1 Einführung

In diesem Kapitel wird die Programmiersprache Python vorgestellt. Nach Bemerkungen zur Installation dieser Sprache wird gezeigt, wie Python interaktiv ausgeführt werden kann. Schließlich wird dargestellt, wie ein vollständiges Python-Programm geschrieben wird. Neben dem Umgang mit Zahlen und Ausdrücken wird insbesondere auf das Konzept der Variablen eingegangen.

1.1 Die Programmiersprache Python

Die Sprache Python wurde in den neunziger Jahren des letzten Jahrhunderts von *Guido van Rossum* entwickelt. Mittlerweile ist Python eine der meistgenutzten Programmiersprachen. Die wesentlichen Vorzüge dieser Sprache sind:

- Python ist *leicht zu erlernen*, unterstützt mehrere Programmierparadigmen und ist klar strukturiert.
- Python eignet sich insbesondere zur *schnellen Entwicklung* von Softwareprototypen (RAD – Rapid Application Development). Dies ist gerade für Anwender in technisch-naturwissenschaftlichen Bereichen ein wichtiger Aspekt.
- Python ist *portabel*. Die Programme, die mit dieser Sprache geschrieben werden, laufen im Allgemeinen ohne Änderungen auf LINUX-, MAC OS- und Windows-Betriebssystemen.
- Python beinhaltet eine *Fülle von Anwendungspaketen* für unterschiedliche Bereiche. Ob grafische Darstellungen, numerische Berechnungen, Datenbanken oder Webanwendungen zu entwickeln sind: Der Anwender hat zumeist nur das Problem, aus der Vielzahl von angebotenen Lösungen, die für ihn geeignete zu finden. Es gibt kaum einen Anwendungsbereich, für den Python nicht schon vorgefertigte „Tools“ zur Verfügung stellt.
- Python ist *kostenlos*. Die Sprache kann auch für kommerzielle Zwecke kostenfrei genutzt werden.

- Python kann *leicht erweitert* oder selbst in Programme anderer Sprachen „eingebettet“ werden. Ein etwas fortgeschrittener Python-Programmierer ist in der Lage, eigene Anwendungspakete zu entwickeln.
- Mit Python können Programme geschrieben werden. Es ist aber auch möglich, einzelne Anweisungen interaktiv in der sogenannten Python-Shell auszuführen. Diese *interaktive Ausführung* hilft beim Ausprobieren von Sprachstrukturen und auch beim Testen von Programmen.

Im Rahmen dieses Buches steht die *leichte Erlernbarkeit* dieser Programmiersprache im Vordergrund.

1.2 Hinweise zur Installation

In diesem Buch wird die Installation der Software *nicht* gezeigt. Für Nutzer des MS-Windows-Betriebssystems gibt es allerdings ein Zusatzdokument im PDF-Format, das die Installation Schritt für Schritt darstellt. Dieses Dokument kann auf den Webseiten:

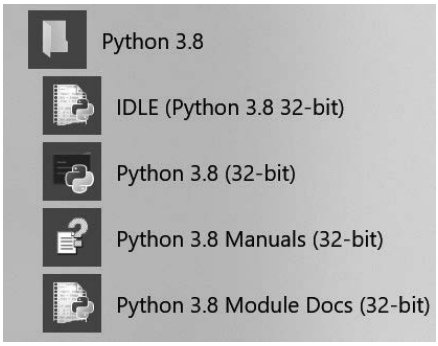
http://www.woyand.eu/python_buch/ bzw. *plus.hanser-fachbuch.de*

heruntergeladen werden. Zusammen mit der Anleitung kann der Leser dort auch ein ZIP-Verzeichnis mit einigen Installationsdateien sowie die Lösungen der Aufgaben und den Programmcode der Beispiele erhalten.

Eine Darstellung der Installation für alle gängigen Betriebssysteme (LINUX, Mac OS) würde den Rahmen des Buchs sprengen. Auf LINUX-Systemen ist Python oftmals schon vorinstalliert. Alle Installationsdateien sind auf der folgenden Python-Homepage unter dem Menüpunkt „Download“ zu finden:

http://www.python.org

Die Sprachversion, die in diesem Buch verwendet wird ist Python 3. Die Beispiele und Aufgaben wurden mit der relativ neuen Version 3.7.6 getestet.



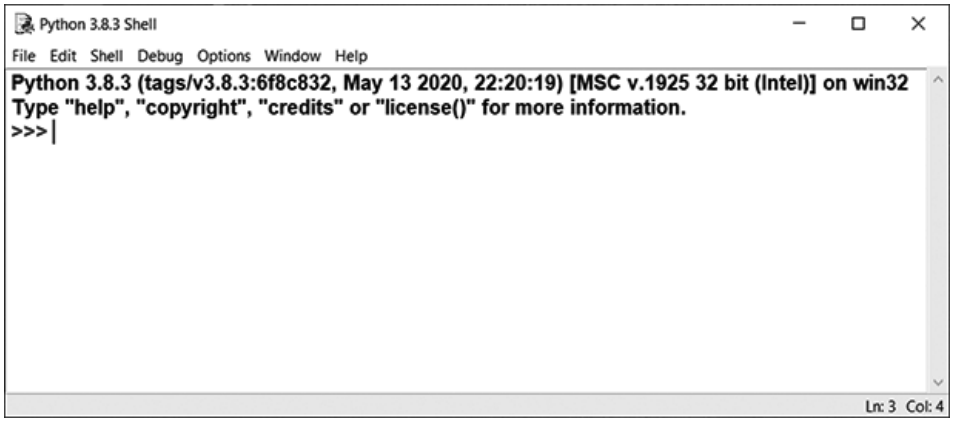
Nach erfolgreicher Installation kann Python gestartet werden. Hierzu wird über das Windows-Startmenü der Eintrag „Alle Programme“ ausgewählt. Um mit Python zu arbeiten, wählen Sie wie im Bild rechts beispielhaft gezeigt einfach „Python 3.8“ und dann den Eintrag „IDLE …“ aus.

IDLE ist der Name der *Standard-Entwicklungsumgebung* für Python. Um einen Brief zu schreiben, kann beispielsweise Microsoft Word zur Eingabe verwendet werden. Soll dagegen ein Python-Programm geschrieben werden, so gibt man dieses Programm mithilfe von *IDLE* ein und kann es anschließend starten. Für den schnellen Zugriff auf IDLE kann eine Verknüpfung auf dem Desktop abgelegt werden.

■ 1.3 Erste Schritte – der Python-Interpreter

Wenn IDLE aufgerufen wird – wie im letzten Abschnitt gezeigt – so wird entweder das Editorfenster wie unten gezeigt oder das Shell-Fenster geöffnet. Welches Fenster geöffnet wird, hängt von den Voreinstellungen der Software ab.

Für den Fall, dass das Editorfenster geöffnet wurde, wählen wir aus dem Menü die Kommandofolge „Run → Python Shell“ aus. Daraufhin öffnet sich die „Python-Shell“ (siehe Bild). In den folgenden Unterkapiteln wird zunächst diese Shell verwendet.



Hinweis

Je nach Software-Version kann sich nach dem Aufruf von IDLE zunächst das Editorfenster oder die Python-Shell öffnen. Welches der Fenster nach dem Aufruf von IDLE geöffnet wird, kann über eine Einstellung im Optionen-Menü festgelegt werden. Wählen Sie hierzu in IDLE „Options“ und dann „Configure IDLE“. Unter der Registerkarte „General“ kann unter dem Eintrag „At startup“ die Option „Open Edit Window“ oder „Open Shell Window“ gewählt werden.

Die drei aufeinanderfolgenden Zeichen „>>>“ werden *Eingabeaufforderung* oder *Eingabeprompt* genannt. Sie können nun damit beginnen, Ausdrücke in der Sprache Python einzugeben und diese unmittelbar ausführen zu lassen. Dieser interaktive Modus der Programm-Eingabe und -Ausführung ist typisch für sogenannte Interpreter-Sprachen wie Python. Gerade Programmieranfänger profitieren viel davon, weil sie einzelne Befehle unmittelbar ausprobieren können.

Wir beginnen nun mit einer kleinen Rundtour durch Python. Wenn wir etwas falsch machen, weil wir beispielsweise gegen die Regeln der Sprache verstoßen, so erhalten wir eine Fehlermeldung, die vom System in *roter Farbe* ausgegeben wird.

1.3.1 Addition und Subtraktion

Im Folgenden wird nicht mehr das ganze Fenster, sondern nur noch der Eingabeprompt und die darauffolgende Antwort des Computers dargestellt. Probieren Sie am besten die nachfolgenden Eingaben selbst aus!

```
>>> 1+2
3
>>> 1-2
-1
```

Wie wir sehen, wird nach jeder unserer Eingaben die Antwort von Python ausgegeben. Der Operator + bedeutet dabei Addition, der Operator - Subtraktion.

1.3.2 Multiplikation und Division

Die folgenden Eingaben zeigen, dass mit den Zeichen * eine Multiplikation und mit / eine Division durchgeführt werden kann.

```
>>> 1+2*3
7
>>> (1+2)*3
9
```

```
>>> 2+7/5
3.4
>>> 1/2
0.5
```

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 sogenannte *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 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)
0.89399666360055785
>>> 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.
- penup() und pendown() hebt und senkt den „Zeichenstift“ der Schildkröte.
- reset(): Dieser Befehl löscht den Inhalt des Grafikfensters und setzt die Schildkröte auf ihren Anfangszustand zurück (Ausrichtung nach rechts).
- goto(x,y): Mit diesem Befehl springt die Schildkröte auf die Position (x,y), wobei x und y in Pixel gemessen werden. Das Bezugskoordinatensystem befindet sich in der oberen rechten Ecke des Zeichenfensters. Die x-Achse ist horizontal ausgerichtet und die y-Achse zeigt senkrecht nach unten.
- bye(): Mit diesem Befehl wird das Turtle-Grafikfenster geschlossen.
- exitonclick(): Mit diesem Befehl wird das Grafikfenster geschlossen, wenn der Benutzer mit der Maus in das Fenster klickt.

Merke

Am Ende einer Turtle-Befehlsfolge sollte der Befehl bye() ausgeführt werden, sonst „hängt“ das Turtle-Grafikfenster unter dem Betriebssystem.

Am Anfang – dies ist so vereinbart – steht die Schildkröte mit Blickrichtung nach rechts. Wir wollen dies nun erproben. Hierzu muss zuerst das Modul „turtle“ importiert werden, damit die obigen Befehle zur Verfügung stehen. Dabei bedeutet das Sternchen *, dass *alle* Befehle aus dem Modul *turtle* importiert werden.