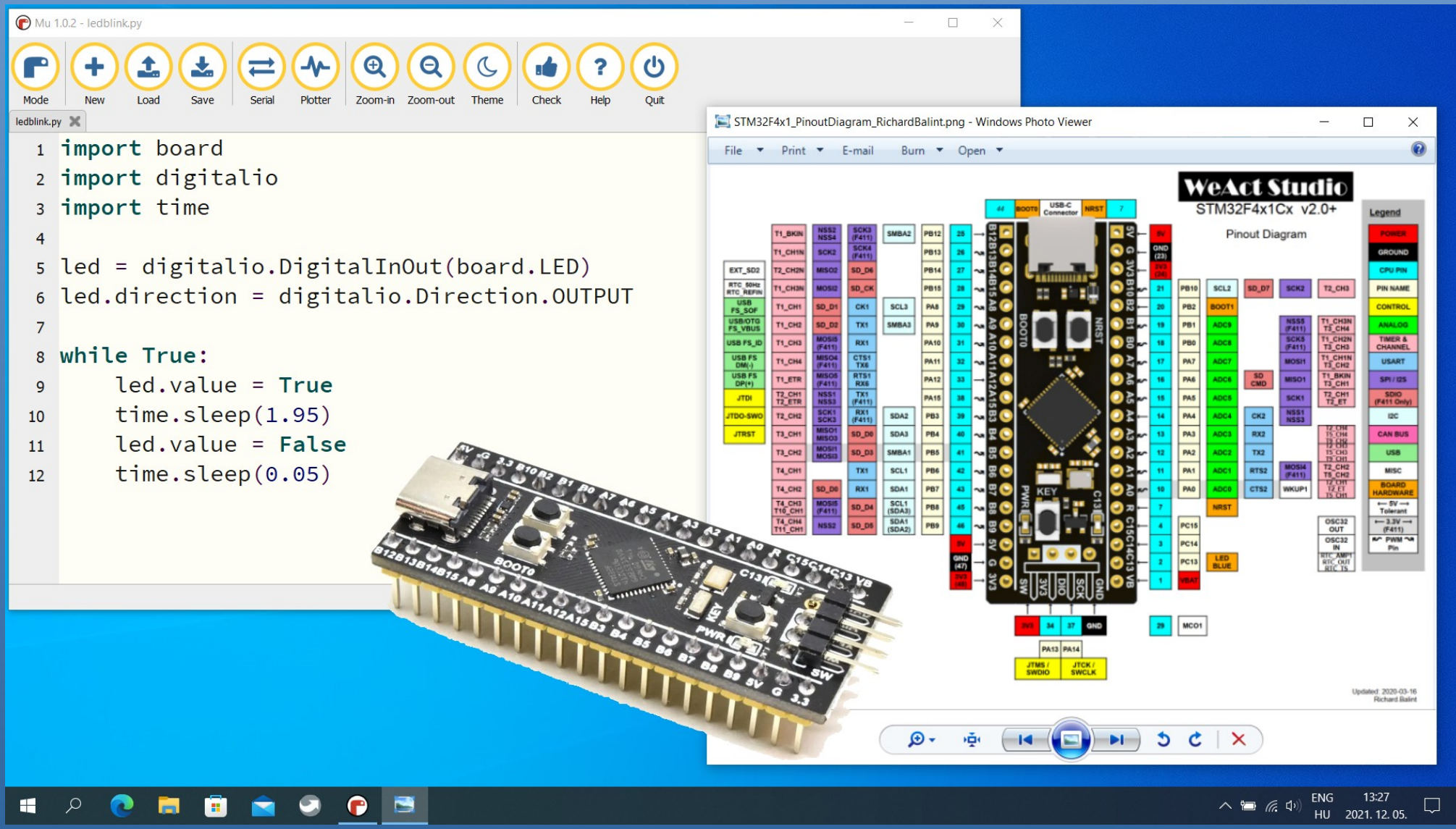
CircuitPython tanfolyam



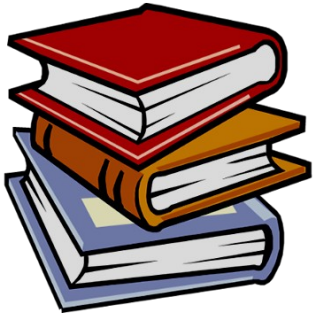
4. Az ST7735 színes TFT kijelző

Felhasznált és ajánlott irodalom

Python:

- Mark Pilgrim/Kelemen Gábor: Ugorj fejest a Python 3-ba!
- P. Wentworth et al. (ford. Biró Piroska, Szeghalmy Szilvia és Varga Imre): Hogyan gondolkozz úgy, mint egy informatikus: Tanulás Python 3 segítségével

CircuitPython:



- Adafruit: https://circuitpython.org/downloads
- Learn Adafruit: Welcome to CircuitPython
- Learn Adafruit: CircuitPython Essentials
- Adafruit: Adafruit CircuitPython API Reference
- Adafruit: github.com/adafruit/Adafruit CircuitPython Bundle

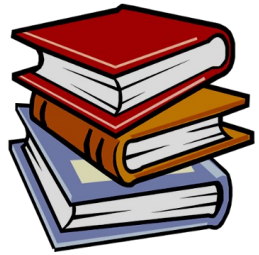
Adatlapok és dokumentáció:

- **STM32F411CE** adatlap és termékinfo
- **STM32F411xC/E** Family Reference Manual
- **WeAct Studio:** STM32F4x1 MiniF4

Felhasznált és ajánlott irodalom

Tananyagok, útmutatók:

- Carter Nelson: CircuitPython Display Support Using displayio
- Tim C.: CircuitPython Display_Text Library
- Ruiz Brothers: Custom Fonts for CircuitPython Displays
- Anna Barela: Creating Slideshows in CircuitPython



CircuitPython kiegészítő programkönyvtárak:

- Adafruit_ST7735R – displayio driver for the ST7735R 1.8” TFT
- Adafruit_CircuitPython_Display_Text – text labels
- Adafruit_CircuitPython_Bitmap_Font – custom font support
- Adafruit_CircuitPython_ImageLoad – load image files
- Adafruit_CircuitPython_Display_Shapes – lines, circles, triangles etc.
- Adafruit_Slideshow – helper library for displaying a slideshow

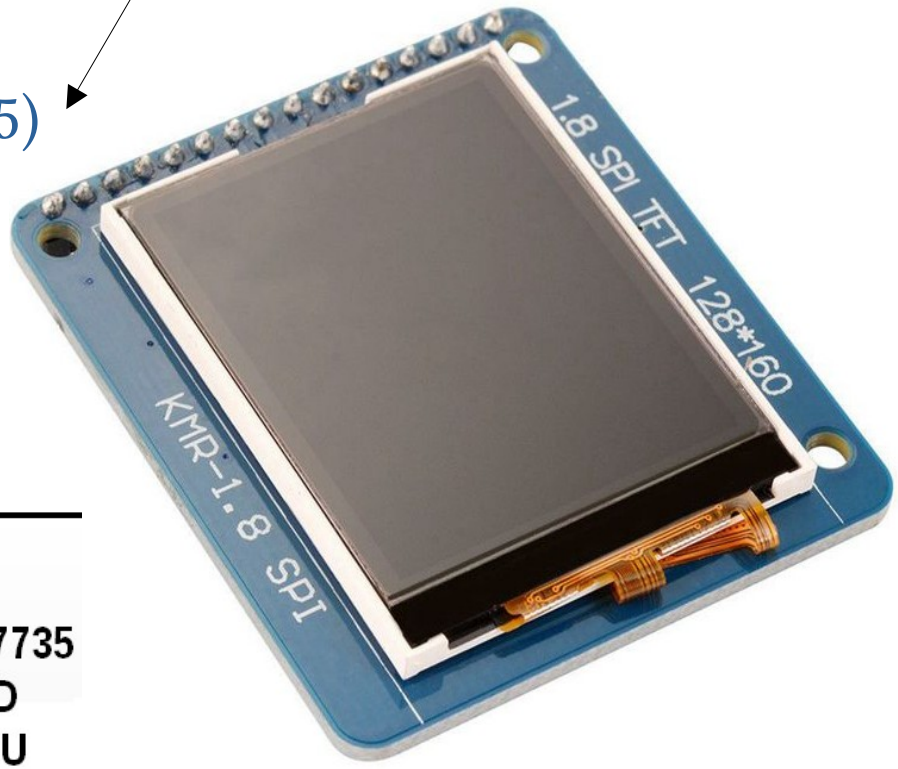
Adatlapok és dokumentáció:

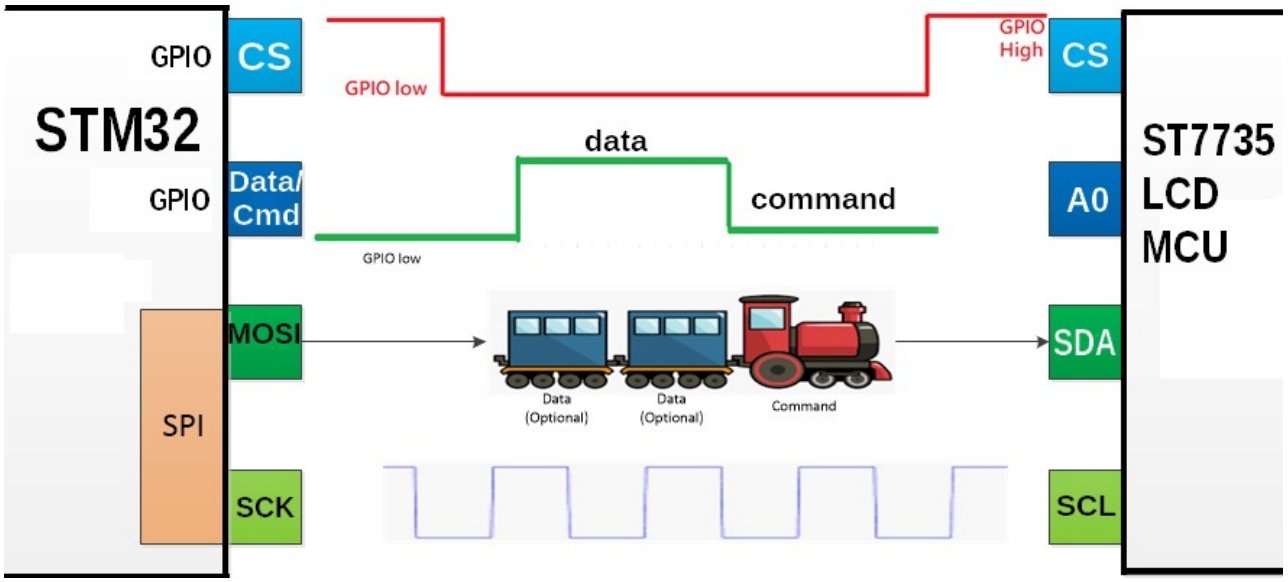
- Sitronix: ST7735 adatlap

ST7735 1.8” színes kijelző

- Vezérlő: **Sitronix ST7735**
- Interfész: SPI, csak írás (max. 10 – 15 MHz)
- Data/Command választás külön vonalon
- Kijelző: TFT, 65 536 szín (16 bit RGB, 5-6-5)
- Felbontás: **128 x 160** pixel
- SD kártya foglalat (SPI mód)
- Háttérvilágítás (LED anód/katód)

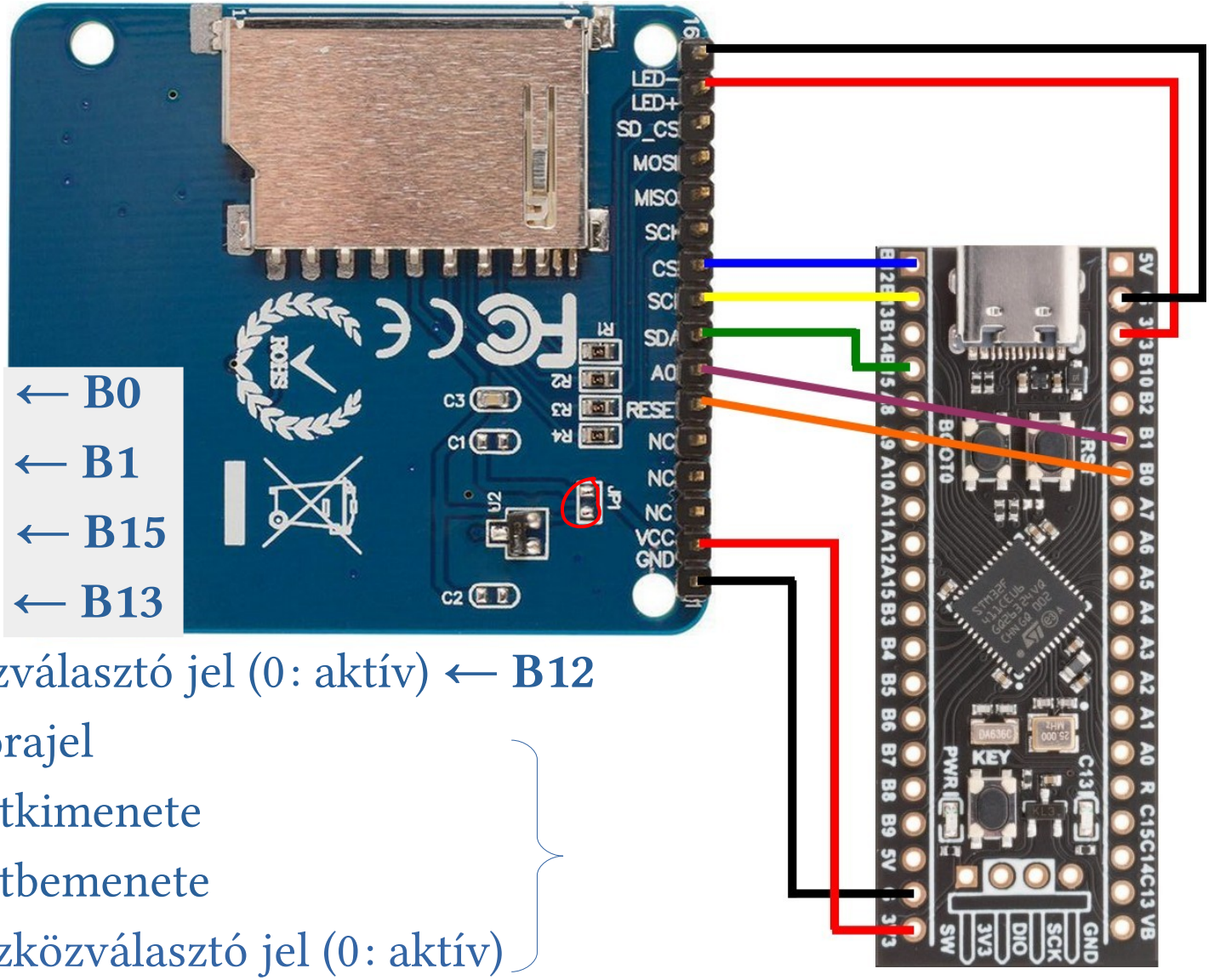
12-bites, 16 bites és 18 bites módban is használható





ST7735 kijelző bekötési vázlat

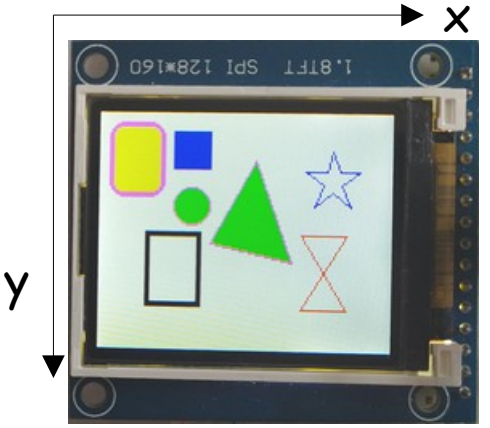
- **GND** – a tápegység közös pontja
- **VCC** – tápfeszültség 5V / 3.3V (JP1)
- **NC** – nincs bekötve
- **$\overline{\text{RESET}}$** – HW reset ← **B0**
- **A0** – adat/parancs ← **B1**
- **SDA** – SPI adatvonal ← **B15**
- **SCL** – SPI órajel ← **B13**
- **$\overline{\text{CS}}$** – kijelző SPI eszközválasztó jel (0: aktív) ← **B12**
- **SCK** – SD kártya SPI órajel
- **MISO** – SD kártya adatkimenete
- **MOSI** – SD kártya adatbemenete
- **$\overline{\text{SD\_CS}}$** – SD kártya eszközválasztó jel (0: aktív)
- **LED+** – kijelző háttérvilágítás (anód) ← **3V3**
- **LED–** – kijelző háttérvilágítás (katód) ← **GND**



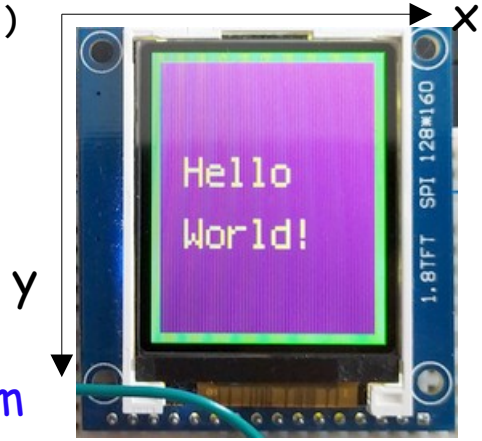
Az Adafruit_ST7735R programkönyvtár

- Az Adafruit_ST7735R programkönyvtár elnevezése nem következetes, valójában ez egy ***displayio*** meghajtó
  (**R** betű nélküli verziója is van, de az az **ST7735B** vezérlőhöz való)
- Dokumentáció: displayio driver for ST7735R TFT-LCD displays
- Inicializálás:

```
import board
import busio
import displayio
from adafruit_st7735r import ST7735R
# Release any resources currently in use for the displays
displayio.release_displays()

# create the spi device and pins we will need
spi = busio.SPI(board.B13, MOSI=board.B15)
display_bus = displayio.FourWire(spi,command=board.B1,chip_select=board.B12,reset=board.B0)
display=ST7735R(display_bus,width=160,height=128,rotation=90,bgr=True)

Vagy álló formátumhoz:
display=ST7735R(display_bus,width=128,height=160,rotation=0,bgr=True)
```



Fekvő formátum

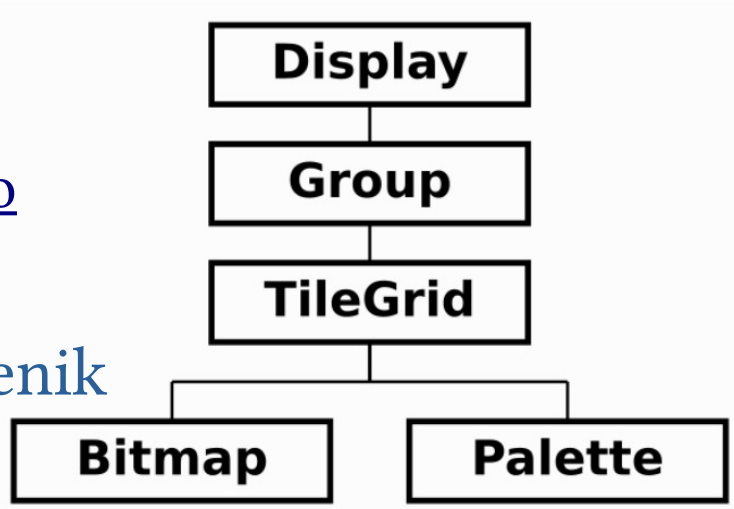


Álló formátum

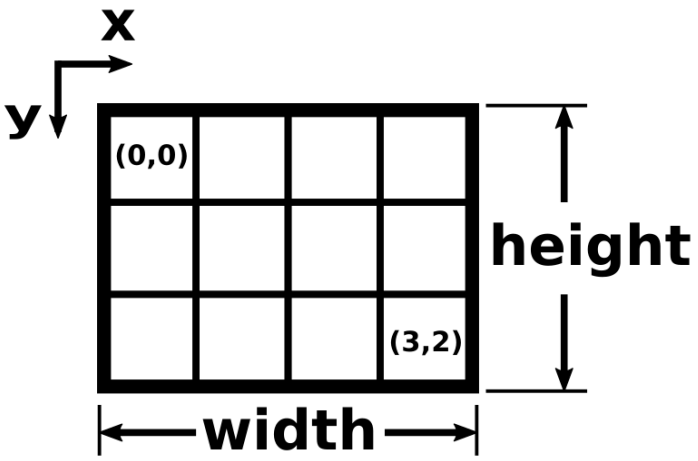
- Az **Adafruit_ST7735R** könyvtár nem beépített könyvtár, nekünk kell telepíteni

A displayio modul osztályai

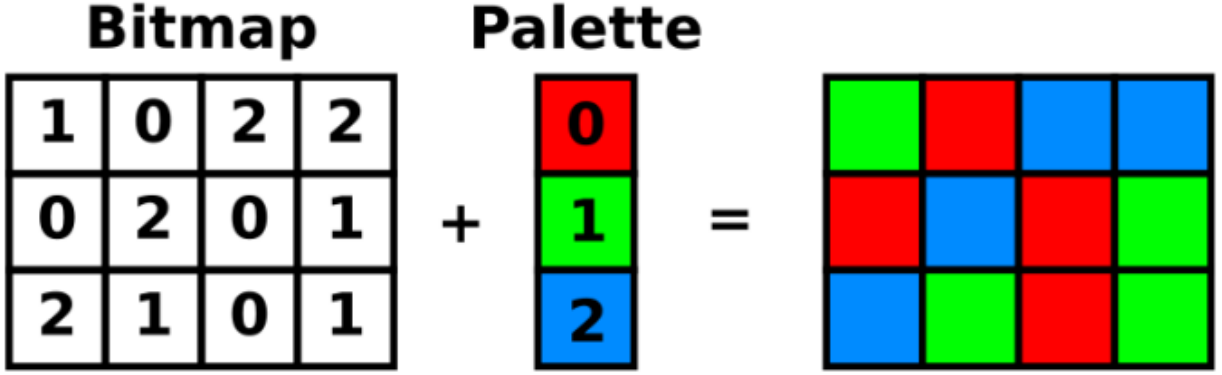
- A **displayio** modul API dokumentációja
- Adafruit *displayio* tutorial:
  CircuitPython Display Support Using displayio
- A **Group** (vagy a legfelső Group) végeredményben az, ami a kijelzőn megjelenik a **show()** metódus meghívásakor
- A **TileGrid Bitmap** és **Palette** elemekből áll



Koordinátarendszer

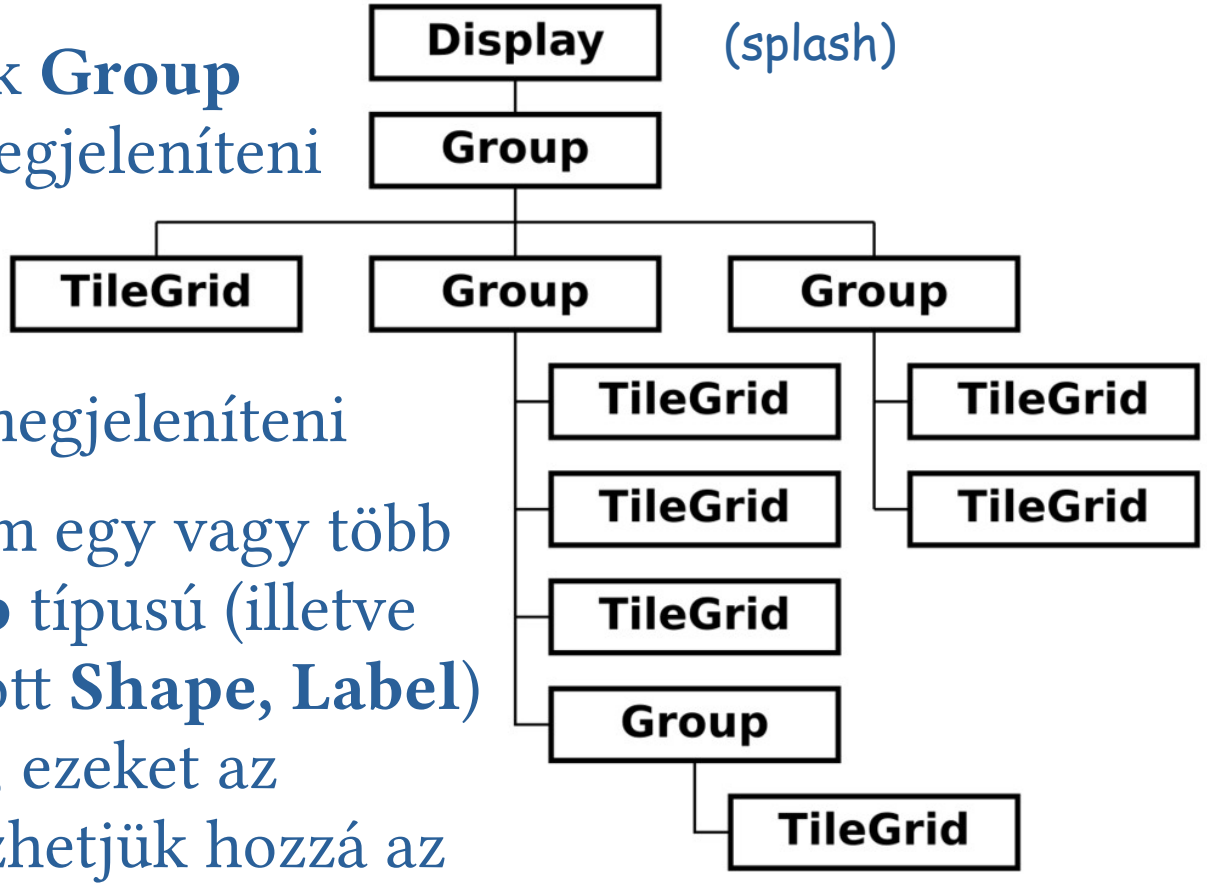


Bitmap és **Palette** kapcsolata:



Hierarchia és összeillesztés

- A **displayio** modul osztályaiból példányosított objektumokból komplex ábrákat is összerakhatunk de be kell tartanunk az alábbi szabályokat:
- A **Display** objektum csak **Group** típusú objektumot tud megjeleníteni
- A **Display** objektum egyidejűleg csak egy **Group** objektumot tud megjeleníteni
- A **Group** típusú objektum egy vagy több **TileGrid** és/vagy **Group** típusú (illetve a **Group**-ból származtatott **Shape, Label**) objektumot tartalmazhat, ezeket az **append()** metódussal fűzhetjük hozzá az általános gyűjtőként funkcionáló, a hierarchia tetején ülő **Group** objektumhoz



Szövegrajzolás – Text Label

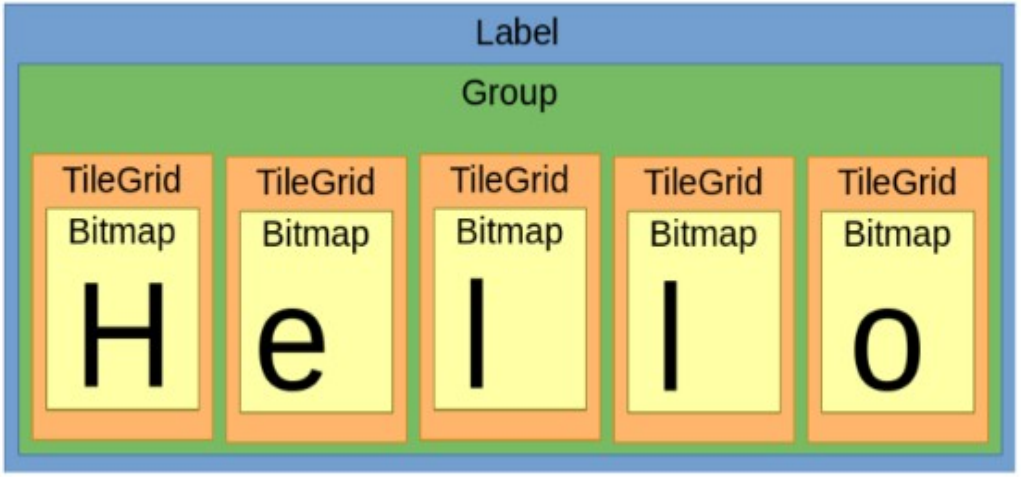
- Szövegcímke (Text Label) rajzolása:

```
text_area=label.Label(terminalio.FONT, text="Hello",color=0x00,x=10,y=15)
splash.append(text_area)
```

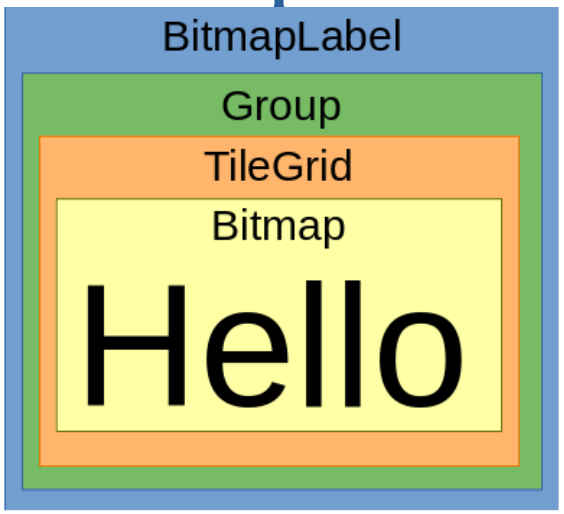
- A CircuitPython Display_Text Library tananyag szerint az **adafruit_display_text modul** kétféle szövegcímke típust kezel, ezek közül importáláskor pl. így lehet választani:

```
from adafruit_display_text import label
```
vagy
```
from adafruit_display_text import bitmap_label as label
```

Label (hagyományos)



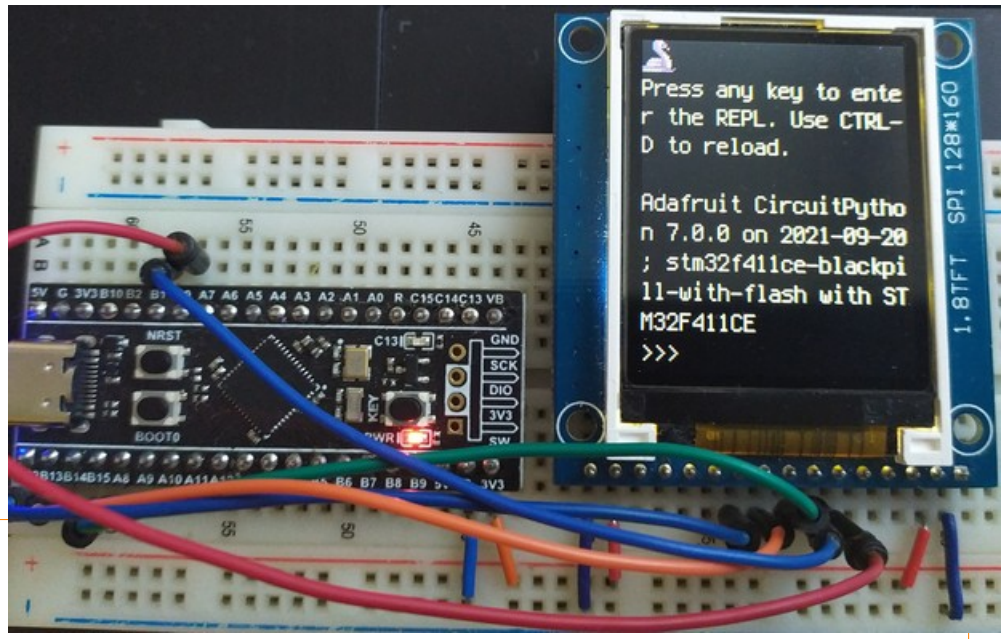
BitmapLabel



- A beépített terminalio.FONT helyett más fontot is használhatunk, kezelésükhöz az adafruit_bitmap_font könyvtárra lesz szükségünk

ST7735R_simpletest.py 2/1.

- Az első mintapélda első részében importáljuk a felhasználni kívánt modulokat és inicializáljuk az SPI 2. csatornát, majd **displayio** kijelzőként inicializáljuk az **ST7735R** képernyőt is




```
import board
import terminalio
import displayio
import busio
from adafruit_display_text import label
from adafruit_st7735r import ST7735R

# Release any resources currently in use for the displays
displayio.release_displays()

# create the spi device and pins we will need
spi = busio.SPI(board.B13, MOSI=board.B15, MISO=board.B14)
tft_cs = board.B12
tft_dc = board.B1
tft_rst = board.B0
display_bus = displayio.FourWire(spi, command=tft_dc, chip_select=tft_cs, reset=tft_rst)
display = ST7735R(display_bus, width=128, height=160, rotation=0, bgr=True)
```

ST7735R_simpletest.py 2/2.

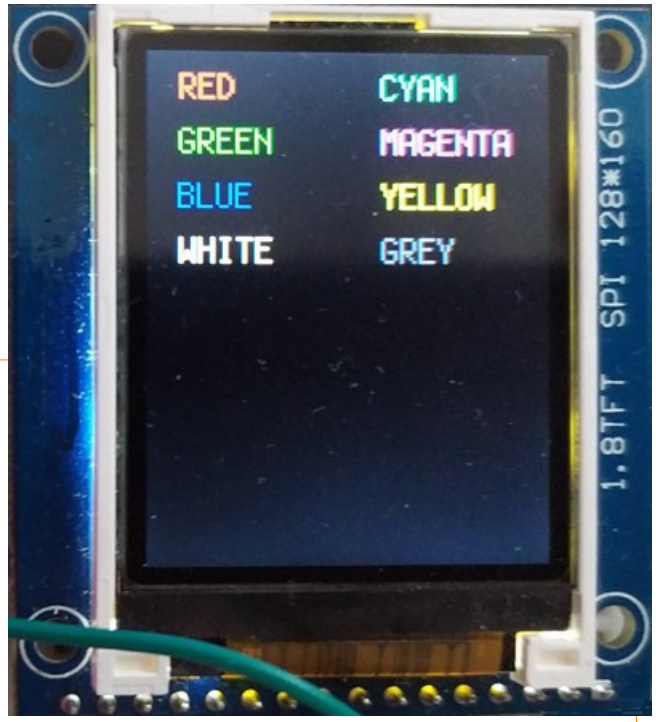
```
# Make the display context
splash = displayio.Group()
display.show(splash)

color_bitmap = displayio.Bitmap(128, 160, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0x00FF00  # Bright Green
bg_sprite = displayio.TileGrid(color_bitmap,
        pixel_shader=color_palette, x=0, y=0)
splash.append(bg_sprite)

# Draw a smaller inner rectangle
inner_bitmap = displayio.Bitmap(118,150,1)
inner_palette = displayio.Palette(1)
inner_palette[0] = 0xAA0088  # Purple
inner_sprite = displayio.TileGrid(inner_bitmap, pixel_shader=inner_palette, x=5, y=5)
splash.append(inner_sprite)

# Draw a label
text_group = displayio.Group(scale=2, x=18, y=64)
text = "Hello \r\nWorld!"
text_area = label.Label(terminalio.FONT, text=text, color=0xFFFF00)
text_group.append(text_area)  # Subgroup for text scaling
splash.append(text_group)

while True:
    pass
```

A szövegcímkét egy Group-ba csomagoljuk, hogy átskálázhassuk

Itt, a második részben kirajzolunk két Bitmap téglalapot (ez adja a keretet) és egy átskálázott szövegecímkét

ST7735R_color_labels.py 2/1.

- Ebben a példában 4-4 szövegcímkét fűzünk egy-egy (al)csoportba, s eközben bátran tobzódunk a színekben…
- Felhasznált forrás: st7735r_colored_labels




```
import board
import terminalio
import displayio
import busio
from adafruit_display_text import label
from adafruit_st7735r import ST7735R

# Release any resources currently in use for the displays
displayio.release_displays()

# create the spi device and pins we will need
spi = busio.SPI(board.B13, MOSI=board.B15, MISO=board.B14)
tft_cs = board.B12
tft_dc = board.B1
tft_rst = board.B0
display_bus = displayio.FourWire(spi, command=tft_dc, chip_select=tft_cs, reset=tft_rst)
display = ST7735R(display_bus, width=128, height=160, rotation=0, bgr=True)

splash = displayio.Group()
display.show(splash)
```

ST7735R_color_labels.py 2/2.

```
# write some text in each font color, rgb, cmyk
text_group_left = displayio.Group(scale=1, x=10, y=10)
text_area_red = label.Label(terminalio.FONT, text="RED", color=0xFF0000)
text_area_green = label.Label(terminalio.FONT, text="\nGREEN", color=0x00FF00)
text_area_blue = label.Label(terminalio.FONT, text="\n\nBLUE", color=0x0000FF)
text_area_white = label.Label(terminalio.FONT, text="\n\n\nWHITE", color=0xFFFFFF)
text_group_left.append(text_area_red)
text_group_left.append(text_area_green)
text_group_left.append(text_area_blue)
text_group_left.append(text_area_white)
splash.append(text_group_left)
```


```
text_group_right = displayio.Group(scale=1, x=74, y=10)
text_area_cyan = label.Label(terminalio.FONT, text="CYAN", color=0x00FFFF)
text_group_right.append(text_area_cyan)
text_area_magenta = label.Label(terminalio.FONT, text="\nMAGENTA", color=0xFF00FF)
text_group_right.append(text_area_magenta)
text_area_yellow = label.Label(terminalio.FONT, text="\n\nYELLOW", color=0xFFFF00)
text_group_right.append(text_area_yellow)
text_area_black = label.Label(terminalio.FONT, text="\n\n\nGREY", color=0x808080)
text_group_right.append(text_area_black)
splash.append(text_group_right)

while True:
    pass
```

Bitmap fontok használata

- A **displayio** kompatibilis kijelzőkön tetszés szerinti fontokat is használhatunk, egy lehetőséget az alábbi példában mutatunk be, ahol a kirajzolandó szöveg egy **Text Label** elemként kezelhető
- Most az első (Hello World!) mintapéldánkat módosítjuk az **adafruit_display_text** modul display_text_bitmap_label_simpletest.py mintapéldájából kiindulva
- Az alábbi kiegészítő könyvtárakra lesz szükségünk:
- Adafruit_CircuitPython_Display_Text - ebből a ***bitmap_label*** osztály kell
- Adafruit_CircuitPython_Bitmap_Font - ebből a ***bimap_font*** osztály kell
- Természetesen kell az Adafruit_ST7735R programkönyvtár is
- A **/fonts** mappába pedig be kell másolnunk a **Chicago-12.bdf**, valamint a **LeagueSpartan-Bold-16.bdf** fontokat
- A kijelző inicializálása megegyezik a korábbiakkal, így a következő oldalon csak a programbeli eltéréseket mutatjuk be

ST7735R_bitmap_labels.py (részletek)


```
from adafruit_display_text import bitmap_label as label
from adafruit_bitmap_font import bitmap_font
...
small_font = bitmap_font.load_font('fonts/Chicago-12.bdf')
big_font = bitmap_font.load_font('fonts/LeagueSpartan-Bold-16.bdf')
splash = displayio.Group()
display.show(splash)

color_bitmap = displayio.Bitmap(128, 160, 1)  # Green background
color_palette = displayio.Palette(1)
color_palette[0] = 0x00FF00  # Bright Green
bg_sprite = displayio.TileGrid(color_bitmap, pixel_shader=color_palette, x=0, y=0)
splash.append(bg_sprite)

inner_bitmap = displayio.Bitmap(118,150,1)    # the inner rectangle
inner_palette = displayio.Palette(1)
inner_palette[0] = 0xAA0088  # Purple
inner_sprite = displayio.TileGrid(inner_bitmap, pixel_shader=inner_palette, x=5, y=5)
splash.append(inner_sprite)

text_group = displayio.Group(scale=2, x=5, y=64) # Draw text as two labels
label1 = label.Label(big_font, text="Hello", color=0xFFFF00,x=3,y=0)       # yellow
label2 = label.Label(small_font, text="World!", color=0xFFFFFF,x=10, y=20) # white
text_group.append(label1)  # Subgroup for text scaling
text_group.append(label2)  # Subgroup for text scaling
splash.append(text_group)
while True:
    pass
```

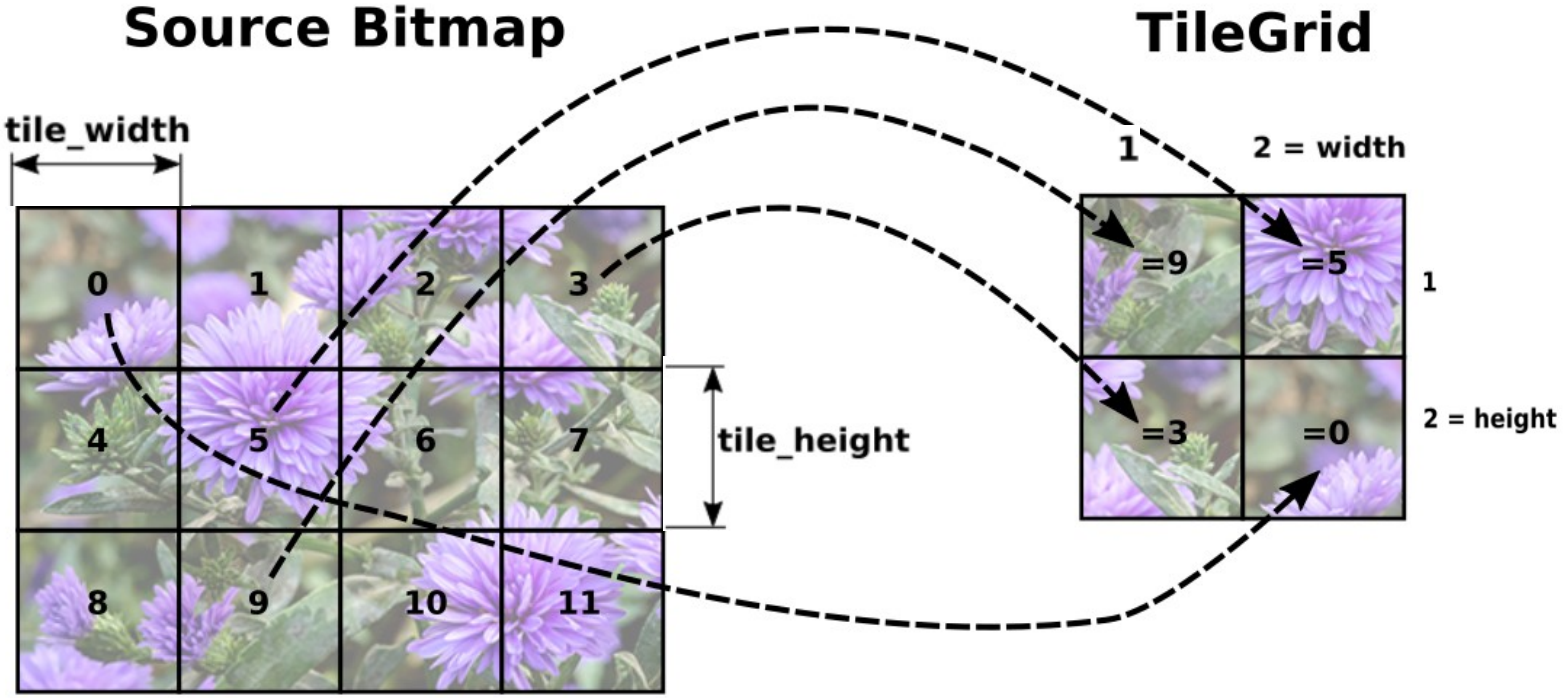
A TileGrid osztály

- A **TileGrid** osztály egy forrásként használt bitkép szeleteit rendezi 2D rácsozatba. A TileGrid (csemperács) egy, vagy több „csempéből” állhat *(ha csak egyetlen csempe van, sprite-nak hívjuk)*

  TileGrid(*bitmap, palette, width, height, tile_width, tile_height,default, x, y)*

- Ha definiáltuk a csempék méretét és a csemperács méretét, akkor sorszámuk alapján összerendelhetjük a csempéket a bitkép szeleteivel

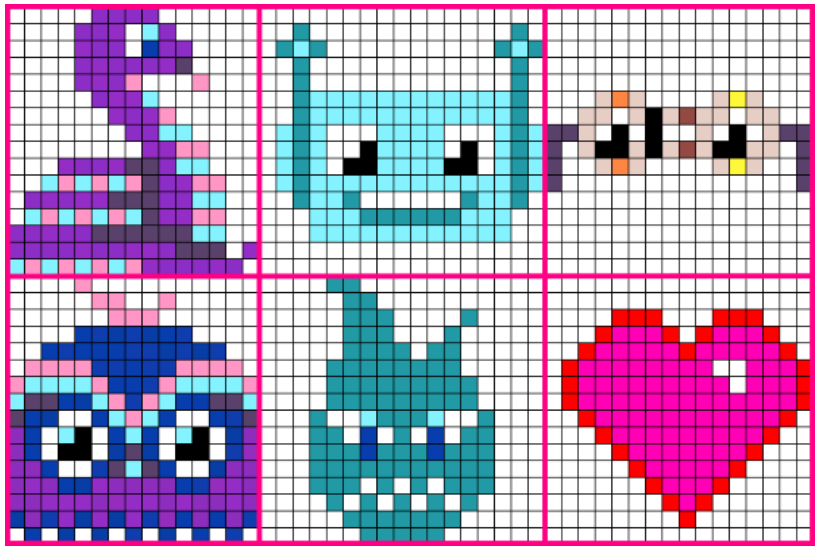
```
tile_grid[0] = 9
tile_grid[1] = 5
tile_grid[2] = 3
tile_grid[3] = 0
```



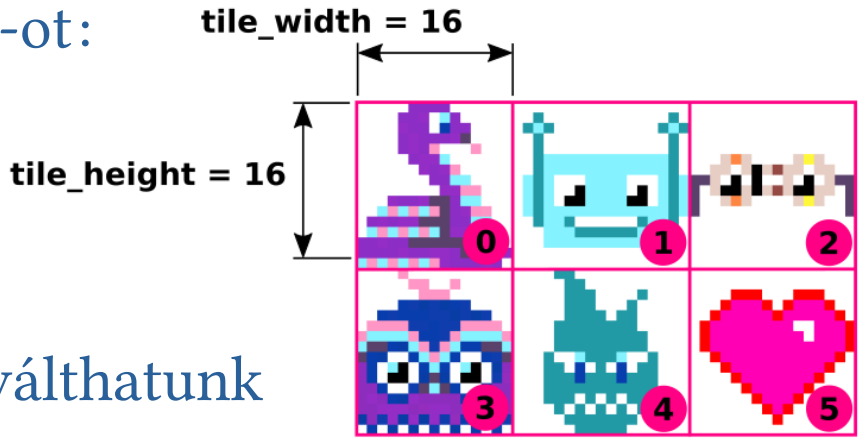
Játsszunk a sprite-okkal!

- Töltsük le a **cp_sprite_sheet.bmp** bitmap állományt és telepítsük az **Adafruit_ImageLoad** programkönyvtárat!

```
# Load the sprite sheet (bitmap)
sprite_sheet, palette = adafruit_imageload.load(
  "/cp_sprite_sheet.bmp", bitmap=displayio.Bitmap,
  palette=displayio.Palette)

# Create a sprite (tilegrid)
sprite = displayio.TileGrid(sprite_sheet,
  pixel_shader=palette,width = 1, height = 1,
  tile_width = 16, tile_height = 16)
```



- A képfájl betöltése és egy sprite kijelölése után a sprite-ot:
  - megjeleníthetjük
  - nagyíthatjuk
  - mozgathatjuk
  - másik képszeletre válthatunk



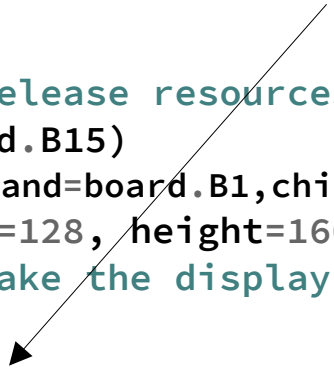
Sprite Sheet
(Source Bitmap)

Sprite
(TileGrid)

- Felhasznált forrás: Sprite sheet

ST7735R_sprite.py

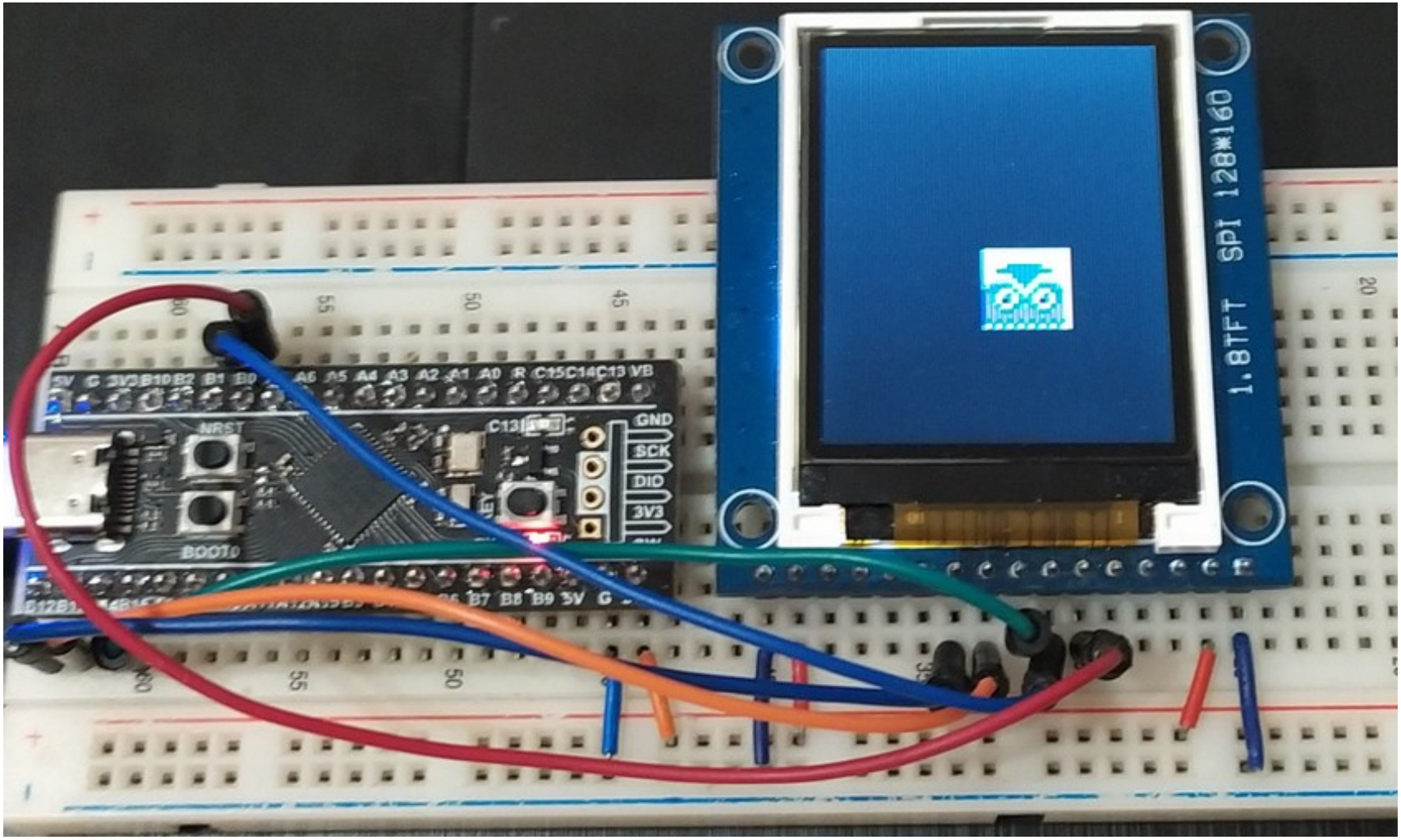
Itt egy alternatív megoldást mutatunk a bitkép fájl beolvasására, ehhez nem kell külön könyvtár



```
import board
import busio
import displayio
import time
from adafruit_st7735r import ST7735R
displayio.release_displays()      # Release resources currently in use
spi = busio.SPI(board.B13, MOSI=board.B15)
display_bus=displayio.FourWire(spi,command=board.B1,chip_select=board.B12, reset=board.B0)
display = ST7735R(display_bus, width=128, height=160, rotation=0, bgr=True)
splash = displayio.Group()        # Make the display context
display.show(splash)

sprite_sheet = displayio.OnDiskBitmap("/cp_sprite_sheet.bmp")     # Load bitmap file
# Create a sprite (tilegrid)
sprite = displayio.TileGrid(sprite_sheet, pixel_shader=sprite_sheet.pixel_shader,
                            width=1, height=1, tile_width=16, tile_height=16)
group = displayio.Group(scale=2) # Create a Group to hold and scale the sprite
group.append(sprite)             # Add the sprite to the Group
splash.append(group)
group.x = 60                     # Set sprite location
group.y = 120
source_index = 0                 # Loop through each sprite in the sprite sheet
while True:
    sprite[0] = source_index%6   # Váltogatjuk a sprite képét
    source_index += 1
    group.x = 20 + (source_index % 6)*10  # fölülírjuk a pozíciót (mozgatás)
    group.y = 20 + (source_index % 6)*16
    time.sleep(2)
```

ST7735R_sprite.py futási eredmény

- A program futási eredménye



Bitmap képek betöltése és megjelenítése

- Az előzőekben már megismerkedtünk a képfájlok betöltésével, amelyre két módszert is láttunk
  - Az **Adafruit_ImageLoad** programkönyvtár, melynek **load()** tagfüggvénye egy ***bitmap*** és egy ***palette*** objektumot szolgáltat számunkra, például:

```
bitmap, palette = adafruit_imageload.load("mypic.bmp",
                                          bitmap=displayio.Bitmap,
                                          palette=displayio.Palette)
```

  - A beépített **displayio** modul **OnDiskBitmap()** osztálya, amely közvetlenül a fájlból tölti be az adatokat (kisebb memóriaigény, de lassabb hozzáférés). Ennél a paletta kinyerése az objektum **pixel_shader** metódusával történik

```
bitmap = displayio.OnDiskBitmap("mypic.bmp")
tilegrid = displayio.TileGrid(bitmap, pixel_shader=bitmap.pixel_shader)
```

- A második módszer felhasználásával töltsünk be egy megadott nevű képet és jelenítsük meg a kijelzőn!
- **Megjegyzés:** Gondoskodjunk róla, hogy a képfájl 128x160 pixel méretű, .BMP típusú (indexelt, vagy 24 bites színmélységű legyen)

ST7735R_bitmap.py

```
import board
import busio
import terminalio
import displayio

from adafruit_st7735r import ST7735R

displayio.release_displays()

# create the spi device and pins we will need
spi = busio.SPI(board.B13, MOSI=board.B15)
display_bus = displayio.FourWire(spi, command=board.B1,
                 chip_select=board.B12, reset=board.B0)
display = ST7735R(display_bus, width=128, height=160,
                                 rotation=0, bgr=True)

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

# Setup the file as the bitmap data source
bitmap = displayio.OnDiskBitmap("/demep.bmp")

# Create a TileGrid to hold the bitmap
sprite = displayio.TileGrid(bitmap, pixel_shader=bitmap.pixel_shader)
splash.append(sprite)       # Add the TileGrid to the Group

while True:                 # Loop forever so you can enjoy your image
    pass
```

ST7735R_slideshow.py

- Készítsünk diabemutatót (slideshow) egyszerűen:
  - Telepítsük az adafruit_slideshow könyvtárat!
  - Helyezzük el a képeket (.bmp) az /**images** nevű könyvtárba
  - Futtassuk az alábbi programot:

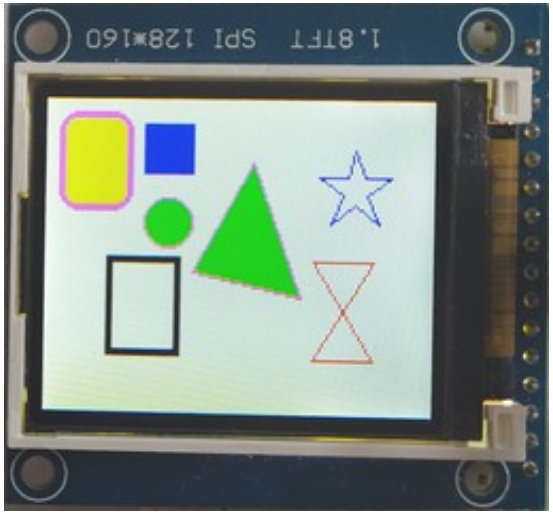
```
import board
import busio
import displayio
import time
from adafruit_st7735r import ST7735R
from adafruit_slideshow import PlayBackOrder, SlideShow
displayio.release_displays()
spi = busio.SPI(board.B13, MOSI=board.B15)
display_bus = displayio.FourWire(spi, command=board.B1,
                  chip_select=board.B12, reset=board.B0)
display = ST7735R(display_bus, width=128, height=160, rotation=0, bgr=True)

# Create the slideshow object that plays through once alphabetically.
slideshow = SlideShow(display,
                      folder="/images",
                      loop=True,
                      order=PlayBackOrder.ALPHABETICAL,
                      Dwell=5)                        # tartózkodási idő

while slideshow.update():
    pass
```

Alakzatok rajzolása – Display Shapes

- Alakzatok rajzolásához az Adafruit_CircuitPython_Display_Shapes könyvtárat használhatjuk (lásd displayio UI quickstart)
  referencia kézikönyv: Adafruit Display_Shapes Library leírása
- **Pont** – egy *bitmap* objektum pontjait közvetlenül címezhetjük, például `bitmap[10,20] = color`
- **Line**(*x0,y0,x1,y1,color*) – végpontokkal adott szakasz rajzolása
- **Triangle**(*x0,y0,x1,y1,x2,y2,fill,outline)* – kitöltött vagy üres háromszög rajzolása (a *None* értékkel definált szín átlátszó)
- **Rect**(*x0,y0,width,height,fill,outline,stroke*) – (kitöltött) téglalap rajzolása
- **RoundRect**(*x0,y0,width,height,r,fill,outline,stroke*) – lekerekített sarkú (kitöltött) téglalap rajzolása (*r* – a sugár)
- **Circle**(*x0,y0,r,fill,outline,stroke*) – (kitöltött) kör rajzolása
- **Polygon**(*points,outline*) – poligon rajzolása (*points: (x,y)* tupletek listája)
- **Sparkline**(*width,height,max_items, y_min, y_max,x0,y0,color*) – egyszerű vonaldiagram rajzolása (**add_value**(*adat*) – adat hozzáfűzése)

ST7735R_shapes.py 3/1.

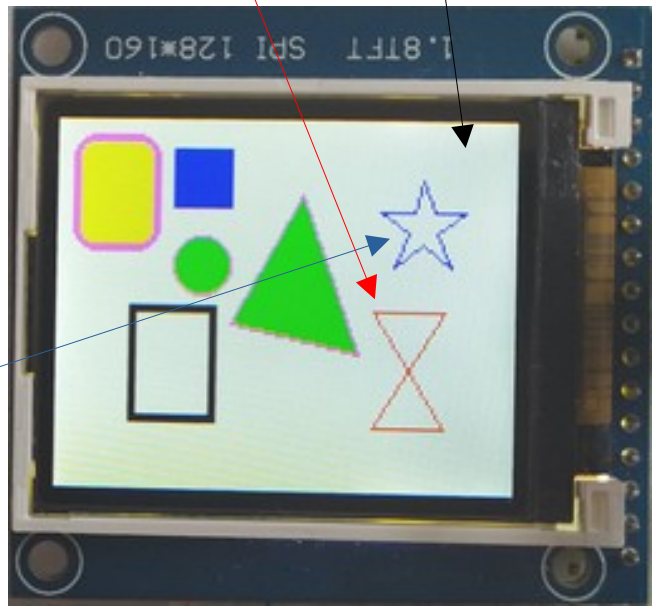
```
import board
import busio
import displayio
from adafruit_st7735r import ST7735R
from adafruit_display_shapes.rect import Rect
from adafruit_display_shapes.circle import Circle
from adafruit_display_shapes.roundrect import RoundRect
from adafruit_display_shapes.triangle import Triangle
from adafruit_display_shapes.line import Line
from adafruit_display_shapes.polygon import Polygon

# Release any resources currently in use for the displays
displayio.release_displays()

# create the spi device and pins we will need
spi = busio.SPI(board.B13, MOSI=board.B15)
display_bus = displayio.FourWire(spi, command=board.B1,
                  chip_select=board.B12, reset=board.B0)
display = ST7735R(display_bus, width=160, height=128, rotation=90, bgr=True)

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



Fekvő formátum!

ST7735R_shapes.py 3/2.

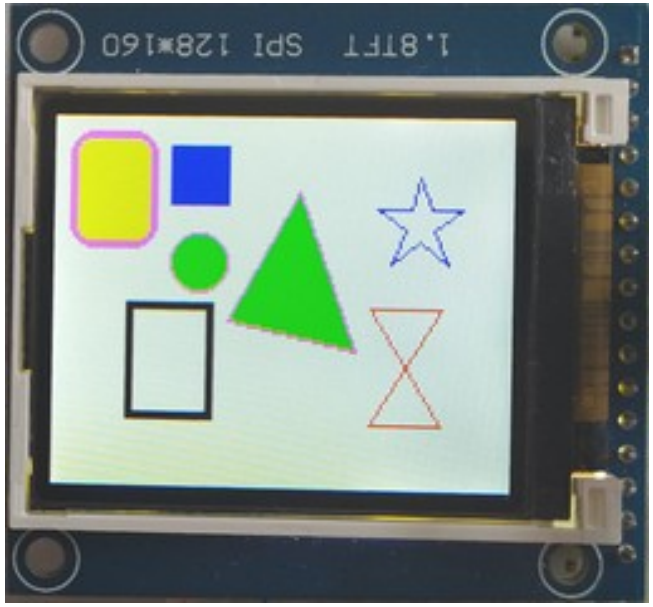
```
# Make a background color fill
color_bitmap = displayio.Bitmap(160, 128, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0xFFFFFF
bg_sprite = displayio.TileGrid(color_bitmap, x=0, y=0, pixel_shader=color_palette)
splash.append(bg_sprite)

splash.append(Line(110,  65, 135, 105, 0xFF0000))
splash.append(Line(135, 105, 110, 105, 0xFF0000))
splash.append(Line(110, 105, 135,  65, 0xFF0000))
splash.append(Line(135,  65, 110,  65, 0xFF0000))

# Draw a blue star
polygon = Polygon(
    [
        (127, 20),
        (131, 31),
        (142, 31),
        (132, 38),
        (137, 50),
        (127, 42),
        (117, 50),
        (122, 38),
        (112, 31),
        (124, 31),
    ], outline=0x0000FF,
)
splash.append(polygon)
```



ST7735R_shapes.py 3/3.

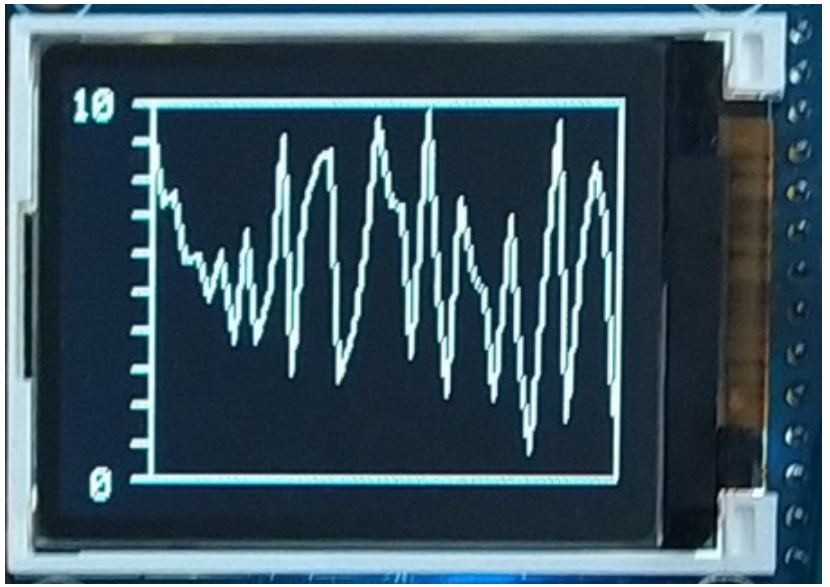
- A kijelző bal oldalán látható alakzatok kirajzolása itt történik
- A fekete szegélyű téglalap kitöltési színe: átlátszó (alapértelmezés)

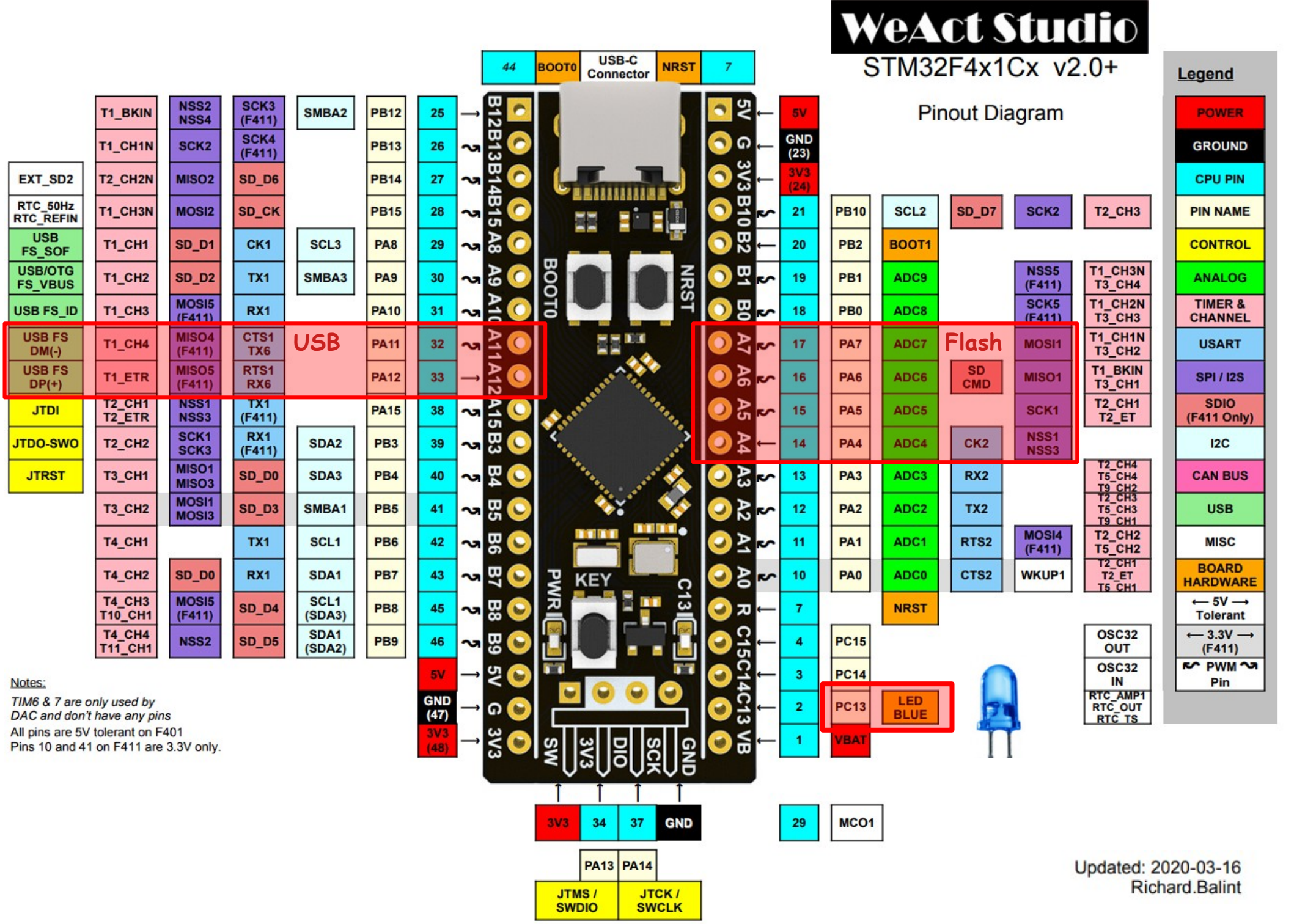
```
triangle = Triangle(85,25,60,70,105,80,fill=0x00FF00,outline=0xFF00FF)
splash.append(triangle)

circle = Circle(50,50,10,fill=0x00FF00,outline=0xFF00FF)
splash.append(circle)

roundrect = RoundRect(5,5,31,41,8,fill=0xFFFF00,outline=0xFF00FF,stroke=3)
splash.append(roundrect)

rect = Rect(40, 10, 21, 21, fill=0x0000FF)
splash.append(rect)

rect2 = Rect(25,64,31,41,outline=0x0,stroke=3)
splash.append(rect2)

while True:
    pass
```



ST7735R_sparkline.py

- A **Sparkline** alakzat segítségével automatikusan frissülő diagramot készíthetünk (újabb adatokat az **add_value()** metódussal vihetünk be)
- A megadott maximális adatszám elérése után a régebbi adatok „kicsorognak” a diagram baloldalán
- A könyvtári mintapéldákban véletlen számokat generálunk (valóságos esetben itt egy-egy ADC mérés, vagy kiolvasott szenzor adat kerülthetne be az adatsor végére)
- Az Y tengely skálázását és feliratozását nem tartalmazza a **Sparkline** osztály, azt külön, a programozónak kell megoldania
- Az adatok beléptetésének ütemét egy ***time.sleep()*** késleltetés szabja meg
- A programlistát itt nem ismertetjük, lényegében csak az egyik könyvtári mintapéldát adaptáltuk (display_shapes_sparkline_ticks.py) az ST7735 TFT képernyőhöz és a 160x128 felbontáshoz




WeAct Studio
STM32F4x1Cx v2.0+
Pinout Diagram
Legend
POWER
GROUND
CPU PIN
PIN NAME
CONTROL
ANALOG
TIMER & CHANNEL
USART
SPI / I2S
SDIO (F411 Only)
I2C
CAN BUS
USB
MISC
BOARD HARDWARE
5V Tolerant
3.3V (F411)
PWM Pin
USB-C Connector
BOOT0
NRST
KEY
PWR
C13
LED BLUE
USB
Flash
SD CMD
SW
3V3
DIO
SCK
GND
JTMS / SWDIO
JTCK / SWCLK
Notes:
TIM6 & 7 are only used by DAC and don't have any pins
All pins are 5V tolerant on F401
Pins 10 and 41 on F411 are 3.3V only.
Updated: 2020-03-16
Richard.Balint