Functions as Values

In this lesson, we'll further explore the concept of functions being values.

In Reason, functions are known as *first-class* values, which means that they can be treated like any other value obtained from an expression.

Functions as Tuple Components

Here's a simple example of functions being used as tuple components. For learning purposes, these functions are fairly simple, but this doesn't mean that we can't use much more complex functions in such a scenario:

```
let sum = (x, y) => x + y;
let concat = (a, b) => a ++ b;
let x = 10;
let y = 4;
let a = "Hello ";
let b = "World";

let t = (sum(x, y), concat(a, b));

Js.log(t);
```

Functions as Field Values in Records

Any field in a record can have a value which is calculated from a function.

Let’s look at the `rectangle` record as an example:

```
let area = (length, width) => length *. width;

type shape = {
  length: float,
  width: float,
  area: float
};

let length = 50.5;
let width = 10.5;

let rectangle = {
  length,
  width,
  area: area(length, width)
};

Js.log(rectangle.area);
```

Alternatively, we could define the `area()` method inside the record type without giving it a name:

```
type shape = {
  length: float,
  width: float,
  area: (float, float) => float /* Define the argument and return types for the function */
};

let length = 50.5;
let width = 10.5;

let rectangle = {
  length,
  width,
  /* Give the function definition for this function */
  area: (length, width) => length *. width
};

Js.log(rectangle.area(length, width));
```

Note that the `area` field is of the function type. This means we have to pass arguments to it in order to get a value.

Functions and Arrays

Functions and Arrays

Functions can be used as elements in an array. However, a more relevant use of functions is that they can be applied on all the elements of an array.

In the Arrays lesson, we used the `Array.init()` method which required a function that will apply itself to all the elements in the array for values `0` to `n-1`, where `n` is the other argument:

```
/* Triple all the values from 0 to n-1 in an array */
let triple = (x) => x * 3;

let arr = Array.init(10, triple);

Js.log(arr);
```

As we can observe, the `triple()` method is applied to all the values of the array. We'll see more examples of such built-in functions in the later lessons.

Calling Functions Within Functions

Functions can also be as used arguments in other functions. Let's write functions for the areas of a triangle and a rectangle. Then, we'll create a function which chooses from the two:

```
let areaTriangle = (h, w) => 0.5 *. h *. w;
let areaRectangle = (l, w) => l *. w;

let getArea = (shape, dim1, dim2) => shape(dim1, dim2);

Js.log(getArea(areaTriangle, 10.0, 15.5));
Js.log(getArea(areaRectangle, 10.0, 15.5));
```

Functions can also be used in `switch` or `if-else` conditionals, but we'll leave that as a "do-it-yourself" exercise.

In the next lesson, we'll learn how ReasonML handles recursion.