02. DATA TYPE

변수

- 개념
  - 값을 저장하고 추출할 수 있는 데이터 바구니
- 이름 규칙
  - 알파벳 또는 마침표(.)로 시작
  - 시작하는 문자 외에 숫자, 언더바(_)를 사용할 수 있으나, 하이픈(-)은 불가
  - Ex) a.val b_12 .a10
- 값 할당
  - 연산자 : = <- <<-
  - = Vs. <-
    - 함수를 내부에서 값 할당을 할 때는 <- 연산자만 사용가능
    - Ex)

```
> mean(a = c(1, 2, 3))  # 불가
Error in mean.default(a = c(1, 2, 3)) :
  argument "x" is missing, with no default
> mean(a <- c(1, 2, 3))  # 가능
```

```
mean(x = a <- c(1, 2, 3))  # 가능
```

- 값 확인
  - Ex)

```
> class('asd')
[1] "character"
> class(b)
[1] "numeric"
```

스칼라

- 개념
  - 단일 차원의 값, 한 개의 값만 수용가능
- 타입
  - 숫자[numeric]
  - 문자[character] : '문자', "문자2"
  - 진릿값/논리값[logical]
  - 팩터[factor] : 범주형 자료 표현, 명목형/순서형으로 나뉨
  - NA[Not Available]
    - missing value, ~~값이 없음~~
    - NA값이 들어있는지 확인 : is.na(x)
    - Ex)

```
> sum(a <- c(1, 2, 3, NA, 4))
[1] NA
> sum(a, na.rm = TRUE)
[1] 10
```

  - NULL
    - 변수가 초기화되지 않음
    - 값이 없음, 또는 값이 정해지지 않았다고 보는 것이 이해하기에 쉬움
    - NULL값의 확인 : is.null(x)

팩터[factor]

- 종류
  - 명목형[Nominal]
    - 크기 비교가 불가능한 팩터 값, default 형태
  - 순서형[Ordinal]
    - 크기 비교가 가능한 팩터 값
- 관련 함수
  - 팩터 값 생성
    - factor(x, levels=c('m','f'), ordered=FALSE)
  - 레벨 개수 조회
    - nlevels(x)
  - 팩터여부 확인
    - is.factor(x)
  - 순서형 팩터 생성
    - ordered(x)
  - 순서형 팩터 여부 확인
    - is.ordered(x)

```
> x = factor(c('a','b'))
> nlevels(x)
[1] 2
> is.ordered(x)
[1] FALSE
```

```
> levels(x)[]
[1] "a" "b"
> levels(x)[0]
character(0)
> levels(x)[1]
[1] "a"
> levels(x)[2]
[1] "b"
```

Values	
x	Ord.factor w/ 2 levels "a"<"b": 1 2
y	Factor w/ 2 levels "a","b": 1 2

벡터[vector]

- 개념
  - 일종의 배열(array)
  - 한 가지 타입만 저장 가능(숫자 or 문자 등)
  - 여러 타입의 값을 벡터에 저장하면 자동으로 형 변환
- 관련 함수
  - 생성 : c('a', 'b') c(1, 2) c(1, 2, c(3, 4, 5)) c('1', 2, '3')
  - 벡터명 확인
    - names(x) # 각각의 값마다 따로 존재
  - 각각의 객체명 저장
    - names(x) <- c('title1', 'title2')
  - 저장된 값 사용
    - x[n] : x의 n번째 값, 여러 개는 x[c(1,3)]과 같이 조회함
    - x[-n] : x에서 n번째를 제외한 나머지 값
    - x[n:m] : n부터 m번째까지의 값
    - x['title1'] : 객체명으로 조회
  - 길이
    - length(x) : x의 길이 조회
    - NROW(x) : 배열의 행 또는 열의 수 조회
    - nrow(x) : 행렬만 조회 가능, 벡터는 불가
- 벡터 연산
  - 객체가 동일한지 판단
    - identical(x, y) # TRUE or FALSE
      - 객체비교에서 == 또는 !=을 사용하면 값을 각각 비교한 T/F를 반환하므로, 조건문에서는 indentical()을 사용해야 함
  - 벡터가 동일한 값을 담고 있는지 판단
    - setequal(x, y) # TRUE or FALSE
    - 값의 중복여부와 상관 없이 비교함
  - 합집합
    - union(x, y) # x, y에 들어있는 모든 값을 반환
  - 교집합
    - intersect(x, y) # 교집합이 되는 값을 반환
  - 차집합
    - setdiff(x, y) #
  - 어떤 값이 포함되어 있는지 판단
    - 'a' %in% c('a','c') # TRUE or FALSE
- 연속된 숫자로 구성된 벡터
  - seq(from, to, by) # default : by=1, 1씩 증가 or 감소
  - seq_along(along.with=x) # 1부터 x의 길이만큼 숫자 생성
  - n:m : n부터 m까지의 숫자 생성
- 반복된 값을 저장한 벡터
  - req(x, times, each) # x를 times만큼 반복, 각 값은 each만큼 반복

리스트[list]

- 개념
  - python의 dictionary 타입과 유사
  - 벡터와 달리 여러 데이터 타입을 담는 것이 가능
  - 리스트 안에 리스트를 중첩 가능
- 생성
  - x <- list(name='foo', height=70)
  - x <- list(name='foo', height=c(1, 2, 3))
- 데이터 접근
  - x$height # height 키 값에 접근
  - x[n]
    - x의 n번째 데이터 접근
    - key와 value를 담은 리스트를 반환
  - x[[n]]
    - x의 n번째 데이터 접근
    - key 없이 value만 반환

행렬[matrix]

- 개념
  - row와 column으로 구성
  - 한 가지 유형의 값만 저장 가능
- 생성
  - matrix(data, nrow, ncol, byrow=FALSE, dimnames=NULL)
    - nrow/ncol : 행/열의 수 지정
    - byrow : 행우선 여부
    - dimnames : 차원의 이름 설정 # list(row,col)
  - dimnames(x) # 각 차원에 대한 이름 반환
    - dimnames <- value # 각 차원에 이름 설정
  - rownames(x) # 행이름 반환
    - rownames <- value # 각 행에 이름 설정
  - colnames(x) # 열이름 반환
    - colnames <- value # 각 열에 이름 설정
- 데이터 접근
  - x[1:2,] x[, 'c1'] # 행번호/열번호 or 행이름/열이름
- 관련 함수
  - t(x) : 전치행렬(행열 전환)
  - solve(x) : 역행렬
  - nrow(x)/ncol(x) : 행의 수 또는 열의 수 반환
  - dim(x) : 차원 수를 반환, 행 by 열

(다차원)배열[array]

- 개념
  - 행렬이 2차원인데 비해 배열은 다차원 데이터
- 생성
  - array(data=NA, dim=1, dimnames=NULL)

```
> array(c(1:12), c(2,2,3))
, , 1

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

, , 2

     [,1] [,2]
[1,]    5    7
[2,]    6    8

, , 3

     [,1] [,2]
[1,]    9   11
[2,]   10   12
```

```
> x[ , , 3]
     [,1] [,2]
[1,]    9   11
[2,]   10   12
```

데이터프레임[data frame]

- 개념
  - 행렬과 유사한 형태의 행/열 구조
  - 여러 가지 데이터 타입을 담을 수 있음
- 생성
  - data.frame(x, y)
  - data.frame(c(1, 2, 3), row.names=c('a','b','c'))
- 관련 함수
  - names(x) / colnames(x) # 열이름 지정
  - rownames(x) # 행이름 지정
  - str(x) # 데이터프레임의 구조 확인
  - head(x, n=6)
  - tail(x, n=6)
  - View(x, title='window title') # 대소문자 주의
- 데이터 접근
  - d$colname
  - d[m, n, drop=FALSE] # drop : 반환 데이터의 형변환 금지
    - n, m은 몇 번째 자리인지(스칼라, 벡터) 또는 행/열이름으로 조회
    - 행이름/컬럼명 지정시 따옴표 사용 필수

타입 판별

- class(x) : 데이터 타입 조회
- str(x) : 데이터의 내부 구조 조회
- is.factor(x) : 팩터 여부 판별
- is.numerix(x) : 숫자를 저장한 벡터인지 판별
- is.character(x) : 문자열을 저장한 벡터인지 판별
- is.matrix(x) : 행렬 객체인지 판별
- is.array(x) : 배열 객체인지 판별
- is.data.frame(x) : 데이터프레임 객체인지 판별

타입 변환

- as.factor(x) : 팩터로 변환
- as.numerix(x) : 숫자 벡터로 변환
- as.character(x) : 문자열 벡터로 변환
- as.matrix(x) : 행렬 객체로 변환
- as.array(x) : 배열 객체로 변환
- as.data.frame(x) : 데이터프레임으로 변환
- as.Date(x, format='%Y%m%d') : 날짜 객체로 변환

03. R PROGRAMMING

R의 특징

- 데이터를 다루는 방법
  - 다른 프로그래밍 언어에서는 for문을 이용하여 1행씩 처리하지만, R에서는 전체 데이터를 한꺼번에 다루는 벡터연산을 더 자주 이용함
  - 단지 코딩스타일의 차이가 아니라 속도도 빠름
- 결측치 처리
  - NULL : 빈 값
  - NA : 관측되지 않은, 기록되지 않은 값
  - 연산 또는 비교시에 주의 필요
- 객체의 불변성
  - 대부분의 R객체는 그 값을 수정 불가
  - 문법적으로 객체의 데이터를 수정하는 코드이더라도 실제로는 수정될 값이 들어있는 새로운 객체가 생성
- Functional Language

흐름제어(조건문/반복문)

- 조건문(if)
  - 문법
    - if (a==b) {
      a==b가 참일 때 실행할 문장
      } else {
      a==b가 거짓일 때 실행할 문장
      }
    - a <- c(1, 2, 3, 4, 5)
      ifelse(a=b, '참', '거짓')
  - if와 ifelse의 차이점
    - if는 한 개의 값에 대한 연산만을 하지만, ifelse는 다중연산이 가능
- 반복문(for/while/repeat)
  - 문법
    - for (i in data) { print(i) }
    - while (i <= 10) { print(i) ; i = i + 1 }
    - repeat {
      if (i >= 10) { break }
      i = i + 1
      }
  - 반복문의 조정
    - break : 반복문을 종료
    - next : 현재 수행중인 반복 term을 종료하고, 다음 term 시작

연산(수치연산/벡터연산)

- 수치연산
  - 연산자와 함수
    - 사칙연산 : +, -, *, /
    - n을 m으로 나눈 나머지 : n %% m
    - n을 m으로 나눈 몫 : n %/% m
    - n의 m 제곱 : n^m
    - e의 n 제곱 : exp(n)
- 벡터연산
  - 다대일 연산
    - c(1, 2, 3, 4, 5) + 1 # 2 3 4 5 6
  - 다대다 연산
    - c(1, 2, 3, 4, 5) + c(1, 2, 3, 4, 5) # 2 4 6 8 10
    - c(1, 2, 3, 4, 5) == c(1, 2, 3, 4, 5) # TRUE TRUE TRUE TRUE TRUE
  - 함수
    - sum(c(1,2,3)) # 6
    - mean(c(1,2,3)) # 2
    - median(c(1,4,5)) # 4
  - 조건부 특정행 선택
    - a[a$x %% 2 == 0,] # 짝수행

NA 처리

- 관련 함수 옵션
  - NA 제외하고 계산
    - 다수의 연산함수에서 na.rm을 옵션으로 제공
    - sum(c(1,2,3), na.rm=TRUE)
- NA 처리 함수
  - na.fail(x) : NA가 포함되어 있을 경우, 에러 발생
  - na.omit(x) : NA가 포함된 행을 제거하고 반환
  - na.exclude(x) : NA가 포함된 행을 제거하되, 별도로 사용가능
  - na.pass(x) : NA 여부와 상관없이 통과

함수[function]

- 개념
  - 코드의 반복을 줄이거나 가독성을 높이기 위해 사용하는 코드 묶음
- 함수 생성
  - new_function_name <- function(인자, 인자, ...) {
    함수 본문
    return(반환 값) # 생략 가능
    }
  - 인자를 받는 란에 '...'을 이용하면 가변 길이로 입력 가능
  - 입력받은 인자 값을 함수 본문에서 사용하기 위해서 'args' 사용
    - for (i in args) {
      print(i)
      }

```r
# substr(x, start, stop)
substr(x = 'abcdefg', 5, 2)
substr(x = 'abcdefg', stop = 5, sta = 2)
```

스코프[scope]

- 개념
  - 변수를 사용가능한 영역
  - 변수 종류 : 전역변수, 지역변수
- 전역변수
  - 함수 내/외부 등 전체 영역에서 사용가능
  - 콘솔에서 선언했을 경우
- 지역변수
  - 지정된 영역, 함수 내부에서 사용가능
  - 함수 내에서 선언하거나 함수의 인자인 경우
  - 함수 내부에 선언된 변수는 외부에서 접근 불가

```r
n <- 3  # global
f <- function() {
  print(n)  #3
  n <- 5 # local
  print(n)  #5
}
f()
print(n)  #3
```

그 밖에..

- 값에 의한 전달
- 객체의 불변성
- 모듈 패턴