


AI and Biotechnology/Bioinformatics

R Crash Course (2025)

Session 3: Basics of R Programming Language Part II (Understanding R Functions)

Welcome back to your R learning journey!

We have commenced learning with the software installation, the setup of environment, and understanding the fundamentals of coding in R.

If you missed any part, all learning resources and R scripts are available on our GitHub repository. Allowing you to learn at your own pace.  [GitHub Repository](#)

Moving ahead in this session we'll delve into R functions, an integral part of programming that facilitate the development of clean, error-free and reusable codes. This is utmost important in data science, and AI-based bioinformatics pipelines where the focus lies on automation & reproducibility of results.

Learning goal: Creating customized functions in R utilizing different scenarios.

What is Function

A function is a modular segment of code designed to execute a specific task when it's called.

Think of it like an automatic bioanalyzer you feed it blood sample (input), and it returns the HBA1C level (output). Whether you provide one sample or hundred, it will process only the ones you choose and return the HBA1C levels accordingly

Types of Functions in R






❖ Built-in Functions:

That are pre-installed, readily available functions in base R you can use them immediately, such as `sum()`, `mean()` & `median` etc.

❖ User-Defined Functions:

These are functions you create to meet specific requirements of your goal.

For example, the `mode()` function in base R does not return the most frequent value; it indicates the data type (like numeric or character). To compute the statistical mode, a user-defined function is necessary.

-  Customized functions reduce the need of code repetition.
-  They allow the reuse of same logic across different variables and datasets.
-  They enable modification of pre-existing functions.
-  They allow you to devise your own functions.
-  They help in task/pipeline automations

How to Create Functions in R?

To define a function, use the `function()` keyword.

A user-defined function in R must have these key elements:

✚ **Function Name:** This is the unique name you provide to your function. It should spell out what the function does.

Example: A function to calculate BMI might be named `calculate_bmi`.

✚ **Parameter/Argument:** These are the input values your function will work with. They go inside the parentheses when you define or call the function

Example: `function(weight, height)`

✚ **Function body:** This part contains the actual logic or task that your function performs with the input. This is the code block inside curly braces `{ }` when you define your function

✚ **Return Statement:** This is the result that the function outputs after it finishes processing. It's written using the `return()` function.

Note: in R, the last line of a function is returned automatically, even without `return ()`.

✚ **Calling a function:** Once defined, you can run the function by calling its name followed by parentheses by passing the required arguments.

Example: `calculate_bmi()`

Together these elements develop comprehensive & functional block that helps make your code more modular, readable and reusable. You can apply the same block of code across different variables or datasets when you need

```
# Create a function
calculate_bmi <- function(weight_kg, height_m) {
  bmi <- weight_kg / (height_m ^ 2)
  return(bmi)
}

# Define Variable
weight <- 70
height_m <- 1.75

# Call a function
bmi <- calculate_bmi(weight, height_m)
```

Parameters vs Arguments

- ✚ **Parameter:** A variable listed in the function definition (e.g., value) a dummy argument
- ✚ **Argument:** The actual value passed to the function when it's called

```
# Create a function  
function(weight_kg, height_m) # Parameter/dummy argument  
  
# Call a function  
calculate_bmi(weight, height_m) # actual argument
```

Things to Remember !

R functions can take different numbers arguments(inputs), and how you define or use them affects your results. Below are key points to remember when working with function arguments.

➤ You Can Pass One or More Arguments

You can define a function with **one or multiple inputs** depending on your task.

Example: With One Argument

```
# Check if a gene is highly expressed with if else statement within function  
check_expression <- function(expression_value) {  
  if (expression_value > 20) {  
    return("Highly expressed")  
  } else {  
    return("Low expression")  
  }  
}  
  
check_expression(25)
```

Example: With Two Arguments

```
# Calculate BMI from weight and height  
calculate_bmi <- function(weight_kg, height_m) {  
  bmi <- weight_kg / (height_m^2)  
  return(bmi)  
}  
  
# Call function  
calculate_bmi(70, 1.75)
```

➤ Number of Argument Must Match

If your function expects two inputs, you must provide both otherwise, R gives an error.

```
calculate_bmi(70)  
Error: argument "height_m" is missing, with no default
```

It throws an error because height was used to compute BMI while defining function (check in above example)

➤ Use Default Values to Handle Missing Arguments

You can define default values for arguments to avoid errors when some inputs are missing.

Example: Assume height is 1.70m if not provided

```
calculate_bmi <- function(weight_kg, height_m = 1.70) {  
  bmi <- weight_kg / (height_m^2)  
  return(bmi)  
}  
calculate_bmi(65)  
# Output: 22.49
```

➤ Lazy Evaluation in R

If a function has unused arguments, R won't throw an error unless those arguments are actually used. This makes R a lazy evaluator, allowing more flexibility.

Define function with three parameters: weight_kg, height_m, and age. But only use the first two to calculate BMI.

```
calculate_bmi <- function(weight_kg, height_m, age) {  
  bmi <- weight_kg / (height_m^2)  
  return(bmi)  
}  
  
# Call the function without providing age  
calculate_bmi(70, 1.75)  
# Output: 22.86
```

Even though age was not provided, there is no error. Because age is not used in the function body this is called lazy evaluation

If we update the function to include age in the output, R will now expect all three inputs and throw an error if any are missing.

```
calculate_bmi <- function(weight_kg, height_m, age) {  
  bmi <- weight_kg / (height_m^2)  
  paste("BMI:", round(bmi, 2), "| Age:", age)  
}  
  
calculate_bmi(70, 1.75)  
# Error: argument "age" is missing, with no default
```

Handle error using default values. This is useful when you have missing values in data

```
calculate_bmi <- function(weight_kg, height_m, age = NA) {  
  bmi <- weight_kg / (height_m^2)  
  paste("BMI:", round(bmi, 2), "| Age:", age)  
}  
  
calculate_bmi(70, 1.75)  
# No Error
```

That's all about for today, in this session, we learned:

- **Fundamentals of R functions**
- **How to create customized functions in R and its**

Want to try it yourself? Get the R script from this GitHub Link  [click here](#)