
▣ Ch17 데이터 분석을 위한 데이터 구조 다루기

```
#int() #float() #str() #bool()  
  
#lower() #upper() #capitalize() #title() #replace() #find() #split()  
  
#list() #tuple() #dict()  
  
#append() #pop() #remove() #insert() #index() #sort()  
  
#len() #sorted() #del() #clear() #max() #min() #sum()  
  
#Series() #DataFrame() #zoneno #menu
```

Ch16과 Ch17 내용을 함께 다룬다. (이미 Python을 다룰 줄 아는 학생들을 대상으로 진행하기 때문에)

> Python의 데이터 구조 형식 중에 데이터 분석에 관련된 쓰임새 위주로 살펴본다.

> 하나의 값 만을 저장하는 스칼라(Scalar) 형 int, float, str, bool 자료형

> 여러 개의 값을 사용할 수 있는 비스칼라 형 list, tuple, dictionary 자료형

> 외장 Pandas 자료형인 series, data frame 자료형

■ Python 자료 구조 종류

[내장 자료형]

>> 스칼라(Scalar) 형

>> 리스트(list) 형

>> 투플(Tuple) 형

>> 딕셔너리(Dictionary) 형

[외장 자료형] Pandas 자료형

<< 시리즈(Series) 형

<< 데이터 프레임(Data Frame) 형

In []:

☒ 스칼라(Scalar) 형

> 하나의 값 만으로 구성된 자료 구조

> Data Type : int, float, str, bool 등

> 관련 함수 : int(), float(), str(), bool()

기본 자료형

In [10]:

```
## 스칼라(Scalar) 형 ##
## 생성 및 초기화
a = 3                      #int형 자료 구조 생성 및 초기화
print(type(a))               #자료구조 확인
b = 4.15                     #float형 자료 구조 생성 및 초기화
print(type(b))
c = 'hello, world!'          #str형 자료 구조 생성 및 초기화
print(type(c))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
```

○ Python에서는 자료 구조가 모두 class 형태로 만들어진다.

○ 따라서 자료형에 대해 메서드들이 제공된다.

데이터 타입 확인하기

```
In [19]: ## 타입 확인하기  
a = 3.14  
type(a)
```

```
Out[19]: float
```

```
In [20]: ## 타입 확인하기  
a = 3.14  
if type(a) is float:  
    print('>> Float type')
```

```
>> Float type
```

데이터 타입 바꾸기

> `int()`, `float()`, `str()`, `bool()`

```
In [13]: ## 타입 바꾸기: int(), float(), str()  
a = 3.14  
print(type(a), a)  
b = int(a)          # float > int  
print(type(b), b)  
c = str(b)          # int > str  
print(type(c), c)  
d = float(c)        # str > float  
print(type(d), d)  
e = str(d)          # float > str  
print(type(a), e)
```



```
<class 'float'> 3.14  
<class 'int'> 3  
<class 'str'> 3  
<class 'float'> 3.0  
<class 'float'> 3.0
```

자료형의 메서드 사용하기

```
In [127]: c.upper() # 'HELLO, WORLD!'
```

```
Out[127]: 'HELLO, WORLD!'
```

```
In [128]: c.capitalize() # 'Hello, world!'
```

```
Out[128]: 'Hello, world!'
```

```
In [129]: c.title() # 'Hello, World!'
```

```
Out[129]: 'Hello, World!'
```

```
In [130]: c.replace('!', '!!!') # 'Hello, World!!!'
```

```
Out[130]: 'hello, world!!!'
```

```
In [131]: c.find(' ', ' ') # 5
```

```
Out[131]: 5
```

```
In [132]: c.split(' ', ' ') # ['Hello', 'World!']
```

```
Out[132]: ['Hello', 'world!']
```

```
In [ ]:
```

▣ 비 스칼라형

> 여러 개의 값을 사용할 수 있도록 구성된 자료 구조

> Data Type : **list, tuple, dictionary**

> 관련 함수 : **list(), tuple(), dict()**

```
In [ ]:
```

○ 리스트(List) 형 : []

- > 여러 개의 값들을 나열된 자료 구조
- > 기본적으로 순서번호(Index)로 값에 접근(index는 자동 관리)
- > 값의 삽입, 삭제가 가능하다.
- > 각 값들은 [와] 안에 ,로 구분하여 초기화 한다.

추출하기

```
In [15]: ## 리스트(List) 형 : [ ] ##
## 생성 및 검색
a = [1, 2, 'a', 'b']           #리스트 생성 및 초기화
print(a[0])    #리스트는 0번째부터 시작 > a[0]는 한 개 값
print(a[-1])   #거꾸로 첫 번째
print(a[2:3])  #2번째부터 3번째 전까지 > a[2:3]는 한 개 값이지만 범위이므로 결과는 묶은 값 list형
print(a[2:])   #2번째부터 끝까지 > 결과는 묶은 값 list형
print(a[:2])   #처음부터 2번째 전까지 > 결과는 묶은 값 list형
```

```
1
b
['a']
['a', 'b']
[1, 2]
```

> 패킹(Packing)과 언패킹(Unpacking)

```
In [98]: ## 패킹(Packing), 언패킹(Unpacking)
b = [1, 2, ['a', 'b']]      #패킹(Packing)
x, y, z = b                #언패킹(Unpacking)
print(x, y, z)

1 2 ['a', 'b']
```

```
In [2]: ## 언패킹(Unpacking)
b = [1, 2, ['a', 'b']]      #패킹(Packing)
for x in b:
    if type(x) is list:
        for y in x:
```

```
    print(y)
else:
    print(x)
```

```
1
2
a
b
```

수정 조작하기

```
In [13]: ## 값 검색: index로 접근
a = [1, 2, 'a', 'b']
a[2]
```

```
Out[13]: 'a'
```

```
In [16]: ## index 검색: 값으로 index를 검색
a.index('a')      #'a'의 위치 값 반환
```

```
Out[16]: 2
```

```
In [5]: ## 값 변경: index로 접근
a[2] = 'A'
a
```

```
Out[5]: [1, 2, 'A', 'b']
```

관련 메서드

- > `append()` 항목 값 마지막에 추가하기
- > `pop()` 마지막 항목 값 제거하면서 반환
- > `remove()` 항목 값으로 제거하기
- > `insert(index, value)` index로 항목 추가하기
- > `index()` 항목 값으로 index 값 얻기
- > `sort()` 항목을 정렬하여 바꾸기 (매개변수 `reverse=False`가 기본값)

```
In [18]: ## [리스트] 조작: append()
a = []
a.append(2)          #리스트에 2를 추가
a.append(3)          #맨 마지막에 2를 추가
a
```

```
Out[18]: [2, 3]
```

```
In [19]: ## [리스트] 조작: pop()
a
a.pop()  #맨 마지막 항목을 반환하고, 제거
print(a)
```

```
[2]
```

```
In [25]: ## [리스트] 조작: insert()
a = [6, 1, 0, 3, 4, 5, 6]
a.insert(2, 9)      #2번째에 값 9를 삽입
a
```

```
Out[25]: [6, 1, 9, 0, 3, 4, 5, 6]
```

```
In [26]: ## [리스트] 조작: insert()
a = [6, 1, 0, 3, 4, 5, 6]
a.insert(2, 9)      #2번째에 값 9를 삽입
a.index(0)
```

```
Out[26]: 3
```

```
In [28]: ## [리스트] 조작: remove()
a.remove(9)        #9를 찾아 제거
a
```

```
Out[28]: [6, 1, 0, 3, 4, 5, 6]
```

```
In [29]: a.index(0)
```

```
Out[29]: 2
```

```
In [ ]: print(a.pop())
a
```

```
6  
Out[ ]: [0, 1, 2, 3, 4, 5, 6]
```

```
In [33]: ## 생성 및 조작  
a = [6, 1, 0, 3, 4, 5, 6]  
a.sort(reverse=False) #오름차순으로 정렬(내림차순은 True 또는 생략), a 자신을 정렬시킴  
a
```

```
Out[33]: [0, 1, 3, 4, 5, 6, 6]
```

```
In [ ]:
```

관련 함수

> `len()` 항목 개수 구하기

> `sorted()` 항목을 정렬하여 리스트를 반환(원 리스트는 변화 없음)

> `del()` 특정 항목 제거하기

> `clear()` 모든 항목 제거하기

> `max(), min(), sum()`

```
In [39]: ## 관련 함수: len()  
a = [5, 3, 7, 8, 2, 1]  
len(a) # 리스트 항목의 수를 반환
```

```
Out[39]: 6
```

```
In [35]: ## 관련 함수: 정렬  
a = [5, 3, 7, 8, 2, 1]  
asort = sorted(a) # 전달 받은 리스트 항목 값들을 정렬하여 반환  
asort
```

```
Out[35]: [1, 2, 3, 5, 7, 8]
```

```
In [36]: a = [5, 3, 7, 8, 2, 1]  
a.sort() #리스트 자체를 정렬  
a
```

```
Out[36]: [1, 2, 3, 5, 7, 8]
```

```
In [37]: ## 관련 함수: 제거, 갯 수  
del(a[1]) #1번째 값을 제거  
print(a)  
len(a)
```

```
[1, 3, 5, 7, 8]
```

```
Out[37]: 5
```

```
In [42]: ## 관련 함수: sum()  
a = [5, 3, 7, 8, 2, 1]  
max(a)  
min(a)  
sum(a)
```

```
Out[42]: 26
```

1, 2차원 리스트 초기화 하기

```
In [4]: ## 1차원 리스트 초기화 방법  
arr = [0, 0, 0, 0, 0]  
arr
```

```
Out[4]: [0, 0, 0, 0, 0]
```

```
In [5]: ## 1차원 리스트 초기화 방법  
arr = [0]*5;  
arr
```

```
Out[5]: [0, 0, 0, 0, 0]
```

```
In [94]: ## 1차원 리스트 초기화 방법  
arr = [0 for i in range(5)]; #[]는 list()와 같은 역할  
arr = list(0 for i in range(5));  
arr
```

```
Out[94]: [0, 0, 0, 0, 0]
```

```
In [7]: ## 1차원 리스트 초기화 방법
arr=[]; #만들고
for i in range(5):
    arr.append(0) #추가하고
arr
```

```
Out[7]: [0, 0, 0, 0, 0]
```

```
In [91]: ## 2차원 리스트 초기화 방법
arr = [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
arr
```

```
Out[91]: [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

```
In [9]: ## 2차원 리스트 초기화 방법
arr = [[0 for j in range(4)] for i in range(3)];
arr
```

```
Out[9]: [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

```
In [11]: ## 2차원 리스트 초기화 방법
arr=[]; #1차원 리스트 생성
for i in range(3):
    temp = [] #열 생성
    for j in range(4):
        temp.append(0) #열 값 추가
    arr.append(temp) #1차원 리스트에 열 추가
arr
```

```
Out[11]: [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

```
In [ ]:
```

[실습-0] 4행4열 리스트 초기화하기

> 0부터 시작되는 일련번호 값으로 초기화하기

>> n행 m열도 적용 가능하도록 일반화 함

```
In [96]: ## 2차원 리스트 초기화  
row, col = 4, 4
```

```
Out[96]: [[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11], [12, 13, 14, 15]]
```

```
In [ ]:
```

반복문으로 리스트 읽어내기

```
In [1]: ## 1차원 리스트 읽어내기  
a = [1, 2, 'a', 'b']      #리스트 생성 및 초기화  
for x in a:              #range() 대신에 리스트 자료 Unpacking  
    print(x, end=' ')
```

```
1 2 a b
```

```
In [2]: ## 2차원 리스트 읽어내기  
a = [1, 2, ['a', 'b'], 4]    #리스트 생성 및 초기화  
for x in a:                #range() 대신에 리스트 자료 Unpacking  
    if type(x) is list:     #x가 list type인지 확인  
        for y in x:  
            print(y, end=' ')  
    else:  
        print(x, end=' ')
```

```
1 2 a b 4
```

[실습] List로 막대 그래프 그리기

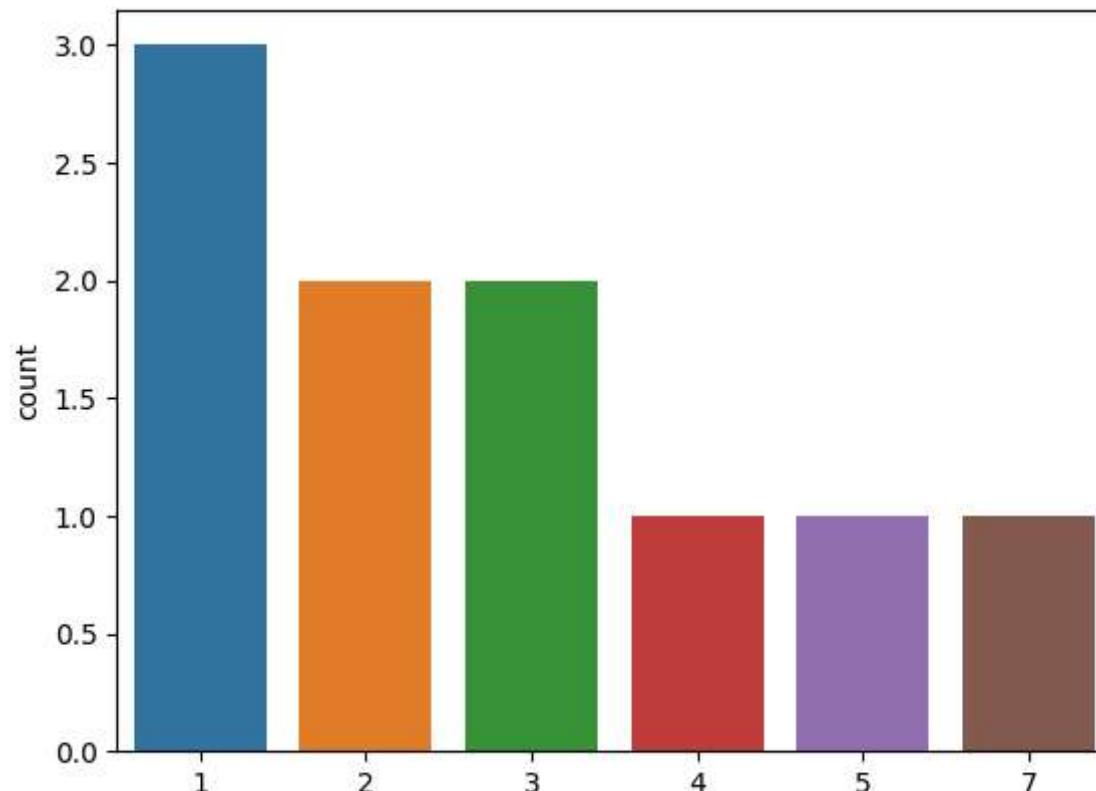
> import seaborn 모듈 사용

```
In [3]: ## 리스트(List) 형 : []  
## List로 막대 그래프 그리기  
alist = [1, 4, 5, 3, 2, 1, 7, 3, 1, 2]  
print(alist)  
  
import seaborn as sns  
sns.countplot(x = alist)
```

```
[1, 4, 5, 3, 2, 1, 7, 3, 1, 2]
```

```
C:\Users\ADMIN\anaconda3\lib\site-packages\scipy\__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.25.2)
... warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}""
C:\Users\ADMIN\anaconda3\lib\site-packages\seaborn\_\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
... if pd.api.types.is_categorical_dtype(vector):
C:\Users\ADMIN\anaconda3\lib\site-packages\seaborn\_\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
... if pd.api.types.is_categorical_dtype(vector):
C:\Users\ADMIN\anaconda3\lib\site-packages\seaborn\_\_oldcore.py:1765: FutureWarning: unique with argument that is not not a Series, Index, ExtensionArray, or np.ndarray is deprecated and will raise in a future version.
... order = pd.unique(vector)
C:\Users\ADMIN\anaconda3\lib\site-packages\seaborn\_\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
... if pd.api.types.is_categorical_dtype(vector):
```

```
Out[3]: <Axes: ylabel='count'>
```



[실습-1] List로 막대 그래프 그리기 (응용)

- > List를 랜덤 수로 초기화 함. (range는 0~9까지)
- > List의 길이는 30으로 함
- > 구성된 리스트를 출력하여 확인
- > `import seaborn` 모듈 사용 빈도수 막대 그래프 그리기

```
In [ ]: ## 리스트(List) 형 : []
## [실습] List를 랜덤 수로 초기화 함
## 랜덤 리스트 구성
import random
```

```
## List로 막대 그래프 그리기
import seaborn as sns
```

```
In [ ]: ## seaborn 패키지 인터페이스 매개변수 확인
import seaborn as sb
sb.countplot?
```

```
In [ ]:
```

○ 튜플(Tuple) 형 : ()

- > 리스트와 같이 여러 개의 값들을 나열된 자료 구조
- > 리스트와 다르게 한 번 초기화된 자료는 **수정할 수 없다.**
- > 고정된 값을 미리 정렬시켜서 튜플로 만들어 놓고 사용하면 검색 속도 빨라짐.

생성하기

```
In [16]: ## Tuple 생성하기
a = (1, 2, 3, 4, 5) #튜플의 생성 및 초기화
a
```

```
Out[16]: (1, 2, 3, 4, 5)
```

```
In [20]: # () 생략하고 튜플 만들기  
a = 1, 2, 3, 4, 5 #튜플의 패킹(Packing)  
a
```

```
Out[20]: (1, 2, 3, 4, 5)
```

추출하기

```
In [19]: ## 튜플(Tuple) 형 : ()  
## 생성 및 검색  
a = (1, 2, 3, 4, 5) #튜플의 생성 및 초기화  
print(a[0])  
print(a[1:3]) #범위로 추출하므로 결과는 tuple 형
```



```
1  
(2, 3)
```

> 패킹(Packing)과 언패킹(Unpacking)

```
In [44]: ## 패킹(Packing), 언패킹(Unpacking)  
b = 10, 20, 30 #튜플의 패킹(Packing)  
x, y, z = b #언패킹(Unpacking)  
print(x, y, z)
```

```
10 20 30
```

```
In [4]: ## 패킹(Packing), 언패킹(Unpacking)  
b = 10, #단일 항목 튜플의 패킹(Packing)  
type(b)
```

```
Out[4]: tuple
```

활용하기: List, Turtle

```
In [5]: #지역 번호 만들기  
zoneno = ([[['010'], '휴대전화'],  
           ['02', ['서울특별시', '광명시', '과천시']], ['031', '경기도'], ['032', ['인천광역시', '부천시']], ['033', '강원도'],  
           ['041', '충청남도'], ['042', ['대전광역시', '계룡시']], ['043', '충청북도'], ['044', '세종특별자치시'],
```

```
[ '051', '부산광역시'], [ '052', '울산광역시'], [ '053', ['대구광역시', '경산시']], [ '054', '경상북도'],
[ '055', '경상남도'], [ '061', '전라남도'], [ '062', '광주광역시'], [ '063', '전라북도'], [ '064', '제주특별자치도'])
```

#지역 번호 표 출력

```
for no, zone in zoneno: #unpacking
    print(no, " : ", zone)
```

```
['010'] : 휴대전화
02 : ['서울특별시', '광명시', '과천시']
031 : 경기도
032 : ['인천광역시', '부천시']
033 : 강원도
041 : 충청남도
042 : ['대전광역시', '계룡시']
043 : 충청북도
044 : 세종특별자치시
051 : 부산광역시
052 : 울산광역시
053 : ['대구광역시', '경산시']
054 : 경상북도
055 : 경상남도
061 : 전라남도
062 : 광주광역시
063 : 전라북도
064 : 제주특별자치도
```

[실습-2] Tuple 항목 검색하기

> 키보드로 지역명을 입력받아 zoneno의 번호를 출력해줌

```
In [17]: #지역번호 검색(지역명으로 찾기)
inzone = input(">전화번호 지역 입력: ")
```

```
032 : ['인천광역시', '부천시']
```

```
In [ ]:
```

○ 딕셔너리(Dictionary) 형 : { }

> 키(Key)와 대응되는 값(Value)이 짹을 이루어 나열된 자료 구조

- > 값의 삽입, 삭제가 가능하다.
- > 순서번호(Index) 대신에 키 값으로 값에 접근

생성하기

```
In [8]: ## 딕셔너리(Dictionary) 형 : { } ##
## Key : Value로 생성
a = {'A':90, 'B':80, 3:70, 4:60}          #딕셔너리 생성 및 초기화
a
```

```
Out[8]: {'A': 90, 'B': 80, 3: 70, 4: 60}
```

조작 하기

> 추가하기

```
In [9]: ## 항목 추가하기
a['C'] = [79, 70] #맨 뒤에 추가됨
a
```

```
Out[9]: {'A': 90, 'B': 80, 3: 70, 4: 60, 'C': [79, 70]}
```

> 삭제하기

```
In [10]: ## 항목 삭제하기
del(a['C']) # id 키:값 쌍 삭제
a
```

```
Out[10]: {'A': 90, 'B': 80, 3: 70, 4: 60}
```

> 값 변경하기

```
In [51]: ## 항목 값 변경하기
a['B'] = [89, 80] #Key가 없으면 추가됨
a
```

```
Out[51]: {'A': 90, 'B': [89, 80], 3: 70, 4: 60}
```

추출하기

```
In [11]: ## 딕셔너리(Dictionary) 형 : { } ##
## 추출 : key 사용
a = {'A':90, 'B':80, 3:70, 4:60}          #딕셔너리 생성 및 초기화
a['A']
```

```
Out[11]: 90
```

```
In [12]: ## 추출 : key 사용
a['a']
```

```
-----
KeyError..... Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_11524\1838990921.py in <module>
      1 ## 추출 : key 사용
----> 2 a['a']

KeyError: 'a'
```

```
In [13]: ## 추출 : .get() 사용
a.get('A')
```

```
Out[13]: 90
```

```
In [14]: ## 추출 : .get() 사용
a.get('a')
```

> Key와 Value 읽어내기

```
In [46]: ## Key 추출
a.keys()      #Key 추출
```

```
Out[46]: dict_keys(['A', 'B', 3, 4])
```

```
In [47]: ## Value 추출
a.values()    #Value 추출
```

```
Out[47]: dict_values([90, 80, 70, 60])
```

```
In [48]: ## Key와 Value 추출  
a.items() #Value 추출
```

```
Out[48]: dict_items([('A', 90), ('B', 80), (3, 70), (4, 60)])
```

반복문으로 Dictionary 읽어내기

```
In [62]: ## Dictionary 키 읽어내기  
a = {'A':90, 'B':80, 3:70, 4:60}  
for x in a.keys(): #range() 대신에 Dictionary 자료 Unpacking  
    print(x)
```

```
A  
B  
3  
4
```

```
In [60]: ## [Dictionary] 반복문으로 값 읽어내기  
for x in a.values(): #range() 대신에 Dictionary 자료 Unpacking  
    print(x)
```

```
90  
80  
70  
60
```

```
In [28]: ## [Dictionary] 반복문으로 키, 값 읽어내기  
for x in a.items(): #range() 대신에 적용  
    print(x)  
for x, y in a.items(): #range() 대신에 적용  
    print(x, y)
```

```
('A', 90)  
(('B', 80)  
(3, 70)  
(4, 60)  
A 90  
B 80  
3 70  
4 60
```

활용하기

```
In [30]: #(소분류) 메뉴표 만들기
menu = {'COFFEE': [['에스프레소', 3.0], ['아메리카노', 3.0], ['카페라떼', 4.0], ['카프치노', 4.0]],
        'LATTE': [['말차라떼', 4.0], ['초코라떼', 4.0], ['카페라떼', 4.0]],
        'TEA': [['청귤차', 4.0], ['자몽차', 4.0], ['레몬차', 4.0], ['카모마일', 4.5]],
        'ADE': [['자몽에이드', 4.5], ['레몬에이드', 4.5], ['청포도에이드', 4.5]],
        'JUICE': [['망고', 4.5], ['바나나', 4.5], ['딸기', 4.5], ['키위', 4.5]],
        'SMOOTHIE': [['청귤스무디', 4.5], ['요거트스무디', 4.5]],
        'MILK TEA': [['흑당밀크티', 4.5], ['달고나밀크티', 4.5]]}

#[출력] 소분류 메뉴
print("[[ 메뉴 ]]")
for title, mlist in menu.items():
    print("## %s ##" %title)
    for mmenu, price in mlist: #리스트 언팩킹
        print("%-10s%t%5.1f" %(mmenu, price))
    print()
```

```
[[ 메뉴 ]]
## COFFEE ##
에스프레소 ..... 3.0
아메리카노 ..... 3.0
카페라떼 ..... 4.0
카프치노 ..... 4.0

## LATTE ##
말차라떼 ..... 4.0
초코라떼 ..... 4.0
카페라떼 ..... 4.0

## TEA ##
청귤차 ..... 4.0
자동차 ..... 4.0
레몬차 ..... 4.0
카모마일 ..... 4.5

## ADE ##
자동에이드 ..... 4.5
레몬에이드 ..... 4.5
청포도에이드 ..... 4.5

## JUICE ##
망고 ..... 4.5
바나나 ..... 4.5
딸기 ..... 4.5
키위 ..... 4.5

## SMOOTHIE ##
청귤스무디 ..... 4.5
요거트스무디 ..... 4.5

## MILK TEA ##
흑당밀크티 ..... 4.5
달고나밀크티 ..... 4.5
```

[실습-3] Dictionary 항목 검색하기

- > 키보드로 대분류 메뉴명을 입력받아 메뉴명 단가를 출력해줌
- > 키보드 입력 값을 받아 모두 대문자 처리 후 작업

```
In [56]: #[주문] 대분류 메뉴 선택  
intitle = input(">대분류 메뉴명 : ")
```

```
에스프레소 ..... 3.0  
아메리카노 ..... 3.0  
카페라떼 ..... 4.0  
카프치노 ..... 4.0
```

```
In [ ]:
```

▣ 외장 Pandas 자료형

- > pandas라는 패키지를 import 해서 사용 가능한 추가 자료 구조
- > Data Type : Series, Dataframe
- > 관련 함수 : Series(), DataFrame()

```
In [ ]:
```

○ 시리즈(Series) 형

- > pandas 패키지를 통해 사용한다.
- > 1차원 배열로 여러 값을 나열한 자료 구조
- > 각 데이터 항목은 index로 식별된다.
- >>index는 자동 부여되거나, 임의 부여가 가능
- > 데이터 프레임의 데이터를 구성하는 열 값을 시리즈로 추출하여 조작할 수 있다.

생성하기

```
In [21]: ## [Series] index 미지정 생성하기  
import pandas as pd  
a = pd.Series([11, 22, 33, 44]) #index 자동 부여, 항목을 리스트 형태로 나열  
a
```

```
Out[21]: 0    11
         1    22
         2    33
         3    44
        dtype: int64
```

```
In [102... type(a)
```

```
Out[102]: pandas.core.series.Series
```

```
In [22]: ## [Series] 시리즈(Series) 형 ##
## index 지정 생성하기
b = pd.Series([11, 22, 33, 44], index = [1, 2, 3, 4]) #index 강제 부여
b
```

```
Out[22]: 1    11
         2    22
         3    33
         4    44
        dtype: int64
```

```
In [23]: ## [Series] index 지정 생성하기
data = {1 : 11, 2 : 22, 3 : 33, 4 : 44}
b = pd.Series(data) #딕셔너리로부터 시리즈 생성
b
```

```
Out[23]: 1    11
         2    22
         3    33
         4    44
        dtype: int64
```

추가/수정 하기

> index가 없으면 추가, 있으면 수정

```
In [76]: ## index=3에 9 할당
a = pd.Series([11, 22, 33, 44]) #index 자동 부여
a[4] = 9 #index가 없으면 추가, 있으면 수정
a
```

```
Out[76]: 0    11  
1    22  
2    33  
3    44  
4     9  
dtype: int64
```

```
In [79]: ## index=3에 9 할당  
b = pd.Series([11, 22, 33, 44], index = [1, 2, 3, 4]) #index 강제 부여  
b[4] = 9      #index가 없으면 추가, 있으면 수정  
b
```

```
Out[79]: 0    11  
1    22  
2    33  
3    44  
4     9  
dtype: int64
```

추출 하기

```
In [88]: ## 시리즈(Series) 형 ##  
## 생성 및 추출  
a = pd.Series([11, 22, 33, 44])  
print(a[1])      #값으로 반환  
print(a[1:2])   #Series로 반환
```

```
22  
1    22  
dtype: int64
```

```
In [17]: ## 생성 및 추출  
b = pd.Series([11, 22, 33, 44], index = ['a', 'b', 'c', 'd'])  
print(b['a'])  
print(b['b'])  
print(b[['a', 'b']])  #List로 검색, Series로 반환
```

```
22  
22  
a    11  
b    22  
dtype: int64
```

```
C:\Users\ADMIN\AppData\Local\Temp\ipykernel_11524\1303696067.py:3: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`  
... print(b[1])
```

삭제하기

```
In [18]: b = pd.Series([11, 22, 33, 44], index = ['a', 'b', 'c', 'd'])  
del(b['b'])  
b
```

```
Out[18]: a    11  
c    33  
d    44  
dtype: int64
```

```
In [90]: ## 시리즈 조작  
a = pd.Series([11, 22, 33, 44])  
  
a['e'] = 50:           #새로운 값 추가  
print(a)  
a['e'] = 55:           #새로운 값으로 변경  
print(a)  
del(a['e']):          #값 제거  
print(a)
```

```
0    11  
1    22  
2    33  
3    44  
e    50  
dtype: int64  
0    11  
1    22  
2    33  
3    44  
e    55  
dtype: int64  
0    11  
1    22  
2    33  
3    44  
dtype: int64
```

반복문으로 읽어내기

```
In [95]: ## 시리즈를 for 반복문으로 읽어내기  
a = pd.Series([11, 22, 33, 44]) #index 자동 부여  
for x in a: #range() 대신에 적용  
    print(x)
```

```
11  
22  
33  
44
```

```
In [25]: ## 시리즈를 for 반복문으로 읽어내기  
a = pd.Series([11, 22, 33, 44]) #index 자동 부여  
for i, v in a.items(): #range() 대신에 적용  
    print(i, v)
```

```
0 11  
1 22  
2 33  
3 44
```

pandas 함수의 출력 결과 활용하기

```
In [76]: ## pandas 함수 활용  
a = pd.Series([11, 22, 33, 44]) #index 자동 부여  
a.value_counts() #항목별 분포 수
```

```
Out[76]: 11    1  
22    1  
33    1  
44    1  
dtype: int64
```

```
In [77]: a.sum() #항목의 합
```

```
Out[77]: 110
```

```
In [61]: a.mean() #항목 평균
```

```
Out[61]: 27.5
```

```
In [62]: a.max()      #최고값 항목
```

```
Out[62]: 44
```

```
In [63]: a.min()      #최소값 항목
```

```
Out[63]: 11
```

```
In [ ]:
```

[실습-4] List로 Series 만들기

> List를 랜덤 수로 초기화 함. (range는 0~9까지)

> List의 길이는 30으로 함

> 각 숫자의 빈도 수를 index로 하는 Series 구성

```
In [69]: ## 리스트(List) 형 : []
## [실습] List를 랜덤 수로 초기화 함
## 랜덤 리스트 구성
import random
alist = []
for _ in range(30):
    x = random.randrange(0, 10)
    alist.append(x)
print(alist)

# 인덱스 리스트 idx 만들기
```

```
# 빈도 리스트 data 만들기
```

```
## pandas 패키지 활용 : 데이터 시리즈(2차원 자료 구조) 구성
```

```
import pandas as pd
ds = pd.Series(data, index = idx)
print(ds)

[1, 5, 2, 4, 1, 7, 1, 8, 0, 2, 4, 8, 6, 5, 9, 0, 9, 5, 7, 4, 7, 0, 8, 0, 1, 1, 8, 3, 8, 2]
0    4
1    5
2    3
3    1
4    3
5    3
6    1
7    3
8    5
9    2
dtype: int64
```

```
In [ ]: ## Series로 빈도수 그래프 그리기
import seaborn as sns
sns.countplot(x = ds)
```

```
In [ ]:
```

○ 데이터 프레임(Data Frame) 형

- > 행과 열로 나열되는 2차원적 자료 구조
- > 행과 열 값들은 딕셔너리 구조를 활용하여 초기화 한다.
- > 하나의 열 값들은 시리즈형으로 추출하여 사용 가능하다.
- > pandas 패키지를 통해 사용할 수 있다.

<> 생성하기

```
In [71]: ## 데이터 프레임(Data Frame) 형 ##
## 생성: index 자동 부여
import pandas as pd
df = pd.DataFrame({'Eng' : [87, 79, 80], #열명:열값에 해당
                   'Mat' : [80, 90, 80]}) #행 index 자동 부여
df
```

```
Out[71]: Eng Mat
```

	Eng	Mat
0	87	80
1	79	90
2	80	80

```
In [70]: ## [Data Frame] 생성: index 강제 부여
```

```
import pandas as pd
df = pd.DataFrame({'Eng' : [87, 79, 80],
                    'Mat' : [80, 90, 80]},
                   index = ['Kims', 'Lees', 'Parks']) #index 강제 부여
df
```

```
Out[70]: Eng Mat
```

	Eng	Mat
Kims	87	80
Lees	79	90
Parks	80	80

<> 추출하기

> Data Frame에서 변수를 추출하면 Series형으로 반환

```
In [72]: ## [Data Frame] 조작 : 열 추출
```

```
x = df['Eng']; # 'Eng' 열 값을 시리즈(Series)로 추출, 결과는 Series
print(type(x))
x
```

```
<class 'pandas.core.series.Series'>
```

```
Out[72]: 0    87
1    79
2    80
Name: Eng, dtype: int64
```

```
In [74]: ## [Data Frame] 조작 : 열 추출
```

```
y = df[['Mat', 'Eng']] # ['Mat', 'Eng']는 List, 결과는 Data Frame
print(type(y))
y
```

```
<class 'pandas.core.frame.DataFrame'>
Out[74]:   Mat  Eng
0    80   87
1    90   79
2    80   80
```

<> 추가/수정 하기

> index가 없으면 추가, 있으면 수정

```
In [151...]: ## [Data Frame] 조작 : 열 추가
df['avg'] = (df['Eng'] + df['Mat']) / 2           # 'avg'열을 만들어서 평균 값 추가
df
```

```
Out[151]:   Eng  Mat  avg
Kims    87   80  83.5
Lees     79   90  84.5
Parks   80   80  80.0
```

<> 행, 열 제거하기

> .drop()

```
In [152...]: ## [Data Frame] 관련 함수 : 행, 열 제거
ddf = df.drop( 'Lees', axis = 'rows' )             # 행 이름으로 열 제거
ddf = ddf.drop( 'avg', axis = 'columns' )          # 컬럼 이름으로 컬럼 제거
ddf
```

Out[152]:

	Eng	Mat
Kims	87	80
Parks	80	80

pandas 함수의 출력 결과 활용하기

In [156...]

```
## [Data Frame] 관련 함수 사용  
df.value_counts()          # 'Eng' 열 값들의 개수 구하기
```

Out[156]:

```
Eng Mat avg  
79 90 84.5 1  
80 80 80.0 1  
87 80 83.5 1  
dtype: int64
```

In [157...]

```
df.mean()                  # 'Eng' 열 값들의 평균 구하기
```

Out[157]:

```
Eng    82.000000  
Mat    83.333333  
avg    82.666667  
dtype: float64
```

In [154...]

```
## [Data Frame] 관련 함수 사용  
df['Mat'].mean()          # 'Mat' 열 값들의 평균 구하기
```

Out[154]:

```
83.33333333333333
```

In []:

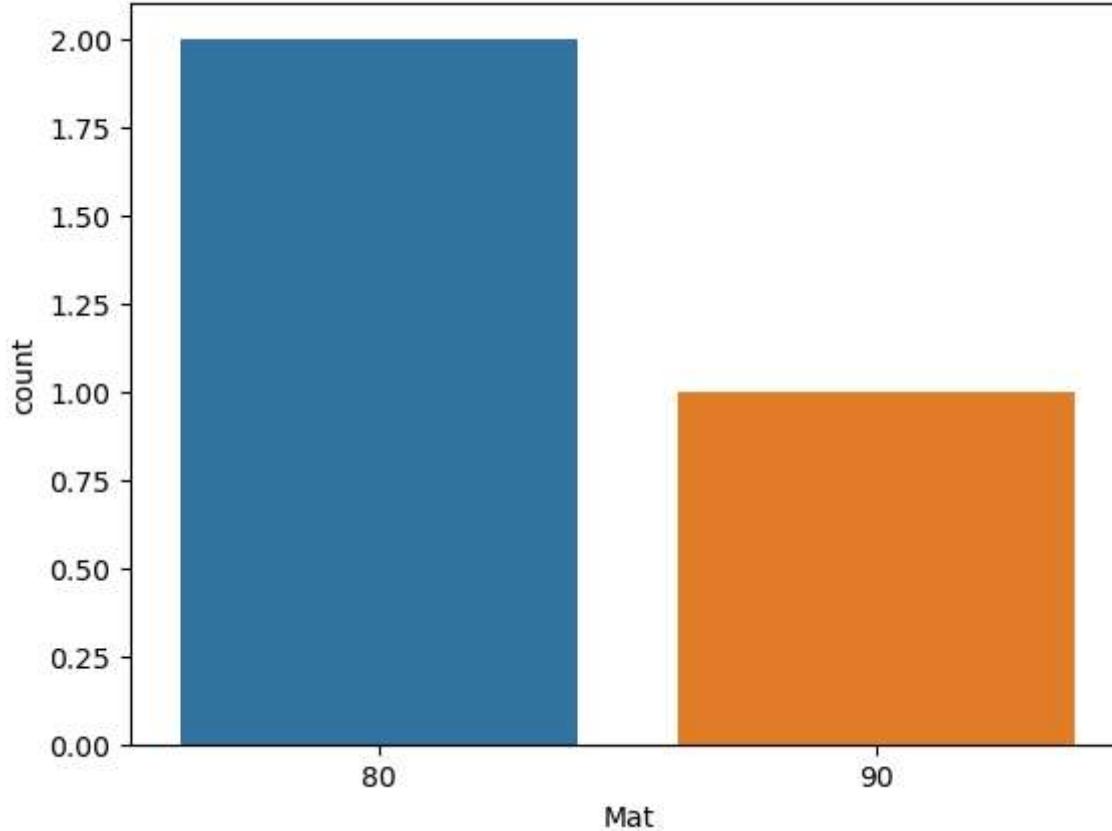
Data Frame으로 그래프 그리기

In [73]:

```
## [Data Frame] 그래프 그리기  
import seaborn as sns  
sns.countplot(data=df, x='Mat')
```

Out[73]:

```
<AxesSubplot:xlabel='Mat', ylabel='count'>
```



```
In [ ]: ## 함수 사용법 보기  
import pandas as pd  
pd.DataFrame?
```

```
In [ ]:
```

[과제] 위의 [실습-0], [실습-1],[실습-2],[실습-3] 문제 해결하기

(해결한 결과를 Ch17_학번.ipynb 파일로 LMS에 첨부로 제출하세요.)

[실습-0] 4행4열 리스트 초기화하기

[실습-1] List로 막대 그래프 그리기 (응용)

[실습-2] Tuple 항목 검색하기

[실습-3] Dictionary 항목 검색하기

In []: