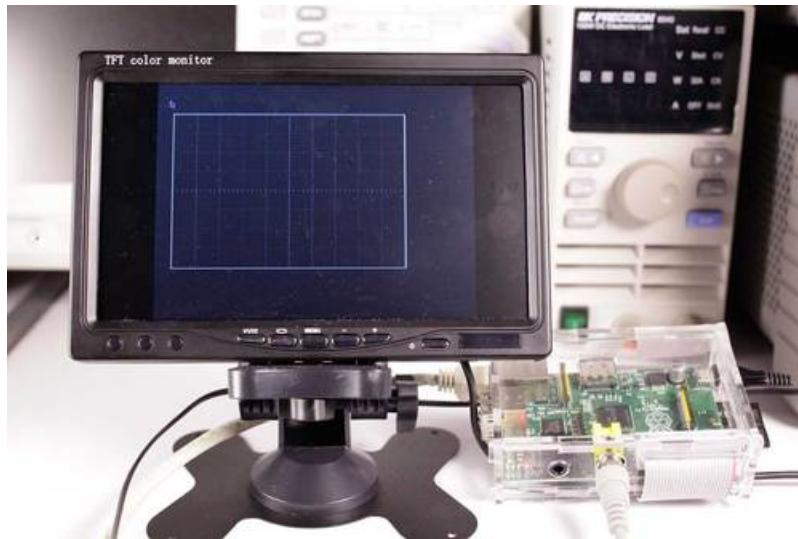


Pi Video Output Using pygame

Created by Kevin Townsend

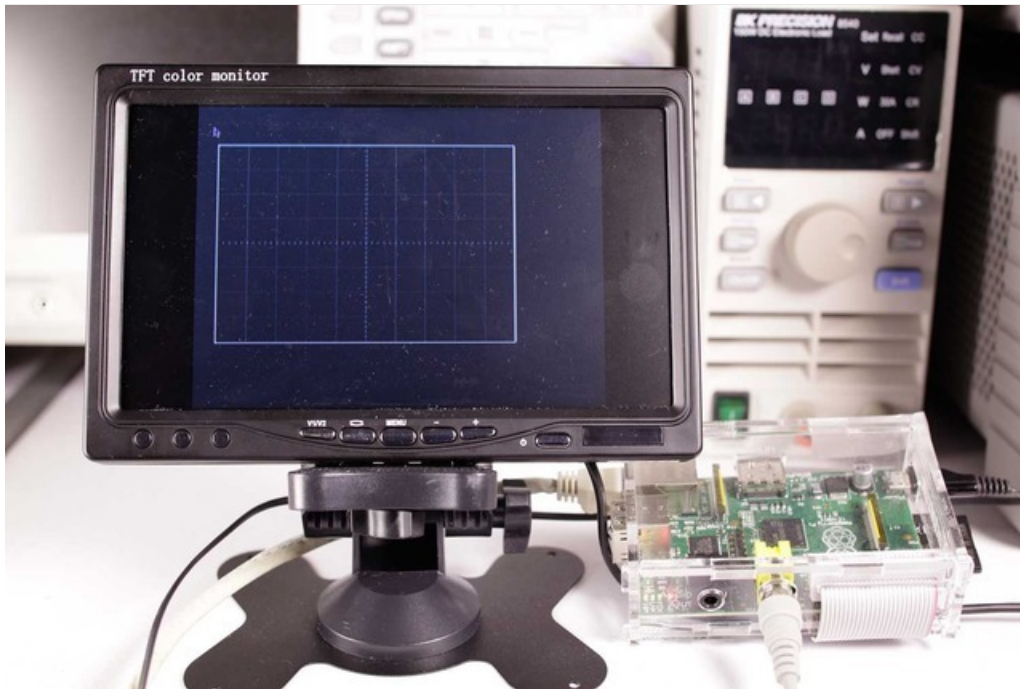


Last updated on 2018-08-22 03:32:19 PM UTC

Guide Contents

Guide Contents	2
Overview	3
What You'll Need	3
Development Tools	3
Getting Started	5
About pygame	5
Creating a New Project in the WebIDE	5
Pointing Pygame to the Framebuffer	7
Huh ... What is all this?	8
How Do I Run It?	8
Drawing Basics	11
The Screen Object	11
Accessing the Screen	11
Pygame Drawing Functions	13
Drawing a Graticule	13
Adding Text	14
Adding Images	15
Wrapping it all Up + Animation	16

Overview



One of the biggest advantages of the Raspberry Pi as a development platform is the easy access you have to a lot of mature, well-designed SW stacks and libraries, making it relatively trivial to perform tasks that would be very complex or time-consuming to implement on deeply embedded mcu-based systems.

One of the areas where this is particularly true is with graphics and user interfaces. The HW requirements are fairly high to be able to work with large displays (larger than say 320x240 pixels), and even if you can meet the timing requirements -- a 7" 800x480 display probably requires ~40MHz for the pixel clock -- there are very few open source or inexpensive graphics frameworks out there to render the kinds of modern UIs people expect today.

The Pi really stands out here since it's trivial to render complex UIs, taking advantage of modern features like Unicode text and complex scripts, and being able to use different fonts without having to worry about memory and rendering time (it can take a couple hundred KB of SRAM to render a TTF on an embedded system, and the libraries to interpret them are both large and complex), etc. You can also easily display the graphics on any inexpensive composite television or HDMI monitor, which is amazing for a \$35 board ... more than the cost of many LCDs!

This tutorial will show you one way to get started drawing graphics on an external display using [pygame](https://adafru.it/aQu) (<https://adafru.it/aQu>) along with [Adafruit's WebIDE](https://adafru.it/aQv) (<https://adafru.it/aQv>), and almost any external display ([several are available here](https://adafru.it/aQw) (<https://adafru.it/aQw>)).

What You'll Need

- A [Raspberry Pi Model B](http://adafru.it/998) (<http://adafru.it/998>)
- [Occidentalis v0.2](https://adafru.it/aQx) (<https://adafru.it/aQx>) or higher
- Some sort of external display (We're using a [7" NTSC/PAL TFT Display](http://adafru.it/947) (<http://adafru.it/947>))

Development Tools

This tutorial uses [Adafruit's WebIDE](https://adafru.it/aQv) (<https://adafru.it/aQv>). It's an ideal development environment to work with external displays, since you can display your debug output and keep the shell visible while the display contents are changed on the remote display, and easily upload files (images, fonts, etc.) to the Pi.

You could also work via SSH, but as you'll see later in the tutorial this is less convenient since you'll need at least two sessions, it's more cumbersome to transfer files, and you'll probably find the text editor in the WebIDE more natural to use than vi or nano if you aren't already an experienced Linux user.

If you're not already using WebIDE -- have a look at our easy to follow learning guides below:

- [Occidentalis v0.2 \(https://adafru.it/aQx\)](https://adafru.it/aQx)
- [Raspberry Pi WebIDE \(https://adafru.it/aQv\)](https://adafru.it/aQv)

Getting Started

About pygame

Pygame is a game development platform for python that handles a lot of the basic plumbing you need to implement in basic 2D or tile-based games.



Despite the name, though, it can be used for much more than gaming, and it includes many features that lend it to designing complex and easy to update UIs without having to get into all the details of X-Windows or some of the more complex UI frameworks like GTK, etc. Given the limited resources on the Pi (for an embedded Linux system), it can actually be advantageous to develop UIs with something like pygame since you have very little additional overhead compared to X-Windows, etc., and the learning curve is pretty flat.

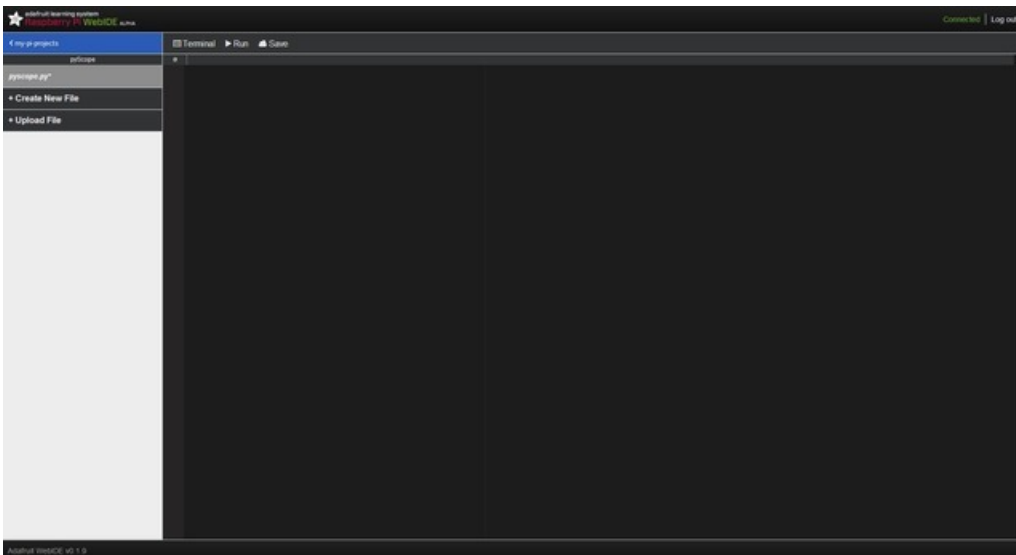
There are also quite a few tutorials (and books!) out there to help you get started with pygame, with some of the following being a good starting point:

- [Official pygame documentation \(https://adafru.it/aQy\)](https://adafru.it/aQy)
- [pygame tutorials wiki \(https://adafru.it/aQz\)](https://adafru.it/aQz)
- A useful tutorial on [fonts and sprites/images](#)

Creating a New Project in the WebIDE

Be sure to see our [Using the WebIDE \(https://adafru.it/aQA\)](https://adafru.it/aQA) tutorial for info on how to create a new file in the WebIDE.

Once you're up and running with the WebIDE, create a new project folder in my-pi-projects named **pyscope**!. Once you're inside this folder, create a file named **'pyscope.py'**



Once this file is create, you're ready to get started with pygame and your display ...

Pointing Pygame to the Framebuffer

To render any sort of graphics from the console, we first need to point pygame to the underlying framebuffer used by Linux. This is probably the most error-prone operation, but the following code should handle this gracefully and report any errors if something does go wrong.

Enter the following code into your new project, and save the file via the 'Save' button in the top-menu, or just enter CTRL+S:

```
import os
import pygame
import time
import random

class pyscope :
    screen = None;

    def __init__(self):
        "Initializes a new pygame screen using the framebuffer"
        # Based on "Python GUI in Linux frame buffer"
        # http://www.karoltomala.com/blog/?p=679
        disp_no = os.getenv("DISPLAY")
        if disp_no:
            print "I'm running under X display = {}".format(disp_no)

        # Check which frame buffer drivers are available
        # Start with fbcon since directfb hangs with composite output
        drivers = ['fbcon', 'directfb', 'svgalib']
        found = False
        for driver in drivers:
            # Make sure that SDL_VIDEODRIVER is set
            if not os.getenv('SDL_VIDEODRIVER'):
                os.putenv('SDL_VIDEODRIVER', driver)
            try:
                pygame.display.init()
            except pygame.error:
                print 'Driver: {} failed.'.format(driver)
                continue
            found = True
            break

        if not found:
            raise Exception('No suitable video driver found!')

        size = (pygame.display.Info().current_w, pygame.display.Info().current_h)
        print "Framebuffer size: %d x %d" % (size[0], size[1])
        self.screen = pygame.display.set_mode(size, pygame.FULLSCREEN)
        # Clear the screen to start
        self.screen.fill((0, 0, 0))
        # Initialise font support
        pygame.font.init()
        # Render the screen
        pygame.display.update()

    def __del__(self):
        "Destructor to make sure pygame shuts down, etc."

    def test(self):
```

```
# Fill the screen with red (255, 0, 0)
red = (255, 0, 0)
self.screen.fill(red)
# Update the display
pygame.display.update()

# Create an instance of the PyScope class
scope = pyscope()
scope.test()
time.sleep(10)
```

Huh ... What is all this?

Don't worry if you don't understand every little bit of the code above. We've provided the code in a way that you simply need to create a new instance of the 'pyscope' class, and the low-level framebuffer implementation should be taken care of. How do you know it works? Let's try it out!

There are three lines of code at the bottom that are important:

```
# Create an instance of the PyScope class
scope = pyscope()
scope.test()
time.sleep(10)
```

The first line (after the comment) instantiates a new pyscope object named scope. As soon as this line is executed, the framebuffer will be configured, or any eventual error messages will be displayed if there were any problems.

The second line simply calls a function named 'test' that is in the example code we entered earlier. This will simply fill the screen with the color red.

The third line causes the program to sleep for 10 seconds before exiting. This is provided simply to give us a brief delay before the display returns to the shell, restoring whatever was displayed before this program was run.

How Do I Run It?

If you've already run off and clicked the 'Run' button, you might have noticed the following error:


```
disp_no = os.getenv("DISPLAY")
if disp_no:
    print "I'm running under X display"

# Check which frame buffer drivers are
# Start with fbcon since directfb hang
drivers = ['fbcon', 'directfb', 'svgalib']
found = False
for driver in drivers:
    # Make sure that SDL_VIDEODRIVER is
    if not os.getenv('SDL_VIDEODRIVER'):
        os.putenv('SDL_VIDEODRIVER', driver)
    try:
        pygame.display.init()
    except pygame.error:
        print 'Driver: {0} failed.'.format(driver)
        continue
    found = True
    break

if not found:
    raise Exception('No suitable video driver found!')

size = (pygame.display.Info().current_w, pygame.display.Info().current_h)
print "Framebuffer size: %d x %d" % (size[0], size[1])
self.screen = pygame.display.set_mode((size[0], size[1]))
# Clear the screen to start
self.screen.fill((0, 0, 0))
# Initialise font support
pygame.font.init()
# Render the screen
pygame.display.update()
```

```
pi@raspberrypi ~/Adafruit/WebIDE/repositories/my-pi-projects/pyScope $ python pscope.py
Driver: fbcon failed.
commandline read: python
commandline read: pscope.py

-----| DirectFB 1.2.10 |-----
(c) 2001-2008 The world wide DirectFB Open Source Community
(c) 2000-2004 Convergence (integrated media) GmbH
-----

(*) DirectFB/Core: Single Application Core. (2012-06-01 00:48)
(!) DirectFB/core/vt: Error opening '/dev/tty0!'
--> Permission denied
(!) DirectFB/Core: Could not initialize 'system_core' core!
--> Initialization error!
Driver: directfb failed.
Driver: svgalib failed.
Traceback (most recent call last):
  File "pscope.py", line 57, in <module>
    scope = pscope()
  File "pscope.py", line 34, in __init__
    raise Exception('No suitable video driver found!')
Exception: No suitable video driver found!
pi@raspberrypi ~/Adafruit/WebIDE/repositories/my-pi-projects/pyScope $
```

You get this error because the application tries to access the framebuffer using first fbcon, then directfb if that fails and finally svgalib. They all fail for one important reason:

You need root access to modify the framebuffer!

Future versions of the WebIDE will add the ability to run as root, but if you're running an older version that doesn't already include this functionality, there's an easy workaround. To give your program access to the framebuffer, simply click the Terminal icon at the top of the text editor, and from the shell that pops up enter:

```
sudo python pscope.py
```

If all goes well, you should see something like the following in your terminal window:

```
bash
pi@raspberrypi ~/Adafruit/WebIDE/repositories/my-pi-projects/pyScope $ sudo pyth
on pyscope.py
Framebuffer size: 656 x 416
pi@raspberrypi ~/Adafruit/WebIDE/repositories/my-pi-projects/pyScope $
```

And during 10 seconds, your display should look something like this:



Drawing Basics

The Screen Object

While the pygame API and documentation is quite clear (see the [pygame.draw \(https://adafru.it/aQB\)](https://adafru.it/aQB) documentation, for example), most of the pygame drawing operations take place on a **screen**, which is rendered on a specific [display \(https://adafru.it/aQC\)](https://adafru.it/aQC).

The way that the code we entered in the previous page works is it initializes the display in the `__init__` function, and it then allows us to access a field named `'screen'`, where all of the actual drawing and graphics work is done in pygame (you pass a reference to this screen to most drawing functions).

You can see how this works by looking at the 'test' function we added and called:

```
# Fill the screen with red (255, 0, 0)
red = (255, 0, 0)
self.screen.fill(red)
# Update the display
pygame.display.update()
```

This code references the 'screen' provided in the class (`self.screen` since we're referencing it from inside the class), and calls the fill method. Once we are done all of our drawing, we tell pygame to update the entire display to take into account these changes, which is done with:

```
pygame.display.update()
```

This is actually an important point about pygame:

No drawing will take place on the display until `pygame.display.update()` is called!

This is both an intentional and an intelligent choice. By doing all of your drawing code at once, and only updating the screen when the drawing is complete, you can benefit from something called double-buffering, which makes all of your drawing appear instant, rather than seeing controls get rendered one at a time in a sequential, sluggish-feeling way.

Accessing the Screen

Since we made a wrapper class called `pyscope` which takes care of the low-level framebuffer initialisation, etc., how do you access the screen object from outside the class? It's easy ... any time you want to do any drawing, you just need to access the screen as follows:

```
# Create an instance of the PyScope class
scope = pyscope()
# Fill the screen with yellow
scope.screen.fill((255, 255, 0))
# Update the display
pygame.display.update()

# Wait 10 seconds
time.sleep(10)
```

You can test this out by placing it at the bottom of your file with no indents, and you should see a yellow screen for ten

seconds when you run it.

Pygame Drawing Functions

Pygame includes a fairly rich collection of drawing objects and functions, making it easy to draw basic shapes, render text, display images, etc. The best way to learn the API is to start exploring it, look at examples on the web, and [read the documentation \(https://adafru.it/aQy\)](https://adafru.it/aQy), but the following sets of functions are probably the most useful if you're just getting started:

- `pygame.draw` (<https://adafru.it/aQB>) (basic drawing primitives)
- `pygame.font` (<https://adafru.it/aQD>) (text rendering)
- `pygame.image` (<https://adafru.it/aQE>) (load and display .gifs, .jpgs, .pngs, etc.)

Drawing a Graticule

As an example, we'll use some basic drawing primitives to render something resembling a graticule on an oscilloscope. Add the following function to your `pyscope` class (with proper indentation to align with the rest of the class):

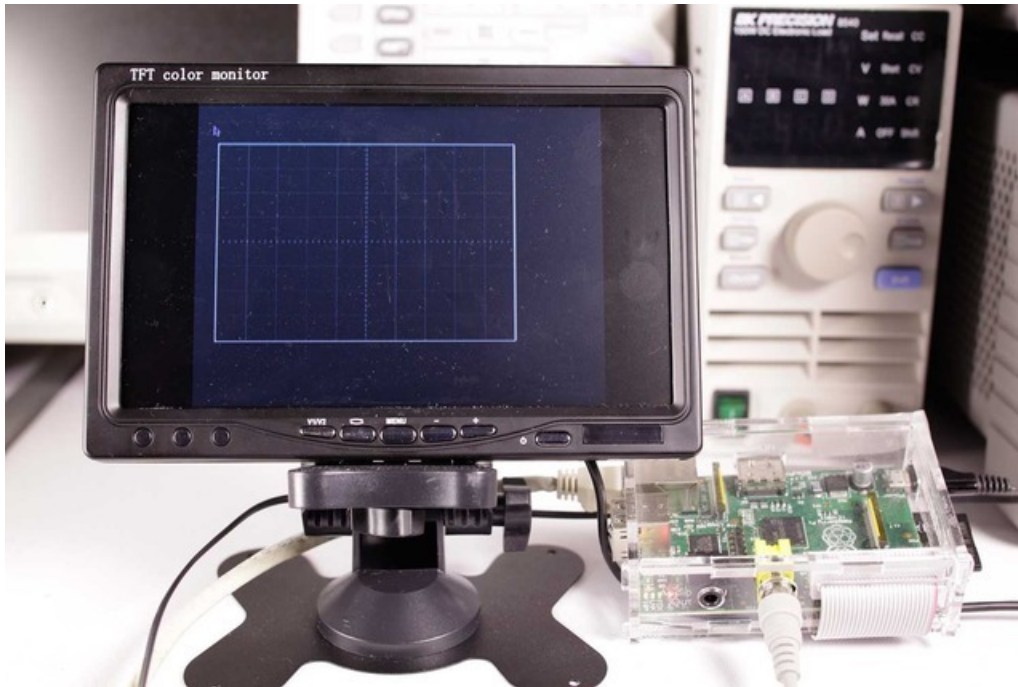
```
def drawGraticule(self):
    "Renders an empty graticule"
    # The graticule is divided into 10 columns x 8 rows
    # Each cell is 50x40 pixels large, with 5 subdivisions per
    # cell, meaning 10x8 pixels each. Subdivision lines are
    # displayed on the central X and Y axis
    # Active area = 10,30 to 510,350 (500x320 pixels)
    borderColor = (255, 255, 255)
    lineColor = (64, 64, 64)
    subDividerColor = (128, 128, 128)
    # Outer border: 2 pixels wide
    pygame.draw.rect(self.screen, borderColor, (8,28,504,324), 2)
    # Horizontal lines (40 pixels apart)
    for i in range(0, 7):
        y = 70+i*40
        pygame.draw.line(self.screen, lineColor, (10, y), (510, y))
    # Vertical lines (50 pixels apart)
    for i in range(0, 9):
        x = 60+i*50
        pygame.draw.line(self.screen, lineColor, (x, 30), (x, 350))
    # Vertical sub-divisions (8 pixels apart)
    for i in range(1, 40):
        y = 30+i*8
        pygame.draw.line(self.screen, subDividerColor, (258, y), (262, y))
    # Horizontal sub-divisions (10 pixels apart)
    for i in range(1, 50):
        x = 10+i*10
        pygame.draw.line(self.screen, subDividerColor, (x, 188), (x, 192))
```

This will draw a 500x320 pixel graticule, with an outer 2 pixel border, and divisions for 10 columns, and 8 rows. You can render the graticule with the following code at the bottom of your class:

```
# Create an instance of the PyScope class
scope = pyscope()
# Add the graticule
scope.drawGraticule()
# Update the display
pygame.display.update()

# Wait 10 seconds
time.sleep(10)
```

That you give you something similar to the following (you may need to tweak the colors, etc., depending on the display you are using):



Adding Text

Adding text is relatively straight-forward using the [pygame.font](https://adafru.it/aQD) (<https://adafru.it/aQD>) functions/objects. If you wish to load specific fonts (you can load fonts from files, etc.) you should consult the documentation, but as a basic example of using the default system font you simply need to enter something similar to the following code:

```

# Create an instance of the PyScope class
scope = pyscope()

# Get a reference to the system font, size 30
font = pygame.font.Font(None, 30)
# Render some white text (pyScope 0.1) onto text_surface
text_surface = font.render('pyScope (%s)' % "0.1",
True, (255, 255, 255)) # White text
# Blit the text at 10, 0
scope.screen.blit(text_surface, (10, 0))
# Update the display
pygame.display.update()

# Wait 10 seconds
time.sleep(10)

```

Note: In order to use the font functions and objects, you need to call 'pygame.font.init()' somewhere in your code. This was already added to the init sequence of the class we created earlier, but it worth mentioning here.

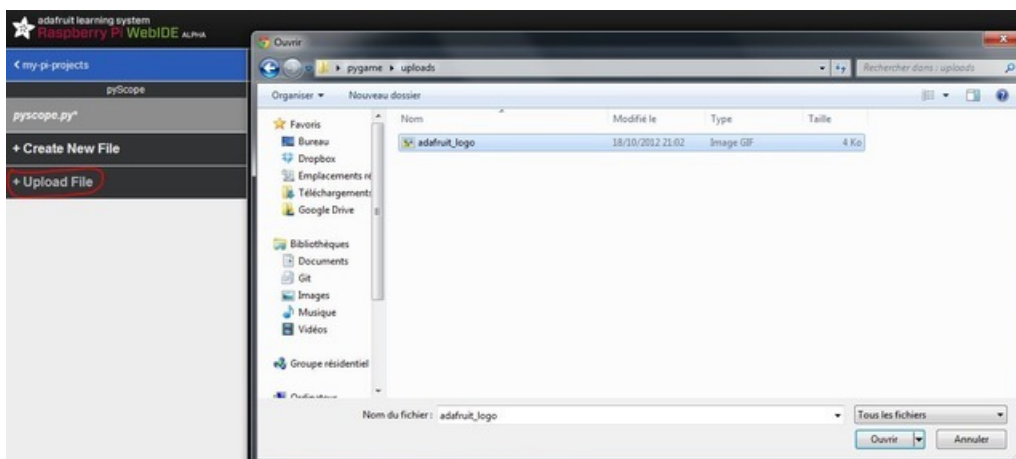
Adding Images

It's just as easy to add images using the [pygame.image](https://adafru.it/aQE) functions/objects.

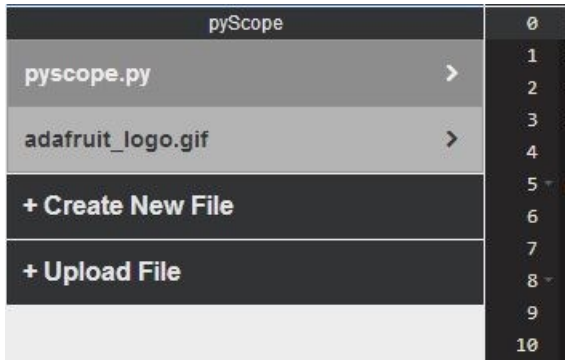
To test this out, upload an image via the WebIDE using the 'Upload File' button on the left-hand menu, and upload the following image:



Select the appropriate image with the popup file dialogue ...



... and the image should show up in your file explorer with whatever name you gave it:



Now you simply need to reference the local file name with the following code:

```
# Create an instance of the PyScope class
scope = pyscope()

# Render the Adafruit logo at 10,360
logo = pygame.image.load('adafruit_logo.gif').convert()
scope.screen.blit(logo, (10, 10))
pygame.display.update()

# Wait 10 seconds
time.sleep(10)
```

Wrapping it all Up + Animation

If you need to animate anything, the secret is simply to call 'pygame.display.update()' at the appropriate moment.

An example using all of the above techniques can be seen in the following complete class that you can simply copy and paste into your pyscope.py file:

```
import os
import pygame
import time
import random

class pyscope :
    screen = None;

    def __init__(self):
        "Initializes a new pygame screen using the framebuffer"
        # Based on "Python GUI in Linux frame buffer"
        # http://www.karoltomala.com/blog/?p=679
        disp_no = os.getenv("DISPLAY")
        if disp_no:
            print "I'm running under X display = {}".format(disp_no)

        # Check which frame buffer drivers are available
        # Start with fbcon since directfb hangs with composite output
        drivers = ['fbcon', 'directfb', 'svgalib']
        found = False
        for driver in drivers:
            # Make sure that SDL_VIDEODRIVER is set
            if not os.getenv('SDL_VIDEODRIVER'):
                os.putenv('SDL_VIDEODRIVER', driver)
            +rv:
```



```

    try:
        pygame.display.init()
    except pygame.error:
        print 'Driver: {0} failed.'.format(driver)
        continue
    found = True
    break

if not found:
    raise Exception('No suitable video driver found!')

size = (pygame.display.Info().current_w, pygame.display.Info().current_h)
print "Framebuffer size: %d x %d" % (size[0], size[1])
self.screen = pygame.display.set_mode(size, pygame.FULLSCREEN)
# Clear the screen to start
self.screen.fill((0, 0, 0))
# Initialise font support
pygame.font.init()
# Render the screen
pygame.display.update()

def __del__(self):
    "Destructor to make sure pygame shuts down, etc."

def drawGraticule(self):
    "Renders an empty graticule"
    # The graticule is divided into 10 columns x 8 rows
    # Each cell is 50x40 pixels large, with 5 subdivisions per
    # cell, meaning 10x8 pixels each. Subdivision lines are
    # displayed on the central X and Y axis
    # Active area = 10,30 to 510,350 (500x320 pixels)
    borderColor = (255, 255, 255)
    lineColor = (64, 64, 64)
    subDividerColor = (128, 128, 128)
    # Outer border: 2 pixels wide
    pygame.draw.rect(self.screen, borderColor, (8,28,504,324), 2)
    # Horizontal lines (40 pixels apart)
    for i in range(0, 7):
        y = 70+i*40
        pygame.draw.line(self.screen, lineColor, (10, y), (510, y))
    # Vertical lines (50 pixels apart)
    for i in range(0, 9):
        x = 60+i*50
        pygame.draw.line(self.screen, lineColor, (x, 30), (x, 350))
    # Vertical sub-divisions (8 pixels apart)
    for i in range(1, 40):
        y = 30+i*8
        pygame.draw.line(self.screen, subDividerColor, (258, y), (262, y))
    # Horizontal sub-divisions (10 pixels apart)
    for i in range(1, 50):
        x = 10+i*10
        pygame.draw.line(self.screen, subDividerColor, (x, 188), (x, 192))

def test(self):
    "Test method to make sure the display is configured correctly"
    adcColor = (255, 255, 0) # Yellow
    self.drawGraticule()
    # Render the Adafruit logo at 10,360
    logo = pygame.image.load('adafruit_logo.gif').convert()
    self.screen.blit(logo, (10, 335))

```

```

# Get a font and use it render some text on a Surface.
font = pygame.font.Font(None, 30)
text_surface = font.render('pyScope (%s)' % "0.1",
    True, (255, 255, 255)) # White text
# Blit the text at 10, 0
self.screen.blit(text_surface, (10, 0))
# Render some text with a background color
text_surface = font.render('Channel 0',
    True, (0, 0, 0), (255, 255, 0)) # Black text with yellow BG
# Blit the text
self.screen.blit(text_surface, (540, 30))
# Update the display
pygame.display.update()
# Random adc data
yLast = 260
for x in range(10, 509):
    y = random.randrange(30, 350, 2) # Even number from 30 to 350
    pygame.draw.line(self.screen, adcColor, (x, yLast), (x+1, y))
    yLast = y
    pygame.display.update()

# Create an instance of the PyScope class
scope = pyscope()
scope.test()
# Wait 10 seconds
time.sleep(10)

```

This should give you something similar to the following (depending on your display):