R functionals

Alberto Garfagnini

Università di Padova

Advanced R 02

Functionals basics

Definition

a FUNCTIONAL is a function that
takes FUNCTION as INPUT
and returns a VECTOR as OUTPUT

- example:

```
randomize <- function(f) f(runif(10^3))

randomize(mean)
#> [1] 0.4954407
randomize(mean)
#> [1] 0.491658

randomize(sum)
#>[1] 507.5148
```

- typical examples in base R:

  `lapply()`, `apply()` and `tapply()`

- other example: `integrate()`

```
integrate(dnorm, -Inf, Inf)
#> 1 with absolute error < 9.4e-05
```

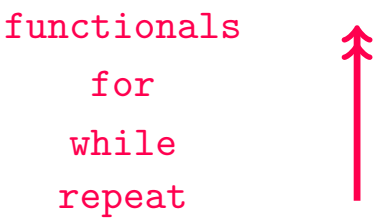
Functionals : replacement for `loops` ?

a common use of functionals is as alternative to for loops

NOTE

- `for` loops are not slow by themselves
- what makes them slow is what programmers do inside the for loop body

ex: modifying a data structure makes the loop slow because each modification creates a copy: `copy-on-modify`



- switching from loop to functional is a pattern matching exercise:
  goal: find a functional that matches the basic loop form

Our first functions: `purrr::map()`

- it takes a vector and a function
- it calls the function for each vector element
- it returns the results in a list

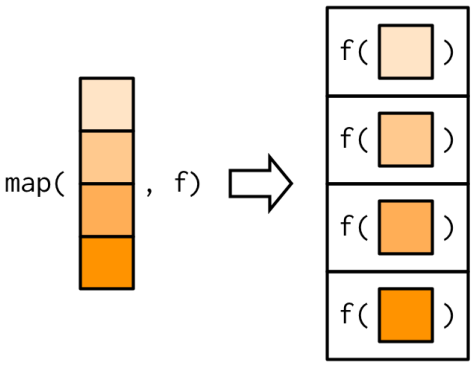

```
purrr::map(1:10, f)
```

is equivalent to

```
list(f(1), f(2), ..., f(10))
```

```
double <- function(x) x*2
xd <- purrr::map(1:10, double)
str(xd)
#>List of 10
#> $ : num 2
#> $ : num 4
...
#> $ : num 18
#> $ : num 20

unlist(xd)
#> [1]  2  4  6  8 10 12 14 16 18 20
```



Example 1

- we have a tibble with different data sets

```
dt <- tibble( a1 = rnorm(10), b1 = runif(10),
              c1 = rpois(10, 3.7), d1 = rbeta(10, 0.3, 5) )
```

- we want to evaluate the median of each colum

```
omed <- vector("double", ncol(dt))
omed
#> [1] 0 0 0 0
for (i in seq_along(dt)) {
    omed[[i]] <- median(dt[[i]])
}
omed
#> [1] 0.165312063 0.487255521 4.000000000 0.009203981
```

- it's possible to wrap up for loops in a function, and call that function instead of using the for loop directly

```
purrr::map_dbl(dt, median)
#>          a1          b1          c1          d1
#> 0.165312063 0.487255521 4.000000000 0.009203981
```

- all the `map_*()` functions use `...` to pass along additional arguments to `.f` each time it's called

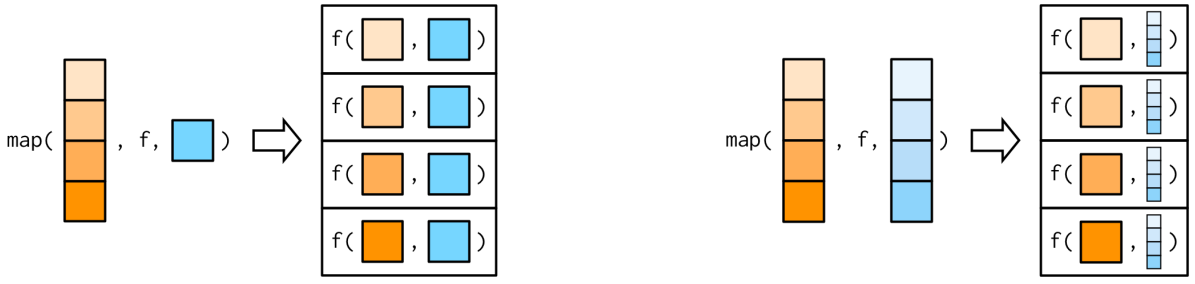
```
purrr::map_dbl(dt, mean, trim=0.5)
#>          a1          b1          c1          d1
#> 0.165312063 0.487255521 4.000000000 0.009203981
```


map()

the function `map()` returns a list:

```
     a list or vector
    ||
    \/
map(.x, .f, ...)
       /\   |-----> pass additional arguments to .f each
       ||           time it is called
       ||
       a function, formula or vector
```

- `map_lgl()`, `map_int()`, `map_dbl()` and `map_chr()` return a vector of specific type (logical, integer, double or character)
- `map_dfr()` and `map_dfc()` return a data frame created by row or by column
- any arguments that come after `f` in the call to `map()` are inserted after the data in individual calls to `f()`



Example 2: map()

- we generate 10 sets of random numbers from a probability distribution

```
1:10 %>% map(rnorm, n=20) -> l1
```

- this can be done using an anonyous function

```
1:10 %>% map(function(x) rnorm(n=20, x)) -> l2
```

- or by using a one-sided formula

```
1:10 %>% map( ~ rnorm(n=20, .x) ) -> l3
```

- there are a few shortcuts that you can use with `.f` in order to save a little typing
  - `.x` and `.y` are used for two argument functions, and `..1`, `..2`, `..3`, `...` for all the additional arguments

- `map()` can be chained:

```
 1:10 %>%
      map(rnorm, n=20) %>%
      map_dbl(mean)
#> [1]  0.8355395  2.0266397  3.1451209  3.9854774  5.0977312
#> [6]  6.0904780  6.9547342  8.4906865  8.9917292 10.1268192
```


Mapping over multiple arguments: map2()

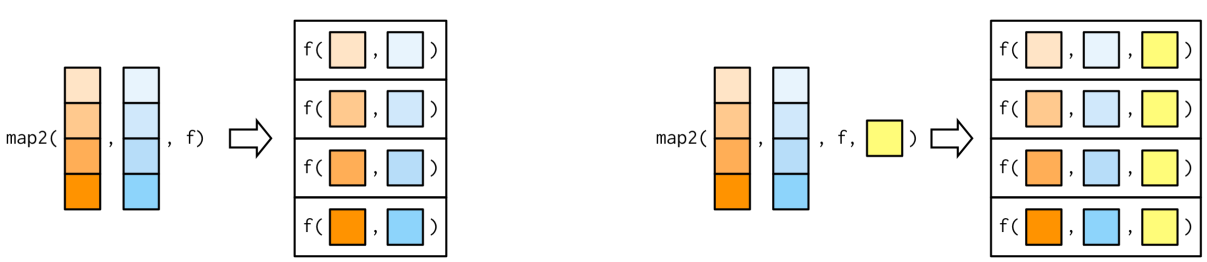
- as an example we want to generate several data sets from a normal distribution with different mean and variance

```
mus <- list(5, 10, -3)
sigmas <- list(1, 5, 10)
map2(mus, sigmas, rnorm, n = 5) %>% str()
#> List of 3
#>  $ : num [1:5] 4.17 5.24 5.54 4.8 5.44
#>  $ : num [1:5] 12.71 7.01 9.56 7.25 10.74
#>  $ : num [1:5] -8.72 9.89 -14.54 3.51 -9.49
```

- the same results could have done iterating over indices

```
seq_along(mus) %>%
              map( ~rnorm(5, mus[[.]], sigmas[[.]]) ) %>% str()
#> List of 3
#>  $ : num [1:5] 4.73 6.52 2.68 5.42 5.35
#>  $ : num [1:5] 14.913 -0.695 11.702 17.911 1.795
#>  $ : num [1:5] -4.14 -1.08 -7.85 5.79 13.73
```

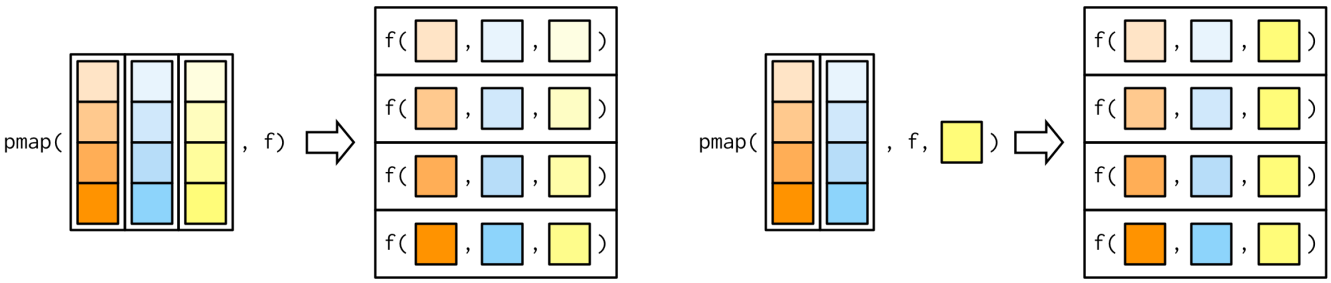
- but the code with map2() is simpler and cleaner



additional functions: `pmap()` and `imap()`

- in case of multiple arguments, purrr provides `pmap()` which takes a list of arguments
- if you don't name the list's elements, `pmap()` will use positional matching when calling the function. This makes the code harder to read ➜ use named arguments:

```
args2 <- list(mean = c(5, 10, -3),
              sd = c(1, 5, 10), n = 5)
args2 %>%
          pmap(rnorm) %>%
                str()
#> List of 3
#>  $ : num [1:5] 5.38 4.54 3.85 5.44 5.42
#>  $ : num [1:5] 17.038 6.072 15.107 0.697 6.488
#>  $ : num [1:5] -10.7 2.99 -1.12 17.47 13.08
```



Invoking different functions: `invoke_map()`

- a setp up in complexity is to invoke different functions with different parameters (values and meanings):

```
fgen <- c("runif", "rnorm", "rpois")

fpar <- list(
            list(min = -1, max = 1),
            list(sd = 3),
            list(lambda = 7.5))

invoke_map(fgen, fpar, n = 5) %>% str()

#> List of 3
#>  $ : num [1:5] -0.7744 -0.0524 0.7523 0.5074 0.5284
#>  $ : num [1:5] 3.162 -2.766 -0.298 -2.849 -2.638
#>  $ : int [1:5] 5 7 10 6 4
```

Example: reading different files and combine in one data.frame

- our data is organized in text files accoring to different years:
- data_2020_Italy.csv, data_2021_Italy.csv
- we want to read the data and combine them in one data.frame

```
read_my_csv <- function(year, country) {
    filename <- paste0(year, "_", country, ".csv")
    mobdata_dir <- "./Region_Mobility_Report_CSVs"
    filepath <- file.path(mobdata_dir, filename)
    message(paste("Reading␣from␣file:", filepath))
    read_csv(filepath)
}

years <- 2020:2021
country <- "Italy"

mbdata <- map_df(years, read_my_csv, country)
```

```
Reading from file: ./Region_Mobility_Report_CSVs/2020_IT.csv
...
Reading from file: ./Region_Mobility_Report_CSVs/2021_IT.csv
...
```