

[Personal](#) [Open source](#) [Business](#) [Explore](#)[Pricing](#) [Blog](#) [Support](#)[This repository](#)[Sign in](#)[Sign up](#)[Igreski](#) / **datasciencectacontent**[Watch](#)

40

[★ Star](#)

107

[Fork](#)

437

[Code](#)[Issues](#) 0[Pull requests](#) 0[Projects](#) 0[Pulse](#)[Graphs](#)

Branch: master ▾

**datasciencectacontent** / [markdown](#) / [rprog-OOPandR.md](#)[Find file](#)[Copy path](#) **Igreski** Add article -- object oriented programming and R 2fce5b5 on 5 Nov

1 contributor

54 lines (29 sloc) 3.36 KB

[Raw](#)[Blame](#)[History](#)

# Object Oriented Programming and R

In September 2016 a student asked the following questions on the *R Programming* discussion forum:

I have a question about the extend to which R packages (should) follow the OOP design principles. This stood out to me when experimenting with the `lm()` function but applies to almost every package that is widely used. Here is a way (or the only way?) to access the coefficients of a regression model in R:

```
fit <- lm(y ~ x, data = myDF);  
  
coef(fit)
```

Coming from an OOP background this rings a lot of bells, for example what stops the user from typing "`coef(2)`" or "`coef("abc")`"? Making `coef()` a function rather than a method of the "`fit`" object not only pollutes the global namespace but also makes code error prone. The same holds true for many other functions that in my opinion should have been class methods. What is the reason behind this design?

R is built on the S language, so it contains the key features of S. In a [2014 use!R presentation](#), John Chambers explained three principles on which S (and R) are based, including:

1. Everything is an object,
2. Everything happens in a function, and
3. Functions are interfaces to algorithms.

In R, there are two underlying object systems: the S3 system and the S4 system. The S3 system is designed around the use of `list()` to create objects. I explain how this relates to Programming Assignment 2 in my articles [makeCacheMatrix\(\) as an Object](#) and [Demystifying makeVector\(\)](#).

Regarding the student's specific question, given #1 above, there is a way to access the coefficients output from `lm()` without using a function. They can be accessed with the `$` form of the extract operator as follows.

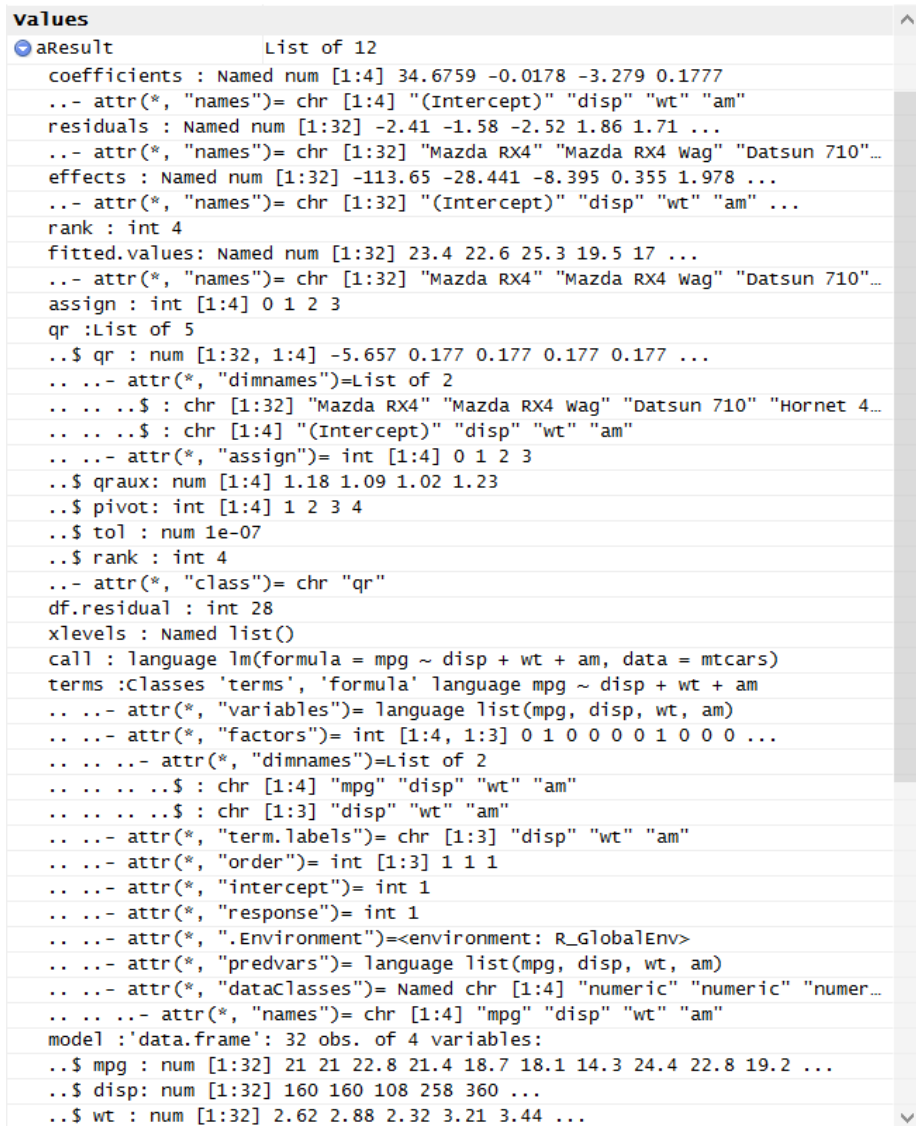
```
#  
# example highlighting object output from lm  
#  
  
library(datasets)  
data(mtcars)  
  
aResult <- lm(mpg ~ disp + wt + am,data=mtcars)  
  
# now, access model coefficients from output object, aResult  
  
aResult$coefficients
```

```

> aResult <- lm(mpg ~ disp + wt + am, data=mtcars)
> #
> # example highlighting object output from lm
> #
>
> library(datasets)
> data(mtcars)
>
> aResult <- lm(mpg ~ disp + wt + am, data=mtcars)
> aResult$coefficients
(Intercept)      disp      wt      am
34.67591088 -0.01780491 -3.27904388  0.17772414
> |

```

In the example above `aResult` is an S3 object, i.e. a `list()`. The output object from `lm()` consists of 12 named list elements, as illustrated from the *RStudio Environment Viewer*:



```

values
aResult      List of 12
 coefficients: Named num [1:4] 34.6759 -0.0178 -3.279 0.1777
 .. attr(*, "names")= chr [1:4] "(Intercept)" "disp" "wt" "am"
 residuals : Named num [1:32] -2.41 -1.58 -2.52 1.86 1.71 ...
 .. attr(*, "names")= chr [1:32] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710" ...
 effects : Named num [1:32] -113.65 -28.441 -8.395 0.355 1.978 ...
 .. attr(*, "names")= chr [1:32] "(Intercept)" "disp" "wt" "am" ...
 rank : int 4
 fitted.values: Named num [1:32] 23.4 22.6 25.3 19.5 17 ...
 .. attr(*, "names")= chr [1:32] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710" ...
 assign : int [1:4] 0 1 2 3
 qr :List of 5
 ..$ qr : num [1:32, 1:4] -5.657 0.177 0.177 0.177 0.177 ...
 .. .. attr(*, "dimnames")=List of 2
 .. .. ..$ : chr [1:32] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710" "Hornet 4..."
 .. .. ..$ : chr [1:4] "(Intercept)" "disp" "wt" "am"
 .. .. attr(*, "assign")= int [1:4] 0 1 2 3
 ..$ graux: num [1:4] 1.18 1.09 1.02 1.23
 ..$ pivot: int [1:4] 1 2 3 4
 ..$ tol : num 1e-07
 ..$ rank : int 4
 .. attr(*, "class")= chr "qr"
 df.residual : int 28
 xlevels : Named list()
 call : language lm(formula = mpg ~ disp + wt + am, data = mtcars)
 terms :classes 'terms', 'formula' language mpg ~ disp + wt + am
 .. .. attr(*, "variables")= language list(mpg, disp, wt, am)
 .. .. attr(*, "factors")= int [1:4, 1:3] 0 1 0 0 0 0 1 0 0 0 ...
 .. .. attr(*, "dimnames")=List of 2
 .. .. ..$ : chr [1:4] "mpg" "disp" "wt" "am"
 .. .. ..$ : chr [1:3] "disp" "wt" "am"
 .. .. attr(*, "term.labels")= chr [1:3] "disp" "wt" "am"
 .. .. attr(*, "order")= int [1:3] 1 1 1
 .. .. attr(*, "intercept")= int 1
 .. .. attr(*, "response")= int 1
 .. .. attr(*, ".Environment")=environment: R_GlobalEnv>
 .. .. attr(*, "predvars")= language list(mpg, disp, wt, am)
 .. .. attr(*, "dataClasses")= Named chr [1:4] "numeric" "numeric" "numer..."
 .. .. attr(*, "names")= chr [1:4] "mpg" "disp" "wt" "am"
 model :data.frame: 32 obs. of 4 variables:
 ..$ mpg : num [1:32] 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 ..$ disp: num [1:32] 160 160 108 258 360 ...
 ..$ wt : num [1:32] 2.62 2.88 2.32 3.21 3.44 ...

```

To understand more of the details behind the design of R including its object oriented features, I highly recommend reading [Software For Data Analysis: Programming with R](#), by John Chambers.

Regarding the student's comments about object orientation, I've written code in at least four object oriented languages (Smalltalk, Java, Objective C, and R). Each has its idiosyncracies. That said, given the focus on R functions in the early lectures in *R Programming*, one can understand why R doesn't appear very "object oriented."

In Programming Assignment 2 one can see how the object created by `makeVector()` is an object. It has state (the elements `x` and `m`), behavior (the methods `set()`, `get()`, `getmean()`, and `setmean()`). Encapsulation exists because once an object of type `makeVector()` is created, one can only access `x` or `m` through the methods exposed through the final `list()` call at the end of the `makeVector()` function.

For more details on how `makeVector()` works, one can review my article [Demystifying makeVector\(\)](#).

