R의 데이터 구조 + 조건문

Seoncheol Park

- 참고문헌: Advanced R

1 R의 데이터 구조

	동질적	비동질적
1d	벡터	리스트
2d	행렬	데이터 프레임
nd	Array	

1.1 벡터

- `c()`: atomic vector 생성

1.2 타입 체크

1. `typeof()` 함수로 체크
2. 특정 타입의 이름 앞에 `is.`를 붙인다: `is.character()`, `is.double()`, `is.integer()`, `is.logical()` 등

```
int_var <- c(1L, 6L, 10L)
typeof(int_var)
```

```
[1] "integer"
```

```
is.integer(int_var)
```

```
[1] TRUE
```

```
dbl_var <- c(1, 2.5, 4.5)
typeof(dbl_var)
```

```
[1] "double"
```

```
is.double(dbl_var)
```

```
[1] TRUE
```

1.3 여러 데이터 타입이 섞이거나 또는 변환될 경우

- character와 integer가 섞이면 character가 된다.

```
str(c("a", 1))
```

```
 chr [1:2] "a" "1"
```

- logical vector가 integer 또는 double로 변환하면, TRUE는 1, FALSE는 0이 된다. 결국 numeric처럼 쓸 수 있는 것이다.

```
x <- c(FALSE, FALSE, TRUE)
as.numeric(x)
```

```
[1] 0 0 1
```

```
sum(x)
```

```
[1] 1
```

```
mean(x)
```

```
[1] 0.3333333
```

1.4 리스트

- (개인적인 의견) R을 다른 프로그램과 가르는 핵심적인 요소가 list이다. 물론 다른 프로그래밍 언어에도 list와 비슷한 역할을 하는 요소가 많이 있지만 R의 리스트는 편리하기도 하고 다양한 기능이 있다.

```
x <- list(1:3, "a", c(TRUE, FALSE, TRUE), c(2.3, 5.9))
str(x)
```

```
List of 4
 $ : int [1:3] 1 2 3
 $ : chr "a"
 $ : logi [1:3] TRUE FALSE TRUE
 $ : num [1:2] 2.3 5.9
```

```
#리스트 원소를 부를땐 [[]]를 쓴다
x[[1]]
```

```
[1] 1 2 3
```

- 리스트 안에 또 리스트를 넣을 수 있다.

```
y <- list(list(list(list())))
str(y)
```

```
List of 1
 $ :List of 1
  ..$ :List of 1
  .. ..$ : list()
```

- c() 함수는 리스트 또한 결합할 수 있다.

```
x <- list(list(1,2), c(3,4))
y <- c(list(1,2), c(3,4))
str(x)
```

```
List of 2
 $ :List of 2
  ..$ : num 1
  ..$ : num 2
 $ : num [1:2] 3 4
```

```
str(y)
```

```
List of 4
 $ : num 1
 $ : num 2
 $ : num 3
 $ : num 4
```

- 우리가 잘 아는 lm 오브젝트 또한 list이다.

```
is.list(mtcars)
```

```
[1] TRUE
```

```
mod <- lm(mpg ~ wt, data = mtcars)
```

1.5 할당 (attributes)

모든 오브젝트들은 특정할 일을 하도록 할당이라는 것을 할 수 있다. 이것 또한 특별한 이름을 가진 list라고 볼 수 있다고 한다.

```
y <- 1:10
attr(y, "my_attribute") <- "This is a vector"
attr(y, "my_attribute")
```

```
[1] "This is a vector"
```

```
str(attributes(y))
```

```
List of 1
 $ my_attribute: chr "This is a vector"
```

1.6 데이터 프레임

- data.frame() 함수로 작성

```
df <- data.frame(x = 1:3, y = c("a", "b", "c"))
str(df)
```

```
'data.frame':   3 obs. of  2 variables:
 $ x: int  1 2 3
 $ y: chr  "a" "b" "c"
```

- 데이터 프레임에 리스트를 넣을 때

```
df <- data.frame(x = 1:3)
df$y <- list(1:2, 1:3, 1:4)
df
```

```
  x          y
1 1       1, 2
2 2    1, 2, 3
3 3 1, 2, 3, 4
```

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

```
Error in (function (..., row.names = NULL, check.rows = FALSE, check.names = TRUE, : arguments imply differing number of ro
```

2 Subsetting

2.1 논리연산과 관련된 operator들

- &: and
- |: or
- all: 모든 값들이 참인가?
- any: 적어도 한 개가 참인가?

```
#집에서 계산해보세요.
TRUE & TRUE
TRUE & FALSE
FALSE & FALSE
TRUE | TRUE
TRUE | FALSE
FALSE | FALSE
all(c(TRUE, TRUE))
all(c(FALSE, TRUE))
all(c(FALSE, FALSE))
any(c(TRUE, TRUE))
any(c(FALSE, TRUE))
any(c(FALSE, FALSE))
```

- !: 결과를 바꿀 때 쓴다.

```
#집에서 계산해보세요.
X <- TRUE
Y <- FALSE
!(X & Y)
!(X | Y)
```

2.2 which()

- 특정 조건을 만족하는 원소를 찾을 때 쓴다. 결과는 특정 조건을 만족하는 값의 위치를 반환한다.

```
x <- c(1:10)
which(x < 5)
```

```
[1] 1 2 3 4
```

- Variation으로 which.min(), which.max()가 있다. 참고로 which류들은 값을 반환하는 것이 아닌 벡터에서 특정 조건을 만족하는 값의 위치를 반환한다는 것에 주의하자.

```
x <- c(1:10, 5:15, -3:-1)
which.min(x)
```

```
[1] 22
```

```
which.max(x)
```

```
[1] 21
```

2.3 rev(), sort(), order() 등

- rev(): 벡터의 순서를 뒤집을 때 쓴다.

```
rev(c(1:10))
```

```
 [1] 10  9  8  7  6  5  4  3  2  1
```

- sort(): 정렬할 때 쓴다.

```
sort(x)
```

```
 [1] -3 -2 -1  1  2  3  4  5  5  6  6  7  7  8  8  9  9 10 10 11 12 13 14 15
```

```
sort(x, decreasing = TRUE)
```

```
 [1] 15 14 13 12 11 10 10  9  9  8  8  7  7  6  6  5  5  4  3  2  1 -1 -2 -3
```

- order(): 가장 작은 수부터 위치를 말해주는 함수

```
z <- c(3,4,1,5,9,10,2,7,6,8)
order(z)
```

```
 [1]  3  7  1  2  4  9  8 10  5  6
```

- union(): 합집합

```
z1 <- c(3,4,1,5,9)
z2 <- c(10,2,7,6,8)
union(z1, z2)
```

```
 [1]  3  4  1  5  9 10  2  7  6  8
```

- intersect(): 교집합

```
z1 <- c(3,4,1,5,9,11)
z2 <- c(10,2,7,6,8,11)
intersect(z1, z2)
```

```
[1] 11
```

- setdiff(): 차집합

```
setdiff(z1, z2)
```

```
[1] 3 4 1 5 9
```

2.4 글자와 관련된 matching들

- %in% 함수: 특정 문자열이 있는지 체크해주고 TRUE, FALSE를 반환

```
sstr <- c("c","ab","B","bba","c",NA,"@","bla","a","Ba","%")
sstr[sstr %in% c(letters, LETTERS)]
```

```
[1] "c" "B" "c" "a"
```

- subset: character의 부분집합을 잡아준다.

```
substr("abcdef", 2, 4)
```

```
[1] "bcd"
```

- strsplit: 자료를 처리할 때, 때때로 특정 문자를 기준으로 character를 갈라놓아야 할 때가 있는데 그럴 때 쓴다.

```
name_vec <- c("Seoul-si", "Seongdong-gu", "Haengdang-dong")
strsplit(name_vec, "-")
```

```
[[1]]
[1] "Seoul" "si"

[[2]]
[1] "Seongdong" "gu"

[[3]]
[1] "Haengdang" "dong"
```

```
unlist(strsplit(name_vec, "-")) #unlist: 리스트 풀때
```

```
[1] "Seoul"     "si"        "Seongdong" "gu"        "Haengdang" "dong"
```

- subset: 말 그대로 부분집합이며 특정 조건을 만족하는 집합을 찾는 것이다.

```
head(subset(airquality, Temp > 80, select = c(Ozone, Temp)))
```

```
   Ozone Temp
29    45   81
35    NA   84
36    NA   85
38    29   82
39    NA   87
40    71   90
```

3 조건문과 apply류 함수들

3.1 for문

- for문은 반복연산시 사용한다.

```
y <- 0
for(i in 1:10){
  y <- y+i
  cat("summation from 0 to ", i, " is ", y, "\n", sep="")
}
```

```
summation from 0 to 1 is 1
summation from 0 to 2 is 3
summation from 0 to 3 is 6
summation from 0 to 4 is 10
summation from 0 to 5 is 15
summation from 0 to 6 is 21
summation from 0 to 7 is 28
summation from 0 to 8 is 36
summation from 0 to 9 is 45
summation from 0 to 10 is 55
```

3.2 if문

- if문은 else if, else 등과 같이 써서 보통 분기 조건을 나타낼 때 쓴다.

```
x <- 0
if (x < 0) {
  print("Negative number")
} else if (x > 0) {
  print("Positive number")
} else {
  print("Zero")
}
```

```
[1] "Zero"
```

3.3 while문

- while문은 해당 조건을 만족할 때까지 계속 계산을 하기 때문에 만약 만족 불가능 (아니면 거의 힘든) 조건을 넣었을 경우 무한히 계산만 하게 된다. 그럴 경우를 방지하기 위해 중간에 break를 넣어주는 게 좋다.

```
j <- 1
while(TRUE){
  j <- j+3
  if(j > 5){
    break
  }
}
j
```

```
[1] 7
```

3.4 apply 함수와 그 친구들

- 교재에 나와있듯이, R에서는 일반적으로 for 문을 쓰는 것보다 apply 류의 함수를 쓰는 것이 계산 속도가 조금은 빠름이 알려져 있다.
- apply: array나 matrix와 같이 쓴다.

```
A <- matrix(c(1:16), nrow=4, ncol=4)
apply(A, 1, sum)
```

```
[1] 28 32 36 40
```

```
apply(A, 2, sum)
```

```
[1] 10 26 42 58
```

- lapply: 리스트에 쓰는 apply 함수라고 생각하면 편하다. 반환도 리스트로 한다.

```
x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE,FALSE,FALSE,TRUE))
lapply(x, mean)
```

```
$a
[1] 5.5

$beta
[1] 4.535125
```

```
$logic
[1] 0.5
```

- sapply: lapply와 거의 같은 역할을 하나 결과가 벡터나 matrix 꼴로 나온다는 점에서 사용자가 사용하기 편하다.

```
region_vec <- c("Chungcheongbuk-do", "Chungcheongnam-do", "Seoul-si")
sapply(region_vec, function(x) substr(x, start=1, stop=11)=="Chungcheong")
```

```
Chungcheongbuk-do Chungcheongnam-do          Seoul-si
             TRUE                TRUE             FALSE
```