Creating Functions

In this lesson, we will learn what functions are and how to create them in the R language.

We'll cover the following

What is a Function?

A function is a set of **statements** that are executed together to achieve a specific goal or task.

You might remember that **everything in R is an object**. From this, we can conclude that a function is also an object in R language.

Syntax of a Function

An R function is created by using the keyword `function`. The basic syntax is as follows:

```
functionName <- function(argument1, argument2, ..., argumentN)
{
   # Statements
}
```

Let’s have a look at the illustration:

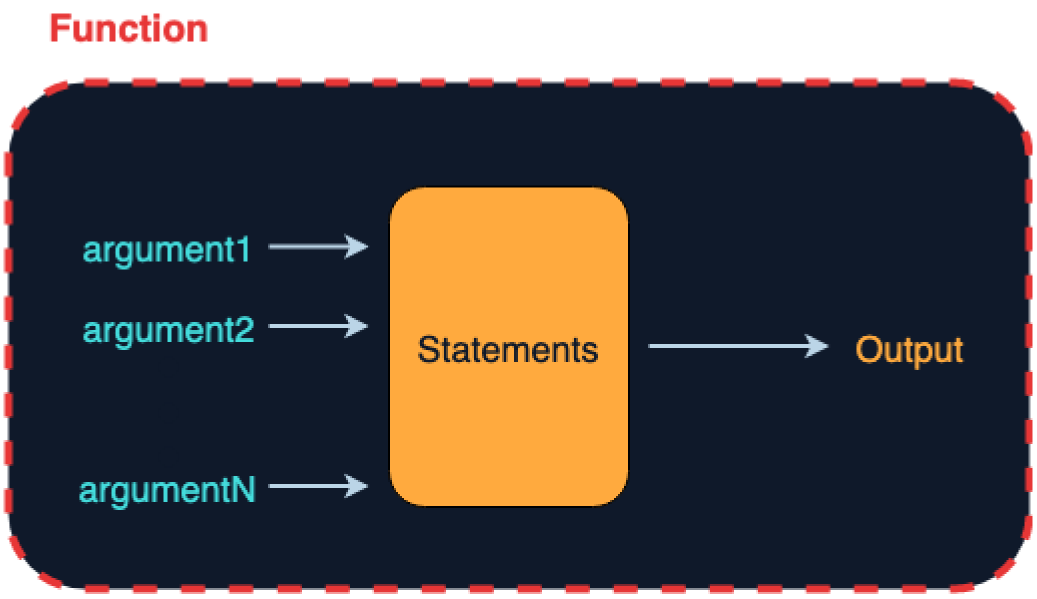


Illustration of a function

Components of a function

functionName: This is the actual name of the function and is stored as an object in R.

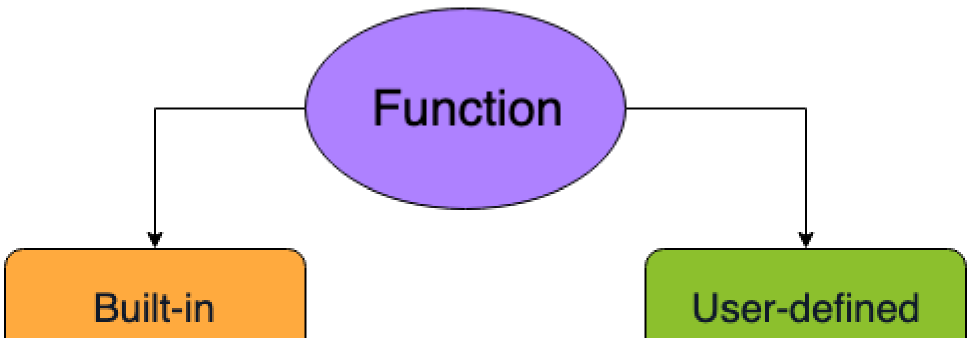
arguments: – We can pass data to the function. These are like the inputs to a function. However, arguments are optional; that is, a function may contain no arguments. Also, arguments can have default values. The **default value** of an argument is that value which it takes if no value is specified for that argument in a function call.

Statements: – These are considered the **function's body** that defines what the function actually does.

Output – This is the return value of a function.

Types of Functions

There are two types of functions in R language.



Types of functions

Built-in R functions

You might have noticed we have been using **built-in** functions from the very first chapter such as `print()`, `cat()`, `length()` etc. Such functions are pre-defined for us programmers and called by user-written programs. We simply pass them the *parameters* or *arguments* that those functions need as input.

However, the interesting part is that we can make our functions and call the function as needed. This way, we can reuse commonly needed blocks of code.

User-defined R functions

Suppose you want to find the maximum number of two numbers:

```r
myNumber1 <- 2
myNumber2 <- 5
if(myNumber1 > myNumber2)
{
  print(myNumber1)
} else
{
  print(myNumber2)
}
```

Finding max between two numbers

The above code simply defines two numbers and uses the `if...else` conditional statement to display the larger number. What if we want to do this again for another pair of numbers, i.e. do the task of finding the maximum number twice:

```r
myNumber1 <- 2
myNumber2 <- 5
if(myNumber1 > myNumber2)
{
  print(myNumber1)
} else
{
  print(myNumber2)
}

myNumber3 <- 7
myNumber4 <- 5
if(myNumber3 > myNumber4)
{
  print(myNumber3)
} else
{

  print(myNumber4)
}
```

Finding maximum between two numbers

As you can see, the same code is copied twice **Line number 3 to 5** and **Line number 13 to 19**. It is performing the same tasks, just on different inputs.

Similarly, we may have to find the maximum of two numbers at various points of the code. Copying and pasting the same code, again and again, is tedious and the probability of errors occurring also increases.

However, we can create a `function` for such a task. We can encapsulate the if-else code that finds the maximum of two numbers. The numbers will become inputs to the `function`. These are the *arguments* on which the task will be performed.

Let’s have a look at the code.

```
maxNumber <- function(myNumber1, myNumber2) { # implementing a function
  # function body
  if(myNumber1 > myNumber2)
  {
    print(myNumber1)
  } else
  {
    print(myNumber2)
  }
} # end function

# Driver Code
m <- 2
n <- 5
maxNumber(m, n) # calling the maxNumber() function

# we can now call this function as many times as we want to
m <- 7
n <- 5
maxNumber(m, n)

m <- 11
n <- 90
maxNumber(m, n)
```

In the next lesson, let’s perform a hands-on exercise on creating functions.