Introduction to functional programming in R

Cecile Murray, Data Scientist, ERD BDS

10/27/2021

Goals

- What is functional programming and why is it useful?
- Describe the components of a function
- Explore how to use functions to reduce repetition via demo

What is functional programming?

"When using a functional style, you strive to decompose components of the problem into isolated functions that operate independently. Each function taken by itself is simple and straightforward to understand; complexity is handled by composing functions in various ways."

- Hadley Wickham, Advanced R

How is functional programming different?

- Procedural programming = repeatedly writing and executing sequential commands
- Functional programming = writing functions to define a set of operations, then repeatedly executing the functions

Functional principles apply outside of R

- R
- Python
- Stata (macros)
- SAS (macros)
- many, many more

What is a function?

A function is a canned bit of code that takes in some inputs, performs some operations, and spits out some outputs.

Example: `mean()`

```
digits <- rnorm(10)
digits
```

```
##  [1]  0.73740119 -0.77795246 -0.26194502  0.02726521 -0.05524863 -0.
##  [7]  0.82919066 -1.44472613 -0.87465311  0.32728651
```

```
mean(digits)
```

```
## [1] -0.1857003
```

Example: procedural code

```
df <- tibble::tibble(
  a = rnorm(10),
  b = rnorm(10),
  c = rnorm(10),
  d = rnorm(10)
)

# rescale from 0-1
df$a <- (df$a - min(df$a, na.rm = TRUE)) /
  (max(df$a, na.rm = TRUE) - min(df$a, na.rm = TRUE))
df$b <- (df$b - min(df$b, na.rm = TRUE)) /
  (max(df$b, na.rm = TRUE) - min(df$a, na.rm = TRUE))
df$c <- (df$c - min(df$c, na.rm = TRUE)) /
  (max(df$c, na.rm = TRUE) - min(df$c, na.rm = TRUE))
df$d <- (df$d - min(df$d, na.rm = TRUE)) /
  (max(df$d, na.rm = TRUE) - min(df$d, na.rm = TRUE))
```

Source: R for Data Science

Example: procedural code

```
df <- tibble::tibble(
  a = rnorm(10),
  b = rnorm(10),
  c = rnorm(10),
  d = rnorm(10)
)

# rescale from 0-1
df$a <- (df$a - min(df$a, na.rm = TRUE)) /
  (max(df$a, na.rm = TRUE) - min(df$a, na.rm = TRUE))
df$b <- (df$b - min(df$b, na.rm = TRUE)) /
  (max(df$b, na.rm = TRUE) - min(df$a, na.rm = TRUE)) # copy-paste bug
df$c <- (df$c - min(df$c, na.rm = TRUE)) /
  (max(df$c, na.rm = TRUE) - min(df$c, na.rm = TRUE))
df$d <- (df$d - min(df$d, na.rm = TRUE)) /
  (max(df$d, na.rm = TRUE) - min(df$d, na.rm = TRUE))
```

Source: R for Data Science

Example: functional code

```r
# define function that rescales values from 0-1
rescale <- function(x) {
  rng <- range(x, na.rm = TRUE)
  (x - rng[1]) / (rng[2] - rng[1])
}

# call the function on each column
df$a <- rescale(df$a)
df$b <- rescale(df$b)
df$c <- rescale(df$c)
df$d <- rescale(df$d)
```

Source: R for Data Science

Benefits of a functional approach

Functions can:

- Reduce repetition/duplication in code
- Enable testing to ensure code produces expected output
- Serve as modular building blocks that can be re-used elsewhere

DRY programming: "Don't repeat yourself"

Duplicated code is:

- Harder for other people to understand
- More likely to contain copy and paste or other "human" errors
- Harder to parse for errors in logic or unexpected side effects
- Harder to maintain over time
- Possibly slower to run (probably only relevant on very large datasets)

Functions can be tested

- Verify that your function produces expected outputs for given inputs
- Know with certainty that you will get correct output anywhere you use it
- Much harder and more labor-intensive to do this with procedural code

Reusability

- Once you write a function, you can use it again anywhere you want
- This also allows you to use functions as building blocks

Key elements of functions

- name
- inputs, also known as arguments
- output, often called return value
- body: where computation is defined

Key elements of functions

```
do_something <- function(input1, input2) {
  out <- input1 %>% some_operation(input2)
  return(out)
}
```

How do we avoid repetitive code?

```
# call the function on each column
df$a <- rescale(df$a)
df$b <- rescale(df$b)
df$c <- rescale(df$c)
df$d <- rescale(df$d)
```

Options:

- loops
- base R `apply()` functions
- `purrr::map()`

Demo

Some topics we didn't cover

Function topics:

- scope + return values vs. side effects
- lazy evaluation: arguments don't get evaluated until they're called
- arbitrary number of inputs using `...`

`purrr`

- `purrr::map2` for two inputs
- `purrr::pmap` for arbitrary number of inputs
- list-columns: a column where each cell is a list instead of a single value (good for simulations/modeling)

Thank you! Questions?

More detail:

- R4DS Ch. 19 https://r4ds.had.co.nz/functions.html
- R4DS Ch. 21 https://r4ds.had.co.nz/iteration.html
- https://purrr.tidyverse.org/index.html

Contact me: cecile.m.murray@census.gov