Variant Constructors

In this lesson, we'll learn how constructors allow the variant to use different types.

Constructor Properties

In the previous lesson, we saw a glimpse of a variant's constructors. A constructor is a special type that uses the basic types available in Reason to create a new value. Its name always starts with a capital letter.

The constructors in a variant are separated by the pipe operator, `|`. We found a similar use for the `|` operator in `switch` expressions.

Just like `switch` chooses one of its cases, a variant allows **us** to select one of its constructors.

Declaring Constructors

Let's get started by creating a `car` variant which has two constructors, `Forward` or `Reverse`.

```
type car =
  | Forward
  | Reverse;
```

Now, we can treat these constructors as objects and assign them to `let` bindings.

```
type car =
  | Forward
  | Reverse;

let move = Forward;
```

The `switch` Expression

`switch` expressions can be used for pattern matching in variants. If a pattern is matched, a certain output is produced:

```
type car =
  | Forward
  | Reverse;

let move = Forward;

let ford =
  switch(move) {
    /* Choose between Forward and Reverse */
    | Forward => "Car moving forward"
    | Reverse => "Car moving in reverse"
  };

Js.log(ford);
```

The `ford` variable takes on a different value depending on the constructor we chose! That is the beauty of variants.

Constructor Arguments

Until now, we've viewed variant constructors as unique types. However, they can also take arguments and behave like a function. These arguments are in the form of Reason's built-in data types and are separated by commas.

Let's modify our `car` variant so that its constructors contain arguments:

```
type car =
  | Forward(int)
  | Reverse(int);
```

Each constructor has been given a parameter of type `int`. The value for this parameter must be specified whenever the constructor is *called*:

```
type car =
  | Forward(int)
  | Reverse(int);

let num = 20;
let move = Reverse(num);

let ford =
  switch(move) {
    | Forward(num) => "Car moving forward with speed " ++ string_of_int(num)
    | Reverse(num) => "Car moving in reverse with speed " ++ string_of_int(num)
  };

Js.log(ford);
```

The example above confirms the idea that a `variant` is a collection of other data types. We are using the `num` integer along with a string to create a different output based on the constructor we chose.

The next lesson contains more examples of the usage of variants to help us understand them better.