PFP-Team
pfp@i2.cs.fau.de

Übungsblatt 10

Abgabe bis 10.07.2019, 18:00 Uhr

Hinweise:

- Verwenden Sie zur Lösung der Aufgaben nur die aus der Vorlesung bekannten, sowie die in den Übungen bekannt gegebenen Methoden und Funktionen der *Scala*-Standardbibliothek.
- Achten Sie bei der Bearbeitung der *Scala*-Aufgaben darauf, dass alle von Ihnen erstellten Methoden und Funktionen keine Seiteneffekte haben dürfen. Diese Regelung gilt sowohl für die Übungsaufgaben (Blätter 10–12) als auch für die Klausur!
- Nutzen Sie nur unveränderliche (immutable) Collection-Klassen. Die Verwendung von Klassen aus dem Paket `scala.collection.mutable` ist untersagt.
- Verwenden Sie nur das Scala-Schlüsselwort `val` zur Deklaration von Attributen und Konstanten. Die Verwendung von `var` ist untersagt.
- Geben Sie Ihre Lösungen für die Blätter 10–12 über die EST-Veranstaltung *PFP (F)* ab.

Aufgabe 10.1: Funktionen in Scala

a) Definieren Sie eine Funktion `f1` mit einem Parameter `x`, die das Vorzeichen des Parameters umkehrt.

b) Definieren Sie eine Funktion `f2` mit einem Parameter `x`, die zu einem nicht-negativen Argument (inkl. 0) 10 addiert und von negativen Argumenten 2 subtrahiert.

c) Definieren Sie eine Funktion `f3` mit zwei Parametern `x` und `y`, die ein Tupel `(y, x)` zurückliefert.

d) Definieren Sie eine Funktion `f4`, die als Parameter ein Tupel `(x, y)` entgegennimmt und das Tupel `(y, x)` zurückliefert. Was ist der Unterschied zwischen `f3` und `f4`?

Aufgabe 10.2: Listen in Scala

a) Notieren Sie in *Scala*-Syntax eine Liste mit allen natürlichen Zahlen von 4 bis 10. Gibt es mehrere Schreibweisen?

b) Welches Ergebnis liefern die Funktionen `head` und `tail` der Liste `List(4, 2, 8, 5, 8)`?

c) Welche Ergebnisse liefern die folgenden Ausdrücke?

- `List(5, 3, 6, 7, 8)(3)`
- `List(5, 3, 6, 7, 8):::(3::(5::7::78::22::Nil))`
- `List.range(3, 11).length`

Aufgabe 10.3: Mustervergleich in Scala

a) Was versteht man unter Mustervergleich in Programmiersprachen?

b) Welche Ergebnisse liefern die Funktionen `f` und `g` mit den Parametern `Nil`, `List()`, `19`, `List(4, 5, 6)`, `List.range(10, 21)`, `List(3)` und `"Hallo Welt"`?

```
def f: List[Int] => Int = {
    case h::g::ts => h * 5 * g
    case _        => -1
}

def g: List[Int] => List[Int] = {
    case Nil   => List(15)
    case _::ts => List(ts.length)
    case _     => List(20)
}
```

Aufgabe 10.4: Scala Read-Evaluate-Print-Loop (REPL)

a) Starten Sie die *Scala* Read-Eval-Print-Loop und berechnen Sie `5 + 4`.

b) Laden Sie unter Verwendung von `:load` die bereitgestellte Datei „Beispiel.scala“.

c) Rufen Sie die Funktionen `f` und `g` mit den Werten `2`, `-5`, `10` und `-20` auf. Was berechnen diese beiden Funktionen?

d) Was berechnet die Funktion `h`? Verwenden Sie dabei die Eingabewerte `List(2, 5, 1)`, `Nil` und `List.range(4, 21)`.

Hinweis: Eine Übersicht über die Befehle erhalten Sie mit `:he` (oder `:help`). Die *REPL* verlassen Sie mit `:q` (oder `:quit`).

Aufgabe 10.5: Funktionen – Vertiefung

Erstellen Sie in der Datei „Functions.scala“ ein `object Functions` und implementieren Sie darin die folgenden Funktionen:

a) `abs: Int => Int`: Berechnet den Absolutwert des übergebenen Wertes. Wenden Sie hierbei das Konzept des Mustervergleichs an.

b) `fac: Int => Int`: Berechnet die Fakultät des übergebenen Wertes.

c) `quadraticEquation: (Double, Double, Double) => List[Double]`: Berechnet die Liste reeller Ergebnisse der quadratischen Gleichung $ax^2 + bx + c = 0$. Die Funktion wird in der Form `quadraticEquation(a, b, c)` aufgerufen. Zum Berechnen der Quadratwurzel dient die vordefinierte Funktion `scala.math.sqrt: Double => Double`.

Aufgabe 10.6: Rekursion, Mustervergleich, Listen

Implementieren Sie die nachfolgend genannten Funktionen in der Datei „Recursion.scala“ im `object Recursion`. Die Implementierung soll dabei unter Verwendung von Rekursion, Mustervergleich, der leeren Liste `Nil`, dem `::`-Operator zum Voranstellen eines einzelnen Elements an eine Liste von Elementen und selbst definierten Hilfsfunktionen erfolgen. Andere Listenfunktionen, wie z.B. `:::`, `++`, `:+`, `+:`, `sum` und `length`, sind nicht erlaubt.

a) `myLength: List[Any] => Int`: Berechnet die Länge der übergebenen Liste.

b) `myConcat: (List[Any], List[Any]) => List[Any]`: Hängt die beiden übergebenen Listen aneinander.

c) `mySum: List[Int] => Int`: Summiert alle Elemente der übergebenen Liste.

Bonusaufgabe 10.7: Texte ordentlich formatieren **2 Punkte**

Gemeinsam mit einigen Kommilitoninnen und Kommilitonen haben Sie eine Vorlesungszusammenfassung für die PFP-Vorlesung geschrieben. Doch durch die vielen Mithelfenden ist der Text sehr inkonsistent formatiert. Sie beschließen, Ihre neu gewonnenen Scala-Kenntnisse zu nutzen, um die Formatierung zu vereinheitlichen. Implementieren Sie daher die folgenden Funktionen in der Datei „TextCleaner.scala“ im `object TextCleaner`. Verwenden Sie dazu nur Mustervergleich und Rekursion.

a) `contains: (List[Char], Char) => Boolean`: Als Einstieg schreiben Sie sich eine Hilfsfunktion `contains(cs, c)`, die überprüft, ob ein Zeichen `c` in einer Liste von Zeichenketten `cs` enthalten ist.

b) `clean: (List[Char], List[Char]) => List[Char]`: Die Funktion `clean(cs, ps)` erhält eine Liste von Zeichen `cs` sowie eine Liste von Satzzeichen `ps` und soll eine „bereinigte“ Form von `cs` zurückgeben. Dazu wendet sie folgende Regeln an:

- Mehrfache Leerzeichen sollen zu einem Leerzeichen verschmolzen werden.
- Leerzeichen vor Satzzeichen (alle Zeichen aus `ps`) sollen entfernt werden (Tipp: Prüfen Sie mit der vorher definierten Funktion `contains`, ob ein Zeichen ein Satzzeichen ist).

Beispiel:

```
clean("Guten␣␣Tag␣!".toList, List('!'))
 == List('G','u','t','e','n','␣','T','a','g','!')
```

Bonusaufgabe 10.8: Sortieren durch Verschmelzen **4 Punkte**

Implementieren Sie in der Datei „MergeSort.scala“ im `object MergeSort` die nachfolgend genannten Funktionen, die den Merge-Sort-Algorithmus umsetzen. Der Sortieralgorithmus folgt dem Teile-und-Herrsche-Prinzip (divide and conquer). Er erhält eine unsortierte Liste mit Zahlen, sortiert diese in aufsteigender Reihenfolge und gibt die sortierte Liste zurück. Die Signatur der Sortierfunktion lautet: `mergeSort: List[Int] => List[Int]`.

a) `divide: List[Int] => (List[Int], List[Int])`: Nimmt eine Liste entgegen und spaltet diese in der Mitte in zwei Teillisten auf. Verwenden Sie zur Implementierung die Listen-Funktionen `length`, `take` und `drop`.

b) `conquer: ((List[Int], List[Int])) => (List[Int], List[Int])`: Nimmt als Parameter ein durch `divide` erzeugtes Listenpaar entgegen und ruft für jede Teilliste `mergeSort` (Rekursion!) auf. Das Ergebnis ist ein Listenpaar mit den sortierten Teillisten.

c) `merge: ((List[Int], List[Int])) => List[Int]`: Verschmilzt die beiden als Listenpaar übergebenen Listen zu einer Ergebnisliste, die alle Elemente beider Teillisten in aufsteigender Reihenfolge sortiert enthält.

d) `mergeSort: List[Int] => List[Int]`: Falls die übergebene Liste weniger als 2 Elemente enthält, wird sie unverändert zurückgegeben. In allen anderen Fällen wird die übergebene Liste mittels der Funktion `divide` zerteilt, deren Ergebnis an die Funktion `conquer` übergeben und abschließend deren Ergebnis an die Funktion `merge` übergeben. Das Ergebnis von `mergeSort` ist dann eine vollständig sortierte Liste.

6 Punkte