

# Lab 3: Testing

## Introduction to Statistical Computing

Elena Conderana & Sergio Cuenca

24/02/2025

### Contents

Installing and loading packages	2
Q1. Unitary tests	2
Q2. Using a test file	3
Q3. Complicate things	3
Q4. Code coverage	4

This lab is to be done outside of class time. You may collaborate with one classmate, but you must identify yourself and his/her name above, in the author's field, and you must submit **your own** lab as this completed .Rmd file.

## Installing and loading packages

In order to perform the exercises in this practice you should install and load the `testthat` and the `covr` package.

```
install.packages("testthat")
install.packages("covr")
```

Load the installed packages.

```
library(testthat)
library(covr)
```

## Q1. Unitary tests

1a. Let's begin with an easy one. You have the following function to convert from Celsius to Fahrenheit. Just check that it works by using an unitary test using the `testthat` package. You may need to see the help for the function `?test_that`:

```
C_to_F <- function(C_temp){
  F_temp <- (C_temp * 9/5) + 32;
  return(F_temp);
}

test_that("Celsius to Fahrenheit correct", {
  expect_equal(C_to_F(0), 32)
  expect_equal(C_to_F(100), 212)
  expect_equal(C_to_F(-40), -40)
  expect_equal(C_to_F(37), 98.6, tolerance = 0.1)
})
```

## Test passed

1b. You just have learned to test a function. Review the concept with the function to convert from Fahrenheit to Celsius:

```
F_to_C <- function(F_temp){
  C_temp <- (F_temp - 32) * 5/9;
  return(C_temp);
}

test_that("Fahrenheit to Celsius correct", {
  expect_equal(F_to_C(32), 0)
  expect_equal(F_to_C(212), 100)
  expect_equal(F_to_C(-40), -40)
  expect_equal(F_to_C(98.6), 37, tolerance = 0.1)
})
```

```
## Test passed
```

## Q2. Using a test file

**2a.** Test functions can be saved like any other R script file (with a .R extension), but with one caveat. A test R script should start with the prefix 'test-'. A good way of doing this is to add the prefix to the name of file that stores the functions to be tested. In this case, your teacher have included the `C_to_F` and `F_to_C` function to the `temp_conversion.R` file. You shall include the tests created in the previous section to the file `test-temp_conversion.R`. After that, run the following command to run all the test files included in the working directory:

```
test_dir(".")

## v | F W S OK | Context
## / | 0 | my_awesome_function / | 0 | My Awesome Function testing v | 7 | My
Awesome Function testing
## / | 0 | number_utility / | 0 | number_utility Tests v | 18 | number_utility
Tests
## / | 0 | temp_conversion / | 0 | Temperature function testing v | 9 |
Temperature function testing
##
## == Results
=====
## [ FAIL 0 | WARN 0 | SKIP 0 | PASS 34 ]
```

**2b.** In the test file provided there is a failing test. Correct it and check that the `test_dir(".")` command now runs smoothly.

```
test_that("Fahrenheit to Celsius correct", {
  temp_F <- F_to_C(50)
  expect_equal(temp_F, 10)
})
```

```
## Test passed
```

## Q3. Complicate things

**3a.** Now that you have the setup done, create a file called `test-my_awesome_function.R` with tests for the function `my_awesome_function()` included in `my_awesome_function.R` so that they: \* Try the function with erroneous data type, for example, using a string instead of a number. \* Try the function with vectorized inputs. \* Check that the results are numeric.

After that, modify the function in order than none of this scenarios produce an error in the tests. You might need to include `stop()` statements in failing cases. Try the tests using the `test_file` function:

```
test_file("test-my_awesome_function.R")

## [ FAIL 0 | WARN 0 | SKIP 0 | PASS 0 ][ FAIL 0 | WARN 0 | SKIP 0 | PASS 0 ][
FAIL 0 | WARN 0 | SKIP 0 | PASS 1 ][ FAIL 0 | WARN 0 | SKIP 0 | PASS 2 ][ FAIL
0 | WARN 0 | SKIP 0 | PASS 3 ][ FAIL 0 | WARN 0 | SKIP 0 | PASS 4 ][ FAIL 0 |
WARN 0 | SKIP 0 | PASS 5 ][ FAIL 0 | WARN 0 | SKIP 0 | PASS 6 ][ FAIL 0 | WARN
0 | SKIP 0 | PASS 7 ]
```

## Q4. Code coverage

4a. Now that you know how to modify tests, let's obtain the code coverage for the `number_utility.R` file. In order to do that, we will use the `covr` package to obtain a code coverage report of this file using the test file `test-number_utility.R`.

```
# Constants
number_utility <- "number_utility"
extension_r <- ".R"
extension_htm <- ".htm"

# Manage file names
code_file_name <- paste(number_utility, extension_r, sep="")
test_file_name <- paste("test-", code_file_name, sep="")
coverage_report_file_name <- paste("coverage_report_", number_utility, extension_htm, sep="")

# Run tests and generate Code Coverage Report
test_file(test_file_name)
```

```
## [ FAIL 0 | WARN 0 | SKIP 0 | PASS 0 ][ FAIL 0 | WARN 0 | SKIP 0 | PASS 0 ][
FAIL 0 | WARN 0 | SKIP 0 | PASS 1 ][ FAIL 0 | WARN 0 | SKIP 0 | PASS 2 ][ FAIL
0 | WARN 0 | SKIP 0 | PASS 3 ][ FAIL 0 | WARN 0 | SKIP 0 | PASS 4 ][ FAIL 0 |
WARN 0 | SKIP 0 | PASS 5 ][ FAIL 0 | WARN 0 | SKIP 0 | PASS 6 ][ FAIL 0 | WARN
0 | SKIP 0 | PASS 7 ][ FAIL 0 | WARN 0 | SKIP 0 | PASS 8 ][ FAIL 0 | WARN 0 |
SKIP 0 | PASS 9 ][ FAIL 0 | WARN 0 | SKIP 0 | PASS 10 ][ FAIL 0 | WARN 0 |
SKIP 0 | PASS 11 ][ FAIL 0 | WARN 0 | SKIP 0 | PASS 12 ][ FAIL 0 | WARN 0 |
SKIP 0 | PASS 13 ][ FAIL 0 | WARN 0 | SKIP 0 | PASS 14 ][ FAIL 0 | WARN 0 |
SKIP 0 | PASS 15 ][ FAIL 0 | WARN 0 | SKIP 0 | PASS 16 ][ FAIL 0 | WARN 0 |
SKIP 0 | PASS 17 ][ FAIL 0 | WARN 0 | SKIP 0 | PASS 18 ]
```

```
res <- file_coverage(code_file_name, test_file_name)
```

```
## Test passed
## Test passed
## Test passed
## Test passed
## Test passed
## Test passed
## Test passed
## Test passed
```

```
print(res)
```

```
## Coverage: 100.00%
```

```
## number_utility.R: 100.00%
```

```
report(res, coverage_report_file_name)
```

The code coverage is 100%.