3. 배열(array)

(1) 배열이란?

통계적 관점의 R에서 행렬의 **행(row)**은 다양한 사람과 같은 실험대상에 해당하고, **열(column)**은 체온이나 혈압 같은 변수에 해당한다. 그래서 행렬은 2차원 구조이다. 하지만 만약 데이터를 주기별로 가져온다면, 하나의 데이터에서 **시간(주기)**은 행과 열에 이어 세 번째 차원이 되는 것이다. R에서 이런 데이터 구조를 '**배열**'이라고 한다.

구분	통계학		데이터분석개론		← 측정요소
	중간고사	기말고사	중간고사	기말고사	← 측정시기
Kim	81	89	70	93	
Lee	78	90	87	88	
Park	94	96	82	84	
Jung	85	80	93	98	

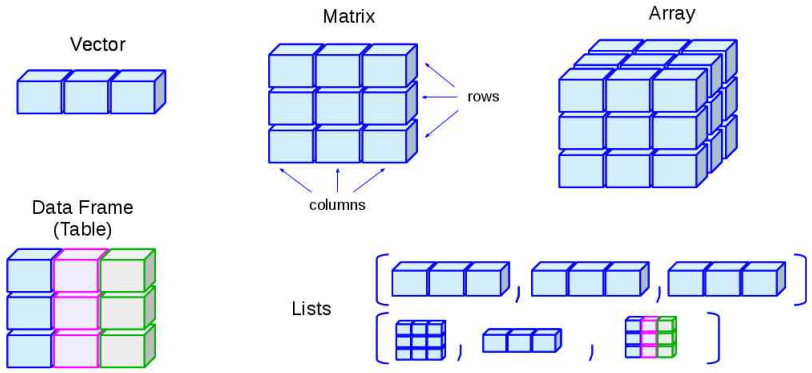
대상 →

이 자료구조를 두 개의 행렬로 표현할 수 있는 배열이다.

구분	통계학	
	중간고사	기말고사
Kim	81	89
Lee	78	90
Park	94	96
Jung	85	80

구분	데이터분석개론	
	중간고사	기말고사
Kim	70	93
Lee	87	88
Park	82	84
Jung	93	98

행렬이 2차원 데이터라면 **배열**(array)은 다차원 데이터, 즉 행렬을 여러 장을 쌓아 올린 형태이다. 예를 들면, 3×3 차원의 데이터를 행렬로 표현한다면 **3×3×3 차원**의 데이터는 **배열**로 표현한다. 이는 **3×3 행렬이 3개의 층으로 이루어진 구조**, 즉 행렬이 다수의 층으로 쌓인 구조이다.



[그림] 자료구조[1)]

(2) 배열을 생성하는 방법: 배열생성함수 array( )

```
array(
    vector,              # 데이터를 저장한 벡터
    dim=c(행의 개수, 열의 개수, 층(페이지)의 개수),  # 배열의 차원. 이 값을 지정하지 않으면 1차원 배열(벡터)이 생성된다.
    dimnames = list(c("1행", "2행", "3행"), c("1열", "2열", "3열"), c("1층", "2층", "3층"))
# 차원별 이름, 행렬과 동일하게 list로 묶어주며, 1번째 요소는 행, 2번째 요소는 열, 3번째 요소는 층의 이름을 뜻한다.
      )
```

예를 들어, **3×4 행렬**은 dim에 c(3, 4)를 지정하여 생성한다.

```
array(1:12, dim=c(3, 4))        #   byrow=F 가 기본값(default).
     [ ,1] [ ,2] [ ,3] [ ,4]
[1, ]    1    4    7   10
[2, ]    2    5    8   11
[3, ]    3    6    9   12
array(1:12, dim=c(3, 4, 1))
```

함수 array( )로 행렬을 만들 수 있으나 **byrow 옵션이 없기 때문에 불편한 점**이 있다. 또한 함수 dim( )으로도 행렬과 배열을 만들 수 있으나, byrow 옵션이 없는 것은 동일하다.

```
x <- 1:9
y <- 1:24
dim(x) = c(3, 3) ; x
dim(y) = c(4, 3, 2) ; y
```

이제 **2×2×3 차원**의 배열을 만들어보자.

```
x <- array(c(-1,0,3,5,-3,4,1,0,7,-2,1,3), dim=c(2, 2, 3)); x
, , 1
     [ ,1] [ ,2]
[1, ]   -1    3
[2, ]    0    5
```

1) 이미지 출처: http://venus.ifca.unican.es/Rintro/dataStruct.html

```
, , 2
     [ ,1] [ ,2]
[1, ]   -3    1
[2, ]    4    0
, , 3
     [ ,1] [ ,2]
[1, ]    7    1
[2, ]   -2    3
```

```
x <- array(c(-1,0,3,5,-3,4,1,0,7,-2,1,3), dim=c(2, 2, 3), dimnames = list(c("1행", "2행"), c("1열", "2열"), c("1층", "2층", "3층"))) ; x
```

```
, , 1층

    1열 2열
1행  -1   3
2행   0   5

, , 2층

    1열 2열
1행  -3   1
2행   4   0

, , 3층

    1열 2열
1행   7   1
2행  -2   3
```

대상 →

구분	통계학		데이터분석개론		← 측정요소
	중간고사	기말고사	중간고사	기말고사	← 측정시기
Kim	81	89	70	93	
Lee	78	90	87	88	
Park	94	96	82	84	
Jung	85	80	93	98	

```
stat <- rbind( c(81, 89), c(78, 90), c(94, 96), c(85, 80) )
stat
data <- rbind( c(70, 93), c(87, 88), c(82, 84), c(93, 98) )
```

```
data
result <- array( stat, data ), dim=c(4, 2, 2), dimnames = list(c("Kim", "Lee", "Park", "Jung"), c("중간고사", "기말고사"), c("통계학", "데이터분석개론"))) ; result
, , 통계학

          중간고사 기말고사
Kim         81       89
Lee         78       90
Park        94       96
Jung        85       80

, , 데이터분석개론

          중간고사 기말고사
Kim         70       93
Lee         87       88
Park        82       84
Jung        93       98
```

(3) 배열의 데이터 접근

배열의 데이터를 접근하는 방법은 행렬과 동일하며, **층(페이지)**의 인덱스만 추가하면 된다.

다음은 배열 데이터를 접근하는 예를 보여준다.

```
x <- array(c(-1,0,3,5,-3,4,1,0,7,-2,1,3), dim=c(2, 2, 3), dimnames = list(c("1행", "2행"), c("1열", "2열"), c("1층", "2층", "3층"))) ; x
x[2, 1, 3]      # 3층의 2행 1열의 원소
x[ ,  , 2]      # 2층의 모든 행과 열의 값, 즉 2층의 행렬을 출력
x[ ,  , c(1,2)]  # 1층과 2층의 행렬을 출력
x[ ,  , -3]
x[ 1,  , ]   # 배열에서 1행의 모든 것(열과 층)을 출력
x[  ,  , ]   # 이 결과는 무엇을 뜻할까요?
```

[실습] 입력된 배열에서

구분	통계학		데이터분석개론	
	중간고사	기말고사	중간고사	기말고사
Kim	81	89	70	93
Lee	78	90	87	88
Park	94	96	82	84
Jung	85	80	93	98

(1) Kim의 아래와 같은 행렬을 추출하는 R-코드를 작성하여라.

	통계학	데이터분석개론
중간고사	81	70
기말고사	89	93

(2) Kim의 전체(두 과목) 평균을 구하는 R-코드를 작성하여라.

(3) Kim의 과목별 평균을 구하는 R-코드를 작성하여라.

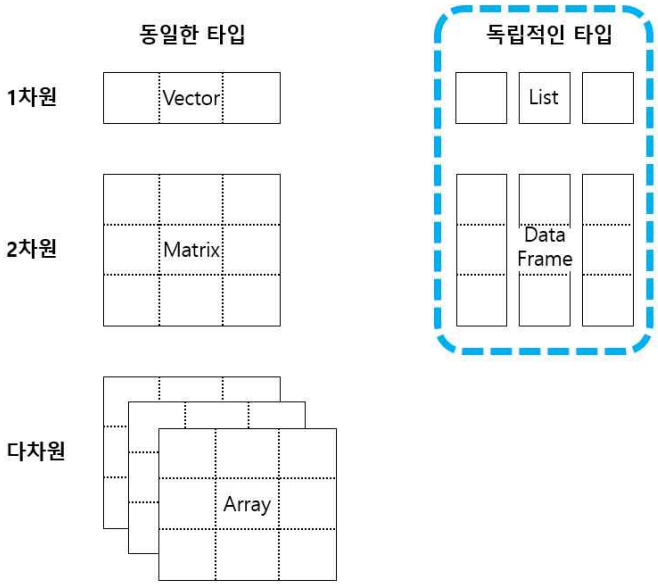
4. 리스트(list)

(1) list란?

아래에 예시에서는 **문자, 숫자**, 그리고 **문자형 vector**로 구성된 List를 생성하였다.

```
x <- list("Samsung", 95, c("서울","강남구"))
```

이처럼 **List**는 **다양한 데이터 유형**을 동시에 저장할 수 있다는 장점이 있다.



[그림] 자료구조[3)]

리스트(list)는 각 성분(원소)마다, **데이터프레임**(data.frame)은 각 column(열)마다 독립적인 데이터 유형을 코딩할 수 있다. 데이터 프레임에서 하나의 컬럼(열)은 벡터와 동일하다. 즉, 각 column마다 독립적인 유형의 데이터를 입력할 수 있지만 **각 column에서는 동일한 데이터 유형**이 되어야 한다.

예를 들어, 숫자가 들어가 있는 열에 문자 값 하나를 집어넣게 되면 해당 열은 전부 문자형으로 변환된다.

3) 이미지 출처: https://kookmindna.tistory.com/6?category=732960

[그림] list와 data.frame

리스트와 데이터 프레임의 가장 큰 차이점은 자유도에 있다.

① **data.frame**은 행렬과 동일하게 행과 열로 구성되어 있으며, 그 이상의 자유도를 갖지 못한다. 즉, 첫 번째 열은 3개의 요소를, 두 번째 열은 5개의 요소를 갖고 있을 수 없으며 한 개의 열이 3개의 요소를 갖고 있으면 나머지 열 또한 3개의 요소만을 가질 수 있다.

② 반면 list는 이와 다르다. 첫 번째 자리에 1개의 값을, 두 번째 자리에 3개의 값을 가질 수 있다. 즉, 각 자리마다 가질 수 있는 요소가 다양하며 R에서 가장 자유로운 형태를 지닐 수 있는 자료구조이다.

(2) list를 생성하는 방법

벡터가 c( )로 감싸주듯 list( ) 함수를 사용해 생성한다. 리스트는 **여러 유형의 변수**를 자유롭게 결합한 구조이다.

```
list(key1=value1, key2=value2, ... )
반환 값은 key1에 value1, key2에 value2 등을 저장한 리스트다.
```

리스트를 생성하는 방법은 다음과 같다.

```
x <- list("John", 500, c("I", "wife", "son")) ; x

[[1]]          # list의 각 구성요소(서브 리스트)의 명칭이 없을 때
[1] "John"
[[2]]           # 리스트 x의 두 번째 서브리스트
[1] 500
```

```
[[3]]          # 리스트 x의 세 번째 서브리스트
[1] "I"   "wife"   "son"
```

지금 보는 것과 같이 list는 각 서브리스트마다 길이가 다름을 알 수 있다. 1, 2번째 서브리스트는 1개, 3번째 서브리스트는 3개의 요소를 갖고 있다. 이런 특징이 벡터의 인덱싱과 유사하나 다르다고 한 이유이다.

이렇게 생성된 리스트의 값들은 **또 하나의 리스트(하위 리스트)**로 처리가 되며 각각의 서브리스트에 이름(명칭)을 부여할 수 있다.

```
y <- list(name="John", Salary=500, family=c("I", "wife", "son")) ; y
$name          # 리스트 x의 첫 번째 서브리스트의 이름(명칭), $name=[[1]]
[1] "John"
$Salary
[1] 500
$family
[1] "I"   "wife"   "son"
```

리스트에는 다양한 값을 혼합해서 저장할 수 있다. 즉, 리스트 안에 **리스트를 중첩**하는 것도 가능하다.

```
list(a=list(vec1=c(1, 2, 3)), b=list(vec2=c(1, 2, 3, 4)))
$a
$a$vec1
[1] 1 2 3
$b
$b$vec2
[1] 1 2 3 4
```

(3) list의 데이터 접근 방법

우리는 데이터가 어디에 위치해 있는지(인덱싱) 원하는 조건의 정보를 뽑아내는 것(필터링)은 매우 중요하다. 리스트의 데이터 접근법은 벡터와 유사한 것 같으나 다른 점이 있

다.

리스트의 n번째 서브리스트에 접근하는 방법과, n번째 서브리스트가 갖고 있는 값들에 접근하는 방법에 대해서 알아보도록 하자.

① 우선 n번째 서브리스트의 값에 접근하는 방법은 다음과 같다.

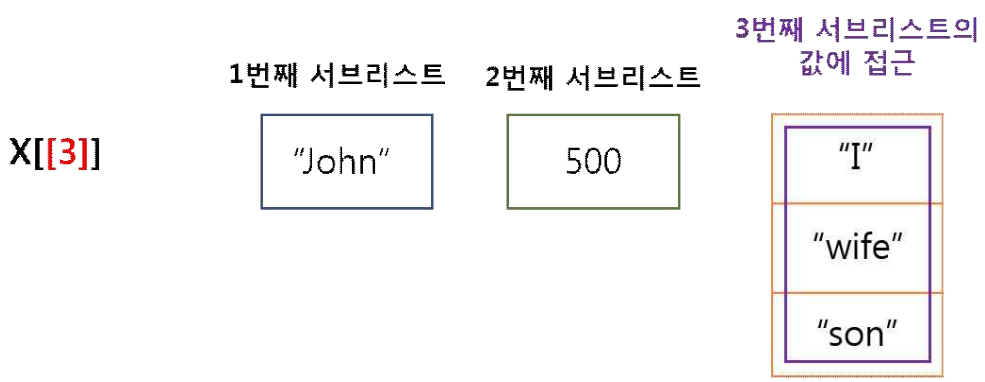
우선 **리스트 x**는 각각 "John", 500, c("I", "wife", "son") 라는 값을 가진 서브리스트들을 감싸고 있는 집합이다. x를 출력시켰을 때 나오는 것 중 **[[n]]**과 같이 대괄호 **[ ]**가 두개로 쓰여 있는 것은 리스트 안의 **n번째 서브리스트가 갖고 있는 값(데이터)**을 출력하라는 의미이다.

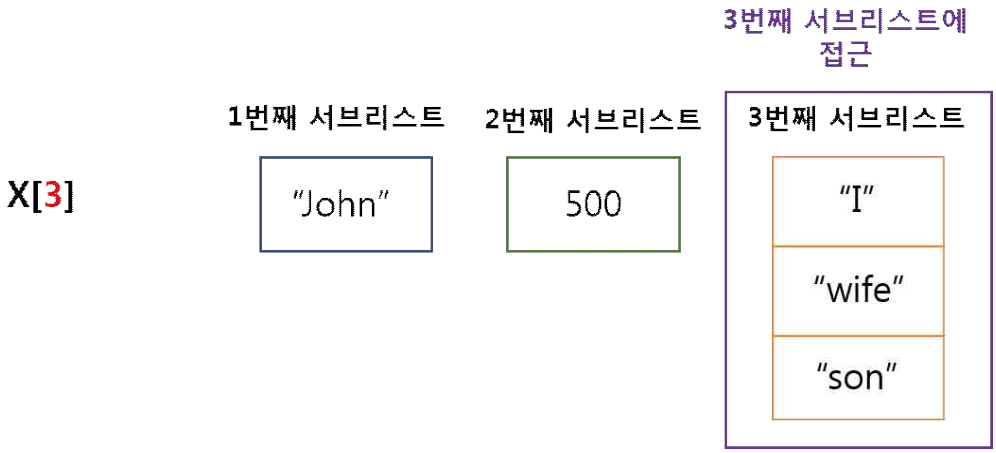
```
x <- list("John", 500, c("I", "wife", "son")) ; x
x[[3]]
[1] "I"   "wife"   "son"
# 3번째 서브리스트의 값에 접근, [[ ]] 사용
# list("John", 500, c("I", "wife", "son"))에서 c("I", "wife", "son")이 해당된다.
```

② n번째 서브리스트에 접근하는 방법은 다음과 같다.

```
x[3]        # 리스트 x의 3번째 서브리스트에 접근, 벡터의 3번째 성분에 접근과 동일
[[1]]
[1] "I"   "wife"   "son"
```

그림으로 표현하면 다음과 같다.





③ 서브리스트에 명칭이 있을 때 접근하는 방법: **변수$인덱스명**을 이용한다.

```
( x <- list(name="John", Salary=500, family=c("I", "wife", "son")) )

x$family     # 리스트 객체(x)에 속한 하위 객체(family) 지정, family에 저장된 값을 출력
[1] "I"   "wife"   "son"     # x$family 은 x[[3]] 와 동일한 결과
```

④ 객체의 유형을 구분하는 class( )를 사용하여 접근하는 방법

```
x <- list("John", 500, c("I", "wife", "son"))

class(x[[3]])
[1] "character"
# 3번째 서브리스트가 갖고 있는 값 "I", "wife", "son"을 의미하며 결과 값은 문자형이다.

class(x[3])
[1] "list"
# 모든 서브리스트에 접근을 하면 결과 값은 리스트이다.
```

⑤ 기타

```
x <- list(c(0, 1), 2:5, c("상", "중", "하"))

x[1]     # x의 첫 번째 서브리스트
x[[3]][3]   # x의 세 번째 서브리스트의 값들 중 3번째 원소
x[[2]][2:4]    # x의 두 번째 서브리스트의 값들 중 2부터 4번째까지 요소
x[2][[1]][4]   # x의 두 번째 서브리스트 -> 첫 번째 리스트의 값 -> 4번째 요소
```