**תרגיל 1.1.9**

**Q1**: Why is 1 == "1" true? Why is -1 < FALSE true? Why is "one" < 2 false?

**A**: These comparisons are carried out by operator-functions ([==](https://rdrr.io/r/base/Comparison.html), [<](https://rdrr.io/r/base/Comparison.html)), which coerce their arguments to a common type. In the examples above, these types will be character, double and character: 1 will be coerced to "1", FALSE is represented as 0 and 2 turns into "2" (and numbers precede letters in lexicographic order (may depend on locale)).

**Q2**: Precisely what do [is.atomic()](https://rdrr.io/r/base/is.recursive.html), [is.numeric()](https://rdrr.io/r/base/numeric.html), and [is.vector()](https://rdrr.io/r/base/vector.html) test for?

**A**: The documentation states that:

* [is.atomic()](https://rdrr.io/r/base/is.recursive.html) tests if an object is an atomic vector (as defined in *Advanced R*) or is NULL (!).
* [is.numeric()](https://rdrr.io/r/base/numeric.html) tests if an object has type integer or double and is not of class factor, Date, POSIXt or difftime.
* [is.vector()](https://rdrr.io/r/base/vector.html) tests if an object is a vector (as defined in *Advanced R*) or an expression and has no attributes, apart from names.

Atomic vectors are defined in *Advanced R* as objects of type logical, integer, double, complex, character or raw. Vectors are defined as atomic vectors or lists.

**תרגיל 1.5**

**Q1**: What does [dim()](https://rdrr.io/r/base/dim.html) return when applied to a 1-dimensional vector? When might you use [NROW()](https://rdrr.io/r/base/nrow.html) or [NCOL()](https://rdrr.io/r/base/nrow.html)?

**A**: From [?nrow](https://rdrr.io/r/base/nrow.html):

[dim()](https://rdrr.io/r/base/dim.html) will return NULL when applied to a 1d vector.

One may want to use [NROW()](https://rdrr.io/r/base/nrow.html) or [NCOL()](https://rdrr.io/r/base/nrow.html) to handle atomic vectors, lists and NULL values in the same way as one column matrices or data frames. For these objects [nrow()](https://rdrr.io/r/base/nrow.html) and [ncol()](https://rdrr.io/r/base/nrow.html) return NULL:

x <- 1:10

# Return NULL

[nrow](https://rdrr.io/r/base/nrow.html)(x)

#> NULL

[ncol](https://rdrr.io/r/base/nrow.html)(x)

#> NULL

# Pretend it's a column vector

[NROW](https://rdrr.io/r/base/nrow.html)(x)

#> [1] 10

[NCOL](https://rdrr.io/r/base/nrow.html)(x)

#> [1] 1

**Q2**: What sort of object does [table()](https://rdrr.io/r/base/table.html) return? What is its type? What attributes does it have? How does the dimensionality change as you tabulate more variables?

**A**: [table()](https://rdrr.io/r/base/table.html) returns a contingency table of its input variables. It is implemented as an integer vector with class table and dimensions (which makes it act like an array). Its attributes are dim (dimensions) and dimnames (one name for each input column). The dimensions correspond to the number of unique values (factor levels) in each input variable.

x <- [table](https://rdrr.io/r/base/table.html)(mtcars[[c](https://rdrr.io/r/base/c.html)("vs", "cyl", "am")])

[typeof](https://rdrr.io/r/base/typeof.html)(x)

#> [1] "integer"

[attributes](https://rdrr.io/r/base/attributes.html)(x)

#> $dim

#> [1] 2 3 2

#>

#> $dimnames

#> $dimnames$vs

#> [1] "0" "1"

#>

#> $dimnames$cyl

#> [1] "4" "6" "8"

#>

#> $dimnames$am

#> [1] "0" "1"

#>

#>

#> $class

#> [1] "table"

# Subset x like it's an array

x[ , , 1]

#> cyl

#> vs 4 6 8

#> 0 0 0 12

#> 1 3 4 0

x[ , , 2]

#> cyl

#> vs 4 6 8

#> 0 1 3 2

#> 1 7 0 0

**Q3**: What happens to a factor when you modify its levels?

f1 <- [factor](https://rdrr.io/r/base/factor.html)(letters)

[levels](https://rdrr.io/r/base/levels.html)(f1) <- [rev](https://rdrr.io/r/base/rev.html)([levels](https://rdrr.io/r/base/levels.html)(f1))

**A**: The underlying integer values stay the same, but the levels are changed, making it look like the data has changed.

f1 <- [factor](https://rdrr.io/r/base/factor.html)(letters)

f1

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

#> Levels: a b c d e f g h i j k l m n o p q r s t u v w x y z

[as.integer](https://rdrr.io/r/base/integer.html)(f1)

#> [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

[levels](https://rdrr.io/r/base/levels.html)(f1) <- [rev](https://rdrr.io/r/base/rev.html)([levels](https://rdrr.io/r/base/levels.html)(f1))

f1

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

#> Levels: z y x w v u t s r q p o n m l k j i h g f e d c b a

[as.integer](https://rdrr.io/r/base/integer.html)(f1)

#> [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

**Q4**: Why do you need to use [unlist()](https://rdrr.io/r/base/unlist.html) to convert a list to an atomic vector? Why doesn’t [as.vector()](https://rdrr.io/r/base/vector.html) work?

**A**: A list is already a vector, though not an atomic one!

Note that [as.vector()](https://rdrr.io/r/base/vector.html) and [is.vector()](https://rdrr.io/r/base/vector.html) use different definitions of “vector!”

[is.vector](https://rdrr.io/r/base/vector.html)([as.vector](https://rdrr.io/r/base/vector.html)(mtcars))

#> [1] FALSE

**תרגיל 1.7**

**Q1**: Can you have a data frame with zero rows? What about zero columns?

**A**: Yes, you can create these data frames easily; either during creation or via subsetting. Even both dimensions can be zero.

Create a 0-row, 0-column, or an empty data frame directly:

[data.frame](https://rdrr.io/r/base/data.frame.html)(a = [integer](https://rdrr.io/r/base/integer.html)(), b = [logical](https://rdrr.io/r/base/logical.html)())

#> [1] a b

#> <0 rows> (or 0-length row.names)

[data.frame](https://rdrr.io/r/base/data.frame.html)(row.names = 1:3) # or data.frame()[1:3, ]

#> data frame with 0 columns and 3 rows

[data.frame](https://rdrr.io/r/base/data.frame.html)()

#> data frame with 0 columns and 0 rows

Create similar data frames via subsetting the respective dimension with either 0, NULL, FALSE or a valid 0-length atomic ([logical(0)](https://rdrr.io/r/base/logical.html), [character(0)](https://rdrr.io/r/base/character.html), [integer(0)](https://rdrr.io/r/base/integer.html), [double(0)](https://rdrr.io/r/base/double.html)). Negative integer sequences would also work. The following example uses a zero:

mtcars[0, ]

#> [1] mpg cyl disp hp drat wt qsec vs am gear carb

#> <0 rows> (or 0-length row.names)

mtcars[ , 0] # or mtcars[0]

#> data frame with 0 columns and 32 rows

mtcars[0, 0]

#> data frame with 0 columns and 0 rows

**Q2**: If df is a data frame, what can you say about [t(df)](https://rdrr.io/r/base/t.html), and [t(t(df))](https://rdrr.io/r/base/t.html)? Perform some experiments, making sure to try different column types.

**A**: Both of [t(df)](https://rdrr.io/r/base/t.html) and [t(t(df))](https://rdrr.io/r/base/t.html) will return matrices:

df <- [data.frame](https://rdrr.io/r/base/data.frame.html)(x = 1:3, y = letters[1:3])

[is.matrix](https://rdrr.io/r/base/matrix.html)(df)

#> [1] FALSE

[is.matrix](https://rdrr.io/r/base/matrix.html)([t](https://rdrr.io/r/base/t.html)(df))

#> [1] TRUE

[is.matrix](https://rdrr.io/r/base/matrix.html)([t](https://rdrr.io/r/base/t.html)([t](https://rdrr.io/r/base/t.html)(df)))

#> [1] TRUE

The dimensions will respect the typical transposition rules:

[dim](https://rdrr.io/r/base/dim.html)(df)

#> [1] 3 2

[dim](https://rdrr.io/r/base/dim.html)([t](https://rdrr.io/r/base/t.html)(df))

#> [1] 2 3

[dim](https://rdrr.io/r/base/dim.html)([t](https://rdrr.io/r/base/t.html)([t](https://rdrr.io/r/base/t.html)(df)))

#> [1] 3 2

Because the output is a matrix, every column is coerced to the same type. (It is implemented within [t.data.frame()](https://rdrr.io/r/base/t.html) via [as.matrix()](https://rdrr.io/r/base/matrix.html) which is described below).

df

#> x y

#> 1 1 a

#> 2 2 b

#> 3 3 c

[t](https://rdrr.io/r/base/t.html)(df)

#> [,1] [,2] [,3]

#> x "1" "2" "3"

#> y "a" "b" "c"

**Q3**: What does [as.matrix()](https://rdrr.io/r/base/matrix.html) do when applied to a data frame with columns of different types? How does it differ from [data.matrix()](https://rdrr.io/r/base/data.matrix.html)?

**A**: The type of the result of as.matrix depends on the types of the input columns (see [?as.matrix](https://rdrr.io/r/base/matrix.html)):

The method for data frames will return a character matrix if there is only atomic columns and any non-(numeric/logical/complex) column, applying as.vector to factors and format to other non-character columns. Otherwise the usual coercion hierarchy (logical < integer < double < complex) will be used, e.g. all-logical data frames will be coerced to a logical matrix, mixed logical-integer will give an integer matrix, etc.

On the other hand, data.matrix will always return a numeric matrix (see ?data.matrix()).

Return the matrix obtained by converting all the variables in a data frame to numeric mode and then binding them together as the columns of a matrix. Factors and ordered factors are replaced by their internal codes. […] Character columns are first converted to factors and then to integers.

We can illustrate and compare the mechanics of these functions using a concrete example. [as.matrix()](https://rdrr.io/r/base/matrix.html) makes it possible to retrieve most of the original information from the data frame but leaves us with characters. To retrieve all information from [data.matrix()](https://rdrr.io/r/base/data.matrix.html)’s output, we would need a lookup table for each column.

df\_coltypes <- [data.frame](https://rdrr.io/r/base/data.frame.html)(

a = [c](https://rdrr.io/r/base/c.html)("a", "b"),

b = [c](https://rdrr.io/r/base/c.html)(TRUE, FALSE),

c = [c](https://rdrr.io/r/base/c.html)(1L, 0L),

d = [c](https://rdrr.io/r/base/c.html)(1.5, 2),

e = [factor](https://rdrr.io/r/base/factor.html)([c](https://rdrr.io/r/base/c.html)("f1", "f2"))

)

[as.matrix](https://rdrr.io/r/base/matrix.html)(df\_coltypes)

#> a b c d e

#> [1,] "a" "TRUE" "1" "1.5" "f1"

#> [2,] "b" "FALSE" "0" "2.0" "f2"

[data.matrix](https://rdrr.io/r/base/data.matrix.html)(df\_coltypes)

#> a b c d e

#> [1,] 1 1 1 1.5 1

#> [2,] 2 0 0 2.0 2

**תרגיל 3**

**Q1**: Fix each of the following common data frame subsetting errors:

mtcars[mtcars$cyl = 4, ]

# use `==` (instead of `=`)

mtcars[-1:4, ]

# use `-(1:4)` (instead of `-1:4`)

mtcars[mtcars$cyl <= 5]

# `,` is missing

mtcars[mtcars$cyl == 4 | 6, ]

# use `mtcars$cyl == 6` (instead of `6`)

# or `%in% c(4, 6)` (instead of `== 4 | 6`)

**Q2**: What does df[is.na(df)] <- 0 do? How does it work?

**A**: This expression replaces the NAs in df with 0. Here [is.na(df)](https://rdrr.io/r/base/NA.html) returns a logical matrix that encodes the position of the missing values in df. Subsetting and assignment are then combined to replace only the missing values.

**Q3**: Brainstorm as many ways as possible to extract the third value from the cyl variable in the mtcars dataset.

**A**: Base R already provides an abundance of possibilities:

# Select column first

mtcars$cyl[[3]]

#> [1] 4

mtcars[ , "cyl"][[3]]

#> [1] 4

mtcars[["cyl"]][[3]]

#> [1] 4

[with](https://rdrr.io/r/base/with.html)(mtcars, cyl[[3]])

#> [1] 4

# Select row first

mtcars[3, ]$cyl

#> [1] 4

mtcars[3, "cyl"]

#> [1] 4

mtcars[3, ][ , "cyl"]

#> [1] 4

mtcars[3, ][["cyl"]]

#> [1] 4

# Select simultaneously

mtcars[3, 2]

#> [1] 4

mtcars[[[c](https://rdrr.io/r/base/c.html)(2, 3)]]

#> [1] 4

**Q4**: Given a linear model, e.g. mod <- lm(mpg ~ wt, data = mtcars), extract the residual degrees of freedom. Extract the R squared from the model summary ([summary(mod)](https://rdrr.io/r/base/summary.html)).

**A**: mod is of type list, which opens up several possibilities. We use [$](https://rdrr.io/r/base/Extract.html) or [[[](https://rdrr.io/r/base/Extract.html) to extract a single element:

mod <- [lm](https://rdrr.io/r/stats/lm.html)(mpg ~ wt, data = mtcars)

mod$df.residual

#> [1] 30

mod[["df.residual"]]

#> [1] 30

The same also applies to [summary(mod)](https://rdrr.io/r/base/summary.html), so we could use, e.g.:

[summary](https://rdrr.io/r/base/summary.html)(mod)$r.squared

#> [1] 0.753

(Tip: The [{broom} package](https://github.com/tidymodels/broom)11 provides a very useful approach to work with models in a tidy way.)

**Q5**: How could you put the columns in a data frame in alphabetical order?

**A**: We combine [[](https://rdrr.io/r/base/Extract.html) with [order()](https://rdrr.io/r/base/order.html) or [sort()](https://rdrr.io/r/base/sort.html):

mtcars[[order](https://rdrr.io/r/base/order.html)([names](https://rdrr.io/r/base/names.html)(mtcars))]

mtcars[[sort](https://rdrr.io/r/base/sort.html)([names](https://rdrr.io/r/base/names.html)(mtcars))]