

# 기초 인공지능 프로그래밍

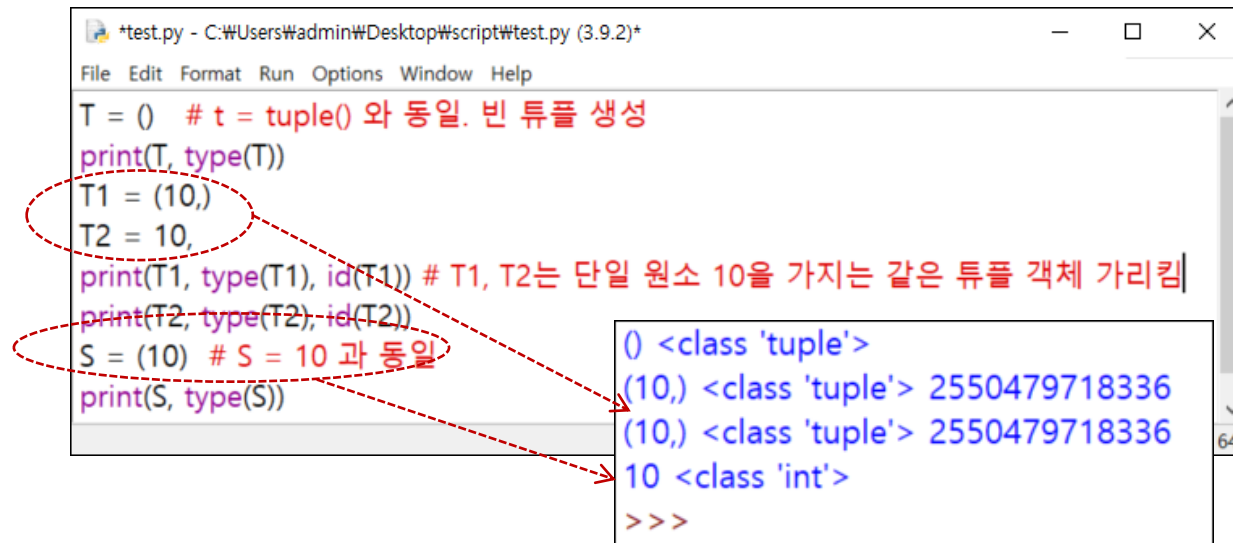
## 5장 튜플, 집합, 사전

# 튜플 (tuple)

- 리스트와 유사한 자료형
- 차이점은 원소의 추가/삭제/수정이 불가능(immutable 객체)하며 여러 발생함
- 원소 값이 변하지 않는 데이터 저장에 효율적
- 또한 반복 가능한 iterable 객체임
- 형식(syntax)
  - `tuple_name = (element1, element2, element3,...)`
    - 소괄호 ( )안에 콤마로 구분하여 원소들을 표시
    - 원소를 콤마로 구분하여 사용하면 괄호가 없어도 튜플로 인식
    - 원소로는 리스트와 동일하게 여러 자료형이 올 수 있음
- + 연결, \* 반복, in, not in 연산자, 내장 함수의 인자로 사용할 수 있음
- 튜플도 원소의 순서가 의미 있는(sequence) 자료형이며 인덱싱, 슬라이싱이 가능
  - 슬라이싱으로 원소를 참조할 수는 있지만 수정할 수는 없음
  - del 명령어로 튜플 자체 삭제만 가능 (원소 삭제에는 사용 못함)

# 튜플 (tuple)

- 튜플 생성
  - 콤마를 사용하여 원소 정의를 하면 괄호가 없어도 튜플로 인식
  - 원소가 하나인 튜플을 생성할 때도 반드시 콤마가 필요



The screenshot shows a Python IDE window titled '\*test.py - C:\Users\admin\Desktop\script\test.py (3.9.2)\*'. The code in the editor is as follows:

```
T = () # t = tuple() 와 동일. 빈 튜플 생성
print(T, type(T))
T1 = (10,)
T2 = 10,
print(T1, type(T1), id(T1)) # T1, T2는 단일 원소 10을 가지는 같은 튜플 객체 가리킴
print(T2, type(T2), id(T2))
S = (10) # S = 10 과 동일
print(S, type(S))
```

Red dashed circles and arrows highlight specific parts of the code and the output. The circles are around `T1 = (10,)`, `T2 = 10,`, and `S = (10)`. Arrows point from these circles to the corresponding output lines in the console:

- From `T1 = (10,)` to `(10,) <class 'tuple'> 2550479718336`
- From `T2 = 10,` to `(10,) <class 'tuple'> 2550479718336`
- From `S = (10)` to `10 <class 'int'>`

The console output shows the results of the print statements:

```
() <class 'tuple'>
(10,) <class 'tuple'> 2550479718336
(10,) <class 'tuple'> 2550479718336
10 <class 'int'>
>>>
```

# 튜플 (tuple)

```
test.py - C:\Users\Wadmin\Desktop\script\test.py (3.9.2)
File Edit Format Run Options Window Help
T3 = "David", 20, "CS" # 원소가 세 개인 튜플. 괄호는 생략 가능, packing 이라 함
print("T3 =", T3)
name, age, major = T3 # unpacking 이라 함
print("name = %s, age = %d, major = %s" %(name, age, major))
T4 = tuple("abc") # 문자열을 튜플로 변환
T5 = ("ABC",) # 문자열이 단일 원소인 튜플
T6 = ("ABC") # 문자열 할당. T6 = "ABC" 와 동일
print("T4 =", T4, ",", "T5 =", T5, ",", "T6 =", T6)
T = T4 + T5 # 두 개의 튜플로 새로운 튜플 생성
print("T =", T)
del T # 튜플 자체는 삭제 가능
```

- 여러 데이터를 하나의 튜플에 저장하는 것을 튜플 packing
- 튜플의 각 원소 값을 각각의 변수에 할당하는 것을 튜플 unpacking
- unpacking할 때는 튜플의 원소 수와 변수 수가 일치해야 함

```
T3 = ('David', 20, 'CS')
name = David, age = 20, major = CS
T4 = ('a', 'b', 'c') , T5 = ('ABC',) , T6 = ABC
T = ('a', 'b', 'c', 'ABC')
>>>
```

```
del t1[1] # Error (원소 삭제 불가)
t1[2] = 9 # Error (원소 수정 불가)
```

```
test.py - C:\Users\Wadmin\Desktop...
File Edit Format Run Options Window Help
x = 23; y = -23
print("x = %d, y = %d" %(x, y))
x, y = y, x # (x, y) = (y, x) 와 동일
print("x = %d, y = %d" %(x, y))
```

```
x = 23, y = -23
x = -23, y = 23
>>>
```

# 튜플 (tuple)

- 튜플 메소드

Method	Description (T : 튜플, x : 임의의 object)
T.index(x)	튜플에서 데이터 x의 인덱스를 반환
T.count(x)	튜플에서 데이터 x의 개수를 반환

- 문자열, 리스트와 동일하게 인덱싱, 슬라이싱 사용 가능(슬라이싱으로 데이터 참조만 가능)

```
test.py - C:\Users\Wadmin\Desktop\script\test.py (3.9.2)
File Edit Format Run Options Window Help
T = (1, [10, 20], [], "Sogang")
T[1][1] = 30 # 튜플의 원소가 리스트인 경우는 리스트의 원소 수정이 가능
T[2].append([100, 200]) # 빈 리스트에 원소 추가도 가능
print("T =", T)
T[1] = ["abc", 20] # 튜플의 원소인 리스트를 다른 값으로 수정하면 에러 발생.
|
T = (1, [10, 30], [[100, 200]], 'Sogang')
Traceback (most recent call last):
  File "C:\Users\Wadmin\Desktop\script\test.py", line 5, in <module>
    T[1] = ["abc", 20] # 튜플의 원소인 리스트를 다른 값으로 수정하면 에러 발생.
TypeError: 'tuple' object does not support item assignment
>>>
```

# 집합 (set)

- 데이터를 순서 없이 모아둔 자료형
- 형식(syntax)
  - `set_name = {element1, element2, element3,...}`
  - 중괄호 { }안에 콤마로 구분하여 원소들을 표시
  - 값을 변경할 수 없는 자료형만 원소로 올 수 있음(리스트형은 불가능)
- 집합 자료형 특징
  - 같은 값을 가지는 원소는 불가능(중복 불가능) : 중복 데이터를 걸러주는 기능을 함
  - 원소 간에 순서가 없음(Unordered) : 인덱싱, 슬라이싱 불가능
  - 원소의 추가 / 삭제 가능, in / not in 연산자 가능
  - set() 함수는 인자로 받은 자료형의 원소들을 자신의 원소로 하는 집합 생성
  - 공집합을 표기할 때는 set()으로 표기

# 집합 메소드

Method	Description (A, B : 집합, x : 임의의 object)
A.add(x)	데이터 x를 A의 원소로 추가
A.clear( )	A의 모든 원소를 제거. A를 공집합으로 만듦
B = A.copy()	A를 B로 복사(shallow copy)
A.discard(x)	집합 A에서 원소 x를 제거. 없는 원소를 삭제하려고 할 때에도 에러 발생하지 않음( $x \notin A$ 이면 무시)
A.remove(x)	집합 A에서 원소 x를 제거. 없는 원소를 삭제하려고 하면 KeyError가 발생( $x \notin A$ 이면 KeyError)
A.pop( )	집합 A에서 임의의 원소를 하나 지우고 그 값을 반환(A가 공집합이면 KeyError). 순서가 없기 때문에 임의의 원소 가져옴
A.union(B)	$A \cup B$ 를 반환 ( $A \mid B$ 와 동일)
A.intersection(B)	$A \cap B$ 를 반환 ( $A \& B$ 와 동일)
A.difference(B)	B에는 없고 A에만 있는 원소의 집합 반환 ( $A - B$ 와 동일)
A.isdisjoint(B)	$A \cap B = \emptyset$ 이면 True
A.issubset(B)	$A \subseteq B$ 이면 True
A.issuperset(B)	$A \supseteq B$ 이면 True
A.update(B)	A를 $A \cup B$ 로 갱신
A.intersection_update(B)	A를 $A \cap B$ 로 갱신
A.difference_update(B)	A를 $A - B$ 로 갱신
A.symmetric_difference(B)	$A \cup B$ 에는 있으나 $A \cap B$ 에는 없는 원소의 집합 반환( $A \wedge B$ 와 동일)
A.symmetric_difference_update(B)	A를 $(A \cup B) - (A \cap B)$ 로 갱신

# 집합 메소드 사용 예시

```
chap4.py - C:\Users\admin\Desktop\script\chap4.py (3.9.2)
File Edit Format Run Options Window Help
S1 = set() # 공집합 생성
S2 = set([10, 3, "abc", 4, 10]) # 리스트 원소로 집합 생성 (중복된 데이터 제거)
S3 = set("Good!! ^^")
print(S1)
print(S2)
print(S3)
S2.add("abc") # 원소 추가. 이미 있는 데이터를 추가하면 변동이 없음
S2.add("def")
S2.update([10, 20, 30]) # 다른 객체의 원소들을 추가. 존재하는 데이터를 추가하면 변동이 없음
print(S2)
S2.remove(30) # 인자로 받은 값이 존재하면 삭제. 없는 데이터이면 KeyError 발생
print(S2)
S2.discard(30) # 인자로 받은 값 삭제. 없는 값이면 무시(Error 발생 없음)
print(S2)
n = S2.pop() # 집합에서 임의의 원소를 삭제하면서 그 값을 반환
print(n)
print(S2)
S2.clear() # 집합 S1을 공집합으로 만들
print(S2)
```

- 공집합을 나타냄.  
공집합을 {}로 표기하면 안됨
- 중괄호 {}는 사전 자료형에도 사용되며 빈사전을 의미

```
set()
{10, 3, 4, 'abc'}
{'^', 'o', ' ', 'd', '!', 'G'}
{3, 4, 'abc', 10, 'def', 20, 30}
{3, 4, 'abc', 10, 'def', 20}
{3, 4, 'abc', 10, 'def', 20}
{3, 4, 'abc', 10, 'def', 20}
3
{4, 'abc', 10, 'def', 20}
set()
>>>
```



# 집합 메소드 사용 예시

```
chap4.py - C:\Users\admin\Desktop\script\chap4.py (3.9.2)
File Edit Format Run Options Window Help
L = [1,1,2,2]; B = {3,4,5,6}
A = set(L)
print(A)
A = A.union({3,4})
print(A)
C = A | B      # 합집합, union
D = A & B      # 교집합, intersection
E = A.difference(B) # E = A - B 와 동일, 차집합
print(C)
print(D)
print(E)
S = {10, 20, [100, 200]} # 에러 : 리스트는 집합의 원소로 올 수 없음
|
{1, 2}
{1, 2, 3, 4}
{1, 2, 3, 4, 5, 6}
{3, 4}
{1, 2}
Traceback (most recent call last):
  File "C:\Users\admin\Desktop\script\chap4.py", line 12, in <module>
    S = {10, 20, [100, 200]} # 에러 : 리스트는 집합의 원소로 올 수 없음
TypeError: unhashable type: 'list'
>>>
```

# 사전 (dictionary)

- 사전은 키(key)와 값(value)의 쌍을 저장하는 순서가 없는 자료형
- 형식(syntax)
  - `dic_name = {key1 : val1, key2 : val2, key3 : val3,...}`
  - 중괄호 { }안에 콤마로 구분하여 키와 값의 쌍을 표시
  - key에는 수정할 수 없는 immutable 자료형 (정수, 실수, 문자열, 튜플)만 허용되며, 수정 가능한 mutable한 자료형 (리스트, 집합, 사전)은 올 수 없음
  - value에는 어떤 자료형도 가능
  - 하나의 사전에서 중복되는 key 값이 존재할 수 없음
- 사전 자료형 특징
  - key 값으로 이에 대응하는 value 값을 얻을 수 있음
  - 인덱싱 기호 [ ] 안에 인덱스 번호가 아닌 key 값을 주어 해당 value 값을 참조함
  - 원소 간에 순서가 없기 때문에(unordered) 인덱싱, 슬라이싱 지원이 안됨
  - 원소의 추가/삭제 가능하며 in/not in 연산자 가능
  - + 연결 연산자, \* 반복 연산자는 사용할 수 없음

# 사전 (dictionary)

The screenshot shows a Python IDE window titled 'test.py - C:\Users\SOGANG\Desktop\test.py (3.9.2)'. The code in the editor demonstrates dictionary operations with several annotations:

```
classInfo = {} #빈 사전 생성 classInfo = dict() 와 동일
print("classInfo = ", classInfo)
classInfo["class 1"] = 50 # 빈 사전에 새로운 원소 추가
classInfo["class 2"] = 30
print("classInfo = ", classInfo)
classInfo["class 3"] = 50
classInfo["class 2"] = 20 # 이미 존재하는 key 값이면 새로운 value 값으로 수정
print("classInfo = ", classInfo)
num = classInfo["class 2"] # 찾고자 하는 key 값으로 value 값 참조
print("The student number of class 2 is %d" %num)
del classInfo["class 1"] # key 값 "class 1"인 원소 제거
print("classInfo = ", classInfo)
```

Annotations and arrows point to specific lines:

- Red arrow from `classInfo["class 1"] = 50` to: **D[key] = value**  
# 사전 D에 key가 존재하지 않으면, key : value 원소 추가
- Red arrow from `classInfo["class 2"] = 20` to: **D[key] = value**  
# 사전 D에 key가 이미 존재한다면 value 값으로 변경

A console output window at the bottom shows the execution results:

```
classInfo = {}
classInfo = {'class 1': 50, 'class 2': 30}
classInfo = {'class 1': 50, 'class 2': 20, 'class 3': 50}
The student number of class 2 is 20
classInfo = {'class 2': 20, 'class 3': 50}
>>>
```

# 사전 (dictionary)

```
test.py - C:\Users\SOGANGW\Desktop\test.py (3.9.2)
File Edit Format Run Options Window Help
D1 = dict([(1,'a'), (2,'b')]) # tuple, set, list 등 iterable한 자료형의
D2= dict({1,'a'}, {2,'b'}) # 원소가 쌍으로 구성되어 있는 경우
D3= dict({1, 'a'}, {2, 'b'}) # 사전 생성할 수 있음
print(D1, D2, D3) # D1, D2, D3는 동일한 데이터를 갖는 사전 |

{1: 'a', 2: 'b'} {1: 'a', 2: 'b'} {1: 'a', 2: 'b'}
>>>
```

set 자료형은 순서가 없기 때문에  
임의의 원소가 key로 설정됨

# 사전 (dictionary)

- 아래 내장 함수들은 리스트 등에서 이미 언급한 것들임

function	Description (인수 D : 사전)
<b>len(D)</b>	D의 원소 개수를 반환
<b>sorted(D)</b>	D의 key를 정렬한 리스트 반환. D는 불변이고 오름차순 또는 내림차순으로 정렬 가능. 예) sorted(D, reverse=True)
<b>list(D)</b>	D의 key들을 리스트로 변환하여 반환
<b>set(D)</b>	D의 key들을 집합으로 변환하여 반환
<b>tuple(D)</b>	D의 key들을 튜플로 변환하여 반환

# 사전 메소드

Method	Description ( D : 사전 )
<b>D.clear( )</b>	D의 모든 원소를 삭제. D = {}가 됨
<b>D.items( )</b>	key와 value의 쌍을 튜플로 묶은 값을 dict_items 객체로 반환. 튜플(key, value)를 원소로 하는 리스트를 만들거나 (key, value)가 필요한 반복문에서 사용
<b>D.keys( )</b>	딕셔너리 D의 Key만을 모아서 dict_keys 객체로 반환. 필요한 반복문에서 사용
<b>D.values( )</b>	value로 구성된 dict_values 객체를 반환
<b>D.get(key) or D.get(key, d)</b>	key에 대한 value 값 반환. 존재하지 않는 key이면 d 반환, d가 주어지지 않았으면 <b>None</b> 반환
<b>D.pop(key) or D.pop(key, d)</b>	key에 대한 value 반환하고 해당 원소를 삭제. 존재하지 않는 key이면 d 반환, d가 주어지지 않았으면 <b>KeyError</b>
<b>D.copy( )</b>	D를 복사하여 반환 (shallow copy).
<b>D.update(other)</b>	존재하는 key이면 value를 갱신, 없으면 쌍을 추가. 예) D.update([('a',2),('b',3)]) #D에 없으면 ('a',2)와 ('b',3)를 추가.

반환값 dict\_keys 객체는 **list(a.keys())**와 같은 명령어로 리스트로 변환 가능. 리스트로 변환하지 않더라도 기본적인 반복문(예: for문)에 적용할 수 있음(dict\_values, dict\_items 객체도 동일)

# 사전 메소드 사용 예시

chap4.py - C:\Users\Wadmin\Desktop\script\chap4.py (3.9.2)

File Edit Format Run Options Window Help

```
grade = {'Kim': 89, 'Park': 45, 'Lee': 78}
num = grade.get('Kim') # num = grade['Kim'] 와 동일
print(num, grade)
print("#####")
num = grade.pop('Kim') # key 값이 'Kim'인 원소 삭제하면서 value 값 반환
print(num, grade)
print("#####")
grade["Kang"] = 55 # key 값이 'Kang', value 값이 55인 원소 추가
print(grade)
print("#####")
dictKeyClass = grade.keys() # 사전의 모든 key 값들을 dict_keys 객체로 반환
print(dictKeyClass, type(dictKeyClass))
keyL = list(dictKeyClass) # dict_keys 객체를 편리한 리스트 객체로 변환
print(keyL, type(keyL))
itemL = list(grade.items()) # 사전의 모든 key, value 쌍을 튜플로 묶은 값들이
# 저장된 dict_items 객체를 리스트로 변환

print(itemL)
print("Kang" in grade)
print(55 in grade) # 사전의 key 값에서만 존재 여부 검사
```

variable = D[x] 와 variable = D.get[x] 차이

: 존재하지 않는 key 값 x일 경우, 메소드 D.get[x]는 None 반환하며 D[x]는 key 값 오류 발생.

dict\_keys, dict\_items, dict\_values 객체로 받은 값은 리스트와 같은 자료형으로 변환하여 사용 (리스트의 메소드를 적용하는 것이 코딩에 유용하기 때문)

```
89 {'Kim': 89, 'Park': 45, 'Lee': 78}
#####
89 {'Park': 45, 'Lee': 78}
#####
{'Park': 45, 'Lee': 78, 'Kang': 55}
#####
dict_keys(['Park', 'Lee', 'Kang']) <class 'dict_keys'>
['Park', 'Lee', 'Kang'] <class 'list'>
[('Park', 45), ('Lee', 78), ('Kang', 55)]
True
False
>>>
```

# 사전 메소드 사용 예시

```
chap4.py - C:\Users\wadmin\Desktop\script\chap4.py (3.9.2)
File Edit Format Run Options Window Help
D = {'a': 1, 'b': 2, 'c': 3, 'd': 4}
print(len(D))    # D의 원소 개수
keyList = sorted(D)    # D의 key 값으로 오름차순 정렬한 리스트
print(keyList)
print("#####")
print(D['b'])    # key가 'b'인 원소의 value 참조
print(D.get('b'))    # D['b']와 동일
print(D.get('e'))    # 사전 D에 없기 때문에 None 반환. D['e']로 할 경우 KeyError 발생
print(D.get('e', 0))    # 사전에 없는 key 값일 경우 0 을 반환하도록 설정(다른 값 가능)
print(D.pop('c'))    # key 값 'c'의 value 반환하면서 원소 'c':3 제거
print(D)
print(D.pop('c', 1))    # key 값 'c'가 없는 경우 value 값으로 1 반환 하도록 설정(다른 값 가능)
                        # D.pop('c')로 실행하면 KeyError (사전에 없기 때문에) 발생
```

```
4
['a', 'b', 'c', 'd']
#####
2
2
None
0
3
{'a': 1, 'b': 2, 'd': 4}
1
>>>
```