# Lecture 3.7 – Refactoring

### **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.

# What is 'Refactoring'?

- **Refactoring** is the process by which code is *restructured* without changing what the code *produces*.
  - Improves readability and reduces complexity
  - Cleaner code that is easier to understand
  - Can help fix bugs and other problems
  - Often makes code more compact and versatile

- Refactoring code is like editing a paper. You're not really changing what the paper says, but making in clearer and easier to understand!

### What are the goals of refactoring?

## Good goals

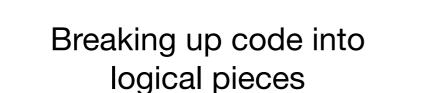


### **NOT** good goals





Improve readability & understandability





Improving naming &

Improving documentation

Bringing code up to best practices standards

"Rewriting" code



Code golfing



Optimization (different step!)



Improving looks (interface)



duplication



location of code



### When should you refactor code?

- After your code works and does the thing you want it to do.
- "Rule of 3" or "Three strikes and refactor" third time you have to use/copy the code, you refactor.
- Around the time you optimize the code.
- Around the time you have a code review (i.e. turn in a project).
- Before you give it to others who will use it (or turn it in!)
- Before doing production runs of data analysis.
- Before publishing/leaving a lab/putting it down for several months.

### Code Smells - Areas where refactoring can help





Nicer code



Too long or too short variable names

Rename during refactor

Commenting/uncommenting to alter behavior

Use conditionals for flow control

Using attach()

Try with() instead

Using setwd()

Use R Projects, standard organization

Excessive use of if and else

Try switch or early exits

Lots and lots of indentation

### **R Style Guide**

### http://adv-r.had.co.nz/Style.html

#### - File names

- be long enough to be informative
- end with .R or .Rmd
- if need to be run in a sequence, prefixed with numbers

```
# Good
fit-models.R
utility-functions.R

# Bad
foo.r
stuff.r
```

```
0-download.R
1-parse.R
2-explore.R
```

#### Object names

- lower case, words separated by underscore \_
- variables are nouns, functions are verbs
- concise and meaningful
- avoid using previously defined functions and special values

```
# Good
day_one
day_1

# Bad
first_day_of_the_month
DayOne
dayone
djm1

# Bad
T <- FALSE
c <- 10
mean <- function(x) sum(x)</pre>
```

### R Style Guide

#### http://adv-r.had.co.nz/Style.html

#### - Spacing

- spaces should be used around operators (<-, +, -, /, \*, =, etc)</li>
- put a space after commas
- do not put spaces around : or ::

```
# Good
average <- mean(feet / 12 + inches, na.rm = TRUE)
# Bad
average<-mean(feet/12+inches, na.rm=TRUE)</pre>
```

```
# Good
x <- 1:10
base::get

# Bad
x <- 1 : 10
base :: get</pre>
```

#### Indentation

- use 80 characters per line or fewer
- use 2 spaces to indent except for using a new line within a function, then indent to opening of function

### **R Style Guide**

### http://adv-r.had.co.nz/Style.html

#### Curly Braces

- opening curly braces should never go on their own line
- closing braces should go on their own line unless followed by else or if else
- always indent code inside braces
- ok to leave short things on one line

```
# Good

if (y < 0 && debug) {
    message("Y is negative")
}

if (y == 0) {
    log(x)
} else {
    y ^ x
}</pre>
```

```
# Bad

if (y < 0 && debug)
message("Y is negative")

if (y == 0) {
  log(x)
}
else {
  y ^ x
}</pre>
```

#### - Assignments

Use <- and not = for assignments</li>

### **Examples of Refactoring**

#### **Example 1: variable names**

Goals of Refactoring:

- 1. Making variable names more meaningful.
- 2. Conform variable names to style guide.

#### **Example 2: improve readability**

**Goals of Refactoring:** 

- Improve syntax by adding spaces.
- 2. Conform braces and brackets to style guide.

#### **Example 3: commenting/uncommenting to alter behavior**

Goals of Refactoring:

1. Add conditional to handle behavior switch

#### **Example 4: simplifying code**

Goals of Refactoring:

1. Simplify code for easier understanding

### **Check Your Understanding**

Refactor the code in the lecture notebook so that it conforms to the style guide.

## **Action Items**

1. Complete Assignment 3.6.