
▣ Ch03 데이터 분석에 필요한 연장 챕기기

#DataStructure #List #Tuple #Dictionary
#lower() #upper() #capitalize() #title() #replace() #find() #split()
#sum() #max() #min() #lambda함수 #turtle #숫자야구게임

In []:

[03-1] 변하는 수, '변수' 이해하기

변수(Variable)

> 상수 값 대신에 이름으로 값을 대신해 사용하는 방식

- Python 자료 구조(변수) 종류

[내장 자료형]

>> 스칼라(Scalar) 형 : int, float, str, bool 등

>> 비스칼라형 : list, tuple, dictionary 등

[외장 자료형] Pandas 자료형

>> 시리즈(Series) 형 : 1차원 나열형 자료

>> 데이터 프레임(Data Frame) 형 : 2차원 나열형 자료

In []:

○ 스칼라(Scalar) 형

> 하나의 값 만으로 구성된 자료 구조

>> int, float, str, bool 등

In [1]:

```
## 스칼라(Scalar) 형 ##
```

```
a, b, c, d = 3, 3.15, 'hello, world!', bool(True)
print(type(a))
print(type(b))
print(type(c))
print(type(d))
```

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

<> 자료형의 메서드 사용하기

> Python에서는 자료형 구조가 Class 형태로 만들어져 관리가 되므로 자료형의 메서드를 사용할 수 있다.

>> 각 메서드는 처리된 결과 문자열을 반환(원 자료는 변형시키지 않음)

In [15]:

```
c = 'HELLO, WORLD!'
c.lower()          # 'HELLO, WORLD!'
```

Out[15]:

```
'hello, world!'
```

In [16]:

```
c = 'hello, world!'
c.upper()          # 'HELLO, WORLD!'
```

Out[16]:

```
'HELLO, WORLD!'
```

In [17]:

```
c = 'hello, world!'
c.capitalize()      # 'Hello, world!' 문장의 첫 문자를 대문자로
```

```
Out[17]: 'Hello, world!'
```

```
In [18]: c = 'hello, world!'  
c.title()          # 'Hello, World!' 단어별 첫 문자를 대문자로
```

```
Out[18]: 'Hello, World!'
```

```
In [22]: c = 'hello, world!'  
c.replace('o', 'X')    # 'Hello, World!!!' 해당 문자열을 찾아 모두 대체
```

```
Out[22]: 'hellX, wXrld!'
```

```
In [26]: c = 'hello, world!'  
c.find(',')          # 찾는 문자열의 시작 index 값(0부터 시작)
```

```
Out[26]: 5
```

```
In [27]: c.split(',')      # 원하는 문자열을 분리자로하여 분리, 결과는 리스트 형태로 반환
```

```
Out[27]: ['hello', 'world!']
```

```
In [ ]:
```

[] 비스칼라(Scalar) 형

> 여러 개의 값을 사용할 수 있도록 구성된 자료 구조 : **list, tuple, dictionary** 등

>> **List [item, ...]** : **index**로 접근하는 방식, 데이터 항목 수정 가능 (**index**는 자동 부여)

>> **Tuple (item, ...)** : **index**로 접근하는 방식, 데이터 항목 수정 불가능 (**index**는 자동 부여)

>> **Dictionary {key:item, ...}**: **Key**로 접근하는 방식, 데이터 항목 수정 가능 (**Key**는 사용자가 부여)

List

```
In [29]: ## 리스트(List) 형 : []  
## 생성 및 검색
```

```
a = [1, 2, 'a', 'b'] #리스트 생성 및 초기화
print(type(a))
a
<class 'list'>
[1, 2, 'a', 'b']
```

```
In [42]: ## 리스트(List) 형 : []
## 생성 및 검색
a = [1, 2, 'a', 'b'] #리스트 생성 및 초기화
a.append('c') #맨 마지막 항목으로 추가
a
Out[42]: [1, 2, 'a', 'b', 'c']
```

```
In [43]: a.index('c') #'c'항목의 index 값 반환
Out[43]: 4
```

```
In [46]: a[2:] #index 2에서 끝까지 추출
Out[46]: ['a', 'b', 'c']
```

```
In [82]: ## 리스트를 반복문으로 접근하기
a = [1, 2, 3, 4, 5]
for x in a: # 리스트 a의 각 항목을 x로 반환
    print(x, end=' ')
1 2 3 4 5
```

Tuple

```
In [38]: ## Tuple 형 : ()
a = (1, 2, 3, 4, 5) #튜플의 생성 및 초기화
print(type(a))
a
<class 'tuple'>
(1, 2, 3, 4, 5)
```

```
In [39]: a.index(1) #1 항목의 index 값 반환
```

```
Out[39]: 0
```

```
In [41]: a[:3] #index 처음부터 3 전까지 추출
```

```
Out[41]: (1, 2, 3)
```

```
In [84]: ## 튜플을 반복문으로 접근하기  
a = (1, 2, 3, 4, 5)  
for x in a: # a의 각 항목을 x로 반환  
    print(x, end=' ')
```

```
1 2 3 4 5
```

Dictionary

```
In [47]: ## 딕셔너리(Dictionary) 형 : {}  
## Key : Value로 생성  
a = {'A':90, 'B':80, 'C':70, 'D':60} #튜플의 생성 및 초기화  
print(type(a))  
a
```

```
<class 'dict'>
```

```
Out[47]: {'A': 90, 'B': 80, 'C': 70, 'D': 60}
```

```
In [54]: a['B'] #Key로 item 추출(못찾으면 에러 메시지 출력)
```

```
Out[54]: 80
```

```
In [56]: a.get('B') #Key로 item 추출(못찾아도 에러 메시지 출력 안함)
```

```
Out[56]: 80
```

```
In [86]: ## 딕셔너리를 반복문으로 접근하기  
a = {'A':90, 'B':80, 'C':70, 'D':60}  
for x in a.items(): # a의 각 항목을 x로 반환  
    print(x, end=' ')
```

```
('A', 90) ('B', 80) ('C', 70) ('D', 60)
```

```
In [92]: for k, v in a.items(): # a의 각 항목을 x로 반환  
    print(k, ':', v, end=', ')
```

A : 90, B : 80, C : 70, D : 60,

```
In [88]: for x in a.keys(): # a의 각 Key 값을 x로 반환  
    print(x, end=' ')
```

A B C D

```
In [89]: for x in a.values(): # a의 각 value 값을 x로 반환  
    print(x, end=' ')
```

90 80 70 60

```
In [ ]:
```

[03-2] 마술 상자 같은 '함수' 이해하기

▣ 내장 함수

```
In [57]: ## 내장 함수  
x = [1, 2, 3, 4]  
print(sum(x))  
print(max(x))  
print(min(x))
```

10
4
1

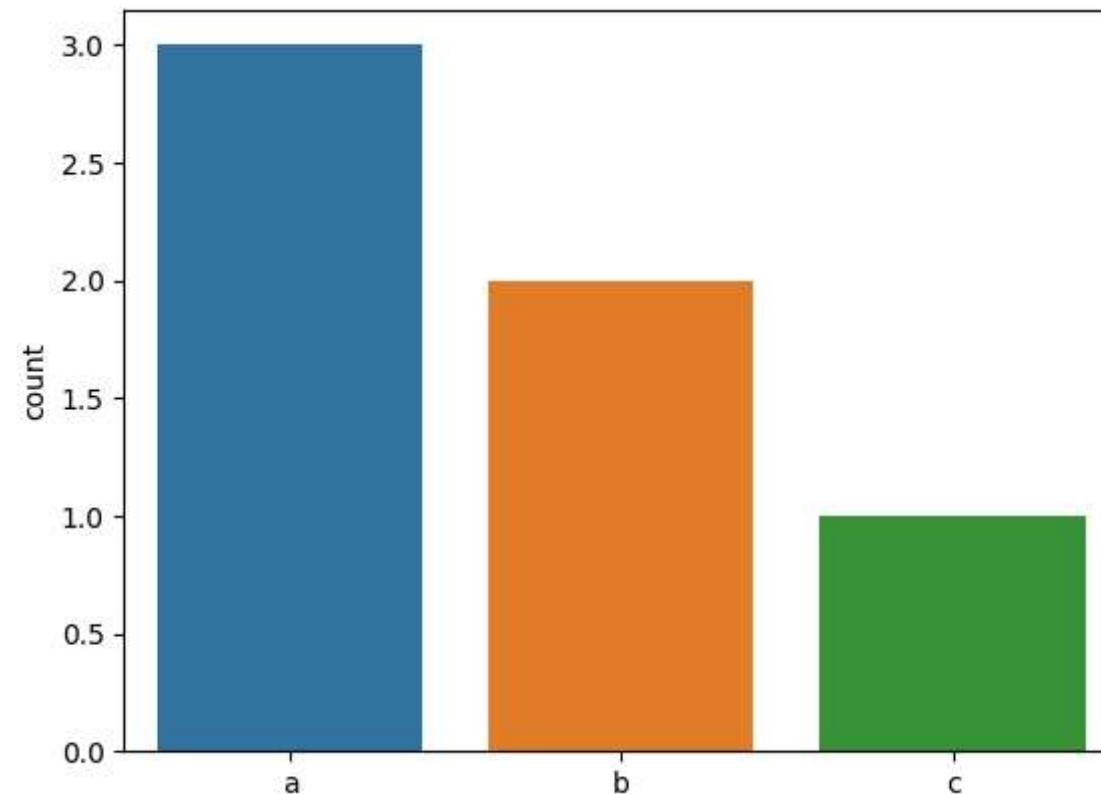
▣ 외장 함수

> 패키지 임포트 후 사용

```
In [60]: ## 외장 함수  
import seaborn as sns  
  
var = ['a', 'b', 'c', 'a', 'b', 'a']  
sns.countplot(x = var)
```

```
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[60]: <Axes: ylabel='count'>



[] lamda 함수

>> 함수 객체를 반환하는 함수

>> return문이 필요없는 함수

>> 간단한 함수를 선언하면서, 바로 실행하도록 하는 용도로도 사용

```
In [74]: ## [lambda 함수 선언] 매개변수 x, y를 사용하여 x+y를 반환하는 lambda 함수 선언  
lambda x, y:x+y #lambda 함수는 실행 후 함수 객체를 반환
```

```
Out[74]: <function __main__.<lambda>(x, y)>
```

```
In [75]: ## [lambda 함수 선언] 선언과 동시에 실행  
(lambda x,y:x*y)(3,4) #선언과 동시에 실행
```

```
Out[75]: 12
```

```
In [76]: ## [lambda 함수 선언]  
multi=lambda x,y:x*y #선언: lambda 함수를 선언하여 함수 객체로 반환  
multi(3,4) #실행: 함수 객체를 통해 lambda 함수 실행
```

```
Out[76]: 12
```

```
In [77]: ## [lambda 함수 선언] 매개변수에 기본값 사용  
multi=lambda x=3,y=4:x*y #매개변수에 기본값 사용  
multi(4)
```

```
Out[77]: 16
```

lambda 함수 활용

```
In [101...]: ## [lambda 함수 활용] 2의 지수승  
numbers = [1, 2, 3, 4, 5]  
squared = list(map(lambda x: 2**x, numbers)) # map(f, d)는 f 함수에 d 데이터 항목을 반복 적용 함수  
print(squared)
```

```
[2, 4, 8, 16, 32]
```

```
In [99]: ## [lambda 함수 활용] 성적순 정렬  
students = [
```

```
{ "name": "Alice", "grade": 85},
{ "name": "Bob", "grade": 90},
{ "name": "Charlie", "grade": 88}
]

# 학생들을 성적 순으로 정렬
sorted_students = sorted(students, key=lambda x: x['grade'], reverse=True)
print(sorted_students)

[{'name': 'Bob', 'grade': 90}, {'name': 'Charlie', 'grade': 88}, {'name': 'Alice', 'grade': 85}]
```

In []:

[03-3] 함수 꾸러미, '패키지' 이해하기

▣ 함수, 모듈, 패키지

- > **함수(Function)** : 단위 기능을 수행하는 명령의 집합
- > **모듈(Module)** : 함수의 모음으로 함수보다는 좀 더 큰 단위의 작업을 수행 (독자적으로 컴파일되어 실행 가능)
- > **패키지(Package)** : 관련이 깊은 모듈들을 하나로 모아놓은 것

<> import 패키지명.모듈명으로 import하여 함수 사용하기

```
In [112...]
## metrics 모듈의 accuracy_score() 사용하기
import sklearn.metrics
predicted = [0, 1, 1, 0, 1, 1, 0, 1, 1, 1] #예측값
actual = [0, 1, 1, 0, 1, 0, 1, 0, 1, 1] #실제값

# 정확도 계산
accuracy = accuracy_score(actual, predicted)
print("Accuracy:", accuracy)
```

Accuracy: 0.7

<> from 패키지명 import 모듈명으로 import하여 함수 사용하기

In [113]:

```
# sklearn 패키지의 metrics 모듈 로드하기
from sklearn import metrics
predicted = [0, 1, 1, 0, 1, 1, 0, 1, 1, 1] #예측값
actual = [0, 1, 1, 0, 1, 0, 1, 0, 1, 1] #실제값

# 정확도 계산
accuracy = accuracy_score(actual, predicted)
print("Accuracy:", accuracy)
```

Accuracy: 0.7

<> from 패키지명.모듈명 import 함수명으로 import하여 함수 사용하기

In [114]:

```
# sklearn 패키지 metrics 모듈의 accuracy_score() 로드하기
from sklearn.metrics import accuracy_score
predicted = [0, 1, 1, 0, 1, 1, 0, 1, 1, 1] #예측값
actual = [0, 1, 1, 0, 1, 0, 1, 0, 1, 1] #실제값

# 정확도 계산
accuracy = accuracy_score(actual, predicted)
print("Accuracy:", accuracy)
```

Accuracy: 0.7

In []:

[실습] Turtle 모듈 활용

In [3]:

```
## [Turtle 모듈 활용]
import turtle as t
t.shape('turtle') #Turtle 모양

t.forward(100)
t.left(90)

t.exitonclick() # 실행 창을 닫지 않도록
```

[연습: Coding] Turtle 모듈 활용 n각형 그리기

>> import turtle 모듈을 활용

>> 키보드로 turtle이 그릴 n각형의 n을 입력받는다.

>> n이 2이하가 입력되면 작업을 종료한다.

```
In [14]: ## [Turtle 모듈 활용]
import turtle as t
t.shape('turtle') #Turtle 모양
n = int(input(">몇 각형(3~n)? "))

t.exitonclick() # 실행 창을 닫지 않도록
##### 메인 끝 #####
```

[연습: Coding] Turtle 모듈 활용 n각형 그리기 + 함수 호출로 그리기

>> 함수 호출을 통해 원하는 n각형을 그린다.

>> n값과 변의 이동 값을 전달

```
In [ ]: ## [Turtle 모듈 활용]
import turtle as t

##### 메인 시작 #####
t.shape('turtle') #Turtle 모양
n = int(input(">몇 각형(3~n)? "))

t.exitonclick() # 실행 창을 닫지 않도록
##### 메인 끝 #####
```

```
In [ ]:
```

[응용: Coding] 숫자 야구 게임

랜덤한 3자리 숫자 맞추기 게임을 완성하시오.

- > 랜덤한 3자리 정수 발생, 단 각 자리 수의 값은 같으면 안됨
- > 게이머는 추측하는 3자리 값을 키보드로 입력 (맞출 때까지)
- > 각 위치(**digit**)에서 값이 같으면 **Strike**, 값은 존재하나 위치가 다르면 **Ball**로 처리
- > 매회 ">>%d Strike, %d Ball"로 결과 제공
- > 게임 종료 조건 : 3 Strike 또는 입력 값 00

In []:

패키지 설치하기

In [116...]

```
pip install pydataset
```

```
Requirement already satisfied: pydataset in c:\Users\Wadmin\Anaconda3\lib\site-packages (0.2.0)
Requirement already satisfied: pandas in c:\Users\Wadmin\Anaconda3\lib\site-packages (from pydataset) (2.1.0)
Requirement already satisfied: pytz>=2020.1 in c:\Users\Wadmin\Anaconda3\lib\site-packages (from pandas->pydataset) (2022.1)
Requirement already satisfied: tzdata>=2022.1 in c:\Users\Wadmin\Anaconda3\lib\site-packages (from pandas->pydataset) (2023.3)
Requirement already satisfied: numpy>=1.22.4 in c:\Users\Wadmin\Anaconda3\lib\site-packages (from pandas->pydataset) (1.25.2)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\Users\Wadmin\Anaconda3\lib\site-packages (from pandas->pydataset) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\Users\Wadmin\Anaconda3\lib\site-packages (from python-dateutil>=2.8.2->pandas->pydataset) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

```
WARNING: Ignoring invalid distribution -atplotlib (c:\Users\Wadmin\Anaconda3\lib\site-packages)
```

<> 패키지 함수 사용하기

In [117]:

```
import pydataset  
pydataset.data()
```

Out[117]:

	dataset_id	title
0	AirPassengers	Monthly Airline Passenger Numbers 1949-1960
1	Bjsales	Sales Data with Leading Indicator
2	BOD	Biochemical Oxygen Demand
3	Formaldehyde	Determination of Formaldehyde
4	HairEyeColor	Hair and Eye Color of Statistics Students
...
752	VerbAgg	Verbal Aggression item responses
753	cake	Breakage Angle of Chocolate Cakes
754	cbpp	Contagious bovine pleuropneumonia
755	grouseticks	Data on red grouse ticks from Elston et al. 2001
756	sleepstudy	Reaction times in a sleep deprivation study

757 rows × 2 columns

In [3]:

```
df = pydataset.data('mtcars') # mtcars 데이터를 df 데이터 프레임 df에 할당  
df # df 출력
```

Out[3]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

In []: