

# R과 Python

---

2021.09.28

# R과 Python\_조건문(if, else)

## [R]의 기본구조

if (조건) {★종괄호로 안에 실행할 명령어 삽입★  
실행할 명령어  
}

```
R 4.1.1 · ~/
> a <- 3
> if (x %% 2 == 2) {
+   print('x는 짝수입니다.')
+ } else if (x %% 2 == 1) {
+   print('x는 홀수입니다.')
+ } else {
+   print('x는 자연수가 아닙니다.')
+ }
[1] "x는 홀수입니다."
```

# 조건에 맞는 논리값이 출력

else는, 조건을 확장시키는 역할을 하게 됩니다.  
첫번째 조건이 만족하지 않으면 if문 밖으로 나오게 되고 이후  
else if를 수행  
해당 조건도 만족하지 않으면 else 수행

## [Python]의 기본구조

if (조건) :  
[실행할 명령어] ★들여 쓰기 주의! ★

```
# 1개 조건
coffee = 4500

if coffee > 4000: # coffee가 4000원 보다 큰 경우 GOOD
    print("Good")


# 2개 이상의 조건
x = 4500

def it_test2(x):
    if x % 2 == 0: # 나누기 연산 후 몫이 아닌 나머지를 구함
        print('x는 짝수입니다.')
    elif x % 2 == 1:
        print('x는 홀수입니다.')
    else:
        print("x는 자연수가 아닙니다.")
```

# R과 Python\_for 반복문

## [R]의 기본구조

```
for (변수 in 벡터) {  
  실행할 명령어  
}
```

```
R 4.1.1 · ~/   
> mysum <- 0 # sum의 초기값 지정  
> for (i in 1:10) { # 1~10  
+   mysum <- mysum + x**2 #1~10까지의 거듭제곱  
+   print(mysum)  
+ }  
[1] 1 4 25  
[1] 2 8 50  
[1] 3 12 75  
[1] 4 16 100  
[1] 5 20 125  
[1] 6 24 150  
[1] 7 28 175  
[1] 8 32 200  
[1] 9 36 225  
[1] 10 40 250
```

# [반복 객체] 순서대로 반복문을 수행

## [Python]의 기본구조

```
for [변수] in [반복객체]:  
  [실행할 명령어]
```


```
mysum = 0  
  
for i in range(1, 11): # [1,2,3,4,5,6,7,8,9,10]와 같은 말  
    mysum = mysum + i ** 2 # 1~10까지의 제곱  
else:  
    print(mysum) # ans: 385
```

# for문은 반복 가능한 리스트, 벡터 등 iterable한 객체 이용  
# for와 range 함께 사용  
# range함수는 연속된 숫자들을 담고 있는 객체를 반환  
# [반복 객체]의 원소의 개수만큼 반복문을 수행

# R과 Python\_while 반복문

## 기본구조

```
while(조건) {  
    조건이 참일 때 수행할 명령어  
}
```

```
R 4.1.1 · ~/   
> x <- 1 # 할당될 변수를 지정  
> while(x <=5){ # x가 5보다 작은 경우 문장 수행  
+   print(x)  
+   x <- x+1 # x 값을 +1 하여 새로운 값으로 갱신  
+ }  
[1] 1  
[1] 2  
[1] 3  
[1] 4  
[1] 5
```

```
# 무한루프  
x <- 1 # 할당될 변수를 지정  
while(x <=5){ # x가 5보다 작은 경우 문장 수행  
  print(x)  
  # x <- x+1  
}  
# x <- x+1 해당 조건을 적지 않으면 조건문이 항상 참이 되기 때문에  
# 무한루프에 빠지면서 1을 계속 찍어냄  
.
```

# while문은 괄호 안에 있는 조건이 참일 경우에 수행 코드 문장을 수행  
# for문과 다른 점으로는 반복 + 조건이 들어간 형태

## [Python]의 기본구조

```
while [조건문]:  
    [실행할 명령어]
```

```
x = 1  
sum = 0  
while x <= 100: # 콜론으로 구분  
    sum = sum + x ** 2  
    x = x + 1  
else: # 필요없이 print(sum)만 입력해도 값은 동일  
    print(sum) # ans: 38350
```

# while문은 작성한 조건이 참인 동안에는 계속 반복문을 수행하는 방식  
# 거짓인 경우에는 while문을 빠져나감으로써 반복문을 종료

# R과 Python\_반복문에서 나오기

반복문 안에서 'break' 함수를 만나면 해당 반복문을 중지하고 빠져 나옴

```
R 4.1.1 · ~/
> for(i in 1:10){
+   if(i == 3){ #i의 값이 3이면 반복문을 멈춤
+     break
+   } # if 조건 end
+   print(i)
+ } #for문 end
[1] 1
[1] 2
```

'next'는 조건에 맞는 것이 나오면 수행 중인 반복을 중지하고 다음 반복으로 넘어가는 함수

```
> for(i in 1:10){
+   if(i == 3){ #i의 값이 3이면 반복문을 멈춤
+     next # next 함수를 사용하여 다음 반복을 실행
+   } # if 조건 end
+   print(i)
+ } #for문 end
[1] 1
[1] 2
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

for문을 나오고 싶다면 'break' 사용

```
x = range(1, 6)
for i in x:
    if i == 3: break # x의 값이 3이후부터는 반복 종료
    print(i, "")
# ans: 1
# ans: 2
```


for문 내의 코드를 끝까지 수행하지 않고 곧바로 다음 반복으로 넘어가고 싶다면 'continue' 사용

```
x = range(1, 6)
for i in x:
    if i == 3: continue # x의 값이 3이면 수행하지 않고 다음 반복으로 넘어감
    print(i, "")
# ans: 1
# ans: 2
# ans: 4
# ans: 5
```

# R과 Python\_함수의 작성

## [R]의 기본구조

```
함수명<- function(매개변수1, 매개변수2, ...){  
  실행할 명령어  
  실행할 명령어  
  return (반환값) #함수 실행결과를 받을 때 사용  
}
```

```
R 4.1.1 · ~/   
> sum2 <- function(x,y) { #매개변수가 있는 함수  
+   sum2 <- x  
+   if (y > x) {  
+     sum2 <- y  
+   }  
+   return(sum2)  
+ }  
> sum2(10,15) #function(x,y)  
[1] 15
```

# 보통 출력 값을 지정하는 print나 cat을 사용하는데 return을 사용하는 이유로는  
print와 cat은 반환 값을 주지 않으며, 연산결과를 출력하고 끝나기 때문에  
return 사용해서 실행한 결과값을 새로운 변수로 지정해서 계속 실행 가능

## [Python]의 기본구조

```
def 함수명(매개변수1, 매개변수2, ...):  
  [실행할 명령어]  
  [실행할 명령어]  
  return (반환값) #함수 실행결과를 받을 때 사용
```

```
def sum2(x, y):  
    mysum = x + y  
    return (mysum)  
  
a = int(input("a=4")); b = int(input("b=7"))  
print(a, "와", b, "의 합: ", sum2(a, b))  
# ans: 4와 7의 합은 11
```

# Thanks😊

---