2장 벡터

1. 벡터: 객체(자료의 저장소)중의 하나 ~ 변수와 비슷하다고 이해
2. 벡터생성: 자료type
3. 속성(attributes):
   객체의 정보를 나타내는 메타데이터(metadata)(다른 데이터를 설명해 주는 데이터)
4. 벡터의 구성요소 선택(서브셋팅):
5. 벡터연산: 구성요소(element-wise) 연산, 벡터연산
6. 벡터 생성 함수: c(), 콜론이용, seq(), rep()
7. NA와 NULL
   NA, NULL, NaN, Inf

1.벡터

벡터이름: 첫글자로 숫자를 사용, 글자사이에 "-" 사용, "." 뒤에 숫자 ~ NO

벡터에 데이터 할당(대입): "<-"와 "="는 대부분의 경우에 구분없이 사용, but "<-"가 우선순위

2. 벡터생성

1) 스칼라(scalar): 하나의 값, 객체, 원소가 하나인 벡터

2) 실수형 벡터:

mode(), typeof(): 객체의 자료 속성

str(): 객체의 자료 속성, 구조, 데이터 미리보기

	mode()	typeof()	str()
실수형	numeric	double	num, 구조, 미리보기
정수형	numeric	integer	int, 구조, 미리보기
문자형	character	character	chr, 구조, 미리보기
논리형	logical	logical	logi, 구조, 미리보기
요인형	numeric	integer	Factor, 구조, 미리보기
복소수형	complex	complex	cplx, 구조, 미리보기
원시형	raw	raw	raw, 구조, 미리보기

1) 실수형 벡터

```
> x1<-c(1,2,3); x1; typeof(x1)
```

2) 정수형 벡터

```
> x2<-c(1L,2L,3L); x2; typeof(x2)
```

3) 문자형 벡터: 문자열은 쌍따옴표 " " 혹은 홑따옴표 ' 사용

> x31<-c('a', 'b', 'c'); x3; typeof(x3)

> x32<-c('Tommy', 'Sally'); x3; typeof(x3)

4) 논리형 벡터: 구성요소가 TRUE, FALSE

> x4<-c(TRUE, FALSE); x4; typeof(x3)

5) 요인형(factor) 벡터:
범주형(categorical) 자료를 표현하는 벡터.
factor() 함수를 이용해서 요인형 벡터로 변환.

```
> x <- c("O","A","B","O","A")
> xf <- factor(x) # xf: 요인형 벡터
> xf
> typeof(x); tyoef(xf); levels(xf)
```

표 2.1 벡터생성과 정보에 관한 함수

```
> x<-c(1,3,5,7)
> names(x)<-c('x1','x2','x3','x4') # 각원소의 이름을 x1~x4로 하기,
> x
> names(x); length(x); is,vector(x) #names(x): x의 각원소의 이름,
```

append() 함수: append(x,y,after=m)

```
> z0<-append(x, 10,  after=0) ; z0
> z1<-append(x, 10,  after=1) ; z1
> z2<-append(x, 10,  after=2) ; z2
> z3<-append(x, 10,  after=3) ; z3
> z4<-append(x, 10,  after=4) ; z4
```

3. 속성(attributes):

객체의 정보를 나타내는 메타데이터(metadata)(다른 데이터를 설명해 주는 데이터),
attributes(), class(), dim(), dimnames(), names(), row.names(), colnames()함수...
클래스(class), 차원(dim), 차원이름(dimnames), 이름(names),
행이름(rowames), 열이름(colnames)

```
> x <-c(1,2,3)
> names(x); class(x)
> attributes(x)
> names(x) <-  c('x1','x2','x3')
> names(x); class(x)
> attributes(x)
```

4. 벡터의 구성요소 선택(서브셋팅):

1) 벡터 서브셋팅: 벡터에서 구성요소를 선택, 즉 벡터의 일부분을 추출

```
> x<-c(5,7,1,3,9,2)
> x[2]; x[-5]
> x[c(2,5)]; x[-c(2,5)]; x[c(-2,-5)]
> x[c(2,-5)]
> x[x>2]; x[x<=3];  x[x>1 & x<6]
> x[c(T,F,F,F,F,T)] # TRUE에 대응하는 원소만 추출
> x[c(T,F)] # (T,F,T,F,T,F)중 TRUE에 대응하는 원소만 추출
> x[c(F,T)] # (F,T,F,T,F,T)중 TRUE에 대응하는 원소만 추출
> names(x) <- c('x1','x2','x3','x4','x5','x6')
> x[c('x1', 'x4')]
```

5. 벡터연산

1) 구성 요소단위 연산

```
> x<-c(1,2); y<-c(3,4)
> x+y; x*y; x/y
```

2) 벡터 재활용

```
# 두 벡터 길이가 약수/배수 관계
> x<-c(1,2,3,4); y<-c(10,20)
> x+y; x*y; x/y
# 두 벡터길이가 약수/배수 관계가 되지 않으므로 경고 메시지
> x<-c(1,2,3); y<-c(10,20)
> x+y; x*y; x/y
```

3) 벡터화 연산

```
> x^2; sqrt(x)
```

4) 비교 연산

표2.3 비교연산자

```
> x<-c(5,7,1,3,9,2)
> x>2; x!=1
> sum(x[x>2]) # 2보다 큰 원소의 합
> y<-c(1,2,3)
> x %in% y # x의 원소가 y에 속하는가? T, F
```

표2.4 논리연산자 : 두 개 이상의 비교연산의 결과를 합치는 논리연산

```
> x<-c(1,2,3);
> x==c(1,3,3); x==c(1,2,2)
```

> x==c(1,3,3) & x==c(1,2,2); x==c(1,3,3) && x==c(1,2,2)

all(), any() 함수 ~ 모두 TRUE, 하나라도 TRUE이면 TRUE

> all(x>1); any(x>1)

6. 벡터 생성 함수

표 2.5 벡터생성함수

> x1<-c(1,2,3); x2<-c(4,5,6); x12<-c(x1,x2)
> x1; x2; x12
> x3<-1:5; x3
> x4<-seq(1,7,2); x4
> x5<-rep(x, 2); x5

7. NA와 NULL

NA: not available, 값이 존재하지 않음(값을 모름)

NULL: empty, 값이 없는 상태

NaN: not a number, 수학적으로 정의가 되지 않는 값

Inf: infinity, 무한대

> x1<-c(10,20,NA)

> x1; length(x1); sum(x1); sum(x1, na.rm=TRUE)

> x2<-c(30,40,NULL)

> x2;length(x2); sum(x2)
> x3<- 1/0; x4<- -1/0
> x3; x4
> x5<-0/0
> x5

3장 행렬과 배열

* 행렬은 행과 열로 구성(2차원, 행의 수*열의 수),

1	3	5
2	4	6

: 2*3 행렬(읽을 때는 2 by 3 행렬)

* 배열은 같은 크기의 행렬을 여러개 포갠 형태(3차원, 행의 수*열의 수*행렬의 수)

1. 행렬(matrix) 생성
2. 행렬 서브셋팅
3. 행렬 연산
4. 배열(array)

1. 행렬(matrix) 생성

행과 열로 구성되는 2차원 형태-열우선 배열방식(열부터 채우기)

(1) 벡터에 차원 속성(dimension) 부여.
(2) matrix() 함수 이용.
(3) rbind() 함수 또는 cbind() 함수 이용.

(1) 벡터에 차원 속성(dimension) 부여.

```
> x<-c(1,2,3,4,5,6)
> x; class(x); nrow(x); ncol(x) # class(): 객체의 구조, 벡터인 경우 자료형태
> dim(x)=c(2,3) # dim(x): x를 2*3 행렬로 변환
> x; class(x); nrow(x); ncol(x); dim(x) # dim(x): x의 행과 열의 수(dimension)
```

(2) matrix() 함수 이용.

matrix(data-벡터, nrow=행의 수, ncol=열의 수, byrow=T)
byrow=T ~ 행우선 배열(행부터 채우기), 없으면 열우선 배열
nrow=해의 개수, ncol=열의 개수 둘 중하나만 입력 가능
벡터길이=nrow*ncol

```
> x<-c(1,2,3,4,5,6)
> x1<-matrix(x, nrow=2, ncol=3, byrow=T) # 행우선 입력
> x11<-matrix(x, nrow=2, byrow=T)
> x12<-matrix(x, ncol=3, byrow=T)
> x1; x11; x12
> x2<-matrix(x, nrow=2, ncol=3) # 열우선 입력
> x21<-matrix(x, nrow=2)
> x22<-matrix(x, ncol=3)
> x2; x21; x22
```

(3) rbind() 함수 또는 cbind() 함수 이용.

rbind(x,y,...) : 벡터 x, y,.. 를 행으로 결합하여 행렬을 만들기, 행의 수=벡터의 수
cbind(x,y,...) : 벡터 x, y,.. 를 열로 결합하여 행렬을 만들기, 열의 수=벡터의 수

```
> x<-1:3 ; y <-4:6; x; y
> zR<-rbind(x,y) # 2*3 matrix, 행이름이 x, y로.
> zR; class(zR)
> zC<-cbind(x,y) # 3*2 matrix, 열이름이 x, y로.
> zC; class(zC)
```

2. 행렬 서브셋팅

"[ ]" 와 subset() 함수를 사용.
(1) "[ ]" 사용.
x[a, b] # a번째 행과 b번째 열 선택

```
> a<-matrix( 9:20, nrow=3) #  3*4 행렬, 열우선 입력
> a
> a[c(1,4)] # 1번째, 4번째 원소. 순서는 열 우선.
> a[1, ]; a[-2, ] # 1번째 행만. 2번째 행 빼고.
> a[c(1,3),]; a[-c(1,3),]; a[c(-1,-3),] # 1,3번째 행. 1,3번째 행 빼고
> a[, 2]; a[, -3] # 2번째 열만. 3번째 열 빼고.
> a[,c(2,4)]; a[,-c(2,4),]; a[,c(-2,-4)] # 2,3번째 열. 2,4번째 열 빼고
> a[c(1,2),c(2,4)] # 1,2번째 행과 2,4번째 열
>a[a[,1]>9, ] # 1번째 열의 원소중 9보다 큰 원소포함 행으로 만들어지는 행렬
>a[, a[1,]>9] # 1번째 행의 원소중 9보다 큰 원소포함 열로 만들어지는 행렬
```

(2) subset() 함수를 사용.
subset(x, subset = a, select = b) : a는 행 선택에 필요한 조건(비교식 혹은 논리식), b는 열 번호 혹은 열 이름. select = b 없으면 모든 열에 대해서.
*행의 수<열의 수 인 경우, 열을 이용한 조건(a[,k]) 사용해야함. a[k,] 사용하면 error

```
> subset(a,select=2)
> subset(a, a[,1]>9) # 1번째열의 원소중 9보다 큰 원소포함 행으로 만들어지는 행렬
```

3. 행렬 연산

1) 행렬 연산
2) 행렬 함수: t(), solve(), diag()

1) 행렬 연산:

+, -, *, /, ^ , sqrt(): 구성요소(원소)끼리 연산

```
> x<-1:2
> x+1; x*2; x/3; x^2; sqrt(x) # x의 각 원소에 +1, *2, /3, 2제곱, 제곱근
> a<-matrix(1:6, ncol=2)
> a+1; a*2; a/3; a^2; sqrt(a) # a의 각 원소에 +1, *2, /3, 2제곱 제곱근
> sum(a); mean(a);
> rowSums(a); colSums(a)
> rowMeans(a); colMeans(a)
```

%*%: 행렬끼리 곱

차원 맞아야 가능 ~ A%*%B 에서 dim(A)=(n,m), dim(B)=(m, ℓ), dim(A%*%B)=(n, ℓ)

```
> dim(a) # (3,2)
> b<-matrix(1:6, ncol=3)
> b
> a%*%b
> c<-matrix(1:6, ncol=2, byrow=T)
> a%*%c # error
```

2) 행렬 함수:

t():전치행렬, solve():역행렬, diag():단위행렬

```
> x <- matrix(1:4, ncol=2)
> x
> xt=t(x); xt
> xi=solve(x); xi
> x%*%xi
> diag(3)
```

eg: 2원1차 연립방정식: 사과 2개와 배 3개의 가격은 1.9만원, 사과 6개와 배 2개의 가격은 2.2만원. 사과1개 가격(x_1), 배1개 가격(x_2)?

$2x_1+3x_2=1.9$

$6x_1+2x_2=2.2$

$$\begin{pmatrix}2 & 3\\ 6 & 2\end{pmatrix}\begin{pmatrix}x_1\\ x_2\end{pmatrix}=\begin{pmatrix}1.9\\ 2.2\end{pmatrix} \Rightarrow \begin{pmatrix}x_1\\ x_2\end{pmatrix}=\begin{pmatrix}2 & 3\\ 6 & 2\end{pmatrix}^{-1}\begin{pmatrix}1.9\\ 2.2\end{pmatrix}$$

```
> A <-matrix(c(2,3,6,2), ncol=2, byrow=T)
> b <-c(1.9, 2.2)
> x<-solve(A)%*%b
```

4. 배열(array)

1) 배열 생성: array() 함수 이용

2명의 학생이 3과목(math, history, science)을 2번(mid, final) 응시한 성적을 저장.

2개의 행렬: 중간, 기말 점수

행: 학생번호별 점수, 열: 과목별 점수

(다르게 표현도 가능 ~ 5개 행렬(5 학생), 행: 중간, 기말, 열: 과목별 점수)

행렬1:

80	90	85
65	70	55

행렬2:

65	80	75
85	90	80

```
> mid<-matrix(c(80,90,85,65,70,55), ncol=3,byrow=T)
> final<-matrix(c(65,80,75,85,90,80), ncol=3, byrow=T)
> test0 <-array(c(mid,final), dim=c(2,3,2)) # 2: 행렬 수
> test0
> test1=array(c(mid, final), dim=c(2,3,2), dimnames=list(c('st1','st2'),
c('math','history','scirnce'),c('mid','final')))
# list와 vector의 차이점은 vector는 한가지 타입의 원소만 담을 수 있지만, list는 여러가지
타입의 원소를 담을 수 있음.
> test1
```

2) 배열 서브셋팅

```
#1번 학생의 두번째 과목(history)의 중간고사 점수:
> test[1,2,1]
#1번 학생의 세과목의 중간고사 점수:
> test[1, ,1]
#1번 학생의 세과목의 중간, 기말고사 점수:
> test[1, , ]
```