컴퓨팅 사고력을 키우는 SW 교육

파이썬

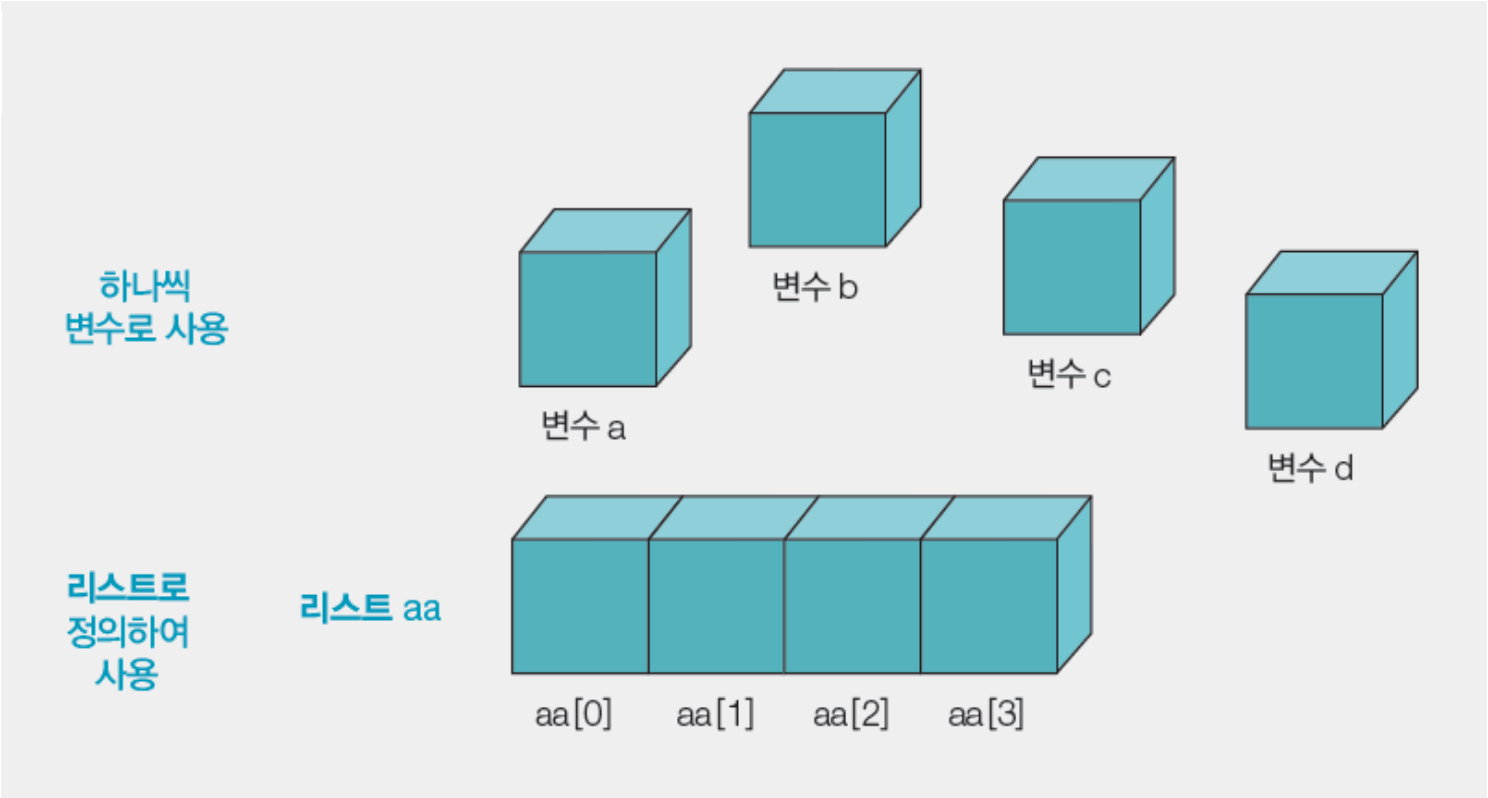
Chapter 04
복합자료형

Section 01 List

리스트 (1)

■ 리스트의 이해

그림 7-4
리스트의 개념



- 리스트는 박스(변수)를 한 줄로 붙인 뒤에 박스 전체의 이름(aa)을 지정.
- 각각은 aa[0], aa[1], aa[2], aa[3]과 같이 번호(첨자)를 붙여서 사용.

리스트 (2)

■ 리스트를 사용하는 이유

- 4개의 정수형 변수를 선언한 다음 변수에 값을 입력 받고 합계를 출력하는 프로그램

소스코드 7-1
(파일명 : 07-01.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  ← 사용자가 입력한 값
2번째 숫자 : 20  ← 사용자가 입력한 값
3번째 숫자 : 30  ← 사용자가 입력한 값
4번째 숫자 : 40  ← 사용자가 입력한 값
 합계==>100
```

리스트 (3)

- 리스트 생성 방법

```
리스트이름=[값1, 값2, 값3, …]
```

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

- 리스트를 사용하지 않는다면 각각의 변수를 a, b, c, d와 같이 선언(아래 ①)
- 하지만 리스트를 사용하면 첨자를 넣어 aa[0], aa[1], aa[2], aa[3]과 같이 선언(아래 ②)
- 이때 항목이 4개인 리스트를 생성한다면 첨자는 1~4가 아닌 0~3을 사용함.

① 각 변수 사용	② 리스트 사용
a, b, c, d = 10, 20, 30, 40	aa = [10, 20, 30, 40]
a 사용	aa[0] 사용
b 사용	aa[1] 사용
c 사용	aa[2] 사용
d 사용	aa[3] 사용

리스트 (4)

- 리스트를 사용해서 [소스코드 7-1] 수정

소스코드 7-2
(파일명 : 07-02.py)

```
1  aa=[0, 0, 0, 0]
2  hap=0
3
4  aa[0]=int(input("1번째 숫자 : "))
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]
10
11 print(" 합계==>%d " % hap)
```

출력 결과

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

리스트 (5)

- **리스트의 일반적인 사용법**
  - 빈 리스트와 리스트의 추가
    - 비어있는 리스트를 만들고 '리스트이름.append(값)' 함수로 리스트에 하나씩 추가

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

출력 결과

```
[0, 0, 0, 0]
```

리스트 (6)

- 100개의 리스트를 만들 경우 append()와 함께 for문을 활용

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

len(aa)
```

출력 결과

```
100
```

- for문으로 100번(0부터 99까지)을 반복해서 리스트이름.append(0)로 100개 크기의 리스트를 만듦
- len 함수로 리스트의 개수를 확인

리스트 (7)

- For문 활용

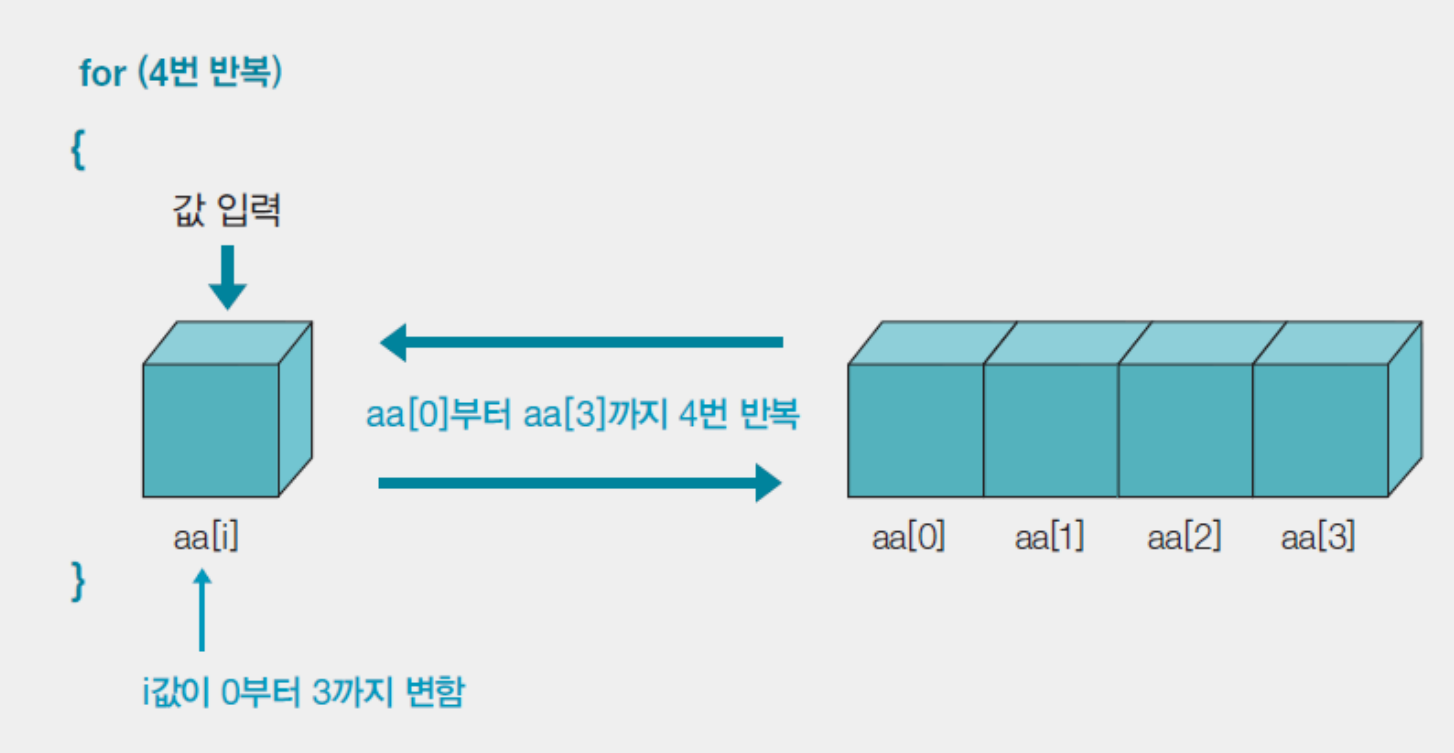


그림 7-5
for문을 활용하여
리스트 값 입력하기

리스트 (8)

소스코드 7-3
(파일명 : 07-03.py)

```
1  aa=[]
2  for i in range(0, 4) :
3      aa.append(0)
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]
10
11 print(" 합계==>%d " % hap)
```

출력 결과

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

리스트 (9)

- 만약 리스트가 100개라면 hap =aa[0] +aa[1] +...aa[99]로 일일이 코딩 하지 안고 9행을 for문으로 변경함

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

- 리스트의 생성과 초기화

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

리스트 (10)

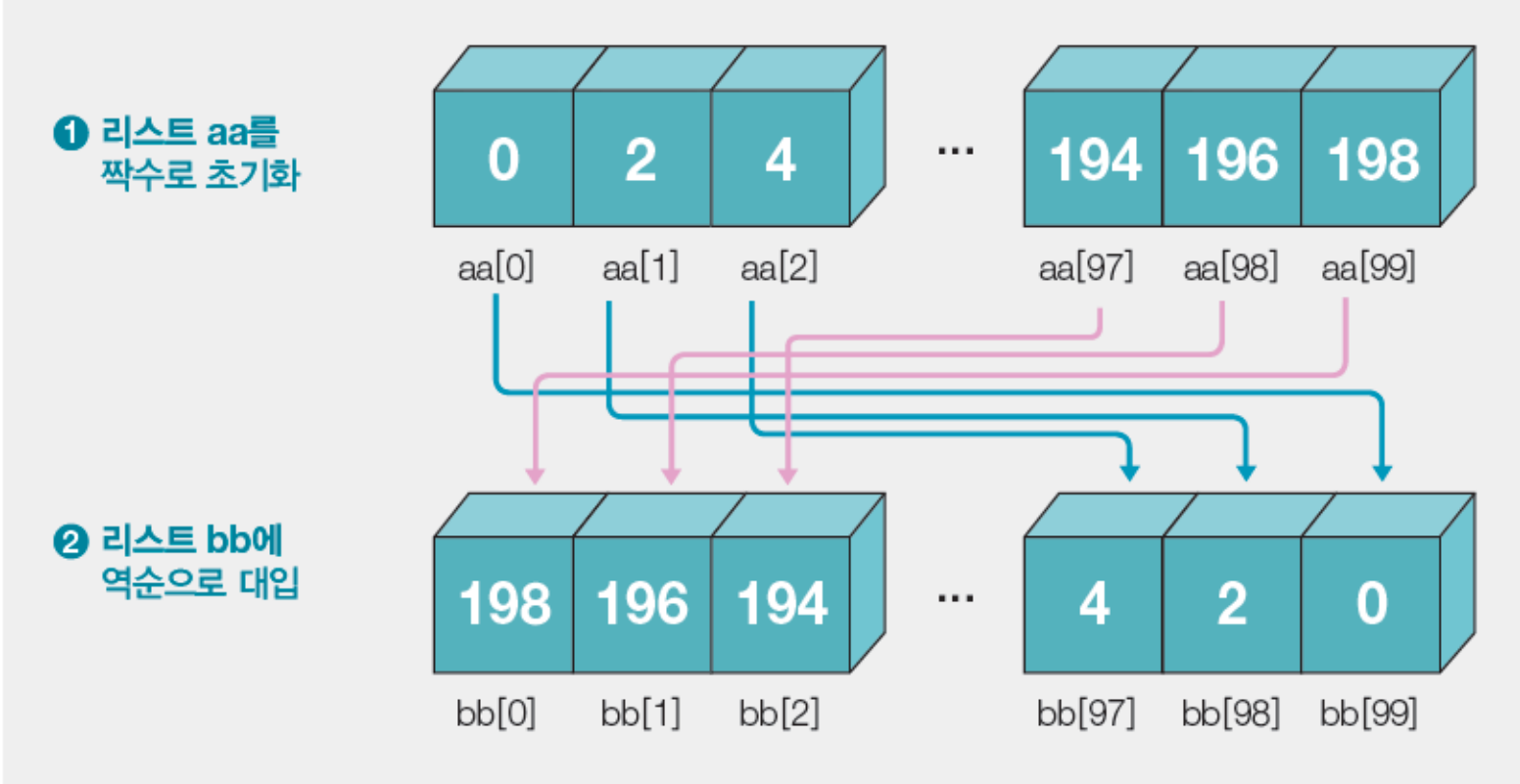


그림 7-6
리스트의 초기화 및
역순 대입

리스트 (11)

소스코드 7-4
(파일명 : 07-04.py)

```
1  aa = []
2  bb = []
3  value = 0
4
5  for i in range(0, 100) :
6      aa.append(value)
7      value += 2
8
9  for i in range(0, 100) :
10     bb.append(aa[99 - i])
11
12 print(" bb[0]은 %d, bb[99]는 %d 입력됨 " % (bb[0], bb[99] ))
```

출력 결과

```
bb[0]은 198, bb[99]는 0 입력됨
```

리스트 (12)

- **여러 개의 리스트 값을 사용하기**

```
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
```

- 리스트이름[시작:끝+1]'로 지정하면 리스트의 모든 값이 나옴

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

출력 결과

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

리스트 (13)

- 콜론의 앞이나 뒤 숫자의 생략도 가능

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

출력 결과

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

- 리스트끼리 더하기, 곱하기 연산도 가능

```
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]
```

리스트 (14)

리스트 값을 변경하기

- 두 번째 위치한 한 개의 값을 변경하는 방법

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

출력 결과

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

- 두 번째 값인 20을 200과 201 두 개의 값으로 변경

```
aa=[10, 20, 30]
aa[1:2]=[200, 201]
aa
```

출력 결과

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

리스트 (15)

- aa[1:2] 대신 aa[1]을 사용 → 리스트 안에 또 리스트로 추가됨. 결과가 틀리지는 않지만 이렇게는 많이 사용하지 않음

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

출력 결과

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

- del() 함수를 사용하여 aa[1] 항목을 삭제하는 방법

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

출력 결과

```
[10, 30]
```

리스트 (16)

- 여러 개의 항목을 삭제하려면 'aa[시작:끝+1]=[ ]' 문장으로 설정

```
aa=[10, 20, 30, 40, 50]
aa[1:4]=[]
aa
```

출력 결과

```
[10, 50]
```

리스트 (17)

- **리스트 조작 함수**

표 7-1
리스트 조작 함수

함수	설명	사용법
append( )	리스트 제일 뒤에 항목을 추가한다.	리스트이름.append(값)
pop( )	리스트 제일 뒤의 항목을 빼내고, 빼낸 항목은 삭제한다.	리스트이름.pop( )
sort( )	리스트의 항목을 정렬한다.	리스트이름.sort( )
reverse( )	리스트 항목의 순서를 역순으로 만든다.	리스트이름.reverse( )
index( )	지정한 값을 찾아서 그 위치를 반환한다.	리스트이름.index(찾을 값)
insert( )	지정된 위치에 값을 삽입한다.	리스트이름.insert(위치, 값)
remove( )	리스트에서 지정한 값을 제거한다. 단 지정한 값이 여러 개일 경우 첫 번째 값만 지운다.	리스트이름.remove(지울 값)
extend( )	리스트 뒤에 리스트를 추가한다. 리스트의 더하기(+) 연산과 동일한 기능을 한다.	리스트이름.extend(리스트)
count( )	리스트에서 찾을 값의 개수를 센다.	리스트이름.count(찾을 값)
del( )	리스트에서 해당 위치의 항목을 삭제한다.	del(리스트이름[위치])
len( )	리스트에 포함된 전체 항목의 개수를 센다.	len(리스트이름)

리스트 (18)

소스코드 7-5
(파일명 : 07-05.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("20 값의 위치 : %d" % myList.index(20))
17
18 myList.insert(2, 222)
19 print("insert(2, 222) 후의 리스트 : %s" % myList)
```

리스트 (19)

```
20
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("77 값의 개수 : %d" % myList.count(77))
```

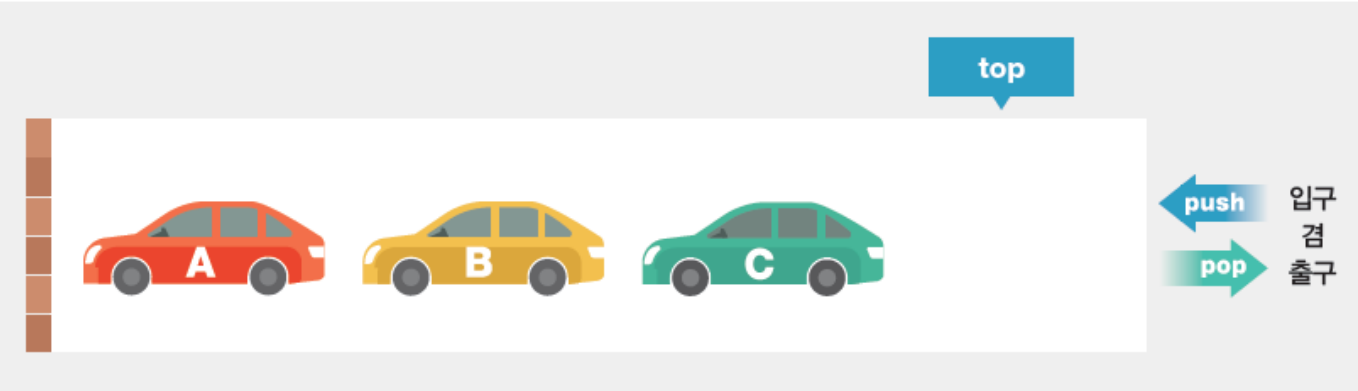
출력 결과

```
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]
20 값의 위치 : 1
insert(2, 222) 후의 리스트 : [30, 20, 222, 10]
remove(222) 후의 리스트 : [30, 20, 10]
extend([77 , 88]) 후의 리스트 : [30, 20, 10, 77, 88, 77]
77 값의 개수 : 2
```

리스트 (20)

- **한쪽이 막힌 주차장 프로그램 완성**
  - 스택은 한쪽 끝이 막힌 자료구조로, 가장 먼저 들어간 것이 가장 나중에 나옴. 그림에서 자동차가 들어간 순서는 A → B → C지만, 나오는 순서는 C → B → A

그림 7-7
스택의 구조와 작동



  - LIFO(Last In First Out) 구조라고도 함. 비어있는 위치를 top이라 칭함. 만약 자동차 C가 빠져나가면 top은 현재 스택 안에 들어있는 데이터 중 가장 마지막 데이터의 다음 위치를 가리킴. 이때 데이터를 넣는 것을 푸시(Push), 빼는 것을 팝(Pop)이라고 함

리스트 (21)

- 자동차 5대가 들어갈 수 있고, 한쪽이 막힌 주차장

```
parking = [ "", "", "", "", "" ]
top = 0
```

그림 7-8
비어있는 주차장

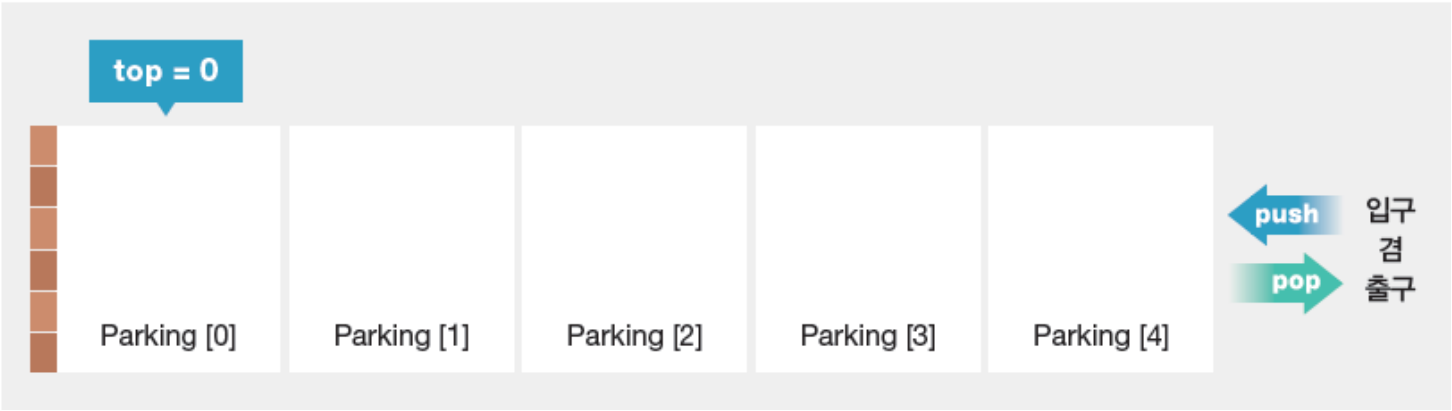


그림 7-9
주차장에 자동차 한 대 넣기

리스트 (22)

그림 7-10
주차장에 자동차 두 대 추가로 넣기

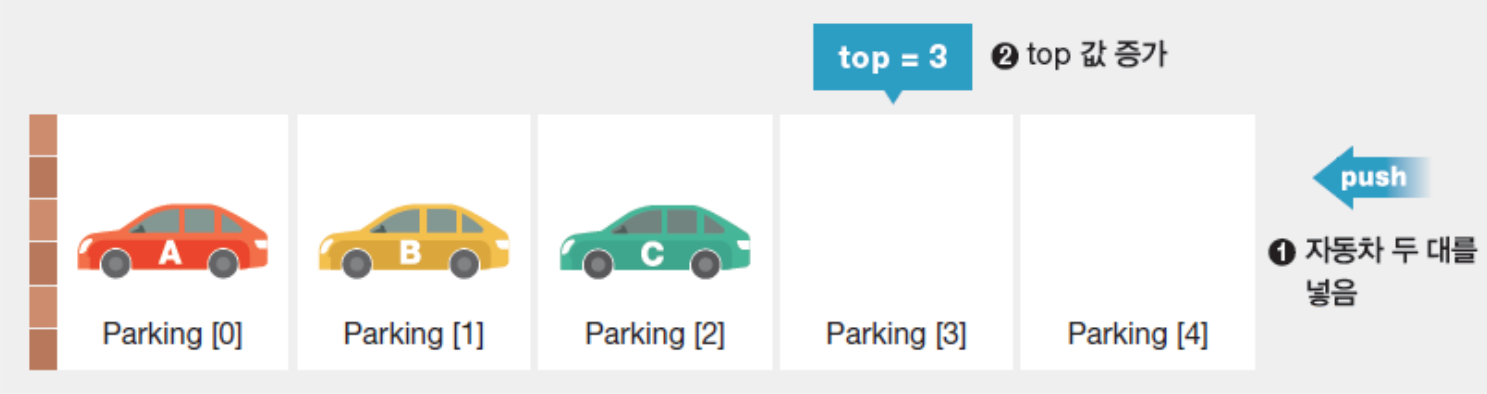


그림 7-11
터널에 자동차 한 대 빼기

리스트 (23)

소스코드 7-6
(파일명 : 07-06.py)

```
## 변수 선언 부분
parking=[]
top, carName, outCar=0, "A", ""
select=9

## 메인(main) 코드 부분
while (select!=3) :
    select=int(input("<1> 자동차 넣기 <2> 자동차 빼기 <3> 끝 : "))

    if (select==1) :
        if( top>=5 ) :
            print(" 주차장이 꽉 차서 못들어감")
        else :
            parking.append(carName)
            print("%s 자동차 들어감. 주차장 상태==>%s " % (parking[top], parking))
            top+=1
            carName=chr(ord(carName)+1)
```

리스트 (24)

```
18      elif (select==2) :
19          if( top<=0 ) :
20              print(" 빠져나갈 자동차가 없음")
21          else :
22              outCar=parking.pop()
23              print(" %s 자동차 나감. 주차장 상태==>%s " % (outCar, parking))
24              top-=1
25              carName=chr(ord(carName)-1)
26      elif (select==3) :
27          break;
28      else :
29          print(" 잘못 입력했네요. 다시 입력하세요.")
30
31  print(" 현재 주차장에 %d 대가 있음" % top)
32  print(" 프로그램을 종료합니다.")
```

리스트 (25)

① <1> 자동차 넣기

10행~17행.

11행에서는 top이 5 이상이라면(= 주차장이 꽉 차 있다면) 12행에서 더 이상 자동차가 들어가지 못한다는 메시지를 출력하고 전체 if문을 종료.

top이 5 미만이라면 14행~17행을 수행. 14행에서 주차장에 자동차(처음에는 'A')를 넣고, 15행에서 들어간 자동차 이름과 현재 주차장의 상태를 출력. 16행에서 top을 1 증가시킴

② <2> 자동차 빼기

18행~25행.

19행에서는 자동차를 빼내야 하는데, top이 0 이하라면(= 차량이 한 대도 없다면) 20행에서 더 이상 빼낼 자동차가 없다는 메시지를 출력하고 전체 if문을 종료.

top이 0보다 크다면 22행~25행을 수행. 22행에서 pop() 함수로 주차장(리스트)의 마지막에 있는 자동차를 outCar로 뺌. 23행에서 빼낸 자동차 이름과 현재 주차장의 상태를 출력. 그리고 24행에서 top을 1 감소시킴

리스트 (26)

③ <3> 끝

26행~27행이 처리되어 while문 종료

31행, 32행에서 현재 주차장에 남아있는 자동차의 대수(top)와 프로그램을 종료한다는 메시지를 출력

④ 그 외의 값 입력할 때

28행의 'else :' 부분이 수행. 29행에서 잘못 입력되었다는 메시지를 출력, 다시 7행으로 감

리스트 (27)

- **2차원 리스트의 기본 개념**
  - 1차원 리스트를 여러 개 연결한 것, 두 개의 첨자를 사용하는 리스트

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

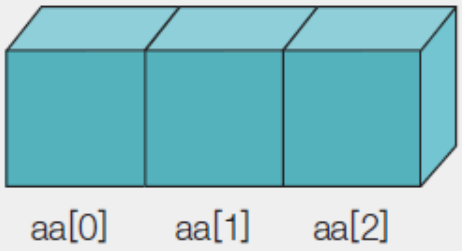


그림 7-12
1차원 리스트의 개념

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

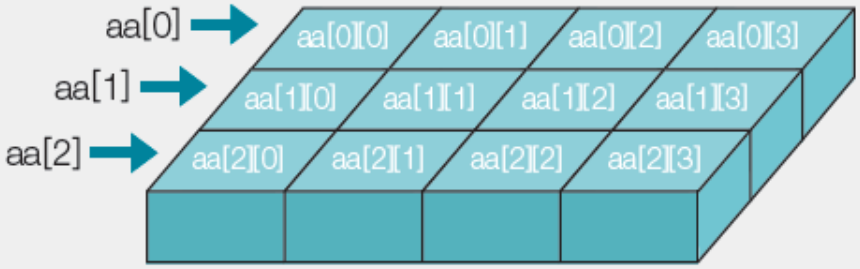


그림 7-13
2차원 리스트의 개념

리스트 (28)

- 첨자가 두 개이므로 중첩 for문을 사용해서 3행 4열 짜리 리스트에 1~12의 숫자를 채움

소스코드 7-7
(파일명 : 07-07.py)

```
1  list1=[]
2  list2=[]
3  value=1
4  for i in range(0, 3) :
5      for k in range(0, 4) :
6          list1.append(value)
7          value+=1
8      list2.append(list1)
9      list1=[]
10
11 for i in range(0, 3) :
12     for k in range(0, 4) :
13         print("%3d" % list2[i][k], end=" ")
14     print("")
```

출력 결과

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

Section 02 Set

Set

- 세트(set)는 수학시간에 배운 집합과 동일
- 세트는 리스트와 마찬가지로 값들의 모임이며, 순서가 없음
- 제공되는 메소드는 리스트와 유사하며, 추가적으로 교집합(intersection)과 합집합(union)이 제공됩니다.

```
>>> a = {1, 2, 3}
>>> b = {3, 4, 5}
>>> a
{1, 2, 3}
>>> b
{3, 4, 5}
Type (a)
<class 'set'>
>>> a.union(b)                    # 합집합  (== a | b)
{1, 2, 3, 4, 5}
>>> a.intersection(b)     # 교집합 (== a & b)
{3}
>>> a – b                         # 차집합
{1, 2}
```

Section 03 Tuple

Tuple (1)

튜플의 생성

- 리스트는 대괄호([ ])로 생성하고 튜플은 괄호( )로 생성.
- 튜플은 값을 수정할 수 없으며 읽기만 가능하므로 읽기 전용의 자료를 저장할 때 사용

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

출력 결과

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

Tuple (2)

- 튜플은 괄호를 생략 가능. 단, 하나의 항목만 가진 튜플을 만들 때는 주의

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

출력 결과

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

Tuple (3)

- 튜플은 읽기 전용이므로 다음 코드는 모두 오류 발생.

```
tt1.append(40)
tt1[0]=40
del(tt1[0])
```

- 하지만 튜플 자체는 del( ) 함수로 지울 수 있음

```
del(tt1)
del(tt2)
```

Tuple (4)

- **튜플의 사용**
  - '튜플이름[위치]'

```
tt1=(10, 20, 30, 40)
tt1[0]
tt1[0]+tt1[1]+tt1[2]
```

출력 결과

```
10
60
```

  - 튜플 범위에 접근 '콜론(시작위치:끝 위치+1)'

```
tt1[1:3]
tt1[1:]
tt1[:3]
```

출력 결과

```
(20, 30)
(20, 30, 40)
(10, 20, 30)
```

Tuple (5)

- 더하기 및 곱하기 연산 가능

```
tt2=('A', 'B')
tt1+tt2
tt2*3
```

출력 결과

```
(10, 20, 30, 40, 'A', 'B')
('A', 'B', 'A', 'B', 'A', 'B')
```

Tuple (6)

- 튜플(tuple)은 리스트와 유사하나, 읽기전용임
- 읽기 전용인 만큼 제공되는 함수도 리스트에 비해 적지만, 속도는 그만큼 빠름.
- 튜플에서 제공되는 메소드는 count, index 정도임

- 튜플을 이용하면 'C'와 같은 언어에서는 변수가 하나 더 필요한 swap 예제를 다음과 같이 간단하게 해결 가능.

```
>>> t = (1, 2, 3)
>>> type(t)
<class 'tuple'>
>>> (a, b) = (1, 2)
>>> print(a, b)
1 2
>>> a, b = 1, 2
>>> print(a, b)
1 2
>>> a, b = b, a                    # a, b값을 swap 하는 것이 간단함
>>> print(a, b)
2 1
```

Tuple (7)

- List, set, tuple간 간단하게 type 변환 가능

```
>>> a = set((1, 2, 3))
>>> a
{1, 2, 3}
>>> type(a)
<class 'set'>
>>> b = list(a)
>>> b
[1, 2, 3]
>>> type(b)
<class 'list'>
>>> c = tuple(b)
>>> c
(1, 2, 3)
>>> type(c)
<class 'tuple'>
>>> d = set(c)
>>> d
{1, 2, 3}
>>> type(d)
<class 'set'>
```

Section 04 Dictionary

Dictionary (1)

- **딕셔너리의 생성**
  - 중괄호({ })로 묶여 있으며 키와 값의 쌍으로 이루어짐

```
딕셔너리변수 = { 키1:값1 , 키2:값2, 키3:값3, … }
```

```
dic1 = { 1:'a', 2:'b', 3:'c'}
dic1
```

출력 결과

```
{1: 'a', 2: 'b', 3: 'c'}
```

```
dic2 = { 'a':1, 'b':2, 'c':3}
dic2
```

출력 결과

```
{'b': 2, 'c': 3, 'a': 1}
```

Dictionary (2)

- 딕셔너리 표현

표 7-1
홍길동 학생의 정보

키	값
학번	1000
이름	홍길동
학과	열공학과

```
student1={ '학번':1000 , '이름':'홍길동', '학과':'열공학과'}
student1
```

출력 결과

```
{'학번': 1000, '이름': '홍길동', '학과': '열공학과'}
```

- 생성한 student1에 연락처 추가

```
student1['연락처']='010-1111-2222'
student1
```

출력 결과

```
{'학번': 1000, '이름': '홍길동', '학과': '열공학과', '연락처': '010-1111-2222'}
```

Dictionary (3)

- 이미 있는 키를 사용하면 쌍이 새로 추가되는 것이 아니라 기존의 값이 변경됨

```
student1['학과']='파이썬학과'
student1
```

출력 결과

```
{'학번': 1000, '이름': '홍길동', '학과': '파이썬학과', '연락처': '010-1111-2222'}
```

- 'del(딕셔너리이름[키])' 함수를 사용하여 삭제

```
del(student1['연락처'])
student1
```

출력 결과

```
{'학번': 1000, '이름': '홍길동', '학과': '파이썬학과'}
```

Dictionary (4)

- **딕셔너리의 사용**
  - 키로 값에 접근하는 예

```
student1['학번']
student1['이름']
student1['학과']
```

출력 결과

```
1000
'홍길동'
'파이썬학과'
```

  - '딕셔너리이름.get(키)' 함수

```
student1.get('이름')
```

출력 결과

```
'홍길동'
```

Dictionary (5)

- 딕셔너리이름.get(키) - 키가 없을 때 아무것도 반환하지 않음
  - 첫째 줄은 에러, 둘째 줄은 값이 없을 경우 반환하는 값이 없음.

```
student1['주소']
student1.get('주소')
```

- '딕셔너리이름.keys( )' 함수 - 딕셔너리의 모든 키 반환

```
student1.keys()
```

출력 결과

```
dict_keys(['학번', '이름', '학과', '연락처'])
```

- 'list(딕셔너리이름.keys( ))' 함수 - 앞에 dict_keys 를 빼줌

```
list(student1.keys())
```

출력 결과

```
['학번', '이름', '학과', '연락처']
```

Dictionary (6)

- '딕셔너리이름.values( )' 함수 - 딕셔너리의 모든 값 반환

```
student1.values()
```

출력 결과

```
dict_values([1000, '홍길동', '파이썬학과', '010-1111-2222'])
```

- '딕셔너리이름.items( )' 함수 - 튜플 형태

```
student1.items()
```

출력 결과

```
dict_items([('학번', 1000), ('이름', '홍길동'), ('학과', '파이썬학과'), ('연
락처', '010-1111-2222')])
```

- In - 딕셔너리에 키가 있으면 True를, 없으면 False 반환

```
'이름' in student1
'주소' in student1
```

출력 결과

```
True
False
```

Dictionary (7)

- for문을 활용하여 딕셔너리의 모든 값 출력

소스코드 7-8
(파일명 : 07-08.py)

```
1  singer={}
2
3  singer['이름']='아이오아이'
4  singer['구성원수']=11
5  singer['데뷔']='프로듀스101'
6  singer['대표곡']='픽미픽미픽미업'
7
8  for k in singer.keys() :
9      print('%s  -->  %s'  % (k, singer[k]))
```

출력 결과

```
이름  -->  아이오아이
구성원수  -->  11
대표곡  -->  픽미픽미픽미업
데뷔  -->  프로듀스101
```

Dictionary (8)

음식 궁합 프로그램 완성

- foods라는 딕셔너리에 키와 값을 넣음. while(True)로 무한 반복.

소스코드 7-9
(파일명 : 07-09.py)

```
## 변수 선언 부분
foods = {   "떡볶이" : "오뎅",
            "짜장면" : "단무지",
            "라면" : "김치",
            "피자" : "피클",
            "맥주" : "땅콩",
            "치킨" : "치킨무",
            "삼겹살" : "상추" };

## 메인(main) 코드 부분
while (True) :
    myfood = input( str(list(foods.keys())) + " 중 좋아하는 것은 ? " )
    if myfood in foods :
        print(" <%s> 궁합 음식은 <%s> 입니다." % (myfood, foods.get(myfood)))
    elif myfood == "끝" :
        break
    else :
        print("그런 음식이 없네요. 확인해보세요. ")
```

Dictionary (9)

- 삭제는 del을 이용할 수도 있으며 clear를 이용해 한번에 삭제할 수도 있습니다.

```
>>> color = {"cherry": :red", "apple": "red", "banana": "yellow"}
>>> color
{'cherry': 'red', 'apple': 'green', 'banana': 'yellow'}
>>> del color['cherry']
>>> color
{'apple': 'green', 'banana': 'yellow'}
>>> color.clear()
>>> color
{}
```

- List, tuple, set, dictionary등 다양한 객체를 섞어 사용 가능.

```
>>> {'age': 40.5, 'job: [1, 2, 3], 'name: {'kim': 2, 'cho': 1}}
>>> [1, 2, 3, ('green'), {'apple: 1}]
```

Thank You