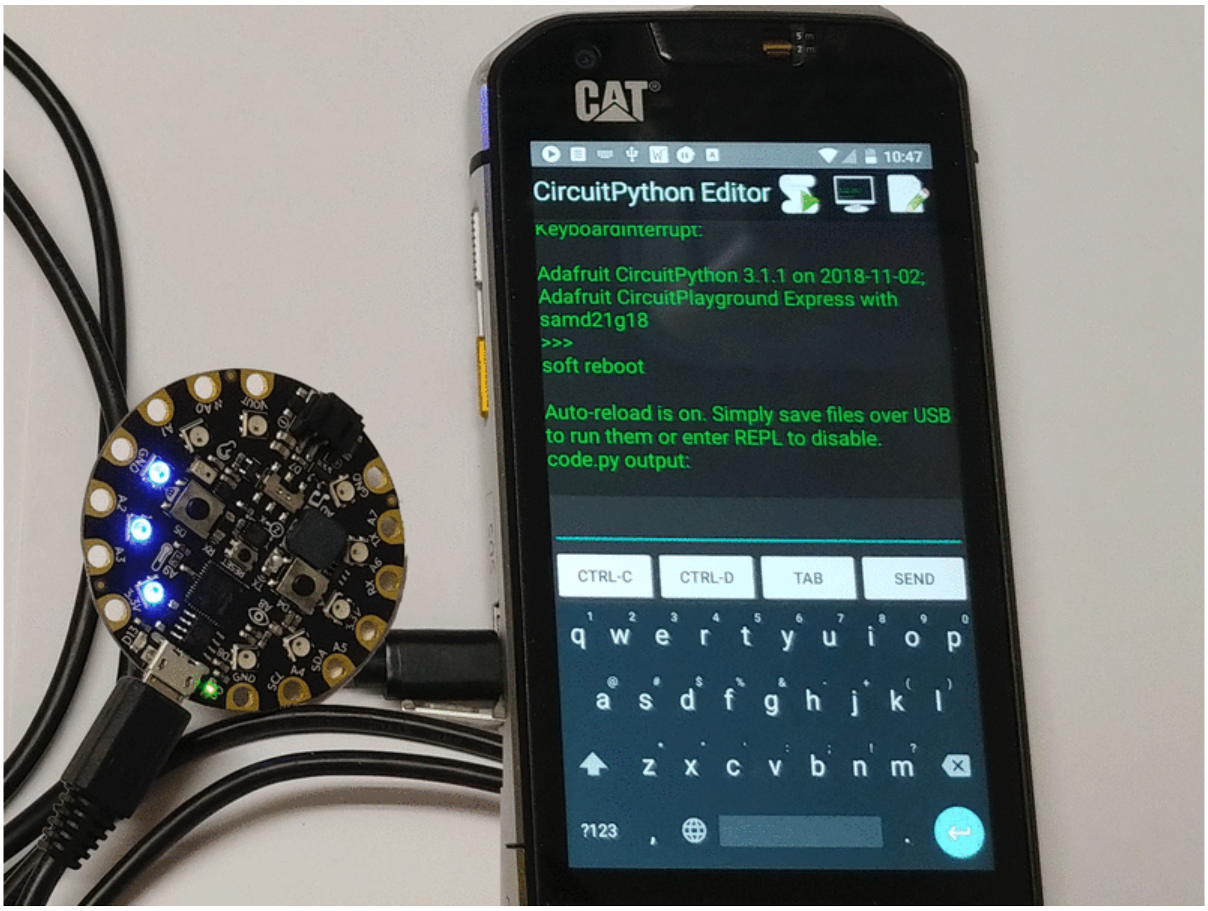
Using the Android CircuitPython Editor

Created by Tim C



```
KeyboardInterrupt:

Adafruit CircuitPython 3.1.1 on 2018-11-02;
Adafruit CircuitPlayground Express with
samd21g18
>>>
soft reboot

Auto-reload is on. Simply save files over USB
to run them or enter REPL to disable.
code.py output:
```

https://learn.adafruit.com/using-the-android-circuitpython-editor

Last updated on 2024-06-03 02:35:02 PM EDT

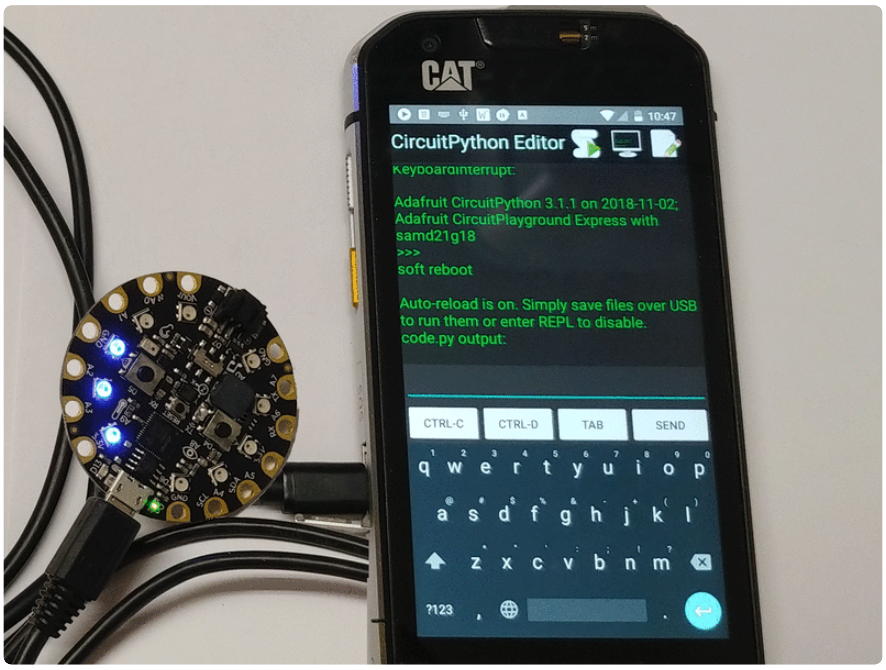
Table of Contents

Overview

Android Circuit Python Editor

This tutorial will show you how to use an Android device to connect with your Circuit Playground Express. Using the Circuit Python Editor application, you can view the output of your code. You can also load, edit, and save the **code.py** file. Additionally, you can use the application to interact with the REPL (Read, Evaluate, Print, Loop) on your board.

Are you new to using CircuitPython? No worries; there is a full getting started guide here. Once you understand the basics, come back here.



```
KeyboardInterrupt:

Adafruit CircuitPython 3.1.1 on 2018-11-02;
Adafruit CircuitPlayground Express with
samd21g18
>>>
soft reboot

Auto-reload is on. Simply save files over USB
to run them or enter REPL to disable.
code.py output:
```

Going Mobile

The Mu Editor (and Pycharm!) are fantastic tools. I use them all the time to debug, create, and edit python code for CircuitPython boards. However, I found myself wanting a way to do some basic debugging and code editing without the need to lug around a laptop, so I created this Android app.

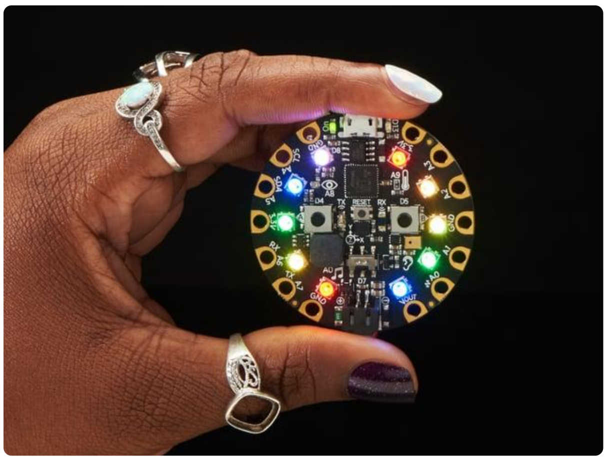
Mobile devices do have their drawbacks: the screen is pretty small, and typing code on a virtual keyboard is not the easiest thing in the world. So, this is not really meant to replace the desktop editors entirely.

But there are some nice upsides, too. The board will be running off of your phone's battery, so you can be on the go without the need for an additional power source.

In particular I wanted to lower the barrier to entry for showing off the magic of CircuitPython to new people. Now I can take a Circuit Playground Express and some cables with me, and I'll already have my phone on me anyway. With this app, I can quickly show people what the board is and how to make it do neat things.

Parts List

In addition to your Android mobile device you will need the following parts:



Circuit Playground Express
Circuit Playground Express is the next step towards a perfect introduction to electronics and programming. We've taken the original Circuit Playground Classic and...
https://www.adafruit.com/product/3333

USB OTG Host Cable - MicroB OTG male to A female
This cable looks like a USB micro cable but it isn't! Instead of a USB A Plug, it has a USB A Socket on the end. This cable is designed for use with OTG (On the Go) host devices...
https://www.adafruit.com/product/1099

USB cable - USB A to Micro-B
This here is your standard A to micro-B USB cable, for USB 1.1 or 2.0. Perfect for connecting a PC to your Metro, Feather, Raspberry Pi or other dev-board or...
https://www.adafruit.com/product/592

Preparation

Installing the Android application

Come to this page on your Android device and click the Link to Play Store (https://adafru.it/DID). Or, if you have the Barcode Scanner installed, scan the QR code to the left.

Link to Google Play Store
https://adafru.it/DID

Mounting the storage as writable

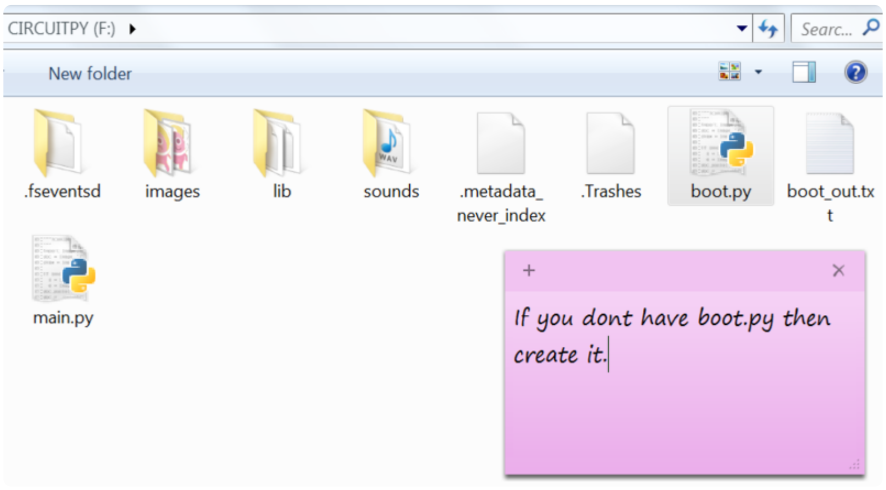
We can't use the standard USB storage method to save code to the Circuit Playground Express because Android does not have support for the type of filesystem formatting used by the board. In order to work around this the application, will use the REPL to load and save the code using Python statements like `f.read()` and `f.write().`

You can only have either your computer edit the CIRCUITPY drive files, or CircuitPython. You cannot have both write to the drive at the same time. (Bad Things Will Happen, so it's one or the other.)

In order for the REPL to be able to save your code, you will need to have the storage mounted as writable. Whenever the storage is mounted as writable for the REPL, it will be not writable using the normal USB storage method.

In this tutorial, we will set it up so that the board will check the Circuit Playground Express on-board slide switch and set the storage mount accordingly. This way, we can switch back and forth easily between standard USB storage, and save from the Android app via REPL. If you want to learn more about using the storage or using the storage on a board besides the Circuit Playground Express, see this guide. (https://adafru.it/DIE) Note that the board must support CircuitPython.

To accomplish this, we'll need to edit the **boot.py** file on the Circuit Playground Express. You'll need to do this step from your computer. If you don't already have a **boot.py** file, it's no problem--just create one.



If you dont have boot.py then create it.

Copy this code into your **boot.py** file:

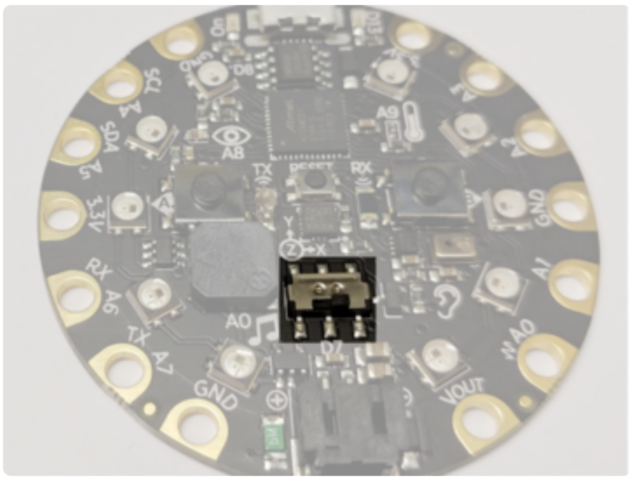
```
import board
import digitalio
import storage

switch = digitalio.DigitalInOut(board.D7)
switch.direction = digitalio.Direction.INPUT
switch.pull = digitalio.Pull.UP

# Mount the storage based on the on-board switch
# Switch position right for REPL/Android app
# Switch position left for normal USB Storage
storage.remount("/", switch.value)
```

As the name suggests, **boot.py** only gets executed when the board first boots up. Because of this, you must unplug the board from power, then flip the switch to the desired position while it's unplugged. After you've flipped the switch, you can plug the board back in.

If the switch is in the left position, the board will behave like the standard USB storage / flash drive.

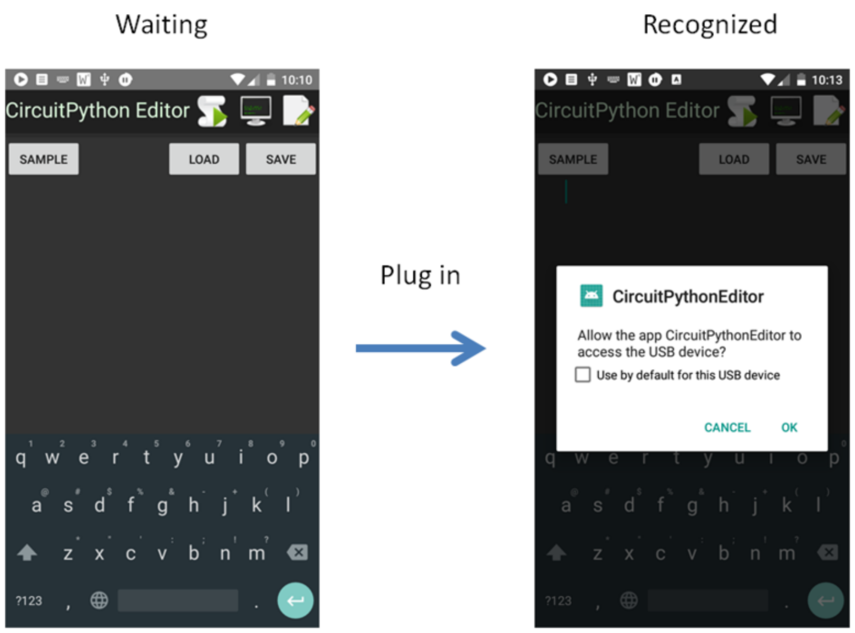
If the switch is in the right position, the board will be mounted so that the Android application can save your code with REPL.

Connecting the Wires

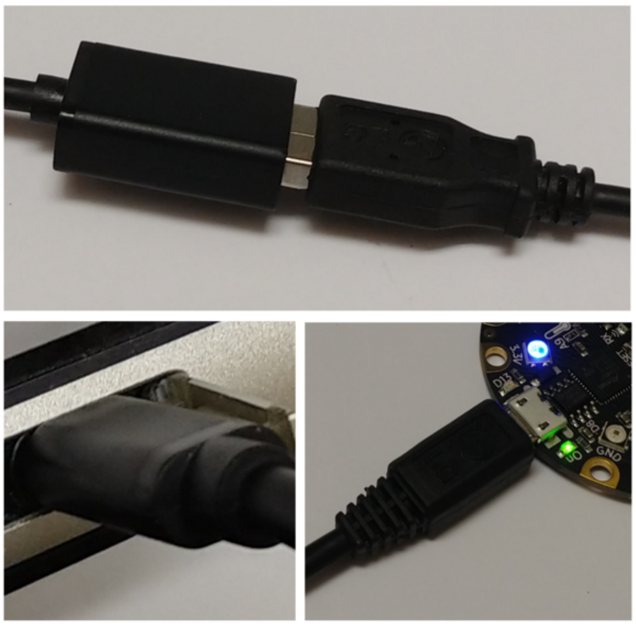


Unplug your board, flip the switch to the right position as shown here

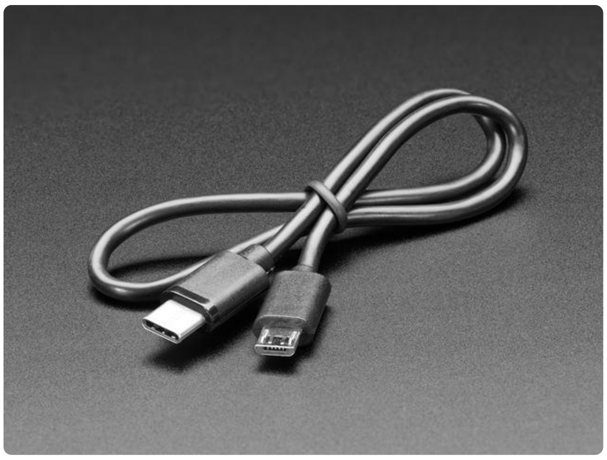
Launch the Android application on your phone and you should see:


Waiting
Recognized
Plug in
CircuitPython Editor
SAMPLE
LOAD
SAVE
CircuitPythonEditor
Allow the app CircuitPythonEditor to access the USB device?
Use by default for this USB device
CANCEL
OK

Use the USB Micro cable and OTG adapter cable to connect the Circuit Playground Express to your Android device. Click 'OK' to allow the application permission to access the Circuit Playground.

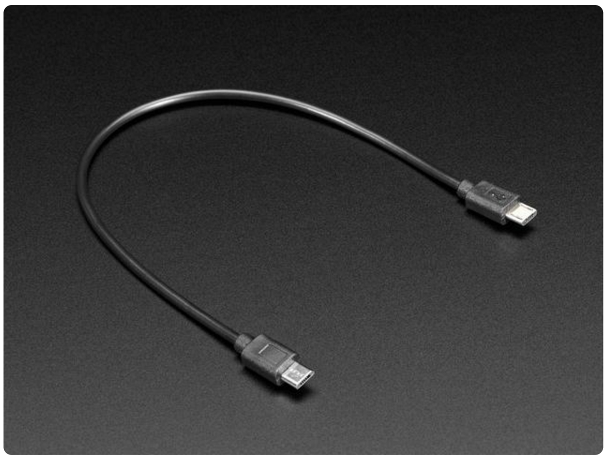


If your Android device has USB C instead of USB Micro, you can use this cable:



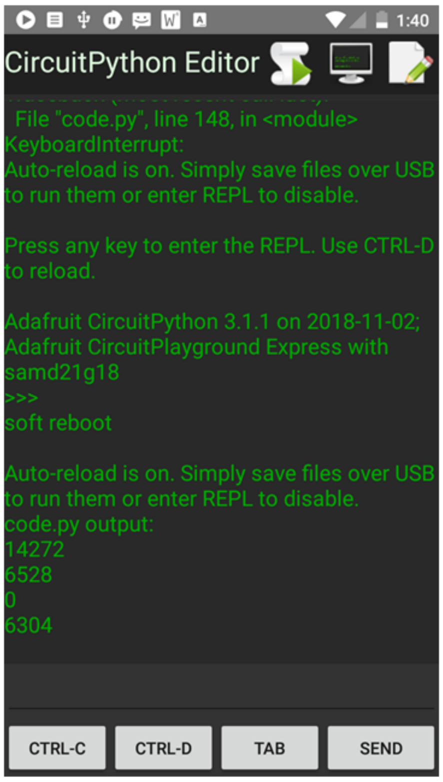
USB C to Micro B Cable - 1 ft 0.3 meter
As technology changes and adapts, so does Adafruit! Rather than the regular USB A, this cable has USB C to Micro B plugs!USB C is the latest...
https://www.adafruit.com/product/3879

If your phone is USB Micro, but you you want to cut the adapter out of the mix, you could also use one of these:

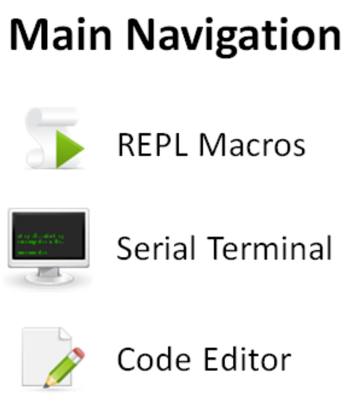


Micro USB to Micro USB OTG Cable - 10-12" / 25-30cm long
This cable is a little unusual, rather than having a USB A plug on one end, it has two Micro B USB connections! What is this for? It's for when you have a "USB...
https://www.adafruit.com/product/3610

Using the Application



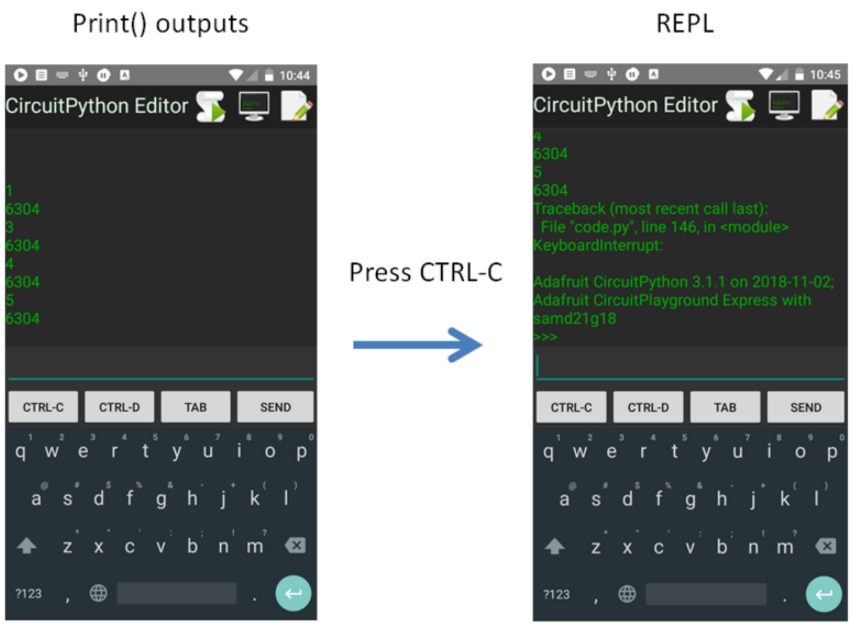
```
  File "code.py", line 148, in <module>
KeyboardInterrupt:
Auto-reload is on. Simply save files over USB
to run them or enter REPL to disable.

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

Adafruit CircuitPython 3.1.1 on 2018-11-02;
Adafruit CircuitPlayground Express with
samd21g18
>>>
soft reboot

Auto-reload is on. Simply save files over USB
to run them or enter REPL to disable.
code.py output:
14272
6528
0
6304
```



Serial Terminal

Once your Circuit Playground Express is connected and you've allowed the application permission to access it, you'll be left at the serial terminal. If your **code.py** has `print()` statements in it, then you should be able to see the output of those statements getting printed into the serial text box.

You can use the **CTRL** buttons to interact with the Circuit Playground. Just like on the computer, **CTRL-C** will break out of your running **code.py** and enter the REPL waiting for you to send it statements. Also, just like on the computer, **CTRL-D** will break out of the REPL and go back to running **code.py**.



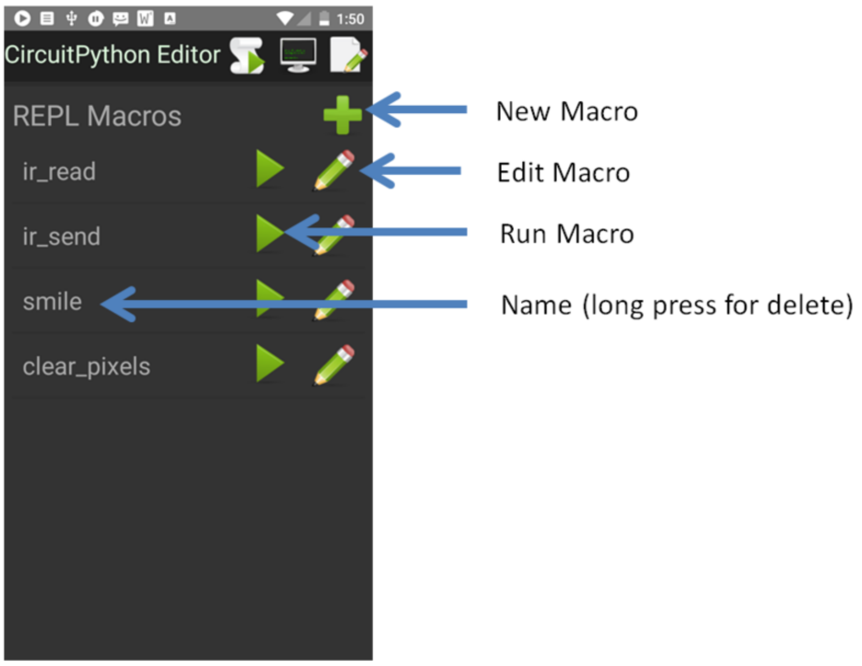
```
1
6304
3
6304
4
6304
5
6304
```

```
4
6304
5
6304
Traceback (most recent call last):
  File "code.py", line 146, in <module>
KeyboardInterrupt:

Adafruit CircuitPython 3.1.1 on 2018-11-02;
Adafruit CircuitPlayground Express with
samd21g18
>>>
```

REPL Macros

This is a feature that is used in conjunction with the Serial Terminal. You can create and store **macros** on your Android device. Then you can enter the REPL on your Circuit Python board and run the macros with the press of a button. Your macro will get sent to the REPL one line at a time.

To get started, go ahead and enter the REPL and then press the **macro icon**. Use the + button to create a macro. You'll be prompted to enter a name. Once you've named the macro, you can use can use the **edit** button to add some code to it. When you're ready, use the **run** button to start sending the macro code into the REPL.



If you want one to get started with macros, here is a script I like to use to make a smiley face with the Circuit Playground Express board:

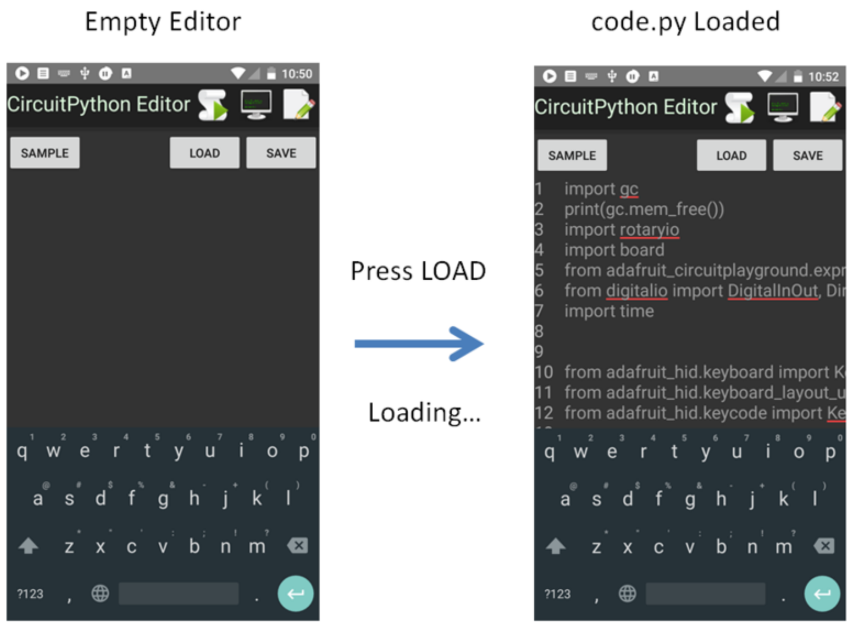
```
from adafruit_circuitplayground.express import cpx
# set color here
color = (0,0,222)

# setup
pixels = cpx.pixels
pixels.brightness = 0.02
pixels.fill((0,0,0))
pixels.show()

# mouth
for i in range(0,5):
    pixels[i] = color


# eyes
pixels[6] = color
pixels[8] = color
```

Code Editor

Before you can access the code editor, you need to make sure that you are not in the REPL. The **load** button is going to try to open the REPL for you, and if you were already in the REPL, it will warn you. If you are ever unsure, you can always either 1) unplug and re-plug your board (it will always start in normal non-REPL mode) or 2) press the **CTRL-D** button (this is the shortcut key to break out of REPL).

If you are unsure, you can always press **CTRL-D** just to be safe. Even if you weren't already in the REPL, it won't do any harm. If you press **CTRL-D** a few times and nothing happens, then you are good to go.

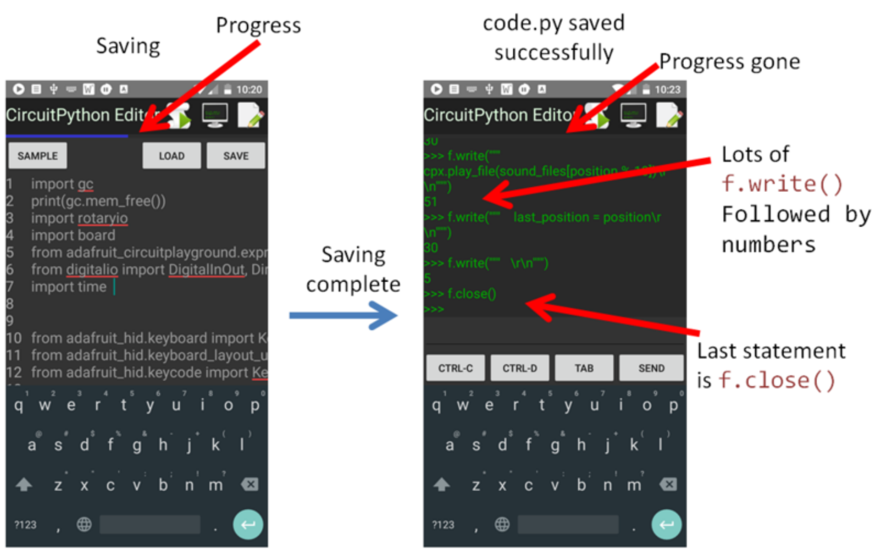
To access the code editor, swipe your finger left to scoot the screen over to the next page on the right.



```
import gc
print(gc.mem_free())
import rotaryio
import board
from adafruit_circuitplayground.expr
from digitalio import DigitalInOut, Di
import time


from adafruit_hid.keyboard import K
from adafruit_hid.keyboard_layout_u
from adafruit_hid.keycode import Ke
```

Press the **LOAD** button. The application will enter REPL and use statements like `f.read()` to send back the contents of **code.py**. Once the loading is complete, the contents of **code.py** will get placed into the editor text for you to view and edit. If you'd like, you can swipe back to the terminal page during loading to view the progress, you should see the contents of your **code.py** file scrolling past in the terminal text box.

If you make edits to the code, you can press the **SAVE** button to save your changes back to the **code.py** file on the board. The saving progress will be shown along the top of the screen. You can also go back to the terminal page to see the progress during saving if you'd like.



```
import gc
print(gc.mem_free())
import rotaryio
import board
from adafruit_circuitplayground.expr
from digitalio import DigitalInOut, Di
import time


from adafruit_hid.keyboard import K
from adafruit_hid.keyboard_layout_u
from adafruit_hid.keycode import Ke
```

```
30
>>> f.write("""
cpx.play_file(sound_files[position % 10])\r
\n""")
51
>>> f.write("""    last_position = position\r
\n""")
30
>>> f.write("""    \r\n""")
5
>>> f.close()
>>>
```

Once the saving is complete, your board will still be left in the REPL waiting for another statement. Press the **CTRL-D** button to break out of REPL and run the newly saved **code.py** file.

You can use this pattern over and over to change and then test your code:

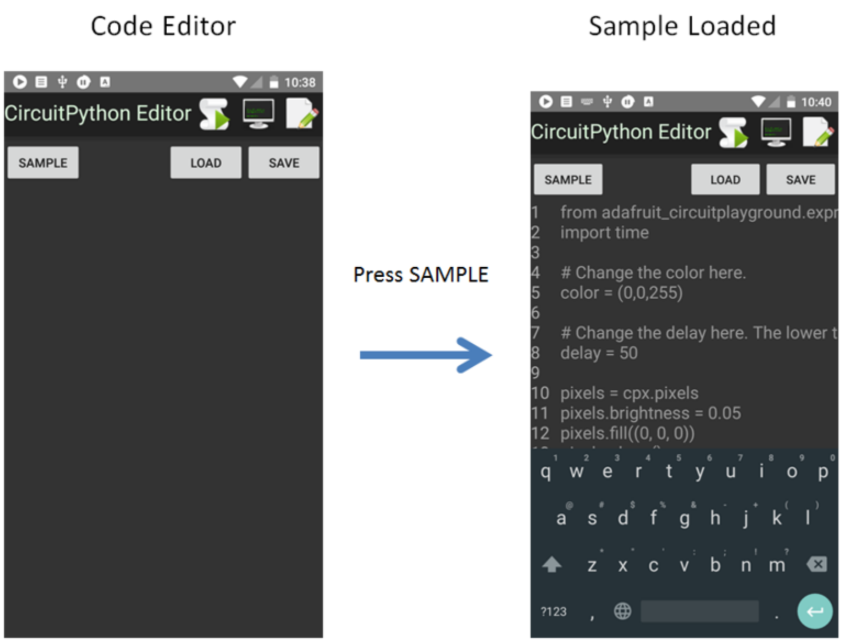
1. Press **CTRL-D**
2. Swipe over to Code Editor page
3. Press **LOAD**
4. Edit code
5. Press **SAVE**
6. Swipe back to Serial Terminal page
7. Press **CTRL-D**

Sample Code

The application comes with a small sample program which will spin a colorful circle around the NeoPixels on the board.

In order to interact with the NeoPixels, the sample code makes use of the `adafruit_circuitplayground` library.

If you haven't already installed the Adafruit libraries onto your Circuit Playground Express board, you can learn about installing the `adafruit_circuitplayground` library in the CircuitPython Essentials Guide on CircuitPlayground Libraries (https://adafru.it/ABU). It is easiest to install the whole library package if you do not plan to use the flash space on the board for other files.



```
from adafruit_circuitplayground.expr
import time

# Change the color here.
color = (0,0,255)

# Change the delay here. The lower t
delay = 50

pixels = cpx.pixels
pixels.brightness = 0.05
pixels.fill((0, 0, 0))
```

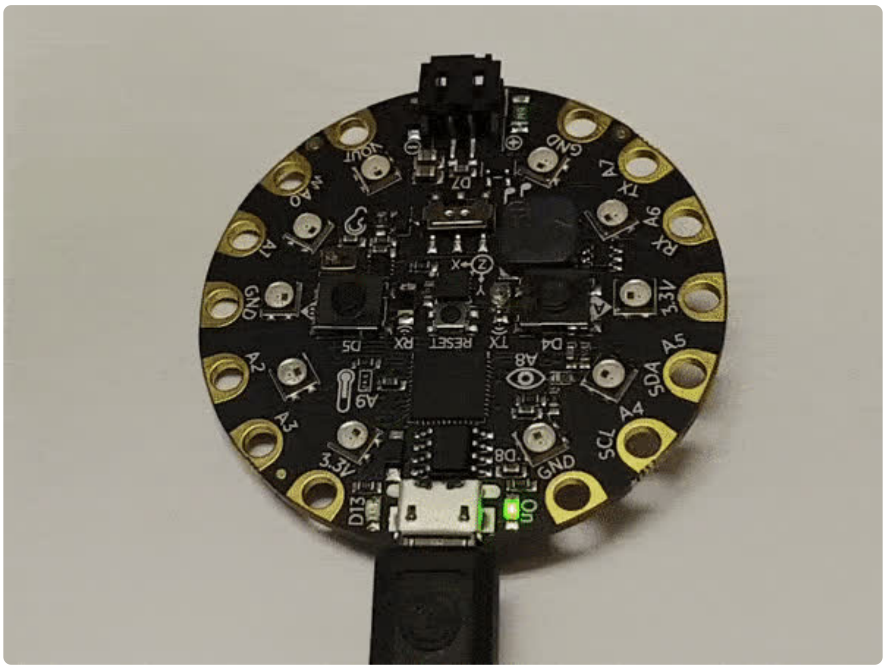
Here is the sample script:

```
from adafruit_circuitplayground.express import cpx
import time

# Change the color here.
color = (0,0,255)

# Change the delay here. The lower the delay value the faster the animation.
delay = 50

pixels = cpx.pixels
pixels.brightness = 0.05
pixels.fill((0, 0, 0))
pixels.show()


while True:
    for i in range(0, len(pixels)):
        pixels[i] = color
        time.sleep(delay * 1/1000)

    for i in range(0, len(pixels)):
        pixels[i] = (0,0,0)
        time.sleep(delay * 1/1000)
```

When you save the sample script it will overwrite whatever was in code.py on your board. Please keep a backup on your computer of all important code.

When you are ready, press the **SAVE** button. This will save the sample script from the code editor into the **code.py** file on your board. If there is anything important in your **code.py** file, please keep a backup on your computer.

Swipe back to the serial terminal page. Once the saving is complete, press the **CTRL-D** button to restart the board and execute **code.py**.



Your board's NeoPixels should animate with a blue circle. If that is working properly, then go ahead and swipe back to the code editor and make a change to the `color` variable in the code on line 5:

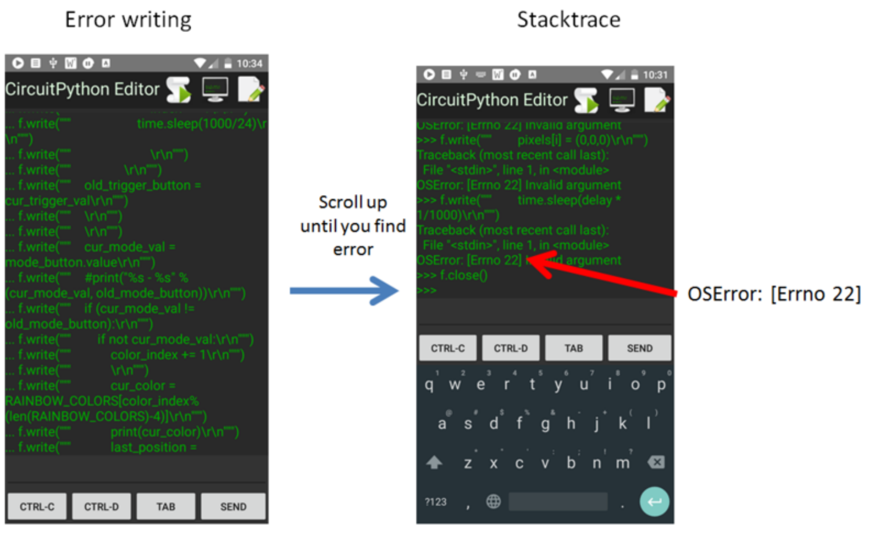
```
color = (0,0,255)
```

The 3 values there represent the RGB values that the NeoPixels will get set to. You could change them to `(0,255,0)` for green, or choose any other color combination you like. If you haven't learned about using the NeoPixels with CircuitPython yet, check out the guide here. (https://adafru.it/Bem)

Once you are ready press the **SAVE** button to save the code back to the boards **code.py** file. Then swipe back again and press the **CTRL-D** button to restart the board and execute **code.py** again you should now see whatever color you chose. Feel free to edit other aspects of the code, another easy one to play with is the `delay` variable.

Problems / FAQ

Problem: When I try to save **code.py** from the Android application the `f.write()` statements in the serial text box don't look correct. There are lots of ellipsis ("...") and/or `OSErrors`, and there are no numbers printed on the lines after. My code does not get saved successfully.



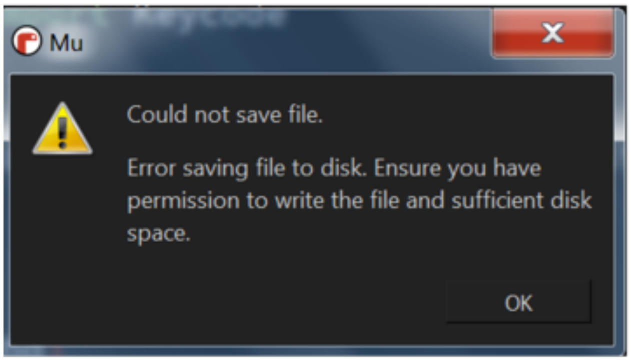
```
... f.write("""           time.sleep(1000/24)\r\n""")
... f.write("""                 \r\n""")
... f.write("""             \r\n""")
... f.write("""    old_trigger_button = cur_trigger_val\r\n""")
... f.write("""    \r\n""")
... f.write("""    \r\n""")
... f.write("""    cur_mode_val = mode_button.value\r\n""")
... f.write("""    #print("%s - %s" % (cur_mode_val, old_mode_button))\r\n""")
... f.write("""    if (cur_mode_val != old_mode_button):\r\n""")
... f.write("""        if not cur_mode_val:\r\n""")
... f.write("""            color_index += 1\r\n""")
... f.write("""            \r\n""")
... f.write("""            cur_color = RAINBOW_COLORS[color_index%(len(RAINBOW_COLORS)-4)]\r\n""")
... f.write("""            print(cur_color)\r\n""")
... f.write("""            last_position =
```

```
OSError: [Errno 22] Invalid argument
>>> f.write("""        pixels[i] = (0,0,0)\r\n""")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
OSError: [Errno 22] Invalid argument
>>> f.write("""        time.sleep(delay * 1/1000)\r\n""")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
OSError: [Errno 22] Invalid argument
>>> f.close()
>>>
```

Explanation: This is the result of trying to write to the storage from REPL when it's not mounted as writable.

Solution: If you are using the **boot.py** file from the preparation page, then you need to unplug the board, flip the on-board switch to the right position, then plug the board back in and try again.

If you aren't using the **boot.py** form the preparation page, you need to edit your own **boot.py** to mount the storage properly for writing.



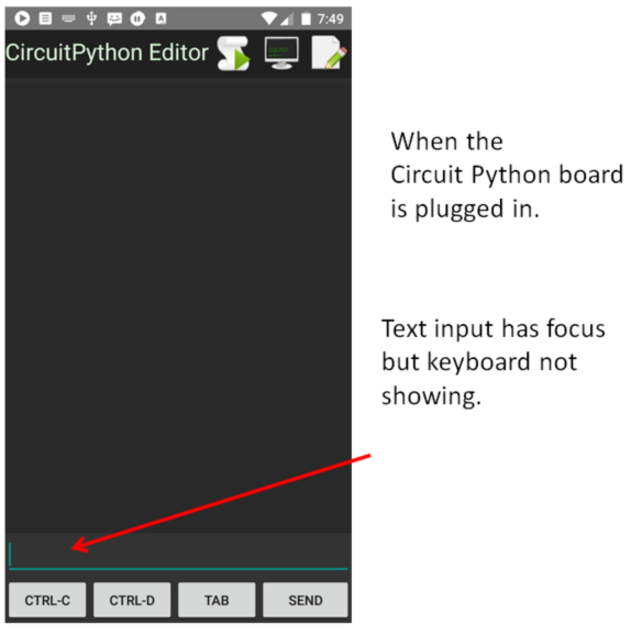
Could not save file.

Error saving file to disk. Ensure you have permission to write the file and sufficient disk space.

Problem: I'm trying to save my code the normal way via Mu and USB storage, but I'm getting an error about permission or write protection.

Explanation: Your board is currently mounted so that the Android app and REPL can write to the storage, but you are trying to save from the PC and USB storage instead.

Solution: If you are using the **boot.py** file from the preparation page, then you need to unplug the board, flip the on-board switch to the left position, then plug the board back in and try again. If you aren't using the **boot.py** form the preparation page you need to edit your own **boot.py** to mount the storage properly to non-writable.

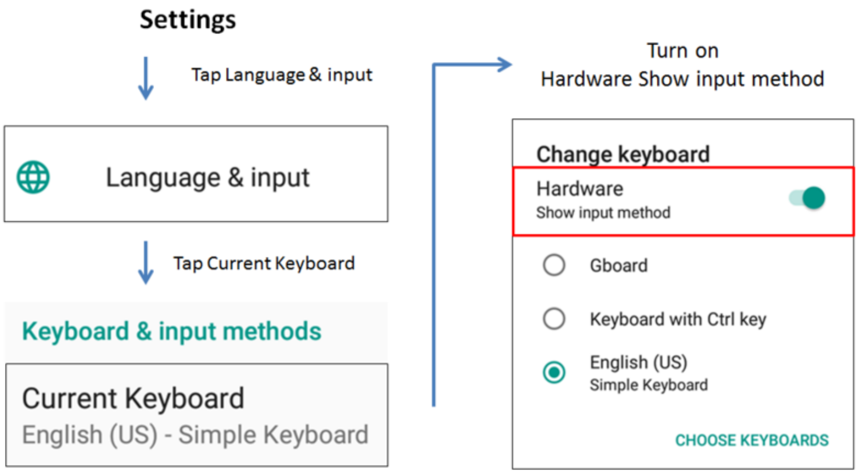

7:49
CircuitPython Editor
When the Circuit Python board is plugged in.
Text input has focus but keyboard not showing.
CTRL-C
CTRL-D
TAB
SEND

Problem: When my CircuitPython board is plugged in to my Android device, the on-screen keyboard does not show, so I cannot type anything into the REPL or code and macro editors.

Explanation: Some Android devices and OS versions have a feature that disables the virtual on-screen keyboard while there is a physical keyboard connected. The idea is that since you're using a hardware keyboard, you won't want the virtual one eating up valuable screen real estate. Since the Circuit Playground Express board is seen as a keyboard, it gets caught by this feature. Luckily, most devices offer a setting to control the behavior.

Solution: First plug in your Circuit Playground Express board (or any hardware keyboard). There must be a hardware keyboard connected for the setting to be visible on many devices. Open the settings and look for a setting called "Language & Input" or "Language & Keyboard" or something similar and tap on it. Look for an entry called "Current Keyboard" or "Default Keyboard" or something similar and tap it. You should see a pop-up dialog allowing you to choose from all of the installed virtual keyboards on your device.

At the top, there should be an on/off switch that controls the behavior of the soft keyboard while a hardware one is connected. It's called "Hardware Show input method" or "Show keyboard on screen" or something similar. Turn it on, and then go back to the CircuitPython Editor application and try again.



Problem: I faced some problem that wasn't listed here and the application does not work for me.

Explanation: This application is in the alpha stage. It is possible it may not work on all Android devices or with all CircuitPython boards.

Solution: Please open an issue on the Github (https://adafru.it/DIF) page and provide as much specific information about the problem you've faced as you can. At the minimum, provide your Android device model, OS version, CircuitPython board, and a detailed description of the problem.

FAQ

Q: Does the application work with other CircuitPython boards beyond the Circuit Playground Express?

A: Yes. The app can work with other boards, but you will need to get the storage mounted as writable. This will likely be a bit less convenient if your board doesn't have an on-board switch. See this guide (https://adafru.it/DIE) about using a board pin as a substitute.

Q: Is the application open source?

A: Yes. The application is hosted on GitHub (https://adafru.it/DIF). Check it out there if you'd like to see how the app works, or lend a hand in development.

Q: The application uses my devices battery kind of fast, is this normal?

A: This varies from device to device, but in general yes powering the CircuitPython board for long periods of time can drain the battery, especially if you use the NeoPixels a lot on high brightness.

Q: Is there an iOS app available to do this?

A: Not that I know of. If you are an iOS developer (or know one) looking to help, please reach out to the project on Github (https://adafru.it/DIF) or on the Hackaday.io project page (https://adafru.it/DIG).