AdafruitDisplay$_{S}hapesLibraryDocumentation$

Release 1.0

Limor Fried

Oct 25, 2021

Contents

Various common shapes for use with displayio

CHAPTER 1

Dependencies

This driver depends on:

- Adafruit CircuitPython

Please ensure all dependencies are available on the CircuitPython filesystem. This is easily achieved by downloading the Adafruit library and driver bundle.

1.1 Installing from PyPI

On supported GNU/Linux systems like the Raspberry Pi, you can install the driver locally from PyPI. To install for current user:

```
pip3 install adafruit-circuitpython-display_shapes
```

To install system-wide (this may be required in some cases):

```
sudo pip3 install adafruit-circuitpython-display_shapes
```

To install in a virtual environment in your current project:

```
mkdir project-name && cd project-name
python3 -m venv .env
source .env/bin/activate
pip3 install adafruit-circuitpython-display_shapes
```

CHAPTER 2

Usage Example

```
import board
import displayio
from adafruit_display_shapes.rect import Rect
from adafruit_display_shapes.circle import Circle
from adafruit_display_shapes.roundrect import RoundRect

splash = displayio.Group()
board.DISPLAY.show(splash)

color_bitmap = displayio.Bitmap(320, 240, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0xFFFFFF
bg_sprite = displayio.TileGrid(color_bitmap, pixel_shader=color_palette, position=(0,␣
↪0))
print(bg_sprite.position)
splash.append(bg_sprite)

triangle = Triangle(170, 50, 120, 140, 210, 160, fill=0x00FF00, outline=0xFF00FF)
splash.append(triangle)

rect = Rect(80, 20, 41, 41, fill=0x0)
splash.append(rect)

circle = Circle(100, 100, 20, fill=0x00FF00, outline=0xFF00FF)
splash.append(circle)

rect2 = Rect(50, 100, 61, 81, outline=0x0, stroke=3)
splash.append(rect2)

roundrect = RoundRect(10, 10, 61, 81, 10, fill=0x0, outline=0xFF00FF, stroke=6)
splash.append(roundrect)

while True:
    pass
```

CHAPTER 3

Documentation

API documentation for this library can be found on Read the Docs.

CHAPTER 4

Contributing

Contributions are welcome! Please read our Code of Conduct before contributing to help this project stay welcoming.

CHAPTER 5

Building locally

5.1 Zip release files

To build this library locally you'll need to install the circuitpython-build-tools package.

```
python3 -m venv .env
source .env/bin/activate
pip install circuitpython-build-tools
```

Once installed, make sure you are in the virtual environment:

```
source .env/bin/activate
```

Then run the build:

```
circuitpython-build-bundles --filename_prefix adafruit-circuitpython-display_shapes --
↪library_location .
```

5.2 Sphinx documentation

Sphinx is used to build the documentation based on rST files and comments in the code. First, install dependencies (feel free to reuse the virtual environment from above):

```
python3 -m venv .env
source .env/bin/activate
pip install Sphinx sphinx-rtd-theme
```

Now, once you have the virtual environment activated:

```
cd docs
sphinx-build -E -W -b html . _build/html
```

This will output the documentation to `docs/_build/html`. Open the index.html in your browser to view them. It will also (due to -W) error out on any warning like Travis will. This is a good way to locally verify it will pass.

CHAPTER 6

Table of Contents

6.1 Simple test

Ensure your device works with this simple test.

Listing 1: examples/display_shapes_simpletest.py

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

import board
import displayio
from adafruit_display_shapes.rect import Rect
from adafruit_display_shapes.circle import Circle
from adafruit_display_shapes.roundrect import RoundRect
from adafruit_display_shapes.triangle import Triangle
from adafruit_display_shapes.line import Line
from adafruit_display_shapes.polygon import Polygon

# Make the display context
splash = displayio.Group()
board.DISPLAY.show(splash)

# Make a background color fill
color_bitmap = displayio.Bitmap(320, 240, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0xFFFFFF
bg_sprite = displayio.TileGrid(color_bitmap, x=0, y=0, pixel_shader=color_palette)
splash.append(bg_sprite)
################################################################################

splash.append(Line(220, 130, 270, 210, 0xFF0000))
splash.append(Line(270, 210, 220, 210, 0xFF0000))
splash.append(Line(220, 210, 270, 130, 0xFF0000))
```

(continues on next page)

(continued from previous page)

```
splash.append(Line(270, 130, 220, 130, 0xFF0000))

# Draw a blue star
polygon = Polygon(
    [
        (255, 40),
        (262, 62),
        (285, 62),
        (265, 76),
        (275, 100),
        (255, 84),
        (235, 100),
        (245, 76),
        (225, 62),
        (248, 62),
    ],
    outline=0x0000FF,
)
splash.append(polygon)

triangle = Triangle(170, 50, 120, 140, 210, 160, fill=0x00FF00, outline=0xFF00FF)
splash.append(triangle)

rect = Rect(80, 20, 41, 41, fill=0x0)
splash.append(rect)

circle = Circle(100, 100, 20, fill=0x00FF00, outline=0xFF00FF)
splash.append(circle)

rect2 = Rect(50, 100, 61, 81, outline=0x0, stroke=3)
splash.append(rect2)

roundrect = RoundRect(10, 10, 61, 81, 10, fill=0x0, outline=0xFF00FF, stroke=6)
splash.append(roundrect)


while True:
    pass
```

6.2 Simple test MagTag

Simple test with the MagTag.

Listing 2: examples/display_shapes_simpletest_magtag.py

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

import board
import displayio
from adafruit_display_shapes.rect import Rect
from adafruit_display_shapes.circle import Circle
from adafruit_display_shapes.roundrect import RoundRect
from adafruit_display_shapes.triangle import Triangle
```

(continues on next page)

(continued from previous page)

```
from adafruit_display_shapes.line import Line
from adafruit_display_shapes.polygon import Polygon

# use built in display (PyPortal, PyGamer, PyBadge, CLUE, etc.)
# see guide for setting up external displays (TFT / OLED breakouts, RGB matrices, etc.)
# https://learn.adafruit.com/circuitpython-display-support-using-displayio/display-and-display-bus
display = board.DISPLAY

# Make the display context
splash = displayio.Group()
display.show(splash)

# Make a background color fill
color_bitmap = displayio.Bitmap(display.width, display.height, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0xFFFFFF
bg_sprite = displayio.TileGrid(color_bitmap, x=0, y=0, pixel_shader=color_palette)
splash.append(bg_sprite)
################################################################################

splash.append(Line(5, 74, 10, 110, 0x000000))
splash.append(Line(15, 74, 20, 110, 0x000000))
splash.append(Line(25, 74, 30, 110, 0x000000))
splash.append(Line(35, 74, 40, 110, 0x000000))

# Draw star
polygon = Polygon(
    [
        (255, 40),
        (262, 62),
        (285, 62),
        (265, 76),
        (275, 100),
        (255, 84),
        (235, 100),
        (245, 76),
        (225, 62),
        (248, 62),
    ],
    outline=0x000000,
)
splash.append(polygon)

triangle = Triangle(170, 20, 140, 90, 210, 100, fill=0x999999, outline=0x000000)
splash.append(triangle)

rect = Rect(80, 20, 41, 41, fill=0x999999, outline=0x666666)
splash.append(rect)

circle = Circle(100, 100, 20, fill=0xFFFFFF, outline=0x000000)
splash.append(circle)

rect2 = Rect(70, 85, 61, 30, outline=0x0, stroke=3)
splash.append(rect2)

```

(continues on next page)

(continued from previous page)

```
roundrect = RoundRect(10, 10, 61, 51, 10, fill=0x666666, outline=0x000000, stroke=6)
splash.append(roundrect)

display.refresh()
while True:
    pass
```

6.3 Sparkline Simple Test

Simple test with Sparklines

Listing 3: examples/display_shapes_sparkline_simpletest.py

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# class of sparklines in CircuitPython
# created by Kevin Matocha - Copyright 2020 (C)

# See the bottom for a code example using the `sparkline` Class.

# # File: display_shapes_sparkline.py
# A sparkline is a scrolling line graph, where any values added to sparkline using
# `add_value` are plotted.
#
# The `sparkline` class creates an element suitable for adding to the display using
# `display.show(mySparkline)`
# or adding to a `displayio.Group` to be displayed.
#
# When creating the sparkline, identify the number of `max_items` that will be
# included in the graph.
# When additional elements are added to the sparkline and the number of items has
# exceeded max_items, any excess values are removed from the left of the graph,
# and new values are added to the right.


# The following is an example that shows the

# setup display
# instance sparklines
# add to the display
# Loop the following steps:
#         add new values to sparkline `add_value`
#         update the sparklines `update`

import time
import random
import board
import displayio


from adafruit_display_shapes.sparkline import Sparkline

if "DISPLAY" not in dir(board):
```

(continues on next page)

(continued from previous page)

```
    # Setup the LCD display with driver
    # You may need to change this to match the display driver for the chipset
    # used on your display
    from adafruit_ili9341 import ILI9341

    displayio.release_displays()

    # setup the SPI bus
    spi = board.SPI()
    tft_cs = board.D9  # arbitrary, pin not used
    tft_dc = board.D10
    tft_backlight = board.D12
    tft_reset = board.D11

    while not spi.try_lock():
        spi.configure(baudrate=32000000)

    spi.unlock()

    display_bus = displayio.FourWire(
        spi,
        command=tft_dc,
        chip_select=tft_cs,
        reset=tft_reset,
        baudrate=32000000,
        polarity=1,
        phase=1,
    )

    print("spi.frequency: {}".format(spi.frequency))

    # Number of pixels in the display
    DISPLAY_WIDTH = 320
    DISPLAY_HEIGHT = 240

    # create the display
    display = ILI9341(
        display_bus,
        width=DISPLAY_WIDTH,
        height=DISPLAY_HEIGHT,
        rotation=180,  # The rotation can be adjusted to match your configuration.
        auto_refresh=True,
        native_frames_per_second=90,
    )

    # reset the display to show nothing.
    display.show(None)
else:
    # built-in display
    display = board.DISPLAY

########################################
# Create background bitmaps and sparklines
########################################

# Baseline size of the sparkline chart, in pixels.
chart_width = display.width
```

(continues on next page)

(continued from previous page)

```
chart_height = display.height

# sparkline1 uses a vertical y range between 0 to 10 and will contain a
# maximum of 40 items
sparkline1 = Sparkline(
    width=chart_width, height=chart_height, max_items=40, y_min=0, y_max=10, x=0, y=0
)

# Create a group to hold the sparkline and append the sparkline into the
# group (my_group)
#
# Note: In cases where display elements will overlap, then the order the elements
# are added to the group will set which is on top.  Latter elements are displayed
# on top of former elements.
my_group = displayio.Group()

# add the sparkline into my_group
my_group.append(sparkline1)


# Add my_group (containing the sparkline) to the display
display.show(my_group)

# Start the main loop
while True:

    # turn off the auto_refresh of the display while modifying the sparkline
    display.auto_refresh = False

    # add_value: add a new value to a sparkline
    # Note: The y-range for mySparkline1 is set to 0 to 10, so all these random
    # values (between 0 and 10) will fit within the visible range of this sparkline
    sparkline1.add_value(random.uniform(0, 10))

    # turn the display auto_refresh back on
    display.auto_refresh = True

    # The display seems to be less jittery if a small sleep time is provided
    # You can adjust this to see if it has any effect
    time.sleep(0.01)
```

6.4 Sparkline Ticks Example

Example using tick with the Sparkline class

Listing 4: examples/display_shapes_sparkline_ticks.py

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# class of sparklines in CircuitPython
# created by Kevin Matocha - Copyright 2020 (C)

# See the bottom for a code example using the `sparkline` Class.
```

(continues on next page)

(continued from previous page)

```

# # File: display_shapes_sparkline.py
# A sparkline is a scrolling line graph, where any values added to sparkline
# using `add_value` are plotted.
#
# The `sparkline` class creates an element suitable for adding to the display
# using `display.show(mySparkline)` or adding to a `displayio.Group` to be displayed.
#
# When creating the sparkline, identify the number of `max_items` that will be
# included in the graph.
# When additional elements are added to the sparkline and the number of items
# has exceeded max_items, any excess values are removed from the left of the
# graph, and new values are added to the right.


# The following is an example that shows the

# setup display
# instance sparklines
# add to the display
# Loop the following steps:
#         add new values to sparkline `add_value`
#         update the sparklines `update`

import random
import time
import board
import displayio
import terminalio
from adafruit_display_text import label
from adafruit_display_shapes.sparkline import Sparkline
from adafruit_display_shapes.line import Line
from adafruit_display_shapes.rect import Rect


if "DISPLAY" not in dir(board):
    # Setup the LCD display with driver
    # You may need to change this to match the display driver for the chipset
    # used on your display
    from adafruit_ili9341 import ILI9341

    displayio.release_displays()

    # setup the SPI bus
    spi = board.SPI()
    tft_cs = board.D9  # arbitrary, pin not used
    tft_dc = board.D10
    tft_backlight = board.D12
    tft_reset = board.D11

    while not spi.try_lock():
        spi.configure(baudrate=32000000)
    spi.unlock()

    display_bus = displayio.FourWire(
        spi,
        command=tft_dc,
```

(continues on next page)

(continued from previous page)

```
        chip_select=tft_cs,
        reset=tft_reset,
        baudrate=32000000,
        polarity=1,
        phase=1,
    )

    print("spi.frequency: {}".format(spi.frequency))

    # Number of pixels in the display
    DISPLAY_WIDTH = 320
    DISPLAY_HEIGHT = 240

    # create the display
    display = ILI9341(
        display_bus,
        width=DISPLAY_WIDTH,
        height=DISPLAY_HEIGHT,
        rotation=180,  # The rotation can be adjusted to match your configuration.
        auto_refresh=True,
        native_frames_per_second=90,
    )

    # reset the display to show nothing.
    display.show(None)
else:
    # built-in display
    display = board.DISPLAY

########################################
# Create background bitmaps and sparklines
########################################

# Baseline size of the sparkline chart, in pixels.
chart_width = display.width - 50
chart_height = display.height - 50

font = terminalio.FONT

line_color = 0xFFFFFF

# Setup the first bitmap and sparkline
# This sparkline has no background bitmap
# mySparkline1 uses a vertical y range between 0 to 10 and will contain a
# maximum of 40 items
sparkline1 = Sparkline(
    width=chart_width,
    height=chart_height,
    max_items=40,
    y_min=0,
    y_max=10,
    x=40,
    y=30,
    color=line_color,
)

# Label the y-axis range
```

(continues on next page)

(continued from previous page)

```
text_xoffset = -10
text_label1a = label.Label(
    font=font, text=str(sparkline1.y_top), color=line_color
)  # yTop label
text_label1a.anchor_point = (1, 0.5)  # set the anchorpoint at right-center
text_label1a.anchored_position = (
    sparkline1.x + text_xoffset,
    sparkline1.y,
)  # set the text anchored position to the upper right of the graph

text_label1b = label.Label(
    font=font, text=str(sparkline1.y_bottom), color=line_color
)  # yTop label
text_label1b.anchor_point = (1, 0.5)  # set the anchorpoint at right-center
text_label1b.anchored_position = (
    sparkline1.x + text_xoffset,
    sparkline1.y + chart_height,
)  # set the text anchored position to the upper right of the graph


bounding_rectangle = Rect(
    sparkline1.x, sparkline1.y, chart_width, chart_height, outline=line_color
)


# Create a group to hold the sparkline, text, rectangle and tickmarks
# append them into the group (my_group)
#
# Note: In cases where display elements will overlap, then the order the
# elements are added to the group will set which is on top.  Latter elements
# are displayed on top of former elemtns.

my_group = displayio.Group()

my_group.append(sparkline1)
my_group.append(text_label1a)
my_group.append(text_label1b)
my_group.append(bounding_rectangle)

total_ticks = 10

for i in range(total_ticks + 1):
    x_start = sparkline1.x - 5
    x_end = sparkline1.x
    y_both = int(round(sparkline1.y + (i * (chart_height) / (total_ticks))))
    if y_both > sparkline1.y + chart_height - 1:
        y_both = sparkline1.y + chart_height - 1
    my_group.append(Line(x_start, y_both, x_end, y_both, color=line_color))


# Set the display to show my_group that contains the sparkline and other graphics
display.show(my_group)

# Start the main loop
while True:

```

(continues on next page)

(continued from previous page)

```
    # Turn off auto_refresh to prevent partial updates of the screen during updates
    # of the sparkline drawing
    display.auto_refresh = False

    # add_value: add a new value to a sparkline
    # Note: The y-range for mySparkline1 is set to 0 to 10, so all these random
    # values (between 0 and 10) will fit within the visible range of this sparkline
    sparkline1.add_value(random.uniform(0, 10))

    # Turn on auto_refresh for the display
    display.auto_refresh = True

    # The display seems to be less jittery if a small sleep time is provided
    # You can adjust this to see if it has any effect
    time.sleep(0.01)
```

6.5 Sparkline Triple Test

reate background bitmaps and sparklines

Listing 5: examples/display_shapes_sparkline_triple.py

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# class of sparklines in CircuitPython
# created by Kevin Matocha - Copyright 2020 (C)

# See the bottom for a code example using the `sparkline` Class.

# # File: display_shapes_sparkline.py
# A sparkline is a scrolling line graph, where any values added to sparkline
# using `add_value` are plotted.
#
# The `sparkline` class creates an element suitable for adding to the display
# using `display.show(mySparkline)` or adding to a `displayio.Group` to be displayed.
#
# When creating the sparkline, identify the number of `max_items` that will be
# included in the graph.
# When additional elements are added to the sparkline and the number of items
# has exceeded max_items, any excess values are removed from the left of the
# graph, and new values are added to the right.


# The following is an example that shows the

# setup display
# instance sparklines
# add to the display
# Loop the following steps:
#         add new values to sparkline `add_value`
#         update the sparklines `update`

import random
```

(continues on next page)

(continued from previous page)

```
import time
import board
import displayio
import terminalio
from adafruit_display_text import label
from adafruit_display_shapes.sparkline import Sparkline


if "DISPLAY" not in dir(board):
    # Setup the LCD display with driver
    # You may need to change this to match the display driver for the chipset
    # used on your display
    from adafruit_ili9341 import ILI9341

    displayio.release_displays()

    # setup the SPI bus
    spi = board.SPI()
    tft_cs = board.D9  # arbitrary, pin not used
    tft_dc = board.D10
    tft_backlight = board.D12
    tft_reset = board.D11

    while not spi.try_lock():
        spi.configure(baudrate=32000000)
    spi.unlock()

    display_bus = displayio.FourWire(
        spi,
        command=tft_dc,
        chip_select=tft_cs,
        reset=tft_reset,
        baudrate=32000000,
        polarity=1,
        phase=1,
    )

    print("spi.frequency: {}".format(spi.frequency))

    # Number of pixels in the display
    DISPLAY_WIDTH = 320
    DISPLAY_HEIGHT = 240

    # create the display
    display = ILI9341(
        display_bus,
        width=DISPLAY_WIDTH,
        height=DISPLAY_HEIGHT,
        rotation=180,  # The rotation can be adjusted to match your configuration.
        auto_refresh=True,
        native_frames_per_second=90,
    )

    # reset the display to show nothing.
    display.show(None)
else:
    # built-in display
```

(continues on next page)

(continued from previous page)

```
    display = board.DISPLAY
    DISPLAY_WIDTH = board.DISPLAY.width

########################################
# Create background bitmaps and sparklines
########################################

# Baseline size of the sparkline chart, in pixels.
chart_width = 50
chart_height = 50

font = terminalio.FONT

# Setup the first bitmap and sparkline
# This sparkline has no background bitmap
# sparkline1 uses a vertical y range between -1 to +1.25 and will contain a maximum␣
↪of 40 items
sparkline1 = Sparkline(
    width=chart_width,
    height=chart_height,
    max_items=40,
    y_min=-1,
    y_max=1.25,
    x=10,
    y=10,
)

# Label the y-axis range
text_label1a = label.Label(
    font=font, text=str(sparkline1.y_top), color=0xFFFFFF
)  # y_top label
text_label1a.anchor_point = (0, 0.5)  # set the anchorpoint
text_label1a.anchored_position = (
    10 + chart_width,
    10,
)  # set the text anchored position to the upper right of the graph

text_label1b = label.Label(
    font=font, text=str(sparkline1.y_bottom), color=0xFFFFFF
)  # y_bottom label
text_label1b.anchor_point = (0, 0.5)  # set the anchorpoint
text_label1b.anchored_position = (
    10 + chart_width,
    10 + chart_height,
)  # set the text anchored position to the upper right of the graph


# Setup the second bitmap and sparkline
# sparkline2 uses a vertical y range between 0 to 1, and will contain a
# maximum of 10 items
#
palette2 = displayio.Palette(1)  # color palette used for bitmap2 (one color)
palette2[0] = 0x0000FF

bitmap2 = displayio.Bitmap(chart_width * 2, chart_height * 2, 1)  # create bitmap2
tilegrid2 = displayio.TileGrid(
    bitmap2, pixel_shader=palette2, x=150, y=10
```

(continues on next page)

(continued from previous page)

```
)  # Add bitmap2 to tilegrid2
sparkline2 = Sparkline(
    width=chart_width * 2,
    height=chart_height * 2,
    max_items=10,
    y_min=0,
    y_max=1,
    x=150,
    y=10,
    color=0xFF00FF,
)


# Setup the third bitmap and third sparkline
# sparkline3 contains a maximum of 10 items
# since y_min and y_max are not specified, sparkline3 uses autoranging for both
# the top and bottom of the y-axis.
# Note1: Any unspecified edge limit (y_min or y_max) will autorange that edge based
# on the data in the list.
# Note2: You can read back the current value of the y-axis limits by using
# sparkline3.y_bottom or sparkline3.y_top


palette3 = displayio.Palette(1)  # color palette used for bitmap (one color)
palette3[0] = 0x11FF44
bitmap3 = displayio.Bitmap(DISPLAY_WIDTH - 30, chart_height * 2, 1)  # create bitmap3
tilegrid3 = displayio.TileGrid(
    bitmap3, pixel_shader=palette3, x=0, y=120
)  # Add bitmap3 to tilegrid3

sparkline3 = Sparkline(
    width=DISPLAY_WIDTH - 30,
    height=chart_height * 2,
    max_items=10,
    x=0,
    y=120,
    color=0xFFFFFF,
)

# Initialize the y-axis labels for mySparkline3 with no text
text_label3a = label.Label(font=font, text="", color=0x11FF44)  # y_top label
text_label3a.anchor_point = (0, 0.5)  # set the anchorpoint
text_label3a.anchored_position = (
    sparkline3.width,
    120,
)  # set the text anchored position to the upper right of the graph

text_label3b = label.Label(font=font, text="", color=0x11FF44)  # y_bottom label
text_label3b.anchor_point = (0, 0.5)  # set the anchorpoint
text_label3b.anchored_position = (
    sparkline3.width,
    120 + sparkline3.height,
)  # set the text anchored position to the upper right of the graph

# Create a group to hold the three bitmap TileGrids and the three sparklines and
# append them into the group (my_group)
#
```

(continues on next page)

(continued from previous page)

```
# Note: In cases where display elements will overlap, then the order the elements
# are added to the group will set which is on top.  Latter elements are displayed
# on top of former elements.
my_group = displayio.Group()

my_group.append(sparkline1)
my_group.append(text_label1a)
my_group.append(text_label1b)

my_group.append(tilegrid2)
my_group.append(sparkline2)

my_group.append(tilegrid3)
my_group.append(sparkline3)
my_group.append(text_label3a)
my_group.append(text_label3b)

# Set the display to show my_group that contains all the bitmap TileGrids and
# sparklines
display.show(my_group)

i = 0  # This is a counter for changing the random values for mySparkline3

# Start the main loop
while True:

    # Turn off auto_refresh to prevent partial updates of the screen during updates
    # of the sparklines
    display.auto_refresh = False

    # add_value: add a new value to a sparkline
    # Note: The y-range for sparkline1 is set to -1 to 1.25, so all these random
    # values (between 0 and 1) will fit within the visible range of this sparkline
    sparkline1.add_value(random.uniform(0, 1))

    # Note: For sparkline2, the y-axis range is set from 0 to 1.
    # With the random values set between -1 and +2, the values will sometimes
    # be out of the y-range.  This example shows how the fixed y-range (0 to 1)
    # will "clip" values (it will not display them) that are above or below the
    # y-range.
    sparkline2.add_value(random.uniform(-1, 2))

    # sparkline3 is set autoranging for both the top and bottom of the Y-axis

    # In this example, for 15 values, this adds points in the range from 0 to 1.
    # Then, for the next 15 values, it adds points in the range of 0 to 10.
    # This is to highlight the autoranging of the y-axis.
    # Notice how the y-axis labels show that the y-scale is changing.
    #
    # An exercise for the reader: You can set only one or the other sparkline axis
    # to autoranging by setting its value to None.
    if i < 15:
        sparkline3.add_value(random.uniform(0, 1))
    else:
        sparkline3.add_value(random.uniform(0, 10))
    text_label3a.text = str(sparkline3.y_top)
    text_label3b.text = str(sparkline3.y_bottom)
```

(continues on next page)

(continued from previous page)

```
    i += 1  # increment the counter
    if i > 30:  # After 30 times through the loop, reset the counter
        i = 0

    # Turn on auto_refresh for the display
    display.auto_refresh = True

    # The display seems to be less jittery if a small sleep time is provided
    # You can adjust this to see if it has any effect
    time.sleep(0.01)
```

6.6 Circle Animation

Example showing the features of the new Circle setter

Listing 6: examples/display_shapes_circle_animation.py

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

"""
This is an animation to demonstrate the use of Circle Setter Attribute.
"""

import time
import gc
import board
import displayio
from adafruit_display_shapes.circle import Circle

# use built in display (MagTag, PyPortal, PyGamer, PyBadge, CLUE, etc.)
# see guide for setting up external displays (TFT / OLED breakouts, RGB matrices, etc.)
# https://learn.adafruit.com/circuitpython-display-support-using-displayio/display-and-display-bus
display = board.DISPLAY

# Make the display context
main_group = displayio.Group()

# Make a background color fill
color_bitmap = displayio.Bitmap(display.width, display.height, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0xFFFFFF
bg_sprite = displayio.TileGrid(color_bitmap, pixel_shader=color_palette, x=0, y=0)
main_group.append(bg_sprite)

# Setting up the Circle starting position
posx = 50
posy = 50

# Define Circle characteristics
circle_radius = 20
circle = Circle(posx, posy, circle_radius, fill=0x00FF00, outline=0xFF00FF)
```

(continues on next page)

(continued from previous page)

```
main_group.append(circle)

# Define Circle Animation Steps
delta_x = 2
delta_y = 2

# Showing the items on the screen
display.show(main_group)

while True:

    if circle.y + circle_radius >= display.height - circle_radius:
        delta_y = -1
    if circle.x + circle_radius >= display.width - circle_radius:
        delta_x = -1
    if circle.x - circle_radius <= 0 - circle_radius:
        delta_x = 1
    if circle.y - circle_radius <= 0 - circle_radius:
        delta_y = 1

    circle.x = circle.x + delta_x
    circle.y = circle.y + delta_y

    time.sleep(0.02)
    gc.collect()
```

6.7 circle

Various common shapes for use with displayio - Circle shape!

- Author(s): Limor Fried

6.7.1 Implementation Notes

Software and Dependencies:

- Adafruit CircuitPython firmware for the supported boards: https://github.com/adafruit/circuitpython/releases

class adafruit_display_shapes.circle.**Circle**(*x0*, *y0*, *r*, ***, *fill=None*, *outline=None*, *stroke=1*)

A circle.

Parameters

- **x0** – The x-position of the center.
- **y0** – The y-position of the center.
- **r** – The radius of the circle.
- **fill** – The color to fill the circle. Can be a hex value for a color or `None` for transparent.
- **outline** – The outline of the circle. Can be a hex value for a color or `None` for no outline.
- **stroke** – Used for the outline. Will not change the radius.

x0

The x-position of the center of the circle.

y0
The y-position of the center of the circle.

6.8 rect

Various common shapes for use with displayio - Rectangle shape!

- Author(s): Limor Fried

6.8.1 Implementation Notes

Software and Dependencies:

- Adafruit CircuitPython firmware for the supported boards: https://github.com/adafruit/circuitpython/releases

class adafruit_display_shapes.rect.**Rect**(*x*, *y*, *width*, *height*, ***, *fill=None*, *outline=None*, *stroke=1*)

A rectangle.

Parameters

- **x** – The x-position of the top left corner.
- **y** – The y-position of the top left corner.
- **width** – The width of the rectangle.
- **height** – The height of the rectangle.
- **fill** – The color to fill the rectangle. Can be a hex value for a color or `None` for transparent.
- **outline** – The outline of the rectangle. Can be a hex value for a color or `None` for no outline.
- **stroke** – Used for the outline. Will not change the outer bound size set by `width` and `height`.

fill
The fill of the rectangle. Can be a hex value for a color or `None` for transparent.

outline
The outline of the rectangle. Can be a hex value for a color or `None` for no outline.

6.9 roundrect

A slightly modified version of Adafruit_CircuitPython_Display_Shapes that includes an explicit call to palette.make_opaque() in the fill color setter function.

class adafruit_display_shapes.roundrect.**RoundRect**(*x*, *y*, *width*, *height*, *r*, ***, *fill=None*, *outline=None*, *stroke=1*)

A round-corner rectangle.

Parameters

- **x** – The x-position of the top left corner.
- **y** – The y-position of the top left corner.

- **width** – The width of the rounded-corner rectangle.
- **height** – The height of the rounded-corner rectangle.
- **r** – The radius of the rounded corner.
- **fill** – The color to fill the rounded-corner rectangle. Can be a hex value for a color or `None` for transparent.
- **outline** – The outline of the rounded-corner rectangle. Can be a hex value for a color or `None` for no outline.
- **stroke** – Used for the outline. Will not change the outer bound size set by `width` and `height`.

fill
The fill of the rounded-corner rectangle. Can be a hex value for a color or `None` for transparent.

outline
The outline of the rounded-corner rectangle. Can be a hex value for a color or `None` for no outline.

6.10 triangle

Various common shapes for use with displayio - Triangle shape!

- Author(s): Melissa LeBlanc-Williams

6.10.1 Implementation Notes

Software and Dependencies:

- Adafruit CircuitPython firmware for the supported boards: https://github.com/adafruit/circuitpython/releases

class adafruit_display_shapes.triangle.**Triangle**(*x0*, *y0*, *x1*, *y1*, *x2*, *y2*, ***, *fill=None*, *outline=None*)

A triangle.

Parameters

- **x0** – The x-position of the first vertex.
- **y0** – The y-position of the first vertex.
- **x1** – The x-position of the second vertex.
- **y1** – The y-position of the second vertex.
- **x2** – The x-position of the third vertex.
- **y2** – The y-position of the third vertex.
- **fill** – The color to fill the triangle. Can be a hex value for a color or `None` for transparent.
- **outline** – The outline of the triangle. Can be a hex value for a color or `None` for no outline.

fill
The fill of the triangle. Can be a hex value for a color or `None` for transparent.

6.11 line

Various common shapes for use with displayio - Line shape!

- Author(s): Melissa LeBlanc-Williams

6.11.1 Implementation Notes

Software and Dependencies:

- Adafruit CircuitPython firmware for the supported boards: https://github.com/adafruit/circuitpython/releases

class adafruit_display_shapes.line.**Line**(*x0*, *y0*, *x1*, *y1*, *color*)
A line.

Parameters

- **x0** – The x-position of the first vertex.
- **y0** – The y-position of the first vertex.
- **x1** – The x-position of the second vertex.
- **y1** – The y-position of the second vertex.
- **color** – The color of the line.

6.12 polygon

Various common shapes for use with displayio - Polygon shape!

- Author(s): Melissa LeBlanc-Williams

6.12.1 Implementation Notes

Software and Dependencies:

- Adafruit CircuitPython firmware for the supported boards: https://github.com/adafruit/circuitpython/releases

class adafruit_display_shapes.polygon.**Polygon**(*points*, *, *outline=None*)
A polygon.

Parameters

- **points** – A list of (x, y) tuples of the points
- **outline** – The outline of the polygon. Can be a hex value for a color or `None` for no outline.

outline
The outline of the polygon. Can be a hex value for a color or `None` for no outline.

6.13 sparkline

Various common shapes for use with displayio - Sparkline!

- Author(s): Kevin Matocha

6.13.1 Implementation Notes

Software and Dependencies:

- Adafruit CircuitPython firmware for the supported boards: https://github.com/adafruit/circuitpython/releases

class adafruit_display_shapes.sparkline.**Sparkline**(*width*, *height*, *max_items*, *y_min=None*, *y_max=None*, *x=0*, *y=0*, *color=16777215*)

A sparkline graph.

Parameters

- **width** – Width of the sparkline graph in pixels
- **height** – Height of the sparkline graph in pixels
- **max_items** – Maximum number of values housed in the sparkline
- **y_min** – Lower range for the y-axis. Set to None for autorange.
- **y_max** – Upper range for the y-axis. Set to None for autorange.
- **x** – X-position on the screen, in pixels
- **y** – Y-position on the screen, in pixels
- **color** – Line color, the default value is 0xFFFFFF (WHITE)

add_value(*value*)

Add a value to the sparkline. :param value: The value to be added to the sparkline

clear_values()

Removes all values from the _spark_list list and removes all lines in the group

update()

Update the drawing of the sparkline.

values()

Returns the values displayed on the sparkline.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index