| Fachbereich<br>Automatisierung und Informatik | ▲ Hochschule Harz<br>Wernigerode |
|---|---|

Vorlesung „Linux“

# Python-Kurzskript

**Version 14.06.2019**

**Dipl.-Inf., Dipl.-Ing. (FH) Michael Wilhelm**
**Friedrichstraße 57 - 59**
**38855 Wernigerode**

**Raum:** **2.202**
**Tel.:** **03943/659-338**
**Fax:** **03943/659-399**
**Email:** **mwilhlem@hs-harz.de**
**Web:** **http://www.miwilhelm.de**

# Inhaltverzeichnis

# Abbildungen

# 1 Python

## *1.1 Eigenschaften*

- Variablen à la Java
- Datentypen à la Java, aber auch die unsigned-Typen
- Klammern{} durch Doppelpunkt und Einrücken
- if à la Java
- Schleifen à la Java, mit foreach
- Prozeduren à la Java-Methoden
- Klassen
    - Attribute anders
    - Keine privaten Attribute (_number)
- I/O
    - Komfortables Framework
    - foreach line file
- **Keine Pointer** ☹
- Threads und Semaphore, Mutex
- UI-Elemente mit Tkinter und Qt

### 1.1.1 Komplexe Zahlen

zahl1 = 3 + 4j
zahl2 = 4j  // 0+4j

**Methoden:**

- zahl1.real
- zahl1.imag
- zahl1.conjugate  Konjugierte komplexe Zahl  3+4j ⇒ 3-4j

### 1.1.2 Decimal

Decimal kann mit beliebiger Genauigkeit rechnen. Das geht natürlich auf die Rechenzeit, aber Rundfehler sind passé. Gut für Buchführungsprogramme. Es ist aber nicht mit dem decimal-Typ von C# zu vergleichen, eher mit BigDecimal (leider).

Einbinden des Moduls:

```
from decimal import Decimal
```

**Beispiele:**

```
Decimal("0.9")
```

## *1.2 If-Anweisungen*

```
a = 5
b = 7
if a > b:
  print("a ist größer als b")
  print("a ist größer als b")
else:
  print("a ist kleiner als b")
  print("a ist kleiner als b")

if a > b:
  pass                    // keine Aktion
else:
  print("a ist kleiner als b")

if a == 1:
  print("a gleich 1")
elif: a ==2
  print("a gleich 2")
elif: a ==3
  print("a gleich 3")
else:
  print("Error")
```

## *1.3 Switch/Case*

Diese Anweisung gibt es nicht in Python.

## *1.4 Schleifen*

### 1.4.1 While-Schleifen

```
d:int = 0
e:int = 10
r:int = 2

while d<=e:
   d++
   print(d)
   if d>r:
      break
else:
   print("Die Schleife wurde nicht mit break abgebrochen")
print("nach der Schleife")
```

### 1.4.2 For-each-Schleife

```
for c in "Einmal...":
   print("c")


for c in "Einmal...":
   print(c)
else:
   print("Die Schleife wurde nicht mit break abgebrochen")
```

### 1.4.3 For-Schleife

Range-Varianten:

- range(n) i=0 i<n i++
- range(start,n) i=start i<n i++
- range(start,n,step) i=start i<n i+=step

```
for i in range(1,10,2):
   print(i)
else:
   print("Die Schleife wurde nicht mit break abgebrochen")
```

```
for i in range(0, -10, -2):
   print(i)
...
Ausgabe:
   0
   -2
   -4
   -6
  -8

nLen=5
ii=nLen-1
while ii>=0:
   print(ii,nLen)
   ii-=1

Ausgabe:
   5
   4
   3
   2
   1
   0
```

# 2 Sequenzen

## *2.1 Allgemeine Methoden der Sequenzen*

- x in str
- x not in str
- w = s+t
- s +=t
- s *=t n× hintereinander
- s[i]
- s[i:j]
- s[i:j:k]
- len(s)
- max(s)
- min(s)
- s.index( x ) sucht das erste Element von x
- s.index( x, i ) sucht das erste Element von x ab i
- s.index( x, i, j ) sucht das erste Element von x ab i bis j
- s.count("Einmal...“) wie oft x?

## *2.2 str / string*

- Speichert String und Texte von Zeichen.
- Hat eigene I/O-Operationen

**Methoden:**

- s.endswith(suffix)
- s.startswith(suffix)
- len(s)
- lstrip(s) # alle Whitespace am Anfang
- rstrip(s) # alle Whitespace am Ende
- strip(s) # alle Whitespace am Anfang und am Ende
- s.lower()
- s.upper()
- s.split(separator) # Suche startet am Anfang
- s.split(separator, maxsplit)
- s.rsplit(separator) # Suche startet am Ende
- s.rsplit(separator, maxsplit)
- s.splitlines() # trennen beim Zeilenende
- s.splitlines(keepends)

- s.partition(separator) # Startet am Anfang, enthält auch separator
- s.rpartition(separator) # Startet am Ende, enthält auch separator
- s.find(sub[,start[,end]])
- s.rfind(sub[,start[,end]])
- s.index(sub[,start[,end]]) # sucht String sub, wirft ev. Exception
- s.rindex(sub[,start[,end]]) # sucht String sub, wirft ev. Exception
- s.count(sub[,start[,end]]) # zählt die Anzahl von sub
- s.replace(old, new[,count])
- s.lower()
- s.upper()
- s.swapcase() # A→a a→A
- c.capitalize() # einmal→Einmal nur am Anfang
- s.title # einmal→Einmal alle Wörter
- s.expandtab([tabsize]) # tab durch n-Leerzeichen ersetzen
- s.strip([chars]) # trim mit beliebigen Zeichen
- s.lstrip([chars]) # ltrim mit beliebigen Zeichen
- s.rstrip([chars]) # rtrim mit beliebigen Zeichen
- s.center(width[,fillchar])
- s.ljust(width[,fillchar]) # justieren linksbündig
- s.rjust(width[,fillchar]) # justieren rechtsbündig
- s.zfill(width) # ausfüllen mit Nullen (padleft, padright in C#)
- s.join(seq)
- substring
    - str = "1234567890"
    - strslice = str[len(str)-4:len(str)]

**Numerische Methoden:**

- s.isalnum()
- s.isalpha()
- s.isdigit()
- s.islower()
- s.isupper
- s.isisspace()
- s.istitle # true, wenn alle Wörter großgeschrieben sind
- s.startswith(prefix[,start[,end]]) # true, wenn s mit prefix beginnt
- s.endswith(prefix[,start[,end]]) # true, wenn s mit prefix ended (Backslash)

**Problem:**

- Zuweisungen sind nicht immer Referenzen (String)
- Bei Änderung ⇒ Scheidung

**Beispiel:**

```
a = "Hallo "
b = a               # hier Referenz
b += "Welt"
b                   # Ausgabe "Hallo Welt"
a                   # Ausgabe "Hallo"

print:
"Summe der Gehälter: {0:.2f} ".format(summe)
"Summe: {0:,d} ".format(summe)        # dezimaltrenner
fobj.write("Value: {0:.2f}\r\n".format(value))
```

## *2.3 bytes*

Wird benutzt zum Einlesen von Binärdateien.

## *2.4 bytearray*

Wird benutzt zum Einlesen von Binärdateien.

## *2.5 list*

- dynamische Liste
- Speichert beliebige Datentypen
- Änderbar

**Erzeugen einer Liste**

- list=[]

**Operationen:**

- s[i] = t — Ersetzt das i-te Element
- s[i:j] = t — Ersetzt die i-te bis j.te Elemente
- s[i:j:k] = t — von I bis j Schrittweite k
- del s[i] — Löscht das i-te Element
- del s[i:j] — Löscht die i-te bis j.te Elemente
- del s[i:j:k]
- del liste[ 0:len(liste) ] — Löschen aller Elemente in einer Liste:

**0 <= i < n**

**Methoden:**

- s.append(x)
- s.extend(t) — t-Liste an s-Liste

- s.insert(i,x) Einfügen von x an i. ter Stelle
- s.pop() Letzte Element + löschen
- s.pop(i) Ausgabe des i. ten Element + löschen
- s.remove(x) x Object
- s.reverse()
- s.sort()
- s.sort(key) s.sort(key=len) nach Länge
- s.sort(key, reverse)

**Problem:**

- Zuweisungen sind nicht immer Referenzen (String)
- Bei Änderung ⇒ Scheidung

**Beispiel:**

```
a = "Hallo "
b = a               # hier Referenz
b += "Welt"
b                   # Ausgabe "Hallo Welt"
a                   # Ausgabe "Hallo"

a = [1337]
b = a               # hier Referenz
c = a[:]            # Kopie
b += [2674]
b                   # Ausgabe [1337, 2674]
a                   # Ausgabe [1337, 2674]
c                   # Ausgabe [1337]
a is b              # true

a=[1,2,3,4,5,6,7,8,9]
a[::-1]
[9, 8, 7, 6, 5, 4, 3, 2, 1]
a[::+1]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
a[::+2]
[1, 3, 5, 7, 9]
len(a)                    # Anzahl der gespeicherten Elemente
```

## *2.6 tuple*

- dynamische Liste
- Speichert beliebige Datentypen
- Nicht änderbar

**Erzeugen eines Tupels**

- tupel=()

**Beispiel:**

```
a = (1,2,3,4)
a[3]                # Ausgabe 4

a = (1,2)          # okay
a = (2)            # nicht okay, ist nur die Zahl
a = (2,)           # okay
print(len(a))      # 4
print(a[0])        # 1
```

## *2.7 dict*

- Hashtable
- Speichert beliebige Datentypen
- Iteration über die Elemente ist sehr einfach

**Beispiele:**

```
noten = {27456:1.3 , 27457:3.3 , 27459:2.7 , "Brücke":"Extrakäse" }
for key in noten:
  print(key)

plants = {}
plants["radish"] = 2
plants["squash"] = 4
plants["carrot"] = 7

# Get syntax 1.
print(plants["radish"])

# Get syntax 2.
print(plants.get("tuna"))
print(plants.get("tuna", "no tuna found"))
```

**Methoden:**

- len(d) # Anzahl der gespeicherten Elemente
- d[k] # hole Wert von k
- del d[k] # lösche Wert k
- k in d # ist k in d?
- k not in d # ist k nicht in d?
- d.clear()
- d.copy()
- d.get(k[,x]) # holt den Wert k oder x
- d.items() # Menge aller Key und Werte, foreach mit Tupel
- d.keys() # Menge aller Keys, foreach mit keys
- d.pop(k) # erhalte Wert und lösche k
- d.popitem() # gibt einen zufälligen Tupel zurück, löscht es
- d.setdefault(k[,x]) # setzt key[k] = x, wenn Schlüssel noch nicht vorhanden ist

- d.update(d2) # Einfügen und ev. Überschreiben
- d.values() # Menge aller Werte, foreach mit einem Wert

## *2.8 Mengen*

Mengen können beliebige Datentypen speichern. Jedes Element darf aber nur einmal enthalten sein. Bei den Mengen unterscheidet man zwischen unveränderlichen und veränderlichen Mengen:

- set() leere veränderliche Menge
- frozenset() leere unveränderliche Menge
  - fs = frozenset( True, 123, "Einmal…")
  - fs = { True, 123, "Einmal…"}

**Durchlaufen der Menge:**

```
s = set( (2,3,5,7) )
for item in s:
  print(item)
```

**Methoden:**

- len(menge)
- x in menge
- x not in menge
- menge1 <= menge2
- menge1 < menge2
- menge1 >= menge2
- menge1 > menge2
- menge1 / menge2 # Vereinigungsmenge
- menge1 & menge2 # Schnittmenge
- menge1 - menge2 # Differenzmenge
- menge1 ^ menge2 # symmetrische Differenzmenge, alle die nur in einer Menge vorhanden sind

**Methoden für set (veränderliche Menge):**

- m1.add(e)
- m1
- m1.clear()
- m1.difference_update(m2) # Äquivalent zu m1 -= m2
- m1.discard(e) # Löscht das Element
- m1.intersection_update(m2) # Äquivalent zu m1 &= m2
- m1.remove(e) # Löscht das Element ev. Exception
- m1.symmetrisch_update(m2) # Äquivalent zu m1 ^= m2
- m1.update(m2) # Äquivalent zu m1 /= m2

# 3 Eingebaute Funktionen

## *3.1 Eingabe- Ausgabefunktionen*

| | |
|---|---|
| `input([prompt])` | Liest einen String vom Eingabestream |
| `input` | |

## *3.2 math*

Die Funktionen sind in der Bibliothek „math“ definiert:

**Konstanten:**

- e
- pi

### 3.2.1 Mathematische Rundungsfunktionen:

| | |
|---|---|
| `round(x[,n])`<br>`round(12.5)  12` | Rundet auf n-Nachkommastellen |
| `math.ceil(x)` | Rundet auf die nächsthöhere Ganzzahl |
| `math.ceil(x)`<br>`math.ceil(12.2)  13`<br>`math.ceil(12.5)  13`<br>`math.ceil(12.8)  13`<br>`math.ceil(-12.2)  -12`<br>`math.ceil(-12.5)  -12`<br>`math.ceil(-12.8)  -12` | Rundet auf die nächsthöhere Ganzzahl |
| `math.floor(x)`<br>`math.floor(12.2)  12`<br>`math.floor(12.5)  12`<br>`math.floor(12.8)  12`<br>`math.floor(-12.2)  -13`<br>`math.floor(-12.5)  -13`<br>`math.floor(-12.8)  -13` | Rundet auf die nächstniedrigere Ganzzahl |
| `isnan(x)` | Gibt True zurück, wenn x nan ist. |
| `ldexp(m,e)` | Bestimmt Zahl aus Mantisse und Exponent |
| `math.trunc(x)`<br>`math.trunc(12.5)  12`<br>`math.trunc(12.2)  12`<br>`math.trunc(12.8)  12`<br>`math.trunc(-12.2)  -12`<br>`math.trunc(-12.5)  -12`<br>`math.trunc(-12.8)  -12` | Gibt den Vorkommaanteil zurück (à floor)<br>Der Unterschied liegt in den negativen Zahlen. |
| **`Quellcode`** | **Ausgabe** |

| | |
|---|---|
| d=12.4<br>i=round(d) | 12 |
| d=12.4<br>i=int(d) | 12 |
| d=12.4<br>i=round(d+0.0001) | 12 |
| d=12.4<br>i=int(d+0.0001) | 12 |
| | |
| d=12.5<br>i=round(d) | 13 |
| d=12.5<br>i=int(d) | 12 |
| d=12.5<br>i=round(d+0.0001) | 13 |
| d=12.5<br>i=int(d+0.0001) | 12 |
| | |
| d=12.45<br>v=round(d,1) | 12,4 |
| d=12.45<br>v=round(d+0.0001,1) | 12,5 |
| | |
| d=12.452<br>v=round(d,2) | 12,45 |
| d=12.452<br>v=round(d+0.00001,2) | 12,45 |
| d=12.456<br>v=round(d,2) | 12,46 |
| d=12.456<br>v=round(d+0.00001,2) | 12,46 |

### 3.2.2 Allgemeine mathematische Funktionen:

import math
abs(-22)

| | |
|---|---|
| abs(x) | Absolutwert bei Integer-Zahlen |
| chr(i)<br>chr(65) 'A' | Ausgabe des i-Zeichens (Ascii-Code) |
| compile(source, filename, mode [,flags [,dont_inherit [,options]]]) | Übersetzt einen String oder eine Datei in ein ausführbares Objekt |
| complex([real, [,imag]]) | Erzeugt eine komplexe Zahl |
| float([x]) | Erzeugt eine Gleitkommazahl |
| hex(x)<br>hex(241) 0xF1<br>hex(241)[2:] F1 | Gibt den Hexadezimalwert der ganzen Zahl x in Form eines Strings zurück |
| min | Minimum |

| | |
|---|---|
| `min([1,2,3]  1` | |
| `max`<br>`max([1,2,3]  3` | Maximum |
| `oct(x)`<br>`oct(16)  '0o20'` | Gibt den Oktalwert der ganzen Zahl x in Form eines zurück |
| `ord(c)`<br>`ord('a')  97` | Gibt den Unicodewert des Zeichen c aus. |
| `pow(x[,y[,y]])`<br>`pow(2,3)  8` | Potenzfunktion |
| `round(x[,n])`<br>`round(12.5)  12` | Rundet auf n-Nachkommastellen |
| `sum`<br>`sum([1,2,3])  6` | Summierung |
| `print(?)` | Ausgabe von String und Variablen |

| | |
|---|---|
| `math.ceil(x)` | Rundet auf die nächsthöhere Ganzzahl |
| `copysign(x)` | Überträgt das Vorzeichen |
| `math.fabs(x)`<br>`math.fabs(-12.3)  12.3` | Absolutwert bei Floating-Points |
| `factorial(x)` | Fakultät |
| `math.ceil(x)`<br>`math.ceil(12.2)  13`<br>`math.ceil(12.5)  13`<br>`math.ceil(12.8)  13`<br>`math.ceil(-12.2)  -12`<br>`math.ceil(-12.5)  -12`<br>`math.ceil(-12.8)  -12` | Rundet auf die nächsthöhere Ganzzahl |
| `math.floor(x)`<br>`math.floor(12.2)  12`<br>`math.floor(12.5)  12`<br>`math.floor(12.8)  12`<br>`math.floor(-12.2)  -13`<br>`math.floor(-12.5)  -13`<br>`math.floor(-12.8)  -13` | Rundet auf die nächstniedrigere Ganzzahl |
| `math.fmod(x,y)`<br>`math.fmod(12,5)  2.0` | Berechnet Modulo |
| `frexp(x)` | Extrahiert Mantisse und Exponent |
| `fsum(iterable)` | Summiert die „Liste, Menge“ |
| `isfinite(x)` | Gibt True zurück, wenn x weder inf, -inf oder nan ist |
| `isinf(x)` | Gibt True zurück, wenn x entweder inf oder -inf ist |
| `isnan(x)` | Gibt True zurück, wenn x nan ist. |
| `ldexp(m,e)` | Bestimmt Zahl aus Mantisse und Exponent |
| `math.trunc(x)`<br>`math.trunc(12.5)  12`<br>`math.trunc(12.2)  12`<br>`math.trunc(12.8)  12`<br>`math.trunc(-12.2)  -12`<br>`math.trunc(-12.5)  -12`<br>`math.trunc(-12.8)  -12` | Gibt den Vorkommaanteil zurück (à floor)<br>Der Unterschied liegt in den negativen Zahlen. |

### 3.2.3 Exponential und Logarithmusfunktionen

| | |
|---|---|
| `math.exp(x)`<br>`math.exp(1)  2.718281828459045` | $e^x$ |
| `math.expm1(x)`<br>`math.expm1(1)  1.718281828459045` | $e^x$-1. Ist genauer als exp(x)-1. |
| `math.log(x[,base])`<br>`math.log(100,10)  2.0` | Logarithmus mit Basis |
| `math.log(x)`<br>`math.log(11) 2.3978952727983707` | Logarithmus Naturalis, zur Basis e |
| `math.log10(x)`<br>`math.log10(12) 1.0791812460476249` | Dekadischer Logarithmus |
| `math.log1p(x)`<br>`math.log1p(2)  1.0986122886681098` | Berechnet ln(1+x). Ist genauer als log(x)+1. |
| `math.pow(x,y)`<br>`math.pow(2,3)  8.0` | $x^y$ |
| `math.sqrt(x)`<br>`math.sqrt(2)  1.4142135623730951` | Quadratwurzel |

### 3.2.4 Trigonometrische Funktionen

| | |
|---|---|
| `double acos(double x)` | arccos(x) im Bereich [0, p], x ∈ [-1, 1] |
| `double asin(double x)` | arcsin(x) im Bereich [-p/2, p/2], x ∈ [-1, 1] |
| `double atan(double x)` | arctan(x) im Bereich [-p/2, p/2] |
| `double atan2(double y, double x)` | arctan(y/x) im Bereich [-p, p] |
| `double cos(double x)` | Kosinus von x |
| `hypot(x)` | Euklidsche Norm |
| `sin(x)` | sin(x) |
| `tan(x)` | Tangens von x |
| `degrees(x)` | Bogenmaß in Grad: *180/π |
| `radians(x)` | Grad in Bogenmaß in Grad: *π/180 |
| `acosh(x)` | ArcCosinus Hyperbolicus von x |
| `asinh(x)` | ArcSinus Hyperbolicus von x |
| `atanh(x)` | ArcTangens Hyperbolicus von x |
| `cosh(x)` | Cosinus Hyperbolicus von x |
| `sinh(x)` | Sinus Hyperbolicus von x |
| `tanh(x)` | Tangens Hyperbolicus von x |

### 3.2.5 Statistikfunktionen

| | |
|---|---|
| `erf(x)` | Gauß'sche Fehlerfunktion |
| `erfc(x)` | Berechnet 1.0-erf(x) |
| `gamma(x)` | Gammafunktion an der Stelle x |
| `lgamma(x)` | Berechnet den natürlichen Logarithmus des Betrags der Gammafunktion an der Stelle x. |

### 3.2.6 cmath

Die Funktionen sind in der Bibliothek cmath definiert:

| | |
|---|---|
| `phase(x)` | Berechnet die Phase der komplexen Zahl x. |
| `polor(x)` | Berechnet die Polardarstellung der komplexen Zahl x. |
| `rect(r,phi)` | Konvertiert die Polardarstellung mit dem Radius und Winkel in ihre kartesische Darstellung. |

### 3.2.7 Random

Dieses Kapitel behandelt die Benutzung von Zufallszahlen.
Dazu muss man das Modul Random einbinden:

```
import random
```

| | |
|---|---|
| `ssed([x],version)` | Initialisiert den Zufallsgenerator |
| `randint(a,b)` | Ganzahlige Zufallszahlen |
| `randrange([start][,stop][,step])` | Erzeugt eine Zufallszahl im angegebenen Bereich |
| `random()` | Zufallszahlen 0 bis <1 |

## *3.3 Datumsfunktionen*

```
import time

now = time.localtime()
print("Tag:", now.tm_mday)
print("Monat:", now.tm_mon)
print("Jahr:", now.tm_year)
print("Stunde:", now.tm_hour)
print("Minute:", now.tm_min)
print("Sekunde:", now.tm_sec)
print("Wochentag:", now.tm_wday)  # Montag = 0
print("Tag des Jahres:", now.tm_yday)
print("Sommerzeit:", now.tm_isdst)  # Sommerzeit: 1; Winterzeit: 0
```

# 4 Eigene Funktionen

Es wird nicht definiert, ob eine Funktion einen Rückgabewert hat.

**Syntax:**

```
def  Funktionsname(Parameter1, Parameter2):
  Anweisung1
  Anweisung2

def  Funktionsname(Parameter1, Parameter2):
  Anweisung1
  Anweisung2
  return x                oder None
```

**Beispiel:**

```
def  fakultaet(n):
  ergebnis = 1
  for i in range(2,n+1):              # Ende zählt nicht mit
    ergebnis *= i
  return ergebnis
```

## *4.1 Optionale Parameter*

```
def  Funktionsname(Parameter1, Parameter2=2):
  Anweisung1
  Anweisung2
```

## *4.2 Beliebige Parameter*

```
def  Funktionsname(Parameter1, *weitereParameter):
  print(weitereParameter)           # hier Ausgabe als Liste
```

## *4.3 lokale Funktionen*

```
def  Funktionsname1(Parameter1):

  def  Funktionsname2(Parameter2):
    Anweisung1

  Anweisung1
  Funktionsname2(42)
  Anweisung2
```

# 5 Exception

## *5.1 Exception abfangen*

**Prinzip:**

```
try:
  Anweisungen
except:
  Anweisungen
else:
  Anweisungen             # wenn keine Exception ausgeführt wird
finally:
  Anweisungen             # wird immer ausgeführt
```

### 5.1.1 Beispiele

```
def openFilename(filename):
  try:
    return open(filename)
  except IOError:
    return None
  except IndexError as e:
    print("Fehlermeldung: ",e.args[0]);
    return None
```

```
def openFilename(filename):
  try:
    return open(filename)
  except (IOError, IndexError) as e:
    print("Fehlermeldung: ",e.args[0]);
    return None
```

```
def openFilename(filename):
  try:
    return open(filename)
  except (IOError, IndexError) as e:
    print("Fehlermeldung: ",e.args[0]);
    return None
  else:
    print("Datei konnte geöffnet werden");
  finally:
    print("ende Fkt openFilename");  # wird immer ausgeführt
```

## *5.2 Dateioperationen*

```
def get(filename):
  try:
    return open(filename)
```

```
    except IOError:
      return None

def get(filename):
    try:
      return open(filename)
    except (IOError, TypeError):
      return None

def get(filename):
    try:
      return open(filename)
    except IOError:
      return None
    except TypeError:
      return None

def get(filename):
    try:
      return open(filename)
    except (IOError, TypeError):
      return None
    else:
```

### 5.2.1 Modi für Dateien

| | |
|---|---|
| "r" | Datei wird zum Lesen geöffnet |
| "w" | Datei wird zum Schreiben geöffnet. Eine vorhandene Datei wird gelöscht. |
| "a" | Datei wird zum Schreiben geöffnet. Eine vorhandene Datei wird nicht gelöscht (append). |
| "r+"<br>"a+" | Datei wird zum Lesen und Schreiben geöffnet. Eine vorhandene Datei wird nicht gelöscht. |
| "w+" | Datei wird zum Lesen und Schreiben geöffnet. Eine vorhandene Datei wird gelöscht. |
| "rb"<br>"wb"<br>"ab"<br>"r+b"<br>"w+b"<br>"a+b" | Datei wird im Binärmodus geöffnet. |

```
fobj = open(filename,modi[,buffersize [,endocing [,errors [,newline]]]] )
fobj = open('filenewline2.txt','r', 2048, None,'\n')
```

### 5.2.2 Methoden

| | |
|---|---|
| read([size] | Liest size Bytes oder weniger. |
| readline([size] | Liest eine Zeile oder bis size-bytes, à la fgets. |
| readlines([size] | Liest alle Zeilen der Datei oder bis size-bytes gelesen wurde. |
| seek(offset[,whence]) | Positioniert den Dateizeiger (nur Binärmodus)<br>whence=0 vom Anfang |

| | |
|---|---|
| | whence=1 relativ zur aktuellen Position<br>whence=2 vom Ende |
| tell() | Liefert die aktuelle Dateiposition. |
| truncate([size]) | Abschneiden des Inhaltes der Datei. |
| write(str) | Schreibt einen String in die Datei. |
| writelines(iterable) | Schreibt die Strings au seiner “Liste” in die Datei. |

### 5.2.3 Datei lesen:

```
fobj = open('io1.py','r')
for line in fobj:
  print(line)

fobj.close()
```

### 5.2.4 Datei schreiben:

```
fobj = open('erg.txt','w')
fobj.write("abc\r\n")  # ohne Zeilenumbruch

value1 = 12345.456789
value2 = 123456789
fobj.write("Value: {:.2f}\r\n".format(value1))
fobj.write("Value: {:.2f}\r\n".format(value2))
fobj.write("Value: {:15.2f}\r\n".format(value1))
fobj.write("Value: {:15.2f}\r\n".format(value2))

fobj.close()
fobj.write("{}{}\r\n".format('abc','def'))
```

## *5.3 Arten der Exception*

BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
    +-- StopIteration
    +-- StandardError
    |  +-- BufferError
    |  +-- ArithmeticError
    |  |  +-- FloatingPointError
    |  |  +-- OverflowError
    |  |  +-- ZeroDivisionError
    |  +-- AssertionError
    |  +-- AttributeError

```
|   +-- EnvironmentError
|   |   +-- IOError
|   |   +-- OSError
|   |       +-- WindowsError (Windows)
|   |       +-- VMSError (VMS)
|   +-- EOFError
|   +-- ImportError
|   +-- LookupError
|   |   +-- IndexError
|   |   +-- KeyError
|   +-- MemoryError
|   +-- NameError
|   |   +-- UnboundLocalError
|   +-- ReferenceError
|   +-- RuntimeError
|   |   +-- NotImplementedError
|   +-- SyntaxError
|   |   +-- IndentationError
|   |       +-- TabError
|   +-- SystemError
|   +-- TypeError
|   +-- ValueError
|       +-- UnicodeError
|           +-- UnicodeDecodeError
|           +-- UnicodeEncodeError
|           +-- UnicodeTranslateError
+-- Warning
    +-- DeprecationWarning
    +-- PendingDeprecationWarning
    +-- RuntimeWarning
    +-- SyntaxWarning
    +-- UserWarning
    +-- FutureWarning
  +-- ImportWarning
  +-- UnicodeWarning
  +-- BytesWarning
```

### 5.3.1 exception BaseException

The base class for all built-in exceptions. It is not meant to be directly inherited by user-defined classes (for that, use Exception). If str() or unicode() is called on an instance of this class, the representation of the argument(s) to the instance are returned, or the empty string when there were no arguments.

args

The tuple of arguments given to the exception constructor. Some built-in exceptions (like IOError) expect a certain number of arguments and assign a special meaning to the elements of this tuple, while others are usually called only with a single string giving an error message.

### 5.3.2 exception ArithmeticError

The base class for those built-in exceptions that are raised for various arithmetic errors: OverflowError, ZeroDivisionError, FloatingPointError.

### 5.3.3 exception AttributeError

Raised when an attribute reference (see Attribute references) or assignment fails. (When an object does not support attribute references or attribute assignments at all, TypeError is raised.)

### 5.3.4 exception EOFError

Raised when one of the built-in functions (input() or raw_input()) hits an end-of-file condition (EOF) without reading any data. (N.B.: the file.read() and file.readline() methods return an empty string when they hit EOF.)

### 5.3.5 exception IOError

Raised when an I/O operation (such as a print statement, the built-in open() function or a method of a file object) fails for an I/O-related reason, e.g., “file not found” or “disk full”.

This class is derived from EnvironmentError. See the discussion above for more information on exception instance attributes.

### 5.3.6 exception IndexError

Raised when a sequence subscript is out of range. (Slice indices are silently truncated to fall in the allowed range; if an index is not a plain integer, TypeError is raised.)

### 5.3.7 exception MemoryError

Raised when an operation runs out of memory but the situation may still be rescued (by deleting some objects). The associated value is a string indicating what kind of (internal) operation ran out of memory. Note that because of the underlying memory management architecture (C’s malloc() function), the interpreter may not always be able to completely recover from this situation; it nevertheless raises an exception so that a stack traceback can be printed, in case a run-away program was the cause.

### 5.3.8 exception OverflowError

Raised when the result of an arithmetic operation is too large to be represented. This cannot occur for long integers (which would rather raise MemoryError than give up) and for most operations with plain integers, which return a long integer instead. Because of the lack of standardization of floating point exception handling in C, most floating point operations also aren't checked.

### 5.3.9 exception ValueError

Raised when a built-in operation or function receives an argument that has the right type but an inappropriate value, and the situation is not described by a more precise exception such as IndexError.

Beispiel:

```
    try:
       a = int('abc')
    except ValueError as e:
       print("fehlerhafte Zahl")
```

```
def isfloat(value):
  try:
    float(value)
    return True
  except:
    return False
```

### 5.3.10 exception ZeroDivisionError

Raised when the second argument of a division or modulo operation is zero. The associated value is a string indicating the type of the operands and the operation.

The following exceptions are used as warning categories; see the warnings module for more information.

## *5.4 Eigene Exception auslösen*

```
def fakultaet(n):
  if n < 0:
    raise fakultaet
  p=1
  for i in range(2,n+1):              # Ende zählt nicht mit
    p *= i
  return p
```

# 6 Klassen

## *6.1 Änderungen gegenüber Java / C#*

- Alle dynamischen Methoden erhalten self als ersten Parameter
- Der Konstruktor heißt **_ _init_ _(self)_ _ (ohne Leerzeichen)**
- Es gibt keinen De-Konstruktor, aber etwas Ähnliches.
- Attribute sollten im Konstruktor angelegt werden.
- Es gibt kein „private“.
- **Konvention:**
  - Zugriff auf _meinePrivateVariable ist per Konvention „verboten“.
  - Stichwort Unterstrich: _
- Setter und Getter sind möglich
  - `def __init__(self):`
    - `# hier sind die Variablendeklarationen!`
    - `self._x = 100`
  - `def getX(self)`
    - `return _x`
  - `def setX(self, wert)`
    - `self._x=wert`
- **`toString`**`()`
  - `def __str__(self):`
  - `return str(self._x)`
- `Compare, sort`
  - `# compare-Methode, lessThan,`
  - `definiert die Reihenfolge zw. Objekten`
  - `def __lt__(self, other):`
  - `return self.getX() < other.getX() # aufsteigend sortiert`
- **Vererbung mehrfach erlaubt**: `class KlasseB(KlasseA, IDrucken)`
- Überschreiben und Überladen erlaubt.
- statische Methoden sind erlaubt, ist aber umständlich
- **Operatoren können überladen werden!**
  - Unäre Operatoren überladen + - abs ~
  - Binäre Operatoren überladen
    - += -= *= /= //= **=
    - %= >>= <<= &= |= ^=

### 6.1.1 Ableitung

```
class A:
  def __init__(self):
    self._x = 1337
    print("Konstruktor von A")

  def m(self):
    print("Methode m von A. Es ist self._x =", self._x)


class B(A):
  def __init__(self):
    A.__init__(self) # nun hat B auch das Attribut X!
    self._y = 10000
    print("Konstruktor von B")

  def n(self):
    print("Methode n von B. Es ist self._y =", self._y)

b = B()
b.n()
b.m()
```

### 6.1.2 Interface

Ein Interface wird nicht benötigt, da man von **mehreren** Oberklassen ableiten kann:

```
#! /usr/bin/env python3
# coding=utf8



class Flugzeug:
⟶def __init__(self, name, laenge):
⟶⟶self.name = name
⟶⟶self.laenge = laenge
⟶⟶print("Konstruktor von Flugzeug")

⟶def drucken(self):
⟶⟶print("Flugzeug ",self.name,"Länge: ", self.laenge)
```

```
#! /usr/bin/env python3
# coding=utf8

# mit interface

class IDrucken(object):
⟶def __init__(self):
⟶⟶pass

⟶def drucken(self):
⟶⟶raise Exception("NotImplemented")


class Flugzeug(IDrucken):
⟶def __init__(self, name, laenge):
⟶⟶self.name = name
⟶⟶self.laenge = laenge
⟶⟶print("Konstruktor von Flugzeug")


⟶def drucken(self):
⟶⟶print("Flugzeug ",self.name,"Länge: ", self.laenge)
```

```
class Frachtflugzeug(Flugzeug):
→def __init__(self, name, laenge, fracht):
→→Flugzeug.__init__(self,name, laenge)
→→self.fracht = fracht
→→print("Konstruktor von Frachtflugzeug")

→def drucken(self):
→→print("Frachtflugzeug",self.name,"Länge: ",
self.laenge,"Fracht: ",self.fracht)
```

```
class Frachtflugzeug(Flugzeug):
→def __init__(self, name, laenge, fracht):
→→Flugzeug.__init__(self,name, laenge)
→→self.fracht = fracht
→→print("Konstruktor von Frachtflugzeug")

→def drucken(self):
→→print("Frachtflugzeug",self.name,"Länge: ",
self.laenge,"Fracht: ",self.fracht)
```

```
class Vogel:
→def __init__(self, name, gewicht):
→→self.name = name
→→self.gewicht= gewicht
→→print("Konstruktor von Vogel")

→def drucken(self):
→→print("Vogel",self.name,"Gewicht: ",
self.gewicht)


class Singvogel(Vogel):
→def __init__(self, name, gewicht, melodie):
→→Vogel.__init__(self, name, gewicht)
→→self.melodie = melodie
→→print("Konstruktor von Singvogel")

→def drucken(self):
→→print("Singvogel",self.name,"Gewicht: ",
self.gewicht,"Melodie:",self.melodie)



f1 = Flugzeug('Airbus 300 M',100.2)
f2 = Frachtflugzeug('Airbus 300 X',100.2,345)

v1 = Vogel('Eule',134)
v2 = Singvogel('Drossel',89, 'gut')

f1.drucken()
f2.drucken()
v1.drucken()
v2.drucken()
```

```
class Vogel(IDrucken):
→def __init__(self, name, gewicht):
→→self.name = name
→→self.gewicht= gewicht
→→print("Konstruktor von Vogel")

→def drucken(self):
→→print("Vogel",self.name,"Gewicht: ",
self.gewicht)


class Singvogel(Vogel):
→def __init__(self, name, gewicht, melodie):
→→Vogel.__init__(self, name, gewicht)
→→self.melodie = melodie
→→print("Konstruktor von Singvogel")

→def drucken(self):
→→print("Singvogel",self.name,"Gewicht: ",
self.gewicht,"Melodie:",self.melodie)



f1 = Flugzeug('Airbus 300 M',100.2)
f2 = Frachtflugzeug('Airbus 300 X',100.2,34505)

v1 = Vogel('Eule',134)
v2 = Singvogel('Drossel',89, 'gut')

liste = []
liste.append(f1)
liste.append(f2)
liste.append(v1)
liste.append(v2)
for item in liste:
→item.drucken()
```

### 6.1.3 OOP-Beispiel:

```
class Konto:

  def buchung(self, zielkontonr, betrag):
    pass              # entspricht NOP

  def einzahlung(self, betrag):
    pass              # entspricht NOP

  def auszahlung(self, betrag):
    pass              # entspricht NOP

  def kontostand(self):
    pass              # entspricht NOP
```

### 6.1.4 Beispiel mit Setter- und Getter-Methoden:

```
class Konto:

  def __init__(self, inhaber, kontonr, kontostand):
    self._inhaber  = inhaber
    self._kontonr = kontonr
    self._kontostand = kontostand

  def getInhaber(self):
    return self._inhaber

  def setInhaber(sefl, inhaber):
    self._inhaber  = inhaber

  def getKontonr(self):
    return self._kontonr

  def setKontonr(self, kontonr):
    if kontonr > 0:
      self._kontonr = kontonr
    else:
      Exception

  def getKontostand(self):
    return self._kontostand

  def setKontostand(self, betrag):
    if betrag > 0:
      self._kontostand = betrag
    else:
      Exception


  def buchung(self, zielkontonr, betrag):
    if betrag<0 or betrag> self._kontostand:
      print("Transfer nicht möglich")
      # Exception
```

```
            return false
        else:
            self._kontostand -= betrag
            # insert Buchung ins DB-System
            return true

    def einzahlung(self, betrag):
        if betrag<0:
            print("Transfer nicht möglich")
            return false
        else:
            self._kontostand += betrag
            # insert Buchung ins DB-System
            return true

    def auszahlung(self, betrag):
        if betrag<0 or betrag>self._kontostand:
            print("Transfer nicht möglich")
            return false
        else:
            self._kontostand += betrag
            # insert Buchung ins DB-System
            return true

    def kontostand(self):
        print("Konto von {0}".format(self._inhaber))
        print("Aktueller Kontostand von {0:.2f}" Euro".format(
            self._kontostand
```

### 6.1.5 Beispiel mit Property-Methoden:

```
class Konto:

    def __init__(self, inhaber, kontonr, kontostand):
        self.inhaber  = inhaber
        self.kontonr = kontonr
        self.kontostand = kontostand

    def getInhaber(self):
        return self.inhaber

    def setInhaber(sefl, inhaber):
        self.inhaber  = inhaber

    Inhaber = property(getInhaber, setInhaber)

    def getKontonr(self):
        return self.kontonr

    def setKontonr(self, kontonr):
        if kontonr>0:
            self.kontonr = kontonr
        else:
            Exception

    Kontonr = property(getKontonr, setKontonr)
```

```
def getKontostand(self):
   return self.kontostand

def setKontostand(self, betrag):
   if betrag>0:
      self.kontostand = betrag
   else:
      raise Exception("Betrag ist negativ")

Kontostand = property(getKontostand, setKontostand)


def buchung(self, zielkontonr, betrag):
   if betrag<0 or betrag> self._kontostand:
      print("Transfer nicht möglich")
      # Exception
      return false
   else:
      self.kontostand -= betrag
      # insert Buchung ins DB-System
      return true

def einzahlung(self, betrag):
   if betrag<0:
      print("Transfer nicht möglich")
      return false
   else:
      self.kontostand += betrag
      # insert Buchung ins DB-System
      return true

def auszahlung(self, betrag):
   if betrag<0 or betrag>self.kontostand:
      print("Transfer nicht möglich")
      return false
   else:
      self.kontostand += betrag
      # insert Buchung ins DB-System
      return true

def kontostand(self):
   print("Konto von {0}".format(self.inhaber))
   print("Aktueller Kontostand von {0:.2f}" Euro".format(
      self.kontostand

def __del__(self):
   print("Hier im DEkonstructor")

def __hash__(self):
   return self.kontonr

def __eq__(self, other):
   return self.kontonr== other.kontonr
```

**Aufruf:**

```
konto = Konto("KäsePizza", 12345, 1000000)
konto.Kontostand = 1000000
print(konto.Kontostand)
```

## *6.2 Statische Methoden*

### 6.2.1 staticmethod

1. Vorgehensweise:
- Definitionen der Methode **außerhalb** der Klasse
- Verbinden mit staticmethod

**Beispiel:**

```
def  quadrat(x):
  return x*x

class Mathematik:

  quadrat = staticmethod(quadrat)
```

### 6.2.2 classmethod

2. Vorgehensweise:
- Definitionen der Methode **innerhalb** der Klasse
- Verbinden mit classmethod

**Beispiel:**

```
class Mathematik:

  def  quadrat(x):
    return x*x

  quadrat = classmethod(quadrat)
```

## *6.3 Magic-Methoden*

Die sogenannten „Magic-Methoden“ sind Spezial-Methoden, die Python für Klassen resp. Instanzen zur Verfügung stellt. Alle Methoden fangen mit zwei Unterstrichen an und hören mit zwei Unterstrichen auf.

| Methode | Erläuterung |
|---|---|
| `__init__(self,..):` | Der Kontruktor wird beim Erzeugen aufgerufen |
| `__del__(self):` | Der Dekontruktor wird beim Entfernen aufgerufen. |
| `__str__(self):` | Die toString() –Methode |
| `__bytes__(self)` | Die byte-Methode zeigt an, welche Bytes die Instanz zurückgibt. Denkbar bei Serialisierung einer Instanz in eine Klasse. |
| `__hash__(self)` | Ausgabe des Hash-Wertes |
| `__eq__(self,other)` | **`def __eq__(self, other):`**<br>return self._kontonr== other._kontonr |

## *6.4 Operator-überladen*

Folgende Operationen lassen sich überladen:

- `+` `__add__(self,other)`
- `-` `__sub__(self,other)`
- `*` `__mul__(self,other)`
- `/` `__truediv__(self,other)`
- `//` `__floordiv__(self,other)`
- `**` `__pow__(self,other)`
- `%` `__mod__(self,other)`
- `>>` `__rshift__(self,other)`
- `<<` `__lshift__(self,other)`
- `&` `__and__(self,other)`
- `|` `__or__(self,other)`
- `^` `__xor__(self,other)`
- `<` `__lt__(self,other)`
- `>` `__gt__(self,other)`
- `<=` `__le__(self,other)`
- `>=` `__ge__(self,other)`
- `!=` `__ne__(self,other)`
- `==` `__eq__(self,other)`
- `+=`
- `-=`
- `*=`
- `/=`

```
class Laenge:

  def __init__(self,zahl):
        self.zahl = zahl
```

```
    def __str__(self):
        return str(self.zahl)

    def __add__(self,other):
        return self.zahl + other.zahl

    def __sub__(self,other):
        return self.zahl - other.zahl
```

**Aufruf:**

```
len1 = Laenge(11)
len2 = Laenge(31)
len3 = len1 + len2
```

# 7 Betriebssystem

## *7.1 os*

Dieses Funktionspaket liefert Funktionen über das Betriebssystem, die Prozesse und das Dateisystem.

### 7.1.1 Prozesse

```
os.environ['HOME']              Hauptordner des Benutzers
os.getpid()                     Prozess-ID
os.system(Kommando)             Entspricht einem DOS-Befehl
os.system("mkdir v1")           Entspricht einem DOS-Befehl
os.popen("ls  *.txt")           Entspricht dem $-Befehl
os.popen("dir *.txt")           Entspricht dem $-Befehl

progfile= "C:\\Daten\\P.exe"
parameterfile = "C:\\Daten\\daten.txt"
os.spawnv(os.P_NOWAIT, progfile,[ "xxxxx",parameterfile])
Unter Windows wird der zweite Parameter ausgewählt
spawnv(e) and spawnl(e)
```

### 7.1.2 Dateisystem

| **Befehl** | **Erläuterung** |
|---|---|
| `access(path,mode)` | Prüft die Rechte der Datei/Ordner mit dem Modus<br>F_OK Existiert der Pfad<br>R_OK Leserechte<br>W_OK Schreibrechte<br>X_OK Ausführbare rechte |
| `chdir(path)` | wechselt das Verzeichnis |
| `getcwd()` | siehe pwd, get Current Working Directory |
| `getcwdb()` | Wie getcwd, Rückgabe als Byte-Array |
| `chmod(path, mode)` | siehe Unix-Befehl |
| `listdir(path)` | Liste der Dateien und Ordnern |
| `mkdir(path[, mode])` | Erzeugt ein Verzeichnis |
| `makedirs(path[, mode])` | Erzeugt auch komplette Pfade |
| `remove(path)` | Löscht eine Datei |
| `removedirs(path)` | Löscht alle Ordner, sofern diese leer sind |
| `rename(src, dest)` | Umbenennen |
| `renames(src, dest)` | Wie rename, legt aber eventuelle Verzeichnisse des Zielpfades an. Außerdem werden in src-Pfad leere Ordner gelöscht. |
| `rmdir(path)` | Löscht den leeren Ordner |

#### 7.1.2.1 walk

Mit der Funktion „walk“ kann ein kompletter Verzeichnisbaum eingelesen werden.

```
for  v in os.walk("/home/user1"):
  print(v)

for  v in os.walk("/home/user1", false):       von unten nach oben
  print(v)
```

#### 7.1.2.2 remove

Eine Datei wird mit folgendem Befehl gelöscht:

```
os.remove("C:\\test.txt")
```

Ein **leeres** Verzeichnis wird mit folgendem Befehl gelöscht:

```
os.rmdir("C:\\test")
```

Rekursives Löschen funktioniert mit diesem Befehl:

```
os.shutil.rmtree("C:\\test")
```

### *7.2 os.path*

| **Befehl** | **Erläuterung** |
|---|---|
| `abspath(pathfile)` | Gibt zu einem relativen Pfad den absoluten Pfad zurück |
| `basename(pathfile)` | Linux-Befehl. Gibt nur den Dateiname zurück, nicht den Pfad. |
| `commonprefix(list)` | Sucht den gemeinsamen „Oberordner“ aller als Liste übergebenen Verzeichnisse. Schlimmstenfalls „/“ |
| `dirname(pathfile)` | Gibt nur den Pfad zurück, nicht den Dateinamen. |
| `exists(pathfile)` | Existiert der Ordner/Datei. |
| `getatime(pathfile)` | Access-Time |
| `getmtime(pathfile)` | Modify-Time |
| `getsize(pathfile)` | Dateilänge |
| `isabs(path)` | Ist es ein absoluter oder relativer Pfad |
| `isfile(pathfile)` | -f |
| `isdir(path)` | -d |
| `islink(pathfile)` | Link-Datei |
| `join(p1[,p2][,p3])` | Verbindet die Pfade |
| `normcase(path)` | Wandelt ein Unix-Pfad in einem Windows-Pfad |
| `split(pathfile)` | Spaltet path in Pfad und Dateinamen.<br>Rückgabewert ist ein Tupel.<br>Drivepath, filename = os.path.split(dbfile1) |
| `splitdrive(path)` | Spaltet path in Laufwerk und Pfad.<br>Rückgabewert ist ein Tupel. |
| `splitext(path)` | Spaltet path in Dateinamen und Extenstion (test und .text).<br>Rückgabewert ist ein Tupel. |

## *7.3 os.sys*

Man benötigt den Befehl „`import sys`“.

| | |
|---|---|
| `argv` | Kommandozeilenparameter |
| `byteorder` | Gibt die Byteorder des Systems an. |
| `path` | die Liste der Verzeichnisse, aus denen Module eingebunden werden können. |
| `stdin` | stdin-Kanal |
| `stdout` | stdout-Kanal |
| `stderr` | stderr-Kanal |
| `version` | Version des Python-Interpreters |
| `getwindowsversion()` | Ausgabe der aktuell verwendeten Windows-Version. |

### 7.3.1 Beispiele

```
programm.py   123  456  678
  argv:   0:programm.py   1:123  2:456  3:678
```

```
# coding=utf8
import sys

print(len(sys.argv))
if len(sys.argv)>0:
  print(sys.argv[0])  # kompletter Name des Skriptes
```

**Parameter analysieren:**

- Das Modul „getopt“ analysiert sehr einfach die übergegeben Parameter.
- Man kann alle Optionen definieren und das Programm testet die Gültigkeit.
- Es gibt zwei Varianten
  - ein Parameter alleine: `-1 -l`
    - in der Liste wird nur der Parameter eingetragen
  - ein Parameter mit einem weiterem Argument: `-i test.txt`
    - in der Liste wird der Parameter mit einem Doppelpunkt eingetragen

**Beispiel:**

```
opts, args = getopt.getopt(argv,"hki:o:",["ifile=","ofile="])
```

- -h ist einfacher Parameter
- -k ist einfacher Parameter
- -i benötigt einen weiteren Parameter
- -o benötigt einen weiteren Parameter
- ifile= Long_Options
  - Parameter mit einem Gleichheitszeichen und dem weiterem Argument

Quellcode:

```
try:
  opts, args = getopt.getopt(argv,"hki:o:",["ifile=","ofile="])
```

```
except getopt.GetoptError:
   print("Error in der Parameterabfrage")
   print('test.py -i <inputfile> -o <outputfile>')
   sys.exit(2)

for opt, arg in opts:
   print('opt:',opt,'opt', len(opt))
   if opt == '-k':
      print('-k')
   elif opt == '-h':
      print('test.py -i <inputfile> -o <outputfile>')
      sys.exit()
   elif opt in ("-i", "--ifile"):
      inputfile = arg
   elif opt in ("-o", "--ofile"):
      outputfile = arg
   else:
      print("Error in der Parameterabfrage",opt)
```

## *7.4 platform*

jeweils mit dem Prefix „platform.“

| | |
|---|---|
| `machine()` | Gibt die Prozessorarchitektur des Betriebssystems aus. |
| `node()` | Ausgabe des Netzwerksname. |
| `processor()` | Hersteller des Prozessors. |
| `system()` | Name des Betriebssystems. |

# 8 Dateioperationen

Dazu benutzt man das Modul „shutil“. Mit diesem Modul kann man Dateien lesen, schreiben und zippen.

| | |
|---|---|
| `copyfile(src,dest)` | Kopiert die Datei „src“ nach „dest“. Löst eventuell die Exception IOError aus. |
| `copyfileobj( handlesrc, handledest, length)` | Kopiert von Quell-Handle eine Anzahl Bytes zum Ziel-Handle. |
| `copymode(src,dst)` | Kopiert die Zugriffrechte von src nach dest. |
| `copystat(src,dst)` | Kopiert die Zugriffrechte von src nach dest UND die Zugriffszeiten. |
| `copy(src,dest)` | Kopiert die Datei „src“ nach „dest“. „dest“ kann auch en Verzeichnis sein. Löst eventuell die Exception IOError aus. |
| `copy2(src,dest)` | Kopiert die Datei „src“ nach „dest“. „dest“ kann auch en Verzeichnis sein. Kopiert die Zugriffrechte von src nach dest. Löst eventuell die Exception IOError aus. |
| `copytree(src,dest, symlinks=False, ignore=None, copy_function=copy2, ignore_dangling_symlinks =False)` | Kopiert eine gesamte Verzeichnisstruktur |
| `ignorepatterns(*patterns)` | Erzeugt eine Funktion, die bei der copy-tree Funktion für den Parameter “ignore” übergeben warden kann, um bestimme Dateien auszuschließen. |
| `move(src,dest)` | Verschieben einer Datei oder ein Ordner. |
| `rmtree(src[,ignore_errors [,onerror])` | Löscht die gesamte Verzeichnisstruktur von src. |

## *8.1 Archiv-Funktionen*

| | |
|---|---|
| `make_archive(base_name, format[,root_dir])` | Erzeugt ein Archive, in der Dateien des Verzeichnisses “root” enthalten sind. Der Rückgabe ist der Dateiname. |
| `get_archive_formats()` | Gibt eine Liste der verfügbaren Archivformate zum Erzeugen der Archive zurück. |
| `get_unpack_formats()` | Gibt eine Liste der verfügbaren Archivformate zum Entpacken der Archive zurück. |
| `unpack_archive(filename [,extract_dir[,format])` | Entpackt das Archiv unter “filename” in das Verzeichnis “extract_dir”. |

### 8.1.1 gzip

Benutzt, um aus einer Zip-Datei zu lesen oder zu schreiben.

```
import  gzip

handle  =   gzip.open("test.gz", "wb")
handle.write("Linux ist toll")
```

```
handle.close()

handle  =   gzip.open("test.gz", "rb")
str = handle.read()
print(str)
```

### 8.1.2 Beispiele

```
shutil.make_archive("test", "zip", "daten")
```
Speichert alle Dateien des Verzeichnis “daten”.

```
shutil.make_archive("test2", "zip", "daten", "unterordner2")
```
Speichert alle Dateien des Verzeichnis “unterordner2”.

# 9 Grafische Oberflächen mit Python

## *9.1 Einführung*

Für die Verwendung von Python mit grafische Oberflächen benötigt man ein entsprechendes Toolkit, sprich Framework. Unter Python stehen mehrere zur Verfügung:

- Tkinter (Tk inter)
  - Toolkit: Tk
  - Ist standardmäßig immer mit Python installiert
- PyGObject (Gtk GIMP Toolkit)
  - Toolkit: Gtk
  - Ist die Grundlage der Linux-Oberfläche GIMP.
  - Das Toolkit ist frei verfügbar
- PyQt
  - Toolkit: Qt
  - Sehr bekannt für C++ und Qt. Es wurde von der norwegischen Firma Qt Software entwickelt. Dient als Grundlage für KDE. Für die kommerzielle Nutzung benötigt man eine Lizenz.
- PySide (Nokia)
  - Toolkit: Qt
  - Benutzt auch als Grundlage das Qt-Framework. Ist weitgehend kompatibel zu PyQt. Ist auch für die kommerzielle Nutzung kostenlos.
- wxPython
  - Toolkit: wxWidgets
  - Ist ein freies UI-Toolkit. Ist für alle wichtigen Plattformen verfügbar. Das Look & Feel ist für jedes Betriebssystem möglichst genau abgebildet. Das Prinzip von Java's SWT.

### 9.1.1 Tkinter

Tkinter verwendet eine ähnliche Strategie wie Java. Es hat drei Layout-Manager. Zum einen der Pack-Manager, der ähnlich strukturiert wie das GridBagLayout ist. Dann als zweites ein Grid-Element, welches die UI_Elemente in einer Tabellenstruktur positioniert, aber auch Colomn- und Rowspan besitzt. Der letzte Layout-Manager ist der place-manager, der vergleichbar mit der setBounds-Anweisung in Java ist. Die set- und –get-Methoden der UI-Elemente werden via Hashtable aufgerufen. Die Namensgebung der UI-Elemente ist auch etwas anders als in anderen UI-Sprachen.

### 9.1.2 UI-Elemente

Folgende UI-Elemente sind verfügbar:

- **Button**
- **Canvas**
  - Zeichen-Element
- **CheckBox**
- **Entry**
  - Eingabefeld
  - Steuerelementvariablen
    - für Strings StringVar
    - für Integerzahlen IntVar
    - für Nachkommazahlen DoubleVar
    - für Boolean-Werte BooleanVar
- **Label**
- **LabelFrame**
  - GroupBox für Radiobuttons und Checkboxen
- **ListBox**
- Menu
- Menubutton
- OptionMenu
  - Popupmenu
- **Radiobutton**
- Scrollbar
- **SpinBox**
  - JSpinner bzw. NumericUpDown
- **Text**
  - Multiline-Editor
- Tree
- Widget
  - Basisklasse

Die Events für die einzelnen UI-Elemente sind sehr vielseitig. So kann man zum Beispiel ein Event definieren, welches die Eingabe „Einmal…“ in einem Textelement auslöst.

## *9.2 Beispiel*

```
import tkinter

class MyApp(tkinter.Frame):
```

```
    def __init__(self, master=None):
      tkinter.Frame.__init__(self, master)
      self.pack()
      self.setGUI()

    def setGUI(self):
      self.nameEntry = tkinter.Entry(self)
      self.nameEntry.pack()

      self.inputui = tkinter.StringVar()
      self.inputui.set("Ihr Name...")
      self.nameEntry["textvariable"] = self.inputui

      self.bnOk = tkinter.Button(self)
      self.bnOk["text"] = "Ok"
      self.bnOk["command"] = self.quit
      self.bnOk.pack(side="right")

      self.bnRev = tkinter.Button(self)
      self.bnRev["text"] = "Umdrehen"
      self.bnRev["command"] = self.onReverse
      self.bnRev.pack(side="right")

    def onReverse(self):
      self.inputui.set( self.inputui.get()[::-1] )


  // Aufruf:

  root = tkinter.Tk()
  app = MyApp(root)
  app.mainloop()
```

Erläuterung:

```
import tkinter
```

- Import des Toolkit

```
class MyApp(tkinter.Frame):
```

- Die Klasse MyApp erbt die Grafikeigenschaft von tkinter.Frame. Das entspricht der Java-Klasse „JFrame“.

```
  def __init__(self, master=None):
    tkinter.Frame.__init__(self, master)
    self.pack()
    self.setGUI()
```

- Aufruf des Konstruktors der Basisklasse
- „pack“ ordnet die UI-Elemente

```
def setGUI(self):
    self.inputui = tkinter.Entry(self)
    self.inputui.pack()
```

- Erstellt ein Textfeld und fügt diese in das Layout

```
self.var_name = tkinter.StringVar()
self.var_name.set("Ihr Name...")
self.inputui["textvariable"] = self.var_name
```

- Erstelle ein „Property“ für das Textfeld vom Typ String

```
self.bnOk = tkinter.Button(self)
self.bnOk["text"] = "Ok"
self.bnOk["command"] = self.quit
self.bnOk.pack(side="right")
```

- Erstellt einen Schalter mit einem Text und dem Event „Close“
- Der Layout-Manager fügt das Element rechtseitig ein.

```
self.bnRev = tkinter.Button(self)
self.bnRev["text"] = "Umdrehen"
self.bnRev["command"] = self.onReverse
self.bnRev.pack(side="right")
```

- Erstellt einen Schalter mit einem Text und dem Event „onReverse“
- Der Layout-Manager fügt das Element rechtseitig ein.

```
def onReverse(self):
    self.var_name.set( self.var_name.get()[::-1] )
```

- „Event-Methode des zweiten Schalters
- [::-1] ist eine For-each-Schleife

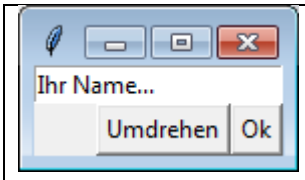
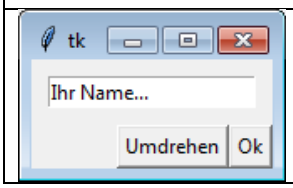
**Aufruf:**

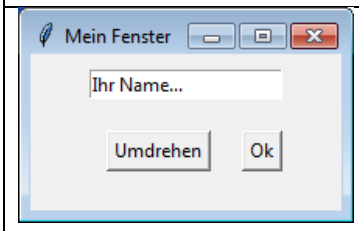
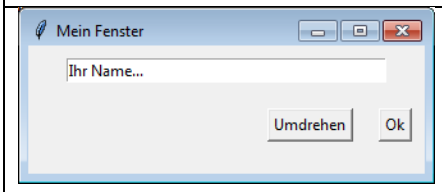
| | | |
|---|---|---|
| `root = tkinter.Tk()`<br>`app = MyApp(root)`<br>`app.mainloop()` | `app = MyApp(tkinter.Tk())`<br>`app.mainloop()` | `app = MyApp()`<br>`app.mainloop()` |

oder

```
MyApp().mainloop()
```

Beispiele:

| | |
|---|---|
|  | **bsp1.py:**<br>self.inputui.pack() |
|  | **bsp1a.py:**<br>self.inputui.pack(padx="10",pady="10") |

| | |
|---|---|
|  | **bsp1b.py:**<br>self.inputui.pack(padx="10",pady="10")<br>self.bnOk.pack(padx="10",pady="10", side="right")<br>self.bnRev.pack(padx="10",pady="10", side="right")<br><br>root = tkinter.Tk()<br>root.title("Mein Fenster")<br>root.geometry("200x100")<br>app = MyApp(root)<br>app.mainloop() |
|  | **bsp1c.py:**<br>self.inputui = tkinter.Entry(self, **width="200"**)<br>self.inputui.pack(fill="x", padx="30", pady="10")<br>self.bnOk.pack(padx="10",pady="10", side="right")<br>self.bnRev.pack(padx="10",pady="10", side="right")<br><br>root.geometry("250x100")<br>root = tkinter.Tk()<br>root.title("Mein Fenster")<br>root.geometry("200x100")<br>app = MyApp(root)<br>app.mainloop() |

## *9.3 Layout-Manager*

### 9.3.1 pack

Der Layout-Manager „pack“ arbeitet ähnlich des GridBagLayouts. Nur werden die Elemente beim Einfügen positioniert.
Parameter der pack-Methode:

| **Parameter** | **Werte** | **Erläuterung** |
|---|---|---|
| after | Widget | Das Element wird nach dem Wigdet-Element gepackt. |
| **anchor** | "n", "ne", "e", "se", "s", "sw", "w", "nw","center" | Anchor à la GridbagLayout |
| before | Widget | Das Element wird vor dem Wigdet-Element gepackt. |
| **expand** | True / False | wx bzw. wy à la GridbagLayout. Dieser Wert sollte nur beim Wert fill="y" oder fill="both" gesetzt werden.<br>Label, Entry, Checkbox, Buttons brauch kein expand. |
| **fill** | "x", "y", "both", "none" | fill-Attribut à la GridbagLayout. Wenn man das Fenster vergrößert, wird das |

| | | |
|---|---|---|
| | | UI-Element mitvergrößert. |
| in | Widget | Das Element wird in das Wigdet-Element gepackt. Meist ein LabelFrame. Diese Eigenschaft wird meistens über den Konstruktor gesetzt. |
| ipax | int | innerer Rand in Pixel |
| ipady | int | innerer Rand in Pixel |
| padx | int | äußerer Rand in Pixel |
| pady | int | äußerer Rand in Pixel |
| **side** | "left", "right", "top", "bottom" | Layout à la DockPanel von WPF |

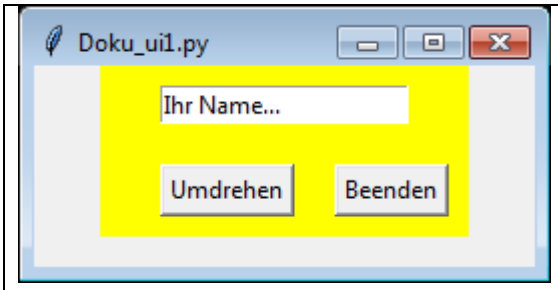
**Beispiel**:

**Definition:**

```
self.inputui = tkinter.Entry(self, width="200")
self.inputui.pack(fill="x", padx="30", pady="10")
```

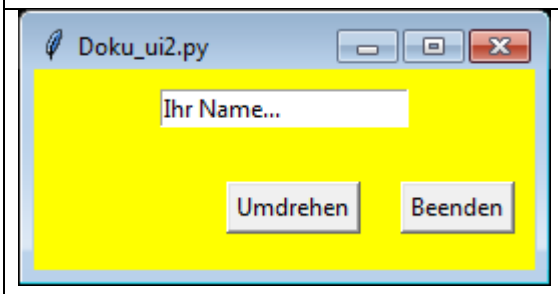
**Aufruf:**

```
root = tkinter.Tk()
root.title("Mein Fenster")
root.geometry("250x100")
app = MyApp(root)
app.mainloop()
```



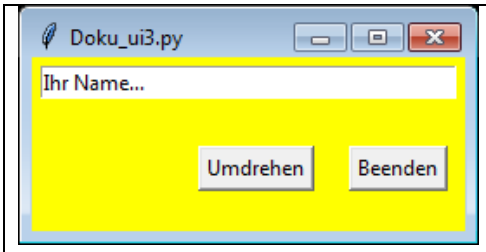

```
def __init__(self, master=None):
   tkinter.Frame.__init__(self, master)
   self.pack()
   self.config(background = "yellow")
   self.setGUI()

def setGUI(self):
   inputui = tkinter.Entry(self)
   inputui.pack(padx="30",pady="10")
```

Der Layout-Manager ist mit gelben Hintergrund dargestellt. Auch eine Vergrößerung des Fensters bewirkt nichts.
Datei: Doku_ui1.py



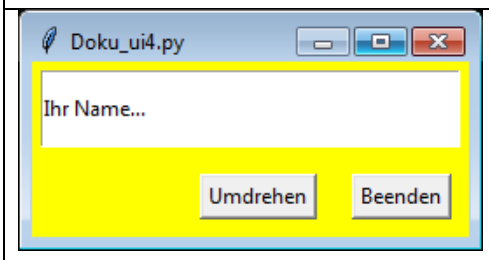

```
def __init__(self, master=None):
   tkinter.Frame.__init__(self, master)
   self.pack(expand=True, fill="both")
   self.config(background = "yellow")
   self.setGUI()

def setGUI(self):
   inputui = tkinter.Entry(self)
   inputui.pack(padx="30",pady="10")
```

Der Layout-Manager ist mit gelben Hintergrund dargestellt. Eine Vergrößerung des Fensters zeigt die Änderung. Damit können nun UI-Elemente korrekt eingefügt werden.
Datei: Doku_ui1.py



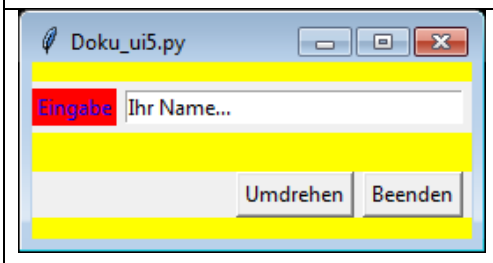

Nun wird das fill-Attribut im Entry-Element eingetragen:

```
self.inputui = tkinter.Entry(self)
self.inputui.pack(fill="x",
padx="5",pady="5")
```




Nun wird das fill- und das expand-Attribut im Entry-Element eingetragen. Dann vergrößert sich das Textfeld auch in der y-Richtung. Das ist aber nicht erwünscht. Das Problem liegt im fill-Attribut. Hier sollte man nur fill="x" eingetragen werden.

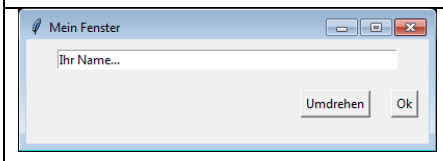
```
self.inputui = tkinter.Entry(self)
self.inputui.pack(expand=True,
fill="both", padx="5",pady="5")
```




Das Textfeld geht über das gesamte Fenster.

Sinnvoll ist das Zusammenfassen eine Labels und eines Textfeldes (Entry) mit einem Frame; entspricht einem JPanel. Damit gruppiert man die Elemente. Das folgende Beispiel verwendet zwei Frames.

```
def __init__(self, master=None):
   tkinter.Frame.__init__(self, master)
   self.pack(expand=True, fill="both")
   self.setGUI()

def setGUI(self):
   inputframe = tkinter.Frame(self)
   inputframe.background = "#FF0000"
   inputframe.pack(fill="x", side="top" )

   self.label1                              =
tkinter.Label(inputframe)
   self.label1["text"] = "Eingabe"
   self.label1.config(foreground = "blue"
)
   self.label1.config(background = "red")
   self.label1.pack(side="left")

   self.inputui                             =
tkinter.Entry(inputframe)
   self.inputui.pack(fill="x",padx="5",
             pady="5")

   self.var_name = tkinter.StringVar()
   self.var_name.set("Ihr Name...")
   self.inputui["textvariable"] =
              self.var_name

   buttonframe = tkinter.Frame(self)
   buttonframe.pack(expand=True,fill="x",
              side="top" )
```

| | |
|---|---|
| | `self.bnEsc = tkinter.Button(buttonframe)`<br>`self.bnEsc["text"] = "Beenden"`<br>`self.bnEsc["command"] = self.quit`<br>`self.bnEsc.pack(padx="5",side="right")`<br><br>`self.bnRev = tkinter.Button(buttonframe)`<br>`self.bnRev["text"] = "Umdrehen"`<br>`self.bnRev["command"] = self.onReverse`<br>`self.bnRev.pack(side="right")`<br>Datei: Doku_ui5.py |
|  | `def __init__(self, master=None):`<br>`  tkinter.Frame.__init__(self, master)`<br>`  self.pack(expand=True, fill="both")`<br>`  self.setGUI()`<br><br>`def setGUI(self):`<br>`  self.inputui = tkinter.Entry(self)`<br>`  self.inputui.pack(fill="x",`<br>`  padx="5",pady="5")`<br>Datei: Doku_ui3.py |

Multiline-Editor mit dem UI-Elemente Text:

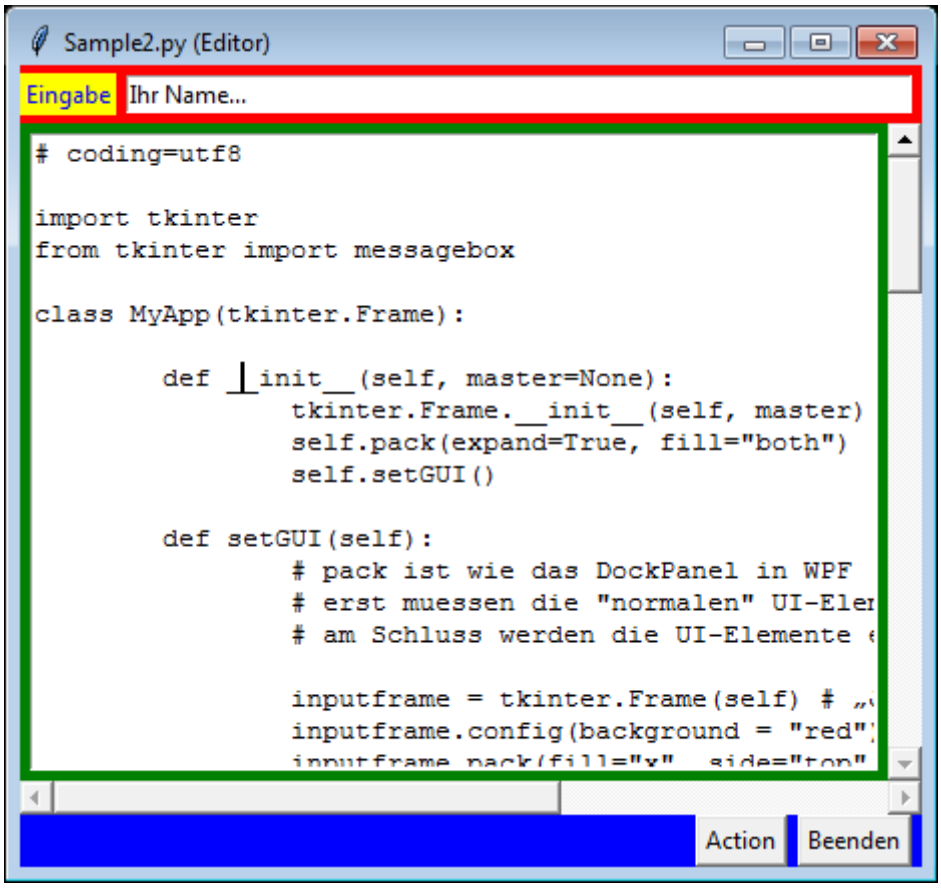



**Abbildung 1 Editor mit Scrollbars**

**Quellcode:**

```
# coding=utf8

import tkinter
from tkinter import messagebox

# http://www.tutorialspoint.com/python/python_gui_programming.htm
# http://www.tutorialspoint.com/python/tk_text.htm

class MyApp(tkinter.Frame):

   def __init__(self, master=None):
      tkinter.Frame.__init__(self, master)
      self.pack(expand=True, fill="both")  # dialog zoomt
      self.setGUI()

   def setGUI(self):
      # pack ist wie das DockPanel in WPF
      # erst muessen die "normalen" UI-Elemente eintgetragen werden
      #  am  Schluss  werden  die  UI-Elemente  eingetragen,  die  fill="both"
haben

      inputframe = tkinter.Frame(self) # „JPanel" fuer die Eingabe
      inputframe.config(background = "red") #"#FF0000"
      inputframe.pack(fill="x", side="top" ) # ohne expand, da fill="x"

      self.label1 = tkinter.Label(inputframe, fg="blue", bg="#FFFF00")
      self.label1["text"] = "Eingabe"
      self.label1.pack(side="left")

      self.inputui = tkinter.Entry(inputframe)
      self.inputui.pack(expand=True,fill="x",padx="5",pady="5")


      self.var_name = tkinter.StringVar()
      self.var_name.set("Ihr Name...")
      self.inputui["textvariable"] = self.var_name


      buttonframe = tkinter.Frame(self)  # „JPanel" fuer die Eingabe
      buttonframe.config(background = "blue") #"#FF0000"
        # ohne expand, da fill="x"
      buttonframe.pack(fill="x", side="bottom" )


      self.bnEsc = tkinter.Button(buttonframe)
      self.bnEsc["text"] = "Beenden"
      self.bnEsc["command"] = self.quit
      self.bnEsc.pack(padx="5", side="right")

      self.bnAction = tkinter.Button(buttonframe)
      self.bnAction["text"] = "Action"
      self.bnAction["command"] = self.onAction
      self.bnAction.pack(side="right")


      # nun ein Editor mit fill=both
```

```
        editorframe = tkinter.Frame(self)   # „JPanel" fuer den Editor
        editorframe.config(background = "green")
        editorframe.pack(expand=True,fill="both", side="top" )

        sbx = tkinter.Scrollbar(editorframe, orient="horizontal")
        sbx.pack(fill="x", side="bottom")

        sby = tkinter.Scrollbar(editorframe)
        sby.pack(fill="y", side="right")

        self.editor = tkinter.Text(editorframe)
        self.editor.config(wrap="none")  # wrap="word"  word char
        self.editor.pack(expand=True, fill="both", padx="5",pady="5")

        self.editor["xscrollcommand"] = sbx.set
        sbx["command"] = self.editor.xview
        self.editor["yscrollcommand"] = sby.set
        sby["command"] = self.editor.yview


    def onAction(self):
        str = self.var_name.get()
        messagebox.showinfo( "Hello Python", str)
        self.editor.insert("end",str+"\r\n")



root = tkinter.Tk()
root.title("Sample2.py (Editor)")
root.geometry("450x400")
app = MyApp(root)
app.mainloop()
```

### 9.3.2 grid

Der pack-Manager hat bei einfachen Dialogen einige Vorteile. Bei komplexeren Dialogen ist der Grid-Manager besser. Man sollte aber nie die Manager mischen. Den Grid-Manager startet man mit der Methode „grid“ und den Parametern row und column. Ähnlich wie der GridbagLayout-Manager von Java, kann man hier auch die Elemente in den Ecken platzieren. Dazu verwendet man den Sticky-Parameter mit den Werten N, S, E und W. Man kann aber pack und grid bei einem Dialog zusammen verwenden. Nur nicht zusammen bei einem Widget.

Parameter der pack-Methode:

| Parameter | Werte | Erläuterung |
|---|---|---|
| row | int | Nummer der Zeile |
| column | int | Nummer der Spalte |
| columnspan | int | Wieviele Spalten benötigt das Elemente |
| rowspan | int | Wieviele Zeilen benötigt das Elemente |
| sticky | "n", "e", "s", "w", "nw", "ne", "sw", "se" | Anchor à la GridbagLayout. Wenn die Zelle größer als das Element ist, kann man das Element ohne Zoom platzieren, |
| ipax | int | innerer Rand in Pixel |
| ipady | int | innerer Rand in Pixel |
| pdax | int | äußerer Rand in Pixel |
| pady | int | äußerer Rand in Pixel |

Beispiel:

```
self.inputui = tkinter.Entry(self)
self.inputui.grid(row=0, padx="30",pady="10")

self.bnOk = tkinter.Button(self)
self.bnOk["text"] = "Ok"
self.bnOk.grid(row=1, column=1, padx="10",pady="10")

self.bnRev = tkinter.Button(self)
self.bnRev["text"] = "Umdrehen"
self.bnRev.grid(row=1, column=0, padx="10",pady="10")
```

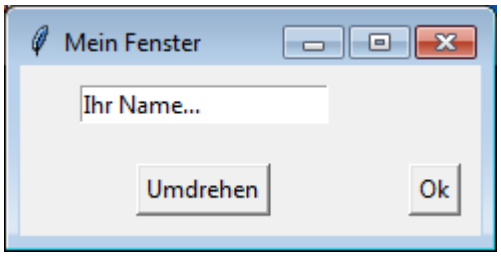



**Abbildung 2 Dialog mit einem grid-Layout (Layout_grid_bsp1.py)**

Man sieht deutlich die beiden Zeilen und Spalten. Wünschenswert wäre nun ein fill über zwei Spalten.

```
self.inputui.grid(row=0, columnspan=2, padx="10",pady="10")
```

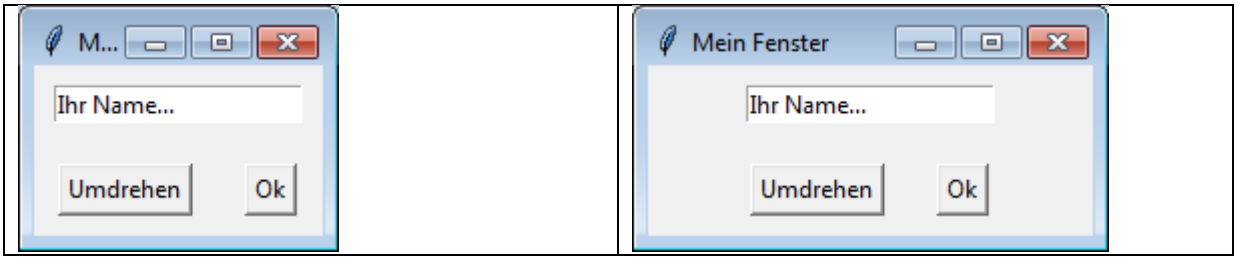



**Abbildung 3 Dialog mit einem grid-Layout und ColumnSpan (Layout_grid_bsp2.py)**

Bei einer Vergrößerung ändern sich leider nicht die Elemente.

## *9.4 Widget-Events*

Die Events sind ähnlich wie in anderen Framework. Man "ver**bindet**" für ein Widget-Element ein Event mit einer Funktion. Interessant ist die Kombination Mausklick mit einem Tastendruck. In Java muss man diese Unterscheidung im ActionListener abfragen.

### 9.4.1 Methode bind

Aufruf:

- `bind( event, function)`
  - event beschreibt das Ereignis, welches aus maximal zwei Ereignissen bestehen kann.
  - function ist ein Delegate, ein Zeiger auf eine Funktion. Das heißt, hier werden keine Klammern am Ende gesetzt.

Tabelle der Modifier:

| **Attribut** | **Erläuterung** |
|---|---|
| Alt | Die Alt-Taste muss gedrückt sein |
| Control / Strg | Die Control-Taste muss gedrückt sein |
| Lock | Die Lock-Taste muss gedrückt sein |
| Shift | Die Shift-Taste muss gedrückt sein |
| ButtonX<br><Button-1><br><Button-2><br><Button-3> | Die X-Maustaste muss gedrückt sein:<br>1: linke Maustaste<br>2: mittlere Maustaste<br>3: rechte Maustaste |
| Double | Es wird der Doppelklick abgefragt. |
| Triple | Das Event muss **dreimal** auftreten. |

Tabelle der Typs

| **Attribut** | **Typ** | **Erläuterung** |
|---|---|---|
| KeyPress | 2 | Eine Taste wurde gedrückt. Folgende festdefinierte Tasten gibt es:<br>`ALT_L, ALT_R, BackSpace, Cancel, Caps_Lock, Control_L, Control_R, Delete, End, Escape, F1, F2, ... F12, Home, Insert, Left, Up, Right, Down, Next (PagDn), Num_Lock, Pause, Print, Prior (PgUp), Return, Scroll, Lock, Shift_L, Shift_R, Tab` |
| KeyRelease | 3 | Die Taste wurde losgelassen. Tastencode wie in KeyPressed. |
| ButtonPress | 4 | Eine Maustaste wurde gedrückt. Die Maustasten sind wie oben codiert. |
| ButtonRelease | 5 | Eine Maustaste wurde losgelassen. Die Maustasten sind wie oben codiert. |
| Motion | 6 | Mouse Hover Event |

| | | |
|---|---|---|
| Enter | 7 | Mouse Enter Event in einem Widget. |
| Leave | 8 | Mouse Exited Event in einem Widget. |
| FocusIn | 9 | Das Widget erhält den Focus. |
| FocusOut | 10 | Das Widget hat den Focus verloren. |
| Expose | 12 | onPaint Event. Es war verdeckt, jetzt ist es wieder sichtbar. |
| Destroy | 17 | Das Widget wurde zerstört. |
| Configure | 22 | Das Widget hat seine Größe oder Position verändert. |
| MouseWheel | 38 | Das Mausrad wurde bewegt. |

**Beispiel:**

```
def onKeyClick(self,event):
        print("onKeyClick ")

def onMouseClick(self,event):
        print("onMouseClick ")

self.spinbox.bind('<KeyPress>', self.onKeyClick)
self.spinbox.bind('<Button-1>', self.onMouseClick)
```

**Attribute des Parameters „event“:**

Beispiel:

```
   def eventMouseClick(self, event):
   def eventMouseWheel(self, event):
```

| **Attribut von event** | **Erläuterung** |
|---|---|
| char | Enthält das Zeichen als String (KeyPress, KeyRelease). |
| delta | Wie weit wurde das Wheelrad gedreht (positiv oder negativ). |
| focus | Gibt an, ob das Widget den Focus hat (Enter, Leave). |
| height | Höhe des Widgets (Configure, Expose). |
| keycode | Enthölt den Tastecode (KeyPress, KeyRelease). |
| keysym | Enthält den symbolischen Name (KeyPress, KeyRelease). |
| time | Enthält den Zeitstempel in Millisekunden). |
| type | Enthält den Typecode. Gilt für alle Events. |
| widget | Ist der „Sender“. |
| width | Enthält die **neue** Breite des Widgets. |
| x | Enthält die X-Koordinate des Mauszeigers in Pixel. Relativ zum Widget. |
| x_root | Enthält die X-Koordinate des Mauszeigers in Pixel. Relativ zum Bildschirm. |
| y | Enthält die Y-Koordinate des Mauszeigers in Pixel. Relativ zum Widget. |
| y_root | Enthält die Y-Koordinate des Mauszeigers in Pixel. Relativ zum Bildschirm. |

**Beispiel:**

```
import tkinter

class MyApp(tkinter.Frame):

  def __init__(self, master=None):
    tkinter.Frame.__init__(self, master)
    self.pack()
    self.createWidgets()
    self.createBindings()

  def createWidgets(self):
    self.label1 = tkinter.Label(self)
    self.label1.pack()
    self.label1["text"] = "Bitte in das Textfeld klicken oder am Rad
drehen"

    self.entry1 = tkinter.Entry(self)
    self.entry1.pack(fill="x")

    self.label2 = tkinter.Label(self)
    self.label2.pack()
    self.label2["text"] = "Bitte in das Textfeld klicken oder am Rad
drehen"

    self.entry2 = tkinter.Entry(self)
    self.entry2.pack()

    self.label3 = tkinter.Label(self)
    self.label3.pack()
    self.label3["text"] = "Meldungen"

    self.label4 = tkinter.Label(self)
    self.label4.pack()
    self.label4["text"] = "Meldungen"

    self.ok = tkinter.Button(self)
    self.ok.pack()
    self.ok["text"] = "Beenden"
    self.ok["command"] = self.quit

  def createBindings(self):
    self.entry1.bind("42", self.event42)
    self.entry1.bind("<ButtonPress-1>", self.eventMouseClick)
    self.entry2.bind("<ButtonPress-1>", self.eventMouseClick)
    self.entry1.bind("<MouseWheel>", self.eventMouseWheel)
    self.entry1.bind("<Enter>", self.eventEnterE1)
    self.entry1.bind("<Leave>", self.eventLeaveE1)
    self.entry2.bind("<Enter>", self.eventEnterE2)
    self.entry2.bind("<Leave>", self.eventLeaveE2)

  def event42(self, event):
    self.label1["text"] = "Sie kennen das geheime Passwort!"

  def eventMouseClick(self, event):
    if event.widget == self.entry1:
```

```
            self.label1["text"]     =     "E1:     Mausklick     an     Position
({},{})".format(event.x, event.y)
        else:
            self.label1["text"]     =     "E2:     Mausklick     an     Position
({},{})".format(event.x, event.y)

    def eventMouseWheel(self, event):
        self.label["text"] = "Mausrad ({})".format(event.delta)

    def eventEnterE1(self, event):
        self.label3["text"] = "Entry 1 hat Foucs"

    def eventLeaveE1(self, event):
        self.label3["text"] = "Entry 1 hat keinen Foucs"

    def eventEnterE2(self, event):
        self.label4["text"] = "Entry 1 hat Foucs"

    def eventLeaveE2(self, event):
        self.label4["text"] = "Entry 1 hat keinen Foucs"

root = tkinter.Tk()
root.title("Mouse Events 2")
root.geometry("300x120")
app = MyApp(root)
app.mainloop()
```

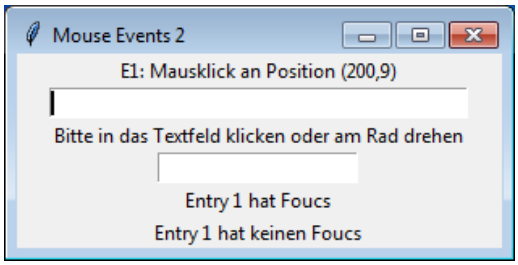



**Abbildung 4 Dialog mit verschiedenen Events**

## *9.5 Widget-Elemente*

Die Klasse Widget ist die Basisklasse alle UI-Elemente in Tkinter.

**Basis-Eigenschaften:**

| Interface | Bedeutung |
|---|---|
| winfo_children() | Gibt die Liste der enthaltenen Sub-Elemente zurück. |
| winfo_class() | Bezeichnung des Widget. Wichtig in einer Liste mit Elementen. |
| winfo_geometry() | Gibt die Position und Abmessungen in Form eines Strings zurück. Format: B×H+X+Y. |
| winfo_height() | Gibt die Höhe des Widget in Pixel zurück. |
| winfo_pointerx() | Gibt Left des Widget in Pixel zurück. |
| winfo_pointerxy() | Gibt Left/Top als Tupel des Widget in Pixel zurück. |

| | |
|---|---|
| winfo_pointery() | Gibt Top des Widget in Pixel zurück. |
| winfo_reqheight() | Gibt die angeforderte Höhe des Widget in Pixel zurück. |
| winfo_reqwidth() | Gibt die angeforderte Breite des Widget in Pixel zurück. |
| winfo_rootx() | Gibt Left des Widget in Pixel in Bezug zum Bildschirm zurück. |
| winfo_rooty() | Gibt Top des Widget in Pixel in Bezug zum Bildschirm zurück. |
| winfo_screenheight() | Gibt die Höhe des Bildschirms in Pixel zurück |
| winfo_screenwidth() | Gibt die Breite des Bildschirms in Pixel zurück |
| winfo_x() | Gibt Left des Widget in Pixel in Bezug zum Parent zurück. |
| winfo_y() | Gibt Top des Widget in Pixel in Bezug zum Parent zurück. |

**Weitere wichtige Methoden:**

| **Interface** | **Bedeutung** |
|---|---|
| bind(event, func[,add]) | Erzeugt eine Verbindung des Widget zum ActionEventListener. |
| bind_all(event, func[,add]) | Erzeugt eine Verbindung **aller** Widgets zum ActionEventListener. |
| unbind(event[,func]) | Löst das Event von Widget. |
| unbind_all(event) | Löst für alle Events und aller Widget die Verbidnung. |
| cget(key) | Gibt den Wert der Option *key* vom Widget zurück. |
| configure([cnf][,**kw]) | Setz eine oder mehrere Optionen. Meist geschieht das in Form einer Hashtable. |
| destroy() | Entfernt das Widget und alle Sub-Elemente. Benötigt bei dynamischen Dialogen. |
| focus_set() | Setzt den Fokus. |
| focus | Setzt den Fokus. |
| info() | Gibt ein Dictionary für den Packer zurück. Ähnlich wie in Swift. |
| keys() | Gibt eine Liste aller für das Wdget gültigen Optionen zurück. |
| mainloop() | Startet die hauptschleife, wie in Win 3.1. |
| pack([cnf][, **kw]) | Setzt Optionen |
| quit() | Entfernt alle Widgets und beendet das Programm. |
| slaves() | Gibt eine Liste aller im Widget liegenden Elementen zurück (Rekursiv). |

## *9.6 Widgets*

### 9.6.1 Button

Erzeugen: `bn = tkinter.Button(self)`

Beschriftung: `self.bn["text"] = "Ok"`

Event: `self.bn["command"] = self.onClickTheButton`

Layout-Manager: `self.bn.pack` oder `self.bn.grid()`

ActiveBackground: `self.bn.config(activebackground= "yellow")`

ActiveForeground:
```
self.bn.config(activeforeground = "green")
```

Background:
```
self.bn. config(background = "green")
self.bn. config(background = "#FF0000")
```

Foreground:
```
self.bn.config(foreground="red")
```

Borderwidth:
```
self.bn.config(borderwidth=="2")  #pixel
```

Borderwidth:
```
self.bn.config(bd=pixel="2")
```

Height:
```
self.bn.config(height="2")          # Textzeilen
```

Justify:
```
self.bn.config(justify("left")
self.bn.config(justify("right")
self.bn.config(justify("justify")
```

overrelief:
```
self.bn.config(overrelief="raised")    Mouse Hover
sunken
flat
ridge
solid
groove
```

relief:
```
self.bn.config(relief="raised")
sunken
flat
ridge
solid
groove
```

state:
```
self.bn.config(state="normal")
active
enabled
disabled
```

Takefokus:
```
self.bn.config(takefocus=bool)
Kann Fokus erhalten?
```

Text:
```
self.bn["text"] = Caption
```

Textvariable:
```
self.bn["textvariable"] = Caption
```
Property für den Schalter

### 9.6.2 Checkbutton

**Checkbutton mit Property:**

```
   chk1 = tkinter.Checkbutton(self)
   chk1["text"] = "Montag, 08:00 Uhr"

   chkProperty = tkinter.BooleanVar()
   chkProperty.set(True)
   chk1["variable"] = chkProperty
```

Beispieldialog:

```
self.group = tkinter.LabelFrame(self, text="Noten")
self.group.pack(fill="both", expand="yes")
self.chk1= tkinter.Checkbutton(self)
self.chk1["text"] = "Eins"
self.chk1.pack()

self.chk2= tkinter.Checkbutton(self)
self.chk2["text"] = "Zwei"
self.chk2.pack()

self.chk3= tkinter.Checkbutton(self)
self.chk3["text"] = "Drei"
self.chk3.pack()

self.chkProperty1 = tkinter.BooleanVar()
self.chkProperty1.set(True)
self.chk1["variable"] = self.chkProperty1

self.chkProperty2 = tkinter.BooleanVar()
self.chkProperty2.set(False)
self.chk2["variable"] = self.chkProperty2

self.chkProperty3 = tkinter.BooleanVar()
self.chkProperty3.set(False)
self.chk3["variable"] = self.chkProperty3

print(str(self.chkProperty1.get()) )
print(str(self.chkProperty2.get()) )
print(str(self.chkProperty3.get()) )
# output =
   "Wert der CheckBox ist {}".format(self.chkProperty1.get())
print(output)
```

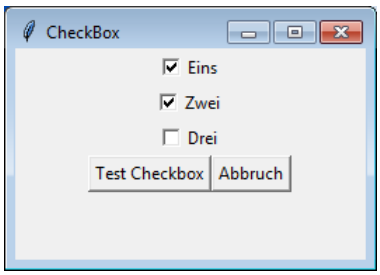



**Abbildung 5 Beispieldialog mit drei Checkbutton und der Abfrage**

### 9.6.3 Radiobutton

Es gibt eine Groupbox à la Delphi oder die Klasse Group aus Java. Alle RadioButton in einem Widget gehören derselben Gruppe an.

**Beispiel:**

```
rb1= tkinter.Radiobutton(self, text="Eins", value=" Eins ")
rb2= tkinter.Radiobutton(self, text="Zwei", value="Zwei")
rbProperty = tkinter.StringVar()
rbProperty.set("Zwei")
rb1["variable"] = rbProperty
rb2["variable"] = rbProperty
```

Mit GroupBox
self.group = tkinter.LabelFrame(self, text="Noten")

```
self.group.pack(fill="both", expand="yes")
self.rb1= tkinter.Radiobutton(self.group, text="Eins", value="one")
self.rb1.pack()
self.rb2= tkinter.Radiobutton(self.group, text="Zwei", value="two")
self.rb2.pack()
self.rb3= tkinter.Radiobutton(self.group, text="Drei", value="three")
self.rb3.pack()

self.rbProperty = tkinter.StringVar()
self.rbProperty.set("Drei")
self.rb1["variable"] = self.rbProperty
self.rb2["variable"] = self.rbProperty
self.rb3["variable"] = self.rbProperty
```

Abfrage:

```
    self.rbProperty.get()     # one,two, three
```

### 9.6.4 Entry (TextField)

Property eines Entry-Elements:

```
self.inputui = tkinter.Entry(self)
self.inputui.pack()

self.propertyInputui = tkinter.StringVar()
self.propertyInputui.set("Ihr Name...")
self.inputui["textvariable"] = self.propertyInputui

self.propertyInputui = "abc"
print(self.propertyInputui)
```

**Eingabezeichen für Paßworteingabe:**

```
self.inputui = tkinter.Entry(self)
self.inputui["show"] = "*"
```

Löschen des Inhalts:

```
self.inputui.delete(0,END)
```

Ausgabe des Inhalts:

```
print("Inhalt: %s" % (self.inputui.get()) )
print( self.inputui.get() )
```

### 9.6.5 Label

```
self.labelui = tkinter.Label(self)
self.labelui["text"] = "Vorname"
self.labelui.pack()     oder grid(…)
```

### 9.6.6 LabelFrame

Entspricht dem GroupBox in verschiedenen Frameworks.

**Beispiel:**

```
self.group = tkinter.LabelFrame(self, text="Noten")
self.group.pack(fill="both", expand="yes")
self.rb1= tkinter.Radiobutton(self.group, text="Eins", value="Eins")
self.rb1.pack()
self.rb2= tkinter.Radiobutton(self.group, text="Zwei", value="Zwei")
self.rb2.pack()
self.rb3= tkinter.Radiobutton(self.group, text="Drei", value="Drei")
self.rb3.pack()

self.rbProperty = tkinter.StringVar()
self.rbProperty.set("Drei")
self.rb1["variable"] = self.rbProperty
self.rb2["variable"] = self.rbProperty
self.rb3["variable"] = self.rbProperty
```

Hinweise:

- Bitte beachten Sie, dass die Radiobutton auch die Methode “pack” aufrufen müssen.
- Mit dem Befehl “`self.rbProperty.set("Drei")`” kann man ein RadioButton setzen bzw. abfragen.
    - `print(self.rbProperty.get())`

Beispieldialog:

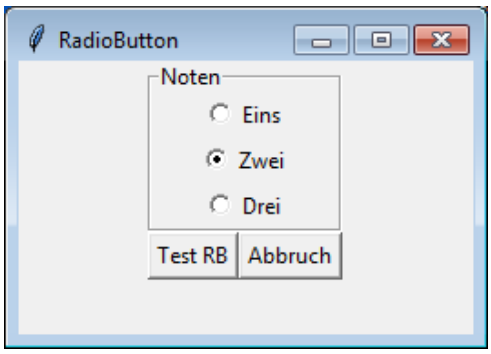



**Abbildung 6 Dialog mit drei RadioButton und einer Abfrage**

### 9.6.7 Listbox

```
listbox = tkinter.Listbox(self)
listbox.pack(fill="both", expand=True)

for i in range(20):
  listbox.insert("end", str(i))
for i in range(10):
  listbox.insert(0, str(i))
```

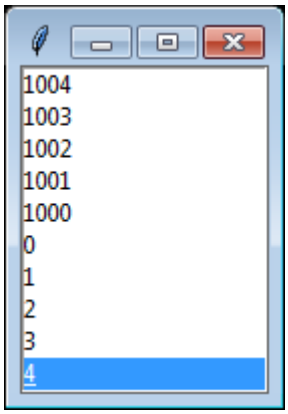



**Abbildung 7 Listbox ohne Scrollbar**

**Weitere wichtige Methoden:**

| Methode | Bedeutung |
|---|---|
| curselection() | getSelectedItems |
| delete(first[,last]) | Löschen von first bis last. |
| get(first[,last]) | Rückgabe der Werte der Listbox von first bis last. |
| insert(index, *elements) | Einfügen eines oder mehrerer Elemente.<br>• listbox.insert(0, "value")<br>• listbox.insert("end", "value") |
| itemcget(index,option) | Gibt den Wert der Option “option” des Eintrags mit dem Index “index” zurück. |

| | |
|---|---|
| itemconfig(index,**option) | Setzt die Optionen |
| selection_clear(first[,last]) | Löscht die Selection in der Liste die Einträge von first bis last. |
| selection_includes(index) | Ist index gesetzt?<br>Rückgabe: True oder False |
| selection_set(first[,last]) | Setzt die Selection in der Liste die Einträge von first bis last. |
| size() | Anzahl der Listenelemente. |
| selectmode | "single", "browse", "multiple", "extended".<br>Single und Browse erlauben nur einen Wert zu markieren. |
| xscrollcommand | Anbindung einer Scrollbar |
| yscrollcommand | Anbindung einer Scrollbar |

**Events SelectedChange**

```
self.listbox = tkinter.Listbox(self)
self.listbox.pack(fill="both", expand=True)

for i in range(20):
  self.listbox.insert("end", str(i))

self.listbox.bind("<<ListboxSelect>>", self.selectionChanged)
self.listbox.selection_set(0)

def selectionChanged:
  print( self.listbox.curselection())
```

**Löschen aller Einträge einer Listbox:**

```
self.listbox.delete(0,"end")
```

**Einfügen eines Eintrags in eine Listbox:**

```
self.listbox.insert("end", "Hallo")
```

### 9.6.8 Spinbox

Die Spinbox ist unter vielfätigem Name bekannt:

- Delphi: SpinEdit
- Java: JSpinner
- Winforms: NumericUpDown

Wie in Java, so kann man mit der Spinbox einen ganzzahligen Zahlenbereich auswählen; aber auch ihn wie eine Listbox benutzen.

Methoden:

| **Methode** | **Wertebereich** | **Erläuterung** |
|---|---|---|
| format | "%pad1.pad2" | Bekannte C-Format |
| from | float | Untere Grenze |
| increment | float | Increment beim Klick der kleinen Pfeile |
| to | float | Obere Grenze |
| values | tuple oder list | Feste Werte |
| xscrollcommand | Delegate | Für eine Scrollbar |

Beispiele:

```
sp = tkinter.Spinbox(self)
s["from"] = 0
s["to"] = 100
s.pack

sp = tkinter.Spinbox(self)
s["values"] = (0,1,1,2,3,5,8,13,21,34,55,89)
s["values"] = ("A","B","C")
s.pack
```

### 9.6.9 Text

Die Textkomponente entspricht eher einer RTF-Komponente. In ihr kann man formatierten und farbigen Text darstellen, aber auch Bilder platzieren. Es ist leider nicht möglich, eine Steuerelementvariable resp. Property für das Textelement einzurichten. Man kann aber den Inhalt abfragen.

Einfügen von Texten:

- "insert"
- "current"
- "end"

Parameter:

- height=2 Zwei Zeilen
- width=30 Dreißig Buchstaben Breite

Beispiel:

```
self.editor = tkinter.Text(self)
self.editor.grid(expand=True, fill="both", padx="5",pady="5")
self.editor.insert("end","\r\nonAction\r\n")
```

**Textwrapping:**

- self.editor.config(wrap="none")
- self.editor.config(wrap="word") Umbrechen beim letzten Wort
- self.editor.config(wrap="char")

**Löschen aller Einträge eines Text-Elementes:**

```
self. editor.delete(0,"end")
```

**Einfügen eines Eintrags in eine Text-Elementes:**

```
self.editor.insert("end", "Hallo\r\n")
```

Abfragen des Inhaltes:

```
input = self.myText_Box.get("1.0",END)
        1.0     1. Linie bis zum Ende
input = self.myText_Box.get("1.0",'end-1c')
        1.0     1. Linie bis zum Ende abzüglich eines Zeichens
input = self.myText_Box.get("1.0",'end-2c')
        1.0     1. Linie bis zum Ende abzüglich zweier Zeichens
```

### 9.6.10 Tree

Der Tree funktioniert genauso wie andere UI-Tree-Elemente. Beim Einfügen wird ein Parent-Konto benötigt, und als Rückgabewert erhält man einen Knoten.

**Besonderheit:**

- Man kann Spaltenüberschriften (heading) oben einfügen.

**Beispiel ohne Spalten:**

```
self.tree = tkinter.ttk.Treeview(frame1)
self.tree.pack(fill="both",expand=True)
self.tree["xscrollcommand"] = xsb.set    #scrolling
xsb["command"]=self.tree.xview           #scrolling
ysb["command"]=self.tree.yview           #scrolling

# 1. Knoten anlegen mit Spalten(Values)
root_node = self.tree.insert('', '1', text='Ordner',
           values=("13-Jun-19 11:28","Verz","-") open=True)

# subKnoten anlegen
node=self.tree.insert(root_node , 'end', text='abc', open=False)
```

**Beispiel mit Spalten:**

```
self.tree = tkinter.ttk.Treeview(frame1)
self.tree.pack(fill="both",expand=True)
self.tree["xscrollcommand"] = xsb.set
xsb["command"]=self.tree.xview
ysb["command"]=self.tree.yview

self.tree["columns"]=("one","two","three")
self.tree.column("#0", width=70, minwidth=27, stretch=tkinter.NO)
self.tree.column("one", width=90, minwidth=50, stretch=tkinter.NO)
self.tree.column("two", width=60, minwidth=45)
self.tree.column("three", width=80, minwidth=50, stretch=tkinter.NO)

self.tree.heading("#0",text="Name",anchor=tkinter.W)
self.tree.heading("one", text="Date modified",anchor=tkinter.W)
self.tree.heading("two", text="Type",anchor=tkinter.W)
self.tree.heading("three", text="Size",anchor=tkinter.W)

# 1. Knoten anlegen mit Spalten(Values)
root_node = self.tree.insert('', '1', text='Ordner',
           values=("13-Jun-19 11:28","Verz","-") open=True)

# subKnoten anlegen
```

```
    node=self.tree.insert(root_node , 'end', text='abc',
              values=("13-Jun-19 11:29","jpg","12.444") open=False)
```

Kompletes Beispiel:

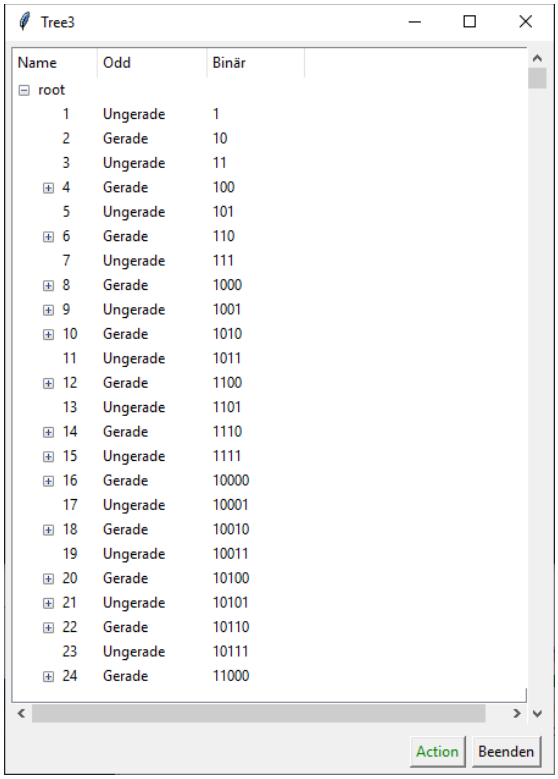



**Abbildung 8 Tree-Beispiel mit Spalten**

**Quellcode:**

```
#! /usr/bin/env python3
# coding=utf8

import os
import tkinter
from tkinter.ttk import *
from tkinter import messagebox

class MyApp(tkinter.Frame):

   def __init__(self, master):
      tkinter.Frame.__init__(self, master)
      self.pack(expand=True, fill="both") #
      self.setGUI()

   def setGUI(self):
      # pack ist wie das DockPanel in WPF
      # erst muessen die "normalen" UI-Elemente eintgetragen werden
      # am Schluss werden die UI-Elemente eingetragen, die fill="both" haben
      buttonframe = tkinter.Frame(self)
      buttonframe.pack(fill="x", side="bottom", padx="5", pady="5")

      self.bnEsc = tkinter.Button(buttonframe)
```

```
        self.bnEsc["text"] = "Beenden"
        self.bnEsc["command"] = self.quit
        self.bnEsc.pack(padx="5", side="right")

        self.bnAction = tkinter.Button(buttonframe)
        self.bnAction["text"] = "Action"
        self.bnAction["command"] = self.onAction
        self.bnAction.pack(side="right")
        self.bnAction.config(foreground= "green")
        self.bnAction.config(activebackground= "yellow")

        frame1 = tkinter.Frame(self)
        frame1.pack(expand=True, fill="both", side="top", padx="5", pady="5")

        ysb = tkinter.Scrollbar(frame1) # , command=self.tree.yview
        ysb.pack(fill="y",side="right")

        xsb = tkinter.Scrollbar(frame1, orient="horizontal")
        xsb.pack(fill="x",side="bottom")

        self.tree = tkinter.ttk.Treeview(frame1)
        self.tree.pack(fill="both",expand=True)
        self.tree["xscrollcommand"] = xsb.set
        xsb["command"]=self.tree.xview
        ysb["command"]=self.tree.yview

        self.tree["columns"]=("one","two","three")
        self.tree.column("#0", width=70, minwidth=27, stretch=tkinter.NO)
        self.tree.column("one", width=90, minwidth=45)
        self.tree.column("two", width=80, minwidth=50, stretch=tkinter.NO)

        self.tree.heading("#0",text="Name",anchor=tkinter.W)
        self.tree.heading("one", text="Odd",anchor=tkinter.W)
        self.tree.heading("two", text="Binär",anchor=tkinter.W)

        root_node = self.tree.insert('', 'end', text='root', open=True)
        self.insertTree(root_node)
        #self.tree.insert(root_node, 'end', text='Line 1', open=False)

    def insertTree(self,parent):
        n=100
        for i in range(1,n+1,1):
            if i%2==0:
                typ="Gerade"
            else:
                typ="Ungerade"
            binaer=int(bin(i)[2:])
            node=self.tree.insert(parent , 'end',values=(typ,binaer) ,
                      text=str(i), open=False)
            for j in range(2,i,1):
                if i%j==0:
                    self.tree.insert(node , 'end', text=str(j), open=False)


    def onAction(self):
        messagebox.showinfo( "Hello Python", "action")

root = tkinter.Tk()
```

```
root.title("Tree3")
root.geometry("450x600")
app = MyApp(root)
app.mainloop()
```

### 9.6.11 Menu

- Das Menü besteht wie in Java aus einem MenuBar, hier aber Menu benannt.
- Die Menüs werden über das Widget “Menu” dargestellt.
- Die Menüeinträge warden über “add_command” realisiert.
- Eingetragen warden die Menüeinträge mitttels des Befehls “add_cascade”.

| **Option** | **Beschreibung** |
|---|---|
| **add_command ( options )** | Hinzufügen eines Menüeintrags |
| **add_radiobutton( options )** | Hinzufügen eines Menüeintrags mit Radiobutton |
| **add_checkbutton( options )** | Hinzufügen eines Menüeintrags mit check button |
| **add_cascade( options )** | Einfügen eines Hauptmenüs oder Submenüs. |
| **add_separator()** | Einfügen eines Separators |
| add( type, options ) | Hinzufügen von Typen von Menüs. |
| delete( startindex [, endindex ]) | Löschen von Menüeinträge |
| entryconfig( index, options ) | Möglichkeit zur Modifizierung. |
| index(item) | Ausgabe des Index eines Menüeintrags. |
| insert_separator ( index ) | Einfügen eines Separator |
| invoke ( index ) | Aufruf der Event-Methode. Setzt die Checkbox oder Radiobutton. |
| type ( index ) | Rückgabe des Types:<br>• "cascade"<br>• "checkbutton"<br>• "command"<br>• "radiobutton"<br>• "separator"<br>• "tearoff". |

Beispiel:

```
self.menuBar = tkinter.Menu(self)
master.config(menu=self.menuBar)
self.menuFile = tkinter.Menu(self.menuBar, tearoff=False)
self.menuFile.add_command(label="Öffnen", command=self.openFile)
self.menuFile.add_command(label="Speichern", command=self.saveFile)
self.menuFile.add_command(label="Speichern unter", command=self.saveasFile)
self.menuFile.add_separator()

self.menuSubFile = tkinter.Menu(self.menuBar, tearoff=False)
self.menuSubFile.add_command(label="Sub1")
self.menuSubFile.add_command(label="Sub2")
```

```
self.menuSubFile.add_command(label="Sub3")
self.menuFile.add_cascade(label="Sub1-3", menu=self.menuSubFile)

self.menuFile.add_command(label="Beenden", command=self.quit)
self.menuBar.add_cascade(label="Datei", menu=self.menuFile)

self.menuEdit = tkinter.Menu(self.menuBar, tearoff=False)
self.menuEdit.add_command(label="Einfügen", command=self.insert)
self.menuEdit.add_command(label="Kopieren", command=self.copy)
self.menuBar.add_cascade(label="Edit", menu=self.menuEdit)
```

**Erzeugen der MenuBar:**
```
self.menuBar = tkinter.Menu(self)
```

**Einfügen in das Dialog:**
```
master.config(menu=self.menuBar)
```

**Erzeugen eines Hauptmenüs:**
```
self.menuFile = tkinter.Menu(self.menuBar, tearoff=False)
```
Mit tearoff kann man das Menü vom fenster lösen.

**Erzeugen eines Menüeintrags mit onClick-Event:**
```
self.menuFile.add_command(label="Öffnen", command=self.openFile)
```

**Einfügen in den Menübar**
```
self.menuBar.add_cascade(label="Datei", menu=self.menuFile)
```

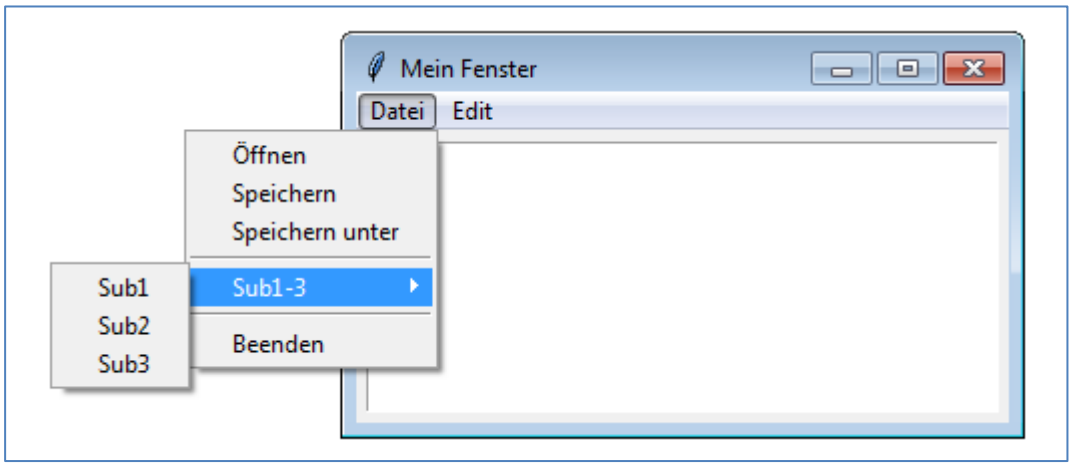



**Abbildung 9 Menü mit Submenüs unter TKinter**

**Kompletter Quellcode:**

```
import tkinter

class MyApp(tkinter.Frame):

   def __init__(self, master=None):
      tkinter.Frame.__init__(self, master)
      self.menuBar = tkinter.Menu(self)
      master.config(menu=self.menuBar)
      self.setGUI()
```

```
        self.pack()

    def setGUI(self):
        self.menuFile = tkinter.Menu(self.menuBar, tearoff=False)
        self.menuFile.add_command(label="Öffnen", command=self.openFile)
        self.menuFile.add_command(label="Speichern", command=self.saveFile)
        self.menuFile.add_command(label="Speichern                          unter",
command=self.saveasFile)
        self.menuFile.add_separator()

        self.menuSubFile = tkinter.Menu(self.menuBar, tearoff=False)
        self.menuSubFile.add_command(label="Sub1")
        self.menuSubFile.add_command(label="Sub2")
        self.menuSubFile.add_command(label="Sub3")
        self.menuFile.add_cascade(label="Sub1-3", menu=self.menuSubFile)

        self.menuFile.add_separator()
        self.menuFile.add_command(label="Beenden", command=self.quit)
        self.menuBar.add_cascade(label="Datei", menu=self.menuFile)

        self.menuEdit = tkinter.Menu(self.menuBar, tearoff=False)
        self.menuEdit.add_command(label="Einfügen", command=self.openFile)
        self.menuEdit.add_command(label="Kopieren", command=self.saveFile)
        self.menuBar.add_cascade(label="Edit", menu=self.menuEdit)

        self.editor = tkinter.Text(self)
        self.editor.pack(padx="5",pady="5")

    def openFile(self):
        self.editor.insert("end","openFile\r\n")

    def saveFile(self):
        self.editor.insert("end","saveFile\r\n")

    def saveasFile(self):
        self.editor.insert("end","saveasFile\r\n")


root = tkinter.Tk()
root.title("Mein Fenster")
root.geometry("250x300")
app = MyApp(root)
app.mainloop()
```

Um Checkboxen in einem Menü zu integrieren, benutzt man folgenden Code:

```
self.menuFont = tkinter.Menu(self.menuBar, tearoff=False)
self.menuFont.add_checkbutton(label="Fett", command=self.bold)
self.menuFont.add_checkbutton(label="Kursiv", command=self.italic)
self.menuBar.add_cascade(label="Schrift", menu=self.menuFont)
```

Um Radiobuttons in ein Menü zu integrieren, benutzt man folgenden Code:

```
    self.menuFont.add_separator()
    self.menuFont.add_radiobutton(label="Arial", command=self.arial)
    self.menuFont.add_radiobutton(label="Helvitica", command=self.helvitica)
    self.menuFont.add_radiobutton(label="Verdana", command=self.verdana)
```

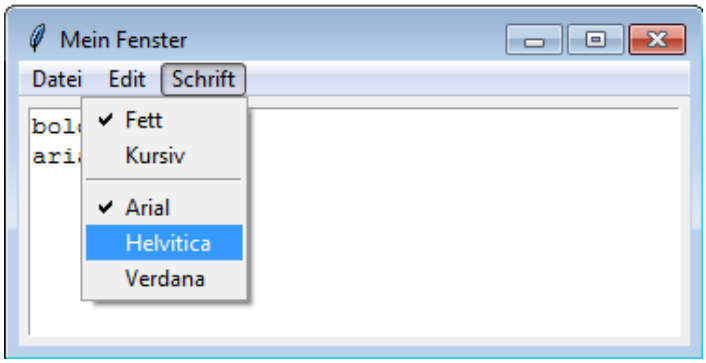



**Abbildung 10 Menü mit zwei Checkbox und drei RadioButtons**

### 9.6.12 Scrollbar

```
sb = tkinter.Scrollbar(self)
sb.pack(fill="y", side="right")

listbox = tkinter.Listbox(self)
listbox.pack(fill="both", expand=True)
listbox["yscollcrommand"] = sb.set
sb["command"] = listbox.yview

for i in range(20):
   listbox.insert("end", str(i))
for i in range(10):
   listbox.insert(0, str(i))
```

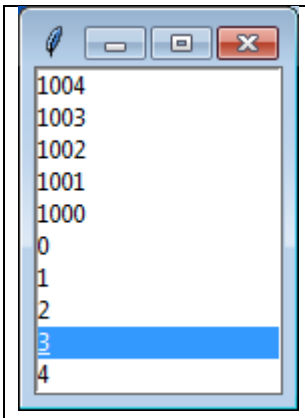



**Abbildung 11 Listbox ohne Scrollbar**

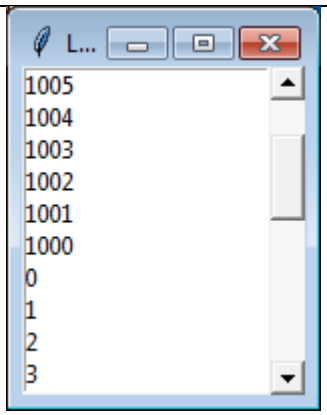



**Abbildung 12 Listbox mit Scrollbar**

**Erzeugen einer Scrollbars:**

```
sbx = tkinter.Scrollbar(frame, orient="horizontal")
sbx.pack(fill="x", side="bottom")

sby = tkinter.Scrollbar(frame)
sby.pack(fill="y", side="right")
```

## *9.7 Standarddialoge*

Mit dem Modul „filedialog“ werden die von anderen Frameworks bekannten Standarddialoge zur Verfügung gestellt.

### 9.7.1 Meldungen

Um die unteren Funktionen aufzurufen, benötigen Sie folgende Import-Anweisung:

```
from tkinter import messagebox
```

#### 9.7.1.1 showinfo

Beispiel:

```
messagebox.showinfo( "Meldung",
                "Sie erhalten eine 1,0 als Note",icon='info')
```

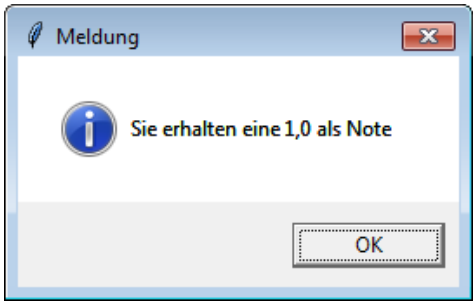



**Abbildung 13 ShowInfo unter Python**

Folgende Symbole sind möglich:

- error
- warning
- info
- question

#### 9.7.1.2 showwarning

Diese Variante ist eigentlich nur showinfo mit einem festen Symbol (warning)
Beispiel:

```
messagebox.showwarning( "Hello Python", "Hello World")
```


**Abbildung 14 ShowWarning unter Python**

Folgende Symbole sind möglich:

- error
- warning
- info
- question

#### 9.7.1.3 showerror

Diese Variante ist eigentlich nur showinfo mit einem festen Symbol (error)
Beispiel:

```
messagebox.showerror( "Hello Python", "Hello World")
```


**Abbildung 15 ShowError unter Python**

Folgende Symbole sind möglich:

- error
- warning
- info
- question

### 9.7.2 Abfragen

Um die unteren Funktionen aufzurufen, benötigen Sie folgende Import-Anweisung:

```
from tkinter import messagebox
```

### 9.7.2.1 askokcancel

Beispiel:

```
messagebox.askokcancel( "Note", "Sie erhalten eine 1,0")
```

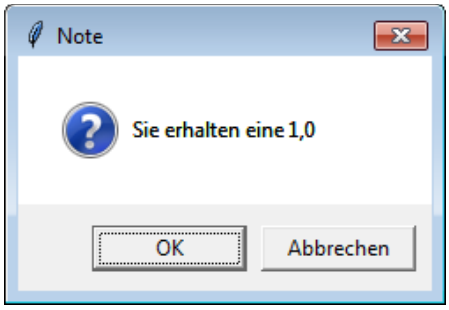



**Abbildung 16 AskOkCancel unter Python**

Folgende Symbole sind möglich:

- error
- warning
- info
- question

### 9.7.2.2 askquestion

Entspricht eigentlich dem yesno.

Beispiel:

```
messagebox.askquestion ( "Note", "Wollen Sie eine 1,0?")
```

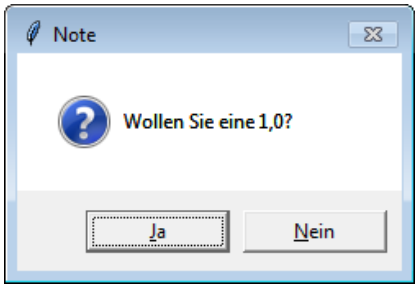



**Abbildung 17 AskQuestion unter Python**

Folgende Symbole sind möglich:

- error
- warning

- info
- question

Mit default kann man den Standardschalter definieren:

- `messagebox.askokcancel( "Note", "Sie erhalten eine 1,0", default='yes')`
- `messagebox.askokcancel( "Note", "Sie erhalten eine 1,0", default='no')`

### 9.7.2.3 askyesno

Beispiel:

```
result = messagebox.askyesno("Delete", "Are You Sure?",
                             icon='question', default='no')
if result == True:
  messagebox.showinfo("Deleted","Deleted")
else:
  messagebox.showinfo("Not Deleted","Not Deleted")
```

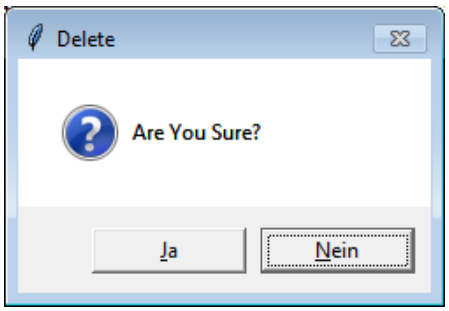



**Abbildung 18 AskYesNo unter Python**

Folgende Symbole sind möglich:

- error
- warning
- info
- question

Mit default kann man den Standardschalter definieren:

- `messagebox.askokcancel( "Note", "Sie erhalten eine 1,0", default='yes')`
- `messagebox.askokcancel( "Note", "Sie erhalten eine 1,0", default='no')`

### 9.7.2.4 askyesnocancel

Beispiel:

```
result = messagebox.askyesnocancel( "Beenden",
                                    "Änderungen speichern?")
if result == True:
  messagebox.showinfo("Beenden","Save")
elif result == False:
  messagebox.showinfo("Beenden","No Save")
```

```
elif result == None:
  messagebox.showinfo("Beenden","No Close")
else:
  messagebox.showinfo("Beenden","Fehlerhafte Abfrage")
```

Eine switch/Case-Anweisung gibt es in Python nicht.

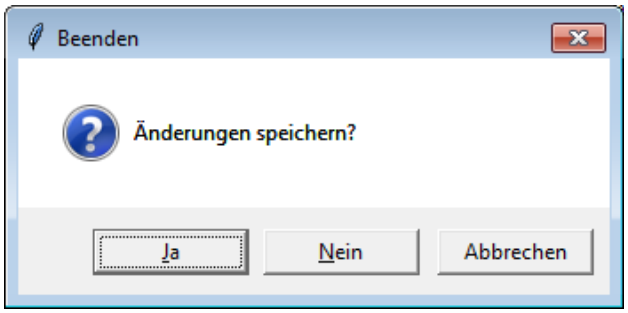



**Abbildung 19 AskYesNoCancel unter Python**

Folgende Symbole sind möglich:

- error
- warning
- info
- question

Mit default kann man den Standardschalter definieren:

- `messagebox.askokcancel( "Note", "Sie erhalten eine 1,0", default='yes')`
- `messagebox.askokcancel( "Note", "Sie erhalten eine 1,0", default='no')`

### 9.7.2.5 askretrycancel

Beispiel:

```
result = messagebox.askretrycancel( "Datei öffnen",
                 "Fehler beim Öffnen, noch einmal versuchen?")
if result == True:
  messagebox.showinfo("Datei öffnen","Noch einmal")
else:
  messagebox.showinfo("Datei öffnen",
                              "Ende, kein weiterer Versuch")
```

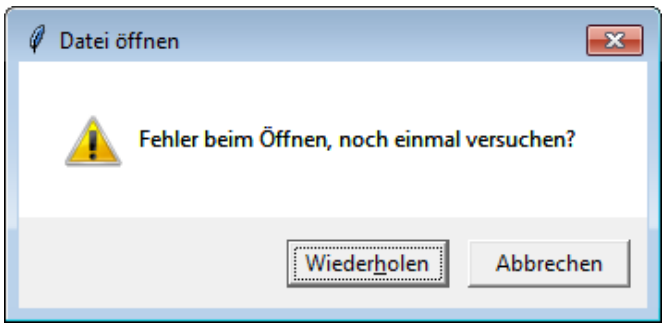



**Abbildung 20 AskYesNoCancel unter Python**

Folgende Symbole sind möglich:

- error
- warning
- info
- question

Mit default kann man den Standardschalter definieren:

- `messagebox.askokcancel( "Note", "Sie erhalten eine 1,0", default='yes')`
- `messagebox.askokcancel( "Note", "Sie erhalten eine 1,0", default='no')`

### 9.7.3 Eingabe-Dialoge

Um die unteren Funktionen aufzurufen, benötigen Sie folgende Import-Anweisung:

```
from tkinter import simpledialog
```

#### 9.7.3.1 askstring

Beispiel:

```
result = simpledialog.askstring( "Hello Python", "Hello World")
if result == None:
  messagebox.showinfo("Eingabe","keine")
else:
  messagebox.showinfo("Eingabe",result)
```

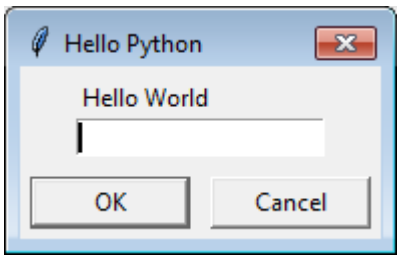



**Abbildung 21 AskString unter Python**

### 9.7.3.2 askint

Beispiel:

```
result = simpledialog.askinteger( "Hello Python", "Hello World",
                      initialvalue=42, minvalue=1, maxvalue=99)
if result == None:
  messagebox.showinfo("Eingabe","keine")
else:
  messagebox.showinfo("Eingabe",result)
```

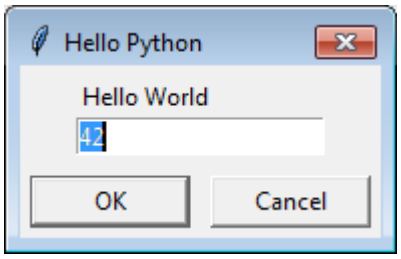



**Abbildung 22 AskInt unter Python**

### 9.7.3.3 askfloat

Beispiel:

```
result = simpledialog.askfloat( "Hello Python", "Hello World",
              initialvalue=42, minvalue=1, maxvalue=99)
if result == None:
  messagebox.showinfo("Eingabe","keine")
else:
  messagebox.showinfo("Eingabe",result)
```

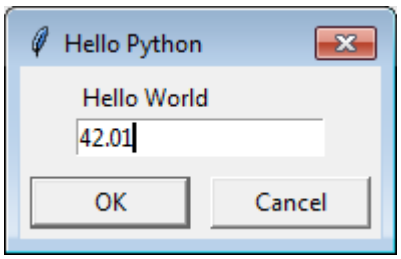



**Abbildung 23 AskFloat unter Python**

### 9.7.4 File-Dialoge

Um die unteren Funktionen aufzurufen, benötigen Sie folgende Import-Anweisung:

```
from tkinter import filedialog
```

#### 9.7.4.1 OpenFile-Dialog

**Aufruf:**

- filedialog.askopenfilename

**Optionen**:

- title
  - title="Dateien laden"
- initialdir
  - initialdir="D:\\progs\\Python\\UI\\"
- multiple
  - Mehrfachauswahl
  - multiple=True
- filetypes
  - Dateimasken
  - filetypes=( ("Pythondateien",".py" ) ,("Alle Dateien","*.*" ) ) )

**Beispiel:**

```
filename = filedialog.askopenfilename(title="Dateien laden",
   initialdir="D:\\progs\\Python\\UI\\", multiple=True,
   filetypes=( ("Pythondateien",".py" ),("Alle Dateien","*.*" ) ) )
if filename:
  messagebox.showinfo("Auswahl",filename)
else:
  messagebox.showinfo("Auswahl","Abbruch")
```

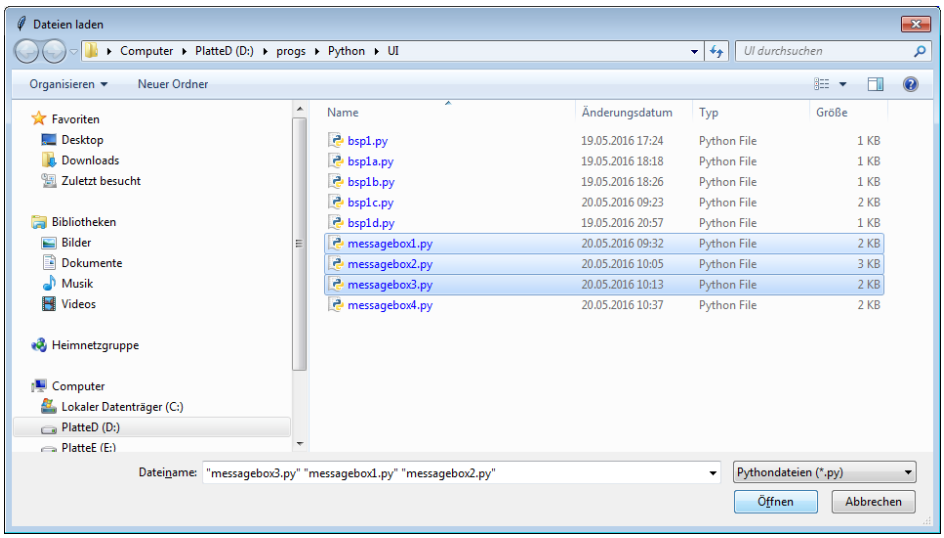



**Abbildung 24 Auswahl von Dateien**

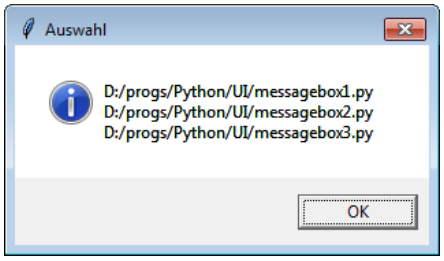



**Abbildung 25 Ausgabe der ausgewählten Dateien**

### 9.7.4.2 SaveAsFile-Dialog

**Aufruf:**

- filedialog.asksavefilename

**Optionen**:

- title
  - title="Dateien speichern"
- initialdir
  - initialdir="D:\\progs \\Python\\UI\\"
- multiple
  - Mehrfachauswahl
  - multiple=True
- filetypes
  - Dateimasken
  - filetypes=( ("Pythondateien",".py" ) ,("Alle Dateien","*.*" ) ) )

**Beispiel:**

```
filename = filedialog.asksaveasfilename(initialfile="Beispiel.py",
    title="Dateien laden",
   initialdir="D:\\progs\\Python\\UI\\",
   filetypes=( ("Pythondateien",".py" ),("Alle Dateien","*.*" ) ) )
if filename:
  messagebox.showinfo("Auswahl",filename)
else:
  messagebox.showinfo("Auswahl","Abbruch")
```

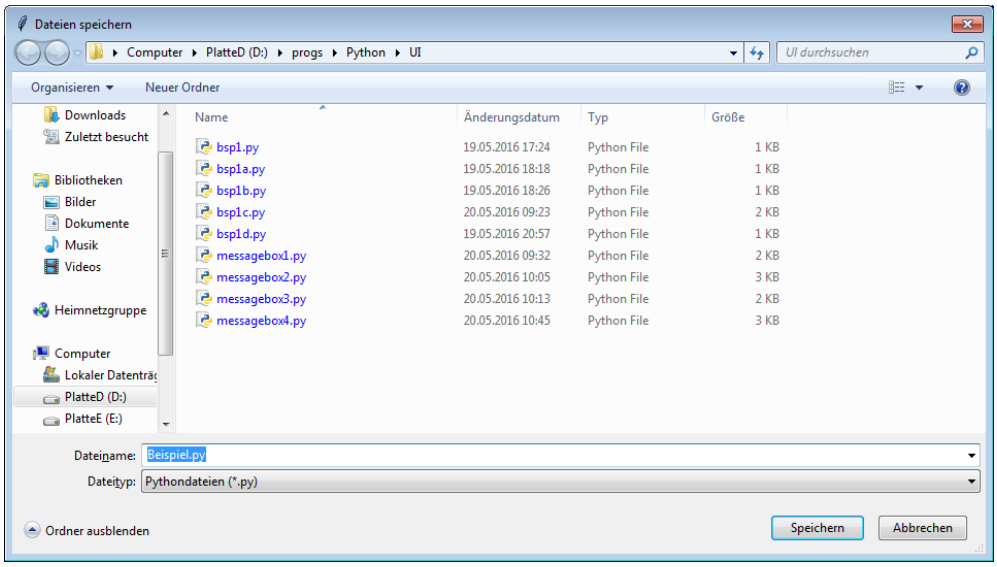



**Abbildung 26 Speichern einer Dateien**

### 9.7.4.3 Directory-Dialog

**Aufruf:**

- filedialog.askdirectory

**Optionen**:

- title
  - title="Dateien speichern"
- initialdir
  - initialdir="D:\\progs\\Python"
- mustexists
  - wenn true, muss das Verzeichnis existieren

**Beispiel:**

```
filename = filedialog.askdirectory(title="Verzeichnis auswählen",
                                   initialdir="D:\\progs\\Python" )
if filename:
  messagebox.showinfo("Auswahl",filename)
```

```
else:
  messagebox.showinfo("Auswahl","Abbruch")
```

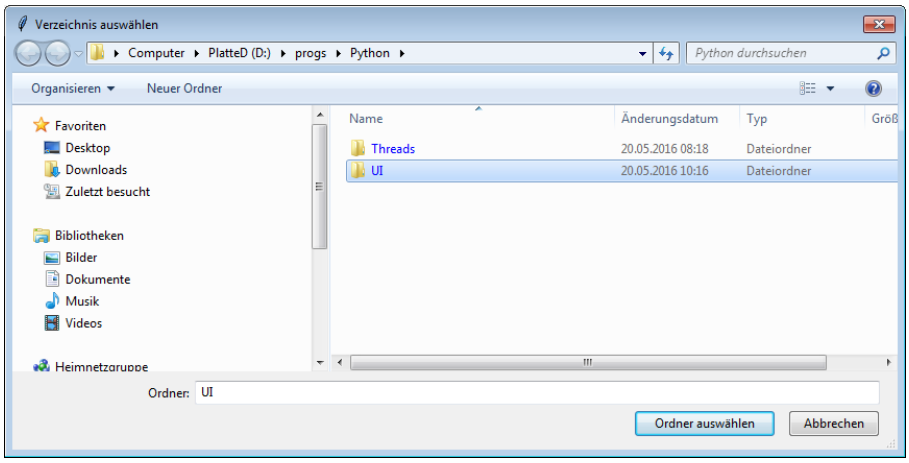



**Abbildung 27 Auswahl eines Verzeichnisses**

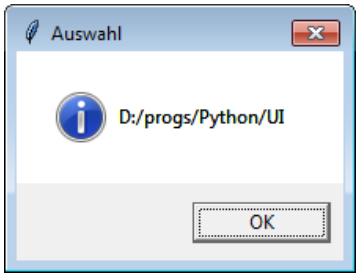



**Abbildung 28 Anzeige des ausgewählten Verzeichnisses**

# 10 Gridlayout

Das Layout mit dem “pack”-Mechanismus ist nicht immer leicht zu verstehen. Deshalb gibt es seit Jahren einen Nachfolger: das Grid-Layout.

**Beachte:**

- **Einmal Grid-Layout immer Grid-Layout!**

## *10.1 Aufbau*

Der Aufbau muss natürlich etwas modifiziert werden.

```
import tkinter

class MyApp():

   def __init__(self, root):
      root.config(bg="yellow")
      self.setGUI()

   def setGUI(self):
      pass

root = tkinter.Tk()
root.title("Mein Fenster")
root.geometry("300x300")
app = MyApp(root)
root.mainloop()
```

**Neu:**

- Die Referenz des Dialogs „root“ wird nun per Parameter übergeben.
- Beim Erzeugen eines UI-Elementes muss man nun root statt self eingeben.
- Für „Ok“ und „Quit“ verwendet man gleichfalls root
  - `self.bnOk["command"] = root.quit`
- Für eigene Action-Methoden verwendet man self.
  - `self.bnRev["command"] = self.onReverse`
- Statt „pack“ fügt man nun ein Element mit ?.grid(…) ein
  - `.grid(row=0, column=0, columnspan=3, padx="10",pady="10")`
  - `cow`
  - `col`
  - `columnspan`
  - `rowspan`
- Die Anzahl der Zeilen und Spalten werden automatisch ermittelt.

## 10.2 Einfaches Beispiel:

```
import tkinter
from tkinter import messagebox

class MyApp():

   def __init__(self, root):
      root.config(bg="yellow")
      self.setGUI()

   def setGUI(self):
      n=5
      for rownr in range(0,n,1):
         for colnr in range(0,n,1):
            bn = tkinter.Button(root)
            bn["text"] = str(rownr)+' / '+str(colnr)
            #bn.grid(row=0, column=0, padx="10",pady="10")
            bn.grid(row=rownr, column=colnr, padx="10",pady="10")
            bn.config(foreground = "blue" )
            bn.config(background = "red") #"#FF0000"


root = tkinter.Tk()
root.title("Mein Fenster")
root.geometry("300x300")
app = MyApp(root)
root.mainloop()
```

Ergebnis:

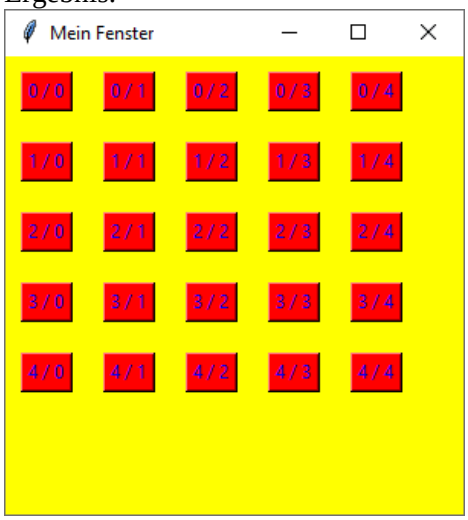



**Abbildung 29 Grid-Layout mit Schaltern**

### *10.3 Beispiel mit dem Vorgabegridlayout aus dem Wizzard:*

```
#! /usr/bin/env python3
# coding=utf8

import tkinter

class MyApp():

   def __init__(self, root):
      root.config(bg="yellow")
      self.setGUI()

   def setGUI(self):
      self.label1 = tkinter.Label(root)
      self.label1["text"]="Eingabe"
      self.label1.config(foreground="blue")
      self.label1.config(background="red")
      self.label1.grid(row=0, column=0, padx="10", pady="10")

      self.inputui = tkinter.Entry(root, background="blue",
             relief=tkinter.SUNKEN)
      self.inputui.grid(row=0, column=1, sticky="ew",
              padx="10", pady="10")

      self.var_name = tkinter.StringVar()
      self.var_name.set("Ihr Name...")
      self.inputui["textvariable"] = self.var_name

      self.bnAction = tkinter.Button(root)
      self.bnAction["text"] = "Action"
      self.bnAction["command"] = self.onAction
      self.bnAction.grid(row=1, column=0, padx="10", pady="10")

      self.bnQuit = tkinter.Button(root)
      self.bnQuit["text"] = "Ende"
      self.bnQuit["command"] = root.quit
      self.bnQuit.grid(row=1, column=1, padx="10", pady="10")

   def onAction(self):
      self.var_name.set(self.var_name.get()[::- 1])
      messagebox.showinfo("Hello Python", "Hello World")

root = tkinter.Tk()
root.title("Mein Fenster")
root.geometry("350x200")
app = MyApp(root)
root.mainloop()
```

**Anzeige**

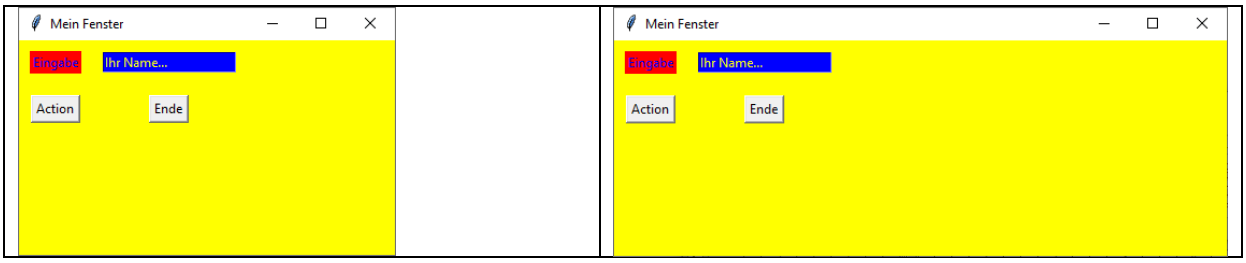



**Abbildung 30 Anzeige der Vorgabe des GridLayouts**

**Nachteil:**

- Das Eingabefeld wird nicht mit vergrößert resp.verkleinert.
- Die Schalter sind in jeweils einer Spalte eingetragen. Besser wäre ein Frame mit zwei Schaltern.

**Umbau:**

```
def setGUI(self):
    root.columnconfigure(1, weight=1)                    # setzt die 1. Spalte auf fill
```

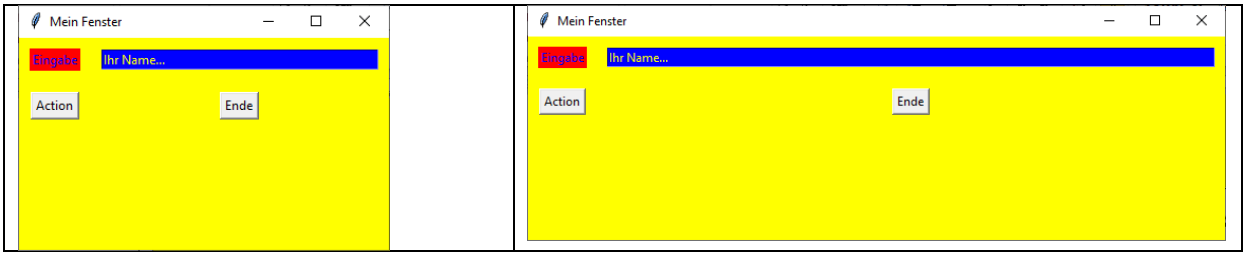



**Abbildung 31 Anzeige der Vorgabe des GridLayouts**

**Umbau:**

```
buttonframe = tkinter.Frame(root)
buttonframe.config(background = "blue") #"#FF0000"
buttonframe.grid(row=1, column=0, columnspan=2, padx="10",pady="10")

self.bnAction = tkinter.Button(buttonframe)
self.bnAction["text"] = "Action"
self.bnAction["command"] = self.onAction
self.bnAction.grid(row=0, column=0, padx="10", pady="10")

self.bnQuit = tkinter.Button(buttonframe)
self.bnQuit["text"] = "Ende"
self.bnQuit["command"] = root.quit
self.bnQuit.grid(row=0, column=1, padx="10", pady="10")
```

**Ergebnis (Layout_grid_bsp3.py):**

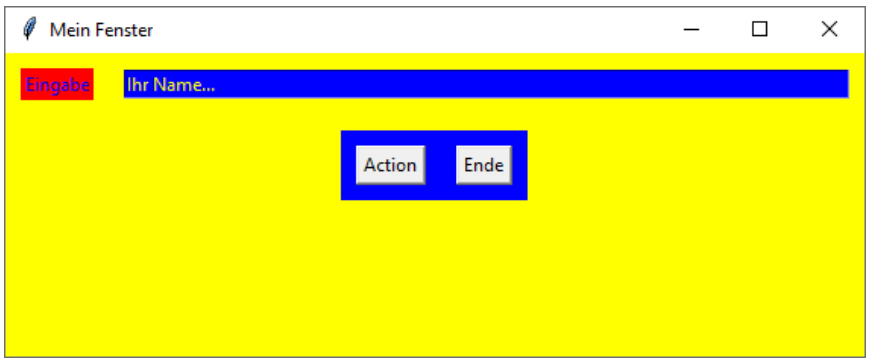



**Abbildung 32 Anzeige der Vorgabe des GridLayouts mit einem „buttonframe“**

Vollständiger Quellcode:

```
#! /usr/bin/env python3
# coding=utf8

import tkinter
from tkinter import messagebox

class MyApp():

  def __init__(self, root):
    root.config(bg="yellow")
    self.setGUI()

  def setGUI(self):
    root.columnconfigure(1, weight=1)  # setzt die 1. Spalte auf fill

    self.label1 = tkinter.Label(root)
    self.label1["text"]="Eingabe"
    self.label1.config(foreground="blue")

    self.label1.config(background="red")
    self.label1.grid(row=0, column=0, padx="10", pady="10")
    self.inputui      =      tkinter.Entry(root,      background="blue",
foreground="yellow", relief=tkinter.SUNKEN)
    self.inputui.grid(row=0,    column=1,    sticky="ew",    padx="10",
pady="10")

    self.var_name = tkinter.StringVar()
    self.var_name.set("Ihr Name...")
    self.inputui["textvariable"] = self.var_name

    buttonframe = tkinter.Frame(root)
    buttonframe.config(background = "blue") #"#FF0000"
    buttonframe.grid(row=1, column=0, columnspan=2, padx="10",pady="10")


    self.bnAction = tkinter.Button(buttonframe)
    self.bnAction["text"] = "Action"
    self.bnAction["command"] = self.onAction
    self.bnAction.grid(row=0, column=0, padx="10", pady="10")

    self.bnQuit = tkinter.Button(buttonframe)
```

```
        self.bnQuit["text"] = "Ende"
        self.bnQuit["command"] = root.quit
        self.bnQuit.grid(row=0, column=1, padx="10", pady="10")

    def onAction(self):
        self.var_name.set(self.var_name.get()[::- 1])
        messagebox.showinfo("Hello Python", "Hello World")

root = tkinter.Tk()
root.title("Mein Fenster")
root.geometry("350x200")
app = MyApp(root)
root.mainloop()
```

## *10.4 Komplexes Beispiels (Layout_grid_bsp4.py)*

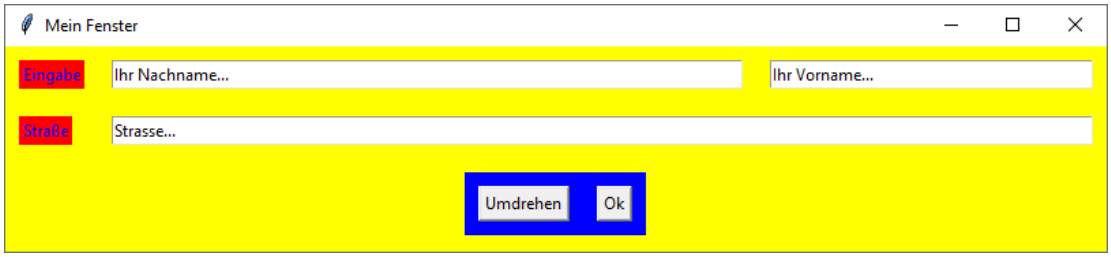



**Abbildung 33 Komplexes Beispiel (3 Eingabefelder)**

### 10.4.1 Grid-Layout

| | | | | |
|---|---|---|---|---|
| Columns: | auto | weight=3 | weight=1 | |
| Rows | auto | auto | auto | # braucht man nicht einzutragen |

```
root.columnconfigure(1, weight=3)  # setzt die 1. Spalte auf fill
root.columnconfigure(2, weight=1)  # setzt die 2. Spalte auf fill
```

### 10.4.2 Labels

```
self.label1 = tkinter.Label(root)
self.label1["text"] = "Eingabe"
self.label1.config(foreground = "blue" )
self.label1.config(background = "red")
self.label1.grid(row=0, column=0, sticky="W", padx="10",pady="10")

self.label3 = tkinter.Label(root)
self.label3["text"] = "Straße"
self.label3.config(foreground = "blue" )
self.label3.config(background = "red")
self.label3.grid(row=1, column=0,sticky="W", padx="10",pady="10")
```

Bei Label sollte man den „sticky“-Parameter setzen. Diese platziert das Labelelement innerhalb der zelle. Der Westparameter ist dabei die sinnvollste Variante.

### 10.4.3 Eingabeelemente

```
self.inputui1    =    tkinter.Entry(root,    background    =    "white",
relief=tkinter.SUNKEN)
self.inputui1.grid(row=0, column=1,sticky="NSEW",padx="10",pady="10")
self.var_name1 = tkinter.StringVar()
self.var_name1.set("Ihr Nachname...")
self.inputui1["textvariable"] = self.var_name1

self.inputui2 = tkinter.Entry(root, background = "white",
       relief=tkinter.SUNKEN)
self.inputui2.grid(row=0, column=2,sticky="NSEW",padx="10",pady="10")
self.var_name2 = tkinter.StringVar()
self.var_name2.set("Ihr Vorname...")
self.inputui2["textvariable"] = self.var_name2

self.inputui3 = tkinter.Entry(root, background = "white",
       relief=tkinter.SUNKEN)
self.inputui3.grid(row=1, column=1, columnspan=2,
       sticky="NSEW",padx="10",pady="10")

self.var_name3 = tkinter.StringVar()
self.var_name3.set("Strasse...")
self.inputui3["textvariable"] = self.var_name3
```

**Wichtig:**

- Ohne den sticky-Eintrag wird nur die Zelle „gezoomt“!
- Also immer eintragen „**sticky="NSEW**"

### 10.4.4 Schalter

```
    buttonframe = tkinter.Frame(root)
    buttonframe.config(background = "blue") #"#FF0000"
    #buttonframe.pack(fill="x", side="bottom" )
    buttonframe.grid(row=2, column=0, columnspan=3, padx="10",pady="10")

# der buttonframe hat nun für die Schalter einj eigenes Koordinatensystem
    self.bnAction = tkinter.Button(buttonframe)
    self.bnAction["text"] = "Umdrehen"
    self.bnAction["command"] = self.onReverse
    self.bnAction.grid(row=0, column=0, padx="10",pady="10")

    self.bnOk = tkinter.Button(buttonframe)
    self.bnOk["text"] = "Ok"
    self.bnOk["command"] = root.quit
    self.bnOk.grid(row=0, column=1, padx="10",pady="10")
```

## *10.5 Komplexes Beispiel (Layout_grid_bsp5.py)*

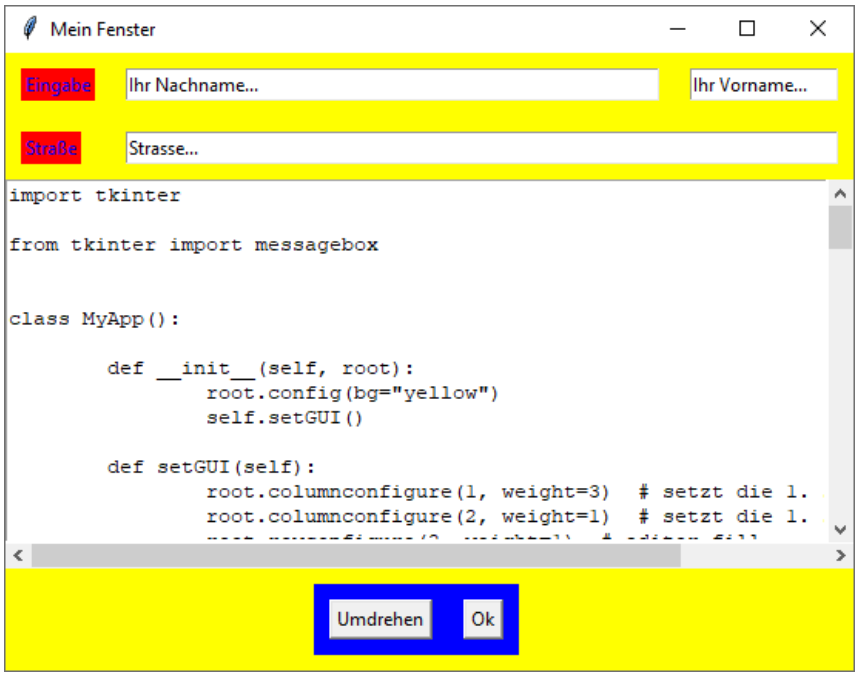



**Abbildung 34 Komplexes Beispiels (3 Eingabefelder)**

### 10.5.1 Grid-Layout

| | | | | |
|---|---|---|---|---|
| Columns: | auto<br>Label | weight=3<br>entry | weight=1<br>entry | auto<br>y-scrollbar |
| Rows | auto | auto | weight=1 | auto |

```
root.columnconfigure(1, weight=3)  # setzt die 1. Spalte auf fill
root.columnconfigure(2, weight=1)  # setzt die 2. Spalte auf fill
root.rowconfigure(2, weight=1)  # editor fill
```

### 10.5.2 Labels

```
self.label1 = tkinter.Label(root)
self.label1["text"] = "Eingabe"
self.label1.config(foreground = "blue" )
self.label1.config(background = "red")
self.label1.grid(row=0, column=0, sticky="W", padx="10",pady="10")

self.label3 = tkinter.Label(root)
self.label3["text"] = "Straße"
self.label3.config(foreground = "blue" )
self.label3.config(background = "red")
self.label3.grid(row=1, column=0,sticky="W", padx="10",pady="10")
```

Bei Label sollte man den „sticky“-Parameter setzen. Diese platziert das Labelelement innerhalb der zelle. Der Westparameter ist dabei die sinnvollste Variante.

### 10.5.3 Einzeilige Eingabeelemente

```
self.inputui1    =    tkinter.Entry(root,    background    =    "white",
relief=tkinter.SUNKEN)
self.inputui1.grid(row=0, column=1, sticky="NSEW", padx="10", pady="10")
self.var_name1 = tkinter.StringVar()
self.var_name1.set("Ihr Nachname...")
self.inputui1["textvariable"] = self.var_name1

self.inputui2 = tkinter.Entry(root, background = "white",
       relief=tkinter.SUNKEN)
self.inputui2.grid(row=0, column=2, columnspan=2,   sticky="NSEW",
           padx="10",pady="10")
self.var_name2 = tkinter.StringVar()
self.var_name2.set("Ihr Vorname...")
self.inputui2["textvariable"] = self.var_name2

self.inputui3 = tkinter.Entry(root, background = "white",
       relief=tkinter.SUNKEN)
self.inputui3.grid(row=1, column=1, columnspan=3,   sticky="NSEW",
           padx="10", pady="10")

self.var_name3 = tkinter.StringVar()
self.var_name3.set("Strasse...")
self.inputui3["textvariable"] = self.var_name3
```

**Wichtig:**

- Ohne den sticky-Eintrag wird nur die Zelle „gezoomt“!
- Also immer eintragen „**sticky="NSEW**"

### 10.5.4 Editor

Als erstes wird der Editor und danach die beiden Scrollbar erstellt und eingefügt. Danach werden die drei Elemente verknüpft.

```
self.editor = tkinter.Text(root)
self.editor.config(wrap="none")  # wrap="word"  word char
self.editor.grid(row=2, column=0, columnspan=3, sticky="NSEW",
  padx="0", pady="0")

sbx = tkinter.Scrollbar(root, orient="horizontal")
sbx.grid(row=3, column=0, columnspan=4, sticky="NSEW", pady="0")

sby = tkinter.Scrollbar(root)
sby.grid(row=2, column=3, columnspan=1, sticky="NSEW", pady="0")

self.editor["xscrollcommand"] = sbx.set
```

```
   sbx["command"] = self.editor.xview
   self.editor["yscrollcommand"] = sby.set
   sby["command"] = self.editor.yview
```

### 10.5.5 Schalter

```
      buttonframe = tkinter.Frame(root)
      buttonframe.config(background = "blue") #"#FF0000"
      #buttonframe.pack(fill="x", side="bottom" )
      buttonframe.grid(row=2, column=0, columnspan=3, padx="10",pady="10")

# der buttonframe hat nun für die Schalter einj eigenes Koordinatensystem
      self.bnAction = tkinter.Button(buttonframe)
      self.bnAction["text"] = "Umdrehen"
      self.bnAction["command"] = self.onReverse
      self.bnAction.grid(row=0, column=0, padx="10",pady="10")

      self.bnOk = tkinter.Button(buttonframe)
      self.bnOk["text"] = "Ok"
      self.bnOk["command"] = root.quit
      self.bnOk.grid(row=0, column=1, padx="10",pady="10")
```

### 10.5.6 Vollständiger Quellcode

```
import tkinter

from tkinter import messagebox


class MyApp():

   def __init__(self, root):
      root.config(bg="yellow")
      self.setGUI()

   def setGUI(self):
      root.columnconfigure(1, weight=3)  # setzt die 1. Spalte auf fill
      root.columnconfigure(2, weight=1)  # setzt die 1. Spalte auf fill
      root.rowconfigure(2, weight=1)  # editor fill

      self.label1 = tkinter.Label(root)
      self.label1["text"] = "Eingabe"
      self.label1.config(foreground = "blue" )
      self.label1.config(background = "red")
      self.label1.grid(row=0, column=0, sticky="W", padx="10",pady="10")

      self.inputui1 = tkinter.Entry(root, background = "white",
             relief=tkinter.SUNKEN)
      self.inputui1.grid(row=0, column=1,sticky="NSEW",padx="10",pady="10")
      self.var_name1 = tkinter.StringVar()
      self.var_name1.set("Ihr Nachname...")
      self.inputui1["textvariable"] = self.var_name1

      self.inputui2 = tkinter.Entry(root, background = "white",
```

```
            relief=tkinter.SUNKEN)
      self.inputui2.grid(row=0, column=2, columnspan=2,sticky="NSEW",
            padx="10",pady="10")
      self.var_name2 = tkinter.StringVar()
      self.var_name2.set("Ihr Vorname...")
      self.inputui2["textvariable"] = self.var_name2

#-----------------------------------------

      self.label3 = tkinter.Label(root)
      self.label3["text"] = "Straße"
      self.label3.config(foreground = "blue" )
      self.label3.config(background = "red")
      self.label3.grid(row=1, column=0, sticky="W",  padx="10",pady="10")

      self.inputui3    =    tkinter.Entry(root,    background    =    "white",
relief=tkinter.SUNKEN)
      self.inputui3.grid(row=1, column=1, columnspan=3, sticky="NSEW",
            padx="10",pady="10")
      self.var_name3 = tkinter.StringVar()
      self.var_name3.set("Strasse...")
      self.inputui3["textvariable"] = self.var_name3

# ----------------------------------------


      self.editor = tkinter.Text(root)
      self.editor.config(wrap="none")  # wrap="word"  word char
      self.editor.grid(row=2, column=0,columnspan=3, sticky="NSEW",
            padx="0",pady="0")

      sbx = tkinter.Scrollbar(root, orient="horizontal")
      sbx.grid(row=3, column=0,columnspan=4, sticky="NSEW",pady="0")

      sby = tkinter.Scrollbar(root)
      sby.grid(row=2, column=3,columnspan=1, sticky="NSEW",pady="0")

      self.editor["xscrollcommand"] = sbx.set
      sbx["command"] = self.editor.xview
      self.editor["yscrollcommand"] = sby.set
      sby["command"] = self.editor.yview


# ----------------------------------------

      buttonframe = tkinter.Frame(root)
      buttonframe.config(background = "blue") #"#FF0000"
      buttonframe.grid(row=4, column=0, columnspan=3, padx="10",pady="10")

      self.bnAction = tkinter.Button(buttonframe)
      self.bnAction["text"] = "Umdrehen"
      self.bnAction["command"] = self.onReverse
      self.bnAction.grid(row=0, column=0, padx="10",pady="10")

      self.bnOk = tkinter.Button(buttonframe)
      self.bnOk["text"] = "Ok"
      self.bnOk["command"] = root.quit
      self.bnOk.grid(row=0, column=1, padx="10",pady="10")
```

```
# ----------------------------------------

   def onReverse(self):
      self.var_name1.set( self.var_name1.get()[::-1] )
      #messagebox.showinfo( "Hello Python", "Hello World")


root = tkinter.Tk()
root.title("Mein Fenster")
root.geometry("550x400")
app = MyApp(root)
root.mainloop()
```

## 10.5.7 Gridlayoutbeispiel mit zwei vertikalen Editoren


**Abbildung 35 Layout_grid_bsp6.py**

## 10.5.8 Gridlayoutbeispiel mit zwei horizontalen Editoren

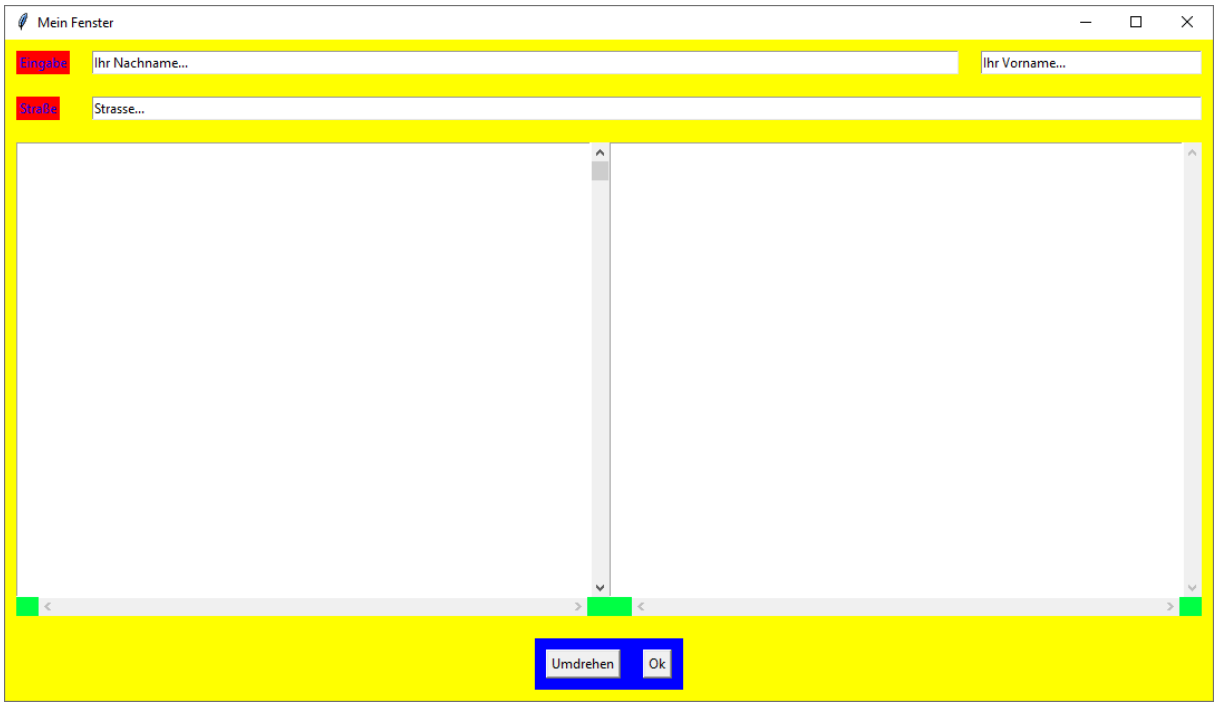



**Abbildung 36 Layout_grid_bsp7.py**

Besonderheit:

```
editorframe = tkinter.Frame(root)
editorframe.config(background = "#00FF44") #"#FF0000"
editorframe.grid(row=2, column=0, columnspan=3,sticky="NSEW",
       padx="10",pady="10")
editorframe.columnconfigure(0, weight=1)
editorframe.columnconfigure(2, weight=1)
editorframe.rowconfigure(0, weight=1)
```

Der „editorframe“ ist ein Regal mit zwei Fächern. Man muss aber die Gewichtung in der horizontalen und vertikalen Richtung **neu** eintragen!

# 11 Indexverzeichnis

## F



## G



## I



## J



## K



## L



## M

## O



## P



## R



## S



## T

## U



## W



## Z