

M12-base

learningSpoonsR@gmail.com



I. 변수와 함수 (Variable & Function)

II. 데이터 타입 (Data Type)

III. 데이터 구조 (Data Structure)

IV. 프로그램 제어 (Control Statement)

Appendix A. Thousand commas & DT & `data.table`

FAQ

I. 변수와 함수 (Variable & Function)

문자(String, Character) 변수의 입력

1. 문자로 된 입력은 따옴표를 넣어줘야 합니다.

```
Hello
```

```
## Error: unexpected symbol in "Hello"
```

```
"Hello"
```

```
## [1] "Hello"
```

2. 변수 입력 (Variable Assignment)

```
greeting <- "Hello" # Assign "Hello" to variable name `greeting`  
greeting
```

```
## [1] "Hello"
```

3. 문자끼리는 덧셈 기호를 이용해서 더할수 없다.

```
greeting + "World"
```

```
## Error in greeting + "Hello" : non-numeric argument to binary operator
```

그렇다면 문자끼리 더하려면 어떻게 해야 할까?

문자 두 개를 합치려면?

- ▶ google 'R combine two strings'

- ▶ google 'R에서 문자를 합치기'

```
paste(greeting, "World")
```

```
## [1] "Hello World"
```

- ▶ R에서는 +로 문자를 합칠 수 없습니다.

- ▶ 그런데 파이썬에서는 됩니다.

- ▶ 결국 다 외울수 없기에 많은 부분에서 검색을 활용해야 합니다.

- ▶ (이 부분이 초보자에게 가장 익숙하지 않습니다.)

- ▶ 이항연산자 (binary operator)

- ▶ 2개의 수치를 이용해서 연산하는 함수

- ▶ +, -, *, /, >, >=, ==, !=

- ▶ 여러분은 이제 모든 언어에서 문자를 합하고 출력하는 방법을 알고 있습니다?!

- ▶ google 'R download'

- ▶ google 'R combine two strings'

변수 입력

입력

- ▶ a라는 변수에 'apple'이라는 문자를 입력하는 명령
 - ▶ `a = "apple"`을 사용해도 같음
 - ▶ 그러나 집어넣는다는 의미로서 `"<-"`을 권장
 - ▶ 홑따옴표와 쌍따옴표의 기능은 대부분 경우에 같음

```
a <- "apple"
```

- ▶ 아래의 4가지 명령이 동일합니다

```
a <- "apple"
```

```
a <- 'apple'
```

```
a = "apple"
```

```
a = 'apple'
```

입력 확인 (1)

▶ '=' (is equal) 는 같으면 **TRUE** 다르면 **FALSE**를 반환합니다.

▶ '!=' (is not equal) 는 같으면 **TRUE** 다르면 **FALSE**를 반환합니다.

```
a == "apple" # is equal?
```

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

```
a != "apple" # is not equal?
```

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

```
a == "banana" # is equal?
```

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

입력 확인 (2)

▶ 현재 메모리에 **a**라는 변수에 어떤 값이 들어있는지 확인

▶ `print(a)`도 같은 기능을 함.

▶ `cat(a)`도 거의 같은 기능을 함. (좀 더 단정하게 출력)

```
a # show a
```

```
## [1] "apple"
```

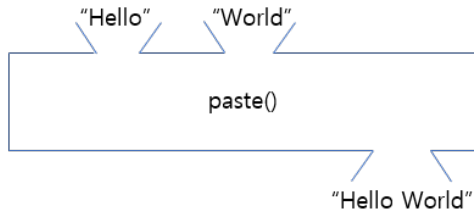
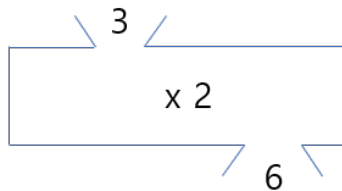
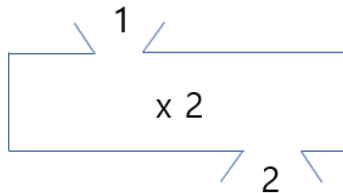
변수의 이름

- ▶ 변수 이름
 - ▶ 영어로 지으세요
 - ▶ 특수 문자는 대부분 사용이 불가능 함. (예외: "_", ".")
 - ▶ 빈칸 없이 지으세요
 - ▶ 길지 않게, 그러나 이해할 수 있게
 1. `learningspoons`: 가독성이 나쁨
 2. `learningSpoons`: Camel 방식
 - ▶ 가독성이 좋음
 - ▶ R에서 가장 추천되는 방식
 3. `learning_spoons`: 전통적인 방식
 - ▶ 대소문자 구분이 없던 시절부터 사용
 - ▶ 아직도 많이 사용되고 있음.
- ‘좋은 프로그래머가 되려면 두 가지를 잘해야 하는데, 하나는 메모리를 잘 관리하는 것이고, 다른 하나는 이름을 잘 짓는 것이다.’
- from [Advanced R]

함수 (Function)

- ▶ 함수는
 1. 입력 (Input)을 가지고
 2. 어떤 행동을 수행하고
 3. 그 결과로서 출력(Output)을 반환(Return)하는 것
- ▶ 함수가 수행하는 행동
 1. 처리 (process)
 2. 생성 (generate, populate)
 3. 변환 (convert)
 4. 표시 (display, print)
 5. 합치기 (aggregate, combine, concatenate, merge)
 6. 추출 (filter)
 7. 저장 (save, write)
 8. 불러오기 (load, infile)

▶ 함수의 도식화



변수(variable)와 함수(function)

1. 변수

- ▶ 존재하는 것
- ▶ 이름은 명사로
- ▶ 데이터
- ▶ ex) **greeting**

2. 함수

- ▶ 행동하는 것
- ▶ 이름은 동사로
- ▶ 데이터를 처리
- ▶ ex) **paste**

3. 데이터 분석 프로그램 대부분은

- ▶ 변수로 시작해서
- ▶ 함수가 계속 작동하면서
- ▶ 결론에 해당하는 변수를 만들어 내는 것

paste 함수와 변형

- ▶ `paste` 함수는 문자열을 합칩니다.

```
paste(greeting, "World")
```

```
## [1] "Hello World"
```

```
paste(greeting, "!")
```

```
## [1] "Hello !"
```

- ▶ `paste0`을 사용하면 띄어쓰기 없이 합쳐집니다.

- ▶ `paste0`은 `paste`의 변형입니다.

- ▶ 어떻게 검색하면 `paste0` 함수를 찾을 수 있을까요?

- ▶ ex) google: 'R paste without blank'

```
paste0(greeting, "!")
```

```
## [1] "Hello!"
```

- ▶ `sep` 옵션을 사용하면 `paste` 함수를 사용하더라도 띄어쓰기를 안 할 수 있습니다.

- ▶ 많은 경우에 이처럼 2가지 이상의 방법이 존재합니다.

- ▶ 익숙한 것을 사용하면서, 경험을 통해서, 타인의 코드를 보면서 발전시켜 나갑니다.

```
paste(greeting, "!", sep = "")
```

```
## [1] "Hello!"
```

II. 데이터 타입 (Data Type)

주민등록번호 Data

The diagram illustrates the structure of a Korean Resident Registration Number (주민등록번호) and how its components map to specific data types in a table. The number is shown as '손흥민 920708-1234567'. Lines connect each part to a column in the table below:

- '손흥민' (Name) connects to the '이름' (Name) column.
- '920708' (Birth Date) connects to the '날짜' (Date) column.
- '1' (Gender) connects to the '분류' (Category) column.
- '2345' (Residence Number) connects to the '분류' (Category) column.
- '6' (Check Digit) connects to the '숫자' (Number) column.
- '7' (Validity Check) connects to the '숫자' (Number) column.

	손흥민	920708	1	2345	6	7
정보	이름	생년월일	1900년대 출생 남성	출생고유지 등록번호	해당 출생지 일련번호	Validity Check
특성	문자	날짜	분류	분류	숫자	숫자
	Character	Date	Category	Category	Number	Number
R Data Type	character	Date	factor	factor	numeric	numeric

- ▶ 하나의 문자열일 것 같은 주민등록번호에도 많은 정보가 담겨 있습니다.
- ▶ Data Type(문자, 날짜, 분류, 숫자)에 따라서 다른 종류의 처리가 다른 함수를 이용해서 가능합니다.
- ▶ Data Types
 1. character (string)
 2. numeric
 3. logical
 4. factor
 5. Date
 6. 기타
- ▶ 각각의 Data Type에 대해서 어떤 연산을 주로 수행하는지 알아보시다.

1. character

```
greeting <- "R says \"Hello World!\""
nchar(greeting) # number of characters

## [1] 21
substr(greeting, 3, 6) # substring from 3 to 6

## [1] "says"
greeting # show

## [1] "R says \"Hello World!\""
cat(greeting) # show cleanly

## R says "Hello World!"
```

- ▶ 따옴표를 입력할때는 backslash를 앞에 붙여줍니다.
- ▶ `substr`은 SUBset of STRing, 즉, 문자열 변수의 부분 집합을 추출합니다.
- ▶ `cat`함수를 사용하면 backslash를 빼고 출력해줍니다.
- ▶ 지금까지 배운 string관련 함수

```
paste0(string1, string2)
paste(string1, string2, sep)
nchar(string)
substr(string, start, end)
cat(string)
```

Strings		Also see the stringr package.	
<code>paste(x, y, sep = ' ')</code>	Join multiple vectors together.		• <i>stringr</i> 패키지는 다른 여러 관련 함수가 있습니다.
<code>paste(x, collapse = ' ')</code>	Join elements of a vector together.		• <i>String</i> 변수들을 합칩니다. (<i>vector</i> 포함)
<code>grep(pattern, x)</code>	Find regular expression matches in x.		• <i>String</i> 으로 된 벡터의 원소 들을 합칩니다. (나중에)
<code>gsub(pattern, replace, x)</code>	Replace matches in x with a string.		• 문자열 <i>x</i> 에 <i>pattern</i> 이라는 문자열이 속해 있는가?
<code>toupper(x)</code>	Convert to uppercase.		• 문자열 <i>x</i> 의 <i>pattern</i> 이라는 문자열을 <i>replace</i> 로 교체
<code>tolower(x)</code>	Convert to lowercase.		• 문자열 <i>x</i> 의 모든 소문자를 대문자로 교체
<code>nchar(x)</code>	Number of characters in a string.		• 문자열 <i>x</i> 의 모든 문자의 개수를 세는 함수

Figure 1: String에 관련된 주요 함수

- ▶ Base Cheatsheet에 소개된 주요 함수입니다.
- ▶ Cheatsheet을 가까이 해주세요.
- ▶ 검색, Cheatsheet, Trial & Error가 여러분의 실력을 올립니다.

2. numeric

▶ 그냥 계산기 처럼 사용할 수 있습니다.

```
10^2 + 36
```

```
## [1] 136
```

```
a <- 4
```

```
a
```

```
## [1] 4
```

```
a*5
```

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

```
a <- a + 10 # assign a+10 to a
```

```
a
```

```
## [1] 14
```

3. logical

▶ 참과 거짓 (TRUE와 FALSE를 나타냅니다.)

▶ TRUE는 1, FALSE는 0에 대응됩니다.

```
2==3
```

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

```
5>3
```

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

```
as.numeric(2==3)
```

```
## [1] 0
```

<code>a == b</code>	Are equal	<code>a > b</code>	Greater than	<code>a >= b</code>	Greater than or equal to	<code>is.na(a)</code>	Is missing
<code>a != b</code>	Not equal	<code>a < b</code>	Less than	<code>a <= b</code>	Less than or equal to	<code>is.null(a)</code>	Is null

Figure 2: logical 값을 반환하는 수치 비교 함수

Data Type의 확인과 변환

- ▶ `is.DATATYPE()` 함수
 - ▶ 해당 DATATYPE이 맞으면 TRUE
 - ▶ 아니면 FALSE값을 반환합니다.
- ▶ `as.DATATYPE()` 함수
 - ▶ DATATYPE으로 변환합니다.

```
is.character(5)
```

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

```
is.character("5")
```

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

```
a <- as.character(5)
```

```
is.character(a)
```

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

```
b <- as.numeric(a)
```

```
is.numeric(b)
```

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

```
as.numeric(2==3)
```

```
## [1] 0
```

- ▶ `class()` 함수
 - ▶ data type을 바로 확인할 수 있습니다.

```
class(5)
```

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

```
class("TRUE")
```

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

```
class(TRUE)
```

```
## [1] "logical"
```

- ▶ 위의 3줄의 코드에서 함수 인수의 색이 다른 것이 보이시나요?
- ▶ 이 노트는 R문법을 이해하고 실행할 수 있는 `rmarkdown`을 이용해서 작성되었습니다. (M21)

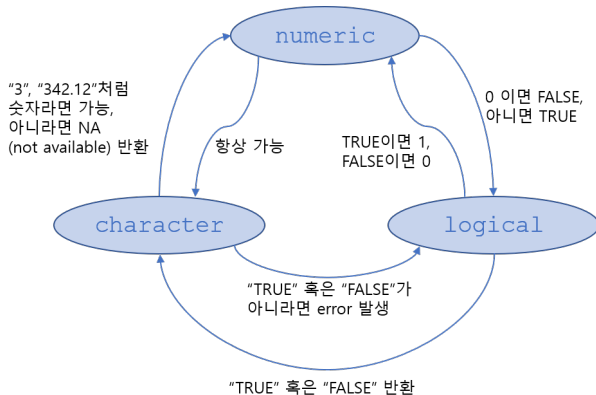


Figure 3: string-numeric-logical간의 변환

- ▶ 함수에 따라서
 - ▶ 입력(input)에 해당하는 인수(argument)의 지정된 type이 있습니다.
 - ▶ 출력(output)값의 지정된 type이 있습니다.
 - ▶ 그렇기에 type을 확인하고 변환할 수 있어야 합니다.

Types		
Converting between common data types in R. Can always go from a higher value in the table to a lower value.		
<code>as.logical</code>	TRUE, FALSE, TRUE	Boolean values (TRUE or FALSE).
<code>as.numeric</code>	1, 0, 1	Integers or floating point numbers.
<code>as.character</code>	'1', '0', '1'	Character strings. Generally preferred to factors.
<code>as.factor</code>	'1', '0', '1', levels: '1', '0'	Character strings with preset levels. Needed for some statistical models.

Figure 4: Base Cheatsheet의 type관련 부분

- ▶ 위에서 아래 방향으로의 언제나 변환이 가능합니다.
- ▶ **logical**
 - ▶ Boolean value라고도 합니다.
- ▶ **numeric**
 - ▶ 정수(integer) 혹은 소수(float)
- ▶ **character**
 - ▶ 문자열, string이라고도 함
- ▶ **factor**
 - ▶ 범주형(Categorical) 문자
 - ▶ 지정된 값만 가능
 - ▶ 통계 모형에 자주 등장

- ▶ 데이터 타입의 개수는 매우 많습니다.
- ▶ 새로운 응용프로그램 (패키지)마다 적합한 타입을 정의하기도 합니다.
- ▶ 마주칠때마다 검색하고 필요한 만큼 알아보고 사용합니다.

4. factor (Categorical, 범주형 변수)

- ▶ 데이터가 소속된 group을 나타내는 변수
 - ▶ 숫자 vs Categorical
 - ▶ 더하고 뺄 수 있다면 numeric
 - ▶ 1,2,3을 A,B,C로 바꿔도 무리가 없다면 Categorical
 - ▶ 문자 vs Categorical
 - ▶ Exclusive (배타적) 집합이고, 각 개체가 1개 그룹에 속한다면 Categorical
 - ▶ Keywords
 - ▶ 'Classification', '분류', '집단', 'Group', '범주'
 - ▶ 예시
 - ▶ 성인남자, 성인여자, 미성년자
 - ▶ 표준 산업 분류 (제조업, 금융업, 광공업)
 - ▶ 날씨 (맑음, 흐림, 비가 올) 따옴
 - ▶ 생성
 - ▶ `data.frame` 생성시에 `stringsAsFactors = TRUE`를 사용하면 모든 문자 객체가 `factor`가 됨
 - ▶ 다른 타입에서 `as.factor()`를 이용해서 변환

5. Date

```
mydates <- as.Date(c("2007-06-22", "2004-02-13"))
mydates[1] - mydates[2]

## Time difference of 1225 days
today <- Sys.Date( ) # today
today

## [1] "2019-04-14"
# year
as.numeric(substr(today, 1, 4))

## [1] 2019
# month
as.numeric(substr(today, 6, 7))

## [1] 4
# day
as.numeric(substr(today, 9, 10))

## [1] 14
# mm-dd-YYYY
format(today, format="%B %d %Y")

## [1] "4 14 2019"
```

- ▶ character인 "2007-06-22"를 as.Date()로 변환하여 Date 변수를 생성
- ▶ c()는 두 개의 data를 combine하여 vector를 만드는 함수
- ▶ Sys.Date()는 오늘 날짜
- ▶ Date 객체도 character처럼 substr() 가능
- ▶ format()함수로 다양한 포맷으로 처리 가능
- ▶ 날짜는 숫자와는 다른 방식으로 더하고 빼죠?
- ▶ R에서는 빨샘, 올림, 버림등의 Date 연산이 가능 (M52 참조)

III. 데이터 구조 (Data Structure)

자료형 (type) vs 자료구조 (structure)

1. 자료형

- ▶ 변수에 입력된 하나의 값의 특성
- ▶ 0차원, 하나의 점, 하나의 값, singleton

2. 자료 구조가 필요한 이유

- ▶ 하나의 값이 하나의 변수가 된다면 → 변수의 갯수가 너무 많아짐
- ▶ 엑셀에서도 A컬럼, 1번행 등으로 묶어서 처리하는 기능을 제공

3. 자료 구조

- ▶ 각각의 값(singleton)들이 모여 있는 구조
- ▶ 대용량 데이터도 한 번에 포함할 수 있기에 데이터 분석에서 중요
- ▶ 엑셀에서 1개의 컬럼, 1개의 네모 블록, 1개의 워크시트, 1개의 파일 모두 자료 구조에 해당

4. 자료 구조의 이해

- 4.1 길다란가? 네모난가? 뽕뽕뽕한다?
- 4.2 몇 개의 관찰값이 있는가?
- 4.3 어떤 규칙을 가지고 있는가?

1. vector

문자 벡터

- ▶ 길다랗게 저장되어 있는 데이터 구조
- ▶ `c()` 함수를 이용해서 벡터를 만들 수 있음
- ▶ `paste` 함수는 `string`으로 된 `vector`에도 적용이 가능!

```
strVec1 <- c("Hello", "Hi", "What's up")
```

```
strVec1
```

```
## [1] "Hello"      "Hi"         "What's up"
```

```
strVec2 <- c("Ma'am", "Sir", "Your Honor")
```

```
strVec3 <- paste(strVec1, strVec2, sep = ", ")
```

```
strVec3
```

```
## [1] "Hello, Ma'am"      "Hi, Sir"           "What's up, Your Honor"
```

숫자 벡터

- ▶ `seq()` 함수는 등차 수열을 만듦
- ▶ `a:b`는 `a`부터 `b`까지의 정수 벡터를 만듦

```
numVec1 <- c(30,50,70)
numVec1

## [1] 30 50 70

numVec2 <- seq(30,70,20)
numVec2

## [1] 30 50 70

numVec3 <- c(25,55,80)
numVec3

## [1] 25 55 80

numVec4 <- seq(from=20, to=1, by=-3)
numVec4

## [1] 20 17 14 11 8 5 2
2:6

## [1] 2 3 4 5 6
```

- ▶ `min` vs `pmin`?

```
min(numVec1) # by all

## [1] 30

min(numVec1, numVec3) # by all

## [1] 25

pmin(numVec1, numVec3) # by element

## [1] 25 50 70

numVec1 > numVec3 # by element

## [1] TRUE FALSE FALSE

▶ subsetting (부분 선택)

numVec1[2]

## [1] 50

numVec1[-2]

## [1] 30 70

numVec1[1:2]

## [1] 30 50

numVec1[c(1,3)]

## [1] 30 70
```

Vectors		
Creating Vectors		
<code>c(2, 4, 6)</code>	2 4 6	Join elements into a vector
<code>2:6</code>	2 3 4 5 6	An integer sequence
<code>seq(2, 3, by=0.5)</code>	2.0 2.5 3.0	A complex sequence
<code>rep(1:2, times=3)</code>	1 2 1 2 1 2	Repeat a vector
<code>rep(1:2, each=3)</code>	1 1 1 2 2 2	Repeat elements of a vector
Vector Functions		
<code>sort(x)</code> Return x sorted.	<code>rev(x)</code> Return x reversed.	
<code>table(x)</code> See counts of values.	<code>unique(x)</code> See unique values.	

- 원소를 나열하여 벡터 생성
- 1간격으로 증가하는 정수로 벡터 생성
- 등차수열의 벡터 생성
- 벡터를 반복
- 벡터의 원소들을 반복
- 정렬 (낮음차순/오름차순); 거꾸로 뒤집기
- 빈도를 표로 표현; 중복되는 값을 제거하여 반환

Figure 5: Base Cheatsheet의 vector관련 부분(1)

Selecting Vector Elements		
By Position		
<code>x[4]</code>	The fourth element.	• 4번째 원소만
<code>x[-4]</code>	All but the fourth.	• 4번째 원소만 빼고
<code>x[2:4]</code>	Elements two to four.	• 2번째 부터 4번째 원소까지
<code>x[-(2:4)]</code>	All elements except two to four.	• 2번째 부터 4번째 원소들을 <u>빼고</u>
<code>x[c(1, 5)]</code>	Elements one and five.	• 1번째와 5번째 원소만
By Value		
<code>x[x == 10]</code>	Elements which are equal to 10.	• 값이 10인 원소만 선택
<code>x[x < 0]</code>	All elements less than zero.	• 0보다 작은 원소만 선택
<code>x[x %in% c(1, 2, 5)]</code>	Elements in the set 1, 2, 5.	• x가 1,2,5중에 하나인 경우만 선택
Named Vectors		
<code>x['apple']</code>	Element with name 'apple'.	• <i>string</i> 의 경우에는 따옴표를 사용할 수 있음

Figure 6: Base Cheatsheet의 vector관련 부분(2)

Maths Functions				
• 자연로그	<code>log(x)</code>	Natural log.	<code>sum(x)</code>	Sum.
• 지수함수	<code>exp(x)</code>	Exponential.	<code>mean(x)</code>	Mean.
• 최대/최소값	<code>max(x)</code>	Largest element.	<code>median(x)</code>	Median.
• 소수점 n 자리까지 반올림	<code>min(x)</code>	Smallest element.	<code>quantile(x)</code>	Percentage quantiles.
• n 개의 유효숫자	<code>round(x, n)</code>	Round to n decimal places.	<code>rank(x)</code>	Rank of elements.
• 상관관계	<code>signif(x, n)</code>	Round to n significant figures.	<code>var(x)</code>	The variance.
	<code>cor(x, y)</code>	Correlation.	<code>sd(x)</code>	The standard deviation.

- 중간값
- 백분위수
- 순위
- 분산
- 표준편차

Figure 7: Base Cheatsheet의 numeric vector 관련 부분

2. matrix (array)

▶ 사각형의 데이터 구조

▶ `matrix()` 또는 `array()` 함수로 생성

- ▶ `data = c(9,2,3,4,5,6)`로 element들을 나열
- ▶ `ncol = 3`으로 3개의 컬럼을 가진 matrix 생성 (Number of COLUMN)
- ▶ `nrow`로도 만들 수 있음 (Number of ROW)

```
mat <- matrix(data = c(9,2,3,4,5,6), ncol = 3)
mat
```

```
##      [,1] [,2] [,3]
## [1,]    9    3    5
## [2,]    2    4    6
```

▶ 비슷한 문법의 subsetting

```
mat[1, 2] # first row, second column
```

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

```
mat[2, ] # second row
```

```
## [1] 2 4 6
```

- ▶ 엑셀이 연상되시나요? 새로운 것을 배우는 만큼 기존에 알고 있는 것과 연관을 많이 시켜보세요.

▶ 연산은 여러가지 방식으로 가능 (원소단위, 행 단위, 열단위)

- ▶ `mean()`은 전체 element들에 대해서 평균을 구함
- ▶ `apply(MATRIX, 2, FUNCTION)`
 - ▶ MATRIX의 각 column에 FUNCTION을 apply (적용)
- ▶ `apply(MATRIX, 1, FUNCTION)`
 - ▶ MATRIX의 각 row에 FUNCTION을 apply (적용)

```
mean(mat)
```

```
## [1] 4.833333
```

```
apply(mat, 2, mean) # colMeans(mat)
```

```
## [1] 5.5 3.5 5.5
```

```
apply(mat, 1, mean) # rowMeans(mat)
```

```
## [1] 5.666667 4.000000
```

▶ `apply()` 함수는 왜 어렵게 느껴질까요?

- ▶ 함수의 input으로 함수가 들어가서 그렇습니다.
- ▶ `apply()`를 편리하게 사용할 수 있다면 이미 중급 사용자!




Matrices			
<pre>m <- matrix(x, nrow = 3, ncol = 3)</pre> <p>Create a matrix from x.</p>			
<ul style="list-style-type: none"> • 2번째 행 • 1번째 열 • 2번째 행의 3번째 원소 	 <pre>m[2,]</pre> <p>- Select a row</p>	<pre>t(m)</pre> <p>Transpose:</p> <pre>m %*% n</pre> <p>Matrix Multiplication:</p> <pre>solve(m, n)</pre> <p>Find x in: $m \cdot x = n$.</p>	<ul style="list-style-type: none"> • 행과 열을 바꿈 • 행렬을 곱함 • $Mx=n$의 방정식을 푸는 x를 찾음
	 <pre>m[, 1]</pre> <p>- Select a column</p>		
	 <pre>m[2, 3]</pre> <p>- Select an element</p>		

Figure 8: Base Cheatsheet의 matrix관련 부분

3. data.frame

- ▶ vector를 모아서 네모나게 만든 것이 data.frame
 - ▶ data.frame() 함수를 이용
 - ▶ date, sky, temp, dust vector가 weather라는 data.frame의 column이 됨
 - ▶ data.frame을 생성할 때는 stringsAsFactors = FALSE 옵션을 넣어줌
 - ▶ (그렇지 않으면 string이 factor로 저장됨)

```
weather <-  
  data.frame(date = c("2017-8-31", "2017-9-1", "2017-9-2"),  
             sky  = c("Sunny", "Cloudy", "Rainy"),  
             temp = c(20, 15, 18),  
             dust = c(24, 50, 23),  
             stringsAsFactors = FALSE)
```

```
weather
```

```
##      date    sky temp dust  
## 1 2017-8-31  Sunny   20   24  
## 2 2017-9-1  Cloudy   15   50  
## 3 2017-9-2  Rainy   18   23
```

- ▶ 이제 정말 엑셀화면이랑 비슷하죠? 어떤 차이점이 있나요?

- ▶ 각 column은 변수에 해당하고 이름도 보존됨
- ▶ `colnames()`로 column의 이름을 확인할 수 있음
- ▶ `weather$sky`와 같이 특정 column만 선택 가능
- ▶ `weather[,2]`와 같이 matrix의 subsetting 방법도 적용 가능

```
colnames(weather)
```

```
## [1] "date" "sky"  "temp" "dust"
```

```
weather$sky
```

```
## [1] "Sunny" "Cloudy" "Rainy"
```

```
weather$sky==weather[,2]
```

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

- ▶ 엑셀에서는 A열, B열, 2행, 3행 이렇게 column을 정의합니다.
- ▶ R의 `colnames()`과 `rownames()`는 2차원 데이터 구조에 index를 부여합니다.

- ▶ `class()`로 data structure도 확인 가능!
- ▶ `class(VECTOR)`의 경우에는 element들의 type 확인!
- ▶ `sapply()`를 `data.frame`에 적용하면 각 column에 같은 함수를 적용

```
class(weather)
```

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

```
class(weather$date)
```

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

```
sapply(weather, class)
```

```
##           date           sky           temp           dust
## "character" "character"  "numeric"    "numeric"
```

- ▶ `date` 벡터의 type이 `character`이므로 `Date`로 변환
- ▶ `str()`함수는 데이터의 구조를 보여주므로 자주 사용

```
weather$date <- as.Date(weather$date)
str(weather)
```

```
## 'data.frame':    3 obs. of  4 variables:
## $ date: Date, format: "2017-08-31" "2017-08-31" "2017-08-31"
## $ sky : chr  "Sunny" "Cloudy" "Rainy"
## $ temp: num  20 15 18
## $ dust: num  24 50 23
```

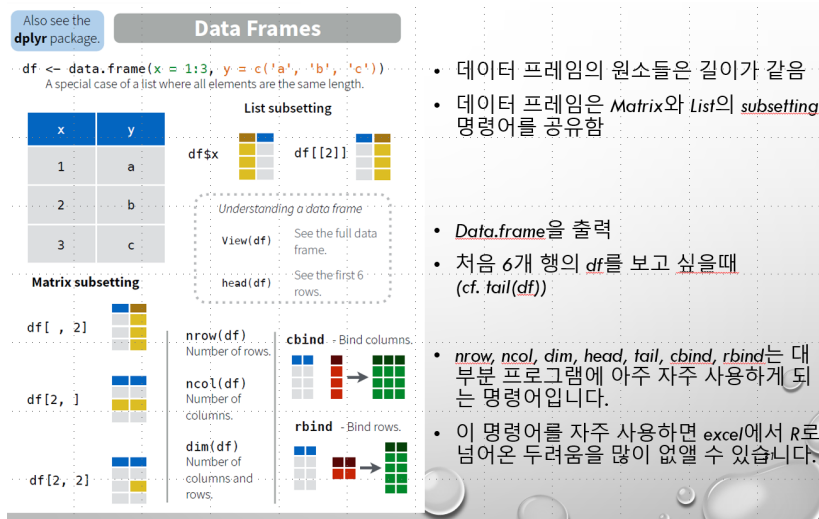


Figure 9: Base Cheatsheet의 data.frame관련 부분

4. list

```
HMSon <-
  list(team = c("Korea", "Tottenham"),
        birth = as.Date("1992-07-08"),
        goals =
          data.frame(team = c("U-17", "U-23", "A"),
                     goals = c(4, 2, 7),
                     stringsAsFactors = FALSE))
```

```
HMSon
```

```
## $team
## [1] "Korea"      "Tottenham"
##
## $birth
## [1] "1992-07-08"
##
## $goals
##   team goals
## 1 U-17     4
## 2 U-23     2
## 3   A      7
```

- ▶ list()로 다양한 데이터 구조를 함께 묶을 수 있음
- ▶ 사용하기 어렵지만, 때로는 유용함
- ▶ str()로 tree형 구조를 파악할 수 있음

```
str(HMSon)
```

```
## List of 3
## $ team : chr [1:2] "Korea" "Tottenham"
## $ birth: Date[1:1], format: "1992-07-08"
## $ goals:'data.frame':  3 obs. of  2 varia
## ..$ team : chr [1:3] "U-17" "U-23" "A"
## ..$ goals: num [1:3] 4 2 7
```

- ▶ 이런 데이터 구조가 아근의 주범

- ▶ `names()`로 level-1 객체(object)의 이름 파악
- ▶ `sapply()`로 level-1 객체에 동시에 함수 적용
- ▶ `[]`로 subsetting 하면 여전히 list
- ▶ `[][]`로 subsetting 하면 level-1 객체의 class 파악 가능

```
names(HMSon)
```

```
## [1] "team" "birth" "goals"
```

```
sapply(HMSon, class)
```

```
##           team           birth           goals
## "character"       "Date" "data.frame"
HMSon[3]
```

```
## $goals
##   team goals
## 1 U-17     4
## 2 U-23     2
## 3   A      7
```

```
class(HMSon[3])
```

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

```
HMSon[[3]]
```

```
##   team goals
## 1 U-17     4
## 2 U-23     2
## 3   A      7
```

```
HMSon[["goals"]]
```

```
##   team goals
## 1 U-17     4
## 2 U-23     2
## 3   A      7
```

```
class(HMSon[[3]])
```

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

```
HMSon[[3]]$team
```

```
## [1] "U-17" "U-23" "A"
```

```
HMSon$team
```

```
## [1] "Korea" "Tottenham"
```

- ▶ `sapply()`: 'simple' apply

- ▶ 바로 하위 구조에 함수를 apply합니다.

- ▶ `data.frame` → column에 적용
- ▶ `list` → level-1 객체에 적용
- ▶ `vector` → 각 원소에 적용

Lists

```
l <- list(x = 1:5, y = c('a', 'b'))
```

A list is a collection of elements which can be of different types.

<code>l[[2]]</code>	<code>l[1]</code>	<code>l\$x</code>	<code>l['y']</code>
Second element of l.	New list with only the first element.	Element named x.	New list with only element named y.

- l의 두번째 *element* (y가 벡터로 반환됨)
- l의 첫번째 *element* (x가 list로 반환됨)
- l\$x 원소중 x라는 이름을 가진 것을 반환 (x가 vector로 반환됨)
- l중에서 이름이 y인 것을 반환 (y가 list로 반환됨)

- List의 *element*는 서로 다른 *type*과 길이일 수 있습니다.

Figure 10: Base Cheatsheet의 list관련 부분

자료 구조 Summary

- ▶ Dimension: 점(0D), 선(1D), 면(2D)
- ▶ Homogeneous (동질성)
 - ▶ 구성 요소의 길이와 type이 같은가?
- ▶ Heterogenous (이질성)
 - ▶ 동질적이지 않으면 이질적

Dimension	Homogenous	Heterogenous
0D	element	X
1D	vector	list
$\geq 2D$	array	data.frame

IV. 프로그램 제어 (Control Statement)

For Loop

```
for (variable in sequence){  
    Do something  
}
```

Example

```
for (i in 1:4){  
    j <- i + 10  
    print(j)  
}
```

While Loop

```
while (condition){  
    Do something  
}
```

Example

```
while (i < 5){  
    print(i)  
    i <- i + 1  
}
```

If Statements

```
if (condition){  
    Do something  
} else {  
    Do something different  
}
```

Example

```
if (i > 3){  
    print('Yes')  
} else {  
    print('No')  
}
```

Functions

```
function_name <- function(var){  
    Do something  
    return(new_variable)  
}
```

Example

```
square <- function(x){  
    squared <- x*x  
    return(squared)  
}
```

- `paste0` 함수, `print` 함수, `for` 문을 활용하여 트리를 그리는 함수를 작성해 보세요.
- `paste0` 함수의 `collapse` 옵션을 사용하세요.

```
function <- tree(h) {
```

```
... ↓
```

```
... ↓
```

```
... ↓
```

```
... ↓
```

```
... ↓
```

```
} ↓
```

```
tree(4) ↓
```

```
## [1] "    A    " ↓
```

```
## [1] "   AAA   " ↓
```

```
## [1] "  AAAAA  " ↓
```

```
## [1] "AAAAAAA" ↓
```

```
tree(6) ↓
```

```
## [1] "      A      " ↓
```

```
## [1] "     AAA     " ↓
```

```
## [1] "    AAAAA    " ↓
```

```
## [1] "   AAAAAA   " ↓
```

```
## [1] "  AAAAAAAA  " ↓
```

```
## [1] " AAAAAAAAAA " ↓
```

Figure 12: For문 실습

Base R Cheat Sheet

Getting Help

Accessing the help files

?mean

Get help of a particular function.

help.search('weighted mean')

Search the help files for a word or phrase.

help(package = 'dplyr')

Find help for a package.

More about an object

str(iris)

Get a summary of an object's structure.

class(iris)

Find the class an object belongs to.

Using Packages

install.packages('dplyr')

Download and install a package from CRAN.

library(dplyr)

Load the package into the session, making all its functions available to use.

dplyr::select

Use a particular function from a package.

data(iris)

Load a built-in dataset into the environment.

Working Directory

getwd()

Find the current working directory (where inputs are found and outputs are sent).

setwd('C:/file/path')

Change the current working directory.

Use projects in RStudio to set the working directory to the folder you are working in.

Vectors

Creating Vectors

c(2, 4, 6)	2 4 6	Join elements into a vector
2:6	2 3 4 5 6	An integer sequence
seq(2, 3, by=0.5)	2.0 2.5 3.0	A complex sequence
rep(1:2, times=3)	1 2 1 2 1 2	Repeat a vector
rep(1:2, each=3)	1 1 1 2 2 2	Repeat elements of a vector

Vector Functions

sort(x)	rev(x)
Return x sorted.	Return x reversed.
table(x)	unique(x)
See counts of values.	See unique values.

Selecting Vector Elements

By Position

x[4]	The fourth element.
x[-4]	All but the fourth.
x[2:4]	Elements two to four.
x[-(2:4)]	All elements except two to four.
x[c(1, 5)]	Elements one and five.

By Value

x[x == 10]	Elements which are equal to 10.
x[x < 0]	All elements less than zero.
x[x %in% c(1, 2, 5)]	Elements in the set 1, 2, 5.

Named Vectors

x['apple']	Element with name 'apple'.
-------------------	----------------------------

Programming

For Loop

```
for (variable in sequence){
  Do something
}
```

Example

```
for (i in 1:4){
  j <- 1 + 10
  print(j)
}
```

While Loop

```
while (condition){
  Do something
}
```

Example

```
while (i < 5){
  print(i)
  i <- i + 1
}
```

If Statements

```
if (condition){
  Do something
} else {
  Do something different
}
```

Example

```
if (i > 3){
  print('Yes')
} else {
  print('No')
}
```

Functions

```
function_name <- function(var){
  Do something
  return(new_variable)
}
```

Example

```
square <- function(x){
  squared <- x*x
  return(squared)
}
```

Reading and Writing Data

Also see the **readr** package.

Input	Output	Description
df <- read.table('file.txt')	write.table(df, 'file.txt')	Read and write a delimited text file.
df <- read.csv('file.csv')	write.csv(df, 'file.csv')	Read and write a comma separated value file. This is a special case of read/table/write/table.
load('file.Rdata')	save(df, file = 'file.Rdata')	Read and write an R data file, a file type special for R.

Conditions	a == b	Are equal	a > b	Greater than	a >= b	Greater than or equal to	is.na(a)	Is missing
	a != b	Not equal	a < b	Less than	a <= b	Less than or equal to	is.null(a)	Is null

IV. 프로그램 제어 (Control Statement)

Types

Converting between common data types in R. Can always go from a higher value in the table to a lower value.

as.logical	TRUE, FALSE, TRUE	Boolean values (TRUE or FALSE).
as.numeric	1, 0, 1	Integers or floating point numbers.
as.character	'1', '0', '1'	Character strings. Generally preferred to factors.
as.factor	'1', '0', '1', Levels: '1', '0'	Character strings with preset levels. Needed for some statistical models.

Maths Functions

log(x)	Natural log.	sum(x)	Sum.
exp(x)	Exponential.	mean(x)	Mean.
max(x)	Largest element.	median(x)	Median.
min(x)	Smallest element.	quantile(x)	Percentage quantiles.
round(x, n)	Round to n decimal places.	rank(x)	Rank of elements.
signif(x, n)	Round to n significant figures.	var(x)	The variance.
cor(x, y)	Correlation.	sd(x)	The standard deviation.

Variable Assignment

```
> a <- 'apple'
> a
[1] 'apple'
```







The Environment

ls()	List all variables in the environment.
rm(x)	Remove x from the environment.
rm(list = ls())	Remove all variables from the environment.

You can use the environment panel in RStudio to browse variables in your environment.

Matrices

```
m <- matrix(x, nrow = 3, ncol = 3)
# Create a matrix from x.
```

 m[2,]	- Select a row	 t(m)	Transpose
 m[, 1]	- Select a column	 m %*% n	Matrix Multiplication
 m[2, 3]	- Select an element	 solve(m, n)	Find x in: m * x = n

Lists

```
l <- list(x = 1:5, y = c('a', 'b'))
# A list is a collection of elements which can be of different types.
```

 l[[2]]	Second element of l	 l[1]	New list with only the first element.
		 l\$x	Element named x.
		 l['y']	New list with only element named y.

Also see the **dplyr** package.

Data Frames

```
df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))
# A special case of a list where all elements are the same length.
```

x	y
1	a
2	b
3	c

Matrix subsetting

 df[, 2]	
 df[2, 1]	
 df[2, 2]	

nrow(df)
Number of rows.

ncol(df)
Number of columns.

din(df)
Number of columns and rows.

cbind - Bind columns.

rbind - Bind rows.








Understanding a data frame

View(df) See the full data frame.



head(df) See the first 6 rows.

Strings







Also see the **stringr** package.

 paste(x, y, sep = ' ')	Join multiple vectors together.
 paste(x, collapse = ' ')	Join elements of a vector together.
 grep(pattern, x)	Find regular expression matches in x.
 gsub(pattern, replace, x)	Replace matches in x with a string.
 toupper(x)	Convert to uppercase.
 tolower(x)	Convert to lowercase.
 nchar(x)	Number of characters in a string.

Factors

 factor(x)	Turn a vector into a factor. Can set the levels of the factor and the order.
 cut(x, breaks = 4)	Turn a numeric vector into a factor by 'cutting' into sections.

Statistics




 lm(y ~ x, data=df)	Linear model.	 t.test(x, y)	Perform a t-test for difference between means.	 prop.test	Test for a difference between proportions.
 glm(y ~ x, data=df)	Generalised linear model.	 summary	Get more detailed information out a model.	 pairwise.t.test	Perform a t-test for paired data.
				 aov	Analysis of variance.

Distributions

	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	rnorm	dnorm	pnorm	qnorm
Poisson	rpois	dpois	ppois	qpois
Binomial	rbinom	dbinom	pbinom	qbinom
Uniform	runif	dunif	punif	qunif

Plotting

Also see the **ggplot2** package.

 plot(x)	Values of x in order.	 plot(x, y)	Values of x against y.	 hist(x)	Histogram of x.
---	-----------------------	---	------------------------	---	-----------------

Dates

See the **lubridate** package.

Appendix A. Thousand commas & DT & `data.table`

Background

- ▶ 자리수(digit)가 긴 숫자의 표현에서 1000 단위마다 쉼표를 찍는 경우가 있습니다.
- ▶ 이를 thousand comma라고 합니다.
- ▶ Appendix A.에서는 thousand comma를 넣거나 빼는 방법을 정리하고
- ▶ `data.frame`의 고급 기능을 제공하는 패키지 DT와 `data.table`을 다룹니다.
- ▶ (미처 배우지 못한 내용이 있지만) `data.frame`과 밀접한 연관이 있기에 M12에 포함합니다.

Acknowledgement

- ▶ 관련 질문을 주셨던 LS 4기 수강생 분들께 감사드립니다.
- ▶ google: 'thousand comma data.table r'로 검색해서 아래의 링크를 얻어서 정리했습니다.
- ▶ <https://stackoverflow.com/questions/29242011/add-comma-to-numbers-every-three-digits-in-datatable-r>

1. 예시로 사용할 `data.frame` 만들기

- ▶ `runif()` 함수는 난수를 생성합니다.
 - ▶ `n`, `min`, `max` 인수가 이해되시나요?
 - ▶ `min`과 `max` 사이에서 `n`개의 난수를 생성!
- ▶ `set.seed()`를 사용하면 매번 같은 난수가 생성됩니다.

```
set.seed(123)
table1 <- data.frame(
  money = runif(n = 5, min = 10000, max = 100000),
  percent = runif(n = 5, min = 0, max = 1),
  stringsAsFactors = FALSE
)
```

```
class(table1)
## [1] "data.frame"
table1
##      money  percent
## 1 35881.98 0.0455565
## 2 80947.46 0.5281055
## 3 46807.92 0.8924190
## 4 89471.57 0.5514350
## 5 94642.06 0.4566147
```

2. base 패키지에 내장된 `format()` 함수를 사용해서 thousand commas를 만들수 있습니다.

```
library(dplyr)
table2 <- table1 %>%
  mutate(money =
    formatC(money,
             format = "f",
             big.mark = ",","))
```

table2

```
##           money  percent
## 1 35,881.9768 0.0455565
## 2 80,947.4622 0.5281055
## 3 46,807.9230 0.8924190
## 4 89,471.5664 0.5514350
## 5 94,642.0556 0.4566147
```

```
sapply(table2, class)
```

```
##           money      percent
## "character" "numeric"
```

- ▶ `money` 변수가 string으로 바뀌면서 thousand comma가 생겼습니다.
- ▶ `money` 변수를 numeric으로 보존하면서 작업중에 thousand comma를 보려면 `rmarkdown` 환경의 R chunk 부분에서 실행하면서 작업하면 됩니다.

3. DT 패키지의 `datatable()`을 이용해서 thousand comma와 함께 표현합니다.

```
library(DT)
a <- datatable(table1) %>%
  formatCurrency(
    "money", currency = "", interval = 3,
    mark = ",", digits = 0) %>%
  formatPercentage("percent", 2)
a
```

```
class(a)
## [1] "datatables" "htmlwidget"
```

- ▶ M3X Web-application 참조
- ▶ `htmlwidget` 객체이므로 웹문서에 삽입할 수 있습니다!
- ▶ `html`, `flexdashboard`, `shiny`에 사용하세요.

Show 10 entries

Search:

	money	percent
1	35,882	4.56%
2	80,947	52.81%
3	46,808	89.24%
4	89,472	55.14%
5	94,642	45.66%

Showing 1 to 5 of 5 entries

Previous

1

Next

Figure 13: `htmlwidget` 객체인 `a`

4. `data.table` is faster than `data.frame`!

```
library(data.table)
table3 <- data.table(table1)
table3

##      money    percent
## 1: 35881.98 0.0455565
## 2: 80947.46 0.5281055
## 3: 46807.92 0.8924190
## 4: 89471.57 0.5514350
## 5: 94642.06 0.4566147
class(table3)

## [1] "data.table" "data.frame"
```

- ▶ `data.table`은 `data.frame`의 상속 클래스(inherited class)입니다.
 - ▶ 즉, `data.table` 객체는 동시에 `data.frame` 객체입니다.
 - ▶ 즉, `data.table` 객체는 `data.frame`의 특성과 명령어를 가지고 있습니다.
- ▶ `data.table`은 기가바이트 단위의 데이터셋도 빠르게 처리합니다.
- ▶ google: 'vignette data.table'을 해보세요. (읽기 쉬운 설명서)

5. thousand commas 제거하기

```
sapply(table2, class)
```

```
##      money      percent
## "character" "numeric"
table2
```

```
##      money      percent
## 1 35,881.9768 0.0455565
## 2 80,947.4622 0.5281055
## 3 46,807.9230 0.8924190
## 4 89,471.5664 0.5514350
## 5 94,642.0556 0.4566147
```

```
library(stringr)
table4 <- table2 %>%
  mutate(
    money =
      str_replace(money, ",", "") %>%
      as.numeric())
sapply(table4, class)
```

```
##      money      percent
## "numeric" "numeric"
table4
##      money      percent
## 1 35881.98 0.0455565
## 2 80947.46 0.5281055
## 3 46807.92 0.8924190
## 4 89471.57 0.5514350
## 5 94642.06 0.4566147
```

- ▶ `str_replace()` 함수로 ",", "를 ""로 바꾸고 `as.numeric()`을 사용합니다.
- ▶ `stringr` 패키지는 여러가지 string 관련 함수를 제공합니다.

FAQ

1. 프롬프트에 명령어를 입력하고 실행할 때, 가장 앞에 나오는 [1]은 무엇인가요?
 - ▶ [1]은 프롬프트의 결과 중에서 1번째 값을 화면에 출력하겠다는 얘기입니다. 그 이후로 한 칸씩 띄우면서 결과를 보여줍니다. 결과의 길이를 확인하려면 `dim()`, `length()`와 같은 함수를 사용하는 것을 권장합니다.
2. 'paste 함수와 변형' 슬라이드에서 'sep 옵션을 사용하면 paste함수로도 띄어쓰기를 안 할 수 있습니다.' 라고 하신 부분과 관련하여, `paste(a,b)`와 `paste(a,b,sep=' ')`은 동일한 결과를 도출하는 것이 맞나요? 맞다면, 단순 띄어쓰기는 `paste`함수를 사용하고, `sep`는 띄어쓰기가 아닌 다른 표시를 넣고 싶을 때만 사용하는 것이 경제적인 것 같은데 굳이 `sep=" "`를 사용하는 이유(혹은 경우)가 있나요?
 - ▶ 맞습니다. 동일한 결과를 만들어 냅니다. 말씀하신 대로 `paste(a, b)`를 안 쓰고 `paste(a, b, sep = ' ')`를 사용할 이유는 없습니다.
 - ▶ `sep = ' '`을 넣지 않으면 `paste`함수에서 `sep`이라는 인수의 default 값이 ' ' 이므로 동일한 결과가 나옵니다.
 - ▶ `sep` 옵션은 주로 `paste("2009", "01", "04", sep = "-")`와 같은 명령으로 "2009-01-04"라는 string 객체를 만들어 내기 위해서 사용합니다. (그리고 `as.Date()`로 Date객체를 만들 수 있겠죠?)

‘몰라서 묻는 것이 부끄러운 것이 아니라 모르면서도 알려고 하지 않는
것이 더 부끄러운 것이다.’
- 공자