

>>>>>>>>>> 제어 구조의 설계 원리를 중심으로 배우는 >>>>>>>>>>

# 프로그래밍의 정석

# 파이썬

도경구 지음



CHAPTER 8

프로젝트 기반 학습 I

퍼즐게임 스도쿠

# 스도쿠

## Sudoku 數獨

게임 시작 전

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

게임 종료 후

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

프로그래밍의 정석  
파이썬

# 8

## 프로젝트 기반 학습 I 퍼즐게임 스도쿠

8.1 중첩 루프 · 8.2 프로그래밍 프로젝트 : 스도쿠

### CHAPTER 8

## 프로젝트 기반 학습 I 퍼즐게임 스도쿠



8.1 중첩 루프

8.2 프로그래밍 프로젝트 : 스도쿠

# 중첩 루프

## Nested Loop

# 중첩 루프

## Nested Loop

code : 8-1.py

```
1 couples = []  
2 for c in ["a", "b"]:  
3     for n in range(3):  
4         couple = (c,n)  
5         couples.append(couple)  
6 print(couples)
```

```
couples = []  
=>
```

# 중첩 루프

## Nested Loop

code : 8-1.py

```
1 couples = []  
2 for c in ["a", "b"]:  
3     for n in range(3):  
4         couple = (c,n)  
5         couples.append(couple)  
6 print(couples)
```

```
couples = []  
=> [("a", 0)]  
=>
```

# 중첩 루프

## Nested Loop

code : 8-1.py

```
1 couples = []  
2 for c in ["a", "b"]:  
3     for n in range(3):  
4         couple = (c,n)  
5         couples.append(couple)  
6 print(couples)
```

```
couples = []  
=> [("a", 0)]  
=> [("a", 0), ("a", 1)]  
=>
```

# 중첩 루프

## Nested Loop

code : 8-1.py

```
1 couples = []
2 for c in ["a", "b"]:
3     for n in range(3):
4         couple = (c,n)
5         couples.append(couple)
6 print(couples)
```

```
couples = []
=> [("a", 0)]
=> [("a", 0), ("a", 1)]
=> [("a", 0), ("a", 1), ("a", 2)]
=>
```



# 중첩 루프

## Nested Loop

code : 8-1.py

```
1 couples = []
2 for c in ["a", "b"]:
3     for n in range(3):
4         couple = (c,n)
5         couples.append(couple)
6 print(couples)
```

```
couples = []
=> [("a", 0)]
=> [("a", 0), ("a", 1)]
=> [("a", 0), ("a", 1), ("a", 2)]
=> [("a", 0), ("a", 1), ("a", 2), ("b", 0)]
=>
```

# 중첩 루프

## Nested Loop

code : 8-1.py

```
1 couples = []
2 for c in ["a", "b"]:
3     for n in range(3):
4         couple = (c,n)
5         couples.append(couple)
6 print(couples)
```

```
couples = []
=> [("a", 0)]
=> [("a", 0), ("a", 1)]
=> [("a", 0), ("a", 1), ("a", 2)]
=> [("a", 0), ("a", 1), ("a", 2), ("b", 0)]
=> [("a", 0), ("a", 1), ("a", 2), ("b", 0), ("b", 1)]
=>
```

# 중첩 루프

## Nested Loop

code : 8-1.py

```
1 couples = []
2 for c in ["a", "b"]:
3     for n in range(3):
4         couple = (c,n)
5         couples.append(couple)
6 print(couples)
```

```
couples = []
=> [("a", 0)]
=> [("a", 0), ("a", 1)]
=> [("a", 0), ("a", 1), ("a", 2)]
=> [("a", 0), ("a", 1), ("a", 2), ("b", 0)]
=> [("a", 0), ("a", 1), ("a", 2), ("b", 0), ("b", 1)]
=> [("a", 0), ("a", 1), ("a", 2), ("b", 0), ("b", 1), ("b", 2)]
```

>>>>>>>>>> 제어 구조의 설계 원리를 중심으로 배우는 >>>>>>>>>>

# 프로그래밍의 정석 파이썬

도경구 지음

pp.370~375



## 8.1.1 아스키 아트



### 실습 8.1 가로 변형

`digital_art_horizontal(7)`

```
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 4 5 6 7
```

### 8.1.1 아스키 아트



## 실습 8.1 가로 변형

[illegible]

```

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5 6 7

```

1	2	3	4	5	6	7
1	2	3	4	5	6	
1	2	3	4	5		
1	2	3	4			
1	2	3				
1	2					
1						

>>>>>>>>>> 제어 구조의 설계 원리를 중심으로 배우는 >>>>>>>>>>

# 프로그래밍의 정석 파이썬

도경구 지음

pp.370~375



## 8.1.1 아스키 아트



### 실습 8.2 세로 변형

`digital_art_vertical(7)`

```
1 1 1 1 1 1 1
2 2 2 2 2 2 2
3 3 3 3 3 3 3
4 4 4 4 4 4 4
5 5 5 5 5 5 5
6 6 6 6 6 6 6
7 7 7 7 7 7 7
```

# 프로그래밍의 정석 파이썬

도경구 지음

pp.370~375



## 8.1.1 아스키 아트



### 실습 8.2 세로 변형

`digital_art_vertical(7)`

```
1 1 1 1 1 1 1
2 2 2 2 2 2 2
3 3 3 3 3 3 3
4 4 4 4 4 4 4
5 5 5 5 5 5 5
6 6 6 6 6 6 6
7 7 7 7 7 7 7
```

`digital_art_vertical_right_down(7)`

```

                                     1
                                2 2
                           3 3 3
                      4 4 4 4
                5 5 5 5 5
           6 6 6 6 6 6
       7 7 7 7 7 7 7
```

# 프로그래밍의 정석 파이썬

도경구 지음

pp.370~375



## 8.1.1 아스키 아트



### 실습 8.3 세로 방향 바꾸기

```
digital_art_horizontal_alternate(7)
```

```
1 2 3 4 5 6 7
7 6 5 4 3 2 1
1 2 3 4 5 6 7
7 6 5 4 3 2 1
1 2 3 4 5 6 7
7 6 5 4 3 2 1
1 2 3 4 5 6 7
```





## 실습 8.3 세로 방향 바꾸기

digital\_art\_vertical\_alternate(7)

1	2	3	4	5	6	7
7	6	5	4	3	2	1
1	2	3	4	5	6	7
7	6	5	4	3	2	1
1	2	3	4	5	6	7
7	6	5	4	3	2	1
1	2	3	4	5	6	7

1	7	1	7	1	7	1
2	6	2	6	2	6	2
3	5	3	5	3	5	3
4	4	4	4	4	4	4
5	3	5	3	5	3	5
6	2	6	2	6	2	6
7	1	7	1	7	1	7

# 프로그래밍의 정석

# 파이썬

도경구 지음



pp.375~377

### 8.1.2 버블정렬



## 실습 8.4 버블정렬 구현

# 버블 정렬

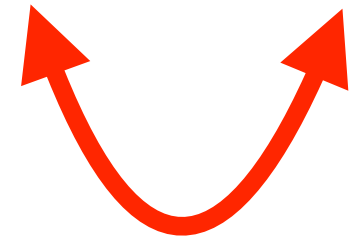
## Bubble Sort

32	23	18	7	11	99	55
----	----	----	---	----	----	----

# 버블 정렬

## Bubble Sort

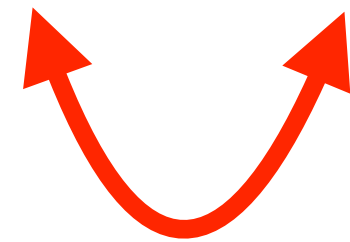
32	23	18	7	11	99	55
----	----	----	---	----	----	----



# 버블 정렬

## Bubble Sort

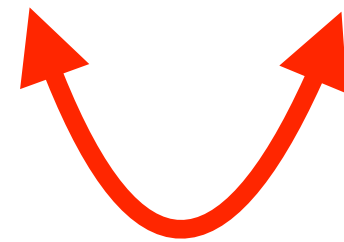
23	32	18	7	11	99	55
----	----	----	---	----	----	----



# 버블 정렬

## Bubble Sort

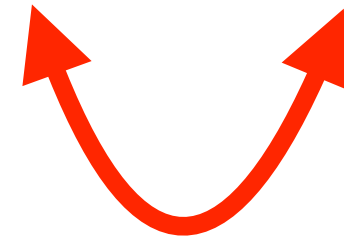
23	18	32	7	11	99	55
----	----	----	---	----	----	----



# 버블 정렬

## Bubble Sort

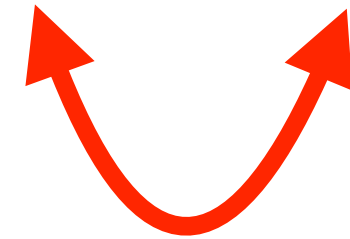
23	18	7	32	11	99	55
----	----	---	----	----	----	----



# 버블 정렬

## Bubble Sort

23	18	7	11	32	99	55
----	----	---	----	----	----	----

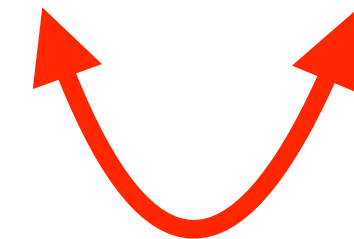




# 버블 정렬

## Bubble Sort

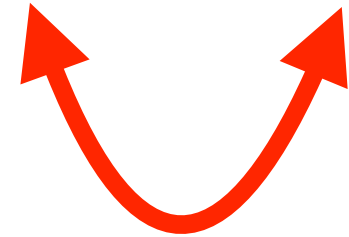
23	18	7	11	32	99	55
----	----	---	----	----	----	----



# 버블 정렬

## Bubble Sort

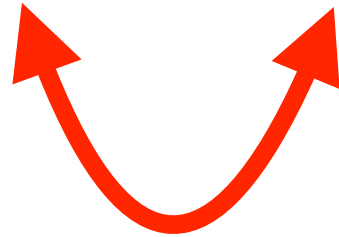
23	18	7	11	32	55	99
----	----	---	----	----	----	----



# 버블 정렬

## Bubble Sort

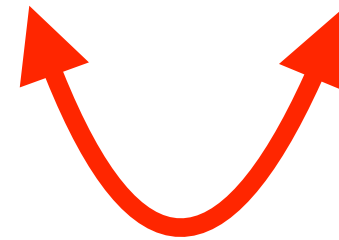
18	23	7	11	32	55	99
----	----	---	----	----	----	----



# 버블 정렬

## Bubble Sort

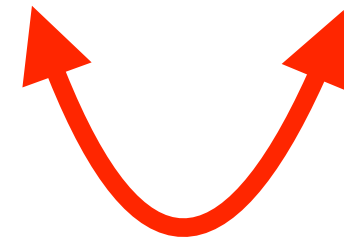
18	7	23	11	32	55	99
----	---	----	----	----	----	----



# 버블 정렬

## Bubble Sort

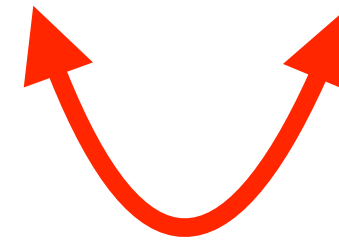
18	7	11	23	32	55	99
----	---	----	----	----	----	----



# 버블 정렬

## Bubble Sort

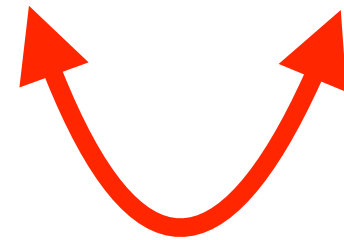
18	7	11	23	32	55	99
----	---	----	----	----	----	----



# 버블 정렬

## Bubble Sort

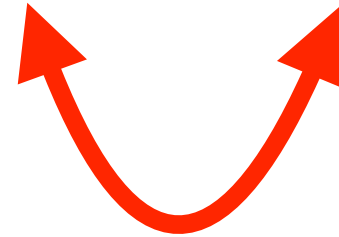
18	7	11	23	32	55	99
----	---	----	----	----	----	----



# 버블 정렬

## Bubble Sort

7	18	11	23	32	55	99
---	----	----	----	----	----	----

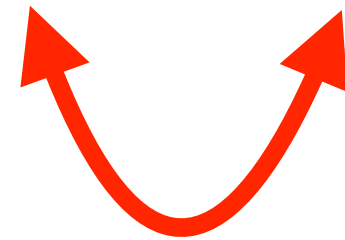




# 버블 정렬

## Bubble Sort

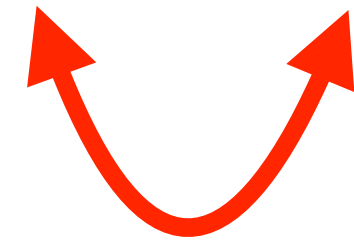
7	11	18	23	32	55	99
---	----	----	----	----	----	----



# 버블 정렬

## Bubble Sort

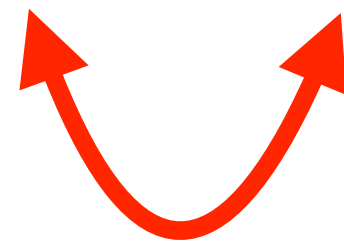
7	11	18	23	32	55	99
---	----	----	----	----	----	----



# 버블 정렬

## Bubble Sort

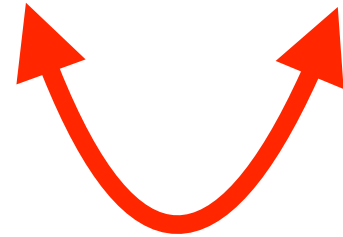
7	11	18	23	32	55	99
---	----	----	----	----	----	----



# 버블 정렬

## Bubble Sort

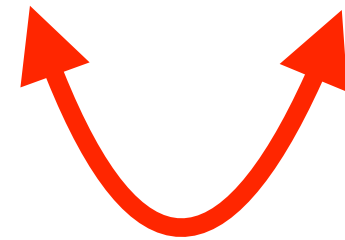
7	11	18	23	32	55	99
---	----	----	----	----	----	----



# 버블 정렬

## Bubble Sort

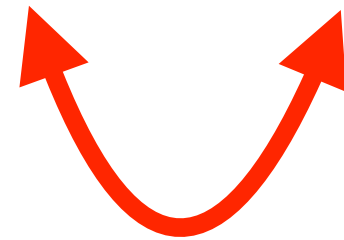
7	11	18	23	32	55	99
---	----	----	----	----	----	----



# 버블 정렬

## Bubble Sort

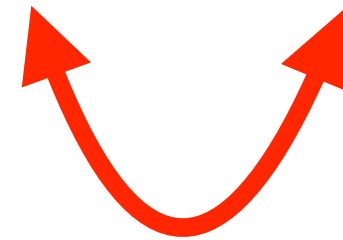
7	11	18	23	32	55	99
---	----	----	----	----	----	----



# 버블 정렬

## Bubble Sort

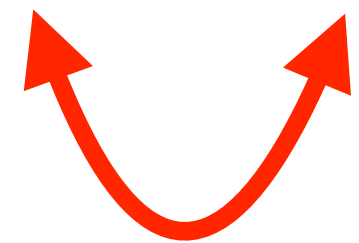
7	11	18	23	32	55	99
---	----	----	----	----	----	----



# 버블 정렬

## Bubble Sort

7	11	18	23	32	55	99
---	----	----	----	----	----	----





# 버블 정렬

## Bubble Sort

7	11	18	23	32	55	99
---	----	----	----	----	----	----

>>>>>>>>>> 제어 구조의 설계 원리를 중심으로 배우는 >>>>>>>>>>

# 프로그래밍의 정석 파이썬

도경구 지음



pp.377~379

## 8.1.3 기수정렬



실습 8.5 기수정렬 구현

# 기수 정렬

## Radix Sort

["0508", "0515", "1225", "0915", "1111", "0101", "0318", "0301", "0815"]

# 기수 정렬

## Radix Sort

["0508", "0515", "1225", "0915", "1111", "0101", "0318", "0301", "0815"]



분배

[[], [], [], [], [], [], [], [], [], []]

0 1 2 3 4 5 6 7 8 9

# 기수 정렬

## Radix Sort

["0508", "0515", "1225", "0915", "1111", "0101", "0318", "0301", "0815"]



분배

[[], ["1111", "0101", "0301"], [], [], [], ["0515", "1225", "0915", "0815"], [], [], ["0508", "0318"], []]

0

1

2

3

4

5

6

7

8

9

# 기수 정렬

## Radix Sort

["0508", "0515", "1225", "0915", "1111", "0101", "0318", "0301", "0815"]



[[], ["1111", "0101", "0301"], [], [], [], ["0515", "1225", "0915", "0815"], [], [], ["0508", "0318"], []]



["1111", "0101", "0301", "0515", "1225", "0915", "0815", "0508", "0318"]

# 기수 정렬

## Radix Sort

["0508", "0515", "1225", "0915", "1111", "0101", "0318", "0301", "0815"]



[[], ["1111", "0101", "0301"], [], [], [], ["0515", "1225", "0915", "0815"], [], [], ["0508", "0318"], []]



["1111", "0101", "0301", "0515", "1225", "0915", "0815", "0508", "0318"]



[[], [], [], [], [], [], [], [], [], []]

0 1 2 3 4 5 6 7 8 9

# 기수 정렬

## Radix Sort

["0508", "0515", "1225", "0915", "1111", "0101", "0318", "0301", "0815"]



[[], ["1111", "0101", "0301"], [], [], [], ["0515", "1225", "0915", "0815"], [], [], ["0508", "0318"], []]



["1111", "0101", "0301", "0515", "1225", "0915", "0815", "0508", "0318"]



[["0101", "0301", "0508"], ["1111", "0515", "0915", "0815", "0318"], ["1225"], [], [], [], [], [], [], []]

0

1

2

3

4

5

6

7

8

9



# 기수 정렬

## Radix Sort

["0508", "0515", "1225", "0915", "1111", "0101", "0318", "0301", "0815"]



[[], ["1111", "0101", "0301"], [], [], [], ["0515", "1225", "0915", "0815"], [], [], ["0508", "0318"], []]



["1111", "0101", "0301", "0515", "1225", "0915", "0815", "0508", "0318"]



[["0101", "0301", "0508"], ["1111", "0515", "0915", "0815", "0318"], ["1225"], [], [], [], [], [], [], []]



["0101", "0301", "0508", "1111", "0515", "0915", "0815", "0318", "1225"]

# 기수 정렬

## Radix Sort

["0508", "0515", "1225", "0915", "1111", "0101", "0318", "0301", "0815"]



[[], ["1111", "0101", "0301"], [], [], [], ["0515", "1225", "0915", "0815"], [], [], ["0508", "0318"], []]



["1111", "0101", "0301", "0515", "1225", "0915", "0815", "0508", "0318"]



["0101", "0301", "0508"], ["1111", "0515", "0915", "0815", "0318"], ["1225"], [], [], [], [], [], []]



["0101", "0301", "0508", "1111", "0515", "0915", "0815", "0318", "1225"]



[[], [], [], [], [], [], [], [], [], []]

0 1 2 3 4 5 6 7 8 9

# 기수 정렬

## Radix Sort

["0508", "0515", "1225", "0915", "1111", "0101", "0318", "0301", "0815"]



[[], ["1111", "0101", "0301"], [], [], [], ["0515", "1225", "0915", "0815"], [], [], ["0508", "0318"], []]



["1111", "0101", "0301", "0515", "1225", "0915", "0815", "0508", "0318"]



["0101", "0301", "0508"], ["1111", "0515", "0915", "0815", "0318"], ["1225"], [], [], [], [], [], []]



["0101", "0301", "0508", "1111", "0515", "0915", "0815", "0318", "1225"]



[[], ["0101", "1111"], ["1225"], ["0301", "0318"], [], ["0508", "0515"], [], [], ["0815"], ["0915"]]

0

1

2

3

4

5

6

7

8

9

# 기수 정렬

## Radix Sort

["0508", "0515", "1225", "0915", "1111", "0101", "0318", "0301", "0815"]



[[], ["1111", "0101", "0301"], [], [], [], ["0515", "1225", "0915", "0815"], [], [], ["0508", "0318"], []]



["1111", "0101", "0301", "0515", "1225", "0915", "0815", "0508", "0318"]



[["0101", "0301", "0508"], ["1111", "0515", "0915", "0815", "0318"], ["1225"], [], [], [], [], [], [], []]



["0101", "0301", "0508", "1111", "0515", "0915", "0815", "0318", "1225"]



[[], ["0101", "1111"], ["1225"], ["0301", "0318"], [], ["0508", "0515"], [], [], ["0815"], ["0915"]]



["0101", "1111", "1225", "0301", "0318", "0508", "0515", "0815", "0915"]

# 기수 정렬

## Radix Sort

["0508", "0515", "1225", "0915", "1111", "0101", "0318", "0301", "0815"]



[[], ["1111", "0101", "0301"], [], [], [], ["0515", "1225", "0915", "0815"], [], [], ["0508", "0318"], []]



["1111", "0101", "0301", "0515", "1225", "0915", "0815", "0508", "0318"]



["0101", "0301", "0508"], ["1111", "0515", "0915", "0815", "0318"], ["1225"], [], [], [], [], [], []]



["0101", "0301", "0508", "1111", "0515", "0915", "0815", "0318", "1225"]



[[], ["0101", "1111"], ["1225"], ["0301", "0318"], [], ["0508", "0515"], [], [], ["0815"], ["0915"]]



["0101", "1111", "1225", "0301", "0318", "0508", "0515", "0815", "0915"]



[[], [], [], [], [], [], [], [], [], []]

0 1 2 3 4 5 6 7 8 9

# 기수 정렬

## Radix Sort

["0508", "0515", "1225", "0915", "1111", "0101", "0318", "0301", "0815"]



[[], ["1111", "0101", "0301"], [], [], [], ["0515", "1225", "0915", "0815"], [], [], ["0508", "0318"], []]



["1111", "0101", "0301", "0515", "1225", "0915", "0815", "0508", "0318"]



[["0101", "0301", "0508"], ["1111", "0515", "0915", "0815", "0318"], ["1225"], [], [], [], [], [], [], []]



["0101", "0301", "0508", "1111", "0515", "0915", "0815", "0318", "1225"]



[[], ["0101", "1111"], ["1225"], ["0301", "0318"], [], ["0508", "0515"], [], [], ["0815"], ["0915"]]



["0101", "1111", "1225", "0301", "0318", "0508", "0515", "0815", "0915"]



[["0101", "0301", "0318", "0508", "0515", "0815", "0915"], ["1111", "1225"], [], [], [], [], [], [], []]

0

1

2

3

4

5

6

7

8

9

# 기수 정렬

## Radix Sort

["0508", "0515", "1225", "0915", "1111", "0101", "0318", "0301", "0815"]



[[], ["1111", "0101", "0301"], [], [], [], ["0515", "1225", "0915", "0815"], [], [], ["0508", "0318"], []]



["1111", "0101", "0301", "0515", "1225", "0915", "0815", "0508", "0318"]



[["0101", "0301", "0508"], ["1111", "0515", "0915", "0815", "0318"], ["1225"], [], [], [], [], [], [], []]



["0101", "0301", "0508", "1111", "0515", "0915", "0815", "0318", "1225"]



[[], ["0101", "1111"], ["1225"], ["0301", "0318"], [], ["0508", "0515"], [], [], ["0815"], ["0915"]]



["0101", "1111", "1225", "0301", "0318", "0508", "0515", "0815", "0915"]



[["0101", "0301", "0318", "0508", "0515", "0815", "0915"], ["1111", "1225"], [], [], [], [], [], [], []]



["0101", "0301", "0318", "0508", "0515", "0815", "0915", "1111", "1225"]

프로그래밍의 정석  
파이썬

# 8

프로젝트 기반 학습 I  
퍼즐게임 스도쿠

8.1 중첩 루프 · 8.2 프로그래밍 프로젝트 : 스도쿠

## CHAPTER 8

# 프로젝트 기반 학습 I 퍼즐게임 스도쿠

8.1 중첩 루프

✓ 8.2 프로그래밍 프로젝트 : 스도쿠



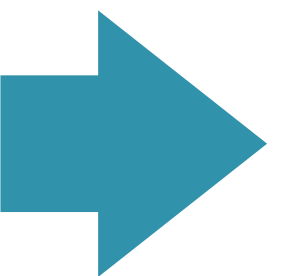
## 4 x 4 미니 스도쿠

3			2
	4	1	
	3	2	
4			1

퍼즐보드

3	1	4	2
2	4	1	3
1	3	2	4
4	2	3	1

정답보드



# 스도쿠 보드의 표현

1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1

```
board = [[1, 2, 3, 4],  
          [3, 4, 1, 2],  
          [2, 1, 4, 3],  
          [4, 3, 2, 1]]
```

# 스도쿠 보드의 표현

	0	1	2	3
0	1	2	3	4
1	3	4	1	2
2	2	1	4	3
3	4	3	2	1

```
board = [[1, 2, 3, 4],  
          [3, 4, 1, 2],  
          [2, 1, 4, 3],  
          [4, 3, 2, 1]]
```

```
board[2][3]
```

# 스도쿠 정답보드 만들기

## 정답 퍼즐보드 배치하기

row0 = [1, 2, 3, 4]

	0	1	2	3
0	1	2	3	4
1				
2				
3				

# 스도쿠 정답보드 만들기

## 정답 퍼즐보드 배치하기

row0 = [1, 2, 3, 4]

row1 = row0[2:4] + row0[0:2]

	0	1	2	3
0	1	2	3	4
1				
2				
3				

# 스도쿠 정답보드 만들기

## 정답 퍼즐보드 배치하기

row0 = [1, 2, 3, 4]

row1 = row0[2:4] + row0[0:2] == [3, 4, 1, 2]

	0	1	2	3
0	1	2	3	4
1	3	4	1	2
2				
3				

# 스도쿠 정답보드 만들기

## 정답 퍼즐보드 배치하기

row0 = [1, 2, 3, 4]

row1 = row0[2:4] + row0[0:2] == [3, 4, 1, 2]

row2 = [row0[1], row0[0], row0[3], row0[2]]

	0	1	2	3
0	1	2	3	4
1	3	4	1	2
2				
3				

# 스도쿠 정답보드 만들기

## 정답 퍼즐보드 배치하기

row0 = [1, 2, 3, 4]

row1 = row0[2:4] + row0[0:2] == [3, 4, 1, 2]

row2 = [row0[1], row0[0], row0[3], row0[2]] == [2, 1, 4, 3]

	0	1	2	3
0	1	2	3	4
1	3	4	1	2
2	2	1	4	3
3				



# 스도쿠 정답보드 만들기

## 정답 퍼즐보드 배치하기

row0 = [1, 2, 3, 4]

row1 = row0[2:4] + row0[0:2] == [3, 4, 1, 2]

row2 = [row0[1], row0[0], row0[3], row0[2]] == [2, 1, 4, 3]

row3 = row2[2:4] + row2[0:2]

	0	1	2	3
0	1	2	3	4
1	3	4	1	2
2	2	1	4	3
3				

# 스도쿠 정답보드 만들기

## 정답 퍼즐보드 배치하기

row0 = [1, 2, 3, 4]

row1 = row0[2:4] + row0[0:2] == [3, 4, 1, 2]

row2 = [row0[1], row0[0], row0[3], row0[2]] == [2, 1, 4, 3]

row3 = row2[2:4] + row2[0:2] == [4, 3, 2, 1]

	0	1	2	3
0	1	2	3	4
1	3	4	1	2
2	2	1	4	3
3	4	3	2	1

# 스도쿠 정답보드 만들기

## 정답 퍼즐보드 배치하기

row0 = [1, 2, 3, 4]

row1 = row0[2:4] + row0[0:2] == [3, 4, 1, 2]

row2 = [row0[1], row0[0], row0[3], row0[2]] == [2, 1, 4, 3]

row3 = row2[2:4] + row2[0:2] == [4, 3, 2, 1]

board = [row0, row1, row2, row3]

	0	1	2	3
0	1	2	3	4
1	3	4	1	2
2	2	1	4	3
3	4	3	2	1

# 스도쿠 정답보드 만들기

## 정답 퍼즐보드 배치하기

row0 = [1, 2, 3, 4]

row1 = row0[2:4] + row0[0:2] == [3, 4, 1, 2]

row2 = [row0[1], row0[0], row0[3], row0[2]] == [2, 1, 4, 3]

row3 = row2[2:4] + row2[0:2] == [4, 3, 2, 1]

board = [row0, row1, row2, row3] == [[1, 2, 3, 4],

[3, 4, 1, 2],

[2, 1, 4, 3],

[4, 3, 2, 1]]

	0	1	2	3
0	1	2	3	4
1	3	4	1	2
2	2	1	4	3
3	4	3	2	1

# 스도쿠 정답보드 만들기

[1, 2, 3, 4] 의 서로 다른 순서 가지 수 =  $4! = 24$

[1, 2, 3, 4]	[2, 1, 3, 4]	[3, 1, 2, 4]	[4, 1, 2, 3]
[1, 2, 4, 3]	[2, 1, 4, 3]	[3, 1, 4, 2]	[4, 1, 3, 2]
[1, 3, 2, 4]	[2, 3, 1, 4]	[3, 2, 1, 4]	[4, 2, 1, 3]
[1, 3, 4, 2]	[2, 3, 4, 1]	[3, 2, 4, 1]	[4, 2, 3, 1]
[1, 4, 2, 3]	[2, 4, 1, 3]	[3, 4, 1, 2]	[4, 3, 1, 2]
[1, 4, 3, 2]	[2, 4, 3, 1]	[3, 4, 2, 1]	[4, 3, 2, 1]

# 스도쿠 정답보드 만들기

단계 1 : 총 24가지 정답보드 중에서 하나 무작위 선택

$$4! = 24$$

```
import random
```

```
row0 = [1, 2, 3, 4]  
random.shuffle(row0)
```

# 스도쿠 정답보드 만들기 : 단계 1 구현

code : 8-7.py

```
1 def initialize_board_4x4():  
2     row0 = [1,2,3,4]  
3     random.shuffle(row0)  
4     row1 = row0[2:4] + row0[0:2]  
5     row2 = [row0[1], row0[0], row0[3], row0[2]]  
6     row3 = row2[2:4] + row2[0:2]  
7     return [row0, row1, row2, row3]
```

# 스도쿠 정답보드 만들기

단계 2 : 추가로 16가지 정답보드 중에서 하나 무작위 선택

	0	1	2	3
0	$n_1$	$n_2$	$n_3$	$n_4$
1	$n_3$	$n_4$	$n_1$	$n_2$
2	$n_2$	$n_1$	$n_4$	$n_3$
3	$n_4$	$n_3$	$n_2$	$n_1$

가로줄 바꾸기

세로줄 바꾸기

$$\text{줄바꾸기 조합} = 2 \times 2 \times 2 \times 2 = 16$$

$$\text{서로 다른 스도쿠 정답보드의 총 가지 수} = 24 \times 16 = 384$$




# 스도쿠 정답보드 만들기 : 단계 2 (가로줄 바꾸기) 구현

code : 8-8.py

```
1  def shuffle_ribbons(board) :  
2      top = board[:2]  
3      bottom = board[2:]  
4      random.shuffle(top)  
5      random.shuffle(bottom)  
6      return top + bottom
```

## 스도쿠 정답보드 만들기 : 단계 2 (세로줄 바꾸기) 구현

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16



1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

가로 세로 바꾸기  
transpose

>>>>>>>>>> 제어 구조의 설계 원리를 중심으로 배우는 >>>>>>>>>>

# 프로그래밍의 정석 파이썬

도경구 지음



pp.387~388



## 실습 8.6 가로세로 뒤집기

# 가로세로 뒤집기 transpose 알고리즘

1. 가로세로 뒤집어 저장할 보드를 다음과 같이 초기화한다.

```
transposed = [[], [], [], []]
```

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

↓ transpose


# 가로세로 뒤집기 transpose 알고리즘

1. 가로세로 뒤집어 저장할 보드를 다음과 같이 초기화한다.

```
transposed = [[], [], [], []]
```

2. 가로로 읽어서 세로로 다음과 같이 차례로 붙여나간다.

```
[[], [], [], []]
```

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

↓ transpose


# 가로세로 뒤집기 transpose 알고리즘

1. 가로세로 뒤집어 저장할 보드를 다음과 같이 초기화한다.

```
transposed = [[[], [], [], []]]
```

2. 가로로 읽어서 세로로 다음과 같이 차례로 붙여나간다.

```
[[[], [], [], []]]
```

```
[[1], [2], [3], [4]]
```

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

↓ transpose

1			
2			
3			
4			

# 가로세로 뒤집기 transpose 알고리즘

1. 가로세로 뒤집어 저장할 보드를 다음과 같이 초기화한다.

```
transposed = [[], [], [], []]
```

2. 가로로 읽어서 세로로 다음과 같이 차례로 붙여나간다.

```
[[], [], [], []]
```

```
[[1], [2], [3], [4]]
```

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

↓ transpose

1			
2			
3			
4			

# 가로세로 뒤집기 transpose 알고리즘

1. 가로세로 뒤집어 저장할 보드를 다음과 같이 초기화한다.

```
transposed = [[], [], [], []]
```

2. 가로로 읽어서 세로로 다음과 같이 차례로 붙여나간다.

```
[[], [], [], []]
```

```
[[1], [2], [3], [4]]
```

```
[[1, 5], [2, 6], [3, 7], [4, 8]]
```

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

↓ transpose

1	5		
2	6		
3	7		
4	8		



# 가로세로 뒤집기 transpose 알고리즘

1. 가로세로 뒤집어 저장할 보드를 다음과 같이 초기화한다.

```
transposed = [[], [], [], []]
```

2. 가로로 읽어서 세로로 다음과 같이 차례로 붙여나간다.

```
[[], [], [], []]
```

```
[[1], [2], [3], [4]]
```

```
[[1, 5], [2, 6], [3, 7], [4, 8]]
```

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

↓ transpose

1	5		
2	6		
3	7		
4	8		

# 가로세로 뒤집기 transpose 알고리즘

1. 가로세로 뒤집어 저장할 보드를 다음과 같이 초기화한다.

```
transposed = [[], [], [], []]
```

2. 가로로 읽어서 세로로 다음과 같이 차례로 붙여나간다.

```
[[], [], [], []]
```

```
[[1], [2], [3], [4]]
```

```
[[1, 5], [2, 6], [3, 7], [4, 8]]
```

```
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

↓ transpose

1	5	9	
2	6	10	
3	7	11	
4	8	12	

# 가로세로 뒤집기 transpose 알고리즘

1. 가로세로 뒤집어 저장할 보드를 다음과 같이 초기화한다.

```
transposed = [[], [], [], []]
```

2. 가로로 읽어서 세로로 다음과 같이 차례로 붙여나간다.

```
[[], [], [], []]
```

```
[[1], [2], [3], [4]]
```

```
[[1, 5], [2, 6], [3, 7], [4, 8]]
```

```
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

↓ transpose

1	5	9	
2	6	10	
3	7	11	
4	8	12	

# 가로세로 뒤집기 transpose 알고리즘

1. 가로세로 뒤집어 저장할 보드를 다음과 같이 초기화한다.

```
transposed = [[], [], [], []]
```

2. 가로로 읽어서 세로로 다음과 같이 차례로 붙여나간다.

```
[[], [], [], []]
```

```
[[1], [2], [3], [4]]
```

```
[[1, 5], [2, 6], [3, 7], [4, 8]]
```

```
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

```
[[1, 5, 9, 13], [2, 6, 10, 14], [3, 7, 11, 15], [4, 8, 12, 16]]
```

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

↓ transpose

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

# 스도쿠 정답보드 만들기 : 단계 2 구현 (완성)

code : 8-10.py

```
1 def create_solution_board_4x4():
2     board = initialize_board_4x4()
3     board = shuffle_ribbons(board)
4     board = transpose(board)
5     board = shuffle_ribbons(board)
6     board = transpose(board)
7     return board
```



# 4 x 4 미니 스도쿠 알고리즘

	0	1	2	3
0	$n_1$	$n_2$	$n_3$	$n_4$
1	$n_3$	$n_4$	$n_1$	$n_2$
2	$n_2$	$n_1$	$n_4$	$n_3$
3	$n_4$	$n_3$	$n_2$	$n_1$

1. 무작위로 스도쿠 정답보드 `solution_board`를 만든다.
2. `solution_board`를 복제하여 `puzzle_board`를 하나 만든다.
3. 사용자에게 난이도를 선택하게 하여 빈칸의 개수 `no_of_holes`를 정한다.
4. `puzzle_board`에 `no_of_holes`만큼 무작위로 선택하여 0으로 채운다.
5. `puzzle_board`를 실행창에 정한 형식대로 보여준다. 실행창에는 빈칸을 (0이 아닌) 점으로 표시한다.
6. 다음 절차를 `no_of_holes`가 0이 될 때까지 반복한다.
  - (a) 숫자를 채울 빈칸의 가로줄번호  $i$ , 세로줄번호  $j$ 를 차례로 입력받는다.
  - (b)  $(i, j)$  위치에 있는 숫자가 0이 아니면 빈칸이 아니므로 재입력받는다.
  - (c) 빈칸이면, 숫자(1,2,3,4)  $n$ 을 입력받는다.
  - (d)  $n$ 이 `solution_board[i][j]`와 같으면, `puzzle_board[i][j]`에 그 숫자를 채우고, 갱신한 `puzzle_board`를 보여준다.
  - (e) 이 숫자가 `solution_board[i][j]`와 다르면, 줄 번호부터 모두 다시 재입력 받는다.

# 4 x 4 미니 스도쿠 구현

code : 8-11.py

	0	1	2	3
0	$n_1$	$n_2$	$n_3$	$n_4$
1	$n_3$	$n_4$	$n_1$	$n_2$
2	$n_2$	$n_1$	$n_4$	$n_3$
3	$n_4$	$n_3$	$n_2$	$n_1$

```
1 def sudoku_mini():
2     solution_board = create_solution_board_4x4()
3     puzzle_board = copy_board(solution_board)
4     no_of_holes = get_level()
5     puzzle_board = make_holes(puzzle_board, no_of_holes)
6     show_board(puzzle_board)
7     while no_of_holes > 0:
8         i = get_integer("Row#(1,2,3,4): ",1,4) - 1
9         j = get_integer("Column#(1,2,3,4): ",1,4) - 1
10        if puzzle_board[i][j] != 0:
11            print("Not empty!")
12            continue
13        n = get_integer("Number(1,2,3,4): ",1,4)
14        if n == solution_board[i][j]:
15            puzzle_board[i][j] = solution_board[i][j]
16            show_board(puzzle_board)
17            no_of_holes -= 1
18        else:
19            print(n,": Wrong number! Try again.")
20    print("Well done! Come again.")
```



# 4 x 4 미니 스도쿠 알고리즘

	0	1	2	3
0	$n_1$	$n_2$	$n_3$	$n_4$
1	$n_3$	$n_4$	$n_1$	$n_2$
2	$n_2$	$n_1$	$n_4$	$n_3$
3	$n_4$	$n_3$	$n_2$	$n_1$

1. 무작위로 스도쿠 정답보드 `solution_board`를 만든다.
2. `solution_board`를 복제하여 `puzzle_board`를 하나 만든다.
3. 사용자에게 난이도를 선택하게 하여 빈칸의 개수 `no_of_holes`를 정한다.
4. `puzzle_board`에 `no_of_holes`만큼 무작위로 선택하여 0으로 채운다.
5. `puzzle_board`를 실행창에 정한 형식대로 보여준다. 실행창에는 빈칸을 (0이 아닌) 점으로 표시한다.
6. 다음 절차를 `no_of_holes`가 0이 될 때까지 반복한다.
  - (a) 숫자를 채울 빈칸의 가로줄번호  $i$ , 세로줄번호  $j$ 를 차례로 입력받는다.
  - (b)  $(i, j)$  위치에 있는 숫자가 0이 아니면 빈칸이 아니므로 재입력받는다.
  - (c) 빈칸이면, 숫자(1,2,3,4)  $n$ 을 입력받는다.
  - (d)  $n$ 이 `solution_board[i][j]`와 같으면, `puzzle_board[i][j]`에 그 숫자를 채우고, 갱신한 `puzzle_board`를 보여준다.
  - (e) 이 숫자가 `solution_board[i][j]`와 다르면, 줄 번호부터 모두 다시 재입력 받는다.

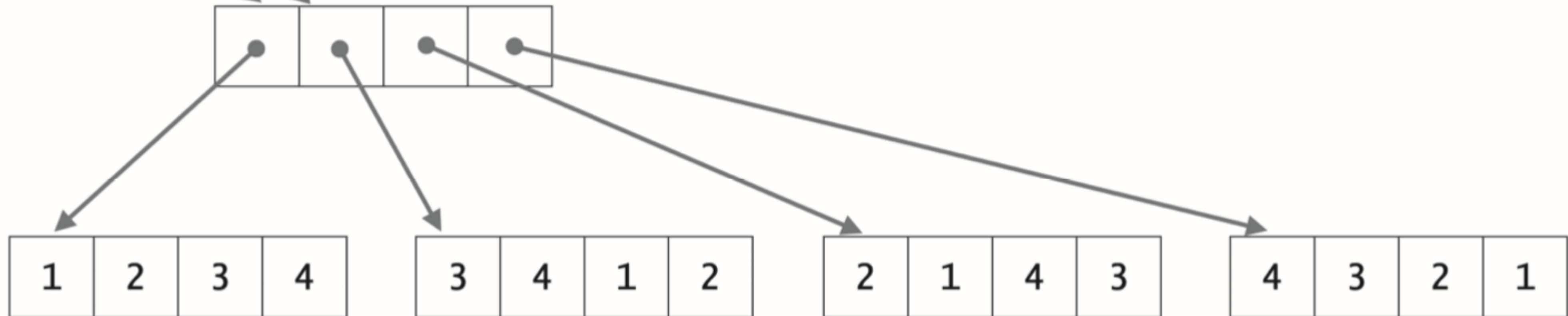


```
puzzle_board = solution_board
```

`puzzle_board = solution_board`

`solution_board`

`puzzle_board`

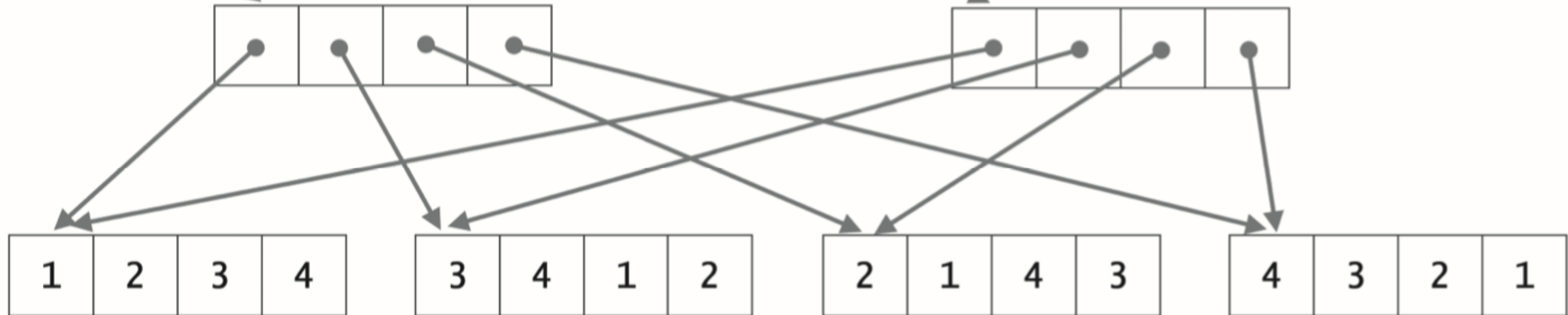


```
puzzle_board = solution_board[:]
```

```
puzzle_board = solution_board[:]
```

solution\_board

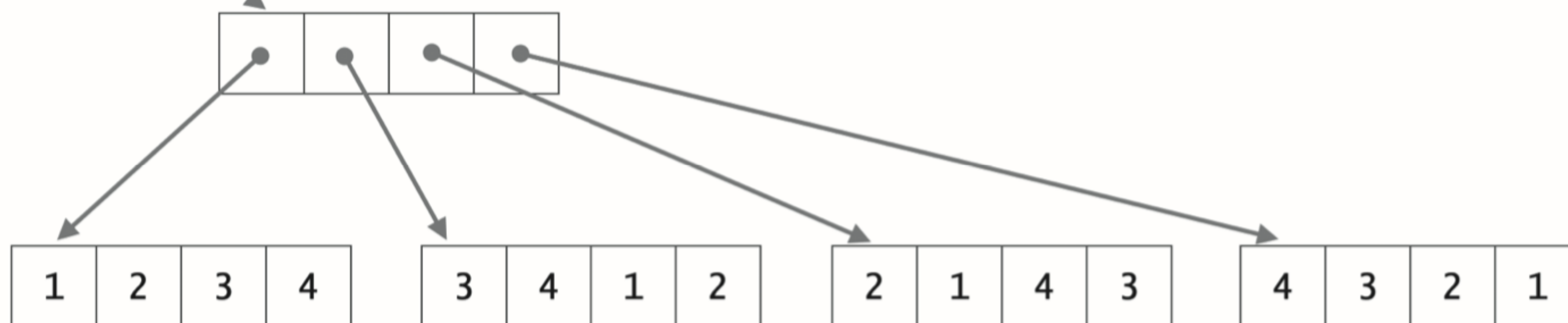
puzzle\_board



```
puzzle_board = copy_board(solution_board)
```

```
1 def copy_board(board):  
2     board_clone = []  
3     for row in board:  
4         board_clone.append(row[:])  
5     return board_clone
```

solution\_board



puzzle\_board

# 4 x 4 미니 스도쿠 구현

code : 8-11.py

	0	1	2	3
0	$n_1$	$n_2$	$n_3$	$n_4$
1	$n_3$	$n_4$	$n_1$	$n_2$
2	$n_2$	$n_1$	$n_4$	$n_3$
3	$n_4$	$n_3$	$n_2$	$n_1$

```
1 def sudoku_mini():
2     solution_board = create_solution_board_4x4()
3     puzzle_board = copy_board(solution_board)
4     no_of_holes = get_level()
5     puzzle_board = make_holes(puzzle_board, no_of_holes)
6     show_board(puzzle_board)
7     while no_of_holes > 0:
8         i = get_integer("Row#(1,2,3,4): ", 1, 4) - 1
9         j = get_integer("Column#(1,2,3,4): ", 1, 4) - 1
10        if puzzle_board[i][j] != 0:
11            print("Not empty!")
12            continue
13        n = get_integer("Number(1,2,3,4): ", 1, 4)
14        if n == solution_board[i][j]:
15            puzzle_board[i][j] = solution_board[i][j]
16            show_board(puzzle_board)
17            no_of_holes -= 1
18        else:
19            print(n, ": Wrong number! Try again.")
20    print("Well done! Come again.")
```



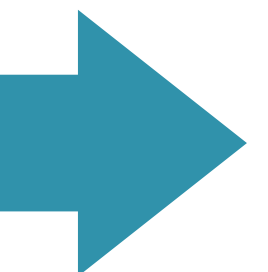
# 4 x 4 미니 스도쿠 알고리즘

1. 무작위로 스도쿠 정답보드 solution\_board를 만든다.
2. solution\_board를 복제하여 puzzle\_board를 하나 만든다.
3. 사용자에게 난이도를 선택하게 하여 빈칸의 개수 no\_of\_holes를 정한다.
4. puzzle\_board에 no\_of\_holes만큼 무작위로 선택하여 0으로 채운다.
5. puzzle\_board를 실행창에 정한 형식대로 보여준다. 실행창에는 빈칸을 (0이 아닌) 점으로 표시한다.
6. 다음 절차를 no\_of\_holes가 0이 될 때까지 반복한다.
  - (a) 숫자를 채울 빈칸의 가로줄번호 i, 세로줄번호 j를 차례로 입력받는다.
  - (b) (i, j) 위치에 있는 숫자가 0이 아니면 빈칸이 아니므로 재입력받는다.
  - (c) 빈칸이면, 숫자(1,2,3,4) n을 입력받는다.
  - (d) n이 solution\_board[i][j]와 같으면, puzzle\_board[i][j]에 그 숫자를 채우고, 갱신한 puzzle\_board를 보여준다.
  - (e) 이 숫자가 solution\_board[i][j]와 다르면, 줄 번호부터 모두 다시 재입력 받는다.

	0	1	2	3
0	n <sub>1</sub>	n <sub>2</sub>	n <sub>3</sub>	n <sub>4</sub>
1	n <sub>3</sub>	n <sub>4</sub>	n <sub>1</sub>	n <sub>2</sub>
2	n <sub>2</sub>	n <sub>1</sub>	n <sub>4</sub>	n <sub>3</sub>
3	n <sub>4</sub>	n <sub>3</sub>	n <sub>2</sub>	n <sub>1</sub>

code : 8-13.py

```
1 def get_level():
2     print("Enter your level.")
3     level = input("Beginner=1, Intermediate=2, Advanced=3: ")
4     while level not in ("1","2","3"):
5         level = input("Beginner=1, Intermediate=2, Advanced=3: ")
6     if level == "1":
7         return 6
8     elif level == "2":
9         return 8
10    else:
11        return 10
```





# 4 x 4 미니 스도쿠 구현

code : 8-11.py

	0	1	2	3
0	$n_1$	$n_2$	$n_3$	$n_4$
1	$n_3$	$n_4$	$n_1$	$n_2$
2	$n_2$	$n_1$	$n_4$	$n_3$
3	$n_4$	$n_3$	$n_2$	$n_1$

```
1 def sudoku_mini():
2     solution_board = create_solution_board_4x4()
3     puzzle_board = copy_board(solution_board)
4     no_of_holes = get_level()
5     puzzle_board = make_holes(puzzle_board, no_of_holes)
6     show_board(puzzle_board)
7     while no_of_holes > 0:
8         i = get_integer("Row#(1,2,3,4): ",1,4) - 1
9         j = get_integer("Column#(1,2,3,4): ",1,4) - 1
10        if puzzle_board[i][j] != 0:
11            print("Not empty!")
12            continue
13        n = get_integer("Number(1,2,3,4): ",1,4)
14        if n == solution_board[i][j]:
15            puzzle_board[i][j] = solution_board[i][j]
16            show_board(puzzle_board)
17            no_of_holes -= 1
18        else:
19            print(n,": Wrong number! Try again.")
20    print("Well done! Come again.")
```

# 4 x 4 미니 스도쿠 알고리즘

1. 무작위로 스도쿠 정답보드 solution\_board를 만든다.
2. solution\_board를 복제하여 puzzle\_board를 하나 만든다.
3. 사용자에게 난이도를 선택하게 하여 빈칸의 개수 no\_of\_holes를 정한다.
4. puzzle\_board에 no\_of\_holes만큼 무작위로 선택하여 0으로 채운다.
5. puzzle\_board를 실행창에 정한 형식대로 보여준다. 실행창에는 빈칸을 (0이 아닌) 점으로 표시한다.
6. 다음 절차를 no\_of\_holes가 0이 될 때까지 반복한다.
  - (a) 숫자를 채울 빈칸의 가로줄번호 i, 세로줄번호 j를 차례로 입력받는다.
  - (b) (i, j) 위치에 있는 숫자가 0이 아니면 빈칸이 아니므로 재입력받는다.
  - (c) 빈칸이면, 숫자(1,2,3,4) n을 입력받는다.
  - (d) n이 solution\_board[i][j]와 같으면, puzzle\_board[i][j]에 그 숫자를 채우고, 갱신한 puzzle\_board를 보여준다.
  - (e) 이 숫자가 solution\_board[i][j]와 다르면, 줄 번호부터 모두 다시 재입력 받는다.

	0	1	2	3
0	n <sub>1</sub>	n <sub>2</sub>	n <sub>3</sub>	n <sub>4</sub>
1	n <sub>3</sub>	n <sub>4</sub>	n <sub>1</sub>	n <sub>2</sub>
2	n <sub>2</sub>	n <sub>1</sub>	n <sub>4</sub>	n <sub>3</sub>
3	n <sub>4</sub>	n <sub>3</sub>	n <sub>2</sub>	n <sub>1</sub>

>>>>>>>>>> 제어 구조의 설계 원리를 중심으로 배우는 >>>>>>>>>>

# 프로그래밍의 정석 파이썬

도경구 지음



pp.395~396



## 실습 8.7 퍼즐보드 만들기

code : 8-14.py

```
1 def make_holes(board, no_of_holes):
2     while no_of_holes > 0:
3         i = random.randint(0,3)
4         j = random.randint(0,3)
5
6
7
8     return board
```



# 4 x 4 미니 스도쿠 구현

code : 8-11.py

	0	1	2	3
0	$n_1$	$n_2$	$n_3$	$n_4$
1	$n_3$	$n_4$	$n_1$	$n_2$
2	$n_2$	$n_1$	$n_4$	$n_3$
3	$n_4$	$n_3$	$n_2$	$n_1$

```
1 def sudoku_mini():
2     solution_board = create_solution_board_4x4()
3     puzzle_board = copy_board(solution_board)
4     no_of_holes = get_level()
5     puzzle_board = make_holes(puzzle_board, no_of_holes)
6     show_board(puzzle_board)
7     while no_of_holes > 0:
8         i = get_integer("Row#(1,2,3,4): ", 1, 4) - 1
9         j = get_integer("Column#(1,2,3,4): ", 1, 4) - 1
10        if puzzle_board[i][j] != 0:
11            print("Not empty!")
12            continue
13        n = get_integer("Number(1,2,3,4): ", 1, 4)
14        if n == solution_board[i][j]:
15            puzzle_board[i][j] = solution_board[i][j]
16            show_board(puzzle_board)
17            no_of_holes -= 1
18        else:
19            print(n, ": Wrong number! Try again.")
20    print("Well done! Come again.")
```

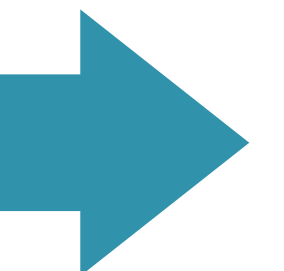
# 4 x 4 미니 스도쿠 알고리즘

1. 무작위로 스도쿠 정답보드 `solution_board`를 만든다.
2. `solution_board`를 복제하여 `puzzle_board`를 하나 만든다.
3. 사용자에게 난이도를 선택하게 하여 빈칸의 개수 `no_of_holes`를 정한다.
4. `puzzle_board`에 `no_of_holes`만큼 무작위로 선택하여 0으로 채운다.
5. `puzzle_board`를 실행창에 정한 형식대로 보여준다. 실행창에는 빈칸을 (0이 아닌) 점으로 표시한다.
6. 다음 절차를 `no_of_holes`가 0이 될 때까지 반복한다.
  - (a) 숫자를 채울 빈칸의 가로줄번호  $i$ , 세로줄번호  $j$ 를 차례로 입력받는다.
  - (b)  $(i, j)$  위치에 있는 숫자가 0이 아니면 빈칸이 아니므로 재입력받는다.
  - (c) 빈칸이면, 숫자(1,2,3,4)  $n$ 을 입력받는다.
  - (d)  $n$ 이 `solution_board[i][j]`와 같으면, `puzzle_board[i][j]`에 그 숫자를 채우고, 갱신한 `puzzle_board`를 보여준다.
  - (e) 이 숫자가 `solution_board[i][j]`와 다르면, 줄 번호부터 모두 다시 재입력 받는다.

	0	1	2	3
0	$n_1$	$n_2$	$n_3$	$n_4$
1	$n_3$	$n_4$	$n_1$	$n_2$
2	$n_2$	$n_1$	$n_4$	$n_3$
3	$n_4$	$n_3$	$n_2$	$n_1$

code : 8-15.py

```
1 def show_board(board):  
2     for row in board:  
3         for entry in row:  
4             if entry == 0:  
5                 print('.', end=' ')  
6             else:  
7                 print(entry, end=' ')  
8     print()
```





# 4 x 4 미니 스도쿠 구현

code : 8-11.py

	0	1	2	3
0	$n_1$	$n_2$	$n_3$	$n_4$
1	$n_3$	$n_4$	$n_1$	$n_2$
2	$n_2$	$n_1$	$n_4$	$n_3$
3	$n_4$	$n_3$	$n_2$	$n_1$

```
1 def sudoku_mini():
2     solution_board = create_solution_board_4x4()
3     puzzle_board = copy_board(solution_board)
4     no_of_holes = get_level()
5     puzzle_board = make_holes(puzzle_board, no_of_holes)
6     show_board(puzzle_board)
7     while no_of_holes > 0:
8         i = get_integer("Row#(1,2,3,4): ", 1, 4) - 1
9         j = get_integer("Column#(1,2,3,4): ", 1, 4) - 1
10        if puzzle_board[i][j] != 0:
11            print("Not empty!")
12            continue
13        n = get_integer("Number(1,2,3,4): ", 1, 4)
14        if n == solution_board[i][j]:
15            puzzle_board[i][j] = solution_board[i][j]
16            show_board(puzzle_board)
17            no_of_holes -= 1
18        else:
19            print(n, ": Wrong number! Try again.")
20    print("Well done! Come again.")
```

# 4 x 4 미니 스도쿠 알고리즘

1. 무작위로 스도쿠 정답보드 solution\_board를 만든다.
2. solution\_board를 복제하여 puzzle\_board를 하나 만든다.
3. 사용자에게 난이도를 선택하게 하여 빈칸의 개수 no\_of\_holes를 정한다.
4. puzzle\_board에 no\_of\_holes만큼 무작위로 선택하여 0으로 채운다.
5. puzzle\_board를 실행창에 정한 형식대로 보여준다. 실행창에는 빈칸을 (0이 아닌) 점으로 표시한다.

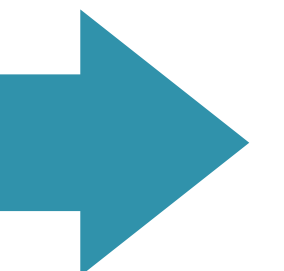
6. 다음 절차를 no\_of\_holes가 0이 될 때까지 반복한다.
  - (a) 숫자를 채울 빈칸의 가로줄번호 i, 세로줄번호 j를 차례로 입력받는다.
  - (b) (i, j) 위치에 있는 숫자가 0이 아니면 빈칸이 아니므로 재입력받는다.
  - (c) 빈칸이면, 숫자(1,2,3,4) n을 입력받는다.
  - (d) n이 solution\_board[i][j]와 같으면, puzzle\_board[i][j]에 그 숫자를 채우고, 갱신한 puzzle\_board를 보여준다.
  - (e) 이 숫자가 solution\_board[i][j]와 다르면, 줄 번호부터 모두 다시 재입력 받는다.

		Column#				
		1	2	3	4	
		0	1	2	3	
Row#	1	0	$n_1$	$n_2$	$n_3$	$n_4$
	2	1	$n_3$	$n_4$	$n_1$	$n_2$
	3	2	$n_2$	$n_1$	$n_4$	$n_3$
	4	3	$n_4$	$n_3$	$n_2$	$n_1$



code : 8-16.py

```
1 def get_integer(message, i, j):  
2     number = input(message)  
3     while not (number.isdigit() and i <= int(number) <= j):  
4         number = input(message)  
5     return int(number)
```



# 4 x 4 미니 스도쿠 구현

		Column#				
		1	2	3	4	
		0	1	2	3	
Row#	1	0	$n_1$	$n_2$	$n_3$	$n_4$
	2	1	$n_3$	$n_4$	$n_1$	$n_2$
	3	2	$n_2$	$n_1$	$n_4$	$n_3$
	4	3	$n_4$	$n_3$	$n_2$	$n_1$

```

1 def sudoku_mini():
2     solution_board = create_solution_board_4x4()
3     puzzle_board = copy_board(solution_board)
4     no_of_holes = get_level()
5     puzzle_board = make_holes(puzzle_board, no_of_holes)
6     show_board(puzzle_board)
7     while no_of_holes > 0:
8         i = get_integer("Row#(1,2,3,4): ",1,4) - 1
9         j = get_integer("Column#(1,2,3,4): ",1,4) - 1
10        if puzzle_board[i][j] != 0:
11            print("Not empty!")
12            continue
13        n = get_integer("Number(1,2,3,4): ",1,4)
14        if n == solution_board[i][j]:
15            puzzle_board[i][j] = solution_board[i][j]
16            show_board(puzzle_board)
17            no_of_holes -= 1
18        else:
19            print(n,": Wrong number! Try again.")
20    print("Well done! Come again.")

```

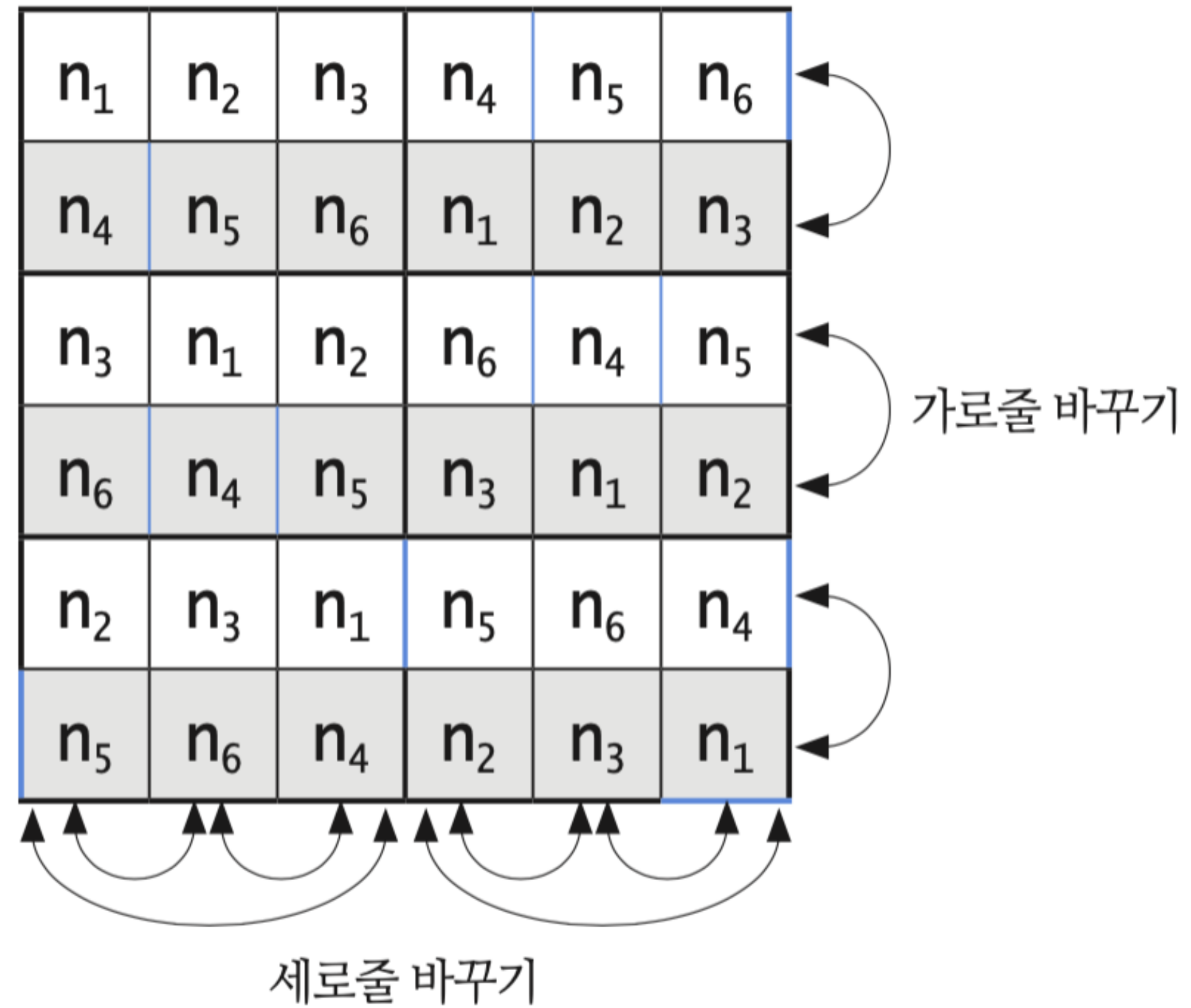
# 4 x 4 미니 스도쿠 구현

code : 8-11.py

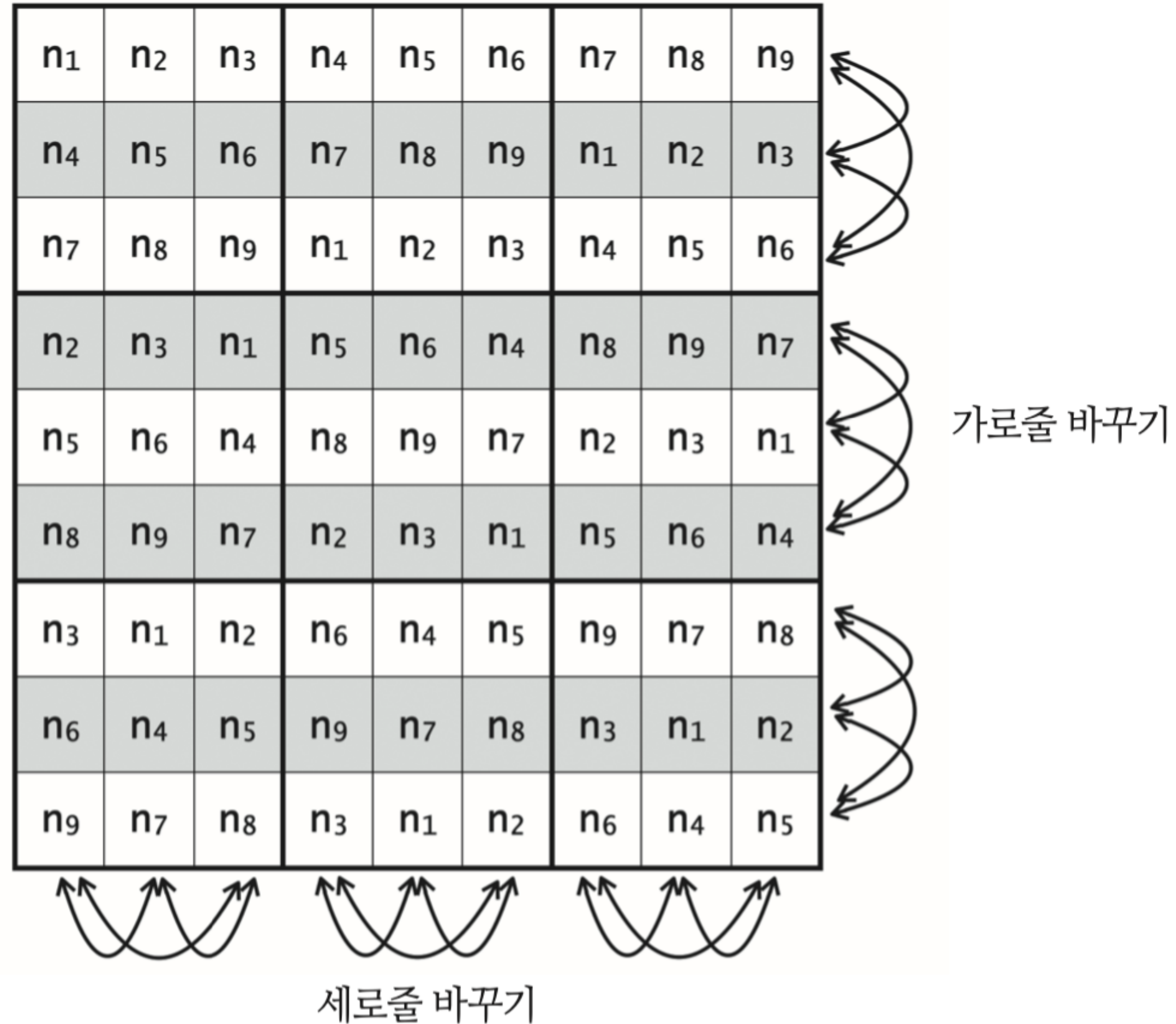
	0	1	2	3
0	$n_1$	$n_2$	$n_3$	$n_4$
1	$n_3$	$n_4$	$n_1$	$n_2$
2	$n_2$	$n_1$	$n_4$	$n_3$
3	$n_4$	$n_3$	$n_2$	$n_1$

```
1 def sudoku_mini():
2     solution_board = create_solution_board_4x4()
3     puzzle_board = copy_board(solution_board)
4     no_of_holes = get_level()
5     puzzle_board = make_holes(puzzle_board, no_of_holes)
6     show_board(puzzle_board)
7     while no_of_holes > 0:
8         i = get_integer("Row#(1,2,3,4): ",1,4) - 1
9         j = get_integer("Column#(1,2,3,4): ",1,4) - 1
10        if puzzle_board[i][j] != 0:
11            print("Not empty!")
12            continue
13        n = get_integer("Number(1,2,3,4): ",1,4)
14        if n == solution_board[i][j]:
15            puzzle_board[i][j] = solution_board[i][j]
16            show_board(puzzle_board)
17            no_of_holes -= 1
18        else:
19            print(n,": Wrong number! Try again.")
20        print("Well done! Come again.")
```

# 프로젝트 옵션 #1 : 6 x 6 스도쿠



## 프로젝트 옵션 #2 : 9 x 9 스도쿠





>>>>>>>>>> 제어 구조의 설계 원리를 중심으로 배우는 >>>>>>>>>>

# 프로그래밍의 정석

# 파이썬

도경구 지음



CHAPTER 8

프로젝트 기반 학습 I  
퍼즐게임 스도쿠