Introduction to R, Data types and Data Structure in R

Data in R, Fall 2022

:"Siddharth Chaudhary"

Getting help

```
help(sqrt)
?sqrt
help.start()
```

```
## starting httpd help server ... done

## If the browser launched by '/usr/bin/open' is already running, it is
##     *not* restarted, and you must switch to its window.
## Otherwise, be patient ...
```

---

basic operations

```
2+2
```

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

```
2-3*4
```

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

```
(2-3)*4
```

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

```r
5/9*4
```

```
## [1] 2.222222
```

```r
3^2   # 3 to the power of 2
```

```
## [1] 9
```

```r
5 < 4
```

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

```r
3 == 3   # this is a test, are these values equal?
```

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

```r
6/3 != 2  # this is a test, are these values NOT equal
```

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

```r
8/2 <= 4  # this is a test, is the left side value less than or equal to the right side?
```

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

Basic functions

```r
sqrt(4)   #square root of four
```

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

```r
sqrt(9)   #notice that comments after the pound sign are ignored
```

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

```r
# notice that case matters in R
sqrt(9)
```

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

```r
print("Help! I've been abducted by aliens!!!")
```

```
## [1] "Help! I've been abducted by aliens!!!"
```

```r
cat("Help!\n\n", "I've been abducted by smart aliens!!!", sep="")
```

```
## Help!
##
## I've been abducted by smart aliens!!!
```

Notice that the output dumps onto the console. Whatever is dumped to the console is not retained.

We can change that by assigning the results to a object.

```r
my_results <- sqrt(9)
my_results
```

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

```r
#My_results   # case matters

my_results = sqrt(16)
my_results
```

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

```r
# Better practice is to use assignment operator
```

---

command separation

```r
print("blah")
```

```
## [1] "blah"
```

```r
print("ughh")
```

```
## [1] "ughh"
```

```r
print("really?")
```

```
## [1] "really?"
```

```
print("blah"); print("ughh"); print("really?")  # semi colons for separation
```

```
## [1] "blah"
```

```
## [1] "ughh"
```

```
## [1] "really?"
```

```
# we can also string a command across multiple lines, though this is not good practice

# notice that R adds a + sign to the new line to indicate continuation of the command
print("blah"
)
```

```
## [1] "blah"
```

In this lecture, you will learn about the most basic data structure in R. # ———— # Part 1 - Basic data types # ————

There are four common types of data in R: double (numeric), integer, logical and character. (We won't work with raw or complex values.)

Numeric

---

double or **numeric** vector

```
vect_dbl <- c(1.1,-2.5,4,6,2.2,12) # create vector data object
vect_dbl
```

```
## [1]  1.1 -2.5  4.0  6.0  2.2 12.0
```

```
typeof(vect_dbl)
```

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

We can also examine a data object using the **str** structure function.

```
str(vect_dbl)
```

```
##  num [1:6] 1.1 -2.5 4 6 2.2 12
```

Notice that the **str** function tells us this is numeric data!

---

integer

---

integer vector

```
vect_int <- -1:10 # create vector data object
vect_int
```

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

```
typeof(vect_int)
```

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

```
str(vect_int)
```

```
##  int [1:12] -1 0 1 2 3 4 5 6 7 8 ...
```

R treats integers the same as numeric values for most functions.

---

logical

---

logical vector

```
vect_log <- c(T, FALSE, TRUE, F,T,T) # create vector data object
vect_log
```

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

```
typeof(vect_log)
```

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

```
str(vect_log)
```

```
##  logi [1:6] TRUE FALSE TRUE FALSE TRUE TRUE
```

```
cat("\nsum of logical vector: ", sum(vect_log))
```

```
##
## sum of logical vector:  4
```

Note that R treats T and TRUE, F and FALSE as equivalent. When treated as numbers, T=1, F=0.

logical and numeric vector

```
vect_log <- c(T, FALSE, TRUE, F,T,T, 5.5) # create vector data object
vect_log
```

```
## [1] 1.0 0.0 1.0 0.0 1.0 1.0 5.5
```

```
typeof(vect_log)
```

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

```
str(vect_log)
```

```
##  num [1:7] 1 0 1 0 1 1 5.5
```

```
cat("\nsum of logical vector: ", sum(vect_log))
```

```
##
## sum of logical vector:  9.5
```

---

character

---

character vector

```
vect_chr <- c("Dave", "Vlad","Chuck","Larry","Bob","Mary","Doug","Daryll","Ron","Sue", "Laura", "Rich")
vect_chr
```

```
##  [1] "Dave"   "Vlad"   "Chuck"  "Larry"  "Bob"    "Mary"   "Doug"   "Daryll"
##  [9] "Ron"    "Sue"    "Laura"  "Rich"
```

```
typeof(vect_chr)
```

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

```
str(vect_chr)
```

```
##  chr [1:12] "Dave" "Vlad" "Chuck" "Larry" "Bob" "Mary" "Doug" "Daryll" ...
```

Note that each new line in the vector printout is labeled with the vector position. Character values are represented with quotes.

The structure function will list specific character values as space allows, then...

character and numeric vector

```
vect_chr <- c("Dave", "Vlad","Chuck","Larry","Bob","Mary","Doug","Daryll","Ron","Sue", "Laura", "Rich",
vect_chr
```

```
##  [1] "Dave"   "Vlad"   "Chuck"  "Larry"  "Bob"    "Mary"   "Doug"   "Daryll"
##  [9] "Ron"    "Sue"    "Laura"  "Rich"   "5.5"
```

```
typeof(vect_chr)
```

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

```
str(vect_chr)
```

```
##  chr [1:13] "Dave" "Vlad" "Chuck" "Larry" "Bob" "Mary" "Doug" "Daryll" ...
```

Atomic vector

An **atomic vector** or **vector** in R is a one-dimensional data structure. A **vector** holds a string of values, all of the same type, with the string having (practically) any length. You can create a vector using the combine or concatenate **c** function.

```
vect <- c(1,-2,4,6,2.2,12) # create vector data object
vect  # print data object to the console
```

```
## [1]  1.0 -2.0  4.0  6.0  2.2 12.0
```

Notice how the vector is printed to the console.

You can test whether an object is an atomic vector using two different commands.

```
is.atomic(vect)   # more robust test
```

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

```
is.vector(vect)   # won't always work, depending upon attributes
```

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

Vector properties

We can assign and extract vector properties.

We can obtain the vector length with the following command.

```
# length function gives length of a vector (number of values stored)
length(vect)
```

```
## [1] 6
```

```
length(c("123456789","0"))
```

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

There are six pieces of data, six positions in this vector.

We can name elements in a vector.

```
# Create a vector with my height and weight in SI units (m, kg)
vect_example <- c(1.75, 70)
vect_example
```

```
## [1]  1.75 70.00
```

```
cat("\n")   # this prints a line space
```

```
#now assign names for meaning, notice these are character strings
names(vect_example) <- c("height (m)", "weight (kg)")
vect_example
```

```
##  height (m) weight (kg)
##        1.75       70.00
```

```
cat("\n")   # this prints a line space
```

```
names(vect_example) <- c("height (m)", "weight (kg)")
vect_example
```

```
##  height (m) weight (kg)
##        1.75       70.00
```

We can also assign vector names when creating the vector.

```
# Create a vector with my height and weight in SI units (m, kg)
vect_example <- c("height (m)"=1.75, "weight(kg)" =70)
vect_example
```

```
## height (m) weight(kg)
##       1.75      70.00
```

We can inquire about vector type using **typeof()** function.

```
typeof(vect_example)
```

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

```
typeof(c("123456789","0"))
```

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

"double" means double precision numeric or commonly thought of as "numeric". We will discuss other types of data.

Finally we can inquire about data object attributes.

```
vect <- c(1,-2,4,6,2.2,12) # create vector data object
attributes(vect)
```

```
## NULL
```

```
cat("\n")
```

```
attributes(vect_example)
```

```
## $names
## [1] "height (m)" "weight(kg)"
```

The first vector has no attributes. The second has names. In object oriented programming, data object attributes can be considered "meta data".

Creating Vectors

You can also create vectors with other commands.

```
vect_rep <- rep(1.2,times=10) # replicate value 10 times
vect_rep
```

```
##  [1] 1.2 1.2 1.2 1.2 1.2 1.2 1.2 1.2 1.2 1.2
```

```
vect_seq <- seq(from=1, to=13, by=2) # sequence from 1 to 12 by 2
vect_seq
```

```
## [1]  1  3  5  7  9 11 13
```

```
vect_seq_by1 <- 1:6 # sequence integers by 1
vect_seq_by1
```

```
## [1] 1 2 3 4 5 6
```

```
vect_seq_by1_neg <- -1:6 # sequence integers by 1
vect_seq_by1_neg
```

```
## [1] -1  0  1  2  3  4  5  6
```

```
vect_seq_by1_pr <- -(1:6) # sequence integers by 1
vect_seq_by1_pr
```

```
## [1] -1 -2 -3 -4 -5 -6
```

Missing values (NA)

All spaces in a vector or matrix must be filled with something. If there is no data for a position, that position is represented by the missing value designation in R, **NA**. Note that a missing value is NOT the same as the character value "NA".

Replace Doug with NA and Sue with "NA".

```
vect_na <- c("Dave", "Vlad","Chuck","Larry","Bob","Mary",NA,"Daryll","Ron","NA", "Laura", "Rich")
vect_na
```

```
##  [1] "Dave"   "Vlad"   "Chuck"  "Larry"  "Bob"    "Mary"   NA       "Daryll"
##  [9] "Ron"    "NA"     "Laura"  "Rich"
```

```
str(vect_na)
```

```
##  chr [1:12] "Dave" "Vlad" "Chuck" "Larry" "Bob" "Mary" NA "Daryll" "Ron" ...
```

```
vect_rep_na <- rep(NA,85)
vect_rep_na
```

```
##  [1] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
## [26] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
## [51] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
## [76] NA NA NA NA NA NA NA NA NA NA
```

```
str(vect_rep_na)
```

```
##  logi [1:85] NA NA NA NA NA NA ...
```

---

coerce data types

We can *coerce* data into a particular type or from one type to another.

as.character

as.character examples

```
vect <- c(1.1,-2.5,4,6,2.2,12) # create vector data object
cat("numeric to character \n")  # cat prints nicely
```

```
## numeric to character
```

```
vect
```

```
## [1]  1.1 -2.5  4.0  6.0  2.2 12.0
```

```
as.character(vect)
```

```
## [1] "1.1"  "-2.5" "4"    "6"    "2.2"  "12"
```

```
# we put a \n in front for an extra return or line
vect <- -1:8 # create vector data object
cat("\ninteger to character\n")
```

```
##
## integer to character
```

```
vect
```

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

```
as.character(vect)
```

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

```
cat("\n")
```

```
vect <- c(T, FALSE, TRUE, F,T,T) # create vector data object
cat("\nlogical to character\n")
```

```
##
## logical to character
```

```
vect
```

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

```
as.character(vect)
```

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

as.numeric

as.numeric examples

```
cat("integer to numeric\n")
```

```
## integer to numeric
```

```
vect
```

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

```
as.numeric(vect)
```

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

```
str(as.numeric(vect))
```

```
##  num [1:6] 1 0 1 0 1 1
```

```
cat("\nlogical to numeric\n")
```

```
##
## logical to numeric
```

```
vect
```

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

```
as.numeric(vect)
```

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

```
vect <- c("Dave", "Vlad","Chuck","Larry","Bob","Mary","Doug","Daryll","Ron","Sue", "Laura", "Rich")
cat("\ncharacter to numeric\n")
```

```
##
## character to numeric
```

```
vect
```

```
##  [1] "Dave"   "Vlad"   "Chuck"  "Larry"  "Bob"    "Mary"   "Doug"   "Daryll"
##  [9] "Ron"    "Sue"    "Laura"  "Rich"
```

```
as.numeric(vect)
```

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

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

Oops! Some data can't be coerced!

as.integer

as.integer examples

```
cat("numeric to integer\n")
```

```
## numeric to integer
```

```
vect2 <- c(1.1,-2.6,4,6.6,2.2,12)
vect2
```

```
## [1]  1.1 -2.6  4.0  6.6  2.2 12.0
```

```
as.integer(vect2)
```

```
## [1]  1 -2  4  6  2 12
```

as.integer rounds off numeric values, otherwise has effect like **as.numeric**.

as.logical

as.logical examples

```
cat("numeric to logical\n")
```

```
## numeric to logical
```

```
vect1 <- c(1.1,-2.6,0,6.6,2.2,12)
vect1
```

```
## [1]  1.1 -2.6  0.0  6.6  2.2 12.0
```

```
as.logical(vect1)
```

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

```
cat("\ninteger to logical\n")
```

```
##
## integer to logical
```

```
vect2
```

```
## [1]  1.1 -2.6  4.0  6.6  2.2 12.0
```

```
as.logical(vect2)
```

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

```
cat("\ncharacter to logical\n")
```

```
##
## character to logical
```

```
vect4 <- c("Dave", "Vlad","Chuck","Larry","Bob","Mary","Doug","Daryll","Ron","Sue", "Laura", "TRUE")
vect4
```

```
##  [1] "Dave"   "Vlad"   "Chuck"  "Larry"  "Bob"    "Mary"   "Doug"   "Daryll"
##  [9] "Ron"    "Sue"    "Laura"  "TRUE"
```

```
as.logical(vect4)
```

```
##  [1]   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA TRUE
```

coercion of mixed data

Finally, data are also coerced when creating a vector, if necessary, since vectors can only contain one type of data.

```
vect5 <- c(T, F, 1, T)
str(vect5)
```

```
##  num [1:4] 1 0 1 1
```

```
cat("\n")
```

```
vect6 <- c(T, F, "bob", T)
str(vect6)
```

```
##  chr [1:4] "TRUE" "FALSE" "bob" "TRUE"
```

```
cat("\n")
```

```
vect7 <- c(1, 2.3, "bob",7)
str(vect7)
```

```
##  chr [1:4] "1" "2.3" "bob" "7"
```

Logical is coerced to numeric. Everything is coerced to character if there is a character element. As we have seen previously, when you combine numeric and integers, a numeric vector results.

Notes

- Where coersion fails, NA values result.
- MANY functions coerse data and objects into what is needed. This can be both useful and dangerous!

Matrices

A **matrix** in R is a 2-dimensional data structure. A **matrix** holds a n by m grid of values, with each of the two dimensions (practically) any length. As with atomic vectors, a matrix must contain values of just one type.

Creating a matrix

You can create a **matrix** using the **matrix** function.

```
# 'data' is a vector
# 'nrow' gives the number of rows
# 'ncol' gives the number of columns
# 'byrow' by default is FALSE, so data is read in column-wise
vect <- c(1,-2,4,6,2.2,12)
mat <- matrix(data = vect, nrow = 3, ncol = 2)
mat
```

```
##      [,1] [,2]
## [1,]    1  6.0
## [2,]   -2  2.2
## [3,]    4 12.0
```

Examine matrix properties

We can examine the properties of this matrix data object.

```
# dim function gives the dimensions of a matrix
cat("matrix row-column dimensions:", dim(mat),"\n")
```

```
## matrix row-column dimensions: 3 2
```

```
# nrow gives number of rows
cat("\nmatrix row dimension:", nrow(mat),"\n")
```

```
##
## matrix row dimension: 3
```

```
# ncol gives number of rows
cat("\nmatrix column dimension:", ncol(mat),"\n")
```

```
##
## matrix column dimension: 2
```

```
# typeof gives use the type of data
cat("\nmatrix data type:", typeof(mat),"\n")
```

```
##
## matrix data type: double
```

```
# str is also useful
cat("\nmatrix structure:\n")
```

```
##
## matrix structure:
```

```
str(mat)
```

```
##  num [1:3, 1:2] 1 -2 4 6 2.2 12
```

Note that you use *dim* for matrices, *length* to extract the dimensions of vectors.

name attributes for matrix

We can also name rows or columns by assignment.

```
#first look at attributes
attributes(mat)
```

```
## $dim
## [1] 3 2
```

```
cat("\n")    # blank line
```

```
#colnames and rownames functions are used to address column and row name attributes
colnames(mat) <- c("Var1","Var2")
mat
```

```
##      Var1 Var2
## [1,]    1  6.0
## [2,]   -2  2.2
## [3,]    4 12.0
```

```
cat("\n")
```

```
rownames(mat) <- c("a","b","c")
mat
```

```
##   Var1 Var2
## a    1  6.0
## b   -2  2.2
## c    4 12.0
```

```
#now look at attributes again
cat("\n")    # blank line
```

```
attributes(mat)
```

```
## $dim
## [1] 3 2
##
## $dimnames
## $dimnames[[1]]
## [1] "a" "b" "c"
##
## $dimnames[[2]]
## [1] "Var1" "Var2"
```

use seq to generate matrix data

Now we can create a larger matrix with sequenced numbers.

```
# create a larger matrix using sequenced numbers
mat <- matrix(data = 1:20, nrow = 4, ncol = 5)
mat
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    5    9   13   17
## [2,]    2    6   10   14   18
## [3,]    3    7   11   15   19
## [4,]    4    8   12   16   20
```

```
cat("\n")
```

```
str(mat)
```

```
##  int [1:4, 1:5] 1 2 3 4 5 6 7 8 9 10 ...
```

Read by rows

What happens if we read in by rows?

```
# create a larger matrix using sequenced numbers, read in by row
mat <- matrix(data = 1:20, nrow = 4, ncol = 5, byrow=T)
mat
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
## [3,]   11   12   13   14   15
## [4,]   16   17   18   19   20
```

```
cat("\n")
```

```
str(mat)
```

```
##  int [1:4, 1:5] 1 6 11 16 2 7 12 17 3 8 ...
```

Note that matrix data is always stored column-wise, even if read in row-wise.

A few matrix operations

There are many useful matrix operations.

```
mat
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
## [3,]   11   12   13   14   15
## [4,]   16   17   18   19   20
```

```
cat("\nTranspose matrix\n")
```

```
##
## Transpose matrix
```

```
t(mat)    # transpose matrix
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    6   11   16
## [2,]    2    7   12   17
## [3,]    3    8   13   18
## [4,]    4    9   14   19
## [5,]    5   10   15   20
```

```
# num [1:3, 1:2] 1 -2 4 6 2.2 12

cat("\nAnother way to reshape matrix\n")
```

```
##
## Another way to reshape matrix
```

```
dim(mat) <- c(5,4)
mat
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    7   13   19
## [2,]    6   12   18    5
## [3,]   11   17    4   10
## [4,]   16    3    9   15
## [5,]    2    8   14   20
```

```
#cat("\nAnother way to reshape matrix\n")
#dim(mat1) <- c(4,3)
#mat1

cat("\nCreate square matrix\n")
```

```
##
## Create square matrix
```

```
mat <- matrix(c(1,3,5,2,0,0,-1,6,0.5),nrow=3,ncol=3)
mat
```

```
##      [,1] [,2] [,3]
## [1,]    1    2 -1.0
## [2,]    3    0  6.0
## [3,]    5    0  0.5
```

```
cat("\nInvert square matrix\n")
```

```
##
## Invert square matrix
```

```
solve(mat)
```

```
##      [,1]        [,2]       [,3]
## [1,]  0.0 -0.01754386  0.2105263
## [2,]  0.5  0.09649123 -0.1578947
## [3,]  0.0  0.17543860 -0.1052632
```

```
cat("\nmultiply matrices\n")
```

```
##
## multiply matrices
```

```
round(solve(mat) %*% mat, 3)  # round off multiplication
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

functions

For expediency, you can write your own functions

```
?sq   # there is no sq function, but we can create one
```

```
## No documentation for 'sq' in specified packages and libraries:
## you could try '??sq'
```

```
sq <- function(x) {
  y <- x*x
  return(y)
}

x <- 4
sq(4)
```

```
## [1] 16
```

Notice that there is no y or x value in our environment. Variables created within a function are not saved outside of the function. You need to return these values or otherwise save them if needed. There is a new sq object in our environment.

```
#source("Rlib.txt")
```