

Unit 3 : Function

Function definition, built in functions: `mean()`, `paste()`, `sum()`, `min()`, `max()`, `seq()`, user-defined function, calling a function, calling a function without an argument, calling a function with argument values - Strings – manipulating text in data: `substr()`, `strsplit()`, `paste()`, `grep()`, `toupper()`, `tolower()`.

What is a Function in R?

In programming, functions are instructions organized together to carry out a specific task. The rationale behind functions is to create self-contained programs that can be called only when needed.

With functions, programmers no longer need to write a program from scratch, thereby avoiding repetition, and improving code robustness and readability. That's why, as a rule of thumb, it's good practice to create a function whenever you expect to run a particular set of instructions more than twice in your code.

Functions can be used for endless purposes and can take various forms. Generically, the vast majority of functions will take input data, process it, and return a result. The data on which the function operates is specified by the so-called arguments, which can also be used to control or alter the way the function carries out the tasks.

Depending on the origin of the function, we can distinguish three main types of functions in R:

- Built-in functions
- Functions available in R packages
- User-Defined functions (UDF)

Built-in Functions in R

R is a powerful programming language that comes with a wide catalog of built-in functions that can be called anytime. As a math-oriented language, R comes with a good number of functions to perform numeric operations. Below you can find a list of some of the most useful:

- `print()`. Displays an R object on the R console
- `min()`, `max()`. Calculates the minimum and maximum of a numeric vector
- `sum()`. Calculates the sum of a numeric vector
- `mean()`. Calculates the mean of a numeric vector
- `range()`. Calculates the minimum and maximum values of a numeric vector
- `str()`. Displays the structure of an R object
- `ncol()`. Returns the number of columns of a matrix or a dataframe
- `length()`. Returns the number of items in an R object, such as a vector, a list, and a matrix.

- In the code below, you can see how simple is to use these functions to calculate certain statistics from a vector:

```
• >>> v <- c(1, 3, 0.2, 1.5, 1.7)
• >>> print(v)
• [1] 1.0 3.0 0.2 1.5 1.7
• >>> sum(v)
• [1] 7.4
• >>> mean(v)
• [1] 1.48
• >>> length(v)
• [1] 5
```

Built-in Function

Simple examples of in-built functions are seq(), mean(), max(), sum(x) and paste(...) etc. They are directly called by user written programs. You can refer most widely used R functions.

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 from 41 to 68.

```
print(sum(41:68))
```

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

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

```
[1] 53.5
```

```
[1] 1526
```

Math Functions - Built in function

R provides the various mathematical functions to perform the mathematical calculation. These mathematical functions are very helpful to find absolute value, square value and much more calculations.

In R, there are the following functions which are used

S. No	Function	Description	Example
1.	abs(x)	It returns the absolute value of input x.	<pre>x<- -4 print(abs(x))</pre> Output [1] 4
2.	sqrt(x)	It returns the square root of input x.	<pre>x<- 4 print(sqrt(x))</pre> Output [1] 2
3.	ceiling(x)	It returns the smallest integer which is larger than or equal to x.	<pre>x<- 4.5 print(ceiling(x))</pre> Output [1] 5
4.	floor(x)	It returns the largest integer, which is smaller than or equal to x.	<pre>x<- 2.5 print(floor(x))</pre> Output [1] 2
5.	trunc(x)	It returns the truncate value of input x.	<pre>x<- c(1.2,2.5,8.1) print(trunc(x))</pre> Output [1] 1 2 8
6.	round(x, digits=n)	It returns round value of input x.	<pre>x<- -4 print(abs(x))</pre> Output 4

7.	cos(x), sin(x), tan(x)	It returns cos(x), sin(x) value of input x.	<pre>x<- 4 print(cos(x)) print(sin(x)) print(tan(x))</pre> Output <pre>[1] -0.6536436 [2] -0.7568025 [3] 1.157821</pre>
8.	log(x)	It returns natural logarithm of input x.	<pre>x<- 4 print(log(x))</pre> Output <pre>[1] 1.386294</pre>
9.	log10(x)	It returns common logarithm of input x.	<pre>x<- 4 print(log10(x))</pre> Output <pre>[1] 0.60206</pre>
10.	exp(x)	It returns exponent.	<pre>x<- 4 print(exp(x))</pre> Output <pre>[1] 54.59815</pre>

String Function

R provides various string functions to perform tasks. These string functions allow us to extract sub string from string, search pattern etc. There are the following string functions in R:

S. No	Function	Description	Example
1.	substr(x, start=n1, stop=n2)	It is used to extract substrings in a character vector.	<pre>a <- "987654321" substr(a, 3, 3)</pre> Output <pre>[1] "3"</pre>
2.	grep(pattern, x, ignore.case=FALSE, fixed=FALSE)	It searches for pattern in x.	<pre>st1 c('abcd', 'bdcd', 'abcdabcd') pattern<- '^abc' print(grep(pattern, st1))</pre> Output <pre>[1] 1 3</pre>

3.	sub(pattern, replacement, x, ignore.case = FALSE, fixed=FALSE)	It finds pattern in x and replaces it with replacement (new) text.	<pre>st1<- "England is beautiful but no the part of EU" sub("England", "UK", st1)</pre> Output <pre>[1] "UK is beautiful but not a part of EU"</pre>
4.	paste(..., sep="")	It concatenates strings after using sep string to separate them.	<pre>paste('one',2,'three',4,'five')</pre> Output <pre>[1] one 2 three 4 five</pre>
5.	strsplit(x, split)	It splits the elements of character vector x at split point.	<pre>a<-"Split all the character" print(strsplit(a, ""))</pre> Output <pre>[[1]] [1] "split" "all" "the" "character"</pre>
6.	tolower(x)	It is used to convert the string into lower case.	<pre>st1<- "shuBHAm" print(tolower(st1))</pre> Output <pre>[1] shubham</pre>
7.	toupper(x)	It is used to convert the string into upper case.	<pre>st1<- "shuBHAm" print(toupper(st1))</pre> Output <pre>[1] SHUBHAM</pre>

How to Use paste & paste0 Functions in R to Concatenate Strings

paste () — this function concatenates the input values and returns a single character string. **paste0 ()** — this function concatenates the input values in a single character string. The difference between: **paste** and **paste0** is that **paste** function provides a separator operator, whereas **paste0** does not.

Example 1

Probably, function `paste` is one of the most used function in R. The objective of this function is concatenate a series of strings.

```
# First example  
paste("file", "number", "32")  
[1] "file number 32"
```

```
# Second example  
paste("file", "number", "32", sep = "_")  
[1] "file_number_32"
```

You can use the **`paste()`** and **`paste0()`** functions in R to concatenate elements of a vector into a single string.

The **`paste()`** function concatenates strings using **a space** as the default separator.

The **`paste0()`** function concatenates strings using **no space** as the default separator.

These functions use the following basic syntax:

```
paste(x, sep = " ", collapse = NULL)  
paste0(x, collapse = NULL)
```

where:

- **x**: The vector of elements to concatenate
- **sep**: The separator to use when concatenating
- **collapse**: Value to use when joining elements into single string

Example 2: Use `paste()`

The following code shows how to use the **`paste()`** function to concatenate several strings into a single string:

```
#concatenate several elements into one string
paste("I", "ride", "my", "bike", 25, "times")
```

```
[1] "I ride my bike 25 times"
```

Each element is concatenated into one string using a space as the default separator.

Example 3: Use paste() with sep

The following code shows how to use the **paste()** function with the **sep** argument to concatenate several strings into one string, using an underscore as the separator:

```
#concatenate elements using _ as separator
paste("I", "ride", "my", "bike", 25, "times", sep="_")
```

```
[1] "I_ride_my_bike_25_times"
```

Each element is concatenated into one string using an underscore as the separator.

Example 4: Use paste() with sep and collapse

The following code shows how to use the **paste()** function with the **sep** and **collapse** arguments to concatenate several strings into one string:

```
#concatenate elements using sep and collapse arguments
paste(c("A", "B", "C"), c(1, 2, 3), sep="_", collapse=" and ")
```

```
[1] "A_1 and B_2 and C_3"
```

The **sep** argument was used to join together corresponding elements in each vector and the **collapse** argument was used to join together all of the elements into one string.

Example 1: Use paste0()

The following code shows how to use the **paste0()** function to concatenate several strings into a single string:

```
#concatenate several elements into one string
paste0("I", "ride", "my", "bike", 25, "times")
```

[1] "Iridemybike25times"

Each element is concatenated into one string using no space as the separator.

Functions in R Packages

Yet numerous and diverse, built-in functions are not enough to do all the cool stuff you can do with R, from plotting compelling data visualizations to training powerful machine learning models.

The great majority of functions to perform these tasks are available in external packages or libraries. Packages are collections of R functions, data, and compiled code in a well-defined format created to add specific functionality. Most of these packages can be used for free, and can be found in popular packages repositories, such as CRAN, which currently feature nearly 20,000 contributed packages.

To use the functions available in a package, you first will need to install it. For example, if you want to install `stringr`, a popular package to work with regular expressions, you can use the following statement:

```
install.packages('stringr')
```

Once you have installed it, to load it into your R environment, use the `library` statement

```
library(stringr)
```

Now you're ready to use all the functions available in the `stringr` packages. For example, let's try the `str_detect()` function, which returns a logical vector with `TRUE` for each element of the string that matches pattern and `FALSE` otherwise.


```
str_detect('DataCamp', "Data")
```

```
[1] TRUE
```

User-Defined Functions

Function Definition

An R function is created by using the keyword **function**. The basic syntax of an R function definition is as follows –

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

The best way to understand how functions in R work is by creating your own functions. The so-called User-Defined functions (UDF) are designed by programmers to carry out a specific task.

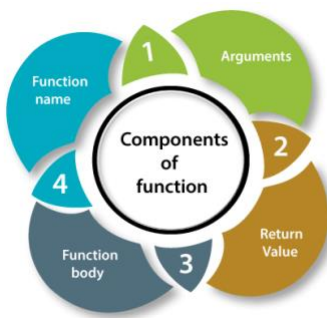
R functions normally adopt the following syntax:

```
function_name <- function(argument_1, argument_2) {  
  
  function body  
  
  return (output)  
  
}
```

We can distinguish the four main elements:

- **Function name.** To create a UDF, first you have to assign it a name and save it as a new object. You just have to call the name whenever you want to use the function.

- **Arguments.** The function arguments (also known as parameters) are provided within the parentheses. Arguments are key for the function to know what data to take as input and/or how to modify the behavior of the function.
- **Function body.** Within curly brackets comes the body of the function, that is, the instructions to solve a specific task based on the information provided by the arguments.
- **Return statement.** The return statement is required if you want the function to save as variables the result or results following the operations in the function body.



User-defined function

R allows us to create our own function in our program. A user defines a user-defined function to fulfill the requirement of user. Once these functions are created, we can use these functions like in-built function.

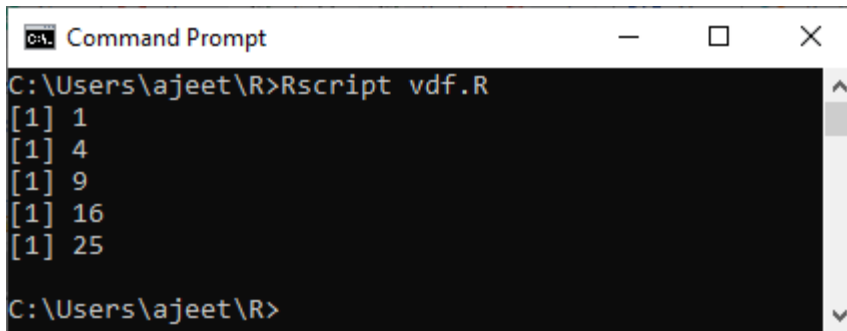
Creating a function without an argument.

```

new.function <- function() {
  for(i in 1:5)
  {
    print(i^2)
  }
}

new.function()
  
```

output

A screenshot of a Windows Command Prompt window. The title bar reads "Command Prompt". The command prompt shows the directory "C:\Users\ajeet\R" and the command "Rscript vdf.R" being executed. The output of the script is displayed as five lines, each starting with "[1]" followed by a number: 1, 4, 9, 16, and 25. The prompt then returns to "C:\Users\ajeet\R>".

```
Command Prompt
C:\Users\ajeet\R>Rscript vdf.R
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
C:\Users\ajeet\R>
```

Function calling with an argument

We can easily call a function by passing an appropriate argument in the function. Let see an example to see how a function is called.

Creating a function to print squares of numbers in sequence.

```
new.function <- function(a)
```

```
{
  for(i in 1:a) {
    b <- i^2
    print(b)
  }
}
```

Calling the function **new.function** supplying 10 as an argument.

```
new.function(10)
```

Output

```
Command Prompt
Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\ajeet>cd R

C:\Users\ajeet\R>Rscript calling.R
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36
[1] 49
[1] 64
[1] 81
[1] 100

C:\Users\ajeet\R>
```

Function calling with no argument

In R, we can call a function without an argument in the following way

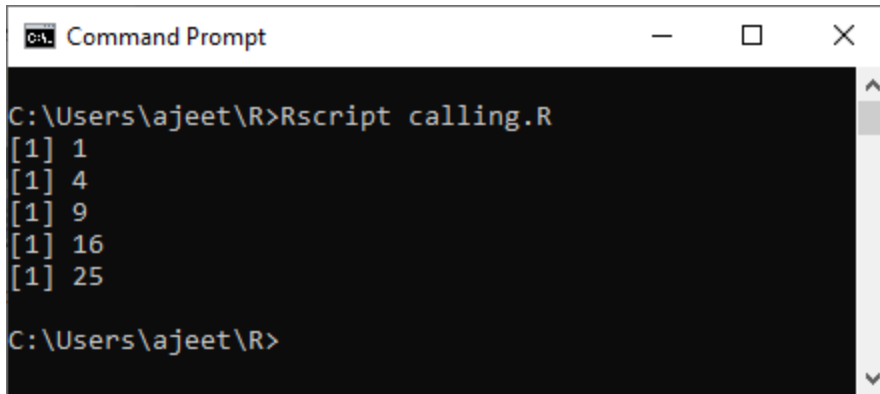
Creating a function to print squares of numbers in sequence.

```
new.function <- function() {  
  for(i in 1:5) {  
    a <- i^2  
    print(a)  
  }  
}
```

Calling the function **new.function** with no argument.

```
new.function()
```

Output



```
C:\Users\ajeet\R>Rscript calling.R
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
C:\Users\ajeet\R>
```

Function calling with Argument Values

We can supply the arguments to a function call in the same sequence as defined in the function or can supply in a different sequence but assigned them to the names of the arguments.

Creating a function with arguments.

```
new.function <- function(x,y,z) {  
  result <- x * y + z  
  print(result)  
}
```

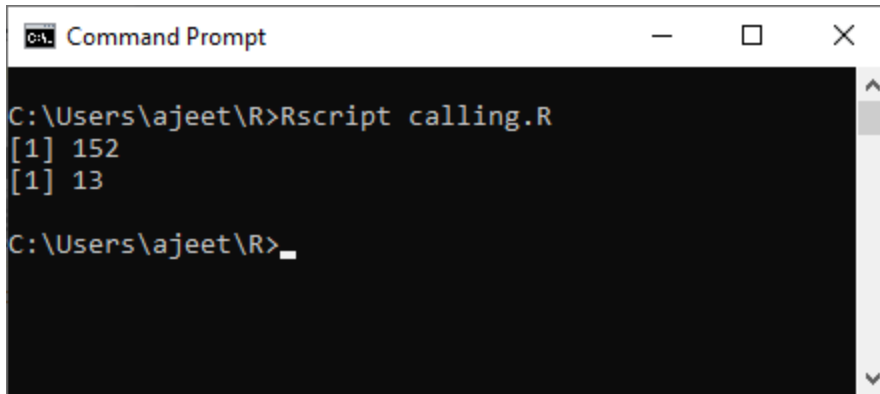
Calling the function by position of arguments.

```
new.function(11,13,9)
```

Calling the function by names of the arguments.

```
new.function(x = 2, y = 5, z = 3)
```

Output

A screenshot of a Windows Command Prompt window. The title bar reads "C:\ Command Prompt". The command prompt shows the following text:

```
C:\Users\ajeet\R>Rscript calling.R
[1] 152
[1] 13
C:\Users\ajeet\R>_
```

Function calling with default arguments

To get the default result, we assign the value to the arguments in the function definition, and then we call the function without supplying argument. If we pass any argument in the function call, then it will get replaced with the default value of the argument in the function definition.

Creating a function with arguments.

```
new.function <- function(x = 11, y = 24) {
  result <- x * y
  print(result)
}
```

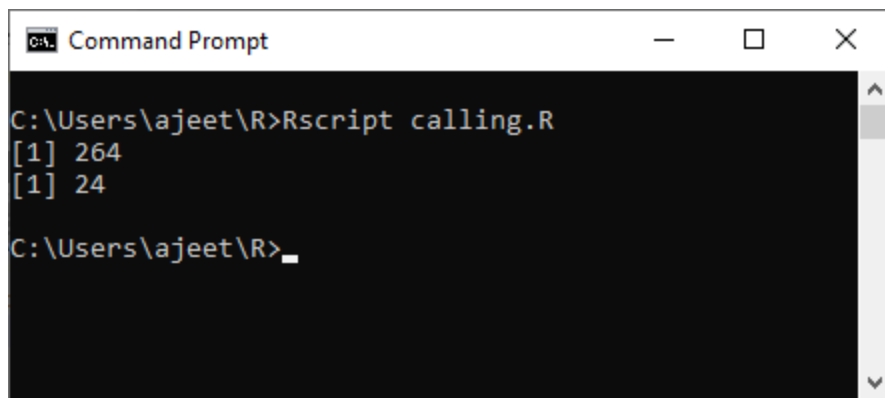
Calling the function without giving any argument.

```
new.function()
```

Calling the function with giving **new** values of the argument.

```
new.function(4,6)
```

Output



A screenshot of a Windows Command Prompt window. The title bar at the top reads "C:\> Command Prompt" and includes standard window controls (minimize, maximize, close). The command prompt shows the following text:

```
C:\Users\ajeet\R>Rscript calling.R  
[1] 264  
[1] 24  
  
C:\Users\ajeet\R>_
```

The output consists of two lines of R console output: "[1] 264" and "[1] 24". The prompt "C:\Users\ajeet\R>" is followed by an underscore character, indicating the command has been executed and the prompt is ready for the next input. A vertical scrollbar is visible on the right side of the command prompt window.