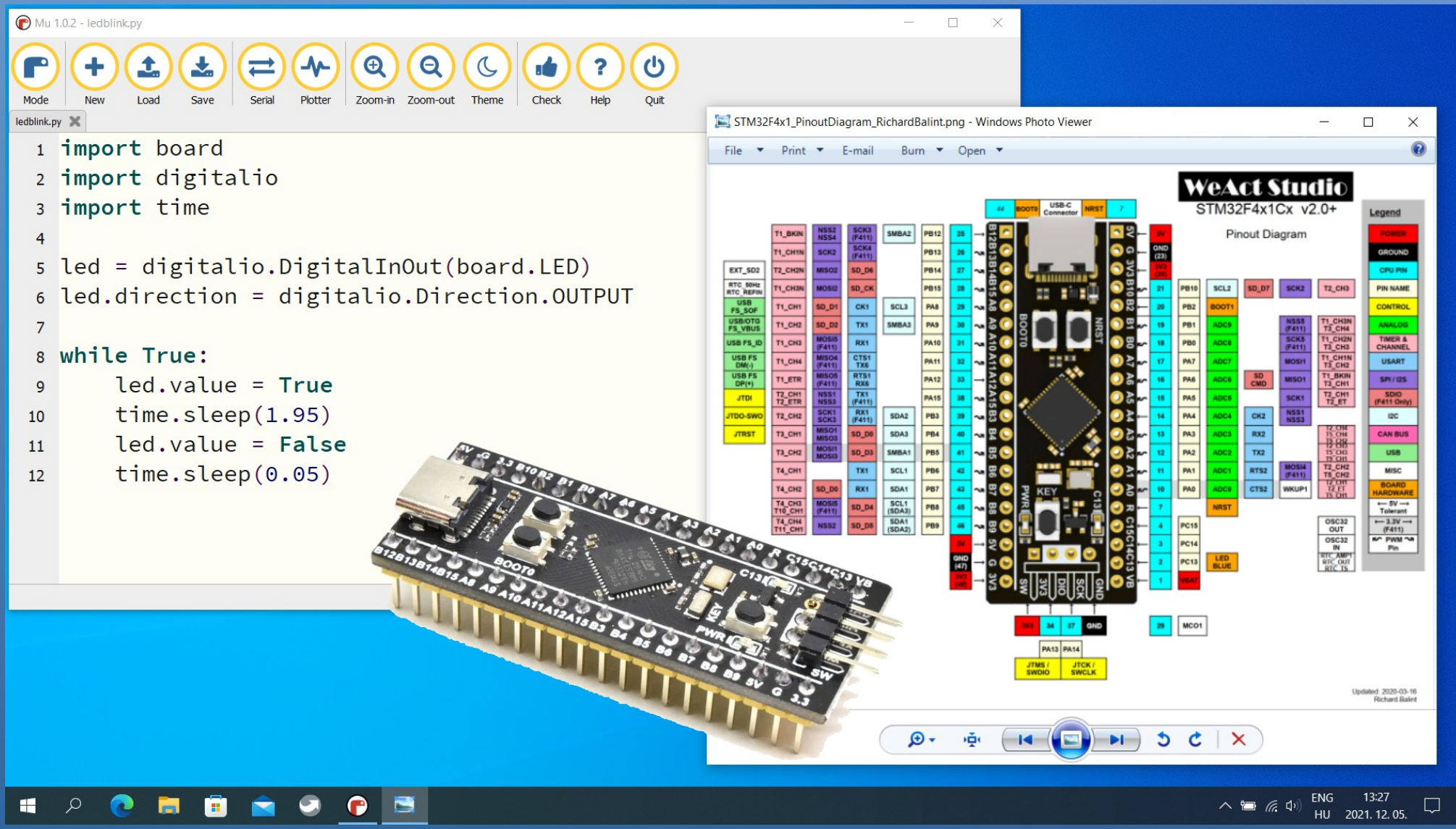
CircuitPython tanfolyam

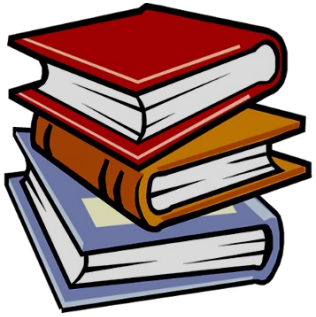


4. Interaktív fejlesztés Jupyter Notebookkal

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
- **Anaconda data science toolkit: www.anaconda.com**
- **Jupyter Notebook: jupyter.org**

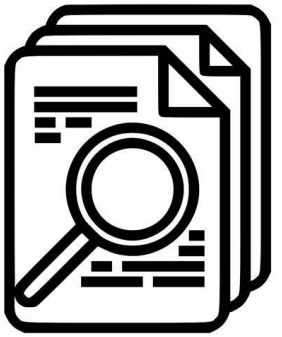


CircuitPython:

- Adafruit: https://circuitpython.org/downloads
- Learn Adafruit: CircuitPython Essentials
- Brent Rubell: **CircuitPython with Jupyter Notebooks**
- 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



Mi az a Jupyter Notebook?

- A **Jupyter Notebook** egy nyílt forráskódú webalkalmazás, amely lehetővé teszi az „élő” kódot, egyenleteket, adatvizualizációt és narratív szöveget tartalmazó dokumentumok létrehozását és megosztását
- A **Jupyter projekt** több, mint 40 programozási nyelvet támogat, köztük a **Python,** C++, R, Julia és Scala nyelveket
- Bennünket most természetesen csak a **Python** nyelv érdekel, és bizonyos okokból a webes használat helyett a **helyi telepítést** választjuk

A Jupyter Notebook telepítése

- Ha már telepítve van a gépünkön a **Python 3** csomag, akkor használjuk a **pip** parancsot egy parancsablakban:

```
pip install notebook
```

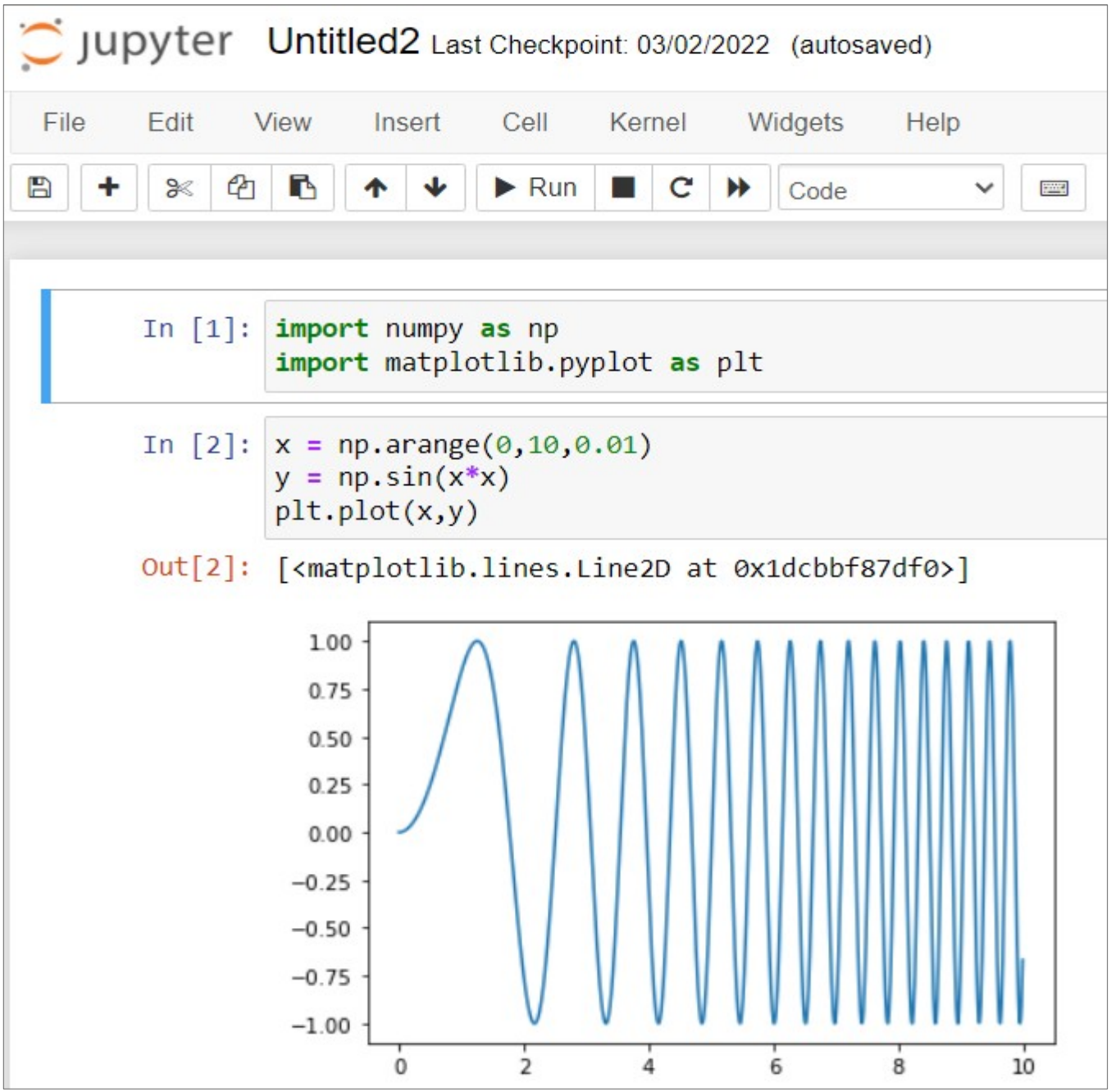
- A telepített **Jupyter Notebook** futtatása az alábbi paranccsal:

```
jupyter notebook
```

- A másik lehetőség a **Python** és a **Jupyter** együttes telepítése az **Anaconda** csomagban (az ún. ***Individual Edition*** ingyenesen használható)
  - Anaconda Distribution Starter Guide
  - Installation (telepítési útmutató)
  - Getting started with Anaconda
- Az **Anaconda** csomag telepítése után a **Jupyter Lab** vagy a **Jupyter Notebook** indítása a **Start** menüből történhet

A Jupiter Notebook használata

- A **Jupyter Notebook** dokumentumok cellákból állnak, melyekben vagy kód vagy *markdown* formázott szöveg áll
- A programkódot tartalmazó cellákat a **Run** gombbal lehet végrehajtatni
- A végrehajtás eredménye nemcsak szöveg, hanem a mellékelt példa szerint ábra is lehet



Numerikus integrálás: a trapéz-szabály

A numerikus integrálás egy közelítő integrálási módszert jelent, melynek alkalmazásával a határozott integrálok közelítő értékét számoljuk ki. Az egyik legismertebb módszer a trapéz-szabály alkalmazása, ahol a két szomszédos pontot összekötő egyenes alatti trapéz területével közelítjük az adott szakaszra az integrált.

$$\int_a^b f(x)\,dx \approx \frac{1}{2}\sum_{k=1}^{N}(x_k - x_{k-1})\,(f(x_k) + f(x_{k-1}))\,.$$

Az alábbi példában először definiálunk egy egyszerű függvényt, majd mintavételezzük azt x = 0 - 10 között, 200 pontban.

In [1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

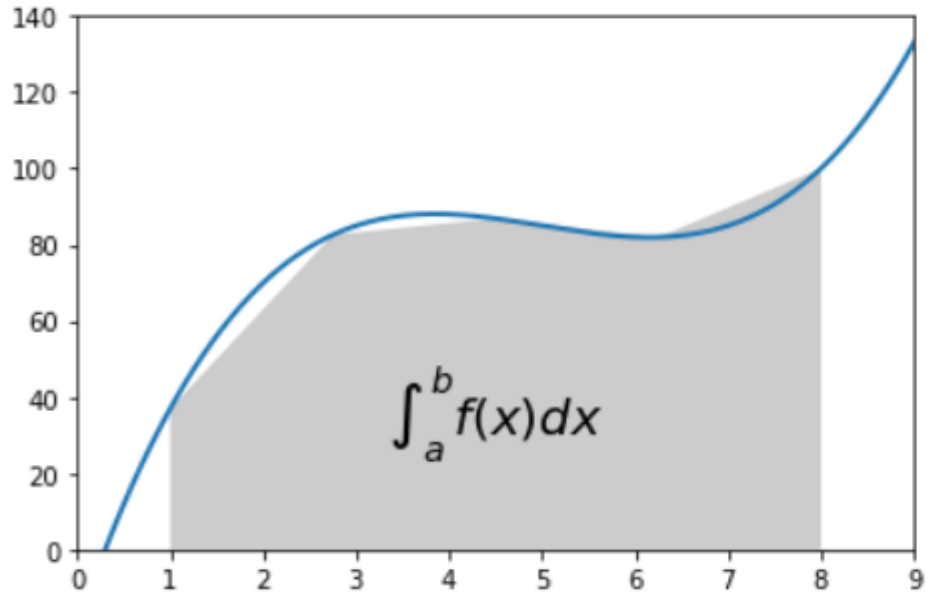
In [2]:
```
def f(x):
    return (x-3)*(x-5)*(x-7)+85

x = np.linspace(0, 10, 200)
y = f(x)
```

Adjuk meg az integrálás határait és osszuk fel ezt a tartományt néhány szeletre!

In [3]:
```
a, b = 1, 8 # az integrálás határai
N = 5       # a pontok száma
xint = np.linspace(a, b, N)
yint = f(xint)
```

Ábrázoljuk a függvényt és a görbe alatti terület trapéz közelítését!

In [4]:
```
plt.plot(x, y, lw=2)
plt.axis([0, 9, 0, 140])
plt.fill_between(xint, 0, yint, facecolor='gray', alpha=0.4)
plt.text(0.5 * (a + b), 30,r"$\int_a^b f(x)dx$", horizontalalignment='center', fontsize=20);
```



számítsuk ki az integrál értékét nagy pontosságú közelítéssel és trapéz közelítéssel is!

In [5]:
```
from __future__ import print_function
from scipy.integrate import quad
integral, error = quad(f, a, b)
integral_trapezoid = sum( (xint[1:] - xint[:-1]) * (yint[1:] + yint[:-1]) ) / 2
print("The integral is:", integral, "+/-", error)
print("The trapezoid approximation with", len(xint), "points is:", integral_trapezoid)
```

```
The integral is: 565.2499999999999 +/- 6.275535646693696e-12
The trapezoid approximation with 5 points is: 559.890625
```

Python dekorátorok

- **Nagy Donát** „*Python dekorátorok*” c. előadása az ELTE Bolyai kollégium Informatikai Szakszemináriumán hangzott el 2018. március 7-én (Letölthető előadásvázlat: HTML, Jupyter Notebook)
- Egy függvény definíciója (a **def** utasítás) két dolgot csinál:
  - létrehoz egy függvény objektumot
  - azt eltárolja olyan néven, amit megadtunk
- **A dekorátorok** lehetővé teszik, hogy valamit „beszúrjunk” eközé a két lépés közé: létrejön a függvény objektum, meghívódik a dekorátor és megkapja paraméterként az éppen létrejött függvény objektumot, majd a dekorátor visszatérési értéke eltárolódik olyan néven, amit a függvény definíciójánál megadtunk
- A dekorátorokat kukac karakterrel kell bevezetni:

```
@callable_used_as_decorator
def new_function(arguments):
    #... function body
```

- Ez nagyjából annak felel meg, mintha ezt írtuk volna:

```
def _temporary_function_object(arguments):
    #... function body
new_function =  callable_used_as_decorator(_temporary_function_object)
```

Python dekorátorok

- **Property-k használata:** Egy **property** közönséges adattagnak látszik, de valójában függvényeket hív meg, amikor adatot írnak bele/adatot olvasnak ki belőle. Egy **property** legegyszerűbben **dekorátorok** segítségével hozható létre

In [11]:
```
import math
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    @property
    def angle(self):
        return math.atan2(self.y, self.x) # Az eredeti dokumentumban itt fel volt cserélve x és y!!!
    @property
    def r(self):
        return math.sqrt(self.x**2 + self.y**2)

p = Point(3,4)
print(p.angle, p.r)
```

```
0.6435011087932844 5.0
```

Ezek most csak olvasható adattagként viselkednek:

In [12]:
```
p.r = 10
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_5300/1164171893.py in <module>
----> 1 p.r = 10

AttributeError: can't set attribute
```

Python dekorátorok

... de definiálhatóak hozzájuk setterek is:

In [13]:
```
import math
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    @property
    def angle(self):
        return math.atan2(self.y, self.x) # Az eredetiben fel volt cserélve x és y!!!
    @angle.setter
    def angle(self, value):
        r = self.r
        self.x = math.cos(value)*r
        self.y = math.sin(value)*r
    @property
    def r(self):
        return math.sqrt(self.x**2 + self.y**2)
    @r.setter
    def r(self, value):
        angle = self.angle
        self.x = math.cos(angle)*value
        self.y = math.sin(angle)*value

p = Point(3,4)
p.r = 10
print(p.x, p.y)
```

```
6.000000000000001 7.999999999999999
```

CircuitPython Jupyter Notebookkal

- A **Jupyter Notebook** segítségével interaktív jegyzeteket készíthetünk, ha pedig telepítjük hozzá a **CircuitPyton** „kernelt”, akkor a jegyzeteinkben szereplő **CircuitPython** kód a mikrovezérlő kártyán fog futni
- **CircuitPython Kernel telepítése Windows alatt:**
  - Töltsük le és valahová bontsuk ki a CircuitPython Kernel csomagot
  - Indítsunk el egy **Anaconda promptot,** lépjünk be a kibontott könyvtárba (**circuitpython_jupyter_kernel_master)** és adjuk ki a parancsot:

    ```
    python setup.py install
    ```

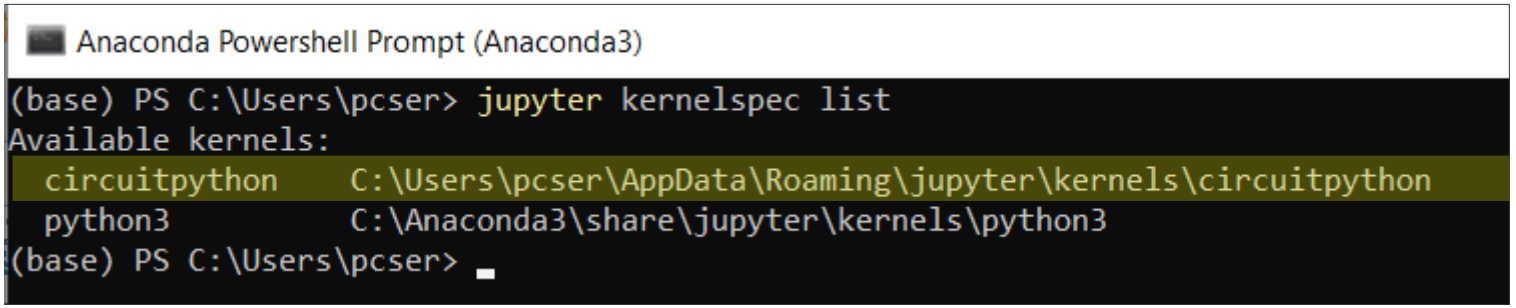
  - A `setup.py` lefutása után adjuk ki a következő parancsot is:

    ```
    python -m circuitpython_kernel.install
    ```

  - A telepítés ezzel kész, de a következő paranccsal leellenőrizhetjük:

    ```
    jupyter kernelspec list
    ```

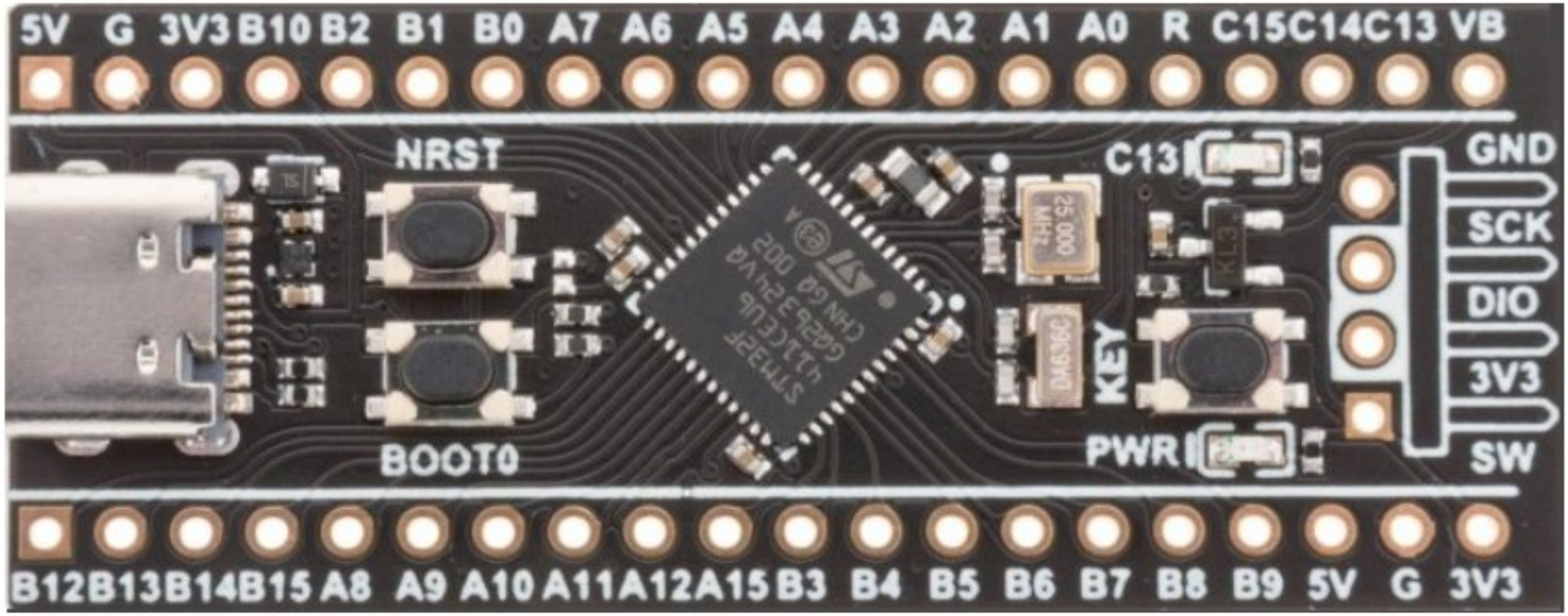
- Ha szerepel a listán a **circuitpython** kernel, akkor minden rendben van



blackpill_ledblink.ipynb 3/1.

LED villogtatása

A flash memóriával bővített STM32F411CE blackpill kártya beépített LED-jét fogjuk villogtatni, közvetlenül ebből a Jupyter Notebook jegyzetből!



Először ellenőrizzük, hogy a Jupyter Notebook tud-e kommunikálni a CircuitPython kártyával:

In [1]:
```
import os
print("Running CircuitPython release", os.uname().release, "on board", os.uname().machine)
```

```
Running CircuitPython release 7.0.0 on board stm32f411ce-blackpill-with-flash with STM32F411CE
```

blackpill_ledblink.ipynb 3/2.

Először importáljuk a `board` és `digitalio` modulokat:

In [2]:
```
import digitalio
import board
```

Példányosítsunk egy `digitalio.DigitalInOut` objektumot a beépített LED kezelésére, amelynek katódja a blackpill kártya C13 kivezetéséhez van kötve

In [3]:
```
led = digitalio.DigitalInOut(board.C13)
```

Állítsuk be az adatáramlás irányát kimenetre (`digitialio.Direction.OUTPUT`)

In [4]:
```
led.direction = digitalio.Direction.OUTPUT
```

Végül kapcsoljuk fel a LED-et a katód lehúzásával (a kimenet `False` állapotba állításával)!

In [5]:
```
led.value = False  # LED on: Negatív logika, a katódvezérlés miatt...
```

Nem akarjuk, hogy világítson a LED? Állítsuk át a kimenetet `False` helyett `True` állapotba!

In [6]:
```
led.value = True   # LED off: Negatív logika, a katódvezérlés miatt...
```

blackpill_ledblink.ipynb 3/3.

A folvamatos villogtatáshoz szükségünk lesz egy időzítésre, ehhez azonban importálni kell a `time` modult, majd szerveznünk kell egy programciklust.

Az alábbi példában tízszer vált a LED állapotot, azaz ötször gyullad fel és alszik ki. Az állapotváltások közötti idő 0.25 s, azaz negyed másodperc.

In [7]:
```
import time

for i in range(10):
    led.value = not led.value
    time.sleep(0.25)
```

- A beépített LED gyári definiálását (board.LED) is használhatjuk:

```
import board
import digitalio
import time

led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

while True:
    led.value = True      # LED off
    time.sleep(1.95)
    led.value = False     # LED on
    time.sleep(0.05)
```

Ennek futtatásához a Kernelt újra kell indítani, hogy a C13 kimenetet felszabadítsuk az újabb digitalio.DigitalInOut példányosításhoz

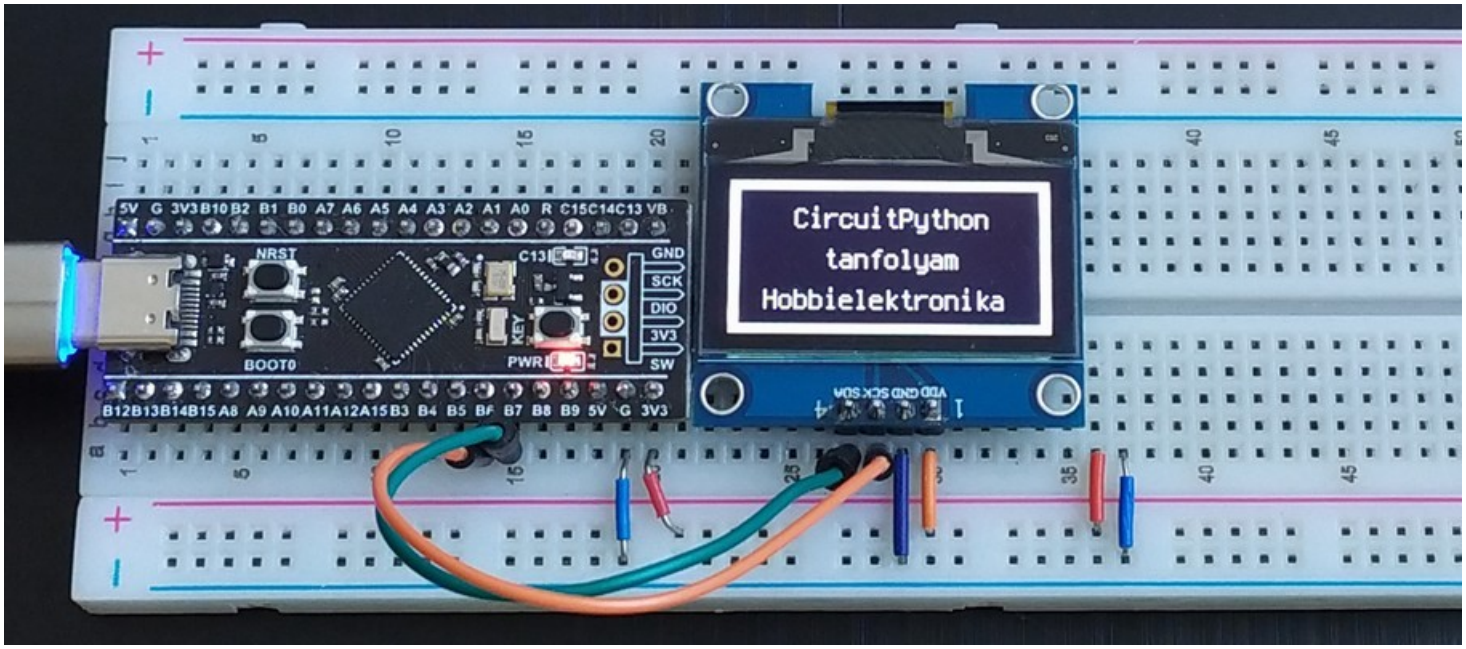
SH1106_displayio_test.py.ipynb

- Ez az **`SH1106_displayio_test.py`** program a 2022. február 10-i előadásból:

Az alábbi mintapéldában a **displayio könyvtárat** fogjuk használni, amely része a **CircuitPython firmware** csomagnak, tehát eleve "kéznél van".

A kijelző kezeléséhez azonban, a vezérlő típusától függően, telepítenünk kell az **`adafruit_displayio_ssd1306.mpy`** vagy az **`adafruit_displayio_sh1106.mpy`** könyvtárat (az **Adafruit CircuitPython Bundle** tartalmazza ezeket). **Megjegyzés:** ügyeljünk az elnevezésre, ne keverjük össze ezek nevét a korábbi programoknál használt `displayio` nélküliek meghajtókkal!

Telepítenünk kell továbbá az **`adafruit_display_text`** programkönyvtárat is



SH1106_displayio_test.py.ipynb

A program elején importáljuk a szükséges könyvtári modulokat!

```
import board
import displayio
import terminalio
from adafruit_display_text import label
import adafruit_displayio_sh1106
# SSD1306 kijelzőhöz az `adafruit_displayio_ssd1306` könyvtárat kell importálni
```

Ha már van megnyitott kijelző eszköz, akkor engedjük el ...

```
displayio.release_displays()
```

Nyissuk meg az I2C csatornát, a display_bus-t és a kijelzőt!

A `rotation=180` beállítás azt eredményezi, hogy akkor látjuk egyenes állásban a képet, ha a kijelző tüskéi alul helyezkednek el ("6 óránál")

```
i2c = board.I2C()
display_bus = displayio.I2CDisplay(i2c, device_address=0x3C)  # 0x3C a kijelző I2C címe
display = adafruit_displayio_sh1106.SH1106(display_bus, width=132, height=64, rotation=180)
# SH1106 esetén 132x64-ra definiáljuk a kijelzőt, de csak 128x64 képpont látható a kijelzőn,
# emiatt majd egy 2 pixeles x eltoltást kell alkalmaznunk minden pozicionálásnál

# SSD1306 kijelző esetén így nézne ki a display definiálása:
# display = adafruit_displayio_ssd1306.SSD1306(display_bus, width=128, height=64, rotation=180)
```

SH1106_displayio_test.py.ipynb

A kijezőre szánt tartalmat a memóriában gyűjtjük össze, a képelemeket leíró objektumokat a `splash` nevű általános gyűjtő (**Group** típusú objektum) fogje egybe. **Várható eredmény:** elsötétedik a teljes kijező

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

Az első rajzobjektum egy kitöltött fehér téglalap lesz, amit egy **Bitmap** és egy **Palette** objektumból rakunk össze és a `bg_sprite` nevű **TileGrid** objektum tárolja

Megjegyzés: A **TileGrid** objektum pozicionálásánál 2 pixelnyi *x* irányú eltolást alkalmazunk, ha **SH1106** a kijelzővezérlő, (**SSD1306** esetén természetesen X=0 kellene)

```
# Draw a big white rectangle as backround
color_bitmap = displayio.Bitmap(128, 64, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0xFFFFFF  # White
bg_sprite = displayio.TileGrid(color_bitmap, pixel_shader=color_palette, x=2, y=0)
```

Miután létrehoztuk a `bg_sprite` rajzelemet, adjuk hozzá a gyűjtőhöz!

Várható eredmény: Kifehéredik a teljes kijező

```
splash.append(bg_sprite)
```

SH1106_displayio_test.py.ipynb

A második rajzobjektum egy kisebb kitöltött fekete téglalap lesz (ezt is `Bitmap` és egy `Palette` objektumból rakjuk össze) és az `inner_sprite` nevű `TileGrid` objektum tárolja. Definiálás után ezt is hozzáadjuk a gyűjtőhöz.

Megjegyzés: A képelem pozicionálásánál itt is figyelembe vettük az **SH1106** kijelző esetén alkalmazandó 2 pixelnyi eltolást! (**SSD1306** kijelző esetén x=4 kellene)

Várható eredmény: A fehér kijező közepén megjelenik egy fekete téglalap (így végeredményben egy 4 pixel széles fehér keretet kaptunk)

```
# Draw a smaller inner rectangle
inner_bitmap = displayio.Bitmap(120, 56, 1)
inner_palette = displayio.Palette(1)
inner_palette[0] = 0x000000  # Black
inner_sprite = displayio.TileGrid(inner_bitmap, pixel_shader=inner_palette, x=6, y=4)
splash.append(inner_sprite)
display.show(splash)
```

A szöveg kiírásához a `terminalio.FONT` -ot használjuk és a `text_area` nevű, **Label** típusú objektum tárolja a létrehozott képelemet. Amint látjuk, a kocsi-vissza/soremelés karakterekkel többsoros kiírást is készíthetünk. A képelem pozicionálásánál itt is vegyük figyelembe az **SH1106** kijelző esetén alkalmazandó 2 pixelnyi eltolást!

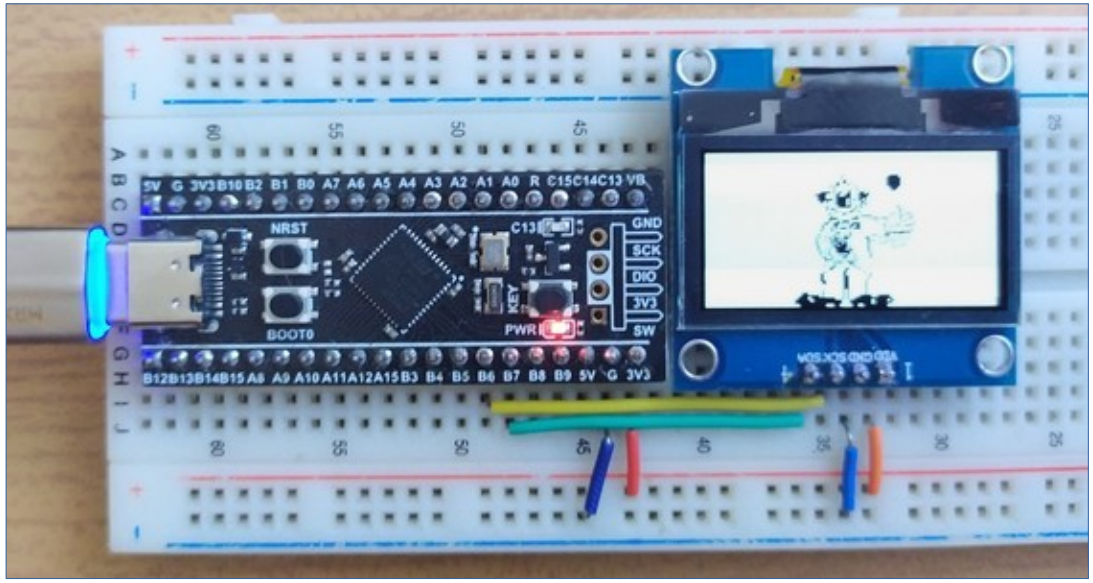
Várható eredmény: A `text_area` objektum gyűjtőhöz fűzése után megjelenik a szöveg is.

```
# Draw a label
text = "   CircuitPython\r\n     tanfolyam\r\n Hobbielektronika"
text_area = label.Label(terminalio.FONT, text=text, color=0xFFFFFF, x=10, y=15)
splash.append(text_area)
```

SH1106_clown_animation.py

- Animációt készítünk egy **SH1106** I2C OLED kijelzőn
  A szükséges ismeretek megtalálhatók a **2022. február 10-i** és a **2022. március 3-i** előadásokban. (lásd itt: Foglalkozások 2021/2022-ben)

- **Hardver:**
  - **STM32F411CE** kártya, flash memóriával
  - **SH1106** I2C OLED kijelző (128x64 felbontás)



- **Szoftver:**
  - CircuitPython 7.0.0 firmware
  - Adafruit_displayio_SH1106 meghajtó
  - Adafruit_imageload programkönyvtár
  - small_clonws.zip (Nuts and Volts magazin)

A bitkép fázisképek egyesítése

- Első lépésként a **small_clowns.zip** fájlból kibontott fázisképeket összedolgozzuk egyetlen képpé, például így:

- A fázisképek mérete 56 x 64 képpont, így a fenti elrendezésben a teljes kép 224 x 128 képpont
- Monokróm képként, BMP formátumban mentsük el, **clowns.bmp** néven

SH1106_clown_animation.py

A kijelző kelezéséhez szükségünk lesz az `Adafruit_displayio_sh1106` meghajtóra, a kép beolvasásához pedig az `Adafruit_imageload` programkönyvtárra.

```
import board
import displayio
from adafruit_displayio_sh1106 import SH1106
import adafruit_imageload
import time
```

A képernyő inicializálása a 2022. február 10-i előadás szerint történik. Az **SH1106** vezérlő 132 x 64 felbontást kezel, de tényleges kijelzés csak a középre igazított 128 x 64-es mezőben van, ezért a 2 képpontnyi eltolásról majd nekünk kell gondoskodni!

```
displayio.release_displays() # Elengedjük a kijelzőt...

# Inintialize display
i2c = board.I2C()
display_bus = displayio.I2CDisplay(i2c, device_address=0x3C)
display=SH1106(display_bus, width=132, height=64, rotation=180)
```

SH1106_clown_animation.py

A megjelenítés gyűjtőeleme a legfelső szintű **Group** objektum

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

Elkészítjük a fehér hátteret, ami egy 128 x 64 pixeles **TileGrid** (csempe) objektum, x = 2 eltolással)

```
# Make white backgound
color_bitmap = displayio.Bitmap(128, 64, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0xFFFFFF # White
bg_sprite = displayio.TileGrid(color_bitmap, pixel_shader=color_palette, x=2, y=0)
splash.append(bg_sprite)
```

Betöltjük a 8 db fázisképet tartalmazó `clons.bmp` képfájlt és **sprite** néven készítünk belőle egy 56x64 pixeles csempét, amely alapértelmezetten a betöltött kép bal felső sarkából kivágott szeletet jelöli ki. A `sprite[0] = n` értékadással (ahol n = 0..7) meg tudjuk változtatni, hogy a sprite nevű csempe melyik fázisképet hasítsa ki a betöltött képfájlból (lásd: a 2022. március 3-i előadásban)

```
# Load the sprite sheet (bitmap)
sprite_sheet,palette=adafruit_imageload.load("/clowns.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=56,tile_height=64,x=40,y=0)
splash.append(sprite)
```

SH1106_clown_animation.py

Végül a fázisképek váltogatásával futtatjuk az animációt

```
source_index = 0

# Run the animation by changing the source tile
while True:
    sprite[0] = source_index # Váltogatjuk a sprite képét
    source_index = (source_index + 1) % 8
    time.sleep(0.1)
```

Alakzatok rajzolása – Display Shapes

- Alakzatok rajzolásához az Adafruit_CircuitPython_Display_Shapes könyvtárat használhatjuk (lásd displayio UI quickstart)
  referencia kézikönyv: Adafruit Display_Shapes Library leírása

A legfontosabb alakzatok:

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)

Alakzatok rajzolása – Display Shapes

Az alakzatokat célszerű egyesével importálni, hogy később kevesebbet kelljen írni:

```
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
```

Az SPI kijelző inicializálása

```
# 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)
```

Alakzatok rajzolása – Display Shapes

A kijelzendő tartalmat összefogó csoport (Group) inicializálása

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

A fehér háttér (bg_sprite) definiálása és megjelenítése

```
# 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)
```

"Homokóra" rajzolása piros színű, egyenes szakaszokkal

```
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))
```

Alakzatok rajzolása – Display Shapes

Poligon (csillag) rajzolása kék színű vonalakkal

```
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)
```

Kitöltött háromszög rajzolása (zöld kitöltés, magenta körvonal)

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

Alakzatok rajzolása – Display Shapes

Kitöltött kör rajzolása (zöld kitöltés, magenta körvonal)

```
circle = Circle(50,50,15,fill=0x00FF00,outline=0xFF00FF)
splash.append(circle)
```

Lekerekített sarkú téglalap rajzolása (lekerekítés sugara = 8, sárga kitöltés, magenta színű vastag körvonal)

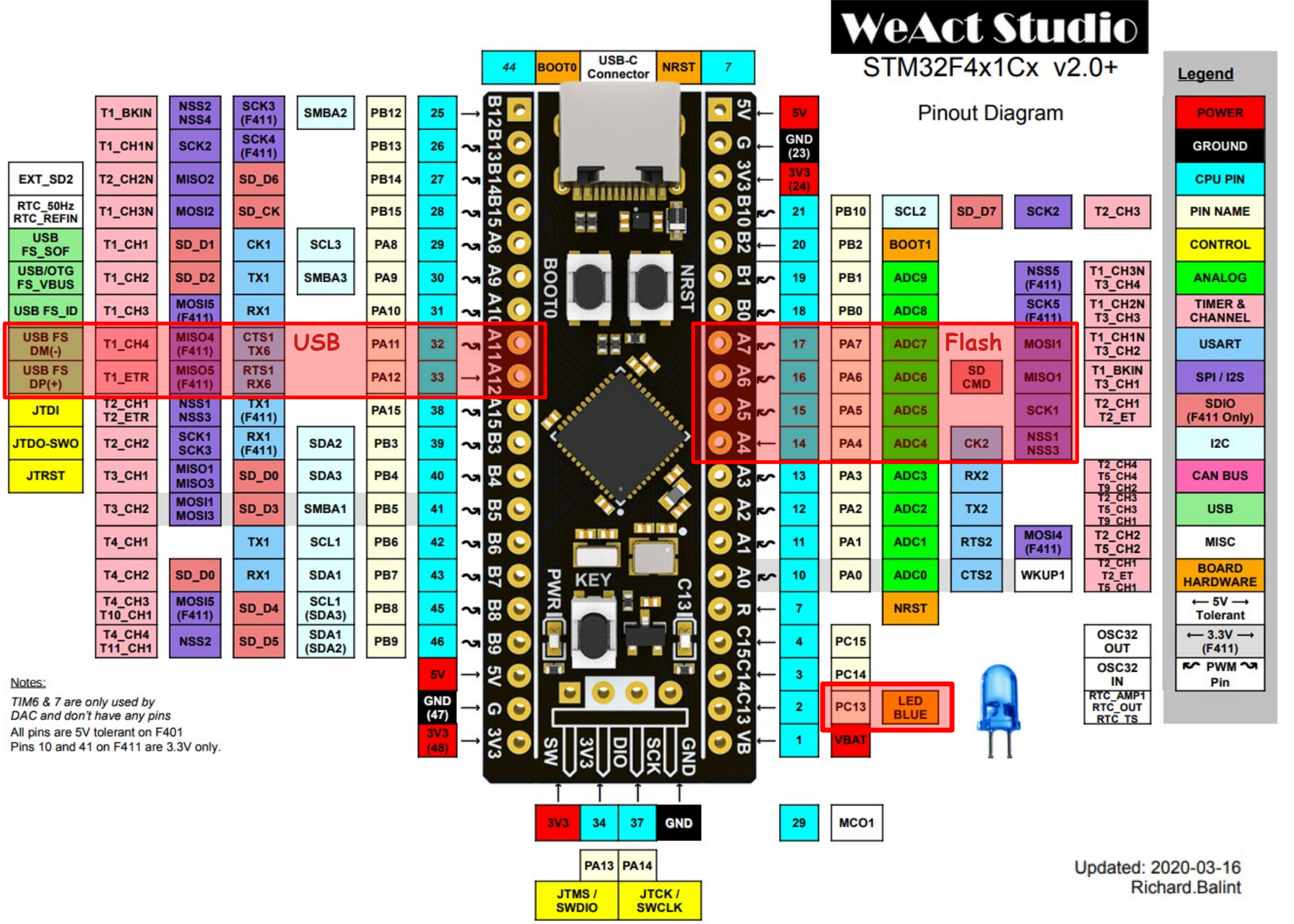
```
roundrect = RoundRect(5,5,31,41,8,fill=0xFFFF00,outline=0xFF00FF,stroke=3)
splash.append(roundrect)
```

Kitöltött kék négyzet rajzolása:

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

Vastag, fekete körvonalú, üres téglalap rajzolása. Kitöltő szín nincs megadva, ezért a közepe átlátszó (ez az alapértelmezett "szín")

```
rect2 = Rect(25,74,31,41,outline=0x0,stroke=3)
splash.append(rect2)
```


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
Flash
BOOT0
NRST
KEY
PWR
C13
SW
3V3
DIO
SCK
GND
LED BLUE
USB-C Connector
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