Funktionen für ein frei erfundenes Kartenspiel (Solitär)

(Prüfungsvorleistung)

- Das Spiel beinhaltet einen *Kartenstapel* und einen *Zielwert*.
- Die Spielerin besitzt eine Liste *gezogener Karten*, die zu Beginn des Spiels leer ist.
- Sie führt einen *Zug* aus, indem sie entweder
  - die oberste Karte vom Kartenstapel *zieht* und zur Liste gezogener Karten hinzufügt oder
  - eine Karte aus Liste gezogener Karten *ablegt*, d. h. aus der Liste gezogener Karten entfernt.
- Das Spiel ist beendet, wenn entweder
  - die Spielerin entscheidet, keinen weiteren Zug auszuführen oder
  - die Summe der Werte der gezogenen Karten den Zielwert überschreitet.
- Das Ziel des Spiels besteht darin, es mit einem möglichst niedrigen *Punktestand* (0 ist das Optimum.)zu beenden.
- Der Punktestand wird wie folgt ermittelt: Sei s die Summe der Kartenwerte der gezogenen Karten. Dann gilt:

  $$vorlaeufigerPunktestand = \begin{cases} s > Zielwert & 3 \cdot (s - Zielwert) \\ s \leq Zielwert & Zielwert - s \end{cases}$$

  Der Punktestand ist der $vorlaeufigerPunktestand$, es sei denn, alle gezogenen Karte haben die gleiche Farbe. In diesem Fall ist der Punktestand gleich $\frac{vorlaeufigerPunktestand}{2}$, wobei hier ganzzahlig dividiert wird (Operator `div` in ML).

Ihre Lösungen müssen Mustervergleich benutzen. Die Nutzung der Standard-Funktionen `null`, `hd`, `tl`, `isSome` und `valOf` ist nicht erlaubt. Auch ist nichts erlaubt, was das Zeichen # benutzt.

Die Lösung benötigt ingesamt ca. hundert Code-Zeilen.

Benutzen Sie die folgenden Datentypen:

```
(* Sie dürfen davon ausgehen, dass für Zahl nur die Werte
   2, 3, ..., 10 benutzt werden. *)
datatype sorte = Kreuz | Pik | Herz | Karo
datatype wert = Bube | Dame | Koenig | Ass | Zahl of int
type karte = sorte * wert

datatype farbe = Rot | Schwarz
datatype zug = Ablegen of karte | Ziehen

exception IllegalerZug
```

Weitere Typdefinitionen sind nicht erforderlich.

Schreiben Sie nun die folgenden Funktionen:

`kartenfarbe` nimmt ein Exemplar von `karte` und liefert seine Farbe (Karo und Herz sind rot, Pik und Kreuz sind schwarz)

`kartenwert` nimmt ein Exemplar von `karte` und liefert seinen Wert (numerierte Karten haben die Zahl als Wert, Asse zählen 11, alle anderen 10)

`entferne_karte` nimmt eine Liste von Karten `kn`, eine Karte `k` sowie eine Exception `e` und liefert die Liste `kn` ohne `k`. Falls `k` mehrfach in `kn` vorkommt, wird nur das erste Exemplar entfernt. Wenn `k` in `kn` nicht vorkommt, soll die Exception `e` erzeugt werden. Karten können mit dem Operator `=` verglichen werden.

`alle_farben_gleich` nimmt eine Liste von Karten und liefert `true`, wenn alle Karten in der Liste die gleiche Farbe haben

`kartensumme` nimmt eine Liste von Karten und summiert ihre Werte. Benutzen Sie eine endrekursive Hilfsfunktion.

`punktestand` nimmt eine Liste von Karten (die gezogenen Karten) und berechnet den Punktestand nach oben angegebener Formel

`spielablauf` nimmt eine Liste von Karten (der Kartenstapel), eine Liste von Zügen, die die Spielerin der Reihe nach abarbeitet, sowie einen Zielwert und liefert den Punktestand am Ende des Spiels. Benutzen Sie eine lokal definierte Hilfsfunktion, deren Argumente den aktuellen Spielstand repräsentieren.

Benutzen Sie (mindestens) die folgenden Tests:

```
val test1 = kartenfarbe (Kreuz, Zahl 2) = Schwarz
val test2 = kartenwert (Kreuz, Zahl 2) = 2
val test3 = entferne_karte ([(Herz, Ass)], (Herz, Ass), IllegalerZug) = []
val test4 = alle_farben_gleich [(Herz, Ass), (Herz, Ass)] = true
val test5 = kartensumme [(Kreuz, Zahl 2),(Kreuz, Zahl 2)] = 4
val test6 = punktestand ([(Herz, Zahl 2),(Kreuz, Zahl 4)],10) = 4
val test7 = spielablauf ([(Herz, Zahl 2),(Kreuz, Zahl 4)],[Ziehen], 15) = 6
val test8 = spielablauf ([(Kreuz,Ass),(Pik,Ass),(Kreuz,Ass),(Pik,Ass)],
                         [Ziehen,Ziehen,Ziehen,Ziehen,Ziehen],
                          42)
            = 3
val test9 = ((spielablauf([(Kreuz,Bube),(Pik,Zahl(8))],
                          [Ziehen,Ablegen(Herz,Bube)],
                           42);
              false)
             handle IllegalerZug => true)
```