Weitere Programmiersprache
- Serie 1 -

Aufgabe 1-1 (GHCI)

Der Interpreter enthält einige nützliche Funktionen, die nicht Teil der Sprache Haskell sind, aber beim Programmieren in Haskell helfen. Diese Funktionen fangen mit einem Doppelpunkt an.

1. Öffnen Sie GHCi
2. Geben Sie folgendes ein
   ```
   :t 1
   :t Bool
   :t (+)
   ```
   Was bewirkt das `:t`? Erklären Sie die Ausgaben.
3. Speichern Sie eine beliebige Haskell-Funktion in einer Datei mit dem Dateitypen `*.hs`
4. Ändern Sie das Working-Directory des GHCi mit `:cd [PFAD]` zu dem Pfad, an dem Sie die Datei in Schritt 3 gespeichert haben.
5. Finden Sie mit `:?` heraus, wie Sie die Datei mit ihrer Funktion in den GHCi laden können.
6. Verwenden Sie die geladene Funktion im GHCi.

Aufgabe 1-2 (Fehler)

Bestimmen Sie die Ursache der Fehler/falschen Ergebnisse folgender Ausdrücke und geben Sie, falls möglich, einen syntaktisch korrekten Ausdruck an, der das wahrscheinlich gewollte Ergebnis produziert.

1. `ghci> 4 +- 2`
2. `ghci> 4 -- 2`
3. `ghci> 4.0 ^^ 2.0`
4. `ghci> 4 mod 2`
5. `ghci> 9 * 'c'`
6. `ghci> [9 , 'c']`
7. `ghci> 4 ^ sqrt(-0.25)`
8. `ghci> mod pi 8`

Aufgabe 1-3 (Funktionen)

Klassische mathematische Operatoren (*, +, -, ^) werden in Haskell in der Infix-Notation verwendet:

1. Werten Sie folgende Ausdrücke aus:
   a. `ghci> 3 * 2`
   b. `ghci> 10 / 4`
   c. `ghci> 2 ^ 5`
   d. `ghci> (100 + 40) * 3`
2. Standardmäßig wird sonst in Haskell die Prefix-Notation verwendet, bei der der Operator vor den Operanden steht. Jede Infix-Funktion kann durch Klammern in der Prefix-Notation verwendet werden, z.B.:
   ```
   ghci> (-) 12 3
   ghci> 9
   ```
   Schreiben Sie die Ausdrücke aus 1. in Prefix-Notation um.
3. Probieren Sie folgende Ausdrücke aus:
   a. `ghci> mod 10 3`

Besprechung: 02.10./07.10.

```
b. ghci> multiply a b = a * b
c. ghci> multiply 5 4
d. ghci> 10 `mod` 3
e. ghci> 5 `multiply` 4
```

Welchen Effekt haben die Häkchen ` … ` ?

4. Warum gibt es einen Fehler bei der Ausführung folgender Code-Zeilen?

```
ghci> multiplyTuple (a, b) = a * b
ghci> multiplyTuple (5, 4)
ghci> multiplyTuple 5 4
```

5. Schreiben Sie eine Funktion, die drei Zahlen addiert. Schreiben Sie diese Funktion als:
   1. `add1` : Normale Eingabe der Zahlen (wie bei `multiply`)
   2. `add2` : Eingabe der Zahlen als Tupel (wie bei `multiplyTuple`)
   3. `add3` : anonyme Funktion

   Geben Sie den Typen der Funktionen an.
   Können die Funktionen in Infix-Notation verwendet werden? Warum / warum nicht?

Aufgabe 1-4 (Nachfolger)

Schreiben Sie die Funktion `successor :: Int -> Int` , welche die Eingabe um 1 erhöht. Geben Sie zwei Eingaben unterschiedlichen Typs an, für die die Funktion nicht definiert ist. Was sagt die Fehlermeldung aus?

Aufgabe 1-5 (Gesetz von De-Morgan)

Beweisen Sie das De-Morgan'sche Gesetz ¬ (a∧b) ≡ ¬ a ∨ ¬ b, indem Sie eine passende Wahrheitstabelle aufstellen, die Funktion `proofMorgan :: Bool -> Bool -> Bool` schreiben, die das Gesetz enthält und anschließend die Wahrheitstabelle durch Aufruf der Funktion mit booleschen Werten füllen.

Besprechung: 02.10./07.10.