

# FOP Recap #6



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

## Erste Schritte mit Racket



# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

## Organisatorisches

Casting

Integer-Division

String

Objektorientierung vs. Funktionales Design

Racket



Abbildung: <https://pad.tudalgo.org/p/RX5J0xR5uv5vhP1c1gHy>

- Schreibt hier eure Themenwünsche auf
- Auch Wiederholung von bereits besprochenen Themen möglich

# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Organisatorisches

Casting

Bei primitiven Datentypen

Bei Objekttypen

Integer-Division

String

Objektorientierung vs. Funktionales Design

Racket



- Bei primitiven Datentypen
  - ▣ um einen Zahlenwert in einem anderen Typen zu speichern
  - ▣ passiert implizit oder explizit
- Bei Objekttypen
  - ▣ um den statischen Typen zu ändern
  - ▣ kann jedoch je nach dynamischen Typen fehlschlagen



- Jeder Zahltyp hat einen Bereich an Zahlen, den er repräsentieren kann
- `long`, `int`, `char`, `short`, `byte` für Ganzzahlen
- `double`, `float` für Komma-Zahlen
- Hierbei erkennt man folgende Rangordnung der Bereiche:
- `double > float > long > int > char > short > byte`

# Casting

## Bei primitiven Datentypen – Widening Casting



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

- Passiert implizit, automatisch

```
1 int a = 5;  
2 int b = 27;  
3 long c = a - b;
```



- Muss explizit angegeben werden, da Verlust von Präzision stattfinden kann!

```
1  int a = 288;  
2  char c = a; // ERROR!  
3  char c2 = (char) a; // OK!  
4  
5  char c3 = a + 5; // ERROR!  
6  char c4 = ((char) a) + 5; // ERROR!  
7  char c5 = (char) (a + 5); // OK!
```



- Funktioniert nur ohne Fehler bei passenden dynamischen Typen
- Sonst wird eine Exception geworfen

```
1  class A { .... }
2  class B extends A { .... }
3  class C extends B { .... }
4
5  A a = ....;
6  C castedC = (C) a; // Nur möglich, wenn a dynamischen Typen von C
   ↪ oder Subtypen hat
```

# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Organisatorisches

Casting

**Integer-Division**

String

Objektorientierung vs. Funktionales Design

Racket



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



```
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!

# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Organisatorisches

Casting

Integer-Division

String

    Interner Aufbau

    char

    Beispiele

Objektorientierung vs. Funktionales Design

Racket



- Im Prinzip nur ein `char [ ]`
- Jeder `String` ist unveränderbar

- Ist wie `int` ein primitiver Datentyp
- Normale Verwendung:
  - ▣ Repräsentiert (im Normalfall) genau einen Buchstaben/Zeichen
  - ▣ Jedes Zeichen hat nach Unicode einen festen Zahlenwert zugeordnet



```
1 String s = "ABC";  
2 char c0 = s.charAt(0); // == 'A'  
3 char c1 = s.charAt(1); // == 'B'  
4 char c2 = s.charAt(2); // == 'C'
```



# String

## Beispiele



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 String s = "ABC";  
2 char[] arr = s.toCharArray();  
3 char c0 = arr[0]; // == 'A'  
4 char c1 = arr[1]; // == 'B'  
5 char c2 = arr[2]; // == 'C'
```



```
1 String s = "ABC";  
2 String result = "";  
3 result += s.charAt(2);  
4 result += s.charAt(1);  
5 result += s.charAt(0);
```



```
1 String s = "ABC";  
2 String result = "";  
3 for(int i = s.length() - 1; i >= 0; i--) {  
4     result += s.charAt(i);  
5 }  
6 System.out.println(result);
```

# String

## Beispiele



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

```
1 String s = "ABC";  
2 String result = "";  
3 for(int i = s.length() - 1; i >= 0; i--) {  
4     result += s.charAt(i);  
5 }  
6 System.out.println(result);
```

\$ CBA



### ■ Für Groß und Kleinschreibung!

```
1 char c = 'a';  
2 char c2 = Character.toUpperCase(c); // == 'A'  
3 boolean check = Character.isUpperCase(c2); // == true  
4 char c3 = Character.toLowerCase(c2);
```



```
1 String message = "hallo";  
2 String upperCaseMessage = message.toUpperCase();  
3 System.out.println(upperCaseMessage);
```



```
1 String message = "hallo";  
2 String upperCaseMessage = message.toUpperCase();  
3 System.out.println(upperCaseMessage);
```

```
$ HALLO
```

# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Organisatorisches

Casting

Integer-Division

String

**Objektorientierung vs. Funktionales Design**

Racket





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



- Funktionen stehen im Vordergrund
  - ▣ Können als Daten weitergegeben werden
  - ▣ Haben immer einen Rückgabewert
- Eine Funktion liefert mit denselben Parametern immer diesselbe Rückgabe
- Es gibt keinen Zustand
- Keine Variablen
- Fast alles ist ein Ausdruck
- Keine statischen Typen



---

# Racket

# Das steht heute auf dem Plan



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Organisatorisches

Casting

Integer-Division

String

Objektorientierung vs. Funktionales Design

Racket

- Funktionen in Racket

- Operatoren und Zahlen

- 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 )
```



## ■ 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





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



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



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

# Verzweigungen

## mit if

```
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



```
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 )
```



---

# Live-Coding!