Vorgabe für den funktionalen Teil:

Scala-Funktionen sind immer gemäß der Definition im Kapitel „Typen und Funktionen", also rein-funktional zu implementieren, es sei denn diese Vorgabe ist in der Aufgabenstellung aufgehoben.

Aufgabe 1 (20 Punkte)

a) Wann nennt man eine Funktion tail-rekursiv?
b) Welcher Vorteil liegt bei tail-rekursiven Funktionen vor?
c) Geben Sie eine tail-rekursive Scala-Funktion zur Implementierung der Summe der ersten n natürlichen Zahlen an.
d) Begründen Sie, warum Ihre Implementierung aus c) tail-rekursiv ist.

Lösung:

a) Wenn der Ausdruck nur einen rekursiven Aufruf enthält, der am Ende ausgeführt wird.
b) Man benötigt einen konstanten Platzaufwand, da durch die Tail-Rekursion immer der gleiche Stack-Frame verwendet werden kann.
c)
```
def sumfirstn(n:Int) : Int = sumfirstnTR(n,0)
def sumfirstnTR(n:Int,res:Int) : Int =
if (n==0) res else sumfirstnTR(n-1,res+n)
```
d) Die Funktion aus c) ist tail-rekursiv, da beim Ausdruck mit rekursivem Aufruf der Ausdruck nur aus einem Funktionsaufruf besteht.

Aufgabe 2 (20 Punkte)

a) Beschreiben Sie, wie sich die Call-By-Value und Call-By-Name-Auswertungsstrategien unterscheiden.

Gegeben seien folgende Funktionsdefinitionen in Scala:

```
def f1(i: Int) : Int = i - 1
def f2(i: Int) : Int = f1(i-1) * i
def f3(i: Int) : Int = f2(i-1) * f3(i)
```

Führen Sie die ersten 4 Schritte der Auswertung des Ausdrucks `f3(f1(3))` durch und unterstreichen Sie bei jedem Schritt den Teilausdruck, der als nächstes ausgewertet wird. Nutzen Sie als Auswertungsstrategie

b) Call-by-Value
c) Call-By-Name

Lösung:

a) Bei Call-By-Value werden die Parameter eines Funktionsaufrufs leftmost-innermost ausgewertet und nach vollständiger Auswertung der Parameter wird der Aufruf durch die rechte Seite der Funktionsdefinition ersetzt und die formalen Parameter werden durch die ausgewerteten Werte ersetzt.
Bei Call-By-Name wird der Funktionsaufruf durch die rechte Seite ersetzt, wobei die formalen Parameter durch die Ausdrücke (noch nicht ausgewertet) in den aktuellen Parametern ersetzt werden. Die dabei erhaltenen Ausdrücke werden dann leftmost-outermost ausgewertet.

b) f3(f1(3)) ->

f3(3 – 1) ->

f3(2) ->

f2(2 – 1) * f3(2) ->

f2(1) * f3(2)

c) f3(f1(3)) ->

f2(f1(3) – 1) * f3(f1(3)) ->

f1(f1(3) – 1 – 1) * (f1(3) – 1) * f3(f1(3)) ->

(f1(3) – 1 – 1 – 1) * (f1(3) – 1) * f3(f1(3)) - >

(3 – 1 – 1 – 1 – 1) * (f1(3) – 1) * f3(f1(3))

Aufgabe 3 **(20 Punkte)**

Seien `map` und `filter` die aus der Vorlesung bekannten Funktionen auf Integer-Listen in Scala-Notation.

a) Geben Sie die Typ-Definition der Funktion `map` auf Integer-Listen in Scala-Notation an.
b) Implementieren Sie die Funktion `filter` auf Integer-Listen als Scala-Funktion.
c) Gegeben sei die Eingabeliste liste0. Konstruieren Sie aus Aufrufen von filter und map einen Ausdruck, der aus der Eingabeliste eine Liste generiert, für die Folgendes gilt: Die neue Liste enthält die Hälfte jeder geraden Zahl aus der Eingabeliste (Die ungeraden Zahlen werden verworfen)
Bsp.: Eingabe: `[1,2,3,4,7,8]`, Ergebnis: `[1,2,4]`

Lösung:

```
a) def map(list: List[Int], func: Int => Int) : List[Int]
b) def filter(list: List[Int], condition: Int => Boolean): List[Int] =
   list match {
   case x::xs => if(condition(x))        x::filter(xs, condition)
                              else       filter(xs, condition)
   case Nil => Nil
   }
c) map(filter(liste0, (x: Int) => x % 2 == 0), (x: Int) => x / 2)
```