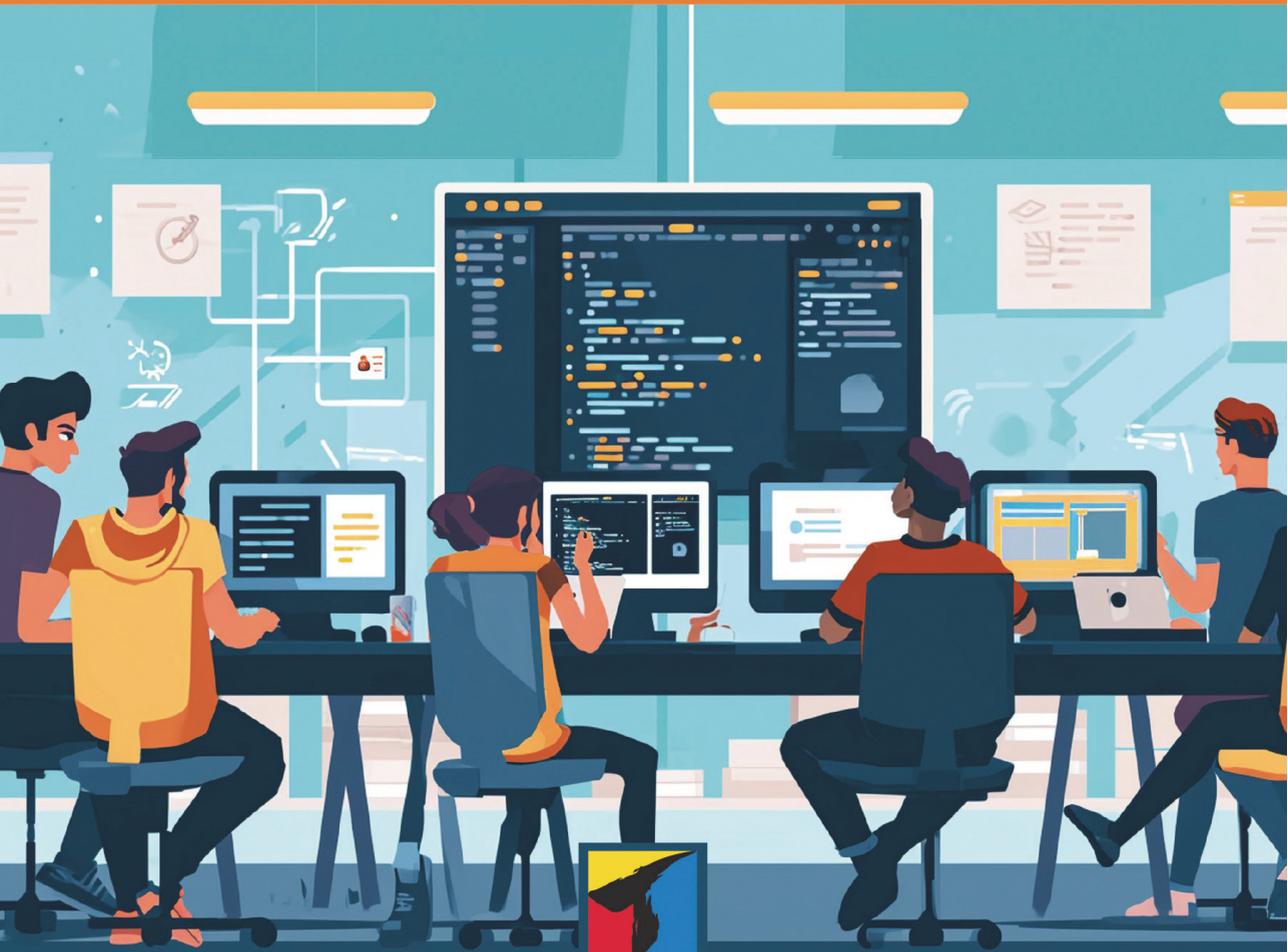


Andreas Hofmann

Programmieren lernen

Grundlagen für Studium und Beruf –
praxisnah und sprachunabhängig



Inhaltsverzeichnis

	Einleitung	9
1	Einführung in die Welt der Computerprogramme	13
1.1	Eine kleine Reise durch die Geschichte der Programmierung	13
1.2	Wie funktioniert das Programmieren von Computern?	15
1.2.1	Kompilierte Programmiersprachen	16
1.2.2	Interpretierte Programmiersprachen	17
1.2.3	Kompilieren vs. Interpretieren	18
1.2.4	Bytecode und Laufzeitumgebungen	20
2	Daten abbilden und miteinander in Beziehung setzen	23
2.1	Mit einer Maschine kommunizieren	23
2.2	Informationen speichern	25
2.2.1	Variablen	25
2.2.2	Datentypen	27
2.2.3	Deklaration, Initialisierung und Zuweisungen	34
2.2.4	Arrays und Collections	38
2.3	Operatoren und Ausdrücke	40
2.3.1	Mathematische Operatoren	40
2.3.2	Vergleiche	42
2.3.3	Zuweisungen	43
2.3.4	Logische Operatoren	44
2.3.5	Operatoren-Ränge und verschachtelte Ausdrücke	49
3	Den Programmfluss kontrollieren	51
3.1	Verzweigungen	51
3.1.1	Bedingte Verzweigung	51
3.1.2	Mehrfachverzweigung	56
3.2	Schleifen	60
3.2.1	Bedingte Schleifen	60
3.2.2	Zählschleife	63
3.2.3	Mengenschleife	65
3.2.4	Schleifen flexibler steuern	66
3.2.5	Häufige Fehler im Umgang mit Schleifen	67
3.3	Funktionen	68
3.3.1	Gültigkeitsbereich (Scope)	71
3.3.2	Rekursion	72

3.4	Typumwandlungen	75
3.5	Algorithmische Probleme lösen	76
3.5.1	Sternchen-Dreieck	77
3.5.2	Mehr Sternchen-Dreiecke	80
3.5.3	Primzahlen berechnen	82
3.5.4	Lineare Suche	84
3.5.5	Binäre Suche	88
3.5.6	Ein Array sortieren	92
4	Objektorientierte Programmierung	97
4.1	Konzept der Objektorientierung	98
4.2	Klassen und Objekte	99
4.2.1	Attribute	99
4.2.2	Methoden	100
4.2.3	Instanzen	102
4.2.4	Kapselung	107
4.2.5	Statische Attribute und Methoden	111
4.3	Vererbung	114
4.3.1	Konstruktoren verketteten	118
4.3.2	Methoden überschreiben	119
4.3.3	Abstrakte Klassen und Methoden	120
4.3.4	Mehrfachvererbung	123
4.4	Interfaces	124
4.4.1	Interfaces definieren	125
4.4.2	Interfaces verwenden	126
4.4.3	Interfaces anwenden	128
4.5	Übungsbeispiele	131
4.5.1	Schulnoten	131
4.5.2	Onlineshop	132
5	Werkzeuge für die Programmierung	135
5.1	Onlinehilfe	135
5.1.1	Sprachdokumentation	135
5.1.2	Stack Overflow und Stack Exchange	136
5.2	Die Entwicklungsumgebung	137
5.2.1	Gängige Funktionen	137
5.2.2	Beliebte IDEs	139
5.3	Arbeiten im Team	144
5.3.1	Versionskontrolle	144
5.3.2	Code-Style-Guides und Coding-Standards	149
5.4	Dokumentation	150
5.4.1	Kommentare	150

5.4.2	Separate Dokumentation	152
5.4.3	UML-Diagramme	153
5.5	Auf Fehlersuche	155
5.5.1	Arten von Fehlern	155
5.5.2	Unit-Tests	158
5.5.3	Fehlermanagement im Quellcode	160
5.5.4	Debugging	163
5.5.5	Softwaretests	165
6	Programmieren in der Praxis	169
6.1	Compiler und Entwicklungsumgebung installieren	169
6.1.1	Ein Projekt anlegen	172
6.1.2	Das erste Programm: »Hello, World!«	174
6.2	Beispiele in die Praxis umsetzen	175
6.2.1	Ausgabe und Einlesen von Daten	175
6.2.2	Sternchen-Dreiecke	178
6.2.3	Potenzfunktion	180
6.2.4	Berechnung der Fakultät	182
6.2.5	Binäre Suche	183
6.2.6	Bubblesort	192
6.2.7	Klassen erstellen und verwenden	192
7	Welche Programmiersprache ist die richtige für mich?	201
7.1	Aufgabengebiete der Programmierung	201
7.1.1	Desktop-Programme, Konsolen- und Serveranwendungen	201
7.1.2	Webprogrammierung	203
7.1.3	Datenbanken	206
7.1.4	Mobile Applikationen	208
7.1.5	Skripte	209
7.1.6	Mikrocontroller/Hardwareprogrammierung	211
7.1.7	Spiele und andere Echtzeit-Rendering-Anwendungen	212
7.1.8	Komponenten, Bibliotheken, Frameworks und SDKs	214
7.2	Ein kleiner Überblick zu den Programmiersprachen	215
7.2.1	C/C++	215
7.2.2	C# (C Sharp)	216
7.2.3	CSS	217
7.2.4	HTML	217
7.2.5	Java	218
7.2.6	JavaScript	218
7.2.7	PHP	219
7.2.8	Python	219

A	Lösungen	223
A.1	Lösungen zu Kapitel 2	223
	A.1.1 Negieren von Ausdrücken	223
	A.1.2 Reihenfolge der Operatoren	224
A.2	Lösungen zu Kapitel 4	226
	A.2.1 Schulnoten	226
	A.2.2 Onlineshop	230
A.3	Lösungen zu Kapitel 6	236
	A.3.1 Binäre Suche und Bubblesort	236
	A.3.2 Smart-Home	239
B	Die Welt aus 0 und 1	245
B.1	Zahlensysteme	245
	B.1.1 Vom Binärsystem ins Dezimalsystem umrechnen	246
	B.1.2 Vom Dezimalsystem ins Binärsystem umrechnen	248
	B.1.3 Weitere Zahlensysteme	248
B.2	Von Bits und Bytes	249
	B.2.1 Einheitenpräfixe	250
C	Glossar	253
	 Stichwortverzeichnis	 263



Einleitung

Das Programmieren ist zwar noch eine relativ junge Disziplin – etwa im Vergleich zur Fotografie oder dem Maschinenbau –, allerdings ist die Informationstechnologie ein sehr schnelllebiges Gebiet. Das Programmieren von Computerprogrammen ist also von diesem Standpunkt aus gesehen schon recht alt. Verändert hat sich dennoch recht wenig. Das Ziel ist immer noch dasselbe, nämlich einer Maschine beizubringen, bestimmte Aufgaben zu erfüllen. Zwar haben sich die Methoden weiterentwickelt, die Grundbausteine eines Programms sind aber immer noch dieselben und auch der grundlegende technische Mechanismus (siehe Abschnitt 1.2 »Wie funktioniert das Programmieren von Computern?«) ist gleich geblieben.

Die Entwicklungen auf dem Gebiet der Software-Programmierung haben, neben neuen technischen Möglichkeiten der Maschinen, lediglich ein schnelleres und leichteres Arbeiten des Programmierers zur Folge. Selbst die künstliche Intelligenz (KI), die gerade in aller Munde ist, revolutioniert das Programmieren – zumindest vorerst – nicht. Künstliche Intelligenz ist, anders als der Name vermuten lässt, nämlich gar nicht so intelligent und auch nichts anderes als ein von Menschen erschaffenes Programm.

Wer heute Programmieren lernen will, muss also immer noch beinahe dieselben Grundlagen lernen wie jemand, der vor einigen Jahrzehnten programmieren gelernt hat.

Das Ziel dieses Buchs

Dieses Buch richtet sich an Programmieranfänger und ist als Einstieg für jene gedacht, die von Grund auf Programmieren lernen wollen. Der Fokus liegt dabei auf der Vermittlung der grundlegenden Konzepte und Bausteine, die Sie als Programmierer oder Programmiererin beherrschen müssen, nicht auf einer bestimmten Programmiersprache.

Es dient aber auch als Orientierung für die weiteren Schritte und Themen auf ihrem Lernpfad und als Entscheidungshilfe für die Wahl einer oder mehrerer Programmiersprachen.

Sehen Sie dieses Buch als ersten Schritt auf dem Weg zum Programmierer. Es ist kein »Komplettwerk für Programmierer«, denn so etwas gibt es nicht. Das Programmieren von Software ist eine derart umfangreiche Disziplin, dass alle dafür nötigen Themen unmöglich in einem Buch abgedruckt werden können. Bücher können ohnehin nur gewisse Grundlagen vermitteln. Die Fähigkeit zu programmieren erwirbt man viel mehr durch die Praxis und die Erfahrung, die man sammelt, wenn man Programmcode schreibt. Dafür erhalten Sie mit diesem Buch eine solide Grundlage.

Das Buch beinhaltet keine fertigen Lösungen für bestimmte Problemstellungen und Aufgaben. Diese finden Sie in sogenannten *Cookbooks* (Kochbüchern) oder noch umfangreicher und aktueller im Internet.

Auch wenn dieses Buch keinerlei Vorwissen zum Thema »Programmierung« erfordert, so setzt es doch ein gewisses IT-Basiswissen voraus oder benötigt zumindest eine gewisse IT-Affinität und Lernbereitschaft, um sich das notwendige Wissen aus anderen Quellen anzueignen.

Programmieren lernen

Wie lernt man den nun Programmieren? Wie bereits erwähnt, ist das Programmieren bzw. Schreiben von Computerprogrammen eine komplexe Disziplin, denn das Erlernen einer Programmiersprache macht nur einen geringen Teil aus. Wichtiger sind die grundlegenden Konzepte, die die Programmierung ausmacht. Möchte man Programmierer werden, ist es viel essenzieller, eine gewisse Denkweise zu erlernen. Einen guten Programmierer zeichnet insbesondere eine hohe Problemlösungskompetenz aus.

Natürlich stellt das Erlernen einer Programmiersprache eine nicht zu unterschätzende Hürde für jeden Anfänger dar, haben Sie diese aber erst einmal gemeistert, lassen sich weitere Sprachen mit erheblich weniger Aufwand erlernen. Man muss eine Sprache auch nicht bis ins kleinste technische Detail kennen oder jeden Befehl in- und auswendig beherrschen, um erfolgreich Computerprogramme zu schreiben.

Programmieren ist keine Inseldisziplin, es ist nur ein Werkzeug, um Probleme zu lösen. Dafür ist, je nach Art des Problems, aber noch weiteres Wissen und Können erforderlich. Sie können die bestausgestattete Werkstatt Ihr Eigen nennen, ohne das nötige Fachwissen werden Sie kein Auto reparieren können. Nehmen Sie sich also etwas Zeit und lernen Sie, wie Sie Problemstellungen mit dem Schreiben von Programmen lösen. Mit diesem Buch liefere ich Ihnen die Grundlagen und die notwendige Denkweise hierfür.

Aufbau des Buchs

In diesem Buch finden Sie nicht nur Grundwissen zur Programmierung, sondern auch die nötigen Grundbausteine, um selbst ein Computerprogramm zu schreiben. Die entsprechenden Kapitel zielen auf ein allgemeines Verständnis ab, ohne an eine konkrete Programmiersprache gebunden zu sein. Dennoch soll die Praxis nicht zu kurz kommen. Im Praxisteil erfahren Sie, wie Sie das Gelernte in ein tatsächliches Stück Software verwandeln. Auch wenn ich hierfür auf eine konkrete Programmiersprache zurückgreifen muss, so können Sie die Techniken in jeder beliebigen anderen Programmiersprache ausprobieren.

- **Theorie (Kapitel 1 bis 4):** Sie erfahren zuerst, wie das Programmieren von Maschinen ganz grundsätzlich funktioniert. Anschließend lernen Sie die Basiselemente eines Computerprogramms kennen sowie weitere Methoden und Konzepte, die in der modernen Softwareentwicklung unerlässlich sind.
- **Praxis (Kapitel 5 bis 7):** Sie lernen anschließend, wie der Arbeitsprozess eines Programmierers aussieht, welche Werkzeuge und Abläufe notwendig sind und wie Sie sich das Programmieren einfacher gestalten können. Zum Abschluss zeige ich Ihnen, wie Sie das Gelernte in tatsächlichen Quellcode umsetzen.

Kapitel 2 »Daten abbilden und miteinander in Beziehung setzen«, Kapitel 3 »Den Programmfluss kontrollieren« und Kapitel 4 »Objektorientierte Programmierung« bauen aufeinander auf. Kapitel 6 »Programmieren in der Praxis« setzt das erworbene Wissen aus Kapitel 2 bis Kapitel 4 in die Praxis um, daher empfehle ich Ihnen, diese Kapitel in der vorgesehenen Reihenfolge zu lesen.

Viele Themen der Programmierung sind miteinander verknüpft und lassen sich nicht immer eindeutig in einer gewissen Reihenfolge beschreiben. Sie finden an den entsprechenden Stellen immer Querverweise auf die passenden Abschnitte dieses Buchs. Manchmal kann es auch sinnvoll sein, nach einem Kapitel nochmals zu vorangegangenen Abschnitten zurückzukehren und diese mit mehr Wissen erneut zu beleuchten.

Die Kapitel 1 »Einführung in die Welt der Computerprogramme«, Kapitel 5 »Werkzeuge für die Programmierung« sowie Kapitel 7 »Welche Programmiersprache ist die richtige für mich?« liefern Zusatzwissen, die zum Erlernen der Programmierung nicht direkt erforderlich, aber durchaus hilfreich und wissenswert sind. Sie können sie in beliebiger Reihenfolge lesen.

Konventionen in diesem Buch

Beim Programmieren dreht sich alles um Programmcode. Dieser besteht aus speziellen Schlüsselwörtern und Befehlen, die im Text durch eine eigene Schriftart hervorgehoben sind.

Programmcode, Befehle und spezielle Ausdrücke können aber auch in einem eigenen Absatz stehen.

Mehrere Befehle, die zusammengehören bzw. einen Codeabschnitt darstellen, sind ebenfalls in einem eigenen Absatz formatiert und mit Zeilennummern versehen (siehe Listing). Auch moderne Quellcode-Editoren verwenden nummerierte Zeilen, da das den Verweis auf einzelne Befehle erleichtert. Speziell in Fehlerfällen findet sich eine Zeilennummer schneller als ein bestimmter Befehl.

```
1 namespace HelloWorld
2 {
3     internal class Program
4     {
5         static void Main(string[] args)
6         {
7             Console.WriteLine("Hello, World!");
8         }
9     }
10 }
```

Listing 1: So wird zusammengehörender Quellcode im Buch formatiert.

Hin und wieder finden Sie im formatierten Quellcode ergänzende Kommentare, diese werden beginnend mit zwei Schrägstrichen (Slashes) gekennzeichnet. In etwa so: `//ich bin ein Kommentar`.

Kommentare helfen, bestimmte Stellen im Quellcode näher zu erläutern bzw. deren Auswirkung zu zeigen, ohne im Fließtext auf jedes Detail eingehen zu müssen.

Manchmal würden Codeabschnitte aus mehreren Zeilen Code bestehen, aber nicht immer sind alle davon relevant für ein Beispiel bzw. würden von den eigentlichen Befehlen ablenken. Nicht relevante Zeilen und Abschnitte werden in den Beispielen ausgelassen und durch `...` ersetzt.

Manchmal werden Beispiele laufend erweitert, auch hier ist es nicht sinnvoll, immer wieder den kompletten Code abzdrukken, also werden die Stellen aus vorherigen Listings zusammengekürzt und ebenfalls durch `...` ersetzt.

Mehr Wissen

In diesen Boxen finden Sie Zusatzwissen, das zu einem gewissen Themenbereich gehört, aber für Anfänger noch zu speziell oder einfach weniger relevant ist. Die Themen in diesen Boxen werden von mir nur grob umrissen und können als Grundlage für eigene Recherchen dienen.

Auch Sachverhalte, die sich in den einzelnen Programmiersprachen stärker unterscheiden oder bei bestimmten Sprachen eine Ausnahme darstellen, finden Sie in diesen Boxen.

Einführung in die Welt der Computerprogramme

Bevor ich Sie mit den Grundlagen und Konzepten der Programmierung vertraut mache, erhalten Sie in diesem Kapitel eine allgemeine Einführung zum Thema Programmieren. Dazu gehört ein kurzer Abriss aus der Geschichte, hier erfahren Sie, wie es zur Erfindung von Computerprogrammen kam und wie sich Programme und Programmiersprachen über die Jahrzehnte entwickelt haben. Danach erkläre ich Ihnen kurz, wie Programmiersprachen und Programmcode funktioniert und anschließend bekommen Sie eine Übersicht über die Einsatzgebiete von Computerprogrammen sowie über die gängigen Programmiersprachen.

1.1 Eine kleine Reise durch die Geschichte der Programmierung

Auch wenn wir heute auf Computern programmieren, überlegt man kurz, wird klar, dass ein Computer nur funktioniert, weil darauf ein Programm läuft, das das Arbeiten mit dem Computer erst möglich macht. Programmiersprachen, Programme und das Schreiben eben jener wurde also vor dem modernen Computer erfunden. Die Programmierung von Maschinen ist älter, als viele denken.

Vor den Computern gab es Rechenmaschinen und vor den Rechenmaschinen gab es nur mechanische Maschinen. Diese Maschinen waren in der Lage, bestimmte Aufgaben oder Arbeitsschritte automatisch zu erledigen. Die Funktion basiert auf deren Bauweise bzw. der mechanischen Konstruktion. Die Rede ist von industriellen Maschinen wie etwa Stanzen und Pressen, aber auch komplexeren Konstruktionen, wie etwa mechanischen Webstühlen. Aber egal wie komplex die Arbeitsschritte auch waren, die Ausführung erfolgte ohne jegliche Logik. Zwar konnten Werkzeuge und Aufsätze getauscht werden, dafür war aber ein menschliches Eingreifen erforderlich. Um andere Arbeitsschritte auszuführen, benötigte man eine andere mechanische Konstruktion. So konnten die ersten mechanischen Webstühle immer nur ein bestimmtes Muster weben.

Zu Beginn des 19. Jahrhunderts erfand der Franzose Joseph-Marie Jacquard den programmierbaren Webstuhl. Und auch wenn dieser noch keine Rechenmaschine war, folgte dieser dem Grundprinzip der elektronischen Datenverarbeitung: Ein-

gabe – Verarbeitung – Ausgabe (EVA). Über Lochstreifen konnte dem Webstuhl das zu webende Muster übermittelt werden.

Die erste mechanische Rechenmaschine wurde 1837 von Charles Babbage erfunden. Als erste mathematisch-logische Programmierung gilt die Vorschrift zur Berechnung von Bernoulli-Zahlen¹ von Ada Lovelace, die als erste Programmiererin der Geschichte gilt.

Zu Beginn des 20. Jahrhunderts wurden weitere Rechenmaschinen entworfen und dafür wiederum Vorschriften zum Lösen mathematischer Probleme entwickelt. Als Meilensteine gelten die vom deutschen Bauingenieur Konrad Zuse erfundenen und gebauten Rechenmaschinen. In den Vierzigerjahren des 20. Jahrhunderts griff der österreichisch-ungarische Mathematiker John von Neumann einige Ideen Zuses auf und beschrieb mit der Von-Neumann-Architektur ein Referenzmodell für Computer. Diese bildet die Grundlage für die meisten der heute bekannten Computer.

Nach dem Zweiten Weltkrieg entstanden, in den USA, die ersten höheren Programmiersprachen FORTRAN, Lisp und COBOL. Die an der Entwicklung von COBOL beteiligte Grace Hopper entwickelte auch den ersten Compiler (Abschnitt 1.2.1 »Kompilierte Programmiersprachen«). In den Sechzigern und Siebzigern des vergangenen Jahrhunderts wurde eine Vielzahl weiterer Programmiersprachen entwickelt. Grundlage war der technische Fortschritt in der Entwicklung von Computer-Hardware. Viele dieser Sprachen sind längst vergessen, andere werden noch heute verwendet oder entwickelten sich zu einigen der aktuellen Sprachen weiter. Nennenswerteste Vertreter sind die Sprachen BASIC, welche durch die ersten leistbaren Heimcomputer der Siebziger populär wurde, sowie C, welche 1972 für das neue Betriebssystem Unix entwickelt wurde.

Ende der Siebzigerjahre stellte das US-Verteidigungsministerium erschrocken fest, dass über 450 teils nicht standardisierte Programmiersprachen in deren Projekten genutzt wurden. Der Versuch, eine Sprache zu finden, die alle Anforderungen des US-Militärs erfüllte, scheiterte. Das Militär entwickelte daraufhin eine eigene Sprache, Ada, benannt nach Ada Lovelace.

1983 stellte Bjarne Stroustrup C++ vor, eine Erweiterung von C, die jene neuen Konzepte enthielt, die seit der Erfindung von C Einzug in die Computerprogrammierung gehalten hatten. Das Wichtigste hierbei ist die objektorientierte Programmierung, bei der die Architektur eines Programms sich an jenen Objekten der wirklichen Welt anlehnt, die die Aufgabenstellung des Programms betreffen (siehe Kapitel 4 »Objektorientierte Programmierung«).

1 Als Bernoulli-Zahlen wird eine Reihe von rationalen Zahlen bezeichnet, die in verschiedenen mathematischen Bereichen Anwendung finden.

Die schnelle Ausbreitung des Internets stellte eine ganze eigene Herausforderung dar, denn plötzlich mussten Inhalte auf eine Vielzahl verschiedener Endgeräte angepasst werden. Der Erfolg des Internets begründet sich unter anderem auf HTML – keine Programmiersprache, sondern eine Auszeichnungssprache –, welche die Gestaltung von Webinhalten übernahm, und PHP, der Programmiersprache für Serverlogik und dynamische/interaktive Inhalte.

1995 folgte die plattformunabhängige, objektorientierte Programmiersprache Java und 2001 die von Microsoft beauftragte Sprache C#.

Die nachfolgenden Entwicklungen brachten Sprachen hervor, die entweder Schwachstellen in den bereits verbreiteten Sprachen beheben sollten oder die für spezielle Einsatzgebiete gedacht waren (mehr dazu siehe Kapitel 7 »Welche Programmiersprache ist die richtige für mich?«).

1.2 Wie funktioniert das Programmieren von Computern?

Was ist Programmieren eigentlich? In der deutschen Sprache spricht man vom Programmieren, wenn man sein Haushaltsgerät oder ein anderes elektronisches Gerät konfiguriert. Wer alt genug ist, hat vielleicht schon mal einen Videorekorder »programmiert«. Unter Software- oder Computerprogrammierung versteht man aber das Schreiben von Befehlsabläufen, die einen Computer anweisen, bestimmte Aufgaben durchzuführen.

Wichtig zu erwähnen ist, dass der Computer bzw. genauer gesagt dessen Prozessor die Programmiersprache, in der die Befehlsabläufe geschrieben sind, gar nicht versteht. Sie dient dazu, die Befehlsabläufe, sogenannte Algorithmen, dem menschlichen Verständnis möglichst nahekommend zu formulieren. Die Befehle, die mit einer bestimmten Programmiersprache geschrieben werden, müssen erst in einen Maschinencode übersetzt werden, um vom Computer ausgeführt werden zu können.

Hinweis

Die folgende Erklärung zur Programmierung von Computern, die Sie hier finden, soll ein grundlegendes Verständnis für die Thematik schaffen. Der tatsächliche Vorgang ist sehr technisch und komplex und eine Erläuterung aller Details passt nicht in den Rahmen dieses Buchs. Sollte Sie das Thema interessieren, finden Sie dazu online umfangreiche Erläuterungen. Auch in der Fachliteratur gibt es ein umfangreiches Angebot an Werken zu diesem Thema, etwa Code von Charles Petzold, ebenfalls erhältlich beim mitp-Verlag.

1.2.1 Kompilierte Programmiersprachen

Diese Art von Programmiersprachen verwenden einen sogenannten *Compiler* (deutsch: Übersetzer). Dabei handelt es sich um ein Computerprogramm, das den Quellcode, der in einer bestimmten Programmiersprache geschrieben wurde, in vom Prozessor verständlichen Maschinencode übersetzt (kompiliert).

Der Maschinencode enthält Befehle, die vom jeweiligen Prozessor, der den Code ausführt, verstanden werden. Unterschiedliche Hardware kann dabei unterschiedliche Befehlssätze aufweisen. Beim Kompilieren wird der Code auch optimiert, das heißt, er wird gekürzt, anders angeordnet und durchläuft weitere Änderungen, die das Programm schneller machen und weniger Speicher verbrauchen lassen.

```
1 int main() {  
2     int a = 4;  
3     int b = 1;  
4     int c = a + b;  
5     return c;  
6 }
```

Listing 1.1: Simpler Programmcode, der in C, Java, C# und anderen Sprachen geschrieben worden sein könnte

```
1 55  
2 48 89 E5  
3 C7 45 FC 04  
4 C7 45 F8 01  
5 8B 45 F8  
6 8B 55 FC  
7 01 D0  
8 89 45 F4  
9 8B 45 F4  
10 5D  
11 C3
```

Listing 1.2: So könnte der Maschinencode für das Programm aus Listing 1.1 aussehen.

Ein simples Programm wie das aus Listing 1.1 ist selbst für Menschen ohne Programmiererfahrung halbwegs verständlich, die Variablen *a* und *b* werden addiert und das Ergebnis in die Variable *c* gespeichert, welche an die aufrufende Stelle zurückgegeben wird. Maschinencode (Listing 1.2) hingegen ist für einen Menschen nicht mehr lesbar. Natürlich wäre es möglich, Maschinencode zu lernen, das Unterfangen wäre aber eher mühsam. Dazu kommt noch die Tatsache, dass der Maschinencode je nach Plattform ganz unterschiedlich ausfallen kann.

Sie sehen also, es handelt sich um ein eher sinnloses Unterfangen. Dank des Compilers können Programmierer den Code mithilfe von Programmiersprachen in einem relativ einfach lesbaren Format schreiben und diesen dann für verschiedene Plattformen übersetzen lassen, ohne den Code für jede Plattform einzeln anpassen zu müssen. Bekannteste Vertreter kompilierter Programmiersprachen sind C bzw. C++.

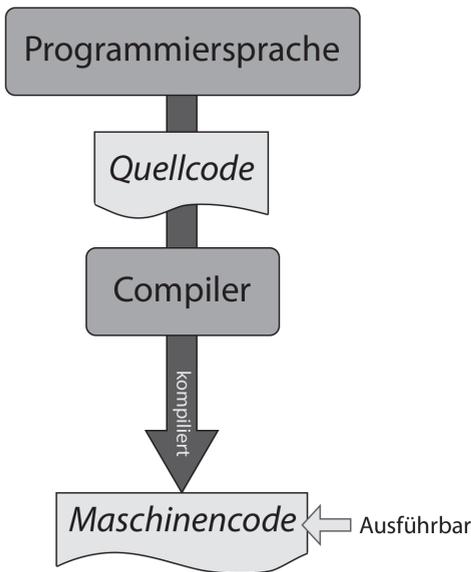


Abb. 1.1: Schematischer Ablauf bei kompilierten Programmiersprachen

Bis zur Entwicklung des ersten Compilers durch Grace Hopper Mitte der Fünfzigerjahre war das direkte Schreiben von Maschinencode allerdings die einzige Möglichkeit, Programme zu entwickeln. Damals gab es aber auch weniger verschiedene Plattformen und Funktionalitäten der damaligen Computer – und somit war die Anzahl an Befehlen geringer als heute.

1.2.2 Interpretierte Programmiersprachen

Technisch etwas anders sieht es bei den sogenannten *interpretierten Sprachen* aus. Diese werden nicht direkt in Maschinencode übersetzt, ehe das Programm ausgeführt werden kann. Der Programmcode bleibt grundsätzlich, wie er ist. Erst beim Ausführen des Programms wird er an eine spezielle Software – den sogenannten *Interpreter* – übergeben. Dieser interpretiert den geschriebenen Code in Echtzeit und kennt alle Anweisungen der Programmiersprache, die er dann als Maschinencode ausführt. Bekannte Vertreter von interpretierten Sprachen sind Python und JavaScript.

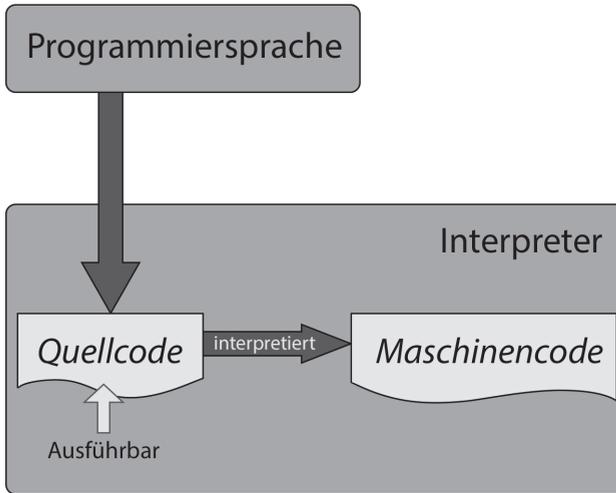


Abb. 1.2: Schematischer Ablauf bei interpretierten Programmiersprachen

1.2.3 Kompilieren vs. Interpretieren

Der Unterschied zwischen den beiden Vorgängen kann etwas verwirrend sein. Am besten hilft hier ein analoges Beispiel. Denken Sie an eine Gebrauchsanleitung oder eine Aufbauanleitung und nehmen Sie an, diese ist in einer Sprache geschrieben, die Sie nicht verstehen. Sie haben zwei Möglichkeiten, die Anweisungen der Anleitung umzusetzen:

- Option A: Es gibt bereits eine Übersetzung der Anleitung. Diese wurde im Vorfeld von einer Person (Compiler) angefertigt, die sowohl die Sprache der Originalanleitung als auch Ihre Sprache beherrscht.
- Option B: Sie kennen jemanden (Interpreter), der sowohl Ihre Sprache spricht als auch die Sprache der Anleitung versteht, und bitten denjenigen, die Anleitung direkt für Sie zu übersetzen.

Vereinfacht ausgedrückt, machen beide Methoden dieselbe Arbeit, aber zu einem anderen Zeitpunkt (vor dem Ausführen des Programms vs. während dem Ausführen) und an einem anderen Ort (Rechner des Entwicklers vs. Rechner des Anwenders).

Natürlich haben beide Methoden gewisse Vor- und Nachteile. Diese sind für Anfänger eventuell noch schwieriger zu verstehen als die unterschiedlichen Funktionsweisen der beiden Methoden. Aber gerade als Anfänger sollten Sie sich auch nicht allzu viele Gedanken darüber machen. Ihr Fokus sollte auf dem Lernen der Grundlagen liegen, dem Üben und dem Sammeln von Erfahrung durch das Schreiben von Code.

Dennoch möchte ich Ihnen im Folgenden eine vereinfachte Übersicht über Vor- und Nachteile der beiden Methoden bieten.

An dieser Stelle sollte noch erwähnt werden, dass, auch wenn von interpretierten bzw. kompilierten Programmiersprachen die Rede ist, diese Übersetzungsmethoden keine Eigenschaft der Sprache selbst sind, sondern davon abhängen, wie die Sprache implementiert, also technisch umgesetzt wird. Es gibt Programmiersprachen, die sowohl kompiliert als auch interpretiert werden können. Python ist hier ein gutes Beispiel. Primär als Skriptsprache eingesetzt, wird Python interpretiert. Durch die wachsende Beliebtheit und die breit gefächerten Einsatzmöglichkeiten gibt es für diverse Plattformen auch Python-Compiler.

Just-in-time-Compiler

Für immer mehr Sprachen stehen Just-in-time-Compiler zur Verfügung. Diese speziellen Compiler übersetzten Code bzw. die Änderungen am Code in Echtzeit. Sie vereinen den flexiblen Entwicklungszyklus von interpretierten Sprachen mit der schnellen Ausführung von kompilierten Sprachen.

Ausgeliefert wird immer noch kompilierter Code. Durch die Echtzeitübersetzung muss der Code aber nicht nach jeder Änderung neu kompiliert werden, um ihn zu testen. JIT-Compiler gehen so weit, dass das Programm auch nicht mehr beendet werden muss, um Änderungen am Code vorzunehmen. Das ist besonders bei sehr komplexen Programmen hilfreich, wo es mitunter einige Anwendungsschritte und Zeit benötigt, um die getätigte Änderung sichtbar zu machen. Auch bei Fehlern, die nur schwierig zu reproduzieren sind, helfen JIT-Compiler enorm.

Vorteile kompilierter Programmiersprachen

Kompilierte Programme sind in der Regel schneller, da das Übersetzen in Maschinencode schon vor dem Ausführen passiert. Der Anwender erhält das bereits übersetzte Programm und benötigt dafür auch keine zusätzlichen Ressourcen.

Nachteile kompilierter Programmiersprachen

Das Kompilieren erfordert einen zusätzlichen Zeit- und Ressourcenaufwand zwischen dem Schreiben und Ausführen des Programms. Dieser Aufwand kann bei großen Programmen stark anwachsen und verlangsamt den Entwicklungszyklus. Code muss immer wieder getestet werden, aber bevor das geschehen kann, muss der Code kompiliert werden. Dasselbe gilt für die Fehlersuche. Um den Code zu verändern, muss das Programm gestoppt und nach der Änderung wieder kompiliert werden.

Bei großen Softwareprojekten arbeiten mehrere Entwickler am Code. Dieser wird dann nur noch zu Testzwecken am jeweiligen Rechner des Entwicklers kompiliert. Das Kompilieren des finalen Programms, ehe dieses produktiv genutzt werden kann, erfolgt auf eigenen Servern. Je mehr Code kompiliert werden muss, umso mehr Ressourcen muss für diese Server bereitgestellt werden.

Ein weiterer Nachteil ist, dass ein Compiler immer plattformabhängig ist. Soll ein Programm auf verschiedenen Plattformen laufen, muss der Code für jede Plattform separat kompiliert werden.

Vorteile interpretierter Programmiersprachen

Interpretierte Programmiersprachen sind in der Regel etwas flexibler. Da der Interpreter das Übersetzen auf der Zielplattform übernimmt, ist der Code selbst immer plattformunabhängig und kann von jeder Plattform genutzt werden. Interpreter erlauben bzw. vereinfachen gewisse Programmierparadigmen und -funktionen (z.B.: Reflection und dynamic typing), deren Erklärung aber über den Rahmen dieses Buchs hinausgeht.

Nachteile interpretierter Programmiersprachen

Primärer Nachteil ist die langsamere Ausführungszeit von Programmen, die erst interpretiert werden müssen. Außerdem muss der Anwender einen Interpreter für die verwendete Sprache installiert haben.

Ein weiterer Nachteil ist, dass gewisse Fehler, die Kompilierfehler (compiler error), wie der Name schon verrät, nicht vorab gefunden werden. Normalerweise werden sie zum Zeitpunkt des Kompilierens erkannt und führen zu einem Abbruch des Kompilierens. Der Interpreter aber arbeitet eine Codezeile nach der anderen ab und vergisst, was in den Zeilen zuvor passiert ist. Fehler treten dann zum Zeitpunkt des Ausführens auf und lassen das Programm abstürzen oder führen zu ungewolltem Verhalten bzw. falschen Ergebnissen. Wird Code nicht gewissenhaft getestet, bevor er ausgeliefert wird, fallen Fehler erst beim Anwender auf und das Programm ist nicht ausführbar.

Moderne Programmierumgebungen (IDEs, siehe Abschnitt 5.2 »Die Entwicklungsumgebung«) bieten diverse Funktionen, um das Schreiben von Code zu erleichtern. Diese können für interpretierte Sprachen eingeschränkt oder gar nicht zur Verfügung stehen.

1.2.4 Bytecode und Laufzeitumgebungen

Da Computerprogramme lange nicht mehr nur mathematische Probleme lösen, sondern komplexe Aufgaben übernehmen sollen, wie Dateien zu erstellen und zu manipulieren, oder über das Netzwerk zu kommunizieren, reicht ein Compiler heute nicht mehr aus. Als Programmierer wollen Sie aber den Maschinencode für

diese Aufgaben (meist) nicht selbst schreiben. Höhere Programmiersprachen bieten eine Vielzahl von Funktionen und Schnittstellen zu allen möglichen Hardwarekomponenten eines Computers, wie etwa grafische Ausgabe, Tonwiedergabe, Arbeits- und Festplattenspeicher oder eben Netzwerkschnittstellen. Damit Ihr Programm diese Funktionen nutzen kann, muss der Compiler entweder all diese Funktionen zusammen mit Ihrem Quellcode in das Programm integrieren oder der erzeugte Code wird in einer sogenannten *Laufzeitumgebung* (Runtime Environment) ausgeführt.

Hierbei wird der Quellcode nicht direkt in Maschinencode kompiliert, sondern in sogenannten *Bytecode*. Der Bytecode liegt in binärer Form (Folgen von Nullen und Einsen) vor, er ist also vom Menschen nicht mehr lesbar. Er ist immer noch plattformunabhängig, hat aber bereits einige Optimierungen durchlaufen. Der Bytecode wird von der Laufzeitumgebung entweder zu Maschinencode kompiliert oder direkt interpretiert.

Je nach Betriebssystem sind Laufzeitumgebungen für einige Sprachen bereits integriert. Andere Laufzeitumgebungen müssen Sie selbst nachinstallieren. Das trifft im Übrigen auch dann zu, wenn Sie ein fertiges Programm einfach nur ausführen wollen und nicht selbst programmieren. Die bekanntesten Laufzeitumgebungen sind das Java Runtime Environment (JRE) oder das .Net-Framework für C# und weitere verwandte Sprachen.

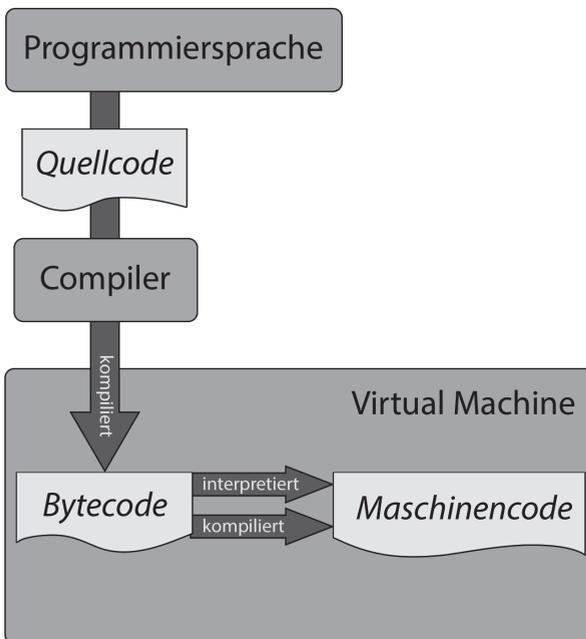


Abb. 1.3: Schematischer Ablauf bei Programmiersprachen mit Laufzeitumgebung

Stichwortverzeichnis

.NET 173
.Net-Framework *siehe* Laufzeitumgebung

A

Ableitung 114
Abstrakte Klasse 120, 124, 193
Abstrakte Methode 120
Abstraktion 212
Acceptancetest 166
Access Modifier 108
Ada (Programmiersprache) 14
Addition 41, 176
Aggregation 33
Aktion 98
Algorithmus 15, 72, 128, 253
Android 141, 209
Android SDK 261
Android Studio 141
API Documentation 135
Apple 143
Applet 218
App *siehe* Mobile Applikation
Arbeitsspeicher 27, 104, 249
Arduino 211
Arithmetischer Überlauf 31
Array 32, 38, 184, 192
 Deklaration 38
 Initialisierung 38
 sortieren 92
 Zugriff 38
Attribut 99, 103, 109, 193
 Datentyp 106
 gültige Werte 111
 Wert initialisieren 106
Aufzählung *siehe* Sammlung
Ausdruck 40, 44, 52
Ausgabe 174, 175
Ausnahme *siehe* Exception
Ausrufezeichen 47, 73
Auszeichnungssprache 217
Automatisierung 210

B

Babbage, Charles 14
Backend 203, 253
Bash *siehe* Konsole
BASIC 14
Basisklasse *siehe* Elternklasse
Bedingte Verzweigung 51

Bedingung 51, 52, 55, 61, 64
 mehrere verketteten 55
Bedingte Schleife 60
Befehlssatz 209
Benutzerdefinierter Datentyp 33, 102
Benutzereingabe
 prüfen 189
Benutzeroberfläche 159, 210, 213
Berechnungsfehler 156, 159
Bernoulli 14
Betriebssystem 14, 201, 209
Beziehung 97
Bibliothek 75, 108, 121, 125, 138, 150, 180, 195,
 214, 253
Binäre Daten 33, 254
Binärsystem 245, 246
 Addition 247
 Kennzeichnung 248
Bit 249
Bitweiser Operator 48
Block 52, 53, 72
Boolean 29, 43, 44, 52
Bool *siehe* Boolean
Branch 146, 148
Break 58, 60, 66, 191
Breakpoint 164
Browser 202, 203, 205, 217, 256
Bubblesort 92, 93
Bug 155
Byte 31, 249
Bytecode 20, 21, 254

C

C 14, 17, 208, 215
C# 15, 35, 139, 169, 205, 208, 216
C++ 14, 17, 139, 215
Camelcase 26
Case 58
Cast *siehe* Type Cast
Catch *siehe* Exception
Character 29
Char *siehe* Character
Checkout 147, 148
Client 202, 205, 254
Clone 148
Cloud 254
COBOL 14
Code 16, 21, 144, 158, 255
Codebibliothek *siehe* Bibliothek

- Code-Completion 138
 - Code-Style-Guides *siehe* Programmierrichtlinie
 - Coding Convention *siehe* Programmierrichtlinie
 - Coding-Standards *siehe* Programmierrichtlinie
 - Collection 33, 38, 39
 - Commit 146, 147
 - Compiler 14, 16, 17, 112, 138, 157, 169, 174, 255
 - JIT 19
 - Nachteile 19
 - Vorteile 19
 - Compile Time Error *siehe* Kompilierfehler
 - Computer 14, 201, 208
 - Conditional *siehe* Verzweigung
 - Continue 66
 - CSS 203, 217
- D**
- Database query *siehe* Datenbankabfrage
 - Dateiendung 254
 - Datei lesen 34
 - Datei schreiben 34
 - Datenbank 206
 - Datenbankabfrage 206
 - Datenbankserver 207
 - Datenbanksystem *siehe* Datenbank
 - Datenkapselung *siehe* Kapselung
 - Datenklasse 229
 - Datentyp 27, 99, 177
 - benutzerdefinierter 102
 - generischer 131
 - primitiver 31
 - Datentypüberprüfung 117
 - Datenübertragungsrate 252
 - Datum 34
 - Debugger 138
 - Debugging 155, 163
 - Zeit 164
 - Default 59
 - Default-Konstruktor 106
 - Default value *siehe* Standardwert
 - Deklaration 35, 102, 112
 - Dekrement 43
 - Dependency Injection 159
 - Designer 211, 212
 - Designfehler 157
 - Desktop 255
 - Desktop-Programm 201, 208
 - Dezimalsystem 245
 - Diamond-Problem 123
 - Dictionary 33
 - DirectX 213
 - Division 41, 42
 - Dokumentation 135, 150, 152, 172
 - API 151
 - inline 150
 - Double 30
 - Do-while 60
 - Dynamic typing *siehe* Dynamische Typisierung
 - Dynamische Sprache 220
 - Dynamische Typisierung 28
- E**
- Echtzeit-Rendering *siehe* Rendering
 - Echzeitkompilierung 138
 - Eclipse 141
 - Editor 212
 - Effekt 213
 - Eigenschaft 98
 - Einchecken *siehe* Commit
 - Eingabe 175
 - Eingebettetes System *siehe* Mikrocontroller
 - Einheitenpräfix 250
 - Einlesen 189
 - Einlesen *siehe* Eingabe
 - else 54
 - Elternklasse 114
 - Embedded System 258
 - Endlosschleife 61, 67, 189, 191
 - End-to-End-Test 165
 - Engine 212, 216
 - Englisch 26
 - Entwicklerwerkzeug 135, 137, 144, 151, 209, 212
 - Entwicklungsumgebung *siehe* IDE
 - Erweitern 119, 195
 - Erweiterung *siehe* Plugin
 - EVA 14
 - Exception 161, 163, 177, 186
- F**
- Faktoriell *siehe* Fakultät
 - Fakultät 73, 182
 - rekursiv 74
 - Fallthrough behaviour 60
 - Fallunterscheidung *siehe* Mehrfachverzweigung
 - False *siehe* Boolean
 - Fehler 20, 155, 160, 163
 - im Bedienkonzept 157
 - Fehlersuche *siehe* Debugging
 - Festplatte 249
 - Fetch 148
 - Float 30
 - Foo 110
 - Foobar 105
 - For 63
 - Foreach 65
 - FORTRAN 14
 - Framework 214, 219, 256
 - Frontend 203, 256
 - Frontend-Programmierung 219
 - Full-Stack 206, 256
 - Funktion 68, 100, 181

Name 68
 Parameter 68
 Rückgabewert 69
 Funktionstest 165

G

Gameplay-System *siehe* Spieleprogrammierung
 Ganzzahl 29
 Generic 131
 Generischer Datentyp *siehe* Generic
 Getter 110
 Git 146, 148
 Gleitkommazahl 30
 Google 141
 GUI 257
 Gültigkeitsbereich 71

H

Haltemarke *siehe* Breakpoint
 Handbuch 135
 Hardware 211, 212
 Heap 104
 Hexadezimalsystem 248
 Hierarchie 115, 121, 123, 125
 Hilfe 135, 136
 Hilfsklasse 113
 Hochsprache *siehe* Programmiersprache
 Hopper, Grace 14, 17, 155
 HTML 15, 203, 217

I

IDE 20, 137, 157, 163, 169, 220, 255
 If-else *siehe* Bedingte Verzweigung
 Implementieren 256
 Implementierung 24
 Import *siehe* using-Direktive
 Index 33, 38, 65, 161
 Initialisierung 36, 37, 112
 Inkrement 43
 Input-Parameter *siehe* Parameter
 Instanz 102
 Instanziierung 102, 112, 116, 121
 Integer 30
 Integrationstest 165
 IntelliJ 142
 Interface 124, 128, 158, 198, 209
 Access Modifier 126
 anwenden 128
 definieren 125
 verwenden 126
 Internet 15
 Internet of Things *siehe* IoT
 Interpreter 17, 219, 257
 Nachteile 20
 Vorteile 20
 Interpretierte Sprache *siehe* Interpreter
 Int *siehe* Integer
 iOS 143, 209

IoT 211, 257
 Iteration 60, 65, 72

J

Jacquard, Joseph-Marie 13
 Java 15, 32, 141, 142, 205, 208, 218
 Javadoc 151
 Java Runtime Environment *siehe* Laufzeitumgebung
 JavaScript 17, 204, 218
 JetBrains 144
 Just-in-time-Compiler 19

K

Kapselung 107
 KI 212
 Kindklasse 114
 Klasse 99, 114, 192
 abstrakte 121
 Hierarchie 115
 Klassendiagramm 154
 Klassenvariable *siehe* Attribut
 Kombinierte Zuweisung 43
 Kommandozeile 137, 210, 257
 Kommentar 12, 150, 180
 auskommentieren 151
 Kompilieren *siehe* Compiler
 Kompilierfehler 156
 Kompilierte Sprache *siehe* Compiler
 Komplexität 56
 Komponententest *siehe* Unit-Test
 Konflikt 145, 148
 Konsole 173, 210, 257
 Konstante 37
 Konstruktor 106, 118, 197
 parameterloser 106
 parametrisierter 106
 Konvention *siehe* Programmierrichtlinie
 Konvertierung 76
 Kotlin 261

L

Lambda Expression 65
 Language Documentation 135
 Laptop *siehe* Computer
 Laufzeit 257
 Laufzeitfehler 156, 159, 163
 Laufzeitumgebung 21, 27, 28, 137, 171, 202,
 209, 258
 Leibnitz, Gottfried Wilhelm 245
 Lesbarkeit 149
 Linux 14, 139, 201, 216
 Lisp 14
 Liste 33, 38, 40, 190
 Loadtest 166
 Logging 163, 164
 Logikgatter 48
 Logischer Fehler 156, 159
 Logischer Operator 44

Logisches Nicht 47
 Logisches Oder 45
 Logisches Und 44
 Loglevel 163
 Long 30
 Lösung 223
 Lovelace, Ada 14

M

macOS 143, 201, 216
 Main-Funktion 70, 72, 101, 173
 Makro 217
 Markdown 152
 Maschinencode 15, 16, 17, 21, 258
 Maschinsprache 23
 Maschine *siehe* Rechenmaschine
 Mehrfachvererbung 123
 Mehrfachverzweigung 56
 Mengenschleife 65
 Merge 146, 148
 Metadaten 217
 Methode 100, 101, 103, 107, 109
 abstrakte 121
 erweitern 120
 schützen 120
 überschreiben 119
 verdecken 120
 Zugriff 114
 Methodensignatur 100
 Microsoft 139
 Microsoft Office 217
 Mikrochip 211
 Mikrocontroller 211, 215, 218, 258
 Mobile Applikation 141, 208, 216, 253
 Mobilgerät 208
 Mock 158
 Modulo-Operator 42, 83
 Modultest *siehe* Unit-Test
 Monkey-Test 166
 Mooresches Gesetz 252
 MSB 250
 Multiplikation 41

N

NetBeans 143
 New 102
 Null 105, 162
 Null-Reference-Fehler 157

O

Objective C 208
 Objekt 98, 99, 102, 106
 Speicherbedarf 105
 vergleichen 128, 130
 Objektorientierte Programmierung 14, 97,
 107, 192
 Off-by-One 67
 Oktalsystem 248

OOP *siehe* Objektorientierte Programmie-
 rung
 OpenGL 213
 Open-Source 139, 148, 150, 258
 Operator 40, 176, 180
 arithmetisch 41
 logisch *siehe* Logischer Operator
 mathematisch 40
 Modulo 42
 Rang 49
 Vergleich 42
 Vorzeichen 40
 Oracle 143
 Overflow 32
 Override *siehe* überschreiben

P

Paradigma 210
 Parameter 68
 Parameterliste 68, 100, 131
 PC *siehe* Computer
 PHP 15, 204, 219
 Physik 212
 Plattform 205, 208, 216, 259
 Plattformabhängig 259
 Plattformunabhängig 169, 218
 Plugin 138
 Polymorphie 117, 122
 Postfix 43
 Potenz 180
 Präfix 43
 Primzahl 82
 private 108
 Programm 14, 16, 17, 62, 201, 211
 Client 202
 Programmfluss 51
 Programmieren 13, 15, 23
 Programmierer 17, 203, 206, 211
 Programmierrichtlinie 26, 52, 149
 Programmiersprache 14, 15, 23, 215, 220
 Programmierumgebung *siehe* IDE
 Projekt 138
 Property 109, 110
 Protected 108
 Prozentwert 42
 Prozessor 15, 16, 249, 260
 Pseudocode 34, 52, 178
 Public 108
 Pull 148
 Push 148
 PyCharm 140
 Python 17, 19, 140, 169, 219

Q

Quellcode 255
 Queue 33

R

Realtime-Rendering *siehe* Rendering

- Rechenmaschine 13, 14, 23
- Rechenzeichen 41
- Rechnen 41
- Redundanz 68, 114, 118
- Refactoring 236
- Reference 135
- Regex 189
- Reihen *siehe* Array
- Rekursion 72
- Remote 148
- Rendering 213, 260
- Repository 146
- Rest 42
- Return value *siehe* Rückgabewert
- Rider 144
- Rückgabewert 69, 71, 100, 131
- Runtime Environment *siehe* Laufzeitumgebung
- Runtime Error *siehe* Laufzeitfehler
- S**
- Sammlung 32, 33
- Schleife 60, 72
 - abbrechen 66
 - bedingte 60
 - Fehler 67
 - fortsetzen 66
- Schleifenrumpf 60
- Schleifenzähler 63, 65, 67, 72
- Schlüsselwort 24, 100, 102
- Schnittstelle *siehe* Interface
- Scope *siehe* Gültigkeitsbereich
- SDK 209, 212, 214, 261
- Self 106
- Semantik 24
- Semantikfehler 155
- Semikolon 35
- Sequenzdiagramm 154
- Server 15, 202, 205, 210, 260
- Setter 110
- Shader 213
- Shell *siehe* Konsole
- Short 29
- Signed 31, 250
- Skript 204, 209
- Skriptsprache 204, 209, 210, 216, 218, 219
- Smart Home 193
- Smartphone 208
- Smoke-Test 166
- Software-Development-Kit *siehe* SDK
- Software *siehe* Programm
- Softwaretest 158, 165
 - automatischer 166
 - manueller 166
 - Nachteile automatischer Tests 167
 - Vorteile automatischer Tests 167
- Sortieralgorithmus 128
- Sortieren 92, 128, 130, 192
- Sortierte Menge 88
- Sourcecode 255
- Speicher 25, 31, 74, 162, 216, 218, 251
- Speicheradresse 25, 104
- Speicherbedarf 29
- Speichermanagement 27
- Spiele-Engine 212, 213
- Spieleentwicklung 216, 218
- Spieleprogrammierung 212, 216
- Sprachdokumentation 151
- Stack 104
- Stack Exchange 136
- Stack Overflow 136
- Stacktrace 187
- Standardbibliothek *siehe* Bibliothek
- Standardwert 106
- Static 111, 181
- Static typing *siehe* Statische Typisierung
- Statische Methode 111
- Statisches Attribut 112
- Statische Typisierung 28
- String 32, 176, 184
- String-Operation 32
- Stroustrup, Bjarne 14
- Subklasse *siehe* Kindklasse
- Subtraktion 41
- Subversion *siehe* SVN
- Suche 84, 183
 - binär 88
 - linear 84
- Sun Microsystems 143
- Superklasse *siehe* Elternklasse
- SVN 146
- switch 56
 - abbrechen 58
 - Syntax 57
- Syntax 24
- Syntaxfehler 24, 155
- Syntax-Highlighting 138
- Systemprogrammierung 215
- T**
- Tag 151, 203
- Tag (VCS) 146, 148
- Team 144
- Tebi 251
- Tera 251
- Test Driven Development (TDD) 159
- Test *siehe* Softwaretest
- Texteditor 137
- This 106
- Throw *siehe* Exception
- Torvalds, Linus 148
- Treiber 215
- True *siehe* Boolean
- Trunk 146
- Try *siehe* Exception
- Tuple 229
- Tutorial 136
- Type Cast 75, 162, 177

explizit 76
 implizit 75
 Typkonvertierung 76
 Typ *siehe* Datentyp
 Typumwandlung *siehe* Type Cast

U

Überschreiben 119, 195
 UML 153, 236, 261
 Umlaut 26
 Unicode 29
 Unit-Test 158
 Unity 212
 Unix *siehe* Linux
 Unreal Engine 212
 Unsigned 31, 250
 Using-Direktive 195

V

Variable 25, 28, 34, 71, 99, 102
 deklarieren 35
 Gültigkeitsbereich 63
 initialisieren 36
 lokale Variable 72
 Parametervariable 72
 Variablenname 26
 VCS 145
 Verdecken 120
 Vererbung 114, 120, 193
 Konstruktor 118
 Mehrfachvererbung 123
 private 116
 Syntax 116
 Vergleich *siehe* Vergleichsoperator
 Vergleichsoperator 59, 128
 Verschachtelung 56
 Version Control System *siehe* VCS
 Versionskontrolle 144, 145, 152
 Verzweigung 51, 72
 Virtual 119
 Visual Basic 217
 Visual Studio 137, 139, 170
 Projekt anlegen 172
 Visual Studio Code 139
 VLA 186
 Void 69
 von Neumann, John 14
 Von-Neumann-Architektur 14
 Vorschrift *siehe* Programm
 Vorzeichen 31, 40, 249, 250

W

Webanwendung *siehe* Webentwicklung
 Webdesigner 217
 Webentwicklung 15, 202, 203, 205, 217, 218, 219
 WebGL 218
 Webprogrammierung *siehe* Webentwicklung
 Webseite 203, 217, 219
 Webserver 202, 218, 219, 260
 Webservice 205
 Webstuhl, programmierbarer 13
 Werkzeug *siehe* Entwicklerwerkzeug
 Wertepaar 33
 While 60
 Wiederholung *siehe* Iteration
 Windows 139, 201, 216
 Workaround 120
 Wörterbuch 33
 WWW 217

X

Xcode 137, 143
 XML 151, 152, 153

Z

Zählen 43
 Zahlensystem 245
 Kennzeichnung 248
 Zählschleife 63
 Zeichen 25, 29, 32
 Zeichenkette 25, 28, 32
 Zeiger 27
 Zeit 34
 Zugriffsart 108
 Zugriffskontrolle 108
 Zusammenarbeit *siehe* Team
 Zuse, Konrad 14
 Zuweisung 36, 38, 43
 kombinierte 43
 Zuweisungsoperator 36
 Zweierkomplement 250