Chapter 11: Function Operators

Tony ElHabr

R4DS Reading Group

What are function operators (FO)?

Chapter 9 is about functionals. Chapter 10 is about function factories. What makes function operators different?

Term	Required Input	Optional Input	Output
Functionals	Function	Vector	Vector
Function Factory		Vector, Function	Function
Function Operator	Function	Vector	Function

FOs are probably best studied by thinking about how they operate on functions

- Behavioral FO: Changes the behavior of a function, e.g. logging, running a function only when necessary
  - `memoise::memoise()`
- Output FO: Manipulates the output of a function
  - `purrr::possibly()`, `purrr::safely()`, `purrr::quietly()`
- Input FO: Maniuplates the input of a function
  - `purrr::partial()`

Behavioral FO Example #1

Passing in only function `f`

```
slowly <- function(f){
  force(f)
  function(n, ...){
    cat(
      glue::glue('Sleeping for {n} seconds.'),
      sep = '\n'
    )
    Sys.sleep(n)
    # Need to do this to prevent extra printing.
    res <- utils::capture.output(f(n, ...))
  }
}

purrr::walk(
  seq(0.4, 0.2, by = -0.2),
  slowly(cat)
)
```

```
## Sleeping for 0.4 seconds.
## Sleeping for 0.2 seconds.
```

https://gist.github.com/ColinFay/d32cf4c9c5fb8d849f12a4e98d6c0549

Behavior FO Example #1

Now with an additional input, vector `n`

```
slowly <- function(f, n){
  force(f)
  force(n)
  function(...){
    stopifnot(is.numeric(n))
    cat(
      glue::glue('Sleeping for {n} seconds.'),
      sep = '\n'
    )
    Sys.sleep(n)
    f(...)
  }
}

slowly_cat <- slowly(cat, 0.5)
slowly_cat('hello world')
```

```
## Sleeping for 0.5 seconds.
## hello world
```

Behavioral FO Example #2


```
twice <- function(f){
  force(f)
  function(...){
    f(...)
    f(...)
  }
}

purrr::walk(
  c('hello', 'world'),
  twice(cat),
  sep = '\n' # Passed to `cat()` via `...`
)
```

```
## hello
## hello
## world
## world
```

Inspiration: https://realpython.com/primer-on-python-decorators/

Behavioral FO Example #2

With python

```
def do_twice(f):
    def wrapper(*args, **kwargs):
        f(*args, **kwargs)
        f(*args, **kwargs)
    return wrapper

@do_twice
def say(x):
    print(x)

list(map(say, ['hello', 'world']))
```

```
## hello
## hello
## world
## world
## [None, None]
```

Behavioral FO Example #3


```
download_beers <- function(name, verbose = TRUE) {
  base_url <- 'https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/202
  url <- glue::glue('{base_url}{name}.csv')
  if(verbose) {
    cat(glue::glue('Downloading {name}.csv'), sep = '\n')
  }
  readr::read_csv(url)
}
```

Using `memoise::memoise()` for caching

```
## # A tibble: 2 x 3
##   expression                                                  min   median
##   <bch:expr>                                                <bch:tm> <bch:tm>
## 1 download_beers("brewer_size", verbose = FALSE)             81.7ms   88.2ms
## 2 download_beers_quickly("brewer_size", verbose = FALSE)   170.2us  181.9us
```

Behavioral FO Example #4

Testing the speed of `memoise::memoise()`

```
# Forgive the contrived function.
slow_function <- function(x) {
  Sys.sleep(0.2)
  x * runif(1)
}
fast_function <- memoise::memoise(slow_function)

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

```
##    user  system elapsed
##     0.0     0.0     0.2
```

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

```
##    user  system elapsed
##    0.00    0.00    0.21
```

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

```
##    user  system elapsed
##    0.00    0.00    0.21
```

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

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

Behavioral FO Example #4

Even if you've changed the inputs since the most recent call, it will still be fast.

```
system.time(fast_function(2))
```

```
##    user  system elapsed
##     0.0     0.0     0.2
```

```
system.time(fast_function(3))
```

```
##    user  system elapsed
##    0.00    0.00    0.21
```

```
system.time(fast_function(2))
```

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

In fact, it remembers everything from the same session (assuming you haven't used `memoise::forget()`).

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

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

```
system.time(fast_function(2))
```

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

```
system.time(fast_function(3))
```

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

Behavioral FO Example #5

How does memoisization work with a factory?

```
fast_function_factory <- function(x) {
    memoise::memoise(function(...) {
    slow_function(...)
  })
}
```

```
system.time(fast_function(42))
```

```
##    user  system elapsed
##     0.0     0.0     0.2
```

```
system.time(fast_function(42))
```

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

```
system.time(fast_function_factory(-1))
```

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

```
system.time(fast_function_factory(-1))
```

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

Input FO Example #1

```
stat_robust <- function(f, ...) {
  function(...) {
    f(..., na.rm = TRUE)
  }
}
mean_robust <- stat_robust(mean)
min_robust <- stat_robust(min)
quantile_robust <- stat_robust(quantile)
```

```
x1 <- 1L:10L
mean_robust(x1)
```

```
## [1] 5.5
```

```
min_robust(x1)
```

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

```
quantile_robust(x1, 0.25)
```

```
##  25%
## 3.25
```

```
x2 <- x1; x2[1] <- NA
mean_robust(x2)
```

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

```
min_robust(x2)
```

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

```
quantile_robust(x2, 0.25)
```

```
## 25%
##   4
```

Input FO Example #1

Using `purrr::partial()`

```
mean_partial <- partial(mean, na.rm = TRUE)
min_partial <- partial(min, na.rm = TRUE)
quantile_partial <- partial(quantile, na.rm = TRUE, ... = )
```

Without `purrr::partial()`

```
mean_wrapper <- function(...) {
  mean(..., na.rm = TRUE)
}
```

Input FO Example #2

Using the `brewer_size` data set

```
brewer_size %>%
  summarize_at(
    vars(total_barrels, total_shipped),
    list(mean = mean, mean_robust = mean_robust)
  ) %>%
  mutate_all(
    ~scales::number(., scale = 1e-3, big.mark = ',', suffix = ' k')
  ) %>%
  glimpse()
```

```
## Rows: 1
## Columns: 4
## $ total_barrels_mean        <chr> NA
## $ total_shipped_mean        <chr> NA
## $ total_barrels_mean_robust <chr> "30,796 k"
## $ total_shipped_mean_robust <chr> "885 k"
```

Output FO Example #1

Using `purrr::safely()`

```
download_beers_safely <- purrr::safely(download_beers)

brewing_material <- download_beers_safely('brewing_material') # Oops!
```

```
## Downloading brewing_material.csv
```

```
brewing_material
```

```
## $result
## NULL
##
## $error
## <simpleError in open.connection(con, "rb"): HTTP error 404.>
```

```
brewing_materials <- download_beers_safely('brewing_materials') # Good
```

```
## Downloading brewing_materials.csv
```

```
brewing_materials$result %>% head(5)
```

```
## # A tibble: 5 x 9
##   data_type   material_type  year month type   month_current month_prior_year ytd_current ytd_pric
##   <chr>       <chr>         <dbl> <dbl> <chr>          <dbl>            <dbl>       <dbl>
## 1 Pounds of ~ Grain Produc~  2008     1 Malt ~     374165152        365300134   374165152      365
## 2 Pounds of ~ Grain Produc~  2008     1 Corn ~      57563519         41647092    57563519       41
## 3 Pounds of ~ Grain Produc~  2008     1 Rice ~      72402143         81050102    72402143       81
## 4 Pounds of ~ Grain Produc~  2008     1 Barle~       3800844          2362162     3800844        2
## 5 Pounds of ~ Grain Produc~  2008     1 Wheat~       1177186          1195381     1177186        1
```

Output FO Example #2

Using `purrr::possibly()`

```
download_beers_possibly <- purrr::possibly(download_beers, otherwise = tibble())

brewing_material <- download_beers_possibly('brewing_material') # Oops!
```

```
## Downloading brewing_material.csv
```

```
brewing_material
```

```
## # A tibble: 0 x 0
```

Output FO Example #3

Using `purrr::quietly()`

```
download_beers_quietly <- purrr::quietly(download_beers)

brewing_materials <- download_beers_quietly('brewing_materials') # Oops!
names(brewing_materials)
```

```
## [1] "result"   "output"   "warnings" "messages"
```

```
brewing_materials$result %>% head(5)
```

```
## # A tibble: 5 x 9
##   data_type   material_type  year month type   month_current month_prior_year ytd_current ytd_pric
##   <chr>       <chr>         <dbl> <dbl> <chr>          <dbl>            <dbl>       <dbl>
## 1 Pounds of ~ Grain Produc~  2008     1 Malt ~     374165152        365300134   374165152      365
## 2 Pounds of ~ Grain Produc~  2008     1 Corn ~      57563519         41647092    57563519       41
## 3 Pounds of ~ Grain Produc~  2008     1 Rice ~      72402143         81050102    72402143       81
## 4 Pounds of ~ Grain Produc~  2008     1 Barle~       3800844          2362162     3800844        2
## 5 Pounds of ~ Grain Produc~  2008     1 Wheat~       1177186          1195381     1177186        1
```

Combining FOs Example


```
nms <- c('woops', 'brewing_materials', 'beer_taxed', 'brewer_size', 'beer_states') %>%
  setNames(., .)

download_beers_nicely <- slowly(download_beers_safely, 0.1)
beers <- nms %>%
  map(.,
      ~download_beers_nicely(..1) %>%
        purrr::pluck('result')
    )
```

```
## Sleeping for 0.1 seconds.
## Downloading woops.csv
## Sleeping for 0.1 seconds.
## Downloading brewing_materials.csv
## Sleeping for 0.1 seconds.
## Downloading beer_taxed.csv
## Sleeping for 0.1 seconds.
## Downloading brewer_size.csv
## Sleeping for 0.1 seconds.
## Downloading beer_states.csv
```

```
beers %>% map(dim) %>% str()
```

```
## List of 5
##  $ woops            : NULL
##  $ brewing_materials: int [1:2] 1440 9
##  $ beer_taxed       : int [1:2] 1580 10
##  $ brewer_size      : int [1:2] 137 6
##  $ beer_states      : int [1:2] 1872 4
```

Combining FOs Example

And a real-world use-case for `purrr::reduce()`!

```
beers %>%
  purrr::discard(is.null) %>%
  purrr::reduce(dplyr::left_join) %>%
  dim()
```

```
## [1] 15984    18
```

FOs in the Wild

- `{scales}` and `{ggplot2}`'s `scale_(color|fill)_*()`
- `{glue}` with it's transformers
- Sparingly in `{styler}` and `{lintr}`
- `{plumber}` uses R6

FIN