# 제9주 : 흐름제어

## 9.1. 조건문

주어진 조건에 따라 조건이 참(TRUE)일 경우와 거짓(FALSE)일 경우 선택적으로 문장을 실행시킬 수 있다.

### (1) 조건문 함수 ifelse()

```
ifelse(test, yes, no)
```

- test: 조건 판별식
- yes: test가 TRUE일 경우 반환할 값
- no: test가 FALSE일 경우 반환할 값

실행 예:

```
> x <- c(6:-4)
> options(digits = 3)
```

➔출력물을 세자리로 설정

```
> sqrt(x)
[1] 2.45 2.24 2.00 1.73 1.41 1.00 0.00  NaN  NaN  NaN  NaN
```

경고 메시지가 손실되었습니다

```
In sqrt(x) : 계산 결과가 NaN가 생성되었습니다
```

➔음수에 제곱근을 구할 경우 NaN가 발생하여 경고 발생

```
> options(digits = 3)
> sqrt(ifelse(x >= 0, x, NA))
[1] 2.45 2.24 2.00 1.73 1.41 1.00 0.00   NA   NA   NA   NA
```

➔ifelse()를 이용하여 0 이상일 경우에 x값을 반환하고 그렇지 않을 경우 NA 반환

### (2) 조건문 if

```
if (cond) expr

if (cond) cons.expr else alt.expr
```

- cond: 논리형 값 (TRUE/FALSE)
- expr, cons.expr: cond가 TRUE일 경우 실행할 문장
- alt.expr: cond가 FALSE일 경우 실행할 문장

실행 예:

```
> x <- c(1, 2, 3)
> x <- factor(x)
```
➔숫자로 구성된 벡터 x를 팩터로 재구성

```
> if(is.factor(x)) length(x)
[1] 3
```
➔is.factor(x) 는 x가 팩터일 경우 TRUE를 그렇지 않으면 FALSE 반환.
➔x가 팩터이므로 TRUE
➔if 문에서 TRUE일 경우 length(x)를 실행한다. 이 경우 TRUE 이므로 length(x) 실행, FALSE 일 경우 실행하지 않고 다음으로 이동
➔length(x)의 실행값인 3 출력

```
> if(is.factor(x)) {
+   length(x)
+ } else {
+   sum(x)
+ }
[1] 3
```
➔if문에서 조건을 판단해서 TRUE일 경우와 FALSE일 경우 각각 나눠서 처리할 경우 else를 같이 사용
➔중괄호({)는 여러개의 코드를 하나의 코드블럭으로 묶는 것으로 하나의 처리 단위로 인식. 중괄호 내의 코드는 순차적으로 실행
➔else를 사용하기 위해 위와 같이 중괄호로 묶어서 사용
➔else 이후는 주어진 조건이 거짓일 경우 실행, 이 예에서는 조건이 참이므로 length(x)의 결과 출력

```
> if(is.factor(x)) {
+   length(x)
+ } else if(is.integer(x)){
```

```
+   sum(x)
+ } else {
+   paste(x, "element")
+ }
```

➔else 이후 다시 등장한 if는 앞선 if 조건이 FALSE가 되는 나머지 상황에서 또다시 조건을 판단할 경우 사용하는 것으로 이 경우 앞선 if가 판별한 것은 x가 팩터인지 여부에서 팩터가 아닐 경우 즉, 팩터가 아닌 다양한 상황일 경우 중에서 다시 조건 판단을 하겠다는 의미

➔if문은 이와같이 else 이후에 중첩해서 또 쓸수 있음.

➔이 문장은 처음 판단하는 것은 주어진 x가 팩터인 경우이고, 주어진 x가 팩터가 아닐 경우, 정수형인지 판단하고 정수형이면 개수를 그렇지 않다면, 즉, 주어진 x가 팩터가 아니고 나머지 상황 중 정수가 아닌 모든 상황에 대해 paste() 함수를 실행

## 9.2. 반복문

반복문은 주어진 문장을 반복하면서 수행하는 문장으로, 구구단과 같이 *일정한 패턴으로 반복되는 문장을 작성할 때* 편이를 제공합니다. 현대의 수치해석학에서는 컴퓨터를 이용하여 근사값을 찾는 경우가 많으며 이 중 반복처리는 필수 요소입니다. 반복문을 잘 사용하면 전체 코드의 길이를 획기적으로 줄일 수 있으며 아주 쉽게 계산할 수 있지만 반복문을 잘못 사용할 경우 “무한루프”에 빠져 쓸데없는 자원의 낭비는 물론 심한 경우 컴퓨터 자체를 먹통으로 만들 수 있으니 처음 사용하실 때 많은 주의가 필요하며, 반복문은 무한루프에 빠지지 않게 하기 위해 종료 조건을 직접 주거나 판단하여 처리하는 경우가 있으므로 잘 살펴봐 주시기 바랍니다.

### (1) 무조건 반복하고 본다: repeat

```
repeat expr

break

next
```

- expr: 반복 실행할 문장

반복이라는 기능에 가장 충실한 repeat 문입니다. repeat은 단순히 repeat 뒤에 나오는 코드 혹은 코드블록을 반복하므로 코드 자체의 요소가 어느 일정한 시점에 멈출 수 있는 기능이 있거나 판단할 부분을 넣어 주어야 합니다.

- 반복문 실행 중간에 break를 사용하면 반복문을 벗어난다.
- 반복문 실행 중간에 next를 사용하면 실행을 중지하고 반복문의 시작위치로 돌아가 반복문을 실행한다.

실행 예:

```
> i <- 20
> repeat {
    ➔중괄호로 둘러싸인 부분 반복
+   if(i > 25) {
       ➔ 반복문을 중지하기 위한 조건:
       ➔ i가 25보다 크면 아래의 break 문을 통해 반복 중지
+      break
       ➔ break 문: 현재의 반복문 정지
+   } else {
       ➔ 반복을 중지하지 않을 경우, 즉 반복되는 부분
+      print(i)
+      i <- i + 1
       ➔ i를 1씩 증가시켜 주는 코드
       ➔ 생략시 i는 계속해서 앞서 지정한 값 20만 사용되고 무한반복에 빠짐
+   }
+ }
[1] 20
[1] 21
[1] 22
[1] 23
[1] 24
```

```
[1] 25
➔ 이 코드는 결과적으로 20부터 25까지 1씩 증가하면서 값을 출력
```

### (2) 주어진 조건이 참일 때만 반복 실행한다: while

```
while (cond) expr
```

- expr: 반복 실행할 문장

while 문은 다른 프로그래밍 언어에서도 흔히 볼 수 있는 반복문입니다. while은 조건을 주어 해당 조건이 참일 경우에만 코드를 반복하는 문장으로 repeat과 마찬가지로 조건 자체가 FALSE를 가질 수 있도록 하여 어느 일정 시점에서 멈춰줘야 합니다.

사용 예:

```
> dan <- 2
> i <- 2
> while (i < 10) {
    ➔ i가 10보다 작을 때까지 코드 반복 수행
+   times <- i * dan
    ➔ 변수 times에 현재의 i값과 dan의 값인 2를 곱한 값 저장
+   print(paste(dan, "X", i, " = ", times))
    ➔ 출력 결과물은 단수 곱하기(“X”) i의 현재 값 “=” 곱한 값의 형태
+   i <- i + 1
    ➔ i 값 1 증가
+ }
[1] "2 X 2  =  4"
[1] "2 X 3  =  6"
[1] "2 X 4  =  8"
[1] "2 X 5  =10"
[1] "2 X 6  =  12"
[1] "2 X 7  =  14"
[1] "2 X 8  =  16"
[1] "2 X 9  =  18"
```

➔ 출력 결과는 구구단의 2단

### (3) 전통의 반복문! 하지만 R에선 조금 다르다: for

for (*var* in *seq*) *expr*

- seq: 벡터 또는 리스트
- var: 변수이름
- expr: 반복 실행할 문장

seq의 각 원소들을 순차적으로 객체 var에 할당하고 문장 expr을 수행하는 일을 반복합니다. 따라서, 총 seq의 길이만큼 반복 수행합니다.

```
> dan <-9
> for( i in 2:9) {
    ➔반복은 벡터 2:9 이며 매 반복 시 2부터 9 사이의 값을
       가져와 i에 저장
    ➔첫 번째 반복에서 i에 벡터(2, 3, 4, ……, 9) 중 첫 번째
       원소인 2 저장
+   times <-i * dan
+   print(paste(dan, "X", i, " = ", times))
+ }
➔ 반복 횟수는 벡터(2, 3, 4, ……, 9)의 원소의 개수인 8회
➔ 반복이 종료되는 시점은 벡터의 마지막 원소까지 가져와 반복하고 종료
[1] "9 X 2  =  18"
[1] "9 X 3  =  27"
[1] "9 X 4  =  36"
[1] "9 X 5  =  45"
[1] "9 X 6  =  54"
[1] "9 X 7  =  63"
[1] "9 X 8  =  72"
[1] "9 X 9  =  81"
> str <-c("a", "b", "c")
```

```
> for (i in str) {
```
➔ i는 벡터 str을 순회하면서 각 원소를 가져옴

➔처음 i에 저장되는 값은 벡터 str의 첫 번째 값인“a”

```
+   print(i)
+ }
[1] "a"
[1] "b"
[1] "c"
```
➔결과적으로 벡터 str에 있는 모든 원소 출력