





Vorname:	Nachname:	Matrikelnummer:
----------	-----------	-----------------

### Aufgabe 1: Wissensfragen

6 Punkte

Beantworten Sie die folgenden Fragen in 1–2 kurzen Sätzen!

- (a) Beschreiben Sie, was ein **bindendes** Vorkommen einer Variablen ist. Geben Sie ein Beispiel.

2

- (b) Beschreiben Sie kurz in eigenen Worten das Entwurfsprinzip **Bottom-Up**.

2

- (c) Was ist in Prolog eine **existentielle Abfrage**? Beschreiben Sie, was für eine Form und was für eine Bedeutung so eine Abfrage hat.

2

## Aufgabe 2: Ausdrücke

9 Punkte

Für die Teilaufgaben a) bis c): Welches Ergebnis liefern die folgenden Ausdrücke? Es genügt, wenn Sie das Ergebnis notieren. Die angewendeten Regeln müssen nicht aufgeschrieben werden. Tritt beim Auswerten ein Fehler auf, geben Sie die Art und Ursache des Fehlers an.

(a) `(< (- 3 6) (* -2 3))`

2

(b) Schreiben Sie in dem Ergebnis Listen in der Darstellung `(list ...)`.

2

`^( (+ 1 2) , (+ 3 4))`

(c) `(and (equal? "(+ 1 2)" "(+ 1 2)") "true")`

2

Vorname:	Nachname:	Matrikelnummer:
----------	-----------	-----------------

- (d) Reduzieren Sie den Ausdruck (Zeile 5) in dem folgenden Programm unter der gegebenen Umgebung zu einem Wert. **Geben Sie alle nötigen Reduktionsschritte unter Angabe der Regelnamen an.** Benutzen Sie die Reduktionsregeln der BSL-Sprache.

3

```

1  ; In der Umgebung befindet sich diese Definition
2  (define (f a) (posn-x a))
3
4  ; reduzieren Sie den folgenden Aufruf
5  (f (make-posn 4 2))

```

### Aufgabe 3: Typ- und Funktionsdefinition

20 Punkte

Sie entwickeln eine Software zur Planung von Fahrradausflügen. Hierfür müssen Sie verschiedene Informationen verarbeiten:

- Ein Streckenabschnitt hat die Eigenschaften: Distanz in Kilometern und Steigung.
- Die Steigung<sup>1</sup> ist entweder **flach** (unter 4%), **mittel** (4% – 8%) oder **steil** (mehr als 8%). **Vereinfachend gehen wir davon aus, dass es immer nur bergauf geht.**
- Eine Strecke ist eine Liste von Streckenabschnitten.

- (a) Definieren Sie Datentypen für **Steigung** und **Abschnitt** (für den Streckenabschnitt) und geben Sie an, um welche Art von Datentyp es sich jeweils handelt.

5

- (b) Schreiben Sie eine Funktion, (**fahrtzeit a**), die für einen Streckenabschnitt die Fahrtzeit in Stunden berechnet. Nehmen Sie dabei folgende Durchschnittsgeschwindigkeiten an: 25 km/h bei flacher Steigung, 15 km/h bei mittlerer Steigung und 5 km/h bei steiler Steigung. **Denken Sie daran, das Entwurfsrezept für Funktionen zu beachten.**

7

(define (**fahrtzeit** a)

<sup>1</sup>Die Steigung gibt an, wie viel Prozent der zurückgelegten Distanz an Höhe zurückgelegt werden. Zum Beispiel ist bei einer Steigung von 1% und einer Distanz von 1km die zurückgelegte Höhe:  $1km * 1/100 = 0,01km$ .

Vorname:	Nachname:	Matrikelnummer:
----------	-----------	-----------------

- (c) Schreiben Sie nun eine Funktion (`hoehe loa`), die für eine übergebene Strecke (eine Liste von `Abschnitt`) die insgesamt zurückgelegte Höhe berechnet. Sie müssen in dieser Teilaufgabe keine Tests definieren.

5

```
(define (hoehe loa)
```

Für die folgenden Teilaufgaben benötigen Sie **nicht mehr** die obigen Angaben.

- (d) Gegeben ist die untenstehende Funktion mit einer unvollständigen Signatur. Geben Sie den fehlenden Typ für den Parameter `z` an und begründen Sie den Typ indem Sie auf die Funktionsdefinition Bezug nehmen. Gehen Sie davon aus, dass **alle verwendeten Listen homogen** sind, also nur Werte von einem Typ enthalten. Dabei kommen als Element-Typen einer List **keine** Summentypen in Frage. Begründen Sie alle Teile der Signatur, und nehmen Sie dabei auf die Funktionsdefinition Bezug!

3

```
1 [U V] (list-of U) (U -> Number) _____ -> (list-of V)
2 (define (f x y z)
3   (cond [(empty? x) empty]
4         [(< 0 (y (first x))) (cons (z (first x)) (f (rest x) y z))]
5         [else (f (rest x) y z)]))
```

#### Aufgabe 4: Listen und Funktionen höherer Ordnung

10 Punkte

**Hinweis:** Die folgenden aus der Vorlesung bekannten Funktionen höherer Ordnung dürfen Sie in Aufgabenteil a) und b) verwenden:

```
; [X] (X -> Boolean) (listof X) -> (listof X)
; Gibt eine Liste zurück mit allen Elementen aus l, die das Prädikat p erfüllen
(filter p l)

; [X Y] (X -> Y) (listof X) -> (listof Y)
; Bildet alle Elemente aus l mit f ab und liefert die Liste der Ergebnisse.
(map f l)

; [X Y] (X Y -> Y) Y (listof X) -> Y
; Kombiniert alle Elemente der Liste l durch f, wobei das letzte Element mit
; base kombiniert wird. Die leere Liste wird auf base abgebildet, die Elemente
; werden von rechts nach links durchlaufen.
(foldr f base l)
```

Definieren Sie die nachfolgenden Racket-Funktionen in Teilaufgabe a) und b) **nur durch Aufrufe der oben genannten Funktionen höherer Ordnung**. Das heißt, der Body-Ausdruck Ihrer Funktion **darf keine Aufrufe an andere primitive oder selbst-definierte Funktionen enthalten**. Primitive oder selbst-definierte Funktionen dürfen aber als Argument an die Funktionen höherer Ordnung übergeben werden.

(a) ; [X] (listof X) (listof X) -> (listof X)  
; Fügt die beiden Listen zusammen,  
; so dass der Anfang der Ergebnisliste l1 und das Ende l2 ist.

```
(check-expect (append1 '(1 2 3 4) '(5 6 7 8)) '(1 2 3 4 5 6 7 8))
(define (append1 l1 l2)
```

3



Vorname:	Nachname:	Matrikelnummer:
----------	-----------	-----------------

(b) ; (listof (listof Number)) -> (listof Number)  
; Gibt für eine Liste von Zahlen l eine neue Liste zurück, die  
; jeweils die Quadrate der Zahlen enthält.

2

```
(check-expect (f '(0 1 2 3 4)) '(0 1 4 9 16))
(check-expect (f '()) '())
```

```
(define (f l)
```

(c) Implementieren Sie die untenstehende Funktion, **ohne** die Funktionen `foldr`, `map` und `filter` zu benutzen.

5

```
; [X] (X -> boolean) (listof X) -> (listof X)
; Gibt eine Liste der ersten Elemente der übergebenen Liste l
; zurück, für die das Prädikat p erfüllt ist. Ab dem ersten
; Element, das das Prädikat nicht erfüllt werden keine weiteren
; Elemente mehr in die Ergebnisliste übernommen.
```

```
(define (takewhile p l)
```

### Aufgabe 5: Rekursion & Terminierung

12 Punkte

- (a) Implementieren Sie mittels **Akkumulator** die Funktion (`remove-from l v`). Diese Funktion gibt eine Liste zurück, die der Liste `l` nach der Entfernung aller Vorkommen von `v` entspricht. Außerdem ist als letztes Element an die Ergebnisliste die Anzahl der entfernten Vorkommen von `v` angehängt. **Verwenden Sie die Schablone für Funktionen mit Akkumulator und geben Sie die Akkumulator-Invariante mit an.**

8

```
; List X -> List
(check-expect (remove-from '(1 2 3 1 4) 1) '(2 3 4 2))
(check-expect (remove-from '() 42) '(0))
(define (remove-from l v)
```

Vorname:	Nachname:	Matrikelnummer:
----------	-----------	-----------------

4

- (b) Die folgende Funktion `count-squares` soll berechnen, wie viele Quadrate in das Rechteck passen, welches durch die übergebene Breite und Höhe bestimmt ist. Gehen Sie davon aus, dass nur **Ganzzahlen größer als Null** als Argumente übergeben werden.

```
(define (count-squares breite hoehe)
  (if (= breite hoehe)
      1
      (if (> breite hoehe)
          (+ 1 (count-squares (- breite hoehe) hoehe))
          (+ 1 (count-squares breite (- hoehe breite))))))
)
```

Um welche Form der Rekursion handelt es sich? Terminiert diese Funktion? Begründen Sie Ihre Antwort.

### Aufgabe 6: Prolog

14 Punkte

- (a) Definieren Sie die folgende Prozedur in Prolog. Die Verwendung von Bibliotheks-Prozeduren aus Prolog ist dabei **nicht** gestattet. Definieren Sie benötigte Hilfsprozeduren selbst.

5

**Hinweis:** Die arithmetischen Vergleichsprädikate `<`, `>`, `=<`, `>=`, `==` (Gleichheit), `\=` (Ungleichheit) dürfen verwendet werden und werden in Prolog infix geschrieben, zum Beispiel: `?- 2 =< 3.`

`listafter(L1, V, L2)`: `L1` ist eine Liste, `V` ist ein beliebiger Wert und `L2` ist ebenfalls eine Liste. Dabei ist `L2` die Teilliste von `L1`, die mit dem ersten Element von `L1` beginnt, das dem Wert `V` entspricht, und mit dem letzten Element von `L1` endet. Wenn `V` nicht in `L1` vorkommt, ist `L2` die leere Liste.

Vorname:	Nachname:	Matrikelnummer:
----------	-----------	-----------------

Gegeben seien die folgenden Definitionen der Prozeduren b – d und die Abfragen in Teilaufgaben b) – d), die diese verwenden. Geben Sie für jede Abfrage an, was das **Resultat** ist. Wenn eine Abfrage erfüllt ist, geben Sie eine gültige **Substitution** aller Variablen an. Im Fall eines **Fehlers** geben Sie eine kurze **Begründung** an.

$b(z, [], X).$

$b(x(D), [X|Xs], X) :- b(D, Xs, X).$

$b(o(D), [X|Xs], Y) :- Y \backslash = X, b(D, Xs, Y).$

$c(X, [apfel, banane]).$

$d(eins, zwei, drei, vier).$

(b)  $?- b(Desc, [1, 2, 3], 2).$

3

(c)  $?- c([Apfel, Banane], [Apfel, Banane]).$

3

(d)  $?- d(Eins, Zwei, Rest).$

3

## Aufgabe 7: Reduktion und Äquivalenz

20 Punkte

Für die folgenden Teilaufgaben a) und b) gehen Sie davon aus, dass sich die folgenden Definitionen in der Umgebung befindet.

```
; (listof Number) -> (listof Number)
; Verdreifacht den Wert jedes Listenelements
(define (trpl l)
  (cond [(empty? l) empty]
        [else (cons (* 3 (first l)) (trpl (rest l)))]))

; (listof Number) -> Number
; Berechnet die Summe der Listenelemente
(define (sum l)
  (cond [(empty? l) 0]
        [else (+ (first l) (sum (rest l)))]))
```

Es soll die Äquivalenz  $(\text{sum } (\text{trpl } l)) \equiv (* 3 (\text{sum } l))$  durch strukturelle Induktion über  $l$  bewiesen werden. Folgende Äquivalenz-Regeln können ohne Beweis verwendet werden:

**(EFUN-sum-empty)**  $(\text{sum } l) \equiv 0$ , falls  $l$  die leere Liste ist

**(EFUN-sum-non-empty)**  $(\text{sum } l) \equiv (+ (\text{first } l) (\text{sum } (\text{rest } l)))$ , falls  $l$  eine nicht-leere Liste ist

**(EFUN-trpl-empty)**  $(\text{trpl } l) \equiv \text{empty}$ , falls  $l$  die leere Liste ist

**(EFUN-trpl-non-empty)**  $(\text{trpl } l) \equiv (\text{cons } (* 3 (\text{first } l)) (\text{trpl } (\text{rest } l)))$ , falls  $l$  eine nicht-leere Liste ist

- (a) Stellen Sie die im **Induktionsanfang zu beweisende Äquivalenz** auf und führen Sie den **Induktionsanfang** durch.

5

Vorname:	Nachname:	Matrikelnummer:
----------	-----------	-----------------

- (b) Stellen Sie die im **Induktionsschritt zu beweisende Äquivalenz** und die **Induktionsannahme** auf. Führen Sie anschließend den **Induktionsschritt** durch.

11

Die folgende Teilaufgabe ist nicht mehr Teil des Äquivalenzbeweises und die Funktionen `trpl` und `sum` werden nicht mehr benötigt.

- (c) Implementieren Sie die Funktion `(paar-sum lon)` in Racket **mithilfe von Pattern-Matching**. Sie dürfen in dieser Teilaufgabe keine Selektorfunktionen verwenden!

4

Diese Funktion bekommt als Argument eine Liste von Zahlen `lon` und erzeugt eine neue Liste. Die Ergebnisliste enthält die Summen von jeweils zwei aufeinander folgenden Zahlen aus `lon`. Sollte `lon` eine ungerade Anzahl von Werten enthalten, wird das letzte Element aus `lon` direkt in die Ergebnisliste übernommen.

```
(check-expect (paar-sum '()) '())  
(check-expect (paar-sum ' (1 2 3 4)) '(3 7))  
(check-expect (paar-sum '(1 2 3 4 5)) '(3 7 5))
```

```
(define (paar-sum lon)
```



Vorname:	Nachname:	Matrikelnummer:
----------	-----------	-----------------

### Aufgabe 8: Algebraische Datentypen

9 Punkte

Es soll ein Programm zur Verwaltung von Immobilien geschrieben werden. Im ersten Schritt soll das Programm mit Einfamilienhäusern und Mehrfamilienhäusern arbeiten. Ein Einfamilienhaus wird durch die Adresse und die Anzahl der Zimmer beschrieben. Mehrfamilienhäuser werden beschrieben durch die Adresse, sowie eine Liste, die für jede Wohnung die Anzahl an Zimmern enthält.

- (a) Definieren Sie geeignete Strukturen zur Repräsentation von *Einfamilienhäusern* (EFH) und *Mehrfamilienhäusern* (MFH) und geben Sie eine Datendefinition für den **Algebraischen Typen** *Immobilie* an.

5

- (b) Implementieren Sie eine Funktion `miete immo` zur Berechnung der Mieteinnahmen einer Immobilie, basierend auf der Definition aus Aufgabenteil a). Gehen Sie dabei davon aus, dass pro Raum eine Miete von 100 Euro fällig ist. **Wenden Sie die Schablone für algebraische Datentypen an.**

4

```
(define (miete immo)
```