R PACKAGE DEVELOPMENT AND VALIDATION

Package Elements
& Structure

Welcome

- Functions
- Unit Testing
- Building a package

FUNCTIONS

UDFs (User defined functions)

1. Functions help keep your code DRY
2. Functions abstract concepts
3. Functions are testable

Don’t Repeat Yourself

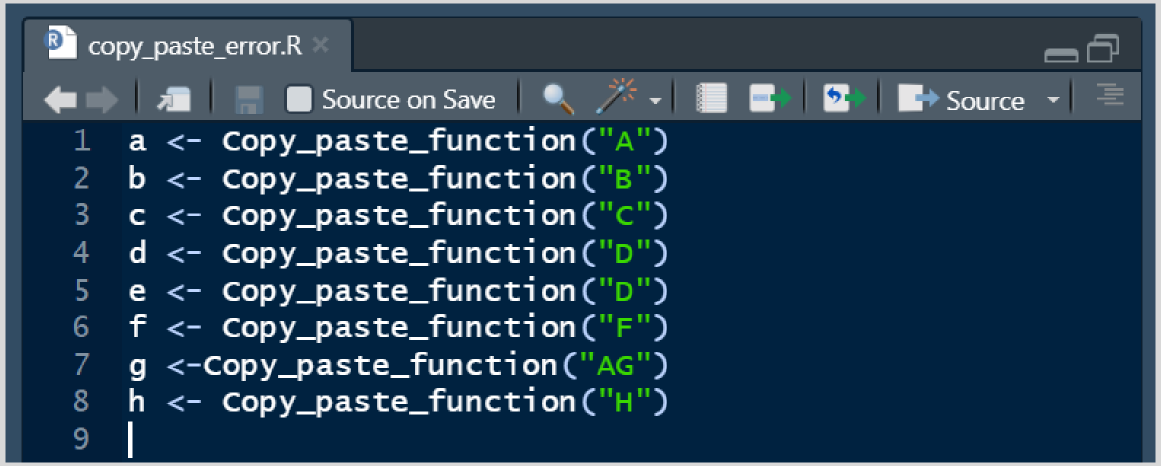
Don’t Repeat Yourself

Don’t Repeat Yourself

DRY

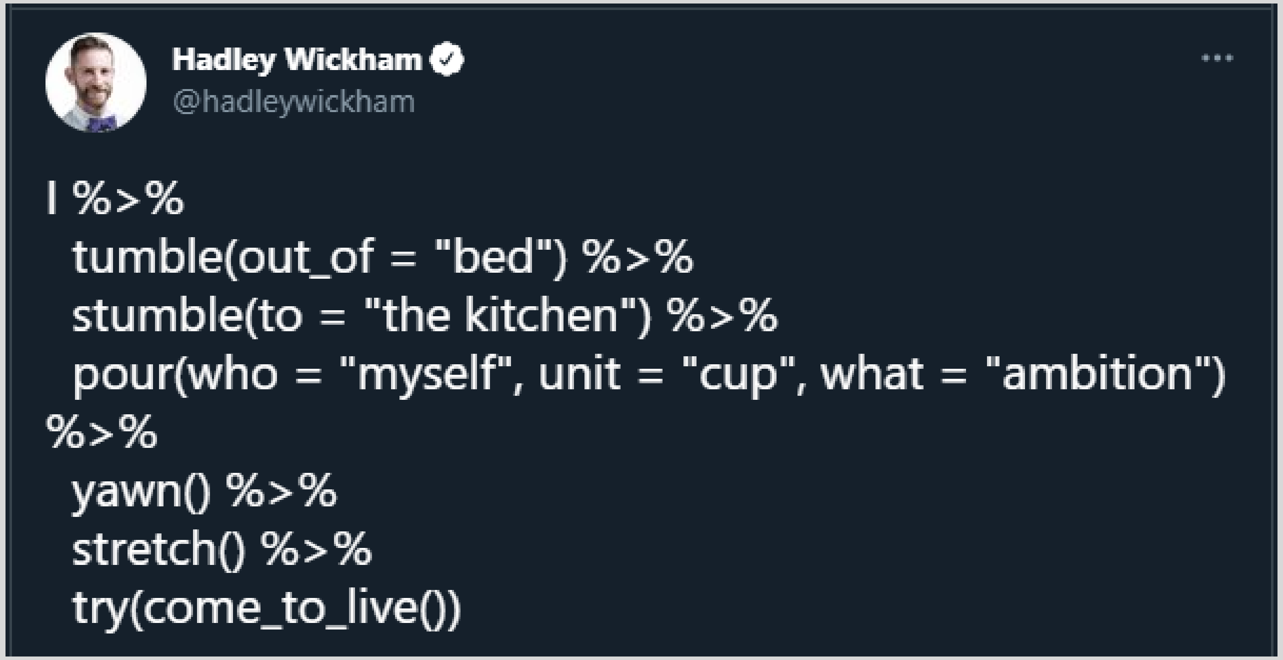
DRY

1. Copy Paste Errors
2. Updating errors
3. Propagation of updates



```r
a <- Copy_paste_function("A")
b <- Copy_paste_function("B")
c <- Copy_paste_function("C")
d <- Copy_paste_function("D")
e <- Copy_paste_function("D")
f <- Copy_paste_function("F")
g <-Copy_paste_function("AG")
h <- Copy_paste_function("H")

```

Abstracting Concepts


Hadley Wickham
@hadleywickham

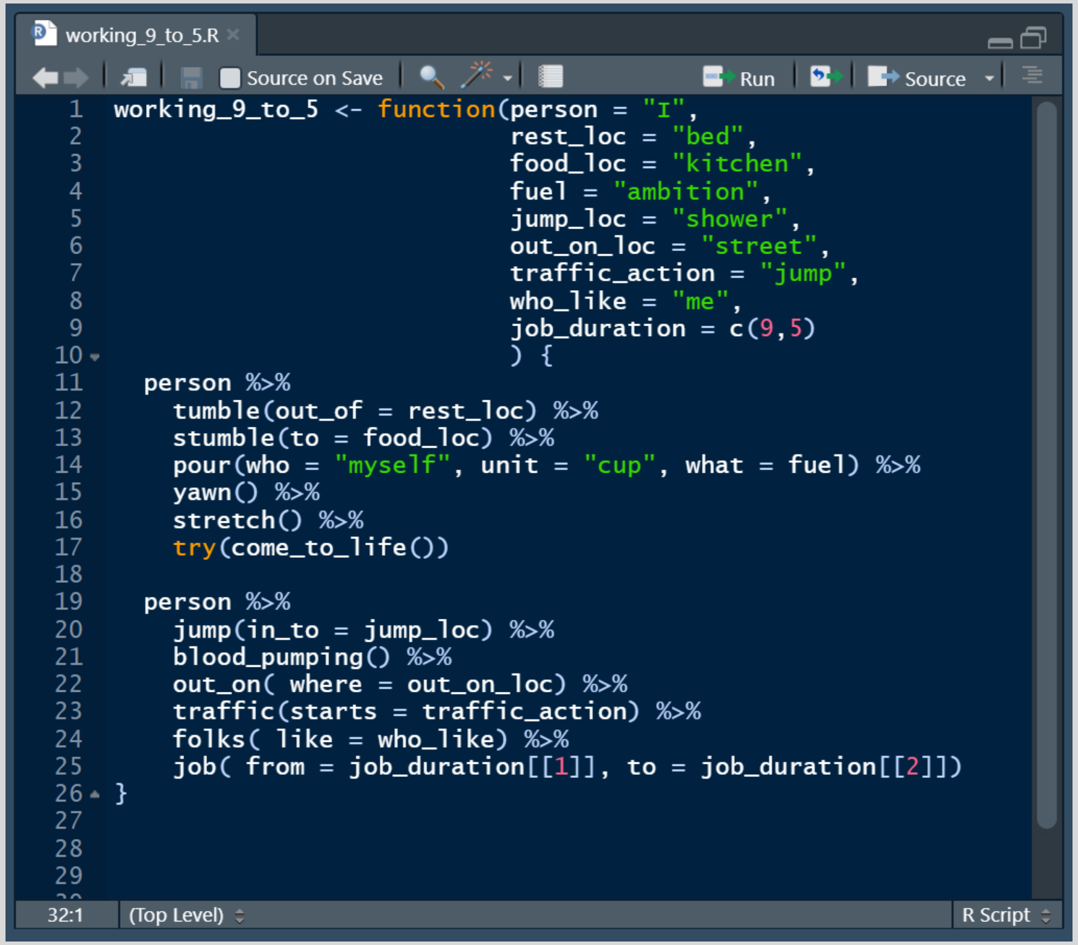
```
I %>%
  tumble(out_of = "bed") %>%
  stumble(to = "the kitchen") %>%
  pour(who = "myself", unit = "cup", what = "ambition")
%>%
  yawn() %>%
  stretch() %>%
  try(come_to_live())
```

Quote Tweet from Hadley Wickham (@hadleywickham)
February 11, 2021

https://twitter.com/hadleywickham/status/1359852563726819332

Abstracting Concepts

- Functions allow you to abstract processes
- Functions can be built from other functions
- Functions can be used for specific returned objects or side effects
  - Side effects: Plots, changes to environment or object



```r
working_9_to_5 <- function(person = "I",
                           rest_loc = "bed",
                           food_loc = "kitchen",
                           fuel = "ambition",
                           jump_loc = "shower",
                           out_on_loc = "street",
                           traffic_action = "jump",
                           who_like = "me",
                           job_duration = c(9,5)
                           ) {
  person %>%
    tumble(out_of = rest_loc) %>%
    stumble(to = food_loc) %>%
    pour(who = "myself", unit = "cup", what = fuel) %>%
    yawn() %>%
    stretch() %>%
    try(come_to_life())

  person %>%
    jump(in_to = jump_loc) %>%
    blood_pumping() %>%
    out_on( where = out_on_loc) %>%
    traffic(starts = traffic_action) %>%
    folks( like = who_like) %>%
    job( from = job_duration[[1]], to = job_duration[[2]])
}
```

Testable

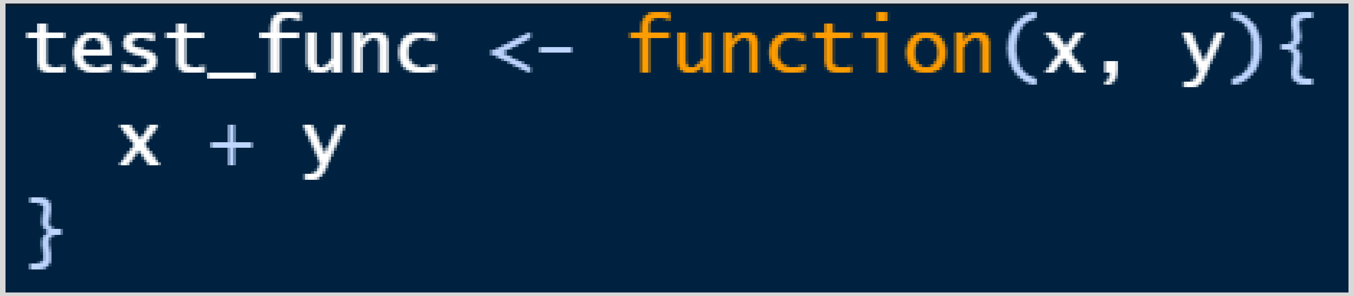
- Specific behavior
- Known input means known output
- Reproducible

UNIT TESTING

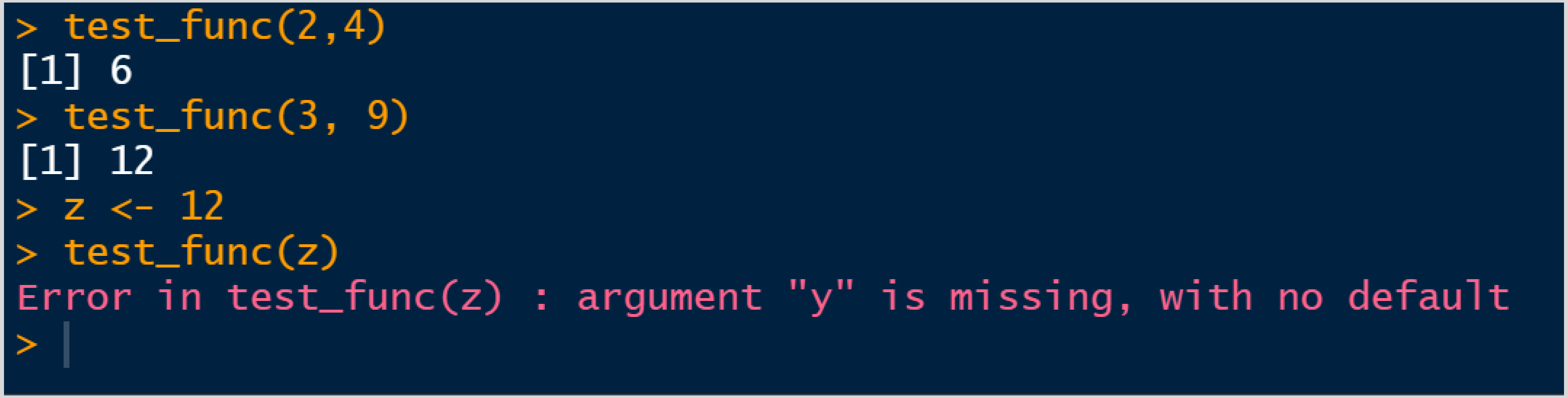
Unit Testing

1. Test to show code behave as function author expects
2. Repeatable to show code behavior does not change over time

Unit Testing



```
test_func <- function(x, y){
  x + y
}
```



```
> test_func(2,4)
[1] 6
> test_func(3, 9)
[1] 12
> z <- 12
> test_func(z)
Error in test_func(z) : argument "y" is missing, with no default
> 
```

🎉 Congratulations!

You were **UNIT TESTING!**

🎉 Congratulations!

You were **Manually UNIT TESTING!**

Test Function behavior as unit tests

- Formalize the testing being done
  - Units
- Given an input, function returns an expected output
  - Expectations

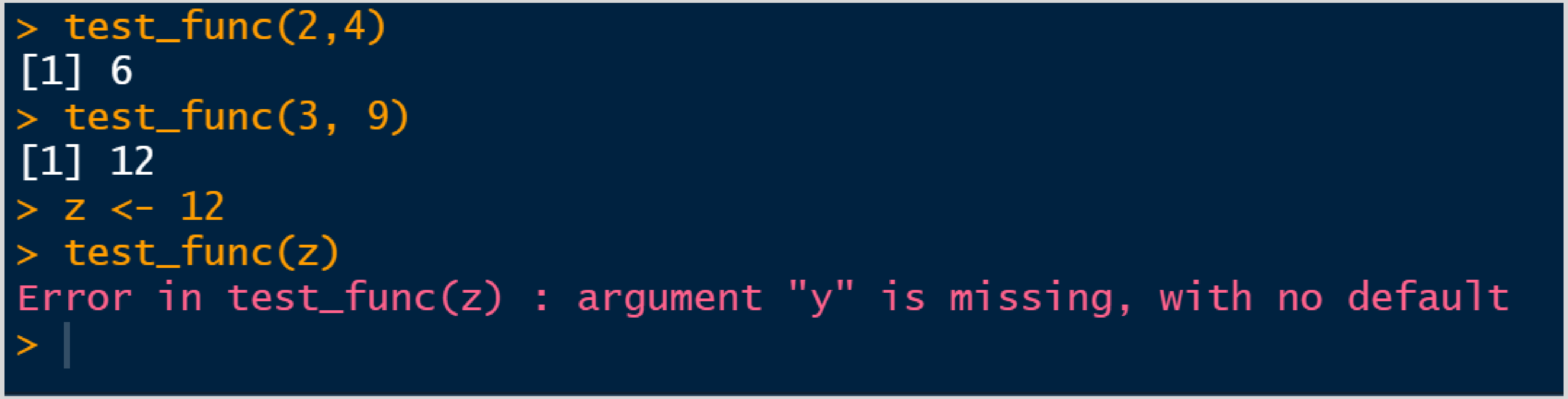
Testthat unit testing framework

Units ➔ test_that()

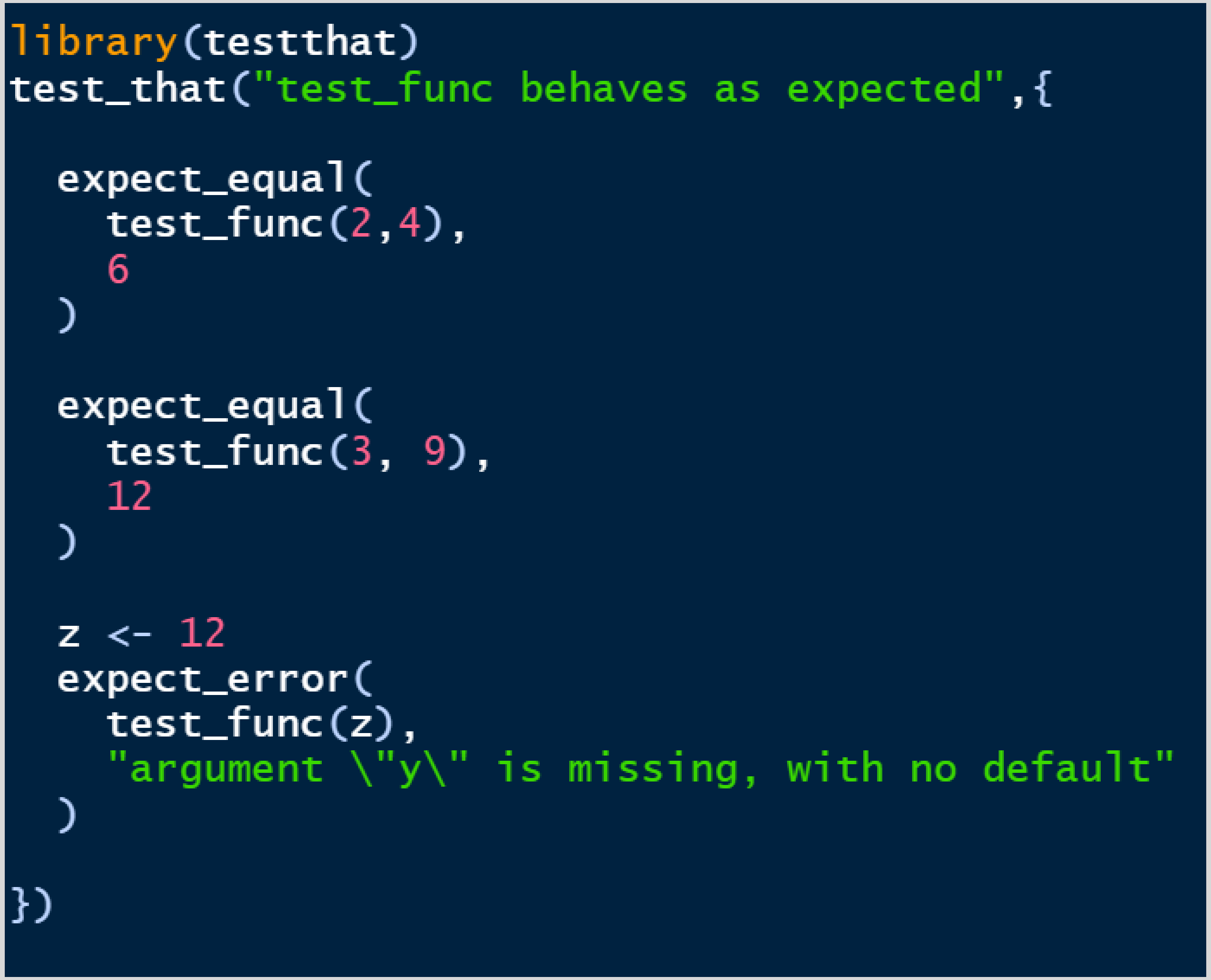
Expectations ➔ expect_* family of functions

- expect_equal()
- expect_true() / expect_false()
- expect_error()
- expect_warning()
- https://testthat.r-lib.org/reference/index.html - full list of expectations

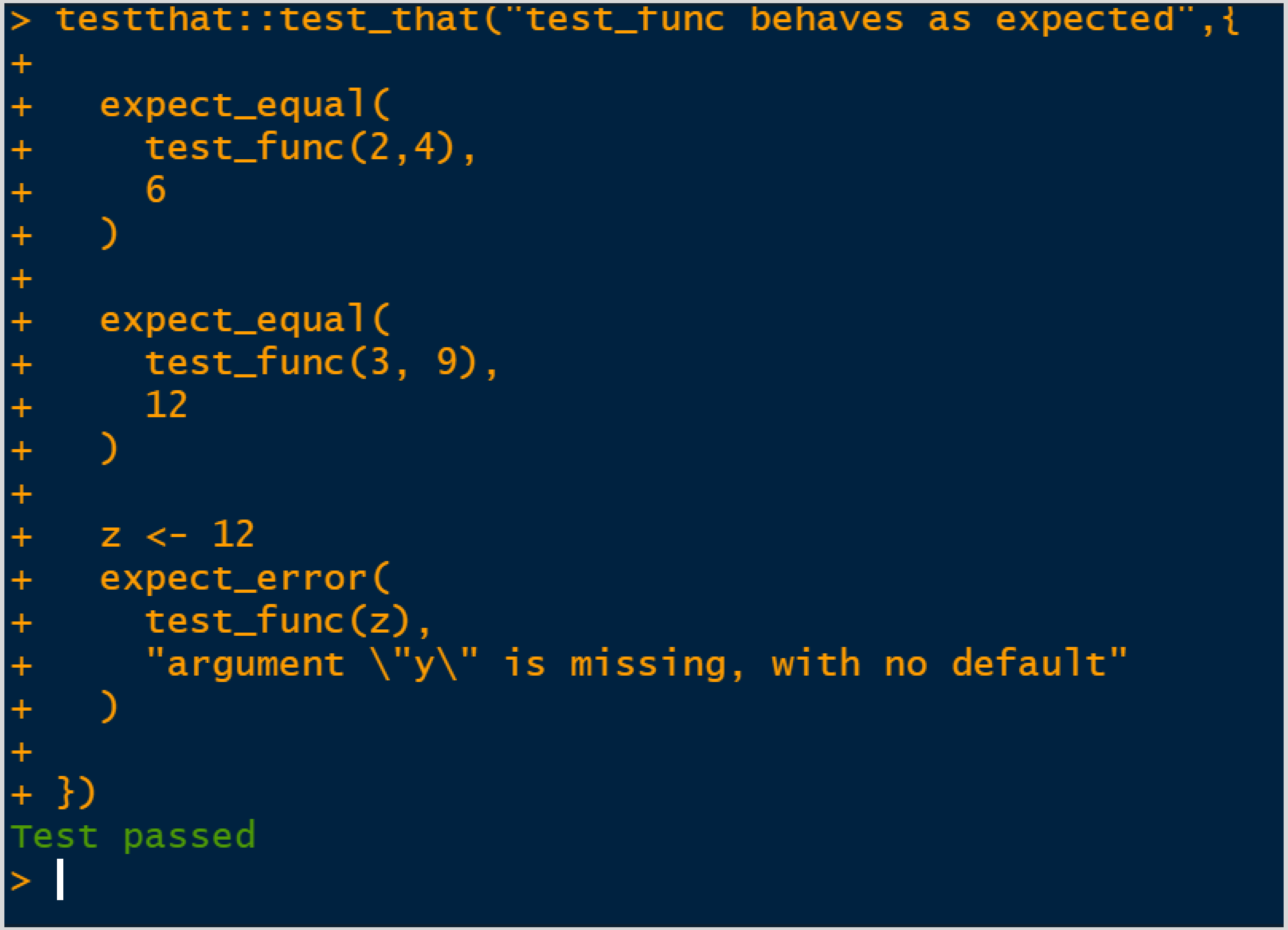
Formalize Testing



```
> test_func(2,4)
[1] 6
> test_func(3, 9)
[1] 12
> z <- 12
> test_func(z)
Error in test_func(z) : argument "y" is missing, with no default
> 
```

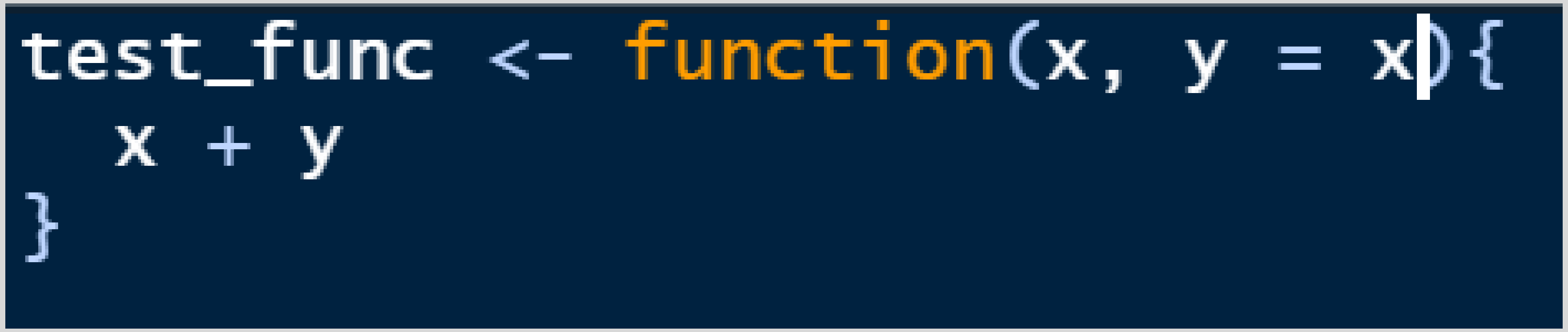


```r
library(testthat)
test_that("test_func behaves as expected",{

  expect_equal(
    test_func(2,4),
    6
  )

  expect_equal(
    test_func(3, 9),
    12
  )

  z <- 12
  expect_error(
    test_func(z),
    "argument \"y\" is missing, with no default"
  )

})
```

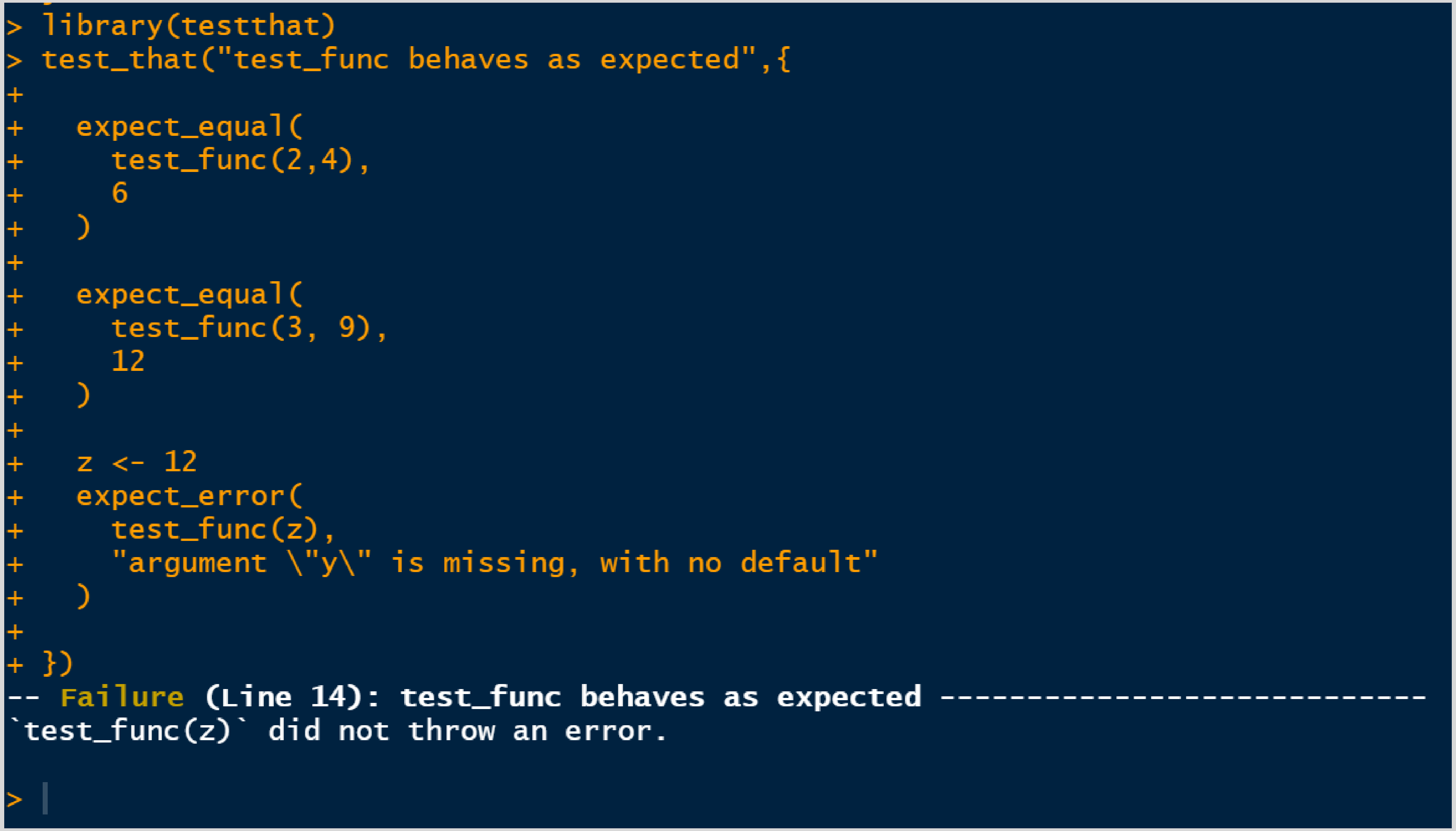

```
> testthat::test_that("test_func behaves as expected",{
+ 
+   expect_equal(
+     test_func(2,4),
+     6
+   )
+ 
+   expect_equal(
+     test_func(3, 9),
+     12
+   )
+ 
+   z <- 12
+   expect_error(
+     test_func(z),
+     "argument \"y\" is missing, with no default"
+   )
+ 
+ })
Test passed
> 
```

Repeatably Test Function when it updates

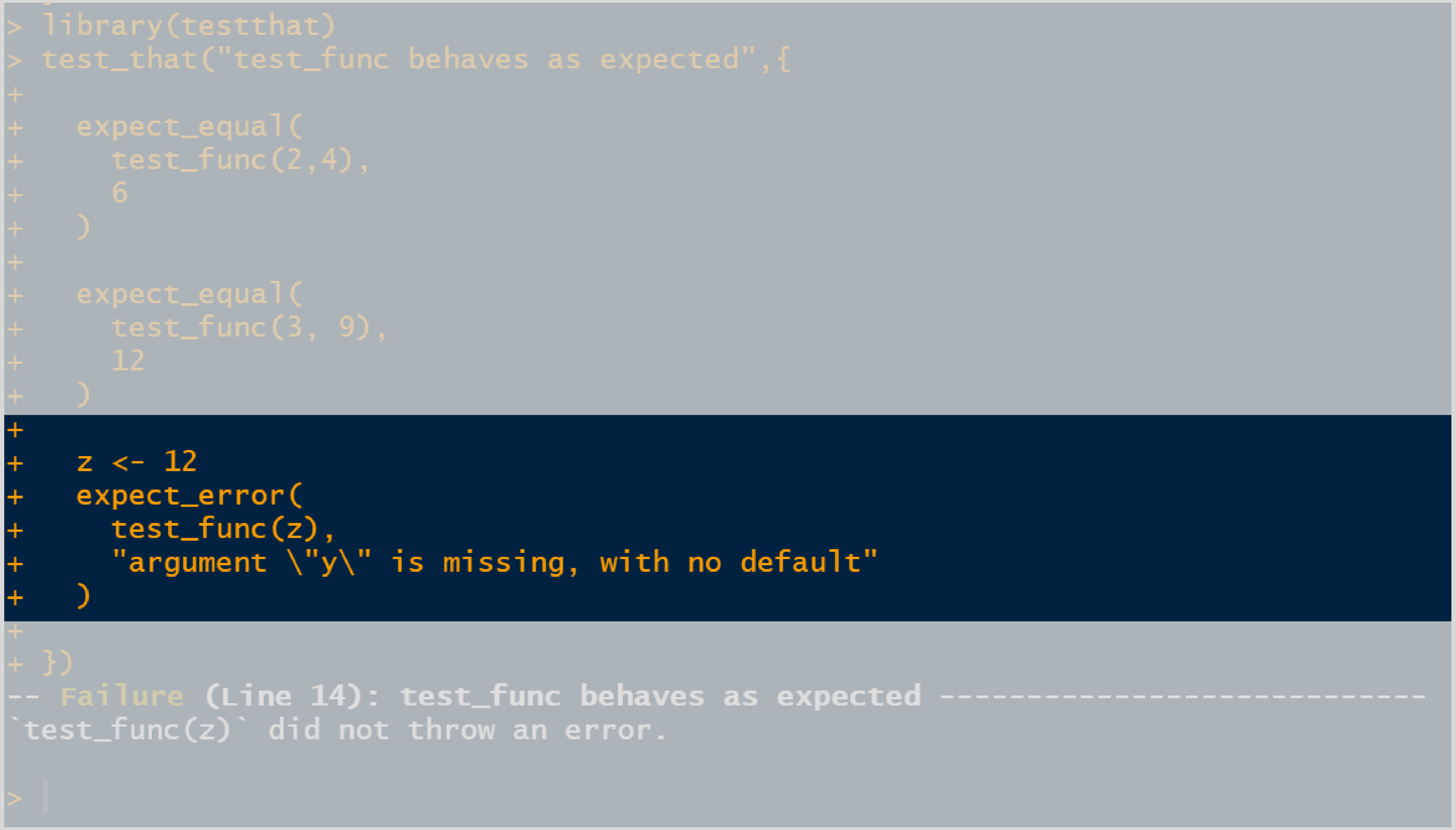
- When you update the function, you can now test that your expectations still hold!



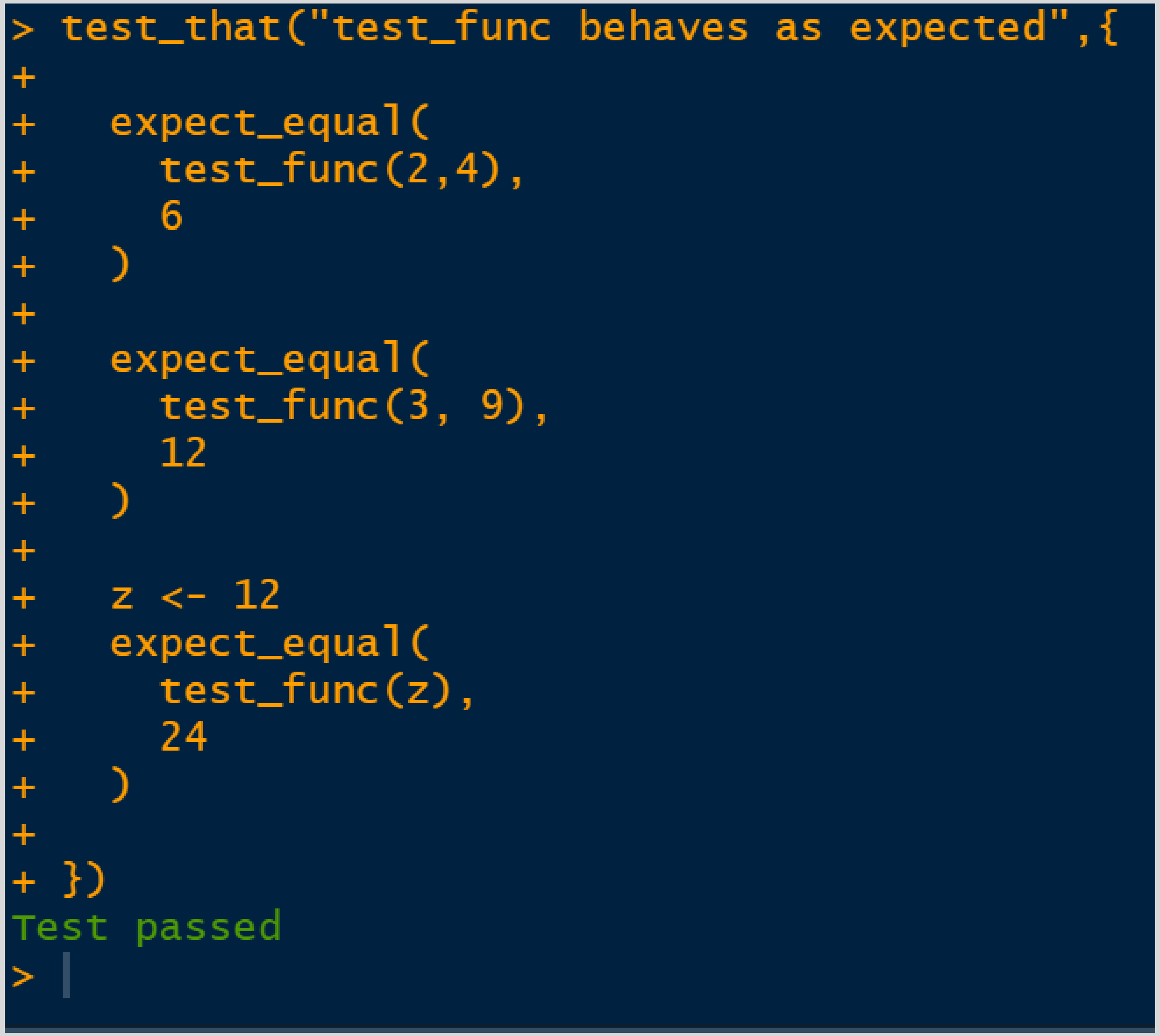
```
test_func <- function(x, y = x){
  x + y
}
```



```
> library(testthat)
> test_that("test_func behaves as expected",{
+
+   expect_equal(
+     test_func(2,4),
+     6
+   )
+
+   expect_equal(
+     test_func(3, 9),
+     12
+   )
+
+   z <- 12
+   expect_error(
+     test_func(z),
+     "argument \"y\" is missing, with no default"
+   )
+
+ })
-- Failure (Line 14): test_func behaves as expected ----------------------------
`test_func(z)` did not throw an error.

> 
```



```
> library(testthat)
> test_that("test_func behaves as expected",{
+ 
+   expect_equal(
+     test_func(2,4),
+     6
+   )
+ 
+   expect_equal(
+     test_func(3, 9),
+     12
+   )
+ 
+   z <- 12
+   expect_error(
+     test_func(z),
+     "argument \"y\" is missing, with no default"
+   )
+ 
+ })
-- Failure (Line 14): test_func behaves as expected ----------------------------
`test_func(z)` did not throw an error.

> 
```



```
> test_that("test_func behaves as expected",{
+ 
+   expect_equal(
+     test_func(2,4),
+     6
+   )
+ 
+   expect_equal(
+     test_func(3, 9),
+     12
+   )
+ 
+   z <- 12
+   expect_equal(
+     test_func(z),
+     24
+   )
+ 
+ })
Test passed
> 
```

WHATS A PACKAGE

A collection of functions with a set of conventions that is intended to make code reusable and shareable

MAKING YOUR PACKAGE

usethis::create_package()

DESCRIPTION File	Basic package information – name, version, authors, dependencies
NAMESPACE File	What information does R need to know – Functions that are shared, dependencies, methods
R Folder (/R)	Folder to house the R code of the package

.Rbuildignore, .gitignore, .Rproj.user

usethis::use_r()

Creates a new R script

Within /R

Organize Code within R Script to be related

Give a meaningful name to the script

```
usethis::use_r("new_file_name")
```

usethis::use_test()

Creates a new test script for unit testing

Unit testing allows the developer to repeatedly test across all expectations

Creates tests/testthat folders

Creates new test within tests/testthat folder

Organize tests within to be related

```
usethis::use_test(“new-test-name”)
```

usethis

- Automates workflows and creation of files and folders done as part of package development
  - Creating the R package skeleton – usethis::create_package()
  - Adding a license – usethis::use_license()
  - Create a new R script – usethis::use_r()
  - Add package to DESCRIPTION file – usethis::use_package()
  - Create a new test – usethis::use_test()
  - Create Vignette – useths::use_vignette()

usethis.r-lib.org

MATERIALS 02-E1

Materials/Materials-02-Package_Elements_and_Structure/Materials-02-E01-Package_Basics/