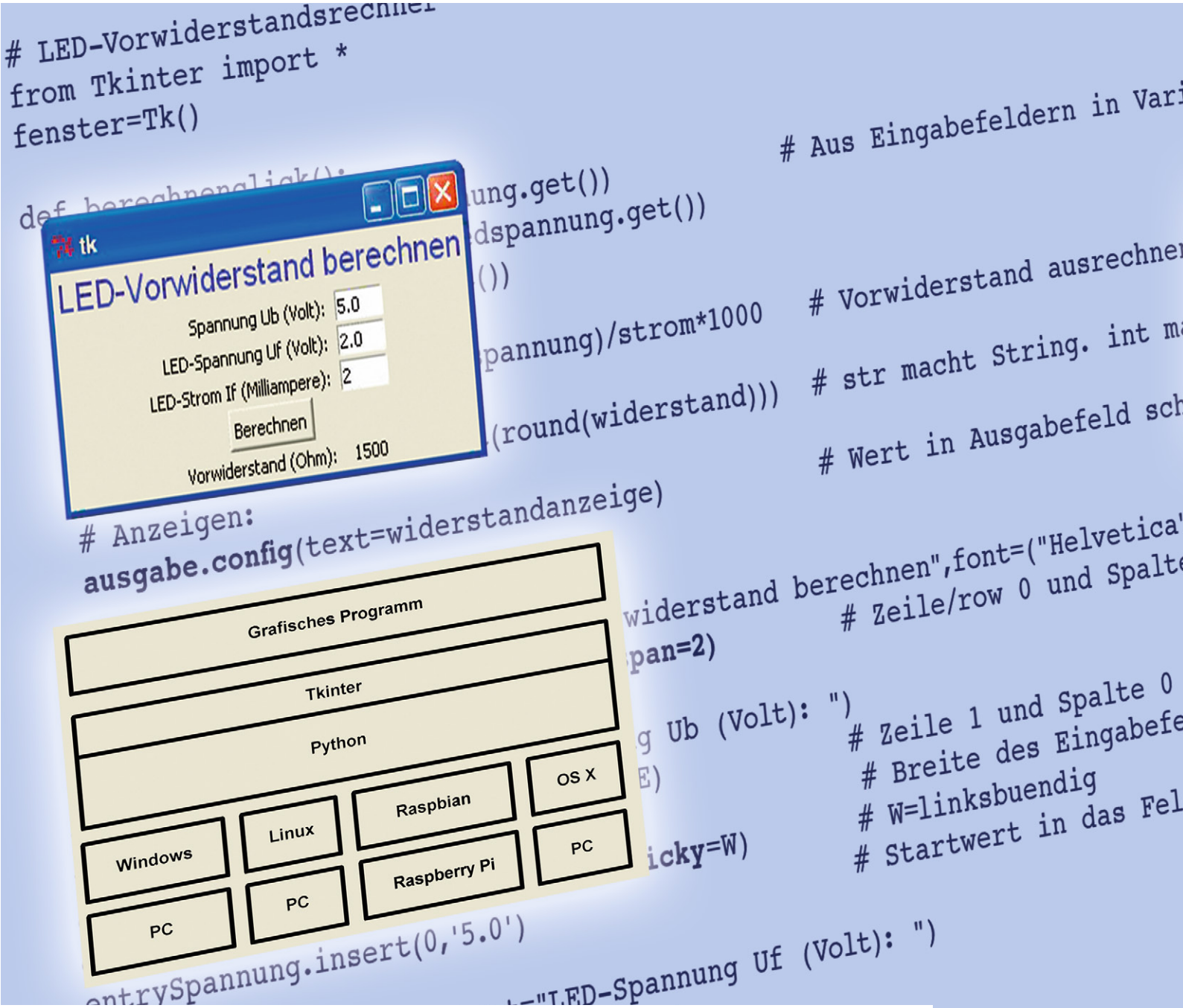
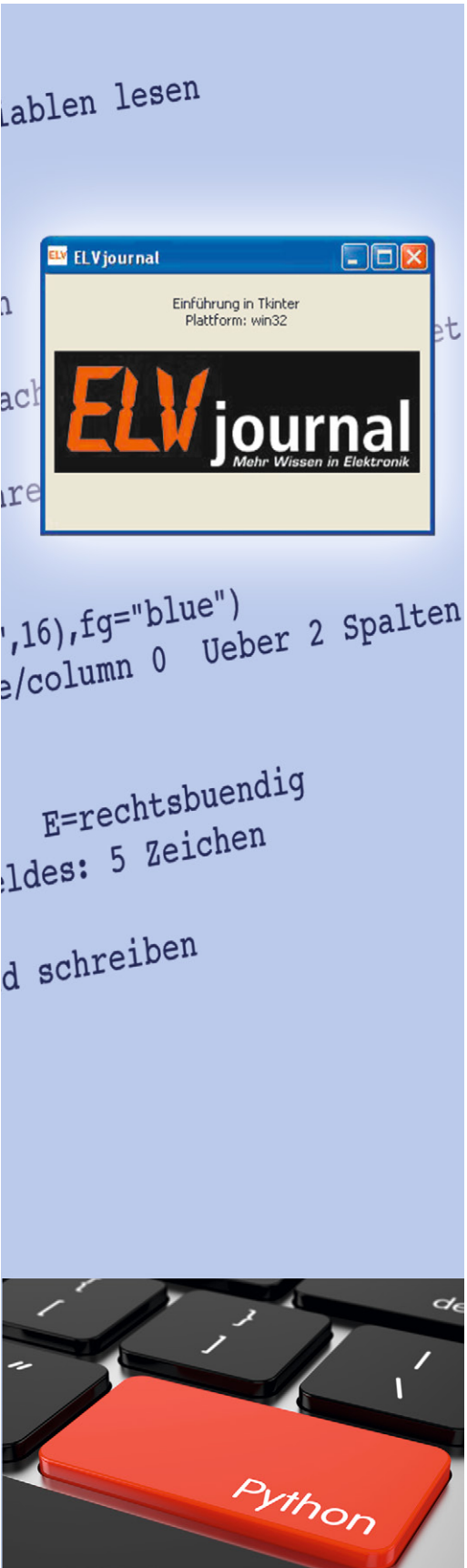
Tkinter

Grafische Anwendungen erstellen für Windows, OS X, Linux oder Raspbian

Teil 2



Mit der kostenlosen Programmiersprache Python und dem ebenso kostenlosen Python-Modul Tkinter lassen sich schnell und einfach grafische Anwendungen für Windows, Linux, Raspbian oder OS X erstellen. Nachdem im ersten Teil dieses Artikels die Grundlagen von Tkinter dargestellt wurden, geht es im vorliegenden Teil um den Zugriff zu Dateien bzw. der seriellen Schnittstelle. Die Steuerung eines seriell angesteuerten Gerätes oder das Darstellen von Daten von anderen Systemen kann mit professionellen grafischen Anwendungsoberflächen individuell gelöst werden.




```
Wert1=23.5
Wert2=19.8
Wert3=21.3
999
```

Bild 14: testdatei.txt

Bild 15: Screenshot „Datei lesen“

Dateizugriff (Daten lesen)

Ein kleines Programm zum Auslesen von Werten aus einer Datei wie in Bild 14 mit einer grafischen Oberfläche wie in Bild 15 soll den Dateizugriff illustrieren. In Bild 15 sieht man links den Zustand nach dem Starten des Programms. Es werden zunächst nur zwei Striche als Wert angezeigt. Mit jedem Klicken auf den Pushbutton wird eine neue Zeile aus der Datei „testdatei.txt“ eingelesen und im Fenster angezeigt, bis die Enderkennung *999* erreicht wird, was durch Anzeige des Textes „Ende“ signalisiert wird (siehe Bild 15). Das Tkinter-Programm ist in Bild 16 zu sehen.

```
#Wert aus Konfigurationsdatei lesen
from Tkinter import *

fenster=Tk()

#------- Routine zum Einlesen einer neuen Zeile und Anzeigen des Zeilenwertes
def neuezeileclick():
  zeile = info.readline()          #Eine Zeile einlesen
  zeile = zeile.strip()            #letztes Zeichen abschneiden
  wertausgabe.config(text=zeile)   # Label updaten
  # Testen auf Dateiende:
  zeile = zeile[0:3]               # Die ersten drei Zeichen abschneiden  Index 0, 1 und 2 !
  if(zeile == "999"):
    info.close()
    wertausgabe.config(text="Ende") # Label updaten
    buttonNeuezeile.config(state=DISABLED)


#------ Statischer Text-------------------------------------
textfeld=Label(fenster,text="Wert aus Datei")
textfeld.pack()

#------ Datei zum Lesen oeffnen ----------------------------
info=open("testdatei.txt","r")      # Datei zum Lesen oeffnen


#------ Defaultwert als Label anzeigen ------------------
wertausgabe=Label(fenster,text="---")
wertausgabe.pack()


#--- Pushbutton definieren und anzeigen -------------------------
buttonNeuezeile=Button(fenster,text="Neue Zeile",command=neuezeileclick)
buttonNeuezeile.pack()


fenster.mainloop()
```

Bild 16: Programm „Datei lesen“

```
Wert= 1234.56
```

Bild 17: testdatei2.txt

Bild 18: Screenshots „Datei lesen" und „Daten schreiben"

```
Wert= 99.7
```

Bild 19: testdatei2.txt nach dem Schreiben

Erläuterungen:
Nach dem Anzeigen des Überschriften-Labels wird die Datei „testdatei.txt" mit *open()* für Lesezugriff geöffnet und dem Handler *info* zugewiesen. In der Prozedur *neuezeileclick* wird mit *readline()* eine neue Zeile aus der Datei gelesen und der Variablen *zeile* zugewiesen. Mit *config()* wird der Inhalt der Variablen *zeile* in das Label geschrieben und damit angezeigt. Wenn die ersten drei Zeichen der Zeile *999* sind, wird die Datei geschlossen, ein entsprechender Text im Label ausgegeben und der Pushbutton disabled (siehe Bild 15 rechts).

Dateizugriff (Daten lesen und schreiben)

Oft möchte man beim Programmstart Standardwerte aus einer Konfigurationsdatei lesen und verwenden. Im Lauf des Programms werden die Konfigurationswerte evtl. für zukünftige Verwendung geändert und müssen dafür wieder in die Konfigurationsdatei geschrieben werden. Bild 17 zeigt den Inhalt einer Konfigurationsdatei mit dem Namen „testdatei2.txt". Beim Programmstart soll der Wert aus der Datei eingelesen und angezeigt werden (Bild 18 oben). Bei Anklicken des Pushbuttons wird ein neuer Wert (hier *99.7*, siehe Bild 18 unten) in die Konfigurationsdatei geschrieben, die danach aussieht wie in Bild 19. Das Tkinter-Program zeigt Bild 20.

Erläuterungen:
Beim Öffnen des Programms wird in Block 4 die Datei „testdatei2.txt" zum Lesen geöffnet, eine Zeile mit *readline()* in die Variable *zeile* gelesen und die Datei wieder geschlossen. Die hinteren Zeichen der eingelesenen Zeile werden in Block 5 im Label *wertausgabe* angezeigt. Wenn der Pushbutton geklickt wird, wird in der Prozedur in Block 2 die Datei zum Schreiben geöffnet, der Wert aus dem Eingabefenster mit *write()* in die Datei geschrieben und die Datei wieder geschlossen.

```
#Wert aus Konfigurationsdatei lesen und schreiben
# 0 ------------------------------------------------------------------------
from Tkinter import *

# 1 ------------------------------------------------------------------------
fenster=Tk()

# 2 ------- Routine zum Schreiben eines neuen Wertes in die Datei
def neuenwertclick():
  ausgabe=open("testdatei2.txt","w")             # Zum Schreiben oeffnen
  ausgabe.write("Wert= "+ entrywert.get())       # Schreiben in Datei:  "Wert=  und Wert
  ausgabe.close()                                # Datei wieder schliessen
  wertausgabe.config(text=entrywert.get())       # Aus EINgabefeld in AUSgabefeld uebertragen

# 3 ------ Statischer Text-------------------------------------
textfeld=Label(fenster,text="Wert aus Datei")
textfeld.pack()

# 4 ------ Datei zum Lesen oeffnen und Startwert einlesen----------------------------
info=open("testdatei2.txt","r")                  # Datei zum Lesen oeffnen
zeile = info.readline()                          # Eine Zeile einlesen
info.close()                                     # Datei wieder schliessen

# 5 ------ Defaultwert aus Datei als Label anzeigen -------------------
wertausgabe=Label(fenster,text=zeile[6:])        # Die hinteren Zeichen ausschneiden, damit Wert= nicht mit eingelesen wird.
wertausgabe.pack()


textfeld=Label(fenster,text="Eingabewert: ")     # Ein Ausgabefeld definieren
textfeld.pack()
entrywert=Entry(fenster,bg='white')              # Ein EINgabefeld definieren
entrywert.pack()
entrywert.insert(0,zeile[6:])                    #Defaultwert

# 6 --- Pushbutton definieren und anzeigen -------------------------
buttonNeuerwert=Button(fenster,text="Neuer Wert",command=neuenwertclick)  # Pushbotton definieren
buttonNeuerwert.pack()


# 7 ------------------------------------------------------------------------
fenster.mainloop()
```

Bild 20: Tkinter-Programm „aus Datei lesen" und „in Datei schreiben"

Serielle Datenübertragung vom/zum PC

Oft steht man vor der Aufgabe, Daten über die serielle Schnittstelle ausgeben oder einlesen zu müssen. Mit Tkinter lässt sich eine grafische Oberfläche für eine Maschinen- oder Robotersteuerung über die serielle Schnittstelle oder auch ein Programm zur Anzeige von seriell empfangenen Werten am PC erstellen. Sehr nützlich ist dabei ein USB-TTL-Adapterkabel (Best.-Nr. CG-11 55 33).

Zunächst muss zur Benutzung der seriellen Schnittstelle durch Python das Modul „Pyserial" installiert sein/werden.

1. Prüfen:
   Ein Eingabefenster/Terminalfenster öffnen und Python mit *python* starten. Hinter dem Python-Prompt (>>>) *import serial* eingeben. Wenn KEINE Fehlermeldung erscheint, dann können die nächsten Schritte übersprungen werden. Ansonsten:
2. Download:
   Von [4] die Datei *tar.gz* herunterladen.
3. Entpacken:
   Unter Windows: Mit 7-Zip [5] in einen Ordner entpacken.
   Unter Linux oder OS X: In einem Terminalfenster Downloadverzeichnis *tar –xzf pyserial-3.0.1.tar.gz* eingeben.
4. Installieren:
   python setup.py install bzw. unter Linux
   sudo python setup.py install eingeben.
5. Prüfen:
   In einem Python-Fenster *import serial* eingeben. Es sollte keine Fehlermeldung erscheinen.

Danach sollte man den USB-zu-TTL-Adapter an USB am PC einstecken und Windows die Treiber installieren lassen.

Im Geräte-Manager wird nachgesehen, welcher Port (COMx) dem Adapter zugewiesen wurde (Bild 21).

Für eine Maschinen- oder Robotersteuerung wird eine Bedienoberfläche wie in Bild 22 mit einem Programm wie in Bild 23 erstellt. Durch Klicken auf einen Pushbutton soll jeweils ein serielles Kommando an die Maschine bzw. den Roboter gesendet werden.

Erläuterungen:
Im Block 4 (siehe Bild 23) wird die serielle Schnittstelle geöffnet und ein Teststring seriell geschrieben. Mit *grid()* werden in Block 6 die Überschrift und die Pushbuttons platziert. Wenn ein Pushbutton geklickt wird, dann wird entsprechend Block 5 ein dem Pushbutton entsprechendes Kommando seriell geschrieben. Der „Start/Stopp"-Pushbutton wird im Togglemodus verwendet. Die Variable *maschinenstatus* enthält den jeweiligen Betriebsstatus und wird jeweils umgeschaltet. In Block 2 sieht man eine Besonderheit: Manchmal möchte man, wenn der Benutzer eines Programms „Fenster schließen" anklickt (rechts oben bei Windows), noch „letzte Aktionen" wie beispielsweise Schreiben von Konfigurationswerten in eine Datei oder Ähnliches ausführen lassen. In Block 2 sieht man, wie man derartige Aktionen in ein Tkinter-Programm einbauen kann.

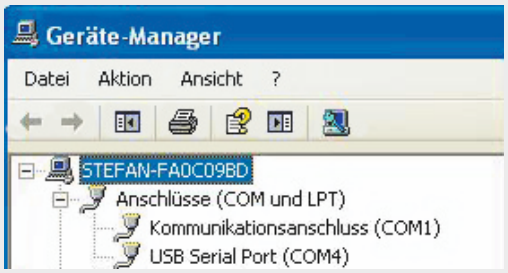


Bild 21: USB-TTL-Adapter im Geräte-Manager

Bild 22: Screenshot „seriell senden"

Auf gleiche Weise wie in diesem Beispiel kann man sich auch sehr schnell eine individuelle Bedienoberfläche für die Ansteuerung des ELV FS20-UART-Senders FS20 US (Best.-Nr. CG-09 87 89) erstellen und dadurch vom PC aus Verbraucher schalten. Es werden lediglich ein Programm ähnlich wie das in Bild 23, ein UART-TTL-Adapter (z. B. Best.-Nr. CG-11 55 33) und zwei Leitungen (Trx und GND) zum FS20-Sender benötigt.

Seriell empfangen

Als letztes Beispiel soll eine Idee gegeben werden, wie Daten seriell – zum Beispiel von einem Sensor, einem Messgerät oder einem GPS-Empfängermodul – empfangen und in einem Anwendungsfenster angezeigt werden können. Die Daten sollen wie in Bild 24 über die serielle Schnittstelle empfangen werden. Die Daten kommen zeilenweise an und jede Zeile beginnt mit einem Dollarzeichen und einem Buchstaben A, B oder C als Kennzeichen, gefolgt von dem eigentlichen Wert. Im Beispiel in Bild 24 werden also *$A523, $B9.0, $Co.k.* usw. von einem Sensor o. Ä. empfangen und gemäß Bild 25 grafisch dargestellt.

Erläuterungen:
In diesem Programm (Bild 26), das ein paar neue Konstrukte benutzt *(queue, thread, try)* werden in Block 0 alle benötigten Module importiert. Außer dem bekannten Tkinter-Modul für die grafische Oberfläche und dem Modul „serial" für serielle Verbindungen wird

hier ein Modul „Threading“, das es ermöglicht, Programmcode parallel laufen zu lassen, sowie ein Modul „Queue“, mit dessen Methoden man Datenelemente in eine Warteschlange schreiben und später wieder aus der Warteschlange lesen kann, importiert. Hier wird nun kein Modul mehr mit der * Methode importiert, Deshalb wird jede Methode mit dem selbst vergebenen Namen „tk“ bzw. dem Modulnamen angesprochen (vgl. „Elektronikwissen“). In Block 1 werden das Tkinter-Fenster und die drei Ausgabelabel definiert.

```
# Seriell senden
# 0 ------------------------------------------------------------------------------
from Tkinter import *
import serial                       # Modul fuer serielle Verbindungen importieren

# 1 ------------------------------------------------------------------------------
fenster=Tk()
fenster.resizable(width=FALSE, height=FALSE)            # Fenstergroesse nicht veraenderbar

# 2 ----Seriellen Port beim Fensterschliessen auch schliessen----------------------
def callback():
  UART.close()                                          # Seriellen Port schliessen
  print("Tschuess")                                     # Letzte Aktionen ausfuehren
  fenster.destroy()                                     # Fenster schliessen
  fenster.protocol("WM_DELETE_WINDOW", callback)
# 3 ------------------------------------------------------------------------------
defaultbg = fenster.cget('bg')                          # Standardhintergrundfarbe fuer spaeter speichern
maschinenstatus = 0                                     # Startwert: Maschine ist gestoppt

# 4 ------------------------------------------------------------------------------
# Seriellen Port oeffnen
#UART = serial.Serial("/dev/ttyAMA0", 9600)             # fuer Raspberry
#bzw.
UART = serial.Serial('COM4', 9600)                      # fuer Windows
UART.write("Hallo serielle Welt")

# 5 ------------------------------------------------------------------------------
# Prozeduren definieren
def hochclick():
  print("hoch")                                         # Kontrollausgabe im Kommandofenster/Terminal
  UART.write("up")                                      # Serielle Ausgabe

def linksclick():
  print("links")
  UART.write("left")

def startstoppclick():
  global maschinenstatus                                # Globale Variable zugreifbar machen
  print("Start/Stopp gedrueckt")
  if maschinenstatus == 0:
    print ("Maschine wird gestartet\n")
    maschinenstatus = 1
    buttonStartstopp.config(text='Stopp',bg='red')      # Buttontext aendern und Farbe rot
    UART.write("start")
  else:
    print ("Maschine wird gestoppt\n")
    maschinenstatus = 0
    buttonStartstopp.config(text='Start',bg=defaultbg)  # Button auf Standardhintergrundfarbe setzen
    UART.write("stop")

def rechtsclick():
  print("rechts")
  UART.write("right")

def runterclick():
  print("runter")
  UART.write("down")
# 6 ------------------------------------------------------------------------------
# Fensterueberschrift
textfeld=Label(fenster,text="Robotersteuerung",font=("Helvetica",16),fg="blue")
textfeld.grid(row=0,column=0,columnspan=3)              # Zeile/row 0 und Spalte/column 0  Ueber 3 Spalten

# Pushbuttons
buttonHoch = Button(fenster,text='hoch', command=hochclick)
buttonHoch.grid(row=1,column=1)

buttonLinks = Button(fenster,text='links', command=linksclick)
buttonLinks.grid(row=2,column=0, sticky=N+E+S+W)

buttonStartstopp = Button(fenster,text='Start', command=startstoppclick)
buttonStartstopp.grid(row=2,column=1, sticky=N+E+S+W)

buttonRechts = Button(fenster,text='rechts', command=rechtsclick)
buttonRechts.grid(row=2,column=2,sticky=N+E+S+W)

buttonRunter = Button(fenster,text='runter', command=runterclick)
buttonRunter.grid(row=3,column=1, sticky=N+E+S+W)

# 7 -----Endlosschleife ----------------------------------------------------------
fenster.mainloop()
```

Bild 23: Tkinter-Programm „seriell senden“

Nach der Definition der Prozeduren bzw. Threads in Block 2 wird in Block 3 zunächst eine Queue angelegt. Eine Queue (= Warteschlange) ist ein Puffer zur Datenspeicherung, der geeignet ist, Daten in einer anderen Geschwindigkeit aufzunehmen, als sie ausgelesen werden. Jeder kennt die Druckerwarteschlange, in die mehrere Dokumente eingestellt werden können, die dann in der Reihenfolge ihres Eingangs gedruckt werden. In diesem Beispiel kommen die Daten über die serielle Schnittstelle am Computer an und werden zunächst in diesen Puffer gelesen. Von anderer Stelle her kann man dann die Datenelemente in derselben Reihenfolge, wie sie angekommen sind, zeitlich unabhängig „abgreifen". Dadurch sind die Prozesse für das Schreiben in die Queue und das Auslesen aus der Queue zeitlich entkoppelt. Als Nächstes wird in Block 3 ein „Thread" erstellt. Das ist ein unabhängig vom sonstigen Programm ablaufender Programmteil. Der Thread heißt *seriell_lesen* und hat nach dem Öffnen der seriellen Schnittstelle nichts anderes zu tun, als Daten von der seriellen Schnittstelle in die Queue einzulesen. Dadurch, dass das Einlesen der Daten in einem unabhängigen Thread geschieht, werden andere Programmteile nicht blockiert, wenn im Thread bis zum nächsten Eintreffen von Daten gewartet werden muss.

In der Prozedur *aus_queue_lesen* in Block 2 wird alle 20 Millisekunden versucht, ein Datenelement aus der Queue zu lesen. Wenn etwas aus der Queue gelesen wurde, dann wird geprüft, ob die Zeichenkette mit *$A*, *$B* oder *$C* beginnt. Wenn das der Fall ist, dann wird der Rest der

```
$A523
$B9.0
$Co.k.
...
```

Bild 24: Serieller Datenstrom von einer Sensoreinheit

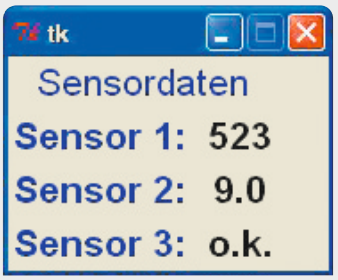


Bild 25: Screenshot „seriell empfangen"

```
#!/usr/bin/python
# Daten im Format $A523 $B9.0 $Co.k. ueber die serielle Schnittstelle empfangen
# 0 ------------------------------------------------------------------------------
import Tkinter as tk
import serial
import threading
import Queue
# 1 ------------------------------------------------------------------------------
fenster=tk.Tk()
Ausgabelabel1 = tk.Label(fenster,text="--", fg="black", font="Helvetica 16 bold")
Ausgabelabel2 = tk.Label(fenster,text="--", fg="black", font="Helvetica 16 bold")
Ausgabelabel3 = tk.Label(fenster,text="--", fg="black", font="Helvetica 16 bold")
# 2 ------------------------------------------------------------------------------
def seriell_lesen(queue):
    meinUART = serial.Serial('COM4', 9600 )                    # Seriellen Port definieren und oeffnen
    while True:                                                # Endlos ..
        daten = meinUART.readline().strip()                    # .. Zeile seriell lesen
        queue.put(daten)                                       # .. und Element in die Queue schreiben

def gui_erstellen():
    # Ueberschrift
    tk.Label(fenster,text="Sensordaten",font=("Helvetica",16),fg="blue")
                                                               .grid(row=0,column=0,columnspan=2)
    # Feste Texte
    tk.Label(fenster,text="Sensor 1: ", fg="blue", font="Helvetica 16 bold").grid(row=1,column=0)
    tk.Label(fenster,text="Sensor 2: ", fg="blue", font="Helvetica 16 bold").grid(row=2,column=0)
    tk.Label(fenster,text="Sensor 3: ", fg="blue", font="Helvetica 16 bold").grid(row=3,column=0)
    # Ausgabelabel
    Ausgabelabel1.grid(row=1,column=1)
    Ausgabelabel2.grid(row=2,column=1)
    Ausgabelabel3.grid(row=3,column=1)
    return fenster

def aus_queue_lesen(fenster, queue):
    try:
        daten = queue.get_nowait()                             # Element aus der Queue holen
        if daten[0:2] == "$A":                                 # Wenn $A ...
            Ausgabelabel1.config(text=daten[2:])               # ... dann Ausgabelabel updaten
        if daten[0:2] == "$B":                                 # usw.
            Ausgabelabel2.config(text=daten[2:])
        if daten[0:2] == "$C":
            Ausgabelabel3.config(text=daten[2:])
    except Queue.Empty:                                        # Wenn die Queue leer ist ...
         pass                                                  # ... nichts tun
    fenster.after(20, aus_queue_lesen, fenster, queue) # Nach 20 ms wieder
# 3 ------------------------------------------------------------------------------
queue = Queue.Queue()                                          # Eine Queue erstellen
thread = threading.Thread(target=seriell_lesen, args=(queue,))   # Thread starten
thread.daemon = True
thread.start()
gui_erstellen()                                                # Gui erstellen
fenster.after(20, aus_queue_lesen, fenster, queue)             # Aus Queue lesen
fenster.mainloop()
```

Bild 26: Programm „seriell empfangen"

Bild 27: Screenshot (links Windows, Mitte Raspbian, rechts OS X)

Zeichenkette (ohne *$A*, *$B* bzw. *$C*) in das jeweilige Label geschrieben. Wenn der Versuch (englisch: try) misslingt, ein Datenelement aus der Queue zu lesen, weil die Queue leer ist, springt der Programmablauf zu dem Programmcode hinter *except Queue.Empty*, wo in diesem Fall nichts getan wird (pass). Durch dieses Try-Except-Konstrukt wird verhindert, dass es bei einer Fehlermeldung wegen des Versuches, aus einer leeren Queue zu lesen, zu einem Programmabbruch kommt.

Die von einem FS20- und Wetterdatenempfänger (Best.-Nr. CG-10 38 66) oder einem TTL-GPS-Empfängermodul generierten seriellen Daten können auf ähnliche Weise „eingesammelt" und am PC-Bildschirm dargestellt oder weiterverarbeitet werden.

Icon, Titel und Bild

Es gibt verschiedene Möglichkeiten, die Darstellung von Umlauten, Bildern oder Icons im Programm zu programmieren. Als kleine Anregung wird hier eine

```
# Hallo Welt mit Tkinter fuer Raspberry
from Tkinter import *                  # Bei Python 3: tkinter
import sys                             # Modul für Systemzugriffs-Methoden importieren

meinfenster = Tk()

meinfenster.title("ELVjournal")        # Titeltext
meinfenster.geometry("300x200")        # Groesse des Fensters

# Icon fuer Fenster:
img = PhotoImage(file='ELV_2015.gif')
meinfenster.tk.call('wm','iconphoto', meinfenster._w, img)

# Betriebssystemplattform in einem Label anzeigen
meinlabel=Label(meinfenster, text=u'\n Einf\xfchrung in Tkinter \nPlattform: '+ sys.platform +'\n', fg='#0A116B')
meinlabel.pack()

# gif-Bild einbinden:
foto = PhotoImage(file="ELVjournal.gif")
Label(meinfenster,image=foto).pack()    # Hier Platzieren gleich bei der Erstellung

meinfenster.mainloop()
```

Bild 28: Programm mit Grafikeinbindung, Umlauten, Icon

Elektronikwissen

Import von Modulen in Python

Python ist eine sehr mächtige Programmiersprache, die von sich aus bereits viele Möglichkeiten mitbringt und die sich durch Einbinden sogenannter Module noch erweitern lässt. Module müssen durch den Import-Befehl für das Python-Programm nutzbar gemacht werden, dann können die Methoden der Module im Programm verwendet werden.

Es gibt drei Möglichkeiten für den Import (hier des Tkinter-Moduls):
1. *from Tkinter import ** # Importiert alle Tkinter-Methoden. Verpönt, weil zu viel importiert wird und es zu Kollisionen kommen kann, aber praktisch, weil man sich Schreibarbeit im Programm erspart, denn Methoden können direkt aufgerufen werden wie zum Beispiel *Tk()* oder *Label()* oder *TkVersion* (um die Tkinter-Version zu sehen).
2. *import Tkinter* # Es wird nicht gleich alles importiert, aber man muss im Programm jeder Methode den Modulnamen Tkinter voranstellen, also beispielsweise *Tkinter.Tk()* oder *Tkinter.Label()*, um die Methode *Tk* bzw. *Label* des Moduls Tkinter zu benutzen.
3. *import Tkinter as grafik* # Stellt sozusagen einen Kompromiss dar. Es wird nicht gleich alles importiert, aber man kann den Namen für die Benutzung im Programm selbst wählen. Hier beispielsweise *grafik.Tk()* oder *grafik.Label()* oder *grafik.TkVersion*. Diese dritte Methode wird allgemein empfohlen, und oft benutzt man *import Tkinter as tk* und spricht die Methoden dann mit *tk.methode()* an. Beispiel: *tk.Label()*.

mögliche Lösung gezeigt, die auf allen Plattformen funktioniert (Bild 27).

Erläuterungen:
Die Verwendung des deutschen Umlauts wurde durch Voranstellen des Buchstabens *u* (für Unicode) und Verwendung des entsprechenden Codes für den Buchstaben in der Zeichenkette ermöglicht. Umlaute und Sonderzeichen werden gemäß folgender Tabelle verwendet:
ä: \xe4
ö: \xf6
ü: \xfc
Ä: \xc4
Ö: \xd6
Ü: \xdc
ß: \xdf
€ (Eurozeichen): \u20ac
Das *u* vor dem String beachten!

Es gibt auch andere Möglichkeiten (zum Beispiel als erste Zeile: *# -*- coding: utf-8 -*-*), um Umlaute zu ermöglichen, ohne kryptische Codes in der Zeichenkette benutzen zu müssen.

Durch *import sys* und Zugriff auf die Variable *sys.platform* wird in diesem Beispiel die Betriebssystemplattform ermittelt, auf der das Programm läuft. Hier wird die Betriebssystemplattform durch ein Label zur Anzeige gebracht (siehe Bild 27) (Darwin ist die Linux-Basis von OS X). Man kann auch in Abhängigkeit der Plattform unterschiedliche Aktionen im Programm ausführen lassen. Durch *fg='#0A116B'* wird die Vordergrundfarbe des Labels festgelegt. Generell lassen sich Farben entweder – wie bisher – durch festgelegte Farbkonstanten wie *black, red, green, blue, yellow* usw. festlegen oder man verwendet – wie hier – die hexadezimale Darstellung der Farbanteile von Rot, Grün und Blau (*'#rrggbb'*). *#FFFFFF* entspricht Weiß, *#0000FF* entspricht Blau, *#FF0000* entspricht Rot usw. In diesem Programm wurde ein Label nicht zur Darstellung von Text verwendet, sondern zur Darstellung einer Bilddatei. Hier wurde beim Platzieren des Labels in derselben Zeile *.pack()* angehängt. Das kann man machen, wenn man keinen Objekthandler für späteren Zugriff benötigt. Bei statischen Texten könnte immer diese einzeilige Kurzform verwendet werden.

Zusammenfassung

Während in diesem Artikel ausschließlich der Weg gewählt wurde, ein Python/Tkinter-Programm in einem Texteditor zu schreiben und in der Kommandozeile/im Terminalfenster mit *python dateiname.py* zu starten, kann man auch die bei der Installation von Python automatisch mitinstallierte Entwicklungsumgebung IDLE verwenden (bei Windows und Raspbian im Startmenü, bei OS X über Eingabe von *idle* im Terminal), wo man Syntax-Highlighting, integrierte Hilfe usw. hat. Es ist auch möglich, mit entsprechenden zusätzlichen Tools aus dem Internet Python/Tkinter-Programme zur Verteilung an andere Benutzer zu paketieren [6].

Mit Python und Tkinter lassen sich schnell und ohne lange Einarbeitungszeit ansprechende Programme mit grafischer Oberfläche erstellen.

In diesem Artikel konnten die Möglichkeiten von Python und die Möglichkeiten von Tkinter nur gestreift und ein grober Eindruck gegeben werden. Python bietet sehr viele Möglichkeiten objektorientierter Programmierung, die hier überhaupt nicht dargestellt wurden, und Tkinter besitzt noch weit mehr Widgets und Methoden, als hier gezeigt werden konnten.

Mit Internet-Tutorials oder einem Buch an der Seite kann man sofort mit Tkinter beginnen und die faszinierende Welt der GUI-Erstellung betreten. ELV

Weitere Infos:

[4] Pyserial –Download: https://pypi.python.org/pypi/pyserial
[5] 7-Zip: www.7-zip.org/
[6] www.py2exe.org/ (py2exe für Windows)
https://pythonhosted.org/py2app/ (py2app für OS X)
www.pyinstaller.org/ (PyInstaller für Windows, Linux, OS X)

Empfohlene Produkte/Bauteile:	Best.-Nr.	Preis
Raspberry Pi 2 B, Starter-Set	CG-11 93 80	€ 89,95
Raspberry Zero inkl. Noobs auf microSD-Karte	CG-12 26 19	€ 22,95
Mini-USB-Modul UM2102	CG-09 18 59	€ 5,95
UART-Adapterkabel USB auf TTL, 5 V	CG-11 55 33	€ 24,95
FS20-UART-Sender FS20 US	CG-09 87 89	€ 19,95
FS20- und Wetterdaten-UART-Empfänger FS20 WUE	CG-10 38 66	€ 14,95
Buch „Raspberry Pi programmieren mit Python“	CG-12 16 55	€ 29,99

Infos zu den Produkten/Bauteilen finden Sie im Web-Shop. Preisstellung Februar 2016 – aktuelle Preise im Web-Shop.