

AZ-Delivery

Willkommen!

Vielen Dank, dass Sie sich für unser AZ-Delivery 1,77" TFT LCD SPI Display entschieden haben. In den nachfolgenden Seiten werden wir Ihnen erklären, wie Sie das Gerät einrichten und nutzen können.

Viel Spaß!



Az-Delivery

Die Flüssigkristallanzeige, engl. Liquid Crystal Display oder LCD, ist ein Gerät, das mit Hilfe von Flüssigkristallen Text oder Grafiken anzeigen kann. Dabei emittieren diese Kristalle kein Licht, sondern blockieren das Licht der Hintergrundbeleuchtung und erzeugen so das Bild.

Bei den monochromen LCDs sehen wir also nur die Schatten auf dem Bildschirm. Bei farbigen LCDs blockieren Farbfilter bestimmte Farbanteile der Hintergrundbeleuchtung und erzeugen so ein farbiges Bild.

TFT LCDs oder kurz nur TFTs, also Thin Film Transistor Liquid Crystal Display, stellen eine Variante der LCD Bildschirm mit der sogenannten Dünnschicht Transistor Technologie dar. Ohne weiter ins Details zu gehen, wird bei der TFT-Technologie die Größe der Flüssigkristall-Filter verkleinert, so dass entweder die Auflösung, d.h. die Anzahl der Pixel, für bessere Lesbarkeit vergrößert wird, oder kleinere Displays gebaut werden können.

Ihre 1,77" TFT LCD Anzeige kann auf 128x160 Pixel bis zu 65536 verschiedene Farben anzeigen. Der Treiberbaustein der Anzeige heißt "ST7735". Dieser nutzt für den Datenaustausch mit dem Mikro-Controller das Serial Peripheral Interface, kurz SPI.

Spezifikation

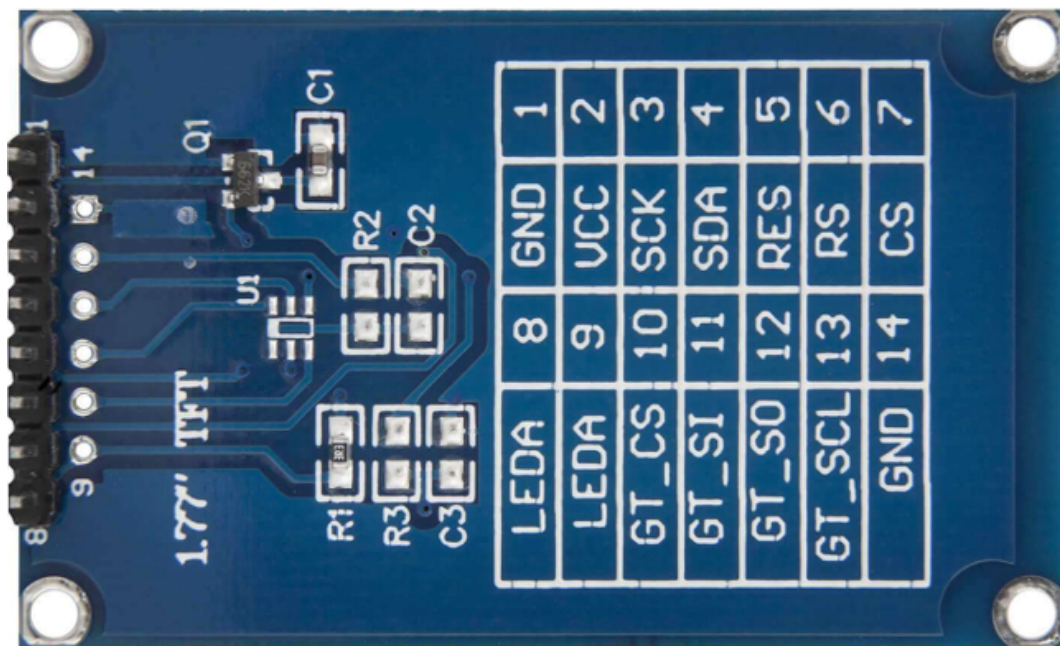
» Betriebsspannung:	von 2.7V bis 3.3V DC
» Betriebsstrom:	~50mA
» Typ :	LCD TFT
» Treiber IC:	ST7735
» Schnittstelle:	SPI interface
» Nenngröße:	1,77" diagonal
» Auflösung:	128x160 Pixels
» Farben:	16 bit oder 65536
» Baugröße (Anzeige):	32 x 38mm [1.26 x 1.50in]
» Baugröße (Platine):	34 x 56mm [1.34 x 2.20in]

Für die Ansteuerung dieser Art von Klein-Bildschirmen werden sowohl für die Atmega328P-Familie als auch die Raspberry Pis sinnvollerweise Programm-Bibliotheken verwendet. Diese kommen häufig mit einer Vielzahl von Beispielprogrammen, die geometrische Figuren wie Linien, Rechtecke, Kreise/Ellipsen oder Text anzeigen. Die von uns verwendete Adafruit-Bibliothek liefert Beispiele für den ESP32 oder ESP8266.

Der *ATMega328* Micro-Controller hat eigentlich nicht genügend Speicher für aufwendige Grafiken; deshalb haben wir hier ein einfaches Beispiel für den *ATMega328* Micro-Controller gewählt.

Pinbelegung

LEDA
CS
RS
RES
SDA
SCK
VCC
GND



Anmerkung: Pins 9 bis 14 werden nicht benutzt

Az-Delivery

Verbinden Sie den *LEDA* pin mit +3.3V, um die Hintergrundbeleuchtung einzuschalten. Zum Dimmen der Beleuchtung kann man entweder zusätzlich ein Potentiometer verbauen oder einen PWM-fähigen Pin benutzen. Die Hintergrundbeleuchtung verbraucht weniger als 1 mA Strom.

Warnung: Schließen Sie den *LEDA* pin niemals direkt an +5V an; das würde den Bildschirm beschädigen!

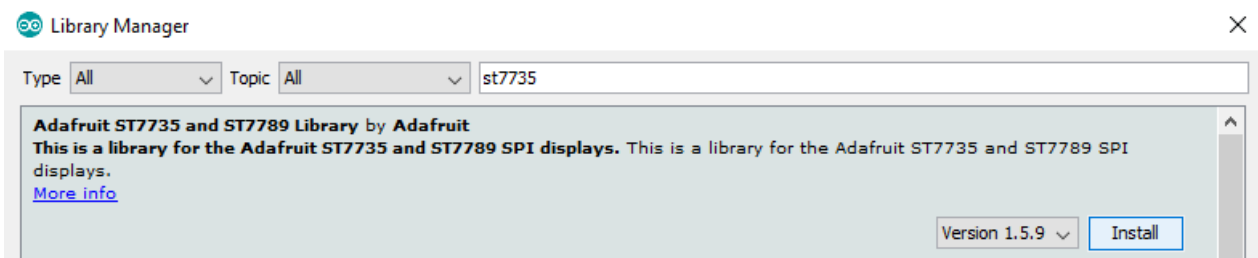
Die Spannungsversorgung der Platine VCC wird an 5V angeschlossen, intern sofort über einen Spannungsregler vom Typ 662K auf 3,3V heruntergeregelt.

Die weiteren Pins gewährleisten die Kommunikation zwischen Micro-Controller und Bildschirm, vornehmlich die SPI-Schnittstelle.

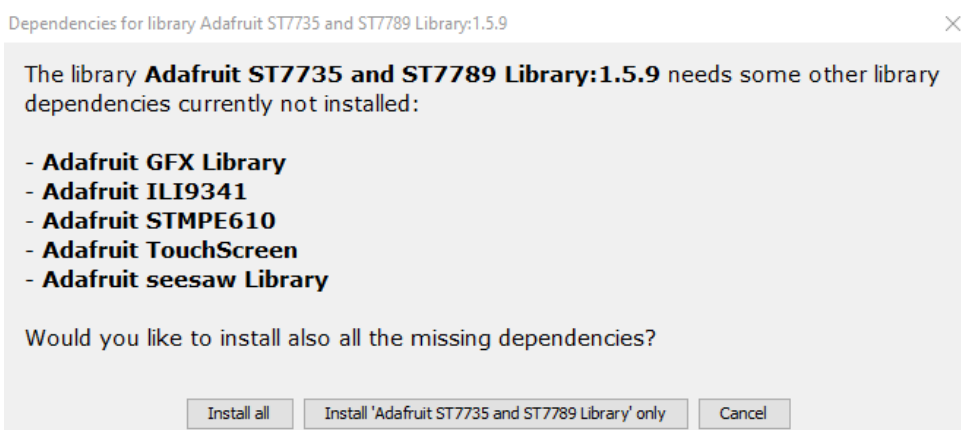
Warnung: Gemäß Datenblatt des Herstellers vertragen diese Datenpins nur 3,3V. Für den sicheren Betrieb an einem 5V Micro-Controller empfehlen wir dringend einen Spannungskonverter, einen sogenannten logic level converter (LLC) (siehe unten). Nichtsdestotrotz haben wir auch den direkten Anschluss an die SPI Pins des *ATMega328* Micro-Controller ausprobiert und es hat funktioniert, ohne dass das Display Schaden nahm.

1,77" Bildschirm und ATmega328 Micro-Controller

Um das 1,77" TFT LCD SPI Display mit dem *ATmega328* Micro-Controller, wird empfohlen, eine vorhandene Programm-Bibliothek im Internet herunter zu laden und zu installieren. Wir empfehlen und nutzen hier "*Adafruit_ST7735*", die innerhalb der Arduino IDE mit dem Bibliotheksverwalter installiert wird: *Werkzeuge > Bibliotheken verwalten*. Dabei wird ein neues Fenster geöffnet, das zunächst die Liste der Programmpakete aktualisiert. Nach wenigen Sekunden kann man dann im Suchfenster "*ST7735*" eingeben und die angebotene Bibliothek "*Adafruit_ST7735 and ST7789 Library by Adafruit*" wie auf dem folgenden Bild zu installieren.

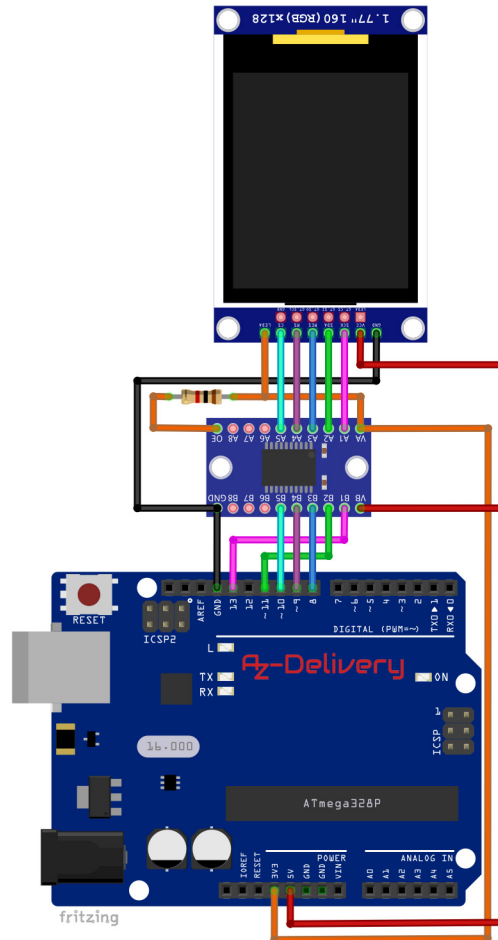


Wenn Sie auf *Installieren* klicken, öffnet sich hier ein weiteres Fenster, das Sie dazu auffordert, weitere erforderliche Pakete zu installieren. Klicken Sie links im Fenster auf "Alle installieren".



Az-Delivery

Verbinden Sie das 1,77" TFT LCD SPI Display nun wie auf dem Schaltbild mit dem *ATmega328* Micro-Controller:



TFT LCD pin	>	LLC pin	>	Pin
SCK	>	A1 - B1	>	D13
SDA (MOSI)	>	A2 - B2	>	D11
RES (RESET)	>	A3 - B3	>	D8
RS (REG. SEL.)	>	A4 - B4	>	D9
CS (SS)	>	A5 - B5	>	D10
LEDA	>	VA	>	3.3V
VCC	>	VB	>	5V
GND	>	GND	>	GND
	>	OE (via 1kΩ)	>	3.3V

Rosa Draht

Grüner Draht

Blauer Draht

Lila Draht

Cyan Draht

Orange Draht

Roter Draht

Schwarzer Dr.

Orange Draht



Anmerkung: Die Logik Pins des 1,77" Bildschirms sind gemäß Hersteller-Datenblatt nicht 5V tolerant, so dass wir hier in der Schaltung einen sogenannten "logic level converter" (oder shifter), oder kurz LLC verwenden. AZ-Delivery bietet ein geeignetes Bauteil mit acht Kanälen an: "*TXS0108E Logic level converter*". Weitere Einzelheiten finden Sie im kostenlosen eBook:

https://www.az-delivery.de/products/logiklevel-wandler-kostenfreies-e-book?_pos=1&_sid=06ac64d83&_ss=r



Sketch Beispiel

Mit der Installation der Adafruit Bibliothek werden auch etliche Beispielprogramme abgespeichert, die unter Datei > *Beispiel* > Adafruit ST7735 and ST7789 gefunden werden können. Wie bereits oben erwähnt, erfordern die meisten Beispielprogramme die Rechenleistung eines ESP32 oder ESP8266 Micro Controllers. Deshalb haben wir das Programm **graphicstest** für den Mikrocontroller Board mit ATmega328P angepasst.

```
#include <Adafruit_GFX.h>    // Core graphics library
#include <Fonts/FreeSansBold9pt7b.h>
#include <Adafruit_ST7735.h> // Hardware-specific library
#include <SPI.h>

Adafruit_ST7735 tft = Adafruit_ST7735(10, 9, 8);

void setup(void) {
  tft.initR(INITR_GREENTAB);
  tft.fillScreen(ST77XX_BLACK);
  delay(500);

  show_page();
  tft.setTextColor(ST77XX_WHITE, ST77XX_BLACK);
  tft.setTextSize(2);
  tft.setFont(); // reset font to default
}

void loop() {
  for(uint8_t i = 0; i < 100; i++) {
    changing_value(i);
    // delay(100);
  }
}
```

Az-Delivery

}

AZ-Delivery

```
void show_page() {
    tft.setFont(&FreeSansBold9pt7b);
    tft.fillScreen(ST77XX_BLACK);
    tft.setTextColor(ST77XX_RED);
    tft.setCursor(14, 22);
    tft.print("AZ-Delivery");

    tft.drawFastHLine(0, 35, 128, ST77XX_GREEN);

    tft.drawTriangle(1, 45, 28, 70, 55, 45, ST77XX_WHITE);
    tft.fillTriangle(78, 70, 104, 45, 127, 70, 0xA3F6);

    tft.drawRect(1, 80, 50, 30, ST77XX_BLUE);
    tft.fillRoundRect(78, 80, 50, 30, 5, 0x2D4E);

    tft.fillCircle(25, 135, 15, 0x5BA9);
    tft.drawCircle(102, 135, 15, ST77XX_GREEN);

    tft.drawLine(45, 150, 80, 40, ST77XX_ORANGE);
}

void changing_value(uint8_t value) {
    if(value < 10) {
        tft.setCursor(15, 88);
        tft.print("0");
        tft.print(value);
    }
    else {
        tft.setCursor(15, 88);
        tft.print(value);
    }
}
```

Az-Delivery

Der Sketch startet mit dem Importieren von vier Programmbibliotheken. Die erste ist die sogenannte `core graphics library`, eine querschnittlich verwendete Bibliothek von Adafruit. Die zweite lädt einen bestimmten Zeichensatz (font), die dritte ist die Hardware-spezifische Unterstützung für den Treiberbaustein ST7735, und die vierte ist notwendig für die SPI Schnittstelle.

Wir instantiieren ein Object mit dem Namen `"tft"` mit:

```
Adafruit_ST7735 tft = Adafruit_ST7735(10, 9, 8)
```

Hierbei repräsentiert das erste Argument den Pin 10 des Mikrocontroller Board mit ATmega328P, an dem der CS Pin des Displays angeschlossen wird; 9 steht für den digitalen I/O pin des Atmega328P Board, an dem der RS Pin des Displays angeschlossen wird; und, 8 steht für den digitalen I/O pin des Mikrocontroller Board mit ATmega328P, an dem der RES Pin des Displays angeschlossen wird. Dieses Objekt `tft` wird beim Set-up und den Einstellungen des Displays angewendet.

In der Funktion `setup()` initialisieren wir das Object `tft` mit:

```
tft.initR(INITR_GREENTAB);
```

Diese Befehlszeile ist nur dem 1,77" TFT LCD SPI Display zugeordnet. Wenn die Bibliothek für andere Bildschirme verwendet wird, sieht diese Zeile anders aus.

Az-Delivery

Die Programmbibliothek stellt uns vordefinierte Funktionen ("built-in functions") zur Verfügung, um z.B. den ganzen Bildschirm mit einer bestimmten Farbe zu füllen: `tft.fillScreen(ST77XX_BLACK)`, wobei `ST77XX_BLACK` die vordefinierte Farbe schwarz bedeutet.

Folgende Farben sind vordefiniert:

<code>ST77XX_BLACK</code>	<code>ST77XX_WHITE</code>
<code>ST77XX_RED</code>	<code>ST77XX_GREEN</code>
<code>ST77XX_BLUE</code>	<code>ST77XX_CYAN</code>
<code>ST77XX_MAGENTA</code>	<code>ST77XX_YELLOW</code>
<code>ST77XX_ORANGE</code>	

Andere Farben können mit einer 4-stelligen Hexadezimalzahl gewählt werden (Zum Beispiel: `0x2AFF`).

Dann wird die Funktion "`show_page()`" ausgeführt, die weiter unten erklärt wird.

Am Ende der Funktion `setup()` werden drei Funktionen für die Einstellungen zum Anzeigen von Text verwendet.

Die Namen der Funktionen sind selbsterklärend. Die Argumente für "`setTextColor()`" legen die Farben für den Text sowie den Hintergrund des Textes fest. Das zweite Argument ist optional.

Das Argument für `setTextSize()` ist eine Integerzahl zwischen 1 und 5 für die Größe des Textes.

Az-Delivery

Mit der dritten Funktion wird die Schriftart (font) festgelegt, z.B. `setFont(&FreeSansBold9pt7b)`. Wird kein Argument übergeben, wird die voreingestellte Schriftart (default font) genommen.

Mögliche Schriftarten finden Sie unter:

... > *Arduino* > *libraries* > *Adafruit_GFX_Library* > *Fonts*

Vor dem Importieren einer bestimmten Schriftart ist die *Adafruit_GFX* Bibliothek einzubinden, zum Beispiel:

```
#include <Adafruit_GFX.h>
```

```
#include <Fonts/FreeSansBold9pt7b.h>
```

Eine Zusammenstellung aller Fonts aus der *Adafruit_GFX* Bibliothek finden Sie auch am Ende des eBooks.

In der Funktion `loop()` wird in einer sich wiederholenden for-Schleife eine Zählvariable bis 99 hochgezählt, anschließend die selbst-definierte Funktion `changing_value()` aufgerufen; diese wird weiter unten erklärt.

In der selbst-definierten Funktion `show_page()` werden mehrere eingebaute Funktionen der *Adafruit_ST7735* Bibliothek aufgerufen.

Die ersten drei dieser Funktionen wurden bereits erklärt: `setFont()`, `setTextColor()` und `fillScreen()`.

Mit der Funktion wird die Position des Cursors festgelegt. Koordinatenursprung (0,0) ist die obere linke Ecke. Die x-Koordinate kann Integer-Werte zwischen 0 und 127 annehmen, Die y-Koordinate Werte zwischen 0 und 159. Diese Funktion muss vor der Funktion `print()`

Az-Delivery

ausgeführt werden, die ansonsten die letzte bekannte Cursor-Position verwendet.

Mit der Funktion `print()` wird der auszugebende Text übergeben. Der Datentyp kann dabei String, integer number, float/double number oder character sein. Bei Datentyp Integer ist ein optionales zweites Argument mit folgenden Daten möglich: *DEC* = decimal, *OCT* = octal oder *HEX* = hexadecimal.

Eine besondere Art der `print()`-Funktion stellt `println()` dar. Damit wird der Cursor nach der Ausgabe in die nächste Zeile gesetzt.

Die nächsten Funktionen zeichnen geometrische Figuren:

`drawFastHLine()` mit vier Argumenten zeichnet eine horizontale Linie. Die ersten beiden Argumente legen die x- und y-Koordinaten des Startpunkt fest, das dritte Argumente die Länge und das vierte Argument die Farbe der Linie.

`drawTriangle()` mit sieben Argumenten zeichnet ein Dreieck. Die ersten sechs Argumente sind jeweils die x- und y-Koordinaten der drei Ecken, das siebte Argument wieder die Farbe der Linien.

Die gleiche Systematik verwendet `fillTriangle()`. Dabei ist das siebte Argument die Füllfarbe des Dreiecks.

`drawRect()` mit fünf Argumenten zeichnet ein Rechteck. Die Koordinaten der oberen linken Ecke und der unteren rechten Ecke sowie die Farbe der Linien werden eingegeben.

Az-Delivery

Die gleiche Systematik verwendet auch `fillRect()`. Dabei ist das fünfte Argument die Füllfarbe des Rechtecks.

`drawRoundRect()` mit sechs Argumenten zeichnet ein Rechteck mit abgerundeten Ecken. Nach den Koordinaten gibt das fünfte Argument den Kurvenradius der abgerundeten Ecke an, die Farbe der Linie folgt als sechstes und letztes Argument.

Das gleiche macht `fillRoundRect()`, wobei das letzte Argument die Füllfarbe ist.

`drawCircle()` mit vier Argumenten zeichnet einen Kreis um den Punkt, dessen x- und y-Koordinate in den ersten beiden Argumente steht. Das dritte übergibt den Radius des Kreises und das vierte wieder die Linienfarbe

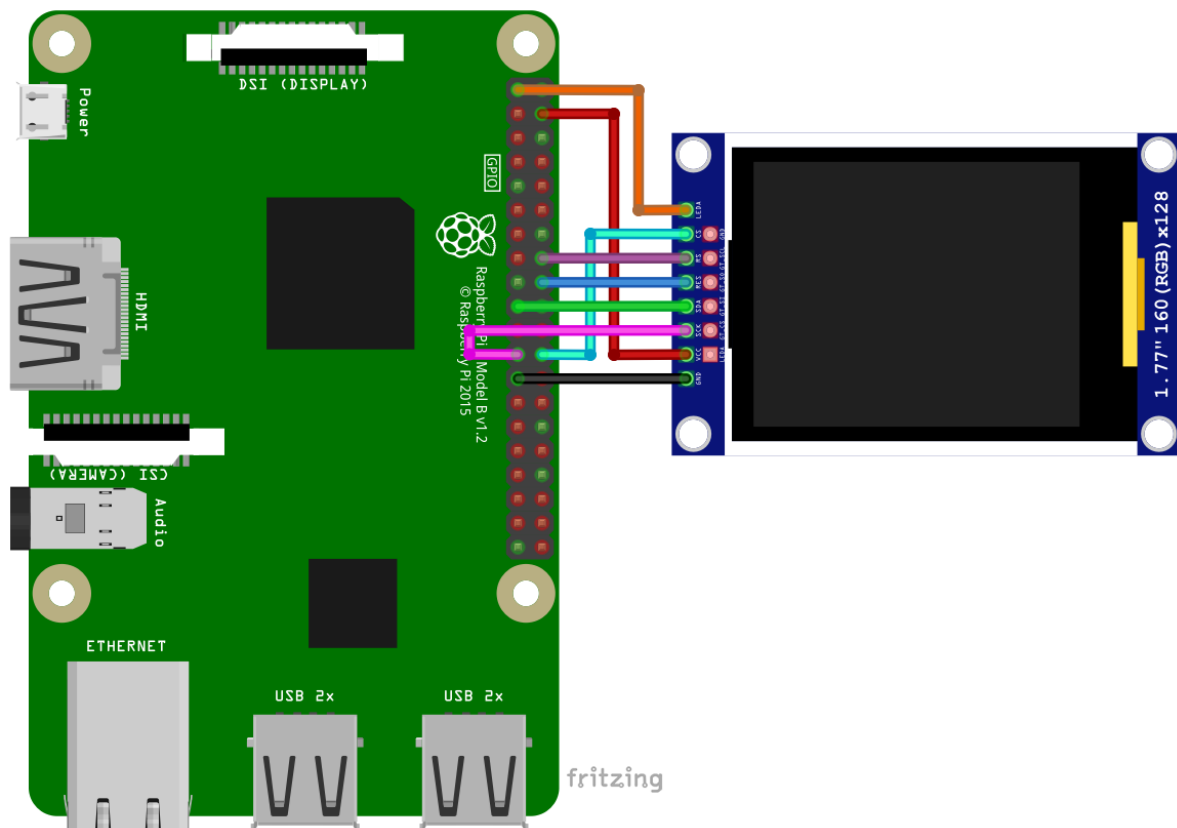
Bei `fillCircle()` ist das vierte Argument wieder die Füllfarbe.

`drawLine()` mit fünf Argumenten zeichnet eine Linie zwischen zwei Punkten mit der als fünftes Argument angegebenen Linienfarbe.

Die selbst-definierte Funktion `changing_value()` gibt die Zählvariable aus. Die Ausgabe beginnt an der mit `setCursor` angegebenen Cursor-Position. Bei Zahlen kleiner als zehn wird eine 0 vorangestellt.

Anschluss an den Raspberry Pi

Verbinden Sie das 1,77" TFT LCD SPI Display mit einem Raspberry Pi wie auf dem folgenden Schaltbild:



TFT LCD pin

LEDA

CS (SS)

RS (REG. SEL.)

RES (RESET)

SDA (MOSI)

SCK

VCC

GND

> Raspberry Pi pin

> 3.3V [pin 1]

> GPIO8/CE0 [pin 24]

> GPIO23 [pin 16]

> GPIO24 [pin 18]

> GPIO10/MOSI [pin 19]

> GPIO11/SCLK [pin 23]

> 5V [pin 4]

> GND [pin 25]

Orange Draht

Cyan Draht

Lila Draht

Blauer Draht

Grüner Draht

Rosa Draht

Roter Draht

Schwarzer Dr.

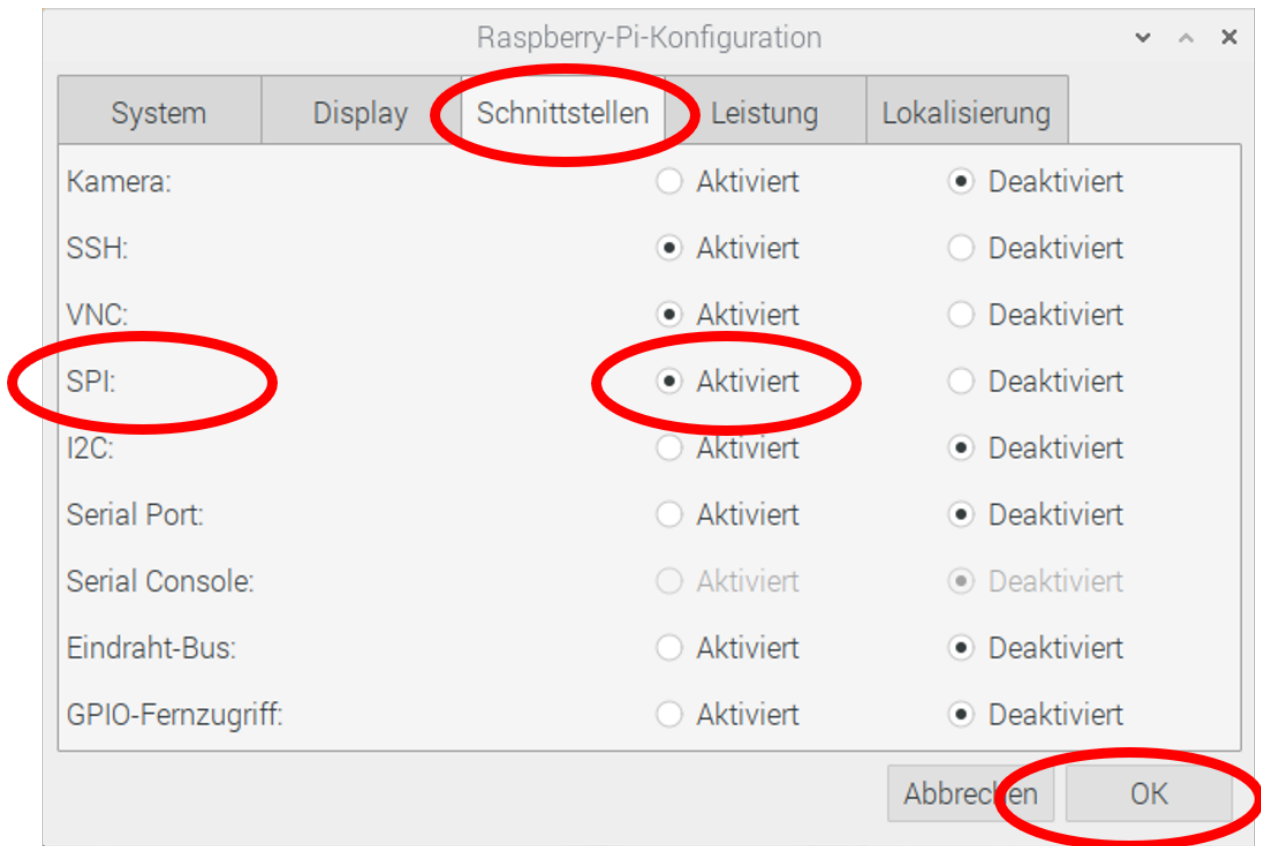
Aktivieren der SPI Schnittstelle

Bevor das 1,77" TFT LCD SPI Display am Raspberry Pi betrieben werden kann, ist die SPI Schnittstelle im Betriebssystem Raspberry Pi OS (bislang bekannt als Raspbian) zu aktivieren. Dazu öffnen Sie die Konfiguration:

Menü > Einstellungen > Raspberry-Pi-Konfiguration

Ein neues Fenster wird geöffnet. Klicken Sie auf die Registerkarte Schnittstellen (engl. *Interfaces*), dann bei *SPI auf Aktiviert* (engl. *Enable*) und schließlich auf den *OK* button, like on the following image:

Az-Delivery





Installation der Python-Programmbibliothek

Für die Verwendung des 1,77" TFT LCD SPI Displays mit dem Raspberry Pi wird die Installation einer speziellen Programmbibliothek empfohlen.

Eine gut geeignete Bibliothek, die für viele Displays angewendet wird, heißt "*luma.examples*". Vor dem Download und der Installation wird wie üblich das Betriebssystem aktualisiert.

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

Die Bibliothek wird wie folgt im Terminal heruntergeladen:

```
git clone https://github.com/rm-hull/luma.examples.git
```

Dann wechselt man in das neue Verzeichnis *luma.examples*:

```
cd luma.examples
```

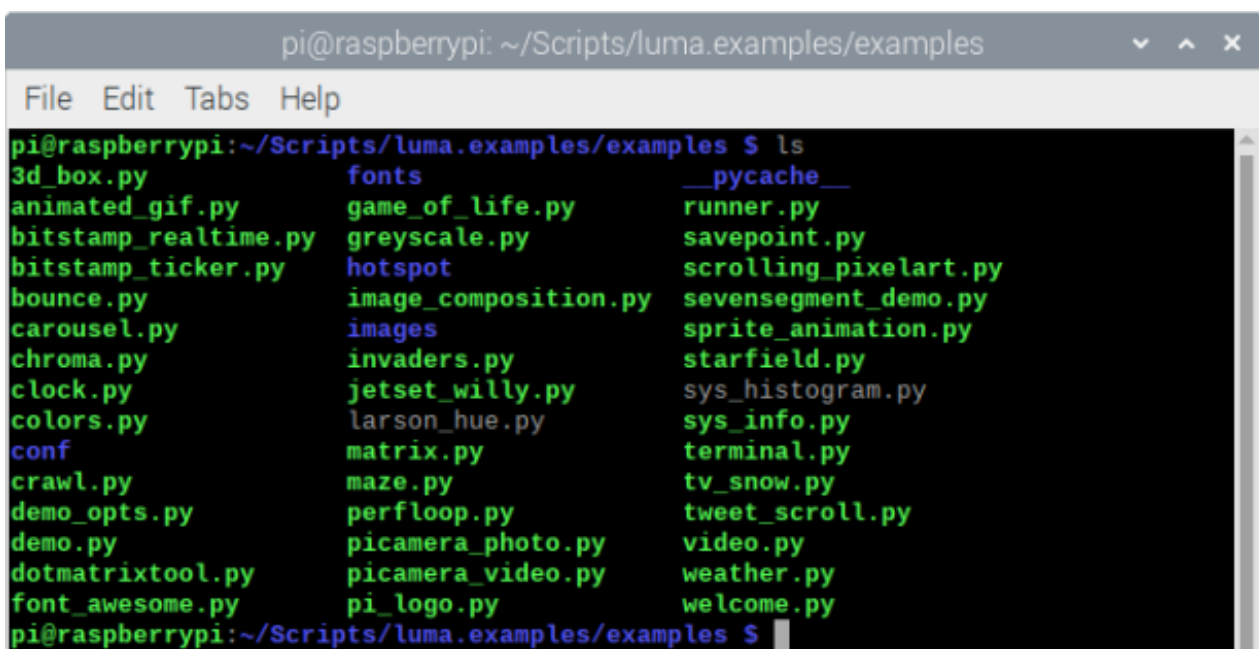
Und installiert die Bibliothek mit dem folgenden Befehl (mit dem Punkt!):

```
sudo -H pip3 install -e .
```

Az-Delivery

Beispielprogramm

Wenn man das Unterverzeichnis `luma.examples/examples` öffnet, sieht man eine Liste mit vielen Beispielen, siehe nachfolgendes Bild:



```
pi@raspberrypi: ~/Scripts/luma.examples/examples
File Edit Tabs Help
pi@raspberrypi:~/Scripts/luma.examples/examples $ ls
3d_box.py          fonts              __pycache__
animated_gif.py    game_of_life.py   runner.py
bitstamp_realtime.py greyscale.py       savepoint.py
bitstamp_ticker.py hotspot           scrolling_pixelart.py
bounce.py          image_composition.py sevensegment_demo.py
carousel.py        images            sprite_animation.py
chroma.py          invaders.py        starfield.py
clock.py           jetset_willy.py   sys_histogram.py
colors.py          larsen_hue.py     sys_info.py
conf              matrix.py          terminal.py
crawl.py           maze.py            tv_snow.py
demo_opts.py       perfloop.py        tweet_scroll.py
demo.py            picamera_photo.py  video.py
dotmatrixtool.py   picamera_video.py  weather.py
font_awesome.py    pi_logo.py         welcome.py
pi@raspberrypi:~/Scripts/luma.examples/examples $
```

Die luma.Bibliothek kann wie erwähnt für viele verschiedene Displays verwendet werden. Die oben gezeigten Python Beispiele haben als Voreinstellung (default) die Displays mit SSD1306-Treiber, I2C-Schnittstelle und I2C-Adresse 0x3C.

Deshalb funktioniert das 1,77" TFT mit seinem ST7735-Treiber und SPI-Schnittstelle nicht ohne umfangreiche Änderungen der Konfigurationseinstellungen. Diese müssten für dieses Display angepasst werden. Diese Änderungen haben wir im folgenden Beispielprogramm eingearbeitet.

AZ-Delivery

Script example

```
# If not yet done please install the luma library by
# git clone https://github.com/rm-hull/luma.examples.git
# cd luma.examples

# sudo -H pip3 install -e .      (mind the dot at the end)

import time
import sys
from luma.core.interface.serial import spi
from luma.lcd.device import st7735
from luma.core.render import canvas
from PIL import ImageFont

font_path = '/home/luma.examples/examples/fonts/ChiKareGo.ttf'
s = spi(port=0, device=0, cs_high=True, gpio_DC=23, gpio_RST=24)
device=st7735(s, rotate=0, width=160, height=128, h_offset=0, v_offset=0, bgr=False)

def primitives(device, draw):
    # Draw a rectangle of the same size of screen
    draw.rectangle(device.bounding_box, outline='white')
    # Draw a rectangle
    draw.rectangle((4, 4, 80, 124), outline='blue', fill=(22, 55, 55))
    # Draw an ellipse
    draw.ellipse((6, 6, 78, 122), outline=(254, 155, 0), fill='green')
    # Draw a triangle
    draw.polygon([(90,124), (100,4), (120,124)], outline='blue', fill='red')
    # Draw an X
    draw.line((130, 4, 155, 124), fill='yellow')
    draw.line((130, 124, 155, 4), fill='yellow')
    # Print 'AZ-Delivery'
    draw.text((10, 60), 'AZ-Delivery', fill='red')
    # Change font and size of text and print 'AZ-Delivery'
    size = 22
    new_font = ImageFont.truetype(font_path, size)
    draw.text((10, 70), 'AZ-Delivery', font=new_font, fill='red')

def changing_var(device):
    size = 40
    new_font = ImageFont.truetype(font_path, size)
```

Az-Delivery

```
for i in range(100):
    with canvas(device) as draw:
        draw.text((40, 38), 'Changing var.', fill='red')
        if i < 10:
            draw.text((63, 55), '0{}'.format(str(i)), font=new_font, fill='red')
        else:
            draw.text((63, 55), str(i), font=new_font, fill='red')
        time.sleep(0.02)

print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        device.backlight(False)    # if backlight connected to GPIO18
        print('Testing printing variable.\n')
        changing_var(device)
        time.sleep(1)

        print('Testing basic graphics.\n')
        with canvas(device) as draw:
            primitives(device, draw)
        time.sleep(3)

        print('Testing display ON/OFF.\n')
        for _ in range(5):
            time.sleep(0.5)
            device.hide()
            time.sleep(0.5)
            device.show()

        print('Testing clear display.\n')
        time.sleep(2)
        device.clear()
        print()
        time.sleep(2)

except KeyboardInterrupt:
    device.backlight(True)    # if backlight connected to GPIO18
    device.cleanup()
    print('Script end!')
```

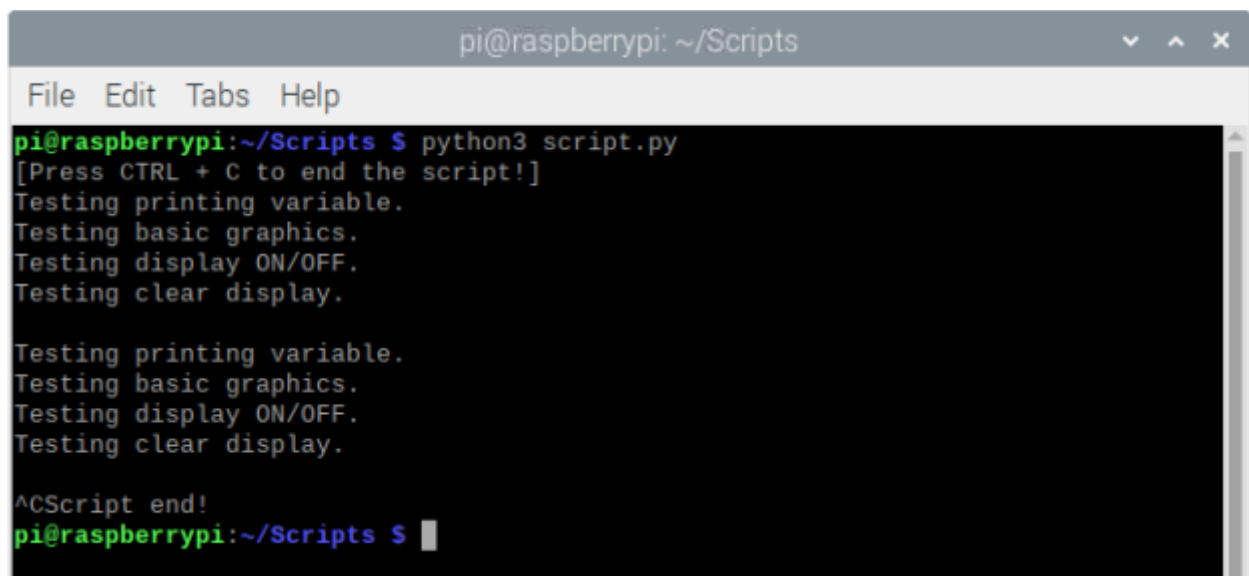
Az-Delivery

Sichern Sie das Python-Programm unter dem Namen “*script.py*”

Öffnen Sie das Terminal, wechseln Sie ggf. In das Verzeichnis mit dem Programm und starten Sie es mit:

python3 script.py

Die Ausgabe im Terminal sollte wie folgt aussehen:

A screenshot of a terminal window titled "pi@raspberrypi: ~/Scripts". The window has a menu bar with "File", "Edit", "Tabs", and "Help". The terminal output shows the command "python3 script.py" being executed. The script prints several test messages: "[Press CTRL + C to end the script!]", "Testing printing variable.", "Testing basic graphics.", "Testing display ON/OFF.", and "Testing clear display.". These messages are repeated twice. After the second repetition, the script ends with "^CScript end!". The prompt "pi@raspberrypi:~/Scripts \$" is visible at the bottom with a cursor.

```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 script.py
[Press CTRL + C to end the script!]
Testing printing variable.
Testing basic graphics.
Testing display ON/OFF.
Testing clear display.

Testing printing variable.
Testing basic graphics.
Testing display ON/OFF.
Testing clear display.

^CScript end!
pi@raspberrypi:~/Scripts $
```

Um das Programm zu beenden, drücken Sie “*Strg* + *C*” auf der Tastatur.

Az-Delivery

Am Anfang des Programms werden die Programmbibliotheken, die Module, importiert.

Damit können zwei Objekte instantiiert werden: *s* und *device*.

Unter der Variablen *font_path* wird der Pfad zum verwendeten Zeichensatz deklariert. Diese Zeile müssen Sie ggf. anpassen.

Das Objekt *s* ermöglicht die Kommunikation zwischen Raspberry Pi und Treiber-IC des Displays über die SPI-Schnittstelle. Dafür wird der Konstruktor namens *spi()* aus der *luma.core.interface.serial* Bibliothek verwendet. Dieser Konstruktor verarbeitet die übergebenen Schlüsselwort-Argumente sowie ggf. weitere vor-definierte Argumente. Diese Argumente sind u.a. *port* und *device* Nummer oder die GPIO-Pin-Zuweisung für die Anschlüsse CD und RST. In unserem Beispiel verwenden wir den SPI port *spidev0.0*, also: *port=0* und *device=0*. *gpio_DC=23* deklariert den Raspberry Pi Pin für den Pin *DC* (RS) des Displays, *gpio_RST=24* entsprechend für Reset (*RES*).

Az-Delivery

Bei der Instantiierung des Objekts *device* werden mittels der Constructors `st7735()` aus der *luma.lcd.device Bibliothek* die Einstellungen für den Treiber-IC des Displays übergeben, in unserem Fall sieben Argumente. Das erste Argument ist das Objekt für die serielle Schnittstelle (das zuvor instantiierte Objekt *s*); das zweite Argument steht für die Rotation der Anzeige (s.u.); das dritte Argument definiert die Breite (*width*), das vierte die Höhe (*height*) des Displays in Pixel; im fünften und sechsten Argument werden ggf. Korrekturen für die Kanten der Anzeige *h_offset* und *v_offset* übergeben; und mit dem siebten Argument wird ggf. die Farbe des Hintergrundes invertiert. In unserem Beispiel nutzen wir *rotate=0*, das bedeutet keine Rotation (Winkel 0°). Als weitere mögliche Werte können die Integerzahlen 1 = 90°, 2 = 180° und 3 = 270° angegeben werden; *width=160* und *height=128* stehen für die Anzahl der Pixel, siehe auch Beschriftung des Displays 160x128.

Bei dem getesteten Display war der Bildschirmrand perfekt. Deshalb sind die Werte für horizontal offset *h_offset=0* und für vertical offset *v_offset=0*. Wenn Sie einen kleinen farbigen Rand bei sich erkennen, experimentieren Sie hier mit kleinen Integerzahlen, z.B. *h_offset=1* oder *v_offset=2*.

Bei dem verwendeten Display haben wir für die Hintergrundfarbe *bgr=False* eingegeben, bei *bgr=True* wurden alle Farben invertiert dargestellt. Diese Werte können bauartbedingt ggf. bei Ihrem Display genau anders herum nötig sein.

Az-Delivery

Die selbst-definierte Funktion `primitives()` akzeptiert zwei Argumente und gibt keinen Wert zurück. Das erste Argument ist unser Objekt `"device"`, das zweite das Objekt `"draw"`, mit dem die Display-Befehle übergeben werden. Damit werden innerhalb der Funktion `primitives()` die vor-definierten Grafik-Funktionen der `luma.core` Bibliothek aufgerufen.

Die erste vor-definierte Funktion ist `rectangle()` mit drei Argumenten, mit dem ein Rechteck gezeichnet wird. Das erste Argument ist ein Tupel mit vier Zahlen, der x- und y-Position der oberen linken Ecke sowie der x- und y-Position der unteren rechten Ecke. Dabei ist die oberste linke Ecke des Displays der Punkt (0,0). Anstelle des Tupels kann `bounding_box` eingegeben werden; dann wird das Rechteck auf dem gesamten Bildschirm gezeichnet.

Das zweite Argument ist ein Schlüsselwort-Argument mit dem Namen `"outline"`. Hiermit wird die Rahmenfarbe des Rechtecks festgelegt. Das dritte Argument ist ein Schlüsselwort-Argument mit dem Namen `"fill"`. Hiermit wird die Füllfarbe des Rechtecks festgelegt. Das zweite und dritte Argument sind optional; wenn diese entfallen, wird ein schwarzes Rechteck mit weißen Rändern gezeichnet.

Für die Angabe der Farbe kann man vordefinierte Farben als "String" oder ein Zahlentupel mit den drei RGB-Farbanteilen eingeben.

Mehr zu Farben siehe z.B.

<https://pillow.readthedocs.io/en/3.1.x/reference/Imagecolour.html>

Az-Delivery

Die Funktion `ellipse()`, ebenfalls mit drei Argumenten, zeichnet eine Ellipse auf dem Bildschirm. Das erste Argument ist ein Tupel mit vier Zahlen, der x- und y-Position der oberen linken Ecke eines Rechtecks, das die Ellipse umschließt, sowie der x- und y-Position der unteren rechten Ecke dieses Rechtecks.

Das zweite und dritte Argument bestimmen wieder die Rahmen- und Füllfarbe der Ellipse; sie sind ebenfalls optional.

Die Funktion `polygon()` mit drei Argumenten in der gleichen Systematik beschreibt ein Vieleck. Dabei ist das erste Argument diesmal eine Liste mit mindestens drei Tupeln für die Positionen der Eckpunkte. Die Anzahl der Tupel in der Liste bestimmt damit die Anzahl der Eckpunkte des Vielecks. Der Polygonzug geht von einem Punkt zum nächsten und vom letzten zurück zum Anfangspunkt.

Das zweite und dritte Argument bestimmen wieder die Rahmen- und Füllfarbe des Vielecks; sie sind ebenfalls optional.

Die Funktion `line()` zum Zeichnen einer Linie akzeptiert zwei Argumente: Das erste Argument ist ein Tupel mit vier Zahlen, der x- und y-Position des Anfangspunktes sowie der x- und y-Position des Endpunktes der Linie. Das zweite, optionale Schlüsselwort-Argument `fill` legt die Linienfarbe fest; bei Weglassen des Arguments ist die Linienfarbe weiß.

AZ-Delivery

Die Funktion `text()` mit vier Argumenten schreibt Text auf dem Bildschirm. Das erste Argument ist ein Tupel mit der Cursor-Position. Das zweite Argument enthält den String mit dem auszugebenden Text. Der dritte, optionale Parameter wählt den Schrifttyp aus der `luma.core.legacy.font`-Bibliothek (Erklärung weiter unten). Und der vierte Parameter `fill` ist wie oben für die Farbe.

Werden für diese Parameter keine Werte definiert, wird der default font mit der Farbe weiß dargestellt.

Nun zu den Schriftarten, den fonts. Es gibt einige vor-definierte Schriftarten in der `luma.core`-Bibliothek. Sie finden sie im Unterverzeichnis `luma.examples/examples/fonts`. Einige Schriftarten müssen lizenziert werden. Aber man kann grundsätzlich auch eigene Zeichensätze definieren. (Details dazu in der `luma.core` library documentation).

Um die Fonts zu benutzen, muss man zunächst den Datei-Pfad und die Datei (*.ttf) spezifizieren, dann die Textgröße festlegen. Mit der Funktion `ImageFont.truetype()` aus dem Modul `PIL` wird dann der Zeichensatz kreiert.

Der Programmcode dazu lautet:

```
font_path = '/home/pi/luma...../ChiKareGo.ttf'
size = 22
new_font = ImageFont.truetype(font_path, size)
```

Dann kann dieser neue Zeichensatz als drittes Argument angegeben werden.

```
draw.text((10, 70), 'AZ-Delivery', font=new_font, fill="red")
```

Az-Delivery

In der selbst-definierten Funktion *changing_var()* wird eine Zählvariable in einer for-Schleife angezeigt. Dazu definieren wir zunächst die Schriftart und – gröÙe. Dann verwenden wir erneut die Funktion *text()*.

Dafür instantiieren wir das Objekt *draw* mit dem Befehl

```
with canvas(device) as draw:  
    draw.text((40, 38), 'Changing var.', fill='red')
```

Mit der if-Bedingung werden Zahlen kleiner als 10 mit einer vorangehenden Null dargestellt. Dann wird kurz pausiert (*sleep(0.02)*), damit die einzelnen Zahlen noch erkennbar sind.

Im Hauptprogramm werden in einem try-except-Block die selbst-definierten Funktionen in einer Endlos-Schleife nacheinander aufgerufen mit der Möglichkeit, das Programm ordnungsgemäß mit dem sogenannten KeyboardInterrupt “*Strg + C*” zu beenden. Dann erfolgt das *clean-up()* und im Terminal wird der Text ‘*Script end!*’ angezeigt.



Fonts der Adafruit_GFX library

... > Arduino > libraries > Adafruit_GFX_Library > Fonts

FreeMono12pt7b.h	FreeSansBoldOblique12pt7b.h
FreeMono18pt7b.h	FreeSansBoldOblique18pt7b.h
FreeMono24pt7b.h	FreeSansBoldOblique24pt7b.h
FreeMono9pt7b.h	FreeSansBoldOblique9pt7b.h
FreeMonoBold12pt7b.h	FreeSansOblique12pt7b.h
FreeMonoBold18pt7b.h	FreeSansOblique18pt7b.h
FreeMonoBold24pt7b.h	FreeSansOblique24pt7b.h
FreeMonoBold9pt7b.h	FreeSansOblique9pt7b.h
FreeMonoBoldOblique12pt7b.h	FreeSerif12pt7b.h
FreeMonoBoldOblique18pt7b.h	FreeSerif18pt7b.h
FreeMonoBoldOblique24pt7b.h	FreeSerif24pt7b.h
FreeMonoBoldOblique9pt7b.h	FreeSerif9pt7b.h
FreeMonoOblique12pt7b.h	FreeSerifBold12pt7b.h
FreeMonoOblique18pt7b.h	FreeSerifBold18pt7b.h
FreeMonoOblique24pt7b.h	FreeSerifBold24pt7b.h
FreeMonoOblique9pt7b.h	FreeSerifBold9pt7b.h
FreeSans12pt7b.h	FreeSerifBoldItalic12pt7b.h
FreeSans18pt7b.h	FreeSerifBoldItalic18pt7b.h
FreeSans24pt7b.h	FreeSerifBoldItalic24pt7b.h
FreeSans9pt7b.h	FreeSerifBoldItalic9pt7b.h
FreeSansBold12pt7b.h	FreeSerifItalic12pt7b.h
FreeSansBold18pt7b.h	FreeSerifItalic18pt7b.h
FreeSansBold24pt7b.h	FreeSerifItalic24pt7b.h
FreeSansBold9pt7b.h	FreeSerifItalic9pt7b.h



Sie haben es geschafft. Sie können jetzt unser Modul für Ihre Projekte nutzen.

Jetzt sind Sie dran! Entwickeln Sie Ihre eigenen Projekte und Smart- Home Installationen. Wie Sie das bewerkstelligen können, zeigen wir Ihnen unkompliziert und verständlich auf unserem Blog. Dort bieten wir Ihnen Beispielskripte und Tutorials mit interessanten kleinen Projekten an, um schnell in die Welt der Mikroelektronik einzusteigen. Zusätzlich bietet Ihnen auch das Internet unzählige Möglichkeiten, um sich in Sachen Mikroelektronik weiterzubilden.

Falls Sie noch nach weiteren hochwertigen Produkten für hochwertige Mikroelektronik und Zubehör suchen, sind Sie bei der AZ-Delivery Vertriebs GmbH goldrichtig. Wir bieten Ihnen zahlreiche Anwendungsbeispiele, ausführliche Installationsanleitungen, Ebooks, Bibliotheken und natürlich die Unterstützung unserer technischen Experten.

<https://az-delivery.de>

Have Fun!

Impressum

<https://az-delivery.de/pages/about-us>

Az-Delivery