www.uni-frankfurt.de

Modul: Programmierung B-PRG
Grundlagen der Programmierung 1

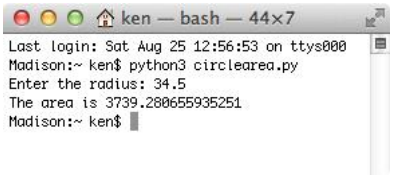
V14 Entwicklung von GUIs (1)

Prof. Dr. Detlef Krömker
Professur für Graphische Datenverarbeitung
Institut für Informatik
Fachbereich Informatik und Mathematik (12)

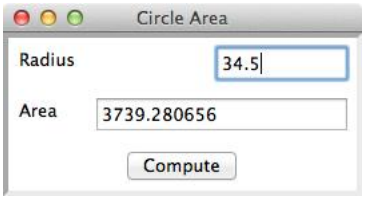
Rückblick

➔ Die **Hälfte des Aufwands** zur Entwicklung eines Softwaresystems fällt typischerweise **für die Benutzungsschnittstelle** an!

CLI



GUI

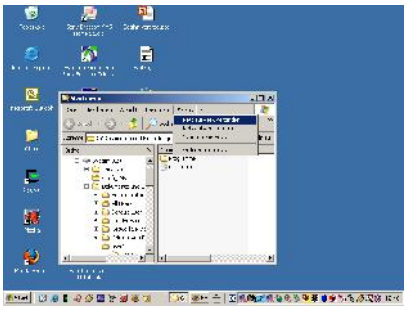


Wir behandeln beide Ausprägungen … heute **GUIs**

GUIs – Das WIMP-Paradigma

Immer noch **grundlegendes Paradigma** zur Gestaltung der Mensch-Maschine-Interaktion (MMI)

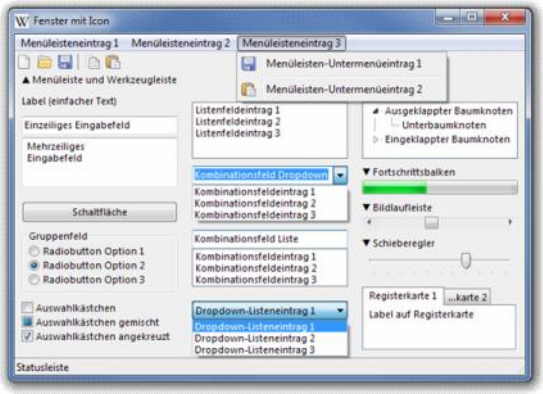
- **WIMP (Windows, Icons, Menues, Pointer)**
- **Werkzeugmetapher (Tool)**
- **Direct Manipulation**



zumindest bei Desktop-Rechnern und Laptops

Entwicklung mit Widgets aus entsprechenden Bibliotheken, bei uns **tkinter**

So sieht das dann (unter Windows 7) aus und wie man das macht, schauen wir uns jetzt an:



Aus: de.wikipedia.org/wiki/Steuerelement

unter anderen Betriebssystemen sieht es etwas anders aus.

Widget

Ein **Steuerelement** (*graphical control element* oder auch ***widget***) ist ein Interaktionselement in einer grafischen Benutzeroberfläche (GUI), beispielsweise eine Schaltfläche oder eine Bildlaufleiste.

Jedes Widget hat verschiedene Eigenschaften (Attribute), wie Größe, Position auf dem Bildschirm, usw.

Inhalt

- tkinter … was ist das?
- Nur die allerwichtigsten Widgets und Methoden:
  - Das root-Window
  - Das Label-Widget
  - Die Layout- (Geometrie-) Manager: pack und grid
  - Das Message-Widget
  - Das Button-Widget
  - Das Entry-Widget
  - Das Menu-Widget
- Vollständige Dialoge
- Zusammenfassung und Ausblick

Tkinter (1)

- **…** ist die Python-Schnittstelle oder das Python-Interface zu **Tk**.
- "Tkinter" ist ein Akronym für "Tk interface".
- Tkinter ist für Python als Modul in der Standard-Bibliothek verfügbar (realisiert als ein "Wrapper"), die TK widgets als Python-Klassen bereitstellen.
- **Tk** wurde als eine GUI-Erweiterung für Tcl (eine Programmier-/Skriptsprache) von John Ousterhout entwickelt. Das erste Release von Tcl erfolgte im Jahre 1991.
- Es wurden Sprachanbindungen für diverse andere Programmiersprachen entwickelt: u.a. Perl, Ruby, Common Lisp, Ada (TASH) und R
- Aktuell ist **Tk version 8.6**

TKinter (2)

Tkinter gibt es für mehrere Betriebssysteme: Je nach Windowmanager und gewählter (natürlichen) Sprache ist die Erscheinung verschieden.

Von: http://www.tkdocs.com/tutorial/install.html#helloworld

Als Beispiel: IDLE ist ein TKinter Programm!

Auf der anderen Seite gibt es natürlich auch viele andere GUI-Tools für Python (1) hier nur Tools mit Python Binding und 'Open Source' von http://martin-thoma.com/gui-programming-with-python/

Name	GTK+	QT	Tk	wxWidgets
Latest version (23.01.2015)	3.14.1	5.4.0	8.6.3	3.0.2
Official Website	gtk.org	qt-project.org	tcl.tk	wxwidgets.org
Initial release	1998	1995	1991	1992
Written in	C	C++	C	C++
Documentation	gtk.org/documentation.php	doc.qt.io	tkdocs.com	wxwidgets.org/docs
Tutorial	developer.gnome.org/gtk-tutorial/stable	qt-project.org/doc/qt-4.8/tutorials.html	tkdocs.com/tutorial	wxwidgets.org/docs/tutorials

Auf der anderen Seite gibt es natürlich auch viele andere GUI-Tools für Python (2) hier nur Tools mit Python Binding und 'Open Source' von http://martin-thoma.com/gui-programming-with-python/

Name	GTK+	Qt	Tk	wxWidgets
Python binding	PyGTK (docs)	PySide (docs), PyQt	Tkinter (docs)	wxPython (docs)
Python 3 support	yes	yes	(yes?) (YES (dk))	yes
Designer	Glade Interface Designer	QtDesigner, QtCreator, QDevelop, Edyuk	SpecTcl	wxGlade
Famous applications	Gnome applications like Inkscape, OTR-Verwaltung	vlc, Virtual Box, KDE applications like K3B, Anki	(I could not find any) IDLE, dk	Code::Blocks FileZilla 0 A.D.

Viele weitere Toolkits unter https://wiki.python.org/moin/GuiProgramming

Welche Widgets gibt es in Tkinter (als Klassen implementiert)?

- **Button**
- Canvas
- Checkbutton
- **Entry**
- **Frame**
- **Label**
- LabelFrame
- Listbox
- **Menue**
- **Menuebutton**
- **Message**
- PanedWindow
- Radiobutton
- Scale
- Scrollbar
- Spinbox
- Text

Tk stellt außerdem drei verschiede Geometrie Manager zur Verfügung:

- place
- **grid**
- **pack**

grün markiert sind die in dieser Vorlesung behandelten Widgets.

Welche Widgets gibt es in Tkinter (als Klassen implementiert)?

- Button
- Canvas
- Checkbutton
- Entry
- Frame
- Label
- LabelFrame
- Listbox
- Menue
- Menuebutton
- Message
- PanedWindow
- Radiobutton
- Scale
- Scrollbar
- Spinbox
- Text

Tkinter hat außerdem ein "themed" (Unter-)modul **ttk.**
Die oben blau markierten Widgests werden durch ttk ersetzt/verändert.

… und hier noch diverse weitere Widgets im (Unter-)Modul **ttk**:

- Combobox
- Notebook
- Progressbar
- Separator

Außerdem stellt Tk folgende (kompletten) Standarddialoge als Popups zur Verfügung:

- `tk_chooseColor` – lässt ein Pop-up-Fenster zur Farbauswahl erscheinen.
- `tk_chooseDirectory` – lässt ein Pop-up-Fenster zur Auswahl eines Verzeichnisses erscheinen.
- `tk_dialog` – ein Pop-up-Fenster in Form eines Dialogfenster
- `tk_getOpenFile` – ein Pop-up-Fenster, dass erlaubt, interaktiv eine Datei zum Öffnen auszuwählen.
- `tk_getSaveFile` – Pop-up-Fenster, dass erlaubt interaktiv eine Datei zum Schreiben auszuwählen.
- **`tk_messageBox`** – Pop-up-Fenster mit Message.
- `tk_popup` – Pop-up-Fenster.
- `toplevel` – erzeugt und verändert Widgets auf höchster Ebene.

Hier steigen wir tief herab und haben dann entsprechend viel Arbeit: Events

- Die Widgets und Standarddialoge nehmen uns viel Arbeit ab, aber erfüllen ggf. nicht alle Wünsche:
- Die Behandlung von elementaren Ereignissen, z.B. Keystrokes, Mousebewegungen, etc. ist mit sogenannten **Events** möglich.
- Ein **Event** ist das Auftreten eines (elementaren) Ereignisses über das Ihr Programm Bescheid wissen soll.
- Ein Event-Handler ist eine Funktion, die in solchen Fällen gerufen wird.
- Ein Event-Handler muss an das Ereignis gebunden (bind) warden
- **…. Aber das führt jetzt zu weit. Sorry.**

… wir erzeugen uns einmal ein Fenster mit tkinter (1)

```
import tkinter as tk  #(1)
root = tk.Tk()  #(2)

root.mainloop()  #(3)
```

(1) Das tkinter-Modul muss importiert werden. Hier In unserem Beispiel importieren wir es mit dem Sclhlüsselwoert `as` als `tk` Hierdurch sparen wir uns Tipparbeit.

In fremden Beispielen (aus dem Internet) finden Sie häufig:

`from tkinter import *`
(Aber Vorsicht!)

… wir erzeugen uns einmal ein Fenster mit tkinter (1)

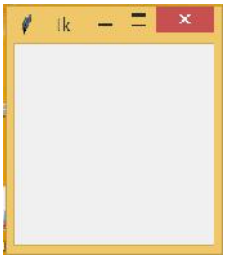
```
import tkinter as tk  #(1)
root = tk.Tk()  #(2)

root.mainloop()  #(3)
```

(2) Zur Initialisierung, müssen wir ein TKinter Root-Widget instanzieren. Dies geschieht mit dem **Aufruf `Tk()`**. Dieses Widget liefert die Titelleiste und die Dekorationen, die der verwendete Fenstermanager zur Verfügung stellt. Das Root-Widget muss erzeugt werden, bevor irgendwelche anderen Widgets benutzt werden können. Es kann **in jeder Anwendung** (= Programm) **nur ein** Root-Widget geben.

Durch diese Anweisung haben wir ein Objjekt der Klasse "Window" instanziert.
Wg. der benutzten Konventionen in der
➔ Objektorientierung, wird `Tk()` groß geschrieben.

Eventloop

```
import tkinter as tk  #(1)
root = tk.Tk()  #(2)

root.mainloop()  #(3)
```



(3) Der Befehl mainloop in root.mainloop() steht für eine **Endlosschleife(!) = Ereignisschleife.** Sonst läuft Ihr Programm nur einmal durch und wartet nicht auf Inputs.

Das durch unser Skript erzeugte Fenster bleibt solange in der Ereignis-Schleife (Event loop), das heißt es wartet auf eine Eingabe durch Tastatur oder Maus)

Das Fenster hat die üblichen Bedienmöglichkeiten: Verschieben, verkleinern, vergrößern.

Beendet wird dieses Event loop dadurch, dass wir das Fensterr schließen (Kreuz).

Noch eine Kleinigkeit:

Der Name des Fenster ist default-mäßig tk. Das lässt sich ändern durch:

```
root.title('Mein EPR Window')
```

Beim pop-up sehen wir dann

Unser erstes Widget: Label

```
import tkinter as tk  #(1)
root = tk.Tk()  #(2)

w = tk.Label(root, text = \
"Hallo EPR-TeilnehmerInnen")  #(4)
w.pack()  #(5)


root.mainloop()  #(3)
```

(4) Bei dieser eingefügten Zeile geht es um die Instanzierung des **Label Widget**. Der erste Parameter der Label-Methode enthält das **Eltern-Widget**, in unserem Fall "root". Unser Label-Widget ist also ein Kind des Root-Widget. **Dem Schlüsselwort-Parameter "text"** wird der darzustellende String zugewiesen.

Das Label-widget kann sowohl Texte als auch Bilder enthalten – später.

(5) Gibt an, welcher Layout-Manager (**Geometrie-Manager**) verwendet werden soll.

Geometrie Manager

- Geometrie-Manager haben verschiedene Aufgaben:
  - Sie bestimmen die Position und Größe des Widgets innerhalb des Fensters und ordnen die Widgets auf dem Bildschirm an;
  - Sie registrieren Widgets bei dem zugrundeliegenden Fenstermanager;
  - Sie managen die Darstellung der Widgets auf dem Bildschirm
- Widgets können aber auch dem Geometrie-Manager Angaben zu Größe und Position geben, aber dies sind nur Wünsche. Der Geometrie-Manager entscheidet bezüglich der Poistionierung und die Größendarstellung.
- Der tk-Geometrie Manager bietet drei Vorgehensweisen (drei Methoden) an:

Geometrie Manager, Methode `.pack`

- **`.pack`** ist der am einfachsten zu benutzende Geometrie-Manager von Tkinter.
- Man muss nicht präzise angeben, wo ein Widget auf dem Bildschirm erscheinen soll, sondern die Details der Darstellung (Position, Größe, etc.) werden von pack automatisch bestimmt,
- Weil die pack-Methode so einfach zu benutzen ist, ist dieser Layout-Manager ist der Kontrolleinfluss des Programmierers im Vergleich zu der grid- und der place-Methode stark eingeschränkt (und manchmal überraschend gut, aber im Detail dann schwierig zu steuern)
- Für **sehr einfache Anwendungen** ist dieser Manager aber meistens die beste Wahl.

Ein weiteres Beispiel mit der Methode pack

```
import tkinter as tk
root = tk.Tk()

w1 = tk.Label(root, text = "Red Moon", bg = "red", fg = "white")
w1.pack()
w2 = tk.Label(root, text = "Green Grass", bg = "green", fg = "black")
w2.pack()
w3 = tk.Label(root, text = "Blue Sky", bg = "blue", fg = "white")
w3.pack()

root.mainloop()
```

Das `root.mainloop()`steht am Ende! - Wir müssen also den Geometrie-Manager als Methode .pack hinzufügen. Theoretisch könnten Sie verschiedene Manager mischen. Don't do that!

Ein weiteres Beispiel mit Label und pack und dem Parameter `fill`

```
import tkinter as tk
root = tk.Tk()

w1 = tk.Label(root, text = "Red Moon", bg = "red", fg = "white")
w1.pack(fill = tk.X)
w2 = tk.Label(root, text = "Green Grass", bg = "green", fg = "black"
w2.pack(fill = tk.X)
w3 = tk.Label(root, text = "Blue Sky", bg = "blue", fg = "white")
w3.pack(fill = tk.X)

root.mainloop()
```

`tk.X` ist die Konstante X, die in tk definiert ist!

Weitere Parameter der methode pack: `padx`

```
import tkinter as tk
root = tk.Tk()

w1 = tk.Label(root, text = "Red Moon", bg = "red", fg = "white")
w1.pack(fill = tk.X, padx = 10)
w2 = tk.Label(root, text = "Green Grass", bg = "green", fg = "black"
w2.pack(fill = tk.X, padx = 10)
w3 = tk.Label(root, text = "Blue Sky", bg = "blue", fg = "white")
w3.pack(fill = tk.X, padx = 10)

root.mainloop()
```

`fill =` kann auch sein: NONE (default), Y (fill vertically), or BOTH

"padding"

Padding allgemein heißt "polstern": Gemeint ist ein Leerraum relativ zum eigenen Elementrand, also z.B. zwischen dem Text eines Elements und dem Rand dieses Elements.

Der Geometrie-Manager pack () kennt neben dem `padx =` drei weitere padding-Methoden, zusätzlich:

`pady =`
Externes Padding
Vertikal

`ipadx =`
Internes Padding
horizontal

`ipady =`
Internes Padding
vertikal

Widgets nebeneinander plazieren, Parameter `side =`

```
import tkinter as tk
root = tk.Tk()

w1 = tk.Label(root, text = "Red Moon", bg = "red", fg = "white")
w1.pack(fill = tk.X, padx = 10, side = tk.LEFT)
w2 = tk.Label(root, text = "Green Grass", bg = "green", fg = "black"
w2.pack(fill = tk.X, padx = 10, side = tk.LEFT)
w3 = tk.Label(root, text = "Blue Sky", bg = "blue", fg = "white")
w3.pack(fill = tk.X, pady = 10, side = tk.LEFT )

root.mainloop()
```

Achtung: bei w3 ist pady gesetzt.
`side` kann auch sein: RIGHT oder TOP oder BOTTOM
Dies sind Konstanten des tkinter. Aufpassen.

Geometrie-Manager place

- Der pack-Manager war historisch der erste Geometrie-Manager.
- Der Pack-Manager hat noch einige weitere Parameter (Optionen), siehe z.B.

http://effbot.org/tkinterbook/pack.htm#Tkinter.Pack.pack_slaves-method

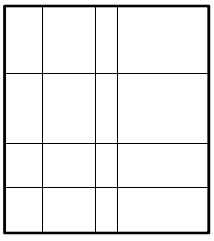
- Der Algorithmus des pack-Managers ist nicht immer leicht zu verstehen und es kann schwierig sein ein existierendes Design zu ändern.
- Für nichttriviale Layouts benutzt man lieber den **Grid-Manager (**im Jahr 1996 als Alternative zu pack eingeführt).

.

Geometrie Manager, Methode `.grid` (1)

- Der Grid-Geometrie-Manager platziert die Widgets in einer 2-dimensionalen Tabelle, die in Reihen und Spalten angeordnet ist.
- Die Position eines Widgets wird durch einen row und einen column-Wert bestimmt.
- Widgets mit der selben column-Zahl und verschiedenen row-Zahlen werden übereinander angeordnet.
- Entsprechend werden Widgets mit der selben row-Zahl und verschiedenen column-Zahlen in der selben Zeile platziert, d.h. sie stehen nebeneinander.
- Die Größe der Zellen bestimmt der `.grid` Algorithmus.

Das Grid

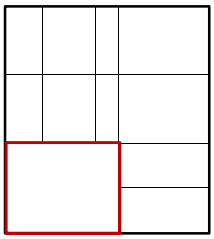


Beispiel `grid`-Manager

```
import tkinter as tk

root = tk.Tk()
colours = ['red', 'green', 'orange', 'white', 'yellow', 'blue']

r = 0
for c in colours:
    tk.Label(text=c).grid(row=r, column=0)
    tk.Label(bg=c, width = 10).grid(row=r, column=1)
    r += 1

root.mainloop()
```

`.grid(Optionen)` Hier nur die wichtigsten!

Häufiger genutzte Optionen sind:
`columnspan = x`, x ist Integer > 1 fast x Spalten zusammen
`cowspan = y`, y ist Integer > 1 fast y Zeilen zusammen



Beispiel: Das rot-umrandeten Rechteck ist

```
.grid(row=2, rowspan=2, columnspan =3
# column=0 ist default
```

`Sticky =`
bestimmt, wo das Widget plaziert wird, wenn die Zelle größer ist als das Widget (an welcher Kante):
Folgende Konstanten gibt es: **N**, **S**, **E**, únd **W** oder **NE**, **NW, SW** und **SE.**
Achtung: Diese Konstanten sind nur in TKinter definiert, also z.B. `tk.N`
Kombinationen der Art `tk.N+tk,E+tk.S` sind möglich.

Geometry (Layout) Manager

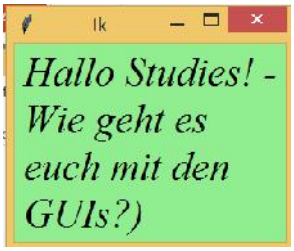
- tkinter hat neben `.pack()` und `.grid()` noch einen weiteren Geometry Manager `.place()`
- `.place()` erlaubt es Widgets pixelgenau zu platzieren (absolut oder relativ zueinander).
- Erfordert aber dafür auch pixelgenaues Planen, was man dann unbedingt wieder auf einem Raster (ähnlich zu Grid) aufbaut.
- Die allermeisten Programmieraufgaben können mit `.grid()` erledigt werden.
- Alle Manager haben weitere Parameter - Denken Sie z.B. an `help(tkinter.Grid).` Achtung **Grid** wird als Klasse groß geschreiben.

Das Widget `Message()`

- `Message()` wird manchmal auch Nachrichten-Box genannt, ist in seiner Funktionalität dem Label-Widget ähnlich, aber es ist flexibler, was die Darstellung von Text betrifft.
- So kann zum Beispiel die Schriftart (englisch font) geändert werden, (beim Label-Widget immer gleich).
- Aber Achtung: Man kann für ein Nachrichten-Element eine bestimmte Schriftart wählen, kann diese dann aber nicht mehr innerhalb dieses Elementes ändern.
- Der Text kann sich über mehrere Zeilen erstrecken. Der Umbruch in die verschiedenen Zeilen erfolgt automatisch.

Message() Beispiel

```
import tkinter as tk

master = tk.Tk()

beispiel_text = "Hallo! \
Wie geht es euch mit den GUIs?)"

msg = tk.Message(master, \
text = beispiel_text)
msg.config(bg='lightgreen', \
font=('times', 24, 'italic'))
msg.pack( )

master.mainloop( )
```



Die Parameter auf vier Blicke

von der Seite http://www.python-kurs.eu/tkinter_message_widget.php

Option	Bedeutung
anchor	Die Position, an der der Text im Message-Widget platziert wird: N (Nord), NE (Nord-Osten), E (Osten), SE (Süd-Osten), S (Süden), SW (Süd-Westen), W (Westen), NW (Nord-Westen), or CENTER (Zentriert). The Default is CENTER (Zentrum).
aspect	Abbildungsverhältnis (Aspect ratio). Es wird als Breiten-Höhenverhältnis in Prozent angegeben. Voreingestellt ist der Wert 150, d.h. das Fenster wird 50 % breiter als hoch sein. Falls die Breite explizit gesetzt wird, wird dieser Wert ignoriert.
background	Diese Option definiert die Hintergrundfarbe des Benachrichtigungsfensters. Die Grundeinstellung ist systemabhängig.
bg	Abkürzung für background

Die Parameter auf vier Blicke

von der Seite http://www.python-kurs.eu/tkinter_message_widget.php

Option	Bedeutung
borderwidth	Die Randbreite. Sie ist auf 2 voreingestellt.
bd	Abkürzung für borderwidth.
cursor	Die Art des Mauszeigers, der erscheint, wenn man die Maus über das Widget bewegt.
font	Die Schriftart. Die Grundeinstellung ist systemspezifisch.
foreground	Die Textfarbe. Die Grundeinstellung ist systemspezifisch.
fg	wie foreground.
highlight \ background	Zusammen mit highlightcolor und highlightthickness, definiert diese Option wie die Highlight-Region darzustellen ist.

Die Parameter auf vier Blicke

von der Seite http://www.python-kurs.eu/tkinter_message_widget.php

Option	Bedeutung
highlight\ thickness	Siehe highlightbackground.
justify	Definiert wie mehrzeiliger Text anzuordnen ist: LEFT, RIGHT, oder CENTER. Um den Text innerhalb des Widgets zu platzieren, benutzt man die anchor-Option. Die Grundeinstellung ist LEFT.
padx	Horizontales Padding. Grundeinstellung ist -1 (kein padding).
pady	Vertikales Padding. Grundeinstellung ist -1 (kein padding).
relief	Randverzierung. Grundeinstellung ist FLAT. Andere mögliche Werte: SUNKEN, RAISED, GROOVE und RIDGE.
takefocus	Ist dieser Wert auf true gesetzt, akzeptiert das Widget den Eingebefokus. Die Grundeinstellung steht auf false.

Die Parameter auf vier Blicke

von der Seite http://www.python-kurs.eu/tkinter_message_widget.php

Option	Bedeutung
text	Der Text des Widgets. Zeilenenden werden automatisch eingefügt, falls notwendig um das eingestellte Abbildungsverhältnis /siehe aspect) zu gewährleisten.
textvariable	Eine Tkinter-Variable wird mit dem Text assoziiert. Wird diese Variable verändert, wird automatisch die Nachrichten-Box aktualisiert.
width	Die Breite des Widgets in Zeicheneinheiten. Eine passende Breite basierend auf dem eingestellten Abbildungsverhältnis (siehe aspect) wird automatisch eingestellt, falls diese Option nicht eingestellt wird.

Und denken Sie immer an:

```
import tkinter as tk
help(tk.Message)
```


Das Widget `Button()`

- Klickt der User auf die Schaltfläche (Button), führt dies i.d.R. zur Ausführung einer bestimmten Aktion.
- Buttons können ebenso wie Labels Bilder und Text enthalten. Der Text eines Buttons kann nur in einer einzigen Schriftart dargestellt werden aber sich ggf. über mehrere Zeilen erstrecken ➔ \n.
- Mit einem Button kann eine Python-Funktion oder Methode assoziiert werden. Diese Funktion oder Methode wird ausgeführt, sobald die Schaltfläche gedrückt wurde.

Button() Beispiel

```
import tkinter as tk

master = tk.Tk()

def callback():
    print ("clicked!")

b = tk.Button(text = "Click me", command = callback)
b.pack()

master.mainloop()
```


```
====== RESTART: C:/Users/kroemker/Desktop… ======
clicked!
clicked!
```


Button() Beispiel -- etwas erweitert

```
import tkinter as tk

master = tk.Tk()


def callback():
    print ("clicked!")

b_1 = tk.Button(master, text = "Click me", command = callback)
b_1.pack()

b_2 = tk.Button(master, text="QUIT", fg="red", command=master.quit)
b_2.pack()

master.mainloop()
```

Das Widget Entry (Eingabefelder)

- Ein Entry-Widget (Eingabefeld) kann beliebigen Text, in einem **einzeiligen** Eingabefeld vom Benutzer einer Applikation eingegeben lassen.
- Falls der Benutzer einen Text eingibt, der mehr Zeichen enthält, als dem Raum des Eingabefeldes entspricht, wird der Text verschoben. Das bedeutet, dass die Anfangsbuchstaben des Strings in der Darstellung verschwinden. Mit den Pfeiltasten kann man wieder den Anfang sehen, indem man nach links geht.
- Möchte man mehr als eine Zeile Text eingeben, so ist dieses Widget ungeeignet. Man sollte statt dessen ein Text-Widget benutzen.

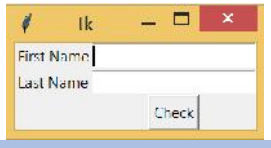
`Entry()` Beispiel (1)

```
import tkinter as tk

master = tk.Tk()
tk.Label(master, text="Vorname:").grid(row=0)
tk.Label(master, text="Nachname:").grid(row=1)

e1 = tk.Entry(master)
e2 = tk.Entry(master)


e1.grid(row=0, column=1)
e2.grid(row=1, column=1)

master.mainloop( )
```

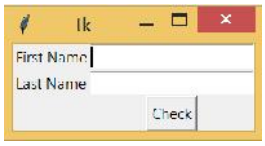
`Entry()` Beispiel (2)

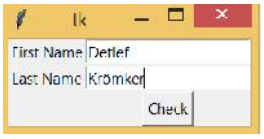
```
"""Entry Example"""

import tkinter as tk

def names():
    """Read in Names and print them"""
    first_name = e1.get()
    last_name = e2.get()
    print(first_name, last_name)

master = tk.Tk()
tk.Label(master, text = "First Name").grid(row = 0, column = 0)
tk.Label(master, text = "Last Name").grid(row = 1, column = 0)
e1 = tk.Entry(master)
e2 = tk.Entry(master)
button = tk.Button(master, text = "Check", command = names)

e1.grid(row = 0, column = 1)
e2.grid(row = 1, column = 1)
button.grid(row = 2, column = 1)

master.mainloop()
```

Bitte achten Sie auf die Bearbeitungsabfolge. names() wird durch das Drücken des Check-button aufgerufen (call-back).



So reagiert das Demo-Programm






```
======= RESTART: C:\Users\kroemker\Desktop\PRG1-EPR-2015\test_Entry.py =======
Detlef Krömker
```

Zusammenfassung

- Dies waren die ersten Schritte zur Entwicklung von GUIs – mehr folgt am Freitag.
- Mehr Infos zu Tkinter finden Sie hier:
- Es bieten sich an: Das "TKinter-Tutorial" von Rolf Klein:
  http://www.python-kurs.eu/python_tkinter.php (deutsch)
  dies war zum Teil auch die Vorlage für diese Vorlesung (ist aber leider nicht vollständig!
- Die "Tkinter 8.5 reference: a GUI for Python" von John W. Shipman.
  http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html
  (sieht sehr vollständig aus).

Ausblick

… und weiter geht es am **Freitag, den 8. 12. 2016.**

Und … weiterhin viel Erfolg