R 이해하기

- #### update : 2019. 11.
- #### Written by: YunGil Jun

I. R 이해하기

> ## 1. 데이터 타입
>
> ## 2. 데이터 구조
>
> ## 3. 선언문(Statements)

1. 데이터 타입(Data Type)

> ### 1.1 수치형
>
> ### 1.2 문자형
>
> ### 1.3 Bool

1.1 수치형 데이터

In [3]:

```
# 숫자(numeric) : 실수
1.3
```

1.3

In [2]:

```
# 숫자 : 정수(integer)
3
3L
```

3

3

In [3]:

```
# 숫자 : 복소수(complex)
2i
```

0+2i

계산 연산자 및 함수

Operator	Description	Operator	Description
+	덧셈	sum	합
-	뺄셈	max	최대값
*	곱셈	min	최소값
/	나눗셈	mean	평균
%/%	몫	var	분산
%%	나머지	sd	표준편차
^ or **	거듭제곱	median	중앙값
sqrt	제곱근	range	범위

In [4]:

```
1 + 2
```

3

In [5]:

```
3 - 7
```

-4

In [6]:

```
3 * 2
```

6

In [7]:

```
8/2
```

4

In [8]:

```
14 / 3
```

4.66666666666667

In [9]:

```
# 몫
14 %/% 3
```

4

In [10]:

```
# 나머지
14 %% 3
```

2

1.2 문자형 데이터

In [11]:

```
"문자"
```

'문자'

In [12]:

```
string_var <- "string"
string_var
```

'string'

In [13]:

```
# 문자에서 인자(factor)를 나타내는 데이터 타입은
# level의 종류중에 한가지로만 결정
# 통계에서 범주형 변수에 활용

x <- factor(c("남", "여", "여", "여", "남")) # 데이터가 두가지 범주로만 구분되어 있는 경우
x
levels(x)
```

남 여 여 여 남

▶ **Levels**:

'남' '여'

In [14]:

```
as.Date('2019-09-23')
```

2019-09-23

In [15]:

```
weekdays(as.Date('2019-09-23'))
```

'월요일'

In [16]:

```
Sys.time()
```

```
[1] "2019-08-31 11:45:22 KST"
```

In [6]:

```
# install.packages("lubridate")
library(lubridate)

hm("08:00")
```

8H 0M 0S

1.3 논리값 (bool)

In [3]:

```
class("TRUE")
class(TRUE)
```

'character'

'logical'

In [19]:

```
T
```

TRUE

In [22]:

```
TRUE == 1
```

TRUE

In [23]:

```
TRUE == 0
```

FALSE

In [24]:

```
FALSE == 0
```

TRUE

논리 연산자

Operator	Description	Operator	Description
<	작음	!=	같지 않음
<=	작거나 같음	!x	NOT x
>	큼	|	x OR y
>=	크거나 같음	x&y	x AND y

In [25]:

```
3 < 2
```

FALSE

In [2]:

```
3 == 3
```

TRUE

In [27]:

```
TRUE | FALSE
```

TRUE

In [28]:

```
TRUE & FALSE
```

FALSE

변수 할당

In [4]:

```
x <- 3.14
x
```

3.14

In [30]:

```
y <- "문자"
y
```

'문자'

In [31]:

```
v <- c(1, 2, 3, 4, 5)
```

In [32]:

```
a <- b <- 7
a
b
```

7

7

In [33]:

```
assign("z", 3)
z
```

3

In [34]:

```
rm(z)
```

2. 데이터 구조(Data Structure)

source: http://venus.ifca.unican.es/Rintro/dataStruct.html (http://venus.ifca.unican.es/Rintro/dataStruct.html)

2.1 vector

- R에서 데이터(Numerical, Character, Boolean를 다루는 가장 작은 단위
- 동일 타입의 스칼라가 여러개 모인 것

- 형태
  - 나의_벡터_변수명 <- 숫자 or 문자 or 논리값
  - 나의_벡터_변수명 <- c(숫자, 숫자, 숫자)

- Indexing (slicing) : 항목의 값에 접근 할때
  - 나의_벡터_변수명[2] : 벡터에서 2번째 값을 확인함

In [35]:

```
# R에서 가장 작은 데이터 구조
a <- 3
```

In [36]:

```
# 스칼라값이 들어가 있는 변수 a가 벡터인지 논리값을 물어보는 함수 : is.vector()
is.vector(a)
```

TRUE

In [37]:

```
# 벡터는 동일 데이터 타입임으로 논리값 문자(character)값으로 변경
c(T, "a")
```

'TRUE' 'a'

In [38]:

```
# 벡터는 동일 데이터 타입임으로 숫자값을 문자(character)값으로 변경
c("a", 1)
```

'a' '1'

In [39]:

```
# 숫자 벡터 생성
v_01 <- c(1, 2, 3, 4, 5)
v_01
```

1 2 3 4 5

In [40]:

```
v_02 <- 1:5
v_02
```

1 2 3 4 5

In [41]:

```
v_03 <- c(1:5)
v_03
```

1 2 3 4 5

In [42]:

```
# 문자값으로벡터 생성
v_04 <- c("AAA", "AA" , "BBB", "B")
v_04
```

'AAA' 'AA' 'BBB' 'B'

In [43]:

```
# 내장함수 통해서 벡터 생성

v_05 <- seq(from = 1, to = 3, by = 0.5)
v_05
```

1 1.5 2 2.5 3

In [44]:

```
# 매개변수(argument)는 순서대로 자동으로 함수에서 인식
# 필수 매개변수가 아닌경우 자동 초기 default값으로 진행

seq(1, 3, 0.5)
```

1 1.5 2 2.5 3

In [45]:

```
rep(1:3, times = 2)
```

1 2 3 1 2 3

In [46]:

```
rep(1:3, each = 3)
```

1 1 1 2 2 2 3 3 3

In [47]:

```
# indexing, slicing : 변수에서 특정 위치 값 확인/접근
x <- c( "E팀", "F팀", "A팀", "B팀", "C팀", "D팀")
x[2]
```

'F팀'

In [48]:

```
x[c(1, 3, 5)]
```

'E팀' 'A팀' 'C팀'

In [49]:

```
# 벡터에 논리연산자 활용 논리값 얻음
x == "E팀"

# 논리값이 TRUE인 값만 획득
x[x == "E팀"]
```

TRUE FALSE FALSE FALSE FALSE FALSE

'E팀'

In [50]:

```
x %in% c("G팀", "H팀", "A팀", "E팀")

x[x %in% c("G팀", "H팀", "A팀", "E팀")]
```

TRUE FALSE TRUE FALSE FALSE FALSE

'E팀' 'A팀'

In [51]:

```
x <- c( "E팀", "F팀", "A팀", "B팀", "C팀", "D팀")
```

In [52]:

```
sort(x, decreasing = T)
```

'F팀' 'E팀' 'D팀' 'C팀' 'B팀' 'A팀'

In [53]:

```
sort(x, decreasing = FALSE)
```

'A팀' 'B팀' 'C팀' 'D팀' 'E팀' 'F팀'

In [54]:

```
y <- c(2, 4, 8, 1, 4)
y
```

2 4 8 1 4

In [55]:

```
sort(y, decreasing = T)
```

8 4 4 2 1

In [56]:

```
rev(y)
```

4 1 8 4 2

In [7]:

```
# 미팅 참석 멤버이름
z <- c("김노랑님", "이빨강님", "이파랑님", '박회색님',"이파랑님", '박회색님
')
z
```

'김노랑님' '이빨강님' '이파랑님' '박회색님' '이파랑님' '박회색님'

In [8]:

```
# 분수에 있는 값이 중복되도 한번만 만듬

unique(z)  # 미팅의 멤버
```

'김노랑님' '이빨강님' '이파랑님' '박회색님'

In [9]:

```
# table 함수로 빈도수 확인

table(z) # 미팅 출석부
```

z
김노랑님 박회색님 이빨강님 이파랑님
1 2 1 2

In [ ]:

```

```

2.2 Matrix

- 행과 열로 구성된 2차원 데이터
- 한가지 데이터 타입으로 구성

- 형태
  - 나의_행렬_변수명 <- matrix(숫자 or 문자 or 논리값)
  - 나의_행렬_변수명 <- matrix(벡터값, nrow = 2, ncol = 3, byrow = FALSE)

- Indexing (slicing) : 항목의 값에 접근 할때
  - 나의_행렬_변수명[행, 열]
  - 나의_벡터_변수명[2, 1] : 벡터에서 2번째 행에서 1번째 열의 값

In [60]:

```
# 매트릭스 생성
m_01 <- matrix("T", ncol = 2, nrow = 3)
m_01
```

```
T T
T T
T T
```

In [61]:

```
# 데이터 구조가 매트릭스인지 논리값을 물어보는 함수 is.matrix활용
is.matrix(m_01)
```

```
TRUE
```

In [62]:

```
# 열을 지정 : ncol
m_02 <- matrix(c(1, 2, 3, 4), ncol = 2)
m_02
```

```
1 3
2 4
```

In [63]:

```
# 벡터의 값을 matrix함수에서 by = 매개변수를 통해 행을 먼저 채울지 열을 먼저 채울지 결정
# 첫번째 열의 행을 먼저 채움
m_03 <- matrix(c(1, 2, 3, 4), ncol = 2, by = 0)
m_03
```

```
1  3
2  4
```

In [64]:

```
# 첫번째 행의 열을 먼저 채움
m_04 <- matrix(c(1, 2, 3, 4), ncol = 2, by = 1)
m_04
```

```
1  2
3  4
```

In [65]:

```
m_05 <- matrix(c("김노랑님", "이빨강님", "이보라님", '박회색님',"이파랑님", '박회색님'),
                ncol = 3)
m_05
```

```
김노랑님  이보라님  이파랑님
이빨강님  박회색님  박회색님
```

In [66]:

```
# Matrix의 요소(element) 접근

m_05[2, ] # 두번째 행 모든 값
```

```
'이빨강님'  '박회색님'  '박회색님'
```

In [67]:

```
m_05[, 3] # 첫번째 열의 모든 값
```

'이파랑님' '박회색님'

In [68]:

```
m_05[2, 2:3]
```

'박회색님' '박회색님'

2.3 Arrays

- 2차원 이상의 데이터
- 한가지 데이터 타입으로 구성

- 형태
  - 나의_배열_변수명 <-array(숫자 or 문자 or 논리값, dim = c(행, 열, 갯수))
  - 갯수: 동일 행과 열이 된 행렬의 갯수

- Indexing (slicing) : 항목의 값에 접근 할때
  - 나의_행렬_변수명[행, 열, 몇번째]
  - 나의_벡터_변수명[2, 1, 3] : 세번째 행렬의 2번째 행에서 1번째 열의 값

In [69]:

```
array(c(1:4))
```

1 2 3 4

In [70]:

```
array(c(1, "a", '3'), dim = c(2, 3, 2))
```

'1' 'a' '3' '1' 'a' '3' '1' 'a' '3' '1' 'a' '3'

In [71]:

```
arr1 <- array(1:20, dim = c(2, 5, 2))
arr1
```

```
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17
18  19  20
```

In [72]:

```
matrix(1:10, ncol = 5)
matrix(11:20, ncol = 5)
```

```
 1  3  5  7  9

 2  4  6  8  10
```

```
 11  13  15  17  19

 12  14  16  18  20
```

In [73]:

```
arr1[2, 2, 1]
```

```
4
```

In [74]:

```
arr1[1, 5, 2]
```

```
19
```

2.4 data frame

- 2차원 데이터 구조
- 다양한 데이터 타입으로 구성

- 형태
  - 나의_데이터프레임_변수명 <- data.frame(열이름1 = 벡터, 열이름2 = 벡터)
  - 행의 길이가 같은경우: data.frame(열이름1 = 벡터, 열이름2 = 데이터프레임, 열이름3 = 행렬)

- Indexing (slicing) : 항목의 값에 접근 할때
  - 나의_데이터프레임_변수명[행, 열]
  - 나의_데이터프레임_변수명[2, 1] : 2번째 행에서 1번째 열의 값

In [75]:

```
x <- data.frame(var1  = 1:3)
x
```

var1
1
2
3

In [76]:

```
y <-  matrix(3:8, ncol = 2)
y
```

3	6
4	7
5	8

In [77]:

```
X <- data.frame(var1 = x, var2 = y)
X
```

var1	var2.1	var2.2
1	3	6
2	4	7
3	5	8

In [78]:

```
X[1, ]
```

var1	var2.1	var2.2
1	3	6

In [79]:

```
X[, 2]
```

3 4 5

In [80]:

```
X[, c("var2.1", "var2.1")]
```

var2.1	var2.1.1
3	3
4	4
5	5

In [81]:

```
X$var2.1
```

3 4 5

2.5 list

- 다양한 데이터구조를 하나 저장할수 있는 데이터 구조
- 다양한 데이터 타입으로 구성
- 데이터의 길이나 행과 열이 중요하지 않으며, 항목의 순서대로 데이터 저장

- 형태
  - 나의_리스트_변수명 <- list(이름1 = 벡터, 이름2 = 행렬, 이름3 = 데이터프레임, 이름4 = Array)

- Indexing (slicing) : 항목의 값에 접근 할때
  - 나의_행렬_변수명[행, 열]
  - 나의_벡터_변수명[2, 1] : 2번째 행에서 1번째 열의 값

In [82]:

```
a <- 100
b <- 'B'
c <- matrix(c(1:9), ncol = 3)
d <- array(1:20, dim = c(2, 5, 2))
e <- data.frame(var1 = c(1:3), var2 = c(4:6))
my_list <- list(a, b, c, d, e)
my_list
```

1. 100
2. 'B'
3.
   1 4 7

   2 5 8

   3 6 9
4. 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
   17 18 19 20
5.

var1	var2
1	4
2	5
3	6

In [83]:

```
my_list[1]
```

1. 100

In [84]:

```
my_list[[1]]
```

100

In [85]:

```
my_list[[5]]
```

var1	var2
1	4
2	5
3	6

In [86]:

```
my_list[[5]][2, 1]
```

2

3. 선언문(Statements)

3.1 조건문(if)

3.2 반복문(loop)

3.3 함수

3.1 조건문

- 주어진 조건에 따라 논리값(boolean) 값이 TRUE 조건에 실행
- if ~
- if ~ else ~
- if ~ else if ~ else ~
- ifelse

if ~

> if (조건) {
>
> 조건이 참일 때 실행
>
> }

In [87]:

```
x <- "배가 고프다"

if (x == "배가 고프다"){
    print("밥 먹으러 가자!")
}
```

```
[1] "밥 먹으러 가자!"
```

if ~ else ~

> if (조건1) {
>
> 조건1일 참일 때 실행
>
> } else { 조건1일 거짓일 때 실행 }

In [88]:

```
x <- 4

if (x %% 2 == 0) {
    print("짝수")
} else {
    print("홀수")
}
```

```
[1] "짝수"
```

if ~ else if ~ else ~

> if (조건1의 논리값) {
>
> TRUE인 경우 실행
>
> } else if (조건2의 논리값) {
>
> 조건 2의 논리 값이 TRUE인 경우
>
> } else {
>
> 조건 2의 논리값이 FALSE일 때 실행
>
> }

In [89]:

```
x <- 4

if (x %% 2 == 0) {
    print("짝수")
} else {
    print("홀수")
}
```

```
[1] "짝수"
```

ifelse ~

> - ifelse(조건, 표현식1, 표현식2)

In [90]:

```
ifelse( 3 > 4, "참", "거짓")
```

'거짓'

3.2 반복문

- for ~
- while ~

for

> for (변수명 in 범위) {
>
> 반복할 함수
>
> }

In [91]:

```
for (i in 1:10) {
    print(i)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

In [92]:

```
for (i in 1:10){
    if(i == 3){
        next
    }
    print(i)
}
```

```
[1] 1
[1] 2
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

In [93]:

```
for (i in 1:10){
    if (i == 4){
        break
    }
    print(i)
}
```

```
[1] 1
[1] 2
[1] 3
```

In [2]:

```
tmp <- c("팀장1", "팀장2", "팀장1", "팀장3")
tmp[1]
```

'팀장1'

In [20]:

```
for (i in 1: length(tmp)) {
    if ( tmp[i] == "고장2" ) {
        print("고장2 입니다.") }
    else {print("고장이 아닙니다.")}
}
```

```
[1] "고장이 아닙니다."
[1] "고장2 입니다."
[1] "고장이 아닙니다."
[1] "고장이 아닙니다."
```

In [1]:

```
# for ~ else ~

tmp <- c("서울", "익산", "익산", "서울")

for (i in 1: length(tmp)) {
    if(tmp[i] == "익산") {
        print("익산, Yes!")
} else {
    print("서울, 머네요!")
}
}
```

```
[1] "서울, 머네요!"
[1] "익산, Yes!"
[1] "익산, Yes!"
[1] "서울, 머네요!"
```

while

> while (조건) {
>
> 반복할 함수
>
> }

In [95]:

```
x <- 1
while (x <=5){
    print(x)
    x <- x+1
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

3.3 함수

> - 프로그램 속의 작은 프로그램
> - 명령문의 연속열에 이름을 붙여주는 것

함수 사용 이유

> - 복잡한 프로그램 작성할 때 피요한 도구 : 코드 중복 방지
> - 가독성 : 프로그램을 이해하고 유지 보수하기 쉽게 해줌
> - 재사용성 : 프로그램의 어느 지점에서든지 함수의 이름을 통해 실행

구분

- Built-in function(내장함수) : ex) install.packages()
- 패키지(외장함수) : 패키지 설치, 로드하여 사용
  - 패키지 설치 : install.packages함수 활용 ("설치하려는 패키지 이름")
  - 패키지 로드 : library("설치된 패키지 이름"), require("설치된 패키지 이름")
- 사용자 함수

In [96]:

```
# install.packages("ggplot2")
# library(ggplot2)
```

함수 형태

- 함수의 호출 : 함수의 사용
- function(argument): 함수이름(매개변수) --> 결과 (리턴값; return)
- ex) type(3) : 함수이름 type(), 매개변수: 3

In [97]:

```
# library(plyr)
```

In [98]:

```
# ?library
```

In [99]:

```
# available.packages()
```

user function

- 나의_사용자함수_이름 <- function(매개변수) {실행 연산}

In [100]:

```
f <- function() {
    return('안녕하세요')
}
```

In [101]:

```
# inspect
f
```

```
function ()
{
    return("안녕하세요")
}
```

In [102]:

```
f()
```

'안녕하세요'

In [103]:

```
# 매개변수 2개인 함수

my_sum <- function(a, b) {
    return(a + b)
}
```

In [104]:

```
my_sum(1, 2)
```

3

Reuseable function

1. 새 스크립트에 함수를 저장
2. source("함수들이 저장된 스크립트.R")

In [4]:

```
source("my_sum_func.R")
```

In [6]:

```
my_sum(1, 2)
```

3