Writing functions in R

Brendan Clarke

brendan.clarke2@nhs.scot

NHS Education for Scotland

18/06/2024

KIND Learning Network

Welcome!

- this session is an 🌶🌶 **intermediate practical** designed for those with some R experience
- we'll get going properly at 14.35
- practical and interactive, so please chat away - and get whatever version of R you're using ready
- if you can't access the chat, you might need to join our Teams channel: **tinyurl.com/kindnetwork**

KIND Learning Network
NHS Education for Scotland

Future R training sessions

Session	Date	Level
Scope of the possible with R	15:00-16:00 Mon 24th June 2024	💼: **non-technical**
Iteration in R	09:30-11:00 Fri 5th July 2024	🌶🌶 : **intermediate-level**
Getting more out of dplyr	10:30-12:00 Wed 17th July 2024	🌶🌶 : **intermediate-level**

KIND Learning Network
NHS Education for Scotland

Session outline

- why functions?
- basic syntax
- adding arguments
- vectorised functions
- the mystery of the paired brackets
- …

Why functions?

- most beginners write repetitious code
- repetitious code is hard to maintain
- functions give you an easy way of repeating chunks of code

Basic syntax

- think of this as a way of repeating yourself
- in time-honoured fashion…

```
1 hi_wrld <- function(){
2   "hello world"
3 }
4
5 hi_wrld()
```

```
[1] "hello world"
```

Adding arguments

- most of the time, you'll want to add **arguments** to your function
  - add a variable name inside the round bracket of `function`
  - use that variable name in your function body

```
1 hi_wrld_n <- function(n){
2   paste(rep("hello world", n))
3 }
4
5 hi_wrld_n(4)
```

```
[1] "hello world" "hello world" "hello world" "hello world"
```

KIND Learning Network
NHS Education for Scotland

Another argument

- you can add another argument
- either position or name can be used in the function call

```
hi_name_n <- function(name, n){
  rep(paste("hello", name) , n)
}

hi_name_n("sue", 4)
```

```
[1] "hello sue" "hello sue" "hello sue" "hello sue"
```

Even...

```
hi_name_n(n = 3, name = "tango") # evil but legal
```

```
[1] "hello tango" "hello tango" "hello tango"
```

Defaults

```
1 hi_name_n_def <- function(n, name = "janelle"){
2   rep(paste("hello", name) , n)
3 }
4
5 hi_name_n_def(n = 4)
```

```
[1] "hello janelle" "hello janelle" "hello janelle" "hello janelle"
```

```
1 hi_name_n_def(n = 2, name = "bruce")
```

```
[1] "hello bruce" "hello bruce"
```

Vectorised functions

- most functions in R are vectorised

```
round(c(1.2, 3.2, 5.4, 2.7), 0)
```

```
[1] 1 3 5 3
```

Vectorised functions

- that means that mostly, our functions will end up vectorised without us doing any work at all

```
1 div_seven_n_round <- function(nums){
2   round(nums / 7, 0)
3 }
4
5 numbers <- rnorm(4, 5, 50)
6
7 numbers
```

```
[1] -38.50315 -56.07585 -45.53310 -17.99504
```

```
1 div_seven_n_round(numbers)
```

```
[1] -6 -8 -7 -3
```

Vectorised functions

- **but** there are a few cases where that can fail: most famously, using `if`/`else`

```
is_even <- function(n){

  if(n %% 2){
  paste(n, "is odd")
} else {
  paste(n, "is even")
}

}
is_even(9)
```

```
[1] "9 is odd"
```

```
is_even(10)
```

```
[1] "10 is even"
```

```
try(is_even(9:10))
```

```
Error in if (n%%2) { : the condition has length > 1
```

KIND Learning Network

NHS Education for Scotland

vectorize with `Vectorize`

```
1  is_even_v <- Vectorize(is_even)
2  is_even_v(9:10)
```

```
[1] "9 is odd"   "10 is even"
```

apply

- apply with `lapply` / `purrr::map` with `Vectorize`

```
1  lapply(9:10, is_even)
```

```
[[1]]
[1] "9 is odd"

[[2]]
[1] "10 is even"
```

```
1  purrr::map(9:10, is_even)
```

```
[[1]]
[1] "9 is odd"

[[2]]
[1] "10 is even"
```

refactor

- refactor to avoid scalar functions

```
1 is_even_rf <- function(n){
2   ifelse(n %% 2, paste(n, "is odd"), paste(n, "is even"))
3 }
4 is_even_rf(9:10)
```

```
[1] "9 is odd"   "10 is even"
```

KIND Learning Network

what's the problem with {{}}?

```
mtcars |>
  dplyr::summarise(average = round(mean(hp)))
```

```
  average
1     147
```

- so

```
carmo <- function(column){
  mtcars |>
    dplyr::summarise(average = round(mean(column)))
  }
```

- but...

```
try(carmo(hp))
```

```
Error in dplyr::summarise(mtcars, average = round(mean(column))) :
  ℹ In argument: `average = round(mean(column))`.
Caused by error:
! object 'hp' not found
```

KIND Learning Network

NHS Education for Scotland

- we get used to R (and particularly tidyverse) helping us with some sugar when selecting column by their names
  - `mtcars$hp` / `mtcars |> select(hp)`
  - effectively, we're just able to specify `hp` like an object, and R figures out the scope etc for us
- that misfires inside functions. R isn't sure where to look for an object called `hp`

Enter {{}}

```
carmo_woo <- function(column){
  mtcars |>
    dplyr::summarise(average = round(mean({{column}})))
}

carmo_woo(hp)
```

```
  average
1     147
```

- for 95% of purposes, take {{}} as a purely empirical fix
- but, if you're very enthusiastic:
  - {{}} defuses and injects the column name
  - equivalent to !!enquo(var)

KIND Learning Network

- pass arbitrary arguments into/through a function with …

```
dotty <- function(n, ...){
  rep(paste(..., collapse = ""), n)
}

dotty(4, letters[1:5])
```

```
[1] "abcde" "abcde" "abcde" "abcde"
```

Resources

- **best = home made!** Refactor something simple in your code today.
- hard to beat the treatment of functions in R4DS
- the Rlang page on data masking is surprisingly sane for such a complicated area

Feedback

Writing functions in R feedback link

KIND Learning Network
NHS Education for Scotland