1 Einführung

Dieses Kapitel gibt Ihnen einen kurzen Überblick über die Inhalte des Buches. Es stellt außerdem zwei Entwicklungsumgebungen vor und beschreibt die Problematik der Rundungsfehler. Eine Übersicht über die Schlüsselwörter von Python soll die erste Orientierung erleichtern. Zum Ende des Kapitels werden verschiedene Aspekte des Algorithmusbegriffes analysiert.

Wenn Sie in die Welt der Algorithmen eintauchen wollen, dann möchten Sie sicherlich auch lernen, wie Sie Algorithmen in ein Computerprogramm umsetzen können. Dazu müssen Sie eine Programmiersprache beherrschen. Bei der Auswahl einer Programmiersprache haben Sie unglaublich viele Möglichkeiten, denn es gibt inzwischen Hunderte Sprachen für unterschiedliche Einsatzgebiete.

Die Entscheidung ist zwar mit dem Titel des Buches bereits gefallen, sie lässt sich aber leicht begründen, denn Python ist laut PYPL-Index (*PopularitY of Programming Languages*) die aktuell beliebteste Programmiersprache [Oktober 2021]. Dafür gibt es gute Gründe, denn:

- Python ermöglicht als interpretierende Programmiersprache eine interaktive Entwicklung von Programmen. Python ist deshalb für den Einstieg in das Programmieren besser geeignet als Compilersprachen, die erst in einem zweiten Schritt ausführbar gemacht werden.
- Der Syntax von Python ist im Vergleich zu anderen Programmiersprachen relativ einfach zu erlernen.
- Python ist durch Module (*Softwarebibliotheken*) erweiterbar. Das gilt zwar auch für viele andere Programmiersprachen. In Python gestaltet sich die Einbindung von Erweiterungsmodulen aber einfacher. Für numerische und symbolische Berechnungen sowie Computergrafik werden spezifische Module bereitgestellt, die ich Ihnen in diesem Buch vorstellen möchte.
- Python steht kostenlos zur Verfügung und läuft auf den Betriebssystemen Windows, Linux und macOS.

Das Konzept

Wenn Sie eine Programmiersprache erlernen wollen, dann gibt es verschiedene Wege, die zum Ziel führen. Sie könnten die Dokumentation der Programmiersprache durcharbeiten und versuchen, mit den so erworbenen Kenntnissen eigene Programme zu schreiben. Dieser Weg ist zwar denkbar, aber dann müssen Sie sich durch eine unübersichtliche Vielfalt an Informationen arbeiten, denn die Python-Dokumentation und die Unterlagen der Zusatzmodule haben einen Umfang von mehreren Tausend Seiten. Tutorials für den Programmiereinstieg können dieses Problem zwar abfedern, die dort behandelten Beispiele haben aber in der Regel keinen direkten Bezug zu dem konkreten Programmierproblem, das Sie gerade umtreibt, und benötigen damit eine ziemlich schwierige Transferleistung.

Ich habe die Erfahrung gemacht, dass ein anderer Lernweg besser ist: Konzentrieren Sie sich auf konkrete Inhalte aus der Praxis des wissenschaftlichen Arbeitens, und üben Sie damit das Programmieren. Das Lernen orientiert sich dann nicht an der Fachsystematik der Programmiersprache, sondern an der Fachsystematik einer wissenschaftlichen Disziplin, die Sie idealerweise bereits kennen.

Doch welche Wissenschaft sollte man aus dem reichhaltigen Angebot wählen? Hier bietet sich die Mathematik an, weil sie als Grundlagenwissenschaft der Natur- und Ingenieurswissenschaften von allen Fachkundigen ohnehin beherrscht werden muss. Wenn Sie also das Erlernen einer Programmiersprache mit den Inhalten der Mathematik verknüpfen, dann beherrschen Sie am Ende nicht nur das Programmieren, sondern haben auch noch Ihre Mathematikkenntnisse vertieft.

Dieser Weg hat jedoch auch einen Nachteil: Wenn sich das Erlernen einer Programmiersprache an den Inhalten der Mathematik orientiert, dann werden zwangsläufig nicht alle Möglichkeiten erfasst, die die Syntax einer Programmiersprache bietet. Es gibt hier aber einen einfachen Ausweg: Sollten beim Programmieren Probleme vorkommen, dann schlagen Sie gezielt die fehlenden Informationen in den entsprechenden Dokumentationen oder in einem einschlägigen Fachbuch nach. Wenn Sie bereits über grundlegendes Wissen verfügen, lässt sich nämlich viel zielgerichteter mit diesen Informationsquellen arbeiten.

Weil dieses Buch sich an der Fachsystematik der Mathematik orientiert, wiederholen sich zwangsläufig viele Programmierkonstruktionen, z. B. Summenalgorithmen in Schleifen oder Fallunterscheidungen in Verzweigungsstrukturen. Das muss jedoch kein Nachteil sein, denn durch die vielen Wiederholungen wird eine Schwäche unseres Gedächtnisses, nämlich das Vergessen, kompensiert.

Die Quelltexte werden, bis auf wenige Ausnahmen, vollständig mit der zugehörigen Ausgabe abgedruckt, damit sie auch ohne Nutzung des Computers analysiert werden können. Für das Verstehen des Codes eines Programms ist es besonders wichtig, die Ausgabe der Ergebnisse mit in die Quelltextanalyse einzubeziehen, denn nur durch die Einsicht in den kausalen Zusammenhang zwischen Struktur des Quelltextes und Ausgabe der Resultate kann der Quelltext vollständig verstanden werden.

Der Lernprozess beginnt mit der Analyse von zunächst einfachen Programmquelltexten. Es wird nicht immer die hinsichtlich Rechenaufwand effizienteste Lösung gewählt, sondern eher Wert auf Nachvollziehbarkeit des Algorithmus gelegt.

Erst wenn Sie einige Beispielprogramme vollständig verstanden und auch getestet haben, sollten Sie mit eigenen Programmierprojekten beginnen. Beim Testen von Programmbeispielen ist es mitunter sinnvoll, bewusst Fehler in den Quelltext einzubauen und nach dem Programmstart die Fehlermeldungen zu analysieren. Auch das Einfügen von eignen Ergänzungen in bestehende Quelltexte kann den Lernfortschritt unterstützen. Zögern Sie also nicht, den Code zu verändern und sich z. B. Zwischenergebnisse auf der Kommandozeile ausgeben zu lassen, wenn Sie den Ablauf eines Programms besser verstehen möchten.

Die Zielgruppe

Dieses Buch wendet sich an alle, die Python als erste Programmiersprache erlernen wollen. Als Leserkreis werden dabei besonders Studierende der Natur- und Ingenieurwissenschaften und

der Informatik angesprochen. Das Buch kann aber auch für den Mathematikunterricht von großem Nutzen sein, wenn mathematische Begriffe, z. B. Grenzwertprozesse, durch Visualisierung dynamisch veranschaulicht werden sollen. Im Informatikunterricht könnten Programmierprojekte, die sich an den aktuellen Themen des Mathematikunterrichts orientieren, mathematische Lernprozesse unterstützen und zusätzlich das Lernen in der Programmierausbildung durch einen sinnhaften Anwendungsbezug stärker motivieren. Angesprochen werden auch Schülerinnen und Schüler der gymnasialen Oberstufe mit den Leistungskursen Mathematik und Informatik. Durch Synergieeffekte könnten Verständnisprobleme in beiden Fächern überwunden werden. Die Notwendigkeit, präzise Unterscheidungen zu treffen, könnte auch dazu beitragen, die Begriffsbildungen in den beiden Lerngebieten zu präzisieren.

Als Voraussetzung für das Verständnis der Inhalte des Buches sind Mathematikkenntnisse erforderlich, wie sie auf einem Gymnasium oder auf einer Fachoberschule vermittelt werden. Sie sollten die mathematischen Grundbegriffe *Funktion*, *Konvergenz*, *Grenzwert*, *Ableitung* und *Integral* kennen. Bei Verständnisschwierigkeiten können Sie Ihr Wissen mithilfe der Lehrbücher [Bossek] oder [Kammermeyer] auffrischen. Eine (ganz) kurze Zusammenfassung finden Sie auch im Anhang dieses Buches.

Ein kurzer Überblick über die Inhalte

Sie können bis auf die Kapitel 2 und Kapitel 3 alle anderen Kapitel unabhängig voneinander lesen. Kapitel 2 und Kapitel 3 sollten Sie jedoch intensiv studieren und die zugehörigen Aufgaben bearbeiten. Denn dort werden die Grundlagen für das Verständnis der nachfolgenden Kapitel gelegt.

In Kapitel 2, **»Datentypen und Datenstrukturen«**, werden die grundlegenden Datenstrukturen von Python behandelt, wie Tupel, Sets (Mengen), Listen und Dictionary. Bis auf Sets werden diese Datenstrukturen in allen folgenden Kapiteln wieder verwendet.

Kapitel 3, **»Programmstrukturen«**, beschreibt die Programmstrukturen Sequenz, Verzweigung und Wiederholung anhand einfacher Beispiele. Hier wird auch der Begriff der Laufzeitkomplexität eingeführt.

In Kapitel 4, **»Die Python-Erweiterungsmodule NumPy, Matplotlib, SymPy und SciPy«**, werden die Module `numpy`, `matplotlib`, `sympy` und `scipy` kurz vorgestellt. Methoden dieser Module, z. B. `solve()`, `dsolve()`, `diff()`, `integrate()` und `quad()`, werden später benutzt, um zu überprüfen, ob die selbst erstellten Algorithmen korrekte Berechnungen ausführen. Das Modul `matplotlib` wird verwendet, um 2D- und 3D-Funktionsplots zu erstellen und zu animieren.

Kapitel 5, **»Zahlen«**, handelt von Algorithmen mit Zahlen. Hier beschreibe ich, wie Sie mit dem euklidischen Algorithmus den größten gemeinsamen Teiler berechnen. Es wird gezeigt, wie das Sieb des Eratosthenes Primzahlen findet. Mit der Implementierung des RSA-Algorithmus wird die praktische Bedeutung der Primzahlen herausgestellt. Der Heron-Algorithmus berechnet näherungsweise Werte für irrationale Zahlen als Quadrat- und Kubikwurzeln. Die Kreiszahl π wird näherungsweise mit dem Algorithmus von Archimedes berechnet. Das Kapitel endet mit einem Überblick über die Eigenschaften der eulerschen Zahl e.

Kapitel 6, **»Gleichungssysteme«**, behandelt die numerische Lösung linearer und nichtlinearer Gleichungssysteme. Mit dem klassischen Gauß-Algorithmus und iterativen Verfahren wie dem

Jakobi- und dem Gauß-Seidel-Verfahren werden lineare Gleichungssysteme gelöst. Nichtlineare Gleichungssysteme werden mit der SciPy-Methode `fsolve()` und der SymPy-Methode `nonlinsolve()` gelöst.

In Kapitel 7, **»Folgen«**, wird gezeigt, wie mit der Matplotlib-Methode `scatter()` arithmetische und geometrische Folgen visualisiert werden können. Im Mittelpunkt steht die grafische Veranschaulichung des Grenzwertbegriffes. Abschließend wird demonstriert, wie sich der Grenzwert einer Folge mit der SymPy-Methode `limit()` symbolisch berechnen lässt.

Kapitel 8, **»Stetige Funktionen«**, veranschaulicht die dynamische Veränderung von Parametern elementarer mathematischer Funktionen. Gezeigt wird, wie Sie mit dem Slider-Steuerelement die Kurvenformen von 2-D- und 3D-Funktionsplots dynamisch während der Programmlaufzeit ändern können. Wie Animationen von 2D- und 3D-Funktionsplots mit der Matplotlib-Methode `FuncAnimation()` erstellt werden können, wird zum Abschluss des Kapitels erklärt.

Kapitel 9, **»Differenzialrechnung«**, behandelt die klassischen Themen der Differenzialrechnung: Differenzenquotient, Differenzialquotient und Kurvendiskussion. Der Übergang einer Sekanten- zur Tangentensteigung wird als Grenzwertprozess mit dem Slider-Steuerelement simuliert. Es werden sowohl selbst erstellte Algorithmen als auch Methoden von SciPy und SymPy genutzt.

In Kapitel 10, **»Reihen«**, werden Reihenentwicklungen visualisiert. Veranschaulicht wird die Divergenz und Konvergenz von Reihen. Summen und Grenzwerte von Reihen werden mit den SymPy-Methoden `Sum()` und `limit()` symbolisch berechnet. Die SymPy-Methode `series()` berechnet Taylor-Reihen symbolisch. Das Konvergenzverhalten von Potenz- und Taylorreihen wird durch Funktionsplots veranschaulicht.

In Kapitel 11, **»Integralrechnung«**, werden die Begriffe Stammfunktion und bestimmtes Integral dynamisch mithilfe des Slider-Steuerelements veranschaulicht. Von den numerischen Integrationsverfahren wurden das Rechtecksummen-, das Trapezsummen- und das Simpson-Verfahren ausgewählt. Bogenlängen, Mantelflächen, Volumina von Rotationskörpern und Mehrfachintegrale werden mit selbst erstellten Algorithmen numerisch berechnet.

Kapitel 12, **»Differenzialgleichungen«**, beschreibt, wie Differenzialgleichung numerisch mit dem Euler-Verfahren und symbolisch mit der SymPy-Methode `dsolve()` gelöst werden können. Abschließend wird die Genauigkeit des Euler-Verfahrens an den Verfahren von Heun und Runge-Kutta gemessen.

Kapitel 13, **»Ausgleichsrechnungen«**, behandelt ausgewählte Themen der Ausgleichsrechnung wie lineare, exponentielle und potenzartige Regression. Für die Speicherung der Messdaten, für die Lösung der linearen Gleichungssysteme und für die erforderlichen Operationen auf Matrizen werden NumPy-Methoden verwendet.

In Kapitel 14, **»Algorithmen für die Berechnung statistischer Kennzahlen«**, werden Algorithmen für die Berechnung von Mittel- und Streuwerten sowie Formparameter wie Schiefe und Wölbung entwickelt. Da für die Berechnung des Medians die Messdaten sortiert vorliegen müssen, werden in einem Exkurs zwei einfache Sortieralgorithmen kurz beschrieben. Für die Berechnung der Spannweite wird in einem weiteren Exkurs gezeigt, wie das Maximum und Minimum aus einer Messreihe bestimmt werden kann.

In Kapitel 15, **»Fraktale«**, zeige ich Ihnen, wie Sie mithilfe der Turtle-Grafik von Python die Koch-Kurve, das Sierpinski-Dreieck und den Pythagoras-Baum zeichnen können. Bei der Darstellung

der Mandlbrot- und Julia-Menge wird demonstriert, wie aus einer einfachen rekursiven Gleichung mit komplexen Variablen komplizierte Fraktale entstehen.

Am Ende des Buches werden Sie in der Lage sein, Algorithmen zur Lösung Ihrer eigenen praktischen Problemstellungen in Python zu programmieren. Den Code für einen Algorithmus tatsächlich selbst zu schreiben und sein Verhalten unter verschiedenen Anfangsbedingungen zu beobachten, hilft enorm dabei, das Verständnis seiner Funktionsweise zu vertiefen.

1.1 Entwicklungsumgebungen

Eine Python-Entwicklungsumgebung (IDE, kurz für *Integrated Developer Environment*) besteht aus einem Editor, einem Interpreter und einem Debugger. Im Editor wird der Quelltext des Programms eingegeben. Durch einen Mausklick auf den Start- bzw. Run-Button oder durch Betätigung der Funktionstaste F5 wird das Python-Programm mit einem Interpreter übersetzt. Das Resultat wird in einem Ausgabefenster, in der Shell oder in einer anderen Konsole angezeigt. Wenn das Programm Syntaxfehler enthält, dann wird eine Fehlermeldung ausgegeben. Der Debugger hilft Ihnen bei der Suche nach Programmierfehlern. In der Praxis werden Programme oft debuggt, indem man Zwischenergebnisse in der Kommandozeile ausgeben lässt und überprüft, ob diese die Werte haben, die man erwartet. Es lohnt sich dennoch, den Umgang mit einem Debugger zu erlernen, da dieser oft viele hilfreiche Werkzeuge mitbringt, die Ihnen die Suche nach Fehlern deutlich erleichtern.

Es gibt eine Vielzahl von Python-Entwicklungsumgebungen. Für den Einstieg empfehle ich, eine einfach zu bedienende Umgebung wie *IDLE* oder *Thonny* zu benutzen. Selbstverständlich gibt es auch professionelle Entwicklungsumgebungen, die Sie vielseitig unterstützen, aber wegen ihres Funktionsumfangs machen sie den Einstieg eher schwieriger als einfacher. Für komplexe Projekte ist der Umgang mit professionellen Entwicklungsumgebungen zwar notwendig, erste Lernerfolge stellen sich aber mit einfachen Werkzeugen schneller ein.

IDLE und Thonny werden von den verbreiteten Betriebssystemen wie Linux, macOS und Windows unterstützt. Beide Entwicklungsumgebungen stehen im Internet kostenlos als Download zur Verfügung. Dort finden Sie auch die Installations- und Bedienungsanleitungen.

1.1.1 IDLE

Die Abkürzung IDLE steht für *Integrated Development and Learning Environment*. Bei der Installation von Python wird diese Entwicklungsumgebung automatisch mitinstalliert.

Wichtige Leistungsmerkmale sind:

- Interaktive Python-Shell (Konsolenprogramm) mit Einfärbung der Schlüsselwörter und Ausgabe von Fehlermeldungen.
- Text-Editor mit automatischer Einrückung bei Verzweigungs- und Schleifenkonstrukten sowie Einfärbung der Schlüsselwörter. Im Menü Edit gibt es unter anderem die Optionen Rückgängigmachen (Undo), Suchen (Find), Ersetzen (Replace) und Autovervollständigung (Expand Word).
- Debugger mit Haltepunkten und schrittweisem Ausführen des Programms.