

Vectors - r4ds

Tomoya Fukumoto

2019-08-23

ベクトル

- ▶ あるデータの集まりを R ではベクトルと呼ぶ
 - ▶ R のデータはすべてベクトルである
- ▶ R 特有の概念

準備

```
library(tidyverse)
```

二種類のベクトル

アトムック

- ▶ 同じ型のデータの集まり
 - ▶ 1:10
 - ▶ c("a", "b")
 - ▶ TRUE

リスト

- ▶ リスト、アトムック、NULLの集まり
 - ▶ list()
 - ▶ list(list())
 - ▶ list(list(), 1:10, c("a","b"), TRUE)

Vectors

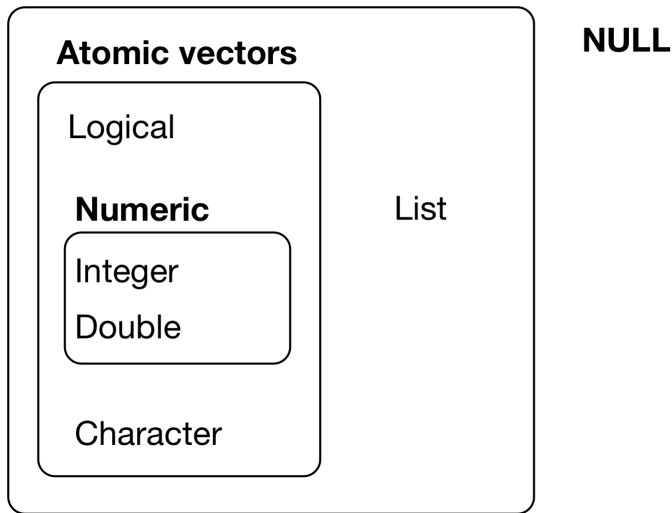


Figure 1: Hierarchy of R's vector types

ベクトルのプロパティ

型

```
typeof(1:10)
```

```
## [1] "integer"
```

```
typeof(list(1,"a"))
```

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

長さ

```
length(1:10)
```

```
## [1] 10
```

拡張されたベクトル

一部のベクトルは属性 (attribute) という付加情報を持たせて複雑な操作ができる。

- ▶ factor : 実体は integer ベクトル
- ▶ date: 実体は double ベクトル
- ▶ data.frame: 実体はリスト

20.3 Important types of atomic vector

- ▶ logical, integer, double, character
 - ▶ complex, raw は扱わない

20.3.1 Logical

最も原子的

値

TRUE, FALSE, NA の三種類のみ

生成

```
1:10 %% 3 == 0
```

```
## [1] FALSE FALSE TRUE FALSE FALSE TRUE FALSE FALSE TR
```

```
c(TRUE, TRUE, FALSE, NA)
```

```
## [1] TRUE TRUE FALSE NA
```


20.3.2 Numeric

Logical の次に原子的

二種類に分類できる - integer - double(デフォルト)

```
typeof(1)
```

```
## [1] "double"
```

```
typeof(1L)
```

```
## [1] "integer"
```

double は近似

```
x <- sqrt(2) ^ 2
```

```
x
```

```
## [1] 2
```

```
x - 2
```

```
## [1] 4.440892e-16
```

教訓

== じゃなくて near を使う

特殊な値

- ▶ NA
- ▶ NaN (double のみ)
- ▶ Inf, -Inf (double のみ)

check

`is.na`, `is.finite`, `is.nan`

20.3.3 Character

要素は任意の文字列

データの大きさが一定でない

Global string pool

文字列の実体は一箇所に保管されている

⇒ データの実体は pool へのリンク

```
x <- "This is a reasonably long string."  
pryr::object_size(x)
```

```
## 152 B
```

```
y <- rep(x, 1000)  
pryr::object_size(y)
```

```
## 8.14 kB
```

20.3.4 Missing values

実は NA にも型が存在する

```
NA           # logical  
NA_integer_  # integer  
NA_real_     # double  
NA_character_ # character
```

20.3.5 Exercises

20.4 Using atomic vectors

1. どうやって型を変換するか
2. ベクトルの型の調べ方
3. 異なる長さのベクトルがどのように作用するか
4. ベクトルの要素に名前を付ける方法
5. ベクトルの要素の抽出方法

20.4.1 Coercion

型の変換

明示的 (Explicit) な方法と暗黙的 (Implicit) な方法

明示的な方法

関数

- ▶ `as.logical`
- ▶ `as.integer`
- ▶ `as.double`
- ▶ `as.character`

`logical > integer > double > character`

の順に変換すればとりあえず間違いない。

逆も不可能ではないけど扱いに注意

暗黙的な方法: 例 1

logical を実数として処理する方法

```
x <- sample(20, 100, replace = TRUE)
sum(x > 10)  # how many are greater than 10?
```

```
## [1] 61
```

TRUE は 1 に、FALSE は 0 になる

暗黙的な方法: 例 2

異なる型の値（ベクトル）を `c` で結合

```
typeof(c(TRUE, 1L))
```

```
## [1] "integer"
```

```
typeof(c(1.5, "a"))
```

```
## [1] "character"
```

logical > integer > double > complex > character
の強さの順で統一される

はっきりと認識しておくこと

ベクトルは複数種類の型の値を要素に持つことはできない

20.4.2 Test functions

どの型のアトミックなのかテストする関数

	lgl	int	dbl	chr	list
<code>is_logical</code>	x				
<code>is_integer</code>		x			
<code>is_double</code>			x		
<code>is_numeric</code>		x	x		
<code>is_character</code>				x	
<code>is_atomic</code>	x	x	x	x	
<code>is_list</code>					x
<code>is_vector</code>	x	x	x	x	x

scalar

`is_scalar_logical` で長さ 1 の lgl かどうかテスト

20.4.3 Scalars and recycling rules

アトムックベクトルどおしの演算

基本ルール

要素ごとに演算される

```
c(1, 2, 3) * c(1, 10, 100)
```

```
## [1]    1   20  300
```

長さが違う場合は？

リサイクル

演算入力のベクトルの長さが異なる場合は短い方が繰り返される

```
1:10 + 100
```

```
## [1] 101 102 103 104 105 106 107 108 109 110
```

```
1:10 + 1:2 * 100
```

```
## [1] 101 202 103 204 105 206 107 208 109 210
```


リサイクル 2

繰り返し回数が合わなければ途中までリサイクル

```
1:10 + 1:3 * 100
```

```
## Warning in 1:10 + 1:3 * 100: longer object length is not  
## shorter object length
```

```
## [1] 101 202 303 104 205 306 107 208 309 110
```

tidyverse

tidyverse な世界ではベクトル-スカラー以外のリサイクルは禁止

```
tibble(x = 1:3, y = 1)
```

```
## # A tibble: 3 x 2
```

```
##       x       y
```

```
##   <int> <dbl>
```

```
## 1     1     1
```

```
## 2     2     1
```

```
## 3     3     1
```

```
tibble(x = 1:3, y = 1:2)
```

```
## Tibble columns must have consistent lengths, only values
```

```
## * Length 2: Column `y`
```

```
## * Length 3: Column `x`
```

20.4.4 Naming vectors

ベクトル要素に名前をつける

ベクトル要素の名前

```
v <- c(x = 1, y = 2, z = 4)
v
```

```
## x y z
## 1 2 4
```

```
names(v)
```

```
## [1] "x" "y" "z"
```

名前の変更

```
v %>% set_names(c("a", "b", "c"))
```

```
## a b c
```

```
## 1 2 4
```

20.4.5 Subsetting

ベクトルの一部の要素を抽出する

operator

ベクトルの後ろに [...] を付ける

[...] の中に入れる値は三種類

- ▶ index
- ▶ logical
- ▶ name strings

index

前から数えた位置で指定する

```
x <- c("one", "two", "three", "four", "five")  
x[c(3, 2, 5)]
```

```
## [1] "three" "two"    "five"
```

```
x[c(1, 1, 5, 5, 5, 2)] #繰り返し
```

```
## [1] "one"  "one"  "five" "five" "five" "two"
```

```
x[c(-1, -3, -5)] #マイナス指定
```

```
## [1] "two"  "four"
```

logical

元と同じ長さの論理ベクトルの TRUE の位置で指定

```
x <- c(10, 3, NA, 5, 8, 1, NA)
x[!is.na(x)] #NA を除く
```

```
## [1] 10 3 5 8 1
```

```
x[x %% 2 == 0] #偶数と NA
```

```
## [1] 10 NA 8 NA
```


name strings

ベクトル要素の名前で指定

```
x <- c(abc = 1, def = 2, xyz = 5)
x[c("xyz", "def")]
```

```
## xyz def
##   5   2
```

nothing

要素を指定せずに全体を得る

```
iris[1,]
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1              5.1           3.5           1.4           0.2 setosa
```

```
iris[,1]
```

```
##      [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.
##      [18] 5.1 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.
##      [35] 4.9 5.0 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.
##      [52] 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.
##      [69] 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.
##      [86] 6.0 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.
##     [103] 7.1 6.3 6.5 7.6 4.9 7.3 6.7 7.2 6.5 6.4 6.8 5.7 5.
##     [120] 6.0 6.9 5.6 7.7 6.3 6.7 7.2 6.2 6.1 6.4 7.2 7.4 7.
##     [137] 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8 6.7 6.7 6.3 6.5 6.
```

subsetting function

実は関数 "[" がコールされている

```
"["(x, 1)
```

```
## abc
```

```
## 1
```

```
x %>% "["(1)
```

```
## abc
```

```
## 1
```

20.4.6 Exercises

20.5 Recursive vectors (lists)

制約が無さ過ぎるベクトル

型の制約なし

アトミックのように要素の型が統一されている必要がない

```
y <- list("a", 1L, 1.5, TRUE)
str(y)
```

```
## List of 4
##  $ : chr "a"
##  $ : int 1
##  $ : num 1.5
##  $ : logi TRUE
```

繰り返し

list は list を要素に持てる

```
z <- list(list(1, 2), list(3, 4, 5))  
str(z)
```

```
## List of 2  
## $ :List of 2  
## ..$ : num 1  
## ..$ : num 2  
## $ :List of 3  
## ..$ : num 3  
## ..$ : num 4  
## ..$ : num 5
```

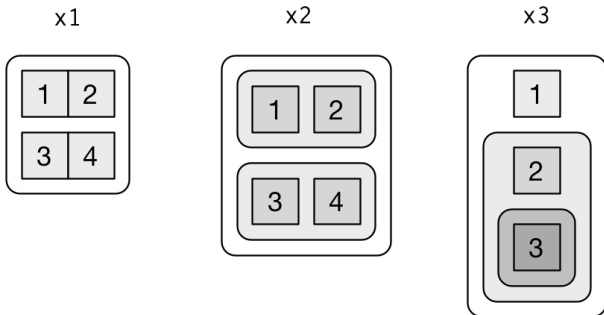
20.5.1 Visualising lists

list 構造の可視化方法

この本のための Hadley さんオリジナル

リストの可視化ルール

```
x1 <- list(c(1, 2), c(3, 4))  
x2 <- list(list(1, 2), list(3, 4))  
x3 <- list(1, list(2, list(3)))
```



- ▶ リストは角が丸、アトミックは四角
- ▶ 子は親の中に入れる。階層が深いとグレー
- ▶ 向きや順序に意味はない

20.5.2 Subsetting

リストには三種の要素抽出方法がある

```
a <- list(a = 1:3, b = "a string", c = pi, d = list(-1, -5))  
a[1]
```

```
## $a  
## [1] 1 2 3
```

```
a[[1]]
```

```
## [1] 1 2 3
```

```
a$a
```

```
## [1] 1 2 3
```

可視化

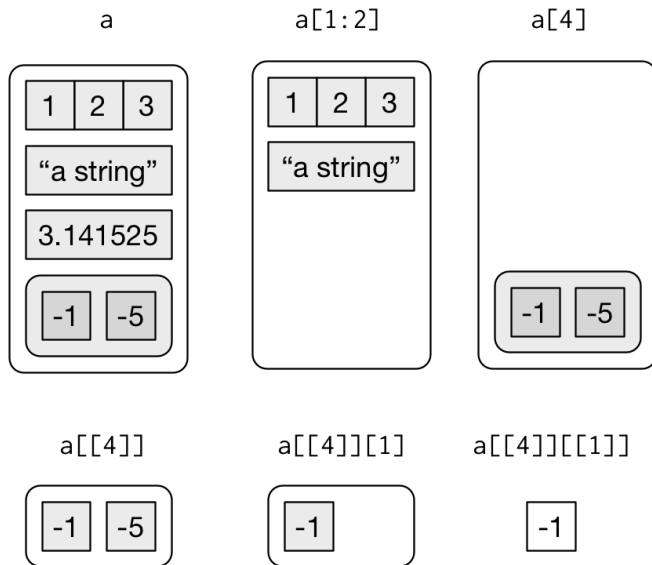



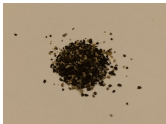


Figure 2: Subsetting a list, visually

20.5.3 Lists of condiments

たとえ話

a	a[1]	a[[1]]	a[[1]][[1]]
			

20.5.4 Exercises

20.6 Attributes

ベクトルの属性（付加的な情報）

設定方法

```
x <- 1:10  
attr(x, "greeting")
```

```
## NULL
```

```
attr(x, "greeting") <- "Hi!"  
attr(x, "farewell") <- "Bye!"  
attributes(x)
```

```
## $greeting  
## [1] "Hi!"  
##  
## $farewell  
## [1] "Bye!"
```

重要な属性

1. names: 要素の名前
2. dimensions: 横や縦の次元を与えればベクトルを行列にできる
3. class: S3 オブジェクト指向プログラミングのために使う
 - ▶ 汎用関数の作用を制御する

汎用関数の例

```
as.Date("2019/8/20")
```

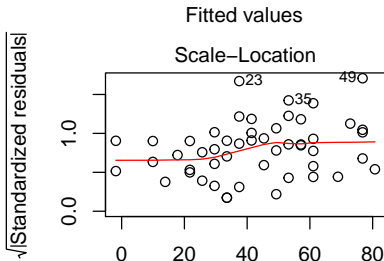
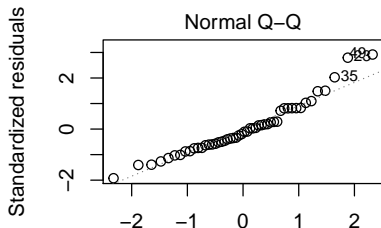
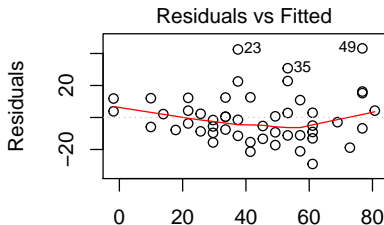
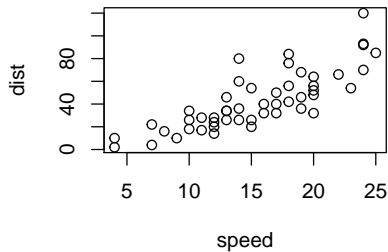
```
## [1] "2019-08-20"
```

```
as.Date(18128)
```

```
## Error in as.Date.numeric(18128): 'origin' must be supplied
```

もう一つの例

```
par(mfrow = c(1,2))  
plot(cars)  
plot(lm(dist ~ speed, cars))
```



20.7 Augmented vectors

attributes の実践的な活用例

- ▶ factor
- ▶ dates
- ▶ datetimes
- ▶ tibbles

20.7.1 Factors

```
x <- factor(c("ab", "cd", "ab"), levels = c("ab", "cd", "ef"),  
typeof(x)
```

```
## [1] "integer"
```

```
attributes(x)
```

```
## $levels  
## [1] "ab" "cd" "ef"  
##  
## $class  
## [1] "factor"
```

20.7.2 Dates and datetimes

date

```
x <- as.Date("1971-01-01")  
unclass(x)
```

```
## [1] 365
```

```
typeof(x)
```

```
## [1] "double"
```

```
attributes(x)
```

```
## $class
```

```
## [1] "Date"
```

datetime

```
x <- lubridate::ymd_hm("1970-01-01 01:00")  
typeof(x)
```

```
## [1] "double"
```

```
attributes(x)
```

```
## $class  
## [1] "POSIXct" "POSIXt"  
##  
## $tzone  
## [1] "UTC"
```

update attribute

```
attr(x, "tzone") <- "US/Pacific"  
x
```

```
## [1] "1969-12-31 17:00:00 PST"
```

```
attr(x, "tzone") <- "US/Eastern"  
x
```

```
## [1] "1969-12-31 20:00:00 EST"
```

20.7.3 Tibbles

```
tb <- tibble::tibble(x = 1:5, y = 5:1)
typeof(tb)
```

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

```
attributes(tb)
```

```
## $names
```

```
## [1] "x" "y"
```

```
##
```

```
## $row.names
```

```
## [1] 1 2 3 4 5
```

```
##
```

```
## $class
```

```
## [1] "tbl_df"      "tbl"        "data.frame"
```


dataframe

```
df <- data.frame(x = 1:5, y = 5:1)
typeof(df)
```

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

```
attributes(df)
```

```
## $names
## [1] "x" "y"
##
## $class
## [1] "data.frame"
##
## $row.names
## [1] 1 2 3 4 5
```

20.7.4 Exercises

参考文献

- ▶ <http://adv-r.had.co.nz/Functions.html#lazy-evaluation>
- ▶ <http://adv-r.had.co.nz/Subsetting.html#applications>
- ▶ <http://adv-r.had.co.nz/OO-essentials.html#s3>