

Foundations of Functional Programming with purrr_Troubleshooting lists with purrr

dizhen

5/16/2020

```
library(repurrrsive)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##   filter, lag

## The following objects are masked from 'package:base':
##
##   intersect, setdiff, setequal, union
```

```
library(ggplot2)
library(purrr)
```

How to purrr safely()

```
# Map safely over log
a <- list(-10, 1, 10, 0) %>%
  map(safely(log, otherwise = NA_real_)) %>%
  # Transpose the result
  transpose()
```

```
## Warning in .f(...): NaNs produced
```

```
# Print the list
a
```

```
## $result
## $result[[1]]
## [1] NaN
##
## $result[[2]]
```

```
## [1] 0
##
## $result[[3]]
## [1] 2.302585
##
## $result[[4]]
## [1] -Inf
##
##
## $error
## $error[[1]]
## NULL
##
## $error[[2]]
## NULL
##
## $error[[3]]
## NULL
##
## $error[[4]]
## NULL
```

```
# Print the result element in the list
a["result"]
```

```
## [[1]]
## [1] NaN
##
## [[2]]
## [1] 0
##
## [[3]]
## [1] 2.302585
##
## [[4]]
## [1] -Inf
```

```
# Print the error element in the list
a["error"]
```

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

```
# Map over sw_people and pull out the height element
height_ft <- map(sw_people , "height") %>%
  map(safely(function(x){
    x * 0.0328084
  }, quiet = FALSE)) %>%
  transpose()
```

3

[illegible]

```
# Print your list, the result element, and the error element
# height_ft
```

```
# height_ft[["result"]]
# height_ft[["error"]]
```

Another way to possibly() purrr

safely() outputs an NA in place of an error

possibly() will run through your code and implement your desired changes without printing out the error messages.

```
# Take the log of each element in the list
a <- list(-10, 1, 10, 0) %>%
  map(possibly(function(x){
    log(x)
  }, NA_real_))
```

```
## Warning in log(x): NaNs produced
```

```
# Create a piped workflow that returns double vectors
height_cm %>%
  map_dbl(possibly(function(x){
    # Convert centimeters to feet
    x * 0.0328084
  }, NA_real_))
```

```
## [1] 5.643045 5.479003 3.149606 6.627297 4.921260 5.839895 5.413386 3.182415
## [9] 6.003937 5.971129 6.167979 5.905512 7.480315 5.905512 5.675853 5.741470
## [17] 5.577428 5.905512 2.165354 5.577428 6.003937 6.561680 6.233596 5.807087
## [25] 5.741470 5.905512 4.921260 NA 2.887139 5.249344 6.332021 6.266404
## [33] 5.577428 6.430446 7.349082 6.758530 6.003937 4.494751 3.674541 6.003937
## [41] 5.347769 5.741470 5.905512 5.839895 3.083990 4.002625 5.347769 6.167979
## [49] 6.496063 6.430446 5.610236 6.036746 6.167979 8.661418 6.167979 6.430446
## [57] 6.069554 5.150919 6.003937 6.003937 5.577428 5.446194 5.413386 6.332021
## [65] 6.266404 6.003937 5.511811 6.496063 7.513124 6.988189 5.479003 2.591864
## [73] 3.149606 6.332021 6.266404 5.839895 7.086614 7.677166 6.167979 5.839895
## [81] 6.758530 NA NA NA NA NA 5.413386
```

purrr is a walk() in the park

If you need to print, using walk() prints out lists in a more compact and human-readable way, without all those brackets. walk() is also great for printing out plots without printing anything to the console.

```
# Print with walk
walk(people_by_film, print)
sw_films
```

One more use of walk(), specifically creating plots using walk()

```
# Load the gap_split data
data(gap_split)

# Map over the first 10 elements of gap_split
plots <- map2(gap_split[1:10],
              names(gap_split[1:10]),
              ~ ggplot(.x, aes(year, lifeExp)) +
                geom_line() +
                labs(title = .y))

# Object name, then function name
walk(plots, print)
```



















