# Iteration - r4ds

Tomoya Fukumoto

2019-09-06

# Iteration

繰り返し作業をどうやって自動化するための二つの手法

1. ループ
2. 関数型プログラミング (functional programming)

# 準備

```
library(tidyverse)
```

ループに関わるのは base ライブラリ

FP に関わるのは purrr ライブラリ

# 21.2 For loops

最も標準的なループ

# 例：各行の median を求める（ループなし）

```
df <- tibble(
          a = rnorm(10),
          b = rnorm(10),
          c = rnorm(10),
          d = rnorm(10)
)
median(df$a)
median(df$b)
median(df$c)
median(df$d)
```

# 例：各行の median を求める（ループ）

```
output <- vector("double", ncol(df))  # 1. output
for (i in seq_along(df)) {            # 2. sequence
  output[[i]] <- median(df[[i]])      # 3. body
}
output
```

```
## [1]  0.6716290 -0.0496664  0.3351858  0.3524706
```

# ループの構成要素 output

```
output <- vector("double", ncol(df))
```

- ループの出力の器
- ループが始まる前に作る
- vector 関数で型と長さを指定する
- 長さを指定せずとも処理が遅くなる

# ループの構成要素 sequence

```
i in seq_along(df)
```

- どうループを回すか
- 一周するたびに `i` がベクトル `seq_along(df)` の中で値を変化させる
- (`seq_along(df)` は `1:length(df)` とほぼ同じ)
  - `length(df)` が `0` のときだけ違う

# ループの構成要素 body

```
output[[i]] <- median(df[[i]])
```

- ループで実際に処理する内容
- 1 回目は `output[[1]] <- median(df[[1]])`
- 2 回目は `output[[2]] <- median(df[[2]])`

## 21.2.1 Exercises

1 を見てみる

# 21.3 For loop variations

1. オブジェクトを修正するループ（作成するのではなく）
2. 要素の名前や値で回すループ（インデックスではなく）
3. 出力の長さが不明の場合
4. ループ回数が不明の場合

## 21.3.1 Modifying an existing object

```
df <- tibble(a = rnorm(10), b = rnorm(10), c = rnorm(10), d
rescale01 <- function(x) {
    rng <- range(x, na.rm = TRUE)
  (x - rng[1]) / (rng[2] - rng[1])
}
df$a <- rescale01(df$a)
df$b <- rescale01(df$b)
df$c <- rescale01(df$c)
df$d <- rescale01(df$d)
```

等価

```
for (i in seq_along(df)) {
    df[[i]] <- rescale01(df[[i]])
}
```

# 21.3.2 Looping patterns

ループを回すときの sequence の作法

1. `for (i in seq_along(xs))` インデックスとして 1 から順に数え上げる
   - 最も一般的かつ汎用的
2. `for (x in xs)` ベクトル `xs` の要素を一つずつとる
   - 出力を保存しにくい
     - オブジェクト操作でなく副作用 (plot, write など) に使う
3. `for (nm in names(xs))` 要素の名前を一つずつとる
   - 要素の名前を使いたい場合
     - plot のタイトルとか

# インデックスを使った方法が最も汎用的

インデックスを使った方法は他の２つをシミュレートできる

```
for (i in seq_along(x)) {
  name <- names(x)[[i]]
  value <- x[[i]]
}
```

## 21.3.3 Unknown output length

ループする前に出力の長さがわからないとき

# 乱数でベクトルの長さが変わる場合

## ダメな例

```
means <- c(0, 1, 2)
output <- double()
for (i in seq_along(means)) {
  n <- sample(100, 1)
  output <- c(output, rnorm(n, means[[i]]))
}
```

# 良い例

```
out <- vector("list", length(means))
for (i in seq_along(means)) {
  n <- sample(100, 1)
  out[[i]] <- rnorm(n, means[[i]])
}
output <- unlist(out)
```

## 他の例

- paste(out, x) と追加していくのではなく、out をまとめて作ったあと paste(out, collapse=TRUE) でくっつける
- データフレームの行を追加していくのではなく、リストで作って bind_rows(output) でくっつける

# 21.3.4 Unknown sequence length

事前にループする回数がわからないとき

# while

```
while (condition){
  #body
}
```

シミュレーションでよく使う

- 精度が出るまで反復する、とか

## 21.3.5 Exercises

# 21.4 For loops vs. functionals

# 参考文献