Unit testing

Data Wrangling and Husbandry

04/13/2020

Unit testing

- If you write the same code repeatedly, you should write a function
  - for coding efficiency
  - for clarity
  - to reduce the introduction of bugs when modifying your code
- If you use the same function in different settings, it may be useful to write a package
- As that package evolves, it is important that your functions not break on your old code

Unit testing

- The concept: you write a script that automatically evaluates pieces of your code and checks it against expected behavior
- The `testthat` package is a convenient way to do so

A toy example

```
find_real_roots <- function(p, q){
  # Find roots of x^2 + px + q = 0
  (-p  + c(-1, 1) * sqrt(p^2 - 4 * q))/2
}



find_real_roots(-5, 1)
```

```
## [1] 0.2087122 4.7912878
```

```
find_real_roots(2, 1)
```

```
## [1] -1 -1
```

```r
find_real_roots(0, 1)
```

```
## Warning in sqrt(p^2 - 4 * q): NaNs produced
```

```
## [1] NaN NaN
```

```r
find_real_roots(0, NA)
```

```
## [1] NA NA
```

```r
find_real_roots("A", "B")
```

```
## Error in -p: invalid argument to unary operator
```

```r
find_real_roots(1:4, 0)
```

```
## [1] -1  0 -3  0
```

Improving the function

```
find_real_roots <- function(p, q){
  # Find roots of x^2 + px + q = 0
  if
    (!(class(p) %in% c("numeric", "integer")) |
    !(class(q) %in% c("numeric", "integer"))
  ) stop("Input must be two numbers")
  if (length(p) != 1 | length(q) != 1) stop ("p and q must
  (-p  + c(-1, 1) * sqrt(p^2 - 4 * q))/2
}

find_real_roots("A", "B")
```

```
## Error in find_real_roots("A", "B"): Input must be two nu
```

```
find_real_roots(1:4, 0)
```

```
## Error in find_real_roots(1:4, 0): p and q must each have
```

Back to unit testing

- Might want to test that
  - arguments that are not numbers lead to errors
  - if both arguments aren't of length 1 then there is an error
  - if an argument is NA, then the result is c(NA, NA)
  - the result is a numeric vector of length 2
  - maybe confirm a few values

Using the testthat package

- You write a series of expectations, as below:

```
library(testthat)
```

```
##
## Attaching package: 'testthat'

## The following object is masked from 'package:dplyr':
##
##     matches

## The following object is masked from 'package:purrr':
##
##     is_null

## The following object is masked from 'package:tidyr':
##
##     matches
```

```
test_that("find_real_roots returns a numeric vector of leng
  expect_is(find_real_roots(5, 2), "numeric")
```

If there is no problem then each test will return nothing. If there is a problem you'll get something like this

```
expect_equal(find_real_roots(2, 1), c(-1, 5))
```

```
## Error: find_real_roots(2, 1) not equal to c(-1, 5).
## 1/2 mismatches
## [2] -1 - 5 == -6
```

or

```
test_that("intentional failure", {
  expect_equal(find_real_roots(2, 1), c(-1, 5))
})
```

```
## Error: Test failed: 'intentional failure'
## * <text>:2: find_real_roots(2, 1) not equal to c(-1, 5).
## 1/2 mismatches
## [2] -1 - 5 == -6
```

Typically you would put all of these in a file, say `test_find_real_roots.R`.

The **very first** line of the file has to be

```
context("Some explanatory string")
```

where "Some explanatory string" is a description of what you are testing, like "String processing".

With the file created, you can run all of the tests with

```
test_file("test_find_real_roots.R")
```

```
## v |  OK F W S | Context
## / |   0       | Unit Test Examplesx |   7 1     | Unit T
## ----------------------------------------------------------
## test_find_real_roots.R:23: failure: intentional error
## find_real_roots(2, 1) not equal to c(-1, 5).
## 1/2 mismatches
## [2] -1 - 5 == -6
## ----------------------------------------------------------
##
## == Results ===============================================
## OK:       7
## Failed:   1
## Warnings: 0
## Skipped:  0
##
## Keep trying!
```

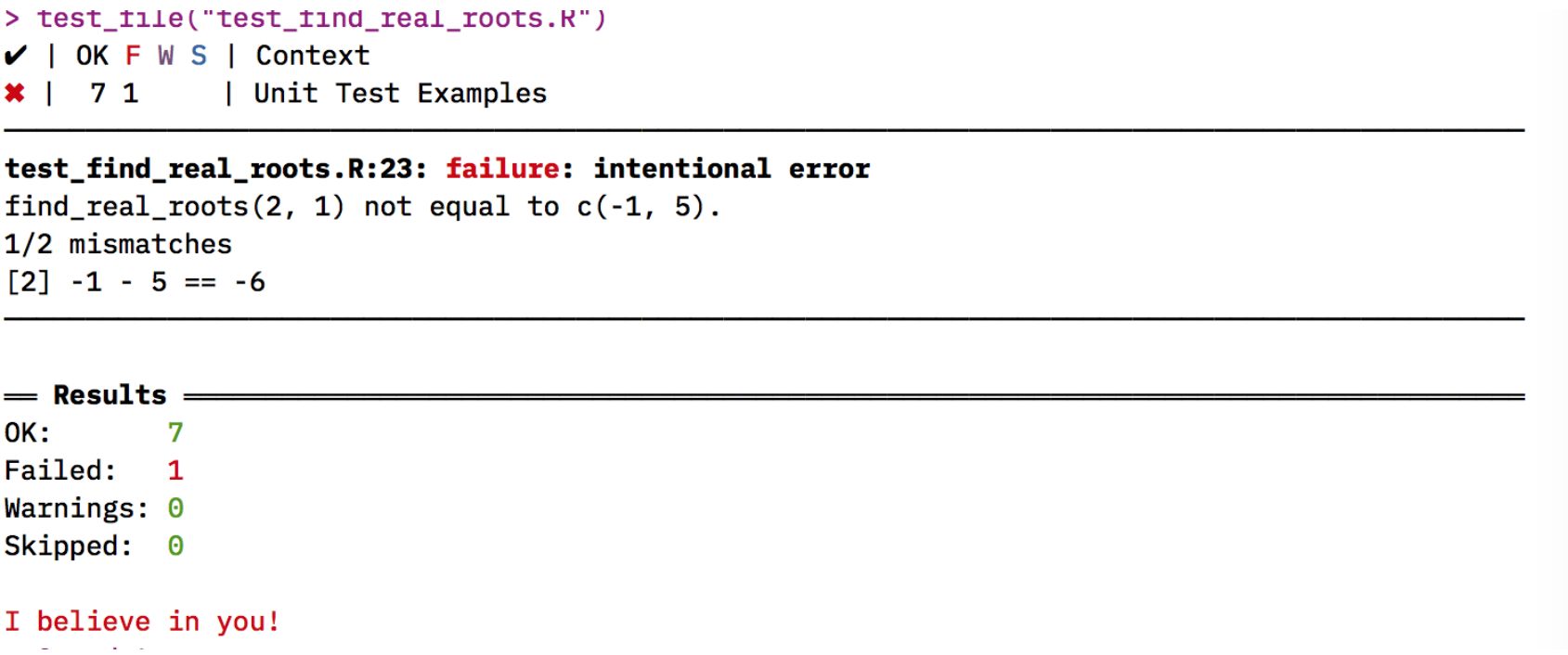

Figure 1: test_file output

To add this approach to a package,

- Type `devtools::use_testthat()` to add a `tests` directory to your package
- Create a test file as above:
  - in goes in the `tests/testthat/` directory
  - its name must start with `test`
- Use `devtools::test()` or, from the RStudio menu, Build > Test Package