Labels

In this lesson, we'll explore labeled parameters and why they are useful.

The Position of a Parameter

So far in the course, the arguments in a function have had fixed positions. In other words, the order of inputs must be maintained when calling the function.

Here's an example:

```
type superhero = {
  realName: string,
  heroName: string
};

let createHero = (realName, heroName) => {
  realName,
  heroName
};

let real = "Bruce Wayne";
let hero = "Batman";

/* Correct Order */
let correctBatman = createHero(real, hero);
Js.log(correctBatman);

/* Incorrect Order */
let incorrectBatman = createHero(hero, real);
Js.log(incorrectBatman);
```

In **line 19**, we reversed the order of our parameters, however, since the function was defined with the first parameter always being `realName`, it has mistakenly set the `heroName` as the record's `realName` value.

Fixed parameters such as these are known as **positional parameters**.

Even though the logical error above can be avoided by being careful, it may sometimes be hard to remember the order of parameters if a complex function has a large number of arguments.

Luckily, we can solve this problem in the form of **labeled arguments**.

Labeled Arguments

Reason allows us to define labels for our arguments and use these labels for all operations. In this way, we set the value of a particular label. Since each label is mapped to a parameter, each parameter receives the appropriate value, regardless of its position.

A label is prepended with the `~` character. Let's apply labels to the previous example:

```
type superhero = {
  realName: string,
  heroName: string
};

let createHero = (~realName, ~heroName) => {
  realName,
  heroName
};

let real = "Bruce Wayne";
let hero = "Batman";

/* Order does not matter */
let batman = createHero(~heroName = hero, ~realName = real);
Js.log(batman);
```

In the function call, we explicitly assign the appropriate value to each label. Because of this, we do not need to keep track of the order.

Different Names for Labels and Parameters

Sometimes, we create long labels which increase the readability of the code. However, dealing with them in the actual function implementation can be a hassle.

Therefore, define separate names for our arguments and labels by using the `as` keyword:

```
let replaceVillain = (~arrayOfVillains as arr, ~newVillain as nv, ~oldVillainIndex as ov) =>
  if (ov < Array.length(arr)){
    arr[ov] = nv;
  }
};

let a = [|"Darth Vader", "Cersei Lannister", "The Joker"|];
replaceVillain(~arrayOfVillains = a, ~oldVillainIndex = 2, ~newVillain = "Voldemort");
Js.log(a);
```

In the next lesson, we'll create customized operators using the function syntax.