BIND SOFT

리스트와 반복문

BIND SOFT ㈜바인드소프트

BIND SOFT

목차

시작하기 전에

[핵심 키워드] 리스트, 요소, 인덱스, for 반복문

[핵심 포인트] 여러 개의 값을 나타낼 수 있게 해주는 리스트, 딕셔너리 등의 자료형도 존재한다. 이번 절에서는 리스트에 대해 알아보고, 이러한 자료가 반복문에 의해 어떻게 활용되는지 살펴본다.

시작하기 전에

- 리스트 (list)
  - 여러 가지 자료를 저장할 수 있는 자료
  - 자료들을 모아서 사용할 수 있게 해 줌
  - 대괄호 내부에 자료들 넣어 선언

```
>>> array = [273, 32, 103, "문자열", True, False]
>>> print(array)
[273, 32, 103, '문자열', True, False]
```

리스트 선언하고 요소에 접근하기

- 요소 (element)
  - 리스트의 대괄호 내부에 넣는 자료

```
[요소, 요소, 요소...]
```

```
>>> [1, 2, 3, 4]                          # 숫자만으로 구성된 리스트
[1, 2, 3, 4]
>>> ["안", "녕", "하", "세", "요"]          # 문자열만으로 구성된 리스트
['안', '녕', '하', '세', '요']
>>> [273, 32, 103, "문자열", True, False]   # 여러 자료형으로 구성된 리스트
[273, 32, 103, '문자열', True, False]
```

리스트 선언하고 요소에 접근하기

- 리스트 내부의 요소 각각 사용하려면 리스트 이름 바로 뒤에 대괄호 입력 후 자료의 위치 나타내는 숫자 입력

```
list_a = [273, 32, 103, "문자열", True, False]
```

list_a	273	32	103	문자열	True	False
	[0]	[1]	[2]	[3]	[4]	[5]

  - 인덱스 (index)

    - 대괄호 안에 들어간 숫자

리스트 선언하고 요소에 접근하기

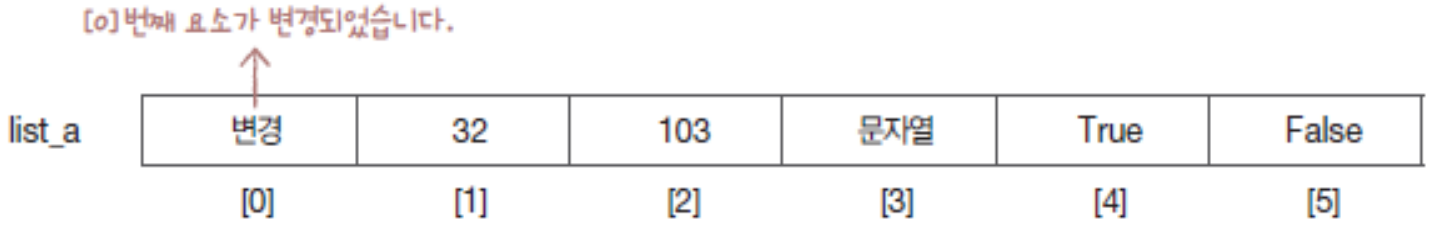
```
>>> list_a = [273, 32, 103, "문자열", True, False]
>>> list_a[0]
273
>>> list_a[1]
32
>>> list_a[2]
103
>>> list_a[1:3]
[32, 103]
```

- 결과로 [32, 103] 출력

리스트 선언하고 요소에 접근하기

- 리스트 특정 요소를 변경할 수 있음

```
>>> list_a = [273, 32, 103, "문자열", True, False]
>>> list_a[0] = "변경"
>>> list_a
['변경', 32, 103, '문자열', True, False]
```



리스트 선언하고 요소에 접근하기

- 대괄호 안에 음수 넣어 뒤에서부터 요소 선택하기

```
>>> list_a = [273, 32, 103, "문자열", True, False]
>>> list_a[-1]
False
>>> list_a[-2]
True
>>> list_a[-3]
'문자열'
```

273	32	103	문자열	True	False
[-6]	[-5]	[-4]	[-3]	[-2]	[-1]

리스트 선언하고 요소에 접근하기

- 리스트 접근 연산자를 이중으로 사용할 수 있음

```
>>> list_a = [273, 32, 103, "문자열", True, False]
>>> list_a[3]
'문자열'
>>> list_a[3][0]
'문'
```

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

리스트 선언하고 요소에 접근하기

- 리스트에서의 IndexError 예외
  - 리스트의 길이 넘는 인덱스로 요소에 접근하려는 경우 발생

```
>>> list_a = [273, 32, 103]
>>> list_a[3]
```

오류

```
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
IndexError: list index out of range
```

리스트 연산자: 연결(+), 반복(*), len()

- 예시 – 리스트 연산자

```
01  # 리스트를 선언합니다.
02  list_a = [1, 2, 3]
03  list_b = [4, 5, 6]
04
05  # 출력합니다.
06  print("# 리스트")
07  print("list_a =", list_a)
```

리스트 연산자: 연결(+), 반복(*), len()

```
08  print("list_b =", list_b)
09  print()
10
11  # 기본 연산자
12  print("# 리스트 기본 연산자")
13  print("list_a + list_b =", list_a + list_b)
14  print("list_a * 3 =", list_a * 3)
15  print()
16
17  # 함수
18  print("# 길이 구하기")
19  print("len(list_a) =", len(list_a))
```

실행결과

```
# 리스트
list_a = [1, 2, 3]
list_b = [4, 5, 6]

# 리스트 기본 연산자
list_a + list_b = [1, 2, 3, 4, 5, 6]
list_a * 3 = [1, 2, 3, 1, 2, 3, 1, 2, 3]

# 길이 구하기
len(list_a) = 3
```

리스트 연산자: 연결(+), 반복(*), len()

- 13행에서 문자열 연결 연산자 사용해 2, 3행과 7, 8행에서 선언 및 출력된 list_a와 list_b의 자료 연결

- 14행에서 문자열 반복 연산자 사용해 list_a의 자료 3번 반복

- 19행에서 len() 함수로 list_a에 들어있는 요소의 개수 구함

리스트에 요소 추가하기: append, insert

- append() 함수
  - 리스트 뒤에 요소를 추가

```
리스트명.append(요소)
```

- insert() 함수
  - 리스트 중간에 요소를 추가

```
리스트명.insert(위치, 요소)
```

리스트에 요소 추가하기: append, insert

- 예시

```
01  # 리스트를 선언합니다.
02  list_a = [1, 2, 3]
03
04  # 리스트 뒤에 요소 추가하기
05  print("# 리스트 뒤에 요소 추가하기")
06  list_a.append(4)
07  list_a.append(5)
08  print(list_a)
09  print()
10
11  # 리스트 중간에 요소 추가하기
12  print("# 리스트 중간에 요소 추가하기")
13  list_a.insert(0, 10)
14  print(list_a)
```

실행결과

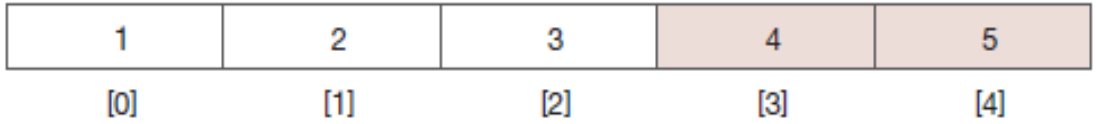
```
# 리스트 뒤에 요소 추가하기
[1, 2, 3, 4, 5]

# 리스트 중간에 요소 추가하기
[10, 1, 2, 3, 4, 5]
```

리스트에 요소 추가하기: append, insert

- 6 및 7행 실행 결과

```
list_a.append(4)
list_a.append(5)
```

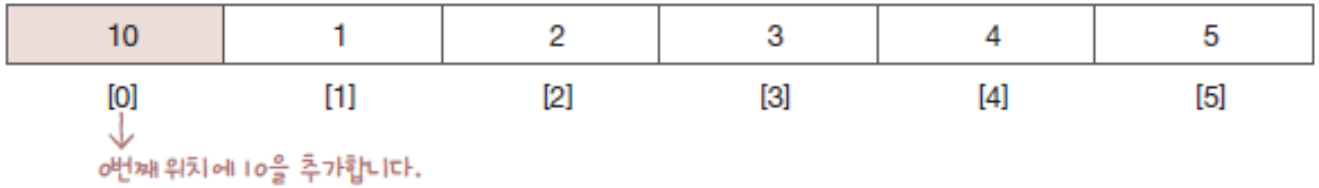


- 13행 실행 결과

```
list_a.insert(0, 10)
```

삽입할 값

삽입할 위치



리스트에 요소 추가하기: append, insert

- extend() 함수
  - 원래 리스트 뒤에 새로운 리스트의 요소 모두 추가
  - 매개변수로 리스트 입력

```
>>> list_a = [1, 2, 3]
>>> list_a.extend([4, 5, 6])
>>> print(list_a)
[1, 2, 3, 4, 5, 6]
```

리스트에 요소 추가하기: append, insert

- 리스트 연결 연산자와 요소 추가의 차이
  - 리스트 연결 연산자 사용하면 결과상 원본에 변화는 없음

```
>>> list_a = [1, 2, 3]
>>> list_b = [4, 5, 6]
>>> list_a + list_b  ⟶ 리스트 연결 연산자로 연결하니,
[1, 2, 3, 4, 5, 6]  ⟶ 실행결과로 [1, 2, 3, 4, 5, 6]이 나왔습니다.
>>> list_a  ⟶ list_a와 list_b에는 어떠한 변화도 없습니다(비파괴적 처리).
[1, 2, 3]
>>> list_b
[4, 5, 6]
```

리스트에 요소 추가하기: append, insert

- extend() 함수 사용할 경우

```
>>> list_a = [1, 2, 3]
>>> list_b = [4, 5, 6]
>>> list_a.extend(list_b)  ⟶ 실행결과로 아무 것도 출력하지 않았습니다.
>>> list_a  ⟶ 앞에 입력했던 list_a 자체에 직접적인 변화가 있습니다(파괴적 처리).
[1, 2, 3, 4, 5, 6]
>>> list_b
[4, 5, 6]
```

리스트에 요소 제거하기

- 인덱스로 제거하기: del 키워드, pop() 함수

```
del 리스트명[인덱스]
```

```
리스트명.pop(인덱스)
```

```
01  list_a = [0, 1, 2, 3, 4, 5]
02  print("# 리스트의 요소 하나 제거하기")
03
04  # 제거 방법[1] - del
05  del list_a[1]
06  print("del list_a[1]:", list_a)
07
08  # 제거 방법[2] - pop()
09  list_a.pop(2)
10  print("pop(2):", list_a)
```

실행결과

```
# 리스트의 요소 하나 제거하기
del list_a[1]: [0, 2, 3, 4, 5]
pop(2): [0, 2, 4, 5]
```

리스트에 요소 제거하기

- 5행 실행하면 자료에서 1 제거

```
del list_a[1]
```

0	2	3	4	5
[0]	[1]	[2]	[3]	[4]

- 9행에서 2번째 요소인 3 제거

```
list_a.pop(2)
```

0	2	4	5
[0]	[1]	[2]	[3]

리스트에 요소 제거하기

- del 키워드 사용할 경우 범위 지정해 리스트 요소를 한꺼번에 제거 가능

```
>>> list_b = [0, 1, 2, 3, 4, 5, 6]
>>> del list_b[3:6]
>>> list_b
[0, 1, 2, 6]
```

- 범위 한 쪽을 입력하지 않으면 지정 위치 기준으로 한쪽을 전부 제거

```
>>> list_c = [0, 1, 2, 3, 4, 5, 6]
>>> del list_c[:3]
>>> list_c
[3, 4, 5, 6]
```

리스트에 요소 제거하기

- 값으로 제거하기: remove() 함수
  - 특정 값을 지정하여 제거

```
리스트.remove(값)
```

```
>>> list_c = [1, 2, 1, 2]     # 리스트 선언하기
>>> list_c.remove(2)          # 리스트의 요소를 값으로 제거하기
>>> list_c
[1, 1, 2]
```

리스트에 요소 제거하기

- 모두 제거하기 : clear() 함수
  - 리스트 내부의 요소를 모두 제거

```
리스트.clear()
```

```
>>> list_d = [0, 1, 2, 3, 4, 5]
>>> list_d.clear()
>>> list_d
[] ⟶ 요소가 모두 제거되었습니다.
```

리스트 내부에 있는지 확인하기 : in/not in 연산자

- in 연산자
  - 특정 값이 리스트 내부에 있는지 확인

```
값 in 리스트
```

```
>>> list_a = [273, 32, 103, 57, 52]
>>> 273 in list_a
True
>>> 99 in list_a
False
>>> 100 in list_a
False
>>> 52 in list_a
True
```

리스트 내부에 있는지 확인하기 : in/not in 연산자

- not in 연산자
  - 리스트 내부에 해당 값이 없는지 확인

```
>>> list_a = [273, 32, 103, 57, 52]
>>> 273 not in list_a
False
>>> 99 not in list_a
True
>>> 100 not in list_a
True
>>> 52 not in list_a
False
>>> not 273 in list_a
False
```

for 반복문

- 반복문
  - 컴퓨터에 반복 작업을 지시

```
print("출력")
print("출력")
print("출력")
print("출력")
print("출력")
```

```
for i in range(100):
    print("출력")
```

→ 반복에 사용할 수 있는 자료

for 반복문 : 리스트와 함께 사용하기

- 문자열, 리스트, 딕셔너리 등과 조합하여 for 반복문을 사용

```
for 반복자 in 반복할 수 있는 것:
    코드
```

```
01  # 리스트를 선언합니다.
02  array = [273, 32, 103, 57, 52]
03
04  # 리스트에 반복문을 적용합니다.
05  for element in array:
06      # 출력합니다.
07      print(element)
```

실행결과

```
273
32
103
57
52
```

키워드로 정리하는 핵심 포인트

- **리스트** : 여러 가지 자료를 저장할 수 있는 자료형
- **요소** : 리스트 내부에 있는 각각의 내용을 의미
- **인덱스** : 리스트 내부에서 값의 위치를 의미
- **for 반복문** : 특정 코드를 반복해서 실행할 때 사용하는 기본 구문

확인문제

- list_a = [0, 1, 2, 3, 4, 5, 6, 7] 입니다. 다음 표의 함수들을 실행했을 때 list_a의 결과가 어떻게 나오는지 적어보세요

함수	list_a의 값
list_a.extend(list_a)	
list_a.append(10)	
list_a.insert(3, 0)	
list_a.remove(3)	
list_a.pop(3)	
list_a.clear( )	

확인문제

- 다음 반복문 내부에 if 조건문의 조건식을 채워서 100 이상의 숫자만 출력하게 만들어보세요.

```
numbers = [273, 103, 5, 32, 65, 9, 72, 800, 99]

for number in numbers:
    if 
        print("- 100 이상의 수:", number)
```

실행결과

```
- 100 이상의 수: 273
- 100 이상의 수: 103
- 100 이상의 수: 800
```

확인문제

- 다음 빈칸을 채워서 실행결과처럼 숫자를 하나하나 모두 출력해보세요

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

실행결과

```
1
2
3
4
5
6
7
8
9
```