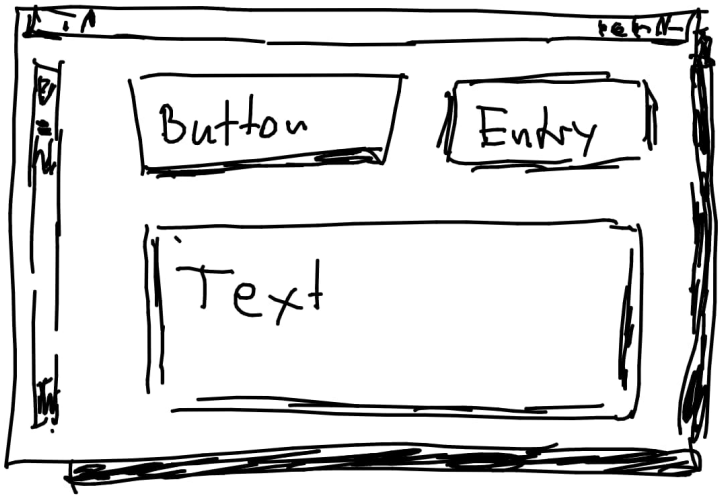


Kapitel 1 - EVA-Prinzip und Standardwidgets zur Eingabe/Ausgabe/Verarbeitung

B E G R I F F S K L Ä R U N G

Begrifflichkeiten:
GUI = *Grafical User Interface* (eine Benutzeroberfläche in einer Programmiersprache)
tkinter = *tool-kit-Interface* (Bibliothek, in der alle grafischen Elemente verwaltet werden)

Aufgaben einer Bibliothek, die grafische Elemente verwaltet:
1.) Definition grafischer Elemente (Button, Textfenster, Auswahlbox, etc.), in Python ***widgets*** genannt *(**window gadgets***)
2.) Anordnung der Elemente über einen ***Layout-Manager*** (so dass die Elemente nach z.B. Größenänderung des Fensters immer noch am selben Platz sind)
3.) Bereitstellen der Interaktion des Benutzers mit dem grafischen Element (***Eventhandling - Ereignisverarbeitung***)

Ü B E R S I C H T

Klasse	Beschreibung

Entry	**E**ingabe → Eingabefeld
	`EingabeFeld = tkinter.Entry(Fenster)`
	`EingabeFeld.grid(row=1, column = 0, padx = 5, pady = 5)`
	`Zahl = int(EingabeFeld.get())` *oder* `Wort = EingabeFeld.get()`

Button	**V**erarbeitung → Button mit Prozedur, die bei Click ausgeführt wird
	`But = tkinter.Button(Fenster, text = " ", command = ButClick)`
	`But.grid(row = 1, column = 2, padx = 5, pady = 5)`

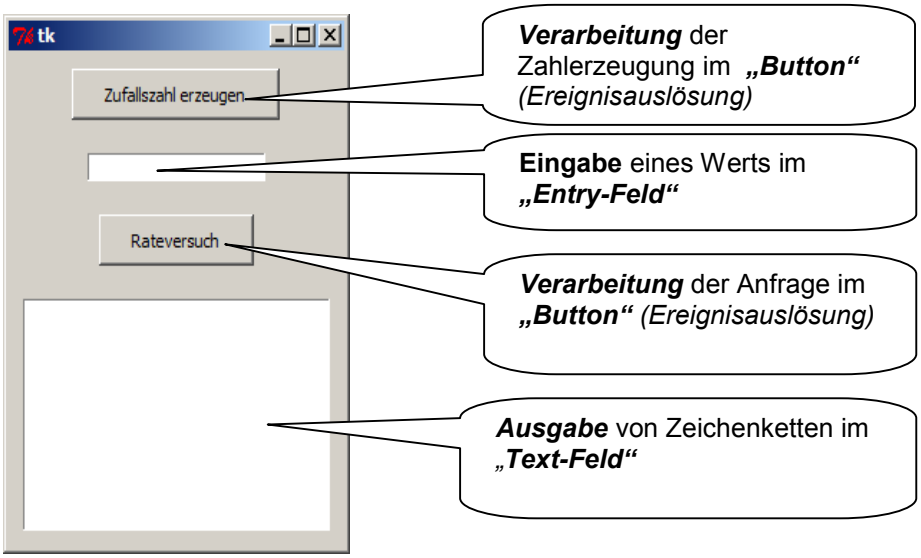
Text	**A**usgabe → Ausgabefeld für Texte
	`AusgabeFeld = tkinter.Text(Fenster, width = 30, height = 15)`
	`AusgabeFeld.grid(row = 3, column = 0, padx = 5, pady = 5)`
	`AusgabeFeld.delete('1.0', "end")`
	`AusgabeFeld.insert("end"," Ausgabetext \n")`

M O T I V A T I O N

Das Zahlenratspiel
Der Rechner „denkt“ sich eine Zahl zwischen 1 und 100.

Der Spieler tippt so lange Zahlen, bis gedachte und getippte Zahl übereinstimmen. Als Hinweis gibt der Rechner bei jeder getippten Zahl aus, ob seine gedachte Zahl größer oder kleiner als die getippte Zahl ist.

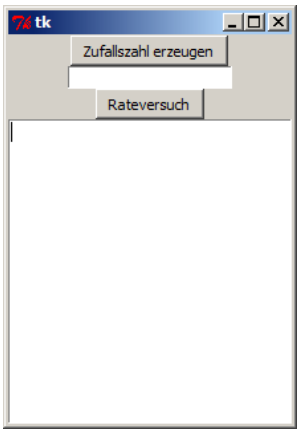
1. Schritt – Die GUI (graphical user interface)



I) Ein "leeres" Fenster als Grundlage

`import tkinter`
 	
`#GUI Anfang`
`Fenster = tkinter.Tk()`
`#GUI Ende`
 	
`Fenster.mainloop()`
	Das Modul, das die grafische Oberfläche bereitstellt heißt "tkinter"
	
	Tk() erzeugt ein Fenster, dessen Variablenname "fenster" lautet.
	
	Ein Schleife prüft ständig, ob Aktionen (z.B. Clicks, Schließen über das Kreuz, Minimieren) für das Fenster ausgeführt werden müssen.

IIa) Im Fenster werden Ein- / Ausgabe und Anzeigeelemente untergebracht



Das Layout (Anordnung der visuellen Objekte) erfolgt über den "grid-Layout-Manager".

column (Spalten)

row (Zeilen)	**column=0**	**column=1**	**column=2**
row=0	*column=0, row=0*	*column=1, row=0*	
row=1	*column=0, row=1*	*column=1, row=1*	
row=2			
row=3			
row=4	*column=0, row=4,* ***columnspan=3***		
row =5			
row = 6		*column=1,row=6,* ***columnspan=2***	

```
#GUI Anfang
Fenster = tkinter.Tk()

ButtonZufall = tkinter.Button(Fenster, text = "  Zufallszahl erzeugen  ")
ButtonZufall.grid(row=0, column=0)

EingabeFeld = tkinter.Entry(Fenster)
EingabeFeld.grid(row=1, column = 0)

ButtonRaten = tkinter.Button(Fenster, text = "  Rateversuch  ")
ButtonRaten.grid(row=2, column = 0)

AusgabeFeld = tkinter.Text(Fenster, width = 30, height = 15)
AusgabeFeld.grid(row = 3, column = 0)
#GUI Ende

Fenster.mainloop()
```

Anmerkung: Da das grid-Layout universeller und leichter zu verstehen ist als ein evtl. geschachteltes pack-Layout, wird im Folgenden nur mit dem grid-Layout geabeitet.

IIb) Ein bisschen Kosmetik – Abstand vom Fensterrand


```
#GUI Anfang
Fenster = tkinter.Tk()

ButtonZufall = tkinter.Button(Fenster, text = "  Zufallszahl erzeugen  ")
ButtonZufall.grid(row=0, column=0, padx = 5, pady = 5)

EingabeFeld = tkinter.Entry(Fenster)
EingabeFeld.grid(row=1, column = 0, padx = 5, pady = 5)

ButtonRaten = tkinter.Button(Fenster, text = "  Rateversuch  ")
ButtonRaten.grid(row=2, column = 0, padx = 5, pady = 5)

AusgabeFeld = tkinter.Text(Fenster, width = 30, height = 15)
AusgabeFeld.grid(row = 3, column = 0, padx = 5, pady = 5)
#GUI Ende
```

"padx = 10" schafft eine Rahmen von jeweils 10 Pixel links und rechts vom z.B. Button,
"pady = 5" schafft eine Rahmen von jeweils 5 Pixel oben und unten vom z.B. Button.

2. Schritt – Die algorithmische Implementierung

I) Zufallszahl erzeugen

```
import random
import tkinter
random.seed()
```

```
Zufallszahl = 0

def ButtonZufallClick():
   global Zufallszahl
   Zufallszahl = random.randint(1,100)
   AusgabeFeld.delete('1.0', "end")

#    Da die Zufallszahl eine globale Variable ist, muss in der
#    Prozedur deklariert werden, dass auf eine globale Variable zugegriffen
#    werden soll. global Zufallszahl
#    Sonst würde einfach eine lokale Variable mit dem Namen "Zufallszahl"
#    angelegt und benutzt.

#GUI Anfang
Fenster = tkinter.Tk()

ButtonZufall = tkinter.Button(Fenster,
        text = "  Zufallszahl erzeugen  ", command = ButtonZufallClick)
ButtonZufall.grid(row=0, column=0, padx = 5, pady = 5)
```

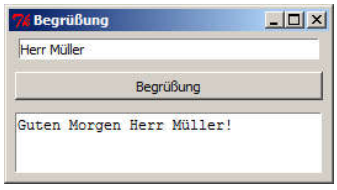
II) Nun fehlt nur noch die Implementierung der Funktion "Rateversuch":

```
def ButtonRatenClick():
   Ratezahl = int(EingabeFeld.get())
   if (Ratezahl < Zufallszahl) :
      AusgabeFeld.insert("end","Die Zufallszahl ist größer\n")
   elif (Ratezahl > Zufallszahl) :
      AusgabeFeld.insert("end", "Die Zufallszahl ist kleiner \n")
   elif  (Ratezahl == Zufallszahl):
      AusgabeFeld.insert("end", " TREFFER \n")

#GUI Anfang
[ . . .]

ButtonRaten = tkinter.Button(Fenster,
 text = "  Rateversuch  ", command = ButtonRatenClick)
ButtonRaten.grid(row=2, column = 0, padx = 5, pady = 5)

[ . . .]
#GUI Ende
```

Ü B U N G E N

Aufgabe 1
Erweitere das Programm, dass die Feldgröße über ein Eingabe-Feld eingelesen wird.

Aufgabe 2
a) Schreibe ein kleines Begrüßungsprogramm, das einen Namen einliest und daraus einen Willkommenstext gestaltet.



b) Passe das Layout so an, dass alle Komponenten gleich breit sind.

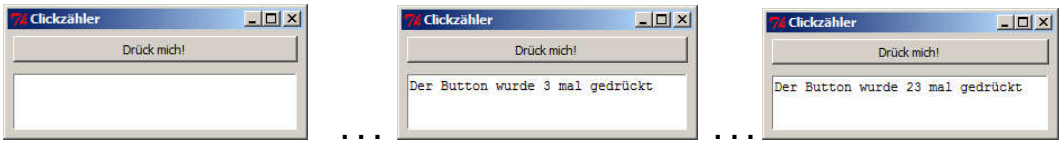
Hinweis:
1) Zusammenfügen von Zeichenketten über "+"

```
" Guten " + " Morgen " + " Herr " + " Müller"
```

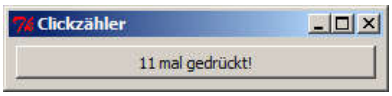
2) Umwandeln einer Zahl in eine Zeichenkette

```
"Der Button wurde " + str(AnzahlClick)+ " mal gedrückt"
```

Aufgabe 3
a) Schreiben Sie ein Programm das zählt, wie oft ein Button gedrückt wurde.
b) Erweitern Sie das Programm um einen Reset-Button, mit dem der Zähler wieder auf 0 zurückgesetzt werden kann.



c) Erweitere das Programm so, dass die Ausgabe des Zählers direkt auf den Button geschrieben wird:



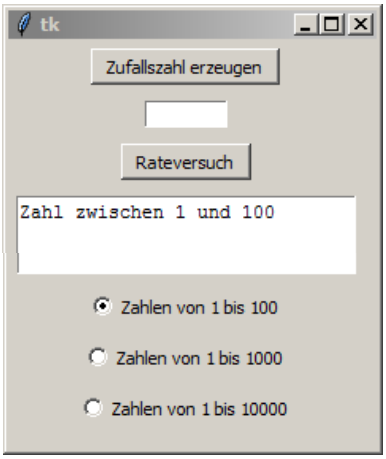
Aufgabe 4
Entwerfen Sie den Taschenrechner.
a) Implementieren Sie zuerst nur die GUI,
b) dann den algorithmischen Kern.

Kapitel 2: Auswahlfelder (Radiobutton)

Ü B E R S I C H T

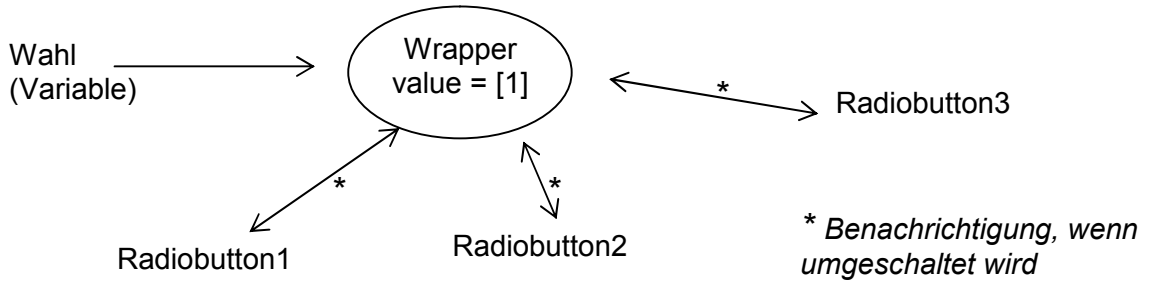
Radiobutton	**Radiobutton** → Auswahlfelder, bei denen immer nur eine Auswahl möglich ist
	
	`if Wahl.get()==1:`
	`  # ----------`
	`if Wahl.get()==2:`
	`  # ----------`
	
	`Wahl = tkinter.IntVar()`
	
	`RadioButton1 = tkinter.Radiobutton(Fenster, text="Auswahl 1", variable = Wahl, value = 1)`
	`RadioButton1.grid(padx=5, pady=5, row = 4)`
	
	`RadioButton2 = tkinter.Radiobutton(Fenster, text="Auswahl 2", variable = Wahl, value = 2)`
	`RadioButton2.grid(padx=5, pady=5, row = 5)`
	
	Erläuterung: `IntVar` ist eine sog. Wrapper-Klasse, die eine Variable (hier: Integer) kapselt und speichert, wenn die Variable verändert wurde.
	`Wahl = tkinter.IntVar()` erzeugt ein Objekt dieser Klasse mit der Referenzvariablen "Wahl".
	Ein Radiobutton-Widget wird beim Erstellen mit diesem Objekt verknüpft und erhält die Information, wenn ein Auswahlbutton angeklickt worden ist. Dann nämlich muss das Widget den Punkt in dem Kreis umschalten, gerade auch, wenn ein anderer Radioknopf betätigt worden ist (Markierung ausschalten).
	Über `Wahl.get()` werden die Inhalte des Wrapper-Objekts (also der Variablenwert) ausgelesen. `Wahl.set()` würde die Variable setzen.

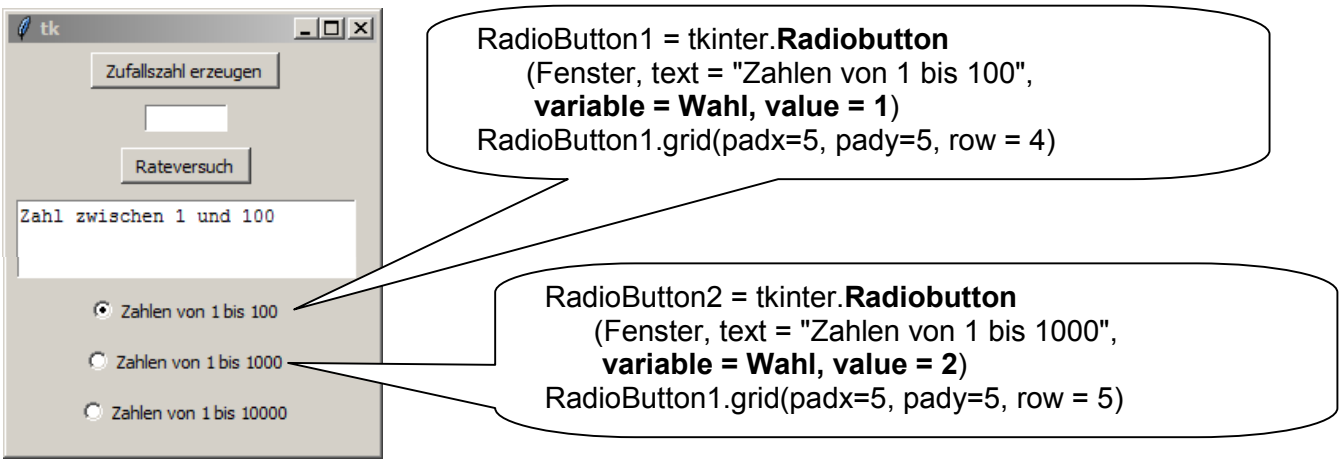
M O T I V A T I O N



Zahlenraten soll so erweitert werden, dass die Zahlbereiche über Auswahlfelder eingegeben werden können.

Notwendig ist dazu eine Wrapper-Klasse, die eine Variable (hier: Integer) kapselt und speichert, wenn die Variable verändert wurde. Ein Radiobutton-Widget wird beim Erstellen mit diesem Objekt verknüpft und erhält die Information, wenn ein anderer Auswahlbutton angeklickt worden ist. Dann muss z.B. die Markierung geändert werden.






```
Wahl = tkinter.IntVar()   #Wrapper Klasse

RadioButton1 = tkinter.Radiobutton (Fenster, text = "Zahlen von 1 bis 100",
          variable = Wahl, value = 1)
RadioButton1.grid(padx=5, pady=5, row = 4)

RadioButton2 = tkinter.Radiobutton  (Fenster, text = "Zahlen von 1 bis 1000",
      variable = Wahl, value = 2)
RadioButton2.grid(padx=5, pady=5, row = 5)

RadioButton3 = tkinter.Radiobutton (Fenster, text = "Zahlen von 1 bis 10000",
      variable = Wahl, value = 3)
RadioButton3.grid(padx=5, pady=5, row = 6)
```

Kapitel 3 - Sortieralgorithmen mit ListBox und ScrollBar

Ü B E R S I C H T

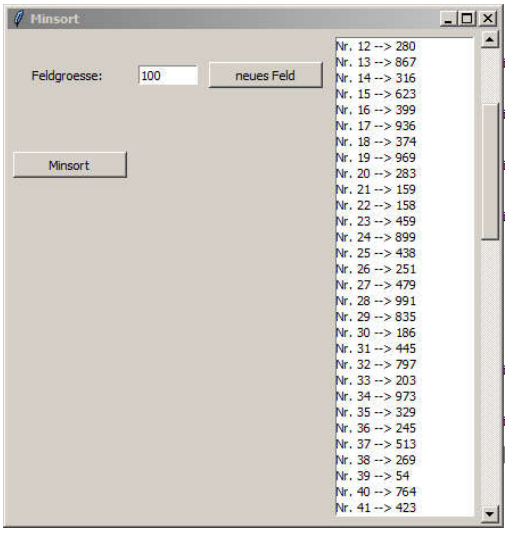
ListBox	Der Vorteil einer Listbox ist, dass über eine zusätzliche ScrollBar navigiert werden kann und dass auf die Zeileninhalte per Mausklick zugegriffen werden kann
	
	`ListBox1 = tkinter.Listbox (Fenster, width=20, height = 30)`
	`ListBox1.grid(padx = 5, pady = 5, row = 1, column = 4, columnspan = 1, rowspan=8)`
	
	Zeilen löschen:
	`ListBox1.delete("0","end")`
	
	Zeilen anhängen, z.B. aus einer Zahlenliste:
	`ListBox1.insert("end", str(a[i]))`
	`ListBox1.insert("end", "Hallo Welt")`
	`ListBox1.insert("Hallo Welt")`

ScrollBar	Scrollbar erstellen: ("sticky", damit die Scrollbar von oben nach unten über die 8 Zeilen reicht)
	
	`scrollbar = tkinter.Scrollbar(Fenster)`
	`scrollbar.grid(row = 1, column = 5, rowspan = 8,  sticky="NS")`
	
	Verknüpfung von Scrollbar und Listbox herstellen:
	1.) Die Scrollbar an die Listbox binden, damit ein Bewegen des Scrollbalkens den Inhalt der Listbox bewegt
	`scrollbar.config(command=ListBox1.yview)`
	
	2.) Die Listbox an die Scrollbar binden, damit sich der Balken mitbewegt, wenn in der Listbox mit der Maus gescrolled wird
	`ListBox1.config(yscrollcommand=scrollbar.set)`

Clickereignis der Listbox	Ein Clickereignis wird zunächst an die Liste gebunden
	
	`def Liste1Click(event):`
	`  [. . .}`
	
	`#GUI`
	`Liste1 = tkinter.Listbox (width=30, height = 10)`
	`Liste1.bind('<<ListboxSelect>>', Liste1Click)`
	`# Die doppelten eckigen Klammern sind richtig`
	
	Dann kann die markierte Zeilennummer
	
	`def Liste1Click(event):`
	`  Tupel = Liste.curselection()`
	`  index = int(Tupel[0])`
	
	oder der markierten Text der Zeile ausgelesen werden
	
	`def Liste1Click(event):`
	`  Tupel = Liste1.curselection()`
	`  index = int(Tupel[0])`
	`  s = Liste1.get(index)`

M O T I V A T I O N

Es soll ein Sortieralgorithmus (hier: Minsort) implementiert werden, der Zahlen in einer Listbox zufällig erzeugt und dann sortiert. Die Listbox soll bequem über eine Scrollbar bedienbar sein.



1.) Erstmal die GUI --> eine Listbox ohne Scrollbar

```
Fenster = tkinter.Tk()
Fenster.title("Minsort")

[. . .]

ListBox1 = tkinter.Listbox (width=20, height = 30)
ListBox1.grid(padx = 5, pady = 5, row = 1, column = 4, columnspan = 1, rowspan=8)

# ---------------------
Fenster.mainloop()
```

2.) Zufallsfeld erzeugen → ohne sortieren

```
a = []
for i in range(100000):
  zufall = random.randint(1,1000)
  a.append(zufall)

n = 100

def neuesFeld():
   global a
   global n
   n = int(EingabeGroesse.get())
   for i in range(n):
      a[i] = random.randint(1,1000)
   feldAusgeben()

def feldAusgeben():
   ListBox1.delete("0","end")
   for i in range(n):
      ListBox1.insert("end", "Nr. " + str(i)
        + " --> " + str(a[i]))


   global a
```

```
   global n
   ListBox1.delete("0","end")
   for i in range(n):
      ListBox1.insert("end", "Nr. " + str(i) + " --> " + str(a[i]))
```

3.) Listbox erstellen und Verknüpfung von Scrollbar und Listbox herstellen:

```
scrollbar = tkinter.Scrollbar(Fenster)
scrollbar.grid(row = 1, column = 5, rowspan = 8,  sticky="NS")
```

"sticky", damit die Scrollbar von oben nach unten über die 8 Zeilen reicht

Die Scrollbar an die Listbox binden, damit ein Bewegen des Scrollbalkens den Inhalt der Listbox bewegt:

```
scrollbar.config(command=ListBox1.yview)
```

Die Listbox an die Scrollbar binden, damit sich der Balken mitbewegt, wenn in der Listbox mit der Maus gescrolled wird:

```
ListBox1.config(yscrollcommand=scrollbar.set)
```

4.) Minsort

```
def minsort():
   index = 0
   minpos = 0
   hilf = 0
   global a
   for index in range (n):
      minpos = index
      for i in range (index, n):
          if (a[i] < a[minpos]):
            minpos = i
      hilf =a[index]
      a[index] =a[minpos]
      a[minpos] =hilf

   feldAusgeben()
```

Ü B U N G E N

Aufgabe 1
In einer Listbox werden Zufallszahlen angezeigt. Beim Klick auf eine Zeile der Listbox wird gezählt, wie oft die Zahl im Feld vorkommt.

Hinweise zur Aufgabe:

I) Clickereignis an die Liste binden:

```
def Liste1Click(event):
     [. . .}

#GUI

Liste1 = tkinter.Listbox (width=30, height = 10)
Liste1.bind('<<ListboxSelect>>', Liste1Click)

# Die doppelten eckigen Klammern sind richtig
```

II) Markierte Zeilennummer auslesen:

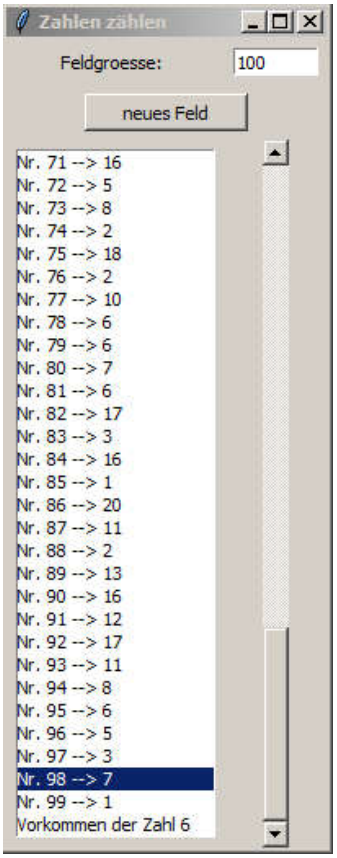
```
def Liste1Click(event):
     Tupel = Liste.curselection()
     index = int(Tupel[0])
```

III) Markierten Text auslesen:

```
def Liste1Click(event):
   Tupel = Liste1.curselection()
   index = int(Tupel[0])
   s = Liste1.get(index)
```

IV) Befüllen:

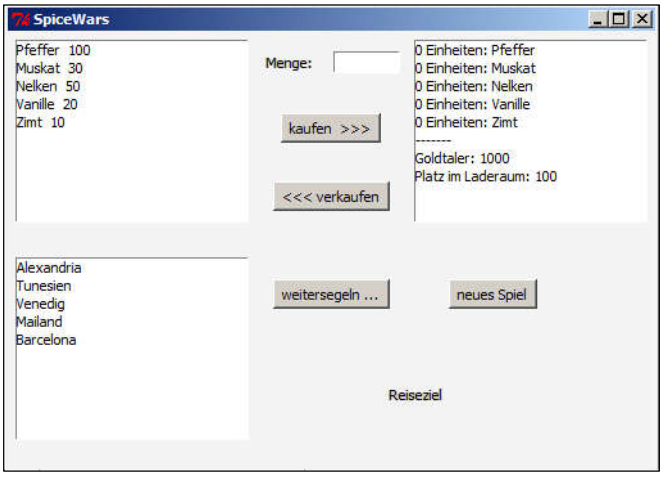
```
Liste1.delete("0","end")
Liste1.insert(" Beispieltext"))
```



Aufgabe 2

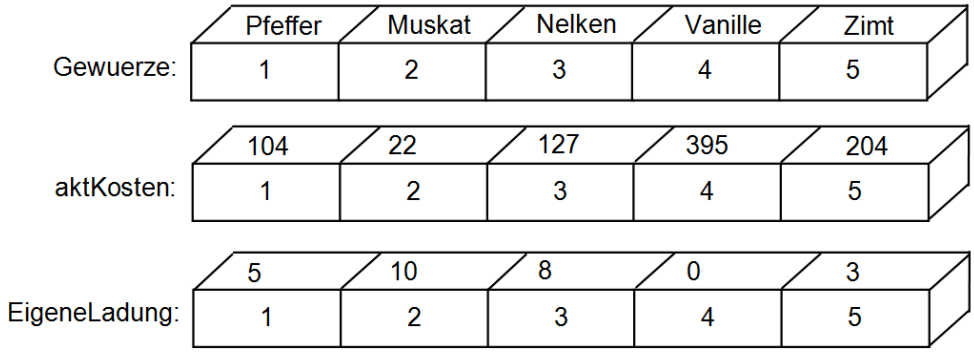
SpiceWars ist ein *Handelssimulationsspiel*. Der Spieler befindet sich im fiktiven 18. Jhdt und betreibt ein kleines Handelsunternehmen. Mit einer Kogge segelt er zwischen einigen Städten des Mittelmeers hin und her, kauft Gewürze an einem Ort und verkauft diese wieder woanders.
Nach jeder Segeltour fallen und steigen die Preise. Spielgeschehen und Spielorte sind nicht historisch nachempfunden.

Die Grundversion des Spiels orientiert sich an Handelsspielen wie "Hanse" oder "Kaiser" und ordnet sich in die Liste der "rundenbasierten Spiele" ein. Während die Welle der Handelsspiele in den 90er Jahren mittlerweile abgeebbt ist, erleben Spiele dieser Art auf mobilen Plattformen wie Android ihre Renaissance.



Oftmals handelt es sich allerdings um Spiele mit antisozialem Hintergrund, wie z.B. "DopeWars", das sich von der MS DOS Version bis hin auf die Android-Plattform über die Jahre gerettet hat. Als Drogendealer schlägt sich der Spieler dabei durch den Untergrund New Yorks. An dieser Stelle muss (auch im Unterricht) deutlich darauf hingewiesen werden, dass Spiele mit antigesellschaftlichem, drogen- oder gewaltverherrlichendem Hintergrund nicht tolerabel und abzulehnen sind.

Datenhaltung:



Schritt 1: Die Daten und Anzeige der Daten
Die Daten (Gewürze, Preise und Ladungsmenge) werden in Arrays abgespeichert. Die entsprechenden Indizes markieren ein Gewürz.
Damit ein Gewürz bequem mit der Maus auswählbar ist, und die maximal zu kaufende und verkaufende Menge automatisch berechnet und angezeigt wird, wird zur Darstellung der Arrayinhalte eine Listen anzeigende visuelle Komponente gewählt (Lazarus: ListBox / Java: JList / Python: Listbox), die den Index des gewählten Gewürzes auslesen lässt.

Die Größe des Laderaums (Mengenbegrenzung) und das zur Verfügung stehende Geld lässt sich über eine Integer-Variable nachhalten.

Das Anzeigen der Daten regelt eine Prozedur. Eine Anfangsbelegung der Daten erfolgt im Konstruktor.

```
import tkinter, random, math

Geld = 1000
Laderaum = 100
Gewuerze = ["Pfeffer", "Muskat", "Nelken", "Vanille", "Zimt"]
aktKosten = [100, 30, 50 , 20 ,10]
EigeneLadung = [0,0,0,0,0]



def DisplayAktualisieren():
  global Gewuerze
  global aktKosten
  global EigeneLadung
  global Geld
  global Laderaum
  Liste.delete("0","end")
  Liste.insert("end", Gewuerze[0] + "  " + str(aktKosten[0]))
  Liste.insert("end", Gewuerze[1] + "  " + str(aktKosten[1]))
  Liste.insert("end", Gewuerze[2] + "  " + str(aktKosten[2]))
  Liste.insert("end", Gewuerze[3] + "  " + str(aktKosten[3]))
  Liste.insert("end", Gewuerze[4] + "  " + str(aktKosten[4]))

  ListeLaderaum.delete("0","end")
  ListeLaderaum.insert("end", str(EigeneLadung[0]) +
      ← " Einheiten: " + str(Gewuerze[0]))
  ListeLaderaum.insert("end", str(EigeneLadung[1]) +
      ←  " Einheiten: " + str(Gewuerze[1]))
  ListeLaderaum.insert("end", str(EigeneLadung[2]) +
      ←  " Einheiten: " + str(Gewuerze[2]))
  ListeLaderaum.insert("end", str(EigeneLadung[3]) +
      ←  " Einheiten: " + str(Gewuerze[3]))
  ListeLaderaum.insert("end", str(EigeneLadung[4]) +
      ←  " Einheiten: " + str(Gewuerze[4]))
```

```
  ListeLaderaum.insert("end", "-------")
  ListeLaderaum.insert("end", "Goldtaler: " + str(Geld))
  ListeLaderaum.insert("end", "Platz im Laderaum: " + str(Laderaum))

  ListeStaedte.delete("0","end")
  ListeStaedte.insert("end", "Alexandria")
  ListeStaedte.insert("end", "Tunesien")
  ListeStaedte.insert("end", "Venedig")
  ListeStaedte.insert("end", "Mailand")
  ListeStaedte.insert("end", "Barcelona")



def NeuesSpiel():
  global aktKosten
  global EigeneLadung
  global Geld
  global Laderaum
  aktKosten = [100, 30, 50 , 20 ,10]
  EigeneLadung = [0,0,0,0,0]
  Geld = 1000
  Laderaum = 100
  DisplayAktualisieren()



#GUI ----------------------------------
#Hauptfenster
Fenster = tkinter.Tk()
Fenster.title("SpiceWars")

#-------------------------
# Die fett kursiven Ereignisverknüpfungen müssen ohne Prozedur
# erstmal noch rausgelassen werden
#-------------------------
Liste = tkinter.Listbox (width=30, height = 10)
Liste.grid(padx = 5, pady = 5, row = 1, column = 1,
     ← columnspan = 1, rowspan=3)
Liste.bind('<<ListboxSelect>>', Liste1Click)

LabelMenge = tkinter.Label(Fenster, text = 'Menge: ')
LabelMenge.grid (padx = 5, pady=5, row = 1, column = 2)

EingabeMenge = tkinter.Entry(Fenster, width = 8)
EingabeMenge.grid(padx = 5, pady = 5, row = 1, column = 3)

ButtonKaufen = tkinter.Button
   ← (Fenster, text=' kaufen  >>> ', command = kaufen)
ButtonKaufen.grid(padx=5, pady = 5, row =2, column = 2, columnspan=2)
ButtonVerkaufen = tkinter.Button
   ← (Fenster, text=' <<< verkaufen ', command = verkaufen)
ButtonVerkaufen.grid(padx=5, pady = 5, row =3, column = 2, columnspan=2)

ListeLaderaum = tkinter.Listbox (width=30, height = 10)
ListeLaderaum.grid(padx = 5, pady = 5, row = 1, column = 5,
   ← columnspan = 2, rowspan=3)
ListeLaderaum.bind('<<ListboxSelect>>', Liste2Click)



#-----------------------
ListeStaedte = tkinter.Listbox (width=30, height = 10)
ListeStaedte.grid(padx = 5, pady = 20, row = 4, column = 1,
columnspan = 1, rowspan=3)

ButtonBewegen = tkinter.Button(Fenster, text = ' weitersegeln ... ',
   ←  command = Weitersegeln)
ButtonBewegen.grid(row=4, column=2, padx=5, pady=25, columnspan = 2)
```

```
LabelReiseziel = tkinter.Label(Fenster, text = 'Reiseziel ')
LabelReiseziel.grid (padx = 5, pady=5, row = 5, column = 2,
  ← columnspan = 4)

ButtonNeustart = tkinter.Button(Fenster, text = ' neues Spiel ',
   ← command = NeuesSpiel)
ButtonNeustart.grid(row=4, column=5, padx=5, pady=25)

NeuesSpiel()
# ---------------------
Fenster.mainloop()
```

Schritt 2: Kaufs- und Verkaufsmenge wählen
Die Menge der zu kaufenden oder zu verkaufenden Güter wird in einem "Edit-Feld" angezeigt. Um die Bedienung komfortabel zu gestalten, soll beim Klicken auf ein Gewürz gleich die maximal kaufbare bzw. verkaufbare Menge in dieses Feld eingetragen werden.

```
# Die Listenauswahl ist ein Tupel (für Mehrfachauswahl),
# aus dem des erste Element (index) ausgelesen werden muss
def Liste1Click(event):
     Tupel = Liste.curselection()
     index = int(Tupel[0])
     EingabeMenge.delete(0, "end")
     EingabeMenge.insert(0, math.floor(Geld/aktKosten[index]))

def Liste2Click(event):
     Tupel = ListeLaderaum.curselection()
     index = int(Tupel[0])
     if (index < 4):
        EingabeMenge.delete(0, "end")
        EingabeMenge.insert(0, EigeneLadung[index])
```

Schritt 3 – Kaufen und Verkaufen
Beim Kauf wird dabei die Anzahl der Gewürzeinheiten mit den Kosten multipliziert und geprüft, ob das Geld ausreicht und ob genug Laderaum zur Verfügung steht. Beim Verkauf wird geprüft, ob die zu verkaufende Menge im Laderaum überhaupt vorhanden ist.

```
def kaufen():
  global Geld
  global aktKosten
  global EigeneLadung
  global Laderaum
  Anzahl = int(EingabeMenge.get())
  Nummer = int(Liste.curselection()[0])
  if (Laderaum >= Anzahl) and (Geld >= aktKosten[Nummer]*Anzahl):
      Laderaum = Laderaum - Anzahl
      Geld = Geld - int(aktKosten[Nummer])*Anzahl
      EigeneLadung[Nummer] = EigeneLadung[Nummer]+Anzahl
  DisplayAktualisieren()


def verkaufen():
  global Geld
  global aktKosten
  global EigeneLadung
  global ListeLaderaum
  global Laderaum
  Anzahl = int(EingabeMenge.get())
  Nummer = int(ListeLaderaum.curselection()[0])
  if (Anzahl <= EigeneLadung[Nummer]):
      Laderaum = Laderaum + Anzahl
      Geld = Geld + int(aktKosten[Nummer])*Anzahl
      EigeneLadung[Nummer] = EigeneLadung[Nummer] - Anzahl
  DisplayAktualisieren()
```

4. Schritt - Weitersegeln

Beim "Weitersegeln" zu einer neuen Stadt ändern sich die Preise. Tatsächlich ist es jedoch egal, wo der Spieler hin segelt, da die Preise unabhängig von einem Ort zufällig ermittelt werden. (Wäre das nicht so, würde ein Spieler das Gewürzgefälle eines Gewürzes zwischen zwei Orten ausnutzten und nur mit einem Gewürz in zwei verschiedenen Orten handeln.)

Die neuen Gewürzkosten richten sich nach einem vorher festgelegten Maximal bzw. Minimalwert.

```
KostenMin = [50, 5 , 20 , 100, 60]
KostenMax = [150, 35 , 170 , 450, 300]


def Weitersegeln():
  global ListeStaedte
  global aktKosten
  global KostenMin
  global KostenMax
  Tupel = ListeStaedte.curselection()
  index = int(Tupel[0])
  if (index != 0):
    Stadt = ListeStaedte.get(index)
    LabelReiseziel.config(text="Ihre Reise geht nach " + Stadt
        ←   + ".\n Der Wind steht gut.\n Sie brauchen 2 Wochen")
    for i in range (5):
       aktKosten[i] =  random.randint(KostenMin[i],KostenMax[i])
    DisplayAktualisieren()
```

Kapitel 4 - Die Canvas benutzen und Tastensteuerung

Ü B E R S I C H T

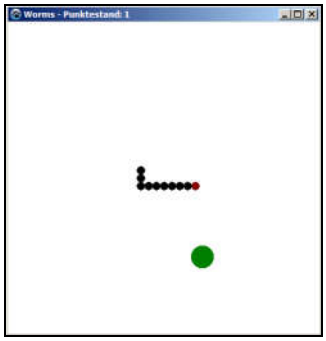
Canvas und Objekte erstellen	`can = tkinter.Canvas(Fenster, height = 500, width = 500, bg="white")`
	`can.grid(row = 1, column = 1)`
	
	`ov = can.create_oval(100,50,110,60, fill = "red")`

Bilder und Objekte in der Canvas bewegen	Wenn Images (oder andere Komponenten) in einem Fenster erstellt werden, so werden sie intern durchnummeriert. Mit der Nummer kann das entsprechende Objekt relativ zu seiner aktuellen Position bewegt werden.
	
	`canvas.create_image(200,YPosition,image=Bild, anchor=tkinter.NW)`
	
	`canvas.move(1,10,-20)`
	
	Hier wird das erste erstellte Objekt –unser Bild- um 10 in x-Richtung und um -20 in y-Richtung bewegt.
	
	Das Arbeiten mit diesen Nummern ist allerdings sehr unsicher, da die Nummern in Erstellreihenfolge der Objekte angelegt werden. Vertauscht sich da was, ist alles hin.
	
	Um diese Unsicherheit mit den Nummern zu umgehen, sollte eine Referenz auf das Bild angelegt werden:
	
	`Bild = tkinter.PhotoImage(file="FlappyBirdGif.gif")`
	`b = canvas.create_image(200,200,image=Bild, anchor=tkinter.NW)`
	
	`canvas.move (b, 10, -20)`
	`-------------------------------------`
	`Kreis1 = canvas.create_oval(90,50,100,60, fill = "blue")`
	
	`canvas.move(Kreis1, 10,-20)`

Tasten an widgets binden	`Fenster.bind("<KeyPress-Left>", links)`
	`Fenster.bind("<KeyPress-Right>", rechts)`
	`Fenster.bind("<KeyPress-Down>", runter)`
	`Fenster.bind("<KeyPress-Up>", hoch)`
	
	`canvas.bind(sequence="<Button-1>", func = ButtonDown)`
	`#sequence und func ist optional`
	
	`Fenster.bind("<Key>", taste)`
	`#<key> beschreibt hier irgendeine beliebige Taste`

M O T I V A T I O N

Das Spiel "Snake" gibt es in allen Varianten seit ca. 1980 und wurde wegen der simplen Grafik in seiner Umsetzung auf Handys populär.



Schritt 1: Schlange zeichnen
Die 10 Glieder der Schlange werden über Arrays verwaltet. Beim Start werden die Arrayeinträge mit Startwerten belegt.

Nun kann die Schlange in einer Prozedur "move" schonmal gezeichnet werden. "move" wird vom einem Timer aufgerufen.

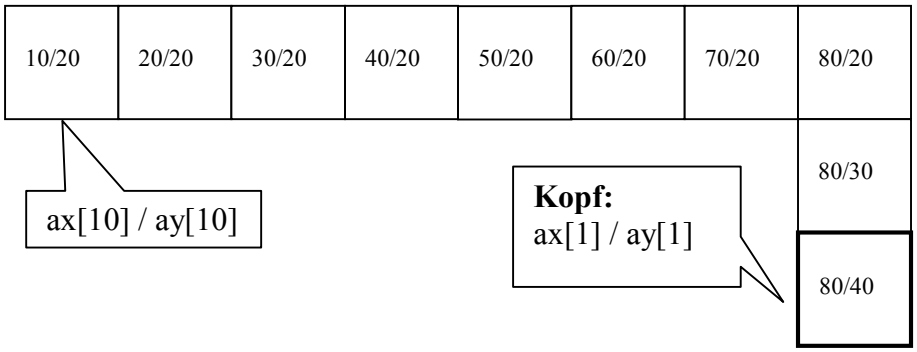
```
timerOn = True


def Timer():
    [. . .]
    Fenster.after(500, Timer)

#GUI
Fenster = tkinter.Tk()
   if (timerOn == True):
       Fenster.after(50, Timer)
```

Dabei wird hier schon das Zeichnen des Rumpfes in Schwarz vom Zeichnen des Kopfes (in einer anderen Farbe) getrennt.
Erkennbar sind dem Kopf die Arrayfelder WormX[1] und WormY[1] zugeordnet (bzw. WormX[0]), dem Rumpf die übrigen Felder.
Die einzelnen Glieder werden als Punkte dargestellt mit einem Durchmesser von 10 Pixel. Damit bewegt sich die Schlange in 10-Pixel-Schritten.
Das Formular wird weiß gemacht.

```
Snake = [0 for i in range(11)]    #Zu zeichnende Schlangenglieder
Snakex = [10 for i in range(11)]  #Koordianten der Schlange
Snakey = [0 for i in range(11)]   #Koordinaten der Schlange

#GUI
Fenster = tkinter.Tk()
Fenster.title("Snake")

Fenster.after(500,timer) #Der Timer

Z = tkinter.Canvas(Fenster, height = 500, width = 500, bg="white")
Z.grid(row = 1, column = 1)

#Snake - Kopf
Snake[0] = Z.create_oval(100,50,110,60, fill = "red")
#Snake - Schwanz
Snake[1] = Z.create_oval(90,50,100,60, fill = "blue")
Snake[2] = Z.create_oval(80,50,90,60, fill = "blue")
Snake[3] = Z.create_oval(70,50,80,60, fill = "blue")
```

```
Snake[4] = Z.create_oval(60,50,70,60, fill = "blue")
Snake[5] = Z.create_oval(50,50,60,60, fill = "blue")
Snake[6] = Z.create_oval(40,50,50,60, fill = "blue")
Snake[7] = Z.create_oval(30,50,40,60, fill = "blue")
Snake[8] = Z.create_oval(20,50,30,60, fill = "blue")
Snake[9] = Z.create_oval(10,50,20,60, fill = "blue")
#GUI
Fenster.mainloop()
```

Schritt 2: Schlange bewegen
Idee der Schlangenbewegung über ein Array:



a) Bewegung des Rumpfes
Die neuen Array-Koordinaten des ***Rumpfes*** können über eine Schleife aus den "Vorgängern" berechnet werden.

```
def timer():
  Bewegung()
  if (timerOn == True):
    Fenster.after(200, timer)

def Bewegung():
  for i in range(10,0,-1): #Rückwärtszählschleife
    Z.move(Snake[i],Snakex[i],Snakey[i])
    Snakex[i] = Snakex[i-1]
    Snakey[i] = Snakey[i-1]
```

b) Richtungssteuerung → Bewegung des Kopfes
Die Schlange soll über die ***Richtungstasten (Pfeiltasten)*** bewegt werden. Es gibt vier Bewegungszustände **z**:

1 (oben)
4 (links) 2 (rechts)
3 (unten)

```
Richtung = 2

def rechts (event):
  global Richtung
  Richtung = 2


def links (event):
    global Richtung
    Richtung = 4

def hoch(event):
    global Richtung
    Richtung = 1

def runter(event):
```

```
    global Richtung
    Richtung = 3

def Bewegung():
  global Bewegungx
  global Bewegungy
  if (Richtung == 1):
      Bewegungx = 0
      Bewegungy = -10
  if (Richtung == 2):
      Bewegungx = 10
      Bewegungy = 0
  if (Richtung == 3):
      Bewegungx = 0
      Bewegungy = 10
  if (Richtung == 4):
      Bewegungx = -10
      Bewegungy = 0
[. . .]

Fenster.bind("<KeyPress-Left>", links)
Fenster.bind("<KeyPress-Right>", rechts)
Fenster.bind("<KeyPress-Down>", runter)
Fenster.bind("<KeyPress-Up>", hoch)
```

Schritt 3: „Randcrash“ und “Schlangenselbstcrash” programmieren
Randcrash und Schlangencrash sind recht einfach zu implementieren. Es werden lediglich die Koordinaten des Schlangenkopfes mit den Dimensionen des Fensters, bzw. den Koordinaten des Schlangenrumpfes verglichen.

```
def testeKollisionRand():
  global timerOn
  Tupel = Z.bbox(Snake[0])
  if (Tupel[0]<0) or (Tupel[1]<0) or (Tupel[2] > Z.winfo_width()) or (Tupel[3] >
Z.winfo_height()):
      timerOn = False

def testeKollisionSchwanz():
   global timerOn
   for i in range (1, 10):
      TupelKopf = Z.bbox(Snake[0])
      TupelGlied = Z.bbox(Snake[i])
      if (TupelKopf[0] ==
          TupelGlied[0]) and (TupelKopf[1] == TupelGlied[1]):
         timerOn = False
```

Schritt 4: Apfel fangen
Der grüne Apfel kann einfach als grüner Punkt zufällig irgendwohin gezeichnet werden. Zuvor wird der alte Apfel weiß übermalt. Dann wird geprüft, ob die Koordinaten des Schlangenkopfes in einem Toleranzbereich um die Apfelmitte (Durchmesser des Apfels) liegen.

```
def testeKollisionApfel():
  global Apfelx
  global Apfely
  Liste = Z.find_overlapping(Apfelx,Apfely, Apfelx+20, Apfely+20)
  if len(Liste) > 1:
    Apfelx = random.randint(20,480)
    Apfely = random.randint(20,480)
    Z.coords(Apfel, Apfelx, Apfely, Apfelx+20, Apfely+20)
```

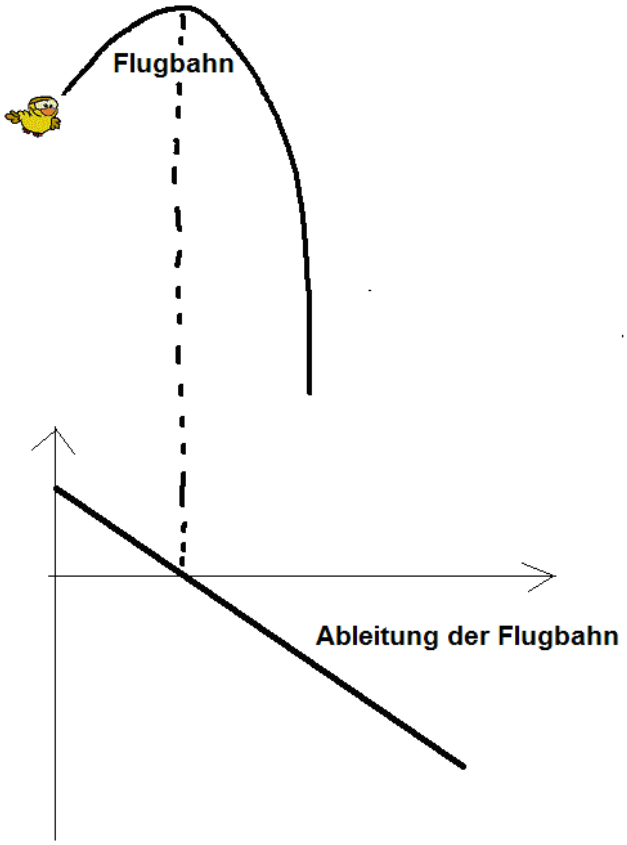
Ü B U N G E N

Aufgabe 1
Flappy Bird ist ein Spiel, bei dem ein Vogel zwischen von links nach rechts kommenden Säulen durchfliegen muss. Das Spiel wurde 2013 fürs Smartphone entwickelt.

1. Schritt: Der Vogel

Der Vogel fliegt auf Mausklick parabelförmig bis zum Scheitelpunkt nach oben und sinkt dann immer schneller ab. Dabei bleibt die x-Koordinate des Vogels unverändert, da die Bewegung durch den sich bewegenden Hintergrund simuliert wird.

Die "Flugbahn" des Vogels wird über einen Timer gesteuert. Würde sich der Vogel nicht nur nach oben oder unten bewegen, so würde seine Flugbahn eine nach unten geöffnete Parabel beschreiben. Ein Flügelschlag setzt den Vogel dabei auf einen Punkt vor dem Scheitelpunkt.



Von dieser Flugbahn ist allerdings nur die Vertikalbewegung parallel zur y-Achse zu sehen.
Damit der Vogel sich tatsächlich auf der Parabel bewegt, muss sich die Position des Vogels um die Momentangeschwindigkeit ändern. Im obigen Beispiel steigt der Vogel zunächst recht schnell und steigt umso langsamer, je näher er dem Scheitel kommt. Ist der Scheitel überrundet, so sinkt der Vogel erst langsam und wird dann immer schneller.
Die Momentangeschwindigkeit entspricht bei der Parabel der Ableitung, hier also einer Geraden.

Zur Anzeige des Vogels wird ein Image verwendet, das entsprechend des Flugs bewegt wird. Die Flugbahn über die Parabel verlangt zwei Koordinaten (x und y → XPosition, YPosition). Lazarus und Python stellen eine Grafikkomponente zur Verfügung, bei Java sollte eine entsprechende Komponente als Klasser erstellt und benutzt werden.

```
XPosition = -5
YPosition = 400

#GUI
Fenster = tkinter.Tk()
Fenster.title("Flappy Bird")
Bird = tkinter.PhotoImage(file="FlappyBirdGif.gif")

canvas=tkinter.Canvas(Fenster, width=600, height=600, bg='white')
canvas.pack()

b = canvas.create_image(200,YPosition,image=Bird, anchor=tkinter.NW)
```

Wenn Images (oder andere Komponenten) in einem Fenster erstellt werden, so werden sie intern durchnummeriert. Mit der Nummer kann das entsprechende Objekt bewegt werden. Beispiel:

```
 # canvas.create_image(200,YPosition,image=Bird, anchor=tkinter.NW)
# canvas.move(1,10,-20)
```

würde das erste erstellt Objekt –hier unseren Vogel- um 10 in x-Richtung und um -20 in y-Richtung bewegen.

Das Arbeiten mit diesen Nummern ist allerdings sehr unsicher, da die Nummern in Erstellreihenfolge der Objekte angelegt werden. Vertauscht sich da was, ist alles hin.

Um diese Unsicherheit mit den Nummern zu umgehen, kann eine Referenz auf den Vogel angelegt werden:

```
# b = canvas.create_image(200,YPosition,image=Bird, anchor=tkinter.NW)
# canvas.move (b, 10, -20)
```

bewegt nun immer unseren Vogel, egal, wann er erstellt wurde.
Die Gerade der Parabelableitung könnte so implementiert werden:

```
def Gerade(x):
     return -3*x + 4
```

Im Timer wird die x-Position des Vogel erhöht und der entsprechende y-Wert berechnet. Der x-Wert wird allerdings nicht dargestellt, nur die y-Position.

```
timerOn = True

[. . . ]

def Timer():
      global XPosition
      global YPosition
      global timerOn
      XPosition = XPosition +1
      YPosition = Gerade (XPosition)

 canvas.move(b, 0,  -YPosition)

 # b ist die Referenz auf den Vogel
 # über move wird das referenzierte Objekt in x-Richtung
 # und y-Richtung bewegt

      if (timerOn == True):
           Fenster.after(50, Timer)
```

Beim Klick wird die X-Position auf -5 gesetzt, der Vogel startet bzgl. der y-Koordinate wieder vor dem Scheitelpunkt, er fliegt also etwas nach oben, bevor es wieder runter geht:

```
def ButtonDown(event):
      global XPosition
      XPosition = -5

#GUI
canvas.bind(sequence="<Button-1>", func = ButtonDown)
```

Alternativ kann statt des Mausklicks auch eine Taste gedrückt werden.

```
def key(event):
      global XPosition
      XPosition = -5
```

```
# GUI
Fenster.bind("<Key>", key)
```

2. Schritt – Kollision des Vogels mit dem Bildschirmrand oben und unten

In einer Funktion wird mit jeder Timerausführung getestet, ob die Y-Koordinate des Vogel größer oder kleiner der Fensterdimensionen ist. Verlässt der Vogel den Bildschirm, stoppt der Timer.

```
def testeKollisionObenUnten():
      global timerOn
      a = canvas.bbox(1)
      u = canvas.winfo_height()
      if (a[1]<0):
          timerOn = False
      if ((a[1]+Bird.height()) > canvas.winfo_height()):
          timerOn = False

# Bei Python ist die Position eines Objekts auf dem
# Formular (Canvas) bbox (BoundingBox) aubfragbar. bbox liefert
# ein 4-Tupel zurück (X1,Y1, X2, Y2), das die linke obere
# und die rechte untere Ecke eines Rechtecks um das
# entsprechende Objekt markiert.
```

3. Schritt – Die Säulen und die Säulenbewegung

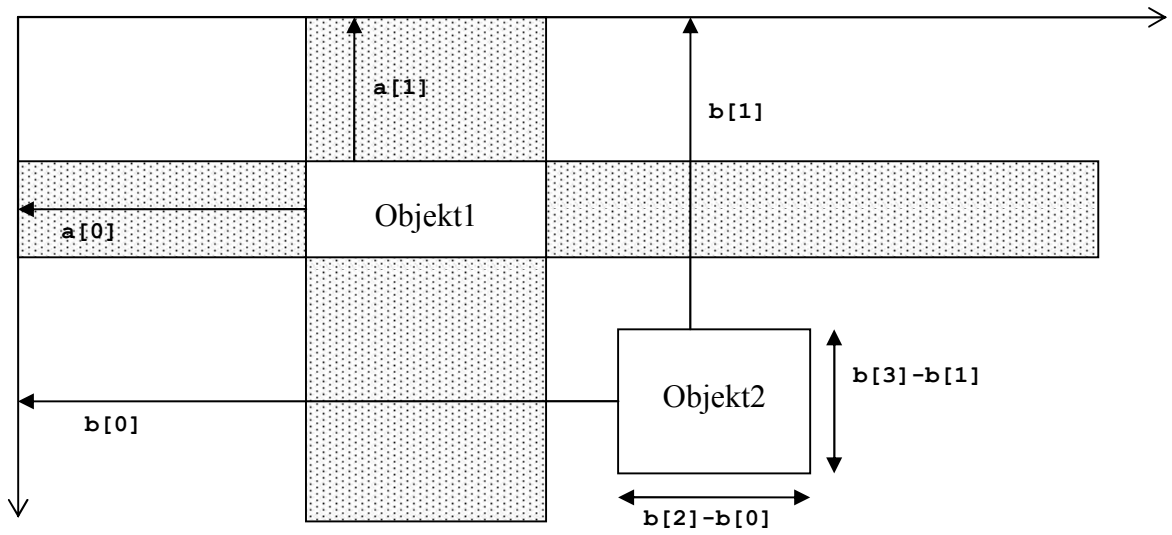
Insgesamt reichen 4 Säulen aus (insgesamt also 2 Säulenpaare), um die Bewegung zu simulieren. Für die spätere Kollisionsabfrage und die Bewegung lohnt die Verwaltung der Säulenimages in einem Array.
Die Bewegung erfolgt im Timer, die Säulen (Hindernisse) werden, sobald sie den linken Bildrand erreichen, in einer Prozedur wieder nach rechts gesetzt und zufällig in der Höhe versetzt.

```
def Timer():
      [. . .]
      # Säulen warden nach links bewegt
      for i in range(4):
            canvas.move(s[i], -10,0)


      # Säulen werden beim Erreichen des linken Rands
      # nach rechts gesetzt
      for i in range(4):
            a = canvas.bbox(s[i])
            if (a[0] < 0):
                  canvas.move(s[i],600,0)

      if (timerOn == True):
           Fenster.after(50, Timer)


#GUI
Saeule = tkinter.PhotoImage(file="Saeule.gif")
[. . .]
s[0] = canvas.create_image(300,500, image=Saeule, anchor=tkinter.NW)
s[1] = canvas.create_image(300,0,image=Saeule, anchor=tkinter.NW)
s[2] = canvas.create_image(580,450,image=Saeule, anchor=tkinter.NW)
s[3] = canvas.create_image(580,0,image=Saeule, anchor=tkinter.NW)
```

4. Schritt: Kollisionsabfrage des Vogels mit den Säulen

Kollisionsabfragen zwischen zwei Objekten ist im Bereich der Spiele eine oft auftauchende Standardoperation. Zu beachten ist, dass der Ursprung des Bildschirmkoordinatensystems oben links liegt und nur in die positiven Bereche ragt.

Objekt2 kollidiert mit Objekt2, wenn es gleichzeitig in den horizontalen Schraffurbereich und in den vertikalen Schraffurbereich des Objekts1 hineinragt.



Die Funktion *bbox* liefert ein 4-Tupel, das die linke obere und die rechte untere Ecke des Rahmens eines Objekts (x1, y1, x2, y2) beinhaltet

```
a = canvas.bbox(Objekt1)
b = canvas.bbox(Objekt2)
```

Damit sieht die Kollisionsabfrage so aus:

```
def Timer():
      [. . .]
      if testeKollisionMitHindernis():
         timerOn = False


def testeKollisionMitHindernis():
    global timerOn
    Rueckgabe = False
    for i in range (3):
          Hindernis = canvas.bbox(s[i])
          Vogel = canvas.bbox(b)
          if ((Vogel[1] < Hindernis[3]) and (Vogel[3] >  Hindernis[1])
                 and (Vogel[2] > Hindernis[0]) and
                 (Vogel[0] <  Hindernis[2])):
                Rueckgabe = True
    return Rueckgabe
```