Hans-Bernhard
WOYAND

PYTHON
FÜR INGENIEURE UND NATURWISSENSCHAFTLER

Einführung in die Programmierung, mathematische Anwendungen und Visualisierungen

4., vollständig überarbeitete und erweiterte Auflage

Im Internet: Beispiele und Lösungen zu den Aufgaben

HANSER

Woyand

Python für Ingenieure und Naturwissenschaftler

Ihr Plus – digitale Zusatzinhalte!

Auf unserem Download-Portal finden Sie zu diesem Titel kostenloses Zusatzmaterial. Geben Sie dazu einfach diesen Code ein:

plus-wt9h3-e58lw

plus.hanser-fachbuch.de

Bleiben Sie auf dem Laufenden!

Unser **Computerbuch-Newsletter** informiert Sie monatlich über neue Bücher und Termine. Profitieren Sie auch von Gewinnspielen und exklusiven Leseproben. Gleich anmelden unter:

www.hanser-fachbuch.de/newsletter

Hans-Bernhard Woyand

Python für Ingenieure und Naturwissenschaftler

Einführung in die Programmierung, mathematische Anwendungen und Visualisierungen

4., vollständig überarbeitete und erweiterte Auflage

HANSER

Autor:

Prof. Dr.-Ing. Hans-Bernhard Woyand
Bergische Universität Wuppertal
Fakultät für Maschinenbau und Sicherheitstechnik

Bibliografische Information der Deutschen Nationalbibliothek:
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über
http://dnb.d-nb.de abrufbar.

Internet: www.hanser-fachbuch.de

Lektorat: Dipl.-Ing. Natalia Silakova-Herzberg
Herstellung: Anne Kurth
Covergestaltung: Max Kostopoulos
Coverkonzept: Marc Müller-Bremer, www.rebranding.de, München
Satz: Kösel Media GmbH, Krugzell
Druck und Bindung: CPI books GmbH, Leck
Printed in Germany

Print-ISBN 978-3-446-46483-4
E-Book-ISBN 978-3-446-46501-5
E-Pub-ISBN 978-3-446-46484-1

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

Außerdem sind die Daten zum Buch unter folgender Webadresse aufrufbar:

http://www.plus.hanser-fachbuch.de.

Die Zugangsdaten finden Sie auf Seite 1 des Buches.

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

Vorwort zur vierten Auflage

Für die vierte Auflage des Buches wurden einige kleinere, inhaltliche Verbesserungen vorgenommen. Zudem wurde ein neues Kapitel 11 mit dem Titel „Bildverarbeitung mit scikit-image“ hinzugefügt. Scikit-image ist ein Erweiterungspaket der Programmiersprache Python, das zur automatisierten Manipulation und Analyse von Bildern (Fotos, Grafiken) entwickelt wurde. Es wird gezeigt, wie Bilder eingelesen, gewandelt und geschrieben werden können. Weiterhin werden Algorithmen zum Auffinden von Kanten, Ecken und Kreisen sowie zum Abgleich von Vorlagen vorgestellt und erläutert. Zwei weitere Aufgaben und deren Lösungen wurden ergänzt.

Witten, im November 2020

Hans-Bernhard Woyand

Inhalt

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:

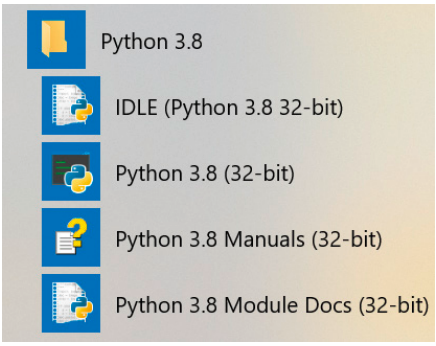
http://www.woyand.eu/python_buch/ bzw. *plus.hanser-fachbuch.de*

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.



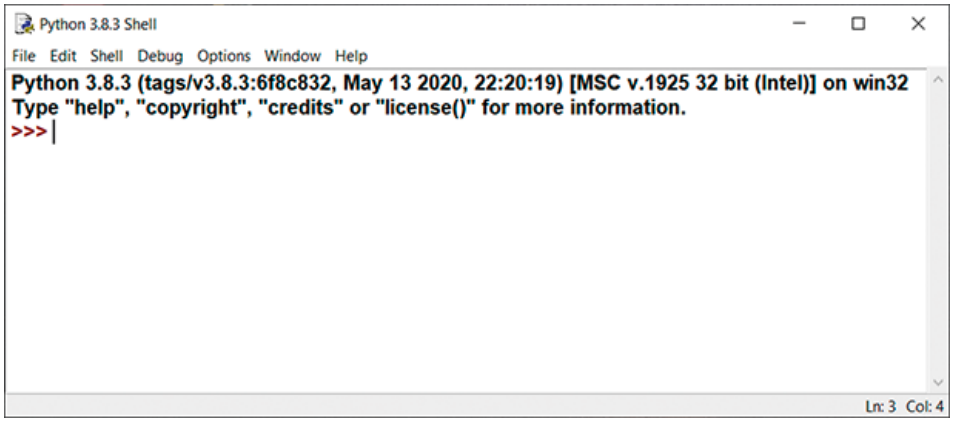
Nach erfolgreicher Installation kann Python gestartet werden. Hierzu wird über das Windows-Startmenü der Eintrag „Alle Programme“ ausgewählt. Um mit Python zu arbeiten, wählen Sie wie im Bild rechts beispielhaft gezeigt einfach „Python 3.8“ und dann den Eintrag „IDLE …“ aus.

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)
0.8939966636005578 5
>>> 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.

```
>>> from turtle import *
>>> reset()
>>> forward(100)
>>> left(90)
>>> forward(70)
>>> left(90)
>>> forward(100)
>>> left(90)
>>> forward(70)
```

Im Bild ist *links* der Ursprungszustand der Schildkröte zu sehen, nachdem die ersten beiden Zeilen ausgeführt wurden. Die Schildkröte erscheint auf dem Bildschirm als *Pfeilspitze* und ist in horizontaler Richtung ausgerichtet.

Mit dem Befehl forward(100) kriecht die Schildkröte um hundert Pixel nach rechts und erzeugt dabei eine horizontale Linie. Daraufhin wird mit dem Befehl left(90) die Schildkröte um 90 Grad nach links gedreht. Sie blickt jetzt nach oben. Im nächsten Schritt geht die Schildkröte dann um 70 Schritte vorwärts. Es entsteht eine vertikale Linie mit einer Länge von 70 Pixeln. Der nächste Befehl dreht die Schildkröte wieder um 90 Grad nach links. Sie steht nun wieder horizontal; aber nach links ausgerichtet usw.

Rechts ist das Endergebnis zu sehen. Es wurde ein Rechteck gezeichnet. Zum Schluss behält die Schildkröte ihre Ausrichtung bei. Sie blickt jetzt nach unten. Zum Abschluss des Dialogs geben wir bye() ein, um das Grafikfenster wieder zu schließen:

```
>>> bye()
```

Jeder Befehl benötigt einige Zeit zu seiner Ausführung. Die Schildkröte bewegt sich deshalb so langsam über das Grafikfenster, damit der Programmierer genau die Auswirkungen seiner Befehle beobachten kann.

■ 1.4 Python-Programme mit IDLE erstellen

Alle Programmieranweisungen, die wir bis jetzt eingegeben haben, wurden Zeile für Zeile in die „Python-Shell“ eingegeben und dann von Python unmittelbar ausgeführt. Dies ist sehr bequem und Programmieranfänger können damit auf einfache Art ihre ersten Gehversuche machen.

Allerdings sind mit dieser Vorgehensweise auch einige Nachteile verbunden. Wenn wir die Python-Shell beenden, so sind alle Eingaben verloren. Und wenn wir Varianten ausprobieren wollen, so müssen die Programmieranweisungen immer wieder von neuem eingegeben werden. Deshalb soll jetzt eine zweite Vorgehensweise vorgestellt werden: das Schreiben von Python-Programmen. Diese Programme werden auch „Python-Scripts“ genannt. In den folgenden Kapiteln werden wir viel häufiger diese zweite Technik verwenden. Solche Python-Programme werden auf dem Computer dauerhaft gespeichert und können beliebig oft ausgeführt werden. Weiterhin können sie kopiert und anschließend abgeändert werden.

Beispiel: Umrechnung von Temperaturen

Im Folgenden soll Schritt für Schritt ein erstes einfaches Programm erstellt werden. Die Problemstellung, die wir mit diesem Programm lösen wollen, ist die Berechnung von Temperaturen. In Mitteleuropa werden Temperaturen im Allgemeinen in Grad Celsius angegeben. Bei 0 Grad Celsius gefriert das Wasser, der Siedepunkt liegt bei 100 Grad. In Großbritannien und den USA werden Temperaturen aber zumeist in Grad Fahrenheit angegeben. Die Umrechnung von Grad Celsius in Grad Fahrenheit kann als einfache Formel angegeben werden:

Fahrenheit = 9/5 * Celsius + 32.

Wasser gefriert also bei 32 Grad Fahrenheit und siedet bei 212 Grad Fahrenheit.

Wir wollen zur Umrechnung der Temperaturen von Grad Celsius in Grad Fahrenheit ein Computerprogramm in Python schreiben, das mit dem Benutzer einen Dialog führen soll. Am besten ist es, wenn nach dem Starten des Programms zunächst eine Art Überschrift erscheint, die etwas darüber aussagt, was das Programm zu tun beabsichtigt. Diese Überschrift soll sein: „Umrechnung der Temperaturen von Celsius in Fahrenheit“. Der Vorteil einer solchen Überschrift ist, dass ein anderer Benutzer sofort erkennt, wozu das Programm gut sein soll.

Weiterhin stellen wir uns vor, dass das Programm den Benutzer auffordern soll, eine Temperatur in Celsius einzugeben. Daraufhin gibt der Benutzer eine Temperatur ein. Das Programm wartet solange, bis die Benutzereingabe erfolgt ist und der Anwender die Return-Taste gedrückt hat. Anschließend fährt das Programm fort, berechnet aus der Benutzereingabe die entsprechende Temperatur in Fahrenheit und gibt diese dann – zusammen mit der Benutzereingabe – aus. Der Dialog mit dem Anwender könnte also folgendermaßen aussehen:

```
Umrechnung der Temperaturen von Celsius in Fahrenheit
------------------------------------------------------------------------

Bitte geben Sie eine Temperatur in Celsius ein: 100

Sie haben 100 Grad Celsius eingegeben.

Diese Temperatur entspricht 212 Grad Fahrenheit.
```

Die fett gesetzte Zahl 100 ist die *Benutzereingabe*. Das Programm soll solange warten, bis *irgendeine* Zahl an dieser Stelle eingegeben wird.

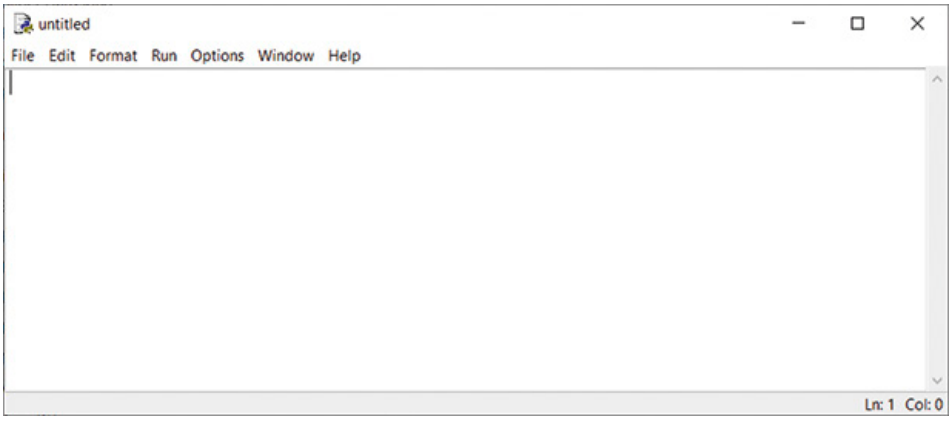
An diesem Beispiel demonstrieren wir eine Grundstruktur vieler Programme (nicht nur in der Programmiersprache Python), die wir folgendermaßen symbolisieren können:

Eingabe > *Verarbeitung* > *Ausgabe*

Die Eingabe ist in dem betrachteten Fall eine Temperatur in Grad Celsius. Nach der Eingabe wird die Umrechnung vorgenommen und anschließend das Ergebnis in Fahrenheit ausgegeben. Nach diesem Schema arbeiten viele Programme. Es wird auch das EVA-Prinzip genannt. EVA ist ein Akronym für *E*ingabe, *V*erarbeitung und *A*usgabe.

Nachdem wir die Aufgabenstellung klar definiert haben, beginnen wir nun mit der Erstellung des Programms. Wir rufen die Entwicklungsumgebung IDLE – wie in Abschnitt 1.2 gezeigt – auf.

Daraufhin öffnet sich das Editorfenster, in welches das Programm eingegeben werden kann. Das Fenster erhält den gleichen Namen wie das Programm. Da wir noch keinen Namen für das Programm eingegeben haben, heißt es „Untitled“ (engl. unbenannt).



Wir beginnen nun damit, den folgenden Programmtext einzutippen. Die hierzu notwendigen Befehle bzw. Programmanweisungen werden im Folgenden ausführlich besprochen.

```
# ------------------------------------------------                    #1
# Programm zur Umrechnung der
# Temperaturen von Grad Celsius
# in Grad Fahrenheit
#
# Autor:   H.-B. Woyand
# Datum: 5.1.2008
# ------------------------------------------------                    #2
```

```
print("Umrechnung der Temperaturen von Celsius in Fahrenheit")          #3
print("-------------------------------------------------------------")  #4
print()                                                                 #5
Celsius = input("Bitte geben Sie eine Temperatur in Celsius ein: "      #6
Celsius = float(Celsius)
Fahrenheit = 9/5 * Celsius + 32                                         #7

print()
print("Sie haben %f Grad Celsius eingegeben." % Celsius)                #8
print()
print("Diese Temperatur entspricht %f Grad Fahrenheit." % Fahrenheit)   #9
```

Hinweis

Am Ende einiger Zeilen sehen Sie Markierungen (zum Beispiel #1, #9 usw.). Diese Markierungen dienen einzig und allein dazu, Sie im folgenden Text auf bestimmte Zeilen hinzuweisen. Es handelt sich bei diesen Markierungen um sogenannte Kommentare. Diese beeinflussen nicht die Funktionsweise des Programms und können deshalb bei der Eingabe auch weggelassen werden.

Sehen wir uns nun die Grundelemente dieses Programms genauer an:

- **Zeilen #1 bis #2:** Jedes Programm sollte *Kommentare* enthalten. Diese Kommentare beeinflussen *nicht* die Programmausführung. Für die Funktionsweise des Programms sind sie eigentlich völlig überflüssig. Diese Kommentare werden mit den anderen Programmanweisungen gespeichert und nennen beispielsweise den Autor des Programms sowie Urheberrechtshinweise (Copyright), das Erstellungsdatum, die Programmversion, den Zweck des Programms, Hinweise zur Bedienung des Programms usw. Kommentare können an beliebiger Stelle im Programm stehen und werden mit dem Zeichen „#“ eröffnet. An diesem Zeichen erkennt Python, dass nun ein Kommentar folgt. Alles, was nach dem Zeichen „#“ bis zum Ende der Zeile steht, wird als Kommentar betrachtet.
- **Zeile #3:** Mit der print()-Funktion wird die Überschrift „Umrechnung der Temperaturen von Celsius in Fahrenheit“ auf den Bildschirm (genauer in die Python-Shell) geschrieben.
- **Zeile #4:** Eine Unterstreichung der Überschrift durch eine Folge von Minuszeichen wird ebenfalls durch Ausgabe mit print() erreicht.
- **Zeile #5:** Damit der folgende Text etwas Abstand zur Überschrift hat, wird die print-Funktion ohne Argumente aufgerufen. Dies bewirkt die Ausgabe einer Leerzeile in die Python-Shell.
- **Zeile #6:** Die input()-Funktion dient dazu, eine Eingabe durch den Benutzer des Programms zu ermöglichen. Wird input() aufgerufen, dann erscheint zunächst der Text auf dem Bildschirm, der dieser Funktion in den Klammern

übergeben wird. Das Programm stoppt dann und wartet, bis der Benutzer etwas eingetippt und die Return-Taste gedrückt hat. Diese Eingabe wird dann der Variablen Celsius zugewiesen. Wir stellen uns Celsius als einen Behälter vor, der nun eine Zeichenkette enthält, die der Benutzer des Programms eingegeben hat. In der Folgezeile wird diese Zeichenkette durch Aufruf der Funktion float() in eine Gleitpunktzahl umgewandelt.

- **Zeile #7:** Für die Anweisung Fahrenheit = 9/5 * Celsius + 32 wird zunächst die rechte Seite ausgewertet. Der Wert der Variablen Celsius wird dazu verwendet, um den Ausdruck „9/5 * Celsius + 32“ zu berechnen.
  Dieser arithmetische Ausdruck wird von links nach rechts ausgeführt. Ist das Ergebnis dieses Ausdrucks ermittelt, so wird es der Variablen (dem Namen) Fahrenheit zugewiesen. Wie Sie sehen, wird damit durch das Gleichheitszeichen etwas anderes ausgedrückt, als bei vergleichbaren Formeln der Mathematik. Im mathematischen Kontext könnte diese Zeile als Gleichung verstanden werden. In der Programmierung dagegen bedeutet das Gleichheitszeichen eine Zuweisung. Dabei muss auf der linken Seite immer ein Variablenname stehen. Die rechte Seite der Zuweisung wird – wenn es sich um einen komplexeren Ausdruck handelt – zunächst ausgewertet und das Ergebnis der Variablen zugeordnet.
- **Zeile #8 und #9:** Durch diese beiden Zeilen werden die Werte der Variablen Celsius und Fahrenheit auf dem Bildschirm ausgegeben. Hierbei hilft uns wieder die print()-Funktion. Wie wir gelernt haben, wird mit dieser Funktion der in Hochkommata stehende Text ausgegeben. In diesem Text finden Sie ein sogenanntes Formatelement %f vor, das als Platzhalter für einen auszugebenden Gleitpunktzahlenwert fungiert. Die auszugebende Variable wird an die Zeichenkette durch ein „%“-Zeichen angehängt. Deren Wert wird bei der Ausgabe statt des Platzhalters in die Zeichenkette eingefügt. Es ist ein guter Programmierstil, die Werte, die der Benutzer eingetippt hat, nochmals durch das Programm ausgeben zu lassen. Damit ist der Anwender sicher, dass die von ihm getätigten Eingaben auch „angekommen“ sind. Zuletzt wird das eigentliche Ergebnis dieses kleinen Berechnungsprogramms ausgegeben: die Temperatur in Grad Fahrenheit.

Merke

Formatelemente sind Platzhalter, die bei der Ausgabe durch den Wert jener Variablen ersetzt werden, die an die print()-Funktion durch das Zeichen „%“ angehängt werden. Das Zeichen „%f“ ist ein Platzhalter für Gleitpunktzahlen (Dezimalzahlen).

Wichtig!

Wir müssen deutlich zwischen Gleichungen in der Mathematik und Zuweisungen in der Programmierung unterscheiden! Ein Ausdruck der Form c = a + b meint in der Mathematik eine Gleichung und kann nach den Gesetzen der Algebra umgestellt werden. In der Programmierung bedeutet ein Ausdruck der Form c = a + b eine Zuweisung. Diese kann so beschrieben werden: berechne den Wert der rechten Seite (a+b) und lege das Ergebnis unter dem Namen c ab. Dies bedeutet wiederum, dass auf der linken Seite immer ein Variablenname stehen muss!

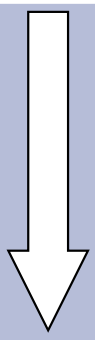
Nachfolgend ist das Programm nochmals ohne die Kommentare abgebildet. Der Pfeil auf der rechten Seite der Programmanweisungen soll darauf hinweisen, dass das Programm *von oben nach unten abgearbeitet* wird.

Python verarbeitet also das Programm so wie wir es lesen: von oben nach unten. Dies wird auch *sequenzielle Programmabarbeitung* oder kurz *Sequenz* genannt. Wie wir später sehen werden, ist diese sequenzielle Verarbeitung nicht die einzige Möglichkeit der Programmabarbeitung. Es gibt zwei weitere Grundstrukturen in der Programmierung: die Wiederholung und die Programmverzweigung. Bei der Wiederholung – auch Iteration genannt – werden bestimmte Programmteile mehrfach durchlaufen. Programmverzweigungen gestatten es, Programmteile nur dann abzuarbeiten, wenn bestimmte Bedingungen vorliegen.

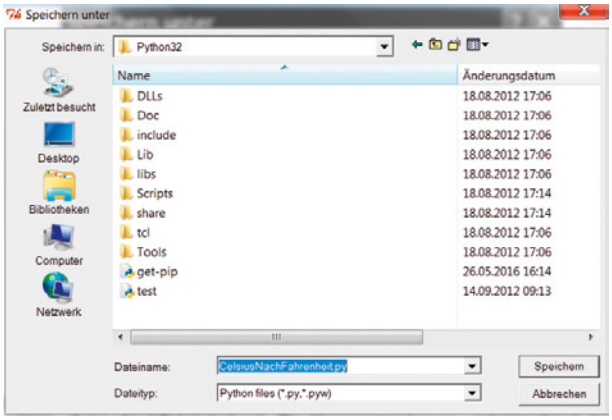
```
print("Umrechnung der Temperaturen von Celsius in Fahrenheit")
print("-----------------------------------------------------")
print()

Celsius = input("Bitte geben Sie eine Temperatur in Celsius ein: ")
Celsius = float(Celsius)
Fahrenheit = 9/5 * Celsius + 32

print()
print("Sie haben %f Grad Celsius eingegeben." % Celsius)
print()
print("Diese Temperatur entspricht %f Grad Fahrenheit." % Fahrenheit
```



Abschließend wollen wir unser erstes Python-Programm abspeichern, um es langfristig aufzubewahren. Hierzu verwenden wir das Hauptmenü unseres Eingabefensters. Dort wählen wir den Menüpunkt „File“ und dann „Save As …“. Daraufhin öffnet sich das folgende Fenster „Speichern unter“:



Dort geben wir in das Eingabefeld „Dateiname“ den Namen des Programms ein. Wir wählen „CelsiusNachFahrenheit.py“. Die *Namenserweiterung* „.py“ signalisiert dem Computer, dass es sich bei der gespeicherten Datei um ein *Python-Programm* handelt. Anschließend betätigen wir den Button „Speichern“. Damit haben wir unser erstes Programm erfolgreich abgespeichert.

Zuletzt kommt der wichtigste Schritt. Wir starten jetzt das Programm, in dem wir im Hauptmenü von IDLE den Menüpunkt „Run“ und dann „Run Module“ anwählen. Alternativ kann das Programm auch durch Drücken der F5-Taste gestartet werden. Die Ausgaben des Programms erscheinen in der Python-Shell. Dort wird auch die Eingabeaufforderung sichtbar, damit Sie den Zahlenwert eingeben können.

1.5 Aufgaben

Aufgabe 1.1

Schreiben Sie ein Python-Programm, das vom Benutzer einen Temperaturwert (Gleitpunktzahl) in Grad Fahrenheit anfordert. Dieser Temperaturwert soll dann in Grad Celsius umgerechnet und ausgegeben werden. Benennen Sie das Programm „FahrenheitNachCelsius.py“ und testen Sie das Programm.

Aufgabe 1.2

Schreiben Sie ein Python-Programm, das vom Benutzer einen Winkel in Grad anfordert und anschließend den Sinus- und den Kosinuswert zu diesem Winkel berechnet. Anschließend soll der Winkel und die beiden berechneten Werte auf dem Bildschirm ausgegeben werden. Testen Sie das Programm.

Aufgabe 1.3

Schreiben Sie ein Python-Programm, das vom Benutzer einen Zahlenwert x (Gleitpunktzahl) anfordert. Für diesen Zahlenwert soll dann die mathematische Funktion

$$y = \frac{1}{2}x^3 - \frac{1}{2}x^2 + 2x + 5$$

berechnet werden. Anschließend soll das Programm die Werte x und y auf dem Bildschirm ausgeben. Testen Sie das Programm.

Aufgabe 1.4

Verwenden Sie den Eingabeprompt der Entwicklungsumgebung (Python-Shell), um den folgenden – in Umgangssprache vorgegebenen – logischen Ausdruck auf Wahrheit bzw. Falschheit zu testen:

((8 kleiner 5) oder (4 ungleich 7)) und (nicht (4 gleich 5))

Erklären Sie, wie das Ergebnis zustande kommt.

Aufgabe 1.5

Schreiben Sie ein Python-Programm, das den Benutzer zur Eingabe seines Namens auffordert. Verwenden Sie für die Eingabe des Namens die input()-Funktion. Anschließend soll ein Willkommenstext auf dem Bildschirm ausgegeben werden, in dem der Name des Anwenders erscheint.

Hinweis: Zur Ausgabe von Zeichenketten durch die print()-Funktion dient das Formatelement %s. Ein typischer Dialog könnte etwa so aussehen:

```
Bitte geben Sie Ihren Namen ein: Frau Beate Schmidt
Wir begrüßen Frau Beate Schmidt in unserem Programmierkurs!
```

Aufgabe 1.6

Schreiben Sie ein Python-Programm, das ein gleichseitiges Dreieck, ein Quadrat, ein gleichseitiges Fünfeck und ein gleichseitiges Sechseck mit der Turtle-Grafik erzeugt. Die Seitenlänge der Figuren soll dabei immer 100 Pixel betragen. Wechseln Sie die Farben und Strichstärken zwischen den einzelnen Grafiken.

Aufgabe 1.7

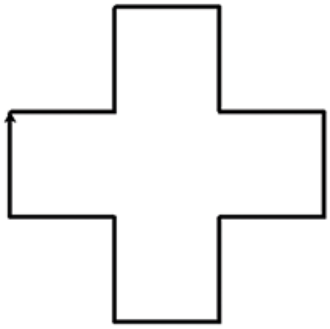
Entscheiden Sie, welche der folgenden Worte zulässige Variablennamen in der Programmiersprache Python sind. Begründen Sie Ihre Entscheidung! Beachten Sie hierbei auch die Tabelle der reservierten Worte (siehe Kapitel 2).

Hinweis: Sie können die Namen auch im Interpreter-Modus verwenden, indem Sie einfache Zuweisungen der Form *while = 100* ausprobieren. Dann entscheidet Python für Sie, ob der Name zulässig ist oder nicht!

4menWagen, while, While, while100, _while, $wert, whilst, ___, 3SterneHotel, HotelMit3Sternen, X, AnzahlWerte, AnzahlDerWerte, Anzahl_Werte, _12345

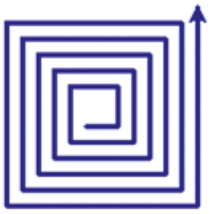
Aufgabe 1.8

Schreiben Sie mithilfe der Turtle-Grafik ein Python-Programm, das die Grafik rechts erzeugt. Die Geraden sind 100 Pixel lang und die Linienstärke soll 3 Pixel betragen. Benutzen Sie bei der Lösung der Aufgabe auch die Kurzbefehle der Turtle-Grafik.



Aufgabe 1.9

Schreiben Sie mithilfe der Turtle-Grafik ein Python-Programm, das die nebenstehende Grafik erzeugt. Die erste Linie soll 20 Pixel lang sein. Jede folgende Linie soll 5 Pixel länger sein. Die Striche sollen drei Pixel dick und von blauer Farbe sein.



Aufgabe 1.10

Schreiben Sie mithilfe der Turtle-Grafik ein Python-Programm, das die Grafik rechts erzeugt. Die Linien sollen 25 Pixel lang sein. Die Strichstärke soll drei Pixel betragen. Die zu erzeugende Grafik soll rot sein.

Aufgabe 1.11

Schreiben Sie ein Programm, das vom Benutzer einen Druckwert in der Einheit Pascal (Pa) anfordert. Anschließend soll das Programm diesen Druck in die folgenden Druckeinheiten umrechnen (siehe Tabelle) und diese anschließend ausgeben.

Druckeinheit	Umrechnungsfaktor (1 Pa entspricht)
Bar	1,0 10^{-5}
Technische Atmosphäre (at)	1,0197 10^{-5}
Physikalische Atmosphäre (atm)	9,8692 10^{-6}
Torr	7,5006 10^{-3}
Pounds per square inch (psi) „Pfund-Kraft pro Quadratzoll“	1,4504 10^{-4}

Hinweis: Zur Lösung dieser Aufgabe sollen Sie die sogenannte Exponentendarstellung für Dezimalzahlen (Gleitpunktzahlen) verwenden. Zum Beispiel gilt für die Zahl 1,0197 10^{-5} die Pythonschreibweise 1.0197e-5.

Aufgabe 1.12

Erstellen Sie ein Programm, das die Oberfläche O, die Mantelfläche M sowie das Volumen V eines Kreiszylinders berechnet. Der Kreiszylinder soll durch die Höhe H und den Radius R charakterisiert sein. Diese Größen soll der Benutzer des Programms über die Tastatur eingeben. Anschließend soll das Programm die Formeln

$$V = \pi R^2 H$$

$$M = 2\pi RH$$

$$O = 2\pi R(R + H)$$

berechnen und anschließend die Werte für O, M und V ausgeben. Gestalten Sie das Programm so, dass die Ausgabe gut verständlich ist.

Aufgabe 1.13

Vektoren in der Ebene werden durch die Zusammenfassung zweier Zahlen (x-Komponente und y-Komponente) definiert. Wir schreiben Vektoren in einer Zeilendarstellung und verwenden dabei das Zeichen T für die Transponierung, d. h. Vektoren werden eigentlich in Spaltenform geschrieben. Wir verwenden die folgende Darstellung in einer Zeile nur, um Platz zu sparen.

$\mathrm{a} = [\mathrm{x}, \mathrm{y}]^{\mathrm{T}}$ und $\mathrm{b} = [\mathrm{u}, \mathrm{v}]^{\mathrm{T}}$

sind zwei Vektoren. Dann ist das innere Produkt folgendermaßen definiert

$$\mathrm{a} \cdot \mathrm{b} = \mathrm{x} \cdot \mathrm{u} + \mathrm{y} \cdot \mathrm{v} = |\mathrm{a}| \cdot |\mathrm{b}| \cdot \cos(\phi)$$

wobei

$$|\mathrm{a}| = \sqrt{\mathrm{x}^2 + \mathrm{y}^2}$$

und

$$|\mathrm{b}| = \sqrt{\mathrm{u}^2 + \mathrm{v}^2}$$

die Beträge (Längen) der Vektoren sind. Das Symbol ϕ ist der Winkel, den die beiden Vektoren einschließen.

Erstellen Sie ein Python-Programm, das die Komponenten der Vektoren x, y und u, v einliest. Vergessen Sie nicht, sinnvolle Aufforderungstexte vorzusehen, damit ein neuer Benutzer auch weiß, was er eingeben soll! Nach der Eingabe sollen zunächst die Beträge der Vektoren und das innere Produkt berechnet werden. Über die Umstellung der obigen Formel erhält man

$$\cos(\phi) = (x \cdot u + y \cdot v) / (|a| \cdot |b|)$$

Aus dem Ergebnis kann der Winkel (im Bogenmaß) mittels Arcus-Cosinus-Funktion ermittelt werden.

Das Programm soll schließlich die Beträge der Vektoren, den Wert des inneren Produkts sowie den Winkel, den die Vektoren einschließen, im Bogenmaß und in Grad ausgeben.

Aufgabe 1.14

Schreiben Sie ein Programm, das einen horizontalen Pfeil mithilfe der Turtle-Grafik zeichnet. Der Pfeil soll in roter Farbe mit einer Liniendicke von 6 Pixeln erstellt werden.

Aufgabe 1.15

Zeichnen Sie mittels Turtle-Grafik die rechte Figur. Diese soll mit Linien der Stärke 60 Pixel (!) in blauer Farbe gezeichnet werden.

1.6 Lösungen

Sie finden die vollständigen Python-Programme auch im Download-Bereich zu diesem Buch.

Lösung 1.1

Für das folgende Programm kann das Beispiel aus Abschnitt 1.4 als Vorlage dienen. Zur Berechnung der Temperatur in Grad Celsius muss die Berechnungsformel umgestellt werden. Die Ein- und Ausgaben müssen entsprechend angepasst werden.