Check und Prepare

Lösungsvorschläge

TECHNISCHE UNIVERSITÄT DARMSTADT

Autoren: Nhan Huynh und Darya Nikitina
Fachbereich: Informatik
Übungsblatt: 05

Version: 20. November 2021
Semesterübergreifend

V1 DrRacket installieren

Installieren Sie zu Beginn DrRacket von folgendem Link auf Ihrem Rechner:

`https://racket-lang.org/download/`

In Abbildung 1 sehen Sie, wie die Oberfläche dann aussieht. Vergewissern Sie sich, dass links unten (im Screenshot gelb hinterlegt) auch *Advanced Student* steht. Andernfalls kicken Sie auf die im Bild gelb markierte Fläche und stellen es darauf um.

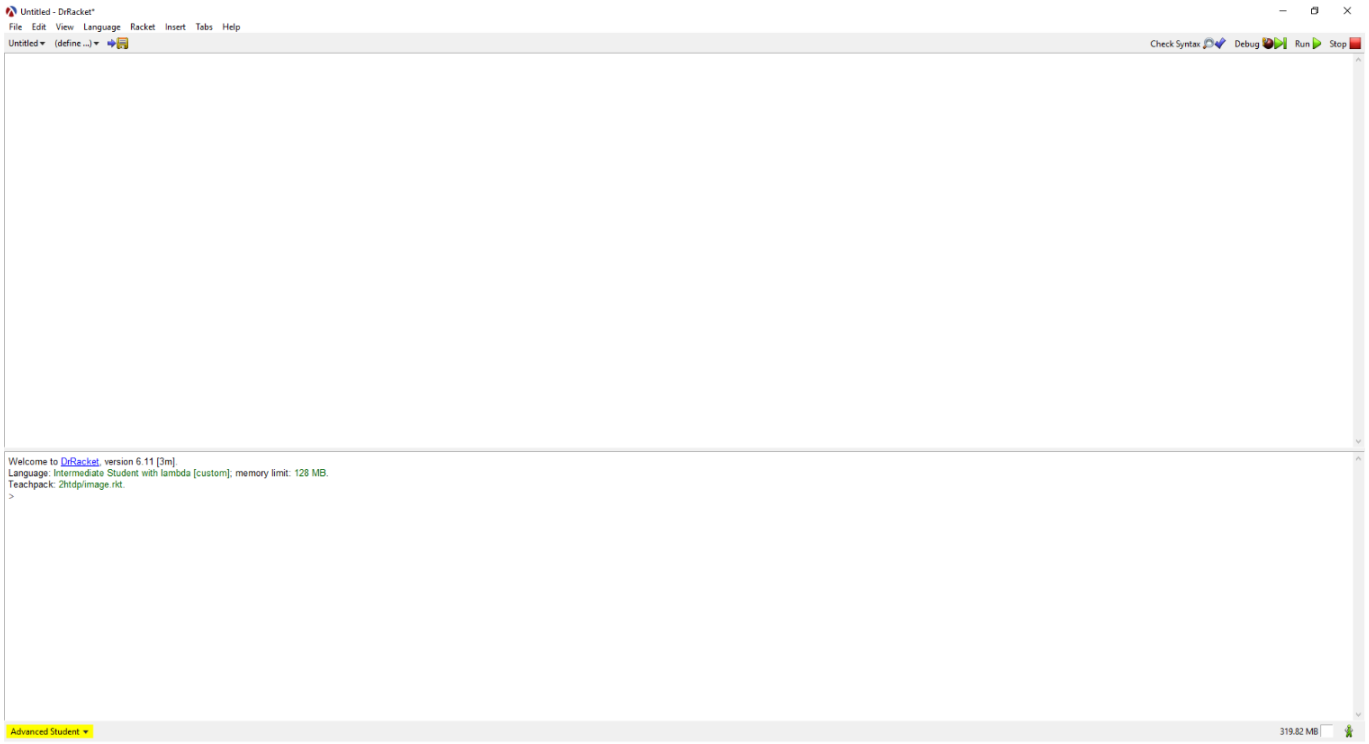


Abbildung 1: Oberfläche von DrRacket

Übungsblatt 05: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

Wichtig: Denken Sie an den Vertrag und die drei Tests bei jeder Funktion.

Achtung: Beachten Sie, dass ein Funktionsaufruf erwartet wird, falls Sie eine geöffnende Klammer schreiben. Wenn Sie das nicht beachten, erhalten Sie folgende Fehlermeldung:

function call: expected a function after the open parenthesis, but received XYZ

V2 Temperaturumrechnung ★★★

Im angloamerikanischen Maßsystem wird die Temperatur nicht wie hierzulande in Grad Celsius gemessen, sondern in Grad Fahrenheit. Da Sie damit nichts anfangen können, wollen Sie sich nun eine Funktion `fahr->cel` definieren, welche die aktuelle Temperatur in Grad Fahrenheit übergeben bekommt und in Grad Celsius umwandelt.

Hinweis: Für eine gegebene Temperatur T_F in Grad Fahrenheit berechnet sich die dazugehörige Temperatur T_C in Grad Celsius über den folgenden Zusammenhang:

$$T_C = (T_F - 32) \cdot \frac{5}{9}$$

Lösungsvorschlag:

```
;; Type: real -> real
;; Returns: the Fahrenheit value in Celsius
(define (fahr->cel degree)
  (* (- degree 32) (/ 5 9)))

;; Tests
(check-expect (fahr->cel 32) 0)
(check-within (fahr->cel 0) -17.77 0.1)
(check-within (fahr->cel 255) 123.889  0.1)
```

Übungsblatt 05: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V3 Volumen eines Tetraeders

Das Tetraeder ist ein Körper mit vier dreieckigen Seitenflächen. Sein Volumen berechnet sich über die Formel $V(a) = \frac{\sqrt{2}}{12}a^3$, wobei a hier die Länge einer Kante ist. Sie sollen in dieser Aufgabe nun eine Funktion `tetrahedron-volume` schreiben, die für eine übergebene Kantenlänge a das Volumen des dazugehörigen Tetraeders zurückgibt. Gehen Sie dazu in folgenden Schritten vor:

1. Definieren Sie eine Funktion `pow3`, welche einen Parameter `x` bekommt, und den Wert x^3 zurückgibt. Erstellen Sie außerdem eine Konstante `k`, mit dem Wert $\frac{\sqrt{2}}{12}$.
2. Nutzen Sie die zwei vorherigen Schritte, um nun die Funktion `tetrahedron-volume` zu definieren, welche nur einen Parameter `a` bekommt.

Lösungsvorschlag:

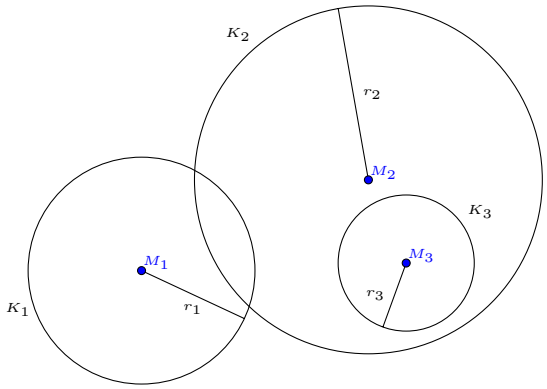
```
;; Type: real -> real
;; Returns: the third power of the given number
(define (pow3 x)
  (* x x x))

;; Tests
(check-expect (pow3 3) 27)
(check-expect (pow3 -10) -1000)
(check-expect (pow3 99) 970299)

(define k (/ (sqrt 2) 12))

;; Type: real -> real
;; Returns: the volume of a tetrahedron
(define (tetrahedron-volume a)
  (* k (pow3 a)))

;; Tests
(check-within (tetrahedron-volume 22) 1254.87 0.1)
(check-within (tetrahedron-volume 4) 7.54 0.1)
(check-within (tetrahedron-volume 54) 18557.31 0.1)
```

V4 Relative Lage zweier Kreise zueinander ★☆☆

Wir wollen die Lage zweier Kreise zueinander bestimmen. Definieren Sie dazu eine Prozedur `circles-position`, welche die Zahlen `x1, y1, r1, x2, y2, r2` in dieser Reihenfolge entgegennimmt und einen String zurückgibt, welche die Lage der beiden Kreise zueinander beschreibt. Dabei sind `x` und `y` die Koordinaten eines Kreismittelpunktes und `r` der Radius des Kreises.

Zurückgegeben werden soll `"Intersect"` bei einem Schnitt der beiden Kreise, `"External"` bei keinerlei Überlappung oder `"Interior"` wenn einer der beiden Kreise vollständig im anderen liegt. Eine Berührung der beiden Kreise, egal ob von innen oder von außen, soll als Schnitt der beiden Kreise erkannt werden.

Zum besseren Verständnis finden Sie im Anschluss an die Aufgabe ein Beispiel und die mathematische Konkretisierung des Sachverhalts (dabei steht `d` für den Abstand der Mittelpunkte). Bei den Kreisen K_1 und K_2 im Beispiel sollte `"Intersect"`, bei den Kreisen K_1 und K_3 sollte `"External"` und bei den Kreisen K_2 und K_3 sollte `"Interior"` zurückgegeben werden.



$$\text{circles-position} = \begin{cases} \text{Interior} & \text{if } d < |r_1 - r_2| \\ \text{External} & \text{if } r_1 + r_2 < d \\ \text{Intersect} & \text{otherwise} \end{cases}$$

Zur Berechnung von `d` verwenden Sie den euklidischen Abstand der Mittelpunkte der zwei zu vergleichenden Kreise. Der Abstand zweier Punkt $p_1 = (x_1, y_1)$ und $p_2 = (x_2, y_2)$ ist folgendermaßen definiert $d_{p_1,p_2} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

Übungsblatt 05: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

Lösungsvorschlag:

```
;; Type: number number number number number number -> string
;; Returns: the position of the two circles relative to each other
(define (circles-position x1 y1 r1 x2 y2 r2)
  (cond
    [(< (euclid-difference x1 y1 x2 y2) (abs (- r1 r2))) "Interior"]
    [(< (+ r1 r2) (euclid-difference x1 y1 x2 y2)) "External"]
    [else "Intersect"]))

;; Tests
(check-expect (circles-position 4 4 3 2 2 6) "Interior")
(check-expect (circles-position 1 1 3 3 2 4) "Intersect")
(check-expect (circles-position 1 1 1 5 5 1) "External")

;; Type: number number number number -> number
;; Returns: the Euclidean distance between two points
(define (euclid-difference x1 y1 x2 y2)
  (sqrt (+ (* (- x1 x2) (- x1 x2)) (* (- y1 y2) (- y1 y2)))))

;; Tests
(check-within (euclid-difference 1 2 3 4) 2.82 0.1)
(check-within (euclid-difference 3 7 12 6) 9.05 0.1)
(check-within (euclid-difference 32 45 2 3) 51.61 0.1)
```

Übungsblatt 05: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V5 Euklidischer Algorithmus

In der folgenden Aufgabe soll das Konzept der Rekursion verinnerlicht werden. Dazu werfen wir einen Blick auf eine Version des euklidischen Algorithmus, welcher Ihnen vielleicht schon aus der Mathematik bekannt ist. Für zwei natürliche Zahlen `a` und `b` lässt sich mit ihm der größte gemeinsame Teiler der beiden Zahlen berechnen. Dabei geht der Algorithmus wie folgt vor:

Gilt `b = 0` so wird `a` zurückgegeben, ist hingegen `a = 0` so wird `b` zurückgegeben. Gilt `a > b` so wird der Algorithmus mit einem neuen $\hat{a} = a - b$ und dem "alten" `b` aufgerufen. Im anderen Fall wird der Algorithmus mit dem "alten" a und einem neuen $\hat{b} = b - a$ aufgerufen. Definieren Sie eine Prozedur `(euclid a b)`, welche diese Version des euklidischen Algorithmus rekursiv umsetzt.

Lösungsvorschlag:

```
1  ;; Type: natural natural -> natural
2  ;; Returns: the greatest common divisor of the two given numbers
3  (define (euclid a b)
4    (cond
5      [(= 0 a) b]
6      [(= 0 b) a]
7      [(> a b) (euclid (- a b) b)]
8      [else (euclid a (- b a))]))
9
10 ;; Tests
11 (check-expect (euclid 21 2) 1)
12 (check-expect (euclid 55 20) 5)
13 (check-expect (euclid 3 123456789) 3)
```

Übungsblatt 05: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V6 Listenausdrücke auswerten ★☆☆

Teil A: Werden die folgenden Ausdrücke ohne Fehler durch Racket ausgeführt? Falls nein, begründen Sie wo und warum es zu Problemen kommt.

1. `(cons 1 (cons 2 (cons 3)))`
2. `(cons 1 (list 2 (list (list 3 + 4))))`
3. `(list (cons empty 1)(cons 2 empty)(cons 3 empty))`
4. `(first empty)`

Teil B: Liefern die folgenden Listenausdrücke dasselbe Ergebnis zurück?

1. `(cons 1 (cons 2 (cons 3 empty)))` und `(list 1 2 3 empty)`
2. `(cons (list "(list)") empty)` und `(list "list" empty)`
3. `(list 7 "*" 6 "=" 42)` und `(cons 7 (cons "*" (cons 6 (cons "=" (list 42)))))`

Teil C: Gehen Sie nun von folgendem Codeschnipsel aus:

```
1 (define A (list (cons 1 empty)(list 7) 9 ))
2 (define B (cons 42 (list "Hello" "world" "!")))
```

Was liefern die folgenden Aufrufe zurück?

1. `(first (rest A))`
2. `(res (first A))`
3. `(append (first B)(rest (rest A))(first A))`

Lösungsvorschlag:

Teil A:

1. Der Ausdruck ist inkorrekt, da `cons` als 2. Argument immer eine Liste erwartet.
2. Der Ausdruck ist korrekt.
3. Der Ausdruck ist inkorrekt, da `cons` als 2. Argument immer eine Liste erwartet.
4. Der Ausdruck ist inkorrekt, da `first` eine nichtleere Liste erwartet.

Teil B:

1. Nein, denn `(list 1 2 3)` $\neq$ `(list 1 2 3 '())`.
2. Nein, denn `(list (list "(list )" ))` $\neq$ `(list "list" '())`
3. Ja, denn `(list 7 "*" 6 "=" 42)` = `(list 7 "*" 6 "=" 42)`

Teil C:

1. `(list 7)`
2. `'()` (leere Liste)
3. Es wird eine Fehlermeldung geworfen, da `append` als Argumente Listen erwartet. Jedoch ist das erste Argument eine Zahl.

Übungsblatt 05: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V7 Strukturausdrücke auswerten ★☆☆

Gegeben sei folgende Strukturdefinition:

```
1 (define-struct my-pair (one two))
```

Was liefern die folgenden Aufrufe zurück?

1. `(my-pair? (make-my-pair "a" "b"))`
2. `(make-my-pair 1 (make-my-pair 2 empty))`
3. `(* (my-pair-two (make-my-pair 1 2))(my-pair-one (make-my-pair 3 4)))`

Lösungsvorschlag:

1. Wahr, da dies ein Struct von my-pair ist.
2. `(make-my-pair 1 (make-my-pair 2 empty))`
3. 6

Information: Ein Struct ist eine Strukturtypbeschreibung mit null oder mehr Feldern. Structs werden mit einer bestimmten Syntax definiert, die es ermöglicht Prozeduren für die Erstellung von Instanzen vom Typ des Structs und Prozeduren für den Zugriff auf die Felder dieser Instanzen einzusetzen.

Befehl	Syntax	Beispiel
Instanziierung	`(define-struct structname (field1 field2 ...))`	`(define-struct point (x y))`
Zugriff auf ein Feld des Structs	`(structname-fieldname element)`	`(point-x point1`[a]`)`
Überprüfung, ob ein Element vom Typ Struktur X ist	`(structname? element)`	`(point? point1)` (Gibt in diesem Fall `#true` zurück)

Tabelle 1: Einige Befehle von Strukturtypen

[a] Angenommen `point1` ist eine Struktur von Typ `point`

Übungsblatt 05: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V8 Listen in Strukturen

Gegeben ist ein Struct-Typ `abc` mit zwei Feldern `a` und `b`. Definieren Sie eine Funktion `foo` mit einem Parameter `p`. Falls `p` vom Typ `abc` und zudem der Wert im Feld `b` von `p` eine Liste ist, liefert `foo` eine Liste zurück, deren erstes Element der Wert von Feld `a` in `p` ist,und der Rest der zurückgelieferten Liste ist die Liste im Feld `b` von `p` (also eine Liste in der Liste). Andernfalls liefert `foo` einfach `false` zurück.

Lösungsvorschlag:

```
;; Type: ANY -> boolean | (list of ANY (list of ANY))
;; Returns: a list with the first and second
;; field of the input, if the input is a struct of abc and the second field is a list
;; , else false
(define (foo p)
  (if (and (abc? p) (list? (abc-b p)))
      (list (abc-a p) (abc-b p))
      #false))

;; Tests
(check-expect (foo (make-abc 232 (list 439 (list "XY"))))
              (list 232 (list 439 (list "XY"))))
(check-expect (foo (make-abc 232 41241)) #false)
(check-expect (foo "Test") #false)
```

V9 Suche in Zahlenliste

Definieren Sie eine Funktion `contains-x?`. Diese bekommt eine Liste und eine Zahl übergeben und liefert genau dann `true` zurück, wenn die übergebene Zahl mindestens einmal in der übergebenen Liste vorkommt.

Lösungsvorschlag:

```
;; Type: (list of number) number -> boolean
;; Returns: true, if the given number is at least once in the list
(define (contains-x? lst x)
  (cond
    [(empty? lst) #false]
    [(= (first lst) x) #true]
    [else (contains-x? (rest lst) x)]))

;; Tests
(check-expect (contains-x? (list 5.5 5.2 105 2421 2432 54.9) 5) #false)
(check-expect (contains-x? (list 1 5 2 9 8 6 9 2 1 0 9 8 3) 5) #true)
(check-expect (contains-x? (list 5 352 21 432 25 5.2 5.0) 5) #true)
```

Alternative

Der alternative Lösungsvorschlag stammt von Kim Berninger.

```
;; Type: (list of number) number -> boolean
;; Returns: true, if the given number is at least once in the list
(define (contains-x? nums x)
  (and (not (empty? nums))
       (or (= (first nums) x)
           (contains-x? (rest nums) x))))

;; Tests
(check-expect (contains-x? (list 5.5 5.2 105 2421 2432 54.9) 5) #false)
(check-expect (contains-x? (list 1 5 2 9 8 6 9 2 1 0 9 8 3) 5) #true)
(check-expect (contains-x? (list 5 352 21 432 25 5.2 5.0) 5) #true)
```

V10 Duplikate in Zahlenliste

Definieren Sie eine Prozedur `duplicates?` mit einem Parameter `lst`, die genau dann `true` zurückliefert, wenn eine Zahl mehr als einmal in `lst` vorkommt.

> **Hinweis:** Können Sie hier vielleicht Ihre Funktion aus Aufgabe V9 verwenden?

Lösungsvorschlag:

```
;; Type: (list of number) -> boolean
;; Returns: true, if the same number appears more than once in the list
(define (duplicates? lst)
  (cond
    [(empty? lst) #false]
    [(contains-x? (rest lst) (first lst)) #true]
    [else (duplicates? (rest lst))]))

;; Tests
(check-expect (duplicates? (list 1 2 3 4 5 6 7 8 9 0)) #false)
(check-expect (duplicates? (list 1 1.1 1.11 1.5 2 2.5 9 4 3 22)) #false)
(check-expect (duplicates? (list 1 1.1 2.2 3 4 5.324 353 123 12 43 1)) #true)
```

Alternative

Der alternative Lösungsvorschlag stammt von Kim Berninger.

```
;; Type: (list of number) -> boolean
;; Returns: true, if the same number appears more than once in the list
(define (duplicates? nums)
  (and (not (empty? nums))
       (or (contains-x? (rest nums) (first nums))
           (duplicates? (rest nums)))))

;; Tests
(check-expect (duplicates? (list 1 2 3 4 5 6 7 8 9 0)) #false)
(check-expect (duplicates? (list 1 1.1 1.11 1.5 2 2.5 9 4 3 22)) #false)
(check-expect (duplicates? (list 1 1.1 2.2 3 4 5.324 353 123 12 43 1)) #true)
```

Übungsblatt 05: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V11 Verschachtelte Listen ★★★

Definieren Sie eine Prozedur `duplicates-deep?` mit einem Parameter `deep-lst`. Es wird erwartet, dass `deep-lst` entweder eine Zahl oder eine Liste ist, deren Elemente wiederum Zahlen oder Listen sind usw. (Liste von Listen von Zahlen). Die Prozedur `duplicates-deep?` soll `true` oder `false` zurückliefern, und zwar `true` genau dann, wenn mindestens eine Zahl mehr als einmal vorkommt.

> **Hinweis:** Schreiben Sie sich eine Hilfsfunktion `collect` mit zwei Parametern `lst` und `oracle`. Nutzen Sie `oracle` als Akkumulator in dem Sie alle bereits vorgekommenen Zahlen in `oracle` speichern. Machen Sie also aus der verschachtelten Liste wieder eine normale Liste mithilfe von `collect`. Die Funktion aus V10 kann Ihnen hier sehr hilfreich sein.

Lösungsvorschlag:

```
;; Type: number | (list of (number | (list of (number | ...)))) -> boolean
;; Returns: true, if the same number appears more than once in the list
(define (duplicates-deep? deep-lst)
  (local
    (
     ;; Type: (list of (number | (list of (number | ...)))) -> (list of number)
     ;; Precondtion: oracle must be empty at the beginning
     ;; Returns: a singleton list from a nested list
     (define (collect lst oracle)
       (cond
         [(empty? lst) oracle]
         [(cons? (first lst)) (collect (first lst) (collect (rest lst) oracle))]
         [else (collect (rest lst) (cons (first lst) oracle))]))
     )
    (if (number? deep-lst) #false (duplicates? (collect deep-lst empty)))))

;; Tests
(check-expect (duplicates-deep?
               (list (list 1 2 3 4 5) (list (list 6 7 8 (list 0 2))))) #true)
(check-expect (duplicates-deep?
               (list (list 1 2 3 4 5) (list (list 6 7 8 (list 0 (list 22)))))) #false)
(check-expect (duplicates-deep?(list (list 1 2 3 4 5) (list (list 6 7 8 (list 0 (list 22))
               list 99 (list (list 11 958 32 (list 3212) (list 2313)))))))) #false)
```

Alternative

Der alternative Lösungsvorschlag stammt von Kim Berninger.

```
;; Type: number | (list of (number | (list of (number | ...)))) -> boolean
;; Returns: true, if the same number appears more than once in the list
(define (duplicates-deep? deep-lst)
  (local
    (
     ;; Type: number | (list of (number | (list of (number | ...)))) -> (list of number)
     ;; Precondtion: oracle must be empty at the beginning
     ;; Returns: a singleton list from a nested list
     (define (collect lst oracle)
       (cond
         [(empty? lst) oracle]
         [(list? lst) (collect (rest lst) (append oracle (collect (first lst) empty)))]
         [else (list lst)]))
     )
    (duplicates? (collect deep-lst empty))))

;; Tests
(check-expect (duplicates-deep?
               (list (list 1 2 3 4 5) (list (list 6 7 8 (list 0 2))))) #true)
(check-expect (duplicates-deep?
               (list (list 1 2 3 4 5) (list (list 6 7 8 (list 0 (list 22)))))) #false)
(check-expect (duplicates-deep?(list (list 1 2 3 4 5) (list (list 6 7 8 (list 0 (list 22))
               list 99 (list (list 11 958 32 (list 3212) (list 2313)))))))) #false)
```

Information: Lokale Definitionen haben zwei wichtige Funktionen: Sie bieten lokale Namen für Zwischenwerte, was nützlich ist, um komplizierte Ausdrücke in eine Reihe von einfacheren Ausdrücken aufzuteilen. Außerdem bieten sie eine Möglichkeit, die Ergebnisse von Berechnungen mehrfach zu nutzen (anstatt denselben Ausdruck mehrmals zu berechnen).

Als Alternative können globale Definitionen zu demselben Zweck verwendet werden.

Übungsblatt 05: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V12 Arithmetisches Mittel

Gegeben seien folgende Strukturdefinition:

```
1 (define-struct person (age sex))
2 (define-struct student (person id))
```

Eine Person hat ein Alter (als Zahl) und ein Geschlecht (als String). Ein Student wiederum besteht aus einer Person (vorheriges Struct) und einer Matrikelnummer (ein String).

Definieren Sie nun eine Funktion `mean-of-ages`. Diese bekommt eine Liste von Studenten übergeben und gibt das arithmetische Mittel ihrer Alter zurück.

Hinweis: Das arithmetische Mittel $\bar{x}$ berechnen Sie für n Alter $x_1, ..., x_n$ mithilfe von:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i = \frac{x_1 + ... + x_n}{n}$$

Übungsblatt 05: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

Lösungsvorschlag:

```
;; Type: (list of student) -> real
;; Returns: the mean of student ages
(define (mean-of-ages students)
  (local
    (
     ;; Type (list of student) natural -> natural
     ;; Precondtion: sum must be 0 at the beginning
     ;; Returns: the sum of student ages
     (define (sum-age students sum)
       (if (empty? students)
           sum
           (sum-age (rest students)
                    (+ sum (person-age (student-person (first student)))))))
     )
    (if (empty? students)
        0
        (/ (sum-age students 0) (length students))))

  ; Students
  (define student1 (make-student (make-person 22 "Männlich") "7874596"))
  (define student2 (make-student (make-person 33 "Weiblich") "1529563"))
  (define student3 (make-student (make-person 65 "Männlich") "5244253"))
  (define student4 (make-student (make-person 43 "Weiblich") "3276122"))
  (define student5 (make-student (make-person 18 "Männlich") "1265894"))
  (define student6 (make-student (make-person 30 "Weiblich") "1234567"))

  ;; Tests
  (check-expect (mean-of-ages (list student1 student2 student3)) 40)
  (check-expect (mean-of-ages empty) 0)
  (check-within (mean-of-ages
                 (list student1 student2 student3 student4 student5 student6)) 35.16 0.1)
```