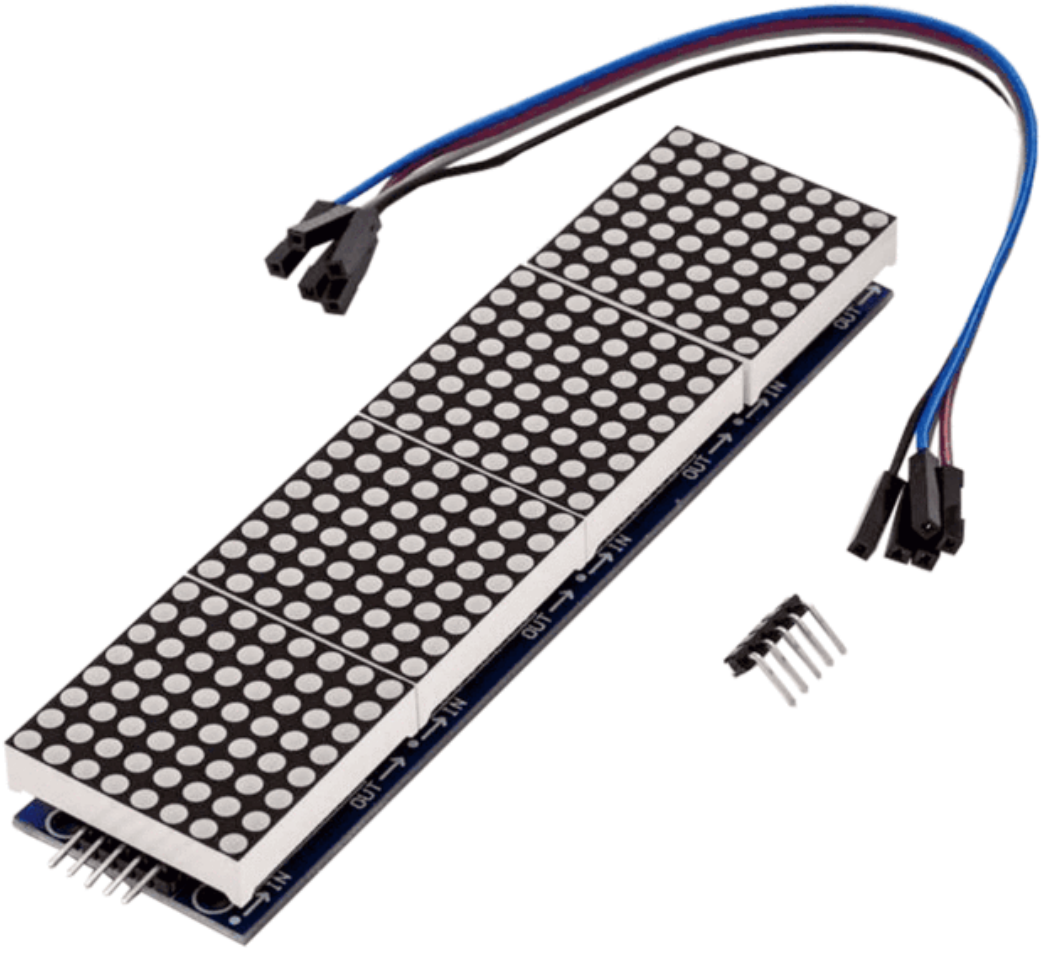
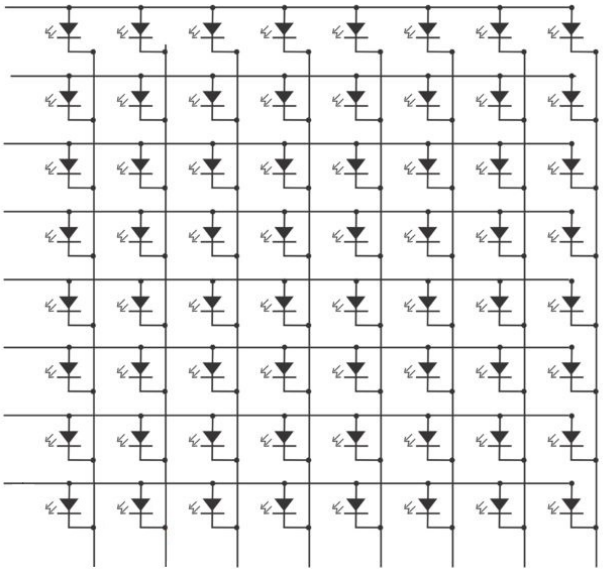
Willkommen!

Vielen Dank, dass Sie sich für unser 4x*64-LED-Matrix-Screen-Modul* von *AZ-Delivery* entschieden haben. In den nachfolgenden Seiten werden wir Ihnen erklären wie Sie das Gerät einrichten und nutzen können.

Viel Spaß!



Der *4x64-LED-Matrix-Bildschirm ist ein Bildschirm mit LED-Anordnungen, die in einer Matrix* aus *4x64 LEDs gestapelt sind*. Der *4x64*-LED-Bildschirm besteht aus vier kleineren *8x8-Bildschirmen,* welche *Segmente* genannt werden. Sie werden 8x8 genannt, weil sie 8 Reihen mit 8 LEDs haben, wodurch eine Matrix von 64 LEDs (*64 Pixel*) erzeugt wird. Jede LED benötigt zwei Drähte (Einen für den Strom und einen für die Masse). Um zu vermeiden, dass alle 64 LEDs einzeln verdrahtet werden, sind die LEDs zu einer LED-Matrix verbunden, wie unten abgebildet:



Durch die Anordnung der LEDs in einer Matrix wird die Anzahl der Ausgangspins auf 16 reduziert, was immer noch zu viel ist, daher wird auf diesem Modul ein Treiberchip, mit SPI-Schnittstelle, mit dem Namen "*MAX7219"* verwendet. Der 4x64-LED-Bildschirm hat vier Segmente mit jeweils einem Treiberchip. Die Treiberchips werden seriell mit anderen Treiberchips in der Daisy-chain verbunden. Es ist möglich, mehr zu stapeln, aber Sie müssen eine externe Stromversorgung für zusätzliche Bildschirme verwenden.

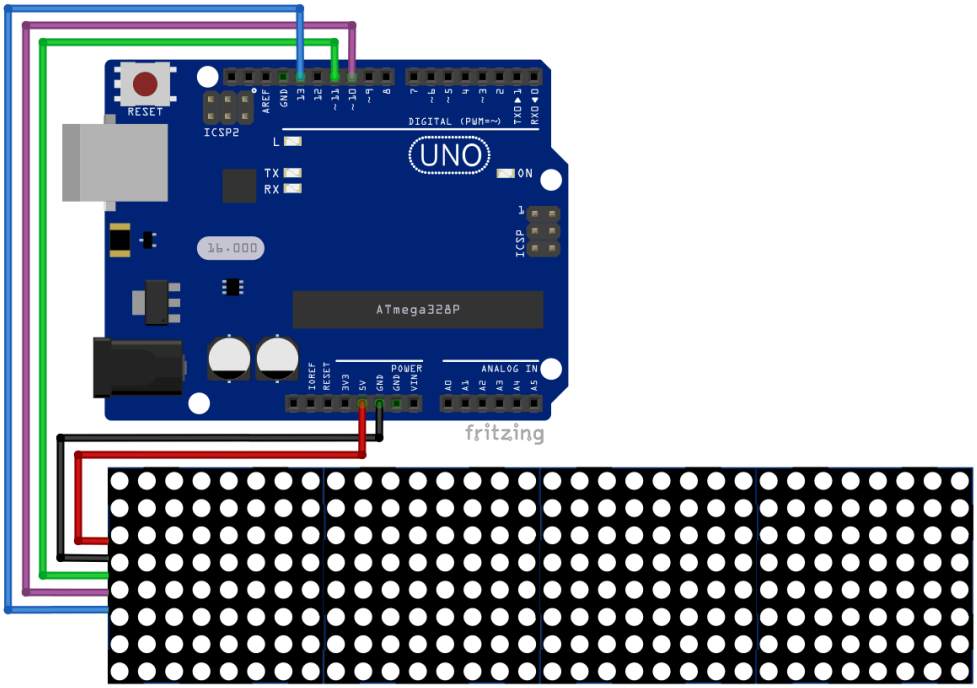
Der *MAX7219*-Chip verwendet die SPI-Schnittstelle zur Kommunikation mit Mikrocontrollern. "*SPI*" steht für eine synchrone serielle Peripherie-Kommunikationsschnittstelle und wird für die Kommunikation über kurze Entfernungen, hauptsächlich in eingebetteten Systemen, verwendet. Synchron bedeutet, dass die Daten innerhalb eines bestimmten Taktzyklus gesendet und empfangen werden, und seriell bedeutet, dass die Daten seriell, Bit für Bit, gesendet werden.

Technische Daten

»	Betriebsspannungsbereich:	+3.3V bis +5V DC
»	Farbe der LEDs:	Rot
»	Helligkeitsregelung:	digital und analog
»	Dimensionen:	32 x 129mm [1,26 x 5,1in]
»	10MHz SPI-Schnittstelle	
»	Anzeige beim Einschalten ausgeblendet	
»	Ansteuerung von LED-Bildschirmen mit gemeinsamer Kathode	

Verbindung des Displays mit dem Uno

Verbinden Sie das Modul mit dem Uno, wie unten abgebildet:

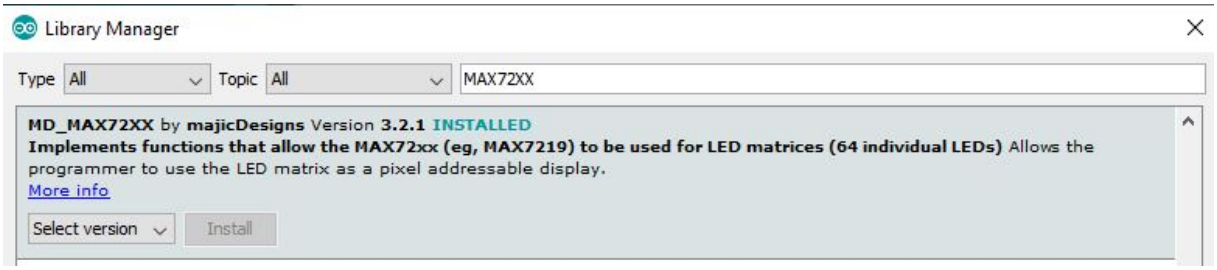


Modul Pin			**Uno Pin**	
VCC		>	5V	**Roter Draht**
GND		>	GND	**Schwarzer Draht**
CS	(SS)	>	D10 pin	**Lila Draht**
DATA	(MOSI)	>	D11 pin	**Grüner Draht**
CLK	(SCK)	>	D13 pin	**Blauer Draht**

Lesen Sie unbedingt die Pin-Bezeichnungen, bevor Sie mit dem Anschließen beginnen, da es verschiedene Versionen des Moduls gibt. Wir haben einige alternative Pinbezeichnungen in Klammern beigefügt.

Library für die Arduino IDE

Um den Bildschirm mit dem Uno zu benutzen, wird empfohlen, eine externe Library herunterzuladen. Gehen Sie zuerst zu: *`Tools > Manage Libraries`* und wenn ein neues Fenster erscheint, geben Sie *MAX72XX* in das Suchfeld ein. Installieren Sie die Library *MD_MAX72XX* von "*majicDesigns"* , wie unten abgebildet:



Drücken Sie auf die "I*nstall*"-Schaltfläche, die erscheint, wenn Sie mit der Maus über das Suchergebnis fahren.

Es gibt viele Versionen dieser Bildschirme. Um festzustellen, welcher verwendet wird, muss der mit der Library mitgelieferte HardwareMapper Sketch ausgeführt werden. Um die Sketch zu öffnen, gehen Sie zu:
File > Examples > MD_MAX72XX > MD_MAX72xx_HW_Mapper,
und ändern Sie folgende Codezeile:

```
#define SERIAL_SPEED 57600
```

into:

```
#define SERIAL_SPEED 9600
```

Laden Sie die Sketch in den Uno und öffnen Sie den Serial Monitor *(`Tools > Serial Monitor`)*. Befolgen Sie die dreistufigen Anweisungen im Serial Monitor, wie unten abgebildet:

```
[MD_MAX72xx Hardware mapping utility]

INTRODUCTION
------------
How the LED matrix is wired is important for the MD_MAX72xx library as different
LED modules are wired differently. The library can accommodate these, but it
needs to know what transformations need to be carried out to map your board to the
standard coordinate system. This utility shows you how the matrix is wired so that
you can set the correct *_HW module type for your application.

The standard functions in the library expect that:
o COLUMNS are addressed through the SEGMENT selection lines, and
o ROWS are addressed through the DIGIT selection lines.

The DISPLAY always has its origin in the top right corner of a display:
o LED matrix module numbers increase from right to left,
o Column numbers (ie, the hardware segment numbers) increase from right to left (0..7), and
o Row numbers (ie, the hardware digit numbers) increase down (0..7).

There are three hardware setting that describe your hardware configuration:
- HW_DIG_ROWS - HardWare DIGits are ROWS. This will be 1 if the digits map to the rows
                of the matrix, 0 otherwise
- HW_REV_COLS - HardWare REVerse COLumnS. The normal column coordinates orientation
                is col 0 on the right side of the display. This will be 1 if reversed.
                (ie, hardware 0 is on the left).
- HW_REV_ROWS - HardWare REVerse ROWS. The normal row coordinates orientation is row
                0 at top of the display. This will be 1 if reversed (ie, row 0
                is at the bottom).

These individual setting then determine the model type of the hardware you are using.

INSTRUCTIONS
------------
1. Wire up one matrix only, or cover up the other modules, to avoid confusion.
2. Enter the answers to the question in the edit field at the top of Serial Monitor.

=====================================================

STEP 1 - DIGITS MAPPING (rows)
------------------------------
In this step you will see a line moving across the LED matrix.
You need to observe whether the bar is scanning ROWS or COLUMNS,
and the direction it is moving.
>> Enter Y when you are ready to start: Y
Dig-0-1-2-3-4-5-6-7
>> Enter Y if you saw ROWS animated, N if you saw COLUMNS animated: Y
>> Enter Y if you saw the line moving BOTTOM to TOP, or enter N otherwise: N

STEP 2 - SEGMENT MAPPING (columns)
----------------------------------
In this step you will see a dot moving along one edge of the LED matrix.
You need to observe the direction it is moving.
>> Enter Y when you are ready to start: Y
Seg-G-F-E-D-C-B-A-DP
>> Enter Y if you saw the LED moving LEFT to RIGHT, or enter N otherwise: N

STEP 3 - RESULTS
----------------
Your responses produce these hardware parameters

HW_DIG_ROWS     1
HW_REV_COLS     0
HW_REV_ROWS     0

Your hardware matches the setting for FC-16 modules. Please set FC16_HW.
```

Um das richtige Ergebnis zu erhalten, richten Sie den Bildschirm wie auf dem Anschlussplan aus. Die linke Seite des Bildschirms ist die Seite, auf der sich die Eingangspins des Bildschirms befinden (wo die Drähte verbunden wurden).

Die Ergebnis-Hardware-Abbildung ist "*FC16_HW*". Wenn Sie also das AZ-Delivery 64 LED-Matrix-Bildschirmmodul verwenden, setzen Sie das Makro von *HARDWARE_TYPE* auf diesen Ergebniswert.

Sketch-Code:

```
#include <MD_MAX72xx.h >

#define PRINT(s, x) { Serial.print(F(s)); Serial.print(x); }
#define PRINTS(x) Serial.print(F(x))
#define PRINTD(x) Serial.println(x, DEC)
#define PRINT(s, x)
#define PRINTS(x)
#define PRINTD(x)

#define HARDWARE_TYPE MD_MAX72XX::FC16_HW

#define MAX_DEVICES  4

#define CLK_PIN   13  // or SCK
#define DATA_PIN  11  // or MOSI
#define CS_PIN    10  // or SS

MD_MAX72XX mx = MD_MAX72XX(HARDWARE_TYPE, CS_PIN, MAX_DEVICES);

#define  DELAYTIME  100  // in milliseconds
```

```
void scrollText(char *p) {
  uint8_t charWidth;
  uint8_t cBuf[8];  // this should be ok for all built-in fonts
  PRINTS("\nScrolling text");
  mx.clear();
  while (*p != '\0') {
    charWidth = mx.getChar(*p++, sizeof(cBuf) / sizeof(cBuf[0]), cBuf);
    // allow space between characters
    for (uint8_t i=0; i<=charWidth; i++) {
      mx.transform(MD_MAX72XX::TSL);
      if (i < charWidth) {
        mx.setColumn(0, cBuf[i]);
      }
      delay(DELAYTIME);
    }
  }
}

void rows() {
  mx.clear();
  for (uint8_t row=0; row<ROW_SIZE; row++) {
    mx.setRow(row, 0xff); delay(2*DELAYTIME);
    mx.setRow(row, 0x00);
  }
}

void columns() {
  mx.clear();
  for (uint8_t col=0; col<mx.getColumnCount(); col++) {
    mx.setColumn(col, 0xff); delay(DELAYTIME/MAX_DEVICES);
    mx.setColumn(col, 0x00);
  }
}
```

```
void stripe() {
  const uint16_t maxCol = MAX_DEVICES*ROW_SIZE;
  const uint8_t stripeWidth = 10;
  mx.clear();
  for (uint16_t col=0; col<maxCol + ROW_SIZE + stripeWidth; col++) {
    for(uint8_t row=0; row < ROW_SIZE; row++) {
      mx.setPoint(row, col-row, true);
      mx.setPoint(row, col-row - stripeWidth, false);
    }
    delay(DELAYTIME);
  }
}

void spiral() {
  int  rmin = 0, rmax = ROW_SIZE-1;
  int  cmin = 0, cmax = (COL_SIZE*MAX_DEVICES)-1;
  mx.clear();
  while ((rmax > rmin) && (cmax > cmin)) {
    for (int i=cmin; i<=cmax; i++) { // do row
      mx.setPoint(rmin, i, true); delay(DELAYTIME/MAX_DEVICES);
    }
    rmin++;
    for (uint8_t i=rmin; i<=rmax; i++) { // do column
      mx.setPoint(i, cmax, true); delay(DELAYTIME/MAX_DEVICES);
    }
    cmax--;
    for (int i=cmax; i>=cmin; i--) { // do row
      mx.setPoint(rmax, i, true); delay(DELAYTIME/MAX_DEVICES);
    }
    rmax--;
    for (uint8_t i=rmax; i>=rmin; i--) { // do column
      mx.setPoint(i, cmin, true); delay(DELAYTIME/MAX_DEVICES);
    }
    cmin++;
  }
}
```

```
void bounce() {
  const int minC = 0;
  const int maxC = mx.getColumnCount()-1;
  const int minR = 0;
  const int maxR = ROW_SIZE-1;
  int  nCounter = 0;
  int  r = 0, c = 2;
  int8_t dR = 1, dC = 1;  // delta row and column
  mx.clear();
  while (nCounter++ < 200) {
    mx.setPoint(r, c, false);
    r += dR; c += dC;
    mx.setPoint(r, c, true);
    delay(DELAYTIME/2);
    if ((r == minR) || (r == maxR)) {
      dR = -dR;
    }
    if ((c == minC) || (c == maxC)) {
      dC = -dC;
    }
  }
}

void intensity() {
  uint8_t row;
  mx.clear();
  for (int8_t i=0; i<=MAX_INTENSITY; i++) {
    mx.control(MD_MAX72XX::INTENSITY, i);
    mx.setRow(0, 0xff); delay(DELAYTIME*10);
  }
  mx.control(MD_MAX72XX::INTENSITY, 8);
}
```

```
void transformation() {
  uint8_t arrow[COL_SIZE] = {
    0b00001000,
    0b00011100,
    0b00111110,
    0b01111111,
    0b00011100,
    0b00011100,
    0b00111110,
    0b00000000 };
  MD_MAX72XX::transformType_t  t[] = {
    MD_MAX72XX::TSL, MD_MAX72XX::TSL, MD_MAX72XX::TSL, MD_MAX72XX::TSL,
    MD_MAX72XX::TSL, MD_MAX72XX::TSL, MD_MAX72XX::TSL, MD_MAX72XX::TSL,
    MD_MAX72XX::TSL, MD_MAX72XX::TSL, MD_MAX72XX::TSL, MD_MAX72XX::TSL,
    MD_MAX72XX::TSL, MD_MAX72XX::TSL, MD_MAX72XX::TSL, MD_MAX72XX::TSL,
    MD_MAX72XX::TFLR,
    MD_MAX72XX::TSR, MD_MAX72XX::TSR, MD_MAX72XX::TSR, MD_MAX72XX::TSR,
    MD_MAX72XX::TSR, MD_MAX72XX::TSR, MD_MAX72XX::TSR, MD_MAX72XX::TSR,
    MD_MAX72XX::TSR, MD_MAX72XX::TSR, MD_MAX72XX::TSR, MD_MAX72XX::TSR,
    MD_MAX72XX::TSR, MD_MAX72XX::TSR, MD_MAX72XX::TSR, MD_MAX72XX::TSR,
    MD_MAX72XX::TRC,
    MD_MAX72XX::TSD, MD_MAX72XX::TSD, MD_MAX72XX::TSD, MD_MAX72XX::TSD,
    MD_MAX72XX::TSD, MD_MAX72XX::TSD, MD_MAX72XX::TSD, MD_MAX72XX::TSD,
    MD_MAX72XX::TFUD,
    MD_MAX72XX::TSU, MD_MAX72XX::TSU, MD_MAX72XX::TSU, MD_MAX72XX::TSU,
    MD_MAX72XX::TSU, MD_MAX72XX::TSU, MD_MAX72XX::TSU, MD_MAX72XX::TSU,
    MD_MAX72XX::TINV,
    MD_MAX72XX::TRC, MD_MAX72XX::TRC, MD_MAX72XX::TRC, MD_MAX72XX::TRC,
    MD_MAX72XX::TINV };
  mx.clear();
  mx.control(MD_MAX72XX::UPDATE, MD_MAX72XX::OFF);
  for (uint8_t j=0; j<mx.getDeviceCount(); j++) {
    mx.setBuffer(((j+1)*COL_SIZE)-1, COL_SIZE, arrow);
  }
  mx.control(MD_MAX72XX::UPDATE, MD_MAX72XX::ON);
  delay(DELAYTIME);
  mx.control(MD_MAX72XX::WRAPAROUND, MD_MAX72XX::ON);
```

```
// one tab
  for (uint8_t i=0; i<(sizeof(t)/sizeof(t[0])); i++) {
    mx.transform(t[i]); delay(DELAYTIME*4);
  }
  mx.control(MD_MAX72XX::WRAPAROUND, MD_MAX72XX::OFF);
}

void showCharset() {
  mx.clear();
  mx.update(MD_MAX72XX::OFF);
  for (uint16_t i=0; i<256; i++) {
    mx.clear(0);
    mx.setChar(COL_SIZE-1, i);
    if (MAX_DEVICES >= 3) {
      char hex[3];
      sprintf(hex, "%02X", i);
      mx.clear(1); mx.setChar((2*COL_SIZE)-1, hex[1]);
      mx.clear(2); mx.setChar((3*COL_SIZE)-1, hex[0]);
    }
    mx.update();
    delay(DELAYTIME*2);
  }
  mx.update(MD_MAX72XX::ON);
}
void setup() { mx.begin(); }
void loop() {
  scrollText("Graphics");
  rows();
  columns();
  stripe();
  bounce();
  spiral();
  intensity();
  transformation();
  showCharset();
  delay(3000);
}
```

Der Sketch beginnt mit der Einbeziehung von der “*MD_MAX72XX.h*” Library, und fährt mit verschiedenen Definitionen für Makros fort. Ein besonderes Beispiel dafür ist `Serial.print(F(x))`. `F(x)` ist ein Makro, dass die Zeichenfolge “*x*” im Flash-Speicher und nicht im SRAM speichert.. Flash-Speicher ist ein Speicher in Mikrocontrollern, indem Programme gespeichert werden. SRAM-Speicher ist eine Art von RAM-Speicher. Wir verringern die Speichergrenze für unsere Programme, wenn wir das `F(x)` Makro benutzen, was den SRAM erhöht (wichtig für Programme auf Mikrocontrollern). Wenn Sie das `F(x)` Makro nicht verwenden, ist die Wahrscheinlichkeit groß, dass Programme, nicht wie vorhergesehen, funktionieren werden. Das liegt daran, dass im Sketch zu viele Zeichenketten enthalten sind, die in der Programmiersprache C Char-Arrays sind, was den SRAM-Speicher belegt.

Geben Sie dann an, welches Display verwendet wird, indem Sie das Makro `HARDWARE_TYPE` auf den Wert `FC16_HW` (den Ergebniswert aus der `HardwareMapper`-Sketch) setzen.

Danach wird das Makro `MAX_DEVICES` auf 4 festgelegt, was die Anzahl der Segmente darstellt. Wenn mehrere LED-Matrix-Bildschirme in einer Daisy-Chain verbunden sind, ändern Sie diesen Wert entsprechend.

Es gibt drei weitere Makros, die die SPI-Schnittstelle-Pins definieren.

Danach wird das *mx*-Objekt erstellt, das das Bildschirmobjekt darstellt, das in dem gesamten Sketch verwendet wird.

Danach werden viele Funktionen erstellt:

Die erste Funktion heißt *scrollingText()*, mit der Text auf dem Bildschirm gescrolled wird. Die Funktion *scrollingText()* verwendet die Funktion aus der *MD_MAX72XX.h*-Library mit dem Namen *setColumn()*. Die Funktion *set Column()* schaltet eine Spaltenanordnung von LEDs auf dem Bildschirm EIN/AUS. Der Rest der scrollingText()-Funktion ist der Algorithmus für den Scrolling-Text-Effekt. Wir nehmen das Zeichen *"p"* aus dem Flash-Speicher und wandeln die Zeichenfolge in ein Array aus vorzeichenlosen ganzen Zahlen um, das Zeichen der Textzeichenfolge darstellt. Dann wird dieses Array auf dem Bildschirm angezeigt. Am Ende der Funktion wird die Verzögerungszeit eingestellt, die der Geschwindigkeit des Lauftextes entspricht.

Die zweite Funktion heißt *rows()*, die keine Argumente hat und keinen Wert zurückgibt. Die Funktion verwendet eine Funktion *setRow()* aus der Library *MD_MAX72XX*. Die Funktion setRow() hat zwei Argumente. Das erste Argument stellt die Zeilennummer dar, wobei *0* die erste Zeile von oben beginnend bedeutet. Das zweite Argument ist eine ganze Zahl in hexadezimaler Form, die einen *8*-Bit-Wert für ein Array von Pixeln (*8* Pixel in einer Zeile) darstellt. In diesem Fall wird der Wert *0xFF* verwendet, der alle Pixel einer bestimmten Zeile *EIN-schaltet.*

Die dritte Funktion heißt *columns().* Sie ist fast genauso wie die Funktion *rows()*, aber in diesem Fall wird das Spaltenarray von Pixeln *EIN-geschaltet*. Die Funktion verwendet *setColumn()* aus der Library *MD_MAX72XX.h.* *Setcolumn()* und *setRow()*,sind ähnlich, jedoch betrifft die Funktion *setColumn()* die Spalten.

Die vierte Funktion heißt `stripe()`. Sie zeigt einen scrollenden, diagonalen Streifen an (*4* Pixel breit), auf dem ganzen Bildschirm an. Die Funktion verwendet die `setPoint()` Funktion aus der Library "*MD_MAX72XX.h*". Die `setPoint()` Funktion wird verwendet, um bestimmte LED auf dem Screen einzuschalten, und akzeptiert drei Argumente. Die ersten beiden Argumente sind die *X- und Y-Position* der LED (oben links ist *X=0* und *Y=0*, rechts unten auf dem Bildschirm ist *X=31* and *Y=7*). Das dritte Argument ist ein boleescher Wert, der den Zustand der LED darstellt, *AN = true* und *AUS = false*. Am Ende der Funktion geben wir die Verzögerungszeit an, die für die Änderung der Geschwindigkeit des Bildlaufstreifens verwendet wird.

Die fünfte Funktion heißt `spiral()`, die eine schlangenartige Linie anzeigt, die alle Pixel auf dem Bildschirm spiralförmig bewegt. Die Funktion verwendet `setPoint()` aus der "*MD_MAX72XX.h*" Library. Der Rest der Funktion besteht aus dem Algorithmus mit dem man LED für LED einschaltet bis alle LEDs an sind.

Die sechste Funktion heißt `bounce()` und sie zeigt einen sich bewegenden Punkt an, der von den Rändern des Bildschirms abprallt. Die Funktion benutzt neben einigen, der bereits erläuterten Funktionen , `getColumnCont()` die die Anzahl der Spalten auf dem LED-Bildschirm zurückgibt. Der Rest der Funktion besteht aus dem Algorithmus für sich bewegende Pixel.

Die vierte Funktion heißt *`intensity().`* Sie wird verwendet, um nur die erste Reihe der LEDs auf dem Bildschirm *EIN-zuschalten*. Die Funktion verwendet die Funktion *`control()`* aus der Library *MD_MAX72XX.h*. Die Funktion *`control()`* akzeptiert zwei Argumente, das erste definiert welche Eigenschaft des mx-Objekts geändert wird, wo wir den Wert *`MD_MAX72XX::INTENSITY`* speichern. Wir ändern also die Intensität der Helligkeit. Das zweite Argument ist eine Zahl, die den Grad der Intensität repräsentiert und deren Wert im Bereich von *0* bis *15, oder 16* liegt. Der Standardwert ist *8*.

Die achte Funktion heißt *`transformation()`* mit der Bitmap-Bilder auf dem Bildschirm angezeigt werden können. Die Bitmap-Bilddaten werden in dem Array vorzeichenloser Ganzzahlen gespeichert. Dieses Array enthält *8* Elemente, die aus Binärzahlen bestehen. Diese Binärzahlen stellen die Zeilen eines Segments dar, wobei der Wert 0 für eine ausgeschaltete LED und der Wert 1 für eine eingeschaltete LED steht. Wir haben ein weiteres Array mit dem Namen "t" definiert, das Aufzählungen enthält, die für die Animation verwendet werden. Das Format dieser Aufzählungen ist "*MD_MAX72XX::{enumeration}*". Folgende Aufzählungen können verwendet werden:

TSL – Verschiebung nach links* *TSR* – Verschiebung nach rechts*

TSU - Verschiebung nach oben* *TSD* – Verschiebung nach unten*

TFLR – Von links nach rechts spiegeln;

TFUD – Von oben nach unten spiegeln;

TRC – Drehen im Uhrzeigersinn um 90 Grad

TINV – Pixel invertieren (alle, die AN waren, gehen AUS, u. vice versa).

*um ein Pixel

Danach wird die Funktion *setBuffer()* verwendet. Mit ihr senden wir Anzeigedaten an den MAX7219-Treiberchip. Diese Funktion akzeptiert drei Argumente. Das erste und das zweite Argument sind die *X*- und *Y*-Position des Bildes. Das dritte Argument sind die Bitmap-Bilddaten.

Um Daten anzuzeigen, verwenden wir folgende Codezeile:

```
mx.control(MD_MAX72XX::UPDATE, MD_MAX72XX::ON);
```

Dann animieren wir die angezeigten Daten. Dies wird mit der *WRAPAROUND* Eigenschaft der *control()* und *transform()* Funktionen gemacht. Wir verwenden *for* loop to loop hinsichtlich aller Aufzählungen im "*t*" Array.

Die letzte Funktion heißt *showCharsset()* mit der alle UNICODE-Zeichen auf dem LED-Matrix-Bildschirm anzgezeigt werden könnnen. Hier verwenden wir die integrierte Funktion *update()*, die sich wie die *control()* Funktion verhält mit der Eigenschaft *UPDATE*. Die *update()* Funktion akzeptiert ein Argument, dessen Wert einer der folgenden sein kann: *MD_MAX72XX::OFF* and *MD_MAX72XX::ON*
Als Erstes setzten wir den Argumentwert auf *MD_MAX72XX::OFF*, dann ändern wir die Zeichen, die angezeigt werden und ändern dann den Argumentwert zu *MD_MAX72XX::ON*. Dann verwenden wir eine *for* loop to loop bezüglich aller UNICODE-Zeichen.

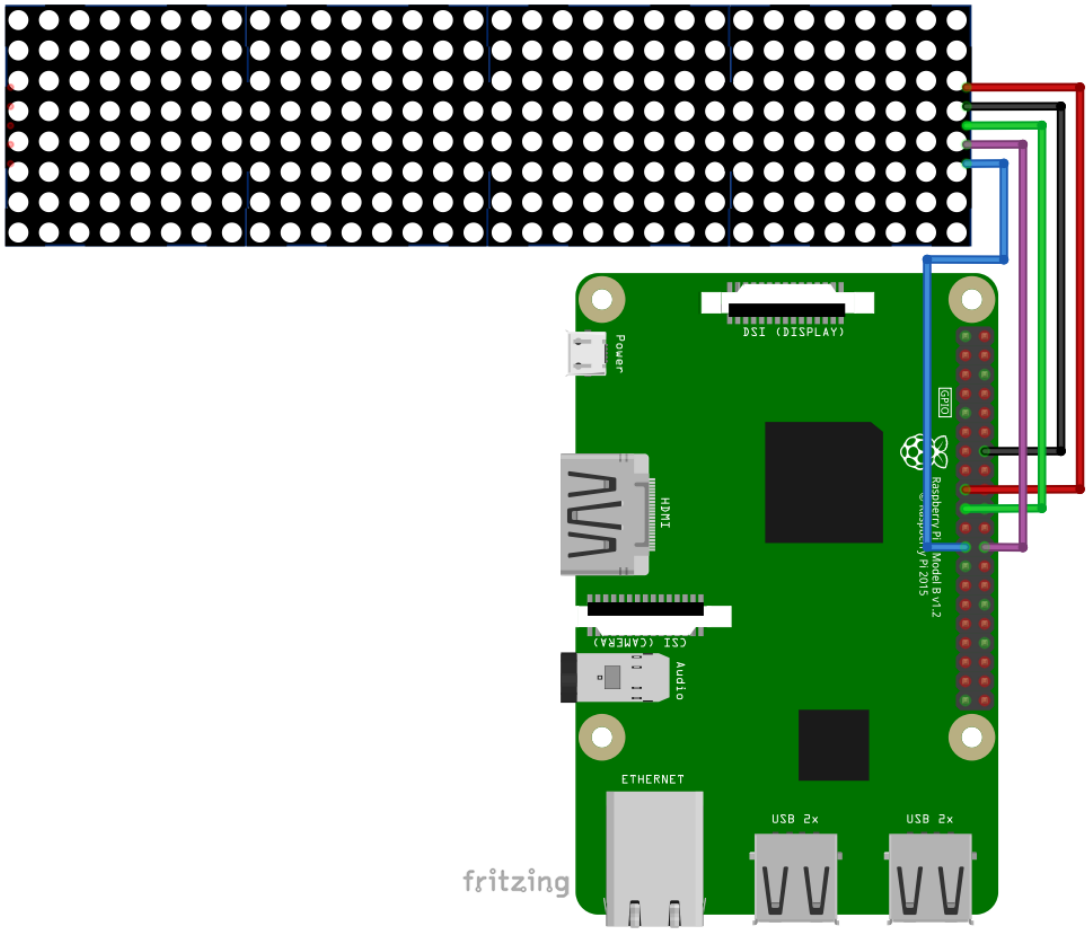
In der *setup()* Funktion initialisieren wir das Bildschirm-Objekt mit der folgenden Codezeile: `mx.begin()`

In der *loop()* Funktion rufen wir nacheinander alle Funktionen auf, die wir erstellt haben.

Wenn Sie mehr über diese Funktionen der "*MD_MAX72XX.h*" library wissen möchten, gehen Sie zur offiziellen Dokumentationsseite der Library:
https://majicdesigns.github.io/MD_MAX72XX/class_m_d___m_a_x72_x_x.html

Verbindung des Displays mit dem Raspberry Pi

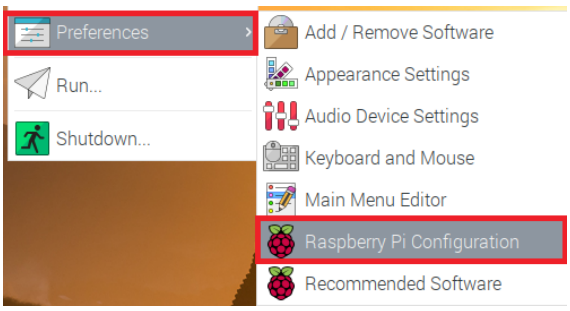
Verbinden Sie das Modul mit dem Raspberry Pi, wie unten abgebildet:



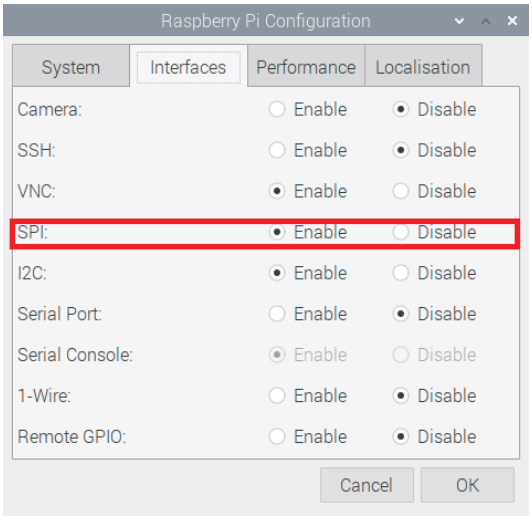
Modul Pin		>	**Raspberry Pi Pin**	
VCC		>	5V	**Roter Draht**
GND		>	GND	**Schwarzer Draht**
DATA	(MOSI)	>	GPIO10 [pin 19]	**Blauer Draht**
CLK	(SCK)	>	GPIO11 [pin 23]	**Brauner Draht**
CS	(SS)	>	GPIO8 [pin 24]	**Grüner Draht**

Aktivieren der SPI-Schnittstelle

Um den Bildschirm zunächst mit einer Raspberry Pi zu verwenden, muss die SPI-Schnittstelle in Raspbian aktiviert werden. Gehen Sie dafür zu:
Preferences > *Raspberry Pi Configuration*
wie unten abgebildet:



Als nächstes öffnen Sie die Registerkarte *`Interfaces`*, stellen Sie die *`SPI`* Radiobuttons auf und aktivieren Sie es, wie auf dem folgenden Bild gezeigt:



Sie werden aufgefordert das System neuzustarten. Starten Sie es neu.

Libraries und Tools für Python

Um das Modul mit dem Raspberry Pi zu verwenden, wird empfohlen, eine externe Library dafür herunterzuladen. Die Library, die in diesem eBook verwendet wird, heißt *luma.led_matrix*. Um die Library zu benutzen, ist es notwendig, die Abhängigkeiten zu installieren. Öffnen Sie dazu das Terminal und führen Sie die folgenden Befehle aus:

```
sudo apt install build-essential python-dev
sudo apt install python-pip libfreetype6-dev
sudo apt install libjpeg-dev libopenjp2-7 libtiff5
```

Wir müssen *"pip"* und *"setuptools"* aktualisieren. Das können Sie mit folgendem Befehl machen:

```
sudo -H pip install --upgrade --ignore-installed pip setuptools
```

Installieren Sie anschließend die neueste Version der *luma.led_matrix*-Library, indem Sie den folgenden Befehl ausführen:

```
sudo -H pip install --upgrade luma.led_matrix
```

Python-Skript:

```
import re
import time
import argparse
from luma.led_matrix.device import max7219
from luma.core.interface.serial import spi, noop
from luma.core.render import canvas
from luma.core.virtual import viewport
from luma.core.legacy import text, show_message
from luma.core.legacy.font import proportional, CP437_FONT, TINY_FONT,
SINCLAIR_FONT, LCD_FONT

def demo(n, block_orientation, rotate, inreverse):

    serial = spi(port=0, device=0, gpio=noop())
    device = max7219(serial, cascaded=n or 1,
                     block_orientation=block_orientation, rotate=rotate or 0,
                     blocks_arranged_in_reverse_order=inreverse)
    print('Created device')
    msg = 'MAX7219 LED Matrix Demo'
    print(msg)
    show_message(device, msg, fill='white',font=proportional(CP437_FONT),
                                                scroll_delay=0)
    time.sleep(1)
    print('Vertical scrolling')
    words = ['Victor', 'Echo', 'Romeo', 'Tango', 'India', 'Charlie',
              'Alpha', 'Lima', ' ', 'Sierra', 'Charlie', 'Romeo', 'Oscar',
              'Lima', 'Lima', 'India', 'November', 'Golf', ' ']

    virtual = viewport(device, width=device.width, height=len(words) * 8)
    with canvas(virtual) as draw:
        for i, word in enumerate(words):
           text(draw, (0, i * 8), word,     fill='white',
                                            font=proportional(CP437_FONT))
```

```
# one tab
    for i in range(virtual.height - device.height):
        virtual.set_position((0, i))
        time.sleep(0.05)

    msg = 'Brightness'
    print(msg)
    show_message(device, msg, fill='white')
    time.sleep(1)

    for _ in range(5):
        for intensity in range(16):
            device.contrast(intensity * 16)
            time.sleep(1)

    device.contrast(0x80)
    time.sleep(1)
    msg = 'Alternative font!'
    print(msg)
    show_message(device, msg, fill='white', font=SINCLAIR_FONT)
    time.sleep(1)
    msg = 'Proportional font - characters are squeezed together!'
    print(msg)
    show_message(device,msg,fill='white',font=proportional(SINCLAIR_FONT))
    time.sleep(1)
    msg = 'Tiny is, I believe, the smallest possible font \
    (in pixel size). It stands at a lofty four pixels \
    tall (five if you count descenders), yet it still \
    contains all the printable ASCII characters.'
    msg = re.sub(' +', ' ', msg)
    print(msg)
    show_message(device, msg, fill='white', font=proportional(TINY_FONT))
    time.sleep(1)
```

```
# one tab
    msg = 'CP437 Characters'
    print(msg)
    show_message(device, msg)
    time.sleep(1)
    for x in range(256):
        with canvas(device) as draw:
            text(draw, (0, 0), chr(x), fill='white')
            time.sleep(0.3)

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='matrix_demo arguments',
                    formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('--cascaded', '-n', type=int, default=1,
                    help='Number of cascaded MAX7219 LED matrices')
    parser.add_argument('--block-orientation', type=int, default=0,
                    choices=[0, 90, -90],
                    help='Corrects block orientation when wiRoter
vertically')
    parser.add_argument('--rotate',type=int,default=0,choices=[0, 1, 2, 3],
                    help='Rotate display 0=0°, 1=90°, 2=180°, 3=270°')
    parser.add_argument('--reverse-order', type=bool, default=False,
                    help='Set to true if blocks are in reverse order')
    args = parser.parse_args()
    try:
        while True:
        demo(args.cascaded,args.block_orientation,
                                            args.rotate,args.reverse_order)
        time.sleep(1)

    except KeyboardInterrupt:
        print('Script end!')
```

Der Code basiert auf:

https://github.com/rm-hull/luma.led_matrix/blob/master/examples/matrix_demo.py

Das Skript beginnt mit dem Import notwendiger Libraries. Als Erstes erstellen wir die Funktion *demo()* mit der alle Funktionen der "*luma.led_matrix*" Library ausgeführt werden können. Die Funktion akzeptiert vier Argumente und gibt keinen Wert zurück. Die Argumente sind:

» *n* - was die Anzahl der kaskadierten LED-Matrix-Segmente darstellt

» *block_orientation* - für die Korrektur der Ausrichtung von Inhalten auf dem Bildschirm ; deren Werte sind: *0*, *90* und *-90*

» *rotate* – dreht den Inhalt des Bildschirms; Werte sind: *0*, *1*, *2* und *3*, stellt jeweils die Winkel *0°*, *90°*, *180°* und *270°* dar

» *reverse* – ist ein boolescher Wert und wenn auf *true* gesetzt wird die Segementfolge umgekehrt.

Innerhalb der Funktion werden ein SPI-Kommunikationsobjekt und ein Geräteobjekt erstellt.

Um eine scrollende Nachricht (horizontal) auf dem Screen anzuzeigen, verwenden wir die *show_message()* Funktion der "*luma.led_matrix*" Library. Diese Funktion akzeptiert fünf Argumente und gibt keinen Wert zurück. Die letzten drei Argumente sind optional. Das erste Argument ist das Geräteobjekt, das zweite die Nachricht, das dritte das *fill* Argument, das vierte das *font* Argument und das fünfte das *scroll_delay* Argument. Der Wert des *fill* Arguments ist die Farbe: "*white*", "*blue*", "*red*", usw. Wenn Sie eine andere Farbe als "*black*" für die Nachricht verwenden wird sie in Rot angezeigt, da die Farbe Rot die Farbe der LEDs auf dem Bildschirm sind.

Wenn Sie “*black*” im fill Argument verwenden, wird nichts angezeigt.

Mit dem *font* Argument wird die Schriftart festgelegt und mit dem *scroll_delay* Argument die Geschwindigkeit des scrollenden Texts.

Um vertikal scrollende Nachrichten anzuzeigen, müssen Sie die *viewport* Library-Funktion verwenden. Diese Library-Funktionen werden verwendet, um ein Objekt oder eine Leinwand mit Nachrichteninhalt zu erstellen, dann wird der Inhalt dieser Leinwand über `for` loop durch den Bildschirm läuft, wodurch ein vertikaler Scrolling-Effekt erzielt wird.

Um die Helligkeit zu ändern, verwenden wir die *contrast()* Funktion, wie in der folgenden Codezeile: `device.contrast(number)`
wobei *number* die Helligkeitsstufe ist. Es gibt *16* Helligkeitsstufenwerte, aber der Wert muss, wenn durch *16* geteilt eine ganze Zahl ergeben, da die internen Register des MAX7219-Treiberchips auf diese Weise aufgebaut sind.

Man kann die Schriftarten auf zwei Weisen verwenden. Einmal kann man den Schriftartnamen als Argument in der *show_message()* Funktion setzen. Zweitens kann man in der *proportional()* Funktion dasselbe Argument setzen. Diese Funktion akzeptiert ein Argument, nämlich den Schriftartnamen. Der Unterschied besteht darin, dass mit der *proportional()* Funktion die Schriftart proportional zur Bildschirmgröße anzeigt wird.

Um ein bestimmtes Zeichen auf einem Segment des Bildschirms anzuzeigen, verwenden wir die *`text()`* Funktion. Die Funktion akzeptiert vier Argumente und gibt keinen Wert zurück. Das Erste ist ein draw-Objekt. Dieses Objekt enthält einen Teil der Leinwand oder einen Teil der Daten des Leinwandobjekts. Wir haben das Canvas-Objekt zum vertikalen Scrollen von Nachrichten bereits behandelt. Das zweite Argument ist eine Liste mit zwei Elementen, der X- und Y-Position der linken oberen Ecke des Zeichens. Das dritte Argument ist das Zeichen, das angezeigt werden soll. Das vierte Argument ist das *`fill`* Argument.

Speichern Sie das Skript unter dem Namen "*ledMatrix.py*". Um es auszuführen, öffnen Sie es im selben Verzeichnis, in dem Sie das Skript gespeichert haben und führen Sie folgenden Befehl aus:

```
python3 ledMatrix.py
```

Es gibt die Möglichkeit, vier Argumente an das Skript zu senden, wenn wir es ausführen. Diese Argumente sind optional. Dies ist mit der Hilfe der Library argparse möglich. Es gibt vier optionale Argumente, die Sie an das Skript senden können:

» *`cascaded`* – Anzahl der kaskadierten LED-Matrix-Segmente

» *`block-orientation`* – korrigiert die Ausrichtung der Segmente

» *`rotate`* – dreht den Inhalt des Bildschirms

» *`reverse-order`* – ändert die Bildlaufrichtung des Inhalts

Diese Argumente werden verwendet, wenn wir die *`demo()`* Funktion aufrufen. Die *`demo()`* Funktion wird im infinity loop block (*`while True:`*) aufgerufen.

Der Infinity loop Block ist im *try-except* Block. Wir verwenden den *try-except* Block, um auf den Keyboard Interrupt zu warten. Der Keyboard Interrupt findet statt, wenn wir *STRG + C* auf der Tastatur drücken. Dies beendet das Skript.

Ein Beispiel, in dem alle Argumente verwendet werden, wenn wir das Skript ausführen.

```
python3 ledMatrix.py --cascaded 4 --block-orientation 90
                     --rotate 2   --reverse-order false
```

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 nach weiteren hochwertigen Produkten für Arduino und Raspberry Pi suchen, sind Sie bei AZ-Delivery Vertriebs GmbH goldrichtig. Wir bieten Ihnen zahlreiche Anwendungsbeispiele, ausführliche Installationsanleitungen, E-Books, Bibliotheken und natürlich die Unterstützung unserer technischen Experten.

https://az-delivery.de

Viel Spaß!

Impressum

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