MAP 2112 – Introdução à Lógica de Programação e Modelagem Computacional

1º Semestre - 2020

Prof. Dr. Luis Carlos de Castro Santos

lsantos@ime.usp.br

NÃO ESQUEÇA DE INICIAR A GRAVAÇÃO

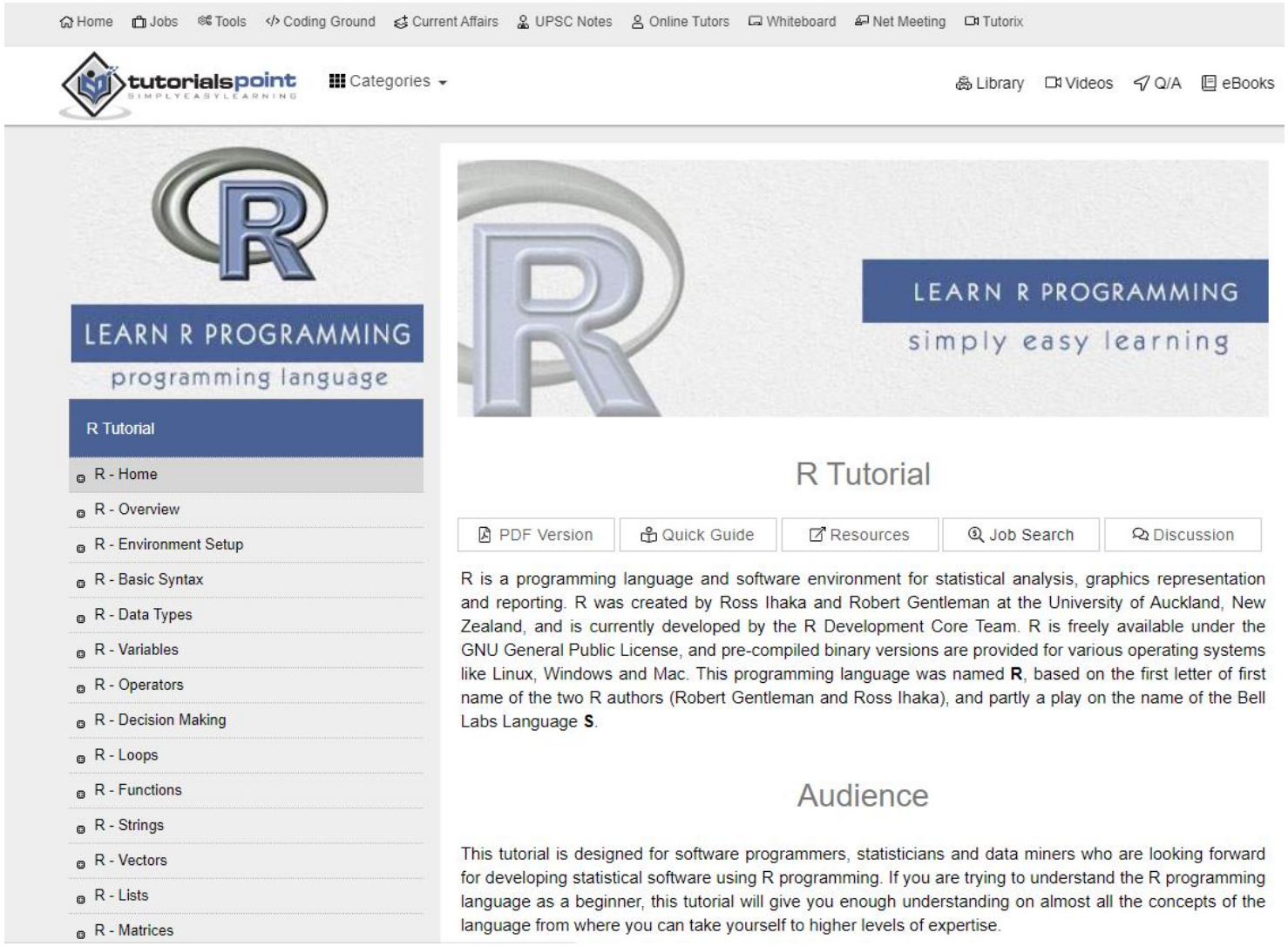
MAP 2112 – Introdução à Lógica de Programação e Modelagem Computacional

1º Semestre - 2020

Prof. Dr. Luis Carlos de Castro Santos

lsantos@ime.usp.br

MATERIAL ADICIONAL EM VETORES, MATRIZES, LISTAS, DATAFRAMES E OPERADORES



https://www.tutorialspoint.com/r/index.htm

R - Vectors

Vectors are the most basic R data objects and there are six types of atomic vectors. They are logical, integer, double, complex, character and raw.

Vector Creation

Single Element Vector

Even when you write just one value in R, it becomes a vector of length 1 and belongs to one of the above vector types.

Accessing Vector Elements

Elements of a Vector are accessed using indexing. The **[ ] brackets** are used for indexing. Indexing starts with position 1. Giving a negative value in the index drops that element from result. **TRUE**, **FALSE** or **0** and **1** can also be used for indexing.

Live Demo

```
# Accessing vector elements using position.
t <- c("Sun","Mon","Tue","Wed","Thurs","Fri","Sat")
u <- t[c(2,3,6)]
print(u)

# Accessing vector elements using logical indexing.
v <- t[c(TRUE,FALSE,FALSE,FALSE,FALSE,TRUE,FALSE)]
print(v)

# Accessing vector elements using negative indexing.
x <- t[c(-2,-5)]
print(x)

# Accessing vector elements using 0/1 indexing.
y <- t[c(0,0,0,0,0,0,1)]
print(y)
```

When we execute the above code, it produces the following result −

```
[1] "Mon" "Tue" "Fri"
[1] "Sun" "Fri"
[1] "Sun" "Tue" "Wed" "Fri" "Sat"
[1] "Sun"
```

Vector Manipulation

Vector arithmetic

Two vectors of same length can be added, subtracted, multiplied or divided giving the result as a vector output.

Live Demo

```
# Create two vectors.
v1 <- c(3,8,4,5,0,11)
v2 <- c(4,11,0,8,1,2)

# Vector addition.
add.result <- v1+v2
print(add.result)

# Vector subtraction.
sub.result <- v1-v2
print(sub.result)

# Vector multiplication.
multi.result <- v1*v2
print(multi.result)

# Vector division.
divi.result <- v1/v2
print(divi.result)
```

When we execute the above code, it produces the following result −

```
[1]  7 19  4 13  1 13
[1] -1 -3  4 -3 -1  9
[1] 12 88  0 40  0 22
[1] 0.7500000 0.7272727       Inf 0.6250000 0.0000000 5.5000000
```

Vector Element Recycling

If we apply arithmetic operations to two vectors of unequal length, then the elements of the shorter vector are recycled to complete the operations.

Live Demo

```
v1 <- c(3,8,4,5,0,11)
v2 <- c(4,11)
# V2 becomes c(4,11,4,11,4,11)

add.result <- v1+v2
print(add.result)

sub.result <- v1-v2
print(sub.result)
```

When we execute the above code, it produces the following result −

```
[1]  7 19  8 16  4 22
[1] -1 -3  0 -6 -4  0
```

Vector Element Sorting

Elements in a vector can be sorted using the **sort()** function.

Live Demo

```
v <- c(3,8,4,5,0,11, -9, 304)

# Sort the elements of the vector.
sort.result <- sort(v)
print(sort.result)

# Sort the elements in the reverse order.
revsort.result <- sort(v, decreasing = TRUE)
print(revsort.result)

# Sorting character vectors.
v <- c("Red","Blue","yellow","violet")
sort.result <- sort(v)
print(sort.result)

# Sorting character vectors in reverse order.
revsort.result <- sort(v, decreasing = TRUE)
print(revsort.result)
```

When we execute the above code, it produces the following result −

```
[1]  -9   0   3   4   5   8  11 304
[1] 304  11   8   5   4   3   0  -9
[1] "Blue"   "Red"    "violet" "yellow"
[1] "yellow" "violet" "Red"    "Blue"
```

R - Matrices

Matrices are the R objects in which the elements are arranged in a two-dimensional rectangular layout. They contain elements of the same atomic types. Though we can create a matrix containing only characters or only logical values, they are not of much use. We use matrices containing numeric elements to be used in mathematical calculations.

A Matrix is created using the **matrix()** function.

Syntax

The basic syntax for creating a matrix in R is −

```
matrix(data, nrow, ncol, byrow, dimnames)
```

Following is the description of the parameters used −

- **data** is the input vector which becomes the data elements of the matrix.
- **nrow** is the number of rows to be created.
- **ncol** is the number of columns to be created.
- **byrow** is a logical clue. If TRUE then the input vector elements are arranged by row.
- **dimname** is the names assigned to the rows and columns.

Create a matrix taking a vector of numbers as input.

Live Demo

```
# Elements are arranged sequentially by row.
M <- matrix(c(3:14), nrow = 4, byrow = TRUE)
print(M)

# Elements are arranged sequentially by column.
N <- matrix(c(3:14), nrow = 4, byrow = FALSE)
print(N)

# Define the column and row names.
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")

P <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))
print(P)
```

When we execute the above code, it produces the following result −

```
     [,1] [,2] [,3]
[1,]    3    4    5
[2,]    6    7    8
[3,]    9   10   11
[4,]   12   13   14
     [,1] [,2] [,3]
[1,]    3    7   11
[2,]    4    8   12
[3,]    5    9   13
[4,]    6   10   14
     col1 col2 col3
row1    3    4    5
row2    6    7    8
row3    9   10   11
row4   12   13   14
```

Accessing Elements of a Matrix

Elements of a matrix can be accessed by using the column and row index of the element. We consider the matrix P above to find the specific elements below.

Live Demo

```
# Define the column and row names.
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")

# Create the matrix.
P <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))

# Access the element at 3rd column and 1st row.
print(P[1,3])

# Access the element at 2nd column and 4th row.
print(P[4,2])

# Access only the  2nd row.
print(P[2,])

# Access only the 3rd column.
print(P[,3])
```

When we execute the above code, it produces the following result −

```
[1] 5
[1] 13
col1 col2 col3
   6    7    8
row1 row2 row3 row4
   5    8   11   14
```

Matrix Computations

Various mathematical operations are performed on the matrices using the R operators. The result of the operation is also a matrix.

The dimensions (number of rows and columns) should be same for the matrices involved in the operation.

Matrix Addition & Subtraction

Live Demo

```
# Create two 2x3 matrices.
matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
print(matrix1)

matrix2 <- matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)
print(matrix2)

# Add the matrices.
result <- matrix1 + matrix2
cat("Result of addition","\n")
print(result)

# Subtract the matrices
result <- matrix1 - matrix2
cat("Result of subtraction","\n")
print(result)
```

When we execute the above code, it produces the following result −

```
     [,1] [,2] [,3]
[1,]    3   -1    2
[2,]    9    4    6
     [,1] [,2] [,3]
[1,]    5    0    3
[2,]    2    9    4
Result of addition
     [,1] [,2] [,3]
[1,]    8   -1    5
[2,]   11   13   10
Result of subtraction
     [,1] [,2] [,3]
[1,]   -2   -1   -1
[2,]    7   -5    2
```

Matrix Multiplication & Division

Live Demo

```
# Create two 2x3 matrices.
matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
print(matrix1)

matrix2 <- matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)
print(matrix2)

# Multiply the matrices.
result <- matrix1 * matrix2
cat("Result of multiplication","\n")
print(result)

# Divide the matrices
result <- matrix1 / matrix2
cat("Result of division","\n")
print(result)
```

When we execute the above code, it produces the following result −

```
     [,1] [,2] [,3]
[1,]    3   -1    2
[2,]    9    4    6
     [,1] [,2] [,3]
[1,]    5    0    3
[2,]    2    9    4
Result of multiplication
     [,1] [,2] [,3]
[1,]   15    0    6
[2,]   18   36   24
Result of division
     [,1]      [,2]      [,3]
[1,]  0.6      -Inf 0.6666667
[2,]  4.5 0.4444444 1.5000000
```

Essas operações são componente a componente e não correspondem as operações corretas da álgebra linear

```
> matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
> matrix1
     [,1] [,2] [,3]
[1,]    3   -1    2
[2,]    9    4    6
> matrix2 <- matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)
> print(matrix2)
     [,1] [,2] [,3]
[1,]    5    0    3
[2,]    2    9    4
> matrix1 %*% matrix2
Error in matrix1 %*% matrix2 : argumentos não compatíveis
> matrix3 <- t(matrix2)
> matrix3
     [,1] [,2]
[1,]    5    2
[2,]    0    9
[3,]    3    4
> matrix1 %*% matrix3
     [,1] [,2]
[1,]   21    5
[2,]   63   78
```

A multiplicação de matrizes só é definida se o número de colunas da matriz da esquerda é igual ao número de linhas da matriz da direita

matmult {base} R Documentation

Matrix Multiplication

Description

Multiplies two matrices, if they are conformable. If one argument is a vector, it will be promoted to either a row or column matrix to make the two arguments conformable. If both are vectors of the same length, it will return the inner product (as a matrix).

Usage

```
x %*% y
```

Arguments

x, y	numeric or complex matrices or vectors.

A “Divisão” de matrizes equivale a multiplicação pela matriz inversa

```
> A <- matrix( c(5, 1, 0,
+                3,-1, 2,
+                4, 0,-1), nrow=3, byrow=TRUE)
> A
     [,1] [,2] [,3]
[1,]    5    1    0
[2,]    3   -1    2
[3,]    4    0   -1
> AI <- ginv(A)
> AI
       [,1]    [,2]   [,3]
[1,] 0.0625  0.0625  0.125
[2,] 0.6875 -0.3125 -0.625
[3,] 0.2500  0.2500 -0.500
> AI %*% A
              [,1]          [,2]          [,3]
[1,]  1.000000e+00  3.469447e-17 -2.775558e-17
[2,] -8.881784e-16  1.000000e+00  7.771561e-16
[3,]  0.000000e+00 -1.498801e-15  1.000000e+00
```

Perceba que os valores fora da diagonal são pequenos

```
> zapsmall(AI %*% A)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

https://cran.r-project.org/web/packages/matlib/vignettes/inv-ex1.html

R - Lists

Lists are the R objects which contain elements of different types like − numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. List is created using **list()** function.

Creating a List

Following is an example to create a list containing strings, numbers, vectors and a logical values.

Live Demo

```
# Create a list containing strings, numbers, vectors and a logical
# values.
list_data <- list("Red", "Green", c(21,32,11), TRUE, 51.23, 119.1)
print(list_data)
```

When we execute the above code, it produces the following result −

```
[[1]]
[1] "Red"

[[2]]
[1] "Green"

[[3]]
[1] 21 32 11

[[4]]
[1] TRUE

[[5]]
[1] 51.23

[[6]]
[1] 119.1
```

Naming List Elements

The list elements can be given names and they can be accessed using these names.

Live Demo

```
# Create a list containing a vector, a matrix and a list.
list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
   list("green",12.3))

# Give names to the elements in the list.
names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")

# Show the list.
print(list_data)
```

When we execute the above code, it produces the following result −

```
$`1st_Quarter`
[1] "Jan" "Feb" "Mar"

$A_Matrix
     [,1] [,2] [,3]
[1,]    3    5   -2
[2,]    9    1    8

$A_Inner_list
$A_Inner_list[[1]]
[1] "green"

$A_Inner_list[[2]]
[1] 12.3
```

Accessing List Elements

Elements of the list can be accessed by the index of the element in the list. In case of named lists it can also be accessed using the names.

We continue to use the list in the above example −

Live Demo

```
# Create a list containing a vector, a matrix and a list.
list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
   list("green",12.3))

# Give names to the elements in the list.
names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")

# Access the first element of the list.
print(list_data[1])

# Access the thrid element. As it is also a list, all its elements will be printed.
print(list_data[3])

# Access the list element using the name of the element.
print(list_data$A_Matrix)
```

When we execute the above code, it produces the following result −

```
$`1st_Quarter`
[1] "Jan" "Feb" "Mar"

$A_Inner_list
$A_Inner_list[[1]]
[1] "green"

$A_Inner_list[[2]]
[1] 12.3

     [,1] [,2] [,3]
[1,]    3    5   -2
[2,]    9    1    8
```

Console | Terminal | Jobs

D:/OneDrive/My_OneDrive/USP/2020/MAP2112/scripts/

```
> list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
+                   list("green",12.3))
> names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")
> list_data[1]
$`1st Quarter`
[1] "Jan" "Feb" "Mar"

> list_data[2]
$A_Matrix
     [,1] [,2] [,3]
[1,]    3    5   -2
[2,]    9    1    8

> list_data[3]
$`A Inner list`
$`A Inner list`[[1]]
[1] "green"

$`A Inner list`[[2]]
[1] 12.3


> list_data[[3]][[1]]
[1] "green"
> list_data[[3]][[2]]
[1] 12.3
> list_data$`A Inner list`
[[1]]
[1] "green"

[[2]]
[1] 12.3

> list_data$`A Inner list`[1]
[[1]]
[1] "green"

> list_data$`A Inner list`[2]
[[1]]
[1] 12.3
```

Manipulating List Elements

We can add, delete and update list elements as shown below. We can add and delete elements only at the end of a list. But we can update any element.

Live Demo

```
# Create a list containing a vector, a matrix and a list.
list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
   list("green",12.3))

# Give names to the elements in the list.
names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")

# Add element at the end of the list.
list_data[4] <- "New element"
print(list_data[4])

# Remove the last element.
list_data[4] <- NULL

# Print the 4th Element.
print(list_data[4])

# Update the 3rd Element.
list_data[3] <- "updated element"
print(list_data[3])
```

When we execute the above code, it produces the following result −

```
[[1]]
[1] "New element"

$<NA>
NULL

$`A Inner list`
[1] "updated element"
```

Merging Lists

You can merge many lists into one list by placing all the lists inside one list() function.

Live Demo

```
# Create two lists.
list1 <- list(1,2,3)
list2 <- list("Sun","Mon","Tue")

# Merge the two lists.
merged.list <- c(list1,list2)

# Print the merged list.
print(merged.list)
```

When we execute the above code, it produces the following result −

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

[[2]]
[1] 2

[[3]]
[1] 3

[[4]]
[1] "Sun"

[[5]]
[1] "Mon"

[[6]]
[1] "Tue"
```

Converting List to Vector

A list can be converted to a vector so that the elements of the vector can be used for further manipulation. All the arithmetic operations on vectors can be applied after the list is converted into vectors. To do this conversion, we use the **unlist()** function. It takes the list as input and produces a vector.

Live Demo

```
# Create lists.
list1 <- list(1:5)
print(list1)

list2 <-list(10:14)
print(list2)

# Convert the lists to vectors.
v1 <- unlist(list1)
v2 <- unlist(list2)

print(v1)
print(v2)

# Now add the vectors
result <- v1+v2
print(result)
```

When we execute the above code, it produces the following result −

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

[[1]]
[1] 10 11 12 13 14

[1] 1 2 3 4 5
[1] 10 11 12 13 14
[1] 11 13 15 17 19
```

R - Data Frames

A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column.

Following are the characteristics of a data frame.

- The column names should be non-empty.
- The row names should be unique.
- The data stored in a data frame can be of numeric, factor or character type.
- Each column should contain same number of data items.

Create Data Frame

Live Demo

```
# Create the data frame.
emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),
   stringsAsFactors = FALSE
)
# Print the data frame.
print(emp.data)
```

When we execute the above code, it produces the following result −

```
  emp_id    emp_name     salary     start_date
1      1     Rick         623.30     2012-01-01
2      2     Dan          515.20     2013-09-23
3      3     Michelle     611.00     2014-11-15
4      4     Ryan         729.00     2014-05-11
5      5     Gary         843.25     2015-03-27
```

Live Demo

```
# Create the data frame.
emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),
   stringsAsFactors = FALSE
)
# Get the structure of the data frame.
str(emp.data)
```

When we execute the above code, it produces the following result −

```
'data.frame':   5 obs. of  4 variables:
 $ emp_id    : int  1 2 3 4 5
 $ emp_name  : chr  "Rick" "Dan" "Michelle" "Ryan" ...
 $ salary    : num  623 515 611 729 843
 $ start_date: Date, format: "2012-01-01" "2013-09-23" "2014-11-15" "2014-05-11" ...
```

Summary of Data in Data Frame

The statistical summary and nature of the data can be obtained by applying **summary()** function.

Live Demo

```
# Create the data frame.
emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),
   stringsAsFactors = FALSE
)
# Print the summary.
print(summary(emp.data))
```

When we execute the above code, it produces the following result −

```
     emp_id    emp_name            salary        start_date
 Min.   :1   Length:5           Min.   :515.2   Min.   :2012-01-01
 1st Qu.:2   Class :character   1st Qu.:611.0   1st Qu.:2013-09-23
 Median :3   Mode  :character   Median :623.3   Median :2014-05-11
 Mean   :3                      Mean   :664.4   Mean   :2014-01-14
 3rd Qu.:4                      3rd Qu.:729.0   3rd Qu.:2014-11-15
 Max.   :5                      Max.   :843.2   Max.   :2015-03-27
```

Extract Data from Data Frame

Extract specific column from a data frame using column name.

Live Demo

```
# Create the data frame.
emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01","2013-09-23","2014-11-15","2014-05-11",
      "2015-03-27")),
   stringsAsFactors = FALSE
)
# Extract Specific columns.
result <- data.frame(emp.data$emp_name,emp.data$salary)
print(result)
```

When we execute the above code, it produces the following result −

```
  emp.data.emp_name emp.data.salary
1              Rick          623.30
2               Dan          515.20
3          Michelle          611.00
4              Ryan          729.00
5              Gary          843.25
```

Extract the first two rows and then all columns

Live Demo

```
# Create the data frame.
emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),
   stringsAsFactors = FALSE
)
# Extract first two rows.
result <- emp.data[1:2,]
print(result)
```

When we execute the above code, it produces the following result −

```
  emp_id    emp_name   salary    start_date
1      1     Rick       623.3     2012-01-01
2      2     Dan        515.2     2013-09-23
```

Extract 3rd and 5th row with 2nd and 4th column

Live Demo

```
# Create the data frame.
emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),

        start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),
   stringsAsFactors = FALSE
)

# Extract 3rd and 5th row with 2nd and 4th column.
result <- emp.data[c(3,5),c(2,4)]
print(result)
```

When we execute the above code, it produces the following result −

```
  emp_name start_date
3 Michelle 2014-11-15
5     Gary 2015-03-27
```

Expand Data Frame

A data frame can be expanded by adding columns and rows.

Add Column

Just add the column vector using a new column name.

Live Demo

```
# Create the data frame.
emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),
   stringsAsFactors = FALSE
)

# Add the "dept" coulmn.
emp.data$dept <- c("IT","Operations","IT","HR","Finance")
v <- emp.data
print(v)
```

When we execute the above code, it produces the following result −

```
  emp_id   emp_name    salary    start_date       dept
1      1    Rick        623.30    2012-01-01       IT
2      2    Dan         515.20    2013-09-23       Operations
3      3    Michelle    611.00    2014-11-15       IT
4      4    Ryan        729.00    2014-05-11       HR
5      5    Gary        843.25    2015-03-27       Finance
```

Add Row

To add more rows permanently to an existing data frame, we need to bring in the new rows in the same structure as the existing data frame and use the **rbind()** function.

Live Demo

```
# Create the first data frame.
emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),
   dept = c("IT","Operations","IT","HR","Finance"),
   stringsAsFactors = FALSE
)

# Create the second data frame
emp.newdata <-  data.frame(
   emp_id = c (6:8),
   emp_name = c("Rasmi","Pranab","Tusar"),
   salary = c(578.0,722.5,632.8),
   start_date = as.Date(c("2013-05-21","2013-07-30","2014-06-17")),
   dept = c("IT","Operations","Fianance"),
   stringsAsFactors = FALSE
)

# Bind the two data frames.
emp.finaldata <- rbind(emp.data,emp.newdata)
print(emp.finaldata)
```

When we execute the above code, it produces the following result −

```
  emp_id     emp_name    salary     start_date       dept
1      1     Rick        623.30     2012-01-01       IT
2      2     Dan         515.20     2013-09-23       Operations
3      3     Michelle    611.00     2014-11-15       IT
4      4     Ryan        729.00     2014-05-11       HR
5      5     Gary        843.25     2015-03-27       Finance
6      6     Rasmi       578.00     2013-05-21       IT
7      7     Pranab      722.50     2013-07-30       Operations
8      8     Tusar       632.80     2014-06-17       Fianance
```

R - Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. R language is rich in built-in operators and provides following types of operators.

Types of Operators

We have the following types of operators in R programming −

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Miscellaneous Operators

Arithmetic Operators

Following table shows the arithmetic operators supported by R language. The operators act on each element of the vector.

Operator	Description	Example

+	Adds two vectors	Live Demo
		`v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`
		`print(v+t)`
		it produces the following result −
		`[1] 10.0  8.5  10.0`

−	Subtracts second vector from the first	Live Demo
		`v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`
		`print(v-t)`
		it produces the following result −
		`[1] -6.0  2.5  2.0`

*	Multiplies both vectors	Live Demo
		`v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`
		`print(v*t)`
		it produces the following result −
		`[1] 16.0 16.5 24.0`

Arithmetic Operators

Following table shows the arithmetic operators supported by R language. The operators act on each element of the vector.

Operator	Description	Example

/	Divide the first vector with the second	Live Demo
		`v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`
		`print(v/t)`
		When we execute the above code, it produces the following result −
		`[1] 0.250000 1.833333 1.500000`

%%	Give the remainder of the first vector with the second	`> v <- c(3, 4,5)`
		`> t <- c(2, 2, 2)`
		`> print(v%%t)`
		`[1] 1 0 1`
		`>`

%/%	The result of division of first vector with second (quotient)	`> v <- c(3, 4,5)`
		`> t <- c(2, 2, 2)`
		`> print(v%/%t)`
		`[1] 1 2 2`
		`>`

Arithmetic Operators

Following table shows the arithmetic operators supported by R language. The operators act on each element of the vector.

Operator	Description	Example

^	The first vector raised to the exponent of second vector	Live Demo
		`v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`
		`print(v^t)`
		it produces the following result −
		`[1]  256.000  166.375 1296.000`

Relational Operators

Following table shows the relational operators supported by R language. Each element of the first vector is compared with the corresponding element of the second vector. The result of comparison is a Boolean value.

Operator	Description	Example

>	Checks if each element of the first vector is greater than the corresponding element of the second vector.	Live Demo
		`v <- c(2,5.5,6,9)`
		`t <- c(8,2.5,14,9)`
		`print(v>t)`
		it produces the following result –
		`[1] FALSE TRUE FALSE FALSE`

<	Checks if each element of the first vector is less than the corresponding element of the second vector.	Live Demo
		`v <- c(2,5.5,6,9)`
		`t <- c(8,2.5,14,9)`
		`print(v < t)`
		it produces the following result –
		`[1] TRUE FALSE TRUE FALSE`

==	Checks if each element of the first vector is equal to the corresponding element of the second vector.	Live Demo
		`v <- c(2,5.5,6,9)`
		`t <- c(8,2.5,14,9)`
		`print(v == t)`
		it produces the following result –
		`[1] FALSE FALSE FALSE TRUE`

Relational Operators

Following table shows the relational operators supported by R language. Each element of the first vector is compared with the corresponding element of the second vector. The result of comparison is a Boolean value.

Operator	Description	Example

<=	Checks if each element of the first vector is less than or equal to the corresponding element of the second vector.	Live Demo
		`v <- c(2,5.5,6,9)`
		`t <- c(8,2.5,14,9)`
		`print(v<=t)`
		it produces the following result −
		`[1] TRUE FALSE TRUE TRUE`

>=	Checks if each element of the first vector is greater than or equal to the corresponding element of the second vector.	Live Demo
		`v <- c(2,5.5,6,9)`
		`t <- c(8,2.5,14,9)`
		`print(v>=t)`
		it produces the following result −
		`[1] FALSE TRUE FALSE TRUE`

!=	Checks if each element of the first vector is unequal to the corresponding element of the second vector.	Live Demo
		`v <- c(2,5.5,6,9)`
		`t <- c(8,2.5,14,9)`
		`print(v!=t)`
		it produces the following result −
		`[1] TRUE TRUE TRUE FALSE`

Logical Operators

Following table shows the logical operators supported by R language. It is applicable only to vectors of type logical, numeric or complex. All numbers greater than 1 are considered as logical value TRUE.

Each element of the first vector is compared with the corresponding element of the second vector. The result of comparison is a Boolean value.

Operator	Description	Example

&	It is called Element-wise Logical AND operator. It combines each element of the first vector with the corresponding element of the second vector and gives a output TRUE if both the elements are TRUE.	Live Demo
		`v <- c(3,1,TRUE,2+3i)`
		`t <- c(4,1,FALSE,2+3i)`
		`print(v&t)`
		it produces the following result −
		`[1]  TRUE  TRUE FALSE  TRUE`

|	It is called Element-wise Logical OR operator. It combines each element of the first vector with the corresponding element of the second vector and gives a output TRUE if one the elements is TRUE.	Live Demo
		`v <- c(3,0,TRUE,2+2i)`
		`t <- c(4,0,FALSE,2+3i)`
		`print(v|t)`
		it produces the following result −
		`[1]  TRUE FALSE  TRUE  TRUE`

!	It is called Logical NOT operator. Takes each element of the vector and gives the opposite logical value.	Live Demo
		`v <- c(3,0,TRUE,2+2i)`
		`print(!v)`
		it produces the following result −
		`[1] FALSE  TRUE FALSE FALSE`

The logical operator && and || considers only the first element of the vectors and give a vector of single element as output.

Operator	Description	Example

&&	Called Logical AND operator. Takes first element of both the vectors and gives the TRUE only if both are TRUE.	Live Demo
		`v <- c(3,0,TRUE,2+2i)`
		`t <- c(1,3,TRUE,2+3i)`
		`print(v&&t)`
		it produces the following result –
		`[1] TRUE`

||	Called Logical OR operator. Takes first element of both the vectors and gives the TRUE if one of them is TRUE.	Live Demo
		`v <- c(0,0,TRUE,2+2i)`
		`t <- c(0,3,TRUE,2+3i)`
		`print(v||t)`
		it produces the following result –
		`[1] FALSE`

Uso previsto: a) se a primeira posição tem um significado específico não precisa checar as outras b) se os vetores tiverem dimensão diferente apenas a primeira é verificada

Assignment Operators

These operators are used to assign values to vectors.

Operator	Description	Example

<–
or
=
or
<<–
	Called Left Assignment	Live Demo
		`v1 <- c(3,1,TRUE,2+3i)`
		`v2 <<- c(3,1,TRUE,2+3i)`
		`v3 = c(3,1,TRUE,2+3i)`
		`print(v1)`
		`print(v2)`
		`print(v3)`
		it produces the following result −
		`[1] 3+0i 1+0i 1+0i 2+3i`
		`[1] 3+0i 1+0i 1+0i 2+3i`
		`[1] 3+0i 1+0i 1+0i 2+3i`

–>
or
–>>
	Called Right Assignment	Live Demo
		`c(3,1,TRUE,2+3i) -> v1`
		`c(3,1,TRUE,2+3i) ->> v2`
		`print(v1)`
		`print(v2)`
		it produces the following result −
		`[1] 3+0i 1+0i 1+0i 2+3i`
		`[1] 3+0i 1+0i 1+0i 2+3i`

Miscellaneous Operators

These operators are used to for specific purpose and not general mathematical or logical computation.

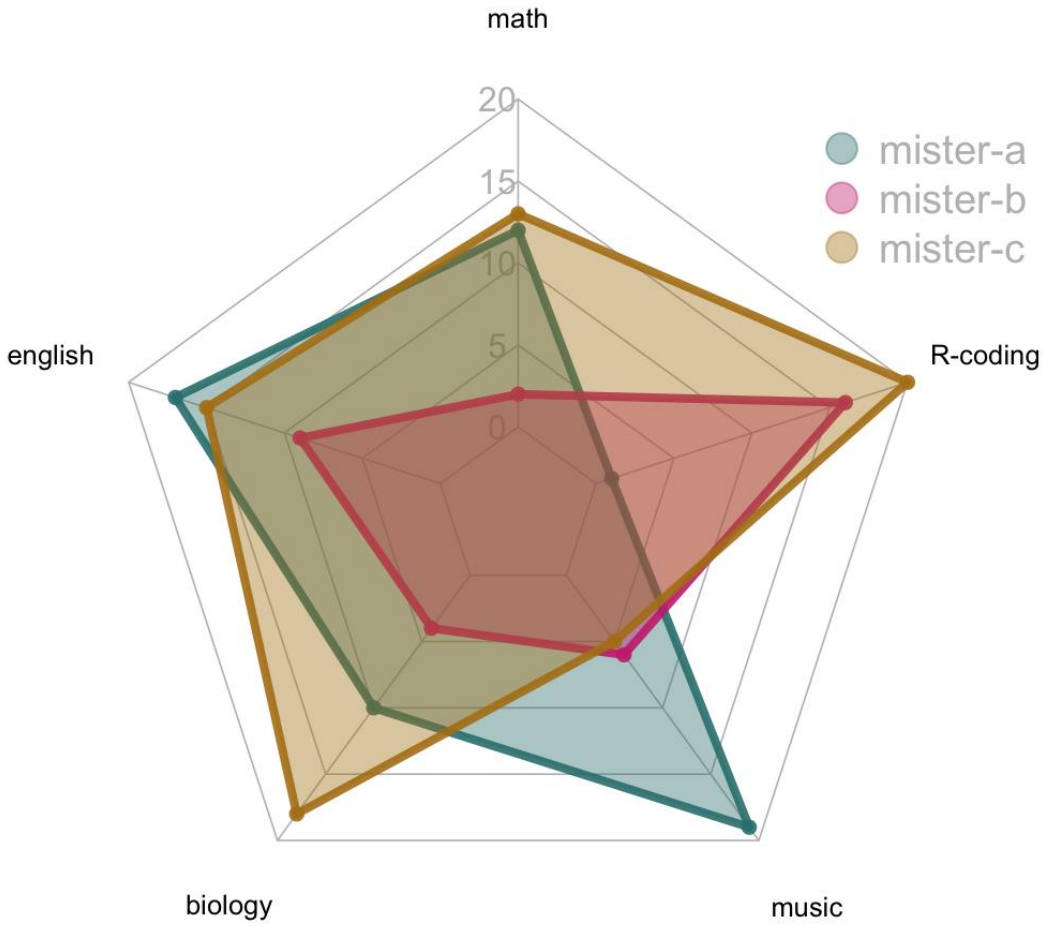
Operator	Description	Example

:	Colon operator. It creates the series of numbers in sequence for a vector.	Live Demo
		`v <- 2:8`
		`print(v)`
		it produces the following result –
		`[1] 2 3 4 5 6 7 8`

%in%	This operator is used to identify if an element belongs to a vector.	Live Demo
		`v1 <- 8`
		`v2 <- 12`
		`t <- 1:10`
		`print(v1 %in% t)`
		`print(v2 %in% t)`
		it produces the following result –
		`[1] TRUE`
		`[1] FALSE`

%*%	Representa a Multiplicação de matrizes conforme a definição	Live Demo
		`M = matrix( c(2,6,5,1,10,4), nrow = 2,ncol = 3,byrow = TRUE)`
		`t = M %*% t(M)`
		`print(t)`
		it produces the following result –
		`[,1] [,2]`
		`[1,] 65 82`
		`[2,] 82 117`

?



https://www.r-graph-gallery.com/143-spider-chart-with-saveral-individuals.html