

Introduction to Object-Oriented Programming and S3 System in R

S.Ravichandran (<https://github.com/ravichas/OOP-S3-in-R>)

April 27, 2019

based on Carl Broman's code

Preliminary information about object types in R

Let us create a logical object, x.

```
(x <- TRUE) # logical
```

```
## [1] TRUE
```

```
print(class(x))
```

```
## [1] "logical"
```

Let us create a list, also called x.

```
(x <- list(nums = 1:10,  
           chars = c("one", "two", "three"),  
           ints = c(1L, 2L, 3L)  
           ))
```

```
## $nums
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

```
##
```

```
## $chars
```

```
## [1] "one" "two" "three"
```

```
##
```

```
## $ints
```

```
## [1] 1 2 3
```

```
print(class(x))
```

```
## [1] "list"
```

BMI is a data.frame with four variables, Gender, Height, Weight and Age.

```
(BMI <- data.frame(  
  Gender = c("Male", "Male", "Female"),  
  Height = c(153.1, 173.6, 165.0),  
  Weight = c(81, 93, 78),  
  Age = c(42, 38, 26)  
))
```

```
##   Gender Height Weight Age
```

```
## 1   Male  153.1    81  42
```

```
## 2   Male  173.6    93  38
```

```
## 3 Female  165.0    78  26
```

```
print(class(BMI))
```

```
## [1] "data.frame"
```

Hands-on 1

One of the important concept of OOP is functions can respond in different ways depending on the input object type. To explain this concept, let us create the following objects:

- Numeric vector of 10 random numbers
- Categorical vector of length 6
- A linear model object

First, let us create a numerical vector with 10 elements.

```
set.seed(111)
```

```
(x_num <- rnorm(10) )
```

```
## [1] 0.2352 -0.3307 -0.3116 -2.3023 -0.1709 0.1403 -1.4974 -1.0102
```

```
## [9] -0.9485 -0.4940
```

Next, we build a categorical vector with 6 elements.

```
(x_fac <- factor(c("A", "B", "A", "C", "A", "B")))
```

```
## [1] A B A C A B
```

```
## Levels: A B C
```

Finally, let us create a linear model variable. But, first let us create two variables x and y

```
# setting seed
```

```
set.seed(123)
```

```
(x <- 1:10)
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

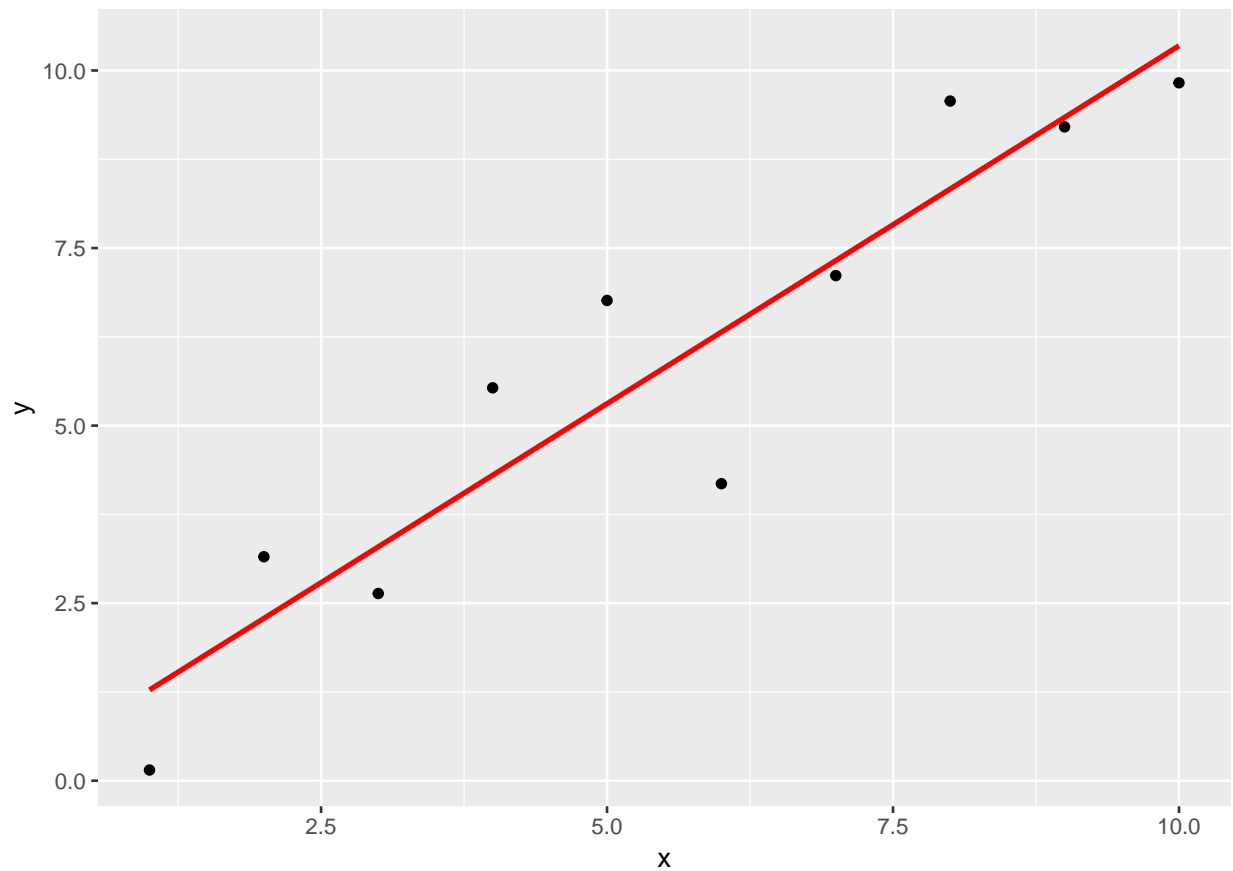
```
(y <- jitter(x, amount = 2))
```

```
## [1] 0.1503 3.1532 2.6359 5.5321 6.7619 4.1822 7.1124 9.5697 9.2057 9.8265
```

```
data.frame(x, y) %>% ggplot(aes(x, y)) +
```

```
  geom_point() +
```

```
  geom_smooth(method = "lm", col = "red", se = FALSE)
```



Build a model

```
model <- lm(y ~ x)
model
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Coefficients:
## (Intercept)          x
##          0.27          1.01
```

Behavior of summary function on different class of objects

```
x_num
```

```
## [1]  0.2352 -0.3307 -0.3116 -2.3023 -0.1709  0.1403 -1.4974 -1.0102
## [9] -0.9485 -0.4940
```

```
summary(x_num)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.302  -0.995  -0.412  -0.669  -0.206   0.235
```

```
x_fac
```

```
## [1] A B A C A B
## Levels: A B C
```

```
summary(x_fac)
```

```
## A B C
## 3 2 1
```

```
model
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Coefficients:
## (Intercept)          x
##          0.27          1.01
```

```
summary(model)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.135 -0.624 -0.173  1.140  1.453
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.270      0.854    0.32   0.76
## x             1.008      0.138    7.32 8.2e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.25 on 8 degrees of freedom
## Multiple R-squared:  0.87,    Adjusted R-squared:  0.854
## F-statistic: 53.6 on 1 and 8 DF,  p-value: 8.22e-05
```

How does R distinguish types of variables?

what command(s) can be used for this task?

```
# matrix
(int_mat <- matrix(1:12, nrow = 4, ncol = 3)) # column major
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

```
# determine the variable
class(int_mat) # obj is a matrix
```

```
## [1] "matrix"
```

```

# what type of matrix (elements are of what type)
typeof(int_mat) # int matrix; content of the matrix

## [1] "integer"

(float_mat <- matrix(rnorm(12), nrow = 4, ncol = 3))

##          [,1]      [,2]      [,3]
## [1,]  1.7151 -0.4457  0.1107
## [2,]  0.4609  1.2241 -0.5558
## [3,] -1.2651  0.3598  1.7869
## [4,] -0.6869  0.4008  0.4979

class(float_mat) # matrix

## [1] "matrix"

typeof(float_mat) # double; type of var that makes up matrix

## [1] "double"

# c code; in C floating point #s are double

```

Hands-on 2:

Interrogation of objects to see whether they are S3 objects

```

(int_mat <- matrix(1:12, nrow = 4, ncol = 3)) # column major

##          [,1] [,2] [,3]
## [1,]      1      5      9
## [2,]      2      6     10
## [3,]      3      7     11
## [4,]      4      8     12

sloop::otype(int_mat) # package::command(object)

## [1] "base"

head(mtcars)

##           mpg  cyl  disp  hp  drat    wt  qsec vs  am  gear  carb
## Mazda RX4    21.0   6  160  110 3.90 2.620 16.46 0   1    4    4
## Mazda RX4 Wag 21.0   6  160  110 3.90 2.875 17.02 0   1    4    4
## Datsun 710    22.8   4  108   93 3.85 2.320 18.61 1   1    4    1
## Hornet 4 Drive 21.4   6  258  110 3.08 3.215 19.44 1   0    3    1
## Hornet Sportabout 18.7   8  360  175 3.15 3.440 17.02 0   0    3    2
## Valiant      18.1   6  225  105 2.76 3.460 20.22 1   0    3    1

sloop::otype(mtcars)

## [1] "S3"

```

S3 & R6: How to assign classes?

- Can I override the class?
- Yes
- And as expected, it won't break the functionality

- Can I woverride the type?
- No

```
x_num

## [1]  0.2352 -0.3307 -0.3116 -2.3023 -0.1709  0.1403 -1.4974 -1.0102
## [9] -0.9485 -0.4940

class(x_num)

## [1] "numeric"

typeof(x_num)

## [1] "double"

(class(x_num) <- "random-numbers")

## [1] "random-numbers"
# the class that we have added has become an attribute
x_num

## [1]  0.2352 -0.3307 -0.3116 -2.3023 -0.1709  0.1403 -1.4974 -1.0102
## [9] -0.9485 -0.4940
## attr("class")
## [1] "random-numbers"
# we cannot override typeof
typeof(x_num)

## [1] "double"

is.numeric(x_num) # no matter what the class says

## [1] TRUE
```

S3 & R6: Function overloading

S3 exists so that we dont have to write many many functions to take care of different data types.

How does it work?

- S3 splits a function into generic and method functions.
- Methods named generic.class (Ex. print.Date)

Example of generic functions are print, summary etc.

```
(x_Date <- Sys.Date()) # "YYYY-MM-DD"

## [1] "2019-05-08"

class(x_Date) # "Date"

## [1] "Date"

print(x_Date) # "YYYY-MM-DD", 2019-03-26

## [1] "2019-05-08"
# is same as calling print.Date
print.Date(x_Date)

## [1] "2019-05-08"
```

```
# Let us explore the print function
print
```

```
## function (x, ...)
## UseMethod("print")
## <bytecode: 0x00000000189fb968>
## <environment: namespace:base>
```

print function is just a simple one line function. You can ignore the last two lines that shows the memory location and the object environment. print function calls UseMethod("print") to provide the final output.

What methods exist for a generic function?

- For example, for the generic function what methods are available
- generic.class1, generic.class2, generic.class3

Exmaple. print (generic), print.data.frame, print.Date etc.

```
head(methods(print)) # too many methods
```

```
## [1] "print.acf"      "print.AES"      "print.all_vars" "print.anova"
## [5] "print.any_vars" "print.aov"
```

What methods are available for a given class of an object?

- The methods could be coming from different generic classes. For example, generic1.class, generic2.class etc.
- Note this methods call for this case will return both S3 and s4 objects.

```
# gives both S3 and S4
methods(class = "lm") # or methods(class=lm)
```

```
## [1] add1      alias      anova      case.names
## [5] coerce    confint    cooks.distance deviance
## [9] dfbetas   dfbetas    drop1      dummy.coef
## [13] effects   extractAIC family      formula
## [17] fortify    hatvalues  influence   initialize
## [21] kappa     labels     logLik      model.frame
## [25] model.matrix nobs      plot        predict
## [29] print      proj       qr          residuals
## [33] rstandard rstudent  show        simulate
## [37] slotsFromS3 summary    variable.names vcov
## see '?methods' for accessing help and source code
```

```
# of them which ones are S3
.S3methods(class = "lm")
```

```
## [1] add1      alias      anova      case.names
## [5] confint    cooks.distance deviance    dfbetas
## [9] dfbetas    drop1      dummy.coef effects
## [13] extractAIC family      formula     fortify
## [17] hatvalues  influence   kappa       labels
## [21] logLik     model.frame model.matrix nobs
## [25] plot       predict     print       proj
## [29] qr         residuals   rstandard   rstudent
## [33] simulate   summary     variable.names vcov
```

```
## see '?methods' for accessing help and source code
```

Is the object/function generic or method?

```
pryr::is_s3_generic("print") # TRUE
```

```
## [1] TRUE
```

```
pryr::is_s3_method("print") # FALSE
```

```
## [1] FALSE
```

```
pryr::is_s3_method("print.Date") # TRUE
```

```
## [1] TRUE
```

Let us define our object.

```
(people <- c("Frank Blanchard",  
            "Andrea Gnuschke",  
            "Max Cole",  
            "Maryellen Hackett",  
            "Victoria Brun",  
            "Jonathan Summers",  
            "Christopher Worthington",  
            "Samuel Lopez",  
            "Richard Frederickson",  
            "Chris Hu") )
```

```
## [1] "Frank Blanchard"      "Andrea Gnuschke"  
## [3] "Max Cole"              "Maryellen Hackett"  
## [5] "Victoria Brun"         "Jonathan Summers"  
## [7] "Christopher Worthington" "Samuel Lopez"  
## [9] "Richard Frederickson"  "Chris Hu"
```

```
class(people)
```

```
## [1] "character"
```

```
(class(people) <- "InsiteGroup")
```

```
## [1] "InsiteGroup"
```

Suppose, we want to write an S3 function that gets the first name from the InsiteGroup object.

```
GetFirst <- function(obj) {  
  UseMethod("GetFirst",obj)  
}  
  
# create methods function  
GetFirst.InsiteGroup <- function(obj) {  
  return(obj[1])  
}  
  
# create default function  
GetFirst.default <- function(obj){  
  cat("This is a generic class\n")  
  # do something
```



```
}
methods(class = "InsiteGroup")
```

```
## [1] GetFirst
## see '?methods' for accessing help and source code
```

```
GetFirst(people)
```

```
## [1] "Frank Blanchard"
```

If no suitable methods can be found for a generic, then an error is thrown. For example, at the moment, `get_n_elements()` only has 2 methods available. If you pass a `data.frame`/matrix to `get_n_elements()` instead, you'll see an error. One could use `generic.default` to deal with all the missing class of objects.

Can variables have more than one class?

```
(human <- "laugh")
```

```
## [1] "laugh"
```

```
# less specific to more specific; final default class, character
class(human) <- c("mammalia", "eukaryota", "character")
```

```
# create a generic method for who_am_i
who_am_i <- function(x, ...) {
  UseMethod("who_am_i")
}
```

```
# create mammalia method for who_am_i
who_am_i.mammalia <- function(x, ...) {
  # let us write a message
  message("I am a Mammal")
  NextMethod("x")
}
```

```
# create eukarota method for who_am_i
who_am_i.eukaryota <- function(x, ...) {
  # let us write a message
  message("I am a Eukaryote")
  NextMethod("x")
}
```

```
# finally one for character method
who_am_i.character <- function(x, ...) {
  # let us write a message
  message("I am a simple character!")
  # since this is the last, no NextMethod
}
```

```
# call human to see all the 3 messages are displayed
class(human)
```

```
## [1] "mammalia" "eukaryota" "character"
```

```
who_am_i(human)
```

```
## I am a Mammal
```

```
## I am a Eukaryote
```

```
## I am a simple character!
```