



Python Programming for Beginners

06 Part1 List와 Tuple

2023학년도 2학기
Suk-Hwan Lee

Computer Engineering
Artificial Intelligence

Creating the Future

Dong-A University
Division of Computer Engineering &
Artificial Intelligence

Section01 리스트의 기본

■ 리스트의 개념

- 딕셔너리를 활용해 음식 궁합을 출력하는 프로그램

- 리스트를 생성하는 방법

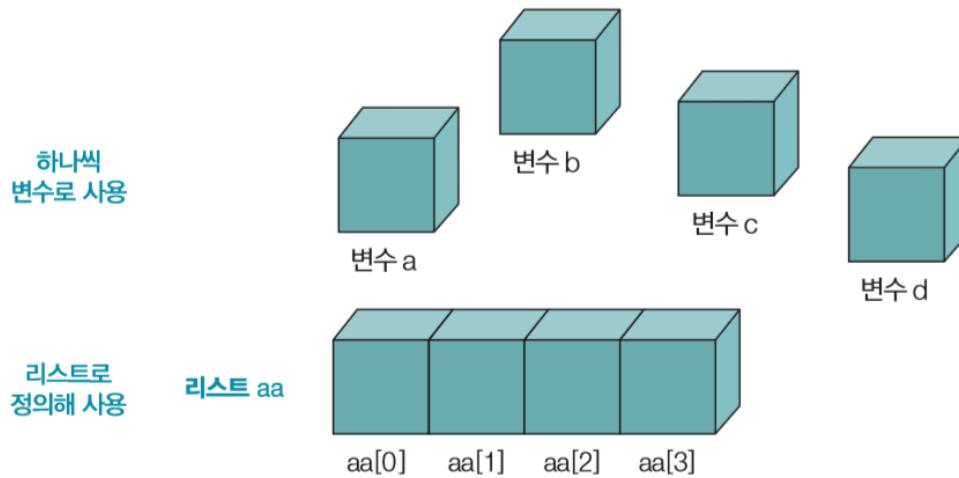


그림 7-1 리스트의 개념

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

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

리스트 정의

[출처] 유틸 파이썬, “5장 리스트”

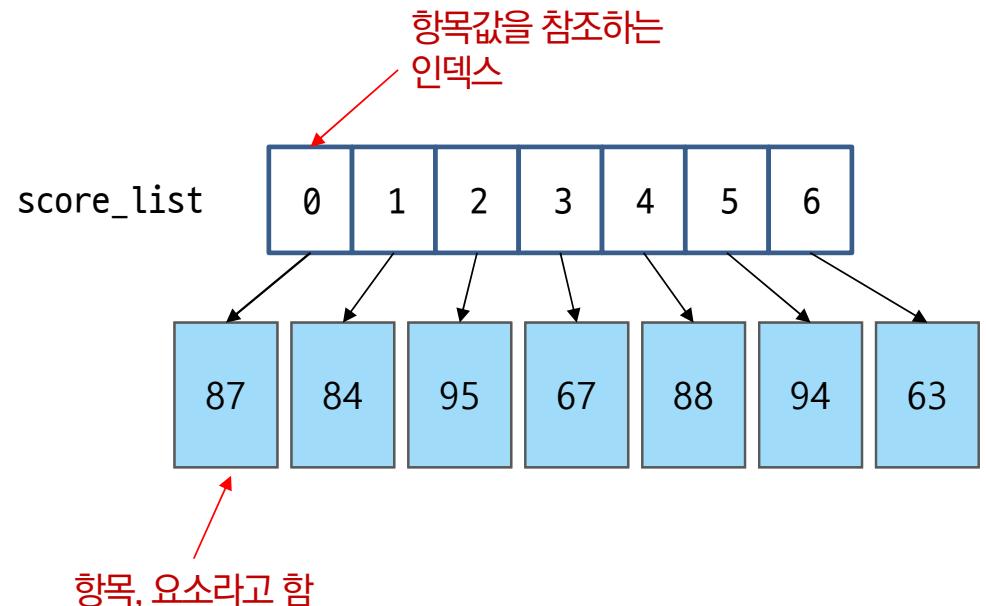
- 개별적인 값을 하나의 변수에 담아서 처리
 - 매번 변수의 이름을 작성하고 관리하는 것보다 편리하고 효율적
 - 한꺼번에 복사하고 조작할 수 있다.
- 항목item 또는 요소element
 - 리스트를 이루는 원소로 쉼표로 구분된 자료 값

대화창 실습 : 수치 값을 가진 리스트 만들기

```
>>> score_list = [87, 84, 95, 67, 88, 94, 63]  
>>> score_list  
[87, 84, 95, 67, 88, 94, 63]  
>>> print(score_list[0], score_list[3])  
87 67
```

요소의 참조는 리스트 이름과 인덱스를 사용한다

- 연속적인 자료값들은 score_list라는 변수와 인덱스를 통해서 참조하는 것이 가능
- 리스트는 대괄호 []내에 쉼표를 이용하여 값을 구분
- ‘세 번째 변수’와 같이 위치를 지정해서 원하는 값을 불러오는 것도 가능



Section01 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
```

- 리스트의 첨자가 순서대로 변할 수 있도록 반복문과 함께 활용

for (4번 반복)

{

값 입력
↓

aa[i]

}

←———— aa[0]부터 aa[3]까지 4번 반복 —————→

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

i값이 0부터 3까지 변함

그림 7-2 for 문으로 리스트값 입력

Section01 리스트의 기본

Code07-03.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)
```

- 1행 : 빈 리스트 생성
- 2~3행 : 4번을 반복해 항목이 4개인 리스트로 만듦
- 6행 : i가 0에서 3까지 4번 반복
- 7행 : input() 함수는 첨자 i가 0부터 시작하므로 i+1로 출력. 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

■ 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.

Section01 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 $0, 2, 4, 8, \dots$ (2의 배수)로 초기화 후 리스트 bb에 역순으로 넣는 과정

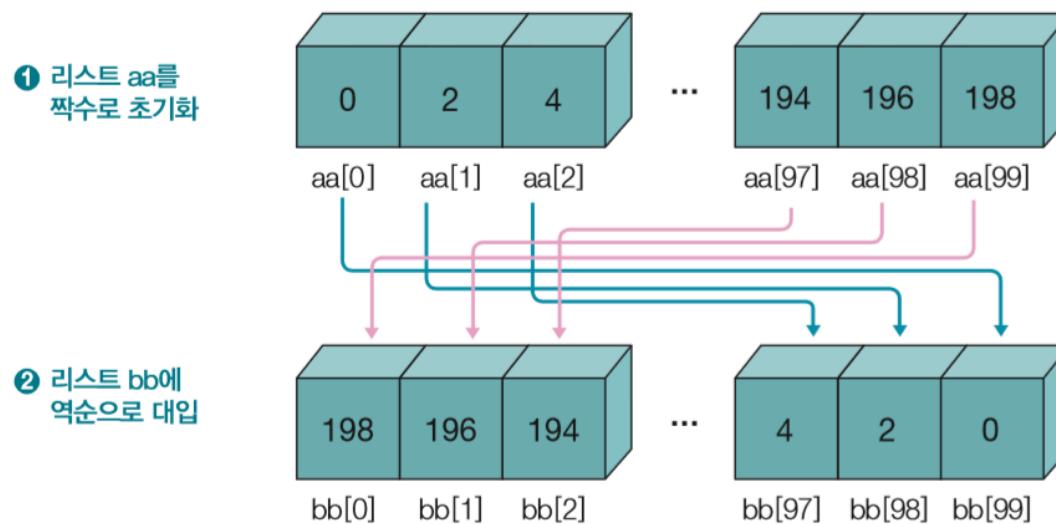


그림 7-3 리스트의 초기화 및 역순 대입

Section01 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
4                      • 5행 : 100번을 반복
5 for i in range(0, 100) : • 6~7행 : 리스트 aa에 value를 추가한 후 2씩 증가
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]에는 %d이, bb[99]에는 %d이 입력됩니다." % (bb[0], bb[99]))
```

- 9행 : 0~99로 100번 반복
- 10행 : i가 0일 때 99-i는 99가 됨. i가 1일 때는 98, i가 2일 때는 97처럼 계속 변해 마지막으로 i가 99일 때는 0이 됨.
- 리스트 bb에는 aa[99], aa[98], aa[97], …, aa[0]의 값이 추가되므로 결국 리스트 aa값이 리스트 bb에 역순으로 입력
- 12행 : 확인을 하려고 bb[0]과 bb[99]를 출력

출력 결과

bb[0]에는 198이, bb[99]에는 0이 입력됩니다.

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.

Section01 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

▪ 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
```

▪ 정수 리스트를 서로 더했을 때는 대응되는 원소끼리 더 해지는 것이 아니라 리스트가 합쳐진다는 것을 명심하자. 대응되는 원소끼리 더하려면 뒤에서 설명하는 넘파이(NumPy)를 사용하여야 한다.

출력 결과

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

슬라이싱에서 설명

▪ 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]

Section01 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]
```

■ 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 2010이라는 값 2개로 변경하는 방법

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

출력 결과

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

Section01 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]에서 네 번째인 aa[3]까지 삭제

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

출력 결과

```
[10, 50]
```

- 두 번째인 aa[1]의 항목 삭제

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

출력 결과

```
[10, 30]
```

- 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
```

Section01 리스트의 기본

■ 리스트 객체에 대한 methods

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)



파이썬 리스트의 내장함수 및 메소드 (Python List built-in functions and methods)

Built-in functions for List

- len(list) : 리스트 길이
- max(list) : 최대 요소
- min(list) : 최소 요소
- list(seq) : 리스트로 변환
- cmp(list1, list2) : 리스트 간 비교
(only for python 2.x version)

List methods

- list.append(obj) : 요소 추가
- list.extend(seq) : 리스트 추가
- list.count(obj) : 요소 개수
- list.index(obj) : 요소 위치 index
- list.insert(index, obj) : index 위치에 obj 삽입
- list.pop(obj=list[-1]) : 마지막 요소 제거
- list.remove(obj) : obj 객체 제거
- list.reverse() : 거꾸로 뒤집기
- list.sort(obj) : 정렬 (디폴트 오름차순)

R, Python 분석과 프로그래밍의 친구 <http://rfriend.tistory.com>

리스트의 메소드(Method)

[출처] 뇌를 자극하는 파이썬3, “05. 데이터 다루기 : 리스트와 튜플, 딕셔너리”

메소드	설명	메소드	설명
append()	<p>리스트의 끝에 새 요소를 추가합니다.</p> <pre>>>> a = [1, 2, 3] >>> a.append(4) >>> a [1, 2, 3, 4]</pre>	remove()	<p>매개 변수로 입력한 데이터를 리스트에서 찾아 발견한 첫 번째 요소를 제거합니다.</p> <pre>>>> a = ['BMW', 'BENZ', 'VOLKSWAGEN', 'AUDI'] >>> a.remove('BMW') >>> a ['BENZ', 'VOLKSWAGEN', 'AUDI']</pre>
extend()	<p>기존 리스트에 다른 리스트를 이어 붙입니다. + 연산자와 같은 기능을 한다고 할 수 있습니다.</p> <pre>>>> a = [1, 2, 3] >>> a.extend([4, 5, 6]) >>> a [1, 2, 3, 4, 5, 6]</pre>	pop()	<p>리스트의 마지막 요소를 뽑아내어 리스트에서 제거합니다.</p> <pre>>>> a = [1, 2, 3, 4, 5] >>> a.pop() 5 >>> a [1, 2, 3, 4] >>> a.pop() 4 >>> a [1, 2, 3]</pre>
insert()	<p>첨자로 명시한 리스트 내의 위치에 새 요소를 삽입합니다. insert(첨자, 데이터)의 형식으로 사용합니다.</p> <pre>>>> a = [2, 4, 5] >>> a.insert(0, 1) # 0 위치(첫 번째)에 데이터 1을 삽입합니다. >>> a [1, 2, 4, 5] >>> a.insert(2, 3) # 2 위치(세 번째)에 데이터 3을 삽입합니다. >>> a [1, 2, 3, 4, 5]</pre>		<p>한편, 마지막이 아닌 특정 요소를 제거하고 싶을 때에는 pop() 메소드에 제거하고자 하는 요소의 인덱스를 입력하면 됩니다.</p> <pre>>>> a = [1, 2, 3, 4, 5] >>> a.pop(2) # 3번째 요소 제거 3 >>> a [1, 2, 4, 5]</pre>

리스트의 메소드(Method)

메소드	설명	메소드	설명
index()	<p>리스트 내에서 매개변수로 입력한 데이터와 일치하는 첫번째 요소의 첨자를 알려줍니다. 찾고자 하는 데이터와 일치하는 요소가 없으면 오류를 일으킵니다.</p> <pre>>>> a = ['abc', 'def', 'ghi'] >>> a.index('def') 1 >>> a.index('jkl') Traceback (most recent call last): File "<pyshell#2>", line 1, in <module> a.index('jkl') ValueError: 'jkl' is not in list</pre>	sort()	<p>리스트 내의 요소를 정렬합니다. 매개변수로 reverse = True를 입력하면 내림차순, 아무 것도 입력하지 않으면 오름차순으로 정렬합니다. reverse = True와 같이 이름을 명시하여 사용하는 매개변수를 일컬어 키워드 매개변수라고 합니다.</p> <pre>>>> a = [3, 4, 5, 1, 2] >>> a.sort() >>> a [1, 2, 3, 4, 5] >>> a.sort(reverse = True) >>> a [5, 4, 3, 2, 1]</pre>
count()	<p>매개변수로 입력한 데이터와 일치하는 요소가 몇 개 존재하는지 셹니다.</p> <pre>>>> a = [1, 100, 2, 100, 3, 100] >>> a.count(100) 3 >>> a.count(200) 0</pre>	reverse()	<p>리스트 내 요소의 순서를 반대로 뒤집습니다.</p> <pre>>>> a = [3, 4, 5, 1, 2] >>> a.reverse() >>> a [2, 1, 5, 4, 3] >>> b = ['안', '녕', '하', '세', '요'] >>> b.reverse() >>> b ['요', '세', '하', '녕', '안']</pre>

append() vs extend()

[출처] 유틸 파일썬, “5장 리스트”



주의 : append() 메소드와 extend() 메소드의 차이점

만일 아래의 list1에 append()를 시도하게 되면 우리가 흔히 생각하는 [11, 22, 33, 44, 55, 66]의 결과가 나오지 않는다. list1.append([55, 66]) 메소드를 호출하면 [55, 66]이라는 리스트 항목을 [11, 22, 33, 44] 다음에 추가하여 list1은 [11, 22, 33, 44, [55, 66]]이라는 항목을 가지게 된다.

```
>>> list1 = [11, 22, 33, 44]
>>> list1.append([55, 66])
>>> list1
[11, 22, 33, 44, [55, 66]]
```

만일 리스트 내에 새로운 리스트의 항목을 추가하고자 할 경우에는 extend() 메소드를 사용할 수 있다. 다음과 같은 프로그램을 작성하여 수행해 보자.

```
>>> list1 = [11, 22, 33, 44]
>>> list1.extend([55, 66])
>>> list1
[11, 22, 33, 44, 55, 66]
```

Section01 리스트의 기본

■ 간단한 예제

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

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

```
myList.append(40)  
print(myList)
```

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

```
print(myList.pop())  
print(myList)
```

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

```
myList.sort()  
myList
```

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

```
myList.reverse()  
myList
```

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

```
print(myList.index(20))
```

```
1
```

```
myList.insert(2, 222)  
myList
```

```
[30, 20, 222, 10]
```

```
myList.remove(222)  
myList
```

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

```
myList.extend([77, 88, 00])  
myList
```

```
[30, 20, 10, 77, 88, 0]
```

```
myList.count(77)
```

```
1
```

```
len(myList)
```

```
6
```

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

myList.insert(2, 222)에서 2는
myList[2]의 위치를 의미, 리스트
는 0번부터 시작 하므로 세 번째 위치가 뒤로 밀리고 그 자리에 222가
삽입

Section01 리스트의 기본

- 기존 리스트는 변경하지 않고 정렬된 새로운 리스트 생성

```
myList
```

```
[30, 20, 10, 77, 88, 0]
```

```
newList = sorted(myList)  
newList
```

```
[0, 10, 20, 30, 77, 88]
```

리스트명.sort

newList = sorted(리스트명)

range()나 문자열을 이용하여 리스트 만들기

- range(1, 10)이라는 함수를 통해 1부터 9까지의 숫자의 **열 sequence**을 얻은 후 이 열을 원소로 가지는 리스트를 list() 함수를 통해 생성

대화창 실습 : 다양한 방법으로 리스트 만들기

```
>>> list1 = list()          # 빈 리스트 생성하기 1
>>> list2 = []              # 빈 리스트 생성하기 2
>>> list3 = list((1, 2, 3))  # 투플로부터 리스트 생성
>>> list3
[1, 2, 3]
>>> list4 = list(range(1, 10))    # range() 함수로부터 리스트 생성
>>> list4
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list5 = list('ABCDEF')      # 문자열로부터 리스트 생성
>>> list5
['A', 'B', 'C', 'D', 'E', 'F']
```

[출처] 유틸 파이썬, “5장 리스트”

LAB 5-1 : 리스트의 생성

- 1부터 10까지의 숫자 중에서 짝수를 요소로 가지는 even_list라는 리스트를 생성하여라(10을 포함). print() 함수를 사용하여 이 리스트를 다음과 같이 출력하여라.

```
even_list = [2, 4, 6, 8, 10]
```

- range() 함수를 이용하여 1번의 문제를 다시 풀어보시오.

- 'Korea', 'China', 'India', 'Nepal' 의 네 원소를 가지는 nations라는 리스트를 생성하여라. print() 함수를 사용하여 이 리스트를 다음과 같이 출력하여라.

```
nations = ['Korea', 'China', 'India', 'Nepal']
```

- 여러분의 친한 친구 5명의 이름을 원소로 가지는 friends라는 리스트를 생성하여라. 그리고 다음과 같이 출력하여라.

```
friends = ['길동', '철수', '은지', '지은', '영민']
```

- 'XYZ' 문자열을 이용하여 'X', 'Y', 'Z'라는 요소를 가지는 string이라는 이름의 리스트 생성하고 다음과 같이 출력하여라.

```
string = ['X', 'Y', 'Z']
```

리스트내의 항목을 지우는 방법

- 파이썬의 키워드 `del` 사용(명령어임)
- 리스트 클래스에 있는 `remove()`라는 메소드 사용
- `pop()` 메소드를 사용 : 리스트의 특정 위치에 있는 항목을 삭제함과 동시에 그 위치에 있는 항목을 반환

■ `del` 키워드로 삭제하는 방법

코드 5-1 : `del` 명령어를 통한 리스트의 항목 삭제

```
list_del_ex.py
n_list = [11, 22, 33, 44, 55, 66]
print(n_list)      # 전체 항목 출력

del n_list[3]      # 네 번째 항목(44) 삭제
print(n_list)
```

실행결과

```
[11, 22, 33, 44, 55, 66]
[11, 22, 33, 55, 66]
```

- 지정된 인덱스에 위치한 항목을 삭제
- `del 44`와 같은 방법으로 삭제할 수 없다

[출처] 유틸 파일, “5장 리스트”

■ `remove` 메소드로 삭제하는 방법

코드 5-2 : `remove()` 메소드를 이용한 리스트의 항목 삭제

```
list_remove_ex1.py
n_list = [11, 22, 33, 44, 55, 66]
print(n_list)
```

```
n_list.remove(44)          # 44라는 값을 가진 항목 삭제
print(n_list)
```

실행결과

```
[11, 22, 33, 44, 55, 66]
[11, 22, 33, 55, 66]
```

- `list`가 가진 메소드로 특정한 값을 리스트의 항목에서 삭제
- `.remove(44)` 와 같은 방법을 사용할 수 있다

■ `remove` 메소드의 문제점

코드 5-3 : 리스트 내부에 존재하지 않는 항목을 삭제하는 경우

```
list_remove_ex2.py
n_list = [11, 22, 33, 44, 55, 66]
print(n_list)
```

```
n_list.remove(88)
print(n_list)
```

실행결과

```
...
ValueError: list.remove(x): x not in list
```

존재하지 않는 항목을
remove()로 삭제하면
오류가 발생

멤버 연산자: in, not in

- 멤버 연산자는 참(True) 혹은 거짓(False)를 반환하는 연산자
- 특정한 원소가 문자열, 리스트, 튜플과 같은 자료구조의 내부에 포함되어 있는지를 검사하는 용도로 사용됨
- ❖ 존재하지 않는 항목을 remove()로 삭제하려 하면 오류가 발생함 : 존재 여부를 미리 확인

in 연산으로 존재여부를 확인하고 True인 경우에만 삭제할 경우는 오류가 생기지 않음

대화창 실습 : 멤버 연산자 in과 리스트

```
>>> a_list = [10, 20, 30, 40]
>>> 10 in a_list          # 리스트의 요소로 10이 있으므로 참
True
>>> 50 in a_list         # 리스트의 요소로 50이 없으므로 거짓
False
>>> 10 not in a_list     # 리스트의 요소로 10이 있으므로 거짓
False
>>> 50 not in a_list     # 리스트의 요소로 50이 없으므로 참
True
```

- in 연산을 사용하여 오류 발생을 미리 방지 : 리스트 내부에 지우고자 하는 값이 있는지 확인

[출처] 유틸 파일썬, “5장 리스트”

■ in 연산자의 사용

코드 5-4 : 리스트 내부에 값이 존재하는가를 확인하는 기능

value_in_list.py

```
n_list = [11, 22, 33, 44, 55, 66]

print(88 in n_list)      # 88은 n_list에 없음
print(55 in n_list)      # 55는 n_list에 있음
```

실행결과

False

True

코드 5-5 : in 연산자를 이용한 안전한 원소 삭제

```
list_remove_ex3.py
n_list = [11, 22, 33, 44, 55, 66]
if (55 in n_list) :      # 리스트의 요소로 55가 있을 경우
    n_list.remove(55)    # 리스트에서 55를 삭제함
if (88 in n_list) :      # 리스트의 요소로 88이 있을 경우
    n_list.remove(88)    # 리스트에서 88을 삭제함
print(n_list)
```

실행결과

[11, 22, 33, 44, 66]

리스트에 적용되는 내장함수

[출처] 유틸 파이썬, “5장 리스트”

- `min()`, `max()`, `sum()`과 같은 파이썬 내장함수의 인자로 리스트를 넘겨 주면 각각 리스트 안의 최솟값, 최댓값, 합 연산 가능
- `len()` 함수는 리스트 내 항목의 개수를 반환

대화창 실습 : 리스트와 내장함수 `min()`, `max()`, `sum()`

```
>>> list1 = [20, 10, 40, 50, 30]
>>> min(list1)      # 리스트의 원소들 중 가장 작은 원소를 구한다.
10
>>> max(list1)      # 리스트의 원소들 중 가장 큰 원소를 구한다.
50
>>> sum(list1)      # 리스트내의 원소의 합을 구한다.
150
```

파이썬은 좋은 내장함수를 많이 가지고 있음
`min`, `max`, `sum` 내장함수는 리스트를 인자로 가질 수 있음

- `list1`은 5개의 항목을 가짐
- `len(list1)`은 리스트 내 항목 개수 반환

- 문자열이 들어있는 리스트 변수에도 `min()`, `max()` 함수 동작
- `sum()` 함수는 문자열에 대해서는 동작하지 않음

대화창 실습 : 문자열 리스트와 내장함수 `min()`, `max()`

```
>>> fruits = ['banana', 'orange', 'apple', 'kiwi']
>>> min(fruits)      # 영어사전 순서로 가장 앞에 있는 단어를 반환
'apple'
>>> max(fruits)      # 영어사전 순서로 가장 뒤에 있는 단어를 반환
'orange'
```

- `min()`과 `max()`은 한글 문자열을 요소로 가지는 리스트에도 동작

대화창 실습 : 한글 문자열 리스트와 내장함수 `min()`, `max()`

```
>>> k_fruits = ['사과', '오렌지', '포도', '바나나']
>>> min(k_fruits)
'바나나'
>>> max(k_fruits)
'포도'
```

Section02 2차원 리스트

■ 2차원 리스트의 개념

- 1차원 리스트를 여러 개 연결한 것, 첨자를 2개 사용

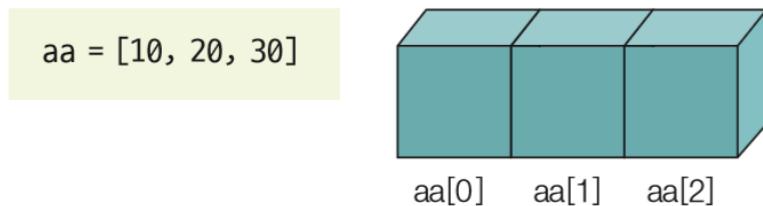


그림 7-4 1차원 리스트의 개념

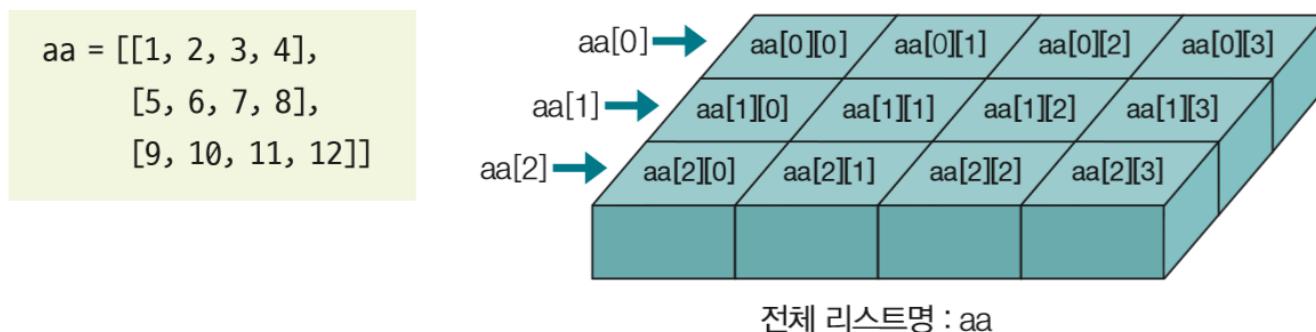


그림 7-5 2차원 리스트의 개념

Section02 2차원 리스트

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

CookPython(2019.10.15) > Code07-06.py > [실행] i

```
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("")
```

- 1~2행 : 1차원 리스트로 사용할 list1과 2차원 리스트로 사용할 list2 준비
- 3행 : value는 리스트에 입력할 1~12의 값으로 사용할 변수
- 4~9행 : 리스트의 행 단위 만들기 위해 3회 반복
- 5~7행 : 4회 반복해 항목 4개인 1차원 리스트 생성하는데, 처음에는 [1, 2, 3, 4] 의 리스트를 만듬
- 8행 : 2차원 리스트에 추가
- 9행 : 1차원 리스트를 다시 비움
- 11~14행 : 2차원 리스트 출력
- 13행 : '리스트명[행][열]' 방식으로 각 항목 출력
- 14행 : 행 하나 출력 후 한 행 띄워서 출력 위해 print() 문 사용

PROBLEMS OUTPUT TERMINAL JUPYTER DEBUG CONSOLE

Lecture/CookPython(2019.10.15)/Code07-06.py"

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

Section02 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
```

■ 불규칙한 크기의 2차원 리스트

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

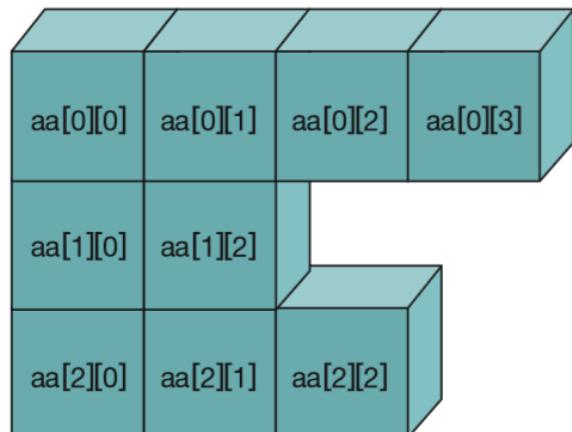


그림 7-6 불규칙한 크기의 2차원 리스트

Section02 2차원 리스트

■ [프로그램 1]의 완성

- 리스트를 이용 터틀 그래픽 응용 프로그램 만들기

- 거북이 한마리의 1차원 리스트
 - 2차원 리스트

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

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

CookPython(2019.10.15) > Code07-07.py > main

```
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 = []      # 거북 2차원 리스트
8 swidth, sheight=500, 500
9
10 def main():
11     turtle.title('거북 리스트 활용')
12     turtle.setup(width=swidth + 50, height=sheight + 50)
13     turtle.screensize(swidth, sheight)
```

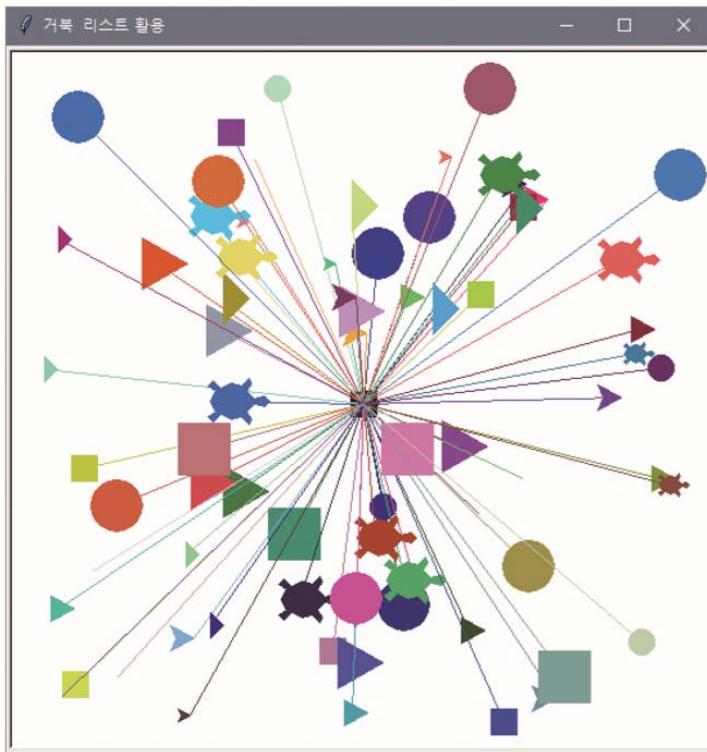
7행: playerTurtles가 거북이 100마리를 저장할 2차원 리스트 준비

Section02 2차원 리스트

```
15 shapeList = turtle.getshapes()
16 for i in range(0, 100):
17     random.shuffle(shapeList)
18     myTurtle = turtle.Turtle(shapeList[0])
19     tX = random.randrange(-swidth / 2, swidth / 2)
20     tY = random.randrange(-sheight / 2, sheight / 2)
21     r = random.random();
22     g = random.random();
23     b = random.random()
24     tsize = random.randrange(1, 3)
25     playerTurtles.append([myTurtle, tX, tY, tsize, r, g, b])
26
27 for tList in playerTurtles:
28     myTurtle = tList[0]
29     myTurtle.color((tList[4], tList[5], tList[6]))
30     myTurtle.pencolor((tList[4], tList[5], tList[6]))
31     myTurtle.turtlesize(tList[3])
32     myTurtle.goto(tList[1], tList[2])
33 turtle.done()
34
35 ## 메인 코드 부분 ##
36 if __name__ == "__main__":
37     main()
```

- 15행 : `turtle.getshapes()` – Return a list of names of all currently available turtle shapes.
- ['arrow', 'blank', 'circle', ..., 'turtle']
- 15행 : `shapeList` 추출하면 ['arrow', 'blank', 'circle', 'classic', 'square', 'triangle', 'turtle'] 등 추출
- 16~25행 : 거북이 100마리를 1차원 리스트로 생성
 - 18행 : `myTurtle`은 거북이 객체 생성
 - 19~20행 : 거북이 위치 `tX`, `tY` 생성
 - 24행 : `tSize`는 거북이 크기 생성
- 25행 : `playerTurtles`에 1차원 리스트 추가
- 27~32행 : 거북이를 `playerTurtles`에서 하나씩 추출해 `tList`에 넣고 반복하여 거북이 100마리를 꺼내서 화면에 선을 그림

Section02 2차원 리스트



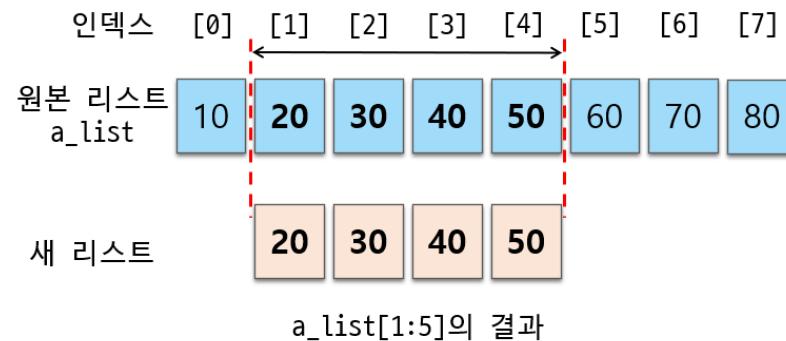
Tip • random.choice 17~18행은 다음과 같이 random.choice(리스트) 이용
하도록 변경 가능. random.choice(리스트)는 리스트 중에서 임의값을
하나 추출해 반환

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

리스트의 슬라이싱

• 슬라이싱 slicing

- 리스트내의 항목을 특정한 구간별로 선택하여 잘라내는 기능
- 구간을 명시하기 위해 리스트_이름[start : end] 문법 사용
- end-1까지의 항목을 새 리스트에 삽입



[출처] 유틸 파일썬, “5장 리스트”

- 콜론(:)을 사용하여 가져올 항목의 범위를 지정
- 시작 인덱스와 끝 인덱스는 생략가능

대화창 실습 : 리스트와 슬라이싱

```
>>> a_list = [10, 20, 30, 40, 50, 60, 70, 80]
>>> a_list[1:5]
[20, 30, 40, 50]
>>> a_list[0:1]
[10]
>>> a_list[0:2]
[10, 20]
>>> a_list[0:5]
[10, 20, 30, 40, 50]
>>> a_list[1:]
[20, 30, 40, 50, 60, 70, 80]
>>> a_list[:5]
[10, 20, 30, 40, 50]
```



주의

슬라이싱을 할 때 콜론(:) 뒤에 명시한 마지막 인덱스는 슬라이싱 리스트에 포함하지 않는다는 것을 항상 기억하자.

리스트의 슬라이싱

[출처] 유틸 파이썬, “5장 리스트”

- 파이썬의 슬라이싱 기능은 음수 인덱스와 음수 스텝 값을 지원

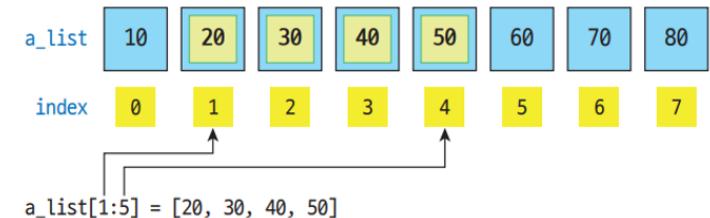
[표 5-2] 슬라이싱 문법과 하는 일

문법	하는 일
<code>a_list[start:end]</code>	start부터 (end-1)까지의 항목들을 슬라이싱(end 인덱스의 항목은 포함하지 않음)
<code>a_list[start:]</code>	start부터 리스트의 끝까지, 즉 뒷부분 모두를 슬라이싱
<code>a_list[:end]</code>	처음부터 end-1번째 인덱스 항목을 슬라이싱
<code>a_list[:]</code>	전체를 슬라이싱
<code>a_list[start:end:step]</code>	start부터 end-1까지를 step만큼 건너뛰며 슬라이싱
<code>a_list[-2:]</code>	뒤에서부터 두 개의 항목을 슬라이싱
<code>a_list[:-2]</code>	처음부터 끝의 두 개를 제외한 모든 항목을 슬라이싱
<code>a_list[::-1]</code>	모든 항목을 가져오되 역순으로 슬라이싱
<code>a_list[1::-1]</code>	처음의 두 개 항목만 슬라이싱

- 시작 인덱스와 마지막 인덱스가 명시된 리스트 슬라이싱

- `a_list[1]`부터 `a_list[5-1]`까지 항목을 가져온다.

```
>>> a_list = [10, 20, 30, 40, 50, 60, 70, 80]
>>> a_list[1:5]
[20, 30, 40, 50]
```

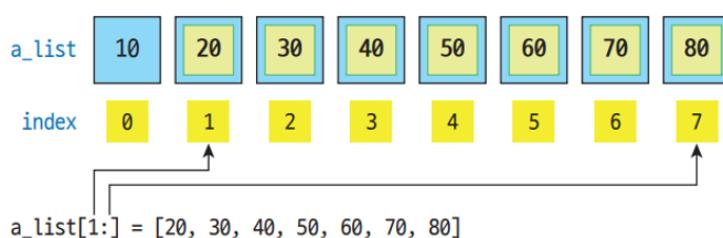


[그림 5-6] 시작 인덱스와 마지막 인덱스가 명시된 리스트 슬라이싱

리스트의 슬라이싱

- 마지막 슬라이싱 인덱스를 생략하는 경우
 - 리스트의 마지막 항목까지 모두 가져옴

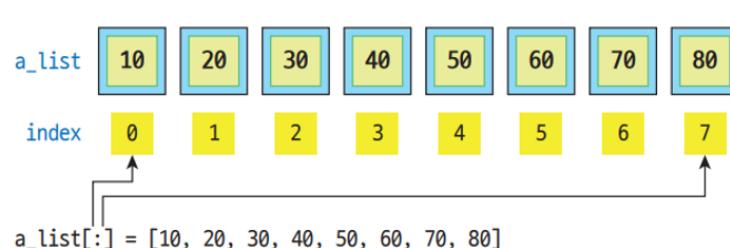
```
>>> a_list[1:]  
[20, 30, 40, 50, 60, 70, 80]
```



[그림 5-7] 마지막 슬라이싱 인덱스를 생략한 결과

- 시작 인덱스와 마지막 인덱스를 모두 생략
 - 리스트의 모든 항목을 다 가져옴

```
>>> a_list[:]  
[10, 20, 30, 40, 50, 60, 70, 80]
```



[그림 5-8] 전체 슬라이싱 인덱스를 생략한 리스트의 슬라이싱

[출처] 유틸 파일썬, “5장 리스트”

- 음수 인덱스를 사용한 슬라이싱

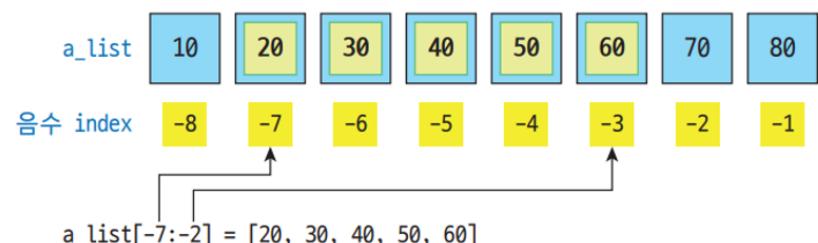
- 가장 끝 원소의 인덱스가 -1이 되며 그 앞의 원소가 -2, -3, ...과 같이 부여됨



- a_list[-7:-2]를 설정한다면 [그림 5-10]과 같이 [20, 30, 40, 50, 60] 항목을 포함하게 된다.

```
>>> a_list[-7:-2]
```

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



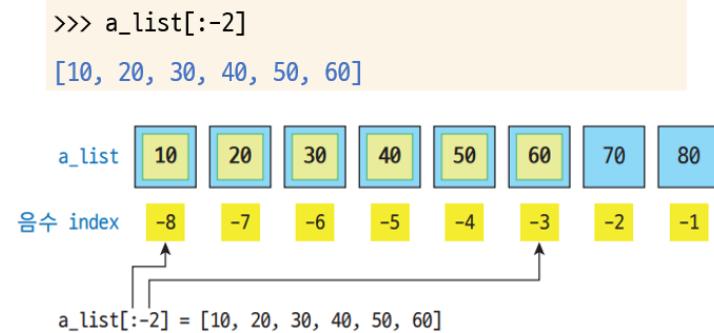
[그림 5-10] 음수 인덱스를 사용한 리스트 슬라이싱

리스트의 슬라이싱

[출처] 유틸 파일, “5장 리스트”

- 음수 인덱스를 사용한 슬라이싱

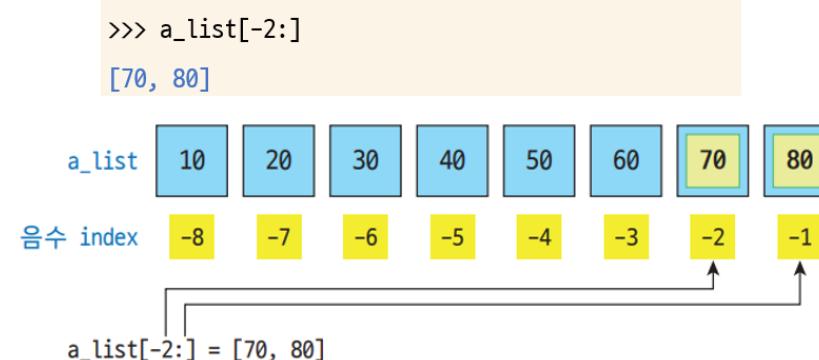
- 첫 번째 인덱스를 생략하여 슬라이싱
✓ $(-2-1)=-3$ 인덱스 항목 값을 가져옴



[그림 5-12] 음수 인덱스 사용시의 리스트 슬라이싱(시작 인덱스 생략)

- a_list[-2:]를 통해 슬라이싱

- 맨 뒤에서부터 2개의 인덱스를 슬라이싱한다.



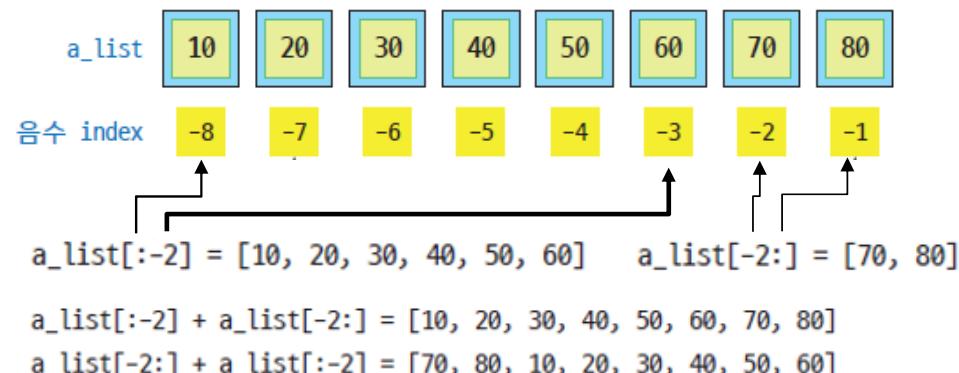
[그림 5-13] 음수 인덱스 사용시의 리스트 슬라이싱(끝 인덱스 생략)

- 슬라이싱의 덧셈 결과

- ✓ a_list[:-2] + a_list[-2:]은 a_list[:]와 같다
- ✓ a_list[-2:] + a_list[:-2]의 결과는 a_list[:]와 같지 않다

```
>>> a_list[:-2] + a_list[-2:]
[10, 20, 30, 40, 50, 60, 70, 80]
```

```
>>> a_list[-2:] + a_list[:-2]
[70, 80, 10, 20, 30, 40, 50, 60]
```



리스트의 슬라이싱

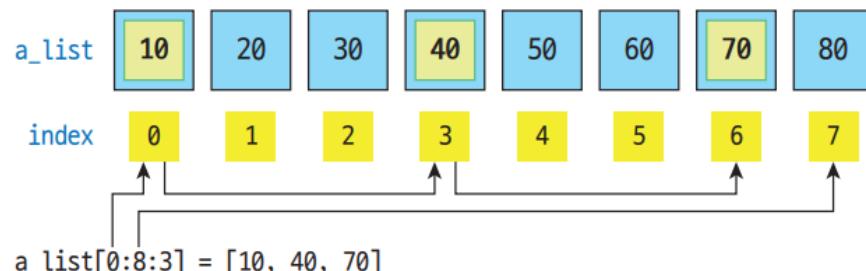
[출처] 유틸 파일썬, “5장 리스트”

- 슬라이싱에 사용하는 스텝

- ✓ 특정 구간의 원소들을 일정한 간격(스텝)만큼 건너뛰며 가져오는 역할

```
>>> a_list[0:8:3]
```

```
[10, 40, 70]
```

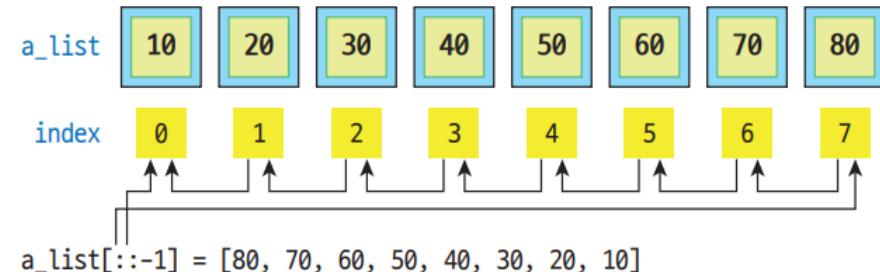


[그림 5-15] 스텝을 이용한 슬라이싱(스텝 값이 3인 경우)

- ✓ 음수의 스텝 값을 줄 경우 전체 구간의 뒤에서부터 앞으로 나아가며 슬라이싱
- ✓ `a_list[::-1]`의 예시

```
>>> a_list[::-1]
```

```
[80, 70, 60, 50, 40, 30, 20, 10]
```



[그림 5-16] 음수 스텝 값을 사용한 리스트 슬라이싱

리스트의 슬라이싱

[출처] 유틸 파이썬, “5장 리스트”

LAB 5-7 : 리스트의 슬라이싱

- range(15) 함수를 사용하여 다음과 같은 리스트를 생성하여라.

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

- 문제 1번의 n_list로부터 슬라이싱을 수행하여 다음과 같은 리스트를 생성하여라.

```
s_list1 = [0, 1, 2, 3, 4]
```

```
s_list2 = [5, 6, 7, 8, 9, 10]
```

```
s_list3 = [11, 12, 13, 14]
```

```
s_list4 = [2, 4, 6, 8, 10]
```

```
s_list5 = [10, 9, 8, 7, 6]
```

```
s_list6 = [10, 8, 6, 4, 2]
```

Section03 튜플

[출처] 유틸 파이썬, “5장 리스트”

• 튜플

- 여러 개의 요소(항목값)를 가지는 컬렉션 자료형 **collection data type**
- 리스트와는 달리 한 번 정해진 요소의 순서를 바꿀 수 없다
- t = ('one', 'two', 'three) 와 같이 괄호로 둘러싸인 형태

• 교환불가능 **immutable** 속성

- 튜플 내부의 객체를 변경하거나 삭제하는 것도 불가능하다는 특징을 가짐

■ 튜플을 생성하는 여러가지 방법

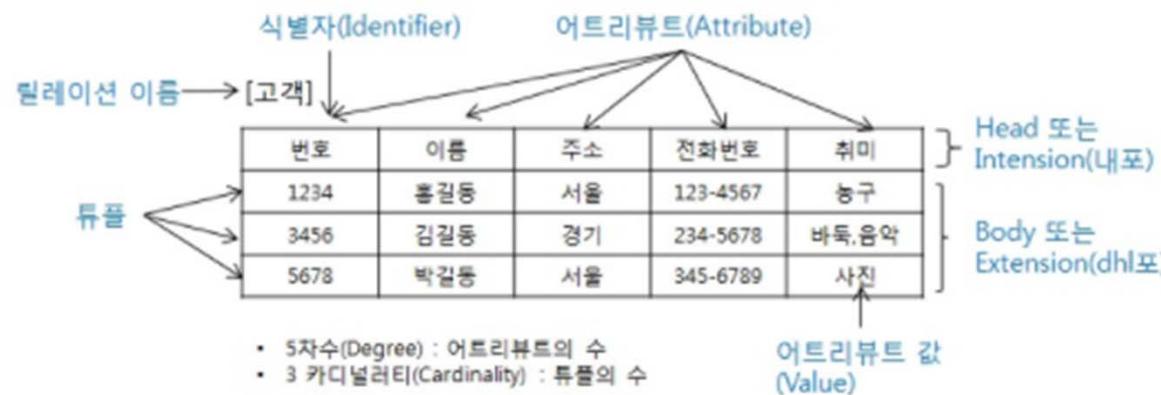
[표 6-10] 튜플을 생성하는 여러가지 방법

빈 튜플 만들기	tuple0 = ()	대화창 실습 : <u>튜플의</u> 생성과 조회
하나의 요소를 가진 튜플 만들기	tuple1 = (1,) # 반드시 쉼표를 사용해야 함에 유의	>>> t = (1, 2, 3, 4)
기본적인 튜플 만들기	tuple2 = (1, 2, 3, 4)	>>> t[0] # 튜플 t의 첫번째 항목을 조회 함
간단한 방식의 튜플 만들기	tuple3 = 1, 2, 3, 4	1
리스트로부터 튜플 만들기	n_list = [1, 2, 3, 4] tuple4 = tuple(n_list)	>>> t[1] 2

튜플 in 데이터베이스

[출처] <http://wiki.hash.kr/index.php/튜플>

- 튜플(tuple) 또는 multiplier라고 부르는 것은 영어의 숫자 체계이다. 그 의미상 '몇배'라는 의미를 포함한다. [1]
- 튜플이란 데이터베이스내의 주어진 목록과 관계있는 속성값의 모음이다. 관련 테이블에서 행한 수치 이상으로 혼합된 자료 요소를 의미 한다. 예를 들어, 지리적 위치는 가끔 2개의 수치로 인해 특성이 명확히 밝혀준다. 한편, 튜플은 Relations을 구성하는 각각의 행을 의미한 다. 튜플의 수를 카디널리티(Cardinality) 또는 기수라고 한다. Relation은 기본적으로 테이블이다.[2]



- Relation에는 이름이 존재하며 2차원의 테이블 형태로 데이터(Value, 어트리뷰트 값)가 관리 된다.
- 각 행을 튜플(Tuple) 이라 하며 각 열을 Attribute 라고 한다.
- Attribute 중에서 튜플을 유일하게 식별할 수 있는 Attribute를 식별자(Identifier)라고 한다.
- 이름.주소와 같은 Attribute 의 이름을 내포(Intension 또는 Head) 라고 한다.
- 234, 홍길동 같은 실제 데이터를 외포(Extension 또는 Body) 라고 한다.
- Relation의 이름과 내포를 스키마(Schema)라고도 한다.
- Relation에는 튜플의 개수를 Cardinality 라고 하며 Attribute의 개수를 차수(Degree) 라고 한다.
- 차수가 1개 이상이고 Cardinality가 0개 이상이면 유효한 Relation이다.
- 튜플을 흔히 Record.Row.Instance라고도 부르며 Cardinality는 컬럼(Column).필드(Field)라고 부른다.

Section03 투플

[출처] 유틸 파이썬, “5장 리스트”



주의 : 하나의 요소를 가지는 투플 선언

투플 생성시 하나의 항목만 갖는 투플을 생성하겠다고 tup = (100)과 같은 할당할 경우 tup = 100과 동일하게 취급되어 tup은 투플이 아니라 정수가 된다. 따라서 tup를 투플로 사용하고 싶을 경우에는 tup = (100,)과 같이 쉼표를 반드시 삽입해야 한다.

1) 잘못된 투플 선언 :

```
>>> tup = (100)
>>> tup      # 정수형 100
100
>>> type(tup)
<class 'int'>
```

2) 올바른 투플 선언 :

```
>>> tup = (100,)
>>> tup      # 투플형 (100,)
(100,)
>>> type(tup)
<class 'tuple'>
```

Section03 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,)
```

■ 튜플의 오류

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

■ 튜플의 삭제

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

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

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

```
tt1[0]=20
```

```
-----  
TypeError                                         Traceback (most recent call last)  
~\AppData\Local\Temp\ipykernel_4836\1689009730.py in <module>  
----> 1 tt1[0]=20  
  
TypeError: 'tuple' object does not support item assignment
```

Section03 튜플

■ 튜플의 사용

■ 튜플 항목에 접근

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

출력 결과

10

60

■ 튜플의 덧셈 및 곱셈 연산

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

출력 결과

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

SELF STUDY 7-4

다음과 같이 2차원 튜플을 생성한 후 모든 값을 출력해 보자.

```
tt = ((1, 2, 3),  
      (4, 5, 6),  
      (7, 8, 9))
```

출력 결과

1 2 3
4 5 6
7 8 9

■ 튜플 범위에 접근

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

출력 결과

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

Section03 튜플

- 예 : 튜플 → 리스트 → 튜플 변환

```
myTuple = (10, 20, 30)
myList = list(myTuple)
myList.append(40)
myTuple = tuple(myList)
myTuple
```

출력 결과

(10, 20, 30, 40)

코드 6-5 : 튜플의 항목 값을 리스트를 이용하여 변경

tuple_item_change.py

```
t_fruits = ('apple', 'orange', 'water melon')

print('변경 전 :', t_fruits)

f_list = list(t_fruits)      # 1) 튜플을 리스트로 변환

f_list[1] = 'kiwi' # 2) 리스트의 두 번째 항목 값을 'kiwi'로 변경

t_fruits = tuple(f_list)    # 3) 리스트를 튜플로 다시 변환함

print('변경 후 :', t_fruits)
```

실행결과

```
변경 전 : ('apple', 'orange', 'water melon')
변경 후 : ('apple', 'kiwi', 'water melon')
```

- 튜플은 내부의 값 변경 불가
 - 하지만 리스트는 내부의 값 변경 가능
- list() 함수를 사용하여 튜플을 리스트로 만들고,
 - 이 리스트의 값을 변경시킨 후,
 - tuple() 함수를 사용하여 리스트를 다시 튜플로 만드는 방법을 사용

Section03 튜플

LAB⁷⁻⁵ 함수는 튜플을 돌려줄 수 있다

C나 C++, Java와 같은 프로그래밍 언어에서는 함수가 하나의 값만을 반환할 수 있다. **파이썬에서**는 함수가 튜플을 반환하게 하면 함수가 여러 개의 값을 동시에 반환할 수 있다. 원의 넓이와 둘레를 동시에 반환하는 함수를 작성하고 테스트 해보자.

원하는 결과

원의 반지름을 입력하시오: 10

원의 넓이는 314.15926535897930이고 원의 둘레는 62.831853071795860이다.

```
import math

def calCircle(r):
    # 반지름이 r인 원의 넓이와 둘레를 동시에 반환하는 함수 (area, circum)
    area = math.pi * r * r
    circum = 2 * math.pi * r
    return area, circum    # 튜플을 반환함

radius = float(input("원의 반지름을 입력하시오: "))
(a, c) = calCircle(radius)
print("원의 넓이는 "+str(a)+"이고 원의 둘레는"+str(c)+"이다.")
```

튜플의 패킹과 언패킹

[출처] 노를 자극하는 파이썬3, “05. 데이터 다루기 : 리스트와 튜플, 딕셔너리”

■ 실습 1 (튜플 패킹(Tuple Packing))

```
>>> a = 1, 2, 3      # 패킹 : 여러 데이터를 튜플로 묶는 것  
>>> a  
(1, 2, 3)
```

■ 실습 2 (튜플 언패킹(Tuple Unpacking))

```
>>> one, two, three = a # 언패킹 : 튜플의 각 요소를  
>>> one                  # 여러 개의 변수에 할당하는 것.  
1  
>>> two  
2  
>>> three  
3
```

■ 실습 3 (언패킹 실패)

```
>>> a = 1, 2, 3      # 튜플 요소 수와  
>>> one, two = a      # 언패킹할 요소의 수가 일치  
Traceback (most recent call last):  
  File "<pyshell#18>", line 1, in <module>  
    one, two = a  
ValueError: too many values to unpack (expected 2)
```

■ 실습 4 (언패킹을 이용한 변수 다중 할당)

```
>>> city, latitude, longitude = 'Seoul', 37.541, 126.986  
>>> city  
'Seoul'  
>>> latitude  
37.541  
>>> longitude  
126.986
```

‘Seoul’, 37.541, 126.986는 괄호 없이
만들어진 튜플

튜플 메소드

[출처] 노를 자극하는 파이썬3, “05. 데이터 다루기 : 리스트와 튜플, 딕셔너리”

메소드	설명
index()	<p>매개변수로 입력한 데이터와 일치하는 튜플 내 요소의 첨자를 알려줍니다. 찾고자 하는 데이터와 일치하는 요소가 없으면 에러를 일으킵니다.</p> <pre>>>> a = ('abc', 'def', 'ghi') >>> a.index('def') 1 >>> a.index('jkl') Traceback (most recent call last): File "<pyshell#4>", line 1, in <module> a.index('jkl') ValueError: tuple.index(x): x not in tuple</pre>
count()	<p>매개변수로 입력한 데이터와 일치하는 요소가 몇 개 존재하는지 셉니다.</p> <pre>>>> a = (1, 100, 2, 100, 3, 100) >>> a.count(100) 3 >>> a.count(200) 0</pre>

함수의 return문과 튜플 활용

[출처] 으뜸 파이썬, “5장 리스트”

코드 6-6 : 원의 면적과 둘레를 튜플 형식으로 반환하는 함수

```
circle_area_and_circum.py
def area_and_circum(radius):      # 원의 면적과 둘레 구하기
    area = 3.14 * radius ** 2
    circum = 2 * 3.14 * radius
    return area, circum           # 튜플을 반환함 - 반환값 (area, circum)
r = 4
a, c = area_and_circum(r)        # 반환받은 튜플을 언패킹함
print('반지름 {}인 원의 면적과 둘레 : {}, {}'.format(r, a, c))
```

실행결과

반지름 4인 원의 면적과 둘레 : 50.24, 25.12

LAB 6-8 : 튜플의 반환

1. x, y의 두 값을 입력으로 받아서 x, y 각각의 제곱 값을 각각 반환하는 square(x, y) 함수를 구현하시오(다음은 구현한 함수의 호출에 관한 예시이다).

```
x = 10
y = 20
x_sq, y_sq = square(x, y)
print('{} 제곱 = {}, {} 제곱 = {}'.format(x, x_sq, y, y_sq))
```

실행결과

10 제곱 = 100, 20 제곱 = 400

2. 다음 튜플 덧셈의 결과는 무엇인가?

(10, 20, 30) + (40, 50, 60)

3. 다음 두 출력문의 결과는 무엇인가? 두 출력문을 실행한 후의 결과를 적고, 어떤 차이가 있는지를 서술하시오(힌트 : 문자열의 반복과 튜플의 반복).

>>> print('Hello ' * 3)

>>> print(('Hello ',) * 3)
