```
#=============================================================

#2.1 _ 학습하기 전에 알아두면 좋은 내용

#=============================================================

#love 변수에 1을 지정

love <- 1


#love 변수 출력

print(love)


#-------------------------------------------------------------


#=============================================================

#2.2 _ 너는 내가 정의한다 - "<-"

#=============================================================



# 아직 아무것도 설정돼 있지 않은 love 변수를 출력

print(love)


#-------------------------------------------------------------


# love 변수에 1을 지정

love <- 1


# love 값 출력

print(love)
```

```
#------------------------------------------------------------

# 값이 1로 설정된 love 변수에 문자열 지정
love <- "안녕하세요"

# love 값 출력
print(love)

#------------------------------------------------------------

# love 출력 – 현재는 문자열
print(love)

# 현재 문자열인 love를 함수처럼 사용해 보면 오류가 발생
love("이제 나는 함수가 될 수 있을까?")

# love 변수에 print 함수를 지정
love <- print

# print 함수 대신 love로도 출력 가능, 데이터프레임, 벡터 등 모든 객체를 변수에 저장 가능
love("이제 나는 함수가 되었다!")



#------------------------------------------------------------
```

```
# 객체 설정
love_num <- 1
love_str <- "안녕하세요"
love_vec <- c(1,1,1,1)
love_fun <- print

# str 함수를 통해 객체의 정보를 확인할 수 있음
# love_num은 숫자 1
str(love_num)

# love_str은 문자 "안녕하세요"
str(love_str)

# love_vec은 길이가 4인 숫자형 벡터/ 값은 1,1,1,1
str(love_vec)

# love_fun은 함수
str(love_fun)

#------------------------------------------------------------

# B는 A로부터 지정된 객체
A<-1
B<-A
print(A)
```

```
print(B)


# A의 값을 변경하더라도 B의 값은 여전히 1이다.
A<-9
print(A)
print(B)


#------------------------------------------------------------


a <- 1


# print 함수를 사용해 출력
print(a)


# 객체의 이름만 입력해도 출력됨
a


#================================================================
#2.3 _ 데이터 구조의 기본 - 벡터
# 1차원 배열, 요소는  모두 같은 데이터 타입
# 'c' 함수를 통해 생성 (combine)
#================================================================


# 1, 2, 3, 4를 요소로 가지는 벡터 생성
vec_t <- c(1,2,3,4)
```

```
# 생성된 벡터의 값 확인

vec_t

# 벡터 정보 확인 – 길이가 4인 숫자 벡터 / 값은 1, 2, 3, 4

str(vec_t)

# 벡터의 길이 확인

length(vec_t)

#------------------------------------------------------------

# 벡터를 생성할 때 문자와 숫자를 함께 사용한다면?

vec_t <- c(1, "hi", 2)

# 생성된 벡터 값 확인

vec_t

# 벡터 정보 확인 – 길이가 3인 문자 벡터 / 값은 "1", "hi", "2" / 숫자를 문자로 변환

str(vec_t)

#------------------------------------------------------------
```

```
# 1로 선언

scalar_item <- 1

# 객체 값 출력

scalar_item

# 1을 요소로 가지는 길이가 1인 벡터 선언

vector_item <- c(1)

# scalar_item과 출력 값이 같음

vector_item

# identical은 두 객체가 같은지 확인하는 R의 함수 - 한개의 문자나 숫자도 길이 1인 벡터로 인식

identical(scalar_item, vector_item)

#------------------------------------------------------------

# 숫자형 벡터 - 양수, 음수, 소수점 모두 하나의 벡터에 담을 수 있음
```

```
numeric_vector <- c( 0.2, -1, 2 , -0.5 )

# 출력

numeric_vector

# numeric_vector의 타입 확인

mode(numeric_vector)

#------------------------------------------------------------

# 숫자 벡터 선언

n_vector <- c( 1,2,3,4,5,6,7,8,9)

# 최솟값

min(n_vector)

# 최댓값

max(n_vector)

# 평균
```

```
mean(n_vector)

# 중간값

median(n_vector)

# 합계

sum(n_vector)

#------------------------------------------------------------

# Inf 와 NaN이 포함된 벡터 만들기

numeric_vector <- c(1/0, 2/2, -2/2, -1/0 , 0/0)

# 벡터 확인

numeric_vector

#------------------------------------------------------------

# TRUE, FALSE로 설정

ex_logical_1 <- c(TRUE, FALSE, TRUE, FALSE)

ex_logical_1
```

```
# mode 함수를 통해 벡터의 데이터 타입 확인

mode(ex_logical_1)

# T, F로도 설정 가능

ex_logical_2 <- c(T, F, T, F)
ex_logical_2

# mode 함수를 통해 벡터의 데이터 타입 확인

mode(ex_logical_2)

#------------------------------------------------------------

# 소문자는 오류

ex_logical_3 <- c(true, false, true, false)

# 따옴표를 붙이면 문자열로 인식

ex_logical_4 <- c("TRUE", "FALSE", "TRUE", "FALSE")
ex_logical_4

# mode 함수를 통해 벡터의 타입 확인
```

```
mode(ex_logical_4)

#------------------------------------------------------------

# 논리 벡터 – TRUE, FALSE, T, F 모두 가능

ex_logical <- c(TRUE, T, FALSE, F)

ex_logical

# ! 기호를 이용해 TRUE는 FALSE로, FALSE는 TURE로 변환

!ex_logical

#------------------------------------------------------------

# as.logical: logical 변수로 형변환

ex_logical <- as.logical(c( 0,-1 , 1 , 100 , -7 ))

# 확인- 0 값만 FALSE로 인식

print(ex_logical)

# as.numeric: 숫자형으로 형변환

# FALSE는 0, TRUE는 1로 변환
```

```
as.numeric(ex_logical)

#------------------------------------------------------------

# 문자열 벡터 생성

v_charater <- c("문자열1", "문자열2", "A", "1")
v_charater

# mode 함수를 통해 데이터 타입 확인

mode(v_charater)

#------------------------------------------------------------

# 문자 개수 출력

nchar(c("F123", "F124", "F125", "F126"))

# 문자열 자르기 – 두 번째부터 네 번째 문자 사이의 문자열 추출

substr("1234567", 2, 4)
substr(c("F123", "F124", "F125", "F126"), 2, 4)

# 특정 문자로 데이터 나누기 – split에 정의한 구분자를 기준으로 문자열을 나누어 벡터로 반환
```

```
strsplit('2014/11/22', split="/")

# 문자열 합치기 – 합칠 때 문자열 사이의 문자는 sep에 정의
# sep을 정의하지 않으면 문자열 사이에 공백을 붙여 합침

paste("50 = ", "30 + ", "20", sep="")

paste("50", "30", "20", sep="*")

# 대문자 변환
toupper("AbCdEfGhIjKlMn")

# 소문자 변환
tolower("AbCdEfGhIjKlMn")

#------------------------------------------------------------

# 팩터로 변환할 문자 벡터

v_character <- c("사과", "복숭아", "사과", "오렌지", "사과", "오렌지", "복숭아")
v_character

# factor 함수로 팩터를 생성 - 범주형 데이터(성별, 수강과목, 혈액형, 등급 - 제한된 범주)
# factor(x, levels, ordered) - ordered(서열형), levels(범주)

v_factor <- factor(v_character)
```

```
v_factor

#-------------------------------------------------------------

# 팩터 출력
v_factor

# 데이터 타입 확인 – 문자로 출력되나 데이터 타입은 숫자(numeric)

mode(v_factor)

# 객체 정보 확인

str(v_factor)

#-------------------------------------------------------------

# 팩터를 문자 벡터로 변환

v_factor_to_char <- as.character(v_factor)
v_factor_to_char

# 팩터를 숫자 벡터로 변환

v_factor_to_num <- as.numeric(v_factor)
v_factor_to_num
```

```
#------------------------------------------------------------

# 팩터로 변환할 문자 벡터

v_character <- c("사과", "복숭아", "사과", "오렌지", "사과", "오렌지", "복숭아")

# factor 함수로 팩터 객체 생성

v_factor <- factor(v_character, levels=c("사과", "복숭아"))

# 생성된 팩터 출력

v_factor

#------------------------------------------------------------

# levels 범주 순서 변경 1

v_factor <- factor(v_character, levels=c("사과", "복숭아","오렌지"))

# 생성된 팩터 출력

v_factor

# levels 범주 순서 변경 2
```

```
v_factor <- factor(v_character, levels=c("복숭아","오렌지","사과"))

# 생성된 팩터 출력

v_factor

#------------------------------------------------------------

# 등급을 나타내는 문자 벡터

ex_label <- c("하하", "중하", "중", "중상", "상상")

# 서열형 데이터 팩터 생성

ordered_factor <- factor(ex_label, ordered=T)

ordered_factor

#------------------------------------------------------------

# levels 입력 항목을 이용해 서열 순으로 범주 순서를 정의

factor(ex_label, levels=c("하하","중하","중","중상","상상"), ordered=T)

#------------------------------------------------------------

# 숫자 벡터
```

```
v_num <- c(1000,2000,1000,2000,3000,2000,3000)

# 숫자 벡터를 팩터로 변환 – 범주가 1000, 2000, 3000인 팩터

v_num_factor <- factor(v_num)

v_num_factor

# 팩터->숫자 벡터로 변환(1000, 2000 등이 아닌 내부 코드 값이 나옴)

as.numeric(v_num_factor)

# 팩터->문자 벡터로 변환

v_char <- as.character(v_num_factor)

v_char

# 문자 벡터-> 숫자 벡터로 변환

v_num <- as.numeric(v_char)

v_num

#-------------------------------------------------------------

ex_trans <- c(1, 0, 1, 0, 0, 0)

# 문자 타입 변환
```

```
as.character(ex_trans)

# 논리 타입으로 변환/논리 타입에서는 0은 FALSE, 0이 아닌 것은 TRUE
as.logical(ex_trans)

#-------------------------------------------------------------

# 벡터 생성
t_vector <- c(11, 12, 13, 14, 15, 16, 17, 18, 19, 20)



# t_vector 전체 출력
t_vector

# t_vector의 3번째 요소 선택
t_vector[3]

# 1, 3, 5, 6번째 요소 선택
idx <- c(1, 3, 5, 6)
t_vector[idx]

# 곧바로 t_vector의 1, 3, 5, 6번째 요소 선택
t_vector[c(1, 3, 5, 6)]

#-------------------------------------------------------------
```

```
# t_vector

t_vector


# 일부 요소 선택

t_vector[c(1, 3, 5, 6)]


# 출력하는 순서까지 지정 가능

t_vector[c(6, 5, 3, 1)]


#------------------------------------------------------------


# :를 이용해 연속된 숫자 벡터 생성

seq_vector <- 3:7


# seq_vector 값 확인

seq_vector


#------------------------------------------------------------


# 50 ~ 100까지 벡터 생성

seq_vector <- 51:100


# 변수 출력

seq_vector


# 30번째 ~ 40번째 요소까지만 출력
```

```
seq_vector[30:40]

#------------------------------------------------------------

# seq 함수
# from: 시작숫자, to : 종료숫자, by : 건너뛸 숫자

# 10 ~ 20까지 2씩 커지는 숫자 생성
seq(from=10, to=20, by=2)

# 20 ~ 10까지 2씩 작아지는 숫자 생성
seq(from=20, to=10, by=-2)

#------------------------------------------------------------

# 벡터 생성
t_vector <- c(11, 12, 13, 14, 15)

# 3번째 요소 선택
# 선택할 요소의 위치에는 TRUE, 선택하지 않을 요소의 위치에는 FALSE를 지정
logical_idx <- c(F, F, T, F, F)

# TRUE로 설정된 3번째 요소만 출력됨
t_vector[logical_idx]

# 바로 3번째 요소 출력
```

```
t_vector[c(F,F,T,F,F)]


# 3, 4번째 요소 출력
t_vector[c(F,F,T,T,F)]


#------------------------------------------------------------


t_vector


# 논리 연산: 3보다 같거나 작은가?
t_vector <= 3


# 논리 연산: 3보다 큰가?
t_vector > 3


#------------------------------------------------------------


t_vector


# 3보다 같거나 작은 요소 선택
t_vector[t_vector <= 12]


# 3보다 큰 요소 선택
t_vector[t_vector > 12]


#------------------------------------------------------------
```

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

# vector_m의 3번째 요소의 값을 10으로 변경
vector_m[3] <- 10
vector_m

# vector_m의 2, 4번째 요소의 값을 9로 변경
vector_m[c(2, 4)] <- 9
vector_m

# 5보다 큰 vector_m의 요소의 값을 모두 3으로 변경
vector_m[vector_m > 5] <- 3
vector_m

# vector_m의 2 ~ 5 번째 요소의 값을 모두 0으로 변경
vector_m[2:5] <- 0
vector_m

#------------------------------------------------------------

vector_m

# 전체 요소의 값을 1로 설정?
```

```
vector_m <- 1

# 결과 확인
vector_m

#------------------------------------------------------------

# 벡터 생성
vector_m <- c(1, 2, 3, 4, 5)

# 벡터 길이
length(vector_m)

# 요소 각각을 1로 설정
# vector[1:5] <- 1과 같은 것임, 인덱스를 명시적으로 지정해야
vector_m[1:length(vector_m)] <- 1

# 설정한 값 확인
vector_m

#------------------------------------------------------------

# 벡터 변수 생성
v_add <- c(1, 2, 3, 4, 5)

# 앞에 0 추가하기(0과 v_add를 합쳐서 다시 v_add에 지정)
```

```
v_add <- c(0, v_add)

# 추가된 값 확인
v_add

# 앞에 -2, -1 추가
v_add <- c(c(-2, -1), v_add)

# 추가된 값 확인
v_add

# 뒤에 6 추가
v_add <- c(v_add, 6)

# 추가된 값 확인
v_add

# 뒤에 7~10 추가
v_add <- c(v_add, 7:10)

# 추가된 값 확인
v_add

#------------------------------------------------------------

# 벡터 생성
```

```
t_add1 <- c(1, 2, 3)

t_add2 <- c(4, 5, 6)

t_add3 <- c(7, 8, 9)


# 3개의 벡터를 엮어(combine) 하나의 벡터 생성

new_add <- c(t_add1, t_add2, t_add3)


# 결과 확인

new_add


#------------------------------------------------------------


# 벡터 생성

vector_a <- c("A", "B", "C", "F", "G")

vector_b <- c("D", "E")


# append(원본 벡터, 추가할 벡터, 추가할 위치)

# vector_a의 3번째 요소 뒤에 vector_b를 추가

append(vector_a, vector_b, 3)


# vector_a 출력

vector_a


#------------------------------------------------------------


# append의 결과를 vector_a에 지정
```

```
vector_a <- append(vector_a, vector_b, 3)

# vector_a 출력
vector_a

#------------------------------------------------------------

t_vector

# 1, 3, 5, 6번째 요소 선택
t_vector[c(1, 3, 5, 6)]

# 1, 3, 5, 6번째 요소를 제외하고 선택 - 요소 삭제
t_vector[-c(1, 3, 5, 6)]

# t_vector의 길이
length(t_vector)

# t_vector 중 맨 마지막 요소를 제외하고 선택
t_vector[-length(t_vector)]

#------------------------------------------------------------

# 2, 3번째 항목이 TRUE로 설정된 논리 벡터 생성
logical_var <- c(FALSE, TRUE, TRUE, FALSE, FALSE)
logical_var
```

```
# 논리 벡터에 ! 기호 적용
!logical_var

# 벡터 생성
v_str <- c("첫째", "둘째", "셋째", "넷째", "다섯째")
v_str

# 두 번째, 세 번째 항목 선택
v_str[logical_var]

# 두 번째, 세 번째 항목 제외
v_str[!logical_var]

#------------------------------------------------------------

# 벡터 생성
a <- c(1, 2, 3, 4)
b <- c(5, 6, 7, 8)

# 더하기
c <- a+b
c

# 빼기
c <- a-b
```

```
c


# 곱하기
c <- a*b
c


#------------------------------------------------------------


# 서로 길이가 다른 벡터 생성
a <- c(1, 2, 3, 4)
b <- c(5, 6)


# 더하기
c <- a+b
c


# 거꾸로 더해보기 -> a+b와 결과가 같음
c <- b+a
c


# 빼기
c <- a-b
c


# 곱하기
c <- a*b
```

```
c


#------------------------------------------------------------


# 벡터 생성
a <- c(1, 2, 3, 4)


# 벡터에 숫자 2 곱하기(숫자 직접 대입)
c <- a*2
# 계산 결과
c


# 벡터에 숫자 2 곱하기(별도 변수로 대입)
b <- 2
c <- a*b
# 계산 결과
c


# 벡터 길이가 1인 벡터 곱하기
b <- c(2)
c <- a*b
# 계산 결과
c


# 같은 길이의 벡터 곱하기
b <- c(2, 2, 2, 2)
```

```
c <- a*b

# 계산결과

c

#------------------------------------------------------------


# 두 벡터의 길이가 서로 배수인 경우(4는 2의 배수)

a <- c(1, 2, 3, 4)

b <- c(1, 2)


# 경고 없이 계산됨

print(a+b)


# 두 벡터의 길이가 서로 배수가 아닌 경우

a <- c(1, 2, 3, 4)

b <- c(1, 2, 3)


# 계산은 되나 경고 메시지 출력

print(a+b)
```