180511

3장 R 프로그래밍

01 R의 특징

- 벡터연산 / 데이터 각각이 아닌, 벡터 차원에서의 연산 가능
- 결측치(NA) / 항상 먼저 확인해본 후 처리
- 객체의 불변성

02 흐름제어 (조건문과 반복문)

조건문과 반복문에서 사용 가능한 비교 연산자

기호	의미
==	같다 (**두 개 사용** 주의)
!=	같지 않다
>=	크거나 같다
>	크다
<=	작거나 같다
<	작다

조건문과 반복문에서 사용 가능한 논리 연산자

기호	의미
!	~~가 아니다 (NOT)
&&	AND (두 번 쓰는 것 권장)
||	Or (두 번 쓰기!)

| : 요소별로 T/F 리턴

|| : 조건에 따른 단 하나의 T / F 리턴 -> in 조건문, 반복문

cf. 여러 요소들간의 비교 -> 한 가지 결과 return 하려면

== - 사용하지 않는다. why? 요소별 TRUE, FALSE값 모두 뜸

identical() - [요소 값, 요소 개수 일치]

setequal() - [요소 값 일치]

(1) IF문 : 조건문

1 if문

if (cond) {

cond가 참일 때 실행할 문장

} else {

cond가 거짓일 때 실행할 문장

}

---

if (조건식1) {

조건식1일 때 실행되는

}

else if (조건식2 {

조건식 2일 때 실행되는

}

else {

}

cf. 출력의 목적 - print, cat 함수 사용

```
> if(TRUE) {
+   print("TRUE")
+   print("hello")
+ } else {
+   print("FALSE")
+   print("world")
+ }
[1] "TRUE"
[1] "hello"
```

2 ifelse () 함수

ifelse (test, yes, no) **cf. 오라클 decode와 비슷**

- return만 됨 (출력목적) / 벡터형태의 연산 가능

```
> x <- c(1,2,3,4,5)
> ifelse(x %% 2 == 0, "even", "odd")
[1] "odd"  "even" "odd"  "even" "odd"
```

---

if ; 변경문

ifelse : 결과를 나타낼 목적(출력) in R 장점!

실습

1. 다음과 같은 데이터 프레임을 만들고 grade 칼럼을 새로 추가하여라. (3000이상 A, 2000이상 3000미만 B, 나머지 C)

```
emp <- data.frame(Name=c("홍길동", "김길동", "최길동", "박길동")
,pay=c(3000, 2000, 2500, 1000), stringsAsFactors = F)
emp$grade <- ifelse (emp$pay >= 3000, "A", ifelse(emp$pay >= 2000, "B", "C"))
```

emp[,2] 보다는 emp$pay 형태로 살펴보자~~~~

if : 벡터형식 연산 X, for문으로 만들어야 함
ifelse : 결과를 나타낼 목적(출력) in R 장점!

stringsAsFactors = F 잘 쓰기!

(2) 반복문 : for, while, repeat

1 for문 - 반복 횟수를 아는 경우

for문 (반복변수 in data~횟수) {

반복할 식

}

-벡터 값을 횟수에 넣고 치환하는 for문 (값 삽입) ~ append / 기존벡터에 기존벡터 추가

-> 벡터에 원소 추가하는 방법 중요

2 while문 - 조건이 참인 한 계속해서 반복됨 (광범위함)

while (조건) {

반복할 식

}

* for문과 다르게 초기변수 설정 필요.

* i는 증가시키지 않으면 계속 초기값. 따라서 기존의 i에 덮어쓰는 명령 필요.

3 repeat 문

- if 를 안으로 넣어야 함 그런데 무한반복임

------> 해당 조건을 만났을 때에 종료하도록 break 사용

---

next : 반복문 중단 (p99)

```
> for (i in 1:10) {
+   if (i %% 2 !=0){
+     next                                  # 반복문의 맨 위로 올라감
+   }
+   print(i)
+ }
```

break : 반복문을 종료한다

```
> i<-1
> repeat {
+   print(i)
+   if (i>=9) {
+     break
+   }
+   i <- i+1
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
```

실습 for문, while문 차이점 살펴보기
Q 1~100 더하기?

규칙 살펴보기
i
1 1
2 1+2
3 1+2+3
4 1+2+3+4
5 1+2+3+4+5

for문

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

while문

```
i<-1                #초기화
sum<-0              #초기화

while (i<=100) {
sum <- sum + i
   i<- i+1
}
print(sum)
```

cf. 한 줄에 쓸 때 여러 연산 표시는 세미콜론(;) 사용

```
i<-1
sum<-0
while (i<=100) { sum2 <- sum2 + i  ;  i<- e+1 }
print(sum)
```

같은 세션에서 같은 변수 이름을 쓰려면 초기화를 시켜야 함
아니면 다른 변수 이름을 사용

---

cf. 벡터에 삽입하는 for문 중요

파이썬 - 벡터 연산 불가능 ~ for문으로

```
a <- c(1,2,3,4,5)
aa<-c()                    #값을 갖지 않는 벡터를 생성

for (name in a) {
   aa <- c(aa, name + 4)                    #not 출력
}
```

벡터의 확장이 일어남. 계속 반복적으로 특정 벡터에 새로운 값을 추가하는 형식이 완성.
나중에 색깔벡터에 name + 4부분에 색으로 해서 !

---

cf. 여태까지 i 1:100 까지 추이를 벡터로 저장하려면

while문

```
i<-1
bb <-c()
while (i <=100) {
   bb<-c(bb,i) # append(bb,i)
   i<-i+1
}
```

for문

```
rm(i) # i 초기화
cc<-c()
for(i in 1:100){
   cc<-c(cc,i)              # append(cc,i)
}
cc
```

Q 1~100 더하는 중에 더한 값들의 추이를 데이터프레임으로 저장하려면? for 그래프그리기

for문

```
a <- c()
b <- c()
sum <- 0
for (i in 1:100) {
   sum <- sum+ i
   a <- append(a,i)
   b <- c(b,sum)
}
data.frame(i = a, total = b)
```

while문

```
c <- c()
d <- c()
sum <- 0
i <- 1
while (i <= 100) {
   sum = sum + i
   c <- c(c,i)
   d <- append(d,sum)
   i = i+1
}

data.frame(i = c, total = d)
```

정리

벡터의 요소 추가

a <- c(a,1) # a벡터에 1의 값을 맨 뒤에 추가

a <- append(a,1)

+ sum 값이 100이 넘지 않을 때 print 하고 싶은데 어떠케하지

집에서 찾아보기

03 연산

(1) 수치연산

연산자와 함수	의미
n %/% m	n 을 m으로 나눈 몫
n %% m	n 을 m으로 나눈 나머지
Exp(n)	E의 n승
Log(x,base=exp(1))	만일 base가 지정되지 않으면 자연로그(x)계산

(2) 벡터연산 - R의 특징

: 벡터 또는 리스트를 한 번에 연산하는 것

```
> x<-1:5

> x+1
[1] 2 3 4 5 6

> x+x
[1]  2  4  6  8 10

----------------------------------------
#벡터끼리 요소별 비교
# =가 아닌, ==사용
> x == x
[1] TRUE TRUE TRUE TRUE TRUE

> x == c(1,2,3,5,4)
[1]  TRUE  TRUE  TRUE FALSE FALSE

# & ->요소간 비교로 T/F 만 리턴-> 진릿값을 가지고 색인하는 불리언 인덱싱(색인) 사용 가능
> c(T, T, T) & c(T, F, T)
[1]  TRUE FALSE  TRUE
```

- 불리언 인덱싱(색인)

```
> (d<- data.frame(x=1:5,y=c("a","b","c","d","e")))
  x y
1 1 a
2 2 b
3 3 c
4 4 d
5 5 e

> d[c(TRUE,FALSE,TRUE,FALSE,TRUE),] # 불리언 색인
  x y
1 1 a
3 3 c
5 5 e

> d[d$x %% 2 == 0,]       # 불리언 색인 활용
  x y
2 2 b
4 4 d
```

```
------------------------------------------------
# ifelse 역시 벡터 연산이 가능한 함수

> ifelse(x %% 2 == 0, "even", "odd")
[1] "odd"  "even" "odd"  "even" "odd"
------------------------------------------------
```

(3) NA의 처리

- NA : 결측치
- **na.rm** : NA값을 연산에서 제외할 것인지 지정하는 인자 (많은 함수에서 사용됨

```
> sum(c(1, 2, 3, 4, NA))
[1] NA

> sum(c(1, 2, 3, 4, NA), na.rm=TRUE)
[1] 10
```

- NA 처리 함수

na.fail()	NA가 포함되어있으면 실패한다.
na.omit()	NA가 포함되어있으면 이를 제외한다.
na.exclude()	NA가 포함되어있으면 제외한다. (na.rm과 비슷) 그러나 naresid, napredict를 사용하는 함수에서 NA로 제외한 행을 결과에 다시 추가한다는 점이 다르다.
na.pass()	NA가 포함되어있어도 통과시킨다.

```
> (x<-data.frame(a=c(1,2,3),b=c("a",NA,"c"),c=c("a","b",NA)))
  a    b    c
1 1    a    a
2 2 <NA>    b
3 3    c <NA>

> na.fail(x)
Error in na.fail.default(x) : 객체안에 결측값들이 있습니다

> na.omit(x)
  a b c
1 1 a a

> na.exclude(x)
  a b c
1 1 a a

> na.pass(x)
  a    b    c
1 1    a    a
2 2 <NA>    b
3 3    c <NA>
```

04 함수의 정의

사용자 정의 함수 : 코드의 반복을 줄이거나 가독성을 높이기 위한 사용자 정의 함수

기본 정의

function_name <- function(인자, 인자, ...){

함수 본문

return(반환 값 # 반환 값이 없다면 생략

}

- 정의 된 이후 호출할 때만 수행됨
- 인자는 위치에 맞게 순서대로 전달하거나 인자의 이름으로 전달 가능

```
#함수의 정의 - sign 함수 만들어보기
> f_sign <- function(x) {
+  if (x > 0) {
+    return(1)
+    }
+  else if (x == 0) {
+     return(0)
+   }
+   else { return (-1) }
+ }

> f_sign(23894)
[1] 1
```

#피보나치수열 만들기

f(n) = 1 (n <=2일 때)

f(n) = f(n-2)+f(n-1) (n>2일 때)

```
> fibo <- function(n) {
+   if (n==1 || n==2) {
+     return(1)
+   }
+  return(fibo(n-1) + fibo(n-2))
+ }
```

- if (n<=2) 로 써도 됨
- 대체적으로는 if문에 **두개씩** 연산자 쓰기
- if, else만 있으면 else+{} 생략할 수 있음
- 사용자함수를 정의할 때 사용자함수를 스스로 호출할 수 있음
  -> 중첩함수(재귀함수)

주의

return을 쓰지 않으면 잘못 해석됨

/ return 생략이 가능하긴 하지만, 마지막 문장의 반환 값이 최종 함수의 반환 값이 됨

연습문제

if 연습문제 1.

입력되는 인수가 5보다 클 때는 1을 출력하고 5 보다 작거나 같으면 무조건 0을 출력하는 myf1( ) 을 생성하세요.

```
> myf1 <- function(m){
+   if(m>5) {
+     return(1)
+   }
+   else {
+     return(0)
+   }
+ }
```

if 연습문제 2.

입력되는 인수가 양수일 때는 1을 출력하고 음수일 때는 늘 0 을 출력하는 myf2( ) 를 생성하세요.

```
> myf2 <- function(x) {
+  if (x > 0) {
+    return(1)
+    }
+  else if (x == 0) {
+     return(0)
+   }
+   else { return (-1) }
+ }
```

if 연습문제 3.

두 숫자를 입력해서 첫 번째 숫자가 두 번째 숫자보다 클 경우 첫 번째 숫자에서 두 번째 숫자를 뺀 값을 출력하고 두번째 숫자가 첫번째 숫자보다 클 경우 두번째 숫자에서 첫번째 숫자를 뺀 값을 출력하는 함수 myf3( ) 을 작성하세요

```
> myf3 <- function(m,n){
+   if(m>n) {
+     return(m-n)
+   }
+   else {
+     return(n-m)
+   }
+ }
```

if 연습문제 4.
입력값이 0보다 작을 경우 0으로 출력하고 1-5 사이의 값일 경우 1 을 출력하고 5 이상의 값이 입력될 경우 10을 출력하는 함수 myf4( ) 를 생성하세요.

```
myf4 <- function (m) {
  if(m>=5) {
    return(10)
  }
  else if (m>=1) {
    return(1)
  }
  else {
    return(0)
  }
}
```

if 연습문제 5.
사용자가 대문자 'Y'나 소문자'y'를 입력하면 화면에'Yes'를 출력하고 그 외 다른 글자를 입력하면 'Not Yes'를 출력하는 사용자 정의 함수를 만드세요.(함수명과 변수명은 자유롭게~)

```
myf5 <- function (m){
  if(m == "Y" || m == "y"){
    return("Yes")
  }
  else {
    return("Not Yes")
  }
}
```

if 연습문제 6.
두 개의 숫자를 입력 받은 후 두 값이 모두 양수일 경우 두 수의 곱을 출력하고
두 값 중 하나라도 0이나 음수일 경우는 두 수의 합을 출력하세요.

```
> myf6 <- function(m,n) {
+   if (m > 0 && n > 0){
+     return(m*n)
+   }
+   else {
+     return(m+n)
+   }
+ }
```

반복문 연습문제 1.
사용자에게 정수 n 을 입력 받은 후 1 부터 n 까지의 합을 출력하기

```
sum <- 0
myf6 <- function(x) {
   for (i in 1 : x ) {
     sum <- sum + i
 }
 return(sum)
}
```

```
f_sum <- function(m){
   return(sum(1:m))
}
```

----------또는

```
f_sum2 <- function(m){
   if(m==1) {
      return(1)
   }
   else {return(m+f_sum2(m-1))}
}
```

반복문 연습문제 2. / 재귀함수

팩토리얼 계산

반복문

```
fac<-1
f__fac<-function(m){
 for (i in 1:m){
 fac <- fac * i
   }
   return(fac)
}
```

재귀함수

```
f_fac <- function(m){
  if (m==1) {
    return(1)
  }
  else {
    return(m*f_fac(m-1))
  }
}
```

반복문 ex. 1~10까지

```
fac <- 1
for (i in 1:10){
  fac <- fac * i
}
print(fac)
```

180514

04)함수의 정의

'...' : 가변 길이 인자

- 함수 인자 목록에 '...' 표시
- 함수 인자 개수가 정해지지 않은 경우

```
> f <- function(...){
+   args <- list(...)
+   for (a in args){
+     print(a)
+   }
+ }
> f('3','4')
[1] "3"
[1] "4"
```

ex. 사용자가 입력한 값을 모두 더해라

```
> sum<-0
> f <- function(...){
+   args <- list(...)
+   for (a in args){
+     sum <- sum + a
+   }
+   return(sum)
+ }

> f(3,4)
[1] 7
> f(3,4,45,1)
[1] 53
```

```
> f <- function(x,y){
+   print(x)
+   print(y)
+ }
> g <- function(z, ...){
+   print(z)
+   f(...)
+ }

# g 함수에 '...' 있지만 사실은 세 개의 인자만 입력 가능!
 가변형일지라도 다른 함수 관계 연결에 의해 제한될 수 있는 예시


> g(1,2,3)
[1] 1
[1] 2
```

중첩 함수

정의 속에 또 다른 함수 정의 가능. 사용도 낮음

```
> f <- function(x,y){
+   print(x)
+   g<-function(y){
+     print(y)
+   }
+   g(y)
+ }
> f(1,2)
[1] 1
[1] 2
> f(1,3)
[1] 1
[1] 3
> f<- function(x1){
+   return(function(x2){
+     return(x1+x2)
+   })
+ }

> g<-f(1)
> g(2) # x1=1, x2=2
[1] 3

> g2<-f(2)
> g2(3) # x1=2, x2=3
[1] 5
```

?

print와 return의 차이점

DEFAULT 값 -> 함수 정의단계에서 '='를 사용해서 씀

ex. myf3 <- function(x, y=3) {

스코프

- 코드에 사용한 이름의 사용 범위를 지정하는 규칙
- 문법적 스코프를 주로 사용
- 변수 선언시 콘솔에서 전역변수로 생성
- source()사용해서 다른 파일에서 해당 변수 사용 가능

전역 변수와 지역 변수

전역 변수 -> 모든 블록 내에서 사용 가능한 변수 실행 중인 현재 세션에서만 유효
(cf. in python, 글로벌 예약어)
지역 변수 -> 변수가 정의된 블록 내부에서만 변수를 접근할 수 있는 규칙

```
> n <- 1
> f <- function() {
+   print(n)
+ }
> f()
[1] 1
> n<-2
> f()
[1] 2
```

```
# 인자를 입력하지 않는 함수. 무조건 정해진 값을 출력하는 형태도 가능함
```

```
> n<-100
> f<-function(){
+   n<-1
+   print(n)
+ }
```

```
# 함수 실행마다 초기화됨! 초기화를 위한 내부에 n 값 정의
```

변수는 내부 블록에서만 접근할 수 있으므로 함수 내부에 정의한 이름은 함수 바깥에서 접근할 수 없다.

```
> rm(list=ls())
> f<-function(){
+   n<-1
+   print(n)
+ }
> f()
[1] 1
> n
Error: object 'n' not found
```