



# 난생 처음

## [ R 코딩 & 데이터 분석 ]

## Chapter 07. 제어문과 사용자 정의 함수 사용하기

# 목차

1. 이것이 조건문입니다
2. 이것이 반복문입니다
3. 사용자 정의 함수는 무엇인가요?
4. 조건에 맞는 데이터의 위치를 찾아봅니다

01

이것이 조건문입니다

# 01. 이것이 조건문입니다

## I. if-else문

- 조건문(conditional statement) : 조건에 따라 실행할 명령문

### [기본 구조]

```
if (비교 조건) {  
    조건이 참일 때 실행할 명령문(들)  
} else {  
    조건이 거짓일 때 실행할 명령문(들)  
}
```

# 01. 이것이 조건문입니다

## I. if-else문

[코드 7-1]

```
job.type <- 'A'
if (job.type == 'B') {
  bonus <- 200      # 직군이 B일 때 실행
} else {
  bonus <- 100      # 직군이 B가 아닌 나머지 경우 실행
}
print(bonus)
```

- 직군(job.type)이 'B'이면 보너스(bonus)를 200으로 하고 나머지 경우는 100으로 하는 조건문 if( ) 함수

# 01. 이것이 조건문입니다

## I. if-else문

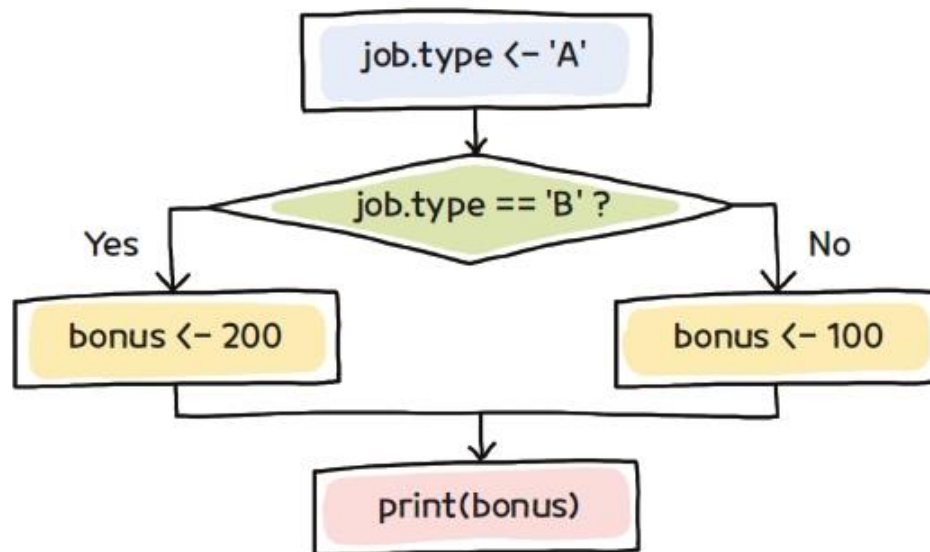


그림 7-1 조건문의 순서도

```
job.type <- 'A'
if (job.type == 'B') {
    bonus <- 200
} else {
    bonus <- 100
}
print(bonus)
```

# 01. 이것이 조건문입니다

## I. if-else문

[코드 7-2 (1)]

```
# else가 생략된 if문
job.type <- 'A'
bonus <- 100
if (job.type == 'B') {
    bonus <- 200
}
print(bonus)
# -----
a <- 10
if (a<5) {
    print(a)
} else {
    print(a*10)
    print(a/10)
}
```

# 직군이 B일 때 실행

# 01. 이것이 조건문입니다

## I. if-else문

[코드 7-2 (2)]

```
# -----  
a <- 10  
b <- 20  
if (a>5 & b>5) {           # and  
  print(a+b)  
}  
if (a>5 | b>30) {          # or  
  print(a*b)  
}
```



# 01. 이것이 조건문입니다

## I. if-else문

### [코드 7-2] 실행 결과(1)

```
> job.type <- 'A'
> bonus <- 100
> if (job.type == 'B') {
+   bonus <- 200          # 직군이 B일 때 실행
+ }
> print(bonus)
[1] 100
```

- if( ) 함수만으로도 조건문이 성립 (else 없어도 됨)
- 직군(job.type)이 'B'인 경우 보너스(bonus)는 200이고, 아닌 경우는 아무 명령도 실행하지 않음

# 01. 이것이 조건문입니다

## I. if-else문

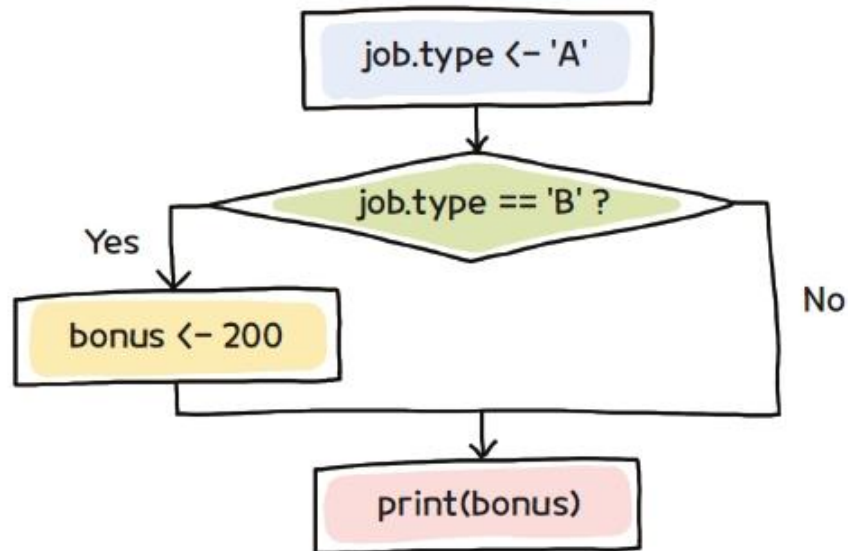


그림 7-2 if() 함수만을 사용한 조건문의 순서도

```
job.type <- 'A'
bonus <- 100
if (job.type == 'B') {
  bonus <- 200
}
print(bonus)
```

# 01. 이것이 조건문입니다

## I. if-else문

### [코드 7-2] 실행 결과(2)

```
> a <- 10
> if (a<5) {
+   print(a)
+ } else {
+   print(a*10)
+   print(a/10)
+ }
[1] 100
[1] 1
```

- $a < 5$ 라는 조건을 만족하면 `print(a)`가 실행되고, 만족하지 않으면 `print(a*10)`과 `print(a/10)`이 실행
- `{ }`는 프로그래밍에서 코드블록이 코드블록에 의해 묶인 명령문은 조건에 의해 모두 실행이 되든지, 모두 실행이 안 되든지 둘 중 하나

# 01. 이것이 조건문입니다

## I. if-else문

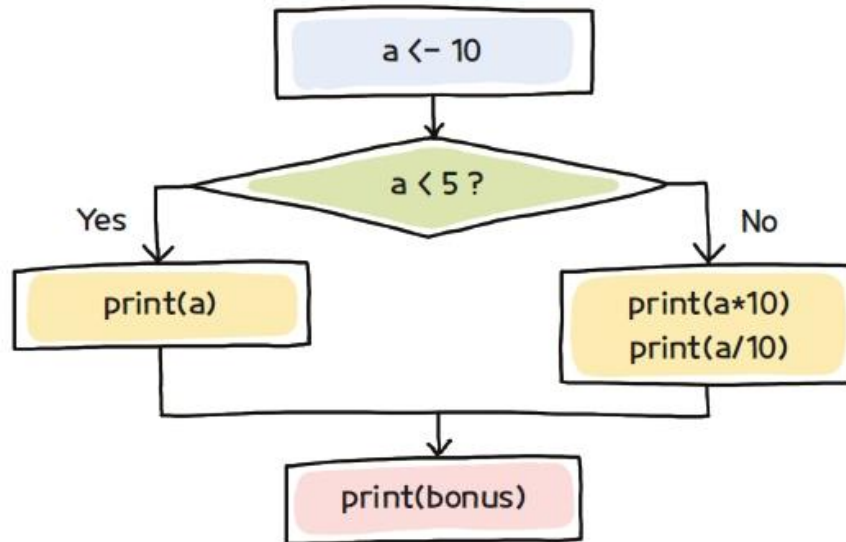


그림 7-3 코드블록으로 묶인 명령문이 있는 조건문의 순서도

```
a <- 10
if (a<5) {
  print(a)
} else {
  print(a*10)
  print(a/10)
}
```

코드블록 {}는 여러 개의 명령문을 하나의 작업 단위로 묶는 역할을 합니다.

# 01. 이것이 조건문입니다

## I. if-else문

### [코드 7-2] 실행 결과 (3)

```
> a <- 10
> b <- 20
> if (a>5 & b>5) {                                # and
+   print(a+b)
+ }
[1] 30
> if (a>5 | b>30) {                                # or
+   print(a*b)
+ }
[1] 200
```

- 조건문과 조건문을 연결할 때 앞에서 배운 논리연산자를 사용
- &로 연결되면 두 조건이 모두 만족을 해야 '참(True)'이 되고, |로 연결되면 두 조건 중 어느 하나만 만족하면 '참(True)'이 됨

# 01. 이것이 조건문입니다

## I. if-else문

### 하나 더 알기

if-else문을 서술할 때 다음과 같이 쓰면 오류가 발생합니다.

```
job.type <- 'A'
if (job.type == 'B') {
  bonus <- 200
}
else {                # 에러 발생, 윗줄로 옮겨야 한다
  bonus <- 100
}
```

else는 반드시 if문의 코드블록이 끝나는 표시 }와 같은 줄에 서술해야 합니다.

```
if (job.type == 'B') {
  bonus <- 200
}
```

# 01. 이것이 조건문입니다

## II. ifelse문

- 조건에 따라 선택할 값이 각각 하나씩이면 ifelse문을 이용하는 것이 편리

[기본 구조]

```
ifelse(비교 조건, 조건이 참일 때 선택할 값, 조건이 거짓일 때 선택할 값)
```

# 01. 이것이 조건문입니다

## II. ifelse문

### [코드 7-3]

```
# if-else를 이용한 처리 #####  
a <- 10  
b <- 20  
  
if (a>b) {  
  c <- a  
} else {  
  c <- b  
}  
print(c)  
  
# ifelse를 이용한 처리 #####  
a <- 10  
b <- 20  
  
c <- ifelse(a>b, a, b)  
print(c)
```



# 01. 이것이 조건문입니다

## III. if-else의 반복

[코드 7-4]

```
score <- 85

if (score > 90) {
  grade <- 'A'
} else if (score > 80) {
  grade <- 'B'
} else if (score > 70) {
  grade <- 'C'
} else if (score > 60) {
  grade <- 'D'
} else {
  grade <- 'F'
}

print(grade)
```

## 01. 이것이 조건문입니다

---

### III. if-else의 반복

[실행 결과]

```
> print(grade)
[1] "B"
```

## LAB. 회원 등급 분류하고 혜택 부여하기

한 서점은 앞서와 같이 최근 3개월간 구매 금액에 따라 회원 등급을 분류하고 등급별 혜택을 다르게 하고 있습니다. 고객의 3개월간 구매 금액을 입력받으면 회원 등급을 분류하고 적립 혜택을 부여하는 프로그램을 만들어봅시다.

	 플래티넘	 골드	 실버	 프렌즈
최근 3개월 순수구매액	30만원 이상	20~30만원 이상	10~20만원 이상	10만원 미만
적립 혜택	구매액의 7%	구매액의 5%	구매액의 3%	구매액의 1%

## LAB. 회원 등급 분류하고 혜택 부여하기

1. svDialogs 패키지를 사용해 구매 금액을 입력받는 팝업창을 만들고 그 값을 변수 purchase에 저장한 다음 숫자형으로 변환합니다.

```
library(svDialogs)
purchase <- dlgInput('Enter the purchase amount')$res
purchase <- as.numeric(purchase )
```

2. 회원 등급(type)과 적립 혜택 비율(ratio)을 저장할 변수를 선언합니다.

```
type <- NULL
ratio <- NULL
```

## LAB. 회원 등급 분류하고 혜택 부여하기

3. 구매 금액에 따른 조건문을 작성하고 조건에 맞는 회원 등급과 적립 혜택을 저장합니다.

```
if (purchase >= 300000) {  
  type <- '플래티넘'  
  ratio <- 0.07  
} else if (purchase >= 200000) {  
  type <- '골드'  
  ratio <- 0.05  
} else if (purchase >= 100000) {  
  type <- '실버'  
  ratio <- 0.03  
} else {  
  type <- '프렌즈'  
  ratio <- 0.01  
}
```

## LAB. 회원 등급 분류하고 혜택 부여하기

4. 고객의 회원 등급과 적립 혜택 정보를 출력합니다.

```
cat('고객님은', type, '회원으로 구매액의', ratio*100, '%가 적립됩니다.')
```

02

이것이 반복문입니다

## 02. 이것이 반복문입니다

- 반복 작업에 사용되는 명령문이 for문과 while문입니다.

### I. for문

- 반복 범위는 반복 변수에 할당할 값을 모아둔 벡터로 이 벡터의 길이만큼 for문은 반복됨
- for문이 한 번씩 수행될 때마다 반복 범위의 값을 하나씩 가져와 반복 변수에 저장한 뒤 코드블록 안에 있는 명령문을 실행함

#### [for문의 구조]

```
for(반복 변수 in 반복 범위) {  
    반복할 명령문(들)  
}
```



## 02. 이것이 반복문입니다

### I. for문

[코드 7-5]

```
for(i in 1:5) {  
    print('*')  
}
```

[실행 결과]

```
> for(i in 1:5) {  
+   print('*')  
+ }  
[1] "*"  
[1] "*"  
[1] "*"  
[1] "*"  
[1] "*"
```

## 02. 이것이 반복문입니다

### I. for문

#### [코드 7-5] 실행 결과

- '\*'를 5회 출력하는 for문의 반복 변수 i는 반복이 어디까지 이루어졌는지에 대한 정보를 저장하고 있는 변수
- i에는 반복이 진행될 때마다 반복 범위의 값이 하나씩 차례대로 저장됨

첫 번째 반복 시 i에 저장된 값은 1  
두 번째 반복 시 i에 저장된 값은 2  
세 번째 반복 시 i에 저장된 값은 3  
네 번째 반복 시 i에 저장된 값은 4  
다섯 번째 반복 시 i에 저장된 값은 5

- for( ) 함수 다음에 있는 코드블록 { }에는 반복할 명령문들이 포함됨

## 02. 이것이 반복문입니다

### I. for문

[코드 7-6]

```
for(i in 6:10) {  
    print(i)  
}
```

[실행 결과]

```
> for(i in 6:10) {  
+   print(i)  
+ }  
[1] 6  
[1] 7  
[1] 8  
[1] 9  
[1] 10
```

## 02. 이것이 반복문입니다

### I. for문

[코드 7-7]

```
for(i in 1:9) {  
    cat('2 *', i, '=', 2*i, '\n')  
}
```

[실행 결과]

```
> for(i in 1:9) {  
+   cat('2 *', i, '=', 2*i, '\n')  
+ }  
2 * 1 = 2  
2 * 2 = 4  
...(생략)  
2 * 8 = 16  
2 * 9 = 18
```

- cat() 함수는 한 줄에 여러 개의 값을 결합하여 출력할 때 사용
- 'Wn'은 줄바꿈을 하기 위한 특수 문자

## 02. 이것이 반복문입니다

### I. for문

[코드 7-8]

```
for(i in 1:20) {  
  if(i%%2==0) {                # 짝수인지 확인  
    cat(i, ' ')  
  }  
}
```

[실행 결과]

```
> for(i in 1:20) {  
+   if(i%%2==0) {                # 짝수인지 확인  
+     cat(i, ' ')  
+   }  
+ }  
2  4  6  8 10 12 14 16 18 20
```

## 02. 이것이 반복문입니다

### I. for문

[코드 7-9]

```
sum <- 0
for(i in 1:100) {
  sum <- sum + i          # sum에 i 값을 누적
}
print(sum)
```

[실행 결과]

```
> sum <- 0
> for(i in 1:100) {
+   sum <- sum + i      # sum에 i 값을 누적
+ }
> print(sum)
[1] 5050
```

## 02. 이것이 반복문입니다

### I. for문

[코드 7-10]

```
norow <- nrow(iris)           # iris의 행의 수
mylabel <- c()                 # 비어있는 벡터 선언
for(i in 1:norow) {
  if (iris$Petal.Length[i] <= 1.6) { # 꽃잎의 길이에 따라 레이블 결정
    mylabel[i] <- 'L'
  } else if (iris$Petal.Length[i] >= 5.1) {
    mylabel[i] <- 'H'
  } else {
    mylabel[i] <- 'M'
  }
}
print(mylabel)                # 레이블 출력
newds <- data.frame(iris$Petal.Length, mylabel) # 꽃잎 길이와 레이블 결합
head(newds)                   # 새로운 데이터셋 내용 출력
```

## 02. 이것이 반복문입니다

### I. for문

#### [코드 7-10] 실행 결과

```
...(생략)
> print(mylabel)                                # 레이블 출력
[1] "L" "L" "L" "L" "L" "M" "L" "L" "L" "L" "L" "L" "L" "L" "L" "L" "L"
[18] "L" "M" "L" "M" "L" "L" "M" "M" "L" "L" "L" "L" "L" "L" "L" "L" "L"
[35] "L" "L" "L" "L" "L" "L" "L" "L" "L" "L" "M" "L" "L" "L" "L" "L" "M"
... (생략)
[137] "H" "H" "M" "H" "H" "H" "H" "H" "H" "H" "M" "H" "H" "H"
> newds <- data.frame(iris$Petal.Length, mylabel) # 꽃잎 길이와 레이블 결합
> head(newds)                                     # 새로운 데이터셋 내용 출력
  iris.Petal.Length mylabel
1                1.4      L
2                1.4      L
3                1.3      L
4                1.5      L
5                1.4      L
6                1.7      M
```



## 02. 이것이 반복문입니다

### II. while문

#### [구조]

```
while (비교 조건) {  
    반복할 명령문(들)  
}
```

- while문은 비교 조건을 만족하는 동안 {} 안의 명령문들을 반복 실행함
- While문은 for문과 달리 몇 번이나 반복이 실행될지 쉽게 알 수 없음

## 02. 이것이 반복문입니다

### II. while문

[코드 7-11]

```
sum <- 0
i <- 1
while(i <=100) {
    sum <- sum + i          # sum에 i 값을 누적
    i <- i + 1             # i 값을 1 증가시킴
}
print(sum)
```

## 02. 이것이 반복문입니다

### II. while문

#### [코드 7-11] 실행 결과

```
> sum <- 0
> i <- 1
> while(i <=100) {
+     sum <- sum + i           # sum에 i 값을 누적
+     i <- i + 1              # i 값을 1 증가시킴
+ }
> print(sum)
[1] 5050
```

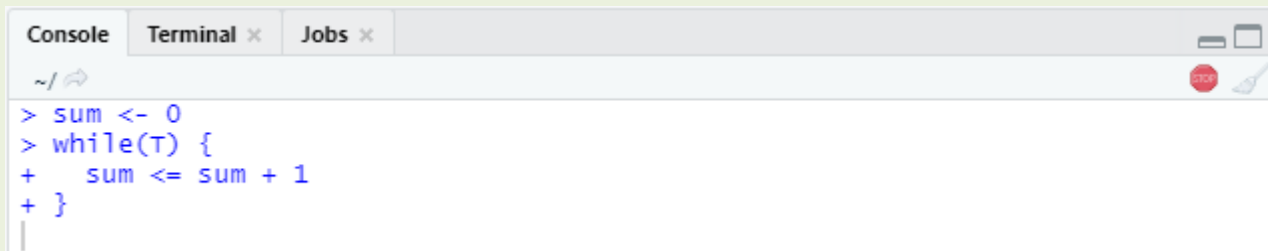
- while문을 실행하기 전에 sum과 i를 초기화하는 것과 { } 안에서 반복 변수에 해당하는 i값을 1씩 증가시키는 것에 주의.
- \* 조건문이 계속 충족되어 영원히 실행되는 반복문을 무한루프라고 함

## 02. 이것이 반복문입니다

### II. while문

#### 하나 더 알기

R 스튜디오에서 현재 실행 중인 작업을 중단시키려면 콘솔창 상단 오른쪽에 있는 빨간색 아이콘을 클릭하거나 키보드에서 <ESC>를 누르면 됩니다. 무한 루프에 빠진 경우에도 이 기능을 사용하여 중단시킬 수 있습니다.



```
Console Terminal x Jobs x
~/
> sum <- 0
> while(T) {
+   sum <= sum + 1
+ }
```

- 어떤 작업의 반복 횟수가 정해져 있다면 for문을 사용하고, 반복 횟수가 가변적이라면 while문을 사용하는 것이 편리함

## 02. 이것이 반복문입니다

### III. apply( ) 계열 함수

- apply( ) 함수는 매트릭스나 데이터프레임에 있는 행들이나 열들을 하나하나 차례로 꺼내어 평균이나 합계 등을 구하는 작업을 수행하고자 할 때 유용

[구조]

```
apply(데이터셋, 행/열 방향 지정, 적용 함수)
```

[매개변수]

- 데이터셋 : 반복 작업을 적용할 대상인 매트릭스나 데이터프레임 이름 입력.
- 행/열 방향 지정 : 행 방향 작업의 경우 1, 열 방향 작업의 경우 2를 지정.
- 적용 함수 : 반복 작업의 내용을 알려주는 것으로, R 함수이거나 다음 절에서 배울 사용자 정의 함수를 지정.

## 02. 이것이 반복문입니다

### III. apply( ) 계열 함수

[코드 7-12]

```
apply(iris[,1:4], 1, mean)      # 행 방향으로 함수 적용
apply(iris[,1:4], 2, mean)      # 열 방향으로 함수 적용
```

[코드 7-12] 실행 결과(1)

```
> apply(iris[,1:4], 1, mean)      # 행 방향으로 함수 적용
[1] 2.550 2.375 2.350 2.350 2.550 2.850 2.425 2.525 2.225
[10] 2.400 2.700 2.500 2.325 2.125 2.800 3.000 2.750 2.575
...(중간 생략)
[136] 4.775 4.425 4.200 3.900 4.375 4.450 4.350 3.875 4.550
[145] 4.550 4.300 3.925 4.175 4.325 3.950
```

- 명령문은 iris 데이터셋에서 4개의 열에 대해 행 방향으로 진행하면서 각 행의 평균(mean)을 계산하여 출력

## 02. 이것이 반복문입니다

### III. apply( ) 계열 함수

[코드 7-12] 실행 결과(1)

Sepal.Length	Sepal.width	Petal.Length	Petal.width	
5.1	3.5	1.4	0.2	mean()
4.9	3.0	1.4	0.2	mean()
4.7	3.2	1.3	0.2	
4.6	3.1	1.5	0.2	
5.0	3.6	1.4	0.2	
5.4	3.9	1.7	0.4	
4.6	3.4	1.4	0.3	
5.0	3.4	1.5	0.2	
4.4	2.9	1.4	0.2	
4.9	3.1	1.5	0.1	
5.4	3.7	1.5	0.2	
4.8	3.4	1.6	0.2	
4.8	3.0	1.4	0.1	
4.3	3.0	1.1	0.1	
5.8	4.0	1.2	0.2	mean()

그림 7-4 `apply(iris[,1:4], 1, mean)`

## 02. 이것이 반복문입니다

### III. apply( ) 계열 함수

#### [코드 7-12] 실행 결과(2)

```
> apply(iris[,1:4], 2, mean)           # 열 방향으로 함수 적용
Sepal.Length Sepal.Width Petal.Length Petal.Width
      5.843333      3.057333      3.758000      1.199333
```


- 이 명령문은 iris 데이터셋에서 4개의 열에 대해 열 방향으로 진행하면서 각 열의 평균(mean)을 계산하여 출력



## 02. 이것이 반복문입니다

### III. apply( ) 계열 함수

[코드 7-12] 실행 결과(2)



Sepal.Length	Sepal.width	Petal.Length	Petal.width
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2
5.4	3.9	1.7	0.4
4.6	3.4	1.4	0.3
5.0	3.4	1.5	0.2
4.4	2.9	1.4	0.2
4.9	3.1	1.5	0.1
5.4	3.7	1.5	0.2
4.8	3.4	1.6	0.2
4.8	3.0	1.4	0.1
4.3	3.0	1.1	0.1
5.8	4.0	1.2	0.2

mean()      mean()      mean()      mean()

그림 7-5 `apply(iris[,1:4], 2, mean)`

## LAB. 자격증 합격 여부 판단하기 I

응시자 10명의 점수가 주어졌을 때 각 응시자의 합격 여부를 판단하는 프로그램을 만들어봅시다.



1. 응시자의 과목별 성적을 데이터프레임으로 만듭니다.

```
sub1 <- c(14, 16, 12, 20, 8, 6, 12, 18, 16, 10)
sub2 <- c(18, 14, 14, 16, 10, 12, 10, 20, 14, 14)
sub3 <- c(44, 38, 30, 48, 42, 50, 36, 52, 54, 32)
score <- data.frame(sub1, sub2, sub3)
```

2. 응시자의 과목별 점수를 합하여 총점을 구하고 데이터프레임 `scoreset`에 추가합니다.

```
total <- apply(score, 1, sum)
scoreset <- cbind(score, total)
```

## LAB. 자격증 합격 여부 판단하기 I

3. 응시자의 합격 여부를 저장할 변수 result를 선언하고 초기화합니다.

```
result <- c( )
```

4. scoreset에 있는 데이터의 수만큼 코드블록 {} 안의 명령문들을 반복하며 i번째 응시생들의 합격 여부를 result에 저장합니다. . for문의 마지막에는 i번째 응시생의 합격 여부를 출력합니다.

```
for(i in 1:nrow(scoreset)){  
  if(scoreset[i,1] < 20*0.4 | scoreset[i,2] < 20*0.4 |  
    scoreset[i,3] < 60*0.4){  
    result[i] <- '불합격'  
  }else if(scoreset[i,4] >= 60){  
    result[i] <- '합격'  
  }else{  
    result[i] <- '불합격'  
  }  
  cat(i, '번째 응시생은', result[i], '입니다.\n')  
}
```

03

사용자 정의 함수는 무엇인가요?

## 03. 사용자 정의 함수는 무엇인가요?

### I. 사용자 정의 함수의 개념

- 사용자 정의 함수 : 사용자가 스스로 만드는 함수

[문법]

```
함수명 <- function(매개변수 목록) {  
  실행할 명령문(들)  
  return(함수의 실행 결과)  
}
```

[매개변수]

- 함수명 : 사용자 정의 함수의 이름으로 사용자가 만들 수 있다.
- 매개변수 목록 : 함수에 입력할 매개변수 이름을 지정한다.
- 실행할 명령문(들) : 함수에서 처리하고 싶은 내용을 작성한다.
- 함수의 실행 결과 : 함수의 실행 결과를 반환하며, 반환 결과가 없으면 `return()` 함수를 생략한다.

## 03. 사용자 정의 함수는 무엇인가요?

### I. 사용자 정의 함수의 개념

[코드 7-13]

```
mymax <- function(x,y) {  
  num.max <- x  
  if (y>x) {  
    num.max <- y  
  }  
  return(num.max)  
}
```

- 만들고자 하는 함수의 이름은 mymax, 이 함수가 입력받는 매개변수는 x와 y
- 코드블록 { } 안에 있는 if문은 x와 y 중 큰 값을 num.max에 저장하는 역할
- x와 y 중 큰 값이 반환(return)되는 것
- 이렇게 작성된 함수는 작성 코드 자체를 한 번 실행해 주어야 함

## 03. 사용자 정의 함수는 무엇인가요?

### I. 사용자 정의 함수의 개념

[코드 7-14]

```
mymax(10,15)
a <- mymax(20,15)
b <- mymax(31,45)
print(a+b)
```

[실행 결과]

```
> mymax(10,15)
[1] 15
> a <- mymax(20,15)
> b <- mymax(31,45)
> print(a+b)
[1] 65
```

### 03. 사용자 정의 함수는 무엇인가요?

#### I. 사용자 정의 함수의 개념

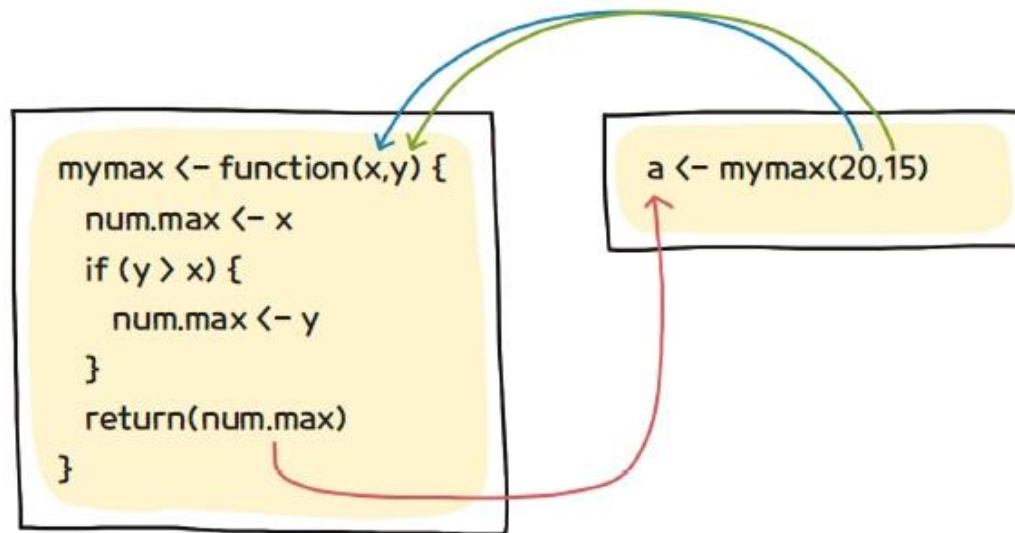


그림 7-6 mymax() 함수의 실행 과정



## 03. 사용자 정의 함수는 무엇인가요?

### II. 매개변수에 기본값 설정

[코드 7-15]

```
mydiv <- function(x,y=2) {  
  result <- x/y  
  return(result)  
}
```

```
mydiv(x=10,y=3)
```

```
mydiv(10,3)
```

```
mydiv(10)
```

# 매개변수 이름과 매개변수값을 쌍으로 입력

# 매개변수값만 입력

# x에 대한 값만 입력(y값이 생략됨)

## 03. 사용자 정의 함수는 무엇인가요?

### II. 매개변수에 기본값 설정

#### [코드 7-15] 실행 결과

```
> mydiv <- function(x,y=2) {  
+   result <- x/y  
+   return(result)  
+ }  
>  
> mydiv(x=10,y=3)           # 매개변수 이름과 매개변수값을 쌍으로 입력  
[1] 3.333333  
> mydiv(10,3)              # 매개변수값만 입력  
[1] 3.333333  
> mydiv(10)                # x에 대한 값만 입력(y값이 생략됨)  
[1] 5
```

## 03. 사용자 정의 함수는 무엇인가요?

### III. 여러 개의 값을 반환하는 경우

- list() 함수를 이용하여 여러 개의 결과값을 하나로 묶고 이것을 반환
- [코드 7-15]는 두 매개변수 x와 y를 입력받아 x/y값을 반환하는데, 사용자가 y값을 입력하지 않으면 y=2를 자동 적용하는 mydiv() 함수의 사용 예

#### [코드 7-16]

```
myfunc <- function(x,y) {  
  val.sum <- x+y  
  val.mul <- x*y  
  return(list(sum=val.sum, mul=val.mul))  
}  
  
result <- myfunc(5,8)  
s <- result$sum           # 5와 8의 합  
m <- result$mul           # 5와 8의 곱  
cat('5+8 =', s, '\n')  
cat('5*8 =', m, '\n')
```

## 03. 사용자 정의 함수는 무엇인가요?

### III. 여러 개의 값을 반환하는 경우

#### [코드 7-16] 실행 결과

```
> myfunc <- function(x,y) {  
+   val.sum <- x+y  
+   val.mul <- x*y  
+   return(list(sum=val.sum, mul=val.mul))  
+ }  
>  
> result <- myfunc(5,8)  
> s <- result$sum           # 5와 8의 합  
> m <- result$mul           # 5와 8의 곱  
> cat('5+8=', s, '\n')  
5+8 = 13  
> cat('5*8=', m, '\n')  
5*8 = 40
```

### 03. 사용자 정의 함수는 무엇인가요?

#### IV. 사용자 정의 함수의 저장과 재실행

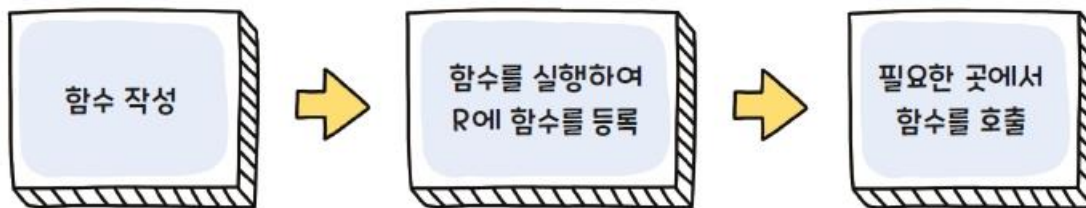


그림 7-7 사용자 정의 함수 사용 절차

- 자주 사용하게 될 사용자 정의 함수는 파일에 따로 모아두었다가 필요시 호출

```
mydiv <- function(x,y=2) {  
  result <- x/y  
  return(result)  
}
```

## 03. 사용자 정의 함수는 무엇인가요?

### IV. 사용자 정의 함수의 저장과 재실행

[코드 7-17]

```
setwd('c:/Rworks')           # myfunc.R이 저장된 폴더
source('myfunc.R')           # myfunc.R 안에 있는 함수 실행

# 함수 사용
a <- mydiv(20,4)              # 함수 호출
b <- mydiv(30,4)              # 함수 호출
a+b
mydiv(mydiv(20,2),5)          # 함수 호출
```

### 03. 사용자 정의 함수는 무엇인가요?

#### IV. 사용자 정의 함수의 저장과 재실행

##### [실행 결과]

```
> setwd('c:/Rworks')           # myfunc.R이 저장된 폴더
> source('myfunc.R')           # myfunc.R 안에 있는 함수 실행
>
> # 함수 사용
> a <- mydiv(20,4)              # 함수 호출
> b <- mydiv(30,4)              # 함수 호출
> a+b
[1] 12.5
> mydiv(mydiv(20,2),5)          # 함수 호출
[1] 2
```

- source('myfunc.R') 명령문의 의미는 myfunc.R 파일에 저장되어 있는 함수나 명령문들을 실행하라는 의미

## LAB. 자격증 합격 여부 판단하기 II

합격 여부를 판단하는 코드에 대해 별도의 함수로 정의하여 코드 관리를 용이하도록 만들겠습니다.



1. 우선 새로운 스크립트 파일을 만들어 사용자가 만들 함수의 이름(determine)과 사용할 매개변수(score)를 정의합니다. 작업 폴더는 'c:/Rworks'라 가정합니다.

```
determine <- function(score){ }
```

2. 정의한 함수의 코드블록 안에 이전 LAB에서 작성한 코드를 붙여 넣습니다. 그리고 함수 코드블록 마지막에 합격 여부의 결과가 저장된 값 result를 반환하는 명령문을 추가합니다. 작성을 마치면 스크립트 파일을 'myfunc.R'로 저장합니다.



## LAB. 자격증 합격 여부 판단하기 II

```
determine <- function(score){  
  total <- apply(score, 1, sum)  
  scoreset <- cbind(score, total)  
  result <- c( )  
  
  for(i in 1:nrow(scoreset)){  
    if(scoreset[i,1] < 20*0.4 | scoreset[i,2] < 20*0.4 |  
      scoreset[i,3] < 60*0.4){  
      result[i] <- '불합격'  
    }else if(scoreset[i, 4] >= 60){  
      result[i] <- '합격'  
    }else{  
      result[i] <- '불합격'  
    }  
    cat(i, '번째 응시생은', result[i], '입니다.\n')  
  }  
  return(result) # 저장된 결과 반환  
}
```

## LAB. 자격증 합격 여부 판단하기 II

3. 새로운 스크립트를 열어 방금 작성한 함수를 사용하기 위해 'myfunc.R'을 불러옵니다.

```
source('myfunc.R')
```

4. determine() 함수에 시험 점수를 입력해 실행해보고 반환되는 결과를 result에 저장합니다.

```
sub1 <- c(14, 16, 12, 20, 8, 6, 12, 18, 16, 10)
sub2 <- c(18, 14, 14, 16, 10, 12, 10, 20, 14, 14)
sub3 <- c(44, 38, 30, 48, 42, 50, 36, 52, 54, 32)
score <- data.frame(sub1, sub2, sub3)

result <- determine(score)
result
```

## LAB. 자격증 합격 여부 판단하기 II

5. 앞에서 3번째까지의 시험 성적을 입력해 실행해보고 결과를 확인합니다.

```
result <- determine(head(score, 3))  
result
```

6. 함수를 실행할 때 '1 번째 응시생은 합격 입니다.'와 같은 문구가 나오지 않게 수정하겠습니다.

```
# cat(i, '번째 응시생은', result[i], '입니다.\n')
```

7. 다시 'myfunc.R' 파일을 불러와 determine( ) 함수를 실행하고 결과를 확인합니다.

```
source('myfunc.R')  
result <- determine(score)  
result
```

04

조건에 맞는 데이터의 위치를  
찾아봅니다

## 04. 조건에 맞는 데이터의 위치를 찾아봅니다

- 데이터 분석을 하다 보면 자신이 원하는 데이터가 벡터나 매트릭스, 데이터프레임 안에서 어디에 위치하는지 알아야 할 때가 있음
- 편리하게 사용할 수 있는 함수가 `which( )`, `which.max( )`, `which.min( )` 함수

### [코드 7-18]

```
score <- c(76, 84, 69, 50, 95, 60, 82, 71, 88, 84)
which(score==69)      # 성적이 69인 학생은 몇 번째에 있나?
which(score>=85)      # 성적이 85 이상인 학생은 몇 번째에 있나?
max(score)           # 최고 점수는 몇 점인가?
which.max(score)      # 최고 점수는 몇 번째에 있나?
min(score)           # 최저 점수는 몇 점인가?
which.min(score)      # 최저 점수는 몇 번째에 있나?
```

## 04. 조건에 맞는 데이터의 위치를 찾아봅니다

### [코드 7-18] 실행 결과

```
> score <- c(76, 84, 69, 50, 95, 60, 82, 71, 88, 84)
> which(score==69)      # 성적이 69인 학생은 몇 번째에 있나?
[1] 3
> which(score>=85)      # 성적이 85 이상인 학생은 몇 번째에 있나?
[1] 5 9
> max(score)           # 최고 점수는 몇 점인가?
[1] 95
> which.max(score)      # 최고 점수는 몇 번째에 있나?
[1] 5
> min(score)           # 최저 점수는 몇 점인가?
[1] 50
> which.min(score)      # 최저 점수는 몇 번째에 있나?
[1] 4
```

- 벡터 안에서 어떤 값의 위치를 인덱스(index)라고 함
- which( ) 함수는 찾고자 하는 값의 인덱스를 알아내는 함수, which.max( ) 함수는 벡터 안에서 최대값의 인덱스, which.min( ) 함수는 최소값의 인덱스를 알아내는 함수임

## 04. 조건에 맞는 데이터의 위치를 찾아봅니다

### [코드 7-19]

```
score <- c(76, 84, 69, 50, 95, 60, 82, 71, 88, 84)
idx <- which(score<=60) # 성적이 60 이하인 값들의 인덱스
score[idx] <- 61        # 성적이 60 이하인 값들은 61점으로 성적 상향조정
score                  # 상향조정된 성적 확인

idx <- which(score>=80) # 성적이 80 이상인 값들의 인덱스
score.high <- score[idx] # 성적이 80 이상인 값들만 추출하여 저장
score.high      # score.high의 내용 확인
```

## 04. 조건에 맞는 데이터의 위치를 찾아봅니다

### [코드 7-19] 실행결과

```
> score <- c(76, 84, 69, 50, 95, 60, 82, 71, 88, 84)
> idx <- which(score<=60) # 성적이 60 이하인 값들의 인덱스
> score[idx] <- 61      # 성적이 60 이하인 값들은 61점으로 성적 상향조정
> score                # 상향조정된 성적 확인
[1] 76 84 69 61 95 61 82 71 88 84
>
> idx <- which(score>=80) # 성적이 80 이상인 값들의 인덱스
> score.high <- score[idx] # 성적이 80 이상인 값들만 추출하여 저장
> score.high           # score.high의 내용 확인
[1] 84 95 82 88 84
```



## 04. 조건에 맞는 데이터의 위치를 찾아봅니다

### 하나 더 알기

[코드 7-19]에서 조건에 맞는 값을 추출하는 작업의 경우 `subset()` 함수를 이용할 수도 있습니다.

```
score <- c(76, 84, 69, 50, 95, 60, 82, 71, 88, 84)
idx <- which(score>=80)
score.high <- score[idx]
```

`subset()` 함수를 사용하여 고쳐보면 다음과 같습니다.

```
score <- c(76, 84, 69, 50, 95, 60, 82, 71, 88, 84)
score.high <- subset(score, score>=80)
```

## 04. 조건에 맞는 데이터의 위치를 찾아봅니다

[코드 7-20]

```
idx <- which(iris$Petal.Length>5.0) # 꽃잎 길이가 5.0 이상인 값들의 인덱스  
idx  
iris.big <- iris[idx,] # 인덱스에 해당하는 값만 추출하여 저장  
iris.big
```

## 04. 조건에 맞는 데이터의 위치를 찾아봅니다

### [코드 7-20] 실행 결과

```
> idx <- which(iris$Petal.Length>5.0) # 꽃잎 길이가 5.0 이상인 값들의 인덱스
> idx
[1] 84 101 102 103 104 105 106 108 109 110 111 112 113 115 116 117 118
[18] 119 121 123 125 126 129 130 131 132 133 134 135 136 137 138 140 141
[35] 142 143 144 145 146 148 149 150
> iris.big <- iris[idx,] # 인덱스에 해당하는 값만 추출하여 저장
> iris.big
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
84	6.0	2.7	5.1	1.6	versicolor
101	6.3	3.3	6.0	2.5	virginica
102	5.8	2.7	5.1	1.9	virginica
103	7.1	3.0	5.9	2.1	virginica
...(생략)					

- which( ) 함수를 이용하면 원하는 값의 이름을 알아낼 수도 있음

## 04. 조건에 맞는 데이터의 위치를 찾아봅니다

### [코드 7-21]

```
score <- c(60,40,95,80)
names(score) <- c('John','Jane','Tom','David')
score                                     # 성적 데이터 출력
idx <- which.max(score)
names(score)[idx]                       # 성적이 제일 좋은 학생의 이름
```

### [실행 결과]

```
> score <- c(60,40,95,80)
> names(score) <- c('John','Jane','Tom','David')
> score                                     # 성적 데이터 출력
  John Jane Tom David
    60   40  95   80
> idx <- which.max(score)
> names(score)[idx]                       # 성적이 제일 좋은 학생의 이름
[1] "Tom"
```

## LAB. 언어 발달 상황 진단 프로그램 개발하기

한 검진 기관에서 A 개발자에게 언어 발달 진단 프로그램 개발을 의뢰했습니다. 진단 절차가 다음과 같다고 가정했을 때 다음 문제를 해결해봅시다.



말문이 트인 시기와 언어 이해력 점수 입력 → 각각을 등급으로 변환( [보기 1] 참고)  
→ 두 등급을 합산하여 진단( [보기 2] 참고)

보기 1

말문 트인 시기	~ 9개월 미만	9개월 ~ 14개월 미만	14개월 ~ 20개월 미만	20개월 ~ 26개월 미만	26개월 ~
등급	5	4	3	2	1
언어 이해력 점수	~70점 미만	70~90점 미만	90~110점 미만	110~130점 미만	130점 ~
등급	1	2	3	4	5

보기 2

총점	3점 미만	3~4점	5~6점	7~8점	9~10점
언어 발달 진단	매우 늦음	늦음	보통	빠름	매우 빠름

## LAB. 언어 발달 상황 진단 프로그램 개발하기

1. Stat2Data 패키지 안에 있는 ChildSpeaks 데이터셋을 불러온 뒤 구조를 살펴봅니다.

```
install.packages('Stat2Data')  
library(Stat2Data)  
data(ChildSpeaks)  
str(ChildSpeaks)
```

2. 말문이 트인 시기를 등급으로 환산하여 ChildSpeaks 데이터셋에 m1 열을 추가하여 저장합니다.

```
idx <- which(ChildSpeaks$Age < 9)  
ChildSpeaks[idx, 'm1'] <- 5  
idx <- which(ChildSpeaks$Age >= 9 & ChildSpeaks$Age < 15 )  
ChildSpeaks[idx, 'm1'] <- 4  
idx <- which(ChildSpeaks$Age >= 15 & ChildSpeaks$Age < 21 )  
ChildSpeaks[idx, 'm1'] <- 3  
idx <- which(ChildSpeaks$Age >= 21 & ChildSpeaks$Age < 27 )  
ChildSpeaks[idx, 'm1'] <- 2  
idx <- which(ChildSpeaks$Age >= 27 )  
ChildSpeaks[idx, 'm1'] <- 1
```

## LAB. 언어 발달 상황 진단 프로그램 개발하기

- 언어 이해력 점수를 등급으로 환산하여 ChildSpeaks 데이터셋에 m2 열을 추가하여 저장합니다.

```
idx <- which(ChildSpeaks$Gesell < 70)
ChildSpeaks$m2[idx] <- 1
idx <- which(ChildSpeaks$Gesell >= 70 & ChildSpeaks$Gesell < 90 )
ChildSpeaks$m2[idx] <- 2
idx <- which(ChildSpeaks$Gesell >= 90 & ChildSpeaks$Gesell < 110 )
ChildSpeaks$m2[idx] <- 3
idx <- which(ChildSpeaks$Gesell >= 110 & ChildSpeaks$Gesell < 130 )
ChildSpeaks$m2[idx] <- 4
idx <- which(ChildSpeaks$Gesell >= 130 )
ChildSpeaks$m2[idx] <- 5
```

## LAB. 언어 발달 상황 진단 프로그램 개발하기

4. ChildSpeaks 데이터셋에 total 열을 추가하고, m1과 m2를 합산하여 저장합니다.

```
ChildSpeaks$total <- ChildSpeaks$m1 + ChildSpeaks$m2
```

5. ChildSpeaks 데이터셋에 result 열을 추가하고, 총점을 기준으로 언어 발달 상황 진단 결과를 저장합니다.

```
idx <- which(ChildSpeaks$total < 3)
ChildSpeaks$result[idx] <- '매우느림'
idx <- which(ChildSpeaks$total >= 3 & ChildSpeaks$total < 5 )
ChildSpeaks$result[idx] <- '느림'
idx <- which(ChildSpeaks$total >= 5 & ChildSpeaks$total < 7 )
ChildSpeaks$result[idx] <- '보통'
idx <- which(ChildSpeaks$total >= 7 & ChildSpeaks$total < 9 )
ChildSpeaks$result[idx] <- '빠름'
idx <- which(ChildSpeaks$total >= 9 )
ChildSpeaks$result[idx] <- '매우빠름'
```



## LAB. 언어 발달 상황 진단 프로그램 개발하기

6. ChildSpeaks 데이터셋을 출력해 결과를 확인합니다.

```
ChildSpeaks  
ChildSpeaks[which.min(ChildSpeaks$total),]
```

# 실전분석. 종업원의 팁 분석하기

## [문제]

**reshape2** 패키지 안에 들어있는 **tips** 데이터셋은 한 종업원이 식당에서 일하면서 몇 달 동안 받은 여러 가지 팁에 대한 데이터를 저장하고 있습니다. 이 데이터를 살펴보고 각 질문에 따라 분석하겠습니다.



# 실전분석. 종업원의 팁 분석하기

## [해결]

1. reshape2 패키지 안에 들어있는 tips 데이터셋을 불러와 살펴봅니다.

```
> install.packages('reshape2')
> library(reshape2)
> str(tips)
'data.frame' :244 obs. of 7 variables:
 $ total_bill : num 17 10.3 21 23.7 24.6 ...
 $ tip        : num 1.01 1.66 3.5 3.31 3.61 4.71 2 3.12 1.96 3.23 ...
 $ sex        : Factor w/ 2 levels "Female","Male": 1 2 2 2 1 2 2 2 2 2 ...
 $ smoker     : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
 $ day        : Factor w/ 4 levels "Fri","Sat","Sun",...: 3 3 3 3 3 3 3 3 3 3 ...
 $ time       : Factor w/ 2 levels "Dinner","Lunch": 1 1 1 1 1 1 1 1 1 1 ...
 $ size       : int 2 3 3 2 4 4 2 4 2 2 ...
```

# 실전분석. 종업원의 팁 분석하기

## 2. 먼저 성별에 따라 팁을 주는 정도를 비교합니다.

```
> unique(tips$sex)
[1] Female Male
Levels: Female Male
>
> idx <- which(tips[, 'sex'] == 'Female')
> avg.female <- mean(tips[idx, 'tip'])
>
> idx <- which(tips[, 'sex'] == 'Male')
> avg.male <- mean(tips[idx, 'tip'])
>
> avg.female
[1] 2.833448
> avg.male
[1] 3.08961
```

## 실전분석. 종업원의 팁 분석하기

3. 그 다음 흡연 유무에 따라 팁을 주는 정도를 비교합니다.

```
> unique(tips$smoker)
[1] No Yes
Levels: No Yes
>
> idx <- which(tips$smoker == 'Yes')
> avg.smoker <- mean(tips[idx, 'tip'])
>
> idx <- which(tips$smoker == 'No')
> avg.nonsmoker <- mean(tips[idx, 'tip'])
>
> avg.smoker
[1] 3.00871
> avg.nonsmoker
[1] 2.991854
```

## 실전분석. 종업원의 팁 분석하기

4. 2번과 3번의 코드를 자세히 살펴보면 열 이름과 열의 값의 종류만 바뀔 뿐 코드에서 공통된 구조가 반복되는 것을 알 수 있습니다. 따라서 동일한 방법으로 모든 열에 대해서 반복하는 대신 데이터셋(tips)과 열 이름(colname)만 입력하고 열의 값에 대해 종류별로 평균을 구해 결과를 돌려주는 함수를 작성하면 코드의 중복을 줄일 수 있습니다.

```
meanbycol.tip <- function(tips, colname){ }
```

# 실전분석. 종업원의 팁 분석하기

## 5. 코드블록 안에 다음 절차를 수행하도록 코드를 작성합니다.

- ① 값의 종류를 구한다.
- ② 값의 종류별로 평균을 구한다.
  - ②-1 하나의 값에 해당하는 인덱스를 구한다.
  - ②-2 인덱스로 해당 값의 평균을 구해 저장한다.
- ③ 결과값에 값의 종류로 이름을 붙인다.
- ④ 결과값을 반환한다.
- ⑤ 작성을 마치면 스크립트 파일을 'myfunc.R'로 저장한다.

```
meanbycol.tip <- function(colname) {  
  value <- unique(tips[, colname])      # ① 값의 종류를 구함  
  result <- list()  
  for(i in 1:length(value)) {           # ② 값의 종류별로 평균을 구함  
    idx <- which(tips[,colname] == value[i]) # ②-1  
    result[i] <- mean(tips[idx,'tip']) # ②-2  
  }  
  names(result) <- value                # ③ 결과값에 이름을 붙임  
  return(result)                       # ④ list 결과값 반환
```

## 실전분석. 종업원의 팁 분석하기

### 6. 함수 작성 이전 스크립트로 돌아와 함수가 작성되어 있는 myfunc.R 파일을 로드합니다

```
> source('myfunc.R')
> meanbycol.tip('sex')
$Female
[1] 2.833448

$Male
[1] 3.089618

> meanbycol.tip('smoker')
$No
[1] 2.991854

$Yes
[1] 3.00871
```



## 실전분석. 종업원의 팁 분석하기

7. 나머지 열 이름도 입력하여 함수가 잘 작동하는지 확인합니다.

```
meanbycol.tip('size')  
meanbycol.tip('day')
```

8. 지금까지의 결과를 간단히 살펴보면 평균적으로 남자가 여자보다 팁을 더 많이 주며, 일행 중 흡연자의 포함 여부는 큰 차이가 없다는 것을 알 수 있습니다. 또한 일행이 많아질수록 받는 팁도 늘어났으며 일요일에 가장 많은 팁을 받았다는 것을 알 수 있습니다.

# 실전분석. 종업원의 팁 분석하기

## 9. 그렇다면 결제 금액 대비 받은 팁의 비율은 어떻게 달라질까요?

```
category.tip <- function(tips){  
  tip_ratio <- tips$tip/tips$total_bill * 100  
  
  class <- c()  
  
  for(i in 1:nrow(tips)){  
    if(tip_ratio[i] < 10){  
      class[i] <- 1  
    }else if(tip_ratio[i] < 15){  
      class[i] <- 2  
    }else if(tip_ratio[i] < 20){  
      class[i] <- 3  
    }else{  
      class[i] <- 4  
    }  
  }  
  tips.new <- cbind(tips, type = class, ratio = tip_ratio)  
  return(tips.new)  
}
```

## 실전분석. 종업원의 팁 분석하기

10. 'myfunc.R'을 다시 로드한 후 `categorize.tip` 함수를 실행합니다.

```
> source('myfunc.R')
> tips.new <- categorize.tip(tips)
> head(tips.new)
```

	total_bill	tip	sex	smoker	day	time	size	type	ratio
1	16.99	1.01	Female	No	Sun	Dinner	2	1	5.944673
2	10.34	1.66	Male	No	Sun	Dinner	3	3	16.054159
3	21.01	3.50	Male	No	Sun	Dinner	3	3	16.658734
4	23.68	3.31	Male	No	Sun	Dinner	2	2	13.978041
5	24.59	3.61	Female	No	Sun	Dinner	4	2	14.680765
6	25.29	4.71	Male	No	Sun	Dinner	4	3	18.623962

## 실전분석. 종업원의 팁 분석하기

11. 마지막으로 열 type의 값에 따라 결제 금액(total\_bill), 팁(tip), 결제 금액 대비 팁의 비율(ratio)에 대한 평균값을 계산합니다.

```
> res <- c()
> for(i in 1:4){
+   idx <- which(tips.new[, 'type'] == i)
+   tips.tmp <- tips.new[idx, ]
+   res.tmp <- apply(tips.tmp[c('type', 'total_bill', 'tip', 'ratio')], 2, mean)
+   res <- rbind(res, res.tmp)
+ }
> rownames(res) <- 1:4
> res
```

	type	total_bill	tip	ratio
1	1	26.95889	2.074444	7.849503
2	2	21.22691	2.725432	12.999096
3	3	18.15742	3.178866	17.436573
4	4	15.87769	3.755385	24.804411

# Thank You !