CALLING FUNCTIONS - SCOPING

DSC 205 - Advanced introduction to data science

Marcus Birkenkrahe

March 8, 2024

README

You will learn:

- How variable names are compartmentalized in R
- What the rules for naming arguments and objects are
- How R searches for arguments and variables
- How you can specify arguments when calling a function

Download the codealong and practice files from GitHub:

- tinyurl.com/4-R-codealong-org
- tinyurl.com/4-R-practice-org

The first practice file is code along while I lecture, the second practice file is an independent exercise. Solutions in the pdf repo.

Scoping

- Scoping rules determine how R stores and retrieves objects
- Applied e.g. when handling duplicate object names
- Example: `data` as a function parameter, and as a function -
  1. create a row-wise 3x3 matrix of numbers {1..9}
  2. list all built-in datasets in the MASS package

```
## create row-wise 2x2 matrics of 1...9
#matrix(data=1:9, nrow=3, byrow=TRUE)

## list all datasets in the MASS package
library(MASS) ## load MASS
data(package='MASS') ## list datasets in MASS

Data sets in package 'MASS':

Aids2                   Australian AIDS Survival Data
Animals                 Brain and Body Weights for 28 Species
Boston                  Housing Values in Suburbs of Boston
Cars93                  Data from 93 Cars on Sale in the USA in 1993
Cushings                Diagnostic Tests on Patients with Cushing's Syndrome
DDT                     DDT in Kale
GAGurine                Level of GAG in Urine of Children
Insurance               Numbers of Car Insurance claims
Melanoma                Survival from Malignant Melanoma
OME                     Tests of Auditory Perception in Children with OME
Pima.te                 Diabetes in Pima Indian Women
Pima.tr                 Diabetes in Pima Indian Women
Pima.tr2                Diabetes in Pima Indian Women
Rabbit                  Blood Pressure in Rabbits
Rubber                  Accelerated Testing of Tyre Rubber
SP500                   Returns of the Standard and Poors 500
Sitka                   Growth Curves for Sitka Spruce Trees in 1988
Sitka89                 Growth Curves for Sitka Spruce Trees in 1989
Skye                    AFM Compositions of Aphyric Skye Lavas
Traffic                 Effect of Swedish Speed Limits on Accidents
UScereal                Nutritional and Marketing Information on US Cereals
UScrime                 The Effect of Punishment Regimes on Crime Rates
VA                      Veteran's Administration Lung Cancer Trial
abbey                   Determinations of Nickel Content
accdeaths               Accidental Deaths in the US 1973-1978
anorexia                Anorexia Data on Weight Change
bacteria                Presence of Bacteria after Drug Treatments
beav1                   Body Temperature Series of Beaver 1
beav2                   Body Temperature Series of Beaver 2
biopsy                  Biopsy Data on Breast Cancer Patients
birthwt                 Risk Factors Associated with Low Infant Birth Weight
```

cabbages	Data from a cabbage field trial
caith	Colours of Eyes and Hair of People in Caithness
cats	Anatomical Data from Domestic Cats
cement	Heat Evolved by Setting Cements
chem	Copper in Wholemeal Flour
coop	Co-operative Trial in Analytical Chemistry
cpus	Performance of Computer CPUs
crabs	Morphological Measurements on Leptograpsus Crabs
deaths	Monthly Deaths from Lung Diseases in the UK
drivers	Deaths of Car Drivers in Great Britain 1969-84
eagles	Foraging Ecology of Bald Eagles
epil	Seizure Counts for Epileptics
farms	Ecological Factors in Farm Management
fgl	Measurements of Forensic Glass Fragments
forbes	Forbes' Data on Boiling Points in the Alps
galaxies	Velocities for 82 Galaxies
gehan	Remission Times of Leukaemia Patients
genotype	Rat Genotype Data
geyser	Old Faithful Geyser Data
gilgais	Line Transect of Soil in Gilgai Territory
hills	Record Times in Scottish Hill Races
housing	Frequency Table from a Copenhagen Housing Conditions Survey
immer	Yields from a Barley Field Trial
leuk	Survival Times and White Blood Counts for Leukaemia Patients
mammals	Brain and Body Weights for 62 Species of Land Mammals
mcycle	Data from a Simulated Motorcycle Accident
menarche	Age of Menarche in Warsaw
michelson	Michelson's Speed of Light Data
minn38	Minnesota High School Graduates of 1938
motors	Accelerated Life Testing of Motorettes
muscle	Effect of Calcium Chloride on Muscle Contraction in Ra Hearts
newcomb	Newcomb's Measurements of the Passage Time of Light
nlschools	Eighth-Grade Pupils in the Netherlands
npk	Classical N, P, K Factorial Experiment
npr1	US Naval Petroleum Reserve No. 1 data
oats	Data from an Oats Field Trial
painters	The Painter's Data of de Piles

petrol	N. L. Prater's Petrol Refinery Data
phones	Belgium Phone Calls 1950-1973
quine	Absenteeism from School in Rural New South Wales
road	Road Accident Deaths in US States
rotifer	Numbers of Rotifers by Fluid Density
ships	Ships Damage Data
shoes	Shoe wear data of Box, Hunter and Hunter
shrimp	Percentage of Shrimp in Shrimp Cocktail
shuttle	Space Shuttle Autolander Problem
snails	Snail Mortality Data
steam	The Saturated Steam Pressure Data
stormer	The Stormer Viscometer Data
survey	Student Survey Data
synth.te	Synthetic Classification Problem
synth.tr	Synthetic Classification Problem
topo	Spatial Topographic Data
waders	Counts of Waders at 15 Sites in South Africa
whiteside	House Insulation: Whiteside's Data
wtloss	Weight Loss Data from an Obese Patient

Environments

- R enforces scoping rules with virtual *environment*
- An environment is a separate compartment for data structures (like vectors) and functions (like `data`).
- Environments are *dynamic* - they can be created, manipulated and removed.
- Technically, an environment is a pointer to the memory location where the R objects are stored.
- There are three types of environments:
  1. **Global** environments
  2. **Package** environments and namespaces
  3. **Local** or lexical environments

Global environments

- Every object you've created or overwritten resides in the global environment of your R session.
- A call to `ls()` lists all objects, variables, and user-defined functions in the global environment
- **Example:** create three new objects and confirm their existence in the global environment:
  1. a `numeric` variable `foo`
  2. a `character` variable `bar`
  3. An anonymous (non-argument) function `hello`
  4. check the contents of the global environment with `ls`
  5. run `hello`

```
foo <- 4 + 5
bar <- "stringtastic"
hello <- function() print("hello, Marcus")
ls()
hello()
```

```
[1] "bar"   "foo"   "h"     "hello"
[1] "hello, Marcus"
```

Package environments and namespaces

- Package environments are items made available by each package in R.
- You can use `ls` to list the items in a package environment: for example, to list the content of built-in `datasets` (no functions)

```
ls("package:datasets")
```

```
  [1] "ability.cov"           "airmiles"              "AirPassengers"
  [4] "airquality"            "anscombe"              "attenu"
  [7] "attitude"              "austres"               "beaver1"
 [10] "beaver2"               "BJsales"               "BJsales.lead"
 [13] "BOD"                   "cars"                  "ChickWeight"
```

```
 [16] "chickwts"              "co2"                   "CO2"
 [19] "crimtab"               "discoveries"           "DNase"
 [22] "esoph"                 "euro"                  "euro.cross"
 [25] "eurodist"              "EuStockMarkets"        "faithful"
 [28] "fdeaths"               "Formaldehyde"          "freeny"
 [31] "freeny.x"              "freeny.y"              "HairEyeColor"
 [34] "Harman23.cor"          "Harman74.cor"          "Indometh"
 [37] "infert"                "InsectSprays"          "iris"
 [40] "iris3"                 "islands"               "JohnsonJohnson"
 [43] "LakeHuron"             "ldeaths"               "lh"
 [46] "LifeCycleSavings"      "Loblolly"              "longley"
 [49] "lynx"                  "mdeaths"               "morley"
 [52] "mtcars"                "nhtemp"                "Nile"
 [55] "nottem"                "npk"                   "occupationalStatus"
 [58] "Orange"                "OrchardSprays"         "PlantGrowth"
 [61] "precip"                "presidents"            "pressure"
 [64] "Puromycin"             "quakes"                "randu"
 [67] "rivers"                "rock"                  "Seatbelts"
 [70] "sleep"                 "stack.loss"            "stack.x"
 [73] "stackloss"             "state.abb"             "state.area"
 [76] "state.center"          "state.division"        "state.name"
 [79] "state.region"          "state.x77"             "sunspot.month"
 [82] "sunspot.year"          "sunspots"              "swiss"
 [85] "Theoph"                "Titanic"               "ToothGrowth"
 [88] "treering"              "trees"                 "UCBAdmissions"
 [91] "UKDriverDeaths"        "UKgas"                 "USAccDeaths"
 [94] "USArrests"             "UScitiesD"             "USJudgeRatings"
 [97] "USPersonalExpenditure" "uspop"                 "VADeaths"
[100] "volcano"               "warpbreaks"            "women"
[103] "WorldPhones"           "WWWusage"
```

Or to list the visible objects of the `graphics` package:

```
ls("package:graphics")
```

```
 [1] "abline"          "arrows"          "assocplot"       "axis"            "Axis
 [6] "axis.Date"       "axis.POSIXct"    "axTicks"         "barplot"         "barp
[11] "box"             "boxplot"         "boxplot.default" "boxplot.matrix"  "bxp'
[16] "cdplot"          "clip"            "close.screen"    "co.intervals"    "cont
[21] "contour.default" "coplot"          "curve"           "dotchart"        "eras
```

```
[26] "filled.contour"  "fourfoldplot"    "frame"           "grconvertX"      "grco
[31] "grid"            "hist"            "hist.default"    "identify"        "imag
[36] "image.default"   "layout"          "layout.show"     "lcm"             "lege
[41] "lines"           "lines.default"   "locator"         "matlines"        "matp
[46] "matpoints"       "mosaicplot"      "mtext"           "pairs"           "pair
[51] "panel.smooth"    "par"             "persp"           "pie"             "plot
[56] "plot.default"    "plot.design"     "plot.function"   "plot.new"        "plot
[61] "plot.xy"         "points"          "points.default"  "polygon"         "poly
[66] "rasterImage"     "rect"            "rug"             "screen"          "segm
[71] "smoothScatter"   "spineplot"       "split.screen"    "stars"           "stem
[76] "strheight"       "stripchart"      "strwidth"        "sunflowerplot"   "symb
[81] "text"            "text.default"    "title"           "xinch"           "xspl
[86] "xyinch"          "yinch"
```

- A package *namespace* allows the package writer to hide functions and data that are only for internal use, and stops functions from breaking when a user or another package writer uses a duplicate name.
- As an example, load (after installation) the `dplyr` package (don't print the content - it has 300 functions!) and run `dplyr::filter`.

```
library(dplyr)
dplyr::filter
```

```
function (.data, ..., .by = NULL, .preserve = FALSE)
{
    check_by_typo(...)
    by <- enquo(.by)
    if (!quo_is_null(by) && !is_false(.preserve)) {
abort("Can't supply both `.by` and `.preserve`.")
    }
    UseMethod("filter")
}
<bytecode: 0x5623f0b64810>
<environment: namespace:dplyr>
```

- If you look at the output (the definition of `filter` in this package, you notice an internal (`base`) function, `UseMethod`, which is not listed in the visible content of `dplyr`, and the name of the `namespace` environment.

- When loading `dplyr`, you were informed that `dplyr::filter` masks another function, `stats::filter`. This means that using `filter` without the namespace reverts to `dplyr::filter`. If you want to use the function of the same name in `stats`, you need to call `stats::filter`.

Local or lexical environments

- Each time a function is called, a new environment called *local* or *lexical* is created.
- It contains all objects and variables created in and visible to the function, including any arguments you've supplied during execution.
- Example: create a 2x2 `matrix` and pass in the argument `data`: "OMG", "LOL", "IMO", "YOLO":

```
youthspeak <- matrix(data = c("OMG", "LOL", "IMO", "YOLO"),
                     nrow=2, ncol=2)
youthspeak
```

```
     [,1]  [,2]
[1,] "OMG" "IMO"
[2,] "LOL" "YOLO"
```

- Calling `matrix` like this creates a local environment containing the `data` vector
- When you execute the function, it begins by looking for `data` in this local environment. It is not confused by other objects named `data`, such as `utils::data`.
- If a required item is not found in the local environment, R does begin to widen its search.
- Once the function has completed, the local environment is automatically removed. The same goes for `nrow` and `ncol`.

Search Path

- To access data structures and functions other than the immediate global environment (of user-created objects), R follows a *search path*.

- You can view the search path with `search()`:

```
search()
```

```
[1] ".GlobalEnv"        "package:dplyr"     "package:MASS"      "ESSR"
[5] "package:stats"     "package:graphics"  "package:grDevices" "package:utils"
[9] "package:datasets"  "package:methods"   "Autoloads"         "package:base"
```

- The path always begins at `.GlobalEnv` and ends after `base`. It stops if an object is found in any environment along the path.
- If it does not find what it wanted, the *empty environment* is reached.
- Example: let's see what happens when we create a vector with `seq`:
  1. create a vector of 5 elements with `seq`
  2. the values should lay between the (included) values 0 and 3

```
baz <- seq(from=0, to=3, length.out=5)
baz
```

```
[1] 0.00 0.75 1.50 2.25 3.00
```

- R searches `.GlobalEnv` for `seq`, goes through the list and finds it in `base`. `seq` is executed and `baz` is created in the global environment.
- In the subsequent call to `baz`, R finds it immediately in `.GlobalEnv`.
- You can look up the environment of any function using `environment`:

```
environment(seq)
environment(abline)
environment(filter)
environment(stats::filter)
```

```
<environment: namespace:base>
<environment: namespace:graphics>
<environment: namespace:dplyr>
<environment: namespace:stats>
```

- When a package is loaded with `library`, it is inserted in the search path right after the global environment, along with all its dependencies:

```
library('car')
search()
```

```
Error in library("car") : there is no package called ‘car’
 [1] ".GlobalEnv"        "package:dplyr"     "package:MASS"      "ESSR"
 [5] "package:stats"     "package:graphics"  "package:grDevices" "package:utils"
 [9] "package:datasets"  "package:methods"   "Autoloads"         "package:base"
```

- In the example, loading `car` lead to the inclusion of the function package and its accompanying dataset package: do you remember how to list the contents of `carData`?

```
ls('package:carData')
```

```
Error in as.environment(pos) :
  no item called "package:carData" on the search list
```

- An error is thrown if you request a function or object
  - that you haven't **defined**,
  - that doesn't **exist**,
  - that is in a contributed package that you've forgotten to **load**

```
neither.here()  # undefined function
nor.there       # undefined object
```

```
Error in neither.here() : could not find function "neither.here"
Error: object 'nor.there' not found
```

- Read Gupta (2012) for more details on R environments. (This would also make an excellent term project topic.)

Reserved and protected names

- Key terms that are forbidden from being used as R object names:
  - `if` and `else`
  - `for`, `while`, and `in`
  - `repeat`, `break`, and `next`
  - `TRUE`, and `FALSE`
  - `Inf` and `-Inf`
  - `NA`, `NaN`, and `NULL`
- The first four line items are the core tools for programming in R, followed by Boolean values and special values.
- What happens when you assign a value to an `NaN`?

```
NaN <- 5
```

```
Error in NaN <- 5 : invalid (do_set) left-hand side to assignment
```

- Since R is case-sensitive, you can assign values to case variants of these keywords, causing much confusion:

```
False <- "confusing"
nan <- "this"
inf <- "is"
Null <- "very"
paste(nan,inf,Null,False)
```

```
[1] "this is very confusing"
```

- `T` and `F` can also be overwritten - don't do it since they are the abbreviations for `TRUE` and `FALSE`:

```
T <- FALSE
F <- TRUE
paste(T,"is",F)
paste("2+2=5 is", (2+2==5) == T)
(2+2==5) == TRUE
```

```
[1] "FALSE is TRUE"
[1] "2+2=5 is TRUE"
[1] FALSE
```

- With all these confusing changes, clear the global environment now!

```
ls()
rm(list=ls())  ## remove the list of user-defined R objects
ls()
```

```
 [1] "bar"        "baz"        "F"          "False"      "foo"        "h"
 [7] "hello"      "inf"        "nan"        "Null"       "T"          "youthspeak'
character(0)
```

Glossary

TERM	MEANING
Scoping	Rules of storing/retrieving objects
Environment	Virtual compartment for data and functions
Global environment	All user-created objects
Package environments	Objects contained in packages
Namespace	Defines visibility of package functions
	E.g. in `base::` for the `base` package
`ls()`	List global environment
`ls(package:base)`	List functions in the `base` package
Local environment	Objects created when function is called
Search path	List of environments searched, `search()`
`matrix`	Create matrix
`seq`	Create numerical sequence vector
`base::data`	List or load dataset
`NaN`	Not a number
`Inf`	Infinite numerical value
`NA`	Missing value
`NULL`	Null object - returned when value undefined
`paste`	Paste arguments together as string
`rm`	Remove R objectts, e.g. `rm(list=ls())`

References

- Gupta, S. (Mar 29, 2012). How R Searches and Finds Stuff. URL: blog.thatbuthow.com.