

Unit Testing: Cover all bases

Testing Scope

- Functional (as specified in the requirements)
- Non-functional
 - Usability, graphical appearance
 - Scalability, performance
 - Compatibility, portability
 - Reliability

Testing can be:

- Static (e.g., proofreading, reviews, verification)
- Dynamic (e.g., automated unit tests)

Unit tests

What are they?

A test is a series of expectation functions, which collectively test one small unit of functionality.

Why should we use them?

Typically, we test code to ensure its output meets our expectations and to assure others that our code works correctly.

- Today: To identify bugs in your current code.
- Future: To make current package functionality robust to future changes.

When creating an R package, unit tests can be included. These will be run automatically whenever the package is built and checked. In order to do so, a framework for the tests must be created.

There are multiple options to do so:

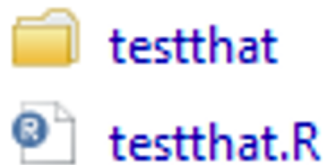
RUnit package

testthat package

Unit testing framework

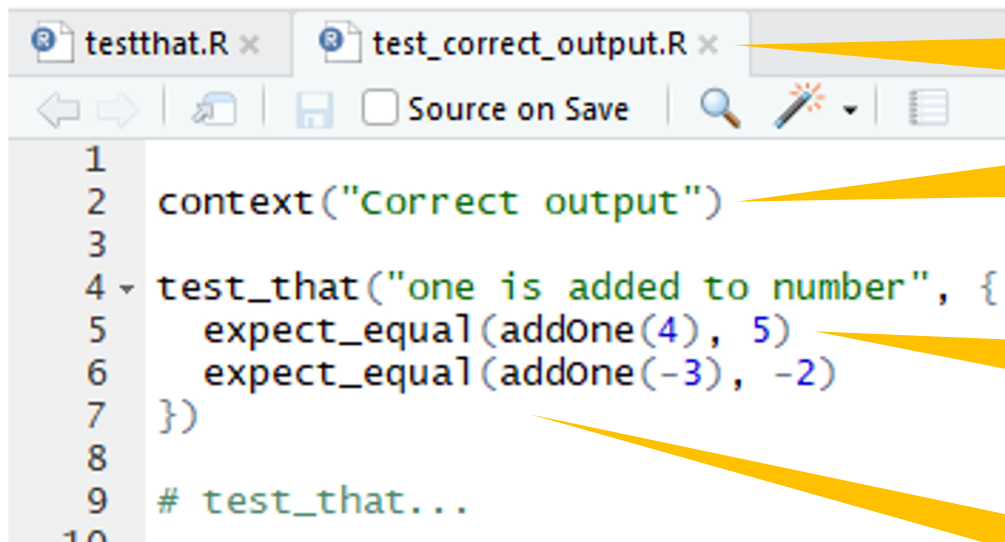
The testing framework can be created using the `use_testthat()` function from the `usethis` package (`usethis::use_testthat()`). This creates a directory in the package called 'tests', which contains:

- An R script called 'testthat.R', which contains the code for running the tests.
- A directory called 'testthat'. All tests must be contained in R scripts in this directory, with file names beginning 'test'.



Test example

A test is created using the `test_that()` function. The first argument is the name of the test (as a character string), while the second argument contains the test code, including all of the expectation statements, between curly brackets. When this function is run, the test code will be evaluated in its own environment. An error will be produced if any of the test code produces an error.



```
1  
2 context("Correct output")  
3  
4 test_that("one is added to number", {  
5   expect_equal(addone(4), 5)  
6   expect_equal(addone(-3), -2)  
7 })  
8  
9 # test_that...  
10
```

Located in "testthat" directory and begins with the word "test" so it can be recognized as a test.

Provides a brief description of what the file is testing.

Expectations are the atom of testing. They describe the expected result.

A test groups together expectations that are related.

Expectation functions in testthat

- These functions form the core of the testing framework. They all have the prefix `expect_`.
- Several of these functions take an object to be tested as the first argument, and check some aspect of it against a reference value, given as the second argument.
- If the two values do not match, an error is produced. (If they do match, the function invisibly returns the object being tested.)

`expect_equal()` compares a value of an object `a` to a given reference value.

```
a <- 3 + 2

expect_equal(a, 5) # Runs without error

expect_equal(a, 6) # Produces an error:
# Error: `a` not equal to 6.
# 1/1 mismatches
# [1] 5 - 6 == -1
```

Similarly, you can test based on an object class, error messages, warnings...

► `expect_is`, which checks the class of an object.

```
val <- 43.7
expect_is(val, "numeric") # Runs without error
expect_is(val, "character") # Produces an error
```

► `expect_length`, which checks whether the object has a given length.

```
vec <- c(1, 7, 6, 4, 9)
expect_length(vec, 5) # Runs without error
expect_length(vec, 8) # Produces an error
```

...or the length of an object.

Run Tests

- To run the tests, the `test()` function from the `devtools` package may be used.
- When checking a built package using R CMD check, any tests included in the package will be run automatically, alongside all other usual package checks. If any tests fail, an error message will be produced.

```
> devtools::test()
Loading TestFramework
Testing TestFramework
√ | OK F W S | Context
x | 2 1      | Correct output
-----
test_correct_output.R:7: error: one is added to number
non-numeric argument to binary operator
1: expect_equal(addOne("cat"), -1) at C:\Users\lweibel\Desktop\R - LE9 - Debugging &
stFramework/tests/testthat/test_correct_output.R:7
2: quasi_label(enquo(object), label)
3: eval_bare(get_expr(quo), get_env(quo))
4: addOne("cat")
-----
== Results ==
OK:      2
Failed:  1
Warnings: 0
Skipped: 0
```

Now it's your turn!