Lab 1: Intro to Arduino with CircuitPython

A. Trahan, P.E.

September 19, 2019

Introduction

This lab provides an introduction to Arduino with Circuit Python. Arduino devices generally run on Arduino code, which is a set of C/C++ functions; however, recent devices are fast enough to run a python interpreter, opening them to python developers. This course uses an Adafruit Circuit Playground Express, which supports Circuit Python, a lightweight python environment. Since the rest of the course is taught in python, the Arduino segment will be as well. Students should be aware that not all python packages are available in Circuit Python, and that all of these tasks and more can be performed in Arduino code, and may need to be for some Arduino devices.

In this lab we will:

1. Install an Arduino interpreter
2. Connect the Circuit Playground Express (CPX)
3. Run basic programs on the CPX

Required materials (per group):

- (1) Circuit Playground Express (CPX)
- (1) Micro USB cable

1 Installing the Mu Interpreter

Adafruit documentation and tutorials recommend using the python editor Mu, a simple editor with easy-to-use REPL and live plotting. Students are welcome to use an editor/IDE of their choice, but the REPL and Plotter make this is a good place to start.

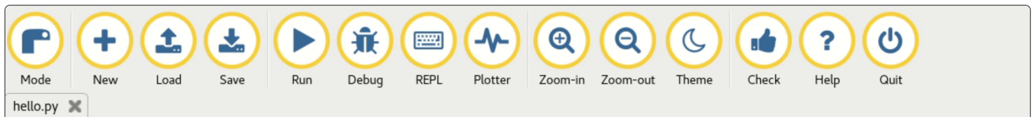


Figure 1: Mu Control Ribbon

To install:

1. Download the latest (stable) version from: https://codewith.mu/
2. Follow installation instructions
3. Open Mu and select your mode: Python 3
4. Test Mu
   (a) Enter `print('Hello World.')`
   (b) Click the **Run** button (Figure 1, button 5)
   (c) Choose a location to save the file
   (d) Review the output in the frame at the bottom

2 Connecting to the CPX

Connecting to a CircuitPython device is relatively easy. In this class, we will be using a Circuit Playground Express (CPX) from Adafruit. Plug the device into the computer via the USB port, and the computer should recognize it as a drive. In that drive will be a `code.py` file. Mu should also update to include an "Adafruit" mode, which will allow the computer to connect via Serial or Plotter. If Mu is open, it may prompt the user to change mode to "Adafruit CitcuitPython Mode". This can also be done manually via the

Mode button (Figure 1, button 1). In Adafruit CitcuitPython Mode, the ribbon will change to reflect options available on the CPX (Figure 2).

If, for some reason, the Arduino device does not already have CircuitPython installed, CircuitPython and installation instructions can be found here: `https://circuitpython.org/downloads`.

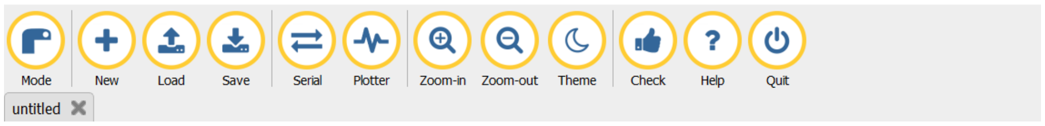


Figure 2: Mu CPX Control Ribbon

3 Basic Programs

Now that the CPX is connected, it's time to run some basic programs. Programs on Arduino consist of two main sections: Setup and Loop. In Setup, sensors, actuators, and program variables are defined. In Loop, a set of actions that will be run on repeat is defined. Because the CPX runs any code in the `code.py` file, it's good practice to keep a copy of the code in a file with a different name. Code from different projects can then be copied from clearly named files into `code.py` on the CPX to run.

3.1 Hello CircuitPy

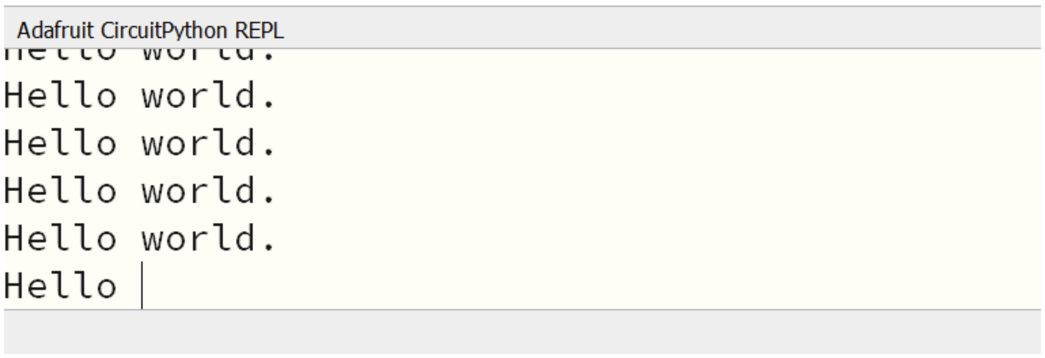
To begin, we will use only the Loop section, but by the time we're done you will have the opportunity to use both sections.

Mu is capable of running a Read-Execute-Print-Loop (REPL) console, which, like the name implies, reads commands, executes them, prints the output, then loops to wait for another command. This is a live interpreter, like iPython or IDLE. In Python 3 mode, Mu opens an iPython session on your computer. In Adafruit Circuit Python mode, the button is replaced by a connection to the device's Serial console, which lets a developper inspect information from print statements. Hitting Ctrl-C (**C**ancel) in the Serial console stops code execution on the CPX and drops into a REPL console.

```
#-- HelloCircuitPy.py - CircuitPython code for CPX --#

# Setup Section
# [Nothing here to start]

# Loop Section
while True:
    print('Hello world.')
```

Steps:

1. Enter the code above in Mu, and save to the CPX as `code.py`
2. The device should reboot automatically ane begin running the script. The green "On" LED should be lit.
3. Click the **Serial** button (Figure 2, button 5)
4. Inspect the output to be sure it is as expected. An example is provided in Figure 3.



```
Hello world.
Hello world.
Hello world.
Hello world.
Hello world.
Hello
```

Figure 3: Example output from hello circuitpy program

5. Update the script to include a Setup Section
   (a) Use the Setup Section to define a variable containing a string (e.g. "Hello from your CPX!")

(b) Print the variable you have created in the Loop Section

(c) Save the code to the CPX and check the Serial console to be sure it is as expected

3.2 Exponential Growth

The Serial console is useful for text output, but with numerical output it can be a bit daunting, and often a plot would be more useful. Enter the Plotter pane. Mu's Plotter pane checks the Serial console for numerical values and plots them on a line chart as they are generated. As an example, we will plot an exponential growth curve: $y(t) = y_0 \cdot e^{kt}$

```
#-- ExponentialGrowth.py - CircuitPython code for CPX --#

# Import Section
from time import sleep
from math import e

# Setup Section
dt = 0.5
t = 0
y0 = 1e-2
k = 1.01

# Loop Section
y = y0
while True:
    y = y + y0*k*e**(k*t)*dt  # Validate this equation by hand
    print(y)
    t+=dt
    sleep(dt)
```

Steps:

1. Take a minute to find the value of Δy and the equation for y_{i+1} based on the exponential growth equation. You can take the derivative of the equation above, then rearange to get an estimate of Δy as a function of Δt. Compare that to the calculation line in the code.

2. Enter the code above in Mu, and save to the CPX as `code.py`

3. The device should reboot automatically ane begin running the script. The green "On" LED should be lit.

4. Click the **Plotter** button (Figure 2, button 6)

5. Inspect the output to be sure it is as expected. An example is presented in Figure 4.

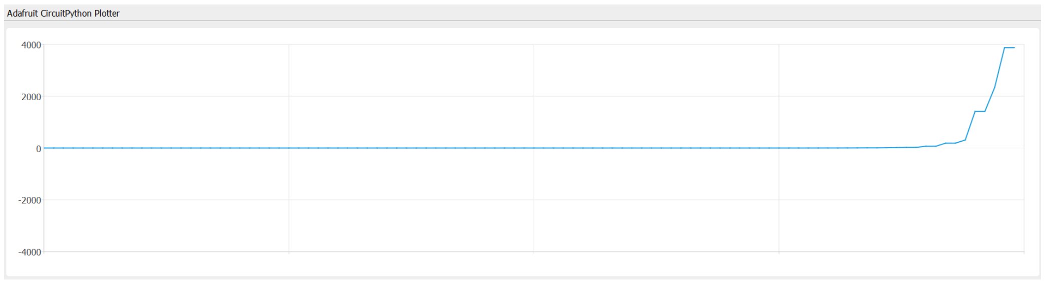


Figure 4: Example plot from exponential growth program (blockiness is due to large dt)

3.3 Logistic Growth and Decay

With python and a bit of math running on the CPX, it's time to start using some of the sensors (inputs) on the device. While unbounded exponential growth can be interesting, systems with some form of upper and lower limit are more common and often represented with a logistic function. This function is often defined by its derivative w.r.t. time (see below), which describes growth or decay, depending on the sign of r, the growth rate. The logistic function grows toward a defined maximum or descends toward a defined minimum, slowing as it approaches either asymptote.

Logistic growth/decay examples range from population dynamics to turbine power curves.We will use the slide switch on the CPX (Figure 5) to switch between growth and decay by multiplying the growth rate, r, by 1 (growth) or -1 (decay). This is similar to turning on and off a light with a dimmer switch, where it slowly warms up to full brightness and cools to darkness when the switch is flipped. In this example, we will watch the results on the Plotter pane when the switch is flipped on and off.

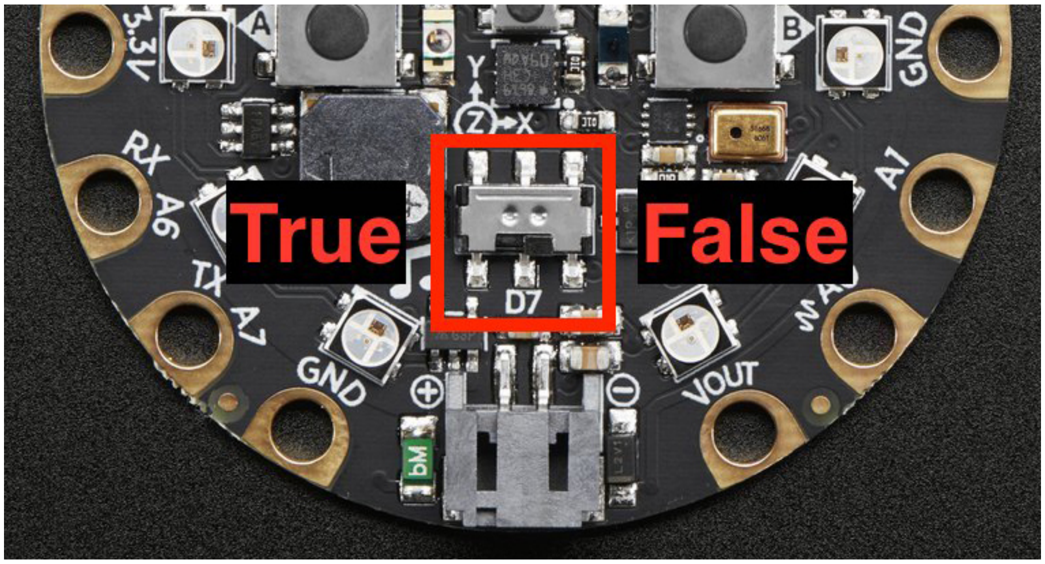


Figure 5: Slide switch on the CPX with values indicated

Steps:

1. Determine the equation for y_{i+1} based on the logistic growth equation

(a) The logistic growth equation is similar to the exponential growth equation, but it is bounded by a maximum value N. The logistic rate (r) drives growth if positive and decay if negative. The equation can be written:

$$\frac{dy}{dt} = r \cdot y \cdot \left(1 - \frac{y}{N}\right)$$

(b) Rearrange this equation to find the equation for Δy and y_{i+1}

2. Enter the code below in Mu, and save to the CPX as `code.py`. A few things to note:

   (a) We import `adafruit_circuitplayground.express.cpx` to use built-in elements of the CPX.

   (b) The slide switch state is accessed via `cpx.switch`, and the values are shown in Figure 5.

   (c) The API reference for this package can be found at `https://circuitpython.readthedocs.io/projects/circuitplayground/en/latest/index.html`.

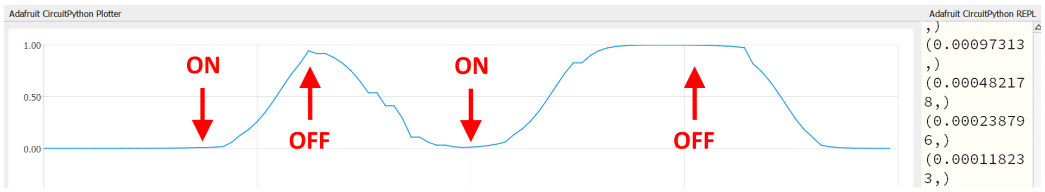
3. The device should reboot automatically ane begin running the script. The green "On" LED should be lit.

4. Click the **Plotter** button (Figure 2, button 6)

5. Inspect the output to be sure it is as expected. An example is presented in Figure 6.

```
#-- LogisticWithSwitch.py - CircuitPython code for CPX --#

# Import Section
from time import sleep
from adafruit_circuitplayground.express import cpx

# Setup Section
dt = 0.5
k = 1.01
```

```
y_max = 1.0
y0 = y_max/2

y = y0  # Start y at y0
growth_decay_factor = 1  # Default to growth

# Loop Section
while True:
    if cpx.switch == True:
        growth_decay_factor = 1
    else:
        growth_decay_factor = -1

    # Validate this equation by hand
    y = y*(1 + (growth_decay_factor*k)*(1-y/y_max)*dt)
    print((y,))
    sleep(dt)
```

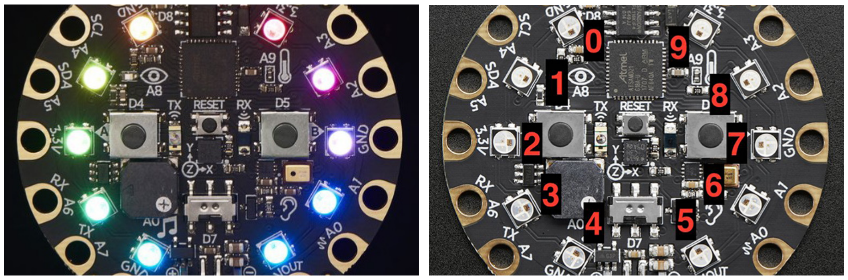


```
,)
(0.00097313
,)
(0.00048217
8,)
(0.00023879
6,)
(0.00011823
3,)
```

Figure 6: Plot from logistic growth/decay program, showing several switch points

3.4 Output with neopixels

Building on the example of a dimmer switch, we will use the switch and logistic function to drive LEDs on the CPX. There are several actuators (outputs) on the CPX, the most prominent of which are the ten neopixels. Each neopixel is a combination of three LEDs (RGB) and a variable resistor to control brightness. They're aranged in a circle around the CPX (Figure 7). In this exercise, we will use neopixel brightness to show the value of the logistic function, bringing together an onboard sensor (the slide switch) and actuator (the neopixels).



(a) Neopixels (b) Neopixel numbering

Figure 7: Neopixel locations and numbering on the CPX

Steps:

1. The logistic equation from above can be used directly in this example. The LED brightness will be defined as y/y_{max}.

2. Enter the code below in Mu, and save to the CPX as `code.py`. A few things to note:

   (a) The neopixels are accessed via `cpx.pixels`

   (b) To set brightness: `cpx.pixels.brightness = 0.5`

   (c) To set color (all): `cpx.pixels.fill((255, 0, 0))`

   (d) To set color (neopixel index i): `cpx.pixels[i] = (255, 0, 0)`

   (e) The API reference for this package can be found at `https://circuitpython.readthedocs.io/projects/circuitplayground/en/latest/index.html`.

3. The device should reboot automatically ane begin running the script. The green "On" LED should be lit.
4. The neopixels should grow brighter as the function increases and dimmer as it decreases. The switch can be used to switch between growth and decay modes, and the results can be compared against the Plotter pane like in the previous example.

```
#-- LogisticWithSwitchAndLeds.py - CircuitPython code for CPX --#

# Import Section
from time import sleep
from adafruit_circuitplayground.express import cpx

# Setup Section
dt = 0.05
k = 1.01
y_max = 1.0
y0 = y_max/2

y = y0  # Start y at y0
growth_decay_factor = 1  # Default to growth

# Initialize Neopixels
cpx.pixels.brightness = y0/y_max
cpx.pixels.fill((0,255,0))

# Loop Section
while True:
    if cpx.switch == True:
        growth_decay_factor = 1
    else:
        growth_decay_factor = -1

    # Validate this equation by hand
    y = y*(1 + (growth_decay_factor*k)*(1-y/y_max)*dt)
    cpx.pixels.brightness = y/y_max

    print((y,))
    sleep(dt)
```

Additional ideas and questions:

1. How would you change the color of the neopixels? Try programming in a color of your choice.
2. In this example, the brightness represents the value of the function. Try modifying the code to make the color represent the value of the function.
3. In this example, all the neopixels light up at together. Try to make them fill as the function grows (i.e. neopixel zero for 0-10%, noepixels 0 and 1 for 10-20%, etc.).
4. Try to combine some of these updates:
   (a) To start, implement the previous idea, where each pixel represents 10%. Turn them on/off as the function grows/decays.
   (b) Instead of turning each pixel on/off at the percentage breaks, try programming them to change brightness within their percentage windows (i.e. neopixel 1 is dim at 11%, brighter at 15%, and fully on at 20%).
   (c) Try using color instead of (or in addition to) brightness to represent value within each percentage window.