Functional Programming

Data Wrangling in R

Functional Programming

Allows you to flexibly iterate functions to each element of a list or vector

- https://purrr.tidyverse.org/
- https://github.com/rstudio/cheatsheets/raw/master/purrr.pdf

Examples we will use

- https://jennybc.github.io/purrr-tutorial/
- https://cran.r-project.org/web/packages/repurrrsive/index.html
- https://tidyr.tidyverse.org/articles/rectangle.html

Why do this at all?

https://jennybc.github.io/purrr-tutorial/bk01_base-functions.html

You need a way to iterate in R in a data-structure-informed way. What does that mean?

- Iterate over elements of a list
- Iterate over rows or columns of a 2-dimensional object
- Iterate over sub data frames induced by one or more factors
- Iterate over tuples formed from the i-th element of several vectors of equal length

What is a 'list'?

- Lists are the most flexible/"generic" data class in R
- Can be created using list()
- Can hold vectors, strings, matrices, models, list of other list, lists upon lists!
- Can reference data using $ (if the elements are named), or using [], or [[]]

```
> mylist <- list(letters=c("A", "b", "c"),
+          numbers=1:3, matrix(1:25, ncol=5))
```

List Structure

```
> head(mylist)

$letters
[1] "A" "b" "c"

$numbers
[1] 1 2 3

[[3]]
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    8   13   18   23
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25
```

List referencing

```
> mylist[1] # returns a list

$letters
[1] "A" "b" "c"

> mylist["letters"] # returns a list

$letters
[1] "A" "b" "c"
```

List referencing

```
> mylist[[1]] # returns the vector 'letters'
[1] "A" "b" "c"
> mylist$letters # returns vector
[1] "A" "b" "c"
> mylist[["letters"]] # returns the vector 'letters'
[1] "A" "b" "c"
```

List referencing

You can also select multiple lists with the single brackets.

```
> mylist[1:2] # returns a list
```

```
$letters
[1] "A" "b" "c"

$numbers
[1] 1 2 3
```

List referencing

You can also select down several levels of a list at once

```
> mylist$letters[1]
```

```
[1] "A"
```

```
> mylist[[2]][1]
```

```
[1] 1
```

```
> mylist[[3]][1:2,1:2]
```

```
     [,1] [,2]
[1,]    1    6
[2,]    2    7
```

How would I encounter lists?

This comes up a lot in data cleaning (although many tasks can be accomplished by `separate`)

```
h = c("I like performing", "much data wrangling in R", "it is oh so fun")
ll = str_split(h, " ")
ll
```

```
## [[1]]
## [1] "I"          "like"       "performing"
##
## [[2]]
## [1] "much"      "data"      "wrangling" "in"        "R"
##
## [[3]]
## [1] "it"  "is"  "oh"  "so"  "fun"
```

Why do this at all?

```
class(got_chars)
```

```
## [1] "list"
```

```
length(got_chars)
```

```
## [1] 30
```

```
lengths(got_chars)
```

```
##  [1] 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 1
## [26] 18 18 18 18 18
```

```
names(got_chars[[1]])
```

```
##  [1] "url"         "id"          "name"        "gender"      "culture"
##  [6] "born"        "died"        "alive"       "titles"      "aliases"
## [11] "father"      "mother"      "spouse"      "allegiances" "books"
## [16] "povBooks"    "tvSeries"    "playedBy"
```

How would we get the names of each GoT character?

The really old way involved `for` loops:

```
char_names = vector("character", length(got_chars)) # initiate
for (i in seq(along=got_chars)) { # iterate
  char_names[i] = got_chars[[i]]$name
}
char_names[1:5] # examine
```

```
## [1] "Theon Greyjoy"     "Tyrion Lannister"  "Victarion Greyjoy"
## [4] "Will"              "Areo Hotah"
```

How would we get the names of each GoT character?

The kinda old way involved `apply` statements:

```
sapply(got_chars[1:5], function(x) x$name)
```

```
## [1] "Theon Greyjoy"     "Tyrion Lannister"  "Victarion Greyjoy"
## [4] "Will"              "Areo Hotah"
```

How would we get the names of each GoT character?

This can still be pretty efficient:

```
sapply(got_chars[1:5], "[[", "name")
```

```
## [1] "Theon Greyjoy"     "Tyrion Lannister"  "Victarion Greyjoy"
## [4] "Will"              "Areo Hotah"
```

The user interface of the “apply” functions is not as consistent as it could be, which slows down learning. The return objects frequently require further checking and massage to use downstream. In particular, there’s a tendency to return a vector (atomic or otherwise) or array, instead of data frame, with the original factor levels appearing in a names attribute.
[https://jennybc.github.io/purrr-tutorial/bk01_base-functions.html]

How would we get the names of each GoT character?

This can still be pretty efficient:

```
map_chr(got_chars[1:5], "name")
```

```
## [1] "Theon Greyjoy"     "Tyrion Lannister"  "Victarion Greyjoy"
## [4] "Will"              "Areo Hotah"
```

`sapply` versus `map_chr`

Potentially confusing output:

```
sapply(got_chars[2:3], "[[", "aliases")
```

```
## [[1]]
##  [1] "The Imp"            "Halfman"            "The boyman"
##  [4] "Giant of Lannister" "Lord Tywin's Doom"  "Lord Tywin's Bane"
##  [7] "Yollo"              "Hugor Hill"         "No-Nose"
## [10] "Freak"              "Dwarf"
##
## [[2]]
## [1] "The Iron Captain"
```

Compared to error:

```
map_chr(got_chars[2:3], "aliases")
```

```
## Error: Result 1 must be a single string, not a character vector of length 1
```

Example using string split output

```
map_chr(ll, first)
```

```
## [1] "I"    "much" "it"
```

```
map_chr(ll, last)
```

```
## [1] "performing" "R"           "fun"
```

```
map_chr(ll, nth, 3)
```

```
## [1] "performing" "wrangling"  "oh"
```

More extensive examples

You can create tibbles where each observation is a list:

https://tidyr.tidyverse.org/articles/rectangle.html

```
chars <- tibble(char = got_chars)
chars
```

```
## # A tibble: 30 x 1
##    char
##    <list>
##  1 <named list [18]>
##  2 <named list [18]>
##  3 <named list [18]>
##  4 <named list [18]>
##  5 <named list [18]>
##  6 <named list [18]>
##  7 <named list [18]>
##  8 <named list [18]>
##  9 <named list [18]>
## 10 <named list [18]>
## # … with 20 more rows
```

More extensive examples

```
chars2 <- chars %>% unnest_wider(char)
chars2
```

```
## # A tibble: 30 x 18
##    url      id name  gender culture born  died  alive titles aliases father
##    <chr> <int> <chr> <chr>  <chr>   <chr> <chr> <lgl> <list> <list>  <chr>
##  1 http…  1022 Theo… Male   "Ironb… "In … ""    TRUE  <chr … <chr [… ""
##  2 http…  1052 Tyri… Male   ""      "In … ""    TRUE  <chr … <chr [… ""
##  3 http…  1074 Vict… Male   "Ironb… "In … ""    TRUE  <chr … <chr [… ""
##  4 http…  1109 Will  Male   ""      ""    "In … FALSE <chr … <chr [… ""
##  5 http…  1166 Areo… Male   "Norvo… "In … ""    TRUE  <chr … <chr [… ""
##  6 http…  1267 Chett Male   ""      "At … "In … FALSE <chr … <chr [… ""
##  7 http…  1295 Cres… Male   ""      "In … "In … FALSE <chr … <chr [… ""
##  8 http…   130 Aria… Female "Dorni… "In … ""    TRUE  <chr … <chr [… ""
##  9 http…  1303 Daen… Female "Valyr… "In … ""    TRUE  <chr … <chr [… ""
## 10 http…  1319 Davo… Male   "Weste… "In … ""    TRUE  <chr … <chr [… ""
## # … with 20 more rows, and 7 more variables: mother <chr>, spouse <chr>,
## #   allegiances <list>, books <list>, povBooks <list>, tvSeries <list>,
## #   playedBy <list>
```

Say you wanted all characters and their titles:

```
chars2 %>%
  select(name, title = titles) %>%
  unnest_longer(title)
```

```
## # A tibble: 60 x 2
##    name              title
##    <chr>             <chr>
##  1 Theon Greyjoy     "Prince of Winterfell"
##  2 Theon Greyjoy     "Captain of Sea Bitch"
##  3 Theon Greyjoy     "Lord of the Iron Islands (by law of the green lands)"
##  4 Tyrion Lannister  "Acting Hand of the King (former)"
##  5 Tyrion Lannister  "Master of Coin (former)"
##  6 Victarion Greyjoy "Lord Captain of the Iron Fleet"
##  7 Victarion Greyjoy "Master of the Iron Victory"
##  8 Will              ""
##  9 Areo Hotah        "Captain of the Guard at Sunspear"
## 10 Chett             ""
## # … with 50 more rows
```