

# Lösungen zu den Aufgaben

## Lösungen zu Kapitel 1

### Lösung 1

```
>>> from math import *
>>> (1 + e)**97
2.1047644793816324e+55

>>> sqrt(23/14)
1.2817398889233114
>>> (2**8 * 91**12)/pi
2.6277666738095865e+25
```

### Lösung 2

```
>>> (30*10 + 6*15)*7.5
2925.0
```

### Lösung 3

```
>>> pi*(4/2)**2 * 5
62.83185307179586
```

### Lösung 4

```
>>> float(9)
9.0
>>> int(7/2)
3
```

Kommentar:  $7/2$  ergibt 3.5, die `int()`-Funktion lässt die Nachkommastellen weg.

```
>>> 0b100 + 0b100
8
```

Kommentar: Die Summe zweier Binärzahlen wird als Dezimalzahl ausgegeben.

```
>>> bin(2**4)
'0b10000'
bin(2**5 + 2**0)
'0b100001'
```

Kommentar: An den Beispielen erkennt man den Aufbau von Binärzahlen als Summe von Zweierpotenzen.

```
>>> 10*10/10
10.0
>>> 2/2
1.0
>>> 3**2
9
>>> 9**0.5
3.0
```

Kommentar: Division / und Potenzieren \*\* mit einer Gleitkommazahl liefern immer ein Objekt vom Typ float, auch wenn – rein mathematisch – eine ganze Zahl herauskommt.

## Lösung 5

```
abs(12*-2 + 2)
betrag = 34.5
z = y * x
```

## Lösung 6

Nicht erlaubt sind False (Schlüsselwort) und x(1) (Klammern sind nicht erlaubt). Dagegen ist das Wort false als Name erlaubt, da es mit einem kleinen f beginnt und sich somit von dem Schlüsselwort False unterscheidet.

# Lösungen zu Kapitel 2

## Lösung 1

```
# anpeilen.py
from math import tan, radians
```

```
# Eingabe
höhe = input("Höhe des Winkelmessers gegenüber dem Boden (m): ")
entfernung = input("Entfernung vom Gebäude (m): ")
winkel = input("Winkel (Grad): ")

# Verarbeitung
h = float(höhe) #1
e = float(entfernung)
alpha = radians(float(winkel)) #2
gebäudehöhe = h + e*tan(alpha)

# Ausgabe
print("Das Gebäude ist",
      round(gebäudehöhe, 2),
      "m hoch.") #3
```

### Kommentare

**#1:** Die Funktion `input()` liefert eine Zeichenkette, die in eine Gleitkommazahl umgewandelt werden muss.

**#2:** Der Winkel (in Grad) muss in den Radianen (Kreisbogen) umgewandelt werden.

**#3:** Die Ausgabe wird aus konstanten und variablen Teilen zusammengesetzt.

### Lösung 2

Ausdruck	Ergebnis
<code>not 2 &gt; 1</code>	False
<code>not False and True</code>	True
<code>not not False</code>	False
<code>not "Fisch" and not "Fleisch"</code>	False
<code>1 or 0</code>	1
<code>(2 &gt; 2)</code>	False

### Lösung 3

#### Programm

Die Erweiterungen des Programms sind fett gedruckt.

```
# zylinder_mehrmals.py
from math import pi
```

```
antwort = "j" #1
while antwort == "j":
    eingabe_durchmesser = input("Durchmesser in m: ")
    eingabe_höhe = input("Höhe in m: ")
    d = float(eingabe_durchmesser)
    h = float(eingabe_höhe)
    volumen = pi * (d/2)**2 * h
    print("Der Zylinder hat ein Volumen von ",
          round(volumen, 2), " Kubikmetern.")
    antwort = input("Noch eine Rechnung (j/n)? ")
print("Auf Wiedersehen.")
```

### Kommentare

**#1:** Die Variable `antwort` muss auf den Anfangswert `"j"` gesetzt werden, damit die Schleife wenigstens einmal durchlaufen wird.

## Lösung 4

### Programm

```
from gpiozero import LED, Button
from time import sleep
schalter = Button("BOARD8") #1
led = LED("BOARD10") #2
while True:
    if schalter.is_pressed: #3
        led.on() #4
        sleep(0.5)
        led.off() #5
        sleep(0.5)
```

### Kommentare

**#1:** Das Objekt `schalter` repräsentiert einen Schalter an Pin 8 des GPIO.

**#2:** Das Objekt `led` repräsentiert eine LED an Pin 10 des GPIO.

**#3:** Wenn der Schalter geschlossen ist, wird der folgende (eingerückte) Block ausgeführt.

**#4:** Die LED wird eingeschaltet.

**#5:** Die LED wird ausgeschaltet.

# Lösungen zu Kapitel 3

## Lösung 1

Sequenz	Länge	Erklärung
"A\nB"	3	Die Escape-Sequenz \n ist <i>ein</i> Zeichen.
b"1-2"	3	Das Präfix b ist kein Teil des Bytestrings.
[]	0	Die leere Liste enthält kein Item.
[[]]	1	Die Liste enthält eine leere Liste als Item.
[1, [2, 3]]	2	Zwei Items: Die Zahl 1 und die Liste [2, 3]
"Power" + "Pop"	8	Die Konkatination enthält 5+3 Items.
"Boden"[:2]	2	Der Slice "Bo" enthält 2 Zeichen.
{1, 1, 1, 1}	1	Die Menge hat nur ein (einmaliges) Element.
{1,2} {2, 3}	3	Die Vereinigung ist {1, 2, 3}.

## Lösung 2

### Programm

```
# lottozahlen.py
from random import choice
zahlen = list(range(1,50))          #1
gezogen = []                        #2

for i in range(6):                  #3
    zahl = choice(zahlen)            #4
    zahlen.remove(zahl)
    gezogen.append(zahl)

gezogen.sort()                      #5
print("Ziehung der Lottozahlen")
for z in gezogen:
    print(z, end=" ")
```

### Kommentare

**#1:** Hier wird zunächst ein `range`-Objekt mit Zahlen zwischen 1 und 49 erzeugt und daraus eine Liste gebildet. Warum eine Liste? Wir brauchen eine *veränderbare* Kollektion, weil später die bereits gezogenen Zahlen daraus entfernt werden sollen.

**#2:** Eine zunächst leere Liste für die gezogenen Lottozahlen wird instanziiert.

**#3:** Der nächste Anweisungsblock wird sechs Mal ausgeführt.

**#4:** Ein zufälliges Element der Liste `zahlen` wird ausgewählt, dann aus der Liste entfernt und der Liste `gezogen` zugefügt. Beachten Sie, dass der Name `zahl` keinen *Index*, sondern ein *Item* der Liste `zahlen` (also eine Zahl zwischen 1 und 49) bezeichnet.

**#5:** Die gezogenen Lottozahlen werden sortiert.

### Lösung 3

Die Bilder visualisieren in der Reihenfolge von oben nach unten die folgenden Aufrufe der Funktion `range()`:

```
range(0, 10, 2)
range(1, 5)
range(6)
range(2, 8, 2)
```

## Lösungen zu Kapitel 4

### Lösung 1

#### Programm

```
# morsen.py
from gpiozero import LED
from time import sleep
DIT = 0.2

def blink (led, länge):
    # Das Argument ...
    led.on()                #1
    sleep(länge)            #2
    led.off()
    sleep(DIT)

def morsen(led, text):
    # Das Argument ...
    for zeichen in text:    #3
        if zeichen == ".":
            blink(led, DIT)
        elif zeichen == "-":
```

```

        blink(led, 3*DIT)
    else:
        sleep(3*DIT)

# Hauptprogramm
...
```

### Kommentare

**#1:** Die LED wird eingeschaltet. Hier wird die Methode `on()` des Objekts `led` aufgerufen, das als erstes Argument übergeben worden ist.

**#2:** Warte für `länge` Sekunden. Der Wert von `länge` wurde als zweites Argument der Funktion übergeben.

**#3:** Der String `text` wird Zeichen für Zeichen durchlaufen. Dieser String wurde als zweites Argument der Funktion übergeben.

## Lösung 2

### Programm

```

from gpiozero import Button
from signal import pause

def zurücksetzen():
    global zähler          #1
    zähler = 0
    print(zähler)

def zählen():
    global zähler
    zähler += 1            #2
    print(zähler)

zähler = 0
print(zähler)
knopf = Button("BOARD12", hold_time=1)
knopf.when_held=zurücksetzen
knopf.when_pressed=zählen
pause()                   #3
```

## Kommentare

**#1:** Die Variable `zähler` wird global gemacht. So können beide Funktionen ihren Wert verändern.

**#2:** Der Wert der globalen Variablen `zähler` wird um 1 erhöht.

**#3:** Das Programm pausiert. Anderenfalls wäre es hier beendet und der Zähler würde nicht funktionieren.

# Lösungen zu Kapitel 5

## Lösung 1

### Programm

```
# schicksal.pyw
...

def start():
    global label
    label.config(text="Drücke den Knopf!", fg="red",
                  font=("arial", 25, 'bold'))

def starteVorhersage():                                #1
    global label, startzeit                             #2
    label.config(text="Lasse den Knopf wieder los!")
    startzeit = time()                                  #3

def treffeVorhersage():                                #4
    global label
    zeit = time() - startzeit                           #5
    n = len(VORHERSAGEN)
    i = round(zeit * 10000) % n                          #6
    label.config(text=VORHERSAGEN[i], fg="black",
                  font=("times", 16))                    #7
    sleep(3)                                              #8
    start()

def beendeProgramm():
    fenster.destroy()

fenster = Tk()
...
```



## Kommentare

**#1:** Diese Funktion wird aufgerufen, wenn der Knopf gedrückt worden ist.

**#2:** Diese beiden Variablen müssen global sein, weil in dieser Funktion auf sie schreibend zugegriffen wird.

**#3:** Die Anzahl der Sekunden seit dem 1.1.1970 wird in der globalen Variablen `startzeit` gespeichert.

**#4:** Diese Funktion wird aufgerufen, wenn der Knopf wieder losgelassen worden ist.

**#5:** Von der aktuellen Zeit (Anzahl der Sekunden seit dem 1.1.1970) wird die Startzeit abgezogen. Das Ergebnis ist die Anzahl der Sekunden, die der Knopf gedrückt gehalten wurde. Die ist eine Gleitpunktzahl mit mindestens sechs Stellen nach dem Punkt.

**#6:** Die Drückzeit `zeit` ist eine kleine Gleitpunktzahl (z.B. 1.234567). Diese Zahl wird mit 1000 multipliziert und gerundet. So erhält man eine ziemlich große ganze Zahl (z.B. 1234). Diese Zahl wird modulo `n` gerechnet, wobei `n` die Anzahl der Strings in der Liste `VORHERSAGEN` ist. Das Ergebnis ist eine Zahl zwischen 0 und `n - 1`. Sie wird der Variablen `i` zugewiesen.

**#7:** Aus der Liste `VORHERSAGEN` wird das Element mit Index `i` ausgewählt. Das ist ein String mit einer Vorhersage. Dieser String wird auf dem Label angezeigt.

**#8:** Die Programmausführung wird für 3 Sekunden unterbrochen. Danach geht es weiter.

## Lösung 2

### Programm

```
# reaktionstest.pyw
from gpiozero import Button
from tkinter import Tk, Label
from random import choice, randint
from time import sleep

def start():
    global p1, p2, links, rechts
    p1 = 0 #1
    p2 = 0
    links.config(text=str(p1)) #2
    rechts.config(text=str(p2))
    spielen()
```

```
def spielen():
    global p1, p2, links, rechts, oben
    for i in range(3):
        ...
        break
    if knopf2.is_pressed: #3
        p2 +=1
        rechts.config(text=str(p2))
        break

fenster = Tk()
fenster.geometry("1920x1080")
#fenster.attributes("-fullscreen", True) #4
fenster.rowconfigure(0, weight=3) #5
fenster.rowconfigure(1, weight=1)
fenster.columnconfigure(0, weight=1)
fenster.columnconfigure(1, weight=1)
oben = Label(master=fenster, bg="lightblue") #6
links = Label(master=fenster, text="0",
               font=("Arial",200)) #7
rechts = Label(master=fenster, text="0",
               font=("Arial",200))
oben.grid(column=0, row=0, columnspan=2,
          sticky=("W", "E", "N", "S")) #8
links.grid(column=0, row=1, sticky=("W", "E", "N", "S"))
rechts.grid(column=1, row=1, sticky=("W", "E", "N", "S"))
startKnopf = Button("BOARD8")
...
```

### Kommentare

**#1:** Der Punktestand wird für beide Spieler auf 0 gesetzt.

**#2:** Der Punktestand des ersten Spielers wird im unteren Label links angezeigt. Mit `str()` wird aus der Zahl eine Zeichenkette gewonnen.

**#3:** Wenn der zweite Knopf gerade gedrückt ist, war der zweite Spieler schneller und erhält einen Punkt.

**#4:** Das Kommentarzeichen zu Beginn der Zeile wird erst entfernt, wenn das Programm fehlerfrei funktioniert. Erst dann wird die Anweisung ausgeführt und sorgt dafür, dass der Rahmen des Anwendungsfensters nicht gezeigt wird. Die Label füllen den gesamten Bildschirm aus (Vollbildmodus).

**#5:** Die erste Zeile des Rasters erhält das Gewicht 3. Das heißt, diese Zeile kann etwas größer werden als die untere Zeile.

**#6:** Hier wird ein Label mit hellblauem Hintergrund erzeugt.

**#7:** Hier wird ein Label erzeugt, das die Zahl 0 in großer Schriftgröße anzeigt.

**#8:** Das Label **oben** wird in das Raster eingefügt. Es ist in der oberen Zeile (`row=0`) und füllt die beiden Zellen des Rasters aus (`columnspan=2`). Mit der Option `sticky` wird dafür gesorgt, dass das Label die gesamten Rasterfelder ausfüllt (es »klebt« an allen vier Seiten).

## Lösungen zu Kapitel 6

### Lösung 1

Problem	Regulärer Ausdruck
a) Wörter, die ss oder ll enthalten.	<code>"\w*ll\w*"</code>
b) Telefonnummern, die mit +49 beginnen	<code>"\+49\s*\d+\s*\d+"</code>
c) Meterangaben, z.B. 2,35 m, 2.35m oder 2m	<code>"\d+([\.,]\d+)?\s*m"</code>
d) zwei oder mehr Leerzeichen hintereinander	<code>"( )+"</code>

### Lösung 2

Wir lösen die Aufgabe in der Python-Shell. Nehmen wir das Textbeispiel aus der Aufgabenstellung:

```
>>> text = "Auf dem Dachboden, ..."
>>> from re import *
```

Aus dem Text suchen wir eine Liste mit allen Textpassagen, die aus einem Satzzeichen, beliebig vielen Whitespaces und einem Buchstaben bestehen:

```
>>> fragments = findall("[!?:;,.]\s*\w", text)
>>> fragments
[' ', 'w', ' ', 'i', ' ', 'E', ' ', 'N']
```

Aus den letzten Buchstaben dieser Textfragmente (als Großbuchstaben) setzen wir das geheime Wort zusammen:

```
>>> word = ""
>>> for s in fragments:
    word += s[-1].upper()
```

```
>>> print (word)
WIEN
```

## Lösung 3

### Programm

```
# reim.pyw
from tkinter import *
from re import *

def search():
    postfix = entry.get()
    wordlist = findall("\w*" + postfix + "[.,!? ]", faust)
    wordlist2 = [w[:-1] for w in wordlist]      #1
    words = set(wordlist2)                     #2
    result = ", ".join(words)
    text.delete(1.0, END)
    text.insert(END, result)

# Widgets
window = Tk()
entry=Entry(master=window, font=("Arial", 14))
button = Button(master=window, font=("Arial", 14),
                 text = "Suche Reime!", command=search)
text=Text(master=window, font=("Arial", 14),
           width=40, height=7, wrap=WORD)
text.pack()
entry.pack(side=LEFT)
button.pack(side=LEFT)

faust = open("faust.txt", mode="r", encoding="utf-8").read()
window.mainloop()
```

### Kommentare

**#1:** Hier wird eine Liste von Wörtern aus dem Faust-Text konstruiert, die mit der Zeichenkette `postfix` enden. Man beachte, dass jedes Element der Liste `wordlist` ein Wort plus Leerzeichen oder Satzzeichen ist, z.B. "laufen, ". Dieses letzte Zeichen brauchten wir im ersten Schritt, um jeweils das Ende eines Worts zu erkennen. Nun wird eine Liste von Wörtern geschaffen, die die Wörter aus

`wordlist` jeweils ohne das letzte Zeichen enthält. Der Slice `[:-1]` bezeichnet die Items einer Sequenz ohne das letzte Item, das ja den Index `-1` hat.

**#2:** Aus der Liste, die wahrscheinlich noch viele Wiederholungen hat, wird eine Menge gebildet, in der jedes Element nur einmal vorkommt.

*Hinweis:* Die Funktionalität des Programms kann noch erheblich verbessert werden, indem auch Wörter gefunden werden, deren Endung zwar nicht genauso wie das gegebene Postfix geschrieben wird, die aber genauso klingt.

Auf `a1` reimen sich z.B. nicht nur Kanal und Schal, sondern auch Wahl und Pfahl. Um das zu berücksichtigen, kann man das Postfix überarbeiten, bevor man es in den regulären Ausdruck einbaut. Beispiel:

```
postfix = postfix.replace( "a", "ah?" )
```

## Lösungen zu Kapitel 7

### Lösung 1

#### Programm

```
# weckzeit.pyw
from tkinter import *
import _thread, time

def start(event):
    _thread.start_new_thread(changeNumber, ())           #1

def changeNumber():
    label.unbind("<1>")                                #2
    global control
    control = True
    while control:                                     #3
        hour.set(hour.get() + 1)                       #4
        if hour.get() > 23:
            hour.set(0)
            time.sleep(0.2)
        label.bind("<1>", start)                         #5

def stop(event):
    global control                                     #6
```

```

control = False #7

window = Tk()
hour = IntVar(master=window, value=0)
label = Label(master=window, height=1, width=2,
               font=("Arial", 40), textvariable=hour)
label.bind("<1>", start) #8
label.bind("<ButtonRelease-1>", stop) #9
label.pack()
window.mainloop()

```

### Kommentare

**#1:** Der Eventhandler `start()` wird aufgerufen, wenn die linke Maustaste gedrückt worden ist. Sie startet die Funktion `changeNumber()` in einem neuen Thread. Das ist notwendig, weil `changeNumber()` nur »von außen« beendet werden kann (nämlich indem ein anderer Prozess die globale Variable `control` auf `False` setzt). Die Funktion `changeNumber()` sorgt dafür, dass die Zahl auf dem Label fortlaufend erhöht wird – so lange, bis der Benutzer die Maustaste wieder loslässt.

**#2:** Hier wird verhindert, dass `start()` und damit `changeNumber()` durch schnelles Klicken noch ein zweites Mal aufgerufen werden.

**#3:** Die Variable `control` ist global. Sie wird von mehreren Funktionen gemeinsam genutzt. Sie dient zur Steuerung der `while`-Schleife. Diese Schleife wird so lange durchlaufen, bis die Maustaste wieder losgelassen wird. Dann wird nämlich (durch einen Eventhandler) die Variable `control` auf `False` gesetzt.

**#4:** Alle 0,2 Sekunden wird der Wert der Integer-Kontrollvariable um 1 erhöht. Wenn die Zahl größer als 23 ist, wird sie auf 0 gesetzt. So kann diese Variable die Stunde einer Uhrzeit darstellen.

**#5:** Hier wird die linke Maustaste wieder »freigegeben«.

**#6:** Das `global`-Statement ist notwendig, weil der Wert der globalen Variablen `control` geändert wird.

**#7:** Damit wird die Wiederholung in der Funktion `changeNumber()`, die parallel in einem eigenen Thread läuft, gestoppt.

**#8:** Der Event »Linke Maustaste wird gedrückt« wird an den Eventhandler `start()` gebunden.

**#9:** Der Event »Gedrückte linke Maustaste wird losgelassen« wird an den Eventhandler `stop()` gebunden.

## Lösung 2

### Programm

```
# schloss_einfach.pyw
import time, _thread
from tkinter import *

KEYS = ["1204", "8897", "1805"] #1

def close(): #2
    frame.destroy()
    label.config(text="")
    label.bind("<1>", createKeyboard)

def insert():
    global keyId
    label.after_cancel(keyId) #3
    num = label.cget("text") + key.get()
    label.config(text=num)
    _thread.start_new_thread(check, (num,)) #4

def check(num):
    global keyId
    if num in KEYS:
        label.config(text="Tür auf!") #5
        time.sleep(2)
        label.config(text="")
        keyId = label.after(2000, close) #6

def createKeyboard(event): #7
    global frame, keyId
    keyId = label.after(2000, close) #8
    label.unbind("<1>") #9
    frame = Frame(master=window)
    frame.pack()
    numButtons = [Radiobutton(master=frame, command=insert,
                              font=("Arial", 14), width=3,
                              variable=key, value=n,
                              text=n, indicatoron=False)
                  for n in "1234567890"]
    for i in range(3):
        numButtons[i].grid(row=0, column=i)
```

```

for i in range(3,6):
    numButtons[i].grid(row=1, column=i-3)
for i in range(6,9):
    numButtons[i].grid(row=2, column=i-6)
numButtons[9].grid(row=3, column=0)

window=Tk()
key = StringVar(master=window)
label = Label(master=window, bd=4, relief=SUNKEN,
              font=("Arial",14), width=14)          #10
label.pack(padx=30, pady=2)
label.bind("<1>", createKeyboard)

window.mainloop()

```

### Kommentare

**#1:** Liste mit Zahlenkombinationen, die gültige Schlüssel sind. In einem realistischen Programm würde die Liste aus einer Datei gelesen. Denn solche Schlüssel Listen werden immer wieder verändert.

**#2:** Diese Prozedur wird aufgerufen, wenn zwei Sekunden lang keine Eingabe erfolgte. Sie sorgt dafür, dass die Eingabe gelöscht wird und die Tastatur verschwindet.

**#3:** Die Funktion `insert()` wird aufgerufen, wenn eine Zahlentaste angeklickt worden ist. Deshalb soll die Tastatur zunächst nicht verschwinden. Der Prozess, der die Tastatur nach zwei Sekunden entfernen soll, wird gelöscht.

**#4:** Nun muss geprüft werden, ob die eingegebene Zahl ein Schlüssel ist. Weil diese Routine eine Wartezeit (`time.sleep()`) beinhaltet, wird sie in einem neuen Thread gestartet. So wird vermieden, dass der Hauptprozess in dieser Zeit blockiert ist.

**#5:** Wenn die eingegebene Zahl in der Liste der Schlüssel vorkommt, erscheint in der Anzeige für kurze Zeit der Schriftzug *Tür auf*. In einem realen digitalen Türschloss wird stattdessen der elektrische Türöffner betätigt.

**#6:** Nun wird ein Prozess gestartet, der nach 2 Sekunden die Funktion `close()` aufruft. Sie sorgt dafür, dass die Tastatur wieder entfernt wird und der Inhalt des Anzeigefeldes verschwindet. Die globale Variable `keyId` wird an anderer Stelle für das Löschen (Canceln) dieses Prozesses benötigt.

**#7:** Die Funktion ist ein Eventhandler. Sie wird aufgerufen, sobald der Benutzer das Anzeigefeld anklickt. Sie sorgt dafür, dass eine Tastatur für die Eingabe von Ziffern erscheint.



**#8:** Hier wird ein Prozess gestartet, der nach 2 Sekunden Inaktivität das Tastaturfeld wieder entfernt.

**#9:** Hier wird der Event »Linke Maustaste gedrückt« vom zugehörigen Eventhandler abgekoppelt. Damit wird verhindert, dass das Tastaturfeld ein zweites Mal erzeugt wird, wenn der Benutzer wieder auf das Label klickt.

**#10:** Das Label für die Darstellung der Ziffern wird mit einem **SUNKEN-Relief** (mit einem Rahmen der Breite 4 Pixel) versehen.

## Anschluss an ein elektromagnetisches Türschloss

Mit einem Relais-Modul und einem elektromagnetischen Türschloss können Sie das Projekt umsetzen. Sie verwenden die Schaltung aus Abb. 1.13. Pin 2 des GPIO (+ 5V) ist mit VCC des Relais verbunden, Pin 6 des GPIO mit GND und Pin 10 des GPIO mit IN1 des Relaismoduls. Anstelle einer Taschenlampe ist an Relais 1 der Türöffner angeschlossen. Der Stromkreis wird geschlossen (und damit der Türöffner eingeschaltet), wenn IN1 in den Zustand `False` versetzt wird.

```
# schloss_einfach.pyw
import time, _thread
from tkinter import *
from gpiozero import LED
...

def check(num):
    global keyId
    if num in KEYS:
        label.config(text="Tür auf!")
        relaisInput.off() # Stromkreis geschlossen
        time.sleep(3)
        relaisInput.on() # Stromkreis geöffnet
        label.config(text="")
        keyId = label.after(10000, close)
    ...
relaisInput = LED("BOARD10")
relaisInput.on() # Stromkreis geöffnet

window=Tk()

...
```

## Lösung 3

### Programm

```
# farbuhr.py
from time import localtime, sleep
from gpiozero import RGBLED, Button
FARBEN = [(0, 0, 0.8), (0, 0, 0.6), (0, 0.1, 0.5),
          (0, 0.2, 0.6), (0, 0.2, 0.8), (0, 0.3, 0.7),
          (0.1, 0.4, 0.7), (0.3, 0.3, 0.6), (1, 0.8, 0.4),
          (1, 1, 0.2), (1, 1, 0.4), (1, 1, 0.6),
          (1, 1, 0.8), (1, 1, 1), (0.8, 1, 0.8),
          (0.6, 1, 0.8), (0.4, 1, 0.8), (0.8, 0.8, 0.6),
          (0.8, 0.8, 0.4), (1, 0.8, 0.4), (1, 0.8, 0.2),
          (0.8, 1, 0.2), (0.6, 0.8, 0.4), (0.2, 0.8, 0.2)]    #1

licht = RGBLED(red="BOARD8", green="BOARD10", blue="BOARD12")
schalter = Button("BOARD16")
licht_an = False                                              #2
while True:                                                  #3
    if schalter.is_pressed:
        licht_an = not licht_an                            #4
    if licht_an:                                             #5
        t = localtime()                                     #6
        farbe = FARBEN[t.tm_hour]                          #7
        licht.color = farbe                                #8
    else:
        licht.off()                                         #9
    sleep(1)
```

### Kommentare

**#1:** Die Konstante `FARBEN` ist eine Liste mit 24 Tupeln. Jedes Tupel besteht aus drei Zahlen zwischen 0 und 1 und definiert eine Farbe (RGB-Werte).

**#2:** Die Variable `licht_an` erhält den Anfangswert `False`. Nur wenn sie den Wert `True` hat, sollen die LEDs leuchten.

**#3:** Hier beginnt eine Endloswiederholung.

**#4:** Wenn gerade der Schalter gedrückt worden ist, ändert die Variable `licht_an` ihren Wert. Wenn sie vorher `True` war, wird sie jetzt auf `False` gesetzt. Und wenn sie `False` war, wird sie auf `True` gesetzt.

**#5:** Wenn die Variable `licht_an` den Wert `True` hat, werden die Farben des LED-Strips neu gesetzt.

**#6:** Der Variablen `t` wird ein Zeitobjekt zugewiesen, das die aktuelle Uhrzeit repräsentiert.

**#7:** Das Zeitobjekt-Attribut `tm_hour` ist eine Zahl zwischen 0 und 23, das die Stunde der aktuellen Uhrzeit darstellt. Diese Zahl wird als Index verwendet, um ein Tupel aus der Liste `FARBEN` zu gewinnen.

**#8:** Der LED-Strip leuchtet in der Farbe, die durch das Tupel `farbe` bestimmt ist.

**#9:** Wenn `licht_an` den Wert `False` hat, wird der LED-Strip ausgeschaltet.

## Lösungen zu Kapitel 8

### Lösung 1

#### Programm

```
# verschieben.pyw
HEX = '0123456789abcdef' #1
WORDS = ['FLIEGE', 'MÜCKE', 'BIENE'] #2
from tkinter import *
from random import randint, choice

class Letter: #3
    def __init__(self, canvas, letter):
        self.c = canvas
        x = randint(30, int(self.c.cget('width'))-30) #4
        y = randint(30, int(self.c.cget('height'))-30)
        self.color = "#" + choice(HEX) + choice(HEX) + choice(HEX)
        self.iD = self.c.create_text(x, y, text=letter,
                                     font=('Arial', 100, 'bold'),
                                     fill=self.color) #5

    def move(self, x1, y1): #6
        x0, y0 = self.c.coords(self.iD)
        dx, dy = x1 - x0, y1 - y0
        self.c.move(self.iD, dx, dy) #7

class App(object):
    def __init__(self):
```

```

self.window=Tk()
self.c=Canvas(self.window, bg='white',
               width='12c', height='8c')

self.c.pack()
self.elements = [Letter(self.c, ch)
                  for ch in choice(WORDS)]           #8
self.window.bind('<Double-Button-1>', self.new)      #9
self.c.bind('<1>', self.drag)
self.c.bind('<Motion>', self.move)
self.window.bind('<ButtonRelease-1>', self.drop)
self.selected = None                                #10
self.window.mainloop()

def new(self, event):
    for element in self.elements:
        self.c.delete(element.iD)                   #11
    self.elements = [Letter(self.c, ch)
                     for ch in choice(WORDS)]        #12

def drag(self, event):                               #13
    for element in self.elements:
        if element.iD in self.c.find_closest(event.x, event.y):
            self.selected = element

def move(self, event):                               #14
    if self.selected:
        self.selected.move(event.x, event.y)

def drop(self, event):                               #15
    self.selected = None

App()

```

### Kommentare

**#1:** Hexadezimalziffern für die Konstruktion einer Zufallsfarbe.

**#2:** Liste von Wörtern aus einem Themenbereich für die Konstruktion der verschiebbaren Buchstaben.

**#3:** Die Klasse modelliert verschiebbare Buchstaben. Jedes **Letter**-Objekt muss den Canvas kennen, auf den es gezeichnet wird. Alle visuellen Operationen laufen über das Canvas-Objekt.

**#4:** Zufällige Anfangspositionen für die Buchstaben.

**#5:** Ein Buchstabe wird auf den Canvas geschrieben. Die aufgerufene Canvas-Methode gibt eine ID-Nummer zurück. Sie wird benötigt, wenn der Buchstabe verschoben werden soll.

**#6:** Die Methode verschiebt den Buchstaben auf dem Canvas an die Position  $x1$ ,  $y1$ . Zuerst wird (in  $x$ - und  $y$ -Richtung) die Differenz zwischen der Position des Buchstabens und der Position des Mauszeigers ausgerechnet ( $dx$ , und  $dy$ ) ...

**#7:** ... und schließlich wird der Buchstabe auf dem Canvas um diese Differenz verschoben.

**#8:** Hier entsteht eine Liste, die für jeden Buchstaben des zufällig ausgewählten Worts ein **Letter**-Objekt erzeugt.

**#9:** Hier werden Events an Eventhandler gebunden.

**#10:** Das Attribut bezeichnet das (durch Anklicken) zum Verschieben ausgewählte Objekt. Anfangs ist noch nichts angeklickt worden.

**#11:** Alle Buchstaben werden gelöscht.

**#12:** Es wird eine neue Liste von **Letter**-Objekten erzeugt. Wieder stellen sie Buchstaben eines zufällig ausgewählten Worts dar.

**#13:** Die Methode wird aufgerufen, wenn die linke Maustaste gedrückt worden ist. Dann wird das **Letter**-Objekt ausgewählt, das der aktuellen Mausposition am nächsten ist.

**#14:** Die Methode wird aufgerufen, wenn die Maus bewegt worden ist. Falls die Maustaste gedrückt ist und damit ein Buchstabe ausgewählt ist, wird dieser Buchstabe bewegt.

**#15:** Die Methode wird aufgerufen, wenn die Maustaste losgelassen wird. Dann ist der zuvor ausgewählte Buchstabe nicht mehr ausgewählt. Nichts ist mehr ausgewählt und bei Mausbewegungen wird nun nichts mehr verschoben ... bis zum nächsten Klick.

## Lösung 2

### Programm (1) – die Klasse Drum

Damit es übersichtlicher ist, wird der Programmtext in drei Teilen erklärt. Am Anfang des Codes stehen die Import-Anweisungen und einige nützliche Konstanten. Die Klasse **Drum** ist für die Ausgabe des Drum-Patterns zuständig.

```
# drum_machine.pyw
from tkinter import *
from gpiozero import LED
from time import sleep

import _thread

BEAT = 0.1          # Länge eines Schlages in Sekunden
PIN = "BOARD10"     # Pinnummer des GPIO
NOTES = ['halbe.gif', 'viertel.gif', 'achtel.gif', 'blank.gif']

class Drum:
    def __init__ (self, bpm):

self.led = LED("BOARD10")
        self.time = 60 / bpm                                #1
        self.running = False                                #2

    def play(self, pattern):
        self.running = True
        while self.running:
            for note in pattern:
                if note:
                    self.led.on()                            #3
                    sleep(BEAT)
                    self.led.off()                            #4
                    sleep(self.time * note - BEAT)

    def stop(self):                                          #5
        self.running = False
```

### Kommentare

**#1:** Das ist die Länge einer ganzen Note. Sie wird aus der Geschwindigkeit in bpm (*beats per minute* = Schläge pro Minute) ermittelt.

**#2:** Im Attribut `running` merkt sich das Objekt, ob gerade ein Drumloop abgespielt wird (`True`) oder nicht (`False`).

**#3:** Die LED wird eingeschaltet.

**#4:** Nach einem kurzen Moment wird sie wieder ausgeschaltet.

**#5:** Nun entsteht eine Pause. Sie ist so lang, dass das Aufblinken der LED plus Pause gerade der Länge des Beats entspricht.

## Programm (2) – die Klasse Note

Objekte der Klasse `Note` sind spezielle anklickbare Labels mit Notenbildern.

```
class Note(Label): #6
    def __init__(self, master):
        Label.__init__(self, master=master, bg='white')
        self.images = [PhotoImage(file=i) for i in NOTES]
        self.index = 0 #7
        self.values = [1.0/2, 1.0/4, 1.0/8, 0] #8
        self.config(image=self.images[0],
                     width='1cm', height='1.5cm')
        self.bind('<1>', self.next_) #9

    def next_(self, event):
        if self.index < 3: #10
            self.index += 1
        else: self.index = 0
        self.config(image=self.images[self.index]) #11

    def value(self):
        return self.values[self.index] #12
```

## Kommentare

**#6:** Die Klasse `Note` wird von `Label` abgeleitet. Alle `Label`-Methoden werden geerbt. Man kann sagen: `Note`-Objekte sind `Label`-Objekte mit zusätzlichen Eigenschaften.

**#7:** Das Attribut `index` ist eine Zahl zwischen 0 und 3. Es legt fest, welcher Notenwert und welches Bild diesem Objekt zugeordnet sind.

**#8:** Das ist eine Liste mit den vier möglichen Beatlängen: 1/2, 1/4, 1/8 und 0.

**#9:** Das Ereignis »linker Mausklick« wird an die Methode `next()` gebunden.

**#10:** Hier wird das Attribut `index` auf den Nachfolgewert gesetzt. Das Attribut `index` durchläuft zyklisch die Werte 0, 1, 2, 3, 0, 1, ...

**#11:** Das Label erhält ein neues Bild mit der nächsten Note.

**#12:** Hier wird die aktuelle Beatlänge zurückgegeben.

### Programm (3) – die Klasse DrumMachine

Die Klasse `DrumMachine` ist das Anwendungsfenster. Sie verwendet ein Objekt der Klasse `Drum` für die Ausgabe des Drumloops und eine Liste von Objekten der Klasse `Note` zur Darstellung der Notenwerte.

```
class DrumMachine:
    def __init__(self):
        self.window = Tk()
        self.drum = Drum(20) #13
        self.notes = [Note(self.window) for i in range(8)]
        for note in self.notes: #14
            note.pack(side=LEFT)
        self.buttonPlay = Button(master=self.window,
                                text='Play', command=self.play) #15
        self.buttonStop = Button(master=self.window,
                                text='Stop', command=self.drum.stop) #16
        self.buttonPlay.pack(side=LEFT, fill=Y)
        self.buttonStop.pack(side=LEFT, fill=Y)
        self.window.mainloop()

    def play(self):
        pattern = [note.value() for note in self.notes]
        if not self.drum.running: #17
            _thread.start_new_thread(self.drum.play, (pattern,))

DrumMachine()
```

#### Kommentare

**#13:** Es wird ein Objekt der Klasse `Drum` erzeugt. Es soll Drumloops mit der Geschwindigkeit 20 bpm (beats per minute) abspielen.

**#14:** In der vorigen Zeile wurde eine Liste mit acht Noten erzeugt. Jetzt werden sie in das Anwendungsfenster gepackt.

**#15:** Schaltfläche zum Abspielen des Drumloops. Sie wird mit der Methode `play()` der Klasse `DrumMachine` verbunden.

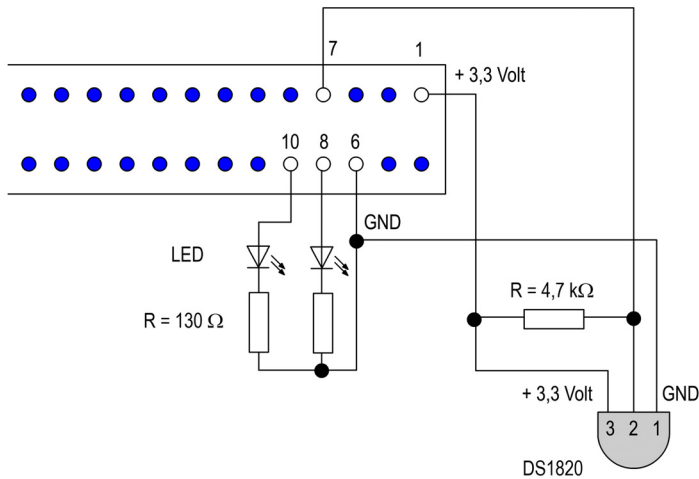
**#16:** Schaltfläche zum Anhalten des Drumloops. Sie wird mit der Methode `stop()` der Klasse `Drum` verbunden.

**#17:** Hier wird aus den acht aktuellen Notenwerten (Beatlängen) eine Liste erzeugt. Wenn das `Drum`-Objekt nicht gerade schon etwas abspielt, wird es nun aufgefordert, die Beat-Folge immer wieder abzuspielen. Der Aufruf der Methode `play()` muss in einem eigenen Thread erfolgen.



# Lösungen zu Kapitel 9

## Lösung 1



**Abb. 1:** Schaltplan für das Suchgerät

Das folgende Programm orientiert sich an diesem Schaltplan. Wir verwenden Pin 8 und Pin 10 für die Steuerung der LEDs.

### Programm

```
# cold_warmer.py
import time
from temperature import TempDevices
from gpiozero import LED                                #1

class Search:
    def __init__(self):
        self.red = LED("BOARD8")                        #2
        self.green = LED("BOARD10")
        self.t = TempDevices()[0]
        self.oldTemp = self.t.read()
        self.run()

    def run(self):
        while True:
            newTemp = self.t.read()                      #3
            if newTemp < self.oldTemp:                   #4
```

```

        self.green.on()
        self.red.off()
    elif newTemp > self.oldTemp:      #5
        self.green.off()
        self.red.on()
    else:                             #6
        self.green.off()
        self.red.off()
    self.oldTemp = newTemp           #7

```

### Kommentare

**#1:** Die Klasse LED modelliert eine LED, die man ein- und ausschalten kann.

**#2:** Die rote LED wird über Pin 8 gesteuert und die grüne über Pin 10 des GPIO.

**#3:** Die aktuelle Temperatur wird gelesen und mit der zuletzt gemessenen Temperatur verglichen.

**#4:** Wenn die Temperatur gestiegen ist, wird die rote LED eingeschaltet und die grüne ausgeschaltet.

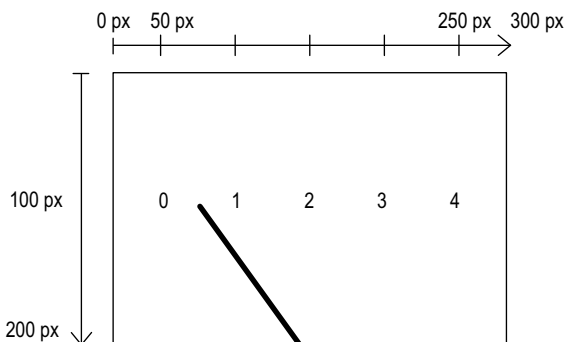
**#5:** Wenn die Temperatur gesunken ist, wird die grüne LED eingeschaltet und die rote ausgeschaltet.

**#6:** Sonst werden beide LEDs ausgeschaltet.

**#7:** Die soeben gemessene Temperatur wird gespeichert und kann im nächsten Schleifendurchlauf mit der neu gemessenen Temperatur verglichen werden.

### Lösung 2

Sie verwenden exakt die gleiche Schaltung wie in Abschnitt 9.6. Zur Planung der grafischen Oberfläche fertigt man am besten eine Skizze an (Abbildung 2).



**Abb. 2:** Planung der Oberfläche des Voltmeters

Während der Messungen wird der Zeiger (eine Linie) immer wieder gelöscht und neu gezeichnet. Die x-y-Koordinaten der einzelnen Bildelemente können Sie der Skizze entnehmen.

## Programm

```
#voltage.pyw
from read_mpc import readVoltage
from tkinter import *

class Pointer:                                     #1
    def __init__(self, canvas):
        self.c = canvas
        self.id = self.c.create_line(100, 100, 150, 200,
                                      width=4, fill="red")

    def read(self):
        self.c.delete(self.id)                   #2
        x = readVoltage() * 50 + 50               #3
        self.id = self.c.create_line(x, 100, 150, 200,
                                      width=4, fill="red")

class Device:
    def __init__(self):
        self.window = Tk()
        self.window.title("Spannungsmesser")
        self.canvas = Canvas(master=self.window,
                              width=300, height=200,
                              bg = "white")        #4
        self.canvas.pack()
        for i in range(5):
            self.canvas.create_text(50*(i+1), 100,
                                    text=str(i))    #5
        self.pointer = Pointer(self.canvas)
        self.run()
        self.window.mainloop()

    def run(self):                                 #6
        self.pointer.read()
        self.window.after(50, self.run)            #7

Device()                                          #8
```

## Kommentare

**#1:** Die Klasse `Pointer` modelliert den Zeiger. Ein `Pointer`-Objekt kann auf dem Analogkanal `Ch0` die aktuelle Spannung lesen und den Zeiger (eine Linie) auf dem Canvas in die passende Position bringen.

**#2:** Der Zeiger wird vom Canvas gelöscht.

**#3:** Diese Anweisung hat es in sich. Hier wird aus einem Messwert die x-Koordinate der Spitze des Zeigers (oberes Ende einer Linie) berechnet. Für eine Linie braucht man vier Koordinaten (`x0`, `y0`, `x1`, `y1`), die den Anfangs- und den Endpunkt der Linie bestimmen. Der untere Endpunkt der Linie (`x1`, `y1`), also der Punkt, um den sich der Zeiger dreht, ändert sich nicht. Auch die y-Koordinate des oberen Endpunkts `y0` bleibt konstant. Allein die Koordinate `x0` ändert sich. Ihr Wert hängt von der gemessenen Spannung ab.

Nun wurde in der Planungsskizze für 4 Volt ein Bereich von 200 px vorgesehen. Das heißt für jedes Volt 50 px. Deshalb muss die Spannung mit 50 multipliziert werden. Schließlich liegt der 0-Punkt der Skala bei 50 px. Deshalb müssen noch 50 Pixel addiert werden.

**#4:** Entsprechend der Planungsskizze brauchen wir einen Canvas, der 300 px breit und 200 px hoch ist.

**#5:** Hier wird die Beschriftung der Skala mit den Zahlen 0, 1, 2, 3, 4 (genau wie in der Planungsskizze) umgesetzt.

**#6:** Die Methode `run()` wird einmal bei der Initialisierung gestartet und ruft sich dann alle 50 ms selbst auf (**#7**). Sie sorgt somit dafür, dass alle 50 ms die anliegende Spannung gemessen und die Zeigerposition aktualisiert wird.

**#8:** Ein Objekt der Klasse `Device` wird instanziiert. Damit wird die Applikation gestartet.

## Lösung 3

Abbildung 3 illustriert die Arbeitsweise des Autosimulators. Ein langer Bildstreifen wandert von oben nach unten über den Canvas. Man sieht im Applikationsfenster immer nur einen Ausschnitt.

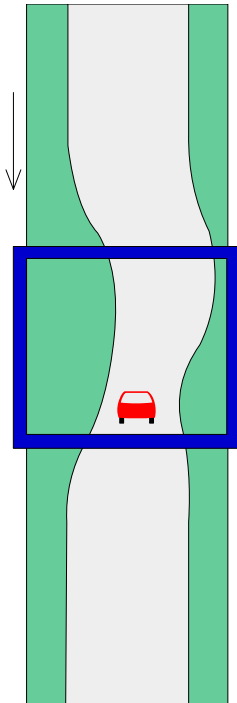
Fertigen Sie mit einem Grafikeditor zwei Bilder an, die Sie als GIF-Datei abspeichern:

- Ein Hintergrundbild mit einer Rennstrecke. Die Breite muss zum Canvas Ihres Programms passen. Hier haben wir nur ein kleines Bild einer Breite von 240 px. Aber man kann es natürlich auch viel größer machen. Es kann den ganzen Bildschirm ausfüllen. Der obere Teil des Bilds muss genauso aussehen

wie das untere Ende. Dann gibt es keine Sprünge, wenn das Bild wieder nach oben geschoben wird, nachdem beim Abspulen die untere Kante erreicht ist.

- Ein kleines Auto, das vom Programm auf die Rennstrecke gesetzt wird. Der Hintergrund des Autos muss transparent sein.

Speichern Sie die Bilddateien in das Verzeichnis mit dem Programm.



**Abb. 3:** Die Arbeitsweise eines einfachen Autosimulators

### Programm

```
#racing.pyw
import time
from read_mpc import readVoltage
from tkinter import *

WIDTH = 240                                #1
HEIGHT = 200
CAR = "car.gif"                             #2
ROAD = "road.gif"
```

```

class Car: #3
    def __init__(self, road):
        self.img = PhotoImage(file=CAR)
        self.road = road
        self.iD = self.road.create_image(150, 170,
                                           image=self.img) #4
        self.x = WIDTH/2

    def step(self):
        if self.x < WIDTH * readVoltage(): #5
            self.x += 1
            self.road.move(self.iD, 1, 0)
        elif self.x > WIDTH * readVoltage():
            self.x -= 1
            self.road.move(self.iD, -1, 0)

class Road(Canvas): #6
    def __init__(self, master):
        Canvas.__init__(self, master=master,
                        width=WIDTH, height=HEIGHT)

        self.pack()
        self.img = PhotoImage(file=ROAD)
        self.y = 0
        self.iD = self.create_image(0, HEIGHT,
                                    anchor=SW, image=self.img) #7

    def step(self):
        if self.y <= self.img.height() - HEIGHT: #8
            self.y += 3
            self.move(self.iD, 0, 3)
        else:
            self.y = 0
            self.coords(self.iD, 0, HEIGHT)

class Racing: #9
    def __init__(self):
        self.window = Tk()
        self.window.title("Autorennen")
        self.road = Road(master=self.window)
        self.car = Car(self.road)
        self.run()

```

```

        self.window.mainloop()

    def run(self):
        self.road.step()
        self.car.step()
        self.window.after(50, self.run)

Racing()

```

### Kommentare

**#1:** Höhe und Breite des Canvas werden in Konstanten gespeichert, weil im Programm häufiger darauf zugegriffen wird.

**#2:** Das sind die Namen der Bilddateien für das Auto und die Straße.

**#3:** Die Klasse Car modelliert ein Auto. Das Car-Objekt kann nicht »von allein fahren«, sondern wird über die Methode `step()` immer wieder veranlasst, seine Position zu aktualisieren.

**#4:** Hier wird das Bild des Autos auf den Canvas in die Mitte nah am unteren Rand ( $y = 170$ ) gesetzt. Die ID-Nummer, die zurückgegeben wird, wird später immer wieder benötigt, um die Position des Autos zu verändern.

**#5:** Jeder gemessene Spannungswert (zwischen 0 und 3,3 Volt) entspricht einer Position in x-Richtung zwischen 0 px und WIDTH px. Die Spannung wird mit dem Potentiometer eingestellt. Hier wird geprüft, ob sich das Auto links oder rechts von dieser »Spannungsposition« befindet. Ist die x-Koordinate des Autos kleiner (es steht zu weit links), wird es ein Stück nach rechts bewegt. Ist sie größer, wird es nach links bewegt. Durch diesen Mechanismus bewegt sich das Auto niemals ruckartig, sondern gleichförmig.

**#6:** Die Klasse Road wird von der Klasse Canvas abgeleitet und stellt die Straße dar, über die das Auto fährt.

**#7:** Auf den Canvas wird das Bild der Straße gesetzt, und zwar so, dass der untere Teil zu sehen ist. Die Option `anchor=SW` bedeutet, dass die linke untere Ecke des Bilds als Bezugspunkt genommen wird. Auf dem Canvas hat die linke untere Ecke die Koordinaten (0, HEIGHT), denn der Nullpunkt der y-Achse ist am oberen Rand des Canvas.

**#8:** Jeder Aufruf der Methode `step()` bewirkt, dass das Bild mit der Straße auf dem Canvas ein Stück nach unten verschoben wird. Das passiert so lange, bis der obere Rand erreicht ist. Dann wird das Bild wieder auf seinen Anfangspunkt zurückgesetzt.

**#9:** Die Klasse Racing modelliert die gesamte Applikation.

**#10:** Die Methode `run()` wird bei der Initialisierung des `Racing`-Objekts einmal gestartet und ruft sich dann mit `after()` alle 50 ms immer wieder selbst auf. Sie steuert die Bewegung des Autos und der Straße, indem sie die `Road`- und `Car`-Objekte veranlasst, die Positionen ihrer Bilder auf dem Canvas zu aktualisieren.

**#11:** Ein `Racing`-Objekt wird instanziiert und damit die Applikation gestartet.

## Lösung zu Kapitel 10

### Programm

```
#!/usr/bin/python3
# howbig.py
import os, _thread, math, time                                #1
from tkinter import Tk, Label, Radiobutton, StringVar
from PIL import Image, ImageTk
WIDTH, HEIGHT = 800, 600
BBOX = (200,150,600, 450)                                    #2
PATH = "image.jpg"
COMMAND = "rpicam-still -t 200 --width {} --height {} -o {} -n"

class App:
    def __init__(self):
        # Widgets
        self.window = Tk()
        self.state = StringVar()                               #3
        self.labelImage = Label(master=self.window)
        self.labelImage.pack()
        self.label = Label(master=self.window)
        self.label.pack()
        self.measureRB = Radiobutton(master=self.window,
                                       text="Messen",
                                       variable=self.state,
                                       value="Messen")

        self.measureRB.select()
        self.measureRB.pack()
        self.calRB = Radiobutton(master=self.window,
                                   text="Kalibrieren",
                                   variable=self.state,
                                   value="Kalibrieren")

        self.calRB.deselect()
```



```

self.calRB.pack()

# Attribute
self.p0 = ()
self.factor = 1

# Operationen
self.reset()
self.labelImage.bind("<1>", self.click)
_thread.start_new_thread(self.takePhotos, ())    #4
self.window.mainloop()

def takePhotos (self):
    while True:

        os.system(COMMAND.format(WIDTH, HEIGHT, PATH))
        self.image = Image.open(PATH).crop(BBOX)    #5
        self.pImage=ImageTk.PhotoImage(self.image)
        self.labelImage.config(image=self.pImage)
        time.sleep(1)

def reset(self):
    self.p0 = ()    #6
    self.label.config(text="Auf den Anfang klicken")

def click(self, event):
    self.labelImage.unbind("<1>")    #7
    if not self.p0:    #8
        self.p0 = event.x, event.y    #9
        self.label.config(text='Auf das Ende klicken')
    else:
        x0, y0 = self.p0    #10
        x1, y1 = event.x, event.y    #11
        d = math.sqrt((x1 - x0)**2 + (y1 - y0)**2)    #12
        if self.state.get() == "Kalibrieren":
            self.factor = 10/d
            print self.factor
            text = "1 cm sind {:.1f} Pixel"
            self.label.config(text= text.format(1/self.factor))
        else:
            text = "Länge {:.2f} cm"

```

```
self.label.config(text=text.format(d * self.factor))
self.window.after(5000, self.reset)
self.labelImage.bind("<1>", self.click)
App()
```

### Kommentare

**#1:** Aus den Modulen werden die benötigten Namen gezielt importiert (und nicht mit \*), weil der Name `Image` sowohl in `PIL` als auch in `tkinter` vorkommt.

**#2:** Die Bounding-Box definiert ein Rechteck in der Mitte des Bilds. Der gesamte Bildausschnitt ist zu groß für eine schöne Darstellung kleiner Objekte, da die Kamera einen Mindestabstand von ca. 1 m wahren muss.

**#3:** Das Objekt repräsentiert den Zustand (MESSEN oder KALIBRIEREN) und wird mit den beiden Radiobuttons verbunden.

**#4:** Die Methode (die eine Endlosschleife enthält) wird in einem eigenen Thread ausgeführt.

**#5:** Wir verwenden nur einen Ausschnitt aus dem Foto (siehe **#2**).

**#6:** Zu Beginn eines Messvorgangs enthält das Attribut `self.p0`, das den ersten Messpunkt repräsentiert, das leere Tupel.

**#7:** Ab jetzt hat das Klicken der Maus keine Wirkung.

**#8:** Es gibt noch keinen Messpunkt.

**#9:** Speichere die Maus-Koordinaten als ersten Messpunkt.

**#10:** Koordinaten des ersten Messpunkts.

**#11:** Koordinaten des zweiten Messpunkts.

**#12:** Der Abstand zwischen den beiden Messpunkten wird mit dem Satz des Pythagoras berechnet.

## Lösungen zu Kapitel 12

### Lösung 1

#### Programm

```
# sekunden_animiert.py
import adafruit_ssd1306
import busio
from board import SCL, SDA
```

```

from PIL import Image, ImageFont, ImageDraw, ImageTk
from time import sleep
W, H = 128, 64 #1
i2c = busio.I2C(SCL, SDA)
disp = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c) #2
disp.fill(0)
image = Image.new('1', (W, H)) #3
draw = ImageDraw.Draw(image) #4
sec = 0 #5

while True: #6
    angle = -90 + sec * 6 #7
    draw.rectangle((0,0, W, H), fill=False) #8
    draw.pieslice((36, 4, 92, 60),
                  -90, angle, fill=True) #9
    disp.image(image) #10
    disp.show()
    sec = (sec + 1)%60 #11
    sleep(1) #12

```

### Kommentare

**#1:** In diesen beiden Konstanten merken wir uns Breite und Höhe des Displays.

**#2:** Hier wird ein Objekt erzeugt, das das Display repräsentiert.

**#3:** Hier wird ein Image-Objekt für ein Schwarz-Weiß-Bild erzeugt. Es hat genau die Maße des OLED-Displays.

**#4:** Ein neues Objekt zum Zeichnen wird erzeugt und mit dem Image-Objekt verbunden.

**#5:** Die Variable speichert die Anzahl der seit dem Start vergangenen Sekunden.

**#6:** Hier beginnt eine Endlosschleife.

**#7:** Hier wird der zweite Winkel des Kreisausschnitts berechnet. Beachten Sie, dass der Nullpunkt der Kreiswinkel rechts liegt (Osten). In der Animation soll aber der Kreisausschnitt oben beginnen, also bei  $-90^\circ$ .

Da eine Minute 60 Sekunden hat und ein Vollkreis einen Winkel von  $360^\circ$  umfasst, wird jede Sekunde durch einen Kreisausschnitt von  $6^\circ$  dargestellt.

**#8:** Ein schwarzes Rechteck wird gezeichnet und damit das Bild auf dem Display gelöscht.

**#9:** Das Tupel (36, 4, 92, 60) sind die Koordinaten der linken oberen und der rechten unteren Ecke eines Quadrats in der Mitte des Displays. Es ist die Bounding-Box für den Kreisausschnitt, der gezeichnet werden soll. Der Kreisausschnitt beginnt oben (-90°) bis zu einem Winkel, der der Sekundenzahl entspricht.

**#10:** Das aktuelle Bild mit einem Kreisausschnitt wird auf dem Display dargestellt.

**#11:** Die Sekundenzahl wird um 1 erhöht. Nach 60 Sekunden wird wieder bei 0 begonnen.

**#12:** Eine Sekunde lang warten.

## Lösung 2

### Programm

Die Abweichungen vom ursprünglichen Programmtext sind fett gesetzt.

```
import sense_hat
from time import sleep
from random import randint
WHITE = (255, 255, 255) #1

def move_up(event):
    global y
    if event.action != "released":
        y = (y - 1) % 8
    ...
def refresh():
    sh.clear()
    if (x, y) == (x_, y_): #2
        new_game()
    else:
        sh.set_pixel(x_, y_, (0, 255, 0)) #3
        sh.set_pixel(x, y, (255, 0, 0))

def new_game(): #4
    global x_, y_ #5
    sh.clear(WHITE) #6
    sleep(0.2)
    sh.clear() #7
    x_, y_ = randint(0, 7), randint(0, 7) #8
    refresh()
```

```
x = y = 0
sh = sense_hat.SenseHat()
new_game()
sh.stick.direction_any = refresh
sh.stick.direction_up = move_up
sh.stick.direction_down = move_down
sh.stick.direction_left = move_left
sh.stick.direction_right = move_right
```

### Kommentare

**#1:** Hier wird eine Konstante für die Farbe Weiß definiert (maximale Werte für Rot, Grün und Blau).

**#2:** Wenn das rote Pixel die gleiche Position wie das grüne Pixel hat, ist das Ziel erreicht und es wird ein neues Spiel gestartet.

**#3:** Ansonsten werden das grüne und das rote Pixel auf die inzwischen eventuell geänderten Positionen gesetzt.

**#4:** Diese Funktion startet ein neues Spiel: Es gibt einen Blitz und das grüne Pixel wird auf eine Zufallsposition gesetzt.

**#5:** Dies sind die Koordinaten des grünen Pixels. Die Variablen sind global, weil beim Refresh darauf zugegriffen werden muss.

**#6:** Alle LEDs leuchten weiß.

**#7:** Nach 0,2 Sekunden werden alle LEDs dunkel.

**#8:** Für das grüne Pixel werden neue Zufallspositionen bestimmt.