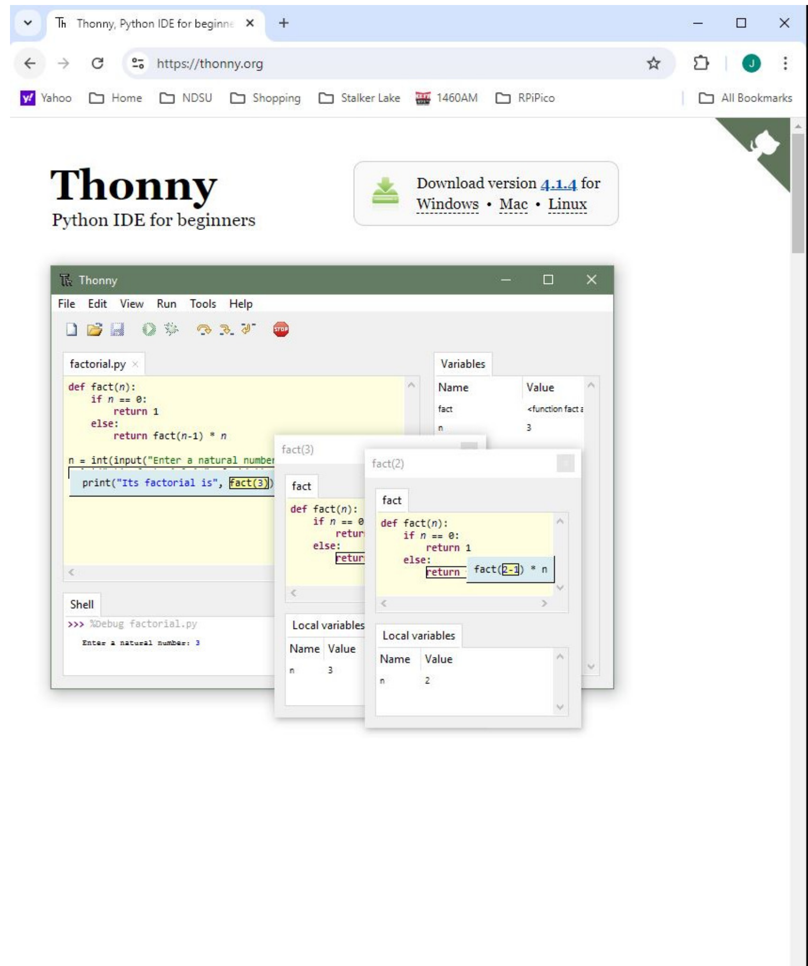


Software: Thonny and MicroPython

Several programming languages are available for a Pi-Pico, including

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

among others. In this lecture, we focus on MicroPython.



MicroPython is version of Python

- Reduced functionality
- Fits on a microcontroller like a Pi-Pico
- Free (!)
  - www.Thonny.org

Python is very similar to Matlab:

Both are interpretive languages:

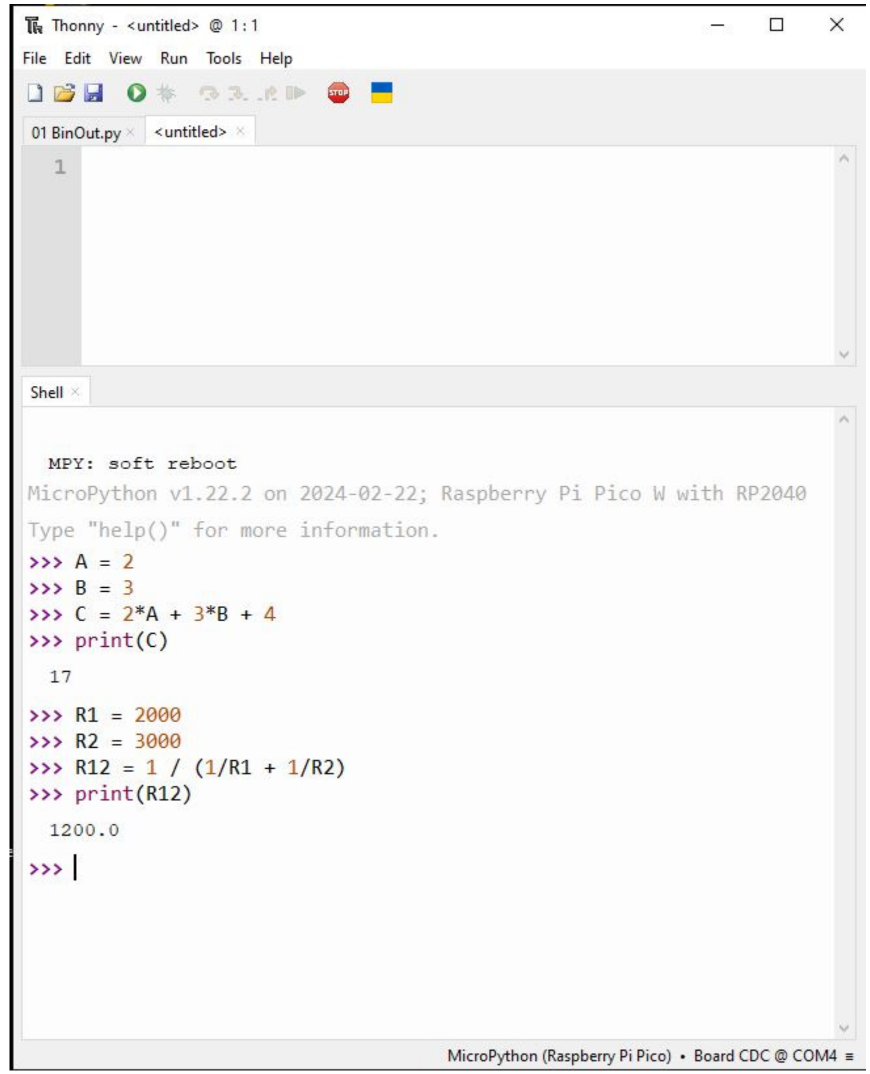
- Program executes line by line
- You can see the result after each line executes

Both use similar syntax

- Code that works in Matlab mostly works in Python

Both use a similar console

- Program window
- Command window



Thonny: On Start-Up

Icons

- File New / Open / Save
- Run (run the script)
- Stop (stop the program - clear memory)
- Donate to Ukraine

Script Window

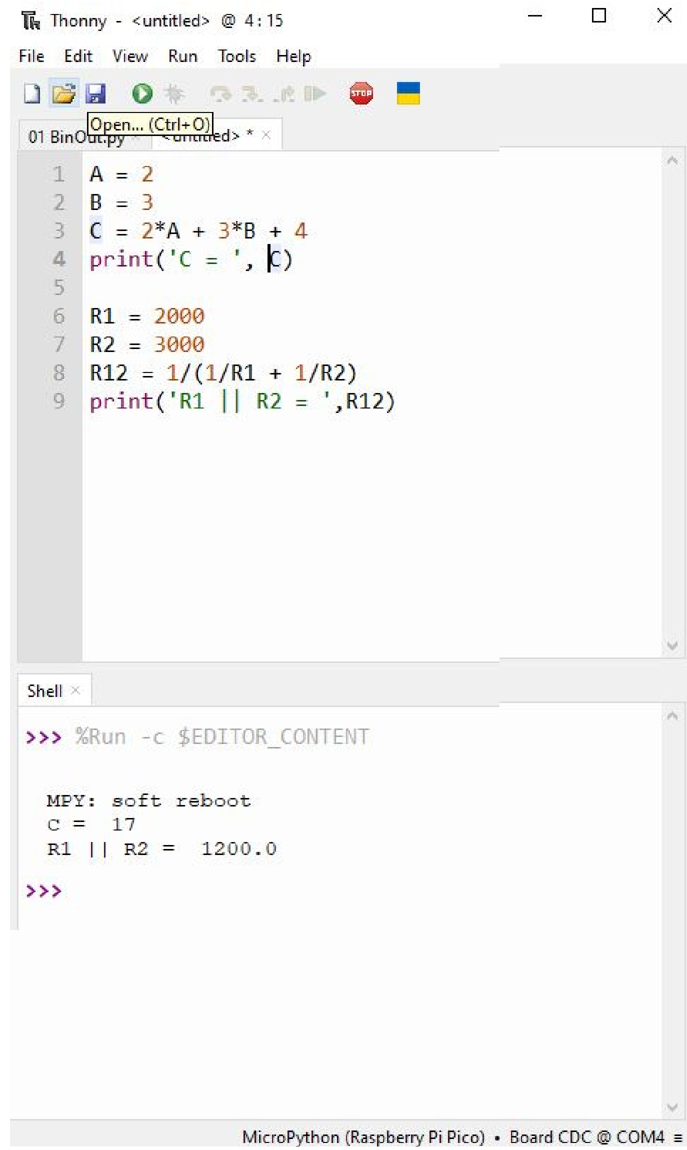
- programs you can execute

Shell Window

- Command-window in Matlab-speak
- Type in code by hand
- See result of program execution

Lower-Right Corner

- What connected to



Python vs. Matlab

Python is similar to Matlab

- You can type commands directly in the shell window.
- You can use Python like a calculator

Shell

```
>>>  a = 2
>>>  b = 3
>>>  c = 2*a + 3*b + 4
>>>  print c
    17
```

- You can also place this code in the script window
- Run executes the program

```
a = 2
b = 3
c = 2*a + 3*b + 4
print(c)
```

Shell

```
>>>
    17
```

Declaring Variables

You don't have to declare variables

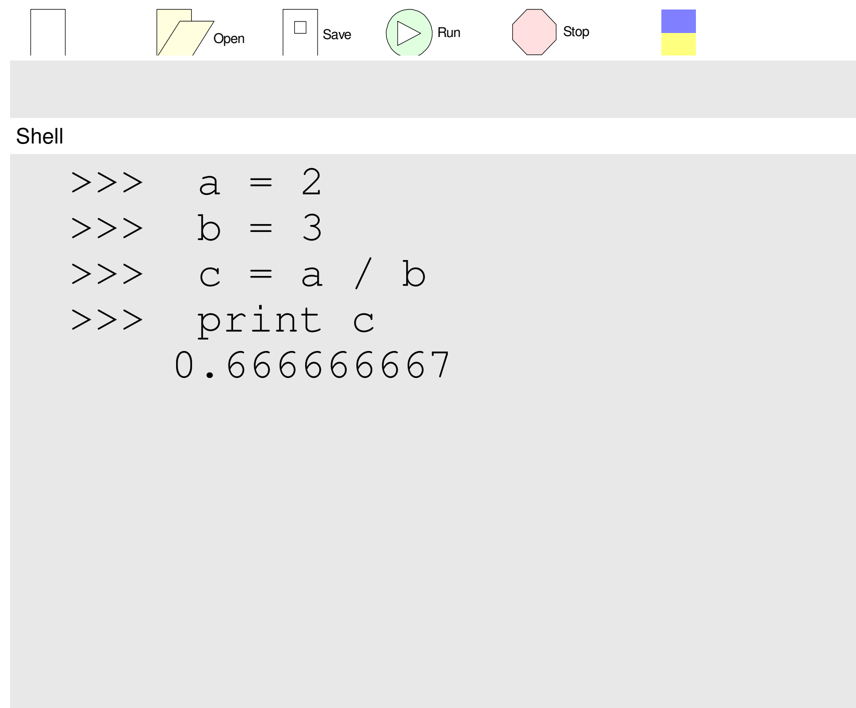
- Creating them on the fly is OK

Variable types are automatic

- a = 3: integer
- a = 3.3: float

Python will change types as needed

- c = a / b is a float



Binary I/O with Python

Python is a little different than Matlab. For one thing, to use functions in a library, you have to use the *import* command. This makes that library available for use in your program.

Two important libraries are the *machine* and *time* library.

- Machine contains routines specific to the microcontroller you're using, such as setting I/O pins to input, output setting the frequency and duty cycle for square waves, etc.
- Time contains wait routines.

Within *machine* is the function *Pin* - which controls whether a pin is input or output. Options are:

```
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 = maching.Pin(8, Pin.IN, Pin.PULL_DOWN)
```

machine.Pin()

- Use the routine *Pin* from library *machine.*
- Allows different libraries to have identical function names
- It does get a little unwieldy, however.

Shortcut:

Pull in routine *Pin*

Outputs:

- Pin 0 is output
- Capable of 12mA

Inputs:

- Pin 6 is input
- Pin 7 with a pull-up resistor
- Pin 8 with a pull-down R

```
from machine import Pin

# line 3
Button = Pin(0, Pin.OUT)

#line 4-6
LED0 = Pin(6, Pin.IN)
LED1 = Pin(7, Pin.IN, Pin.PULL_UP)
LED2 = Pin(8, Pin.IN, Pin.PULL_DOWN)
```

Accessing I/O Pins

Reading:

- Returns 1 or 0

Writing:

- Toggle: switch on/off
- value()
  - 0 = off
  - non-zero = on

```
# 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

- 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:

- Turn on an LED
- For 2 seconds
- Then turn off

Note:

- print() sends a message to the console
- nice for debugging


```
from machine import Pin
from time import sleep

LED = Pin(6, Pin.OUT)

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

Shell

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

Loops

Python supports

- for-loops
- while-loops
- if - else if - else

statements

Syntax is different

- Colon:
  - Start of loop
- Indendation:
  - Part of loop
- Remove indentation
  - End of loop

In Python, carriage returns and indentation have meaning

```
for i in range(0,5):
    print(i, i*i)

x = 3
while(x > 0):
    x -= 1

a = b = 4
if(a > b):
    print('a is greater than b')
elif(a == b):
    print('a is equal to b')
else:
    print('a is less than b')
```

For-Loops:

- Starts with the first number
- Stops when equal or greater than second
  - different than matlab

Example:

- range(0,5)
- Counts from 0 to 4


```
for i in range(0,5):
    print(i, i*i)
```

Shell

```
>>>
    0   0
    1   1
    2   4
    3   9
    4  16
```

If you add a third term, this is the step-size

```
for i in range(0,5, 2):
    print(i, i*i)
```

Shell

```
>>>
    0    0
    2    4
    4   16
```

If you include an array, the for-loop steps through the array

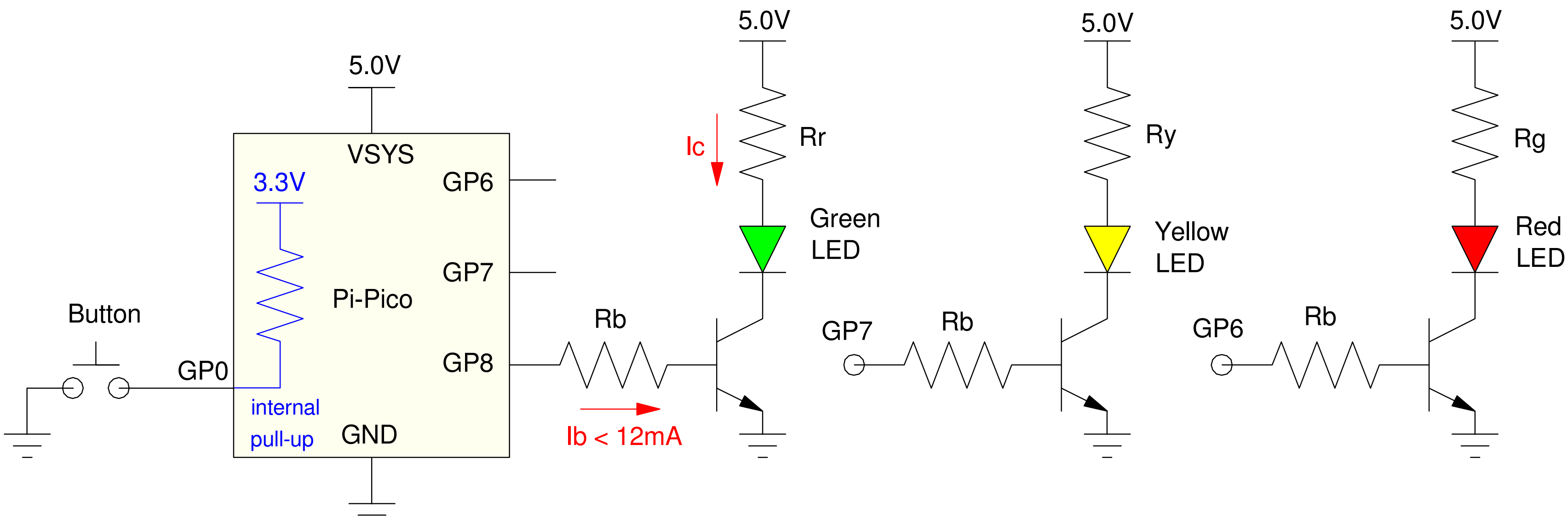
```
for i in [1,3,5,7,11]:
    print(i, i*i)
```

Shell

```
>>>
     1     1
     3     9
     5     25
     7     49
    11     121
```

Example: Counter in Python

As an example, write a Python program that counts how many times a button was pressed. Assume the hardware is:



Counter in Python

- Counts button presses
- Displays on LEDs
- Count in binary

```
from machine import Pin
from time import sleep

Button = Pin(0, Pin.IN, Pin.PULL_UP)
g = Pin(8, Pin.OUT)
y = Pin(7, Pin.OUT)
r = Pin(6, Pin.OUT)

N = 0

while(1):
    while(Button.value() == 0):
        pass
    while(Button.value() == 1):
        pass
    N = (N + 1) % 8
    g.value(N & 0x01)
    y.value(N & 0x02)
    r.value(N & 0x04)
    print(N, r.value(), y.value(), b.value())
```

Shell

```
1   0   0   1
2   0   1   0
3   0   1   1
```

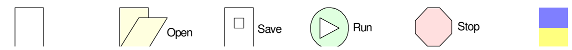
Subroutines in Python

Subroutines are defined by the keyword *def*

- short for *define*.

Example: A routine which

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




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

# Start of main routine
SayHello()
```

shell

```
>>>
hello
```

In this example, note that

- The subroutine is called *SayHello*
- Nothing is passes to this routine as indicated by the ()
- The definition is terminated with a colon (:)
- The code within the subroutine must be indented as per the Python standard

Passing Parameters

You can pass parameters to subroutines.

Example: Pass N

- CountToN(5)
  - Pass the number 5
  - Received as N=5


```
def CountToN(N):
    for i in range(1,N+1):
        print(i)

# Start of main routine
CountToN(5)
```

Thonny Program Window

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

Passing Multiple Parameters

Include 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

Python can

- Return zero numbers, or
- One variable

Example:

- Return one number

Open Save Run Stop

```
# Example of Returning One Number
def Multiply(A, B):
    C = A * B
    return(C0)

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

shell

```
>>>
24

>>> C = Multiply(8,7)
>>> print(C)
56
```

Open Save Run Stop

Example:

- Return multiple numbers
- Returned as an array

```
# 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)
```

When receiving the results

- Can receive as a single array
- Can receive as separate variables

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
```

Going back to the counter program, you could clean up the code with a subroutine:

Open Save Run Stop

```
from machine import Pin
from time import sleep

Button = Pin(0, Pin.IN, Pin.PULL_UP)
g = Pin(8, Pin.OUT)
y = Pin(7, Pin.OUT)
r = Pin(6, Pin.OUT)

def Display(X):
    g.value(X & 0x01)
    y.value(X & 0x02)
    r.value(X & 0x04)

while(1):
    while(Button.value() == 0):
        pass
    while(Button.value() == 1):
        pass
    N = (N + 1) % 8
    Display(N)
    print(N, r.value(), y.value(), b.value())
```

Shell

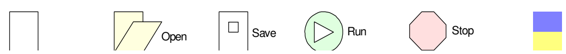
```
1    0    0    1
2    0    1    0
3    0    1    1
```

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)



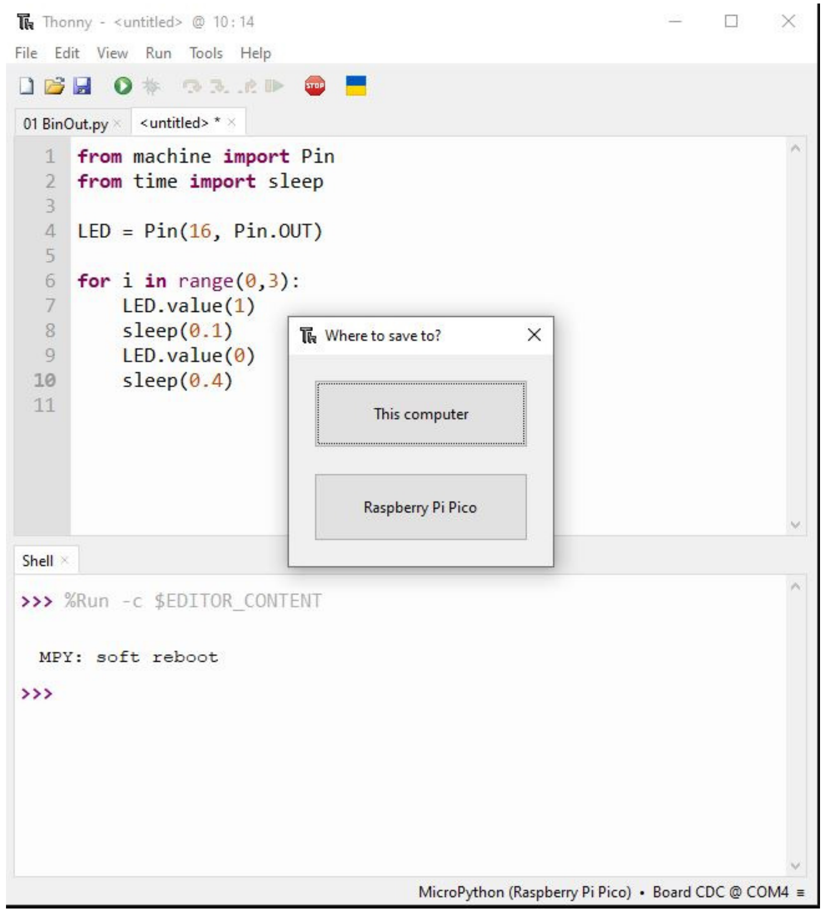

```
from machine import Pin
from time import sleep

LED = Pin(16, Pin.OUT)

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

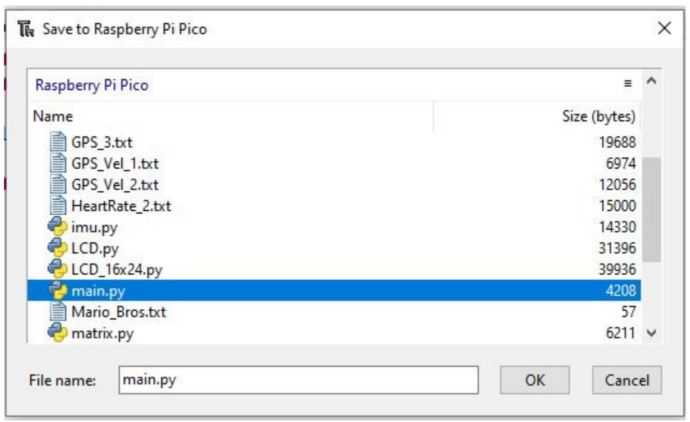
Shell

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.





Appendix: MicroPython Syntax

Assigning values to variables:

```
X = 123           decimal 123
X = 0x123         hex 123
x, y, z = 1, 2, 3
X = [1,2,3,4,5]    matrix or array
X = range(1,6)    same matrix
X = [[1,2],[3,4]]2x2 matrix
```

Operations

```
+           add
-           subtract
*           multiply
/           divide (result is usually a float)
//          divide and round down (result is integer)
%           modulus (remainder)
**          raise to the power

X.append(6)   append 6 to the end of array X
```

Logic Operations

```
&           logical AND (bitwise)
|           logical OR (bitwise)
^           logical XOR (bitwise)
```

```
>>          shift right
<<          shift left
```

comment statement

```
# this is a comment statement
```

Conditionals:

```
X > Y
X < Y
X >= Y
X == Y
X != Y
```

Converting variable types:

```
int(X)     convert to an integer, round down
round(X)   round to nearest integer
float(X)   convert to a floating point number
```

note: Python automatically adjusts variable types - you don't need to declare them like you do in C. For example:

```
>>> X = 3   X is automatically treated like an integer
>>> Y = 4   Y is automatically treated like an integer
```

```
>>> Z = X/Y Z becomes a float  (0.75)
>>> Z = X//Y     Z is an integer (0)
```

print() Information can be sent to the shell window using a *print()* statement

```
>>> print('Hello World')
Hello World

>>> X = 2**0.5
>>> print('X = ',X)
X =  1.414214
```

X = input() Information can be passed to your program using the *input()* statement. For example, prompt the user to input a number for X:

```
>>> X = input('Type in a number')
```

This will result in X being a string (typing in *Hello World* is valid). If you want to receive the input as a number, convert the result as:

```
>>> X = int( input('Type in a number') )
>>> X = float( input('Type in a number') )
```

When writing to the shell, numbers can be formatted if desired. Examples follow:

```
>>> msg = '27 in binary = {:b}'.format(27)
>>> msg
'27 in binary = 11011'

>>> msg = '27 in binary = {:b}'.format(27)
>>> msg
'27 in binary = 11011'
>>> msg = '27 in hex = {:X}'.format(27)
>>> msg
'27 in hex = 1B'

>>> msg = '0x2134 in decimal = {:d}'.format(0x1234)
>>> msg
'0x2134 in decimal = 4660'

>>> msg = '123.4567 rounded to 2 decimal = {:.2f}'.format(123.4567)
>>> msg
'123.4567 rounded to 2 decimal = 123.46'

>>> msg = '123.4567 rounded to 2 decimal = {:.2e}'.format(123.4567)
>>> msg
'123.4567 rounded to 2 decimal = 1.23e+02'

>>> msg = '79/255 = {:.2%}'.format(79/255)
```

```
>>> msg
'79/255 = 30.98%'
```