Veit **STEINKAMP**

Mechanik verstehen

MIT PYTHON

HANSER

Steinkamp

Mechanik verstehen mit Python

Veit Steinkamp

Mechanik verstehen mit Python

HANSER

Der Autor: Veit Steinkamp, Lübbecke

Print-ISBN: 978-3-446-48223-4
E-Book-ISBN: 978-3-446-48479-5

Die allgemein verwendeten Personenbezeichnungen gelten gleichermaßen für alle Geschlechter.

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

Vilshofener Straße 10 | 81679 München | info@hanser.de
www.hanser-fachbuch.de
Lektorat: Brigitte Bauer-Schiewek, Kristin Rothe
Herstellung: Gina Lada
Copy editing: Petra Kienle, Fürstenfeldbruck
Covergestaltung: Thomas West
Titelmotiv: © Max Kostopoulos, unter Verwendung von Grafiken von stock.adobe.com/Vitaly und shutterstock.com/Sudowoodo
Satz: Eberl & Koesel Studio, Kempten
Druck: Elanders Waiblingen GmbH, Waiblingen
Printed in Germany

Inhalt

1 Einführung

Physik ist neben der Mathematik ein wichtiges Grundlagenfach der Ingenieurwissenschaften. Ohne solides physikalisches und mathematisches Wissen kann ein Studium der Ingenieurwissenschaften nicht erfolgreich abgeschlossen werden. Dieses Wissen sollte eigentlich in den allgemeinbildenden Schulen vermittelt werden. Doch Lehrkräftemangel in den MINT-Fächern und fehlendes Problembewusstsein der Verantwortlichen stärken den negativen Trend defizitärer naturwissenschaftlicher Grundbildung. Für eine prosperierende und innovative Wirtschaftsentwicklung sind naturwissenschaftliche Kenntnisse der Akteure unerlässlich: Es braucht mehr qualifizierte Ingenieure und Naturwissenschaftler, die die Probleme von Gegenwart und Zukunft lösen können. Dieses Buch soll einen Beitrag dazu leisten, das Interesse an der Physik und Informatik zu wecken. Um den Lernumfang zu begrenzen, wird nur eine grundlegende Teildisziplin der Physik behandelt: die Mechanik.

1.1 Das Konzept

Es stellt sich zunächst die Frage, ob es überhaupt sinnvoll ist, eigene Programme zu schreiben, mit denen mechanische Gesetzmäßigkeiten simuliert werden können. Zahlreiche Simulationsprogramme unterstützen bereits die Lernenden bei ihren Lernprozessen. Sie verbergen aber ihre Arbeitsweise, die Lernenden wissen nicht, wie die Ergebnisse zustande gekommen sind. Wie wäre es, wenn sie selbst versuchen würden, eigene Simulationsprogramme zu schreiben? Der Lernaufwand wäre zunächst mit einer zusätzlichen Anstrengung verbunden. Er würde sich aber lohnen, weil man (fast nebenbei) eine Programmiersprache erlernen würde. Und was vielleicht noch stärker zu gewichten ist: Beim Programmieren entdeckt man eventuell Verständnisschwierigkeiten, die einem vorher so nicht bewusst waren. Eine Program-

miersprache, die sich besonders gut dazu eignet, physikalische Prozesse zu visualisieren, ist Python. Mithilfe der Erweiterungsmodule VPython und Matplotlib können mit relativ geringem Aufwand Animations- und Simulationsprogramme erstellt werden.

Fast jedes Kapitel beginnt mit einem interaktiven VPython-Programm. Wenn Sie dieses Programm starten, können Sie einzelne Parameter ändern und die Auswirkungen auf die Animation beobachten. Sie verschaffen sich so einen ersten Zugang zum Thema des jeweiligen Kapitels. Anschließend folgen Matplotlib-Programme, mit denen die Thematik weiter vertieft wird. Die Quelltexte der VPython-Programme werden nicht mit abgedruckt, Sie können diese kommentierten Quelltexte aber mit einer Python-Entwicklungsumgebung öffnen, analysieren und gegebenenfalls nach eigenen Wünschen ändern. Die übrigen Python-Quelltexte werden abgedruckt und besprochen.

Die Beispiele stammen zu einem großen Anteil aus dem Sport: Golf, Hammerwurf, Fußball, Radfahren usw. Die Matplotlib-Programme simulieren und animieren Flugbahnen, Schwingungen und Wellen. Differenzialgleichungen werden mit SciPy (Modul für numerische Berechnungen) und SymPy (Modul für symbolische Berechnungen) gelöst. Bevor ein Programm vorgestellt wird, werden die theoretischen Grundlagen kurz dargestellt. Die Programmanalyse erfolgt fallbezogen, es werden nur diejenigen Aspekte besprochen, die in dem Programm neu sind.

Wie können Sie dieses Buch nutzen? Sie können es kapitelweise durcharbeiten, die Programme testen und analysieren sowie die vorgeschlagenen Übungen ausführen. Sie können aber auch, je nach eigenem Interesse, einzelne Programme auswählen und starten, ohne den Quelltext genau verstanden zu haben. Die Programme eignen sich sowohl für individuelle Übungen als auch für Vorträge zum Einstieg in ein Thema.

Dieses Buch soll und kann kein Physiklehrbuch ersetzen. Es verfolgt ein didaktisches Ziel: Das Lernen physikalischer Zusammenhänge soll durch interaktive Visualisierung unterstützt werden. Gegebenenfalls sollten Sie zur fachlichen Ergänzung ein Physiklehrbuch ihrer Wahl heranziehen.

An welchen Leserkreis richtet sich dieses Buch? Das Buch richtet sich an alle Studentinnen und Studenten der Physik, des Maschinenbaus, der Mechatronik und Elektrotechnik. Auch Schülerinnen und Schüler, die sich für einen Leistungskurs Physik an einer gymnasialen Oberstufe entschieden haben, sollten sich angesprochen fühlen.

Den Code aller Beispiele aus dem Buch sowie die Musterlösungen der Übungsaufgaben finden Sie unter *https://drsteinkamp.de*.

1.2 Die Programmiersprache Python

Python ist eine universell einsetzbare Programmiersprache. Sie wurde Anfang der 1990er-Jahre von Guido van Rossum entwickelt. Die Namensgebung hat nichts mit der gleichnamigen Schlangengattung Python zu tun; sie bezieht sich auf die englische

Komikergruppe Monty Python. Laut PYPL-Index (Januar 2025) ist Python weltweit die beliebteste Programmiersprache.

1.2.1 Eigenschaften von Python

Die besonderen Eigenschaften von Python sind:

- Python-Quelltexte werden interpretiert, d. h., bei jedem neuen Programmstart wird das Programm Zeile für Zeile neu übersetzt. Die Ausführungszeit von Python-Programmen verlängert sich zwar dadurch, für Programmieranfänger sind interpretierende Programmiersprachen jedoch gegenüber Compilersprachen (C, C++, Pascal usw.) vorzuziehen, weil die Reaktionen des Interpreters auf Änderungen im Quelltext schneller erfolgen als bei einer Kompilierung. Die Interaktion zwischen Benutzer und einer Python-Entwicklungsumgebung gestaltet sich dadurch flexibler.
- Python unterstützt das imperative, das funktionale und das objektorientierte Programmierparadigma (Programmierstil).
- Python kann plattformübergreifend eingesetzt werden. Es läuft auf den Betriebssystemen Linux, macOS und Windows.
- Im Vergleich zu anderen Programmiersprachen sind Python-Programme wesentlich kompakter. Die Fehlersuche gestaltet sich deshalb einfacher und die Entwicklungszeiten verkürzen sich.
- Python kann durch Module (Programmbibliotheken, engl. *program library*) erweitert werden. Das gilt zwar auch für andere Programmiersprachen. In Python lassen sich diese Erweiterungsmodule jedoch besonders einfach in eigene Programme integrieren.

1.2.2 Erweiterungsmodule für Python

Die Funktionalität von Python lässt sich durch Module erheblich erweitern. In diesem Buch werden die Module NumPy (*numeric python*), Matplotlib, SciPy (*scientific python*), Numdifftools, SymPy (*symbolic python*) und VPython (*visual python*) benutzt, um Gesetzmäßigkeiten der Mechanik zu simulieren und zu visualisieren.

NumPy (*https://numpy.org*) stellt alle trigonometrischen Funktionen zur Verfügung. Es können *n*-dimensionale Arrays erzeugt werden, alle mathematischen Operationen auf Matrizen sind möglich, umfangreiche lineare Gleichungssysteme können gelöst werden.

Mit Matplotlib (*https://matplotlib.org*) können zwei- oder dreidimensionale Funktionsplots und Animationen erstellt werden. Verschiedene Varianten von Koordinatenachsen können aussagekräftig beschriftet werden. Mathematische Formeln können mit-

hilfe der LaTex-Notation auf der Zeichenfläche platziert werden. Auch interaktive Steuerelemente (*Slider*, *Button*, *Radio Button* usw.) sind verfügbar. Die Funktionsplots können als Datei in allen gebräuchlichen Grafikformaten auf eine Festplatte gespeichert werden.

Das Modul SciPy (*https://scipy.org*) stellt Algorithmen für Optimierung, Integration, Interpolation, Eigenwertprobleme, algebraische Gleichungen, Differentialgleichungen, Statistik und vieles mehr zur Verfügung.

Das Modul Numdifftools (*https://pypi.org/project/numdifftools*) ist ein Werkzeug für die numerische Differentiation. Es können Ableitung bis zur 10. Ordnung numerisch berechnet werden.

SymPy (*https://www.sympy.org/en/index.html*) ist ein Modul für symbolische Mathematik. Vereinfachung mathematischer Terme, symbolisches Differenzieren und Integrieren sind möglich. Gleichungen, Gleichungssysteme und lineare Differenzialgleichungen können symbolisch gelöst werden. Ziel ist es, ein voll funktionsfähiges Computeralgebra-System (CAS) zur Verfügung zu stellen.

Mit VPython (*https://vpython.org*) kann man fotorealistische 3D-Animationen programmieren: Bälle bewegen sich durch den Raum, Feder-Masse-Systeme schwingen, mechanische Wellen breiten sich aus (usw.).

1.3 Entwicklungsumgebungen

Eine Entwicklungsumgebung für Python (engl. *integrated development environment*, IDE) ist ein Programm, mit dem man eigene Programme schreiben und testen kann. Eine IDE besteht mindestens aus einem Texteditor, einem Interpreter und einem Debugger. In den Texteditor wird der Quelltext (Programmcode) des Programms eingegeben. Jeder Python-Texteditor rückt den Quelltext automatisch ein, wenn nach einer Anweisungszeile ein Doppelpunkt eingegeben wird. Suchen und automatisches Ersetzen gehören ebenso zum Funktionsumfang einer Python-IDE.

Mit einem Klick auf einen Startbutton oder durch Betätigen der Funktionstaste F5 wird das Programm gestartet. Alternativ kann man auch im Menü **Ausführen** auf den Menüeintrag **Ausführen** klicken (gilt für Thonny und Spyder). Nach dem Programmstart wird das Programm Zeile für Zeile vom Python-Interpreter übersetzt. Das Ergebnis wird in der Python-Shell (Kommandozeile bei Thonny, Konsole bei Spyder) ausgegeben. Wenn der Quelltext nicht den vorgeschriebenen Syntaxregeln entspricht, erscheint nach dem Programmstart eine Fehlermeldung in der Python-Shell. Bei der Fehlersuche unterstützt Sie ein Fehlersuchprogramm, das in der Fachsprache der praktischen Informatik als Debugger bezeichnet wird.

Es gibt zahlreiche Entwicklungsumgebungen für Python. Es sollen hier aber nur zwei Entwicklungsumgebungen kurz vorgestellt werden: Thonny und Spyder. Wenn Sie

sich für diese Entwicklungsumgebungen entscheiden, müssen Sie den Python-Interpreter nicht extra installieren. In beide IDEs ist bereits ein Interpreter integriert.

1.3.1 Thonny

Thonny ist eine Entwicklungsumgebung, die speziell für Programmieranfänger konzipiert wurde. Die Installation von Python-Erweiterungsmodulen ist mit Thonny besonders einfach, denn sie wird menügesteuert durchgeführt (funktioniert nur für die Windows- und macOs-Version). Sie können sich Thonny unter der URL *https://thonny.org* aus dem Internet herunterladen und auf Ihren PC-installieren. Die Nutzung ist kostenfrei. Bild 1.1 zeigt den Screenshot der Benutzeroberfläche.

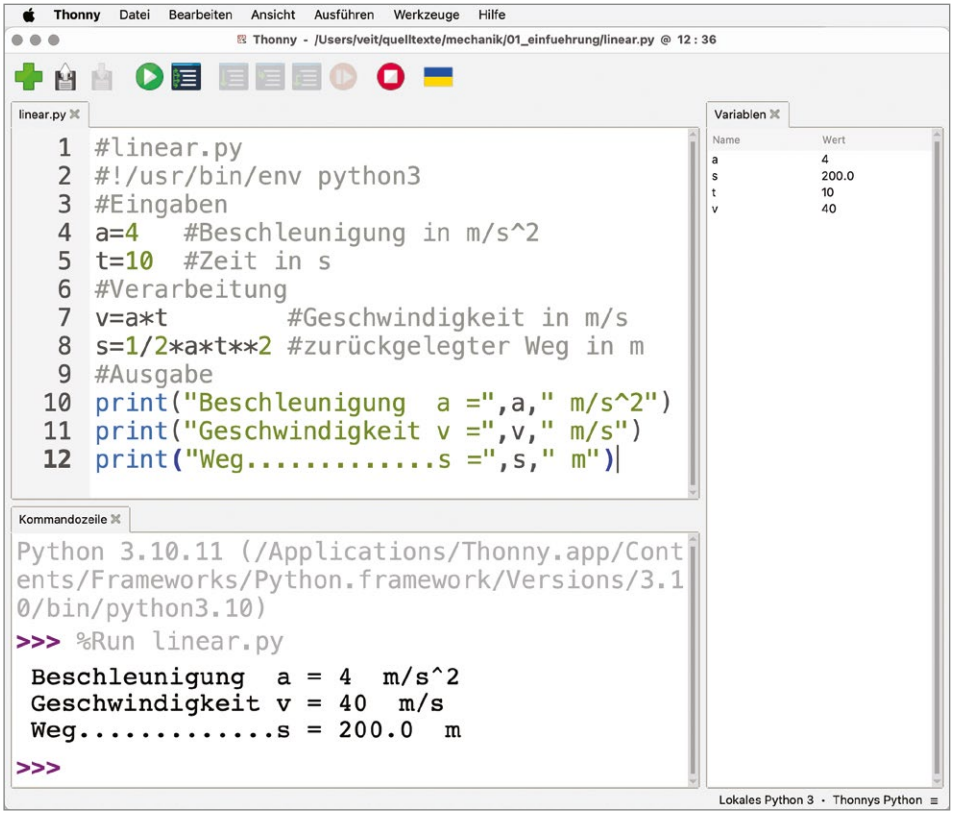

Bild 1.1 Entwicklungsumgebung von Thonny

Der Texteditor ist im oberen linken Fenster angeordnet. Die Registerkarte (Reiter) zeigt den Dateinamen des geöffneten Programms an. In diesem Fenster müssen Sie Ihren Quelltext (Programmcode) eingeben. Wenn Sie auf das grüne Icon (Startbutton)

klicken, wird das Programm ausgeführt. Das Ergebnis erscheint in dem direkt darunter liegenden Fenster, der **Kommandozeile** (andere Bezeichnung für die Python-Shell). Mit der Tastenkombination **Strg** + **K** (macOs) oder **Strg** + **L** (Windows) können Sie den Inhalt der Kommandozeile löschen. Das rechte Fenster mit der Registrierkarte **Variablen** gibt die Namen der im Programm verwendeten Variablen und deren Werte aus.

Thonny bietet verschiedene Designs der Benutzeroberfläche an. Wenn Sie das Erscheinungsbild der IDE ändern wollen, müssen Sie im Menü **Werkzeuge** den Menüeintrag **Optionen** auswählen, dann auf die Registerkarte **Theme & Schriftart** klicken und abschließend in dem Aufklapp-Menü das gewünschte Design auswählen. In Bild 1.1 ist das **UI Theme** *Raspbery Pi* dargestellt.

Berechnungen mit der Python-Shell

Wenn Sie die Installation von Thonny abgeschlossen haben, können Sie bereits mit der Kommandozeile (Python-Shell) erste Tests durchführen. Probieren Sie die folgenden Rechenoperationen aus:

```
>>> 3+4
7
>>> 7/2
3.5
>>> 7//2  #Ganzzahldivision
3
>>> 12*4-2*3
42
```

Schlüsselwörter von Python

Mit den Befehlen `import keyword` und `keyword.kwlist` werden die 35 Schlüsselwörter von Python in der Python-Shell eingegeben.

```
>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class',
'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global',
'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise',
'return', 'try', 'while', 'with', 'yield']
>>> len(keyword.kwlist)
35
```

Schlüsselwörter sind festgelegte Bezeichner für Python-Befehle. Sie dürfen diese Schlüsselwörter nicht als Namen (Bezeichner) für Variablen verwenden. Wenn Sie z. B. das Schlüsselwort `for` als Variable verwenden und ihr einen Wert zuweisen, erhalten Sie eine Fehlermeldung:

```
>>> for=12
  File "<stdin>", line 1
    for=12
       ^
SyntaxError: invalid syntax
```

Um einfache Python-Programme zu schreiben, benötigen Sie nur die Schlüsselwörter `if`, `else`, `elif`, `while` und die eingebaute Python-Funktion `print()`.

1.3.2 Spyder

Spyder ist eine professionelle Entwicklungsumgebung. Sie ist ebenfalls für macOs, Windows und Linux verfügbar. Sie können Spyder unter der URL *https://www.spyder-ide.org/download* aus dem Internet herunterladen und auf Ihrem PC installieren. Ein besonderer Vorteil dieser IDE besteht darin, dass die Module NumPy, Matplotlib, SymPy und SciPy bereits installiert sind. Probleme, die bei der Installation dieser Module auftreten können, werden so von vornherein ausgeschlossen. Bild 1.2 zeigt den Screenshot der Benutzeroberfläche.

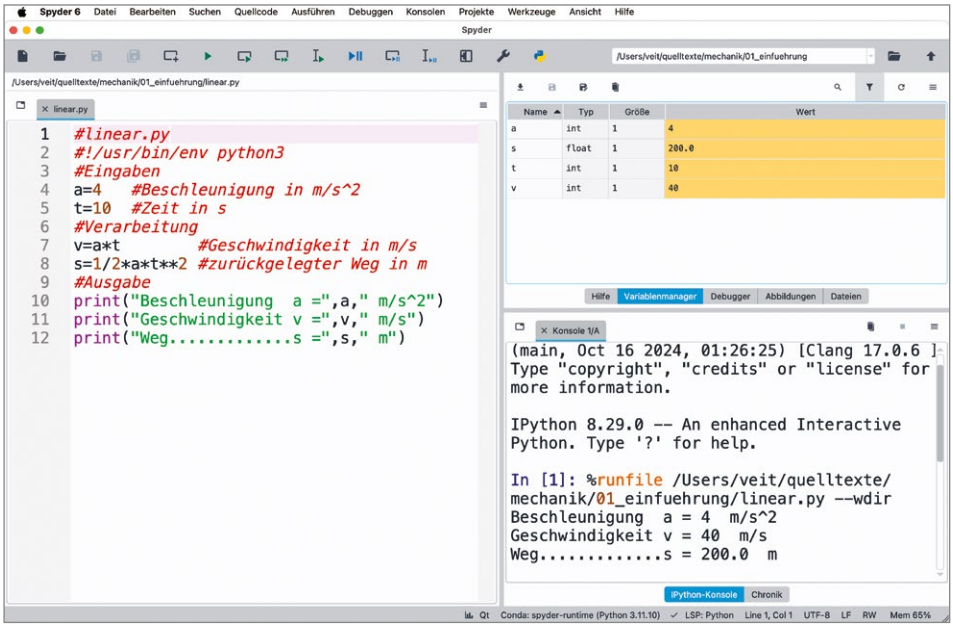

Bild 1.2 Entwicklungsumgebung von Spyder

Für die Ausführung von Matplotlib-Programmen ist es notwendig, dass Sie in den Einstellungen im Eintrag **IPython-Konsole** in der Registerkarte **Grafik**, **Grafik-Backend** die Option **Automatisch** auswählen. Ansonsten werden die Funktionsplots und Animationen nicht angezeigt.

1.4 Installation der Module

Mit Thonny ist es besonders einfach, die in diesem Buch verwendeten Module zu installieren. Wählen Sie in der IDE die Option **Werkzeuge** aus und wählen Sie dann den Menüeintrag **Verwalte Pakete** aus (funktioniert nur unter macOS und Windows). Es öffnet sich dann ein Dialogfenster (Bild 1.3).

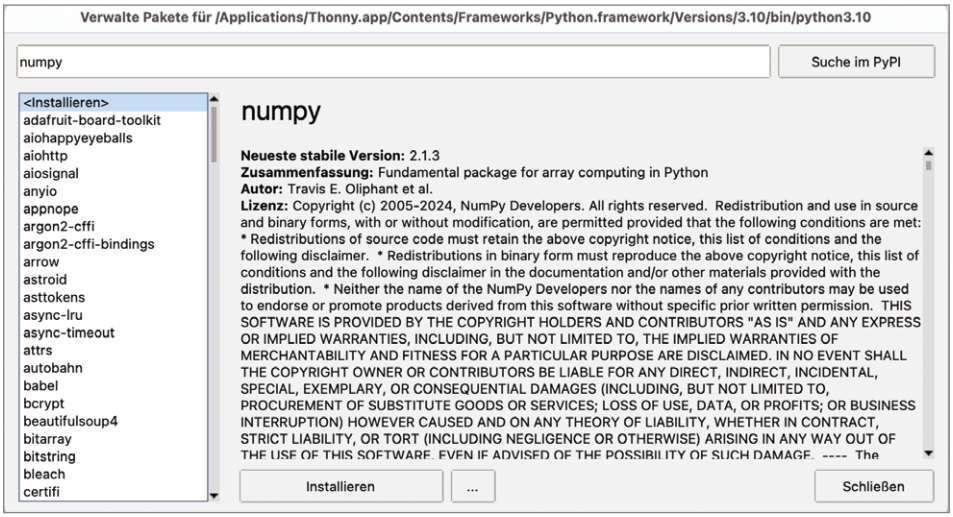

Bild 1.3 Installation der Module

Geben Sie den Namen des Moduls in das Textfeld ein und klicken Sie dann auf die Befehlsschaltfläche (Button) **Suche im PyPy**. Wenn Sie auf die Befehlsschaltfläche **Installieren** klicken, wird das Modul installiert. Installieren Sie die Module in der folgenden Reihenfolge: *numpy*, *matplotlib*, *sympy*, *scipy*, *numdifftools* und *vpython*. Sie können auch einzelne Module wieder deinstallieren und ältere Versionen der Module installieren. Dazu müssen Sie auf die Befehlsschaltfläche **...** klicken und in dem Aufklapp-Menü **Gewünschte Version** die gewünschte Versionsnummer auswählen.

Das Installationsprogramm pip

Wenn Sie nicht Thonny oder Spyder nutzen oder Ihre Python-Programme in einem Terminal ausführen wollen, dann müssen Sie die Module mit dem Installationsprogramm *pip* installieren. Dazu müssen Sie zuerst Python installieren (*https://www.python.org/downloads/*). Das Installationsprogramm *pip* wird dann automatisch mit installiert. Anschließend geben Sie in einem Terminal-Fenster folgenden Befehl ein: *pip3 install numpy*.

Diesen Vorgang müssen Sie für die anderen Module wiederholen, indem Sie anstatt *numpy* die Namen der Module eingeben, die Sie installieren wollen.

Wenn Sie eine neue Version von Python installieren, z. B. Version 3.13, dann werden die zuvor installierten Module bei der Ausführung eines Programms nicht mehr importiert. In diesem Fall müssen Sie alle Module mit *pip3.13 install modulname* neu installieren.

Unter Debian-basierten Linux-Betriebssystemen müssen Sie die Module mit *sudo apt install python3-xyz* installieren, wobei *xyz* für die Namen der Module steht, die Sie installieren wollen. Leider lassen sich zum Zeitpunkt der Drucklegung (Juni 2025) die Module *VPython* und *Numdifftools* nicht auf Debian-basierten Linux-Betriebssystem installieren. Alle anderen Programme aus diesem Buch können Sie mit den Debian-basierten Betriebssystemen *Raspberry Pi 5* und *Mint* ohne Einschränkungen ausführen.

Wenn Sie Spyder nutzen, werden Programme mit den Modulen *VPython* und *Numdifftools* in der IDE **nicht** ausgeführt. In diesem Fall können Sie diese Programme in einem Terminal ausführen. Starten Sie ein Terminal und wechseln Sie in das Verzeichnis, in dem das VPython-Programm gespeichert ist, und führen Sie den folgenden Befehl aus: *python3 vp_programmname.py*. Alle VPython-Programme erkennen Sie an dem Präfix *vp*.

Hinweis

Die oben beschrieben Installationsanweisungen können sich im Laufe der Zeit ändern. Falls sie nicht mehr funktionieren sollten, müssen Sie im Internet nach den aktualisierten Installationsanweisungen suchen.

1.5 Programmstrukturen

Die praktische Informatik kennt nur drei Programmstrukturen: die lineare Programmstruktur, die Verzweigungs- und die Wiederholstruktur. Ein Programm besteht in der Regel aus einem Hauptprogramm und mehreren Unterprogrammen. Die Unterprogramme werden in Python als Funktionen oder Methoden bezeichnet. Im Folgenden wird davon ausgegangen, dass die Beispielprogramme numerische Berechnungen durchführen. Für eine vorgegebene mathematische Gleichung soll der gesuchte Zahlenwert berechnet werden.

1.5.1 Lineare Programmstruktur

Ein Programm mit linearer Struktur besteht aus drei Teilen: der Eingabe, der Verarbeitung und der Ausgabe. Im Eingabeteil werden den Variablen Zahlenwerte zugewiesen. Im Verarbeitungsteil werden die Berechnungen durchgeführt und im Ausgabeteil werden die Ergebnisse dieser Berechnungen auf dem Bildschirm (Monitor) ausgegeben.

Am Beispiel der gleichförmig beschleunigten Bewegung soll gezeigt werden, wie Berechnungen mit einem Python-Programm prinzipiell durchgeführt werden können. Die Beschleunigung a und die Zeit t sind gegeben. Für die Geschwindigkeit v gilt dann:

$$v = a \cdot t$$

Für den zurückgelegten Weg s gilt:

$$s = \frac{1}{2} a \cdot t^2$$

Geben Sie den nachfolgenden Quelltext in den Texteditor Ihrer IDE ein und starten Sie das Programm.

Listing 1.1 Lineare Programmstruktur

```
1   #linear.py
2   #!/usr/bin/env python3
3   #Eingaben
4   a=4   #Beschleunigung in m/s^2
5   t=10  #Zeit in s
6   #Verarbeitung
7   v=a*t        #Geschwindigkeit in m/s
8   s=1/2*a*t**2 #zurückgelegter Weg in m
9   #Ausgabe
10  print("Geschwindigkeit v =",v," m/s")
11  print("Weg.............s =",s," m")
```

Wenn Sie alles richtig abgetippt haben, erscheint in der Python-Shell (Kommandozeile) die folgende Ausgabe:

```
Geschwindigkeit v = 40 m/s
Weg.............s = 200.0 m
```

Analyse

Das Rautezeichen `#` kennzeichnet einen Kommentar. Die auskommentierten Programmzeilen werden vom Python-Interpreter nicht berücksichtigt. Kommentare sollen die Anweisungen näher beschreiben, damit man das Programm besser verstehen kann. In Zeile 1 steht der Name des Programms `#linear.py`. Dies ist der Dateiname, unter dem das Programm abgespeichert wird. Die Zeile 2 `#!/usr/bin/env python3` teilt dem Betriebssystem den Dateipfad des Python-Interpreters mit. Die ersten beiden

Zeichen `#!` werden als **Shebang** bezeichnet. Der Interpreter befindet sich im Verzeichnis `/usr/bin`. Der Linux-Befehl `env` (*environment*) bewirkt, dass das Programm mit dem richtigen Interpreter ausgeführt wird, wenn der genaue Pfad zum Interpreter nicht bekannt ist. Unter Windows wird die Shebang-Zeile ignoriert. Wenn Sie das Programm in einem Terminal ausführen wollen, dann müssen Sie, falls Sie macOS oder Linux nutzen, für die Datei *linear.py* das Attribut *ausführbar* `+x` setzen. Das bewirken Sie mit dem Terminal-Befehl `chmod +x linear.py`. Um das Programm `linear.py` in einem Terminal auszuführen, geben Sie `python3 linear.py` in das Terminalfenster ein und drücken Sie die **Return-Taste**. Das Ergebnis wird im Terminal ausgegeben.

In den Zeilen 4 und 5 stehen die Eingaben. Streng genommen handelt es sich hier nicht um typische Eingaben, sondern um Zuweisungen. Eingaben werden mit der eingebauten Python-Funktion `input(...)` realisiert (siehe Listing 1.6). Eine Zuweisung erfolgt mit dem Operator `=`. Das Gleichheitszeichen darf nicht mit der Gleichsetzung einer mathematischen Gleichung verwechselt werden. In Python wird die Gleichheit mit dem Operator `==` überprüft. Links vom Zuweisungsoperator steht eine Variable (hier: a,t,v,s), rechts vom Zuweisungsoperator steht ein Zahlenwert. Eine Variable ist eine symbolische Bezeichnung für eine Speicheradresse im Arbeitsspeicher eines Computers. In der Variablen `a` ist der Zahlenwert 4 und in der Variablen `t` ist der Zahlenwert 10 gespeichert. Der Zuweisungsoperator bewirkt, dass ein Zahlenwert in eine Speicherzelle abgespeichert wird. Wenn Sie nach dem Programmstart in der Python-Shell (Kommandozeile) `>>> a` eingeben, wird der unter der symbolischen Adresse `a` gespeicherte Wert `4` ausgegeben.

In den Zeilen 7 und 8 werden die Berechnungen ausgeführt: Es werden die Geschwindigkeit `v` und der zurückgelegte Weg `s` berechnet. Die Multiplikationen werden mit dem Operator `*` (Asterisk-Symbol) realisiert. Mit dem Operator `**` wird eine Variable potenziert (Zeile 8). Die Variable `t` ist die Basis und die Konstante `2` ist der Exponent der Potenz.

In den Zeilen 10 und 11 gibt die eingebaute Python-Funktion `print("...",variable)` die in den Zeilen 7 und 8 berechneten Ergebnisse aus. Wenn eine Zeichenkette zwischen Anführungszeichen eingebettet wird, wird sie als Text auf dem Bildschirm ausgegeben. Durch ein Komma getrennt folgt der Bezeichner der Variablen, dessen Wert auf dem Bildschirm ausgegeben werden soll.

Übung: Ermittlung von Datentypen

Geben Sie nach dem Programmstart folgende Befehle in die Python-Shell ein:

```
>>> type(a)
<class 'int'>
>>> type(v)
<class 'int'>
>>> type(s)
<class 'float'>
>>> id(a)
```

```
4470784336
>>> id(t)
4470784528
>>> id(v)
4470785488
>>> id(s)
4474991952
```

Ändern Sie in den Zeilen 4 und 5 die Zuweisungen: `a = 4.0` und `t = 10.0`. Welchen Datentyp haben die Variablen `a` und `t` jetzt?

Mit der eingebauten Python-Funktion `type()` können Sie den Datentyp einer Variablen ermitteln und mit der eingebauten Funktion `id()` ermitteln Sie die Identität (temporäre Adresse im Speicher) einer Variablen. Bei jedem neuen Programmstart ändern sich die Identitäten. Python stellt fünf Datentypen zur Verfügung: Integer (Ganzzahlen), Float (Gleitpunktzahlen), String (Zeichenketten), Complex (komplexe Zahlen) und Boolean (boolesche Variable). Eine Python-Variable hat also einen Namen (Bezeichner), einen Datentyp und eine Identität.

1.5.2 Verzweigungsstrukturen

Während der Programmausführung kann es vorkommen, dass eine Variable ihren Wert ändert. Wenn an diese Änderung bestimmte Bedingungen geknüpft sind, ist eine Fallunterscheidung notwendig. Je nach vorgegebener Bedingung muss eine Entscheidung getroffen werden, welcher Programmzweig ausgeführt werden soll. Es wird zwischen Einfachverzweigung und Mehrfachverzweigung unterschieden. Die Einfachverzweigung bzw. Mehrfachverzweigung wird auch als Einfachauswahl bzw. Mehrfachauswahl bezeichnet.

Einfachverzweigung

Eine Einfachverzweigung wird mit den Schlüsselwörtern `if` und `else` realisiert. Geben Sie den folgenden Quelltext in den Texteditor Ihrer IDE ein. Wenn Sie die Anweisung in Zeile 3 mit einem Doppelpunkt abschließen und die Return-Taste drücken, wird die Anweisung in Zeile 4 automatisch eingerückt. Durch diese Einrückung erkennt der Python-Interpreter, dass alle folgenden eingerückten Anweisungen ausgeführt werden, wenn sie die Bedingung der `if`-Anweisung erfüllen.

Listing 1.2 Einfachverzweigung

```
1  #einfachauswahl.py
2  v=50
3  if v <= 50:
4      print("Sie haben die Geschwindigkeit nicht überschritten.")
5  else:
6      print("Sie haben die Geschwindigkeit überschritten!")
```

Wenn Sie den Quelltext korrekt eingegeben haben, erhalten Sie die folgende Ausgabe:

```
Sie haben die Geschwindigkeit nicht überschritten.
```

Analyse

In Zeile 2 wird der Variablen `v` der Wert 50 zugewiesen. In Zeile 4 überprüft die `if`-Anweisung, ob der Wert der Variablen `v` kleiner gleich 50 ist. Da dies der Fall ist, wird die Meldung aus Zeile 4 ausgegeben.

Wenn dieser Fall nicht zutrifft, weil z. B. in der Variablen `v` der Wert 60 gespeichert wurde, wird der `else`-Zweig in Zeile 5 ausgeführt und die Meldung aus Zeile 6 ausgegeben.

Übung: Einfachverzweigung

Weisen Sie der Variablen `v` in Zeile 2 den Wert 50.1 zu und starten Sie das Programm neu.

Machen Sie die Einrückung in Zeile 4 rückgängig und starten Sie das Programm neu. Sie erhalten dann folgende Fehlermeldung:

```
IndentationError: expected an indented block after 'if' statement on line 3
```

Mehrfachverzweigung

Bei einer Mehrfachverzweigung werden mehrere Bedingungen abgefragt. Mehrfachverzeigungen werden mit den Schlüsselwörtern `if` und `elif` implementiert.

Listing 1.3 Mehrfachauswahl

```
1   #mehrfachauswahl.py
2   v=65
3   if v > 10 and v < 20:
4       print("In den 2. Gang schalten.")
5   elif v > 20 and v < 40:
6       print("In den 3. Gang schalten.")
7   elif v > 40 and v < 60:
8       print("In den 4. Gang schalten.")
9   elif v > 60:
10      print("In den 5. Gang schalten.")
11  else:
12      print("Fehler!")
```

Wenn Sie das Programm starten, erhalten Sie die folgende Ausgabe:

```
In den 5. Gang schalten.
```

Analyse

In Zeile 3 wird die erste Bedingung festgelegt. Wenn Bedingung „v größer 10 **und** kleiner 20“ erfüllt ist, soll die Anweisung in Zeile 3 ausgeführt werden. Entsprechendes gilt für die Bedingungen, die mit dem Schlüsselwort `elif` in den Zeilen 5, 7 und 9 abgefragt werden. Das Schlüsselwort `and` ist ein logischer Operator, der zwei Fälle miteinander logisch verknüpft. Wenn z. B. in der Variablen `v` der Wert 15 gespeichert wurde, dann liegt der Wert von `v` zwischen 10 und 20, die Aussage `v > 10 and v < 20` ist also wahr (`True`).

Übung: Mehrfachauswahl

Testen Sie das Programm mit verschiedenen Geschwindigkeiten.

Weisen Sie der Variablen `v` (Zeile 2) den Wert 5 zu und starten Sie das Programm neu. Begründen Sie die Ausgabe.

Weisen Sie der Variablen `v` (Zeile 2) den String '65' zu und starten Sie das Programm neu. Analysieren Sie die Fehlermeldung.

Weisen Sie der Variablen `v` den Wert 30 zu und führen Sie nach einem Neustart den folgenden Konsolendialog aus:

```
>>> v > 10
True
>>> v < 20
False
>>> v > 10 and v < 20
False
>>> v > 20
True
```

1.5.3 Wiederholstrukturen

Python stellt zwei Schleifenkonstrukte (Wiederholstrukturen) zur Verfügung: die zählergesteuerte `for`-Schleife und die `while`-Schleife. Schleifenkonstrukte werden immer dann benötigt, wenn eine oder mehrere Berechnungen wiederholt werden müssen, wie das z. B. bei der Erstellung einer Wertetabelle der Fall ist.

Die for-Schleife

Die `for`-Schleife ist eine zählergesteuerte Schleife. Zu Beginn der Programmausführung muss bekannt sein, wie oft die Schleife durchlaufen werden soll. Das Programm *for_schleife.py* (Listing 1.4) berechnet eine Wertetabelle für die beschleunigte Bewegung. Geben Sie den Quelltext (Listing 1.4) in den Texteditor Ihrer IDE ein.

Listing 1.4 for-Schleife für Wertetabelle

```
1  #for_schleife.py
2  tmax=10 #s
3  v=10    #m/s
4  print("Zeit\t Weg")
5  for t in range(0,tmax,1):
6      s=v*t
7      print(t,"\t",s)
```

Nach dem Programmstart erscheint folgende Ausgabe auf dem Bildschirm:

```
Zeit   Weg
0      0
1      10
2      20
3      30
4      40
5      50
6      60
7      70
8      80
9      90
```

In Zeile 4 gibt die eingebaute `print`-Funktion die Tabellenüberschrift aus. Die Escapesequenz `\t` in Zeile 4 und Zeile 7 (horizontaler Tabulator) bewirkt, dass zwischen den Spalten `Zeit` und `Weg` vier Leerzeichen eingefügt werden.

Die Zählvariable `t` in Zeile 5 nimmt nacheinander die Werte von 0 bis 9 an. In Zeile 6 wird bei jedem Schleifendurchlauf der zurückgelegte Weg neu berechnet und dann in Zeile 7 ausgegeben.

Die Definition einer `for`-Schleife beginnt mit dem Schlüsselwort `for`. Danach folgt eine Zählvariable (hier `t`). Für Zählvariablen wählt man traditionell die Bezeichner `i`, `j` oder `k`. Direkt hinter der Zählvariable steht das Schlüsselwort `in`. Die eingebaute Python-Funktion `range(0,tmax,1)` enthält den Anfangswert der Zählvariablen (hier: 0), gefolgt von dem Endwert (hier: `tmax`) und als drittes Argument die Schrittweite. Wenn kein Anfangswert und keine Schrittweite angegeben werden, ist der Anfangswert 0 und die Schrittweite 1. Jede `for`-Schleife muss mit einem Doppelpunkt abgeschlossen werden.

Eine Schleife besteht immer aus einem **Schleifenkopf** (Zeile 5) und einem **Schleifenkörper** (Zeile 6 und 7). Alle Anweisungen im Schleifenkörper müssen eingerückt werden. Das erledigt der Texteditor der Python-IDE automatisch, wenn der Schleifenkopf mit einem Doppelpunkt abgeschlossen wird und die **Return**-Taste gedrückt wird.

Die Anweisung in Zeile 5 kann umgangssprachlich etwa so ausgedrückt werden: Setze für die Variable `t` nacheinander die Werte 0, 1, … 9 ein. Erhöhe den Wert für `t` bei jedem neuen Schleifendurchlauf um den ganzzahligen Wert 1. Beende die Wiederholungen, wenn die Variable `t < tmax` ist.

Übung: for-Schleife

Löschen Sie den Doppelpunkt in Zeile 5 und starten Sie das Programm neu. Analysieren Sie die Fehlermeldung.

Fügen Sie in der `range`-Funktion (Zeile 5) für den Startwert 1 ein und für die Schrittweite den Wert 2. Starten Sie das Programm neu und analysieren Sie die Ausgabe.

Wählen Sie für die Schrittweite den Wert 0.5 und starten Sie das Programm neu. Analysieren Sie die Fehlermeldung.

Die while-Schleife

Bei einer `while`-Schleife steht die Abbruchbedingung im Schleifenkopf. Die Zählvariable wird im Schleifenrumpf hochgezählt. Das Programm *while_schleife.py* (Listing 1.5) berechnet eine Wertetabelle für den zurückgelegten Weg $s = v \cdot t$ einer gleichförmigen Bewegung. In Zeile 3 können Sie eine andere Geschwindigkeit vorgeben.

Listing 1.5 while-Schleife für Wertetabelle

```
1   #while_schleife.py
2   tmax=10 #s
3   v=10    #m/s
4   dt=1
5   t=0
6   print("Zeit\t Weg")
7   while t<tmax:
8       s=v*t
9       print(t,"\t",s)
10      t=t+dt
```

Nach dem Programmstart erscheint folgende Ausgabe:

```
Zeit    Weg
0       0
1       10
2       20
3       30
4       40
5       50
6       60
7       70
8       80
9       90
```

Analyse

In Zeile 2 wird die maximale Zeit `tmax`, für die die Wertetabelle berechnet werden soll, festgelegt. Zeile 4 legt die Schrittweite `dt` und Zeile 5 legt den Startwert `t = 0` der Schleife fest.

In Zeile 7 beginnt die `while`-Schleife. Sie wird mit dem Schlüsselwort `while` eingeleitet. Es folgt die Abbruchbedingung `t < tmax`. Der Schleifenkopf muss wie bei der `for`-Schleife wieder mit einem Doppelpunkt `:` abgeschlossen werden. Die Variable `t` wird in Zeile 10 bei jedem neuen Schleifendurchlauf um den Wert von `dt` erhöht. Der Schleifenrumpf (Zeile 8 bis 10) wird so lange ausgeführt, bis die Abbruchbedingung in Zeile 7 zutrifft. Die Einrückung des Schleifenkörpers erfolgt wieder automatisch, nachdem Sie am Ende der Zeile 7 die Return-Taste betätigt haben. Wenn Sie am Zeilenende von Zeile 7 den Doppelpunkt vergessen haben, wird der nachfolgende Quelltext nicht automatisch eingerückt.

Übung: while-Schleife

Weisen Sie der Variablen `t` in Zeile 5 den Wert 10 zu und starten Sie das Programm. Was passiert?

Testen Sie das Programm mit der Schrittweite `dt = 0.5` (Zeile 4).

Testen Sie das Programm mit dem Anfangswert `t = 1` (Zeile 5).

Kopieren Sie die Anweisung `t = t + dt` aus Zeile 10 direkt unter dem Schleifenkopf und starten Sie das Programm neu. Welche Auswirkung hat diese Änderung?

Testen Sie die `while`-Schleife mit der Abbruchbedingung `t <= tmax`.

Testen Sie das Programm mit der Anweisung `t += dt` (Zeile 10).

Programm mit Auswahlmenü

In dem nächsten Programm *while_mehrfachauswahl.py* (Listing 1.6) wird eine `while`-Schleife mit einer Mehrfachauswahl `if – elif` kombiniert. Das Programm berechnet den Weg *s* oder die Geschwindigkeit *v* oder die Zeit *t* für eine gleichförmig beschleunigte Bewegung. In einem Auswahlmenü kann die gesuchte Größe ausgewählt werden. Nach dem Programmstart erscheint ein Auswahlmenü. Durch die Eingabe einer Ganzzahl können Sie die Größe auswählen, die Sie berechnen wollen.

Listing 1.6 Mehrfachauswahl mit while-Schleife

```
1   #while_mehrfachauswahl.py
2   auswahl=1
3   while True:
4       print("-----------------")
5       print("Weg.............:1")
6       print("Geschwindigkeit:2")
7       print("Zeit............:3")
8       print("Ende............:0")
9       print("-----------------")
10      try:
11          auswahl=int(input("Wählen Sie aus:"))
12      except ValueError:
13          print("Falsche Eingabe!")
14      #Weg
```