Econometrics 1 *Applied Econometrics with R*

Lecture 4: Programming with R

黄嘉平

中国经济特区研究中心 讲师

办公室：文科楼1726

E-mail: huangjp@szu.edu.cn

Tel: (0755) 2695 0548

Office hour: Mon./Tue. 13:00-14:00

Continued from last week
Plot a function (2): a faster way

- We have plotted the density function of the standard normal distribution.

- This function can be calculated with the built-in command `dnorm()`

- Use `curve()` to draw the function

```
> curve(dnorm, from = -3, to = 3)
```

- Try it!

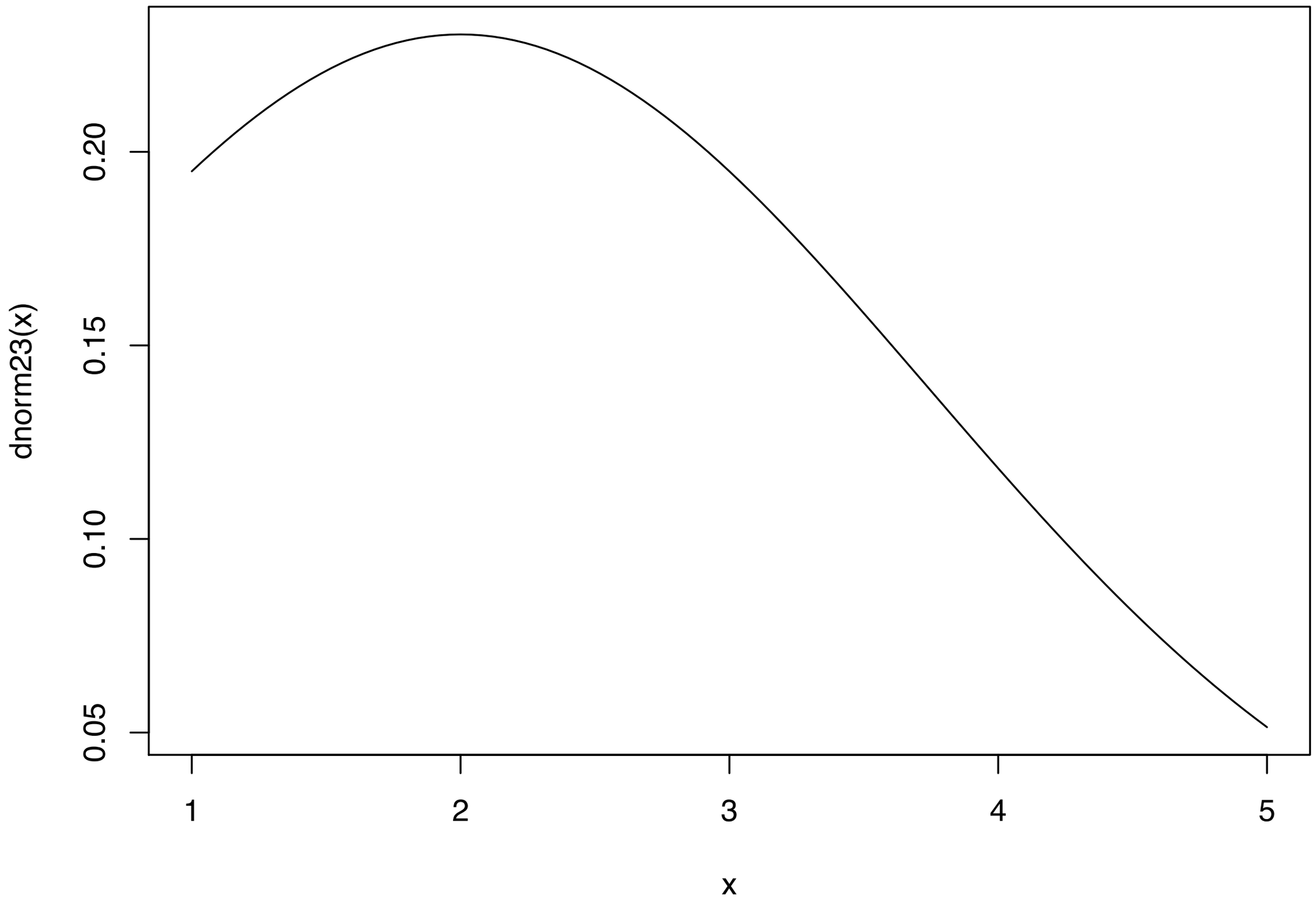
Define an R function

- You can define your own functions.

- For example, the density of the normal distribution with mean 2 and variance 3 under the name `dnorm23` can be defined as

```
> dnorm23 <- function (x) {exp(-
(x-2)^2 / (2*3)) / sqrt(2 * 3 * pi)}
> dnorm23(1)
```

- Plot a curve with `dnorm23` on domain $[1, 5]$


dnorm23(x)
0.05
0.10
0.15
0.20
1
2
3
4
5
x

Function

- General expression of a function

defined by user

```
function_name <- function (x) {
   ……
   return(…)
}
```

- If the `return` statement is missing, the value of the last evaluated expression is returned.

Programming with R

What is a program?

- One line command is a program, e.g.

```
> fractal(10)
> 1 / sqrt(2 * pi) * exp(- 0.5^2 / 2)
```

- More complicated programs are combinations of basic commands, plus some *controlling* statements, e.g., `if`, `for`, etc.

Conditional: the `if` statement

- Logical conditions: `<, <=, >, >=, ==, !=`

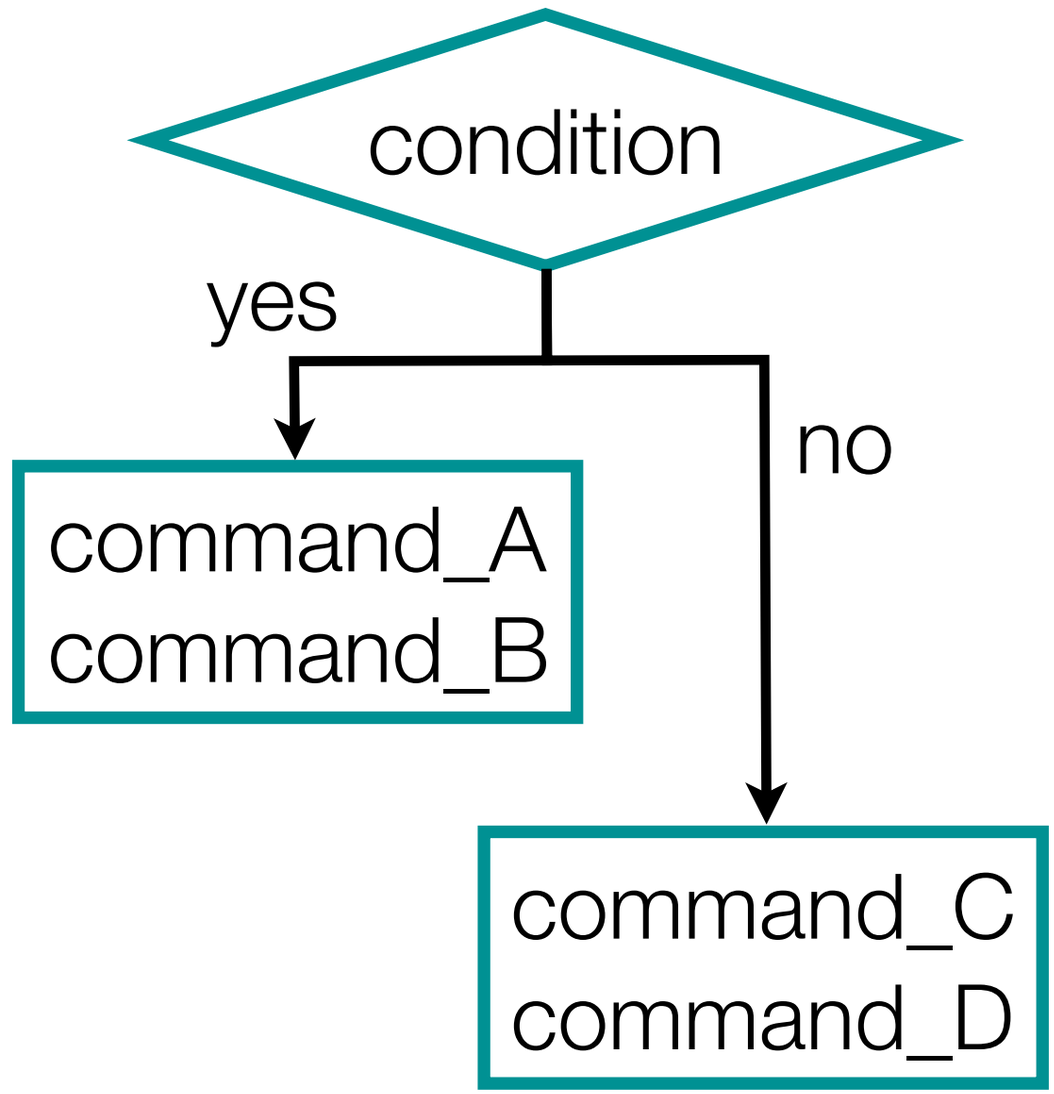
```
> 3 < 5
> class(3 < 5)
```

- `if (`*condition*`)` *command_A*
  `if (`*condition*`)` *command_A* `else` *command_B*

```
> x <- rbinom(1, 1, 0.5)
> if (x == 0) y <- 1 else y <- 0
```

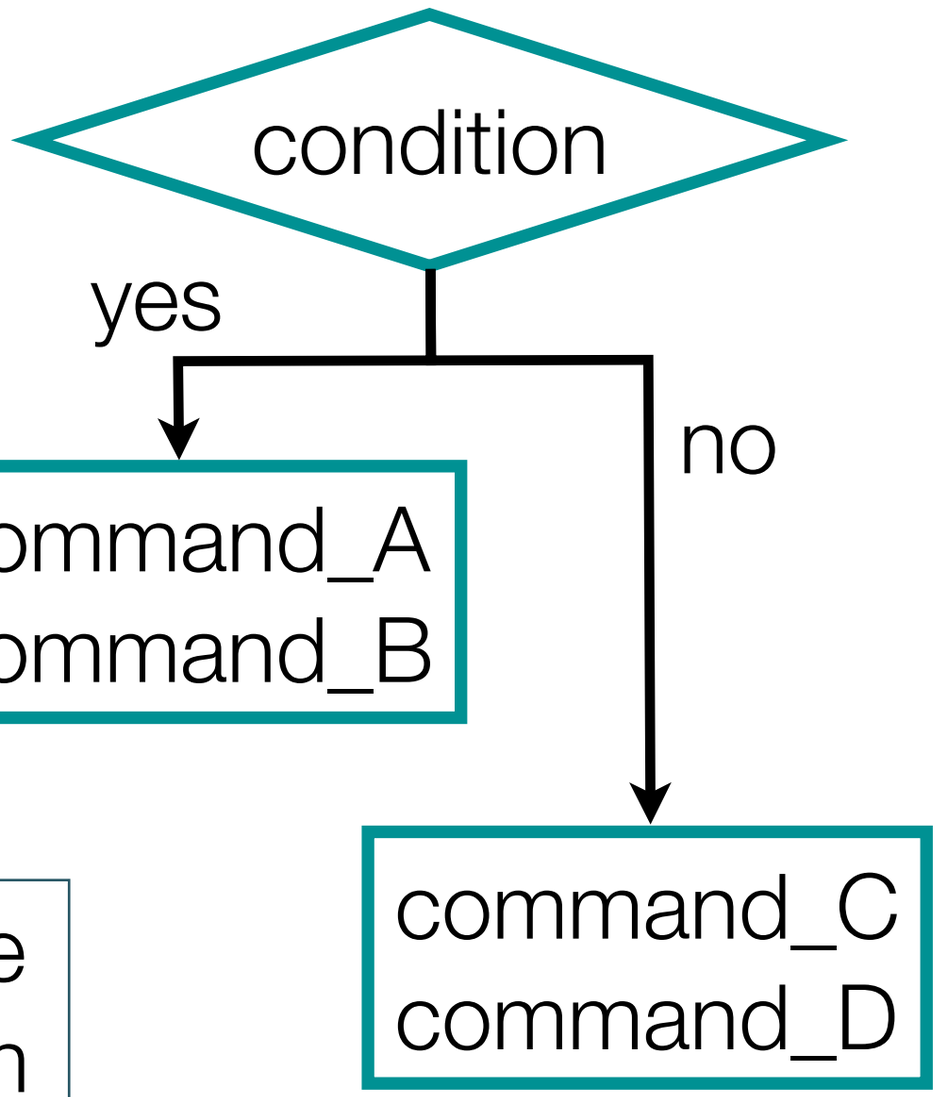
- `if` with commands in multiple lines (in a script file)

```
if (condition) {
     command_A
     command_B
     ……
} else {
     command_C
     command_D
     ……
}
```



- `if` with commands in multiple lines (in a script file)

```
if (condition) {
␣␣␣␣command_A
␣␣␣␣command_B
␣␣␣␣.....
} else {
␣␣␣␣command_C
␣␣␣␣command_D
␣␣␣␣.....
}
```



Use *tab* or *space* here to make your program readable (automatically added in RStudio).

Practice: the absolute value

- Define a function named `absvalue` that calculates the absolute value of a real number.
- Use `if` statement in your function.

Practice: the absolute value

- Define a function named `absvalue` that calculates the absolute value of a real number.

- Use `if` statement in your function.

```
absvalue <- function (x) {
    if (x > 0) x else -x
}
```

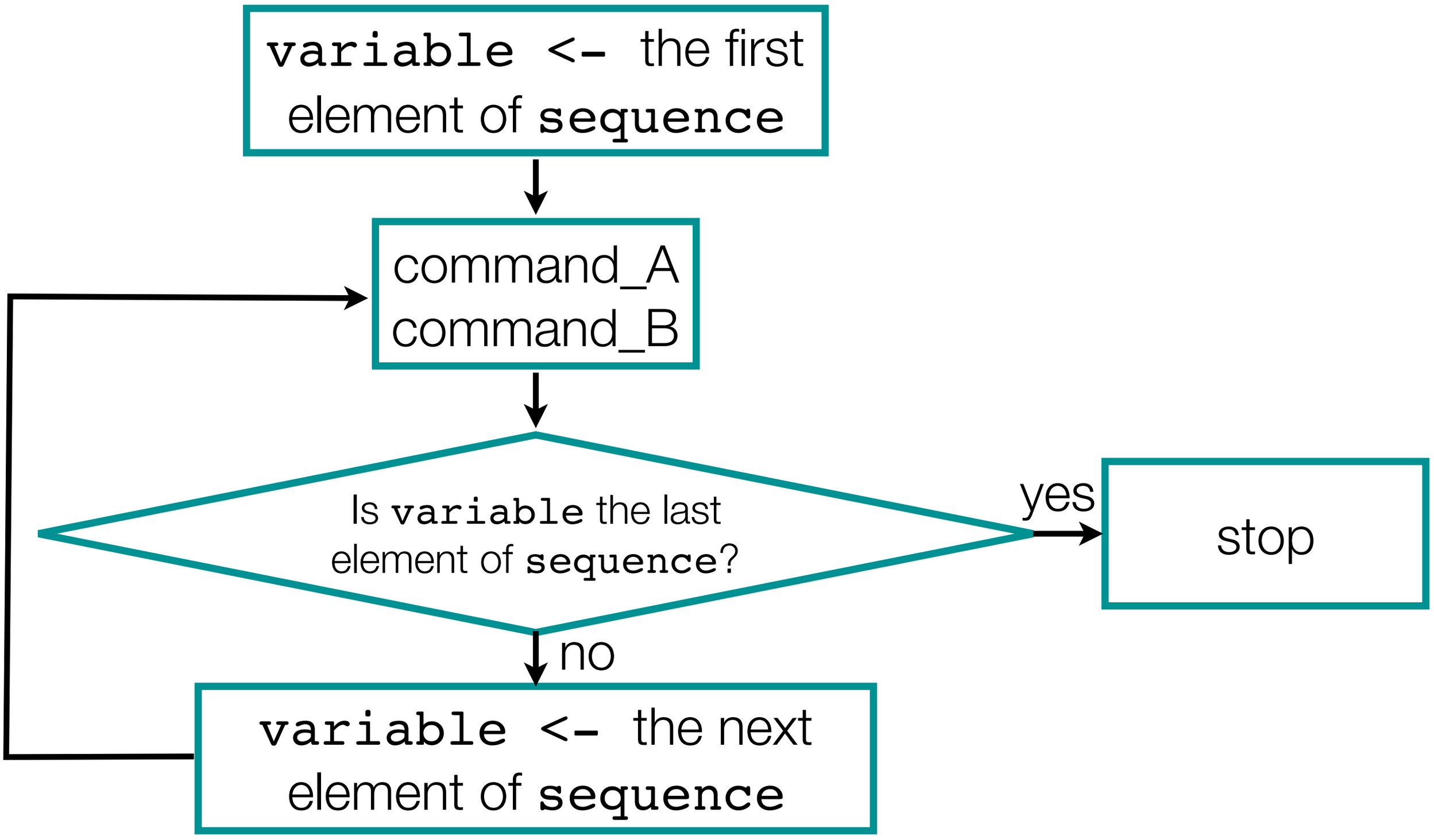
Iteration (loop): the `for` statement

- If you want to repeatedly do something, or apply the same commands to different objects, you can use the `for` statement.

```
> for (variable in sequence) commands
```

```
for (variable in sequence) {
     command_A
     command_B
     ……
}
```

Iteration (loop): the `for` statement



Apply a calculation to every element of a vector

```
> x <- 1:10

> y <- rep(0, 10)

> for (i in 1:length(x)) y[i] <- 1/x[i]

> z <- 0

> for (i in 1:length(x)) {
      z <- z + x[i] * y[i]
  }
```

Calculate the factorial $f(n) = n!$

```
fac <- function (n) {
    if (n == 0) return(1)

    f <- 1
    for (i in 1:n) {
        f <- f * i
    }
    return(f)
}
```

Practice: matrix multiplication

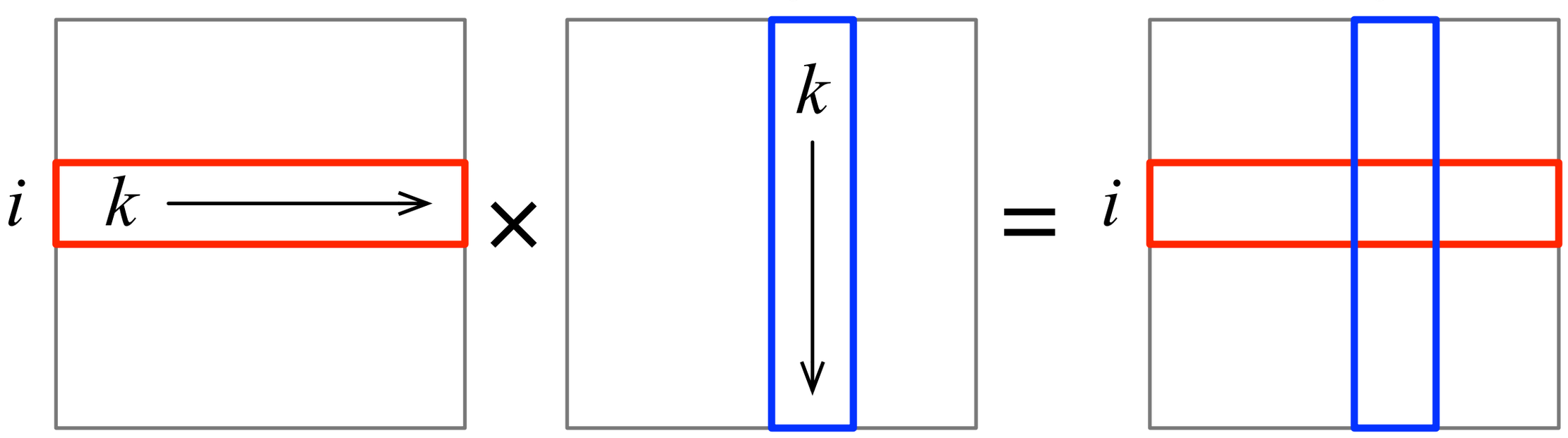
- Write a function that calculates the product of two square matrices with the same size using `for` statement (and without using `%*%` command). Create two matrices of arbitrary size and test your function.

- For example:

```
matmul <- function (a, b) {
      ……
      ……
      return(…)
}
```

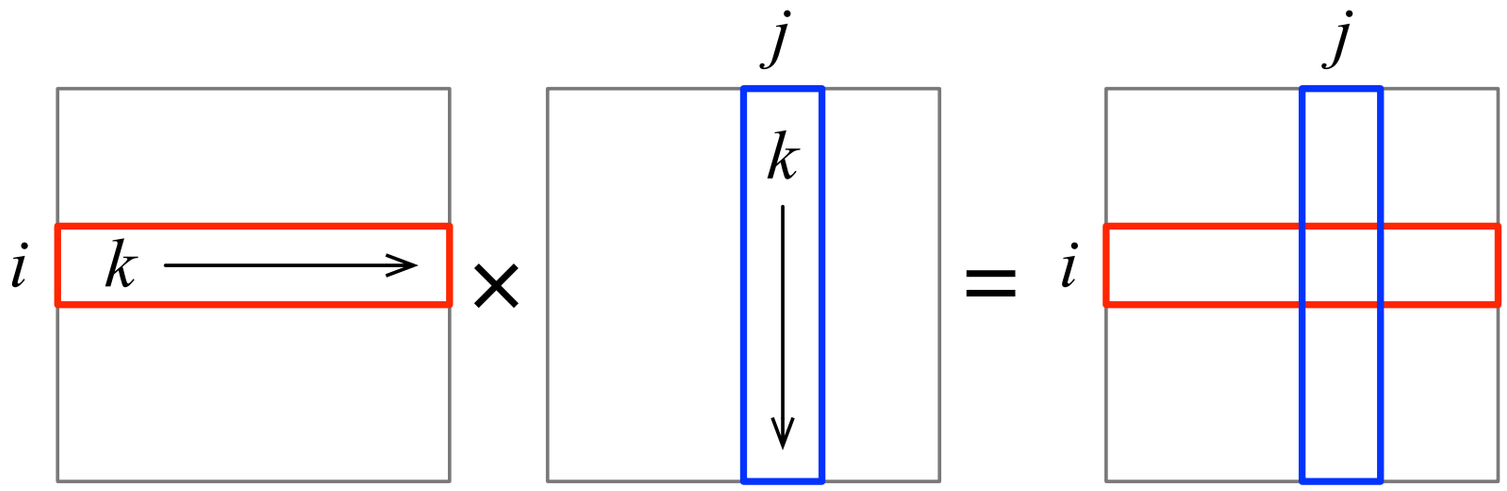
Practice: matrix multiplication

$$A \times B = C \quad \Leftrightarrow \quad C_{ij} = \sum_k A_{ik} \times B_{kj}$$



Practice: matrix multiplication

```
matmul <- function (x, y) {
   n <- nrow(x)
   z <- matrix(rep(0, n^2), n, n)
   for (i in 1:n) {
      for (j in 1:n) {
         for (k in 1:n) {
            z[i,j] <- z[i,j] + x[i,k] * y[k,j]
         }
      }
   }
   return(z)
}
```



Recursion

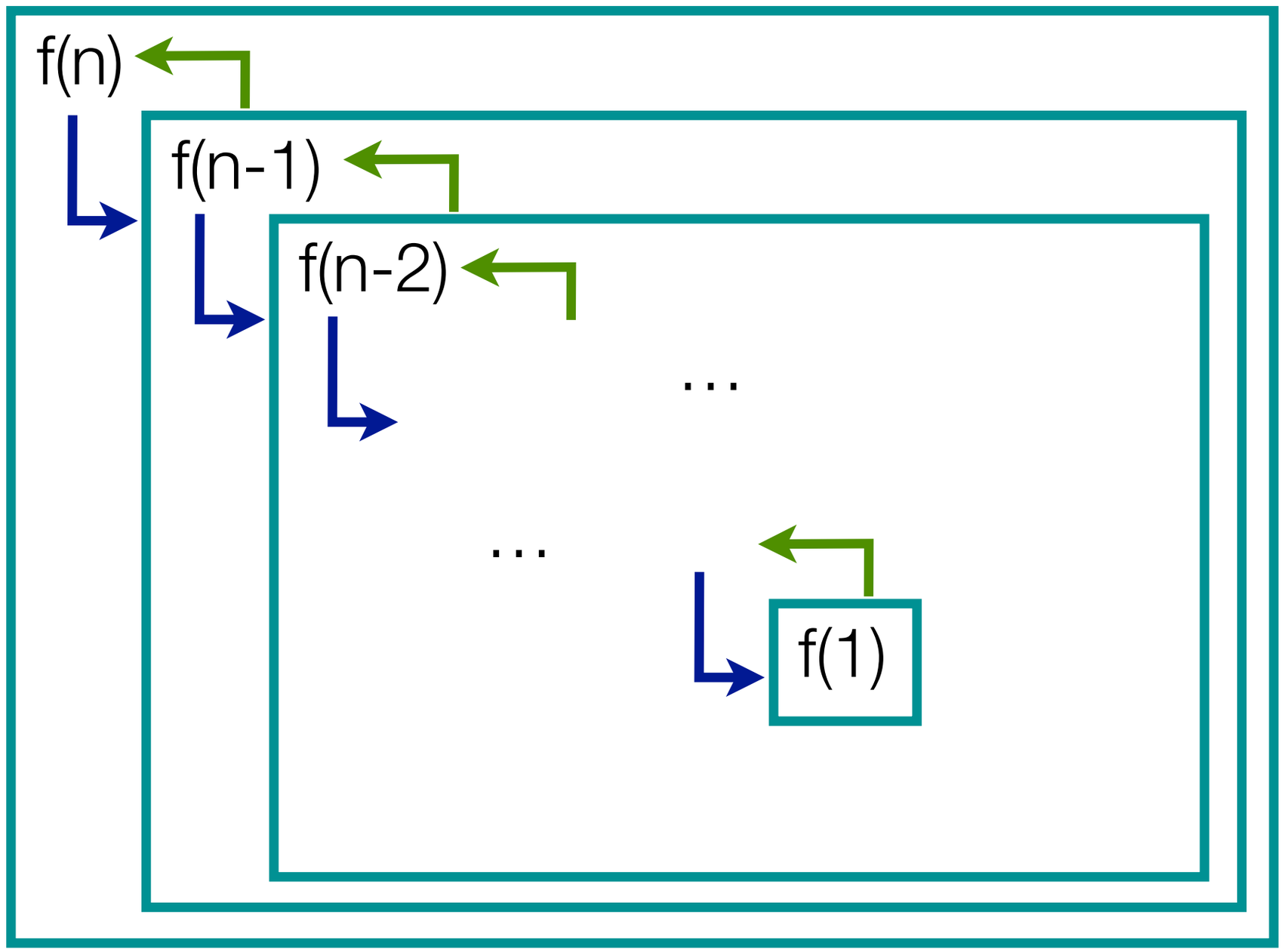
- A recurrence relation of the factorial function

$$f(n) = n! \quad \Leftrightarrow \quad f(n) = n \times f(n-1) \text{ with } f(1) = 1$$

- Calculate factorial using recursion:

```
facrec <- function (n) {
    if (n <= 1) {
        return(1)
    } else {
        return(n * facrec(n-1))
    }
}
```

Recursion



Practice: find the maximum

- Define a function `maxfun` which returns the maximum number of a set of given real numbers.

- For example, if we let

  ```
  > x <- c(3, -2.5, 0, 34, pi)
  ```

  then it is expected `maxfun(x) = 34`

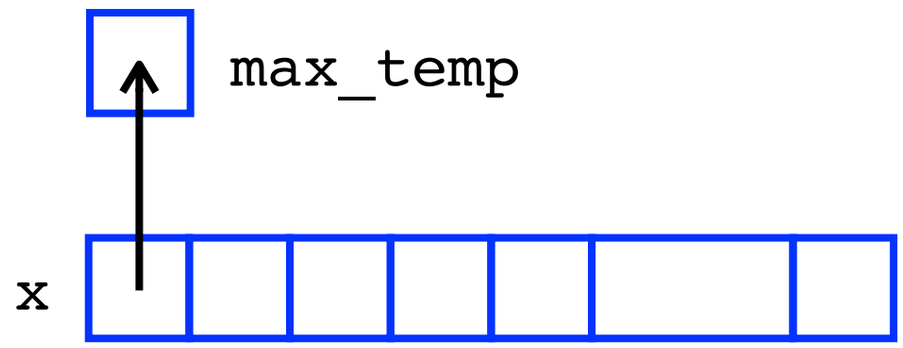
- Note: your program must take care of the case that `x` contains only one number.

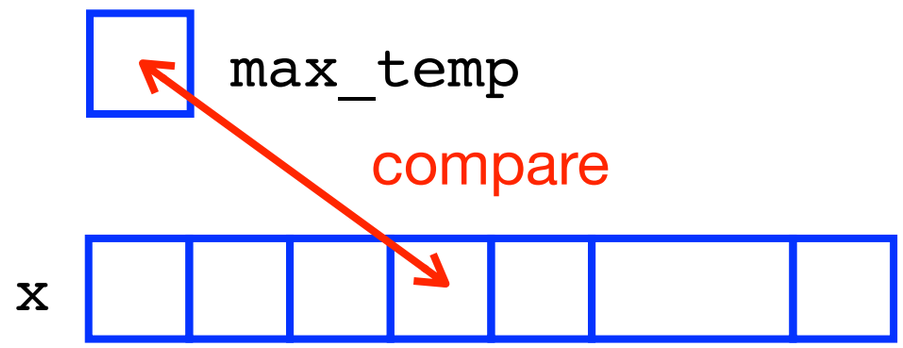
Practice: find the maximum

```
maxfun <- function (x) {
  n <- length(x)

  max_temp <- x[1]

  for (i in 1:n) {
    if (x[i] > max_temp) {
      max_temp <- x[i]
    }
  }
  return(max_temp)
}
```

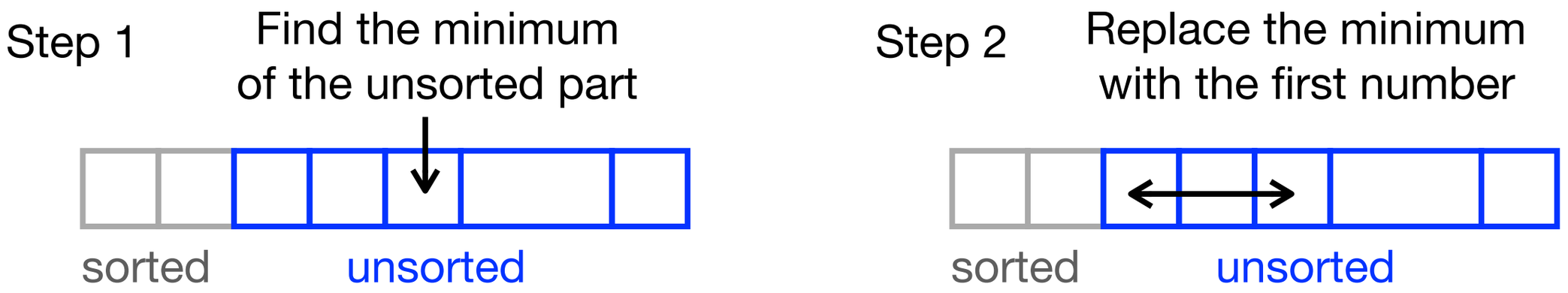




Practice: sort a list of numbers

- In many situations, we want to *sort* a list of numbers in ascending (or descending) order, i.e., from the smallest to the largest (from the largest to the smallest).

- Selection sort (ascending order):



Practice: sort a list of numbers

- Write a function `selesort` which implement the selection sort in ascending order. Apply it to previously defined `x`

- Hint:

1. You need to find both the *value* and the *location* of the minimum of the unsorted part.

2. The output (the `return` part) of a function can be a vector.

Practice: sort a list of numbers

```
minfun <- function (x) {
   n <- length(x)
   min_temp <- x[1]
   loc_temp <- 1
   for (i in 1:n) {
      if (x[i] < min_temp) {
         min_temp <- x[i]
         loc_temp <- i
      }
   }
   return(c(min_temp, loc_temp))
}
```

Practice: sort a list of numbers

```
selesort <- function (x) {
  n <- length(x)
  for (i in 1:n) {
    unsorted <- x[i:n]
    min_us <- minfun(unsorted)

    i_temp <- x[i]
    x[i] <- min_us[1]
    x[i + min_us[2] - 1] <- i_temp
  }
  return(x)
}
```

Specify the unsorted part

Find the minimum

Swap

Assignment 1

References

1. Kleiber, C. and Zeileis, A., *Applied Econometrics with R*, Springer, 2008.

2. Fox, C. and Weinberg, S. *An R Companion to Applied Regression*, 2nd Edition, SAGE, 2011.

3. Venables, W. N., Smith, D. M., and the R Core Team, *An Introduction to R*, Version 3.3.0, 2016. https://cran.r-project.org/manuals.html