The purrr package in R

Brendan Clarke, NHS Education for Scotland, brendan.clarke2@nhs.scot

29/07/2024

KIND Learning Network

Welcome

- this session is for 🌶🌶 intermediate users
- we’ll get going properly at 15.05
- you’ll need R + Rstudio / Posit Workbench / posit.cloud to
- if you can’t access the chat, you might need to join our Teams channel: tinyurl.com/kindnetwork
- you can find session materials at tinyurl.com/kindtrp

The KIND network

- a social learning space for staff working with **k**nowledge, **i**nformation, and **d**ata across health, social care, and housing in Scotland
- we offer social support, free training, mentoring, community events, …
- Teams channel / mailing list

KIND Learning Network

NHS Education for Scotland

R training sessions

Session	Date	Area	Level
Testing R code	15:00-16:30 Wed 7th August 2024	R	🌶🌶 : intermediate-level
Flexdashboard	13:00-14:30 Thu 15th August 2024	R	🌶🌶 : intermediate-level

KIND Learning Network

NHS Education for Scotland

Session outline

- a digression about Linnaeus
- functionals
- base-R functional programming
- `map` and `walk`
- `map2` and `pmap`
- niceties and add-ons

A digression about Linnaeus

Functionals

Here are some numbers:

```r
n1 <- 7:9
```

Let’s find their average. We’d usually do this by passing those numbers to a function:

```r
mean(n1)
```

```
[1] 8
```

Functionals

But in R, interestingly, we can also do this the other way round by passing a function name:

```r
my_num_f <- function(funct = mean) funct(n1)
my_num_f(mean)
```

```
[1] 8
```

```r
my_num_f(sum)
```

```
[1] 24
```

We'd describe this as a functional. It's fun, but a bit messy and annoying (e.g. how to change the numbers you're averaging??).

KIND Learning Network

Functional programming in base R

Say we've got a function we want to apply:

```r
round_root <- function(n) round(n ^ 0.5, 1)
```

There are several ways of applying functions to stuff in base R. + we could use a loop: that's another session + we could just exploit the vectorised nature of most functions in R

```r
round_root(n1)
```

```
[1] 2.6 2.8 3.0
```

- or we could use some of the `apply` family of functions

KIND Learning Network

lapply and sapply

```r
lapply(n1, round_root) # returns a list
```

```
[[1]]
[1] 2.6

[[2]]
[1] 2.8

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

```r
sapply(n1, round_root) # simplifies that list to a vector
```

```
[1] 2.6 2.8 3.0
```

lapply and sapply

There's no real reason to use these functions when things are this simple, but when our applications become more complicated…

```r
n2 <- 11:13

lapply(list(n1, n2), round_root)
```

```
[[1]]
[1] 2.6 2.8 3.0

[[2]]
[1] 3.3 3.5 3.6
```

```r
sapply(list(n1, n2), round_root) # oddball output
```

```
     [,1] [,2]
[1,]  2.6  3.3
[2,]  2.8  3.5
[3,]  3.0  3.6
```

```r
lapply(list(n1, n2[1:2]), round_root) # quirky
```

```
[[1]]
[1] 2.6 2.8 3.0
```

KIND Learning Network
NHS Education for Scotland

```
[[2]]
[1] 3.3 3.5
```

KIND Learning Network

NHS Education for Scotland

purrr

- purrr is a functional programming toolkit
- main advantage = **consistency**
- very useful cheatsheet

map

map is our purrr type specimen

```r
library(purrr)
map(n1, round_root)
```

```
[[1]]
[1] 2.6

[[2]]
[1] 2.8

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

map

Pleasingly, map will handle all kinds of odd inputs without fuss:

```
map(c(n1, n2), round_root)
```

```
[[1]]
[1] 2.6

[[2]]
[1] 2.8

[[3]]
[1] 3

[[4]]
[1] 3.3

[[5]]
[1] 3.5
```

```
map(dplyr::tibble(n1 = n1,
```

```
$n1
[1] 2.6 2.8 3.0

$n2
[1] 3.3 3.5 3.6
```

```
map(rbind(n1, n2), round_r
```

```
[[1]]
[1] 2.6

[[2]]
[1] 3.3

[[3]]
[1] 2.8

[[4]]
[1] 3.5

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

KIND Learning Network

map

map will always return a list - that's because, no matter what the output, you can always cram it into a list. If you want different output, you can have it. You just need to find the right *species*:

```r
map_vec(n1, round_root)
```

```
[1] 2.6 2.8 3.0
```

```r
try(map_int(n1, round_root)) # surly and strict
```

```
Error in map_int(n1, round_root) : ℹ In index: 1.
Caused by error:
! Can't coerce from a number to an integer.
```

```r
round_root_int <- function(n) as.integer(n ^ 0.5)
map_int(n1, round_root_int)
```

```
[1] 2 2 3
```

```r
round_root_lgl <- function(n) as.integer(n ^ 0.5) %% 2 == 0
map_lgl(n1, round_root_lgl)
```

```
[1]  TRUE  TRUE FALSE
```

KIND Learning Network

anonymous functions

If you’re comfortable with the new anonymous function syntax, you can build an anonymous function in place:

```r
map_lgl(1:4, \(x) x %% 2 == 0)
```

```
[1] FALSE  TRUE FALSE  TRUE
```

walk

walk is intended for code where the side-effect is the point: graphs, pipes, and Rmarkdown especially. Otherwise, it’s as map:

```r
walk(n1, round_root) # wtf?
round_root_print <- function(n) print(n ^ 0.5)
walk(n1, round_root_print)
```

```
[1] 2.645751
[1] 2.828427
[1] 3
```

```r
round_root_cat <- function(n) cat(n ^ 0.5, "   \n")
walk(n1, round_root_cat)
```

```
2.645751
2.828427
3
```

KIND Learning Network

map2

map2 is for 2-argument functions:

```r
map2_int(n1, n2, `+`) # the best terrible way of adding I know
```

```
[1] 18 20 22
```

```r
round_root_places <- function(n, dp = 1) round(n ^ 0.5, dp)
round_root_places(n1, 0)
```

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

```r
map2(n1, 0, round_root_places)
```

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

[[2]]
[1] 3

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

KIND Learning Network

beware of recycling rules

You'll be unable to use `map2` if your inputs are different lengths:

```r
try(map2(1:3, 0:3, round_root_places))
```

```
Error in map2(1:3, 0:3, round_root_places) :
  Can't recycle `.x` (size 3) to match `.y` (size 4).
```

This makes `expand.grid` valuable if you're looking to try out all the combinations of two vectors, for example.

```r
dat <- expand.grid(nums = 1:3, dplaces = 0:3)

map2(dat$nums, dat$dplaces, round_root_places)
```

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

[[2]]
[1] 1

[[3]]
[1] 2
```

KIND Learning Network

```
[[4]]
[1] 1

[[5]]
[1] 1.4
```

```r
# or in a tibble
expand.grid(nums = n1, dplaces = 0:3) |>
  dplyr::as_tibble() |>
  dplyr::mutate(rr = map2_vec(nums, dplaces, round_root_places))
```

```
# A tibble: 12 × 3
    nums dplaces    rr
   <int>   <int> <dbl>
 1     7       0  3
 2     8       0  3
 3     9       0  3
 4     7       1  2.6
 5     8       1  2.8
 6     9       1  3
 7     7       2  2.65
 8     8       2  2.83
 9     9       2  3
10     7       3  2.65
11     8       3  2.83
12     9       3  3
```

pmap

pmap is for n argument functions.

```r
round_roots_places <- function(n, root = 2, places = 1) round(n ^ 1/root, places)

round_roots_places(n1, root = 4, places = 2) # use named arguments to avoid misery
```

```
[1] 1.75 2.00 2.25
```

```r
pmap(list(n = n1, root = 4, places = 2), round_roots_places)
```

```
[[1]]
[1] 1.75

[[2]]
[1] 2

[[3]]
[1] 2.25
```

KIND Learning Network

NHS Education for Scotland

Niceties and addons

```r
imap(list("a", "b", "c"), \(x, y) paste0(y, ": ", x)) |> # index map where y is the name or index
  list_c()
```

```
[1] "1: a" "2: b" "3: c"
```

```r
map(n1, \(x) dplyr::tibble("Val" = x, "sq_val" = x^2)) |>
  list_rbind()
```

```
# A tibble: 3 × 2
    Val sq_val
  <int>  <dbl>
1     7     49
2     8     64
3     9     81
```

KIND Learning Network

NHS Education for Scotland

Feedback and resources

- functionals chapter in Advanced R
- purrr cheatsheet
- please can I ask for some feedback - takes less than a minute, completely anonymous, helps people like you find the right training for them

```r
#KINDR::training_sessions("Excel", "2024/07/11")
```

KIND Learning Network