**R Style Guide  
Jesse Lecy, Syracuse University**

**Naming Conventions**

**File Names**

The file name should end in .R or .r

Make them meaningful

|  |
| --- |
| # GOOD  Mackey Clustering Algorithm.R  # BAD  homework.r  ty2.r |

If the file is a function that will be sourced, name the file the same as the function

|  |
| --- |
| # Function  plotInColor <- function( x, y )  {  plot( x, y, col=factor(x) )  return( NULL )  }  # File name  plotInColor.r |

Avoid spaces in file names, as these can cause problems on GitHub!

**Variable Names**

There is agreement on some naming conventions, such as use nouns to name variable and datasets; use verbs to name functions.

There is disagreement, though, over the specific syntax style for functions in R. You can see this across conventions used in each package. So these rules are far from universal, but it is useful to get in the habit of distinguishing between data names and function names in your own code. My suggestion is to use all lowercase letters with words separated by periods for names of variables and datasets:

|  |
| --- |
| # GOOD  my.data.frame <- matrix( rnorm(1000), nrow=100 )  dat.atl <- dat[ dat$FIPS %in% atl.fips , ]  lm.01 <- lm( y ~ x )  # BAD  MyDataFrame # use camel caps for functions  my\_data\_frame # separate by periods  My.data # don’t use upper case  01.lm <- lm( y ~ x ) # R won’t allow names to start with num’s |

**Function Names**

Use verbs that describe what the functions do. Use camelCaps to differentiate functions from datasets in your code.

|  |
| --- |
| # GOOD  makePretty <- function( ) { … }  subsetMyData <- function( ) { … }  # BAD  make.pretty <- function( ) { … } # use camelCaps  prettyGraph <- function( ) { … } # should be a verb |

**Spacing**

Place spaces around all operators: = , <- , + , - , / , > etc.

Place spaces after commas, but not before

Places spaces after the parenthesis in a function and after brackets [ ]

|  |
| --- |
| # GOOD  x <- intersect( z1, z2 )  table.01 <- tapply( y, x1, mean, na.rm=T )  # BAD  x<-intersect(z1,z2)  table.01 <- tapply(y,x1,mean,na.rm=T)  table (x1,x2) |

It doesn’t hurt to add an extra space before a comma in the middle of a data frame or matrix bracket set to make it clear that there are two indices.

|  |
| --- |
| # GOOD  dat[ 1:10 , 2:5 ]  sub.dat[ dat$EIN == i , ]  # BAD  dat[ 1:10,2:5 ]  sub.dat[dat$EIN==i,] |

**Indentation**

**Loops and Functions**

Use a three-space indentation to delineate a loop or the body of a function. The rule is recursive. Put curly brackets on their own line.

|  |
| --- |
| for( i in 1:10 )  {  print( i + 1 )  for( j in 1:5 )  {  print( i\*j )  count <- count + 1  } # end of j loop #  } # end of i loop # |

**Lists of Arguments**

Also use indentations to separate a long list of arguments in a function:

|  |
| --- |
| plot( x, y,  main = ”This is my graph title”,  xlab = ”The X Label”,  ylab = “The Y Axis”,  color = col.vector  ) |

**Documenting Functions**

A function is an input, output device. Directly above the function describe the input – what arguments the function accepts (including the data type of each), what the function does, and the output – what the function returns.

It is good practice to always include a return call in a function, even if it is just NULL. If the end of a function is reached without a return call R the value of the last evaluated expression is returned. So if you want the function to return nothing you need a NULL return.

|  |
| --- |
| # ===================================================  #  # The ‘plot.residual.colors’ function.  #  # Plots an x and y variable and adds color to the  # data points to indicate distance from the  # regression line.  #  # Arguments:  # x = a vector of numbers  # y = a vector of numbers  # res.col = number of color bands on the plot  #  # Returns: NULL  #  # ===================================================  plot.residual.colors <- function( x, y, num.cols=3 )  {  m01 <- lm( y ~ x )  cats <- cut( rank( abs( m01$residuals ) ), num.cols )    plot( x, y, col=cats )    return( NULL )  } # end of function # |

|  |
| --- |
| plot.residual.colors( x, y )    plot.residual.colors( x, y, 10 ) |
|  |

**Writing Scripts**

A script is a short program for data analysis. It is important to organize your scripts in a consistent manner. There are several things that should be included at the beginning of each script:

(1) Documentation which can include the purpose, author, copyright info, and version of the script

(2) Load all packages needed for the program

(3) Source any custom functions needed for the analysis

(4) Declare any universal variables (constants)

(5) Set any session options

|  |
| --- |
| # ======================================================  #  # Step 1 of analysis of tree frog growth rates.  # By: Keyser Söze  # Last updated: May 1, 2013  #  # Merges data on tree frogs with ecological data  # from the observation sites.  #  # ======================================================  library(foreign)  library(ggplot2)  source(“./Functions/estimateByGrid.R”)  options( digits=2 )  # Start your script here |

See below for instructions in how to organize your scripts into a directory structure that will streamline your work-flow.

**Miscellaneous Operators**

**Assignment**

R uses a specific throw-and-catch convention in order to make it easier to interact with data without writing a lot of print functions. If you type the name of an object or evaluate an expression, the default behavior is to print the object or result. If you want to save the changes, you need to include a ‘catch’ statement, i.e. assign the results to a new variable. Assignment in R is done through the **<-** operator. Technically the **=** operator works, but it is discouraged for assignment. Instead, it should be used for arguments inside of functions.

|  |
| --- |
| # GOOD  x <- 10  10 -> x # this is not as common, but allowed  plot( x=edu, y=income, main=”Education and Income” )    # use the = operator for arguments in a function  # BAD  x = 10 # this works but is discouraged  x < - 10 # be careful! this reads, x is less than -10 |

**Quote Marks**

In R you reference objects (datasets or functions that are loaded in your environment) by name directly, and you reference arguments and strings using quotation marks. R allows you to use single or double quotation marks.

Double quotes are encouraged in order to avoid a subtle bug that can creep into your code when working across platforms. The single quote ‘ is not the same as the prime ` even though they look similar. Some text processors will replace quotes with primes for style purposes. R can interpret the quotes, not the prime symbol, though.

|  |
| --- |
| x <- c(1,2,3)  > x # prints the object as expected  [1] 1 2 3    > "x" # prints the character “x” as expected  [1] "x"  > “x” # pretty quotes not recognized  Error: unexpected input in "“"  > 'x' # single quotes are interpreted same as double  [1] "x"  > `x` # unexpected behavior – ignores primes  [1] 1 2 3 |

~~~

**Examples of Other Style Guides**

<http://csgillespie.wordpress.com/2010/11/23/r-style-guide/>

<http://stat405.had.co.nz/r-style.html>

<http://google-styleguide.googlecode.com/svn/trunk/google-r-style.html>

**Resources**

News and tutorials from the R community:

<http://www.r-bloggers.com/>

NY Times blog on producing graphics with R:

<http://chartsnthings.tumblr.com/>

Quick-R, a great guide to basic analysis with examples:

<http://www.statmethods.net/>

A reference card for common R functions:

<http://cran.r-project.org/doc/contrib/Short-refcard.pdf>

Stack Overflow thread on R:

<http://stackoverflow.com/questions/tagged/r>

R packages sorted by topic:

<http://cran.r-project.org/web/views/>

Color guide:

<http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf>

Great Graphics Blog

[www.flowingdata.com](http://www.flowingdata.com)