Preparing your defences

DEFENSIVE R PROGRAMMING

Dr. Colin Gillespie
Jumping Rivers

datacamp

Preparing your defenses

> The wise warrior avoids the battle.
>
> ? **Sun Tzu**, The Art of War

Avoid problems!

- We all make mistakes
- Let's minimise the number!

Are you wet or dry

- DRY: a standard principle of software development
  - Do not repeat yourself
- WET: write everything twice
  - We enjoy typing

The copy and paste rule

1. Copying and pasting once is OK
2. Twice is suspect
3. Three times is almost always wrong

Functions and for loops

Whenever you copy & paste

- A function
- Or a for loop

Let's see this in action

DEFENSIVE R PROGRAMMING

Just one comment

DEFENSIVE R PROGRAMMING

Colin Gillespie
Jumping Rivers

I don't know about you...

- Code that is obvious today
- Is often a lot less obvious in a few weeks times

Comments

- You can add comments to your R code via `#`
- It turns out that writing good comments is tricky!

```
# This is a comment
# The above comment isn't very helpful
# Or is it?
```

Tip 1: Avoid obvious comments

- What's obvious is sometimes hard to decide
  - For example, the comments

```
# Loop through data sets
for (dataset in datasets) {
 # Read in data set
 r <- read.csv(dataset)
}
```

    look reasonable

  - But are perhaps a little too obvious

Tip 2: Avoid comments that you will never update

The most common example is header comments at the top of the file

```
# Last updated: 1967-02-25
# Author: D Law
# Status: No 1
```

- These sorts of comments are almost never updated
- I once saw `# list of packages used: XXX, YYY`

Tip 3: Be consistent

- Always start with a single `#` or double `##`
- Start with a capital letter - follow the rules of grammar
- Be careful with jokes
  - What you find funny, others may take offense
- Be sure to comment on code that "looks wrong"
- Use **# TODO** or **# XXX** to indicate a future problem

Let's practice

DEFENSIVE R PROGRAMMING

A little bit dotty

DEFENSIVE R PROGRAMMING

Dr. Colin Gillespie
Jumping Rivers

The full stop

In R, the full stop has a very special meaning

- It is the mechanism that is used in S3 OOP
- When you call the `summary()` function
  - R looks for the function `summary.class_name`

Example: the summary() function

When you call

```
m <- lm(mpg ~ disp, data = mtcars)
```

```
class(m)
```

```
#[1] "lm"
```

Example: the summary() function

So when you call

```
summary(m)
```

you end up calling

```
summary.lm(m)
```

The **key** point here, is that the full stop is very important

One bit of advice

- There are few R rules that everyone agrees on
- But **everyone** agrees that you should avoid `.` in variable names
- It just prevents confusion

The final stop

DEFENSIVE R PROGRAMMING

Coding Style

DEFENSIVE R PROGRAMMING

Dr. Colin Gillespie
Jumping Rivers

Consistency

Everyone agrees that consistency is key

This may mean changing styles when in different teams!

Uncontroversial rules

- Assignment wars: `=` vs `->`

```
x = 5
# or
x <- 5
```

- Everyone agrees you shouldn't mix & match
- I prefer the superior `=` for assignment but
- DataCamp prefers `<-` for their courses

So be **consistent**

Spacing

Consistent spacing makes code far easier to read

Compare

```
res<-t.test(x,paired=FALSE)
```

with

```
res <- t.test(x, paired = FALSE)
```

Spacing

Two widely accepted rules are

- spaces around assignment `x <- 5`
- spaces after a comma - `x[1, 1]` instead of `x[1,1]`

Let's practice!

DEFENSIVE R PROGRAMMING

Static Code Analysis for R

DEFENSIVE R PROGRAMMING

Dr. Colin Gillespie
Jumping Rivers

The lintr Package

- **lintr** is an R package offering static code analysis for R
  - It checks adherence to a
    - given style
    - syntax errors
    - possible semantic issues

Similar to how spell checkers work

Using lintr

To use **lintr**

- We store the code in a file
- Pass the code to the `lint()` function

lintr in Action

Suppose I have the following code

```
my_bad<-function(x, y) {
  x+y
}
```

saved in the file `code.R`.

- Running `lint::lintr("code.R")` highlights two issues

Issue 1

```
my_bad<-function(x,y) {
  x+y
}
```

```
r[[1]]
tmp.R:1:7: style: Put spaces around all infix operators.
my_bad<-function(x,y) {
     ~^~~
```

```
my_bad <- function()
```

Issue 2

```
my_bad<-function(x,y) {
  x+y
}
```

```
r[[3]]
tmp.R:2:4: style: Put spaces around all infix operators.
  x+y
  ~^~
```

```
 x + y
```

Let's see Lintr in Action

DEFENSIVE R PROGRAMMING