R Functions in Detail (With Examples)

Perth Coding Club

R Functions in Detail (With Examples)

In this article, you'll learn everything about functions in R programming; how to create them, why it is used and so on.

Functions are used to logically break our code into simpler parts which become easy to maintain and understand.

It's pretty straightforward to create your own function in R programming.

Syntax for Writing Functions in R

```
func_name <- function (argument) {
statement
}
```

- Here, we can see that the reserved word `function` is used to declare a function in R.
- The statements within the curly braces form the body of the function. These braces are optional if the body contains only a single expression.
- Finally, this function object is given a name by assigning it to a variable, `func_name`.

Example of a Function

```
pow <- function(x, y) {
# function to print x raised to the power y
result <- x^y
print(paste(x,"raised to the power", y, "is", result))
}
```

Here, we created a function called `pow()`.

It takes two arguments, finds the first argument raised to the power of second argument and prints the result in appropriate format.

We have used a built-in function `paste()` which is used to concatenate strings.

How to call a function?

We can call the above function as follows.

```
>pow(8, 2)
[1] "8 raised to the power 2 is 64"
> pow(2, 8)
[1] "2 raised to the power 8 is 256"
```

Here, the arguments used in the function declaration (`x` and `y`) are called formal arguments and those used while calling the function are called actual arguments.

Named Arguments

In the above function calls, the argument matching of formal argument to the actual arguments takes place in positional order.

This means that, in the call `pow(8,2)`, the formal arguments `x` and `y` are assigned 8 and 2 respectively.

We can also call the function using named arguments.

When calling a function in this way, the order of the actual arguments doesn't matter. For example, all of the function calls given below are equivalent.

```
> pow(8, 2)
[1] "8 raised to the power 2 is 64"
> pow(x = 8, y = 2)
[1] "8 raised to the power 2 is 64"
> pow(y = 2, x = 8)
[1] "8 raised to the power 2 is 64"
```

Furthermore, we can use named and unnamed arguments in a single call.

In such case, all the named arguments are matched first and then the remaining unnamed arguments are matched in a positional order.

```
> pow(x=8, 2)
[1] "8 raised to the power 2 is 64"
> pow(2, x=8)
[1] "8 raised to the power 2 is 64"
```

In all the examples above, `x` gets the value 8 and `y` gets the value 2.

Default Values for Arguments

We can assign default values to arguments in a function in R.

This is done by providing an appropriate value to the formal argument in the function declaration.

Here is the above function with a default value for `y`.

```
pow <- function(x, y = 2) {
# function to print x raised to the power y
result <- x^y
print(paste(x,"raised to the power", y, "is", result))
}
```

The use of default value to an argument makes it optional when calling the function.

```
> pow(3)
[1] "3 raised to the power 2 is 9"
> pow(3,1)
[1] "3 raised to the power 1 is 3"
```

Here, `y` is optional and will take the value 2 when not provided.

Check out these examples to learn more:

- Find Sum, Mean and Product of Vector in R Programming
- Generate Random Number from Standard Distributions
- Sample from a Population