Introduction to Programming with purrr

INTERMEDIATE FUNCTIONAL PROGRAMMING WITH PURRR

Colin Fay
Data Scientist & R Hacker at ThinkR

datacamp

$whoami

Discovering purrr

- **R for Data Science**
  H. Wickham & G. Grolemund
- **purrr Tutorial**
  J. Bryan
- **A purrr tutorial - useR! 2017**
  C. Wickham
- **Happy dev with {purrr}**
  C. Fay
- **Foundations of Functional Programming with purrr**

What will this course cover?

```
map( .x, .f, ...)
```

for each element of `.x` do `.f`

> From: Charlotte Wickham — **A introduction to purrr**

Communauté d'Agglomération du Pays de Saint-Malo

Data.StMalo-Agglomération

Le portail des données publiques du territoire

Inscription
Connexion

ACCUEIL | DONNÉES | DÉMARCHE | LICENCE | API | COMMUNES | CARTOGRAPHIE | RÉUTILISATIONS

1 828 enregistrements

Aucun filtre actif.

Filtres

Rechercher...

Jour

2012	366
2013	365
2014	365
2015	365
2016	366

Fréquentation du site saint-malo.fr

Informations | Tableau | Analyse | Export | API

Statistiques quotidiennes du site web de la commune, disponible à l'adresse http://www.saint-malo.fr :

Les données disponibles sont :

- nombre de visites totales
- nombre de visiteurs uniques
- nombre de pages vues

Identifiant du jeu de données	`frequentation-du-site-saint-malofr`
Téléchargements	29
Thèmes	Administration, Gouvernement, Finances publiques, Citoyenneté
Mots clés	visite, web, internet

purrr basics - a refresher (Part 1)

`map(.x, .f, ...)`

- for each element of .x
- do .f(.x, ...)
- return a list

```
res <- map(visit_2015, sum)
class(res)
```

```
"list"
```

`map_dbl(.x, .f, ...)`

- for each element of .x
- do .f(.x, ...)
- return a numeric vector

```
res <- map_dbl(visit_2015, sum)
class(res)
```

```
"numeric"
```

purrr basics - a refresher (Part 2)

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

- for each element of .x and .y
- do .f(.x, .y, ...)
- return a list

```
res <- map2(visit_2015,
            visit_2016,
            sum)
class(res)
```

```
"list"
```

`map2_dbl(.x, .f, ...)`

- for each element of .x and .y
- do .f(.x, .y, ...)
- return a numeric vector

```
res <- map2_dbl(visit_2015,
                visit_2016,
                sum)
class(res)
```

```
"numeric"
```

purrr basics - a refresher (Part 3)

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

- for each sublist of .l
- do f(..1, ..2, ..3, [etc], ...)
- return a list

```
l <- list(visit_2014,
          visit_2015,
          visit_2016)
res  <- pmap(l, sum)
class(res)
```

```
"list"
```

`pmap_dbl(.l, .f, ...)`

- for each sublist of .l
- do f(..1, ..2, ..3, [etc], ...)
- return a numeric vector

```
l <- list(visit_2014,
          visit_2015,
          visit_2016)
res <- pmap_dbl(l, sum)
class(res)
```

```
"numeric"
```

Let's practice!

INTERMEDIATE FUNCTIONAL PROGRAMMING WITH PURRR

Introduction to mappers

INTERMEDIATE FUNCTIONAL PROGRAMMING WITH PURRR

Colin Fay
Data Scientist & R Hacker at ThinkR

.f in purrr

A function:

- for each elements of .x
- do .f(.x, ...)

A number n:

- for each elements of .x
- do .x[n]

A character vector z

- for each elements of .x
- do .x[z]

.f as a function

When a function, `.f` can be either:

- A classical function

```
my_fun <- function(x) {
  round(mean(x))
}
map_dbl(visit_2014, my_fun)
```

```
[1]  5526  6546  6097  7760
[5]  7025  7162 10484  8256
[9]  6558  7686  5723  5053
```

- A lambda (or anonymous) function

```
map_dbl(visit_2014, function(x) {
  round(mean(x))
})
```

```
[1]  5526  6546  6097  7760
[5]  7025  7162 10484  8256
[9]  6558  7686  5723  5053
```

Mappers: part 1

mapper: anonymous function with a one-sided formula

```
# With one parameter
map_dbl(visits2017, ~ round(mean(.x)))

# Is equivalent to
map_dbl(visits2017, ~ round(mean(.)))

# Is equivalent to
map_dbl(visits2017, ~ round(mean(..1)))
```

Mappers: part 2

mapper: anonymous function with a one-sided formula

```
# With two parameters
map2(visits2016, visits2017, ~ .x + .y)

# Is equivalent to
map2(visits2016, visits2017, ~ ..1 + ..2)
```

```
# With more than two parameters
pmap(list, ~ ..1 + ..2 + ..3)
```

as_mapper()

`as_mapper()` : create mapper objects from a lambda function

```
# Classical function
round_mean <- function(x){
    round(mean(x))
}
```

```
# As a mapper
round_mean <- as_mapper(~ round(mean(.x))))
```

Why mappers?

Mappers are:

- Concise
- Easy to read
- Reusable

Let's practice!

INTERMEDIATE FUNCTIONAL PROGRAMMING WITH PURRR

Using mappers to clean up your data

INTERMEDIATE FUNCTIONAL PROGRAMMING WITH PURRR

Colin Fay
Data Scientist & R Hacker at ThinkR

datacamp

Setting the name of your objects

`set_names()` : sets the names of an unnamed list

```
names(visits2016)
length(visits2016)
month.abb
```

```
NULL
12
"Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

```
visits2016 <- set_names(visits2016, month.abb)
names(visits2016)
```

```
"Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

Setting names with `map()`:

```
all_visits <- list(visits2015, visits2016, visits2017)
named_all_visits <- map(all_visits, ~ set_names(.x, month.abb))
names(named_all_visits[[1]])
```

```
"Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

```
names(named_all_visits[[2]])
```

```
"Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

```
names(named_all_visits[[3]])
```

```
"Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

keep()

`keep()` : extract elements that satisfy a condition

```
# Which month has received more than 30000 visits?
over_30000 <- keep(visits2016, ~ sum(.x) > 30000)
names(over_30000)
```

```
"Jan" "Mar" "Apr" "May" "Jul" "Aug" "Oct" "Nov"
```

```
limit <- as_mapper(~ sum(.x) > 30000)
# Which month has received more than 30000 visits?
over_mapper <- keep(visits2016, limit)
names(over_mapper)
```

```
"Jan" "Mar" "Apr" "May" "Jul" "Aug" "Oct" "Nov"
```

discard()

`discard()` : remove elements that satisfy a condition

```
# Which month has received less than 30000 visits?
under_30000 <- discard(visits2016, ~ sum(.x) > 30000)
names(under_30000)
```

```
"Feb" "Jun" "Sep" "Dec"
```

```
limit <- as_mapper(~ sum(.x) > 30000)
# Which month has received less than 30000 visits?
under_mapper <- discard(visits2016, limit)
names(under_mapper)
```

```
"Feb" "Jun" "Sep" "Dec"
```

keep(), discard(), and map()

Using `map()` & `keep()` :

```
df_list <- list(iris, airquality) %>% map(head)
map(df_list, ~ keep(.x, is.factor))
```

```
[[1]]
  Species
1  setosa
2  setosa
3  setosa
4  setosa
5  setosa
6  setosa

[[2]]
data frame with 0 columns and 6 rows
```

Let's practice!

INTERMEDIATE FUNCTIONAL PROGRAMMING WITH PURRR

Predicates

INTERMEDIATE FUNCTIONAL PROGRAMMING WITH PURRR

Colin Fay
Data Scientist & R Hacker at ThinkR

What is a predicate?

Predicates: return TRUE or FALSE

- Test for conditions
- Exist in base R: `is.numeric()` , `%in%` , `is.character()` , etc.

```
is.numeric(10)
```

```
TRUE
```

What is a predicate functional?

Predicate functionals:

- Take an element & a predicate
- Use the predicate on the element

```
keep(airquality, is.numeric)
```

`every()` : does *every* element satisfy a condition?

```
# Are all elements of visits2016 numeric?
every(visits2016, is.numeric)
# Is the mean of every months above 1000?
every(visits2016, ~ mean(.x) > 1000)
```

```
TRUE
FALSE
```

`some()` : do *some* elements satisfy a condition?

```
# Is the mean of some months above 1000?
some(visits2016, ~ mean(.x) > 1000)
```

```
TRUE
```

detect_index()

```
# Which is the first element with a mean above 1000?
detect_index(visits2016, ~ mean(.x) > 1000)
```

```
1
```

```
# Which is the last element with a mean above 1000?
detect_index(visits2016, ~ mean(.x) > 1000, .right = TRUE)
```

```
11
```

has_element() and detect()

```
# What is the value of the first element with a mean above 1000?
detect(visits2016, ~ mean(.x) > 1000, .right = TRUE)
```

```
[1] 1289  782 1432 1171 1094 1015  582  946 1191 1393 1307 1125 1267 1345 1066  810  583
[18] 733  795  766  873  656 1018  645  949  938 1118 1106 1134 1126
```

```
# Does one month has a mean of 981?
visits2016_mean <- map(visits2016, mean)
has_element(visits2016_mean,981)
```

```
TRUE
```

Let's practice!

INTERMEDIATE FUNCTIONAL PROGRAMMING WITH PURRR