

Clean Code Examples

Anthea Alberto

2023-02-17

Principles of clean code

Code for people, not machines

“Always assume that others will read your source code.” (Mayer, 2022, p.56)

```
# Bad!
xxx <- 10000
yyy <- 0.1
zzz <- 1:10

for (iii in zzz) {
  print(xxx*(1+yyy)^iii)
}
```

```
## [1] 11000
## [1] 12100
## [1] 13310
## [1] 14641
## [1] 16105.1
## [1] 17715.61
## [1] 19487.17
## [1] 21435.89
## [1] 23579.48
## [1] 25937.42
```

While it does work (i.e. it produces a reasonable output) is difficult to understand what the code above does.

```
# Better:
investments <- 10000
yearly_return <- 0.1
years <- 1:10

for (year in years) {
  value <- investments*(1+yearly_return)^year
  print(paste(year, value, sep = ": "))
}
```

```
## [1] "1: 11000"
## [1] "2: 12100"
## [1] "3: 13310"
## [1] "4: 14641"
## [1] "5: 16105.1"
## [1] "6: 17715.61"
## [1] "7: 19487.171"
```

```
## [1] "8: 21435.8881"
## [1] "9: 23579.47691"
## [1] "10: 25937.424601"
```

This is better! Giving the variables proper names makes the purpose of the code and its output easier to understand.

Use the right names

This principle is similar to the one above – it’s a lot easier to make sense of code when variables, data frames and functions have descriptive names. To use an example from *The Art of Clean Code*, say you want to program a currency converter. The code in the chunk below is taken from StackOverflow (slightly amended to reflect conversion rates on Jan 3rd, 2023 and include CHF).

```
currencyCon <- function(x, from = "USD", to = "EUR"){
  # assign values: 1 usd in each currency
  values <- c(1.000000, 0.945799, 0.831797, 130.411928, 0.934267)
  # names attribute
  names(values) <- c("USD", "EUR", "GBP", "YEN", "CHF")
  # calculate (convert from into USD, and divide to)
  values[to] / (values[from] / x)
}
```

```
# Testing
currencyCon(1, "USD", "EUR")
```

```
##      EUR
## 0.945799
```

```
currencyCon(1, "EUR", "EUR")
```

```
## EUR
## 1
```

```
currencyCon(1, "GBP", "YEN")
```

```
##      YEN
## 156.7834
```

```
currencyCon(100, "CHF", "USD")
```

```
##      USD
## 107.0358
```

currencyCon is a more meaningful name than just `con()` or `fun()` or something along those lines, because it concisely describes what the function does.

Now let’s say you want a simpler function that transforms USD to EUR. The way it is done in the chunk below works, but is not ideal:

```
dollar_to_euro <- function(x){
  x*0.945799
}
```

First, the name is not unambiguous: *dollar* will likely be associated with USD in most of the world, but Canadians, Australians and people in Hong Kong among others also use a currency called dollar, which makes the name problematic. It’s better to use something like `usd_to_eur()`.

Furthermore, it’s discouraged to use numbers in functions, but rather define things like conversion rates beforehand. This will make changing it later easier, because you won’t have to go through the entire function to find the number to adjust. For small functions like this one, it does not make much of a difference, however.

```
conv_rate <- 0.945799

usd_to_euro <- function(x){
  x*conv_rate
}

usd_to_euro(1)
```

```
## [1] 0.945799
```

`conv_rate` is not exactly unambiguous, so the chunk above could be improved further still.

Adhere to standards and be consistent

If you go back to the code chunks above, you'll notice some inconsistencies in terms of naming conventions. In particular, the conversion function from StackOverflow uses a capital C to distinguish the two words from each other (*currency* and *Con*), whereas in the other chunks I've primarily used underscores for variable and function names. This is not ideal, since it is *inconsistent*.

Different people prefer different labelling conventions, but you should aim to be consistent with what you use and don't mix styles. Here are some examples:

- Use capital letters to distinguish words (e.g. `UsdToEur()`, `currencyCon()`)
- Use underscores (e.g. `usd_to_eur()`, `currency_con()`)
- Either use numbers directly or spell them out (e.g. `var1` OR `var_one`)
- Avoid mixing styles, e.g. `currency_Con()`, it will just make your life harder

Use comments (& avoid unnecessary comments)

```
text <- c("Ha! let me see her; out, alas! She's cold:
        Her blood is settled, and her joints are stiff,
        Life and these lips have long been separated:
        Death lies on her like an untimely frost
        Upon the sweetest flower of all the field.")
```

Look at the code below. While running it will tell you immediately what it does, it's hard to understand if you're not very familiar with regular expressions.

```
f_words <- str_extract_all(text, "\\bf\\w+\\b")

l_words <- str_extract_all(text, "\\bl\\w+\\b")
```

This is a situation where comments briefly outlining what each line does come in handy. Comments not only make it easier to understand code someone else wrote, but it can also help you getting back into an old script of yours.

```
# Find all words starting with letter 'f'
f_words <- str_extract_all(text, "\\bf\\w+\\b")
f_words
```

```
## [[1]]
## [1] "frost" "flower" "field"
```

```
# Find all words starting with letter 'l'
l_words <- str_extract_all(text, "\\bl\\w+\\b")
l_words
```

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

```
## [1] "let" "lips" "long" "lies" "like"
```

Comments are a useful tool for the readability of scripts. There is such a thing as excessive commenting, however. Unless you are using a script for teaching purposes, you best refrain from explaining every single line. Let's go back to the investment example and add some unnecessary comments.

```
investments <- 10000 # Your investments, change if needed
yearly_return <- 0.1 # Annual return (e.g., 0.1 --> 10%)
years <- 1:10 # Number of years to compound

# Go over each year
for (year in years) {
  # Calculate value of your investment in current year
  value <- investments*(1+yearly_return)^year
  # Print year and value of investment
  print(paste(year, value, sep = ": "))
}
```

```
## [1] "1: 11000"
## [1] "2: 12100"
## [1] "3: 13310"
## [1] "4: 14641"
## [1] "5: 16105.1"
## [1] "6: 17715.61"
## [1] "7: 19487.171"
## [1] "8: 21435.8881"
## [1] "9: 23579.47691"
## [1] "10: 25937.424601"
```

Because some thought was put into how to name the variables, the comments are quite superfluous and only add clutter to the code.

Don't repeat yourself (DRY)

```
print("Hello, world!")
```

```
## [1] "Hello, world!"
```

```
print("Hello, world!")
```

```
## [1] "Hello, world!"
```

```
print("Hello, world!")
```

```
## [1] "Hello, world!"
```

```
print("Hello, world!")
```

```
## [1] "Hello, world!"
```

```
print("Hello, world!")
```

```
## [1] "Hello, world!"
```

```
for (i in 1:5) {
  print("Hello, world!")
}
```

```
## [1] "Hello, world!"
```

```
## [1] "Hello, world!"
```

```
## [1] "Hello, world!"
## [1] "Hello, world!"
## [1] "Hello, world!"
```

The code chunks above do the same thing, but the last one avoids unnecessary clutter. In general, I recommend using loops or the `apply()`-family of functions to perform a task repeatedly.

Another useful tool to avoid repeating yourself are *functions*. In the code chunk below, the same multiplication by 1.60934 is used twice.

```
miles = 100
kilometers = miles * 1.60934

distance = 20 * 1.60934

print(kilometers)
```

```
## [1] 160.934
print(distance)
```

```
## [1] 32.1868
```

Let's write a function, which will get rid of the repetition and is easier to maintain.

```
miles_to_km <- function(miles){
  return(miles*1.60934)
}
```

```
miles = 100
miles_to_km(100)
```

```
## [1] 160.934
distance <- miles_to_km(20)
print(distance)
```

```
## [1] 32.1868
```

Not only is the repetition gone, but the code is easier to maintain this way. Should you wish to update e.g. a conversion rate (unlikely for miles to km, more likely for currencies etc.) it only needs to be done once, within the function, and not every time you perform a conversion.

Alternatively, you can define the conversion rate beforehand, like we did earlier. Either way, you'll only have to change the rate once, which reduces the potential for errors.

Bibliography

Mayer, C. (2022): The Art of Clean Code. San Francisco: No Starch Press, Inc.