통계학 실습

9. 프로그래밍 구조

R 프로그래밍 구조

- R에도 다른 프로그래밍 언어에서 사용되는 다양한 프로그래밍 문법이 사용된다.
  - 함수, 조건문, 반복문, 산술, 부울, 연산, 반환, ...

함수의 사용 (조건문과 출력문)

```
> functionName <- function(a, b) {
+   num <- a + b
+   if (num > 5) {
+     print("Bigger than 5")
+   }
+   else {
+     print("Smaller than 5")
+   }
+ }
> functionName(4, 7)
[1] "Bigger than 5"
> functionName(1, 3)
[1] "Smaller than 5"
>
```

- 함수의 선언
  - 함수이름 <– function(파라미터) { }
- 조건문 사용
  - 다른 프로그래밍 언어의 if / else 문법과 동일하게 사용한다.
- 출력문
  - print() 함수를 이용하여 바로 출력되도록 할 수 있다.

반복문 (while, for)

```
> cumulative <- function(n) {
+   sum <- 0
+   start <- 1
+   while(start < n+1) {
+     sum <- sum + start
+     start <- start + 1
+   }
+   print(sum)
+
+   sum <- 0
+   start <- 1
+   for(i in start:n) {
+     sum <- sum + i
+   }
+   print(sum)
+ }
> cumulative(5)
[1] 15
[1] 15
> cumulative(10)
[1] 55
[1] 55
>
```

- ‘cumulative’ 함수는 1부터 주어진 숫자까지의 누적합을 구하는 함수이다.

- while 반복문
  - while(조건문)을 이용하여 반복한다.
  - 조건이 만족되면 while문 안이 실행된다.

- for 반복문
  - for(변수 in 범위)을 이용하여 반복한다.
  - 범위에 해당되는 수가 1씩 증가하면서 앞의 변수에 반복에 따라 대입된다.
    ex) i in 1:5 –〉 i = 1, 2, 3, 4, 5

산술 및 부울 연산

산술 연산자	설명	부울 연산자	설명
x + y	덧셈	x == y	일치 여부
x - y	뺄셈	x >= y	크기 비교 여부
x * y	곱셈	x && y	부울 AND
x / y	나눗셈	x || y	부울 OR
x ^ y	제곱	x & y	부울 AND (벡터)
x %% y	나머지 (%가 2개)	x | y	부울 OR (벡터)
x %/% y	정수형 결과 나눗셈	!x	부울 부정

부울 사용

```
> x <- c(TRUE, FALSE, TRUE)
> y <- c(TRUE, FALSE, FALSE)
> x & y
[1]  TRUE FALSE FALSE
> x && y
[1] TRUE
> x[1] && y[1]
[1] TRUE
> x * 3
[1] 3 0 3
> x <- c(TRUE, FALSE, TRUE)
> y <- c(TRUE, FALSE, FALSE)
> x & y
[1]  TRUE FALSE FALSE
> x && y
[1] TRUE
> x[1] && y[1]
[1] TRUE
> x * 3
[1] 3 0 3
> (x * 3)[1] == 3
[1] TRUE
>
```

- & 연산은 벡터의 각 요소마다 모두 실행되어 결과를 나타낸다.
- && 연산은 벡터에서 첫 번째 인덱스만 실행된다.

- 부울에서 TRUE는 1로, FALSE는 0으로 숫자로 연결된다.
  - 연산에 직접적으로 사용할 수 있으며, 결과 또한 벡터로 나오므로 바로 접근 가능

반환과 전역변수

```
> len1 <- 0
> len2 <<- 0
> max2nd <- function(nums) {
+   maxNum <- max(nums)
+   array <- c()
+   for(i in nums) {
+     if (maxNum - i != 0) {
+       array <- c(array, maxNum - i)
+     }
+   }
+   len1 <- length(array)
+   len2 <<- length(array)
+   return(maxNum - min(array))
+ }
> nums <- c(3, -4, 1, -6, 3, 11, 7, 8, 9, 12)
> max2nd(nums)
[1] 11
> len1
[1] 0
> len2
[1] 9
>
```

- 반환은 return()함수를 사용한다.
  - 반환이 없는 경우 따로 출력문이 없는 경우 결과가 존재하지 않는다. 여기서 특별한 점은 R에서는 변수만 적힌 라인 자체가 결과를 돌려주는 문구이므로, return() 없이 변수명만 적어도 반환이 된다.
  - 반환이 있는 경우에 앞에 저장할 변수가 있다면 해당 변수로 저장되고, 변수가 저장되지 않으면, 변수를 그대로 입력하는 것과 같은 역할을 하여 바로 출력된다.

- 전역변수는 〈〈– 기호를 이용한다.
  - 전역변수는 함수 내에서 수정하여도 수정된 내용이 반영된다.
    ex) len1과 len2의 비교

함수를 이용하여 예시 자료구조 쉽게 만들기

```
> randomScore <- function(n) {
+   score <- c()
+   for (i in 1:n) {
+     score <- c(score, sample(100)[1])
+   }
+   score
+ }
>
> determine <- function(Scores) {
+   ss <- c()
+   for(s in Scores) {
+     if (s >= 60)
+       ss <- c(ss, "P")
+     else {
+       ss <- c(ss, "F")
+     }
+   }
+   ss
+ }
>
> No <- seq(1:30)
> Score <- randomScore(30)
> Pass <- determine(Score)
>
> classData <- data.frame(No, Score, Pass)
```

- randomScore() 함수는 무작위로 점수를 뽑아 벡터를 만들어준다.
- determine() 함수는 주어진 점수가 60점 이상일 경우 P를 내어준다.
- 샘플로 새로운 데이터셋을 만들 때, 좌측과 같이 자신이 만든 함수를 이용하여 쉽게 예시를 만들어낼 수 있다.

실습

```
> escape()
Choose random number (1~10) : 3
[1] "Incorrect"
Choose random number (1~10) : 4
[1] "Incorrect"
Choose random number (1~10) : 1
[1] "Incorrect"
Choose random number (1~10) : 5
[1] "Incorrect"
Choose random number (1~10) : 6
[1] "Incorrect"
Choose random number (1~10) : 7
[1] "Correct"
>
```

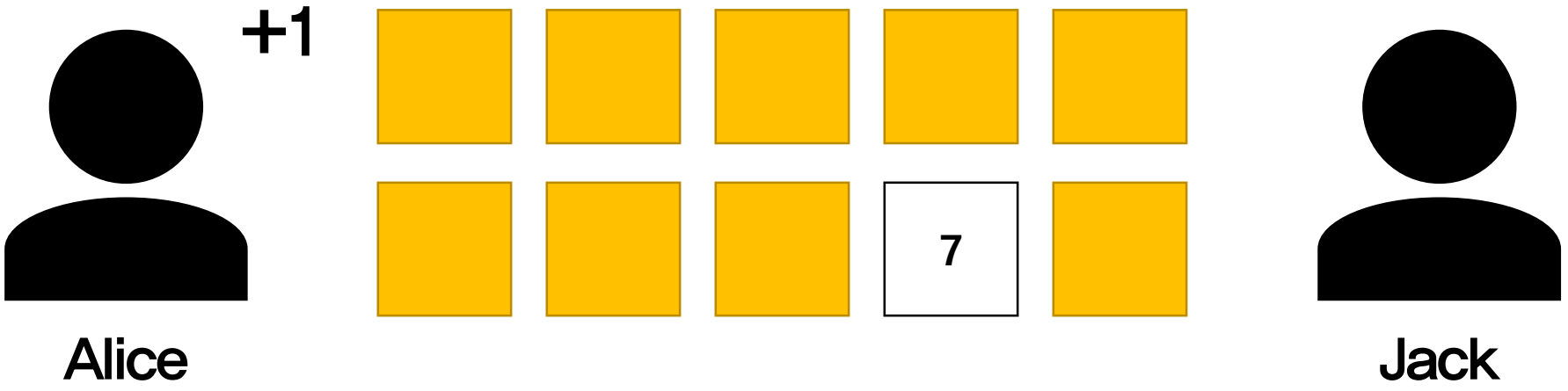
- 1부터 10까지의 수 중 하나를 무작위로 선택 후 맞춰야 탈출하는 게임을 만들어보자.

- sample(A:B, C) 함수
  - A부터 B까지의 숫자 중 C개를 뽑아서 벡터로 반환한다.

- 입력 받기
  - readline(prompt= “PRINT”) 함수를 이용하여 “PRINT” 의 내용을 출력하고 입력 받은 것을 벡터로 반환한다.

문제

- Jack과 Alice는 1부터 10까지 적힌 카드로 게임을 시작한다. 카드를 섞어 한 카드를 뽑아 그 카드가 짝수이면 Jack이 1점을, 홀수이면 Alice가 1점을 가져간다. 매 시도마다 카드를 뽑고 확인한 다음 다시 넣고 섞는다. 게임을 한 번 할 때, 카드 뽑기를 100번 한다. (즉, 한 게임에서 최대 획득 가능 점수는 100점)
- 이 게임을 10번하여 Alice가 각 게임에서 가져가는 평균 점수는 얼마일까?



문제 (2)

- 1. Jack과 Alice의 전역 변수 벡터를 설정한다. (전역 변수 설정은 <<- 로 한다.)

- 2. give <- function() {}
  - random <- sample(1:10, 1) # 1부터 10까지의 수에서 무작위 숫자 하나를 뽑는다.
  - 짝수일 경우, Jack이 1을 가져가고, 홀수일 경우, Alice가 1을 가져간다. (조건문)
  - Alice 벡터의 총 합을 반환한다.

- 3. average <- function() {}
  - 반복문을 통해 give 함수를 반복시키면서 give 함수로부터 반환되는 값을 모두 더한다.
  - 모두 더한 값을 10으로 나누어 반환한다. (또는 출력)

제출

- 문제에서 작성한 코드와 실행 결과 제출
  - 코드는 TXT나 HWP에 옮겨서 제출
  - 실행 결과는 스크린샷으로 찍어서 이미지 파일로 제출

- 제출 서식
  - 위 두 개의 파일 이름과 이메일 제목은 아래 서식으로 통일할 것
  - [9주차][학번][이름]통계학실습

- 제출 이메일
  - gtsk623@gmail.com