

세상의 속도를
따라잡고 싶다면



점프 투 파이썬

박응용 지음(위키독스 운영자)



파이썬 프로그래밍의 기초, 자료형



02-5 딕셔너리 자료형

02-6 집합 자료형

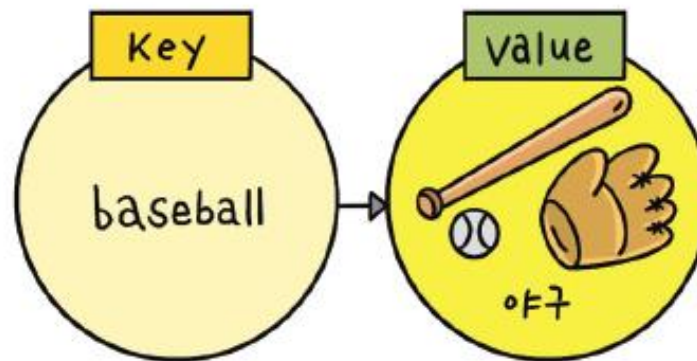
02-7 불 자료형

02-8 자료형의 값을 저장하는 공간, 변수



■ 딕셔너리(dictionary)란?

- 대응 관계를 나타내는 자료형
- 연관 배열(associative array) 또는 해시(hash)
- Key와 Value를 한 쌍으로 갖는 자료형
- 순차적으로(sequential) 해당 요솟값을 구하지 않고, Key를 통해 Value를 바로 얻는 것이 특징



■ 딕셔너리는 어떻게 만들까?

- Key와 Value의 쌍 여러 개 (Key : Value)가 { }로 둘러싸임
- 각 요소는 쉼표(,)로 구분됨

```
{Key1: Value1, Key2: Value2, Key3: Value3, ...}
```

```
>>> dic = {'name': 'pey', 'phone': '010-9999-1234', 'birth': '1118'}
```

```
>>> a = {1: 'hi'}
```

```
>>> a = {'a': [1, 2, 3]}
```

■ 딕셔너리 쌍 추가, 삭제하기

■ 딕셔너리 쌍 추가하기

```
>>> a = {1: 'a'}  
>>> a[2] = 'b' ← {2: 'b'} 쌍 추가  
>>> a  
{1: 'a', 2: 'b'}
```

```
>>> a['name'] = 'pey' ← {'name': 'pey'} 쌍 추가  
>>> a  
{1: 'a', 2: 'b', 'name': 'pey'}
```

```
>>> a[3] = [1, 2, 3] ← {3: [1, 2, 3]} 쌍 추가  
>>> a  
{1: 'a', 2: 'b', 'name': 'pey', 3: [1, 2, 3]}
```

■ 딕셔너리 요소 삭제하기

■ del 함수 사용

```
>>> del a[1] ← Key가 1인 Key: Value 쌍 삭제  
>>> a  
{2: 'b', 'name': 'pey', 3: [1, 2, 3]}
```

■ 딕셔너리를 사용하는 방법

■ 딕셔너리에서 Key를 사용해 Value 얻기

```
>>> grade = {'pey': 10, 'julliet': 99}
>>> grade['pey']      ← Key가 'pey'인 딕셔너리의 Value를 반환
10
>>> grade['julliet'] ← Key가 'julliet'인 딕셔너리의 Value를 반환
99
```

```
>>> a = {1:'a', 2:'b'}
>>> a[1] ← Key가 1인 요소의 Value를 반환
'a'
>>> a[2] ← Key가 2인 요소의 Value를 반환
'b'
```

- 리스트나 튜플, 문자열은 요솟값 접근 시 인덱싱이나 슬라이싱 기법 중 하나를 사용
- 딕셔너리는 Key를 사용해 Value 접근

■ 딕셔너리를 사용하는 방법

■ 딕셔너리 만들 때 주의할 사항

- 딕셔너리에는 동일한 Key가 중복으로 존재할 수 없음

```
>>> a = {1:'a', 1:'b'} ← 1이라는 Key 값이 중복으로 사용
>>> a
{1: 'b'} ← 1: 'a' 쌍이 무시됨.
```

- 리스트는 그 값이 변할 수 있기 때문에 Key로 쓸 수 없으나 Value에는 아무 값이나 가능

```
>>> a = {[1,2] : 'hi'} ← 리스트를 key로 사용
Traceback (most recent call last):
  File "<stdin>", line 1, in <module> ← 오류 발생
TypeError: unhashable type: 'list'
```

■ 딕셔너리 관련 함수

■ Key 리스트 만들기 - **Keys**

- Key만을 모아서 dict_keys 객체 리턴

```
>>> a = {'name': 'pey', 'phone': '010-9999-1234', 'birth': '1118'}
>>> a.keys()
dict_keys(['name', 'phone', 'birth'])
```

- 리스트처럼 사용할 수 있지만, 리스트 관련 함수(append, insert, pop, remove, sort 등)는 사용 불가능

```
>>> for k in a.keys():
...     print(k)
...
name
phone
birth
```

- dict_keys 객체를 리스트로 변환하는 방법

```
>>> list(a.keys())
['name', 'phone', 'birth']
```


■ 딕셔너리 관련 함수

■ Value 리스트 만들기 - **values**

- Value만을 모아 dict_values 객체 리턴

```
>>> a.values()  
dict_values(['pey', '010-9999-1234', '1118'])
```

■ Key, Value 쌍 얻기 – **items**

- Key와 Value의 쌍을 튜플로 묶어 dict_items 객체 리턴

```
>>> a.items()  
dict_items([('name', 'pey'), ('phone', '010-9999-1234'), ('birth', '1118')])
```

■ 딕셔너리 관련 함수

- Key: Value 쌍 모두 지우기 – **clear**
 - 딕셔너리 내의 모든 요소 삭제
 - 빈 딕셔너리는 {}로 표현

```
>>> a.clear()
>>> a
{}

```

- 해당 Key가 딕셔너리 안에 있는지 조사하기 – **in**
 - Key가 딕셔너리 안에 존재하면 True, 존재하지 않으면 False 리턴

```
>>> a = {'name': 'pey', 'phone': '010-9999-1234', 'birth': '1118'}
>>> 'name' in a
True
>>> 'email' in a
False

```

■ 딕셔너리 관련 함수

- Key로 Value 얻기 – **get**
 - Key에 대응되는 Value 리턴
 - 존재하지 않는 키 사용 시 None 리턴
 - 오류를 발생시키는 list와 차이가 있음
- Key 값이 없을 경우 미리 정해 둔 디폴트 값을 대신 가져오도록 지정 가능

```
>>> a = {'name': 'pey', 'phone': '010-9999-1234', 'birth': '1118'}
>>> a.get('name')
'pey'
>>> a.get('phone')
'010-9999-1234'
```

```
>>> a = {'name': 'pey', 'phone': '010-9999-1234', 'birth': '1118'}
>>> print(a.get('nokey'))
None
>>> print(a['nokey'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'nokey'
```

```
>>> a.get('nokey', 'foo')
'foo'
```

■ 집합(set)이란?

- 집합에 관련된 것을 쉽게 처리하기 위해 만든 자료형

- set 키워드를 사용하여 생성

- set()에 리스트를 입력하여 생성

```
>>> s1 = set([1, 2, 3])  
>>> s1  
{1, 2, 3}
```

- set()에 문자열을 입력하여 생성

```
>>> s2 = set("Hello")  
>>> s2  
{ 'e', 'H', 'l', 'o' }
```

■ 집합 자료형의 특징

- 1) 중복을 허용하지 않음
 - 2) 순서가 없음(Unordered)
- 리스트나 튜플은 순서가 있기 때문에 인덱싱을 통해 요솟값을 얻지만 set 자료형은 순서가 없기 때문에 인덱싱 사용 불가 (딕셔너리와 유사)
 - 인덱싱 사용을 원할 경우 리스트나 튜플로 변환 필요
 - list(), tuple() 함수 사용

```
>>> s1 = set([1, 2, 3])
>>> l1 = list(s1) ← 리스트로 변환
>>> l1
[1, 2, 3]
>>> l1[0]
1
>>> t1 = tuple(s1) ← 튜플로 변환
>>> t1
(1, 2, 3)
>>> t1[0]
1
```

■ 교집합, 합집합, 차집합 구하기

- set 자료형을 유용하게 사용 가능

```
>>> s1 = set([1, 2, 3, 4, 5, 6])  
>>> s2 = set([4, 5, 6, 7, 8, 9])
```

- 교집합 구하기

- '&' 기호나 intersection() 함수 사용

```
>>> s1 & s2  
{4, 5, 6}
```

```
>>> s1.intersection(s2)  
{4, 5, 6}
```

- 합집합 구하기

- '|' 기호나 union() 함수 사용

```
>>> s1 | s2  
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
>>> s1.union(s2)  
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

- 차집합 구하기

- '-' 기호나 difference() 함수 사용

```
>>> s1 - s2  
{1, 2, 3}  
>>> s2 - s1  
{8, 9, 7}
```

```
>>> s1.difference(s2)  
{1, 2, 3}  
>>> s2.difference(s1)  
{8, 9, 7}
```

■ 집합 자료형 관련 함수

■ 값 1개 추가하기 - **add**

```
>>> s1 = set([1, 2, 3])
>>> s1.add(4)
>>> s1
{1, 2, 3, 4}
```

■ 값 여러 개 추가하기 - **update**

```
>>> s1 = set([1, 2, 3])
>>> s1.update([4, 5, 6])
>>> s1
{1, 2, 3, 4, 5, 6}
```

■ 특정 값 제거하기 - **remove**

```
>>> s1 = set([1, 2, 3])
>>> s1.remove(2)
>>> s1
{1, 3}
```

■ 불 자료형(bool)이란?

- 참(True)과 거짓(False)을 나타내는 자료형

```
>>> a = True
>>> b = False
```

- type() 함수를 사용하여 자료형 확인

```
>>> type(a)
<class 'bool'>
>>> type(b)
<class 'bool'>
```

- 조건문의 반환 값으로도 사용됨

```
>>> 1 == 1
True
```

```
>>> 2 > 1
True
```

```
>>> 2 < 1
False
```


■ 자료형의 참과 거짓

값	참 또는 거짓
"python"	참
""	거짓
[1, 2, 3]	참
[]	거짓
(1, 2, 3)	참
()	거짓
{'a': 1}	참
{}	거짓
1	참
0	거짓
None	거짓

```
>>> if []:  ← 만약 []가 참이면
...     print("참")  ← '참' 문자열 출력
... else:  ← 만약 []가 거짓이면
...     print("거짓") ← '거짓' 문자열 출력
...
거짓
```

■ 불 연산

■ bool() 함수

- 자료형의 참/거짓을 식별하는 내장 함수

■ 예제 1: 문자열의 참/거짓

- 'python' 문자열은 빈 문자열이 아니므로 bool 연산의 결과로 True 리턴

```
>>> bool('python')
True
```

- '' 문자열은 빈 문자열이므로 bool 연산의 결과로 False 반환

```
>>> bool('')
False
```

■ 예제 2: 리스트, 숫자의 참/거짓

```
>>> bool([1, 2, 3])
True
>>> bool([])
False
>>> bool(0)
False
>>> bool(3)
True
```

- 변수는 어떻게 만들까?

- 다음 예에서 a, b, c가 변수

```
>>> a = 1
>>> b = "python"
>>> c = [1, 2, 3]
```

- 변수 선언 방법
 - =(assignment) 기호를 사용

```
변수_이름 = 변수에_저장할_값
```

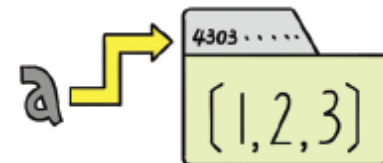
■ 변수란?

- 파이썬에서 사용하는 변수는 객체를 가리키는 것이라고 할 수 있음
 - 객체 = 자료형과 같은 것을 의미하는 말

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

- [1, 2, 3] 값을 가지는 리스트 데이터(객체)가 자동으로 메모리에 생성됨
- 변수 a는 [1, 2, 3] 리스트가 저장된 메모리의 주소를 가리킴
 - id() 함수를 사용하여 메모리 주소 확인

```
>>> a = [1, 2, 3]
>>> id(a)
4303029896
```



- 리스트를 복사하고자 할 때

- b 변수에 a 변수를 대입하면 완전히 동일

```
>>> a = [1, 2, 3]
>>> b = a
```

- id() 함수로 확인

```
>>> id(a)
4303029896
>>> id(b)
4303029896
```

- 파이썬 명령어로 확인

```
>>> a is b
True
```

- 다른 예제로 확인

```
>>> a[1] = 4
>>> a
[1, 4, 3]
>>> b
[1, 4, 3]
```

■ 리스트를 복사하고자 할 때

1. [:] 이용하기

```
>>> a = [1, 2, 3]
>>> b = a[:] ← 리스트 a의 처음 요소부터 끝 요소까지 슬라이싱
>>> a[1] = 4
>>> a
[1, 4, 3]
>>> b
[1, 2, 3]
```

2. copy 모듈 이용하기

- `b = copy(a)`는 `b = a[:]`과 동일

```
>>> from copy import copy ← copy 모듈에 있는 copy 함수 import
>>> a = [1, 2, 3]
>>> b = copy(a) ← copy 함수 사용
```

- 서로 다른 객체를 가리키고 있는지 확인

```
>>> b is a
False
```

■ 변수를 만드는 여러 가지 방법

■ 튜플

```
>>> a, b = ('python', 'life')
```

■ 괄호 생략 가능

```
>>> (a, b) = 'python', 'life'
```

■ 리스트

```
>>> [a, b] = ['python', 'life']
```

■ 여러 개 변수에 같은 값 대입하기

```
>>> a = b = 'python'
```

■ 파이썬에서 두 변수 값 바꾸기

```
>>> a = 3
>>> b = 5
>>> a, b = b, a ← a와 b의 값을 바꿈.
>>> a
5
>>> b
3
```



*"Life is too short,
You need Python!"*

인생은 너무 짧으니,
파이썬이 필요해!

감사합니다.