

Chapter 2

Basics of R

2.1 Matrices

A matrix is a two dimensional data set with columns and rows. A column is a vertical representation of data, while a row is a horizontal representation of data. A matrix can be created with the **matrix()** function. Specify the **nrow** and **ncol** parameters to get the amount of rows and columns:

Example

```
# Create a matrix
thismatrix <- matrix(c(1,2,3,4,5,6), nrow = 3, ncol = 2)

# Print the matrix
thismatrix
```

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

Example

```
thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2,
ncol = 2)

thismatrix
```

```
      [,1]      [,2]
[1,] "apple"  "cherry"
[2,] "banana" "orange"
```

Access Matrix Items

You can access the items by using **[]** brackets. The first number "1" in the bracket specifies the row-position, while the second number "2" specifies the column-position:

Example

```
thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2,  
ncol = 2)
```

```
thismatrix[1, 2]
```

```
[1] "cherry"
```

The whole row can be accessed if you specify a comma **after** the number in the bracket:

Example

```
thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2,  
ncol = 2)
```

```
thismatrix[2,]
```

```
[1] "banana" "orange"
```

The whole column can be accessed if you specify a comma **before** the number in the bracket:

Example

```
thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2,  
ncol = 2)
```

```
thismatrix[,2]
```

```
[1] "cherry" "orange"
```

2.2 Arrays

Compared to matrices, arrays can have more than two dimensions. We can use the **array()** function to create an array, and the **dim** parameter to specify the dimensions:

Example

```
# An array with one dimension with values ranging from 1 to 24  
thisarray <- c(1:24)  
thisarray
```

```
# An array with more than one dimension
multiarray <- array(thisarray, dim = c(4, 3, 2))
multiarray
```

```
[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
, , 1

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

, , 2

      [,1] [,2] [,3]
[1,]   13   17   21
[2,]   14   18   22
[3,]   15   19   23
[4,]   16   20   24
```

Explanation of the above Example

- In the example above we create an array with the values 1 to 24.
- How does `dim=c(4,3,2)` work?
The first and second number in the bracket specifies the amount of rows and columns.
The last number in the bracket specifies how many dimensions we want.

Note: Arrays can only have one data type.

```
# An array with more than one dimension
multiarray <- array(thisarray, dim = c(3, 4, 2))
multiarray
```

```
[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
, , 1

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

, , 2

      [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24
```

Access Array Items: You can access the array elements by referring to the **index** position. You can use the `[]` brackets to access the desired elements from an array.

Example

```
thisarray <- c(1:24)
multiarray <- array(thisarray, dim = c(4, 3, 2))

multiarray[2, 3, 2]
```

```
[1] 22
```

Example 2

```
thisarray <- c(1:24)
multiarray <- array(thisarray, dim = c(4, 3, 2))

multiarray[4, 3, 1]
```

```
[1] 12
```

The syntax is as follow: `array[row position, column position, matrix level]`

You can also access the whole row or column from a matrix in an array, by using the `c()` function:

Example

```
thisarray <- c(1:24)

# Access all the items from the first row from matrix one
multiarray <- array(thisarray, dim = c(4, 3, 2))
multiarray[c(1),,1]

# Access all the items from the first column from matrix one
multiarray <- array(thisarray, dim = c(4, 3, 2))
multiarray[,c(1),1]
```

```
[1] 1 5 9
[1] 1 2 3 4
```

- A comma (,) before c() means that we want to access the column.
- A comma (,) after c() means that we want to access the row.

Check if an Item Exists

To find out if a specified item is present in an array, use the `%in%` operator:

Example 1: Check if the value "2" is present in the array:

```
thisarray <- c(1:24)
multiarray <- array(thisarray, dim = c(4, 3, 2))
```

```
2 %in% multiarray
```

```
[1] TRUE
```

Example 2: Check if the value "25" is present in the array:

```
thisarray <- c(1:24)
multiarray <- array(thisarray, dim = c(4, 3, 2))
```

```
25 %in% multiarray
```

```
[1] FALSE
```

Amount of Rows and Columns: Use the `dim()` function to find the amount of rows and columns in an array:

Example

```
thisarray <- c(1:24)
multiarray <- array(thisarray, dim = c(4, 3, 2))
```

```
dim(multiarray)
```

```
[1] 4 3 2
```

Array Length: Use the `length()` function to find the dimension of an array:

Example

```
thisarray <- c(1:24)
multiarray <- array(thisarray, dim = c(4, 3, 2))
```

```
length(multiarray)
```

```
[1] 24
```

Loop Through an Array: You can loop through the array items by using a **for** loop:

Example

```
thisarray <- c(1:24)
multiarray <- array(thisarray, dim = c(4, 3, 2))

for(x in multiarray){
  print(x)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
[1] 11
[1] 12
[1] 13
[1] 14
[1] 15
[1] 16
[1] 17
[1] 18
[1] 19
[1] 20
[1] 21
[1] 22
[1] 23
[1] 24
```

2.3 Data Frames

Data Frames are data displayed in a format as a **table**. Data Frames can have different types of data inside it. While the first column can be **character**, the second and third can be **numeric** or **logical**. However, each column should have the same type of data. Use the **data.frame()** function to create a data frame:

Example

```
# Create a data frame
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
```

```
Pulse = c(100, 150, 120),
Duration = c(60, 30, 45)
)
```

```
# Print the data frame
Data_Frame
```

	Training	Pulse	Duration
1	Strength	100	60
2	Stamina	150	30
3	Other	120	45

Summarize the Data: Use the `summary()` function to summarize the data from a Data Frame

Example

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)
```

```
Data_Frame
```

```
summary(Data_Frame)
```

	Training	Pulse	Duration
1	Strength	100	60
2	Stamina	150	30
3	Other	120	45
	Training	Pulse	Duration
Other	:1	Min. :100.0	Min. :30.0
Stamina	:1	1st Qu.:110.0	1st Qu.:37.5
Strength	:1	Median :120.0	Median :45.0
		Mean :123.3	Mean :45.0
		3rd Qu.:135.0	3rd Qu.:52.5
		Max. :150.0	Max. :60.0

Access Items: We can use single brackets `[]`, double brackets `[[]]` or `$` to access columns from a data frame

Example 1

```
Data_Frame <- data.frame (  
  Training = c("Strength", "Stamina", "Other"),  
  Pulse = c(100, 150, 120),  
  Duration = c(60, 30, 45)  
)
```

```
Data_Frame[1]
```

```
Data_Frame[["Training"]]
```

```
Data_Frame$Training
```

```
Training  
1 Strength  
2  Stamina  
3   Other  
[1] Strength Stamina  Other  
Levels: Other Stamina Strength  
[1] Strength Stamina  Other  
Levels: Other Stamina Strength
```

Example 2

```
Data_Frame <- data.frame (  
  Training = c("Strength", "Stamina", "Other"),  
  Pulse = c(100, 150, 120),  
  Duration = c(60, 30, 45)  
)
```

```
Data_Frame[2]
```


	Pulse
1	100
2	150
3	120

Add Rows: Use the **rbind()** function to add new rows in a Data Frame

Example

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

# Add a new row
New_row_DF <- rbind(Data_Frame, c("Strength", 110, 110))

# Print the new row
New_row_DF
```

	Training	Pulse	Duration
1	Strength	100	60
2	Stamina	150	30
3	Other	120	45
4	Strength	110	110

Add Columns: Use the **cbind()** function to add new columns in a Data Frame

Example

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

# Add a new column
New_col_DF <- cbind(Data_Frame, Steps = c(1000, 6000, 2000))
```

```
# Print the new column  
New_col_DF
```

	Training	Pulse	Duration	Steps
1	Strength	100	60	1000
2	Stamina	150	30	6000
3	Other	120	45	2000

Remove Rows and Columns: Use the `c()` function to remove rows and columns in a Data Frame:

Example

```
Data_Frame <- data.frame (  
  Training = c("Strength", "Stamina", "Other"),  
  Pulse = c(100, 150, 120),  
  Duration = c(60, 30, 45)  
)
```

```
# Remove the first row and column  
Data_Frame_New <- Data_Frame[-c(1), -c(1)]
```

```
# Print the new data frame  
Data_Frame_New
```

	Pulse	Duration
2	150	30
3	120	45

Amount of Rows and Columns: Use the `dim()` function to find the amount of rows and columns in a Data Frame

Example

```
Data_Frame <- data.frame (  
  Training = c("Strength", "Stamina", "Other"),  
  Pulse = c(100, 150, 120),  
  Duration = c(60, 30, 45)  
)
```

```
dim(Data_Frame)
```

```
[1] 3 3
```

You can also use the **ncol()** function to find the number of columns and **nrow()** to find the number of rows:

Example

```
Data_Frame <- data.frame (  
  Training = c("Strength", "Stamina", "Other"),  
  Pulse = c(100, 150, 120),  
  Duration = c(60, 30, 45)  
)
```

```
ncol(Data_Frame)  
nrow(Data_Frame)
```

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

Data Frame Length: Use the **length()** function to find the number of columns in a Data Frame (similar to **ncol()**)

Example

```
Data_Frame <- data.frame (  
  Training = c("Strength", "Stamina", "Other"),  
  Pulse = c(100, 150, 120),  
  Duration = c(60, 30, 45)  
)
```

```
length(Data_Frame)
```

```
[1] 3
```

Combining Data Frames: Use the **rbind()** function to combine two or more data frames in R vertically

Example

```
Data_Frame1 <- data.frame (  
  Training = c("Strength", "Stamina", "Other"),  
  Pulse = c(100, 150, 120),
```

```

    Duration = c(60, 30, 45)
  )

Data_Frame2 <- data.frame (
  Training = c("Stamina", "Stamina", "Strength"),
  Pulse = c(140, 150, 160),
  Duration = c(30, 30, 20)
)

New_Data_Frame <- rbind(Data_Frame1, Data_Frame2)
New_Data_Frame

```

	Training	Pulse	Duration
1	Strength	100	60
2	Stamina	150	30
3	Other	120	45
4	Stamina	140	30
5	Stamina	150	30
6	Strength	160	20

And use the **cbind()** function to combine two or more data frames in R horizontally:

Example

```

Data_Frame3 <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

Data_Frame4 <- data.frame (
  Steps = c(3000, 6000, 2000),
  Calories = c(300, 400, 300)
)

New_Data_Frame1 <- cbind(Data_Frame3, Data_Frame4)
New_Data_Frame1

```

	Training	Pulse	Duration	Steps	Calories
1	Strength	100	60	3000	300
2	Stamina	150	30	6000	400
3	Other	120	45	2000	300

2.4 Factors

Factors are used to categorize data. Examples of factors are:

- Demography: Male/Female
- Music: Rock, Pop, Classic, Jazz
- Training: Strength, Stamina

To create a factor, use the `factor()` function and add a vector as argument:

Example

```
# Create a factor
music_genre <-
factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"))

# Print the factor
music_genre
```

Result:

```
[1] Jazz    Rock    Classic Classic Pop     Jazz    Rock    Jazz
Levels: Classic Jazz Pop Rock
```

You can see from the example above that the factor has four levels (categories): Classic, Jazz, Pop and Rock. To only print the levels, use the `levels()` function:

Example

```
music_genre <-
factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"))

levels(music_genre)
```

Result:

```
[1] "Classic" "Jazz"    "Pop"     "Rock"
```

You can also set the levels, by adding the **levels** argument inside the **factor()** function:

Example

```
music_genre <-  
factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jaz  
z"), levels = c("Classic", "Jazz", "Pop", "Rock", "Other"))  
  
levels(music_genre)
```

Result:

```
[1] "Classic" "Jazz"    "Pop"     "Rock"    "Other"
```

Factor Length: Use the **length()** function to find out how many items there are in the factor:

Example

```
music_genre <-  
factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jaz  
z"))  
  
length(music_genre)
```

Result:

```
[1] 8
```

Access Factors: To access the items in a factor, refer to the index number, using **[]** brackets:

Example

Access the third item:

```
music_genre <-  
factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jaz  
z"))  
  
music_genre[3]
```

Result:

```
[1] Classic  
Levels: Classic Jazz Pop Rock
```

Change Item Value: To change the value of a specific item, refer to the index number:

Example

Change the value of the third item:

```
music_genre <-  
factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jaz  
z"))
```

```
music_genre[3] <- "Pop"
```

```
music_genre[3]
```

Result:

```
[1] Pop  
Levels: Classic Jazz Pop Rock
```

Note that you cannot change the value of a specific item if it is not already specified in the factor. The following example will produce an error:

Example

Trying to change the value of the third item ("Classic") to an item that does not exist/not predefined ("Opera"):

```
music_genre <-  
factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jaz  
z"))
```

```
music_genre[3] <- "Opera"
```

```
music_genre[3]
```

Result:

```
Warning message:  
In `[<-.factor`(`*tmp*`, 3, value = "Opera") :  
  invalid factor level, NA generated
```

However, if you have already specified it inside the `levels` argument, it will work:

Example

Change the value of the third item:

```
music_genre <-  
factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jaz  
z"), levels = c("Classic", "Jazz", "Pop", "Rock", "Opera"))  
  
music_genre[3] <- "Opera"  
  
music_genre[3]
```

Result:

```
[1] Opera  
Levels: Classic Jazz Pop Rock Opera
```