

Chapter 3 Functions in R

Most of what you do in R involves **using functions**. A function takes inputs (called **arguments**), performs an operation, and returns an output.

You have already used several functions, such as `mean()` , `summary()` , and `seq()` .

```
mean(ages)
```

```
[1] 23.5
```

```
summary(ages)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
18.00	20.25	23.00	23.50	26.25	30.00

3.1 Function arguments

Functions often require one or more arguments. For example:

```
mean(ages)
```

```
[1] 23.5
```

Here, `ages` is passed to the function `mean()` as its first argument.

Some functions require multiple arguments:

```
seq(from = 0, to = 1, by = 0.2)
```

```
[1] 0.0 0.2 0.4 0.6 0.8 1.0
```

3.2 Positional vs named arguments

Arguments can be passed to a function in **two ways**.

3.2.1 Positional arguments

If arguments are supplied **in the correct order**, you do not need to name them.

```
seq(0, 1, 0.2)
```

```
[1] 0.0 0.2 0.4 0.6 0.8 1.0
```

This works because R knows the expected order of arguments for `seq()` :

1. `from`
2. `to`
3. `by`

Positional arguments are concise, but they can make code harder to read and easier to misuse.

3.2.2 Named arguments (recommended)

You can explicitly name arguments using `argument = value` .

```
seq(from = 0, to = 1, by = 0.2)
```

```
[1] 0.0 0.2 0.4 0.6 0.8 1.0
```

Advantages of named arguments:

- Code is easier to read
- Order does not matter
- Fewer mistakes when functions have many arguments

For example, this works even though the order is different:

```
seq(by = 0.2, to = 1, from = 0)
```

```
[1] 0.0 0.2 0.4 0.6 0.8 1.0
```

In this course, **naming arguments is recommended**, especially for complex functions.

3.3 Default argument values

Many function arguments have **default values**. If you do not specify them, R uses the default.

Example:

```
mean(ages)
```

```
[1] 23.5
```

The function `mean()` has optional arguments such as `na.rm`, which defaults to `FALSE`.

```
mean(ages, na.rm = TRUE)
```

```
[1] 23.5
```

You only need to specify arguments when:

- You want a non-default behavior
 - The function requires the argument
-

3.4 Mixing positional and named arguments

You can mix positional and named arguments, but **positional arguments must come first**.

This is valid

```
mean(ages, na.rm = TRUE)
```

```
[1] 23.5
```

This will result in an error.

```
mean(na.rm = TRUE, ages)
```

```
[1] 23.5
```

A good rule of thumb:

- Use positional arguments for the **main input**
 - Use named arguments for **options and settings**
-

3.5 Viewing function documentation

To understand how a function works and what arguments it accepts, use the help system.

```
?mean
```

The help page shows:

- What the function does
- Required and optional arguments
- Default values
- Examples

When in doubt, **read the argument list first**.

3.6 Common mistakes to avoid

- Forgetting parentheses. This refers to the function itself, not the result.:

```
mean
```

```
function (x, ...)
  UseMethod("mean")
<bytecode: 0x0000029c291e1a90>
<environment: namespace:base>
```

- Misspelling argument names:

```
mean(ages, na_remove = TRUE)
```

```
[1] 23.5
```

- Assuming argument order without checking documentation
-

3.7 Key takeaway

You do **not** need to memorize every function or argument. Instead, focus on:

- Understanding what a function expects as input
- Knowing when to use named arguments
- Reading help files when unsure

These habits will make your R code more readable, more reliable, and easier to debug.

3.8 What's next

In the next chapter, we turn to **working with course data**.

You will learn how to:

- inspect and understand real datasets used in the course,
- distinguish between numeric and factor variables,
- identify missing or problematic values, and
- perform basic data manipulation using the `dplyr` package.

These skills will allow you to move from isolated examples to analyzing complete datasets and will serve as the foundation for descriptive analysis, visualization, and modeling later in the semester.