Lecture 2: Vectors

STAT 450, Fall 2021

Numeric Vectors

A `numeric vector` in R is just a collection of numbers. To create a vector use `c()`, which is short for combine.

```
x <- c(1, 2, 3, 47)
y <- c(1, -1, 2, -2)
```

We can add and multiply vectors. We can also multiply or divide a vector by a scalar. Operations are performed element-wise.

```
x + y
```

```
## [1]  2  1  5 45
```

```
x * y
```

```
## [1]   1  -2   6 -94
```

```
2*x # multiplies each element by 2
```

```
## [1]  2  4  6 94
```

```
x/2 # divides each element by 2
```

```
## [1]  0.5  1.0  1.5 23.5
```

```
x^2 # squares each element of the vector
```

```
## [1]    1    4    9 2209
```

Sequences

Some ways to create vectors containing sequences of numbers:

```
x <-  1:40
x
```

```
##  [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 32 33 34 35 36 37 38 39 40
```

```
y <- seq(2, 20, by=2)
y
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

```
z <- seq(0, 1, by=0.1)
z
```

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

Use `rep()` to repeat elements in a vector:

```
rep(1, 10) # repeats 1 ten times
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1
```

```
x <- c(rep(2,3), rep(3,4))
x
```

```
## [1] 2 2 2 3 3 3 3
```

Exercise: Use `seq()` to create a numeric vector containing the sequence 0, 10, 20, . . . , 100

Vector Functions

R provides some easy-to-use functions for computing numerical summaries of vectors. Below are some examples using a vector containing the ages of 10 people.

```
age <- c(54, 40, 53, 26, 56, 52, 28, 39, 15, 17)
length(age) # length of vector
```

```
## [1] 10
```

```
sort(age)
```

```
##  [1] 15 17 26 28 39 40 52 53 54 56
```

```
sort(age, decreasing = TRUE)
```

```
##  [1] 56 54 53 52 40 39 28 26 17 15
```

```
min(age)
```

```
## [1] 15
```

```
max(age)
```

```
## [1] 56
```

```
mean(age)
```

```
## [1] 38
```

```
median(age)
```

```
## [1] 39.5
```

```
sd(age) # standard deviation
```

```
## [1] 15.70563
```

Or more conveniently, use `summary()` to compute several summary statistics at once.

```
summary(age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   15.00   26.50   39.50   38.00   52.75   56.00
```

Note that `1st Qu.` in the first quartile (25th percentile) and `3rd Qu.` is the third quartile (75th percentile).

Documentation on these functions is provided in the help menu. To access the help menu from the console use the `help()` function. For example, enter the following command to read about the `sort` function in the help menu:

```
help(sort)
```

Data Types

There are four common data types for vectors: numeric (also called double), integer, character, and logical. Some examples:

```
x <- c(1, 0.5) # numeric
x <- 1:10 # integer
x <- c("a", "b", "c") # character
x <- c(TRUE, FALSE) # logical
```

You can use `class()` to check the data type.

```
y <- c(TRUE, TRUE, FALSE, FALSE)
class(y)
```

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

```
letters
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
```

```
class(letters)
```

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

Coercion

Vectors in R can only contain data of the same type. When different data types are mixed in a vector, R will implicitly coerce all elements to the same type.

```
x <- c("a", 2)
x
```

```
## [1] "a" "2"
```

```
class(x)
```

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

Exercise: Consider the R code below with vectors that contain different data types. Since this is not allowed, what data type does R coerce each vector to?

```
x <- c(1, 0, FALSE, TRUE)
y <- c(1, 2, "three")
z <- c("TRUE", FALSE)
```

Subsetting a vector

To subset specific elements of a vector use brackets `[]`. The number in the bracket is the index, that is, the position of the element within the vector.

```
age <- c(54, 40, 53, 26, 56, 52, 28, 39, 15, 17)
age[1]
```

```
## [1] 54
```

```
age[5]
```

```
## [1] 56
```

```
age[1:5]
```

```
## [1] 54 40 53 26 56
```

```
age[c(1, 3)]
```

```
## [1] 54 53
```

```
age[-1] # drop first element
```

```
## [1] 40 53 26 56 52 28 39 15 17
```

```
age[-c(1, 3)] # drop first and third element
```

```
## [1] 40 26 56 52 28 39 15 17
```

Vectors can also be subsetted using logical values.

```
age > 50
```

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

```
age[age > 50] # subset ages greater than 50
```

```
## [1] 54 53 56 52
```

```
sum(age > 50) # counts the number of ages greater than 50
```

```
## [1] 4
```

Exercise: Test you knowledge of vector subsetting by predicting the output of the following code:

```
weight <- c(140, 139, 187, 181, 131)
weight[2:3]
weight[weight > 150]
sum(weight > 150)
```

Missing Data

Missing values are denoted as `NA` (not available). The function `is.na()` can be used to check for missing data; it returns a logical vector that is `TRUE` if an element is `NA`, and `FALSE` otherwise.

```
x <- c(7, 11, NA, NA, 12)
is.na(x)
```

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

A common task in data analysis is to remove missing values:

```
!is.na(x)
```

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

```
x[!is.na(x)]
```

```
## [1]  7 11 12
```

The `!` symbol is the `not` operator. It negates the elements of a logical vector.