Variable Types in R

Tony Yao-Jen Kuo

An overview

We are gonna talk about 7 of them

- numeric

We are gonna talk about 7 of them

- numeric
- (optional)integer

We are gonna talk about 7 of them

- numeric
- (optional)integer
- (optional)complex

We are gonna talk about 7 of them

- numeric
- (optional)integer
- (optional)complex
- character

We are gonna talk about 7 of them

- numeric
- (optional)integer
- (optional)complex
- character
- logical

We are gonna talk about 7 of them

- numeric
- (optional)integer
- (optional)complex
- character
- logical
- Date

We are gonna talk about 7 of them

- numeric
- (optional)integer
- (optional)complex
- character
- logical
- Date
- POSIXct

Using `class()` to examine variable types

```
class(87)
class("Luke Skywalker")
class(TRUE)
```

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

Numerics

Using numeric as the primary digit type

```r
my_lucky_number <- 24
class(my_lucky_number) # numeric
class(2.4) # numeric
class(-8.7) # numeric
class(0) # numeric
```

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

Using L to denote the integer type

```
my_lucky_integer <- 87L
class(my_lucky_integer) # integer
my_lucky_integer
```

```
## [1] "integer"
## [1] 87
```

Using i to denote the imaginary part

```
my_lucky_complex <- 8 + 7i
class(my_lucky_complex) # complex
my_lucky_complex
```

```
## [1] "complex"
## [1] 8+7i
```

Numeric operations

Operators	Usage
+	add
-	minus
*	multiply
/	divide
** or ^	power
%%	modulo
%/%	quotient

- Using () to prioritize operations

Practices: Jeremy Lin's BMI

$$BMI = \frac{weight_{kg}}{height_m^2}$$

```
jeremyLin_height <- 191
jeremyLin_weight <- 91
```

Practices: NBA players' BMIs

```
steveNash_height <- 191
steveNash_weight <- 82
shaq_height <- 216
shaq_weight <- 148
jordan_height <- 198
jordan_weight <- 98
```

We need FUNCTION to help us

```
MY_FUNCTION <- function(x, y, arg1, arg2, ...) {
  # using x, y, arg1, arg2 to get output
  return(OUTPUT)
}
```

get_bmi() function

```
get_bmi <- function(height, weight) {
  bmi <- weight/(height*0.01)**2
  return(bmi)
}
get_bmi(steveNash_height, steveNash_weight)
get_bmi(shaq_height, shaq_weight)
get_bmi(jordan_height, jordan_weight)
```

```
## [1] 22.47745
## [1] 31.72154
## [1] 24.99745
```

character

Using ” or “” for characters

```
mj <- "Michael Jordan"
class(mj)
mj <- 'Michael Jordan'
class(mj)
```

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

When to use ” or “”?

```
# Try to assign one of the greatest center in NBA history
Shaquille O'Neal
```

Practices: What did Ross Geller say?

```
Let's put aside the fact that you "accidentally" pick up my
```

Using `sprintf()` for string print with format

```
jordan_BMI <- get_bmi(jordan_height, jordan_weight)
sprintf("Michael Jordan's BMI is %s", jordan_BMI)
```

```
## [1] "Michael Jordan's BMI is 24.9974492398735"
```

Common formats using `sprintf()`

- `%s`: pure text
- `%f`: float
- `%e`: scientific notation

```
?sprintf # this function is well-documented!
```

logical

Black or White, Head or Tail

```
class(TRUE)
class(FALSE)
class(True)  # error
class(False) # error
class(true)  # error
class(false) # error
```

Logical operators

Operators	Usage
`>`	larger than
`>=`	larger than or equal to
`<`	smaller than
`<=`	smaller than or equal to
`==`	equal to
`!=`	not equal to
`%in%`	belong to
`!`	not

Comparing characters

```
"Z" > "z" > "Y" > "y" > "X" > "x" ... > "B" > "b" >
"A" > "a"
```

Connecting logicals

- & for logical AND
- | for logical OR

Is Shaq overweight?

- Well, that depends on the definition of overweight

```
# Overweight: BMI > 30 AND body fat > 25%

# Overweight: BMI > 30 OR body fat > 25%
```

Using logicals in control statements

```
# 1 branch
if (CONDITION) {
  # do something when CONDITION equals TRUE
}
```

Using `if-else` for 2 branches

```
# 2 branches
if (CONDITION) {
  # do something when CONDITION equals TRUE
} else {
  # do something when CONDITION equals FALSE
}
```

Using `if-else` `if-else` for more than 3 branches

```
# 2 branches
if (CONDITION_1) {
  # do something when CONDITION_1 equals TRUE
} else if (CONDITION_2) {
  # do something when CONDITION_2 equals TRUE
} else {
  # do something when both CONDITION_1 and CONDITION_2 equ
}
```

Practices: get_bmi_label()

https://en.wikipedia.org/wiki/Body_mass_index

```
get_bmi_label <- function(player_name, height, weight) {
  # ...
}
get_bmi_label("Jeremy Lin", 191, 91)
# [1] "Jeremy Lin's BMI label is Normal"
```

Practices: `fizz_buzz()`

- 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

```
fizz_buzz <- function(x) {
  # ...
}
fizz_buzz(6)  # [1] "fizz"
fizz_buzz(10) # [1] "buzz"
fizz_buzz(30) # [1] "fizz buzz"
fizz_buzz(31) # [1] 31
```

Class judgement and class conversion

Using `is.___()` for judgement

- `is.numeric()`

Using `is.___()` for judgement

- `is.numeric()`
- `is.character()`

Using `is.___()` for judgement

- `is.numeric()`
- `is.character()`
- `is.logical()`

Using `as.___()` for conversion

- `as.numeric()`

Using `as.___()` for conversion

- `as.numeric()`
- `as.character()`

Using `as.___()` for conversion

- `as.numeric()`
- `as.character()`
- `as.logical()`

Date and Datetime

Using `Sys.Date()` for current date

```
Sys.Date()
```

```
## [1] "2018-09-04"
```

Mysterious number

```
sys_date <- Sys.Date()
as.numeric(sys_date) # what is this number?
```

```
## [1] 17778
```

Date originates from 1970-01-01

```
sys_date - as.numeric(sys_date)
```

```
## [1] "1970-01-01"
```

Every integer stands for a specific date

```
original_date <- sys_date - as.numeric(sys_date)
original_date - 1
original_date
original_date + 1
```

```
## [1] "1969-12-31"
## [1] "1970-01-01"
## [1] "1970-01-02"
```

Using `Sys.time()` for current datetime

```
Sys.time()
```

```
## [1] "2018-09-04 09:37:02 CST"
```

Datetime originates from 1970-01-01 08:00:00

```
sys_datetime <- Sys.time()
original_datetime <- sys_datetime - as.numeric(sys_datetime
original_datetime
```

```
## [1] "1970-01-01 08:00:00 CST"
```

Every integer stands for a specific second

```
sys_datetime <- Sys.time()
original_datetime <- sys_datetime - as.numeric(sys_datetime
original_datetime - 1
original_datetime
original_datetime + 1
```

```
## [1] "1970-01-01 07:59:59 CST"
## [1] "1970-01-01 08:00:00 CST"
## [1] "1970-01-01 08:00:01 CST"
```

Using `OlsonNames()` for specific timezone

Practices: The 911 earthquake

1999-09-21 01:47:16 the Jiji earthquake occured in Nantou, Taiwan with a Richter scale of 7.3. The first major after-shock occured on 1999-09-21 01:57:15. Let us know how long was the time between these two shocks.