

## Functions

## Table of Content

Introduction .....	3
1. User-defined functions .....	4
1.1 Function with Argument.....	5
1.2 Function without Argument .....	7
1.3 Function with Default Argument .....	7
1.4 Functions with Returning Values.....	9
1.5 Functions with Variable Number of Arguments (The three dots construct in R ) .....	10
Summary .....	12

## Introduction

User-defined functions are an integral part of every programming language. They are useful in facilitating code re-use. This topic would discuss the concept of user-defined functions and their uses. Several use cases demonstrating different ways of using functions and functions returning values are given.

## Learning Objectives

Upon completion of this topic, you will be able to:

- Explain user-defined functions
- Create user-defined functions and use them
- Write functions with arguments and without arguments
- Demonstrate the ability to program with functions taking variable number of arguments
- Develop functions returning values and use them in programming

## 1. User-defined functions

A function is a block of code which is intended to do a specific task, the idea here is to facilitate code reuse. A function can be written in several ways as mentioned below:

Functions without arguments

Functions with default arguments

Functions with returning values

Functions with variable number of arguments

There are a lot of built-in functions in R such as `print()`, `sort()`, `order()`, `revert()` and so on. However, the programmer can also define functions for a specific purpose by following the below syntax.

### Pseudocode:

```
1 function_name <- 2 function(arg_1, arg_2, ...) {  
    Function body 3  
}
```

1. *function\_name*: It is the actual function name of interest
2. *arg\_1*, *arg\_2*: Arguments hold values than be can be passed to the function. A function may or may not contain arguments, and the number of arguments is not fixed.
3. *Function Body*: It consists the code block.

## 1.1 Function with Argument

The function below contains arguments within the function declaration.

### Example 1:

*#myfunc is the function name and, "a" is the argument it takes*

```
myfunc = function(a) { # A function to print hello, a certain number times
```

```
    i = 1
```

```
    while( i <= a) {
```

```
        print("Hello")
```

```
        i=i+1
```

```
    }
```

```
} # end of function
```

*#Above is the example to invoke (call) the user-defined function "myfunc"*

*myfunc(5) # The value 5, is passed to the function and gets stored in the #variable "a" in the block where the function is defined!*

Demonstrating an example of writing a user-defined function to print the square of a number.

**Example 2:**

```
findsquare <- function(x) {  
  
  square <- x * x  
  
  return(square) #The return statement sends the computed value back to  
  the #point from where the function was called  
  
}  
  
val=findsquare(5) #The value returned by the findsquare function is  
stored #in the object "val"  
  
print(val)
```

**Output:**


```
[1] 25
```

**Example 3:**

```
mysq.func("hi") # a character is squared, -which will give an error
```

**Output:**

```
Error in x * x: non-numeric argument to binary operator
```



**Note:** The type of the argument passed to a function must be in sync with the operation performed on the data object inside the function.

## 1.2 Function without Argument

The function below does not contain any arguments in the function definition.

### Example:

```
greetings = function() {  
  print("Hello..good morning")  
}  
  
greetings() #it will just call the function and print the message
```

## 1.3 Function with Default Argument

Functions with default arguments contain default values that are passed to the parameters in the function definition.

### Example 1:

```
samp.func <- function(a, b=2) {  
  a^2  
}  
  
samp.func(2) #Function is called from this point
```

### Output:

```
[1] 4
```

The `samp.func()` function never uses the argument `b`, so calling `samp.func(2)` will not return an error because the value "2" gets matched to "a" with respect to its position i.e 2 will be passed on to the first argument

**Example 2:**

```
samp.func2 <- function(a, b) {  
  print(a)  
  print(b)  
}
```

```
samp.func2(2)
```


**Output:**

```
[1] 2
```

```
Error in print(b): argument "b" is missing, with no default
```

The output shows that 2 got printed before the error was triggered.

This is because the argument **b** was not evaluated until after `print(a)`. Once the function tried to evaluate `print(b)`, it had to throw an error because there was no value passed corresponding to the second argument "b".



**Note:** Default arguments based on the variables inside the body of the function, is generally a bad practice.



## 1.4 Functions with Returning Values

Function with return statements will explicitly return values on execution of the function.

### Example 1:

```
mysum = function(a,b)

{

s = a+b

return(s)

}

mysum(10,20)
```

### Output:

```
[1] 30
The return(s) command explicitly returns the value stored in object "s"
after executing the code in the mysum() function.
```

### Example 2: Demonstrating a default way of returning values from a function

```
mysum = function(a,b)

{

s = a+b

b = 100 #Since b is the last object in this block, b will be returned

}

print(mysum(10,20))
```

### Output:

```
[1] 100
```



**Note:** There is no explicit return statement used in the function body. However, the value 100 is printed, the variable “b” was holding the value 100 and that’s the last object in the code inside the function which is returned by default.

### 1.5 Functions with Variable Number of Arguments (The three dots construct in R )

The three dots construct is a mechanism which allows variable number of arguments to be passed on to a given function. Generic functions use (...), so that extra arguments can be passed to methods inside the functions.

```
print(...) # It means that the function can take any number of named or unnamed arguments.
```

#### Example 1:

*Example 1:*

```
Greetings <- function(...) {  
  arguments <- list(...)  
  paste(arguments)  
}  
  
Greetings("Good", "Morning", "!")
```

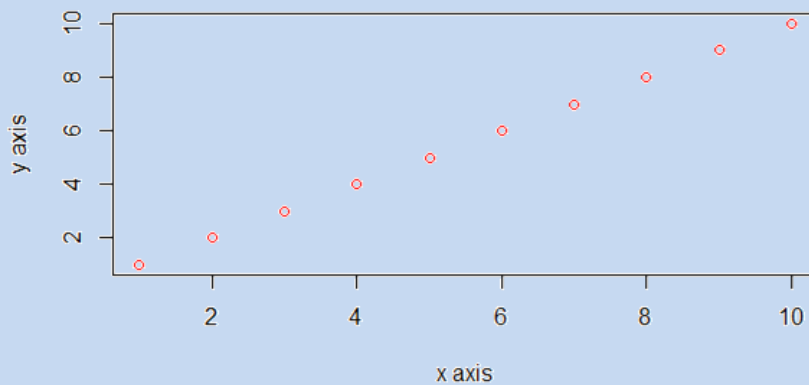
#### Output:

```
[1] "Good"      "Morning"  "!"
```

*The above function Greetings() accepts 3 arguments, however we have not mentioned any specific number of arguments in the function definition*

**Example 2:**

```
scatter.plot <- function(x, y, ...) {  
  plot(x, y, col="red", ...)  
}  
  
scatter.plot(1:10, 1:10, xlab="x axis", ylab="y axis")
```

**Output:**

In the above example the last two attributes xlab and ylab are passed through the plot even though it was not specified initially in the function declaration.

## Summary

The core idea of functions is to enable the programmer to develop modular and easy to maintain code keeping in mind the idea of code reusability.

Examples of functions are functions with arguments, without arguments, functions with variable number of arguments, and functions returning values.

Functions with default arguments contain default values that are passed to the parameters in the function definition.

Function with return statements explicitly returns values on execution of the function.

The three dots construct is a mechanism which allows variable number of arguments to be passed on to a given function.