

FOP Recap #7



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Rekursion und Racket





Hinweise

Objektorientiert oder Funktional?

Funktionen in Racket

Funktionen in Racket

Boolsche Logik

Verzweigungen

Rekursiv vs Iterativ

Das steht heute auf dem Plan



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Hinweise

Integer-Division

Klammern

Objektorientiert oder Funktional?

Funktionen in Racket

Funktionen in Racket

Boolsche Logik



```
1 int a = 5 / 10;  
2 int b = 21 / 20;  
3 System.out.println(a);  
4 System.out.println(b);
```

Hinweise

Integer-Division



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 int a = 5 / 10;  
2 int b = 21 / 20;  
3 System.out.println(a);  
4 System.out.println(b);
```

\$ 0

\$ 1

■ Achtung! Java Integer-Division!



```
1 int a = (5 + 10) * 5;  
2 int b = 5 + 10 * 5;
```

Hinweise

Klammern



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 int a = (5 + 10) * 5;  
2 int b = 5 + 10 * 5;
```

\$ 75

\$ 55



```
1 int a = 5;  
2 int b = (a == 5)? (a + 5) : -2;
```




```
1 int a = 5;  
2 int b = (a == 5)? (a + 5) : -2;
```

\$ 5

\$ 10



- Klassen stehen im Vordergrund
 - ▣ Und hieraus abgeleitete Objekte
- Objekte
 - ▣ Haben einen momentanen Zustand
- Methoden
 - ▣ Gehören immer zu einer Klasse

Das steht heute auf dem Plan



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Hinweise

Objektorientiert oder Funktional?

Funktionen in Racket

Funktionen in Racket

Boolsche Logik

Verzweigungen



- Funktionen stehen im Vordergrund
 - ▣ Können als Daten weitergegeben werden
 - ▣ Haben immer einen Rückgabewert
- Eine Funktion liefert mit denselben Parametern immer diesselbe Rückgabe
- Keine Variablen
- Keine statischen Typen

Achtung: Racket

Das steht heute auf dem Plan



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Hinweise

Objektorientiert oder Funktional?

Funktionen in Racket

Funktionen in Racket

Boolsche Logik

Verzweigungen

Funktionen in Racket

Deklaration und Aufruf



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 (define (my-function-name parameter second-parameter)
2     (+ parameter 1)
3 )
```

```
1 (my-function-name 25 #true)
```

```
1 (+ 25 23)
```



```
1 ;; Type: number ANY -> number
2 ;; Returns: The given number plus one
3 (define (my-function-name parameter second-parameter)
4     (+ parameter 1)
5 )
```




```
1 (define my-constant-name 28.25)
```

```
1 ;; Type: number ANY -> number  
2 ;; Returns: The given number plus 28.25  
3 (define (my-function-name parameter second-parameter)  
4     (+ parameter my-constant-name)  
5 )
```

Das steht heute auf dem Plan



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Hinweise

Objektorientiert oder Funktional?

Funktionen in Racket

Funktionen in Racket

Boolsche Logik

Verzweigungen



```
1  (+ 1 2)
2  (- 5 3)
3  (* 2 15)
4  (/ 18 5)
5  (modulo 9 4)
```



■ Prefixnotation

- Operator vor Operand:
- **Operator** Operand1 Operand2 OperandN
- + 1 2 3 4 5

■ Infixnotation

- Operator zwischen Operanden:
- Operand1 **Operator** Operand2
- 1 + 2 + 3 + 4 + 5
- (((1 + 2) + 3) + 4) + 5



- Können alles sein
 - ▣ Ganze Zahlen
 - ▣ Rationale Zahlen
 - ▣ Komplexe Zahlen
 - ▣ Nichtexakte Zahlen

Das steht heute auf dem Plan



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Hinweise

Objektorientiert oder Funktional?

Funktionen in Racket

Funktionen in Racket

Boolsche Logik

Verzweigungen



- true
 - Auch #true oder #t
- false
 - Auch #false oder #f
- Und:
 - and
- Oder:
 - or
- Arithmetische Vergleichsoperatoren:
 - >, >=, <, <=, =

Das steht heute auf dem Plan



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Hinweise

Objektorientiert oder Funktional?

Funktionen in Racket

Funktionen in Racket

Boolsche Logik

Verzweigungen

Verzweigungen

mit if



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 (if condition expression-true expression-false)
```

```
1 (if (= number 5) -2 8)
```

```
1 (if (= number 5) (function-one #true) (function-two #false 5))
```

```
1 (if (= number 5)
2     (function-one #true)
3     (function-two #false 5)
4 )
```

Verzweigungen

mit cond



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 (cond
2   [condition-one expression-one]
3   [condition-two expression-two]
4   [...])
5   [else expression-else]
6 )
```

```
1 (cond
2   [(= number 5) #true]
3   [(= number 2) 9]
4   [...])
5   [else #false]
6 )
```

Das steht heute auf dem Plan



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Hinweise

Objektorientiert oder Funktional?

Funktionen in Racket

Funktionen in Racket

Boolsche Logik

Verzweigungen



```
1  int[] result = new int[5];  
2  
3  for(int i = 0; i < 5; i++) {  
4      result[i] = i;  
5  }
```

```
1 public void recursiveStep(int[] result, int currentIndex) {  
2     // Rekursionsanker  
3     if(currentIndex == result.length) {  
4         return;  
5     }  
6     result[currentIndex] = currentIndex;  
7     // Rekursiver Aufruf  
8     recursiveStep(array, currentIndex + 1);  
9 }
```

```
1 // Rekursions Start  
2 int[] result = new int[5];  
3 recursiveStep(result, 0);
```



```
1 (define (recursive-function lst)
2   (cond
3     ; Rekursionsanker
4     [(empty? lst) empty]
5     [else (cons
6             (+ 1 (first lst))
7             ; Rekursiver Aufruf
8             (recursive-function (rest lst))
9             )
10    ]
11  )
12 )
```

Typische Rekursion in Racket



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
1 (define (recursive-function lst)
2   (cond
3     [(empty? lst) empty]
4     [else (cons
5             (+ 1 (first lst))
6             (recursive-function (rest lst))
7             )
8     ]
9   )
10 )
```

```
1 ; Rekursions Start
2 (recursive-function (list 1 2 3 4 5))
```

Das steht heute auf dem Plan



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Hinweise

Objektorientiert oder Funktional?

Funktionen in Racket

Funktionen in Racket

Boolsche Logik

Verzweigungen



- Arrays haben eine feste Länge
- Man kann auf einen beliebigen Index zugreifen



- Listen haben eine variable Länge
- Man kann nur auf
 - ▢ Das erste Element mit (`first`) zugreifen
 - ▢ Den Rest mit (`rest`) zugreifen
- Man erstellt sie über (`list element1 element2 elementN`)
- Man kann über (`empty?`) prüfen, ob sie leer ist
- Eine leere Liste erstellt man über `empty`
- Mit (`append lst1 lst2`) konkateniert man zwei Listen
- Mit (`cons element1 lst`) fügt man vorne an eine Liste ein neues Element an



Live-Coding!