Appendix A: Essential Functions Support

In this appendix, you'll find some basic JavaScript implementations of various functions described in the book. Keep in mind that these implementations may not be the fastest or the most efficient implementation out there; they *solely serve an educational purpose*.

In order to find functions that are more production-ready, have a peek at ramda, lodash, or folktale.

Note that some functions also refer to algebraic structures defined in the Appendix B

always

```
// always :: a -> b -> a
const always = curry((a, b) => a);
```

compose

```
// compose :: ((a -> b), (b -> c),  ..., (y -> z)) -> a -> z
const compose = (...fns) => (...args) => fns.reduceRight((res, fn) => [fn.call(null
```

curry

```
// curry :: ((a, b, ...) -> c) -> a -> b -> ... -> c
const curry = (fn) => {
  const arity = fn.length;

  return function $curry(...args) {
    if (args.length < arity) {
      return $curry.bind(null, ...args);
    }

    return fn.call(null, ...args);
  };
};
```

either

```
// either :: (a -> c) -> (b -> c) -> Either a b -> c
const either = curry((f, g, e) => {
  if (e.isLeft) {
    return f(e.$value);
  }

  return g(e.$value);
});
```

identity

```
// identity :: x -> x
const identity = x => x;
```

inspect

```
// inspect :: a -> String
const inspect = (x) => {
  if (x && typeof x.inspect === 'function') {
    return x.inspect();
  }

  function inspectFn(f) {
    return f.name ? f.name : f.toString();
  }

  function inspectTerm(t) {
    switch (typeof t) {
      case 'string':
        return `'${t}'`;
      case 'object': {
        const ts = Object.keys(t).map(k => [k, inspect(t[k])]);
        return `{${ts.map(kv => kv.join(': ')).join(', ')}}`;
      }
      default:
        return String(t);
    }
  }
```

```
  function inspectArgs(args) {
    return Array.isArray(args) ? `[${args.map(inspect).join(', ')}]` : inspectTerm(
  }

  return (typeof x === 'function') ? inspectFn(x) : inspectArgs(x);
};
```

left

```
// left :: a -> Either a b
const left = a => new Left(a);
```

liftA*

```
// liftA2 :: (Applicative f) => (a1 -> a2 -> b) -> f a1 -> f a2 -> f b
const liftA2 = curry((fn, a1, a2) => a1.map(fn).ap(a2));
```

```
// liftA3 :: (Applicative f) => (a1 -> a2 -> a3 -> b) -> f a1 -> f a2 -> f a3 -> f
const liftA3 = curry((fn, a1, a2, a3) => a1.map(fn).ap(a2).ap(a3));
```

maybe

```
// maybe :: b -> (a -> b) -> Maybe a -> b
const maybe = curry((v, f, m) => {
  if (m.isNothing) {
    return v;
  }

  return f(m.$value);
});
```

nothing

```
// nothing :: () -> Maybe a
const nothing = () => Maybe.of(null);
```

reject

```
// reject :: a -> Task a b
const reject = a => Task.rejected(a);
```