R 프로그래밍 기초다지기

7강 - 사용자 정의 함수와 루프(Loop)

슬기로운통계생활

Issac Lee

사용자 함수(Function)와 루프 (Loops)

R 프로그래밍의 구조

블록 구조

- R은 C, C++, Python, Perl 과 같은 블록 구조 (block-structured) 프로그래밍 언어 중 하나.
- `{}`를 사용하여 블록을 구분
- 블록 안에 여러개의 구문들 (statements) 이 존재
- 구문들은 줄 바꿈 혹은 세미 콜론 `;` 으로 구분

```
a <- 3
b <- 2
a; b # 2개 statements
```

```
## [1] 3
```

```
## [1] 2
```

사용자 함수 정의

선언 방법

- `function()` 함수를 사용하여 정의
- 함수의 입력값 (argument)과 내용 (body) 을 정의해줘야 함.
- 구문

```
function_name <- function(input) {
   # body part
   result <- input + 1
   return(result)
}
```

R의 모든 것은 `Object`

함수 역시 객체 (Object)

```
g <- function(x) {
  return(x + 1)
}
class(g)
```

```
## [1] "function"
```

- 입력값을 넣으면 결과값을 뱉어내는 객체

```
g(3)
```

```
## [1] 4
```

- 함수 `g`의 입력값과 내용 접근

```
formals(g)
```

```
## $x
```

```
body(g)
```

```
## {
##     return(x + 1)
## }
```

함수의 내용 출력 (Printing)

정의된 함수 내용확인

- 그냥 함수 이름만 치고 돌리면 됨.

```
g
```

```
## function(x) {
##   return(x + 1)
## }
```

- `abline` 함수는 어떻게 생겼을까?

- R의 함수들 중에서 fundarmental 한 함수의 경우는 안보임
- 보통 R 기반 함수 (`.Primitive`) 들 C로 짜여짐

```
sum
```

```
## function (..., na.rm = FALSE)
```

함수 기본 입력값 설정

입력값 설정에 따른 함수 결과

- 기본 입력값 정의가 안된 경우

```
g <- function(x) {
  result <- x + 1
  return(result)
}
g()
```

```
## Error in g(): 기본값이 없는 인수
```

- 기본 입력값 설정 후

```
g <- function(x = 3) {
  result <- x + 1
  return(result)
}
g()
```

```
## [1] 4
```

결과값 반환 함수 `return()`

R의 스타일 가이드

- R의 함수는 자동으로 마지막 구문을 결과값으로 반환하게 되어 있음.
- `tidyverse` 스타일 가이드에 따르면 `return()` 함수의 경우 early return의 경우에만 사용을 권장함.

```
# Good
find_abs <- function(x) {
  if (x > 0) {
    return(x)
  }
  -1 * x
}

# Bad
add_two <- function(x, y) {
  return(x + y)
}
```

함수 이름 정할 때 유의사항

snake case

- 변수와 함수 이름은 무조건 영어 소문자, 숫자, 그리고 밑줄 `_` 만을 사용
- 함수 이름에 `.` 사용하는 것은 추후 배울 OOP를 사용해서 코딩을 할 때 혼란을 초래할 수 있음.
- 함수 이름의 경우 동사를 먼저 사용
- 변수 이름은 명사를 사용

- 좋은 함수 이름 예시

```
# Good
add_two()

# Bad
AddTwo()
number_adder()
```

유용한 조건문 `if` and `ifelse`

꼭 알아둘 것

- `if ... else ...` 구문

```
if (condition) {
  statement 1
} else {
  statement 2
}
```

- `ifelse()` 함수

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

- 예제

```
x <- 3; y <- 0
if (x > 4) {
    y <- 1
} else {
    y <- 2
}
y
```

```
## [1] 2
```

조건 3개 이상의 경우 `switch()`

알아두기만 하고 찾아볼 것

```
x <- 1; y <- 2; input <- "good"
switch(
    input,
    "good"= cat("score =", x + y),
    "normal"= cat("score =", 2 * x),
    "bad"= cat("score = ", -y)
)
```

```
## score = 3
```

반복되는 작업을 쉽게 Loops

꼭 알아야하는 루프 2개

- `for` 문

```
x <- 1:3
for (i in x) {
  print(paste("Here is", i))
}
```

```
## [1] "Here is 1"
## [1] "Here is 2"
## [1] "Here is 3"
```

- 루프에 목 매지 마세요.

- `while` & `break` 문

```
i <- 1
while (TRUE) {
  i <- i + 3
  if (i > 10) break
  print(i)
}
```

```
## [1] 4
## [1] 7
## [1] 10
```

`apply()` 함수 완벽 정복

행렬과 배열을 사용한 루프

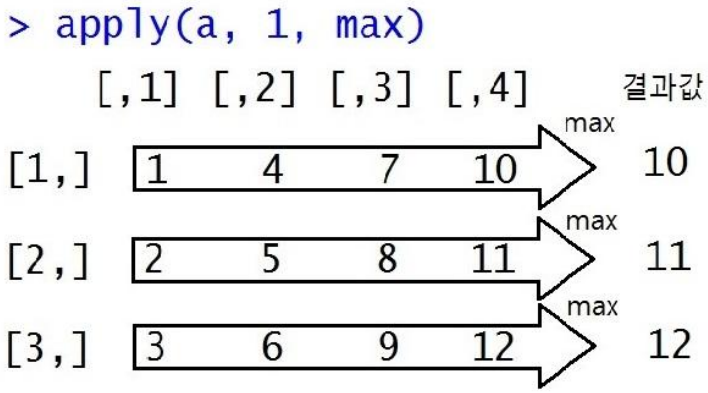
- 구문: `apply(object, direction, function)`
- `direction`: 함수 적용방향
  - 1: 행 (row) 별 입력
  - 2: 열 (column) 별 입력

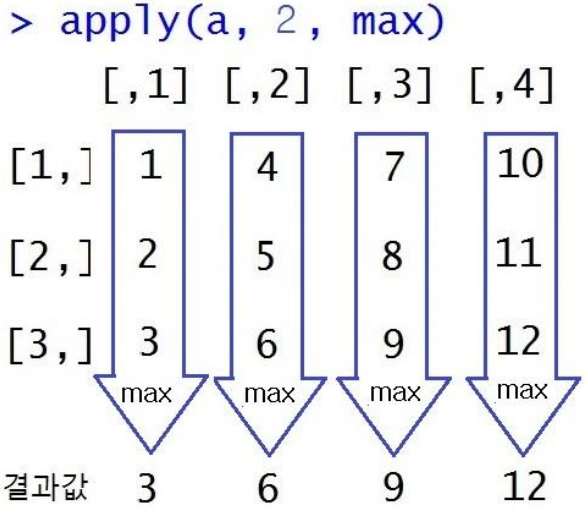
```
a <- matrix(1:12,
                nrow = 3,
                ncol = 4)
a
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

함수 적용

- 함수 적용방향에 따른 결과 값 이해






```
## [1] 10 11 12
```

```
## [1]  3  6  9 12
```

사용자 함수를 사용한 apply()

사용자 함수

- 맨 마지막 인풋을 사용자 정의 함수로 설정이 가능함.

```
my_f <- function(vec){
    max(vec)^2 + 3
}
apply(a, 1, my_f)
```

```
## [1] 103 124 147
```

3차원 배열에 `apply()` 함수 적용

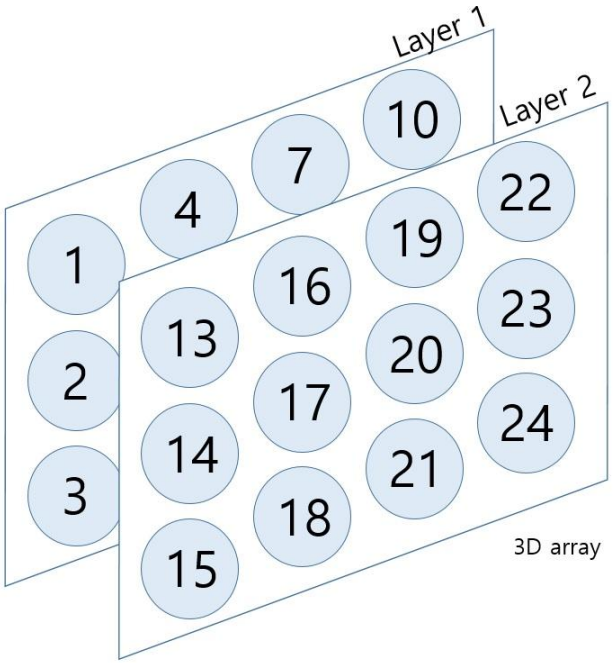
- `apply()`는 배열을 입력값으로 받을 수 있음
- 3차원 배열의 경우 함수 적용 방향이 3개

```
array_3d <- array(1:24,
                  dim = c(3,4,2))
array_3d
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,]   13   16   19   22
## [2,]   14   17   20   23
## [3,]   15   18   21   24
```

3차원 배열

우리가 만든 3차원 배열



- 방향 1은 어떻게 적용이 된 걸까?

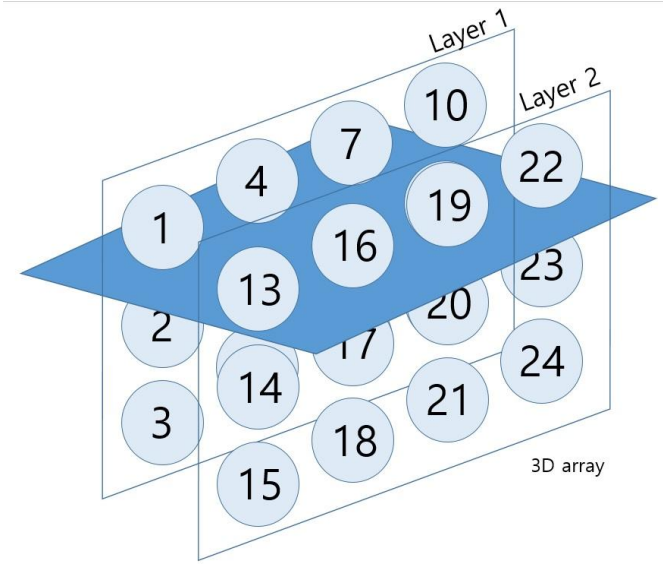
```
apply(array_3d, 1, max)
```

```
## [1] 22 23 24
```

3차원 배열의 3가지 방향

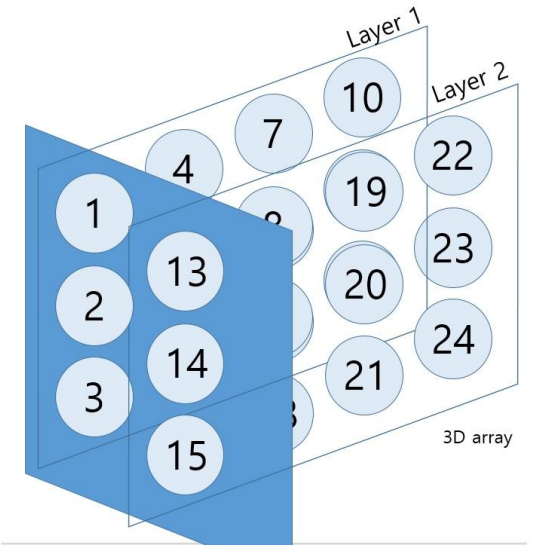
- 방향 1

`apply(array_3d, 1, max)`



- 방향 2

`apply(array_3d, 2, max)`



Q. 마지막 방향 `apply(array_3d, 3, max)`의 결과는?

함수와 환경

함수에는 대응하는 환경이 따로 존재

- **R** 함수의 구성요소
  - 입력값 (arguments)
  - 내용 (body)
  - **환경 (environment)**
- 함수가 만들어 질 때 존재하는 객체들 모음
- Global 환경에 존재하는 변수 `y`와 `my_f()` 안에 존재하는 `y`는 **다른 환경**에 같은 이름을 가진 변수

```
y <- 2
my_f <- function(x) {
  y <- 1
  result <- x + y
  result
}
y
```

```
## [1] 2
```

R 환경의 계층 구조

함수 안에 함수 있다.

- top-level인 `Global Env.` 에 정의된 `outer_f()`

```
outer_f <- function() {
  inner_f <- function(input) {
    input + 2
  }

  # check env of inner_f()
  print(environment(inner_f))
}
```

```
environment(outer_f)
```

```
## <environment: R_GlobalEnv>
```

- `inner_f()`의 환경은 `outer_f()`에서 접근가능

```
outer_f()
```

```
## <environment: 0x000000001316
```

`<<-` 연산자와 환경구조

상위 환경과의 교류

- `my_f()`에 묶여있는 환경이 아닌 상위 환경에 접근 가능

```
y <- 2
my_f <- function() {
  y <<- 1
}
```

```
y
```

```
## [1] 2
```

```
my_f()
y
```

```
## [1] 1
```

다음시간

문자열(String) 다루기와 시각화

참고자료 및 사용교재

[1] The art of R programming

- R 공부하시는 분이면 꼭 한번 보셔야 하는 책입니다.
- 위 교재의 한글 번역본 빅데이터 분석 도구 R 프로그래밍도 있습니다. 도서 제목 클릭하셔서 구매하시면 저의 사리사욕을 충당하는데 도움이 됩니다.