Lecture 3.6 – Debugging

Specific Learning Objectives:

1.2.1 – Understand the way computers execute commands.

1.2.11 – Use a conditional statement to add exception handling to a function or script.

1.3.9 – Learn basic skills in debugging and troubleshooting error messages.

1.3.10 – Search for effective solutions and tools using online resources.

3.5 – Think and work independently with code.

Learning How to Debug

- **Debugging** is the process of removing errors in code that cause it to malfunction.
    - This can include removing error messages or unintended/unexpected behavior or results.
    - Problem solving these issues is a skill that requires practice and a solid understanding of what is happening within your script and function.

- **Four General Strategies of Debugging:**

I. Check for the most common (i.e. “screaming toddler”) problems.

II. Figure out where the error is.

III. Google the error.

IV. Create a hypothesis and test it.

I. How R is like a toddler

https://reasonsmysoniscrying.tumblr.com/

I ate a cracker she shoved into my mouth.

He shoved his train off the track.

I ripped her pancakes up into smaller pieces like she asked me to.

R will give you error messages that don't make sense and don't seem connected to the problem. It's like a toddler: it can tell you things, but those things aren't always useful to solving the problem at hand.

I. How R is like a toddler

https://reasonsmysoniscrying.tumblr.com/

I ate a cracker she shoved into my mouth.

He shoved his train off the track.

I ripped her pancakes up into smaller pieces like she asked me to.

Why is the toddler really screaming?

1. **They are hungry.**
2. **They are tired.**
3. **They have to potty.**

Why is R not working?

1. **It can’t find the object/function.**
2. **It’s using an unexpected value.**
3. **It wants a different data class.**

I. R's Screaming Toddler Problems

1. It can't find the object/function.

- Errors look like:

```
Error: object 'a' not found
```

```
Error in C("sum", "mean", "average") :
  object not interpretable as a factor
```

```
Error in file(file, "rt") : cannot open the connection
In addition: Warning message:
In file(file, "rt") :
  cannot open file 'data.csv': No such file or directory
```

```
Error in ggplot() : could not find function "ggplot"
```

- Try these things to fix it:
  - Check spelling
  - Check capitalization/case
  - Check working directory/location of file
  - Check to make sure object is in your "local" environment (is the package loaded? is your function including the right inputs?)

I. R's Screaming Toddler Problems

2. It's using an unexpected value.

- Errors look like:

```
Error in a[1, ] : object of type 'closure' is not subsettable
```

```
Error in b[, 1] : incorrect number of dimensions
```

```
> mean(1,32,25,2)
[1] 1
```

```
Error in sum(1, 2, 4, ) : argument 4 is empty
```

- Try these things to fix it:
  - Is R calling the object you think it is?
  - Clear your environment and try again.
  - Look closely at the arguments in a function, are you sure they are in the right order/the correct number?
  - Is the object the same size as what you're calling?

I. R's Screaming Toddler Problems

3. It wants a different data class.

- Errors look like:

```
Error in a[1, ] : object of type 'closure' is not subsettable
```

```
Error in b[, 1] : incorrect number of dimensions
```

```
Error in sum("a") : invalid 'type' (character) of argument
```

```
Error in C("sum", "mean", "average") :
  object not interpretable as a factor
```

- Try these things to fix it:
  - Is what you are doing to the object appropriate? (specifically indexing)
  - Is that the data type that a function wants as an input?
  - Look closely at the arguments in a function, are you sure they are in the right order/the correct number?

II. Figure Out Where the Error Is

- If it's not one of the most common problems, next step is to figure out where the error is and get more information about it.

  - **READ THE ERROR MESSAGE.** Even if it's not helpful, it will usually help you figure out what line the problem is at.

  - Use **`traceback()`**, this can help by identifying the lines in each environment that are throwing errors.

  - Use the script editor to help point out any syntax errors.

  - Trace back the error yourself by running a script line-by-line until you hit the error.

III. Google it

- The most helpful thing to do sometimes is to google an error message to see if someone else has already fixed it!

  - To maximize search results for success, don't include too much specific information about the error such as object names.

For example, you get this error:

```
Error in a[1, ] : object of type 'closure' is not subsettable
```

Don't google this object name, not many people will name objects "a"

Try googling the general statement

IV. Create a Hypothesis and Test It

- After reading potential answers, form a hypothesis and figure out how to test it.

  - This will depend on the particular problem, but will almost always involve simply trying different things until both of these things happen:

    1. **The error goes away**
    2. **You understand the error**

```
Error in a[1, ] : object of type 'closure' is not subsettable
```

Google says don’t use square brackets, why?

Does taking away the square brackets fix the problem?

Debugging Functions

- Remember that functions create their own temporary environments, so it's not easy to know what's going on inside of them.

  - Running each line of a function will produce an error in a regular console (the global environment) because the arguments aren't defined there!

  - We'll practice using this function:

```
myfunction1 <- function(x,a,b,c){
  t <- x+a
  t2 <- t/b
  t3 <- t2 %% c
  return(t3)
}

myfunction1(11,4,2,4)
```

- Here are two strategies for dealing with errors in functions:

  1. Try defining the arguments in the global environment and running each line of the function.

  2. Try using the debugger in RStudio.

Using the Global Environment to Debug Functions

- Define each argument as a variable in the global environment.

- Run each line of the function.

```r
myfunction1 <- function(x,a,b,c){
  t <- x+a
  t2 <- t/b
  t3 <- t2 %% c
  return(t3)
}

myfunction1(11,4,2,4)
```

```r
# Assign the values used in the internal workings of the function:
x <-11
a <- 4
b <- 2
c <- 4

# Now run each line of the function in the global environment:
#myfunction1 <- function(x,a,b,c){
  t <- x+a
  t2 <- t/b
  t3 <- t2 %% c
  #return(t3)
#}
```

Don't run these lines!

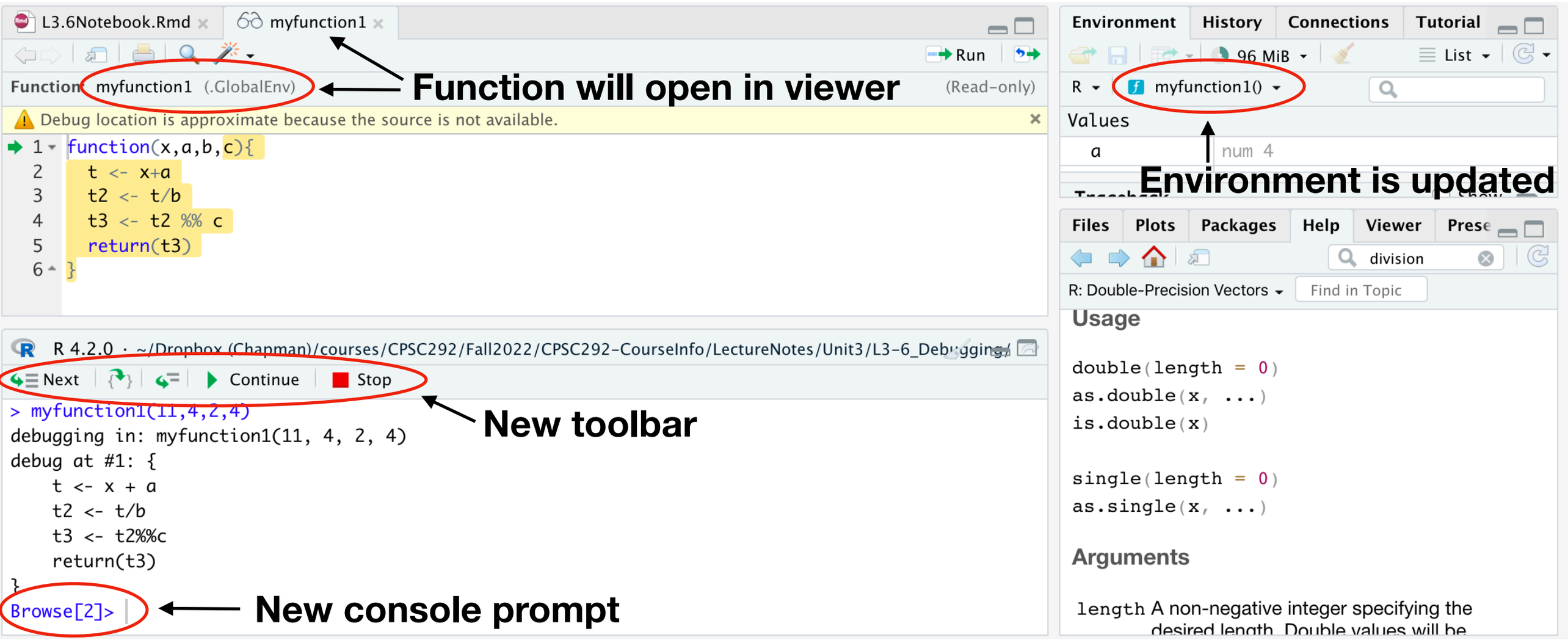
- Here you'll be able to see which line throws the error, and what is happening in the internal environment.

Using the RStudio Debugging Tool

- RStudio's debugging tool provides a nice platform for finding and correcting problems in complex code with nested environments.

  - Running `browser()` in the console will enter debugging mode. `Q` will quit.

  - Adding `debug()` with your function as the argument will open the debug mode when your function is called. To end, run `undebug()` on your function.



Adding Custom Error Messages

- Sometimes it's easy to anticipate errors you (or someone else) might make with a function, so you can build in "early stops" to make sure the user is providing appropriate arguments AND to make specific error messages to help the user.

  - In the case of `myfunction1`, the internal operations only accept numeric data, and passing a character will result in the error:

```
> myfunction1("11",4,2,4)
Error in x + a : non-numeric argument to binary operator
```

  - Adding a `stop()` for values of `x` that are not numeric gives the user more info about the error:

```
myfunction1 <- function(x,a,b,c){
  if(!is.numeric(x)) stop("Please only give numbers!")
  t <- x+a
  t2 <- t/b
  t3 <- t2 %% c
  return(t3)
}
```

```
> myfunction1("11",4,2,4)
Error in myfunction1("11", 4, 2, 4) : Please only give numbers!
```

Adding Custom Warning Messages

- If you'd like to give a warning (which will allow the code to keep running) as opposed to an error (which will halt execution of the code), use `warning()`.

```
243 myfunction1 <- function(x,a,b,c){
244   if(x <10) warning("You should use a number bigger than 10 for x!\n")
245   t <- x+a
246   t2 <- t/b
247   t3 <- t2 %% c
248   return(t3)
249 }
```

(The `\n` will be evaluated as a new line when printed.)

```
> myfunction1(9,4,2,4)
Warning: You should use a number bigger than 10 for x!
[1] 2.5
```

Check Your Understanding

See the Lecture3.6.Rmd notebook for a sample function to debug.

Debug the function. What conditionals can you add to prevent this problem in the future?

Action Items

1. **Complete previous assignments.**

2. **Read Davies Ch. 11 for next time.**