R 프로그래밍 #4

2019. 03. 26

한국생명공학연구원
김하성

Summary so far..

- **R objects**: vector, factor, matrix, data.frame, list
- **R object types**: numeric, character, logical
- **Indexing** of vector, matrix: M[1], M[1,1]
- Indexing of data.frame: M[1,2], M$A[1], M[,"A"][1], M[,1][1]
- Indexing of list: M[[1]][1], M$A[1]
- **Make/use a function**
  - Parameter vs. Argument
  - Local vs. Global variables

Function

- Define a function

```
my_sine <- function(x){
        y <- sin(x)
        return(y)
}
```

- Load once (Ctrl + Enter)
- Use

```
> my_sine(pi)
```

- This returns the sine of pi
  - one parameter: x
  - one argument: pi

Function

Quiz 4-1) Parameters vs Arguments Global vs Local

```
a <- 10
myfunc <- function(b){
  a <- b/2
  return(a)
}
myfunc(a)
cat(a)
```

What are the parameter and argument?
What are the values of global and local variable a?

Function

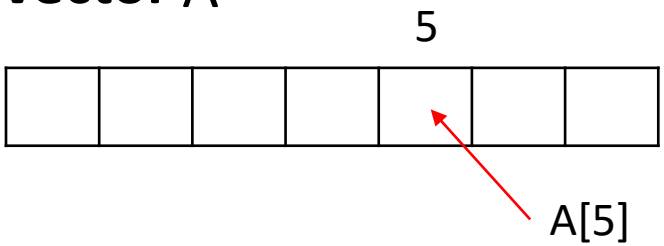
Quiz 4-2) Parameters vs Arguments Global vs Local

```
a <- 10
myfunc <- function(a){
  a <- a/10
  return(a)
}
myfunc(a)
cat(a)
```

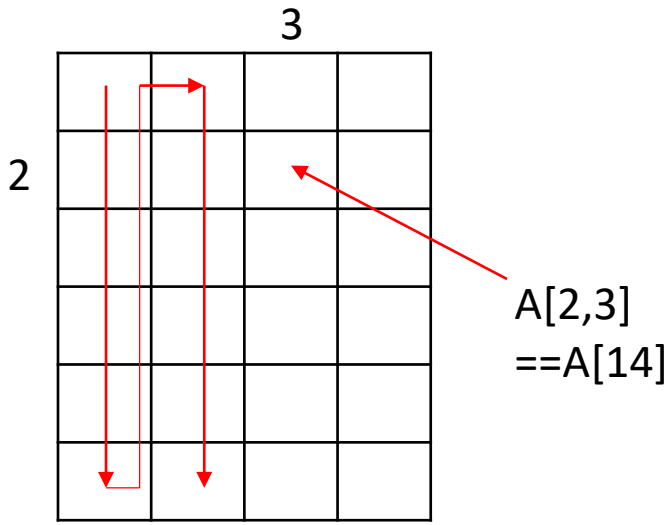
What are the parameter and argument?
What are the values of global and local variable a?

Object – vector, factor, matrix, data.frame, list

vector A

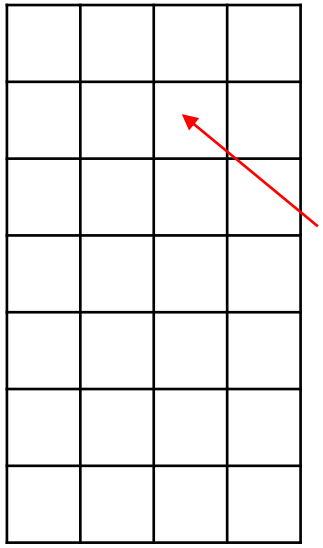


matrix (=2D array) A



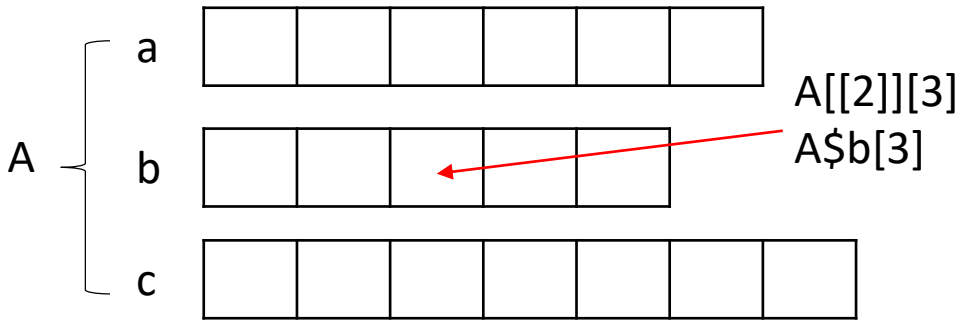
* All the elements are the same type
* matrix function

data.frame A



* A set of vector variables
* All variables should have the same length
* No necessary to be the same type

list * Same as data.frame but different length of variables



Exercise 4-1) matrix index

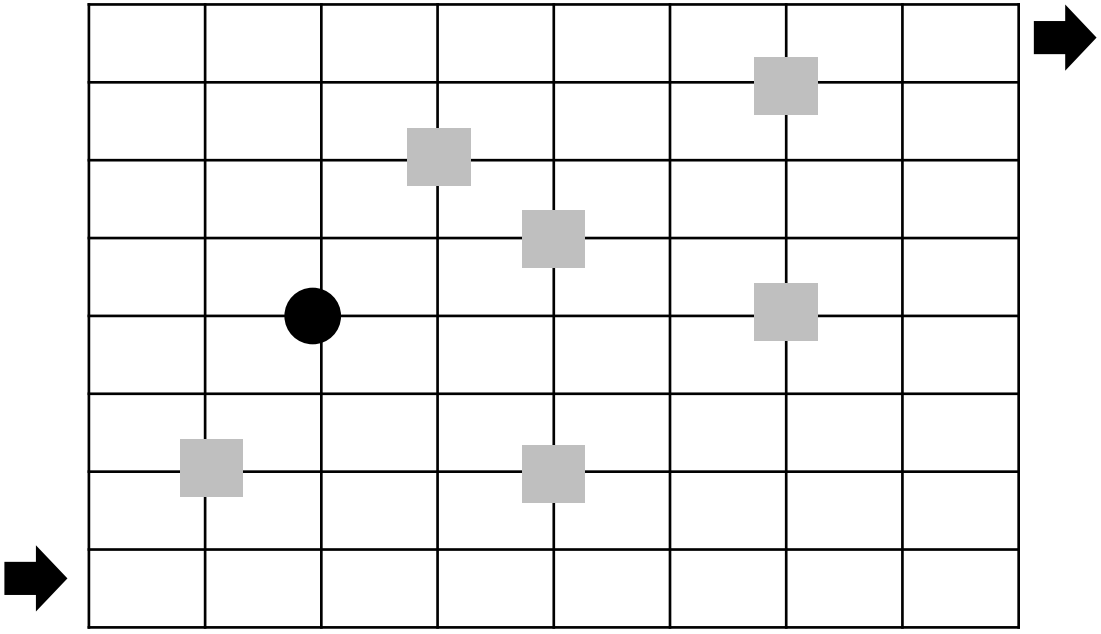
- Build a 3 x 3 matrix M with 0
- Fill all the edge values to 1

Top-down design in programming

1. Divide a big problem into smaller problems
2. keep dividing until the small problem can be solved easily
3. Solve the small problems
4. Merge the solutions to solve the big problem

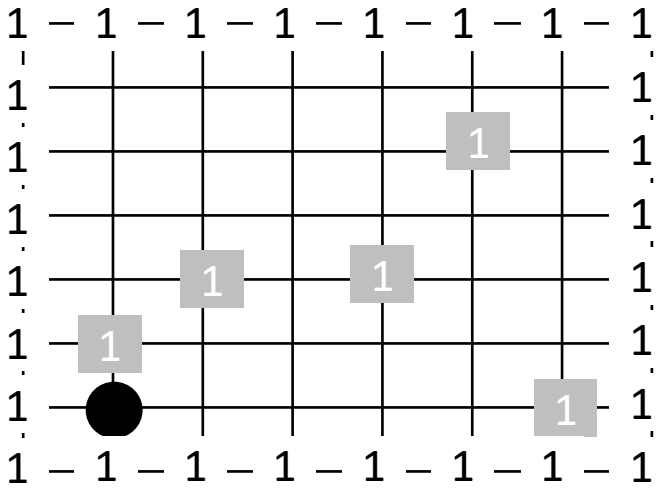
Create a robot finding an exit in a maze

- Big problem → Smaller ones
  - Map: grid, start, exit, blocks
  - Robot: position, move, rules
  - Visualization

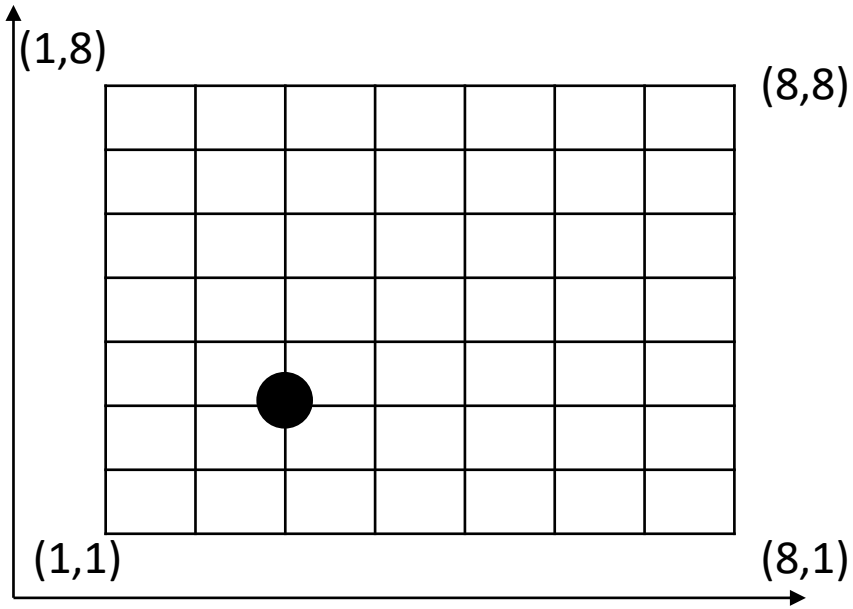


Make it into more smaller problems

- Map: 8 x 8 matrix
  - All zero except blocks and walls which are coded by 1
  - Start/exit positions: (2,2), (7,7)
- Robot: black dot
  - Remember current position
  - Move one step at a time,
  - Choose one of available directions which are 0 in the map
- Visualization:
  - Draw the world
  - Draw robot at every step

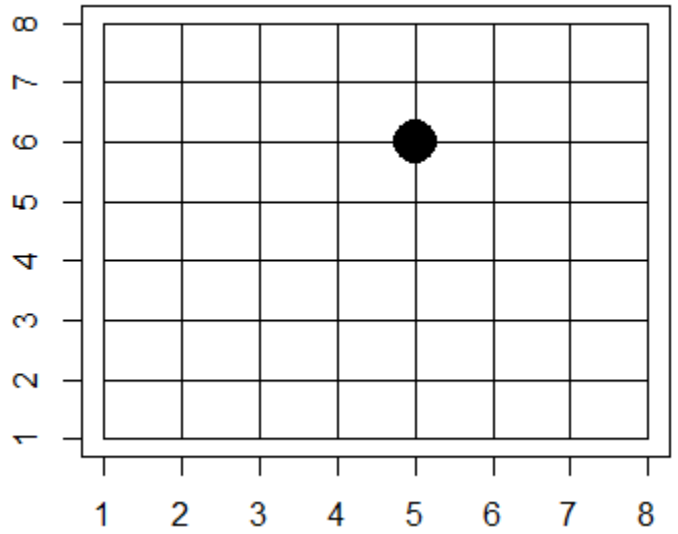


Function for drawing the whole world



Exercise 4-2) Code for drawing the world

```
plot(0, type="n", ylim=c(1,8), xlim=c(1,8))

lines(x=c(1,1), y=c(1,8))
lines(x=c(2,2), y=c(1,8))
…

lines(x=c(1,8), y=c(1,1))
lines(x=c(1,8), y=c(2,2))
…

points(3,3, pch=16, cex=5) # robot
?pch
```

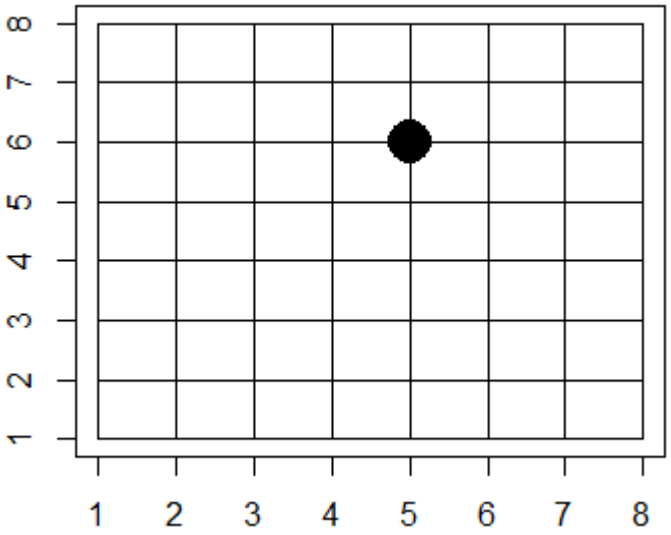


Use **for** loop

Exercise 4-3) function for drawing the world

- name: draw_world
- Input parameters: cur_x, cur_y
- Draw the robot at position (cur_x, cur_y)
  - points(x=cur_x, y=cur_y, pch=16, cex=3)
- return NULL

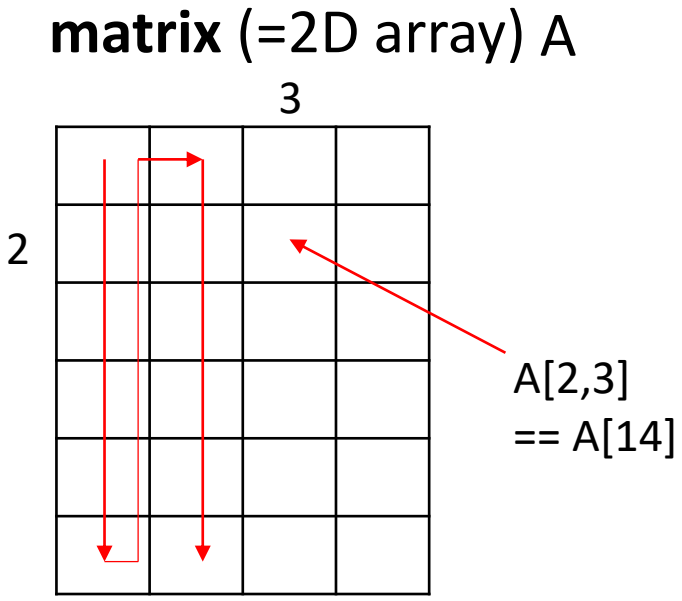
```
cur_x <- 1
cur_y <- 1

draw_world(cur_x, cur_y)
draw_world(cur_x+1, cur_y)
draw_world(cur_x+1, cur_y+1)
```



Generate maze map

- Construct a 8x8 matrix ‘maze’ for the map
  - 1 for wall and everything else 0
  - For blocks
    - Make block_index variable by randomly choose 5 numbers in 1:64
    - maze[block_index] <- 1

```
maze <- matrix(0, nrow=8, ncol=8)
maze[1,] <- 1
maze[8,] <- 1
maze[,1] <- 1
maze[,8] <- 1
## blocks
maze[sample(1:64,5)] <- 1
maze
```



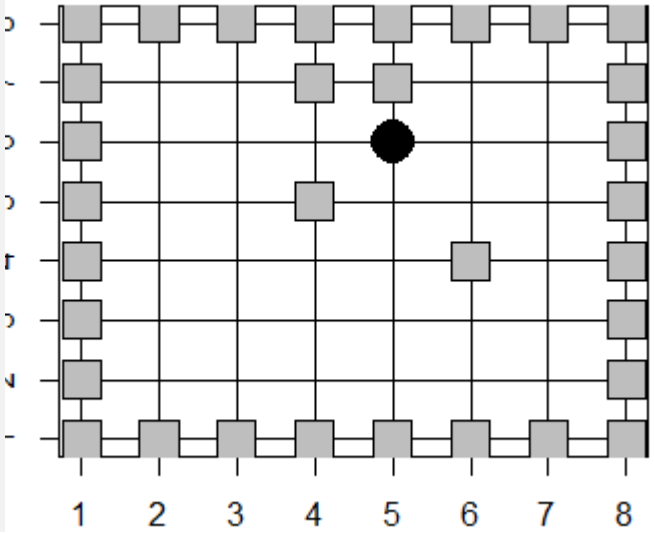
Exercise 4-4) function for generating maze

- name: generate_maze
- Input parameter: n (matrix ncol or nrow)
- Build a nxn matrix ‘maze’
- Fill the elements with 0 or 1 for blocks and walls
- return maze

```
> mymaze <- generate_maze(8)
> mymaze
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    1    1    1    1    1    1    1
[2,]    1    0    0    0    0    1    0    1
[3,]    1    0    0    0    1    0    0    1
[4,]    1    0    0    0    0    0    0    1
[5,]    1    1    0    0    0    0    1    1
[6,]    1    0    0    0    0    0    0    1
[7,]    1    0    0    0    0    0    0    1
[8,]    1    1    1    1    1    1    1    1
```

Draw blocks and walls

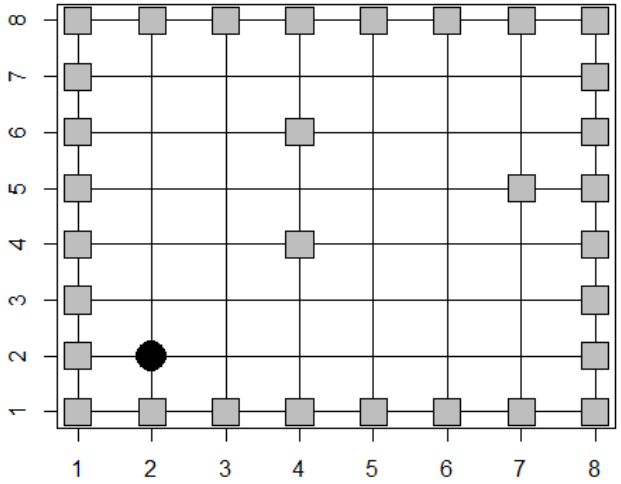
```
mymaze <- generate_maze(8)
mymaze

for(i in 1:nrow(mymaze)){
  for(j in 1:nrow(mymaze)){
    if(mymaze[i,j]==1){
      points(i,j, pch=22, cex=3, bg="gray")
    }
  }
}
```



Exercise 4-5) Update draw_world

- Add input parameter: maze
- Draw blocks and walls with gray squares

```
cur_x <- 2
cur_y <- 2
mymaze <- generate_maze(8)
draw_world(mymaze, cur_x, cur_y)
```



Move one step

```
cur_x <- 2
cur_y <- 2
draw_world(mymaze, cur_x, cur_y)
one_step_direction <- sample(c("E","W", "S", "N"), 1)
if(one_step_direction=="E"){
  next_x <- cur_x + 1
  next_y <- cur_y
}else if(one_step_direction=="W"){
  next_x <- cur_x - 1
  next_y <- cur_y
}else if(one_step_direction=="S"){
  next_x <- cur_x
  next_y <- cur_y - 1
}else if(one_step_direction=="N"){
  next_x <- cur_x
  next_y <- cur_y + 1
}
draw_world(mymaze, next_x, next_y)
```

Sleep delay

```
cur_x <- 2
cur_y <- 2
draw_world(mymaze, cur_x, cur_y)
Sys.sleep(1)
one_step_direction <- sample(c("E","W", "S", "N"), 1)
if(one_step_direction=="E"){
  next_x <- cur_x + 1
  next_y <- cur_y
}else if(one_step_direction=="W"){
  next_x <- cur_x - 1
  next_y <- cur_y
}else if(one_step_direction=="S"){
  next_x <- cur_x
  next_y <- cur_y - 1
}else if(one_step_direction=="N"){
  next_x <- cur_x
  next_y <- cur_y + 1
}
draw_world(mymaze, next_x, next_y)
```

If the next step goes to the block or wall

```
cur_x <- 2
cur_y <- 2
draw_world(mymaze, cur_x, cur_y)
Sys.sleep(1)
one_step_direction <- sample(c("E","W", "S", "N"), 1)
if(one_step_direction=="E"){
  next_x <- cur_x + 1
  next_y <- cur_y
}else if(one_step_direction=="W"){
  next_x <- cur_x - 1
  next_y <- cur_y
}else if(one_step_direction=="S"){
  next_x <- cur_x
  next_y <- cur_y - 1
}else if(one_step_direction=="N"){
  next_x <- cur_x
  next_y <- cur_y + 1
}
if(mymaze[next_x, next_y]==0){
  draw_world(mymaze, next_x, next_y)
}
```

Move 50 steps

```
mymaze <- generate_maze(8)
cur_x <- 2
cur_y <- 2
for(i in 1:50){
  draw_world(mymaze, cur_x, cur_y)
  Sys.sleep(1)
  one_step_direction <- sample(c("E","W", "S", "N"), 1)
  cat(i, "/", one_step_direction, "\n");flush.console()
  if(one_step_direction=="E"){
    next_x <- cur_x + 1
    next_y <- cur_y
  }else if(one_step_direction=="W"){
    next_x <- cur_x - 1
    next_y <- cur_y
  }else if(one_step_direction=="S"){
    next_x <- cur_x
    next_y <- cur_y - 1
  }else if(one_step_direction=="N"){
    next_x <- cur_x
    next_y <- cur_y + 1
  }
  if(mymaze[next_x, next_y]==0){
    cur_x <- next_x
    cur_y <- next_y
  }
}
```

Exit

```
    if(cur_x==7 && cur_y==7){
      stop("Exit success!")
    }
```

More improvement etc..

```
delete_robot <- function(cur_x, cur_y){
  points(x=cur_x, y=cur_y, pch=16, cex=3, col="white")
}

draw_robot <- function(cur_x, cur_y){
  points(x=cur_x, y=cur_y, pch=16, cex=3, col="black")
}
```

Good programming habits

1. Use comments as many as possible
2. Use meaningful variable/function names
3. Starts with simple case
4. One function for one behavior
5. Code independency

Next

- Excel data read/write
- Data manipulation
  - dplyr
- Visualization