Gehasste und geliebte Sprachen

In der Vorlesung wurde ich gestern nach Programmiersprachen gefragt: welche werden gehasst und welche werden in Freier Software geliebt? Und wie gerechtfertigt ist das? Knappe Kommentare, zugespitzt.

Spoiler: die Realität ist weit differenzierter als verbreitete Vorurteile. Lasst euch nicht durch eure Vorurteile blockieren. Auch nicht durch meine.

Etwas überarbeitete Fassung der aus dem Ärmel geschüttelten Einschätzung in der Vorlesung. Immernoch aus dem Ärmel geschüttelt, aber mit ein paar Links.

Mehr Sprachen und Sichtweisen: TIOBE Index und StackOverflow Developer Survey.

Disclaimer: das hier ist meine Meinung. Andere sehen das anders. Vielleicht zurecht. Ich habe das mal anders gesehen. Und werde es in Zukunft anders sehen. Niemand weiß wirklich, was gut ist. Ergänzungen gerne über Mastodon.

Inhaltsverzeichnis

1 Gehasste Sprachen

1.1 Fortran

Alte Leute mussten alle damit arbeiten und hassen es. Im Studium fanden wir es alle schrecklich und ich habe in der Doktorarbeit zwei Jahre gebraucht, um die Vorurteile zu überwinden.

Es ist alt und hat einiges an Deklarationsoverhead, schreibt sich für Mathematik aber sogar angenehmer als Python und ist fast 100x schneller. Ich habe einen kleinen Einstieg geschrieben.

Fortran 90 ist allerdings auch eine viel elegantere Sprache als Fortran 77 (das Die Alten™ lernten).

Und Fortran-Code von vor 40 Jahren läuft heute noch.

Wenn eine Sprache nicht gehasst wird, war sie nie beliebt …

1.2 COBOL

Alte Leute in Finanzen und Verwaltung mussten damit arbeiten. Elon Musk findet, es sollte über KI-Übersetzung durch Java ersetzt werden.

Die proprietären Compiler sind teuer und gnuCOBOL ist noch langsam, über die EU wird aber Arbeit an freien COBOL-Werkzeugen unterstützt.

Ich erwarte ein Comeback.

1.3 Java

Lange Zeit die am weitesten verbreitete Programmiersprache, mit viel Overhead und harter Abstraktion, gleichzeitig aber Reflection, die sie bricht. Es braucht eher mehr RAM, floating Point Mathematik ist suboptimal und es ist für low-level Arbeit unpraktisch (keine unsigned bytes …).

Langatmig: Braucht viel Code für wenig Logik.

Es gibt Bibliotheken für alles. Oft schlecht dokumentierte Bibliotheken. Industriestandard.

Java ist in den letzten Jahren aber eleganter geworden (z.B. durch Lambdas und Records und `var`) und die JVM ist flexibel und schnell.

1.4 Javascript

Für Jahrzehnte funktionierte nur Javascript für Web-Frontends ohne Neuladen. Die Sprache ist inkonsistent und gefährlich und falsy und truthy und das scoping in geschachtelten lambdas haben schon einige fiese Bugs produziert. Braucht durch Geschwindigkeitsoptimierungen (Beispiel: Multi-Tier-JIT) vergleichsweise viel Speicher.

Aber Javascript von vor 10 Jahren läuft heute noch.

Und es schafft gleichzeitig eine minimale Startverzögerung von Quellcode aus.

Dass Javascript schnell wurde (Node.js im Benchmarksgame) beweist, dass jede Sprache bis auf Faktor 3 an C herankommen kann, wenn nur genug Geld und Fachkenntnis auf die Optimierung geworfen wird.

1.5 PHP

Kann alle Paradigmen und fällt dir sicher auf die Füße. Wurde viel von Leuten genutzt, bevor sie richtig Programmieren konnten. War früher sehr langsam, aber Facebooks Hack-Versuch hat bewiesen, dass auch PHP schnell sein kann und PHP 7+ ist schneller als Python. Der einfachste Schritt von statischen zu dynamischen Seiten.

PHP war auf jedem Server verfügbar, weil seine Sandbox den Betrieb dort einfach und (für den Serverbetreiber) verlässlich machte.

PHP läuft immernoch hinter vielen großen Webseiten.

1.6 Prolog

Wird an Unis gelehrt, um Logikprogrammierung zu lernen.

Es war vor dem zweiten KI Winter 1987 die Europäische Antwort auf Lisp. Sega hat damit 1986 einen KI-Computer veröffentlicht. MAME kann ihn seit 2024 emulieren.

1.7 Lisp/Scheme

Lots of Irritating Superfluous Parentheses. Nach dem KI-Winter wurde es durch andere Sprachen verdrängt, außer ihr nutzt Emacs (wie ich). Seine Features tauchen immer wieder auf.

Mehr dazu bei den geliebten Sprachen.

1.8 XML und XSLT

Komplex, schmerzhaft durch nur halb funktionierendes XSLT, das versucht zu liefern was Lisp's match kann, aber immer nur auf eine von fünf dokumentierten Arten funktioinert. Und XML zu Parsen braucht signifikant Rechenleistung.

Nimm Lisp und verzehnfache den Syntax-Overhead. Dann benutz es als Programmcode für Remote Procedure Calls (RPC). Um ein längeres Hello World zu schreiben, braucht ihr Shakespeare Lang.

Aber es gibt in allen Sprachen Parser dafür, es kann mit Schemata validiert werden, und es ist streng definitert.

1.9 YAML

Menschenlesbares XML trifft Pythonisiertes JSON mit mehr Fallstricken und weniger Bibliotheken - und noch mehr Falschverwendung. Liest sich fast wie Markdown, aber mit Einrückung.

Ich habe 2008 aufgehört es zu nutzen, weil unsortierte Hashmaps in einem menschenlesbaren Format blöd sind: der Roundtrip Programm-zu-Programm funktionierte gut, aber Serialisiert-zu-Serialisiert funktionierte nur mit zusätzlichen Listen, weil Hashmaps ihre Schlüssel umsortieren können.

Zehn Jahre später kam es über Docker u.ä. zurück. Hashmaps sind jetzt in den meisten Sprachen nach insert-Reihenfolge sortiert (mein Problem ist also gelöst) und es ist sicherer für Zahlen als JSON (außer von Javascript aus, das castet Nummer-Strings zu Zahlen . . .), aber Leute haben angefangen, statt Makefiles Pseudo-Shell-Skripte als YAML-Listen zu schreiben, die alle Fallstricke von Shell-Skripten mit Sonderregeln in YAML kombinieren.

YAML kann viel und hat viele Sonderregeln für Menschenlesbarkeit, die bei Serialisierung stören können.

Version 1.2 von 2009 löst viele der Probleme, aber das verwendet fast niemand.

Inzwischen gibt es in vielen Sprachen langsame, inkompatible Parser dafür, aber ich lese YAML gerne. Der Lynchmob ruft schon

2 Geliebte Sprachen

2.1 Rust

Von Mozilla gefördert, hat in Firefox große Fortschritte ermöglicht, wird von allen geliebt und produziert 500 MiB große Projekte mit massiven Compilezeiten aus ein paar dutzend Seiten Code.

Wird für mich erst interessant, wenn mit GCCrs eine zweite Implementierung verfügbar ist, deren Hardware-Support nicht durch das von Apple kontrollierte LLVM begrenzt ist

2.2 Go

Einfach aufzusetzen, baut kleine Binaries, einfache Parallelität, einfach zu lernen durch absichtliche Begrenzung, begrenzt durch absichtliche Begrenzung (Leute beschweren sich, dass sie gegen Wände laufen).

2.3 Lua

Kann leicht als Skriptsprache in Spiele eingebunden werden, kann mit LuaJIT schnell sein. Ich habe nie mehr als ein Hello World gebaut.

2.4 C

Elegant in der Nähe zur Hardware. Leicht für Leute mit viel Hintergrundwissen. Pointer sind gefährlich. Doppelpointer sind doppelt gefährlich.

Großes Risiko von Speicherfehlern, durch Tooling aber teilweise erschlagen.

2.5 C++

Für Leute, die glauben, dass sie programmieren können. Wahnsinnige Komplexität mit mehr Fußangeln als Javascript, aber für maximale Leistung von komplexen Systemen nützlich.

Fügt Klassen zu C hinzu. Und ein Turing-Vollständiges Template-System, mit dem Laufzeiteinsparungen durch Compilezeitkosten erkauft werden können.

Smart Pointer sind sicherer als Pointer, müssen dafür aber genutzt werden.

Nicht ganz so großes Risiko von Speicherfehlern wie C, durch Tooling teilweise erschlagen.

2.6 Typescript

Typen in Javascript: Die Fußangeln von Javascript gepaart mit der Langatmigkeit von Java, aber etwas leichteres Refaktorieren durch Typ-Prüfungen. Deren Korrektheit aber nicht für Laufzeit gilt ⇒ es ist ein teurer Linter mit viel Overhead, findet aber echte Fehler.

Löst keine Kommunikationsprobleme. Kontrolliert von Microsoft.

2.7 Python

Leicht zu lernen, Sehr gute Dokumentation, sehr nette Community, langsam, braucht in großen Projekten viele Hacks für gute Startzeit.

Früher: „schreib nur Performance-Kritisches in C“. Heute: „Schreib ein Python-Frontend für ein riesiges C++-KI-Modell“. Früher genau eine verständliche Syntax. Heute stehlen comprehensions den eingerückten for-loops die Show.

Es gibt Bibliotheken für alles. Gut dokumentierte. Darunter scipy und matplotlib.

Unter der Haube von scipy läuft viel Fortran. Teils sehr, sehr alt. Und schnell.

2.8 Lisp/Scheme

Lots of Irritating Superfluous Parentheses. Nach dem KI-Winter wurde es durch andere Sprachen verdrängt, aber seine Features tauchen immer wieder auf.

(ja, Lisp wird geliebt und gehasst)

Für die Leute, die die Grenzen von Softwareentwicklung austesten und ihr Hobby sicher nicht im Job verwenden wollen.

(vielleicht ändert sich das noch)

Es gibt ein Scheme für jede Domäne – von Embedded Device bis Webservice.

Um der Lisp Curse zu entgehen, beachtet den Lisp Curse Redemption Ark und den Zen for Scheme. Um die Klammern zu verringern, nutzt Wisp (Scheme) oder Whisper (Common Lisp). Dann gibt es noch weniger Sprachkundige.

Ich verwende in meinen Hobby-Projekten Guile Scheme (⇒ py2guile und Programming Essentials with Scheme und with Wisp). Das hier ist also mit Vorsicht zu genießen.

3 Gästebuch :-)

Händisches Gästebuch: schreib einfach über Mastodon, dann füge ich es ein.

3.1 Ansible

Siehe YAML

3.2 Forth

Mix aus interpretierter und kompilierter Sprache mit einer extrem kleinen und schnellen Laufzeitumgebung.

Auf PalmOS die Sprache, um Apps on-device zu entwickeln (das ging).

1982 kam der Jupiter Ace mit Forth im ROM heraus, während alle anderen Heimcomputer noch BASIC im ROM hatten. Deshalb wurde er kein Erfolg, aber ein spannendes Experiment.
— deBaer

3.3 JSON

Javascript Object Notation. Hat keine Kommentare, deswegen ist es anders als m3u-Listen kompatibel geblieben. Und deswegen gibt es X „JSON mit Kommentaren"-Projekte.

Und JSON will bite us badly.

Danke für den Hinweis an knoppi!

3.4 AWK

AWK is kind of perfect for what it does (data-driven stuff and combining it with other specialized command line tools) and it doesn't pretend to be the one language for everything.
– eruwero