11_Function_Operations

2023-02-21

11 Function operators

11.1 Introduction

```
chatty <- function(f) {
  force(f)

  function(x, ...) {
    res <- f(x, ...)
    cat("Processing ", x, "\n", sep = "")
    res
  }
}
f <- function(x) x ^ 2
s <- c(3, 2, 1)

purrr::map_dbl(s, chatty(f))
```

```
## Processing 3
## Processing 2
## Processing 1
```

```
## [1] 9 4 1
```

```
library(purrr)
library(memoise)
```

11.2 Existing function operators

11.2.1 Capturing errors with purrr::safely()

```
x <- list(
  c(0.512, 0.165, 0.717),
  c(0.064, 0.781, 0.427),
  c(0.890, 0.785, 0.495),
  "oops"
)

out <- rep(NA_real_, length(x))
```

```
for (i in seq_along(x)) {
  out[[i]] <- sum(x[[i]])
}
```

```
## Error in sum(x[[i]]): invalid 'type' (character) of argument
```

```
out
```

```
## [1] 1.394 1.272 2.170    NA
```

```
map_dbl(x, sum)
```

```
## Error in `map_dbl()`:
## i In index: 4.
## Caused by error:
## ! invalid 'type' (character) of argument
```

```
safe_sum <- safely(sum)
safe_sum
```

```
## function (...)
## capture_error(.f(...), otherwise, quiet)
## <bytecode: 0x0000020177950028>
## <environment: 0x000002017795baf0>
```

```
str(safe_sum(x[[1]]))
```

```
## List of 2
##  $ result: num 1.39
##  $ error : NULL
```

```
str(safe_sum(x[[4]]))
```

```
## List of 2
##  $ result: NULL
##  $ error :List of 2
##   ..$ message: chr "invalid 'type' (character) of argument"
##   ..$ call   : language .Primitive("sum")(..., na.rm = na.rm)
##   ..- attr(*, "class")= chr [1:3] "simpleError" "error" "condition"
```

```
out <- map(x, safely(sum))
str(out)
```

```
## List of 4
##  $ :List of 2
##   ..$ result: num 1.39
##   ..$ error : NULL
##  $ :List of 2
##   ..$ result: num 1.27
```

```
##   ..$ error : NULL
##  $ :List of 2
##   ..$ result: num 2.17
##   ..$ error : NULL
##  $ :List of 2
##   ..$ result: NULL
##   ..$ error :List of 2
##   .. ..$ message: chr "invalid 'type' (character) of argument"
##   .. ..$ call   : language .Primitive("sum")(..., na.rm = na.rm)
##   .. ..- attr(*, "class")= chr [1:3] "simpleError" "error" "condition"
```

```
out <- transpose(map(x, safely(sum)))
str(out)
```

```
## List of 2
##  $ result:List of 4
##   ..$ : num 1.39
##   ..$ : num 1.27
##   ..$ : num 2.17
##   ..$ : NULL
##  $ error :List of 4
##   ..$ : NULL
##   ..$ : NULL
##   ..$ : NULL
##   ..$ :List of 2
##   .. ..$ message: chr "invalid 'type' (character) of argument"
##   .. ..$ call   : language .Primitive("sum")(..., na.rm = na.rm)
##   .. ..- attr(*, "class")= chr [1:3] "simpleError" "error" "condition"
```

```
ok <- map_lgl(out$error, is.null)
ok
```

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

```
x[!ok]
```

```
## [[1]]
## [1] "oops"
```

```
out$result[ok]
```

```
## [[1]]
## [1] 1.394
##
## [[2]]
## [1] 1.272
##
## [[3]]
## [1] 2.17
```

```r
fit_model <- function(df) {
  glm(Petal.Length ~ Petal.Width + Sepal.Length * Sepal.Width, data = df)
}

datasets <- list(iris, PlantGrowth, attitude)

models <- transpose(map(datasets, safely(fit_model)))
ok <- map_lgl(models$error, is.null)

# which data failed to converge?
datasets[!ok]
```

```
## [[1]]
##    weight group
## 1    4.17  ctrl
## 2    5.58  ctrl
## 3    5.18  ctrl
## 4    6.11  ctrl
## 5    4.50  ctrl
## 6    4.61  ctrl
## 7    5.17  ctrl
## 8    4.53  ctrl
## 9    5.33  ctrl
## 10   5.14  ctrl
## 11   4.81  trt1
## 12   4.17  trt1
## 13   4.41  trt1
## 14   3.59  trt1
## 15   5.87  trt1
## 16   3.83  trt1
## 17   6.03  trt1
## 18   4.89  trt1
## 19   4.32  trt1
## 20   4.69  trt1
## 21   6.31  trt2
## 22   5.12  trt2
## 23   5.54  trt2
## 24   5.50  trt2
## 25   5.37  trt2
## 26   5.29  trt2
## 27   4.92  trt2
## 28   6.15  trt2
## 29   5.80  trt2
## 30   5.26  trt2
##
## [[2]]
##    rating complaints privileges learning raises critical advance
## 1      43         51         30       39     61       92      45
## 2      63         64         51       54     63       73      47
## 3      71         70         68       69     76       86      48
## 4      61         63         45       47     54       84      35
## 5      81         78         56       66     71       83      47
## 6      43         55         49       44     54       49      34
```

```
## 7      58         67         42         56     66       68       35
## 8      71         75         50         55     70       66       41
## 9      72         82         72         67     71       83       31
## 10     67         61         45         47     62       80       41
## 11     64         53         53         58     58       67       34
## 12     67         60         47         39     59       74       41
## 13     69         62         57         42     55       63       25
## 14     68         83         83         45     59       77       35
## 15     77         77         54         72     79       77       46
## 16     81         90         50         72     60       54       36
## 17     74         85         64         69     79       79       63
## 18     65         60         65         75     55       80       60
## 19     65         70         46         57     75       85       46
## 20     50         58         68         54     64       78       52
## 21     50         40         33         34     43       64       33
## 22     64         61         52         62     66       80       41
## 23     53         66         52         50     63       80       37
## 24     40         37         42         58     50       57       49
## 25     63         54         42         48     66       75       33
## 26     66         77         66         63     88       76       72
## 27     78         75         58         74     80       78       49
## 28     48         57         44         45     51       83       38
## 29     85         85         71         71     77       74       55
## 30     82         82         39         59     64       78       39
```

```
# which models were successful?
models[ok]
```

```
## $result
## $result[[1]]
##
## Call:  glm(formula = Petal.Length ~ Petal.Width + Sepal.Length * Sepal.Width,
##     data = df)
##
## Coefficients:
##              (Intercept)               Petal.Width              Sepal.Length
##                  0.71482                   1.43584                   0.56175
##              Sepal.Width  Sepal.Length:Sepal.Width
##                 -0.97041                   0.05642
##
## Degrees of Freedom: 149 Total (i.e. Null);  145 Residual
## Null Deviance:       464.3
## Residual Deviance: 14.81     AIC: 90.4
##
## $result[[2]]
## NULL
##
## $result[[3]]
## NULL
```

11.2.2 Caching computations with memoise::memoise()

```
slow_function <- function(x) {
  Sys.sleep(1)
  x * 10 * runif(1)
}
system.time(print(slow_function(1)))
```

```
## [1] 8.588515

##    user  system elapsed
##    0.00    0.00    1.03
```

```
system.time(print(slow_function(1)))
```

```
## [1] 1.909614

##    user  system elapsed
##    0.00    0.00    1.02
```

```
fast_function <- memoise::memoise(slow_function)
system.time(print(fast_function(1)))
```

```
## [1] 1.777706

##    user  system elapsed
##    0.00    0.00    1.03
```

```
system.time(print(fast_function(1)))
```

```
## [1] 1.777706

##    user  system elapsed
##    0.01    0.00    0.01
```

```
fib <- function(n) {
  if (n < 2) return(1)
  fib(n - 2) + fib(n - 1)
}
system.time(fib(23))
```

```
##    user  system elapsed
##    0.05    0.00    0.05
```

```
system.time(fib(24))
```

```
##    user  system elapsed
##    0.07    0.00    0.08
```

```
fib2 <- memoise::memoise(function(n) {
  if (n < 2) return(1)
  fib2(n - 2) + fib2(n - 1)
})
system.time(fib2(23))
```

```
##    user  system elapsed
##       0       0       0
```

```
system.time(fib2(24))
```

```
##    user  system elapsed
##       0       0       0
```

11.2.3 Exercises

1. Base R provides a function operator in the form of `Vectorize()`. What does it do? When might you use it?

```
f <- function(x = 1:3, y) c(x, y)
vf <- Vectorize(f, SIMPLIFY = FALSE)
f(1:3, 1:3)
```

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

```
vf(1:3, 1:3)
```

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

```
vf(y = 1:3)
```

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

`Vectorize` creates a function wrapper that vectorizes the action of its argument `FUN`. I probably won't use it unless I see a better example

2. Read the source code for `possibly()`. How does it work?

```
x <- list(
  c(0.512, 0.165, 0.717),
  c(0.064, 0.781, 0.427),
  c(0.890, 0.785, 0.495),
  "oops"
)
possibly_sum <- possibly(sum, otherwise = "You Dumb")
out <- map(x, possibly_sum)
str(out)
```

```
## List of 4
##  $ : num 1.39
##  $ : num 1.27
##  $ : num 2.17
##  $ : chr "You Dumb"
```

Create a modified version of `.f` that return a default value (`otherwise`) whenever an error occurs.

3. Read the source code for `safely()`. How does it work?

```
safely_sum <- safely(sum)
out <- map(x, safely_sum)
str(out)
```

```
## List of 4
##  $ :List of 2
##   ..$ result: num 1.39
##   ..$ error : NULL
##  $ :List of 2
##   ..$ result: num 1.27
##   ..$ error : NULL
##  $ :List of 2
##   ..$ result: num 2.17
##   ..$ error : NULL
##  $ :List of 2
##   ..$ result: NULL
##   ..$ error :List of 2
##   .. ..$ message: chr "invalid 'type' (character) of argument"
##   .. ..$ call   : language .Primitive("sum")(..., na.rm = na.rm)
##   .. ..- attr(*, "class")= chr [1:3] "simpleError" "error" "condition"
```

Creates a modified version of .f that always succeeds. It returns a list with components result and error. If the function succeeds, result contains the returned value and error is NULL. If an error occurred, error is an error object and result is either NULL or otherwise.

11.3 Case study: Creating your own function operators

```
urls <- c(
  "adv-r" = "https://adv-r.hadley.nz",
  "r4ds" = "http://r4ds.had.co.nz/"
  # and many many more
)
path <- paste(tempdir(), names(urls), ".html")

walk2(urls, path, download.file, quiet = TRUE)
```

```
paths <- path
for(i in seq_along(urls)) {
  Sys.sleep(0.1)
  if (i %% 10 == 0) cat(".")
  download.file(urls[[i]], paths[[i]])
}
```

```
delay_by <- function(f, amount) {
  force(f)
  force(amount)

  function(...) {
    Sys.sleep(amount)
    f(...)
  }
}
system.time(runif(100))
```

```
##    user  system elapsed
##       0       0       0
```

```
system.time(delay_by(runif, 0.1)(100))
```

```
##    user  system elapsed
##    0.00    0.00    0.11
```

```
walk2(urls, path, delay_by(download.file, 0.1), quiet = TRUE)
```

```
dot_every <- function(f, n) {
  force(f)
  force(n)

  i <- 0
  function(...) {
    i <<- i + 1
    if (i %% n == 0) cat(".")
    f(...)
  }
}
walk(1:100, runif)
walk(1:100, dot_every(runif, 10))
```

```
## ..........
```

```
walk2(
  urls, path,
  dot_every(delay_by(download.file, 0.1), 10),
  quiet = TRUE
)
```

```
walk2(
  urls,
  path,
  download.file %>% dot_every(10) %>% delay_by(0.1),
  quiet = TRUE
)
```

11.3.1 Exercises

1. Weigh the pros and cons of `download.file %>% dot_every(10) %>% delay_by(0.1)` versus `download.file %>% delay_by(0.1) %>% dot_every(10)`.

```
walk2(
  urls,
  path,
  download.file %>% dot_every(10) %>% delay_by(0.1),
  quiet = TRUE
)

walk2(
  urls,
  path,
  download.file %>% delay_by(0.1) %>% dot_every(10),
  quiet = TRUE
)
```

Dot when the 10th download starts vs dot when the 9th download finishs

2. Should you memoise `file.download()`? Why or why not?

I wouldn't. Why store large files in memory. If the file changes you'd messed up your analysis

3. Create a function operator that reports whenever a file is created or deleted in the working directory, using `dir()` and `setdiff()`. What other global function effects might you want to track?

```
dir_checker <- function(past, present){
  added <- setdiff(present,past)
  removed <- setdiff(past,present)
  if(length(added == 0 & removed == 0)){
    return()
  }
  if(length(added) > 0) cat(paste(added, "was added\n"))
  if(length(removed) > 0) cat(paste(removed, "was removed\n"))
```

```
}

dir_tracker <- function(f){
  function(...){
    on.exit(dir_checker(past, dir()), add = T)
    past <- dir()
    f(...)
  }
}

#Download Hadley Books
urls <- c(
  "adv-r" = "https://adv-r.hadley.nz",
  "r4ds" = "http://r4ds.had.co.nz/"
  # and many many more
)
paths <- paste0(names(urls), ".html")

book_get <- function(urls, paths){
  walk2(urls, paths, download.file, quiet = TRUE)
}

book_remove <- function(paths){
  walk(paths, file.remove)
}

# Get books
get_books <- dir_tracker(book_get)
get_books(urls,paths)
```

```
## adv-r.html was added
##  r4ds.html was added
```

```
# Remove books
remove_books <- dir_tracker(book_remove)
remove_books(paths)
```

```
## adv-r.html was removed
##  r4ds.html was removed
```

Know when the working directory changes. When functions are masked by other functions. If default plotting arguments have been changed.

4. Write a function operator that logs a timestamp and message to a file every time a function is run.

```
function_tracker <- function(f, log.file){
  if(file.exists(log.file)) file.remove(log.file)
  i <- 0
  function(...){
    i <<- i + 1
    cat(paste0("Function has been run ", i, " time(s) as of ", date(), "\n"), file = log.file, append =
```

```
    f(...)
  }
}

new_mean <- function_tracker(mean, "mean.log")

new_mean(x[[1]])
```

```
## [1] 0.4646667
```

```
readLines("mean.log")
```

```
## [1] "Function has been run 1 time(s) as of Tue Feb 21 14:04:31 2023"
```

```
new_mean(x[[2]])
```

```
## [1] 0.424
```

```
new_mean(x[[3]])
```

```
## [1] 0.7233333
```

```
readLines("mean.log")
```

```
## [1] "Function has been run 1 time(s) as of Tue Feb 21 14:04:31 2023"
## [2] "Function has been run 2 time(s) as of Tue Feb 21 14:04:31 2023"
## [3] "Function has been run 3 time(s) as of Tue Feb 21 14:04:31 2023"
```

5. Modify `delay_by()` so that instead of delaying by a fixed amount of time, it ensures that a certain amount of time has elapsed since the function was last called. That is, if you called `g <- delay_by(1, f); g(); Sys.sleep(2); g()` there shouldn't be an extra delay.

```
delay_by <- function(f, amount) {
  force(f)
  force(amount)
  timer <- NULL
  function(...) {
    if (!is.null(timer)) {
      if (Sys.time() - timer < amount) {
        Sys.sleep(amount - (Sys.time() - timer))
      }
    }
    on.exit(timer <<- Sys.time())
    f(...)
  }
}

timed_random <- delay_by(runif, 5)
Sys.time()
```

```
## [1] "2023-02-21 14:04:31 PST"
```

```
system.time(timed_random(10))
```

```
##    user  system elapsed
##       0       0       0
```

```
system.time(timed_random(10))
```

```
##    user  system elapsed
##    0.00    0.00    5.06
```

```
Sys.sleep(11)
system.time(timed_random(10))
```

```
##    user  system elapsed
##       0       0       0
```