2장
데이터 타입

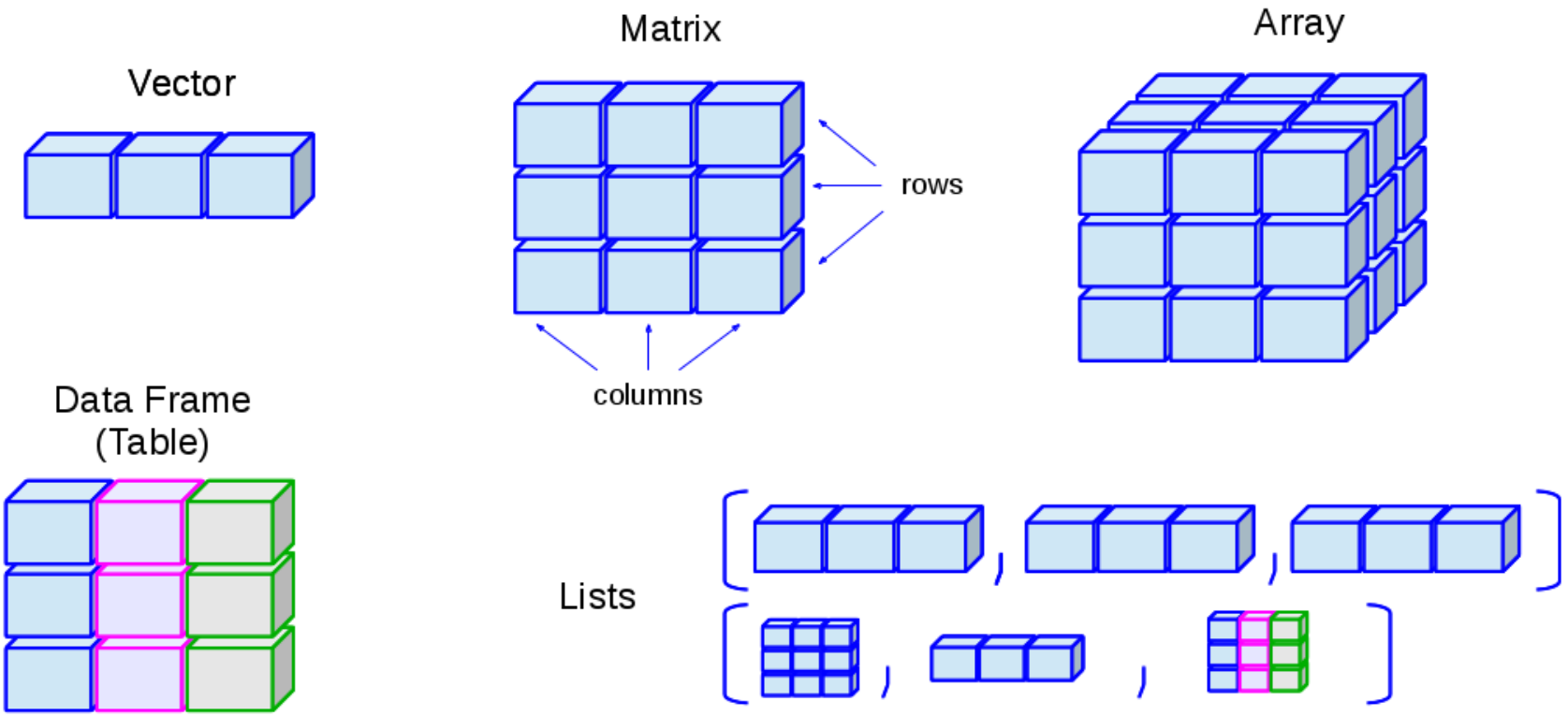
기초 데이터 타입

- integer: 정수
- numeric: 실수
- character: 문자열
- logical: 논리형(부울형)
- complex: 복소수

복합 데이터 구조 타입

- 벡터(vector)
- 행렬(matrix)
- 데이터 프레임(data frame)
- 리스트(list)

복합 데이터 구조 타입



변수

- 사용가능 예

```
a
b
a1
a2
.x
```

- 불가능 예

```
2a
.2
```

- 또 다른 예

  data.training

  data.validation

- 변수값 할당

  <- <<- =

  대부분 <-를 사용

함수 호출시 인자지정

- 예)
  - 정의: foo(a, b, c=1, d=1)

    - c와 d는 호출시 생략 가능하고 default값이 지정됨
    - 예)
      - foo(1, 2)
      - foo(1, 2, 1)
      - foo(1, 2, 3, 4)

    - 순서 무시 가능
    - 예)
      - foo(d=5, a=3, b=4)

스칼라

- 숫자

```
> a <- 3
> b <- 4.5
> c <- a + b
> print(c)
[1] 7.5
```

- NA
  - 값이 존재하지 않음

```
> one <- 100
> two <- 75
> three <- 80
> four <- NA
> is.na(four)
[1] TRUE
```

스칼라

- NULL
  - 변수가 초기화 되지 않은 경우
  - 결측치(NA)와 구분

```
> x <- NULL
> is.null(x)
[1] TRUE
> is.null(1)
[1] FALSE
> is.null(NA)
[1] FALSE
> is.na(NULL)
logical(0)
Warning message:
In is.na(NULL) : is.na() applied to non-(list or vector) of type 'NULL
   '
```

스칼라

- 문자열
  - ‘this is string’ 또는 “this is string”

```
> a <- "hello"
> print(a)
[1] "hello"
```

- 진리값
  - TRUE, T
  - FALSE, F

& (AND), | (OR), ! (NOT) 연산자 사용가능

```
> TRUE & TRUE
[1] TRUE
> TRUE & FALSE
[1] FALSE
```

```
> TRUE | TRUE
[1] TRUE
> TRUE | FALSE
[1] TRUE
> !TRUE
[1] FALSE
> !FALSE
[1] TRUE
```

스칼라

- 진리값
  - AND나 OR연산자에는 &, | 외에도 &&와 || 가 있다.
    - &, |는 boolean이 저장된 벡터(Vector) 끼리의 연산시 각 원소간 계산을 할 때 사용
    - &&와 ||는 for나 if문 등에서 개별 값의 논리값을 계산할 때 사용

스칼라

- 팩터(factor) – 범주형 데이터
  - factor(data, levels)

```
> sex <- factor("m", c("m", "f"))
> sex
[1] m
Levels: m f
```

  - factor(data)
    - 자동적으로 level을 생성

```
> sex <- factor(c('m', 'f', 'm'))
> sex
[1] m f m
Levels: f m
```

```
> nlevels(sex)
[1] 2


> levels(sex)
[1] "m" "f"
```

스칼라

- factor level 수정

```
> sex
[1] m
Levels: m f
> levels(sex) <- c('male', 'female')
> sex
[1] male
Levels: male female
```

- 순서가 있는 factor
  - eg) 나쁨 < 조금 나쁨< 보통< 조금 좋음< 아주 좋음

```
> ordered(c("a", "b", "c"))
[1] a b c
Levels: a < b < c
> factor(c("a", "b", "c"), ordered=TRUE)
[1] a b c
Levels: a < b < c
```

벡터

- 배열과 유사
- 한가지 타입의 데이터가 순서대로 저장

```
> x <- c(1, 2, 3, 4, 5)
> x
[1] 1 2 3 4 5
```

```
> x <- c("1", 2, "3")
> x
[1] "1" "2" "3"
```

```
> c(1, 2, 3)
[1] 1 2 3
> c(1, 2, 3, c(1, 2, 3))
[1] 1 2 3 1 2 3
```

```
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> x <- 5:10
> x
[1]  5  6  7  8  9 10
> seq(1, 10, 2)
[1] 1 3 5 7 9
```

벡터

- 각 원소에 이름 부여 가능

```
> x <- c(1, 3, 4)
> names(x) <- c("kim", "seo", "park")
> x
 kim  seo park
   1    3    4
```

벡터 내 데이터 접근

```
> x <- c("a", "b", "c")
> x[1]
[1] "a"
> x[3]
[1] "c"
```

```
> x <- c("a", "b", "c")
> x[-1]
[1] "b" "c"
> x[-2]
[1] "a" "c"
```

특정요소 제외

```
> x <- c("a", "b", "c")
> x[c(1, 2)]
[1] "a" "b"
> x[c(1, 3)]
[1] "a" "c"
```

```
> x <- c("a", "b", "c")
> x[1:2]
[1] "a" "b"
> x[1:3]
[1] "a" "b" "c"
```

벡터 내 데이터 접근

```
> x <- c(1, 3, 4)
> names(x) <- c("kim", "seo", "park")
> x
 kim  seo park
   1    3    4
> x["seo"]
seo
  3
> x[c("seo", "park")]
 seo park
   3    4
```

```
> x <- c("a", "b", "c")
> length(x)
[1] 3
> nrow(x)  # nrow()는 행렬만가능
NULL
> NROW(x)  # NROW()는 벡터와행렬모두사용가능
[1] 3
```

```
> names(x)[2]
[1] "seo"
```

벡터 연산

```
> "a" %in% c("a", "b", "c")
[1] TRUE
> "d" %in% c("a", "b", "c")
[1] FALSE
```

```
> identical(c(1,2,3), c(1,2,3))
[1] TRUE
> identical(c(1,2,3), c(1,2,100))
[1] FALSE
```

```
> x <- c(1,2,3,4,5)
> x+1
[1] 2 3 4 5 6
> 10-x
[1] 9 8 7 6 5
```

```
> c(1,2,3) == c(1,2,100)
[1]  TRUE  TRUE FALSE
```

```
> setdiff(c("a", "b", "c"), c("a", "d"))
[1] "b" "c"
> union(c("a", "b", "c"), c("a", "d"))
[1] "a" "b" "c" "d"
> intersect(c("a", "b", "c"), c("a", "d"))
[1] "a"
```

연속된 숫자의 벡터

```
> seq(1, 5)
[1] 1 2 3 4 5
> seq(1, 5, 2)
[1] 1 3 5
```

```
> 1:5
[1] 1 2 3 4 5
```

리스트

- (키, 값) 형태의 데이터
- 서로 다른 데이터 타입저장 가능

```
> x <- list(name="foo", height=70)
> x
$name
[1] "foo"

$height
[1] 70
```

```
> x <- list(name="foo", height=c(1, 3, 5))
> x
$name
[1] "foo"

$height
[1] 1 3 5
```

리스트

- 리스트 내의 리스트

```
> list(a=list(val=c(1, 2, 3)), b=list(val=c(1, 2, 3, 4)))

$a
$a$val
[1] 1 2 3


$b
$b$val
[1] 1 2 3 4
```

리스트내의 데이터 접근

- 방법 : x$key x[n] x[[n]]

```
> x <- list(name="foo", height=c(1, 3, 5))
> x$name
[1] "foo"
> x$height
[1] 1 3 5
> x[[1]]
[1] "foo"
> x[[2]]
[1] 1 3 5
```

```
> x[1]
$name
[1] "foo"

> x[2]
$height
[1] 1 3 5
```

결과는 리스트, 즉, 1개의 원소를 갖는 서브리스트 반환

행렬(matrix)

- 2차원 벡터
  - 원소의 타입이 동일 데이터 형

```
> matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow=3)
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), ncol=3)
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

행렬(matrix)

```
> matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow=3, byrow=T)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

행과 열에 이름 부여

```
> matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow=3,
+        dimnames=list(c("item1", "item2", "item3"),
+                      c("feature1", "feature2", "feature3")))
      feature1 feature2 feature3
item1        1        4        7
item2        2        5        8
item3        3        6        9
```

행렬(matrix)

- 행과 열 이름을 별도로 부여

```
> x <- matrix (c(1, 2, 3, 4, 5, 6, 7, 8, 9) , nrow =3)
> x
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> rownames(x) <- c('r1', 'r2', 'r3')
> colnames(x) <- c('c1', 'c2', 'c3')
> x
   c1 c2 c3
r1  1  4  7
r2  2  5  8
r3  3  6  9
```

행렬 데이터 접근

- 행과 열 번호로 접근
  - 1번부터 시작함

```
> x <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), ncol=3)
> x
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> x[1,1]
[1] 1
> x[1,2]
[1] 4
> x[2,1]
[1] 2
> x[2,2]
[1] 5
```

행렬 데이터 접근

```
> x[1:2, ]
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
```

```
> x[-3, ]
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
```

```
> x[c(1, 3), c(1, 3)]
     [,1] [,2]
[1,]    1    7
[2,]    3    9
```

행렬 데이터 접근

- 행과 열 이름으로 접근

```
> x <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9),
+             nrow=3,
+             dimnames=list(c("item1", "item2", "item3"),
+                           c("feature1", "feature2", "feature3")))
> x
      feature1 feature2 feature3
item1        1        4        7
item2        2        5        8
item3        3        6        9
> x["item1", ]
feature1 feature2 feature3
       1        4        7
```

x[x[, "features"] >= 5 ,] ← 두 번째 컬럼의 값이 5보다 큰 행을 출력
x[x[, 2] >= 5 ,]

행렬연산

```
> x <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow=3)
> x * 2
     [,1] [,2] [,3]
[1,]    2    8   14
[2,]    4   10   16
[3,]    6   12   18
> x / 2
     [,1] [,2] [,3]
[1,]  0.5  2.0  3.5
[2,]  1.0  2.5  4.0
[3,]  1.5  3.0  4.5
```

행렬연산

```
> x <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow=3)
> x + x
     [,1] [,2] [,3]
[1,]    2    8   14
[2,]    4   10   16
[3,]    6   12   18
> x - x
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
[3,]    0    0    0
```

전치행렬

```
> x <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow=3)
> x
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> t(x)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

행렬연산

- 행과 열의 개수 구하기

```
> x <- matrix(c(1, 2, 3, 4, 5, 6), ncol=3)
> x
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> ncol(x)
[1] 3
> nrow(x)
[1] 2
```

배열

- 2차원 이상의 행렬

Data Frame

- R에서 가장 중요한 자료형
- Database의 테이블과 같은 형태를 가짐

```
> d <- data.frame(x=c(1, 2, 3, 4, 5), y=c(2, 4, 6, 8, 10))
> d
  x  y
1 1  2
2 2  4
3 3  6
4 4  8
5 5 10
```

다른 방법

```
a <- c(...)
b <- c(...)
d <- data.frame(a, b)
```

Data Frame

- Factor 형태의 z 열이 추가된 예이다

```
> d <- data.frame (x=c(1, 2, 3, 4, 5) ,
+                  y=c(2, 4, 6, 8, 10) ,
+                  z=c('M', 'F', 'M', 'F', 'M'))
> d
  x  y z
1 1  2 M
2 2  4 F
3 3  6 M
4 4  8 F
5 5 10 M
> str(d)
'data.frame': 5 obs. of  3 variables:
 $ x: num  1 2 3 4 5
 $ y: num  2 4 6 8 10
 $ z: Factor w/ 2 levels "F","M": 2 1 2 1 2
```

Data Frame

- z는 문자열 형태

```
> d <- data.frame (x=c(1, 2, 3, 4, 5) ,
+                  y=c(2, 4, 6, 8, 10) ,
+                  z=c('M', 'F', 'M', 'F', 'M'), stringsAsFactors=F)
> d
  x  y z
1 1  2 M
2 2  4 F
3 3  6 M
4 4  8 F
5 5 10 M
> str(d)
'data.frame': 5 obs. of  3 variables:
 $ x: num  1 2 3 4 5
 $ y: num  2 4 6 8 10
 $ z: chr  "M" "F" "M" "F" ...
```

Data Frame

- 컬럼 추가

```
> d <- data.frame(x=c(1, 2, 3, 4, 5),
+                 y=c(2, 4, 6, 8, 10),
+                 z=c('M', 'F', 'M', 'F', 'M'))
> d$v <- c(3, 6, 9, 12, 15)
> d
  x  y z  v
1 1  2 M  3
2 2  4 F  6
3 3  6 M  9
4 4  8 F 12
5 5 10 M 15
```

데이터 프레임 접근

```
> d <- data.frame(x=c(1, 2, 3, 4, 5), y=c(2, 4, 6, 8, 10))
> d$x
[1] 1 2 3 4 5
> d[1,]
  x y
1 1 2
> d[1,2]
[1] 2
```

컬럼명 지정

```
> d[, c("x", "y")]
  x  y
1 1  2
2 2  4
3 3  6
4 4  8
5 5 10

> d[, c("x")]
[1] 1 2 3 4 5
```

d[d$x >3,] ← x column이 3보다 큰 행을 출력

데이터 프레임 접근

- 한 컬럼만 선택시 벡터처럼 출력됨.
- 이를 피하려면

```
> d[, c("x"), drop=FALSE]
  x
1 1
2 2
3 3
4 4
5 5
```

데이터 프레임 접근

- 데이터 프레임의 행 이름, 열 이름은 각각 rownames(), colnames() 함수로 지정
- 행과 열이름 부여

  colnames(x) <- c('val')

  rownames(x) <- c('a', 'b', 'c')

```
> x <- data.frame(1:3)
> x
  X1.3
1    1
2    2
3    3
> colnames(x) <- c('val')
> x
  val
1 1
2 2
3 3
> rownames(x) <- c('a', 'b', 'c')
> x
  val
a 1
b 2
c 3
```

데이터 브라우징

- 엑셀형태로의 출력
- view(x)

타입판별

```
> class(c(1, 2))
[1] "numeric"
> class(matrix(c(1, 2)))
[1] "matrix"
> class(list(c(1,2)))
[1] "list"
```

```
> class(data.frame(x=c(1,2)))
[1] "data.frame"
```

```
> is.numeric(c(1, 2, 3))
[1] TRUE
> is.numeric(c('a', 'b', 'c'))
[1] FALSE
> is.matrix(matrix(c(1, 2)))
[1] TRUE
```

is.factor()
is.character()
is.data.frame() 등

타입변환

- 행렬이나 리스트를 데이터 프레임으로 변환 예

```
> x <- data.frame(matrix(c(1, 2, 3, 4), ncol=2))
> x
  X1 X2
1  1  3
2  2  4
> colnames(x) <- c("X", "Y")
> x
  X Y
1 1 3
2 2 4
```

```
> data.frame(list(x=c(1, 2), y=c(3, 4)))
  x y
1 1 3
2 2 4
```

타입변환(as.xxx)

```
> x <- c("m", "f")
> as.factor(x)
[1] m f
Levels: f m
> as.numeric(as.factor(x))
[1] 2 1
```

알파벳 순서로 지정됨

기타
- as.character()
- as.matrix()
- as.data.frame()

```
> factor(c("m", "f"), levels=c("m", "f"))
[1] m f
Levels: m f
```

순서를 m, f로 강제 지정