

[Python]



Python으로 배우는 소프트웨어 원리

Chapter 07. 리스트

목차

1. 리스트의 개념
2. 리스트의 사용
3. 리스트의 활용

01

리스트의 개념

01. 리스트의 개념

[데이터 관리의 중요성]

- 많은 데이터를 간편하게 (동일 알고리즘으로) 다루기 위한 데이터 구조가 필요하다.
- 파이썬에는 **리스트(List)**와 **튜플(Tuple)**이 있다.
- 리스트나 튜플은 데이터 집단에 대한 대표 **변수**를 정하고
- 각 데이터 항목에는 그 변수의 **순서번호(index)**로 접근하는 방식 사용

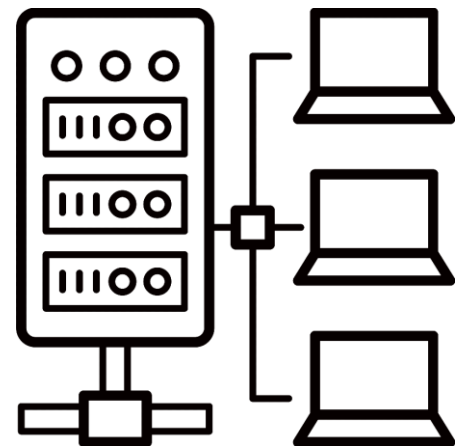
```
## 튜플과 리스트의 차이 > update  
l_scr = [78, 85, 68, 90, 58] # 리스트 구조  
t_scr = (78, 85, 68, 90, 58) # 튜플 구조
```

```
sum = 0  
len_item = len(l_scr)  
for i in range(len_item):  
    l_scr[i] += 5
```

```
for x in l_scr:  
    sum += x  
avg = sum / len_item
```

```
print(sum, avg)
```

❖ Ch07_List03.py



01. 리스트의 개념

I. 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 관리**하는 리스트가 필요!
- ❖ 학생 n명의 성적을 동일 알고리즘으로 처리 하려면
 - 학생들의 성적 전체를 score **변수**로 정하고,
 - 각 학생의 성적은 score의 **순서번호(index)**로 접근하면 가능

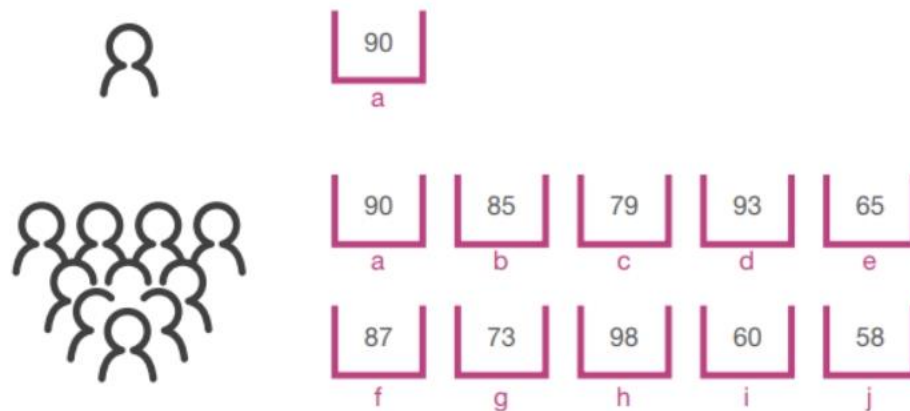


그림 7-1 리스트의 필요성

01. 리스트의 개념

II. 리스트의 구조

- **컬렉션(collection)** : 많은 데이터를 처리할 때 변수처럼 개별 요소로 처리하는 대신 , 그룹으로 묶어서 처리하기 위한 데이터 구조를 지원
 - 리스트, 튜플, 딕셔너리 등이 있음
- **리스트(list)**는 여러 개의 데이터를 하나의 이름으로 저장하는 가장 일반적인 형태
- 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

```
리스트명 = [항목1, 항목2, 항목3, 항목4, 항목5, ... 항목n]
인덱스   → [0]    [1]    [2]    [3]    [4]    ... [n-1]
```

```
a = [90, 85, 79, 93, 65, 87, 73, 98, 60, 58]
b = ['홍길동', '김유신', 3.5, 6, ['a', 'b']]
```

01. 리스트의 개념

II. 리스트의 구조

실습 7-1

베스트셀러 도서의 판매 합계와 평균 구하기

code07-01.py

- 베스트셀러 도서 5권의 판매 수량 합계와 평균을 계산

① [리스트를 사용하지 않는 경우] 데이터 수에 따라 알고리즘이 가변적 변화

➤ 변수 이름을 각각 다르게 정의, 합계를 구할 때는 모든 변수를 덧셈 연산자로 연결

```
01 b1 = 234
02 b2 = 82
03 b3 = 128
04 b4 = 50
05 b5 = 155
06
07 total = b1 + b2 + b3 + b4 + b5
08 print("판매 수량 합계 =", total)
09 print("판매 수량 평균 =", total / 5)
```

리스트 사례

```
l_scr = [78, 85, 68, 90, 58]
```

```
sum = l_scr[0] + l_scr[1] + l_scr[2] + l_scr[3] + l_scr[4]
```

```
avg = sum / 5
```

```
print(sum, avg)
```

❖ Ch07_List00.py

❖ 데이터 수가 바뀌면 알고리즘도 바뀜

→ 알고리즘과 데이터가 종속되어 유지보수가 어려움

01. 리스트의 개념

II. 리스트의 구조

실습 7-1

베스트셀러 도서의 판매 합계와 평균 구하기

code07-01.py

① [리스트를 하는 경우] 데이터 수와 관계 없이 동일 알고리즘 사용

- 가변적 구조인 리스트를 사용하여 이름을 일일이 정의하는 번거로움을 줄어듬
- 합계를 구할 때도 반복문 적용으로 알고리즘이 단순해짐

```
01 books = [234, 82, 128, 50, 155]
02 total = 0
03
04 for x in books :           # 리스트를 이용한 합계 구하기
05     total += x
06 print("판매 수량 합계 =", total)
07 print("판매 수량 평균 =", total / len(books))
```

판매 수량 합계 = 649

판매 수량 평균 = 129.8

```
## 리스트 사례
l_scr = [78, 85, 68, 90, 58] #리스트 구조
sum = 0
```

```
for x in l_scr:
    sum += x
avg = sum / len(l_scr)
```

```
print(sum, avg) ❖ Ch07_List03.py
```

❖ 데이터 수가 바뀌어도 알고리즘은 동일

→ 데이터 수의 가변성은 리스트의 저장 순서인 인덱스를 사용하여 해결

01. 리스트의 개념

II. 리스트의 구조

여기서 잠깐 in 연산자

- in 연산자는 단독으로 사용되기도 하는데, 다음과 같이 특정 항목이 리스트에 있는지를 검사해서 True나 False로 결과를 알려줌

```
>>> alist = ['홍길동', '김유신', 3]
>>> '이순신' in alist          # '이순신'이 리스트에 있으면(in) True를 반환
False
>>> 5 not in alist             # 5가 리스트에 없으면(not in) True를 반환
True
```

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

```
if 85 in l_scr:
    print('True')
```

01. 리스트의 개념

II. 리스트의 구조

여기서 잠깐

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

01. 리스트의 개념

II. 리스트의 구조

여기서 잠깐

리스트(list)와 배열(array)의 구조 차이

| | 배열(Array) | 리스트(Linked List) |
|--------|--------------------|------------------------|
| 저장 방식 | 인접한 위치에 동일한 구조로 저장 | 각 데이터를 연결 구조(비인접)로 저장 |
| 데이터 형식 | 동일한 데이터형식만 저장 | 다양한 데이터형식을 함께 저장 |
| 저장 크기 | 운영 중에 크기 고정 | 가변 크기; 운영 중에 삽입, 삭제 가능 |
| 운영 효율성 | 단순 | 복잡 |



02

리스트의 사용

02. 리스트의 사용

I. 리스트 인덱싱

- 위치 값인 **인덱스(index)**를 이용하여 해당 항목의 값을 가져오거나 수정하는 작업
- 인덱스는 **0**부터 시작
- 리스트 데이터 항목들은 '['과 ']'로 감싼다.

```
>>> a = [90, 85, 79, 93, 65]
```



```
>>> a[0]  
90
```



```
>>> a[3] = 'xy'  
>>> a  
[90, 85, 79, 'xy', 65]
```



그림 7-3 인덱스를 사용한 리스트 항목의 참조

- ① 리스트 항목(item) 중에 리스트를 포함시키려면 마찬가지로 '['과 ']'로 감싼다.

```
>>> alist = ['홍길동', '김유신', 3.5, 6, ['a', 'b']]    # 또 다른 리스트인 ['a', 'b']도 포함  
>>> type(alist)
```

02. 리스트의 사용

I. 리스트 인덱싱

실습 7-2

리스트의 인덱싱 다양하게 활용하기

② 인덱스를 이용해 항목의 값을 출력

```
>>> alist[1]                # 두 번째 항목의 참조
'김유신'
>>> len(alist)              # 전체 항목 개수
5
>>> alist[5]                # 인덱스의 범위(0~4)를 벗어난 사용으로 에러 발생
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    alist[5]
IndexError: list index out of range
```

```
alist = [1, 2, [31, 32], '우주', 5.0]
```

```
print(alist[0])    # 1
print(alist[2][1]) # 32
```

③ alist의 마지막 항목은 리스트 유형

```
>>> alist[4]
['a', 'b']
>>> alist[4][0]
'a'
>>> alist[4][1]
'b'
```

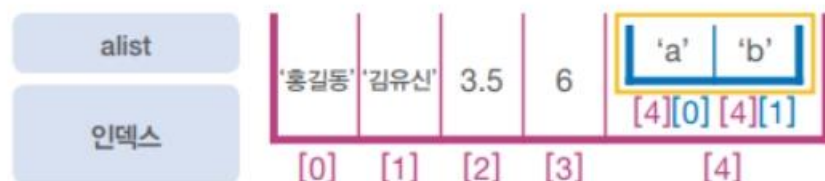


그림 7-4 리스트 중첩

02. 리스트의 사용

I. 리스트 인덱싱

실습 7-2

리스트의 인덱싱 다양하게 활용하기

- ④ 인덱스는 음수를 사용할 수도 있는데, 마지막 항목의 인덱스는 [-1]

➤ `alist[-1];` #뒤에서 1-번째 의미로 사용

```
>>> alist[-1]
['a', 'b']
>>> alist[-2]
6
>>> alist[-5]
'홍길동'
```



그림 7-5 음수를 사용한 인덱스

- ⑤ 인덱싱으로 항목의 값을 바꾸거나 수식에도 사용

```
>>> alist[2] = 7.8
>>> alist[2] + alist[3]
13.8
```



그림 7-6 항목의 값 바꾸기

02. 리스트의 사용

I. 리스트 인덱싱

실습 7-2

리스트의 인덱싱 다양하게 활용하기

- ⑥ 덧셈(+)과 곱셈(*) 연산자에서는 리스트 자체를 피연산자로 하는 연산을 수행

```
>>> alist + alist
['홍길동', '김유신', 7.8, 6, ['a', 'b'], '홍길동', '김유신', 7.8, 6, ['a', 'b']]
>>> alist * 3
['홍길동', '김유신', 7.8, 6, ['a', 'b'], '홍길동', '김유신', 7.8, 6, ['a', 'b'], '홍길동', '김유신', 7.8, 6, ['a', 'b']]
```

- ⑦ 리스트를 출력하려면 리스트 이름을 사용해서 전체 항목을 한 번에 출력하거나 반복문으로 항목을 하나씩 출력

```
>>> print(alist)
['홍길동', '김유신', 7.8, 6, ['a', 'b']]
>>> for x in alist :
    print(x)
```

print(x) 입력 후 Enter를 2번 누르면 실행

```
홍길동
김유신
7.8
6
['a', 'b']
```


02. 리스트의 사용

II. 항목의 추가와 삭제

- ◆ 리스트의 각 항목들은 연결구조(linked list) 형태로 관리되기 때문에
- ◆ 기존 항목에 항목을 추가하거나 제거 하려면 제공되는 메소드를 사용해야 한다.
- 항목 추가 메소드

```
리스트명.append(항목)  
리스트명.insert(위치, 항목)
```

- 항목 위치 검색 메소드

```
리스트명.index(항목)
```

- 항목 삭제 메소드

```
리스트명.remove(항목)  
del(리스트명[위치]) 또는 del 리스트명[위치]  
리스트명.pop()  
리스트명.clear()
```

```
## 항목의 추가와 삭제  
l_scr = [78, 85, 68, 90, 58] #리스트 구조
```

```
l_scr.append(100) #항목 추가  
print(l_scr)
```

```
l_scr.insert(3, 60) #항목 삽입  
print(l_scr)
```

```
print(l_scr.index(60)) #인덱스 검색
```

```
l_scr.remove(60) #항목 제거  
print(l_scr)
```

```
# 맨 뒤의 항목 하나를 삭제
```

```
# 모든 항목 삭제
```

02. 리스트의 사용

II. 항목의 추가와 삭제

실습 7-3

리스트에 항목 추가하기

- ① 빈 리스트를 만들고 리스트의 내용을 확인

```
>>> alist = []  
>>> alist  
[]
```

alist



그림 7-8 빈 리스트 생성

- ② 리스트에 항목을 추가하고, 변경된 리스트를 확인

```
>>> alist.append(30)  
>>> alist.append('홍길동')  
>>> alist  
[30, '홍길동']
```

alist



그림 7-9 실행 순서대로 항목 추가

- ③ 리스트의 특정 위치에 항목을 추가하려면 insert() 메소드를 사용

```
>>> alist.insert(1, '김유신')  
>>> alist  
[30, '김유신', '홍길동']  
>>> alist.index('김유신')  
1
```

alist



그림 7-10 위치를 지정하여 항목 추가

02. 리스트의 사용

[실습] 리스트 항목 추가/변경 방법

I. 빈 리스트에 추가 방법

```
alist = []          #빈 리스트
alist[0] = 1         #없는 위치에 대입을 하므로 Error 발생
alist.append(1)      #끝에 값 추가
alist.append(2)      #끝에 값 추가
alist
```

```
alist = []
alist[0] = 1
Traceback (most recent call last):
  File "<pyshell#70>", line 1, in <module>
    alist[0] = 1
IndexError: list assignment index out of range
alist.append(1)
alist.append(2)
alist
[1, 2]
```

II. 초기화된 리스트에 변경 방법

```
alist = [0, 0, 0]   #초기화된 리스트
alist[0] = 1         #[0] 위치에 값 변경
alist[1] = 2         #[1] 위치에 값 변경
alist
alist.append(3)      #끝에 값 추가
alist
```

```
alist = [0, 0, 0]    #초기화된 리스트
alist[0] = 1
alist[1] = 2
alist
[1, 2, 0]
alist.append(3)
alist
[1, 2, 0, 3]
```

02. 리스트의 사용

II. 항목의 추가와 삭제

실습 7-4

리스트에서 항목 삭제하기

- ① remove() 메소드를 사용하면 리스트에서 특정 항목을 삭제
 - 제거할 값과 일치하는 항목 중에 앞선 **하나만** 삭제됨

```
>>> alist.remove('홍길동')
>>> alist
[30, '김유신']
```



그림 7-11 특정 항목 삭제

- ① del() 메소드를 사용하여 리스트 전체 또는 특정 위치의 항목을 삭제
 - 해당 항목만 제거되고, 뒤 항목들은 앞으로 당겨져서 리스트 유지

```
>>> del(alist[0])
>>> alist
['김유신']
```



그림 7-12 특정 위치에 있는 항목 삭제

02. 리스트의 사용

[실습] 리스트 메소드 사용

```
alist = [1, 2, 2, 3, [1, 2, 3]]
alist.append(4)
alist      #[1, 2, 2, 3, [1, 2, 3], 4]
alist.insert(0, 0)
alist      #[0, 1, 2, 2, 3, [1, 2, 3], 4]
alist[5].insert(0, 0)
alist      #[0, 1, 2, 2, 3, [0, 1, 2, 3], 4]
alist.index(3)      #4
alist[5].index(3)    #3
alist.remove(2)
alist      #[0, 1, 2, 3, [0, 1, 2, 3], 4]
alist[4].remove(2)
alist      #[0, 1, 2, 3, [0, 1, 3], 4]
```

```
del(alist)  #del()는 존재 자체를 제거
alist
```

```
>>> alist = [1, 2, 2, 3, [1, 2, 3]]
>>> alist.append(4)
>>> alist
[1, 2, 2, 3, [1, 2, 3], 4]
>>> alist.insert(0, 0)
>>> alist
[0, 1, 2, 2, 3, [1, 2, 3], 4]
>>> alist[5].insert(0, 0)
>>> alist
[0, 1, 2, 2, 3, [0, 1, 2, 3], 4]
>>> alist.index(3)
4
>>> alist[5].index(3)
3
>>> alist.remove(2)
>>> alist
[0, 1, 2, 3, [0, 1, 2, 3], 4]
>>> alist[4].remove(2)
>>> alist
[0, 1, 2, 3, [0, 1, 3], 4]
>>>
```

02. 리스트의 사용

II. 항목의 추가와 삭제

실습 7-4

리스트에서 항목 삭제하기

- ③ pop() 메소드는 리스트의 가장 마지막 항목을 끄집어내는데, 리스트에서 해당 항목은 리스트에서 제거된다.
- 리스트 값을 하나씩 끄집어내어 사용하는 용도로 사용

```
>>> deleted = alist.pop()
>>> alist
[]
>>> deleted           # 삭제한 항목이 저장된 변수
'김유신'
```



그림 7-13 마지막 항목 삭제

- ③ 리스트의 모든 항목을 한 번에 삭제하려면 clear() 메소드를 사용

```
>>> alist = [1, 2, 3, 4, 5]
>>> alist.clear()
>>> alist
[]
```

02. 리스트의 사용

[실습] 리스트의 pop() 메소드 사용

I. 맨 끝 항목 끄집어내기

```
alist = [1, 2, 3, 4, 5]
a = alist.pop()    #맨 끝 항목
b = alist.pop()
c = a + b
alist
a, b, c
```

II. 지정 항목 끄집어내기

```
alist = [1, 2, 3, 4, 5]
a = alist.pop(1)    #앞에서 1번째
b = alist.pop(-1)   #뒤에서 1번째
c = a + b
alist
a, b, c
```

```
alist = [1, 2, 3, 4, 5]
a = alist.pop()
b = alist.pop()
c = a + b
alist
[1, 2, 3]
a, b, c
(5, 4, 9)
```

```
alist = [1, 2, 3, 4, 5]
a = alist.pop(1)
b = alist.pop(-1)
c = a + b
alist
[1, 3, 4]
a, b, c
(2, 5, 7)
```

02. 리스트의 사용

II. 항목의 추가와 삭제

여기서 잠깐 clear()와 del()의 차이

- 리스트 항목을 모두 삭제하여 빈 리스트로 만드는 clear()와는 달리
- del() 함수는 객체 자체를 메모리에서 삭제

```
>>> alist = [1, 2, 3, 4, 5]
>>> del(alist)
>>> alist
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    alist
NameError: name 'alist' is not defined
```


02. 리스트의 사용

[실습] 키보드로 입력하는 값 리스트에 저장하기

◆ append() 메소드 사용

```
alist = []
```

```
for _ in range(3):
```

```
    alist.append(int(input("> ")))
```

```
alist = []  
for _ in range(3):  
    alist.append(int(input("> ")))
```

```
> 1
```

```
> 2
```

```
> 3
```

```
alist
```

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

02. 리스트의 사용

[실습] 키보드로 입력하는 값 리스트에 저장하기 (split() 사용)

- ◆ list() 또는 [] 사용 (입력값을 문자열로 처리)

```
slist = list( input("> ").split() )
```

```
slist = [ input("> ").split() ]
```

- ◆ 숫자로 변환하려면

```
ilist = list( int(x) for x in input("> ").split() )
```

```
ilist = [ int(x) for x in input("> ").split() ]
```

```
slist = list(input(">리스트 값 입력(공백으로 구분): ").split())
>리스트 값 입력(공백으로 구분): a b 1 2 5
slist
['a', 'b', '1', '2', '5']
ilist = list(int(x) for x in input(">숫자 값 입력(공백으로 구분): ").split())
>숫자 값 입력(공백으로 구분): 1 2 3 4 5
ilist
[1, 2, 3, 4, 5]
ilist = [int(x) for x in input(">숫자 값 입력(공백으로 구분): ").split()]
>숫자 값 입력(공백으로 구분): 1 2 3 4 5
ilist
[1, 2, 3, 4, 5]
```

리스트 값 입력(기본적으로 문자열로 인식)

```
slist = list(input(">리스트 값 입력(공백으로 구분): ").split())
```

```
print(slist)
```

리스트 값 입력(문자열을 integer로 변환)

```
ilist = list(int(x) for x in input(">숫자 값 입력(공백으로 구분): ").split())
```

```
print(ilist)
```

```
ilist = [int(x) for x in input(">숫자 값 입력(공백으로 구분): ").split()]
```

```
print(ilist)
```

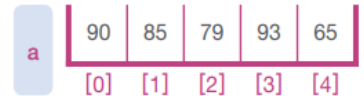
02. 리스트의 사용

III. 리스트 슬라이싱

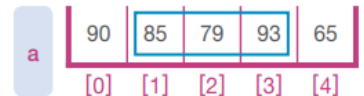
- 범위로 여러 개의 항목을 참조하는 것, 범위 표시에는 콜론(:) 기호를 사용
- 슬라이싱 결과도 리스트 자료형
- 리스트명[시작 index : 종료 index]
- ❖ alist[0:3] # [0], [1], [2]가 해당
- ✓ #for in range(시작 index, 종료 index)와 같은 방식

```
리스트명[인덱스1:인덱스2] # 인덱스1부터 (인덱스2 앞)까지 추출  
리스트명[인덱스1:]       # 인덱스1부터 마지막까지 추출  
리스트명[:인덱스2]       # 첫 번째부터 (인덱스2 앞)까지 추출  
리스트명[:]              # 리스트의 전체 항목 추출
```

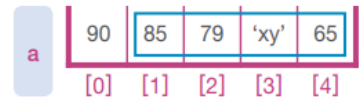
```
>>> a = [90, 85, 79, 93, 65]
```



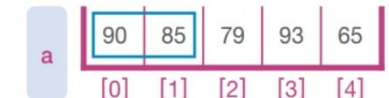
```
>>> a[1:4]  
[85, 79, 93]
```



```
>>> a[1:]  
[85, 79, 93, 65]
```



```
>>> a[:2]  
[90, 85]
```



```
>>> a[:]  
[90, 85, 79, 93, 65]
```



그림 7-14 다양한 슬라이싱 방법

02. 리스트의 사용

III. 리스트 슬라이싱

실습 7-5

리스트 슬라이싱으로 항목 추출, 변경, 삭제하기

- ① 항목을 입력해서 리스트를 만들고, 여러 방법으로 슬라이싱

```
>>> blist = ['정약용', '유관순', 50]
```

```
>>> blist[1:2]
```

```
['유관순']
```

```
>>> blist[:2]
```

```
['정약용', '유관순']
```

```
>>> blist[2:]
```

```
[50]
```

```
>>> blist[:]
```

```
['정약용', '유관순', 50]
```

슬라이싱의 결과는 항목이 하나이더라도 항상 리스트임
참고로, blist[1]로 인덱싱하면 문자열 '유관순'으로 출력

02. 리스트의 사용

III. 리스트 슬라이싱

실습 7-5

리스트 슬라이싱으로 항목 추출, 변경, 삭제하기

- ② 슬라이싱을 사용해서 리스트 항목을 다른 값으로 변경 가능

```
>>> blist; len(blist)           # 현재의 리스트 값과 개수 확인
['정약용', '유관순', 50]
3
>>> blist[1:2] = [30, '이순신']  # '유관순'을 2개의 항목으로 변경
>>> blist
['정약용', 30, '이순신', 50]
>>> len(blist)                 # 항목의 개수가 4로 증가
4
```

- ③ 슬라이싱을 이용하면 여러 항목을 한 번에 삭제할 수도 있음

```
>>> blist                       # 현재의 리스트 값 확인
['정약용', 30, '이순신', 50]
>>> del(blist[2:])              # 세 번째 항목부터 끝까지 삭제
>>> blist                       # 삭제 후 리스트 값 확인
['정약용', 30]
```

02. 리스트의 사용

III. 리스트 슬라이싱

- 리스트 항목 건너뛰며 추출

```
aa = [10, 20, 30, 40, 50, 60, 70]
aa[::2]      #앞에서 2개 주기로 건너뛰며 추출
aa[::-2]     #뒤에서 2개 주기로 건너뛰며 추출
aa[::-1]     #뒤에서 1개 주기로 건너뛰며 추출
```

출력 결과

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

02. 리스트의 사용

III. 리스트 슬라이싱

■ 리스트 값 변경

- 20을 200과 201이라는 값 2개로 변경
 - 변경 대상을 범위로 복수 개(리스트 형태) 지정하면 대입(=) 대상도 리스트 형태([])로 표현
 - 결국 범위 안에 있는 개별 항목에 대해 우측 리스트의 각 항목이 대응되어 대체됨

```
aa = [10, 20, 30]
aa[1:2] = [200, 201]    #슬라이싱 결과 리스트에 리스트 대입
aa
```

출력 결과

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

- aa[1] 위치에 리스트 [200, 201]로 대신(replace) 사용
 - 변경 대상이 특정 항목이므로 우측 리스트의 대상이 그대로 대체됨

```
aa = [10, 20, 30]
aa[1] = [200, 201]    # index 1 위치 값에 리스트로 대입
aa
```

출력 결과

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

02. 리스트의 사용

[실습] 리스트 값 변경

I. 개별 항목 변경

```
alist = [1, 2, 3]
alist[1] = 20 #값(value)에 값을 대입
alist[2] = [31, 32] #값에 리스트를 대입
alist
```

```
alist = [1, 2, 3]
alist[1] = 20
alist
[1, 20, 3]
alist[2] = [31, 32]
alist
[1, 20, [31, 32]]
```

II. 범위 항목 변경 (★)

```
alist = [1, 2, 3]
alist[1:2] = 20 #리스트에 값을 대입 > Error 발생
alist[1:2] = [20] #리스트에 리스트를 대입
alist
alist[2:3] = [[30, 31]] #리스트에 리스트 값을 대입
alist
```

```
alist = [1, 2, 3]
alist[1:2] = 20
Traceback (most recent call last):
  File "<pyshell#79>", line 1, in <module>
    alist[1:2] = 20
TypeError: can only assign an iterable
alist[1:2] = [20]
alist
[1, 20, 3]
alist[2:3] = [[30, 31]]
alist
[1, 20, [30, 31]]
```


02. 리스트의 사용

III. 리스트 슬라이싱

- 리스트 값 변경

- 두 번째인 aa[1]의 항목 삭제

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

출력 결과

```
[10, 30]
```

- 두 번째인 aa[1]에서 네 번째인 aa[3]까지 삭제

```
aa = [10, 20, 30, 40, 50]  
aa[1:4] = [] #리스트에 빈 리스트를 대입  
aa
```

출력 결과

```
[10, 50]
```

02. 리스트의 사용

III. 리스트 슬라이싱

■ 리스트 값 변경

I. 리스트 항목 제거

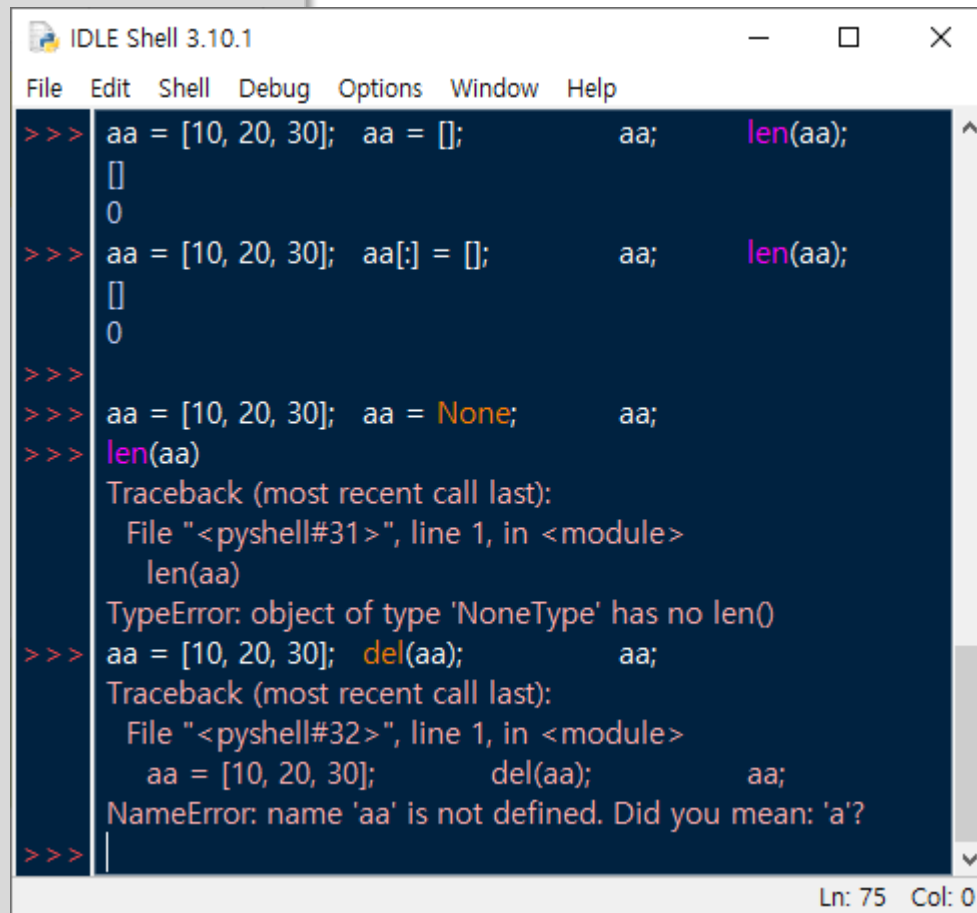
```
aa = [10, 20, 30];  
aa = [];  
aa; len(aa);
```

```
aa = [10, 20, 30];  
aa[:] = [];  
aa; len(aa);
```

II. 리스트 변수 삭제

```
aa = [10, 20, 30];  
aa = None; #빈 변수로 만들  
aa; type(aa);
```

```
aa = [10, 20, 30];  
del(aa);  
aa; len(aa); #변수 삭제
```



```
IDLE Shell 3.10.1  
File Edit Shell Debug Options Window Help  
>>> aa = [10, 20, 30]; aa = []; aa; len(aa);  
[]  
0  
>>> aa = [10, 20, 30]; aa[:] = []; aa; len(aa);  
[]  
0  
>>>  
>>> aa = [10, 20, 30]; aa = None; aa;  
>>> len(aa)  
Traceback (most recent call last):  
  File "<pyshell#31>", line 1, in <module>  
    len(aa)  
TypeError: object of type 'NoneType' has no len()  
>>> aa = [10, 20, 30]; del(aa); aa;  
Traceback (most recent call last):  
  File "<pyshell#32>", line 1, in <module>  
    aa = [10, 20, 30]; del(aa); aa;  
NameError: name 'aa' is not defined. Did you mean: 'a'?  
>>>
```

02. 리스트의 사용

IV. 리스트 조작 함수

- 글자 순서나 값의 크기를 기준으로 나열

```
리스트명.sort()           # 오름차순 정렬(1, 2, 3 혹은 'a', 'b', 'c')
리스트명.sort(reverse=True) # 내림차순 정렬(3, 2, 1 혹은 'c', 'b', 'a')
sorted(리스트명)          # 정렬된 리스트를 반환
```

실습 7-6

리스트에서 sort() 함수 활용하기

- ① 리스트를 만들고 데이터를 저장, sort() 메소드로 항목을 정렬한 다음 결과를 확인

```
>>> contact = ['정약용', '이순신', '김유신', '유관순']
>>> contact.sort()
>>> contact
['김유신', '유관순', '이순신', '정약용']
```

- ② 리스트의 정렬 방식은 오름차순을 기본으로 하므로, 내림차순으로 정렬하려면 'reverse' 옵션을 사용

```
>>> contact.sort(reverse=True)
>>> contact
['정약용', '이순신', '유관순', '김유신']
```

02. 리스트의 사용

IV. 리스트 조작 함수

실습 7-6

리스트에서 `sort()` 함수 활용하기

- ③ `sorted()` 함수는 인수로 사용한 `numbers` 리스트는 바꾸지 않고 다른 리스트 `alist`에 정렬 결과를 저장

```
>>> numbers = [78, 45, 90, 88, 62]
>>> alist = sorted(numbers)
>>> numbers                                # 항목이 정렬되지 않은 원본 리스트
[78, 45, 90, 88, 62]
>>> alist                                  # 정렬된 결과를 반환받은 리스트
[45, 62, 78, 88, 90]
```

02. 리스트의 사용

IV. 리스트 조작 함수

| 함수 | 설명 | 사용법 |
|-----------|--|---------------------|
| append() | 리스트 맨 뒤에 항목을 추가한다. | 리스트명.append(값) |
| pop() | 리스트 맨 뒤의 항목을 뺀다(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) |

02. 리스트의 사용

[실습] for 반복문을 통한 1차원 리스트 값 접근 [index 사용]

```
◆ for i in range(len(alist)):  #index 생성  
    print(alist[i])
```

```
alist = [3, 4, 2]  
for i in range(len(alist)):  
    print(alist[i])
```

```
alist = [3, 4, 2]  
for i in range(len(alist)) :  
    print(alist[i])
```

```
3  
4  
2
```

02. 리스트의 사용

[실습] for 반복문을 통한 1차원 리스트 값 접근 [리스트 직접 사용]

◆ for x in alist: #리스트 항목 값 반환
 print(x)

```
alist = [3, 4, 2]
for x in alist:
    print(x)
```

```
alist = [3, 4, 2]
for x in alist :
    print(x)
```

3
4
2

```
alist = [3, 4, 2]
for x in alist[1:3]:    #리스트 슬라이싱
    print(x)
```

```
alist = [3, 4, 2]
for x in alist[1:3]:
    print(x)
```

4
2

02. 리스트의 사용

[실습] for 반복문을 통한 2차원 리스트 값 접근 [index 사용]

◆ for i in range(len(alist)): #index 생성
 print(alist[i])

```
alist = [[0, 3, 2], [6, 1, 0], [8, 3, 6], [6, 9, 7]]  
for i in range(len(alist)):  
    print(alist[i])
```

```
alist = [[0, 3, 2], [6, 1, 0], [8, 3, 6], [6, 9, 7]]  
for i in range(len(alist)) :  
    print(alist[i])
```

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

```
alist = [[0, 3, 2], [6, 1, 0], [8, 3, 6], [6, 9, 7]]  
for i in range(len(alist)):    #행 접근  
    for j in range(len(alist[i])):    #열 접근  
        print(alist[i][j], end=' ')  
    print()
```

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


02. 리스트의 사용

[실습] for 반복문을 통한 2차원 리스트 값 접근 [리스트 직접 사용]

◆ for x in alist: #리스트 항목 값 반환
 print(x)

```
alist = [[0, 3, 2], [6, 1, 0], [8, 3, 6], [6, 9, 7]]  
  
for x in alist: :      #x는 리스트  
    print(x)
```

```
alist = [[0, 3, 2], [6, 1, 0], [8, 3, 6], [6, 9, 7]]  
for x in alist :  
    print(x)  
  
[0, 3, 2]  
[6, 1, 0]  
[8, 3, 6]  
[6, 9, 7]
```

```
alist = [[0, 3, 2], [6, 1, 0], [8, 3, 6], [6, 9, 7]]  
  
for x in alist:  
    for y in x:      #x는 리스트, y는 값  
        print(y, end=' ')  
    print()
```

```
alist = [[0, 3, 2], [6, 1, 0], [8, 3, 6], [6, 9, 7]]  
for x in alist:  
    for y in x:  
        print(y, end=' ')  
    print()  
  
0 3 2  
6 1 0  
8 3 6  
6 9 7
```

02. 리스트의 사용

[실습] for 반복문을 통한 2차원 리스트 값 접근 [리스트 직접 사용]

◆ for c1, c2, c3 in alist: #elements unpacking
 print(c1, c2, c3)

```
alist = [[0, 3, 2], [6, 1, 0], [8, 3, 6], [6, 9, 7]]  
for c1, c2, c3 in alist: #리스트 언팩킹  
    print(c1, c2, c3)
```

```
alist = [[0, 3, 2], [6, 1, 0], [8, 3, 6], [6, 9, 7]]  
for c1, c2, c3 in alist:  
    print(c1, c2, c3)
```

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

```
alist = [[0, 3, 2], [6, 1, 0], [8, 3, 6], [6, 9, 7]]  
for x in alist:  
    c1, c2, c3 = x #리스트 언팩킹  
    print(c1, c2, c3)
```

```
alist = [[0, 3, 2], [6, 1, 0], [8, 3, 6], [6, 9, 7]]  
for x in alist:  
    c1, c2, c3 = x  
    print(c1, c2, c3)
```

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

02. 리스트의 사용

여기서 잠깐

2차원 리스트

#elements unpacking

- 2차원 리스트를 for 반복문으로 참조할 때는 다음과 같이 2개의 변수를 사용할 수 있음

| | |
|---|-------|
| | 열 |
| | 'A' 3 |
| 행 | 'B' 5 |
| | 'C' 7 |

```
>>> multi = [['A', 3], ['B', 5], ['C', 7]]
>>> for x, y in multi :
    print(x, '=', y)
```

A = 3

B = 5

C = 7

그림 7-7 2차원 리스트 사용법

02. 리스트의 사용

V. 리스트 초기화

■ 1차원 리스트 초기화 방법

0. `arr = [0, 0, 0, 0, 0]`

I. `n = 5; arr = [0]*n;`

II. `n = 5; arr = [0 for i in range(n)];`

III. `n = 5; arr=[];` #만들고
`for i in range(n):`
`arr.append(0)` #추가하고

```
>>> n = 5
>>> arr = [0]*n
>>> arr
[0, 0, 0, 0, 0]
>>> arr = [0 for i in range(n)]
>>> arr
[0, 0, 0, 0, 0]
>>> arr = [i for i in range(n)]
>>> arr
[0, 1, 2, 3, 4]
>>> arr = []
>>> for i in range(n):
...     arr.append(i)
...
...
>>> arr
[0, 1, 2, 3, 4]
>>>
```

02. 리스트의 사용

V. 리스트 초기화

■ 2차원 리스트 초기화 방법

- 0. `arr = [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]`
- I. `row, col = (3, 4); arr = [[0]*col]*row;` #가급적 사용하지 말 것
- II. `row, col = (3, 4); arr = [[0 for j in range(col)] for i in range(row)];`
- III. `row, col = (3, 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
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]

arr = [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
arr
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
row, col = (3, 4); arr = [ [0]*col ]*row;
arr
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
row, col = (3, 4); arr = [ [0 for j in range(col)] for i in range(row)];
arr
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
row, col = (3, 4); arr=[];
for i in range(row):
    clist = []      #열 생성
    for j in range(col):
        clist.append(0) #열 값 추가
    arr.append(clist)

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

02. 리스트의 사용

[실습] 1차원 리스트 초기화

◆ 1차원 배열 리스트를 초기화 하시오.

- 열(col)의 크기는 키보드로 입력받아서 사용

col = int(input("열 개수는? "))

I. arr = [0]*col

arr

arr[0], arr[1] = (1, 2)

arr

sum = arr[0] + arr[1]

sum

II. arr = [i for i in range(n)]

arr[0], arr[1] = (1, 2)

arr

```
col = int(input("열 개수는? "))
```

```
열 개수는? 5
```

```
arr = [0]*col
```

```
arr[0], arr[1] = (1, 2)
```

```
arr
```

```
[1, 2, 0, 0, 0]
```

```
arr = [i for i in range(n)]
```

```
arr
```

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

```
arr[0], arr[1] = (1, 2)
```

```
arr
```

```
[1, 2, 2, 3, 4]
```

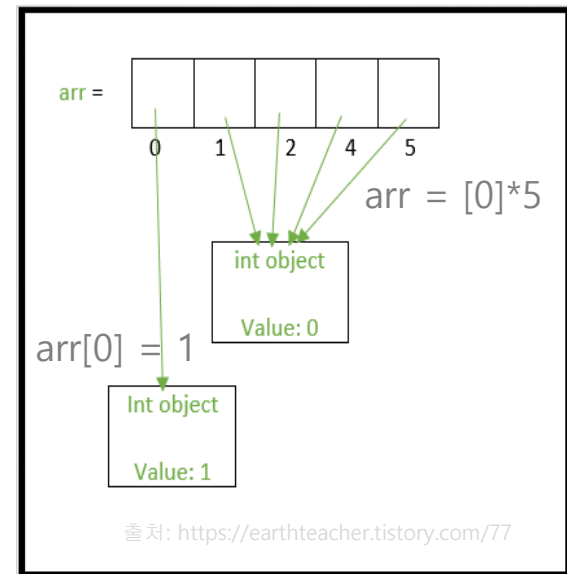
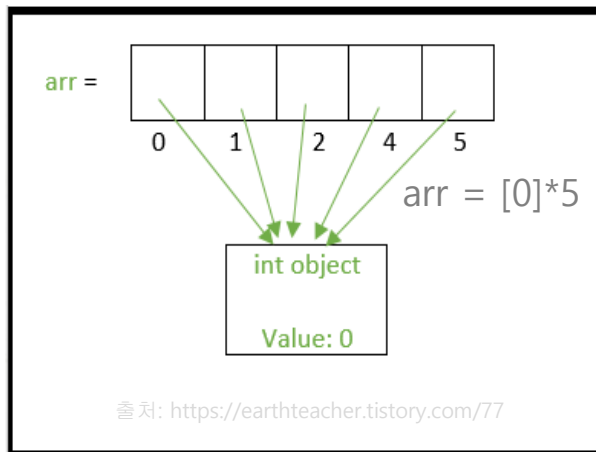
02. 리스트의 사용

[실습] 1차원 리스트 생성 방식 (Liked list)

◆ 1차원 배열 리스트를 초기화 하시오.

- 열(col)의 크기는 키보드로 입력받아서 사용

- `arr = [0]*5` #int value 0인 객체를 만든 후에 5개 열이 가르킴(pointing)
- `arr[0] = 1` #또 하나의 int value 1인 객체를 새롭게 만든 후에 `arr[0]`열이 가르킴



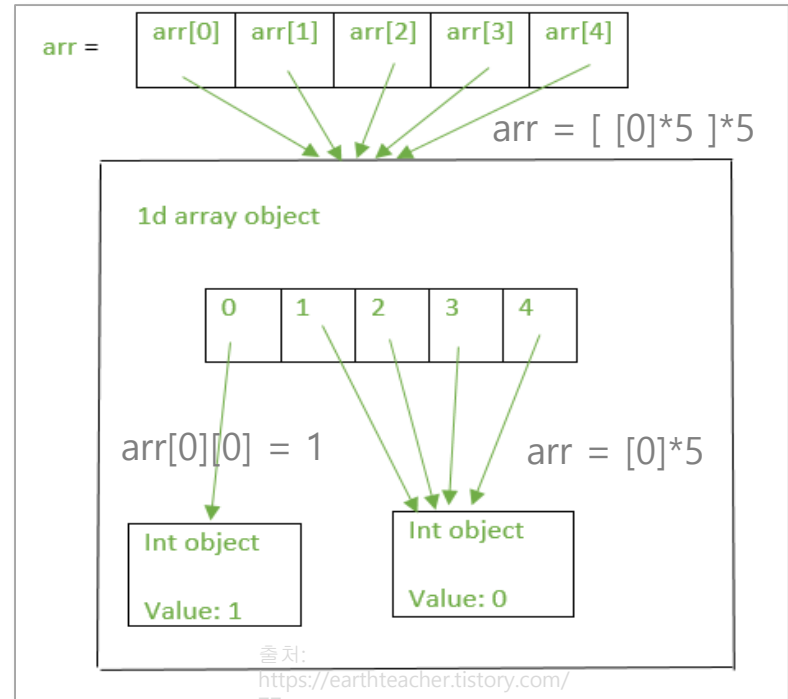
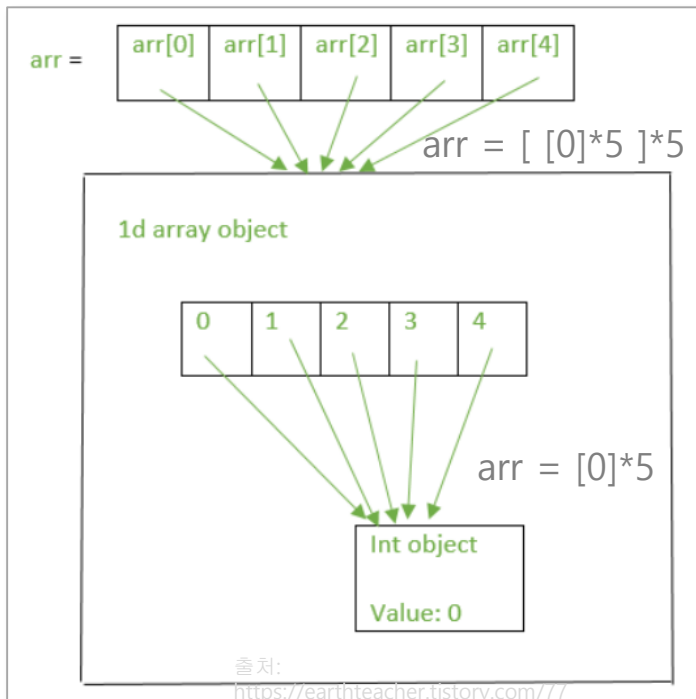
02. 리스트의 사용

[실습] 2차원 리스트 초기화 방식 (Liked list)

◆ 2차원 배열 리스트를 초기화 하시오.

- 행(row)와 열(col)의 크기는 키보드로 입력받아서 사용

- `arr = [[0]*5]*5` # `[0]*5` 를 만들어서, 그것을 각 행이 가르킴(pointing)
- `arr[0][0] = 1` # `[0]*5` 객체 안에 또 하나의 int value 1인 객체를 생성하여 `arr[][0]`가 가르키게 함
 - ✓ 따라서 모든 행의 `[0]`열의 값이 동일해지는 의도하지 않는 일이 발생함



02. 리스트의 사용

[실습] 2차원 리스트 초기화

◆ 2차원 배열 리스트를 초기화 하시오.

• 행(row)와 열(col)의 크기는 키보드로 입력받아서 사용

```
I. row = int(input("행 개수는? "))
    col = int(input("열 개수는? "))
    arr = [ [0]*col ]*row
    arr
    arr[0][0], arr[0][1] = (1, 2)
    arr  # [[1, 2, 0, 0], [1, 2, 0, 0], [1, 2, 0, 0]]

II. arr = [ [0 for j in range(col)] for i in range(row)]
    arr[0][0], arr[0][1] = (1, 2)
    arr  # [[1, 2, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

```
row = int(input("행 개수는? "))
행 개수는? 3
col = int(input("열 개수는? "))
열 개수는? 4
arr = [ [0]*col ]*row
arr
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
arr[0][0], arr[0][1] = (1, 2)
arr
[[1, 2, 0, 0], [1, 2, 0, 0], [1, 2, 0, 0]]

arr = [ [i for i in range(col)] for j in range(row)]
arr
[[0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3]]
arr[0][0], arr[0][1] = (1, 2)
arr
[[1, 2, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3]]
```

02. 리스트의 사용

[실습] 2차원 리스트 (순서번호로) 초기화

◆ 2차원 배열 리스트를 초기화 하시오.

• 행(row)와 열(col)의 크기는 키보드로 입력받아서 사용

```
row = int(input("행 개수는? "))
```

```
col = int(input("열 개수는? "))
```

```
# 0 대신 순번으로 초기화
```

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

```
arr
```

행 개수는? 3

열 개수는? 4

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

III. `arr = []`

```
for i in range(row):
```

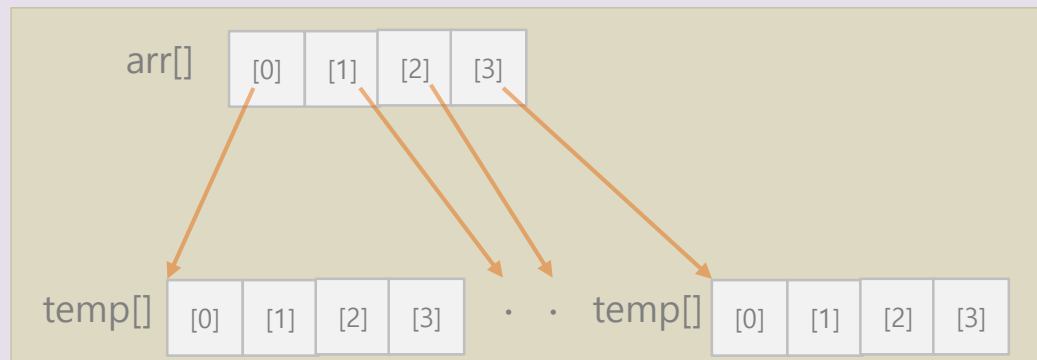
```
    temp = []
```

```
    for j in range(col):
```

```
        temp.append(col*i+j)
```

```
    arr.append(temp)
```

```
arr
```



02. 리스트의 사용

[실습] 1차원 리스트에서 가장 작은 값 찾기

Selection Sort

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

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

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

```
print(alist)
```

```
minx = 0  #최소값 항목 index
```

```
for i in range(1, len(alist)):
```

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

```
        minx = i
```

```
print("[%d] %d" %(minx, alist[minx]))
```

❖ Ch07_SelSort00.py

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

02. 리스트의 사용

[실습] 1차원 리스트에서 가장 작은 값을 찾아 위치 바꾸기 (1) Selection Sort

◆ 가장 작은 값을 찾아 그 항목과 0번째 값을 교환 하시오.

• `alist = [4, 3, 2, 6, 1, 0, 8, 3, 6, 6, 9, 7]`

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

```
print(alist)
```

```
minx = 0 #최소값 항목 index
```

```
for i in range(1, len(alist)):
```

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

```
        minx = i
```

```
alist[0], alist[minx] = alist[minx], alist[0] #항목 교환
```

```
print(alist)
```

❖ Ch07_SelSort01.py

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

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

02. 리스트의 사용

[실습] 1차원 리스트에서 가장 작은 값을 찾아 위치 바꾸기 (2) Selection Sort

◆ 가장 작은 값 순으로 리스트 항목을 재배치 하시오.

- `alist = [4, 3, 2, 6, 1, 0, 8, 3, 6, 6, 9, 7]`

- 계속해서 `i=1, 2, ...` 순으로 옮겨가며 그 이하 항목에서 가장 작은 값을 찾아 `i`번째와 교환

➤ 이러한 정렬 방식을 **Selection Sort**라고 한다.

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

❖ Ch07_SelSort02.py

```
print(alist)
```

```
#Selection Sort (오름차 순)
```

```
for s in range(len(alist)):
```

```
    minx = s          #최소값 항목 index를 교환 기준값 index로 시작
```

```
    for i in range(s+1, len(alist)):
```

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

```
            minx = i
```

```
    alist[s], alist[minx] = alist[minx], alist[s]
```

```
print(alist)
```

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

02. 리스트의 사용

[실습] 1차원 리스트에서 가장 작은 값을 오른쪽 끝에 위치 시키기

Bubble Sort

◆ Bubble Sort (내림차 순)

- 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

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

```
print(alist)
```

```
#Bubble Sort (내림차 순)
```

```
for s in range(len(alist), 0, -1): #가장 작은 값부터 오른쪽 끝에 배치하고자 하는 오른쪽 제어
```

```
    for i in range(s-1):
```

```
        if alist[i] < alist[i+1]: #맨 왼쪽부터 다음과 값 비교
```

```
            alist[i], alist[i+1] = alist[i+1], alist[i] #위치 교환
```

```
        print(alist)
```

```
print(alist)
```

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

02. 리스트의 사용

[실습] 1차원 리스트 출력 함수 prt_list() 만들기

◆ prt_list() 함수

- 전달하는 1차원 리스트를 오름차순 또는 내림차순으로 출력
- 오름차순은 reverse=False 매개변수로 호출, prt_list(alist) 처럼 생략 가능
- 내림차순은 reverse=True로 매개변수로 호출 > prt_list(alist, reverse=True)

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

❖ Ch07_PrtList00.py

```
def prt_list(alist, reverse=False): #출력 순서 기본값은 오름차순
```

```
    if reverse == False:
```

```
        for i in range(len(alist)):          #순방향 제어
```

```
            print(alist[i], end=' ')
```

```
    else:
```

```
        for i in range(len(alist)-1, 0-1, -1): #역방향 제어
```

```
            print(alist[i], end=' ')
```

```
    print()
```

```
prt_list(alist) # reverse의 기본값은 False
```

```
prt_list(alist, reverse=True)
```

02. 리스트의 사용

[과제-1] 1차원 리스트의 항목 값들을 정렬해서 출력해주는 **bub_sort()** 함수 완성

◆ Bubble sort 알고리즘을 활용하여 다음과 같은 함수를 완성하십시오.

- 함수명은 `bub_sort()`로 한다.
- 함수 호출은
 - [오름차순 정렬] `pub_sort(alist)` 또는 `pub_sort(alist, reverse=False)` 로 호출
 - [내림차순 정렬] `pub_sort(alist, reverse=True)` 로 호출

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

```
def bub_sort(alist, reverse=False): #출력 순서 기본값은 오름차순
```

```
    ## 완성해야 할 부분
```

```
#함수 호출
```

```
bub_sort(alist) # reverse의 기본값은 False
```

```
bub_sort(alist, reverse=True)
```


02. 리스트의 사용

[실습] 1차원 리스트 정렬하기 (제공 함수 사용)

◆ 1차원 리스트 항목들을 함수를 제공되는 함수나 메서드를 사용하여 정렬

• `alist = [4, 3, 2, 6, 1, 0, 8, 3, 6, 6, 9, 7]`

➤ `max(alist)`; `min(alist)`, `alist.sort()` 활용

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

```
print(alist)
```

```
#max(alist); min(alist) 사용
```

```
temp = []
```

```
for i in range(len(alist)):
```

```
    temp.append(max(alist)) #리스트에서 가장 큰 값 찾기
```

```
    alist.remove(max(alist)) #리스트에서 가장 큰 값 제거하기
```

```
alist = temp #리스트 복사
```

```
print(alist)
```

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

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

```
alist.sort(); print(alist)
```

```
alist.sort(reverse=True); print(alist)
```

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

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

02. 리스트의 사용

VI. 리스트 복사

- 얕은 복사(shallow copy)와 깊은 복사(deep copy)가 있다.
- **얕은 복사**는 복사 대상 리스트를 공유하는 형태로 **참조 방식**
- **깊은 복사**는 복사 대상 리스트와 똑같은 리스트를 생성하여 **항목을 복사**하는 방식

```
alist = [1, 2, 3, 4]
blist = alist    #얕은 복사 (리스트 참조)
blist           #alist를 참조
alist[1] = 20
alist; blist
```

```
alist = [1, 2, 3, 4]
blist = alist    #얕은 복사
blist
[1, 2, 3, 4]
alist[1] = 20
alist
[1, 20, 3, 4]
blist
[1, 20, 3, 4]
```

```
alist = [1, 2, 3, 4]
blist = alist[:]  #깊은 복사 (슬라이싱된 리스트 값들을 복사)
blist           #alist와 독립적
alist[1] = 20
alist; blist
```

```
alist = [1, 2, 3, 4]
blist = alist[:]  #깊은 복사
blist
[1, 2, 3, 4]
alist[1] = 20
alist
[1, 20, 3, 4]
blist
[1, 2, 3, 4]
```

02. 리스트의 사용

[실습] 2차원 리스트를 1차원 리스트로 복사하기

◆ 2차원 리스트 alist 값을 순서대로 1차원 리스트 arr로 복사를 하시오.

• alist = [[0, 3, 2], [6, 1, 0], [8, 3, 6], [6, 9, 7]]

```
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]
```

02. 리스트의 사용

[실습] 1차원 리스트를 2차원 리스트로 복사하기

◆ 1차원 리스트 alist 값을 순서대로 2차원 리스트 arr로 복사를 하시오.

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

```
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]]
```

02. 리스트의 사용

[실습] 1차원 리스트 초기화 후 값 저장 순서 반대로 바꾸기

◆ 1차원 리스트 초기화 후 값 저장 순서 반대로 바꾸시오.

- 열(col)의 크기를 키보드로 입력 받아서 사용
- 초기 값은 0부터 시작하여 1씩 증가하는 수 사용
- i-열 값과 (col-i-1) 열 값을 상호 교환하여 순서를 반대로 바꿈

```
col = int(input("열 개수는? "))
arr = [i for i in range(col)] #1차원배열 초기화(오름차순)
print(arr)
for i in range(len(arr)//2): #교환할 위치 index 생성
    arr[i], arr[-i-1] = arr[-i-1], arr[i] #위치 교환
print(arr)
```

```
for x in arr:
    i = arr.index(x) #교환할 위치 index 찾기
    if i >= col//2:
        break
    arr[i], arr[-i-1] = arr[-i-1], arr[i] #위치 교환
print(arr)
```

열 개수는? 10

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

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

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

02. 리스트의 사용

[실습] (중복없이) 랜덤한 값으로 1차원 리스트 초기화

◆ 1차원 배열 리스트를 중복 없는 랜덤한 값으로 초기화 하시오.

- 선택된 랜덤 수를 저장할 리스트의 크기는 3
- 선택되는 중복없는 랜덤 수는 0~9 중에 선택

```
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)
```

```
[6, 3, 2]
```

02. 리스트의 사용

[실습] (중복없이) 랜덤한 값으로 2차원 리스트 초기화

(1) 1차원 랜덤 씨드 리스트 생성

◆ 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

```
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)
print(arr)
```

행 개수는? 3
열 개수는? 4
[11, 1, 4, 8, 3, 7, 2, 5, 10, 6, 0, 9]

02. 리스트의 사용

[실습] (중복없이) 랜덤한 값으로 2차원 리스트 초기화

(2) 랜덤 씨드 리스트로 2차원 리스트 생성

◆ 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

1차원 씨드 리스트로 2차원 리스트 생성(랜덤한 1차원 씨드 리스트 사용)

```
randarr = []
```

```
for i in range(0, len(arr), col):
```

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

```
print(randarr)
```

행 개수는? 3

열 개수는? 4

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

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


03. 리스트의 활용

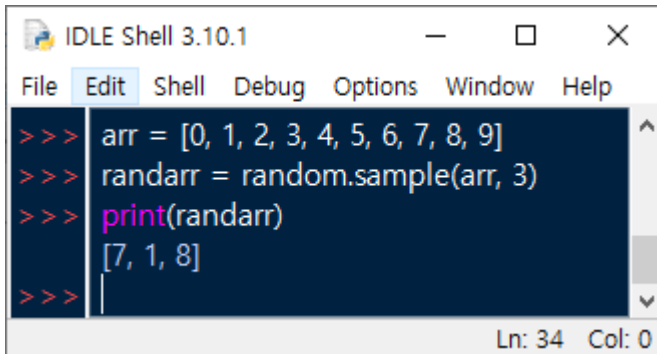
[과제-2] 숫자 야구 게임

◆ 랜덤한 3자리 숫자 맞추기 게임을 완성하시오.

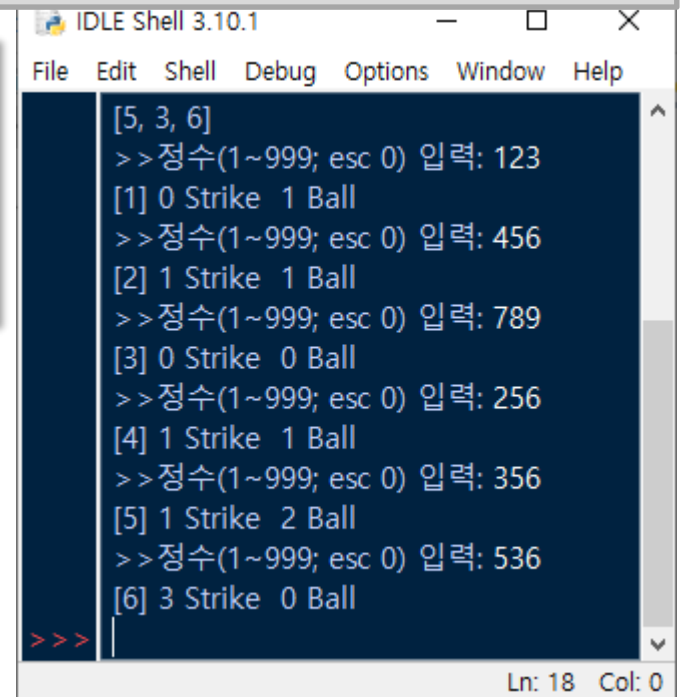
- 랜덤한 3자리 정수(1~999) 발생, 단 각 자리 수의 값은 같으면 안됨
- 게이머는 추측하는 3자리 값을 키보드로 입력 (맞출 때까지)
 - 각 위치(digit)에서 값이 같으면 Strike, 값은 존재하나 위치가 다르면 Ball로 처리
 - 매회 ">>%d Strike, %d Ball"로 결과 제공
- 게임 종료 조건 : 3 Strike 또는 입력 값 00

❖ 리스트는 정수형과 문자형 중에 한 방식 선택 필요

```
arr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] #정수형 리스트  
randarr = random.sample(arr, 3)  
print(randarr)
```



```
IDLE Shell 3.10.1  
File Edit Shell Debug Options Window Help  
>>> arr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
>>> randarr = random.sample(arr, 3)  
>>> print(randarr)  
[7, 1, 8]  
>>> |  
Ln: 34 Col: 0
```



```
IDLE Shell 3.10.1  
File Edit Shell Debug Options Window Help  
[5, 3, 6]  
>>정수(1~999; esc 0) 입력: 123  
[1] 0 Strike 1 Ball  
>>정수(1~999; esc 0) 입력: 456  
[2] 1 Strike 1 Ball  
>>정수(1~999; esc 0) 입력: 789  
[3] 0 Strike 0 Ball  
>>정수(1~999; esc 0) 입력: 256  
[4] 1 Strike 1 Ball  
>>정수(1~999; esc 0) 입력: 356  
[5] 1 Strike 2 Ball  
>>정수(1~999; esc 0) 입력: 536  
[6] 3 Strike 0 Ball  
>>> |  
Ln: 18 Col: 0
```

03. 리스트의 활용

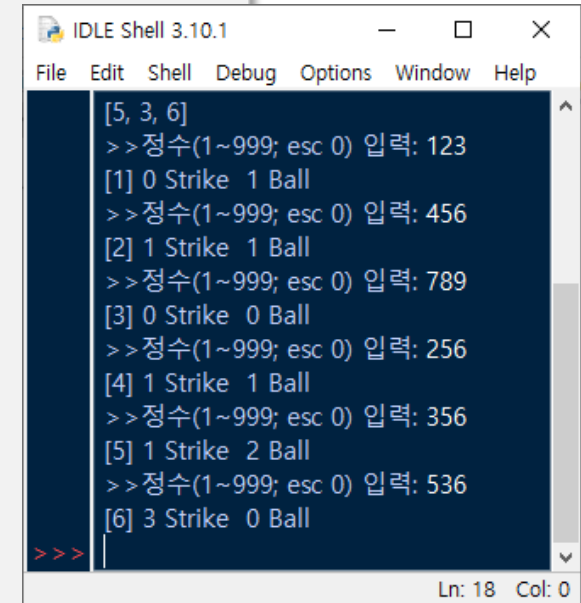
[과제-2] 숫자 야구 게임

◆ 랜덤한 3자리 숫자 맞추기 게임을 완성하시오.

- 각 위치(digit)에서 값이 같으면 Strike, 값은 존재하나 위치가 다르면 Ball로 처리

❖ 숫자형 리스트 사용 (예시) #입력된 3자리 숫자값들을 분리(split)하는 과정 필요

```
import random
arr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]    #중복방지 난수 발생용 씨드
strike = 0; ball = 0
randarr = []        #3자리 난수 리스트
guessarr = [0, 0, 0] #3자리 추측 리스트
bcnt = 0    #시도 횟수
randarr = random.sample(arr, 3) #랜덤 샘플링
while True :
    ...
    innum = int(input(">>정수(1~999; esc 00) 입력: "))
    #입력된 3자리 숫자에서 각 단위 자리수를 추출하는 스플릿 과정 필요
    guessarr[0] = innum // 100        #100자리
    guessarr[1] =                    #10자리
    guessarr[2] =                    #1자리
```



```
IDLE Shell 3.10.1
File Edit Shell Debug Options Window Help
[5, 3, 6]
>>정수(1~999; esc 0) 입력: 123
[1] 0 Strike 1 Ball
>>정수(1~999; esc 0) 입력: 456
[2] 1 Strike 1 Ball
>>정수(1~999; esc 0) 입력: 789
[3] 0 Strike 0 Ball
>>정수(1~999; esc 0) 입력: 256
[4] 1 Strike 1 Ball
>>정수(1~999; esc 0) 입력: 356
[5] 1 Strike 2 Ball
>>정수(1~999; esc 0) 입력: 536
[6] 3 Strike 0 Ball
>>>
```

03. 리스트의 활용

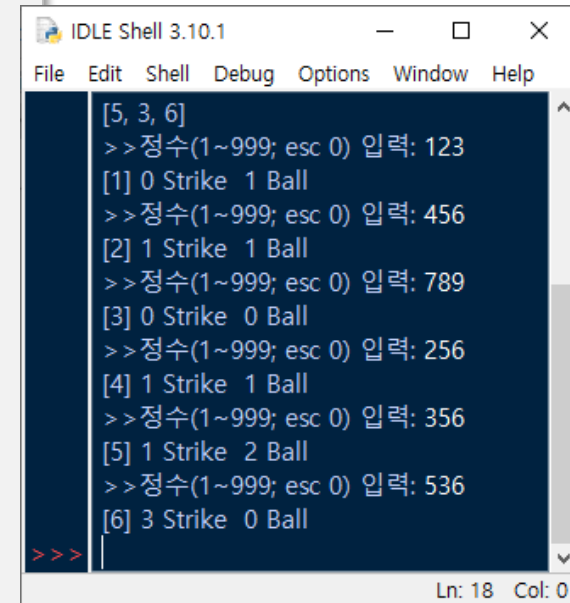
[과제-2] 숫자 야구 게임

◆ 랜덤한 3자리 숫자 맞추기 게임을 완성하시오.

- 각 위치(digit)에서 값이 같으면 Strike, 값은 존재하나 위치가 다른 Ball로 처리

❖ 문자형 리스트 사용 (예시)

```
import random
arr = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'] #중복방지 난수 발생용 씨드
strike = 0; ball = 0
randarr = ['0', '0', '0'] #3자리 난수 리스트
guessarr = ['0', '0', '0'] #3자리 추측 리스트
bcnt = 0 #시도 횟수
randarr = random.sample(arr, 3)
print(randarr)
while True :
    ...
    guessarr = list(input(">>3자리 수(001~999; esc 00) 입력: "))
    #입력된 3자리 문자값들이 한 문자씩 분리(split)되어 리스트 항목으로 저장
```



```
IDLE Shell 3.10.1
File Edit Shell Debug Options Window Help
[5, 3, 6]
>>정수(1~999; esc 0) 입력: 123
[1] 0 Strike 1 Ball
>>정수(1~999; esc 0) 입력: 456
[2] 1 Strike 1 Ball
>>정수(1~999; esc 0) 입력: 789
[3] 0 Strike 0 Ball
>>정수(1~999; esc 0) 입력: 256
[4] 1 Strike 1 Ball
>>정수(1~999; esc 0) 입력: 356
[5] 1 Strike 2 Ball
>>정수(1~999; esc 0) 입력: 536
[6] 3 Strike 0 Ball
>>>
```

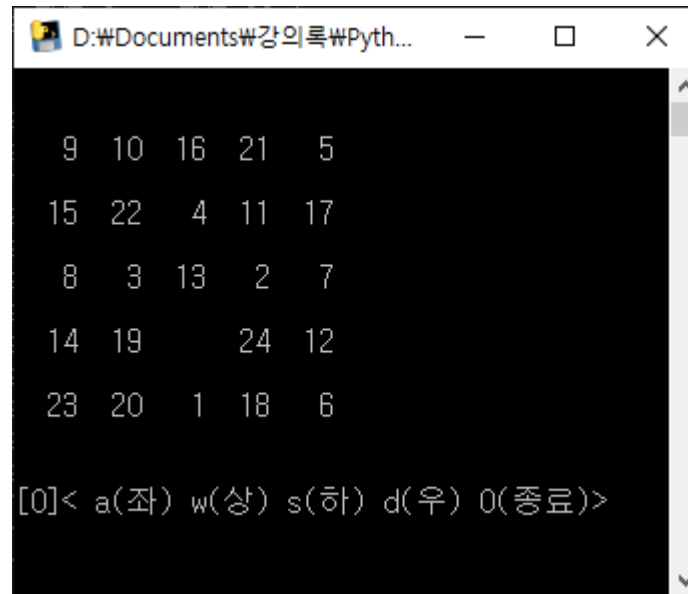
Ln: 18 Col: 0

01. 숫자 퍼즐 초기화

● [과제-3] nxn 숫자 퍼즐 게임 만들기

◆ 다음과 같은 숫자 퍼즐 정렬하기 게임을 완성 하시오.

- 퍼즐의 크기는 키보드로 입력 받음
- 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 아래와 같이 좌측 하단에 표시
→ 잘 못된 키 값이거나 이동할 수 없는 방향 값일 경우는 회 수에서 제외



```
D:\Documents\강의록\Pyth...  
  
9 10 16 21 5  
15 22 4 11 17  
8 3 13 2 7  
14 19 24 12  
23 20 1 18 6  
  
[0]< a(좌) w(상) s(하) d(우) 0(종료)>
```

03

리스트의 활용

03. 리스트의 활용

실습 7-7

리스트로 점수의 평균 구하기

code07-07.py

- 리스트를 이용하여 수강생 5명의 점수를 입력받아 평균을 출력하는 프로그램

①



그림 7-15 5명의 점수 입력 후 평균 구하기

- ② 리스트의 합계를 구하는 함수 **sum()**과 리스트 항목의 개수를 구하는 **len()**을 사용해서 간단하게 평균을 구할 수 있음

```
01 numbers = []
02
03 for _ in range(5):
04     numbers.append(int(input("점수 입력 : ")))
05
06 print("평균 =", sum(numbers) / len(numbers))
```

- ③ 프로그램 실행하고 결과 확인

03. 리스트의 활용

실습 7-8

도서 목록 만들기

code07-08.py

- 읽은 도서명을 모두 입력하면 정렬해서 가나다순으로 출력하는 프로그램



그림 7-17 도서 목록 만들고 정렬하기

- ② 무한 반복되는 입력 과정을 종료하기 위해 입력 값이 빈 문자열인지 판단하는 if 문을 사용

```
01 book = ''                # 입력하는 도서명을 저장하는 변수
02 bookList = []            # 도서 목록을 저장하는 리스트
03 number = 0               # 출력할 때 표시할 순서 번호
04
05 print("입력을 종료하려면 [Enter] 키를 누르세요.")
06 print('=' * 30)
07
08 while True :
09     book = input("도서명 입력 : ")
10     if book == '' :        # 도서명을 입력하지 않고 [Enter]를 눌렀을 때, 빈 문자열로 판단
```

03. 리스트의 활용

실습 7-8

도서 목록 만들기

code07-08.py

- ② 무한 반복되는 입력 과정을 종료하기 위해 입력 값이 빈 문자열인지 판단하는 if 문을 사용

```
11         break
12     bookList.append(book)
13
14     bookList.sort()
15
16     print('=' * 30)
17     for b in bookList :
18         number += 1
19         print(number, ':', b)
```

- ③ 결과 출력

입력을 종료하려면 [Enter] 키를 누르세요.

```
=====
도서명 입력 : 여행의 이유
도서명 입력 : 소년으로
도서명 입력 : 희랍인 조르바
도서명 입력 : 세 여자
도서명 입력 : 아픔이 길이 되려면
도서명 입력 : 관찰의 인문학
도서명 입력 :
```

```
=====
1 : 관찰의 인문학
2 : 세 여자
3 : 소년으로
4 : 아픔이 길이 되려면
5 : 여행의 이유
6 : 희랍인 조르바
```


03. 리스트의 활용

실습 7-9

스크린 세이버 수정하기

code07-09.py

- ① **choice()**는 리스트에서 하나의 항목을 랜덤으로 선택하는 메소드

```
import random
colorList = ['red', 'yellow', 'green', 'orange', 'blue']
random.choice(colorList)
```

- ②
- ```
01 from turtle import *
02 from random import *
03
04 x, y, radius = 0, 0, 0
05 colorList=['red', 'yellow', 'green', 'orange', 'blue', 'violet',
06 'tan', 'brown', 'navy', 'cyan']
07
08 setup(1200, 800) # 창 크기
09 bgcolor("black") # 배경 색상
10 speed(0) # 그리기 최고 속도
11
12 for i in range(30):
13 x = randint(-500, 500)
14 y = randint(-400, 300)
15 radius = randint(80, 130)
```

## 03. 리스트의 활용

실습 7-9

스크린 세이버 수정하기

code07-09.py

```
② 16 penup()
 17 goto(x, y)
 18 pendown()
 19 color(choice(colorList)) # 리스트의 색상 하나를 선택해서 채우기 색으로 설정
 20 begin_fill()
 21 circle(radius)
 22 end_fill()
```

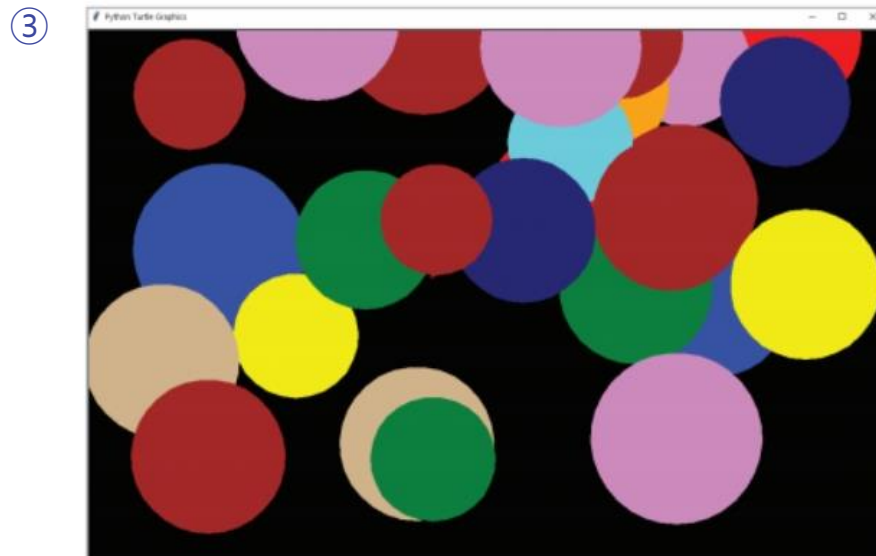


그림 7-19 스크린 세이버 수정 결과

### 03. 리스트의 활용

실습 7-10

버킷 리스트 만들기

code07-10.py

- ① 추가(1)나 삭제(2) 메뉴를 선택하면 리스트 추가나 삭제 작업 후, 목록을 출력  
메뉴를 잘못 입력했을 때 다시 선택할 수 있도록 처리하고, 종료(0) 메뉴를 선택하  
는 경우 프로그램을 종료

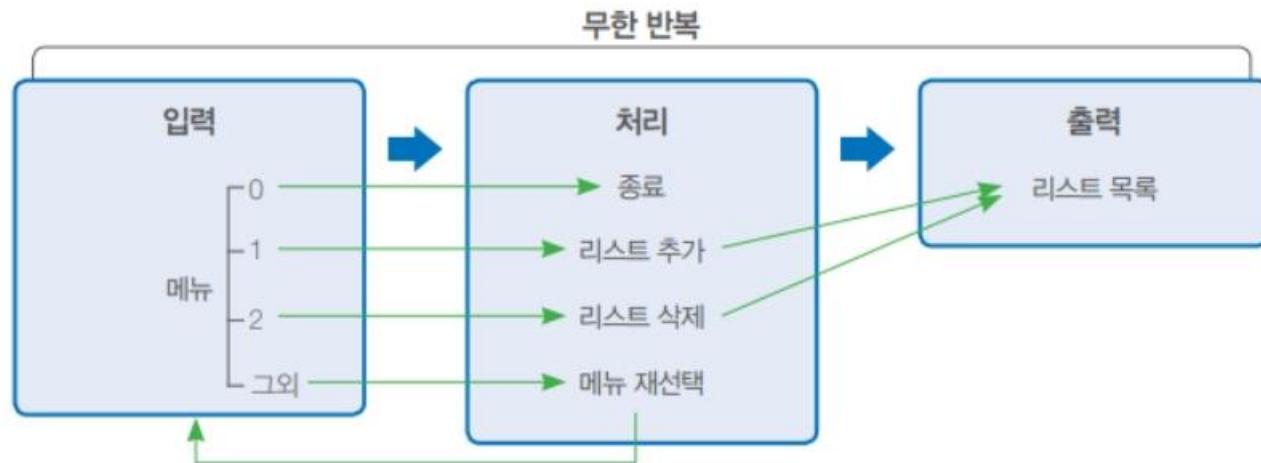


그림 7-20 버킷 리스트 프로그램의 실행 순서

## 03. 리스트의 활용

실습 7-10

버킷 리스트 만들기

code07-10.py

```
② 01 menu = 0
 02 bucket = []
 03
 04 while True :
 05 menu = int(input("메뉴 선택(1.추가 2.삭제 0.종료) : "))
 06 if menu == 1 :
 07 bucket.append(input("추가할 내용 : "))
 08 elif menu == 2 :
 09 bucket.remove(input("삭제할 내용 : "))
 10 elif menu == 0 :
 11 print("프로그램을 종료합니다.")
 12 break
 13 else :
 14 print("메뉴 선택 오류입니다. 다시 선택하세요.")
 15 continue
 16 print('*' * 30)
 17 for x in bucket :
 18 print(x)
 19 print('*' * 30)
```

## 03. 리스트의 활용

실습 7-10

버킷 리스트 만들기

code07-10.py

### ③ 프로그램을 실행하고 버킷 리스트를 추가

```
메뉴 선택(1.추가 2.삭제 0.종료) : 1
```

```
추가할 내용 : 번지 점프
```

```

```

```
번지 점프
```

```

```

```
메뉴 선택(1.추가 2.삭제 0.종료) : 1
```

```
추가할 내용 : 혼자 여행
```

```

```

```
번지 점프
```

```
혼자 여행
```

```

```

```
메뉴 선택(1.추가 2.삭제 0.종료) : 1
```

```
추가할 내용 : 독서 백권
```

```

```

```
번지 점프
```

```
혼자 여행
```

```
독서 백권
```

## 03. 리스트의 활용

실습 7-10

버킷 리스트 만들기

code07-10.py

### ④ 버킷 리스트에 저장된 항목을 삭제

```
메뉴 선택(1.추가 2.삭제 0.종료) : 2
```

```
삭제할 내용 : 혼자 여행
```

```

```

```
번지 점프
```

```
독서 백권
```

```

```

### ⑤ 메뉴를 잘못 입력하거나 프로그램 종료 메뉴도 선택

```
메뉴 선택(1.추가 2.삭제 0.종료) : 4
```

```
메뉴 선택 오류입니다. 다시 선택하세요.
```

```
메뉴 선택(1.추가 2.삭제 0.종료) : 0
```

```
프로그램을 종료합니다.
```

```
>>>
```

# Thank You !

[Python]