# Functions – Topic Explanation

2026-01-12

## Functions – Topic Explanation

### 1. What Is a Function?

While writing code, we often need to perform the same operations repeatedly (for example, calculating GC content multiple times).

Instead of writing the same code over and over again, we encapsulate this operation inside a function.

In other words:

Instead of writing the same code many times

We write it once

And call it whenever we need it

We can think of a function like a food processor:

Input: You put fruits into the processor

Process: The processor chops and mixes them

Output: You get fruit juice

In computers, a function is a black box that:

takes data as input,

processes that data,

and returns a result.

### 2. Structure of a Function

```
function_name <- function(parameter) {
  # Operations to be performed
  return(result) # Final output returned
}
```

function_name: The name we give to call the function (e.g. gc_calculate)

parameter: The material/data the function expects from outside

return: The final result the function gives back after finishing its task

### 3. A Simple Example

```
multiply_by_two <- function(x) {
  result <- x * 2
  return(result)
}
```

```r
multiply_by_two(5)
```

```
## [1] 10
```

multiply_by_two is the function name

x represents the element passed into the function

The expression x * 2 inside the function:

multiplies every value given to x by 2

The resulting value is stored in the result variable and returned at the end of the function.

To run the function:

we write the function name,

and put the value we want to multiply by 2 inside the parentheses.

## Function + Biological Example

Goal: Write a function that calculates GC content

Previously, we calculated GC content using:

for

if

a counter (accumulator)

Now, let's convert this logic into a function.

```r
gc_content_calculate <- function(dna) {

  GC_total <- 0

  for (n in dna) {
    if (n == "G" || n == "C") {
      GC_total <- GC_total + 1
    }
  }

  ratio <- GC_total / length(dna)
  return(ratio)
}

dna1 <- c("A", "G", "C", "T", "G")
gc_content_calculate(dna1)
```

```
## [1] 0.6
```

```r
#gc_content_calculate <- function(dna) {
```

gc_content_calculate → the name of the function (can be any valid name)

function(dna) → indicates that the function takes one input

dna → the DNA sequence passed into the function (e.g. c("A","G","C","T"))

{ } → the code block where the function operates

```
# GC_total <- 0
```

We create a variable named GC_total

We initialize it to 0

This variable will count how many G and C bases are present

```
#for (n in dna) {
```

Using a for loop, we go through each element of the DNA sequence

n represents the current nucleotide (A/T/G/C)

Meaning:

first iteration → first base

second iteration → second base

continues until the DNA sequence ends

```
#if (n == "G" || n == "C") {
```

This line asks the question:

"Is the current base G or C?"

== → equals

|| → logical OR

If n is G or C, the block below is executed.

```
#GC_total <- GC_total + 1
```

For every G or C found:

we increase GC_total by 1

This means:

"I found one more GC base, add it to the counter."

```
#ratio <- GC_total / length(dna)
```

GC_total → number of G and C bases

length(dna) → total number of bases in the DNA sequence

GC content formula:

GC content = (G + C) / total number of bases

```
#return(ratio)
```

We return ratio as the output of the function

When the function is called, this value is sent back to the user

## Bioinformatics Example: Leaf Status Analysis

```
leaf_analysis <- function(length) {
  if (length > 5) {
    result <- "Long leaf"
  } else {
```

```
    result <- "Short leaf"
  }
  return(result)
}

# Usage:
leaf_analysis(3.2) # Output: "Short leaf"
```

```
## [1] "Short leaf"
```

```
leaf_analysis(6.1) # Output: "Long leaf"
```

```
## [1] "Long leaf"
```