Polymorphic Comparison

In this lesson, we'll get familiarized with the concept of polymorphism in ReasonML.

WE'LL COVER THE FOLLOWING

- What is Polymorphism?
- Comparison Operators for Different Types
  - Examples

What is Polymorphism?

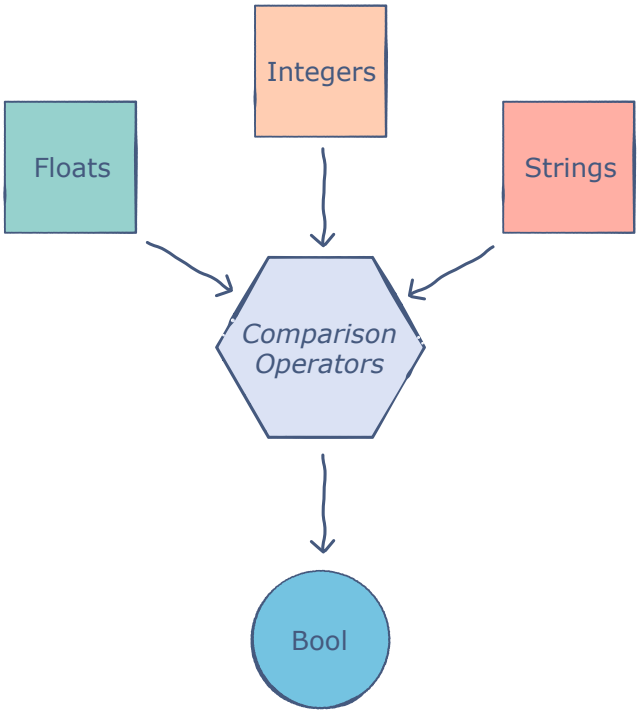
The meaning behind the intimidating term, “polymorphism”, is really quite simple. Polymorphism refers to a certain method changing its behavior according to different situations.

This can prove very useful as the same method can be used for different purposes. ReasonML also exhibits polymorphism in some cases, and we’ve already been through one of them.

Comparison Operators for Different Types

Consider a comparison operator which always gives us a boolean depending on the condition we specify. How many types can it support? We can compare *integers*, *floats*, *characters*, and even *strings*!

The operator knows the difference between these types and how to perform comparisons for each type. This is polymorphism at its finest.



If we think about it, arithmetic or logical operators are not as flexible as comparison operators. Integers use operators like `+`, whereas floats require `+.`, etc.

Logical operators work solely with bools, which makes them non-polymorphic as well.

This property makes the comparison operators unique. They work regardless of the data type.

Examples

```
Js.log(5 < 2);
Js.log('r' >= 'm');
Js.log("hello" == "Hello"); /* Compares ASCII values, hence, "Hello" is larger */
Js.log("hello" > "a"); /* "a" has a smaller ASCII value so this returns true */
```

Later in the course, we'll see the true potential of polymorphism in ReasonML, even if it seems trivial right now.

In the next lesson, we'll study type conversions in Reason.