한빛아카데미 Hanbit Academy, Inc.

R 코딩 &
데이터 분석

Chapter 03. 변수, 벡터, 함수 알아보기

목차

01

이것이 변수입니다

01. 이것이 변수입니다

I. 변수의 개념

- **변수(variable) :** 변수는 프로그램 내에서 어떤 값을 저장해 놓을 수 있는 '보관 상자'의 역할을 함.
- [그림 3-1]의 오른쪽 그림을 보면 상자는 **변수**, a는 **변수의 이름**, 10은 **변수에 저장되는 값**을 의미.

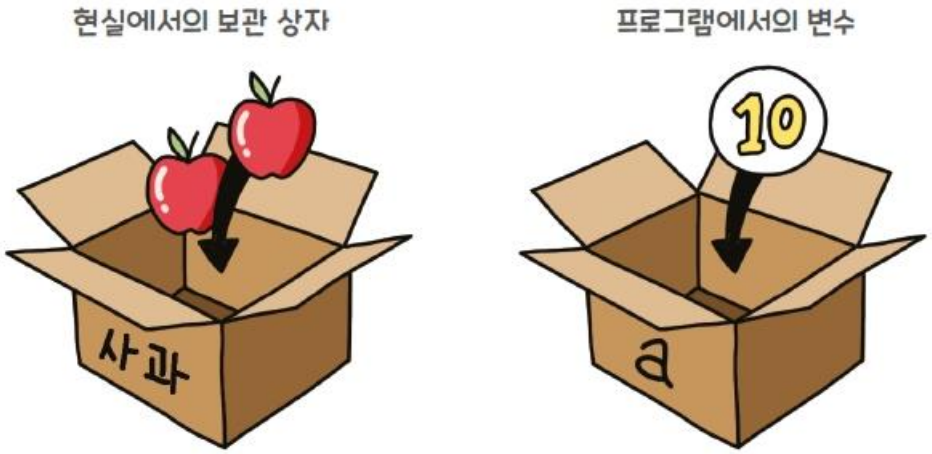


그림 3-1 보관 상자와 변수

01. 이것이 변수입니다

I. 변수의 개념

표 3-1 보관 상자와 변수의 공통점과 차이점

	공통점	차이점
현실에서의 보관 상자	• 어떤 것을 안에 보관할 수 있다. • 보관된 내용물이 바뀔 수 있다. • 내용물을 구분하기 위해 이름을 붙일 수 있다.	• 저장할 수 있는 내용물이 다양하다. • 여러 물건을 한 상자에 보관 가능하다.
변수		• 저장할 수 있는 내용물은 오직 숫자나 문자이다. • 하나의 변수에는 하나의 값만 저장할 수 있다.

01. 이것이 변수입니다

I. 변수의 개념

▪ 변수 만들기

```
a <- 10
```

- a라는 변수에 10이라는 값을 저장하고 싶다면 위와 같은 명령문 실행.
- 10을 a로 보내라는 모양을 하고 있는데, 이것이 '10을 a에 저장하라'는 명령임.
- a는 변수명, 10은 변수에 저장할 값, <-는 값을 변수에 저장하는 연산자.

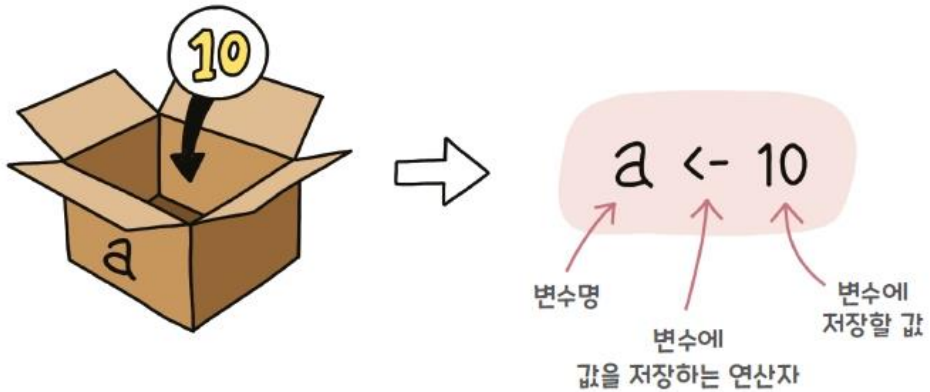


그림 3-2 변수 a에 10을 저장하기

01. 이것이 변수입니다

I. 변수의 개념

- **변수의 내용 확인하기**

[코드 3-1]

```
total <- 5050                # 5050을 변수 total에 저장
total                        # 방법 1
print(total)                 # 방법 2
cat('합계 :', total)         # 방법 3
```

01. 이것이 변수입니다

I. 변수의 개념

▪ 변수의 내용 확인하기

[코드 3-1] 실행 결과(1)

```
> total <- 5050          # 5050을 변수 total에 저장
> total                  # 방법 1
[1] 5050
```

– 첫 번째 방법은 가장 쉬운 방법으로 변수명 total을 입력하여 실행.

– 또는 [그림 3-3]과 같이 total <- 5050에 있는 변수명 total을 마우스로 블록 설정한 다음 <Ctrl>+<Enter>를 누르면 됨.

그림 3-3 변수 total을 마우스로 선택한 후 Ctrl + Enter 누르기

01. 이것이 변수입니다

I. 변수의 개념

▪ 변수의 내용 확인하기

[코드 3-1] 실행 결과(2)

```
> print(total)           # 방법 2
[1] 5050
```

– 두 번째 방법은 print() 함수 사용.

[코드 3-1] 실행 결과(3)

```
> cat('합계 :', total)   # 방법 3
합계 : 5050
```

– 세 번째 방법은 cat() 함수 사용.

– cat() 함수는 여러 개의 변수 또는 값을 연결하여 한번에 출력할 때 사용.

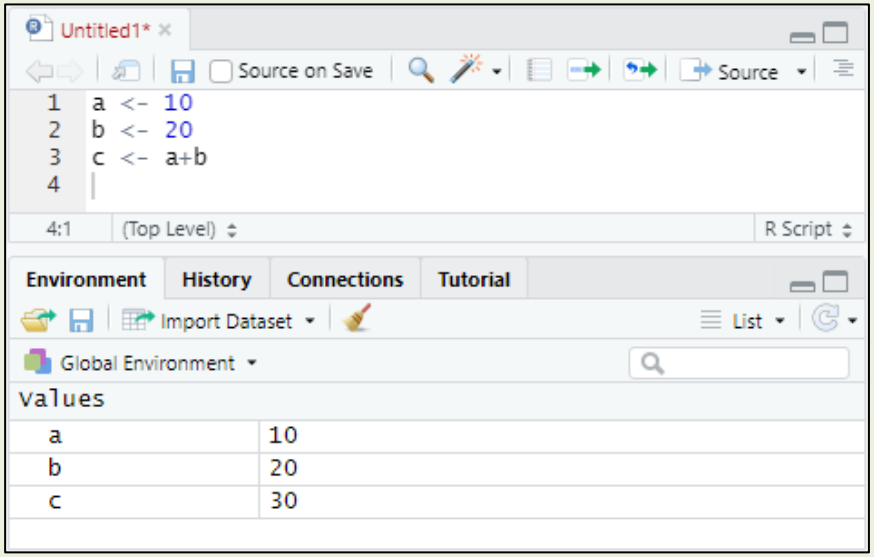
01. 이것이 변수입니다

I. 변수의 개념

▪ 변수의 내용 확인하기

하나 더 알기 **R 스튜디오에서 변수의 내용을 확인하는 방법**

- R 스튜디오에서 다음과 같이 환경창(Environment Panel)을 보면 현재 만들어진 변수와 저장된 값을 확인할 수 있습니다.



01. 이것이 변수입니다

II. 변수를 이용한 산술연산

[코드 3-2]

```
a <- 10          # 변수 a에 10을 저장
b <- 20          # 변수 b에 20을 저장
c <- a + b       # 변수 a, b에 저장된 값을 더하여 변수 c에 저장
print(a)         # 변수 a의 내용 출력
print(b)         # 변수 b의 내용 출력
print(c)         # 변수 c의 내용 출력
```

II. 변수를 이용한 산술연산

[코드 3-2] 실행 결과(1)

```
> a <- 10          # 변수 a에 10을 저장
> b <- 20          # 변수 b에 20을 저장
> c <- a + b       # 변수 a, b에 저장된 값을 더하여 변수 c에 저장
```

- 첫 번째 줄 : 변수 a에 10이라는 값 저장하는 명령문.
- 두 번째 줄 : 변수 b에 20이라는 값 저장하는 명령문.
- 세 번째 줄 : 변수 a에 저장된 값과 변수 b에 저장된 값을 더하여 변수 c에 저장하는 명령문으로, 그 결과 변수 c에는 30이 저장.

II. 변수를 이용한 산술연산

[코드 3-2] 실행 결과(2)

```
> print(a)      # 변수 a의 내용 출력
[1] 10
> print(b)      # 변수 b의 내용 출력
[1] 20
> print(c)      # 변수 c의 내용 출력
[1] 30
```

– 변수 a, b, c에 저장된 값을 차례로 화면에 출력하는 명령문.

III. 변수에 대해 알아야 할 것들

- 변수의 값은 바뀔 수 있다

[코드 3-3]

```
a <- 10
print(a)
a <- 50
print(a)
```

[실행 결과]

```
> a <- 10
> print(a)
[1] 10
> a <- 50
> print(a)
[1] 50
```

III. 변수에 대해 알아야 할 것들

▪ 하나의 변수에는 하나의 값만 저장 가능

- 다음과 같이 num이라는 변수에 두 개의 값 2, 5를 저장하려고 하면 에러 발생.
- R에서는 여러 개의 값을 한 곳에 저장하는 방법으로 벡터(vector)를 제공.

```
num <- 2, 5                    # 에러 발생
```

III. 변수에 대해 알아야 할 것들

- **변수와 변수의 연산은 변수에 저장된 값들의 연산으로 바뀌어 실행**

```
c <- a + b
```

– 이 명령문은 [그림 3-4]와 같이 실행.

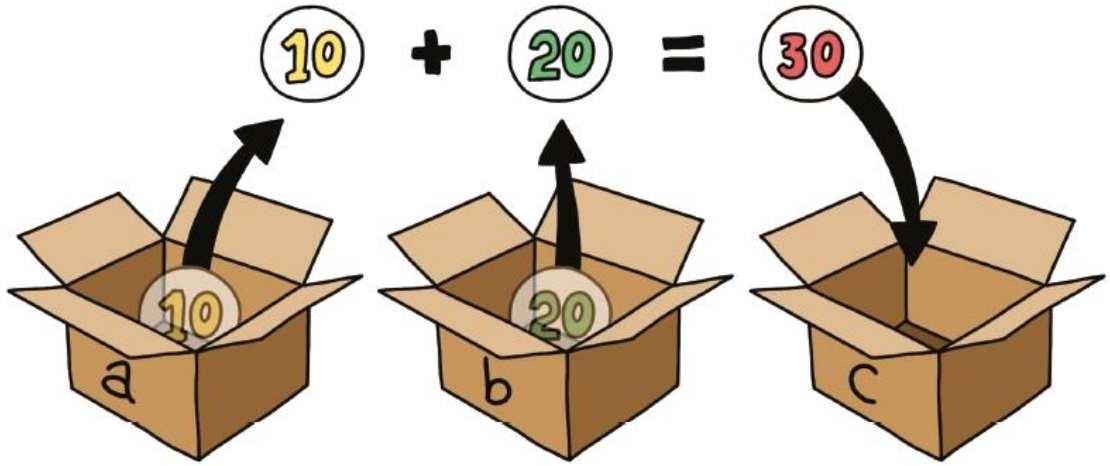


그림 3-4 c <- a + b의 실행 과정

01. 이것이 변수입니다

IV. 변수명의 작명 규칙

- 첫 글자는 영문자나 마침표(.)로 시작하는데, 일반적으로 영문자로 시작.

  ex) avg, .total

  ex) 12th는 숫자로 시작했기 때문에 변수명 사용 불가

- 두 번째 글자부터는 영문자, 숫자, 마침표(.), 밑줄(_)을 사용 가능.

  ex) value.1, sub_total, d10

  ex) this-data, this@data는 특수문자를 사용했으므로 변수명 사용 불가

- 변수명에서 대문자와 소문자는 별개의 문자 취급.

  ex) var_A와 var_a는 서로 다른 변수

- 변수명 중간에 빈 칸을 넣을 수 없습니다.

  ex) first ds는 변수명 사용 불가

01. 이것이 변수입니다

IV. 변수명의 작명 규칙

- 변수명은 저장되는 값이 무엇인지 알 수 있는 단어를 사용하는 것이 좋음.

```
abc <- 850      # 어떤 작업을 하려는 것인지 알기 어렵다.
mid.sum <- 850  # 중간 합계로 850을 저장하려는 것임을 예상할 수 있다.
```

- 자주 사용하는 함수의 이름은 변수명으로 사용하지 않는 것이 좋음.

```
a <- sqrt(120)   # 제곱근을 구하는 함수로서의 sqrt
a                # a에 저장된 값을 출력
sqrt <- 340      # 변수명으로서의 sqrt
sqrt             # sqrt에 저장된 값을 출력
```

01. 이것이 변수입니다

V. 변수에 저장될 수 있는 값의 종류

- **자료형(data type) :** 변수에 저장할 수 있는 값의 종류.

표 3-2 R에서 사용할 수 있는 값에 대한 자료형

자료형	사용 예	비고
숫자형	1, 2, -4, 12.8	정수, 실수 모두 가능
문자형	'Tom', "Jane"	작은따옴표나 큰따옴표로 묶어서 표현
논리형	TRUE, FALSE	T, F로 줄여서 사용하는 것도 가능
특수 값	NULL	정의되어 있지 않음을 의미하며, 자료형도 없고 길이도 0
	NA	결측값(missing value)
	NaN	수학적으로 정의가 불가능한 값(예 sqrt(-3))
	Inf, -Inf	양의 무한대(Inf), 음의 무한대(-Inf)

V. 변수에 저장될 수 있는 값의 종류

▪ 숫자형과 문자형

- **숫자형 :** 자연수를 포함하여 양의 정수, 음의 정수, 0, 실수 등의 값으로, 산술연산 가능.
- **문자형 :** 숫자형과는 달리 산술연산을 할 수 없고, 문자형의 값은 반드시 작은따옴표나 큰따옴표로 묶어 표시해야 함.
- 모양은 숫자라도 따옴표로 묶으면 문자로 간주되어 산술연산에 사용할 수 없음.

V. 변수에 저장될 수 있는 값의 종류

- 숫자형과 문자형

[코드 3-4]

```
age.1 <- 20                 # 숫자 저장
age.2 <- 25                 # 숫자 저장
print(age.1 + age.2)        # 정상 실행
name.1 <- 'John’            # 문자 저장
print(name.1)               # 정상 실행
grade.1 <- '3’              # 문자 저장
print(age.1 + grade.1)      # 에러 발생
```

V. 변수에 저장될 수 있는 값의 종류

- 숫자형과 문자형

[코드 3-4] 실행 결과

```
> age.1 <- 20                    # 숫자 저장
> age.2 <- 25                    # 숫자 저장
> print(age.1 + age.2)           # 정상 실행
[1] 45
> name.1 <- 'John’               # 문자 저장
> print(name.1)                  # 정상 실행
[1] "John"
> grade.1 <- '3’                 # 문자 저장
> print(age.1 + grade.1)         # 에러 발생
Error in age.1 + grade.1 : 이항연산자에 수치가 아닌 인수입니다
```

V. 변수에 저장될 수 있는 값의 종류

▪ 논리형

- 논리형 값 TRUE와 FALSE는 '참'과 '거짓'을 표현하는 값으로, 보통 비교연산의 결과값으로 주어짐.
- TRUE와 FALSE는 덧셈과 같은 산술연산을 적용하면 TRUE는 1로 FALSE는 0으로 변환되어 계산에 사용.

V. 변수에 저장될 수 있는 값의 종류

- **논리형**

[코드 3-5]

```
5 > 3                # 비교연산
2 > 7                # 비교연산
TRUE + TRUE          # 산술연산에서 TRUE는 1
a <- TRUE            # a에 논리값 TRUE 저장
b <- F               # b에 논리값 FALSE 저장

a                    # a의 내용 출력
b                    # b의 내용 출력
a + b                # 논리값의 산술연산 결과
```

V. 변수에 저장될 수 있는 값의 종류

- **논리형**

[코드 3-5] 실행 결과

```
> 5 > 3          # 비교연산
[1] TRUE
> 2 > 7          # 비교연산
[1] FALSE
> TRUE + TRUE    # 산술연산에서 TRUE는 1
[1] 2
> a <- TRUE      # a에 논리값 TRUE 저장
> b <- F         # b에 논리값 FALSE 저장
>
> a              # a의 내용 출력
[1] TRUE
> b              # b의 내용 출력
[1] FALSE
> a + b          # 논리값의 산술연산 결과
[1] 1
```

V. 변수에 저장될 수 있는 값의 종류

▪ 특수한 값들

- **NULL :** '정의되지 않음'을 의미하는 특수한 값으로, 초깃값을 어떤 것으로 정해야 할지 애매할 때 사용.
- **초깃값(initial value)** : 변수 생성 시 최초로 넣는 값.

[코드 3-6]

```
addr <- NULL                # NULL 저장
print(addr)
```

[실행 결과]

```
> addr <- NULL              # NULL 저장
> print(addr)
NULL
```

V. 변수에 저장될 수 있는 값의 종류

▪ 특수한 값들

- **NA :** 'Not Applicable'의 약자로 결측값 또는 누락된 값을 나타낼 때 사용.

[코드 3-7]

```
a <- NA             # 결측값 저장
b <- 'NA'           # 문자형 값 저장
a                   # a의 내용 출력
b                   # b의 내용 출력
```

[실행 결과]

```
> a <- NA           # 결측값 저장
> b <- 'NA'         # 문자형 값 저장
> a                 # a의 내용 출력
[1] NA
> b                 # b의 내용 출력
[1] "NA"
```

V. 변수에 저장될 수 있는 값의 종류

▪ 특수한 값들

- **NaN, Inf, -Inf :** 각각 수학적으로 정의가 불가능한 값, 양의 무한대, 음의 무한대를 나타내는 특수한 값으로 프로그래밍에서 접하기 어려운 값

[코드 3-8]

```
1/0
-2/0
sqrt(-5)                    # -5의 제곱근
```

V. 변수에 저장될 수 있는 값의 종류

- **특수한 값들**

[코드 3-8] 실행 결과

```
> 1/0
[1] Inf
> -2/0
[1] -Inf
> sqrt(-5) # -5의 제곱근
[1] NaN
경고메시지(들):
In sqrt(-5) : NaN이 생성되었습니다
```

LAB. 소금물의 농도 구하기

이번 LAB에서는 변수와 cat() 함수를 활용해봅니다. [순서]를 참고하여 소금물의 농도 구하는 프로그램을 만들어봅시다.

[순서]

① 소금의 양을 변수 salt에 저장합니다.
② 물의 양을 변수 water에 저장합니다.
③ 소금물의 농도를 계산하고 그 결과를 result에 저장합니다.
④ 결과를 다음과 같은 형식으로 출력합니다(cat() 함수 이용).
"소금 = ( ) , 물 = ( ) : 농도 = ( ) "%

LAB. 소금물의 농도 구하기

1. 소금 50g과 물 100g을 섞었을 때 소금물의 농도(%) 구하는 명령문을 작성합니다.

```
salt <- 50
water <- 100
result <- salt / (salt + water) *100
cat("소금 =", salt, ", 물 =", water, " : 농도 =", result, "%")
```

```
…(생략)
소금 = 50 , 물 = 100 : 농도 = 33.33333 %
```

LAB. 소금물의 농도 구하기

2. 소금 70g과 물 110g을 섞었을 때 소금물의 농도(%) 구하는 명령문을 작성합니다.

```
salt <- 70
water <- 110
result <- salt / (salt + water) *100
cat("소금 =", salt, ", 물 =", water, " : 농도 =", result, "%")
```

```
…(생략)
소금 = 70 , 물 = 110 : 농도 = 38.88889 %
```

02

벡터란 무엇인가요?

02. 벡터란 무엇인가요?

I. 벡터의 개념

- 10명의 학생 성적을 변수에 저장한 후 평균을 계산하는 문제

[코드 3-9]

```
score.1 <- 68; score.2 <- 95; score.3 <- 83; score.4 <- 76;
score.5 <- 90
score.6 <- 80; score.7 <- 85; score.8 <- 91; score.9 <- 82;
score.10 <- 70
total <- score.1 + score.2 + score.3 + score.4 + score.5 +
           score.6 + score.7 + score.8 + score.9 + score.10
avg <- total / 10          # 10명의 평균 계산
avg                        # 평균 출력
```

I. 벡터의 개념

[코드 3-9] 실행 결과

```
> score.1 <- 68; score.2 <- 95; score.3 <- 83; score.4 <- 76;
score.5 <- 90
> score.6 <- 80; score.7 <- 85; score.8 <- 91; score.9 <- 82;
score.10 <- 70
> total <- score.1 + score.2 + score.3 + score.4 + score.5 +
+ score.6 + score.7 + score.8 + score.9 + score.10
> avg <- total / 10      # 10명의 평균 계산
> avg                    # 평균 출력
[1] 82
```

02. 벡터란 무엇인가요?

I. 벡터의 개념

[코드 3-10]

```
score <- c(68, 95, 83, 76, 90, 80, 85, 91, 82, 70)
mean(score)                 # 평균 출력
```

[실행 결과]

```
> score <- c(68, 95, 83, 76, 90, 80, 85, 91, 82, 70)
> mean(score)               # 평균 출력
[1] 82
```

02. 벡터란 무엇인가요?

I. 벡터의 개념

- **벡터(vector)** : R에서 제공하는 여러 개의 값을 한꺼번에 저장하는 기능으로, 일반적인 프로그래밍 용어로는 1차원 배열(array)이라고도 함
- 벡터는 다음과 같이 동일한 성격을 갖는 값들이 여러 개 있는 경우 이를 저장하고 처리하는 데 사용함.
    - 1학년 학생들의 몸무게 자료
    - 2학년 학생들의 영어 성적 자료
    - 1학년 학생들의 선호하는 색상 자료

02. 벡터란 무엇인가요?

I. 벡터의 개념

- 벡터의 목적은 여러 개의 값들을 하나의 묶음으로 처리하기 위한 것

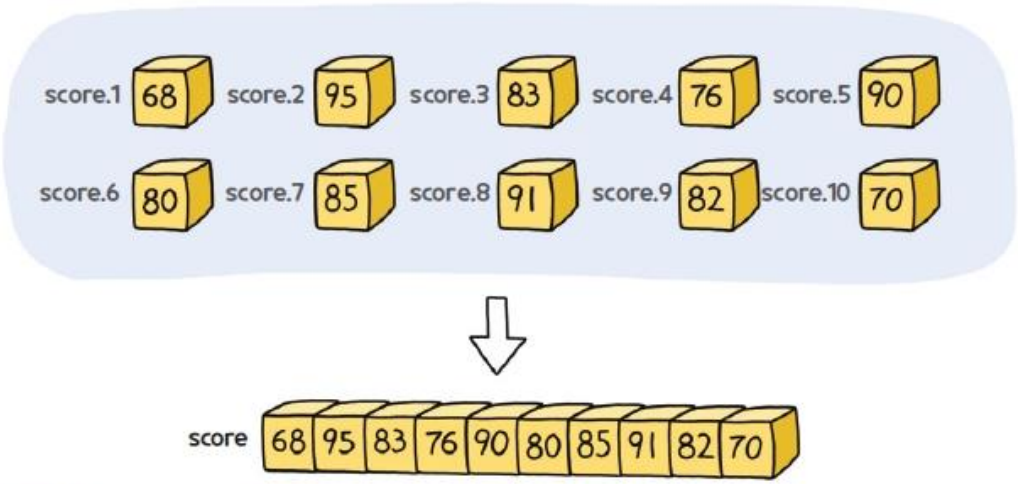


그림 3-5 10개의 일반 변수와 1개의 벡터

02. 벡터란 무엇인가요?

II. 벡터 만들기

[코드 3-11]

```
x <- c(1,2,3)                          # 숫자형 벡터
y <- c('a','b','c')                    # 문자형 벡터
z <- c(TRUE,TRUE,FALSE,TRUE)           # 논리형 벡터
x                                      # 벡터 x에 저장된 값을 출력
y                                      # 벡터 y에 저장된 값을 출력
z                                      # 벡터 z에 저장된 값을 출력
```

II. 벡터 만들기

[코드 3-11] 실행 결과

```
> x <- c(1,2,3)                      # 숫자형 벡터
> y <- c('a','b','c')                # 문자형 벡터
> z <- c(TRUE,TRUE, FALSE, TRUE) # 논리형 벡터
> x                                  # 벡터 x에 저장된 값을 출력
[1] 1 2 3
> y                                  # 벡터 y에 저장된 값을 출력
[1] "a" "b" "c"
> z                                  # 벡터 z에 저장된 값을 출력
[1] TRUE TRUE FALSE TRUE
```

- 벡터를 만드는 데 사용되는 함수는 c() 함수는 하나의 벡터에 여러 개의 값이 저장하는데, 벡터 생성 시 주의할 점은 하나의 벡터에는 동일한 자료형(data type)의 값이 저장되어야 함

```
> w <- c(1,2,3, 'a','b','c')
> w
[1] "1" "2" "3" "a" "b" "c"
```

II. 벡터 만들기

- **연속적인 숫자로 이루어진 벡터**

```
> v1 <- 50:90
> v1
 [1] 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
[23] 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
```

– 콜론(:)을 이용하면 연속된 정수로 이루어진 벡터 지정

```
> v2 <- c(1,2,5, 50:90)
> v2
 [1]  1  2  5 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68
[23] 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
```

– 콜론(:)을 이용하여 벡터를 만드는 방법은 c( ) 함수 안에서 사용해도 됨

II. 벡터 만들기

- **일정한 간격의 숫자로 이루어진 벡터**

```
> v3 <- seq(1,101,3)                          # 시작, 종료, 간격
> v3
 [1]  1  4  7 10 13 16 19 22 25 28 31 34 37 40 43 46
[17] 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91 94
[33] 97 100
>
> v4 <- seq(0.1,1.0,0.1)                      # 시작, 종료, 간격
> v4
 [1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

  - 일정한 간격의 숫자로 이루어진 벡터를 만들어야 할 때 seq(시작값, 종료값, 간격)

II. 벡터 만들기

- **반복된 숫자로 이루어진 벡터**

```
> v5 <- rep(1,times=5)            # 1을 5번 반복
> v5
[1] 1 1 1 1 1
>
> v6 <- rep(1:5,times=3)          # 1, 2, 3, 4, 5를 3번 반복
> v6
 [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
>
> v7 <- rep(c(1,5,9), times=3)    # 1, 5, 9를 3번 반복
> v7
[1] 1 5 9 1 5 9 1 5 9
```

– 반복된 숫자로 이루어진 벡터는 rep(반복대상값, times=반복횟수) 함수

II. 벡터 만들기

- **반복된 숫자로 이루어진 벡터**

```
> v8 <- rep(c('a','b','c'), each = 3)
> v8
[1] "a" "a" "a" "b" "b" "b" "c" "c" "c"
```

  - 매개변수 times 대신에 each 사용

III. 벡터의 값에 이름 붙이기

[코드 3-12]

```
absent <- c(8,2,0,4,1)    # absent 벡터에 결근 인원수 저장
absent                    # absent 벡터의 내용 출력
names(absent)             # absent 벡터의 값들의 이름을 확인
names(absent) <- c('Mon','Tue','Wed','Thu','Fri') # 값들의 이름을 입력
absent                    # absent 벡터의 내용 출력
names(absent)             # absent 벡터의 값들의 이름을 확인
```

[코드 3-12] 실행 결과(1)

```
> absent <- c(8,2,0,4,1) # absent 벡터에 결근 인원수 저장
> absent                 # absent 벡터의 내용 출력
[1] 8 2 0 4 1
```

– 벡터 absent에 5개의 결근 인원수를 입력한 후 벡터의 내용을 출력

III. 벡터의 값에 이름 붙이기

[코드 3-12] 실행 결과(2)

```
> names(absent)                    # absent 벡터의 값들의 이름을 확인
NULL
```

– 벡터 absent에 입력된 값들의 이름을 출력

[코드 3-12] 실행 결과(3)

```
> names(absent) <- c('Mon','Tue','Wed','Thu','Fri') # 값들의 이름을 입력
```

– names() 함수를 이용하여 5개의 값에 요일을 이름으로 지정하는 명령문

III. 벡터의 값에 이름 붙이기

[코드 3-12] 실행 결과(4)

```
> absent                          # absent 벡터의 내용 출력
Mon Tue Wed Thu Fri
  8   2   0   4   1
```

– 지정된 이름이 값들과 함께 출력

[코드 3-12] 실행 결과(5)

```
> names(absent)                   # absent 벡터의 값들의 이름을 확인
[1] "Mon" "Tue" "Wed" "Thu" "Fri"
```

– 지정된 이름들이 출력

02. 벡터란 무엇인가요?

IV. 벡터에서 원소값 확인하기

- **인덱스(index)** : R에서 벡터에 저장된 각각의 값들을 구별하기 위하여 앞쪽의 값부터 순서를 부여하는 것.

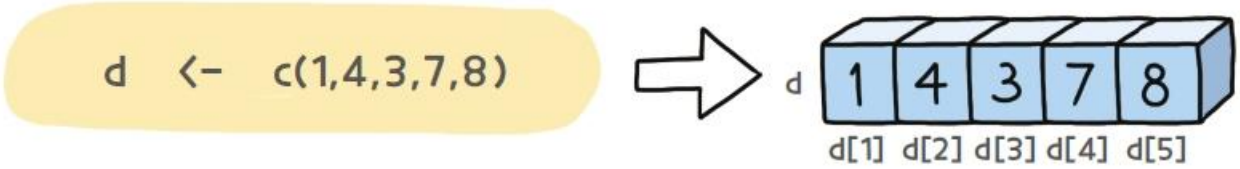


그림 3-6 벡터의 인덱스

IV. 벡터에서 원소값 확인하기

[코드 3-13]

```
d <- c(1,4,3,7,8)
d                  # 벡터 전체를 출력
d[1]
d[2]
d[3]
d[4]
d[5]
d[6]
```

IV. 벡터에서 원소값 확인하기

[코드 3-13] 실행 결과

```
> d <- c(1,4,3,7,8)
> d                 # 벡터 전체를 출력
[1] 1 4 3 7 8
> d[1]
[1] 1
> d[2]
[1] 4
> d[3]
[1] 3
> d[4]
[1] 7
> d[5]
[1] 8
> d[6]
[1] NA
```

IV. 벡터에서 원소값 확인하기

- **벡터에서 여러 개의 값을 한번에 추출하기**

[코드 3-14]

```
d <- c(1,4,3,7,8)
d[c(1,3,5)]                 # 1, 3, 5번째 자료 출력
d[1:3]                      # 1, 2, 3번째 자료 출력
d[seq(1,5,2)]               # 홀수 번째 자료 출력
d[-2]                       # -는 '제외하고'의 의미
d[-c(3:5)]                  # 세 번째에서 다섯 번째 값은 제외
```

IV. 벡터에서 원소값 확인하기

- **연속적인 숫자로 이루어진 벡터**

[코드 3-14] 실행 결과(1)

```
> d <- c(1,4,3,7,8)
```

– 벡터 d에 5개의 값을 저장

[코드 3-14] 실행 결과(2)

```
> d[c(1,3,5)]                    # 1, 3, 5번째 자료 출력
[1] 1 3 8
```

– 여러 개의 인덱스를 한번에 지정하여 값을 가져오는 가장 일반적인 방법

IV. 벡터에서 원소값 확인하기

- **연속적인 숫자로 이루어진 벡터**

[코드 3-14] 실행 결과(3)

```
> d[1:3]                    # 1, 2, 3번째 자료 출력
[1] 1 4 3
```

– 가져오고 싶은 값들이 연속해서 존재하면 콜론(:)을 이용하여 인덱스를 지정

[코드 3-14] 실행 결과(4)

```
> d[seq(1,5,2)] # 홀수 번째 자료 출력
[1] 1 3 8
```

– 1부터 5까지의 값을 2씩 건너뛰어 가져온다는 의미

IV. 벡터에서 원소값 확인하기

- **연속적인 숫자로 이루어진 벡터**

[코드 3-14] 실행 결과(5)

```
> d[-2]          # -는 '제외하고'의 의미
[1] 1 3 7 8
```

– 두 번째를 제외하고 나머지 값들을 모두 가져오라는 의미

[코드 3-14] 실행 결과(6)

```
> d[-c(3:5)]     # 세 번째에서 다섯 번째 값은 제외
[1] 1 4
```

– 세 번째, 네 번째, 다섯 번째를 제외한 값을 가져오게 됨

IV. 벡터에서 원소값 확인하기

- 이름으로 값을 추출하기

[코드 3-15]

```
sales <- c(640,720,680,540)                    # 1~4월 매출액
names(sales) <- c('M1','M2','M3','M4') # 매출액에 월을 이름으로 붙임
sales                                          # 1~4월 매출액 출력
sales[1]                                       # 1월 매출액 출력
sales['M2']                                    # 2월 매출액 출력
sales[c('M1','M4')]                            # 1월, 4월 매출액 출력
```

IV. 벡터에서 원소값 확인하기

- **이름으로 값을 추출하기**

[코드 3-15] 실행 결과(1)

```
> sales <- c(640,720,680,540)     # 1~4월 매출액
> names(sales) <- c('M1','M2','M3','M4') # 매출액에 월을 이름으로 붙임
> sales                           # 1~4월 매출액 출력
 M1  M2  M3  M4
640 720 680 540
```

– sales 벡터에 1~4월 매출액을 저장하고 월을 이름으로 지정

IV. 벡터에서 원소값 확인하기

- **이름으로 값을 추출하기**

[코드 3-15] 실행 결과(2)

```
> sales[1]                    # 1월 매출액 출력
 M1
640
```

– 인덱스를 이용하여 1월의 매출액을 출력

[코드 3-15] 실행 결과(3)

```
> sales['M2']                  # 2월 매출액 출력
 M2
720
```

– 이번에는 인덱스 대신 값의 이름을 이용하여 2월 매출액을 출력

IV. 벡터에서 원소값 확인하기

- **이름으로 값을 추출하기**

[코드 3-15] 실행 결과(4)

```
> sales[c('M1','M4')]    # 1월, 4월 매출액 출력
 M1  M4
640 540
```

– 두 개의 월 이름을 c( ) 함수로 묶어서 인덱스 대신 지정

V. 벡터에 저장된 원소값 변경하기

[코드 3-16]

```
v1 <- c(1,5,7,8,9)
v1
v1[2] <- 3                    # v1의 두 번째 값을 3으로 변경
v1
v1[c(1,5)] <- c(10,20)        # v1의 1, 5번째 값을 각각 10, 20으로 변경
v1
v1 <- c(100,200,300)          # v1의 내용을 100, 200, 300으로 변경
```

V. 벡터에 저장된 원소값 변경하기

[코드 3-16] 실행 결과(1)

```
> v1 <- c(1,5,7,8,9)
> v1
[1] 1 5 7 8 9
```

– 벡터 v1에 5개의 값을 저장한 후 v1의 내용을 출력

[코드 3-16] 실행 결과(2)

```
> v1[2] <- 3                    # v1의 두 번째 값을 3으로 변경
> v1
[1] 1 3 7 8 9
```

– 벡터 v1의 두 번째 원소값을 3으로 변경한 후 v1의 내용을 출력

V. 벡터에 저장된 원소값 변경하기

[코드 3-16] 실행 결과(3)

```
> v1[c(1,5)] <- c(10,20)    # v1의 1, 5번째 값을 각각 10, 20으로 변경
> v1
[1] 10 3 7 8 20
```

– 벡터 v1의 첫 번째와 다섯 번째 원소값을 10과 20으로 변경한 후 v1의 내용을 출력

[코드 3-16] 실행 결과(4)

```
> v1 <- c(100,200,300)   # v1의 내용을 100, 200, 300으로 변경
> v1
[1] 100 200 300
```

– 이 예는 벡터 v1의 내용을 통째로 다른 값으로 변경

LAB. 적금 만기 금액 계산하기

다음 표에서 5명의 사람은 서로 다른 연이율(%)과 예금 기간을 가진 정기적금에 가입했습니다. 각각의 사람이 만기시 받을 금액은 얼마인지 계산하는 명령문을 작성해 봅니다. 이때, 이율은 복리로 계산합니다.

이름	kim	lee	park	choi	seo
원금	5,000,000	4,500,000	4,000,000	5,500,000	6,000,000
이율	3.5%	3%	4%	5%	4.5%
기간	2	2	5	7	4

[복리계산식]

만기 금액 = 원금 × (1+ 연이율/100)^{기간}

LAB. 적금 만기 금액 계산하기

1. 이름, 원금, 이율, 기간을 각각의 벡터로 만듭니다.

```
customer <- c('kim', 'lee', 'park', 'choi', 'seo')
deposit <- c(5000000, 4500000, 4000000, 5500000, 6000000)
rate <- c(3.5, 3, 4, 5, 4.5)
period <- c(2, 2, 5, 7, 4)
```

2. 원금, 이율, 기간 벡터 이름에 각 사람의 이름을 설정합니다.

```
names(deposit) <- customer
names(rate) <- customer
names(period) <- customer
```

3. 사람을 지정할 수 있는 변수 who를 만듭니다.

```
who <- 'kim'
```

LAB. 적금 만기 금액 계산하기

4. 만기 금액을 계산하는 식을 작성합니다.

```
sum <- deposit[who] * ( 1 + rate[who] / 100)^ period[who]
sum
```

```
    kim
5356125
```

5. 변수 who에 다른 사람의 이름을 넣고 4번 명령문을 실행하면 그 사람의 결과를 바로 얻을 수 있습니다. 만약 변수 who에 벡터 customer를 대입하고 4번 명령문을 실행하면 모든 고객에 대한 결과를 확인할 수 있습니다.

03

함수를 소개합니다

03. 함수를 소개합니다

I. 함수의 개념

- 함수를 포함한 식은 다음과 같은 형태로 표현됨

$$y = f(x)$$

- f : 함수
- $f(x)$: x를 입력하고 함수를 실행한 결과값
- x : 함수의 입력값(input)
- y : 함수를 실행한 결과값을 받는 변수

- 어떤 값 x를 입력받아 정해진 계산을 수행한 후 그 결과값 f(x)를 돌려주는 장치라고 표현

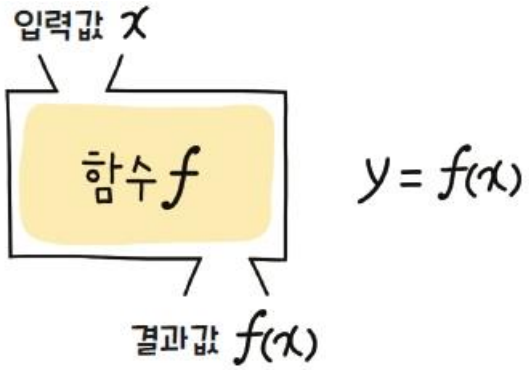


그림 3-7 함수의 개념

03. 함수를 소개합니다

I. 함수의 개념

- 함수 f( )를 다음과 같이 정의하면,

  입력값 x=1일 때 결과값은 f(1)=3,

  x=2일 때 결과값은 f(2)=5가 됨

```
f(x) = 2x + 1
```

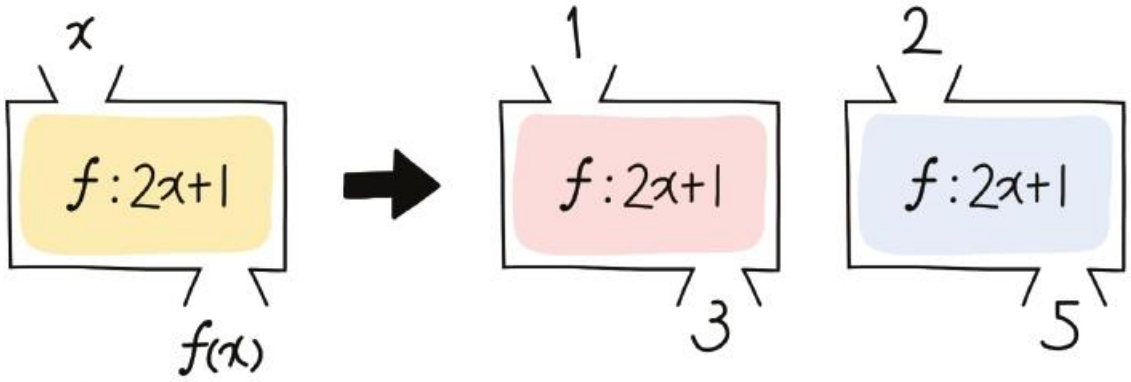


그림 3-8 함수 f(x)=2x+1의 실행

03. 함수를 소개합니다

I. 함수의 개념

[코드 3-17]

```
y <- sqrt(100)   # 100의 제곱근을 구하여 변수 y에 저장
y                # y에 저장된 값을 출력
```

[실행 결과]

```
> y <- sqrt(100) # 100의 제곱근을 구하여 변수 y에 저장
> y              # y에 저장된 값을 출력
[1] 10
```

– 100이라는 값을 sqrt( ) 함수에 입력하면 그 결과값으로 10을 돌려줌

03. 함수를 소개합니다

I. 함수의 개념

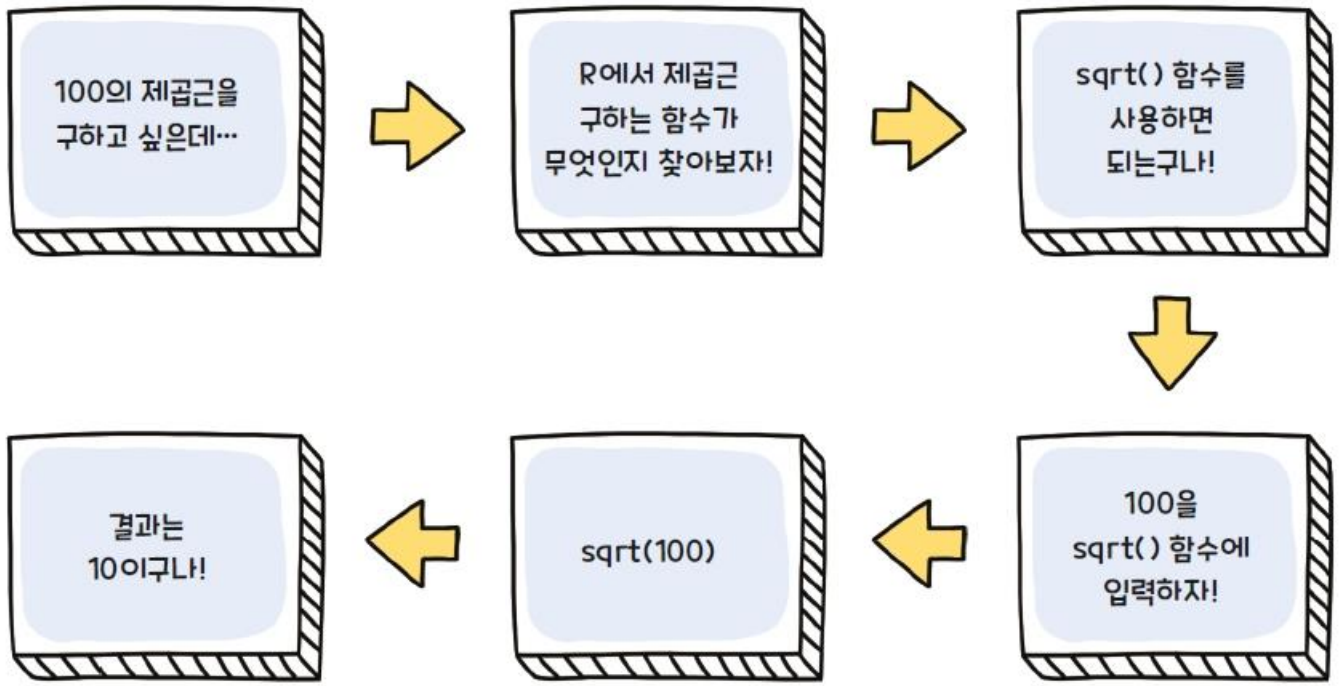


그림 3-9 함수를 이용하여 문제를 해결하는 과정

II. 함수의 매개변수

- 프로그래밍에서 함수의 입력값을 받는 변수를 매개변수(parameter)라 함
- 함수의 정의에 맞추어 매개변수를 입력하면 정의된 결과값을 얻을 수 있음

▪ 매개변수의 입력

[코드 3-18]

```
d <- c(1,7,4,2,3)                # 벡터 d에 5개의 값을 저장
sort(d)                          # 벡터 d의 값들을 오름차순으로 정렬하여 출력
sort(d,decreasing=TRUE)          # 벡터 d의 값들을 내림차순으로 정렬하여 출력
```

II. 함수의 매개변수

▪ 매개변수의 입력

[코드 3-18] 실행 결과(1)

```
> d <- c(1,7,4,2,3)              # 벡터 d에 5개의 값을 저장
> sort(d)                        # 벡터 d의 값들을 오름차순으로 정렬하여 출력
[1] 1 2 3 4 7
```

– 벡터 d에 5개의 값을 저장한 후 이를 sort( ) 함수의 입력값으로 함

– 결과값으로 벡터 d의 값들을 오름차순으로 정렬하여 출력함

II. 함수의 매개변수

▪ 매개변수의 입력

[코드 3-18] 실행 결과(2)

```
> sort(d,decreasing=TRUE)      # 벡터 d의 값들을 내림차순으로 정렬하여 출력
[1] 7 4 3 2 1
```

– 벡터 'd'와 'decreasing=TRUE'를 입력값으로 함

– 결과값으로 벡터 d의 값들을 내림차순으로 정렬하여 출력

<TIP>

① R 함수에서는 숫자뿐만 아니라 벡터도 함수의 입력값으로 넣을 수 있고, 결과값도 숫자가 아닌 벡터가 될 수 있음.

② sort( ) 함수는 매개변수를 하나만 입력하는 경우도 있고 두 개를 입력하는 경우도 있습니다. 이것이 수학 시간에 배운 함수와 다른 점.

II. 함수의 매개변수

▪ 매개변수의 입력

- 벡터 d의 값들을 오름차순으로 정렬하는 정확한 명령문

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

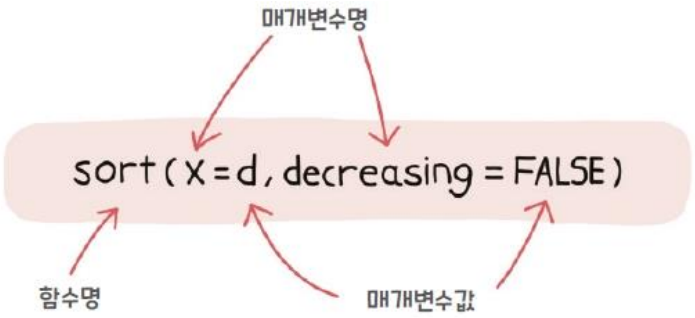


그림 3-10 함수와 매개변수

- x와 decreasing을 매개변수명이라고 하고 d와 FALSE를 매개변수값이라 함
- 함수의 입력값을 'x=d', 'decreasing=FALSE'와 같이 '매개변수명=매개변수값'의 형태로 입력

II. 함수의 매개변수

▪ 매개변수의 생략

```
sort(x=d, decreasing = FALSE)     # 정식 문법
sort(d, FALSE)                    # 매개변수 이름의 생략
```

- 매개변수값을 함수에 지정된 순서대로 입력하는 경우, 매개변수명을 생략 가능

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

II. 함수의 매개변수

▪ 매개변수의 생략

- 매개변수는 반드시 입력해야 하는 필수 매개변수와 선택적 매개변수로 구분하는데, 선택적 매개변수는 생략 가능
- 선택적 매개변수의 경우는 사용자가 생략할 경우를 대비해서 미리 어떤 값을 지정해 놓는데, 이런 값을 기본값(default value)이라고 함

```
sort(x=d, decreasing = FALSE)    # 정식 문법
sort(x=d)                        # 선택적 매개변수의 생략
sort(d)                          # 매개변수명의 생략
```

LAB. paste() 함수로 문자열 만들기

paste( ) 함수를 이용하여 원하는 문자열을 만들어보겠습니다.

1. **paste( )의 기본적인 사용법은 다음과 같습니다.**

```
str <- paste('good', 'morning', sep=' / ')
str
```

```
...(생략)
[1] "good / morning"
```

   - 문자열 good과 morning을 결합하여 변수 str에 저장한 후 str을 출력

LAB. paste() 함수로 문자열 만들기

2. 변수 a, b, c에 다음과 같은 값이 저장되어 있다고 할 때 a, b, c를 paste( ) 함수로 연결하여 출력해봅시다.

```
a <- '나의 나이는'
b <- 20
c <- '입니다'
paste(a, b, c, sep=' ')
```

```
...(생략)
[1] "나의 나이는 20 입니다"
```

LAB. paste() 함수로 문자열 만들기

3. 벡터 a와 값 b를 연결하면 벡터 a에 있는 값을 차례로 가져와서 b와 연결한 뒤 새로운 벡터에 저장합니다.

```
a <- 1:12
b <- '월'
c <- paste(a,b, sep='')
c
```

```
...(생략)
 [1] "1월"  "2월" "3월" "4월" "5월" "6월" "7월" "8월" "9월" "10월"
[11] "11월" "12월"
```

실전분석. 월별 매출액 분석하기

[문제]

A 기업의 월별 매출액 자료를 바탕으로 매출액을 분석해보겠습니다.

1월	2월	3월	4월	5월	6월
750	740	760	680	700	710
7월	8월	9월	10월	11월	12월
850	890	700	720	690	730

[해결]

1. **매출액을 분석하기 위해 자료를 sales에 저장합니다.**

```
sales <- c(750,740,760,680,700,710,850,890,700,720,690,730)
names(sales) <- paste(1:12, '월', sep="")
sales
```

실전분석. 월별 매출액 분석하기

2. 7월의 매출액을 알아봅니다.

```
sales['7월']
```

```
7월
850
```

3. 1월과 2월의 매출액 합계를 알아봅니다.

```
sales['1월']+sales['2월']
```

```
 1월
1490
```

4. sort() 함수를 이용해서 매출액이 가장 많은 달을 알아봅니다.

```
max.month <- sort(sales, decreasing=T)
max.month[1]
```

```
8월
890
```

실전분석. 월별 매출액 분석하기

5. sum() 함수를 이용해서 상반기 매출액의 합계를 알아봅니다.

```
sum(sales[1:6])
```

```
[1] 4340
```

Thank You !

한빛아카데미 Hanbit Academy, Inc.