



INFORMATIK I

Tutorium 20.12.2016

BESPRECHUNG

Blatt 8



WIEDERHOLUNG

.....
Vorlesung & Für Blatt 9



WIEDERHOLUNG: HIGHER ORDER PROCEDURES

- Prozeduren höherer Ordnung (Higher-order-procedures)
- 1. Akzeptieren Prozeduren als Parameter oder/und
- 2. liefern eine Prozedur als Ergebnis
- Führen zu: kompakteren Programmen, verbesserter Les- und Wartbarkeit

WIEDERHOLUNG: FILTER

- H.O.P filter
- Akzeptiert ein Prädikat $p?$ und Liste xs . Liefert alle Elemente von xs die Prädikat $p?$ erfüllen

```
(: filter ((%a -> boolean) (list-of %a) -> (list-of %a)))  
(define filter  
  (lambda (p? xs)  
    (cond ((empty? xs) empty)  
          ((pair? xs) (if (p? (first xs))  
                          (make-pair (first xs)  
                                      (filter p? (rest xs)))  
                          (filter p? (rest xs)))))))
```

WIEDERHOLUNG: FILTER

- Anwendungsbeispiel: Gebe nur Elemente der Liste zurück, die ≥ 5 sind

```
(filter  
  (lambda (x)  
    ( $\geq$  x 5))  
  (list 1 2 3 4 5 6))
```

WIEDERHOLUNG: MAP

- H.O.P: map
- Akzeptiert Prozedur f und Liste xs und wendet f auf alle Elemente von xs an

```
; Wende f auf alle Elemente von xs an
(: map ((%a -> %b) (list %a) -> (list %b)))
(define map
  (lambda (f xs)
    (cond ((empty? xs) empty)
          ((pair? xs) (make-pair (f (first xs))
                                  (map f (rest xs)))))))
```

WIEDERHOLUNG: MAP

- Anwendungsbeispiel: Multipliziere alle Elemente der Liste xs mit 3

```
(map  
  (lambda (x)  
    (* x 3))  
  (list 1 2 3 4 5))
```


WIEDERHOLUNG: LIST FOLDING

- Idee: Ersetze die Listenkonstruktoren make-pair und empty systematisch
- Ersetze empty durch z
- Ersetze make-pair durch c

```
; Falte Liste xs bzgl. z und c
(: foldr (%b (%a %b -> %b) (list-of %a) -> %b))
(define foldr
  (lambda (z c xs)
    (cond ((empty? xs) z)
          ((pair? xs) (c (first xs) (foldr z c (rest xs)))))))
```

WIEDERHOLUNG: LIST FOLDING

- Anwendungsbeispiel: Summe einer Liste
- Nutze foldr um empty mit 0 und make-pair mit + zu ersetzen

```
(define sum  
  (lambda (xs)  
    (foldr 0 + xs)))
```

WIEDERHOLUNG: COMPOSE

- Konstruiert aus Prozedur f und g eine neue Prozedur
- Entspricht der Hintereinanderausführung (f nach g)

```
; Komponiere Prozeduren f,g  
(: compose ((%b -> %c) (%a -> %b) -> (%a -> %c)))  
(define compose  
  (lambda (f g)  
    (lambda (x)  
      (f (g x))))))
```

ÜBUNGSAUFGABE

- Gesucht ist die Summe aller Zahlen von 1 bis 100, die durch 3 teilbar sind
- Überlegungen: Implementiere Prozedur die Liste von natürlichen Zahlen von 1 bis 100 erstellt
- Wende darauf filter an

```
(: from-to (natural natural -> (list-of natural)))  
(check-expect (from-to 2 4) (list 2 3 4))  
(define from-to  
  (lambda (a b)  
    (letrec  
      ((worker  
        (lambda (a b acc)  
          (if  
            (> a b)  
            acc  
            (worker (+ a 1) b (append acc (list a)))))))  
      (worker a b empty))))
```

ÜBUNGSAUFGABE

```
; Addiert alle Zahlen von 1 bis n, die durch 3 teilbar sind
(: sum-dividable-by-3 (natural -> natural))
(check-expect (sum-dividable-by-3 6) 9)
(define sum-dividable-by-3
  (lambda (n)
    (let
      ((numbers (from-to 1 n)))
      (fold 0 +
        (filter (lambda (x)
                  (= (modulo x 3) 0)) numbers)))))
```


PROBLEM: 6

The sum of the squares of the first ten natural numbers is,

$$1^2 + 2^2 + \dots + 10^2 = 385$$

The square of the sum of the first ten natural numbers is,

$$(1 + 2 + \dots + 10)^2 = 55^2 = 3025$$

Hence the difference between the sum of the squares of the first ten natural numbers and the square of the sum is $3025 - 385 = 2640$.

Find the difference between the sum of the squares of the first one hundred natural numbers and the square of the sum.

➤ Lösung: 25164150

PROBLEM: 16

$2^{15} = 32768$ and the sum of its digits is $3 + 2 + 7 + 6 + 8 = 26$.

What is the sum of the digits of the number 2^{1000} ?

- Hint: string->number, string->strings-list, number->string
- Lösung: 1366