데이터 사이언스와 파이썬

데이터 타입 II (리스트, 튜플, 딕셔너리, 셋)

2020년 2학기
데이터사이언스융합전공

남세진
jordse@gmail.com

Contents

리스트와 튜플

- 파이썬의 문자열은 문자의 시퀀스
  - 파이썬에는 튜플과 리스트라는 다른 시퀀스 구조가 있음.
  - 튜플은 불변함. 튜플에 항목을 할당하고 나서, 이를 바꿀 수 없음.
  - 리스트는 변경 가능함. 항목을 할당하고, 자유롭게 수정하거나 삭제할 수 있음.

- 리스트 사용예

```
>>> list_example = ['one', 'two', 'three']
>>> print (list_example)
['one', 'two', 'three']
```

리스트 생성하기

- 리스트는 0 혹은 그 이상의 요소로 만들어짐. 콤마로 구분하고 대괄호로 둘러싸여 있음

```
example_list = []
weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
big_birds = ['emu', 'ostrich', 'cassowary']
first_name = ['Graham', 'John', 'Terry', 'Terry', 'Michael']

print(example_list)
print(weekdays)
print(big_birds)
print(first_name)
```

```
[]
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
['emu', 'ostrich', 'cassowary']
['Graham', 'John', 'Terry', 'Terry', 'Michael']
```

- list()함수를 사용한 리스트 할당

```
another_empty_list = list()
print(another_empty_list)
```

다른 데이터 타입을 리스트로 변환 : list()

- list()함수는 다른 데이터 타입을 리스트로 변환함
- list() 함수를 이용한 여러 예

```
>>> list('cat')
['c', 'a', 't']
>>> a_tuple = ('ready', 'fire', 'aim')
>>> list(a_tuple)
['ready', 'fire', 'aim']
>>> birthday = '1/6/1952'
>>> birthday.split('/')
['1', '6', '1952']
>>> splitme = 'a/b//c/d///e'
>>> splitme.split()
['a/b//c/d///e']
>>> splitme.split('/')
['a', 'b', '', 'c', 'd', '', '', 'e']
>>> aplitme = 'a/b//c/d///e'
>>> splitme.split('//')
['a/b', 'c/d', '/e']
```

[offset]으로 항목 얻기/항목 바꾸기

- 오프셋으로 리스트에서 항목 얻기

```
>>> numbers = ['one', 'two', 'three']
>>> numbers[0]
'one'
>>> numbers[1]
'two'
>>> numbers[2]
'three'
>>> numbers[-1]
'three'
>>> numbers[-2]
'two'
>>> numbers[-3]
'one'
```

- 오프셋을 이용하여 리스트의 항목을 변경 가능

```
>>> numbers = ['one', 'two', 'three']
>>> numbers[2]
'three'
>>> numbers
['one', 'two', 'three']
>>> numbers[2] = 'four'
>>> numbers
['one', 'two', 'four']
```

리스트의 리스트

- 리스트는 다음과 같이 리스트뿐만 아니라 다른 타입의 요소도 포함할 수 있음

```
>>> small_birds = ['hummingbird', 'finch']
>>> extinct_birds = ['dodo', 'passenger pigeon', 'Norwegian Blue']
>>> carol_birds = [3, 'French hens', 2, 'turtledoves']
>>> all_birds = [small_birds, extinct_birds, 'macaw', carol_birds]
>>> all_birds
[['hummingbird', 'finch'], ['dodo', 'passenger pigeon', 'Norwegian Blue'], 'macaw', [3, 'French hens', 2, 'turtledoves']]
```

- 오프셋을 이용한 접근
  - 리스트안에 포함된 리스트의 요소에 접근 예 : all_birds[1][0]

```
>>> all_birds[0]
['hummingbird', 'finch']
>>> all_birds[1]
['dodo', 'passenger pigeon', 'Norwegian Blue']
>>> all_birds[1][0]
'dodo'
```

슬라이스로 항목 추출하기

- 슬라이스를 이용하여 리스트의 서브시퀀스를 추출할 수 있음
  - 리스트의 슬라이스 또한 리스트

```
>>> numbers = ['one', 'two', 'three']
>>> numbers[0:2]
['one', 'two']
>>> numbers[::2]
['one', 'three']
>>> numbers[::-2]
['three', 'one']
>>> numbers[::-1]
['three', 'two', 'one']
>>> reverse_numbers = numbers[::-1]
>>> reverse_numbers
['three', 'two', 'one']
```

리스트와 리스트의 연결

- + 연산자를 이용한 방법과 python 3.5 이상부터 사용 가능한 방법

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

append() / extend() / +=

- append()는 리스트의 끝에 새 항목을 추가할 때 사용

```
>>> numbers = ['one', 'two', 'three']
>>> numbers.append('four')
>>> numbers
['one', 'two', 'three', 'four']
```

- extend()함수 += 는 리스트에 다른 리스트를 병합(merge)할 때 사용

```
>>> numbers = ['one', 'two', 'three']
>>> real_nums = [4,5,6]
>>> numbers.extend(real_nums)
>>> numbers
['one', 'two', 'three', 4, 5, 6]
>>> others = ['seven', 'eight']
>>> numbers += others
>>> numbers
['one', 'two', 'three', 4, 5, 6, 'seven', 'eight']
```

리스트에 항목추가와 삭제

- append()함수가 리스트의 끝에 항목을 추가한다면, 원하는 위치에 항목을 추가하려면?

```
['one', 'two', 'three']
>>> numbers.insert(2, 2.5)
>>> numbers
['one', 'two', 2.5, 'three']
>>> numbers.insert(10, 10)
>>> numbers
['one', 'two', 2.5, 'three', 10]
```

- del 을 이용한 항목 삭제하기

```
>>> numbers = ['one', 'two', 'three']
>>> numbers[2]
'three'
>>> del numbers[2]
>>> numbers
['one', 'two']
```

리스트에 항목추가와 삭제(2)

- remove()함수를 이용하여 항목의 값으로 삭제

```
>>> numbers = ['one', 'two', 'three', 'four']
>>> numbers.remove('four')
>>> numbers
['one', 'two', 'three']
>>> numbers.append('four')
>>> numbers.append('four')
>>> numbers
['one', 'two', 'three', 'four', 'four']
>>> numbers.remove('four')
>>> numbers
['one', 'two', 'three', 'four']
>>> numbers.remove('two')
>>> numbers
['one', 'three', 'four']
```

list.**remove**(*x*)

Remove the first item from the list whose value is equal to *x*. It raises a `ValueError` if there is no such item.

https://docs.python.org/3/tutorial/datastructures.html

리스트에 항목추가와 삭제(2)

- pop()함수를 이용한 삭제 및 항목 얻기
  - 얻고자 하는 항목의 오프셋을 사용
  - 인자가 없다면, 마지막 항목을 반환

list.**pop**([*i*])

Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list. (The square brackets around the *i* in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

```
>>> numbers = [1,2,3,4,5]
>>> numbers.pop()
5
>>> numbers
[1, 2, 3, 4]
>>> numbers.pop(1)
2
>>> numbers
[1, 3, 4]
```

리스트에서 항목 찾기 / 존재여부 확인하기 / 값 세기

- index()함수를 이용해 값으로 항목의 오프셋 찾기

```
>>> numbers = [1,2,3,9,4,5,6]
>>> numbers.index(9)
3
>>> numbers.pop(3)
9
>>> numbers
[1, 2, 3, 4, 5, 6]
```

- in을 이용하여 존재여부 확인하기, 그리고 값 세기 하기

```
>>> numbers = [1,2,3,4,5,6,9,10,34,534,342,23]
>>> 23 in numbers
True
>>> 11 in numbers
False
>>> numbers.append(1)
>>> numbers
[1, 2, 3, 4, 5, 6, 9, 10, 34, 534, 342, 23, 1]
>>> 1 in numbers
True
>>> numbers.count(1)
2
```

count() / sort() / join()

- count()
  - 리스트에 특정 값이 얼마나 있는지 세고자할 때 사용
- sort()
  - 오프세을 이용하여 리스트를 정렬할 때 사용
  - sort()는 리스트 자체를 내부적으로 정렬할 때 사용, sorted()는 리스트의 정렬된 복사본을 반환

```
>>> numbers = [1, 2, 3, 4, 1, 2, 5]
>>> numbers.count(1)
2
>>> numbers.count(5)
1
```

```
numbers = [1, 2, 3, 4, 1, 2, 5]
newone = sorted(numbers)
print(newone)
print(numbers)
```

join() 함수의 다양한 사용방법

- 문자열 결합시 사용
  - join()의 인자는 반복 가능한(iterable) 객체

```
>>> words = ['one', 'two', 'three']
>>> ', '.join(words)
'one, two, three'
```

```
>>> '|'.join('key=' + keyword for keyword in words)
'key=one|key=two|key=three'
```

- **[Quiz] 아래의 코드를 "Test_1_Test_2_Test_3_Test_4"를 출력하도록 수정하라**

```
inputTuple = ("Test", 1, "Test", 2, "Test", 3, "Test", 4)
sep = '_'
out = sep.join(inputTuple)
print(out)
```

할당 = / 복사 copy()

- = 연산자를 이용한 리스트의 할당

```
>>> a = [1,2,3]
>>> b = a
>>> id(a)
140518249334144
>>> id(b)
140518249334144
>>> b = [2,3,4]
```

- copy(), list(), [:] 를 이용한 리스트 복사

```
>>> a = [1,2,3]
>>> b = a.copy()
>>> id(a)
140518248847168
>>> id(b)
140518245282624
>>> c = list(b)
>>> id(c)
140518249457472
>>> d = a[:]
>>> id(d)
140518247735680
```

id 연산자(Identity Operators)

is : 양쪽 Operand가 동일한 Object를 가리키는지 아닌지를 검사
is not : 양쪽 Operand가 다른 Object를 가리키는지 아닌지를 검사

- 동일한 객체 여부를 판별하는 연산자
- id() 함수는 객체를 입력값으로 받아서 객체의 고유값(레퍼런스)을 반환하는 함수
- id는 파이썬이 객체를 구별하기 위해서 부여하는 일련번호.
- 숫자로서 의미는 없음.
- id는 동일한 객체 여부를 판별할 때 사용

튜플

- 임의적인 항목의 시퀀스
  - 리스트와 다르게 튜플은 불변, 즉 한번 정의한 후 추가, 삭제 수정할 수 없음
  - 상수의 리스트

- 튜플 생성
  - ()로 튜플 생성

```
>>> number_tuple = 'one', 'two', 'three'
>>> type(number_tuple)
<class 'tuple'>
>>> number_tuple
('one', 'two', 'three')
```

```
>>> number_tuple
('one', 'two', 'three')
>>> a,b,c = number_tuple
>>> a
'one'
>>> b
'two'
>>> c
'three'
```

튜플과 리스트

```
>>> a_tuple = ("a", "b", "mpilgrim", "z", "example")
>>> a_tuple
('a', 'b', 'mpilgrim', 'z', 'example')
>>> a_tuple[0]
'a'
>>> a_tuple[-1]
'example'
>>> a_tuple[1:3]
('b', 'mpilgrim')
```

1. A tuple is defined in the same way as a list, except that the whole set of elements is enclosed in parentheses instead of square brackets.
2. The elements of a tuple have a defined order, just like a list. Tuple indices are zero-based, just like a list, so the first element of a non-empty tuple is always a_tuple[0].
3. Negative indices count from the end of the tuple, just like a list.
4. Slicing works too, just like a list. When you slice a list, you get a new list; when you slice a tuple, you get a new tuple.

딕셔너리

- 딕셔너리는 리스트와 비슷함.
  - 값에 상응하는 고유한 키를 지정함.
  - 딕셔너리는 변경 가능하므로 키-값 요소를 추가, 삭제, 수정할 수 있음

- 딕셔너리 생성 : {}
  - 딕셔너리를 생성하기 위해서는 중괄호({})안에 콤마로 구분된 키:값쌍을 지정
  - 텅빈 딕셔너리는 {}

```
>>> new_dict = {}
>>> new_dict['first'] = 1
>>> new_dict['second'] = 2
>>> new_dict
{'first': 1, 'second': 2}
```

dict() 형변환 / 항목추가 및 변경

- dict()함수를 사용해 두 값으로 이루어진 시퀀스를 딕셔너리로 변환 가능
  - 시퀀스의 첫 번째의 항목은 Key로, 두 번째 항목은 Value로 사용됨

```
>>> numbers = [[1, 'first'], [2, 'second'], [3, 'third']]
>>> dict(numbers)
{1: 'first', 2: 'second', 3: 'third'}
>>> tps = 'ab', 'cd', 'ef'
>>> dict(tps)
{'a': 'b', 'c': 'd', 'e': 'f'}
>>> lt = [('key1', 1), ('key2', 2), ('key3', 3)]
>>> dict(lt)
{'key1': 1, 'key2': 2, 'key3': 3}
```

- 키에 의해 참조되는 항목에 값을 할당, 키가 딕셔너리에 이미 존재하는 경우, 값은 대체되고, 존재하지 않으면 추가됨

```
>>> lt
[('key1', 1), ('key2', 2), ('key3', 3)]
>>> lt_dict = dict(lt)
>>> lt_dict
{'key1': 1, 'key2': 2, 'key3': 3}
>>> lt_dict['key4'] = 4
>>> lt_dict
{'key1': 1, 'key2': 2, 'key3': 3, 'key4': 4}
```

딕셔너리 병합 update()

- update()함수는 한 딕셔너리의 키와 값들을 복사해서 다른 딕셔너리에 붙여줌

```
>>> numbers = {1 : 'one', 2: 'two', 3: 'three'}
>>> supp = {4:'four', 5:'five'}
>>> numbers.update(supp)
>>> numbers
{1: 'one', 2: 'two', 3: 'three', 4: 'four', 5: 'five'}
```

- 만약 병합할 딕셔너리에 중복되는 값이 있으면 새로운 값으로 갱신됨

```
>>> numbers
{1: 'one', 2: 'two', 3: 'three', 4: 'four', 5: 'five'}
>>> supp2 = {4:'4'}
>>> numbers.update(supp2)
```

```
>>> numbers
{1: 'one', 2: 'two', 3: 'three', 4: '4', 5: 'five'}
```

키와 del로 항목 삭제 / 모든 항목 삭제

- 딕셔너리의 키를 이용한 항목 삭제

```
>>> numbers
{1: 'one', 2: 'two', 3: 'three', 4: '4', 5: 'five'}
>>> del numbers[3]
>>> numbers
{1: 'one', 2: 'two', 4: '4', 5: 'five'}
```

- clear()를 이용한 딕셔너리의 모든 키값 삭제

```
>>> numbers
{1: 'one', 2: 'two', 4: '4', 5: 'five'}
>>> numbers.clear()
>>> numbers
{}
```

in으로 키 멤버십 테스트 / 항목 얻기 [key]

- 딕셔너리에 키가 존재하는지 알고 싶은 경우 in 을 사용

```
>>> numbers = {1: 'one', 2: 'two', 3: 'three', 4: '4', 5: 'five'}
>>> 3 in numbers
True
>>> 6 in numbers
False
>>> 'one' in numbers
False
>>> 'one' in numbers.values()
True
>>> 3 in numbers.keys()
True
```

- 딕셔너리에 키를 지정하여 상응하는 값을 얻는 방법

```
>>> numbers
{1: 'one', 2: 'two', 3: 'three', 4: '4', 5: 'five'}
>>> numbers[1]
'one'
>>> numbers[6]
Traceback (most recent call last):
  File "<input>", line 1, in <module>
KeyError: 6
>>> numbers.get(2)
'two'
>>> numbers.get(6, 0)
0
```

모든 키 / 모든 값 얻기

- 딕셔너리의 모든 키를 가져오기 위한 keys(), 딕셔너리의 모든 값을 가져오기 위한 values()
  - python 3에서는 python 2가 리스트를 반환하는 것과 달리, 순회 가능한(iterable) 키들을 보여주는 dict_keys()를 반환
  - 사용되지 않을 리스트를 생성하고 저장하기 위한 메모리와 시간을 소비하지 않기 때문에, 큰 딕셔너리의 처리에 효율적

```
>>> numbers
{1: 'one', 2: 'two', 3: 'three', 4: '4', 5: 'five'}
>>> numbers.keys()
dict_keys([1, 2, 3, 4, 5])
>>> numbers.values()
dict_values(['one', 'two', 'three', '4', 'five'])
>>> type(numbers.keys())
<class 'dict_keys'>
```

```
>>> for key, value in numbers.items():
...     print(key, value)
...
1 one
2 two
3 three
4 4
5 five
```

할당 / 복사

- 리스트와 마찬가지로 딕셔너리를 할당한 후 변경할 때 딕셔너리를 참조하는 모든 이름에 변경된 딕셔너리를 반영

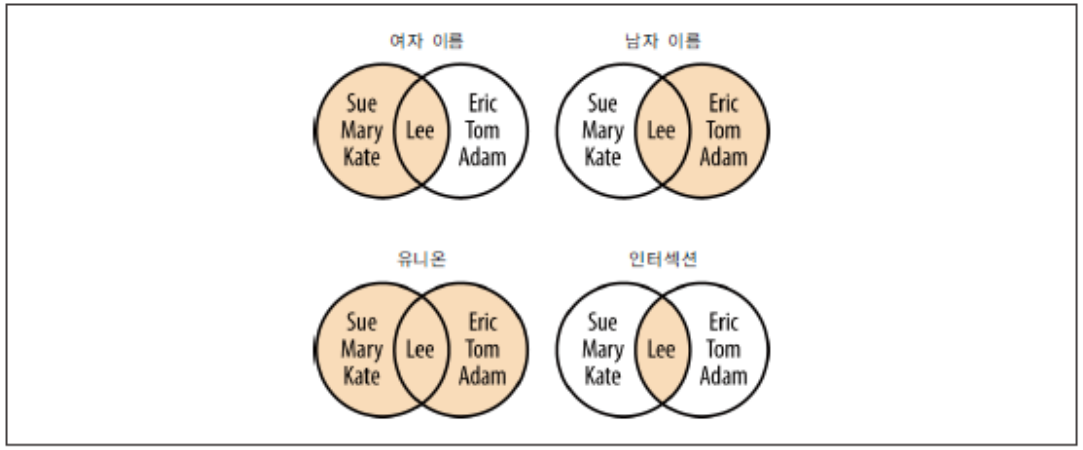
```
>>> numbers
{1: 'one', 2: 'two', 3: 'three', 4: '4', 5: 'five'}
>>> new_numbers = numbers
>>> id(new_numbers)
140518249450240
>>> id(numbers)
140518249450240
>>> new_numbers[5] = '5'
>>> numbers
{1: 'one', 2: 'two', 3: 'three', 4: '4', 5: '5'}
>>> new_numbers
{1: 'one', 2: 'two', 3: 'three', 4: '4', 5: '5'}
```

```
>>> numbers = {1: 'one', 2: 'two', 3: 'three', 4: '4', 5: 'five'}
>>> new_numbers = numbers.copy()
>>> id(numbers)
140518249450496
>>> id(new_numbers)
140518249313216
>>> numbers[5] = 5
>>> numbers
{1: 'one', 2: 'two', 3: 'three', 4: '4', 5: 5}
>>> new_numbers
{1: 'one', 2: 'two', 3: 'three', 4: '4', 5: 'five'}
```

셋 Set

- Set은 값은 버리고 키만 남은 딕셔너리와 같음
  - Set으로 할 수 있는 일반적인 것들

그림 3-1 셋으로 할 수 있는 일반적인 것들




```
>>> a_set = set()
>>> a_set.add(1)
>>> a_set.add(2)
>>> a_set.add(3)
>>> b_set = {4,5,6}
>>> type(a_set)
<class 'set'>
>>> type(b_set)
<class 'set'>
>>> c_list = [7,8,9]
>>> c_set = set(c_list)
>>> c_set
{8, 9, 7}
>>> d_dict = {10:'ten', 11:'eleven'}
>>> d_set = set(d_dict)
>>> d_set
{10, 11}
>>> e_set = set(d_dict.values())
>>> e_set
{'eleven', 'ten'}
```

Set Types

A *set* object is **an unordered collection** of distinct hashable objects. Common uses include membership testing, removing duplicates from a sequence, and computing mathematical operations such as intersection, union, difference, and symmetric difference. (For other containers see the built-in dict, list, and tuple classes, and the collections module.)

Set를 위한 다양한 연산

- 교집합(intersection), 합집합(union), 차집합(difference), 대칭 차집합(symmetric difference), 부분집합(subset), 진부분집합(proper subset)

```
>>> numbers = {
...     'even' : {2,4,6,8},
...     'odd' : {1,3,5,7,9},
...     'random' : { 1,2,4}
... }
>>> even = numbers['even']
>>> odd = numbers['odd']
>>> random = numbers['random']
>>> even & random
{2, 4}
>>> even - random
{8, 6}
>>> even | random
{1, 2, 4, 6, 8}
```

```
>>> even ^ random
{1, 6, 8}
>>> even <= odd
False
>>> even <= even
True
>>> even < even
False
```

내장 자료구조의 조합

- 딕셔너리에 딕셔너리를 값으로 가지는 경우

```
>>> freq_dict1 = {2000: 3, 2001:4, 2002:0, 2003:10}
>>> kf_matrix = {}
>>> kf_matrix['keyword1'] = freq_dict1
>>> kf_matrix
{'keyword1': {2000: 3, 2001: 4, 2002: 0, 2003: 10}}
```

- 딕셔너리의 값이 리스트인 경우

```
>>> pmids = [1111, 2222, 3333, 44444, 55555]
>>> keywordfreq = {}
>>> keywordfreq['keyword1'] = pmids
>>> keywordfreq
{'keyword1': [1111, 2222, 3333, 44444, 55555]}
>>> len(keywordfreq['keyword1'])
5
```

list, tuple, dict, set 비교

- list : ordered, indexing, mutable
- tuple : ordered, immutable
- dictionary : key:value, not duplicable(key), unordered, hash-table based
- set : key, not duplicable, unordered, hash-table based

- list vs tuple
  - mutable vs immutable

```
>>> list = [1,2,3,4,5]
>>> list[0] = 0
>>> list
[0, 2, 3, 4, 5]
>>> tu = (1,2,3,4,5)
>>> tu[0] = 0
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> tu
(1, 2, 3, 4, 5)
```

list, tuple, dict, set 비교

```python
import time
iteration = 20000

start = time.time()
test_set = set(range(iteration))
for i in range(iteration):
    if i in test_set:
        pass
print (time.time() - start)


start = time.time()
test_list = list(range(iteration))
for i in range(iteration):
    if i in test_list:
        pass
print (time.time() - start)
```

다음 코드를 보고, 어떤 자료 구조를 사용하는 것이 효율적인지 토론해 봅시다.

Quiz

- test_str = "citrus fruit,tropical fruit,whole milk,butter,curd,yogurt,flour,bottled water,dishes,fruit,tropical fruit,whole milk, yogurt,flour,bottled water,dishes,tropical fruit,cream cheese,processed"

- quiz 1
  - test_str은 ','로 구분된 식표품명으로 구성된 문자열이다. 식료품명이 각각 몇 번씩 사용되었는지 다음과 같은 형식으로 출력하라. 단 프로그램 작성시 딕셔너리를 사용하라

    word1 : ##

    word2 : ##

    word3 : ##

    ...

- quiz 2
  - test_str은 ','로 구분된 식표품명으로 구성된 문자열이다. 몇 종류의 식료품명이 사용되었는지 다음과 같은 형식으로 출력하라. 단 프로그램 작성시 셋을 사용하라.

    식료품의 종류 : ##

    식료품명 : word1, word2, word3...

Homework2

- 기본 문제 1 : 2019년에 판매된 식료품의 종류는 몇개인지 출력하는 프로그램을 작성하라
- 조금 더 생각하는 문제 2 : 2019년 가장 많이 판매된 식용품 Top 10을 출력하는 프로그램을 작성하라
- 첨부한 파일 homework2.txt파일은 다음과 같은 구조를 가짐
  - 파일의 내용은 2019년 월별로 정리된 식용품 판매 기록으로, 각 라인은 판매된 식용품 목록
  - 라인의 첫번째 열은 판매가 이루어진 년월(YYYY.MM)
  - 두번째 열부터 판매된 식용품이 ','로 구분되어 표시

```
2019.01,citrus fruit,semi-finished bread,margarine,ready soups,,,,,,,,,,,,,,,,,,,,,,,,,,,,
2019.01,tropical fruit,yogurt,coffee,,,,,,,,,,,,,,,,,,,,,,,,,,,,
2019.01,whole milk,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
2019.01,pip fruit,yogurt,cream cheese,meat spreads,,,,,,,,,,,,,,,,,,,,,,,,,,,
2019.01,other vegetables,whole milk,condensed milk,long life bakery product,,,,,,,,,,,,,,,,,,,,,,,,,
2019.01,whole milk,butter,yogurt,rice,abrasive cleaner,,,,,,,,,,,,,,,,,,,,,,,,,,
2019.01,rolls/buns,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
2019.01,other vegetables,UHT-milk,rolls/buns,bottled beer,liquor (appetizer),,,,,,,,,,,,,,,,,,,,,,,,
2019.01,potted plants,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
2019.01,whole milk,cereals,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
2019.01,tropical fruit,other vegetables,white bread,bottled water,chocolate,,,,,,,,,,,,,,,,,,,,,,,,
2019.01,citrus fruit,tropical fruit,whole milk,butter,curd,yogurt,flour,bottled water,dishes,,,,,,,,,,,,,,,,,,,,,
2019.01,beef,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
2019.01,frankfurter,rolls/buns,soda,,,,,,,,,,,,,,,,,,,,,,,,,,,
2019.01,chicken,tropical fruit,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
2019.01,butter,sugar,fruit/vegetable juice,newspapers,,,,,,,,,,,,,,,,,,,,,,,,,,
2019.01,fruit/vegetable juice,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
2019.01,packaged fruit/vegetables,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
2019.01,chocolate,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
2019.01,specialty bar,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
2019.01,other vegetables,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
2019.01,butter milk,pastry,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
```