

Section01 이 장에서 만들 프로그램

■ [프로그램 2] 딕셔너리를 활용한 음식 궁합 출력

- 딕셔너리를 활용해 음식 궁합을 출력하는 프로그램

The screenshot shows a Python Shell window with the following text:

```
Python Shell
File Edit Shell Debug Options Window Help
=====
RESTART: C:\CookPython\Code07-10.py
=====
['떡볶이', '짜장면', '라면', '피자', '맥주', '치킨', '삼겹살'] 중 좋아하는 음식은?치킨
<치킨> 궁합 음식은 <치킨무>입니다.
['떡볶이', '짜장면', '라면', '피자', '맥주', '치킨', '삼겹살'] 중 좋아하는 음식은?라면
<라면> 궁합 음식은 <김치>입니다.
['떡볶이', '짜장면', '라면', '피자', '맥주', '치킨', '삼겹살'] 중 좋아하는 음식은?짬뽕
그런 음식이 없습니다. 확인해 보세요.
['떡볶이', '짜장면', '라면', '피자', '맥주', '치킨', '삼겹살'] 중 좋아하는 음식은?꼴
>>>
```

A red callout box labeled "사용자가 입력한 값" points to the user input "치킨", "라면", "짬뽕", and "꼴".

Section02 리스트의 기본

■ 리스트의 개념

- 딕셔너리를 활용해 음식 궁합을 출력하는 프로그램

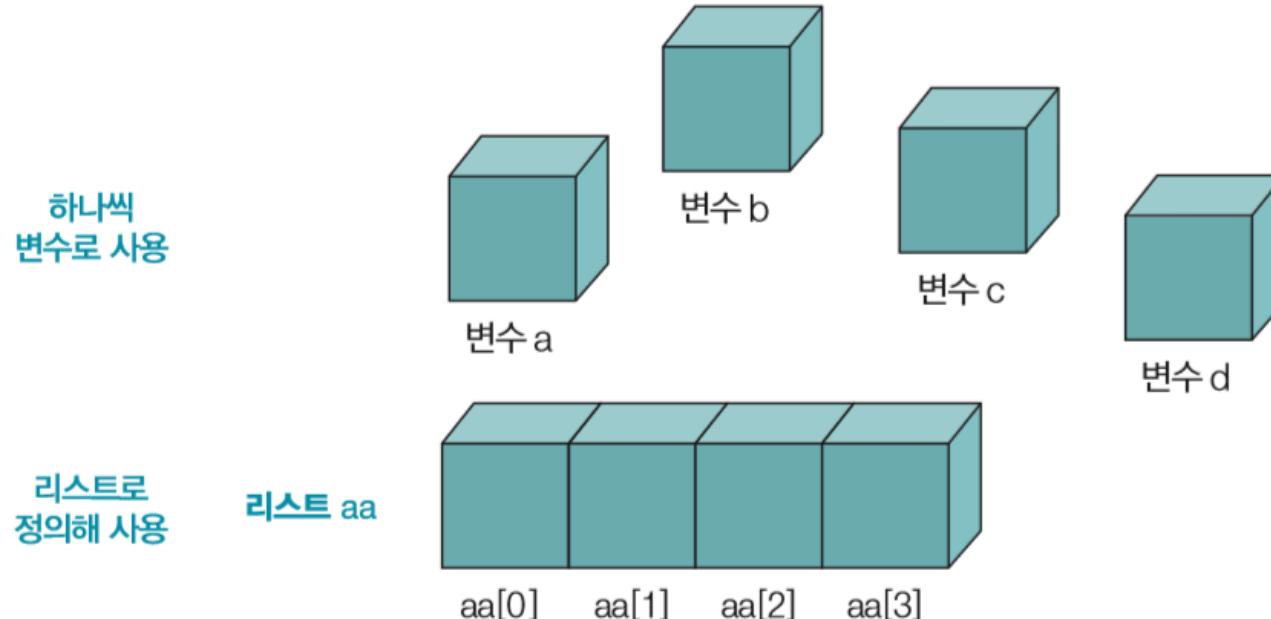


그림 7-1 리스트의 개념

Tip • C/C++나 자바 같은 프로그래밍 언어에는 리스트가 없음, 리스트와 비슷한 개념인 배열(Array)을 사용. 리스트는 정수, 문자열, 실수 등 서로 다른 데이터형도 하나로 묶을 수 있지만, 배열은 동일한 데이터형만 묶을 수 있다. 정수 배열은 정수로만 묶어서 사용

Section02 리스트의 기본

■ 리스트의 필요성

- a, b, c, d라는 정수형 변수 선언 후 이 변수에 값을 입력받고 합계 출력하는 프로그램

Code07-01.py

```
1 a, b, c, d = 0, 0, 0, 0
2 hap = 0
3
4 a = int(input("1번째 숫자 : "))
5 b = int(input("2번째 숫자 : "))
6 c = int(input("3번째 숫자 : "))
7 d = int(input("4번째 숫자 : "))
8
9 hap = a + b + c + d
10
11 print("합계 ==> %d" % hap)
```

출력 결과

1번째 숫자 : 10

2번째 숫자 : 20

3번째 숫자 : 30

4번째 숫자 : 40

합계 ==> 100

Section02 리스트의 기본

■ 리스트를 생성하는 방법

리스트명 = [값1, 값2, 값3, …]

aa = [10, 20, 30, 40]

① 각 변수 사용

a, b, c, d = 10, 20, 30, 40

a 사용

b 사용

c 사용

d 사용

② 리스트 사용

aa = [10, 20, 30, 40]

aa[0] 사용

aa[1] 사용

aa[2] 사용

aa[3] 사용

Section02 리스트의 기본

- Code07-01.py를 리스트를 사용하는 프로그램으로 수정

Code07-02.py

```
1 aa = [0, 0, 0, 0]          1행 : aa=[0, 0, 0, 0]으로 항목이 4개 있는 리스트 생성
2 hap = 0                    4행 : a 대신 aa[0]을 사용
3
4 aa[0] = int(input("1번째 숫자 : ")) 5~7행 : 리스트 aa를 사용
5 aa[1] = int(input("2번째 숫자 : "))
6 aa[2] = int(input("3번째 숫자 : "))
7 aa[3] = int(input("4번째 숫자 : "))
8
9 hap = aa[0] + aa[1] + aa[2] + aa[3] 9행 : 각 변수 대신 aa[0]+aa[1]+aa[2]+aa[3]으로 수정
10
11 print("합계 ==> %d" % hap)
```

출력 결과

1번째 숫자 : 10

2번째 숫자 : 20

3번째 숫자 : 30

4번째 숫자 : 40

합계 ==> 100

출력 결과는 리스트를 사용하기 전과 동일

숫자 100개를 더하려면 aa=[0, 1, 0, ..., 99]를 생성한 후
aa[0]+aa[1]+aa[2] +...+aa[99]로 작성

Section02 리스트의 기본

- 리스트의 일반적인 사용
 - 빈 리스트의 생성과 항목 추가

```
aa = []
aa.append(0)
aa.append(0)
aa.append(0)
aa.append(0)
print(aa)
```

출력 결과

[0, 0, 0, 0]

```
aa = []
for i in range(0, 100) :
    aa.append(0)
len(aa)
```

출력 결과

100

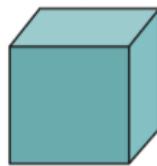
Section02 리스트의 기본

- 리스트의 첨자가 순서대로 변할 수 있도록 반복문과 함께 활용

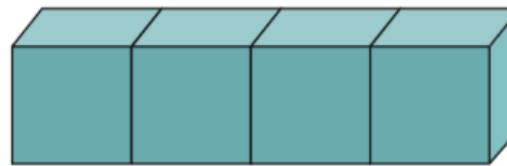
for (4번 반복)

{

값 입력



aa[0]부터 aa[3]까지 4번 반복



aa[0] aa[1] aa[2] aa[3]

}

aa[i]

i값이 0부터 3까지 변함

그림 7-2 for 문으로 리스트값 입력

Section02 리스트의 기본

Code07-03.py

```
1 aa = []           1행 : 빈 리스트 생성
2 for i in range(0, 4) : 2~3행 : 4번을 반복해 항목이 4개인 리스트로 만듦
3     aa.append(0)    6행 : i가 0에서 3까지 4번 반복
4 hap = 0
5
6 for i in range(0, 4) :
7     aa[i] = int(input(str(i + 1) + "번째 숫자 : "))
8
9 hap = aa[0] + aa[1] + aa[2] + aa[3] 7행 : input() 함수는 첨자 i가 0부터 시작하므로 i+1로 출력.
10
11 print("합계 ==> %d" % hap)          str() 함수가 숫자를 문자로 변환한 후 '번째 숫자 :'와 합쳐지므로 결국 '1번째 숫자 : ', '2번째 숫자 : ' 등으로 출력. 7행의 첨자 i가 0에서 3까지 4번 변경되므로 aa[0], aa[1], aa[2], aa[3] 등 변수 4개에 값을 차례대로 입력해 [그림 7-2]와 같이 작동
9행 : 변수 4개를 더함
```

출력 결과

```
1번째 숫자 : 10
2번째 숫자 : 20
3번째 숫자 : 30
4번째 숫자 : 40
합계 ==> 100
```

Section02 리스트의 기본

- 9행의 변경

```
for i in range(0, 4) :  
    hap = hap + aa[i]
```

SELF STUDY 7-1

값을 4개가 아닌 10개를 입력받아 합계를 출력하도록 Code07-03.py를 수정해 보자. 또 합계를 구하는 마지막 for 문 대신 while 문을 사용해 보자.

Section02 리스트의 기본

■ 리스트의 생성과 초기화

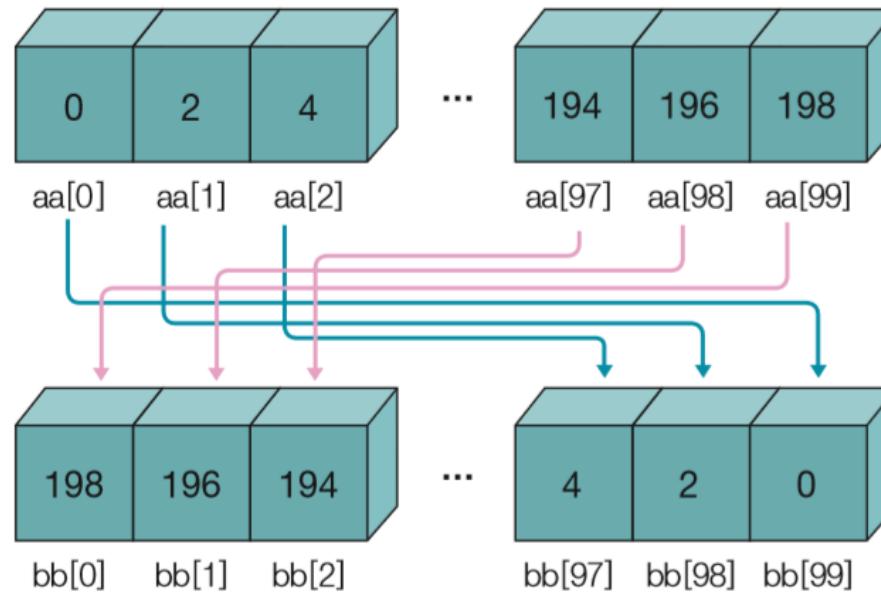
▪ 리스트 생성 코드

```
❶ aa = []
❷ bb = [10, 20, 30]
❸ cc = ['파이썬', '공부는', '꿀잼']
❹ dd = [10, 20, '파이썬']
```

- ❶ 빈 리스트 생성
- ❷ 정수로만 구성된 리스트 생성
- ❸ 문자열로만 구성된 리스트 생성
- ❹ 다양한 데이터형을 섞어 리스트 생성

▪ 리스트 100개인 aa 0, 2, 4, 8, ... (2의 배수로 초기화 후 리스트 bb에 역순으로 넣는 과정

❶ 리스트 aa를
짝수로 초기화



❷ 리스트 bb에
역순으로 대입

그림 7-3 리스트의 초기화 및 역순 대입

Section02 리스트의 기본

- [그림 7-3] 코드로 구현

Code07-04.py

```
1 aa = []          1~2행 : 빈 리스트 aa, bb 생성
2 bb = []          3행 : value는 0, 2, 4, ...로 증가시킬 값
3 value = 0        5행 : 100번을 반복
4
5 for i in range(0, 100) : 6~7행 : 리스트 aa에 value를 추가한 후 2씩 증가
6     aa.append(value) 9행 : 0~99로 100번 반복
7     value += 2        10행 : i가 0일 때 99-i는 99가 됨. i가 1일 때는 98, i가 2일 때는 97
8
9 for i in range(0, 100) :  처럼 계속 변해 마지막으로 i가 99일 때는 0이 됨.
10    bb.append(aa[99 - i])  리스트 bb에는 aa[99], aa[98], aa[97], ..., aa[0]의 값이 추가되므로
11
12 print("bb[0]에는 %d이, bb[99]에는 %d이 입력됩니다." % (bb[0], bb[99]))  결국 리스트 aa값이 리스트 bb에 역순으로 입력
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

출력 결과

bb[0]에는 198이, bb[99]에는 0이 입력됩니다.

Section02 리스트의 기본

SELF STUDY 7-2

Code07-04.py를 리스트 aa에 3의 배수를 200개 입력하도록 수정해 보자. 그리고 리스트 bb에는 리스트 aa의 역순으로 입력해 보자. 최종적으로 bb[0]과 bb[199]의 값을 출력하면 다음과 같다.

출력 결과

bb[0]에는 597이, bb[199]에는 0이 입력됩니다.

Section02 리스트의 기본

■ 리스트 값에 접근하는 다양한 방법

■ 음수값으로 접근

```
aa = [10, 20, 30, 40]
print("aa[-1]은 %d, aa[-2]는 %d" % (aa[-1], aa[-2]))
```

출력 결과

aa[-1]은 40, aa[-2]는 30 aa[-4]까지 접근되므로 aa[-5]는 오류 발생

■ 리스트에 접근할 때 콜론(:)을 사용해 범위를 지정

```
aa = [10, 20, 30, 40]
aa[0:3]
aa[2:4]
```

출력 결과

[10, 20, 30]
[30, 40]

Section02 리스트의 기본

- 콜론의 앞이나 뒤 숫자 생략

```
aa = [10, 20, 30, 40]
```

```
aa[2:]
```

```
aa[:2]
```

출력 결과

```
[30, 40]
```

```
[10, 20]
```

- 리스트끼리 덧셈, 곱셈 연산

```
aa = [10, 20, 30]
```

```
bb = [40, 50, 60]
```

```
aa + bb
```

```
aa * 3
```

출력 결과

```
[10, 20, 30, 40, 50, 60]
```

```
[10, 20, 30, 10, 20, 30, 10, 20, 30]
```

Section02 리스트의 기본

- 리스트의 항목 건너뛰며 추출

```
aa = [10, 20, 30, 40, 50, 60, 70]  
aa[::2]  
aa[::-2]  
aa[::-1]
```

출력 결과

```
[10, 30, 50, 70]  
[70, 50, 30, 10]  
[70, 60, 50, 40, 30, 20, 10]
```

Section02 리스트의 기본

■ 리스트 값의 변경

- 두 번째에 위치한 값을 1개 변경하는 방법

```
aa = [10, 20, 30]
```

```
aa[1] = 200
```

```
aa
```

출력 결과

```
[10, 200, 30]
```

- 두 번째 값인 20을 200과 201이라는 값 2개로 변경하는 방법

```
aa = [10, 20, 30]
```

```
aa[1:2] = [200, 201]
```

```
aa
```

출력 결과

```
[10, 200, 201, 30]
```

Section02 리스트의 기본

- aa[1:2] 대신 그냥 aa[1] 사용

```
aa = [10, 20, 30]
aa[1] = [200, 201]
aa
```

출력 결과

```
[10, [200, 201], 30]
```

- 두 번째인 aa[1]의 항목 삭제

```
aa = [10, 20, 30]
del(aa[1])
aa
```

출력 결과

```
[10, 30]
```

Section02 리스트의 기본

- 두 번째인 aa[1]에 서 네 번째인 aa[3]까지 삭제

```
aa = [10, 20, 30, 40, 50]
```

```
aa[1:4] = []
```

```
aa
```

출력 결과

```
[10, 50]
```

- 리스트 자체를 삭제하는 방법

① aa = [10, 20, 30]; aa = []; aa

② aa = [10, 20, 30]; aa = None; aa

③ aa = [10, 20, 30]; del(aa); aa

출력 결과

```
[]
```

아무것도 안 나옴

오류 발생

Section02 리스트의 기본

■ 리스트 조작 함수

표 7-1 리스트 조작 함수

| 함수 | 설명 | 사용법 |
|-----------|--|---------------------|
| append() | 리스트 맨 뒤에 항목을 추가한다. | 리스트명.append(값) |
| pop() | 리스트 맨 뒤의 항목을 빼낸다(리스트에서 해당 항목이 삭제된다). | 리스트명.pop() |
| sort() | 리스트의 항목을 정렬한다. | 리스트명.sort() |
| reverse() | 리스트 항목의 순서를 역순으로 만든다. | 리스트명.reverse() |
| index() | 지정한 값을 찾아 해당 위치를 반환한다. | 리스트명.index(찾을값) |
| insert() | 지정된 위치에 값을 삽입한다. | 리스트명.insert(위치, 값) |
| remove() | 리스트에서 지정한 값을 삭제한다. 단 지정한 값이 여러 개면 첫 번째 값만 지운다. | 리스트명.remove(지울값) |
| extend() | 리스트 뒤에 리스트를 추가한다. 리스트의 더하기(+) 연산과 기능이 동일하다. | 리스트명.extend(추가할리스트) |
| count() | 리스트에서 해당 값의 개수를 센다. | 리스트명.count(찾을값) |
| clear() | 리스트의 내용을 모두 지운다. | 리스트명.clear() |
| del() | 리스트에서 해당 위치의 항목을 삭제한다. | del(리스트명[위치]) |
| len() | 리스트에 포함된 전체 항목의 개수를 센다. | len(리스트명) |
| copy() | 리스트의 내용을 새로운 리스트에 복사한다. | 새리스트=리스트명.copy() |
| sorted() | 리스트의 항목을 정렬해서 새로운 리스트에 대입한다. | 새리스트=sorted(리스트) |

Section02 리스트의 기본

■ 예

Code07-05.py

```
1 myList = [30, 10, 20]
2 print("현재 리스트 : %s" % myList)
3
4 myList.append(40)
5 print("append(40) 후의 리스트 : %s" % myList)
6
7 print("pop()으로 추출한 값 : %s" % myList.pop())
8 print("pop() 후의 리스트 : %s" % myList)
9
10 myList.sort()
11 print("sort() 후의 리스트 : %s" % myList)
12
13 myList.reverse()
14 print("reverse() 후의 리스트 : %s" % myList)
15
16 print("20값의 위치 : %d" % myList.index(20)) 18행 : myList.insert(2, 222)에서 2는 myList[2]의 위치
17
18 myList.insert(2, 222)를 의미, 리스트는 0번부터 시작 하므로 세 번째 위치
19 print("insert(2, 222) 후의 리스트 : %s" % myList)
20
```

10행 : '리스트.sort()'는 리스트 자체 정렬 'sorted(리스트)'는 리스트는 그대로 두고 정렬된 결과만 반환

18행 : myList.insert(2, 222)에서 2는 myList[2]의 위치
를 의미, 리스트는 0번부터 시작 하므로 세 번째 위치
가 뒤로 밀리고 그 자리에 222가 삽입

Section02 리스트의 기본

```
21 myList.remove(222)
22 print("remove(222) 후의 리스트 : %s" % myList)
23
24 myList.extend([77, 88, 77])
25 print("extend([77, 88, 77]) 후의 리스트 : %s" % myList)
26
27 print("77값의 개수 : %d" % myList.count(77))
```

출력 결과

현재 리스트 : [30, 10, 20]

append(40) 후의 리스트 : [30, 10, 20, 40]

pop()으로 추출한 값 : 40

pop() 후의 리스트 : [30, 10, 20]

sort() 후의 리스트 : [10, 20, 30]

reverse() 후의 리스트 : [30, 20, 10]

20값의 위치 : 1

insert(2, 222) 후의 리스트 : [30, 20, 222, 10]

remove(222) 후의 리스트 : [30, 20, 10]

extend([77, 88, 77]) 후의 리스트 : [30, 20, 10, 77, 88, 77]

77값의 개수 : 2

Section02 리스트의 기본

- 기존 리스트는 변경하지 않고 정렬된 새로운 리스트 생성

```
myList = [30, 10, 20]
newList = sorted(myList)
print("sorted() 후의 myList : %s" % myList)
print("sorted() 후의 newList : %s" % newList)
```

출력 결과

```
sorted() 후의 myList : [30, 10, 20]
sorted() 후의 newList : [10, 20, 30]
```

Section03 2차원 리스트

■ 2차원 리스트의 개념

- 1차원 리스트를 여러 개 연결한 것, 첨자를 2개 사용

aa = [10, 20, 30]

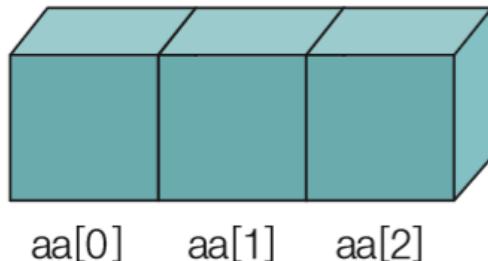


그림 7-4 1차원 리스트의 개념

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



그림 7-5 2차원 리스트의 개념

Section03 2차원 리스트

- 예 : 중첩 for 문을 사용, 3행 4열짜리 리스트 생성 후 항목 1~12를 입력하고 출력

Code07-06.py

```
1 list1 = []          1~2행 : 1차원 리스트로 사용할 list1과 2차원 리스트로 사용할 list2 준비
2 list2 = []          3행 : value는 리스트에 입력할 1~12의 값으로 사용할 변수
3 value = 1           4~9행 : 리스트의 행 단위 만들기 위해 3회 반복
4 for i in range(0, 3) : 5~7행 : 4회 반복해 항목 4개인 1차원 리스트 생성하는데, 처음에는 [1,
5     for k in range(0, 4) : 2, 3, 4]의 리스트를 만듬
6         list1.append(value)
7         value += 1
8     list2.append(list1)      8행 : 2차원 리스트에 추가
9     list1 = []              9행 : 1차원 리스트를 다시 비움
10
11    for i in range(0, 3) : 11~14행 : 2차원 리스트 출력
12        for k in range(0, 4) : 13행 : '리스트명[행][열]' 방식으로 각 항목 출력
13            print("%3d" % list2[i][k], end = " ")
14        print("")             14행 : 행 하나 출력 후 한 행 띄워서 출력 위해 print() 문 사용
```

출력 결과

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

Section03 2차원 리스트

SELF STUDY 7-3

4행 5열의 2차원 리스트를 만들고, 0부터 3의 배수를 입력하고 출력하도록 Code07-06.py를 수정해 보자. 출력 결과는 다음과 같다.

출력 결과

```
0  3  6  9  12  
15 18 21 24 27  
30 33 36 39 42  
45 48 51 54 57
```

Section03 2차원 리스트

- 불규칙한 크기의 2차원 리스트

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

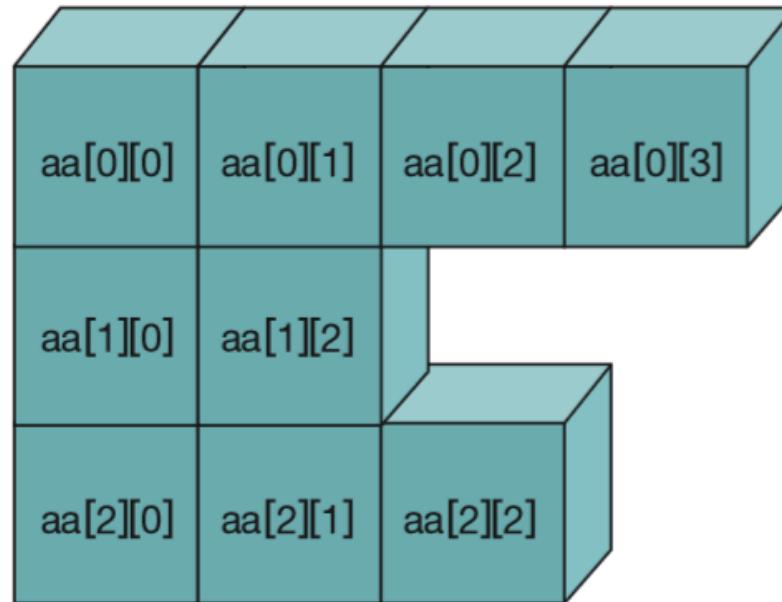


그림 7-6 불규칙한 크기의 2차원 리스트

Section03 2차원 리스트

■ [프로그램 1]의 완성

- 리스트를 이용 터틀 그래픽 응용 프로그램 만들기
 - 거북이 한마리의 1차원 리스트

[거북이, X위치, Y위치, 거북이크기, 거북이색상(R), 거북이색상(G), 거북이색상(B)]

- 2차원 리스트

[[거북이1, X, Y, 크기, R, G, B], [거북이2, X, Y, 크기, R, G, B], [거북이3, X, Y, 크기, R, G, B]…]

Section03 2차원 리스트

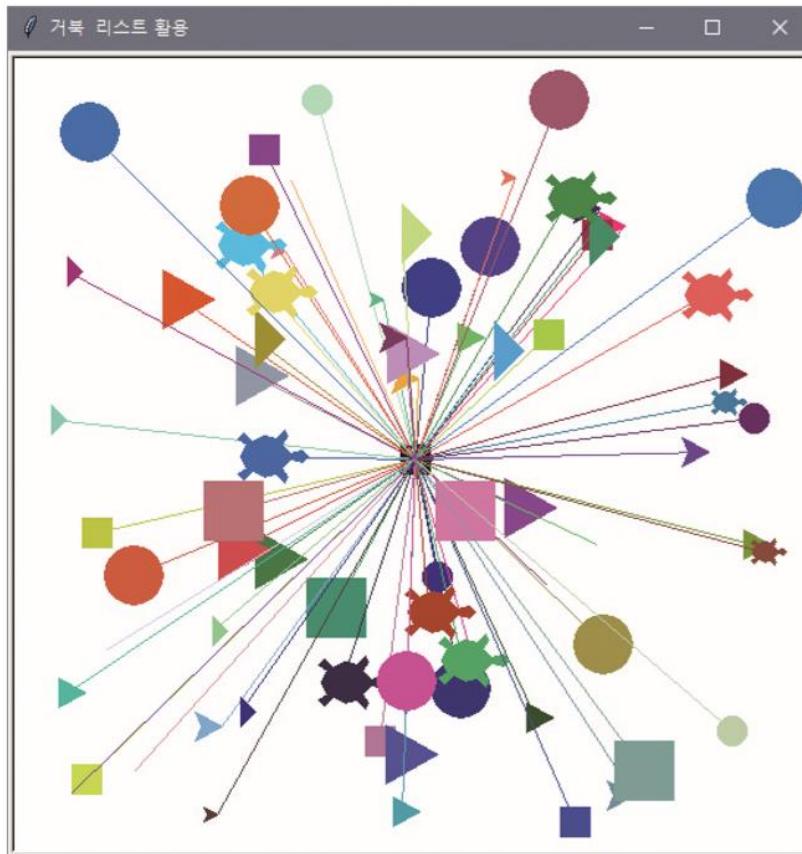
Code07-07.py

```
1 import turtle  
2 import random  
3  
4 ## 전역 변수 선언 부분 ##  
5 myTurtle, tX, tY, tColor, tSize, tShape = [None] * 6      5~8행 : 전역 변수 준비  
6 shapeList = []  
7 playerTurtles = []      # 거북이 2차원 리스트  
8 swidth, sheight = 500, 500    7행 : playerTurtles가 거북이 100마리를 저장할 2차원 리스트 준비  
9  
10 ## 메인 코드 부분 ##  
11 if __name__ == "__main__":  
12     turtle.title('거북 리스트 활용')  
13     turtle.setup(width = swidth + 50, height = sheight + 50)  
14     turtle.screensize(swidth, sheight)  
15
```

Section03 2차원 리스트

```
16 shapeList = turtle.getshapes()           19행 : myTurtle은 거북이 개체 생성
17 for i in range(0, 100) :
18     random.shuffle(shapeList)
19     myTurtle = turtle.Turtle(shapeList[0])
20     tX = random.randrange(-swidth / 2, swidth / 2)
21     tY = random.randrange(-sheight / 2, sheight / 2)
22     r = random.random(); g = random.random(); b = random.random()
23     tSize = random.randrange(1, 3)
24     playerTurtles.append([myTurtle, tX, tY, tSize, r, g, b])
25
26 for tList in playerTurtles :           16행 : shapeList 추출하면 ['arrow', 'blank', 'circle', 'classic',
27     myTurtle = tList[0]                   'square', 'triangle', 'turtle'] 등 추출
28     myTurtle.color((tList[4], tList[5], tList[6]))
29     myTurtle.pencolor((tList[4], tList[5], tList[6]))
30     myTurtle.turtlesize(tList[3])
31     myTurtle.goto(tList[1], tList[2])
32 turtle.done()                         17~24행 : 거북이 100마리를 1차원 리스트로 생성
                                         24행 : playerTurtles에 1차원 리스트 추가
                                         26~31행 : 거북이를 playerTurtles에서 하나씩 추출해 tList에 넣고 반복하여 거북이 100마리를 꺼내서 화면에 선을 그림
```

Section03 2차원 리스트



Tip • `random.choice` 18~19행은 다음과 같이 `random.choice(리스트)` 이용하도록 변경 가능
`random.choice(리스트)`는 리스트 중에서 임의값을 하나 추출해 반환

```
tmpShape = random.choice(shapeList)  
myTurtle = turtle.Turtle(tmpShape)
```

Section04 튜플

■ 튜플의 생성

- 리스트는 대괄호 []로 생성, 튜플은 소괄호 ()로 생성
- 튜플은 값을 수정할 수 없으며, 읽기만 가능해 읽기 전용 자료를 저장할 때 사용

```
tt1 = (10, 20, 30); tt1  
tt2 = 10, 20, 30; tt2
```

출력 결과

```
(10, 20, 30)  
(10, 20, 30)
```

- 튜플은 소괄호 ()를 생략 가능, 항목이 하나인 튜플은 tt5와 tt6처럼 뒤에 쉼표(,) 붙임

```
tt3 = (10); tt3  
tt4 = 10; tt4  
tt5 = (10,); tt5  
tt6 = 10,; tt6
```

출력 결과

```
10  
10  
(10,)  
(10,)
```