

Bitte alle einsteigen

Programmiersprachen für Ein-, Um- und Wiedereinsteiger



Die richtige Sprache	Seite 18
C und C++	Seite 22
C#	Seite 23
PHP	Seite 24
JavaScript und TypeScript	Seite 25
Java	Seite 26
Kotlin	Seite 27

Scala	Seite 28
Python	Seite 29
Swift	Seite 30
Go	Seite 31
R	Seite 32
Haskell	Seite 33

Programmierersprachen gibt es viele – und es gibt noch mehr Überzeugungen, welche die beste sei. Ein- und Umsteigern fällt die Entscheidung für eine Sprache daher nicht leicht. Unser Überblick zeigt, worauf es bei der Auswahl ankommt und welche Sprachen einen Blick wert sein könnten.

Von Jan Mahn

Programmieren können ist großartig! Mit den eigenen Händen auf der Tastatur erschafft man nützliche Werkzeuge, beeindruckende Spielwelten oder lässt den Computer komplizierte Probleme lösen. Doch so schön es ist, Programmieren zu können, so mühsam ist es für Neulinge, es zu lernen.

Programmierersprachen sind leider immer ein Kompromiss – eine Übersetzungsschicht zwischen einer Sprache, die wir Menschen schreiben und verstehen können, und einer Sprache, mit der ein Prozessor klarkommt. Jede Programmiersprache geht ein wenig anders an diese Übersetzungsaufgabe heran.

Wer bereit ist, als absoluter Neuling mit dem Programmieren zu beginnen, wird auf der Suche nach einer geeigneten Programmiersprache von starken Meinungen erschlagen: „Fang unbedingt mit einer objektorientierten Sprache an“, raten die einen. „Nimm unbedingt eine stark typisierte Sprache“, sagen viele. Wieder andere haben konkrete Empfehlungen, welches die einzig wahre Sprache sei, die man lernen sollte – und welche auf keinen Fall.

Die Debatten werden in Büros und in Foren oft mit einem fast schon religiösen Eifer geführt, viele wollen ihre Sprache gern gegen jegliche Kritik verteidigen. Für Einsteiger sind diese Debatten aber überhaupt nicht hilfreich: Sie müssen befürchten, mit der Entscheidung für eine Programmiersprache eine fatale Entscheidung gegen eine andere, vielleicht viel bessere Sprache zu treffen. Ihnen sei gesagt, dass die erste Sprache ziemlich sicher nicht die letzte sein wird, die sie lernen. Wer sich nach den ersten Erfahrungen langfristig fürs Programmieren begeistern kann, wird irgendwann auch eine weitere Sprache aus-

probieren und diese vergleichsweise zügig erlernen können. Nehmen Sie sich in der ersten Sprache die Zeit, ein paar Grundkonzepte wie Datentypen, Variablen und Klassen systematisch zu verinnerlichen – dann macht der Umstieg später mehr Spaß. Vieles wird Ihnen in anderen Sprachen erstaunlich vertraut vorkommen.

Die folgenden Artikel sind entstanden, um Ihnen einen Wegweiser durch die Landschaft der Programmiersprachen zu liefern. Statt ideologische Diskussionen zu führen, charakterisieren wir knapp und so objektiv wie möglich viele unterschiedliche Programmiersprachen – inklusive einem kleinen Einblick in die Syntax der Sprache und einer Einordnung, für wen wir die Sprache empfehlen würden. Die Autoren haben alle bereits viel Zeit mit der Sprache verbracht, über die sie schreiben. Sie haben über sie geflucht, sind an ihr verzweifelt und haben sie irgendwann zu schätzen gelernt. Aus eigener Erfahrung kennen sie viele der Stärken und Schwächen und haben grammatikalische Veränderungen oft selbst miterlebt.

Umsteigen lohnt sich

Diese Artikel richten sich aber ausdrücklich nicht nur an Einsteiger, die noch nie mit dem Programmieren in Berührung gekommen sind. Die Steckbriefe sollen auch erfahrene Entwickler und solche, die vor Jahren oder Jahrzehnten zuletzt vor einem Code-Editor saßen, dazu inspirieren, ihre bisherige Meinung über ihnen

fremde Sprachen zu überdenken. Vielleicht entdecken Sie ja eine, die eines Ihrer Probleme ganz anders und viel effizienter lösen kann als Ihre Programmier-Muttersprache. Auch ein Blick auf Artikel zu Sprachen, um die Sie bisher einen großen Bogen gemacht haben, kann sich lohnen – Programmiersprachen sind im steten Wandel und Ihre Überzeugungen von vor 15 Jahren sind meist nicht mehr aktuell.

Index-Verwirrungen

Auf der Suche nach der „richtigen“ Sprache werfen viele einen Blick auf Rankings und Indizes. Bekanntester und häufig zitierter Vertreter ist der TIOBE-Index, der stumpf die Anzahl der Treffer für die Suchanfrage „<Name der Sprache> language“ in gängigen und weniger gängigen Suchmaschinen berücksichtigt. Dieser Index kann aber keine Auskunft darüber geben, wie beliebt, effizient oder nützlich eine Sprache ist – er sagt vor allem, wie viel schon darüber im Internet geschrieben wurde. Da ist es logisch, dass auf den Spitzenplätzen meist C und Java landen. Beide sind schon lange im Geschäft und immer noch aktiv im Einsatz. Auch der PYPL-Index (Popularity of Programming Language), der dem Namen nach die Popularität messen soll, misst in

Wirklichkeit etwas anderes: Auch er nutzt Daten von Suchmaschinen, aber nicht die Anzahl der Treffer, sondern die Anzahl von Suchanfragen nach Tutorials, und basiert auf Google Trends. Je mehr Nutzer nach „<Name der Sprache> tutorial“ bei Google suchen, desto „beliebter“ wird die Sprache nach der Logik von PYPL. So landet dort aktuell Python

auf Platz 1, C++ nur auf Platz 6. Auch gegen dieses Ranking spricht eine Menge. Ist eine Sprache wirklich gut oder beliebt, weil viele Hilfesuchende bei Google nach Rat suchen, oder ist das vielleicht sogar ein Nachteil?

All diese Rankings mögen ein nettes Gesprächsthema für die Kaffeeküche von Entwicklerbüros sein (als Alternative zu den Bundesliga-Ergebnissen vom Wochenende), niemand sollte sich jedoch davon leiten lassen, wenn es um die Wahl einer Programmiersprache geht. Was man in diesen Rankings höchstens ablesen kann, sind auffällige Veränderungen über

Bei der Wahl einer Programmiersprache sollte man sich nicht nur an einem Index orientieren.

die Zeit – wenn eine Sprache plötzlich verstärkt in Suchmaschinen auftaucht, kann das darauf hindeuten, dass die Nutzerschaft gewachsen ist. Aber auch TIOBE musste schon 2004 zugeben, dass es da Verzerrungseffekte geben kann und ein plötzlicher Abstieg von Java schlicht durch eine Änderung im Google-Algorithmus zu klären war.

Beim Einsatz der Indizes gilt daher: Programmieren ist weder sportlicher Wettkampf noch Religion, sondern ein Handwerk. Die Programmiersprache ist das Werkzeug des Entwicklers zum Lösen von Problemen. Und ein Werkzeug muss nicht das meist-gegoogelte sein, sondern vor allem passen: zum Problem und auch zum Entwickler.

Hauptsache angemessen

Bei der Entscheidungsfindung hilft es, sich darüber klar zu werden, welche Art von Projekten man mit der Sprache angehen will. Meist haben Sie ein konkretes Ziel im Kopf, wenn Sie beginnen. Das ist auch gut so – Einsteiger mit der vagen Idee, Programmieren zu lernen, nur um Programmieren zu können (zum Beispiel, weil das auf dem Arbeitsmarkt angeblich wichtig ist), steigen meist schnell frustriert aus. Am besten gehen Sie mit einem konkreten Ziel an das Thema heran. Dann hält die Motivation, am Ball zu bleiben, deutlich länger. Das Ziel sollte so gewählt sein, dass Sie es mit dem aktuellen Wissen noch nicht erreichen können, nur so können Sie ja Neues lernen. Gleichzeitig darf es nicht utopisch sein – die Idee, einen Ersatz für den Linux-Kernel zu schreiben, wird scheitern. Ein Terminplaner für die Familie kann ein gutes Projekt sein, vielleicht eine App, mit der man den Bestand des Vorratschranks zu Hause überwachen kann, oder eine datenbankgestützte Verwaltung der Kaffeekasse im Büro. Je mehr potenzielle Nutzer Ihr erstes Projekt hat, desto besser ist das für die Motivation.

Wenn Sie Ihr erstes Projekt vor Augen haben, können Sie die Wahl der ersten Programmiersprache schon eingrenzen. Soll die Anwendung im Browser laufen? Planen Sie eine grafische Desktop- oder eine Kommandozeilen-Anwendung? Für

Windows, Linux, macOS, iOS oder Android? Keine Programmiersprache ist für jede dieser Ideen gleichzeitig die perfekte Wahl und vieles fällt schnell durch das Raster.

Gerade Informatiker mit akademischer Ausbildung neigen dazu, eine Sprache vor allem nach ihren sprachlichen Konstrukten zu bewerten. Wie gut ist das Ver-

erbungskonzept, wie sauber wird mit Datentypen und Ausnahmen umgegangen und welche Kunststücke beherrscht der Compiler? Für Einsteiger sind solche Fragen dagegen eher unerheblich. Für sie steht eine verständliche Syntax im Vordergrund – in der Lernphase tippt man lie-

ber drei Zeilen mehr Code als eine raffinierte Abkürzung, die man nicht mehr intuitiv lesen kann.

Eine Programmiersprache ist ohnehin mehr als die Summe ihrer sprachlichen Raffinessen und bemerkenswerten Compilertricks. Zu einer Sprache gehört auch eine Community, die Bibliotheken für Standardprobleme entwickelt und bei Fragen in Foren oder auf Stackoverflow.com weiterhilft. Auch die Community muss zu Ihrem Problem passen. In der PHP-Community findet man beispielsweise Hilfe zu fast allen erdenklichen Problemen rund um Webentwicklung, während zur Python-Community viele Spezialisten für KI oder Datenanalyse gehören.

Typfrage

Eine Programmiersprache und ihre Community müssen nicht nur zum Projekt, sondern auch zum Programmierer (oder Einsteiger) kompatibel sein. Nicht jeder geht zum Beispiel mit einem mathematischen Hintergrund an das Lernen einer Programmiersprache heran und das ist auch nicht nötig.

Wer Programmieren lernen will, um eine Notizanwendung für Android zu schreiben oder mit einem geisteswissenschaftlichen Hintergrund Textanalyse betreiben will, bricht eine Einführung in eine Programmiersprache vielleicht schnell ab, wenn in einem der ersten Beispiele Matrizen multipliziert werden – dem Autor der Einführung kam das dagegen wie ein einfaches und plausibles Beispiel vor. Schließ-

lich ist das ja Grundstoff der naturwissenschaftlichen Disziplinen. Manchmal hilft es, einfach die Einstiegslektüre zu wechseln, bevor man eine Sprache gleich aufgibt.

Richtige und falsche, gute und schlechte Programmiersprachen gibt es also nicht. Daher haben wir uns bei der Auswahl nicht nur an Indizes orientiert, sondern Sprachen ausgewählt, die wir für relevant, lernwürdig und nützlich halten – für bestimmte Aufgaben und bestimmte Entwicklertypen. Das Spektrum der Sprachen ist dabei recht groß.

Viele der Sprachen haben eine große Nutzercommunity (C, Java, Python, PHP, JavaScript) oder sind beliebte Lehrsprachen in den Universitäten (Java und Python). Andere sind trotz kleiner Nutzercommunity in der Auswahl gelandet, weil sie ein Teilproblem der Software-Entwicklung auf eine interessante Weise lösen – Haskell (Seite 33), Go (Seite 31) und R (Seite 32).

Auch mit dabei sind Sprachen, die neue Wege gehen oder gerade dabei sind, vermeintlich alternativlose Sprachen zu verdrängen. So haben es etwa Kotlin (Seite 27), das im Android-Umfeld die Omnipräsenz von Java beendet hat, oder Scala (Seite 28), das als Java-Ersatz für Unternehmensanwendungen antritt, in diese Auswahl geschafft.

Glaubensfrage

Die kurzen Charakterisierungen liefern neben Orientierung auch Anregungen, sich auf eine neue Welt einzulassen. Keine der vorgestellten Sprachen kostet für den Einstieg Geld – die Hürde zum Ausprobieren ist oft niedrig. Meist reicht schon ein einfacher Editor, um die ersten Zeilen Code zu schreiben.

Bei allem Bemühen, einen objektiven Überblick über lernenswerte Programmiersprachen zu liefern, wird nicht jeder mit unserer Darstellung zufrieden sein – jede Auswahl ist unvollständig und möglicherweise vermissen Sie eine Sprache, die Ihnen viele gute Dienste erwiesen hat. Vielleicht kommt eine Sprache aus Ihrer Sicht auch zu gut oder zu schlecht weg. Nutzen Sie gern das Forum zu diesem Artikel (siehe ct.de/yzfd) für eine sachliche Diskussion. Aber vergessen Sie dabei nicht: Es geht um Programmiersprachen, nicht um Religion oder Sport. Nutzen Sie die Zeit lieber, eine neue Sprache zu lernen, als über sie im Forum zu schimpfen.

(jam@ct.de) 



Alt, aber nicht veraltet

C und C++: Läuft und läuft und läuft

Die Programmiersprachen C und C++ wirken auf den ersten Blick wie gut abgehangene Altertümchen. Die Wahrheit ist aber, dass beide wichtige Fundamente moderner Software sind.

Von Merlin Schumacher

Als Dennis Ritchie und Brian Kerningham vor fast 50 Jahren mit der Entwicklung von C begannen, wollten sie nur eine Programmiersprache haben, mit der sie Unix weiterentwickeln können. Dass dieser Antrieb eine der langlebigsten und flexibelsten Familien von Programmiersprachen begründen würde, war nicht abzusehen. In den 1980ern erweiterte der Däne Bjarne Stroustrup dann C um Klassen und Objekte und nannte das objekt-orientierte Ergebnis C++.

Beide Sprachen sind für die moderne Welt noch immer essenziell. So basiert der Linux-Kernel zum größten Teil auf C-Code, und der treibt wohl mehr Geräte an als jeder andere Kernel. Moderne High-End-Computerspiele, aber auch Browser-Engines, werden mit C++ geschrieben. All das sind Beispiele für Software, bei der es auf Hardwarenähe, hohe Performance und Flexibilität ankommt. Wer das letzte bisschen Leistung aus seinem Rechner quetschen will und trotzdem eine Hochsprache verwenden möchte, landet oft bei der C-Familie. Deren Einfluss ist groß: Die meisten Sprachen haben ihre Syntax und Konzepte auf die eine oder andere Weise bei C(++) abgegriffen.

Die Familie ist im Lauf der Jahrzehnte gewachsen. So gibt es als Verwandte noch Objective-C und C#. Allen gemeinsam ist, dass man den Code kompilieren muss. Die Variablen von C und C++ sind stark typisiert, man muss also explizit angeben, welche Art von Daten man speichern möchte. Macht man dabei Fehler, beschwert sich der Compiler sofort.

Ein großer Vorteil von C(++) ist seine große Portabilität. Ein C-Programm läuft ohne großes Zutun auf einem Microcontroller genauso wie auf einem Supercomputer. Ein schönes Beispiel dafür ist das Spiel Doom. Als der C-Quellcode veröffentlicht wurde, haben Fans Doom auf praktisch alles portiert, was zwischen 1 und 0 unterscheiden kann. Weil der Code so einfach gehalten ist, erfordert das oft nur wenige Anpassungen, und einen C-Compiler gibt es eigentlich für jede Prozessorarchitektur.

C++ und C enthalten eigentlich nicht viele Funktionen und Strukturen. Dank unzähliger Bibliotheken lässt sich damit alles umsetzen. Die Einarbeitung ist aber oft hart. Wer seine ersten Schritte mit dem Programmieren macht, sollte sie nicht mit C oder C++ tun. Zu viel Abstraktion und Verständnis von Computer-Architektur sind nötig, und man verdirbt sich nur den Spaß. Wer schon eine Programmiersprache kann und eine neue, spannende Herausforderung sucht, ist genau richtig.

Für den Anfang schnappt man sich vielleicht einen Arduino oder etwas Kompatibles. Die Umgebung bringt alles mit, um die Syntax zu lernen und die Fehler zu machen, die jeder C(++)-Programmierer am Anfang macht. Bei den Arduinos kommt dazu, dass man eine große Community, viele Informationsquellen und Beispiele sowie zahlreiche einfach zu nutzende Bibliotheken hat. Alle zusammen helfen einem dann zu verstehen, was Pointer eigentlich sind und warum die einen manchmal in den Wahnsinn treiben. Später kann man immer noch für den eigenen PC programmieren. Grundsätzlich ist es gut, eine Portion Schmerzresistenz und starken Willen zum Weiterlernen mitzubringen. Viele Dinge, die in Python oder JavaScript ein Klacks sind, brauchen hier liebevolle Handarbeit und intensive Dokumentationslektüre. So muss man sich bei C selber um die Verwaltung von Speicher und Daten kümmern. Oft bedeu-

tet das Fehler zu jagen, die schwer nachvollziehbar sind.

Das Ganze hat dann aber auch ein erfreuliches Ergebnis: Hat man C(++) gelernt, ist man befähigt, für fast alles schnellen und effizienten Code zu programmieren und versteht besser, wie der eigene PC wirklich arbeitet.

Für wen?

C und C++ sind Sprachen für jeden Zweck und für Menschen, die schon ein Grundwissen im Bereich Programmierung haben. Umsetzen kann man damit alles. Wirklich alles. Wer sich sicher fühlt in C oder C++, hat eine mächtige, aber auch gefährliche Sprache gelernt, denn gerade weil man so viel von Hand machen muss, inklusive des Speichermanagements, kann man sich diese Sicherheitslücken einfangen.

(mls@ct.de) 

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string str = "Ich bin ein String";
    // Das hier ist ein Beispiel in
    C++, zu erkennen an cout für die
    Textausgabe.
    cout << str << endl;

    int i = 5;
    float f = 2.5f;

    // Moderne C++-Compiler machen auch
    implizite Typumwandlungen mit.
    float sum = f + i;

    if (sum > 7) {
        cout << sum << endl;
    }
}
```

Die Syntax von C++ ist einfach, dennoch ist die Sprache sehr mächtig.



Ein Halbton höher

C#: Microsofts Universalsprache

Wurde C# anfangs gern als Microsofts Antwort auf Java bezeichnet, hat die Programmiersprache ihr Vorbild in einigen Aspekten mittlerweile überholt. Auch Open Source und Plattformunabhängigkeit sind keine Fremdwörter mehr.

Von Hajo Schulz

Der Name C# spricht sich wie „ßie scharp“, die englische Bezeichnung für die Musikknote Cis, einen Halbton über dem C. An die Programmiersprache C/C++ angelehnt ist auch die Syntax mit geschweiften Klammern zur Kennzeichnung von Blöcken und Semikolons an den Zeilenenden.

Version 1.0 von C# wurde von Microsoft entwickelt und 2002 veröffentlicht. Mittlerweile ist die Versionsnummer bei 8.0 angelangt. Microsoft versteht sich nach wie vor als Hüter der Sprache, hat aber den aktuellen Compiler (Codename Roslyn) unter der quelloffenen MIT-Lizenz veröffentlicht.

Die Geschichte von C# ist untrennbar mit der von Microsoft .NET verwoben: Das .NET Framework (aktuelle Versionsnummer: 4.8) stellt die grundlegenden Klassen für C#-Programme zur Verfügung. Außerdem enthält es die „Common Language Runtime“ (CLR): C#-Compiler erzeugen keinen Maschinencode, sondern einen „Common Intermediate Language“ (CIL) genannten, CPU-unabhängigen Zwischencode, den ein in der CLR enthaltener Just-in-time-Compiler (JIT) zur Laufzeit in Maschinencode übersetzt. Auch die .NET Plattform gibt es mittlerweile in einer Open-Source-Version, die .NET Core heißt und außer für Windows auch für Linux und macOS zur Verfügung steht.

C# ist statisch typisiert, Variablen gehören also stets zu einem bestimmten Typ.

Das gilt auch für Variablen, die mit dem Schlüsselwort `var` deklariert werden: Bei einer Definition wie

```
var temp = 23;
```

deduziert der Compiler anhand der Zuweisung, dass die Variable vom Typ `int` ist, und weist in der Folge jeden Versuch als Fehler zurück, ihr etwa einen `String` oder einen booleschen Wert zuzuweisen.

C# hat sich im Laufe seiner Versionsgeschichte von einer einfachen, objektorientierten Sprache in verschiedene Richtungen weiterentwickelt. So gibt es mit Lambda-Ausdrücken und Funktionen höherer Ordnung Anleihen aus der funktionalen Programmierung. Generische Klassen – vergleichbar mit Template-Klassen aus C++ – gehören ebenso zum Sprachumfang wie Iteratoren oder die Möglichkeit, bestehende Klassen über Extensions zu erweitern. Asynchrone Methoden und das Schlüsselwort `await` vereinfachen die nebenläufige Programmierung.

```
using System;
using static System.Console; // macht statische Member global verfügbar

// Klassen in Namensräume einzusortieren ist keine Pflicht, aber guter Stil
namespace Beispiel {
    public class MyApp {
        // Die Funktion Main() bildet den Einstiegspunkt
        public static void Main() {
            var str = "Ich bin ein ";
            WriteLine(str + "String."); // Eigentlich Console.WriteLine(...)

            int n = 5;
            double d = 2.5;
            var sum = n + d; // Automatische Konvertierung; sum ist double
            if (sum > 7) {
                WriteLine($"Die Summe ist {sum}."); // $"..." markiert Format-String
            }
        }
    }
}
```

Als objektorientierte Programmiersprache erlaubt C# Code nur innerhalb von Klassen.

Für wen?

Auch wenn es andere Programmiersprachen für .NET gibt – von Microsoft zum Beispiel Visual Basic .NET und F# –, ist C# die mit Abstand populärste. Die Sprache ist dank ihrer klaren Strukturen bestens für Programmierneinsteiger geeignet, auch weil der Compiler und die Laufzeitumgebung Fehler wie Zugriffsverletzungen praktisch unmöglich machen. Mit C# lässt sich so ziemlich jede Art von Programm erstellen, von einfachen Konsolenprogrammen über Anwendungen für den Desktop oder UWP-(Touch-)Apps bis hin zu Web- und Serveranwendungen mit Datenbankbindung. Bei der Beherrschung der teils recht komplexen Frameworks, die dabei zum Einsatz kommen, hilft Microsofts Entwicklungsumgebung Visual Studio. Deren Community Edition ist trotz üppiger Ausstattung für den privaten Gebrauch und für kleine Firmen kostenlos. (hos@ct.de) 

Literatur und Werkzeuge: ct.de/yqav

Elefant im Webserver

PHP: Für mehr als Besucherzähler und Gästebücher



PHP ist nicht nur eine leicht zu erlernende Sprache für Einsteiger. Mit modernen Frameworks und objektorientierter Programmierung entstehen in PHP gute Webanwendungen – allen Kritikern zum Trotz.

Von Jan Mahn

Begonnen hat PHP seine Karriere als Werkzeugkasten für dynamische Inhalte auf Webseiten schon 1995. Anfänglich ging es um kleine Schnipsel, eingebettet in eine HTML-Seite, beispielsweise für einen Besucherzähler. Mit der Zeit wurde PHP aber zur ausgewachsenen Programmiersprache. Während viele große Anwendungen wie WordPress oder der Webshop Magento in PHP entstanden, schleppten die PHP-Macher einige konzeptionelle Fehler aus der Zeit als kleine Skriptsprache über viele Versionen durch.

PHP ist eine Skriptsprache, sie braucht also einen Interpreter, der den Code zur Laufzeit übersetzt und ausführt. Der PHP-Interpreter läuft meist auf einem Webserver und der Code kann in andere Dokumente wie HTML-Seiten eingebettet werden. Außerhalb von Web-

entwicklung spielt PHP dagegen kaum eine Rolle.

Viele Elemente der Syntax hat PHP von C geerbt. Die Sprache ist aber, anders als C, schwach typisiert. Wenn man eine Variable anlegt, bekommt sie einen Typ zugewiesen, der zum Inhalt passt – wenn man Variablen unterschiedlicher Typen zusammenfügt, macht PHP eine der beiden Variablen passend (siehe Kasten). Ob das ein Vor- oder ein Nachteil ist, ist strittig: Auf der Seite ist schwache Typisierung für Einsteiger ohne informationstechnischen Hintergrund schneller zu lernen, und man kommt sehr weit, ohne sich je mit Integern, Strings und Floats zu beschäftigen. Andererseits kann man sich später unangenehme Fehler oder gar Sicherheitslücken einhandeln, wenn man die Konzepte nie gelernt hat und allzu lässig mit Typen umgeht. PHP macht es leicht, schlechten Code zu verfassen – das sollte man beim Lernen im Hinterkopf behalten.

Mit PHP 7.4, erschienen Ende 2019, kam etwas mehr starke Typisierung hinzu: Eigenschaften von Klassen können jetzt mit einem Typen versehen sein. Verpflichtend ist das allerdings nicht, und bis sich dieses Konstrukt durchsetzt, wird es sicher noch etwas dauern: Wer die Typisierung jetzt schon einsetzt, macht den Code nämlich inkompatibel zu PHP 7.3, das noch bis

Dezember 2021 Sicherheitsupdates bekommt.

Zu den größten Umbrüchen in der Sprache gehört die Einführung von objektorientierter Programmierung mit PHP 5 im Jahr 2004. Seitdem kann man viele Aufgaben sowohl funktional als auch objektorientiert lösen. Für Einsteiger hat das durchaus seinen Reiz: Sie können erste Erfahrungen im imperativen Stil sammeln und später auf Objektorientierung wechseln, ohne eine neue Sprache lernen zu müssen. Kritiker halten PHP dagegen vor, dass die Sprache inkonsequent sei, weil es häufig mehrere Wege zum gleichen Ergebnis gibt.

Zu den Stärken von PHP gehören die mitgelieferten Funktionen und Erweiterungen, die zahlreiche Probleme rund um Webentwicklung lösen. Es gibt zum Beispiel Anbindungen an Datenbanken oder Klassen zum Erzeugen und Bearbeiten von Bildern und PDF-Dateien. Wer ernsthaft in PHP entwickeln will, kommt um den Einsatz eines Frameworks nicht umhin. Zu den größten Angeboten gehören Laravel, Symfony und CakePHP. Die Frameworks lösen grundlegende Aufgaben wie Autorisierung und Routing und machen das Entwickeln von APIs oder Webseiten recht angenehm.

Für wen?

PHP ist noch immer die am weitesten verbreitete Sprache für dynamische Webseiten, außerhalb des Webs aber bedeutungslos. Für Programmierneulinge ist PHP ein guter Einstieg. Sie werden schnell erste Erfolgserlebnisse sammeln und funktionierende Anwendungen schreiben. Anfänger sollten sich aber rechtzeitig mit objektorientierter Programmierung vertraut machen.

Als erfahrener PHP-Entwickler schreiben Sie mit dem richtigen Framework zügig auch große Anwendungen – und darauf kommt es letztlich an. Die Sticheleien von C- oder Java-Entwicklern können Sie dann getrost ignorieren. (jam@ct.de) **ct**

```
<?php
$string = "Ich bin ein ";
echo $string . "String";

$integer = 5;
$float = 2.5;

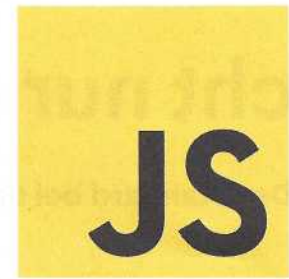
// PHP ist schwach typisiert.
// Integer und Float kann man problemlos addieren,
// heraus kommt ein Float:
$summe = $integer + $float;

if($summe > 7){
    echo $summe;
}
?>
```

Viele Elemente der Syntax wurden von C vererbt. So endet jede Zeile mit einem Semikolon. Mit Typisierung nimmt es PHP nicht so genau.

Späte Reife

JavaScript: (Schon lange nicht mehr nur) die Sprache des Webs



Quelle: github.com/voodoorkid/god/logo.js

Lange als Spielzeug belächelt, ist JavaScript aus dem Browser ausgebrochen und in rasantem Tempo zur mächtigen Allzweck-Sprache herangereift.

Von Herbert Braun

Obwohl schon 25 Jahre alt, entwickeln sich JavaScript und sein umfangreiches Ökosystem an APIs, Frameworks, Bibliotheken und Werkzeugen heute schneller denn je fort: Jährlich erscheinen JavaScript-Versionen mit neuen Features. Eineinviertel Millionen Pakete listet die Paket-Registry npm, bei GitHub belegt die Sprache Platz eins, bei StackOverflow Platz zwei: keine schlechte Karriere für eine Sprache, die im Frühjahr 1995 binnen weniger Wochen von einem einzelnen Entwickler zusammengeschraubt wurde.

Der Netscape-Mitarbeiter Brendan Eich wollte den Lisp-Dialekt Scheme in den Browser bringen, aber das Management drängte ihn in Richtung des boomenden Java. Der Name „JavaScript“ für den resultierenden Bastard war kaum mehr als PR. Die hektische Entwicklung führte zu problematischen Details, die wegen Abwärtskompatibilität bis heute Teil der Sprache sind. Viele haben mit der schwachen und dynamischen Typisierung der Variablen zu tun, die oft zu irritierenden Ergebnissen führt:

```
1 + '1'
```

funktioniert, ergibt aber etwas anderes als

```
'1' + 1
```

Dennoch war die dynamische Typisierung für die ursprünglich einfachen Aufgaben im Browser wie Aufklappenmenüs, Formularüberprüfung oder Bildergalerien keine schlechte Wahl. Entgegen dem Mainstream setzt JavaScript bei der Vererbung

nicht auf Klassen, sondern auf Prototypen. Statt von einer Blaupause erben Objekte Eigenschaften also direkt von anderen Objekten. Schmuckstück der Sprache sind die „First-Class“-Funktionen: Da Funktionen in JavaScript auch Objekte sind, können sie Variablen zugewiesen, als Funktionsparameter übergeben und verschachtelt werden. Anonyme Funktionen, Kontext-Änderungen, Closures und Currying sind kein Problem mit JavaScript. Diese Sprach-Features weisen über die Objektorientierung hinaus auf das funktionale Paradigma.

Vom Web 2.0 zum Server

Einige Meilensteine des JavaScript-Ökosystems: In den Nuller-Jahren brachte das „Web 2.0“ die ersten komplexen Browser-Anwendungen und JavaScript-Bibliotheken wie Prototype und jQuery. Google Chrome kompilierte JavaScript-Code als erster Browser zur Laufzeit, was gewaltige

```
// JavaScript hat Altlasten ...
(null < 0 || null == 0) !==
null <= 0;
NaN !== NaN;
Number() !== Number(undefined);
with ([1, 2]) push(3);
```

```
// Prototypen
const obj = {x: 1, y: 2};
// obj wird Prototyp von obj2:
const obj2 = Object.create(obj,
  {y: {value: 5}});
obj2.x === 1;
obj.x = 8; // Prototyp-Änderung
obj2.x === 8; // x kommt vom
Prototyp
obj.y = 9;
obj2.y !== 9; // obj2 hat eigenes y
delete obj2.y;
obj2.y === 9;
```

```
// TypeScript: Typdeklarationen
const sum = (...summands: number[]):
number => summands.reduce((a, b):
number => a + b);
sum(3, 4, 5) === 12;
```

TypeScript erweitert JavaScript um saubere Typüberprüfung.

Performance-Gewinne brachte und die Konkurrenz zum Nachziehen zwang. Node.js schließlich holte JavaScript aus dem Browser heraus. Damit lassen sich mit der lange unterschätzten Sprache Server-Anwendungen, Konsolenwerkzeuge, Build-Tools, Desktop- und Mobil-Apps umsetzen.

Für einen Standard braucht es ein Gremium, und dafür hielt merkwürdigerweise der Verband europäischer Computer-Hersteller – die ECMA – her. Seit 1997 ist die ECMA-Arbeitsgruppe TC39 für ES-262 verantwortlich: „ECMAScript“ ist der Standard, der JavaScript zugrundeliegt; in der Praxis sind beide Begriffe synonym. Seit Version 6 (ECMAScript 2015) gibt es jährliche Updates, und die Browser-Hersteller setzen die neuen Features zügig um; ES 2020 ist auf dem Weg und zum Teil bereits in Chrome, Firefox und Node.js implementiert.

Als größter Kritikpunkt bleibt die dynamische Typisierung der Variablen. Die von Microsoft erfundene Sprache TypeScript löst das Problem. Sie kompiliert nach JavaScript und erweitert dieses um optionale statische Typisierung und erweitert die Objektorientierung um Dinge wie private Methoden, Interfaces oder Dekoratoren. Zugleich dient TypeScript als Testgebiet für vielleicht kommende JavaScript-Features – und davon wird es sicher noch viele geben.

Für wen?

JavaScript eignet sich trotz ihrer Altlasten gut für Programmieranfänger. Die Sprache ist vielseitig einsetzbar und vor allem kann man sie Stück für Stück erlernen. Man kann etwa prozedural anfangen und dann objektorientierter arbeiten. Es gibt reichlich Dokumentation und gute Einstiegs-hilfen, etwa beim Mozilla Developer Network. Frontend-Entwicklung ist heute fast gleichbedeutend mit JavaScript-Programmierung. Aber auch für Backend-Anwendungen wird JavaScript zunehmend wichtiger. (jo@ct.de)

Einstieg und weitere Hilfen: ct.de/yyk6

Nicht nur eine Insel

Java: Der Standard bei der objektorientierten Programmierung



Mit der Wahl von Java als erster Programmiersprache kann man nicht viel falsch machen. Die Sprache ist übersichtlich, leicht zu lernen und weit verbreitet. Auch wenn der Name an die Web-Sprache JavaScript erinnert, hat Java damit praktisch nichts gemeinsam.

Von Andreas Linke

Java ist seit vielen Jahren eine der beliebtesten Programmiersprachen überhaupt und belegt in Rankings fast immer die Plätze 1 oder 2. Auch in der Ausbildung wird sie häufig verwendet. Das liegt daran, dass Java objektorientiert und stark typisiert ist und eine vergleichsweise einfache und übersichtliche Syntax hat. Es gibt nur wenige Schlüsselwörter, auf das Überladen von Operatoren wurde verzichtet und anders als etwa bei C++ gibt es keine Mehrfachvererbung. Jede Klasse ist Teil eines Pakets (Package), wobei Pakete direkt Verzeichnissen im Dateisystem entsprechen.

Der Java-Compiler übersetzt Quellcode nicht direkt in Maschinensprache, sondern in sogenannten Bytecode oder

Zwischencode. Dieser ist prozessorunabhängig und wird auf der Zielplattform von der Java Virtual Machine (JVM) ausgeführt. Um die Performance zu steigern, kommen dabei sogenannte Just-in-time-Compiler zum Einsatz, die den Bytecode je nach Bedarf während der Ausführung in Maschinensprache übersetzen. Java-Programme sind daher trotz der zusätzlichen Abstraktionsschicht in vielen Fällen ähnlich schnell wie Programme, die direkt in Maschinencode für die CPU übersetzt wurden.

Weder Windows noch macOS können von Haus aus mit Java-Code umgehen. Vor der ersten Ausführung von Java-Programmen muss daher eine Java-Laufzeitumgebung (JRE) installiert werden, die die JVM und die Standardbibliotheken sowie einige Tools mitbringt.

Die Freigabe von nicht mehr benötigtem Speicher übernimmt in Java ein asynchron laufender Garbage Collector. Leider heißt das nicht, dass Java-Programme keine Speicherprobleme kennen. Gerade bei größeren Anwendungen kommt es immer mal wieder zu den gefürchteten `java.lang.OutOfMemory`-Abstürzen.

Das Ausnahme-Konzept ist kompliziert: Die Sprache unterstützt sowohl deklarierte (checked) Exceptions als auch Runtime-Exceptions, die ohne Deklara-

tion an beliebiger Stelle erzeugt werden können. Die Notwendigkeit, Checked Exceptions in jeder Funktionsschicht entweder zu behandeln oder erneut zu deklarieren, kann den Code unübersichtlich machen und wird daher kontrovers diskutiert.

Ebenso wie die Lizenzpolitik: Java wurde ursprünglich von Sun entwickelt und kam im Zuge der Übernahme des Hardware-Herstellers in die Hände von Oracle. Wem die restriktiven Lizenzbedingungen von Oracle nicht schmecken, der kann auf die unter GPL stehende Implementierung OpenJDK ausweichen.

Die Sprache kennt viele moderne Konstrukte wie Typinferenz, anonyme Funktionen (Closures beziehungsweise Lambdas) oder Futures. Seit Java 11 lassen sich Übersetzung und Ausführung einer einzelnen Java-Datei wie bei Skriptsprachen in einem Schritt zusammenfassen:

```
java com\heise\beispiel\Beispiel.java
```

Neben der Java Platform, Standard Edition (Java SE), die für Windows, macOS und Unix-Derivate verfügbar ist, gibt es auch eine Java Enterprise Edition (Java EE) für die Verwendung auf dem Server. Rund um die Sprache ist ein ganzes Ökosystem von Frameworks entstanden, darunter GUI-Bibliotheken wie Swing oder JFace, JDBC für die Datenbank-Anbindung und Java Enterprise Beans für die Arbeit mit Business-Objekten. Nach wie vor werden die meisten Android-Apps in Java geschrieben, auch wenn dazu mit Kotlin (siehe S. 27) eine modernere, kompaktere Sprache zur Verfügung steht.

Für wen?

Java eignet sich ideal sowohl für Einsteiger als auch für professionelle Entwickler. Die Sprache ist einfach zu erlernen und es gibt eine nahezu unüberschaubare Menge an Literatur und Beispiel-Code. In den vergangenen Jahren kamen zahlreiche moderne Sprach-Features hinzu.

(hos@ct.de)

```
package com.heise.example;
// die Quelldatei muss im Verzeichnis com/heise/example liegen
// Code ist immer Teil einer Klasse
public class Beispiel {
    // Diese Methode wird automatisch aufgerufen
    public static void main(String[] args) {
        var str = "Ich bin ein ";
        System.out.println(str + "String");
        int in = 5;
        float fl = 2.5f;
        // einfache Typen werden automatisch erkannt und konvertiert
        var sum = in + fl;
        if(sum > 7) {
            System.out.println(sum);
        }
    }
}
```

Java-Code ist in Packages organisiert, die Verzeichnissen auf der Festplatte entsprechen.

Android in effizient

Kotlin: Moderne Java-Alternative für kompakteren Code



Bild: kotlinlang.org

Kotlin löst Java als Programmiersprache für Android-Apps ab. Sie glänzt mit kurzen und verständlichen Sprachkonstrukten, die Arbeit sparen und die Übersicht verbessern.

Von Pina Merkert

App-Entwicklung für Android war einst eine Tipporgie in Java. Dank Kotlin hat sich das gründlich geändert: Inzwischen reichen ein paar dutzend Zeilen Quellcode für eine eigene App. Zusätzlich hat Google viele Interfaces in Android und den Play-Services renoviert und an Kotlins kurzen und effizienten Stil angepasst. Kotlin entlehnt als moderne Sprache Konzepte aus funktionaler Programmierung und Skript-Dialekten, um kürzer, schneller und verständlicher zu werden, ohne die Vorteile der Java-Welt zu opfern.

Wie Java – nur besser

Javas Vorteile liegen in der plattformunabhängigen virtuellen Maschine (Java-VM) und im pedantischen Compiler. Kotlin erzeugt Programme für dieselbe VM, prüft jeden Typ und meckert über mögliche Fehler. Toll für Entwickler, die gewachsenen Java-Code pflegen: Kotlin bleibt vollständig kompatibel. Eine Java-Bibliothek hat kein Kotlin-Pendant? Kein Problem! Java-Klassen lassen sich ohne Umwege instanziiieren, und Kotlin stellt Setter und Getter sogar übersichtlich als Properties dar. Eine Java-Klasse muss auf Kotlin-Code zugreifen? Auch kein Problem! In Kotlin definierte Objekte stehen auch in Java zur Verfügung. Als besonderes Schmankerl für Umsteiger stellen die Kotlin-Spracherfinder von JetBrains einen Konverter bereit, der Java-Code automatisch übersetzt. Java-Profis können die neue Sprache damit in Rekordzeit lernen.

Kotlins kurzen und dennoch verständlichen Sprachkonstrukten merkt man an, dass JetBrains, Hersteller der berühmten Java-IDE IntelliJ-Idea, den Alltag und die Nöte von Java-Entwicklern gut kennt. Kotlin stellt nämlich überall dort Abkürzungen

bereit, wo Java-Entwickler stupiden Boilerplate-Code abtippen müssen. Gute IDEs haben manchen dieser Tipp-Exzesse zwar auch in Java den Schrecken genommen, der längliche Java-Code ist aber letztlich nicht so gut lesbar wie das funktionsgleiche Kotlin-Äquivalent. Und weniger Zeilen Code bedeuten auch weniger Fehler.

Google empfiehlt Kotlin für alle neuen Android-Apps. Dass Android noch viele Java-Interfaces enthält, stört ja nicht; Programmierer erkennen diese lediglich an den längeren Klassennamen. Ein weiteres Produkt der Zusammenarbeit mit JetBrains ist die hervorragende (und kostenlose) Entwicklungsumgebung Android Studio. Die hilft Entwicklern mit Inspections, die zu gutem Programmierstil auffordern. Zum Konvertieren von Java nach Kotlin genügt in Android Studio ein Rechtsklick.

Ein paar Beispiele für Sprachkonstrukte, die einen Blick wert sind: Die when-Anweisung ist eine elegante Variante von switch...case. listOf() erzeugt Arrays, 0..10 einen Iterator für Zahlen von 0 bis 10 und for(item in list) iteriert über beides ohne

Zählvariable. do {} while (bedingung) definiert Schleifen, die am Ende prüfen. let {} prüft implizit auf null und sorgt mit einem eigenen Scope für Ordnung. Ausführliche Code-Beispiele finden Sie unter ct.de/yumk.

Für wen?

Kotlin ist eher eine Sprache für Umsteiger als für Einsteiger. Android-Entwickler sollten alle zu Kotlin wechseln. Da dafür kein Rewrite bestehender Apps nötig ist, geht das ohne Schmerz. Außerhalb der Android-Welt hat Kotlin bislang weniger Fuß gefasst. Für Entwickler von Java-Anwendungen für den Desktop ist Kotlin dank der perfekten Kompatibilität dennoch ein Blick wert. JetBrains arbeitet auch an einem nativen Compiler, der direkt Maschinencode produzieren soll. Die Java-VM würde so überflüssig. Sollte dieser Ansatz Erfolg haben, wird Kotlin in Zukunft auch C++, Rust und Go Konkurrenz machen. (pmk@ct.de) **ct**

Kotlin-Doku und Online-Spielwiese:
ct.de/yumk

```
data class Wood(var fuel: Int = 0) // Data Classes kapseln Property's
//annotation Functions erweitern beliebige Klassen um Methoden
fun Wood.burn(): String {
    // Int ist eine Zahl in Kotlin, Integer die Klasse aus Java
    val lost_fuel: Integer = Integer(this.fuel)
    this.fuel = 0
    // String Templates sparen format()-Funktionen
    return "Heat for $lost_fuel Joule!"
}
fun main() {
    // Variablen sind stark typisiert. Ein ? erlaubt null
    var logOfWood: Wood? = null
    // Conditional Expressions sparen Zeilen
    println(if (logOfWood != null) logOfWood.fuel.toString() else "kein Ast!")
    // Konstruktoren von Data Classes akzeptieren Parameter mit Namen
    logOfWood = Wood(fuel=1500)
    // let erzeugt einen eigenen scope mit dem Objekt "it"
    println(logOfWood.let{ it.burn() })
    // Wie bei Java gibt es einen Garbage Collector
    logOfWood = null
    // let verträgt sich mit einer Prüfung auf null mit ?
    logOfWood?.let{ println(it.burn()) }
}
```

Kotlin bringt schicke moderne Sprachkonstrukte in die Java-Welt. Trotzdem bleibt der Code kompatibel.

Tu, was du willst

Scala: Objektorientiert und funktional, das Beste aus beiden Welten



Quelle: scala-lang.org

Objektorientierung ist schön, funktionale Programmierung auch – zusammen ist beides noch besser. Dieses Versprechen löst Scala ein. Die Sprache eignet sich nicht nur für experimentierfreudige Java-Entwickler.

Von Sylvester Tremmel

Scala wurde von Martin Odersky begründet, der sich zuvor um den Java-Compiler javac verdient gemacht hatte. Diesen Hintergrund merkt man Scala an, das primäre Kompilier-Ziel ist die Java Virtual Machine (JVM). Das hat den schönen Effekt, dass Scala mit Java kompatibel ist: Ein in Java geschriebenes Programm kann in Scala geschriebene Bibliotheken nutzen. Der Trick funktioniert aber auch anders herum: Scala-Programme können Java-Bibliotheken nutzen, was dem Scala-Programmierer die immense Vielfalt des Java-Ökosystems eröffnet.

Zwar haben auch Sprachen wie Java in letzter Zeit funktionale Features bekommen, aber Scala hat diesbezüglich nicht nur Pionierarbeit geleistet, sondern geht nach wie vor einige Schritte weiter: Pattern Matching, Currying, Endrekursion – fast alle funktionalen Wünsche werden erfüllt.

Außerdem setzt Scala die Konzepte konsequenter um als viele andere Sprachen: Jeder Wert ist ein Objekt, auch Ganzzahlen wie 7 oder Strings wie "Hallo". Auch Funktionen sind Werte – und damit Objekte. All solche Objekte kann man über Klassen definieren und instanziiieren (Scala nutzt klassenbasierte Vererbung) oder sie direkt angeben – und wie andere Objekte auch, haben sie Methoden:

```
3.max(4) // -> 4
"Hallo".toUpperCase() // -> "HALLO"
(x:Int) => x * 2).apply(3) // -> 6
```

Das alles sortiert Scala in eine einheitliche Hierarchie von Typen: Scalas Typsystem ist statisch, aber man kann sich – wie bei

vielen funktionalen Sprachen – die meisten Typangaben sparen, weil der Compiler sie selbst ableitet: Nach `val x = 3` weiß der Compiler, dass `x` vom Typ `Int` ist, schließlich hat man dem Wert gerade die Ganzzahl 3 zugewiesen. Wer will, kann den Typ auch explizit angeben: `val x:Int = 3`.


Überhaupt hat man bei Scala sehr oft die Wahl: Wer statt Werten, die im funktionalen Stil unveränderbar sind, lieber Variablen nutzt, der schreibt eben `var x = 3`. Die Wahlmöglichkeiten setzen sich fort in der Standardbibliothek, die zum Beispiel fast jede Art von `Collection` in den Varianten `mutable` und `immutable` anbietet.

Auch die Syntax von Scala lässt viele Freiheiten: Zum Beispiel sind Semikolons am Zeilenende optional, geschweifte Klammern um Blöcke nicht nötig, wenn es nur um eine Zeile geht, und Namen können fast beliebige Zeichen enthalten. Das geht so weit, dass Scala eigentlich keine Operatoren kennt, sondern nur entsprechend benannte Methoden: `3 + 4` ist der Aufruf der Methode `+` auf dem Objekt `3`, mit dem Objekt `4` als Parameter. Man kann auch `3.+(4)` schreiben. Durch solche und andere Features (Stichwort: `implicit`, siehe Listing) entfaltet Scala enorme Flexibilität.

Der Scala-Styleguide hilft dabei sie zu beherrschen (alle Links: ct.de/ypn4).

Für wen?

Scala eignet sich hervorragend für Java-Programmierer, die über den Tellerand blicken wollen – sie büßen durch Experimente mit Scala nicht mal Interoperabilität ein. Unternehmen nutzen das, um graduell auf Scala umzusatteln oder Scala und Java parallel zu nutzen. JavaScript-Entwickler sollten einen Blick auf den Cross-Compiler `Scala.js` werfen und Fans rein funktionaler Sprachen wie Haskell, die auf die JVM wollen oder müssen, werden Scala ohnehin im Auge haben – sie sollten einen Blick auf streng funktionale Bibliotheken wie `Scalaz` werfen.

Unter fachkundiger Anleitung bietet sich Scala auch für Programmieranfänger an. Die Sprache ist elegant, erlaubt das Erlernen verschiedener Programmierkonzepte und bietet Zugriff auf reichhaltige Bibliotheken. Mit `Scala Native` besteht zudem die Option, nativen Code zu generieren. (syt@ct.de) 

Anleitungen und Dokumentation:
ct.de/ypn4

```
import scala.language.implicitConversions

class Cmplx(val re: Float, val im: Float) { // Klassendefinition
  def +(that: Cmplx) = Cmplx(this.re + that.re, this.im + that.im)
  def -(that: Cmplx) = Cmplx(this.re - that.re, this.im - that.im)
  def *(that: Cmplx) = Cmplx(this.re * that.re - this.im * that.im,
    this.re * that.im + this.im * that.re)

  override def toString = if (this.im >= 0) s"${this.re}+${this.im}i"
    else s"${this.re}${this.im}i"
}

object Cmplx { // Companion-Objekt zur Klasse
  implicit def frmReal(x: Float) = new Cmplx(x, 0) // Für automatische
  // Konvertierung von reellen Zahlen zu komplexen Zahlen
  def apply(re: Float, im: Float) = new Cmplx(re, im) // Das überall
  // genutzte "Cmplx(...)" ist die Kurzform von "Cmplx.apply(...)"
}

Cmplx(1,2) * (5f + Cmplx(3,-7)) // -> Cmplx = 22.0+9.0i
```

Flexible Syntax: Eine kleine Klasse samt zugehörigem Objekt und schon kann man mit komplexen Zahlen rechnen, als ob sie ein Feature der Sprache wären.

Elegante Entschleunigung



Quelle: python.org

Python: Schöner Code für Einsteiger und Wissenschaftler

Python-Code ist kurz und verständlich. Mit dem Interpreter gehen Experimente schnell, während die Programme verglichen mit C langsam laufen. KI-Forscher entwerfen damit trotzdem schnelle neuronale Netze.

Von Pina Merkert

Python begeistert Entwickler, die verständlichen Code schreiben möchten. Gut für die Übersicht: Einrückungen statt Klammern definieren Blöcke; in jeder Zeile steht nur ein Befehl ohne Semikolon. Besonders kompakter und lesbarer Code gilt als „pythonic“ und erntet das Lob der Community. Gleichzeitig kann man mit Python alles programmieren: Vom Bash-Skript-Ersatz über grafische Desktop-Programme bis zum neuronalen Netz – Python kann alles!

Das Programm `python` interpretiert den Code direkt und überspringt daher das Kompilieren in Maschinencode. Deshalb eignet sich Python besonders für Programmier-Experimente. Python-Code ist schnell geschrieben und schnell ausprobiert. `python` ohne Argument startet eine interaktive Konsole, die getippte Zeilen sofort ausführt – toll zum Testen, ob eine Anweisung wie gedacht funktioniert. Noch bequemer geht das mit Jupyter-Notebooks im Browser, die nebenbei auch Text und Grafik anzeigen und schon manche GUI überflüssig gemacht haben.

Der Preis für den Interpreter ist Rechenzeit: Der gleiche Algorithmus braucht in Python bis zu 100-mal so lang wie in C. Python selbst und die meisten Bibliotheken sind daher in C geschrieben und stellen nur ein „pythonic“ Interface bereit. Im Alltag merkt man das daran, dass Pythons Paketmanager `pip` öfter mal den C-Compiler des Systems anwirft und scheitert, wenn es den nicht findet.

Angesichts der Langsamkeit überrascht es, dass Python die Sprache aller

KI-Forscher ist. Die nutzen Frameworks wie Caffe oder TensorFlow, die ihre Berechnungen automatisch für die vorhandene Hardware optimieren und dabei beispielsweise den CUDA-Compiler anwerfen, um auf der Grafikkarte zu rechnen. Für die Auswertung der Ergebnisse nutzen die Wissenschaftler optimierte Frameworks wie Numpy und Pandas. Dass ein paar Zeilen Python die aufwendigen Berechnungen antoßen und verwalten, spielt für die Laufzeit letztlich keine Rolle. Die Bequemlichkeit beim Experimentieren bleibt jedoch.

Generell gibt es für fast jede C-Bibliothek einen Python-Wrapper. Dank derer sieht es so aus, als wäre jedes Kunststück, zu dem der Rechner fähig ist, auch Teil von Python. Damit steht man auf den Schultern der vielen Entwickler, die ihre Lösungen längst in Bibliotheken verpackt haben. Die passende Doku lagert im Web meist bei readthedocs.org.

Für wen?

Python empfiehlt sich für Anfänger. Die Sprache erlaubt jede Art von Programm und die Kenntnisse wachsen mit jedem

neuen Projekt. Auch für funktionale oder probabilistische Programmierung muss man die Python-Welt nicht verlassen.

Maschinelles Lernen sollte man gar nicht erst versuchen in einer anderen Sprache zu entwerfen: Das Python-API ist bei TensorFlow & Co. stets die vollständigste und verbreitetste Schnittstelle.

Es gibt Frameworks für alles: Mit Django gelingen große Webapplikationen, mit Flask schreibt man schnell HTTP-Microservices in Python. Das passende Web-Frontend sollte man allerdings mit einem JavaScript-Framework wie React schreiben, da Browser kein Python verstehen. Auch Apps für iOS und Android entstehen besser in Swift oder Kotlin. Gegen Desktop-Anwendungen in Python spricht dagegen wieder nichts.

Python ist ein Schweizer Messer, mit behäbigem Interpreter, was man meist mit Bibliotheken ausgleichen kann. Die Community feiert Ästhetik und Verständlichkeit, was Anfänger wie Profis gleichermaßen beglückt. (pmk@ct.de) **ct**

Python.org und Anaconda: ct.de/ybra

```
# -*- coding: utf-8 -*-
import os
print("Diese Anweisung wird ausgeführt, sobald Python die Datei importiert.")

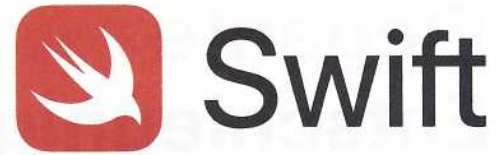
def funktion_mit_parameter(x):
    print("Der Typ einer Variable kann sich ändern:", type(x), x)
    x = 2
    return "str() wandelt ein " + str(type(x)) + " in einen String um: " + str(x)

if __name__ == "__main__":
    print("Diese führt Python nur aus, wenn die Datei direkt aufgerufen wird.")
    print(funktion_mit_parameter({"{}-Klammern": "definieren dictionaries.",
                                "float": 2.3, "liste": [1, "zwei", int]}))
    typ_der_klammer, wort, _ = (tuple, 'Einrückungen', "definieren Blöcke:")
    for i in range(4):
        print(wort, " " * i + "Zeile", i)
    wortliste = 'String-Funktionen-erleichtern-die-Arbeit.'.split("-")
    # Kommentar: List-Comprehensions (for definiert neue Liste) sparen Platz
    print(" ".join(["*" + wort + "*" for wort in wortliste]))
    print("Module nehmen Arbeit ab. Pfad:", os.path.abspath(os.path.curdir))
```

Ungewohnt: Python nutzt Einrückungen statt Klammern, um im Code Blöcke zu bilden. Programme sind dadurch gut lesbar.

Schnell und sauber

Swift: Die Apple-Sprache



Swift ist der Shooting-Star unter den modernen Programmiersprachen. Kaum eine andere Sprache hat in so kurzer Zeit eine so große Fangemeinde aufgebaut. Bei der Programmierung von Apps für iPhone, iPad und Mac kommt man kaum an ihr vorbei.

Von Andreas Linke

Apple hat Swift als Nachfolger für das schon recht betagte, auf C basierende Objective-C von Grund auf neu designt. Seit ihrer Erstveröffentlichung 2014 hat die Sprache eine rasante Entwicklung hingelegt, mit mehreren inkompatiblen Änderungen in der Syntax. Inzwischen versprechen die Macher aber nur noch kleine und kompatible Erweiterungen.

Hohe Geschwindigkeit bei der Ausführung und gute Lesbarkeit des Codes waren wichtige Design-Kriterien. So gibt es ein spezielles `guard`-Statement, mit dem man Voraussetzungen prüfen kann, unter denen der nachfolgende Code laufen soll. Sind die Bedingungen nicht erfüllt, muss die Funktion verlassen werden. Swift unterscheidet konstante Werte (mit `let` deklariert) und änderbare Variablen (mit `var` deklariert). Das erlaubt dem Compiler eine Reihe von Optimierungen und sorgt nebenbei für eine bessere Verständlichkeit des Codes.

Als eine von nur wenigen Sprachen erzwingt Swift die saubere Deklaration von optionalen Attributen, also Variablen, die `nil` (null) sein können, mit `var attr:Type?`. Das verhindert die gefürchteten Null-Referenzen – aber nur, wenn man diesen Mechanismus nicht mit dem Ausrufezeichen-Operator wieder aushebelt: `var attr:Type!` besagt, dass die Variable beim ersten lesenden Zugriff nicht mehr `nil` ist, sie kann es aber zum Instantzeitpunkt der Klasse noch sein.

Aufzählungen (enums) sind in Swift sehr mächtig und können sogar je nach Fall unterschiedliche Attribute enthalten, so

Objekt enthalten und der Erfolgsfall das Ergebnis – die Syntax ist allerdings etwas gewöhnungsbedürftig. Mit Extensions lassen sich Klassen – sowohl eigene als auch aus Frameworks eingebundene – um weitere Methoden und Eigenschaften erweitern.

Die Speicherverwaltung, also die Freigabe von nicht mehr verwendeten Instanzen, erfolgt in Swift über Automatic Reference Counting. Der Vorteil: Anders als bei einem asynchron laufenden Garbage Collector werden Instanzen immer in klar definierter Reihenfolge abgeräumt, es gibt keine überraschenden Pausen im Programmablauf. Das bedeutet aber auch, dass zirkuläre Referenzen, also Klassen oder Closures, die einander gegenseitig referenzieren, vermieden werden müssen. Das lässt sich über schwache (`weak`) Referenzen erreichen.

Für erste Gehversuche genügt ein iPad oder iPhone mit der kostenlosen App „Swift Playgrounds“. Ernsthafte Entwicklung ist aber nur auf Mac-Hardware möglich. Dort nutzt man die kostenlose und äußerst mächtige Entwicklungsumgebung Xcode, die allerdings etwas Einarbeitungszeit benötigt. Die Probleme, die Xcode anfangs beim Umgang mit Swift-Code hatte, hat

Apple mittlerweile recht gut im Griff. Trotzdem gibt es immer noch Situationen, in denen die IDE 100 Prozent der CPU-Zeit verbraucht oder mit eigenartigen Fehlermeldungen aussteigt, etwa bei dem modernen funktionalen UI-Ansatz Swift UI.

Die Dokumentation der Sprache durch Apple ist vorbildlich. Bedingt durch die große Verbreitung von Apple-Geräten findet man zu fast jedem Problem eine Lösung auf Stackoverflow oder Spezial-Sites wie raywenderlich.com. Aufgrund der dynamischen Sprachentwicklung sind aber immer wieder Beispiele in veralteter Syntax dabei – das macht es Einsteigern schwer.

Für wen?

Swift eignet sich weniger gut für Anfänger. Erfahrene Programmierer schätzen nach etwas Einarbeitungszeit die klare und kompakte Darstellung des Codes sowie die hohe Ausführungsgeschwindigkeit. Server Side Swift ist ein interessantes und viel versprechendes Projekt, die Sprache auch für Projekte außerhalb des Apple-Universums verfügbar zu machen.

(hos@ct.de) **ct**

Literatur zum Einstieg: ct.de/y7nc

```
import Foundation

class Beispiel {
    func test() {
        var str = "Ich bin ein "
        print(str + "String.")

        let intVar = 5
        let flVar: Float = 2.5

        // Typen müssen explizit konvertiert werden
        let sum = Float(intVar) + flVar

        if sum > 7 {
            print(sum)
        }
    }
}

Beispiel().test()
```

Swift unterscheidet veränderbare Variablen (`var`) und konstante Werte (`let`). Klassen definiert man mit „`class`“.



Schneller vorwärts

Go: Effizient programmieren für fast alle Plattformen

Go ist mehr als eine akademische Spielerei von Google-Mitarbeitern: Mit der Programmiersprache entstehen Kommandozeilenprogramme oder Serverdienste. Die größten Anwendungen im Cloud-Umfeld wurden in Go entwickelt.

Von Jan Mahn

Die Programmiersprache Go entstand als Privatprojekt von Google-Mitarbeitern, die vor allem vom langsamen C-Kompilieren genervt waren. Nicht nur dank eines schnellen Compilers stieg Go schnell zur beliebtesten Sprache bei Cloud- und Container-Entwicklern auf. Populärstes Go-Projekt ist Kubernetes mit 1,4 Millionen Zeilen Go-Code.

Ein C-Nachbau ist Go aber nicht: Die Syntax erinnert höchstens entfernt an C und seine Verwandten. Stattdessen überrascht sie mit ungewohnten Konzepten, die sich nach kurzer Umgewöhnung als erstaunlich durchdacht entpuppen. So ist die Sprache zum Beispiel objektorientiert, verwendet aber keine Klassen.

Variablen sind typischer und der Compiler beklagt sich mit aussagekräftigen und verständlichen Fehlermeldungen, wenn man nachlässig mit Typen umgegangen ist. Eine der Stärken der Sprache ist die Umsetzung von Nebenläufigkeit – mit einem vorangestellten `go` wird ein Befehl nebenläufig ausgeführt. Weil die Umsetzung so einfach ist, kommt man als Entwickler gar nicht auf die dumme Idee, etwa mit Netzwerkaktivitäten den Haupt-Thread zu blockieren.

Go-Code wird kompiliert – und im Compiler stecken viele gute Ideen: Cross-Compiling ist seit 2015 eingebaut, ausführbare Binärdateien für Windows, Linux und macOS für gängige und seltene Prozessorarchitekturen erzeugt man ohne Einrichtungsaufwand von jeder Plattform. Die fertigen Binaries haben keine externen Abhängigkeiten – wer einen Server-


dienst oder ein Kommandozeilenprogramm entwickeln muss, wird Go schätzen. Adapter für GUI-Frameworks gibt es auch, für grafische Desktop-Programme ist Go aber nicht die erste Wahl.

Für Programmier-Einsteiger ist Go nicht unbedingt empfehlenswert. Sie sollten die Konzepte lieber in einer Skriptsprache wie Python oder JavaScript lernen und erst dann auf Go umsteigen. Go-Entwickler mögen oft gute Programmierer sein, von systematischen Dokumentationen halten sie meist nichts oder haben schlicht keine Zeit dafür. So passiert es häufig, dass man eine Bibliothek findet, die das Problem lösen könnte. Aber statt eine hilfreiche Dokumentation mit einem Beispiel zu schreiben, verweisen die Macher nur auf die trockene Klassendokumentation. Konkrete Beispiele muss man sich in Foren oder GitHub-Issues zusammensuchen. Für Umsteiger eine entmutigende Arbeit. Dass fast alle Dokus und Blogs über Go kein Syntax-Highlight beherrschen, ist ebenfalls nicht hilfreich.

Die größte Schwachstelle von Go war aber eine architektonische. Die Erfinder haben den Fehler mittlerweile eingesehen: Ursprünglich gab es auf jeder Entwickler-

maschine einen Pfad (den `GOPATH`), in dem alle Projekte zusammen mit dem Go-Compiler und nachgeladenen Paketen lagen. Probleme mit unterschiedlichen Versionen von Abhängigkeiten waren an der Tagesordnung. Seit Version 1.14 (erschienen Ende Februar 2020) ist das neue Konzept von „Go Modules“ aus der Testphase entlassen und Projekte können unabhängig von anderen auf Ihrer Festplatte liegen. Wenn Sie Go lernen, sollten Sie darauf achten, gleich mit „Go Modules“ zu arbeiten.

Für wen?

Go ist eine Sprache für Fortgeschrittene, nicht nur aus dem Cloud- und Containerumfeld. Die Syntax ist zunächst ungewohnt, wenn man C und seine Verwandten gewohnt ist. Schmutzige Tricks beim Programmieren gewöhnt Ihnen der Compiler knallhart ab und zwingt Sie zu gutem Code. Im Gegenzug für etwas Umgewöhnung bekommen Sie schnelle und leicht zu verteilende Programme sowie Nebenläufigkeit mit vergleichsweise wenig Schmerzen – und als Go-Entwickler sind Sie gefragter Experte. (jam@ct.de) 

Einstieg in Go: ct.de/yspb

```
01 package main
02
03 import (
04     "fmt"
05 )
06
07 func main() {
08     str := "Ich bin ein "
09     fmt.Println(str + "String.")
10
11     // Go-Entwickler lieben kurze Variablennamen:
12     in := 5
13     fl := 2.5
14
15     // Vorsicht mit Typen, umwandeln ist leicht:
16     sum := float64(in) + fl
17
18     if sum > 7 {
19         fmt.Println(sum)
20     }
21 }
```

Der Go-Compiler achtet auf die Typen von Variablen.

Sprache der Statistiker

R: Daten analysieren und visualisieren



Die Sprache R richtet sich an Wissenschaftler und Datenanalysten. Sie berechnet vom Median bis Konfidenzintervall alles, was das Statistiker-Herz begehrt – und bereitet Daten in übersichtlichen Grafiken auf.

Von Achim Barczok

R ist von Statistikern für Statistiker entwickelt. Mit diesem Hintergrund eignet sich die Spezialsprache hervorragend, um Datensätze zu analysieren und mit wenigen Code-Zeilen Korrelationen, Wahrscheinlichkeiten, Varianzen und viele andere Werte einer Stichprobe zu ermitteln. Beliebte ist sie vor allem an Unis, in der Forschung und im neueren Feld der Data Science in Unternehmen.

Viele nutzen R im Alltag vor allem als mächtigen Taschenrechner, den man per Konsole bedient und mit Daten aus CSV-Dateien, Datenbanken, Excel-Tabellen oder Statistikprogrammen wie SPSS füttert. Alle wichtigen Berechnungsmodelle in der Statistik sind enthalten oder über Bibliotheken zu bekommen. Doch R kann noch mehr: Mit der Programmiersprache lassen sich Datensätze bereinigen, Auswertungsverfahren skripten und automatisieren sowie eigene Modelle entwickeln und beschreiben.

Die Open-Source-Community rund um R ist sehr aktiv und trägt dazu bei, dass die Sprache inzwischen mit den wichtigsten Datenbankformaten spricht, an LaTeX andockt und schnell neue Modelle in der Statistik übernimmt. Das hat im akademischen Bereich dazu geführt, dass sich die Sprache zu einer echten Alternative für teure kommerzielle Produkte wie SPSS, Stata und ArcGIS entwickelt hat.

Man kann per Konsole, aber auch prima per GUI mit R arbeiten: Die Software der Wahl ist das für nichtkommerzielle Zwecke kostenlose RStudio. Auch die Integration in die Datenanalyse-IDE Anaconda und das beliebte Online-Notebook Jupyter

haben geholfen, R einem größeren Publikum schmackhaft zu machen.

Beliebt ist R auch wegen seiner Visualisierungsfähigkeiten. Mit `pie()`, `hist()` oder `plot()` etwa sind im Nu gängige Diagrammtypen zusammengestellt und exportiert. Empfehlenswert ist die Zusatzbibliothek `ggplot2`, die Daten deutlich schöner veranschaulicht als das Standardpaket. Mit weiteren Paketen modelliert und visualisiert R auch Geodaten für die räumliche Statistik (Spatial Analytics).

Als Programmiersprache ist R relativ eingängig und man kommt relativ weit, ohne die grundsätzlichen Prinzipien verstehen zu müssen. Aber R bringt eigentlich alles mit, was man von einer modernen Sprache erwartet. Sie ist objektorientiert – auch Variablen, Funktionen und Operatoren sind Objekte. Mit Schleifen und Nutzereingaben vereinfacht man wiederkehrende Datenanalysen und baut Funktionen und Klassen für eigene Modelle.

Das populäre Package `shiny` macht R zur serverseitigen Programmiersprache – was sich aufgrund der verfügbaren Covid-19-Daten zur Zeit besonders großer Beliebtheit erfreut und zahlreiche Online-Dashboards hervorgebracht hat. Auch sonst findet man haufenweise Erweiterun-

gen, etwa für Bereiche wie Machine Learning, Genomdaten und Textanalyse.

Für wen?

R ist die richtige Wahl für Wissenschaftler und Datenanalysten, die wenig Erfahrung mit Programmiersprachen haben, aber in ihrem Bereich mit SPSS, Excel und ähnlichen Programmen an ihre Grenzen kommen. Sie führen damit auch komplexe Analysen in wenigen Schritten durch, erstellen Diagramme und beschreiben eigene Modelle. Ein typischer Anwendungsfall sind Berechnungen und Datenvisualisierungen für eine Forschungsarbeit. Fürs „klassische“ Programmieren eignet sich R kaum.

Schwächen hat R beim Verarbeiten von Daten aus dem Netz, extrem großen Datensätzen und Echtzeitdaten – dafür hat sich eher Python durchgesetzt. Auch bei KI und Machine Learning läuft Python R den Rang ab. Häufig lassen sich beide Sprachen aber auch hervorragend kombinieren – beispielsweise wenn man Daten von Webseiten per Python aggregiert und sie später mit R analysiert.

(acb@ct.de) 

Programme, Beispiele: ct.de/ybkn

```
id <- c(1:10)
seiten <- c(2,1,1,1.66,1,0.5,2,2,1,0.75)
woerter <- c(1367,607,706,852,513,334,1417,1196,659,407)
ct_df <- data.frame(id,seiten,woerter)
# Vektoren in Data Frames lassen sich wie Tabellenspalten nutzen

quantile(ct_df$woerter)[2]
# Für Informatiker verwirrend: R zählt ab 1, nicht ab 0

summary(ct_df$woerter)
# Summary fasst Kennziffern der deskriptiven Statistik zusammen

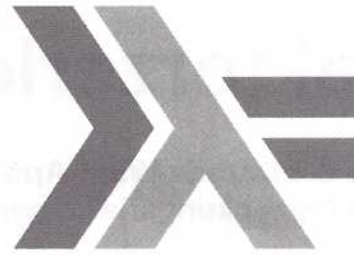
korrelation <- cor(ct_df$seiten,ct_df$woerter)
regression <- lm(ct_df$seiten~ct_df$woerter)
# Korrelationskoeffizient und lineare Regression ermitteln

plot(ct_df$woerter, ct_df$seiten, xlab="Anzahl Wörter", ylab="Anzahl Seiten",
     main="ct-Artikel", sub=paste("Pearson's r = ", round(korrelation,2)))
abline(regression, col="blue")
# Das Streudiagramm setzt die Variablen in Bezug
```

Mit wenigen Zeilen sind in R Daten importiert, analysiert und visualisiert.

Aus dem Elfenbeinturm

Haskell: Funktionale Programmierung für fehlerarme Software



Quelle: haskell.org

Die schönste Geistesverrenkung, seit es Programmiersprachen gibt und hochgradig suchterzeugend: Das ist Haskell. Nach dreißig Jahren Entwicklung ist die ursprünglich rein akademische Sprache reif für die Praxis.

Von Harald Bögeholz

Haskell gilt als schwierig zu lernen und als praxisuntaugliches Spielzeug für Akademiker. An Ersterem ist was dran und Haskell ist sicherlich ein Underdog unter den Programmiersprachen. Es taugt aber sehr wohl für praktische Großprojekte. So hat etwa Facebook vor einigen Jahren seine Software zur Spam-Erkennung komplett auf Haskell umgestellt und dabei sogar Code an die Open-Source-Community zurückgegeben. Apropos Open Source: Das Haskell-Ökosystem ist komplett quell-offen, der optimierende Compiler ghc läuft unter Linux, Windows und macOS.

Wer bereits in einer imperativen Sprache wie C++, Java oder Python programmiert, muss bei Haskell komplett umlernen, denn die Sprache ist rein funktional. Ein Programm besteht aus Funktionen im mathematischen Sinne: Ihr Rückgabewert hängt nur von den Eingabewerten ab. Ruft man eine Funktion wiederholt mit denselben Eingabewerten auf, so ist das Ergebnis garantiert immer dasselbe.

Es gibt keine veränderlichen Variablen, keine Objekte, die im Laufe der Zeit ihren inneren Zustand ändern, alles ist konstant. Die klassische for-Schleife existiert deshalb in Haskell so nicht: Wie, $i=2$? Eben hast Du noch gesagt $i=1$! Stattdessen sind Listen und Rekursion die Grundbausteine von Algorithmen. Wer von einer imperativen Sprache umsteigt, muss daher neu denken lernen. Statt einer for-Schleife heißt es nun zum Beispiel: Wende eine Funktion auf die Liste der Zahlen von 1 bis n an.

Weil Funktionen keine Seiteneffekte haben, ist es leichter, sich von ihrer Korrektheit zu überzeugen oder diese gar,

formal zu beweisen. Hängt die Eingabe einer Funktion B nicht von der Ausgabe der Funktion A ab, so ist es egal, in welcher Reihenfolge sie ausgewertet werden. Vielleicht gleichzeitig – oder, und das ist der Standard in Haskell, nur wenn nötig.

Diese Bedarfsauswertung, englisch lazy evaluation, hat mehrere Vorteile. Man kann elegant mit unendlichen Listen hantieren, ohne Endlosschleifen zu produzieren. Oder Funktionen mit short-circuit evaluation selbst definieren, ein Verhalten wie beim logischen Und-Operator $a \ \&\& \ b$, wo b nur ausgewertet wird, wenn a wahr ist. Und man kann sich beim Benchmarken ins Knie schießen, denn `timer_start`; rechnen; `timer_stop`; dauert genau null Sekunden, wenn das Ergebnis der Berechnung nicht verwendet wird.

Eine Stärke von Haskell ist das strenge Typsystem – und, dass man es kaum sieht. Jede Funktion, jeder Parameter, jede Variable hat einen festen Typ, den der Compiler durch Analyse des gesamten Quelltexts ermittelt (Typ-Inferenz). Es gilt als guter Stil, Typen hinzuschreiben, doch hauptsächlich für menschliche Leser. Der Compiler wüsste sie auch so.

Die leichtgewichtige Syntax fördert die Verwendung selbst definierter Datentypen, was aufgrund der statischen Typprüfung Fehlern vorbeugt. So ist es eine typische Erfahrung im Haskell-Alltag, dass es vielleicht lange dauert, bis der Compiler nicht mehr über Typen meckert. Aber wenn das Programm erst mal kompiliert, ist es auch fehlerfrei (hüstel).

Für wen?

Haskell erfordert mathematisches, abstraktes Denken. Es mag etwas länger dauern, funktionale Programmierung zu durchschauen, aber der Lohn ist kompakter und fehlerarmer Code. Selbst wenn man hinterher in einer anderen Sprache programmiert, öffnet Haskell die Augen für all die funktionalen Konstrukte und sonstigen Anleihen, die moderne Sprachen bei Haskell genommen haben. Einsteiger oder lösungsorientierte Praktiker haben mit einer Sprache wie Python schnellere Erfolgserlebnisse.

(jam@ct.de) 

Beispiele und Dokumentation:
ct.de/yu43

```
-- Demo: komplexe Zahlen, selbst definiert statt aus der Bibliothek
data Complex = Complex Double Double
Complex a b .+ Complex c d = Complex (a+c) (b+d)
Complex a b .* Complex c d = Complex (a*c - b*d) (a*d + b*c)
abs2 (Complex a b) = a*a + b*b -- Betragsfunktion: abs2 z = |z|^2

f c z = z .* z .+ c -- Iterationsvorschrift für die Mandelbrot-Menge
mandel c = iterate (f c) (Complex 0 0) -- unendliche Liste

-- Typ-Signaturen (nächste Zeile) sind guter Stil, aber optional
dot :: Complex -> String
dot c = if null xs then "." else ""
  where xs = drop 100 $ takeWhile inside $ mandel c
        inside z = abs2 z <= 4 -- divergent, wenn außerhalb

main = putStr $ concatMap row [0..24]
  where row r = concatMap (col r) [0..78] ++ "\n"
        col r c = dot $ Complex (fromIntegral c / 30 - 2.1)
                      (fromIntegral r / 12 - 1)
```

Mathematische Probleme lassen sich in Haskell besonders elegant formulieren, zum Beispiel die Darstellung der komplexen Mandelbrot-Menge „Apfelmännchen“ als ASCII-Art.