Data Structures in R

Tony Yao-Jen Kuo



collects scalars

- collects scalars
- ▶ can be indexing

- collects scalars
- can be indexing
- can be slicing

- collects scalars
- can be indexing
- can be slicing
- ▶ are iterable

vector

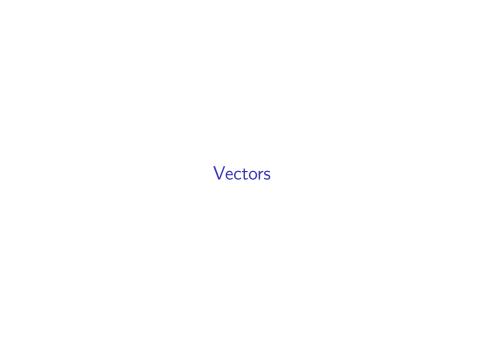
- vector
- ▶ list

- vector
- ▶ list
- ► (optional)factor

- vector
- ► list
- ▶ (optional)factor
- data.frame

- vector
- ▶ list
- ► (optional)factor
- data.frame
- ▶ (optional)matrix

- vector
- ▶ list
- ► (optional)factor
- data.frame
- ▶ (optional)matrix
- ▶ (optional)array



Characteristics of a vector

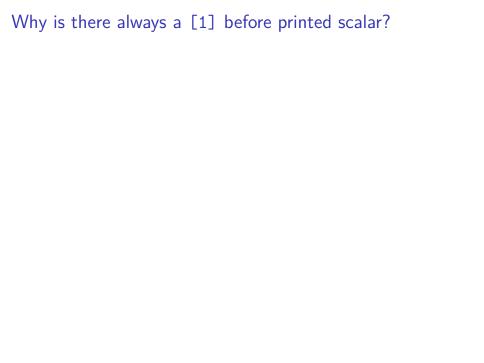
▶ element-wise operation

Characteristics of a vector

- element-wise operation
- uniformed class

Characteristics of a vector

- element-wise operation
- uniformed class
- supports logical filtering



Using c() to create vectors

```
player_names <- c("Jeremy Lin", "Michael Jordan", "Shaquil"
player_heights <- c(191, 198, 216)
player_weights <- c(91, 98, 148)
player_names
player_heights
player_weights</pre>
```

```
## [1] "Jeremy Lin"
## [1] 191 198 216
## [1] 91 98 148
```

"Michael Jordan" "Shaquille 0'1

Using [INDEX] indexing a value from vectors

[1] "Shaquille O'Neal"
[1] "Shaquille O'Neal"

```
player_names[1]
player_names[2]
player_names[3]
player_names[length(player_names)] # in case we have a long
## [1] "Jeremy Lin"
## [1] "Michael Jordan"
```

Using [c(INDICE)] slicing values from vectors

```
player_names[2:3]
player_names[c(1, 3)]
```

```
## [1] "Michael Jordan" "Shaquille O'Neal"
## [1] "Jeremy Lin" "Shaquille O'Neal"
```

What will happen if we set a NEGATIVE index?

```
# Try it yourself
```

Vectors are best known for its...

Element-wise operation

```
player_heights_m <- player_heights / 100
player_heights
player_heights_m</pre>
```

```
## [1] 191 198 216
## [1] 1.91 1.98 2.16
```

Practices: Using vector operations for players' BMIs

```
player_bmis <- # ...</pre>
```

Beware of the types

[1] "character"
[1] "character"
[1] "character"

```
# Name, height, weight, has_ring
mj <- c("Michael Jordan", 198, 98, TRUE)
тj
class(mj[1])
class(mj[2])
class(mj[3])
class(mj[4])
## [1] "Michael Jordan" "198"
                                           "98"
## [1] "character"
```

How to generate vectors quickly

[1] 11 13 15 17 19 21 ## [1] 7 7 7 7 7 7 7

```
11:21

seq(from = 11, to = 21)

seq(from = 11, to = 21, by = 2)

seq(from = 11, to = 21, length.out = 6)

rep(7, times = 7)

## [1] 11 12 13 14 15 16 17 18 19 20 21

## [1] 11 12 13 14 15 16 17 18 19 20 21

## [1] 11 13 15 17 19 21
```

Getting logical values

```
player_heights <- c(191, 198, 216)
player_weights <- c(91, 98, 148)
player_bmis <- player_weights/(player_heights*0.01)**2
player_bmis > 30
```

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

Logical filtering

```
player_bmis[player_bmis > 30]
```

```
## [1] 31.72154
```

Practices: finding odd numbers in random_numbers

```
set.seed(87)
random_numbers <- sample(1:500, size = 100, replace = FALS)</pre>
```

Vector is iterable

```
for (ITERATOR in ITERABLE) {
    # do something iteratively until ITERATOR hits the end o
}
```

Iterator as values

[1] 1.91 ## [1] 1.98 ## [1] 2.16

```
player_heights <- c(191, 198, 216)
for (ph in player_heights) {
   print(ph*0.01)
}</pre>
```

Not just printing it out...

[1] 1.91 1.98 2.16

```
player_heights <- c(191, 198, 216)
player_heights_m <- c()
for (ph in player_heights) {
   player_heights_m <- c(player_heights_m, ph*0.01)
}
player_heights_m</pre>
```

Practices: Applying fizz_buzz() on 1:100

- ▶ if input can be divided by 3, return "fizz"
- ▶ if input can be divided by 5, return "buzz"
- ▶ if input can be divided by 15, return "fizz buzz"
- otherwise, return input itself

```
## [1] 1 2 "fizz" 4 "buzz" ... 14 "fizz buzz" 16 ... 99 "bu
```

Iterators as indice

```
player_names <- c("Jeremy Lin", "Michael Jordan", "Shaquil"
player_heights <- c(191, 198, 216)
for (i in 1:length(player_names)) {
   player_height_m <- player_heights[i]/100
   print(sprintf("%s is %s meter tall", player_names[i], player_names[i]);
}</pre>
```

```
## [1] "Jeremy Lin is 1.91 meter tall"
## [1] "Michael Jordan is 1.98 meter tall"
## [1] "Shaquille O'Neal is 2.16 meter tall"
```

Practices: Is x a prime?

```
is_prime(87) ## FALSE
is_prime(89) ## TRUE
is_prime(91) ## FALSE
```

Practices: How many primes are there between x and y?

```
count_primes(5, 11) ## 3
count_primes(5, 13) ## 4
count_primes(5, 15) ## 4
```

Iterate with another style

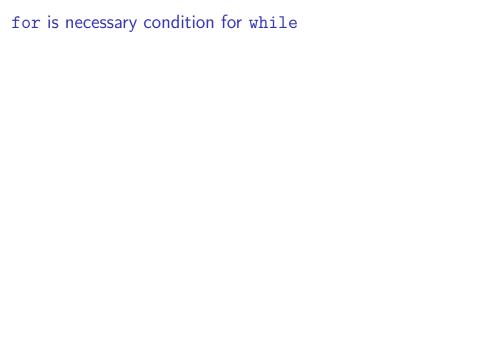
```
while (CONDITION) {
    # do something iteratively when CONDITION == TRUE
}
```

Iterators as indice

```
i <- 1
while (i <= length(player_names)) {
   player_height_m <- player_heights[i]/100
   print(sprintf("%s is %s meter tall", player_names[i], player_i + 1
}</pre>
```

```
## [1] "Jeremy Lin is 1.91 meter tall"
## [1] "Michael Jordan is 1.98 meter tall"
## [1] "Shaquille O'Neal is 2.16 meter tall"
```

Practices: How many times do I have to flip a coin to get 6 heads?



Practices: Fibonacci

▶ Try using 2 types of loop to generate a certain fibonacci array.

```
fibonacci(0, 1, 5) ## [1] 0, 1, 1, 2, 3
fibonacci(0, 1, 7) ## [1] 0, 1, 1, 2, 3, 5, 8
fibonacci(0, 1, 9) ## [1] 0, 1, 1, 2, 3, 5, 8, 13, 21
```

Practices: Poker card deck

```
suits <- c("Spade", "Heart", "Diamond", "Clover")
ranks <- c("Ace", 2:10, "Jack", "Queen", "King")</pre>
```

Lists

Characteristics of lists

► Different classes

Characteristics of lists

- Different classes
- Supports \$ selection like attributes

Using list() to create a list

```
infinity_war <- list(
  "Avengers: Infinity War",
  2018,
  8.6,
  c("Action", "Adventure", "Fantasy")
)
class(infinity_war)</pre>
```

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

Check the apperance of a list

```
infinity_war
```

```
## [[1]]
## [1] "Avengers: Infinity War"
##
## [[2]]
## [1] 2018
##
## [[3]]
## [1] 8.6
##
## [[4]]
## [1] "Action"
                   "Adventure" "Fantasy"
```

Using [[INDEX]] indexing list

[1] 8.6

```
for (i in 1:length(infinity_war)) {
  print(infinity_war[[i]])
}

## [1] "Avengers: Infinity War"
## [1] 2018
```

[1] "Action" "Adventure" "Fantasy"

Giving names to elements in list

```
infinity_war <- list(
  movieTitle = "Avengers: Infinity War",
  releaseYear = 2018,
  rating = 8.6,
  genre = c("Action", "Adventure", "Fantasy")
)
infinity_war

## $movieTitle</pre>
```

```
## [1] "Avengers: Infinity War"
##
## $releaseYear
## [1] 2018
```

\$rating ## [1] 8.6

\$genre

##

Using [["ELEMENT"]] indexing list

```
for (e in names(infinity_war)) {
   print(infinity_war[[e]])
}

## [1] "Avengers: Infinity War"

## [1] 2018

## [1] 8.6

## [1] "Action" "Adventure" "Fantasy"
```

Using \$ELEMENT indexing list

```
infinity_war$movieTitle
infinity_war$releaseYear
infinity_war$rating
infinity_war$genre
```

```
## [1] "Avengers: Infinity War"
## [1] 2018
## [1] 8.6
## [1] "Action" "Adventure" "Fantasy"
```

Every element keeps its original class

[1] "numeric"
[1] "character"

```
for (e in names(infinity_war)) {
  print(class(infinity_war[[e]]))
}

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

Practices: Getting favorite players' last names in upper cases

Hint: using strsplit() to split players' name and using toupper() for upper cases.

```
fav_players <- c("Steve Nash", "Paul Pierce", "Dirk Nowitz"
# [1] "NASH" "PIERCE" "NOWITZKI" "GARNETT" "OLAJUWON"</pre>
```



Acts like a character vector

- Acts like a character vector
- Unique character is recorded as Levels

- Acts like a character vector
- Unique character is recorded as Levels
- Supports ordinal values and each character is encoded as integers

- Acts like a character vector
- Unique character is recorded as Levels
- Supports ordinal values and each character is encoded as integers
- ▶ Default class of a character column

Using factor() to create a factor

```
all_time_fantasy <- c("Steve Nash", "Paul Pierce", "Dirk No
class(all_time_fantasy)
all_time_fantasy <- factor(all_time_fantasy)
class(all_time_fantasy)
```

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

Unique character in factor is recorded with levels

```
rgbs <- factor(c("red", "green", "blue", "blue", "green",
rgbs</pre>
```

```
## [1] red green blue blue green green
## Levels: blue green red
```

Supports ordinal values

[1] TRUE

Levels: cold < cool < freezing < hot < warm

Adjusting the order of a factor

```
## [1] freezing cold cool warm hot
## Levels: freezing < cold < cool < warm < hot</pre>
```

Elements in factor are encoded as integers

```
temperatures <- c("freezing", "cold", "cool", "warm", "hot"
as.numeric(temperatures) # Error
temperatures <- factor(c("freezing", "cold", "cool", "warm"
as.numeric(temperatures)</pre>
```

Factors sometimes are hard to handle...

```
all_time_fantasy <- factor(c("Steve Nash", "Paul Pierce",
all_time_fantasy <- c(all_time_fantasy, "Ray Allen")
all_time_fantasy</pre>
```

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

Data Frames

▶ Has 2 dimensions m x n as in rows x columns

- ▶ Has 2 dimensions m x n as in rows x columns
- Rows are denoted as observations, while columns are denoted as variables

- ▶ Has 2 dimensions m x n as in rows x columns
- Rows are denoted as observations, while columns are denoted as variables
- Each column has its own class

- ▶ Has 2 dimensions m x n as in rows x columns
- Rows are denoted as observations, while columns are denoted as variables
- Each column has its own class
- Supports \$ selection like attributes

Using data.frame() to create a data frame

```
player_names <- c("Jeremy Lin", "Michael Jordan", "Shaquill
player_heights <- c(191, 198, 216)
player_weights <- c(91, 98, 148)
has_rings <- c(FALSE, TRUE, TRUE)
player_df <- data.frame(player_names, player_heights, player_heights)</pre>
```

player_names	player_heights	player_weights	has_rings	
Jeremy Lin	191	91	FALSE	
Michael Jordan	198	98	TRUE	
Shaquille O'Neal	216	148	TRUE	

Character vectors are encoded as factors by default

str(player df)

```
## 'data.frame': 3 obs. of 4 variables:
## $ player_names : Factor w/ 3 levels "Jeremy Lin", "Mic!
## $ player_heights: num 191 198 216
## $ player_weights: num 91 98 148
## $ has rings : logi FALSE TRUE TRUE
```

Using stringsAsFactors = FALSE for character class

```
## 'data.frame': 3 obs. of 4 variables:
## $ player_names : chr "Jeremy Lin" "Michael Jordan" "3
## $ player_heights: num 191 198 216
## $ player_weights: num 91 98 148
```

\$ has_rings : logi FALSE TRUE TRUE

##

player_df <- data.frame(player_names, player_heights, player_</pre>

Selecting column from data frames as a vector

- Using column names in double quotes
- or column indice

[1] "Jeremy Lin"

```
player_df[["player_names"]]
player_df[, "player_names"]
player_df[, 1]
```

```
## [1] "Jeremy Lin" "Michael Jordan" "Shaquille 0'!
## [1] "Jeremy Lin" "Michael Jordan" "Shaquille 0'!
```

"Michael Jordan"

"Shaquille 0'

Or using \$ like attributes

```
player_df$player_names
```

[1] "Jeremy Lin"

"Michael Jordan" "Shaquille O'l

Subsetting observations from data frames

Using row indice

3 Shaquille O'Neal

```
player_df[c(2, 3), ]
```

```
## player_names player_heights player_weights has_ris
## 2 Michael Jordan 198 98 Tl
```

216

148

Tl

More commonly, using a logical vector

```
player_df[player_df$has_rings, ] # players with rings
player df[!player df$has rings, ] # players without rings
##
         player names player heights player weights has ris
       Michael Jordan
                                  198
## 2
                                                  98
## 3 Shaquille O'Neal
                                 216
                                                 148
                                                          Tl
     player_names player_heights player_weights has_rings
##
       Jeremy Lin
                                              91
                                                     FALSE
## 1
                             191
```

Creating logical vectors using operators

Remember putting logical vector at the row index

```
player_df$player_heights > 200
player_df[player_df$player_heights > 200, ]
```

```
## [1] FALSE FALSE TRUE
## player_names player_heights player_weights has_rim
## 3 Shaquille O'Neal 216 148 TI
```



Creating a matrix using matrix()

```
my_mat <- matrix(1:4, nrow = 2)
class(my_mat)</pre>
```

```
## [1] "matrix"
```

matrix operations

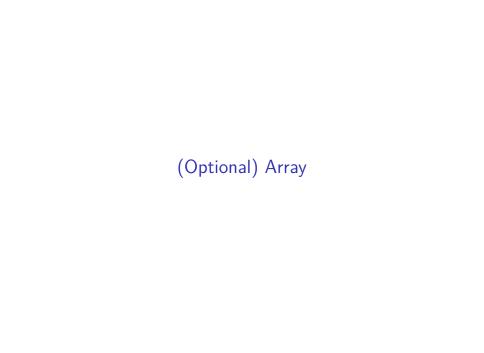
- Using * for element-wise multiplication
- Using t for transpose
- Using %*% for matrix multiplication

```
my_mat <- matrix(1:4)
my_mat * my_mat
t(my_mat) %*% my_mat</pre>
```

```
## [,1]
## [1,] 1
## [2,] 4
## [3,] 9
## [4,] 16
## [,1]
## [1,] 30
```

Practices: Make a 9×9 multiplication matrix

##		[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]
##	[1,]	1	2	3	4	5	6	7	8	9
##	[2,]	2	4	6	8	10	12	14	16	18
##	[3,]	3	6	9	12	15	18	21	24	27
##	[4,]	4	8	12	16	20	24	28	32	36
##	[5,]	5	10	15	20	25	30	35	40	45
##	[6,]	6	12	18	24	30	36	42	48	54
##	[7,]	7	14	21	28	35	42	49	56	63
##	[8,]	8	16	24	32	40	48	56	64	72
##	[9,]	9	18	27	36	45	54	63	72	81



[,1] [,2] [,3]

19

20

21

23

24

[1,] 13 17

[3,] 15

[2,] 14 18 22

16

##

##

[4]

```
Using array() to create an array
   my_arr \leftarrow array(1:24, dim = c(4, 3, 2))
   my_arr
   ## , , 1
   ##
   ##
          [,1] [,2] [,3]
   ## [1,]
             1
                 5
   ## [2,] 2 6 10
   ## [3,] 3 7 11
   ## [4,] 4
              8
                   12
   ##
   ## , , 2
```