



Getting Started with Microcontrollers: Circuit Playground Express

Topic: Intro to Microcontrollers

Suggested Grades: 6th - 12th

Summary: Learn about microcontrollers, set up your Circuit Playground Express and use CircuitPython to program it to light up

Length: 45-90 minutes

Lesson Objectives

Learn how to program an Adafruit Circuit Playground Express microcontroller, download free coding software, use EduBlocks (a block coding software), and write basic code to light up your microcontroller. Be prepared for a taste of the amazing world of microcontrollers and start building your coding and electronic skills!

NASA TechRise Student Challenge Connection

You might be wondering - what does this lesson have to do with the TechRise Challenge? Well, the TechRise Challenge is all about using science *and* technology to discover more about our world! Because microcontrollers can be programmed to perform many functions, they are incredibly useful when it comes to spaceflight research. You can use a microcontroller in your TechRise experiment, and the possibilities are nearly endless.

Contents

Set Up Circuit Playground Express (Pages 1-3)

Download CircuitPython Libraries (Pages 3-4)

Set up the Mu Editor (Pages 4-5)

Light up the Circuit Playground Neopixels with Edublocks (Pages 5-8)

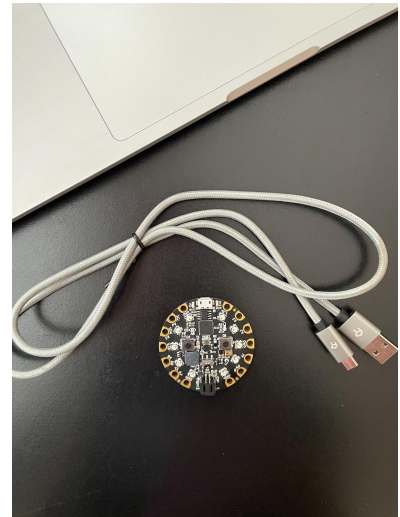
Write CircuitPython Code and Print the Button Values (Page 9)

Light up NeoPixels with Buttons (Pages 10-12)

Troubleshooting & Syntax Errors (Pages 12-14)

Circuit Python REPL (Pages 14-15)

Background Information & Vocabulary (Pages 15-17)

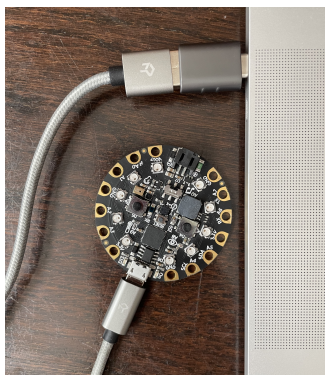


Materials

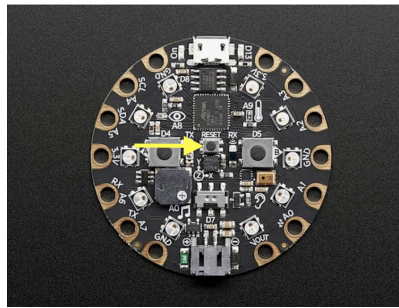
- A. Computer with USB port OR USB C to USB adapters if needed (like the one linked [here](#))
- B. Adafruit Circuit Playground Express Microcontroller Board (like the one linked [here](#))
- C. Micro USB Cord (like the one linked [here](#))

Set up Circuit Playground Express

1. Begin by plugging your Micro USB cord into the USB port on your computer. If you only have USB C ports on your computer, use a USB to USB C adapter to plug into your computer like the one [here](#). Then plug the other end into your Circuit Playground Board.

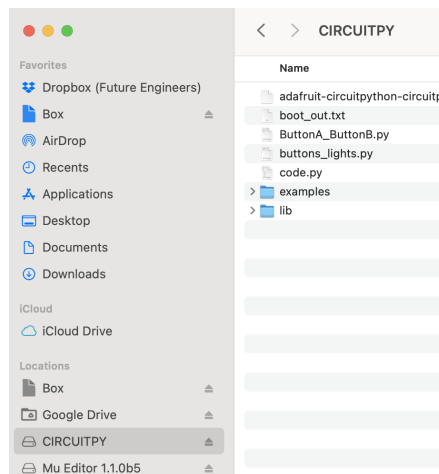


Note that your microcontroller may have been pre-programmed and may start lighting up automatically. If it does, press the tiny “reset” button in the middle of your microcontroller board. If it doesn’t reset, that’s okay! We will reset it later in this lesson.

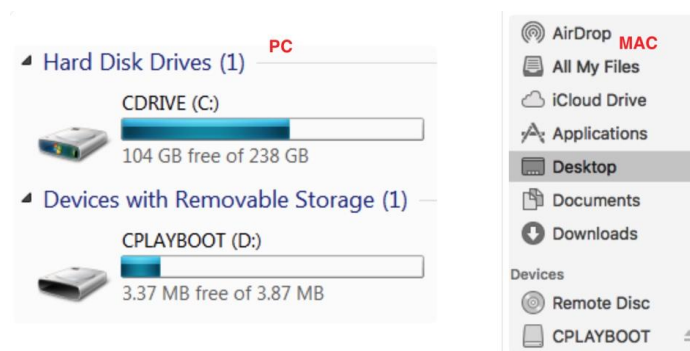


Next, one of two things will happen:

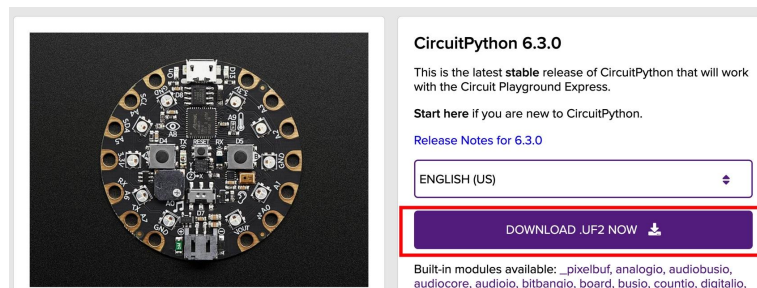
- a. You will see the name of the drive change to CIRCUITPY immediately. If using a Mac, you will find the drive in the “Finder” under “Locations” on the left-hand side of the screen. This means CircuitPython is already installed and the board is ready to use and you can skip step B, and move on to step 2!



- b. **OR**, if you have never downloaded the CircuitPython bootloader before, the drive will appear and will be named CPLAYBOOT.



- i. If you see CPLAYBOOT, this means that CircuitPython is not installed on the board. To download it, go to https://circuitpython.org/board/circuitplayground_express/. Download the “.UF2” file in the top right hand corner of the page.



- ii. **It will download as a file that will not open.** To open it, you must first drag the downloaded file into the CPLAYBOOT drive. From your downloads, drag the downloaded file into the CPLAYBOOT drive as seen.



- iii. You will then see the drive name change to CIRCUITPY. This means CircuitPython is now downloaded to the board.

Download CircuitPython Libraries

2. Go to <https://circuitpython.org/libraries> to download the CircuitPython libraries that need to be installed to your Circuit Playground Express. You will download the most recent Bundle Version (Circled in the picture below). Keep in mind, the bundles are updated frequently and you may see a different version number on the CircuitPython site than the one pictured below.

To install, download the appropriate bundle for your version of CircuitPython. Unzip the file, open the resulting folder and find the lib folder. Open the lib folder and find the library files you need to load. Create a lib folder on your CIRCUITPY drive. Copy the individual library files you need to the lib folder on your CIRCUITPY drive.

You can always find the [latest releases of the libraries bundle](#) on GitHub.

Bundle Version 6.x

This bundle is built for use with CircuitPython 6.x.x. If you are using CircuitPython 6, please download this bundle.

[adafruit-circuitpython-bundle-6.x-mpy-20210608.zip](#)

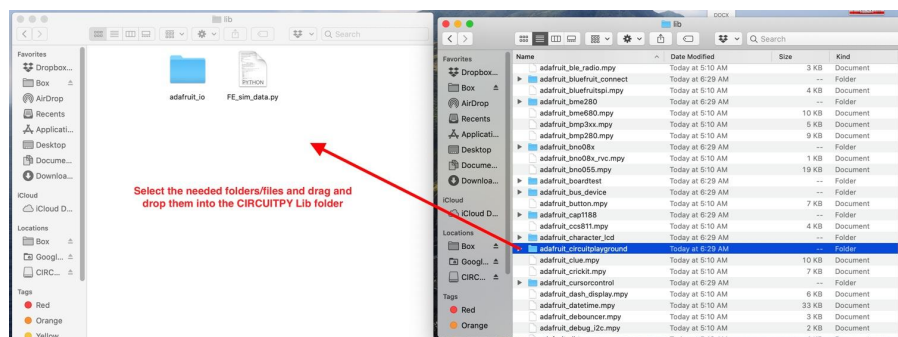
Bundle Version 7.x

This bundle is built for use with CircuitPython 7.x.x. If you are using CircuitPython 7, please download this bundle.

[adafruit-circuitpython-bundle-7.x-mpy-20210608.zip](#)

Bundle Version py

3. The file will be downloaded as a Zip file. Open the file to unzip it. Once the file is open, click to open the folder called "lib". You will see many files open up.
4. Now, click on your CIRCUITPY drive and click to open the folder labeled "lib". You do not need all of the files in the lib folder, you only need the ones listed below. In the downloaded Zip file under "lib", search for and select the files/folders listed below and then drag them into the CIRCUITPY folder "lib". (See photo below) Now your microcontroller will be loaded with the libraries needed for it to be programmed!
 - a. Adafruit_circuitplayground [folder]
 - b. Adafruit_bus_device [folder]
 - c. Adafruit_lis3dh.mpy [file]
 - d. Adafruit_thermistor.mpy [file]
 - e. Neopixel.mpy [file]



Set up the Mu Editor

5. Next, we need to install the **Mu Editor** - this is a text editor that allows us to write our code and save it to our Circuit Playground board. To download, go to the following link and [Download Mu](#) to your computer:
 - a. Choose the installer that is required for your computer, whether it's Mac or Windows and follow the instructions to download.

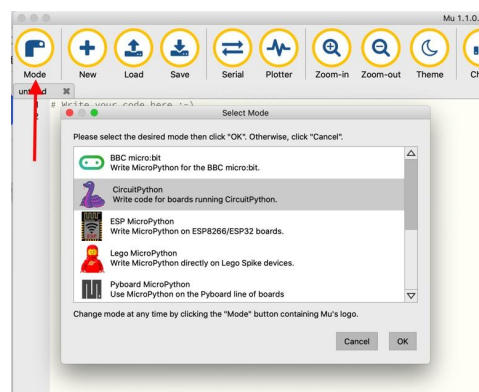
Download Mu

The simplest and easiest way to get Mu is via the OS (we no longer support 32bit Windows).

The current recommended version is Mu 1.1.0-1 this version via the links for each supported operating system version of Mu can be [downloaded from here](#).



6. Once Mu is downloaded, open the application. Once it opens, click "Mode" and then click "CircuitPython". This sets the Mu Editor up to write code in the CircuitPython which is the coding language used to program our Circuit Playground board.

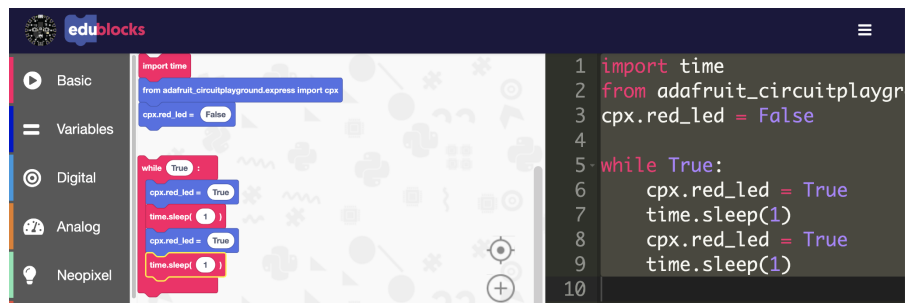


- Keep the Mu Editor open for now as we will come back and use it in the next section.

EduBlocks

Now we are going to light up the neopixels on the Circuit Playground Microcontroller using **EduBlocks**, which is a block-based coding software that is useful for learning to code.

Within EduBlocks, coding scripts can be written through dragging and dropping color coded blocks, which will generate written CircuitPython code that can be used to program the Circuit Playground microcontroller. One of the advantages of Edublocks is that as you drag and drop your blocks, you are able to see the native code on the right hand side of the screen. That code on the right hand side can then be copied and pasted into a text editor like the Mu Editor and then uploaded to the microcontroller.

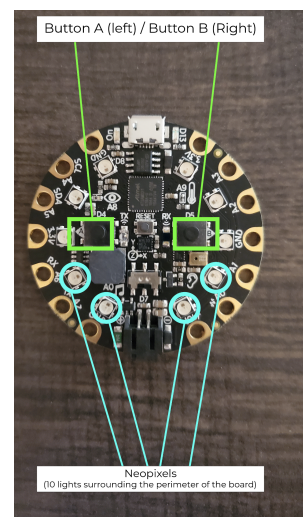
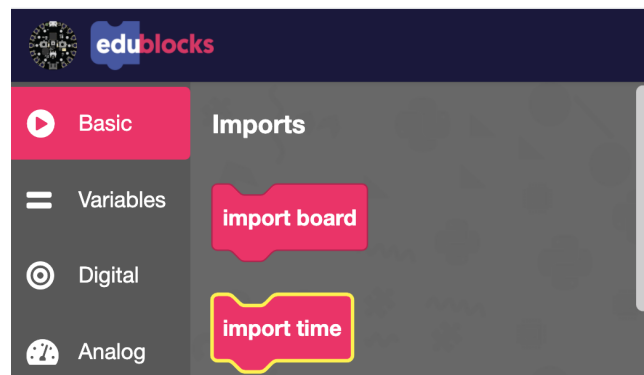


Before we dive deeper into edublocks it is important to first understand how our code is interpreted by the microcontroller. We can think of our program in the code.py file as a script and each line of the script is read, interpreted and executed one by one in sequence. Some lines can be conditional where they are only executed if some condition is met but generally the code is read and executed one line at a time and in an ordered sequence. Edublocks creates code in a similar way by "stacking" blocks together. You can think of each block as a line of code and they will be read and executed in a top to bottom way (the first block is executed first, then second, third, and so on)

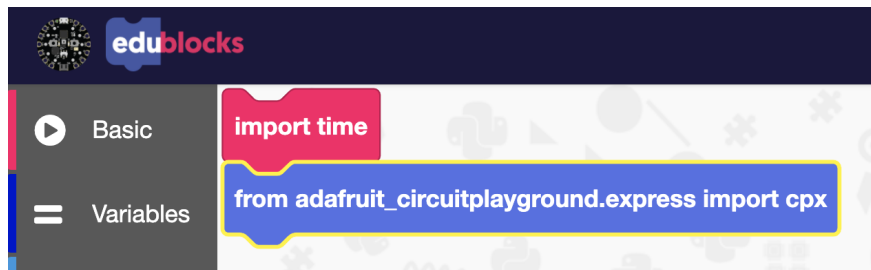
Navigate to <https://www.futureengineers.org/learncode> to begin working with EduBlocks.

Light up the Circuit Playground Neopixels with EduBlocks

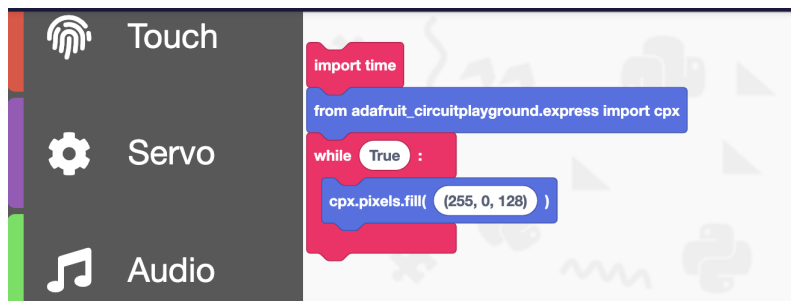
- Now we are going to use [EduBlocks](#) to write a script that will make the neopixels that are embedded on the perimeter of the Circuit Playground Microcontroller blink.
- Open up the EduBlocks link and take a look around at the features. Note that the website is automatically set to write programs in the CircuitPython language, which is the language that the Circuit Playground Express Microcontroller understands.
- Our first lines of code are going to import the libraries we need in order to write this program to the microcontroller. First, we are going to import the time library so that we can utilize functions of time. On the left-hand side of the screen, click "Basic" and choose the "import time" block. Drag it into the white coding area.



- Next: On the left hand side of the Edublocks screen, scroll down the menu and click the “CPX Easy” button. Drag and drop the first block which says, “*from adafruit_circuitplayground.express import cpx*”. Drag this into the coding area and connect it to the “*import time*” block.



- To turn the neopixels on, we are going to create a “loop” using the CircuitPython phrase “*while True*.” In our written program, anything that is indented under “*while True*.” will tell our microcontroller to repeat the function repeatedly. In EduBlocks, any code that we want the microcontroller to repeat will be connected under the “*while True*” block. Since we want the neopixels to turn on, all of our next lines of code will be stacked under “*while True*”.
- In EduBlocks on the left hand menu, click “Basic” and scroll down to “Loops”. Drag and drop the “*while True*” block to our program. Connect it to the bottom of the “*from adafruit_circuitplayground.express import cpx*” block.
- Next, click “CPX Easy” from the left menu and find the “*cpx.pixels.fill(255,0,128)*” block and drag it into the “*while True*” block. The “*cpx.pixels.fill*” command will turn the lights on.

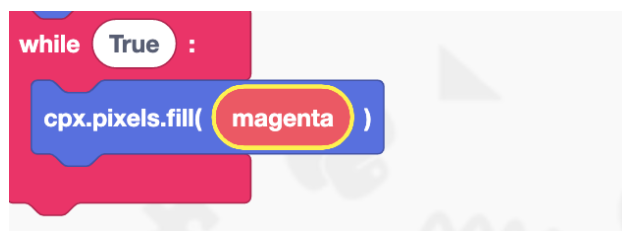


The neopixels are full color, RGB LED lights. This means we can manually alter the red, blue, and green colors within each pixel using numbers between 0-255 with the following code format:

cpx.pixels.fill(R, G, B)

OR we can simply drag and drop the color block in EduBlocks!

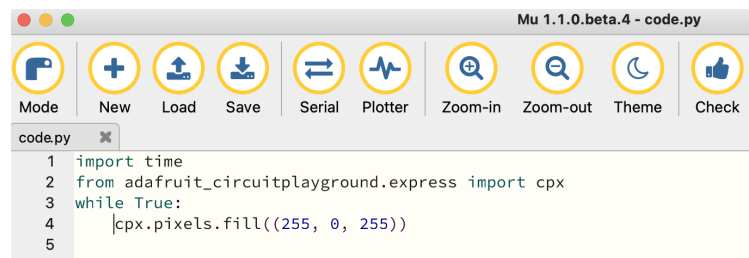
- To change the color of the neopixels in Edublocks, click “Color” at the bottom of the left menu. Click on a color and drag it over the RGB values in the “*cpx.pixels.fill*” block.



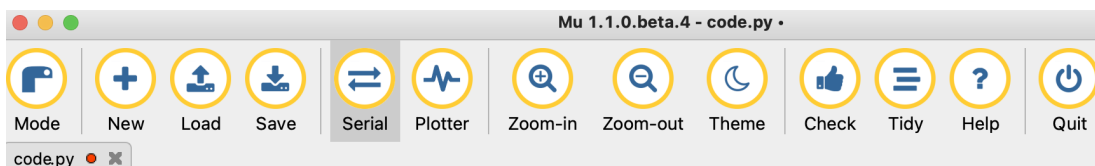
16. Take note of how the native CircuitPython code is building on the right hand of the screen in EduBlocks. We are now going to open our Mu Editor and copy and paste the native code into Mu and upload it to our Circuit Playground microcontroller.



17. Select all of the text on the right hand side of the screen and copy and paste it into Line 1 of the Mu Editor.



18. Before pressing save, ensure that there is no current code running on the microcontroller by clicking the “Serial” button at the top of the Mu Editor.



19. The Serial Monitor, which is also called the “Circuit Python REPL” window will open at the bottom of the screen. Use your cursor to click into the window. Once clicked in, press “Control-C” on your keyboard. This will stop any current program from running on the microcontroller. Once you have typed “Control-C” you should see the following message in the CircuitPython REPL window:

```
CircuitPython REPL
Auto-reload is on. Simply save files over USB to run them

code.py output:
Traceback (most recent call last):
  File "code.py", line 7, in <module>
KeyboardInterrupt:

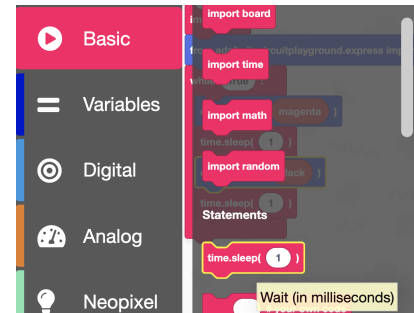
Code done running.

Press any key to enter the REPL. Use CTRL-D to reload.
|
```

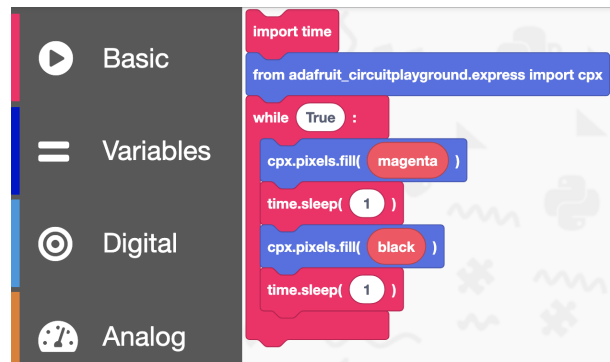
20. Now we are ready to save our new code! Ensure that your microcontroller is still plugged into your computer and click the “Save” button at the top of the Mu Editor. Save the file with the name “*code.py*”. If your computer asks if you would like to replace the current “*code.py*” file click Yes. The neopixels should now be lit up on your microcontroller!
21. Next we will make the neopixels blink and discover how we can get creative and build upon our code! Before we add to our code, let’s stop the current code from running. Click the “Serial” button at the top of Mu and a panel called the “CircuitPython REPL” will appear at the bottom of the screen. Click into it with your mouse, and press “Control-C” on your keyboard. You should see a reply back in the window that says “Code done running.” Now we will be able to go back and add more code and save it to our microcontroller.

22. To blink our neopixels, we are going to write code that essentially fills the Neopixels with a color (turns them on), and then fills them with 0 colors (turns them off). Because the microcontroller performs each function very quickly, if we do not tell the microcontroller to take a pause in between each action, we will not be able to detect the light blinking due to how quickly it will blink. In order to see it blink, we need to tell the microcontroller to take a pause in between each function. The “*time.sleep*” command will tell our microcontroller to pause the function for one second.

23. To build upon our current code and make the lights blink, go to the left menu, click “Basic”, and under “Statements”, click and drag the *time.sleep(1)* block directly under the *cpx.fill* block.

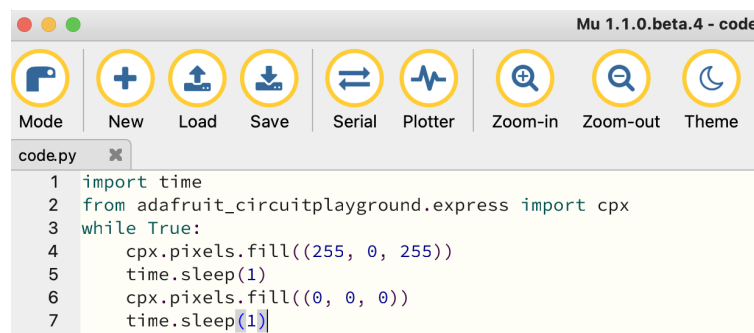


24. Next, under CPX Easy, click and drag another *cpx.pixels.fill(255,0,128)* block under the *time.sleep(1)* block. You can also do this by right-clicking the first *cpx.fill* block and clicking “duplicate”, then drag it into the correct spot.
25. Now go to “Colors” in the left menu and click and drag the color Black, which will fill the RGB values with 0 and turn the neopixels off.
26. Lastly, drag one more *time.sleep(1)* block (or duplicate it) under the *cpx.fill* block as seen in the picture below:



27. We now have created a loop to blink the neopixels! The coding program is now complete and ready to upload to our microcontroller.

28. Select all of the native code on the right hand side of the screen and copy and paste it into Line 1 of the Mu Editor (erasing any previous code). When you paste it into Mu, make sure all the indents are correctly formatted. It should look like this within the Mu Editor:



29. Make sure your microcontroller is still plugged into your computer. Click the “Save” button at the top of the Mu Editor and save over the “code.py” file. Congratulations - the Neopixels should now be blinking on the microcontroller and you have written your first script! If for some reason it is not working, know that errors are more than common, and troubleshooting is a normal part of

coding. Please refer to the Troubleshooting section further down in this lesson.

Write Circuit Python Code and Print the Button Values

In the following sections we are going to practice writing code directly in the Mu Editor, but you are more than welcome to use EduBlocks as well. In this section we are going to see what kind of data the microcontroller prints to the **serial monitor** when the buttons on the board are pressed and not pressed. The serial monitor is essentially like a computer screen for our microcontroller. Just like a computer screen gives us feedback about what our computer screen is doing, the serial monitor allows us to see feedback and data from the microcontroller.

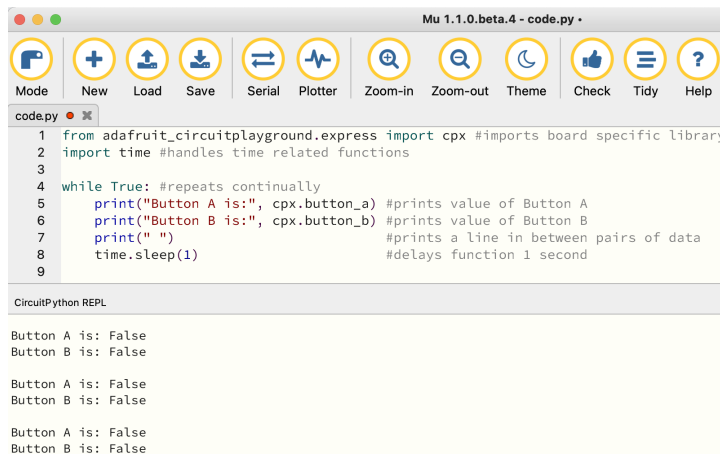
30. First start by erasing any previous code written in the Mu Editor. You can do this by highlighting the code and deleting it. Next, click "Serial" and click into the CircuitPython REPL window and press "Control-C" to stop any code from running.
31. Starting again at line 1, write or copy/paste the following code into the Mu Editor. Note that it is vitally important to ensure that the indents, spelling, and punctuation of the code is 100% accurate and as seen below, or else the microcontroller will not be able to process it.

```
from adafruit_circuitplayground.express import cpx
import time

while True:
    print("Button A is:", cpx.button_a)
    print("Button B is:", cpx.button_b)
    print(" ")
    time.sleep(1)
```

32. After your code is written in the Mu Editor, click "Save" and save the file as code.py. Mu may ask if you would like to replace the current code.py file. You will click "Yes" to replace it. Click into the Serial monitor by clicking the "serial" button at the top of Mu. Once the serial monitor is open, you should begin seeing data being printed as seen in the photo below. *If your code does not run, please refer to the "Troubleshooting" section further down in this lesson.

Notice in the REPL that it is showing Button A and Button B are "False" while we are not pressing the buttons. Press and hold Button A or Button B down and look at the data being printed in the serial monitor simultaneously. What do you think the data will show once we press one of the buttons?



The screenshot shows the Mu Editor window titled "Mu 1.1.0.beta.4 - code.py". The top toolbar includes buttons for Mode, New, Load, Save, Serial, Plotter, Zoom-in, Zoom-out, Theme, Check, Tidy, and Help. The code editor displays the following Python code:

```
1 from adafruit_circuitplayground.express import cpx #imports board specific library
2 import time #handles time related functions
3
4 while True: #repeats continually
5     print("Button A is:", cpx.button_a) #prints value of Button A
6     print("Button B is:", cpx.button_b) #prints value of Button B
7     print(" ") #prints a line in between pairs of data
8     time.sleep(1) #delays function 1 second
9
```

Below the code editor is the "CircuitPython REPL" window, which shows the output of the code:

```
Button A is: False
Button B is: False

Button A is: False
Button B is: False

Button A is: False
Button B is: False
```

Once you press one of the buttons, the data returning to the serial monitor, or REPL, will show "True." Therefore, when the buttons on the board are pressed, the data reads "True," and when they are not pressed, the data reads "False." We can now understand the values that are coming back from these simple digital sensors that we also know as buttons.



The screenshot shows the "CircuitPython REPL" window with the following output:

```
Button A is: True
Button B is: False

Button A is: True
Button B is: False

Button A is: True
Button B is: False

Button A is: True
Button B is: False
```


Light up the NeoPixels with a Button

Now that we understand what data is coming in when we press the buttons, we can do something with the data. At this point, we can add to our current code and tell our microcontroller to perform a certain function “if” the data is at a certain value (True or False). Specifically, we are going to light up certain neopixels on our board if we press a certain button on the board. This will help give us an understanding of “if statements” within CircuitPython coding.

In this section we will write some code that is repetitive and requires changing only a few characters on each line. Remember that we can use Copy and Paste to do this quickly!

33. First, let's adjust the brightness of our neopixels. At full brightness, they are nearly too bright to look at, so we can tell the microcontroller to dim it to 10% brightness by writing this next line of code. Click to line 3 in the Mu Editor down and press “Enter” or “Return” to enter down to line 4 and write, or copy and paste the following code:

```
cpx.pixels.brightness = 0.1
```

This value is from 0 - 1, so by using .1 as the value, we set the brightness at 10%.

34. Next, we are going to add to our current code and tell neopixels 0-4 to turn on when button A on the microcontroller is pressed, and 5-9 to turn on when button B is pressed. We can tell the individual pixels what to do with the syntax `cpx.pixels[N]`, where N is a number between 0 and 9 that corresponds to one of the neopixels on the board.

The neopixels are full color, RGB LED lights. This means we can alter the red, blue, and green colors within each pixel using numbers between 0-255 with the following code format:

```
cpx.pixels[N] = (R, G, B)
```

For example:

To make neopixel #1 red: `cpx.pixels[1] = (255, 0, 0)`

To make neopixel #1 blue: `cpx.pixels[1] = (0, 0, 255)`

To make neopixel #1 pink: `cpx.pixels[1] = (150, 0, 150)`

....And so on! You can get creative and change the values to get different colors in this part.

35. In the Mu Editor, begin at line 13 after our other lines of code. Keep this line of code indented since it will still be under the “while True” loop. Since we know that the data value that is received when the buttons are pressed is “True”, we are going to use that data to write an “if” statement.

```
if cpx.button_a is True:
    cpx.pixels[0] = (0, 0, 255)
    cpx.pixels[1] = (0, 0, 255)
    cpx.pixels[2] = (0, 0, 255)
    cpx.pixels[3] = (0, 0, 255)
    cpx.pixels[4] = (0, 0, 255)
```

36. Now that we have an “if ____ is True” statement, we need a statement that instructs the microcontroller what to do “if ____ is not True”. We can do this by using “else” commands. We will first add an “else” statement for Button A. Click into Line 20 and paste the following code:

```
else:
    cpx.pixels[0] = (0, 0, 0)
    cpx.pixels[1] = (0, 0, 0)
    cpx.pixels[2] = (0, 0, 0)
    cpx.pixels[3] = (0, 0, 0)
    cpx.pixels[4] = (0, 0, 0)
```

37. After that, we will do exactly what we just did in the previous steps but for Button B. We will add both an “if” and an “else” statement for Button B. Add the following code into line 27 of the Mu Editor:

```
if cpx.button_b is True:
    cpx.pixels[5] = (255, 0, 0)
    cpx.pixels[6] = (255, 0, 0)
    cpx.pixels[7] = (255, 0, 0)
    cpx.pixels[8] = (255, 0, 0)
    cpx.pixels[9] = (255, 0, 0)

else:
    cpx.pixels[5] = (0, 0, 0)
    cpx.pixels[6] = (0, 0, 0)
    cpx.pixels[7] = (0, 0, 0)
    cpx.pixels[8] = (0, 0, 0)
    cpx.pixels[9] = (0, 0, 0)
```

38. With everything put together, the entire program will look like this:

```
import time
from adafruit_circuitplayground.express import cpx

cpx.pixels.brightness=0.1

while True:

    print("Button A is:", cpx.button_a)
    print("Button B is:", cpx.button_b)
    print(" ")
    time.sleep(0.1)

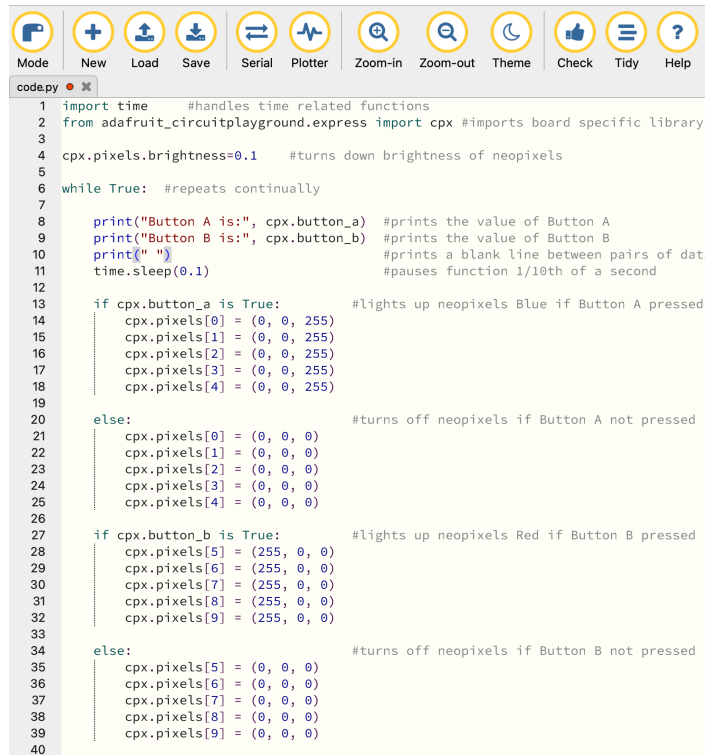
    if cpx.button_a is True:
        cpx.pixels[0] = (0, 0, 255)
        cpx.pixels[1] = (0, 0, 255)
        cpx.pixels[2] = (0, 0, 255)
        cpx.pixels[3] = (0, 0, 255)
        cpx.pixels[4] = (0, 0, 255)

    else:
        cpx.pixels[0] = (0, 0, 0)
        cpx.pixels[1] = (0, 0, 0)
        cpx.pixels[2] = (0, 0, 0)
        cpx.pixels[3] = (0, 0, 0)
        cpx.pixels[4] = (0, 0, 0)

    if cpx.button_b is True:
        cpx.pixels[5] = (255, 0, 0)
        cpx.pixels[6] = (255, 0, 0)
        cpx.pixels[7] = (255, 0, 0)
        cpx.pixels[8] = (255, 0, 0)
        cpx.pixels[9] = (255, 0, 0)

    else:
        cpx.pixels[5] = (0, 0, 0)
        cpx.pixels[6] = (0, 0, 0)
        cpx.pixels[7] = (0, 0, 0)
        cpx.pixels[8] = (0, 0, 0)
        cpx.pixels[9] = (0, 0, 0)
```

39. Press “Save” in the Mu Editor. It will replace the current ‘code.py’ file. Now press the buttons, and the lights should light up! If for some reason it does not light up, try clicking into the Serial Monitor and pressing Control-C, and then press save again. If the code still does not run after trying this, please refer to the troubleshooting section.



```
1 import time #handles time related functions
2 from adafruit_circuitplayground.express import cpx #imports board specific library
3
4 cpx.pixels.brightness=0.1 #turns down brightness of neopixels
5
6 while True: #repeats continually
7
8     print("Button A is:", cpx.button_a) #prints the value of Button A
9     print("Button B is:", cpx.button_b) #prints the value of Button B
10    print("") #prints a blank line between pairs of data
11    time.sleep(0.1) #pauses function 1/10th of a second
12
13    if cpx.button_a is True: #lights up neopixels Blue if Button A pressed
14        cpx.pixels[0] = (0, 0, 255)
15        cpx.pixels[1] = (0, 0, 255)
16        cpx.pixels[2] = (0, 0, 255)
17        cpx.pixels[3] = (0, 0, 255)
18        cpx.pixels[4] = (0, 0, 255)
19
20    else: #turns off neopixels if Button A not pressed
21        cpx.pixels[0] = (0, 0, 0)
22        cpx.pixels[1] = (0, 0, 0)
23        cpx.pixels[2] = (0, 0, 0)
24        cpx.pixels[3] = (0, 0, 0)
25        cpx.pixels[4] = (0, 0, 0)
26
27    if cpx.button_b is True: #lights up neopixels Red if Button B pressed
28        cpx.pixels[5] = (255, 0, 0)
29        cpx.pixels[6] = (255, 0, 0)
30        cpx.pixels[7] = (255, 0, 0)
31        cpx.pixels[8] = (255, 0, 0)
32        cpx.pixels[9] = (255, 0, 0)
33
34    else: #turns off neopixels if Button B not pressed
35        cpx.pixels[5] = (0, 0, 0)
36        cpx.pixels[6] = (0, 0, 0)
37        cpx.pixels[7] = (0, 0, 0)
38        cpx.pixels[8] = (0, 0, 0)
39        cpx.pixels[9] = (0, 0, 0)
40
```

Now that you’ve got an idea about how a microcontroller can be programmed to do something based on a trigger(pressing a button), you can check out our follow up lesson “Use Blue Origin Flight Data to Trigger your Microcontroller: Circuit Playground Express”. This lesson will show you how you can program a microcontroller to light up based on NASA TechRise Flight Simulator data.

Troubleshooting

Experiencing errors is a common part of coding. Even experts experience errors and have to troubleshoot. If you are experiencing an error within your code, or the code will not run, try the following:

- a. Check all the hardware connections including the USB ports to ensure everything is connected properly.
- b. Sometimes if you have been working with the Serial Monitor, it will change to REPL mode and won’t run any new code until the current code is stopped.
 - i. Click the “serial” button at the top of Mu, click into the bottom panel with your cursor and press “Control-C” on your keyboard. This should stop any current code from running and make it so that you can now save new code to the microcontroller. Once you see “Code done running” in the REPL window, you can now press the “Save” button at the top of Mu to save the new program.
 - ii. If after pressing “Control-C” the code still will not run, click into the REPL window again with your cursor. Now press “Control-D” -- this will reload the code that you have saved. Press “Save” to upload the new code.
- c. Click the “Check” button at the top of the Mu Editor - it will show you if you have any Syntax Errors (too many spaces, no indent, etc.)
 - i. If you do have syntax errors, check the following: Did you include all the correct brackets, quotation marks and indents?
 - ii. Additional Spaces: Check to see if there are too many spaces in your indents under the while True statements. If you have copied and pasted your code from an external source, it sometimes does not paste in the correct format. For example: a “tab” over in Google Docs is 4 spaces. In Mu, a “tab” is 3 spaces. If you have copied and pasted from Google Docs into Mu, the indents may contain too many spaces and CircuitPython will not be able to process the line of code. Correct your indented lines of code in the Mu

Editor by backspacing each indented line all the way to the left, and then press the tab button back over to make it indent correctly. Pressing “tab” in the Mu Editor will space it 3 spaces, which is the correct amount needed for lines of code.

- d. Have you copied the necessary library files to the board? Check the lib folder in the CIRCUITPY drive and make sure all of the files for the exercise are there! Refer to Page 2 of the lesson plan to ensure you have the correct libraries.
- e. Make sure you are saving your code as “code.py” since that is the file that the Circuit Playground is reading. If you save it as a different file name and press save, the microcontroller may not be able to read the code.

If you have tried the above and are still experiencing errors, please check CircuitPython’s troubleshooting link:

<https://circuitpython.readthedocs.io/en/latest/docs/troubleshooting.html> , or reach out to us via email at support@futureengineers.org

Syntax Errors and Troubleshooting

Errors in code happen often, making troubleshooting a vital part of each person’s coding journey - even the coding experts! We are going to intentionally trigger an error in the next steps to show how the REPL can help us determine what the error is with our code. Specifically we will focus on a syntax error, which occurs when the interpreter does not recognize code because of a spelling error or punctuation error in the code. Coding languages have strict guidelines and when those guidelines are not followed, the microcontroller will not be able to read the code, and the Mu Editor will show that there is an error within the code.

Common syntax errors include, but are not limited to: lack of quotation marks, erroneous capitalizations, incorrect spacing within indents or lack of indents. Additionally, too many spaces, typos and more can trigger syntax errors.

In the next steps, we will intentionally write an error in our code to show what the REPL does when there is an error in the code.

1. We are going to use a simple coding program to demonstrate an error. Begin by erasing the current code in your Mu Editor. Additionally, click the “Serial” button to open up the REPL window at the bottom. Click into the REPL window with your cursor, and click “Control-C” on the keyboard to ensure that no prior code is running.
2. Next, at line 1 at the top of the Mu Editor, copy and paste the following code into the editor. Make sure the indents are indented exactly as seen in the code below:

```
import time
from adafruit_circuitplayground.express import cpx
cpx.red_led = False
```

```
While True:
    cpx.red_led = True
    time.sleep(0.5)
    cpx.red_led = False
    time.sleep(0.5)
```

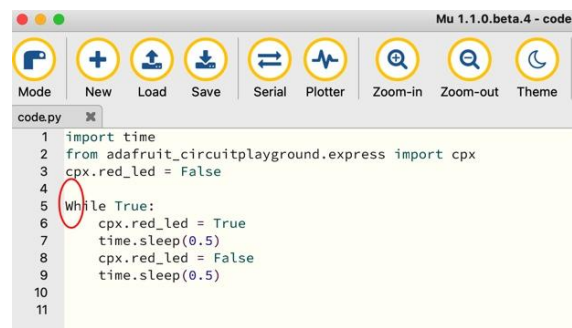
3. Press the Save button at the top of the editor and you should notice that the code does not run. A message appear within the REPL window:
4. Instead of running the program, we see within the CircuitPython REPL window that there is a “Syntax Error: invalid syntax”, and right above that line it says line 5. Here the REPL is letting us know that there is an error in our code, in line 5. If you take a look at line 5 of our code, it is the “While True” statement. The reason it has triggered an error is that “while” in “While True:” is supposed to be *uncapitalized*.

```
CircuitPython REPL
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

code.py output:
Traceback (most recent call last):
  File "code.py", line 5
SyntaxError: invalid syntax

Code done running.

Press any key to enter the REPL. Use CTRL-D to reload.
```



5. To correct the code, we can change the Capitalized “W” in “While” to a lowercase w. Press the “Save” button in the Mu Editor and your code will begin running and your LED light on the microcontroller will begin blinking.

Circuit Python REPL

What is the CircuitPython REPL? The **CircuitPython REPL** is a window within the Mu Editor that opens up when we click the SERIAL button within the Mu Editor. The REPL, in short, is an interactive command line tool, which the user can use to work out code. It can read what the user inputs, and then provide instant feedback. It is especially useful for detecting errors in a written program and can be useful when creating new coding projects.

REPL (pronounced “REP-UL”) stands for:

- a. **Read** (the computer reads the user’s input, or code)
- b. **Evaluate** (the computer evaluates the code to try to understand what the user wants to happen)
- c. **Print** (the computer prints any results for the user to see)
- d. **Loop** (the computer loops back to the beginning to repeat the conversation)

For more information on the use of the REPL, check out <https://codewith.mu/en/tutorials/1.1/repl>

1. To see how the REPL works, follow the next steps: Click the “Serial” button at the top of the Mu Editor. A panel at the bottom should open up.
2. Click into the panel window and press Control-C. Control-C commands the controller to stop running any code it may have been pre-programmed with.
3. Now hit ENTER and you will enter a live window called “CircuitPython REPL.”
4. Now the REPL is ready to receive commands. Staying within the REPL window, press ENTER until you see this: “>>>”
5. To see how the REPL provides instant feedback, let’s test out a command. Directly following the three chevrons, or “>>>”, type the following words and characters:

`print("hello world!")`

Keep in mind, any code we type into our editor must be written exactly as seen here with no other spaces, characters, capitalizations, etc.

6. After you have input the code, press ENTER. You will see the REPL instantly responds and prints the phrase “hello world!” as we specified.

```
Press any key to enter the REPL. Use CTRL-D to reload.

Adafruit CircuitPython 6.2.0 on 2021-04-05; Adafruit CircuitPlayground Express with samd21g18
>>> print("hello world!")
hello world!
>>>
```

7. Next, let’s intentionally write our print command incorrectly to trigger an error and see how the REPL responds to errors. Within the REPL window, directly following the ‘>>>’, type the following command as written below:

`print(hello world!)`

8. Press ENTER, and you should see the following message within the REPL:

```
>>> print(hello world!)
Traceback (most recent call last):
  File "<stdin>", line 1
SyntaxError: invalid syntax
>>> |
```


- Note that the REPL says there is a `SyntaxError` in line 1. Do you know what our print command is missing that made it trigger a syntax error? You may have guessed it: quotation marks! The correct print command should be `print("hello world!")`. Without the quotation marks, the microcontroller is unable to read the command, and therefore the REPL will notify us that there is an error.

From this exercise, we can understand that the REPL is useful for testing out code, as it provides instant feedback, including what line of our code may be causing errors. We will touch more on common Syntax Errors further in this lesson.

- The code that is currently running can be stopped at any time by pressing the "Serial" button, clicking into the REPL window, and pressing 'Control-C' on the keyboard. Control-D will reload the program. "Enter" or "Return" will enter you back into REPL mode and provide the characters "`>>>`".

The CircuitPython REPL is a tool that has two modes. Most often, the REPL functions in the serial monitor mode that provides an easy way to see output data from the microcontroller with the `print()` command. In this lesson we wrote code to print the values of the buttons, and those values printed back to us as "True" or "False" in the serial monitor. The serial monitor is sort of like receiving "text messages" from the microcontroller.

The second mode of the CircuitPython REPL, which functions after pressing any key to get to the "`>>>`" prompt, is a way to write and send single commands to the microcontroller, one line at a time, without saving any files or writing long lines of code. The REPL is especially useful for debugging and figuring out errors within a program.

Common REPL Commands:

- Control-C: Keyboard interrupt; or, stops current code from running
- Control-D: Reloads the program
- Enter(PC) or Return(Mac): Enters down to a new line and provides the "`>>>`" prompt.

Background Information & Vocabulary

What is a Microcontroller?

Microcontrollers are a part of everyday life and are everywhere. From computer keyboards, to microwave ovens, the electrical devices around you rely on microcontrollers to work. Even a car can have over 65 microcontrollers in it! Microcontrollers are programmable integrated circuits that come in all shapes and sizes. They are essentially mini computers that can do basic math and logical computer operations. Many microcontrollers can perform millions of functions per second! Typically, they are used in applications that require a repetitive function - such as, but not limited to, traffic lights, microwaves, washer/dryers, hi-tech devices, satellites, rockets, and more! In our modern world, microcontrollers are all around us.

Microcontroller Board

A microcontroller board is an electronic device that carries a microcontroller embedded onto it. There are many different types of boards used for different purposes. Often microcontroller boards incorporate various kinds of hardware and circuitry to support experimentation.

Circuit Playground Express

The Circuit Playground Express is a microcontroller board that has many built in features, making it an excellent introductory microcontroller that can be used for endless projects. It can be programmed using Microsoft Make-Code block-coding, EduBlocks block-coding, Java-script Programming, CircuitPython, Python, and others. Each Circuit Playground Express has the following hardware onboard:

- 10 x mini NeoPixels, each one can display any color
- 1 x Motion sensor (LIS3DH triple-axis accelerometer with tap detection, free-fall detection)
- 1 x Temperature sensor (thermistor)
- 1 x Light sensor (phototransistor). Can also act as a color sensor and pulse sensor.
- 1 x Sound sensor (MEMS microphone)
- 1 x Mini speaker with class D amplifier (7.5mm magnetic speaker/buzzer)
- 2 x Push buttons, labeled A and B
- 1 x Slide switch
- Infrared receiver and transmitter - can receive and transmit any remote control codes, as well as send messages between Circuit Playground Expresses. Can also act as a proximity sensor.
- 8 x alligator-clip friendly input/output pins
- Includes I2C, UART, 8 pins that can do analog inputs, multiple PWM output
- 7 pads can act as capacitive touch inputs and the 1 remaining is a true analog output
- Green "ON" LED so you know its powered
- Red "#13" LED for basic blinking
- Reset button

- ATSAM21 ARM Cortex M0 Processor, running at 3.3V and 48MHz
- 2 MB of SPI Flash storage, used primarily with CircuitPython to store code and libraries.
- MicroUSB port for programming and debugging
- USB port can act like serial port, keyboard, mouse, joystick or MIDI

Check out this [Circuit Playground Express Tour](#) to learn more!

What is Coding?

Coding is the process used to communicate with a microcontroller in order to get it to perform something or behave a certain way. Code is referred to as a language because you can write code to tell your microcontroller what to do. There are many different coding languages and different microcontrollers can only understand certain types of code, but some of the most popular ones include JavaScript, CircuitPython, Python, Java, C#, C++ and more! In order to get a microcontroller to do something, we must upload instructional code that the microcontroller can understand and respond to accordingly.

What is a Program?

Coding can also be referred to as programming. A program is a set of written coding instructions given to your microcontroller to carry out.

Block Code

Block code is a type of software tool that converts text-based coding languages into a visual block format. Block coding is useful for learning and practicing code as the blocks are already prepared and the user can simply drag and drop them to create a program. Because the code is already pre written within each block of code, it allows the user to practice coding without experiencing syntax or writing based errors which can often frustrate a beginner. The visual experience allows users to grow in computational thinking skills and other foundational coding skills.

EduBlocks

EduBlocks is a block-based coding software that is useful for learning code. Future Engineers has a customized EduBlocks learning page intended to provide a learning environment specific to the NASA TechRise Challenge <https://www.futureengineers.org/learncode>

CircuitPython

CircuitPython is a programming language that was created to simplify learning and experimenting with microcontrollers. Learn more about CircuitPython here: <https://learn.adafruit.com/adafruit-circuit-playground-express/what-is-circuitpython>

Mu Editor

The Mu Editor is a simple text editor that can be used to write code and program microcontrollers. Code can be written within the editor and then saved as a text file to the microcontroller.

Serial Monitor

The serial monitor is the tool that is used for communication between the computer and the microcontroller. The Mu Editor has a serial monitor directly built into the application making it easy to see serial monitor feedback. Essentially, data is exchanged between the serial monitor and the microcontroller via a USB cord. The data can travel two ways:

1. Data or commands can be sent from the computer to the microcontroller (PC → Microcontroller)
 2. Data can be received from the microcontroller to the PC (Microcontroller → PC) and displayed in the serial monitor.
- This is usually used for debugging and monitoring purposes.

Neopixels

Neopixels is the name Adafruit uses to refer to the LED lights embedded upon Adafruit microcontrollers. Adafruit neopixels are unique in that they are full color RGB LED lights that are controlled through a single wire, meaning they can be controlled individually, as well as chained into longer strings.

Input/Output

Input and Output in terms of microcontrollers is the communication between the microcontroller and the outside world. **Input** is the information that the microcontroller receives from the outside world, which is often from a human or another information processing system. **Output** is the information or signal that the microcontroller puts out. Many microcontrollers have input/output (IO) pins which can function as both input and output.

For example, we sent an input to our microcontroller in this lesson when we wrote code that instructed it to print a message or turn on a light. It then sent an output to the serial monitor where we could read the data coming from it, and turned a light on that we could see turn on.

VOCABULARY LIST

- microcontroller
- microcontroller board
- payload
- code
- program
- block code
- CircuitPython
- Mu Editor
- Serial Monitor
- REPL
- NeoPixels
- LED
- RGB
- output
- input

Resources

- Circuit Python Download for Circuit Playground Express https://circuitpython.org/board/circuitplayground_express/
- Circuit Python Libraries <https://circuitpython.org/libraries>
- Detailed Guide and Overview of the Adafruit Circuit Playground Express Microcontroller <https://learn.adafruit.com/adafruit-circuit-playground-express>
- Mu Editor <https://codewith.mu/en/download>
- What is the REPL? <https://codewith.mu/en/tutorials/1.1/repl>
- Getting Started with Mu, a Python Editor <https://opensource.com/article/18/8/getting-started-mu-python-editor-beginners>