

Fortgeschrittene funktionale Programmierung in Haskell

Übungszettel 2

Aufgabe 2.1:

In der Vorlesung wurde neben dem Datentypen Maybe auch Either vorgestellt. Zur Erinnerung: Either ist definiert als

```
data Either a b = Left a
                | Right b
```

Erstellen sie hierzu die Instanzen für

- Functor

```
class Functor f where
    fmap :: (a -> b) -> f a -> f b
```

- Applicative

```
class Functor f => Applicative f where
    pure  :: a -> f a
    (<*>) :: f (a -> b) -> f a -> f b
```

- Monad

```
class Applicative m => Monad m where
    return :: a -> m a
    (>>=)  :: m a -> (a -> m b) -> m b
```

Aufgabe 2.2:

Ein weiterer einfacher Datentyp ist Identity, welcher der Datentyp ist, der keinen Effekt hat. Definiert ist Identity als

```
newtype Identity a = Identity {runIdentity :: a}
```

Dieses definiert 2 Funktionen: Eine, um eine Identity zu kreieren und eine um wieder an ihren Inhalt zu kommen:

```
Identity    :: a -> Identity a
runIdentity :: Identity a -> a
```

Erstellen sie hier ebenfalls Instanzen für Functor, Applicative und Monad **ohne** durch Pattern-matching an den Inhalt zu kommen. (Nutzen sie hierfür die Funktion runIdentity).

Aufgabe 2.3:

In der Vorlesung wurde ebenfalls die State-Monade besprochen. Die Definition ist wie folgt:

```
newtype State s a = State {runState :: s -> (a,s)}
```

Dieses definiert ebenfalls 2 Funktionen:

```
State    :: (s -> (a,s)) -> State s a
runState :: State s a -> s -> (a,s)
```

Erstellen sie hier ebenfalls die Instanzen:

```
Functor (State s)
Applicative (State s)
Monad (State s)
```