The Ternary Operator

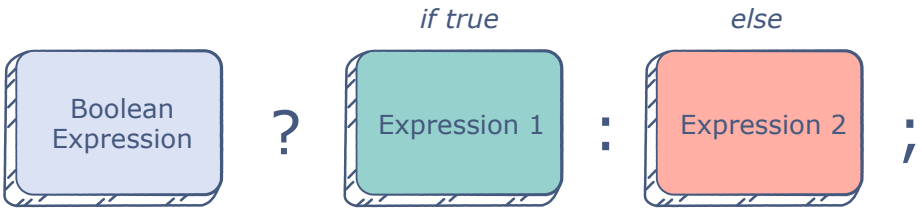
In this lesson, we'll understand how the ternary operator serves as a substitute for the if-else expression.

WE'LL COVER THE FOLLOWING

Much like the `if-else` expression, the ternary operator is used when our condition has two possible outcomes. It is generally represented by the `?:` characters.

The Structure

A ternary expression asks a question, and then presents two possible expressions to be executed:



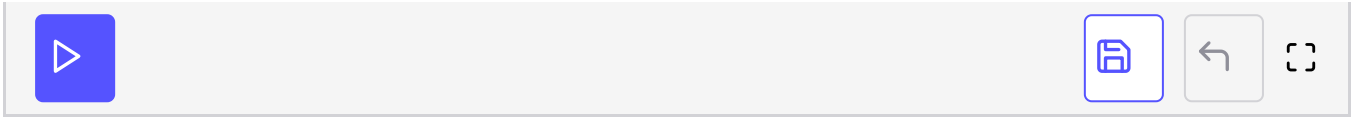
The purpose of ternary expressions is to make code more concise and readable as compared to the `if-else` syntax.

Creating A Ternary Expression

To get started, we'll write a simple `if-else` expression in Reason:

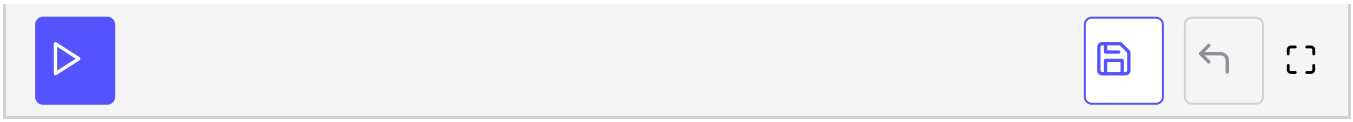
```
let a : float = 10.56 *. 5.67;
let b : float = 10.67 *. 5.56;

let result = if (a < b) {
  10;
}
```

```
else {
  20;
};

Js.log(result);
```



Nothing tricky going on here. Yet, a simple conditional is taking up about 6 lines of code! Let's refactor this code into ternary terms:

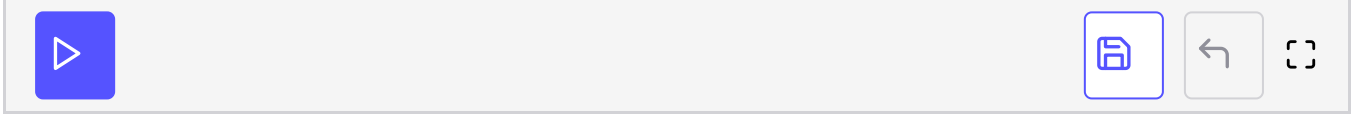
```
let a : float = 10.56 *. 5.67;
let b : float = 10.67 *. 5.56;

let result = (a < b) ? 10 : 20;
Js.log(result);
```



The benefit of the ternary operator is quite evident here. We've encapsulated a verbose conditional expression into a single line.

The ternary operator can also be used when the expression we are returning is longer than one line. However, this is not a popular practice.

```
let result = (10 < 20) ? {
  let c = 10;
  c * 49 ;
}: {
  let f = 80;
  f * 90;
};
Js.log(result);
```



As with all conditionals, we need to make sure that the type of returning expressions is the same, otherwise, we'll get a compilation error.

That brings us to the end of our discussion on conditionals. While we do have several conditionals available, Reason code relies usually on the `switch` expression due to its ability to handle several cases.

Check out the quiz in the next lesson to test your understanding of conditionals.