Part 1. R 프로그래밍 (데이터 분석 전문가 양성과정)

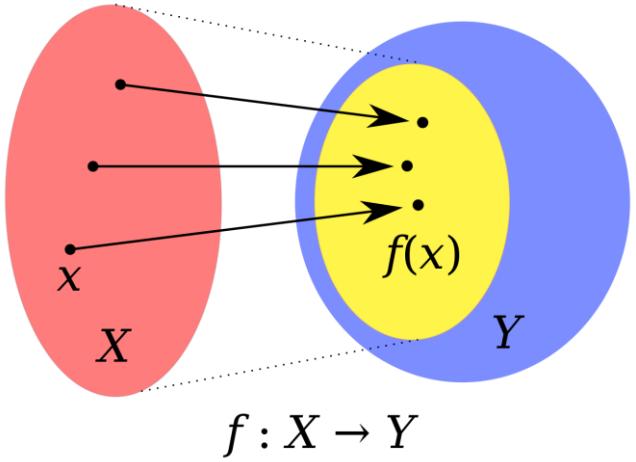
05

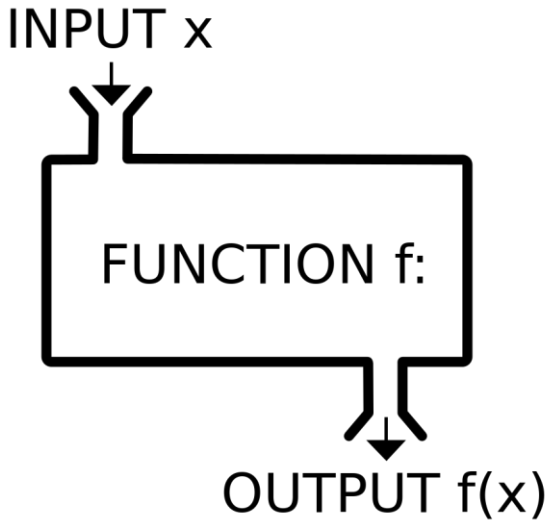
함수의 이해

경북대학교 배준현 교수
(joonion@knu.ac.kr)

05. 함수의 이해

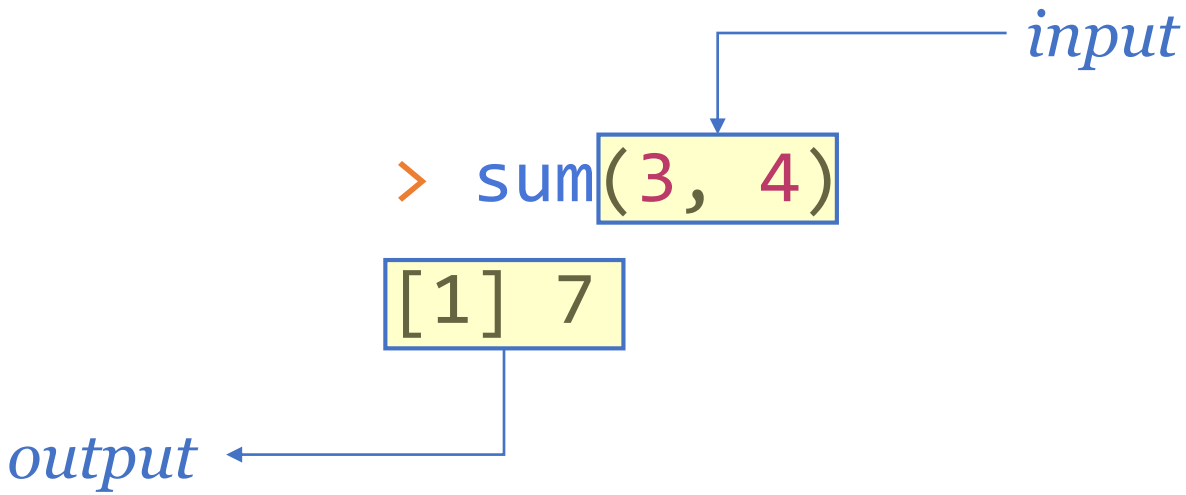
- **수학에서의 함수:** *function* in mathematics
  - 두 집합 X, Y에 대해서 X의 각 원소에 Y의 원소가 일대일 대응 관계일 때
    - 함수 $f: X \to Y$
  - $y = f(x)$: X의 원소 x와 Y의 원소 y의 대응 관계
    - x는 함수의 입력값(*input*), y는 함수의 출력값(*output*)





05. 함수의 이해

- **프로그래밍 언어에서의 함수:** *function* in programming language
  - 어떤 **입력값**에 대해서 계산된 **출력값**을 되돌려 주는 일련의 실행 절차
  - 다른 용어: *procedure*, *subroutine*



- **내장 함수:** *built-in* functions
  - 외부 패키지를 불러오지 않고도 쓸 수 있도록 기본적으로 제공되는 함수들

구분	함수 정의	출력값
수학 관련	`abs(x)`	절대값
	`sqrt(x)`	제곱근
	`log(x), log10(x)`	자연로그, 밑이 10인 로그
	`exp(x)`	자연 상수 e의 거듭제곱
	`sin(x), cos(x), tan(x)`	삼각 함수
	`ceiling(x), floor(x)`	천장값, 바닥값
	`round(x, digits = n)`	반올림수

구분	함수 정의	출력값
통계 관련	sum(x)	합계
	mean(x)	평균
	var(x)	분산
	sd(x)	표준편차
	median(x)	중앙값
	quantile(x, probs)	분위값
	min(x), max(x)	최솟값, 최댓값
	range(x)	범위값

- **사용자 정의 함수:** *user-defined* functions
  - 사용자가 직접 함수를 정의하여 사용할 수 있음

```
function.name <- function (parameters) {
    function.bodies
    return (return_value)
}
```

```
> add <- function (x, y) {
+     z <- x + y
+     return (z)
+ }
```

```
> add
function (x, y) {
    z <- x + y
    return (z)
}

> add(3, 4)
[1] 7
```

- **매개변수**: *parameters*, *arguments*
  - 함수의 입력값을 전달받는 변수
    - 다른 용어: 형식인자, 인자, 인수
  - 매개변수를 전달할 때 매개변수명과 매개변수값을 지정할 수 있음
    - 매개변수명을 전달하지 않으면 매개변수 순서에 따라 전달됨

```
> func1 <- function (x, y, z) {
+     return (x + 2 * y + 3 * z)
+ }

> func1(1, 2, 3)
[1] 14
> func1(x = 1, y = 2, z = 3)
[1] 14
> func1(3, 2, 1)
[1] 10
> func1(z = 3, x = 2, y = 1)
[1] 13
> func1(1, z = 2, y = 3)
[1] 13
```

- **매개변수의 기본값:** *default* value
  - 함수를 정의할 때 매개변수에 **기본값을 지정**할 수 있음
  - 기본값이 지정되어 있는 매개변수는 함수를 호출할 때 **생략 가능함**
  - 기본값이 지정되어 있지 않은 매개변수는 **생략하면 안됨**

```
> func2 <- function (x, y = 1, z = 0) {
+     return (x + 2 * y + 3 * z)
+ }
> func2(1, 2, 3)
[1] 14
> func2(1)
[1] 3
> func2(1, 2)
[1] 5
> func2(y = 2, z = 3)
Error in func2(y = 2, z = 3) : argument "x" is missing, with no default
> func2(2, 3, x = 1)
[1] 14
> func2(z = 1, x = 2)
[1] 7
```

- 여러 매개변수가 있는 내장 함수를 호출할 때 기본값이 있으면 생략 가능

```
> pi <- 3.141592
> round(x = pi, digits = 4)
[1] 3.1416
> round(pi, 2)
[1] 3.14
> round(digits = 4, x = pi)
[1] 3.1416
> round(digits = 3)
Error: argument "x" is missing, with no default
```

```
> head(iris)
……

> head(x = iris, n = 3)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa

> head(n = 3, iris)
……

> head(3, iris)
Error in checkHT(n, dx <- dim(x)) :
  invalid 'n' - must have length one when dim(x) is NULL, got 5
```

- return 문장을 생략하면 마지막 계산값이 리턴됨
- 블록에 들어갈 문장이 하나뿐이라면 {}는 생략 가능

```
> sadd1 <- function(x, y) {
+     return (x + y)
+ }
> sadd1(3, 4)
[1] 7

> sadd2 <- function(x, y) {
+     x + y
+ }
> sadd2(3, 4)
[1] 7

> sadd3 <- function(x, y) x + y
> sadd2(3, 4)
[1] 7
```

- 매개변수는 벡터 등의 다른 R 데이터 오브젝트로 지정 가능

```
> square <- function(x) x ^ 2
> square(1:10)
[1]   1   4   9  16  25  36  49  64  81 100

> vadd <- function(x, y) x + y
> vadd(1:3, 3:1)
[1] 4 4 4

> vmult <- function(x, y = 0) x * y
> vmult(1:3)
[1] 0 0 0
> vmult(1:3, 3:1)
[1] 3 4 3
```

- **함수형 프로그래밍:** *functional programming*
  - 모든 코드를 함수를 위주로 구현하고자 하는 프로그래밍 패러다임
    - 조건문과 반복문을 매우 싫어함
  - R 코드는 함수형 프로그래밍으로 많이 작성함
    - 조건문을 대체할 함수: ifelse()
    - 반복문을 대체할 함수: apply() 계열 함수 등

- ifelse() 함수
  - R에서 대부분의 조건문은 ifelse() 함수로 대체 가능

```
ifelse(condition, true.value, false.value)
```

```r
x <- 10
y <- 20
ifelse(x < y, x, y)
ifelse(x > y, x, y)
```

```r
score <- 88
grade <- ifelse(score >= 90,
                "A",
                ifelse(score >= 80,
                       "B",
                       "C"))
grade
```

- sapply() 함수
  - 벡터의 각 원소에 어떤 함수를 적용한 결과를 벡터로 리턴

```
sapply(x, FUN, ...)
```

```r
is.odd <- function(n) n %% 2 == 1
is.odd(7)
```

```r
odd.cnt.1 <- function (n, m) {
    count <- 0
    for (i in n:m) {
        if (is.odd(i))
            count <- count + 1
    }
    count
}
odd.cnt.1(10, 20)
```

```r
is.odd <- function(n) n %% 2 == 1
is.odd(7)
```

```r
odd.cnt.2 <- function(n, m) {
    sum(sapply(n:m, is.odd))
}
odd.cnt.2(10, 20)
```

- **연습문제 4.1:**
  - 임의의 자연수 n에 대하여
    - 약수의 개수를 구하는 `div.cnt()` 함수를 작성하시오.
      - 입력값: 임의의 자연수 n
      - 출력값: n의 약수의 개수
    - 1에서 15까지의 n에 대해서 약수의 개수를 확인하시오.

- **연습문제 4.2:**
  - 임의의 자연수 n에 대하여
    - 1부터 n까지 소수의 개수를 리턴하는 `prime.cnt()` 함수를 작성하시오.
      - 입력값: 임의의 자연수 n
      - 출력값: 1부터 n까지 소수의 개수
    - n이 $10, 100, 1000, 10000, 100000$일 때 소수의 개수를 확인하시오.
      - 소수인지 판단하는 `is.prime()` 함수를 먼저 작성하시오.
      - `is.prime()` 함수를 이용하여 `prime.cnt()` 함수를 작성하시오.
    - `is.prime()`에서 $\lceil \sqrt{n} \rceil$까지만 확인해도 됨을 확인하고,
      - 내장 함수를 이용해서 `is.prime()`을 수정하시오.

Any Questions?