HANSER

Leseprobe

zu

„Python für Ingenieure und Naturwissenschaftler“

von Hans-Bernhard Woyand

Print-ISBN: 978-3-446-46108-6
E-Book-ISBN: 978-3-446-46109-3

Weitere Informationen und Bestellungen unter
http://www.hanser-fachbuch.de/978-3-446-46108-6
sowie im Buchhandel

Vorwort

Seit vielen Jahren halte ich an der Bergischen Universität Wuppertal die Lehrveranstaltung „Informatik“ im Grundstudium Maschinenbau. Mehr als 10 Jahre haben wir in diesen Kursen die Programmiersprache C/C++ eingesetzt. Seit mehreren Jahren verwenden wir stattdessen nun *Python* als erste Programmiersprache und das mit großem Erfolg. Es zeigte sich, dass gerade bei Ingenieuren, die oftmals Schwierigkeiten mit dem algorithmischen Denken haben, Python einen leichteren Zugang ermöglicht.

Im Studiengang Maschinenbau wie auch in vielen anderen Studiengängen spielt das wissenschaftliche Rechnen und Visualisieren eine große Rolle. In der Literatur zu Python ist das nicht so: entweder werden die wissenschaftlichen Anwendungen gar nicht bzw. nur am Rande behandelt, oder die Bücher sind so umfangreich, dass sie als vorlesungsbegleitender Text ungeeignet sind. So entstand die Idee, das Skript zur Vorlesung zu diesem Buch auszuarbeiten.

Zielgruppe
Das Buch richtet sich an *Programmieranfänger*, die Python als erste Programmiersprache lernen möchten. Es wird also zuerst in Python eingeführt und dann werden die Grundlagen erläutert. Es folgen Vertiefungen und ein eigenes Kapitel über die objektorientierte Programmierung. Dann beginnt das wissenschaftliche Rechnen und Visualisieren. Hierzu wird in die Nutzung der wichtigsten Programmbibliotheken (Packages) eingeführt. Hauptzielgruppe sind also Studierende wissenschaftlicher Studiengänge, die Python erlernen wollen und bei denen die mathematischen und grafischen Anwendungen eine wichtige Rolle spielen. Vorausgesetzt wird nur mathematisch-naturwissenschaftliches Wissen, wie es an höheren Schulen vermittelt wird.

Aufgabenorientierte Lehre
Mehr als 90 Aufgabenstellungen mit fast immer kommentierten Lösungen werden im Buch behandelt. Nach meiner Erfahrung lernen Studierende am meisten durch das selbstständige Lösen von Aufgaben.

Software
Die Software, die benötigt wird, um mit diesem Buch zu arbeiten, ist kostenfrei erhältlich. Es handelt sich dabei um die neuere Variante 3.2 bzw. 3.5 der Programmiersprache Python, sowie deren Vorgängerversionen 2.6 und 2.7. Die wissenschaftlichen Anwendungspakete sind nämlich *noch nicht alle* mit der neuen Python-Version kompatibel. Es ist zu erwarten, dass es auch noch einige Zeit dauern wird, bis die wissenschaftlichen Pakete unter der neuesten Version laufen. Für Programmieranfänger ist der Unterschied zwischen diesen Versionen sowieso nicht sehr bedeutsam.

Webseite zum Buch
Zu dem vorliegenden Buch existiert eine Webseite, auf der in loser Folge Ergänzungen, Fehlerberichtigungen usw. bereitgestellt werden. Weiterhin können alle Beispiele sowie die Lösungen der Aufgaben dort abgerufen werden. Für Nutzer des Betriebssystems MS-Windows sind auch Hinweise zur Installation der Software dort verfügbar. Die Web-Adresse ist

http://woyand.eu/python_buch

Falls sich diese Adresse einmal ändern sollte, so kann der Leser den neuen Standort über die Homepage des Verlags *http://www.hanser-fachbuch.de* erfahren. Dort bestehen ebenfalls Download-Möglichkeiten.

Hinweise, Fehlermeldungen und Anregungen werden über die E-Mail-Adresse

info@woyand.eu

gern entgegengenommen.

Haftungsausschluss
Das Buch wurde mit größtmöglicher Sorgfalt erstellt. Trotzdem können Fehler nicht ganz ausgeschlossen werden. Aus diesem Grund sind die in diesem Buch dargestellten Verfahren mit keinerlei Garantie verbunden. Ich weise darauf hin, dass weder Autor noch Verlag eine Haftung für direkt oder indirekt entstandene Schäden übernehmen, die sich aus der Benutzung dieses Buches ergeben könnten.

Danksagung
Ich danke meiner Frau Annette Woyand für die sorgfältige Durchsicht des Manuskripts. Frau Mirja Werner vom Carl Hanser Verlag danke ich für Ihr Engagement beim Zustandekommen dieses Buches.

Ich wünsche allen Lesern viel Erfolg und Spaß beim Einstieg in Python.

Wuppertal, im März 2017 *Hans-Bernhard Woyand*

Vorwort zur zweiten Auflage

Für die zweite Auflage wurden einige inhaltliche Verbesserungen vorgenommen und ein neues Kapitel hinzugefügt. In diesem Kapitel wird gezeigt, wie numerische Berechnungen mit der Python-Bibliothek Scipy durchgeführt werden können. Scipy ist sehr umfangreich. Deshalb werden – dem Ansatz dieses Buches entsprechend – wichtige Teilbereiche dieser Software-Bibliothek auf wenigen Seiten vorgestellt. Behandelt wird die numerische Berechnung von Integralen, die Interpolation, die Berechnung von Nullstellen, die numerische Optimierung, die Signalanalyse mit der schnellen Fourier Transformation (FFT) sowie die numerische Integration gewöhnlicher Differenzialgleichungen.

Ich hoffe, dass diese Erweiterung des Buchs für viele Leser hilfreich und anregend ist und wünsche viel Erfolg und Spaß mit Python.

Wuppertal, im Juli 2018 *Hans-Bernhard Woyand*

Vorwort zur dritten Auflage

Für die dritte Auflage wurde das Kapitel 8 (3D-Grafik mit VPython) völlig überarbeitet, da sich die neue Version VPython 7 deutlich von den Vorgängerversionen unterscheidet. Mit dieser neuen Version von VPython können die dreidimensionalen Szenen im Webbrowser dargestellt werden. Auch die Benutzerinteraktionen wurden erheblich vereinfacht und neu strukturiert.

Weiterhin wurde das Kapitel 10 (Numerische Analysen mit Scipy) um zwei Themen erweitert. Es handelt sich um die Erzeugung von Dreiecksnetzen mit der Delaunay-Triangulierung sowie um die Berechnung der konvexen Hülle einer Punktmenge. Vier weitere Aufgaben mit kommentierten Lösungen wurden dem Buch hinzugefügt.

Wuppertal, im Januar 2019 *Hans-Bernhard Woyand*

Inhaltsverzeichnis

Diese Beispiele zeigen, dass wir mithilfe von Klammern die Auswertungsreihenfolge innerhalb von mathematischen Ausdrücken beeinflussen können. Dies geschieht so, wie wir das intuitiv aus dem Prinzip „Punkt- vor Strichrechnung“ vermuten. Zuerst wird die Multiplikation bzw. die Division ausgeführt, anschließend die Addition bzw. die Subtraktion. Das Setzen von Klammern erzwingt gegebenenfalls eine andere Auswertungsreihenfolge.

Der Ausdruck 2+7/5 ergibt die Zahl 3.4. Statt des Kommas, das wir beim Schreiben auf Papier verwenden, werden Dezimalzahlen in fast allen Programmiersprachen mithilfe eines *Dezimalpunkts* codiert. Statt 5,0 schreiben wir also 5.0. Wir sprechen deshalb in der Programmierung von *Gleitpunktzahlen*. Wenn wir das Divisionszeichen in Python 3.x verwenden, so wird immer eine so genannte *Gleitpunkt-Division* durchgeführt, d.h. auch wenn beide Operanden ganzzahlig sind, ist das Ergebnis eine Gleitpunktzahl.

```
>>> 7/5
1.4
```

Das ist wichtig zu wissen! In der älteren Python-Version 2.x wurde eine Ganzzahl-Division durchgeführt, wenn beide Operanden ganze Zahlen waren. Dies führte oft zu unbeabsichtigten Fehlern bei Ausdrücken wie 1/2. Das Ergebnis war dann 0. Soll in Python 3.x eine Ganzzahl-Division durchgeführt werden, so muss dies mit dem speziellen Operator // codiert werden. Hierzu auch ein Beispiel:

```
>>> 7//5
1
>>> 22//4
5
>>> 22.0//4.0
5.0
```

Wenn man sich für den Rest bei der Durchführung einer ganzzahligen Division interessiert, so kann dieser mit dem Operator % ermittelt werden. Dieser Operator wird *Modulo-Operator* genannt.

```
>>> 7%5
2
>>> 22%4
2
```

Merke

In Python 2.x – wie auch in anderen Programmiersprachen (z.B. C/C++) – muss beachtet werden, dass bei der Division von ganzen Zahlen eine Ganzzahl-Division durchgeführt wird. Ist jedoch nur einer der Operanden eine Dezimalzahl, so wird eine Gleitpunkt-Division ausgeführt. In der neueren Sprachversion Python 3.x, *die wir in diesem Buch benutzen,* wurde dies abgeschafft. Dort wird *jede* Division, die mit dem Operator / ausgeführt wird, als Gleitpunkt-Division durchgeführt. Der Programmierer kann jedoch auch eine Ganzzahl-Division durch die Anwendung eines besonderen Operators (// statt /) ausführen lassen.

Das folgende Beispiel zeigt, dass sogenannte rein periodische Zahlen natürlich nur mit einer endlichen Anzahl von Nachkommastellen dargestellt werden können.

```
>>> 32/3
10.666666666666666
```

Der Digitalrechner kann *nicht jede Zahl exakt* darstellen. Aufgrund der internen Darstellung in Form von binären Zuständen (jede Zahl wird intern durch eine Folge von Nullen und Einsen aufgebaut) können nicht alle Zahlen völlig präzise dargestellt werden. Dieser Sachverhalt spielt eine wichtige Rolle beim numerischen Rechnen. In unserem Fall ist die Abweichung klein und belanglos. Werden jedoch sehr viele arithmetische Operationen durchgeführt, so kann sich ein erheblicher Gesamtfehler akkumulieren.

1.3.3 Vergleichsausdrücke

Fahren wir mit unserer Erkundungstour durch Python fort. Von Handrechnungen kennen wir die Vergleichsoperatoren < und >. Wir probieren sie aus:

```
>>> 2<7
True
>>> 2>7
False
```

Vergleichsausdrücke werden von Python als wahr (engl. true) und falsch (engl. false) ausgewertet.

```
>>> 2==7
False
>>> 2!=7
True
>>> 5==5
True
```

Beim Testen auf Gleichheit wird kein Gleichheitszeichen geschrieben, sondern *zwei aufeinanderfolgende Gleichheitszeichen*. Beim Test auf Ungleichheit wird der Operator „!=“ verwendet. Dieser Operator wird auch in den populären Programmiersprachen C und C++ verwendet. Python entlehnt viele Sprachelemente aus diesen Sprachen.

```
>>> 3<3
False
>>> 3<=3
True
```

Das letzte Beispiel zeigt uns den Unterschied zwischen „<“ und „<=“. Probieren Sie selbst ein Beispiel mit „>“ und „>=“ aus!

1.3.4 Logische Ausdrücke

Ausdrücke, die wahr oder falsch sind, können zu komplexeren Ausdrücken mithilfe der logischen Operatoren *and, or* und *not* zusammengefügt werden. Hier einige Beispiele für solche Ausdrücke, die auch boolesche (engl. boolean) Ausdrücke genannt werden.

```
>>> not(5==5)
False
>>> (2<7) and (5>4)
True
>>> (2>7) or (4>5)
False
>>> not (3!=3)
True
```

Ein Ausdruck der mit dem Und-Operator *and* gebildet wird, ist dann und nur dann wahr, wenn beide Teilausdrücke bzw. Operanden wahr sind. Umgekehrt ist ein mit dem Oder-Operator gebildeter Ausdruck schon dann wahr, wenn nur ein einziger Operand wahr ist.

1.3.5 Mathematische Funktionen

Wir wollen mathematische Funktionen anwenden und probieren Folgendes aus:

```
>>> sin(90)

Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    sin(90)
NameError: name 'sin' is not defined
```

Unsere Absicht war es, den Sinuswert von 90 (Grad) auszurechnen. Nun werden wir zum ersten Mal mit einer Fehlermeldung von Python konfrontiert. Diese ist auch hinreichend klar in englischer Sprache formuliert: „name 'sin' is not defined“. Dies bedeutet übersetzt etwa: „der Name ‚sin’ ist nicht definiert“. Sollte Python keine mathematischen Standardfunktionen bereitstellen? Dann wäre ja jeder Taschenrechner „intelligenter“. Tatsächlich ist es so, dass wir die mathematischen Funktionen zunächst „laden“ müssen, bevor wir diese anwenden können. Dies geschieht durch eine import-Anweisung. Mithilfe dieser Anweisung wird ein sogenanntes „Modul“ geladen. Was dies genau bedeutet, werden wir im Kapitel 3 genauer verstehen lernen.

```
>>> from math import *
>>> sin(90)
0.8939966636005785
>>> sin(180)
-0.80115263573383044
```

Nachdem wir die mathematischen Funktionen aus dem Modul *math* importiert haben, können wir die Sinusfunktion problemlos aufrufen. Dies gilt auch für andere mathematische Standardfunktionen. Probieren Sie einfach mal den Kosinus (cos) aus!

Allerdings wird uns auch klar, dass der Zahlenwert, der dieser Funktion übergeben wird, *nicht* als Winkel interpretiert wird. Sonst müssten im vorigen Beispiel die Werte 1 und 0 herauskommen. Wir vermuten, dass das Argument *im Bogenmaß* einzugeben ist und wandeln das Beispiel etwas ab.

```
>>> pi
3.141592653589793
>>> sin(pi/2)
1.0
>>> sin(pi)
1.2246467991473532e-16
```

Der math-Modul muss nur einmal pro Sitzung importiert werden. In diesem Modul ist eine Variable mit Namen pi definiert. Wir berechnen den Sinuswert von pi/2 (entsprechend 90 Grad) und von pi (entsprechend 180 Grad). Die letzte Zeile im letzten Codeabschnitt ist auf den ersten Blick verwirrend. Es sollte eigentlich genau 0 herauskommen. Stattdessen erhalten wir 1.2246467991473532e-16. Dies bedeutet in der Schreibweise für *Dezimalzahlen:* $1{,}2246467991473532\ 10^{-16}$.

Dies ist eine sehr kleine Zahl, fast null. Damit haben wir auch eine zweite Darstellungsform für Gleitpunktzahlen kennen gelernt: die *Exponentendarstellung*.

Merke

Die mathematischen Standardfunktionen Sinus, Cosinus, Tangens etc. werden mit dem *Bogenmaß* als Argument aufgerufen. Sie müssen einen Winkel also zuerst ins Bogenmaß umrechnen, um dann den Wert der Funktion zu ermitteln.

1.3.6 Grundlegendes über Variablen und Zuweisungen

Wir wollen nun das wichtige Konzept der *Variablen* kennenlernen. Betrachten Sie dazu den folgenden Programmcode:

```
>>> a

Traceback (most recent call last):
  File "<pyshell#44>", line 1, in <module>
    a
NameError: name 'a' is not defined
>>> a = 7
>>> b = 3
>>> c = a*b
>>> print("c = ",c)
c =  21
>>> a
7
```

Zunächst geben wir den Buchstaben a ein. Python antwortet daraufhin mit der Fehlermeldung, dass der Name a nicht definiert ist.

Dies wird mit der folgenden Zeile durch die Anweisung a=7 nachgeholt. Bei dieser Anweisung handelt es sich um eine sogenannte *Zuweisung*. Es wird ein Name a erklärt und diesem Namen – genannt Variable – wird ein konstanter Wert 7 zugewiesen. Wir wollen uns vorläufig vorstellen, dass die Variable a von nun an stellvertretend für die Zahl 7 steht. Genauer gesagt, bildet die Variable einen „Behälter", der den Wert 7 aufgenommen hat. Etwas Vergleichbares geschieht in den folgenden beiden Zeilen. Dort wird zunächst eine Variable b erklärt und erhält den Wert 3 zugewiesen. Schließlich wird eine Variable mit Namen c erzeugt. Diese Variable beinhaltet das Ergebnis der Multiplikation von a mit b. Damit dieser Wert am Bildschirm ausgegeben wird, verwenden wir eine eingebaute Funktion von Python: die print()-Funktion. Diese Funktion gibt den Text „c = " in der Python-Shell aus, gefolgt von dem Inhalt der Variablen c. Zusammen wird das so geschrieben:

```
print("c = ",c)
```

Schließlich sehen wir an den letzten beiden Zeilen, dass der Name a nun bekannt ist, nachdem ihm ein Wert zugewiesen wurde. Gibt man einfach nur den Buchstaben a ein, antwortet Python mit dem Inhalt der Variable, d.h. mit dem Wert, der dieser Variablen zugewiesen wurde. Ebenso hätten wir den Inhalt von c ausgeben können.

Variablen (sie werden auch Bezeichner) genannt, können beliebig viele Zeichen umfassen. Die Regeln zur Bildung solcher Namen sind in Kurzform:

- Namen für Variablen können aus Buchstaben, Ziffern (0…9) und einem einzigen Sonderzeichen bestehen. Dieses Sonderzeichen ist der Unterstrich „_" (engl. underscore).
- Das erste Zeichen in einem Variablennamen darf *keine Ziffer* sein. Ein Unterstrich ist allerdings als erstes Zeichen erlaubt.
- Deutsche Umlaute (Ä, ä usw.) dürfen in Python 3.x verwendet werden. In Python 2.x sind diese Zeichen *nicht* erlaubt.
- Es wird zwischen Groß- und Kleinschreibung unterschieden.
- Variablennamen dürfen *nicht* mit den reservierten Worten der Programmiersprache Python übereinstimmen.

Es wird empfohlen, möglichst selbsterklärende Namen für die Variablen zu erfinden, die einen Zusammenhang mit der durch das Programm zu lösenden Problematik schon im Namen ausdrücken. Also möglichst nicht „a", „b" und „c" wie im vorangegangenen Beispiel, sondern „Zylinder_Durchmesser", „StartZeit", „Gesamt_Summe" usw. Damit werden die Programme verständlicher und besser lesbar.

1.3.7 Zeichenketten (Strings)

Auch im technisch-wissenschaftlichen Bereich müssen oft Programme geschrieben werden, die Texte verarbeiten. Eine sogenannte Zeichenkette (engl. string) besteht aus beliebigen Zeichen und wird in Anführungszeichen gesetzt. Das folgende Beispiel zeigt, dass auch für solche Zeichenketten der „+“-Operator existiert. Dieser ist also *kontextsensitiv* und führt etwas anderes durch, wenn seine Operanden Zeichenketten statt Zahlen sind. In diesem Fall werden die Zeichenketten aneinander gehängt (engl. concatenation). In einem String dürfen Umlaute auch verwendet werden.

```
>>> Erster_Name="Bergische "
>>> Zweiter_Name="Universität"
>>> Name = Erster_Name + Zweiter_Name
>>> print(Name)
Bergische Universität
```

1.3.8 Turtle-Grafik

Python beinhaltet ein Modul zur Erstellung von einfachen Liniengrafiken. Die Programmierung funktioniert nach dem „Schildkrötenprinzip“, d.h. der Programmierer steuert ein Symbol – Schildkröte genannt – mit einfachen Befehlen innerhalb eines Grafikfensters. Durch die Bewegung der Schildkröte (engl. turtle) wird dann die Grafik erzeugt. Die Turtle-Grafik ist ein gutes Hilfsmittel zum Erlernen der Programmierung. Wir werden allerdings im Rahmen dieses Buches auch anspruchsvollere Grafiken mit diesem Tool erzeugen.

Im Prinzip genügen zunächst elf Befehle (besser gesagt: Funktionsaufrufe), um mit der Schildkröte erste interessante Grafiken erzeugen zu können.

- forward(steps): Mit diesem Befehl kriecht die Schildkröte in ihrer Blickrichtung vorwärts. Sie geht dabei um so viele Schritte bzw. *Pixel* nach vorn, wie der Parameter *steps* vorgibt. fd(steps) ist die Abkürzung dieses Befehls.

- left(angle): Mit diesem Befehl ändert die Schildkröte ihre Richtung. Sie dreht sich um den Winkel *angle* nach links. Der Winkel wird dabei in *Grad* angegeben. lt(angle) ist eine Abkürzung des Befehls „left“.

- right(angle): Mit diesem Befehl ändert die Schildkröte ihre Richtung. Sie dreht sich um den Winkel *angle* nach rechts. Der Winkel wird dabei auch in *Grad* angegeben. rt(angle) ist eine Abkürzung dieses Befehls.

- color(col): Mit diesem Befehl kann die Farbe der Linien, welche die Schildkröte zeichnet, beeinflusst werden. Die Variable col kann die Werte „red“, „green“, „blue“, „yellow“ etc. annehmen. Statt der Anführungszeichen können auch Apostrophe verwendet werden (z.B. color(’red’)).

- pensize(pixel): Mit diesem Befehl wird die Dicke der zu zeichnenden Linien in Pixel gesteuert.