




# 1주차 : R 설치, R과 친해지기

Created	@Oct 30, 2020
Created by	 Eunhee Kim
Files	note_01.pdf

## 안녕하세요!

😊 안녕하세요, <R을 활용한 통계분석 기초>의 실습세션 진행을 맡은 김은희입니다.

저는 경희대학교에서 사회학과 응용수학을 공부하고 있습니다.

- 실습은 Computer Science적인 내용보다는 이미 알고 계신 통계지식을 R을 통해 실제 데이터에 적용하는 방식으로 진행합니다.
  - 코딩을 해본 적 없어도 괜찮습니다.
  - 수학적으로 엄밀하게 증명을 하는 부분은 없습니다.
  - 다만, 약간의 수학은 들어갑니다. (약간의 행렬, 약간의 미적분, 약간의 확률과 통계)
- 모든 내용을 암기할 필요가 없습니다. 쓰다보면 익숙해지고, 잘 모르는 함수는 그때그때 알아가면 됩니다.
- 강의 중 따라해보며 진행할 부분을 함께 진행하며, (시간적 여유가 된다면) 부담없는 퀴즈를 통해 수업 내용을 직접 적용해보는 시간을 갖습니다.
- 50분 실습 후 10분 휴식하는 것을 두번 반복합니다.

실습 진행에 대해 의견이 있으시다면 편하게 말씀해 주세요 :)

## 실습 환경

- Microsoft Windows 10 (64-bit)

- CPU : Intel i5-10400
- RAM : 8GB

이 실습에는 '좋은' PC가 필요하지 않습니다.

## R 설치

### The R Project for Statistical Computing

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To , please choose your

 <https://www.r-project.org/>



기본 옵션대로 설치하면 i386 4.0.3이랑 x64 4.0.3 두 버전이 설치됩니다.



R i386은 32비트, x64는 64비트 운영체제를 위한 것입니다. 사용하고 계신 Windows의 버전을 확인하여 이용하시면 됩니다. (검색 > PC 정보 > 장치 사양)

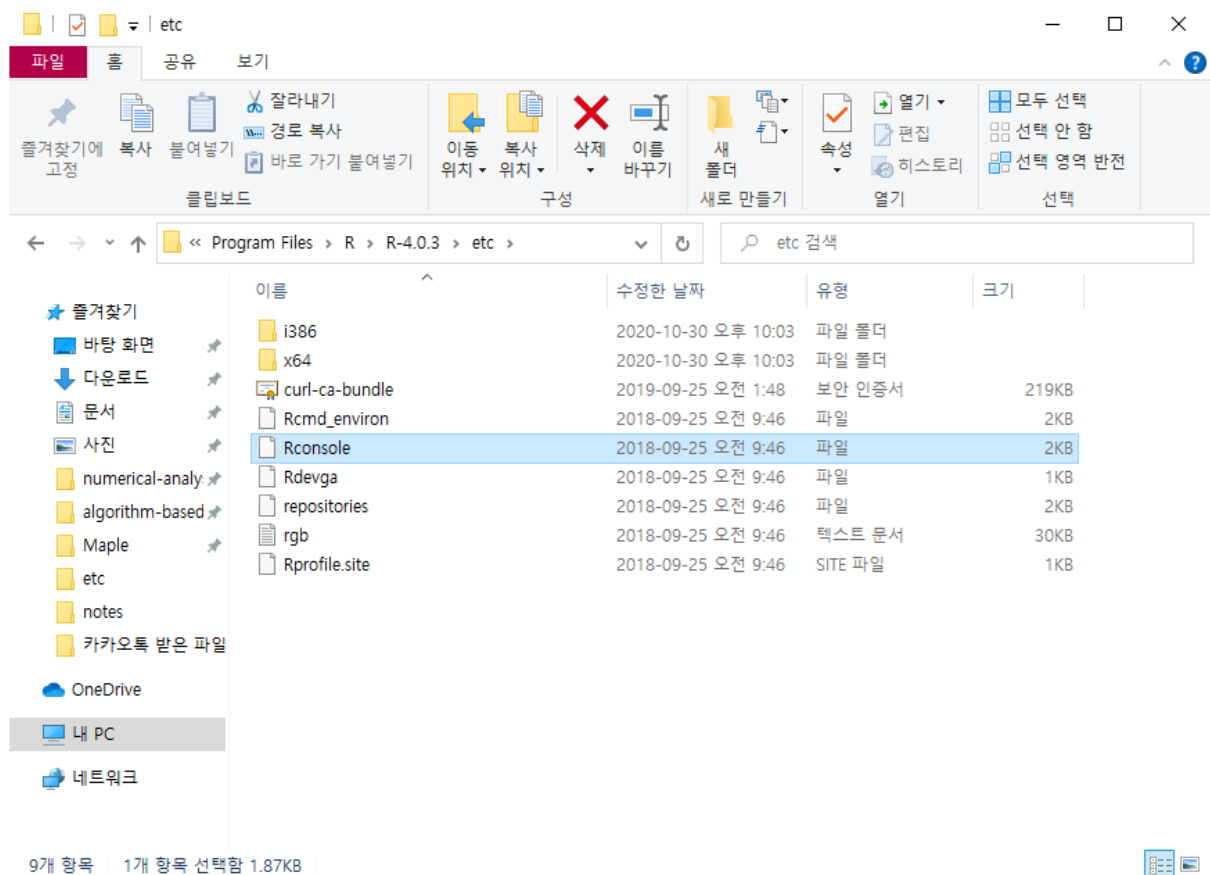
### 사용언어를 영어로 바꾸기

- [Edit] → [GUI Preferences..] → [Language for menus and messages]에서 [en]으로 바꾼 후 [Save] → [OK]
- [C:\Program Files\R\R-4.0.3\etc]로 이동 후 [Rconsole] 파일을 워드패드 등 편집기에서 열고 다음 부분을 찾아 수정 후 저장

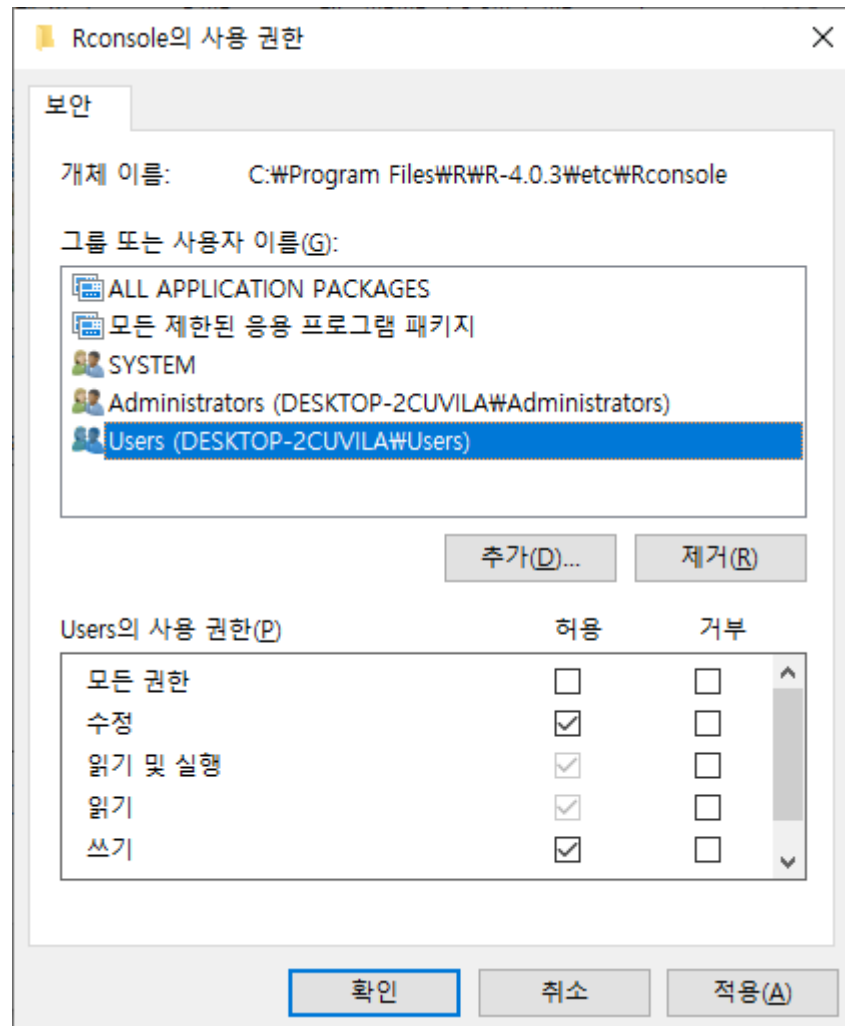
```
## Language for messages
language = en
```

## 관리자 권한이 없어 Rconsole 파일을 수정할 수 없는 경우

관리자 권한이 없는 상태에서 Rconsole 파일을 수정하려고 하면 '액세스가 거부되었습니다' 팝업 메시지가 뜹니다.



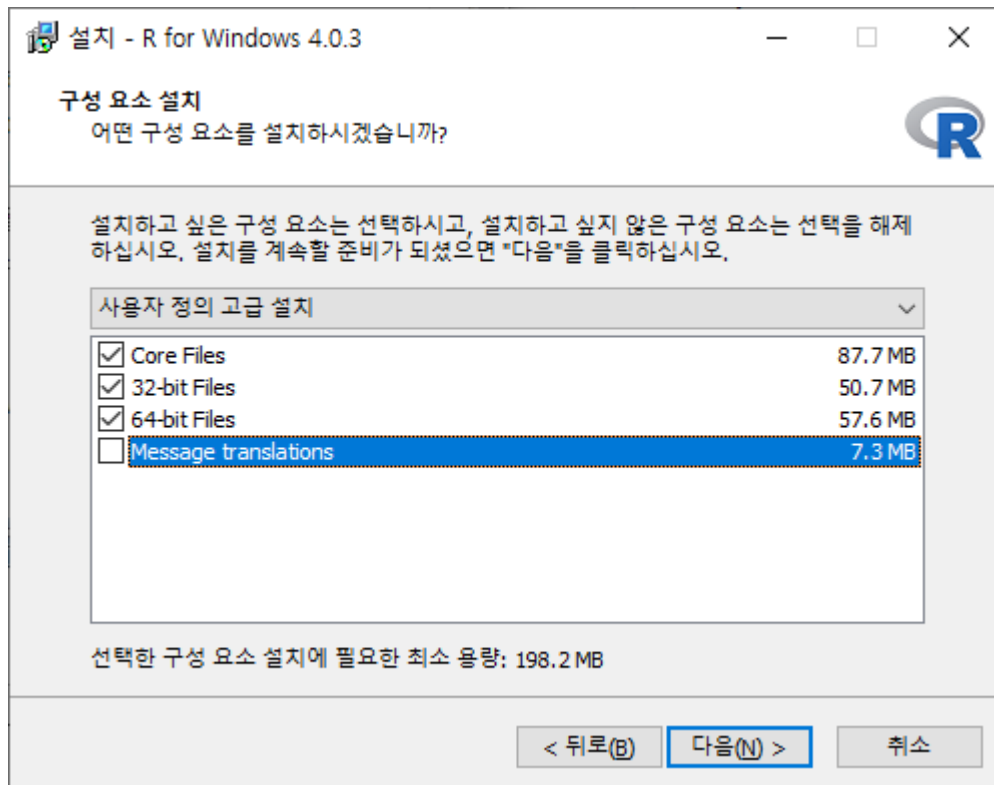
Rconsole 파일을 마우스 우클릭 하여 [속성] → [보안] → [사용 권한 편집] 탭에 접근해 주세요.



현재 사용하고 있는 계정(제 경우에는 Users입니다) 의 사용 권한 중 [수정] 허용 체크박스를 클릭하시면 Rconsole 파일을 수정할 수 있습니다.

## 그래도 메시지가 한글로 출력되는 경우

이렇게 하면 영어로 메시지가 출력되는데요, 그래도 문제가 해결되지 않다면 R을 삭제 후 재설치하는 과정에서 [Message translations]의 체크를 해제하시면 됩니다.



## IDE를 이용하는 경우

이 수업에서는 별도의 IDE(R studio가 대표적입니다)를 사용하지 않습니다. 그러나 IDE를 이용하시는 경우, 출력 메시지를 영어로 설정하기 위해서는 추가적인 설정이 필요합니다.

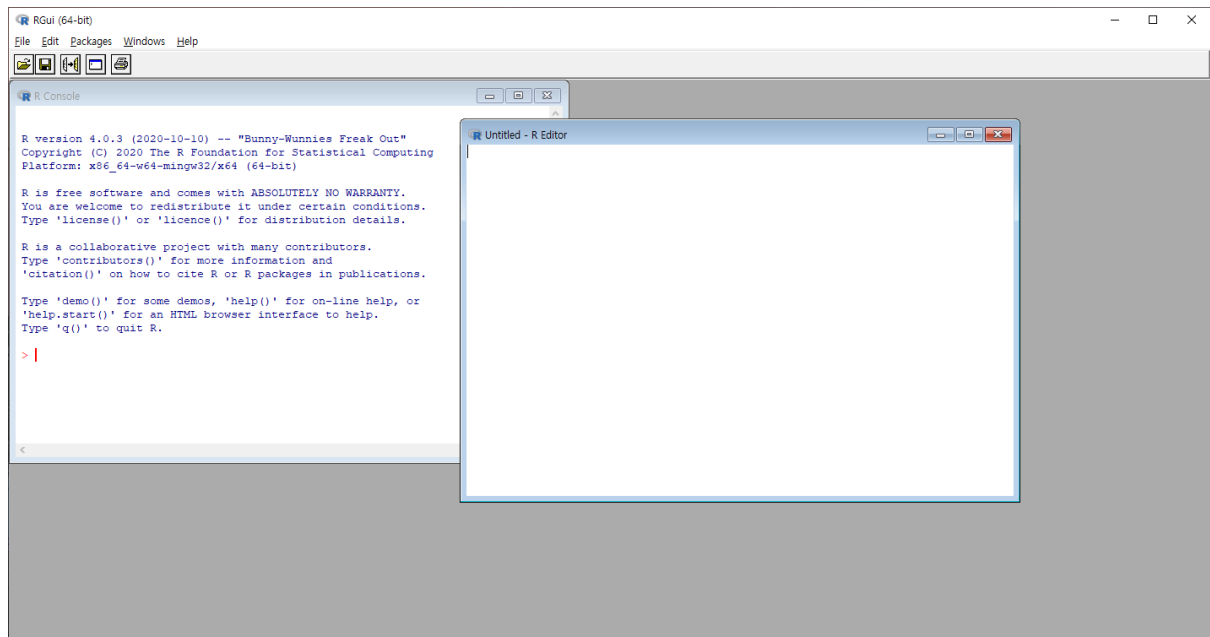
`Sys.getlocale()` 을 콘솔에 입력하면 언어가 한국어로 설정되어 있을 거예요.

`Sys.setlocale("LC_ALL", "English")` 명령어를 사용하면 영어로 출력됩니다.

## R Editor 이용하기

R을 실행하면 나오는 R Console은 여러분의 코드를 실행해줄 뿐 저장하지 않습니다. Console에 긴 코드를 작성했는데, 저장이 되지 않다면 낭패겠죠?

[File] → [New Script]를 클릭하여 script를 만들 수 있습니다. 이것은 저장해 재사용하는 것이 가능해요.



저는 간단한 계산이나 실험은 console에, 그 외에 (거의) 모든 작업은 script에 작성합니다.

## 사칙연산

기본적인 사칙연산부터 시작해 볼까요?  $+$ ,  $-$ ,  $*$ ,  $/$  을 사용하여 간단한 사칙연산을 할 수 있습니다.

- Script에서 [Ctrl] + [R] 을 눌러 코드를 한 줄씩 실행할 수 있습니다.
- Script에서 원하는 부분을 선택하여 [Ctrl] + [R]을 눌러 코드를 동시에 실행할 수 있습니다.
- Console에서 [enter]를 눌러 코드를 한 줄씩 실행할 수 있습니다.
- Console에서 [Ctrl] + [L]을 눌러 내용을 지울 수 있습니다. (그래도 내용이 날아가는 건 아니에요!)

R에서 기본적인 연산이 잘 작동합니다. 이 외에도 R에는 다양한 산술 연산자가 있습니다.

## R의 산술 연산자

Aa 산술 연산자	≡ 기능
$\pm$	
$=$	
$*$	
$/$	
<u><math>^</math> 또는 <math>**</math></u>	n제곱
<u><math>\% \%</math></u>	나머지
<u><math>\% / \%</math></u>	몫
<u><math>\text{sqrt}</math></u>	제곱근

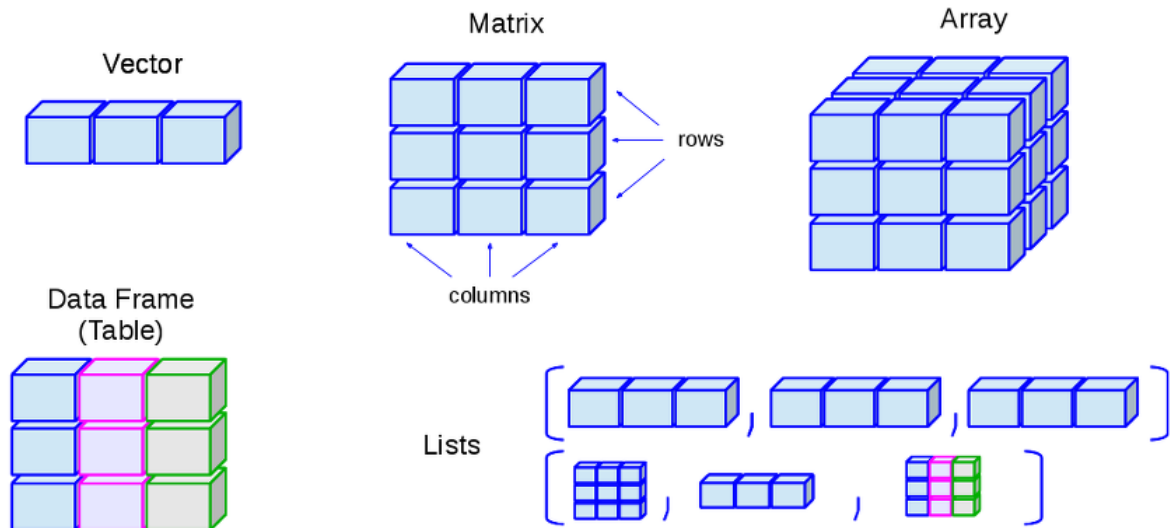
- `2**5` 혹은 `2^5`
- `7%%3`
- `7%/%3`
- `sqrt(9)`

예상한 값이 잘 출력되나요?

## Object

R에서 모든 것은 object예요. R이 다루는 **가장 기본 단위**라고 이해하시면 좋습니다.

- object에 `<-` 기호를 사용하여 value를 할당할 수 있습니다.



source : <https://ryuhyun.tistory.com/14>

잠깐 Element를 소개한 다음 다시 Object 이야기로 되돌아올게요.

# Element

## element의 종류

- logical : TRUE와 FALSE의 값만 가지는 요소 (간단히 T와 F로 쓸 수도 있습니다)
- integer : 정수
- double : 유리수
- complex : 복소수
- character : 문자열
- raw : 기타

`typeof()` 명령어로 element type을 확인할 수 있습니다.

integer → character로 갈수록 많은 정보를 포함하고 있습니다. 정보가 적은 element와 많은 element가 동시에 있다면, 강제로 정보가 많은 element로 변합니다. 이를 강제 변환(coercion)이라고 합니다.



다시 Object 이야기로 되돌아올까요?

## Object의 종류



핵심은 vector

### ■ Vector

Vector



: 동일한 **element**를 가진 요소로 이루어진 **1차원 object**입니다. `c()` 명령어로 벡터를 만들 수 있습니다.

```
vec_lo <- c(TRUE, FALSE, TRUE, TRUE) ## logical vector
vec_in <- c(1L, 2L, 3L, 4L) ## integer vector
vec_do <- c(1, 2, 3, 4) ## double vector
vec_ch <- c("1", "2", "3", "4") ## character vector
```

아까 정보와 적은 element와 많은 element가 동시에 있다면, 많은 element로 바뀐다고 말씀드렸습니다. 예를 들어

```
cb1 <- c(1,2,3,"4")
typeof(cb1)

> typeof(cb1)
[1] "character"
```

double과 character를 같은 벡터에 넣었더니, 더 많은 정보를 가지고 있는 character로 변했습니다.

vector에 서로 다른 element를 넣으면, 더 많은 정보를 갖고 있는 걸로 강제변환 (coercion) 해서 결과적으로 '동일한 특성을 가진 요소'로 이루어지도록 합니다.

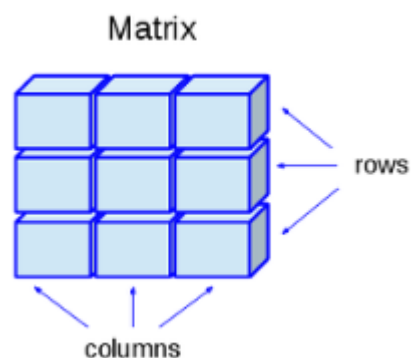
그럼 vector끼리 새로운 vector를 만들 수 있을까요?

```
cb2 <- c(vec_lo, vec_in)

> cb2
[1] 1 0 1 1 1 2 3 4
```

vector가 잘 만들어집니다. `vec_lo`의 `TRUE`와 `FALSE` 값이 각각 `1`과 `0`으로 바뀌었네요.

## ■ Matrix



: 두개 이상의 벡터로 만들어진 행렬로, 행과 열로 이루어진 **2D 데이터 형태**입니다.

`matrix()` 명령어로 행렬을 만들 수 있습니다. 이 때에는 input data와 함께 **column**과 **row**의 수를 함께 지정해 주어야 합니다.

```
basic_mt <- matrix(1:12, ncol = 3, nrow = 4)

> basic_mt
      [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

이미 존재하는 벡터들로 행렬을 만들기 위해서는

- `cbind()` : 열을 기준으로
- `rbind()` : 행을 기준으로

두 명령어를 사용할 수 있습니다.

object의 종류를 말씀드릴 때, '핵심은 벡터' 라고 말씀드렸죠?

vector가 동일한 특성을 가진 애들로만 이루어져 있으니까 matrix도 마찬가지일 것입니다.

```
mt1 <- cbind(vec_lo, vec_in)

> mt1
      vec_lo vec_in
[1,]      1      1
[2,]      0      2
[3,]      1      3
[4,]      1      4
```

참고로, `rbind` 를 하면 이런 결과가 나옵니다.

```
> rbind(vec_lo, vec_in)
      [,1] [,2] [,3] [,4]
vec_lo   1    0    1    1
vec_in   1    2    3    4
```

행렬은 행과 열로 이루어졌다고 말씀드렸습니다. 그러면 R에서도 행과 열을 전치 (transpose, 반전)할 수 있을까요?

R에서는 `t()` 명령어로 행렬의 전치가 가능합니다.

뿐만 아니라, 행렬의 다양한 연산도 가능합니다.

- 행렬의 덧셈과 뺄셈 : `+`, `-`
- 행렬의 성분별 곱연산 : `*`

$mt1 = \begin{bmatrix} 1 & 1 \\ 0 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix}$ 의 각 성분별 연산이 가능할까요? R에서는 `*` 연산자로 성분별 곱을 계산할 수 있습니다.

```
> mt1*mt1
      vec_lo vec_in
[1,]      1      1
[2,]      0      4
[3,]      1      9
[4,]      1     16
```

- 행렬의 곱 : `%*%`

$\begin{bmatrix} 1 & 1 \\ 0 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 0 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix}$ 은 불가능한 행렬 연산입니다. 행렬곱은  $(m \times n) \times (n \times r)$  형태일 때  $m \times r$  크기 행렬로 나타낼 수 있기 때문입니다.

$$A \times B = \begin{bmatrix} a \\ b \end{bmatrix} \times \begin{bmatrix} c & d \end{bmatrix} = \begin{bmatrix} ac & ad \\ bc & bd \end{bmatrix}$$

`t()` 명령어로 전치한 뒤 행렬의 곱을 연산할 수 있습니다.

```
> t(mt1)%*%mt1
      vec_lo vec_in
vec_lo      3      8
vec_in      8     30

> mt1 %*% t(mt1)
      [,1] [,2] [,3] [,4]
[1,]     2     2     4     5
[2,]     2     4     6     8
```

```
[3,] 4 6 10 13
[4,] 5 8 13 17
```

- 역행렬 : `solve()`

역행렬은 어떤 행렬 **A**와 곱했을 때 단위행렬 **I**가 나오게 하는 행렬을 의미합니다. 그런데 왜 `inverse`가 아니라 `solve()` 로 정의했을까요?

연립방정식  $Ax = b$  이 주어졌을 때 해를  $x = A^{-1}b$ 로 구할 수 있기 때문입니다. 즉, 역행렬을 구한다는 것은 연립방정식의 해를 구하는 것이기 때문에 `solve()` 역시 자연스럽습니다.

```
mt2 <- matrix(1:4, nrow = 2, ncol = 2)
imt2 <- solve(mt2)

> imt2
      [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5
```

행과 열의 이름을 바꾸어 줄 수 있습니다.

- `rownames()` : 행의 이름을 바꾸는 함수
- `colnames()` : 열의 이름을 바꾸는 함수

아까 전에 만들었던 행렬 `mt2` 를 볼까요?

```
> mt2
      [,1] [,2]
[1,]    1    3
[2,]    2    4
```

행렬 `mt2` 는 이미 존재하는 벡터들로 만든 행렬이 아니기 때문에, 임의로 행과 열의 이름이 붙여져 있습니다.

`[,1]`, `[,2]` 같은 이름은 아무런 의미가 없는데요, 이름을 붙여 주신다면 행렬을 더 잘 활용할 수 있겠습니다.

모르는 함수를 이용하실 때에는 googling을 해도 좋지만, R의 basic function인 `help()` 를 이용하는 습관을 들이시면 좋습니다. 저도 두 방법을 사용하면서 모르는 함수들을 하나

씩 익히고 있어요.

`help()` 함수를 이용하는 방법은 `help`의 괄호 안에 궁금한 함수의 이름을 넣는 것입니다.

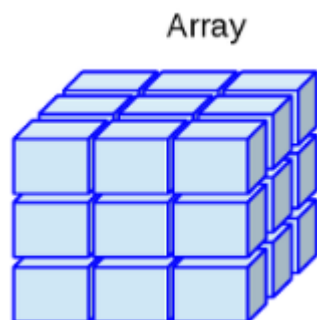
```
colnames(x, do.NULL = TRUE, prefix = "col")
colnames(x) <- value
```

`colnames()`의 괄호 안에는 이름붙일 행렬의 이름을, `value`에는 이름의 벡터를 입력하면 됩니다.

`colnames()` 함수를 이용하여 행렬의 열 이름을 `male`, `female`로 붙여 봅시다.

```
colnames(mt2) <- c("male", "female")
> mt2
      male female
[1,]    1      3
[2,]    2      4
```

## ■ Array



: 동일한 **2D 데이터 구조**를 쌓아 올린 형태를 의미합니다. 행렬이 2D 차원인 것이라면, array는 행렬들을 n개의 층으로(n 차원으로) 쌓아 올렸다고 이해할 수 있습니다.

`array()` 명령어로 array를 만들 수 있습니다. 단, `array()` 에는 input data 외에 추가적으로 차원(dimension)을 설정해 주어야 합니다.

```
a1 <- array(1:12, c(2,3,2))

> a1
, , 1
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

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

위 예시는 `1:12` 를 이용하여 행렬을 만들되,  $2 \times 3$  matrix로, 2차원으로 만들라는 명령어입니다.

## ■ factor

: 범주형 변수의 레벨을 만들어 integer로 저장하여(dummy 변수화), 범주형 변수를 효율적으로 다루는 방법입니다.

```
vec_on <- c("하나", "둘", "셋", "둘")
vec_onf <- factor(vec_on)

> typeof(vec_onf)
[1] "integer"
```

그렇다면 `factor()` 명령어로 만들어진 벡터끼리 연산할 수 있을까요?

```
> vec_onf + vec_onf
[1] NA NA NA NA
Warning message:
In Ops.factor(vec_onf, vec_onf) : '+' not meaningful for factors
```

`integer`처럼 보이지만 실제 연산을 할 수는 없습니다. 본질적으로 범주형 변수이기 때문입니다.

## ■ list

: 벡터와 유사하지만, **여러 자료형의 데이터**를 보존하면서 저장이 가능합니다.

```
vec <- c(v1,v2)
> vec
[1] "1"    "2"    "3"    "1"    "1.2" "2"
```

벡터에 서로 다른 자료형을 입력했을 때 오류가 발생하지는 않았습니다.

대신 적은 정보를 가지고 있는 element를 많은 정보를 가지고 있는 element로 강제 변환하여, 결론적으로 '같은 자료형의 데이터'를 가지고 있게끔 만듭니다.

하지만 리스트는 벡터와 달리 **서로 다른 자료형**을 보존하면서 저장할 수 있습니다.

```
## list
v1 <- c(1,2,3)      ## double
v2 <- c("1",1.2,2)  ## character
lst <- list(v1,v2)  ## both double and character

> lst
[[1]]
[1] 1 2 3

[[2]]
[1] "1"    "1.2" "2"
```

참고로 리스트에 이름(label)을 붙여 관리할 수 있습니다.



```
lst2 <- list(double=v1, character=v2)

> lst2
$double
[1] 1 2 3

$character
[1] "1" "1.2" "2"
```

`names()` 명령어로 리스트의 이름들을 확인할 수 있으며, 활용해서 이름을 뒤늦게 붙여줄 수도 있습니다.

```
labeled_list <- list(c(1,2,3), c(10,20,30))
names(labeled_list) <- c("male", "female")

> labeled_list
$male
[1] 1 2 3

$female
[1] 10 20 30
```

## ■ data frame (🌟 R에서 가장 빈번하게 사용)

: 각 열이 서로 다른 데이터 타입을 가질 수 있으나, 열 안에서 데이터 타입은 동일해야 하며, 변수의 길이가 같아야 한다는 조건이 있습니다(엑셀 시트를 생각해 보시면 쉽습니다). 단, 하나의 객체 크기가 다른 객체 크기의 배수이면 recycle하여 생성합니다.

`data.frame()` 명령어로 data frame을 만들 수 있습니다.

그럼 이런 의문이 자연스럽게 들 수 있습니다.

list와 data frame 모두 서로 다른 데이터 타입을 보존하는데,  
그럼 자유도가 더 높은 list를 쓰는 게 맞지 않은가?

맞는 이야기이지만, 실무에서 다루는 데이터의 특징을 생각해 보면 왜 data frame을 자주 사용하는지 이해할 수 있습니다.

통계 문제를 해결하기 위해서는 **데이터 간의 관계**가 필수적으로 고려되어야 합니다. 왜 엑셀이 가장 흔하게 사용되는지를 생각해 보시면, 표 형태로 표현된 데이터가 얼마나 강력한지 체감하실 수 있습니다.

	<i>Name</i>	<i>Team</i>	<i>Number</i>	<i>Position</i>	<i>Age</i>
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

source : <https://www.geeksforgeeks.org/dataframe-operations-in-r/>

위의 예시를 이용해 데이터프레임을 만들어 봅시다.

```
vec1 <- c("john", "terry", "evan")
vec2 <- c(10, 20, 30)
df <- data.frame(vec1, vec2)
```

vec1과 vec2 열로 데이터프레임이 만들어졌습니다. 그렇다면 행렬에서처럼 데이터프레임의 이름을 바꾸어 줄 수 있을까요? `names()` 함수를 이용해 열의 이름을 바꾸어 줄 수 있

습니다.

```
names(df) <- c("name", "age")
df
```

엑셀을 비롯한 스프레드시트를 사용하실 때 행의 이름을 딱히 바꾼 기억이 없으시죠? 데이터프레임을 이용할 때에도 행의 이름을 바꾸는 거의 없습니다만, `dimnames()` 함수를 이용해 행의 이름을 바꿀 수도 있습니다.

## ■ function

element와 object를 설명드리는 과정에서 다양한 함수를 사용했습니다. R에서는 function 역시 object으로 취급합니다.

이렇게 function을 제외한 6가지가 R에서 주로 사용하는 object입니다.

## Object와 indexing

혹시 다른 프로그래밍 언어를 해보셨다면, 인덱싱에 익숙하실 겁니다. 인덱스는 쉽게 말해 데이터에 접근하기 위한 '색인'입니다.

6	1	3	6	10	5
---	---	---	---	----	---

`vec[5]`

Python, MATLAB 등 다른 언어에서 인덱싱은 0부터 시작하는데, R은 1부터 시작함에 유의해 주세요.

- 방법 1 `[]` 로
  - vector : `[i]`
  - matrix : `[i, j]` i행 j열
  - array : `[i, j, k]` i행 j열 k층
  - dataframe : `[i, j]` i행 j열 → 데이터 프레임으로 출력 / 혹은 `[[i, j]]` → 벡터로 출력
  - list : `[[i]]` : i층
- 방법 2 `$` 로 : 벡터 구조의 인덱싱, `$변수이름`

간단하게 index를 확인해 보는 코드를 작성해 봅시다. 아까 만든 벡터 `vec2` 의 두 번째 요소에 접근하기 위해서는

```
vec2 <- c(10, 20, 30)
> vec2
[1] 10 20 30

vec2[2]
> vec2[2]
[1] 20
```

위의 코드를 작성하면 됩니다. vector, matrix, array, dataframe, list에 대해서 인덱싱을 해 보고, 예상하는 값이 잘 리턴되었는지 잠깐 실험을 해 보세요.

# Loop programming

R에서는 Python만큼 들여쓰기를 신경쓰지 않아도 되지만, 일반적으로는 가독성을 위해 탭을 이용합니다.

## 1부터 100까지 더하기

```
res <- 0
for (i in 1:100){
  res <- res + i
}
res
```

## 1부터 100까지 더하되 70에서 80은 더하지 않기

```
res <- 0
for (i in 1:100){
  if (i>=70 && i<=80) next ## i가 70 이상 80 이하라면, 바로 아래줄을 실행하지 않고 넘깁니다.
  res <- res + i
}
res
```

`&`은 element-wise하게 비교가 이루어지는 반면, `&&`은 첫 번째 값만 확인합니다.

## 몇 번 루프할 지 확실하지 않은 경우

"1에서 어떤 수까지 더하되 그 합이 10000을 처음으로 넘는 수 구하기" 처럼 정확히 몇 번 반복해야 할지 정해지지 않은 경우가 있습니다.

이 경우에는 특정 조건을 만족할 때까지 루프를 반복하는 `repeat()` 문을 사용하여 문제를 해결할 수 있습니다.

그러나 `repeat()` 문은 루프를 탈출할 수 있는 `break` 조건을 명시해 주어야 합니다. 그렇지 않으면 영원히 루프를 맴돌게 되는 '무한 루프'가 일어나 버립니다.

## Quiz



google, `help()` 과 친해지면 좋습니다.

특히 `help()` 명령어로 공식 문서를 확인하는 습관을 들이면 좋습니다.



변수 이름을 의미있게 짓는 습관을 들이면 좋습니다. `a`, `b` 같은 변수 이름보다

`edu`, `gender` 같은 변수 이름이 더 많은 의미를 가지고 있어요.

감사합니다.