

리스트 학습하기

천양하

파이썬 프로그래밍의 기초, 자료형

⋮

리스트 자료형

튜플 자료형

딕셔너리 자료형

집합 자료형

02-3 리스트 자료형

■ 리스트(List)란?

- 자료형의 집합을 표현할 수 있는 자료형

```
>>> odd = [1, 3, 5, 7, 9]
```

- 숫자와 문자열만으로 프로그래밍을 하기엔 부족한 점이 많음
 - 예) 1부터 10까지의 숫자 중 홀수 모음인 집합 {1, 3, 5, 7, 9}는 숫자나 문자열로 표현 불가능
 - 리스트로 해결 가능!

02-3 리스트 자료형

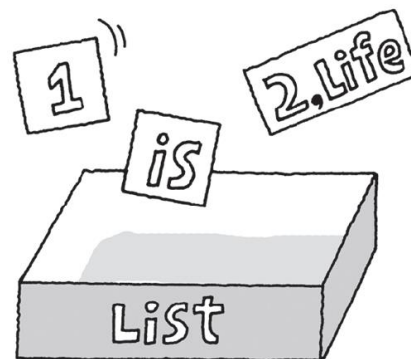
■ 리스트 사용법

- 대괄호([])로 감싸고 각 요소값은 쉼표(,)로 구분

```
리스트명 = [요소1, 요소2, 요소3, ...]
```

- 리스트 안에 어떠한 자료형도 포함 가능

```
>>> a = []  
>>> b = [1, 2, 3]  
>>> c = ['Life', 'is', 'too', 'short']  
>>> d = [1, 2, 'Life', 'is']  
>>> e = [1, 2, ['Life', 'is']]
```



```
1 aa = [0, 0, 0, 0]
2 hap = 0
3
4 aa[0] = int(input("1번째 숫자 : "))
5 aa[1] = int(input("2번째 숫자 : "))
6 aa[2] = int(input("3번째 숫자 : "))
7 aa[3] = int(input("4번째 숫자 : "))
8
9 hap = aa[0] + aa[1] + aa[2] + aa[3]
10
11 print("합계 ==> %d" % hap)
```

1행 : aa=[0, 0, 0, 0]으로 항목이 4개 있는 리스트 생성
4행 : a 대신 aa[0]을 사용
5~7행 : 리스트 aa를 사용
9행 : 각 변수 대신 aa[0]+aa[1]+aa[2]+aa[3]으로 수정

출력 결과

1번째 숫자 : 10
2번째 숫자 : 20
3번째 숫자 : 30
4번째 숫자 : 40
합계 ==> 100

출력 결과는 리스트를 사용하기 전과 동일
숫자 100개를 더하려면 aa=[0, 1, 0, ..., 99]를 생성한 후
aa[0]+aa[1]+aa[2] +...+aa[99]로 작성

02-3 리스트 자료형

■ 리스트 인덱싱

- 문자열과 같이 인덱싱 적용 가능

```
>>> a = [1, 2, 3]
>>> a
[1, 2, 3]
```

- 파이썬은 숫자를 0부터 세기 때문에 a[0]이 리스트 a의 첫 번째 요소

```
>>> a[0]
1
```

- a[-1]은 리스트 a의 마지막 요솟값

```
>>> a[-1]
3
```

- 요솟값 간의 덧셈

```
>>> a[0] + a[2] ← 1 + 3
4
```

02-3 리스트 자료형

■ 리스트 인덱싱

- 리스트 내에 리스트가 있는 경우

```
>>> a = [1, 2, 3, ['a', 'b', 'c']]
```

- a[-1]은 마지막 요솟값인 리스트 ['a', 'b', 'c'] 반환

```
>>> a[0]
1
>>> a[-1]
['a', 'b', 'c']
```

- 리스트 a에 포함된 ['a', 'b', 'c'] 리스트에서 'a' 값을 인덱싱을 사용해 반환할 방법은?

```
>>> a[-1][0]
'a'
```

- a[-1]로 리스트 ['a', 'b', 'c']에 접근하고, [0]으로 요소 'a'에 접근

02-3 리스트 자료형

■ 리스트 슬라이싱

- 문자열과 같이 슬라이싱 적용 가능

```
>>> a = [1, 2, 3, 4, 5]
>>> a[0:2]
[1, 2]
```

```
>>> a = [1, 2, 3, 4, 5]
>>> b = a[:2] ← 처음부터 a[1]까지
>>> c = a[2:] ← a[2]부터 마지막까지
>>> b
[1, 2]
>>> c
[3, 4, 5]
```


02-3 리스트 자료형

■ 리스트 연산하기

■ 더하기(+)

- + 기호는 2개의 리스트를 합치는 기능
- 문자열에서 "abc" + "def" = "abcdef"가 되는 것과 같은 의미

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> a + b
[1, 2, 3, 4, 5, 6]
```

■ 반복하기(*)

- * 기호는 리스트의 반복을 의미
- 문자열에서 "abc" * 3 = "abccabccabc"가 되는 것과 같은 의미

```
>>> a = [1, 2, 3]
>>> a * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

02-3 리스트 자료형

■ 리스트 연산하기

■ 리스트 길이 구하기

- len() 함수 사용
- 문자열, 리스트 외에 앞으로 배울 튜플과 딕셔너리에서도 사용 가능한 내장 함수

```
>>> a = [1, 2, 3]
>>> len(a)
3
```

02-3 리스트 자료형

■ 리스트의 수정과 삭제

■ 리스트에서 값 수정하기

```
>>> a = [1, 2, 3]
>>> a[2] = 4
>>> a
[1, 2, 4]
```

■ 리스트 요소 삭제하기

■ del 키워드 사용

del 객체

```
>>> a = [1, 2, 3]
>>> del a[1]
>>> a
[1, 3]
```

```
>>> a = [1, 2, 3, 4, 5]
>>> del a[2:]
>>> a
[1, 2]
```

※ 슬라이싱 기법 활용 가능

02-3 리스트 자료형

■ 리스트 관련 함수

■ append()

- 리스트의 맨 마지막에 요소 추가

```
>>> a = [1, 2, 3]
>>> a.append(4) ← 리스트의 맨 마지막에 4를 추가
>>> a
[1, 2, 3, 4]
```

- 어떤 자료형도 추가 가능

```
>>> a.append([5,6]) ← 리스트의 맨 마지막에 [5,6]을 추가
>>> a
[1, 2, 3, 4, [5, 6]]
```

■ sort()

- 리스트의 요소를 순서대로 정렬

```
>>> a = [1, 4, 3, 2]
>>> a.sort()
>>> a
[1, 2, 3, 4]
```

- 문자의 경우 알파벳 순서로 정렬 가능

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
```

```

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 만
들

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 3
까지 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

02-3 리스트 자료형

■ 리스트 관련 함수

■ reverse()

- 리스트를 역순으로 뒤집어 줌
- 요소를 역순으로 정렬하는 것이 아닌, 현재의 리스트 그대로 뒤집음

```
>>> a = ['a', 'c', 'b']
>>> a.reverse()
>>> a
['b', 'c', 'a']
```

■ index()

- 요소를 검색하여 위치 값 반환

```
>>> a = [1,2,3]
>>> a.index(3) ← 3은 리스트 a의 세 번째(a[2]) 요소
2
```

- 값이 존재하지 않으면, 값 오류 발생

```
>>> a.index(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 0 is not in list
```

오류 메시지

02-3 리스트 자료형

■ 리스트 관련 함수

■ insert()

- 리스트에 요소 삽입
- insert(a, b)
 - a번째 위치에 b를 삽입하는 함수

```
>>> a = [1, 2, 3]
>>> a.insert(0, 4) ← a[0] 위치에 4 삽입
[4, 1, 2, 3]
```

■ remove()

- remove(x)
 - 리스트에서 첫 번째로 나오는 x를 삭제

```
>>> a = [1, 2, 3, 1, 2, 3]
>>> a.remove(3)
[1, 2, 1, 2, 3]
```

- 값이 여러 개인 경우 첫 번째 것만 삭제

```
>>> a.remove(3)
[1, 2, 1, 2]
```


02-3 리스트 자료형

■ 리스트 관련 함수

■ pop()

- 리스트의 맨 마지막 요소를 돌려주고 해당 요소 삭제
- pop(x)
 - 리스트의 x번째 요소를 돌려주고 해당 요소 삭제

```
>>> a = [1, 2, 3]
>>> a.pop()
3
>>> a
[1, 2]
```

```
>>> a = [1, 2, 3]
>>> a.pop(1)
2
>>> a
[1, 3]
```

■ count()

- 리스트에 포함된 요소의 개수 반환
- count(x)
 - 리스트 안에 x가 몇 개 있는지 조사하여 그 개수를 돌려주는 함수

```
>>> a = [1, 2, 3, 1]
>>> a.count(1)
2
```

02-3 리스트 자료형

■ 리스트 관련 함수

■ extend()

- 리스트에 리스트를 더하는 함수
- extend(x)
 - x에는 리스트만 올 수 있음

a.extend([4, 5])

=

a += [4, 5]

```
>>> a = [1, 2, 3]
>>> a.extend([4, 5])
>>> a
[1, 2, 3, 4, 5]
>>> b = [6, 7]
>>> a.extend(b)
>>> a
[1, 2, 3, 4, 5, 6, 7]
```

02-4 튜플 자료형

■ 튜플(Tuple)이란?

■ 리스트와 유사한 자료형

리스트	튜플
[]로 둘러쌘	()로 둘러쌘
생성 / 삭제 / 수정 가능	값 변경 불가능

```
>>> t1 = ()
>>> t2 = (1,)
>>> t3 = (1, 2, 3)
>>> t4 = 1, 2, 3
>>> t5 = ('a', 'b', ('ab', 'cd'))
```

- 튜플은 1개의 요소만을 가질 때는 요소 뒤에 콤마(,)를 반드시 붙여야 함 (예) t2 = (1,)
- 괄호()를 생략해도 무방함 (예) t4 = 1, 2, 3
- 프로그램이 실행되는 동안 값을 유지해야 한다면 튜플을, 수시로 값을 변경해야 하면 리스트 사용

02-4 튜플 자료형

- 튜플의 요솟값을 지울 수 있을까?

- 튜플의 요솟값은 한 번 정하면 지우거나 변경할 수 없음!

```
>>> t1 = (1, 2, 'a', 'b')
>>> del t1[0] ← 튜플 t1의 첫 번째 요소를 지우려고 시도
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'tuple' object doesn't support item deletion

형 오류(Type Error) 발생

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[0] = 'c' ← 튜플 t1의 첫 번째 요솟값을 변경하려고 시도
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'tuple' object does not support item assignment

형 오류 발생

02-4 튜플 자료형

■ 튜플 다루기

■ 인덱싱하기

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[0]
1
>>> t1[3]
'b'
```

■ 슬라이싱하기

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[1:] ← t1[1]부터 끝까지
(2, 'a', 'b')
```

■ 튜플 더하기와 곱하기

```
>>> t2 = (3, 4)
>>> t1 + t2
(1, 2, 'a', 'b', 3, 4)
>>> t2 * 3
(3, 4, 3, 4, 3, 4)
```

■ 튜플 길이 구하기

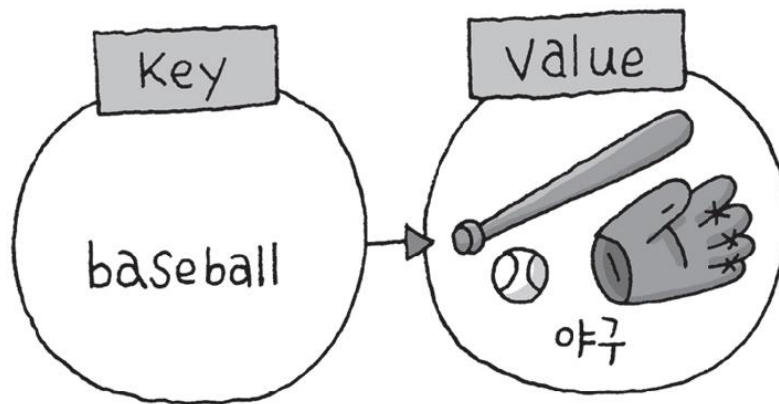
■ len() 함수

```
>>> t1 = (1, 2, 'a', 'b')
>>> len(t1)
4
```

02-5 딕셔너리 자료형

■ 딕셔너리(Dictionary)란?

- 대응 관계를 나타내는 자료형
- 연관 배열(Associative array) 또는 해시(Hash)
- Key와 Value를 한 쌍으로 갖는 자료형
- 순차적으로 해당 요솟값을 구하지 않고, Key를 통해 Value를 바로 얻는 특징



02-5 딕셔너리 자료형

■ 딕셔너리의 모습

```
{Key1:Value1, Key2:Value2, Key3:Value3, ...}
```

- Key와 Value의 쌍 여러 개 (Key : Value)
- {}로 둘러싸임
- 각 요소는 쉼표(,)로 구분됨

```
>>> dic = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
```

```
>>> a = {1: 'hi'}
```

```
>>> a = {'a': [1,2,3]}
```

02-5 딕셔너리 자료형

■ 딕셔너리 쌍 추가, 삭제하기

■ 딕셔너리 쌍 추가

```
>>> a = {1: 'a'}  
>>> a[2] = 'b' ← {2: 'b'} 쌍 추가  
>>> a  
{1: 'a', 2: 'b'}
```

■ 딕셔너리 요소 삭제

```
>>> del a[1] ← key가 1인 key:value 쌍 삭제  
>>> a  
{'name': 'pey', 3: [1, 2, 3], 2: 'b'}
```

■ 딕셔너리에서 Key 사용해 Value 얻기

```
>>> grade = {'pey': 10, 'julliet': 99}  
>>> grade['pey'] ← Key가 'pey'인 딕셔너리의 Value를 반환  
10  
>>> grade['julliet'] ← Key가 'julliet'인 딕셔너리의 Value를 반환  
99
```

■ 리스트나 튜플, 문자열은 요솟값 접근 시 인덱싱이나 슬라이싱 기법을 사용

■ 딕셔너리는 Key를 사용해 Value 접근

02-5 딕셔너리 자료형

■ 딕셔너리 관련 함수

■ keys()

- Key만을 모아서 dict_keys 객체 반환

```
>>> a = {'name': 'pey', 'phone': '0119993323', 'birth': '1118'}
>>> a.keys()
dict_keys(['name', 'phone', 'birth'])
```

- 리스트처럼 사용할 수 있지만,
리스트 관련 함수(append, insert, pop 등)는
사용 불가능

```
>>> for k in a.keys():
...     print(k)
...
name
phone
birth
```

- dict_keys 객체를 리스트로 변환하는 방법

```
>>> list(a.keys())
['name', 'phone', 'birth']
```

02-5 딕셔너리 자료형

■ 딕셔너리 관련 함수

■ `values()`

- Value만을 모아서 `dict_values` 객체 반환

```
>>> a.values()  
dict_values(['pey', '0119993323', '1118'])
```

■ `items()`

- Key와 Value의 쌍을 튜플로 묶은 값을 모아서 `dict_items` 객체 반환

```
>>> a.items()  
dict_items([('name', 'pey'), ('phone', '0119993323'), ('birth', '1118')])
```

02-5 딕셔너리 자료형

■ 딕셔너리 관련 함수

■ clear()

- 딕셔너리 내의 모든 요소 삭제
- 빈 딕셔너리는 {}로 표현

```
>>> a.clear()
>>> a
{}

```

■ in

- Key가 딕셔너리 안에 있는지 조사
- Key가 딕셔너리 안에 존재하면 True, 존재하지 않으면 False 반환

```
>>> a = {'name': 'pey', 'phone': '0119993323', 'birth': '1118'}
>>> 'name' in a
True
>>> 'email' in a
False

```

02-5 딕셔너리 자료형

■ 딕셔너리 관련 함수

■ get()

- Key에 대응되는 Value 반환
- 존재하지 않는 키 사용 시 None 반환
 - 오류를 발생시키는 list와 차이가 있음
- Key 값이 없을 경우
디폴트 값을 대신 반환하도록 지정 가능

```
>>> a = {'name': 'pey', 'phone': '0119993323', 'birth': '1118'}
>>> a.get('name')
'pey'
>>> a.get('phone')
'0119993323'
```

```
>>> a = {'name': 'pey', 'phone': '0119993323', 'birth': '1118'}
>>> print(a.get('nokey')) ← None을 리턴함
None
>>> print(a['nokey'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'nokey'
```

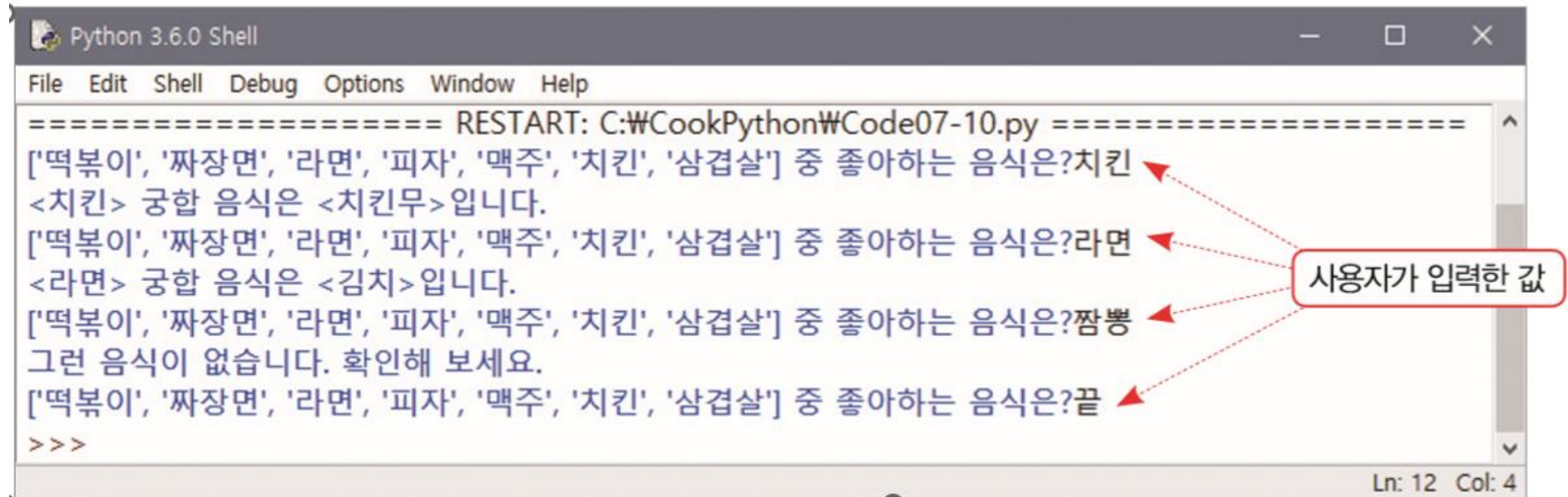
```
>>> a.get('foo', 'bar')
'bar'
```

■ 딕셔너리를 활용해 음식 궁합을 출력하는 프로그램

```
1  ## 변수 선언 부분 ##
2  foods = {"떡볶이":"오뎅",
3           "짜장면":"단무지",
4           "라면":"김치",
5           "피자":"피클",
6           "맥주":"땅콩",
7           "치킨":"치킨무",
8           "삼겹살":"상추"};
9
10 ## 메인 코드 부분 ##
11 while (True) :
12     myfood = input(str(list(foods.keys())) + " 중 좋아하는 음식은?")
13     if myfood in foods :
14         print("<%s> 궁합 음식은 <%s>입니다." % (myfood, foods.get(myfood)))
15     elif myfood == "끝" :
16         break
17     else :
18         print("그런 음식이 없습니다. 확인해 보세요.")
```

11~18행 : 무한 반복
12행 : 딕셔너리의 키 목록 출력
13행 : 입력한 음식이 딕셔너리에 있으면 14행 출력, '끝' 입력하면 16행에서 while 문 빠져나감, 모두 해당되지 않으면 17~18행에서 메시지 출력

출력 결과



The screenshot shows a Python 3.6.0 Shell window with the following text:

```
==== RESTART: C:\CookPython\Code07-10.py =====  
['떡볶이', '짜장면', '라면', '피자', '맥주', '치킨', '삼겹살'] 중 좋아하는 음식은?치킨  
<치킨> 궁합 음식은 <치킨무>입니다.  
['떡볶이', '짜장면', '라면', '피자', '맥주', '치킨', '삼겹살'] 중 좋아하는 음식은?라면  
<라면> 궁합 음식은 <김치>입니다.  
['떡볶이', '짜장면', '라면', '피자', '맥주', '치킨', '삼겹살'] 중 좋아하는 음식은?짬뽕  
그런 음식이 없습니다. 확인해 보세요.  
['떡볶이', '짜장면', '라면', '피자', '맥주', '치킨', '삼겹살'] 중 좋아하는 음식은?끝  
>>>
```

Four red dashed arrows point from a red-bordered box labeled "사용자가 입력한 값" (Value entered by user) to the input strings "치킨", "라면", "짬뽕", and "끝" in the code output.

Ln: 12 Col: 4

02-6 집합 자료형

■ 집합(Set)이란?

- 집합에 관련된 것을 쉽게 처리하기 위해 만든 자료형
- 파이썬 2.3부터 지원

- set 키워드를 사용하여 생성
 - set()에 리스트를 입력하여 생성

```
>>> s1 = set([1,2,3])  
>>> s1  
{1, 2, 3}
```

set()에 문자열을 입력하여 생성

```
>>> s2 = set("Hello")  
>>> s2  
{ 'e', 'H', 'l', 'o' }
```

02-6 집합 자료형

■ 집합의 특징

- 1) 중복을 허용하지 않음
- 2) 순서가 없음(Unordered)

```
>>> s2 = set("Hello")
>>> s2
{'e', 'H', 'l', 'o'}
```

- 리스트나 튜플은 순서가 있기 때문에 인덱싱을 통해 자료형의 값을 얻지만 set 자료형은 순서가 없기 때문에 인덱싱 사용 불가 (딕셔너리와 유사)
- 인덱싱 사용을 원할 경우 리스트나 튜플로 변환 필요
 - list(), tuple() 함수 사용

```
>>> t1 = tuple(s1) ← 튜플로 변환
>>> t1
(1, 2, 3)
>>> t1[0]
1
```

```
>>> s1 = set([1,2,3])
>>> l1 = list(s1) ← 리스트로 변환
>>> l1
[1, 2, 3]
>>> l1[0]
1
```


02-6 집합 자료형

■ 교집합, 합집합, 차집합 구하기

- set 자료형을 유용하게 사용할 수 있음

```
>>> s1 = set([1, 2, 3, 4, 5, 6])  
>>> s2 = set([4, 5, 6, 7, 8, 9])
```

- 교집합

- '&' 기호나 intersection() 함수 사용

```
>>> s1 & s2  
{4, 5, 6}
```

```
>>> s1.intersection(s2)  
{4, 5, 6}
```

- 합집합

- '|' 기호나 union() 함수 사용

```
>>> s1 | s2  
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
>>> s1.union(s2)  
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

- 차집합

- '-' 기호나 difference() 함수 사용

```
>>> s1 - s2  
{1, 2, 3}  
>>> s2 - s1  
{8, 9, 7}
```

```
>>> s1.difference(s2)  
{1, 2, 3}  
>>> s2.difference(s1)  
{8, 9, 7}
```

02-6 집합 자료형

■ 집합 관련 함수

■ add()

- 이미 만들어진 set 자료형에 값 추가

```
>>> s1 = set([1, 2, 3])
>>> s1.add(4)
>>> s1
{1, 2, 3, 4}
```

■ update()

- 여러 개의 값을 한꺼번에 추가

```
>>> s1 = set([1, 2, 3])
>>> s1.update([4, 5, 6])
>>> s1
{1, 2, 3, 4, 5, 6}
```

■ remove()

- 특정 값을 제거

```
>>> s1 = set([1, 2, 3])
>>> s1.remove(2)
>>> s1
{1, 3}
```