

# 군집자료형

- 수치 자료형
  - int : 1, 0, -10
  - float : 3.14
  - complex : 2+3j
- bool 자료형
  - True False
- 군집 자료형
  - str : "Hello"
  - list : [1, 2, 3]
  - tuple : (1, 2, 3)
  - dictionary : {'H':1.01, 'He' : 4.00 }
  - set : {1, 2, 3}

\* immutable(변경불가)

int, float, complex, bool, str, tuple

\* mutable(변경가능)

list, dictionary, set

# Immutable(변경불능) vs Mutable(변경가능)

## Immutable한 객체의 값을 변경한 경우

```
1 a = 10 # int는 immutable
2 print(id(a))
3
4 a = a + 1 # 값 변경
5 print(id(a))
```

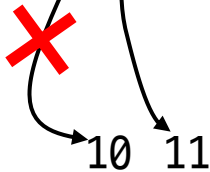
1566679237136  
1566679237168

id값(주소) 자체가 중요한 것이 아니라,  
id값(주소)이 변경된 것에 주목  
즉 변수 a가 가리키는 대상이 바뀌었다는 의미

stack

a

heap



## Mutable한 객체의 값을 변경한 경우

```
1 L = [10] # list는 mutable
2 print(id(L))
3
4 L.append(20) # 값 변경
5 print(id(L))
```

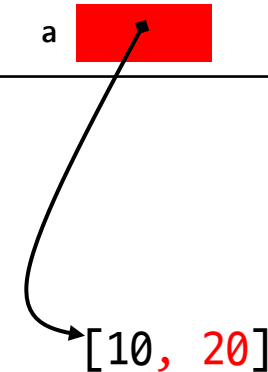
2319537125248  
2319537125248

id값(주소) 자체가 중요한 것이 아니라,  
리스트에 값을 추가하는 변경을 시켜도,  
id값(주소)는 그대로인 것에 주목

stack

a

heap



# 군집자료형 - str

- ▶ str 객체는 immutable(수정불가)한 객체
- ▶ str 객체는 큰따옴표("")나 작은따옴표('')
- ▶ 순서(인덱스 접근)가 있고, 중복이 가능

```
s = "Hello World"
```

'H'	'e'	'l'	'l'	'o'	' '	'W'	'o'	'r'	'l'	'd'
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]

```
print(len(s)) # 11(s의 길이)
```

```
print(s[2]) # l(s의 2번 인덱스 값)
```

```
print(s[2:5]) # llo(s의 2이상 5미만 인덱스 값)
```

```
s[2] = 'a' # 에러 발생!, 값 수정 불가
```

```
print("CN" + "SH") # CNSH
```

```
print("CNSH" * 3) # CNSHCNSHCNSH
```

```
# str의 메소드 함수
```

```
print(s.upper())
```

```
# 문자열 객체 s의 모든 문자를 대문자로
```

```
print(s.count('l'))
```

```
# 문자열 객체 s에서 l의 개수를 리턴
```

```
print(s.index('r'))
```

```
# 문자열 객체 s에서 r의 index값을 리턴
```

```
print(s.replace('l', 'L'))
```

```
# 문자열 객체 s에서 모든 l을 L로 치환
```

```
a, b = s.split()
```

```
# 문자열 객체 s를 공백을 기준으로 분리
```

# 군집자료형 - list

- ▶ list 객체는 mutable(수정가능)한 객체
- ▶ list 객체는 []로 표현
- ▶ 순서(인덱스 접근)가 있고, 중복이 가능
- ▶ 여러 자료형을 한 리스트에 포함 가능(L=[1, 3.5, "sun"])

```
L = [1, 2, 3]
```

1	2	3
[0]	[1]	[2]

```
print(len(L)) # 3
```

```
print(L[1]) # 2
```

```
print(L[1:3]) # [2, 3]
```

```
L[1] = 20 # L은 [1, 20, 3]이 됨
```

리스트에 적용가능한 유용한 내장함수  
sum(L), min(L), max(L)

```
L2 = [1, 2] + [3, 4] # [1, 2, 3, 4]
```

```
L3 = [1, 2] * 2 # [1, 2, 1, 2]
```

```
L.append(10) # 맨 뒤에 10을 추가 - 리턴값 없음  
L.insert(1, 10) # 인덱스 1위치에 10 추가 - 리턴값 없음  
L.remove(10) # 10을 제거 - 리턴값 없음  
L.pop() # 맨 뒤에 값을 제거하고 그 값을 리턴  
L.count(10) # 10의 개수를 리턴  
L.index(10) # 10이 있는 index 위치를 리턴  
L.sort() # 정렬(기본값 : 오름차순) - 리턴 값 없음  
L.reverse() # 순서 뒤집기 - 리턴값 없음
```

# 메소드의 리턴(반환값)의 유무에 따라

```
L = [1, 2, 3]  
print(L.append(3))  
print(L.count(3))
```

```
None  
2
```

# 군집자료형 – tuple 수정 불가능 리스트임

- ▶ tuple 객체는 immutable(수정불가)한 객체
- ▶ tuple 객체는 ()로 표현
- ▶ 순서(인덱스 접근)가 있고, 중복이 가능
- ▶ 여러 자료형을 한 리스트에 포함 가능(T=(1, 3.5, "sun"))

```
T = (1, 2, 3, 3)
```

1	2	3	3
[0]	[1]	[2]	[3]

```
print(T) # (1, 2, 3, 3)
```

```
print(len(T)) # 4
```

```
print(T[1]) # 2
```

```
print(T[1:3]) # (2, 3)
```

```
T[1] = 20 # 불가
```

```
print(sum(T), max(T), min(T)) # 9, 3, 1
```

```
print(T.count(3)) # 2
```

```
print(T.index(2)) # 1
```

```
T = (1, 2) + (3, 4) # (1, 2, 3, 4)
```

```
T = (1, 2) * 2 # (1, 2, 1, 2)
```

- 튜플은 거의 리스트와 유사 (단, 값 변경이 불가)
- 프로그래밍시 값이 변경되면 안되는 값들은 튜플형으로...
- 접근속도면에서 리스트보다 빠름
- 튜플을 수정/변경하는 `append()`, `remove()`, `sort()` 등의 메소드가 없다.



# 군집자료형 – set 수학의 집합과 유사

- ▶ set 객체는 mutable(수정가능)한 객체
- ▶ set 객체는 {키:값, 키:값, ...}로 표현 - 키와 값이 대응관계
- ▶ 순서가 없고(인덱스 접근 불가), 중복 불가

```
S1 = {1, 2, 3, 4, 5, 5, 4} # 중복된 값이 있으면 자동 제거  
S2 = {4, 5, 6, 7, 8}
```

```
print(S1) # {1, 2, 3, 4, 5} 중복된 값은 제거됨.
```

```
print(len(S1)) # 5
```

```
print(S1[1]) # 순서가 없으므로 index로 접근 불가
```

```
print(S1 & S2) # 교집합 {4, 5}
```

```
s1.intersection(s2)
```

```
print(S1 | S2) # 합집합 {1, 2, 3, 4, 5, 6, 7, 8}
```

```
s1.union(s2)
```

```
print(S1 - S2) # 차집합 {1, 2, 3}
```

```
s1.difference(s2)
```

```
S1.add(100) # 값 1개 추가 {1, 2, 3, 4, 5, 100}
```

```
S1.update({200, 300}) # 값 2개이상 추가 {1, 2, 3, 4, 5, 100, 200, 300}
```

```
S1.remove(1) # 원소 중 1 제거 {2, 3, 4, 5, 100, 200, 300}
```

```
print(S1) # {2, 3, 4, 5, 100, 200, 300}
```

# 군집자료형 – dict key:value 쌍들의 묶음

- ▶ dict 객체는 mutable(수정가능)한 객체
- ▶ dict 객체는 {키:값, 키:값, ...}로 표현 - 키와 값이 대응관계
- ▶ 순서가 없고(인덱스 접근 불가), 중복 불가

```
D = {"H":1.01, "He":4.00, "Li":6.94}
```

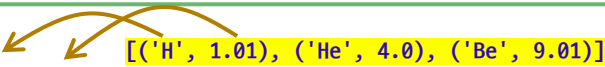
```
print(D) # {'H': 1.01, 'He': 4.0, 'Li': 6.94}
print(len(D)) # 3
```

```
print(D[1]) # 순서가 없으므로 index로 접근 불가
print(D["H"]) # 1.01 혹은 get()메소드를 사용해 print(D.get("H"))
```

```
D["Be"] = 9.01 # "Be":9.01 쌍을 추가
del D["Li"] # "Li":6.94 쌍을 삭제
```

```
print(D.items()) # dict_items([('H', 1.01), ('He', 4.0), ('Be', 9.01)])
print(D.keys()) # dict_keys(['H', 'He', 'Be'])
print(D.values()) # dict_values([1.01, 4.0, 9.01])
```

```
for k, v in D.items():
    print(k, v)
```



```
H 1.01
He 4.0
Be 9.01
```