

Fortgeschrittene Programmierung



Kapitel 1: Python Operatoren und Datentypen

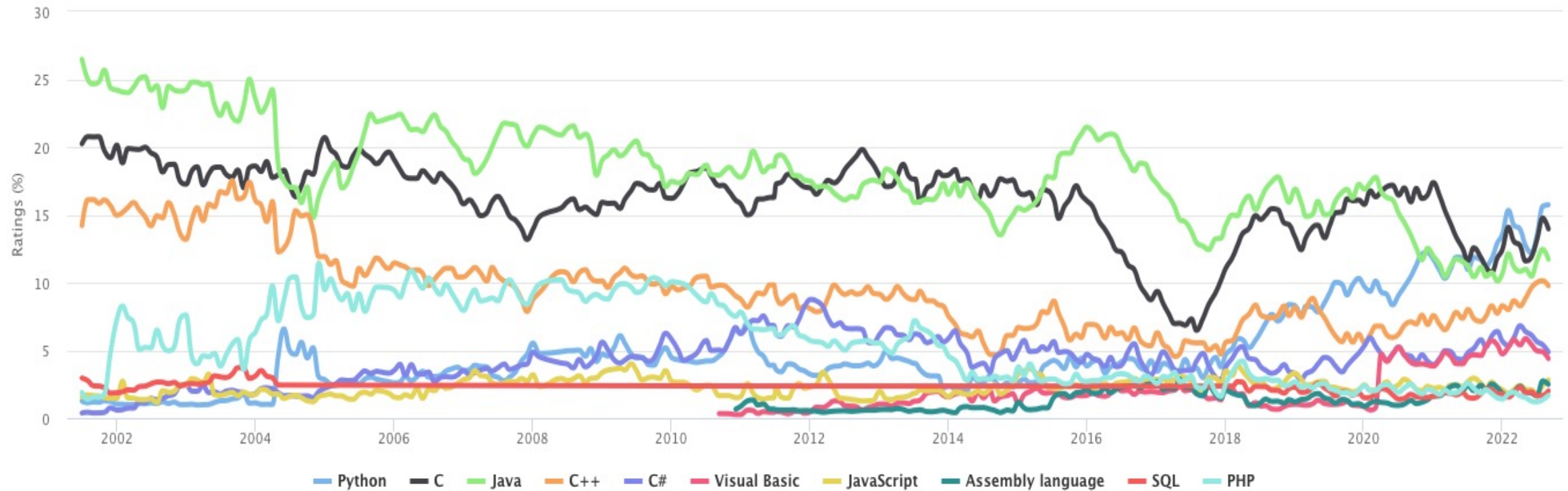
Prof. Dr. Thomas Wieland
WS 2022/2023



Verbreitung von Python

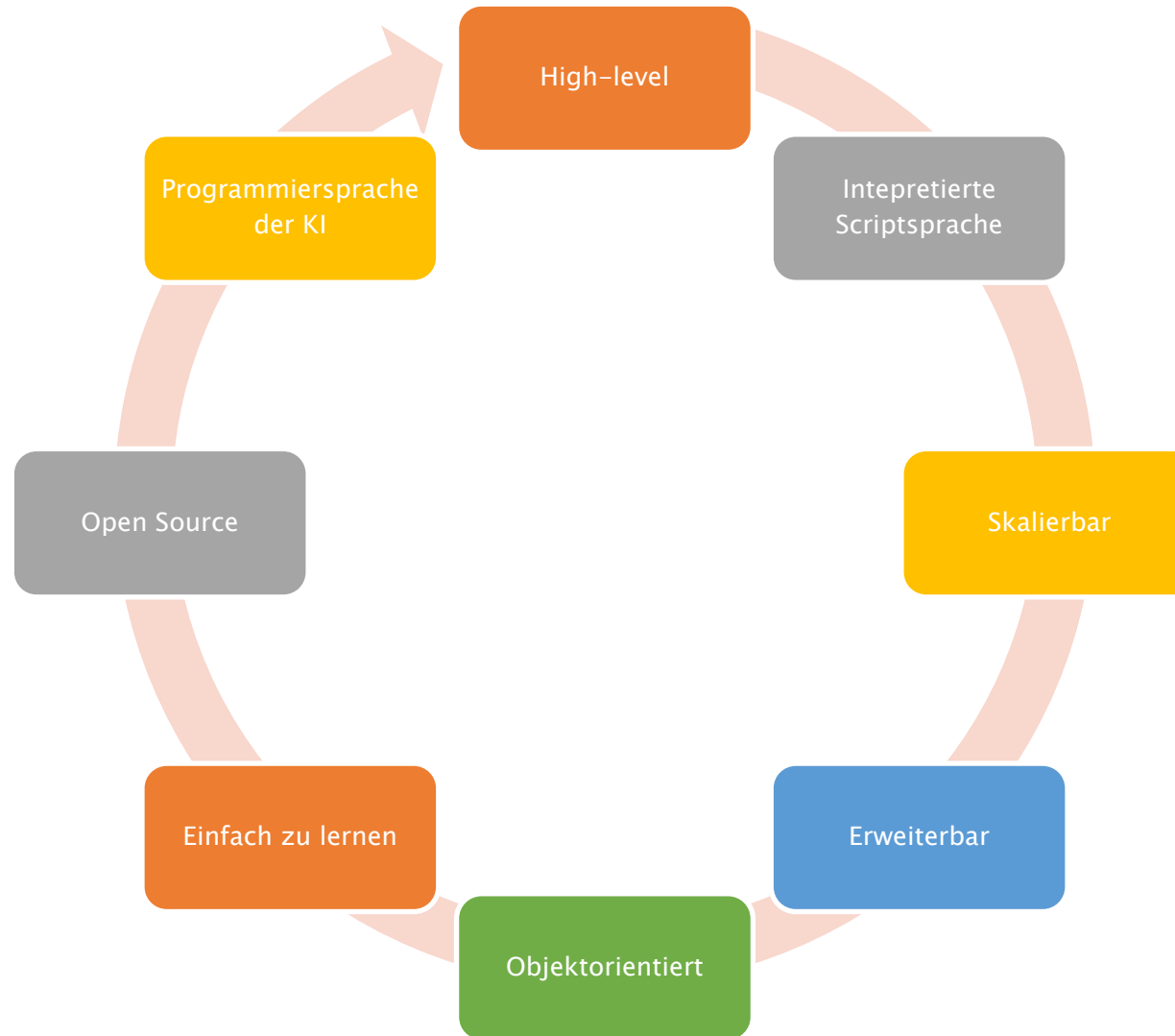
TIOBE Programming Community Index

Source: www.tiobe.com



Quelle: www.tiobe.com (Sep. 2022)

Warum Python?



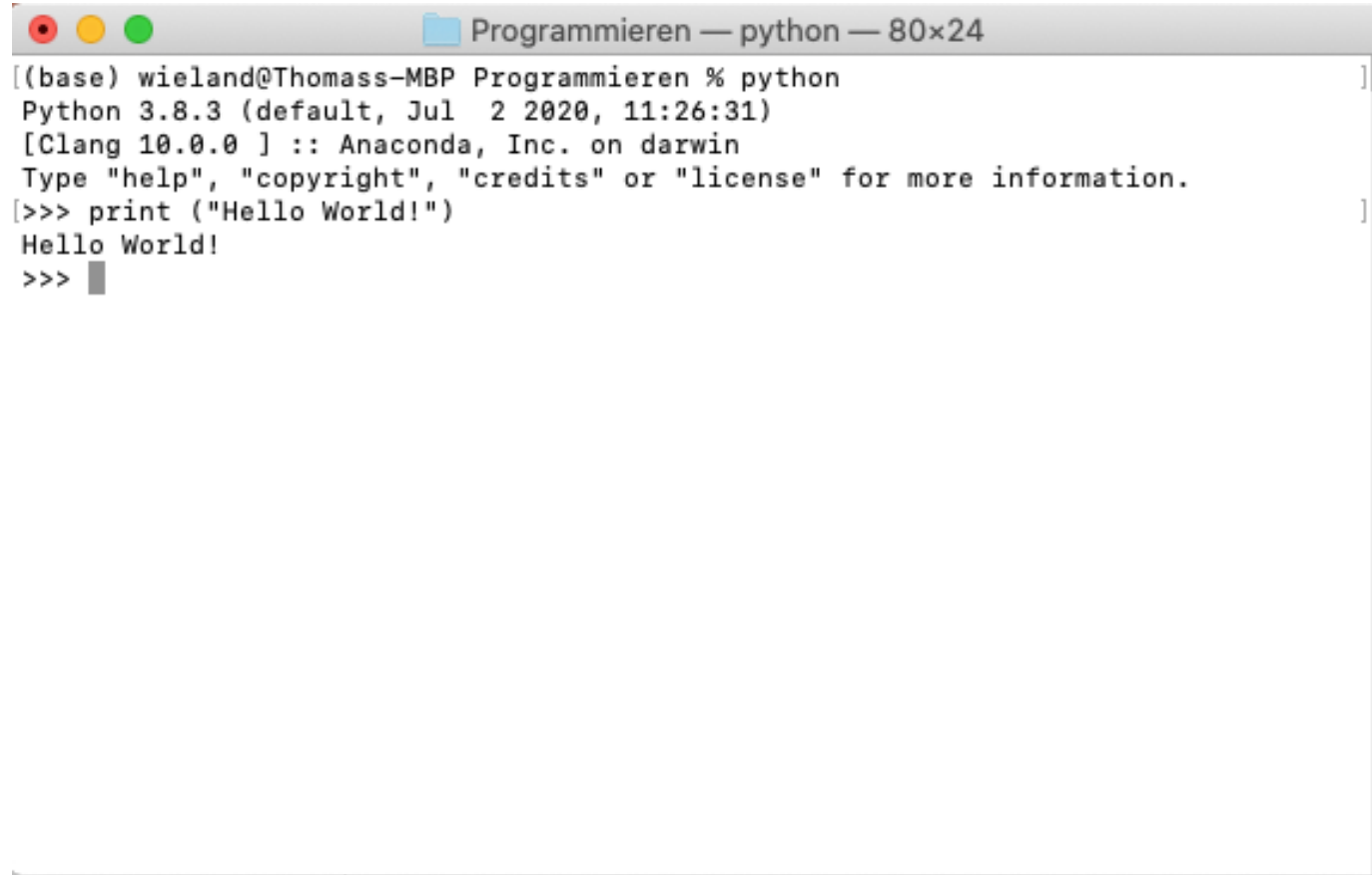
Ursprünge

About the origin of Python, Van Rossum wrote in 1996: Over six years ago, in December 1989, I was looking for a “hobby” programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python’s Flying Circus).

Quelle: [Wikipedia](#)

Erster Start

- Kommandozeile starten, „python“ eingeben



```
[(base) wieland@Thomass-MBP Programmieren % python  
Python 3.8.3 (default, Jul 2 2020, 11:26:31)  
[Clang 10.0.0 ] :: Anaconda, Inc. on darwin  
Type "help", "copyright", "credits" or "license" for more information.  
[>>> print ("Hello World!")  
Hello World!  
>>> ]
```

iPython

- Erweiterte interaktive Shell
- Zusätzliche Shell-Syntax
 - Zugriff auf Bash-Befehle
 - Ausführen von Python-Skripten
 - Übersichtlichere Ausgabe
 - Logging
 - Suchfunktionen
- Tab-Vervollständigung
- Liste bisheriger Befehle
- Automatische Einrückung
- Default-Konsole in Spyder



Quelle: needpix.com

Primitive Datentypen

- Boolesche Werte
 - `True` und `False` (auf 1 und 0 abgebildet)
- Numerische Typen
 - `int`: Ganzzahlige Werte (Größe nur durch Speicher beschränkt), auch als Hex möglich
 - `float`: doppelte Genauigkeit (ähnlich `double` in C)
 - `complex`: Komplexe Zahlen mit Imaginärteil, z.B. `z = 1.2 + 3.4j`. Mit `z.real` und `z.imag` kann man auf die Anteile zugreifen, `z.conjugate()` berechnet die konjugierte Zahl
- **Alles in Python ist ein Objekt!**
 - Daher bei allen Datentypen: Zugriff auf Attribute und Methoden
- **Dynamische Typisierung**: Typ von Variablen durch Zuweisung bestimmt
 - Automatische Konvertierung wo möglich und nötig

Operatoren

- + Addition, - Subtraktion, * Multiplikation, / Division
- // Division, die zum nächsten ganzzahligen Wert abrundet
- ** Potenz
- % Modulo
- Kombinationen: +=, -=, *=, /=, //=
- Relationale Operatoren: <, <=, ==, !=, >, >=
- Bitoperatoren: >>, <<, & (bitweises AND), | (bitweises OR), ^ (bitweises XOR), ~ (Komplement)

Sequentielle Datentypen

- Zeichenketten, Tupel und Listen
 - Zeichenketten: Folgen von Zeichen, nicht änderbar, einfache oder doppelte Anführungszeichen
 - Tupel: Folgen von Objekten, nicht änderbar
 - Listen: Folgen von Objekten, änderbar

```
>> EineZeichenkette = "abcdefäöüß"  
>> EinTupel = (1, 'a', 0.42)  
>> EineListe = [EineZeichenkette, EinTupel, 4711]  
>> EineListe  
['abcdefäöüß', (1, 'a', 0.42), 4711]
```

- Zuweisung zu Listen
nur als flache Kopie!

```
>> li = [1, 2, 3]  
>> cli = li  
>> li[2] = 'a'  
>> print("li = ", li, ", cli =", cli)  
li= [1, 2, 'a'] cli = [1, 2, 'a']
```

Bereiche

- `range(i, j)` liefert ein iterierbares Objekt mit den Zahlen von `i` bis `j-1`
- `range(n)` entspricht `range(0,n)`
- Auch Schrittweite ist möglich

```
:li =[j for j in range(10)]
```

```
:li
```

```
Out[139]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
:li[3], li[2:4], li[-2], li[:3], li[2:7:2]
```

```
Out[140]: (3, [2, 3], 8, [0, 1, 2], [2, 4, 6])
```

- Negativer Index zählt von hinten, `len(s)` ist die Länge der Sequenz

Operationen auf Listen

- `li.append(x)` # fügt das Element x an
- `li.extend(l)` # hängt Liste l an Liste li an
- `li.count(x)` # zählt die Häufigkeit von x in li
- `li.index(x)` # gibt den kleinsten Index i mit `li[i] == x`
- `li.insert(i, x)` # fügt x in li an der Stelle i ein
- `li.pop(i)` # gibt das Element `li[i]` zurück und entfernt es
- `li.remove(x)` # entfernt das erste Vorkommen von x
- `li.reverse()` # dreht die Liste um
- `li.sort()` # sortiert die Liste aufsteigend

Ausgewählte String-Methoden

- `s = "abra cada bra"`
- `s.islower()` # ist alles klein geschrieben
- `s.startswith("abra")` # überprüft den Anfang
- `s.endswith("bra")` # analog
- `s.find("cada")` # findet das erste Vorkommen von „cada“
- `s.rfind("cada")` # findet das letzte Vorkommen
- `s.isalnum()` # testet, ob nur Zeichen drin sind
- `s = s.strip()` # entfernt Leerzeichen vorn/hinten
- `s.lower()` # wandelt in Kleinbuchstaben
- `s.split(z)` # zerlegt in Teilzeichenketten mit z als Trenner
- `s.count("a")` # zählt die Vorkommen von „a“ in s
- `s.splitlines()` # zerlegt s in eine Liste von Zeilen
- `s.format(Z)` # formatiert die Zeichenkette gemäß Z

Formatierung von Strings

- Formatiert wird der Inhalt des Formatstrings gemäß der Angaben in der Zeichenkette
`s = "Winter{}Semester{}2022"`
`s.format("-", " ab 10/")`
- Platzhalter können auch nummeriert sein:
`t = "A {0} with a {1} is still a {0}."`
`t.format("fool", "tool")`
- Damit sind auch Formatierungen für Zahlen möglich:
`z = "{0:d}, {0:f} oder {0:E}"`
`z.format(1234)`
- Konvertierungszeichen wie in C
 - „%d“ für Dezimalewerte, „%f“ für Gleitkomma, „%e“ in Exponentialdarstellung, „%s“ für Zeichenketten, „%c“ ein einzelnes Zeichen, „%o“ oktal, „%x“ hexadezimal
 - Auch Längenangaben und Genauigkeiten sind möglich

Tupel und Listen

- Folgen beliebiger Objekte
- Tupel sind nicht änderbar
 - Können aber änderbare Elemente enthalten, z.B. Mengen oder Listen
- Zugriff und Bearbeitung wie bei Zeichenketten
- Tupel nutzt runde Klammern, Listen eckige
- Tupel werden intern häufig verwendet!

```
x = 'hippo', 13  
a, b = x  
b  
tuple('hippo')
```



Mengen

- Ungeordnete Sammlung eindeutiger Objekte
- Veränderbar (set) oder unveränderbar (frozenset)
- Übliche Operationen als Objektmethoden möglich
 - Mengenminus `m.difference(n)`, Schnittmenge `m.intersection(n)`, Vereinigungsmenge `m.union(n)`, Teilmenge `m.issubset(n)`, Diskunktheit `m.isdisjoint(n)`
 - Auch nur mit Operatoren: Differenz `m - n`, Vereinigung `m | n`, Schnitt `m & n`
- Weitere Methoden für Einfügen, Löschen und Verändern

Lexikon (Dictionary)

- Schlüssel-Wert-Paare
 - Schlüssel unveränderlich, können primitive Typen, Zeichenketten, Tupel oder eingefrorene Mengen sein
 - Werte können von beliebigem Typ sein
 - In geschweiften Klammern notiert, Schlüssel und Wert mit Doppelpunkt getrennt
- Verschiedene Methoden verfügbar (z.B. Zugriff auf Wert `dict[k]`, Defaultwert für nicht enthaltene Schlüssel, Rückgabe aller Schlüssel oder Werte getrennt)

Das können Sie jetzt auch!

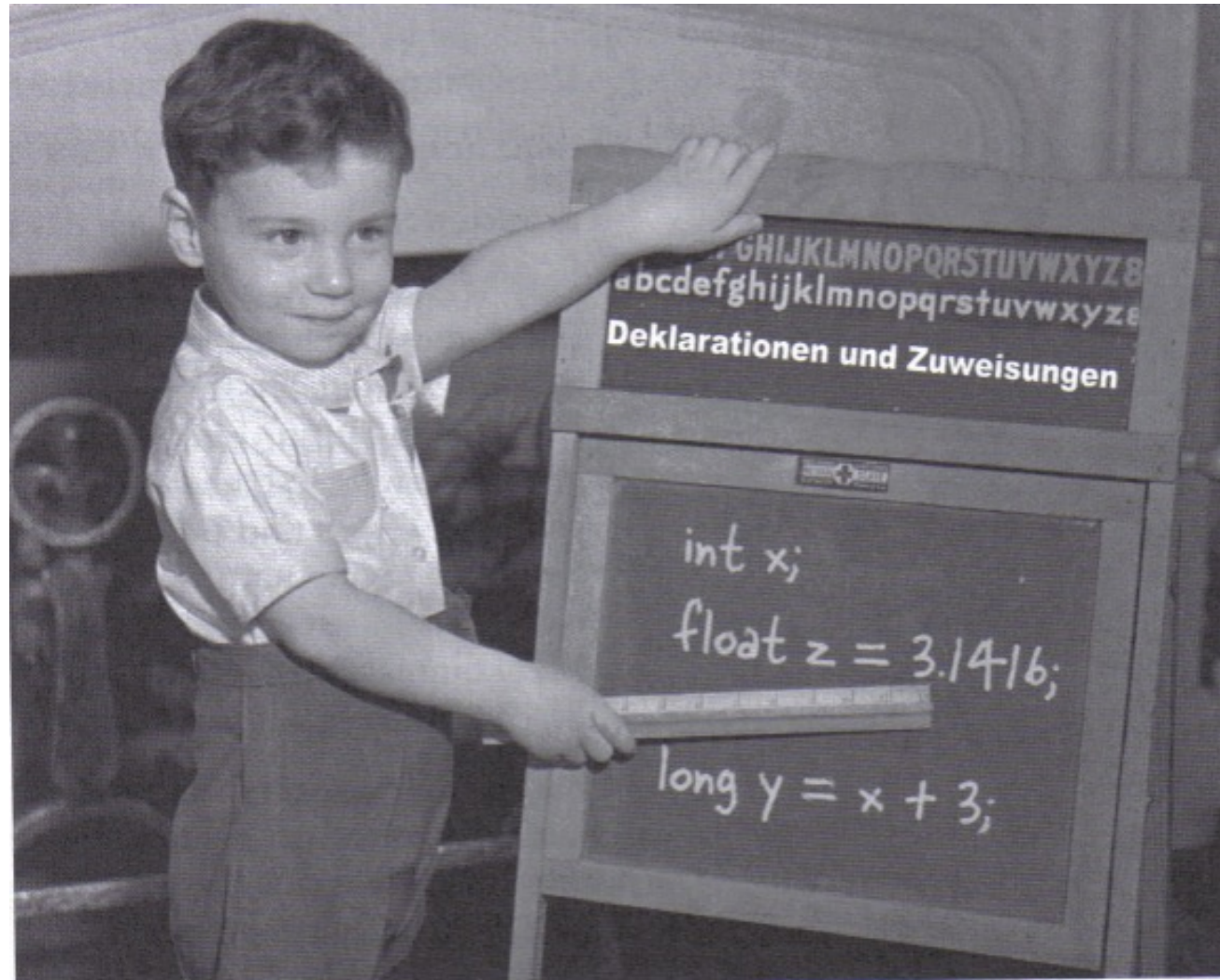


Abbildung aus: K. Sierra/B. Bates: Java von Kopf bis Fuß. O'Reilly, 2006.