

○ 자료 구조(Data Structure)

- > 값을 임시로 저장할 수 있는 변수(Variable)의 모양
- > 자료 구조는 다양한 재료(Value)를 담을 수 있는 그릇(Memory) 역할을 한다.
- > 그릇에 담겨진 재료를 제대로 사용하려면 담겨진 형태를 잘 알아야 한다.

○ Python 자료 구조 종류

[내장 자료형]

- >> 스칼라(Scalar) 형
- >> 리스트(list) 형
- >> 튜플(Tuple) 형
- >> 딕셔너리(Dictionary) 형

[외장 자료형] Pandas 자료형

- << 시리즈(Series) 형
- << 데이터 프레임(Data Frame) 형

☒ 스칼라(Scalar) 형

- > 하나의 값 만으로 구성된 자료 구조
- > Data Type : int, float, str, bool 등
- > 관련 함수 : int(), float(), str(), bool()

```
In [8]: ## 스칼라(Scalar) 형 ##
## 생성 및 초기화
a = 3           #int형 자료 구조 생성 및 초기화
type(a)         #자료구조 확인
```

```
Out[8]: int
```

```
In [7]: b = 4.15      #float형 자료 구조 생성 및 초기화
type(b)
```

```
Out[7]: float
```

```
In [6]: c = 'hello, world!'      #str형 자료 구조 생성 및 초기화
type(c)
```

```
Out[6]: str
```

○ Python에서는 저장할 자료의 형태에 따라 자료 구조가 결정된다.

> 같은 이름의 변수라 하더라도 대입하는 값의 형태가 달라지면, 변수의 자료 구조도 바뀜.

```
In [4]: a = 3  
type(a)
```

```
Out[4]: int
```

```
In [5]: a = 3.14  
type(a)
```

```
Out[5]: float
```

○ Python에서는 자료 구조가 모두 class 형태로 만들어진다.

```
In [9]: c.lower() # 'hello, world!'
```

```
Out[9]: 'hello, world!'
```

```
In [10]: c.upper() # 'HELLO, WORLD!'
```

```
Out[10]: 'HELLO, WORLD!'
```

```
In [11]: c.capitalize() # 'Hello, world!'
```

```
Out[11]: 'Hello, world!'
```

```
In [13]: c.title() # 'Hello, World!'
```

```
Out[13]: 'Hello, World!'
```

```
In [14]: c.replace('!', '!!!!') # 'Hello, World!!!'
```

```
Out[14]: 'hello, world!!!'
```

```
In [15]: c.find(',') # 5
```

```
Out[15]: 5
```

```
In [16]: c.split(',') # ['Hello', 'World!']
```

```
Out[16]: ['Hello', 'World!']
```

Class는?

> 객체(Object)를 생성하는데 사용하는 원형 틀에 해당한다.

> Class는 자료 구조들과 그 구조들을 사용하는 기능(Function)들의 정의로 구성되어 있다.

> Class를 사용해서 객체를 생성하면, Class에 정의된 자료 구조와 기능들을 상속 받아 객체가 사용할 수 있는 장점이 있다.

> 하나의 **Class**로 여러 개의 객체를 이름을 달리 하여 만들어 사용할 수 있다.

>> 위 예제에서

c = 'hello, world!' # 문자열 자료 구조에 해당하는 Class를 사용하여 객체 c를 만들어서 초기화 함.

c.upper() # c 객체는 상속 받은 upper() 함수(원래는 메소드라고 함)를 사용하여 각 문자를 대문자로 바꿀 수 있다.

In []:

☒ 비 스칼라 형

> 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

> 값의 삽입, 삭제가 가능하다.

> 각 값들은 [와] 안에 ,로 구분하여 초기화 한다.

In [8]:

```
## 리스트(List) 형 : [ ] ##
## 생성 및 검색
a = [1, 2, 'a', 'b']           #리스트 생성 및 초기화
print(a[0])    #리스트는 0번째부터 시작
print(a[-1])   #거꾸로 첫 번째
print(a[2:3])  #2번째부터 3번째 전까지
print(a[2:])   #2번째부터 끝까지
print(a[:2])   #처음부터 2번째 전까지
```

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

In [5]:

```
## 생성 및 조작
a.append('c')      #맨 마지막에 'c'를 추가
a.insert(2, 3)     #2번째에 값 3을 추가
a.remove(3)        #3을 찾아 제거
a.index('a')       #'a'의 위치 값 반환
a
```

Out[5]:

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

```
In [10]: ## 관련 함수: 정렬
a = [5, 3, 7, 8, 2, 1]
asort = sorted(a)      # 값들을 정렬
a
asort
```

```
Out[10]: [1, 2, 3, 5, 7, 8]
```

```
In [11]: ## 관련 함수: 제거, 갯 수
del(a[1])           # 1번째 값을 제거
a
len(a)
```

```
Out[11]: 5
```

```
In [21]: ## 리스트 읽어내기
a = [1, 2, 'a', 'b']      #리스트 생성 및 초기화
for x in a:                #range() 대신에 리스트 자료 적
    print(x)
```

```
1
2
a
b
```

```
In [7]: ## 리스트 언팩킹(unpacking)
a = [['A', 90], ['B', 80], ['C', 70]]
x1, x2 = a[0]  #unpacking
print(x1, x2)

for x1, x2 in a:  #unpacking 반복
    print(x1, x2)
```

```
A 90
A 90
B 80
C 70
```

○ 튜플(Tuple) 형 : ()

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

```
In [9]: ## 튜플(Tuple) 형 : () ##
## 생성 및 검색
a = (1, 2, 3, 4, 5)          #튜플의 생성 및 초기화
[a[1], a[2:4], a[2:], a[:2]]
```

```
Out[9]: [2, (3, 4), (3, 4, 5), (1, 2)]
```

```
In [6]: b = 10, 20, 30      #튜플의 패킹(Packing)
x, y, z = b                #튜플의 언패킹(Unpacking)
[x, y, z]
```

```
Out[6]: [10, 20, 30]
```

```
In [12]: ## 튜플 값 for 반복문으로 읽어내기
a = (1, 2, 3, 4, 5)      #튜플의 생성 및 초기화
for x in a:              #range() 대신에 튜플 자료 적용
    print(x)
```

```
1
2
3
4
5
```

```
In [9]: ## 튜플 적용 사례
grades = (["A+", [100, 95]], ["A", [94, 90]], ["B+", [89, 85]], ["B", [84, 80]],
           ["C+", [79, 75]], ["C", [74, 70]], ["D+", [69, 65]], ["D", [64, 60]],
           ["F", [59, 0]])
for grade, zone in grades:
    print("%3s : %3d ~ %2d" %(grade, zone[0], zone[1]))
```

```
A+ : 100 ~ 95
A : 94 ~ 90
B+ : 89 ~ 85
B : 84 ~ 80
C+ : 79 ~ 75
C : 74 ~ 70
D+ : 69 ~ 65
D : 64 ~ 60
F : 59 ~ 0
```

○ 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

```
In [16]: ## 딕셔너리(Dictionary) 형 : { } ##
## 생성 및 검색
a = {'A':90, 'B':80, 'C':70, 'D':60}          #딕셔너리 생성 및 초기화
a['C']
```

```
Out[16]: 70
```

```
In [17]: ## 조작
a['F'] = 40          #딕셔너리 항목 추가
a
```

```
Out[17]: {'A': 90, 'B': 80, 'C': 70, 'D': 60, 'F': 40}
```

```
In [18]: ## 관련 함수
del(a['F'])          #딕셔너리 항목 제거
a
```

```
Out[18]: {'A': 90, 'B': 80, 'C': 70, 'D': 60}
```

```
In [19]: ## 관련 함수
a.keys()             #키 값만 추출
a.values()           #값만 추출
a.items()            #키와 값을 추출
```

```
Out[19]: dict_items([('A', 90), ('B', 80), ('C', 70), ('D', 60)])
```

```
In [22]: ## 반복문으로 키 읽어내기
for x in a.keys():    #키 읽어내기
    print(x)
```

```
A
B
C
D
```

```
In [23]: ## 반복문으로 값 읽어내기
for x in a.values():          #range() 대신에 적용
    print(x)
```

```
90
80
70
60
```

```
In [24]: ## 반복문으로 키, 값 읽어내기
for x in a.items():           #range() 대신에 적용
    print(x)
```

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

▣ 외장 pandas 자료형

> **pandas**라는 패키지를 **import** 해서 사용 가능한 추가 자료 구조

> **Data Type : series, dataframe**

> 관련 함수 : **Series()**, **DataFrame()**

○ 시리즈(Series) 형

> 여러 값을 나열한 자료 구조

> 데이터 프레임의 데이터를 구성하는 열 값을 시리즈로 추출하여 조작할 수 있다.

> **pandas** 패키지를 통해 사용할 수 있다.

```
In [25]: ## 시리즈(Series) 형 ##
## 생성 및 검색
import pandas as pd
a = pd.Series([11, 22, 33, 44])
[a[1], a[1:3], a[2:], a[:2]]
```

```
Out[25]: [22,
          1    22
          2    33
         dtype: int64,
          2    33
          3    44
         dtype: int64,
          0    11
          1    22
         dtype: int64]
```

```
In [28]: ## 시리즈 조작
a = pd.Series([11, 22, 33, 44], index = ['a', 'b', 'c', 'd'])
[a[1], a['b']]
```

```
Out[28]: [22, 22]
```

```
In [29]: a.loc['a']           # location(지정한 index 값)으로 접근
a.iloc[0]            # location의 순서 번호(index)로 접근
```

```
Out[29]: 11
```

```
In [30]: a['e'] = 50;           #새로운 값 추가
a
```

```
Out[30]: a    11
          b    22
          c    33
          d    44
          e    50
         dtype: int64
```

```
In [31]: a['e'] = 55;           #새로운 값으로 변경
a
```

```
Out[31]: a    11
          b    22
          c    33
          d    44
          e    55
         dtype: int64
```

```
In [32]: del(a['e']);           #값 제거
a
```

```
Out[32]: a    11
          b    22
          c    33
          d    44
         dtype: int64
```

```
In [33]: ## 시리즈를 for 반복문으로 읽어내기
for x in a:           #range() 대신에 적용
    print(x)
```

```
11
22
33
44
```

○ 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.

```
In [34]: ## 데이터 프레임(Data Frame) 형 ##
## 생성 및 검색
import pandas as pd
df = pd.DataFrame({'Eng' : [87, 79, 80],
                   'Mat' : [80, 90, 86]})
df
```

Out[34]:

	Eng	Mat
0	87	80
1	79	90
2	80	86

```
In [36]: ## 생성 및 조작
```

```
df = pd.DataFrame({'Eng' : [87, 79, 80],
                   'Mat' : [80, 90, 86]},
                   index = ['Kims', 'Lees', 'Parks'])
df
```

Out[36]:

	Eng	Mat
Kims	87	80
Lees	79	90
Parks	80	86

```
In [37]: x = df['Eng']; # 'Eng' 열 값을 시리즈로 추출
```

```
Out[37]: Kims    87
Lees     79
Parks   80
Name: Eng, dtype: int64
```

```
In [38]: df['avg'] = (df['Eng'] + df['Mat']) / 2
```

'avg' 열을 만들어서 평균 값 추가

Out[38]:

	Eng	Mat	avg
Kims	87	80	83.5
Lees	79	90	84.5
Parks	80	86	83.0

```
In [39]: df = df[['Mat', 'Eng', 'avg']]
```

원하는 컬럼 순으로 재정렬

df

Out[39]:

	Mat	Eng	avg
Kims	80	87	83.5
Lees	90	79	84.5
Parks	86	80	83.0

In [43]:

```
## 관련 함수 사용  
x.mean()          # 'Eng' 열 값들의 평균 구하기  
x.value_counts()    # 'Eng' 열 값들의 개수 구하기
```

Out[43]:

```
87 ... 1  
79 ... 1  
80 ... 1  
Name: Eng, dtype: int64
```

In [44]:

```
sdf = df.sort_values(by = 'avg')           # 'avg' 컬럼 값 순으로 정렬  
sdf
```

Out[44]:

	Mat	Eng	avg
Parks	86	80	83.0
Kims	80	87	83.5
Lees	90	79	84.5

In [46]:

```
ddf = df.drop( 'avg', axis = 'columns' )           # 컬럼 이름으로 컬럼 제거  
ddf
```

Out[46]:

	Mat	Eng
Kims	80	87
Lees	90	79
Parks	86	80

In [47]:

```
df = df.rename(columns = { 'Mat' : 'Math' })           # 컬럼명 변경  
df
```

Out[47]:

	Math	Eng	avg
Kims	80	87	83.5
Lees	90	79	84.5
Parks	86	80	83.0

In [48]:

```
df.head(2)           # 데이터 행 상위 일부만 출력 (숫자 생략하면 알아서)  
df.tail(1)          # 데이터 행 하위 일부만 출력 (숫자 생략하면 알아서)
```

Out[48]:

	Math	Eng	avg
Parks	86	80	83.0

In []: