Übungsblatt 3

Besprechungstermin: 08.11. - 12.11.

Hinweis: Bitte stellen Sie bei *drscheme* im Menü *Sprache, Sprache auswählen ...* den Wert *Essentials of Programming Languages* ein. Dies gilt auch für alle weiteren Übungsblätter.

Aufgabe 1 (schriftlich)

Prozeduren höherer Ordnung wurden schon auf Übungsblatt 2 behandelt. Jetzt sollen entsprechende Prozeduren entwickelt werden.

1. Definieren Sie eine Prozedur (`quadtafel a b h`), die die Werte der Funktion $f(x) = x^2$ zwischen a und b mit Schrittweite h auf den Bildschirm druckt.

Hinweise: Drucken: (`display <wert>`) – Neue Zeile: (`newline`)

Leerer Platz: (`display " "`) – Rekursive Prozedur verwenden

2. Was gibt Ihre Prozedur als Wert zurück?
3. Erstellen Sie eine zu `quadtafel` analoge Prozedur für den Sinus `sin`.

Hinweis: Die Prozedur (`sin z`) berechnet die Winkelfunktion Sinus im Bogenmaß.

4. Definieren Sie eine Prozedur (`wertetafel proc a b h`), wo man im Parameter `proc` die gewünschte Prozedur angeben kann.
5. Rufen Sie diese Prozedur auf mit `sin`, `quad` und der unbenannten Funktion (`lambda (x) (* x x x)`).

Aufgabe 2

Welche Sprach-Konstrukte muß eine Programmiersprache mindestens enthalten? Geben Sie jeweils Beispiele für Scheme an.

Aufgabe 3

Scheme besitzt verschiedene Zahlentypen und unterscheidet außerdem zwischen exakten und unexakten Werten. Lesen Sie dazu gegebenenfalls im R⁵RS Abschnitt 6.2 nach.

1. Welche Arten von Zahlen kennen Sie aus der Mathematik und welche Zahlentypen sind Ihnen von anderen Programmiersprachen bekannt?
2. Welche Zahlentypen gibt es in *drscheme*? Welche Beschränkungen in Bezug auf Größe und Genauigkeit gibt es für diese?
3. Welchen Zahlentyp liefert die Standard-Prozedur `round`. Ist der Wert exakt oder unexakt?
4. Wie sieht der Ausdruck zur Ermittlung der exakten Summe aus den beiden (exakten) Werten 0.01 und 7/9 aus und was ist das Ergebnis?

Aufgabe 4

Scheme ist eine Sprache mit Blockstruktur und statischer (lexikalischer) Bindung. Erläutern Sie diese Eigenschaften anhand der folgenden Programme:

- Programm 1

```
(define x 0)
(let ((x 10))
  (define (wertx) x)
  x )

(wertx)
```

- Programm 2

```
(define y 0)
(define wertx
  (lambda () y) )

(let ((y 10))
  (wertx) )
```

- Programm 3

```
(define z 0)
(define wertx
  (let ((z 10))
    (lambda () z) ))

(wertx)
```

- Programm 4

```
(define v 0)
(define wertv
  (lambda () v) )

(set! v 5)
(let ((v 10))
  (wertv) )
```

- Programm 4

```
(define w 0)
(define wertw
  (let ((u w))
    (lambda () u) ))
(set! w 5)

(wertw)
```

Kennen Sie Sprachen mit dynamischer Bindung?