Begonnen am	Mittwoch, 10. März 2021, 11:32
Status	Beendet
Beendet am	Mittwoch, 10. März 2021, 13:04
Verbrauchte Zeit	1 Stunde 31 Minuten
Bewertung	**25,00** von 90,00 (**28**%)

Information

Aufgabe 1 - Grundlegende Begriffe und Programmierkonzepte - 29 Punkte

Information

Jede richtige Antwort wird mit je 1 Punkt, jede falsche oder nicht gegebene Antwort mit 0 Punkten bewertet.

Frage **1**
Vollständig
Erreichte Punkte 1,00 von 1,00

Ein syntaktisch korrekter Racket-Ausdruck führt durch Auswertung immer zum gewünschten Ergebnis.

Bitte wählen Sie eine Antwort:
- Wahr
- Falsch

Frage **2**
Vollständig
Erreichte Punkte 0,00 von 1,00

Die Einrückung von Racket-Programmen spielt für ihre Semantik keine Rolle.

Bitte wählen Sie eine Antwort:
- Wahr
- Falsch

Frage **3**

Vollständig

Erreichte Punkte 1,00 von 1,00

Pseudofunktionen heißen Pseudofunktionen, weil ihre Syntax der von Funktionen entspricht, aber nicht ihre Semantik.

Bitte wählen Sie eine Antwort:

- (•) Wahr
- ( ) Falsch

Frage **4**

Vollständig

Erreichte Punkte 1,00 von 1,00

Rekursive Aufrufe einer Funktion stehen in der Regel innerhalb einer Fallunterscheidung.

Bitte wählen Sie eine Antwort:

- (•) Wahr
- ( ) Falsch

Frage **5**

Vollständig

Erreichte Punkte 0,00 von 1,00

Zur Verarbeitung gemischter Daten sollte eine Funktion in der Regel einen akkumulierenden Parameter besitzen.

Bitte wählen Sie eine Antwort:

- (•) Wahr
- ( ) Falsch

Frage **6**

Vollständig

Erreichte Punkte 1,00 von 1,00

Zu den Atomen in Racket gehören Zahlen, Symbole und Listen.

Bitte wählen Sie eine Antwort:

- ( ) Wahr
- (•) Falsch

Frage **7**

Vollständig

Erreichte Punkte 3,00 von 4,00

Was unterscheidet problemorientierte von maschinenorientierten Programmiersprachen?

Problemorientierte Sprachen sind Sprachen die zum erlernen des Programmierens genutzt werden wie z.b. PASCAL usw. Dabei wird der geschriebene Code durch einen Compiler (Übersetzer) dem Computer ausführbar gemacht.

Maschienenorientierte sprachen oder auch Assembler Sprachen sind nicht so leicht zu erlernen und heutzutage nur noch in spezial fällen genutzt (z.b. hochleistungsrechnern). Außerdem kann man alle Möglichkeiten des Mikroprozessors nutzen.

Kommentar:
etwas ungenau

Frage **8**

Vollständig

Erreichte Punkte 0,00 von 4,00

Erläutern Sie anhand eines Beispiels den Zusammenhang zwischen einer Datendefinition für beliebig lange Listen und einer dazu passenden Funktionsschablone.

Kommentar:

Frage **9**

Vollständig

Erreichte Punkte 1,00 von 2,00

Erläutern Sie durch ein bis zwei Sätze die Wirkungsweise der bekannten Funktion höherer Ordnung `filter`.

Durch die Funktion filter ist es möglich z.b. beliebig lange Listen nach einer bestimmten Bedingung abzufragen. So ist es möglich aus einer liste mit Elementen die für den Zweck relevanten Daten zu entnehmen.

Kommentar:
ungenau

Frage **10**

Vollständig

Erreichte Punkte 0,00 von 4,00

Geben Sie für den folgenden Ausdruck alle Auswertungsschritte bis zum Endergebnis an:

```
((lambda [l]
   (first (rest (rest l))))
 (cons 3 (list 2 1 0)))
```

```
= beispiel mit einer Liste '(2 3 4 5)
((lambda ['(2 3 4 5)]
   (first (rest (rest l))))
 (cons 3 (list 2 1 0)))

=
(first (rest (rest 3 4 5))))
(cons 3 (list 2 1 0)))

=
(first (rest 4 5)))
(cons 3 (list 2 1 0)))

=
(4)
(3 2 1 0)))

= Als ergebnis bekommen wir 2 Listen mit den Werten '(4) und '(3 2 1 0)
```

Kommentar:
komplett falsch

Frage **11**

Vollständig

Erreichte Punkte 4,00 von 4,00

Geben Sie, falls möglich, jeweils eine Anwendung der folgenden Funktion an, so dass das Resultat `'a`, `'b` bzw. `'c` lautet. Falls es keine solche Anwendung gibt, begründen Sie.

```
(define h
  (lambda [x y]
   (cond [(= x y) 'a]
         [(not (< x y)) 'b]
        [(<= y x) 'c]
         [else 'd])))
```

```
(= (h 1 1) 'a)
(= (h 5 2) 'b)

Für c gibt es keine Anwendung, da [(not (< x y)) 'b] die gleiche
bedingung ist wie [(<= y x) 'c] und immer bei der ersten
Bedingung ein ergebnis rauskommen wird, bevor die zweite
überhaupt abgefragt wird.
```

Kommentar:

Frage **12**

Vollständig

Erreichte Punkte 0,00 von 2,00

Geben Sie einen Ausdruck an, der zu dem symbolischen Ausdruck (S-Ausdruck) `'((2 #true))` äquivalent ist, und weder `'` noch `list` verwendet.

```
'((2 #true)) = 2
```

Kommentar:
richtig wäre:

(cons (cons 2 (cons #true empty)) empty)

Frage **13**

Vollständig

Erreichte Punkte 1,00 von 3,00

Geben Sie eine sinnvolle Sammlung von Tests für eine Funktion `abs` an, die den Absolutbetrag einer ganzen Zahl berechnen soll.

```
(check-expect (abs (-3)) 3)
(check-expect (abs 100) 100)
(check-expect (abs (-123)) 123)
(check-expect (abs 0) 0)
(check-expect (abs (-0)) 0)
```

Kommentar:
3 Tests falsch

Information

Aufgabe 2 - Kartenspiel - 26 Punkte

Information

Alle Funktionen dieser Aufgabe sind regelkonform aufzuschreiben. Es ist nützlich, die Teilaufgaben in der vorgegebenen Reihenfolge zu bearbeiten.

Für ein einfaches Kartenspiel gelte:

- Es gibt 4 Farben: kreuz, pik, herz, karo
- Es gibt nur Kartenwerte von 1 bis 10
- Bube, Dame, König, Ass gibt es **nicht**.

Daraus ergibt sich die folgende Datendefinition:

Eine `karte` ist eine zweielementige Liste
`(list farbe wert)`
für die gilt: `farbe` ist eines der Symbole `'kreuz, 'pik, 'herz, 'karo` und `wert` ist eine natürliche Zahl zwischen 1 und 10.

Frage **14**
Vollständig
Erreichte Punkte 6,00 von 6,00

Schreiben Sie zwei Funktionen mit Namen `farbe` und `wert`, die jeweils eine Karte als Argument akzeptieren und jeweils die Farbe bzw. den Wert der Karte liefern. Der Vertrag der Funktion `farbe` sähe also wie folgt aus:
`farbe: karte -> symbol`

```
;; leifert die farbe einer karte (list)
;; farbe: karte -> symbol

(define farbe (lambda [karte] (cond
                               [(empty? karte) (error 'ungueltige-karte "keine gueltige Karte")]
                               [else (first karte)])))
;; Beispielanwendungen
(check-expect (farbe '("pik" 3)) "pik")
(check-expect (farbe '("kreuz" 10)) "kreuz")

;; leifert den wert einer karte (list)
;; wert : karte -> symbol

(define wert (lambda [karte] (cond
                               [(empty? karte) (error 'ungueltige-karte "keine gueltige Karte")]
```

Kommentar:

Frage **15**

Vollständig

Erreichte Punkte 3,00 von 7,00

Schreiben Sie eine Funktion `gueltige-karte?`, die eine Karte auf Gültigkeit überprüft, d. h. der Wert muss zwischen 1 und 10 liegen und die Farbe eines
der Symbole `'kreuz, 'pik, 'herz` oder `'karo` sein.

Der Aufruf `(gueltige-karte? '(kreuz 7))` sollte also `#true` liefern.

Hinweis: Es könnte hilfreich sein, die Standardfunktion `member?` zu benutzen. Der Aufruf `(member? element liste)` liefert `#true`, wenn *element* in der Liste `liste` vorkommt.

```
;; prueft die Gueltigkeit einer karte (list)
;; pruefe-gueltigkeit?: karte -> boolischer wert

(define pruefe-gueltigkeit? (lambda [karte] (cond
                                [(empty? karte) (error 'ungueltige-karte "keine gueltige Karte")]
                                [(member? "pik" karte) #true]
                                [(member? "kreuz" karte) #true]
                                [(member? "herz" karte) #true]
                                [(member? "karo" karte) #true]
                                [(and (<= 10 (first ( rest karte))) (>= 1 (first ( rest karte))))
                                [else #false])))
;; Beispielanwendungen
(check-expect (pruefe-gueltigkeit? '("pik" 3)) #true)
(check-expect (pruefe-gueltigkeit? '("kreuz" 10)) #true)
```

Kommentar:

zu wenig Tests

Wertprüfung ist falsch, Farben sind Symbole nicht Zeichenketten

Frage **16**

Vollständig

Erreichte Punkte 1,00 von 4,00

Formulieren Sie eine Datendefinition `blatt` für eine beliebig lange (möglicherweise leere) Liste von Karten. Leiten Sie daraus eine passende Funktionsschablone für Funktionen zur Verarbeitung von Listen von Karten ab.

```
(define blatt '(...))

(define blatt (lambda [...] (cond
                                    [(empty? ...) empty]
                                    [else (blatt (cons (... (first ...)) (rest ...)))])))
```

Kommentar:

Datendefinition fehlt; Funktionsschablone fehlerhaft

Frage **17**

Vollständig

Erreichte Punkte 1,00 von 9,00

Schreiben Sie eine Funktion `alle-gueltig?`, die für alle Karten eines Blattes prüft, ob sie gültig sind. Für die Funktion dürfen **keine** Funktionen höherer Ordnung benutzt werden.

Hinweis: Die Existenz der Funktion `gueltige-karte?` aus obigem Aufgabenteil darf vorausgesetzt werden.

```
;; prueft die Gueltigkeit einer ansamlung von karten (list)

(define blatt (lambda [kartendeck] (cond
                                    [(empty? kartendeck) empty]
                                    [else (blatt (cons (pruefe-gueltigkeit? (first kartendeck)) (rest

;; Beispielanwendungen
(check-expect (blatt '(("kreuz" 10) ("pik" 3))) #true)
```

Kommentar:

Der Test ist nahezu korrekt, sonst nichts

Information

Aufgabe 3 - Ahnenforschung - 24 Punkte

Information

Im gesamten Aufgabenteil 3 genügt es, die reine Funktionsdefinition aufzuschreiben. Sie können dabei auf Kommentar, Vertrag, Tests etc. verzichten.

Dieser Aufgabe liegt die Entwicklung einer Datenbank zur Ahnenforschung zugrunde.

Frage **18**

Vollständig

Erreichte Punkte 1,00 von 3,00

Ein Datensatz dieser Datenbank ist durch folgende Struktur definiert:

```
(define-struct datensatz [name mutter])
```

Dabei gibt das Attribut `name` einen eindeutigen Bezeichner eines Nutzers an. Im Attribut `mutter` kann der `datensatz` der Mutter hinterlegt werden. Ist diese unbekannt, so wird `#false` hinterlegt.

Legen Sie vier Beispieldatensätze an, zwei davon sollen in einer Enkel-Beziehung zu einem Datensatz stehen.

```
(make-datensatz "Heinz" "Ulrike")
(make-datensatz "Anna" "Ulrike")
(make-datensatz "Hannah" "Anna")
(make-datensatz "Gustav" #false)
```

Kommentar:

Für die mutter-Komponente kein datensatz hinterlegt

Frage **19**

Vollständig

Erreichte Punkte 0,00 von 4,00

Erstellen Sie eine Funktion `grossmutter`, die für einen übergebenen Nutzer die Großmutter ermittelt, sofern diese angegeben wurde.

```
(define grossmutter (lambda [daten] (cond
                      [(empty? daten) empty]
                      [else (filter (lambda [x] (equal? (datensatz-mutter x) (datensatz-name x))) date

(grossmutter liste)
```

Kommentar:

Funktion ist falsch

Frage **20**

Vollständig

Erreichte Punkte 0,00 von 4,00

Erstellen Sie eine Funktion `enkel`, die alle Enkel (nicht Großenkel, etc.) einer übergebenen Nutzerin aus einer Liste von Nutzern ermittelt. Sie können hierfür die Funktion `grossmutter` als existent ansehen. Für einen möglichen Vergleich von Datenstrukturen nutzen Sie die Funktion `equal?`.

Kommentar:

Frage **21**

Vollständig

Erreichte Punkte 0,00 von 13,00

Erstellen Sie eine Funktion `verwandt?`, die für zwei übergebene Nutzer prüft, ob diese gemeinsame Vorfahren hatten. Dabei wird eine Verwandtschaft väterlicherseits außer Acht gelassen.

- Erstellen Sie dabei zunächst eine Funktion `vorfahren`, die für einen Nutzer alle Vorfahren ermittelt. (6P)
- Erstellen Sie dann eine Funktion `gemeinsameElemente?`, die angibt, ob zwei Listen mindestens ein Element gemeinsam haben. (4P)
- Kombinieren Sie beide Funktionen in der `verwandt?`-Funktion. Denken Sie bitte auch an den Sonderfall, dass die Mutter unbekannt ist. (3P)

Kommentar:

Information

Aufgabe 4 - Rekursive Induktion - 11 Punkte

Frage **22**

Vollständig

Erreichte Punkte 0,00 von 11,00

Gegeben sei folgende Funktion:

```
(define f
  (lambda [n]
    (cond
      [(= 1 n) 2]
      [else (+ (* n (+ n 1)) (f (- n 1)))])))
```

Beweisen Sie mittels rekursiver Induktion, dass der Aufruf `(f n)` für jede natürliche Zahl $n \geq 1$ den Wert $\frac{n(n+1)(n+2)}{3}$ liefert.

```
;= siehe Aufgabenstellung
(/ (* 0 (+ 0 1))2)

;= siehe Aufgabe als Ergebnis
(/ (* n (+ n 1))2)

;Induktionsannahme: Die Behauptung gilt für Rekursionstiefe k = n.
; (f n) = (/ (* n (+ n 1))2) --> I.V.

;I.S. Behauptung aufstellen mit (n+1)
;Kommentar: Die Funktion soll auch für (n+1) gelten

; Behauptung: (f (+ n 1) = ( (n+1) * (n+2)) / 2)


(f (+ n 1))
;= durchgehen bis zum rekursiven Aufruf
(cond
     [(= (+ n 1) 0) 0]
     [else (+ (+ n 1) (f (- (+ n 1) 1)))])
;=
(cond
     [(false 0) 0] ;weil n+1 >= 0
     [else (+ (+ n 1) (f (- n 1)))]
;=
(+ (+ n 1) (f (- (+ n 1) 1)))
;=
(+ (+ n 1) (f n)
;= I.V. einsetzen (f n) = (/ (* n (+ n 1))2)
(+ (+ n 1) (/ (* n (+ n 1))2))
;=
(n+1) + ((n * (n+1))/2)
;=
(((n+1)*2)/2) + ((n * (n+1))/2)
;=
(2n*2+n^2+n)/2
;=
(n^2+3n+2) / 2
;=
( (n+1) * (n+2)) / 2)
```

Kommentar:
alle Teile des "Beweises" sind falsch

◄ Regeln Einführung in die Programmierung

Direkt zu:

Direkt zu:

Klausurabgabe Längerschreiber ►