Video 6: NA, NULL, also Recycling

Stats 102A

Miles Chen, PhD

Special Values

Question: What is the difference between `NA`, `NULL`, and `NaN`?

Special Values

Question: What is the difference between `NA`, `NULL`, and `NaN`?

- **`NA`** is used to represent missing or unknown values. There is an `NA` for each type.
- **`NULL`** is used to represent an empty or nonexistent value. NULL is its own type.
- **`NaN`** is type double and is used to represent indeterminate forms in mathematics (such as `0/0` or `-Inf + Inf`).

Example: You are storing information about people. You have columns for their name, their age, their partner's name, and their partner's age.

Joe is a person. You don't know Joe's age. You enter NA for Joe's age. (The age exists, but you don't know it.) Joe does not have a partner. You would enter NULL for the partner's name and partner's age. (These values do not exist.) If Joe had a partner but you did not know the partner's age, you would enter NA instead of NULL.

NA

Including NA in an atomic vector of matrix will not change the data type. Internally, R has an NA for each data type.

```
NA
NA_integer_
NA_real_
NA_character_
```

NA

To check for NA, you must use the function `is.na()`. You cannot use `==`

```
1  NA == NA
```

```
[1] NA
```

```
1  is.na(NA)
```

```
[1] TRUE
```

NULL

R uses NULL to represent the NULL object. It is its own type.

```
1 typeof(NULL)
```

```
[1] "NULL"
```

```
1 is.null(NULL)
```

```
[1] TRUE
```

```
1 is.na(NULL)
```

```
logical(0)
```

NULL

```
1 is.logical(NULL)
```

```
[1] FALSE
```

```
1 NULL + FALSE  # operations with NULL result in a length 0 vector
```

```
integer(0)
```

```
1 c(4, 5, NULL, 3) # "including" NULL is like including nothing
```

```
[1] 4 5 3
```

```
1 NULL == NULL
```

```
logical(0)
```

0-length atomic vectors

There are also zero-length atomic vectors for each type.

```
1 l <- logical(0)
2 length(l)
```

```
[1] 0
```

```
1 typeof(l)
```

```
[1] "logical"
```

```
1 is.na(l)
```

```
logical(0)
```

```
1 is.null(l)
```

```
[1] FALSE
```

```
1 c(l, TRUE)
```

```
[1] TRUE
```

0-length atomic vectors

```
d <- numeric(0)
length(d)
```

```
[1] 0
```

```
typeof(d)
```

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

```
is.na(d)
```

```
logical(0)
```

```
is.null(d)
```

```
[1] FALSE
```

```
c(d, TRUE)
```

```
[1] 1
```

0-length atomic vectors

You often end up with a zero-length vector if you try to subset a vector and none of the elements meet the criteria.

```
1  i <- 1:3
2  i == 4
```

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

```
1  i[i == 4]
```

```
integer(0)
```

0-length lists

You can also have an empty list, which is different from a list containing NULL.

```
1 li <- list() # 0-length list
2 length(li)
```

```
[1] 0
```

```
1 li_null <- list(NULL) # list containing null
2 li_null
```

```
[[1]]
NULL
```

```
1 length(li_null)
```

```
[1] 1
```

```
1 li_double <- list(numeric(0))
2 li_double
```

```
[[1]]
numeric(0)
```

```
1 length(li_double)
```

```
[1] 1
```

Vector Arithmetic

Arithmetic can be done on numeric vectors using the usual arithmetic operations. The operations are **vectorized**, i.e., they are applied elementwise (to each individual element).

```
1 x <- c(1, 2, 3)
2 y <- c(100, 200, 300)
3 x + y
```

```
[1] 101 202 303
```

```
1 x * y
```

```
[1] 100 400 900
```

Vector Recycling

When applying arithmetic operations to two vectors of different lengths, R will automatically **recycle**, or repeat, the shorter vector until it is long enough to match the longer vector.

Question: What is the output of the following commands?

```
c(1, 2, 3) + c(100, 200, 300, 400, 500, 600)
c(1, 2, 3) + c(100, 200, 300, 400, 500)
```

Question: When will R throw a warning when recycling?

Vector Recycling

```
1 c(1, 2, 3) + c(100, 200, 300, 400, 500, 600)
```

```
[1] 101 202 303 401 502 603
```

```
1 c(1, 2, 3) + c(100, 200, 300, 400, 500)
```

```
Warning in c(1, 2, 3) + c(100, 200, 300, 400, 500): longer object length is
not
a multiple of shorter object length
```

```
[1] 101 202 303 401 502
```

Vector Recycling - Matrices

```r
M <- rbind(c( 1,  2, 3),
           c( 4,  5, 6),
           c( 7,  8, 9),
           c(10, 11, 12))
print(M)
```

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

```r
x <- c(100, 200, 300)
M + x # recycling is done column-wise
```

```
     [,1] [,2] [,3]
[1,]  101  202  303
[2,]  204  305  106
[3,]  307  108  209
[4,]  110  211  312
```

Vector Recycling - Matrices

```
1 t(M) # t() transposes the matrix
```

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

```
1 t(M) + x  # recycling is still done column-wise
```

```
     [,1] [,2] [,3] [,4]
[1,]  101  104  107  110
[2,]  202  205  208  211
[3,]  303  306  309  312
```

```
1 t(t(M) + x) # transposing the result is equivalent to recycling row
```

```
     [,1] [,2] [,3]
[1,]  101  202  303
[2,]  104  205  306
[3,]  107  208  309
[4,]  110  211  312
```