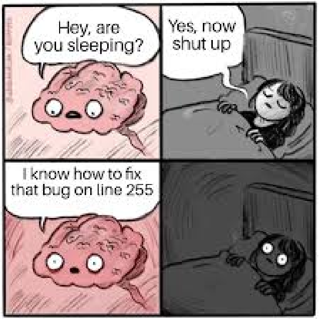

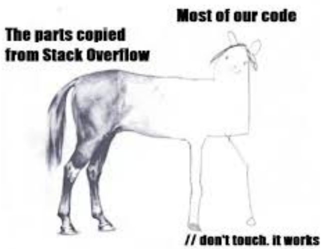

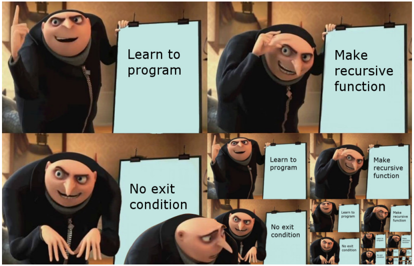

That last one will make sense in 2h

Programming with functions in R

Useful things, funny things and stat theory

Timothée Bonnet

March 14, 2019

1. Functions and where to find them

2. Funny things

Why make your own functions?

Pros

- Less code writing
- Fewer mistakes
- Cleaner code
- More transferable code

Hence reproducibility and mental health

Why make your own functions?

Pros

- Less code writing
- Fewer mistakes
- Cleaner code
- More transferable code

Hence reproducibility and mental health

Cons

- More thinking
- Not always worth time investment

Anatomy of a function

What is inside a function?

```
  mean
  apply
```

Anatomy of a function

What is inside a function?

```
mean
apply
```

Hmm, clearer information?

```
?apply
?apply()
```

Anatomy of a function

What is inside a function?

```
mean
apply
```

Hmm, clearer information?

```
?apply
?apply()
```

Lots of code in one word

How to make a function

```
myfunction <- function(){
  3+5
}

myfunction()

## [1] 8
```

`myfunction` is now an object in the environment

Input/output

- Zero to many input objects (Arguments)
- Return one object (Value)

```r
mean(x = c(3,7,1), na.rm = TRUE)
```

Input/output

- Zero to many input objects (Arguments)
- Return one object (Value)

```
  mean(x = c(3,7,1), na.rm = TRUE)
```

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

myfunction(2, 4); myfunction(3, 5)

## [1] 6
## [1] 8
```

`myfunction` now takes two arguments

Input/output

- Zero to many input objects (Arguments)
- Return one object (Value)

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

myfunction(2, 4); myfunction(3, 5)

## [1] 6
## [1] 8
```

Functions returns only the result from the last line by default

Input/output

- Zero to many input objects (Arguments)
- Return one object (Value)

```
myfunction <- function(x, y){
  subvalue <- x-y
  advalue <- x+y
  return(subvalue)
  x*y
}

myfunction(2, 4); myfunction(3, 5)

## [1] -2
## [1] -2
```

Functions returns only `return()` is one is provided

Exercise 1 and 2

1. Write a function that return the product of three arbitrary numbers together (x*y*z) provided by the user
2. Write a function that return the product of three arbitrary numbers together (x*y*z) as well as their sum (x+y+z)

Scope

”What happens in Functions Stays in Functions” (unless…)

```
x <- 10
myfunction <- function(x){ x <- 5 }
myfunction(x=x)
x
```

What is the value of x ?

Scope

”What happens in Functions Stays in Functions” (unless. . .)
Save the output to an object

```
x <- 10
x <- myfunction(x=x)
x

## [1] 5
```

Scope

”What happens in Functions Stays in Functions” (unless...)
Save the output to an object

```
x <- 10
x <- myfunction(x=x)
x

## [1] 5
```

Or special functions to break environment boundaries (Scoping assignment, see later)

Sourcing as ”primitive package”

Do Exercise 3

Sourcing as "primitive package"

Do Exercise 3

Complex code can easily be turned into a function

Sourcing as ”primitive package”

Do Exercise 3

Complex code can easily be turned into a function

Save the function you just made to a new file ”myfunctions.R”.
You can now call (”source”) this file and all the functions it contains:

```
source("myfunctions.R")
```

Fun time: Big exercise and stat theory!

Exercise 4!

1. Functions and where to find them

2. Funny things

Lists of functions!

```r
funs <- list(
  half = function(x) x / 2,
  double = function(x) x * 2
)
funs$double(10)

## [1] 20
```

The dot-dot-dot

See exercise 5

Scoping assignment

Using `<<-` or `assign(x, value, inherits=TRUE)`
See exercise 6
Can be difficult, but can be useful (e.g. functions that create functions)

Recursive function

A function can call another function, including itself

See Exercises 7 and 8

To go further

Everything you (didn't) want to know about functions in R:
`https://adv-r.hadley.nz/functions.html#introduction-5`