Übungen Funktionale Programmierung (in Clojure) Serie 3

1. Funktionen definieren und verwenden

(a) Rufen Sie folgende Funktion in der REPL mit Ihrem Namen auf:
`(fn [name] (str "Hallo " name))`

(b) Rufen Sie folgende Funktion in der REPL mit Ihrem Namen auf:
`#(str "Hallo " %)`

(c) Geben Sie der Funktion aus (a) den Namen `hallo` und rufen Sie sie dann mit Ihrem Namen auf.

(d) Geben Sie der Funktion aus (b) den Namen `hallo` und rufen Sie sie dann mit Ihrem Namen auf.

(e) Definieren Sie die Funktion `hallo` mit `defn` mitsamt einer Kurzdokumentation und rufen Sie sie dann mit Ihrem Namen auf.

2. Lexikalische Bindung

(a) Betrachten Sie die folgende Schachtelung anonymer Funktionen:

```
(fn [x] ((fn [x] (+ x 1)) (+ x 2)))
```

Probieren Sie mit Eingabewerten aus und machen Sie die Struktur graphisch deutlich.

(b) Wodurch unterscheidet sich die folgende Schachtelung und in welcher Beziehung steht sie zu Aufgabe (a)?

```
(fn [x] ((fn [y] (+ y 1)) (+ x 2)))
```

3. Funktionen definieren

Definieren Sie folgende Funktionen und machen Sie in der REPL einige Beispiele für die Verwendung der definierten Funktionen:

(a) `square` mit $x \mapsto x \times x$

(b) `sum-of-squares` mit $(x, y) \mapsto x \times x + y \times y$

(c) `eval-test` mit $(x, y) \mapsto x \times x$

4. Strikte und verzögerte Auswertung

(Die Aufgabe ist sinngemäß übernommen aus dem Coursera-Kurs von Martin Odersky „Funktional Programming Principles in Scala“)

Gegeben sie die in der vorherigen Aufgabe definierte Funktion `eval-test`.

Spielen Sie für die folgenden Anwendungen der Funktion die strikte und verzögerte Auswertungsstrategie durch und geben Sie jeweils an, wieviele Schritte der Substitution nötig sind.

(a) `(eval-test 2 3)`

(b) `(eval-test (+ 3 4) 8)`

(c) `(eval-test 7 (* 2 4))`

(d) `(eval-test (+ 3 4) (* 2 4))`

5. Polynomiale Funktionen

Schreiben Sie Funktionen für folgende polynomiale Funktionen und berechnen Sie die Funktion für die Werte 0, 2 und -2:

(a) $f(x) = 4x + 2$

(b) $f(x) = 9x^3 + x^2 + 7x - 3$

(c) $f(x) = -3x^2 - 4x + 1$

6. Eine merkwürdige Funktion

(Aufgabe 1.5 aus SICP)

Ben Bitdiddle has invented a test to determine whether the interpreter he is faced with is using applicative-order evaluation or normal-order evaluation. He defines the following two procedures:

```
(define (p) (p))

(define (test x y)
  (if (= x 0)
      0
      y))
```

and a test expression:

```
(test 0 (p))
```

What behavior will Ben observe with an applicative-order interpreter? With a normal-order interpreter?

(Assume that the evaluation rule for the special form if is the same regardless of the interpreter's evaluation order: the predicate expression is evaluated first, and the result determines whether to evaluate the consequent or the alternative expression.)

Übertragen Sie das Beispiel außerdem in die Syntax von Clojure und probieren Sie es mit der obigen Eingabe aus. Was passiert? Und warum?