제3장 데이터 구조의 이해와 코딩의 시작

제3장 데이터 구조의 이해와 코딩의 시작

I. 벡터

1. 벡터 만들기
2. 벡터 원소가 하나일 때
3. 산술연산
    3-1. 벡터의 사칙연산
    3-2. 서로 다른 데이터 유형과 연산
4. 비교 연산자
5. 논리 연산자
6. 연속적인 값들의 벡터 만들기
7. 반복적인 값들의 벡터 만들기
8. 벡터 요소의 확인
    8-1) 벡터의 색인에 의한 확인
    8-2) 요소 값의 조건 비교에 따른 확인
9. 원소 값의 수정
    9-1) 벡터의 색인을 참조한 수정
    9-2) 벡터 요소 값 비교에 의한 수정

[연습문제]

10. 함수의 사용
    10-1. 문자열의 결합 : `paste()`
    10-2. 결측치 : `NULL` , `NA`
    10-3. `Inf` , `NaN`

II. 배열

1. 1차원 배열
2. 2차원 배열 만들기
    2-1. `array()` 함수 사용
    2-2. 벡터 결합으로 배열 만들기
3. 2차원 배열의 요소 값 보기와 수정
4. 2차원 배열의 행과 열 이름 추가
    4-1. (방법 1) : `colnames()` , `rownames()` 함수 이용
    4-2. (방법 2) : `dimnames =` 인수 이용
5. 배열 요소 값 확인
    5-1. 배열의 색인에 의한 확인
    5-2. 배열 요소의 값 비교에 의한 확인
    5-3. 행/열의 이름을 이용한 요소의 확인
6. 배열 요소 값의 수정
    6-1. 배열의 색인을 이용한 수정
    6-2. 배열 요소의 비교에 의한 확인
5. 3차원 배열
6. 함수 적용

III. 행렬

1. 행렬 만들기 : matrix() 함수이용.
2. 행과 열 이름 주기
3. 벡터 결합에 의한 행렬 만들기
4. 행과 열 이름 달기
5. 대각행렬 만들기

6. 행렬의 연산
6-1. 행렬의 곱
6-2. 전치행렬(transpose matrix) : `t()` 함수 사용
6-3. 역행렬(martix inversion) : solve()
6-4. 행렬식(determinant)

IV. 리스트
1. 리스트 만들기 : list( ) 함수 이용
1-1. 리스트의 원소 출력
1-2. []와 [[ ]]의 차이
1-3. 리스트에 원소 추가
1-4. 리스트에서 구성요소 제거
1-5. 리스트 합치기
1-6. 리스트 요소의 갯수 확인 : length()
2. 요소별로 키워드 지정하기
2-1. 리스트의 원소 출력
2-2. 리스트에 원소 추가
2-3. 리스트에서 구성요소 제거
2-4. 리스트 합치기 : `c()` 이용
2-5. 리스트 요소의 갯수 확인 : length()

V. 데이터 프레임
1. 데이터 프레임 생성 <방법 1>
1-1. 옵션 : stringsAsFactors= FALSE
2. 데이터 프레임 생성 <방법 2>
3. 데이터 프레임에 열과 행 단위 추가
3-1. 열 추가
3-2. 행 추가
4. 데이터 프레임의 요소 값 확인
4-1. 색인으로 확인하기
4-2. 열의 이름으로 확인하기
4-3. 요소 값의 비교에 의한 확인
4. 요소 값 수정
5. 데이터 프레임의 factor 데이터 수정

VI. 데이터 세트
1. 데이터 세트 목록 보기
1-1. 데이터 세트 목록 보기
1-2. 데이터 세트의 이용
1-3. 데이터 세트의 구조 보기
1-4. 데이터 세트의 요약 정보
1-5. 데이터 세트의 저장
1-6. 저장한 파일을 data 변수로 불러오기
2. 데이터 세트 구조 보기
3. 데이터 세트 요약 보기
4. 데이터 세트 저장하고 읽기

VII. 웹 사이트의 데이터 불러오기
1. 웹사이트의 데이터 파일 읽기

VIII. 사용자 정의 함수
1. 함수 만들기
1-1. 원의 면적을 구하는 함수의 정의와 호출

I. 벡터

1. 벡터 만들기

```
x <- c(80, 85, 70)          # 처리할 데이터의 변수 정의
x                           # 변수 내용의 확인

c(80, 85, 70) -> x          # 바람직하지 않음.
x

x <- c(1,2,3,4) ; x

y <- c(2,3,4,5) ; y

(z <- c(1,2))
```

2. 벡터 원소가 하나일 때

```
x <- c(80)
x

x <- 80
x
```

3. 산술연산

```
x <- 5+2
x

x <- 5/3
x

x <- 5^2
x

x <- 5%%3
x

x <- 5%/%3
x
```

3-1. 벡터의 사칙연산

```
x <- c(1, 2, 3, 4) # 4개 원소
y <- c(2, 3, 4, 5) # 4개 원소
z <- c(1, 2)       # 2개 원소

w <- x+y
w

w <- x+5
w

w <- y/2
```

```
w

w <- x+z
w

w <- x/z
w

w <- z/x
w

w <- y %/% x

w <- y %% x
```

3-2. 서로 다른 데이터 유형과 연산

```
x <- c(1, 2, 3)                 # 숫자 벡터
x

y <- c("A", "B", "c")           # 문자 벡터 : ""로 묶는다.
y

y <- c("A", 1, 2)               # 문자와 숫자의 혼합 -> 모두 문자로 처리함.
y

z <- y + 1            # Error 발생
```

4. 비교 연산자

(>=, >, ==, <, <=) => 결과는 'TRUE' 또는 'FALSE'

```
x <- 5 < 3
x

y <- c(10, 20, 30)
z <- y <= 10
z
```

5. 논리 연산자

```
x <- TRUE
y <- FALSE
x | y

x & y

# p.67
x <- 3
!x

isTRUE(y)

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

```
z | y
```

6. 연속적인 값들의 벡터 만들기

```
x <- seq(1, 10)
x

x <- 1:10
x

x <- seq(10, 1)
x

y <- 10:1
y

x <- seq(1, 10, by=3)
x

y <- seq(1, 10, length.out=5)
y
```

7. 반복적인 값들의 벡터 만들기

```
x <- c(1, 2, 3)
rep(x, times=2)

rep(x, each=2)

x1 <- rep(1:4, 2)

x2 <- rep(1:4, each = 2)

x3 <- rep(1:4, c(2,2,2,2))

x4 <- rep(1:4, c(2,1,2,1))
```

8. 벡터 요소의 확인

8-1) 벡터의 색인에 의한 확인

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

x[c(1, 3, 5)]

x[-c(2, 4)]
```

8-2) 요소 값의 조건 비교에 따른 확인

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

x[x >=2 & x <=4]
```

9. 원소 값의 수정

9-1) 벡터의 색인을 참조한 수정

```r
x <- c(1, 2, 3, 4, 5)
x[2] <- 20
x

x[c(3, 4)] <- 15
x

x[x <= 15] <- 10
x
```

9-2) 벡터 요소 값 비교에 의한 수정

```r
x <- c(1, 2, 3, 4, 5)

x[x<=15] <- 10
```

[연습문제]

`Ramen` 변수에 (Cup, Bowl, Cup, Bowl)의 데이터를 입력하고, Cup을 Bowl로 Bowl은 컵으로 데이터를 변경하라.

```r
Ramen <- c("Cup", "Bowl", "Cup", "Bowl")

Ramen[Ramen == "Cup"] <- "x"
Ramen
Ramen[Ramen == "Bowl"] <- "Cup"
Ramen
Ramen[Ramen == "x"] <- "Bowl"
Ramen
```

10. 함수의 사용

```r
x <- seq(1:10)

sum(x)
mean(x)
var(x)
sd(x)
sqrt(x)
length(x)

x <- c(1, 2, -3)
abs(x)

summary(x)
```

10-1. 문자열의 결합 : `paste()`

```r
x <- c("이","박","김","최","차")
y <- c("순신", "원순",
"재경", "창원", "태균")
paste(x,y)
paste(x, y, sep=",")
paste(x, y, sep="")
```

10-2. 결측치 : `NULL` , `NA`

```r
# NULL, NA(not available)
x <- NULL
is.null(x)

y <- c(1, 2, 3, NA, 5)
y
```

10-3. `Inf` , `NaN`

```r
z <- 10/0
z

w <- 0/0
w
```

II. 배열

`array()` 함수 사용

1. 1차원 배열

```r
x <- array(1:3, dim=c(3))
x
```

2. 2차원 배열 만들기

2-1. `array()` 함수 사용

```
x <- array(1:6, dim=c(2, 3))     # dim=c(2, 3) : 2행, 3열
x

x <- array(c(2, 4, 6, 8, 10, 12), dim=c(2, 3))
x
```

`array(x, dim = c(a,b))`

인수 :

- `x` : 배열로 표현한 벡터
- `dim = c(a, b)` : 배열의 크기 지정, a개 행 b개 열

2-2. 벡터 결합으로 배열 만들기

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

x <- cbind(v1, v2, v3)   # cbind() : 열(column) 단위로 묶기(bind)
x

y <- rbind(v1, v2, v3)   # rbind() : 행(row) 단위로 묶기(bind)
y
```

3. 2차원 배열의 요소 값 보기와 수정

```
x[1, 3]

x[, 3]

x[,-3]

x[1, 2] <- 20
x
```

4. 2차원 배열의 행과 열 이름 추가

4-1. (방법 1) : `colnames()` , `rownames()` 함수 이용

```
x1 <- array(c(2, 4, 6, 8, 10, 12), dim=c(2, 3))
x1

colnames(x1) <- c("1열", "2열", "3열")
x1

rownames(x1) <- c("1행", "2행")
x1

x1["1행", ]

x1[ ,"2열"]
```

4-2. (방법 2) : `dimnames =` 인수 이용

```
names <- list(c("Row 1", "Row 2"), c("Col 1", "Col 2", "Col 3"))

x2 <- array(c(8, 10, 12, 14, 16, 18), dim=c(2, 3), dimnames=names)
x2
```

5. 배열 요소 값 확인

5-1. 배열의 색인에 의한 확인

```
x <- array(1:6, dim=c(2, 3))

x[1,3]      # 1행 3열 요소의 값 확인

x[1,]       # 1행 전체 요소의 값 확인
x[,3]       # 3열 전체 요소의 값 확인

x[,-3]      # 3열을 제외한 전체 요소의 값 확인
x[-1,]      # 1행을 제외한 전체 요소의 값 확인
```

5-2. 배열 요소의 값 비교에 의한 확인

```
x <- array(1:6, dim=c(2, 3))

x[ x<6 ]
x[ x>2 & x<6 ]
```

5-3. 행/열의 이름을 이용한 요소의 확인

```
names <- list(c("Row 1", "Row 2"), c("Col 1", "Col 2", "Col 3"))
x2 <- array(c(8, 10, 12, 14, 16, 18), dim=c(2, 3), dimnames=names)

x2["1행", ]
x2[ ,"2열"]
```

6. 배열 요소 값의 수정

6-1. 배열의 색인을 이용한 수정

```
x <- array(1:6, dim=c(2, 3))

x[1,2] <- 30
x
```

6-2. 배열 요소의 비교에 의한 확인

```r
x <- array(1:6, dim=c(2, 3))

x[ x<6 ]  <- 3
x
```

5. 3차원 배열

```r
x <- array(1:24, dim=c(2, 3, 4))   # 2행, 3열의 배열 4개
x
```

6. 함수 적용

```r
x <- array(c(2,4,6,8,10,12), dim=c(2,3))

apply(x, MARGIN = 1, sum)
apply(x, 1, mean)
apply(x, 1, sd)

apply(x, MARGIN = 2, sum)
apply(x, 2, mean)
apply(x, 2, sd)
```

`apply(x, MARGIN, FUN, …)`

인수 :

- `x` : 함수를 적용할 배열
- `MARGIN =` : `1` 이면 행단위, `2` 이면 열단위, `c(1, 2)` 이면 행과 열 동시에
- `FUN` : 적용할 함수 (sum, mena, sd, median, ... 사용자정의 함수도 가능)

III. 행렬

1. 행렬 만들기 : matrix() 함수이용.

```r
x <- matrix(1:6, nrow=2)
x

x <- matrix(1:6, nrow=2, byrow=TRUE)
x

x[1, 3]
```

2. 행과 열 이름 주기

```r
names <- list(c("1행", "2행"), c("1열", "2열", "3열"))

matrix(1:6, nrow=2, byrow=TRUE, dimnames=names)
```

3. 벡터 결합에 의한 행렬 만들기

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

x[c(2,4), c(1,3)]   # 행렬 중 2,4 행과, 1,3 열로 구성된 행렬 출력   <<<++++++++++
```

4. 행과 열 이름 달기

```
rownames(x) <- c("1행", "2행", "3행", "4행")
x

colnames(x) <- c("1열", "2열", "3열")
x

rbind(v1, v2, v3)
```

5. 대각행렬 만들기

대각행렬(diagonal matrix) : 행렬의 대각선 요소를 제외한 나머지 요소는 모두 `0` 인 행렬

```
x2 <- diag(1, 5)          # 항등행렬(Identity Matrix)
x2

x3 <- diag(10)
x3

x4 <- diag(1:10) ; x4

( x5 <- diag(c(1,3,5,7,9)) )
```

6. 행렬의 연산

6-1. 행렬의 곱

```
x <- matrix(c(1:6), ncol=3); x
y <- matrix(c(1,-1,2,3,2,-1), ncol=3); y

x*y

a <- matrix(c(2,1,0,1,2,1,0,1,2),ncol=3); a
p <- matrix(c(1,0,0,0,0,1,0,1,0),ncol=3); p

p %*% a       #  p의 열의 갯수와 a의 행의 갯수가 같아야 함.
```

6-2. 전치행렬(transpose matrix) : `t()` 함수 사용

```
x <- matrix(c(1:6), ncol=3); x
t(x)

x <- matrix(c(1:6), ncol=3); x
y <- matrix(c(1,-1,2,3,2,-1), ncol=3); y
z <- t(x)%*%y ; z
```

6-3. 역행렬(martix inversion) : solve()

A가 n×n 행렬일 때, 아래를 만족하는 n×n 행렬 B가 존재하면, B를 A의 역행렬이라고 하고, A−1로 표시함.

AB = BA = I (identity matrix)

```
a <- matrix(c(1,2,3,3,0,1,5,4,2), ncol=3); a
solve(a)
```

6-4. 행렬식(determinant)

```
a <- matrix(c(1,2,3,4,5,6,7,8,9), c(3,3))
det(a)

b <- matrix(c(0,1,2,3,4,5,6,7,9), c(3,3))
det(b)
```

IV. 리스트

1. 리스트 만들기 : list( ) 함수 이용

```
x <- list("홍길동", "2016001", 20, c("IT융합", "데이터 관리"))
x

str(x)

# 참고자료 :
# https://jeongchul.tistory.com/485
# https://jeongchul.tistory.com/486?category=558492
```

1-1. 리스트의 원소 출력

```
x[1]          # 1번째 요소 출력, [] 하나만 사용되면 부분 리스트.

x[[1]]        # 1번째 요소 출력, [[]] 두개면 벡터로 처리됨.

x[4]          # 4번째 요소 출력 {리스트}   : x[4]가 벡터 요소임.

x[4][1]       # x[4]와 같음 (리스트)


x[[4]]        # 4번째 요소를 벡터로 출력

x[4][[1]]     # x[[4]]와 같음.
```

```
x[[4]][1]      # 4번째 요소의 첫 번째 요소를 출력

x[[4]][2]      # 4번째 요소의 두 번째 요소를 출력
```

1-2. []와 [[ ]]의 차이

- `[ ]` : 리스트출력
- `[[ ]]` : 벡터출력
- `[ ][ ]` : 리스트출력
- `[ ][[ ]]` : 벡터출력
- `[[ ]][ ]` : 벡터

```
x[c(1,3)]      # 1번째 요소와 3번째 요소로 구성된 리스트 출력

x[c(1,4)]      # 1번째 요소와 4번째 요소로 구성된 리스트 출력
```

1-3. 리스트에 원소 추가

```
x[[5]] <- c("서울", "동대문")    # 5번째 요소로 문자벡터 추가
x

x[6:8] <- c(FALSE, TRUE, FALSE)  # 6 ~ 8번째 요소 추가
x
```

1-4. 리스트에서 구성요소 제거

```
x[[7]] <- NULL
x
```

1-5. 리스트 합치기

```
x1 <- list(3, 4)
x2 <- list("Son", "Lee")

x3 <- c(x1,x2)
x3
```

1-6. 리스트 요소의 갯수 확인 : length()

```
length(x)

length(x1)

length(x2)

length(x3)
```

2. 요소별로 키워드 지정하기

```
y <- list( 성명 = "홍길동", 학번 = "2016001", 나이 = 20, 수강과목 = c("IT융합", "데이터 관리") )
y

str(y)              # 변수 y의 구조 보기
```

2-1. 리스트의 원소 출력

```
y["성명"]           # '성명' 요소를 리스트로 출력

y[["성명"]]         # '성명' 요소를 벡터로 출력

y$성명              # '성명' 요소를 벡터로 출력

y$수강과목          # '수강과목' 요소를 벡터로 출력

y$수강과목[1]       # '수강과목'의 첫번째 요소를 출력  <<<########
```

2-2. 리스트에 원소 추가

```
y$취미 <- "낚시"
y
```

2-3. 리스트에서 구성요소 제거

```
y$취미 <- NULL
y
```

2-4. 리스트 합치기 : `c()` 이용

2-5. 리스트 요소의 갯수 확인 : length()

V. 데이터 프레임

두 명의 고객 정보에 대한 데이터 프레임 만들기 (열(column)이 하나의 요소가 됨)

```
# ----------------------
#   성명     나이     주소
# ----------------------
#  홍길동     20     서울
#  손오공     30     부산
```

1. 데이터 프레임 생성 <방법 1>

```r
name <- c("홍길동", "손오공")
age <- c(20,30)
address <- c("서울", "부산")

student <- data.frame(name, age, address)  # 3개의 열 벡터로 데이터 프레임 생성
student         # 벡터 이름이 자동으로 열의 이름으로 지정됨

str(student)    # student 변수의 구조 보기
```

1-1. 옵션 : stringsAsFactors= FALSE

```r
student1 <- data.frame(name, age, address, stringsAsFactors=FALSE)
student1

str(student1)
```

2. 데이터 프레임 생성 <방법 2>

```r
x <- data.frame(성명=c("홍길동", "손오공"), 나이=c(20, 30), 주소=c("서울", "부산"))
x

str(x)          # x 변수의 구조 보기
# 동일한 결과

x <- data.frame("성명"=c("홍길동", "손오공"), "나이"=c(20, 30), "주소"=c("서울", "부산"))
x
```

3. 데이터 프레임에 열과 행 단위 추가

3-1. 열 추가

```r
x <- cbind(x, 학과=c("전산학", "경영학"))  # 새로운 열요소로 학과 추가
x
```

3-2. 행 추가

```r
x <- rbind(x, data.frame(성명="장발장", 나이=25, 주소="파리", 학과="전산학"))
x

x2 <- list(성명="장발장", 나이=25, 주소="파리", 학과="전산학")
x3 <- rbind(x, x2 )
x3
```

4. 데이터 프레임의 요소 값 확인

4-1. 색인으로 확인하기

```r
x[3, 2]      # 3번째 행의 2번 째 열에 있는 요소 값 하나 출력

x[3,]        # 3번째 행을 하나의 데이터 프레임으로 출력

x[, 2]              # 2번째 열을 하나의 벡터로 출력
```

```r
x[, 2, drop=FALSE]   # drop=FALSE : 열의 형태로 벡터 출력

x[-2,]       # 2번째 행을 제외한 나머지를 데이터 프레임으로 출력

x[[4]]         # 4번째 요소 -> 벡터 출력

x[4]           # 4번째 벡터의 두 번째 요소

x[1, 2] <- 21  # 요소 값 변경
x
```

4-2. 열의 이름으로 확인하기

```r
x["나이"]   # 나이 열만 하나의 <데이터프레임>으로 출력  -> x[ ,2]와 다름

x$나이      # 나이 열만 하나의 <벡터>로 출력 -> x[ ,2]와 같음

x["학과"]   # 학과 열만 하나의 <데이터프레임>으로 출력

x$학과      # 학과 열만 하나의 <벡터>로 출력

x[["학과"]]  # 학과 열의 factor를 출력 => 경영학, 전산학

x[["성명"]]  # 성명 열의 factor를 출력
```

4-3. 요소 값의 비교에 의한 확인

```r
x[ x$Dept=='전산학', ]

x[x$Age >= 25,   ]   # 나이가 25세 이상(x$Age >= 25)인 행 전체 출력
```

4. 요소 값 수정

```r
x[1, "나이"] <- 22
x

x[1, 2] <- 22
x

x[1, "Age"] <- 20
x
```

5. 데이터 프레임의 factor 데이터 수정

data frame에서 문자열 항목은 데이터 유형이 factor형으로 인식된다.

이를 문자형으로 바꾸려면 다음과 같이 처리한다.

```r
str(x)                                # 데이터 프레임 변수 x의 구조 확인
x[3] <- lapply(x[3], as.character)  # x[3]의 데이터 타입이 factor인 것을 문자 타입으로 변환
str(x)

# 장발장의 주소를 "제주"로 변경
x[3, 3] <- "제주"
x
```

VI. 데이터 세트

1. 데이터 세트 목록 보기

1-1. 데이터 세트 목록 보기

```r
library(help = datasets)    #  패키지 'datasets'에 대한 설명
data()        # Data sets in package 'datasets' (91 ~ 93 페이지)
```

1-2. 데이터 세트의 이용

목록에 있는 변수명이 데이터 세트이다.

```r
AirPassengers
BOD

Titanic
str(Titanic)

quakes
head(quakes, n=10)      # quakes의 앞 부분의 10개 데이터 보기
tail(quakes, n=10)        # quakes의 뒤 부분의 10개 데이터 보기
```

1-3. 데이터 세트의 구조 보기

```r
names(quakes)   # quakes의 변수 명 확인
str(quakes)        # quakes의 구조 확인
dim(quakes)       # quakes의 행과 열의 개수
```

1-4. 데이터 세트의 요약 정보

```r
summary(quakes)
summary(quakes$mag)
```

1-5. 데이터 세트의 저장

```r
setwd("c:/temp")        # 저장할 폴더 지정
write.table(quakes, "quakes.txt", sep=",")    # ,로 구분하여, csv파일로 저장하기
```

1-6. 저장한 파일을 data 변수로 불러오기

```
data <- read.csv("quakes.txt", header=TRUE)  # quakes.txt 파일을 data변수로 불러오기
data
```

2. 데이터 세트 구조 보기

```
names(quakes)
str(quakes)
dim(quakes)
```

3. 데이터 세트 요약 보기

```
summary(quakes)

summary(quakes$mag)
```

4. 데이터 세트 저장하고 읽기

```
write.table(quakes, "c:/temp/quakes.txt", sep=",")

x <- read.csv("c:/temp/quakes.txt", header=T)
x

x <- read.csv(file.choose(), header=T)
x
```

VII. 웹 사이트의 데이터 불러오기

1. 웹사이트의 데이터 파일 읽기

웹 사이트에 저장되어 있는 csv 파일을 직접(하드디스크에 저장하지 않고) R로 가져온다

원하는 데이터 세트의 'CSV'링크에 마우스를 갖다 놓고, 마우스의 오른쪽 버튼 클릭하여, "링크 주소 복사"를 클릭한다.

(예: Titanic의 CSV). https://vincentarelbundock.github.io/Rdatasets/csv/Stat2Data/Titanic.csv

```
url <- "https://vincentarelbundock.github.io/Rdatasets/csv/datasets/
Titanic.csv"
x <- read.csv(url)
x
```

VIII. 사용자 정의 함수

1. 함수 만들기

1-1. 원의 면적을 구하는 함수의 정의와 호출

```r
getCircleArea <- function(r) {
    area <- 3.14 * r^2
    return(area)
}

getCircleArea(3)
```