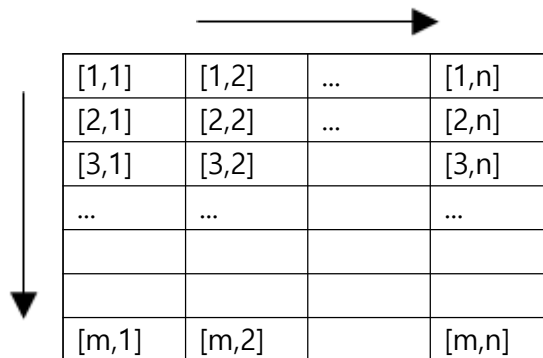


## 제4주 : 행렬과 배열

### 4.1. 행렬(Matrix)

- 행렬: 값들이 행과 열로 배열된 자료구조



[1,1]	[1,2]	...	[1,n]
[2,1]	[2,2]	...	[2,n]
[3,1]	[3,2]		[3,n]
...	...		...
[m,1]	[m,2]		[m,n]

- 행렬의 구성: 원소값들이 동일한 기본 자료형으로만 구성된다.
- 행렬 생성하기

#### ① matrix()함수를 이용한 방법

```
객체 이름 <- matrix(data = NA, nrow = 1, ncol = 1,  
                    byrow = FALSE, dimnames = NULL)
```

- data: 행렬로 재구성할 자료 벡터
- nrow: 행의 개수
- ncol: 열의 개수
- byrow = FALSE: 데이터를 행 방향으로 배치할지 여부
- dimnames: 행과 열의 이름 list

#### ② 벡터에 차원(dim())을 설정하는 방법

```
dim(벡터) <- c(행의 개수, 열의 개수)
```

#### 사용 예

```
> tmp <- 1:12  
> tmp  
[1] 1 2 3 4 5 6 7 8 9 10 11 12  
➔행렬에 배치할 벡터를 준비  
> M1 = matrix(tmp, nrow = 3)
```

```

> M1
      [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
→ 위에서 만든 벡터를 행의 개수가 3인 행렬에 재배치
→ 1열부터 자료를 채워감
> M2 <- matrix(tmp, nrow = 3, byrow = TRUE)
> M2
      [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
→ 위에서 만든 벡터를 행의 개수가 3인 행렬에 재배치
→ 1행부터 자료를 채워감 (byrow=TRUE)
> M3 <- tmp
> dim(M3) <- c(3, 4)
> M3
      [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
→ 벡터의 원소들을 행의 개수가 3인 행렬로 배치. 1열부터 자료를 채워감

```

### ③ cbind(), rbind()를 이용한 방법

- cbind()는 벡터를 열(column)단위로 합치고, rbind()는 벡터를 행(row)단위로 합친다.

#### 사용예

```

> v1 <- c(1, 2, 3, 4)
> v2 <- c(5, 6, 7, 8)
> v3 <- c(9, 10, 11, 12)
> cbind(v1, v2, v3)
      v1 v2 v3
[1,]  1  5  9

```

```

[2,] 2 6 10
[3,] 3 7 11
[4,] 4 8 12
→v1, v2, v3를 열로 묶어 4행3열짜리 행렬 생성
> rbind(v1, v2, v3)
      [,1] [,2] [,3] [,4]
v1      1    2    3    4
v2      5    6    7    8
v3      9   10   11   12
→v1, v2, v3를 행으로 묶어 3행4열짜리 행렬 생성

```

## 4.2. 행렬의 유용한 함수들

- 행렬 간 산술연산: +, -, %\*%, t(), solve()

```

> A <- matrix(1:6, 2, 3)
> B <- matrix(rep(1:2, times = 3), 2, 3)
> A
      [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> B
      [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    2    2
> A + B # 원소별 합을 계산한다.
      [,1] [,2] [,3]
[1,]    2    4    6
[2,]    4    6    8
> A - B # 원소별 차를 계산한다.
      [,1] [,2] [,3]
[1,]    0    2    4
[2,]    0    2    4
> A * B # 원소별 곱을 계산한다.
      [,1] [,2] [,3]
[1,]    1    3    5
[2,]    4    8   12

```

```

> t(B) # 행렬의 전치(transpose)를 계산한다.
      [,1] [,2]
[1,]    1    2
[2,]    1    2
[3,]    1    2
> A %*% t(B) # 행렬곱을 계산한다.
      [,1] [,2]
[1,]    9   18
[2,]   12   24
> C <- matrix(1:4, 2, 2)
> solve(C) # 역행렬을 계산한다.
      [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5
> C %*% solve(C)
      [,1] [,2]
[1,]    1    0
[2,]    0    1
> solve(C) %*% C
      [,1] [,2]
[1,]    1    0
[2,]    0    1
> solve(C, C) # solve(A, B)는 solve(A) %*% B와 결과가 같다.
      [,1] [,2]
[1,]    1    0
[2,]    0    1

```

- 행과 열에 이름 붙이기: `rownames()`, `colnames()`

```

> set.seed(123)
> A <- matrix(round(rnorm(5 * 3, mean = 50, sd = 10))), 5, 3)
> colnames(A)
NULL
> colnames(A) <- c("math", "eng", "science")
> rownames(A) <- paste("S", 1:5, sep = "")
> A

```

	math	eng	science
S1	44	67	62
S2	48	55	54
S3	66	37	54
S4	51	43	51
S5	51	46	44

- 각 행별(또는 각 열별)로 여러 계산을 한 번에 하기: `apply()`
  - `apply()`는 행렬과 배열에 적용하며, 특정 차원별로 원하는 함수를 적용할 수 있도록 해준다. 결과값은 벡터, 행렬, 배열, 리스트 등이다.

사용법:

```
apply(X, MARGIN, FUN, ...)
```

- X: 행렬 또는 배열
- MARGIN: 1이면 각 행 별로 함수FUN을 적용, 2이면 각 열 별로 함수 FUN을 적용한다.
- FUN: 적용할 함수
- ...: FUN의 추가적인 전달인자

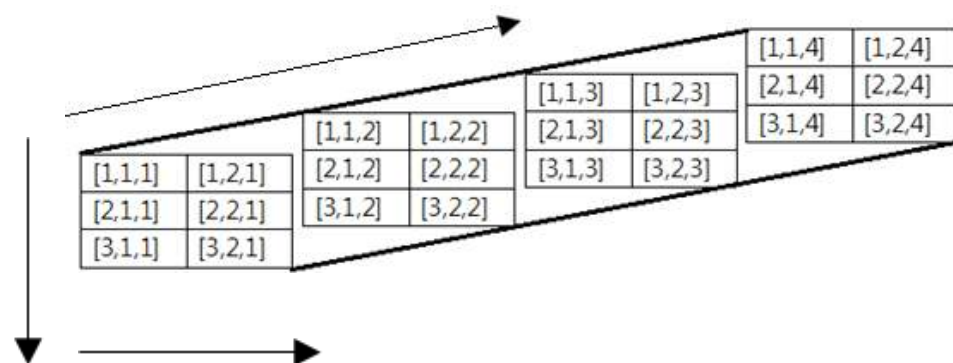
```
> v1 <- 1:4
> v2 <- 5:8
> v3 <- 9:12
> M1 <- cbind(v1, v2, v3)
> M1
      v1 v2 v3
[1,]  1  5  9
[2,]  2  6 10
[3,]  3  7 11
[4,]  4  8 12
> apply(M1, 1, mean)
[1] 5 6 7 8
> apply(M1, 2, mean)
      v1  v2  v3
2.5  6.5 10.5
```

```
> apply(M1, 2, diff)
      v1 v2 v3
[1,]  1  1  1
[2,]  1  1  1
[3,]  1  1  1
```

### 4.3. 배열(Array)

- 자료값들이 다차원의 방향으로 배열된 자료구조.  
행렬은 2차원 배열이다.

3차원 배열의 예:



- 배열의 구성: 원소값들이 동일한 기본 자료형으로만 구성된다.
- 배열 생성하기

#### ① array()함수를 이용한 방법

```
객체 이름 <- array(data = NA, dim=length(data), dimnames = NULL)
```

- data: 배열로 재구성할 자료 벡터
- dim: 각 차원의 크기를 정의하는 벡터  
예: c(2, 5, 10): 배열 전체는 3차원이고, 1차원의 크기는 2, 2차원의 크기는 5, 3차원의 크기는 10. 전체 원소의 개수는  $2 \times 5 \times 10 = 100$ 개

- dimnames: 각 차원의 이름 list

#### ④ 벡터에 차원(dim())을 설정하는 방법

```
dim(벡터) <- c(차원1의 크기, 차원2의 크기, 차원3의 크기, ...)
```

## 사용 예

```
> arr <- array(1:3, c(2, 4))
> arr
```

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

→ 2행4열의 배열을 만들고 원소들은 1, 2, 3 으로 채운다 (입력된 자료벡터 1:3의 길이가 부족할 경우에는 반복재사용한다.)

→ 생성된 배열은 변수 arr이 가리킨다

```
> arr[1,]
```

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

→ 배열arr의 1행의 원소 반환

```
> arr[,3]
```

```
[1] 2 3
```

→ 배열arr의 3열의 원소 반환

```
> dimnamearr <- list(c("1st", "2nd"), c("1st", "2nd", "3rd", "4th"))
```

→ 배열의 행과 열에 이름을 지정.

→ list(c(), c(), c(), ……)의 형태로, 위의 예에서는 첫 번째 전달된 벡터가 행의 이름, 두번째 전달된 벡터가 열의이름

```
> arr2 <- array(1:3, c(2, 4), dimnames = dimnamearr)
```

	1st	2nd	3rd	4th
1st	1	3	2	1
2nd	2	1	3	2

```
> arr2["1st", ]
```

```
1st 2nd 3rd 4th
```

```
1 3 2 1
```

```
> arr2[, "3rd"]
```

```
1st 2nd
```

```
2 3
```