MicroPython Introduction (cont.)

MCHE 201 – Spring 2019

Dr. Joshua Vaughan

Rougeou 225

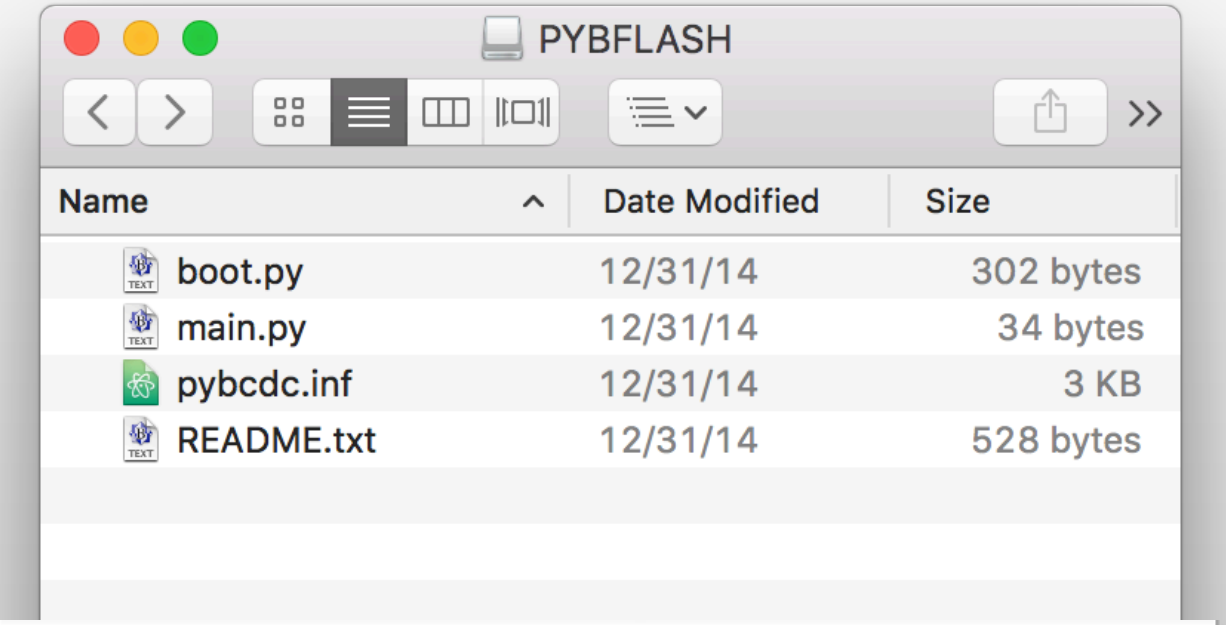
joshua.vaughan@louisiana.edu

@Doc_Vaughan

MicroPython File Review

- `boot.py`
  - Runs every time the pyboard boots
  - Use for setup and configuration

- `main.py`
  - Executed immediately after `boot.py`
  - Use for your “main” code
  - Can reference other files



Name	Date Modified	Size
boot.py	12/31/14	302 bytes
main.py	12/31/14	34 bytes
pybcdc.inf	12/31/14	3 KB
README.txt	12/31/14	528 bytes

`boot.py` and `main.py` are at the "root" of the PYBFLASH drive (*i.e.* They are not in a folder.)

Review of Using `imports`

Just prepend the variable or function you want to use with the “name” that you imported

```
# Import the pyboard functions
import pyb

# To use a function from pyb, put pyb.
# in front of the function name.
RED_LED = pyb.LED(1)
```

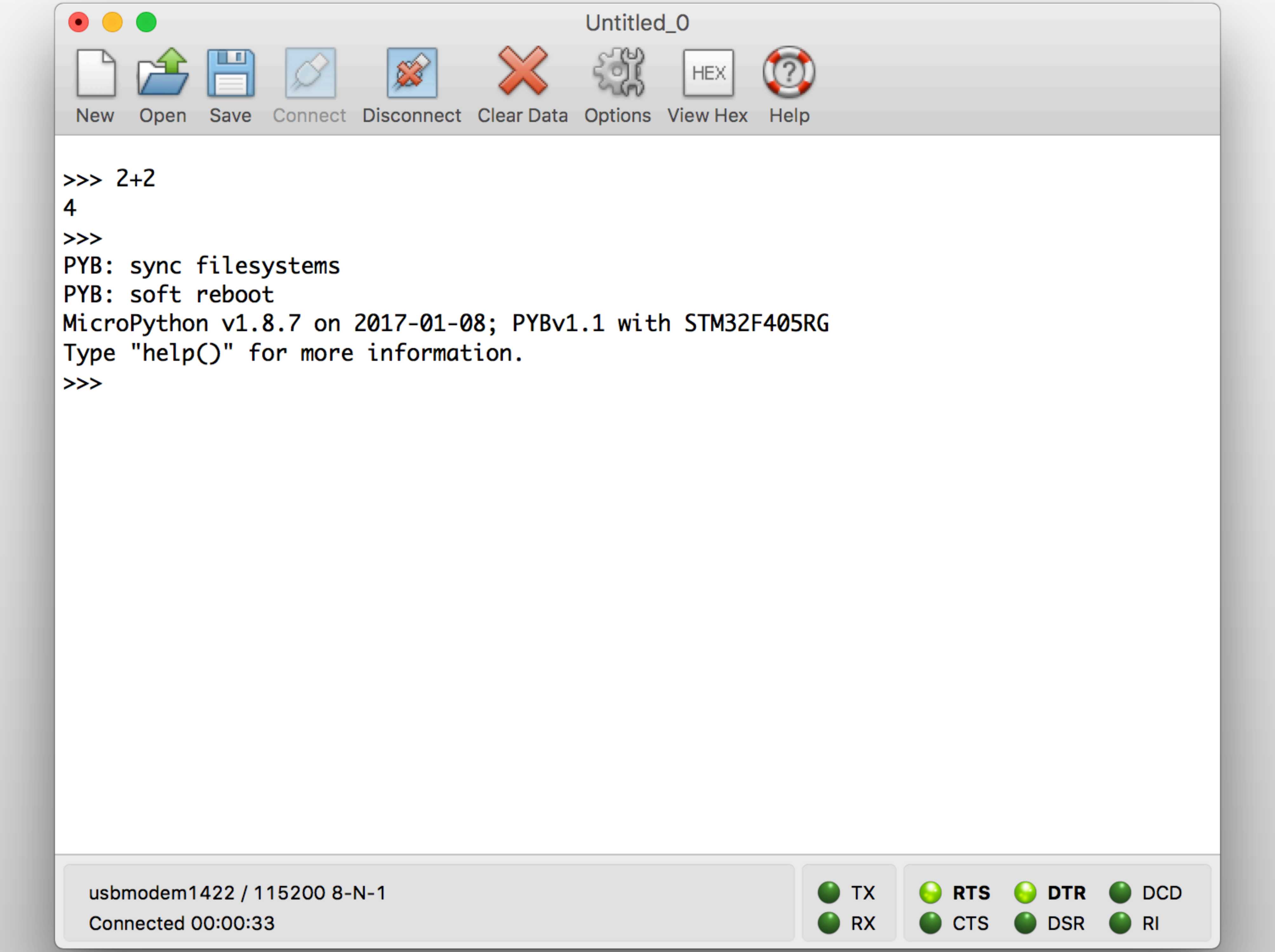
Review of Using `imports`

Just prepend the variable or function you want to use with the “name” that you imported

```
# Import time module
import time

# sleep for 1 second
time.sleep(1)

# sleep for 500 milliseconds
time.sleep_ms(500)

# sleep for 10 microseconds
time.sleep_us(10)
```

REPL Special Command Review

- `Control-d` will perform a soft reboot



```
>>> 2+2
4
>>>
PYB: sync filesystems
PYB: soft reboot
MicroPython v1.8.7 on 2017-01-08; PYBv1.1 with STM32F405RG
Type "help()" for more information.
>>>
```

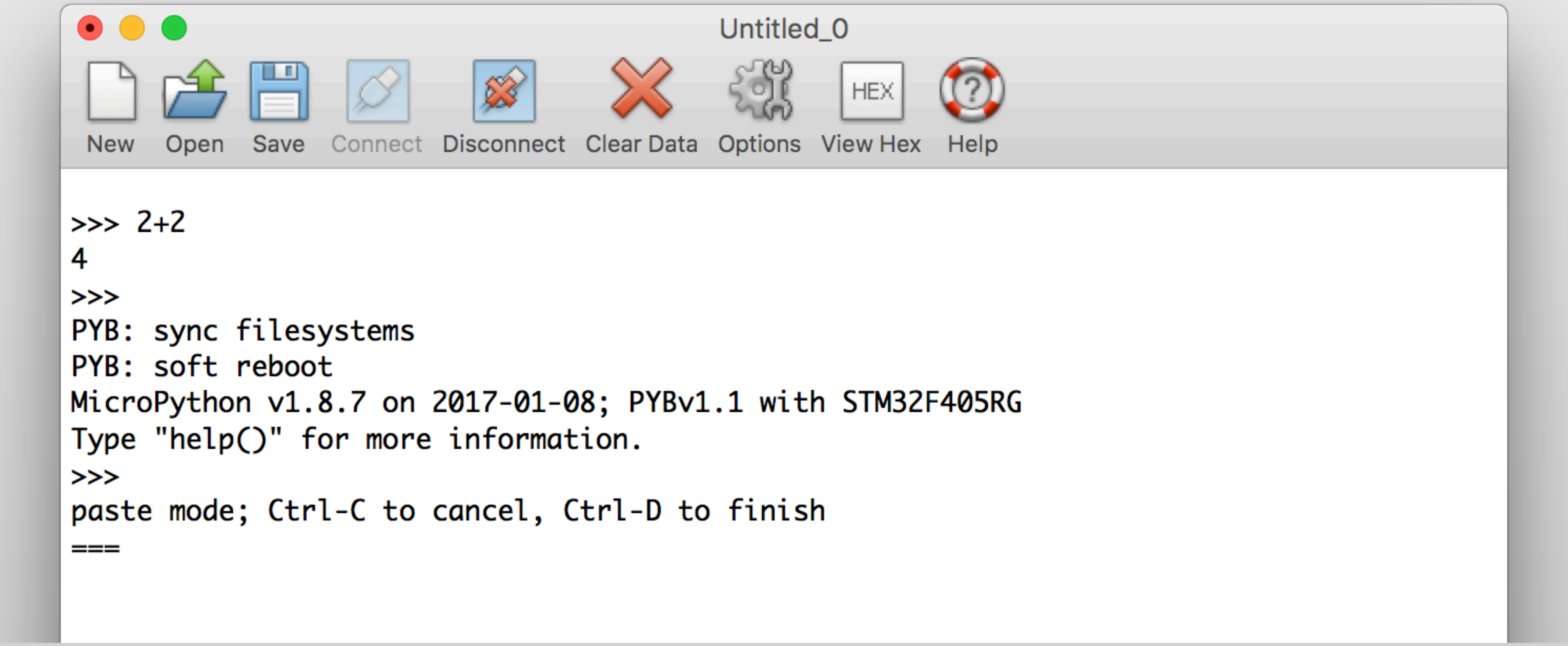
usbmodem1422 / 115200 8-N-1
Connected 00:00:33

REPL Special Command Review

- `Control-d` will perform a soft reboot
- `Control-c` will kill any running script

REPL Special Command Review

- `Control-d` will perform a soft reboot
- `Control-c` will kill any running script
- `Control-e` will enter paste mode
  - Paste as usual
  - Use `Control-d` to exit paste mode



```
>>> 2+2
4
>>>
PYB: sync filesystems
PYB: soft reboot
MicroPython v1.8.7 on 2017-01-08; PYBv1.1 with STM32F405RG
Type "help()" for more information.
>>>
paste mode; Ctrl-C to cancel, Ctrl-D to finish
===
```

Where can I find help?

- Full – `http://docs.micropython.org/en/latest/pyboard/`
- Quick Ref – `http://docs.micropython.org/en/latest/pyboard/pyboard/quickref.html`
- REPL specific – `http://docs.micropython.org/en/latest/pyboard/reference/repl.html`
- More links coming to class webpage
- If you don’t remember the syntax, look it up

Recommended Workflow

- Connect the board to your computer and start the REPL in CoolTerm
- Work on scripts (mostly `main.py` in MCHE201) in a local folder with Atom
- Drag edited versions to `PYBFLASH`
- `Control-d` in the REPL to perform a soft reboot and run edited `main.py`

In-class Exercise 1

- Print the odd numbers between 1 and 27
- *Hint:* A for loop would be a good way to do this.

There are many ways to do this. A script with some is at:

`https://github.com/DocVaughan/MCHE201---Intro-to-Eng-Design/tree/Spring-2019/MicroPython/MCHE201%20-%20In-class%20Exercise%201%20-%2003:07:19`

Exercise 1 – Solution 1

```
# ----- Method 1 -----
# In this first method, we create a range
# of 14 numbers, then simply do the math
# to convert the list to odd numbers

for counter in range(14):
    oddNumber = 2 * counter + 1

    print(oddNumber)
```

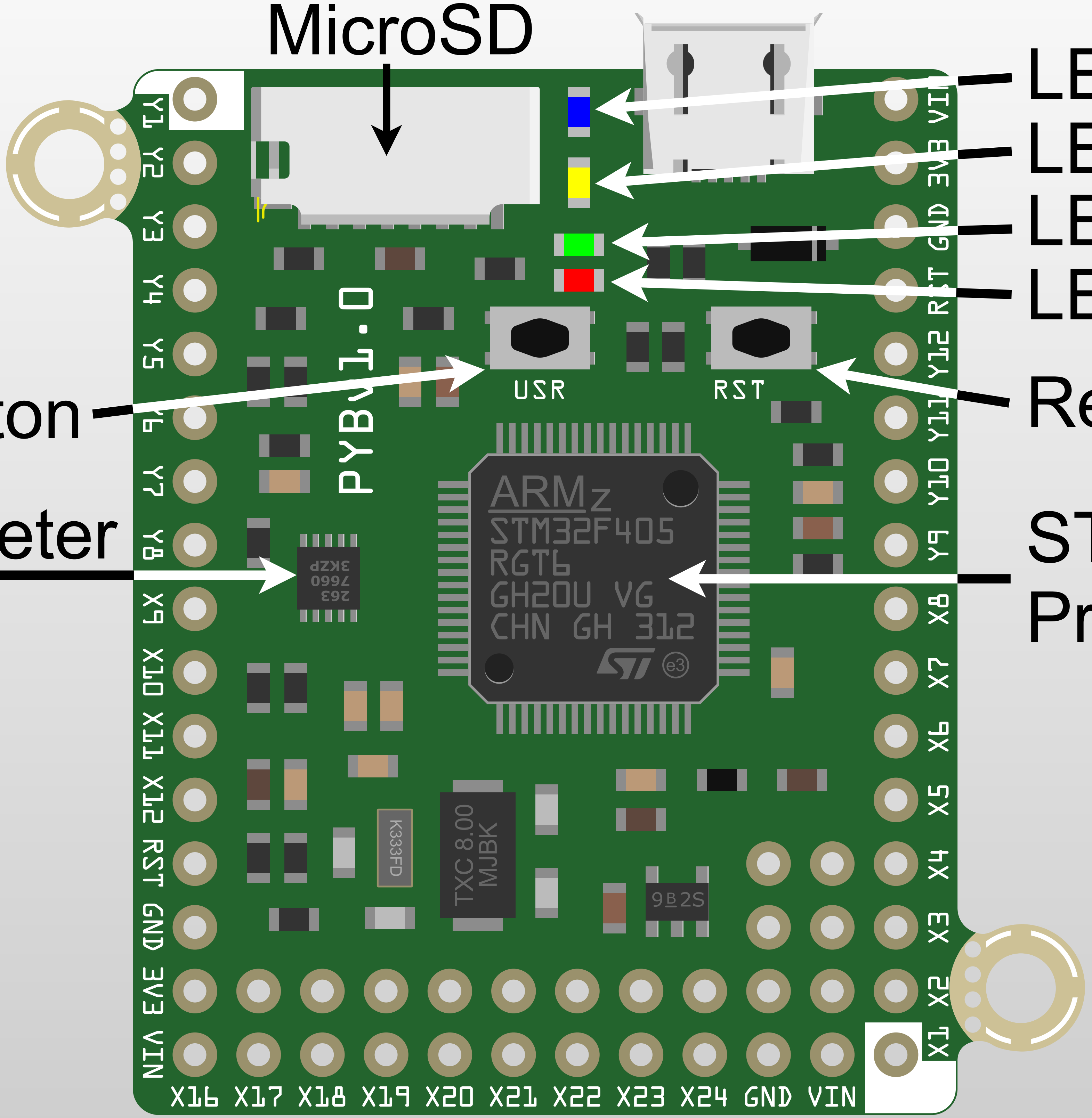
Exercise 1 – Solution 2

```python
# ----- Method 2 -----
# Here, we'll use a for loop with a properly
# defined range. Here, we use the
# extra terms available in the range function.
# The order is
#    range(start, stop, increment)
# We have to extend the range past 27 because
# the last number listed in not included in the
# range.

for counter in range(1, 29, 2):
    print(counter)
```

Exercise 1 – Solution 4

```python
# ----- Method 4 -----
# Here, we'll use a while loop and increment the
# counter ourselves. We'll increment it by 2
# each time to only get the odd numbers. We
# could also increment by 1 and either do math
# on counter to create an odd number, as we did
# in Method 1, or use one an if statement, like
# we did in Method 3

counter = 1

while counter <= 27:
    print(counter)
    counter = counter + 2
```

The Onboard Hardware


PYBv1.0
USR
RST
ARM
STM32F405
RGT6
GH20U VG
CHN GH 312
TXC 8.00
MJBK
K333FD
9B2S
Y1 Y2 Y3 Y4 Y5 Y6 Y7 Y8 X9 X10 X11 X12 RST GND 3V3 VIN
VIN 3V3 GND RST Y12 Y11 Y10 Y9 X8 X7 X6 X5 X4 X3 X2 X1
X16 X17 X18 X19 X20 X21 X22 X23 X24 GND VIN

MicroSD

LED 4

LED 3

LED 2

LED 1

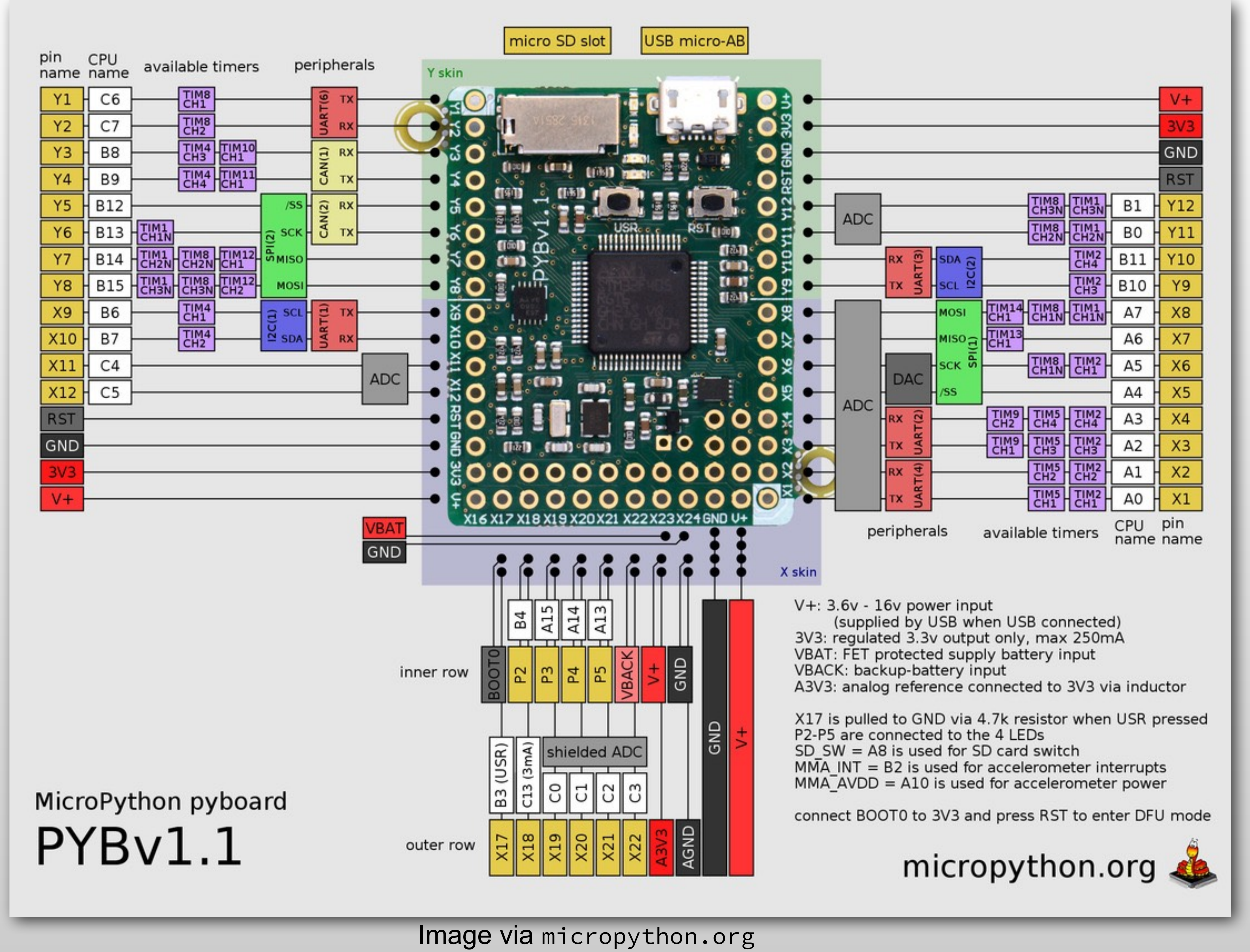
User Button

Reset Button

Accelerometer

STM32
Processor

The pyboard



pin name	CPU name	available timers	peripherals
Y1	C6	TIM8 CH1	UART(6) TX
Y2	C7	TIM8 CH2	UART(6) RX
Y3	B8	TIM4 CH3 TIM10 CH1	CAN(1) RX
Y4	B9	TIM4 CH4 TIM11 CH1	CAN(1) TX
Y5	B12		SPI(2) /SS CAN(2) RX
Y6	B13	TIM1 CH1N	SPI(2) SCK CAN(2) TX
Y7	B14	TIM1 CH2N TIM8 CH2N TIM12 CH1	SPI(2) MISO
Y8	B15	TIM1 CH3N TIM8 CH3N TIM12 CH2	SPI(2) MOSI
X9	B6	TIM4 CH1	I2C(1) SCL UART(1) TX
X10	B7	TIM4 CH2	I2C(1) SDA UART(1) RX
X11	C4		ADC
X12	C5		ADC
RST			
GND			
3V3			
V+			

peripherals	available timers	CPU name	pin name
			V+
			3V3
			GND
			RST
ADC	TIM8 CH3N TIM1 CH3N	B1	Y12
ADC	TIM8 CH2N TIM1 CH2N	B0	Y11
UART(3) RX I2C(2) SDA	TIM2 CH4	B11	Y10
UART(3) TX I2C(2) SCL	TIM2 CH3	B10	Y9
ADC SPI(1) MOSI	TIM14 CH1 TIM8 CH1N TIM1 CH1N	A7	X8
ADC SPI(1) MISO	TIM13 CH1	A6	X7
ADC DAC SPI(1) SCK	TIM8 CH1N TIM2 CH1	A5	X6
ADC DAC SPI(1) /SS		A4	X5
ADC UART(2) RX	TIM9 CH2 TIM5 CH4 TIM2 CH4	A3	X4
ADC UART(2) TX	TIM9 CH1 TIM5 CH3 TIM2 CH3	A2	X3
ADC UART(4) RX	TIM5 CH2 TIM2 CH2	A1	X2
ADC UART(4) TX	TIM5 CH1 TIM2 CH1	A0	X1

		B4	A15	A14	A13					
inner row	BOOT0	P2	P3	P4	P5	VBACK	V+	GND	GND	V+
	B3 (USR)	C13 (3mA)	C0	C1	C2	C3				
outer row	X17	X18	X19	X20	X21	X22	A3V3	AGND		

V+: 3.6v - 16v power input
(supplied by USB when USB connected)
3V3: regulated 3.3v output only, max 250mA
VBAT: FET protected supply battery input
VBACK: backup-battery input
A3V3: analog reference connected to 3V3 via inductor

X17 is pulled to GND via 4.7k resistor when USR pressed
P2-P5 are connected to the 4 LEDs
SD_SW = A8 is used for SD card switch
MMA_INT = B2 is used for accelerometer interrupts
MMA_AVDD = A10 is used for accelerometer power

connect BOOT0 to 3V3 and press RST to enter DFU mode

MicroPython pyboard
PYBv1.1

micropython.org

Image via micropython.org

Controlling the Onboard LEDs

- Numbered 1 – 4
- Follow same pattern as earlier RED_LED example

```
import pyb # import the pyboard module

# Assign the names to the onboard LEDs
RED_LED = pyb.LED(1)
GREEN_LED = pyb.LED(2)
YELLOW_LED = pyb.LED(3)
BLUE_LED = pyb.LED(4)
```

Onboard LED methods

- For all 4 onboard LEDs
  - `on()` – turn the LED on
  - `off()` – turn the LED off
  - `toggle()` – toggle the state of the LED

- For the third (yellow) and fourth (blue) LEDs
  - `intensity()` – set or get the brightness of the LED
    - ✦ If a number is inside, set to that value (between 0-255)
    - ✦ If no argument, get the current intensity

LED Intensity Example

```
# Assign the 4th LED to variable BLUE_LED
BLUE_LED = pyb.LED(4)

BLUE_LED.on()           # Turn fully on
time.sleep(1)           # Sleep 1 second

BLUE_LED.intensity(128) # Set to ~1/2 intensity
time.sleep(1)           # Sleep 1 second

BLUE_LED.intensity(64)  # Set to ~1/4 intensity
time.sleep(1)           # Sleep 1 second

BLUE_LED.intensity(1)   # Set to min. intensity
time.sleep(1)           # Sleep 1 second

BLUE_LED.off()          # Turn it off
```

LED Intensity Example

```
# Assign the 4th LED to variable BLUE_LED
BLUE_LED = pyb.LED(4)

BLUE_LED.on()                # Turn fully on
time.sl

BLUE_LED.intensity(128)      # Set to ~1/2 intensity
time.sleep(1)                # Sleep 1 second

BLUE_LED.intensity(64)       # Set to ~1/4 intensity
time.sleep(1)                # Sleep 1 second

BLUE_LED.intensity(1)        # Set to min. intensity
time.sleep(1)                # Sleep 1 second

BLUE_LED.off()               # Turn it off
```

How could we improve this?

LED Intensity Example – Improved

```
# Assign the 4th LED to variable BLUE_LED
BLUE_LED = pyb.LED(4)

print("Turning on LED")
BLUE_LED.on()          # Turn on at full brightness
time.sleep(1)          # Sleep 1 second

print("Setting to 1/2 intensity")
BLUE_LED.intensity(128)  # Set to ~1/2 intensity
time.sleep(1)            # Sleep 1 second

print("Setting to 1/4 intensity")
BLUE_LED.intensity(64) # Set to ~1/4 intensity
time.sleep(1)          # Sleep 1 second

print("Setting to min. intensity")
BLUE_LED.intensity(1)  # Set to minimum intensity
time.sleep(1)          # Sleep 1 second

print("Turning off LED")
BLUE_LED.off()         # Turn it off
```

In-class Exercise 2

- Print the odd numbers between 1 and 27
- When the number is 13, print “Counter = 13... Bad Luck!!!” and turn on the red LED
- *Hint:* Modify/extend one of the methods used to solve Exercise 1.

There are many ways to do this. A script with some is at:

`https://github.com/DocVaughan/MCHE201---Intro-to-Eng-Design/tree/Spring-2019/MicroPython/MCHE201%20-%20In-class%20Exercise%202%20-%2003:07:19`

Exercise 2 – Solution 1

```
import pyb  # import the pyboard module

# Assign the 1st LED to variable RED_LED
RED_LED = pyb.LED(1)

# ----- Method 1 -----
for counter in range(14):
    # Same math as Exercise 1
    oddNumber = 2 * counter + 1

    if oddNumber == 13:
        print("Counter = 13... Bad Luck!!!")
        RED_LED.on()     # Turn the RED_LED on
    else:
        print(oddNumber)
        RED_LED.off()    # Turn the RED_LED off
```

Exercise 2 – Solution 2

```
import pyb  # import the pyboard module

# Assign the 1st LED to variable RED_LED
RED_LED = pyb.LED(1)

# ----- Method 2 -----
for counter in range(1, 29, 2):

    if counter == 13:
        print("Counter = 13... Bad Luck!!!")
        RED_LED.on()     # Turn the RED_LED on
    else:
        print(counter)
        RED_LED.off()    # Turn the RED_LED off
```

Reading the Onboard Button

- It’s a “switch” in MicroPython
- We can:
  - Get its current state manually and/or
  - Set up code to run automatically any time it’s pressed
- For both, we need to set up a “switch” object

```
import pyb # import the pyboard module

# Assign the Switch object for
# the onboard button to variable button
button = pyb.Switch()
```

Manually Reading the Button

```
import pyb  # import the pyboard module

# Assign the Switch object for
# the onboard button to variable button
button = pyb.Switch()

# call the variable assigned like it’s a
# function. It will return True, if pressed.
button()
```

Reading the Button Indefinitely

```
import pyb  # import the pyboard module
import time # import the time module

# Assign the Switch object for the onboard button
# to variable button
button = pyb.Switch()

# The condition for this while is always true, so
# it runs forever
while (True):
  # button() is True if the button is pressed
  if (button()):
    print("Button Pressed!")

  time.sleep_ms(100) # Sleep 100ms between reading
```

In-class Exercise 3

- Turn on the green LED when the button is pressed
- Turn on the red LED when it is *not* pressed