Behaviour Driven Development

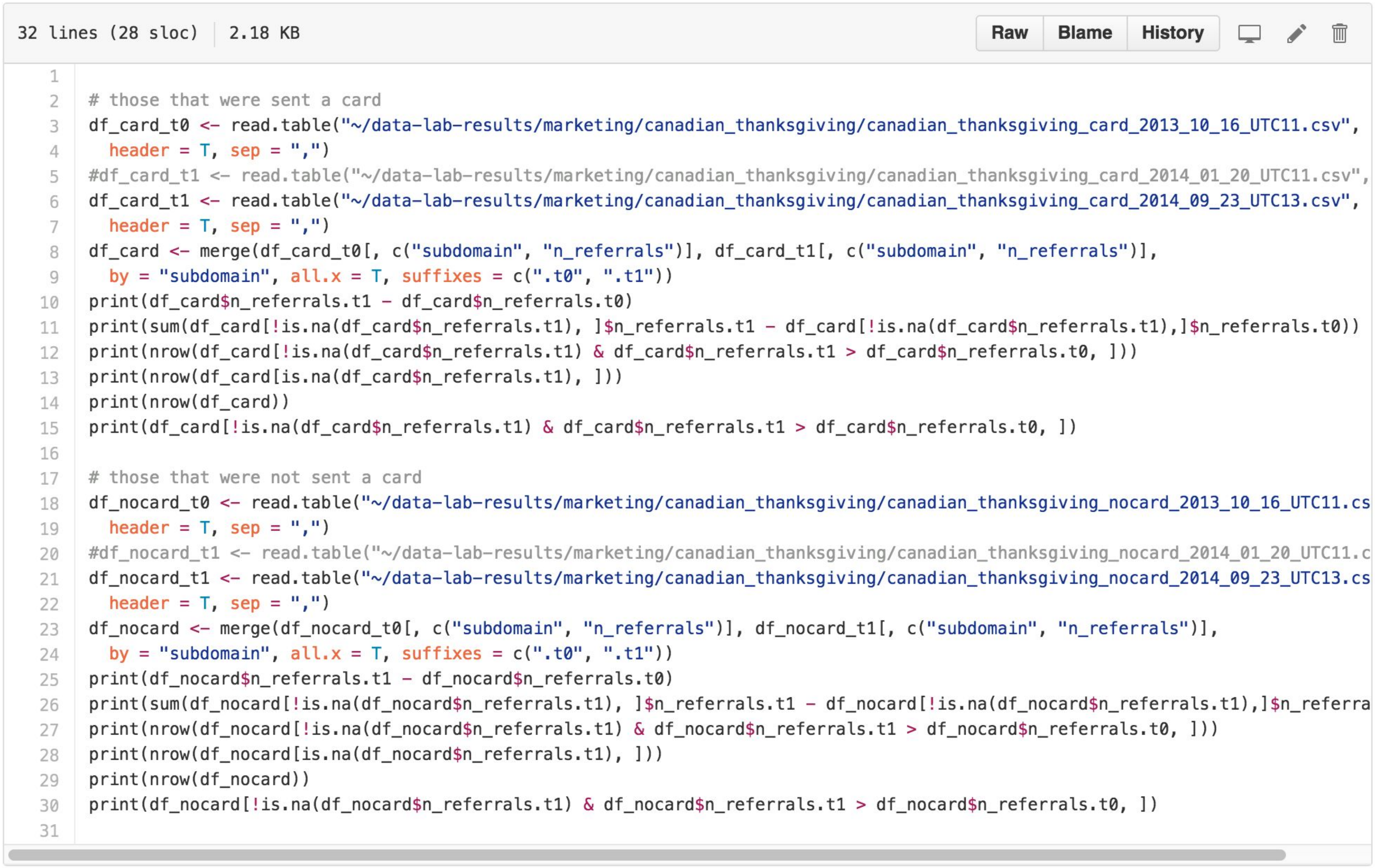
Dave Evans
EdinbR - 2016/02/17

Introduction

Worked around eight years on data analysis at an LHC experiment called CMS, involving substantial amounts of computer programming:

- Core reconstruction software
- User analysis workload management software
- Data analysis code as part of several high profile analyses

None of it was tested!


32 lines (28 sloc) | 2.18 KB

```r

# those that were sent a card
df_card_t0 <- read.table("~/data-lab-results/marketing/canadian_thanksgiving/canadian_thanksgiving_card_2013_10_16_UTC11.csv",
  header = T, sep = ",")
#df_card_t1 <- read.table("~/data-lab-results/marketing/canadian_thanksgiving/canadian_thanksgiving_card_2014_01_20_UTC11.csv",
df_card_t1 <- read.table("~/data-lab-results/marketing/canadian_thanksgiving/canadian_thanksgiving_card_2014_09_23_UTC13.csv",
  header = T, sep = ",")
df_card <- merge(df_card_t0[, c("subdomain", "n_referrals")], df_card_t1[, c("subdomain", "n_referrals")],
  by = "subdomain", all.x = T, suffixes = c(".t0", ".t1"))
print(df_card$n_referrals.t1 - df_card$n_referrals.t0)
print(sum(df_card[!is.na(df_card$n_referrals.t1), ]$n_referrals.t1 - df_card[!is.na(df_card$n_referrals.t1),]$n_referrals.t0))
print(nrow(df_card[!is.na(df_card$n_referrals.t1) & df_card$n_referrals.t1 > df_card$n_referrals.t0, ]))
print(nrow(df_card[is.na(df_card$n_referrals.t1), ]))
print(nrow(df_card))
print(df_card[!is.na(df_card$n_referrals.t1) & df_card$n_referrals.t1 > df_card$n_referrals.t0, ])

# those that were not sent a card
df_nocard_t0 <- read.table("~/data-lab-results/marketing/canadian_thanksgiving/canadian_thanksgiving_nocard_2013_10_16_UTC11.cs
  header = T, sep = ",")
#df_nocard_t1 <- read.table("~/data-lab-results/marketing/canadian_thanksgiving/canadian_thanksgiving_nocard_2014_01_20_UTC11.c
df_nocard_t1 <- read.table("~/data-lab-results/marketing/canadian_thanksgiving/canadian_thanksgiving_nocard_2014_09_23_UTC13.cs
  header = T, sep = ",")
df_nocard <- merge(df_nocard_t0[, c("subdomain", "n_referrals")], df_nocard_t1[, c("subdomain", "n_referrals")],
  by = "subdomain", all.x = T, suffixes = c(".t0", ".t1"))
print(df_nocard$n_referrals.t1 - df_nocard$n_referrals.t0)
print(sum(df_nocard[!is.na(df_nocard$n_referrals.t1), ]$n_referrals.t1 - df_nocard[!is.na(df_nocard$n_referrals.t1),]$n_referra
print(nrow(df_nocard[!is.na(df_nocard$n_referrals.t1) & df_nocard$n_referrals.t1 > df_nocard$n_referrals.t0, ]))
print(nrow(df_nocard[is.na(df_nocard$n_referrals.t1), ]))
print(nrow(df_nocard))
print(df_nocard[!is.na(df_nocard$n_referrals.t1) & df_nocard$n_referrals.t1 > df_nocard$n_referrals.t0, ])

```

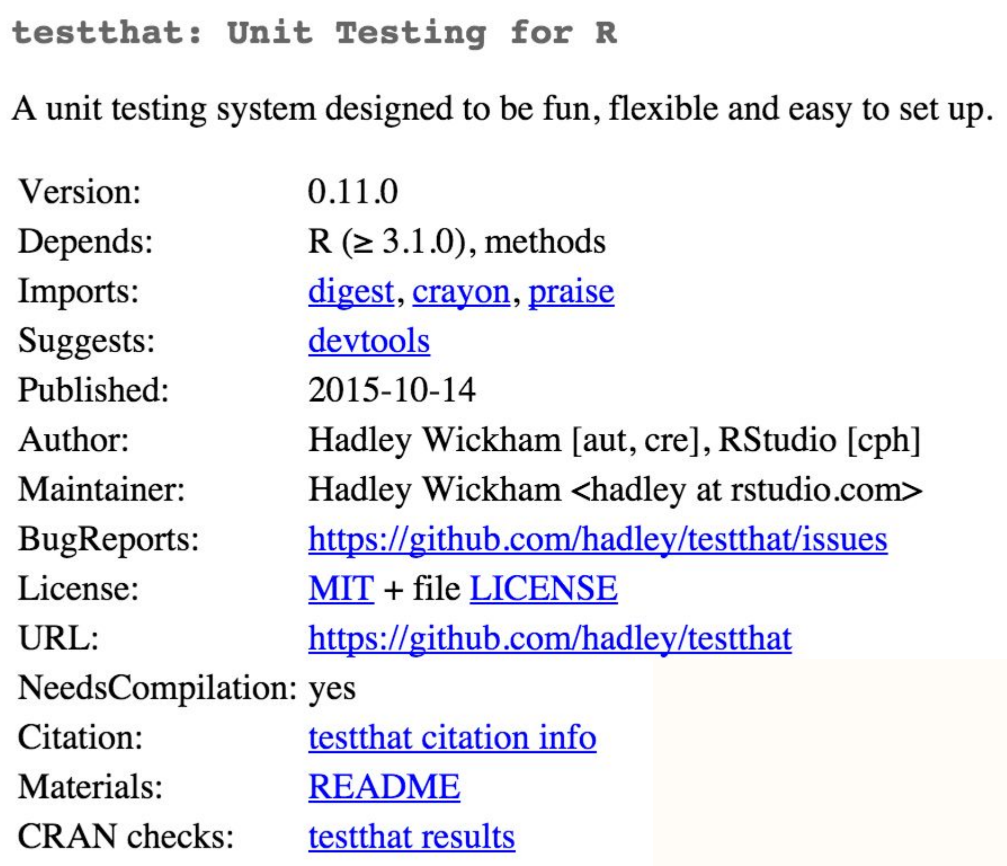
You should add some unit tests for that!

My paper draft is due tomorrow!

I’m done with this analysis…

I don’t really know how to do that.

My analysis script is like... 3000 lines.
How would I even start?



testthat: Unit Testing for R

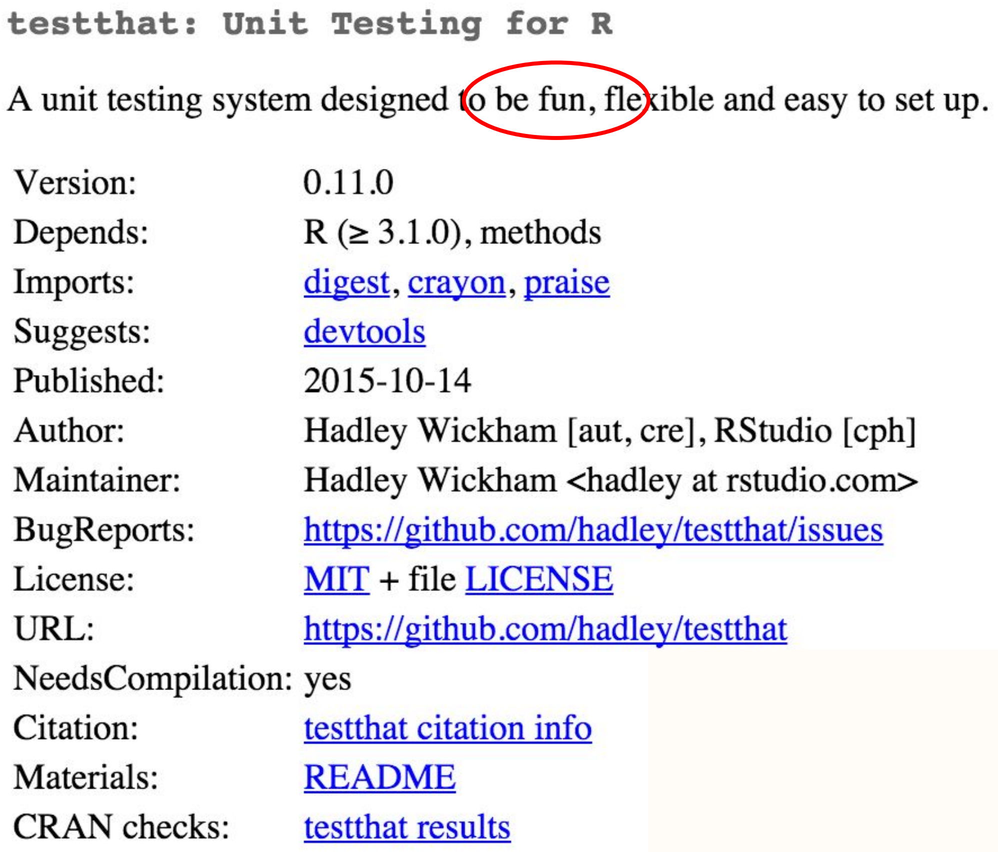
A unit testing system designed to be fun, flexible and easy to set up.

Version:	0.11.0
Depends:	R (≥ 3.1.0), methods
Imports:	digest, crayon, praise
Suggests:	devtools
Published:	2015-10-14
Author:	Hadley Wickham [aut, cre], RStudio [cph]
Maintainer:	Hadley Wickham <hadley at rstudio.com>
BugReports:	https://github.com/hadley/testthat/issues
License:	MIT + file LICENSE
URL:	https://github.com/hadley/testthat
NeedsCompilation:	yes
Citation:	testthat citation info
Materials:	README
CRAN checks:	testthat results

Jasmine

Behavior-Driven JavaScript

Behaviour Driven
Development for Ruby.
Making TDD Productive and Fun.



testthat: Unit Testing for R

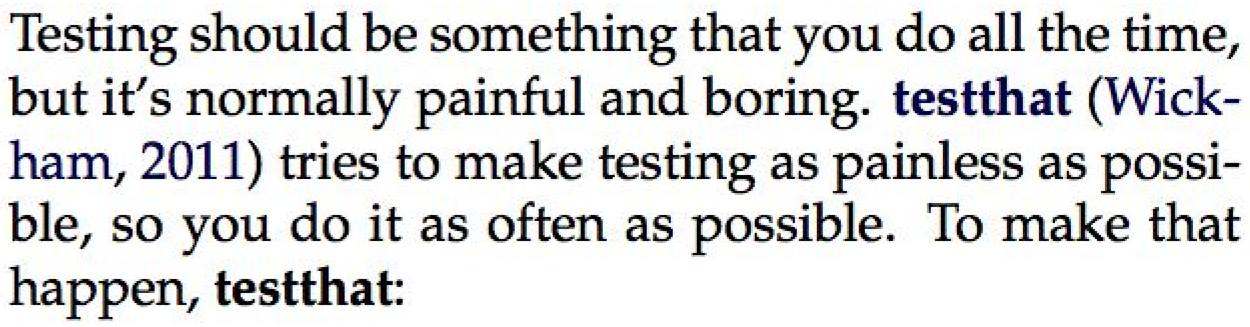
A unit testing system designed to be fun, flexible and easy to set up.

Version:	0.11.0
Depends:	R (≥ 3.1.0), methods
Imports:	digest, crayon, praise
Suggests:	devtools
Published:	2015-10-14
Author:	Hadley Wickham [aut, cre], RStudio [cph]
Maintainer:	Hadley Wickham <hadley at rstudio.com>
BugReports:	https://github.com/hadley/testthat/issues
License:	MIT + file LICENSE
URL:	https://github.com/hadley/testthat
NeedsCompilation:	yes
Citation:	testthat citation info
Materials:	README
CRAN checks:	testthat results

Jasmine

Behavior-Driven JavaScript

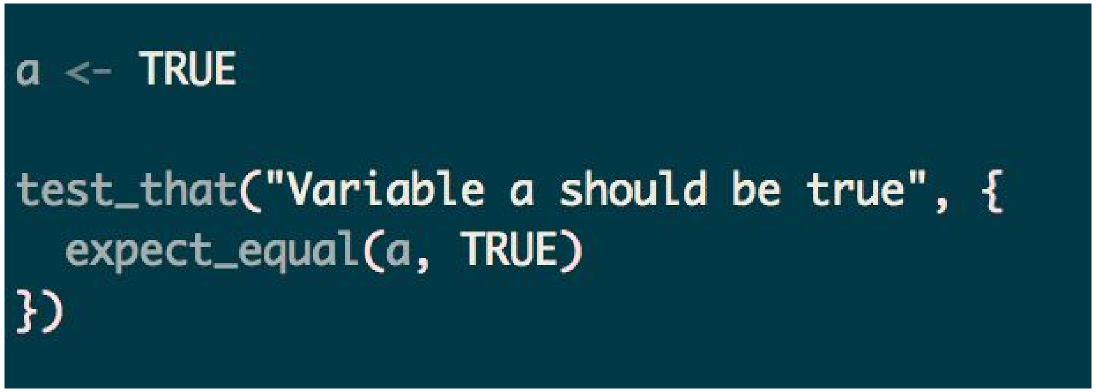
Behaviour Driven
Development for Ruby.
Making TDD Productive and Fun.



Testing should be something that you do all the time, but it's normally painful and boring. **testthat** (Wickham, 2011) tries to make testing as painless as possible, so you do it as often as possible. To make that happen, **testthat**:

https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf

Anatomy of a test



```
a <- TRUE

test_that("Variable a should be true", {
  expect_equal(a, TRUE)
})
```

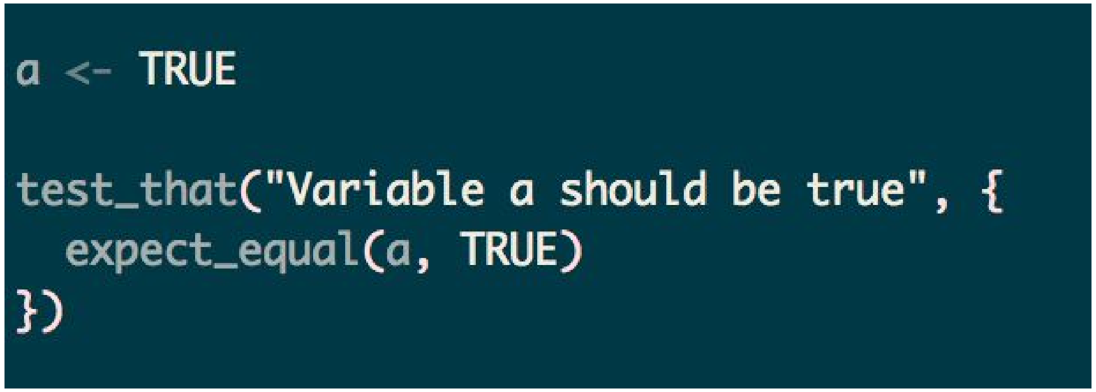
Tests should be described by sentences:

Writing this sentence uncovers confusing behaviour

What does this bit of code ***really*** do?

Is that what you wanted it to do?

Anatomy of a test

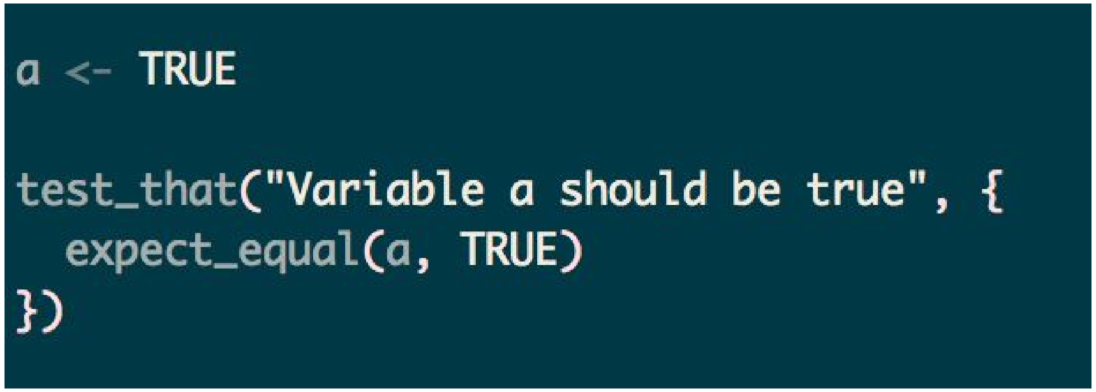


```r
a <- TRUE

test_that("Variable a should be true", {
  expect_equal(a, TRUE)
})
```

Tests should be described by sentences:

Use the word should. It’s a challenge!

Anatomy of a test

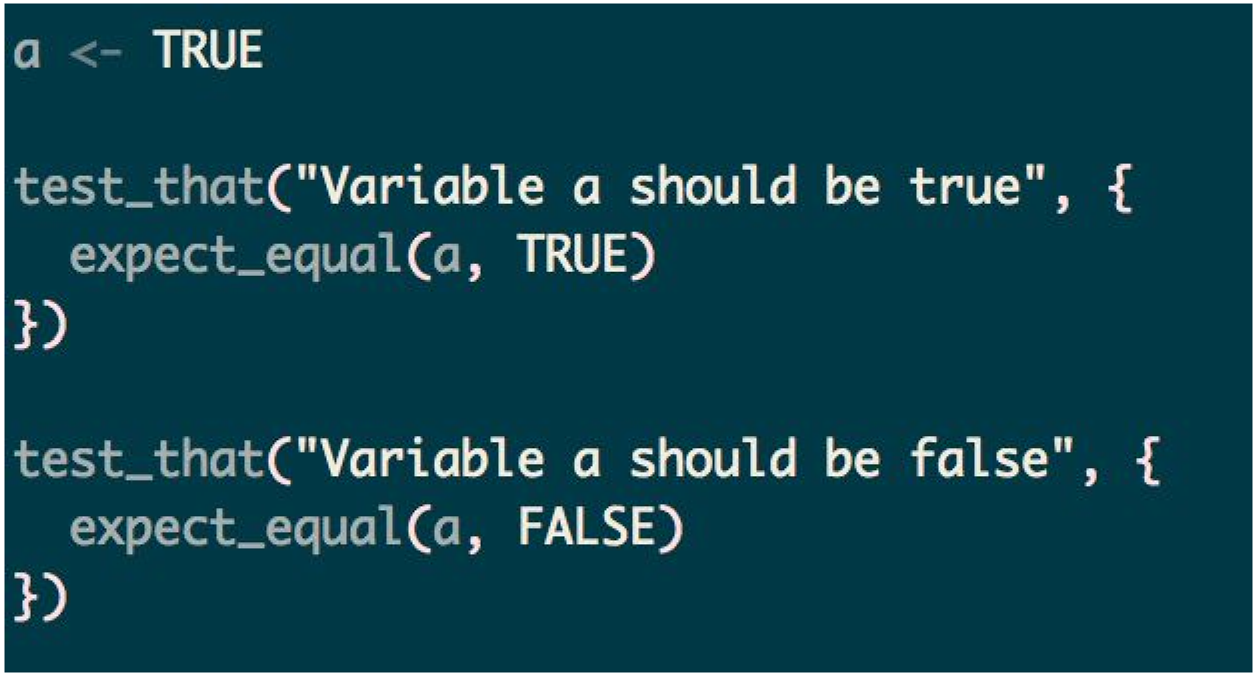


```r
a <- TRUE

test_that("Variable a should be true", {
  expect_equal(a, TRUE)
})
```

Tests contain an expectation:

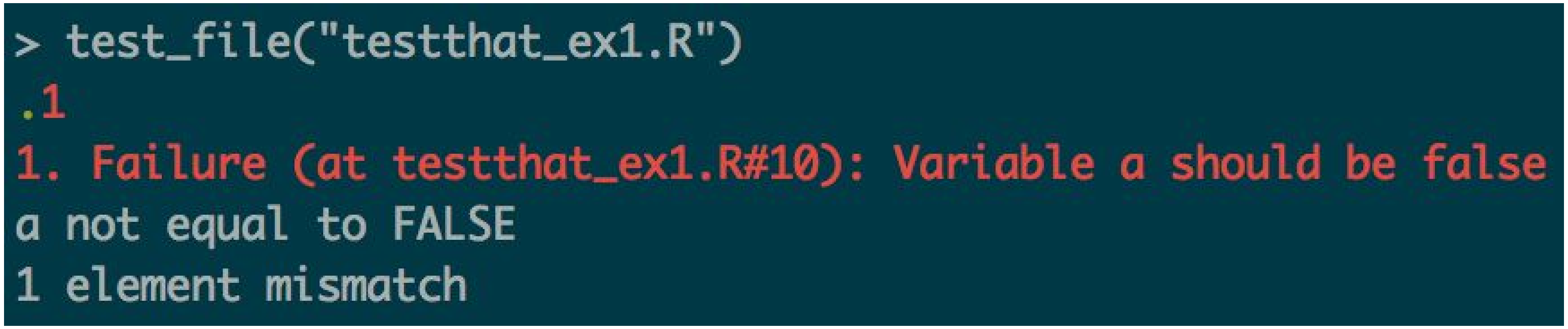
- Commonly test equality
- Can also test string matching, exceptions raised etc.

Test Feedback



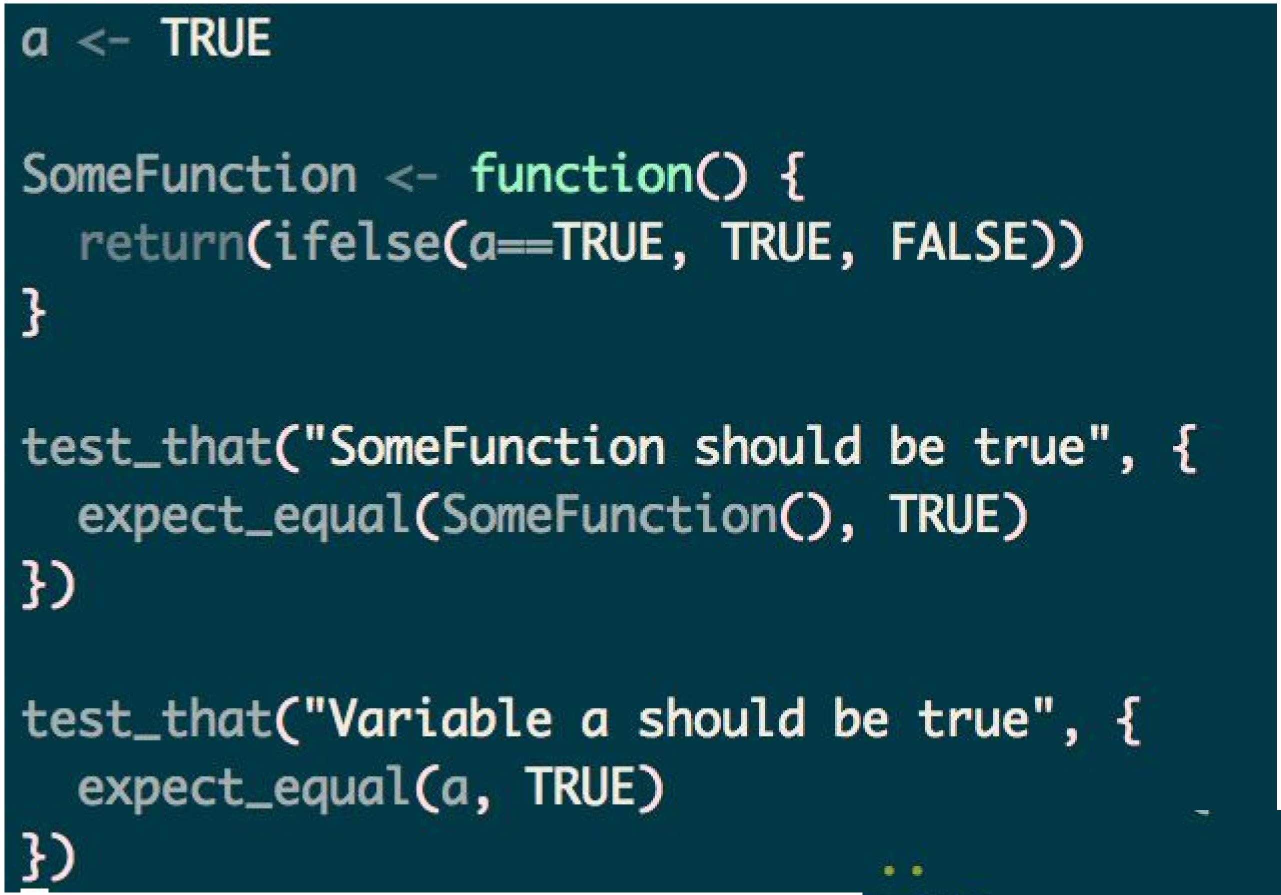
```
a <- TRUE

test_that("Variable a should be true", {
  expect_equal(a, TRUE)
})

test_that("Variable a should be false", {
  expect_equal(a, FALSE)
})
```

Test Feedback

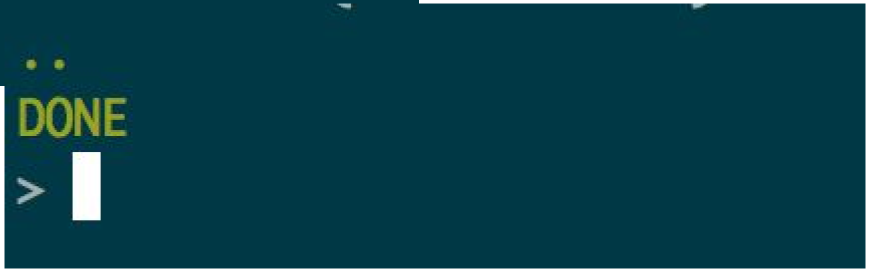

```
> test_file("testthat_ex1.R")
.1
1. Failure (at testthat_ex1.R#10): Variable a should be false
a not equal to FALSE
1 element mismatch
```

What I really need to know is does my code behave in the way I expect it to.

Define some behaviour

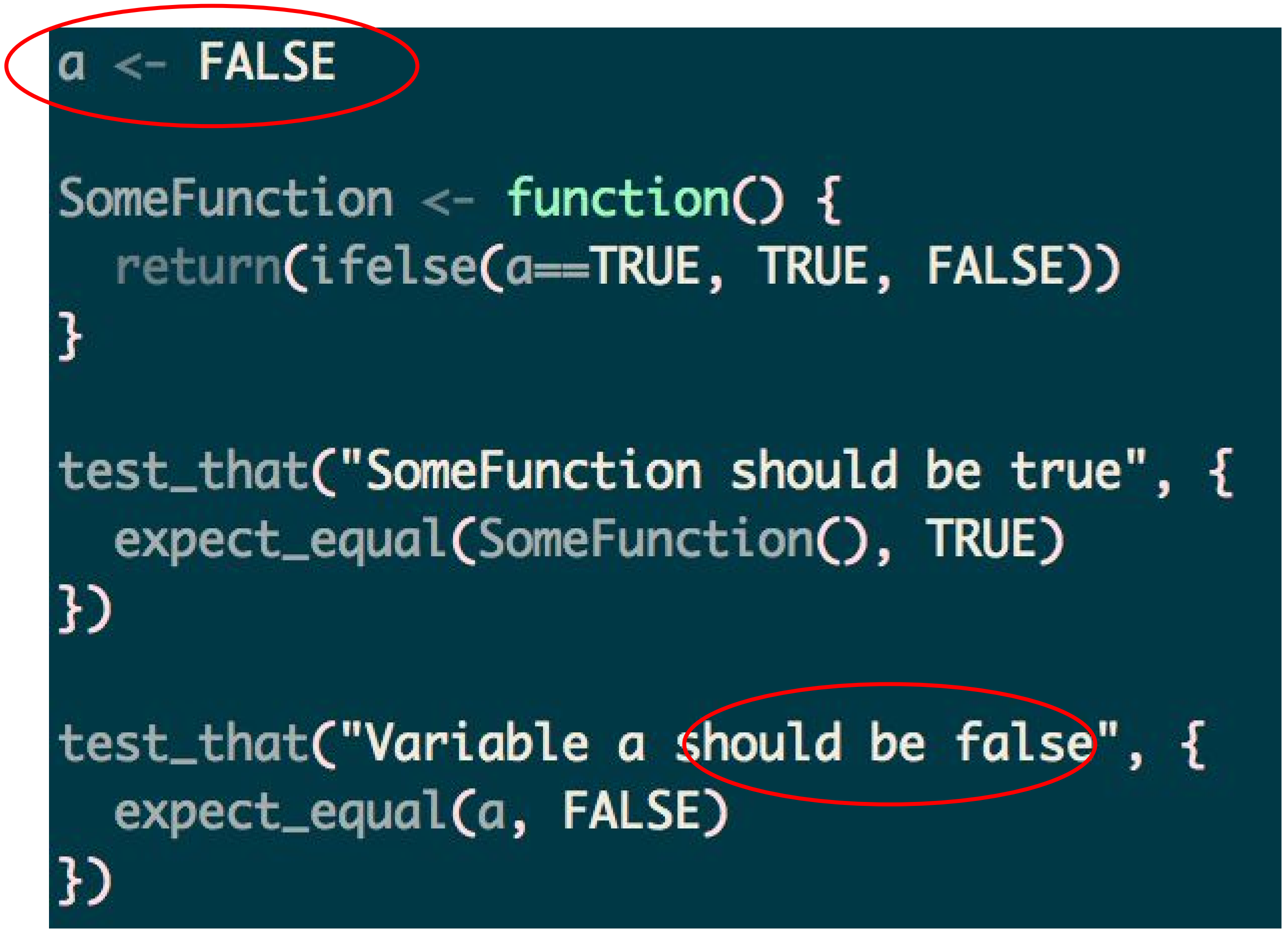


```r
a <- TRUE

SomeFunction <- function() {
  return(ifelse(a==TRUE, TRUE, FALSE))
}

test_that("SomeFunction should be true", {
  expect_equal(SomeFunction(), TRUE)
})

test_that("Variable a should be true", {
  expect_equal(a, TRUE)
})
```



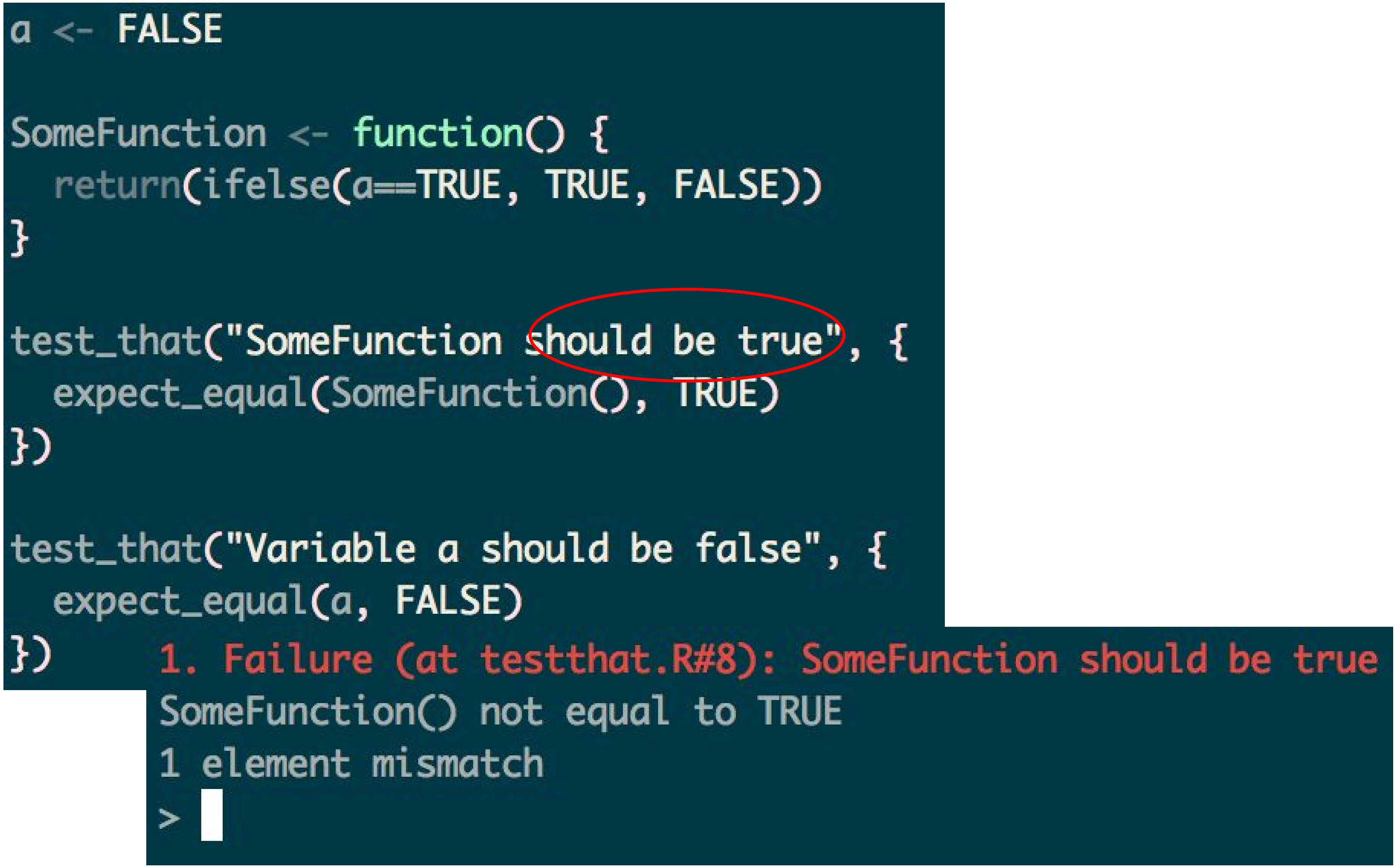
```
..
DONE
> 
```

Change the behaviour



```
a <- FALSE

SomeFunction <- function() {
  return(ifelse(a==TRUE, TRUE, FALSE))
}

test_that("SomeFunction should be true", {
  expect_equal(SomeFunction(), TRUE)
})

test_that("Variable a should be false", {
  expect_equal(a, FALSE)
})
```

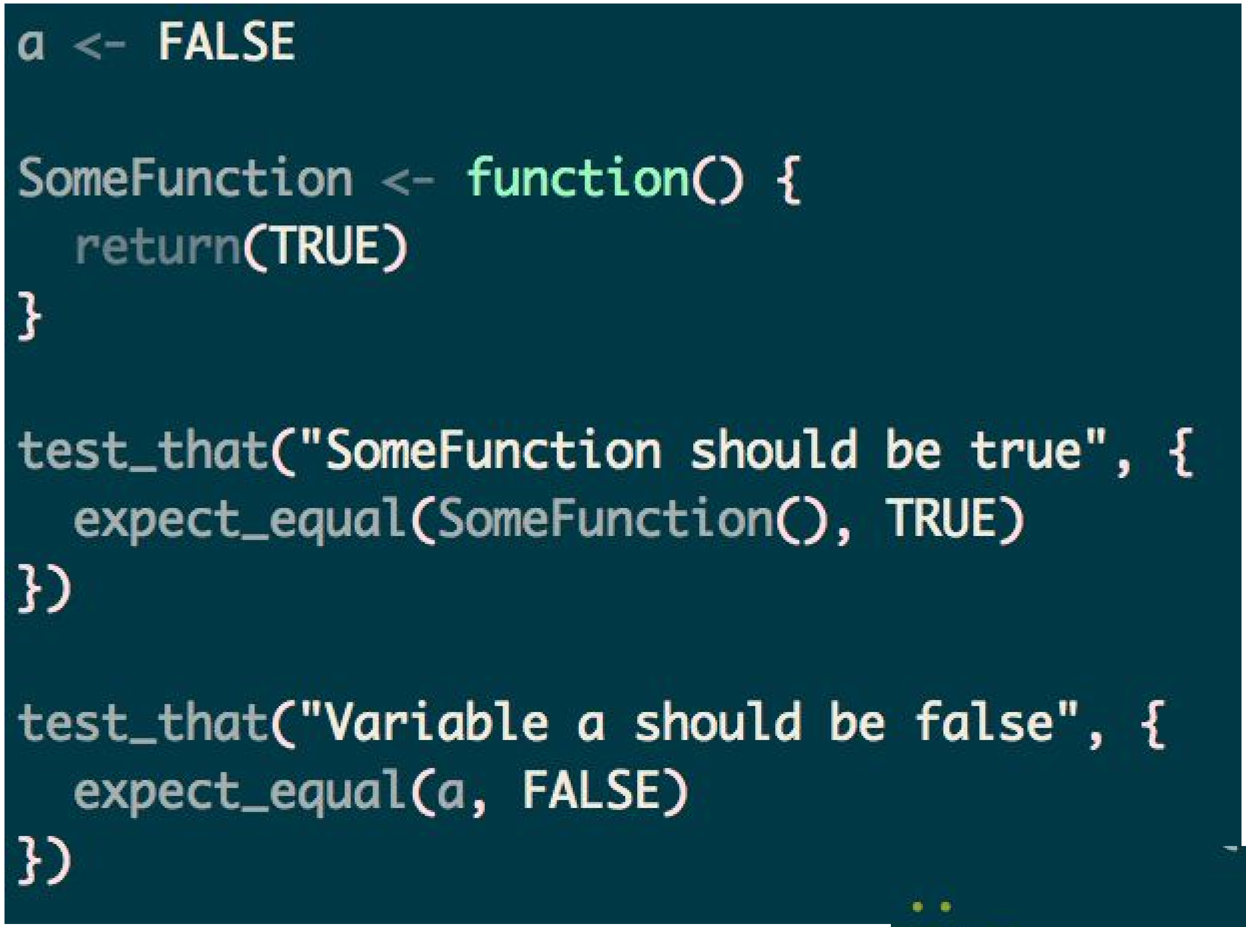
Change the behaviour



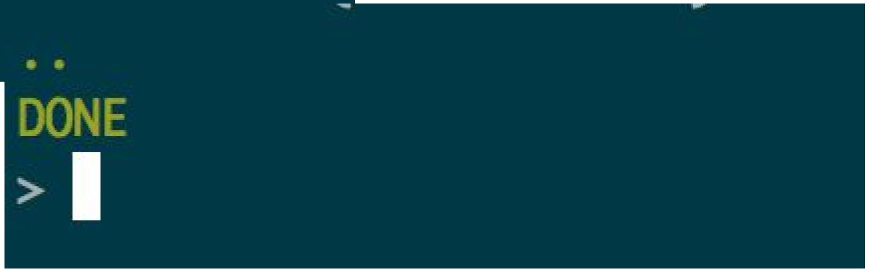
```
a <- FALSE

SomeFunction <- function() {
  return(ifelse(a==TRUE, TRUE, FALSE))
}

test_that("SomeFunction should be true", {
  expect_equal(SomeFunction(), TRUE)
})

test_that("Variable a should be false", {
  expect_equal(a, FALSE)
})
```

```
1. Failure (at testthat.R#8): SomeFunction should be true
SomeFunction() not equal to TRUE
1 element mismatch
>
```

Fix the bug



```r
a <- FALSE

SomeFunction <- function() {
  return(TRUE)
}

test_that("SomeFunction should be true", {
  expect_equal(SomeFunction(), TRUE)
})

test_that("Variable a should be false", {
  expect_equal(a, FALSE)
})
```



```
..
DONE
> 
```

When a test fails

The behaviour moved elsewhere

- Update the test

A (new?) bug was introduced

- Fix the bug

The behaviour is no longer expected

- Update or remove the test

No magic bullet

- Tests might have bugs
- You might have misunderstood the desired behaviour

Further Reading

Dan North - Introducing BDD

- http://dannorth.net/introducing-bdd/

Testthat

- https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf