Tag 1

Haskell als Taschenrechner

Die beiden interaktiven Haskell-Umgebungen GHCi und Hugs ermöglichen das einfache Experimentieren mit Haskell-Expressions.
Diese werden durch Eingabe von `ghci` bzw. `hugs` auf der Kommandozeile gestartet. Nach einigen einleitenden Nachrichten melden beide Programme schließlich ihre Bereitschaft durch den Prompt

```
Prelude>
```

Der Text vor dem > bezeichnet das gegenwärtig geladene Modul. Die sogenannte `Prelude` stellt dabei eine Sammlung von häufig gebrauchten grundlegenden Funktionen zur Verfügung.
Befehle, die sich an den Interpreter richten, also keine Haskell-Ausdrücke darstellen, werden mit einem Doppelpunkt `:` eingeleitet. So kann man mit

```
Prelude> :quit
```

oder auch kürzer

```
Prelude> :q
```

die Umgebung wieder verlassen. Bevor wir beginnen, nehmen wir noch eine Einstellung vor, und zwar mit

```
Prelude> :set +t
```

Dies veranlaßt den Interpreter, nach jeder Auswertung auch eine Typinformation mit auszugeben. Wir werden sofort sehen, was es damit auf sich hat:

```
Prelude> 2
```

erzeugt die Ausgabe

```
2
it :: Integer
```

im GHCi. Im Hugs sieht die Ausgabe geringfügig anders aus:

```
2 :: Integer
```

Beides bedeutet, daß `2` ein Ausdruck ist, der zu `2` ausgewertet wurde und vom Typ `Integer` ist, also eine ganze Zahl. Im GHCi kann man den zuletzt ausgewerteten Ausdruck mit `it` erneut

referenzieren, daher erscheint das `it` auch in der Ausgabe. Hugs bietet die gleiche Funktionalität mittels `$$`.

```
[GHCi] Prelude> it
2
it :: Integer

[Hugs] Prelude> $$
2 :: Integer
```

Künftig werden wir hier die GHCi-Variante der Ausgabe verwenden.
Wir machen weiter mit

```
Prelude> 42.0001
42.0001
it :: Double
Prelude> 'c'
'c'
it :: Char
Prelude> "Hallo"
"Hallo"
it :: [Char]
```

und stellen fest, daß Haskell offenbar mehr Typen als `Integer` alleine kennt. Fließkommazahlen sind ebenso möglich wie einzelne Zeichen und Zeichenketten (Strings). Strings in Haskell werden implementiert als Listen von einzelnen Zeichen. Daher die eckigen Klammern in `[Char]`, man lese also „Liste von Char“.
Haskell, genauer die Prelude, stellt eine ganze Reihe von Funktionen zur Verfügung, die es einem erlauben, die üblichen Dinge mit GHCi oder Hugs zu erledigen, die man normalerweise mit einem Taschenrechner erledigt. Dabei zeigt sich sehr schnell eine große Stärke: der Integer-Datentyp ist nicht größenbegrenzt.
Wir brauchen uns also vor einer Anweisung wie

```
Prelude> 42^42
```

nicht zu scheuen. Die 42ste Potenz von 42 wird ohne zu Murren ermittelt und ist, siehe da, immer noch vom Typ `Integer`.
Haskell kennt auch die Operatoren `+`, `*`, `-`, `/` für die Grundrechenoperationen.

Aufgabe 1

Was ergeben die Eingaben

```
Prelude> 42-6*9
Prelude> 42*6-9
Prelude> 42-(6*9)
```

Was läßt sich durch die Ergebnisse über die Prioritäten von Operatoren in Haskell ableiten?

Wie an dem dritten Beispiel aus Aufgabe 1 absehbar, erlaubt Haskell auch die Klammerung von Ausdrücken. Auch

```
Prelude> (42)
42
it :: Integer
```

ist möglich. Natürlich fallen die Klammern im Ergebnis weg, der Ausdruck `(42)` wird zu `42` evaluiert.

Aufgabe 2

Was ergeben die Eingaben

```
Prelude> 2.1+2.5
Prelude> 2+2.5
Prelude> 2+'c'
```

Die genaue Struktur der dabei auftretenden Fehlermeldung ist derzeit noch nicht relevant. Wichtig ist die Erkenntnis, daß Haskell bestimmte Grundannahmen über die Eingabe macht und versucht, das Geschriebene sinnvoll zu interpretieren. Ist das nicht möglich, wird ein Fehler erzeugt.

Aufgabe 3

Verwende den Haskell-Interpreter Deiner Wahl für ein paar einfache Kalkulationen. Probiere auch andere Funktionen aus, etwa

```
Prelude> pi
Prelude> sin pi
Prelude> abs (-2)
```

oder weitere, die Dir in den Sinn kommen. Vielleicht gibt es sie ja in der Prelude.

An der Funktionsapplikation `sin pi` läßt sich erkennen, daß Haskell normalerweise keine Klammern bei Funktionsanwendungen erfordert. Bei `abs (-2)` werden die Klammern notwendig, um das unäre Minus von der normalen Subtraktion zu unterscheiden. Probiere auch

```
Prelude> abs -2
```

aus, was wieder eine verwirrende Fehlermeldung zur Folge haben wird.

Lernziele Tag 1

- ☐ Den Haskell-Interpreter starten und verlassen.
- ☐ Einfache Taschenrechner-Kalkulationen mit Haskell ausführen.