Programmierung Grundlagen

mit Beispielen in Java und JavaScript

(Stand 2021)

PG_2021

Ralph Steyer

1. Ausgabe, Februar 2021

ISBN: 978-3-86249-942-7

1 Informationen zu diesem Buch

1.1 Voraussetzungen und Ziele

Zielgruppe

- Programmiereinsteiger
- Auszubildende in IT-Berufen
- Studenten in IT-Fächern und in Studiengängen mit IT-Bezug

Empfohlene Vorkenntnisse

Folgende Kenntnisse werden für eine erfolgreiche Benutzung dieses Buchs vorausgesetzt:

- Grundkenntnisse im Umgang mit Windows, MacOS oder Linux
- Grundkenntnisse in der Bedienung von Anwendungsprogrammen

Lernziele

Zu Beginn erhalten Sie einen Überblick über die bekanntesten Programmiersprachen und lernen Methoden und Werkzeuge für die Entwicklung von Programmen kennen.

Im weiteren Verlauf werden grundlegende Elemente von Programmiersprachen und deren Verwendung vorgestellt. Anschließend arbeiten Sie mit einigen elementaren Algorithmen.

Im letzten Kapitel erhalten Sie einen Überblick über die Grundlagen zur Softwareentwicklung, wie Sie für umfangreiche Software-Projekte benötigt werden, und Sie lernen die verschiedenen Methoden des Softwareentwurfs kennen.

Hinweise zur Software

Bei den Beispielen im Buch wird die Programmiersprache Java verwendet. Etliche Beispiele sind zusätzlich in JavaScript codiert.

- Im Buch wird von einer Erstinstallation der Software Java Platform, Standard Edition (Java SE), in der Version 15 ausgegangen, die Sie über das Internet downloaden können:

  (*https://www.oracle.com/java/technologies/javase-downloads.html*)

  Die Software wurde als Entwicklungsumgebung JDK (**J**ava **D**evelopment **K**it) heruntergeladen. Hauptbestandteile des JDK sind ein Java-Compiler zum Übersetzen der Programme, ein Java-Interpreter zum Ausführen der Programme, Bibliotheken mit fertigen Programmteilen und Programme zur Erstellung von Dokumentationen. Die meisten Java-Beispiele lassen sich aber auch anhand einer früheren Java-Version umsetzen. Ausnahmen sind Beispiele, die explizit Funktionalitäten neuer Java-Versionen einsetzen. Es wird empfohlen, mit der neusten Java-Version zu arbeiten.
- Als Editor wird der Freeware-Editor PSPad (*https://www.pspad.com/de/*) verwendet. Hinweise zur Installation und Konfiguration finden Sie im Anhang.
- Als Beispiel für eine komfortable Entwicklungsumgebung wird Eclipse (*https://www.eclipse.org/*) vorgestellt.
- Bei einigen Ausblicken werden bei Bedarf verschiedene weitere Programme und Tools verwendet.

- Das verwendete Betriebssystem ist Windows 10. Die Programme und Skripte funktionieren aber auch unter Linux oder MacOS.
- Als Referenz-Browser (für JavaScript) kommen Firefox 84 und Chrome 87 jeweils in der 64-Bit-Version unter Windows 10 zum Einsatz. Die Beispiele funktionieren aber auch in älteren oder anderen Browsern, sofern sie JavaScript unterstützen.

1.2 Aufbau und Konventionen

Hervorhebungen im Text

Im Text erkennen Sie bestimmte Programmelemente an der Formatierung. So werden z. B. Bezeichnungen für Programmelemente wie Register immer *kursiv* geschrieben und wichtige Begriffe **fett** hervorgehoben.

Kursivschrift	kennzeichnet alle von Programmen vorgegebenen Bezeichnungen für Schaltflächen, Dialogfenster, Symbolleisten, Menüs bzw. Menüpunkte (z. B. *Datei - Schließen*) sowie alle vom Anwender zugewiesenen Namen wie Dateinamen, Ordnernamen, eigene Symbolleisten, Hyperlinks und Pfadnamen.
`Courier New`	kennzeichnet Programmtext, Programmnamen, Funktionsnamen, Variablennamen, Datentypen, Operatoren etc.
`Courier New kursiv`	kennzeichnet Zeichenfolgen, die vom Anwendungsprogramm ausgegeben oder in das Programm eingegeben werden.
[ ]	Bei Darstellungen der Syntax einer Programmiersprache kennzeichnen eckige Klammern optionale Angaben.
|	Bei Darstellungen der Syntax einer Programmiersprache werden alternative Elemente durch einen senkrechten Strich voneinander getrennt.

Was bedeuten die Symbole im Buch?

 Hilfreiche Zusatzinformation

 Praxistipp

 Warnhinweis

HERDT BuchPlus - unser Konzept:

Problemlos einsteigen - Effizient lernen - Zielgerichtet nachschlagen

Nutzen Sie dabei unsere maßgeschneiderten, im Internet frei verfügbaren Medien:

Wie Sie schnell auf diese BuchPlus-Medien zugreifen können, erfahren Sie unter ***www.herdt.com/BuchPlus***

2 Grundlagen zu Programmen und Programmiersprachen

2.1 Grundlagen zu Programmen

Computer unterstützen Sie in vielen Bereichen des täglichen Lebens, z. B. bei der Informationsbeschaffung, bei Bankgeschäften, beim Einkaufen (Onlineshopping) oder im Unterhaltungsbereich.

Der Lösungsweg zur Bearbeitung einer Aufgabe wird dem Computer in Form eines Programms oder Skripts mitgeteilt.

Im Folgenden wird **Programm** als Synonym für Programm und Skript verwendet, wenn nicht ausdrücklich von einem Skript die Rede ist. Ein Programm wird auch **Applikation (engl. application)** oder **Anwendung** genannt. Die Kurzform der englischen Version von Applikation – **App** – ist vor allem bei der Programmierung für mobile Geräte gebräuchlich.

Vom Algorithmus zum Programm

Ein **Programm** bzw. Skript ist eine Beschreibung der Lösung einer vorgegebenen Aufgabe in einer spezifischen Programmiersprache (vgl. Abschnitt 2.3).

Die Lösung kann (und wird in der Regel) aus einzelnen Bearbeitungsvorschriften bestehen. Eine solche Bearbeitungsvorschrift wird **Algorithmus** genannt. Ein Algorithmus muss bei jeder möglichen Eingabe von Daten die Verarbeitung nach endlich vielen Schritten beenden und einen eindeutigen Ablauf besitzen. Ein Programm besteht aus Algorithmen, deren Arbeitsschritte in einer Programmiersprache (z. B. in Java, C#, JavaScript oder PHP) formuliert sind.

Was ist Software?

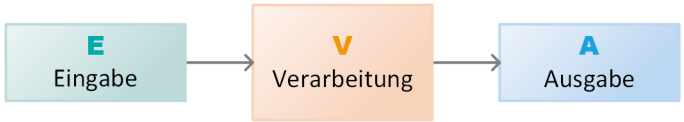
Der Begriff **Software** wird meist umfassender als der Begriff Programm verstanden. Als Software werden alle immateriellen Teile eines Computers verstanden. Das umfasst Programme, aber auch die zugehörigen Daten. Im täglichen Sprachgebrauch werden die Begriffe Software und Programm oft synonym verwendet.

Programme verarbeiten Daten

Ein Programm kann Ihnen beispielsweise Steuern berechnen. Sie teilen dem Programm die Daten mit, die für die Berechnung benötigt werden. Das Programm rechnet nach seinem Algorithmus mit Ihren Daten und liefert am Ende ein Ergebnis.

Programme verarbeiten **Daten**, z. B. Benutzereingaben, und liefern Daten zurück. Diesen Datenfluss durch ein Programm nennt man **EVA-Prinzip** (**E**ingabe – **V**erarbeitung – **A**usgabe):

- **Eingabe**: In das Programm werden Daten eingegeben, z. B. über eine Tastatur oder eine Datenbank.
- **Verarbeitung**: Das Programm verarbeitet diese Daten nach einem vorgegebenen Algorithmus.
- **Ausgabe**: Die Ergebnisse des Programms werden ausgegeben, z. B. auf einen Bildschirm, einen Drucker oder in eine Datenbank.



2.2 Warum programmieren?

Bestehende Programme decken viele Einsatzgebiete ab, wie z. B. die Verwaltung von Geschäftsprozessen, die Textverarbeitung, die Tabellenkalkulation, Spiele usw. Warum sollten Sie noch selbst programmieren?

Standard-Software anpassen

Standardisierte Software, die für ein bestimmtes Einsatzgebiet konzipiert wurde, stößt an ihre Grenzen, wenn spezielle Anforderungen an sie gestellt werden bzw. wenn die Anwendungsgebiete erweitert werden.

Beispiel

Sie haben sich mit einem Programm eine Datenbank mit Ihren Kundendaten aufgebaut. Später müssen Sie auch mit anderen Programmen auf diese Datenbank zugreifen. In diesen Programmen ist keine Möglichkeit vorgesehen, auf die Daten Ihrer Kundendatenbank zuzugreifen. Sie müssten also die Daten wieder manuell eingeben. Oft ist es dann einfacher, wenn Sie ein Programm schreiben, das den Zugriff auf die Daten ermöglicht.

Individual-Software

Hierbei handelt es sich um Software, die eigens für einen bestimmten Anwendungsbereich oder für spezielle Abteilungen innerhalb einer Firma erstellt wird.

Für verschiedene Branchen und Einsatzgebiete gibt es spezielle Anforderungen, sodass nur eine genau auf die Bedürfnisse abgestimmte Software infrage kommt. Individual-Software kann, auf Kundenwunsch hin, erweitert und verändert werden.

2.3 Klassifizierung von Programmiersprachen

Im Gegensatz zu natürlichen Sprachen, z. B. Deutsch, Englisch, gehören **Programmiersprachen** zu den **formalen Sprachen** (künstlichen Sprachen).

Programmiersprachen lassen sich nach verschiedenen Kriterien einordnen. Nach ihrer **historischen Entwicklung** werden verschiedene Generationen unterschieden:

- Erste Generation: Maschinensprachen (Maschinencode)
- Zweite Generation: Assembler-Sprachen
- Dritte Generation: Prozedurale Sprachen
- Vierte Generation: 4GL (**G**eneration **L**anguage)
- Fünfte Generation: Künstliche Intelligenz

Programmiersprachen unterscheiden sich erheblich in der zugrunde liegenden **Programmiertechnik**, auch Programmierparadigma genannt. Nach Programmiertechniken und Konzepten können die Programmiersprachen wie folgt klassifiziert werden, wobei diese Einteilung weder strikt zu trennen noch zwingend ist:

- Prozedurale Programmiersprachen
- Objektorientierte Programmiersprachen
- Hybride Programmiersprachen
- Skriptsprachen
- Funktionale Programmiersprachen
- Logische Programmiersprachen
- Erziehungsorientierte Programmiersprachen und Minisprachen

Sprachen lassen sich also nach verschiedenen Kriterien klassifizieren und zum Teil ist die Zuordnung zu einer bestimmten Gruppe nicht eindeutig.

Funktionale und logische Programmiersprachen werden auch als **deklarative Programmiersprachen** bezeichnet.

2.4 Die Klassifizierung nach Generationen

Erste Generation: Maschinensprachen (Maschinencode)

Damit ein Computer Probleme lösen kann, muss ihm der Lösungsweg in einer ihm verständlichen Art und Weise mitgeteilt werden. Eine Maschinensprache erfüllt genau diese Bedingung.

Beschreibung	Sowohl die Operationen als auch die Daten werden vom Programmierer ausschließlich als Folge aus Nullen und Einsen (vgl. Abschnitt 6.2) eingegeben. Eine übliche Operation ist z. B. der Transport von Daten aus dem Speicher in ein Register.
Nachteile	Für jeden Computer müssen die Maschinenbefehle neu entwickelt werden, da diese Sprache von den Eigenschaften der Hardware (Prozessoren) abhängig ist. Programme in Maschinensprache sind schwer lesbar und mit einem hohen Programmieraufwand verbunden. Deshalb wird diese Sprache heute kaum mehr direkt eingegeben (allerdings werden auch heute noch alle Befehle für Computer letztendlich in Maschinenbefehle umgewandelt).
Beispiel	`00011010 0011 0100`

Zweite Generation: Assembler-Sprachen

Beschreibung	Assembler-Sprachen sind wie Maschinensprachen an bestimmte Prozessoren gebunden. Übersetzungsprogramme, die Assembler-Programme in Maschinencode umwandeln, werden ebenfalls als **Assembler** bezeichnet.
Einsatz	Assembler-Sprachen werden überwiegend zur Programmierung der Hardware oder für schnelle, zeitkritische Programme eingesetzt.
Vorteile	Im Vergleich zur Maschinensprache bieten Assembler-Sprachen dem Programmierer durch Operationskürzel, z. B. `ADD` für addieren, wesentliche Erleichterungen. Da Assembler-Sprachen auf die maschinenspezifischen Besonderheiten des jeweiligen Computers abgestimmt sind, verbrauchen die Programme im Allgemeinen weniger Speicherplatz und sind meist auch schneller als ein entsprechendes Programm in einer anderen Programmiersprache.
Besonderheit	Die einzelnen Befehle der Assembler-Sprachen verwenden direkt die internen Befehle des Prozessors. Wer eine Assembler-Sprache erlernt, erfährt dabei viel über die Arbeitsweise des jeweiligen Prozessortyps.
Beispiel	ADD ax, 10

Dritte Generation: Prozedurale Sprachen

Anstoß zur Weiterentwicklung der Programmiersprachen gaben die mangelhafte Eignung der maschinenorientierten Sprachen zum Erstellen komplexer Anwendungsprogramme und die schlechte Lesbarkeit für Menschen.

Beschreibung	Prozedurale Sprachen sind (weitgehend) unabhängig von einem Computersystem. Da Programmiersprachen ab der dritten Generation vom spezifischen Computersystem abstrahieren, werden sie auch als **höhere Programmiersprachen** bezeichnet. Damit ein Computer ein Programm in einer höheren Programmiersprache versteht, muss das Übersetzungsprogramm (Compiler oder Interpreter) an das jeweilige System angepasst sein und den entsprechenden Maschinencode erzeugen.
Einsatz	Die Sprachen der dritten Generation sind in ihrer Struktur und ihrem Befehlsvorrat auf bestimmte Anwendungsbereiche zugeschnitten.
Vorteile	Höhere Programmiersprachen sind im Allgemeinen leichter zu erlernen als maschinenorientierte Sprachen. Der Programmcode kann auch bei anderen Rechnersystemen wieder verwendet werden – bei einigen (älteren) Sprachen müssen allerdings bei einer Portierung einige Anpassungen vorgenommen werden.
Nachteile	Das Programm der höheren Programmiersprache verbrauchte früher mehr Speicherplatz und war meist auch langsamer als das vergleichbare Maschinenprogramm. Durch optimierte Übersetzung verschwinden diese Nachteile aber bei vielen modernen höheren Programmiersprachen.
Programmiersprachen	Cobol, RPG, Fortran, Pascal, PL/1, Basic, Ada, C/C++, Java und viele mehr
Beispiel (Java)	`flaeche = laenge * breite;`

Vierte Generation: 4GL (Generation Language)

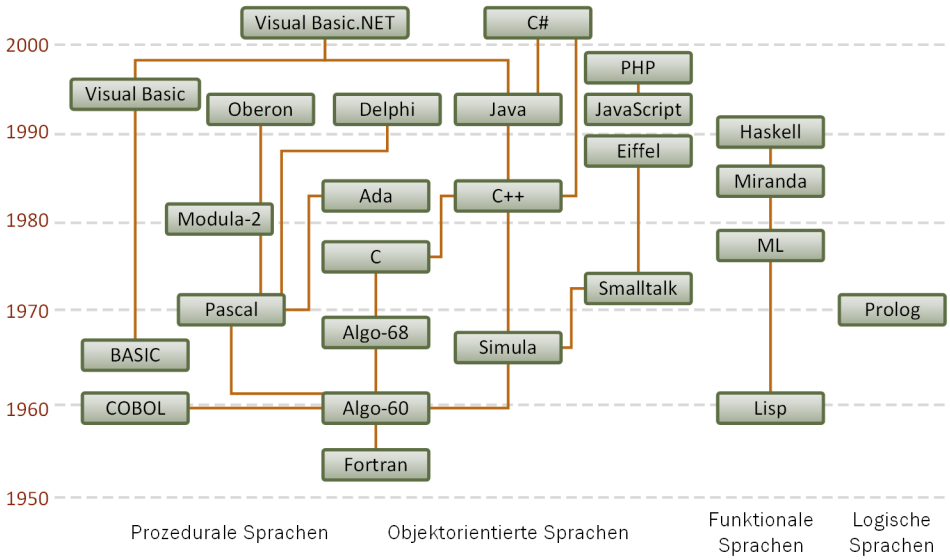
Während die ersten drei Generationen noch relativ klar voneinander getrennt werden können, fehlen bei der folgenden Generation eindeutige Kriterien.

Beschreibung	Bei 4GL-Programmiersprachen beschreibt der Programmierer lediglich, was das Programm leisten soll, ohne den genauen algorithmischen Weg anzugeben. Eine einzelne Anweisung löst eine ganze Folge von internen Einzelschritten aus.
Einsatz	4GL-Programmiersprachen sind auf spezielle Anwendungsgebiete ausgerichtet, z. B. auf das Bearbeiten und Auswerten von Dateien, Datenbanken, Tabellenkalkulationen oder auf das Erstellen von Bildschirmformularen.
Vorteile	Zum Programmieren stehen fortschrittliche, einfach zu bedienende und leistungsfähige Entwicklungssysteme zur Verfügung. Die Erstellung eines Programms wird dadurch wesentlich erleichtert.
Nachteile	Die größtmögliche Effizienz der Programme ist nicht gegeben. Da bestimmte Folgen von Arbeitsschritten innerhalb einer Anweisung automatisch ausgeführt werden, hat der Programmierer kaum einen Einfluss auf die internen Abläufe. Die Ausführungsgeschwindigkeit dieser Programme ist aufgrund der mächtigeren Sprache langsamer als bei prozeduralen Sprachen.
Programmiersprachen	SQL, Natural, Symphony, Open Access
Beispiel (SQL)	`CREATE Adresskartei` `SELECT Kunde FROM Tabelle WHERE KdNr = 10`

Fünfte Generation: Künstliche Intelligenz

Beschreibung	Der Grundgedanke des Forschungsgebietes der künstlichen Intelligenz ist es, zu untersuchen, unter welchen Bedingungen Computer menschliche Verhaltensweisen, die auf Intelligenz beruhen, nachvollziehen können. Für diese Forschungszwecke sind einige spezielle Programmiersprachen entwickelt worden. Diese Sprachen gehören meist zu den logischen oder funktionalen Programmiersprachen, die Sie im weiteren Verlauf dieses Kapitels kennen lernen.
Einsatz	Inzwischen umfasst dieses Gebiet mehrere Fachbereiche, beispielsweise die Robotik, die zur Entwicklung von Robotern und deren komplizierten Bewegungsabläufen geführt hat, die Wissensverarbeitung und die Spracherkennung.
Programmiersprachen	Prolog, Lisp, Smalltalk
Beispiel (Prolog)	`grossvater(X,Y) :- vater(X,Z), vater(Z,Y).`

2.5 Die Klassifizierung nach Sprachtypen



Entwicklung und Verwandtschaft verschiedener Programmiersprachen

2.6 Prozedurale Programmiersprachen

Als prozedurale Programmiersprachen werden höhere Programmiersprachen bezeichnet, die den Weg zur Problemlösung als eine Folge von Anweisungen angeben, die nacheinander (sequenziell) abgearbeitet werden.

Die folgenden Sprachen sind solche prozedurale Programmiersprachen.

```
BeginneProgramm
Anweisung1
Anweisung2
Anweisung3
...
BeendeProgramm
```

Die Programmiersprache Fortran

Bedeutung	Fortran steht für **For**mula **Tran**slator.
Entwicklung	Fortran wurde 1954 von IBM entwickelt und gilt als die erste Sprache der dritten Generation. In der aktuellen Version Fortran 2008 ist Fortran auch objektorientiert.
Einsatz	Fortran ist technisch-wissenschaftlich orientiert und wird hauptsächlich für mathematische oder technische Anwendungen eingesetzt.

Die Programmiersprache COBOL

Bedeutung	COBOL steht für **Co**mmon **B**usiness **O**riented **L**anguage.
Entwicklung	COBOL wurde speziell für betriebswirtschaftliche Anwendungen entwickelt und 1959 eingeführt. 1968 wurde COBOL durch das American National Standard Institute (ANSI) genormt. Seitdem wurde es kontinuierlich erweitert und modifiziert. Der letzte gültige Stand ist Cobol 2002. Diese Version enthält objektorientierte Erweiterungen.
Einsatz	COBOL ist eine noch immer weit verbreitete Programmiersprache. Im Bereich der kaufmännischen Großrechner sind weit über die Hälfte aller Anwendungen in COBOL geschrieben. Selbst auf dem PC werden betriebswirtschaftliche Anwendungen, die auf Großrechnerdaten zugreifen, in COBOL entwickelt.

Die Programmiersprache BASIC/Visual Basic

Bedeutung	BASIC steht für **B**eginners **A**ll-Purpose **S**ymbolic **I**nstruction **C**ode.
Entwicklung	BASIC wurde 1965 für Schulungszwecke entwickelt. Die Weiterentwicklung von BASIC führten viele Hersteller allerdings im Alleingang durch, weshalb für viele Rechnertypen eine eigene BASIC-Version entstand. 1991 wurde BASIC zu Visual Basic weiterentwickelt und ab 2002 auch in das .NET Framework, eine Entwicklungsumgebung für Windows-basierte Anwendungen von Microsoft, integriert.
Einsatz	Haupteinsatzbereich für BASIC waren Mikrocomputer. Die objektorientierte Programmiersprache Visual Basic 2008 wird zur Programmierung von Windows-basierten Anwendungen verwendet.

Die Programmiersprache C

Entwicklung	Die Programmiersprache C wurde Anfang der 70er-Jahre von B. W. Kernighan und D. M. Ritchie im Auftrag der Bell Laboratories während der Arbeit an dem Betriebssystem UNIX entwickelt.
Einsatz	C eignet sich für die Entwicklung von systemnahen Programmen. So ist beispielsweise das Betriebssystem UNIX in C geschrieben worden. Die letzte Variante der Sprache ist C99 von 1999.
Hinweise	C hat alle Vorteile einer höheren Programmiersprache. Gleichzeitig kann damit sehr hardwarenah programmiert werden, was sonst nur bei Assembler-Sprachen möglich ist. Bedingt durch die Normierung von C durch die ANSI-Kommission können Programme relativ leicht auf andere Rechnertypen übertragen (portiert) werden, sofern sich die Programmierer an die normierten Konventionen halten.

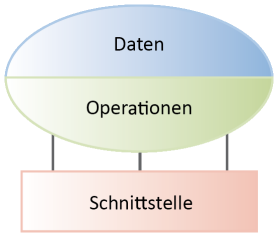
Die Programmiersprache Pascal

Bedeutung	Die Programmiersprache Pascal wurde nach dem französischen Philosophen Blaise Pascal (1623-1662) benannt.
Entwicklung	Entwickelt wurde Pascal 1971 von dem Schweizer Computerwissenschaftler Nikolaus Wirth, um die strukturierte Programmierung zu lehren. Bei der strukturierten Programmierung wird ein Programm in logische Einheiten zerlegt. Zur Steuerung des Ablaufs werden bestimme Kontrollstrukturen definiert, in Pascal Sequenz, Auswahl und Wiederholung (vgl. Kapitel 8). Eine besonders weite Verbreitung erfuhr Pascal durch die von Borland entwickelte Version TURBO Pascal und deren Nachfolger DELPHI, die eine leicht bedienbare Programmierumgebung bestehend aus Editor und Compiler besitzen und bezüglich ihres Befehlssatzes gegenüber der Standardversion von 1971 stark erweitert worden sind. Seit Pascal 7 hat Pascal auch objektorientierte Ansätze, die in DELPHI zu einer objektorientierten Sprache ausgebaut wurden.

2.7 Objektorientierte Programmiersprachen

Die Weiterentwicklung der Computer führte auch zur Entwicklung komplexerer Software. Entsprechend wurde die prozedurale Programmierung zur objektorientierten Programmierung weiterentwickelt. Das Problem der prozeduralen Programmierung ist, dass globale Daten in jedem Teil des Programms manipuliert und überschrieben werden können. Es fehlt eine Verbindung zwischen den Daten eines Programms und den sie manipulierenden Funktionen. Dies führt dazu, dass große Programme sehr leicht unübersichtlich werden und sich schwerer testen lassen.

1970 erkannte David Parnas das Problem und hatte die Idee, jedes einzelne Datenelement in einem Modul zu kapseln. Der direkte Zugriff auf diese Daten wurde nur über eine bestimmte Schnittstelle mit einem Satz von Operationen, wie z. B. über Prozeduren oder Funktionen, erlaubt. Sollen andere Module ebenfalls auf die Variable zugreifen, können sie dies nur indirekt über die Schnittstelle des Moduls tun.



Hier sind einige Beispiele für Sprachen, die nach diesem Konzept funktionieren.

Die Programmiersprache C++

Entwicklung	1982 begann Bjarne Stroustrup eine Erweiterung der prozeduralen Programmiersprache C zu entwickeln. 1989 wurde die Basissprache definiert, 1996 der internationale Standard (ISO/IEC 14882) verabschiedet. Die letzte Überarbeitung der Sprache wurde 2003 veröffentlicht.
Einsatz	C++ eignet sich für die Entwicklung von systemnahen Programmen und von komplexen Anwendungen.
Hinweise	Es gibt zur objektorientierten Programmierung mit C++ mächtige Programmierwerkzeuge. C++-Compiler stehen praktisch auf jeder Rechnerplattform zur Verfügung. C++ besitzt mächtige Sprachmittel, um komplexe Anwendungen zu erzeugen. Außerdem stehen umfangreiche Klassenbibliotheken zur Verfügung.

Die Programmiersprache Java

Entwicklung	Eine Gruppe von Ingenieuren bei Sun Microsystems entwickelte 1991 Software für interaktives Fernsehen und andere Geräte der Konsumelektronik. Diese Programmiersprache nannte sich „Oak“. Mit der zunehmenden Verbreitung des Internet wurde Oak dafür angepasst und in **Java** umbenannt. Java wurde an C/C++ angelehnt, wobei auf verschiedene (fehlerträchtige) Konstrukte verzichtet wurde.
Einsatz	Anwendungen, die auf unterschiedlichen Rechnersystemen laufen sollen, sowie für verteilte Anwendungen
Hinweise	Da nicht alle Sprachelemente von C++ realisiert sind, ist Java eine relativ schlanke und übersichtliche Sprache. Java-Compiler sind für Windows und Linux/Unix sowie MacOS kostenlos erhältlich und erzeugen maschinenunabhängigen Code, sogenannten Bytecode (vgl. Abschnitt 5.2). Dieser wird in einer virtuellen Maschine ausgeführt. Java besitzt eine umfangreiche Programmbibliothek für verschiedene Bereiche.

Bei den Programmcode-Beispielen in diesem Buch wird überwiegend die Programmiersprache Java verwendet.

.NET und die Programmiersprache C#

Entwicklung	Mit .NET bezeichnet Microsoft seine Software-Plattform zur Entwicklung und Ausführung von Anwendungsprogrammen. C# ist eine objektorientierte Programmiersprache für .NET. C# greift Konzepte der Programmiersprachen Java, C++, C und Delphi auf und wurde von Microsoft erstmals im Jahr 2000 standardisiert.
Einsatz	C# eignet sich besonders für eine bequeme Erstellung von sicheren, stabilen und gut an die Zielplattform angepassten Windows-Programmen. Aber auch für andere Betriebssysteme (etwa Linux) gibt es Laufzeitumgebungen.
Hinweise	.NET besteht aus einer Laufzeitumgebung (Common Language Runtime – CLR), in der die Programme ausgeführt werden, sowie einer Sammlung von Klassenbibliotheken, Programmierschnittstellen und Dienstprogrammen (analog Java und seiner virtuellen Maschine). .NET unterstützt neben C# die Verwendung weiterer Programmiersprachen. Unabhängig von der verwendeten Programmiersprache werden .NET-Programme in eine Zwischensprache (Common Intermediate Language – CIL) übersetzt, bevor sie von der Laufzeitumgebung ausgeführt werden (vgl. Abschnitt 4.2).

2.8 Hybride Programmiersprachen und Skriptsprachen

Die strenge Aufteilung in prozedurale und objektorientierte Sprachen ist bei vielen modernen Sprachen nicht möglich oder sinnvoll. Solche Sprachen können sowohl prozedural als auch objektorientiert eingesetzt werden. Man nennt diese Sprachen deshalb oft auch **hybrid**. Können hybride Sprachen mit Objekten umgehen, stellen jedoch nicht den vollen Umfang von objektorientierten Sprachen bereit, werden sie **objektbasiert** oder **objektbasierend** genannt. Gerade sogenannte Skriptsprachen sind oftmals sowohl objektbasierend als auch hybrid. Skriptsprachen sind in der Regel von einfacherer Natur und werden meist über einen Interpreter ausgeführt, der die Anweisungen erst zur Laufzeit in Maschinenbefehle übersetzt.

Programme, die in Skriptsprachen geschrieben sind, werden Skripte (engl. scripts) genannt. Teilweise wird auch die Bezeichnung **Makro** verwendet. Die Anwendungsgebiete und Eigenschaften konventioneller Programmiersprachen und Skriptsprachen überschneiden sich mittlerweile stark, weshalb eine strikte Trennung nur selten möglich ist. Nachfolgend sehen Sie einige Beispiele für solche Sprachen.

JavaScript

Entwicklung	**JavaScript** ist eine Skriptsprache, die ursprünglich von der Firma Netscape für die Erweiterung von HTML in Webbrowsern entwickelt wurde, um Benutzerinteraktionen auszuwerten und dynamisch Inhalte der Webseite zu verändern. Im Jahr 1995 erschien mit dem Netscape Navigator 2.0 der erste Browser mit einer eingebetteten Skriptsprache, die zu diesem Zeitpunkt **LiveScript** genannt wurde. Im Rahmen einer Kooperation zwischen Netscape und Sun Microsystems wurde LiveScript in JavaScript umbenannt. Mittlerweile unterstützen alle modernen Webbrowser JavaScript, wobei Microsoft einen eigenen Dialekt mit Namen **JScript** verwendet und Anwender JavaScript im Browser auch deaktivieren können.
Einsatz	JavaScript wurde früher fast ausschließlich clientseitig im Webbrowser eingesetzt. Heutzutage finden Sie JavaScript-Implementierungen aber auch auf Servern, im Umfeld von Java oder .NET, in Datenbanken, Microcontrollern oder bei Apps.
Hinweise	**ECMAScript** (ECMA 262) bezeichnet den standardisierten Sprachkern von JavaScript. Die Syntax von JavaScript basiert auf C und stimmt in großen Bereichen mit der von Java überein. Aber trotz der Namens- und syntaktischen Ähnlichkeit hat JavaScript **konzeptionell** nur geringe Gemeinsamkeiten mit Java. JavaScript ist nicht streng objektorientiert, sondern objektbasierend und verwendet etwa bei der Vererbung Prototypen statt Klassen. Man kann mit JavaScript sowohl prozedural als auch rein objektorientiert arbeiten, wenn man dabei gewisse Konventionen einhält, die aber nicht zwingend sind.

Bei den Programmcode-Beispielen in diesem Buch wird teilweise JavaScript verwendet. Diese Skripte können in jedem modernen Browser ausgeführt werden, sofern nicht JavaScript deaktiviert ist.

Python

Entwicklung	**Python** ist eine universelle, höhere Programmiersprache, die üblicherweise interpretiert wird. Es gibt also in der Laufzeitumgebung von Python einen Interpreter samt notwendiger weiterer Ressourcen. Python unterstützt sowohl die objektorientierte, die aspektorientierte, die strukturierte als auch die funktionale Programmierung. Das bedeutet, Python zwingt den Programmierer nicht zu einem einzigen Programmierstil, sondern erlaubt das für die jeweilige Aufgabe am besten geeignete Paradigma zu wählen. Objektorientierte und strukturierte Programmierung werden vollständig unterstützt, funktionale und aspektorientierte Programmierung werden zumindest durch einzelne Elemente der Sprache unterstützt. Ein zentrales Feature ist in Python die dynamische Typisierung samt dynamischer Speicherbereinigung. Damit kann man Python auch als reine Skriptsprache nutzen.
Einsatz	Das Einsatzgebiet von Python ist breit gestreut. Man kann damit so gut wie jede Form von Applikation schreiben und es gibt für die wichtigsten Betriebssysteme Implementierungen.
Hinweise	Python hat seine Grundlagen in C und ist den meisten gängigen Programmiersprachen verwandt, wurde aber mit dem Ziel größter Einfachheit und Übersichtlichkeit entworfen. Zentrales Ziel bei der Entwicklung der Sprache ist die Förderung eines gut lesbaren, knappen Programmierstils. So wird beispielsweise der Code nicht durch geschweifte Klammern (wie in fast allen anderen C-basierenden Sprachen), sondern durch zwingende Einrückungen strukturiert. Zudem ist die gesamte Syntax reduziert und auf Übersichtlichkeit optimiert. Wegen dieser klaren und überschaubaren Syntax ist Python einfach zu erlernen, zumal die Sprache mit relativ wenigen Schlüsselwörtern auskommt. Es wird immer wieder zu hören sein, dass sich Python-basierte Skripte deutlich knapper formulieren lassen, als es in anderen Sprachen der Fall ist. In vielen Ländern hat Python bei Anfängerkursen in informatikbezogenen Studiengängen an Universitäten Java, was über viele Jahre die Szene beherrscht hatte und im professionellen Umfeld immer noch das Maß aller Dinge darstellt, abgelöst.

PHP

Entwicklung	**PHP** (rekursives Akronym für „PHP: Hypertext Preprocessor“ – ursprünglich Personal Home Page Tools) ist eine Skriptsprache, die 1995 von Rasmus Lerdorf entwickelt wurde. PHP war ursprünglich als Ersatz für eine Sammlung von Perl-Skripten gedacht, hat sich aber mittlerweile zu einer sehr verbreiteten Sprache bei der serverseitigen Webprogrammierung entwickelt.
Einsatz	PHP wird hauptsächlich zur serverseitigen Erstellung dynamischer Webseiten oder Webanwendungen verwendet. PHP ist OpenSource und zeichnet sich durch breite Datenbankunterstützung sowie die Verfügbarkeit zahlreicher Funktionsbibliotheken aus.
Hinweise	Die Syntax von PHP basiert auf C. Anfangs war PHP nicht objektorientiert, sondern rein prozedural. Mittlerweile wurde PHP erweitert, so dass Sie bei Bedarf mit PHP rein objektorientiert arbeiten können. PHP ist somit eine hybride Sprache, wobei neuere Sprachversionen immer mehr in Richtung Objektorientierung tendieren.

2.9 Funktionale und logische Programmiersprachen

Funktionale Programmiersprachen

Bei funktionalen Programmiersprachen ist das Programm eine Funktion, die sich typischerweise auf einfachere Funktionen stützt, daher auch der Name „funktionale Programmiersprache“. Die Beziehungen zwischen den Funktionen sind einfach: Eine Funktion kann eine andere aufrufen, oder das Ergebnis einer Funktion kann als Parameter für eine andere Funktion genutzt werden.

Programme werden wie mathematische Funktionen geschrieben. Eine Funktion hat einen Definitions- und einen Wertebereich. Die Funktion erhält einen Eingabewert und berechnet, mathematisch gesehen, den Wert der Funktion.

Dies sind Beispiele für solche Sprachen.

Die Programmiersprache LISP

Entwicklung	Die Programmiersprache LISP wurde von John McCarthy und einer Arbeitsgruppe am Massachusetts Institute of Technology Anfang der 60er-Jahre entwickelt. Es war die erste implementierte funktionale Sprache. Heute gibt es viele Dialekte von LISP, wie z. B. die Sprache Scheme.
Einsatz	LISP wird in der Informatik im Forschungsbereich eingesetzt und dort vorwiegend im Bereich der künstlichen Intelligenz und zur Lösung mathematischer Probleme.
Vorteil	Die Sprache ermöglicht es, komplexe Programme, die Zeichenketten bearbeiten, leichter als in anderen Programmiersprachen zu schreiben.
Beispiel zur Fakultätsberechnung	`>(defun fakultaet (x)` → Antwort: Funktionsdefinition `  (if ( x > 0) (* x (fakultaet (- x 1))) 1 ) )` `FAKULTAET` → Antwort: definierter Funktionsname `>(fakultaet 5)` → Funktionsaufruf mit dem Wert 5 `120` → Antwort: errechneter Wert