Video 4: Vectors in R

Stats 102A

Miles Chen, PhD

Vectors

Related Reading

Advanced R, 2nd edition, Chapter: Vectors

https://adv-r.hadley.nz/vectors-chap.html

Vectors

The most important family of objects in R is **vectors**.

There are two types of vectors: atomic vectors and generic vectors (also called lists).

In atomic vectors all elements must be of the same type.

In lists, elements can have different types.

While not a vector, `NULL` is related to vectors and often serves the role of a generic vector of length zero. There are also vectors of specific types with length zero. `NULL` will be covered in more detail in the section on special values.

Atomic Vectors

The most fundamental object in R is an **atomic vector** (or vector), which is an ordered collection of values.

Atomic vectors have six basic types (though we will only work with the first four):

`typeof()`	`mode()`
logical	logical
double	numeric
integer	numeric
character	character
complex	complex
raw	raw

All elements of an atomic vector must be of the same type.

Numeric types: Double and Integer

The conversion between integer and double values is often done automatically in R, so the distinction is typically not needed (they are both numeric types).

There is an important difference: doubles are numeric values stored with floating point precision, while integers are stored as exact integer values.

The default numeric type is double. Integers can be indicated with an `L` after the number and vectors of integers can be created using the colon (`:`) operator.

Doubles and Integers

```
1 typeof(c(1, 2, 3))
```

```
[1] "double"
```

```
1 typeof(1:3)
```

```
[1] "integer"
```

```
1 is.double(1)
```

```
[1] TRUE
```

```
1 is.integer(1L)
```

```
[1] TRUE
```

Coercion

To illustrate the idea of coercion, we create the following vectors.

```
l <- c(TRUE, FALSE)
i <- 1L
d <- c(5, 6, 7)
ch <- c("a", "b")
```

Atomic vectors in R can only contain one data type. When values of different types are combined into a single vector, the values are **coerced** into a single type.

Question: What is the output for the following commands?

```
typeof(c(l, i, d))
typeof(c(l, d, ch))
```

Coercion

Coercion looks at the least restrictive type and coerces everything to that type.

The order from most restrictive to least restrictive is logical < integer < double < character < list

```
c(l, i, d)
```

```
[1] 1 0 1 5 6 7
```

```
typeof(c(l, i, d))
```

```
[1] "double"
```

```
c(l, i, ch)
```

```
[1] "TRUE"  "FALSE" "1"     "a"     "b"
```

```
typeof(c(l, d, ch))
```

```
[1] "character"
```

Implicit Coercion

Coercion often happens automatically. Most mathematical functions (`+`, `log()`, `abs()`, etc.) will coerce to a double or integer, and most logical operations (`&`, `|`, `any()`, etc.) will coerce to a logical.

```
1 trials <- c(FALSE, FALSE, TRUE)
2 as.numeric(trials)
```

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

```
1 sum(trials)  # Total number of TRUEs
```

```
[1] 1
```

```
1 mean(trials) # Proportion that are TRUE
```

```
[1] 0.3333333
```

Explicit Coercion

The `as` functions can be used to explicitly coerce objects, if possible.

```
1 as.character(trials)
```

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

```
1 as.logical(c(0, 1))
```

```
[1] FALSE  TRUE
```

```
1 as.numeric("dog")
```

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

```
[1] NA
```

Explicit Coercion

Explicit coercion rules

```
1 # anything numeric that is not 0 becomes TRUE except NaN becomes NA
2 as.logical(c(0, 1, -1, 0.1, 2, -Inf, 2.2e-308, NaN) )
```

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

```
1 # accepted spellings of logical values
2 as.logical(c("F", "FALSE", "False", "false", "T", "TRUE", "True", "
```

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

```
1 as.logical(c("f","t", "cat", 0, 1)) # other characters not coerced
```

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

Lists

A **list** (or generic vector) is an ordered collection of objects. Lists are the most flexible objects in R, as each component in a list can be *any* other object, including other lists.

The most common objects built from vectors are summarized in the table below.

Dimension	Homogeneous	Heterogeneous
1-dim	Atomic vector	List (generic vector)
2-dim	Matrix	Data frame
n-dim	Array	

Lists

```r
l1 <- list(
  1:3,
  "a",
  c(TRUE, FALSE, TRUE),
  c(2.3, 5.9)
)

typeof(l1)
```

```
[1] "list"
```

```r
str(l1)
```

```
List of 4
 $ : int [1:3] 1 2 3
 $ : chr "a"
 $ : logi [1:3] TRUE FALSE TRUE
 $ : num [1:2] 2.3 5.9
```

Lists

`c()` will combine several lists into one. If given a combination of atomic vectors and lists, `c()` will coerce the vectors to lists before combining them. Compare the results of `list()` and `c()`:

```
1 l4 <- list(list(1, 2), c(3, 4))
2 l5 <- c(list(1, 2), c(3, 4))
3 str(l4)
```

```
List of 2
 $ :List of 2
  ..$ : num 1
  ..$ : num 2
 $ : num [1:2] 3 4
```

```
1 str(l5)
```

```
List of 4
 $ : num 1
 $ : num 2
 $ : num 3
 $ : num 4
```

Attributes

Every vector (atomic or list) can also have **attributes**, which is a named *list* of arbitrary metadata.

Two attributes are particularly important: The **dimension** attribute turns vectors into matrices and arrays, and the **class** attribute develops the **S3** object system.

Attributes

The **`attr()`** function can be used to get or set single attributes of an object.

The **`attributes()`** function can be used to access the entire list of attributes.

Let's take a look at the data frame `trees`.

```
1 head(trees)
```

```
  Girth Height Volume
1   8.3     70   10.3
2   8.6     65   10.3
3   8.8     63   10.2
4  10.5     72   16.4
5  10.7     81   18.8
6  10.8     83   19.7
```

Attributes

The attributes of the trees data frame is a list with three elements: names (the column names), class (data.frame), and row names

```
1 attributes(trees)
```

```
$names
[1] "Girth"  "Height" "Volume"

$class
[1] "data.frame"

$row.names
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
25
[26] 26 27 28 29 30 31
```

Attributes

We can technically add any arbitrary bit of information and throw it into the attributes list and then retrieve that information later. This is not a normal practice, but because attributes are simply a list, it's possible.

```r
attr(trees, "info") <- "This data frame is about trees!!"
attributes(trees)
```

```
$names
[1] "Girth"  "Height" "Volume"

$class
[1] "data.frame"

$row.names
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
25
[26] 26 27 28 29 30 31

$info
[1] "This data frame is about trees!!"
```

Names

You can name a vector

```r
# When creating it:
x <- c(a = 1, b = 2, c = 3)

# By assigning a character vector to names()
x <- 1:3
names(x) <- c("a", "b", "c")

# with setNames():
x <- setNames(1:3, c("a", "b", "c"))
```

Removing Names

```r
# with unname
x <- unname(x)

# setting names to NULL
names(x) <- NULL
```