Polymorphic Functions

This lesson will show us how functions can adapt the principles of polymorphism.

WE'LL COVER THE FOLLOWING

What are Polymorphic Functions?

> Polymorphic functions are functions that work with several data types.

Since Reason is a strongly-typed language, we often explicitly define the type of our values.

Applying this rule to function parameters, we would end up with the same function being redefined again and again for different data types.

Here is an example:

```
let intTuple = (x:int, y:int) => (x , y);
Js.log(intTuple(10, 20));

let floatTuple = (x:float, y:float) => (x , y);
Js.log(floatTuple(10.5, 20.6));

let stringTuple = (x:string, y:string) => (x , y);
Js.log(stringTuple("Hello", "Educative"));

let intStringTuple = (x:int, y:string) => (x , y);
Js.log(intStringTuple(10, "Educative"));

/* And so on... */
```

The code above seems pretty redundant because the actual operation performed by the functions is not dependent on the types of their parameters.

All the functions simply create a tuple from two arguments.

To make things easier, Reason provides us with the **polymorphic type variables**.

Polymorphic Type Variables

A type variable is just a name which specifies that any type can be used in its place. Every type variable is preceded by the `'` symbol.

Let's refactor our previous code so that a single function can create tuples out of any pair of values:

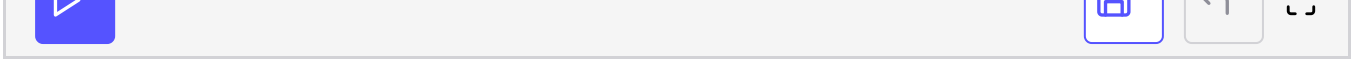
```
let createTuple = (x: 'first, y: 'second) => (x, y);
Js.log(createTuple(10, "Educative"));
Js.log(createTuple(9.9, 5.5));
```

This is a big improvement! All we have done is define our types variables as `'first` and `'second` (they can be named them anything else as well). Now, these variables can be replaced with any of the other data types.

> **Note**: While we can name our type variables whatever we want, Reason interprets them in the form of `'a`, `'b`, `'c` etc. These refer to *Alpha*, *Beta*, *Charlie* and so on.

Sometimes, if we do not explicitly define the types of our parameters, the compiler would automatically infer them as type variables:

```
let createTuple = (x, y) => (x, y);
Js.log(createTuple(10, "Educative"));
Js.log(createTuple(9.9, 5.5));
```



However, this might make the code a little less readable.

This brings us to the end of our discussion on functions.

Test your theoretical understanding of functions by taking the quiz ahead!