

전산통계학 실습

09. R 프로그래밍구조

R 프로그래밍

- R 프로그래밍의 장점

- 기존 통계학 언어 프로그래밍의 코드 작성 확장성을 위해 개발됨
 - 객체지향 언어 및 함수형 언어의 특징을 모두 가지고 있음
- 다수의 통계 라이브러리(패키지)가 존재함
- 분석 방법론 구현 및 빅데이터 시스템 개발 등이 간편함

- 함수형 프로그래밍의 특징

- 코드 수행 속도가 매우 빠른 편에 속함
- 단순한 코드로 디버깅이 쉬움
- 병렬 프로그래밍의 전환이 용이함

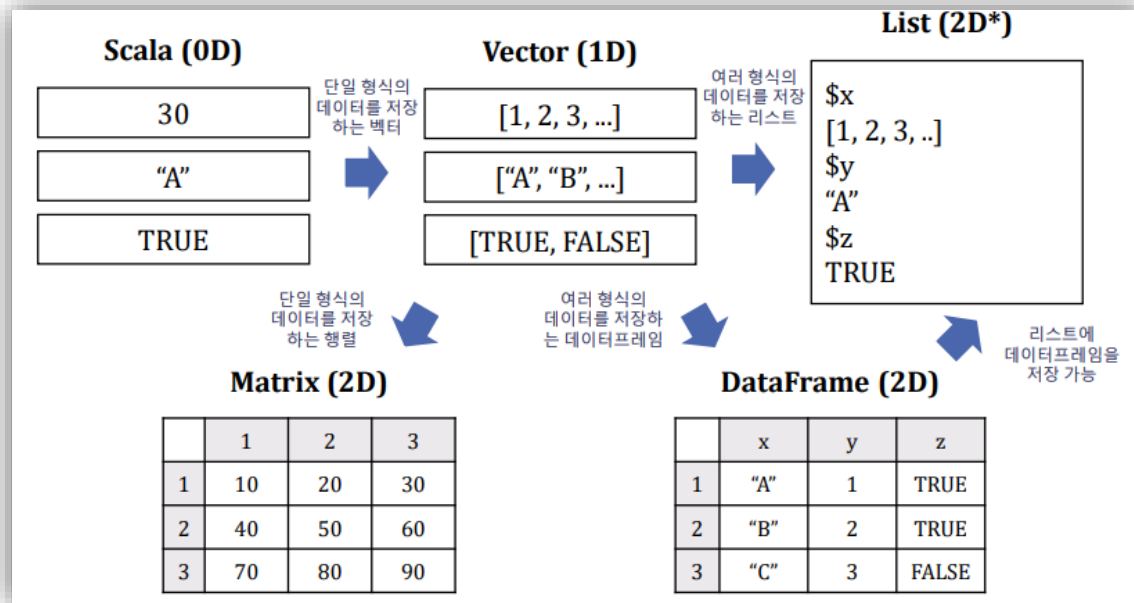


명시적인 반복법을 피하여,
'처리 함수' 등을 통한 반복 수행

R 프로그래밍

• R 프로그래밍

- 다른 컴퓨터 프로그래밍 언어처럼 다양한 프로그래밍이 반영됨
 - 함수, 조건문, 반복문
 - 산술, 부울 연산
 - 문자열 처리
 - 입력과 출력 등
- 그 중심에는 다양한 자료구조와 R 프로그래밍 함수가 존재함



함수와 조건문

```
> comparison <- function(a, b) {  
+   if (a > b) {  
+     return(a)  
+   }  
+   else {  
+     return(b)  
+   }  
+ }  
> comparison(2, 5)  
[1] 5  
> comparison(12, 7)  
[1] 12
```

- 함수

- 입력(input)에 대하여 계산을 수행한 출력(output)을 내어주는 수행(process)이 가능한 명령들 혹은 알고리즘들이 묶인 형태

- 함수의 생성

- > function_name <- function(parameters) { }
 - function_name 이름을 가진 function 을 생성
 - 입력 받을 parameters 를 지정

함수와 조건문

```
> comparison <- function(a=10, b) {  
+   if (a > b) {  
+     return(a)  
+   }  
+   else {  
+     return(b)  
+   }  
+ }  
> comparison(5)  
Error in comparison(5) : 기본값이 없는 인수 "b"가 누락되어 있습니다  
> comparison(b=5)  
[1] 10  
> comparison(a=7, 10)  
[1] 10  
> comparison(7, 10)  
[1] 10
```

• 함수의 기본 파라미터

- 함수 선언 시 기본 파라미터를 설정하면, 함수 이용 시 지정해주지 않은 변수에 대해서는 기본값으로 함수가 수행됨
 - 전체 파라미터를 모두 입력하는 경우 구분해주지 않아도 됨
 - 단, 기본 파라미터를 이용하기 위해서 전체 파라미터를 모두 입력하지 않는 경우에는 입력되는 값이 어떤 파라미터인지 구분해주어야 함

함수와 조건문

```
> sum_odd <- function(...) {  
+   args <- list(...)  
+   result <- 0  
+   for (a in args) {  
+     if (a %% 2 != 0) {  
+       result <- result + a  
+     }  
+   }  
+   return(result)  
+ }  
> sum_odd(1:10)  
[1] 1 2 3 4 5 6 7 8 9 10  
경고메시지(들):  
In if (a%%2 != 0) { :  
  length > 1 이라는 조건이 있고, 첫번째 요소만이 사용될 것입니다  
> sum_odd(1, 7, 8, 2, 5, 6, 3, 3, 5, 7)  
[1] 31
```

- 가변 길이 파라미터

- 정해지지 않은 수만큼 입력 받고 싶을 때, 가변 길이 파라미터 이용
- 입력 받은 매개변수는 리스트 형태로 할당 후 사용

함수와 조건문

```
> sum_odd <- function(...) {  
+   args <- list(...)  
+   print(args)  
+   result <- 0  
+   for (a in args) {  
+     if (a %% 2 != 0) {  
+       result <- result + a  
+     }  
+   }  
+   return(result)  
+ }  
> sum_odd(1:10)  
[[1]]  
[1] 1 2 3 4 5 6 7 8 9 10  
  
[1] 1 2 3 4 5 6 7 8 9 10
```

경고메시지(들):

In if (a%%2 != 0) { :

length > 1 이라는 조건이 있고, 첫번째 요소만이 사용될 것입니다

```
> sum_odd <- function(...) {  
+   args <- unlist(list(...))  
+   result <- 0  
+   for (a in args) {  
+     if (a %% 2 != 0) {  
+       result <- result + a  
+     }  
+   }  
+   return(result)  
+ }  
> sum_odd(1:10)  
[1] 25
```

• 가변 길이 파라미터

- 만약 여러 데이터를 가진 벡터 등을 가변 길이로 대입하는 경우 리스트이기 때문에, 벡터 자체가 대입될 수 있음
- 따라서 `unlist(.)` 함수를 이용하여 ‘벡터화’ 하여 이용

함수와 조건문

```
> g <- function(x) {  
+   y <- 3 * x + 2  
+   h <- function(x, y) {  
+     z <- x + y  
+     z  
+   }  
+   h(x, y)  
+ }  
> g(5)  
[1] 22
```

- 함수 내 함수 선언
 - 메모리 절약 용도
 - 함수 내부에서만 필요한 함수인 경우 내부에서 선언하여 이용 가능
 - 외부에서는 내부에 있는 함수의 존재를 알 수 없음

함수와 조건문

```
> g <- function(x) {  
+   y <- 3 * x + 2  
+   h <- function(x, y) {  
+     z <- x + y  
+     z  
+   }  
+   h(x, y)  
+ }  
> g  
function(x) {  
  y <- 3 * x + 2  
  h <- function(x, y) {  
    z <- x + y  
    z  
  }  
  h(x, y)  
}  
> h  
에러: 객체 'h'를 찾을 수 없습니다  
> sum  
function (..., na.rm = FALSE) .Primitive("sum")
```

- 함수의 객체화

- R 프로그래밍에서 함수는 객체화 되어 있음
- function(.) 은 함수를 생성하는 R 내장 함수
- 콘솔에서 객체의 이름을 입력해도 객체가 출력됨 (함수도 마찬가지)

함수와 조건문

```
> comparison <- function(a=10, b) {  
+   if (a > b) {  
+     a  
+   }  
+   else {  
+     t <- c(a, b)  
+     t  
+   }  
+ }  
> comparison(15, 2)  
[1] 15  
> comparison(7, 10)  
[1] 7 10
```

- 함수의 반환

- > return(x)
 - 함수 결과 하나를 반환
 - 만약 여러 개의 값을 반환하고 싶은 경우 자료구조를 이용
- return 함수를 이용하지 않고 해당 변수를 단순 기재하여 반환 가능

- 조건문

- 다른 프로그래밍 언어처럼 (if / else if / else) 조건문을 사용 가능

반복문

```
> cumsum_for <- function(n) {  
+   sum <- 0  
+   for (i in 1:n){  
+     sum <- sum + i  
+   }  
+   return(sum)  
+ }  
> cumsum_for(10)  
[1] 55
```

```
> square_for <- function(data) {  
+   result <- c()  
+   for (n in data){  
+     result <- c(result, n^2)  
+   }  
+   return(result)  
+ }  
> square_for(1:5)  
[1] 1 4 9 16 25
```

- for 반복문

- > for (element in data) { ... }
 - 여러 값을 가진 주어진 데이터로부터 내부 값을 하나씩 대입하여 반복 수행
 - 주어진 데이터 내부를 모두 탐색하면 자동으로 종료됨
 - 주어진 데이터는 벡터여야 하며, 다른 자료구조의 경우 '벡터화' 하여 이용

반복문

```
> cumsum_while <- function(n) {  
+   sum <- 0  
+   i <- 1  
+   while (i <= n) {  
+     sum <- sum + i  
+     i <- i + 1  
+   }  
+   return(sum)  
+ }
```

```
> cumsum_while(10)  
[1] 55  
> cumsum_repeat(10)  
[1] 55
```

```
> cumsum_repeat <- function(n) {  
+   sum <- 0  
+   i <- 1  
+   repeat {  
+     if (i <= n) {  
+       sum <- sum + i  
+       i <- i + 1  
+     }  
+     else {  
+       break  
+     }  
+   }  
+   return(sum)  
+ }
```

- while 반복문
 - 조건이 만족되면 반복문 내부가 실행됨
 - 조건이 만족되지 않으면 종료
- repeat 반복문
 - 조건 없이 반복문이 무한히 실행됨

반복문

```
> odd3search <- function(data) {  
+   count <- 0  
+   result <- c()  
+   for (n in data) {  
+     if (n %% 2 != 0) {  
+       count <- count + 1  
+       result <- c(result, n)  
+     }  
+     else {  
+       next  
+     }  
+     if (length(result) == 3) {  
+       break  
+     }  
+   }  
+   return(result)  
+ }  
> v <- c(33, 44, 2, 6, 8, 7, 12, 11, 14, 15)  
> odd3search(v)  
[1] 33 7 11
```

- 반복문

- > break
 - 반복문 탈출(종료)
 - return(.) 과 같은 함수 종료를 통해 탈출도 가능
- > next
 - 상위의 가장 가까운 반복문으로 돌아가 수행

함수

```
> max_local <- 0
> max_global <<- 0
> deletemaxnums <- function(...) {
+   args <- unlist(list(...))
+   remains <- c()
+   if (length(args) == 0) {
+     return(remains)
+   }
+   max_local <- max(args)
+   max_global <<- max(args)
+   for (n in args) {
+     if (n != max_local){
+       remains <- c(remains, n)
+     }
+   }
+   return(remains)
+ }
> v <- c(10, 11, 12, 12, 13, 13, 14, 15, 16, 16)
> deletemaxnums(v)
[1] 10 11 12 12 13 13 14 15
> print(max_local)
[1] 0
> print(max_global)
[1] 16
```

- 지역변수와 전역변수

- 함수 바깥에서 유지되지 않는 지역변수와 달리, 전역변수로 생성한 변수는 함수 내부에서 '<<-' 를 이용하여 수정하면 그 내용이 반영됨

함수

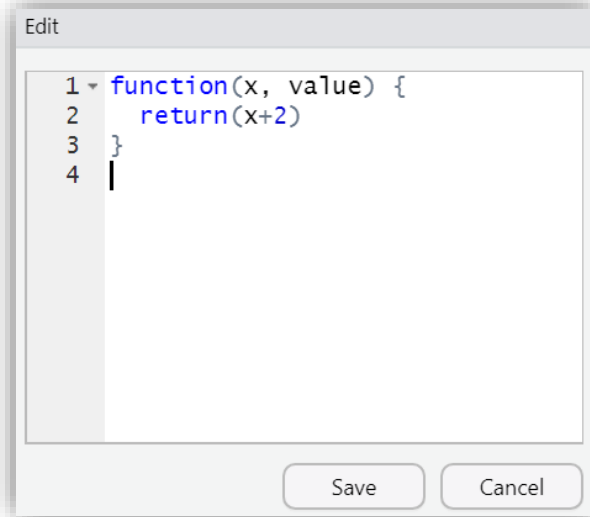
```
> x <- c(55, 77, 39, 82, 100)
> classes <- c("KOR", "MAT", "ENG", "SCI", "SOC")
> names(x) <- classes
> x
KOR MAT ENG SCI SOC
 55  77  39  82 100
> x <- "names<-"(x, value=classes)
> x
KOR MAT ENG SCI SOC
 55  77  39  82 100
```

```
> x[1]
KOR
 55
> x[1] <- 100
> x
KOR MAT ENG SCI SOC
100  77  39  82 100
> x <- "[<-"(x, 2, value=90)
> x
KOR MAT ENG SCI SOC
100  90  39  82 100
```

- 교체함수

- R에서는 함수 호출의 결과에 할당이 가능한 교체함수가 존재
- 명시적으로 기존 함수 호출 자체에 할당하는 '<-'를 통해 대입이 가능
- 교체함수로 이용 가능한 함수는 'value' 파라미터를 포함하고 있음

함수



```
Edit
1 function(x, value) {
2   return(x+2)
3 }
4 |
```

Save Cancel

```
> g <- function(x, value) {
+   return(x+1)
+ }
> g(5)
[1] 6
```

```
> h <- edit(g)
> h(5)
[1] 7
```

- 코드 작성 함수

- > edit(function)

- 에디터(스크립트) 등이 아닌 R 인터랙티브 모드(콘솔)에서 함수 수정 가능한 기능
 - 기본 에디터가 활성화 되어 입력된 파라미터인 function 에 대한 코드 수정 가능
 - 수정 결과를 저장할 변수(함수)를 같이 입력하여 수정 후 이용

산술 연산

산술 연산	설명
$x + y$	덧셈
$x - y$	뺄셈
$x * y$	곱셈
x / y	나눗셈
$x ^ y$	x의 y승
$x \% y$	나머지 연산
$x \%/ \% y$	나눗셈 결과의 정수형
수학 함수	설명
$\log(x, \text{base})$	밑이 base인 log함수 (기본값 = e)
$\log_2(x), \log_{10}(x)$	밑이 2, 10인 log함수
$\sin(x), \cos(x), \tan(x)$	삼각함수
$\exp(x)$	e의 x승

산술 연산

```
> "%2x+3y%" <- function(x,y) return(2*x+3*y)
> 10 %2x+3y% 4
[1] 32
> "%r%" <- function(x,y) return(sqrt(x^2+y^2))
> 3 %r% 4
[1] 5
```

- 자신만의 산술 연산자

- > "%...%" <- function(a, b) return(...)

- 함수 이름과 끝에 %를 붙이고 두 파라미터와 반환값을 설정하여 자신만의 바이너리 산술 연산자를 만들 수 있음

부울 연산

산술 연산	설명
<code>x == y</code>	일치 비교
<code>x >= y, x <= y</code>	크기 비교
<code>x && y</code>	AND (스칼라)
<code>x y</code>	OR (스칼라)
<code>x & y</code>	AND (벡터)
<code>x y</code>	OR (벡터)
<code>!x</code>	NOT

부울 연산

```
> x <- c(TRUE, FALSE, TRUE)
> y <- c(TRUE, FALSE, FALSE)
> x & y
[1] TRUE FALSE FALSE
> x && y
[1] TRUE
> x * 3
[1] 3 0 3
> (1 < 2) * (3 < 4) == TRUE
[1] TRUE
```

- 부울 연산의 사용
 - & 연산은 벡터의 각 원소마다(element-wise) 실행
 - && 연산은 벡터의 첫 번째 원소([1])에 대해서만 실행
- TRUE/FALSE (T/F)
 - TRUE = 1 / FALSE = 0 이므로 산술 연산이 가능

입력과 출력 (I/O)

```
[1] TRUE
> add <- function() {
+   num1 <- readline("First Num: ")
+   num2 <- readline("Second Num: ")
+   result <- as.numeric(num1) + as.numeric(num2)
+   print(result)
+   return(result)
+ }
> add()
First Num: 30
Second Num: 11
[1] 41
[1] 41
```

- 입력

- > readline("comment")
 - 키보드로부터 한 줄 입력 받는 함수
 - 입력 받은 내용은 character 형태로 저장

- 출력

- > print(...)
 - 함수로 입력된 내용을 출력 (만약 source 함수 이용 시 출력을 위해 필요)

문자열 처리

```
> str1 <- "North"
> str2 <- "Pole"
> str3 <- "South Pole"
> nchar(str1)
[1] 5
> paste(str1, str2)
[1] "North Pole"
> paste(str1, str2, sep="")
[1] "NorthPole"
> paste(str1, str2, sep=".")
[1] "North.Pole"
> substr(str3, 7, 10)
[1] "Pole"
> strsplit(str3, split=" ")
[[1]]
[1] "South" "Pole"
```

- 문자열 처리
 - > nchar(x)
 - 해당 문자열의 길이 반환

문자열 처리

```
> str1 <- "North"
> str2 <- "Pole"
> str3 <- "South Pole"
> nchar(str1)
[1] 5
> paste(str1, str2)
[1] "North Pole"
> paste(str1, str2, sep="")
[1] "NorthPole"
> paste(str1, str2, sep=".")
[1] "North.Pole"
> substr(str3, 7, 10)
[1] "Pole"
> strsplit(str3, split=" ")
[[1]]
[1] "South" "Pole"
```

- 문자열 처리

- > paste(..., sep=" ", collapse=" ")
 - 여러 개의 문자열들을 모두 이어 붙여 반환
 - sep 옵션을 통해 각 문자열 사이 들어가는 문자열 지정 가능 (기본값 = 공백)
 - collapse 옵션을 통해 완성된 문자열들을 하나로 합치면서 사이 문자열 지정 가능

문자열 처리

```
> str1 <- "North"
> str2 <- "Pole"
> str3 <- "South Pole"
> nchar(str1)
[1] 5
> paste(str1, str2)
[1] "North Pole"
> paste(str1, str2, sep="")
[1] "NorthPole"
> paste(str1, str2, sep=".")
[1] "North.Pole"
> substr(str3, 7, 10)
[1] "Pole"
> strsplit(str3, split=" ")
[[1]]
[1] "South" "Pole"
```

- 문자열 처리

- > substr(x, start, end)
 - 입력 문자열의 start 부터 end 까지 범위에 위치한 부분 문자열을 반환
- > strsplit(x, split="")
 - 입력 문자열을 split 문자열 기준으로 나눈 결과를 리스트로 반환

예시 데이터셋 만들기

- 예시 데이터셋 `class_data`
 - 학생 N 명의 과목점수 및 관련 계산이 포함된 데이터프레임
 - NUM
 - “STUXXX” 포맷을 가진, 학생들의 번호 1 ~ N 까지의 숫자를 가짐
 - 단, 최대 번호가 가지는 자릿수에 맞도록 포맷이 지켜져야 함
 - KOR, MAT, ENG, SCI, SOC
 - 무작위로 생성된 과목점수가 대입됨
 - 단, 20% 학생은 80 ~ 100점, 60% 학생은 30 ~ 79점
 - 나머지 20% 학생은 0 ~ 29 점의 점수를 가짐
 - AVG, TOT
 - 5가지 과목의 평균과 총합 점수를 가짐
 - P/F
 - MAT 점수가 60점 이상인 학생의 경우 PASS, 아닌 경우 FAIL

예시 데이터셋 만들기

- 예시 데이터셋 class_data
 - NUM
 - 문자열 처리를 이용하여 생성

```
stu_num_n <- function(n) {  
  i <- as.integer(log10(n))  
  result <- c()  
  for (num in 1:n) {  
    num_i <- as.integer(log10(num))  
    if (num_i < i) {  
      zeros <- rep("0", i - num_i)  
      zeros <- paste(zeros, collapse="")  
      num <- paste(zeros, num, sep="")  
    }  
    result <- c(result, num)  
  }  
  result <- paste("STU", result, sep="")  
  return(result)  
}
```

예시 데이터셋 만들기

- 예시 데이터셋 class_data
 - KOR, MAT, ENG, SCI, SOC
 - 무작위 생성 함수 sample(data, size, replace=T/F)
 - 원하는 개수에 맞추어 무작위 점수 생성

```
random_scores <- function(n) {  
  percent20 <- as.integer(0.2 * n)  
  percent60 <- n - 2 * percent20  
  scores1 <- sample(80:100, percent20, replace=T)  
  scores2 <- sample(30:79, percent60, replace=T)  
  scores3 <- sample(0:29, percent20, replace=T)  
  result <- c(scores1, scores2, scores3)  
  return(sample(result))  
}
```

예시 데이터셋 만들기

- 예시 데이터셋 class_data
 - AVG, TOT, P/F
 - 주어진 데이터프레임에 대한 데이터 처리 함수 이용

```
gen_class_data <- function(n) {  
  class_data <- data.frame(NUM=stu_num_n(n))  
  classes <- c("KOR", "MAT", "ENG", "SCI", "SOC")  
  for (c in classes) {  
    class_data[c] <- random_scores(n)  
  }  
  class_data$AVG <- apply(class_data[2:6], 1, mean)  
  class_data$TOT <- apply(class_data[2:6], 1, sum)  
  class_data$"P/F" <- ifelse(class_data$MAT >= 60, "PASS", "FAIL")  
  return(class_data)  
}
```

예시 데이터셋 만들기

- 예시 데이터셋 class_data

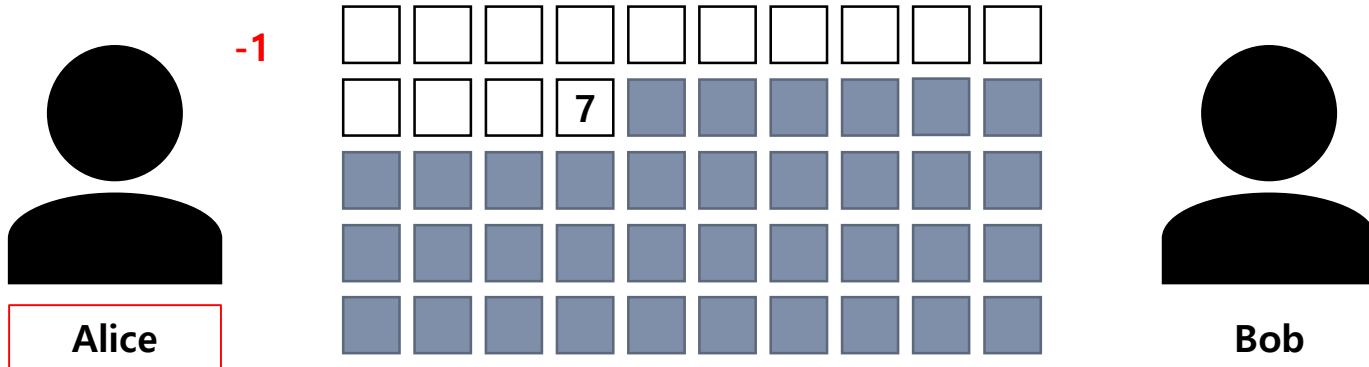
```
stu_num_n <- function(n) {  
  i <- as.integer(log10(n))  
  result <- c()  
  for (num in 1:n) {  
    num_i <- as.integer(log10(num))  
    if (num_i < i) {  
      zeros <- rep("0", i - num_i)  
      zeros <- paste(zeros, collapse="")  
      num <- paste(zeros, num, sep="")  
    }  
    result <- c(result, num)  
  }  
  result <- paste("STU", result, sep="")  
  return(result)  
}  
  
random_scores <- function(n) {  
  percent20 <- as.integer(0.2 * n)  
  percent60 <- n - 2 * percent20  
  scores1 <- sample(80:100, percent20, replace=T)  
  scores2 <- sample(30:79, percent60, replace=T)  
  scores3 <- sample(0:29, percent20, replace=T)  
  result <- c(scores1, scores2, scores3)  
  return(sample(result))  
}  
  
gen_class_data <- function(n) {  
  class_data <- data.frame(NUM=stu_num_n(n))  
  classes <- c("KOR", "MAT", "ENG", "SCI", "SOC")  
  for (c in classes) {  
    class_data[c] <- random_scores(n)  
  }  
  class_data$AVG <- apply(class_data[2:6], 1, mean)  
  class_data$TOT <- apply(class_data[2:6], 1, sum)  
  class_data$P/F <- ifelse(class_data$MAT >= 60, "PASS", "FAIL")  
  return(class_data)  
}  
  
class_data <- gen_class_data(40)  
print(class_data)
```



	NUM	KOR	MAT	ENG	SCI	SOC	AVG	TOT	P/F
1	STU01	82	37	14	19	43	39.0	195	FAIL
2	STU02	75	35	80	66	89	69.0	345	FAIL
3	STU03	68	83	51	1	7	42.0	210	PASS
4	STU04	98	60	13	80	78	65.8	329	PASS
5	STU05	51	35	11	6	91	38.8	194	FAIL
6	STU06	6	35	70	85	26	44.4	222	FAIL
7	STU07	31	99	44	66	23	52.6	263	PASS
8	STU08	21	63	42	40	79	49.0	245	PASS
9	STU09	58	51	65	78	70	64.4	322	FAIL
10	STU10	77	2	0	13	82	34.8	174	FAIL
11	STU11	58	39	41	17	61	43.2	216	FAIL
12	STU12	81	55	45	42	60	56.6	283	FAIL
13	STU13	45	62	93	44	9	50.6	253	PASS
14	STU14	31	91	2	18	91	46.6	233	PASS
15	STU15	99	90	91	2	51	66.6	333	PASS
16	STU16	3	43	37	42	34	31.8	159	FAIL
17	STU17	74	8	79	36	65	52.4	262	FAIL
18	STU18	2	50	31	64	28	35.0	175	FAIL
19	STU19	53	63	77	72	32	59.4	297	PASS
20	STU20	58	19	10	100	63	50.0	250	FAIL
21	STU21	49	40	49	42	43	44.6	223	FAIL
22	STU22	75	55	70	73	77	70.0	350	FAIL
23	STU23	30	9	54	48	81	44.4	222	FAIL
24	STU24	21	50	76	99	31	55.4	277	FAIL
25	STU25	30	79	84	90	42	65.0	325	PASS
26	STU26	41	70	71	2	65	49.8	249	PASS
27	STU27	2	40	36	36	70	36.8	184	FAIL
28	STU28	57	80	60	37	81	63.0	315	PASS
29	STU29	80	86	56	100	75	79.4	397	PASS
30	STU30	91	14	83	62	72	64.4	322	FAIL
31	STU31	48	47	96	49	44	56.8	284	FAIL
32	STU32	80	73	67	40	60	64.0	320	PASS
33	STU33	56	29	56	50	62	50.6	253	FAIL
34	STU34	44	40	2	45	66	39.4	197	FAIL
35	STU35	28	74	6	41	26	35.0	175	PASS
36	STU36	8	10	78	84	97	55.4	277	FAIL
37	STU37	49	82	46	46	50	54.6	273	PASS
38	STU38	48	90	42	89	85	70.8	354	PASS
39	STU39	79	4	81	62	9	47.0	235	FAIL
40	STU40	98	32	80	66	5	56.2	281	FAIL

과제

- 카드 뒤집기 게임 구현을 통한 결과 도출
 - Alice와 Bob이 1~50 숫자가 적힌 카드로 뒤집기 게임을 진행한다.
 - 카드를 섞고 무작위로 카드를 뽑아서 다음과 같이 점수를 받는다.
 - 게임은 Alice부터 시작하며, 각자의 차례마다 하나의 카드를 뽑는다.
 - 둘은 서로 다른 규칙을 통해 점수를 얻는다.
 - Alice: 자신 차례에 뽑은 카드가 짝수인 경우 +2점, 홀수인 경우 -1점
 - Bob: 자신 차례에 뽑은 카드가 짝수인 경우 +0점, 홀수인 경우 +1점
 - 50개의 카드를 모두 뽑으면, 게임이 종료된다.
 - 위 카드 게임을 구현한 뒤, 총 10번 반복하여 Alice가 총 10번의 게임 동안 얻은 평균 점수가 얼마나 되는지 계산하여 출력한다.



과제 제출

- 제출 목록
 - 작성한 코드 파일(.R)
 - 결과 출력 화면 (.PDF)
 - 터미널 캡처(이미지)를 Word 혹은 HWP 에 붙여넣어 PDF 로 변환
- 제출 방법
 - 위 목록의 파일들을 압축
 - 아래 서식으로 압축파일 이름 지정
 - 블랙보드를 통해 제출
- 제출 서식 (XX = 주차번호 ex. 01, 02, ...)
 - 파일 이름: 전산통계학_실습과제_XX주차_학번_이름.zip