# Table of Contents

## Overview
- Parts

## Prep the MatrixPortal
- Power Prep
- Power Terminals
- Panel Power
- Dual Matrix Setup
- Board Connection

## LED Matrix Diffuser
- LED Diffusion Acrylic
- Measure and Cut the Plastic
- Uglu Dashes
- Stand

## Install CircuitPython
- Set up CircuitPython Quick Start!
- Further Information

## Custom Font
- Glyph Bitmap Distribution Format
- Glyph Tweaks

## Code the Interval Timer
- Text Editor
- Upload the Code and Libraries to the Matrix Portal M4
- Import Libraries
- Constants
- Initializing the Matrix Portal
- Drawing setup
- Font
- Label setup
- Update Time

## Use
- Pomodoro Timer
Overview

Transform your home into your own boxing gym with this Matrix Portal M4 Boxing Interval Timer. This guide will show you how to assemble and program your own Boxing Interval Timer using CircuitPython.

This features the Matrix Portal M4, which easily drives a 64 x 32 RGB Matrix display with over 200 colorful LEDs to display the timer with excellent visibility - even for the largest gyms!

The timer uses an IBM Plex mono typeface, using a neon green hue for the first minute and thirty seconds for workout time. The text color changes to a yellowish-amber color for the cooldown portion of the time and red for the resting time.

Parts

Adafruit Matrix Portal Starter Kit - ADABOX 016 Essentials
If you missed out on AdaBox016, it's not too late for you to pick up the parts necessary to build many of the projects! It...
https://www.adafruit.com/product/4812
Adafruit Matrix Portal - CircuitPython
Powered Internet Display
Folks love our wide selection of RGB matrices and accessories, for making custom colorful LED displays... and our RGB Matrix Shields...
https://www.adafruit.com/product/4745

Black LED Diffusion Acrylic Panel - 10.2" x 5.1"
nice whoppin' rectangular slab of some lovely black acrylic to add some extra diffusion to your LED Matrix project. This material is 2.6mm (0.1") thick and is made of...
https://www.adafruit.com/product/4749

5V 2.5A Switching Power Supply with 20AWG MicroUSB Cable
Our all-in-one 5V 2.5 Amp + MicroUSB cable power adapter is the perfect choice for powering single-board computers like Raspberry Pi, BeagleBone, or anything else that's...

Micro B USB to USB C Adapter
As technology changes and adapts, so does Adafruit, and speaking of adapting, this adapter has a Micro B USB jack and a USB C...
https://www.adafruit.com/product/4299
Prep the MatrixPortal

Power Prep
The MatrixPortal supplies power to the matrix display panel via two standoffs. These come with protective tape applied (part of our manufacturing process) which MUST BE REMOVED!

Use some tweezers or a fingernail to remove the two amber circles.
Power Terminals
Next, screw in the spade connectors to the corresponding standoff.

red wire goes to +5V
black wire goes to GND
Panel Power
Plug either one of the four-conductor power plugs into the power connector pins on the panel. The plug can only go in one way, and that way is marked on the board's silkscreen.

Dual Matrix Setup
If you're planning to use a 64x64 matrix, follow these instructions on soldering the Address E Line jumper ().
Board Connection

Now, plug the board into the left side shrouded 8x2 connector as shown. The orientation matters, so take a moment to confirm that the white indicator arrow on the matrix panel is oriented pointing up and right as seen here and the MatrixPortal overhangs the edge of the panel when connected. This allows you to use the edge buttons from the front side.

Check nothing is impeding the board from plugging in firmly. If there’s a plastic nub on the matrix that’s keeping the Portal from sitting flat, cut it off with diagonal cutters.
You can add an LED diffusion acrylic faceplate to your LED matrix display. (Pictured here with the ON AIR project)

This can help protect the LEDs as well as enhance the look of the sign both indoors and out by reducing glare and specular highlights of the plastic matrix grid.
Measure and Cut the Plastic

You can use the sign to measure and mark cut lines on the paper backing of the acrylic sheet.

Then, use a tablesaw or bandsaw with a fine toothed blade and a guide or sled to make the cuts.

Note: it is possible to score and snap acrylic, but it can be very tricky to get an even snap without proper clamping.
Peel away the paper backing from both sides and set the acrylic onto your matrix display.
Uglu Dashes

The best method we've found for adhering acrylic to the matrix display is to use Uglu Dashes clear adhesive rectangles from Pro Tapes (https://protapes.com/). They are incredibly strong (although can be removed if necessary), easy to apply, and are invisible once attached.

Use one at each corner and one each at the halfway point of the long edges, then press the acrylic and matrix panel together for about 20 seconds.

Here you can see the impact of using the diffusion acrylic. (Pictured here with the ON AIR sign project)
Stand
A very simple and attractive way to display your matrix is with the adjustable bent-wire stand ( ).
Alternately, you can use a frame, 3D printed brackets, tape, glue, or even large binder clips to secure the acrylic to the sign and then mount it on a wall, shelf, or display cabinet.

These mini-magnet feet can be used to stick the sign to a ferrous surface.

Install CircuitPython

CircuitPython is a derivative of MicroPython designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the CIRCUITPY drive to iterate.

Set up CircuitPython Quick Start!

Follow this quick step-by-step for super-fast Python power :)  

Download the latest version of CircuitPython for this board via circuitpython.org

Further Information

For more detailed info on installing CircuitPython, check out Installing CircuitPython.
Click the link above and download the latest UF2 file.

Download and save it to your desktop (or wherever is handy).

Plug your MatrixPortal M4 into your computer using a known-good USB cable.

A lot of people end up using charge-only USB cables and it is very frustrating! So make sure you have a USB cable you know is good for data sync.

Double-click the Reset button (indicated by the green arrow) on your board, and you will see the NeoPixel RGB LED (indicated by the magenta arrow) turn green. If it turns red, check the USB cable, try another USB port, etc.

If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!
You will see a new disk drive appear called MATRIXBOOT.

Drag the adafruit_circuitpython_etc.uf2 file to MATRIXBOOT.

The LED will flash. Then, the MATRIXBOOT drive will disappear and a new disk drive called CIRCUITPY will appear.

That's it, you're done! :)

©Adafruit Industries
Instead of the default Terminal typeface, we'll use a lovely typeface converted to a bitmap font for use on the matrix display.

In case you want to make your own font, [this excellent guide](#) shows how to use FontForge to do just that!

I selected the [IBM Plex Mono Medium](#) typeface. I then converted it to a single bitmap font in a size that works for the 64x32 pixel matrix display, and hand kerned the colon and space characters. Here is the modified .bdf font:

```
IBMplexMono-Medium-24_jep.bdf.zip
```

To use it, download and uncompress the zip file and then drag it onto the board's CIR CUITPY drive. In CircuitPython, the font is loaded so it can be used for text display.

**Glyph Bitmap Distribution Format**

Side note about Glyph Bitmap Distribution Format (BDF) fonts -- they are stored as text files that are human and computer readable, and you can even edit them by hand if you feel like getting super particular!

Here's what the letter 'A' glyph looks like in a typical converted .bdf font:
Each of those lines after the word `BITMAP` is a hexadecimal pixel mapping of a row of the glyph. Here's what that looks like converted to binary, with each row padded to 16-bits:

```
0000001000000000
0000011000000000
0000011100000000
0000111100000000
0000111100000000
0000100110000000
0001100110000000
0001100011000000
0011000011000000
0011111111000000
0111111111100000
0110000001100000
0110000000110000
1100000000110000
1100000000011000
```
Glyph Tweaks

In case you wanted to add a stylish hat on top of your 'A', you could change the first row to: 1fc0 which is 0001111111000000 in binary.

This would result it the image to the left.

Now, that's a bit of a silly example, but the great thing about this is that you can fine-tune your glyphs beyond what may be easily accomplished during the conversion process.

Code the Interval Timer

After completing the setup process for your Matrix Portal M4 with CircuitPython, you can gather the required code and libraries by downloading the Project Bundle.

To accomplish this task, click the "Download Project Bundle" button in the window below. The file will be saved to your computer as a zipped folder.

```python
# SPDX-FileCopyrightText: 2023 Trevor Beaton for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import displayio
import terminalio
from adafruit_display_text.label import Label
from adafruit_bitmap_font import bitmap_font
from adafruit_matrixportal.matrix import Matrix

# set the timer length
TIMER_LENGTH = 180  # 3 minutes

BLINK = True
DEBUG = False

# --- Display setup ---
matrix = Matrix()
display = matrix.display
```
# --- Drawing setup ---
group = displayio.Group()  # Create a Group
bitmap = displayio.Bitmap(64, 32, 2)  # Create a bitmap object, width, height, bit depth
color = displayio.Palette(4)  # Create a color palette
color[0] = 0x000000  # black background
color[1] = 0xFF0000  # red
color[2] = 0xFF8C00  # yellow
color[3] = 0x3DEB34  # green

color[0] = 0x000000  # black background
color[1] = 0xFF0000  # red
color[2] = 0xFF8C00  # yellow
color[3] = 0x3DEB34  # green

# Create a TileGrid using the Bitmap and Palette
tile_grid = displayio.TileGrid(bitmap, pixel_shader=color)
group.append(tile_grid)  # Add the TileGrid to the Group
display.show(group)

if not DEBUG:
    font = bitmap_font.load_font("/IBMPlexMono-Medium-24_jep.bdf")
else:
    font = terminalio.FONT
clock_label = Label(font)
def update_time(remaining_time):
    now = time.localtime()  # Get the time values we need
    
    # calculate remaining time in seconds
    seconds = remaining_time % 60
    minutes = remaining_time // 60
    
    if BLINK:
        colon = "":" if now[5] % 2 else " 
    else:
        colon = ":"
    
    clock_label.text = "{minutes:02d}{colon}{seconds:02d}".format(
        minutes=minutes, seconds=seconds, colon=colon
    )

    if remaining_time < 60:
        clock_label.color = color[1]
    elif remaining_time < 90:
        clock_label.color = color[2]
    elif remaining_time > 90:
        clock_label.color = color[3]

    bbx, bby, bbwidth, bbh = clock_label.bounding_box
    # Center the label
    clock_label.x = round(display.width / 2 - bbwidth / 2)
    clock_label.y = display.height // 2
    if DEBUG:
        print("Label bounding box: 

    if remaining_time < 0:
        remaining_time = TIMER_LENGTH
    return remaining_time

def main():
    remaining_time = TIMER_LENGTH
    update_time(remaining_time)
group.append(clock_label)

    while True:
        remaining_time = update_time(remaining_time)
time.sleep(1)
if __name__ == "__main__":
    main()

Text Editor

Adafruit recommends using the Mu editor for editing your CircuitPython code. You can get more info in this guide including info on how to use the REPL/serial terminal.

Alternatively, you can use any text editor that saves simple text files.

Upload the Code and Libraries to the Matrix Portal M4

After downloading the Project Bundle, you can plug your Matrix Portal M4 into the computer's USB port with a suitable USB data+power cable. You should see a new flash drive in the computer's File Explorer or Finder (depending on your operating system) called CIRCUITPY. Unzip the folder and copy the following items to the Matrix Portal M4's CIRCUITPY drive.

- lib folder
- code.py
- IBMplexMono-Medium-24_jep.bdf

Your Matrix Portal M4 CIRCUITPY drive should look like this after copying the lib folder, IBMplexMono-Medium-24_jep.bdf file and the code.py file.
Import Libraries

First thing is to import the modules needed for the code to run. The `time` module allows you to work with time, while the `displayio` and `terminalio` provide classes and methods to render the text to the display.

The `adafruit_bitmap_font` module supports bitmap fonts, and the `adafruit_matrixportal` module supports working with the Matrix Portal.

```python
import time
import displayio
import terminalio
from adafruit_display_text.label import Label
from adafruit_bitmap_font import bitmap_font
from adafruit_matrixportal.matrix import Matrix
```

Constants

These constants are used throughout the code. `TIMER_LENGTH` sets the length of the countdown timer in seconds, `BLINK` determines whether the timer's colon separator blinks, and `DEBUG` allows you to see debug information in the console.
Initializing the Matrix Portal

This code initializes the Matrix Portal and sets it up as a display object. Very important.

```python
matrix = Matrix()
display = matrix.display
```

Drawing setup

Now, to set up the display's drawing objects. Create a `Group` to hold the tile grid and a Bitmap object to represent the display's pixels. You also want to create a Palette to hold the colors used in the display. The TileGrid is created using the Bitmap and Palette and is added to the `Group`. Finally, the Group is shown on the display.

```python
group = displayio.Group()  # Create a Group
bitmap = displayio.Bitmap(64, 32, 2)  # Create a bitmap object, width, height, bit depth
color = displayio.Palette(4)  # Create a color palette
color[0] = 0x000000  # black background
color[1] = 0xFF0000  # red
color[2] = 0xFF8C00  # yellow
color[3] = 0x3DEB34  # green
```

Font

This set of code sets up the font for the countdown interval timer. If `DEBUG` is set to `False`, a bitmap font is loaded from a file. If not, the font is used.

```python
if not DEBUG:
    font = bitmap_font.load_font("/IBMPlexMono-Medium-24_jep.bdf")
else:
    font = terminalio.FONT
```

Label setup

This line creates a `Label` object using the previously loaded font.

```python
clock_label = Label(font)
```
Update Time

This `update_time()` function takes one argument `remaining_time` as input.

Within the function, it first calls `time.localtime()` to observe and store the current local time in the variable.

Next, it calculates the number of minutes and seconds remaining from the input argument `remaining_time` in seconds.

The code then determines if it displays a blinking colon or not based on the value of the `BLINK` variable and the current second of the `now` variable. It sets the `colon` variable to `:` if the current second is even and if the current second is odd.

It then sets the text of the `clock_label` object to display the minutes and seconds, with the `colon`. The values are formatted to display leading zeros when necessary.

The code block then sets the color of the `clock_label` object based on the remaining time. Here's the condition:

- If the remaining time is less than 60 seconds, it sets the color to `color[1]`.
- If it's less than 90 seconds, it sets the color to `color[2]`.
- If it's greater than 90 seconds, it sets the color to `color[3]`.

The code then calculates the bounding box of the `clock_label` object and centers it horizontally on the screen. If the `DEBUG` variable is set to `True`, it prints the bounding box and position of the label.

Finally, the code decrements the `remaining_time` variable by 1 and checks if it's less than 0. If the `remaining_time` is equal to 0, it sets `remaining_time` back to `TIMER_LENGTH`, another variable that must have been defined elsewhere in the code. The function then returns the `remaining_time` variable.

The `main()` function initializes the `remaining_time` variable to `TIMER_LENGTH`, calls `update_time()` with that variable as input, and appends the `clock_label` object to a `group`. Then it enters a `while` loop that repeatedly calls `update_time()` with the current value of `remaining_time`, sleeps for 1 second using `time.sleep(1)`, and continues the loop.

```python
def update_time(remaining_time):
    now = time.localtime()  # Get the time values we need
```
# calculate remaining time in seconds
seconds = remaining_time % 60
minutes = remaining_time // 60

if BLINK:
    colon = ":" if now[5] % 2 else " 
else:
    colon = ":"

clock_label.text = "{minutes:02d}{colon}{seconds:02d}".format(
    minutes=minutes, seconds=seconds, colon=colon
)

if remaining_time < 60:
    clock_label.color = color[1]
elif remaining_time < 90:
    clock_label.color = color[2]
elif remaining_time > 90:
    clock_label.color = color[3]

bbx, bby, bbwidth, bbh = clock_label.bounding_box
# Center the label
clock_label.x = round(display.width / 2 - bbwidth / 2)
clock_label.y = display.height // 2
if DEBUG:
    print("Label bounding box: {},{},{},{}".format(bbx, bby, bbwidth, bbh))
    print("Label x: {} y: {}".format(clock_label.x, clock_label.y))

# decrement remaining time
remaining_time -= 1
if remaining_time < 0:
    remaining_time = TIMER_LENGTH

return remaining_time

def main():
    remaining_time = TIMER_LENGTH
    update_time(remaining_time)
group.append(clock_label)

    while True:
        remaining_time = update_time(remaining_time)
time.sleep(1)

if __name__ == "__main__":
    main()
Use

After writing and uploading your code to your Matrix portal, you should see a bright green countdown timer displayed. Now you can disconnect your timer from your computer, and with your AC adapter, you will be able to use your countdown timer anywhere an electrical outlet is available.

You can also turn this boxing interval timer into a Pomodoro timer to use the Pomodoro technique!

Pomodoro Timer

The Pomodoro Technique is a time management method developed by Francesco Cirillo in the late 1980s. It uses a kitchen timer to break work into intervals, typically...
25 minutes in length, separated by short breaks. Each interval is known as a Pomodoro, from the Italian word for tomato, after the tomato-shaped kitchen timer Cirillo used as a university student.

To change this timer’s countdown time, update the following time:

```
TIMER_LENGTH = 1500  # 25 minutes
```

You can also make the timer’s text red like a tomato! By updating this line:

```
elif remaining_time > 90:
    clock_label.color = color[1]
```