# Hacking a Launchpad Mini MK3 with Python.

Although designed as controller for Ableton and other music software, the Launchpad can be programmed to do a range of other things and today we are going to learn to write some simple programs to write messages, show animations and learn a bit about how displays, from launchpads to your phone, monitor and TV all use the same basic idea to display information.

If you’ve not heard of MIDI or used Python before, don’t worry, if you are interested there is more information below, but you can skip ahead.

**MIDI – Musical Instrument Digital Interface.**

The Launchpad is a Midi device, as such it can connect to, control and be controlled by other midi devices. Midi is an old standard that let 1980s synthesisers talk to each other, regardless of who made them. By today’s standards, it is slow and primitive, but despite many attempts to replace it, lives on and you can still control a 1980’s synthesiser using modern devices such as a launchpad.

Midi works by sending and receiving a series of messages, that act as instructions to a Midi device, or information a midi device sends to a computer or another midi device. The simplest messages are the “Note On” and “Note Off”. Think of pressing a key on a piano keyboard, that is the “note on” message, when you lift your finger off the key, that is a “note off” message. The note on and off messages also contain the pitch of the note, so that a synthesiser knows what the pitch of the note to play should be.

There are other types of message that are more complex, such as a “CC” or “Control Change” message and “System Exclusive” or “sysex” messages, – special messages just for a specific Midi device, in our case the Launchpad Mini MK3. We won’t worry about these now.

Midi is a very deep subject, but all we need to know for now is that the Launchpad is a type of Midi device. It has button or “pads” that act as controllers and underneath each pad there is a colour LED which can be controller to provide feedback and information.

**Python**

Python is a widely used programming language and can run on different types of computers. One of the reasons for its popularity is the huge number of “libraries” that are available. These expand the language to help it do a vast array of tasks, from drawing charts, data science, Machine learning and many others. Libraries are a way of sharing and reusing code, letting us concentrate on what we want to do without getting too deep in technical details.

For us, we are interested in a library that will let Python talk Midi, so we can control our Launchpad. Luckily the Python programming language has several libraries that do this, such as: rtmidi Pygame and Mido. For this workshop we will be using the Rtmidi library, others will work just as well.

The next thing we will need is something that will let us help us write code without having to get too concerned with exactly what Midi messages are needed. This will be done by a special library called pylaunchpad.py, which converts our code into the midi messages that the Launchpad understands.

You may have heard of “OOP”, or “object orientated programming”. This sounds scary, but in our case, our “object” is a real-life Launchpad and we will use Python to make it draw shapes, text and simple animations.

## Setting Up.

The Python environment will be setup in the shop, ready to go, you don’t need to install anything. If you want to explore more code on your own on a different computer, you will need to install rt-midi by using the python command pip:

pip install python-rtmidi

pip install pysimplegui

You might need a parent’s help with this if you are not using Pycharm. If you are a python user and haven’t heard of virtual environments, it is well worth finding out a bit more as it makes installing libraries a lot easier.

Download the pylaunchpad library and examples from:

<https://github.com/joedeller/LPMidi>

Select the “clone or download” button.

If you are familiar with git, you can use the following command:

git clone <https://github.com/joedeller/LPMidi.git>

Git is beyond the scope of this workshop, but it is well worth learning about if you are thinking of developing your programming skills.

## IDLE / Pycharm.

The workshop can be done in Python’s own IDLE program, or in a more advanced program called Pycharm that can make debugging our programs a bit easier. There are plenty of others.

## Talking to the Launchpad.

Once we have connected our Launchpad Mini to our computer using the USB -C cable, we can setup a Python connection using just a few lines of code. If you are familiar with Python, you will know that the “#” character means “everything after this is a comment”, you don’t actually need to type the text that follow the “#”, but commenting your code is a good habit to learn.

import rtmidi # The rtmidi library will let us talk to the Launchpad

import pylaunchpad as lp # Launchpad library, handles input and output

pad = lp.get\_me\_a\_pad() # Scan midi ports for a Launchpad and connect

If you run this, you would see the launchpad turn all the pads off, ready for us to control. Not very exciting, but we have in programming terms, an object called “pad” that we can now use to make things happen on a launchpad.

Each of the pads on the launchpad has a unique number, much like a piano key has a unique note.

Launchpad Mini X & Y co-ordinates

0 1 2 3 4 5 6 7 8

+---+---+---+---+---+---+---+---+ +---+

|0/0| |2/0| | | | | | |8/0| 0

+---+---+---+---+---+---+---+---+ +---+

+---+---+---+---+---+---+---+---+ +---+

|0/1| | | | | | | | | | 1

+---+---+---+---+---+---+---+---+ +---+

| | | | | | | | | | | 2

+---+---+---+---+---+---+---+---+ +---+

|0/3| | | | |5/3| | | |8/3| 3

+---+---+---+---+---+---+---+---+ +---+

| | | | | | | | | | | 4

+---+---+---+---+---+---+---+---+ +---+

|0/5| | | | | | | | | | 5

+---+---+---+---+---+---+---+---+ +---+

| | | | |4/6| | | | |8/6| 6

+---+---+---+---+---+---+---+---+ +---+

| | | | | | | | | | | 7

+---+---+---+---+---+---+---+---+ +---+

|0/8| | | | | | | | |8/8| 8

+---+---+---+---+---+---+---+---+ +---+

The Launchpad buttons, or pads as they are known, each have a number, much like a house address. However, programming them in this way would be quite hard work, so it is easier to think of the launchpad as a grid, like a grid of squared paper.

The top left button is at X, 0, Y, 0 and the bottom right button is at X=8, Y =8. The Novation logo doesn’t have a pad button, but on the mini there is a LED underneath.

Our launchpad library has some code that lets us control a pad by its number or by the X & Y coordinate.

Here is our first Launchpad code. In Pycharm we are going to type this into the file called “my\_code.py”, once you have typed all four lines, then you can run it:

import pylaunchpad as lp  
pad = lp.get\_me\_a\_pad()  
pad.set\_led\_xy\_by\_colour(0,0)  
pad.set\_led\_xy\_by\_colour(8,8)

We can control the colour, but if we don’t tell the launchpad what colour to use, it will choose green.

How do we turn the pads off? If we change our code a little:

import time

import pylaunchpad as lp

pad = lp.get\_me\_a\_pad()

pad.set\_led\_xy\_by\_colour(0,0)

pad.set\_led\_xy\_by\_colour(8,8)

time.sleep(1)

pad.set\_led\_xy\_by\_colour(0,0,’off’)

pad.set\_led\_xy\_by\_colour(8,8,’off’)

In this example, the second set of commands uses a colour of “off” (“black will also work), which turns the pads off. As you might expect, the time.sleep(1) command waits for a second before carrying on.

### Light Emitting Diodes (LEDS) - Red Green and Blue.

Underneath the plastic pads of are 3 very small Light Emitting Diodes, or LEDs, a Red, Green and Blue one. Not only can we turn individual pads, but we can control the three LEDs underneath each pad.

You might be familiar with mixing Primary colours in Art lessons, the Launchpad works in a similar way. By mixing the different amounts of Red, Green & Blue light, we can display a range of colours.

Let’s change our code again:

import time

import pylaunchpad as lp

pad = lp.get\_me\_a\_pad()

pad.set\_led\_xy(0,0,63,0,0)

pad.set\_led\_xy(4,4,0,63,0)

pad.set\_led\_xy(8,8,0,0,63)

This time we are using the **set\_led\_xy(x, y, red, green,blue)** command, which needs the X & Y co-ordinates of a pad, plus how much red, green and blue light to show. In this case, we will turn the first pad on with Red amount 63, green set to 0 and blue set to 0. The middle pad will be green set to 63, red set to 0 and blue set to 0. Finally, we turn the bottom left pad to blue.

If you are feeling adventurous, try adjusting the 3rd, 4th and 5th numbers (parameters) to see what different colours you get. You might notice that numbers above 64 don’t make the pad any brighter.

What about white light? Can you think how we might turn a pad to be white? Try this:

pad.set\_led\_xy(3,4,63,63,63)

*Advanced Topic*.

The pylaunchpad library has a set of commands that can be used to control the launchpad, it is a very simple form of an “***Application Programmers Interface***” or “API”.

The launchpad itself has an API, which details how to control it via lower level midi commands. The Python library is built on top the Launchpad API to make it easier to use. In programming terms, the pylaunchpad.py code “abstracts” the lower level API. The pylaunchpad library can talk to other launchpads, the MK2 and original mini, although that only had Red & Green LEDs, so was much more limited in the colours it could show.

The red, green and blue brightness ranges from 0 – fully off, to 63, maximum brightness. The total shades of colour combinations are 63 \* 63 \* 63, or approximately 250 thousand. Our eyes would have a hard time identifying all the differences, so the practical number of colours we see is somewhat smaller.

## Colour Mixer

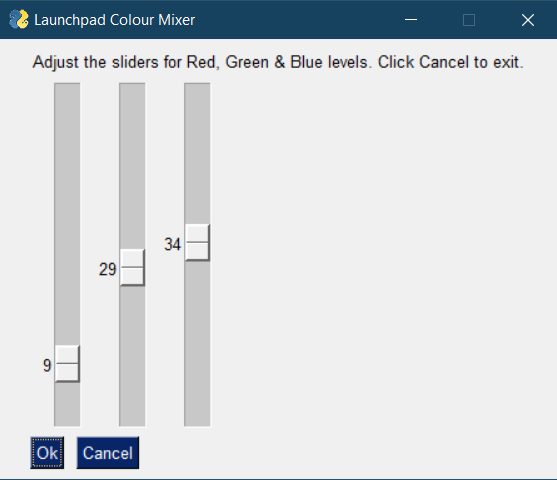
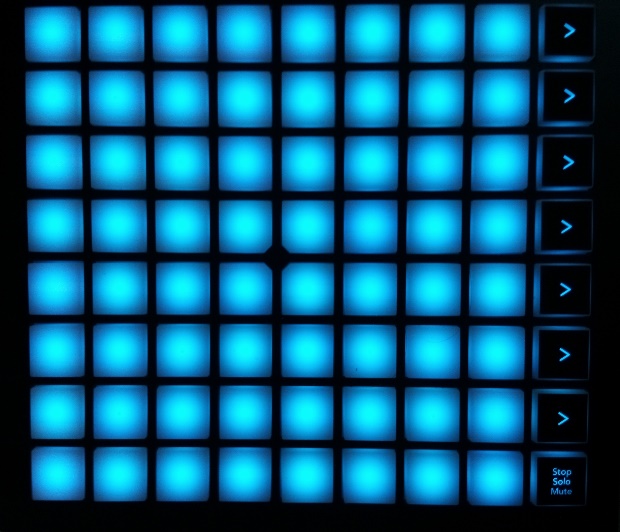
To find a colour we like, we could experiment with adjusting the numbers for red, green and blue, but that would be quite a slow process. Another way is to use some code that allows us to mix the different colours and show us what the numbers for the red, green and blue are. The next set of code calls another small program that does the heavy lifting for us:

import pylaunchpad as lp

from colourmixer import colour\_mix

pad = lp.get\_me\_a\_pad()

colour\_mix(pad)

“Cool blue”, or Red=9, Green=29, Blue=34 as it’s known to its friends.

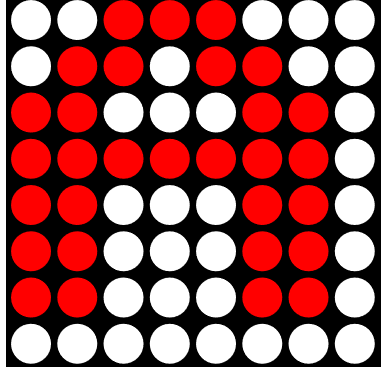
*Advanced Topic*.

If you want to see how the code for making the sliders works, this is in the file colourmixer.py. The code detects when a slider has been moved and sends the colour to all the pads at once. This uses a special midi message which is a bit faster than us doing lots of set\_led\_xy() messages.

## Whole Letters.

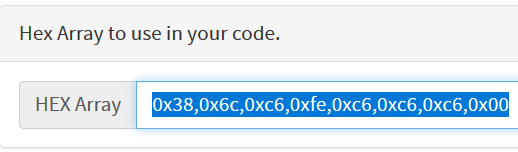
What about displaying text, messages or even shapes? How can we do that?

What about the letter “A”?

There are many web tools to help, but sites like GurgleApps have a set of characters and shapes already drawn for us:

<https://gurgleapps.com/tools/matrix>

When we select a shape or letter from GurgleApps, it gives us a set of codes, which are in Hexadecimal (“hex” for short), or base 16



### Hex, decimal and binary.

How do we get from the letter “A” to the list of Hex numbers? You might know that unlike humans that count in units, tens, hundreds and so on, computers count in binary, using “bits” which in modern computers are grouped in eights, to form a “byte”. This is very convenient for our launchpad code, as we have 8 grey pads on each row, which we can treat as a byte of computer memory. You can skip this as it isn’t essential to understand for now, but as a coder, binary and hexadecimal will become very familiar to you.

Comparing columns in Decimal (base 10), Binary (base 2) and Hexadecimal (base 16)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Decimal | 10,000,000 | 1,000,000 | 100,000 | 10,000 | 1000 | 100 | 10 | Units |
| Binary | 128 | 64 | 32 | 16 | 8 | 4 | 2 | Units |
| Hex | 4294967296 | 268435456 | 16777216 | 1048576 | 65536 | 256 | 16 | Units |

If we convert our Hex numbers to decimal, we get a list of 56,108, 198, 254,198,198,198,0. Still not a lot of sense, but if we now convert to binary:

|  |  |  |
| --- | --- | --- |
| Hex | Decimal | Binary |
| 0x38 | 56 | 00**111**000 |
| 0x6C | 108 | 0**11**0**11**00 |
| 0xC6 | 198 | **11**000**11**0 |
| 0xFE | 254 | **1111111**0 |
| 0xC6 | 198 | **11**000**11**0 |
| 0xC6 | 198 | **11**000**11**0 |
| 0xC6 | 198 | **11**000**11**0 |
| 0x00 | 0 | 00000000 |

There’s our letter “A”. Why do we use hex? One reason is that it takes less type to represent large numbers in Hex compared do decimal and a lot less that typing numbers out in binary.

*Advanced topic:*

In computer terms, the largest number we can store in a byte of computer memory, 8 bits, is 128+64+32+16+8+4+2+1, which is 255, or “FF” in hex. To store bigger numbers, we need to use more than one byte. Two bytes can store the number 65535 or “FFFF” in hex, 4 bytes can store the number 4,294,967,296, which is more commonly known as “Four gigabytes”, which in hex is “FFFF FFFF FFFF FFFF”

## Displaying a letter on the launchpad.

To show the letter A, we can use the **set\_led\_xy\_by\_colour()** command, starting at 0,0 and looking at each bit, one row at a time, if there is a “1” we light the pad up, if there is a “0” we turn the pad off. We ***could*** do that, but the whole purpose of a library is to have someone else do the heavy lifting for us:

import pylaunchpad as lp

pad = lp.get\_me\_a\_pad()

pad.draw\_letter("A")

You can look inside the pylaunchpad library to see how the code works, you might even figure out a better way of coding it 😊

What about a different colour?

import pylaunchpad as lp

pad = lp.get\_me\_a\_pad()

pad.draw\_colour = “green”

pad.draw\_letter("A")

What about a whole message?

import pylaunchpad as lp

pad = lp.get\_me\_a\_pad()

pad.scroll\_message(“Hello!”)

What about shapes?

A set of data from from GurgleApps has already been stored into a python file called “bitmaps.py”.

import pylaunchpad as lp

import bitmaps as bmp

pad = lp.get\_me\_a\_pad()

pad.draw\_colour = "green"

pad.draw\_char(bmp.invader\_one)

The idea is the same, except this time a python dictionary is used, it’s not easy to describe a space invader using letters. Again, you can look inside bitmaps.py to see what shapes are available.

For now, we are dealing with single colour characters, but the idea extends to full colour characters, except that instead of 8 hex numbers, we will have 24, 8 for the red, 8 for the green and 8 for the blue. You might have heard of the term “24 bit colour”, remember 8 bits in a byte, so a red, green & blue pixel needs 8 \* 3, or 24 bits to display.

*Advanced Topic*

If you want to add another shape from a site like gugleapps, then have a look at the bitmaps.py file. You should be able to figure out how to add a new entry. Remember the names – invader\_one, pac\_one – are up to you. If you copy the data exactly, you will be able to use your shape.

## Scrolling.

The pylaunchpad library can be used to scroll on characters:

import pylaunchpad as lp

import bitmaps as bmp

pad = lp.get\_me\_a\_pad()

pad.draw\_colour = "green"

pad.scroll\_on\_right(bmp.invader\_one)

As you might expect, there is a scroll left:

pad.scroll\_on\_left(bmp.invader\_one)

What about scrolling a character on and off again or animating a character?

*Advanced Topic*

The Launchpad can scroll text itself using special midi messages, but for now the library uses its own code. It can also make pads flash and fade. More details can be found in the Programmers reference guide.

## Animating characters.

Animation is made possible by using a series of “frames”, where each frame contains a single picture. If we change what is drawn fast enough, our eyes are fooled into seeing a moving image, rather than a series of images. The most basic animation consists of two frames, or two single pictures.

The bitmaps.py has two “pac man” shapes:

pac\_one = [0x3c, 0x7e, 0xdf, 0xff, 0xf0, 0xff, 0x7e, 0x3c]

pac\_two = [0x3c, 0x7e, 0xdc, 0xf8, 0xf8, 0xfc, 0x7e, 0x3c]

They are mostly the same, except one has a slightly open mouth. Smooth animation often has very small differences between each frame, notice the differences in the numbers in the two lists.

import pylaunchpad as lp

import bitmaps as bmp

pad = lp.get\_me\_a\_pad()

pad.draw\_colour = "red"

pad.scroll\_frames\_right([bmp.pac\_one, bmp.pac\_two])

The frames are displayed in order, then repeated until the animation has fully scrolled off.

The speed of the animation can be controlled by setting:

pad.delay\_time(seconds)

The default is 0.1 seconds, so to scroll twice as fast, add the following line after the red colour:

pad.draw\_colour = “red”

pad.delay\_time = 0.05

The scroll\_frames\_right function needs a list of frames to draw. More complex animations will use a longer list of frames. You could also use:

pad.scroll\_frames\_right([bmp.club, bmp.club])

This just scrolls a single frame across the launchpad.

## Full colour animations.

This is quite a complex topic, so once again there is a small library to help us out for some basic animations, which were created using another piece of software from BlinkinLabs that was designed to control LED matrix on microcontrollers, that I converted the output into something the launchpad could use.

import pylaunchpad as lp

import show\_patterns as patterns

pad = lp.get\_me\_a\_pad()

patterns.show\_file(pad, "fireworks.csv")

The fireworks.csv is the name of a file that is part of the Launchpad library, stored in the Patterns directory. It contains many frames of colour animation. There are several more examples.

To see all the animations, try this:

import pylaunchpad as lp

import show\_patterns as patterns

pad = lp.get\_me\_a\_pad()

patterns.show\_all(pad)

Creating animations can be quite a tricky process and there are often quite sophisticated tools that do a lot of the hard work for us, but ultimately, they generate a file that has a lot of data that is just a series of ones and zeros. You can open one of the CSV files in a notepad or spreadsheet, but they are not particularly easy for humans to decode.

*Advanced Topic – How does it scroll?*

You might be wondering how the launchpad scrolls a character and if you were to look inside the pylaunchpad library, you will come across what are called “bitwise operators”.

If we wanted to scroll a single pixel from left to right, we could use a loop (**note the 5th - 7th line of code have 4 spaces in front of them**):

import pylaunchpad as lp

import time

pad = lp.get\_me\_a\_pad()

for x in range (0, 9):

pad.set\_led\_xy\_by\_colour(x, 0, "red")

time.sleep(0.1)

pad.set\_led\_xy\_by\_colour(x, 0, "off")

That would work just fine, but if we wanted to move more than one pixel, in fact a whole character’s worth of 64 pixels it starts getting a little trickier.

This is where the magic of binary comes in. If you remember the “A” and the 8 numbers that represented a single row of data

|  |  |
| --- | --- |
| 56 | 00111000 |
| 108 | 01101100 |
| 198 | 11000110 |
| 254 | 11111110 |
| 198 | 11000110 |
| 198 | 11000110 |
| 198 | 11000110 |
| 0 | 00000000 |

What if we divided the numbers by two?

|  |  |
| --- | --- |
| 28 | 00011100 |
| 54 | 00110110 |
| 99 | 01100011 |
| 127 | 01111111 |
| 99 | 01100011 |
| 99 | 01100011 |
| 99 | 01100011 |
| 0 | 00000000 |

Notice how the letter has shifted one column to the left?

What if we divide by two again?

|  |  |
| --- | --- |
| 14 | 0000**111**0 |
| 27 | 000**11**0**11** |
| 49 (rounded down) | 00**11**000**1** |
| 63 (rounded down) | 00**111111** |
| 49 | 00**11**000**1** |
| 49 | 00**11**000**1** |
| 49 | 00**11**000**1** |
| 0 | 00000000 |

If we keep on dividing by two eventually, we would be left with nothing but zeros, our character will have completed scrolled off.

To scroll from right to left, we multiply by two:

|  |  |
| --- | --- |
| 56 | 00**111**000 |
| 108 | 0**11**0**11**00 |
| 198 | **11**000**11**0 |
| 254 | **1111111**0 |
| 198 | **11**000**11**0 |
| 198 | **11**000**11**0 |
| 198 | **11**000**11**0 |
| 0 | 00000000 |

|  |  |
| --- | --- |
| 112 | 0**111**0000 |
| 216 | **11**0**11**000 |
| 140 (overflow, so 198\*2 -256) | **1**000**11**00 |
| 252(254 \* 2 = 508, then take 256) | **111111**00 |
| 140 | **1**000**11**00 |
| 140 | **1**000**11**00 |
| 140 | **1**000**11**00 |
| 0 | 00000000 |

### Button presses.

So far, we have looked at setting the colour of pads, but what about pressing pads, how do we find out what buttons are being pressed? How does the launchpad control other devices and software?

## Drawing App.

The following code will light each pressed pad a random colour. Press the “Stop/Solo/Mute” button in the lower right-hand corner of the MK3 (other Launchpads press the bottom right pad) to exit.

pad.reset()

pad.in\_ports.set\_callback(pad.random\_paint)

while True:

if (pad.last\_x >= pad.max\_x) and (pad.last\_y == 8):

break

time.sleep(.4)

pad.in\_ports.cancel\_callback()

pad.reset()

Whilst this is pretty, it’s not very useful for drawing a picture.

**NOTE: Watch the spaces !!** If you haven’t used Python before, you might wonder why some code listings have lines that are “indented”, compared to others. Python uses groups of 4 spaces (some people use tabs) to indent the code. This might seem strange at first, but you will get used to it.

### *Advanced Topic -Polling and Call back.*

There a couple of different ways of checking for a user input when writing a program. The first is called polling, which regularly scans for an input, the second is using a “call back”. If you want to know if someone is at the front door, you could keep checking to see if anyone is there every minute or so. This would work but would stop you doing anything else before having to check again. Another way is to install a doorbell and wait for it to call you whilst you are doing something else.

In this instance, our main program isn’t doing much apart from sleeping and checking if the bottom right pad has been pressed. The code that does the work is inside our pylaunchpad library. Whilst the main program is sleeping, it is getting notifications that a button has been pressed at a specific X,Y co-ordinate, picking a random colour and then turning the pad on at the same location that was pressed.

There is also another piece of code in our library that handles call backs, it will run for 20 seconds before the call back is turned off:

import pylaunchpad as lp

import time

pad = lp.get\_me\_a\_pad()

pad.in\_ports.set\_callback(pad.midi\_in\_cb)

for loop in range(20):

time.sleep(1)

pad.in\_ports.cancel\_callback()

This time the X & Y coordinates of the button we pressed are displayed.

## Mini Drawing App.

With a bit more code, we can select the colour we want to draw. The colours have been predefined in the library as a list of red green and blue values. You can change them if you wish, look in the pylaunchpad.py library for the line starting “self.painter\_palette=”, you will see a list of Red, Green and Blue values.

import pylaunchpad as lp

import time

pad = lp.get\_me\_a\_pad()

pad.setup\_painter\_colours()

pad.in\_ports.set\_callback(pad.paint\_app)

while True:

if pad.last\_x >= pad.max\_x and pad.last\_y == 8:

break

time.sleep(.4)

pad.in\_ports.cancel\_callback()

lp.save\_frame(pad.painter\_frame)

The first bit of code sets the top row to a list of colours. The start colour is white. To erase a pad, press the “User” (top right) button, which is in effect, “off”

*Advanced Topic.*

The “pad.max\_x” is there because some Launchpads have an extra column of buttons, so the code has to be aware that the bottom right button might be at a different X,Y than others

You might notice that the last line saves our drawing. If you are using Pycharm you might notice a file called “my\_picture.csv” appear. Try this code, you can disconnect and reconnect the pad if you like to prove there’s no cheating!

import pylaunchpad as lp

pad = lp.get\_me\_a\_pad()

lp.load\_frame(pad)

## More quick bits of code.

## Christmas Tree

import pylaunchpad as lp

import time

import snow\_tree as tree

pad = lp.get\_me\_a\_pad()

pad.reset()

tree.tree(pad)

for loops in range(10):

tree.snow\_tree(pad)

## Spinning Ghost

import pylaunchpad as lp

from rotate\_bmp import spin\_ghost

pad = lp.get\_me\_a\_pad()

spin\_ghost(pad)

One advantage about having pictures in binary format is that we can manipulate them with relative ease.

## Fancy lighting.

import arduinoPort as ap

pad = lp.get\_me\_a\_pad()

ap.rainbow\_pad(pad)

ap.theatre\_chase(pad,63, 12, 55)

ap.rainbow\_cycle(pad)

ap.theater\_chase\_rainbow(pad)

## Wrapping Up.

The launchpad is a very versatile device and we have only scratched the surface on coding it. Many electronic devices that have displays, controls / buttons use very similar ideas that are easy to take for granted. From the screen on your phone, to the HD TV, controlling individual red, green, blue lights that make up a single a pixel, that in turn make up rows and columns that form pictures, there is some code whose job is just to light them up, or send us X and Y coordinates of where you pressed, so that another piece of code can take some action.

Software like Ableton is really just a much more complex version of what we have done today, but the basic ideas are the same.

## What’s Next?

Try looking inside some of the python .py files and see if you can follow the code. You might find that you can find better ways of writing the code. If so, submit a pull request!