

R Objects and Attributes

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 ## interger
```

```
x <- c(1+0i, 2+4i) ## complex
```

Using the `vector()` function

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

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) ## characte
```

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

```
> x <- 0:6 ## create a sequence of 0-6 >
class(x) ## class of the object [1] "integer"
```

Vectors (cont)

```
> as.numeric(x) ## change the class of the
object to numeric [1] 0 1 2 3 4 5 6
```

```
> as.logical(x) ## change the class of the
object to logical [1] FALSE TRUE TRUE
TRUE TRUE TRUE TRUE
```

```
> as.character(x) ## change the class of the
object to character [1] "0" "1" "2" "3" "4" "5"
"6"
```

```
> as.complex(x) [1] 0+0i 1+0i 2+0i 3+0i 4+0i
5+0i 6+0i
```

Nonsensical coercion results in NA's

```
> x<- c("a","b","c") > as.numeric(x) [1] NA
NA NA Warning message: NAs introduced
by coercion
```

```
> as.logical(x) [1] NA NA NA
```

```
> as.complex(x) [1] NA NA NA
```

Reading Tabular Data

`read.table`, `read.csv` * Tabular data
`readLines` * Reading lines of a text file
`source` * Reading in R code files (inverser of `dump`)
`dget` * Reading in R code files (inverse of `dput`)
`load` * Reading in saved workspaces
`unserialize` * Reading in single R objects in binary form

Analogous functions for WRITING data to files

Reading Tabular Data (cont)

`write.table()`, `writeLines()`, `dump()`, `dput()`, `save()`, `serialize()`

The `read.table` function is one of the most commonly used functions for reading data. Below are a few arguments

`file` * the name of a file or a connection
`header` * logical indicating if the file has a header line
`sep` * a string indicating how the columns are separated
`colClasses` * a character vector indicating the class of each column in the dataset
`nrows` * the number of rows in the dataset
`comment.char` * a character string indicating the comment character
`skip` * the number of lines to skip from the beginning
`stringsAsFactors` * should character variables be coded as factors? defaults to true

For small to moderately sized datasets you can call `read.table` without specifying any other arguments.

```
data <- read.table("sometable.txt")
```

* R will automatically
 * skips lines that being with a `"#"`
 * figure out how many rows there are (and how much memory needs to be allocated)
 * figure what type of variable is in each column of hte table telling R all these things
 * directly makes R run faster and more efficiently.
 * `read.csv` is idental to `read.table` EXCEPT that the default separator is a comma (,)



By **bwaldo**
cheatography.com/bwaldo/

Published 27th February, 2017.
 Last updated 27th February, 2017.
 Page 1 of 3.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>

Lists

Lists - Lists are a special type of vector that can contain elements of different classes.

Lists are a very IMPORTANT data type in R.

```
> x <- list(1,"a",TRUE, 1 +4i) ##list function
```

```
> x [[1]] ##vector element 1 [1] 1 [[2]]
##vector element 2 [1] "a" [[3]] ##vector
element 3 [1] TRUE [[4]] ##vector element 4
[1] 1+4i
```

Matrices

Matrices are vectors with a dimension attribute. The dimension attribute is itself an integer vector of length 2(nrow,ncol)

```
> m <- matrix(nrow = 2, ncol = 3) #matrix
function
```

```
> 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 columns

```
> m <- matrix(1:6, nrow = 2, ncol = 3) #
create a matrix with a sequence of 1-6
```

```
> m [,1] [,2] [,3] [1,] 1 3 5 [2,] 2 4 6
```

Matrices (cont)

Matrices can also be created directly from vectors by adding a dimension attribute

```
> m <- 1:10 # create a vector that is a
sequence from 1-10 > m [1] 1 2 3 4 5 6 7 8
9 10
```

Adding the dim function but assigning and attribute to m, take this vector and transform it into a matrix that is 2 rows and 5 columns

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

```
> m [,1] [,2] [,3] [,4] [,5] [1,] 1 3 5 7 9 [2,] 2 4
6 8 10
```

Matrices can be created by column-binding cbind() or row-binding 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
```

Reading Large Tables

A quick and dirty way to figure out the classes of each column is the following:

```
* initial <- read.table("datatable.txt", nrow = 100)
* classes <- sapply(initial, class)
* tabALL <- read.table("datatable.txt",
colClasses = classes)
* Set nrow. This doesn't make R run faster but it helps with memory usage
```

Textual Data Formats

* dput(): writes R code to which can be used to reconstruct a R object.

* dget(): single R objects

* dump(): similar to dget but can be used on multiple R objects.

dput() example

```
> y <- data.frame(a=1,b="a")
> dput(y) # output to console
structure(list(a = 1, b = structure(1L, .Label
= "a", class = "factor"))
, .Names = c("a", "b"), row.names = c(NA, -
1L), class = "data.frame")
> dput(y, file = "y.R") # save to a file named
y.R
> new.y <- dget("y.R") # get the file
> new.y
a b
1 1 a
```

dump() example

```
> x <- "foo"
> y <- data.frame(a=1,b="a")
> dump(c("x","y"), file = "data.R") # pass
the objects x and y and create and store in
data.R
> rm(x,y) # remove them from R
> source("data.R") # calls them back into R
> y
a b
1 1 a
> x
[1] "foo"
```

Factors

Factors are used to represent categorical data and can be unordered or ordered. One can think of a factor as an integer vector where each integer has a label

```
> x <- factor(c("yes","yes", "no", "yes", "no"))
> x [1] yes yes no yes no
Levels: no yes
```

```
> table(x) ##if you table x then it will tell how many of each value there are
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 modelling because the first level is used as the baseline level

So in the example below we set the levels of c because default would put no first and we want yes first

```
> x <- factor(c("yes","yes", "no", "yes", "no"), levels = c("yes","no"))
> x [1] yes yes no yes no
Levels: yes no
```

Missing Values

Missing values are denoted by NA or NaN for undefined mathematical operations.

Missing Values (cont)

- * 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 interger NA, character NA etc
- * a NaN value is also NA but the converse is not true

```
> x <- c(1,2,NA,10,3)
> x [1] 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
```

Names Attribute

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"
```

List can also have names

```
> x <- list(a=1,b=2,c=3)
> x
$a
[1] 1
$b
[1] 2
$c
[1] 3
```

Matrices can have names

Names Attribute (cont)

```
> m <- matrix(1:4, nrow=2, ncol=2) ##
create a matrix sequence 1-4
> dimnames(m) <- list(c("a","b"), c("c","d"))
> m
c d
a 1 3
b 2 4
```

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 # 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 attribute 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
```