Programming in Data science
Intro to Functions

Asad Waqar Malik

Dept. of Information Systems
Faculty of Computer Science and Information Technology
University of Malaya
Malaysia

Oct 17, 2019

Lecture Outline

1. `Functions`
2. Function with Default Values
3. Re-factoring code
4. Dependency Checking
5. Vectorization
6. Exercise

Outline

Functions

- One of the great strengths of R is the user's ability to add functions.
- In fact, many of the functions in R are actually functions of functions.
- The structure of a function is given below.

```
1  myfunction <- function(arg1, arg2, ... ){
2  statements
3  return(object)
4  }
```

Functions Defination

- The development of a functions in R represents the next level of R programming, beyond executing commands at the command line and writing scripts containing multiple R expressions.
- When writing R functions, one has to consider the following things.
  - Functions are used to encapsulate a sequence of expressions that are executed together to achieve a specific goal.
  - A single function typically does "one thing well".
  - The user will desire the ability to modify certain aspects of your code to match their specific needs or application.
  - Aspects of your code that can be modified often become function arguments that can be specified by the user.
  - When writing any function it's important to ask *what will the user want to modify in this function?*
  - Ultimately, the answer to this question will lead to the function's interface.

Defining a Function

- Let's start by defining a function fahrenheit_to_celsius that converts temperatures from Fahrenheit to Celsius:

```
1  fahrenheit_to_celsius <- function(temp_F) {
2    temp_C <- (temp_F - 32) * 5 / 9
3    return(temp_C)
4  }
```

- Define fahrenheit_to_celsius by assigning it to the output function.
- List of argument names are contained within parentheses.
- The body of the function–the statements that are executed is contained within curly braces ({}).
- When call the function, the values pass to it are assigned to argument variables.
- Use a return statement to send a result back.

Functions

- In R, it is not necessary to include the return statement.
- R automatically returns whichever variable is on the last line of the body of the function.
- Its a good practise to explicitly define the return statement.

Function Calling!

- Let's try running our function. Calling our own function is no different from calling any other function:

```
1  # freezing point of water
2  > fahrenheit_to_celsius(32)
3  [1] 0
4
5  # boiling point of water
6  > fahrenheit_to_celsius(212)
7  [1] 100
```

Composing Functions

- Now that we've seen how to turn Fahrenheit into Celsius, it's easy to turn Celsius into Kelvin:

Try yourself!

- What about converting Fahrenheit to Kelvin? We could write out the formula, but we don't need to. Instead, we can compose the two functions we have already created:

```
1  fahrenheit_to_kelvin <- function(temp_F) {
2    temp_C <- fahrenheit_to_celsius(temp_F)
3    temp_K <- celsius_to_kelvin(temp_C)
4    return(temp_K)
5  }
```

Nesting Functions

- This example showed the output of fahrenheit_to_celsius assigned to temp_C, which is then passed to celsius_to_kelvin to get the final result. It is also possible to perform this calculation in one line of code, by "nesting" one function inside another, like so:

```
# freezing point of water in Fahrenheit
> celsius_to_kelvin(fahrenheit_to_celsius(32.0))
[1] 273.15
```

Create a Function

- Question: Write a function called highlight that takes two vectors as arguments, called content and wrapper, and returns a new vector that has the wrapper vector at the beginning and end of the content.

```
1  content <- c("Write", "programs", "for", "people", "not", ↩
       "computers")
2  wrapper <- c("Hello")
3  highlight(content, wrapper)
```

Create a Function

- Question: Write a function called highlight that takes two vectors as arguments, called content and wrapper, and returns a new vector that has the wrapper vector at the beginning and end of the content.

Solution

Available on request!

Multiple Return values from Function

- The return() function can return only a single object. If we want to return multiple values in R, we can use a list (or other objects) and return it.

```
1  > multi_return <- function() {
2  my_list <- list("color" = "red", "size" = 20, "shape" = "↩
       round")
3  return(my_list)
4  }
```

Create a Function

Question

Write a function called edges that returns a vector made up of just the first and last elements of its input?

```
1  > dryprinciple <- c("Dont", "repeat", "yourself", "or", "↩
       others")
2  > edges(dryprinciple)
3  [1] "Dont"   "others"
```

Default values

- Functions can accept arguments explicitly assigned to a variable name in the function call functionName(variable = value).

```
1  > input_1 <- 20
2  > mySum <- function(input_1, input_2 = 10) {
3    output <- input_1 + input_2
4    return(output)
5  }
```

```
1  Given the above code was run, which value does mySum(input↩
       _1 = 1, 3) produce?
2  Given the above code was run, which value does mySum(3) ↩
       produce?
3  Given the above code was run, which value does mySum(input↩
       _2 = 3) produce?
```

Function Example

⇒ This code counts the number of times the filehash package was downloaded on July 20, 2016.

Function Example

```
1  library(magrittr)
2  library(dplyr)
3  library(readr)
4  if(!file.exists("data/2016-07-20.csv.gz"))
5  {
6     download.file("http://cran-logs.rstudio.com/2016/↩
          2016-07-20.csv.gz","data/2016-07-20.csv.gz")
7  }
8  dset_csv <- read_csv("data/2016-07-20.csv.gz", col_types =↩
       "cciccccccci")
9  dset_csv %>% filter(package == "filehash") %>% nrow
10 View(dset_csv)
```

Function – Reading from drive

Reading from drive

```
1  dset_csv <- read_csv("E:/dataset/2016-07-20.csv.gz")
```

Output

```
1  > num_download("filehash", "2016-07-20")
2  Output:
3  [1] 179
```

Functions

$\Rightarrow$ Can we convert the above mention code into function call, that takes two parameters "Package name" and "date".

Calling

```
1  > num_download("filehash", "2016-07-20")
2  [1] 179
3  > num_download("Rcpp", "2016-07-19")
4  [1]  ???
```

Outline

Default Values

- num_download(), user must enter the date and package name each time the function is called
- default value means, if the date argument is not explicitly set by the user, the function can use the default value.

Sample

```
1  num_download <- function(pkgname, date = "2016-07-20") {
2      # Code remains same
3  }
4  # Calling
5  > num_download("Rcpp")
6  > num_download("Rcpp", "2016-07-19")
```

Outline

Re-factoring code

- The function written handles the task at hand in a more general manner.
- Lets take a closer look at the function and asking whether it is written in the most useful possible manner.
  - Construct the path to the remote and local log file
  - Download the log file (if it doesn't already exist locally)
  - Read the log file into R
  - Find the package and return the number of downloads

Re-factoring code

Sample Code

```
1  check_for_logfile <- function(date) {
2      year <- substr(date, 1, 4)
3      src <- sprintf("http://cran-logs.rstudio.com/%s/%s.csv↩
           .gz", year, date)
4      dest <- file.path("data", basename(src))
5      if(!file.exists(dest)) {
6        val <- download.file(src, dest, quiet = TRUE)
7        if(!val)
8          stop("unable to download file ", src)
9      }
10     print(dest)
11   }
```

Re-factoring code

Sample Code Continue

```
1  num_download <- function(pkgname, date = "2016-07-20") {
2      dest <- check_for_logfile(date)
3      cran <- read_csv(dest, col_types = "ccicccccci", ↩
           progress = FALSE)
4      cran %>% filter(package == pkgname) %>% nrow
5      }
```

Calling

num_download("filehash", "2016-07-20")

–num_download() function does not need to know anything about downloading or URLs or files.
–There is a function check_for_logfile() that just deals with getting the data to your computer.

Outline

Dependency Checking

- The num_downloads() function depends on the readr and dplyr packages.
- Without them installed, the function won't run.
- Sometimes it is useful to check to see that the needed packages are installed so that a useful error message can be provided for the user.

Sample Function

```
1  check_pkg_deps <- function() {
2    if(!require(readr)) {
3      message("installing the readr package")
4      install.packages("readr")
5    }
6    if(!require(dplyr))
7      stop("the dplyr package needs to be installed first")
8  }
```

Dependency Checking

- require(..) function is similar to library(..), however library(..) stops with an error if the package cannot be loaded whereas require() returns TRUE or FALSE depending on whether the package can be loaded or not.

Dependency Check

Function revisit

```
1    num_download <- function(pkgname, date = "2016-07-20") {
2      check_pkg_deps()
3      dest <- check_for_logfile(date)
4      cran <- read_csv(dest, col_types = "ccicccccci", ↩
           progress = FALSE)
5      cran %>% filter(package == pkgname) %>% nrow
6    }
```

Outline

Vectorization

- The function covered are not vectorized. This means that each argument must be a single value—a single package name and a single date.
- In R, it is a common paradigm for functions to take vector arguments and for those functions to return vector or list results.
- To vectorize this function is to allow the pkgname argument to be a character vector of package names. This way we can get download statistics for multiple packages with a single function.
- The two things we need to do are:
  - Adjust our call to filter(..) to grab rows of the data frame that fall within a vector of package names
  - Use a group_by() %>% summarize() combination to count the downloads for each package.

Vectorization

Sample Code

```
1  num_download <- function(pkgname, date = "2016-07-20") {
2      check_pkg_deps()
3      dest <- check_for_logfile(date)
4      cran <- read_csv(dest, col_types = "ccicccccci", ↩
           progress = FALSE)
5      cran %>% filter(package %in% pkgname) %>%
6      group_by(package) %>%
7      summarize(n = n())
8    }
```

Calling Method

```
1  num_download(c("filehash", "weathermetrics"))
```

Vectorization

Output

```
[1] "data/2016-07-20.csv.gz"
# A tibble: 2 x 2
  package            n
  <chr>          <int>
1 filehash         179
2 weathermetrics     7
```

Vectorization

Excercise

–Vectorizing the date argument is similarly possible, but it has the added complication that for each date you need to download another log file.

Argument Checking

- Checking the arguments supplied by the reader are proper to prevent confusing results or error messages from occurring later on in the function.
- num_download() function is expecting both the pkgname and date arguments to be character vectors.
- date argument should be a character vector of length 1.
- Check the class of an argument using `is.character()` and the length using the `length()`. function

Argument Checking

Sample Code

```
num_download <- function(pkgname, date = "2016-07-20") {
    check_pkg_deps()
    # Check arguments
    if(!is.character(pkgname))
      stop("pkgname should be character")
    if(!is.character(date))
      stop("date should be character")
    if(length(date) != 1)
      stop("date should be length 1")
    dest <- check_for_logfile(date)
    cran <- read_csv(dest, col_types = "ccicccccci", ↩
        progress = FALSE)
    cran %>% filter(package %in% pkgname) %>% group_by(↩
        package) %>% summarize(n = n())
  }
```

Outline

Exercise

Questions

```
1  Write a program that computes the average on a given ↩
       column. The function takes a data frame and column ↩
       name as an input and returns the mean of the mentioned↩
        column.
2
3  Revisit the above question and add that the mentioned ↩
       column should be numeric for the average calculation.
4
5  Complete the following function
6  f.sum <- function (x, y) {
7  # Write code
8  }
9  > f.sum(5, 10)
10 [1] 15
```

Exercise

Questions

Consider the function $f(x) = 2x^2 - 0.9x - 1$, and calculate $f(0)$ and $f(1.5)$

Exercise

Consider an experiment where the biomass has been measured for 8 random plants grown under certain conditions. The sample values are denoted y1,..., yn. Assume that we are interested in an estimate of the population average, denoted μ. It is well known that the sample mean $\bar{y}$ is an estimate of the population average. The sample mean is also the least squares estimate: Consider the sum of squared deviations from μ, regarded as a function of μ:

$$f(\mu) = (y1 - \mu)^2 + \cdots + (yn - \mu)^2$$

This function has its minimum for $\mu = \bar{y}$. Consider in the following the sample consisting of 24.7 32.5 22.6 23.9 19.6 21.6 19.9 20.9

121.2887

Exercise

$\Rightarrow$ Make a vector with the sample values, denoted y. Then make a function that takes μ (mu) as argument and calculates the f(μ). Call the function f.
$\Rightarrow$ Recall that the sample standard variance is defined as:

$$s^2 = \frac{1}{n-1}\left((y_1 - \bar{y})^2 + ... + (y_n - \bar{y})^2\right) \quad (1)$$

Write a function var(x) that computes the variance of a sample.

Self explore

Variable scope using Functions in R - A sample file is uploaded on Spectrum.