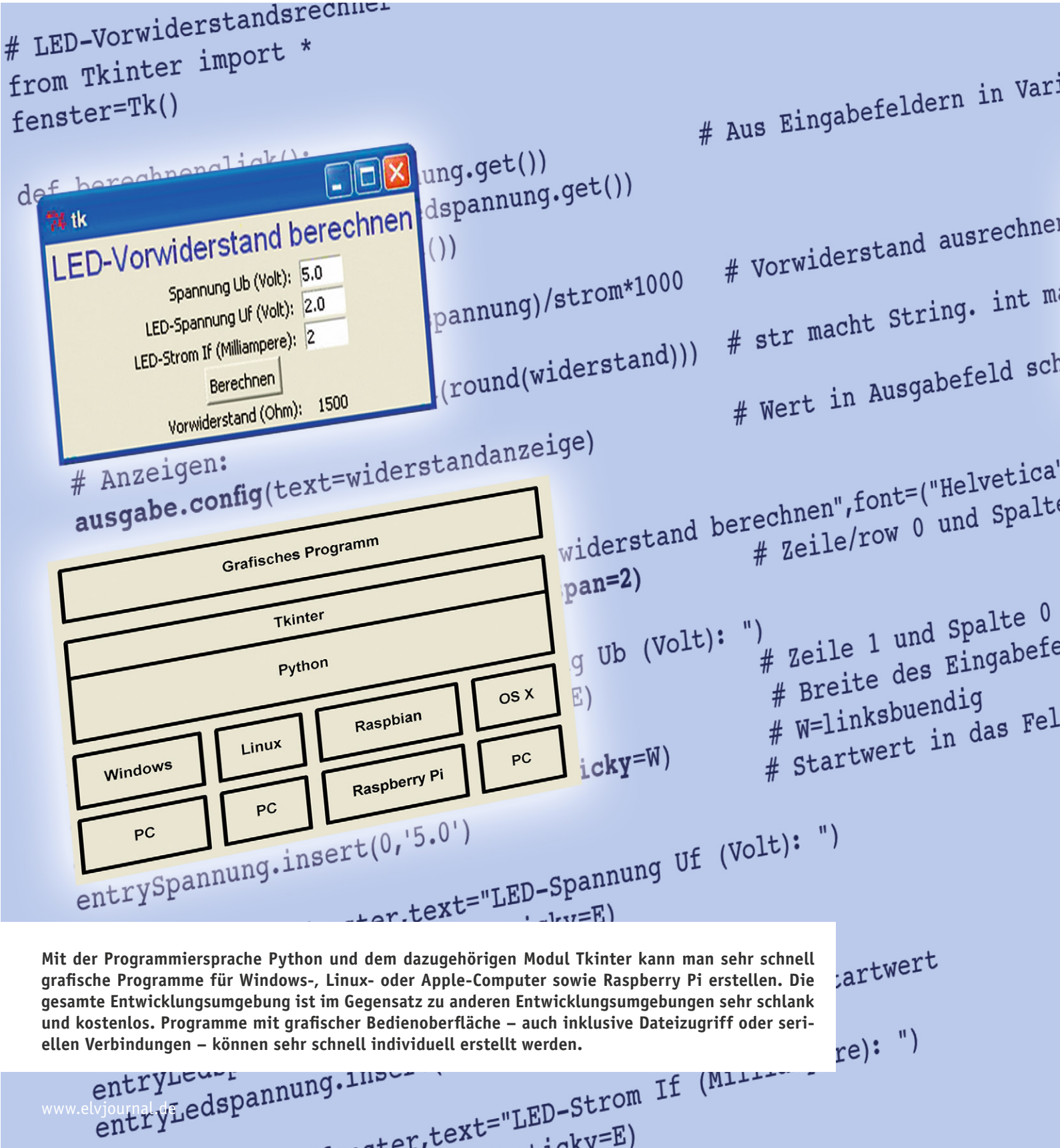
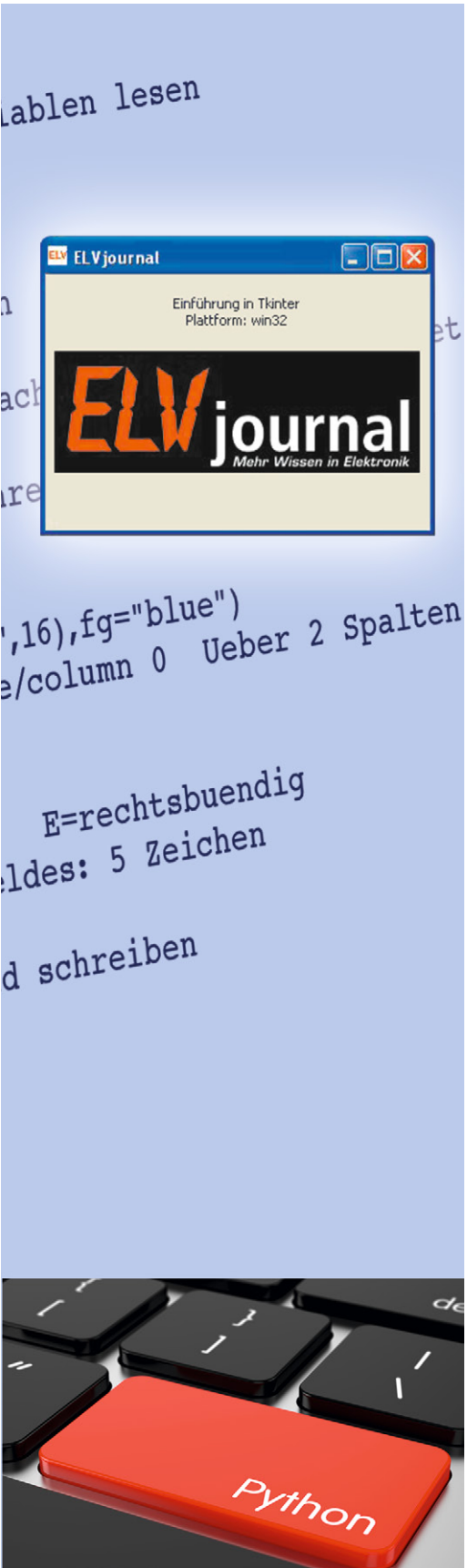
Tkinter

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

Teil 1



Mit der Programmiersprache Python und dem dazugehörigen Modul Tkinter kann man sehr schnell grafische Programme für Windows-, Linux- oder Apple-Computer sowie Raspberry Pi erstellen. Die gesamte Entwicklungsumgebung ist im Gegensatz zu anderen Entwicklungsumgebungen sehr schlank und kostenlos. Programme mit grafischer Bedienoberfläche – auch inklusive Dateizugriff oder seriellen Verbindungen – können sehr schnell individuell erstellt werden.



Im vorliegenden Artikel wird anhand einfacher Beispiele eine Einführung in die Programmierung grafischer Programme mit Tkinter und Python gegeben. Dabei werden die Installation von Python und Tkinter, die grundsätzliche Vorgehensweise und der Aufbau eines Tkinter-Programms gezeigt. Kleine Beispiele führen grundlegende Konstrukte ein und zur Abrundung werden Programme mit Dateizugriff und mit seriellen Verbindungen gezeigt. Der Artikel gibt einen „Vorgeschmack" von den Möglichkeiten. Dabei werden Sachverhalte und Aufgaben zum Teil vereinfacht dargestellt bzw. gelöst, da eine vollständige Darstellung aller Möglichkeiten den Rahmen des Artikels sprengen würde.

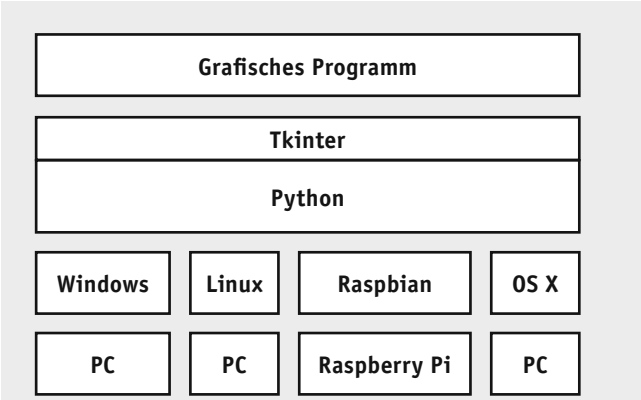


Bild 1: Entwicklungssysteme und Zielsysteme

Warum Tkinter?

Es gibt viele Möglichkeiten, grafische Anwendungen für verschiedene Systeme zu erstellen, die aber oft den Nachteil haben, dass sie kostenpflichtig, überladen oder auf eine Betriebssystemplattform beschränkt sind. Tkinter (= „Toolkit-Interface") ist ein Zusatz (Modul/Bibliothek) zu der weit verbreiteten Programmiersprache Python. Weil Tkinter im Python-Bereich die Standardbibliothek zur Erstellung grafischer Programme ist, wird es mit jeder Python-Installation automatisch mitinstalliert. Python wiederum ist bei praktisch allen Linux-basierten Systemen und (so) auch auf OS-X-Computern bereits vorinstalliert und auf Windows-Systemen lässt es sich sehr schnell kostenlos installieren.

Die Einarbeitung in Tkinter ist nicht schwierig. Man fängt – wie im vorliegenden Artikel – mit einem Hallo-Welt-Beispiel an und baut das Programm dann Schritt für Schritt aus. Im Internet gibt es jede Menge Tutorials über Tkinter [1] und in den meisten Python-Büchern gibt es ein mehr oder weniger großes Kapitel über die Arbeit mit Tkinter. Empfehlenswert ist das Buch „Raspberry Pi programmieren mit Python" (Best.-Nr. CG-12 16 55).

Python – und damit auch Tkinter – gibt es für praktisch alle gängigen Betriebssysteme. Man kann sich schnell ein kleines Programm für einen Windows-PC schreiben und man kann dasselbe Programm auch auf einem Raspberry Pi oder einem OS-X-Computer laufen lassen. Jede Entwicklungsplattform (Bild 1) kann auch zur Laufzeit verwendet werden. Ein LED-Vorwiderstandsrechner, ein Baudratenfehler-Rechner, ein Rechenlernprogramm für die Kinder, eine grafische Oberfläche für einen Sensor mit serieller Ausgabe, ein Datenlogging-Programm und vieles mehr ist leicht umsetzbar – für Windows, OS X, Linux oder Raspbian.

Tkinter ist ebenso wie Python kostenlos und es lässt sich einerseits schnell erlernen und benutzen und ist andererseits so mächtig, dass die meisten Aufgaben damit gelöst werden können. Auf den ersten Blick mag es verwundern, dass bei Tkinter textorientiert in einem reinen

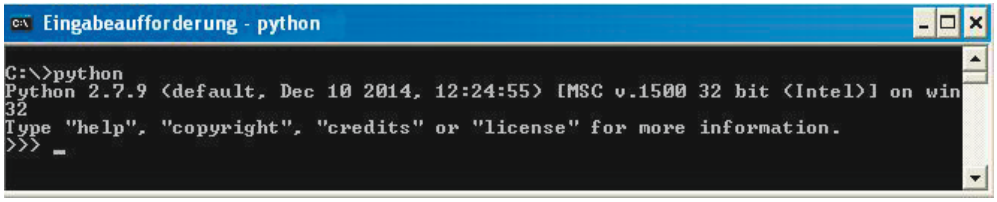


Bild 2: Python-Eingabefenster

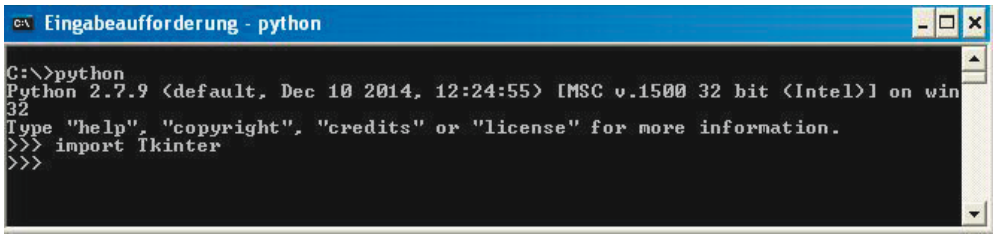


Bild 3: Tkinter-Import

Texteditor programmiert wird. Man sieht aber bald, dass dadurch nicht die oftmals komplexe Bedienung einer grafischen Entwicklungsoberfläche erlernt werden muss. Man benutzt nur das, was man benötigt, und die Programmdateien werden nicht groß.

Installation

Testen, ob Python bereits installiert ist

Zunächst sollte getestet werden, ob Python bereits auf dem Entwicklungssystem (und auch auf dem Zielsystem) installiert ist. Dazu ist zu erläutern, dass es verschiedene Python-Versionen gibt. Es gibt eine Python-2.x-Schiene und eine Python-3.x-Schiene. Es ist zwar, wie zu vermuten, so, dass Python 3.x die neuere Entwicklung darstellt und langfristig strategisch ist, aber anders als die Zahlen andeuten, ist Python 2.x damit noch längst nicht überholt und wird auch sehr aktiv weiter (parallel) genutzt. Da Python 2.7 bei OS X enthalten ist und da es sehr viele Beispiele im Internet und in Büchern gibt, wird im vorliegenden Artikel Python 2.7 verwendet. Die Unterschiede bei Benutzung von Python 3.x sind, was Tkinter betrifft, praktisch null und bezüglich der Python-Kommandos überschaubar. Man sollte sich auf eine Schiene – Python 2.x oder Python 3.x – festlegen und dabei bleiben. Als Erstes öffnet man die Eingabeaufforderung/Kommandozeile („DOS-Fenster") bzw. bei Linux-Systemen ein Terminalfenster. Dort gibt man *python* ein (Bild 2). Das kann man in einem Windows-, OS-X-, Linux- oder Raspbian-System tun. Wenn Python bereits installiert ist, wird sich ein Python-System mit einem Python-Prompt (>>>) melden (siehe Bild 2). Verlassen des Fensters ist mit *quit()* oder *Strg-z* (Windows) bzw. *Strg-c* (Linux) möglich.

Falls Python (wie unter Windows wahrscheinlich) noch nicht installiert ist, wird es unter [2] unter „Downloads" in der Version 2.7.x heruntergeladen und installiert. Bei der Installation bitte ankreuzen, dass python.exe zum Pfad hinzugefügt werden soll. Nach der Installation und Start von Python mit *python* können hinter dem Python-Prompt (>>>) Python-Befehle eingegeben werden.

Testen, ob Tkinter einbindbar ist

Beim Python-Prompt (>>>) wird nun testweise *import Tkinter* eingegeben (Bild 3). Dabei ist darauf zu achten, dass Tkinter mit großem Anfangsbuchstaben geschrieben wird (bei Python 3.x klein). Python unterscheidet generell Groß-/Kleinschreibung! Wenn es keine Fehlermeldung gibt, dann ist Tkinter einbindbar und damit nutzbar (siehe Bild 3). (Wer extrem neugierig ist, kann jetzt sogar schon einmal *Tkinter.Tk()* eingeben und wird staunend ein sich öffnendes, funktionsfähiges Fenster sehen!!)

Hello World

Nachdem verifiziert wurde, dass Python mit Tkinter installiert ist, wird ein erstes kleines Programm in einem Texteditor eingegeben. Als Texteditor kann man jeden beliebigen (vorinstallierten) Texteditor benutzen – was vollkommen ausreicht – oder einen Editor einsetzen, der Syntax-Highlighting bietet wie Notepad++ [3], Editra [3] oder die mit Python installierte IDLE-Entwicklungsumgebung (s. u.). Das berühmte Programm „Hello World" oder „Hallo Welt" ist in Bild 4 zu sehen.

Sobald das Programm im Texteditor eingegeben und unter einem Namen wie *hallowelt.py* gespeichert wurde, kann es ausgeführt werden. Hierzu bewegt man sich im Kommandofenster (mit *CD xx*) in das Verzeichnis, in dem die Programmdatei abgespeichert ist und gibt dort *python hallowelt.py* ein. Das Fenster, welches sich dann öffnet, entspricht den Standards des jeweiligen Betriebssystems (Bild 5).

Erläuterung des Programms:

Mit einem Doppelkreuz (#) werden in Python Kommentare eingeleitet. Mit *from Tkinter import* * wird der Python-Interpreter angewiesen, alle Methoden des Tkinter-Moduls zu importieren. Das ist nötig, um im weiteren Programm Tkinter-Methoden benutzen zu können (vgl. „Elektronikwissen" zum Thema Importieren im zweiten Teil dieses Artikels). Mit *root=Tk()*

```
# Das Programm „Hallo Welt mit Tkinter"

from Tkinter import *

root = Tk()

meinlabel=Label(root, text='\n     ELVjournal     \n      Programmieren mit Tkinter    \n')
meinlabel.pack()

root.mainloop()
```

Bild 4: Das Programm „Hallo Welt"

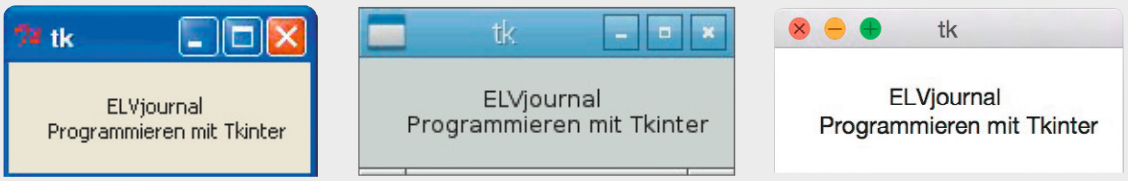


Bild 5: Hallo-Welt-Fenster (links Windows, Mitte Raspbian, rechts OS X)

wird ein Tkinter-Fensterobjekt mit dem Namen „root" erzeugt. Der Name ist frei wählbar. Statt „root" könnte man das Fenster auch „meinfenster" oder anders nennen. In diesem ersten Beispiel wurde „root" gewählt, weil man diese Bezeichnung in der Literatur oft wiederfindet. Wichtig ist, dass dieser Name im späteren Programmverlauf referenziert wird. Durch sogenannte Widgets (Bausteine) kann das erzeugte Fenster mit Inhalt gefüllt werden. Das Label-Widget erzeugt einen Text im Fenster. Durch die Parameter in der Klammer wird definiert, in welchem Fenster (hier „root") das Label erzeugt wird und welcher Text angezeigt werden soll. Durch *\n* wird ein Zeilenumbruch erzeugt. Das Label wird zunächst nur mit dem Namen „meinlabel" definiert, aber noch nicht angezeigt. Erst durch die Zeile *meinlabel.pack()* wird das Label tatsächlich auch wie in Bild 5 angezeigt.

Die letzte Zeile – *root.mainloop()* – ist sehr wichtig, weil sie bewirkt, dass das Fenster für Interaktionen durch den Benutzer oder Ereignisse vom System bereit ist.

Durch Eingabe von *python hallowelt.py* wird das Programm ausgeführt. Eventuelle Fehler werden im Kommandofenster angezeigt. Unter Windows kann man das Programm auch durch Doppelklick auf den Dateinamen im Datei-Explorer starten oder einen Shortcut auf den Desktop legen. Wenn man als Dateierweiterung .pyw statt .py verwendet, öffnet sich das Fenster direkt – ohne Umweg über ein Kommandofenster.

Unter Raspbian/Linux kann das Programm mit dem CHMOD-Befehl *(CHMOD 777 hallowelt.py)* ausführbar gemacht werden. Im Dateimanager von Raspbian (Menü ◁⇆⇨ Zubehör → Dateimanager) kann man unter Bearbeiten → Einstellungen → Allgemein den Haken bei *Abfragen von Optionen beim Starten von ausführbaren Dateien deaktivieren* setzen, damit das Programm durch Doppelklick auf den Dateinamen sofort startet. Auch hier kann man eine Kopie der Datei auf den Desktop legen und hat dann ein durch Doppelklick startbares Programm. Das Programm kann unter den verschiedenen Betriebssystemen ohne jede Änderung benutzt werden! Tkinter setzt die Darstellung passend zu den Betriebssystemstandards um (siehe Bild 5).

Generell besteht ein Tkinter-Programm aus folgenden Blöcken/Aspekten:

- **Importieren** des Tkinter-Moduls in das Python-Programm
- **Hauptfenster** erstellen mit *Tk()*
- Ggf. **Prozeduren** definieren
- **Widgets:** Was soll auf dem Fenster erscheinen?
- **Layout:** Wo und wie sollen die Widgets erscheinen?
- **Interaktivität:** Was soll bei Interaktionen oder Ereignissen geschehen?
- Starten der **Endlosschleife** für das Hauptfenster mit *root.mainloop()*

Button und Messagebox

Das Hallo-Welt-Beispiel soll nun um einen Pushbutton und eine Messagebox erweitert werden. Ziel ist ein kleines Programm, wie in Bild 6 zu sehen. Dafür wird das Programm aus Bild 7 in den Texteditor eingegeben und gespeichert.

Erläuterungen:
Nach dem Importieren der benötigten Module und dem Erzeugen eines Fensters mit dem Namen „meinfenster" und der Startgröße 200 x 100 wird eine Prozedur definiert, welche später aufgerufen wird. Neu ist hier das Widget „Button", durch welches ein Pushbutton mit den in den Parametern *text, command, background* und *foreground* beschriebenen Eigenschaften und dem Namen „meinbutton" erzeugt wird. Der Parameter *text* gibt an, welcher Text auf dem Pushbutton angezeigt werden soll. Der Parameter *command* gibt an, welche Prozedur ausgeführt werden soll, wenn der Button gedrückt wird. Es soll ein Sprung zu „mein_unterprogramm" erfolgen. Dort wird dann eine Messagebox mit dem Titel „Titel" und dem Text „Danke" angezeigt.

Blockbildungen erfolgen bei Python durch Einrückungen, wobei man per Konvention jeweils um

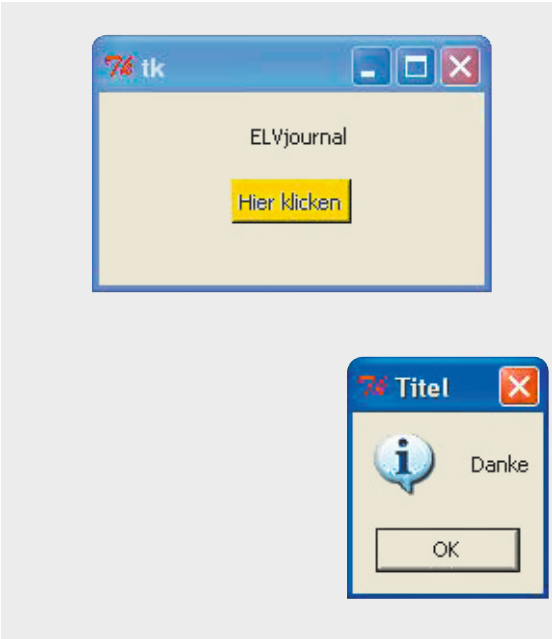


Bild 6: Button und Messagebox

```
# Button und Messagebox

from Tkinter import *  # Bei Python 3: from tkinter import *
import tkMessageBox    # Bei Python 3: import tkinter.messagebox

meinfenster= Tk()
meinfenster.geometry("200x100")  # Optional: Groesse des Fensters

#Unterprogramme
def mein_unterprogramm() :
    tkMessageBox.showinfo("Titel", "Danke")

meinlabel=Label(meinfenster , text='\n     ELVjournal  \n')
meinlabel.pack()

meinbutton=Button(text="Hier klicken", command=mein_unterprogramm, background="yellow", foreground="blue")
meinbutton.pack()

meinfenster= meinfenster.mainloop()
```

Bild 7: Button und Messagebox

4 Zeichen einrücken sollte (was hier aus Layoutgründen nicht überall gemacht wurde). Wichtig ist auch hier wieder die letzte Zeile des Programms, die es ermöglicht, dass das Hauptfenster überhaupt auf das Klicken des Buttons reagieren kann.

Pushbuttons lassen sich vielfältig gestalten (Farben, Größe, Inhalt usw.) und es gibt diverse Arten von Messageboxen: *showinfo()*, *showwarning()*, *showerror()*, *askquestion()*, *askokcancel()*, *askyesno()*, *askretrycancel()*.

Layout-Manager

Es gibt drei unterschiedliche Methoden, Widgets platzieren zu lassen. Man spricht von Layout-Management. Bisher wurde es Tkinter überlassen, die Widgets zu platzieren – also das Layout zu machen. Die Widgets wurden in der Reihenfolge, wie sie aktiviert wurden, in das Fenster „gepackt" (Pack). Bei dieser Methode erscheinen alle Widgets im Fenster, aber man hat nicht viele Gestaltungsmöglichkeiten, was für kleine Anwendungen passen kann, aber nicht immer schöne bzw. sinnvolle Oberflächen ergibt. Daher gibt es zwei weitere Layout-Manager.

Die drei Layout-Manager von Tkinter heißen:

- *pack()* Die Widgets werden in der Reihenfolge, wie sie aktiviert werden, dargestellt.
- *grid()* Der Programmierer platziert die Widgets gezielt in einem Raster/Gitter.
- *place()* Der Programmierer platziert die Widgets anhand von Koordinaten.

pack() und *grid()* können nicht in einem Frame gemischt verwendet werden, während *place()* mit *pack()* oder *grid()* im selben Frame benutzt werden kann. *pack()* eignet sich für kleine Layouts, die auf die Schnelle erstellt werden sollen. *grid()* ist sehr komfortabel zum Erstellen schöner, strukturierter Oberflächen, und *place()* wird eher selten benutzt.

LED-Vorwiderstandsrechner (Grid-Layout)

Zur Demonstration des Grid-Layout-Managers soll ein kleines Programm zur Berechnung eines LED-Vorwiderstands erstellt werden. Ziel ist ein Programm wie in Bild 8 links.

Die Widgets werden in ein Raster (= Grid) aus Zeilen (= rows) und Spalten (= columns) platziert.

Dabei stellt man sich das gesamte Fenster in ein Raster unterteilt vor (wie bei einem Tabellenkalkulationsprogramm) (Bild 8 rechts). Man muss das Raster nicht in seinen Dimensionen definieren, sondern es wird von Tkinter aufgrund der Gesamtheit der Grid-Platzierungen berechnet.

Bei der Aktivierung eines Widgets wird jeweils mit *meinwidget.grid(row=y, column=x)* angegeben, in welchem Feld das Widget platziert werden soll. Zusätzlich kann angegeben werden, ob ein Widget sich über mehrere Spalten oder Zeilen erstrecken soll (span) oder ob es ausgerichtet (sticky) statt mittig sein soll (Tabelle 1).

Das Programm wird wie in Bild 9 im Editor eingegeben und gespeichert.

Erläuterungen:

Nach dem Importieren des Tkinter-Moduls und dem Erstellen des Fensters sieht man die Definition der Prozedur, die später durch Klicken des „Berechnen"-Pushbuttons aufgerufen wird. Blockbildung erfolgt wie bei Python üblich durch Einrückung von Zeilen. Variablen müssen bei

Bild 8: Screenshot Vorwiderstandsrechner (rechts mit Grid)

Python nicht explizit deklariert werden, sondern können sofort benutzt werden.

Wie bei Tkinter üblich, werden zunächst alle Widgets erstellt und – diesmal mit *grid()* – platziert. Die Überschrift ist ein Label, das im Fenster mit dem Namen „fenster" und einem Text in einer definierten Größe und Farbe definiert wird. Für die Spannung Ub, die LED-Flussspannung Uf und den LED-Strom If werden jeweils ein Textlabel (Label) und ein Eingabefeld (Entry) definiert. Für jedes Widget wird als erster Parameter angegeben, in welchem Fenster es sein soll.

Bei den Entry-Feldern wird die Breite auf 5 Zeichen begrenzt. Mit *grid()* wird für jedes Widget definiert, an welcher Stelle im Raster es eingetragen werden soll. Außer der Breite (width) könnte man beispielsweise auch die Textfarbe (*fg* = foreground) oder die Hintergrundfarbe (*bg* = background) definieren.

Dem Button-Widget wird durch *columnspan = 2* erlaubt, zwei Spalten Platz zusammenhängend zu beanspruchen, innerhalb deren es wiederum (standardmäßig) zentriert erscheint. Die Entry-Felder bekommen mit *insert()* schon einmal Startwerte zugewiesen. Das Label „ausgabe" bekommt zunächst einen Starttext („___"), der dann in der Prozedur später mit *ausgabe.config()* nach Berechnung des Widerstandswertes aktualisiert wird.

Tabelle 1

Optionen des Grid-Layout-Managers

row	Zeilennummer
column	Spaltennummer
columnspan	das Widget soll sich über n Spalten erstrecken
rowspan	das Widget soll sich über n Zeilen erstrecken
sticky	das Widget soll an einer Seite oder Ecke der Zelle „festgeklebt" werden (sticky = klebrig) Die Bezeichnungen entsprechen den Himmelsrichtungen: N = oben, S = unten, E = rechts, W = links NE = rechts oben, SE = rechts unten, SW = links unten, NW = links oben N+S = vertikal strecken, E+W = horizontal strecken, N+E+S+W = in alle Richtungen strecken

```
# LED-Vorwiderstandsrechner
from Tkinter import *
fenster=Tk()

def berechnenclick():
  spannung = float(entrySpannung.get())           # Aus Eingabefeldern in Variablen lesen
  ledspannung = float(entryLedspannung.get())
  strom= float(entryStrom.get())

  widerstand= (spannung-ledspannung)/strom*1000   # Vorwiderstand ausrechnen   Rv=(Ub-Uf)/If*1000

  widerstandanzeige= str(int(round(widerstand)))  # str macht String. int macht Ganzzahl. round rundet
  # Anzeigen:
  ausgabe.config(text=widerstandanzeige)          # Wert in Ausgabefeld schreiben


textfeld=Label(fenster,text="LED-Vorwiderstand berechnen",font=("Helvetica",16),fg="blue")
textfeld.grid(row=0,column=0,columnspan=2)        # Zeile/row 0 und Spalte/column 0  Ueber 2 Spalten

textfeld=Label(fenster,text="Spannung Ub (Volt): ")
textfeld.grid(row=1,column=0,sticky=E)            # Zeile 1 und Spalte 0   E=rechtsbuendig
entrySpannung=Entry(fenster,width=5)              # Breite des Eingabefeldes: 5 Zeichen
entrySpannung.grid(row=1,column=1,sticky=W)       # W=linksbuendig
entrySpannung.insert(0,'5.0')                     # Startwert in das Feld schreiben

textfeld=Label(fenster,text="LED-Spannung Uf (Volt): ")
textfeld.grid(row=2,column=0,sticky=E)
entryLedspannung=Entry(fenster,width=5)
entryLedspannung.grid(row=2,column=1,sticky=W)
entryLedspannung.insert(0,'2.0')                  # Startwert

textfeld=Label(fenster,text="LED-Strom If (Milliampere): ")
textfeld.grid(row=3,column=0,sticky=E)
entryStrom=Entry(fenster,width=5)
entryStrom.grid(row=3,column=1,sticky=W)
entryStrom.insert(0,'2')                          # Startwert

buttonBerechnen = Button(fenster,text='Berechnen', command=berechnenclick)
buttonBerechnen.grid(row=4,column=0, columnspan=2)

textfeld=Label(fenster,text="Vorwiderstand (Ohm): ")
textfeld.grid(row=5,column=0,sticky=E)
ausgabe=Label(fenster,text="___",width=5)
ausgabe.grid(row=5,column=1,sticky=W)

fenster.mainloop()
```

Bild 9: Tkinter-Programm Vorwiderstandsrechner

URI-Rechner (ohmsches Gesetz) (Place-Layout)

Anhand eines kleinen Rechenprogramms zum ohmschen Gesetz (Bild 11) wird nun der dritte Layout-Manager *place()* vorgestellt. Der selektierte Wert soll aus den anderen beiden Werten nach Klicken des Pushbuttons berechnet werden.

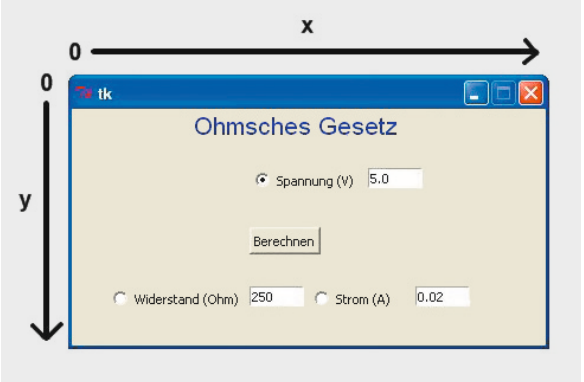


Bild 10: Rechner zum ohmschen Gesetz mit Place-Layout-Manager

Erläuterungen:

Mit *place()* werden in diesem Programm Labels (die Überschrift), Pushbuttons (buttonBerechnen), Entry-Felder und (hier neu) Radiobuttons per Koordinatenangabe im Fenster platziert. Der Koordinatenursprung ist links oben (Bild 10). Jedes einzelne Widget kann gezielt im Fenster irgendwo platziert werden. Die Radiobutton-Widgets sind dadurch miteinander inhaltlich verbunden, dass sie sich auf dieselbe oben im Programm deklarierte Variable *zuberechnen* beziehen. Der Parameter *value* legt den Wert der Variablen für das jeweilige Radiobut-

ton-Widget fest. Wenn ein Radiobutton angeklickt wird, werden die anderen dazugehörigen Radiobuttons deselektiert.

Man kennt Radiobuttons normalerweise als untereinander stehende, sich ausschließende Auswahlmöglichkeiten. Hier sind die Radiobuttons ungewöhnlich weit voneinander entfernt, aber funktionieren wie gewohnt. Sobald der Pushbutton *buttonBerechnen* geklickt wird, wird diejenige Größe aus dem „U=R*I"-Dreieck berechnet, die per Radiobutton angewählt ist. Dafür wird die Variable *zuberechnen* mit IF-Abfragen ausgewertet. Verglichen wird jeweils mit den Werten, die die Radiobuttons per *value*-Parameter zugewiesen bekommen hatten.

Countdown-Timer

Anhand eines kleinen Countdown-Timers, den man beispielsweise als Belichtungsgerät-Timer beim Platinenätzen verwenden könnte, kann man sehen, wie kurz mit Tkinter erstellte einsatzfähige Programme sein können. Die Idee ist, ein Anwendungsfenster mit einem Startwert (Sekunden) zu haben, bei dem man den Countdown durch Anklicken eines Pushbuttons starten kann (Bild 12 oben). Bei Ablauf der Zeit stoppt der Countdown, der Hintergrund wird rot und der Pushbutton wird disabled (Bild 12 unten). Die Umsetzung als Tkinter-Programm zeigt Bild 13.

Erläuterungen:

Das Programm ist erstaunlich kurz! Mit *place()* wird ein Textlabel platziert, das hier eine Position und Größe bekommt. Der Inhalt des Labels ist der Inhalt der oben initialisierten Variablen *stand* – hier 15. Die Hintergrundfarbe (*bg* = background) des Labels ist zunächst grau *(gray)* und es wird eine Schriftart in der Größe 36 gewählt. Auch der Pushbutton erhält eine Position, eine Größe und eine Hintergrundfarbe *(bg)*. Wird der Pushbutton angeklickt, dann wird in der Prozedur *buttonCountdownClick* die Variable dekrementiert. Die Zeile *labelZahl.config(text=str(stand))* schreibt den neuen Wert der Variablen *stand* in das Label *labelZahl*. Die Zeile *tkFenster.after(1000, buttonCountdownClick)* bewirkt, dass nach (after) 1000 Millisekunden, also nach einer Sekunde, die Prozedur

```
# Rechner zum ohmschen Gesetz
from Tkinter import *
fenster=Tk()
fenster.geometry("400x200")                     # Groesse des Fensters
fenster.resizable(width=FALSE, height=FALSE)    # Fenstergroesse nicht veraenderbar
zuberechnen=IntVar()                             # Variable fuer die Radiobuttons deklarieren

#Prozedur zum Berechnen des gesuchten Wertes:
def berechnenclick():
  spannung = float(entrySpannung.get())         # Aus Eingabefeldern in Variablen lesen
  widerstand = float(entryWiderstand.get())
  strom= float(entryStrom.get())
                                                 # Je nach Radiobutton ....
  if zuberechnen.get()==0:                       # .. Spannung berechnen
    spannung=widerstand*strom
    spannunganzeige= str(spannung)
    entrySpannung.delete(0,END)
    entrySpannung.insert(0,spannunganzeige)
  elif zuberechnen.get()==1:                     # .. Widerstand berechnen
    widerstand=spannung/strom
    widerstandanzeige= str(widerstand)
    entryWiderstand.delete(0,END)
    entryWiderstand.insert(0,widerstandanzeige)
  elif zuberechnen.get()==2:                     # .. Strom berechnen
    strom=spannung/widerstand
    stromanzeige= str(strom)
    entryStrom.delete(0,END)
    entryStrom.insert(0,stromanzeige)

# Hauptfenster:
textfeld=Label(fenster,text="Ohmsches Gesetz",font=("Helvetica",16),fg="blue")
textfeld.place(x=100,y=0)                       # An Position x=100 und y=0

spannung_rb=Radiobutton(fenster,value=0, text="Spannung (V)",variable=zuberechnen)   # Radiobutton Spannung
spannung_rb.place(x=150,y=50)                    # Radiobutton Spannung platzieren
entrySpannung=Entry(fenster,width=7)             # Breite des Eingabefensts: 7 Zeichen
entrySpannung.place(x=250,y=50)                  # Eingabefenster platzieren
entrySpannung.insert(0,'5.0')                   # Startwert Spannung eintragen

buttonBerechnen = Button(fenster,text='Berechnen', command=berechnenclick)
buttonBerechnen.place(x=150,y=100)

widerstand_rb=Radiobutton(fenster,value=1,text="Widerstand (Ohm)",variable=zuberechnen)
widerstand_rb.place(x=30,y=150)
entryWiderstand=Entry(fenster,width=7)
entryWiderstand.place(x=150,y=150)
entryWiderstand.insert(0,'250')                 # Startwert Widerstand

strom_rb=Radiobutton(fenster,value=2,text="Strom (A)",variable=zuberechnen)
strom_rb.place(x=200,y=150)
entryStrom=Entry(fenster,width=7)
entryStrom.place(x=290,y=150)
entryStrom.insert(0,'0.02')                     # Startwert Strom

fenster.mainloop()
```

Bild 11: Programm zum ohmschen Gesetz

Bild 12: Countdown-Timer

```
# Countdown-Zaehler, der im Sekundentakt von einer festen Zahl herunterzaehlt.
from Tkinter import *

stand = 15                                          # Startwert

# Ereignisbehandlung
def buttonCountdownClick():
  stand = int(labelZahl.cget('text'))
  if stand > 0:                                     # Wenn > 0  ..
    stand = stand - 1                               #   ... herunterzaehlen
    labelZahl.config(text=str(stand))              #   Labeltext updaten
    tkFenster.after(1000, buttonCountdownClick)    #   Nach 1000 ms = 1 Sekunde buttonCountdownClick
  if stand == 0:                                    # sonst ..
    labelZahl.config(bg='red')                     #   Hintergrund auf rot
    buttonCountdown.config(text='ENDE',bg='white',state=DISABLED)


# Fenster
tkFenster = Tk()
tkFenster.geometry("170x125")                       # Groesse des Fensters
tkFenster.resizable(width=FALSE, height=FALSE)     # Fenstergroesse nicht veraenderbar

# Label
labelZahl = Label(master=tkFenster, text=stand, bg='gray', font=('Arial', 36))
labelZahl.place(x=5, y=5, width=160, height=80)

# Button
buttonCountdown = Button(master=tkFenster, text='countdown', bg='yellow',
                         command=buttonCountdownClick)
buttonCountdown.place(x=5, y=90, width=160, height=30)


# Aktivierung des Fensters
tkFenster.mainloop()
```

Bild 13: Programm Stoppuhr

wieder neu aufgerufen wird. Dadurch erfolgt das sekündliche Herunterzählen. Wenn die Variable *stand* den Inhalt null bekommt, wird die Hintergrundfarbe des Labels auf Rot gesetzt und der Pusbutton wird disabled. Ein erneuter Aufruf der Prozedur nach einer weiteren Sekunde wird nicht wieder angestoßen.

Im zweiten Teil dieses Artikels wird der Zugriff auf Dateien bzw. die serielle Schnittstelle mit kleinen Python/Tkinter-Programmen sowie die weitere Verschönerung der Fenster beschrieben. ELV

Weitere Infos:

[1] Tkinter-Tutorials (Auswahl):
www.python-kurs.eu/python_tkinter.php
www.wspiegel.de/tkinter/tkinter01.htm
www.weigu.lu/c/python/python_tkinter.html
www.inf-schule.de/software/gui/entwicklung_tkinter
wiki.python.org/moin/TkInter
[2] www.python.org
[3] Editoren mit Syntax-Highlighting:
https://notepad-plus-plus.org
http://editra.org
IDLE: Entwicklungsumgebung mit Editor, Python-Interpreter, Python-Hilfe. In Python enthalten.

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.