Tips for Writing Good Code

CSC121
Mark Kazakevich
Let’s talk about how to write better code

- We’ve learned a lot about how to write code in R
- As you work through the course material and write your own code, you may develop bad habits and misconceptions.
- That’s ok!
  - …as long as you fix them :)
- Let’s go through a few issues I’ve been seeing as I look at your code
Function names in function bodies are bad

- Function names should not appear in function bodies as a variable.

- I see a lot of people returning the name of the function they are using:

  ```r
  FunctionName <- function(argument) {
    .
    .
    .
    return(FunctionName)
  }
  ```

  **Rule:** The name of the function should **never** appear as a variable in the function body. Not in the return statement, not as an intermediate variable, not anywhere in the function body.
Calling functions in other functions

- Functions can be used in other functions only by *calling* them

Function1 <- function(argument) {
  value <- 1
  return(value)
}

Function2 <- function(argument) {
  value <- Function1 * 7
  return(value)
}

**Rule:** You cannot use the name of a function in another function unless you are calling that function. How do you call the function?...
Calling functions in other functions

- Functions can be used in other functions only by *calling* them

Function1 <- function(argument) {
  .
  value <- 1
  return(value)
}

Function2 <- function(argument) {
  .
  value <- Function1(4) * 7
  return(value)
}

Correct.
We called the function with an argument: 4
Variable names in one function don’t have any connection to those of another function.

```r
Function1 <- function(argument) {
  .
  value <- 1
  return(value)
}

Function2 <- function(argument) {
  .
  .
  value <- Function1(4) * 7
  return value
}
```

These two variables are both called `value`. BUT...they are in different functions. So they will never know about each other.

Variables inside functions only live in their own environment.

Let’s see this in RStudio.
Functions and if statements must be properly indented for good style and readability.

```r
FunctionName <- function(argument) {
  # Returns...
  if (argument > 5) {
    argumentToSquare <- argument + 7
    squaredArgument <- argumentToSquare^2
  }
}
```

If-statement indented

Function Indented
Indenting

- Make sure any open curly brackets are closed and indented properly
- Line up the closing brackets with the function name or if statement

```r
FunctionName <- function(argument) {
  # Returns...
  if (argument > 5) {
    argumentToSquare <- argument + 7
    squaredArgument <- argumentToSquare^2
  }

  # If-statement bracket closed
  .
  .
}

# Function bracket closed
Same for else ifs and else, and nested ifs

```java
if (condition1) {
    ..statements1..
} else if (condition2) {
    if (condition3) {
        ...
    }
} else {
    ..statementsN
}
```

Nested if-statement is indented, and the closing bracket is also indented
Redundant intermediate variables

- Do not assign variables another variable whose value you haven’t changed.

```r
FunctionName <- function(argument) {
  # Returns...
  value <- argument + 3
  valueToReturn <- value
  return(valueToReturn)
}
```

Wrong!
Redundant variable. There is no need to re-assign value to another variable if you haven’t changed it.
Redundant intermediate variables

- Do not assign variables another variable whose value you haven’t changed.

```r
FunctionName <- function(argument) {
  # Returns...

  value <- argument + 3

  return(value)
}
```

Just use `value`. 
Helping yourself write correct code

● Let’s say you wanted to write a function.

● Often when you are learning how to code, you want to just start typing immediately, and you get lost as to what you need to do

● This often leads to:
  ○ Hours of trying to figure out what you have to do to write a function
  ○ Functions that makes little sense when you read them
  ○ A function that works, but you don’t understand why it works
Helping yourself write correct code

- What are some good ways to start writing your function?

- First, use `# comments` to explain to yourself in English what your code has to do
  - This is called writing Pseudocode

- By writing out in English what your code has to do, you make it easier to understand what code you have to write.

- If you don’t understand your pseudocode, don’t start coding!
  - This usually means you need to think more about how you would write the function.
VectorAdding <- function(v) {
    # Returns a vector of all elements in v which are
    # less than 4, with all elements increased by 2
}

VectorAdding <- function(v) {
    # Returns a vector of all elements in v which are
    # less than 4, with all elements increased by 2

    # Need to get a vector of all elements less than 4

    # Need to increase all elements by 2
}

Add pseudocode
comments to explain what
your function should do
Pseudocode Example

VectorAdding <- function(v) {
  # Returns a vector of all elements in v which are
  # less than 4, with all elements increased by 2

  # Need to get a vector of all elements less than 4
  resultVector <- v[v < 4]

  # Need to increase all elements by 2
  resultVector <- resultVector + 2

  return(resultVector)
}

Write the code after you understand what you have to do
Print Statements

- Just like we used `print` statements to see the result of our function in the console, we can use them to check the value of intermediate variables as we run our functions.

- Use `print(variableName)` to see the value of the variable in the console.

- Use them to make sure your code is doing the right thing.

- Don’t forget to delete your print statements before you submit your code. You will lose marks otherwise!
VectorAdding <- function(v) {

  # Need to get a vector of all elements less than 4
  resultVector <- v[v < 4]
  print(resultVector)

  # Need to increase all elements by 2
  resultVector <- resultVector + 2
  print(resultVector)

  return(resultVector)
}

Check in the console to make sure you got back the values you wanted
Examples in RStudio