RProgramming

tubako

8/30/2020

# R Console Input and Evaluation

### Entering input

x <- 1  
print(x)

## [1] 1

x

## [1] 1

msg <- "hello"  
  
## x <- ##Incomplete expression

### Evaluation

x <- 5 ## nothing happens  
x ## auto-printing occurs

## [1] 5

print(x) ## explicit printing

## [1] 5

## [1] indicates that x is a vector and 5 is the first element.

### Printing

## The : operator is used to create integer sequences.  
x <- 1:20   
x

## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

# Data Types

### R Objects and Attributes

#### Objects

R has 5 basic classes of objects

* character
* numeric (real numbers)
* integer
* complex
* logical (T/F)

The most basic object is a vector

* A vector can only contain objects of the same class
* BUT: The one exception is a *list*, which is represented as a vector but can contain objects of different classes (indeed, that’s usually why we use them)

Empty vectors can be created with the

vector()

## logical(0)

function.

#### Numbers

* Numbers in R are generally treated as numeric objects (i.e. double precision real numbers)
* If you explicitly want an integer, you need to specify the **L** suffix
* Ex: Entering 1 gives you a numeric object; entering 1**L** explicitly gives you an integer.
* There is also a special number **Inf** which represents infinity: e.g. 1/0; **Inf** can be used in ordinary calculations; e.g. 1/**Inf** is 0
* The value **NaN** represents an undefined value ("not a number); e.g. 0/0; **NaN** can also be thought of as a missing value

#### Attributes

R objects can have attributes

* names, dimensions
* dimensions (e.g. matrices, arrays)
* class
* length
* other user-defined attributes/metadata

Attributes of an object can be accessed using the

##attributes()

function.

### Vectors and Lists

#### Creating Vectors

The **c()** function can be used to create vectors of objects.

x <- c(0.5, 0.6) ## numeric  
x <- c(TRUE, FALSE) ## logical  
x <- c(T, F) ## logical  
x <- c("a", "b", "c") ## character  
x <- 9:29 ## integer  
x <- c(1+0i, 2+4i) ## complex

Using the **vector()** function

x <- vector("numeric", length = 10)  
x

## [1] 0 0 0 0 0 0 0 0 0 0

#### Mixing Objects

When different objects are mixed in a vector, *coercion* occurs so that every element in the vector is of the same class.

y <- c(1.7, "a") ## character  
y <- c(TRUE, 2) ## numeric  
y <- c("a", TRUE) ## character  
y

## [1] "a" "TRUE"

#### Explicit Coercion

Objects can be explicitly coerced from one class to another using the as.\* functions, if available.

x <- 0:6   
class(x)

## [1] "integer"

as.numeric(x)

## [1] 0 1 2 3 4 5 6

as.logical(x)

## [1] FALSE TRUE TRUE TRUE TRUE TRUE TRUE

as.character(x)

## [1] "0" "1" "2" "3" "4" "5" "6"

Nonsensical coercion results in **NA**s.

x <- c("a", "b", "c")  
as.numeric(x)

## Warning: NAs introduced by coercion

## [1] NA NA NA

as.logical(x)

## [1] NA NA NA

as.complex(x)

## Warning: NAs introduced by coercion

## [1] NA NA NA

#### Lists

x <- list(1, "a", TRUE, 1+4i)  
x

## [[1]]  
## [1] 1  
##   
## [[2]]  
## [1] "a"  
##   
## [[3]]  
## [1] TRUE  
##   
## [[4]]  
## [1] 1+4i

### Matrices

The dimension attribute is itself an integer vector of length 2 (nrow, ncol).

m <- matrix(nrow = 2, ncol = 3)  
m

## [,1] [,2] [,3]  
## [1,] NA NA NA  
## [2,] NA NA NA

dim(m)

## [1] 2 3

attributes(m)

## $dim  
## [1] 2 3

Matrices are constructed column wise, so entries can be thought of starting in the “upper left” corner and running down the coloumns.

m <- matrix (1:6, nrow= 2, ncol = 3)  
m

## [,1] [,2] [,3]  
## [1,] 1 3 5  
## [2,] 2 4 6

You can also create a matrix by creating the dimension attribute on a vector.

m <- 1:10  
m

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

dim (m) <- c(2, 5)  
m

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

### cbind-ing and rbind-ing

Matrices can be created by *column-binding* or *row-binding* with **cbind()** and **rbind()**.

x <- 1:3  
y <- 10:12  
cbind(x, y)

## x y  
## [1,] 1 10  
## [2,] 2 11  
## [3,] 3 12

rbind(x, y)

## [,1] [,2] [,3]  
## x 1 2 3  
## y 10 11 12

### Factors

Factors are used to represent categorical data. There’s two types of factor, there is unordered or ordered. One can think of a factor as an integer vector where each integer has a *label*.

* Factors are treated specially by modeling functions like **lm()** and **glm()**
* Using factors with labels is better than using using integers because factors are self-describing; having a variable that has values “Male” and “Female” is better than a variable that has values 1 and 2.

x <- factor(c("yes", "yes", "no", "yes", "no"))  
x

## [1] yes yes no yes no   
## Levels: no yes

table(x)

## x  
## no yes   
## 2 3

unclass(x)

## [1] 2 2 1 2 1  
## attr(,"levels")  
## [1] "no" "yes"

The order of the levels can be set using the **levels** argument to **factor()**. This can be important in linear modeling because the fist level is used as the baseline level.

x <- factor(c("yes", "yes", "no", "yes", "no"), level = c("yes", "no"))  
x

## [1] yes yes no yes no   
## Levels: yes no

unclass(x)

## [1] 1 1 2 1 2  
## attr(,"levels")  
## [1] "yes" "no"

### Missing Values

Missing values in R are denoted by either **NA** or **NAN**. **NAN** is used for undefined mathematical operations. And **NA** is used for everything else.

* **is.na()** is used to test objects if they are **NA**
* **is.nan()** is used to test for **NaN**
* **NA** values have a class also, so there are integer **NA** character **NA**, etc.
* A **NaN** value is also **NA** but the converse is not true.

x <- c(1, 2, NA, 10, 3)  
is.na(x)

## [1] FALSE FALSE TRUE FALSE FALSE

is.nan(x)

## [1] FALSE FALSE FALSE FALSE FALSE

x <- c(1, 2, NaN, NA, 4)  
is.na(x)

## [1] FALSE FALSE TRUE TRUE FALSE

is.nan(x)

## [1] FALSE FALSE TRUE FALSE FALSE

### Data Frames

Data frames are used to store tabular data.

* They are represented as a special type of list where every element of the list has to have the same length
* Each element of the list can be thought of as a column and the length of each element of the list is the number of rows
* Unlike matrices, data frames can store different classes of objects in each column (just like lists); matrices must have every element be the same class
* Data frames also have a special atttribute called row.names
* Data frames are usually created by calling **read.table()** or **read.csv()**
* Can be converted to a matrix by calling **data.matrix()**

x <- data.frame(foo = 1:4, bar = c(T, T, F, F))  
x

## foo bar  
## 1 1 TRUE  
## 2 2 TRUE  
## 3 3 FALSE  
## 4 4 FALSE

nrow(x)

## [1] 4

ncol(x)

## [1] 2

### Names Attribute

#### Names

R objects can also have names, which is very useful for writing readable code and self-describing objects.

x <- 1:3  
names(x)

## NULL

names(x) <- c("foo", "bar", "norf")  
x

## foo bar norf   
## 1 2 3

names(x)

## [1] "foo" "bar" "norf"

Lists can also have names.

x <- list(a = 1, b = 2, c = 3)  
x

## $a  
## [1] 1  
##   
## $b  
## [1] 2  
##   
## $c  
## [1] 3

And matrices.

m <- matrix(1:4, nrow = 2, ncol = 2)  
dimnames(m) <- list(c("a", "b"), c("c", "d"))  
m

## c d  
## a 1 3  
## b 2 4