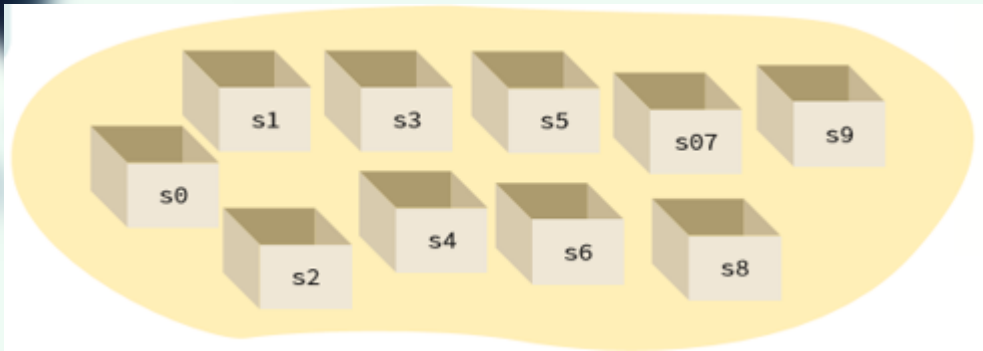
제8장
리스트-1

1. 리스트란?

1) 리스트란?

- 리스트는 이미 앞선 강의에서 몇 번 다룬 적이 있다. 이번 장에서는 리스트에 대해서 자세하게 보는 장이다. 예를 들어서 학생 10명의 성적 평균을 계산한다고 가정하자. 평균을 계산하려면 먼지 각 학생들의 성적을 읽어서 어딘가에 저장하여야 한다. 데이터를 저장할 수 있는 곳은 변수이다. 학생이 10명이므로 10개의 변수가 필요할 것이다.

- 만약 학생이 30명이라면 어떻게 해야 할까? 위의 방법대로라면 30개의 정수 변수를 생성하고 성적을 저장하여야 한다. 만약 100명이라면, 아니 10000명이라면 어떻게 할 것인가? 이런 식으로 변수를 일일이 생성하다가는 프로그래머의 생활이 아주 힘들어질 것이다. 따라서 다른 방법이 필요하다.



1. 리스트란?

1) 리스트란?

- 쉽게 대량의 데이터를 저장할 수 있는 공간을 만들 수 있어야 하고 이 이 데이터들을 손쉽게 처리할 수 있는 방법이 필요하다. 그래서 탄생하게 된 것이 리스트이다. 리스트를 사용하면 대량의 데이터를 효율적이고 간편하게 치리할 수 있다. 리스트(list)는 [ ] 안에 값을 여러 개의 데이터가 저장되어 있는 장소이다. 리스트는 아래와 같이 [ ]안에 값을 나열하면 생성된다.

```
리스트 = [ 값1 , 값2 , ...]
```

- 예를 들어서 10개의 정수를 저장하고 있는 리스트를 생성하면 다음과 같다.

```
scores  = [ 32, 56, 64, 72, 12, 37, 98, 77, 59, 69]
```

초기값이 있으면 위와 같이 생성하면 되지만 공백 리스트를 생성한 후에 사용자로부터 값을 받아서 리스트에 추가하려면 append() 메소드를 사용한다.

```
scores = [ ]
for i in range(10):
        scores.append(int(input("성적을 입력하시오:")))
print(scores)
```

1. 리스트란?

1) 리스트란?

- 특히 리스트가 꼭 필요한 경우는 서로 관련된 데이터를 차례로 접근하여서 처리하고 싶은 경우이다. 만약 관련된 데이터들이 서로 다른 이름을 사용하고 있다면 이들 이름을 일일이 기억해야 할 것이다. 그러나 하나의 이름을 공유하고 단지 번호만 다를 뿐이라면 아주 쉽게 기억할 수 있고 편리하게 사용할 수 있다. 리스트은 근본적으로 데이터들에게 하나하나 이름을 붙이지 않고 전체 집단에 하나의 이름을 부여한 다음, 숫자로 된 번호를 통하여 각각의 데이터에 접근하는 방법이다.

- 파이썬에서는 다양한 종류의 데이터를 하나의 리스트 안에 함께 저장할 수 있다. 이 부분은 이미 앞서서 강의를 한 적이 있다.

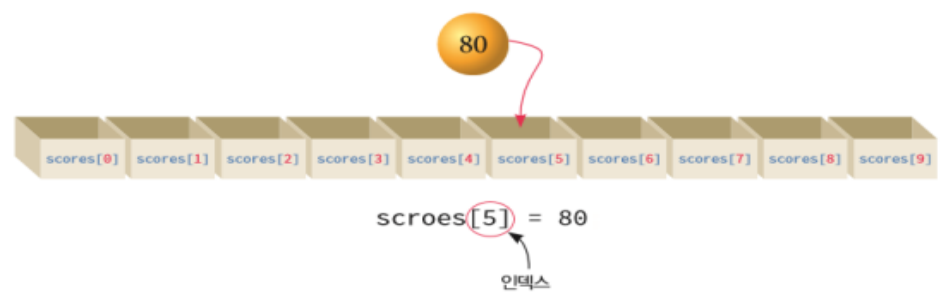
```
myList = [1, "computer", 3. 4]
myList = ["apple", [8, 4, 5]]
```

- 파이썬의 리스트는 다른 언어의 배열(array)과 유사하다.하지만 배열의 크기는 고정되어 있는 반면에 리스트의 크기는 가변적이다. 즉 요소의 개수에 따라서 커지거나 작아질 수 있다. 또 배열은 같은 데이터 타입만 저장할 수 있지만 리스트에는 다양한 종류의 데이터 타입형태를 섞어서 저장할 수 있다. 하여 파이썬의 리스트가 배열보다 훨씬 사용하기 편하다.

1. 리스트란?

2) 리스트 요소 접근하기

- 앞에서 리스트를 어떻게 생성하는지를 살펴보았다. 리스트에 저장된 데이터들을 리스트 요소(array element)라고 한다. 그렇다면 리스트 요소들은 어떤 식으로 접근 해야 하는가? 리스트의 요소에는 번호가 붙어 있는데 이것을 인덱스(index, 첨자)라고 칭한다. 리스트의 이름을 쓰고 괄호 [] 안에 번호를 표시하면 리스트 요소가 된다. 예를 들어서 리스트의 이름이 scores라면 리스트 요소는 scores[0], scores[1], scores[2], …. scores[9]으로 표시된다. 예를 들어서 scores 리스트에서 번호가 5인 요소에 접근하려면 scores[5] 와 같이 적어주면 되는 것이다.



- 유효한 인덱스의 범위는 0에서 (리스트 크기 -1)까지이다.
- 리스트 요소는 변수와 100% 동일하다. 리스트 요소에 값을 저장할 수 있고 리스트 요소에 저장된 값을 꺼낼 수도 있다.

```
scores[0] = 80          # 0번째 요소에 80을 저장함.
scores[1] = scores[0]   # 0번째 요소의 값을 1번째 요소로 복사함.
```

```
scores[i] = 10      # i는 정수 변수
scores[i+2] = 20    # 수식이 인덱스가 된다.
```

1. 리스트란?

2) 리스트 요소 접근하기

- 리스트 요소에 인덱스 범위를 확인하고 값을 저장하는 코드는 다음과 같다.

```
if i >= o and i < len(scores) :
        scores[i] = number
```

3) 리스트 순회하기

- 리스트에 있는 요소들을 순서대로 방문하는 작업은 아주 많이 나타난다. 기본적으로 2가지의 방법이 있다. 첫 번째 방법은 인덱스를 사용하여 방문하는 방법이다. 예를 들어서 크기가 10인 scores 리스트를 가정하자. 변수를 0에서 시작하여 하나씩 증가시키면서 거기에 해당되는 리스트 요소를 방문하는 방법이다.

```
for i in range (len(scores)):
        print(i, score[i])
```

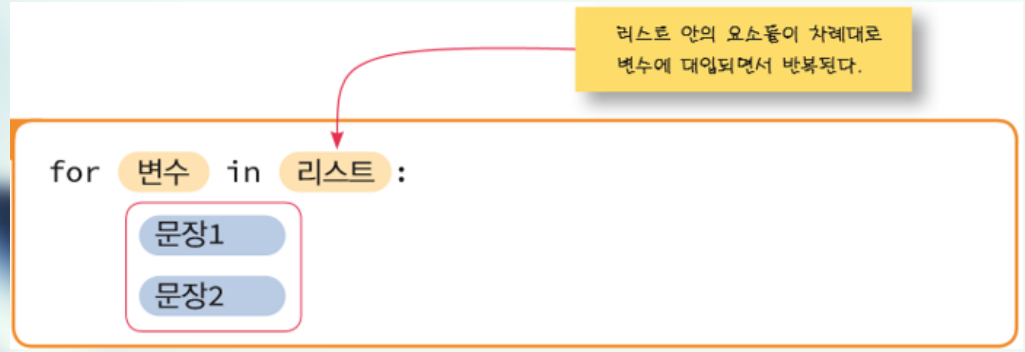
- 여기서 len(scores)는 리스트의 크기를 반환하므로 10이다. 변수 i는 0, 1, 2, 3, … 9와 같이 변경되고 scores[i]는 그 번호에 해당되는 리스트 요소가 된다. 만약 리스트 요소를 변경할 필요가 있다면 이것이 유일한 방법이다.

```
for i in range (len(scores)):
        score[i] = i*10
```

1. 리스트란?

3) 리스트 순회하기

- 단순히 리스트 요소의 값을 알고 싶은 경우에는 아래와 같은 형식을 사용할 수 있다.



- 예를 들어서 scores 리스트의 모든 요소를 출력하려면 아래와 같이 할 수 있다.

```
for element in scores:
        print(element)
```

- scores 리스트의 첫 번째 요소부터 변수 element에 할당되고 반복 루프 안의 문장들이 실행된다. 한번의 반복이 끝나민 두 번째 요소가 변수 element에 할당된다. 이 경우에는 우리가 리스트의 크기에 대하여 신경 쓰지 않아도 된다.

1. 리스트란?

4) list 클래스

- 리스트는 list 클래스에 의하여 정의된다. list 클래스의 생성자를 이용해서도 생성할 수 있다. 아래는 모두 리스트를 생성하는 방법들이다.

```
list1 = list()                  # 공백 리스트 생성
list2 = list("Hello")           # 문자 H, e, l, l, o를 요소로 가지는 리스트 생성
list3 = list(range(0, 5))       # 0, 1, 2, 3, 4를 요소가 가지는 리스트 생성
```

- 위의 방법들은 초기값을 사용하여 리스트를 생성하는 방법과 동일하다. 예를 들어서 위의 문장들은 아래의 문장들과 동일하다.

```
list1 = [ ]                     # 공백 리스트 생성
list2 = [ "H", "e", "l", "l", "o" ]     # 문자 H, e, l, l, o를 요소로 가지는 리스트
list3 = [ 0, 1, 2, 3, 4 ]       # 0, 1, 2, 3, 4를 요소가 가지는 리스트 생성
```

1. 리스트란?

5) 복잡한 리스트

- 동일한 자료형 뿐만 아니라 서로 다른 자료형의 요소를 하나의 리스트 안에 포함할 수 있다. 또 리스트 안에 다른 리스트를 포함시키는 것도 가능하다. 다음 문장들을 참고하자.

```
list1 = [12, "dog", 180.14]                                  # 혼합 자료형
list2 =  [["Seoul", 10], ["Paris", 12], ["London", 50]]      # 내장 리스트
list3 = ["aaa", ["bbb", ["ccc", ["ddd", "eee", 45]]]]   # 내장 리스트
```

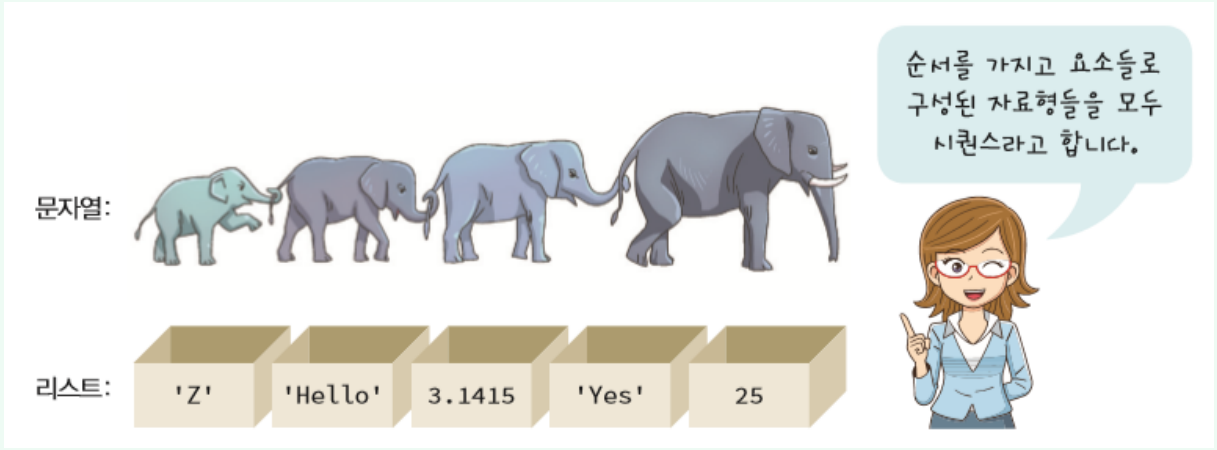
2. 시퀀스 자료형

1) 시퀀스란?

- 파이썬에서 리스트는 넓게 보면 시퀀스(sequence) 자료형에 속한다. 시퀀스에 속하는 자료형들은 순서를 가진 요소들의 집합이라는 공통적인 특성을 가지고 있으며 문자열, 리스트, 튜플이 모두 시퀀스의 일종이다.

  공식적으로 시퀀스 자료형에는 다음과 같은 6가지의 자료형이 있다.

  - 문자열
  - 바이트 시퀀스
  - 바이트 배열
  - 리스트
  - 튜플
  - range 객체



- 문자열, 리스트, 튜플, 바이트, range 객체들은 상당히 다르게 보이지만, 자세히 살펴보면 다음과 같은 공통적인 특징을 가지고 있다

  - 이 요소들은 순서를 가지고 있다.
  - 요소들은 인덱스를 사용하여 참조할 수 있다.

2. 시퀀스 자료형

1) 시퀀스란?

- 간단한 예로 문자열과 리스트를 비교해 보자.

```
text = "Will is power."
print(text[0], text[3], text[-1])
flist = ["apple", "banana", "tomato", "peach", "pear" ]
print(flist[0], flist[3], flist[-1])
```

```
W l .
apple peach pear
```

- 문자열과 마찬가지로 리스트도 인덱싱이 가능하다는 것을 알 수 있다. 다른 언어와는 다르게 파이썬은 시퀀스에 해당되는 자료형에는 동일한 연산자와 함수를 사용할 수 있다. 예를 들어서 문자열의 길이나 리스트의 길이는 len() 이라고 불리는 함수로 계산할 수 있다.

```
text = "Will is power.”
print(len(text))
flist = ["apple", "banana“,"tomato", "peach", "pear"]
print(len(flist))
```

```
14
5
```

2. 시퀀스 자료형

2) 시퀀스에서 가능한 연산과 함수

- 리스트에서 사용할 수 있는 연산자와 함수는 무척 많다. 그 중에서도 가장 많이 사용되는 연산자나 함수는
과 같다.

함수나 연산자	설명	예	결과
len()	길이 계산	len([1, 2, 3])	3
+	2개의 시퀀스 연결	[1, 2] + [3, 4, 5]	[1, 2, 3, 4, 5]
*	반복	['Welcome!'] * 3	['Welcome!', 'Welcome!', 'Welcome!']
in	소속	3 in [1, 2, 3]	True
not in	소속하지 않음	5 not in [1, 2, 3]	True
[]	인덱스	myList[1]	myList의 1번째 요소
min()	시퀀스에서 가장 작은 요소	min([1, 2, 3])	1
max()	시퀀스에서 가장 큰 요소	max([1, 2, 3])	3
for 루프	반복	for x in [1, 2, 3]: print (x)	1 2 3

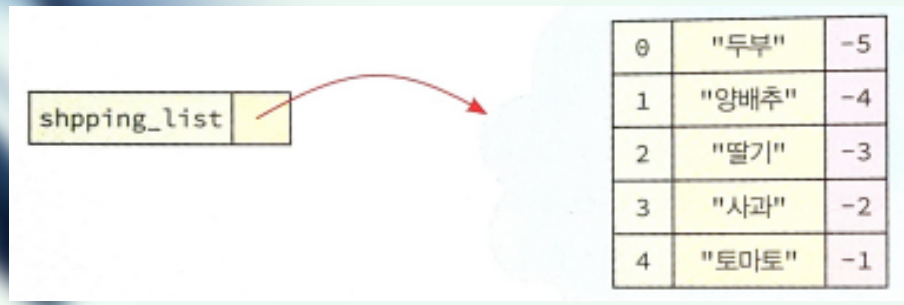
3. 인덱싱과 슬라이싱

1) 인덱싱

- 인덱싱(indexing)이란 리스트에서 하나의 요소를 인덱스 연산자를 통하여 참조(접근)하는 것을 의미한다. 인덱스는 정수이며, 항상 0 에서부터 시작한다는 것을 잊으면 안된다.

```
shopping_list = [ "두부", "양배추", "딸기", "사과", "토마토" ]
shopping_list[0]
```

```
'두부'
```



- 리스트의 인덱스는 0에서 (len(shopping_list) -1)까지의 범위를 가진다. shopping_list[index]는 하나의 변수와 똑같이 사용이 가능하다. 예를 들어서 shopping_list[0] 의 값을 변수 item에 할당할 수 있다.
- 파이썬에서는 음수 인덱스가 가능하다. shopping_listf[-1]은 은 리스트의 마지막 원소를 나타낸다. shopping_list[-3]은 리스트의 마지막에서 세 번째에 있는 요소를 나타낸다. 음수 인덱스가 사용될 때는 list[음수인덱스+리스트길이]와 같이 계산하면 된다.

3. 인덱싱과 슬라이싱

1) 인덱싱

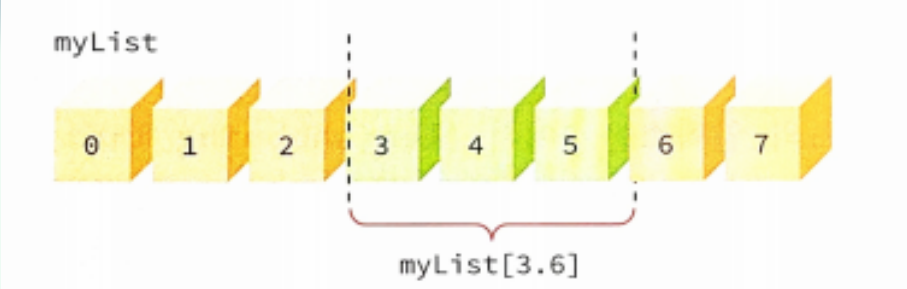
- 리스트의 크기를 넘어서는 인덱스를 사용하면 IndexError가 발생된다. 예를 들어서 다음과 같은 경우에는 오류가 발생한다.

```
shopping_list = [ "두부", "양배추", "딸기" ]
shopping_list[4]
```

```
…
IndexError: list index out of range
```

2) 슬라이싱

- 슬라이싱(slicing)은 리스트 안에서 범위를 지정하여서 원하는 요소들을 선택하는 연산이다. myList[start : end]와 같은 형식을 사용한다. myList[start : end] 라고 하면 start 인덱스에 있는 요소부터 (end-1) 인덱스에 있는 요소까지 선택된다.



3. 인덱싱과 슬라이싱

2) 슬라이싱

- 예를 들어서 거듭 제곱 값들을 모아서 리스트 안에 저장하였다고 하자.

```
squares = [0, 1, 4, 9, 16, 25, 36, 49]
squares[3:6]                          # 슬라이싱은 새로운 리스트를 반환한다.
```

```
[9, 16, 25]
```

- 슬라이싱 연산은 요구된 요소를 포함하는 부분 리스트를 반환한다. 즉 슬라이싱 연산을 하면 리스트의 새로운 복사본을 얻을 수 있다는 이야기이다. 슬라이싱 연산에서 시작 인덱스와 종료 인덱스는 생략될 수 있다. 인덱스가 생략되면 시작 인덱스는 0이 되고 종료 인덱스는 (리스트길이-1)이 된다.

```
squares = [0, 1, 4, 9, 16, 25, 36, 49]
squares [:3]
squares[4:]
squares[:]
```

```
[0, 1, 4]
[16, 25, 36, 49]
[0, 1, 4, 9, 16, 25, 36, 49]
```

- squares[: 3]은 squares[0 : 3]과 동일하고 squares[4 :]는 squares[4 : len(squares)-1]와 동일하다. squares[:]에서는 시작 인덱스와 종료 인덱스가 모두 생략된 형태로 squares[0 : len(squares)-1]와 같다.

3. 인덱싱과 슬라이싱

2) 슬라이싱

- 문자열과는 다르게 리스트는 변경 가능하다. 즉 우리는 언제든지 리스트의 내용을 변경할 수 있다.

```
squares = [0, 1, 4, 9, 16, 25, 36, 48]   # 잘못된 부분이 있음
7 ** 2                                   # 7의 제곱은 49임!
49
squares[7] = 49                          # 잘못된 값을 변경한다.
squares
```

```
[0, 1, 4, 9, 16, 25, 36, 49]
```

- 우리는 슬라이싱에 값을 대입하는 것도 가능하다. 리스트의 크기를 변경하는 것도 가능하다. 공백 리스트로 모든 요소들을 대체하여서 리스트의 내용을 완전히 삭제하는 것도 가능하다.

```
letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
letters
letters[2:5] = ['C', 'D', 'E']    # 리스트의 일부를 변경해보자
letters
letters[2:5] = []                 # 리스트의 일부를 삭제해보자
letters
```

```
['a', 'b', 'c', 'd', 'e', 'f', 'g']
['a', 'b', 'C', 'D', 'E', 'f', 'g']
['a', 'b', 'f', 'g']
```

4. 리스트의 기초 연산들

1) 리스트 합병과 반복

- 앞에서 두 개의 리스트를 합칠 때는 연결 연산자인 + 연산자를 사용할 수 있었다.

```
marvel_heroes = [ "스파이더맨", "헐크“ , “아이언맨" ]
dc_heroes = [ "슈퍼맨", "배트맨", "원더우먼" ]
heroes = marvel_heroes + dc_heroes
print(heroes)
```

```
['스파이더맨', '헐크', '아이언맨', '슈퍼맨', '배트맨', '원더우먼']
```

- 리스트를 반복하는 것도 반복 연산자인 *을 사용하면 된다.

```
values = [ 1,2,3 ] * 3
print(values)
```

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

2) 리스트의 길이

- len() 연산은 리스트의 길이를 계산하여 반환한다.

3) 요소 추가하기

- append()를 사용하여서 리스트의 끝에 새로운 항목을 추가할 수 있다.

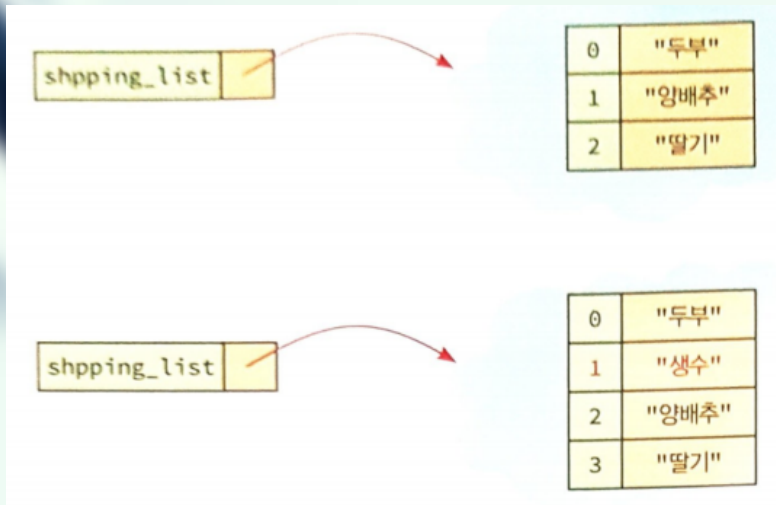
4. 리스트의 기초 연산들

4) 요소 삽입하기

- append() 메소드는 리스트의 끝에 새로운 요소를 추가한다. 우리는 종종 기존 리스트의 특정한 위치에 새로운 요소를 추가하기를 원한다. 이런 경우에 사용할 수 있는 메소드가 insert()이다. 다음과 같은 리스트의 인덱스 1에 "생수"를 추가하여 보자.

```
shopping_list = ["두부", "양배추", "딸기"]
shopping_list.insert(1, "생수")
print(shopping_list)
```

```
['두부', '생수', '양배추', '딸기']
```



" 생수" 항목이 인덱스 1에 추가되었고 이후의 항목들은 모두 뒤로 한 칸 이동한 것을 알 수 있다.

4. 리스트의 기초 연산들

5) 요소 찾기

- 리스트에 어떤 요소가 있는지를 찾는 연산도 많이 사용된다. 어떤 요소가 리스트에 있는지 없는지만 알려면 in 연산자를 사용하면 된다.

```
heroes = [ "스파이더맨", "슈퍼맨", "헐크", "아이언맨", "배트맨" ]
if "배트맨" in heroes :
        print("배트맨은 영웅입니다. ")
```

```
배트맨은 영웅입니다.
```

- 우리는 종종 어떤 요소의 리스트 안에서의 위치를 알아야 한다. 이런 경우에 사용할 수 있는 메소드가 index()이다. 예를 들어서 리스트에서 "슈퍼맨"의 인덱스를 알고 싶으면 다음과 같이 한다.

```
heroes [ "스파이더맨", "슈퍼맨", "헐크", "아이언맨", "배트맨" ]
index = heroes.index ("슈퍼맨")                    # index는 1이 된다.
```

- 만약 리스트에 없는 항목을 index()로 찾으면 오류가 발생할 수 있다. 따라서 다음과 같이 먼저 리스트에 있는지를 확인한 후에 항목의 인덱스를 찾는 것이 안전하다.

```
heroes = [ "스파이더맨", "슈퍼맨", '헐크", "아이언맨", "배트맨" ]
if "슈퍼맨" in heroes :
        index = heroes.index ("슈퍼맨")
```

4. 리스트의 기초 연산들

6) 요소 삭제하기

- pop() 메소드는 특정한 위치에 있는 항목을 삭제한다. pop(1) 하면 인덱스 1에 있는 항목이 삭제되는 동시에 반환된다.

```
heroes = [ "스파이더맨", "슈퍼맨", "헐크", "아이언맨", "배트맨" ]
print(heroes.pop(1))
heroes
```

```
슈퍼맨
['스파이더맨', '헐크', '아이언맨', '배트맨']
```

- remove() 메소드는 항목을 받아서 제거한다. pop()과 다른 점은 항목의 값을 받아서 일치하는 항목을 삭제한다는 것이다.아울러 삭제된 값은 반환되지 않는다.

```
heroes = [ "스파이더맨“, "슈퍼맨", "헐크", "아이언맨", "배트맨", "조커" ]
heroes.remove("조커")
print(heroes)
```

```
['스파이더맨', '슈퍼맨', '헐크', '아이언맨', '배트맨']
```

4. 리스트의 기초 연산들

7) 리스트 일치 검사

- 비교 연산자 ==, !=, >, <를 사용하여 2개의 리스트를 비교할 수 있다. 리스트를 비교하려면, 먼저 2개의 리스트가 동일한 자료형의 요소들을 가지고 있어야 한다. 리스트의 첫 번째 요소들을 비교한다. 첫 번째 요소의 비교에서 False가 나오면 더 이상의 비교는 없고 False가 그대로 출력된다. 첫 번째 요소가 같으면 두 번째 요소를 꺼내서 비교한다. 리스트 안의 모든 요소가 비교될 때까지 동일한 작업을 반복한다. 리스트 안의 모든 요소를 비교하여 모두 True가 나오면 전체 결과가 True가 된다.
- 예를 들어서 list1과 list2를 연산자를 이용하여 비교해보면 다음과 같다. 첫 번째 요소가 ==로 비교되고 이어서 두 번째 요소가 ==로 비교된다. 리스트의 모든 요소에 대하여 연산이 True가 나오면 list1 == list2도 True가 된다.

```
list1 = [ 1, 2, 3 ]
list2 = [ 1, 2, 3 ]
print(list1 == list2)
```

```
True
```

- == 를 이용하여 2개의 리스트를 비교할 때, 리스트의 길이가 다르면 False가 된다.

```
list1 = [ 1, 2, 3 ]
list2 = [ 1, 2 ]
print(list1 == list2)
```

```
False
```

4. 리스트의 기초 연산들

8) 리스트 정렬하기

- 리스트의 요소들을 크기 순으로 정렬시키는 연산은 아주 많이 사용된다. 리스트를 정렬하는 방법에는 다음과 같이 2가지의 방법이 있다. 2가지 방법이 약간 다르다.

  ● 리스트 객체의 sort() 메소드를 사용하는 방법

  ● sorted() 내장 함수를 사용하는 방법

- sort() 메소드는 리스트를 제자리(in-place)에서 정렬한다. 따라서 sort()가 호출되면 원본 리스트가 변경된다. 아래의 코드에서 리스트 li는 정렬된 상태로 변경된다.

```
li = [ 3, 2, 1, 5, 4 ]
li.sort()
print(li)
```

```
[1, 2, 3, 4, 5]
```

- 원본을 유지하고 새로이 정렬된 리스트를 원한다면 내장 함수인 sorted()를 사용하는 것이 좋다. sorted()는 정렬된 새로운 리스트를 반환한다.

```
li = [ 3, 2, 1, 5, 4 ]
s_li = sorted(li)
print(s_li)
```

```
[1, 2, 3, 4, 5]
```

4. 리스트의 기초 연산들

8) 리스트 정렬하기

- 리스트를 정렬할 때, key 매개 변수를 이용하여 요소들을 비교하기 전에 sorted() 함수에 연결하여 호출되는 함수를 지정할 수 있다. 예를 들어서 split() 함수를 연결해서 단어(토큰)별로 분리 후, key 매개변수에 str.lower 값을 지정하면 아스키 코드(한글이면 유니코드)별로 정렬할 수 있다.

```
li = sorted("A picture is worth a thousand words.".split(), key=str. lower))
print(li)
```

```
['A', 'a', 'is', 'picture', 'thousand', 'words.', 'worth']
```

- list.sort()와 sorted()는 모두 부울형의 reverse 매개 변수를 가진다. 이 매개변수는 정렬 방향을 지정하는데 사용된다. 예를 들어서 리스트를 역순으로 정렬하려면 다음과 같이 한다.

```
li = sorted([5, 2, 3, 1, 4], reverse=True)
print(li)
```

```
[5, 4, 3, 2, 1]
```

4. 리스트의 기초 연산들

9) 문자열에서 리스트 만들기

- 문자열의 split() 메소드는 문자열을 분리하고 이것을 리스트로 만들어서 반환한다. 이 때 문자열을 분리하는 분리자(separator)를 지정할 수 있다. 만약, 분리자가 지정되지 않으면 스페이스를 이용하여 문자열을 분리한다.

```
statement = "Where there is a will, there is a way"
li = statement.split()
print(li)
```

```
['Where', 'there', 'is', 'a', 'will,', 'there', 'is', 'a', 'way']
```

- 만약 ,(콤마)를 이용하여 문자열을 분리하려고 한다면 다음과 같이 한다.

```
statement = "Where there is a will, there is a way"
li = statement.split(",")
print(li)
```

```
['Where there is a will,', ' there is a way']
```

4. 리스트의 기초 연산들

리스트 연산 정리

연산의 예	설명
mylist[2]	인덱스 2에 있는 요소
mylist[2] = 3	인덱스 2에 있는 요소를 3으로 설정한다.
del mylist[2]	인덱스 2에 있는 요소를 삭제한다.
len(mylist)	mylist의 길이를 반환한다.
"value" in mylist	"value"가 mylist에 있으면 True
"value" not in mylist	"value"가 mylist에 없으면 True
mylist.sort()	mylist를 정렬한다.
mylist.index("value")	"value"가 발견된 위치를 반환한다.
mylist.append("value")	리스트의 끝에 "value"요소를 추가한다.
mylist.remove("value")	mylist에서 "value"가 나타나는 위치를 찾아서 삭제한다.

감사합니다.