Fragenkatalog Informatik 1 für Medizintechniker (WS 17/18, bis Skript Seite 145 bzw. bis Weihnachten)

Dieser Fragenkatalog erhebt natürlich keinerlei Anspruch auf Vollständigkeit und Richtigkeit, ich habe ihn für mich selber zum Lernen erstellt und jetzt halt ins Forum hochgeladen.

1. Was ist ein Algorithmus?
2. Was sind atomare, was zusammengesetzte Ausdrücke?
3. Was ist die Arithmetik eines Datentyps?
4. Welche eingebauten Datentypen gibt es in BSL?
5. Was sind Magic Literals?
6. Was versteht man unter DRY?
7. Welche 2 grundsätzlichen Typen von Fehlern gibt es?
8. Was sind Bsp für diese Typen?
9. Was ist das Vokabular eines Programms?
10. Was sind primitive Funktionen?
11. Wie lauten und wie unterscheiden sich die Entwurfsrezepte für allgemeine Fkt (Funktionen), Fkt mit Summentypen, Fkt mit Produkttypen, Fkt mit ADTs?
12. Was ist die Spezifikation einer Fkt?
13. Welche Elemente einer Fkt gehören auf die Wunschliste?
14. Was beschreibt das Prinzip des Information Hiding?
15. Was unterscheidet Batchprogramme von interaktiven Programmen?
16. Was bedeutet Konfluenz?
17. Was ist eine „strikte“, was eine „nicht-strikte Funktion?
18. Was sind Summentypen, welche „Untertypen von Summentypen“ gibt es?
19. Wie sehen die zugehörigen Datendefinitionen aus?
20. Woher kommt der Name „Summentyp“, woher der Name „Produkttyp“?
21. Was sind „Vereinigungstypen“, was „union-types“, was „Itemizations“?
22. Was bedeutet es, wenn 2 Mengen „disjunkt“ sind?
23. Was ist wichtig bei der Festlegung der Alternativen eines Summentyps?
24. Was ist eine Instanz einer Struktur?
25. Welche Fkt werden bei einer Strukturdefinition automatisch von DrRacket miterstellt? Wie heißen diese Fkt allgemein, wie ruft man sie in BSL auf?
26. Wie sieht eine Datendefinition für einen Produkttypen aus?
27. Was bezeichnet man als „Tagged Union“?
28. Was sind algebraische Datentypen?
29. Wann sind Datentypen „isomorph“?
30. Was definiert die Syntax einer Programmiersprache?
31. Was definiert die Grammatik einer Sprache?
32. Was ist ein Terminalsymbol, was ein Nichtterminal?
33. Was ist die EBNF (Erweiterte Backus-Naur-Form)?
34. Was ist der Unterschied von einem ALB zu einem SALB? (**A**b**l**eitungs**b**aum und **S**ystem von **A**b**l**eitungs**b**äumen)
35. Was definiert die Metagrammatik?
36. Was ist der Unterschied zwischen konkreter und abstrakter Syntax einer Sprache?
37. Was enthält die (BSL-)Kernsprache im Unterschied zur gesamten Sprache?
38. Was ist das Environment oder die Umgebung eines Programms?
39. Was beschreibt die Auswertungsposition in einem Programm?
40. Was ist ein „Auswertungskontext“?
41. Wie lautet die Definition der Kongruenzregel?
42. Was beschreibt die Auswertungsregel (PROG)?
43. Was bezeichnet man als „Datenuniversum“?
44. Was ist die Grundvoraussetzung, dass Äquivalenz gelten kann?

45. Was bezeichnet man als Selbstähnlichkeit?
46. Wo taucht sie in BSL, bzw. allgemein beim Programmieren, auf?
47. Wie ist ein rekursiver Datentyp aufgebaut?
48. Auf welche 3 Arten können in BSL Listen erzeugt werden? Wie kann jeweils eine leere Liste erzeugt werden?
49. Was ist eine „homogene Liste“ und wie taucht sie in einer Signatur auf?
50. Was bedeutet Konkatenation (von Mengen oder Listen)?
51. Welchen Unterschied haben die primitiven Prädikatsfunktionen (cons? …) und (list? …)?
52. Wie sollte nach Entwurfsrezept beim Schreiben einer rekursiven Fkt vorgegangen werden?
53. Wie lässt sich Äquivalenz bei rekursiven Fkt nachweisen?
54. Was ist beim Pattern Matching die Syntax für „Prädikatsfunktionen“ wie (boolean? …)?
55. Wie „matcht“ man auf eine Struktur, zB. (make-posn 0 4)?
56. Als was für eine Fkt (mit welchen Ein- und Ausgabewerten) kann man sich Pattern Matching vorstellen?
57. Was passiert beim Pattern Matching, wenn der Eingabewert auf keines der Patterns „matcht“?
58. Was ist in BSL ein Symbol?
59. Welche 4 „Regeln“ gibt es für die Quote/Quasiquote/Unquote-Operatoren?
60. Wie sind „S-Expressions“ informell definiert?
61. Wie lautet die Datendefinition für S-Expressions?
62. Was ist ein „sicherer Weg“, um eine S-Expression zu erstellen?
63. Was bedeutet „Homoikonizität“?
64. Was bedeutet „generisch“, z.B. in „generische Funktion“?
65. Was sind „getypte Datentypen“?
66. Werden in BSL getypte oder ungetypte Datentypen verwendet?
67. Was zeichnet eine „ungetypte Sprache“ aus?
68. Was zeichnet eine „dynamisch getypte Sprache“ aus?
69. Was ist ein „Contract“ allgemein?
70. Was sind die Vorteile und Nachteile von Contracts, bzw. von dynamisch getypten Sprachen?
71. Wie kann man einen Contract in BSL „nachbauen“?
72. Was zeichnet „statisch getypte Sprachen“ aus?
73. Was bedeutet Kompositionalität?
74. Was versteht man unter einem „wohlgetypten Programm“?
75. Was ist der Unterschied zwischen einem „Typchecker“ und den bisher bekannten Tests (check-expect usw.)?

Antworten

1. Was ist ein Algorithmus?
   Ein Algorithmus ist eine Abfolge von sinnvollen Anweisungen, um Probleme zu lösen, z.B. ein Algorithmus für eine mathematische Berechnung oder eben die Abfolge von Befehlen/Funktionen in einer Programmiersprache.

2. Was sind atomare, was zusammengesetzte Ausdrücke?
   Ein atomarer Ausdruck ist ein Literal, er kann nicht weiter reduziert werden, wie z.B eine Zahl oder ein String. Ein zusammengesetzter Ausdruck ist dann eine Funktion.

3. Was ist die Arithmetik eines Datentyps?
   Als Arithmetik eines Datentyps bezeichnet man alle Funktionen, die für diesen Datentyp definiert sind, z.B. sind die Fkt +, -, *, /, =, … Teil der Arithmetik der Zahlen.

4. Welche eingebauten Datentypen gibt es in BSL?
   Number, String, Boolean, Symbol, Image (im Teachpack), List, Posn

5. Was sind „Magic Literals“?
   Magic Literals sind Literale im Programmiercode, die nicht durch Konstanten definiert wurden. Sie sind sehr umständlich beim Verstehen eines Codes (man weiß nicht, ob eine Zahl für z.B. eine Temperatur oder eine Zeit steht) und auch beim Ändern des Codes, weil man jedes Exemplar des Magic Literals raussuchen muss. Wenn man für ein Magic Literal eine Konstante definiert, muss man nur einmal die Konstante ändern.

6. Was versteht man unter DRY?
   Don‘t **R**epeat **Y**ourself. Ein Programmiercode sollte keine Redundanzen aufweisen, sondern solche dann in Hilfsfunktionen verpacken, die aufgerufen werden. So macht man sich das Programmieren selber und auch das Lesen und Verstehen eines Codes einfacher. Es treten weniger Fehler auf (bei korrekter Implementation der Hilfsfunktion) und auch die Wartung wird erleichtert. Zum Beachten des DRY-Prinzips gehört auch das Eliminieren von Magic-Literals.

7. Welche 2 grundsätzlichen Typen von Fehlern gibt es?
   Es gibt statische Fehler und dynamische Fehler/Laufzeitfehler. Statische Fehler treten vor, und dynamische Fehler während der Laufzeit des Programms auf.

8. Was sind Bsp für diese Typen?
   Statische Fehler sind v.a. Syntaxfehler, Laufzeitfehler sind Typfehler und Aritätsfehler und auch mathematische Fehler wie (/ 1 0).

9. Was ist das Vokabular eines Programms?
   Das Vokabular eines Programms sind alle dem Programmierer aktuell zur Verfügung stehenden Funktionen, es kann durch Funktionsdefinitionen erweitert werden.

10. Was sind primitive Funktionen?
    Primitive Funktionen sind solche, die fest in die Programmiersprache eingebaut und nicht selbst definiert sind, z.B. *(and …)* oder *(string=? ...)*

11. Wie lauten und wie unterscheiden sich die Entwurfsrezepte für allgemeine Fkt (Funktionen), Fkt mit Summentypen, Fkt mit Produkttypen, Fkt mit ADTs?
    Grundsätzlicher Aufbau:

a) Datendefinition
b) Signatur, Aufgabenbeschreibung, Header/Stub/Funktionskopf
c) Tests schreiben, noch sollten alle fehlschlagen
d) Template schreiben/einfügen
e) Funktionsbody implementieren
f) Tests durchführen, bei fehlgeschlagenen Tests entweder das Programm oder die Tests ändern.
g) Refactoring. Wurde das DRY-Prinzip beachtet? Können Vereinfachungen durchgeführt werden? Wichtig: Niemals Refactoring und Erweiterung der Funktion vermischen, regelmäßig Tests durchführen, ob das geänderte Programm noch die Anforderungen erfüllt.

Die Entwurfsrezepte sind im Grunde für alle Datentypen gleich. Es ändern sich nur Details, wie der andere Aufbau der Datendefinition oder des Templates, oder, dass bei Aufzählungstypen noch jede Alternative getestet werden sollte, bei Produkttypen nur noch, wenn das nicht in unzählige Tests ausartet.

12. Was ist die Spezifikation einer Fkt?
Als Spezifikation bezeichnet man die Signatur, die Aufgabenbeschreibung, den Namen und die Tests einer Funktion. Sie sollte ein Verständnis der Funktion der Funktion ermöglichen, ohne dass man sich mit der Implementation beschäftigen muss.

13. Welche Elemente einer Fkt gehören auf die Wunschliste?
Name, Signatur, Aufgabenbeschreibung, Header.

14. Was beschreibt das Prinzip des Information Hiding?
Der Anwender muss sich auf die Spezifikation einer Funktion verlassen können, das heißt, der Programmierer darf nicht von ihr abweichen, auch wenn er die Funktion ändert (Es gibt viele verschiedene Implementationen zu einer Spezifikation). Das ist wichtig, weil sich ja auch andere Funktionen auf die Spezifikation verlassen, wenn sie eine Funktion aufrufen.

15. Was unterscheidet Batchprogramme von interaktiven Programmen?
In Batchprogrammen startet der Anwender das Programm, dann kann er nicht mehr eingreifen. Es wird eine Funktion, ein Algorithmus durchgeführt. In interaktiven Programmen kann der Anwender während der Programmlaufzeit in dieses eingreifen und das „Ergebnis“ beeinflussen, z.B. in einem Spiel, wie z.B. Breakout.

16. Was bedeutet Konfluenz?
Konfluenz bedeutet, dass die Reihenfolge, in der in einem Programm Unterausdrücke ausgewertet werden, egal ist, weil immer dasselbe Ergebnis herauskommt. Z.B. ist der Ausdruck/die Funktion *(+ (* 3 2) (- 9 8) (/ (+ 5 3) 2))* konfluent. Die Funktion *(cond [(and #t #t) 45] [(or #f #t) 74])* ist hingegen nicht mehr konfluent.

17. Was ist eine „strikte“, was eine „nicht-strikte Funktion“?
Bei einer strikten Funktion werden zunächst alle Unterausdrücke ausgewertet, bevor die „richtige“ Funktion ausgeführt wird. Z.B. ist + eine strikte, *cond* eine nicht-strikte Funktion.

18. Was sind Summentypen, welche „Untertypen von Summentypen“ gibt es?
Summentypen sind selbst erstellte Datentypen, die aus endlich vielen Alternativen bestehen. Zu den Summentypen gehören Aufzählungstypen und Intervalltypen.

19. Wie sehen die zugehörigen Datendefinitionen aus?

Für Aufzählungstypen:	Für Intervalltypen: Konstanten verwenden
; A Signal-Colour is one of: *; - 'red* *; - 'green* *; - 'yellow* *; interp the colour of an LED* *(define RED 'red)* *(define GREEN 'green)* *(define YELLOW 'yellow)*	*; A state is a number that falls into one of the intervals:* *; - between TOP and HIGH* *; - between HIGH and MIDDLE* *; - between MIDDLE and BOTTOM* *; interp the height of an object* (Die Konstanten TOP, MIDDLE, HIGH, BOTTOM sollten schon vorher definiert worden sein.)

Eine Datendefinition geschieht immer vor einer oder mehrerer Funktionsdefinitionen. Daher bietet es sich an, direkt im Anschluss an diese das Template für den zugehörigen Datentyp zu schreiben und es auch dort stehen zu lassen. Das ist (laut Skript) nicht zwingend notwendig, macht aber Sinn. Bsp für Templates von Summentypen und Produkttypen in der nächsten Frage. Die Beispiele dienen dem Verständnis und können später in checks verwendet werden. Außerdem verhindern sie Tippfehler, etwa wenn man später im Programm 'yelow schreibt: Das ist kein Fehler für DrRacket, aber das Programm funktioniert nicht. Schreibt man stattdessen YELOW, meldet DrRacket, dass diese Konstante nicht definiert wurde.

20. Woher kommt der Name „Summentyp“, woher der Name „Produkttyp“?
Kombiniert man zwei oder mehr Summentypen zu einem neuen Summentyp, so ist die Anzahl der neuen Alternativen gleich der Summe der vorherigen Alternativen. Kombiniert man zwei oder mehr Summentypen zu einem Produkttyp (mit so vielen Feldern wie Summentypen, in jedes Feld darf nur ein Summentyp eingesetzt werden), so ist die neue Anzahl der möglichen Alternativen, bzw. die Anzahl der jetzt möglichen Kombinationen gleich dem Produkt der Alternativen der Summentypen.

Summentyp mit 2 Alternativen

```
; A SignalColour is one of
; - 'red
; - 'green
; interp. the Colour of a status-LED
(define RED 'red)
(define GREEN 'green)
(define (signal-template s)
  (cond
    [(symbol=? RED s) …]
    [(symbol=? GREEN s) ...]))
```

Summentyp mit 3 Alternativen

```
; A led is one of
; - 'led1
; - 'led2
; - 'led3
; interp. one of the three leds in an electronic circuit
(define LED1 'led1)
(define LED2 'led2)
(define LED3 'led3)
(define (led-template led)
  (cond
    [(symbol=? LED1 led) …]
    [(symbol=? LED2 led)…]
    [(symbol=? LED3 led)...]))
```

```
(define-struct Lstate (led colour))
; a state is a struct (make-state led SignalColour)
; interp. the name of the led and its actual colour)
(define STATE-1 (make-Lstate  LED1 RED))
(define STATE-2 (make-Lstate LED1 GREEN))
(define STATE-3 (make-Lstate LED2 RED))
(define STATE-4 (make-Lstate LED2 GREEN))
(define STATE-5 (make-Lstate LED3 RED))
(define STATE-6 (make-Lstate LED3 GREEN))
(define (Lstate-template state)
  (...(Lstate-led state)...(Lstate-colour state)...))
```

Produkttyp mit den beiden Summentypen als Feldern

2*3=6 mögliche Kombinationen des Produkttyps (soviele Beispiele sind hier eigtl gar nicht nötig)

21. Was sind „Vereinigungstypen“, was „union-types“, was „Itemizations“?
    Alles andere Namen für Summentypen.

22. Was bedeutet es, wenn 2 Mengen „disjunkt“ sind?
    Zwei Mengen sind disjunkt, wenn kein Element der Menge A in Menge B enthalten ist.

23. Was ist wichtig bei der Festlegung der Alternativen eines Summentyps?
    Die Mengen der möglichen Werte für eine Alternative müssen disjunkt sein.

24. Was ist eine Instanz einer Struktur?
    Eine Instanz ist ein Exemplar oder ein Beispiel einer Struktur, z.B. ist (make-posn 0 4) eine Instanz der Posn-Struktur.

25. Welche Fkt werden bei einer Strukturdefinition automatisch von DrRacket miterstellt? Wie heißen diese Fkt allgemein, wie ruft man sie in BSL auf?
    Ein Konstruktor, z.B. (make-Lstate …), ein Destruktor/Selektor, z.B. (Lstate-led …) und ein Prädikat, z.B. (Lstate? …)

26. Wie sieht eine Datendefinition für einen Produkttypen aus?
    Zuerst kommt die Definition mit (define-struct …), dann als Kommentar eine Instanz, die die Datentypen der Felder kennzeichnet und schließlich ein Kommentar, der beschreibt, was der Produkttyp bedeutet (interp. ….). Dann kommen ein paar Beispiele und idealerweise noch ein Template (siehe Bsp-Code oben).

27. Was bezeichnet man als „Tagged Union“?
    Sind mehrere Alternativen eines Datentyps nicht disjunkt, muss man den Alternativen einen „tag“ verpassen, damit sie wieder unterscheidbar sind. Z.B. :

Alternativen sind NICHT disjunkt:

```
; A Temperature is one of:
; - a number in Kelvin
; - a number in Fahrenheit
; interp a temperature value
```

(das Prädikat *(number? …)* kann nicht zwischen den Alternativen unterscheiden)

Alternativen wurden „getaggt“:

```
; A temperature is one of:
; - (make-kelvin number)
; - (make-fahrenheit number)
; interp a temperature value
```

(die Strukturen Kelvin und Fahrenheit müssten natürlich vorher definiert werden)

28. Was sind algebraische Datentypen?
    Algebraische Datentypen sind Kombinationen von Summentypen und Produkttypen.

29. Wann sind Datentypen „isomorph“?
    Datentypen sind isomorph, wenn es zwischen ihnen eine bijektive Abbildung gibt (also eine Umrechnung, die in beide Richtungen funktioniert).

30. Was definiert die Syntax einer Programmiersprache?
    Eine Syntax definiert, welche Texte Programme der jeweiligen Sprache sind.

31. Was definiert die Grammatik einer Sprache?
    Eine Grammatik definiert eine Syntax.

32. Was ist ein Terminalsymbol, was ein Nichtterminal?
    Ein Terminalsymbol ist ein Literal, das nicht weiter ausgewertet werden kann.
    Ein Nichtterminal kann durch die in seiner Produktion festgelegten Nichtterminale oder Terminalsymbole ersetzt werden, so entsteht ein Ableitungsbaum.

33. Was ist die EBNF (Erweiterte Backus-Naur-Form)?
    Die EBNF ist einfach die Notation, mit der im Skript Grammatiken beschrieben werden.

34. Was ist der Unterschied von einem ALB zu einem SALB?
    (**A**b**l**eitungs**b**aum und **S**equenz von **A**b**l**eitungs**b**äumen)
    Ein ALB kann zu genau einem Terminalsymbol abgeleitet werden. Wird gebildet, indem ein Nichtterminal durch eine seiner Alternativen ersetzt wird.
    Eine SALB beinhaltet die Verzweigungen, die entstehen, wenn <Nichtterminal>* oder <Nichtterminal>$^+$ vorkommen.

35. Was definiert die Metagrammatik?
    Die Metagrammatik definiert, wie ALB erzeugt werden können.

36. Was ist der Unterschied zwischen konkreter und abstrakter Syntax einer Sprache?
    Eine konkrete Syntax enthält die komplette Syntax der Programmiersprache, inklusive Details wie Kommentare oder Leerzeichen. Eine abstrakte Syntax enthält nur „relevante“ Aspekte der Sprache. Im Skript werden nur abstrakte Syntaxen verwendet.

37. Was enthält die (BSL-)Kernsprache im Unterschied zur gesamten Sprache?
    Die BSL-Kernsprache enthält keinen syntaktischen Zucker, wie z.B. die if-Funktion.

38. Was ist das Environment oder die Umgebung eines Programms?
    Das Environment enthält zu einem „Zeitpunkt“ während der Ausführung des Programms alle bis hierher definierten Funktionen und Konstanten. Funktionsdefinitionen werden „1:1“ in die Umgebung geschrieben, Konstantendefinitionen werden erst zu einem Wert ausgewertet und dann in die Umgebung geschrieben.

39. Was beschreibt die Auswertungsposition in einem Programm?
    Die Auswertungsposition ist der Unterausdruck, der als nächstes ausgewertet wird, bevor er als Wert mit der Kongruenzregel zurück in das Programm eingesetzt wird.

40. Was ist ein „Auswertungskontext“?
    Ein Auswertungskontext ist eine Grammatik für Programme, die ein „Loch“ „[ ]“ enthalten. Das „[ ]“ zeigt die Auswertungsposition an. Im Auswertungskontext steht also alles, das gerade nicht ausgewertet wird, und eben das „Loch“ [].

41. Wie lautet die Definition der Kongruenzregel?
    Informell: „Ein Ausdruck wird reduziert, indem ein Unterausdruck in Auswertungsposition reduziert bzw. ausgewertet wird.“
    Formell: (KONG): Falls $e_1 \rightarrow e_2$, dann $E[e_1] \rightarrow E[e_2]$

42. Was beschreibt die Auswertungsregel (PROG)?
    Programme werden von rechts nach links bzw. von oben nach unten ausgeführt; Funktions- und Strukturdefinitionen werden direkt ins Environment übernommen; Ausdrücke werden nach den Auswertungsregeln im aktuellen Kontext/Environment ausgewertet; Konstantendefinitionen werden ausgewertet und dann ins Environment übernommen.

43. Was bezeichnet man als „Datenuniversum“?
    Das Datenuniversum ist die Gesamtheit aller Werte, die potentiell vorkommen können, also gültig sind. Es kann durch Strukturdefinitionen erweitert werden. Eine Datendefinition definiert nur eine Teilmenge des Datenuniversums.

44. Was ist die Grundvoraussetzung, dass Äquivalenz gelten kann?
    Die Ausdrücke, die man auf Äquivalenz überprüft, müssen terminieren.

45. Was bezeichnet man als Selbstähnlichkeit?
    Wenn ein Teil eines Ganzen dieselbe Struktur hat, wie das Ganze selbst, spricht man von Selbstähnlichkeit.

46. Wo taucht sie in BSL, bzw. allgemein beim Programmieren, auf?
    Bei rekursiven Datentypen, also bei Listen. Das sieht man z.B. an einer Datendefinition:

```
; A List-of-Numbers is one of:
; - empty
; - (cons Number List-of-Numbers)
```

47. Wie ist ein rekursiver Datentyp aufgebaut?
    Ein rekursiver Datentyp ist ein Summentyp, mindestens eine Alternative muss immer ein nicht-rekursiver „Basisfall“ sein (hier: *empty*) und eine Alternative muss rekursiv sein, also den „übergeordneten“ Summentyp wieder enthalten (hier die 2. Alternative). So kann ein rekursiver Datentyp beliebig groß sein.

48. Auf welche 3 Arten können in BSL Listen erzeugt werden? Wie kann jeweils eine leere Liste erzeugt werden?

(cons irgendwas Liste)	*(list ...)*	*(quote (…))* bzw. *'(...)*
(cons 2 (cons 4 empty)) *empty*	*(list 2 4)* *(list)*	*(quote (2 4))* bzw. *'(2 4)* *(quote ())* bzw. *'()*

49. Was ist eine „homogene Liste“ und wie taucht sie in einer Signatur auf?
    Eine homogene Liste enthält nur Elemente eines Datentyps.
    Man schreibt *; [X] (List-of X) → X* für eine Fkt, die eine homogene Liste eines beliebigen Datentyps konsumiert und einen Wert ausgibt, der genau diesem Datentyp entspricht, z.B. die Funktion *(first …)*. X ist eine Typvariable, sie steht für einen beliebigen Datentyp, ein Beispiel für das obige Beispiel wäre also *; (List-of Number) → Number*

50. Was bedeutet Konkatenation (von Mengen oder Listen)?
    Konkatenation von Listen bedeutet das Hintereinanderhängen von 2 Listen, ohne die Reihenfolge der Elemente zu ändern. *'(4 5) '(3 2) → '(4 5 3 2)*

Bei Mengen entspricht die Konkatenation der Vereinigung der Mengen, die Reihenfolge der Elemente ist natürlich egal.

51. Welchen Unterschied haben die primitiven Prädikatsfunktionen (cons? …) und (list? …)

> (cons? empty) *#false*	*> (list? empty)* *#true*

52. Wie sollte nach Entwurfsrezept beim Schreiben einer rekursiven Fkt vorgegangen werden?
    Generell wie bei allen anderen Funktionen auch. Es sollte darauf geachtet werden, dass es eine Fallunterscheidung für den Basisfall und den rekursiven Fall gibt. Im rekursiven Fall wird die Funktion selber wieder aufgerufen. Wenn die Funktion selber wieder aufgerufen wird, so muss sie mit einer um mindestens ein Element kürzeren Liste aufgerufen werden, damit die Funktion auch terminieren kann.

53. Wie lässt sich Äquivalenz bei rekursiven Fkt nachweisen?
    Mit Beweis per Induktion. (Induktionsanfang für den Basisfall, Induktionshypothese und Induktionsschritt)

54. Was ist beim Pattern Matching die Syntax für „Prädikatsfunktionen“ wie (boolean? …)?
    (? name?) also z.B. *(? boolean?)*

55. Wie „matcht“ man auf eine Struktur, zB. (make-posn 0 4)?
    (posn 0 4) also ohne das „*make*-“

56. Als was für eine Fkt (mit welchen Ein- und Ausgabewerten) kann man sich Pattern Matching vorstellen?
    Man kann sich für ein Pattern Matching für jede Zeile „match-Funktion“ vorstellen, die als Eingabe alle den zu matchenden Wert und für jede Zeile ein spezifisches Pattern erhalten. Gibt es keinen „match“ zwischen dem Wert und dem Pattern, ist die Rückgabe „no-match“ und die nächste Zeile im Pattern Matching wird geprüft. Ist die Rückgabe „match“, wird jeder Name im Pattern mit dem entsprechenden Wert aus dem zu matchenden Eingabewert zu einer Substitution σ_j kombiniert. Alle Substitutionen σ_j werden dann zu einer Substitution σ „zusammengefasst“. Überschneiden sich Substitutionen σ_j, also gilt z.B. $\sigma_1=[x := 4]$ *und* $\sigma_2=[x := 5]$, wird die „ursprüngliche Rückgabe“ „match“ in „no-match“ geändert. Gibt es keine Überschneidungen, wird die Substitution auf die Expression in derselben Zeile angewandt, die Namen in der Expression werden durch die zugehörigen Werte ersetzt und die Expression wird ausgeführt.

57. Was passiert beim Pattern Matching, wenn der Eingabewert auf keines der Patterns „matcht“?
    Gibt es gar keinen Match, bricht das Programm mit einer Fehlermeldung ab.

58. Was ist in BSL ein Symbol?
    Ein Symbol ist ein Datentyp, der symbolische Daten repräsentiert, wie z.B. Farben. Sie werden erstellt mit einem Hochkomma/Quote. Z.B. *'das-ist-ein-Symbol*

59. Welche 4 „Regeln“ gibt es für die Quote/Quasiquote/Unquote-Operatoren?
    Ein Quote ist ein „ ' “.
    a) '*(*e_1 … e_n*)* wird transformiert zu *(list* 'e_1 … 'e_n*)*
    b) Ist *l* ein Literal, dann gilt '*l* wird transformiert zu *l*
    c) Ist *n* ein Name, dann gilt '*n* wird nicht transformiert, '*n* ist ja ein Symbol (s.o.)
    d) Ein Unquote „ , “ macht ein Quasiquote „ ` “ rückgängig, also wird ,`*e* transformiert zu *e*

Ein Quote kann hingegen nicht rückgängig gemacht werden, wie dieses Bsp zeigt:

```
> '((+ 3 4) ,(+ 5 8))                > `((+ 3 4) ,(+ 5 8))
`((+ 3 4) (,'unquote (+ 5 8)))       `((+ 3 4) 13)
```

60. Wie sind „S-Expressions“ informell definiert?
    S-Expressions sind ein „universelles Datenformat“, bei dem der Datentyp Teil eines Datums sein kann (Datum = Einzahl von Daten). So kann man auch Datentypen „taggen“ (s.o.).

61. Wie lautet die Datendefinition für S-Expressions?

```
; An S-Expression is one of:
; - a Number
; - a String
; - a Symbol
; - a Boolean
; - an Image
; - empty
; - a (list-of S-Expression)
```

62. Was ist ein „sicherer Weg“, um eine S-Expression zu erstellen?
    Mit dem Quote-Operator erstellt man immer eine S-Expression.

63. Was bedeutet „Homoikonizität“?
    Homoikonizität bedeutet, dass Programme einer Sprache Datenstrukturen derselben Sprache sind. Man kann mit dem Quote-Operator ein BSL-Programm zu einer S-Expression machen.

64. Was bedeutet „generisch“, z.B. in „generische Funktion“?
    Eine generische Funktion funktioniert für alle Datentypen.

65. Was sind „getypte Datentypen“?
    Getypte Datentypen enthalten zusätzlich zu ihrem Wert noch die Information, um was für einen Datentyp es sich handelt, so lässt sich z.B. die Fkt (string-append …) nicht auf Zahlen anwenden.

66. Werden in BSL getypte oder ungetypte Datentypen verwendet?
    In BSL werden getypte Datentypen verwendet.

67. Was zeichnet eine „ungetypte Sprache“ aus?
    In einer ungetypten Sprache kann jede Funktion auf jeden Datentyp angewendet werden, ohne, dass es zu einem Fehler kommt. Wenn man dann z.B. zwei Strings addiert, erhält man also ein gültiges (aber schwachsinniges) Ergebnis, weil die Addition auf Strings überhaupt nicht definiert ist.

68. Was zeichnet eine „dynamisch getypte Sprache“ aus?
    In einer dynamisch getypten Sprache gibt es Prädikatfunktionen (wie (number? …)), die den Typ von Daten während der Programmlaufzeit abfragen können. Es können neue Datentypen erstellt werden (z.B. mit Structs oder S-Expressions).

69. Was ist ein „Contract“ allgemein?
    Ein Contract ist eine Hilfsfunktion, die die Datentypen von Eingabe und Ausgabe einer Funktion während der Laufzeit des Programms prüfen, also wie eine auch für das Programm

verbindliche Signatur. In BSL können „Contracts“ von Hand als Hilfsfunktionen programmiert werden.

70. Was sind die Vorteile und Nachteile von Contracts, bzw. von dynamisch getypten Sprachen?
    + Fehler werden früher gefunden als ohne Contracts
    + Fehlermeldungen sind modular und können eine „schuldige“ Funktion nennen, die für den Fehler verantwortlich ist.
    - Fehler werden trotzdem erst während der Laufzeit gefunden
    - Signaturen wie [X] (List-of X) → (List-of X) sind schwierig zu implementieren
    - viele dynamische Prüfungen können die Programmlaufzeit verlängern

71. Wie kann man einen Contract in BSL „nachbauen“?
    Man kann mit Prädikatsfunktionen den Eingabewert jeder Funktion prüfen, und bei einem fehlerhaften Datentyp der Eingabe eine für jede Funktion spezifische Fehlermeldung ausgeben.

72. Was zeichnet „statisch getypte Sprachen“ aus?7
    Schon vor der Ausführung einer Funktion, schon bei ihrer Definition wird die Signatur von einem „Typchecker“ geprüft, indem man jedem Programmteil einen Ausgabetyp zuordnet, der von dem/den Eingabetypen abhängt. Z.B.: „Wenn diese Funktion eine Zahl erhält, muss sie eine Liste von Zahlen ausgeben, wenn sie einen String bekommt, dann eine Liste von Strings usw.. Der Vorteil von statisch getypten Sprachen ist, dass eventuelle Fehler schon vor der Ausführung entdeckt werden, also nicht erst beim Anwender, sondern schon beim Programmierer.

73. Was bedeutet Kompositionalität?
    Prinzip, dass die Bedeutung eines zusammengesetzten Ausdrucks von der Bedeutung seiner Unterausdrücke abhängt.

74. Was versteht man unter einem „wohlgetypten Programm“?
    Ein Programm, das den Typchecker erfolgreich durchläuft.

75. Was ist der Unterschied zwischen einem „Typchecker“ und den bisher bekannten Tests (check-expect usw.)?
    Die bekannten check-expects können nur spezifische Beispiele abdecken und nicht allgemein für einen Datentyp testen.

Nur der S04 <3