Classes in R

Classes in R

R is not actually an object-oriented language in any real sense.

But it has things that it calls “classes”, and things that it calls “methods.”

They look more like “types” and “polymorphic functions” as used in the functional-programming world.

Classes

There are two types of classes in R.

- S3 (older)
- S4 (newer)

S3 Classes

An S3 class is a list with an extra attribute, called “class”

```
> l <- list(foo = "bar", baz = "qux")
> class(l) <- "myclass"
> l

$foo
[1] "bar"

$baz
[1] "qux"

attr(,"class")
[1] "myclass"
```

S3 Classes

Let's revisit our simple regression function from the Functions course:

```
run_regression <- function(y, x) {
    coef <- cov(x,y) / var(x)
    se <- calc_se(y, x, coef)
    list(coef=coef, se=se)
}
```

S3 Classes

Now let’s make it return a class:

```
run_regression <- function(y, x) {
    coef <- cov(x,y) / var(x)
    se <- calc_se(y, x, coef)
    model <- list(coef=coef, se=se)
    class(model) <- "simple_model"
    model
}
```

Why class?

Classes are not useful in-and-of themselves. They are only useful when we use them together with polymorphic functions, called “generic methods” in R.

Polymorphic means that the same function can operate on different types (in R, “classes”), with different behavior.

Polymorphic Functions

Consider the eval_model() function from the Functions course:

```
eval_model <- function(coef, se, beta, conf = 1.96) {
    up <- coef + se*conf
    down <- coef - se*conf
    beta > down & beta < up
}
```

This function requires the user to manually input the coefficients and the standard errors. This is good, because it makes the function general. But it requires the user to be a bit sophisticated.

Polymorphic Functions

It would be simpler to use, if the user only had to give us the model:

```
eval_model <- function(model, beta, conf = 1.96) {
    coef <- model$coef
    se <- model$se
    up <- coef + se*conf
    down <- coef - se*conf
    beta > down & beta < up
}
```

But now the function is very specialized. What if I have a different model, and it does not have the standard errors under a field called “se”?

Polymorphic Functions

This function only works for the exact regression function we created.

We would have to use a different eval_model() function for every model we create.

That sounds tedious, and hard to remember all the different function names.

Polymorphic Functions

This is solved with polymorphic functions.

Polymorphic Functions

Let’s create a polymoprhic eval_model() function:

```
eval_model <- function(model, beta, conf) {
    UseMethod("eval_model")
}
```

Polymorphic Functions

Now, we write a specific function for each “class” that needs to be evaluated:

```
eval_model.simple_model <- function(model, beta, conf) {
    coef <- model$coef
    se <- model$se
    up <- coef + se*conf
    down <- coef - se*conf
    beta > down & beta < up
}
```

This “method”, as it’s called in R, encapsulates intimate knowledge about the class “simple_model”. Like a method in Object-Oriented programming!

Polymorphic Functions

Now we can use eval_model() without us needing to know what the regression model actually looks like, or what eval_model() uses:

```
model <- run_regression(y, x)
eval_model(model, true_beta)
```

Polymorphic Functions

Many functions you will use, and maybe have already used, are polymorphic.

Polymorphic Functions

For example, the mean() function is polymorphic:

```
> methods(mean)
```

```
[1] mean.Date     mean.default  mean.difftime mean.POSIXct
```

In addition to the default version, the creators of R wanted to be able to take the mean of date and timestamp objects.

Polymorphic Functions

Similarly, the coef() function is polymorphic:

```
> methods(coef)

[1] coef.aov*     coef.Arima*   coef.default* coef.listof*
[6] coef.nls*
```

The coef() function gets the coefficients from a model.

Polymorphic Functions

Let’s implement a coef() method for our new class. That way, users can get the coefficients without needed to know where we’ve put them:

```
coef.simple_model <- function(model) model$coef

> coef(model)

[1] 0.4677848
```