Mutability

In this lesson, we'll learn how to make object values mutable.

WE'LL COVER THE FOLLOWING

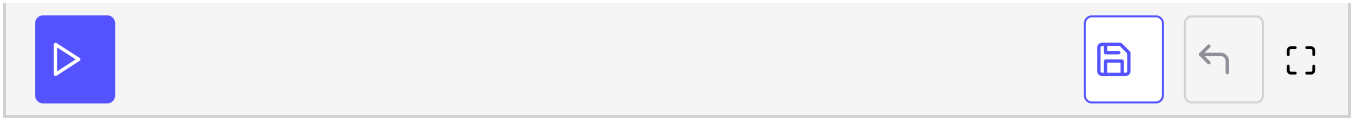
`val` Mutation

Similar to records, we can modify the values in an object by making them mutable. Let's take a look at the `circle` object from the previous lesson.

```
type circle('a) = {.. /* Creating a polymorphic type */
  getRadius: float
} as 'a;

let c: circle({. /* Define method types in construtor */
  getRadius: float,
  getArea: float }) = {
  val r = 10.5;
  pub getRadius = r;
  pub getArea = r *. 3.142;
};

Js.log(c#getArea);
```



In this case, we only have one value, `r`. In order to make it mutable, we'll use the trusted `ref` wrapper. Along with that, we'll create a new `setRadius()` method to alter the value of `r`.

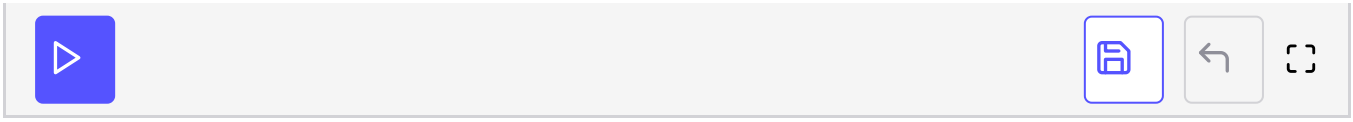
The Syntax

Let's try it now:

```
type circle = {.
  getRadius: float,
  setRadius: (float) => float, /* Setting the type for setRadius */
  getArea: float
};

let c: circle = {
  val r = ref(10.5); /* A ref wrapper for r */
  pub getRadius = r^;
  pub setRadius = (x: float) => {
    r := x; /* Assigning a new value to r */
    r^;
  };
  pub getArea = r^ *. 3.142;
};

Js.log(c#getRadius);
c#setRadius(50.2);
Js.log(c#getRadius);
```



As we can observe, the radius of the circle is now mutable! Remember, since the `setRadius` method requires a parameter, we've specified the parameter type in the type definition.