3장
R프로그래밍

조건문

- 문법

  if(cond) { 문장}

  else {문장}

```
if (TRUE) {
  print('TRUE')
  print('hello')
} else {
  print('FALSE')
  print('world')
}
```

ifelse(test, yes, no)

```
> x <- c(1,2,3,4,5)
> ifelse( x %% 2 == 0, "even", "odd")
[1] "odd"  "even" "odd"  "even" "odd"
```

반복문

```
> for (i in 1:10) {
+   print(i)
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
 ...
```

```
> i <-1
> repeat {
+     print(i)
+     if(i >= 10) {
+         break
+     }
+     i <- i + 1
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
 ...
```

```
> i <- 0
> while (i < 10) {
+   print(i)
+   i <- i + 1
+ }
[1] 0
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
 ...
```

연산

- 수치연산

연산자와 함수	의미
+, −, *, /	사칙 연산
n %% m	n을 m으로 나눈 나머지
n %/% m	n을 m으로 나눈 몫
n^m	n의 m승
exp(n)	e의 n승
log(x, base=exp(1))	$\log_{base}(x)$. 만약 base가 지정되지 않으면 $\log_e(x)$를 계산
log2(x), log10(x)	각각 $\log_2(x)$, $\log_{10}(x)$를 계산
sin(x), cos(x), tan(x)	삼각 함수

벡터 연산

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

```
> x <- c(1, 2, 3, 4, 5)
> sum(x)
[1] 15
> mean(x)
[1] 3
> median(x)
[1] 3
```

```
> x <- c(1, 2, 3, 4, 5)
> x + x
[1]  2  4  6  8 10
> x == x
[1] TRUE TRUE TRUE TRUE TRUE
> x == c(1, 2, 3, 5, 5)
[1]  TRUE  TRUE  TRUE FALSE  TRUE
> c(T, T, T) & c(T, F, T)
[1]  TRUE FALSE  TRUE
```

```
> x <- c(1, 2, 3, 4, 5)
> ifelse(x %% 2 == 0, "even", "odd")
[1] "odd"  "even" "odd"  "even" "odd"
```

벡터 연산

```
> (d <- data.frame(x=c(1, 2, 3, 4, 5), y=c("a", "b", "c", "d", "e")))
  x y
1 1 a
2 2 b
3 3 c
4 4 d
5 5 e
> d[c(TRUE, FALSE, TRUE, FALSE, TRUE), ]
  x y
1 1 a
3 3 c
5 5 e
```

```
> d[d$x %% 2 == 0, ]
  x y
2 2 b
4 4 d
```

결측치(NA)의 처리

```
> NA & TRUE
[1] NA
> NA + 1
[1] NA
```

```
> sum(c(1, 2, 3, NA))
[1] NA
> sum(c(1, 2, 3, NA), na.rm=T)
[1] 6
```

```
> x <- data.frame(a=c(1, 2, 3), b=c("a", NA, "c"), c=c("a", "b", NA))
> x
  a    b    c
1 1    a    A
2 2 <NA>    B
3 3    c <NA>
> na.omit(x)
  a b c
1 1 a A
```

함수

- 정의
  - 함수명 <- function(인자, 인자, ...) { 함수 본문 }

```
> fibo <- function(n) {
+   if (n == 1 || n == 2) {
+     return(1)
+   }
+   return(fibo(n - 1) + fibo(n - 2))
+ }
> fibo(1)
[1] 1
> fibo(5)
[1] 5
```

함수

- 반환 방법
  - return(반환값) # ()생략 불가능
  - return()이 생략된다면 함수 내 마지막 문장의 결과가 반환값이 됨

```
> fibo <- function(n) {
+   if (n == 1 || n == 2) {
+     1
+   } else {
+     fibo(n - 1) + fibo(n - 2)
+   }
+ }
```

함수

- 인자 지정

```
> f <- function(x, y) {
+   print(x)
+   print(y)
+ }
> f(1, 2)
[1] 1
[1] 2
> f(y=1, x=2)
[1] 2
[1] 1
```

- 가변길이 인자

```
> f <- function(x, y) {
+   print(x)
+   print(y)
+ }
> g <- function(z, ...) {
+   print(z)
+   f(...)
+ }
> g(1, 2, 3)
[1] 1
[1] 2
[1] 3
```

변수 scope

- 콘솔에서 변수를 선언하면 모든 곳에서 사용 가능

```
> n <- 1
> f <- function() {
+   print(n)
+ }
> f()
[1] 1
> n <- 2
> f()
[1] 2
```

- 함수내부에서 선언
  - 함수 내에서만 사용가능

서로 다른 변수

```
> n <- 100
> f <- function() {
+   n <- 1
+   print(n)
+ }
> f()
[1] 1
```

변수 scope

- rm(list=ls())는 모든 객체를 삭제하는 명령

```
> n <- 100
> f <- function() {
+   n <- 1
+   print(n)
+ }
> f()
[1] 1
```

```
> rm(list=ls())
> f <- function() {
+   print(x)
+ }
> f()
Error in print(x) : object 'x' not found
```

Call by value

```
> f <- function(df2) {
+   df2$a <- c(1, 2, 3)
+ }
>
> df <- data.frame(a=c(4, 5, 6))
> f(df)
> df
  a
1 4
2 5
3 6
```

```
> f <- function(df) {
+   df$a <- c(1, 2, 3)
+   return(df)
+ }
>
> df <- data.frame(a=c(4, 5, 6))
> df <- f(df)
> df
  a
1 1
2 2
3 3
```

객체의 삭제

- ls()
  - 메모리 상에 만들어진 객체의 목록

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

```
> rm("x")
> ls()
character(0)
```

객체 x를 삭제

```
> rm(list=ls())
```

모든 객체를 삭제

예) Queue의 구현

```
> q <- c()

> q_size <- 0

> enqueue <- function(data) {
+   q <<- c(q, data)
+   q_size <<- q_size + 1
+ }

> dequeue <- function() {
+   first <- q[1]
+   q <<- q[-1]
+   q_size <<- q_size - 1
+   return(first)
+ }

> size <- function() {
+   return(q_size)
+ }
```

함수내부에서
외부변수 수정은 <<-

```
> enqueue(1)
> enqueue(3)
> enqueue(5)
> print(size())
[1] 3
> print(dequeue())
[1] 1
> print(dequeue())
[1] 3
> print(dequeue())
[1] 5
> print(size())
[1] 0
```