

# **III. Funktionale Programmierung**

- 1. Prinzipien der funktionalen Programmierung**
- 2. Deklarationen**
- 3. Ausdrücke**
- 4. Muster (Patterns)**
- 5. Typen und Datenstrukturen**
- 6. Funktionale Programmiertechniken**

# **Kenntnis verschiedener Sprachen**

---

- Eigene Ideen bei der Software-Entwicklung können besser ausgedrückt werden
- Nötig, um in konkreten Projekten geeignete Sprache auszuwählen
- Erleichtert das Erlernen weiterer Programmiersprachen
- Nötig für den Entwurf neuer Programmiersprachen

## Imperative Sprachen

- Folge von nacheinander ausgeführten Anweisungen

## Deklarative Sprachen

- Spezifikation dessen, was berechnet werden soll
- Compiler legt fest, wie Berechnung verläuft

### ■ Prozedurale Sprachen

- Variablen, Zuweisungen, Kontrollstrukturen

### ■ Funktionale Sprachen

- keine Seiteneffekte
- Rekursion

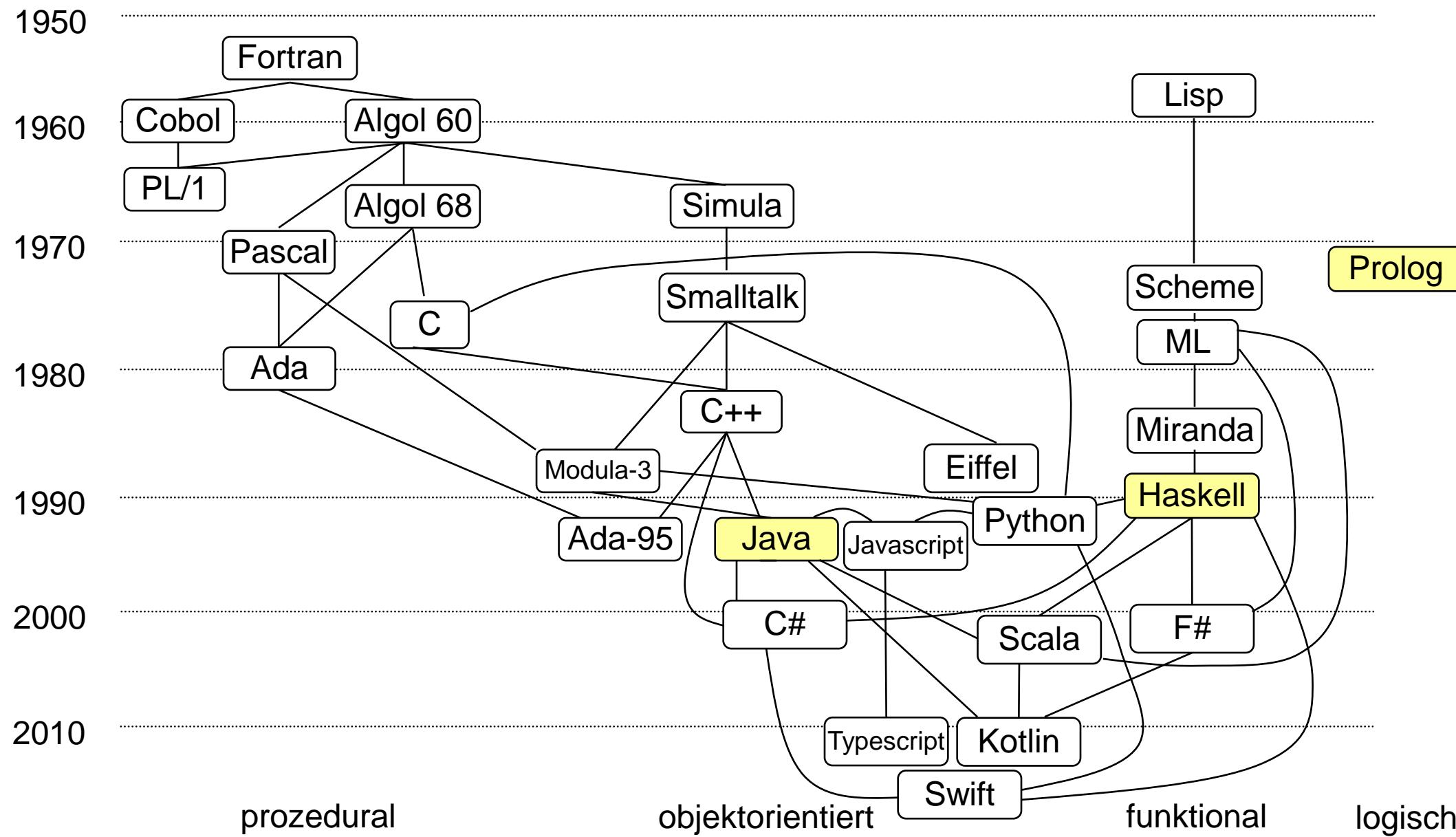
### ■ Objektorientierte Sprachen

- Objekte und Klassen
- Abstrakte Datentypen und Vererbung

### ■ Logische Sprachen

- Regeln zur Definition von Relationen

# Wichtige Programmiersprachen



# Imperative Programmierung (Java)

Eingabe: Eine Liste x

Ausgabe: Länge der Liste x

Algorithmus:

1. Setze n = 0.
2. Falls x nicht die leere Liste ist, dann:
  - 2a. Lösche das erste Element von x.
  - 2b. Erhöhe n um 1.
  - 2c. Gehe zu Schritt 2.
3. Liefere n zurück.

```
class Element {  
    Vergleichbar wert;  
    Element next; ... }  
  
public class Liste {  
    Element kopf;  
    ...  
    static int len (Liste x) {  
        int n = 0;  
        while (x.kopf != null) {  
            x.kopf = x.kopf.next;  
            n = n + 1;  
        }  
        return n;  
    }  
}
```

# Kennzeichen imperativer Programmierung

---

- Programm besteht aus einzelnen Anweisungen, die nacheinander abgearbeitet werden.
- Verschiedene Kontrollstrukturen, um Programmablauf zu steuern.
- Abarbeitung einer Anweisung ändert Werte der Variablen.
- Seiteneffekte
- Programmierer müssen Speicherorganisation bedenken

# Funktionale Programmierung (*Haskell*)

---

- A. Falls die Liste  $x$  leer ist, so ist  $\text{len}(x) = 0$ .
- B. Falls die Liste  $x$  nicht leer ist und "rest" die Liste  $x$  ohne ihr erstes Element ist, so ist  $\text{len}(x) = 1 + \text{len}(\text{rest})$ .

## Programm in *Haskell*:

```
len []          = 0
len (kopf : rest) = 1 + len rest
```

# Ausführung funktionaler Programme

`len []` = 0

`len (kopf : rest)` = 1 + `len rest`

$$\begin{aligned} & \text{len } [15, 70, 36] \\ &= 1 + \text{len } [70, 36] \\ &= 1 + 1 + \text{len } [36] \\ &= 1 + 1 + 1 + \text{len } [] \\ &= 1 + 1 + 1 + 0 \\ &= 3 \end{aligned}$$

Annotations in blue:

- $15$  is underlined and followed by  $: [70, 36]$ .
- $70$  is underlined and followed by  $: [36]$ .
- $36$  is underlined and followed by  $: []$ .

# Kennzeichen funktionaler Programme

---

- Rekursion statt Schleifen
- Keine Seiteneffekte (*referentielle Transparenz*)
- Funktionen als gleichberechtigte Datenobjekte  
(*Funktionen höherer Ordnung*)
- Verwendung von einer Funktions-Implementierung für verschiedene Typen (*Polymorphismus*)
- Programme sind kürzer, klarer, besser zu warten, zuverlässiger, schneller zu erstellen

# Funktionale Programmierung (*Haskell*)

---

## ■ Interaktiver Modus des Glasgow Haskell Compilers (GHCi)

- Eingabe: Auszuwertender Ausdruck
- Ausgabe: Ergebnis des ausgewerteten Ausdrucks

## ■ Bsp:

- Eingabe: len [15, 36, 70]
- Ausgabe: 3

## ■ Interpreter

- führen jede Zeile des Programms nacheinander aus.
- Vorteil: Keine vorherige Übersetzung nötig, gut zur Programmentwicklung geeignet.
- Nachteil: Ineffizienter als Compiler.