실험계획법 실습

- If & Loop
- Function
- Avoiding Loops
- Debug

◆ if-else & Loop

1. if-else의 형태

```
  if( 조건 ){

  참일때 실행

} else {

  거짓일때 실행

}

# R의 기본 연산자
# x&y (불리언 AND)
# x|y   (불리언 OR)
# !x    (불리언 부정)
# x==y (동일인지 판단)
```

```
Ex )
 x<-c(T,F,T)
 y<-c(T,T,F)

 x&y
 x|y

 if(x[1] &&y[1]) {
 print("both True")
}
 g<-c("m", "f" ,"f","m")
 ifelse(g=="m",1,0)
```

◆ if-else & Loop

2. For문

for (n in x) {
 	
변수 n 이 x의 조건을 만족할 때까지 괄호 안의 명령어를 실행
 	
}
	Ex)
	
	x<- c(1,2,3,4,5,6,7,8,9,10)
	sum<-0
	for(i in 1:length(x)){
	
	sum<-sum+x[i]
	
	}

◆ if-else & Loop

3. While문 , repeat문

while (조건) {
 	
조건이 만족할 때까지 괄호 안의 내용을 실행
(break문을 사용하여 중단 가능)
 	
}
 	
repeat {
 	
괄호 안의 내용을 실행 (break문을 사용하여 중단)
 	
}
	Ex)
	i<-1
	while(i<=10) {
	i<- i+4
	}
	while(T) {
	i<- i+4
	if(i>10) {
	break
	}
	}
	repeat {
	i<- i+4
	if(i>10) {
	break
	}
	}

◆ Function

1. 함수의 형태

함수이름 <-function(인자) {
 	
  함수 내용
 	
return (k)
 	
}
# return을 사용하지 않을 때는 마지막으로 계산된 값을 반환한다. 	
	Ex)
	fun1<- function(a, b) {
	a*b
	
	}
	# x 중 홀수의 개수 세기
	oddcount<- function(x) {
	k<-0
	for(n in x){
	if(n%%2==1){
	k<-k+1
	}
	}
	return (k)
	}

◆ Function

2. New binary operators

함수를 이용하여 새로운 binary operators를 생성할 수도 있음.	Ex) x<-rnorm(10) y<-rnorm(10) "%!%" <- function(x, y) { mean(c(x,y))} x%!%y

◆ Avoiding Loops

1. lapply

lapply(li, fct) 리스트 li를 function fct에 실행시킨 후 리스트로 반환	Ex) li = list("klaus","martin","georg") lapply(li, toupper)

◆ Avoiding Loops

2. sapply

sapply(li, fct) 리스트 li를 function fct에 실행 시킨 후 결과를 벡터 또는 행렬로 단순화하려고 한다. 반환된 모든 값이 동일한 길이일 때 반환된 길이가 1이라면 벡터가 나오고, 그렇지 않은 경우엔 행렬이 나온다. 길이가 서로 다른 경우라면 단순화 할 수가 없어서 리스트가 생성된다.	Ex) li <-list("klaus","martin","georg") sapply(li, toupper) Fct <- function(x) { return(c(x, x*x, x*x*x)) } sapply(1:5, fct)

◆ Avoiding Loops

3. apply

apply(mat,1, fct) 행렬 또는 데이터 프레임 mat를 받고 두 번째 인자가 1이면 행별로 2이면 열별로 fct 함수를 실행시킨다.	Ex) x <- matrix(c(5,7,4,6,7,9,6,3,0,8,7,5),nrow=4,ncol=3) apply(x, 1, sum) apply(x, 2, sum)

◆ Avoiding Loops

4. tapply

tapply(data, Index, function ..) data를 index에 지정한 factor 값으로 분류(그룹화)하여 매개변수로 넘어온 function을 적용하는 함수다. 이는 index에 넘어온 level에 대해 그룹화 하는데, sql의 group by와 유사한 기능을 가진다.	Ex) tapply(patients$weight, patients$gender, mean, na.rm=TRUE)

◆ Debug

1. debug()

debug(function) debug안에 함수의 이름을 넣고 실행을 시키면 함수가 debugging 되기 시작한다. undebug(function) debug가 끝났을 때에는 undebug함수를 실행시켜 debug가 끝났음을 컴퓨터에 알려줘야 한다. browser()나 setBreakpoint() 함수를 사용하여 중단점을 설정할 수 있다.	Debug 명령어 n(next) : R이 다음줄을 실행한 후 멈추도록 한다. Enter 키를 처도 동일하게 동작한다. c(continue) : n과 비슷하지만, 다음 멈출 때까지 여러 줄의 코드가 실행 될 수 도 있다. where : 추적값을 출력한다. 현재 위치에서 실행된 함수 호출 수를 보여준다. Q: 브라우저를 종료하고 R의 기본 인터랙티브 창으로 다시 이동한다. R 명령어 : 브라우저 안에서도 R의 인터랙티브 창에 그대로 있는 것으로 x를 입력해 x의 값을 확인하는 등의 작업이 가능하다.

◆ Debug

2. browser()

browser() browser 함수는 원래 중단점(breakpoint)의 역할로 사용되는 함수이나 인터랙티브 창에서 for문을 debug 하고 싶을 때 역시 사용할 수 있음. browser(조건) 조건이 True일 시에만 brower()함수 호출	Ex) browser(s>1) s가 1보다 큰 경우에 browser가 호출

◆ Debug

3. traceback()

디버거를 돌리지 않고 있는데 R 코드에서 충돌이 났을때 traceback()을 호출해 post-mortem(프로그래밍이나 서비스 시행 후 회고를 의미)을 할 수 있다. 이를 통해 어떤 함수에서 문제가 발생했으며, 이 함수까지 어떤 경로로 오게 됐는지 알 수 있다.	Ex)
	f <- function(x) { r <- x - g(x);r}
	g <- function(y) { r <- y * h(y);r}
	h <- function(z) {r <- log(z);
	if (r < 10)
	r^2 else r^3}
	f(-1)
	traceback()

감사합니다.
See You