Pattern Matching

In this lesson, we will be introduced to the concept of pattern matching and how it relates to data structures in ReasonML.

WE'LL COVER THE FOLLOWING

Previously, we learned how to create tuples containing various data objects called components. The components of a tuple, are indeed, immutable; but this does not mean that we cannot use them.

Pattern matching is a technique for accessing the contents of a tuple. All we have to do is create a pattern which corresponds to the values in a particular tuple. In this pattern, we will define identifiers for the values in the tuple.

Creating a Pattern

Let's create a pattern and map a tuple's components to it:

```
let t = ("Jon Snow", "Winterfell");

/* A pattern with two variables */
let (name, place) = t;

/* name and place are now available for use*/
Js.log(name);
Js.log(place);
```

The pattern we created contains a `name` and a `place`. This process of breaking down a tuple into its elements is called **destructuring**.

> **Note**: The identifiers in a pattern must all have *different* names.

Since the size of the pattern matches that of `t`, the identifiers in it can be *matched* with the contents of the tuple. For example, this piece will not work:

```
let (name, place) = ("Jon", "Snow", "Winterfell");
```

Ignoring Values

Reason allows us to pick out the particular values we need in a tuple and ignore all the rest. This is helpful because we won't need to assign identifiers to each and every tuple value.

To ignore a value, we use the `_` keyword in our pattern:

```
let t = (10, 20, 30);

/* Accessing 20 */
let(_, twenty, _) = t;

Js.log(twenty); /* 20 */
```

The program only focuses on the value `20` and ignores all the others.

By now, we are familiar with the primary functionality of tuples and the principles of pattern matching. In the next lesson, we will be introduced to the more powerful form of tuples called the **record**.