Functional Programming with Purrr

#brug

Saurav Ghosh
2019-03-27

What is Functional Programming

Wikipedia

“ -a style of building the structure and elements of computer programs - that treats computation as the evaluation of **mathematical functions** and avoids **changing-state** and **mutable data**. It is a **declarative programming** paradigm, which means programming is done with expressions or declarations instead of statements.”

A brief detour- Lambda calculus

$\lambda x.\, x$

Where the name after λ is the argument and the expression after the (.) is the body of the function. In programming languages, you can rewrite the above expression as:

```
F←function (x) {
    return(x)
}
```

Example:

$$f(x) = x^2$$

Which can be rewritten as:

$$x => x^2$$

Lambda contd...

Example 1

$$p = [1..100].filter((value) \Rightarrow \{return\ value\%2 == 0\})$$

Example 2

$$a = [1..50]$$
$$a.map((value) \Rightarrow \{return\ value * 2\})$$

Question

$\lambda x(\lambda y.\, x + y)$

$$const\ F = x \Rightarrow \big(y \Rightarrow \big(x + y\big)\big)$$

$$F\big(5\big)\big(10\big)$$

Summary

- immutability (thread-safe)
- explicit state management
- side effect programming through data transformation
- expressions vs statements
- higher level functions (function that takes data and a function as arguments to transform the data)

Purrr

Two types of vectors:

- Atomic
- List

Functions

```
library(tidyverse)
```

```
## Warning: package 'tidyr' was built under R version 3.5.3
```

```
mt←mtcars
mt %>% str
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

Normalize

Let us normalize displacement and horsepower

Let us use min-max normalization

```
mt$new_disp←(mt$disp-min(mt$disp))/(max(mt$disp)-min(mt$disp))
mt$new_hp←(mt$hp-min(mt$hp))/(max(mt$hp)-min(mt$hp))
summary(mt$new_hp)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.1572  0.2509  0.3346  0.4523  1.0000
```

```
summary(mt$new_disp)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.1240  0.3123  0.3982  0.6358  1.0000
```

Write functions

Let us rewrite the normalization step as a function

```
func_norm←function(x){
  (x-min(x))/(max(x)-min(x))
}
my_vec←rnorm(15)
my_vec
```

```
##  [1]  1.41573096  0.39667507 -1.85107548 -0.57663227  1.62280307
##  [6]  0.21930478  0.59104896  1.84260397 -1.64519804  0.24566382
## [11] -0.06677544 -0.66745903 -1.06135475  1.38290861 -0.94481526
```

```
# normalized myvec
func_norm(my_vec)
```

```
##  [1] 0.88443150 0.60853969 0.00000000 0.34503352 0.94049270 0.56051974
##  [7] 0.66116307 1.00000000 0.05573777 0.56765600 0.48306846 0.32044374
## [13] 0.21380327 0.87554541 0.24535432
```

Put it all together

Let us apply the function in the dataset mt

```
mt←mt %>% mutate(norm_disp=func_norm(mt$disp))
mt←mt %>% mutate(norm_hp=func_norm(mt$hp))
```

```
summary(mt$norm_disp)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.1240  0.3123  0.3982  0.6358  1.0000
```

```
summary(mt$norm_hp)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.1572  0.2509  0.3346  0.4523  1.0000
```

Use for loop

```
mt_for←mtcars
for(i in seq_along(mtcars)){
  mt_for[i]←func_norm(mtcars[[i]])
}
summary(mt_for)
```

```
##       mpg              cyl              disp              hp        
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000  
##  1st Qu.:0.2138   1st Qu.:0.0000   1st Qu.:0.1240   1st Qu.:0.1572  
##  Median :0.3745   Median :0.5000   Median :0.3123   Median :0.2509  
##  Mean   :0.4124   Mean   :0.5469   Mean   :0.3982   Mean   :0.3346  
##  3rd Qu.:0.5277   3rd Qu.:1.0000   3rd Qu.:0.6358   3rd Qu.:0.4523  
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000  
##       drat              wt              qsec              vs        
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000  
##  1st Qu.:0.1475   1st Qu.:0.2731   1st Qu.:0.2848   1st Qu.:0.0000  
##  Median :0.4309   Median :0.4633   Median :0.3821   Median :0.0000  
##  Mean   :0.3855   Mean   :0.4358   Mean   :0.3987   Mean   :0.4375  
##  3rd Qu.:0.5346   3rd Qu.:0.5362   3rd Qu.:0.5238   3rd Qu.:1.0000  
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000  
##        am              gear             carb       
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000  
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.1429
```

Can we do better?

Using map

```
mt %>%
  map(function(x){
    func_norm(x)
  })
```

```
## $mpg
##  [1] 0.4510638 0.4510638 0.5276596 0.4680851 0.3531915 0.3276596 0.1659574
##  [8] 0.5957447 0.5276596 0.3744681 0.3148936 0.2553191 0.2936170 0.2042553
## [15] 0.0000000 0.0000000 0.1829787 0.9361702 0.8510638 1.0000000 0.4723404
## [22] 0.2170213 0.2042553 0.1234043 0.3744681 0.7191489 0.6638298 0.8510638
## [29] 0.2297872 0.3957447 0.1957447 0.4680851
##
## $cyl
##  [1] 0.5 0.5 0.0 0.5 1.0 0.5 1.0 0.0 0.0 0.5 0.5 1.0 1.0 1.0 1.0 1.0 1.0
## [18] 0.0 0.0 0.0 0.0 1.0 1.0 1.0 1.0 0.0 0.0 0.0 1.0 0.5 1.0 0.0
##
## $disp
##  [1] 0.22175106 0.22175106 0.09204290 0.46620105 0.72062859 0.38388626
##  [7] 0.72062859 0.18857570 0.17385882 0.24070841 0.24070841 0.51060115
## [13] 0.51060115 0.51060115 1.00000000 0.97006735 0.92017960 0.01895735
## [19] 0.01147418 0.00000000 0.12222499 0.61586431 0.58094288 0.69568471
```

Even better

Use anonymous function

```
mt %>%
  map(~func_norm(.))
```

```
## $mpg
##  [1] 0.4510638 0.4510638 0.5276596 0.4680851 0.3531915 0.3276596 0.1659574
##  [8] 0.5957447 0.5276596 0.3744681 0.3148936 0.2553191 0.2936170 0.2042553
## [15] 0.0000000 0.0000000 0.1829787 0.9361702 0.8510638 1.0000000 0.4723404
## [22] 0.2170213 0.2042553 0.1234043 0.3744681 0.7191489 0.6638298 0.8510638
## [29] 0.2297872 0.3957447 0.1957447 0.4680851
##
## $cyl
##  [1] 0.5 0.5 0.0 0.5 1.0 0.5 1.0 0.0 0.0 0.5 0.5 1.0 1.0 1.0 1.0 1.0 1.0
## [18] 0.0 0.0 0.0 0.0 1.0 1.0 1.0 1.0 0.0 0.0 0.0 1.0 0.5 1.0 0.0
##
## $disp
##  [1] 0.22175106 0.22175106 0.09204290 0.46620105 0.72062859 0.38388626
##  [7] 0.72062859 0.18857570 0.17385882 0.24070841 0.24070841 0.51060115
## [13] 0.51060115 0.51060115 1.00000000 0.97006735 0.92017960 0.01895735
## [19] 0.01147418 0.00000000 0.12222499 0.61586431 0.58094288 0.69568471
## [25] 0.82040409 0.01970566 0.12272387 0.05986530 0.69817910 0.18433525
## [31] 0.57345972 0.12446994
```

Return as data frame

```
mt %>%
  map_df(~func_norm(.))
```

```
## # A tibble: 32 x 15
##      mpg   cyl   disp     hp  drat    wt  qsec    vs    am  gear  carb
##    <dbl> <dbl>  <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 0.451   0.5 0.222  0.205  0.525 0.283 0.233     0     1   0.5 0.429
##  2 0.451   0.5 0.222  0.205  0.525 0.348 0.3       0     1   0.5 0.429
##  3 0.528   0   0.0920 0.145  0.502 0.206 0.489     1     1   0.5 0
##  4 0.468   0.5 0.466  0.205  0.147 0.435 0.588     1     0   0   0
##  5 0.353   1   0.721  0.435  0.180 0.493 0.3       0     0   0   0.143
##  6 0.328   0.5 0.384  0.187  0     0.498 0.681     1     0   0   0
##  7 0.166   1   0.721  0.682  0.207 0.526 0.160     0     0   0   0.429
##  8 0.596   0   0.189  0.0353 0.429 0.429 0.655     1     0   0.5 0.143
##  9 0.528   0   0.174  0.152  0.535 0.419 1         1     0   0.5 0.143
## 10 0.374   0.5 0.241  0.251  0.535 0.493 0.452     1     0   0.5 0.429
## # ... with 22 more rows, and 4 more variables: new_disp <dbl>,
## #   new_hp <dbl>, norm_disp <dbl>, norm_hp <dbl>
```

purrr::map(.x,.f,...)

map iterates over a list and returns a list.

- .x list (or vector) to iterate over
- .f function to apply over that list
- ... things that get passed from map() to .f

```
my_list=list(a=1:10,b=20:30)
map(my_list,~mean(.))
```

```
## $a
## [1] 5.5
##
## $b
## [1] 25
```

more maps

- map *list*
- map_lgl *logical*
- map_int *integer*
- map_dbl *double*
- map_chr *character*

```
# map_chr
map_chr(my_list,~mean(.))
```

```
##           a           b
##  "5.500000" "25.000000"
```

```
# map_dbl
map_dbl(my_list,~mean(.))
```

```
##    a    b
##  5.5 25.0
```

```
# map_lgl
# map_lgl(my_list,~mean(.))
# Error: Can't coerce element 1 from a double to a logical
```

map2

For two lists use map2

`map2(.x, .y, .f, ...)`

```
a←c(1,3,5,7,9)
b←c(2,4,6,8,10)
# map2
map2(.x=a,.y = b,~sum(.x,.y))
```

```
## [[1]]
## [1] 3
##
## [[2]]
## [1] 7
##
## [[3]]
## [1] 11
##
## [[4]]
## [1] 15
##
## [[5]]
```

pmap

For more, use pmap

`pmap(.l, .f, ...)`

```
a←c(1,2,3,4)
b←c(5,6,7,8)
c←c(4,3,2,1)
d←c(8,7,6,5)
pmap(list(a,b,c,d),~sum(a,b,c,d))
```

```
## [[1]]
## [1] 72
##
## [[2]]
## [1] 72
##
## [[3]]
## [1] 72
##
## [[4]]
## [1] 72
```

Many models with purrr

Gapminder data

```
library(gapminder)
library(broom)
gapminder %>%
  group_by(country) %>%
  nest() %>%
  mutate(fit = map(data, ~ lm(lifeExp ~ year, data = .x)))
```

```
## # A tibble: 142 x 3
##    country     data              fit
##    <fct>       <list>            <list>
##  1 Afghanistan <tibble [12 x 5]> <S3: lm>
##  2 Albania     <tibble [12 x 5]> <S3: lm>
##  3 Algeria     <tibble [12 x 5]> <S3: lm>
##  4 Angola      <tibble [12 x 5]> <S3: lm>
##  5 Argentina   <tibble [12 x 5]> <S3: lm>
##  6 Australia   <tibble [12 x 5]> <S3: lm>
##  7 Austria     <tibble [12 x 5]> <S3: lm>
##  8 Bahrain     <tibble [12 x 5]> <S3: lm>
##  9 Bangladesh  <tibble [12 x 5]> <S3: lm>
## 10 Belgium     <tibble [12 x 5]> <S3: lm>
## # ... with 132 more rows
```

Many models contd...

View model parameters using broom package

```
gp←gapminder %>%
  group_by(country) %>%
  nest() %>%
  mutate(fit = map(data, ~ lm(lifeExp ~ year, data = .x))) %>%
  mutate(tidied=map(fit,tidy)) %>%
  mutate(glanced=map(fit,glance)) %>%
  mutate(augmented=map(fit,augment))
gp
```

```
## # A tibble: 142 x 6
##    country    data          fit     tidied       glanced      augmented
##    <fct>      <list>        <list>  <list>       <list>       <list>
##  1 Afghanist~ <tibble [12 ~ <S3: l~ <tibble [2 ~ <tibble [1 ~ <tibble [12 ~
##  2 Albania    <tibble [12 ~ <S3: l~ <tibble [2 ~ <tibble [1 ~ <tibble [12 ~
##  3 Algeria    <tibble [12 ~ <S3: l~ <tibble [2 ~ <tibble [1 ~ <tibble [12 ~
##  4 Angola     <tibble [12 ~ <S3: l~ <tibble [2 ~ <tibble [1 ~ <tibble [12 ~
##  5 Argentina  <tibble [12 ~ <S3: l~ <tibble [2 ~ <tibble [1 ~ <tibble [12 ~
##  6 Australia  <tibble [12 ~ <S3: l~ <tibble [2 ~ <tibble [1 ~ <tibble [12 ~
##  7 Austria    <tibble [12 ~ <S3: l~ <tibble [2 ~ <tibble [1 ~ <tibble [12 ~
##  8 Bahrain    <tibble [12 ~ <S3: l~ <tibble [2 ~ <tibble [1 ~ <tibble [12 ~
##  9 Bangladesh <tibble [12 ~ <S3: l~ <tibble [2 ~ <tibble [1 ~ <tibble [12 ~
```

R-squared results

```
gapminder %>%
  group_by(country) %>%
  nest() %>%
  mutate(fit = map(data, ~ lm(lifeExp ~ year, data = .x))) %>%
  mutate(tidied=map(fit,tidy)) %>%
  mutate(glanced=map(fit,glance)) %>%
  mutate(augmented=map(fit,augment)) %>%
  mutate(rsq=map(glanced,~.[["r.squared"]])) %>%
  unnest(rsq) %>%
  arrange(desc(rsq)) %>%
  top_n(5)
```

```
## Selecting by rsq

## # A tibble: 5 x 7
##   country   data        fit     tidied      glanced      augmented      rsq
##   <fct>     <list>      <list>  <list>      <list>       <list>       <dbl>
## 1 Brazil    <tibble [1~ <S3: l~ <tibble [2~ <tibble [1 ~ <tibble [12~ 0.998
## 2 Mauritan~ <tibble [1~ <S3: l~ <tibble [2~ <tibble [1 ~ <tibble [12~ 0.998
## 3 France    <tibble [1~ <S3: l~ <tibble [2~ <tibble [1 ~ <tibble [12~ 0.998
## 4 Switzerl~ <tibble [1~ <S3: l~ <tibble [2~ <tibble [1 ~ <tibble [12~ 0.997
## 5 Pakistan  <tibble [1~ <S3: l~ <tibble [2~ <tibble [1 ~ <tibble [12~ 0.997
```

View Tidy results

Unnest the variable tidied

```
unnest(gp,tidied)
```

```
## # A tibble: 284 x 6
##    country     term         estimate std.error statistic  p.value
##    <fct>       <chr>           <dbl>     <dbl>     <dbl>    <dbl>
##  1 Afghanistan (Intercept)  -508.     40.5        -12.5  1.93e- 7
##  2 Afghanistan year            0.275   0.0205      13.5  9.84e- 8
##  3 Albania     (Intercept)  -594.     65.7         -9.05 3.94e- 6
##  4 Albania     year            0.335   0.0332      10.1  1.46e- 6
##  5 Algeria     (Intercept) -1068.     43.8        -24.4  3.07e-10
##  6 Algeria     year            0.569   0.0221      25.7  1.81e-10
##  7 Angola      (Intercept)  -377.     46.6         -8.08 1.08e- 5
##  8 Angola      year            0.209   0.0235       8.90 4.59e- 6
##  9 Argentina   (Intercept)  -390.      9.68       -40.3  2.14e-12
## 10 Argentina   year            0.232   0.00489     47.4  4.22e-13
## # ... with 274 more rows
```

View Glance results

Unnest the variable glanced

```
unnest(gp,glanced)
```

```
## # A tibble: 142 x 16
##    country data  fit   tidied augmented r.squared adj.r.squared sigma
##    <fct>   <lis> <lis> <list> <list>        <dbl>         <dbl> <dbl>
##  1 Afghan~ <tib~ <S3:~ <tibb~ <tibble ~     0.948         0.942 1.22
##  2 Albania <tib~ <S3:~ <tibb~ <tibble ~     0.911         0.902 1.98
##  3 Algeria <tib~ <S3:~ <tibb~ <tibble ~     0.985         0.984 1.32
##  4 Angola  <tib~ <S3:~ <tibb~ <tibble ~     0.888         0.877 1.41
##  5 Argent~ <tib~ <S3:~ <tibb~ <tibble ~     0.996         0.995 0.292
##  6 Austra~ <tib~ <S3:~ <tibb~ <tibble ~     0.980         0.978 0.621
##  7 Austria <tib~ <S3:~ <tibb~ <tibble ~     0.992         0.991 0.407
##  8 Bahrain <tib~ <S3:~ <tibb~ <tibble ~     0.967         0.963 1.64
##  9 Bangla~ <tib~ <S3:~ <tibb~ <tibble ~     0.989         0.988 0.977
## 10 Belgium <tib~ <S3:~ <tibb~ <tibble ~     0.995         0.994 0.293
## #  ... with 132 more rows, and 8 more variables: statistic <dbl>,
## #   p.value <dbl>, df <int>, logLik <dbl>, AIC <dbl>, BIC <dbl>,
## #   deviance <dbl>, df.residual <int>
```

View Augmented results

Unnest the variable augmented

```
unnest(gp,augmented)
```

```
## # A tibble: 1,704 x 10
##    country lifeExp  year .fitted .se.fit  .resid   .hat .sigma .cooksd
##    <fct>     <dbl> <int>   <dbl>   <dbl>   <dbl>  <dbl>  <dbl>   <dbl>
##  1 Afghan~    28.8  1952    29.9   0.664 -1.11   0.295    1.21 2.43e-1
##  2 Afghan~    30.3  1957    31.3   0.580 -0.952  0.225    1.24 1.13e-1
##  3 Afghan~    32.0  1962    32.7   0.503 -0.664  0.169    1.27 3.60e-2
##  4 Afghan~    34.0  1967    34.0   0.436 -0.0172 0.127    1.29 1.65e-5
##  5 Afghan~    36.1  1972    35.4   0.385  0.674  0.0991   1.27 1.85e-2
##  6 Afghan~    38.4  1977    36.8   0.357  1.65   0.0851   1.15 9.23e-2
##  7 Afghan~    39.9  1982    38.2   0.357  1.69   0.0851   1.15 9.67e-2
##  8 Afghan~    40.8  1987    39.5   0.385  1.28   0.0991   1.21 6.67e-2
##  9 Afghan~    41.7  1992    40.9   0.436  0.754  0.127    1.26 3.17e-2
## 10 Afghan~    41.8  1997    42.3   0.503 -0.534  0.169    1.27 2.33e-2
## # ... with 1,694 more rows, and 1 more variable: .std.resid <dbl>
```

Other map functions

- keep
- discard
- map_if
- every
- some

More details on Hooked on Data blog by Emily Robinson.

Thank you!

QnA