Grafische Benutzeroberflächen in Python

Christoph Schmidt

Oktober 2009

Gliederung

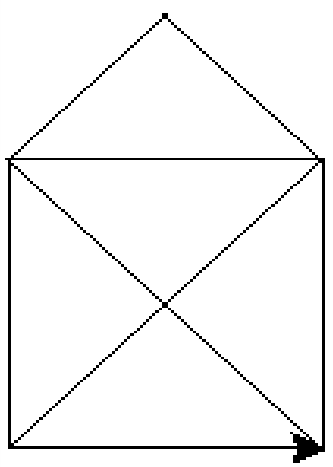
1. Einfache Grafiken mit dem Turtle-Modul
2. Einführung in tkinter
3. Layoutmanager
4. Ereignisverarbeitung

Gliederung

1. Einfache Grafiken mit dem Turtle-Modul
2. Einführung in tkinter
3. Layoutmanager
4. Ereignisverarbeitung

Teil 1: Einfache Grafiken mit Turtle

Die „Turtle“ ist eine programmierbare „Schildkröte“, die auf dem Boden hin- und herlaufen kann und, falls der Zeichenstift abgesenkt ist, ihren zurückgelegten Weg aufzeichnet.



1.1 Turtle laden und aktivieren

- Modul Turtle importieren

```
import turtle
```

oder

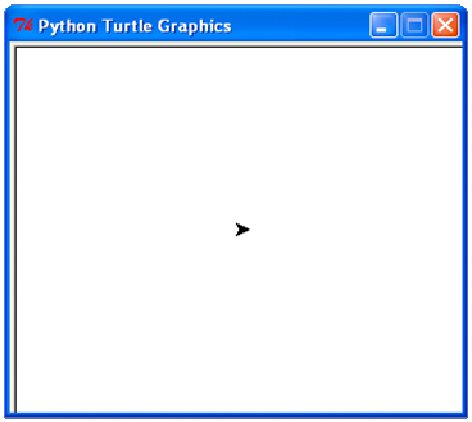
```
from turtle import *
```

- Turtle aktivieren

```
turtle.reset()
```

oder

```
reset()
```



Grafik-Fenster mit der Turtle im Ursprung (0|0) des Koordinatensystems

Wir gehen im Folgenden von der „`from ... import *`“-Notation aus.

1.2 Turtle – Beispiele 1

```
>>> forward(100)
>>> left(90)
>>> forward(200)
```

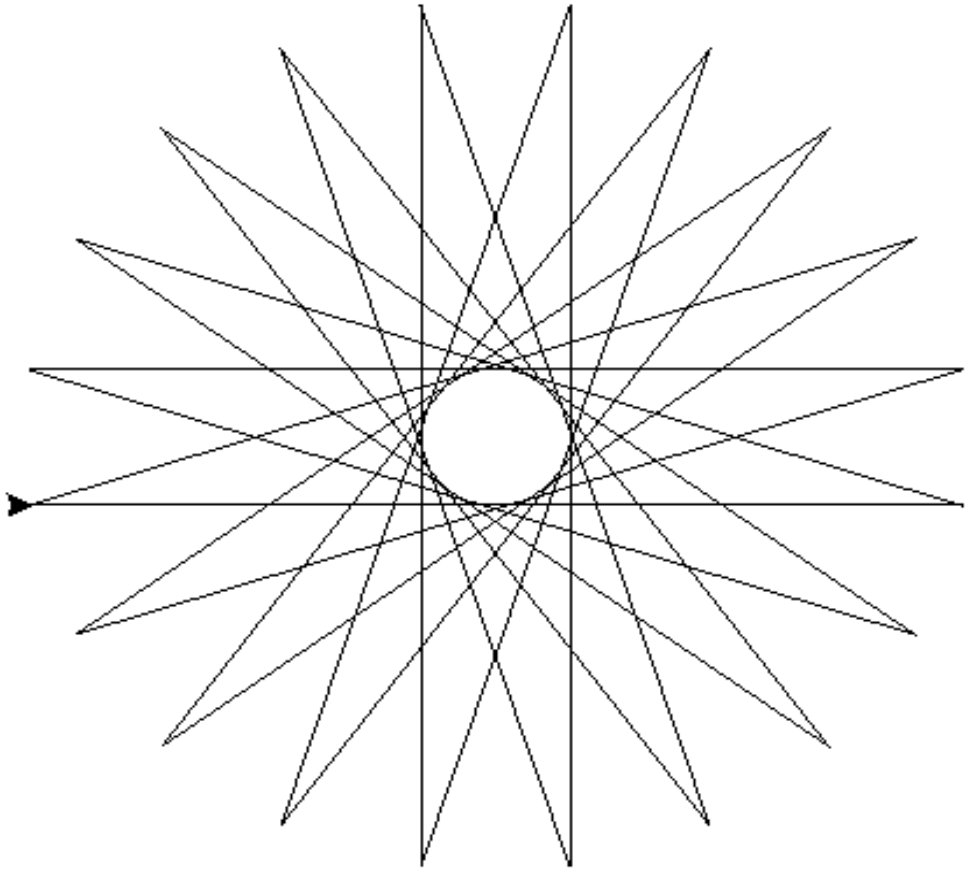
Gehe 100 Pixel geradeaus,
drehe dich um 90° nach links,
gehe 200 Pixel geradeaus.

```
>>> forward(100)
>>> up()
>>> goto(0,-50)
>>> down()
>>> forward(100)
```

Gehe 100 Pixel geradeaus,
hebe den Stift an,
gehe zum Punkt (0|-50),
senke den Stift ab,
gehe 100 Pixel geradeaus.

1.2 Turtle – Beispiele 2

```
>>> for i in range(20):
        forward(400)
        left(162)
```



1.3 Turtle – Befehle 1

Geradlinige Bewegungen	
`forward(n)`	um n Pixel nach vorne bewegen
`backward(n)`	um n Pixel nach hinten bewegen
`goto(x,y)`	zur Position (x,y) bewegen

Drehungen und Kreise	
`left(α)`	um α nach links drehen
`right(α)`	um α nach rechts drehen
`circle(r)`	einen Kreis mit Radius r zeichnen
`circle(r,α)`	einen Kreisbogen mit Radius r und Winkel α zeichnen

1.3 Turtle – Befehle 2

Konfiguration und Erscheinungsbild	
`up()`	den Stift anheben
`down()`	den Stift absenken
`width(w)`	die Linienstärke auf w Pixel stellen
`color(f)`	die Linienfarbe ändern (z.B. ‘red‘)

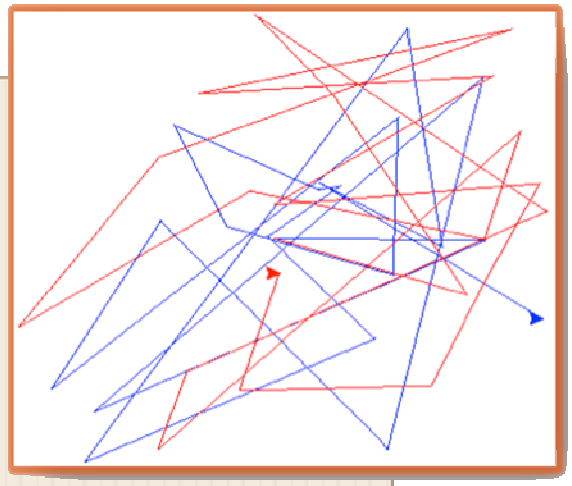
Bildschirmprozeduren	
`clear()`	die Zeichenfläche löschen
`reset()`	die Turtle in den Anfangszustand versetzen
`bye()`	das Turtlefenster schließen

Weitere Informationen unter `http://docs.python.org/3.1/library/turtle.html`

1.4 Mehrere Schildkröten

Sollen mehrere Turtles auf der Zeichenfläche zeichnen, so kann man diese als Objekte der Klasse `Pen` erstellen.

```
from turtle import *
from random import *

reset()
t2 = Pen()

color('blue')
t2.color('red')

for i in range(20):
    goto(randint(-200,200),randint(-200,200))
    t2.goto(randint(-200,200),randint(-200,200))
```



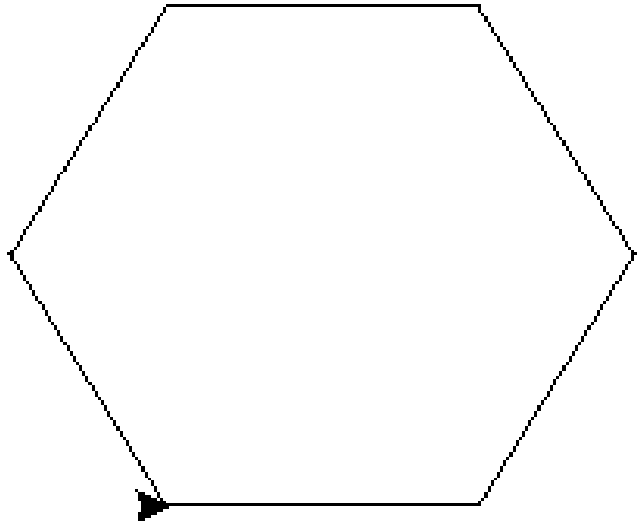
Standard-Turtle: blau
2. Turtle t2: rot

Standard-Turtle bewegen

2. Turtle bewegen

1.5.1 Aufgabe

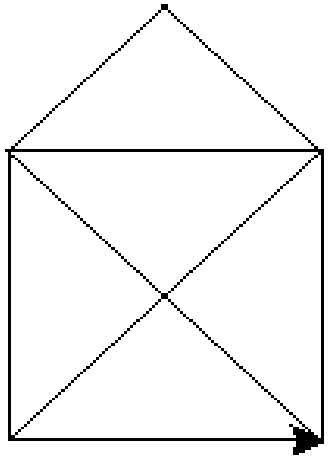
Zeichnen Sie mit der Turtle ein regelmäßiges Sechseck.



Erweiterung: Schreiben Sie eine Prozedur `Vieleck(n)`, die ein regelmäßiges n-Eck zeichnet.

1.5.2 Aufgabe

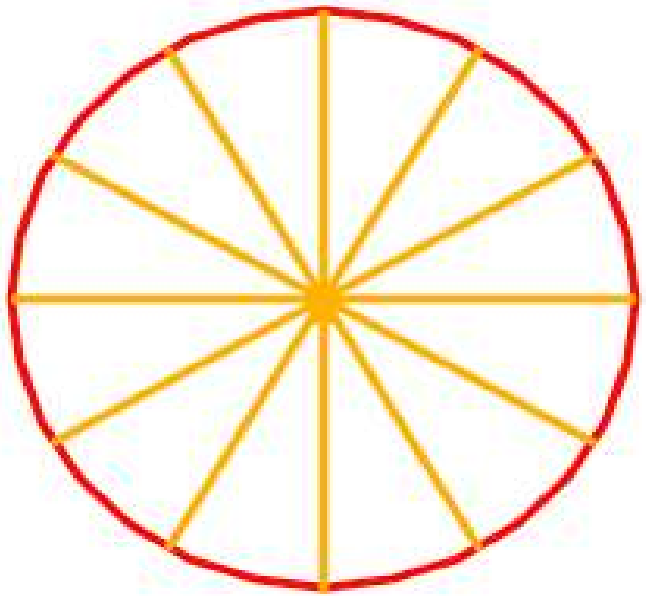
Zeichnen Sie mit der Turtle das Haus vom Nikolaus.



Erweiterung: Schreiben Sie eine Prozedur `Nikolaus(n)`, die ein Haus vom Nikolaus der Breite n zeichnet.

1.5.3 Aufgabe

Zeichnen Sie mit der Turtle den abgebildeten Kuchen.



Erweiterung: Schreiben Sie eine Prozedur `Kuchen(n)`, die einen in n gleich große Stücke geteilten Kuchen zeichnet.

Gliederung

1. Einfache Grafiken mit dem Turtle-Modul
2. Einführung in tkinter
3. Layoutmanager
4. Ereignisverarbeitung

Teil 2: Einführung in Tkinter

Das Modul „tkinter“ gehört zur Standard-Distribution von Python und ermöglicht das Erstellen von grafischen Benutzeroberflächen.

2.1 Einleitung

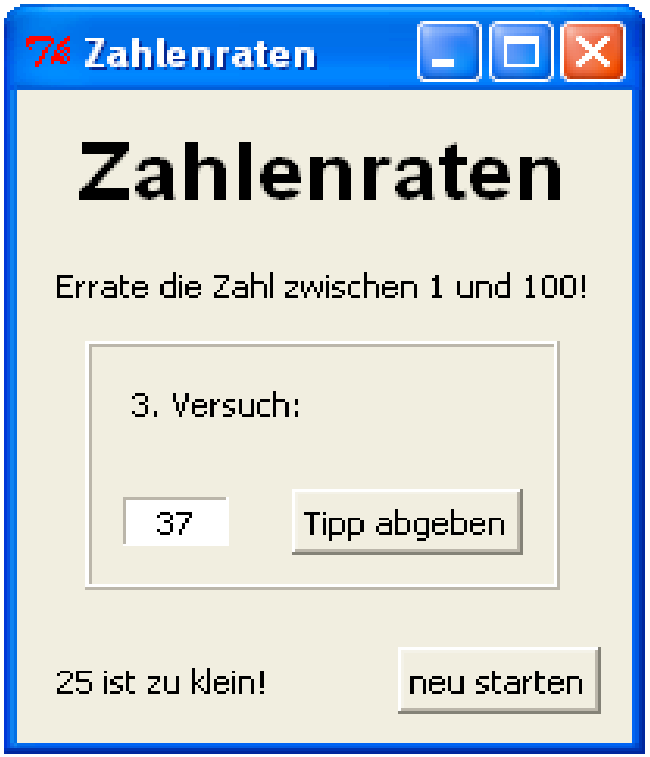
Interaktive Programme ermöglichen die Kommunikation zwischen Mensch und Computer über eine Benutzeroberfläche.

```
Errate die Zahl zwischen 0 und 100!
Dein Tipp: 50
zu klein!
Dein Tipp: 75
zu gross!
Dein Tipp: 67
zu klein!
Dein Tipp: 71
richtig!
Du hast 4 Versuche gebraucht.
>>> |
```

Textbasierte Kommunikation: synchronisiert, die zeitliche Abfolge ist vorgegeben.

2.1 Einleitung

Interaktive Programme ermöglichen die Kommunikation zwischen Mensch und Computer über eine Benutzeroberfläche.



Multimediale Kommunikation:
asynchron, der Benutzer kann in selbst gewählter Reihenfolge
- Fenster verschieben,
-Objekte anklicken,
- Tastatureingaben tätigen.

2.1 Einleitung

Die Erstellung einer GUI (Graphical User Interface) erfolgt in 3 Schritten:

- Definition der Elemente

Schaltflächen, Eingabefelder, Abbildungen, bei Python sog. *Widgets*

- Layout

Anordnung der Elemente mit Hilfe eines *Layoutmanagers*

- Definition der Interaktivität

Verknüpfung von Benutzer-aktivitäten mit einer Funktio-nalität

2.2 Viele verschiede Möglichkeiten

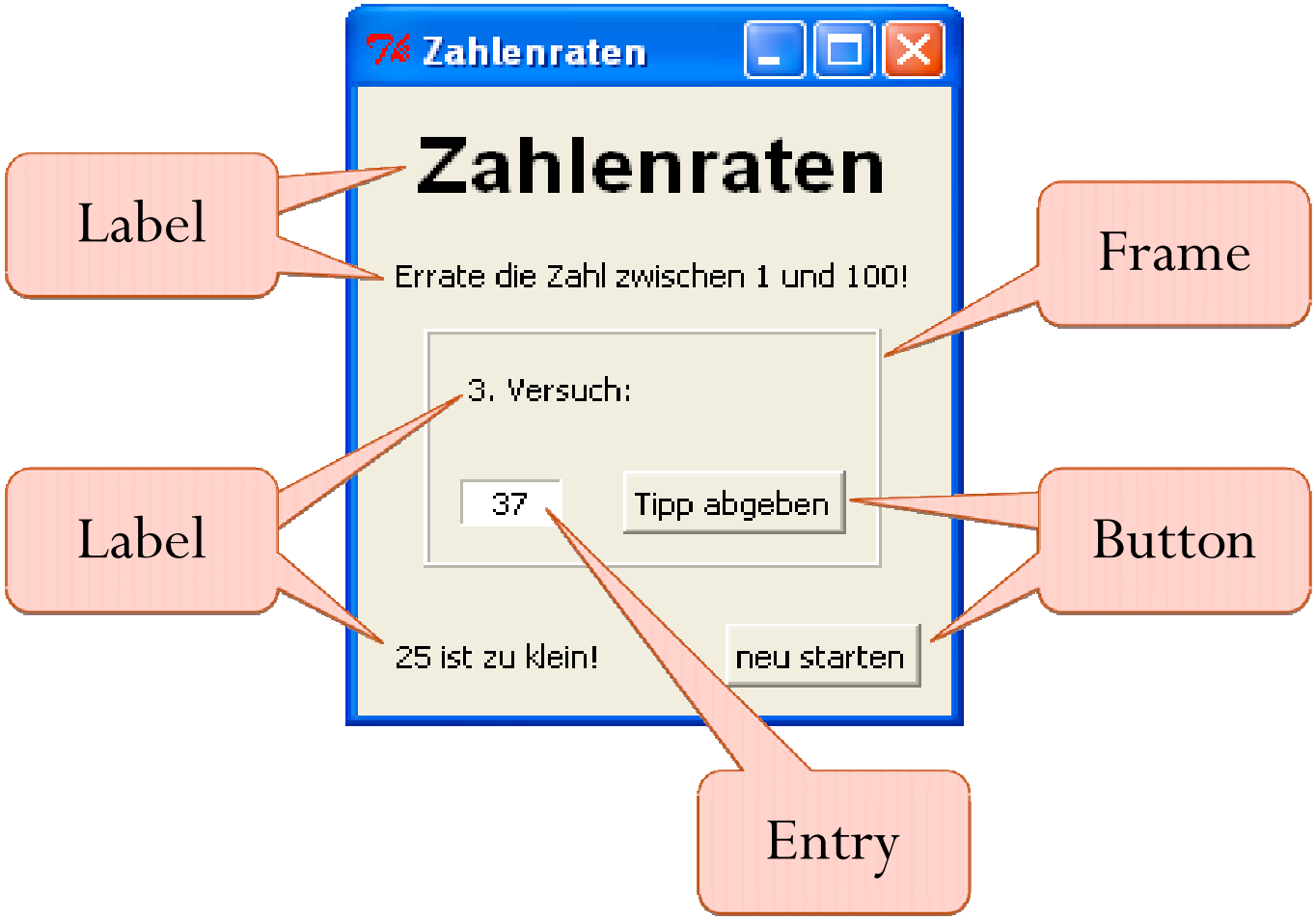
Für die Programmiersprache Python existieren mehrere Bibliotheken, die die GUI-Programmierung erlauben, z.B.

- Tk mit tkinter
- Qt mit PyQt
- Gtk mit PyGtk
- …

enthalten im Standardpaket, geeignet für kleinere Anwendungen

sehr mächtig, wird z.B. für Google Earth, Skype oder Opera verwendet

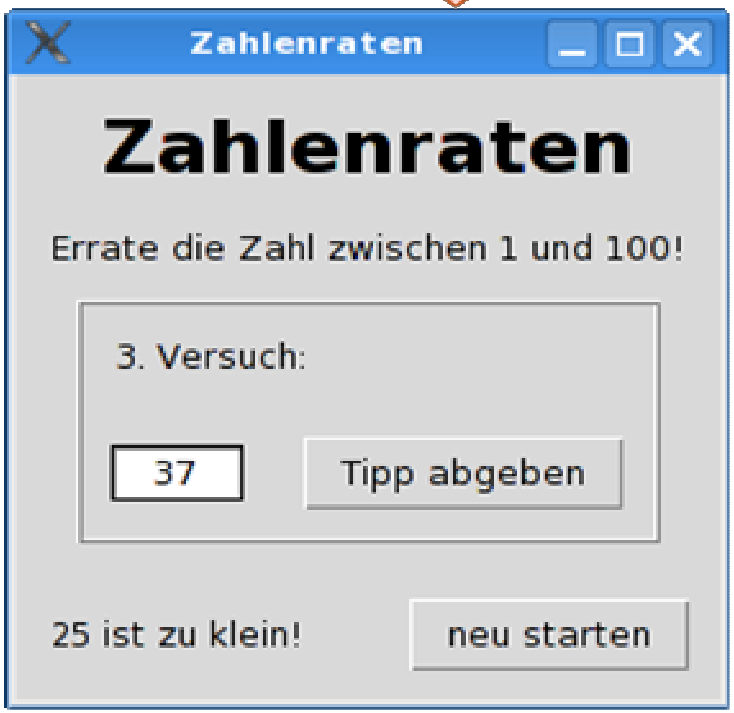
2.3 Beispiel


Zahlenraten
Zahlenraten
Errate die Zahl zwischen 1 und 100!
3. Versuch:
37
Tipp abgeben
25 ist zu klein!
neu starten
Label
Label
Frame
Button
Entry

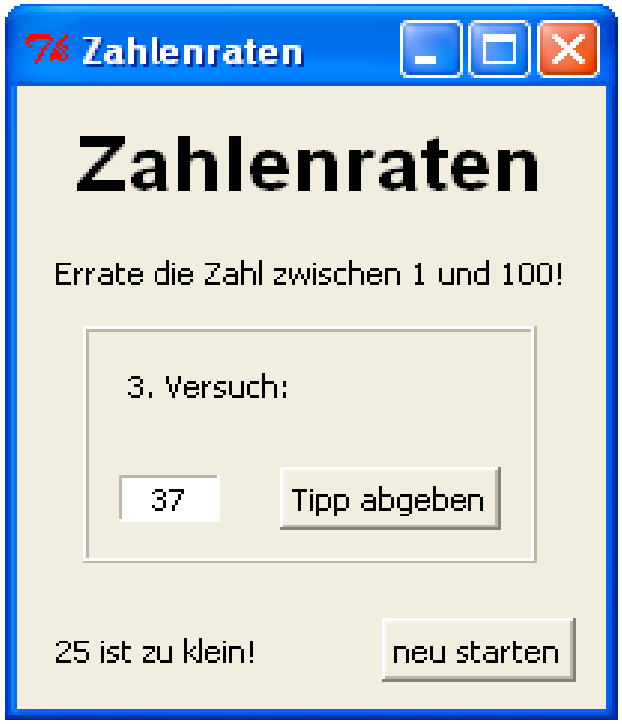
2.3 Beispiel – plattformunabhängig

Der gleiche Quellcode unter 3 verschiedenen Betriebssystemen:

Linux

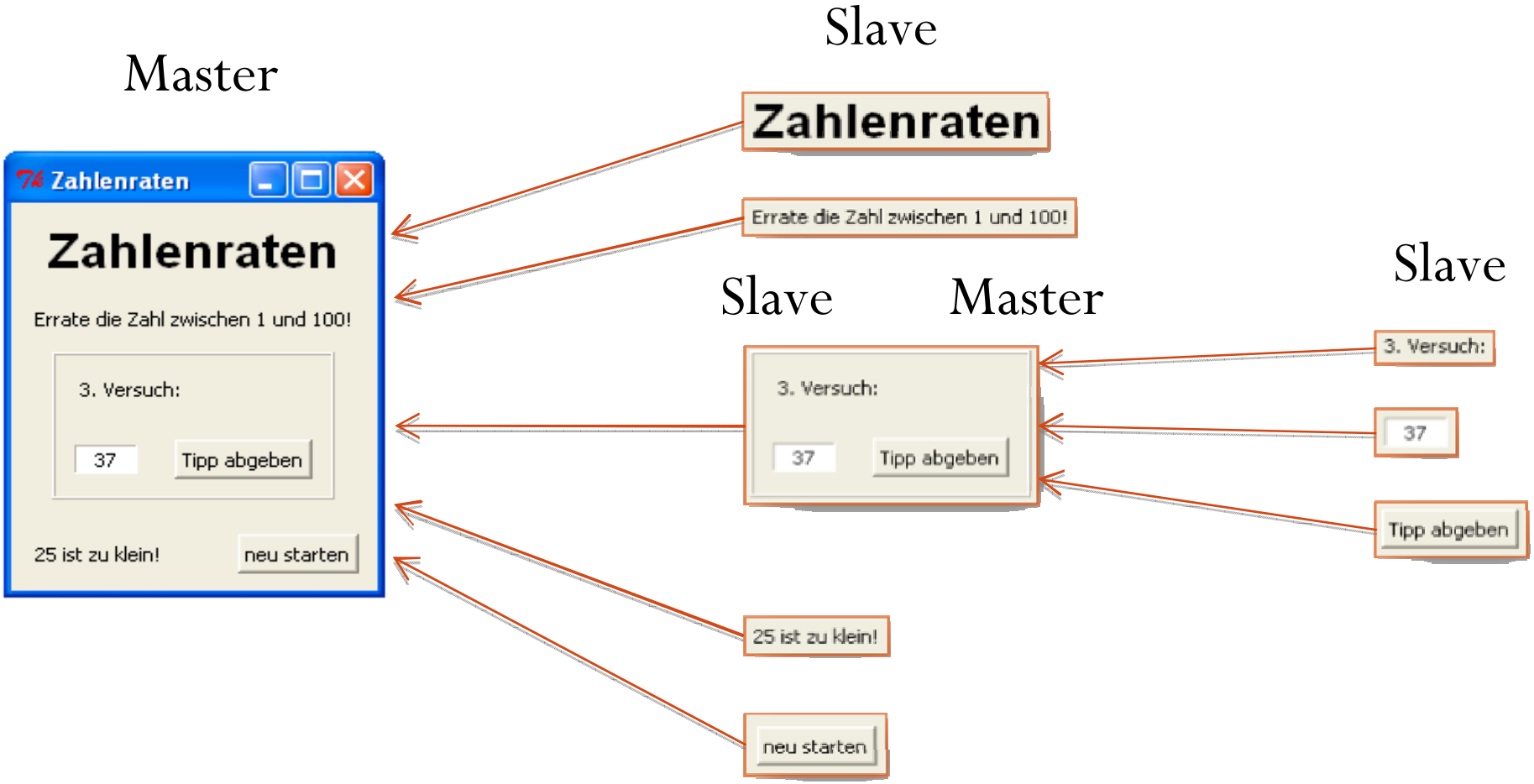


Win XP



Mac OS X

2.4 Master-Slave Hierarchie



Die Master-Slave-Hierarchie bezieht sich nur auf die Darstellung der grafischen Objekte auf dem Bildschirm.

2.5 Erstellung einer GUI

Widgets sind Objekte, die verschiedene Attribute besitzen.

Die Attribute können durch Schlüsselwort-Argumente gesetzt werden. Bei diesem Verfahren spielt die Reihenfolge keine Rolle.

```
titel = Label(master = fenster, text='Hallo', fg='red')
```

2.5 Erstellung einer GUI

```
from tkinter import *
```

Modul importieren

Widgets definieren

```
fenster = Tk()
titel = Label(master=fenster, text='Hallo Welt!',
              font=('Arial',20), fg='red')
```

```
titel.pack()
```

Widgets platzieren

```
fenster.mainloop()
```

Ereignisverarbeitung starten

2.5 Erstellung einer GUI

Das komplette „Hallo-Welt-Programm“:

```
from tkinter import *
fenster = Tk()
titel = Label(master=fenster, text='Hallo Welt!',
              font=('Arial',20), fg='red')
titel.pack()
fenster.mainloop()
```

2.5 Aufgabe

Testen Sie das angegebene „Hallo Welt“-Programm und erweitern Sie es durch weitere Labels.

2.6 Optionen von Widgets

Optionen eines Widgets können bei der Instanzierung gesetzt werden:

```
titel = Label(master=fenster, text='Hallo Welt!',
              font=('Arial',20), fg='red‘)
```

Mit Hilfe der Methode `config()` können ein oder mehrere Optionen eines Widgets `w` nachträglich geändert werden:

```
w.config(option1=wert1[, option2=wert1 ...])
```

Beispiel:

```
titel.config(fg='blue', bg='yellow')
```

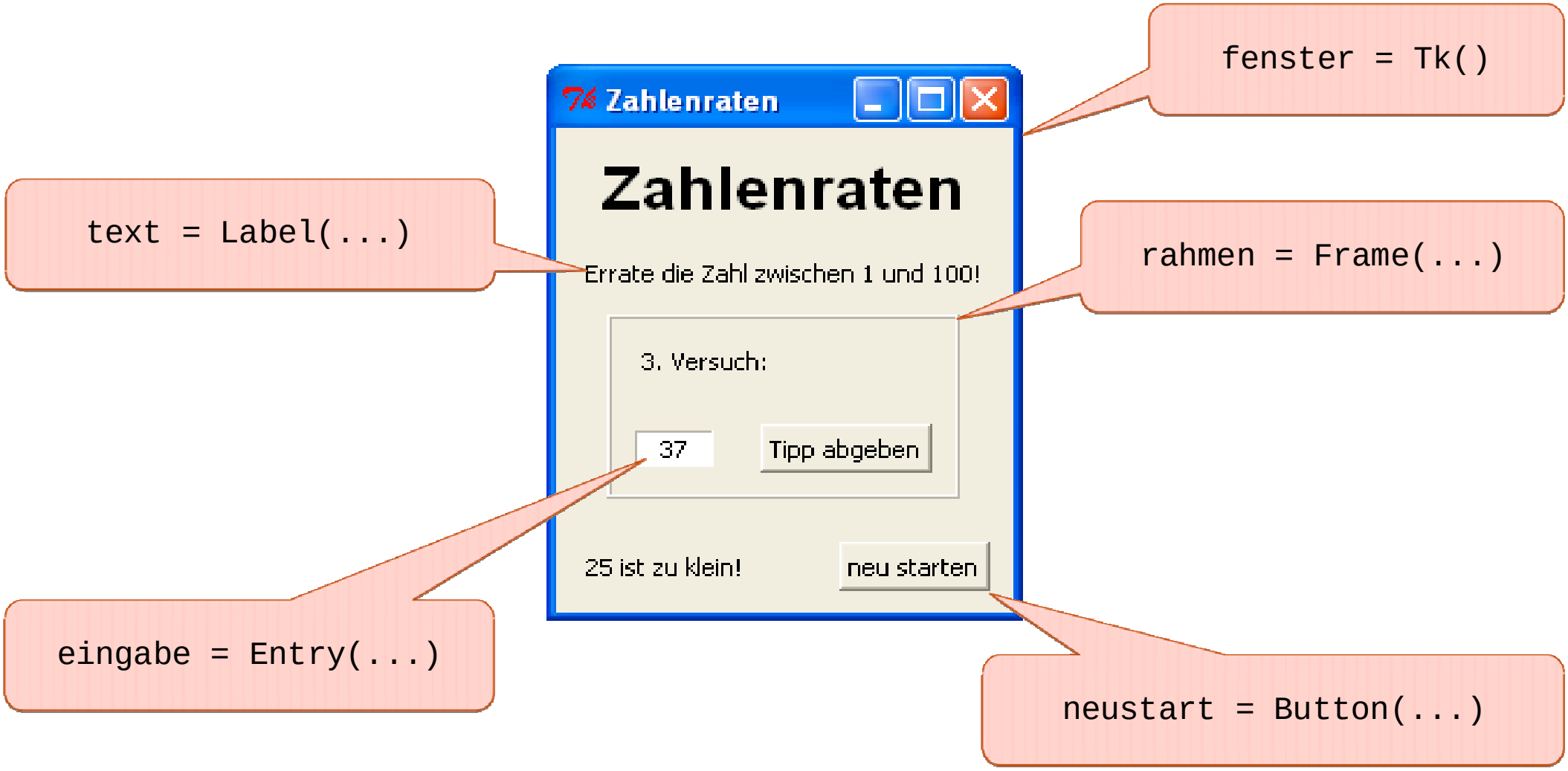
2.6 Optionen von Widgets

Auswahl an Standard-Attributen, die fast alle Widgets besitzen:

`bd, borderwidth`	ganze Zahl, die die Breite des Rahmens angibt
`bg, background`	Hintergrundfarbe in der Form #rgb oder #rrggbb oder ‚red', ‚green', ...
`fg, foreground`	Vordergrundfarbe (Textfarbe)
`padx`	leerer Platz in Pixeln rechts und links vom Widget
`pady`	leerer Platz in Pixeln oben und unten vom Widget
`text`	Beschriftung (Zeichenkette)
`font`	Schriftformat: (Name, Größe[, Stil]), z.B. ('Comic Sans MS',14) oder ('Courier',12,'italic')

Fast alle Attribute besitzen Voreinstellungen, sodass man in der Regel nur an einigen Stellen etwas abändern muss.

2.7 Eine Auswahl von Widgets



2.7.1 Die Klasse Tk

Programme mit GUI laufen immer in Anwendungsfenstern, welche ein Objekt der Klasse Tk sind. Der Konstruktor wird ohne Argumente aufgerufen, das Fenster wird nicht mit einem Layoutmanager platziert.

```
from Tkinter import *
meinfenster = Tk()
meinfenster.title('Mein Demo-Fenster')
meinfenster.mainloop()
```

Mit der Methode `geometry()` lässt sich die Größe und Position des Fensters festlegen.

Der Fenstertitel lässt sich mit der Methode `title()` ändern.

2.7.2 Die Klasse Button

Objekte der Klasse Button sind Schaltflächen, die, wenn sie mit der linken Maustaste angeklickt werden, eine Aktion auslösen.

```
meinButton = Button(master[, option1=wert1[, ...]])
```

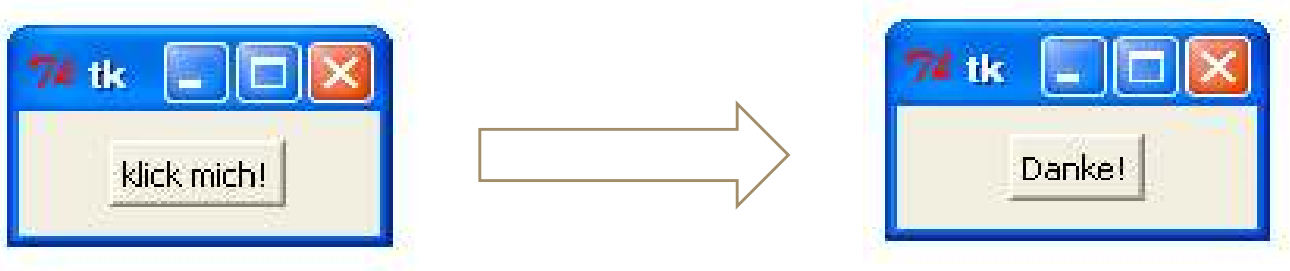
Die Option „master“ ist immer das erste Attribut bei Aufruf eines Konstruktors. Daher kann man „master =“ weglassen.

```
command=prozedurname
```

Die Option „command“ legt fest, welche Prozedur bei Mausklick ausgeführt werden soll.
(Achtung: keine Klammern!)

2.7.2 Die Klasse Button - Beispiel

```
from tkinter import *

def danke():
    button.config(text='Danke!')

fenster = Tk()

button = Button(fenster, text='klick mich!', command=danke)
button.pack(pady=10)

fenster.mainloop()
```



2.7.3 Die Klasse Label

Labels dienen zum Anzeigen von Text, sind also reine Ausgabeobjekte.

```
meinLabel = Label(master[, option1=wert1[, ...]])
```

Das wichtigste Attribut ist sicherlich `text='Text'` bzw. `text="Text"`

Der Text eines Labels lässt sich mit der Methode `config()` dynamisch ändern:

```
meinLabel.config(text='neuer Text‘)
```

2.7.3 Die Klasse Label - Beispiel

```
from tkinter import *

fenster = Tk()

label = [
    Label(fenster, font=('Arial',40), text='riesig', width=10),
    Label(fenster, font=('Arial',20), text='mittel'),
    Label(fenster, font=('Arial',10), text='klein')]

for l in label: l.pack()

fenster.mainloop()
```

2.7.4 Die Klasse Entry

Ein Entry-Widget liefert ein einzeiliges Eingabefeld.

```
eingabe = Entry(master[, option1=wert1[, ...]])
```

Attribut	Erläuterung
`width`	Breite des Eingabefeldes, Voreinstellung ist 20
`justify`	Textausrichtung: LEFT, CENTER oder RIGHT
`show`	Anzeige, z.B. '*' für Kennworteingabe

Methode	Erläuterung
`get()`	liefert den Inhalt des Feldes als String
`delete(a[,b])`	löscht einzelne Zeichen: entweder das Zeichen mit dem Index a oder ab Index a bis Index b.

2.7.5 Die Klasse Scale

Scale-Widgets sind waagrechte oder senkrechte Schieberegler.

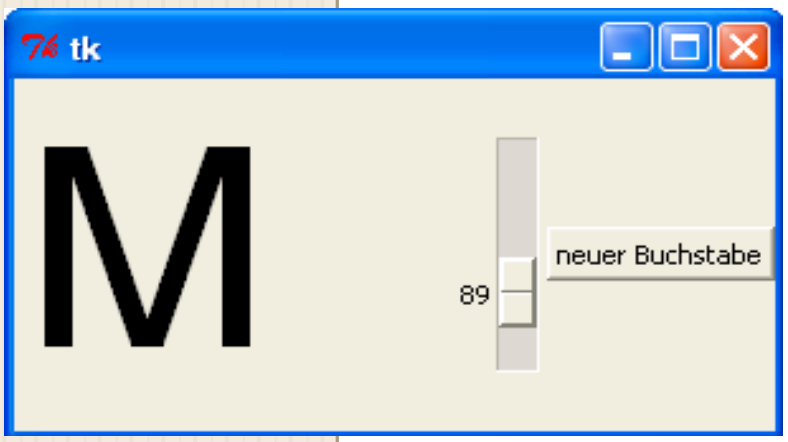
```
regler = Scale(master[, option1=wert1[, ...]])
```

Attribut	Erläuterung
`command`	Prozedur, die bei Betätigung des Reglers ausgeführt wird
`from_, to`	Wertebereich des Schiebereglers
`orient`	Ausrichtung: VERTICAL oder HORIZONTAL
`length`	Länge, Standard ist 100 Pixel

Methode	Erläuterung
`get()`	liefert den aktuellen Wert
`set(wert)`	setzt den Regler auf einen bestimmten Wert

2.7.5 Die Klasse Scale - Beispiel

```
from tkinter import *
from random import *

def neuerBuchstabe():
    a = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    buchstabe.config(text=choice(a))

def setzeGroesse(event):
    buchstabe.config(font=('Arial',regler.get()))

fenster = Tk()
fenster.geometry('300x100')

buchstabe = Label(fenster,text='A', font=('Arial',20))
regler = Scale(fenster,from_=4, to=120, command=setzeGroesse)
regler.set(20)

button = Button(fenster, text='neuer Buchstabe',
                command=neuerBuchstabe)

buchstabe.pack(side=LEFT)
button.pack(side=RIGHT)
regler.pack(side=RIGHT)

fenster.mainloop()
```



2.7.6 Die Klasse Frame

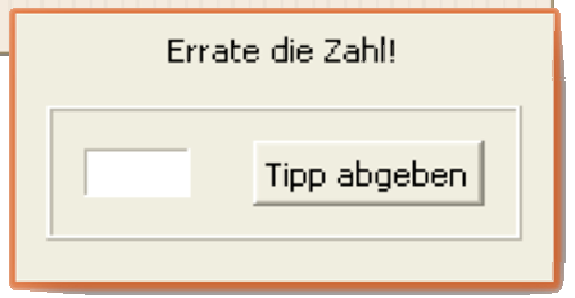
Frames dienen zum Gruppieren von Widgets.

```
rahmen = Frame(master[, option1=wert1[, ...]])
```

```
anleitung = Label(fenster, text='Errate die Zahl!')
rahmen = Frame(fenster, relief=RIDGE, bd=2)
eingabe = Entry(rahmen, width=5, justify=CENTER)
button = Button(rahmen, text='Tipp abgeben')
```

Der Name des Frames wird als master der untergeordneten Widgets angegeben.

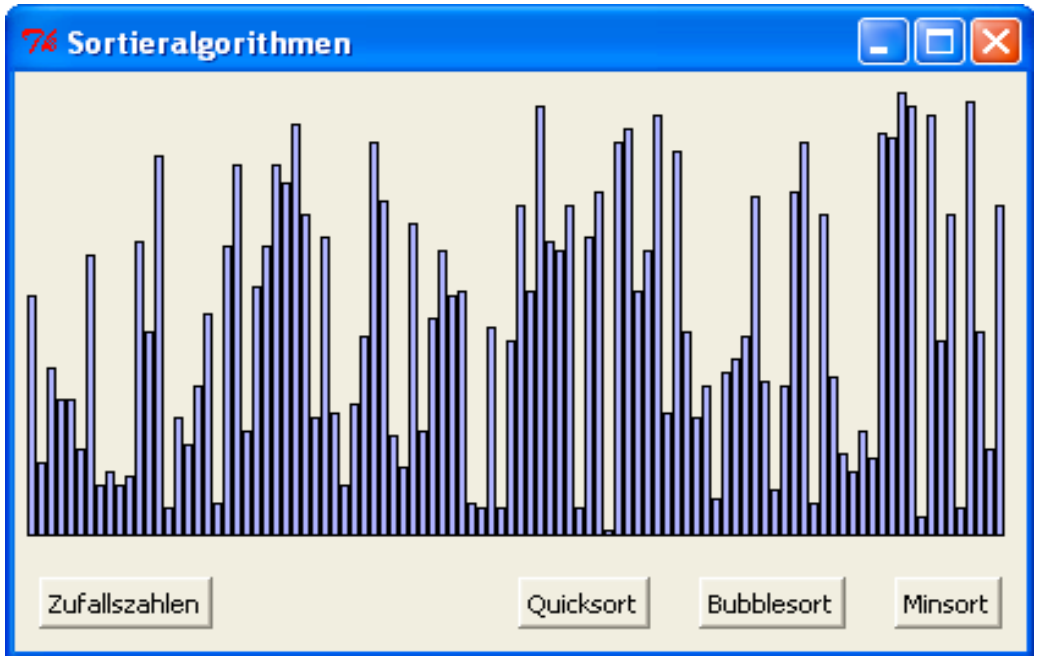
In der Voreinstellung sind Frames unsichtbar.



2.7.7 Die Klasse Canvas

Ein Canvas-Objekt ist eine Zeichenfläche, auf der grafische Elemente (*items*) wie Kreise, Rechtecke platziert werden können.

```
zeichenflaeche = Canvas(master[, option1=wert1[, ...]])
```



2.7.7 Die Klasse Canvas

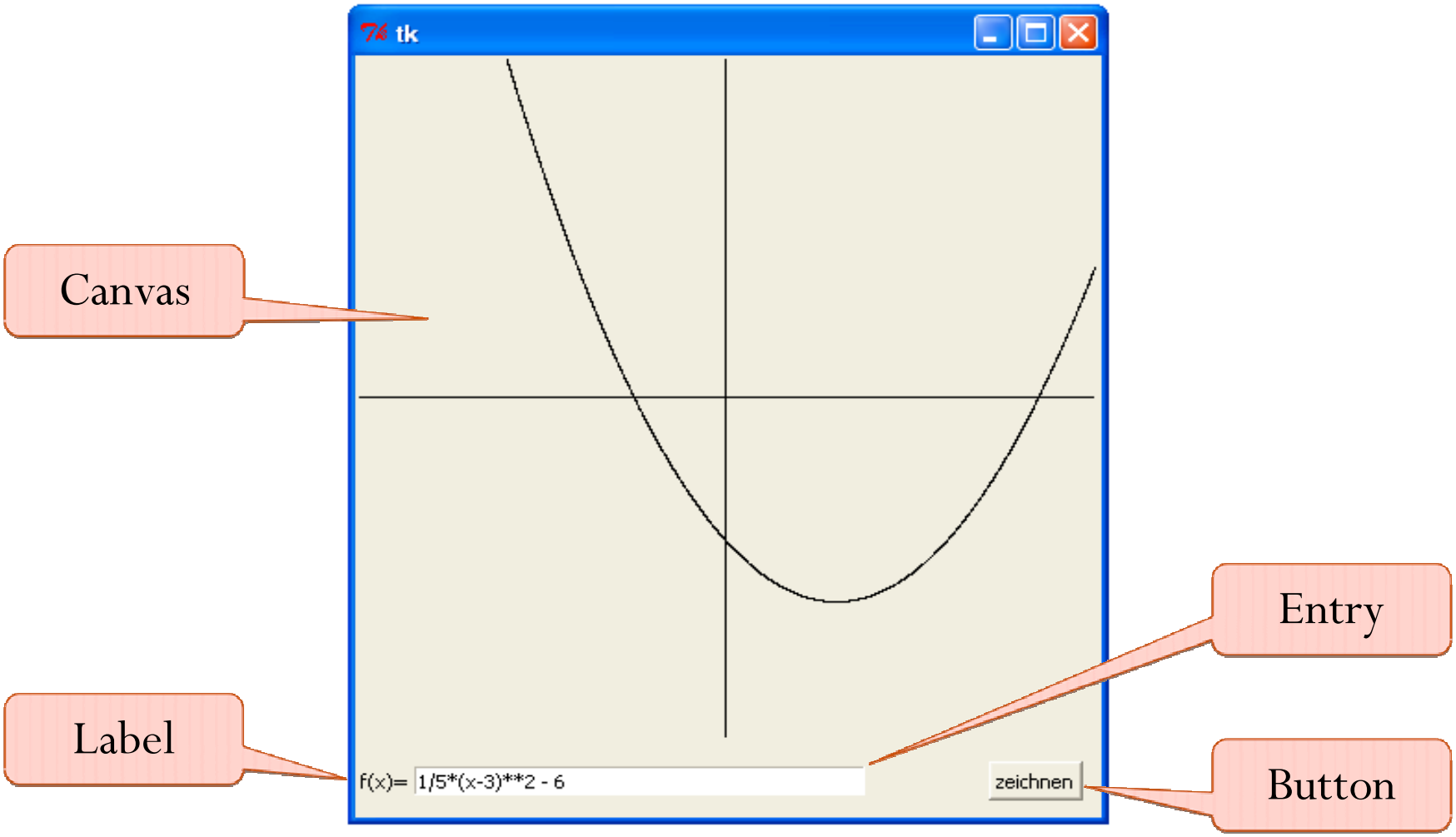
Methode	Erläuterung
`create_line(...)`	erzeugt eine Linie
`create_rectangle(...)`	erzeugt ein Rechteck
`create_oval(...)`	erzeugt eine Ellipse
`create_arc(...)`	erzeugt einen Ellipsenbogen
`create_polygon(...)`	erzeugt ein Polygon
`create_text(...)`	erzeugt ein Textobjekt
`itemconfigure(...)`	konfiguriert ein Item
`move(...)`	bewegt ein Item
`...`	...

Beispiel:

```
flaeche.create_rectangle(10,20,160,180,fill='#a0a0ff')
```

2.7.7 Die Klasse Canvas - Beispiel

Ein sehr einfacher Funktionsplotter:



2.7.7 Die Klasse Canvas - Beispiel

```
from tkinter import *
from math import *

def zeichnen():
    s = eingabe.get()
    for i in range(4000):
        x = i/200.0 - 10
        f = eval(s)
        flaeche.create_line(x*20+200,200-f*20,x*20+201,200-f*20)

fenster = Tk()

flaeche = Canvas(fenster, width=400, height=400)
eingabe = Entry(fenster, width=40)
label = Label(fenster, text='f(x)=')
button = Button(fenster,text='zeichnen',command=zeichnen)

flaeche.pack()
label.pack(side=LEFT)
eingabe.pack(side=LEFT)
button.pack(side=RIGHT,pady=10,padx=10)

flaeche.create_line(0,200,400,200)
flaeche.create_line(200,0,200,400)

fenster.mainloop()
```

2.7.8 Weitere Widgets

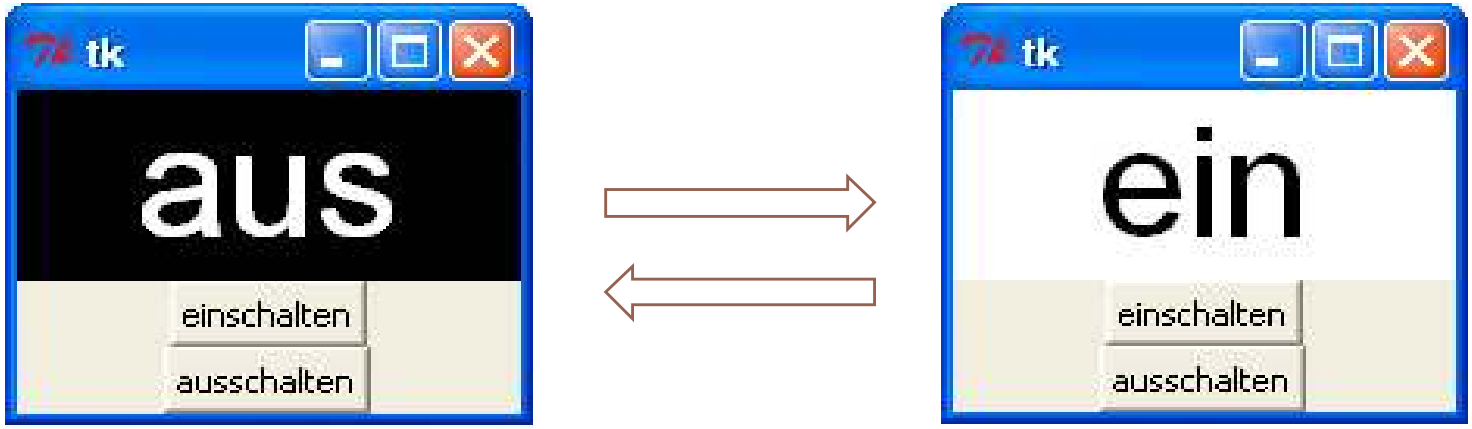
- Radiobutton
- Checkbutton
- Listbox
- Menubutton
- Menu
- Scrollbar
- Toplevel
- Spinbox
- tkMessageBox
- ...

Weitere Informationen z.B. unter

`http://effbot.org/tkinterbook`

2.8 Aufgabe Lichtschalter

Schreiben Sie eine Lichtschalter-Simulation. Die GUI soll ein Label und zwei Schaltflächen enthalten und sich gemäß der Abbildung verhalten.



Gliederung

1. Einfache Grafiken mit dem Turtle-Modul
2. Einführung in tkinter
3. Layoutmanager
4. Ereignisverarbeitung

Teil 3: Layoutmanager

Die grobe Struktur einer GUI wird durch die Konstruktoraufrufe mit der Konfiguration der master-Option festgelegt.

Die Layoutmanager werden verwendet, um die Widgets innerhalb der Benutzeroberfläche anzuordnen.

In Tkinter existieren 3 verschiedene Layoutmanager:

- Die Methode pack()
- Die Methode place()
- Die Methode grid()

3.1 Der Packer

Die Methode pack() haben wir bereits im letzten Abschnitt benutzt, um Widgets zu platzieren:

Widget definieren

```
titel = Label(master=fenster, text='Hallo Welt!',
              font=('Arial',20), fg='red')
```

```
titel.pack()
```

Widget platzieren

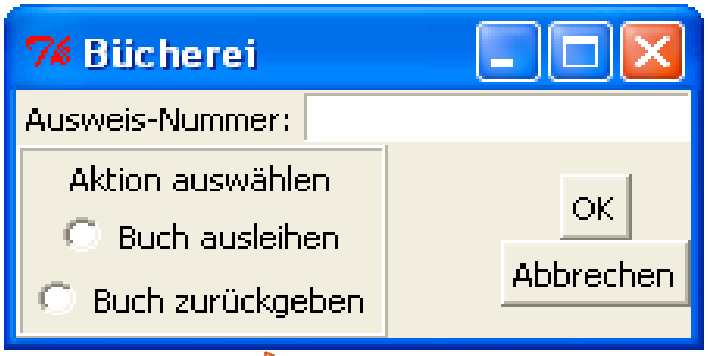
Gibt man keine Option an, werden alle Widgets horizontal zentriert und untereinander angeordnet. Das Fenster ist zu Beginn so groß, dass die Widgets gerade hineinpassen.

3.1 Der Packer - Optionen

Option	Erläuterung
`side`	Mögliche Werte: LEFT, RIGHT, TOP, BOTTOM. Das Widget wird an den linken oder rechten Rand bzw. nach oben oder unten gesetzt.
`anchor`	Das Widget wird an einer Ecke oder mittig an einer Seite platziert. Mögliche Werte sind die „Himmelsrichtungen“ N, NE, E, SE, S, SW, W, NW und CENTER.
`fill`	Mögliche Werte: X, Y, BOTH. Das Widget wird in waagrechter bzw. senkrechter Richtung vergrößert, sodass es die Größe des Masters erreicht.
`padx, pady`	Rechts und Links bzw. oberhalb und unterhalb des Widgets wird leerer Raum gelassen.
`expand`	Mögliche Werte: 0 oder 1. Gibt an, ob sich die Größe ändert, wenn sich die Fenstergröße ändert.

3.1 Der Packer - Beispiel

Die Oberfläche besteht aus 3 Frames, in denen die anderen Widgets angeordnet sind.



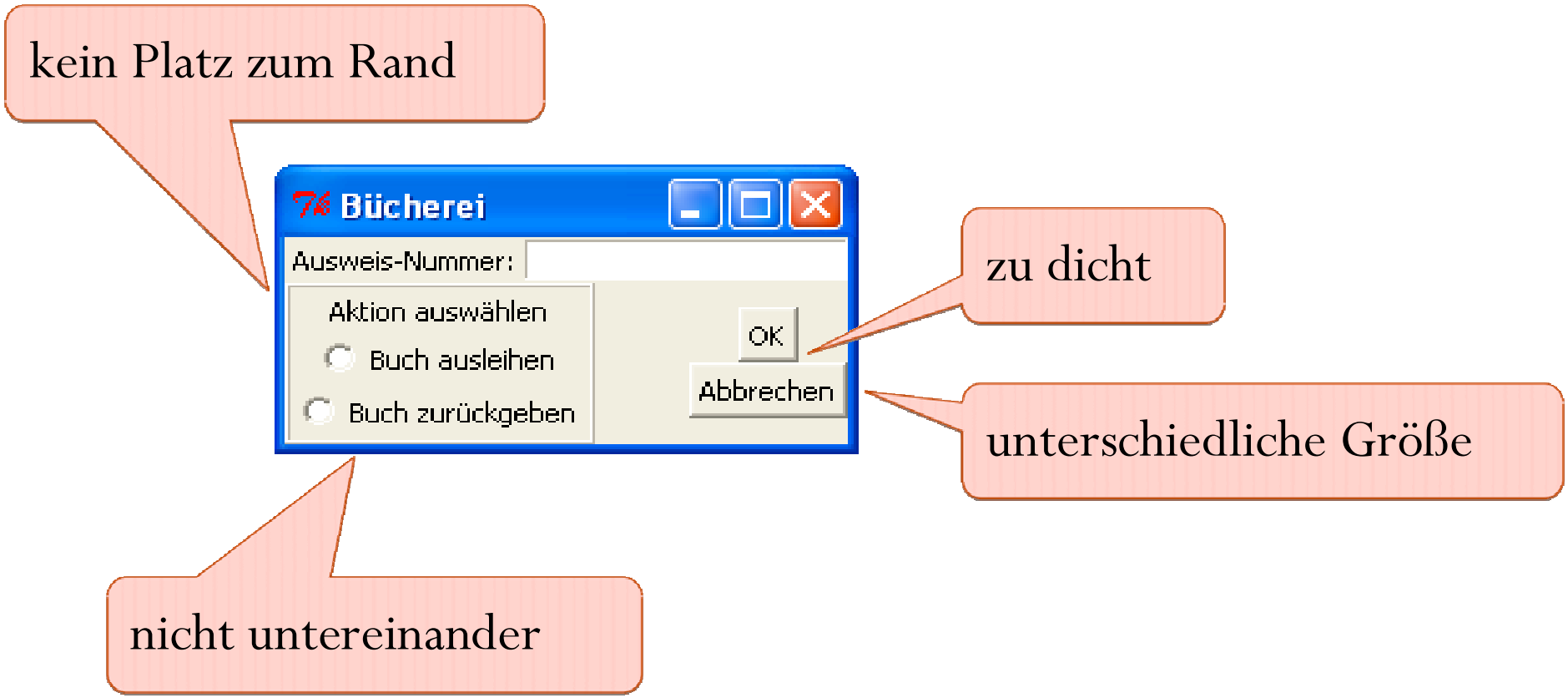

```
rahmenoben = Frame(fenster)
rahmenoben.pack(side=TOP)
```

```
rahmenrechts = Frame(fenster)
rahmenrechts.pack(side=RIGHT)
```

```
rahmenlinks=Frame(fenster,relief=RIDGE,bd=2)
rahmenlinks.pack(side=LEFT)
```

3.1 Der Packer - Beispiel

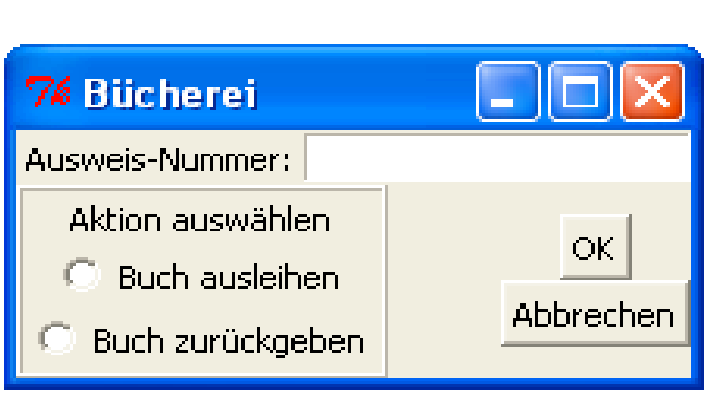
Doch so sollte eine GUI nicht aussehen…

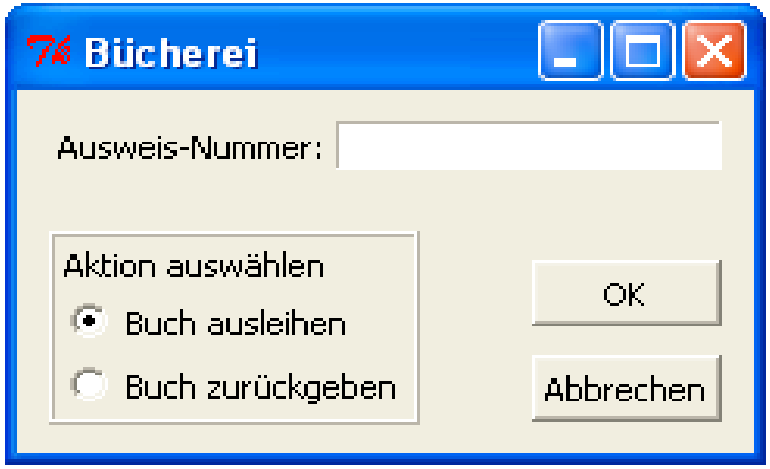


3.1 Der Packer - Beispiel

…aber so!

```
rahmenoben.pack(side=TOP, padx=10, pady=10)
```






```
aktionlabel.pack(anchor=W)
aktion1.pack(anchor=W)
aktion2.pack(anchor=W)
```

```
okbutton.pack(fill=X,pady=10)
abbutton.pack()
```

3.2 Layout mit place()

Mit der Methode place() lässt sich für jedes Widget die genaue Position innerhalb des Masters angeben.

Komplexere Oberflächen lassen sich hiermit gut realisieren, jedoch bedarf es einer sorgfältigen Planung.

Option	Erläuterung
`x, y`	Absolute Position des Widgets
`relx, rely`	Relative Position, zugelassen sind Werte von 0.0 bis 1.0
`anchor`	Gibt an, auf welchen Punkt sich die Koordinaten beziehen. Mögliche Werte: N, NE, E, SE, S, SW, W, NW, CENTER. Voreinstellung ist NW.

3.2 Layout mit place() - Beispiel

```
from tkinter import *
from random import *

def neuePosition():
    button.place(y=randint(50,350),x=randint(50,350))

fenster = Tk()
fenster.geometry('400x400')

button = Button(fenster, text='Klick mich!',
                command=neuePosition)
button.place(x=200,y=200,anchor=CENTER)

fenster.mainloop()
```

3.3 Raster-Layout mit grid()

Bei der Methode grid() wird das Master-Widget in ein Raster aus Zeilen und Spalten zerlegt, wie bei einer Tabelle.

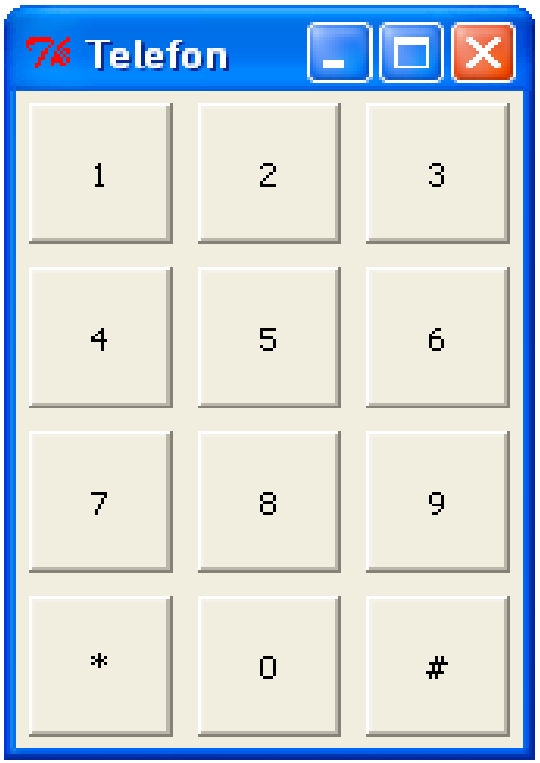
Für jedes Widget wird die Zeile und Spalte angegeben.

Die Spaltenbreite und Zeilenhöhe richtet sich nach dem jeweils größten Widget.

Option	**Erläuterung**
`row, column`	Zeile und Spalte, in der das Widget erscheint.
`padx, pady`	Leerer Platz rechts und links bzw. ober- und unterhalb
`sticky`	Mögliche Werte: N, NE, E, SE, S, SW, W, NW. Gibt an, wie das Widget innerhalb einer Zelle platziert wird, falls die Zelle größer als das Widget ist.

3.3 Raster-Layout - Beispiel

```
from tkinter import *

fenster = Tk()
fenster.title('Telefon')

tasten='123456789*0#'
for i in range(4):
    for j in range(3):
        b = Button(fenster,text=tasten[3*i+j],
                   width=6, height=3)
        b.grid(row=i,column=j,padx=4,pady=4)

fenster.mainloop()
```



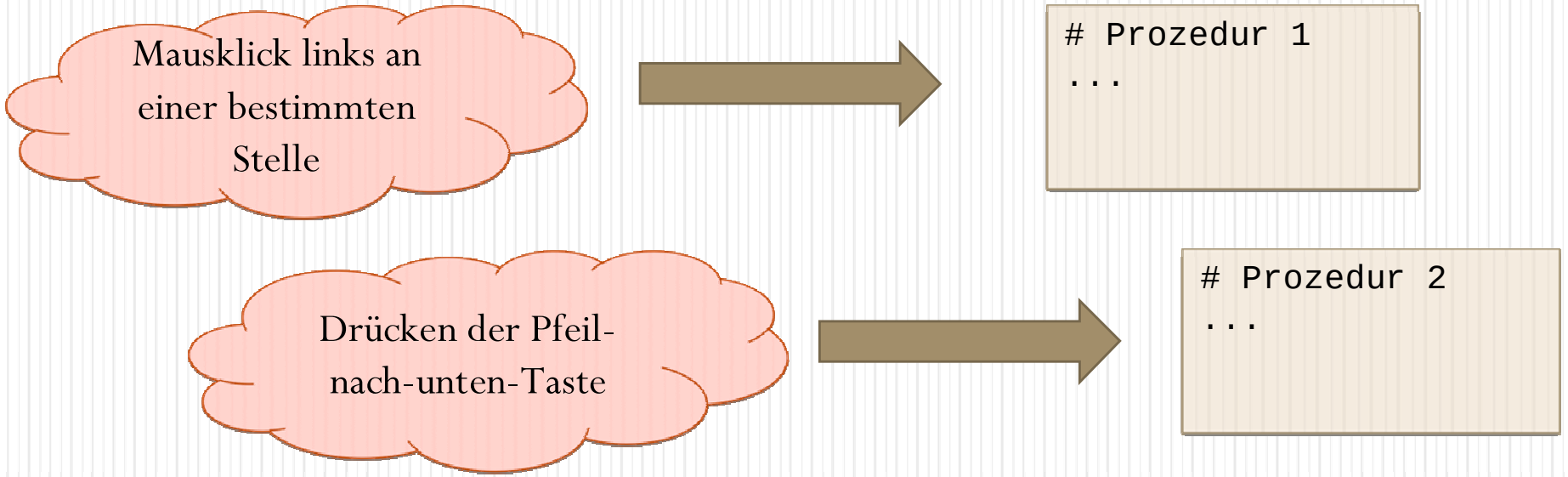
Gliederung

1. Einfache Grafiken mit dem Turtle-Modul
2. Einführung in tkinter
3. Layoutmanager
4. Ereignisverarbeitung

Teil 4: Ereignisverarbeitung

Grundprinzip bei Programmen mit GUI:

Ein Ereignis (Event) löst ein bestimmtes Verhalten aus, das in einer Prozedur (Eventhandler) definiert ist.



4.1 Ereignisverarbeitung

Bei dem Button-Widget haben wir bereits eine Ereignisverarbeitung kennen gelernt:

```
button = Button(fenster, text='klick mich!',
                command=danke)
```

Mit der command-Option wird beim Button-Widget der Eventhandler für das Event „Mausklick links“ festgelegt. In diesem Fall ist es die Prozedur `danke()`.

4.1 Ereignisverarbeitung

1. Was genau soll die Prozedur auslösen?

   Event mit Hilfe von *Eventsequenzen* spezifizieren

2. Wie soll reagiert werden?

   Eventhandler definieren

3. Welche Widgets sollen mit dem Eventhandler verknüpft werden?

   Widgets an Eventhandler *binden*

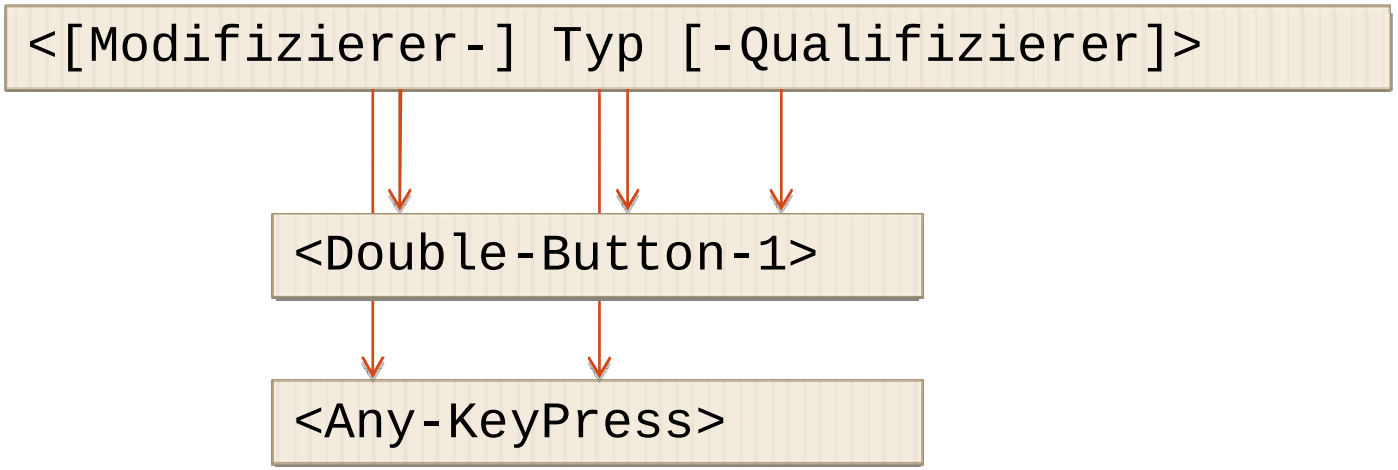
4.2 Events spezifizieren

Events werden durch Event-Sequenzen spezifiziert, z.B.:

Event	**Event-Sequenz**
Klick mit der linken Taste	`<Button-1>`
Klick mit der rechten Taste	`<Button-3>`
Doppelklick links	`<Double-Button-1>`
Die linke Maustaste wird losgelassen	`<ButtonRelease-1>`
Der Mauszeiger wird auf das Widget bewegt	`<Enter>`
Der Mauszeiger verlässt das Widget	`<Leave>`
Der Mauszeiger wurde innerhalb des Widgets bewegt	`<Motion>`
Die Pfeil-nach-links-Taste wird gedrückt	`<KeyPress-Left>`
Die F2-Taste wird losgelassen	`<KeyRelease-F2>`

4.2 Events spezifizieren

Allgemein sind Event-Sequenzen von der Form:



Mögliche Modifizierer sind etwa `Alt`, `Control`, `Shift`, `Double`, `Triple`, `Any`.

4.3 Eventhandler definieren

Eventhandler sind Prozeduren oder Methoden ohne Rückgabewert, die ein Event-Objekt übergeben bekommen:

```
def linksklick(event):
    ...
```

bzw.

```
def linksklick(self,event):
    ...
```

Je nach Event können verschiedene Attribute des Event-Objektes ausgewertet werden:

Attribut	**Erklärung**
x, y	Koordinaten des Mauszeigers bezogen auf die linke obere Ecke des Widgets
char	Zeichen, wenn das Event durch eine Taste ausgelöst wurde
num	Maustaste, wenn es sich um einen Mausklick handelt
widget	Referenz auf das Widget, durch das das Event ausgelöst wurde
time	Zeitwert in Millisekunden zur Bestimmung der Zeitspanne zwischen 2 Events

4.4 Events binden

Zum Binden eines Eventhandlers an ein Widget benutzt man die Methode bind():

```
widget.bind(sequence=event, func=f)
```

Mit Schlüsselwortargumenten:

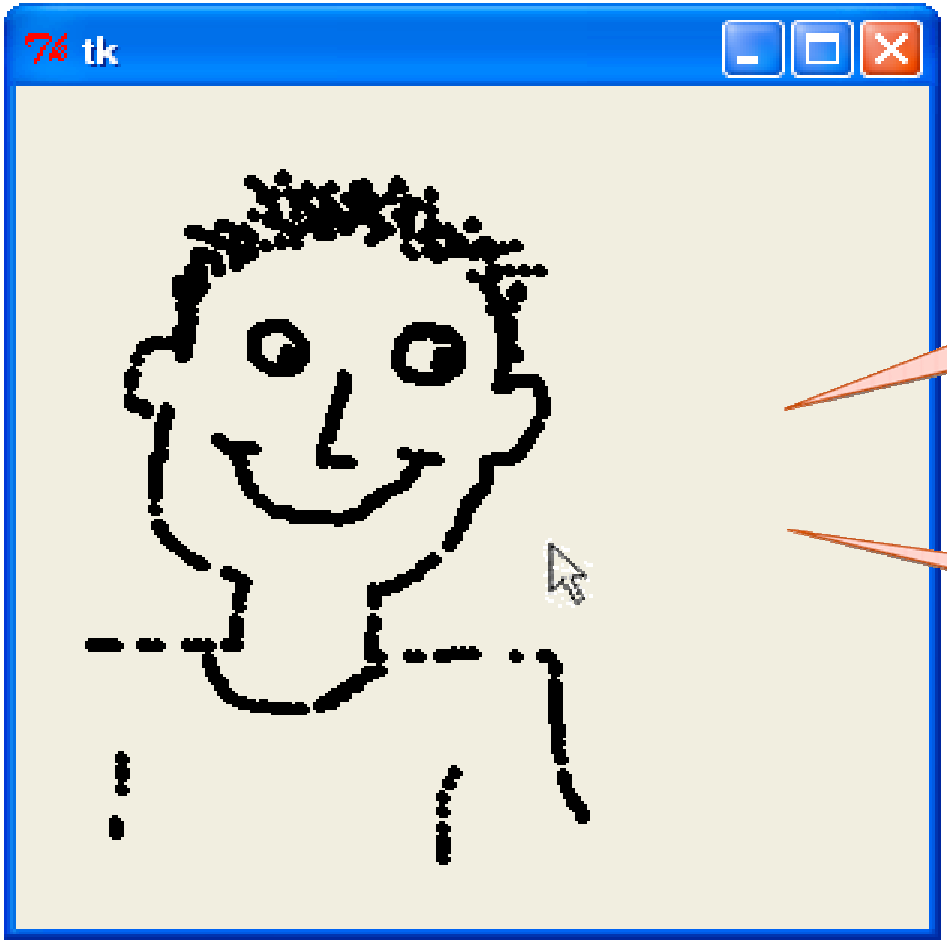
```
flaeche.bind(sequence='<Button-1>',func=klicken)
flaeche.bind(sequence='<Motion>',func=bewegen)
flaeche.bind(sequence='<ButtonRelease-1>',func=loslassen)
```

Mit Positionsargumenten:

```
flaeche.bind('<Button-1>',klicken)
flaeche.bind('<Motion>',bewegen)
flaeche.bind('<ButtonRelease-1>',loslassen)
```

4.5 Ereignisverarbeitung - Beispiel

Ein sehr einfaches Malprogramm:



bei Mausklick: ausgefüllten Kreis bei aktueller Position zeichnen

bei Mausbewegung: wenn Maustaste noch gedrückt ist, ausgefüllten Kreis bei aktueller Position zeichnen

4.5 Ereignisverarbeitung - Beispiel

```
from tkinter import *

class Malflaeche(object):

    def __init__(self,breite,hoehe):    #### Konstruktor ####
        self.fenster = Tk()
        self.flaeche = Canvas(self.fenster, width=breite,height=hoehe)
        self.flaeche.bind('<Button-1>',self.klicken)
        self.flaeche.bind('<Motion>',self.bewegen)
        self.flaeche.bind('<ButtonRelease-1>',self.loslassen)
        self.flaeche.pack()
        self.geklickt = False
        self.fenster.mainloop()

    def klicken(self,event):            #### Eventhandler Linksklick ####
        self.geklickt = True
        self.flaeche.create_oval(event.x,event.y,event.x+4,event.y+4,fill='black')

    def bewegen(self,event):            #### Eventhandler Mausbewegung ####
        if self.geklickt:
            self.flaeche.create_oval(event.x,event.y,event.x+4,event.y+4,fill='black')

    def loslassen(self,event):          #### Eventhandler Maustaste losgelassen ####
         self.geklickt = False

m = Malflaeche(300,300)
```

Zeichenfläche definieren

Eventhandler definieren

Events binden

4.6 Ereignisverarbeitung - Aufgabe

Erweitern Sie das Malprogramm:

Definieren Sie zusätzlich 4 Labels mit verschiedenen Farben.

Bei Mausklick auf eines der Label soll sich die Zeichenfarbe und der Fenstertitel ändern.

4.6 Ereignisverarbeitung - Lösung

```
from tkinter import *

class Malflaeche(object):

    def __init__(self,breite,hoehe):
        self.fenster = Tk()
        self.fenster.title('schwarzer Stift')
        self.flaeche = Canvas(self.fenster, width=breite,height=hoehe)
        self.flaeche.bind('<Button-1>',self.klicken)
        self.flaeche.bind('<Motion>',self.bewegen)
        self.flaeche.bind('<ButtonRelease-1>',self.loslassen)
        self.flaeche.pack()
        self.farbe = 'black'
        self.geklickt = False

        self.rot=Label(self.fenster,width=6,height=3,bg='red')
        self.gruen=Label(self.fenster,width=6,height=3,bg='green')
        self.blau=Label(self.fenster,width=6,height=3,bg='blue')
        self.schwarz=Label(self.fenster,width=6,height=3,bg='black')
        self.rot.bind('<1>',self.roterStift)
        self.gruen.bind('<1>',self.gruenerStift)
        self.blau.bind('<1>',self.blauerStift)
        self.schwarz.bind('<1>',self.schwarzerStift)
        self.rot.pack(side=LEFT)
        self.gruen.pack(side=LEFT)
        self.blau.pack(side=LEFT)
        self.schwarz.pack(side=LEFT)
        self.fenster.mainloop()

# ...Fortsetzung auf der der nächsten Seite
```

4.6 Ereignisverarbeitung - Lösung

```
    def klicken(self,event):
        self.geklickt = True
        self.flaeche.create_oval(event.x,event.y,event.x+4,event.y+4,
                                 fill=self.farbe,outline=self.farbe)

    def bewegen(self,event):
        if self.geklickt:
            self.flaeche.create_oval(event.x,event.y,event.x+4,event.y+4,
                                     fill=self.farbe,outline=self.farbe)

    def loslassen(self,event):
        self.geklickt = False

    def roterStift(self,event):
        self.farbe = 'red'
        self.fenster.title('roter Stift')

    def gruenerStift(self,event):
        self.farbe = 'green'
        self.fenster.title('gruener Stift')

    def blauerStift(self,event):
        self.farbe = 'blue'
        self.fenster.title('blauer Stift')

    def schwarzerStift(self,event):
        self.farbe = 'black'
        self.fenster.title('schwarzer Stift')


m = Malflaeche(300,300)
```

Literatur

- Michael Weigend: Objektorientierte Programmierung mit Python, 3. Auflage (mitp 2006)

  Die 4. Auflage „Objektorientierte Programmierung mit Python 3“ erscheint im Dezember 2009.