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()`

`purrr::safely()` docs: ... "They are all adverbs because they modify the action of a verb (a function)."

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(...)
  }
}

purrr::walk(
  c('hello', 'world'),
  slowly(cat, 0.1),
  sep = '\n' # Passed to `f()` via `...`
)
```

```
## Sleeping for 0.1 seconds.
## hello
## Sleeping for 0.1 seconds.
## world
```

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

Behavioral FO Example #2

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

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

```
download_beers_quickly <- memoise::memoise(download_beers)

bench::mark(
  download_beers('brewer_size', verbose = FALSE),
  download_beers_quickly('brewer_size', verbose = FALSE)
) %>%
  dplyr::select(expression, min, median)
```

```
## # A tibble: 2 x 3
##   expression                                                min   median
##   <bch:expr>                                              <bch:tm> <bch:tm>
## 1 download_beers("brewer_size", verbose = FALSE)           77.8ms   89.6ms
## 2 download_beers_quickly("brewer_size", verbose = FALSE)    162us  188.5us
```

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
##    0.00    0.00    0.36
```

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

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

```
system.time(fast_function(11))
```

```
##    user  system elapsed
##    0.00    0.00    0.34
```

```
system.time(fast_function(11))
```

```
##    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(22))
```

```
##    user  system elapsed
##     0.0     0.0     0.3
```

```
system.time(fast_function(33))
```

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

```
system.time(fast_function(22))
```

```
##    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(11))
```

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

```
system.time(fast_function(22))
```

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

```
system.time(fast_function(33))
```

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

Input FO Example #1

Setting `na.rm = TRUE`

```
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()` to set `na.rm = TRUE`

```
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 mon
##   <chr>                    <chr>          <dbl> <dbl> <chr>                             <dbl>
## 1 Pounds of Materials Used Grain Products  2008     1 Malt and malt products        374165152
## 2 Pounds of Materials Used Grain Products  2008     1 Corn and corn products         57563519
## 3 Pounds of Materials Used Grain Products  2008     1 Rice and rice products         72402143
## 4 Pounds of Materials Used Grain Products  2008     1 Barley and barley products      3800844
## 5 Pounds of Materials Used Grain Products  2008     1 Wheat and wheat products        1177186
```

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')
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 mon
##   <chr>                    <chr>          <dbl> <dbl> <chr>                               <dbl>
## 1 Pounds of Materials Used Grain Products  2008     1 Malt and malt products          374165152
## 2 Pounds of Materials Used Grain Products  2008     1 Corn and corn products           57563519
## 3 Pounds of Materials Used Grain Products  2008     1 Rice and rice products           72402143
## 4 Pounds of Materials Used Grain Products  2008     1 Barley and barley products        3800844
## 5 Pounds of Materials Used Grain Products  2008     1 Wheat and wheat products          1177186
```

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