Object-oriented programming in R

Kevin Lin
@linnylin92

Object-oriented programming (OOP) allows the programmer to have full control of what the user can do/needs to know.

- “Objects” are instances of “classes”

Object-oriented programming (OOP) allows the programmer to have full control of what the user can do/needs to know.

- "Objects" are instances of "classes"

- Classes are associated with a particular set of public and private functions
  - Other "classes" (i.e., the user) can only interact with the objects via the public functions

Object-oriented programming (OOP) allows the programmer to have full control of what the user can do/needs to know.

- "Objects" are instances of "classes"
- Classes are associated with a particular set of public and private functions
  - Other "classes" (i.e., the user) can only interact with the objects via the public functions
- Concepts of "inheritance" and "multiple dispatch" simply this interface that the user needs to learn

Object-oriented programming (OOP) allows the programmer to have full control of what the user can do/needs to know.

```
class David {
     public String getName() {
          return "David";
     }
     // private method... nobody but David's "instances" can use it..
     private int getAge() {
          return 19;
     }

 }
```

```
class Other  {
     David davidOne = new David();
     String davidsName = davidOne.getName();
     int davidsAge = davidOne.getAge(); //<-- Compiler error,
}
```

Object-oriented programming (OOP) allows the programmer to have full control of what the user can do/needs to know.

```
class David {
     public String getName() {
          return "David";
     }
     // private method... nobody but David's "instance" can use it..
     private int getAge() {
          return 19;
     }
     // Here the call to "getAge()" will succeed, because it is visible
     // inside the class
     public boolean hasSameAgeAs( David otherDavid ) {
          return this.getAge() == otherDavid.getAge();
     }
 }
```

R uses a system called S3 to emulate OOP, but it's janky and transparent.

R uses a system called S3 to emulate OOP, but it's janky and transparent.

- "Class" is something in R that the user can manually change at any time (somewhat undesirable).

R uses a system called S3 to emulate OOP, but it's janky and transparent.

- "Class" is something in R that the user can manually change at any time (somewhat undesirable).

- Everything is out in the open, and can be changed by the user at any time.

R uses a system called S3 to emulate OOP, but it's janky and transparent.

- "Class" is something in R that the user can manually change at any time (somewhat undesirable).
- Everything is out in the open, and can be changed by the user at any time.
- More of a "implicit agreement" between the programmer and users that the user won't mess with the object in weird ways.

R uses a system called S3 to emulate OOP, but it's janky and transparent.

```
> x <- rnorm(10); y <- rnorm(10)
> res <- lm(y ~ x)
> res

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
    0.07065     -0.04253
```

R uses a system called S3 to emulate OOP, but it's janky and transparent.

```
> x <- rnorm(10); y <- rnorm(10)
> res <- lm(y ~ x)
> res

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
    0.07065     -0.04253


> class(res)
[1] "lm"
> typeof(res)
[1] "list"
```

R uses a system called S3 to emulate OOP, but it's janky and transparent.

```
> attributes(res)
$names
 [1] "coefficients"  "residuals"
 [3] "effects"       "rank"
 [5] "fitted.values" "assign"
 [7] "qr"            "df.residual"
 [9] "xlevels"       "call"
[11] "terms"         "model"

$class
[1] "lm"
```

R uses a system called S3 to emulate OOP, but it's janky and transparent.

```
> names(res)
 [1] "coefficients"  "residuals"     "effects"
 [4] "rank"          "fitted.values" "assign"
 [7] "qr"            "df.residual"   "xlevels"
[10] "call"          "terms"         "model"
> res[[1]]
(Intercept)           x
 0.07065148 -0.04253140
```

R uses a system called S3 to emulate OOP, but it's janky and transparent.

```
> methods(class = class(res))
 [1] add1             alias            anova
 [4] case.names       coerce           confint
 [7] cooks.distance   deviance         dfbeta
[10] dfbetas          drop1            dummy.coef
[13] effects          extractAIC       family
[16] formula          hatvalues        influence
[19] initialize       kappa            labels
[22] logLik           model.frame      model.matrix
[25] nobs             plot             predict
[28] print            proj             qr
[31] residuals        rstandard        rstudent
[34] show             simulate         slotsFromS3
[37] summary          variable.names   vcov
see '?methods' for accessing help and source code
```

R uses a system called S3 to emulate OOP, but it's janky and transparent.

```
> methods(print)
  [1] print.acf*                                  print.AES*
  [3] print.anova*                                print.aov*
  [5] print.aovlist*                              print.ar*
  [7] print.Arima*                                print.arima0*
  [9] print.AsIs                                  print.aspell*
 [11] print.aspell_inspect_context*               print.bibentry*
 [13] print.Bibtex*                               print.browseVignettes*
 [15] print.by                                    print.bytes*
 [17] print.changedFiles*                         print.check_code_usage_in_package*
 [19] print.check_compiled_code*                  print.check_demo_index*
 [21] print.check_depdef*                         print.check_details*
 [23] print.check_details_changes*                print.check_doi_db*
 [25] print.check_dotInternal*                    print.check_make_vars*
 [27] print.check_nonAPI_calls*                   print.check_package_code_assign_to_globalenv*
 [29] print.check_package_code_attach*            print.check_package_code_data_into_globalenv*
 [31] print.check_package_code_startup_functions* print.check_package_code_syntax*
 [33] print.check_package_code_unload_functions*  print.check_package_compact_datasets*
 [35] print.check_package_CRAN_incoming*          print.check_package_datasets*
```

R uses a system called S3 to emulate OOP, but it's janky and transparent.

```
> stats:::print.lm
function (x, digits = max(3L, getOption("digits") - 3L), ...)
{
    cat("\nCall:\n", paste(deparse(x$call), sep = "\n", collapse =
"\n"),
        "\n\n", sep = "")
    if (length(coef(x))) {
        cat("Coefficients:\n")
        print.default(format(coef(x), digits = digits), print.gap
= 2L,
            quote = FALSE)
    }
    else cat("No coefficients\n")
    cat("\n")
    invisible(x)
}
<bytecode: 0x109b78400>
<environment: namespace:stats>
```

R uses a system called S3 to emulate OOP, but it's janky and transparent.

```
> print.lm = function(x){
+   print("BTS is the best KPop group")
+ }
> res
[1] "BTS is the best KPop group"
```

R uses a system called S3 to emulate OOP, but it's janky and transparent.

```
> class(res) = "no_funny_bznz"
> res
$coefficients
(Intercept)           x
 0.07065148 -0.04253140

$residuals
          1           2           3           4
-0.59637359 -1.11487008  0.35733555 -0.02406449
          5           6           7           8
-1.25389170 -0.37587106 -0.15016972  2.19716624
          9          10
-0.08940000  1.05013885

$effects
(Intercept)           x
-0.21182323 -0.13611454  0.89496783  0.33368967

-1.09972289  0.07556126 -0.68631976  2.23066655
```

R uses a system called S3 to emulate OOP, but it's janky and transparent.

```
> attributes(res)
$names
 [1] "coefficients"  "residuals"
 [3] "effects"       "rank"
 [5] "fitted.values" "assign"
 [7] "qr"            "df.residual"
 [9] "xlevels"       "call"
[11] "terms"         "model"

$class
[1] "no_funny_bznz"
```

R uses a concept of "generics" to decide that `print.lm` was the appropriate `print` function for an object of class `lm`.

```
> print
function (x, ...)
UseMethod("print")
<bytecode: 0x108af9310>
<environment: namespace:base>
```

See http://adv-r.had.co.nz/S3.html for more details

R uses a concept of "generics" to decide that `print.lm` was the appropriate `print` function for an object of class `lm`.

UseMethod {base} R Documentation

Class Methods

Description

R possesses a simple generic function mechanism which can be used for an object-oriented style of programming. Method dispatch takes place based on the class(es) of the first argument to the generic function or of the object supplied as an argument to `UseMethod` or `NextMethod`.

See http://adv-r.had.co.nz/S3.html for more details

R uses a concept of "generics" to decide that `print.lm` was the appropriate `print` function for an object of class `lm`.

```
> print.no_funny_bznz = function(x){
+   print("I went to the Monsta X concert this weekend")
+   print(paste0("Oh btw, the coefficients are ",
+                paste0(round(as.numeric(coef(x)), 2),
+                       collapse = ", ")))
+ }
> res
[1] "I went to the Monsta X concert this weekend"
[1] "Oh btw, the coefficients are 0.27, -0.24"
```

See http://adv-r.had.co.nz/S3.html for more details

R doesn't have a lot of features that a "real" OOP has.

- No barrier between programmer and user (i.e., user can do whatever she wants, no sense of private functions)

R doesn't have a lot of features that a "real" OOP has.

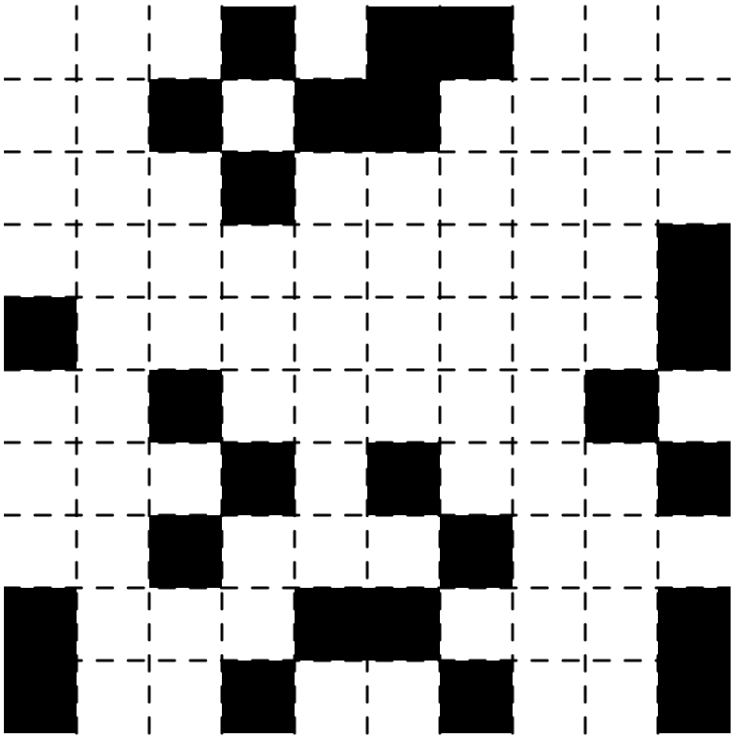
- No barrier between programmer and user (i.e., user can do whatever she wants, no sense of private functions)
- No constructors (i.e., unclear how to make new objects and how behind-the-scenes how memory is allocated)

R doesn't have a lot of features that a "real" OOP has.

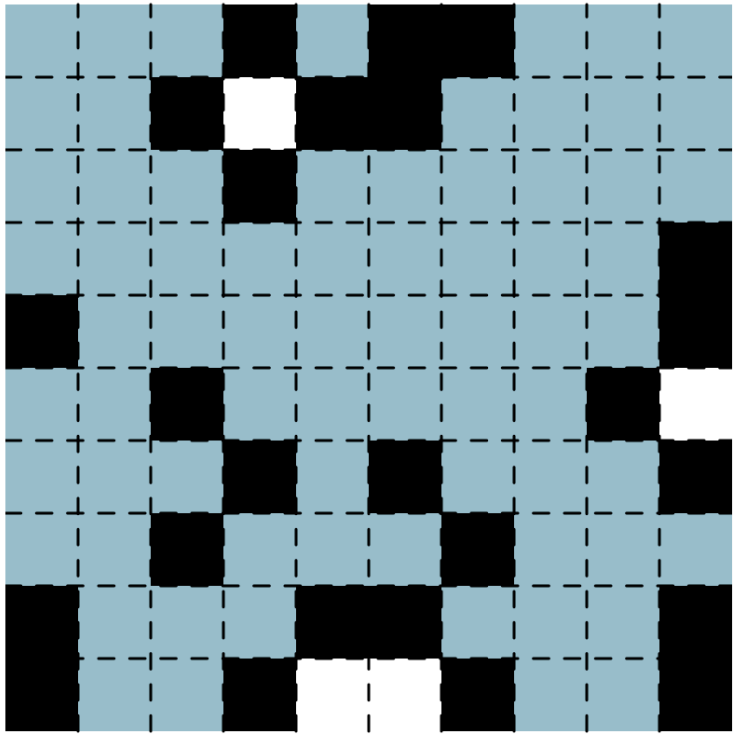
- No barrier between programmer and user (i.e., user can do whatever she wants, no sense of private functions)
- No constructors (i.e., unclear how to make new objects and how behind-the-scenes how memory is allocated)
- No inheritance or multiple dispatch (i.e., have to redefine methods for every new class or new input class even if the change is really minor)

Let's consider a percolation example (which you'll need to think about for HW 5). [DISCLAIMER: This is NOT solutions.]

Size: 10



Size: 10



Let's consider a percolation example (which you'll need to think about for HW 5). [DISCLAIMER: This is NOT solutions.]

- We want a class `board` (the grid of squares that we want to see whether or not percolates).

- The functions associated with class board might be `plot`, `percolate`, and `print`.

Let's consider a percolation example (which you'll need to think about for HW 5). [DISCLAIMER: This is NOT solutions.]

- We want a class `board` (the grid of squares that we want to see whether or not percolates).
- The functions associated with class board might be `plot`, `percolate`, and `print`.
- Another function might be `compare`, which compares two boards to see which one has a shorter path from the top of the board to the bottom.

Let's consider a percolation example (which you'll need to think about for HW 5). [DISCLAIMER: This is NOT solutions.]

```
board = function(n){ # a "constructor"
  structure(list(grid = matrix(0, n, n)),
            class = "board")
}

print.board = function(x){
  cat("This is a board of size", nrow(x$grid), ":\n")
  print(x$grid)
}
```

Let's consider a percolation example (which you'll need to think about for HW 5). [DISCLAIMER: This is NOT solutions.]

Use real OOP language with sophisticated features to ensure proper usage of code by the user

Use S3 in R because… who cares? Not my problem if the user screws things up

Let's consider a percolation example (which you'll need to think about for HW 5). [DISCLAIMER: This is NOT solutions.]

```
> obj = board(5)
> obj
This is a board of size 5 :
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    0    0
[2,]    0    0    0    0    0
[3,]    0    0    0    0    0
[4,]    0    0    0    0    0
[5,]    0    0    0    0    0
```

```
> obj$grid[1,3] = "lol"
> obj
This is a board of size 5 :
     [,1] [,2] [,3]  [,4] [,5]
[1,] "0"  "0"  "lol" "0"  "0"
[2,] "0"  "0"  "0"   "0"  "0"
[3,] "0"  "0"  "0"   "0"  "0"
[4,] "0"  "0"  "0"   "0"  "0"
[5,] "0"  "0"  "0"   "0"  "0"
```

R has a slightly more formal built-in OOP system called S4. It's pretty good, but not everyone uses it.

- There's a formal way to "represent" an object of the class, and more formal way to associating methods to a class.

R has a slightly more formal built-in OOP system called S4. It's pretty good, but not everyone uses it.

- There's a formal way to "represent" an object of the class, and more formal way to associating methods to a class.
- Objects are no longer stored as a list, but using a S4 system where elements are stored in "slots".
- Has multiple dispatch.

R has a slightly more formal built-in OOP system called S4. It's pretty good, but not everyone uses it.

- There's a formal way to "represent" an object of the class, and more formal way to associating methods to a class.
- Objects are no longer stored as a list, but using a S4 system where elements are stored in "slots".
- Has multiple dispatch.
- Still no real barrier between programmer and user though.
- See http://adv-r.had.co.nz/S4.html

R has a slightly more formal built-in OOP system called S4. It's pretty good, but not everyone uses it.

```
setClass("board", representation(grid = "matrix"))
construct_board = function(n){ #explicit constructor
  new("board", grid = matrix(0, n, n))
}
```

R has a slightly more formal built-in OOP system called S4. It's pretty good, but not everyone uses it.

```
setClass("board", representation(grid = "matrix"))
construct_board = function(n){ #explicit constructor
  new("board", grid = matrix(0, n, n))
}
```

```
> obj = construct_board(5)
> obj
An object of class "board"
Slot "grid":
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    0    0
[2,]    0    0    0    0    0
[3,]    0    0    0    0    0
[4,]    0    0    0    0    0
[5,]    0    0    0    0    0
```

```
> class(obj)
[1] "board"
attr(,"package")
[1] ".GlobalEnv"
> typeof(obj)
[1] "S4"
> getSlots("board")
    grid
"matrix"
```

R has a slightly more formal built-in OOP system called S4. It's pretty good, but not everyone uses it.

```
percolate <- function(x) NA
setGeneric("percolate")

setMethod("percolate", signature(x = "board"),
          function(x){ TRUE })
setMethod("percolate", signature(x = "matrix"),
          function(x){ FALSE })
```

Implicit type-checking!!

R has a slightly more formal built-in OOP system called S4. It's pretty good, but not everyone uses it.

```
percolate <- function(x) NA
setGeneric("percolate")

setMethod("percolate", signature(x = "board"),
          function(x){ TRUE })
setMethod("percolate", signature(x = "matrix"),
          function(x){ FALSE })
```

Implicit type-checking!!

```
> percolate(obj)
[1] TRUE
> percolate(matrix(0, 5, 5))
[1] FALSE
> percolate("asdf")
[1] NA
```

R has a slightly more formal built-in OOP system called S4. It's pretty good, but not everyone uses it.

```
percolate <- function(x) NA
setGeneric("percolate")

setMethod("percolate", signature(x = "board"),
          function(x){ TRUE })
setMethod("percolate", signature(x = "matrix"),
          function(x){ FALSE })
```

Implicit type-checking!!

```
> percolate(obj)
[1] TRUE
> percolate(matrix(0, 5, 5))
[1] FALSE
> percolate("asdf")
[1] NA
```

R has a slightly more formal built-in OOP system called S4. It's pretty good, but not everyone uses it.

```
#S4 way of doing multiple dispatch
compare <- function(x, y) NA
setGeneric("compare")

setMethod("compare", signature(x = "board", y = "board"),
          function(x, y){print("Comparing 2 boards")})
setMethod("compare", signature(x = "board", y = "matrix"),
          function(x, y){print("Comparing board to matrix")})
setMethod("compare", signature(x = "matrix", y = "matrix"),
          function(x, y){print("Comparing 2 matrices")})
```

R has a slightly more formal built-in OOP system called S4. It's pretty good, but not everyone uses it.

```
> obj1 = construct_board(5)
> obj2 = construct_board(5)
> compare(obj1, obj2)
[1] "Comparing 2 boards"
> compare(obj1, matrix(0, 5, 5))
[1] "Comparing board to matrix"
> compare(matrix(0, 5, 5), matrix(0, 5, 5))
[1] "Comparing 2 matrices"
```

R has a slightly more formal built-in OOP system called S4. It's pretty good, but not everyone uses it.

```
#We would've needed to something messier in S3
compare = function(x, y){
  if(class(x) == "board" & class(y) == "board"){
    print("Comparing 2 boards"); invisible()
  } else if(class(x) == "board" & is.matrix(y)){
    print("Comparing board to matrix"); invisible()
  } else if(is.matrix(x) & is.matrix(y)){
    print("Comparing 2 matrices"); invisible()
  } else {
    NA
  }
}
```

R has a slightly more formal built-in OOP system called S4. It's pretty good, but not everyone uses it.

```
# or to define explicitly a new function for every possible
#  type of input...
compare_2boards = function(x, y){
  if(class(x) == "board" & class(y) == "board"){
    print("Comparing 2 boards"); invisible()
  } else NA
}
compare_board_to_mat = function(x, y){
  if(class(x) == "board" & is.matrix(y)){
    print("Comparing board to matrix"); invisible()
  } else NA
}
compare_2mat = function(x, y){
  if(is.matrix(x) & is.matrix(y)){
    print("Comparing 2 matrices"); invisible()
  } else NA
}
```

R has a slightly more formal built-in OOP system called S4. It's pretty good, but not everyone uses it.

```
> # S4 objects can still be arbitrarily modified
though...
> obj@grid[1,3] = "lol"
> obj
An object of class "board"
Slot "grid":
     [,1] [,2] [,3]  [,4] [,5]
[1,] "0"  "0"  "lol" "0"  "0"
[2,] "0"  "0"  "0"   "0"  "0"
[3,] "0"  "0"  "0"   "0"  "0"
[4,] "0"  "0"  "0"   "0"  "0"
[5,] "0"  "0"  "0"   "0"  "0"
```

R has a slightly more formal built-in OOP system called S4. It's pretty good, but not everyone uses it.

```
# inheritance example
setClass("polygon", representation(color = "character"),
         prototype = list(color = "blue"))
setClass("triangle", representation(length = "numeric"),
         contains = "polygon")
setClass("circle", representation(radius = "numeric"),
         contains = "polygon")

area = function(x) NA
setGeneric("area")
setMethod("area", signature("triangle"),
          function(x) sqrt(3)*x@length^2/4)
setMethod("area", signature("circle"),
          function(x) pi*x@radius^2)

color = function(x) NA
setGeneric("color")
setMethod("color", signature("polygon"), function(x) x@color)
```

R has a slightly more formal built-in OOP system called S4. It's pretty good, but not everyone uses it.

```
> tri = new("triangle", length = 3)
> color(tri)
[1] "blue"
> area(tri)
[1] 3.897114
```

```
> circ = new("circle", radius = 4, color = "red"
)
> color(circ)
[1] "red"
> area(circ)
[1] 50.26548
```

The “formal” OOP in R is using a package called R6, which makes you feel like you’re writing in a real OOP language.

- You define “public” and “private” functions, whereby the private functions can only be used within the class (i.e., the user cannot access them).

The “formal” OOP in R is using a package called R6, which makes you feel like you’re writing in a real OOP language.

- You define “public” and “private” functions, whereby the private functions can only be used within the class (i.e., the user cannot access them).

- Inheritance allows multiple dispatch as long as you build your classes properly.

- See https://adv-r.hadley.nz/r6.html

The “formal” OOP in R is using a package called R6, which makes you feel like you’re writing in a real OOP language.

```
library(R6)
board <- R6Class("board",
                 public = list(
                   initialize = function(n) {
                     private$grid <- matrix(0, n, n)
                   },
                   print = function(...){
                     cat("This is a board of size", nrow(private$grid), ":\n")
                     print(private$grid)
                   }),

                 private = list(grid = NA)
)
```

The “formal” OOP in R is using a package called R6, which makes you feel like you’re writing in a real OOP language.

```
> obj = board$new(5)
> obj
This is a board of size 5 :
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    0    0
[2,]    0    0    0    0    0
[3,]    0    0    0    0    0
[4,]    0    0    0    0    0
[5,]    0    0    0    0    0
```

```
> class(obj)
[1] "board" "R6"
> typeof(obj)
[1] "environment"
```

The “formal” OOP in R is using a package called R6, which makes you feel like you’re writing in a real OOP language.

```
board <- R6Class("board",
                 public = list(
                   initialize = function(n) {
                     private$grid <- matrix(0, n, n)
                   },
                   print = function(...){
                     cat("This is a board of size", nrow(private$grid), ":\n")
                     print(private$grid)
                   },
                   set = function(val, idx){
                     private$grid[idx] = val
                   }),

                 private = list(grid = NA)
)
```

The “formal” OOP in R is using a package called R6, which makes you feel like you’re writing in a real OOP language.

```
> obj$set(5, 11)
> obj
This is a board of size 5 :
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    5    0    0
[2,]    0    0    0    0    0
[3,]    0    0    0    0    0
[4,]    0    0    0    0    0
[5,]    0    0    0    0    0
```

The “formal” OOP in R is using a package called R6, which makes you feel like you’re writing in a real OOP language.

```
> obj$grid
NULL
> obj$grid = 5
Error in obj$grid = 5 : cannot add bindings to a locked
environment
> obj@grid
Error: trying to get slot "grid" from an object (class "
board") that is not an S4 object
```

```
> class(obj) = "funny_class"
> obj
<environment: 0x10dcc70a0>
attr(,"class")
[1] "funny_class"
```

(Base) R is not really an OOP, as we've seen via S3. But it's not a functional language either.

- See https://arxiv.org/pdf/1409.3531.pdf, "Object-Oriented Programming, Functional Programming and R"

This paper examines two of the most significant paradigms in programming languages generally: object-oriented programming (OOP) and functional programming. R makes use of both, but in its own way. Both paradigms are valuable for serious programming with the language.

Is R a functional programming language in this sense? No. The structure of the language does not enforce functionality;

Is R an OOP language? Not from its inception, but it has added important software reflecting the ideas.

So… what should I use?

- Most users of R only know of S3, so if you want your software widely used, it's best to code in S3.

So… what should I use?

- Most users of R only know of S3, so if you want your software widely used, it's best to code in S3.

- Extending functionality of another programmer's code because "harder and harder" the more you closely abide by the principles of OOP.