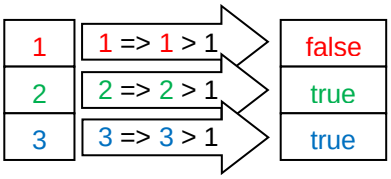
# Übung 9: Filter, Map and Reduce – Parallel List Element Processing

In dieser Übung implementieren Sie elementare Funktionen zur parallelen Prozessierung von Listen. Informieren Sie sich über die Scala List [1] und Pattern Matching [2].

Wir betrachten nun die map Methode [3]. Die map Methode nimmt zwei Argumente. Eine List[A] und eine Funktion A => B.
Die map Methode wendet nun für jedes Element der Liste (vom Typ A) die Funktion A => B an. Das Resultat ist dann eine List[B] mit derselben Länge wie die ursprüngliche Liste.
**Beispiel:**

| Input: List[Int] | Input: Int => Boolean | Output: List[Boolean] |
|---|---|---|
| List(1,2,3) | i => i > 1 | List(false,true,true) |




Die Signatur der Methode könnte so ausschauen:

```
def map[A, B](l: List[A], f: A => B): List[B]
```

map könnte dann so verwendet werden:

```
scala> map(List(1,2,3), (i:Int) => i > 1)
res5: List[Boolean] = List(false, true, true)
```

Und hier ist eine sequentielle, rekursive Implementierung der map Methode.

```
def map[A, B](l: List[A], f: A => B): List[B] = l match {
  case Nil     => Nil
  case x :: xs => f(x) :: map(xs,f)
}
```

Auf der Input Liste l wird ein Pattern-match ausgeführt und es werden zwei Fälle unterschieden:

1. Die Liste ist leer: Das Resultat ist entsprechend die leere Liste (Sie haben keine andere Wahl)
2. Die Liste hat mindestens ein Element x vom Typ A: Die Funktion f wird angewendet auf das Element x und es entsteht ein Element vom Typ B. Dieses neue Element wird der Kopf der Resultat Liste, die durch die rekursive Anwendung der map Funktion mit dem Rest xs entsteht.

**Aufgaben:**

1. Wie Sie der kleinen Illustration oben rechts entnehmen können, lassen sich die Element der Liste *parallel* verarbeiten. Implementieren Sie nun die Methode parMap. Sie hat die gleiche Signatur wie die map Funktion, verwendet aber einen ThreadPool (ExecuterService) um die Elemente zu prozessieren[1].

2. Implementieren Sie eine sequentielle und eine parallele Funktion mit folgender Signatur:
   ```
   def filter[A](l: List[A], p: A => Boolean): List[A]
   ```
   Die Idee ist, dass mit der Funktion p für jedes Element der Input Liste l geprüft wird, ob es Bestandteil der Output Liste sein soll.

3. (Optional) Implementieren Sie eine sequentielle und eine parallele Funktion mit folgender Signatur:
   ```
   def reduce[A](l: List[A], f: (A, A) => A): A
   ```
   Die Input Liste l muss mindestens ein Element beinhalten, sonst wird eine Exception geworfen. Welche Eigenschaft muss die Funktion f haben, dass eine parallele Implementierung dieselben Resultate liefert wie die sequentielle Implementierung?

[1] https://www.scala-lang.org/api/current/scala/collection/immutable/List.html
[2] http://docs.scala-lang.org/tutorials/tour/pattern-matching.html
[3] http://en.wikipedia.org/wiki/Map_(higher-order_function)

[1] Für das obige Beispiel mit der Funktion i => i > 1 lohnt sich die parallele Ausführung nicht.