

Programmierung 1

Vorlesung 2

Livestream beginnt um 10:15 Uhr

Schnellkurs, Teil 2

Programmierung 1

Prozeduren

$$\textit{quadrat}(x) = x \cdot x$$

- ▶ Unter einer **Prozedur** verstehen wir eine Berechnungsvorschrift für eine Funktion.
- ▶ Prozeduren werden durch Gleichungen beschrieben.

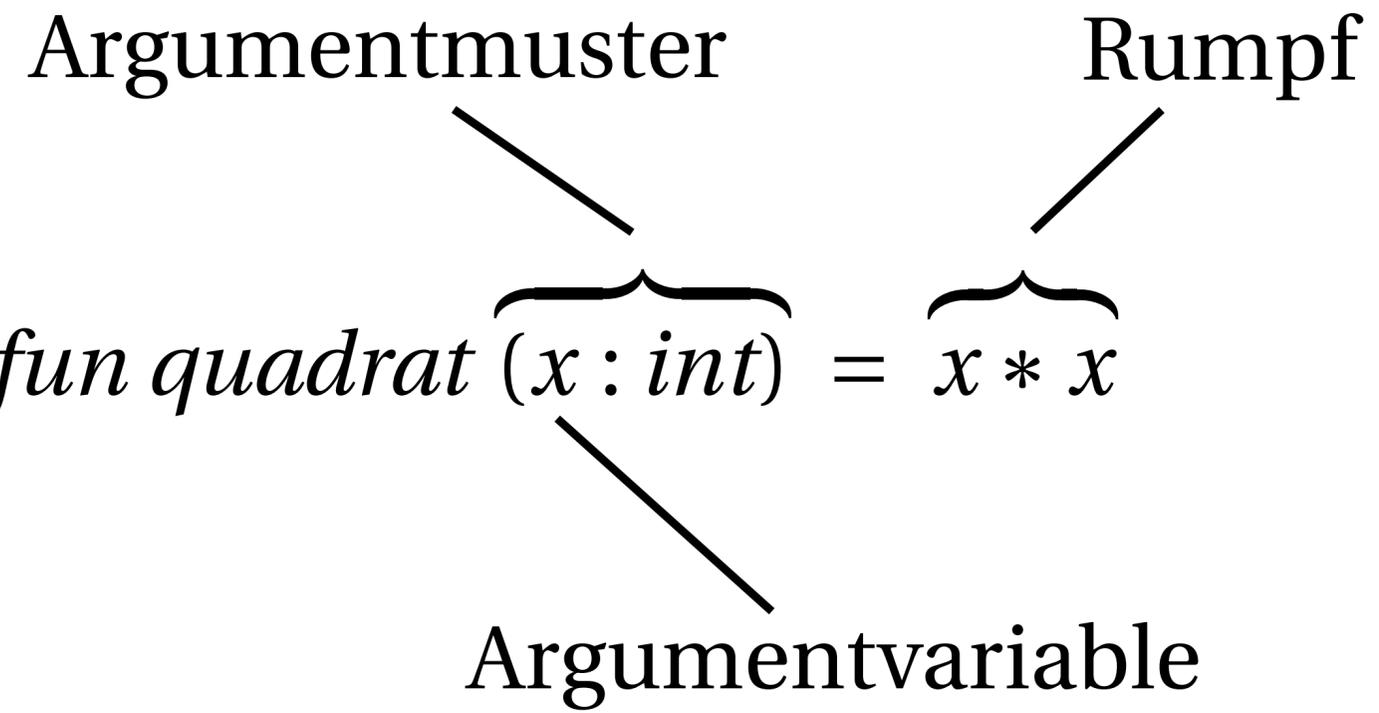
- ▶ **Prozedurdeklaration:**

```
fun quadrat (x:int) = x*x
```

- ▶ **Ein und dieselbe Funktion kann durch verschiedene Prozeduren berechnet werden!**
- ▶ Eine Berechnungsvorschrift heißt **Algorithmus**.

**Abū ‘Abdallāh
Muḥammad ibn
Mūsā al-Khwārizmī
(ca. 780 – 850)**





Prozedurtypen

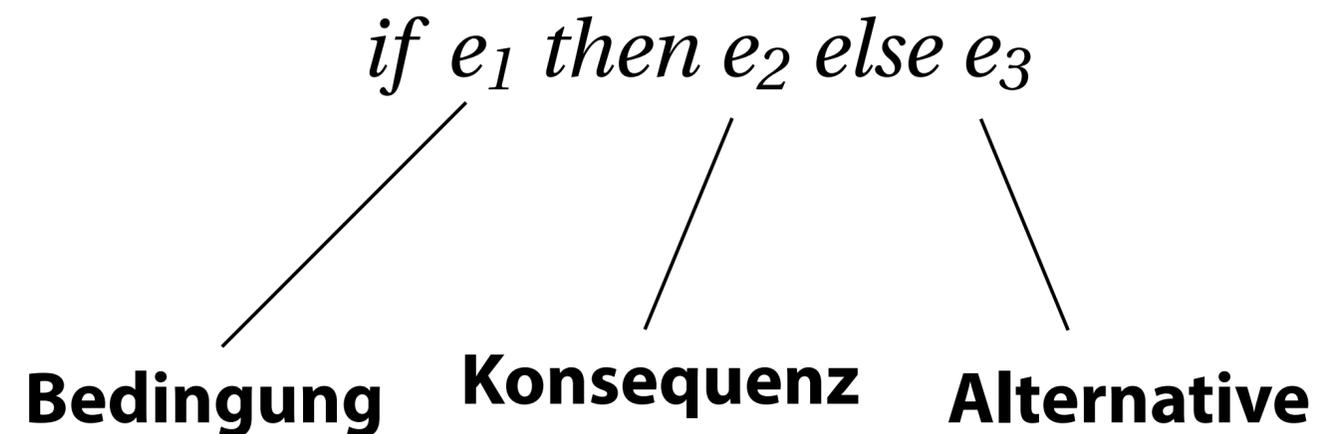
```
fun quadrat (x:int) = x*x
```

```
val quadrat : int → int
```

- ▶ ein **Prozedurtyp** $t1 \rightarrow t2$ besteht aus einem **Argumenttyp** $t1$ und einem **Ergebnistyp** $t2$.
- ▶ Eine Prozedur des Typs $t1 \rightarrow t2$ kann auf Argumente des Typs $t1$ angewendet werden und liefert Ergebnisse des Typs $t2$.

Vergleiche und Konditionale

- ▶ ein **Vergleich** ist eine Operation, die testet, ob eine Bedingung wahr ist
- ▶ ein **Konditional** ist ein Ausdruck, der eine Fallunterscheidung realisiert



$3 < 3$
false : bool

$3 \leq 3$
true : bool

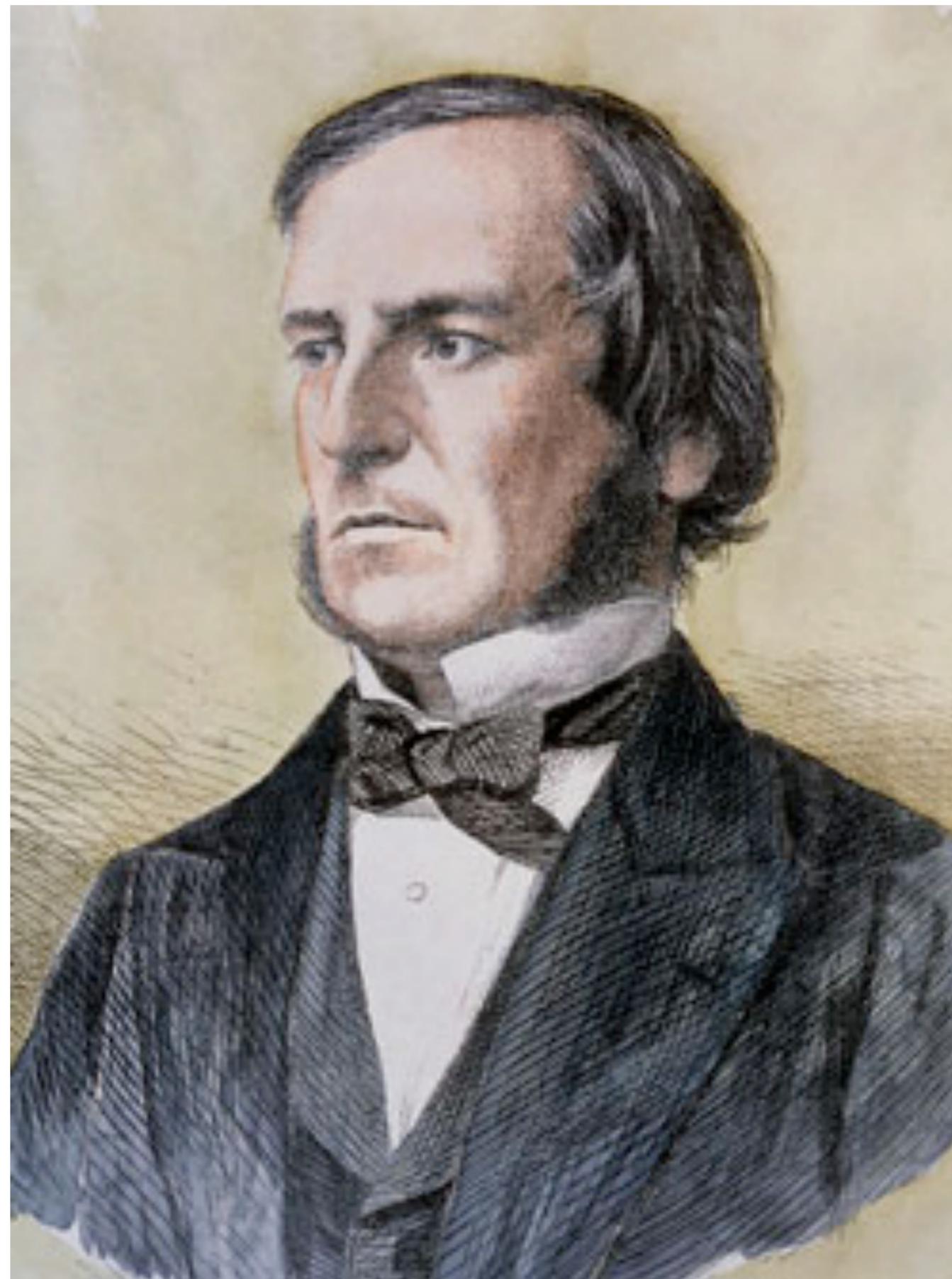
$3 \geq 3$
true : bool

$3 = 3$
true : bool

$3 < > 3$
false : bool

George Boole

(1815 – 1864)



Lokale Deklarationen, Hilfsprozeduren

```
val a = 2*2  
val b = a*a  
val c = b*b
```

► Lokale Deklarationen:

```
fun hoch8 (x:int) =  
  let  
    val a = x*x  
    val b = a*a  
  in  
    b*b  
  end
```

► Hilfsprozedur:

```
fun q (y:int) = y*y  
fun hoch8 (x:int) = q (q (q x))
```

Tupel

`(7, 2, true, 2)`

`(7, 2, true, 2) : int * int * bool * int`

- ▶ **Tupel:** Folge von Werten (v_1, \dots, v_n)
- ▶ **Positionen:** Zahlen $1, \dots, n$
- ▶ **Komponenten:** Werte v_1, \dots, v_n
- ▶ **Länge:** Anzahl der Positionen
- ▶ Der **Typ** eines Tupelausdrucks ergibt sich aus den Typen der Ausdrücke, die die Komponenten beschreiben.

Frage

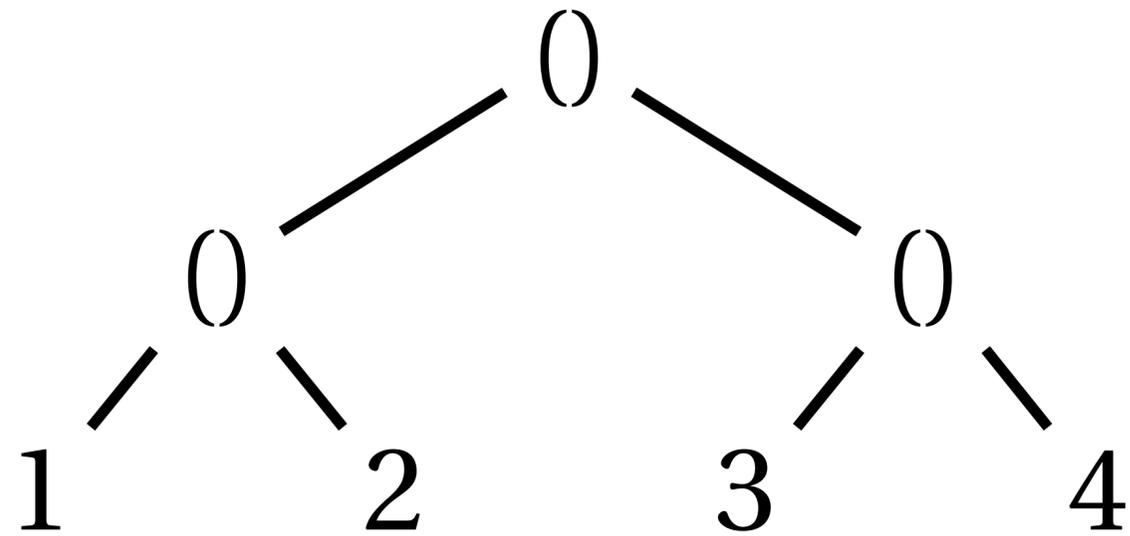
**Welche der folgenden Tupel haben
3 Positionen und 2 Komponenten?**

- ▶ $(1,2,())$
- ▶ $(1,2)$
- ▶ $((),2,())$
- ▶ $((1),2,(3))$

Geschachtelte Tupel

$((1, 2), (3, 4)) : (int * int) * (int * int)$

► **Baumdarstellung:**



Begriffe

- ▶ **n -stellige Tupel:** (v_1, \dots, v_n)
- ▶ **Paare:** $(1,2)$
- ▶ **Tripel:** $(1,(2,3),4)$
- ▶ **Leeres Tupel:** $()$: unit
- ▶ **Projektion** aus Tupeln: #

#2 $(1,2,3)$

Kartesische Muster

```
val (x,y) = (3,4)
```

```
val x = 3 : int
```

```
val y = 4 : int
```

```
val (x,y) = (y,x)
```

```
val x = 4 : int
```

```
val y = 3 : int
```

► Kartesisches Argumentmuster:

```
fun swap (x:int, y:int) = (y,x)
```

Rekursion

- ▶ **Rekursionsgleichung** für die Berechnung der Potenz x^n durch wiederholte Multiplikation:

- ▶ $x^0 = 1$

$$x^n = x \cdot x^{n-1} \quad \text{für } n > 0$$

- ▶ Die wiederholte Anwendung der zweiten Gleichung wird als **Rekursion** bezeichnet.
- ▶ Clou: Größere Potenzen werden auf kleinere zurückgeführt!

Rekursion in ML

```
fun potenz (x:int, n:int) : int =  
    if n>0 then x*potenz(x,n-1) else 1  
val potenz : int * int → int
```

- Rumpf enthält **Selbstanwendung** (auch: **rekursive Anwendung**)
- Bei der Deklaration rekursiver Funktionen geben wir den **Ergebnistyp** an.

Ausführungsprotokoll

potenz(4, 2) = *if 2 > 0 then 4 * potenz(4, 2 - 1) else 1*
= *if true then 4 * potenz(4, 2 - 1) else 1*
= 4 * *potenz(4, 2 - 1)*
= 4 * *potenz(4, 1)*
= 4 * (*if 1 > 0 then 4 * potenz(4, 1 - 1) else 1*)
= 4 * *(if true then 4 * potenz(4, 1 - 1) else 1)*
= 4 * (4 * *potenz(4, 1 - 1)*)
= 4 * (4 * *potenz(4, 0)*)
= 4 * (4 * (*if 0 > 0 then 4 * potenz(4, 0 - 1) else 1*))
= 4 * (4 * *(if false then 4 * potenz(4, 0 - 1) else 1)*)
= 4 * *(4 * 1)*
= *4 * 4*
= 16

Verkürztes Ausführungsprotokoll

```
fun potenz (x:int, n:int) : int =  
  if n>0 then x*potenz(x,n-1) else 1
```

$$\begin{aligned} \text{potenz}(4,2) &= 4 * \text{potenz}(4,1) \\ &= 4 * (4 * \text{potenz}(4,0)) \\ &= 4 * (4 * 1) \\ &= 16 \end{aligned}$$

► Rekursionsfolge:

$$\text{potenz}(4,3) \rightarrow \text{potenz}(4,2) \rightarrow \text{potenz}(4,1) \rightarrow \text{potenz}(4,0)$$

Frage

Betrachten Sie die rekursive Prozedur

```
fun r (x:int) : int = if x<0 then 0 else r (x-1) + 1
```

Was ist der Wert von $r(2)$?

- ▶ 0
- ▶ 1
- ▶ 2
- ▶ 3

Terminierung

- ▶ **Reguläre Terminierung:** Die Ausführung des Programms **endet** nach endlich vielen Schritten **erfolgreich**.
- ▶ **Abbruch wegen Laufzeitfehler:** Eine Operation signalisiert einen **Fehler**.
- ▶ **Abbruch wegen Speichererschöpfung:** Der dem Interpreter zur Verfügung stehende **Speicherplatz** ist erschöpft.
- ▶ **Abbruch durch den Benutzer:** Die noch laufende Ausführung des Programms wird durch den **Benutzer** abgebrochen.

Divergenz

```
fun p (x:int) : int = p x
```

$px = px = px = px = px = \dots$

```
fun q (x:int) : int = 0 + q x
```

$qx = 0 + (qx) = 0 + (0 + (qx)) = 0 + (0 + (0 + (qx))) = \dots$

- ▶ Eine Ausführung, die nach endlich vielen Schritten **endet**, heißt **terminierend**.
- ▶ Eine **nicht** terminierende Ausführung heißt **divergierend**.

Frage

Welche der folgenden Funktionen divergieren (für irgendein Argument)?

- ▶ `fun f (x:int) : int = if (x<0) then 0 else f(x+1)`
- ▶ `fun f (x:int) : int = if (x<0) then 0 else f(x-1)`
- ▶ `fun f (x:int) : int = if (x<0) then f(x+1) else f(x-1)`
- ▶ `fun f (x:int) : int = if (x<0) then f(x-1) else f(x+1)`

www.prog1.saarland