1. Einfache Haskell-Funktion

Im Folgenden soll eine Funktion programmiert werden, die die Wegstrecke $s(t)$ berechnet, welche im freien Fall unter Einfluss der Erdbeschleunigung $g = 9.81$ nach einer Zeit t zurückgelegt wurde. Verwenden Sie die Formel

$$s(t) = \frac{1}{2} \cdot gt^2.$$

Die Wegstrecke soll in Metern und die Zeit in Sekunden angegeben werden.

i).

Mit welche Datentypen sollten die physikalischen Größen Weg, Zeit und Beschleunigung repräsentiert werden? Geben Sie den daraus resultierenden Typ einer Funktion `fallstrecke` an, die $s(t)$ aus t berechnet.

ii).

Schreiben Sie die Typsignatur `fallstrecke :: Typ` sowie die Definition von `fallstrecke` in eine Datei. Laden Sie diese in GHCi und testen Sie Ihre Funktion. Zum Beispiel sollte sich `fallstrecke 1 = 4.905` ergeben.

iii).

Verändern Sie Ihr Programm zunächst so, dass g als globale Variable (Konstante) deklariert ist. Ersetzen Sie dann die globale Deklaration durch lokale Deklarationen, zunächst mit `where` und anschließend mit `let`.

iv).

Die Eingabe eines negativen Wertes soll zu einem undefinierten Funktionswert und zu einer Fehlermeldung führen. Dies erreichen Sie mit Hilfe der Funktion `error`, z.B. durch `error "negative Zeit"`. Realisieren Sie dieses Verhalten einmal mit **If-Then-Else-Ausdrücken** und einmal mit **Guards**.

2. Einfache Haskell-Funktionen

Schreiben Sie Haskell-Funktionen zur Berechnung von Volumen, Oberflächeninhalt und Radius von Kugeln. Beachten Sie, dass einige dieser Funktionen als Komposition der übrigen aufgefasst werden können. Verwenden Sie die vordefinierte Konstante `pi::Double`. Hilfreiche Funktionen über reellen Zahlen finden Sie in den Hinweisen dieses Aufgabenblattes.

i).

Programmieren Sie die folgenden Funktionen:

- `radiusNachOberflaeche :: Double -> Double`
- `oberflaecheNachRadius :: Double -> Double`
- `radiusNachVolumen :: Double -> Double`
- `volumenNachRadius :: Double -> Double`

- `oberflaecheNachVolumen :: Double -> Double`
- `volumenNachOberflaeche :: Double -> Double`

ii).

Da es keine Kugeln mit negativem Volumen, Oberflächeninhalt oder Radius gibt, sollen die Funktionen für negative Eingaben undefiniert bleiben und mit einer Fehlermeldung abbrechen. Erweitern Sie dazu die Funktionen um sinnvolle Fallunterscheidungen und Fehlermeldungen.

3. Bedingte Ausdrücke

Schreiben Sie eine Funktion vom Typ `Integer -> Bool`, die zu einer gegebenen Jahreszahl prüft, ob sie im Gregorianischen Kalender ein Schaltjahr ist oder nicht.

Pseudocode:

```
if year is >= 1582 then
  if year is divisible by 400 then
    is_leap_year
  else if year is divisible by 100 then
    not_leap_year
  else if year is divisible by 4 then
    is_leap_year
  else
    not_leap_year
else
  not_leap_year
```

i).

Verwenden Sie hierfür nur If-Then-Else-Ausdrücke.

ii).

Verwenden Sie hierfür nur Guards.

iii).

Verwenden Sie hierfür weder If-Then-Else-Ausdrücke noch Guards, sondern die booleschen Operatoren.

4. Lokale Deklarationen

i).

Zur Bestimmung der Nullstellen von quadratischen Gleichungen der Form $ax^2 + bx + c$ gibt es neben der bekannten „p-q-Formel“ (mit $p = \frac{b}{a}$ und $q = \frac{c}{a}$) auch die sogenannte „a-b-c-Formel“

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4\,ac}}{2a}$$

Setzen Sie diese Berechnung in Haskell zunächst mit **let-in** und dann mit **where** um. Dabei sollen x_1 und x_2 als 2-Tupel zurückgegeben werden.

ii).

Seit der Veröffentlichung der Gaußschen Wochentagesformel haben verschiedene Mathematiker und Astronomen eine ganze Reihe vergleichbarer Formeln aufgestellt. Diese benötigen meist vorausberechnete Tabellen und verwenden komplizierte Sonderregeln, z.B. für Schaltjahre. Eine im Vergleich dazu sehr elegante Formel wurde 1972 von J. D. Robertson vorgestellt. Diese wird im Folgenden beschrieben. Gegeben sei ein beliebiges Datum (T, M, J) im Gregorianischen Kalender, wobei T den Tag des Monats beginnend mit 1, M den Monat des Jahres beginnend mit 1 und J das Jahr bezeichnet. Dann lässt sich der Wochentag W mit $0 \equiv$ Sonntag, $1 \equiv$ Montag, $\ldots$, $6 \equiv$ Samstag folgendermaßen berechnen:

$$W = \left(D + T + 77 + E + \left\lfloor \frac{B}{400} \right\rfloor - 2\left\lfloor \frac{B}{100} \right\rfloor\right) \ mod\ 7$$

Dabei gelte Folgendes:

$$A = M + 10$$
$$B = \left\lfloor \frac{M - 14}{12} \right\rfloor + J$$
$$C = A - 12\left\lfloor \frac{A}{13} \right\rfloor$$
$$D = \left\lfloor \frac{13 \cdot C - 1}{5} \right\rfloor$$
$$E = \left\lfloor \frac{5(B \ mod\ 100)}{4} \right\rfloor$$
$$\lfloor x \rceil = \begin{cases} \lfloor x \rfloor & x \geq 0 \\ \lceil x \rceil & x < 0 \end{cases}$$
$$\lfloor x \rfloor = max\{z \in \mathbb{Z} | z \leq x\}$$
$$\lceil x \rceil = min\{z \in \mathbb{Z} | z \geq x\}$$

Schreiben Sie eine Haskell-Funktion `robertson :: Integer -> Integer -> Integer -> Integer`, die gemäß der obigen Formel den Wochentag berechnet. Sie können sich mit der Hilfsfunktion *wochentag* den Wochentag als Text anzeigen lassen. Sie erhalten z.B. folgende Ausgaben:

- `wochentag 19 6 1623` ⟶ `"Montag"`
- `wochentag 19 1 2038` ⟶ `"Dienstag"`
- `wochentag 30 4 1777` ⟶ `"Mittwoch"`

5. Logische Operatoren, Infix-Notation

Die booleschen Funktionen und sowie nicht seien wie folgt definiert:

```
und :: Bool -> Bool -> Bool
und = (&&)

nicht :: Bool -> Bool
nicht = not
```

Programmieren Sie die zweistelligen booleschen Funktionen `oder`, `darausFolgt`, `genauDannWenn` und `entwederOder` unter ausschließlicher Verwendung von `und` sowie `nicht`. Testen Sie Ihre Funktionen.

6. Rekursion über den natürlichen Zahlen

In den folgenden Teilaufgaben dürfen keine anderen arithmetischen Operationen als die beiden vordefinierten Funktionen `succ` und `pred` verwendet werden.

i).

Programmieren Sie eine Funktion `plus :: Integer -> Integer -> Integer`, welche Summen natürlicher Zahlen berechnet. Nutzen Sie aus, dass `a + b = succ(pred a + b)` und `0 + a = a` gilt.

ii).

Programmieren Sie eine Funktion `mal :: Integer -> Integer -> Integer`, welche Produkte natürlicher Zahlen berechnet. Nutzen Sie aus, dass `a * b = b + (pred a * b)` und `0 * a = 0` bzw. `1 * a = a` gilt.

iii).

Programmieren Sie eine Funktion `hoch :: Integer -> Integer -> Integer`, welche Potenzen natürlicher Zahlen berechnet.

iv).

Die Reihe `plus`, `mal`, `hoch` kann beliebig fortgesetzt werden. Programmieren Sie eine Funktion `meta :: Integer -> Integer -> Integer -> Integer`, sodass `meta 0 a b` der Summe `plus a b`, `meta 1 a b` dem Produkt `mal a b`, `meta 2 a b` der Potenz `hoch a b` usw. entspricht. Was berechnet `meta 3 a b`?