Functional Programming

Amy Tzu-Yu Chen | amy17519@gmail.com
R Ladies Los Angeles - Advanced R Book Club
August 10, 2019

About me

LA West R Users Group

https://www.meetup.com/Los-Angeles-R-Users-Group-Data-Science/

Next Session Leaders???

- OO Programming
  - August 24 and September 7
- Metaprogramming
  - September 21 and October 5
- Techniques
  - October 19 and November 2

Functional Programming in R

“To understand computations in R, two slogans are helpful:

- Everything that exists is an object.

- Everything that happens is a function call."

— John Chambers

Everything that exists is an object

```
> `+`
function (e1, e2)  .Primitive("+")
> `<-`
.Primitive("<-")
```

Everything that happens is a function call

```
> `<-`(x, 5)
> x
[1] 5
```

```
> `+`(1, 2)
[1] 3
```

Pure Functions

- output only depends on input
- no "side-effect"

It does not mean Impure Functions are bad

- Depends on environment
- Could have side effects

```
# Outputs depends of environment

> Sys.Date()

[1] "2019-08-04"

# Side effect only

> write.csv(iris, "iris.csv")
```

Function's Execution Environment

The enclosing environment of the manufactured function is an execution environment of the function factory.

My take: what happens in the functions, stays in the functions, except the output!

So we should use `<<-` with caution (10.2.4)

1st vs. 2nd Edition

purrr vs. base R's apply family

Well, you still need to understand lapply

What's different

Base R: (1st Edition)

```
> Reduce(merge, list(df1, df2, df3…))
```

Tidyverse: (2nd Edition)

```
> purrr::reduce(list(df1, df2, df3…), merge)

> list(df1, df2, df3) %>% reduce(merge) %>% filter(....) %>% arrange(...)
```

You can pipe it!

Why purrr?

If you do a little bit of stalking, you can find Hadley's answer at

https://stackoverflow.com/questions/45101045/why-use-purrrmap-instead-of-lapply

*** Come to LA West R Users's meetup on September 26 for a deep dive in `purrr`!!

https://www.meetup.com/Los-Angeles-R-Users-Group-Data-Science/

Ahem... I still lapply

- Code readability `> lapply(list, function(x) f(x))` vs. `> map(list, ~ f(.x))`
- It's nice to have %>%, but it's also good to separate code chunks that do different things
- Speed
- No need to load and understand library(rlang) and library(magrittr)
- No need to remember `map_*&#()`. Just `lapply()`
    - it's nice to know what your output type is, but it's also better that you always examine your output and make judgment on whether it's the right type

Personally, I use both purrr and apply family. It really depends on your audience and what you are trying to do

Note on safely() 11.2.1

- Do you really want `safely()`? Use with caution
- Interactive vs. Production

Bottomline

In \ Out	Vector	Function
Vector	Regular function	Function factory
Function	Functional	Function operator

Links to stuff we talked about at the bookclub

- library(future) https://github.com/HenrikBengtsson/future
- library(furrr) https://github.com/DavisVaughan/furrr
- My blogpost on building internal packages https://amy17519.me/post/2019/07/26/notes-on-making-internal-r-package-in-org-team/
- Try picking just fns you need from `purrr` and `rlang` if you don't want to deal with dependencies
- purrr meetup on September 26: https://www.meetup.com/Los-Angeles-R-Users-Group-Data-Science/events/262796755/