

Ch07 리스트

많은 데이터를 간편하게 (동일 알고리즘으로) 다루기 위한 데이터 구조가 필요하다.

- > 파이썬에는 **리스트(List)** 와 **튜플(Tuple)** 이 있다.
- > 리스트나 튜플은 데이터 집단에 대한 대표 변수를 정하고
- > 각 데이터 항목에는 그 변수의 순서번호(index)로 접근하는 방식 사용

리스트 자료형이 없으면

데이터 수가 바뀌면 알고리즘도 바뀜

- > 알고리즘과 데이터가 종속되어 유지보수가 어려움

```
In [2]: ## 성적 처리: 일반 변수를 활용
scr1 = 78; scr2 = 85; scr3 = 68; scr4 = 90; scr5 = 58

summ = scr1 + scr2 + scr3 + scr4 + scr5
avg = summ / 5

print(summ, avg)
```

379 75.8

리스트의 필요성

학생이 10명이면 10개의 변수가 필요하고, 이름도 각각 다르게 만들어야 함

학생이 100명이라면? 저장할 값이 많아질수록 이런 방식으로 변수를 늘려가는 방법은 사용하기 어려워짐 → 변수를 묶어서 관리하는 리스트가 필요!

학생 n명의 성적을 동일 알고리즘으로 처리 하려면

- > 학생들의 성적 전체를 score 변수로 정하고,
- > 각 학생의 성적은 score의 순서번호(index)로 접근하면 가능

데이터 수가 바뀌어도 알고리즘은 동일

> 데이터 수의 가변성은 리스트의 저장 순서인 인덱스를 사용하여 해결

```
In [3]: ## 리스트 변수: index로 읽어내기
alist = [78, 85, 68, 90, 58] # 리스트 구조

print(alist[0])
print(alist[1])
print(alist[2])
print(alist[3])
print(alist[4])
```

```
78
85
68
90
58
```

```
In [2]: ## 리스트 변수: range()로 읽어내기
alist = [78, 85, 68, 90, 58] # 리스트 구조

len_item = len(alist)
for i in range(len_item):
    print(alist[i])
```

```
78
85
68
90
58
```

```
In [4]: ## 리스트 변수: 리스트 항목을 직접 읽어내기
alist = [78, 85, 68, 90, 58] # 리스트 구조

for x in alist:
    print(x)
```

```
78
85
68
90
58
```

리스트의 구조

리스트에 저장되는 각각의 데이터를 항목(item) 또는 원소(element)라고 하며

항목은 숫자나 문자, 다른 리스트 등 다양한 종류로 구성

리스트를 만드는 방법은 대괄호([])를 붙이고 항목 간에는 쉼표(,)로 구분해서 나열

각 항목은 순서대로 인덱스(index)라고 하는 번호가 정해지며 0부터 시작

```
In [4]: ## 성적 처리: 리스트 변수를 활용 > 총점, 평균 구하기  
alist = [78, 85, 68, 90, 58] # 리스트 구조
```

```
summ = 0  
len_item = len(alist)  
for i in range(len_item):  
    summ += alist[i]  
  
avg = summ / len_item  
  
print(summ, avg)
```

379 75.8

```
In [5]: ## 성적 처리: 리스트 변수를 활용 > 총점, 평균 구하기  
alist = [78, 85, 68, 90, 58, 66, 77, 88] # 리스트 구조
```

```
summ = 0  
len_item = len(alist)  
for i in range(len_item):  
    summ += alist[i]  
  
avg = summ / len_item  
  
print(summ, avg)
```

610 76.25

리스트(list)와 배열(array)의 구조 차이

일반적인 프로그래밍 언어에서는 유사한 복수 개의 데이터를 관리하기 위해 배열 구조를 사용

> **배열(Array)**은 연속된 공간에 동일한 형식의 고정된 크기로 저장 관리를 함

> **리스트(List)**는 불연속 공간에 각 데이터 항목을 연결구조(linked list)로 저장 관리하기 때문에 데이터 항목의 삽입, 삭제가 용이하고, 각 항목의 데이터 형식이 달라도 됨

In []:

리스트 항목 초기화와 변경

값 나열로 초기화

```
In [7]: ## 리스트 항목 초기화: 값 나열로 초기화  
alist = [78, 85, 68, 90, 58] # 리스트 구조  
  
len_item = len(alist)  
for i in range(len_item):  
    print(alist[i])
```

78
85
68
90
58

빈 리스트 만들어 항목 추가: .append()

```
In [8]: ## 리스트 항목: 추가 .append()  
alist = []  
  
alist.append(78)  
alist.append(85)  
alist.append(68)  
alist.append(90)  
alist.append(58)  
  
print(alist)
```

[78, 85, 68, 90, 58]

리스트에 항목 삽입: .insert()

```
In [9]: ## 리스트 항목: 삽입 .insert()  
alist = [78, 85, 68]  
  
alist.append(58)  
alist.insert(3, 90)
```

```
print(alist)  
[78, 85, 68, 90, 58]
```

리스트 항목: index 확인 .index()

```
In [10]: ## 리스트 항목: index 확인  
alist = [78, 85, 68, 90, 58] # 리스트 구조  
  
alist.index(90)
```

```
Out[10]: 3
```

리스트 항목: index로 값 변경

```
In [11]: ## 리스트 항목: index 확인  
alist = [78, 85, 68, 90, 58] # 리스트 구조  
  
alist[3] = 99  
alist
```

```
Out[11]: [78, 85, 68, 99, 58]
```

리스트 항목: .remove()로 항목 제거

```
In [12]: ## 리스트 항목: .remove()로 항목 제거  
alist = [78, 85, 68, 90, 58] # 리스트 구조  
  
alist.remove(90)  
alist
```

```
Out[12]: [78, 85, 68, 58]
```

리스트에서 특정 항목을 제거하면 그 뒤 항목들이 당겨져서 연결 리스트 유지

> Linked List이기 때문에 가능

```
In [ ]:
```

리스트 항목 사용

리스트 항목: index로 접근

```
In [13]: ## 리스트 항목: index로 접근
alist = [78, 85, 68, 90, 58] # 리스트 구조

print(alist[0])
print(alist[1])
print(alist[3])
```

```
78
85
90
```

```
In [14]: ## 리스트 항목: index로 접근
alist = [78, 85, 68, 90, 58] # 리스트 구조

for i in range(len(alist)):
    print(alist[i])
```

```
78
85
68
90
58
```

```
In [15]: ## 리스트 항목: index로 접근
alist = [78, 85, 68, 90, 58] # 리스트 구조

print(alist[-1]) # 맨 뒤에서 1번째
print(alist[-2])
print(alist[-3])
```

```
58
90
68
```

리스트 항목: .pop()으로 접근

.pop()은 맨 뒤 항목을 추출 함. 추출된 항목은 리스트에서 제거

> 맨 뒤 항목에 대해서만 적용

```
In [17]: ## 리스트 항목: .pop()으로 접근
alist = [78, 85, 68, 90, 58] # 리스트 구조

a = alist.pop() # 맨 뒤 항목을 추출 함(추출된 항목은 리스트에서 제거)

print(a)
alist
```

58

```
Out[17]: [78, 85, 68, 90]
```

리스트 항목: for ~ 반복문으로 접근

```
In [ ]: ## 리스트 항목: for ~ 반복문으로 접근
alist = [78, 85, 68, 90, 58] # 리스트 구조

len_item = len(alist)
for i in range(len_item):
    print(alist[i])
```

78
85
68
90
58

```
In [19]: ## 리스트 항목: for ~ 반복문으로 접근
alist = [78, 85, 68, 90, 58] # 리스트 구조

for x in alist:
    print(x)
```

78
85
68
90
58

리스트 항목: if ~ 선택문으로 항목 존재 확인

```
In [6]: ## 리스트 항목: if ~ 선택문으로 항목 존재 확인
alist = [78, 85, 68, 90, 58] # 리스트 구조
```

```
x = 90
y = (x in alist)
print(y)
```

True

```
In [9]: ## 리스트 항목: if ~ 선택문으로 항목 존재 확인
alist = [78, 85, 68, 90, 58] # 리스트 구조

x = 90
y = x not in alist # y = not(x in alist)
print(y)
```

False

```
In [22]: ## 리스트 항목: if ~ 선택문으로 항목 존재 확인
alist = [78, 85, 68, 90, 58] # 리스트 구조

x = 90
if x in alist:
    print(alist.index(x))
```

3

```
In [ ]:
```

키보드 입력으로 초기화: list()

> .split()는 문자열을 공백을 분리자로 하여 문자열 분리하여 리스트로 반환

```
In [36]: ## 키보드 입력으로 초기화: 숫자 초기화, 갯수 지정
alist = []

for _ in range(3):
    x = input("숫자 값 입력 후 Enter>")
    alist.append(int(x))

alist
```

```
Out[36]: [11, 22, 33]
```

```
In [10]: ## 키보드 입력으로 초기화: 숫자 초기화, Enter 입력 시까지
alist = []
```

```
x = ''
while len(x) > 0:
    x = input("숫자 값 입력 후 Enter>") #키보드 값은 문자형
    if len(x) > 0:
        alist.append(int(x))

alist
```

Out[10]: [11, 22, 33]

키보드 입력을 리스트로 저장

```
In [15]: ## 키보드로 입력된 문자 분리하기 > 분리된 항목은 복수개 이므로 리스트형으로 저장
alist = input("공백으로 분리된 값 입력 후 Enter>").split() #공백으로 분리

alist
```

Out[15]: ['11', '22', '33']

```
In [14]: ## 키보드로 입력된 문자 분리하기 > 분리된 항목을 integer형으로 저장
alist = list(int(x) for x in input("공백으로 분리된 숫자 값 입력 후 Enter>").split())

alist
```

Out[14]: [11, 22, 33]

In []:

리스트 슬라이싱

리스트 항목의 index 범위를 지정하여 또 다른 리스트로 분리(slice)해 내는 방법

리스트 슬라이스 추출

> 슬라이스(항목 범위)로 추출된 결과는 항상 리스트로 반환

```
In [64]: ## 리스트 슬라이싱
alist = [78, 85, 68, 90, 58] # 리스트 구조
```

```
alist[1:3] #index 1에서 시작해서 index 3일 때 종료로 분리
```

```
Out[64]: [85, 68]
```

```
In [65]: ## 리스트 슬라이싱
```

```
alist = [78, 85, 68, 90, 58] # 리스트 구조
```

```
alist[1:2] #index 1에서 시작해서 index 2일 때 종료로 분리
```

```
Out[65]: [85]
```

```
In [54]: ## 리스트 슬라이싱: 맨 앞 n개 분리
```

```
alist = [78, 85, 68, 90, 58] # 리스트 구조
```

```
alist[:3] #index 0에서 시작해서 3개 항목 분리
```

```
Out[54]: [78, 85, 68]
```

```
In [56]: ## 리스트 슬라이싱: 맨 뒤 n개 분리
```

```
alist = [78, 85, 68, 90, 58] # 리스트 구조
```

```
alist[-3:] #맨 뒤에서 시작해서 3개 항목 분리
```

```
Out[56]: [68, 90, 58]
```

```
In [57]: ## 리스트 슬라이싱: 모두 분리 > deep copy 효과
```

```
alist = [78, 85, 68, 90, 58] # 리스트 구조
```

```
blist = alist[:] #맨 뒤에서 시작해서 3개 항목 분리
```

```
blist
```

```
Out[57]: [78, 85, 68, 90, 58]
```

리스트 슬라이스: 건너뛰기

```
In [5]: ## 리스트 슬라이스: 건너뛰기
```

```
alist = [78, 85, 68, 90, 58, 100] # 리스트 구조
```

```
blist = alist[::2] #앞에서 2개 주기로 건너뛰며 추출
```

```
blist
```

```
Out[5]: [78, 68, 58]
```

```
In [8]: ## 리스트 슬라이스: 건너뛰기  
alist = [78, 85, 68, 90, 58, 100] # 리스트 구조  
  
blist = alist[:::-2] #뒤에서 2개 주기로 건너뛰며 추출  
blist
```

```
Out[8]: [100, 90, 85]
```

리스트 슬라이스로 항목 변경

```
In [58]: ## 리스트 슬라이스 변경  
alist = [78, 85, 68, 90, 58] # 리스트 구조  
  
alist[3:5] = [99, 100]  
alist
```

```
Out[58]: [78, 85, 68, 99, 100]
```

```
In [60]: ## 리스트 슬라이스 변경  
alist = [78, 85, 68, 90, 58] # 리스트 구조  
  
alist[3:5] = [99]  
alist
```

```
Out[60]: [78, 85, 68, 99]
```

리스트 슬라이스로 항목 제거

```
In [62]: ## 리스트 슬라이스 변경  
alist = [78, 85, 68, 90, 58] # 리스트 구조  
  
alist[3:5] = []  
alist
```

```
Out[62]: [78, 85, 68]
```

del() 함수로 항목을 제거

```
In [63]: ## 리스트 슬라이스 변경  
alist = [78, 85, 68, 90, 58] # 리스트 구조  
  
del(alist[3:5])  
alist
```

```
Out[63]: [78, 85, 68]
```

```
In [ ]:
```

리스트 조작 함수

.sort()

리스트 자체를 정렬(default는 ascending)

```
In [70]: ## 리스트 조작 함수: .sort()  
alist = [78, 85, 68, 90, 58] # 리스트 구조  
  
alist.sort() #리스트 자체를 정렬(default는 Ascending)  
alist
```

```
Out[70]: [58, 68, 78, 85, 90]
```

```
In [71]: ## 리스트 조작 함수: .sort()  
alist = [78, 85, 68, 90, 58] # 리스트 구조  
  
alist.sort(reverse=True) #리스트 자체를 정렬(default는 Ascending)  
alist
```

```
Out[71]: [90, 85, 78, 68, 58]
```

sorted()

리스트 정렬 결과만을 반환(default는 ascending), 자신은 원래 유지

```
In [72]: ## 리스트 조작 함수: .sort()  
alist = [78, 85, 68, 90, 58] # 리스트 구조  
  
blist = sorted(alist) #정렬 결과만을 반환  
alist
```

```
Out[72]: [78, 85, 68, 90, 58]
```

```
In [73]: blist
```

```
Out[73]: [58, 68, 78, 85, 90]
```

```
In [75]: ## 리스트 조작 함수: sorted()
alist = [78, 85, 68, 90, 58] # 리스트 구조

blist = sorted(alist, reverse=True) #정렬 결과만을 반환
blist
```

```
Out[75]: [90, 85, 78, 68, 58]
```

.reverse()

리스트 항목 순서를 뒤집는다.

```
In [77]: ## 리스트 조작 함수: .reverse()
alist = [78, 85, 68, 90, 58] # 리스트 구조

alist.reverse() #리스트 자체 항목 순서를 뒤집는다.
alist
```

```
Out[77]: [58, 90, 68, 85, 78]
```

.count()

리스트 항목에서 찾는 값의 갯수를 반환한다.

```
In [79]: ## 리스트 조작 함수: .count()
alist = [78, 85, 68, 90, 58] # 리스트 구조

alist.count(90) #값의 갯수를 반환한다.
```

```
Out[79]: 1
```

.extend()

리스트 항목에 다른 리스트를 이어 붙힌다.

```
In [85]: ## 리스트 조작 함수: .extend()
alist = [78, 85, 68, 90, 58] # 리스트 구조
blist = [30, 40]

alist = alist + blist #리스트끼리 연결
alist
```

```
Out[85]: [78, 85, 68, 90, 58, 30, 40]
```

```
In [86]: ## 리스트 조작 함수: .extend()
alist = [78, 85, 68, 90, 58] # 리스트 구조
blist = [30, 40]

alist.extend(blist) #리스트를 이어 붙힌다.
alist
```

```
Out[86]: [78, 85, 68, 90, 58, 30, 40]
```

```
In [ ]:
```

다차원 리스트 사용

일차원 리스트 초기화

```
In [112...]: ## 다차원 리스트 초기화:
arr = [0, 0, 0, 0, 0] # 일차원 리스트 초기화

arr
```

```
Out[112]: [0, 0, 0, 0, 0]
```

```
In [111...]: ## 다차원 리스트 초기화:
arr = [0]*5 # 일차원 리스트 초기화

arr
```

```
Out[111]: [0, 0, 0, 0, 0]
```

```
In [110]: ## 다차원 리스트 초기화: for 반복문 사용
arr = [0 for _ in range(5)] # 일차원 리스트 초기화

arr
Out[110]: [0, 0, 0, 0, 0]
```

```
In [109]: ## 다차원 리스트 초기화: for 반복문 사용
arr = [i for i in range(5)] # 일차원 리스트 초기화

arr
Out[109]: [0, 1, 2, 3, 4]
```

다차원 리스트 초기화

```
In [108]: ## 다차원 리스트 초기화
arr = [[0, 0, 0], [0, 0, 0], [0, 0, 0]] # 다차원 리스트 구조

arr
Out[108]: [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

```
In [16]: ## 다차원 리스트 초기화 >> 가급적 사용하지 말 것(이상현상 발생)
row = 3; col = 4
arr = [[0]*col]*row

arr
Out[16]: [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

```
In [17]: ## 다차원 리스트 초기화 >> 가급적 사용하지 말 것(이상현상 발생)
row = 3; col = 4
arr = [[0]*col]*row

arr[0][0] = 1 #의도하지 않은 결과가 발생
arr[1][1] = 2 #의도하지 않은 결과가 발생
arr
Out[17]: [[1, 2, 0, 0], [1, 2, 0, 0], [1, 2, 0, 0]]
```

```
In [18]: ## 다차원 리스트 초기화: for 반복문 사용
row = 3; col = 4
arr = [[0 for j in range(col)] for i in range(row)]

arr
```

```
Out[18]: [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

```
In [19]: ## 다차원 리스트 초기화: for 반복문 사용
row = 3; col = 4
arr = [] #1차원 리스트 생성

for i in range(row):
    temp = [] #열 생성
    for j in range(col):
        temp.append(0) #열 값 추가
    arr.append(temp) #1차원 리스트에 열 추가

arr
```

```
Out[19]: [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

[연습] 2차원 리스트 초기화

2차원 배열 리스트를 순서번호(index)로 초기화 하시오.

> 행(row)와 열(col)의 크기는 키보드로 입력받아서 사용

```
In [136...]: ## [실습] 2차원 리스트 초기화
row = int(input("행 개수는? "))
col = int(input("열 개수는? "))

arr = [[col*i+j for j in range(col)] for i in range(row)] # 0 대신 순번으로 초기화
arr
```

```
Out[136]: [[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]]
```

```
In [ ]:
```

다차원 리스트 접근

index를 사용한 접근

```
In [127...]: ## 다차원 리스트 접근: index를 사용
alist = [[0, 3, 2], [6, 1, 0], [8, 3, 6], [6, 9, 7]] # 다차원 리스트 구조

alist[0][2] = 1
alist
```

```
Out[127]: [[0, 3, 1], [6, 1, 0], [8, 3, 6], [6, 9, 7]]
```

```
In [128...]: ## 다차원 리스트 접근: index를 사용
alist = [[0, 3, 2], [6, 1, 0], [8, 3, 6], [6, 9, 7]] # 다차원 리스트 구조

alist[1] = [6, 1, 1]
alist
```

```
Out[128]: [[0, 3, 2], [6, 1, 1], [8, 3, 6], [6, 9, 7]]
```

index를 사용한 접근: for 반복문 사용

```
In [129...]: ## 다차원 리스트 접근: index를 사용
alist = [[0, 3, 2], [6, 1, 0], [8, 3, 6], [6, 9, 7]] # 다차원 리스트 구조

row = len(alist)
for i in range(row):
    col = len(alist[i])
    for j in range(col):
        print(alist[i][j], end=' ')
    print()
```

```
0 3 2
6 1 0
8 3 6
6 9 7
```

항목에 직접 접근: for 반복문 사용

```
In [124...]: ## 다차원 리스트 접근: for 반복문 사용
alist = [[0, 3, 2], [6, 1, 0], [8, 3, 6], [6, 9, 7]] # 다차원 리스트 구조
```

```
for x in alist:  
    print(x)
```

```
[0, 3, 2]  
[6, 1, 0]  
[8, 3, 6]  
[6, 9, 7]
```

In [131...]

```
## 다차원 리스트 접근: for 반복문 사용  
alist = [[0, 3, 2], [6, 1, 0], [8, 3, 6], [6, 9, 7]] # 다차원 리스트 구조  
  
for x in alist:  
    for y in x:  
        print(y, end=' ')  
    print()
```

```
0 3 2  
6 1 0  
8 3 6  
6 9 7
```

복합 자료형으로 구성된 다차원 리스트

In [125...]

```
## 다차원 리스트 접근: for 반복문 사용  
alist = [['Eng', 78], ['Kor', 85], ['Math', 68], ['Art', 90], ['Sci', 58]] # 다차원 리스트 구조  
  
for x in alist:  
    print(x)
```

```
['Eng', 78]  
['Kor', 85]  
['Math', 68]  
['Art', 90]  
['Sci', 58]
```

In [132...]

```
## 다차원 리스트 접근: for 반복문 사용  
alist = [['Eng', 78], ['Kor', 85], ['Math', 68], ['Art', 90], ['Sci', 58]] # 다차원 리스트 구조  
  
for x in alist:  
    for y in x:  
        print(y, end=' ')  
    print()
```

```
Eng 78  
Kor 85  
Math 68  
Art 90  
Sci 58
```

리스트 Unpacking

Unpacking은 하나의 변수에 포함된 복수 개의 값들을 분리해서 읽어내는 방법

```
In [20]: ## 다차원 리스트 Unpacking  
alist = [[0, 3, 2], [6, 1, 0], [8, 3, 6], [6, 9, 7]] # 다차원 리스트 구조  
  
for x in alist:  
    x1, x2, x3 = x #Unpacking  
    print(x1, x2, x3)
```

```
0 3 2  
6 1 0  
8 3 6  
6 9 7
```

```
In [21]: ## 다차원 리스트 Unpacking  
alist = [[0, 3, 2], [6, 1, 0], [8, 3, 6], [6, 9, 7]] # 다차원 리스트 구조  
  
for x1, x2, x3 in alist: #Unpacking  
    print(x1, x2, x3)
```

```
0 3 2  
6 1 0  
8 3 6  
6 9 7
```

```
In [22]: ## 다차원 리스트 접근: for 반복문 사용  
alist = [['Eng', 78], ['Kor', 85], ['Math', 68], ['Art', 90], ['Sci', 58]] # 다차원 리스트 구조  
  
for subject, scr in alist:  
    print(subject, scr)
```

```
Eng 78  
Kor 85  
Math 68  
Art 90  
Sci 58
```

```
In [23]: ## 다차원 리스트 접근: Unpacking
alist = [['Eng', 78], ['Kor', 85], ['Math', 68], ['Art', 90], ['Sci', 58]] # 다차원 리스트 구조

for x in alist:
    subject, scr = x
    print(subject, scr)
```

```
Eng 78
Kor 85
Math 68
Art 90
Sci 58
```

```
In [ ]:
```

[연습] 1차원 리스트에서 가장 작은 값 찾기

1차원 리스트 항목 중에 가장 작은 값을 찾아 그 항목의 index와 값을 출력 하시오.

```
In [24]: ## [연습] 1차원 리스트에서 가장 작은 값 찾기
alist = [4, 3, 2, 6, 1, 0, 8, 3, 6, 6, 9, 7]
print(alist)

minx = 0 #최소값 항목 index 초기화
for i in range(1, len(alist)):
    if alist[i] < alist[minx]:
        minx = i
print(">Min: [%d] %d" %(minx, alist[minx]))
```

```
[4, 3, 2, 6, 1, 0, 8, 3, 6, 6, 9, 7]
>Min: [5] 0
```

[연습] 1차원 리스트에서 가장 작은 값을 찾아 위치 바꾸기

가장 작은 값을 찾아 그 항목과 0번째 값을 교환 하시오.

```
In [26]: ## [연습] 1차원 리스트에서 가장 작은 값 찾기
alist = [4, 3, 2, 6, 1, 0, 8, 3, 6, 6, 9, 7]
print(alist)

minx = 0 #최소값 항목 index 초기화
for i in range(1, len(alist)):
```

```

if alist[i] < alist[minx]:
    minx = i

alist[0], alist[minx] = alist[minx], alist[0]      #항목 교환 > 작은 값이 앞으로
print(alist)

```

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

[실습] 1차원 리스트 오름차순 정렬: Selection Sort

Selection Sort는 가장 작은 값을 왼쪽부터 차례대로 배치(위치 확정)시키는 방법

- >가장 작은 값을 찾아 그 항목과 i번째 값과 교환 하는 것을 순차적으로 적용
- > i번째 순서 i는 0, 1, 2, ... 순으로 옮겨가며 그 이후 항목에서 가장 작은 값을 찾아 i번째와 교환
- > 1차원 리스트의 값들은 키보드로 입력 받아 사용
- >> 공백 입력 때까지 입력 가능토록 함

```

In [1]: ## [실습] 1차원 리스트 오름차순 정렬: Selection Sort
#키보드로 리스트 구성

#Selection Sort (오름차 순)
for pos in range(len(alist)): #기준 index
    minx = pos #최소값 항목 index를 교환 기준값 index로 시작
    for i in range(pos+1, len(alist)):
        if alist[i] < alist[minx]:
            minx = i
    alist[pos], alist[minx] = alist[minx], alist[pos]      #항목 교환 > 작은 값이 앞으로
print(alist)

```

[65, 67, 87, 89, 90, 54, 43, 45, 66, 77, 90]
[43, 45, 54, 65, 66, 67, 77, 87, 89, 90, 90]

[실습-과제] 1차원 리스트 오름차순 정렬: Bubble Sort

Bubble Sort는 인접한 항목과 값을 비교하여 작은 값이 우측으로 가도록 교환하는 방법

- >인접한 항목과 값을 비교하여 작은 값이 우측으로 가도록 교환하는 방법을 반복
- >한 차례를 반복하면, 가장 우측 끝 항목부터 가장 작은 값이 자리를 잡는다.
- >이러한 과정을 좌측으로 하나씩 이동하면서 정렬 완성

> 1차원 리스트의 값들은 키보드로 입력 받아 사용

>> 공백 입력 때까지 입력 가능토록 함

```
In [3]: ## [실습-과제] 1차원 리스트 오름차순 정렬: Bubble Sort (내림차순)
```

```
[45, 56, 67, 89, 90, 98, 87, 65, 45, 34, 56, 77]  
[98, 90, 89, 87, 77, 67, 65, 56, 56, 45, 45, 34]
```

```
In [ ]:
```

[연습] 2차원 리스트로 1차원 리스트 만들기

2차원 리스트의 각 항목을 차례대로 읽어서 1차원 리스트로 구성

```
In [31]: ## [연습] 2차원 리스트로 1차원 리스트 만들기  
alist = [[0, 3, 2], [6, 1, 0], [8, 3, 6], [6, 9, 7]]  
print(alist)  
  
arr = []  
for x in alist:  
    for y in x:  
        arr.append(y)  
print(arr)
```

```
[[0, 3, 2], [6, 1, 0], [8, 3, 6], [6, 9, 7]]  
[0, 3, 2, 6, 1, 0, 8, 3, 6, 6, 9, 7]
```

[연습] 1차원 리스트로 2차원 리스트 만들기

1차원 리스트의 각 항목을 열의 개수만큼 차례대로 읽어서 2차원 리스트로 구성

```
In [32]: ## [연습] 1차원 리스트로 2차원 리스트 만들기 [A]  
alist = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]  
print(alist)  
row = 4; col = 3  
  
arr = []  
for i in range(row):  
    temp = []  
    for j in range(col):  
        temp.append(alist[i*col+j]) # 행 리스트를 만들기
```

```
arr.append(temp) #행 리스트를 추가하여 2차원 리스트 만들기  
print(arr)
```

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

```
In [33]: ## [연습] 1차원 리스트로 2차원 리스트 만들기 [B]  
alist = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]  
print(alist)  
row = 4; col = 3  
  
arr = []  
for i in range(0, len(alist), col):  
    arr.append(alist[i:i+col]) #범위 리스트를 추가하기  
print(arr)
```

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

```
In [ ]:
```

[연습] 랜덤한 값으로 1차원 리스트 초기화 하기

1차원 배열 리스트를 중복 없는 랜덤한 값으로 초기화 하시오.

> 선택된 랜덤 수를 저장할 리스트의 크기는 3

> 선택되는 중복없는 랜덤 수는 0~9 중에 선택

`random.sample(arr, n)` 는 씨드 리스트 arr에서 중복없이 n개 선택

```
In [34]: ## [연습] 랜덤한 값으로 1차원 리스트 초기화 하기  
import random  
  
arr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] #씨드 리스트  
n = 3  
randarr = random.sample(arr, n) #씨드 리스트에서 중복없이 3개 선택  
print(randarr)
```

```
[5, 1, 4]
```

[연습] 랜덤한 값으로 2차원 리스트 초기화 하기

2차원 배열 리스트를 중복 없는 랜덤한 값으로 초기화 하시오.

- > 행(row)과 열(col)의 크기는 키보드로 입력 받아서 사용
- > 초기화 값은 0부터 시작하는 랜덤 수(중복 없이) 사용
- > `arr[row*col]`을 랜덤하게 만들어놓고, `arr[]`을 `randarr[][]`에 복사하는 방법 사용
- `random.shuffle(arr)` 는 리스트를 랜덤순으로 썩음(재정렬)

```
In [36]: ## [연습] 랜덤한 값으로 2차원 리스트 초기화 하기: 랜덤한 1차원 리스트 만들기
import random
row = int(input("행 개수는? "))
col = int(input("열 개수는? "))

## 1차원 씨드(seed) 리스트 생성
arr = [i for i in range(row*col)] #씨드 리스트 초기화(오름차순)

arr = random.sample(arr, row*col) #씨드 리스트 재정렬(랜덤순)
#random.shuffle(arr)           #씨드 리스트 재정렬(랜덤순)

print(arr)
```

[4, 1, 2, 0, 11, 9, 8, 7, 10, 5, 3, 6]

```
In [37]: ## [연습] 랜덤한 값으로 2차원 리스트 초기화 하기: 1차원 리스트로 2차원 리스트 만들기
randarr = []
for i in range(0, len(arr), col):
    randarr.append(arr[i:i+col])

print(randarr)
```

[[4, 1, 2, 0], [11, 9, 8, 7], [10, 5, 3, 6]]

In []:

[실습-과제] 숫자 야구 게임

랜덤한 3자리 숫자 맞추기 게임을 완성하시오.

- > 랜덤한 3자리 정수(1~999) 발생, 단 각 자리 수의 값은 같으면 안됨
- > 게이머는 추측하는 3자리 값을 키보드로 입력 (맞출 때까지)
- >> 각 위치(digit)에서 값이 같으면 Strike, 값은 존재하나 위치가 다르면 Ball로 처리
- >> 매회 ">>%d Strike, %d Ball"로 결과 제공

> 게임 종료 조건 : 3 Strike 또는 입력 값 00

(1) 3자리 난수 리스트 생성

```
In [39]: ## [실습-과제] 숫자 야구 게임: 3자리 난수 리스트 생성
import random

arr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]      #씨드 리스트 > 중복방지 난수 발생용
randarr = random.sample(arr, 3)            #난수 리스트 > 랜덤 샘플링

print(randarr)
```

[8, 7, 6]

(2) 3자리 숫자 입력 > 리스트 구성

예측되는 세자리 난수 값을 입력을 받아 리스트(inger)로 구성

- > 각 자리 수를 분리하여 리스트로 구성 > 예) 123 입력은 [1, 2, 3] 리스트로 구성
- > 0이 입력되면 게임 종료 처리

```
In [42]: ## [실습-과제] 숫자 야구 게임: integer 리스트 사용
import random

arr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]      #씨드 리스트 > 중복방지 난수 발생용
randarr = random.sample(arr, 3)            #난수 리스트 > 랜덤 샘플링
print(randarr)

guessarr = [0, 0, 0] #3자리 추측 리스트

while True : #게임 시작
    innum = int(input("=>정수(1~999; esc 00) 입력: "))
    if innum == 0:
        break
    #입력된 3자리 숫자에서 각 단위 자리수를 추출하는 스플릿 과정 필요

    print(guessarr)
```

[7, 3, 2]

[1, 2, 3]

(3) Strike, Ball 판정

난수 리스트와 입력 리스트 값을 비교하여 Strike 또는 Ball 판정

> 같은 위치에서 일치하면 Strike, 다른 위치에 존재하면 Ball 판정

> Strike, Ball 숫자 출력

만약 strike = 3이면 ">>OK!" 출력 후 종료

In [43]: ## [실습-과제] 숫자 야구 게임: integer 리스트 사용

```
import random

arr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]      #중복방지 난수 발생용 써드
randarr = random.sample(arr, 3)    #3자리 난수 리스트 랜덤 샘플링
print(randarr)

guessarr = [0, 0, 0] #3자리 추측 리스트
strike = 0; ball = 0
bcnt = 0    #시도 횟수

while True :
    bcnt += 1
    strike = 0; ball = 0
    innum = int(input(">>정수(1~999; esc 00) 입력: "))
    if innum == 0:
        break
    #입력된 3자리 숫자에서 각 단위 자리수를 추출하는 스플릿 과정 필요

    print("%d %d Strike %d Ball" %(bcnt, strike, ball))

    if strike == 3:
        print(">>OK!")
        break
```

[2, 3, 7]

```
[1] 0 Strike . 2 Ball  
[2] 0 Strike . 0 Ball  
[3] 0 Strike . 1 Ball  
[4] 0 Strike . 3 Ball  
[5] 3 Strike . 0 Ball  
>>OK!
```

In []:

숫자 야구 게임을 Character 리스트를 사용하여 해결

```
In [10]: ## [실습-과제] 숫자 야구 게임: Character 리스트 사용  
import random  
  
arr = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']      #중복방지 난수 발생용 써드  
randarr = random.sample(arr, 3) #3자리 난수 리스트 랜덤 샘플링  
print(randarr)  
  
guessarr = ['0', '0', '0'] #3자리 추측 리스트  
strike = 0; ball = 0  
bcnt = 0    #시도 횟수  
  
while True :  
    bcnt += 1  
    strike = 0; ball = 0  
    guessarr = input(">>정수(1~999; esc 00) 입력: ")  
    if guessarr[0] == guessarr[1]:  
        break  
    for i in range(3):  
        if guessarr[i] in randarr:  
            if guessarr[i] == randarr[i]:  
                strike += 1  
            else:  
                ball += 1  
    print("[%d] %d Strike %d Ball " %(bcnt, strike, ball))  
  
    if strike == 3:  
        print(">>OK! ")  
        break  
  
['9', '6', '3']
```

```
[1] 1 Strike 0 Ball  
[2] 0 Strike 1 Ball  
[3] 0 Strike 1 Ball  
[4] 1 Strike 2 Ball  
[5] 3 Strike 0 Ball  
>>OK!
```

In []:

[실습-과제] nxn 숫자 퍼즐 게임 만들기

다음과 같은 숫자 퍼즐 정렬하기 게임을 완성 하시오.

- > 퍼즐의 크기는 키보드로 입력 받음 > 정방형으로 사용
- > 잘 못된 키 값('a', 'w', 's', 'd' 이외)이 입력되면 "??잘못된 키 값입니다!"를 출력
- > 이동할 수 없는 방향 값이 입력되면 " !!잘못된 이동입니다!"를 출력
- > 퍼즐 정렬이 완성되면 "%d번만에 성공하였습니다."를 출력하고 종료
- > 현재 시도 횟 수를 하단에 표시
- >> 잘 못된 키 값이거나 이동할 수 없는 방향 값일 경우는 시도 횟수에서 제외

(1) 랜덤한 2차원 숫자 리스트 초기화

- > 퍼즐의 크기는 키보드로 입력 받음 > 정방형으로 사용

```
In [48]: ## [실습-과제] nxn 숫자 퍼즐 게임 만들기(1): 랜덤한 2차원 숫자 리스트 초기화  
import random  
  
##전역 약변수 정의  
col = int(input("열 개수는? "))  
row = col  
puzzle = [] #숫자 퍼즐 리스트  
  
# 퍼즐 리스트 난수 초기화  
arr = [i for i in range(row*col)]  
arr = random.sample(arr, row*col)  
  
for i in range(0, len(arr), col):
```

```
puzzle.append(arr[i:i+col])  
print(puzzle)
```

```
[[12, 14, 0, 4], [6, 1, 8, 13], [10, 3, 2, 9], [7, 5, 11, 15]]
```

(2) 2차원 숫자 리스트를 2차원으로 출력

```
In [47]: ## [실습-과제] nxn 숫자 퍼즐 게임 만들기(2): 2차원 숫자 리스트를 2차원으로 출력
```

```
for i in range(row) :  
    for j in range(col) :  
        if puzzle[i][j] == 0 :  
            print("%3s" % " ", end = "")  
        else :  
            print("%3s" % str(puzzle[i][j]), end = "")  
    print()
```

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

(3) 퍼즐 초기화와 출력을 함수로 처리

> 함수: rand_Puzzle(), prt_puzzle()

> 메인: puzzle 크기를 입력 받아, rand_Puzzle(), prt_puzzle() 함수 호출 처리

```
In [50]: ##### [실습-과제] nxn 숫자 퍼즐 게임 만들기(3): 퍼즐 초기화와 출력을 함수로 처리  
import random
```

```
##전역 약변수 정의  
puzzle = [] #숫자 퍼즐 리스트  
  
##함수 정의  
def rand_puzzle() :  
    #global puzzle, row, col  
    arr = [i for i in range(row*col)]  
    arr = random.sample(arr, row*col)  
    for i in range(0, len(arr), col):  
        puzzle.append(arr[i:i + col])  
  
def prt_puzzle() :
```

```

#global puzzle, row, col
print()
for i in range(row) :
    for j in range(col) :
        if puzzle[i][j] == 0 :
            print("%3s" % " ", end = " ")
        else :
            print("%3s" %str(puzzle[i][j]), end = " ")
print()

##===== 메인 시작 =====##
## 변수 초기화
col = int(input("열 개수는? "))
row = col

rand_puzzle()
prt_puzzle()
##===== 메인 끝 =====##

```

```

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

```

(4) 키보드 방향 값 처리 함수 생성

- > 방향키는 a(좌), w(상), s(하), d(우), 0(종료)를 사용
- > 방향키 값에 따라 해당 함수를 호출: , left(), right(), up(), down()
- > 방향키 처리 함수는 '>Left', '>Right', '>Up', '>Down'을 출력
- > 0입력 시 게임 종료 처리
- > 나머지 키는 경고 메시지 주고 재입력 처리

In [51]: ##### [실습-과제] nxn 숫자 퍼즐 게임 만들기(4): 키보드 방향 값 처리 함수 생성

```

## <이상 생략> ##
def left():
    print(">Left")

def right():
    print(">Right")

def up():

```

```

print(">Up")

def down():
    print(">Down")

##===== 메인 시작 =====##
## 변수 초기화
col = int(input("열 개수는? "))
row = col

rand_puzzle()
prt_puzzle()

while True: #게임 시작
    key = input("a(좌) w(상) s(하) d(우) 0(종료)> ")
    if key == 'a' :
        left()
    elif key == 'd' :
        right()
    elif key == 'w' :
        up()
    elif key == 's' :
        down()
    elif key == '0' :
        break
    else:
        print("??잘못된 키 값입니다!")
        continue
##===== 메인 끝 =====##

```

```

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

```

??잘못된 키 값입니다!

>Left

>Down

>Right

>Up

(5) 퍼즐에서 0의 위치 index 찾기

> 퍼즐에서 0의 위치 index를 찾아 행과 열 값을 출력

```
In [52]: ## [연습] nxn 숫자 퍼즐 게임 만들기(5): 퍼즐에서 0의 위치 index 찾기
r = 0; c = 0 #0의 row, col index

for i in range(row) :
    for j in range(col) :
        if puzzle[i][j] == 0 :
            r = i
            c = j
print(r, c)
```

2 0

(5) 퍼즐에서 0의 위치 index 찾기: 함수 호출로 처리

> 함수는 0의 index 값을 찾아 하나의 값으로 반환

> 메인에서 반환 받은 값으로부터 행과 열 index 값 도출하여 출력

```
In [53]: ## [실습-과제] nxn 숫자 퍼즐 게임 만들기(5): 퍼즐에서 0의 위치 index 찾기 > 함수 호출로 처리
def find_zero():
    for i in range(row) :
        for j in range(col) :
            if puzzle[i][j] == 0 :
                return row*i+j

##===== 메인 시작 =====##
## 변수 초기화
col = int(input("열 개수는? "))
row = col

rand_puzzle()
prt_puzzle()

while True: #게임 시작
    zero_seq = find_zero()
    r = zero_seq // row # zero 위치 행
    c = zero_seq % row # zero 위치 열
    print("r: %d, c: %d" %(r, c))

    key = input("a(좌) w(상) s(하) d(우) 0(종료)> ")
    if key == 'a' :
```

```

    left()
elif key == 'd' :
    right()
elif key == 'w' :
    up()
elif key == 's' :
    down()
elif key == '0' :
    break
else:
    print("??잘못된 키 값입니다!")
    continue
##### 메인 끝 #####

```

```

.. 2 13 . 6 14
12 . 7 . 4 . 5
.... 3 15 . 8
11 . 1 10 . 9
r: 2, c: 0

```

(6) 키보드로 방향키 값 받아 퍼즐 이동 처리

입력 받은 방향으로 퍼즐 이동 결과를 리스트에 반영하여 출력하시오.

> **puzzle**의 0 위치 값을 기준으로 인접 값과 상호 위치 교환

> 위치 교환 후 **puzzle** 리스트 출력

```
In [55]: ## [실습-과제] nxn 숫자 퍼즐 게임 만들기(6): 키보드로 방향키 값 받아 퍼즐 이동
import random

##전역 약변수 정의
puzzle = [] #숫자 퍼즐 리스트

##함수 정의
def rand_puzzle() :
    #global puzzle, row, col
    arr = [i for i in range(row*col)]
    arr = random.sample(arr, row*col)
    for i in range(0, len(arr), col):
        puzzle.append(arr[i:i + col])

def prt_puzzle() :
    #global puzzle, row, col
```

```

print()
for i in range(row) :
    for j in range(col) :
        if puzzle[i][j] == 0 :
            print("%3s" % " ", end = " ")
        else :
            print("%3s" %str(puzzle[i][j]), end = " ")
    print()

def left(r, c):
    puzzle[r][c], puzzle[r][c+1] = puzzle[r][c+1], puzzle[r][c] #0과 우측 값과 교환

def right(r, c):

def up(r, c):

def down(r, c):

def find_zero():
    for i in range(row) :
        for j in range(col) :
            if puzzle[i][j] == 0 :
                return row*i+j

##===== 메인 시작 =====##
## 변수 초기화
col = int(input("열 개수는? "))
row = col

rand_puzzle()
prt_puzzle()

while True: #게임 시작
    zero_seq = find_zero()
    r = zero_seq // row # zero 위치 행
    c = zero_seq % row # zero 위치 열
    print("r: %d, c: %d" %(r, c))

    key = input("a(좌) w(상) s(하) d(우) 0(종료)> ")
    if key == 'a' :
        left(r, c)

```

```

    elif key == 'd' :
        right(r, c)
    elif key == 'w' :
        up(r, c)
    elif key == 's' :
        down(r, c)
    elif key == '0' :
        break
    else:
        print("??잘못된 키 값입니다!")
        continue

    prt_puzzle()
##===== 메인 끝 =====##

```

.. 1 .. 7 .. 8 ..
.. 4 14 10 .. 2 ..
.. 9 .. 5 13 12 ..
11 .. 6 .. 3 15 ..
r: 0, c: 3

??잘못된 키 값입니다!
r: 0, c: 3

.. 1 .. 7 .. 8 ..
.. 4 14 10 .. 2 ..
.. 9 .. 5 13 12 ..
11 .. 6 .. 3 15 ..
r: 0, c: 2

.. 1 .. 7 .. 8 ..
.. 4 14 10 .. 2 ..
.. 9 .. 5 13 12 ..
11 .. 6 .. 3 15 ..
r: 0, c: 1

.. 1 .. 7 .. 8 ..
.. 4 14 10 .. 2 ..
.. 9 .. 5 13 12 ..
11 .. 6 .. 3 15 ..
r: 0, c: 2

```
.. 1 7 10 8  
.. 4 14 ... 2  
.. 9 5 13 12  
11 6 3 15  
r: 1, c: 2
```

```
.. 1 7 ... 8  
.. 4 14 10 2  
.. 9 5 13 12  
11 6 3 15  
r: 0, c: 2
```

(6) 키보드로 방향키 값 받아 퍼즐 이동 처리: 이동 불가 처리

이동이 불가능한 방향이 입력되면 **puzzle** 값 위치 교환 처리를 하지 않음.

```
In [57]: ## [실습-과제] nxn 숫자 퍼즐 게임 만들기(6): 키보드로 방향키 값 받아 퍼즐 이동  
import random
```

```
##전역 변수 정의
```

```
puzzle = [] #숫자 퍼즐 리스트
```

```
##함수 정의
```

```
def rand_puzzle() :
```

```
    #global puzzle, row, col  
    arr = [i for i in range(row*col)]  
    arr = random.sample(arr, row*col)  
    for i in range(0, len(arr), col):  
        puzzle.append(arr[i:i + col])
```

```
def prt_puzzle() :
```

```
    #global puzzle, row, col  
    print()  
    for i in range(row) :  
        for j in range(col) :  
            if puzzle[i][j] == 0 :  
                print("%3s" % " ", end = "")  
            else :  
                print("%3s" % str(puzzle[i][j]), end = "")  
        print()
```

```
def left(r, c):
```

```
    if c+1 < col:  
        puzzle[r][c], puzzle[r][c+1] = puzzle[r][c+1], puzzle[r][c]      #0과 우측 값과 교환
```

```
def right(r, c):

def up(r, c):

def down(r, c):

def find_zero():
    for i in range(row):
        for j in range(col):
            if puzzle[i][j] == 0:
                return row*i+j

##===== 메인 시작 =====##
## 변수 초기화
col = int(input("열 개수는? "))
row = col

rand_puzzle()
prt_puzzle()

while True: #게임 시작
    zero_seq = find_zero()
    r = zero_seq // row # zero 위치 행
    c = zero_seq % row # zero 위치 열
    #print("r: %d, c: %d" %(r, c))

    key = input("a(좌) w(상) s(하) d(우) 0(종료)> ")
    if key == 'a':
        left(r, c)
    elif key == 'd':
        right(r, c)
    elif key == 'w':
        up(r, c)
    elif key == 's':
        down(r, c)
    elif key == '0':
        break
```

```

    else:
        print("??잘못된 키 값입니다!")
        continue

    prt_puzzle()
##===== 메인 끝 =====##

```

```

.... 7 14 5
11 4 8 2
6 12 15 1
10 9 13 3
r: 0, c: 0

```

```

.... 7 14 5
11 4 8 2
6 12 15 1
10 9 13 3
r: 0, c: 0

```

```

.... 7 14 5
11 4 8 2
6 12 15 1
10 9 13 3
r: 0, c: 0

```

(7) 키보드로 방향키 값 받아 퍼즐 이동 처리: 화면 클리어 사용

입력 받은 방향으로 퍼즐 이동 결과를 출력하기 전에 화면을 클리어 처리

```

In [59]: ## [실습-과제] nxn 숫자 퍼즐 게임 만들기(7): 키보드로 방향키 값 받아 퍼즐 이동
import random
from IPython.display import clear_output

##전역 약변수 정의
puzzle = [] #숫자 퍼즐 리스트

##함수 정의
def rand_puzzle():
    #global puzzle, row, col
    arr = [i for i in range(row*col)]
    arr = random.sample(arr, row*col)
    for i in range(0, len(arr), col):
        puzzle.append(arr[i:i + col])

```

```

def prt_puzzle():
    #global puzzle, row, col
    print()
    for i in range(row):
        for j in range(col):
            if puzzle[i][j] == 0:
                print("%3s" % " ", end = " ")
            else:
                print("%3s" % str(puzzle[i][j]), end = " ")
    print()

def left(r, c):
    if c+1 < col:
        puzzle[r][c], puzzle[r][c+1] = puzzle[r][c+1], puzzle[r][c]      #0과 우측 값과 교환

def right(r, c):
    if c-1 >= 0:
        puzzle[r][c], puzzle[r][c-1] = puzzle[r][c-1], puzzle[r][c]      #0과 좌측 값과 교환

def up(r, c):
    if r+1 < row:
        puzzle[r][c], puzzle[r+1][c] = puzzle[r+1][c], puzzle[r][c]      #0과 하단 값과 교환

def down(r, c):
    if r-1 >= 0:
        puzzle[r][c], puzzle[r-1][c] = puzzle[r-1][c], puzzle[r][c]      #0과 상단 값과 교환

def find_zero():
    for i in range(row):
        for j in range(col):
            if puzzle[i][j] == 0:
                return row*i+j

##===== 메인 시작 =====##
## 변수 초기화
col = int(input("열 개수는? "))
row = col

rand_puzzle()
prt_puzzle()

while True: #게임 시작
    zero_seq = find_zero()

```

```

r = zero_seq // row # zero 위치 행
c = zero_seq % row # zero 위치 열
#print("r: %d, c: %d" %(r, c))

key = input("a(좌) w(상) s(하) d(우) 0(종료)> ")
if key == 'a' :
    left(r, c)
elif key == 'd' :
    right(r, c)
elif key == 'w' :
    up(r, c)
elif key == 's' :
    down(r, c)
elif key == '0' :
    break
else:
    print("??잘못된 키 값입니다!")
    continue

clear_output(wait=True)
prt_puzzle()
##===== 메인 끝 =====##

```

```

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

```

(8) 퍼즐 완성 여부 판단

2차원 배열 리스트 정렬이 완성되었는지 판단을 함수 호출로 해결하시오.

- > 전체 중에 0을 제외한 숫자가 모두 일치하는지 판단을 함수 호출로 해결 □ `check_complete()`
- > 정렬이 완성되었으면 1을 반환, 미완성이면 0을 반환
- > 키 입력 할 때마다 시도 횟수를 출력
- >> 잘 못된 키 값이거나 이동할 수 없는 방향 값일 경우는 시도 횟수에서 제외

```

In [61]: ## [실습-과제] nxn 숫자 퍼즐 게임 만들기(8): 퍼즐 완성 여부 판단
import random
from IPython.display import clear_output

##전역 약변수 정의

```

```

puzzle = [] #숫자 퍼즐 리스트

##함수 정의
def rand_puzzle() :
    #global puzzle, row, col
    arr = [i for i in range(row*col)]
    arr = random.sample(arr, row*col)
    for i in range(0, len(arr), col):
        puzzle.append(arr[i:i + col])

def prt_puzzle() :
    #global puzzle, row, col
    print()
    for i in range(row) :
        for j in range(col) :
            if puzzle[i][j] == 0 :
                print("%3s" % " ", end = " ")
            else :
                print("%3s" %str(puzzle[i][j]), end = " ")
    print()

def left(r, c):
    if c+1 < col:
        puzzle[r][c], puzzle[r][c+1] = puzzle[r][c+1], puzzle[r][c]      #0과 우측 값과 교환

def right(r, c):
    if c-1 >= 0:
        puzzle[r][c], puzzle[r][c-1] = puzzle[r][c-1], puzzle[r][c]      #0과 좌측 값과 교환

def up(r, c):
    if r+1 < row:
        puzzle[r][c], puzzle[r+1][c] = puzzle[r+1][c], puzzle[r][c]      #0과 하단 값과 교환

def down(r, c):
    if r-1 >= 0:
        puzzle[r][c], puzzle[r-1][c] = puzzle[r-1][c], puzzle[r][c]      #0과 상단 값과 교환

def find_zero():
    for i in range(row) :
        for j in range(col) :
            if puzzle[i][j] == 0 :
                return row*i+j

def check_complete():

```

```

global puzzle, row, col
chk_cnt = 0
for i in range(row) :
    for j in range(col) :
        if puzzle[i][j] == (row*i + j + 1) :
            chk_cnt += 1
if chk_cnt >= (row*col-1) :
    return 1    #complete
else :
    return 0

##===== 메인 시작 =====##
## 변수 초기화
col = int(input("열 개수는? "))
row = col

rand_puzzle()
prt_puzzle()

while True: #게임 시작
    zero_seq = find_zero()
    r = zero_seq // row    # zero 위치 행
    c = zero_seq % row    # zero 위치 열
    #print("r: %d, c: %d" %(r, c))

    key = input("a(좌) w(상) s(하) d(우) 0(종료)> ")
    if key == 'a' :
        left(r, c)
    elif key == 'd' :
        right(r, c)
    elif key == 'w' :
        up(r, c)
    elif key == 's' :
        down(r, c)
    elif key == '0' :
        break
    else:
        print("??잘못된 키 값입니다!")
        continue

    clear_output(wait=True)
    prt_puzzle()

    if check_complete():

```

```

        print("=>성공하였습니다.")
    else:
        print("=>아직 미완성")
##===== 메인 끝 =====##

```

```

4 3 10 1
14 15 13 8
5 12 2 6
9 11 7
=>아직 미완성

```

(9) 시도 횟수 출력

시도 횟수 카운트해서 출력하시오.

> 키 입력 할 때마다 시도 횟수를 출력

>> 잘 못된 키 값이거나 이동할 수 없는 방향 값일 경우는 시도 횟수에서 제외

```
In [63]: ## [실습-과제] nxn 숫자 퍼즐 게임 만들기(9): 시도 횟수 출력
import random
from IPython.display import clear_output

##전역 약변수 정의
puzzle = [] #숫자 퍼즐 리스트

##함수 정의
def rand_puzzle():
    #global puzzle, row, col
    arr = [i for i in range(row*col)]
    arr = random.sample(arr, row*col)
    for i in range(0, len(arr), col):
        puzzle.append(arr[i:i + col])

def prt_puzzle():
    #global puzzle, row, col
    print()
    for i in range(row):
        for j in range(col):
            if puzzle[i][j] == 0:
                print("%3s" % " ", end = " ")
            else:
                print("%3s" % str(puzzle[i][j]), end = " ")
    print()

# 퍼즐 초기화
row = 4
col = 4
rand_puzzle()
prt_puzzle()
```

```

def left(r, c):
    if c+1 < col:
        puzzle[r][c], puzzle[r][c+1] = puzzle[r][c+1], puzzle[r][c]      #0과 우측 값과 교환
        return 1
    else:
        return 0

def right(r, c):
    if c-1 >= 0:
        puzzle[r][c], puzzle[r][c-1] = puzzle[r][c-1], puzzle[r][c]      #0과 좌측 값과 교환
        return 1
    else:
        return 0

def up(r, c):
    if r+1 < row:
        puzzle[r][c], puzzle[r+1][c] = puzzle[r+1][c], puzzle[r][c]      #0과 하단 값과 교환
        return 1
    else:
        return 0

def down(r, c):
    if r-1 >= 0:
        puzzle[r][c], puzzle[r-1][c] = puzzle[r-1][c], puzzle[r][c]      #0과 상단 값과 교환
        return 1
    else:
        return 0

def find_zero():
    for i in range(row) :
        for j in range(col) :
            if puzzle[i][j] == 0 :
                return row*i+j

def check_complete():
    global puzzle, row, col
    chk_cnt = 0
    for i in range(row):
        for j in range(col):
            if puzzle[i][j] == (row*i + j + 1) :
                chk_cnt += 1
    if chk_cnt >= (row*col-1):
        return 1    #complete
    else:

```

```

    return 0

##### 메인 시작 #####
## 변수 초기화
col = int(input("열 개수는? "))
row = col
cnt_try = 0 #시도 횟수

rand_puzzle()
prt_puzzle()

while True: #게임 시작
    zero_seq = find_zero()
    r = zero_seq // row # zero 위치 행
    c = zero_seq % row # zero 위치 열
    #print("r: %d, c: %d" %(r, c))

    key = input("[%d] a(좌) w(상) s(하) d(우) 0(종료)> " %cnt_try)
    cnt_try += 1

    if key == 'a' :
        left(r, c)
    elif key == 'd' :
        right(r, c)
    elif key == 'w' :
        up(r, c)
    elif key == 's' :
        down(r, c)
    elif key == '0' :
        break
    else:
        print("??잘못된 키 값입니다!")
        continue

    clear_output(wait=True)
    prt_puzzle()

    if check_complete():
        print("Wn[%d]번만에 성공하였습니다." %cnt_try)
##### 메인 끝 #####

```

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

In []: