Python GUI mit TKinter

Kurze Einführung

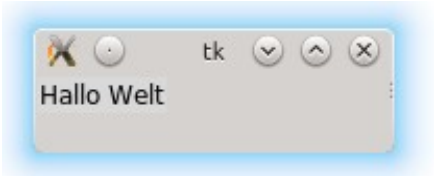
Fast alle moderne Programme besitzen eine graphische Benutzeroberfläche (***Graphical User Interface (GUI)***). Die Bedienung des Programms erfolgt also nicht mehr über Texteingaben im Terminal-Emulator (Konsole) sondern über interaktive Elemente wie Knöpfe (*buttons*) oder Kontrollkästchen (*checkbox*) mit der Maus.

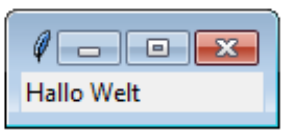
Das Python-Modul **Tkinter** dient der Gestaltung einer solchen graphischen Benutzeroberfläche. Es existieren noch andere solcher GUIs für Python (GTK, Qt), allerdings ist Tkinter schon in Python enthalten und soll hier genutzt werden. Um die modernere Version von TKinter (mit thematischen Ttk-Widgets) nutzen zu können, wird eine neuere Python Version >= 3.1 (>= 2.7) benötigt. Wir verwenden hier eine dreier Version.

Hallo Welt

```
#!/usr/bin/env python3
# tk_hallo.py

from tkinter import *    # Python 2.7 "from Tkinter import *"
from tkinter import ttk  # Python 2.7 "import ttk"

mainWin = Tk()

label_1 = ttk.Label(mainWin, text='Hallo Welt')
label_1.grid()

mainWin.mainloop()
```

Nachdem das erste Programm als **`tk_hallo.py`** (Windows `tk_hallo.pyw` damit die Konsole nicht aufgerufen wird) abgespeichert wurde und mit **`python3 tk_hallo.py`** (bzw.: `C:\Python34\pythonw tk_hallo.pyw`) aufgerufen wurde erscheint ein Fenster mit dem entsprechenden beschrifteten Button. (Linux bzw. Windows).






```
#!/usr/bin/env python3
```

```
# tk_hallo.py
```

Die erste Zeile ist an sich ein Kommentar, der aber unter Linux aufzeigt mit welchem Programm das Python-Script ausgeführt werden soll. Dazu muss die Datei allerdings die richtigen Rechte besitzen, d.h. sie für den Anwender ausführbar sein (Befehl: **chmod**). Die zweite Zeile enthält den Dateinamen als Kommentar.

```
from tkinter import *    # Python 2.7 "from Tkinter import *"
from tkinter import ttk  # Python 2.7 "import ttk"
```

In den nächsten beiden Zeilen werden zwei Module (Klassenbibliotheken) geladen. Beim ersten Modul (Standard Tkinter Klassen) wird alles importiert, so dass auf diese Klassen ohne den Vorsatz "`tkinter.`" zugegriffen werden kann. Beim zweiten Import wird nur Ttk geladen, um die neueren Ttk-Widgets nutzen zu können. Um sie von den Standard-Widgets zu unterscheiden muss hier dann der Vorsatz "`ttk.`" verwendet werden.

```
mainWin = Tk()
...
mainWin.mainloop()
```

Grafische Elemente (Hauptfenster, Knöpfe, Text, Eingabefelder ...) einer GUI werden **Widgets** genannt (*window gadgets*). Ein Widget kann aus mehreren Widgets zusammengesetzt sein.

Jedes Widget entspricht einer Klasse (Bauplan für Objekte), aus der dann die Objekte gebildet werden. Jedes Objekt besitzt Attribute (Daten des Objekts) und Methoden (Funktionen) des Objekts. Eine GUI kann man sich als Baumstruktur vorstellen. Die Wurzel ist das Hauptfenster, ein Objekt der Klasse Tk. In der dritten Zeile wird diese Objekt mit dem (beliebigen) Namen "`mainWin`" erstellt. In der letzten Zeile wird dann die Methode (Funktion) `mainloop()` des Objekts `mainWin` aufgerufen. Dabei werden Objekt und Methode mit einem Punkt verbunden.

```
label_1 = ttk.Label(mainWin, text='Hallo Welt')
label_1.grid()
```

In der vierten Zeile wird ein Ttk-Objekt mit dem Namen `label_1` erzeugt. Dieses Objekt ist ein "Kind" des Hauptfensters. Dies wird durch den ersten Eintrag (Parameter) in der Klammer festgelegt. Die Beschriftung des Labels wird beim zweiten Parameter in der Variablen "`text`" hinterlegt. Dem Widget **`ttk.Label`** können noch andere Parameter übergeben werden (siehe später). Das Objekt wurde zwar erzeugt, ist aber noch nicht sichtbar, da Tk nicht klar ist wie das Widget in Bezug zum Hauptfenster platziert werden soll. Dies übernimmt die Methode `grid()` in der fünften Zeile.

Aufgabe Tk1:

Teste das "Hallo Welt" Programm auf dem Raspberry Pi.

Einige Basis Widgets

Neben dem Hauptfenster und dem Label existieren noch viele andere Widgets wie zum Beispiel Knöpfe (*button, checkbutton, radiobutton*), Auswahlfelder (*combobox, listbox, spinbox*), Eingabefelder (*Entry, Text*) usw.

Um ein erstes sinnvolles Programm zu schreiben wollen wir uns ***Label***, ***Entry*** und ***Button*** etwas näher ansehen:

Label

Labels sollen dem Betrachter Informationen oder Resultate liefern. Hierfür wird das Widget **`ttk.Label`** genutzt. Der darzustellende Text wird meist mit dem "`text=`" Parameter übergeben und ist konstant (siehe Kapitel "Hallo Welt"). Man kann aber stattdessen auch den Parameter "`textvariable=`" benutzen um veränderbaren Text, also eine Variable, zu übergeben:

```
result = StringVar()
result.set('neuer_Wert')
label_1 = ttk.Label(mainWin, textvariable=result)
label_1.grid()
```

Tkinter ermöglicht uns die Klasse `StringVar()` für die Textvariable zu nutzen. Diese kümmert sich um die ganze Logistik zur Überwachung von Änderungen der Variablen und zur Kommunikation zwischen der Variable und dem Label.

Auch ist es möglich Bilder im Label anzuzeigen (Parameter: "`image=`"). Mit dem Parameter "`compound=`" gibt man an, ob nur der Text (`text`), nur das Bild (`image`), Text im Bild (`center`) oder das Bild über, unter, rechts oder links vom Text (`top`,`bottom`,`right`,`left`) angezeigt wird. Hier ein kleines Beispiel:

```
#!/usr/bin/env python3
# tk_label.py

from tkinter import *    # Python 2.7 "from Tkinter import *"
from tkinter import ttk  # Python 2.7 "import ttk"

mainWin = Tk()

result = StringVar()
result.set('weigu.lu')
imageLabel_1 = PhotoImage(file='myimage.png')

label_1 = ttk.Label(mainWin, textvariable=result, image=imageLabel_1, compound='top')
label_1.grid()

mainWin.mainloop()
```

Referenz zum `ttk.Label` Widget: https://www.tcl.tk/man/tcl/TkCmd/ttk_label.htm

Aufgabe Tk2:

Teste das Programm mit eigenem Text und Bild.

Entry

Zur Eingabe von Daten kann man das Widget **`ttk.Entry`** nutzen. Es handelt sich hierbei um eine Textzeile mit der ein String übergeben wird. Das Widget erhält also einen String sobald Tasten im Eingabefeld betätigt werden. Der String wird während des Tippens dauernd aktualisiert. Mit dem Parameter "`textvariable=`" gibt man ab welchen Variablen (von der Klasse `StrinVar()`) die Information weitergereicht werden soll. Sobald Änderungen im `Entry` Textfeld auftreten werden diese an die Variable weitergereicht.

```
password = StringVar()
entry_1 = ttk.Entry(mainWin, textvariable=password, width=6, show='*')
entry_1.grid()
entry_1.focus()
```

Mit dem Parameter "`width=`" kann man die Standardlänge des Textfeldes verändern. Mit dem Parameter "`show=`" kann man verhindern, dass der momentane Text angezeigt wird, zum Beispiel bei einer Passwortabfrage. Mit der Methode "`focus()`" wird der Cursor gleich nach dem Starten des Programms ins Eingabefeld gesetzt, ohne dass dieses zuerst angeklickt werden muss.

Um Text oder Bilder neben dem Textfeld zu nutzen muss ein zusätzliches Label Widget genutzt werden.

Referenz zum `ttk.Entry` Widget: https://www.tcl.tk/man/tcl/TkCmd/ttk_entry.htm

Aufgabe Tk3:

Schreibe ein kleines Programm, das mit Hilfe eines Textfensters und dreier Labels das folgende Fenster erzeugt. Während des Tippens soll die letzte Zeile den Inhalt des Textfensters anzeigen.

Button

Im obigen Programm wäre es eventuell wünschenswert das Passwort erst nach dem Tippen anzuzeigen. Eine Methode um dies zu erreichen ist das Betätigen eines Button. Das Widget "`ttk.Button`" dient mehr als die beiden oberen Widgets der Interaktion mit dem Benutzer. Hauptsächlich soll er eine Aktion auslösen d.h. ein Kommando ausführen.

```
button_1 = ttk.Button(mainWin, text='show', command=showPass, width=6)
button_1.grid()
```

Neben vielen neuen Eigenschaften kann der Button aber auch Text und Bilder enthalten. Dieselben Parameter wie beim Label können genutzt werden ("`text=`", "`textvariable=`", "`compound=`", "`image=`", "`width=`").

Im folgenden wird unsere Aufgabe Tk3 um einen Button erweitert, der das Kommando "`showPass`" ausführt. Die Funktion `showPass()` muss sich vor dem Aufruf des Kommando befinden, damit das Programm das Kommando kennt.

In der Funktion wird das Passwort der Variablen des dritten Labels übergeben. Wir benötigen also jetzt zwei StringVar()-Objekte, eine für das Entry-Widget und eine für das Label-Widget. Das es sich bei einem StringVar()-Objekt nicht um eine normale Variable handelt kann für die Zuweisung nicht einfach das Gleichheits-Zeichen benutzt werden, sondern es müssen die Methoden `.get()` und `.set()` verwendet werden, um die Daten der `StringVar()`-Objekte auszutauschen.

```
def showPass():
    passwordClear.set(password.get())
```

Wem diese Zeile nicht geheuer ist, kann natürlich die Zuweisung mit einer lokalen StringVar()-Variablen auch schrittweise durchführen:

```
def showPass():
    temp = StringVar()
    temp = password.get()
    passwordClear.set(temp)
```

Hier das vollständige Programm:

```
#!/usr/bin/env python3
# tk_button.py

from tkinter import *    # Python 2.7 "from Tkinter import *"
from tkinter import ttk  # Python 2.7 "import ttk"

def showPass():
    passwordClear.set(password.get())

mainWin = Tk()

password = StringVar()
passwordClear = StringVar()

label_1 = ttk.Label(mainWin, text='Type your password: ')
label_1.grid()
entry_1 = ttk.Entry(mainWin, textvariable=password, width=6, show='*')
entry_1.grid()
entry_1.focus()
label_2 = ttk.Label(mainWin, text='Your password is: ')
label_2.grid()
label_3 = ttk.Label(mainWin, textvariable=passwordClear)
label_3.grid()
button_1 = ttk.Button(mainWin, text='show', command=showPass, width=6)
button_1.grid()

mainWin.mainloop()
```

Referenz zum `ttk.Button` Widget: https://www.tcl.tk/man/tcl/TkCmd/ttk_button.htm

Aufgabe Tk4:

a) Erweitere das obige Programm um die folgende Zeile:

```
mainWin.bind('<Return>',showPass)
```

Mit dieser Zeile wird die Return- bzw. Enter-Taste mit unserer Funktion `showPass` verbunden, so dass das Passwort auch beim Drücken der Enter-Taste erscheint. Allerdings erhalten wir nun die Fehlermeldung:

```
TypeError: showPass() takes 0 positional arguments but 1 was given
```

Die Zeile der Funktionsdeklaration muss um "`*args`" erweitert werden, damit das Argument angenommen werden kann:

```
def showPass(*args):
```

b) Teste das vollständige Programm!

Frame

Da das Hauptfenster leider nicht zum Ttk-Widget-Set gehört werden wir einen Ttk- Rahmen (`ttk.Frame`) verwenden, um in diesem die Widgets anzuordnen (Dies ist auch vorteilhaft wenn man den ganzen Bildschirm in eine eigene Klasse einpacken möchte).

Das Widget **`ttk.Frame`** wird meist als Container für andere Widgets genutzt um Ordnung im Layout zu schaffen. Normalerweise erhält der Rahmen automatisch seine Größe, durch die Größe der Widgets die er umfasst. Mit "`width=`" und "`height=`" kann die Größe aber auch statisch festgelegt werden. Wird eine Zahl eingegeben, so handelt es sich um Bildschirmpixel. Mit einem angehängten 'c' (15c) kann Höhe bzw. Breite aber auch in Zentimeter (i für inch, p für printer's point) festgelegt werden. Damit die Höhe und die Breite berücksichtigt werden muss aber mit der Methode `grid_propagate(0)` die automatische Anpassung abgeschaltet werden.

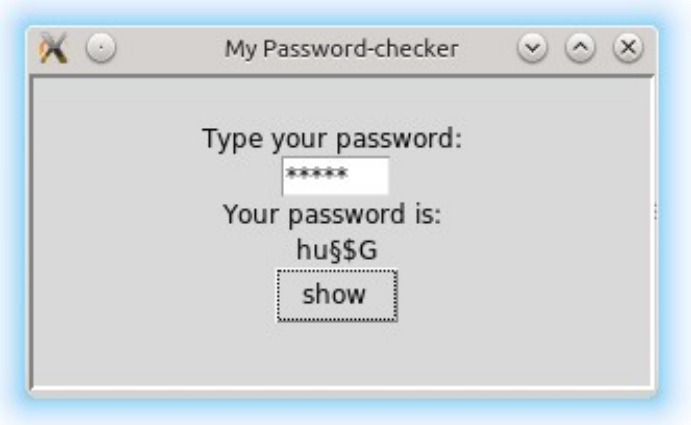
Mit dem Parameter "`padding=`" kann ein Abstand der inneren Widgets zum Rahmen vereinbart werden. Wird eine Zahl eingegeben, so ist der Abstand überall gleich. Mit zwei Zahlen wird der vertikale und der horizontale Abstand definiert. Mit vier Zahlen der Abstand in vier Richtungen (left, top, right, bottom im Uhrzeigersinn).

Natürlich kann der Rahmen auch einen sichtbaren Rand haben. Dies geschieht mit dem Parameter "`borderwidth=`" (default = 0). Mit dem Parameter "`relief=`" kann die Optik des sichtbaren Randes verändert werden. Optionen sind: '`flat`' (default), '`raised`', '`sunken`', '`solid`', '`ridge`' und '`groove`'.

In unser Programm mit Rahmen müssen natürlich die Widgets jetzt "Kinder" des Rahmens sein und nicht mehr des Hauptfensters. Das Programm (Hauptfenster) erhält zusätzlich einen Titel mit der Methode `title()`. Zur Demonstration, und damit der Titel lesbar ist wurde hier mit fester Rahmengröße gearbeitet. Im Normalfall ist es besser die Widgets passen sich der Fenstergröße an, wie wir im nächsten Kapitel sehen werden.

```
#!/usr/bin/env python3
# tk_frame.py

from tkinter import *    # Python 2.7 "from Tkinter import *"
from tkinter import ttk  # Python 2.7 "import ttk"

def showPass(*args):
    passwordClear.set(password.get())

mainWin = Tk()
mainWin.title('My Password-checker')

password = StringVar()
passwordClear = StringVar()

mainFrame = ttk.Frame(mainWin, borderwidth=2, width=300, height=150, relief='groove', padding='80 20
80 20')
mainFrame.grid_propagate(0)
mainFrame.grid()

label_1 = ttk.Label(mainFrame, text='Type your password: ')
label_1.grid()
entry_1 = ttk.Entry(mainFrame, textvariable=password, width=6, show='*')
entry_1.grid()
entry_1.focus()
label_2 = ttk.Label(mainFrame, text='Your password is: ')
label_2.grid()
label_3 = ttk.Label(mainFrame, textvariable=passwordClear)
label_3.grid()
button_1 = ttk.Button(mainFrame, text='show', command=showPass, width=6)
button_1.grid()
```

```
mainWin.bind('<Return>',showPass)

mainWin.mainloop()
```



Referenz zum `ttk.Frame` Widget: https://www.tcl.tk/man/tcl/TkCmd/ttk_frame.htm

Aufgabe Tk5:

a) Entferne die feste Rahmengröße wieder aus dem Programm. Teste das Programm mit unterschiedlichen Reliefs, Randbreiten und Paddings. Wähle eine Kombination die dir gefällt.

b) Ändere dein Programm so um, dass ein Passwort-checker entsteht. Statt der Funktion `showPass()` soll eine neue Funktion `checkPass()` überprüfen wie viele Zeichen das Passwort enthält. Unter 6 Zeichen meldet das Programm dann 'weak', zwischen 6 und 8 Zeichen 'OK' und bei mehr als 8 Zeichen 'strong'. Dazu wird das `StringVar()`-Objekt in einen String geladen (`check = password.get()`), der dann mit der Funktion `len()` überprüft wird. Passe die Namen der Variablen, der Funktion und den Text des Button an.

Der Grid-Layout-Manager

Die Aufgaben des Layout-Manager (Geometrie-Manager) sind recht komplex, da Widgets unterschiedliche Größen haben und Fenster skaliert werden können. In Tk gibt es unterschiedliche Layout Manager. Am intuitivsten einsetzbar ist der Grid-Layout Manager. Mit ihm kann ein Tabellen-Raster (Zeilen und Spalten) erzeugt werden an dem die Widgets ausgerichtet werden. Zeilen und Spalten sind von oben nach unten und rechts nach links durchnummeriert beginnend bei Null.

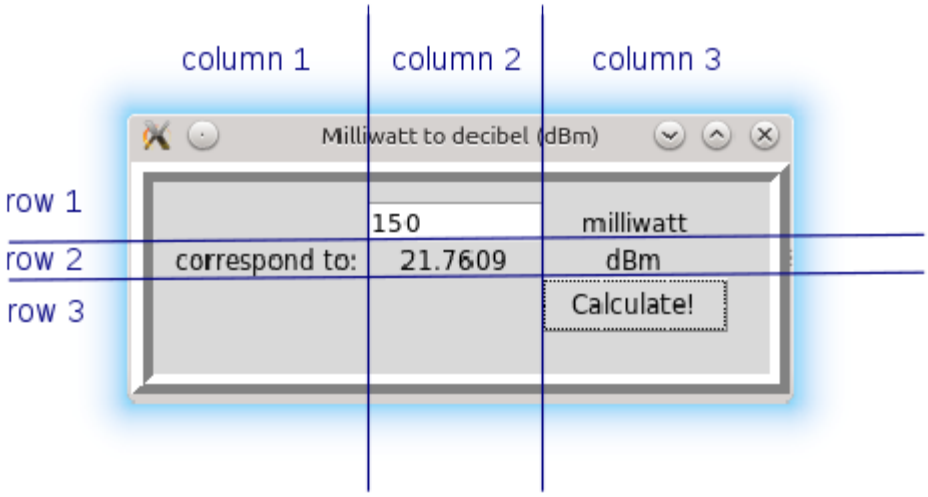
Reihen und Kolonnen

Im folgenden wollen wir ein neues Programm entwickeln. Bevor wir das tun ist es sinnvoll sich zu überlegen, wie das Programm aussehen soll, und die einzelnen Komponenten in

einer Tabellenzelle zuzuordnen. Hier ein Beispiel für unser Programm, das eine Leistung in Milliwatt in Dezibel (bezogen auf ein Milliwatt) umrechnen soll.

$$L_p = 10 \cdot \lg \frac{P}{1\,mW} \rightarrow L_p \cdot \text{in} \cdot \text{dBm}$$

In der ersten Zeile befindet sich ein Entry-Widget und ein Label-Widget. In der zweiten Zeile drei Label-Widgets und in der vierten Zeile das Button-Widget.



Mit dem Erlernten aus dem vorigen Kapitel ist es nicht schwierig das Programm zu erstellen. Neu ist, dass wir der `grid()`-Methode die Parameter "`column=`" und "`row=`" übergeben, und damit die Position der Widgets im Raster festlegen.

Zur Berechnung wird der Inhalt des `StringVar()`-Objekts mit der Methode `get()` abgeholt. Der Inhalt entspricht einer Zeichenkette (string) und muss mit der Funktion `int()` nach Integer konvertiert werden. Wenn Kommastellen erwünscht sind ist die Funktion `float()` zu verwenden. (Das Komma entspricht dabei einem Punkt!) Damit die Logarithmus-Funktion nutzbar ist muss sie aus dem math-Modul importiert werden. Der zweite Parameter ist die Basis des Logarithmus (hier 10). Mit der Funktion round() wird die Stellenzahl hinter dem Komma auf 4 begrenzt. Mit der Funktion `str()` wird das Integer-Resultat schlussendlich wieder in einen String überführt und mit `.set()` dem zweiten `StringVar()`-Objekt übergeben.

Falsche Eingaben werden mit der "`try...except`"-Anweisung abgefangen. Der Programmteil, wo eine Ausnahme zu erwarten ist befindet sich hinter "`try:`". Hinter "`except:`" befindet sich dann der Code der ausgeführt werden soll, falls eine solche Ausnahme auftritt. Da wir in unserem Fall wissen, dass es sich um einen Eingabefehler handeln muss, können wir das mit der "`ValueError`"-Ausnahme (exception) und einer eindeutigen Fehleraussage auch sauber dokumentieren.

Hier das vollständige Programm:

```
#!/usr/bin/env python3
# tk_grid1.py

from tkinter import *    # Python 2.7 "from Tkinter import *"
from tkinter import ttk  # Python 2.7 "import ttk"
from math import log

def calculate(*args):
    try:
        power=int(power_mW.get())
        dBm=round(10*log(power,10),4)
        result_dBm.set(str(dBm))
    except ValueError:
        result_dBm.set('error: entry not valid!')

mainWin = Tk()
mainWin.title('Milliwatt to decibel (dBm)')

power_mW = StringVar()
result_dBm = StringVar()

mainFrame = ttk.Frame(mainWin, borderwidth=10, relief='ridge', padding="20")
mainFrame.grid()

entry_1 = ttk.Entry(mainFrame, textvariable=power_mW, width=10)
entry_1.grid(column=2, row=1)
entry_1.focus()
label_1 = ttk.Label(mainFrame, text='milliwatt')
label_1.grid(column=3, row=1)
label_2 = ttk.Label(mainFrame, text='correspond to: ')
label_2.grid(column=1, row=2)
label_3 = ttk.Label(mainFrame, textvariable=result_dBm)
label_3.grid(column=2, row=2)
label_4 = ttk.Label(mainFrame, text='dBm')
label_4.grid(column=3, row=2)
butt_1 = ttk.Button(mainFrame, text='Calculate!', command=calculate, width=10)
butt_1.grid(column=3, row=3)

mainWin.bind('<Return>',calculate)

mainWin.mainloop()
```

Aufgabe Tk6:

Teste das Programm auf dem Raspberry Pi. Teste auch was bei falscher Eingabe passiert!

Skalierbare Fenster

Die Widgets wurden zwar schön angeordnet, allerdings passen sie sich nicht der Fenstergröße an, falls diese verändert wird. Dazu werden die beiden Methoden `columnconfigure=()` und `rowconfigure=()` benötigt. Sie müssen auf jede Kolonne und jede Reihe des Rahmens angewendet werden. Zusätzlich müssen sie auf das Hauptfenster angewendet werden, da dieses ja verändert wird und der Rahmen sich in diesem befindet.

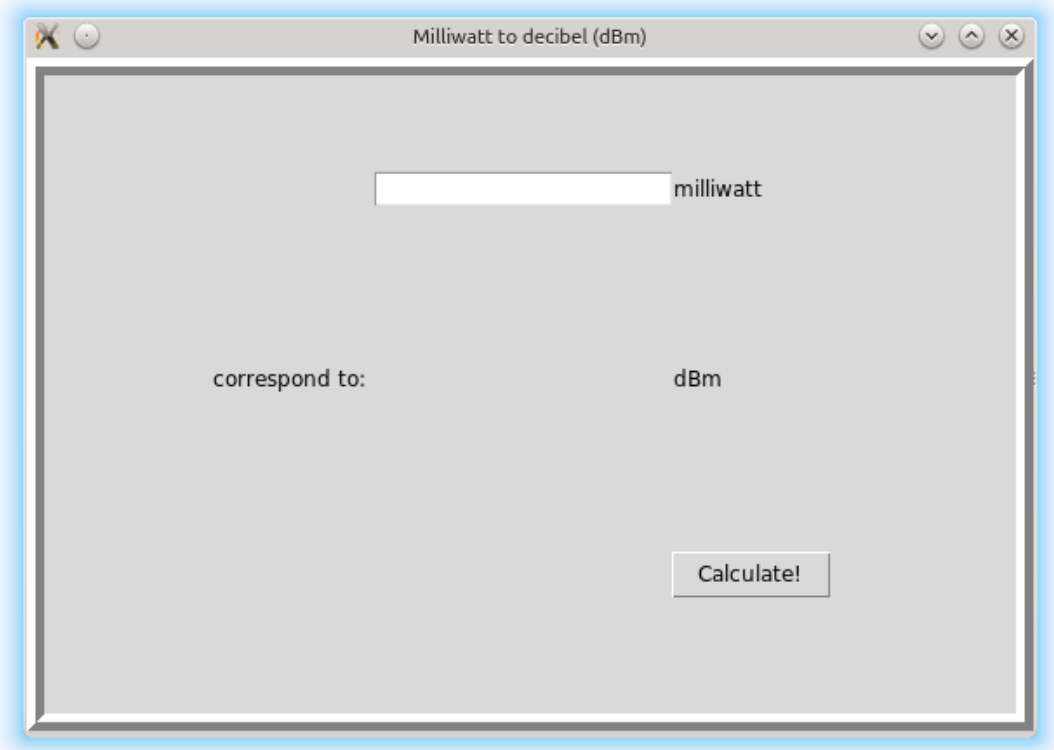
```
mainWin.columnconfigure(0, weight=1)          # numbering beginns with 0 for old Tk widgets
mainWin.rowconfigure(0, weight=1)
...
mainFrame.columnconfigure(1, weight=5)        # numbering beginns with 1 for newer Ttk widgets
mainFrame.columnconfigure(2, weight=5)
mainFrame.columnconfigure(3, weight=5)
mainFrame.rowconfigure(1, weight=1)
mainFrame.rowconfigure(2, weight=1)
mainFrame.rowconfigure(3, weight=1)
```

Der erste Parameter bezeichnet die Reihe bzw. die Kolonne, der zweite Parameter "`weight=`" gibt Gewichtung, wie schnell sich die Widgets bei der Veränderung des Fensters bewegen (Mit `weight=0` bewegen sie sich nicht). Bei den alten Tk-Widgets begann die Nummerierung der Reihen und Kolonnen mit 0 statt 1. Dies gilt hier für unser Hauptfenster!

Zusätzlich muss jedem Widget mitgeteilt werden, zu welchen der vier Kanten es sich hin bewegen soll. Dies wird mit dem Parameter "`sticky=`" der `grid()`-Methode erreicht. Die vier Himmelsrichtungen dienen hier der Ausrichtung (N: Norden, oben, S: Süden, unten, W: Westen, links, E: Osten, rechts). Wird mehr als eine Richtung angegeben müssen Klammern gesetzt werden.

Das veränderte Programm sieht jetzt folgendermaßen aus:

```
#!/usr/bin/env python3
#tk_grid2.py

from tkinter import *    # Python 2.7 "from Tkinter import *"
from tkinter import ttk  # Python 2.7 "import ttk"
from math import log

def calculate(*args):
    try:
        power=int(power_mW.get())
        dBm=round(10*log(power,10),4)
        result_dBm.set(str(dBm))
    except ValueError:
        result_dBm.set('error: entry not valid!')

mainWin = Tk()
mainWin.title('Milliwatt to decibel (dBm)')
mainWin.columnconfigure(0, weight=1)         # numbering beginns with 0 for old Tk widgets
mainWin.rowconfigure(0, weight=1)

power_mW = StringVar()
result_dBm = StringVar()

mainFrame = ttk.Frame(mainWin, borderwidth=10, relief='ridge', padding="20")
mainFrame.grid(column=0, row=0, sticky=(W, N, E, S))
mainFrame.columnconfigure(1, weight=5)       # numbering beginns with 1 for newer Ttk widgets
mainFrame.columnconfigure(2, weight=5)
mainFrame.columnconfigure(3, weight=5)
mainFrame.rowconfigure(1, weight=1)
mainFrame.rowconfigure(2, weight=1)
mainFrame.rowconfigure(3, weight=1)

entry_1 = ttk.Entry(mainFrame, textvariable=power_mW, width=10)
entry_1.grid(column=2, row=1, sticky=(W, E))
entry_1.focus()
label_1 = ttk.Label(mainFrame, text='milliwatt')
label_1.grid(column=3, row=1, sticky=W)
label_2 = ttk.Label(mainFrame, text='correspond to: ')
label_2.grid(column=1, row=2, sticky=E)
label_3 = ttk.Label(mainFrame, textvariable=result_dBm)
label_3.grid(column=2, row=2, sticky=(W, N, E, S))
label_4 = ttk.Label(mainFrame, text='dBm')
label_4.grid(column=3, row=2, sticky=W)
butt_1 = ttk.Button(mainFrame, text='Calculate!', command=calculate, width=10)
butt_1.grid(column=3, row=3, sticky=W)

mainWin.bind('<Return>',calculate)

mainWin.mainloop()
```



Padding und Columnspan

Der Abstand zwischen dem Eingabefeld und der Einheit ist zu klein. Mit Hilfe der Parameter "padx=" und "pady=" kann bei der grid()-Methode ein innerer Abstand zur Tabellenzelle eingehalten werden. Zum Beispiel für den oben erwähnten Abstand:

```
label_1.grid(column=3, row=1, sticky=W, padx=30)
```

In unserem Fall wäre ein Abstand zwischen allen Feldern wünschenswert. Dies lässt sich um Tipparbeit zu sparen auch leicht in einer Schleife mit der grid_configure() Methode erreichen:

```
# Padding
for child in mainFrame.winfo_children():
    child.grid_configure(padx=10, pady=10)
```

Manchmal ist es auch nötig ein Widget über mehrere Felder auszudehnen.Dies kann mit den Parametern "columnspan=" und "rowspan=" der grid()-Methode erfolgen. Als Beispiel soll der Button sich über drei Felder erstrecken (width=10 wurde gelöscht):

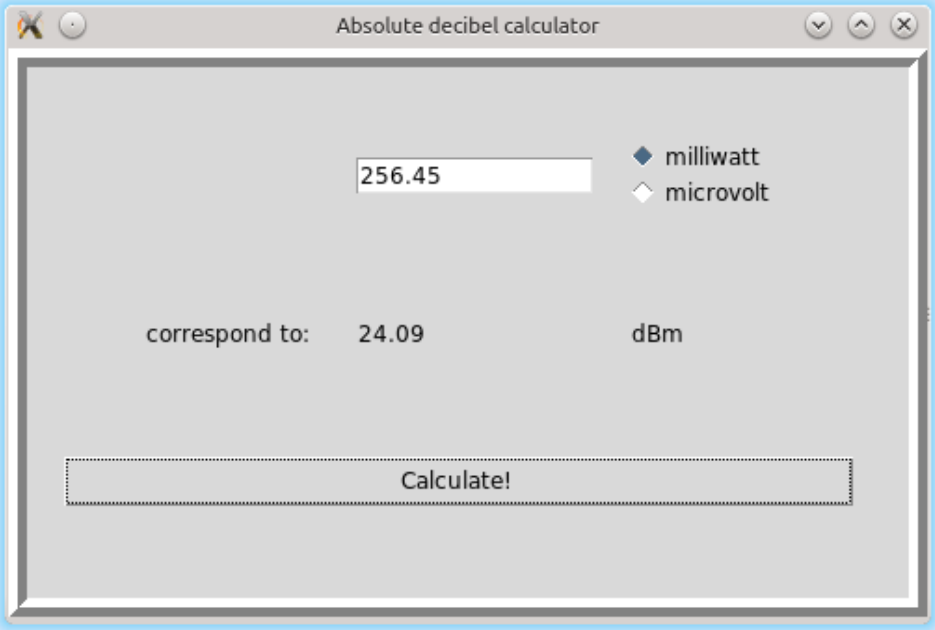
```
butt_1 = ttk.Button(mainFrame, text='Calculate!', command=calculate, width=10)
butt_1.grid(column=1, row=3, sticky=(W, E), columnspan=3)
```

Aufgabe Tk7:

a) Erweitere das Programm indem jede Zelle einen inneren Abstand erhält und der Button sich über 3 Kolonnen erstreckt.

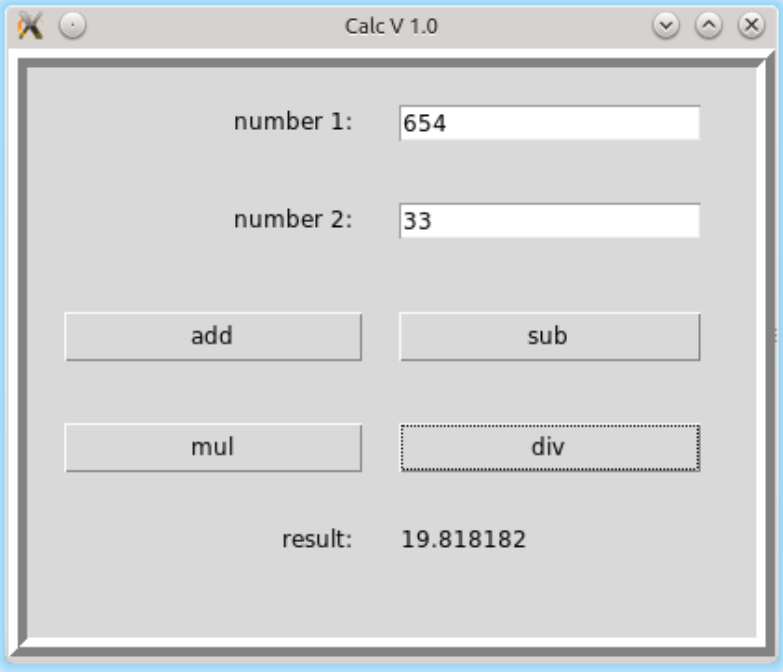
b) Ändere dein Programm so um, dass der erste Label (rechts oben) durch einen Rahmen, der zwei Radiobutton enthält, ersetzt wird. Suche Informationen zum Widget **ttk.Radiobutton** im Netz. Der eine Radiobutton soll eine Berechnung in

dBm (wie bisher, text=’milliwatt’) und der zweite Radiobutton eine Berechnung in dBμV (text='microvolt', in der Formel muss der Multiplikationsfaktor 10 durch 20 ersetzt werden) erlauben. Entsprechend sollen auch der Text des Label mit der Einheit ('dBm' bzw. 'dBμV') geändert werden.



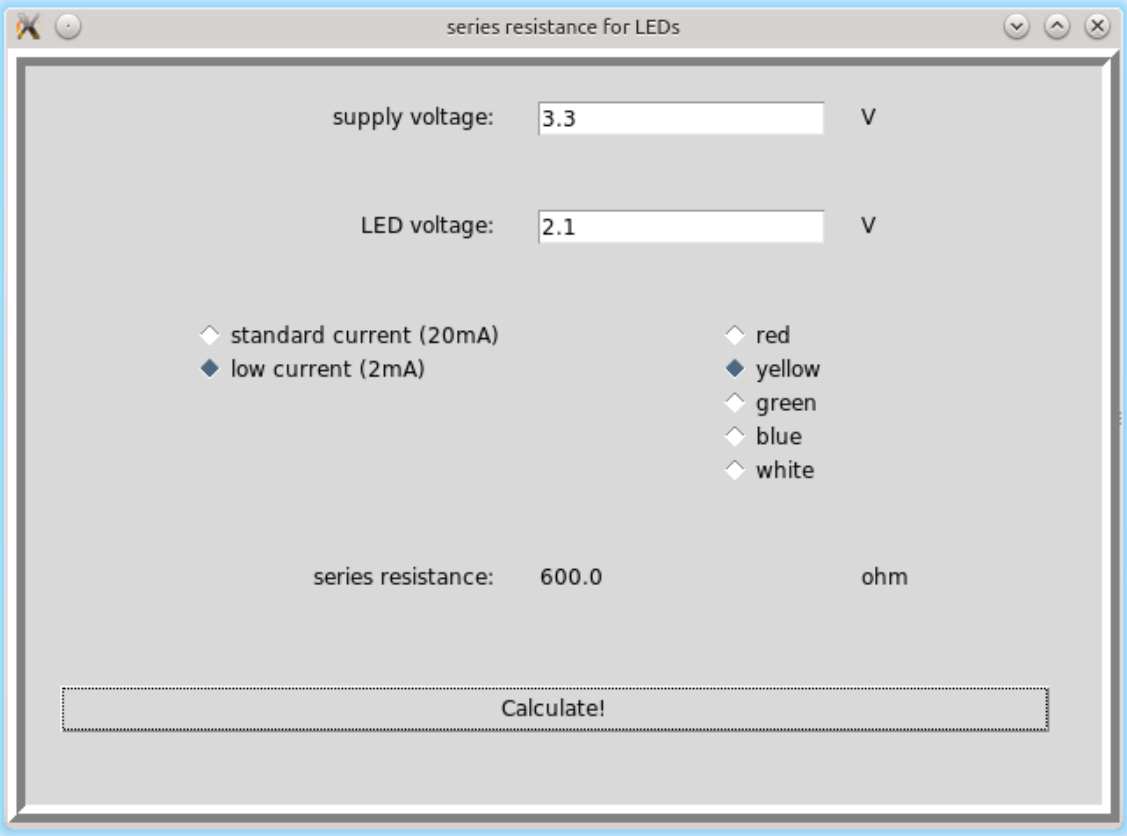
Aufgabe Tk8:

Erstelle ein Programm, das zwei Zahlen mit den Grundrechenarten verknüpfen kann.



Zusatzaufgabe Tk9: (für Fleißige :))

Erstelle ein Programm, das den Vorwiderstand einer LED berechnet. Der Strom wird mit einer Checkbox ausgewählt. Die Spannung an der LED ebenfalls. Sie kann aber wahlweise auch mit einem Entry-Feld manuell eingegeben werden.



Zusatzaufgabe Tk10: (für sehr Fleißige :)))

Programmiere das Spiel TIC TAC TOE. Sobald ein Button gedrückt wird, wird er mit:

```
butt_11.state(["!disabled"])
```

deaktiviert. Eine Zählvariable achtet darauf, dass ein Gleichstand (niemand winnt) erkannt wird. Um das ganze farbiger zu gestalten können "styles" verwendet werden. Eine mögliche Lösung findet man unter: http://www.weigu.lu/c/python/download

- http://www.tutorialspoint.com/python/python_basic_syntax.htm
- http://www.tkdocs.com/tutorial/
- https://docs.python.org/3/library/tkinter.ttk.html
- https://www.tcl.tk/man/tcl/TkCmd/contents.htm
- http://www.python-kurs.eu/python_tkinter.php