Erstellen von großen oder eigenen Zeichensätzen für die Anwendung in MicroPython

Die Software

Verwendete Software:

Fürs Flashen und die Programmierung des ESP32:
Thonny oder
µPyCraft
micropython-font-to-py
Paket font2py.rar mit Batch-Konverter
MicroPython-Modul charset.py

Anzeige in großen Ziffern am OLED-Display.

Der Vorteil eines OLED-Displays ist eindeutig durch die flexible Darstellung von Text und Grafik gegeben. Wir nutzen das aus, um die Anzeige nicht mit den popligen 10x8-Pixel-Zeichen, sondern mit ordentlich lesbaren Ziffern umzusetzen. Für diesen Zweck werden wir einen Windows-Zeichensatz in einen OLED-Zeichensatz portieren und daraus die benötigten Zeichen in ein Modul extrahieren. Beides passiert auf der Basis von CPython und MicroPython. CPython wurde zusammen mit Thonny installiert, denn Thonny ist in Python geschrieben und braucht die CPython-Umgebung, damit es überhaupt funktioniert. Natürlich können wir in Thonny (oder einem anderen Editor) auch CPython-Programme schreiben und dann von der Powershell aus starten. Genau das werden wir tun, sobald unser Zeichensatz umgewandelt ist.

Windows-TTF-Zeichensatz clonen

Um einen Vektorzeichensatz von Windows ins Pixelformat zu übertragen, benutzen wir eine Freeware von Peter Hinch, die der MIT-Licence unterliegt. Sie heißt micropython-font-to-py und wird als ZIP-Datei von Git-Hub heruntergeladen. Für die Installation habe ich das Verzeichnis _fonts im Rootverzeichnis meiner Festplatte F: angelegt. Das und alles Weitere funktioniert übrigens auch zusammen mit einem USB-Stick.

Die Datei **micropython-font-to-py-master.zip** habe ich in **F:_fonts** gespeichert und auch dorthin entpackt. Das entstandene Verzeichnis habe ich in **font2py** umgetauft, ich mag kürzere Pfadnamen. Wechseln wir jetzt in dieses Verzeichnis. Die beiden Pythonprogramme **font_to_py.py** und **font_test.py** werden wir in Kürze benutzen. Zuvor legen wir noch ein Verzeichnis **quellen** an.

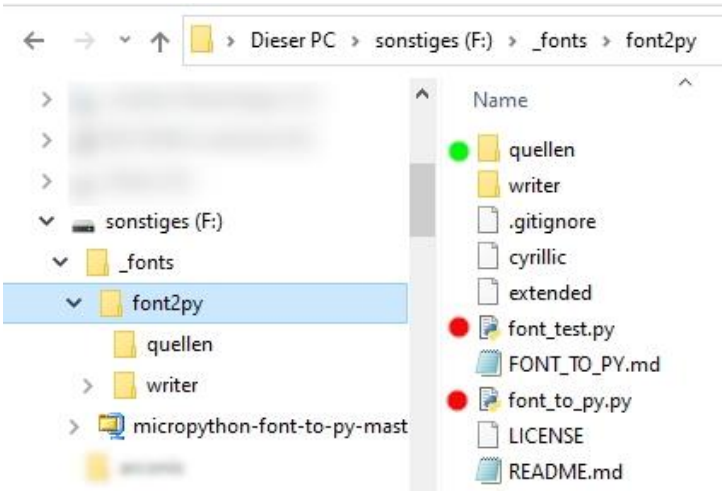

Abbildung 1: Charset01

Öffnen Sie nun den Schriftenordner von Windows, **C:\Windows\Fonts**. Suchen Sie nach einer möglichst klaren Schriftform und kopieren Sie die Datei in das Verzeichnis quellen. Ich habe hier **impact.ttf** gewählt.

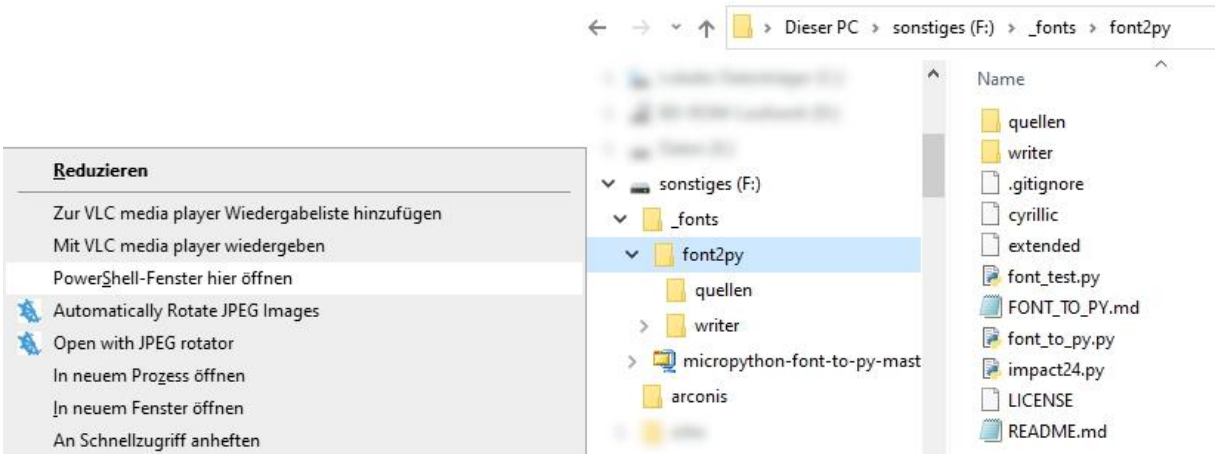

Abbildung 2: Charset02

Die nächsten beiden Schritte erfolgen in einem Powershell-Fenster. Wir öffnen das Contextmenü mit gedrückter **Umschalttaste** (Shift) und einem **Rechtsklick** auf den Ordner **font2py** und wählen **Powershell-Fenster hier öffnen**. Ein **dir** zeigt uns, dass wir im richtigen Verzeichnis sind. Dann setzen wir folgendes Kommando ab:

.\font_to_py.py F:_fonts\font2py\quellen\impact.ttf 24 -f impact24.py

Bitte beachten Sie, dass der Befehl mit einem ".\" eingeleitet werden muss, damit er in der aktuellen Pfadumgebung ausgeführt wird.

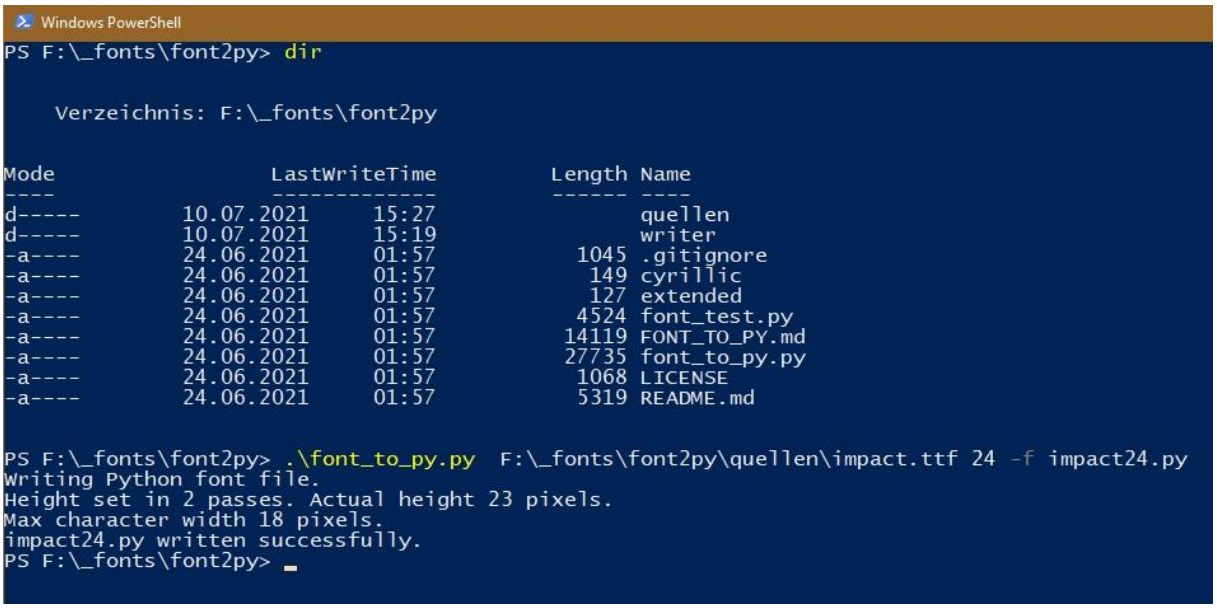

Abbildung 3: Charset03

Soeben haben wir den Zeichensatz Impact mit 24 Pixel Zeichenhöhe (inclusive Unterlängen) in eine Python-Datei umgewandelt. Der Schalter **-f** sorgt dafür, dass zunächst alle Zeichen auf gleiche Breite gebracht werden, auch die von Proportionalzeichensätzen. Das ist wichtig, denn sonst gibt es beim Zerteilen der Strings im nächsten Schritt Zeichensalat.

Wir können auf dem ESP32 Speicherplatz sparen, wenn wir vom gesamten Zeichensatz nur die Zeichen verfügbar machen, die wir wirklich brauchen. Diese Zeichen geben wir zwischen Anführungszeichen an und leiten die Ausgabe in die Datei **impact24.txt** um.

.\font_test.py impact24 "0123456789,-AVW" > impact24.txt

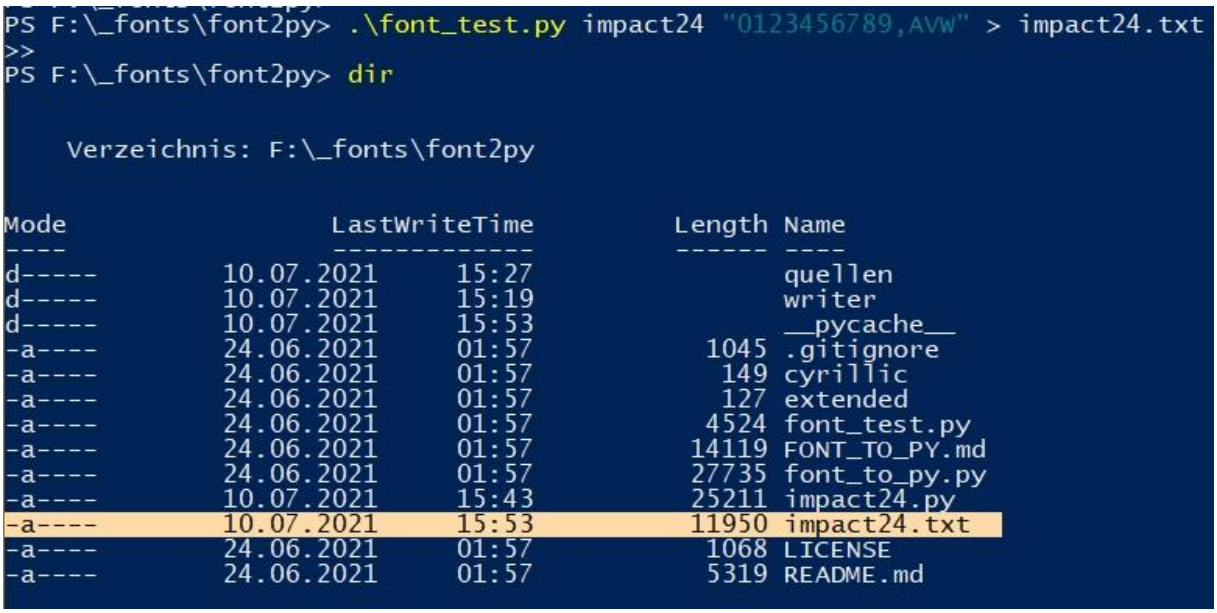

Abbildung 4: Charset04

Der Inhalt dieser Datei sieht etwa so aus.

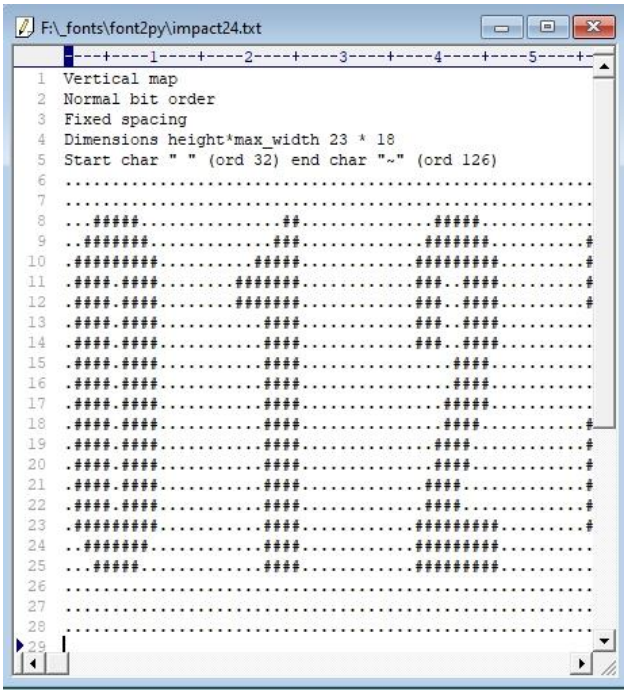

Abbildung 5: Charset-Text

Auf dieser Grundlage erzeugen wir nun abschließend ein Modul, das wir in Programme integrieren können, welche OLED-Displays nutzen.

Starten wir Thonny und wechseln wir in das Arbeitsverzeichnis **F:_fonts\font2py**.

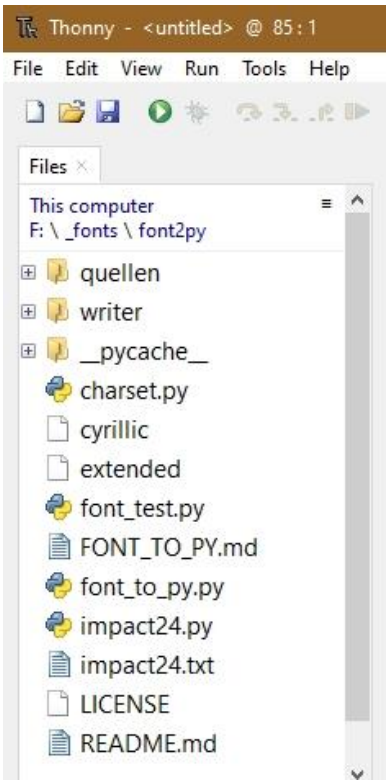

Abbildung 6: Charset_finishing

Wir erzeugen ein neues Programmfenster via **File – New**, kopieren den Text des Programmlistings von text2py.py dort hin und speichern die Datei unter diesem Namen ab, nachdem die Variablen pfad, datei und chars angepasst sind. chars wird derselbe Stringinhalt zugewiesen, wie er beim Aufruf von font_test.py verwendet wurde.

Hinweis:
Bitte beachten Sie, dass der "\" in der Pfadangabe doppelt angegeben sein muss. Dadurch wird die spezielle Bedeutung des "\" in Zeichenketten aufgehoben (Escapezeichen).

```
# text2py.py

import sys

pfad="F:\\_fonts\\font2py\\"
datei="impact24"
chars="0123456789,-AVW"

def readln():
    s=""
    while 1:
        c=f.read(1)
        a=ord(c)
        if a>10 and a<126:
            s+=c
        elif a==10:
            for i in range(3):
                z=f.read(1)
            break
    return s



f=open(pfad+datei+".txt", "r")

# Kopfdaten einlesen und Zeichenhöhe und -breite holen
for j in range(3):
    zeile = readln()
    print(zeile)
zeile=readln()
pos1=zeile.find("width ")
pos2=zeile.find("*",pos1)
height=int(zeile[pos1+5:pos2])
width=int(zeile[pos2+1:])
print(height,width)
zeile=readln()

# jetzt die Zeicheninfo einlesen
a=[]
for z in range(height):
    a.append(readln())
```

```
f.close()

print("****************************************")

f=open("charset.py","w")
print('chars="'+chars+'"')
print('height='+str(height))
print('width='+str(width))
print('number=[')
f.write('chars="'+chars+'"\n')
f.write('height='+str(height)+'\n')
f.write('width='+str(width)+'\n')
f.write('number=[\n')


for i in range(len(chars)):
    last=0
    block=[]
    b=len(a[1][i*width:(i+1)*width])
    for z in range(height):
        row=a[z][i*width:(i+1)*width]
        new=row.rfind("#")
        if new>last:
            last=new
    last=(last+2 if last < b else last)
    for z in range(height):
        row=a[z][i*width:(i+1)*width]
        row=row[:last]
        block.append(row)
    print('    ('+str(last)+',  #  Zeichen: '+chars[i])
    f.write('    ('+str(last)+',  #  Zeichen: '+chars[i]+'\n')
    for z in block:
        z=z.replace(".","0")
        z=z.replace("#","1")
        print('     0b'+z+',')
        f.write('     0b'+z+',\n')
    print('    ),')
    f.write('    ),\n')
print(']')
f.write(']\n')

f.close()
```

Das Programm starten wir nicht in Thonny, sondern auf der Powershell.

.\text2py.py

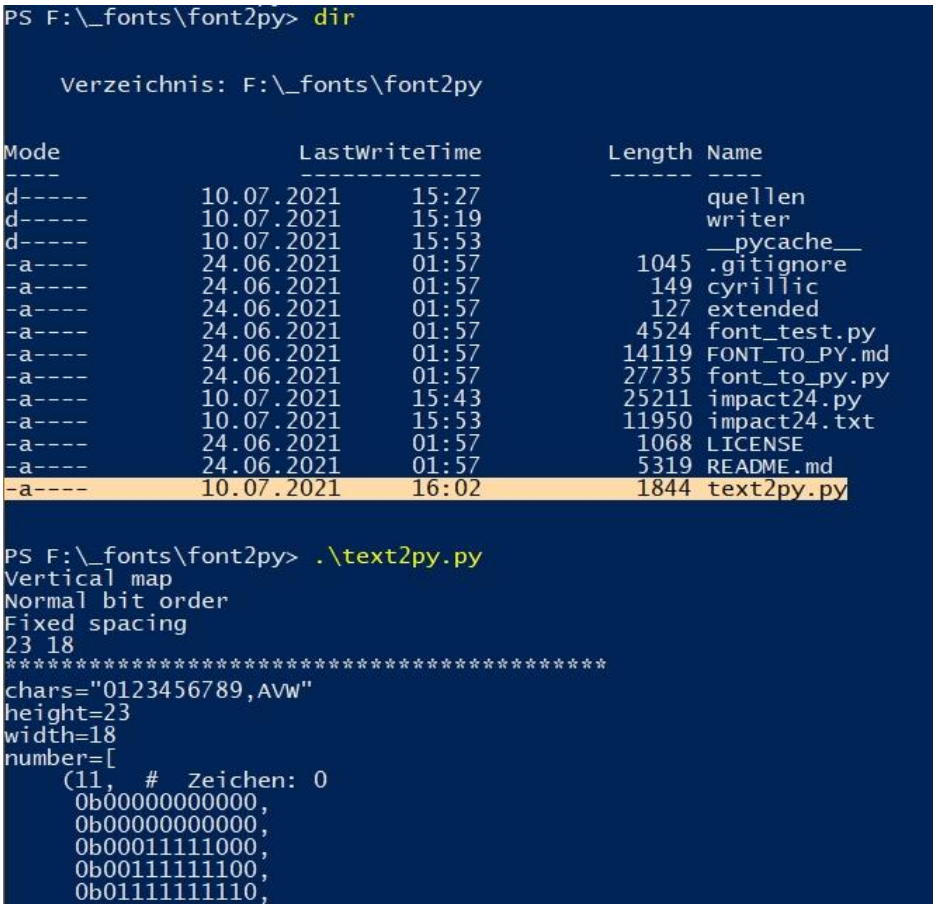

Abbildung 7: Charset05

Das Programm läuft auf dem PC, nicht im ESP8266/ESP32. Es schneidet für jedes Zeichen von jeder Zeile der Zeichendefinition ein Stück der festgesetzten Zeichenbreite ab und sucht die Position des letzten "#". Zum Maximum dieser Positionen wird 1 addiert. Dann werden alle Zeilenstrings auf diese Länge verkürzt. So entsteht die ursprüngliche Gewichtung der Zeichen für ein ausgewogenes proportionales Schriftbild.

Jeder "." der Zeichendefinition wird nun als Nächstes in eine 0 und jedes # in eine 1 umgewandelt. Durch Voranstellen von "0b" entstehen im Listing aus dem Muster Ganzzahlen in Binärschreibweise. Pro Zeichen werden sie in einem Tupel zusammengefasst. Der erste Wert in jedem Tupel ist die Zeichenbreite. Die Tupel aller Zeichen werden in der Liste **number** gesammelt.

Das Ergebnis wird als Datei mit dem Namen **charset.py** abgespeichert, sie ist das angestrebte Modul, das jetzt in beliebige MicroPython-Programme importiert werden kann.

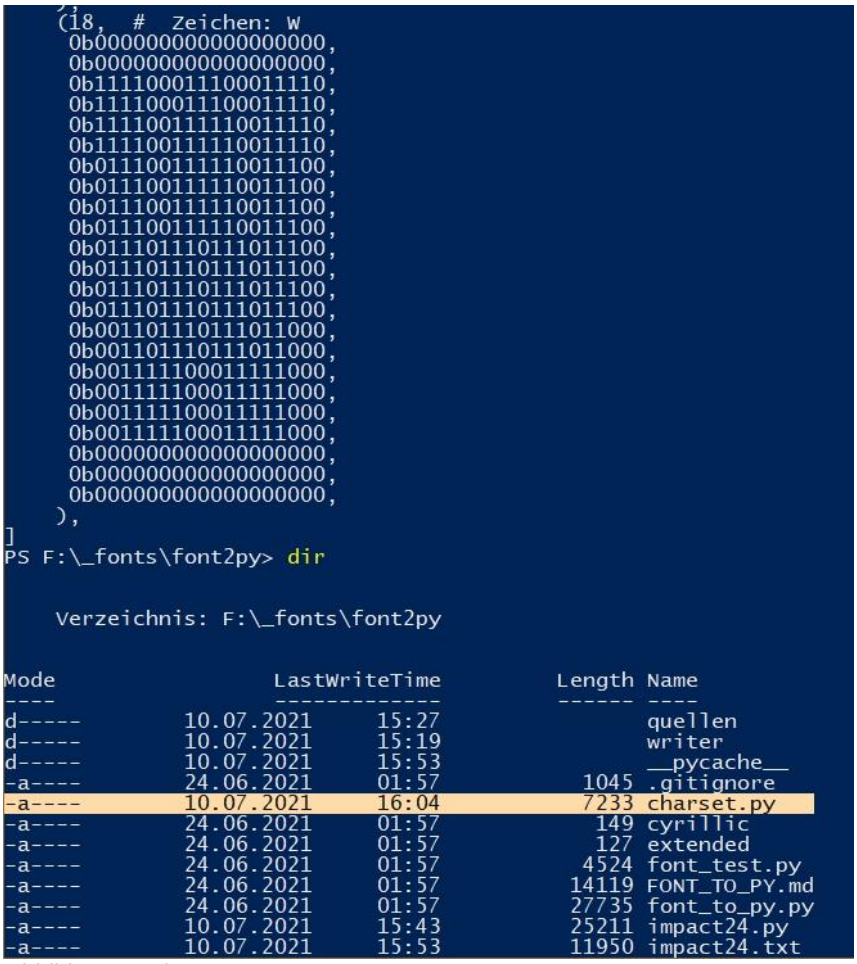

Abbildung 8: Charset06

Das folgende Programm zeigt die Verwendung. **charset.py** wird in den Workspace des Projekts kopiert, in den Flash des ESP8266 geladen und hier als Modul im Programm **current**.py importiert. Wenn Sie bisher selbst noch keinen Zeichensatz erstellt haben, können Sie auch erst einmal meinen 30-er herunterladen. Die Module oled.py und ssd1306.py können auch heruntergeladen werden.

```
# current.py
from machine import Pin, I2C, ADC, Timer
from time import sleep, ticks_ms,sleep_ms
from oled import OLED
import charset as cs

# Pintranslator fuer ESP8266-Boards
# LUA-Pins     D0 D1 D2 D3 D4 D5 D6 D7 D8
# ESP8266 Pins 16  5  4  0  2 14 12 13 15
#                 SC SD

SCL=Pin(21)
SDA=Pin(22)
i2c=I2C(-1,SCL,SDA)
d=OLED(i2c,128,32)
d.clearAll()
```

```
def putDigit(n,xpos,ypos,show=True):
    breite=cs.number[n][0]
    for row in range(1,cs.height):
        for col in range(breite-1,-1,-1):
            c=cs.number[n][row] & 1<<col
            d.setPixel(xpos+breite+3-col,ypos+row,c,False)
    if show:
        d.show()
    return xpos+breite+2

def putValue(wert,unit,xpos,ypos,show=True):
    d.clearAll(False)
    ws=str(wert).upper()
    pos=0
    ws=ws.replace(".",",")
    for z in ws:
        try:
            n=cs.chars.index(z)
            pos=putDigit(n,pos,0,False)
        except:
            print("Zeichen nicht vorhanden:",z)
    try:
        n=cs.chars.index(unit)
        pos=putDigit(n,pos,0,True)
    except:
        print("unerlaubte Einheit:",unit)
    return pos

    putValue(3.48,"A",0,0)
    sleep(3)
    putValue(12.05,"V",0,0)
    sleep(3)
    putValue(52.37,"W",0,0)
    sleep(3)
    d.clearAll()
```

Das Programm benötigt außer dem Modul charset.py noch oled.py und ssd1306.py. Alle wesentlichen Aktionen zur Stromerfassung und Darstellung mit dem vergrößerten Zeichensatz werden durch Funktionen erledigt. Die Darstellung erfolgt durch das Setzen derjenigen Pixel, die einer 1 in der Binärdarstellung einer Pixelzeile entsprechen. Das macht die Funktion putDigit(). Sie nimmt die Ziffer und die Koordinaten für die linke obere Ecke der Darstellung und gibt den x-Wert für die nächste Ausgabeposition zurück. Die Funktion putValue() nimmt einen Messwert und ein Einheitenzeichen. Der Wert, Integer oder Float, wird geparst. Die Funktion wandelt dabei evtl den Dezimalpunkt in ein Komma um, stellt die Digits, falls möglich dar und gibt ebenfalls die nächste freie x-Position zurück.

Beiden Funktionen ist der optionale Parameter show gemeinsam. Der Defaultwert True führt zur sofortigen Anzeige. False führt nur zu Änderungen des Framebufferinhalts, der erst mit d.show() zur Anzeige gebracht wird

Batch-Konvertierung und die Klasse CharSet

Für ganz Eilige gibt es außerdem die Batchdatei makecharset.bat. In ihr sind alle Schritte ab der Ablage der TTF-Datei im Ordner quellen zusammengefasst.

makecharset.bat

```
@rem Testskript zur Parameterübergabe
@rem Verwendung:
rem USAGE: .\makecharset.bat fontname size """chars""" "path_to_ttf"
rem
============================================================
rem
@echo %1 %2 %3 %4
@.\font_to_py %4%1.ttf %2 -f %1_%2_.py
@.\font_test %1_%2_ %3 > %1_%2.txt
@ rem type %1_%2.txt
@ .\text2py_bat %1 %2 %3
```

Die Datei wird nach folgendem Muster aufgerufen.

.\makecharset Schriftenname_ohne_TTF Zeichenhöhe_in_Pixel """Zeichen""" "Pfad_zum_Ordner_quellen"

Beispiel:
.\makecharset impact 18 "0123456789_-*CAVW" " F:_fonts\font2py\quellen\"

Danach steht die Datei impact_18.py im Verzeichnis **F:_fonts\font2py** und kann in das Arbeitsverzeichnis von Thonny übertragen werden.

Die Funktionen zum Darstellen der Zeichen aus dem Programm current.py wurden zur Standardisierung in einem Modul zusammengefasst. Das machte eine Änderung der bisherigen Benennung der Zieldatei des Zeichensatzes nötig, was in dem Batchfile bereits berücksichtigt ist. Das Modul **charset.py** enthält jetzt die **Klasse CharSet** und importiert den jeweiligen Zeichensatz, der mittels **makecharset**.bat erstellt wurde.

Die API der Klasse ist hier beschrieben.