

Debugging functions in Udemy Coding Lectures

There are three types of errors when we code in R Udemy System: Logic, Syntax and Structural errors. In this file we will learn how we can debug the three. **This document is valid for every coding exercise, except the Functions coding exercise where we will create our own functions.**

Our example exercise will be:


```
1 test_example <- function(){
2   # Create a vector
3   # with the elements 1,2,3,4
4   # and name it example_vector
5
6   example_vector <- 0
7   list(example_vector)
8 }
```

Logic Errors:

- Logic errors are examples where the solution developed doesn't match what is asked on the question. This happens for a multitude of reasons and is completely part of the learning process.
- As an example, the exercise asks us for a vector with the elements 1,2,3,4 – imagine we would provide this solution:

```
1 test_example <- function(){
2   # Create a vector
3   # with the elements 1,2,3,4
4   # and name it example_vector
5
6   example_vector <- c(1,2,3,5)
7   list(example_vector)
8 }
```

- In this case, there was clearly a problem with the definition of the last element **that is 5 and not 4** – Udemy platform normally gives us a good error message in this case – after you hit “Check Solution” this is the error message:

 **Oops, your solution is incorrect.**

```
all.equal(result[[1]], c(1, 2, 3, 4)) not equal  
to TRUE. (@test-exercise.R#13)
```

```
Types not compatible: character is not logical
```

The solution states that our **result[[1]]** – meaning the first exercise (the 1 states the exercise where we have the error) is not equal to what is expected. Let’s extend our exercise to two questions:

```
1 test_example <- function(){  
2   # Create a vector  
3   # with the elements 1,2,3,4  
4   # and name it example_vector  
5  
6   example_vector <- 0  
7  
8   # Create a vector where you multiply  
9   # the first vector by 2 - store it  
10  # in a example_vector_2 object  
11  example_vector_2 <- 0  
12  
13  list(example_vector)  
14 }
```

- Now let’s correct our first exercise and make a Logic mistake on exercise 2:

```
1 test_example <- function(){  
2   # Create a vector  
3   # with the elements 1,2,3,4  
4   # and name it example_vector  
5  
6   example_vector <- c(1,2,3,4)  
7  
8   # Create a vector where you multiply  
9   # the first vector by 2 - store it  
10  # in a example_vector_2 object  
11  example_vector_2 <- example_vector/2  
12  
13  list(example_vector)  
14 }
```

- As you can see, on exercise 2 we are dividing the vector by 2, instead of multiplying by 2. But now, our first exercise is correct. Let's see the output:

⚠️ Oops, your solution is incorrect.

```
subscript out of bounds (@test-exercise.R#16)

Backtrace:
 1. testthat::expect_equal(...) test-exercise.R:
16:4
 4. base::all.equal(result[[2]], c(2, 4, 6, 8))
```

- Note that our error now refers **result[[2]]** – which means that our exercise 2 had some kind of trouble.
- Bottom line, with logic errors, you can guide yourself through Udemy feedback system and use the part between **[[]]** to understand which exercise you are failing – **the first exercise that you fail will be the one in the output.**
- The second part of the `all.equal` helps to guide you on the right solution – notice that this test expects that the resulting vector on exercise 2 is `c(2,4,6,8)`, hence, our vector multiplied by 2. You can guide yourself using the second part of the `all.equal` function where you will have what is expected from our exercise output.

Syntax Errors:

- Syntax errors are trickier. Udemy's feedback is not perfect in providing good feedback when you make a syntax mistake – particularly in the color syntax – sometimes if you go around the function and make a syntax mistake adding something accidentally you will have a multitude of errors in the platform, as an example – if we add an extra parenthesis on our vector function:

```
test_example <- function(){
  # Create a vector
  # with the elements 1,2,3,4
  # and name it example_vector

  example_vector <- c(1,2,3,4))

  # Create a vector where you multiply
  # the first vector by 2 - store it
  # in a example_vector_2 object
  example_vector_2 <- example_vector/2

  list(example_vector)
```

- Udey will be able to point you to the problem – but the output is confusing:

```
> test_example <- function(){
+   # Create a vector
+   # with the elements 1,2,3,4
+   # and name it example_vector
+
+   example_vector <- c(1,2,3,4))
Error: unexpected ')' in:
"
      example_vector <- c(1,2,3,4))"
Execution halted
```

- If you find it difficult to deal with the Udey feedback system, just copy your code and paste it in R Studio – and execute it:

```
1 test_example <- function(){
2   # Create a vector
3   # with the elements 1,2,3,4
4   # and name it example_vector
5
6   example_vector <- c(1,2,3,4))
7
8   # Create a vector where you multiply
9   # the first vector by 2 - store it
10  # in a example_vector_2 object
11  example_vector_2 <- example_vector/2
12
13  list(example_vector, example_vector_2)
14  }}
```

- The output on the console:

```
      example_vector <- c(1,2,3,4))"
>
> # Create a vector where you multiply
> # the first vector by 2 - store it
> # in a example_vector_2 object
> example_vector_2 <- example_vector/2
Error: object 'example_vector' not found
>
> list(example_vector)
Error: object 'example_vector' not found
> }}
Error: unexpected '}' in "}"
> |
```

- The red colored lines help a lot in guiding you to the error and debugging the function

Structural Errors:

- These are the crazier types of errors and are not related to R coding in general but the way that exercises were structured. As an example, my exercises are based on a function that outputs a list. The function definition:

```
test_example <- function(){  
  # Create a vector
```

- And the output and closing of the function:

```
12  
13     list(example_vector)  
14 }
```

- **It's important that your code is enclosed between these two blocks of code— this is the function that produces the outputs that will be fed to the test so don't change these lines.**
- Let's see an example of what happens when we delete the function definition (don't worry, this will be much clearer and will make a lot of sense when we get to the Functions section 😊) – this is the code we submit on Udemy, without the function definition:

```
1 |  
2   # Create a vector  
3   # with the elements 1,2,3,4  
4   # and name it example_vector  
5  
6   example_vector <- c(1,2,3,4)  
7  
8   # Create a vector where you multiply  
9   # the first vector by 2 - store it  
10  # in a example_vector_2 object  
11  example_vector_2 <- example_vector/2  
12  
13  list(example_vector)
```

Oops, your solution is incorrect.

```
Error in UseMethod("xml_add_child") :  
  no applicable method for 'xml_add_child' appli  
ed to an object of class "NULL"  
Calls: <Anonymous> ... o_apply -> lapply -> FUN  
-> <Anonymous> -> <Anonymous>  
Execution halted
```

- To avoid these types of errors don't delete the last line of the function or the first function definition. Only change the lines of code that have variables with 0's assigned to them.

Again, thank you for enrolling in my course and if you have additional trouble, message me on LinkedIn – Congratulations for having a go at the coding exercises – this is a super important part of the process!