**R Vector**

Vector is a basic data structure in R. It contains element of the same type. The data types can be logical, integer, double, character, complex or raw.

A vector’s type can be checked with the typeof()

Another important property of a vector is its length. This is the number of elements in the vector and can be checked with the function length().

## How to Create Vector in R?

Vectors are generally created using the c() function.

> x <- c(1, 5, 4, 9, 0)

> typeof(x)

[1] "double"

> length(x)

[1] 5

> x <- c(1, 5.4, TRUE, "hello")

> x

[1] "1" "5.4" "TRUE" "hello"

> typeof(x)

[1] "character"

#### Example 1: Creating a vector using : operator

> x <- 1:7;

[1] 1 2 3 4 5 6 7

> y <- 2:-2; y

[1] 2 1 0 -1 -2

More complex sequences can be created using the seq() function, like defining number of points in an interval, or the step size.

#### Example 2: Creating a vector using seq() function

> seq(1, 3, by=0.2) # specify step size

[1] 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0

> seq(1, 5, length.out=4) # specify length of the vector

[1] 1.000000 2.333333 3.666667 5.000000

## How to access Elements of a Vector?

Elements of a vector can be accessed using vector indexing. The vector used for indexing can be logical, integer or character vector.

### Using integer vector as index

Vector index in R starts from 1, unlike most programming languages where index start from 0.

# R Matrix

Matrix is a two dimensional data structure in R programming.

Matrix is similar to [vector](https://www.datamentor.io/r-programming/vector) but additionally contains the dimension attribute. All attributes of an object can be checked with the attributes() function (dimension can be checked directly with the dim() function).

We can check if a variable is a matrix or not with the class() function.

## How to create a matrix in R programming?

Matrix can be created using the matrix() function.

Dimension of the matrix can be defined by passing appropriate value for arguments nrow and ncol.

Providing value for both dimension is not necessary. If one of the dimension is provided, the other is inferred from length of the data.

> matrix(1:9, nrow = 3, ncol = 3)

[,1] [,2] [,3]

[1,] 1 4 7

[2,] 2 5 8

[3,] 3 6 9

> # same result is obtained by providing only one dimension

> matrix(1:9, nrow = 3)

[,1] [,2] [,3]

[1,] 1 4 7

[2,] 2 5 8

[3,] 3 6 9

We can see that the matrix is filled column-wise. This can be reversed to row-wise filling by passing TRUE to the argument byrow.

> matrix(1:9, nrow=3, byrow=TRUE) # fill matrix row-wise

[,1] [,2] [,3]

[1,] 1 2 3

[2,] 4 5 6

[3,] 7 8 9

In all cases, however, a matrix is stored in column-major order internally as we will see in the subsequent sections.

It is possible to name the rows and columns of matrix during creation by passing a 2 element list to the argument dimnames.

> x <- matrix(1:9, nrow = 3, dimnames = list(c("X","Y","Z"), c("A","B","C")))

> x

A B C

X 1 4 7

Y 2 5 8

Z 3 6 9

These names can be accessed or changed with two helpful functions colnames() and rownames().

> colnames(x)

[1] "A" "B" "C"

> rownames(x)

[1] "X" "Y" "Z"

> # It is also possible to change names

> colnames(x) <- c("C1","C2","C3")

> rownames(x) <- c("R1","R2","R3")

> x

C1 C2 C3

R1 1 4 7

R2 2 5 8

R3 3 6 9

Another way of creating a matrix is by using functions cbind() and rbind() as in column bind and row bind.

> cbind(c(1,2,3),c(4,5,6))

[,1] [,2]

[1,] 1 4

[2,] 2 5

[3,] 3 6

> rbind(c(1,2,3),c(4,5,6))

[,1] [,2] [,3]

[1,] 1 2 3

[2,] 4 5 6

Finally, you can also create a matrix from a vector by setting its dimension using dim().

> x <- c(1,2,3,4,5,6)

> x

[1] 1 2 3 4 5 6

> class(x)

[1] "numeric"

> dim(x) <- c(2,3)

> x

[,1] [,2] [,3]

[1,] 1 3 5

[2,] 2 4 6

> class(x)

[1] "matrix"

## How to access Elements of a matrix?

We can access elements of a matrix using the square bracket [ indexing method. Elements can be accessed as var[row, column]. Here rows and columns are vectors.

### Using integer vector as index

We specify the row numbers and column numbers as vectors and use it for indexing.

If any field inside the bracket is left blank, it selects all.

We can use negative integers to specify rows or columns to be excluded.

> x

[,1] [,2] [,3]

[1,] 1 4 7

[2,] 2 5 8

[3,] 3 6 9

> x[c(1,2),c(2,3)] # select rows 1 & 2 and columns 2 & 3

[,1] [,2]

[1,] 4 7

[2,] 5 8

> x[c(3,2),] # leaving column field blank will select entire columns

[,1] [,2] [,3]

[1,] 3 6 9

[2,] 2 5 8

> x[,] # leaving row as well as column field blank will select entire matrix

[,1] [,2] [,3]

[1,] 1 4 7

[2,] 2 5 8

[3,] 3 6 9

> x[-1,] # select all rows except first

[,1] [,2] [,3]

[1,] 2 5 8

[2,] 3 6 9

## How to modify a matrix in R?

We can combine assignment operator with the above learned methods for accessing elements of a matrix to modify it.

> x

[,1] [,2] [,3]

[1,] 1 4 7

[2,] 2 5 8

[3,] 3 6 9

> x[2,2] <- 10; x # modify a single element

[,1] [,2] [,3]

[1,] 1 4 7

[2,] 2 10 8

[3,] 3 6 9

> x[x<5] <- 0; x # modify elements less than 5

[,1] [,2] [,3]

[1,] 0 0 7

[2,] 0 10 8

[3,] 0 6 9

# R Lists

List is a data structure having components of mixed data types.

A [vector](https://www.datamentor.io/r-programming/vector) having all elements of the same type is called atomic vector but a vector having elements of different type is called list.

We can check if it’s a list with typeof() function and find its length using length().

Following is an example of a list having three components each of different data type.

> x

$a

[1] 2.5

$b

[1] TRUE

$c

[1] 1 2 3

> typeof(x)

[1] "list"

> length(x)

[1] 3

## How to create a list in R programming?

List can be created using the list() function.

> x <- list("a" = 2.5, "b" = TRUE, "c" = 1:3)

Here, we create a list x, of three components with data types double, logical and integervector respectively.

Its structure can be examined with the str() function.

> str(x)

List of 3

$ a: num 2.5

$ b: logi TRUE

$ c: int [1:3] 1 2 3

In this example, a, b and c are called tags which makes it easier to reference the components of the list.

However, tags are optional. We can create the same list without the tags as follows. In such scenario, numeric indices are used by default.

> x <- list(2.5,TRUE,1:3)

> x

[[1]]

[1] 2.5

[[2]]

[1] TRUE

[[3]]

[1] 1 2 3

## How to access components of a list?

Lists can be accessed in similar fashion to vectors. Integer, logical or character vectors can be used for indexing. Let us consider a list as follows.

> x

$name

[1] "John"

$age

[1] 19

$speaks

[1] "English" "French"

> x[c(1:2)] # index using integer vector

$name

[1] "John"

$age

[1] 19

> x[-2] # using negative integer to exclude second component

$name

[1] "John"

$speaks

[1] "English" "French"

> x[c(T,F,F)] # index using logical vector

$name

[1] "John"

> x[c("age","speaks")] # index using character vector

$age

[1] 19

$speaks

[1] "English" "French"

## How to modify a list in R?

We can change components of a list through reassignment. We can choose any of the component accessing techniques discussed above to modify it.

Notice below that modification causes reordering of components.

> x[["name"]] <- "Clair"; x

$age

[1] 19

$speaks

[1] "English" "French"

$name

[1] "Clair"

### How to add components to a list?

Adding new components is easy. We simply assign values using new tags and it will pop into action.

> x[["married"]] <- FALSE

> x

$age

[1] 19

$speaks

[1] "English" "French"

$name

[1] "Clair"

$married

[1] FALSE

### How to delete components from a list?

We can delete a component by assigning NULL to it.

> x[["age"]] <- NULL

> str(x)

List of 3

$ speaks : chr [1:2] "English" "French"

$ name : chr "Clair"

$ married: logi FALSE

> x$married <- NULL

> str(x)

List of 2

$ speaks: chr [1:2] "English" "French"

$ name : chr "Clair"

# R Data Frame

### Check if a variable is a data frame or not

We can check if a variable is a data frame or not using the class() function.

> x

SN Age Name

1 1 21 John

2 2 15 Dora

> typeof(x) # data frame is a special case of list

[1] "list"

> class(x)

[1] "data.frame"

In this example, x can be considered as a list of 3 components with each component having a two element vector. Some useful functions to know more about a data frame are given below.

### Functions of data frame

> names(x)

[1] "SN" "Age" "Name"

> ncol(x)

[1] 3

> nrow(x)

[1] 2

> length(x) # returns length of the list, same as ncol()

[1] 3

## How to create a Data Frame in R?

We can create a data frame using the data.frame() function.

For example, the above shown data frame can be created as follows.

> x <- data.frame("SN" = 1:2, "Age" = c(21,15), "Name" = c("John","Dora"))

> str(x) # structure of x

'data.frame': 2 obs. of 3 variables:

$ SN : int 1 2

$ Age : num 21 15

$ Name: Factor w/ 2 levels "Dora","John": 2 1

Notice above that the third column, Name is of type [factor](https://www.datamentor.io/r-programming/factor), instead of a character [vector](https://www.datamentor.io/r-programming/vector).

By default, data.frame() function converts character vector into factor.

To suppress this behavior, we can pass the argument stringsAsFactors=FALSE.

> x <- data.frame("SN" = 1:2, "Age" = c(21,15), "Name" = c("John", "Dora"), stringsAsFactors = FALSE)

> str(x) # now the third column is a character vector

'data.frame': 2 obs. of 3 variables:

$ SN : int 1 2

$ Age : num 21 15

$ Name: chr "John" "Dora"

Many data input functions of R like, read.table(), read.csv(), read.delim(), read.fwf() also read data into a data frame.

## How to access Components of a Data Frame?

Components of data frame can be accessed like a list or like a matrix.

### Accessing like a list

We can use either [, [[ or $ operator to access columns of data frame.

> x["Name"]

Name

1 John

2 Dora

> x$Name

[1] "John" "Dora"

> x[["Name"]]

[1] "John" "Dora"

> x[[3]]

[1] "John" "Dora"

Accessing with [[ or $ is similar. However, it differs for [ in that, indexing with [ will return us a data frame but the other two will reduce it into a vector.

### Accessing like a matrix

Data frames can be accessed like a matrix by providing index for row and column.

To illustrate this, we use datasets already available in R. Datasets that are available can be listed with the command library(help = "datasets").

We will use the trees dataset which contains Girth, Height and Volume for Black Cherry Trees.

A data frame can be examined using functions like str() and head().

> str(trees)

'data.frame': 31 obs. of 3 variables:

$ Girth : num 8.3 8.6 8.8 10.5 10.7 10.8 11 11 11.1 11.2 ...

$ Height: num 70 65 63 72 81 83 66 75 80 75 ...

$ Volume: num 10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 ...

> head(trees,n=3)

Girth Height Volume

1 8.3 70 10.3

2 8.6 65 10.3

3 8.8 63 10.2

We can see that trees is a data frame with 31 rows and 3 columns. We also display the first 3 rows of the data frame.

Now we proceed to access the data frame like a matrix.

> trees[2:3,] # select 2nd and 3rd row

Girth Height Volume

2 8.6 65 10.3

3 8.8 63 10.2

> trees[trees$Height > 82,] # selects rows with Height greater than 82

Girth Height Volume

6 10.8 83 19.7

17 12.9 85 33.8

18 13.3 86 27.4

31 20.6 87 77.0

> trees[10:12,2]

[1] 75 79 76

We can see in the last case that the returned type is a vector since we extracted data from a single column.

This behavior can be avoided by passing the argument drop=FALSE as follows.

> trees[10:12,2, drop = FALSE]

Height

10 75

11 79

12 76

## How to modify a Data Frame in R?

Data frames can be modified like we modified matrices through reassignment.

> x

SN Age Name

1 1 21 John

2 2 15 Dora

> x[1,"Age"] <- 20; x

SN Age Name

1 1 20 John

2 2 15 Dora

### Adding Components

Rows can be added to a data frame using the rbind() function.

> rbind(x,list(1,16,"Paul"))

SN Age Name

1 1 20 John

2 2 15 Dora

3 1 16 Paul

Similarly, we can add columns using cbind().

> cbind(x,State=c("NY","FL"))

SN Age Name State

1 1 20 John NY

2 2 15 Dora FL

Since data frames are implemented as list, we can also add new columns through simple list-like assignments.

> x

SN Age Name

1 1 20 John

2 2 15 Dora

> x$State <- c("NY","FL"); x

SN Age Name State

1 1 20 John NY

2 2 15 Dora FL

### Deleting Component

Data frame columns can be deleted by assigning NULL to it.

> x$State <- NULL

> x

SN Age Name

1 1 20 John

2 2 15 Dora

Similarly, rows can be deleted through reassignments.

> x <- x[-1,]

> x

SN Age Name

2 2 15 Dora

# R Factors

Factor is a data structure used for fields that takes only predefined, finite number of values (categorical data).

For example, a data field such as marital status may contain only values from single, married, separated, divorced, or widowed.

In such case, we know the possible values beforehand and these predefined, distinct values are called levels. Following is an example of factor in R.

> x

[1] single married married single

Levels: married single

Here, we can see that factor x has four elements and two levels. We can check if a variable is a factor or not using class() function.

Similarly, levels of a factor can be checked using the levels() function.

> class(x)

[1] "factor"

> levels(x)

[1] "married" "single"

## How to create a factor in R?

We can create a factor using the function factor(). Levels of a factor are inferred from the data if not provided.

> x <- factor(c("single", "married", "married", "single"));

> x

[1] single married married single

Levels: married single

> x <- factor(c("single", "married", "married", "single"), levels = c("single", "married", "divorced"));

> x

[1] single married married single

Levels: single married divorced

We can see from the above example that levels may be predefined even if not used.

Factors are closely related with [vectors](https://www.datamentor.io/r-programming/vector). In fact, factors are stored as integer vectors. This is clearly seen from its structure.

> x <- factor(c("single","married","married","single"))

> str(x)

Factor w/ 2 levels "married","single": 2 1 1 2

We see that levels are stored in a character vector and the individual elements are actually stored as indices.

## How to access compoments of a factor?

Accessing components of a factor is very much similar to that of vectors.

> x

[1] single married married single

Levels: married single

> x[3] # access 3rd element

[1] married

Levels: married single

> x[c(2, 4)] # access 2nd and 4th element

[1] married single

Levels: married single

> x[-1] # access all but 1st element

[1] married married single

Levels: married single

> x[c(TRUE, FALSE, FALSE, TRUE)] # using logical vector

[1] single single

Levels: married single

## How to modify a factor?

Components of a factor can be modified using simple assignments. However, we cannot choose values outside of its predefined levels.

> x

[1] single married married single

Levels: single married divorced

> x[2] <- "divorced" # modify second element; x

[1] single divorced married single

Levels: single married divorced

> x[3] <- "widowed" # cannot assign values outside levels

Warning message:

In `[<-.factor`(`\*tmp\*`, 3, value = "widowed") :

invalid factor level, NA generated

> x

[1] single divorced <NA> single

Levels: single married divorced

A workaround to this is to add the value to the level first.

> levels(x) <- c(levels(x), "widowed") # add new level

> x[3] <- "widowed"

> x

[1] single divorced widowed single

Levels: single married divorced widowed