Der Wechsel Python 2 → 3

Die meisten Standardfunktionen sind nicht wesentlich geändert worden. Kurze Hinweise zu vermutlich auftretenden Problemen:

print ist jetzt eine Funktion

Das führt in der Regel zu Fehlern bei vorhandenen Programmen, da beispielsweise **`print 3`** unzulässig ist und stattdessen print(3) verwendet werden muss.
Die Funktion kennt weitere Parameter, z.B.:

```
>>> for i in range(3): print(3,4,5, sep=' < ',end=' , ')

3 < 4 < 5 , 3 < 4 < 5 , 3 < 4 < 5 ,
```

input liefert immer einen String

```
>>> input('Eingabe: ')
Eingabe: 3
'3'
```

Daher muss man gegebenenfalls zu einem Zahlenwert konvertieren:

```
z=input('einen ganzzahligen Wert eingeben: ')
print(type(z))
zahl=int(z)
print (1+zahl)
z=input('Fliesskommazahl eingeben: ')
zahl=float(z)
print (1+zahl)
```

liefert:

```
einen ganzzahligen Wert eingeben: 3
<class 'str'>
4
Fliesskommazahl eingeben: 3.5
4.5
```

range liefert einen Iterator

Bei Zählschleifen musste in der alten Version ein Programm beispielsweise bei

```
for index in range(1000): ...
```

zunächst mit Hilfe von **`range`** eine Liste der Zahlen von 0 bis 999 erzeugen, bevor über ihre Elemente iteriert werden konnte. Die Funktion erzeugt nun einen Iterator, so dass benötigte Elemente erst dann generiert werden, wenn sie benötigt werden.

Das hat aber auch zur Folge, dass man nicht direkt mit range eine Liste erzeugen kann, sondern erst mit list erzeugen muss

```
it=range(1,10)
print(type(it))
eineListeDavon=list(it)
print(eineListeDavon)
```

liefert:

```
<class 'range'>
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

_thread statt thread

Das Paket ist umbenannt.

Man kann in fertigen Programmen bei einem Wechsel von Py2 nach Py3 natürlich die suchen und ersetzen – Funktion des Editors einsetzen. Eine andere Möglichkeit bietet der Import, bei dem man die Benennungen intern lässt, aber statt **`import _tread`** schreibt: **`import _tread as thread`**.

Exceptions

Notwendig zum Zugriff auf das Exception-Objekt ist jetzt die Verwendung von as:

```
except Exception as e:
    print(str(e))
```

Annotations

Zu den übergebenen Parametern einer Funktion sind annotations möglich, auf die beispielsweise zu einer Typprüfung zugegriffen werden kann.

```
def summe(a:int, b:int = 0) -> int:
    if not (type(a)==int): return 'Fehler'
    if not (type(b)==int): return 'Fehler'
    if a<0: return summe(a+1,b-1)
    if a>0: return summe(a-1,b+1)
    return b
```

Ganzzahldivision

Das Divisionszeichen liefert auch bei ganzzahligen Operanden jetzt eine Zahl vom Typ float. Braucht man einen ganzzahligen Wert, kann man ihn daraus mit int(…) bekommen[1] oder man verwendet den doppelten Querstrich.

```
66 / 7 → 9.428571428571429
66 // 7 → 9       [ = int(66/7) ]
```

Die Restfunktion **`66 % 7 → 3`** bleibt wie bisher.

Rekursion

Leider hat sich gegenüber Python 2 keine Änderung bei der Anwendung von Rekursion ergeben, was das Erkennen von Endrekursion[2] angeht, so dass das folgende Programm

```
def summe(n,akku):
    if n==0: return 0
    else: return summe(n-1)+n
print(summe(992,0))
```

die Summe zwar noch zu 492528 berechnen kann,

```
>>> print(summe(993,0))
```

aber wegen zu großer Rekursionstiefe abbricht.

Phoenix statt wxPython

Die zu Python 3 gehörende Version von wxPython heißt nun Phoenix und muss mit Hilfe von pip in die Pythonumgebung eingebaut werden.
Das hat aber nur kleine Änderungen[3] im Projekt Grafikfenster zur Folge.

1 da die int-Funktion alles hinter dem Dezimalpunkt weglässt
2 Siehe dazu die Fähigkeite von Scheme (Racket), bei Endrekursion den vorigen Funktionsaufruf durch den neuen Aufruf zu ersetzen.
3 self._buffer = wx.Bitmap(sz, 32) # AENDERUNG in InitBuffer
gc.SetFont(font, wx.BLACK) statt gc.SetFont(font) in Draw

Dateien vom Typ pyc

Die kompilierten Dateien werden jetzt in einem Unterverzeichnis des Projektes mit dem Namen __**pycache**__ abgelegt.

Seltener wichtig

Daher nur kurz erwähnt: Der Datentyp **`str`** für Strings verwendet jetzt grundsätzlich Unicodezeichen.
Es gibt keinen Wechsel mehr bei großen ganzen Zahlen in den Typ **`long`**, es bleibt bei **`int`**.
Binärdateien verwenden den Datentyp **`bytes`**.
Die Anweisung **`nonlocal`** für Variablen von lokalen Funktionen ermöglicht den Zugriff auf Variablen der Funktion, in die sie eingebettet sind.