Currying

In this lesson, we'll learn what currying is.

WE'LL COVER THE FOLLOWING ^

- The Definition
- Example

The Definition

Currying is the process of dividing a function with multiple parameters into separate functions for each parameter.

A function can be curried when we provide it with fewer arguments than it originally needs and assign this incomplete call to a new let binding.

Now, we only need to pass the remaining arguments to the new function.

Note: Passing more arguments than the function actually has, will produce a compile-time error. Currying only works when we provide an incomplete number of arguments.

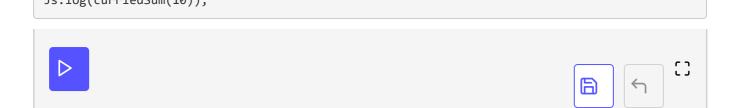
Example

Let's make things clearer with the help of an example.

```
let sum = (f, s, t) => {
  f + s + t;
};

/* Provide the first 2 arguments to sum() */
let curriedSum = sum(2, 5);
Js.log(curriedSum); /* Prints the curried function's name */

/* Now we only need to specify the 3rd value */
Js.log(curriedSum(10));
```



In this example, we've curried the first two values of the sum() function. Now, curriedSum() will always the values x = 2 and y = 5. All we have to do is specify the third argument **only**. We can only pass the remaining arguments to a curried function. Passing extra arguments would produce an error.

If we try to print curriedSum, the compiler simply prints the function name.

Currying can make code cleaner in case of multiple arguments.

We could refactor the example above to make curried functions for each argument:

```
let sum = (f, s, t) => {
    f + s + t;
};
let sumF = sum(2);
let sumS = sumF(5);

Js.log(sum(2, 5, 10));
Js.log(sumF(5, 10));
Js.log(sumF(5, 10));
D
```

Until now, we have seen how functions work with data structures such as tuples and arrays. In the next lesson, we'll observe their interactions with records.