R Essentials

Vector Manipulation and Other Data Structures

Tony Yao-Jen Kuo

Agenda

- An overview
- Vector manipulation
- Flow of control: iteration
- `list`
- `factor`
- `data.frame`
- `matrix`
- `array`

An overview

Data structures have several important properties

- collects smaller vectors
- can be indexing
- can be slicing
- are iterable

Vector manipulation

Characteristics of a vector

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

Why is there always a `[1]` before we `print()` something in console?

```
In [1]: print("Hello world")
```

```
[1] "Hello world"
```

Using c() to create vectors with length larger than 1

In [2]:

```
player_names <- c("Jeremy Lin", "Michael Jordan", "Shaquille O'Neal")
player_heights <- c(191, 198, 216)
player_weights <- c(91, 98, 148)
player_names
player_heights
player_weights
```

```
'Jeremy Lin'   'Michael Jordan'   'Shaquille O\'Neal'

191  198  216

91  98  148
```

Using `[INDEX]` to select a vector of length 1 from a vector of length larger than 1

In [3]:

```
player_names[1]
player_names[2]
player_names[3]
player_names[length(player_names)] # in case we have a long vector
```

```
'Jeremy Lin'

'Michael Jordan'

'Shaquille O\'Neal'

'Shaquille O\'Neal'
```

Using `[c(INDICE)]` to slice values from vectors

```
In [4]: player_names[2:3]
        player_names[c(1, 3)]
```

```
'Michael Jordan'   'Shaquille O\'Neal'

'Jeremy Lin'   'Shaquille O\'Neal'
```

What will happen if we use a NEGATIVE index?

In [5]:

```
# Try it yourself
```

Vectors are best known for its...

- Element-wise operation

In [6]:
```
player_heights_m <- player_heights / 100
player_heights
player_heights_m
```

```
191  198  216

1.91  1.98  2.16
```

Practices: Using vector operations for players’ BMIs

In [7]:

```
# player_bmis <- ...
```

Beware of vector types

In [8]:

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

```
'Michael Jordan'  '198'  '98'  'TRUE'

'character'

'character'

'character'

'character'
```

How to generate vectors quickly

In [9]:

```r
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 = 8)
```

```
11  12  13  14  15  16  17  18  19  20  21

11  12  13  14  15  16  17  18  19  20  21

11  13  15  17  19  21

11  13  15  17  19  21

7  7  7  7  7  7  7  7
```

Getting logical values

In [10]:

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

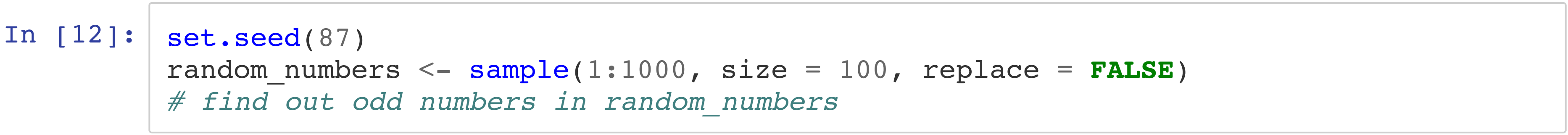
```
FALSE  FALSE  TRUE
```

Logical filtering

```
In [11]: player_bmis[player_bmis > 30]
```

```
31.721536351166
```

Practices: finding odd numbers in `random_numbers`



In [12]:

```
set.seed(87)
random_numbers <- sample(1:1000, size = 100, replace = FALSE)
# find out odd numbers in random_numbers
```

Flow of control: iteration

Vector is iterable

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

Iterator as value

```
In [13]: player_heights <- c(191, 198, 216)
         for (ph in player_heights) {
           print(ph*0.01)
         }
```

```
[1] 1.91
[1] 1.98
[1] 2.16
```

Not just printing it out...

In [14]:

```
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
```

```
1.91  1.98  2.16
```

Practices: Applying fizz buzz on 1:100

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

In [15]:

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

Iterator as index

In [16]:

```
player_names <- c("Jeremy Lin", "Michael Jordan", "Shaquille O'Neal")
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_height_m))
}
```

```
[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?

In [17]:

```
x <- 87
# ...
# FALSE
x <- 89
# ...
# TRUE
```

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

In [18]:

```
x <- 5
y <- 11
# ...
# 3
x <- 5
y <- 13
# ...
# 4
```

Iterate with another style

```
while (EXPR) {
  # do something iteratively when EXPR is evaluated as TRUE
}
```

In [19]:

```
player_names <- c("Jeremy Lin", "Michael Jordan", "Shaquille O'Neal")
player_heights <- c(191, 198, 216)
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_height_m))
  i <- i + 1
}
```

```
[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"
```

Important reserved words when working with iterations

- `break`
- `next`

In [20]:

```
for (i in 1:12) {
  if (i == 4) {
    break
  }
  print(i)
}
```

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

In [21]:

```
for (i in 1:12) {
  if (i == 4) {
    next
  }
  print(i)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
[1] 11
[1] 12
```

Practices: Sampling from 1:1000 until we get a number that can be divided by 89

Practices: How many times do I have to roll a dice to have 3 times of six?

`for` is necessary condition for `while`

Practices: Fibonacci

Try using 2 types of loop to generate a certain fibonacci sequence

https://en.wikipedia.org/wiki/Fibonacci_number
(https://en.wikipedia.org/wiki/Fibonacci_number)

In [22]:

```
fib_1 <- 0
fib_2 <- 1
fib_len <- 5
# ...
# 0, 1, 1, 2, 3
```

Practices: Poker card deck

In [23]:

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

```
list
```

Characteristics of lists

- Different classes
- Supports $ selection like attributes

Using `list()` to create a list

In [24]:

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

'list'

Check the apperance of a list

```
In [25]: infinity_war
```

```
1. 'Avengers: Infinity War'
2. 2018
3. 8.6
4.        'Action'   'Adventure'   'Fantasy'
```

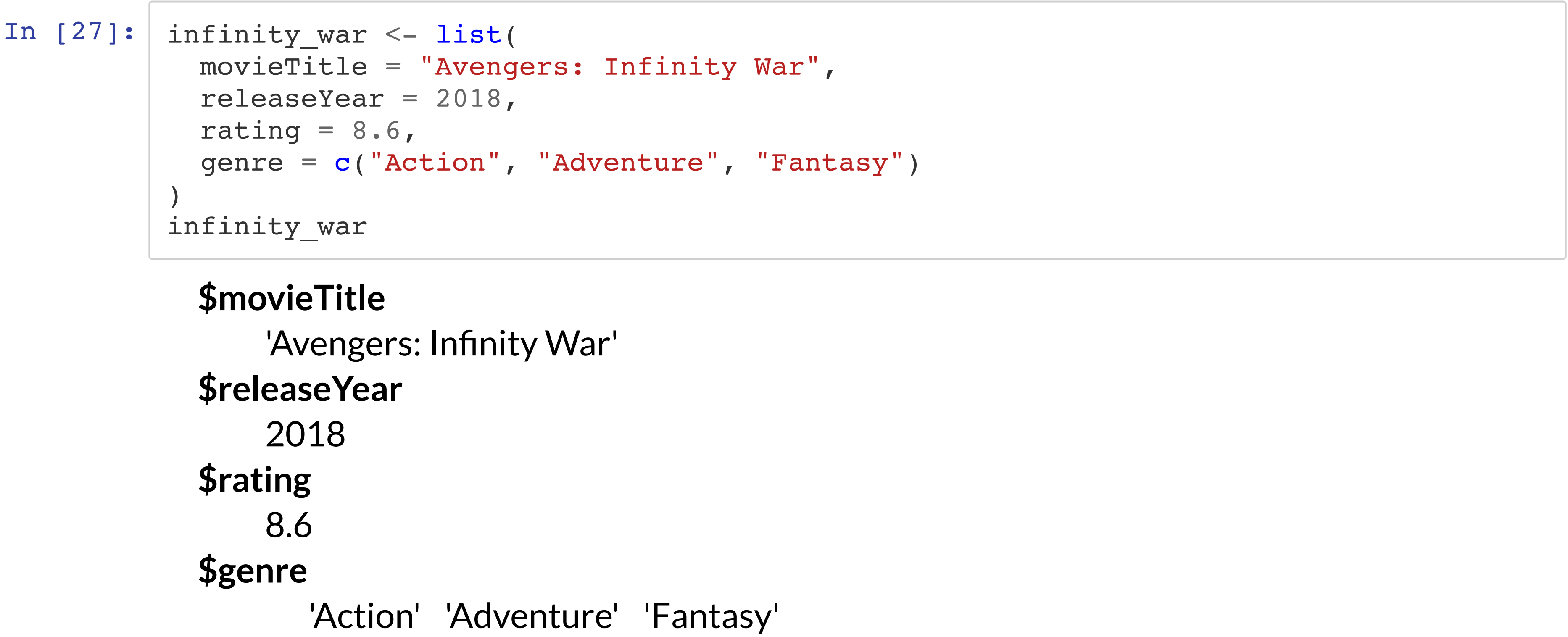
Using `[[INDEX]]` indexing list

In [26]:

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

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

Giving names to elements in list



In [27]:

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

```
$movieTitle
    'Avengers: Infinity War'
$releaseYear
    2018
$rating
    8.6
$genre
       'Action'  'Adventure'  'Fantasy'
```

Using [[”ELEMENT”]] indexing list

```
In [28]: 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

```
In [29]: infinity_war$movieTitle
         infinity_war$releaseYear
         infinity_war$rating
         infinity_war$genre
```

```
'Avengers: Infinity War'

2018

8.6

    'Action'   'Adventure'   'Fantasy'
```

Every element keeps its original class

In [30]:

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

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

Practices: Getting favorite players’ last names in upper cases

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

In [31]:

```
fav_players <- c("Steve Nash", "Paul Pierce", "Dirk Nowitzki", "Kevin Garnett", "Hakeem Olajuwon")
# ...
# [1] "NASH" "PIERCE" "NOWITZKI" "GARNETT" "OLAJUWON"
```

factor

Characteristics of `factor`

- 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

In [32]:

```
all_time_fantasy <- c("Steve Nash", "Paul Pierce", "Dirk Nowitzki", "Kevin Garnett", "Hakeem Olajuwon")
class(all_time_fantasy)
all_time_fantasy <- factor(all_time_fantasy)
class(all_time_fantasy)
```

```
'character'

'factor'
```

Unique character in factor is recorded with levels

In [33]:

```
rgbs <- factor(c("red", "green", "blue", "blue", "green", "green"))
rgbs
```

```
red  green  blue  blue  green  green
```

▶ **Levels:**

Factor supports ordinal values

```
In [34]: temperatures <- factor(c("freezing", "cold", "cool", "warm", "hot"),
                                ordered = TRUE)
         temperatures
         temperatures[1] > temperatures[3]
```

```
 freezing  cold  cool  warm  hot
▶ Levels:

TRUE
```

Adjusting the order of a factor

In [35]:

```
temperatures <- factor(c("freezing", "cold", "cool", "warm", "hot"),
                       ordered = TRUE,
                       levels = c("freezing", "cold", "cool", "warm", "hot"))
temperatures
```

```
freezing  cold  cool  warm  hot
```

▶ **Levels:**

Elements in factor are encoded as integers

```
In [36]: temperatures <- c("freezing", "cold", "cool", "warm", "hot")
         as.numeric(temperatures) # Error
         temperatures <- factor(c("freezing", "cold", "cool", "warm", "hot"))
         as.numeric(temperatures)
```

```
Warning message in eval(expr, envir, enclos):
“NAs introduced by coercion”
```

```
<NA> <NA> <NA> <NA> <NA>
```

```
3 1 2 5 4
```

Factors sometimes are hard to handle...

In [37]:

```
all_time_fantasy <- factor(c("Steve Nash", "Paul Pierce", "Dirk Nowitzki", "Kevin
 Garnett", "Hakeem Olajuwon"))
all_time_fantasy <- c(all_time_fantasy, "Ray Allen")
all_time_fantasy
```

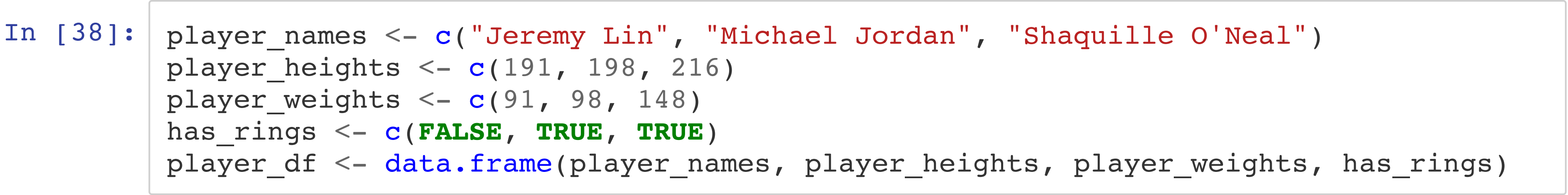
```
'5'  '4'  '1'  '3'  '2'  'Ray Allen'
```

`data.frame`

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
- Each column has its own class
- Supports `$` selection like attributes

Using `data.frame()` to create a data frame



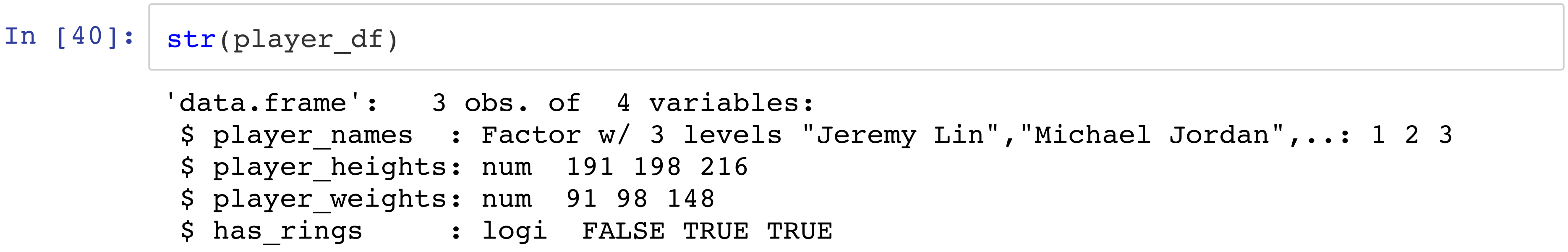
In [38]:

```
player_names <- c("Jeremy Lin", "Michael Jordan", "Shaquille O'Neal")
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_weights, has_rings)
```

```
In [39]: player_df
```

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

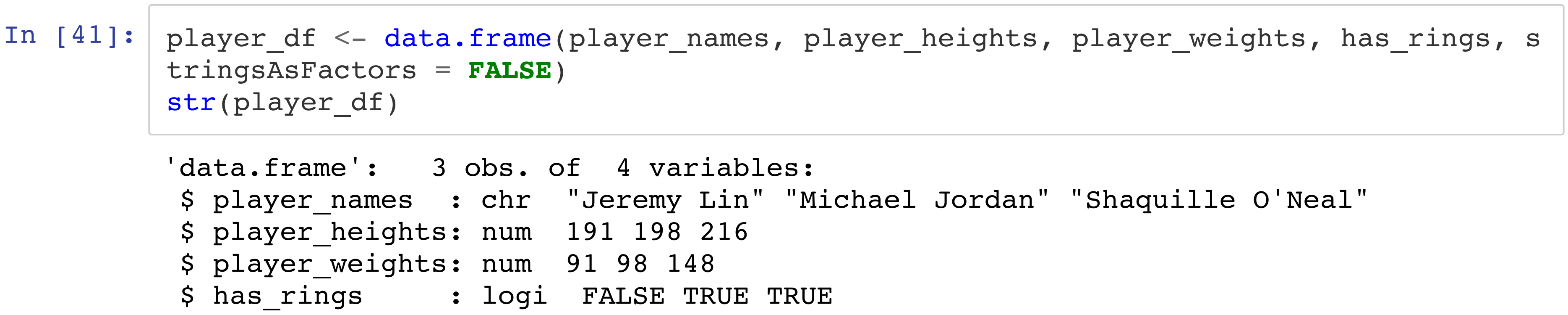
Character vectors are encoded as factors by default



```
In [40]: str(player_df)
```

```
'data.frame':	3 obs. of  4 variables:
 $ player_names  : Factor w/ 3 levels "Jeremy Lin","Michael Jordan",..: 1 2 3
 $ player_heights: num  191 198 216
 $ player_weights: num  91 98 148
 $ has_rings     : logi  FALSE TRUE TRUE
```

Using `stringsAsFactors = FALSE` for character class



```
In [41]: player_df <- data.frame(player_names, player_heights, player_weights, has_rings, s
         tringsAsFactors = FALSE)
         str(player_df)
```

```
'data.frame':	3 obs. of  4 variables:
 $ player_names  : chr  "Jeremy Lin" "Michael Jordan" "Shaquille O'Neal"
 $ player_heights: num  191 198 216
 $ player_weights: num  91 98 148
 $ has_rings     : logi  FALSE TRUE TRUE
```

Selecting column from data frames as a vector

- Using column names in double quotes
- or column indice

In [42]:

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

```
'Jeremy Lin'   'Michael Jordan'   'Shaquille O\'Neal'

'Jeremy Lin'   'Michael Jordan'   'Shaquille O\'Neal'

'Jeremy Lin'   'Michael Jordan'   'Shaquille O\'Neal'
```

Or using $ like attributes

```
In [43]: player_df$player_names
```

```
'Jeremy Lin'   'Michael Jordan'   'Shaquille O\'Neal'
```

Subsetting observations from data frames

Using row indice.

```
In [44]: player_df[c(2, 3), ]
```

	player_names	player_heights	player_weights	has_rings
2	Michael Jordan	198	98	TRUE
3	Shaquille O'Neal	216	148	TRUE

More commonly, using a logical vector

In [45]:

```
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_rings
2	Michael Jordan	198	98	TRUE
3	Shaquille O'Neal	216	148	TRUE

player_names	player_heights	player_weights	has_rings
Jeremy Lin	191	91	FALSE

Creating logical vectors using logical operators

Remember putting logical vector at the **row** index.

In [46]:

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

```
FALSE  FALSE  TRUE
```

	player_names	player_heights	player_weights	has_rings
3	Shaquille O'Neal	216	148	TRUE

`matrix`

Creating a matrix using `matrix()`

In [47]:

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

```
'matrix'
```

matrix operations

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

In [48]:

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

```
 1
 4
 9
16

30
```

Practices: Make a ”99 multiplication matrix”

`array`

Using `array()` to create an array

In [49]:

```
my_arr <- array(1:24, dim = c(4, 3, 2))
my_arr
class(my_arr)
```

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

```
'array'
```