



이차원 리스트

1. 이차원 리스트

2. 입력 받기

3. 순회

4. 전치

5. 회전



1. 이차원 리스트

이차원 리스트는 **리스트를 원소로 가지는 리스트**일 뿐이다.

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

이차원 리스트는 **리스트를 원소로 가지는 리스트**일 뿐이다.

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

0 1 2 0 1 2 0 1 2
0 1 2

이차원 리스트는 **리스트를 원소로 가지는 리스트**일 뿐이다.

	0	1	2		0	1	2		0	1	2		
matrix =	[1,	2,	3],	[4,	5,	6],	[7,	8,	9]]
		0				1				2			

```
print(matrix[0])
```

```
>>> [1, 2, 3]
```

```
print(matrix[1])
```

```
>>> [4, 5, 6]
```

```
print(matrix[2])
```

```
>>> [7, 8, 9]
```

이차원 리스트는 **리스트를 원소로 가지는 리스트**일 뿐이다.

	0	1	2		0	1	2		0	1	2			
matrix =	[[1,	2,	3],	[4,	5,	6],	[7,	8,	9]]
			0				1				2			

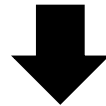
```
print(matrix[0][0])
>>> 1

print(matrix[1][2])
>>> 6

print(matrix[2][0])
>>> 7
```

이차원 리스트는 **리스트를 원소로 가지는 리스트**일 뿐이다.

	0	1	2		0	1	2		0	1	2	
matrix =	[[1, 2, 3], [4, 5, 6], [7, 8, 9]]											
	0			1			2					



보기 좋게 변경하면 **행렬(matrix)**의 형태가 나온다 !

```
matrix = [  
    [1, 2, 3],  
    [4, 5, 6],  
    [7, 8, 9]  
]
```


이차원 리스트는 **리스트를 원소로 가지는 리스트**일 뿐이다.

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

0 1 2 0 1 2 0 1 2
0 1 2



보기 좋게 변경하면 **행렬(matrix)**의 형태가 나온다 !

```
matrix = [  
  0 [1, 2, 3],  
  1 [4, 5, 6],  
  2 [7, 8, 9]  
]      0 1 2
```

이차원 리스트는 **행렬(matrix)**이다.

```
matrix = [  
    [1, 2, 3],  
    [4, 5, 6],  
    [7, 8, 9]  
]
```

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

이차원 리스트는 **행렬(matrix)**이다.

```
matrix = [  
    [1, 2, 3],  
    [4, 5, 6],  
    [7, 8, 9]  
]
```

```
print(matrix[0][0])  
>>> ???
```

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

이차원 리스트는 **행렬(matrix)**이다.

```
matrix = [  
    [1, 2, 3],  
    [4, 5, 6],  
    [7, 8, 9]  
]
```

```
print(matrix[0][0])  
>>> 1
```

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

이차원 리스트는 **행렬(matrix)**이다.

```
matrix = [  
    [1, 2, 3],  
    [4, 5, 6],  
    [7, 8, 9]  
]
```

```
print(matrix[1][2])  
>>> ???
```

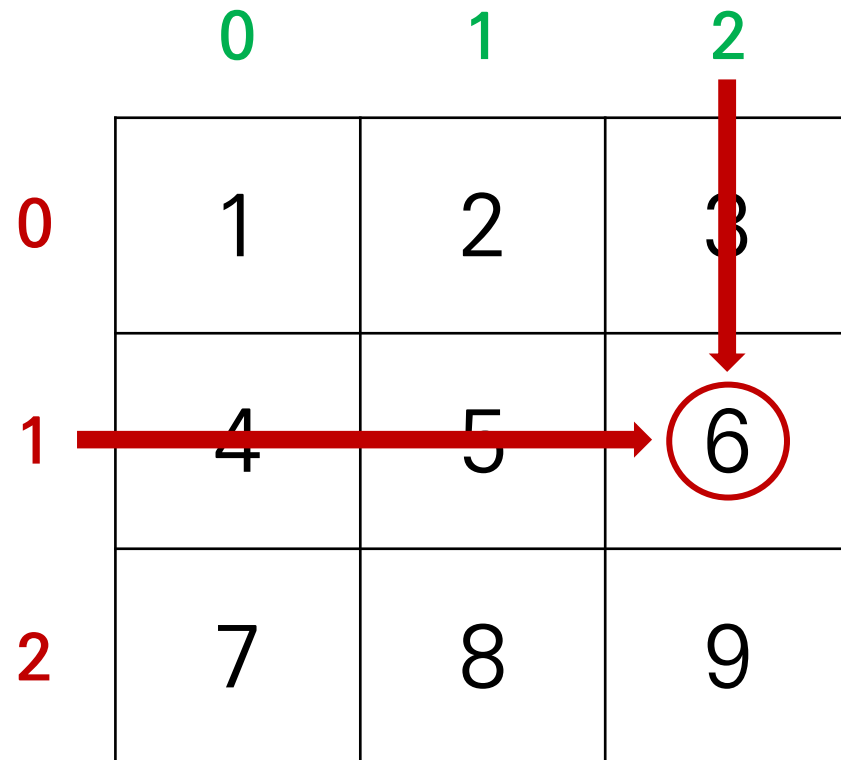
	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

이차원 리스트는 **행렬(matrix)**이다.

```
matrix = [  
    [1, 2, 3],  
    [4, 5, 6],  
    [7, 8, 9]  
]
```

```
print(matrix[1][2])  
>>> 6
```

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9



특정 값으로 초기화 된 이차원 리스트 만들기

1. 직접 작성 (4 x 3 행렬)

```
matrix1 = [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]

matrix2 = [
    [0, 0, 0],
    [0, 0, 0],
    [0, 0, 0],
    [0, 0, 0]
]
```

특정 값으로 초기화 된 이차원 리스트 만들기

1. 직접 작성 (4 x 3 행렬) 100 x 100이라면???

```
matrix1 = [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

```
matrix2 = [  
    [0, 0, 0],  
    [0, 0, 0],  
    [0, 0, 0],  
    [0, 0, 0]  
]
```


특정 값으로 초기화 된 이차원 리스트 만들기

2. 반복문으로 작성 (100 x 100 행렬)

```
matrix = []  
  
for _ in range(100):  
    matrix.append([0] * 100)
```

특정 값으로 초기화 된 이차원 리스트 만들기

2. 반복문으로 작성 (n x m 행렬)

```
n = 4 # 행
m = 3 # 열
matrix = []

for _ in range(n):
    matrix.append([0] * m)

print(matrix)
>>> [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

특정 값으로 초기화 된 이차원 리스트 만들기

3. 리스트 컴프리헨션으로 작성 (n x m 행렬)

```
n = 4 # 행
m = 3 # 열

matrix = [[0] * m for _ in range(n)]

print(matrix)
>>> [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

[주의] 리스트 컴프리헨션 vs 리스트 곱셈 연산

```
n = 4 # 행
m = 3 # 열

matrix1 = [[0] * m for _ in range(n)]
matrix2 = [[0] * m] * n

print(matrix1)
>>> ???
print(matrix2)
>>> ???
```

[주의] 리스트 컴프리헨션 vs 리스트 곱셈 연산

똑같다?

```
n = 4 # 행
m = 3 # 열

matrix1 = [[0] * m for _ in range(n)]
matrix2 = [[0] * m] * n

print(matrix1)
>>> [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
print(matrix2)
>>> [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

[주의] 리스트 컴프리헨션 vs 리스트 곱셈 연산

원소 값 변경

```
n = 4 # 행
m = 3 # 열

matrix1 = [[0] * m for _ in range(n)]
matrix2 = [[0] * m] * n

matrix1[0][0] = 1
matrix2[0][0] = 1

print(matrix1)
>>> ???
print(matrix2)
>>> ???
```

[주의] 리스트 컴프리헨션 vs 리스트 곱셈 연산

```
n = 4 # 행
m = 3 # 열

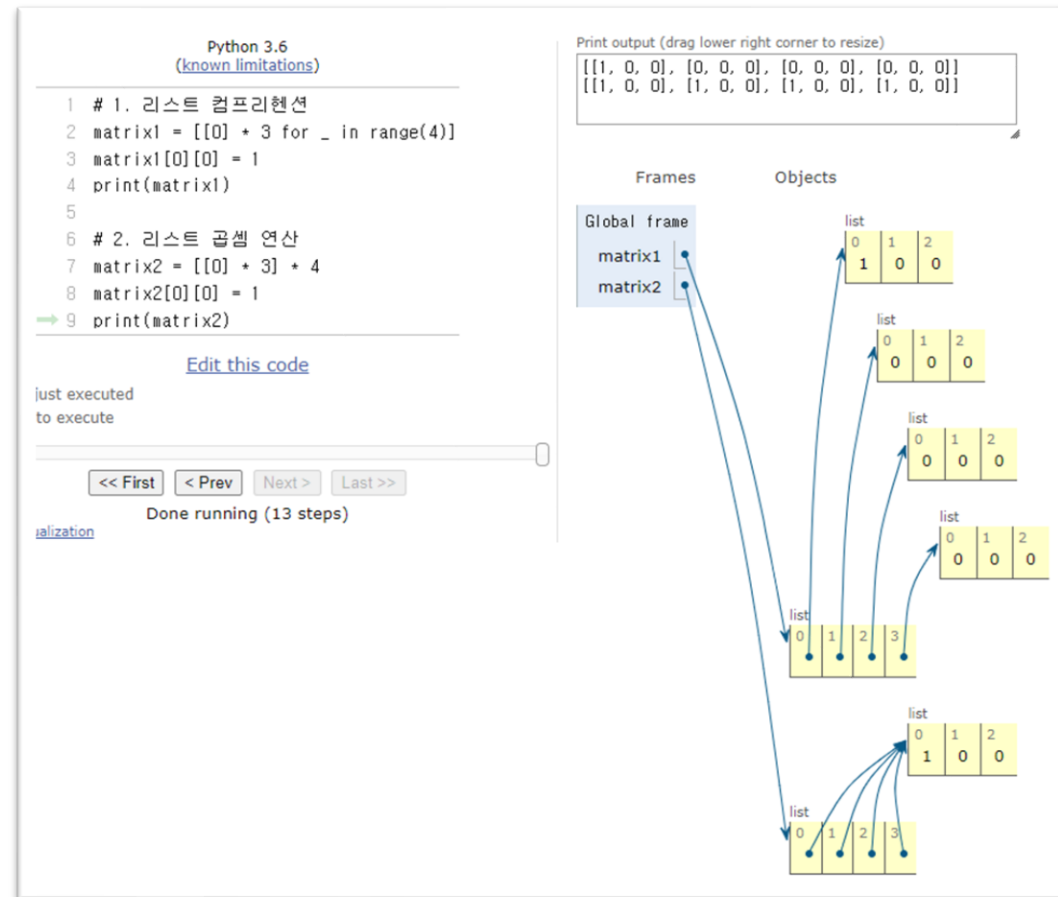
matrix1 = [[0] * m for _ in range(n)]
matrix2 = [[0] * m] * n

matrix1[0][0] = 1
matrix2[0][0] = 1

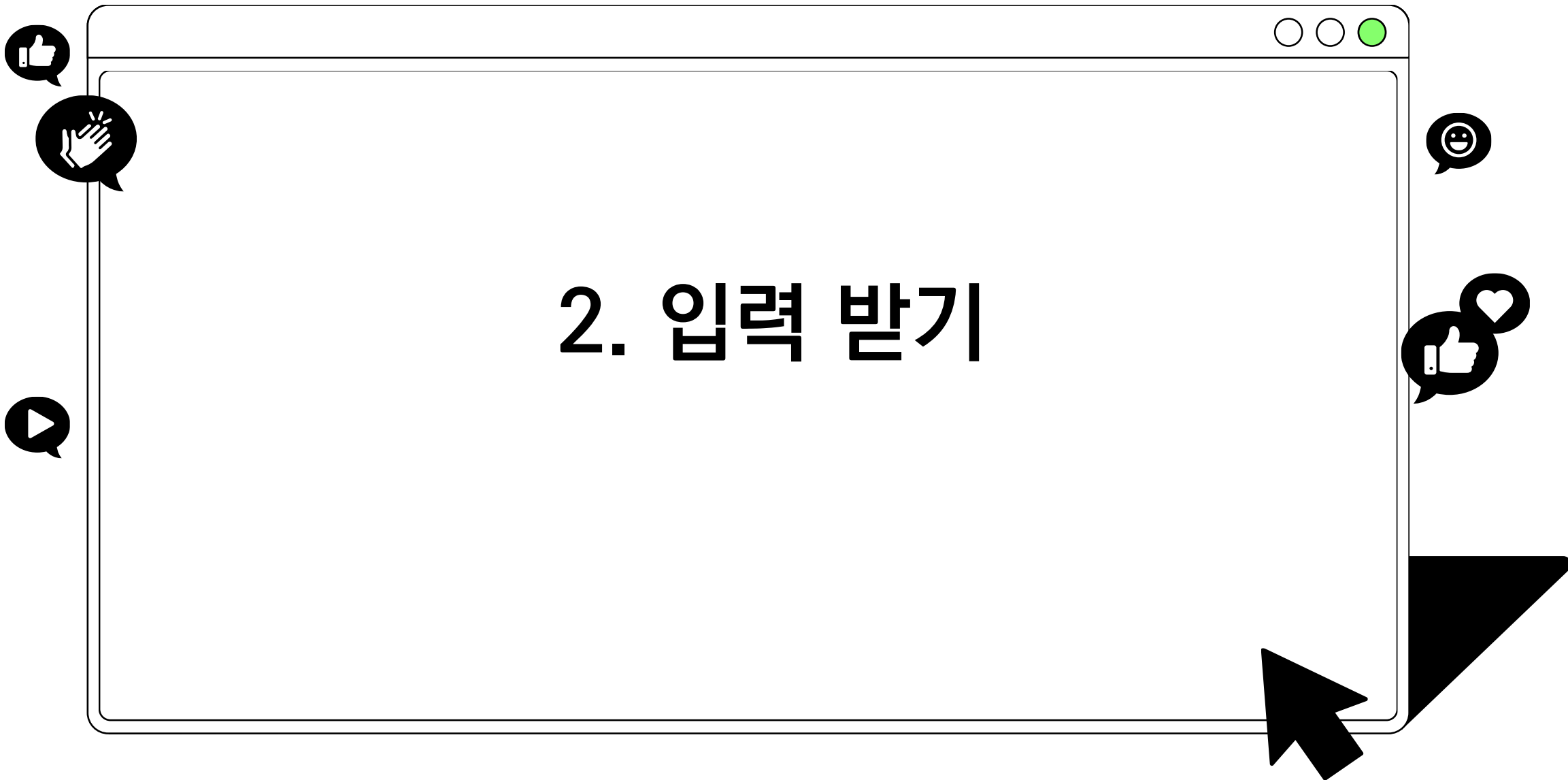
print(matrix1)
>>> [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
print(matrix2)
>>> [[1, 0, 0], [1, 0, 0], [1, 0, 0], [1, 0, 0]]
```

다르다 !

[주의] 리스트 컴프리헨션 vs 리스트 곱셈 연산



2. 입력 받기



1. 행렬의 크기가 미리 주어지는 경우

문제

체스판은 **8×8크기**이고, 검정 칸과 하얀 칸이 번갈아가면서 색칠되어 있다. 가장 왼쪽 위칸 (0,0)은 하얀색이다. 체스판의 상태가 주어졌을 때, 하얀 칸 위에 말이 몇 개 있는지 출력하는 프로그램을 작성하시오.

예제 입력 1 복사

```
.F.F...F
F...F.F.
...F.F.F
F.F...F.
.F...F..
F...F.F.
.F.F.F.F
..FF...F.
```

예제 출력 1 복사

```
1
```

1. 행렬의 크기가 미리 주어지는 경우

문제

체스판은 8×8 크기이고, 검
력하는 프로그램을 작성하시

예제 입력 1 복사

```
.F.F...F
F...F.F.
...F.F.F
F.F...F.
.F...F..
F...F.F.
.F.F.F.F
..FF...F.
```

```
matrix = []
```

```
for _ in range(8):
    line = list(input())
    matrix.append(line)
```

input 함수가 한 줄을 입력 받기 때문에 열의 크기는 사용 되지 않는다.

```
"""
matrix = [
    ['.', 'F', '.', 'F', '.', '.', '.', 'F'],
    ['F', '.', '.', '.', 'F', '.', 'F', '.'],
    ['.', '.', '.', 'F', '.', 'F', '.', 'F'],
    ['F', '.', 'F', '.', '.', '.', 'F', '.'],
    ['.', 'F', '.', '.', '.', 'F', '.', '.'],
    ['F', '.', '.', '.', 'F', '.', 'F', '.'],
    ['.', 'F', '.', 'F', '.', 'F', '.', 'F'],
    ['.', '.', 'F', 'F', '.', '.', 'F', '.']
]
```

2. 입력 받기

1. 행렬의 크기가 미리 주어지는 경우

문제

체스판은 8×8 크기이고, 검정
색을 칠하는 프로그램을 작성하시

예제 입력 1 복사

```
.F.F...F
F...F.F.
...F.F.F
F.F...F.
.F...F..
F...F.F.
.F.F.F.F
..FF...F.
```

```
matrix = [list(input()) for _ in range(8)]
```

리스트 컴프리헨션을 통해 이차원 리스트의 입력을 간단히 받을 수 있다.

```
"""
matrix = [
    ['.', 'F', '.', 'F', '.', '.', '.', 'F'],
    ['F', '.', '.', '.', 'F', '.', 'F', '.'],
    ['.', '.', '.', 'F', '.', 'F', '.', 'F'],
    ['F', '.', 'F', '.', '.', '.', 'F', '.'],
    ['.', 'F', '.', '.', '.', 'F', '.', '.'],
    ['F', '.', '.', '.', 'F', '.', 'F', '.'],
    ['.', 'F', '.', 'F', '.', 'F', '.', 'F'],
    ['.', '.', 'F', 'F', '.', '.', 'F', '.']
]
"""
```

1. 행렬의 크기가 미리 주어지는 경우

```
"""  
3 x 3 크기의 입력을 받아보자.  
  
1 2 3  
4 5 6  
7 8 9  
"""
```

1. 행렬의 크기가 미리 주어지는 경우

```
"""
3 x 3 크기의 입력을 받아보자.

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

matrix = []

for _ in range(3):
    line = list(map(int, input().split()))
    matrix.append(line)
```

1. 행렬의 크기가 미리 주어지는 경우

```
"""
3 x 3 크기의 입력을 받아보자.

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

matrix = [list(map(int, input().split())) for _ in range(3)]
```

2. 행렬의 크기가 입력으로 주어지는 경우

문제

농부 민식이가 관리하는 농장은 $N \times M$ 격자로 이루어져 있다. 민식이는 농장을 관리하기 위해 산봉우리마다 경비원을 배치하려 한다. 이를 위해 농장에 산봉우리가 총 몇 개 있는지를 세는 것이 문제다.

예제 입력 1 복사

```
8 7
4 3 2 2 1 0 1
3 3 3 2 1 0 1
2 2 2 2 1 0 0
2 1 1 1 1 0 0
1 1 0 0 0 1 0
0 0 0 1 1 1 0
0 1 2 2 1 1 0
0 1 1 1 2 1 0
```

예제 출력 1 복사

```
3
```


2. 행렬의 크기가 입력으로 주어지는 경우

문제

농부 민식이가 관리하는 농장은 $N \times M$ 격자로 이루어져 있다. 농장을 세는 것이 문제다.

예제 입력 1 복사

```
8 7
4 3 2 2 1 0 1
3 3 3 2 1 0 1
2 2 2 2 1 0 0
2 1 1 1 1 0 0
1 1 0 0 0 1 0
0 0 0 1 1 1 0
0 1 2 2 1 1 0
0 1 1 1 2 1 0
```

```
n, m = map(int, input().split()) # 8 7
matrix = []

for _ in range(n):
    line = list(map(int, input().split()))
    matrix.append(line)

"""
matrix = [
    [4, 3, 2, 2, 1, 0, 1],
    [3, 3, 3, 2, 1, 0, 1],
    [2, 2, 2, 2, 1, 0, 0],
    [2, 1, 1, 1, 1, 0, 0],
    [1, 1, 0, 0, 0, 1, 0],
    [0, 0, 0, 1, 1, 1, 0],
    [0, 1, 2, 2, 1, 1, 0],
    [0, 1, 1, 1, 2, 1, 0]
]
"""
```

2. 행렬의 크기가 입력으로 주어지는 경우

문제

농부 민식이가 관리하는 농장은 $N \times M$ 격자로 나누어져 있다. 농부 민식이는 농장을 $N \times M$ 격자로 나누어 놓는 것을 문제다.

예제 입력 1 복사

```
8 7
4 3 2 2 1 0 1
3 3 3 2 1 0 1
2 2 2 2 1 0 0
2 1 1 1 1 0 0
1 1 0 0 0 1 0
0 0 0 1 1 1 0
0 1 2 2 1 1 0
0 1 1 1 2 1 0
```

```
n, m = map(int, input().split()) # 8 7
```

```
matrix = [list(map(int, input().split())) for _ in range(n)]
```

리스트 컴프리헨션을 통해 이차원 리스트의 입력을 간단히 받을 수 있다.

```
"""
matrix = [
    [4, 3, 2, 2, 1, 0, 1],
    [3, 3, 3, 2, 1, 0, 1],
    [2, 2, 2, 2, 1, 0, 0],
    [2, 1, 1, 1, 1, 0, 0],
    [1, 1, 0, 0, 0, 1, 0],
    [0, 0, 0, 1, 1, 1, 0],
    [0, 1, 2, 2, 1, 1, 0],
    [0, 1, 1, 1, 2, 1, 0]
]
"""
```

2. 행렬의 크기가 입력으로 주어지는 경우

```
"""  
n x m 크기의 입력을 받아보자.  
  
3 4  
1 2 3 4  
5 6 7 8  
9 0 1 2  
"""
```

2. 행렬의 크기가 입력으로 주어지는 경우

```
"""
n x m 크기의 입력을 받아보자.

3 4
1 2 3 4
5 6 7 8
9 0 1 2
"""

n, m = map(int, input().split()) # 3 4
matrix = []

for _ in range(n):
    line = list(map(int, input().split()))
    matrix.append(line)
```

2. 행렬의 크기가 입력으로 주어지는 경우

```
"""
n x m 크기의 입력을 받아보자.

3 4
1 2 3 4
5 6 7 8
9 0 1 2
"""

n, m = map(int, input().split()) # 3 4

matrix = [list(map(int, input().split())) for _ in range(n)]
```