Lab #1 – Introduction to the PyBoard

How to work as a group

This lab involves learning about simple programming on the PyBoard, thinking about some questions, and writing a report. You only need to turn in one report per group.

The goal of having only one written report per group is to save you time writing and save me time reading. However, my expectation is that everyone in the group does the work together and discusses the questions together. The goal is *not* to have only one person learn the material 😊.

Overview:

In this lab, we'll

- get our first exposure to the PyBoard (a small computer that natively runs Python)
- learn how to use it as a simple calculator and to run small programs
- learn some basic Python skills: variables, expressions and printing

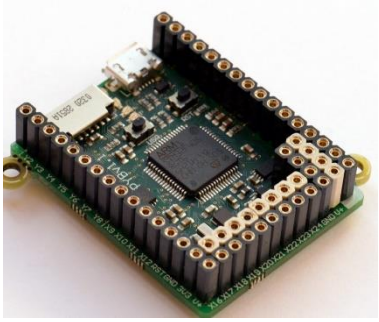
***Background*:**

- The book *Think Python*, Chapters 1 and 2 (if you like)
- The Python videos on Variables and Expressions (again, if you like)

You can find these on the course web page

What is a PyBoard?

A PyBoard is a very small computer. Here's a picture of it:



You'll immediately notice a few things:

- It has lots of things that look like electronic parts. That's probably good 😊
- It has no keyboard or mouse
- It has no screen

Without some way get data into the PyBoard (like a touch screen, mouse or keyboard) or any way to get data out of it, how can you practically use the PyBoard? In several ways:

- Even without a screen, the PyBoard has four LEDs that we'll play with today.
- We'll take advantage of a "host computer" – the PC on your lab bench – to help talk to the PyBoard today.
- Starting in Lab #3, we'll input data to the PyBoard by physically wiring sensors to input pins on the PyBoard.

- The PyBoard's output pins can drive headphones or servo motors – we'll do both of those in future labs.

Logging into your a host PC

The PCs in Halligan are on the Electrical Engineering/Computer Science network, which is distinct from the Tufts network. Thus, your usual Tufts UTLN and password will *not* allow you to log into any of these machines. However, if you are registered for this class, you should already have an account. If not, you can get one by e-mailing staff@eecs.tufts.edu or by walking in at Halligan room 231C (just down the hall from our lab).

If you haven't already, everyone should try now. Then one person in your group should stay logged in so that you can use the PC in this lab!

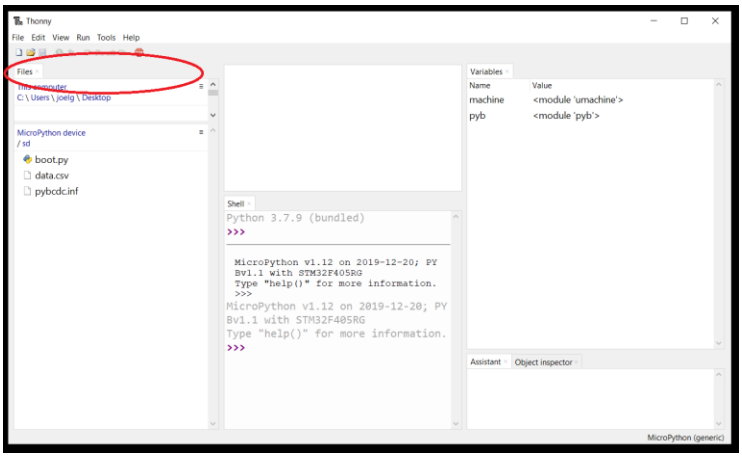
Building out the PyBoard

The course web page, in the *Resources for Labs* section, has instructions and photos on how to mount your PyBoard on a breadboard. Do that now.

Talking to your PyBoard via a host PC

1. You built out your PyBoard with a USB cable attached. Plug the unattached end of the USB cable into any USB port on the host PC.
2. Run a program called *Thonny* on the PC. For those of you who don't use Windows much, you click on the Windows "Start" icon in the lower left of the screen (it looks like a little Window with four panes), scroll down to **Thonny** and click on it.
3. Make sure that Thonny knows the PyBoard exists. The lower right of the Thonny window should say "MicroPython generic" rather than "Python 3.7.9" (or some similar version). If not, click on that lower-right corner and pick the right one. If "MicroPython generic" isn't listed as a choice, then check if you have the USB cable plugged in. The picture below shows the correct version.
4. Now you're ready to go!

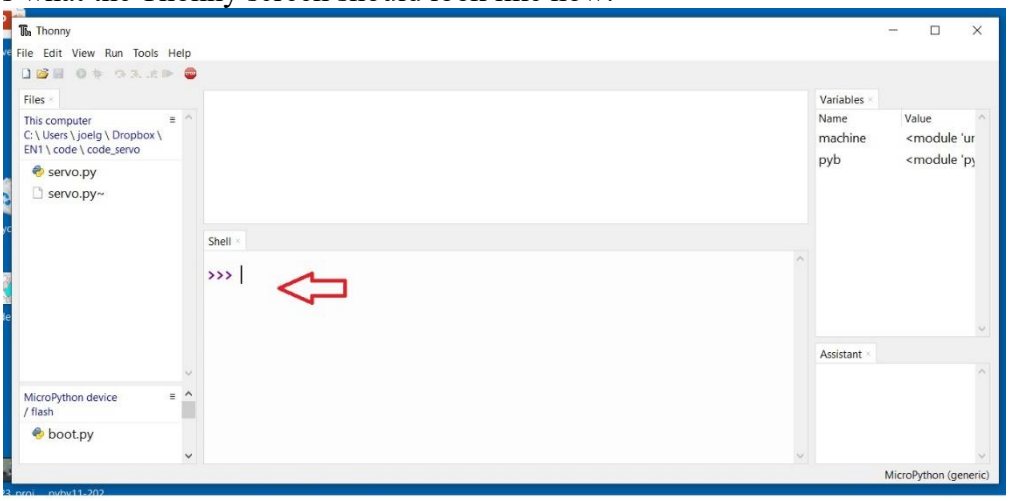
It’s possible that your window might look more like this, with a *Files* pane on the left:



If so, then you can just go to the menus on the top of the window, select View, and then deselect *Files* to remove the window. Similarly, you may have various panes on the right (the picture above has panes for *Variables*, *Assistant* and *Object Inspector*. You don’t really need any of these for now; feel free to keep them or to remove them.

Using Thonny’s REPL as a mini-calculator and more

Here’s what the Thonny screen should look like now:



You’ll notice the bottom half of the screen, labeled the “Shell” window, has a >>> prompt. We’re going to use that to do some simple arithmetic.

The Shell allows us to enter commands directly into Python, and as soon as we enter in a command, Python will execute it and display its result. We'll be using this Shell window a lot when we’re exploring Python: it’s very nice because we get back results immediately. If it helps, you can think of it as a very powerful calculator.

Let’s start out by doing just that: using Python as a calculator. In general, you type a command into the Shell window at the “>>>” prompt, and Python invokes the command and computes the results.

At the prompt, type **`10+15`** and the "Enter" key:

Now do the same thing for a few more calculations; e.g., **`20-5`** and **`10*10`**. Note that you can also use parentheses; e.g., try out both **`3+4*5`** and **`(3+4)*5`**. The order of operations of Python is just what you learned in high school; multiplication and division first, and then addition and subtraction. In Python, multiplication uses "*" and division uses "/".

Try something illegal, like **`1/0`**. What happens? Python's error messages can sometimes be a bit difficult to decipher; with practice, you'll get good at it.

Python has plenty of operators other than +, -, * and /. Try typing **`import math`** first, and then **`math.sqrt(9)`** and also, on a different line, **`math.sqrt(25).`** What do you think "***`sqrt`***" is doing?

Similar to **`sqrt()`**, Python also has **`math.round(), math.abs() and math.log10()`**. Try these out; what do you think they do?

REPL behind the scenes

The feature you've been using is called a ***R**ead-**E**valuate-**P**rint **L**oop* (typically called REPL). You may be wondering what any of this has to do with the PyBoard, since you're running Thonny on the lab's host PC.

The answer is that even though Thonny is serving as the user interface (i.e., letting you use the host computer's keyboard and screen), *all of the computing is happening on the PyBoard*. The PC is batching up all of your keystrokes, sending them to the PyBoard over USB, getting the results back over USB, and displaying them. Yes, this somewhat contorted environment is fairly standard in the world of embedded computing 😊.

Using the PyBoard LEDs

Perhaps you don't really believe that you're really running code on the PyBoard. OK, time to convince the non-believers. Use your REPL to run the following code:

```
import pyb
fred = pyb.LED(1)
fred.on()
```

Hopefully, a red LED turns on. Time to learn by playing a bit:

- Can you turn it off?
- When you typed **`pyb.LED(1)`** you told the PyBoard to use LED #1. What happens if you use LED #2, #3 or #4? How about #5?
- Does it matter if you change "fred" to a different name?
- Can you turn on various combinations of LEDs?

You've actually just taken advantage of multiple concepts:

- You're using *variables*. "Fred" (or whatever else you called it) is a variable.
- "Fred" is holding something Python calls an *object*. We won't really be learning about objects – just think of the variable "fred" as holding as much information about one of the LEDs as it needs to control that LED.

If you like, you can find the full documentation about the PyBoard LEDs at https://docs.micropython.org/en/v1.10/library/pyb.LED.html. Beware – it uses language such as *classes*, *constructors* and *methods* that may not make much sense yet. And you don't really need it all for this course, anyway.

Running an entire program

Typing in your lines of code one by one every time you want to do something can get tedious. Let's make it easier by writing a program that you can save in a file. The book *Think Python* calls this "script mode."

The top half of the Thonny window is the "Program" window. Start off by clicking File/New to open a new program file. Then type in a few lines of your favorite code. E.g.,

```
a = 3+4
b = a + 2
print (a, b)
```

Now that you have your program, you can run it. Click on Run/Run Current Script to run it. What happens next? Two things. First, Thonny saves your program to a file. It asks you whether to put the file on the host computer or on the PyBoard (which it calls the *MicroPython device*; remember that your PyBoard actually runs MicroPython rather than official Python). Click on "Host Computer" and give your file a name.

After that, your program runs! Note that we used the function "print()." This may be confusing in two ways. First, we've only briefly explained what the term "function" even means! Second, we didn't need functions until now for using the REPL, so why do we suddenly need them now?

So what is a function? In short, it's a useful little facility that lets you easily do something (like printing or taking a square root). We won't define it in detail until a bit later; for now, just don't worry, be happy 😊 and trust in good karma (or read the documentation on the course web page under *Python book* and *Python videos and slides*). As for why we didn't need it before – when we were using the REPL (which after all is a read-evaluate-*print* loop), printing was a built-in part of it. But in a program, printing only happens when you specifically tell the program to print. So now that we're writing a real program, we need to explicitly use *print*(). But the upside is that explicitly using *print*() will give you a lot more flexibility in printing exactly what you want to (e.g., printing two things on the same line or not printing intermediate results).

OK, how about this program:

```
import pyb
fred = pyb.LED(1)
jane = pyb.LED(2)
fred.on()
jane.on()
jane.off()
fred.off()
```

What happens when you run this code? Can you see the LEDs flashing? Remember, computers are fast!

If you want to slow things down a bit, you could try this line of code:

```
pyb.delay (1000)
```

It tells Python to stop and do nothing for 1000 milliseconds, which is one second. Try putting this line in various places to see what happens.

Now it's your turn to do this for real. Write a program that lights the LEDs in this sequence:

- green
- green+blue
- blue+yellow
- yellow
- nothing

Each step should take 1.5 seconds

Reverse engineering

Try out writing and running the following short program:

```
import pyb
print (pyb.millis())
```

Run it a few times. Does it print out the same number every time? Do you see any pattern? Hint: try pressing the PyBoard reset button and then run the program a few more times. Hint 2: "millis" is short for "milliseconds."

Reverse-engineering is a fairly common part of life as an engineer!

What to turn in:

A lab report that contains:

- the names of the people in your group
- for each person: your home state and your most likely major (OK, I really don't care about this, but it's mostly to get you to know each other a bit!)
- your program from above
- answer to the questions below

You need only turn in one report for each group.

Questions

- Describe when you might want to use a REPL, and when a program in a file might be more useful.

- What do you think pyb.millis() does?

Bonus

We expect that students will have a wide variety of programming experience. For those who already know a bit more, feel free to experiment with more complex programs. For example, you might use loops to, e.g., flash LEDs repetitively.

What to do if the PyBoard stops working

There is a nice document on the course web page showing how to reset the PyBoard, under *Resources for the labs*.