



문공 A0015

R 프로그래밍

김 태 완

kimtwan21@dongduk.ac.kr

변수와 벡터

- R의 기본 연산


- 산술연산

: R에서 산술연산은 사칙연산을 그대로 사용한다.

연산자	의미
+	덧셈
-	뺄셈
*	곱셈
/	나눗셈
%%	나눗셈의 나머지
^ 또는 **	제곱

```
> 3+5+8
[1] 16
> 9-3
[1] 6
> 7*5
[1] 35
> 8/3
[1] 2.666667
> 8%%3
[1] 2
> 2^3
[1] 8
```

변수와 벡터

- R의 기본 연산
 - 산술연산 함수
 - 함수의 구조 : '함수명 + 괄호' ex)  log(10)

함수	의미
log()	로그함수
sqrt()	제곱근
max()	가장 큰 값
min()	가장 작은 값
abs()	절댓값
factorial()	팩토리얼
sample()	표본 자동 생성
sin(), cos(), tan()	삼각함수

```
> log(10)
[1] 2.302585
> sqrt(36)
[1] 6
> max(3, 9, 5)
[1] 9
> min(3, 9, 5)
[1] 3
> abs(-10)
[1] 10
> factorial(5)
[1] 120
> sample(5) sample(1:10,4)
[1] 5 3 1 4 2
> sin(pi/2)
[1] 1
```

변수와 벡터

- 변수
 - 변수의 개념

```
1  a <- 10
   #10을 변수 a에 저장
2  b <- 20
   #20을 변수 b에 저장
3  c <- a+b
   #변수 a의 값과 변수 b의 값을 더하여 변수 c에 저장
4  print(c)
   #변수 c의 값을 출력
```



```
> a <- 10
> b <- 20
> c <- a+b
> print(c)
[1] 30
```

변수와 벡터

- 변수
 - 변수명 지정

1) 첫 글자는 영문자(알파벳)나 마침표(.)로 시작

ex) avg, .avg

cf) 12th -> 숫자로 시작했기 때문에 불가

2) 두 번째 글자부터는 영문자, 숫자, 마침표(.), 밑줄(_) 사용 가능

ex) v.1, a_sum

cf) this-data, this@data -> @, -같은 특수문자 사용 불가

3) 대문자와 소문자를 구분

ex) var_A와 var_a는 서로 다른 변수

4) 변수명 중간에 빈칸을 넣을 수 없음

ex) first ds -> X

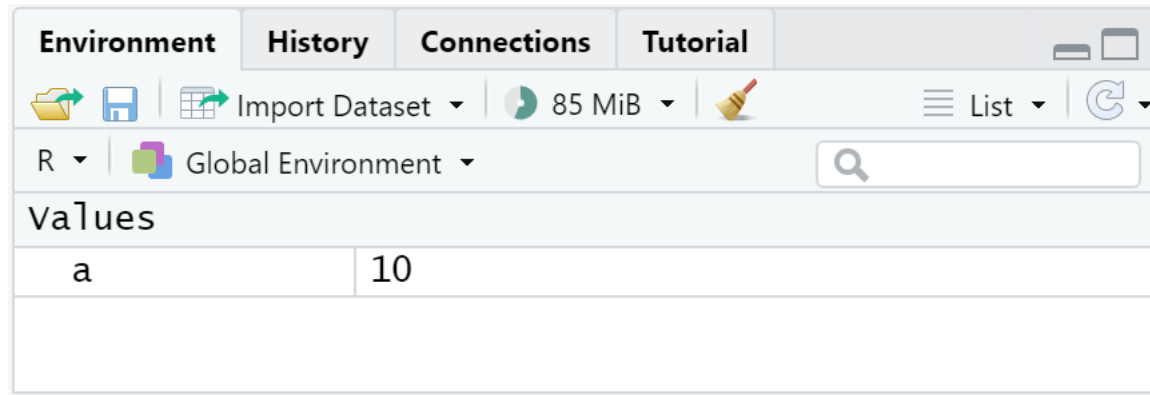
변수와 벡터

- 변수
 - 변수에 값 저장 및 확인

-변수에 값 저장하기

변수명 \leftarrow a \leftarrow 10 값
변수에 값을 저장하는 연산자
([Alt] + [-])

-변수에 저장된 값 확인하기 : 환경창



변수와 벡터

- 변수
 - 변수의 자료형

자료형	사용 예	비고
숫자형	1, 2, 3, -4, 3.14	정수와 실수 모두 가능
문자형	'Tom', "Jane"	작은 따옴표나 큰 따옴표로 묶어서 표현
논리형	TRUE, FALSE	반드시 따옴표가 없는 대문자로 표현 T나 F로 줄여서 사용 가능
특수값	NULL	정의되어 있지 않음을 의미
	NA	결측값(missing value)
	NaN	수학적으로 정의가 불가능한 값 ex) sqrt(-3)
	Inf, -Inf	양의 무한대, 음의 무한대

• 프로그램의 실행 결과가 아무것도 없는 경우에 표시

• 변수를 정의해야 하는데 초기값을 어떤 것으로 해야할지 애매할 때 사용

• 자료형이 없으며 길어도 0

• 'Not Applicable'의 약자로 결측값 또는 누락된 값을 나타냄

ex) 3번 문항으로 구성된 설문조사에서 응답자가 1번 문항에 응답하지 않았다면 (NA, 1, 2)와 같이 표시

변수와 벡터

- 변수
 - 변수의 값 변경
 - 마지막에 변경한 값이 저장됨

```
1 a <- 10
2 b <- 20
3 a + b
4 a <- "A"
5 a + b
6 a <- 30
7 a + b
```



```
> a <- 10
> b <- 20                                # a = 10, b = 20
> a + b
[1] 30
> a <- "A"                                # a = "A", b = 20
> a + b
# 에러 발생 -> 문자형과 숫자형의 산술연산은 불가
> a <- 30                                # a = 30, b = 20
> a + b
[1] 50
```


변수와 벡터

- 변수
 - 변수의 자료형에 이용되는 함수

함수	설명
class()	자료의 데이터의 유형 식별
is.numeric()	자료가 숫자형 여부인지를 판별
is.character()	자료가 문자형 여부인지를 판별
is.logical()	자료가 논리값 여부인지를 판별
is.null()	자료가 NULL 여부인지를 판별

```
> x <- 4
> y <- "홍길동"
> z <- TRUE
> class(x)
[1] "numeric"
# 숫자형
> class(y)
[1] "character"
# 문자형
> class(z)
[1] "logical"
# 논리형
```

변수와 벡터

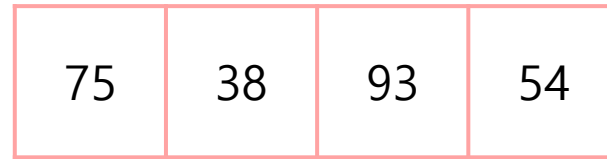
- 벡터의 이해

- 벡터의 개념 : 데이터 구조의 가장 기본적인 형태로, 1차원 형태의 데이터를 저장할 수 있는 저장소



[하나의 값이 저장된 변수]

1개의 값만 저장



[벡터가 저장된 변수]

여러 개의 값을 저장

- 벡터 만들기

- 벡터 생성 함수 : `c()`

'connect'의 약자. 여러 개의 값을 묶어주는 역할

- 하나의 벡터에는 동일한 종류의 자료형이 저장되어야 함

ex) 숫자형 -> 숫자형 벡터, 문자형 -> 문자형 벡터, 논리형 -> 논리형 벡터

- 하나의 벡터에 복수의 자료형이 있을 경우 하나의 자료형으로 변환되어 저장 (주의할 것)

`str()` 함수로 확인 가능

변수와 벡터

- 벡터의 이해
 - 벡터 만들기
 - 하나의 벡터에는 동일한 종류의 자료형이 저장되어야 함

```
1 x <- c(1, 2, 3)
#숫자형 벡터
2 y <- c("a", "b", "c")
#문자형 벡터
3 z <- c(TRUE, FALSE, TRUE)
#논리형 벡터
4 w <- c(1, "two", 3)
5 x
6 y
7 z
8 w
```

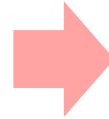


```
> x <- c(1, 2, 3)
> y <- c("a", "b", "c")
> z <- c(TRUE, FALSE, TRUE)
> w <- c(1, "two", 3)
> x
[1] 1 2 3
> y
[1] "a", "b", "c"
> z
[1] TRUE FALSE TRUE
> w
[1] "1", "two", "3"
# 문자형 "two"로 인하여
  문자형 벡터가 됨
```

변수와 벡터

- 벡터의 이해
 - 벡터 만들기
 - 연속적인 숫자로 이루어진 벡터 생성 : 콜론(:) 이용

```
1 v1 <- 10:20
2 v1
3 v2 <- c(1, 2, 3, 10:20)
# c() 함수 안에서 사용하는 것도 가능
4 v2
```



```
> v1 <- 10:20
> v1
[1] 10 11 12 13 14 15 16 17 18 19
[11] 20
> v2 <- c(1, 2, 3, 10:20)
> v2
[1] 1 2 3 10 11 12 13 14 15 16
[11] 17 18 19 20
```

변수와 벡터

- 벡터의 이해

- 벡터 만들기

- 일정한 간격의 숫자로 이루어진 벡터 생성 : seq(시작값, 종료값, 간격) 함수 이용

```
1 v3 <- seq(from=1, to=11, by=3)
2 v3
3 v4 <- seq(0.1, 1.0, 0.1)
# 간격은 소수도 가능
4 v4
5 seq(from=0, to=100, length.out=5)
6 seq(from=-1, to=1, length.out=5)
```

↪ 함수



```
> v3 <- seq(1, 11, 3)
> v3
[1] 1 4 7 10
> v4 <- seq(0.1, 1.0, 0.1)
> v4
[1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8
[9] 0.9 1.0
```

- [숫자]의 의미
[9]는 두 번째 줄의 맨 앞에 있는 0.9가 전체 값 중에 9번째라는 것을 의미
- 출력 결과값이 많을 때 순서 파악 용이

변수와 벡터

- 벡터의 이해

- 벡터 만들기

-반복된 숫자로 이루어진 벡터 생성 : rep(반복대상 값, 반복횟수) 함수 이용

```
1 v5 <- rep(1, times=5)
# 1을 5번 반복
2 v6 <- rep(1:5, times=3)
# 1에서 5까지 3번 반복
3 v7 <- rep(c(1, 5), times=3)
4 v8 <- rep(c(1, 5), each=3)
# each를 쓰면 반복대상 값 하나하나를
  각각 반복할 수 있다.
5 v9 <- rep(c(1, 5), length.out=5)
6 v10 <- rep(c(1, 5), times=c(3, 2))
```



```
> v5 <- rep(1, times=5)
> v5
[1] 1 1 1 1 1
> v6 <- rep(1:3, times=3)
> v6
[1] 1 2 3 1 2 3 1 2 3
> v7 <- rep(c(1, 5), times=3)
> v7
[1] 1 5 1 5 1 5
> v8 <- rep(c(1, 5), each=3)
> v8
[1] 1 1 1 5 5 5
```

변수와 벡터

- 벡터의 이해
 - 벡터의 원소 값에 이름 지정 : names() 함수 이용

```
1  score <- c(90, 85, 70)
# score라는 변수에 벡터를 입력
2  score
3  names(score)
4  names(score) <- c("A", "B", "C")
# 값들에 이름을 부여
5  names(score)
6  score
```



```
> score <- c(90, 85, 70)
> score
[1] 90 85 70
> names(score)
NULL
#아무것도 없다는 의미
(이름을 설정하지 않았기 때문)
> names(score) <- c("A", "B", "C")
> names(score)
[1] "A" "B" "C"
> score
  A  B  C
90 85 70
```

변수와 벡터

- 벡터의 이해

- 벡터에서 원소 값 추출 : 인덱스 이용

-d <- seq(1, 9, 2)

d	1	3	5	7	9
	d[1]	d[2]	d[3]	d[4]	d[5]

```
> d[1]
```

```
[1] 1
```

```
> d[2]
```

```
[1] 3
```

```
> d[3]
```

```
[1] 5
```

```
> d[4]
```

```
[1] 7
```

```
> d[5]
```

```
[1] 9
```

```
> d[6]
```

```
[1] NA
```

d[6]에 대응하는 값이 없으므로
결측 값을 의미하는 NA가 출력

변수와 벡터

- 벡터의 이해

- 벡터에서 원소 값 추출

- 벡터에서 여러 개의 값을 한 번에 추출하기 : 콜론(:), seq함수, 벡터 이용

```
1 d <- seq(1, 9, 2)
2 d[c(1, 3, 5)]
# 1, 3, 5번째 값 출력
3 d[1:3]
# 처음 세 개의 값 출력
4 d[seq(1, 5, 2)]
# 홀수 번째 값 출력 (1, 3, 5번째 값 출력)
5 d[-2]
# 2번째 값 제외하고 출력
6 d[-c(3:5)]
# 3~5번째 값은 제외하고 출력
```



```
> d <- seq(1, 9, 2)
> d[c(1, 3, 5)]
[1] 1 5 9
> d[1:3]
[1] 1 3 5
> d[seq(1, 5, 2)]
[1] 1 5 9
> d[-2]
[1] 1 5 7 9
> d[-c(3:5)]
[1] 1 3
```

변수와 벡터

- 벡터의 이해

- 벡터에서 원소 값 추출
 - 벡터에서 이름으로 값을 추출하기

```
1 score <- c(90, 85, 70)
2 score
3 names(score) <- c("A", "B", "C")
4 score
5 score[1]
# score의 첫 번째 값 출력
6 score["A"]
# 이름이 "A"인 값 출력
7 score[c("A", "C")]
# 이름이 "A", "C"인 값 출력
```



```
> score <- c(90, 85, 70)
> score
[1] 90 85 70
> names(score) <- c("A", "B", "C")
> score
A B C
90 85 70
> score[1]
A
90
> score["A"]
A
90
> score[c("A", "C")]
A C
90 70
```

변수와 벡터

- 벡터의 이해
 - 벡터에서 저장된 원소 값 변경

```
1 d <- seq(1, 9, 2)
2 d
3 d[2] <- 10
# d의 2번째 값을 10으로 변경
4 d
5 d[c(1, 4)] <- c(20, 30)
# d의 1, 4번째 값을
  각각 20, 30으로 변경
6 d
```



```
> d <- seq(1, 9, 2)
> d
[1] 1 3 5 7 9
> d[2] <- 10
> d
[1] 1 10 5 7 9
> d[c(1, 4)] <- c(20, 30)
> d
[1] 20 10 5 30 9
```

변수와 벡터

- 벡터의 연산

- 벡터와 벡터를 연산하기 위해서는 두 벡터의 길이가 같아야 함.

```
1 d <- seq(1, 9, 2)
2 2*d
3 d-5
# 연산을 실행하고 난 뒤
# d의 값에는 변화 없음
4 x <- c(1, 2, 3)
5 y <- c(4, 5, 6)
6 x + y
7 x * y
8 z<- x + y
# x, y를 더하여 z에 저장
9 z
```



```
> d <- seq(1, 9, 2)
> 2*d
[1] 2 6 10 14 18
> d-5
[1] -4 -1 0 2 4
> x <- c(1, 2, 3)
> y <- c(4, 5, 6)
> x + y
[1] 5 7 9
> x * y
[1] 4 10 18
> z<- x + y
> z
[1] 5 7 9
```

변수와 벡터

- 벡터의 연산

- 벡터와 벡터를 연산하기 위해서는 두 벡터의 길이가 같아야 함.
 - 여러 개의 벡터를 합쳐 새로운 벡터 만들기 : `c()` 함수 이용

```
1 x <- c(1, 2, 3)
2 y <- c(4, 5)
3 c(x, y)
# 단순히 x, y를 연결하여 출력
4 z <- c(x, y)
# x, y를 연결하여 z에 저장
5 z
```



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

변수와 벡터

- 벡터의 연산
 - 벡터에 적용 가능한 함수

함수명	설명
sum()	벡터에 포함된 값들의 합
mean()	벡터에 포함된 값들의 평균
median()	벡터에 포함된 값들의 중앙값
max(), min()	벡터에 포함된 값들의 최댓값, 최솟값
var()	벡터에 포함된 값들의 분산
sd()	벡터에 포함된 값들의 표준편차
sort()	벡터에 포함된 값들의 정렬(오름차순이 기본)
range()	벡터에 포함된 값들의 범위(최솟값~최댓값)
length()	벡터에 포함된 값들의 개수(길이)

변수와 벡터

- 벡터의 연산
 - 벡터에 적용 가능한 함수 예시

```
> d <- seq(1, 10, 1)
> sum(d)
[1] 55
> length(d)
[1] 10
> mean(d[1:6])
[1] 3.5
> max(d)
[1] 10
> min(d)
[1] 1
```

```
> sort(d)
#오름차순 정렬(기본)
[1] 1 2 3 4 5 6 7 8 9 10
> sort(d, decreasing = FALSE)
#오름차순 정렬
[1] 1 2 3 4 5 6 7 8 9 10
> sort(d, decreasing = TRUE)
#내림차순 정렬
[1] 10 9 8 7 6 5 4 3 2 1
> median(d)
[1] 5.5
> sum(d) / length(d)
[1] 5.5
```

변수와 벡터

- 벡터의 연산

- 벡터에 적용 가능한 함수 : 함수와 매개변수

- 함수의 입력 값 = 매개변수
 - 하나의 함수가 여러 개의 매개변수를 가질 수 있음
 - 매개변수명 없이 매개변수 값만 입력해도 함수는 작동
 - 하지만 함수에 정의된 매개변수의 순서를 지켜야 함

매개변수명

> sort(d, decreasing = FALSE)

> sort(d, decreasing = TRUE)

매개변수값

```
> sort(x = d, decreasing = TRUE) # 정상 작동
> sort(x = d, TRUE)               # 정상 작동
> sort(d, TRUE)                   # 정상 작동
> sort(TRUE,d)                   # 오류 발생(매개변수의 순서가 잘못됨)
> sort(d)                        # 정상 작동
                                (함수에 decreasing = FALSE가 초기값으로 설정되어 있음)
```


변수와 벡터

- 벡터의 연산
 - 벡터에 논리 연산자 적용
 - 논리연산자 : 연산의 결과가 TRUE 또는 FALSE로 출력되는 연산자

연산자	사용 예	설명
<	A < B	B가 A 보다 크면 TRUE
<=	A <= B	B가 A 보다 크거나 같으면 TRUE
>	A > B	A가 B 보다 크면 TRUE
>=	A >= B	A가 B 보다 크거나 같으면 TRUE
==	A == B	A와 B가 같으면 TRUE
!=	A != B	A와 B가 같지 않으면 TRUE
	A B	A 또는 B 어느 한쪽이라도 TRUE면 TRUE
&	A & B	A와 B 모두 TRUE일 때만 TRUE

변수와 벡터

- 벡터의 연산
 - 벡터에 적용되는 논리연산자 예시

```
1 d <- seq(1, 9, 2)
2 d >= 5
3 d[d>5]
# 5보다 큰 값
4 sum(d>5)
# 5보다 큰 값의 개수를 출력
5 sum(d[d>5])
# 5보다 큰 값의 합계를 출력
```



```
> d <- seq(1, 9, 2)
> d >= 5
[1] FALSE FALSE TRUE TRUE TRUE
> d[d>5]
[1] 7 9
# d      : 1 3 5 7 9
# d>5    : F F F T T
# d[d>5] :      7 9
> sum(d>5)
[1] 2
# sum 함수에서 F = 0, T = 1
# d>5      : F F F T T
# sum[d>5] : (0+0+0+1+1) = 2
```

변수와 벡터

- 벡터의 연산
 - 벡터에 적용되는 논리연산자 예시

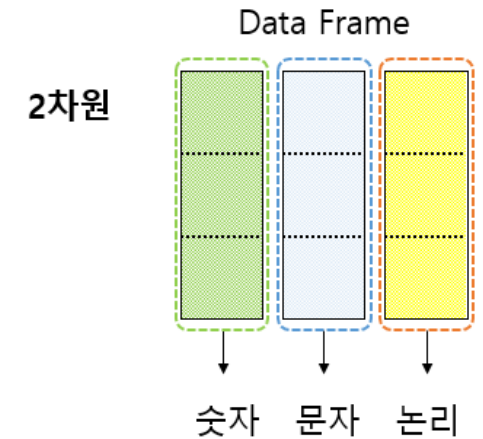
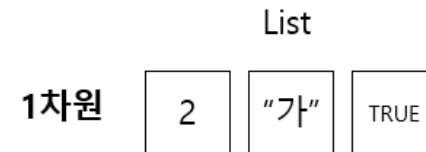
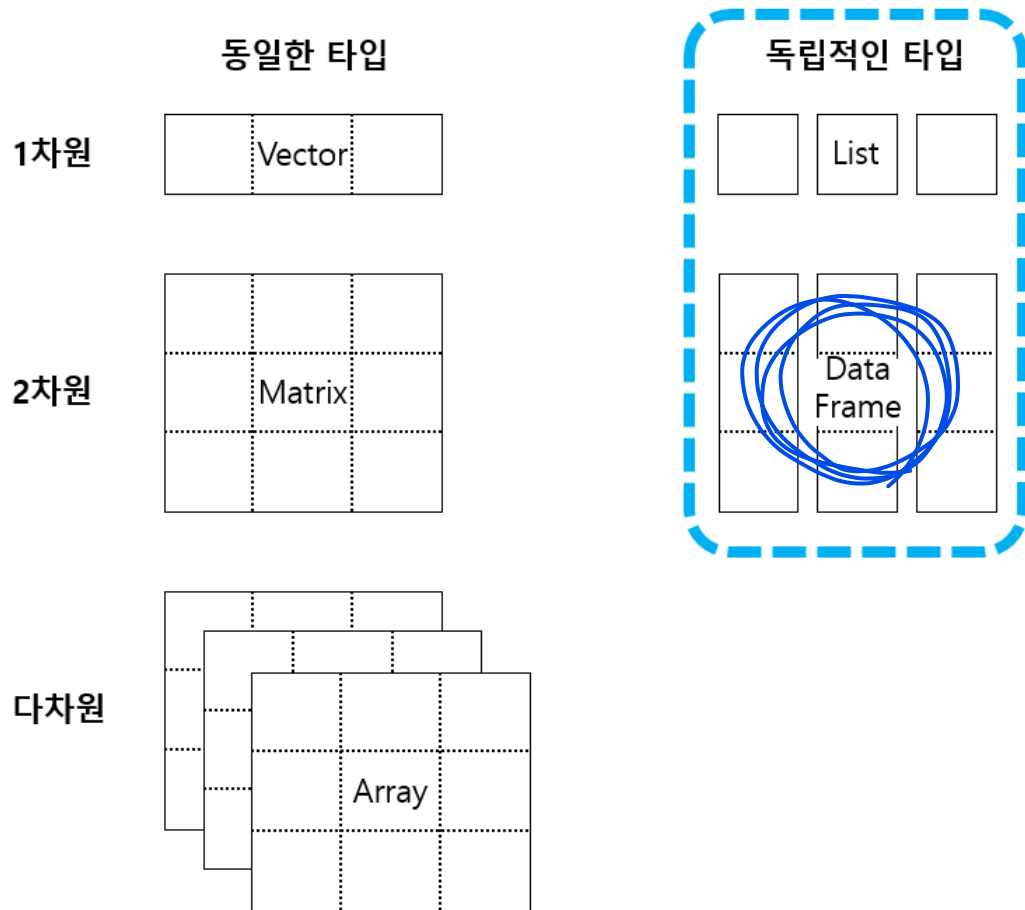
```
1 sum(d[d>5])  
# 5보다 큰 값의 합계를 출력  
2 d==5  
3 condi <- d>1 & d<5  
# 조건을 변수에 저장  
4 d[condi]  
# 조건에 맞는 값들을 선택
```



```
> sum(d[d>5])  
[1] 16  
# d      : 1 3 5 7 9  
# d>5    : F F F T T  
# d[d>5] :      7 9  
# sum(d[d>5]) : 7+9 = 16  
> d==5  
[1] FALSE FALSE TRUE FALSE FALSE  
> condi <- d>1 & d<5  
> d[condi]  
[1] 3
```

변수와 벡터

- 여러 종류의 자료 구조



변수와 벡터

- 리스트와 팩터
 - 리스트
 - 리스트 vs 벡터

	리스트	벡터
정의	서로 다른 자료형의 값들을 1차원 형태로 저장하는 수단	동일한 자료형의 값들을 1차원 형태로 저장하는 수단
생성	list() 함수로 생성	c() 함수로 생성
인덱스	리스트[[숫자]] ex) my.info[[1]] 리스트\$값의 이름 ex) my.info\$age	벡터[1] ex) d[1]

변수와 벡터

- 리스트 생성

```
> x <- list("datascience","tony", 96, TRUE)
> x
[[1]]
[1] "datascience"
[[2]]
[1] "tony"
[[3]]
[1] 96
[[4]]
[1] TRUE
```

변수와 벡터

- 리스트 생성

```
> y <- list(major="datascience",name="tony", score=96, check=TRUE)
```

```
$major
```

```
[1] "datascience"
```

```
$name
```

```
[1] "tony"
```

```
$score
```

```
[1] 96
```

```
$check
```

```
[1] TRUE
```

변수와 벡터

- 리스트 생성

```
> x <- list("datascience","tony", 96, TRUE)
```

```
> x
```

```
[[1]]
```

```
[1] "datascience"
```

```
[[2]]
```

```
[1] "tony"
```

```
[[3]]
```

```
[1] 96
```

```
[[4]]
```

```
[1] TRUE
```

```
> names(x) <- c("major", "name", "score", "check")
```

```
> x
```

#names() 함수를 이용하여 리스트 내부 자료 구조
이름 변경하기

```
$major
```

```
[1] "datascience"
```

```
$name
```

```
[1] "tony"
```

```
$score
```

```
[1] 96
```

```
$check
```

```
[1] TRUE
```


변수와 벡터

- 리스트와 팩터

- 리스트

The diagram illustrates the creation and output of an R list. The code defines a vector `ds` and a list `my.info` containing elements of different types. The output shows the list structure with element names and values. Annotations identify the data types of each element and the storage mechanism.

```
> ds <- c(90, 85, 70, 84)
> my.info <- list(name = 'Tom', age = 60, status = TRUE, score = ds)
> my.info
```

Annotations for the code above:

- `90, 85, 70, 84`: 숫자형 (Numeric)
- `'Tom'`: 문자형 (Character)
- `60`: 숫자형 (Numeric)
- `TRUE`: 논리형 (Logical)
- `ds`: 벡터 (Vector)
- Overall: -> 다양한 자료형 (Various data types)

Annotations for the output below:

- `name`, `age`, `status`, `score`: 저장될 값의 이름 (Names of values to be stored)
- Red box: 저장될 값 (Values to be stored)

Output:

```
$name
[1] "Tom"

$age
[1] 60

$status
[1] TRUE

$score
[1] 90 85 70 84
```

변수와 벡터

- 리스트와 팩터
 - 리스트

```
> ds <- c(90, 85, 70, 84)
> my.info <- list(name = 'Tom', age = 60, status = TRUE, score = ds)
> my.info[[1]]
# 리스트의 첫 번째 값을 출력
[1] "Tom"
> my.info$name
# 리스트에서 값의 이름이 name인 값을 출력
[1] "Tom"
> my.info[[4]]
# 리스트의 네 번째 값을 출력
[1] 90 85 70 8
```

변수와 벡터

- 리스트와 팩터
 - 팩터 : 문자형 데이터가 저장된 벡터의 일종
 - 성별, 혈액형 등과 같이 저장할 문자 값들이 몇 종류로 정해져 있을 때 사용
 - 문자형 벡터를 만든 뒤 'factor()' 함수를 이용하여 변환

```
> bt <- c('A', 'B', 'B', 'O', 'AB', 'A')
# 문자형 벡터 bt 정의
> bt_fac <- factor(bt)
# 팩터 bt_fac 정의
> bt
[1] "A" "B" "B" "O" "AB" "A"
> bt_fac
[1] A B B O AB A
Levels : A AB B O
#팩터 bt_fac에 저장된 혈액형의 종류
```

변수와 벡터

- 리스트와 팩터
 - 팩터

```
> bt[5]                                     # 벡터 bt의 5번째 값 출력
[1] "AB"

> bt_fac[5]                                # 팩터 bt_fac의 5번째 값 출력
[1] AB
Levels : A AB B O

> levels(bt_fac)                            # 팩터에 저장된 값의 종류를 출력
[1] "A" "AB" "B" "O"

> as.integer(bt_fac)                        # as.integer( ) : 팩터의 문자값을 숫자로 바꾸어 출력하는 함수
[1] 1 3 3 4 2 1                             levels 함수에 출력된 순서대로 숫자로 변환
                                           A(1) AB(2) B(3) O(4)
                                           A B B O AB A -> 1 3 3 4 2 1

> summary(bt_fac)                          # summary( ) : 팩터의 값을 요약해주는 함수
  A AB B O
2  1 2  1
```

← as.numeric → 실수
← as.integer → 정수

변수와 벡터

- 리스트와 팩터

- 팩터

- 팩터의 역할 : 사전에 정의된 값 외에 다른 값들은 입력하지 못하도록 함

```
> bt_fac[7] <- 'B'
# 팩터 bt_fac의 7번째에 'B'저장
> bt_fac[8] <- 'C'
# 팩터 bt_fac의 8번째에 'C'저장
> bt_fac
[1] A B B O AB A B <NA>
Levels : A AB B O
```

오류 발생

-> 'C'가 Levels에 없는 값이기 때문

(사전에 정의된 A AB B O이외의 다른 값은 입력 불가

C가 입력되지 않아 결측 값인 <NA>로 표시

예시

- 예시 1 : 100에서 200으로 구성된 벡터 d

```
d <- c(100:200)
# coding here #
```

- 실행 결과

```
> d <- c(100:200)
>  d[50]
[1] 149
>  d[seq(1, 101, 10)]
[1] 100 110 120 130 140 150 160 170 180 190 200
>  d[c(1, 11, 21, 31, 41, 51, 61, 71, 81, 91, 101)]
[1] 100 101 102 103 104 105 106 107 108 109 110 111 112
[14] 113 114 115 116 117 118 119
```

10의 배수만 출력

100에서 119까지 출력

예시

- 예시 2 : 100에서 200으로 구성된 벡터 d의 연산

```
d <- c(100:200)
# coding here #
```

- 실행 결과

```
> d <- c(100:200)
>  length(d)
[1] 101 # d의 길이
>  mean(d)
[1] 150 # d의 평균
>  sum(d < 110)
[1] 10 # 110보다 작은 값의 개수
>  sum(d[d < 110])
[1] 1045 # 110보다 작은 값의 합
```

예시

- 예시 3 : 1에서 20으로 구성된 벡터 x

```
x <- c(1:20)
# coding here #
```

- 실행 결과

```
> x <- c(1:20)
> y <- x[seq(3, 18, 3)]      # 3의 배수로 구성된 벡터 y 생성
> y
[1] 3 6 9 12 15 18
> sort(y, T)                # y 내림차순으로 정렬
[1] 18 15 12 9 6 3
> mean(x[-(2:10)])          # x에서 2~10번째 값을 제외한 값의 평균
[1] 14.18182
```


예시

- 예시 4 : 벡터 a 원소 값에 맞는 이름 지정하기

```
a <- c(1, 6, 3, 8)
# coding here #
```

- 실행 결과

```
> a <- c(1, 6, 3, 8)
> a <- sort(a)
> a
[1] 1 3 6 8
> 
> a
[1] one three six eight
      1      3      6      8
```

예시

- 예시 5 : 리스트 생성하기

```
ages <- c(58, 20, 85)
names <- c("Tony", "Ahn", "Nick")
# cording here #
```

- 실행 결과

```
> ages <- c(58, 20, 85)
> names <- c("Tony", "Ahn", "Nick")
> X ← list(ages, names)
> x

[[1]]
[1] 59 20 85

[[2]]
[1] "Tony" "Ahn" "Nick"
```

예시

- 예시 6 : 리스트 x 값 이름 변경하기

```
ages <- c(58, 20, 85)
names <- c("Tony", "Ahn", "Nick")
x <- list(ages, names)
# cording here #
```

- 실행 결과

```
> ages <- c(58, 20, 85)
> names <- c("Tony", "Ahn", "Nick")
> x <- list(ages, names)
> 
> x
$numbers
[1] 59 20 85

$names
[1] "Tony" "Ahn" "Nick"
```

예시

- 예시 7 : 리스트 z 생성하기

```
x <- c(1,6,8,11)
# coding here #
```

- 실행 결과

```
> x <- c(1,6,8,11)
> z <- list(x*2, x/2)
> 
> z
```

```
$'x*2'          # 이름이 x*2이고 리스트 값들이 x*2의 결과값
[1]  2 12 16 22
```

```
$'x/2'          # 이름이 x/2이고 리스트 값들이 x/2의 결과값
[1]  0.5 3.0 4.0 5.5
```

예시

- 예시 8 : gender_fac 팩터 값 변환하기

```
gender <- c(rep("male", 5), rep("female", 10))  
gender_fac <- factor(gender)  
# coding here #
```

- 실행 결과

```
> gender <- c(rep("male", 5), rep("female", 10))  
> gender_fac <- factor(gender)  
> gender_fac[16] <- "male"  
> summary(gender_fac)  
female male  
    10     6
```

감사합니다

kimtwan21@dongduk.ac.kr

김 태 완