

## 8 Graphische Benutzeroberfläche

In diesem Kapitel werden wir uns mit GUI-Programmen beschäftigen. GUI steht für Graphical User Interface. Es handelt sich also um Programme, welche nicht nur mit Texteingabe und -ausgabe arbeiten, sondern auch eine graphische Benutzeroberfläche haben. In anderen Worten einfach Programme, so wie wir uns das heutzutage gewohnt sind. Wir benutzen dazu das tkinter-Modul, welches in Python integriert ist.

Das Problem bei graphischen Oberflächen ist, dass wir das Programm nicht mehr einfach von oben nach unten durcharbeiten können. Wir haben ja keine Kontrolle darüber, auf welche Schaltfläche der Benutzer zu welchem Zeitpunkt gerade klickt. Darum benutzen wir für graphische Oberflächen sogenanntes ereignisbasiertes Programmieren. Dies bedeutet, dass das Programm nicht linear durchlaufen wird, sondern der Programmablauf durch bestimmte Ereignisse (bei uns werden das Klicks und Eingaben des Benutzers sein) beeinflusst wird.

Für unsere Programme heißt das, dass sie sich meistens in einer Endlosschleife befinden, welche nur dafür zuständig ist, darauf zu warten, dass der Benutzer etwas anklickt. Sobald dies geschieht (also ein Ereignis eintrifft) wird ein bestimmter Teil des Programms ausgeführt, welcher für die Aktion des Benutzers zuständig ist. Sobald dies geschehen ist, geht das Programm wieder in die Endlosschleife zurück und wartet auf weitere Aktionen. Diese Schleife heißt in tkinter `mainloop()`. Im Kapitel Einführung ins Programmieren bei der letzten Aufgabe haben wir bereits ein erstes GUI-Beispiel gesehen.

Im folgenden werden wir Stück für Stück ein GUI erstellen. Beginnen werden wir mit einem einfachen Fenster und dann kontinuierlich weitere Komponenten kennenlernen.

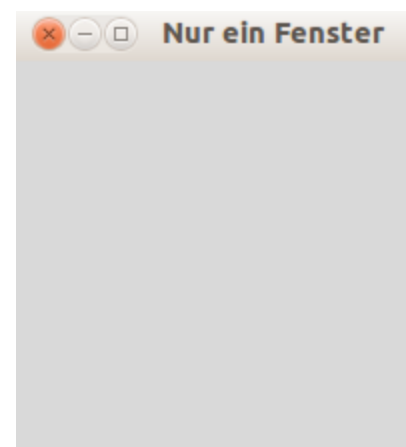
### 8.1 Fenster

Um eine graphische Benutzeroberfläche zu gestalten, brauchen wir als allererstes ein Fenster. Um ein solches zu erstellen benutzen wir, wie bereits erwähnt, das tkinter-Modul von Python, welches uns das nötige Werkzeug zur Verfügung stellt. Dies geschieht wie gewohnt mit dem `from-import` Befehl:

```
from tkinter import *
```

Nun erstellen wir mit `Tk()` ein Fenster:

```
1 from tkinter import *
2
3 # Ein Fenster erstellen
4 fenster = Tk()
5 # Den Fenstertitle erstellen
6 fenster.title("Nur ein Fenster")
7
8 # In der Ereignisschleife auf Eingabe des Benutzers warten.
9 fenster.mainloop()
```



Das ist natürlich noch nicht so interessant, denn dieses Fenster wartet nur, bis es geschlossen wird. Aber nun können wir beginnen, dieses Fenster mit weiteren Komponenten zu bepacken.

### Bemerkung

Je nach Betriebssystem, welches man benutzt, wird das Fenster anders aussehen. Das obige Programm wurde auf einem Rechner mit Ubuntu/Linux ausgeführt und hat somit ein Fenster mit obigem Design erstellt. Auf einem Windows oder OS X Rechner wird es anders aussehen. Dies wird auch bei den folgenden Themen so sein und bleiben.

## 8.2 Buttons und Labels

Nun fügen wir unserem Fenster einige neue Elemente hinzu. Wir beginnen hier mit zwei Buttons und einem Label. Ein Label bietet uns die Möglichkeit einen Text auf dem Fenster anzeigen zu lassen, welcher dem Benutzer z.B. Informationen oder Anweisungen mitteilen kann. Der erste Button soll beim Draufklicken den Infotext ändern und der zweite Button ist ein klassischer Beenden-Button, der das Programm beendet. Solche Elemente werden folgendermaßen erstellt:

```
my_button = Button(fenster, option=value, ... )
my_label = Label(fenster, option=value, ... )
```

Das erste Argument definiert, welchem Fenster die jeweilige Komponente hinzugefügt wird. Mit den darauf folgenden Argumenten können wir unsere Komponente wie gewünscht einstellen, z.B. Größe, Text, Farbe, Textfarbe.<sup>12</sup>

### Achtung


Wir benutzen hier Python 3 und da gibt es manchmal kleine Unterschiede zu Python 2.7. Zum Beispiel:

```
# Python 3
from tkinter import *

# Python 2.7
from Tkinter import *
```

Wichtig für das Button-Objekt ist die command-Option. Ist diese nicht vorhanden, so passiert beim Draufklicken auf den Button gar nichts. Also definieren wir mit def eine Funktion, welche dem Button mit der command-Option als Funktion für das Klick-Ereignis übergeben wird. Mit der Funktion pack() können wir die einzelnen Komponenten dem Fenster übergeben.

<sup>12</sup>Für diese und auch alle weiteren Komponenten im Verlauf des Tutorials gibt es jeweils eine ganze Menge Optionen einzustellen. In den Beispielen werden jedoch nur ein paar gezeigt. Ein vollständigere Auflistung der verschiedenen Optionen der jeweiligen tkinter-Komponente findet sich z.B. auf [http://www.tutorialspoint.com/python/python\\_gui\\_programming.htm](http://www.tutorialspoint.com/python/python_gui_programming.htm)

	Name:	Klasse:	Datum:
	BS Scripte	BFS12 Python - 8	Imr

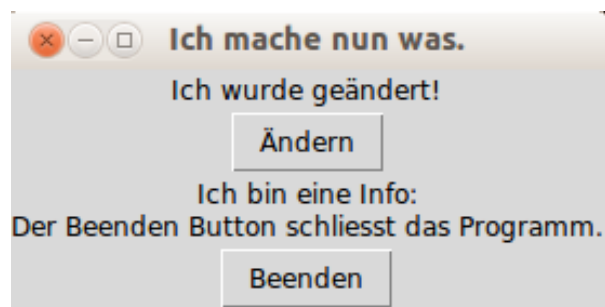
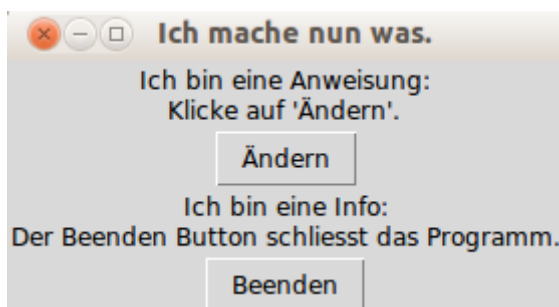
Unser Programm sieht dann folgendermassen aus:

```

1  from tkinter import *
2
3  # Die folgende Funktion soll ausgeführt werden, wenn
4  # der Benutzer den Button anklickt
5  def button_action():
6      anweisungs_label.config(text="Ich wurde geändert!")
7
8
9  # Ein Fenster erstellen
10 fenster = Tk()
11 # Den Fenstertitle erstellen
12 fenster.title("Ich mache nun was.")
13
14 # Label und Buttons erstellen.
15 change_button = Button(fenster, text="Ändern", command=button_action)
16 exit_button = Button(fenster, text="Beenden", command=fenster.quit)
17
18 anweisungs_label = Label(fenster, text="Ich bin eine Anweisung:\n\
19 Klicke auf 'Ändern'.")
20
21 info_label = Label(fenster, text="Ich bin eine Info:\n\
22 Der Beenden Button schliesst das Programm.")
23
24 # Nun fügen wir die Komponenten unserem Fenster
25 # in der gewünschten Reihenfolge hinzu.
26 anweisungs_label.pack()
27 change_button.pack()
28 info_label.pack()
29 exit_button.pack()
30
31 # In der Ereignisschleife auf Eingabe des Benutzers warten.
32 fenster.mainloop()

```

Beim Ausführen erhalten wir das erste Bild und nach dem Klick auf den Ändern-Button das zweite:



Bei einem Klick auf den Beenden-Button wird das Fenster und somit auch das Programm beendet.

## 8.3 Geometrie-Manager

Wir haben oben gesehen, dass mit der Funktion `pack()` die Komponenten dem Fenster hinzugefügt wurden. Doch wie sollen die einzelnen Komponenten auf dem Fenster angeordnet werden? Für solche Angelegenheiten bietet `tkinter` drei verschiedene Geometrie-Manager an:

- `pack`
- `grid`
- `place`

Die drei Layout-Manager `pack`, `grid` und `place` ordnen die verschiedenen Komponenten auf dem Fenster an, jeder auf seine Weise. Sie sollten jedoch nie im gleichen Fenster gemischt werden. Wie sie genau funktionieren und was die Unterschiede zwischen diesen drei Layout-Managern sind, sehen wir gleich.

### 8.3.1 `pack`

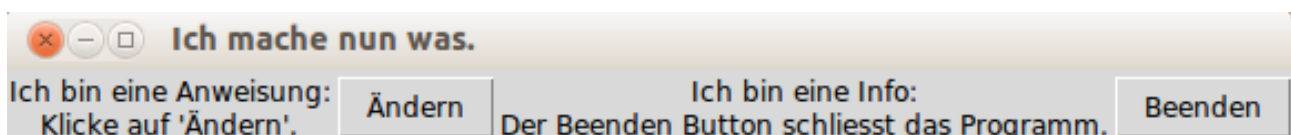
`Pack` ist der am einfachsten zu benutzende der drei Geometrie-Manager. Statt dass man präzise erklären muss, wo eine Komponente auf dem Bildschirm erscheinen soll, werden sie relativ zueinander positioniert. Die Details werden von `pack` automatisch bestimmt. Man kann nur wenig selber bestimmen und ist deshalb in seinen Möglichkeiten im Vergleich zu den anderen Geometrie-Managern eingeschränkt. Hier eine Möglichkeit:

Option:	Beschreibung:
<code>side</code>	Bestimmt auf welcher Seite die Komponente gepackt werden soll: TOP (default), BOTTOM, LEFT, or RIGHT.

Auf das obige Beispiel angewandt, können wir nun unsere Buttons und Labels nebeneinander statt untereinander platzieren:

```
# Die Komponenten nebeneinander platzieren
anweisungs_label.pack(side=LEFT)
change_button.pack(side=LEFT)
info_label.pack(side=LEFT)
exit_button.pack(side=LEFT)
```

Die führt zu folgendem Layout:



Möchte man die Komponenten aber nur etwas anders platzieren, wird es mit dem `pack`-Manager schon recht mühsam. Da ist der `grid`-Manager um einiges angenehmer.

### 8.3.2 grid

Der grid-Geometrie-Manager platziert die Komponenten in einer 2-dimensionalen Tabelle, die in Reihen und Spalten angeordnet ist. Die Position einer Komponente wird durch einen row und einen column-Wert bestimmt. Komponenten mit der selben column-Zahl und verschiedenen row-Zahlen werden übereinander angeordnet. Entsprechend werden Komponenten mit der selben row-Zahl und verschiedenen column-Zahlen in der selben Zeile platziert, d.h. sie stehen nebeneinander, also rechts und links voneinander.

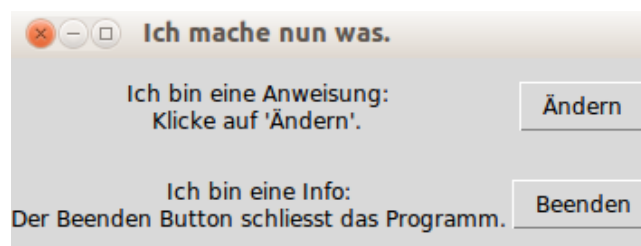
Mit der grid-Methode übergibt man den row- und den column-Wert, wo die Komponente platziert werden soll. Die Grösse braucht nicht definiert zu werden, da der Grid-Manager automatisch die Ausdehnungen für die benutzten Komponenten berechnet.

Option:	Beschreibung:
row	Bestimmt in welcher Zeile man die Komponente setzen möchte.
column	Bestimmt in welcher Spalte man die Komponente setzen möchte.
padx	Diese Option kann man gebrauchen, wenn man in der Horizontalen noch zusätzlich etwas Abstand an die jeweilige Komponente einbauen möchte.
pady	Analog wie padx, einfach in der Vertikalen

Mit dem Grid-Manager lässt sich nun einfach definieren, wo man die einzelnen Komponenten platzieren möchte.

```
# Label und entsprechender Button nebeneinander
# mit etwas Abstand zur anderen Label-Button-Gruppe
anweisungs_label.grid(row=0, column=0, pady = 20)
change_button.grid(row=0, column=1, pady = 20)
info_label.grid(row=1, column=0)
exit_button.grid(row=1, column=1)
```

Und dies sieht dann so aus:



### 8.3.3 place

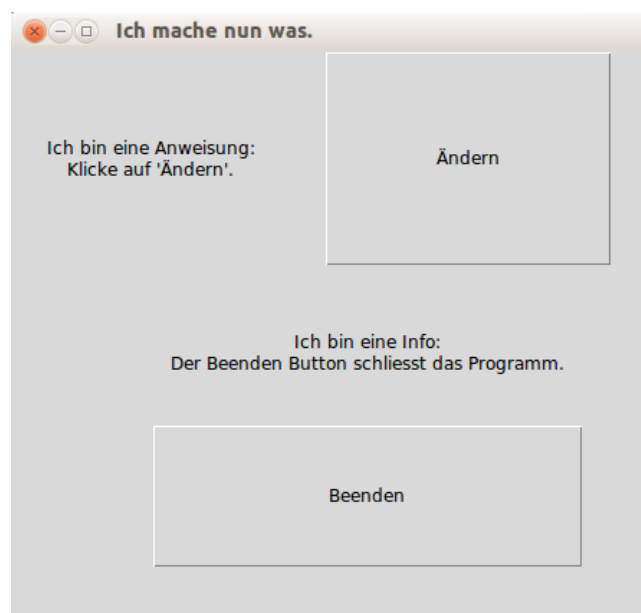
Der Place-Geometrie-Manager erlaubt das explizite Setzen der Position und der Grösse eines Fensters, entweder in absoluten Werten oder relativ zu anderen Komponenten.

Option:	Beschreibung:
x, y	Absolute Positionierung: Horizontale und vertikale Koordinate (in Pixel), in welcher die Komponente gesetzt wird.
relx, rely	Relative Positionierung: Horizontale und vertikale Platzierung bezüglich des Fensters, in welches die Komponente gepackt wird. Der Wert muss zwischen 0.0 und 1.0 liegen.
height	Höhe der Komponente bestimmen (in Pixel)
width	Breite der Komponente bestimmen (in Pixel)

Dieser Manager ist im Vergleich zu den anderen beiden der aufwendigste. Man sollte ihn nur dann benutzen, wenn es nicht anders geht. Hier ein Beispiel:

```
# Zuerst definieren wir die Grösse des Fensters
fenster.geometry("450x400")
# Wir benutzen die absoluten Koordinaten um die Komponenten zu
# setzen und definieren deren Grösse
anweisungs_label.place(x = 0, y = 0, width=200, height=150)
change_button.place(x = 220, y = 0, width=200, height=150)
info_label.place(x = 100, y = 160, width=300, height=100)
exit_button.place(x = 100, y = 260, width=300, height=100)
```

Ausgeführt sieht es dann folgendermassen aus:



Natürlich kann man noch viel mehr machen, doch das würde den Rahmen dieses Tutorials sprengen. Im Internet finden sich zu allen drei Geometrie-Managern noch viele weitere interessante Beispiele und Einstellungsmöglichkeiten.

## 8.4 Eingabefeld

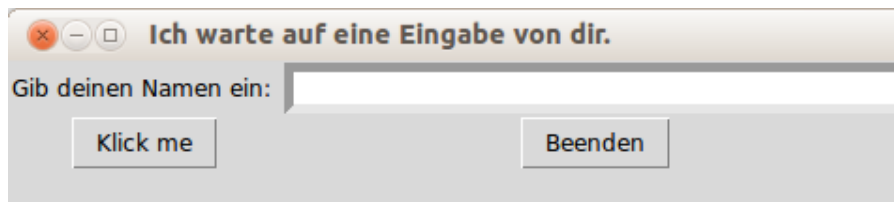
Bis jetzt hatte der Benutzer nur die Möglichkeit, über Button-Klicks mit dem Programm zu interagieren. Nun werden wir sehen, wie der Benutzer über ein Feld eine Eingabe machen kann, welche das Programm danach verarbeiten kann. Die Syntax, um ein solches Eingabefeld zu erstellen, sieht folgendermassen aus:

```
eingabefeld = Entry(fenster, option, ... )
```

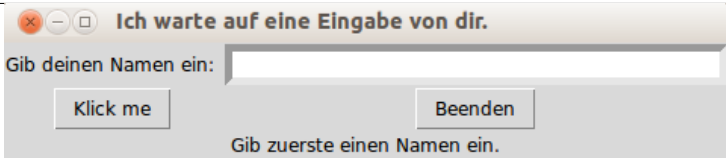
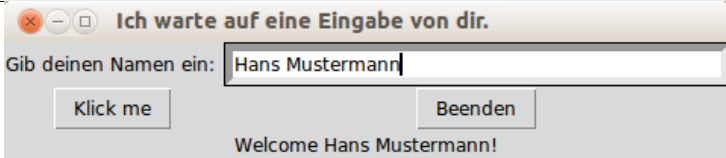
Wir sehen nun gleich ein Programm, welches im Fenster auf die Eingabe des Benutzers wartet und danach eine entsprechende Ausgabe generiert:

```
1  from tkinter import *
2
3  # Die folgende Funktion soll ausgeführt werden, wenn
4  # der Benutzer den Button Klick me anklickt
5  def button_action():
6      entry_text = eingabefeld.get()
7      if (entry_text == ""):
8          welcome_label.config(text="Gib zuerst einen Namen ein.")
9      else:
10         entry_text = "Welcome " + entry_text + "!"
11         welcome_label.config(text=entry_text)
12
13 fenster = Tk()
14 fenster.title("Ich warte auf eine Eingabe von dir.")
15
16 # Anweisungs-Label
17 my_label = Label(fenster, text="Gib deinen Namen ein: ")
18
19 # In diesem Label wird nach dem Klick auf den Button der Benutzer
20 # mit seinem eingegebenen Namen begrüsst.
21 welcome_label = Label(fenster)
22
23 # Hier kann der Benutzer eine Eingabe machen
24 eingabefeld = Entry(fenster, bd=5, width=40)
25
26 welcom_button = Button(fenster, text="Klick me", command=button_action)
27 exit_button = Button(fenster, text="Beenden", command=fenster.quit)
28
29
30 # Nun fügen wir die Komponenten unserem Fenster hinzu
31 my_label.grid(row = 0, column = 0)
32 eingabefeld.grid(row = 0, column = 1)
33 welcom_button.grid(row = 1, column = 0)
34 exit_button.grid(row = 1, column = 1)
35 welcome_label.grid(row = 2, column = 0, columnspan = 2)
36
37 mainloop()
```

Wird das Programm ausgeführt, so bekommen wir folgendes Fenster:



In Zeile 7 im obigen Programm wird überprüft, ob der Benutzer etwas in das Feld geschrieben hat oder nicht. Dementsprechend sieht dann die Ausgabe im Fenster nach dem Klick auf den Klick me-Button jeweils anders aus:

Ohne Eingabe		
Mit Eingabe		

## 8.5 Menü und MessageBox

Bei den meisten Programmen ist man es gewohnt, dass man am oberen Fensterrand ein Menü zur Verfügung hat. Wie man ein solches erstellt, sehen wir in diesem Kapitel. Wir werden ein Pull-down-Menü mit tkinter erstellen. D.h. dass z.B. ganz oben im Fenster, wenn auf die Fläche Datei geklickt wird, ein Pull-down-Menü mit einigen Auswahlmöglichkeiten erscheint, wie z.B. Speichern, Speichern unter oder Exit. Die Syntax dazu sieht folgendermassen aus:

```
my_menu = Menu(fenster, option, ... )
```

In unserem Beispiel erstellen wir eine Menüleiste mit den Einträgen Datei und Help. Beim Klick auf Datei soll ein Drop-Down-Menü aufgehen mit den Einträgen Anwenden und Exit. Wenn man auf Anwenden klickt, wird ein Text auf der Konsole ausgegeben und bei Exit wird das Programm natürlich beendet. Bei Help soll ein Eintrag Info! erscheinen, welcher eine MessageBox öffnet. Die MessageBox ist lediglich ein weiteres Fenster, welches Informationen enthalten kann und beim Klick auf OK wieder geschlossen wird. Eine MessageBox kann z.B. so aufgerufen werden:

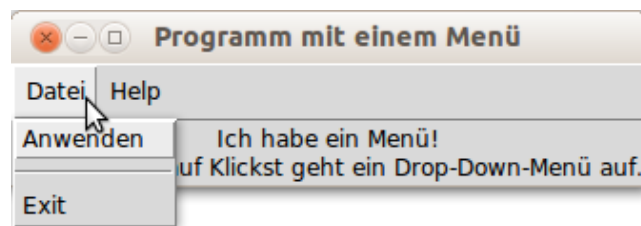
```
from tkinter import messagebox
messagebox.showinfo(message="Infotext", title = "Box-Titel")
```



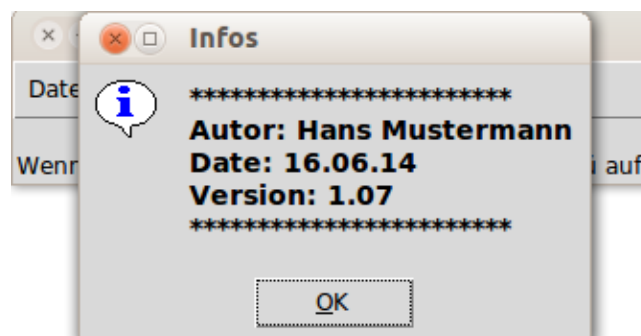
Dies alles in einem Programm umgesetzt kann dann folgendermaßen aussehen:

```
1  from tkinter import *
2  from tkinter import messagebox
3
4  def button_action():
5      print("Ich wurde über das Menü ausgeführt.")
6
7  def action_get_info_dialog():
8      m_text = "\n\
9      *****\n\
10     Autor: Hans Mustermann\n\
11     Date: 16.06.14\n\
12     Version: 1.07\n\
13     *****"
14     messagebox.showinfo(message=m_text, title = "Infos")
15
16 fenster = Tk()
17 fenster.title("Programm mit einem Menü")
18
19 info_text = Label(fenster, text = "Ich habe ein Menü! \n\
20 Wenn du darauf Klickst geht ein Drop-Down-Menü auf.")
21 info_text.pack()
22
23 # Menüleiste erstellen
24 menuleiste = Menu(fenster)
25
26 # Menü Datei und Help erstellen
27 datei_menu = Menu(menuleiste, tearoff=0)
28 help_menu = Menu(menuleiste, tearoff=0)
29
30 # Beim Klick auf Datei oder auf Help sollen nun weitere Einträge er-
31 # scheinen. Diese werden zu "datei_menu" und "help_menu" hinzugefügt
32 datei_menu.add_command(label="Anwenden", command=button_action)
33 datei_menu.add_separator() # Fügt eine Trennlinie hinzu
34 datei_menu.add_command(label="Exit", command=fenster.quit)
35
36 help_menu.add_command(label="Info!", command=action_get_info_dialog)
37
38 # Nun fügen wir die Menüs (Datei und Help) der Menüleiste als
39 # "Drop-Down-Menü" hinzu
40 menuleiste.add_cascade(label="Datei", menu=datei_menu)
41 menuleiste.add_cascade(label="Help", menu=help_menu)
42
43 # Die Menüleiste mit den Menüeinträgen dem Fenster übergeben.
44 fenster.config(menu=menuleiste)
45
46 fenster.mainloop()
```

Ausgeführt kann es dann so aussehen



und bei Klick auf Help -> Info!



Nun haben wir hier ein paar wichtige Komponenten kennengelernt, mit welchen man eine einfache graphische Benutzeroberfläche erstellen kann. Natürlich gibt es noch viele weitere Komponenten, welche man gebrauchen kann. Jedoch sollte man nach diesem Kapitel in der Lage sein, den Gebrauch der restlichen Komponenten selber herauszufinden, natürlich mit der Hilfe des Internets.