How to Learn a Scripting Language

Denis Valle

May 1, 2017

How to learn a scripting language

Instructors of courses that require some level of programming often take for granted that people study and learn how to code in the same way that they do. At least this was my case. However, it has become clear to me that effectively studying and learning code actually requires a very different approach than studying other subjects. For this reason, I have decided to make a list of approaches/ideas that I think are useful when learning a scripting language like R and that might not be that obvious for novices:

1) Avoid the "copy and paste" approach to writing code

Although copying and pasting may help you to avoid typing errors, it can also interfere with your learning process for two reasons:

a) Typing errors can help you gain experience in writing code as R provides informative feedback when you make such mistakes. Making and correcting typing errors is an important skill to develop, particularly when you are typing a lot of code for your own data analysis.

b) Copying and pasting code may give you the impression that you know what you are doing when, in reality, you probably do not fully understand what the individual lines of code are actually doing. Furthermore, this problem will just get worse as you deal with increasingly longer and more complicated scripts

2) Study code line-by-line

This means running one line of code at a time and making sure that you understand why the output is what it is. If things are not clear, it is important to spend more time with that piece of the code. Here are some tricks that are often helpful to understand a particular piece of code:

a) Break a line of code into its components and try to understand the individual pieces. For example, say you are trying to understand the last line in the code below:

```
tst  <- read.csv('final_edited.csv', as.is = TRUE)

tst2 <- tst[, grep('w', colnames(tst))]
```

The last line of code is comprised of three functions nested within each other, whose results are then used to subset specific columns. To better understand what is going on, you can break the last line of code into multiple pieces to see how they operate individually.

```
tst  <- read.csv('final_edited.csv', as.is = TRUE)

col_names <- colnames(tst)

col_nums  <- grep('w', col_names)

tst2 <- tst[, col_nums]
```

Finally, it is SUPER helpful to annotate each line as you find out what these pieces do. Clear documentation is critical as you may not remember what you did when you come back to a piece of code at some point in the future. As a wise programmer once said, "Write code for the future you." Documentation is also useful when you want to adapt or reuse code in some other way. In situations like this, you will immediately know what a specific chunk of code does because it has been clearly documented.

b) Perform mini-experiments: create a simpler example in which you can tinker with the code and see what happens to the output. For instance, if it is not clear what the function "grep" does in the example above, you could try modifying it in multiple ways to gain a better understanding of how it works:

```
vctr <- c('denis','matt','john','sandra') # Create a vector of four names.

idx <- grep('a', vctr) # Find the items in the vector with "a" in them.

idx       # Did we find anything?
```

```
## [1] 2 4
```

```
vctr[idx] # Display what was found.
```

```
## [1] "matt"   "sandra"
```

What happens if I substitute "a" for "e" in my call to the grep function?

```
idx <- grep('e', vctr) # Find the items in the vector with an "e" in them.

idx       # Did we find anything?
```

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

```
vctr[idx] # Display what was found.
```

```
## [1] "denis"
```

In a way, this is not that different from experiments we perform in science, in which we are trying to understand how things work!

3) Use the internet to find answers

Everybody (from novice to more experienced users) relies on the internet when they don't understand something. It is likely that other people have already asked (and received useful answers) for the problem that you are facing. However, finding the exact piece of information that you need might be hard, especially if you don't use the correct terms/key words. Learning how to search for the information that you need is a skill that also takes practice. "Stackoverflow" and existing cheatsheets (e.g., https://www.rstudio.com/resources/cheatsheets/) can be very helpful.

4) Be clear when you post a question online

Knowing how to post your question is very important:

a) Make the problem as simple as possible to explain and, if applicable, to reproduce. Few people are going to be willing to help if first they have to understand the 100+ lines of code that you have given them.

b) If the problem is an error message that you don't understand, providing code that is able to reproduce the error is often critical for others to be able to help you.

c) If the problem is a particular task that you are trying to perform, it is very useful to have simple examples of the type of input that you have and the type of output that you want.

5) Take your time

It is important to realize that it takes time to learn how to code. What this implies is that you should not rush to get things done if you want to master this skill. In particular, everybody goes through some level of struggle and frustration when learning how to code. However, once you have mastered it, you will be amazed by what this skill can do for you.