let Binding

This lesson is all about assigning names to data so that it can be used later.

Keeping Track of Values

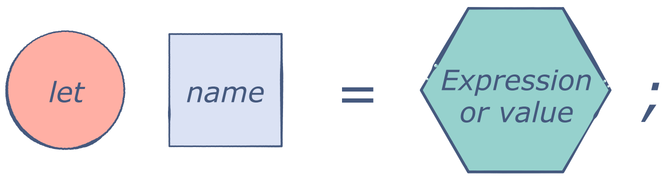
Storing data in an identifier is essential for almost any language. Identifiers facilitate the flow of information throughout a program. If we have a value that is being used in several places, it would be impractical to hardcode it again and again.

Attaching an identifier to the value would make the code more concise, more readable, and more reusable.

The `let` Identifier

The `let` keyword is used to assign, or *bind*, a name to a value. This value can now be accessed and modified through the corresponding name.

Apart from simple data types, `let` bindings can also be assigned to the results of expressions. The basic syntax for a `let` binding can be seen below:



let name = expression;

The assignment operator `=` indicates to the compiler that this is an assignment. We are assigning the expression/value on the right to the label on the left. Using this format, let's create a few identifiers:

```
let x = 10;
let y = 45.6 +. 5.1;
Js.log(x); /* 10 */
Js.log(y); /* 50.7 */

let z = x * int_of_float(y);
Js.log(z); /* 500 */
```

We can use `let` bindings for any type of data. In general, the identifier can guess the type of the value:

```
let str1 = "Hello";
let str2 = "World";
Js.log(str1 ++ str2); /* HelloWorld */
Js.log(String.length(str1)); /* 5 */
let b = true;
Js.log(!b); /* false */
```

Mutability

`let` bindings are **immutable**, which means that the values inside them cannot be directly changed. So, something like this wouldn't work in Reason:

```
let str = "Hello";
str = "Educative";
```

However, Reason has a workaround for this. Each `let` binding can be **redefined** with a new value:

```
let str = "Hello";
Js.log(str); /* Hello */
```

```
let str = "Educative";
Js.log(str); /* Educative */
```

A redefinition does **not** imply that the old definition is lost. It is merely shadowed by the new one and can still be used in some ways. More on this later.

This proves to be very helpful since we have access to both definitions for different purposes.

> **Note**: We cannot declare a `let` binding without setting a value for it, i.e., the right-hand side of the assignment operator cannot be a **null** value.

In the next lesson, we'll learn how to make our `let` bindings safer according to type conventions.