Open Objects

In this lesson, we'll learn about the functionality of open objects in Reason.

WE'LL COVER THE FOLLOWING

The Restrictions of Closed Objects

The objects we've dealt with until now have been **closed objects**. In closed objects, we cannot define public functions unless they have been declared in the type definition. Any new members in the object's implementation would have to be *values* or *private methods*.

For example, take a look at the `circle` type below:

```
type circle = {.
  getRadius: float
};

let c: circle = {
  val r = 10.5;
  pub getRadius = r;
  /* pub getArea = r *. 3.142; */
};
```

If we try to create the `getArea()` method in **line 8**, we'd get a compile-time error. This can be a problem. Sometimes, we need to define separate interfaces (public methods) for different objects of the same type. But with closed objects like the one seen above, this is not possible.

Fortunately, Reason provides us with a solution in the form of **open objects**.

Properties of Open Objects

An open object is a polymorphic object which allows us to create new public methods without specifying them in the type definition.

Open object type definitions begin with double periods, `..`.

Due to their polymorphic nature, open object type definitions require a type variable. We'll use the conventional `'a`.

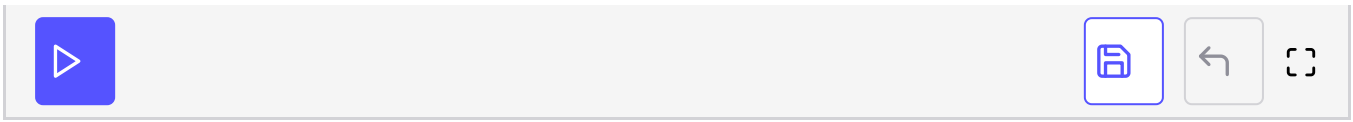
When declaring an open object, we must provide the types of all its public methods as **arguments** of the object.

To help us understand all of this jargon better, let's create the `getArea()` method for an open `circle` object:

```
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);
```



Note that we must still use the single dot when defining types in the constructor (**line 5**). After this, we can create the `getArea()` method just like any other!

Finally, in the next lesson, we'll discuss mutability in Reason objects.