MicroPython
Introduction
MCHE 201 – Spring 2019

Dr. Joshua Vaughan
Rougeou 225
joshua.vaughan@louisiana.edu
@Doc_Vaughan
The pyboard

- LED 4
- LED 3
- LED 2
- LED 1
- Reset Button
- STM32 Processor
- MicroSD
- User Button
- Accelerometer
- STM32 Processor
- LED 4
- LED 3
- LED 2
- LED 1
- Reset Button
- STM32 Processor
- MicroSD
- User Button
- Accelerometer
Why Python?

Growth of major programming languages
Based on Stack Overflow question views in World Bank high-income countries

Figure via: https://stackoverflow.blog/2017/09/06/incredible-growth-python/
Why not Arduino?

- Python is a general-purpose language
  - Instagram, Google, etc. use it extensively
  - Many robotics tools are built around it
    - http://lorenabarba.com/blog/why-i-push-for-python/

- The pyboard is *significantly* more powerful than equivalently-priced Arduino boards
System Setup

• You’ll need a plain-text editor
  - *Many* options that programmers *really* argue about

• Bookmark the documentation and quick reference
  - If you don’t remember the syntax, *look it up*
Connecting to the pyboard

• Just plug in Micro-USB cable

• The board will show up as a USB disk with files:
  - boot.py
  - main.py
  - README.txt
  - pybcdc.inf
STOP – Before anything else

Save those default files to a safe place on your computer!
WARNING!!!

• Do NOT edit the files directly on the PYBFLASH drive

• Instead:
  - Work on a version on your computer
  - Then, copy that file to the pyboard

• Be sure to eject/unmount before unplugging

The pyboard’s flash memory can get corrupted much easier than a normal “thumb drive.”
On Windows...

• You may be asked to set up the device when you plug it in… *cancel* that prompt.

• Try to connect to the board first, you likely will *not* need to install the driver.

• If you do need to install a driver
  - The pybcdc.inf file from the disk is the driver
Getting to the REPL

• We’ll talk to the board over serial, often connecting to the Read, Evaluate, Print, Loop (REPL) prompt

• Like the text editor, there are many options

  - On macOS:
    ✦ CoolTerm – http://freeware.the-meiers.org
    ✦ Using screen from the Terminal app
    ✦ goSerial – http://www.furrysoft.de/?page=goserial
    ✦ Serial Tools – http://www.w7ay.net/site/Applications/Serial%20Tools/index.html

  - On Windows:
    ✦ CoolTerm – http://freeware.the-meiers.org
    ✦ HyperTerminal is still installed by default on some dist.
Code Sharing – GitHub.com

https://github.com/DocVaughan/MCHE201---Intro-to-Eng-Design
Code Sharing – GitHub.com

https://github.com/DocVaughan/MCHE201---Intro-to-Eng-Design
Comments

- Completely ignored by the Python interpreter

- Comments allow you to explain your code inline for:
  - your co-workers/teammates
  - you, in the future

- To comment a single line, use `#` before your comment

- To create a block comment, begin with `"""` and end with `"""`
Comments

""" This is a block comment. It will continue across multiple lines, until it is closed with the proper characters """

# This is a single-line comment

x = 4 # Comments can go here too
Block Comments

• Block comments are also a good way to begin any file you write.

• It's good practice to include:
  - The filename
  - The purpose of the code
  - Any external requirements (other files or hardware needed to make this one work)
  - What inputs are needed, if any
  - What the output is, if any
  - The version number, recent modifications, and your contact info
Block Comments – Boilerplate

"""----------------------------------------------------------
filename.py
Some description of functionality
Optional links to relevant documentation
Created: mm/dd/yy - Name - email@louisiana.edu

Modified:
  * mm/dd/yy - Name (email if not same person as above)
    - major change 1
    - major change 2
  * mm/dd/yy - Name (email if not same person as above)
    - major change 1
"""
# ----------------------------------------------------------
# filename.py
#
# Some description of functionality
#
# Optional Link to relevant documentation
#
# Created: mm/dd/yy - Name - email@louisiana.edu
#
# Modified:
#  * mm/dd/yy - Name (email if not same person as above)
#    - major change 1
#    - major change 2
#  * mm/dd/yy - Name (email if not same person as above)
#    - major change 1
# ----------------------------------------------------------
# main.py
#
# This script will control a single DC motor using a Texas Instruments DRV8871 motor driver. It should work with all DRV8871 driver breakouts, but has only been tested with the Adafruit one:
#   https://www.adafruit.com/product/3190
#
# Motor driver spec sheet
#
# Adafruit Overview of the board:
#   https://learn.adafruit.com/adafruit-drv8871-brushed-dc-motor-driver-breakout
#
# Created: 11/06/17
#   - Joshua Vaughan
#   - joshua.vaughan@louisiana.edu
#   - http://www.ucs.louisiana.edu/~jev9637
#
# Modified:
#   *
#
# TODO:
#   *


Literate Programming

• Write out what you want your code to do in plain English (or your preferred language)… Be explicit about every step

• Translate this into comments in your code file

• Then, write the code to implement the functionality

**Key Point:** If you can't explain what you want the code to do in plain English, writing code to do that will be difficult.
Variables in Python

• Unlike Arduino (or other C-based languages), we don’t need to specify the variable type

• Python is a *dynamically-typed* language
  - It will figure out what type of variable you need
  - That type can/will change if you reassign the variable to a different type

*TIP*: Give your variables meaningful names. A few extra keystrokes are worth the improved understanding and easier debugging.
# Booleans are True or False.
binaryConditionCheck = False
youCantHandleThe = True

# Integers are, well, integers
integerVariable = -1
motorSpeed = 75

# Floats are decimal numbers
floatVariable = 1.0
preciseMotorSpeed = 75.275

# Strings hold text, put between "-"
myString = "some text"
# We can assign multiple variables at the same time

# Note: Be careful with this, only group variables that make sense to group logically.

small, medium, large = 1, 3, 9
IP_ADDRESS, PORT = "192.168.0.100", 2390
Variable Naming Conventions

• Give your variables meaningful names
  - armLength = 15 is much clearer than l = 15
  - delay_time = 0.25 is much clearer than t = 0.25

• Use a consistent variable style
  - camel case – armLength
  - Underscores for spaces – delay_time
  - All caps for constants – LED_PIN

Pick one of these and stick to it
Variable Scope

• Scope – essentially what functions are able to read/write to a particular variable

• Variables defined:
  - Outside of all functions have global scope
    ✦ Can be read anywhere
    ✦ Need some special syntax to write to them
  - Inside a function are accessible inside that function

• Limit scope to as small as possible
def myMultiplyFunction(x, y):
    result = x * y
    return result

- In Python, whitespace matters
- *Note:* All of these have more formal names.

**TIP:** Give your functions meaningful names. A few extra keystrokes are worth the improved understanding and easier debugging.
To Use That Function

```python
def myMultiplyFunction(x, y):
    result = x * y
    return result

# Assign values to a and b
a = 2
b = 3

# Call the function, and store the result in c
c = myMultiplyFunction(a, b) # c=6

# This works fine with other types too
a, b = 1.2, 3.75
c = myMultiplyFunction(a, b) # c=4.5
```
Use Functions!!!

• Aim for each function having a single function

• This makes:
  - execution more predictable and easier to debug
  - the code more-easily reusable
    ✦ Reuse limits likelihood of typos and other bugs
    ✦ Makes code more readable
    ✦ Makes program logic easier to follow

*TIP*: Give your functions meaningful names. A few extra keystrokes are worth the improved understanding and easier debugging.
Example

wait_for_start_button()

pyb.delay_ms(500)  # pause 500ms after start button

drive_forward(4)   # drive forward 4 seconds

rotate_arm(75)     # rotate the arm 75 deg

pyb.delay_ms(1000) # Pause for 1000ms (1s)

rotate_arm(0)      # rotate the arm back to 0

drive_backward(2)  # drive backward 2 seconds
How do I debug my code?

• The computer will only do exactly what you tell it. Nothing more. Nothing less.

• Don’t assume anything!… the computer is dumb.
  - Work line-by-line “What happens on this line?”
  - Output values in runtime via `print` statements
Using the REPL

- Allows the pyboard to communicate with the computer during runtime

- Can be used for:
  - Prototyping
  - Debugging
  - Execution monitoring
Clarity in the print Statements

• We can format the numbers/items that we print out.
• A great overview: https://pyformat.info
• Syntax is:

```
print("String {formatting spec}".format(variable))
```
# Print an integer
print("Integer {:d}.".format(42))

# Print an integer and always include +/- sign
print("Integer {:+d}.".format(42))

# Print an integer and always include at least 4 "places"
print("Integer {:4d}.".format(42))

# Print an integer, always include at least 4 "places," and pad with zeros
print("Integer {:04d}.".format(42))
# Print a float
print("Pi is {:f}.".format(3.141592))

# Print a float with 4 decimal places
print("Pi is {:4f}.".format(3.141592))

# Print a float and always include at least 9 "places" with 2 decimal places
print("Pi is {:9.2f}.".format(3.141592))

# Print a float and always include at least 9 "places" and pad with zeros
print("Pi is {:09.2f}.".format(3.141592))
Special Characters to Know

- \n = new line
- \r = carriage return
- \t = tab

```python
# Define pi
pi = 3.141592

print("Pi is {:.4f}.\n2pi is {:.4f}".format(pi, 2*pi))

print("Pi is {:.4f}.\tt2pi is {:.4f}".format(pi, 2*pi))
```
Control Structures

• Numerous ways to control program flow

• Ways to conditionally execute
  - If… then
  - For loops
  - While loops
Comparison Operators

# ----- Comparison syntax ---------------------------------
# These evaluate to True (1) or False (0)

x == y  # True if x is equal to y, False otherwise
x != y  # True if x is not equal to y, False otherwise
x < y   # True if x is less than y, False otherwise
x > y   # True if x is greater than y, False otherwise
x <= y  # True if x is less than or equal to y, False otherwise
x >= y  # True if x is greater than or equal to y, False otherwise
If... then Example

# ----- if... elif... else example ------------------------
# Note: this assumes all variables have been defined,
# etc.

if (counter < 10):
    # Code indented here will run if counter is less than 10

elif (counter >= 20):
    # Code indented here will run if counter is greater than or equal to 20

else:
    # Code indented here will only run if both counter is neither less than 10 or greater than or equal to 20
If... then Example 2

```python
a = 2  # Define the value of a

if (a > 5):
    print("Tell me something, girl")

elif (a == 2):
    print("Are you happy in this modern world")

else:
    print("Or do you need more?")
```

Here, a is equal to 2, so the `elif` condition is True. The code indented under it is run, meaning *Are you happy in this modern world* would be printed.
If... then Example 3

\[
a = 2 \quad \# \text{Define the value of } a \\
b = 3 \quad \# \text{Define the value of } b
\]

```python
if (a + b > 5):
    print("Kiki, do you love me?")

elif (b - a == 2):
    print("Are you riding?")

else:
    print("Say you'll never ever leave...")
```

Neither the if or the elif condition is True. So, the code in else is run, meaning Say you'll never ever leave... would be printed.
If... then Example 4

sensedStartSignal = True  # Start was sensed

if (sensedStartSignal):
    print("Sensed start signal. Starting robot.")
    # Code to run once the start signal was sensed

else:
    print("Checking start signal...")
    # Code to check the start signal

The **if** is True. So, the code in **if** is run, meaning **Sensed start signal. Starting robot.** would be printed and other code in that indented block would run.
Basic For Loops

# ----- for loop syntax ----------------------------------------

for counter in sequence:
    # do something
    # Everything indented here is run during each
    # loop until the sequence is finished
Basic For Loops

# ----- for loop syntax -----------------------------------------------

for counter in sequence:
    # do something
    # Everything indented here is run during each
    # loop until the sequence is finished

Variable that's incremented

What to loop over... a few options for what

# ----- for loop example -----------------------------------------------

for counter in range(10):
    # do something
    # This would run 10 times
    # The values of counter would be 0, 1, 2, ..., 9
list_of_pies = ['apple', 'cherry', 'pumpkin']

for pie in list_of_pies:
    print(f'I think {pie} pies are delicious!')

Prints out to the REPL:
I think apple pies are delicious!
I think cherry pies are delicious!
I think pumpkin pies are delicious!
For Loop Example

```python
list_of_pies = ['apple', 'cherry', 'pumpkin']

for index, pie in enumerate(list_of_pies):
    print("The number {:d} pie in the list is {}.").format(index, pie))
```

Prints out to the REPL:
The number 0 pie in the list is apple.
The number 1 pie in the list is cherry.
The number 2 pie in the list is pumpkin.
While Loops

// ----- while loop syntax -------------------------

while (condition == True):
    # If the condition is True, run the code here.
    # Once the code in the indented block is finished, check the condition and repeat.
    # If the condition is not True at the first check above, this will never be run.

The condition is tested at the beginning of each iteration.
# ----- while loop example ---------------------

index = 0

while (index < 10):
    print("Index = {:d}".format(index))
    index = index + 2

Prints
Index = 0
Index = 2
Index = 4
Index = 6
Index = 8
```python
# ----- while loop example ---------------------
index = 0

while (index < 10):
    if (index == 3):
        print("Index = {}".format(index))

    index = index + 1

Prints to the Serial Monitor
Index = 3
While Loop Example 3

```python
# ----- while loop example ----------------------------------
keepRunning = True
index = 0

while(keepRunning):
    print("Running.")

    if (index >= 10):
        keepRunning = False

    pyb.delay(100) # sleep 100ms

    index = index + 1

print("Stopped.")
```

Loops 10 times, printing "Running" and delaying 100ms each time. Then, prints "Stopped."
For next Thursday...

• **BEFORE** next week:
  - Install the driver, if necessary for Windows.
  - Install Visual Studio Code (or other text editor) on your computer.
  - Install CoolTerm on your computer.
  - Look through these notes. These are the foundation for all the programming we’ll do.

• Bring laptop and kit to class