Types of objects

AUTHOR
Karl Gregory

As we have demonstrated, you can use the R console like a calculator. However, the capabilities of R extend far beyond arithmetic! The first topic we will cover in exploring these capabilities is the various types of "objects" we can create in R.

Vectors

A vector in R is a list either of numbers, character strings, or logical values. We can create a vector using the command `c()`:

Numeric vectors

A vector with the numbers 2 and 1 can be created by the command

```
x <- c(2,1)
```

We can see what is in the vector `x` by typing `x` or `print(x)` into the console:

```
x
```

```
[1] 2 1
```

```
print(x)
```

```
[1] 2 1
```

Note that we can define objects using `=` instead of `<-` if we wish (the `<-` is referred to as the *assignment operator*, and this is peculiar to R. Most languages just use `=`. If you don't like using `<-` just use `=`.):

```
x = c(1,2)
x
```

```
[1] 1 2
```

A numeric vector with all the integers in a sequence can be created by

```
a <- -1
b <- 10
int <- a:b
int
```

```
 [1] -1  0  1  2  3  4  5  6  7  8  9 10
```

A numeric vector with equally spaced values between any numbers can be created as

```
a <- 0
b <- 1
vals <- seq(a,b,length=11)
vals
```

```
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

or

```
a <- 0
b <- 1
vals <- seq(a,b,by=0.05)
vals
```

```
 [1] 0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70
[16] 0.75 0.80 0.85 0.90 0.95 1.00
```

Character vectors

A character vector with the strings “cat”, “dog”, and “fish” is created as

```
pets <- c("dog","cat","fish")
pets
```

```
[1] "dog"  "cat"  "fish"
```

If we mix numbers and character strings we end up with a character vector:

```
v <- c(1,"fish")
v
```

```
[1] "1"    "fish"
```

The `paste()` function is useful for putting together character strings:

```
paste("fish",1:3)
```

```
[1] "fish 1" "fish 2" "fish 3"
```

```
paste(seq(0,10,by=2),"\u00B0","C",sep="")
```

```
[1] "0°C"  "2°C"  "4°C"  "6°C"  "8°C"  "10°C"
```

```
n <- 10
paste("n = ", n, sep = "")
```

```
[1] "n = 10"
```

```
paste(1:10, collapse = "-") # collapse into a single string
```

```
[1] "1-2-3-4-5-6-7-8-9-10"
```

Logical vectors

Create a logical vector with the values `FALSE`, `TRUE`, `F`, or `T`. These are called logical values or *boolean* values.

```
tf <- c(TRUE,FALSE,F,T)
tf
```

```
[1]  TRUE FALSE FALSE  TRUE
```

If we include a numeric value, the logicals are "coerced" to numeric values so that the whole vector will becomes a numeric vector; `TRUE` is coerced to 1 and `FALSE` to 0:

```
tf2 <- c(TRUE,FALSE,F,pi)
tf2
```

```
[1] 1.000000 0.000000 0.000000 3.141593
```

If we include a character string, the vector becomes a character string, with the logical entries converted to the strings "TRUE" and "FALSE".

```
tf3 <- c(TRUE,FALSE,F,"pi")
tf3
```

```
[1] "TRUE"  "FALSE" "FALSE" "pi"
```

Accessing vector elements

To access an element of a vector, we use square brackets:

```
x <- seq(0.01,0.99,by = 0.01)
x[2]
```

```
[1] 0.02
```

We can select more than one element like this:

```
pets <- c("dog","cat","fish")
pets[2:3]
```

```
[1] "cat"  "fish"
```

We can replace an entry with a new value, for example, with:

```
pets[3] <- "snake"
pets
```

```
[1] "dog"   "cat"   "snake"
```

We can also exclude an element or a set of elements from a vector by putting a negative subscript or a vector of negative subscripts in the brackets:

```
pets[-1]
```

```
[1] "cat"   "snake"
```

```
pets[-c(1,3)]
```

```
[1] "cat"
```

Another useful function for defining vectors is the `rep()` function for repeating a numeric, character, or logical value (or vector) a number of times:

```
z <- rep(pi,3)
z
```

```
[1] 3.141593 3.141593 3.141593
```

```
rep(pets,2)
```

```
[1] "dog"   "cat"   "snake" "dog"   "cat"   "snake"
```

Matrices

A matrix is a table of numbers, character strings, or logical values with a number of rows and columns.

One way to construct a matrix is by populating the rows and columns with the entries of a vector using the `matrix()` function:

```
A <- matrix(1:8,nrow = 2)
A
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
```

If we want to fill the values across the rows we add the option `byrow = T`:

```
A <- matrix(1:8,nrow = 2,byrow = T)
A
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
```

We can also use the `cbind()` or `rbind()` functions to build matrices from vectors given as the columns or the rows:

```
x <- c(1,2,3)
y <- -x
X <- cbind(x,y)
X
```

```
     x  y
[1,] 1 -1
[2,] 2 -2
[3,] 3 -3
```

```
Xt <- rbind(x,y)
Xt
```

```
  [,1] [,2] [,3]
x    1    2    3
y   -1   -2   -3
```

Note that the `cbind()` and `rbind()` operations result in a matrix with columns or rows having names given by the vectors. We can access the names of the columns or rows of a matrix with the `colnames()` and `rownames()` functions, respectively:

```
colnames(X)
```

```
[1] "x" "y"
```

```
rownames(Xt)
```

```
[1] "x" "y"
```

If the rows or columns of a matrix are unnamed, these functions will return the `NULL` value:

```
rownames(X)
```

```
NULL
```

```
colnames(Xt)
```

```
NULL
```

We can have matrices of character values:

```
B <- matrix(letters[1:12],nrow = 3, byrow = T)
B
```

```
     [,1] [,2] [,3] [,4]
[1,] "a"  "b"  "c"  "d"
[2,] "e"  "f"  "g"  "h"
[3,] "i"  "j"  "k"  "l"
```

The above used the built-in vector `letters`.

We can access elements or sub-matrice with square brackets:

```
B[2,3]
```

```
[1] "g"
```

```
B[1:2,3:4]
```

```
     [,1] [,2]
[1,] "c"  "d"
[2,] "g"  "h"
```

Matrices once defined as character matrices remain charactor matrices:

```
B[1,2] <- 1
B
```

```
     [,1] [,2] [,3] [,4]
[1,] "a"  "1"  "c"  "d"
[2,] "e"  "f"  "g"  "h"
[3,] "i"  "j"  "k"  "l"
```

We can query the dimension of a matrix with the `dim()` function:

```
dim(B)
```

```
[1] 3 4
```

We can get the transpose a matrix (rows become columns and vice versa) with `t()`:

```
t(X)
```

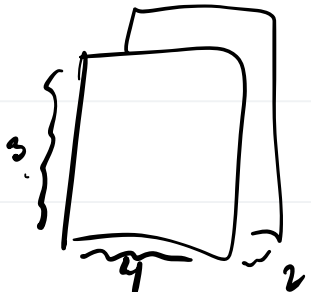
```
  [,1] [,2] [,3]
x    1    2    3
y   -1   -2   -3
```

Later on we will cover how to perform matrix operations such as matrix multiplication and inversion in R.

Arrays

An array consists of several matrices of the same dimension “stacked” as it were as “slices”, on top of each other. These behave just like matrices except that they have row, column, and “slice” indices.

Define an array with the `array()` function:



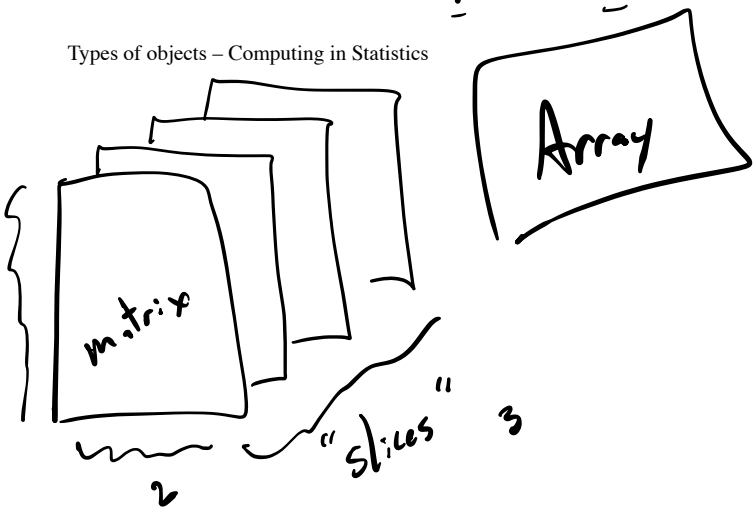
```
M <- array(c(1:24),dim = c(3,4,2))
M
```

```
, , 1

     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

, , 2

     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24
```



We access elements with row, column, and “slice” indices.

```
M[1,3,2]
```

```
[1] 19
```

```
M[1:2,1:2,1]
```

```
     [,1] [,2]
[1,]    1    4
[2,]    2    5
```

Data frames

A data frame is like a matrix except that its columns can be vectors of different types. The data frame is a like the R version of a “spreadsheet”, and its columns are typically given names.

We can construct a data frame with the `data.frame()` function.

```
df <- data.frame(pets = pets, population = c(30,12,34), mammal = c(T,T,F))
df
```

```
   pets population mammal
1   dog         30   TRUE
2   cat         12   TRUE
3 snake         34  FALSE
```

We can access a column of a data frame using its name like this:

```
df$pets
```

```
[1] "dog"   "cat"   "snake"
```

We can get a summary of the columns in a data frame with the `summary()` function:

```
summary(df)
```

```
     pets             population      mammal
 Length:3           Min.   :12.00   Mode :logical
 Class :character   1st Qu.:21.00   FALSE:1
 Mode  :character   Median :30.00   TRUE :2
                    Mean   :25.33
                    3rd Qu.:32.00
                    Max.   :34.00
```

Lists

A list is a collection of objects of any type. If you have several matrices or vectors of different sizes, a list can be a convenient place to store and access them.

We can construct a list with the `list()` function:

```
mylist <- list(pets = pets,int = int)
mylist
```

```
$pets
[1] "dog"   "cat"   "snake"

$int
 [1] -1  0  1  2  3  4  5  6  7  8  9 10
```

We can access the elements in a list using the name or with a double-bracketed index `[[]]`.

```
mylist$pets
```

```
[1] "dog"   "cat"   "snake"
```

```
mylist[[1]]
```

```
[1] "dog"   "cat"   "snake"
```

```
mylist[[1]][3]
```

```
[1] "snake"
```

Functions

A function is a set of commands which execute on some user-supplied arguments, or input values. We have seen some functions above such as the `c()`, `print()`, `seq()`, `paste()`, and so on. Later on we will discuss how to define new functions.

To access the help documentation for a function, just execute `?<nameoffunction>`. For example:

```
?paste
```

or help(paste)

You will do this very often, even after you become proficient in R.

Checking the type of an object

For any of these object classes, we can check whether an object belongs to the class with the functions `is.vector()`, `is.matrix()`, etc., which return a logical value:

```
is.vector(pets)
```

```
[1] TRUE
```

```
is.character(pets)
```

```
[1] TRUE
```

```
is.matrix(df)
```

```
[1] FALSE
```

```
is.data.frame(X)
```

```
[1] FALSE
```

```
is.numeric(X)
```

```
[1] TRUE
```

```
is.data.frame(df)
```

```
[1] TRUE
```

```
is.function(paste)
```

```
[1] TRUE
```

Coercion of one type of object to another type

It is also possible to "coerce" one type of object to another type. For example, we can choose to read numeric vectors as character vectors or to read a matrix as a vector with the functions `as.character()`, `as.vector()`. There are several such functions beginning with `as.` for coercing one type of object to another type.

Here we coerce a numeric vector to a character vector:

```
as.character(int)
```

```
 [1] "-1" "0"  "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"
```

If the coercion does not make sense, we will get missing values, which R represents as `NA`. A warning may be issued. For example, this happens when we try to convert certain character vectors to a numeric vector:

```
as.numeric(pets)
```

```
Warning: NAs introduced by coercion
```

```
[1] NA NA NA
```

If a character string *can* be converted to a number, the coercion will work:

```
ch <- c("1.234","1e04")
as.numeric(ch)
```

```
[1]     1.234 10000.000
```

When logical vectors are coerced to numeric vectors, the values `TRUE` and `FALSE` are converted to 1 and 0, respectively:

```
tf
```

```
[1]  TRUE FALSE FALSE  TRUE
```

```
as.numeric(tf)
```

```
[1] 1 0 0 1
```

If we convert a numeric vector to a logical vector, nonzero values are converted to `TRUE` and zeroes are converted to `FALSE`:

```
int
```

```
 [1] -1  0  1  2  3  4  5  6  7  8  9 10
```

```
as.logical(int)
```

```
 [1]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

If we convert a matrix to a vector the operation “vectorizes” the matrix, which means that it returns a vector containing all the columns of the matrix concatenated into one long vector.

```
X
```

```
     x  y
[1,] 1 -1
[2,] 2 -2
[3,] 3 -3
```

```
as.vector(X)
```