

강원대학교  
소프트웨어학과

---

# 재난안전 프로그래밍

---

Chapter 2. 데이터 구조와 함수 이해

# Chapter 2. 데이터 구조와 함수 이해

## 2.1 데이터 구조의 이해

- 데이터란?
  - 이론을 세우는 데 기초가 되는 사실, 또는 바탕이 되는 자료
  - 관찰이나 실험, 조사로 얻은 사실이나 자료
  - 컴퓨터가 처리할 수 있는 문자, 숫자, 소리, 그림 따위의 형태로 된 자료
  - 데이터는 신호, 기호, 숫자, 문자 등으로 기록 됨
  - 정보를 위한 기초적인 자료를 말함
  - 정보는 데이터를 가공하지 않은 경우

# Chapter 2. 데이터 구조와 함수 이해

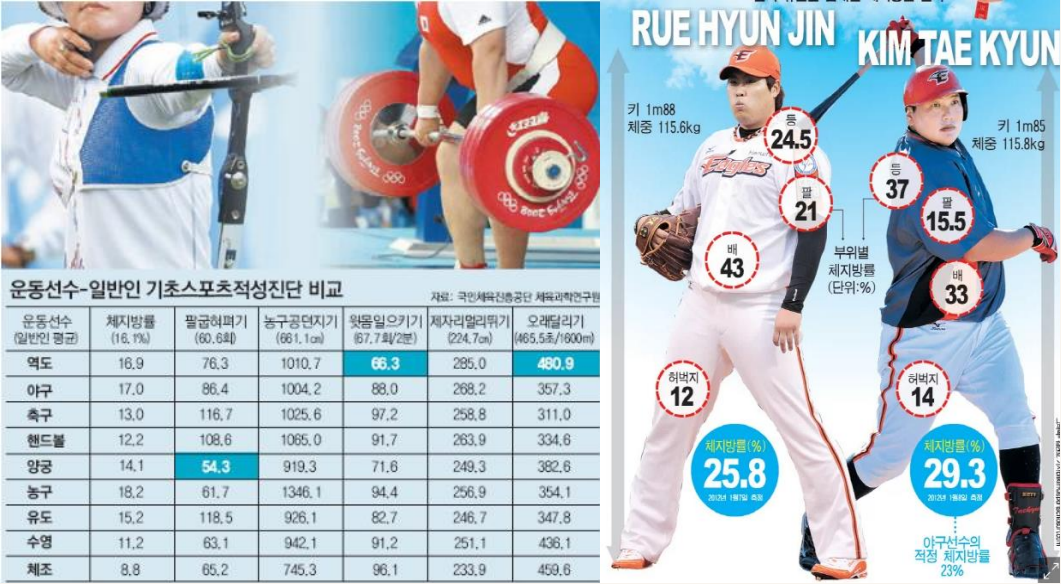
## 2.1 데이터 구조의 이해

- 정보란? → 구성, 해석 및 맥락화 과정을 통해 데이터에서 파생

### 선수들의 수치

PLAYER	DPM	GOLDDIFFAT15	분당 K+A	GPM	분당 골드 차이	분당 데미지 차이
FAKER	375	232.21	0.220	396	11.732	36.575
SHOWMAKER	488	82.86	0.294	404	19.901	31.411
CHOVY	466	352.44	0.223	410	27.059	-30.201
BDD	461	-1.41	0.269	389	4.989	53.180
GORI	459	-400.44	0.239	397	3.552	-29.395
FATE	412	236.74	0.238	406	21.573	-8.689

### 선수들의 신체 조건에 따른 적성 진단



# Chapter 2. 데이터 구조와 함수 이해

## 2.1 데이터 구조의 이해

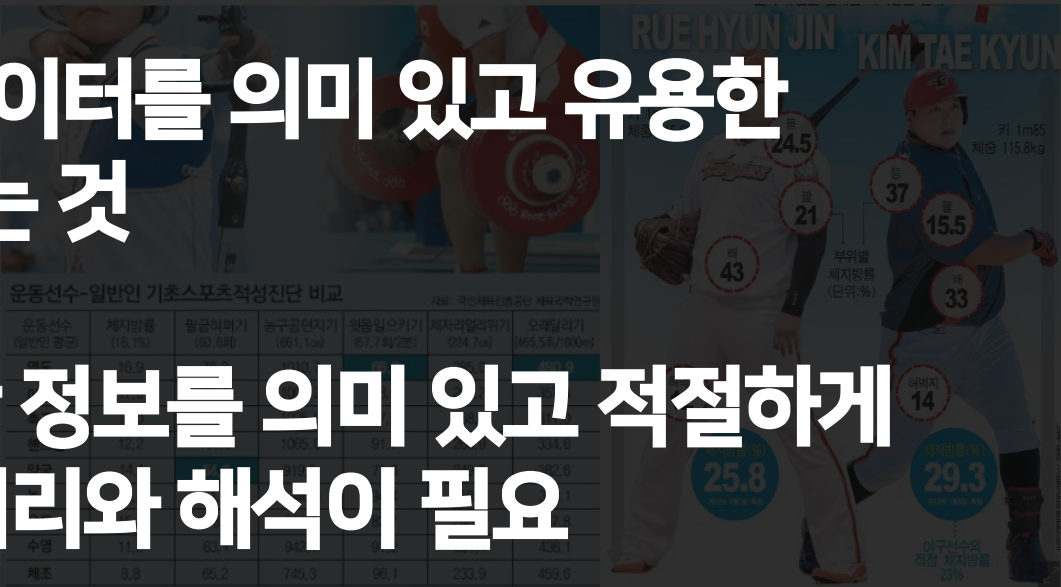
- 정보란? □ 구성, 해석 및 맥락화 과정을 통해 데이터에서 파생

### 데이터는 정보가 생성되는 원재료

선수들의 수치

선수들의 신체 조건에 따른 적성 진단

PLAYER	DPM	GOLDDIFFAT15	본당 K+A	GPM	본당 골드 차이	본당 데미지 차이
FAKER	475	82.86	0.223	410	27.059	30.201
SHOWMAKER	488	82.86	0.223	410	27.059	30.201
CHOVY	466	352.44	0.223	410	27.059	30.201
BDD	461	-1.41	0.269	389	4.989	53.180
GOL	475	82.86	0.223	410	27.059	30.201
FATE	475	82.86	0.223	410	27.059	30.201



정보는 새로운 가치를 생성하고, 데이터를 의미 있고 유용한 형태로 변환하는 것

정보 생성을 위해 데이터가 필요하지만 정보를 의미 있고 적절하게 만들기 위해서는 추가적인 처리와 해석이 필요

## Chapter 2. 데이터 구조와 함수 이해

### 2.1 데이터 구조의 이해

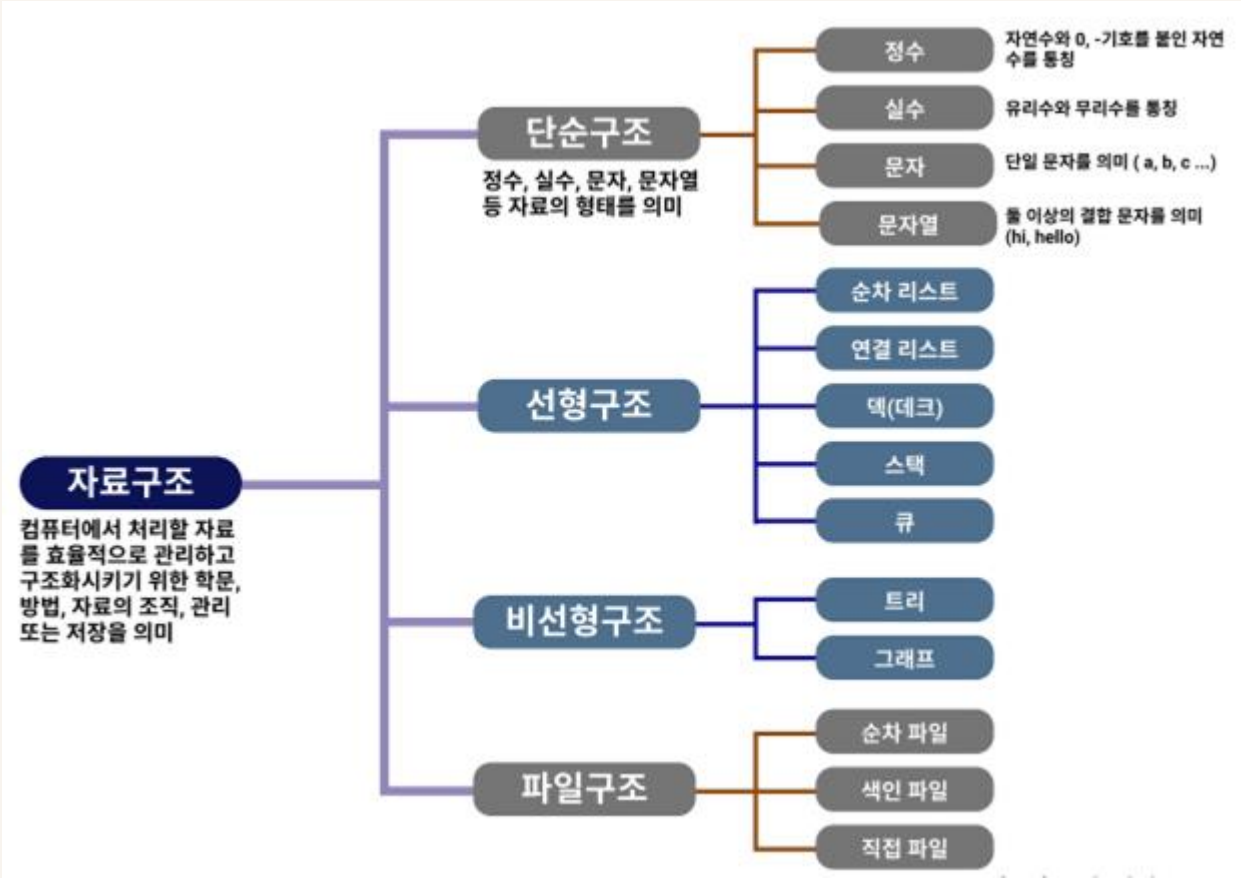
- 프로그램이란 데이터를 표현하고, 표현된 데이터를 처리하는 것



# Chapter 2. 데이터 구조와 함수 이해

## 2.1 데이터 구조의 이해

- 데이터의 구조는

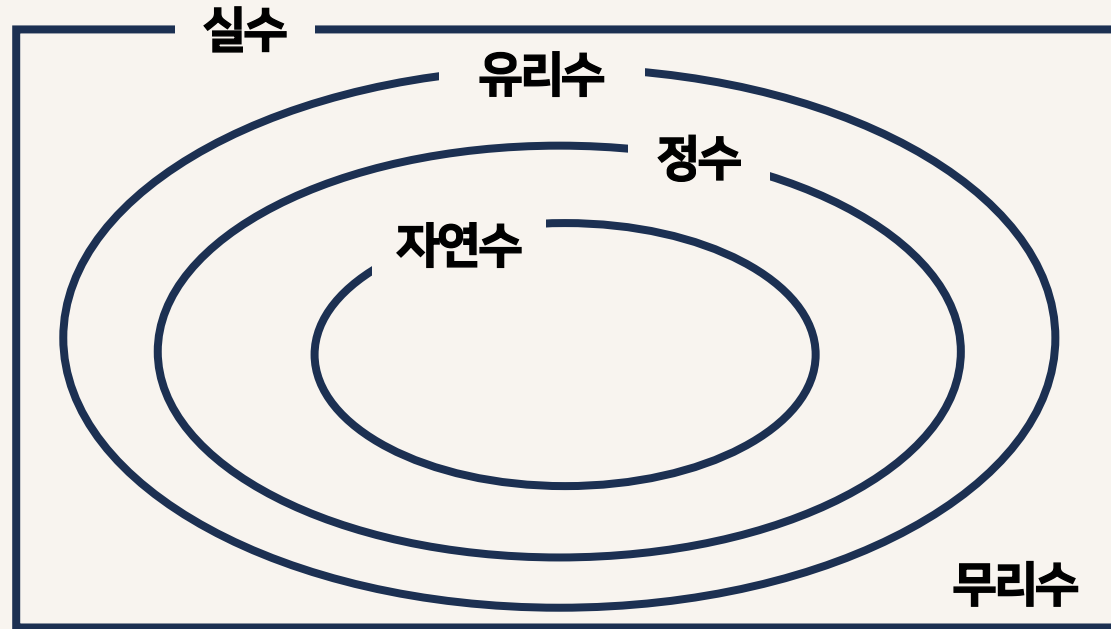


## Chapter 2. 데이터 구조와 함수 이해

### 2.1 데이터 구조의 이해

#### 단순구조

- 정수, 실수, 문자, 문자열 등
- 정수 : 양의 정수, 0, 음의 정수 or 자연수, (-)자연수
- 실수 : 유리수 → 정수와 분수가 존재, 소수로 나타내면 유한 소수나 순환 소수  
무리수 → 간단한 분수로 고칠 수 없는 수, 소수점 아래의 수가 반복되지 않고 무한히 계속되는 소수





# Chapter 2. 데이터 구조와 함수 이해

## 2.1 데이터 구조의 이해

### 단순구조

- 문자(Character) : 숫자이외의 정보를 표현하는 방법
- 문자열(String) : 둘 이상의 문자의 결합
- 문자를 컴퓨터가 이해할 수 있는 숫자로 변환 → 인코딩

ASCII TABLE															
Decimal Hex Char				Decimal Hex Char				Decimal Hex Char				Decimal Hex Char			
0	0		00	16	0		00	32	0		00	48	0		00
1	1		01	17	1		01	33	1		01	49	1		01
2	2		02	18	2		02	34	2		02	50	2		02
3	3		03	19	3		03	35	3		03	51	3		03
4	4		04	20	4		04	36	4		04	52	4		04
5	5		05	21	5		05	37	5		05	53	5		05
6	6		06	22	6		06	38	6		06	54	6		06
7	7		07	23	7		07	39	7		07	55	7		07
8	8		08	24	8		08	40	8		08	56	8		08
9	9		09	25	9		09	41	9		09	57	9		09
10	A		0A	26	A		0A	42	A		0A	58	A		0A
11	B		0B	27	B		0B	43	B		0B	59	B		0B
12	C		0C	28	C		0C	44	C		0C	60	C		0C
13	D		0D	29	D		0D	45	D		0D	61	D		0D
14	E		0E	30	E		0E	46	E		0E	62	E		0E
15	F		0F	31	F		0F	47	F		0F	63	F		0F
16	0		10	32	0		10	48	0		10	64	0		10
17	1		11	33	1		11	49	1		11	65	1		11
18	2		12	34	2		12	50	2		12	66	2		12
19	3		13	35	3		13	51	3		13	67	3		13
20	4		14	36	4		14	52	4		14	68	4		14
21	5		15	37	5		15	53	5		15	69	5		15
22	6		16	38	6		16	54	6		16	70	6		16
23	7		17	39	7		17	55	7		17	71	7		17
24	8		18	40	8		18	56	8		18	72	8		18
25	9		19	41	9		19	57	9		19	73	9		19
26	A		1A	42	A		1A	58	A		1A	74	A		1A
27	B		1B	43	B		1B	59	B		1B	75	B		1B
28	C		1C	44	C		1C	60	C		1C	76	C		1C
29	D		1D	45	D		1D	61	D		1D	77	D		1D
30	E		1E	46	E		1E	62	E		1E	78	E		1E
31	F		1F	47	F		1F	63	F		1F	79	F		1F

American  
Standard  
Code for  
Information  
Interchange



Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	000	NUL (null)	32	20	040	Space	64	40	100	0	96	60	140	0
1	1	001	SOH (start of heading)	33	21	041	!	65	41	101	A	97	61	141	a
2	2	002	STX (start of text)	34	22	042	"	66	42	102	B	98	62	142	b
3	3	003	ETX (end of text)	35	23	043	#	67	43	103	C	99	63	143	c
4	4	004	EOT (end of transmission)	36	24	044	\$	68	44	104	D	100	64	144	d
5	5	005	ENQ (enquiry)	37	25	045	%	69	45	105	E	101	65	145	e
6	6	006	ACK (acknowledge)	38	26	046	&	70	46	106	F	102	66	146	f
7	7	007	BEL (bell)	39	27	047	'	71	47	107	G	103	67	147	g
8	8	010	BS (backspace)	40	28	050	(	72	48	110	H	104	68	150	h
9	9	011	TAB (horizontal tab)	41	29	051	)	73	49	111	I	105	69	151	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	74	4A	112	J	106	70	152	j
11	B	013	VT (vertical tab)	43	2B	053	+	75	4B	113	K	107	71	153	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	76	4C	114	L	108	72	154	l
13	D	015	CR (carriage return)	45	2D	055	-	77	4D	115	M	109	73	155	m
14	E	016	SO (shift out)	46	2E	056	.	78	4E	116	N	110	74	156	n
15	F	017	SI (shift in)	47	2F	057	/	79	4F	117	O	111	75	157	o
16	10	020	DLE (data link escape)	48	30	060	0	80	50	120	P	112	76	160	p
17	11	021	DC1 (device control 1)	49	31	061	1	81	51	121	Q	113	77	161	q
18	12	022	DC2 (device control 2)	50	32	062	2	82	52	122	R	114	78	162	r
19	13	023	DC3 (device control 3)	51	33	063	3	83	53	123	S	115	79	163	s
20	14	024	DC4 (device control 4)	52	34	064	4	84	54	124	T	116	80	164	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	85	55	125	U	117	81	165	u
22	16	026	SYN (synchronous idle)	54	36	066	6	86	56	126	V	118	82	166	v
23	17	027	ETB (end of trans. block)	55	37	067	7	87	57	127	W	119	83	167	w
24	18	030	CAN (cancel)	56	38	070	8	88	58	130	X	120	84	170	x
25	19	031	EM (end of medium)	57	39	071	9	89	59	131	Y	121	85	171	y
26	1A	032	SUB (substitute)	58	3A	072	:	90	5A	132	Z	122	86	172	z
27	1B	033	ESC (escape)	59	3B	073	;	91	5B	133	[	123	87	173	{
28	1C	034	FS (file separator)	60	3C	074	<	92	5C	134	\	124	88	174	
29	1D	035	GS (group separator)	61	3D	075	=	93	5D	135	^	125	89	175	~
30	1E	036	RS (record separator)	62	3E	076	>	94	5E	136	_	126	90	176	
31	1F	037	US (unit separator)	63	3F	077	?	95	5F	137	`	127	91	177	

아스키(ASCII) : 미국 국립 표준협회(ANSI, American National Standards Institute)

현대는 Unicode를 더 많이 사용



# Chapter 2. 데이터 구조와 함수 이해

## 2.1 데이터 구조의 이해

### 데이터 타입

- 컴퓨터 시스템과 프로그래밍 언어에서 실수, 정수, 소수, 자료형 등의 여러 종류의 데이터를 식별 가능하게 하는 것
- 데이터의 형태, 의미, 크기와 해당 자료형의 값이 저장되는 방식

Example	Type
"A", "Hello"	문자(character)
10, 20, 1.178	유리수(numeric)
5, 10, 2	정수(integer)
TRUE, FALSE	논리값(logical)

# Chapter 2. 데이터 구조와 함수 이해

## 2.1 데이터 구조의 이해

### 데이터 타입

- 컴퓨터 시스템과 프로그래밍 언어에서 실수, 정수, 소수, 자료형 등의 여러 종류의 데이터를 식별 가능하게 하는 것
- 데이터의 형태, 의미, 크기와 해당 자료형의 값이 저장되는 방식

```
typeof("Hello")  
[1] "character"
```

```
mode("Hello")  
[1] "character"
```

```
typeof(123)  
[1] "double" → 모든 숫자 값을 포함하는 의미
```

```
mode(123)  
[1] "numeric"
```

```
typeof(1.25)  
[1] "double"
```

```
mode(1.25)  
[1] "numeric"
```

```
typeof(TRUE)  
[1] "logical"
```

```
mode(TRUE)  
[1] "logical"
```

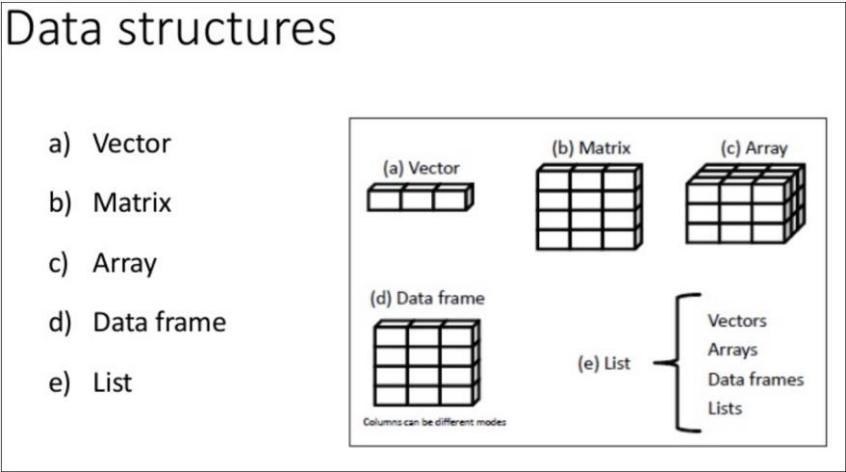
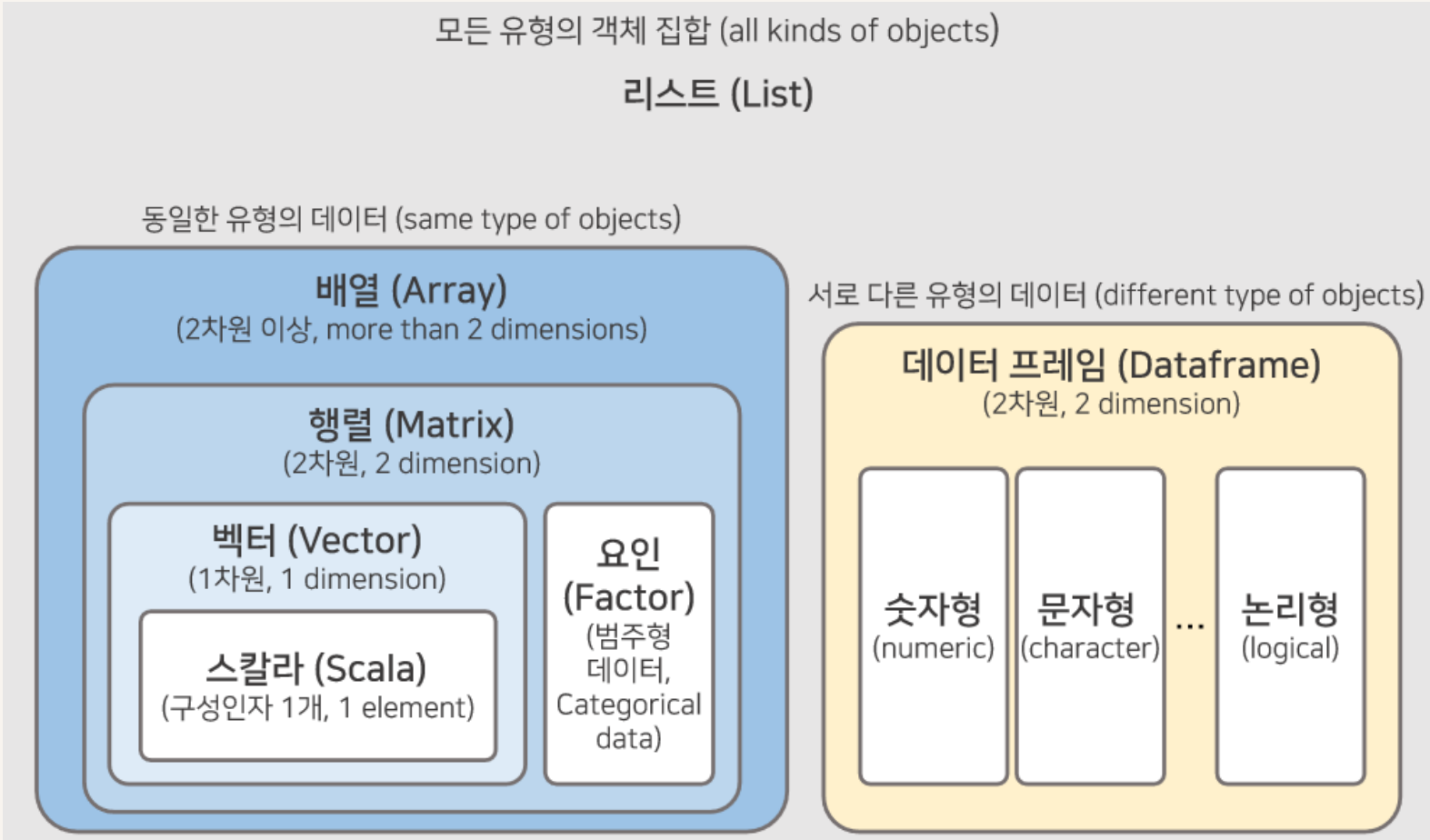
```
typeof("123")  
[1] "character"
```

```
mode("123")  
[1] "character"
```

# Chapter 2. 데이터 구조와 함수 이해

## 2.1 데이터 구조의 이해

### 데이터의 구조



## Chapter 2. 데이터 구조와 함수 이해

### 2.2 리스트

#### 선형구조-리스트

- 데이터 유형을 저장하고, 저장된 데이터들을 그룹화할 수 있는 데이터 구조
- 숫자, 문자, 논리값 ... 등등 다양한 데이터 유형의 요소가 포함될 수 있음

```
List <- list(1, 2, 3)
```

```
[[1]]
```

```
[1] 1
```

```
[[2]]
```

```
[1] 2
```

```
[[3]]
```

```
[1] 3
```

```
typeof(List)
```

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

```
List <-list(1.6, 2.3, 3.5)
```

```
List
```

```
[[1]]
```

```
[1] 1.6
```

```
[[2]]
```

```
[1] 2.3
```

```
[[3]]
```

```
[1] 3.5
```

```
mode(List)
```

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

## Chapter 2. 데이터 구조와 함수 이해

### 2.2 리스트

#### 선형구조-리스트

- 각각의 다른 데이터 형태를 모두 묶어서 그룹화할 수 있음

```
List <- list("apple", "banana", "orange", 1, 1.5, TRUE)
```

List

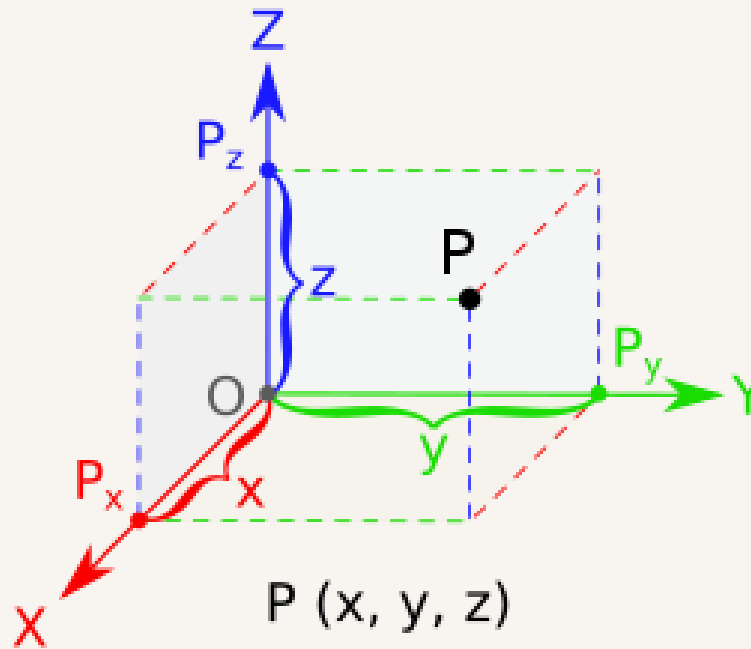
[[1]]	[[4]]
[1] "apple"	[1] 1
[[2]]	[[5]]
[1] "banana"	[1] 1.5
[[3]]	[[6]]
[1] "orange"	[1] TRUE

## Chapter 2. 데이터 구조와 함수 이해

### 2.3 벡터와 배열

#### 물리에서 벡터란?

- 크기와 방향을 갖는 물리량
- 벡터는 사물의 움직임을 프로그래밍하기 위한 가장 기본적인 구성요소





## Chapter 2. 데이터 구조와 함수 이해

### 2.3 벡터와 배열

#### 프로그램에서 벡터란?

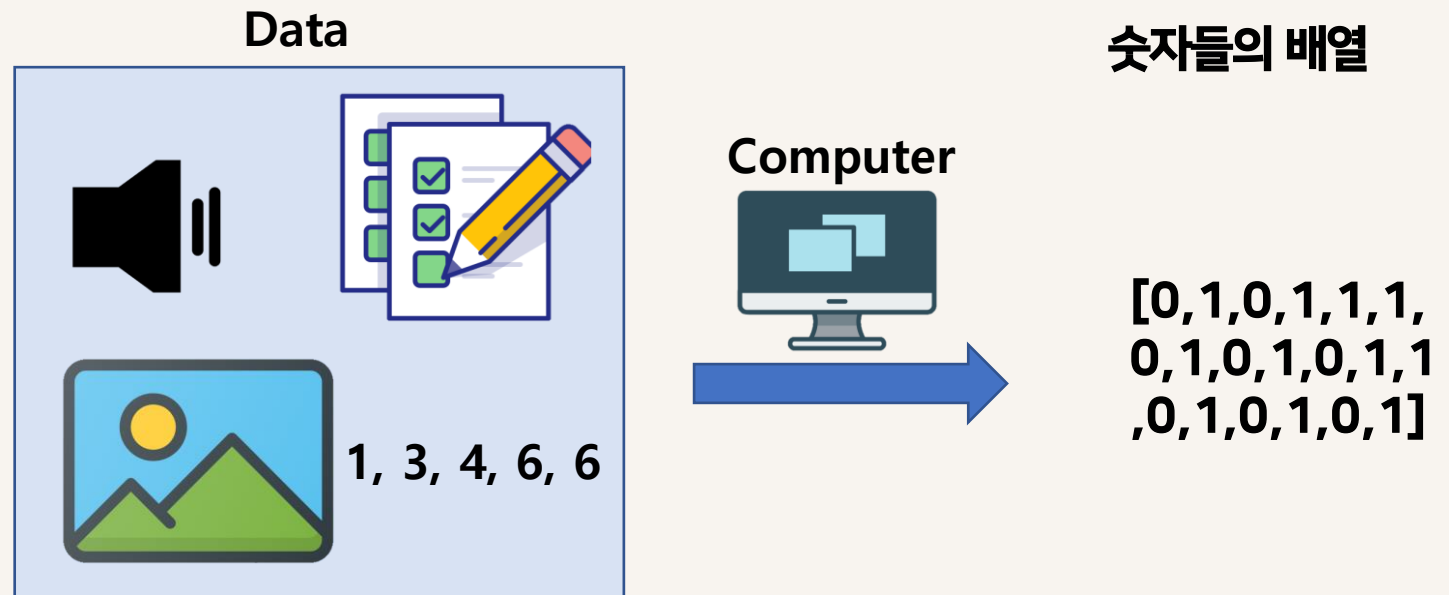
- 값을 저장하고, 조작할 수 있는 기본 데이터 구조
- 숫자, 문자 또는 논리 값과 같은 동일한 데이터 유형의 요소를 보유할 수 있는 1차원 배열
- R의 벡터는 combin을 나타내는 `c()` 함수를 사용하여 만들 수 있음

#### List(리스트)

- 자료를 순서대로 한 줄로 저장하는 자료구조
- 여러 자료가 일직선으로 서로 연결된 선형 구조(리스트에 있는 데이터는 몇 번째 인지 의미를 가짐)

#### Array(배열)

- 단일 타입으로 구성되는 자료구조



## Chapter 2. 데이터 구조와 함수 이해

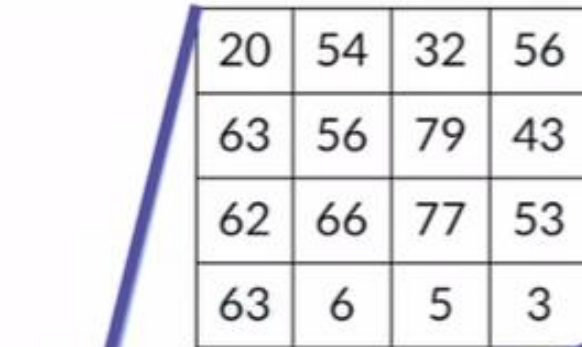
### 2.3 벡터와 배열

#### 대규모 다차원 배열

- 데이터의 대부분은 숫자 배열로 볼 수 있음
- 흑백 이미지는 픽셀의 밝기와 명암을 2차원 배열로 표현할 수 있고 소리 같은 경우는 1차원 배열로 나타낼 수 있음

#### List(리스트)와 Array(배열)

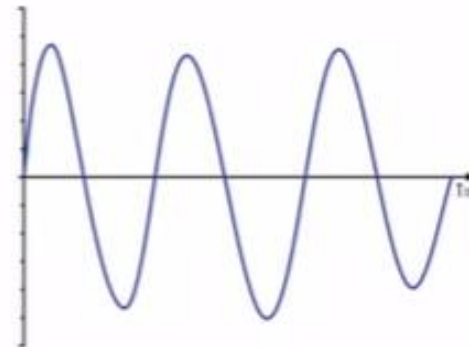
- List는 [1,2,"Kim",2.5,True,False]와 같은 실수형, 정수형, 문자열과 같은 다양하게 관계없이 구성이 가능함
- array(배열)는 모두 단일 타입으로 구성됨



20	54	32	56
63	56	79	43
62	66	77	53
63	6	5	3



23	44	52	56	42	38
----	----	----	----	----	----



## Chapter 2. 데이터 구조와 함수 이해

### 2.3 벡터와 배열

#### 프로그램에서 벡터란?

- 값을 저장하고, 조작할 수 있는 기본 데이터 구조
- 숫자, 문자 또는 논리 값과 같은 동일한 데이터 유형의 요소를 보유할 수 있는 1차원 배열
- R의 벡터는 `combin`을 나타내는 `c()` 함수를 사용하여 만들 수 있음

```
Vector <- c(1, 2, 3)
```

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

```
logical_vector <- c(TRUE, FALSE, TRUE)
```

```
logical_vector  
[1] TRUE FALSE TRUE
```

```
char_vector <- c("apple", "banana", "orange")
```

```
char_vector  
[1] "apple" "banana" "orange"
```

## Chapter 2. 데이터 구조와 함수 이해

### 2.3 벡터와 배열

#### 프로그램에서 벡터란?

- 값을 저장하고, 조작할 수 있는 기본 데이터 구조
- 숫자, 문자 또는 논리 값과 같은 동일한 데이터 유형의 요소를 보유할 수 있는 1차원 배열
- R의 벡터는 `combin`을 나타내는 `c()` 함수를 사용하여 만들 수 있음

```
Vector <- c(1, 2, 3)
```

```
Vector
```

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

```
mode(Vector)
```

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

```
List <- list(1, 2, 3)
```

```
[[1]]
```

```
[1] 1
```

```
[[2]]
```

```
[1] 2
```

```
[[3]]
```

```
[1] 3
```

```
typeof(List)
```

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

## Chapter 2. 데이터 구조와 함수 이해

### 2.3 벡터와 배열

#### 프로그램에서 벡터란?

- 다양한 형태의 값들이 하나의 벡터에 들어갈 수 없음
- 리스트 끼리는 연산이 불가능, 벡터 끼리는 연산이 가능

```
Vector <- c("aa", 2, 3)
```

```
Vector  
[1] "aa" "2" "3"
```

```
mode(Vector)  
[1] "character"
```

```
List_1 <-list(1.6, 2.3, 3.5)  
List_2 <-list(2.6, 5.3, 7.5)  
List_1+List_2
```

```
Vector_1 <-c(1.6, 2.3, 3.5)  
Vector_2 <-c(2.6, 5.3, 7.5)  
Vector_1+Vector_2  
[1] 4.2 7.6 11.0
```

## Chapter 2. 데이터 구조와 함수 이해

### 2.3 벡터와 배열

#### 프로그램에서 벡터란?

- 다양한 형태의 값들이 하나의 벡터에 들어갈 수 없음
- 리스트 끼리는 연산이 불가능, 벡터 끼리는 연산이 가능

```
my_list <- list("apple", 3.14, c(1, 2, 3), TRUE)
```

```
my_list
```

```
[[1]]
```

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

```
[[2]]
```

```
[1] 3.14
```

```
[[3]]
```

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

```
[[4]]
```

```
[1] TRUE
```



## Chapter 2. 데이터 구조와 함수 이해

### 2.3 벡터와 배열

#### 행렬(matrix) & 배열(Array)란?

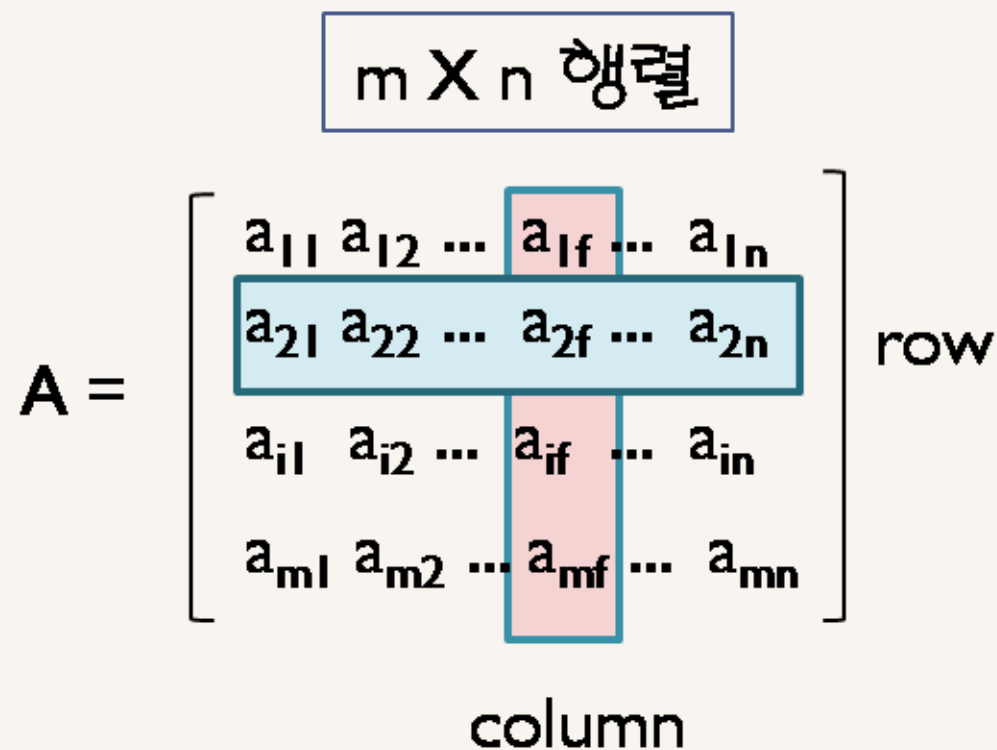
- 벡터의 확장된 개념
- 벡터는 1차원 데이터를 나타내는 데이터의 기본 구조이지만 배열은 다차원 확장으로 표현
- 행렬은 2차원 데이터만을 표현 가능

```
matrix(data = c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)
```

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

```
[1,] 1 3 5
```

```
[2,] 2 4 6
```



## Chapter 2. 데이터 구조와 함수 이해

### 2.3 벡터와 배열

#### 행렬(matrix) & 배열(Array)란?

- 벡터의 확장된 개념
- 벡터는 1차원 데이터를 나타내는 데이터의 기본 구조이지만 배열은 다차원 확장으로 표현
- 행렬은 2차원 데이터만을 표현 가능
- 원하는 행렬을 만들기 위해서는 nrow(행의 개수), ncol(열의 개수)
- 행/열을 우선으로 값 배치 byrow=TRUE or FALSE

```
matrix(data = c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)
```

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

```
a= matrix(c(1,2,3,4,5,6), ncol=2)
```

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

```
a= matrix(c(1,2,3,4,5,6), nrow = 2)
```

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

```
a= matrix(c(1,2,3,4,5,6), nrow=2, byrow = TRUE)
```

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

```
a= matrix(c(1,2,3,4,5,6), nrow=2, byrow = FALSE)
```

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

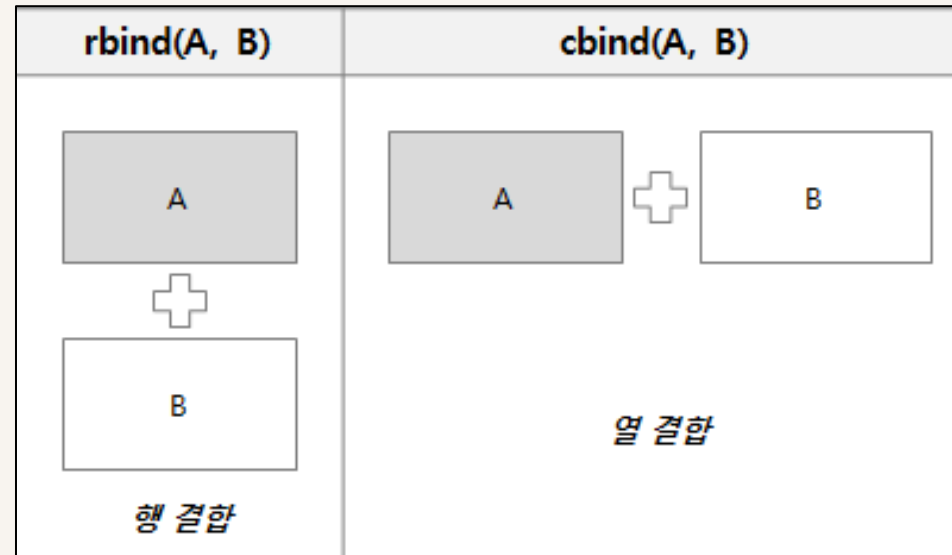
## Chapter 2. 데이터 구조와 함수 이해

### 2.3 벡터와 배열

#### 행렬(matrix) & 배열(Array)란?

- 벡터의 확장된 개념
- 벡터는 1차원 데이터를 나타내는 데이터의 기본 구조이지만 배열은 다차원 확장으로 표현
- 행렬은 2차원 데이터만을 표현 가능
- rbind()는 행 결합 함수이고 cbind()는 열 결합 함수임

```
> rc1<-c(2,4,5,6)
> rc2<-c(7,8,9,10)
> cmatrix<-cbind(rc1, rc2)
> cmatrix
      rc1 rc2
[1,]   2   7
[2,]   4   8
[3,]   5   9
[4,]   6  10
> rmatrix<-rbind(rc1, rc2)
> rmatrix
      [,1] [,2] [,3] [,4]
rc1     2   4   5   6
rc2     7   8   9  10
```



## Chapter 2. 데이터 구조와 함수 이해

### 2.3 벡터와 배열

#### 행렬(matrix) & 배열(Array)란?

- 벡터의 확장된 개념

```
> amatrix<-matrix(c(1,2,3,4,5,6,7,8,9), nrow=3)
> amatrix
      [,1] [,2] [,3]
[1,]     1     4     7
[2,]     2     5     8
[3,]     3     6     9
> bmatrix<-matrix(c(2,5,-6,1,-4,2,-3,8,7),nrow=3, byrow=TRUE)
> bmatrix
      [,1] [,2] [,3]
[1,]     2     5    -6
[2,]     1    -4     2
[3,]    -3     8     7
> amatrix+bmatrix      #행렬의 각 원소끼리 합(행렬합)
      [,1] [,2] [,3]
[1,]     3     9     1
[2,]     3     1    10
[3,]     0    14    16
> 2*amatrix      #행렬의 각 원소를 2배
      [,1] [,2] [,3]
[1,]     2     8    14
[2,]     4    10    16
[3,]     6    12    18
> amatrix*bmatrix      #행렬의 각 원소끼리 곱
      [,1] [,2] [,3]
[1,]     2    20   -42
[2,]     2   -20    16
[3,]    -9    48    63
> amatrix%%bmatrix      #행렬의 곱
      [,1] [,2] [,3]
[1,]   -15    45    51
[2,]   -15    54    54
[3,]   -15    63    57
> sqrt(amatrix)      #원소의 제곱근(음수에 대해서 NaN생성)
      [,1] [,2] [,3]
[1,] 1.000000 2.000000 2.645751
[2,] 1.414214 2.236068 2.828427
[3,] 1.732051 2.449490 3.000000
> amatrix^2      #원소의 거듭제곱
      [,1] [,2] [,3]
[1,]     1    16    49
[2,]     4    25    64
[3,]     9    36    81
```

## Chapter 2. 데이터 구조와 함수 이해

### 2.3 벡터와 배열

#### 행렬(matrix) & 배열(Array)란?

- 벡터의 확장된 개념
- `rownames()`, `colnames()`, `matrix` 함수의 `dimnames` 인자를 이용하여 행과 열에 이름을 붙일 수 있음

```
> m=matrix(c(1,2,3,4), nrow=2)
> m
      [,1] [,2]
[1,]    1    3
[2,]    2    4
```

#### 첫번째 방법

```
rownames(m) = c("Row1", "Row2")
colnames(m) = c("Col1", "Col2")
```

```
      col1 col2
Row1      1    3
Row2      2    4
```

#### 두번째 방법

```
row_names <- c("Row1", "Row2")
col_names <- c("Col1", "Col2")
dimnames(m) = list(row_names, col_names)
```

## Chapter 2. 데이터 구조와 함수 이해

### 2.3 벡터와 배열

#### 행렬(matrix) & 배열(Array)란?

- 벡터의 확장된 개념
- 벡터는 1차원 데이터를 나타내는 데이터의 기본 구조이지만 배열은 다차원 확장으로 표현
- 행렬은 2차원 데이터만을 표현할 수 있지만 배열은 다차원 표현 가능

```
matrix(data = c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)
```

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

```
[1,] 1 3 5
```

```
[2,] 2 4 6
```

```
array(data = c(1, 2, 3, 4, 5, 6), dim = c(2, 3))
```

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

```
[1,] 1 3 5
```

```
[2,] 2 4 6
```



## Chapter 2. 데이터 구조와 함수 이해

### 2.3 벡터와 배열

#### 행렬(matrix) & 배열(Array)란?

- 벡터의 확장된 개념
- 벡터는 1차원 데이터를 나타내는 데이터의 기본 구조이지만 배열은 다차원 확장으로 표현
- 행렬은 2차원 데이터만을 표현할 수 있지만 배열은 다차원 표현 가능

```
array(data = c(1, 2, 3, 4, 5, 6), dim = c(2, 2, 2))
```

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

```
, , 2  
[1,] [2,]  
[1,] 5 1  
[2,] 6 2
```

## Chapter 2. 데이터 구조와 함수 이해

### 2.3 벡터와 배열

#### 행렬(matrix) & 배열(Array)란?

- 벡터의 확장된 개념
- 일반적인 연산이 가능함

```
> mid<-c(23,25,12,25,29,27,26,12)
> fina<-c(21,29,20,22,20,26,27,28)
> array1<-array(mid, dim=c(2,2,2)) #배열1 생성
> array2<-array(fina, dim=c(2,2,2)) #배열2 생성
> array1+array2 #배열 원소끼리의 합
, , 1
     [,1] [,2]
[1,]   44   32
[2,]   54   47
, , 2
     [,1] [,2]
[1,]   49   53
[2,]   53   40
> array1*array2 #배열 원소끼리의 곱
, , 1
     [,1] [,2]
[1,]  483  240
[2,]  725  550
, , 2
     [,1] [,2]
[1,]  580  702
[2,]  702  336
> sqrt(array1) #배열 원소의 제곱근
, , 1
     [,1] [,2]
[1,] 4.795832 3.464102
[2,] 5.000000 5.000000
, , 2
     [,1] [,2]
[1,] 5.385165 5.099020
[2,] 5.196152 3.464102
> array1%%array2 #배열 원소끼리 곱의 합
     [,1]
[1,] 4318
```

# Chapter 2. 데이터 구조와 함수 이해

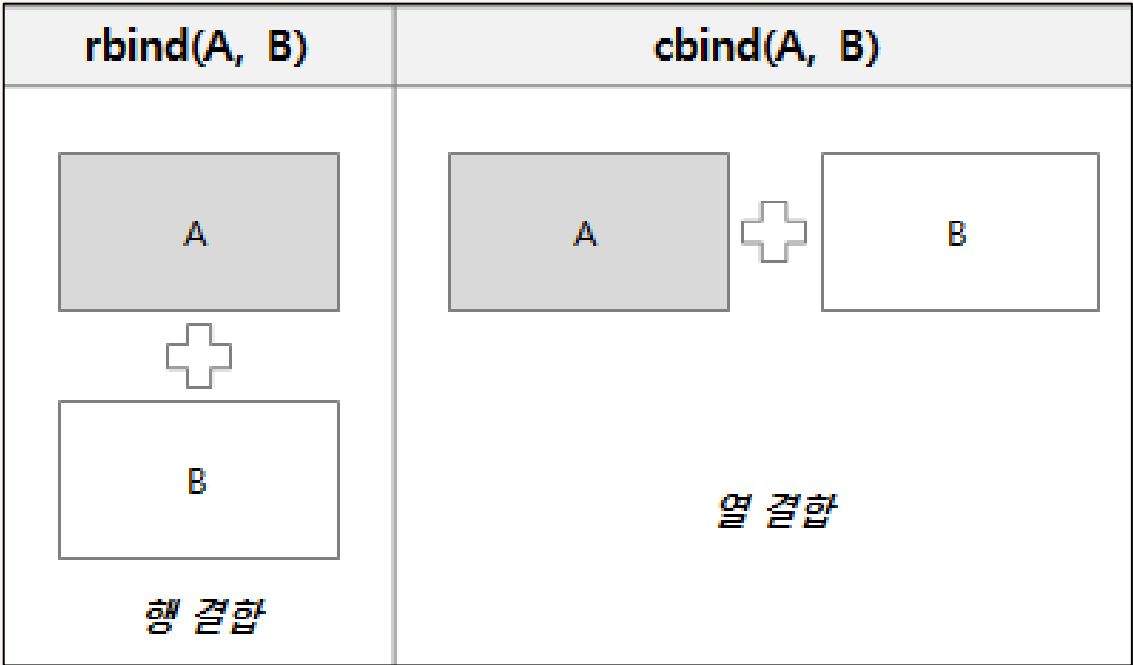
## 2.3 벡터와 배열

### 행렬(matrix) & 배열(Array)란?

- 벡터의 확장된 개념
- 일반적인 연산이 가능함

```
> a=array(data=c(1,2,3,4,5,6), dim=c(2,2))
> b=array(data=c(1,2,3,4,5,6), dim=c(2,2))
> cbind(a,b)
      [,1] [,2] [,3] [,4]
[1,]    1    3    1    3
[2,]    2    4    2    4
```

```
> cbind(a,b)
      [,1] [,2] [,3] [,4]
[1,]    1    3    1    3
[2,]    2    4    2    4
> rbind(a,b)
      [,1] [,2]
[1,]    1    3
[2,]    2    4
[3,]    1    3
[4,]    2    4
```



## Chapter 2. 데이터 구조와 함수 이해

### 2.4 인덱싱

#### 배열의 이름 생성

- **dimnames**를 사용해 차원 별 이름을 부여 할 수 있음

```
my_array <- array(1:12, dim = c(2, 3, 2))
```

```
row_names <- c("Row1", "Row2")  
col_names <- c("Col1", "Col2", "Col3")  
slice_names <- c("Slice1", "Slice2")
```

```
dimnames(my_array) <- list(row_names, col_names, slice_names)
```

#이름을 붙이고 싶지 않을 경우

```
dimnames(my_array) <- list(NULL, col_names, slice_names)
```

# Chapter 2. 데이터 구조와 함수 이해

## 2.3 벡터와 배열

### 행렬(matrix) & 배열(Array)란?

- 벡터의 확장된 개념
- `array(data=x, dim=, dimnames= )` 구조

```
> xarra <-c(28,45,31,60,30,25,50,80) #xarra 벡터 생성
> arra<-array(xarra, dim=c(2,2,2), dimnames=list(c("r1","r2"),c("c1","c2"),c("arr1","arr2")))
> #2x2x2배열을 만들기 위해 array함수를 이용
> #dimnames를 이용하여 각 차원의 이름 지정
> arra
, , arr1
      c1 c2
r1 28 31
r2 45 60

, , arr2
      c1 c2
r1 30 50
r2 25 80

> is.array(arra) #arra가 배열인지 검정
[1] TRUE
> dim(arra)      #arra의 차원
[1] 2 2 2
```

```
> mid <-c(23,25,12,25,29,27)
> fina <-c(21,29,20,22,20,26)
> arrmidfina<-array(data=c(mid,fina),dim=c(2,3,2)) #배열 생성
> rownames(arrmidfina)<-c("Park","Lee")           #행 이름
> colnames(arrmidfina)<-c("Math","Eng","kor")      #열 이름
> arrmidfina
, , 1
      Math Eng kor
Park   23  12  29
Lee    25  25  27

, , 2
      Math Eng kor
Park   21  20  20
Lee    29  22  26
```

→ 행과 열에 이름을 붙일 때 `rownames()`, `colnames()`, `dimnames`인자를 이용

## Chapter 2. 데이터 구조와 함수 이해

### 2.4 인덱싱

#### 인덱싱이란?

- 목록이나 배열에서 특정 값을 추출하기 위해 위치나 인덱스를 지정하는 과정
- 문자, 리스트, 행렬, 배열 모두 위치나 인덱스를 가지고 있음

substr(변수, 시작, 끝)

abcdefg

1 2 3 4 5 6 7



순서

```
library(stringr)
```

```
a<-"abcdefg"
```

```
substr(a, 1,2)
```

```
[1] "ab"
```

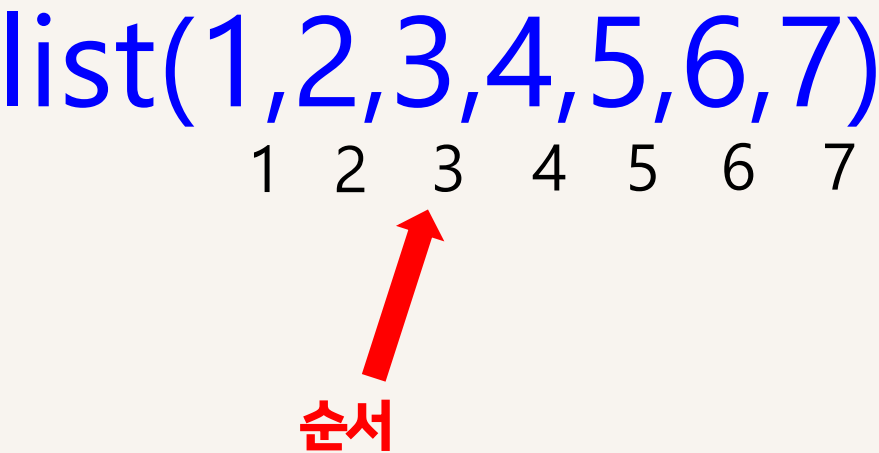


# Chapter 2. 데이터 구조와 함수 이해

## 2.4 인덱싱

### 리스트 인덱싱이란?

- 목록이나 배열에서 특정 값을 추출하기 위해 위치나 인덱스를 지정하는 과정
- 문자, 리스트, 행렬, 배열 모두 위치나 인덱스를 가지고 있음



문법	의미
x[n]	x의 n번째 요소 n은 숫자 또는 셀의 이름을 뜻함
x[-n]	x에서 n번째 요소를 제외한 나머지
x[start:end]	x의 start부터 end까지의 값을 반환함

```
a<-list(1,2,3,4,5,6)
```

```
a[1]
```

```
[[1]]
```

```
[1] 1
```

```
a[[1]]
```

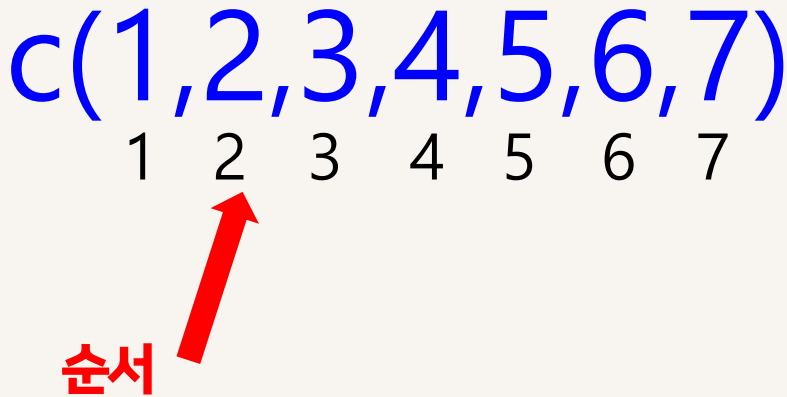
```
[1] 1
```

# Chapter 2. 데이터 구조와 함수 이해

## 2.4 인덱싱

### 벡터 인덱싱이란?(오직 1차원의 형태만 가짐)

- 목록이나 배열에서 특정 값을 추출하기 위해 위치나 인덱스를 지정하는 과정
- 문자, 리스트, 행렬, 배열 모두 위치나 인덱스를 가지고 있음



문법	의미
x[n]	x의 n번째 요소 n은 숫자 또는 셀의 이름을 뜻함
x[-n]	x에서 n번째 요소를 제외한 나머지
x[start:end]	x의 start부터 end까지의 값을 반환함

```
a<-c(1,2,3,4,5,6)
a[1]
[1] 1
```

# Chapter 2. 데이터 구조와 함수 이해

## 2.4 인덱싱

### 배열 인덱싱

- 목록이나 배열에서 특정 값을 추출하기 위해 위치나 인덱스를 지정하는 과정
- 문자, 리스트, 행렬, 배열 모두 위치나 인덱스를 가지고 있음

c(1,2,3,4,5,6,7)

1 2 3 4 5 6 7

순서



a[행, 열]

문법	의미
x[n]	x의 n번째 요소 n은 숫자 또는 셀의 이름을 뜻함
x[-n]	x에서 n번째 요소를 제외한 나머지
x[start:end]	x의 start부터 end까지의 값을 반환함
x[행, 열]	행에 해당하는 위치 값과 열에 해당하는 위치값

a=array(data = c(1, 2, 3, 4, 5, 6), dim = c(2, 3))

a[2]  
[1] 2

a[1,2]  
[1] 3

## Chapter 2. 데이터 구조와 함수 이해

### 2.4 인덱싱

#### 배열 인덱싱

- 목록이나 배열에서 특정 값을 추출하기 위해 위치나 인덱스를 지정하는 과정
- 문자, 리스트, 행렬, 배열 모두 위치나 인덱스를 가지고 있음

[,1] [,2] [,3]

[1,] 1 3 5

[2,] 2 4 6

```
a=array(data = c(1, 2, 3, 4, 5, 6), dim = c(2, 3))
```

```
a[2]  
[1] 2
```

```
a[1,2]  
[1] 3
```

```
a[1:2]  
[1] 1 2
```

```
a[1:3]  
[1] 1 2 3
```

```
a[1:2,2]  
[1] 3 4
```

## Chapter 2. 데이터 구조와 함수 이해

### 2.4 인덱싱

#### 배열 인덱싱

- 목록이나 배열에서 특정 값을 추출하기 위해 위치나 인덱스를 지정하는 과정
- 문자, 리스트, 행렬, 배열 모두 위치나 인덱스를 가지고 있음

[,1] [,2] [,3]

[1,] 1 3 5

[2,] 2 4 6

a[1:2,2:3]

[,1] [,2]

[1,] 3 5

[2,] 4 6

## Chapter 2. 데이터 구조와 함수 이해

### 2.4 인덱싱

#### 인덱싱 비교

```
my_list <- list("apple", 3.14, c(1, 2, 3), TRUE)
```

```
my_list[[3]][2]
```

```
[1] 2
```

`my_list[3]` → 여전히 리스트 형태를 유지하며, 해당 원소가 단독으로 반환

```
[[1]]
```

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

`my_list[[3]]` → 원소의 값 자체가 반환되는 것이 아니라 값을 나타내는 데이터 타입으로 반환

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

## Chapter 2. 데이터 구조와 함수 이해

### 2.4 인덱싱

#### 인덱싱 비교

```
my_vec <- c("apple", 3.14, c(1, 2, 3), TRUE)
```

```
my_vec[[3]][2]  
[1] NA
```

```
my_vec[3]
```

```
[1] "1"
```

```
my_vec[[3]]  
[1] "1"
```

## Chapter 2. 데이터 구조와 함수 이해

### 2.4 인덱싱

#### 인덱싱 비교

```
my_array <- array(c(5, 3, 1, 5, 7, 8, 10), dim = c(2, 3))
```

`my_array[1, 2]`: `my_array` 배열의 첫 번째 행, 두 번째 열에 접근합니다.

`my_array[2, 1]`: `my_array` 배열의 두 번째 행, 첫 번째 열에 접근합니다.

`my_array[1, ]`: `my_array` 배열의 첫 번째 행에 접근합니다.

`my_array[, 2]`: `my_array` 배열의 두 번째 열에 접근합니다.



## Chapter 2. 데이터 구조와 함수 이해

### 2.4 인덱싱

#### 인덱싱 비교

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

```
my_array[2, 3, 1]
```

```
my_array[3, , 2]
```

```
my_array[, 2:3, ]
```

```
my_array[2, 3, 1]
```

```
my_array[3, , 2]
```

```
my_array[, , 2]
```

## Chapter 2. 데이터 구조와 함수 이해

### 2.4 인덱싱

#### 인덱싱 비교

아래와 같은 배열을 만들고, 5와 16을 각각 인덱싱해 값을 추출하시오

, , 1 [,1] [,2] [,3] [,4] [,5]

[1,] 1 3 5 7 9

[2,] 2 4 6 8 10

, , 2 [,1] [,2] [,3] [,4] [,5]

[1,] 11 13 15 17 19

[2,] 12 14 16 18 20

## Chapter 2. 데이터 구조와 함수 이해

### 2.4 인덱싱

1) 아래의 결과가 나오도록 3개의 벡터를 생성하시오.

```
[1] "A" "B" "B" "B" "B" "A" "A" "A" "A" "A" "B" "B"
```

```
[1] 5000 12000 13000 8000 9000 3000 5000 4000 4500 6000 8000 8500
```

```
[1] 2500 5000 6000 5500 7000 4600 3000 2500 3400 4700 6400 4400
```

2) 1에서 32까지의 벡터를 생성하고, 4X8행렬을 만드시오.

3) 1에서 32까지의 벡터를 생성하고, 2X4X2X2 배열을 만드시오.

4) 위의 배열에서 20, 30을 각각 도출하시오.

5) 위의 4차원 배열에서 4차원 두 개의 행렬의 합을 구하시오.

# Chapter 2. 데이터 구조와 함수 이해

## 2.4 인덱싱

6) 아래의 두 배열을 생성하고, 해당 배열을 행을 기준으로 하나로 합치고 열을 기준으로 합쳐서 프린트 하시오.

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

	[,1]	[,2]	[,3]
[1,]	13	17	21
[2,]	14	18	22
[3,]	15	19	23
[4,]	16	20	24

7) 문제6에서 생성한 두 개의 배열 행과 열에 대한 이름을 임의로 부여해 행과 열의 이름을 가지는 두 배열을 만드시오.

## Chapter 2. 데이터 구조와 함수 이해

### 2.5 데이터 구조 파악

#### 테이블(Table 이란?)

- 일반적으로 범주형 변수의 분포를 요약한 빈도 테이블을 나타냄
- 변수 내의 다양한 범주 또는 수준의 개수를 표시함
- R의 table() 함수는 이러한 빈도표를 만드는 데 사용됨

```
data <- c("A", "B", "A", "C", "B", "A")  
frequency_table <- table(data)  
print(frequency_table)  
frequency_table[1]
```

인덱싱 순서	1	2	3
값	A	B	C
빈도	3	2	1

## Chapter 2. 데이터 구조와 함수 이해

### 2.5 데이터 구조 파악

#### 데이터 프레임(Dataframe 이란?)

- 데이터 프레임은 프로그래밍 및 데이터 분석에 일반적으로 사용되는 표 형식의 데이터 구조
- 행과 열로 구성된 다양한 형태를 가지고 있는 리스트의 집합
- 데이터 프레임에서 각 열은 변수 또는 특정 속성을 나타냄
- 각 행은 개별 관찰 또는 데이터 포인트를 나타냄
- 데이터 프레임은 다목적이며 숫자, 범주 및 텍스트 데이터를 포함하여 다양한 유형의 데이터를 처리할 수 있음

```
city <- c("Seoul", "Busan", "Daegu", "Seoul", "Busan", "Daegu", "Ulsan")  
pm25 <- c(18, 21, 21, 17, 8, 11, 25)
```

```
df <- data.frame(city = city, pm25 = pm25)
```

 데이터 프레임의 변수명

 데이터 프레임의 변수 값

## Chapter 2. 데이터 구조와 함수 이해

### 2.5 데이터 구조 파악

#### 데이터 프레임(Dataframe 이란?)

- 데이터 프레임은 프로그래밍 및 데이터 분석에 일반적으로 사용되는 표 형식의 데이터 구조
- 행과 열로 구성된 다양한 형태를 가지고 있는 리스트의 집합
- 데이터 프레임에서 각 열은 변수 또는 특정 속성을 나타냄
- 각 행은 개별 관찰 또는 데이터 포인트를 나타냄
- 데이터 프레임은 다목적이며 숫자, 범주 및 텍스트 데이터를 포함하여 다양한 유형의 데이터를 처리할 수 있음

```
> id<-c(1:5)
> gender<-c("M","F","F","M","M")
> major<-c("Eng","Math","Com","Eng","Busi")
> salary<-c(2500, 2800, 2500, 3000, 2600)
> survey<-data.frame(ID=id, Gender=gender, Major=major, salary=salary,
+                     stringsAsFactors = FALSE)
> survey
```

	ID	Gender	Major	salary
1	1	M	Eng	2500
2	2	F	Math	2800
3	3	F	Com	2500
4	4	M	Eng	3000
5	5	M	Busi	2600

## Chapter 2. 데이터 구조와 함수 이해

### 2.5 데이터 구조 파악

#### 데이터 프레임(Dataframe 이란?)

- 리스트와 배열 조작에서 사용하는 방법으로 데이터프레임 조작 가능
- 데이터 프레임의 모든 열은 길이가 같아야 함
- 데이터 프레임 크기가 큰 경우 내용의 일부를 확인하기 위해 head(), tail()함수를 사용 (n의 디폴트 값은 6)

```
> head(survey, n=3)      #survey의 내용을 앞에서 3행만큼 출력
  ID Gender Major Salary
1  1      M   Eng  2500
2  2      F  Math  2800
3  3      F   Com  2500
> tail(survey, n=3)      #survey의 내용을 뒤에서 3행만큼 출력
  ID Gender Major Salary
3  3      F   Com  2500
4  4      M   Eng  3000
5  5      M  Busi  2600
> survey$Salary          #Salary열을 벡터구조로 추출
[1] 2500 2800 2500 3000 2600
> survey[["Salary"]]     #Salary열을 벡터구조로 추출
[1] 2500 2800 2500 3000 2600
> survey["Major"]        #Major열 추출
Major
1   Eng
2  Math
3   Com
4   Eng
5  Busi
> survey[2]              #2열을 데이터 프레임 구조로 추출
Gender
1      M
2      F
3      F
4      M
5      M
> survey[[2]]            #2열을 벡터 구조로 추출
[1] "M" "F" "F" "M" "M"
```



## Chapter 2. 데이터 구조와 함수 이해

### 2.5 데이터 구조 파악

#### 데이터 프레임(Dataframe 이란?)

- 리스트와 배열 조작에서 사용하는 방법으로 데이터프레임 조작 가능
- \$는 데이터프레임의 특정 변수를 추가하거나 불러올 수 있음

```
> survey
  ID Major salary
1  1   Eng  2500
2  2  Math  2800
3  3   Com  2500
4  4   Eng  3000
5  5  Busi  2600
> survey$score=c(200,300,400,500,600)
> survey
  ID Major salary score
1  1   Eng  2500   200
2  2  Math  2800   300
3  3   Com  2500   400
4  4   Eng  3000   500
5  5  Busi  2600   600
```

```
> survey$ss=survey$salary+survey$score
> survey
  ID Major salary score  ss
1  1   Eng  2500   200 2700
2  2  Math  2800   300 3100
3  3   Com  2500   400 2900
4  4   Eng  3000   500 3500
5  5  Busi  2600   600 3200
```

## Chapter 2. 데이터 구조와 함수 이해

### 2.5 데이터 구조 파악

#### 데이터 프레임(Dataframe 이란?)

- 데이터 프레임의 형태를 파악하고, 잘못된 형태일 경우 변환 가능

변환할 변수=as.변환 값(변환할 변수)

summary(name\_age\_df) → 데이터 프레임의 변수 요약

```
name_age_df$Age=as.integer(name_age_df$Age)
name_age_df$Age=as.numeric(name_age_df$Age)
name_age_df$Age=as.factor(name_age_df$Age)
name_age_df$Age=as.logical(name_age_df$Age)
name_age_df$Age=as.character(name_age_df$Age)
```

```
> name_age_df
  name Age
1  John  28
2  Jane  32
3 smith  45
4   Doe  22
```

## Chapter 2. 데이터 구조와 함수 이해

### 2.5 데이터 구조 파악

#### 데이터 프레임(Dataframe 이란?)

- 변수의 이름 변경

```
names(df)
```

```
names(df)[names(df)=="데이터프레임의 변수명"]="변환할 변수명"
```

```
names(name_age_df)[names(name_age_df)=="name"]="Name"  
name_age_df
```

	Name	Age
1	John	28
2	Jane	32
3	Smith	45
4	Doe	22

## Chapter 2. 데이터 구조와 함수 이해

### 2.5 데이터 구조 파악

#### 데이터 프레임(Dataframe 이란?)

- 특정 조건에 맞게 인덱싱 할 수 있음

```
> survey[c(1,2)]           #1,2열 추출
  ID Gender
1  1      M
2  2      F
3  3      F
4  4      M
5  5      M
> survey[c(-1,-2)]         #1,2열을 제외한 나머지 열 추출
  Major Salary
1   Eng   2500
2  Math   2800
3   Com   2500
4   Eng   3000
5  Busi   2600
> survey[survey$Gender=="F",] #Gender=F인 행만 추출
  ID Gender Major Salary
2  2      F  Math   2800
3  3      F   Com   2500
> survey[survey$Major=="Eng"&survey$Salary>2600,] #Major이 Eng이고 salary>2600인 행추출
  ID Gender Major Salary
4  4      M   Eng   3000
```

# Chapter 2. 데이터 구조와 함수 이해

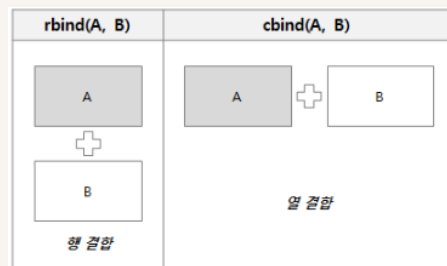
## 2.5 데이터 구조 파악

### 데이터 프레임(Dataframe 이란?)

- `nrow()`, `ncol()`, `names()`, `rownames()`, `colnames()`, `dim()`을 이용하여 행과 열의 정보를 얻을 수 있음
- `sort()`, `order()`을 이용하여 자료를 정렬할 수 있음
- `$`를 통해 데이터 프레임에 새로운 변수를 추가하거나, 기존에 있는 변수의 값을 변환하고, `rbind()`, `cbind()`를 이용해 두 개의

데이터 프레임을 결합하거나 병합할 수 있음

```
> id<-c(1:5)
> gender<-c("M","F","F","M","M")
> major<-c("Eng","Math","Com","Eng","Busi")
> salary<-c(2500,2800,2500,3000,2600)
> survey<-data.frame(ID=id,Gender=gender,Major=major,Salary=salary,
+                     stringsAsFactors = FALSE)
> survey
  ID Gender Major Salary
1  1     M   Eng  2500
2  2     F  Math  2800
3  3     F   Com  2500
4  4     M   Eng  3000
5  5     M  Busi  2600
> nrow(survey)           #행의 수
[1] 5
> ncol(survey)           #열의 수
[1] 4
> dim(survey)            #데이터 프레임 차원
[1] 5 4
> names(survey)          #데이터 프레임 변수 이름
[1] "ID" "Gender" "Major" "Salary"
> colnames(survey)       #데이터 프레임 열 이름
[1] "ID" "Gender" "Major" "Salary"
> rownames(survey)       #데이터 프레임 행 이름
[1] "1" "2" "3" "4" "5"
> sort(survey$Salary)    #salary를 크기 순서대로 정렬
[1] 2500 2500 2600 2800 3000
> order(survey$Salary)   #정렬된 salary 값의 원래 위치
[1] 1 3 5 2 4
> survey[c(order(survey$Salary)),] #1,3,5,2,4행 순서대로 나열
  ID Gender Major Salary
1  1     M   Eng  2500
3  3     F   Com  2500
5  5     M  Busi  2600
2  2     F  Math  2800
4  4     M   Eng  3000
```



```
> survey$grade<-c(3.5,3.7,4.2,3.3,2.9) #새로운 변수 추가
> survey
  ID Gender Major Salary grade
1  1     M   Eng  2500   3.5
2  2     F  Math  2800   3.7
3  3     F   Com  2500   4.2
4  4     M   Eng  3000   3.3
5  5     M  Busi  2600   2.9
> survey1<-data.frame(ID=id[1:3],Gender=gender[1:3],Major=major[1:3],
+                      Salary=salary[1:3],stringsAsFactors = FALSE)
> survey1
  ID Gender Major Salary
1  1     M   Eng  2500
2  2     F  Math  2800
3  3     F   Com  2500
> survey2<-data.frame(ID=id[4:5],Gender=gender[4:5],Major=major[4:5],
+                      Salary=salary[4:5],stringsAsFactors = FALSE)
> survey2
  ID Gender Major Salary
1  4     M   Eng  3000
2  5     M  Busi  2600
> survey3<-rbind(survey1,survey2) #rbind()를 통해 행 결합
> survey3
  ID Gender Major Salary
1  1     M   Eng  2500
2  2     F  Math  2800
3  3     F   Com  2500
4  4     M   Eng  3000
5  5     M  Busi  2600
> Job<-c("office","profession","technician")
> survey4<-cbind(survey1, Job) #cbind()를 통해 열 결합
> survey4
  ID Gender Major Salary      Job
1  1     M   Eng  2500   office
2  2     F  Math  2800 profession
3  3     F   Com  2500  technician
```

## Chapter 2. 데이터 구조와 함수 이해

### 2.5 데이터 구조 파악

#### 데이터 프레임(Dataframe 이란?)

- 특정 데이터만 사용하고 싶을 경우
- 조건에 맞는 데이터를 선택할 수 있음

`subset(데이터 프레임, 조건, select= c(도출하고 싶은 변수 명1, 도출하고 싶은 변수 명2))`

```
> age_gt_24 <- subset(name_age_df, name_age_df$Age > 24)
> age_gt_24
  LastName FirstName Age
5   Kim Min      jun  35
6  Park Min      jun  40
7   Kim Ji     young  34
8   Park Ji     young  35
```

```
> age_gt_24_name_age_only <- subset(name_age_df, name_age_df$Age > 24, select = c("LastName", "Age"))
> print(age_gt_24_name_age_only)
  LastName Age
5   Kim Min  35
6  Park Min  40
7   Kim Ji   34
8   Park Ji  35
```

## Chapter 2. 데이터 구조와 함수 이해

### 2.5 데이터 구조 파악

#### 데이터 프레임(Dataframe 이란?)

- 특정 데이터만 사용하고 싶을 경우
- 조건에 맞는 데이터를 선택할 수 있음

`subset(데이터 프레임, 조건, select= c(도출하고 싶은 변수 명1, 도출하고 싶은 변수 명2))`

```
> subset(survey, Major=="Com",c(Gender,salary))    #Gender와 salary에 대해 Major=Com인 행 추출
  Gender salary
3      F   2500
> survey$Gender<-NULL    #Gender열 삭제
> survey
  ID Major salary
1  1   Eng   2500
2  2  Math   2800
3  3   Com   2500
4  4   Eng   3000
5  5  Busi   2600
```

## Chapter 2. 데이터 구조와 함수 이해

### 2.5 데이터 구조 파악

#### 데이터 프레임(Dataframe 이란?)

- 데이터 프레임의 변수에 특정 기호가 존재할 때, 이를 구분해 두개의 변수로 나눌 수 있음
- 이때 `separate` 함수 사용

`library(tidyr) # 사용 패키지`

`separate(데이터 프레임, col = " 지정한 변수 이름 ", into = c("생성변수1", " 생성변수2"), sep = " 나눌 기준기호")`

```
> name_age_df <- data.frame(
+   Name = c("Kim Cheol-soo", "Lee Cheol-soo", "Kim Young-hee", "Lee Young-hee",
+           "Kim Min-jun", "Park Min-jun", "Kim Ji-young", "Park Ji-young"),
+   Age = c(20, 24, 21, 24, 35, 40, 34, 35),
+   stringsAsFactors = FALSE
+ )

> name_age_df <- separate(name_age_df, col = "Name", into = c("LastName", "FirstName"), sep = "-")
> print(name_age_df)
  LastName FirstName Age
1 Kim Cheol      soo  20
2 Lee Cheol      soo  24
3 Kim Young      hee  21
4 Lee Young      hee  24
5   Kim Min       jun  35
6  Park Min       jun  40
7   Kim Ji      young  34
8  Park Ji      young  35
```

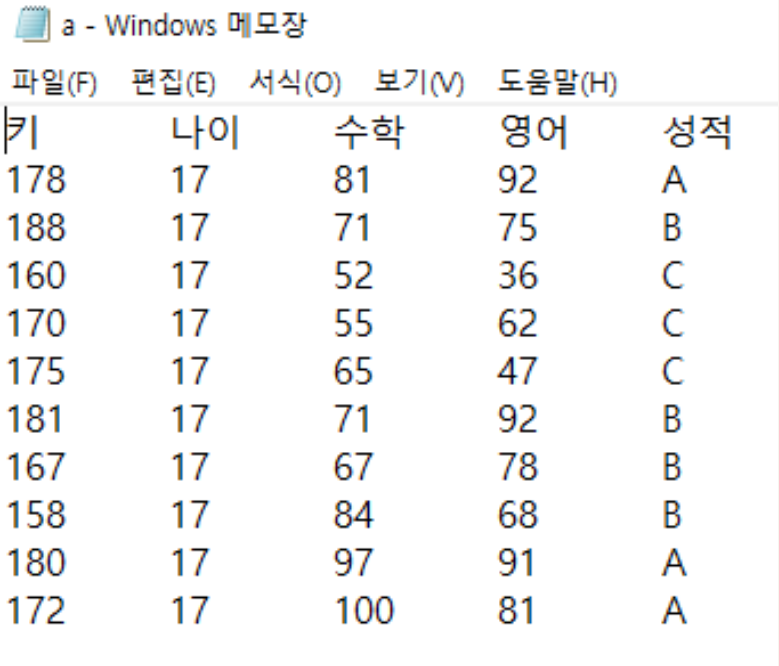


# Chapter 2. 데이터 구조와 함수 이해

## 2.6 데이터 불러오기

### TEXT

- 텍스트 파일은 데이터를 저장하고 표현하기 위해 간단하고 널리 사용되는 형식
- 텍스트 파일의 데이터는 일반적으로 각 데이터 포인트가 구분 기호(예: 쉼표 또는 탭)로 구분된 일반 텍스트로 저장됨
- 텍스트 파일의 주요 이점은 단순성과 다양한 프로그래밍 언어 및 소프트웨어 응용 프로그램과의 호환성이 좋음
- 복잡한 데이터 구조에 대한 지원 부족
- 데이터 조작 및 분석 기능이 제한됨
- 고급 데이터 작업을 위해 수동 처리가 필요함



파일(F)	편집(E)	서식(O)	보기(V)	도움말(H)
키	나이	수학	영어	성적
178	17	81	92	A
188	17	71	75	B
160	17	52	36	C
170	17	55	62	C
175	17	65	47	C
181	17	71	92	B
167	17	67	78	B
158	17	84	68	B
180	17	97	91	A
172	17	100	81	A

## Chapter 2. 데이터 구조와 함수 이해

### 2.6 데이터 불러오기

#### Excel

- 특히 .xlsx 형식의 Excel 파일은 Microsoft Excel에서 만들고 사용하는 스프레드시트 파일
- Excel 파일은 데이터 구성, 조작, 시각화 및 분석을 위한 포괄적인 기능 세트를 제공함
- 복잡한 수식, 조건부 서식, 그래픽 표현 및 다양한 데이터 유형을 지원함
- Excel 파일은 사용자 친화적인 인터페이스와 광범위한 기능을 제공하여 기본 및 고급 데이터 관리 작업에 모두 적합함
- 그러나 Excel 파일은 크기가 상대적으로 큼
- 타사 소프트웨어와의 호환성 문제

	A	B	C	D	E
1	키	나이	수학	영어	성적
2	178	17	81	92	A
3	188	17	71	75	B
4	160	17	52	36	C
5	170	17	55	62	C
6	175	17	65	47	C
7	181	17	71	92	B
8	167	17	67	78	B
9	158	17	84	68	B
10	180	17	97	91	A
11	172	17	100	81	A

# Chapter 2. 데이터 구조와 함수 이해

## 2.6 데이터 불러오기

### CSV(Comma-Separated Values)

- 파일은 테이블 형식 데이터 저장 및 교환에 일반적으로 사용되는 특정 유형의 텍스트 파일 형식
- CSV 파일에서 각 행은 데이터 레코드를 나타내며 행 내의 각 필드는 쉼표 또는 기타 지정된 구분 기호로 구분됨
- CSV 파일은 스프레드시트 소프트웨어 및 데이터베이스 응용 프로그램에서 광범위하게 지원되므로 데이터 공유 및 상호 운용성을 위해 많이 사용 가능함
- 데이터를 행과 열로 구성하기 위한 기본 구조를 제공하지만 복잡한 수식이나 서식 옵션은 지원하지 않음
- 복잡한 데이터 구조 또는 수식에 대한 제한된 지원
- 고급 서식 옵션이 부족함

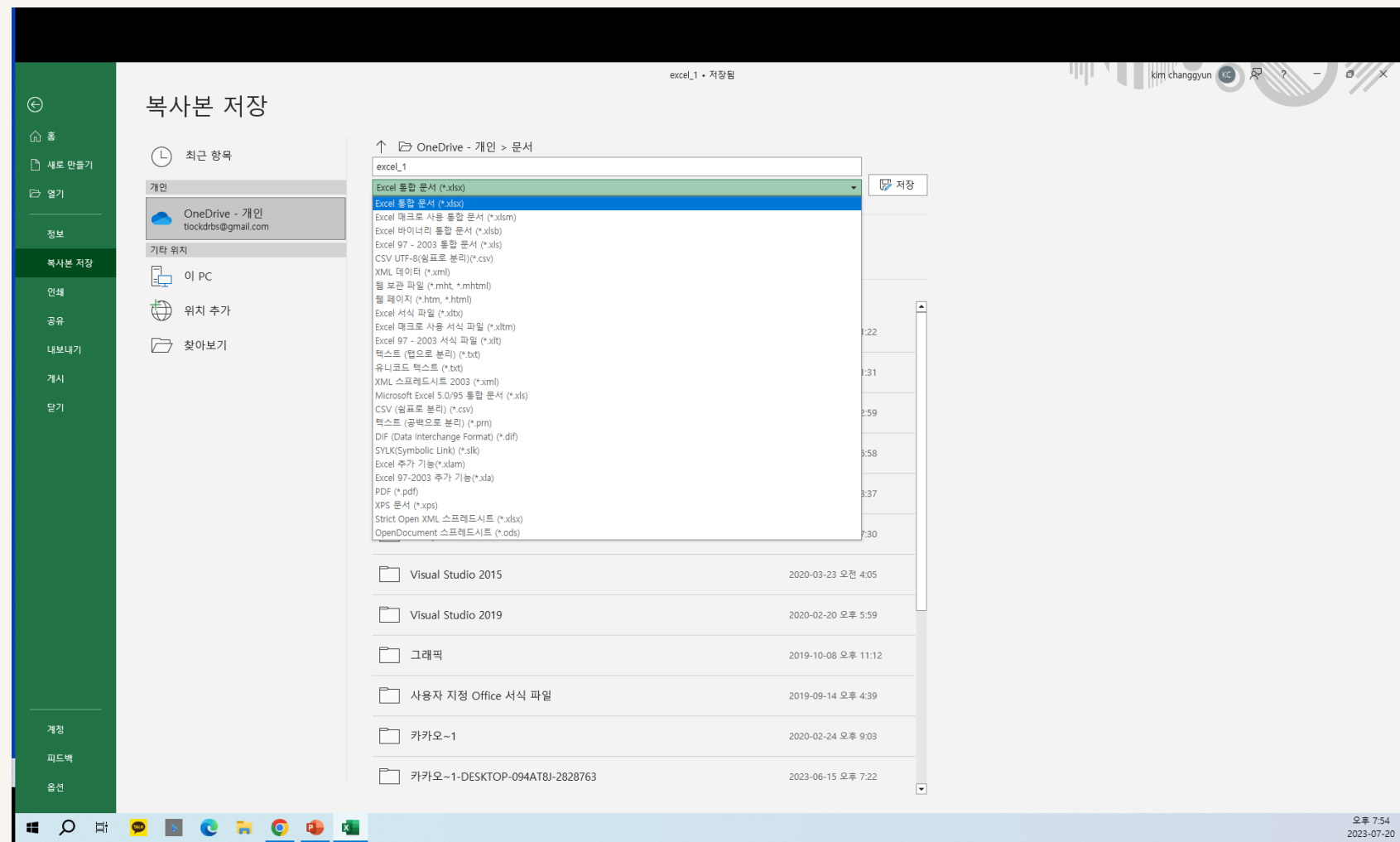
	A	B	C	D	E
1	키	나이	수학	영어	성적
2	178	17	81	92	A
3	188	17	71	75	B
4	160	17	52	36	C
5	170	17	55	62	C
6	175	17	65	47	C
7	181	17	71	92	B
8	167	17	67	78	B
9	158	17	84	68	B
10	180	17	97	91	A
11	172	17	100	81	A

# Chapter 2. 데이터 구조와 함수 이해

## 2.6 데이터 불러오기

### Excel to CSV(Comma-Separated Values)

- 다른이름으로 저장 or 복사본 저장

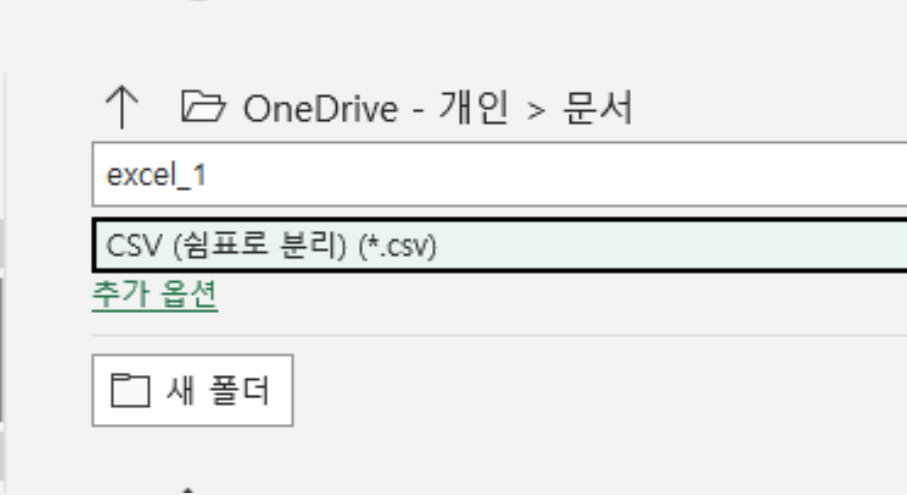
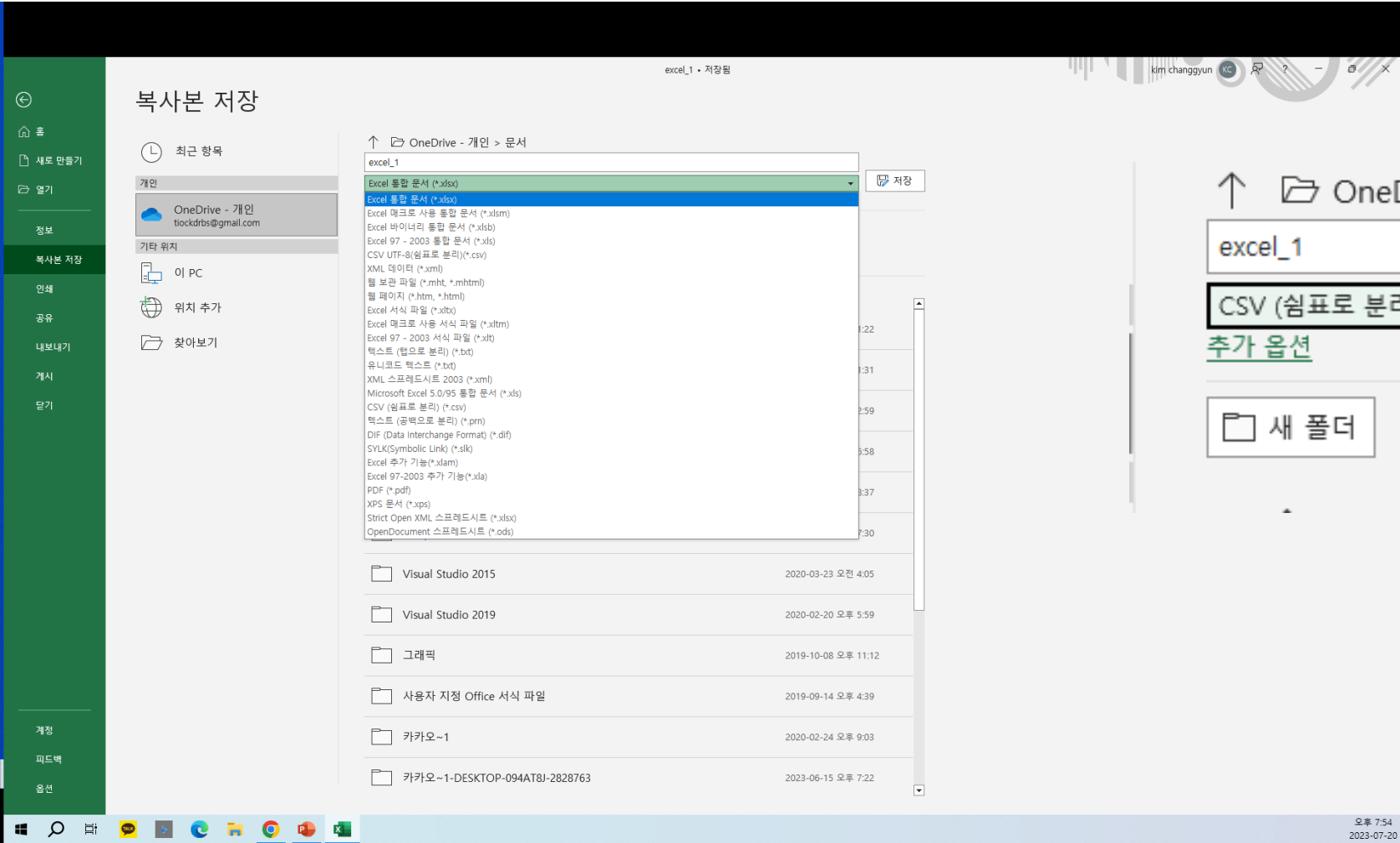


# Chapter 2. 데이터 구조와 함수 이해

## 2.6 데이터 불러오기

### Excel to CSV(Comma-Separated Values)

- 저장 탭에서 csv (쉼표로 분리) 선택

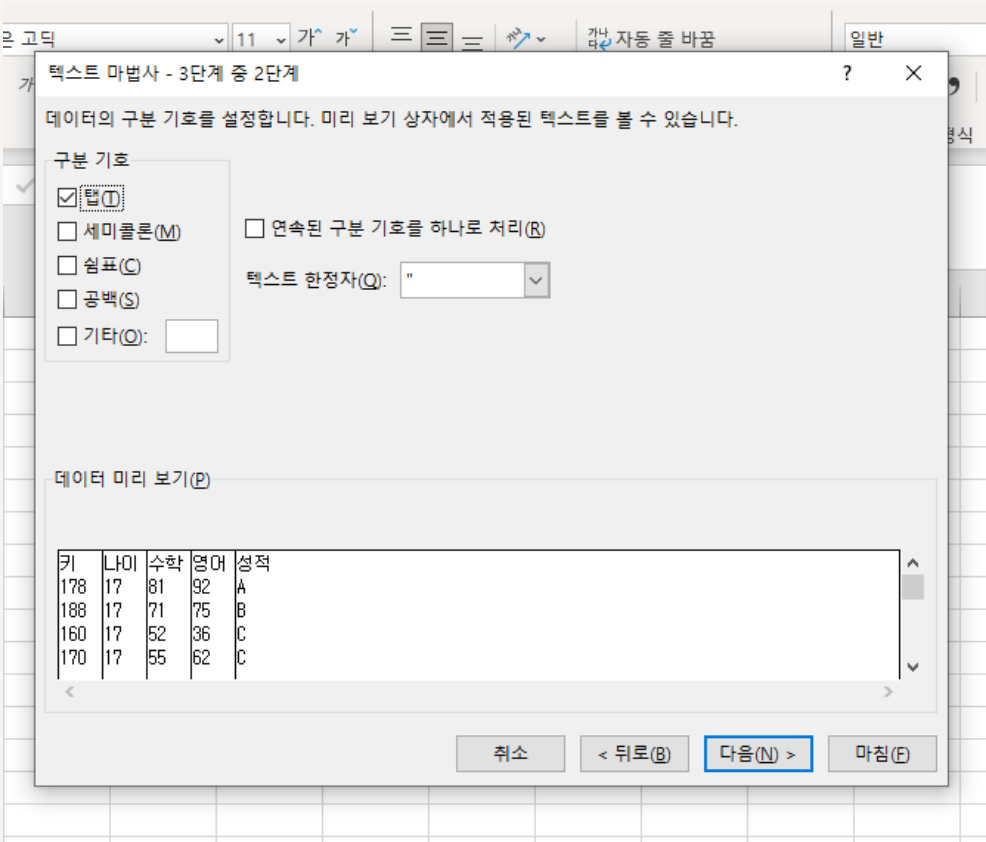
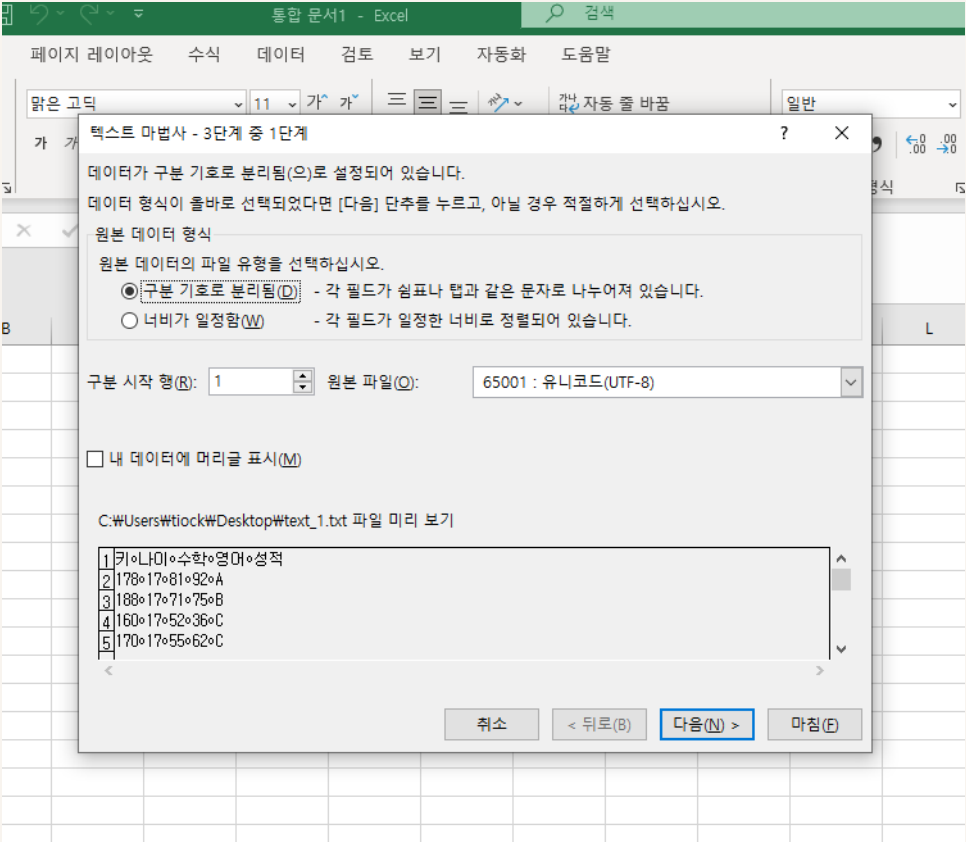


# Chapter 2. 데이터 구조와 함수 이해

## 2.6 데이터 불러오기

### TEXT to CSV(Comma-Separated Values)

- Excel → 열기 → Text파일 선택

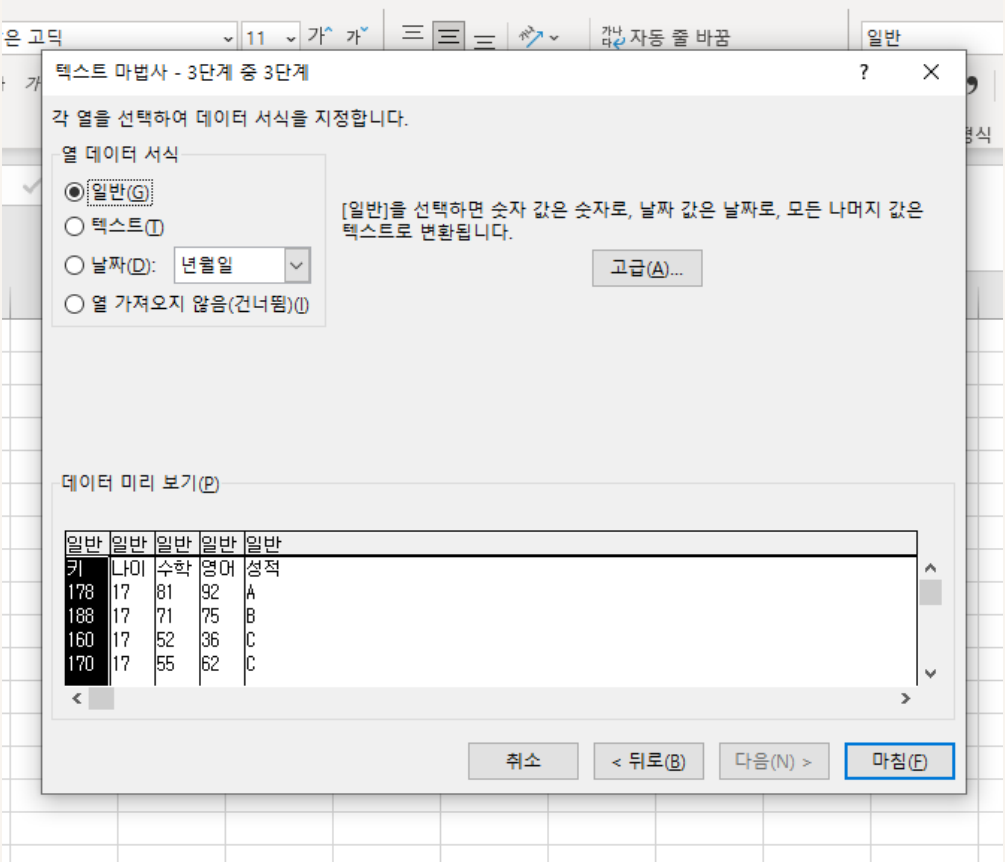


# Chapter 2. 데이터 구조와 함수 이해

## 2.6 데이터 불러오기

### TEXT to CSV(Comma-Separated Values)

- Excel → 열기 → Text파일 선택



## Chapter 2. 데이터 구조와 함수 이해

### 2.6 데이터 불러오기

#### CSV(Comma-Separated Values)

```
data <- read.csv("data.csv", header = ?, stringAsFactors = ?, fileEncoding = ?)
```

- **header** = 데이터 프레임의 첫 행에 변수명이 있는지 없는지에 따라 TRUE와 FALSE로 구분됨
- **stringAsFactors** = 문자열 변수를 요소로 변환할지 여부를 설정함 TRUE와 FALSE로 구분됨
- **encoding** = R은 영어 기반으로 영어 이외의 단어들이 들어 갔을 때, 오류가 발생할 수 있으므로 영어 이외의 단어를 인식할 수 있도록 인코딩 하는 방식(주로 한국어는 cp949, EUC-KR, UTF-8)세 가지로 저장됨

```
write.csv(데이터 프레임, "저장할 경로", row.names=FALSE/TRUE)
```



# Chapter 2. 데이터 구조와 함수 이해

## 2.6 데이터 불러오기

- 1. 0 부터 10까지 포함하는 벡터를 vec함수에 저장하시오
- 2. 0 부터 10까지 포함하는 벡터 vec과 10부터 20까지 포함하는 벡터 vec2를 열 기준으로 합쳐 matrix 형태로 만드시오
- 3. 1번과 2번에서 생성한 vec, vec2를 열 단위로 합쳐 matrix 형태로 만들고 이를 matr에 저장하고 4행에 2열의 값을 100으로 변환하시오
- 4. 20 부터 30 까지 포함하는 벡터 vec3를 생성하고 vec, vec2, vec3를 각각 5X2 를 가지는 배열을 만드시오
- 5. 아래 표와 같은 데이터 프레임을 생성하고 이름을 score라고 지정하시오
- 6. 생성된 score 데이터 프레임 에서 국어 영어 수학 평균값인 mean변수를 생성하고 mean 변수가 80점 이상인 사람의 이름만 뽑으시오

이름	국어	영어	수학	물리	화학	생물	지구과학
Kim	100	60	50	40	40	40	40
Park	80	40	100	30	30	20	30
Lee	80	80	80	20	20	10	50

## Chapter 2. 데이터 구조와 함수 이해

### 2.6 데이터 불러오기

7) 아래와 같이 데이터 프레임으로 저장하고, 아래와 같이 만드시오.

	병원	진료금액	처방전금액
1	A	5000	2500
2	B	12000	5000
3	B	13000	6000
4	B	8000	5500
5	B	9000	7000
6	A	3000	4600
7	A	5000	3000
8	A	4000	2500
9	A	4500	3400
10	A	6000	4700
11	B	8000	6400
12	B	8500	4400

8) 진료금액과 처방전을 합한 값을 만드시오.

9) 합한 값을 기존의 데이터 프레임에 추가하고 데이터 프레임을 csv파일로 저장하시오.

10) 저장한 csv파일을 다시 불러와서 프린트 하시오.

## Chapter 2. 데이터 구조와 함수 이해

### 2.6 데이터 불러오기

11) mtcars 데이터를 불러오고 mpg, cyl, disp, wt, qsec 변수를 하나의 car라는 데이터 프레임으로 저장하시오.

12) car 데이터 프레임에서 wt와 qsec변수를 합하여 qw라는 새로운 변수를 생성하시오.

13) 12번에서 생성한 각 변수들의 형식을 파악하시오.

14) 각 변수들의 평균, 표준편차, 최소값, 최대값, 사분위수(0.25, 0.75)를 계산하시오.

## Chapter 2. 데이터 구조와 함수 이해

### 2.6 데이터 불러오기

15) mtcars의 모든 데이터의 합을 나타내는 total\_car라는 변수를 생성하시오.

#데이터 프레임의 모든 변수를 더한 값 rowSums( )

16) mtcars의 index값을 car\_name이라는 변수로 저장하시오.

#데이터 프레임의 index이름을 전부 불러오는 함수 rownames( )

17) 위에서 생성한 변수를 사용하여 아래와 같은 값을 가지는 car\_model이라는 변수를 생성하시오.

```
car_model
Mazda RX4-328.98
Mazda RX4 Wag-329.795
Datsun 710-259.58
Hornet 4 Drive-426.135
Hornet Sportabout-590.31
Valiant-385.54
Duster 360-656.92
Merc 240D-270.98
Merc 230-299.57
Merc 280-350.46
Merc 280C-349.66
Merc 450SE-510.74
```

## Chapter 2. 데이터 구조와 함수 이해

### 2.6 데이터 불러오기

18) 생성된 `car_model`이라는 변수를 “-” 기준으로 분해해 `car`, `model`이라는 변수로 각각 다시 만드시오.

19) 18번 까지 만들어진 데이터 프레임에서 `mpg`가 20보다 크고, `disp`가 160이상인 데이터들만 도출하시오.  
(이때 `mpg`, `disp`, `car`, `model`) 변수만 도출하시오.