Data Structures in R

Tony Yao-Jen Kuo



vector

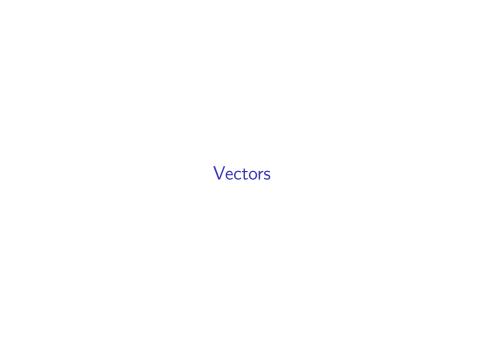
- vector
- ▶ list

- 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

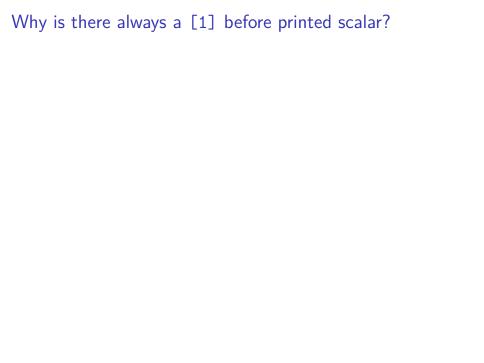


▶ element-wise operation

- element-wise operation
- uniformed class

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

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



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] 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
```

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
}
```

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>
```

Iterators as indice

```
player_names <- c("Jeremy Lin", "Michael Jordan", "Shaquill
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"
```

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"
```

Lists

Characteristics of lists

► Different classes

Characteristics of lists

- Different classes
- Supports \$ selection like attributes

Characteristics of lists

- Different classes
- Supports \$ selection like attributes
- Iterable

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
```

\$rating ## [1] 8.6

\$genre

[1] 2018

##

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>
```

Factors

Data Frames

Characteristics of data frames

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

Characteristics of data frames

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

Characteristics of data frames

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