SAKARYA ÜNİVERSİTESİ

BIG DATA

TOO BIG TO IGNORE

SÜMEYYE KAYNAK

OUTLINE

R programming language

R PROGRAMMING FOR DATA SCIENCE

- The R programming language has become the de facto programming language for data science by its flexibility, power, sophistication, and expressiveness.
- R runs on almost any standard computing platform and operating system.
- It is open-source
- Versions are released frequently.
- R has sophisticated graphics capabilities.

R PROGRAMMING FOR DATA SCIENCE

- R is that platform and thousands of people around the world have come together to make contributions to R, to develop packages, and help each other use R for all kinds of applications.

DESIGN OF THE R SYSTEM

- The primary R system is available from the Comprehensive R Archive Network, also known as CRAN. CRAN also hosts many add-on packages that can be used to extend the functionality of R.
- The R system is divided into 2 conceptual parts:
  - The “base” R system that you download from CRAN
  - Everything else

DESIGN OF THE R SYSTEM

- R functionality is divided into a number of *packages*.
  - The base package which is required to run R and contains the most fundamental functions.
  - The other packages contained in the "base" system include utils, stats, datasets, graphics, grDevices, grid, methods, tools, parallel, compiler, splines, tcltk, stats4.
  - There are also "Recommended" packages: boot, class, cluster, codetools, foreign, KernSmooth, lattice, mgcv, nlme, rpart, survival, MASS, spatial, nnet, Matrix.

EVOLUTION OF R

- R was initially written by Ross Ihaka and Robert Gentleman at the Department of Statistics of the University of Auckland in Auckland, New Zealand. R made its first appearance in 1993.
- A large group of individuals has contributed to R by sending code and bug reports.
- Since mid-1997 there has been a core group (the "R Core Team") who can modify the R source code archive.

FEATURES OF R

- R is a well-developed, simple and effective programming language which includes conditionals, loops, user defined recursive functions and input and output facilities.
- R has an effective data handling and storage facility,
- R provides a suite of operators for calculations on arrays, lists, vectors and matrices.
- R provides a large, coherent and integrated collection of tools for data analysis.
- R provides graphical facilities for data analysis and display either directly at the computer or printing at the papers.

LIMITATIONS OF R

- Objects in R must generally be stored in physical memory.
- Its functionality is based on consumer demand and (voluntary) user contributions. If no one feels like implementing your favorite method, then it’s your job to implement it.
- It lacks any consistency.
- Horrendous error messages.
- Poorly written help files (they seem designed for experts.)
- General attitude is "expert friendly“.
- The burden of avoiding errors falls entirely on the user, and this makes development a lot slow

LIMITATIONS OF R

- Lack of efficient data structures like Hash Tables, Maps and Sets
- Slow for loops.
- Very slow concatenation operations like cbind, rbind etc.
- There are multiple ways to do the same thing which can be confusing for the new user as to which is more efficient (or not).
- a very badly cluttered humungous documentation - reliance on fragmented documentation across the web including blogs.

R- PYTHON

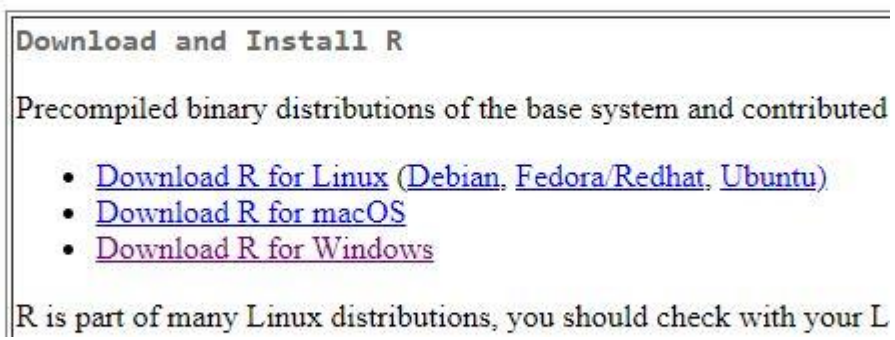
- Python, “Bir şeyi yapmanın tek ve tercihen tek bir yolu olmalı” felsefesi üzerine tasarlanmıştır. Bu nedenle bir görevi yerine getirmek için birkaç ana paketi vardır. R ise aynı görevi gerçekleştirmek için yüzlerce pakete sahiptir.
- R karmaşık matematiksel hesaplamaları ve istatistiksel testleri kullanmayı kolaylaştırır. Python ise sıfırdan yeni bir şey inşa etmek, uygulama geliştirmek için kullanılır.
- R ile başlamak kolaydır çünkü daha basit kütüphanelere ve plot’lara sahiptir. Fakat, Python kütüphanelerini öğrenmek biraz karmaşık olabilir.
- Popülerlik bakımından Python, R’a göre daha popülerdir.
- R veri görselleştirilmesinde, Python ise derin öğrenme alanında daha iyidir.

INSTALLATION

- Installation for Windows OS

https://www.youtube.com/watch?v=Ohnk9hcxf9M

- Enter https://cran.r-project.org/ url

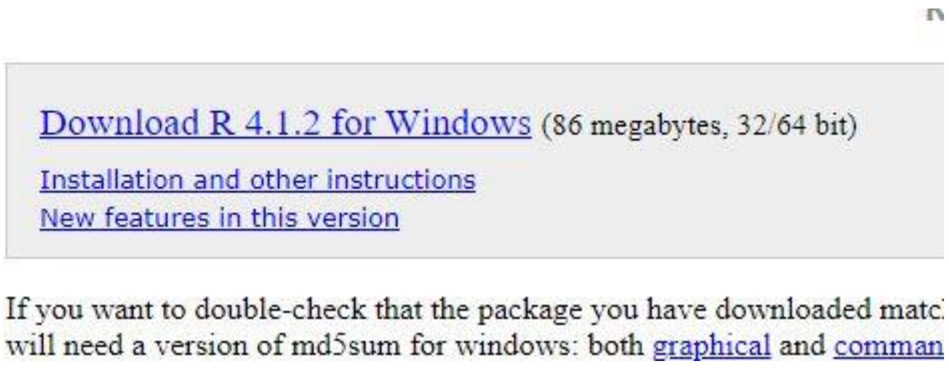


Download and Install R

Precompiled binary distributions of the base system and contributed

- Download R for Linux (Debian, Fedora/Redhat, Ubuntu)
- Download R for macOS
- Download R for Windows

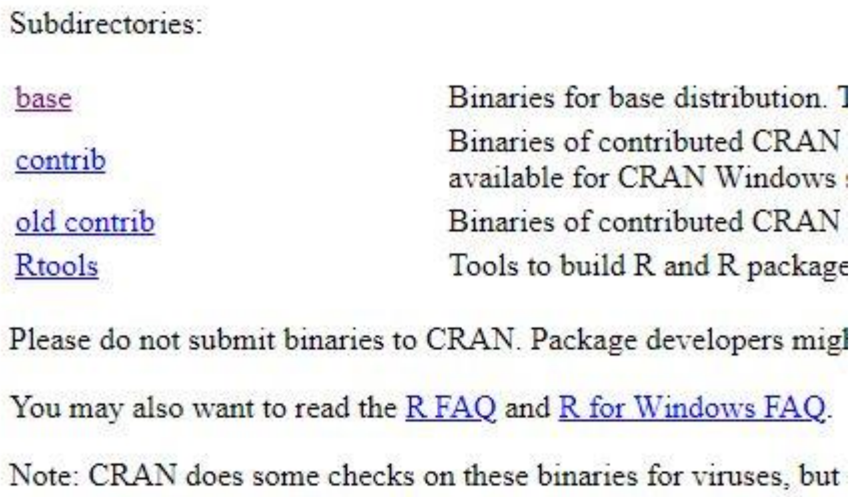
R is part of many Linux distributions, you should check with your L



Download R 4.1.2 for Windows (86 megabytes, 32/64 bit)

Installation and other instructions
New features in this version

If you want to double-check that the package you have downloaded matc
will need a version of md5sum for windows: both graphical and comman



Subdirectories:

base	Binaries for base distribution. T
contrib	Binaries of contributed CRAN available for CRAN Windows :
old contrib	Binaries of contributed CRAN
Rtools	Tools to build R and R package

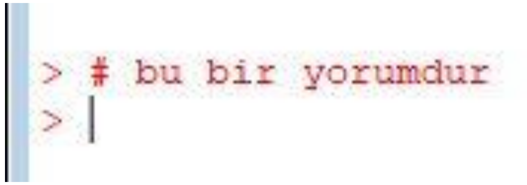
Please do not submit binaries to CRAN. Package developers migl

You may also want to read the R FAQ and R for Windows FAQ.

Note: CRAN does some checks on these binaries for viruses, but

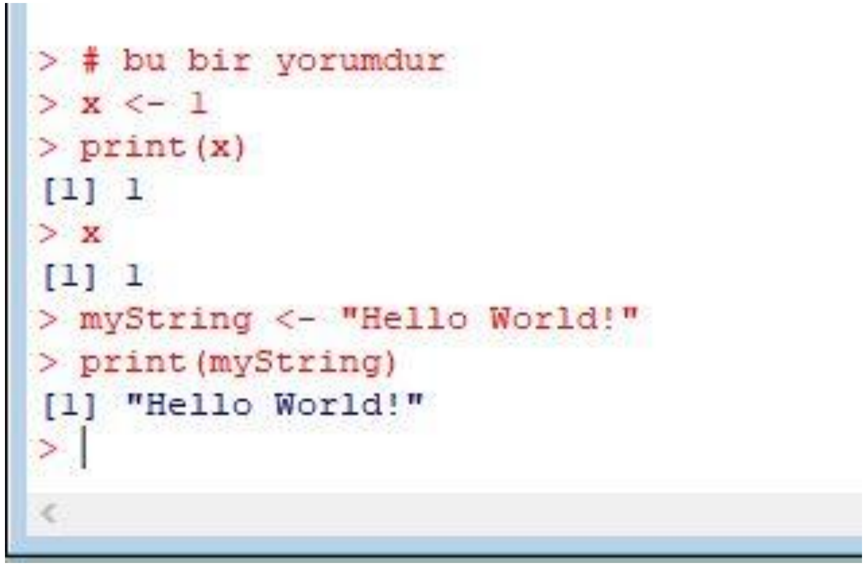
BASIC SYNTAX

- Command prompt



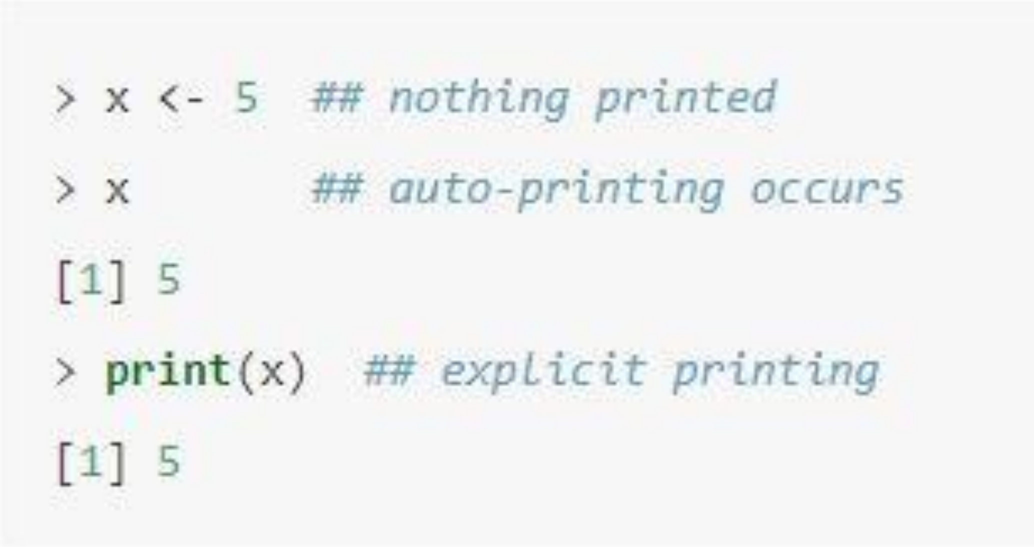
```
> # bu bir yorumdur
> |
```

- The <- symbol is the assignment operator.

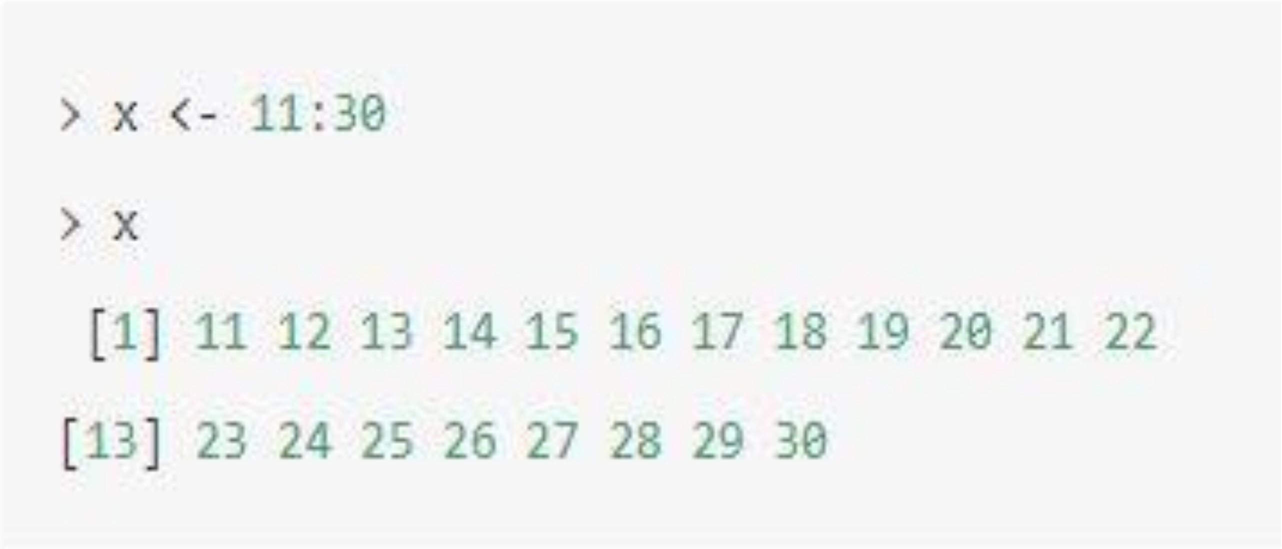


```
> # bu bir yorumdur
> x <- 1
> print(x)
[1] 1
> x
[1] 1
> myString <- "Hello World!"
> print(myString)
[1] "Hello World!"
> |
```

BASIC SYNTAX



```
> x <- 5   ## nothing printed
> x        ## auto-printing occurs
[1] 5
> print(x)  ## explicit printing
[1] 5
```



```
> x <- 11:30
> x
 [1] 11 12 13 14 15 16 17 18 19 20 21 22
[13] 23 24 25 26 27 28 29 30
```

R OBJECTS

R has five basic or “atomic” classes of objects:

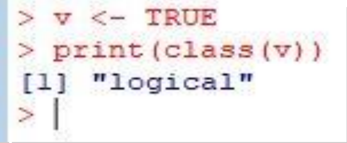
- character
- numeric (real numbers)
- integer
- complex
- logical (True/False)
- Raw

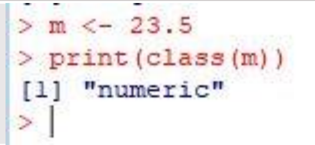
There are many types of R-objects. The frequently used ones are –

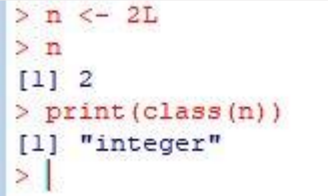
- Vectors
- Lists
- Matrices
- Arrays
- Factors
- Data Frames

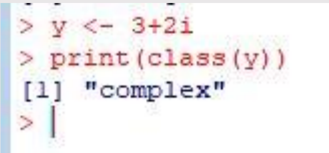
DATA TYPES

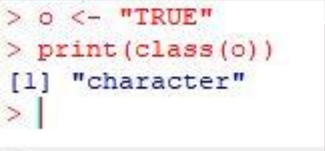
Data type	Example	Verify

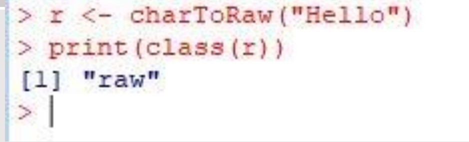
Logical	TRUE, FALSE	
		`> v <- TRUE`
		`> print(class(v))`
		`[1] "logical"`
		`>`

Numeric	12.3, 5, 999	
		`> m <- 23.5`
		`> print(class(m))`
		`[1] "numeric"`
		`>`

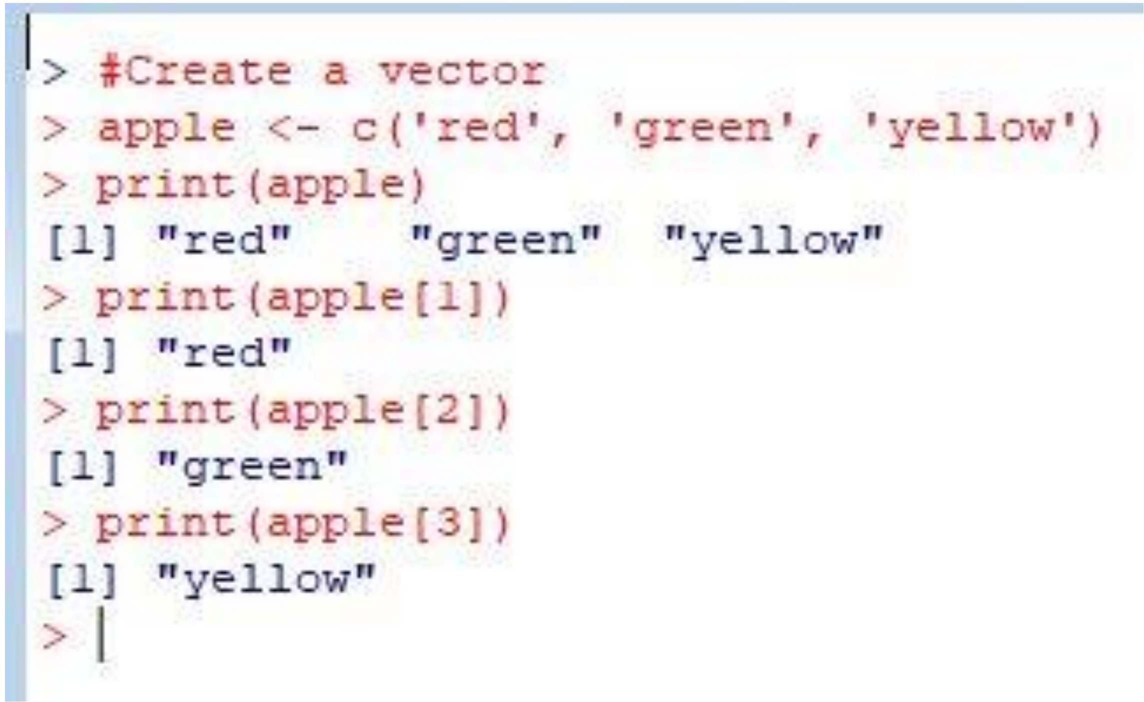
Integer	2L, 34L, 0L	
		`> n <- 2L`
		`> n`
		`[1] 2`
		`> print(class(n))`
		`[1] "integer"`
		`>`

Complex	3+2i	
		`> y <- 3+2i`
		`> print(class(y))`
		`[1] "complex"`
		`>`

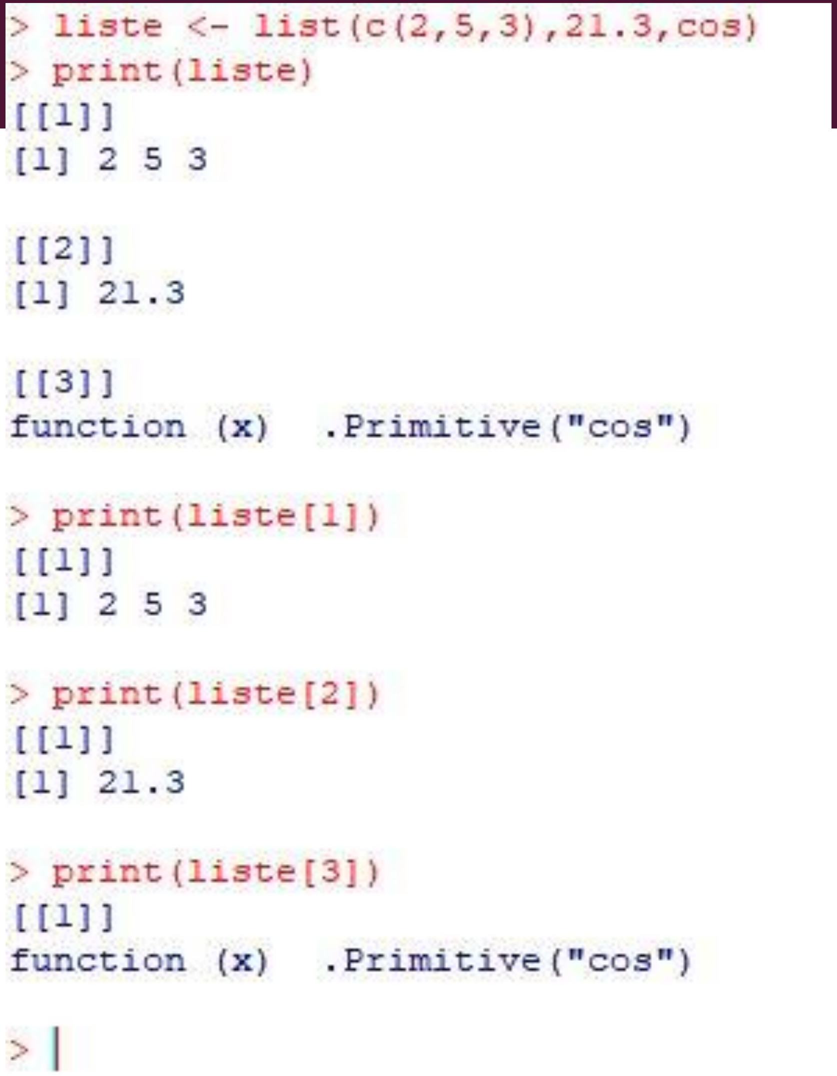
Character	‘a’, “good”, “TRUE”, ‘23.5’	
		`> o <- "TRUE"`
		`> print(class(o))`
		`[1] "character"`
		`>`

Raw	“hello” is stored as 47	
		`> r <- charToRaw("Hello")`
		`> print(class(r))`
		`[1] "raw"`
		`>`

VECTORS - LISTS

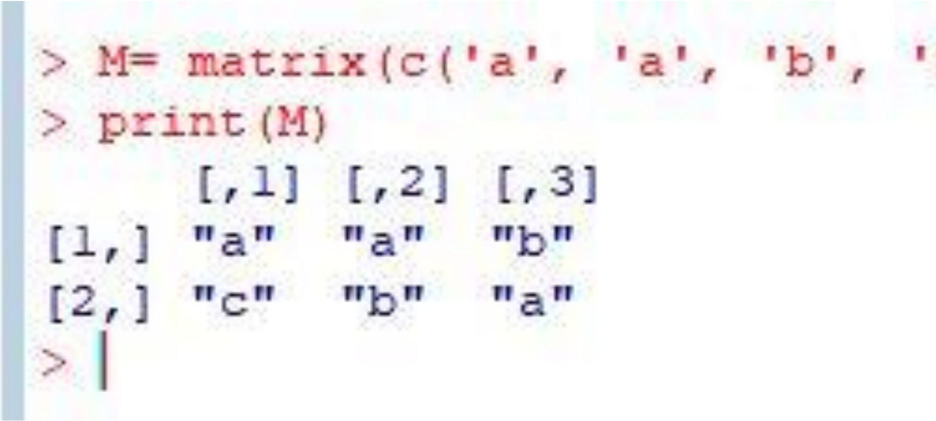


```
> #Create a vector
> apple <- c('red', 'green', 'yellow')
> print(apple)
[1] "red"    "green"  "yellow"
> print(apple[1])
[1] "red"
> print(apple[2])
[1] "green"
> print(apple[3])
[1] "yellow"
> 
```

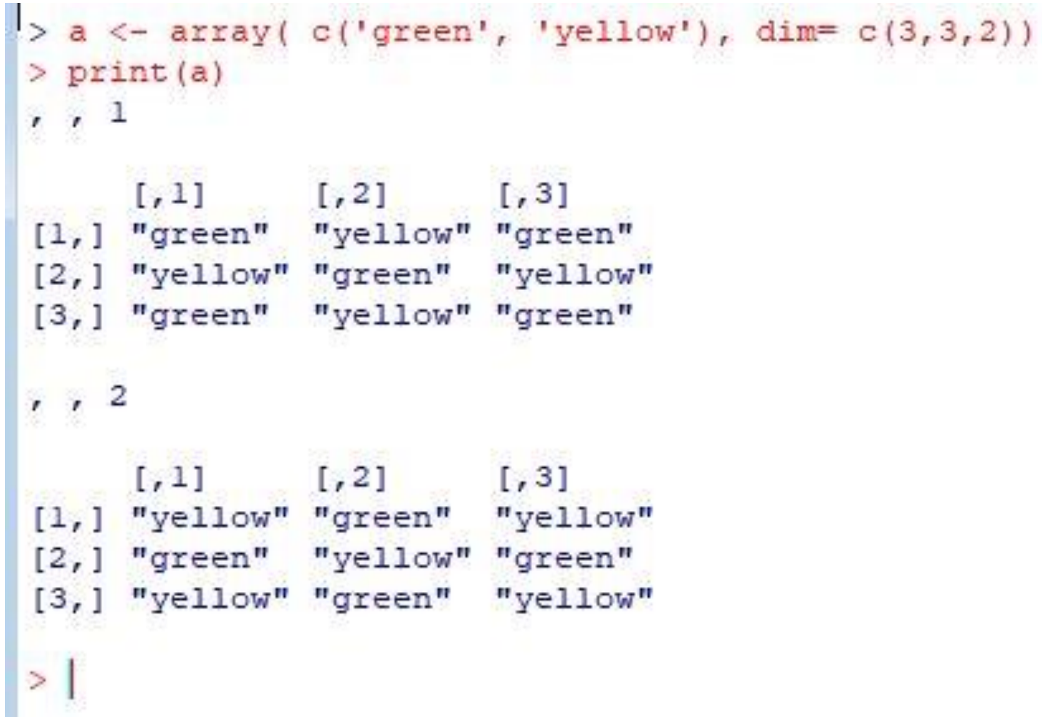


```
> liste <- list(c(2,5,3),21.3,cos)
> print(liste)
[[1]]
[1] 2 5 3

[[2]]
[1] 21.3

[[3]]
function (x)  .Primitive("cos")

> print(liste[1])
[[1]]
[1] 2 5 3

> print(liste[2])
[[1]]
[1] 21.3

> print(liste[3])
[[1]]
function (x)  .Primitive("cos")

> 
```

MATRICES- ARRAYS

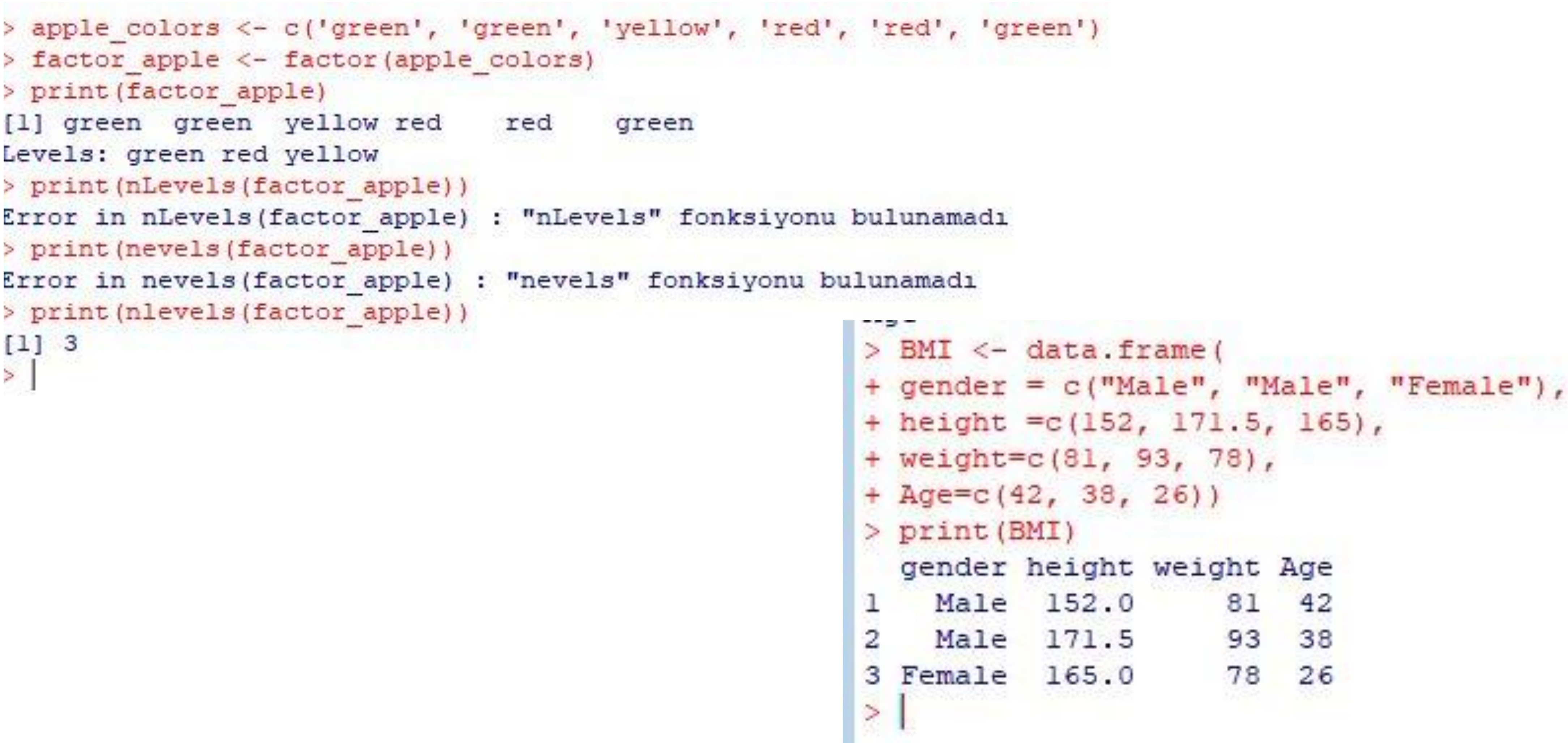


```
> M= matrix(c('a', 'a', 'b', 'c', 'b', 'a'), nrow=2, ncol=3, byrow= TRUE)
> print(M)
     [,1] [,2] [,3]
[1,] "a"  "a"  "b" 
[2,] "c"  "b"  "a" 
> 
```



```
> a <- array( c('green', 'yellow'), dim= c(3,3,2))
> print(a)
, , 1

     [,1]     [,2]     [,3]    
[1,] "green"  "yellow" "green" 
[2,] "yellow" "green"  "yellow"
[3,] "green"  "yellow" "green" 

, , 2

     [,1]     [,2]     [,3]    
[1,] "yellow" "green"  "yellow"
[2,] "green"  "yellow" "green" 
[3,] "yellow" "green"  "yellow"

> 
```

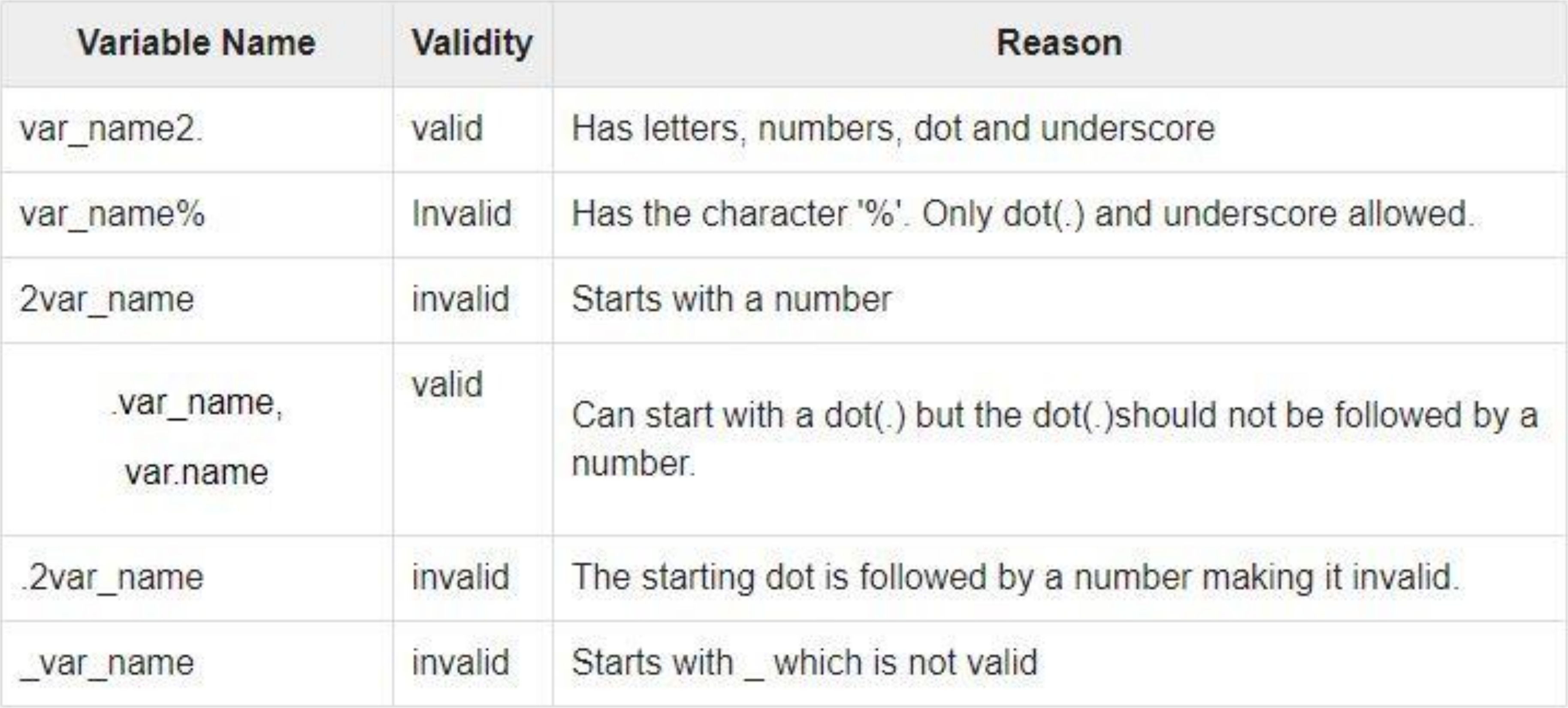
FACTORS-DATA FRAMES



```
> apple_colors <- c('green', 'green', 'yellow', 'red', 'red', 'green')
> factor_apple <- factor(apple_colors)
> print(factor_apple)
[1] green  green  yellow red    red    green
Levels: green red yellow
> print(nLevels(factor_apple))
Error in nLevels(factor_apple) : "nLevels" fonksiyonu bulunamadı
> print(nevels(factor_apple))
Error in nevels(factor_apple) : "nevels" fonksiyonu bulunamadı
> print(nlevels(factor_apple))
[1] 3
> 
```

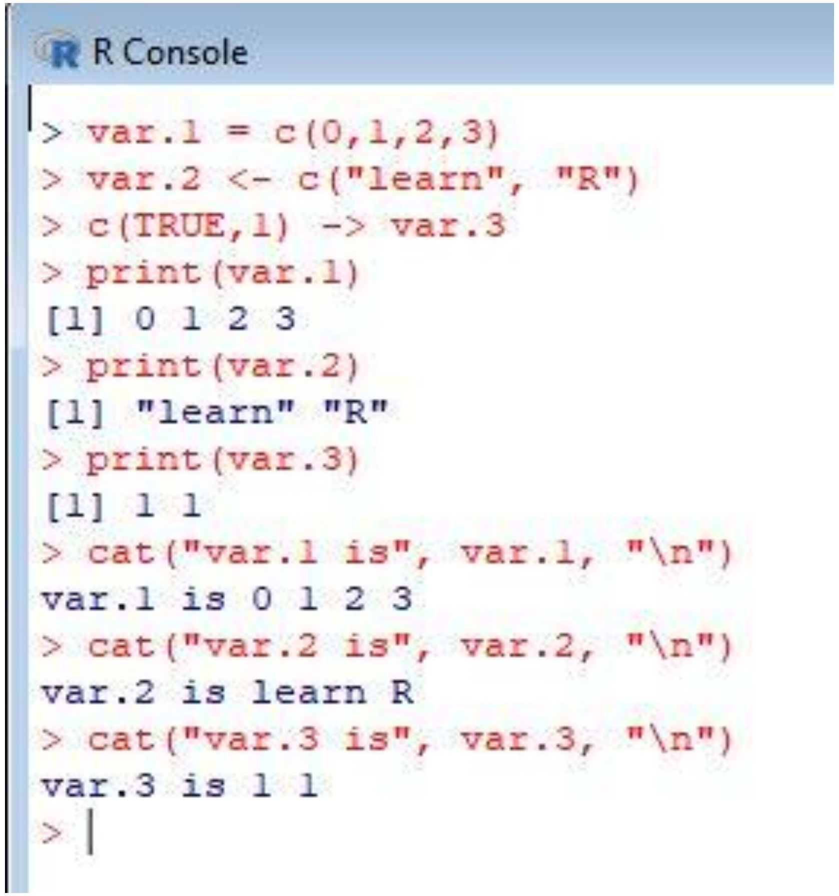
```
> BMI <- data.frame(
+ gender = c("Male", "Male", "Female"),
+ height =c(152, 171.5, 165),
+ weight=c(81, 93, 78),
+ Age=c(42, 38, 26))
> print(BMI)
  gender height weight Age
1   Male  152.0     81  42
2   Male  171.5     93  38
3 Female  165.0     78  26
> 
```

VARIABLES



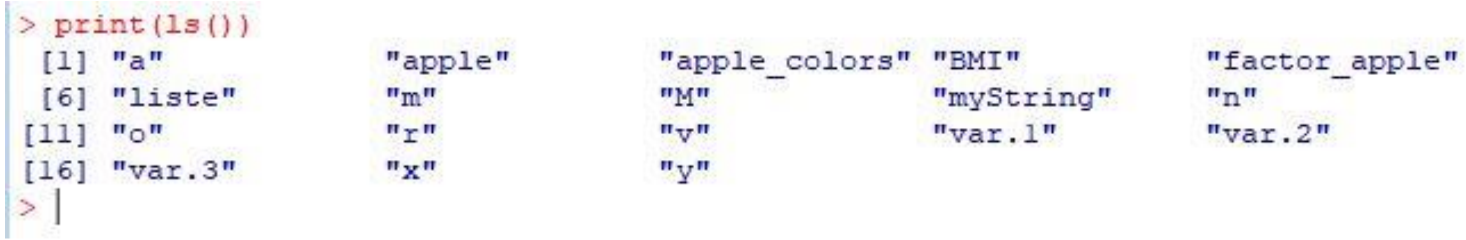
Variable Name	Validity	Reason
var_name2.	valid	Has letters, numbers, dot and underscore
var_name%	Invalid	Has the character '%'. Only dot(.) and underscore allowed.
2var_name	invalid	Starts with a number
.var_name, var.name	valid	Can start with a dot(.) but the dot(.)should not be followed by a number.
.2var_name	invalid	The starting dot is followed by a number making it invalid.
_var_name	invalid	Starts with _ which is not valid

VARIABLE ASSIGNMENT

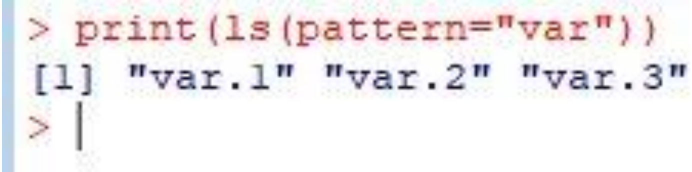


```
> var.1 = c(0,1,2,3)
> var.2 <- c("learn", "R")
> c(TRUE,1) -> var.3
> print(var.1)
[1] 0 1 2 3
> print(var.2)
[1] "learn" "R"
> print(var.3)
[1] 1 1
> cat("var.1 is", var.1, "\n")
var.1 is 0 1 2 3
> cat("var.2 is", var.2, "\n")
var.2 is learn R
> cat("var.3 is", var.3, "\n")
var.3 is 1 1
>
```

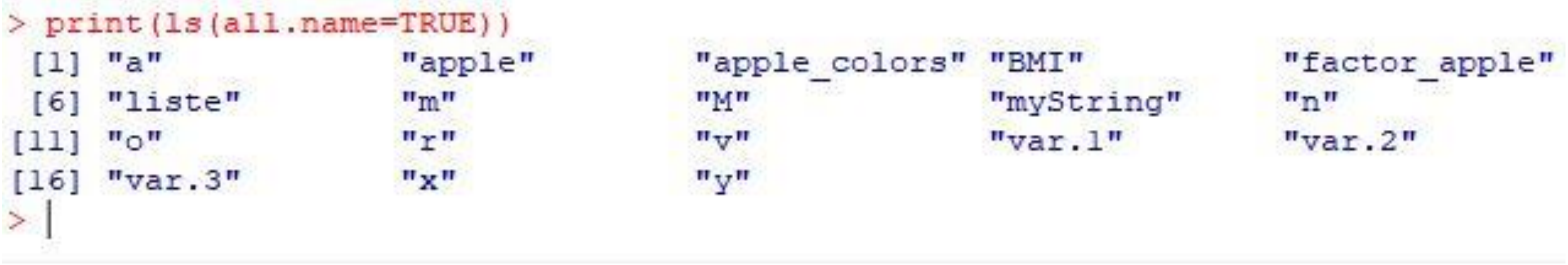
FINDING VARIABLES



```
> print(ls())
 [1] "a"            "apple"        "apple_colors" "BMI"          "factor_apple"
 [6] "liste"        "m"            "M"            "myString"     "n"
[11] "o"            "r"            "v"            "var.1"        "var.2"
[16] "var.3"        "x"            "y"
> |
```

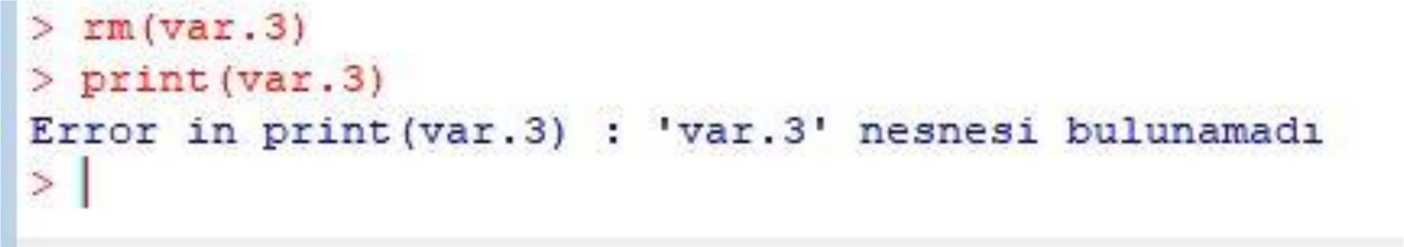


```
> print(ls(pattern="var"))
[1] "var.1" "var.2" "var.3"
> |
```

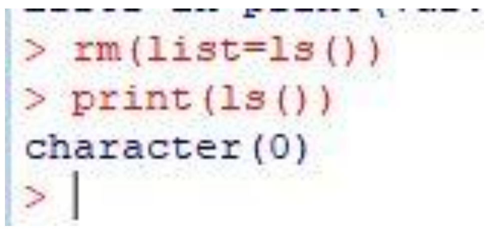


```
> print(ls(all.name=TRUE))
 [1] "a"            "apple"        "apple_colors" "BMI"          "factor_apple"
 [6] "liste"        "m"            "M"            "myString"     "n"
[11] "o"            "r"            "v"            "var.1"        "var.2"
[16] "var.3"        "x"            "y"
> |
```

DELETING VARIABLES

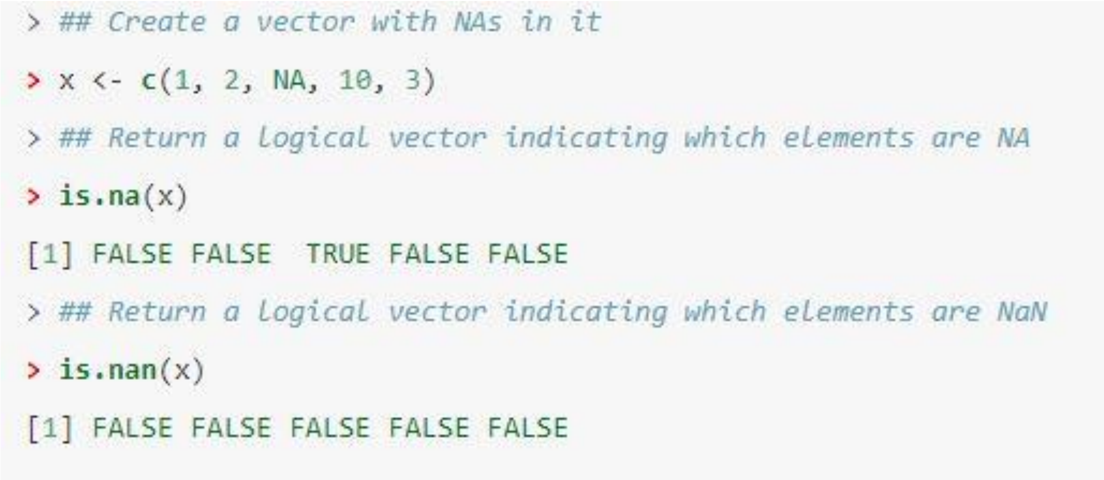


```
> rm(var.3)
> print(var.3)
Error in print(var.3) : 'var.3' nesnesi bulunamadı
> |
```

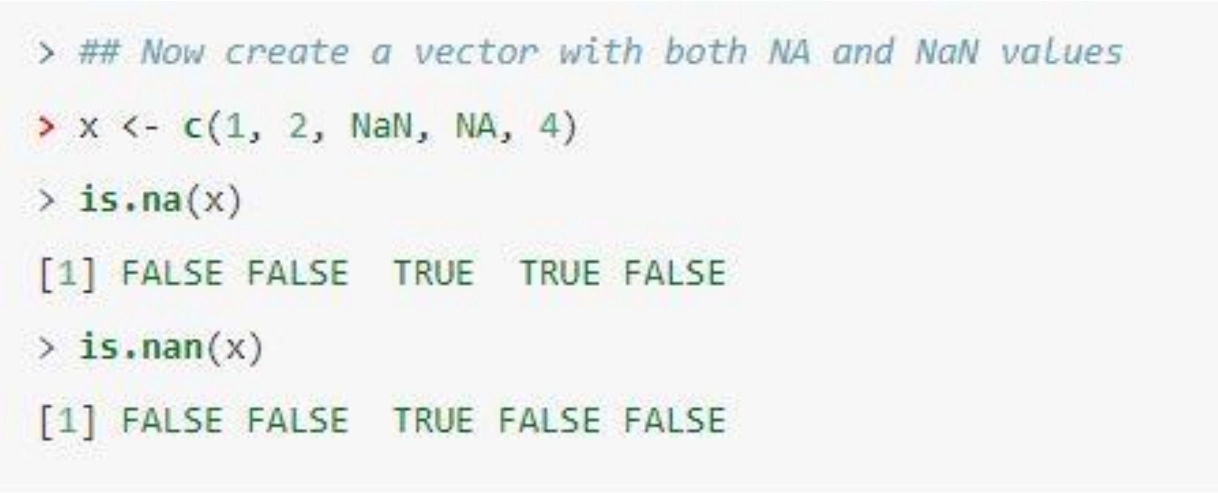


```
> rm(list=ls())
> print(ls())
character(0)
> |
```

MISSING VALUES



```
> ## Create a vector with NAs in it
> x <- c(1, 2, NA, 10, 3)
> ## Return a logical vector indicating which elements are NA
> is.na(x)
[1] FALSE FALSE  TRUE FALSE FALSE
> ## Return a logical vector indicating which elements are NaN
> is.nan(x)
[1] FALSE FALSE FALSE FALSE FALSE
```



```
> ## Now create a vector with both NA and NaN values
> x <- c(1, 2, NaN, NA, 4)
> is.na(x)
[1] FALSE FALSE  TRUE  TRUE FALSE
> is.nan(x)
[1] FALSE FALSE  TRUE FALSE FALSE
```

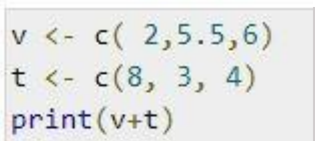
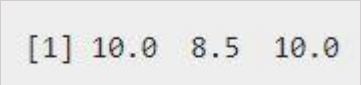
OPERATORS

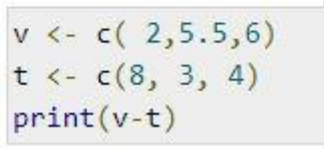
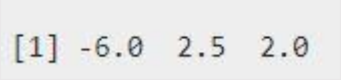
We have the following types of operators in R programming –

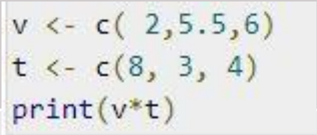
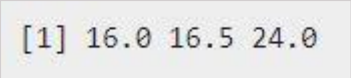
- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Miscellaneous Operators

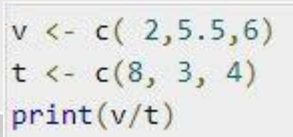
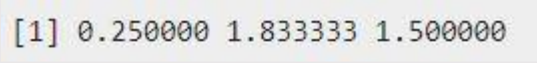
ARITHMETIC OPERATORS

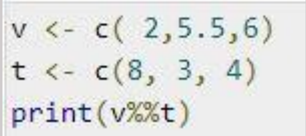
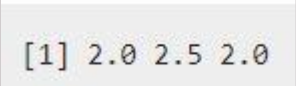
Operator	Description	Example

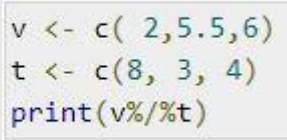
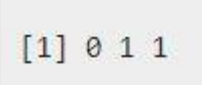
+	Adds two vectors	 `v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`
		`print(v+t)`
		 `[1] 10.0  8.5  10.0`

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

*	Multiplies both vectors	 `v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`
		`print(v*t)`
		 `[1] 16.0 16.5 24.0`

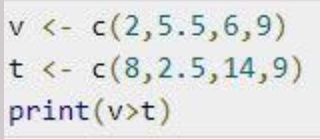
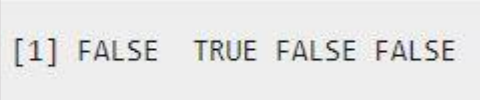
/	Divide the first vector with the second	 `v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`
		`print(v/t)`
		 `[1] 0.250000 1.833333 1.500000`

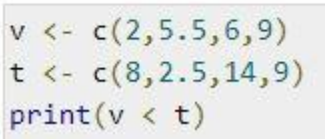
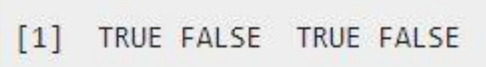
%%	Give the remainder of the first vector with the second	 `v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`
		`print(v%%t)`
		 `[1] 2.0 2.5 2.0`

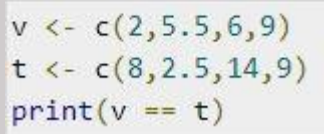
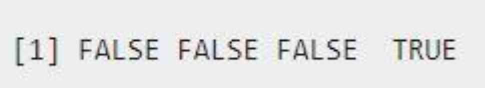
%/%	The result of division of first vector with second (quotient)	 `v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`
		`print(v%/%t)`
		 `[1] 0 1 1`

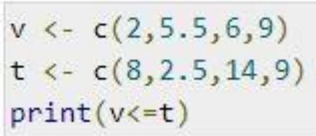
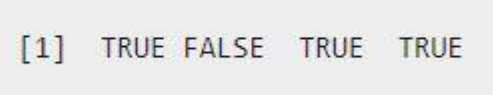
RELATIONAL OPERATORS

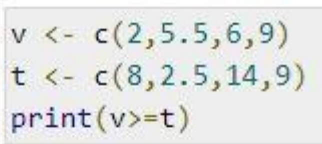
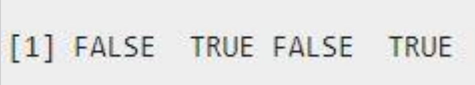
Operator	Description	Example

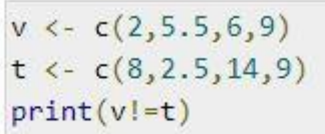
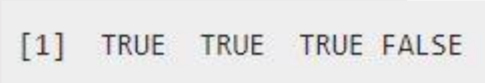
>	Checks if each element of the first vector is greater than the corresponding element of the second vector.	
		`v <- c(2,5.5,6,9)`
		`t <- c(8,2.5,14,9)`
		`print(v>t)`
		
		`[1] FALSE  TRUE FALSE FALSE`

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

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

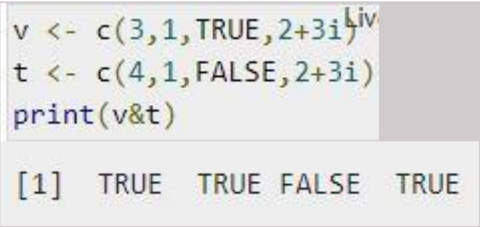
<=	Checks if each element of the first vector is less than or equal to the corresponding element of the second vector.	
		`v <- c(2,5.5,6,9)`
		`t <- c(8,2.5,14,9)`
		`print(v<=t)`
		
		`[1]  TRUE FALSE  TRUE  TRUE`

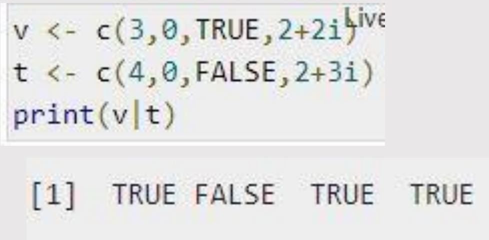
>=	Checks if each element of the first vector is greater than or equal to the corresponding element of the second vector.	
		`v <- c(2,5.5,6,9)`
		`t <- c(8,2.5,14,9)`
		`print(v>=t)`
		
		`[1] FALSE  TRUE FALSE  TRUE`

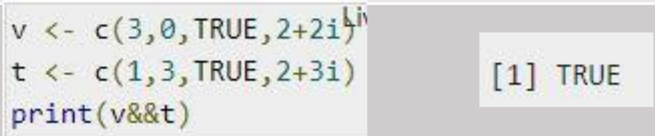
!=	Checks if each element of the first vector is unequal to the corresponding element of the second vector.	
		`v <- c(2,5.5,6,9)`
		`t <- c(8,2.5,14,9)`
		`print(v!=t)`
		
		`[1]  TRUE  TRUE  TRUE FALSE`

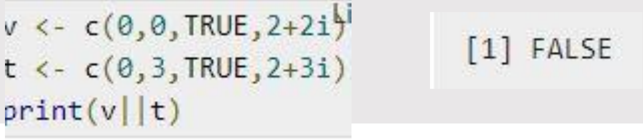
LOGICAL OPERATORS

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.	
		`v <- c(3,1,TRUE,2+3i)`
		`t <- c(4,1,FALSE,2+3i)`
		`print(v&t)`
		`[1]  TRUE  TRUE FALSE  TRUE`

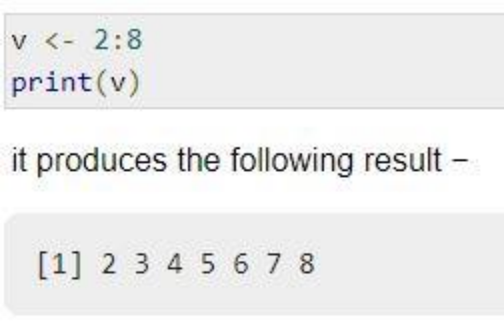
|	It is called Element-wise Logical OR operator. It combines each element of the first vector with the corresponding element of the second vector and gives a output TRUE if one the elements is TRUE.	
		`v <- c(3,0,TRUE,2+2i)`
		`t <- c(4,0,FALSE,2+3i)`
		`print(v|t)`
		`[1]  TRUE FALSE  TRUE  TRUE`

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

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

MISCELLANEOUS OPERATORS

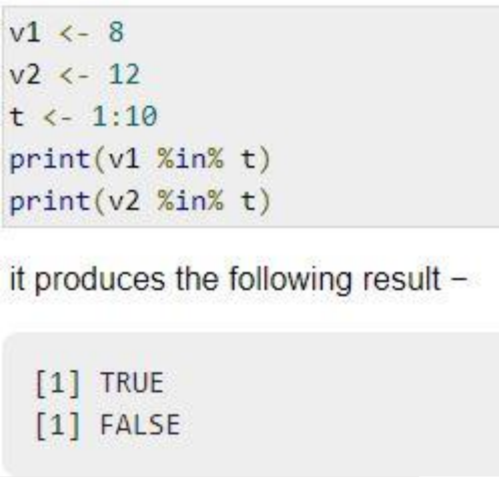
Operator	Description	Example
:	Colon operator. It creates the series of numbers in sequence for a vector.	
%in%	This operator is used to identify if an element belongs to a vector.	
%*%	This operator is used to multiply a matrix with its transpose	



```
v <- 2:8
print(v)
```

it produces the following result –

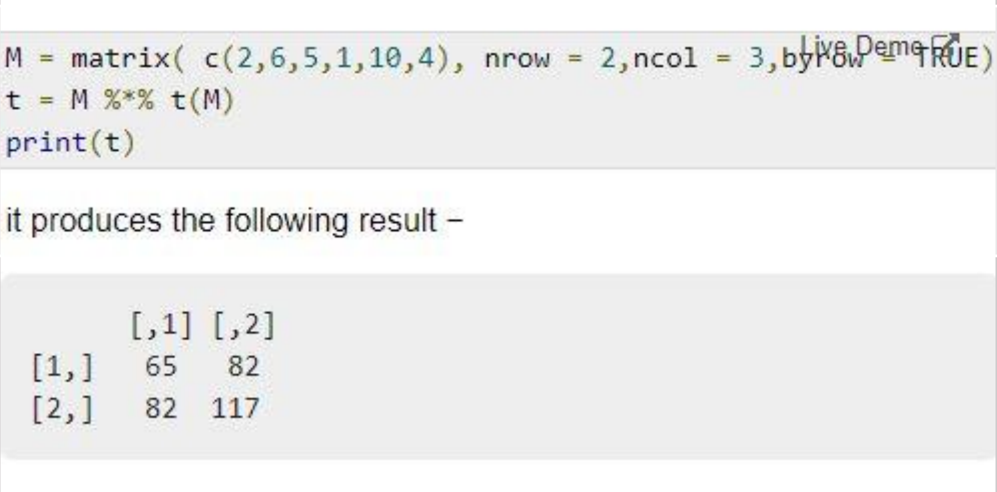
```
[1] 2 3 4 5 6 7 8
```



```
v1 <- 8
v2 <- 12
t <- 1:10
print(v1 %in% t)
print(v2 %in% t)
```

it produces the following result –

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



```
M = matrix( c(2,6,5,1,10,4), nrow = 2,ncol = 3,byrow = TRUE)
t = M %*% t(M)
print(t)
```

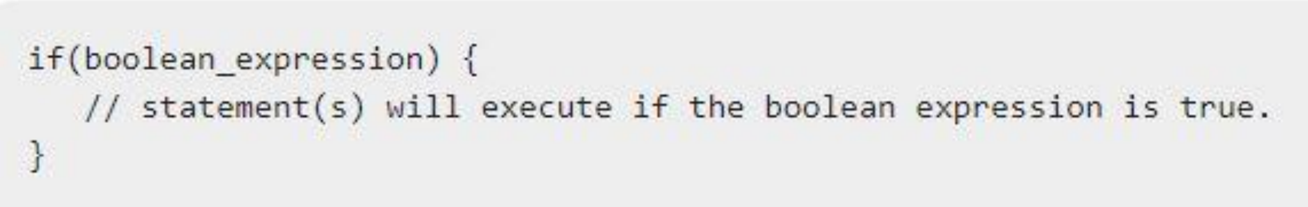
it produces the following result –

```
      [,1] [,2]
[1,]   65   82
[2,]   82  117
```

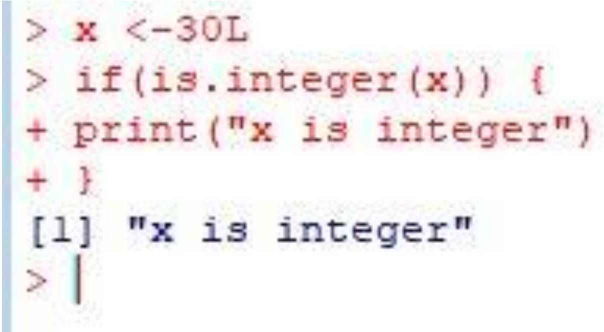
DECISION MAKING

Sr. No	Statement & Description
1	**if statement** An **if** statement consists of a Boolean expression followed by one or more statements.
2	**if...else statement** An **if** statement can be followed by an optional **else** statement, which executes when the Boolean expression is false.
3	**switch statement** A **switch** statement allows a variable to be tested for equality against a list of values.

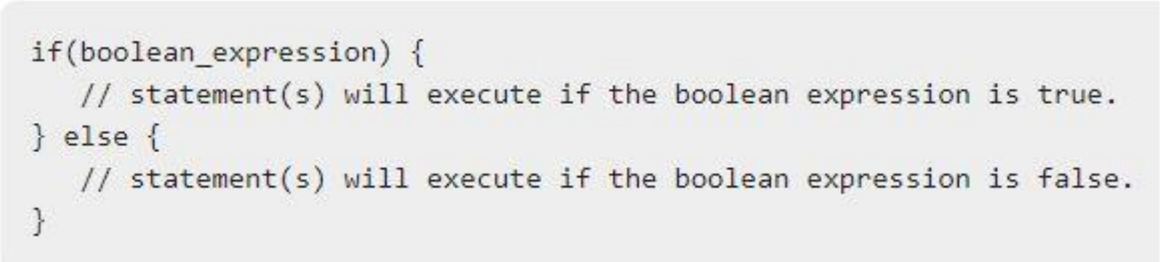
IF STATEMENT



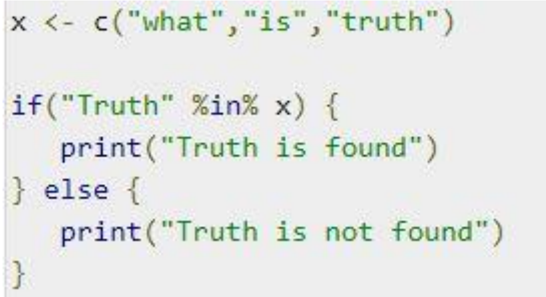
```
if(boolean_expression) {
   // statement(s) will execute if the boolean expression is true.
}
```



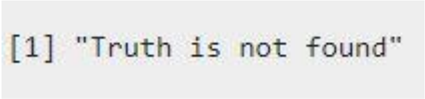
```
> x <-30L
> if(is.integer(x)) {
+ print("x is integer")
+ }
[1] "x is integer"
> 
```



```
if(boolean_expression) {
   // statement(s) will execute if the boolean expression is true.
} else {
   // statement(s) will execute if the boolean expression is false.
}
```



```
x <- c("what","is","truth")

if("Truth" %in% x) {
   print("Truth is found")
} else {
   print("Truth is not found")
}
```

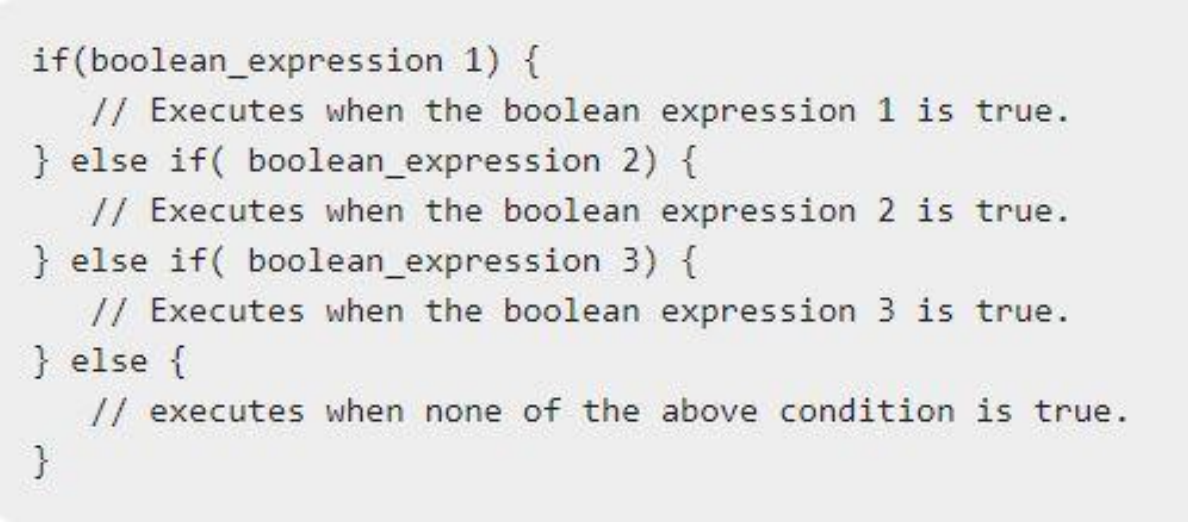


```
[1] "Truth is not found"
```

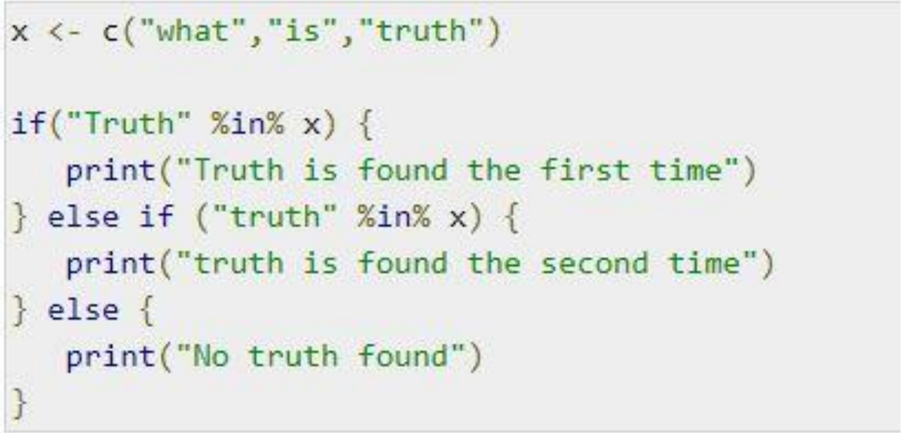
IF.. ELSE IF …ELSE STATEMENT

- An **if** statement can be followed by an optional **else if...else** statement, which is very useful to test various conditions using single if...else if statement.
- When using **if**, **else if**, **else** statements there are few points to keep in mind.
- An **if** can have zero or one **else** and it must come after any **else if**'s.
- An **if** can have zero to many **else if's** and they must come before the else.
- Once an **else if** succeeds, none of the remaining **else if**'s or **else**'s will be tested.

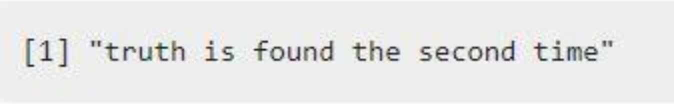
IF.. ELSE IF …ELSE STATEMENT-SYNTAX



```
if(boolean_expression 1) {
   // Executes when the boolean expression 1 is true.
} else if( boolean_expression 2) {
   // Executes when the boolean expression 2 is true.
} else if( boolean_expression 3) {
   // Executes when the boolean expression 3 is true.
} else {
   // executes when none of the above condition is true.
}
```

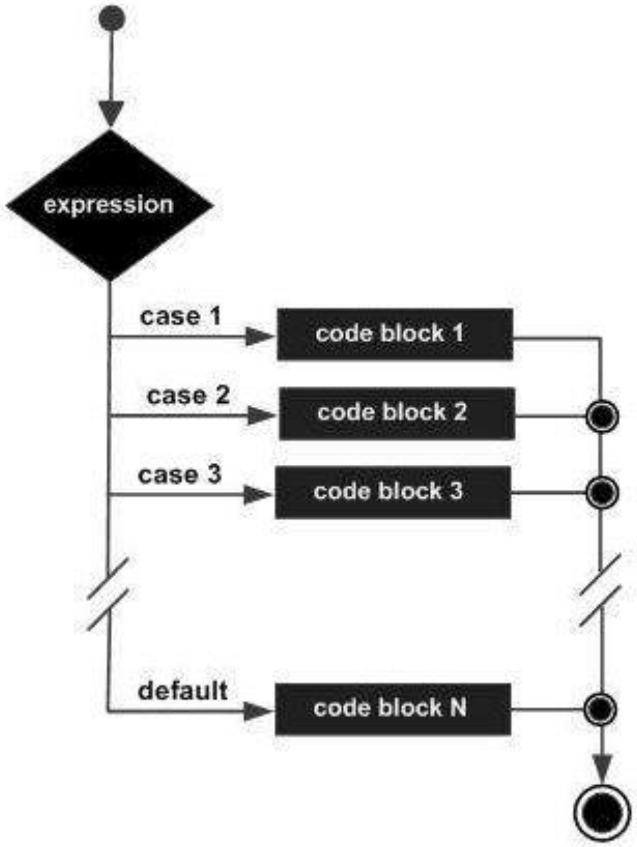


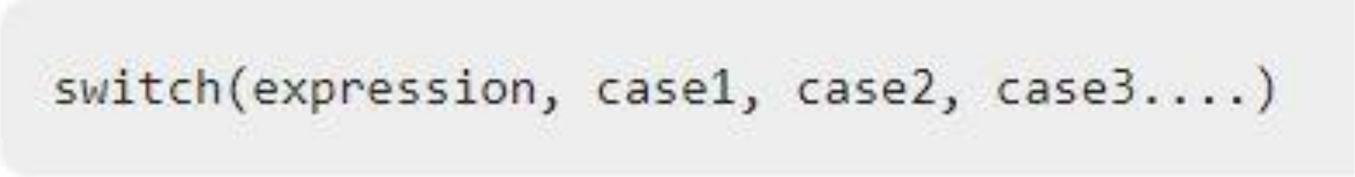
```
x <- c("what","is","truth")

if("Truth" %in% x) {
   print("Truth is found the first time")
} else if ("truth" %in% x) {
   print("truth is found the second time")
} else {
   print("No truth found")
}
```



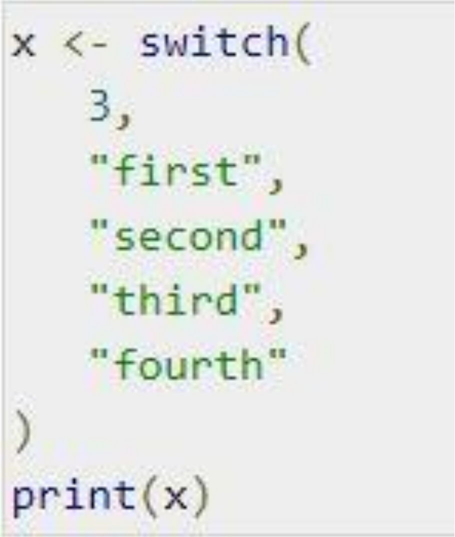
```
[1] "truth is found the second time"
```

SWITCH STATEMENT

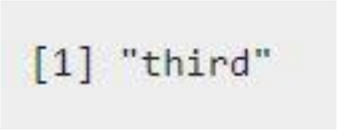




```
switch(expression, case1, case2, case3....)
```

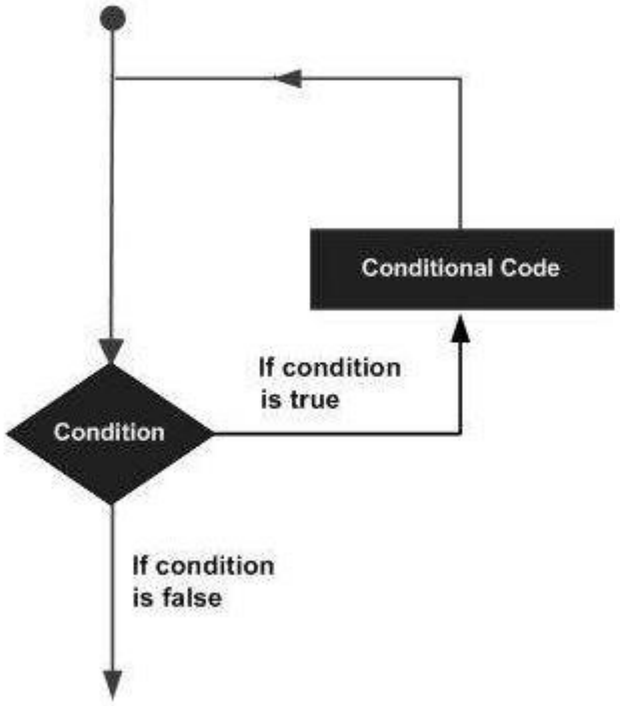


```
x <- switch(
    3,
    "first",
    "second",
    "third",
    "fourth"
)
print(x)
```



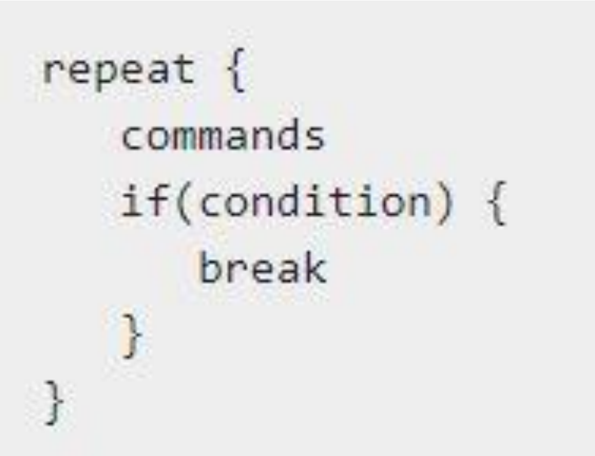
```
[1] "third"
```

LOOPS

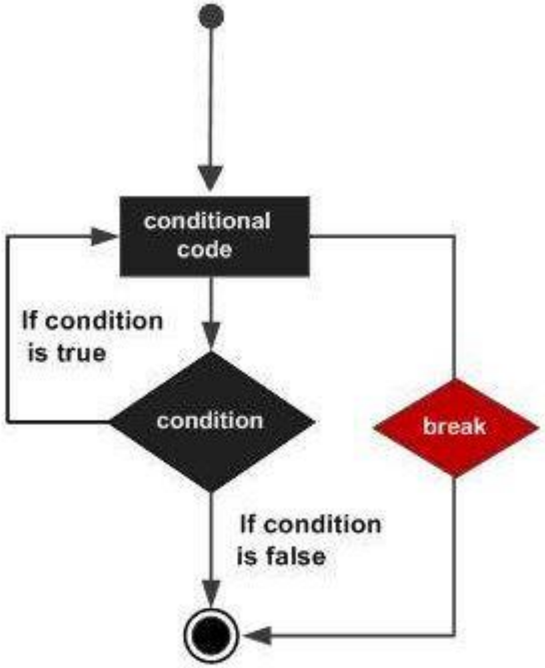


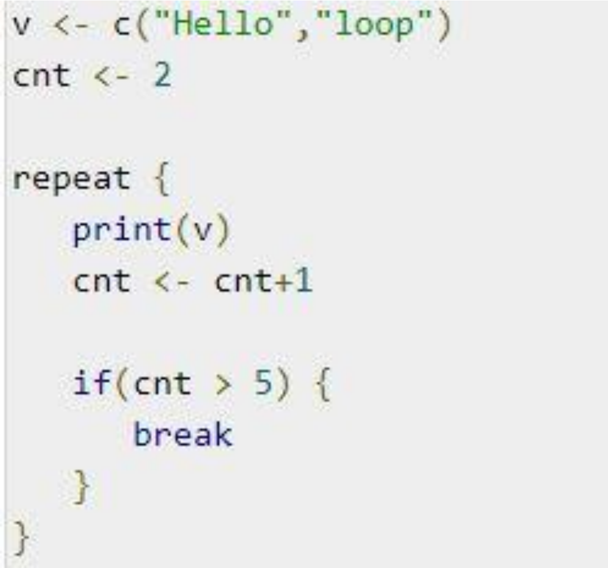
Sr. No	Statement & Description
1	**repeat loop** Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
2	**while loop** Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
3	**for loop** Like a while statement, except that it tests the condition at the end of the loop body.

REPEAT LOOP

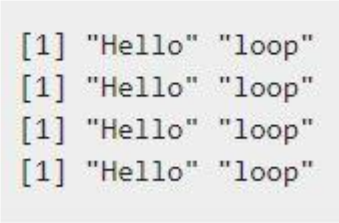


```
repeat {
   commands
   if(condition) {
      break
   }
}
```



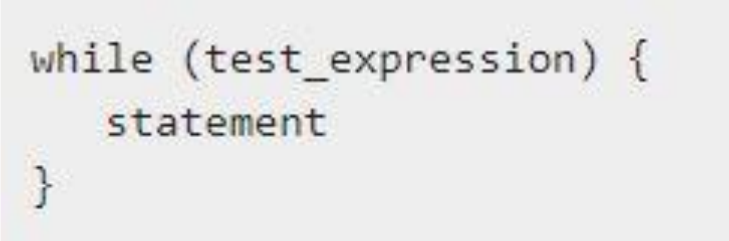


```
v <- c("Hello","loop")
cnt <- 2

repeat {
   print(v)
   cnt <- cnt+1

   if(cnt > 5) {
      break
   }
}
```

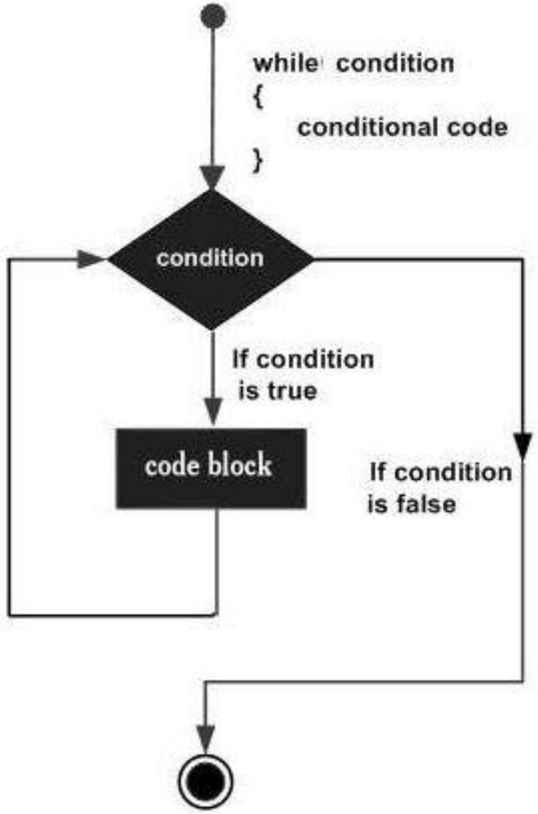


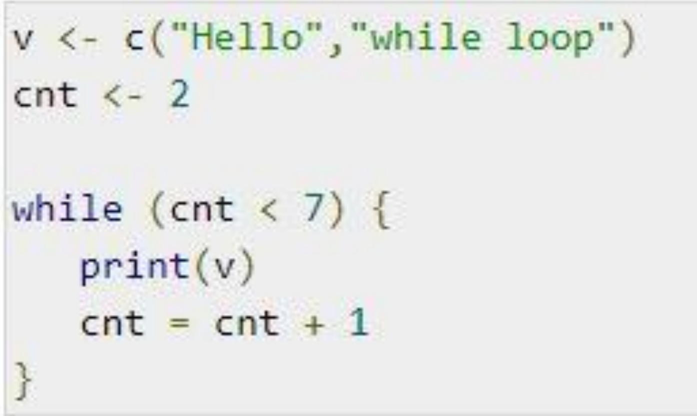
```
[1] "Hello" "loop"
[1] "Hello" "loop"
[1] "Hello" "loop"
[1] "Hello" "loop"
```

WHILE LOOP

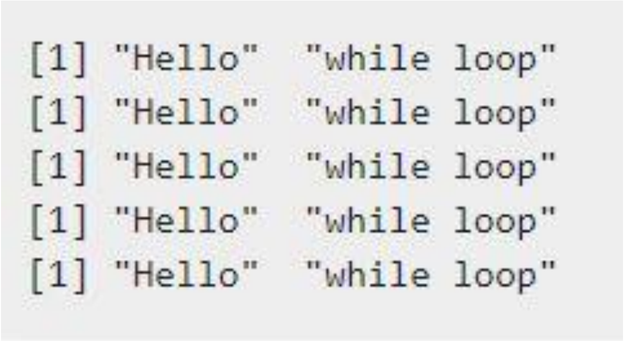


```
while (test_expression) {
   statement
}
```



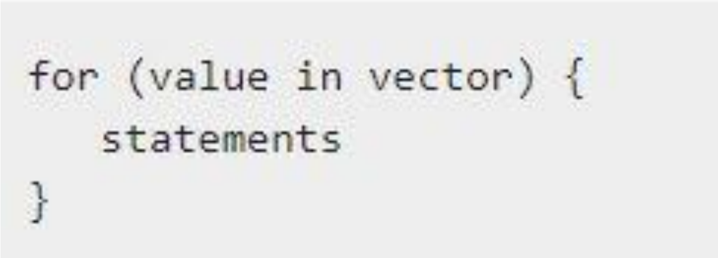


```
v <- c("Hello","while loop")
cnt <- 2

while (cnt < 7) {
   print(v)
   cnt = cnt + 1
}
```

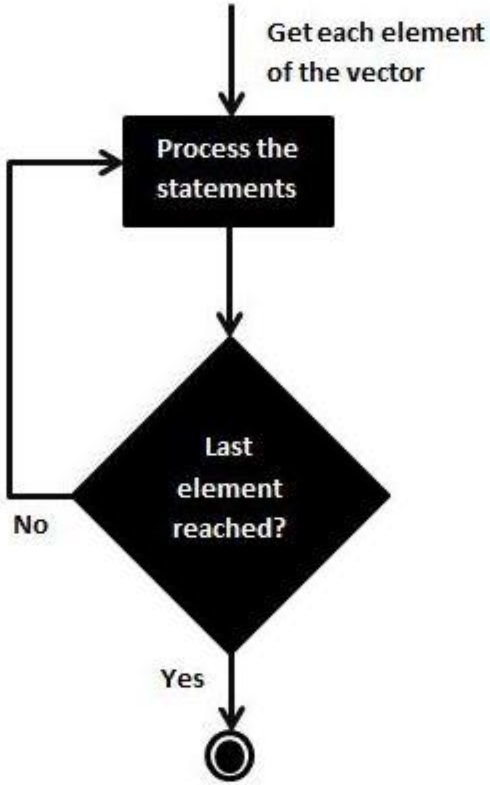


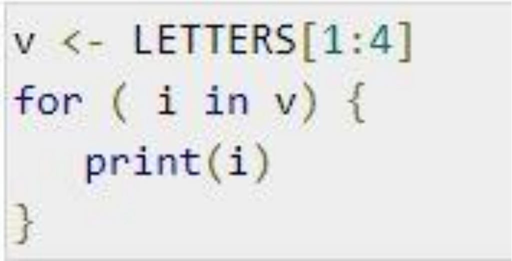
```
[1] "Hello"  "while loop"
[1] "Hello"  "while loop"
[1] "Hello"  "while loop"
[1] "Hello"  "while loop"
[1] "Hello"  "while loop"
```

FOR LOOP

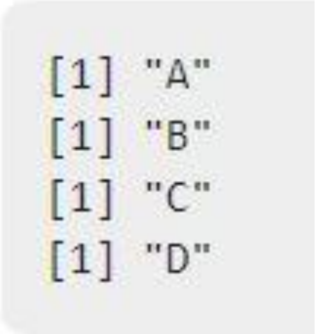


```
for (value in vector) {
   statements
}
```





```
v <- LETTERS[1:4]
for ( i in v) {
   print(i)
}
```

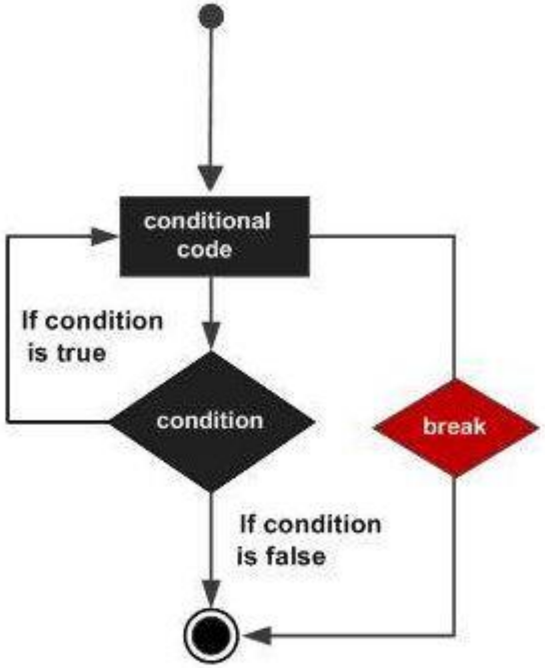


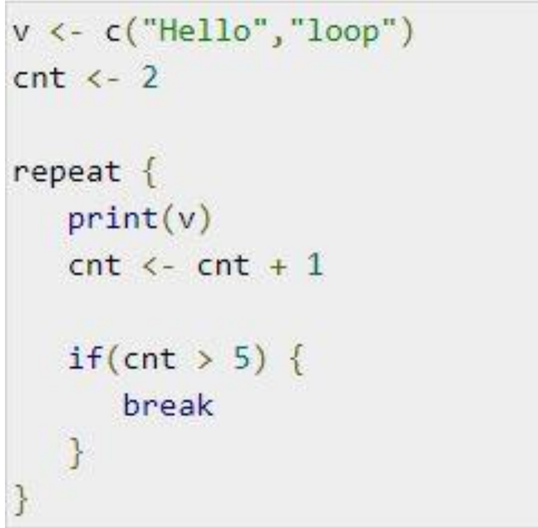
```
[1] "A"
[1] "B"
[1] "C"
[1] "D"
```

LOOP CONTROL STATEMENTS

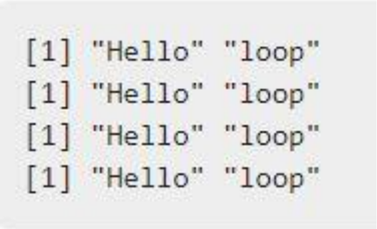
Sr. No	Statement & Description
1	**break statement** Terminates the **loop** statement and transfers execution to the statement immediately following the loop.
2	**Next statement** The **next** statement simulates the behavior of R switch.

BREAK STATEMENT



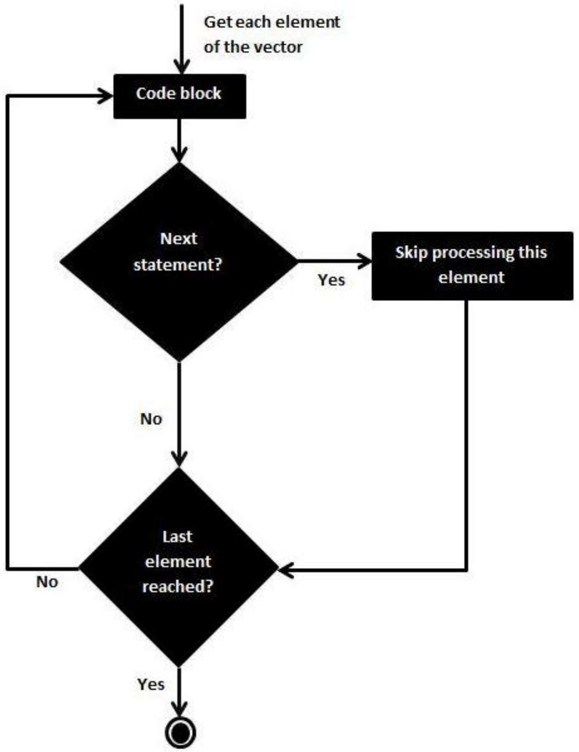


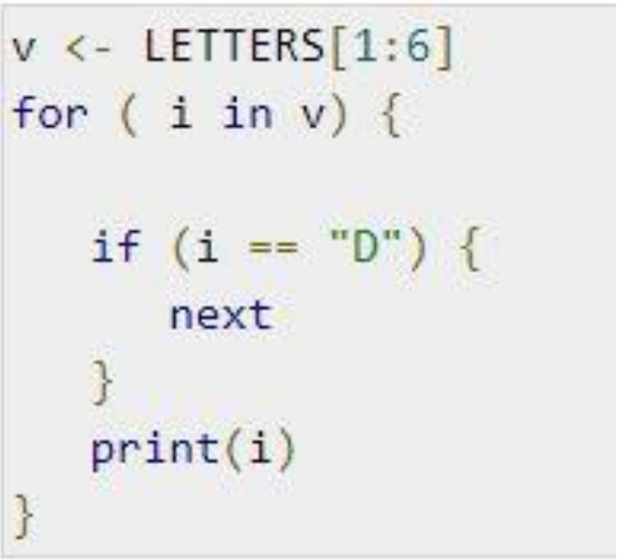
```
v <- c("Hello","loop")
cnt <- 2

repeat {
   print(v)
   cnt <- cnt + 1

   if(cnt > 5) {
      break
   }
}
```



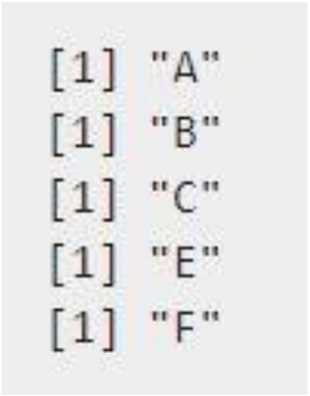
```
[1] "Hello" "loop"
[1] "Hello" "loop"
[1] "Hello" "loop"
[1] "Hello" "loop"
```

NEXT STATEMENT



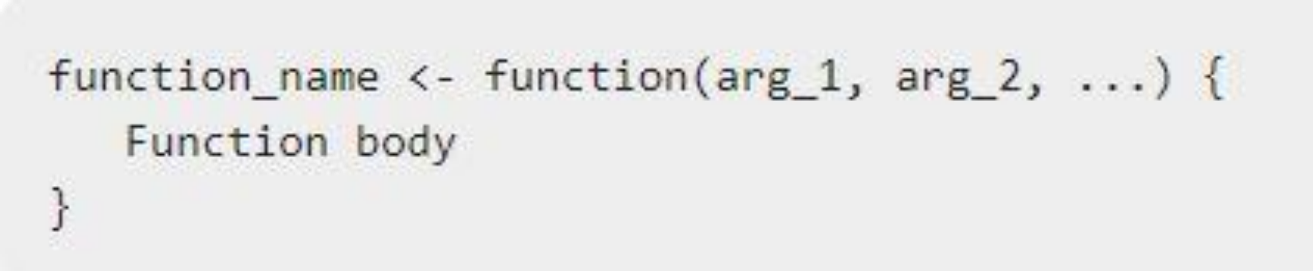


```
v <- LETTERS[1:6]
for ( i in v) {

   if (i == "D") {
      next
   }
   print(i)
}
```



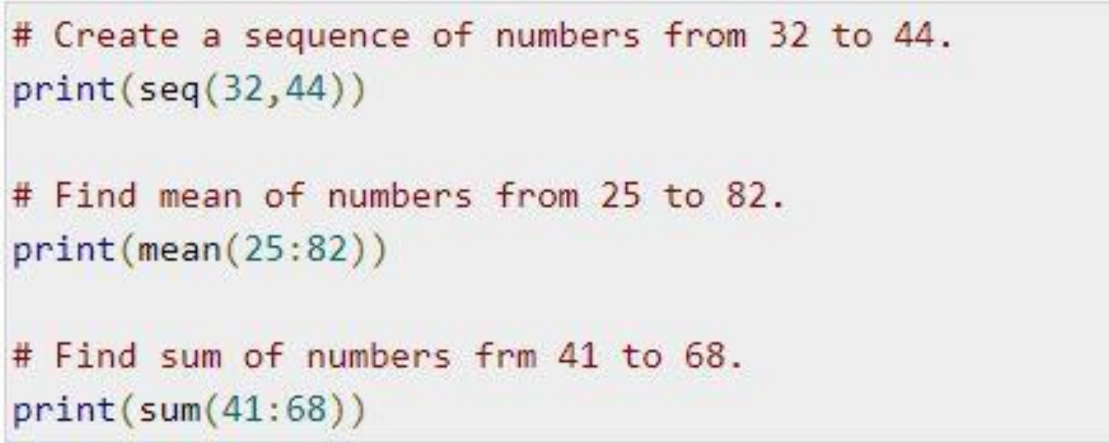
```
[1] "A"
[1] "B"
[1] "C"
[1] "E"
[1] "F"
```

FUNCTIONS

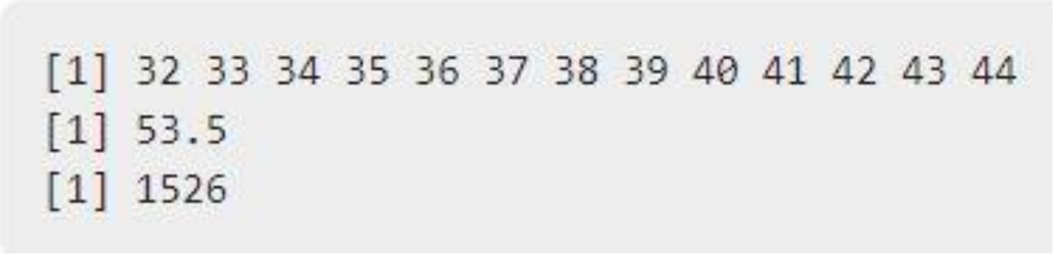


```
function_name <- function(arg_1, arg_2, ...) {
   Function body
}
```

- Built-in function : seq(), mean(), max(), sum(x) and paste(...)



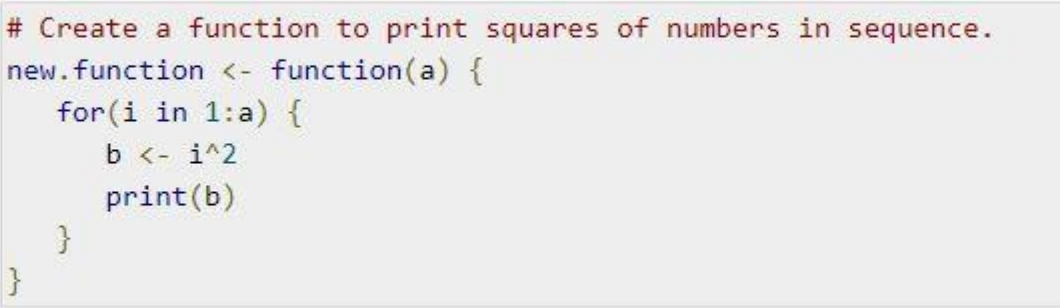
```
# Create a sequence of numbers from 32 to 44.
print(seq(32,44))

# Find mean of numbers from 25 to 82.
print(mean(25:82))

# Find sum of numbers frm 41 to 68.
print(sum(41:68))
```



```
[1] 32 33 34 35 36 37 38 39 40 41 42 43 44
[1] 53.5
[1] 1526
```

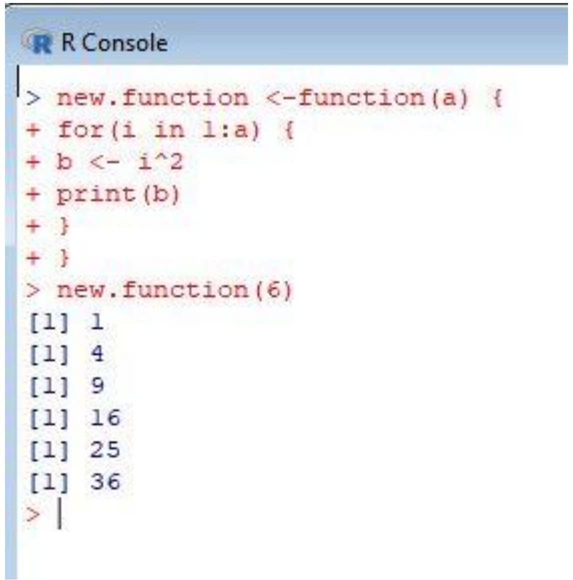
FUNCTIONS

- User-defined functions



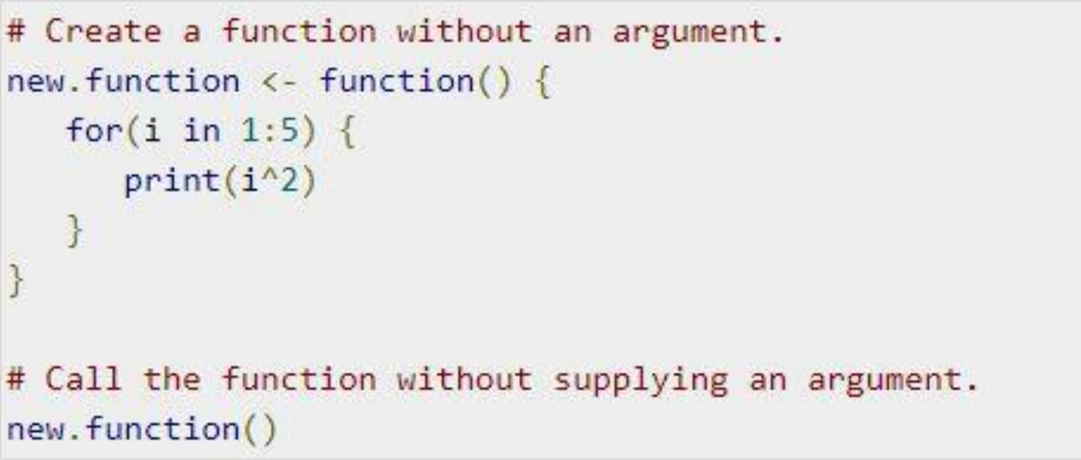
```
# Create a function to print squares of numbers in sequence.
new.function <- function(a) {
   for(i in 1:a) {
      b <- i^2
      print(b)
   }
}
```

- User-defined functions

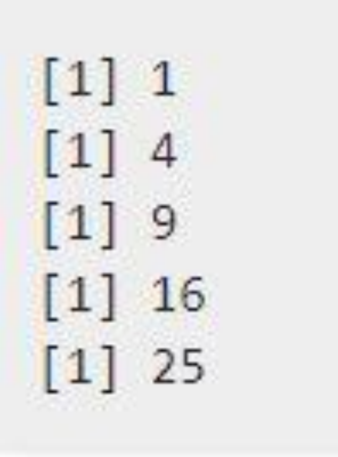


```
> new.function <-function(a) {
+ for(i in 1:a) {
+ b <- i^2
+ print(b)
+ }
+ }
> new.function(6)
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36
> 
```

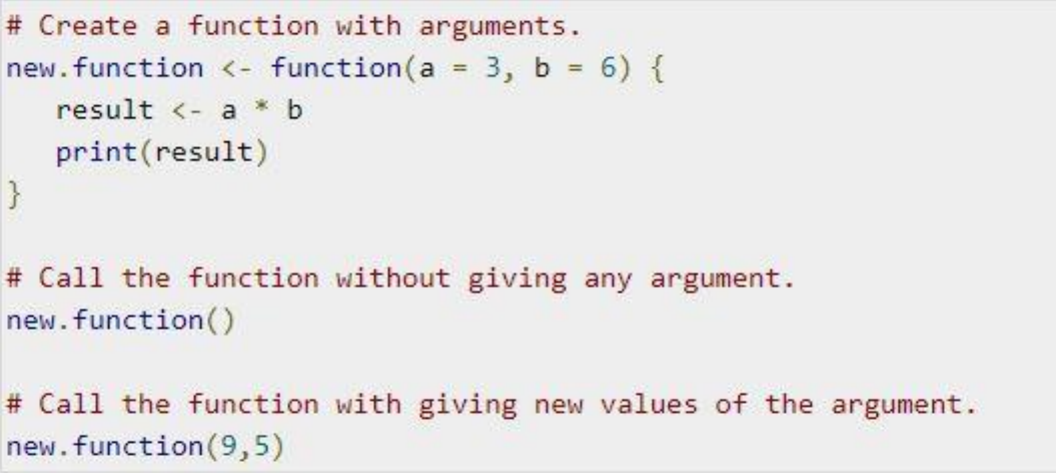
FUNCTIONS



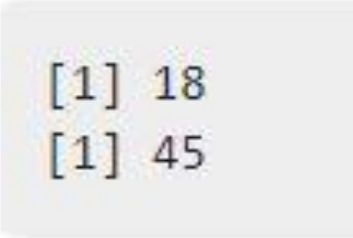
```
# Create a function without an argument.
new.function <- function() {
   for(i in 1:5) {
      print(i^2)
   }
}

# Call the function without supplying an argument.
new.function()
```



```
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
```

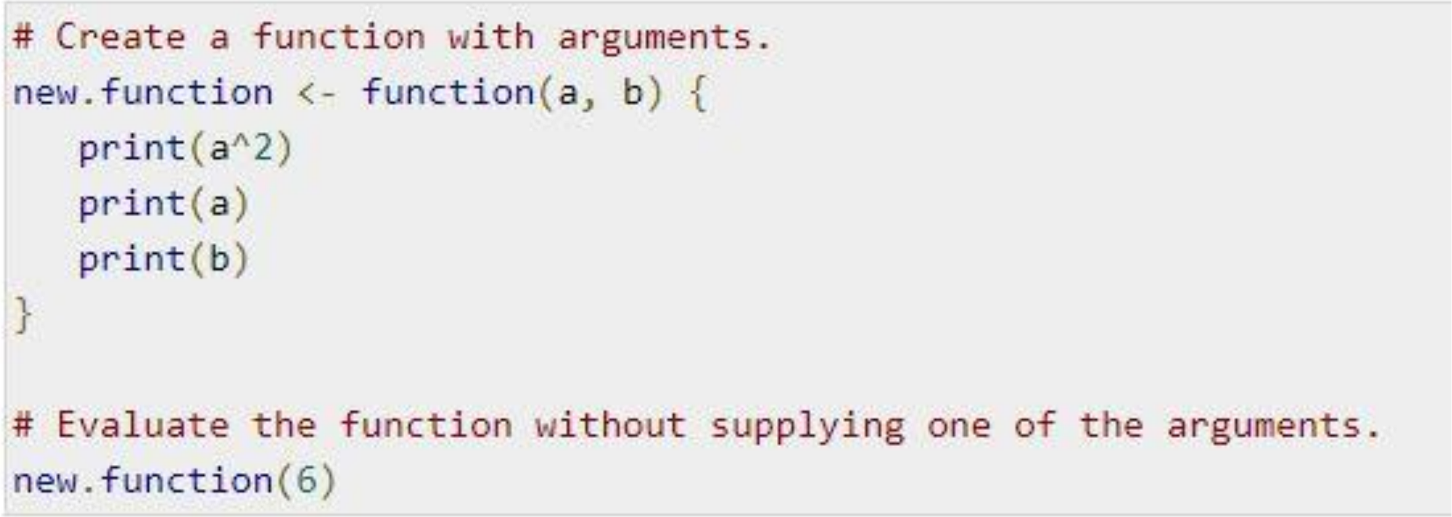


```
# Create a function with arguments.
new.function <- function(a = 3, b = 6) {
   result <- a * b
   print(result)
}

# Call the function without giving any argument.
new.function()

# Call the function with giving new values of the argument.
new.function(9,5)
```

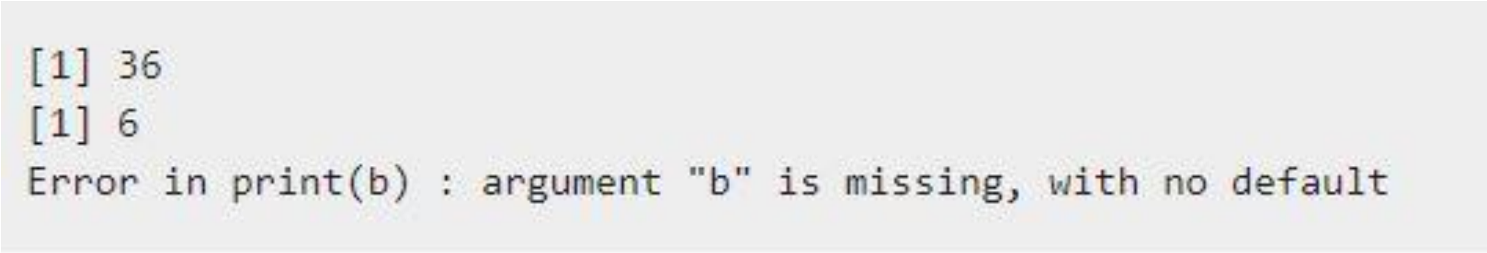


```
[1] 18
[1] 45
```

LAZY EVALUATİON OF FUNCTIONS

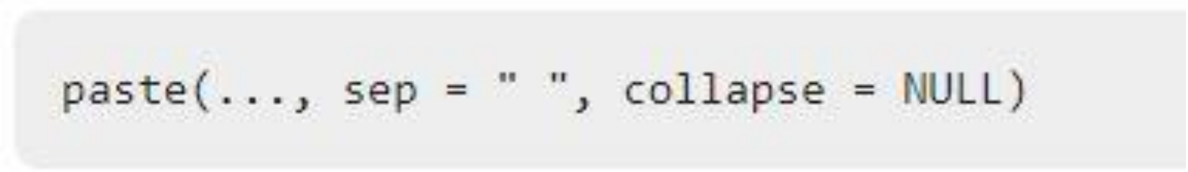


```
# Create a function with arguments.
new.function <- function(a, b) {
   print(a^2)
   print(a)
   print(b)
}

# Evaluate the function without supplying one of the arguments.
new.function(6)
```



```
[1] 36
[1] 6
Error in print(b) : argument "b" is missing, with no default
```

STRING MANIPULATION-PASTE

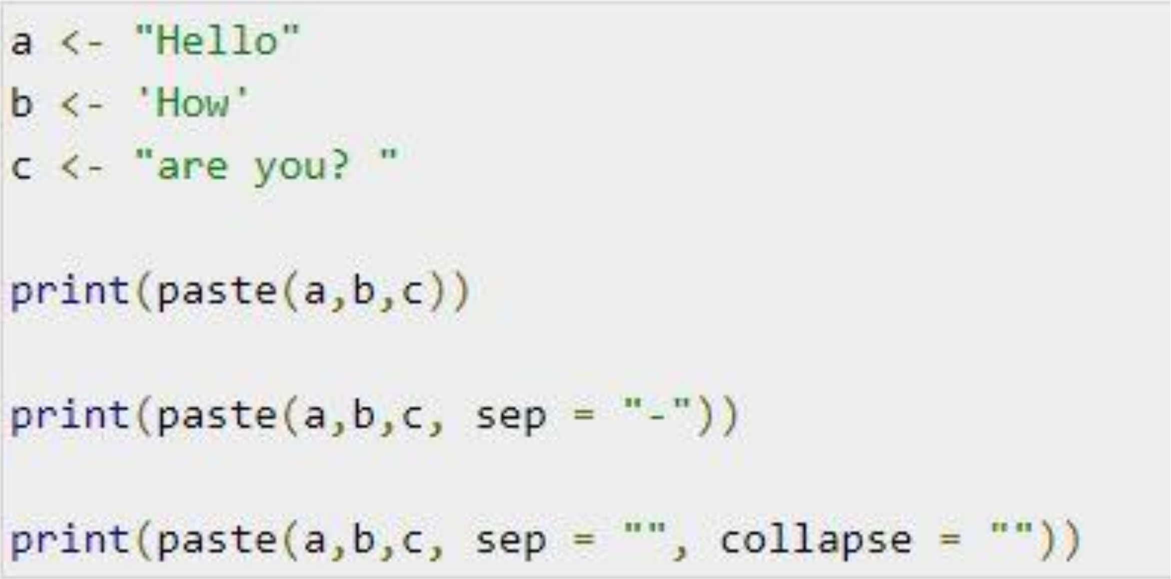


```
paste(..., sep = " ", collapse = NULL)
```

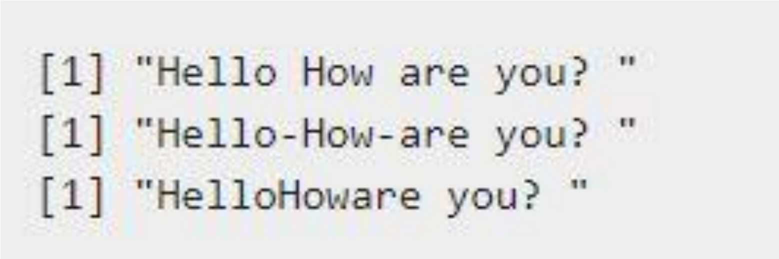
Following is the description of the parameters used −

- ... represents any number of arguments to be combined.
- **sep** represents any separator between the arguments. It is optional.
- **collapse** is used to eliminate the space in between two strings. But not the space within two words of one string.

STRING MANIPULATION-PASTE

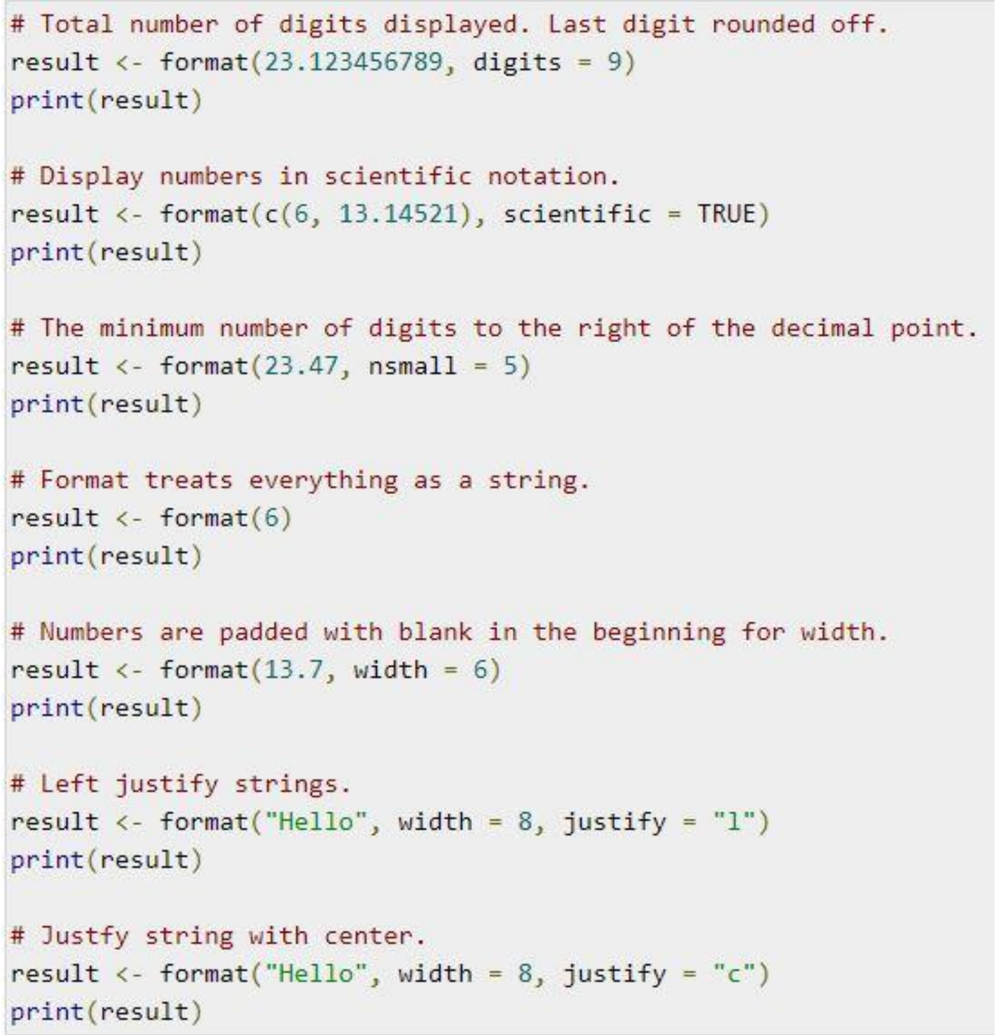


```
a <- "Hello"
b <- 'How'
c <- "are you? "

print(paste(a,b,c))

print(paste(a,b,c, sep = "-"))

print(paste(a,b,c, sep = "", collapse = ""))
```

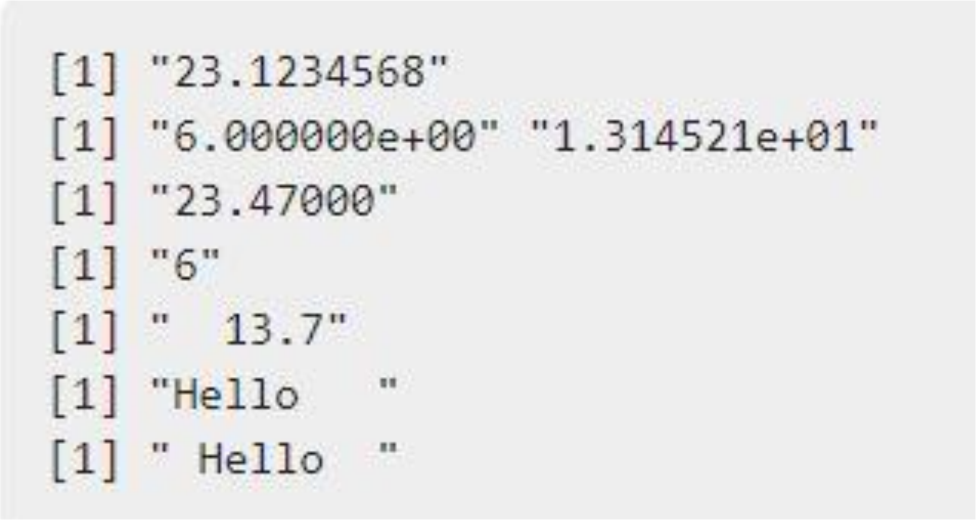


```
[1] "Hello How are you? "
[1] "Hello-How-are you? "
[1] "HelloHoware you? "
```

STRING MANIPULATION-FORMAT

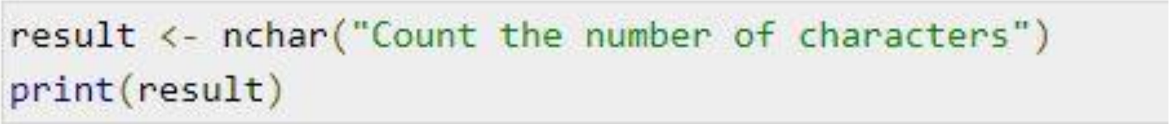


```
# Total number of digits displayed. Last digit rounded off.
result <- format(23.123456789, digits = 9)
print(result)

# Display numbers in scientific notation.
result <- format(c(6, 13.14521), scientific = TRUE)
print(result)

# The minimum number of digits to the right of the decimal point.
result <- format(23.47, nsmall = 5)
print(result)

# Format treats everything as a string.
result <- format(6)
print(result)

# Numbers are padded with blank in the beginning for width.
result <- format(13.7, width = 6)
print(result)

# Left justify strings.
result <- format("Hello", width = 8, justify = "l")
print(result)

# Justfy string with center.
result <- format("Hello", width = 8, justify = "c")
print(result)
```

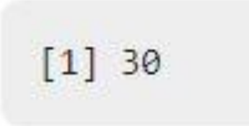


```
[1] "23.1234568"
[1] "6.000000e+00" "1.314521e+01"
[1] "23.47000"
[1] "6"
[1] "  13.7"
[1] "Hello   "
[1] " Hello  "
```

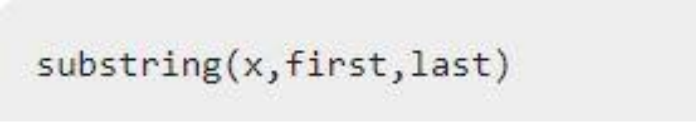
STRING MANIPULATION-NCHAR()-TOUPPER()-TOLOWER()-SUBSTRING()



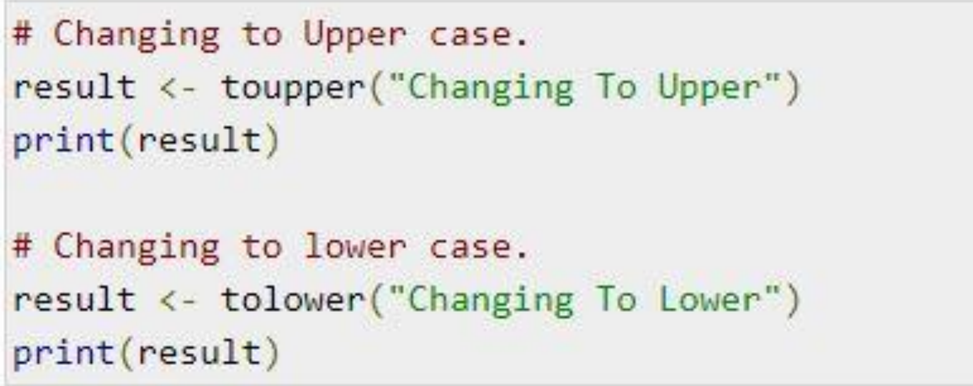
```r
result <- nchar("Count the number of characters")
print(result)
```



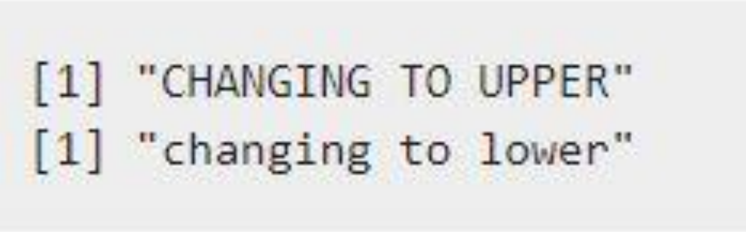
```
[1] 30
```



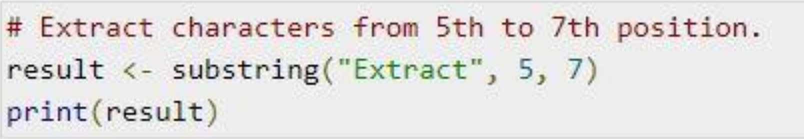
```r
substring(x,first,last)
```



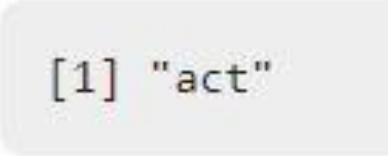
```r
# Changing to Upper case.
result <- toupper("Changing To Upper")
print(result)

# Changing to lower case.
result <- tolower("Changing To Lower")
print(result)
```



```
[1] "CHANGING TO UPPER"
[1] "changing to lower"
```

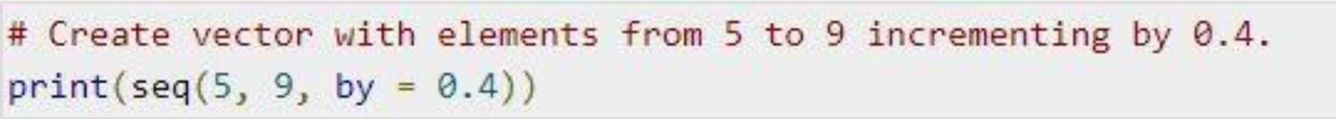


```r
# Extract characters from 5th to 7th position.
result <- substring("Extract", 5, 7)
print(result)
```

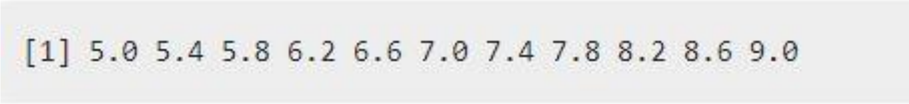


```
[1] "act"
```

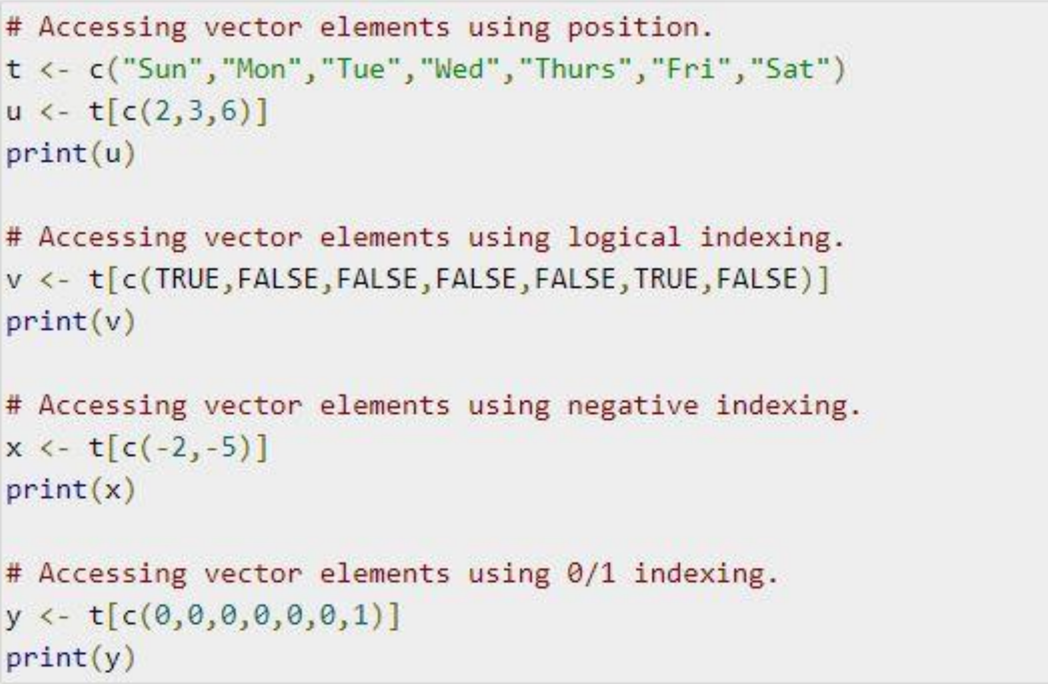
VECTOR



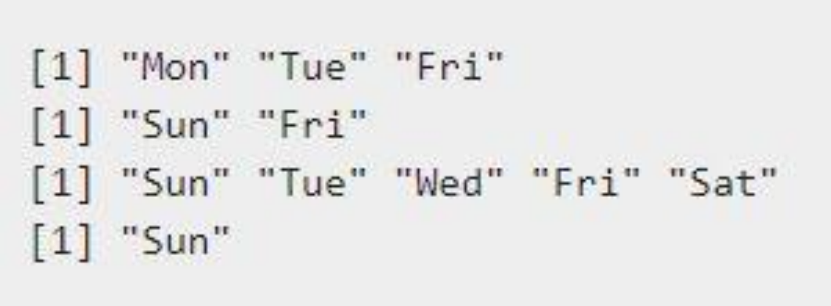
```
# Create vector with elements from 5 to 9 incrementing by 0.4.
print(seq(5, 9, by = 0.4))
```



```
[1] 5.0 5.4 5.8 6.2 6.6 7.0 7.4 7.8 8.2 8.6 9.0
```

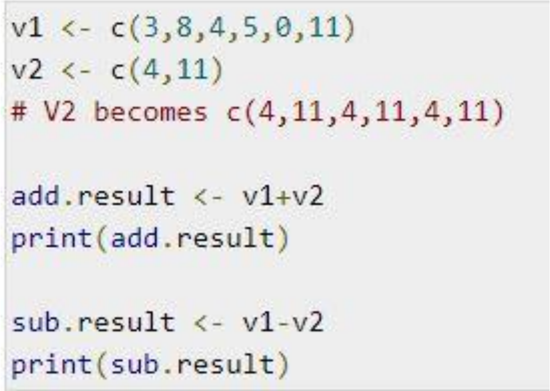


```
# Accessing vector elements using position.
t <- c("Sun","Mon","Tue","Wed","Thurs","Fri","Sat")
u <- t[c(2,3,6)]
print(u)

# 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)
```

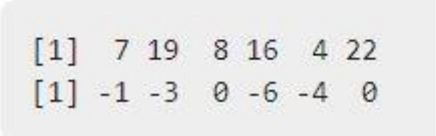


```
[1] "Mon" "Tue" "Fri"
[1] "Sun" "Fri"
[1] "Sun" "Tue" "Wed" "Fri" "Sat"
[1] "Sun"
```

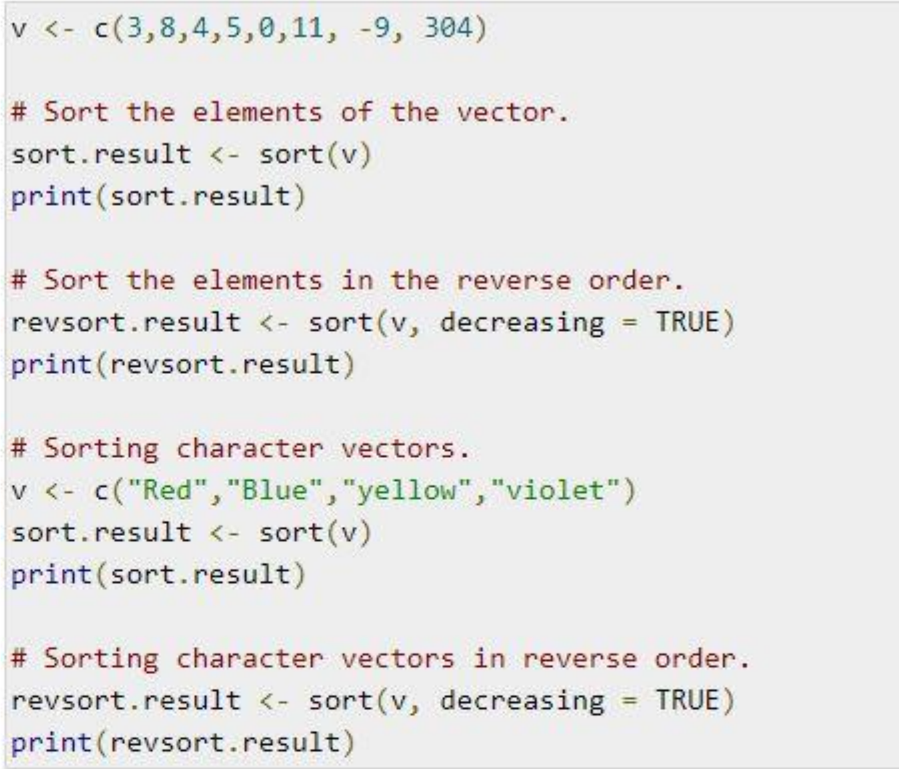
VECTOR



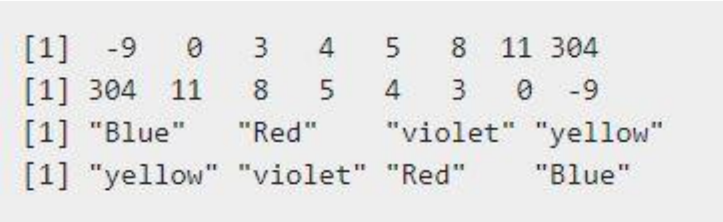
```
v1 <- c(3,8,4,5,0,11)
v2 <- c(4,11)
# V2 becomes c(4,11,4,11,4,11)

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

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



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

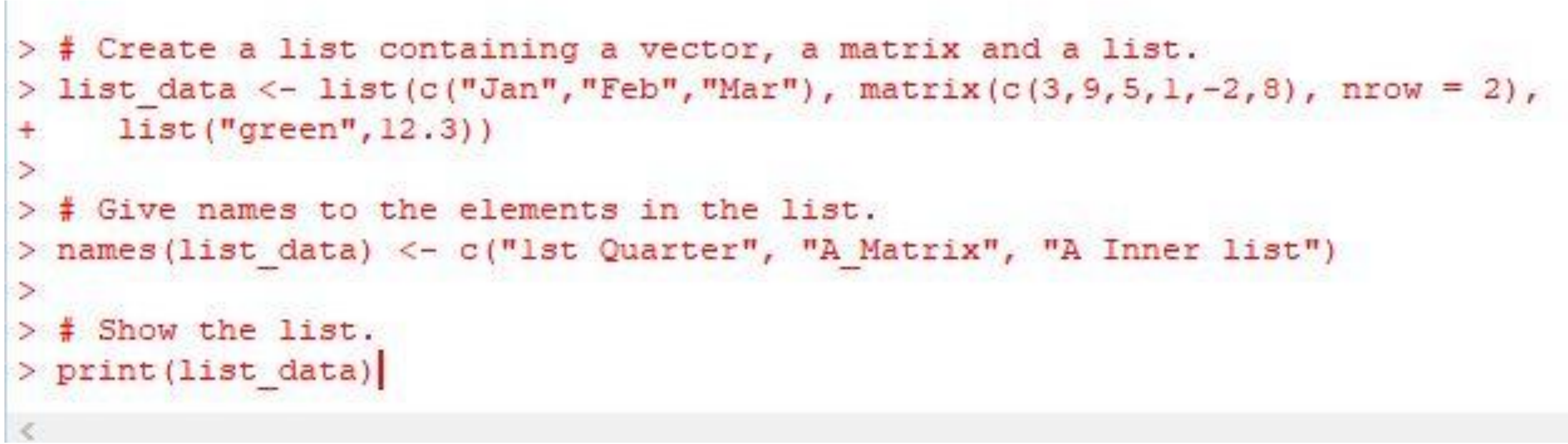


```
v <- c(3,8,4,5,0,11, -9, 304)

# 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)
```

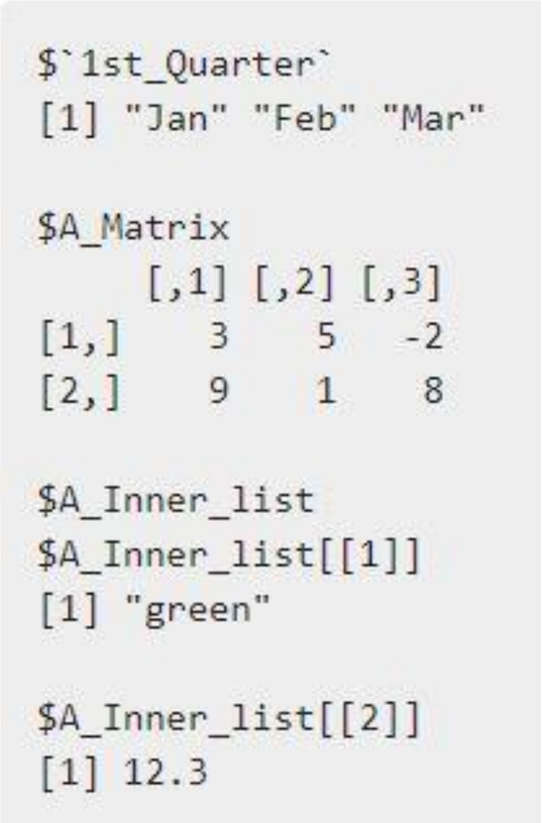


```
[1]  -9   0   3   4   5   8  11 304
[1] 304  11   8   5   4   3   0  -9
[1] "Blue"   "Red"    "violet" "yellow"
[1] "yellow" "violet" "Red"    "Blue"
```

NAMING LIST ELEMENTS

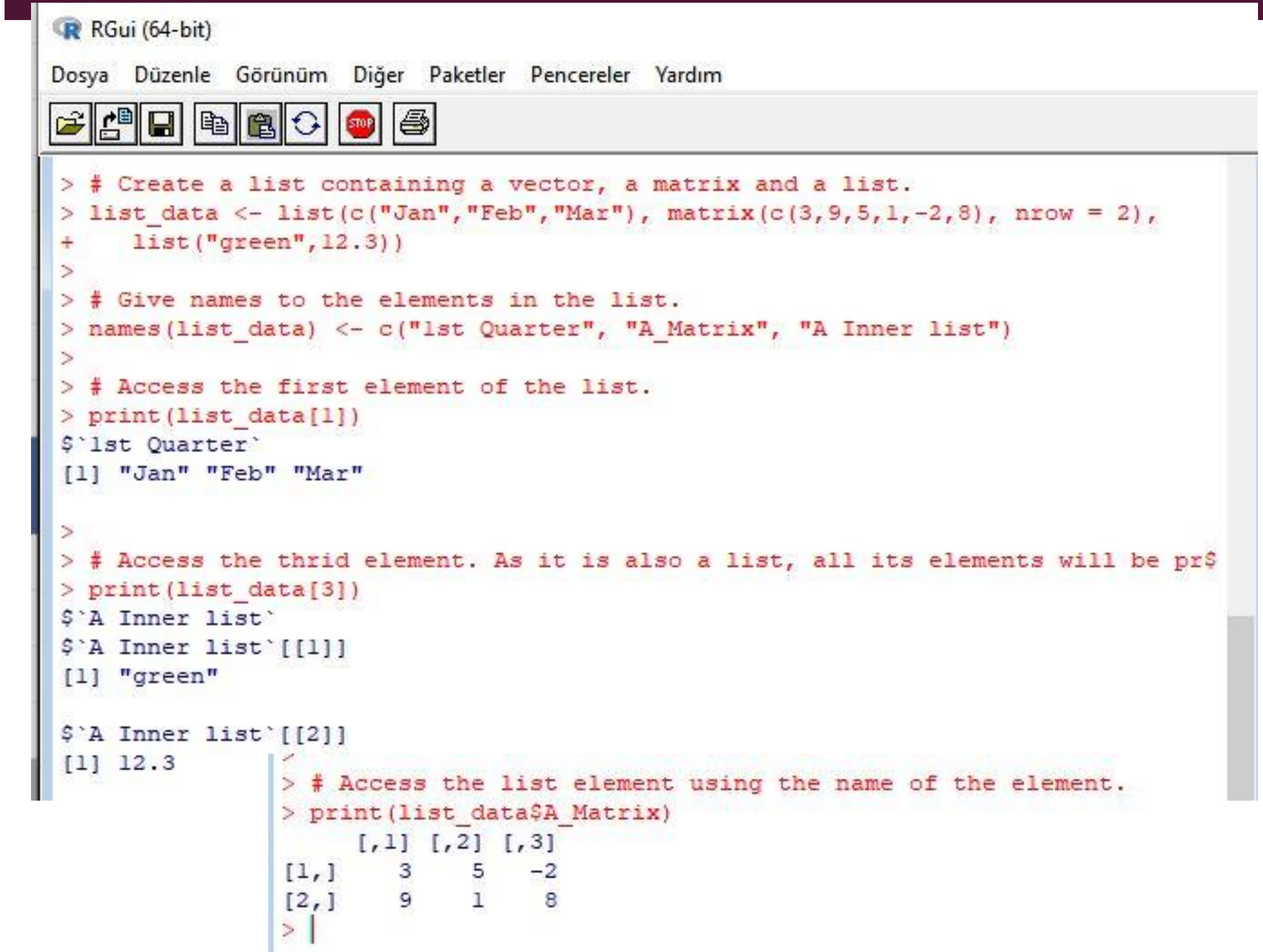


```
> # Create a list containing a vector, a matrix and a list.
> list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
+    list("green",12.3))
>
> # Give names to the elements in the list.
> names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")
>
> # Show the list.
> print(list_data)
```



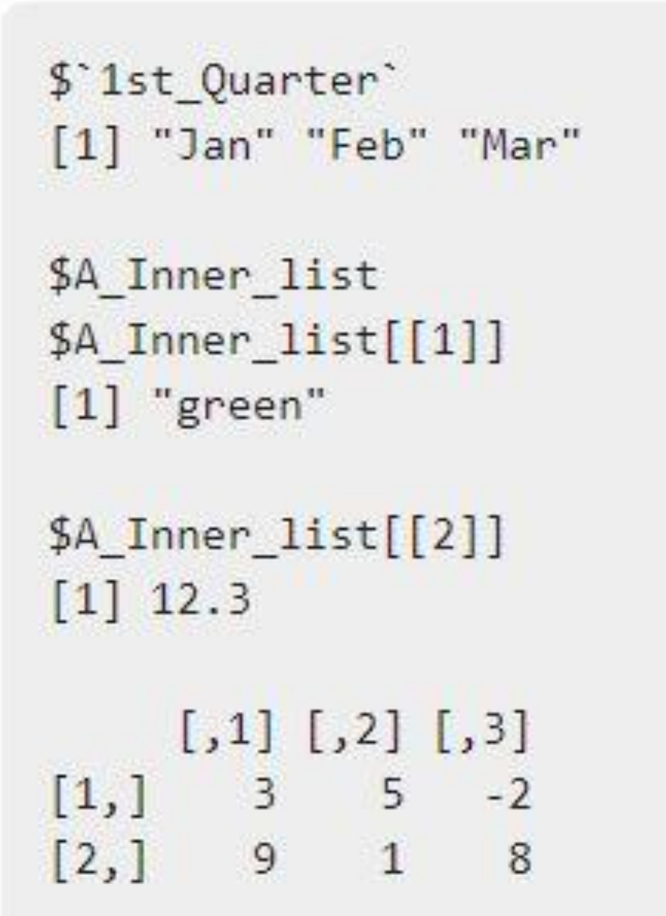
```
$`1st_Quarter`
[1] "Jan" "Feb" "Mar"

$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



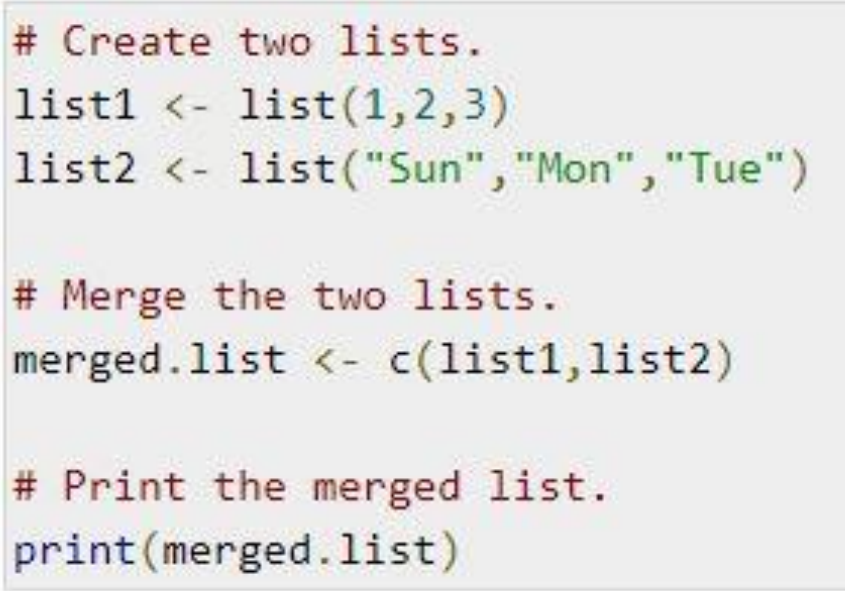
```
> # Create a list containing a vector, a matrix and a list.
> list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
+    list("green",12.3))
>
> # Give names to the elements in the list.
> names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")
>
> # Access the first element of the list.
> print(list_data[1])
$`1st Quarter`
[1] "Jan" "Feb" "Mar"

>
> # Access the thrid element. As it is also a list, all its elements will be pr$
> print(list_data[3])
$`A Inner list`
$`A Inner list`[[1]]
[1] "green"

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

```
> # Access the list element using the name of the element.
> print(list_data$A_Matrix)
     [,1] [,2] [,3]
[1,]    3    5   -2
[2,]    9    1    8
> |
```

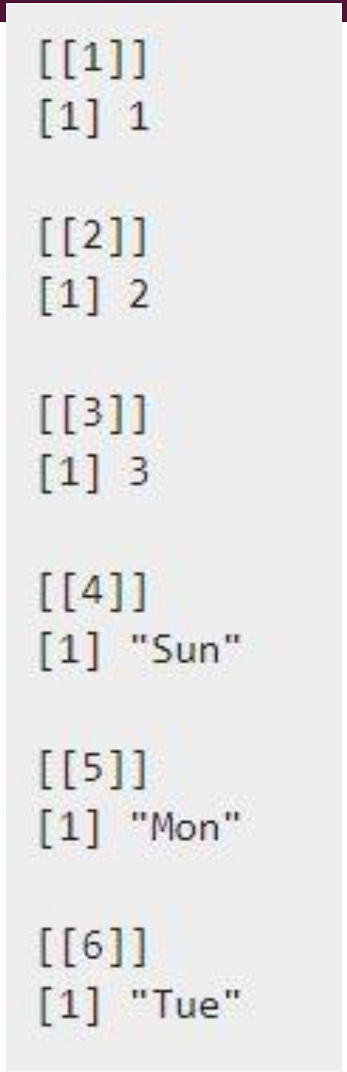


```
$`1st_Quarter`
[1] "Jan" "Feb" "Mar"

$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
```

LISTS

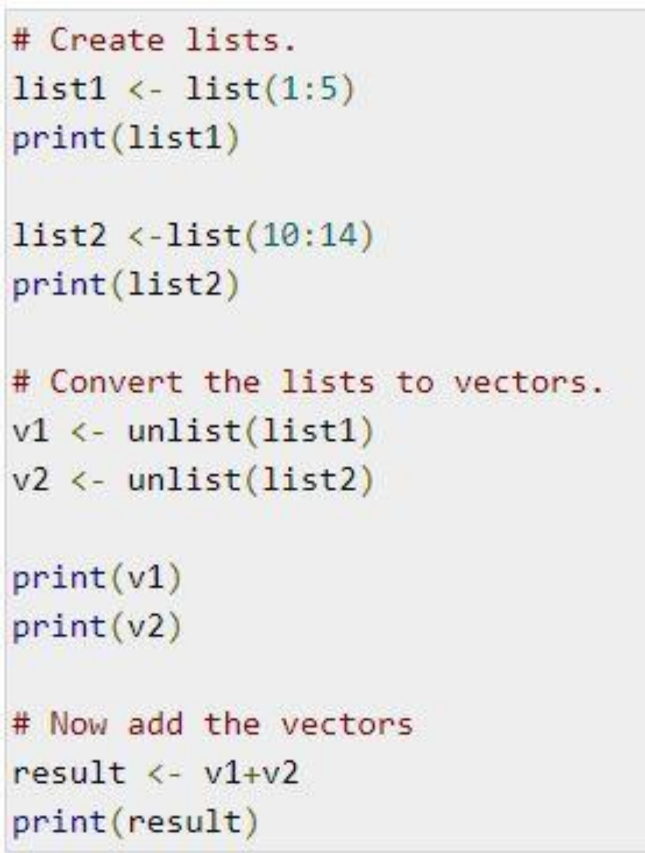


```
# Create two lists.
list1 <- list(1,2,3)
list2 <- list("Sun","Mon","Tue")

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

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

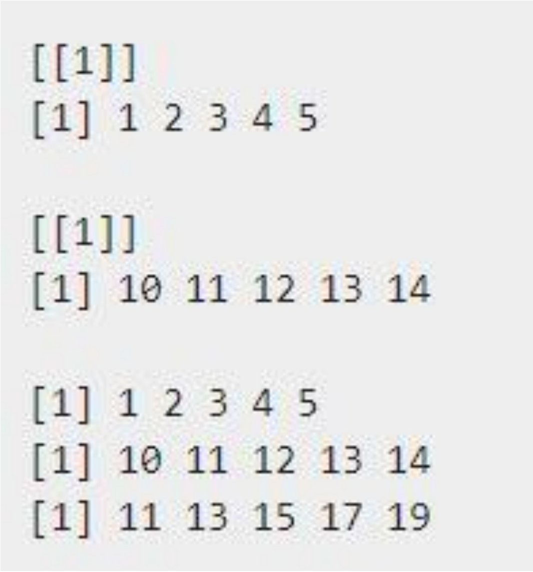


```
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

[[4]]
[1] "Sun"

[[5]]
[1] "Mon"

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

CONVERTING LIST TO VECTOR

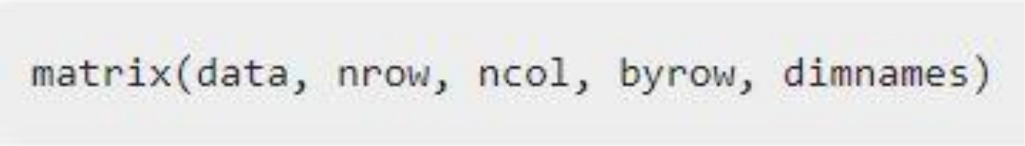


```
# Create lists.
list1 <- list(1:5)
print(list1)

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)
```

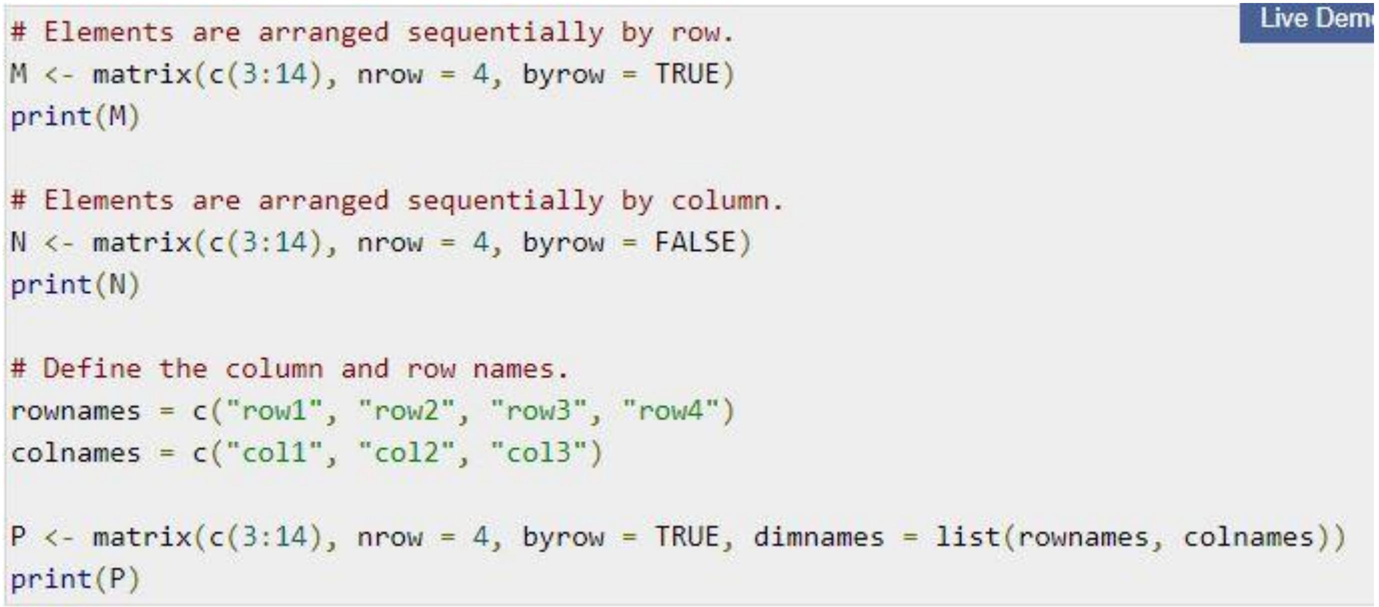


```
[[1]]
[1] 1 2 3 4 5

[[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
```

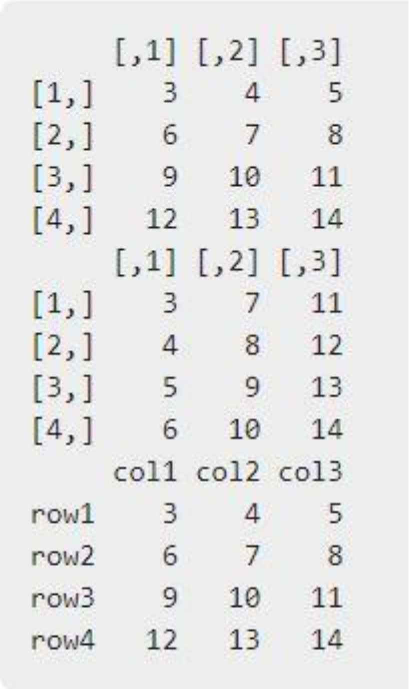
MATRIX



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

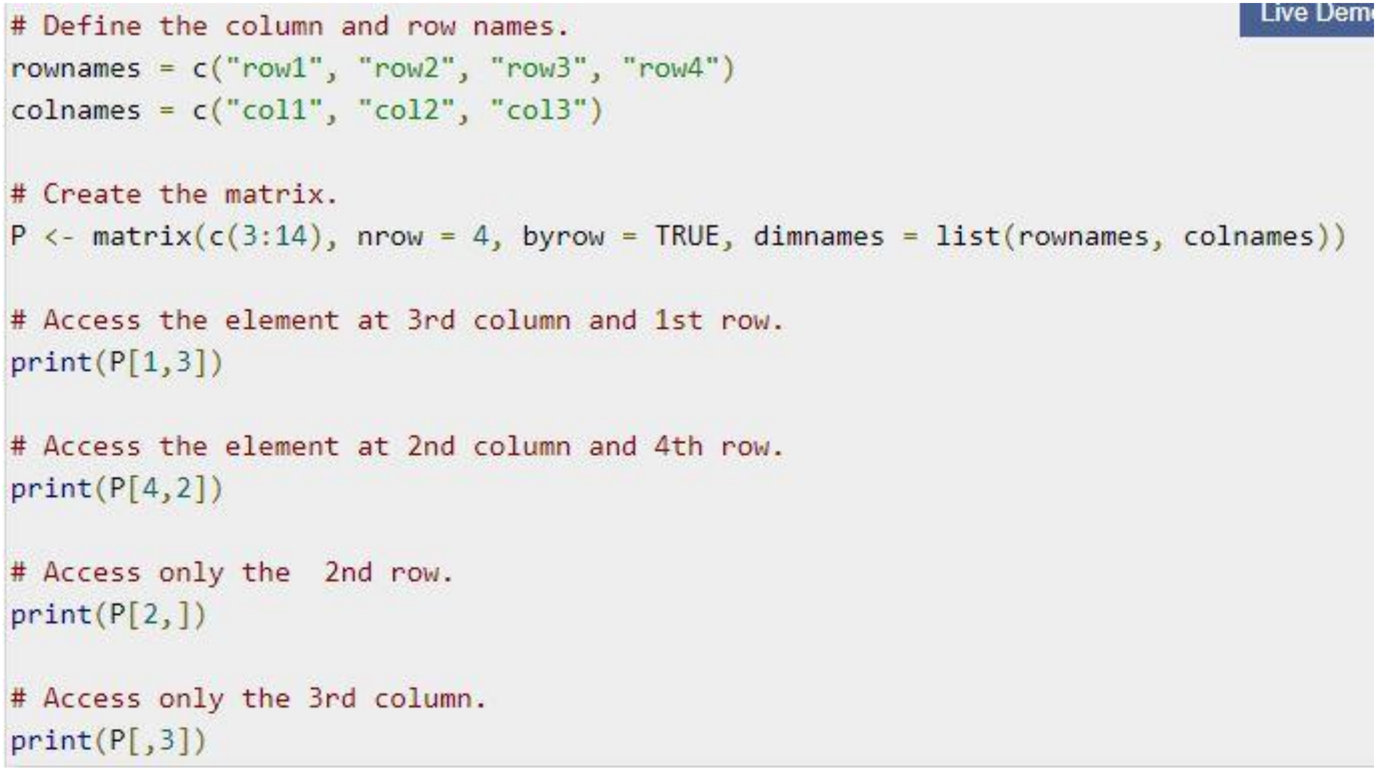


```
# Elements are arranged sequentially by row.
M <- matrix(c(3:14), nrow = 4, byrow = TRUE)
print(M)

# 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)
```

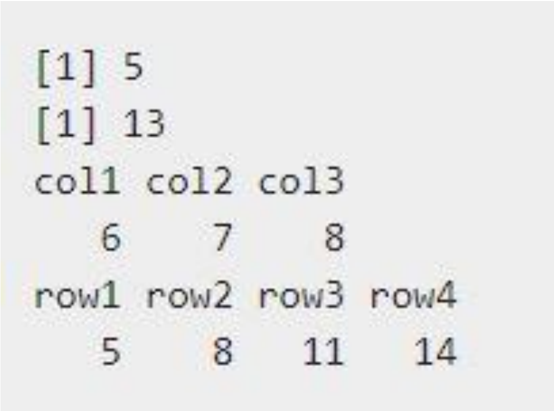


```
     [,1] [,2] [,3]
[1,]    3    4    5
[2,]    6    7    8
[3,]    9   10   11
[4,]   12   13   14
     [,1] [,2] [,3]
[1,]    3    7   11
[2,]    4    8   12
[3,]    5    9   13
[4,]    6   10   14
     col1 col2 col3
row1    3    4    5
row2    6    7    8
row3    9   10   11
row4   12   13  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

ACCESSING ELEMENTS OF A MATRIX

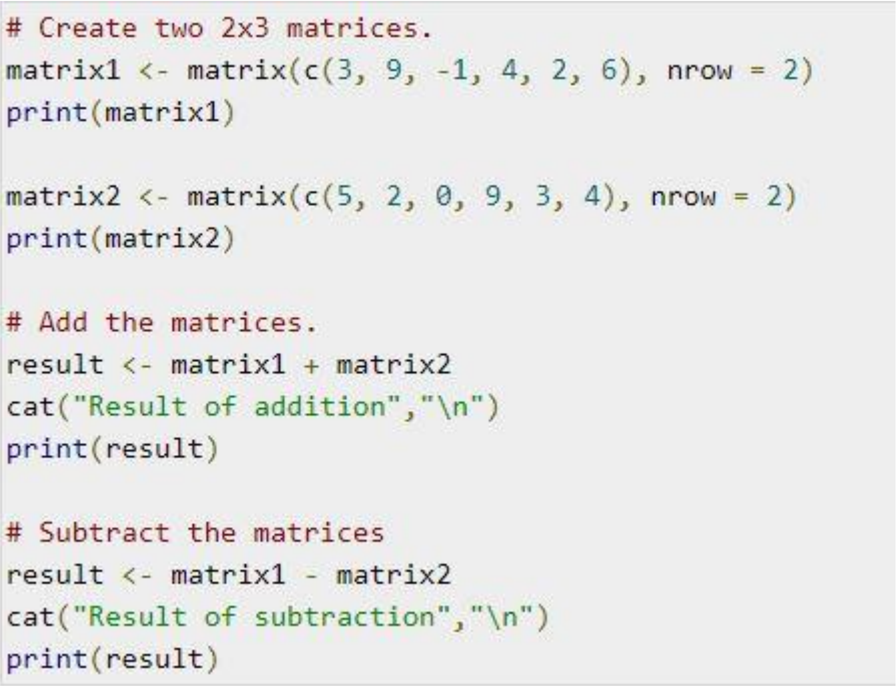


```
# Define the column and row names.
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")

# 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])
```

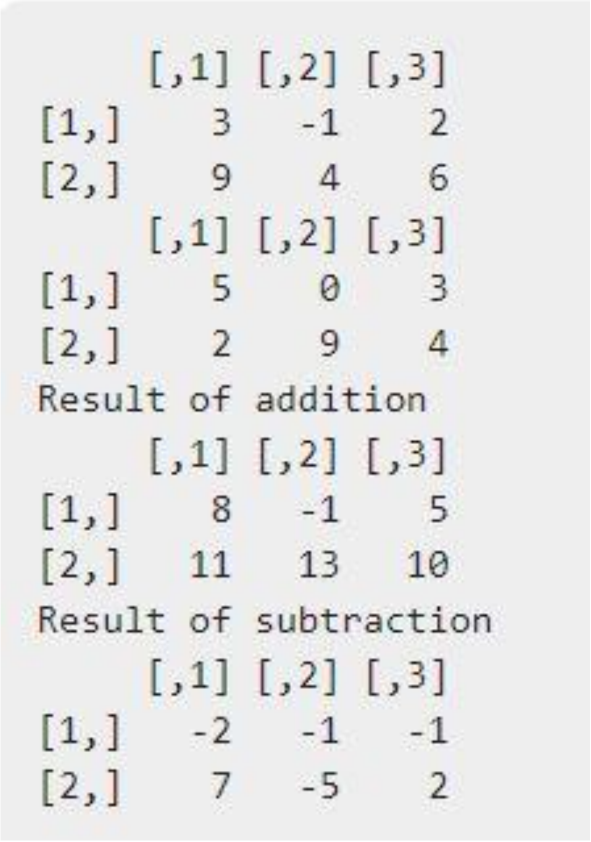


```
[1] 5
[1] 13
col1 col2 col3
   6    7    8
row1 row2 row3 row4
   5    8   11   14
```

MATRIX ADDITION & SUBTRACTION

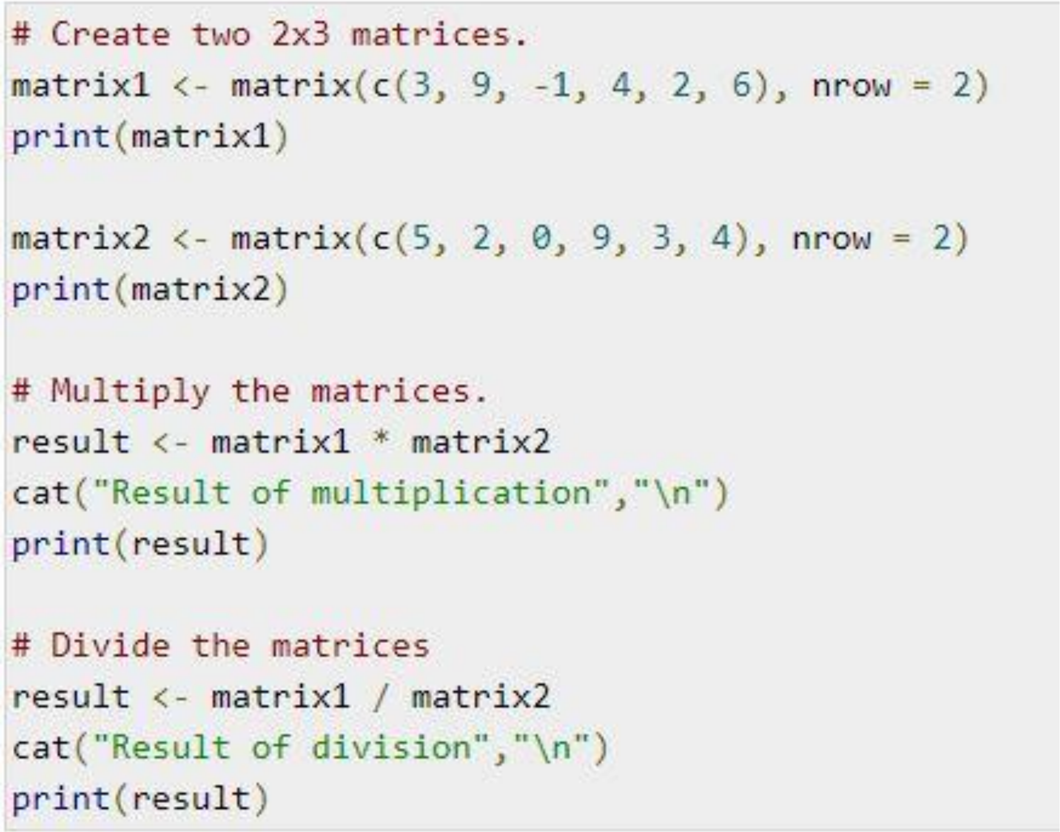


```r
# Create two 2x3 matrices.
matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
print(matrix1)

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)
```

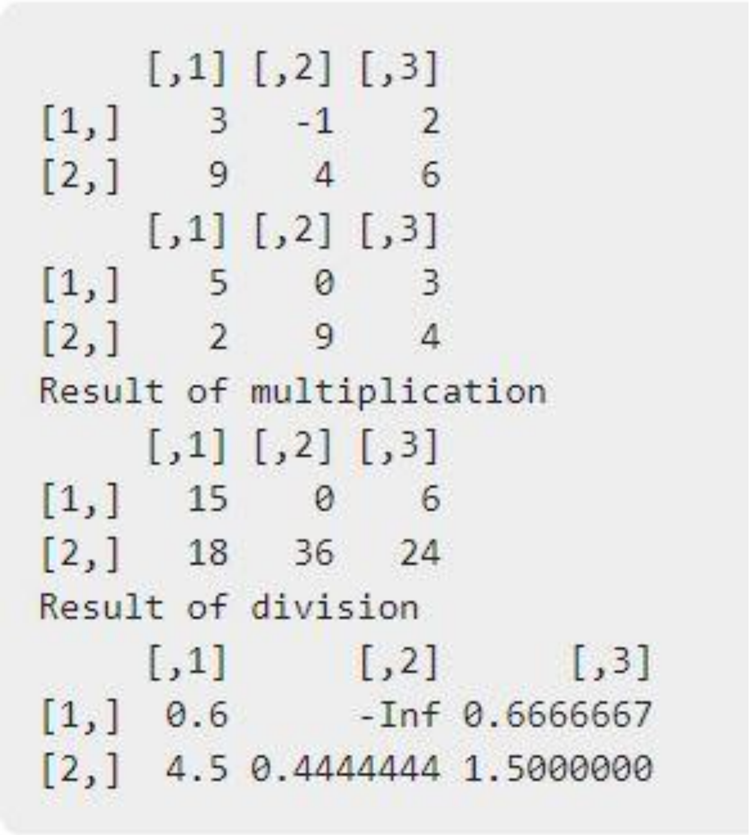


```
     [,1] [,2] [,3]
[1,]    3   -1    2
[2,]    9    4    6
     [,1] [,2] [,3]
[1,]    5    0    3
[2,]    2    9    4
Result of addition
     [,1] [,2] [,3]
[1,]    8   -1    5
[2,]   11   13   10
Result of subtraction
     [,1] [,2] [,3]
[1,]   -2   -1   -1
[2,]    7   -5    2
```

MATRIX MULTIPLICATION & DIVISION

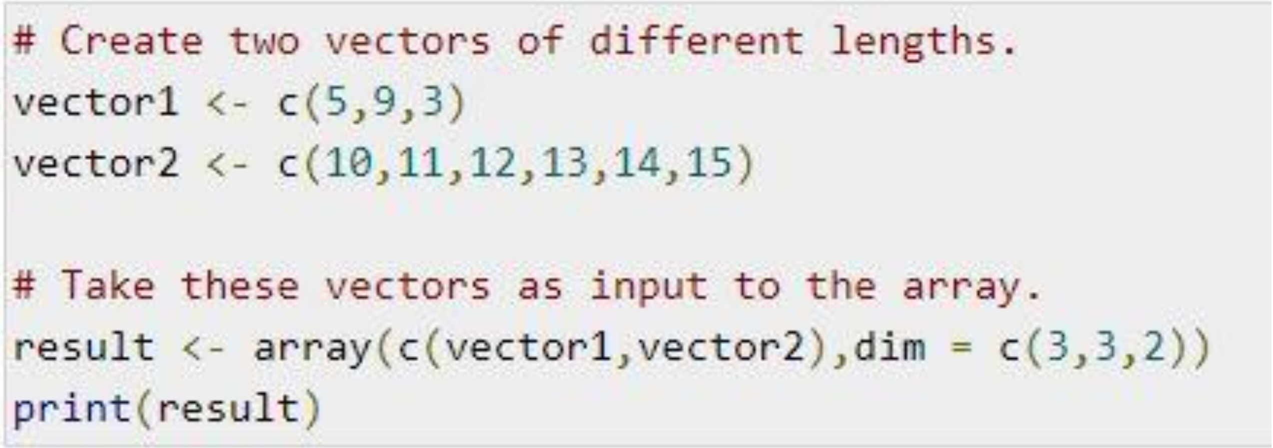


```r
# Create two 2x3 matrices.
matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
print(matrix1)

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)
```

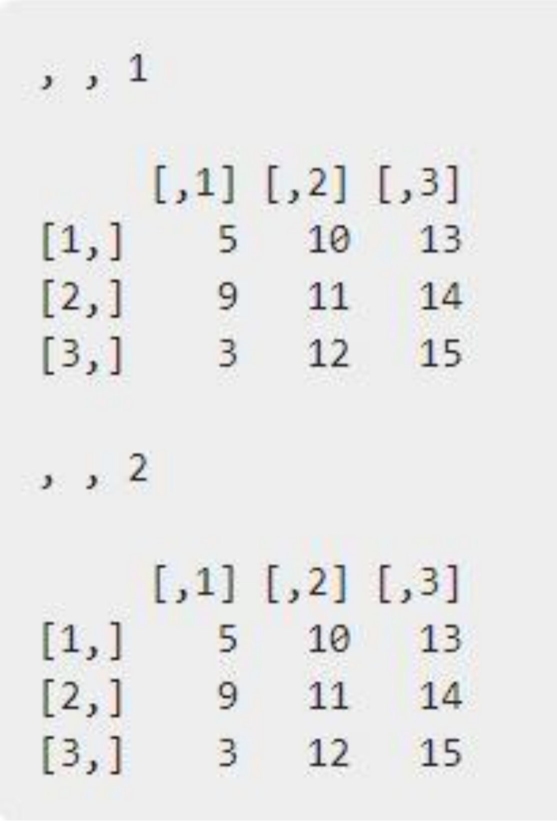


```
     [,1] [,2] [,3]
[1,]    3   -1    2
[2,]    9    4    6
     [,1] [,2] [,3]
[1,]    5    0    3
[2,]    2    9    4
Result of multiplication
     [,1] [,2] [,3]
[1,]   15    0    6
[2,]   18   36   24
Result of division
     [,1]      [,2]      [,3]
[1,]  0.6      -Inf 0.6666667
[2,]  4.5 0.4444444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

ARRAYS

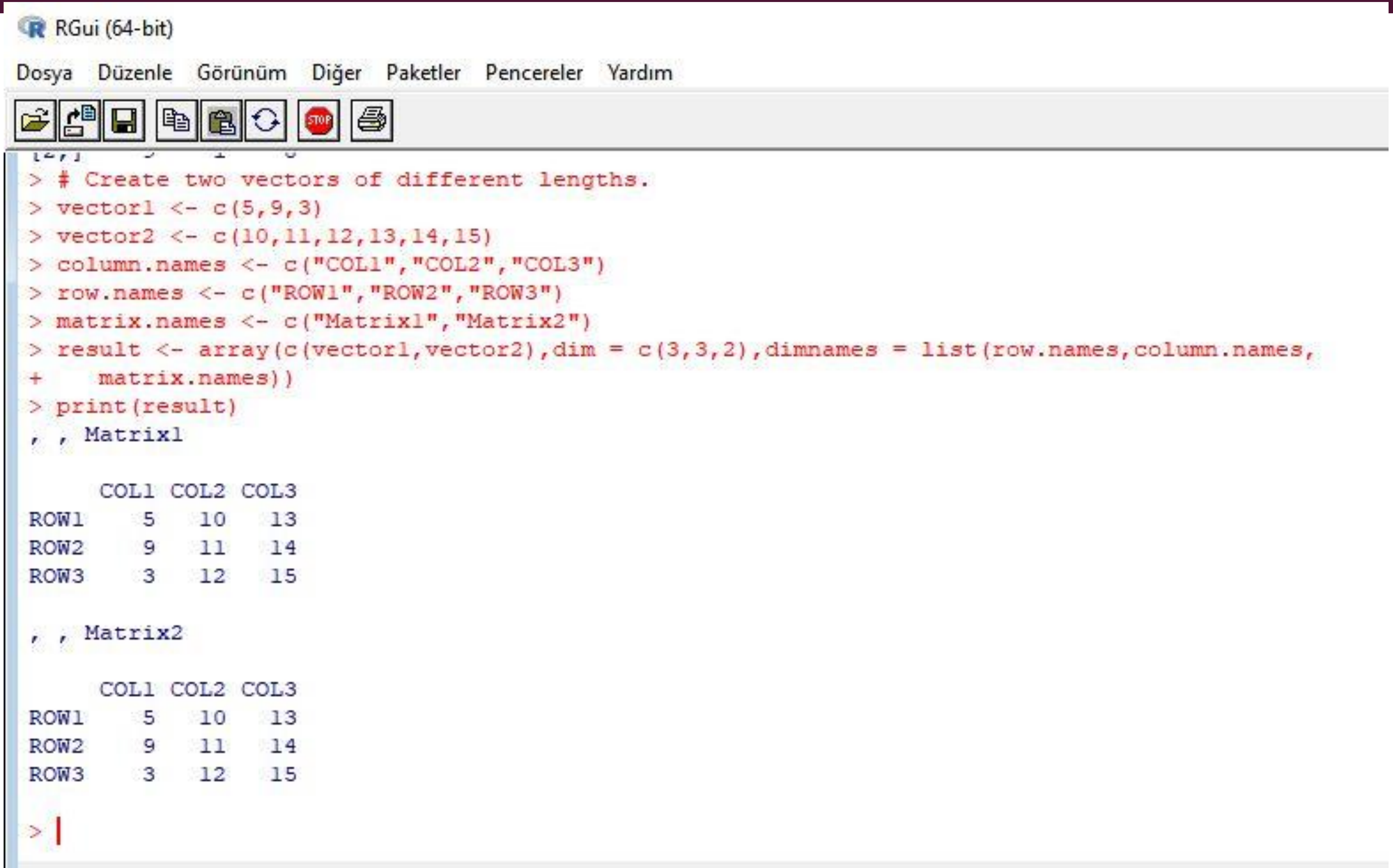


```
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)

# Take these vectors as input to the array.
result <- array(c(vector1,vector2),dim = c(3,3,2))
print(result)
```



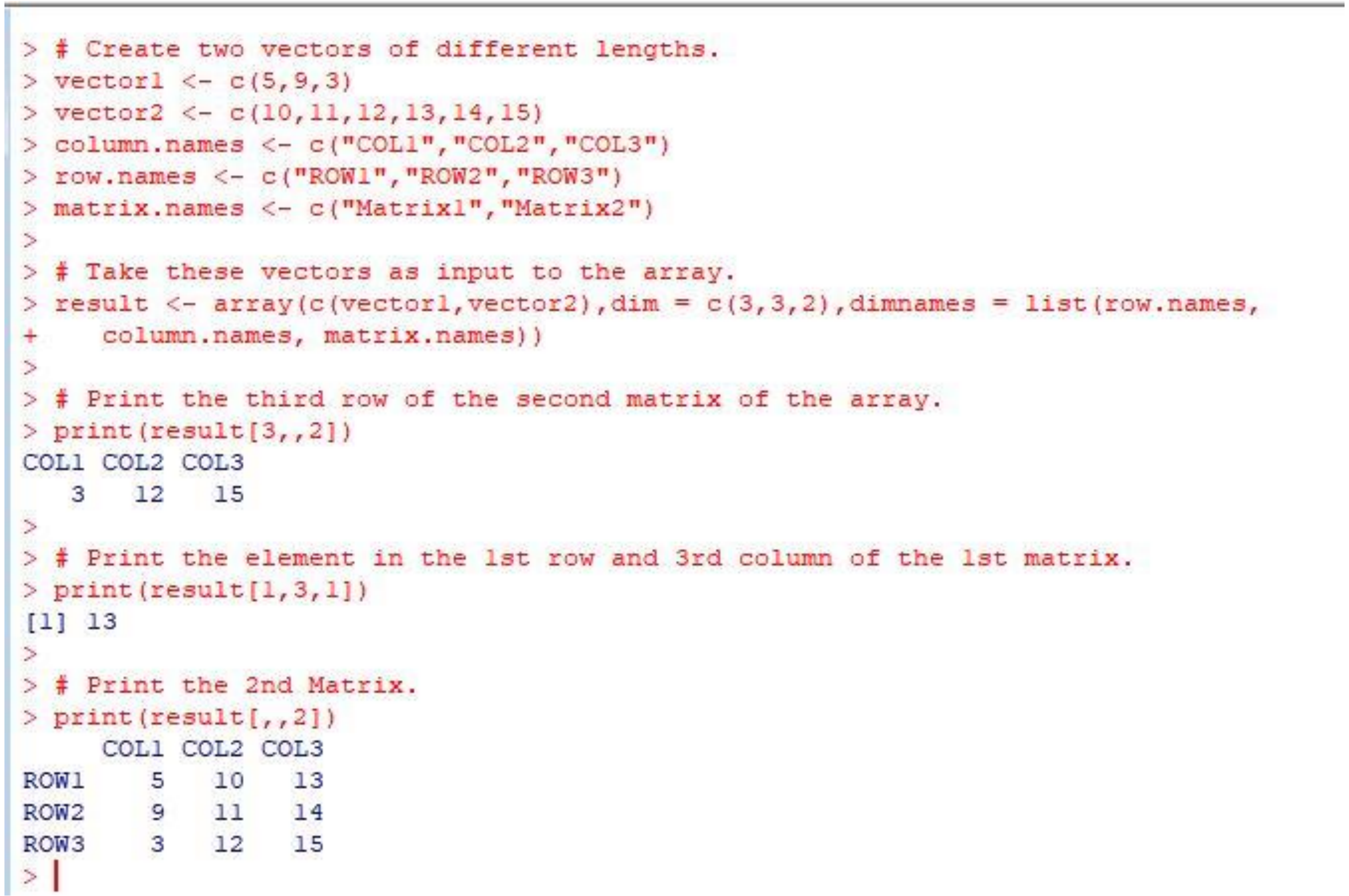
```
, , 1

     [,1] [,2] [,3]
[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15

, , 2

     [,1] [,2] [,3]
[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15
```

NAMING COLUMNS AND ROWS



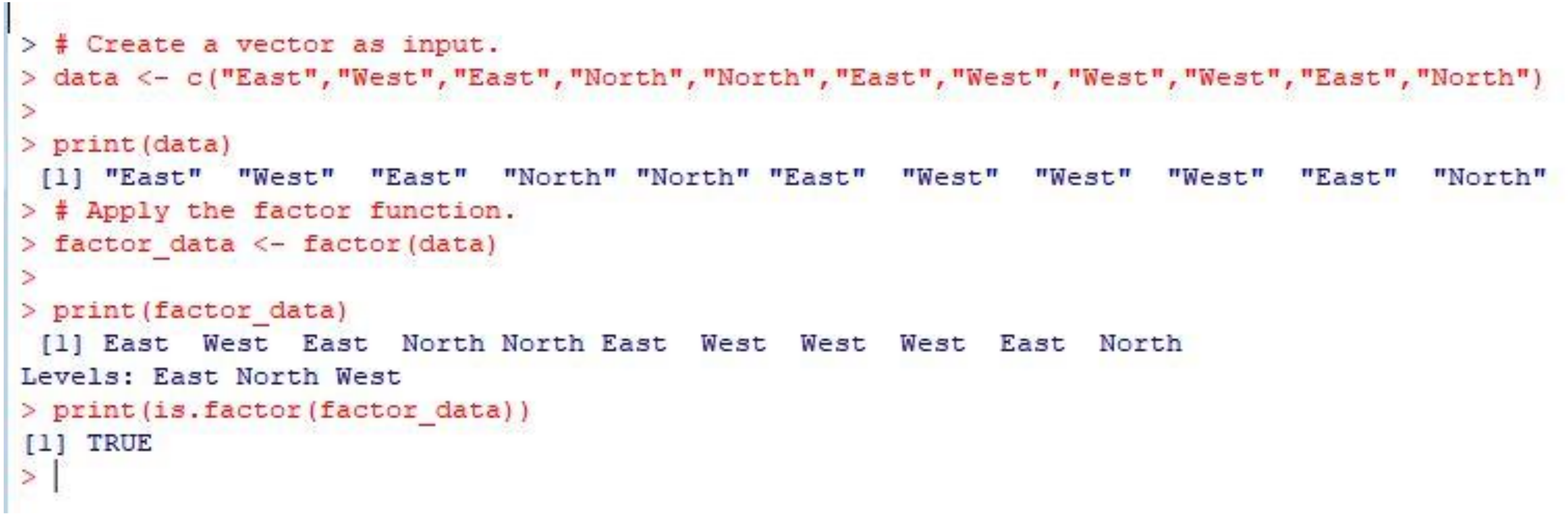
```
> # Create two vectors of different lengths.
> vector1 <- c(5,9,3)
> vector2 <- c(10,11,12,13,14,15)
> column.names <- c("COL1","COL2","COL3")
> row.names <- c("ROW1","ROW2","ROW3")
> matrix.names <- c("Matrix1","Matrix2")
> result <- array(c(vector1,vector2),dim = c(3,3,2),dimnames = list(row.names,column.names,
+    matrix.names))
> print(result)
, , Matrix1

     COL1 COL2 COL3
ROW1    5   10   13
ROW2    9   11   14
ROW3    3   12   15

, , Matrix2

     COL1 COL2 COL3
ROW1    5   10   13
ROW2    9   11   14
ROW3    3   12   15

> 
```

ACCESSING ARRAY ELEMENTS



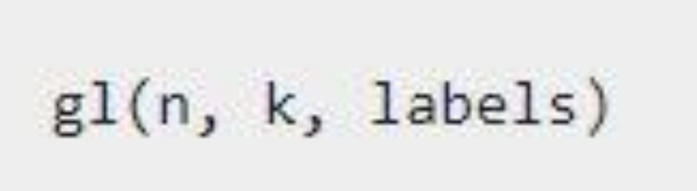
```
> # Create two vectors of different lengths.
> vector1 <- c(5,9,3)
> vector2 <- c(10,11,12,13,14,15)
> column.names <- c("COL1","COL2","COL3")
> row.names <- c("ROW1","ROW2","ROW3")
> matrix.names <- c("Matrix1","Matrix2")
>
> # Take these vectors as input to the array.
> result <- array(c(vector1,vector2),dim = c(3,3,2),dimnames = list(row.names,
+    column.names, matrix.names))
>
> # Print the third row of the second matrix of the array.
> print(result[3,,2])
COL1 COL2 COL3
   3   12   15
>
> # Print the element in the 1st row and 3rd column of the 1st matrix.
> print(result[1,3,1])
[1] 13
>
> # Print the 2nd Matrix.
> print(result[,,2])
     COL1 COL2 COL3
ROW1    5   10   13
ROW2    9   11   14
ROW3    3   12   15
> |
```

FACTORS

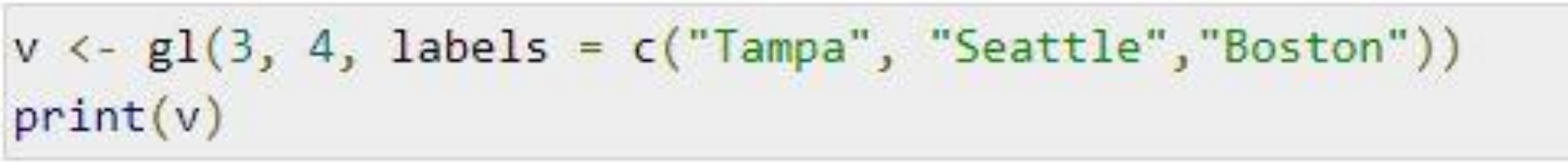


```
> # Create a vector as input.
> data <- c("East","West","East","North","North","East","West","West","West","East","North")
>
> print(data)
 [1] "East"  "West"  "East"  "North" "North" "East"  "West"  "West"  "West"  "East"  "North"
> # Apply the factor function.
> factor_data <- factor(data)
>
> print(factor_data)
 [1] East  West  East  North North East  West  West  West  East  North
Levels: East North West
> print(is.factor(factor_data))
[1] TRUE
> 
```

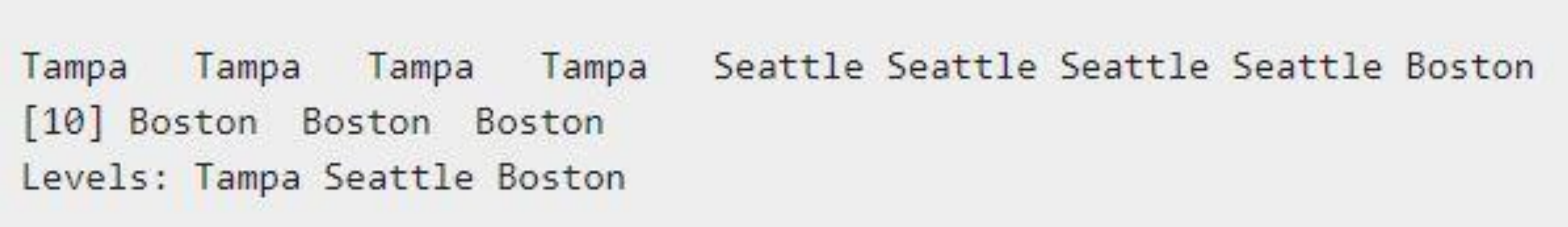
GENERATING FACTOR LEVELS



```
gl(n, k, labels)
```

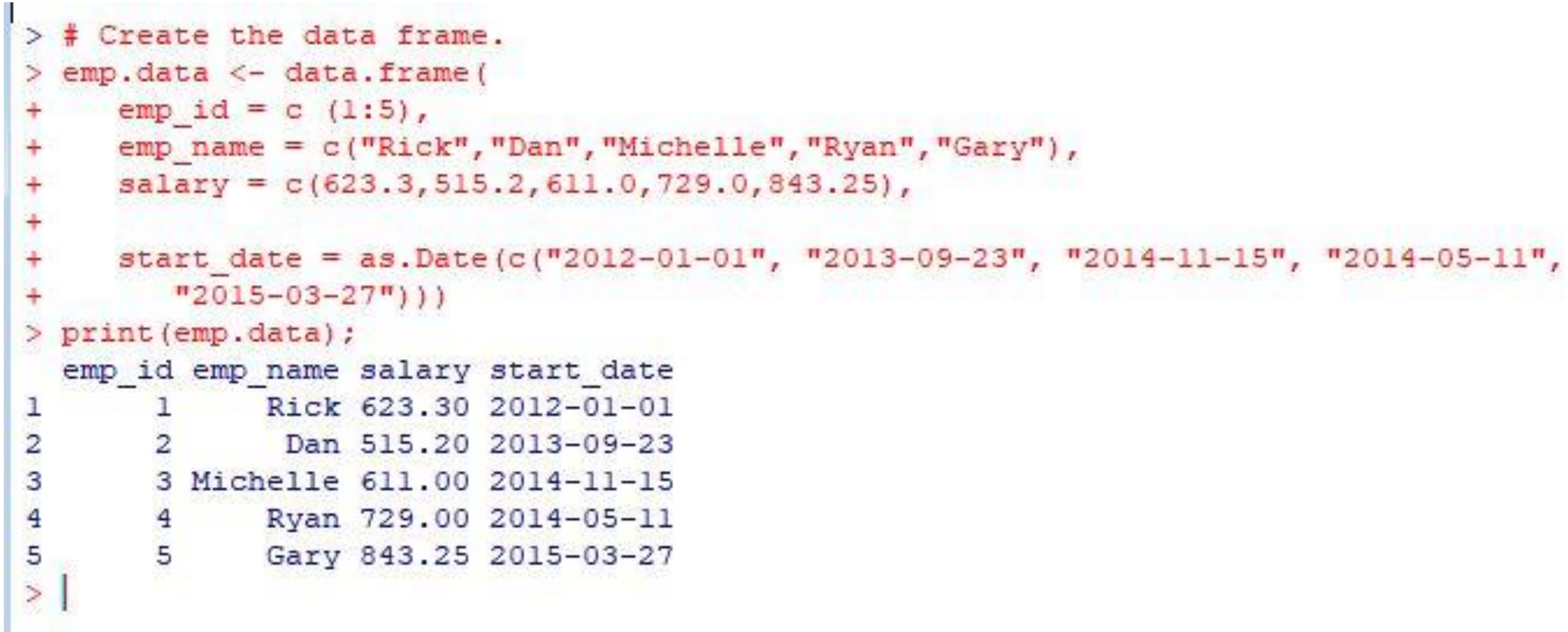


```
v <- gl(3, 4, labels = c("Tampa", "Seattle","Boston"))
print(v)
```



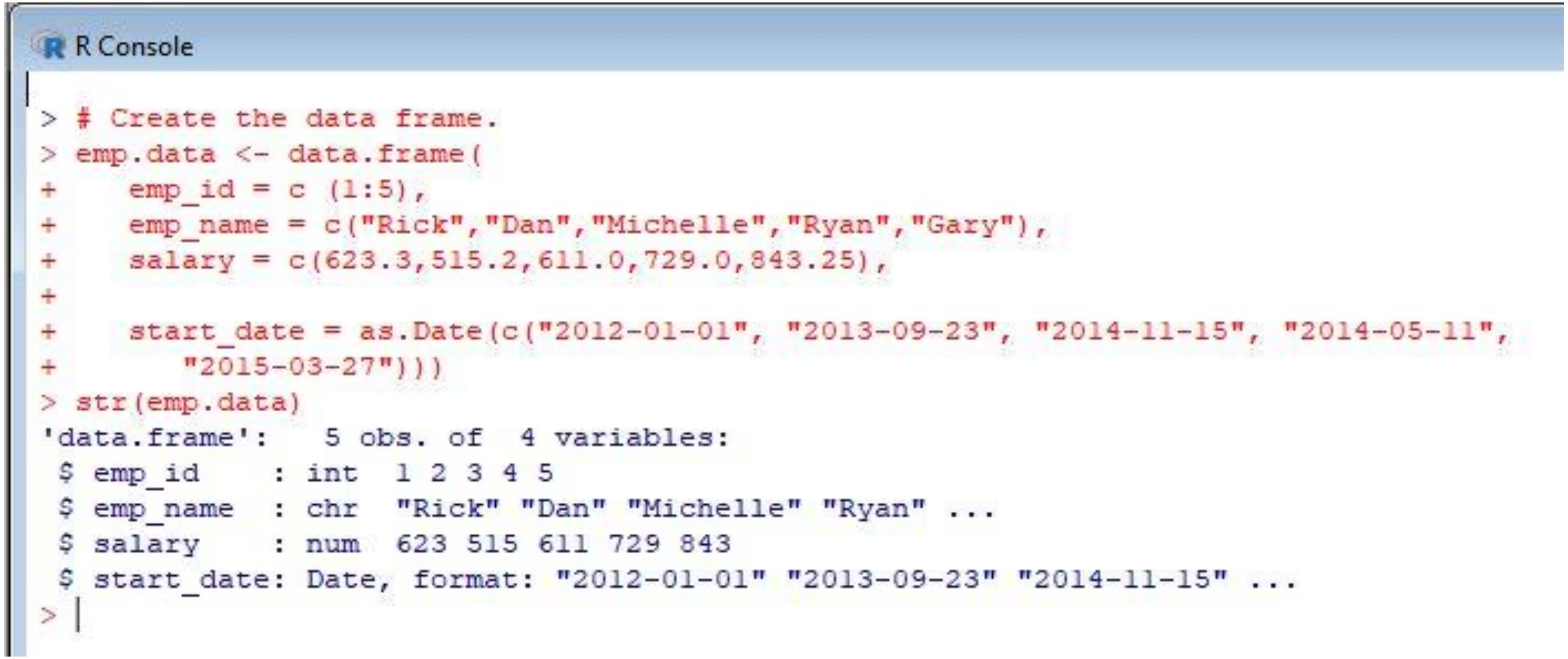
```
Tampa   Tampa   Tampa   Tampa   Seattle Seattle Seattle Seattle Boston
[10] Boston  Boston  Boston
Levels: Tampa Seattle Boston
```

DATA FRAMES



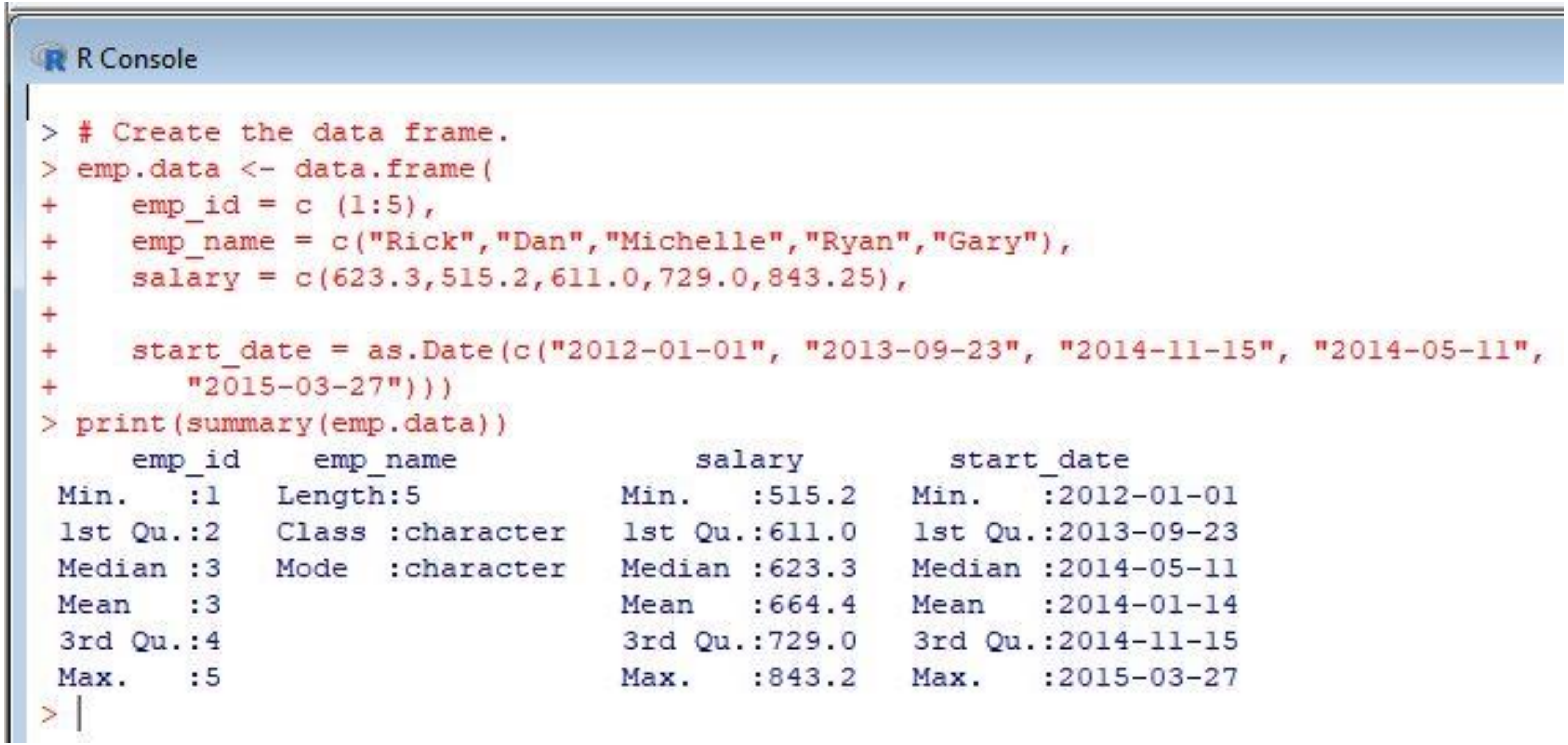
```
> # Create the data frame.
> emp.data <- data.frame(
+    emp_id = c (1:5),
+    emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
+    salary = c(623.3,515.2,611.0,729.0,843.25),
+
+    start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
+       "2015-03-27")))
> print(emp.data);
  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
> 
```

GET THE STRUCTURE OF THE DATA FRAME



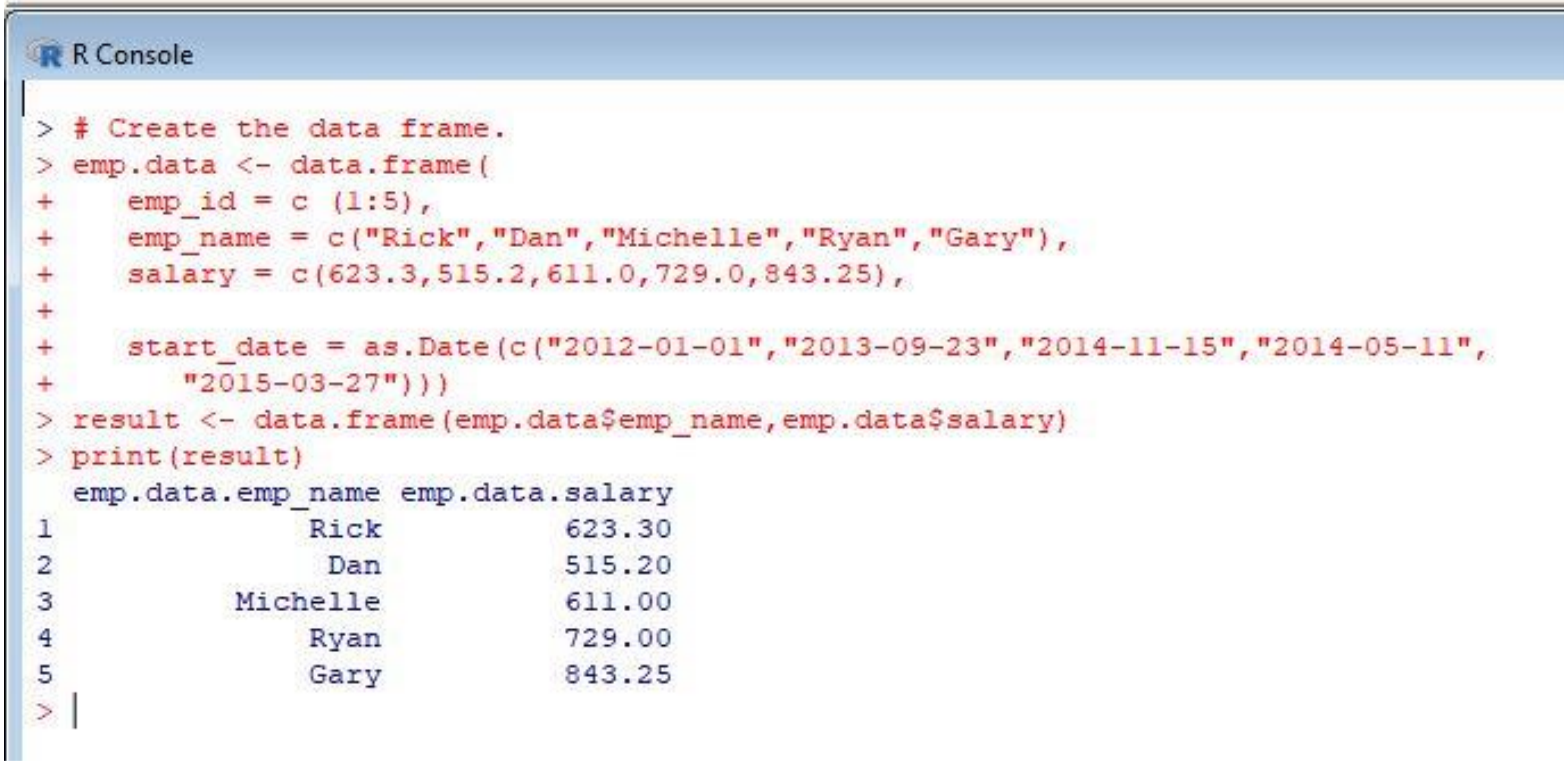
```
> # Create the data frame.
> emp.data <- data.frame(
+    emp_id = c (1:5),
+    emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
+    salary = c(623.3,515.2,611.0,729.0,843.25),
+
+    start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
+       "2015-03-27")))
> str(emp.data)
'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" ...
> |
```

SUMMARY OF DATA IN DATA FRAME



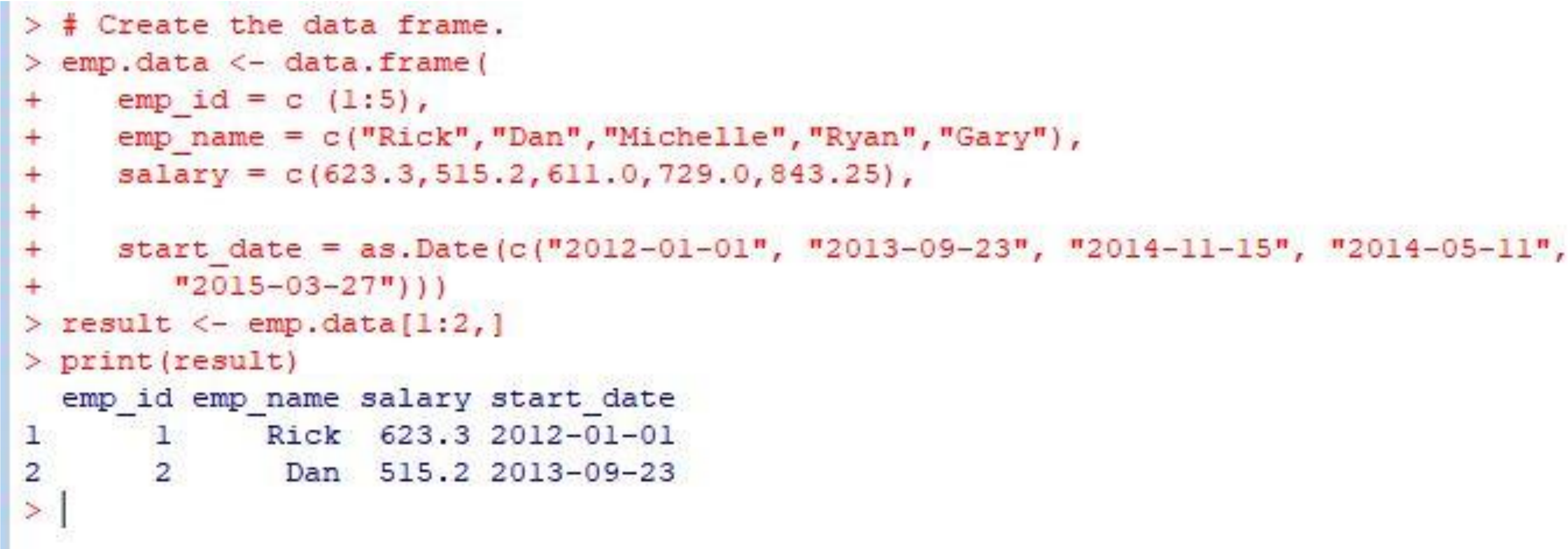
```
> # Create the data frame.
> emp.data <- data.frame(
+    emp_id = c (1:5),
+    emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
+    salary = c(623.3,515.2,611.0,729.0,843.25),
+
+    start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
+       "2015-03-27")))
> print(summary(emp.data))
     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



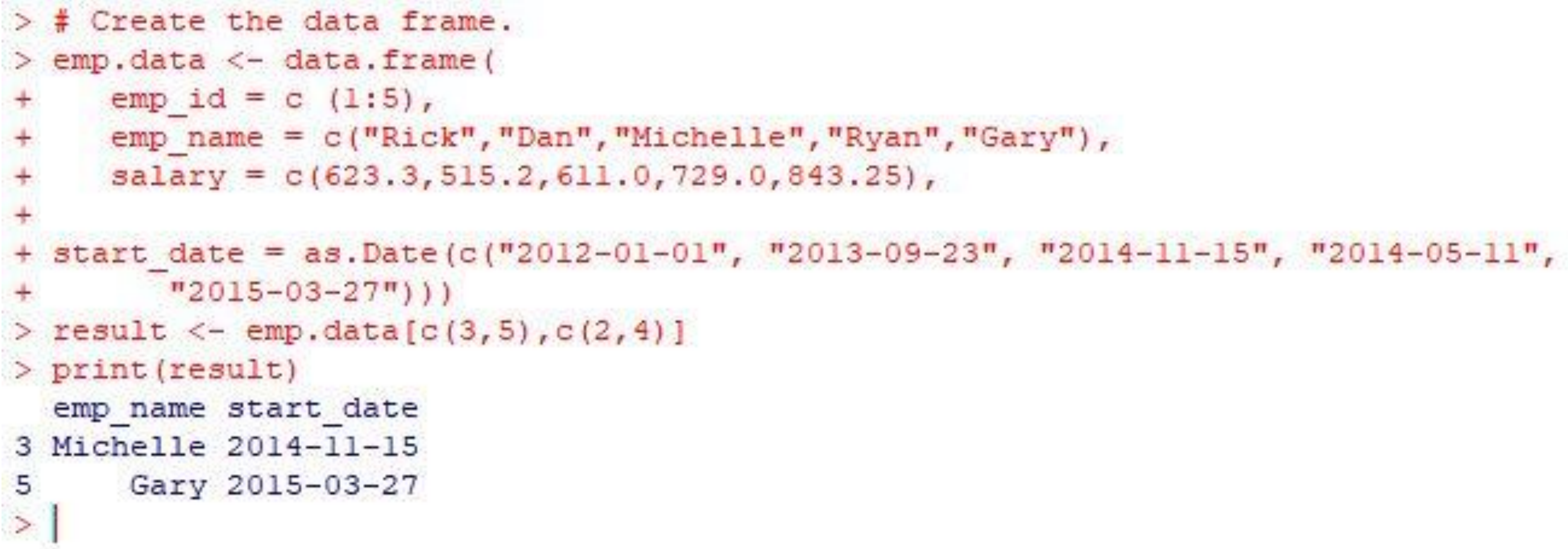
```
> # Create the data frame.
> emp.data <- data.frame(
+    emp_id = c (1:5),
+    emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
+    salary = c(623.3,515.2,611.0,729.0,843.25),
+
+    start_date = as.Date(c("2012-01-01","2013-09-23","2014-11-15","2014-05-11",
+       "2015-03-27")))
> result <- data.frame(emp.data$emp_name,emp.data$salary)
> print(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 DATA FROM DATA FRAME



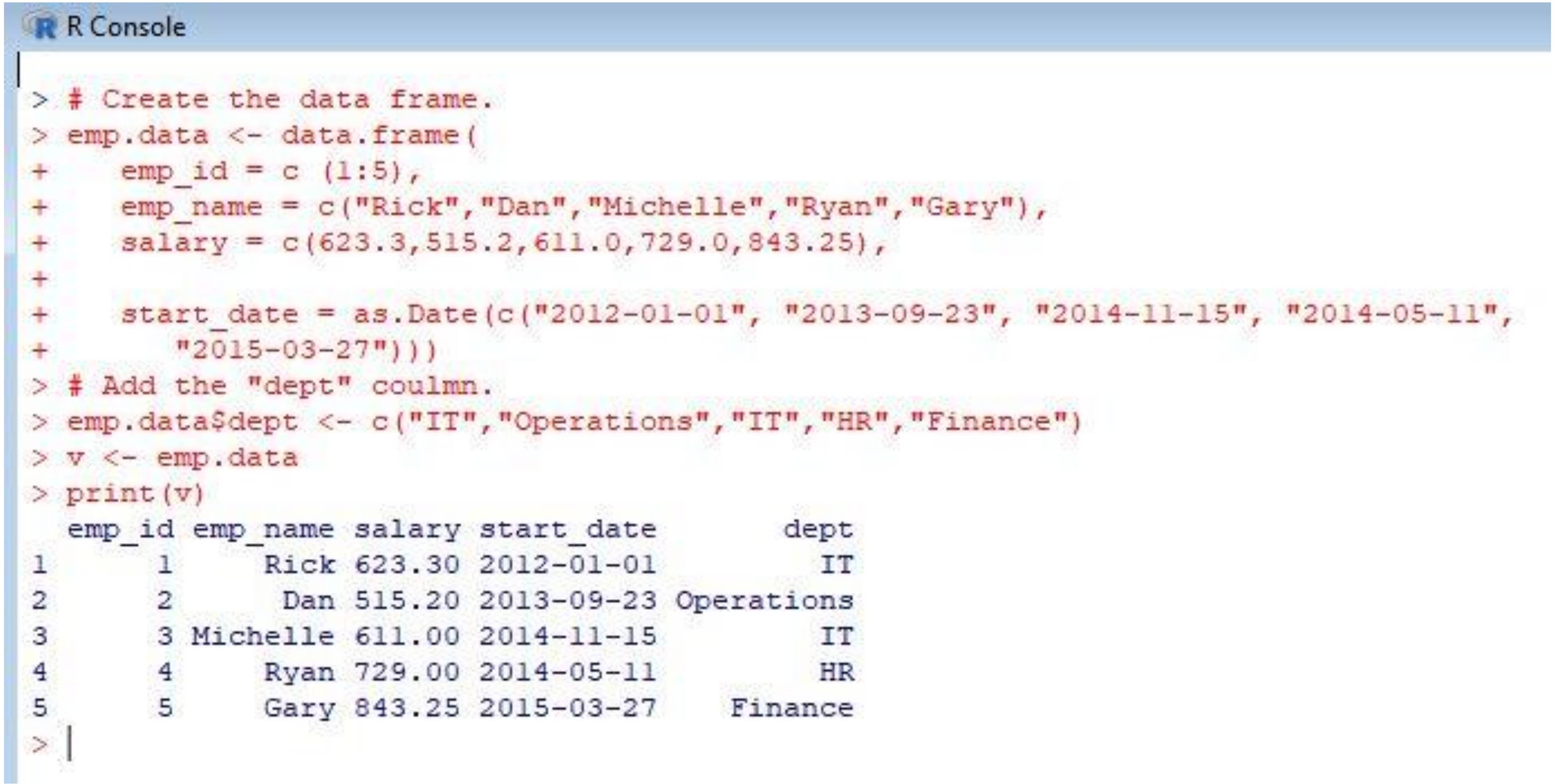
```
> # Create the data frame.
> emp.data <- data.frame(
+    emp_id = c (1:5),
+    emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
+    salary = c(623.3,515.2,611.0,729.0,843.25),
+
+    start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
+       "2015-03-27")))
> result <- emp.data[1:2,]
> print(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 DATA FROM DATA FRAME



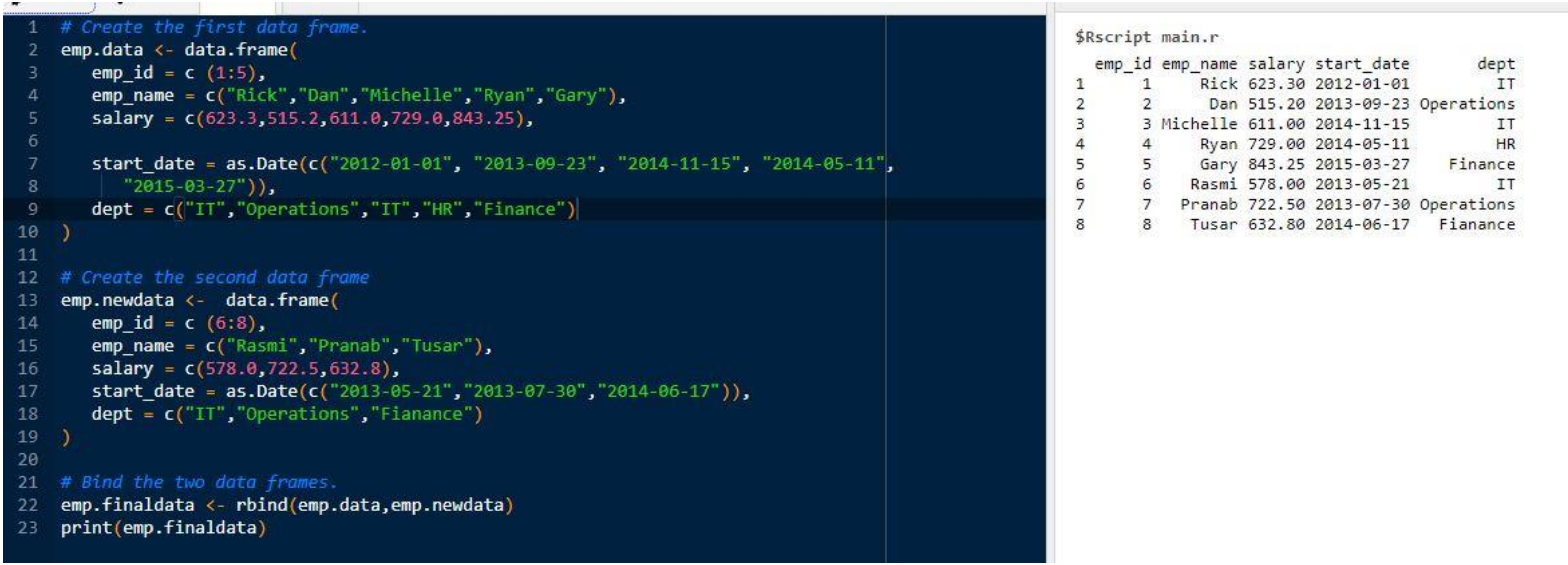
```
> # Create the data frame.
> emp.data <- data.frame(
+    emp_id = c (1:5),
+    emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
+    salary = c(623.3,515.2,611.0,729.0,843.25),
+
+ start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
+       "2015-03-27")))
> result <- emp.data[c(3,5),c(2,4)]
> print(result)
  emp_name start_date
3 Michelle 2014-11-15
5     Gary 2015-03-27
> |
```

EXPAND DATA FRAME-ADD COLUMN



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

EXPAND DATA FRAME-ADD ROW

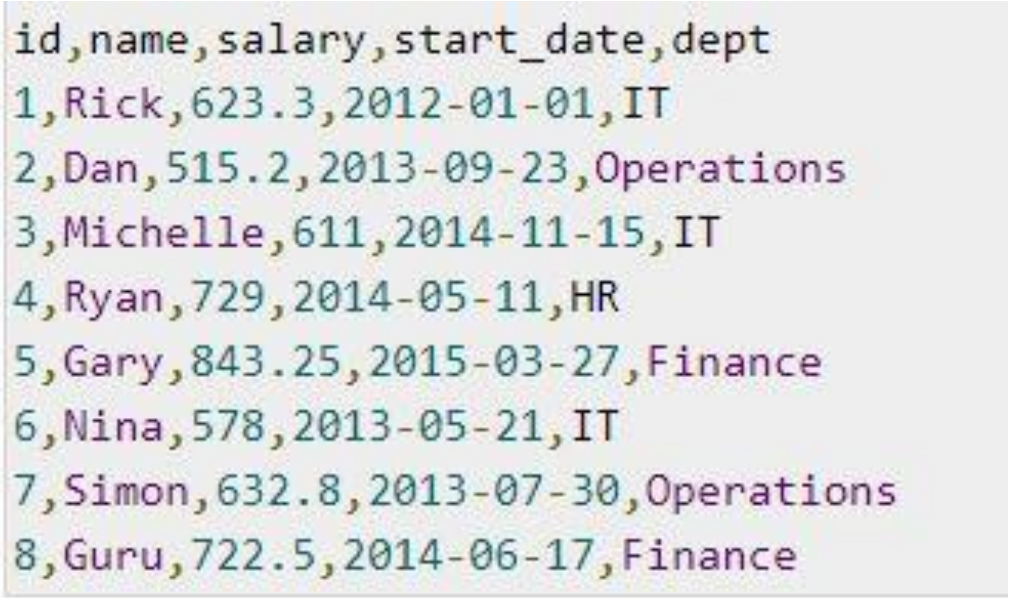


```r
# Create the first data frame.
emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),

   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")
)

# 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")
)

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

```
$Rscript main.r

  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
```

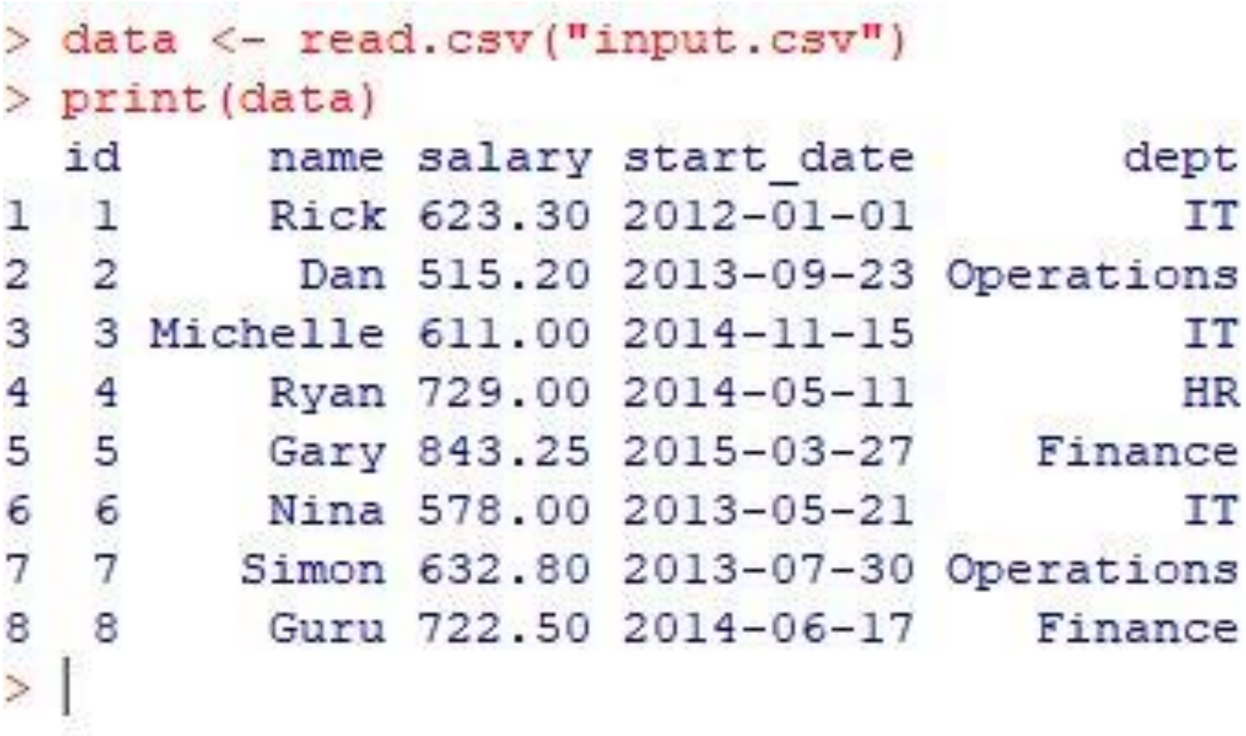
R DATA INTERFACES

- **Getting and Setting the Working Directory :** You can check which directory the R workspace is pointing to using the getwd() function.



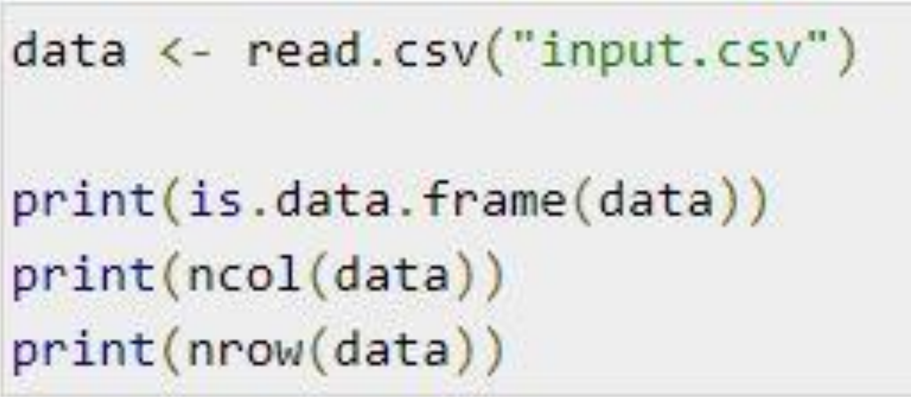
```
id,name,salary,start_date,dept
1,Rick,623.3,2012-01-01,IT
2,Dan,515.2,2013-09-23,Operations
3,Michelle,611,2014-11-15,IT
4,Ryan,729,2014-05-11,HR
5,Gary,843.25,2015-03-27,Finance
6,Nina,578,2013-05-21,IT
7,Simon,632.8,2013-07-30,Operations
8,Guru,722.5,2014-06-17,Finance
```

READING A CSV FILE

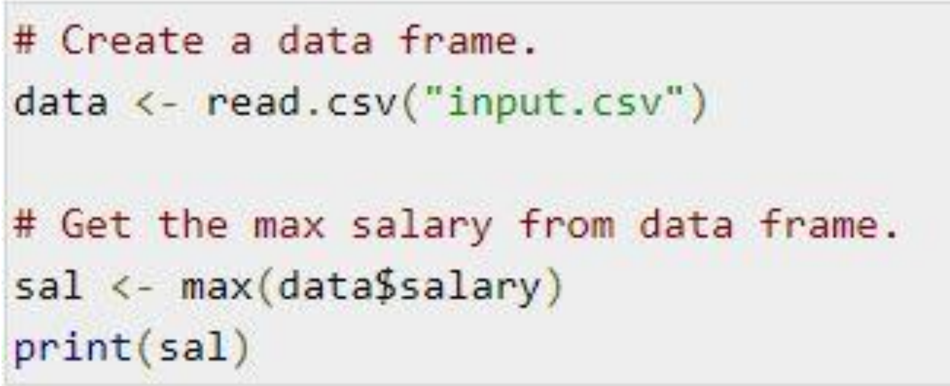


```
> data <- read.csv("input.csv")
> print(data)
  id     name salary start_date       dept
1  1     Rick 623.30 2012-01-01         IT
2  2      Dan 515.20 2013-09-23 Operations
3  3 Michelle 611.00 2014-11-15         IT
4  4     Ryan 729.00 2014-05-11         HR
5  5     Gary 843.25 2015-03-27    Finance
6  6     Nina 578.00 2013-05-21         IT
7  7    Simon 632.80 2013-07-30 Operations
8  8     Guru 722.50 2014-06-17    Finance
> |
```

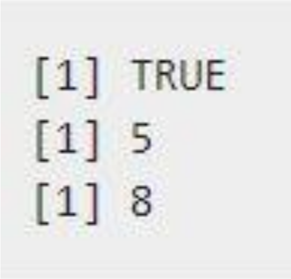
ANALYSIS THE CSV FILE



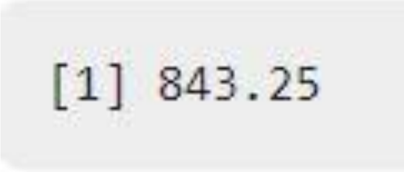
```
data <- read.csv("input.csv")

print(is.data.frame(data))
print(ncol(data))
print(nrow(data))
```



```
# Create a data frame.
data <- read.csv("input.csv")

# Get the max salary from data frame.
sal <- max(data$salary)
print(sal)
```

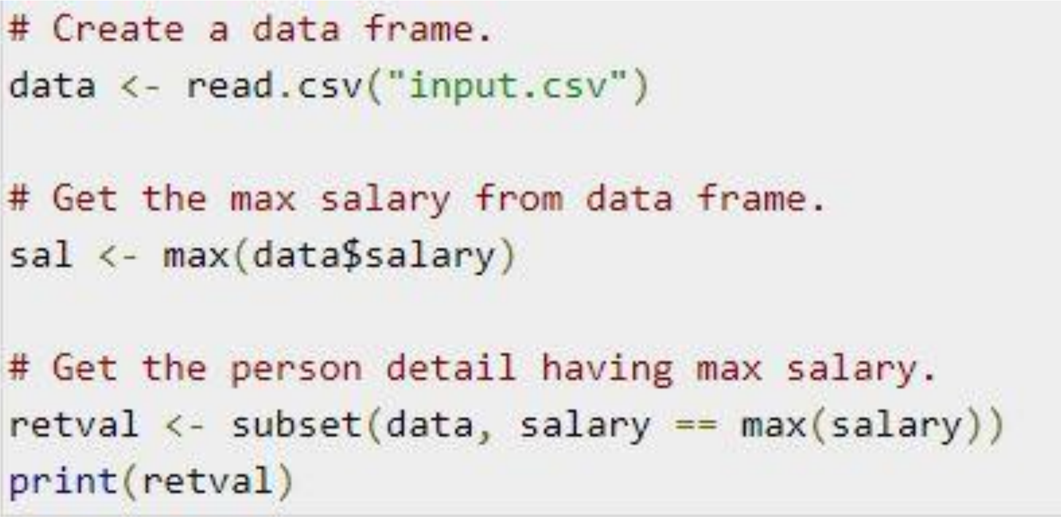


```
[1] TRUE
[1] 5
[1] 8
```

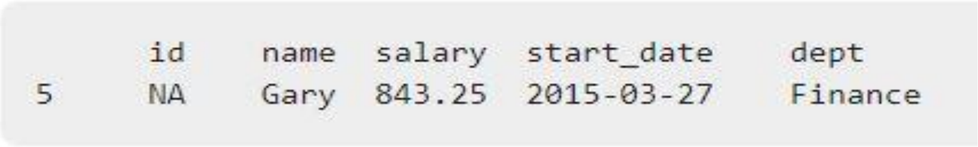


```
[1] 843.25
```

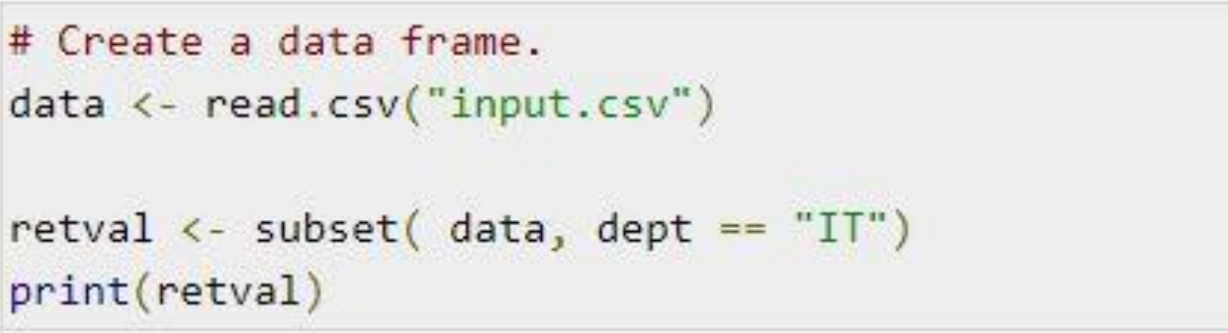
ANALYSIS THE CSV FILE



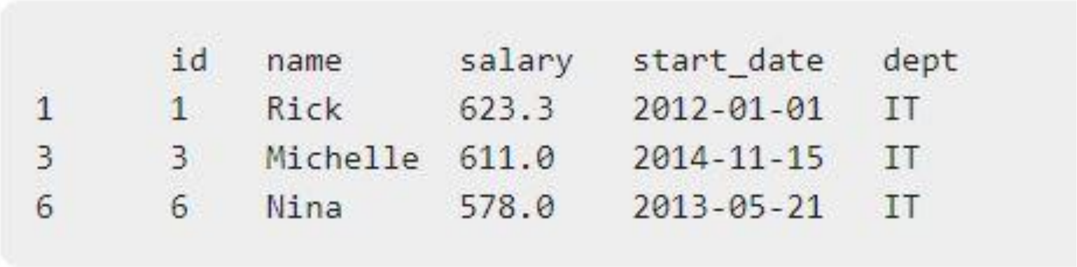
```
# Create a data frame.
data <- read.csv("input.csv")

# Get the max salary from data frame.
sal <- max(data$salary)

# Get the person detail having max salary.
retval <- subset(data, salary == max(salary))
print(retval)
```



```
      id    name   salary   start_date    dept
5     NA    Gary   843.25   2015-03-27    Finance
```

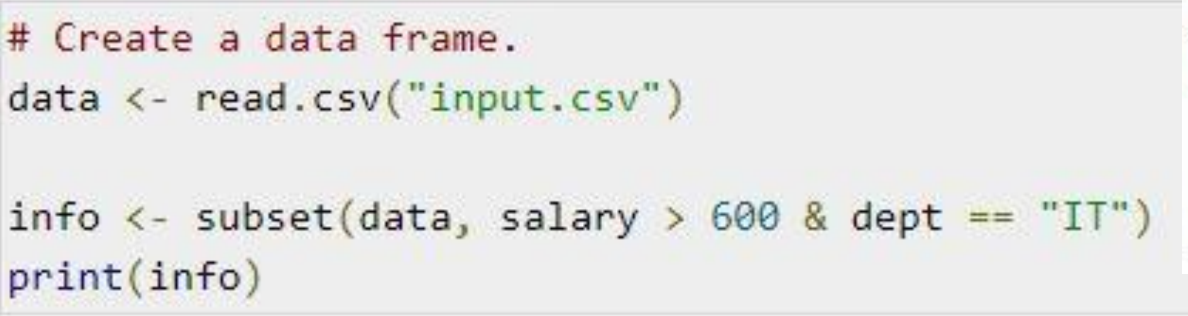


```
# Create a data frame.
data <- read.csv("input.csv")

retval <- subset( data, dept == "IT")
print(retval)
```

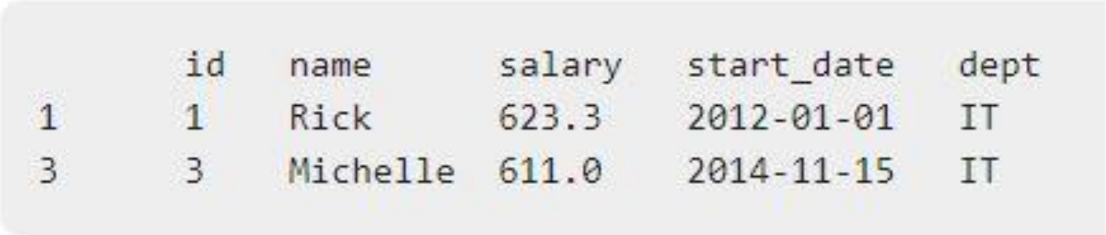


```
        id    name        salary    start_date    dept
1       1     Rick        623.3     2012-01-01    IT
3       3     Michelle    611.0     2014-11-15    IT
6       6     Nina        578.0     2013-05-21    IT
```

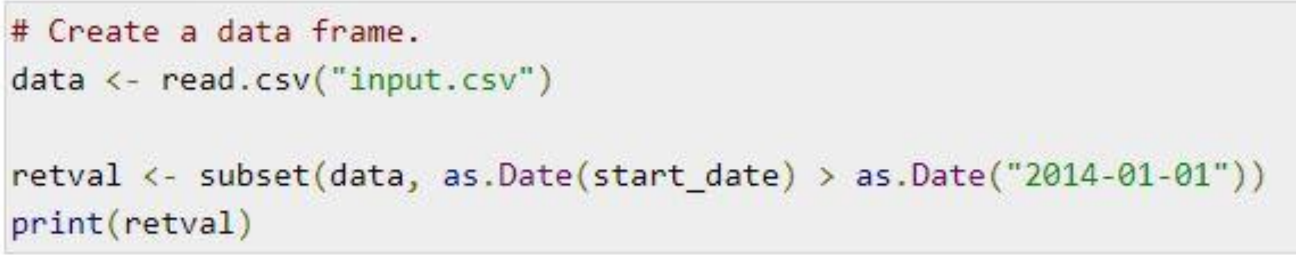
ANALYSIS THE CSV FILE



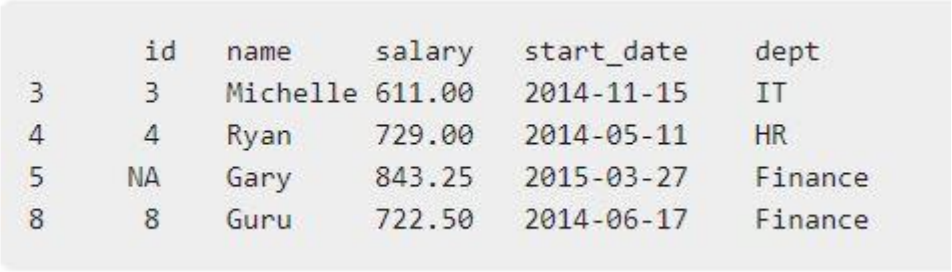
```
# Create a data frame.
data <- read.csv("input.csv")

info <- subset(data, salary > 600 & dept == "IT")
print(info)
```



```
      id    name       salary    start_date    dept
1     1     Rick       623.3     2012-01-01    IT
3     3     Michelle   611.0     2014-11-15    IT
```

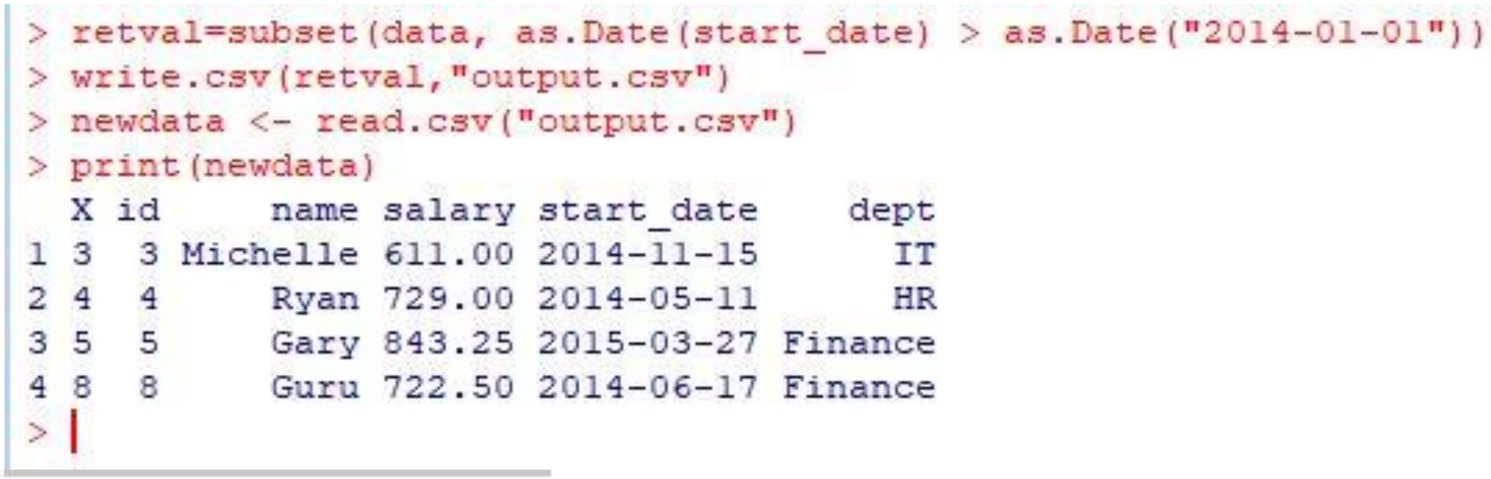


```
# Create a data frame.
data <- read.csv("input.csv")

retval <- subset(data, as.Date(start_date) > as.Date("2014-01-01"))
print(retval)
```

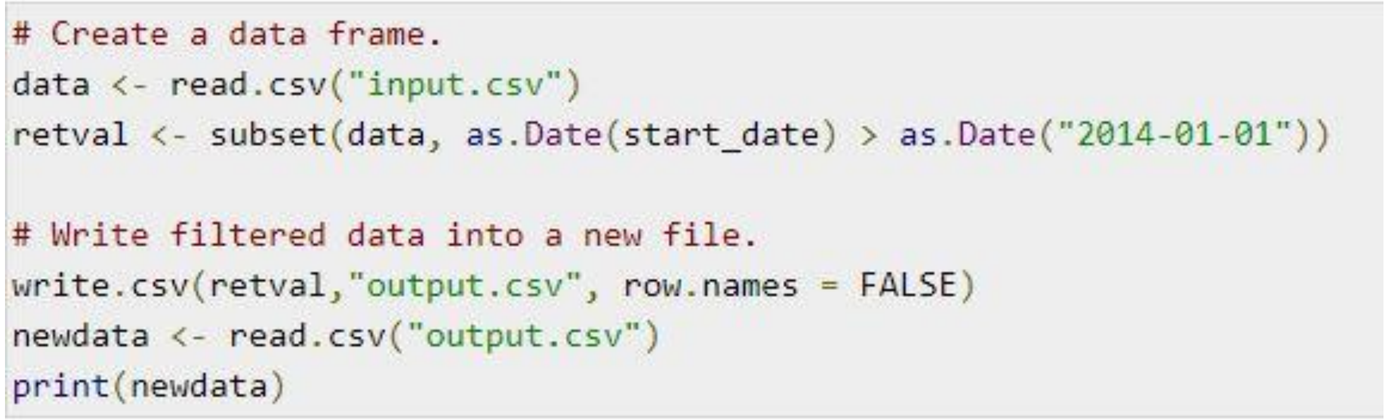


```
       id    name      salary    start_date    dept
3      3     Michelle 611.00    2014-11-15    IT
4      4     Ryan      729.00    2014-05-11    HR
5     NA     Gary      843.25    2015-03-27    Finance
8      8     Guru      722.50    2014-06-17    Finance
```

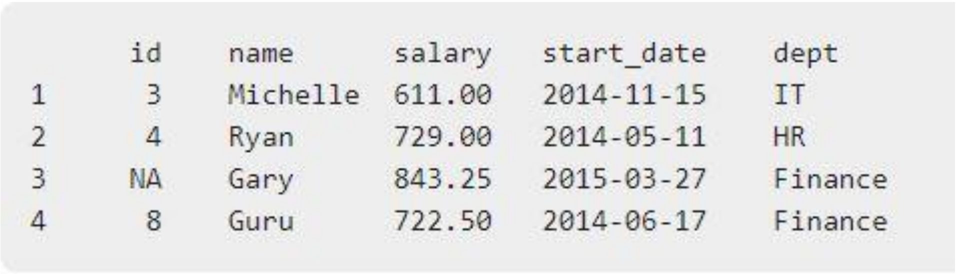
WRITING INTO A CSV FILE



```
> retval=subset(data, as.Date(start_date) > as.Date("2014-01-01"))
> write.csv(retval,"output.csv")
> newdata <- read.csv("output.csv")
> print(newdata)
  X id     name salary start_date    dept
1 3  3 Michelle 611.00 2014-11-15      IT
2 4  4     Ryan 729.00 2014-05-11      HR
3 5  5     Gary 843.25 2015-03-27 Finance
4 8  8     Guru 722.50 2014-06-17 Finance
> 
```



```
# Create a data frame.
data <- read.csv("input.csv")
retval <- subset(data, as.Date(start_date) > as.Date("2014-01-01"))

# Write filtered data into a new file.
write.csv(retval,"output.csv", row.names = FALSE)
newdata <- read.csv("output.csv")
print(newdata)
```



```
      id    name      salary   start_date   dept
1      3    Michelle  611.00   2014-11-15   IT
2      4    Ryan      729.00   2014-05-11   HR
3     NA    Gary      843.25   2015-03-27   Finance
4      8    Guru      722.50   2014-06-17   Finance
```