UNIT 25

딕셔너리 응용하기

25.1 딕셔너리 조작하기

딕셔너리에 키-값 쌍 추가하기

- 다음과 같이 딕셔너리에 키-값 쌍을 추가하는 메서드는 두 가지가 있음

- `setdefault`: 키-값 쌍 추가
- `update`: 키의 값 수정, 키가 없으면 키-값 쌍 추가

25.1 딕셔너리 조작하기

딕셔너리에 키와 기본값 저장하기

- setdefault(키)는 딕셔너리에 키-값 쌍을 추가함
- setdefault에 키만 지정하면 값에 None을 저장함
- 다음은 키 'e'를 추가하고 값에 None을 저장함

```
>>> x = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
>>> x.setdefault('e')
>>> x
{'a': 10, 'b': 20, 'c': 30, 'd': 40, 'e': None}
```

- setdefault(키, 기본값)처럼 키와 기본값을 지정하면 값에 기본값을 저장한 뒤 해당 값을 반환함

```
>>> x.setdefault('f', 100)
100
>>> x
{'a': 10, 'b': 20, 'c': 30, 'd': 40, 'e': None, 'f': 100}
```

25.1 딕셔너리 조작하기

딕셔너리에서 키의 값 수정하기

- update(키=값)은 이름 그대로 딕셔너리에서 키의 값을 수정함

```
>>> x = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
>>> x.update(a=90)
>>> x
{'a': 90, 'b': 20, 'c': 30, 'd': 40}
```

- 딕셔너리에 키가 없으면 키-값 쌍을 추가함

```
>>> x.update(e=50)
>>> x
{'a': 90, 'b': 20, 'c': 30, 'd': 40, 'e': 50}
```

25.1 딕셔너리 조작하기

딕셔너리에서 키의 값 수정하기

```
>>> x.update(a=900, f=60)
>>> x
{'a': 900, 'b': 20, 'c': 30, 'd': 40, 'e': 50, 'f': 60}
```

- update(키=값)은 키가 문자열일 때만 사용할 수 있음
- 만약 키가 숫자일 경우에는 update(딕셔너리)처럼 딕셔너리를 넣어서 값을 수정할 수 있음

```
>>> y = {1: 'one', 2: 'two'}
>>> y.update({1: 'ONE', 3: 'THREE'})
>>> y
{1: 'ONE', 2: 'two', 3: 'THREE'}
```

25.1 딕셔너리 조작하기

딕셔너리에서 키의 값 수정하기

- update(리스트), update(튜플)은 리스트와 튜플로 값을 수정함
- 리스트는 [[키1, 값1], [키2, 값2]] 형식으로 키와 값을 리스트로 만들고 이 리스트를 다시 리스트 안에 넣어서 키-값 쌍을 나열해줌(튜플도 같은 형식)

```
>>> y.update([[2, 'TWO'], [4, 'FOUR']])
>>> y
{1: 'ONE', 2: 'TWO', 3: 'THREE', 4: 'FOUR'}
```

- 특히 update(반복가능한객체)는 키-값 쌍으로 된 반복 가능한 객체로 값을 수정함
- 다음과 같이 키 리스트와 값 리스트를 묶은 zip 객체로 값을 수정할 수 있음

```
>>> y.update(zip([1, 2], ['one', 'two']))
>>> y
{1: 'one', 2: 'two', 3: 'THREE', 4: 'FOUR'}
```

25.1 딕셔너리 조작하기

딕셔너리에서 키-값 쌍 삭제하기

- pop(키)는 딕셔너리에서 특정 키-값 쌍을 삭제한 뒤 삭제한 값을 반환함
- 다음은 딕셔너리 x에서 키 'a'를 삭제한 뒤 10을 반환함

```
>>> x = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
>>> x.pop('a')
10
>>> x
{'b': 20, 'c': 30, 'd': 40}
```

- pop(키, 기본값)처럼 기본값을 지정하면 딕셔너리에 키가 있을 때는 해당 키-값 쌍을 삭제한 뒤 삭제한 값을 반환하지만 키가 없을 때는 기본값만 반환함

```
>>> x.pop('z', 0)
0
```

25.1 딕셔너리 조작하기

딕셔너리에서 키-값 쌍 삭제하기

- pop 대신 del로 특정 키-값 쌍을 삭제할 수도 있음
- 이때는 [ ]에 키를 지정하여 del을 사용함

```
>>> x = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
>>> del x['a']
>>> x
{'b': 20, 'c': 30, 'd': 40}
```

25.1 딕셔너리 조작하기

딕셔너리에서 임의의 키-값 쌍 삭제하기

- popitem()은 딕셔너리에서 임의의 키-값 쌍을 삭제한 뒤 삭제한 키-값 쌍을 튜플로 반환함
- 다음은 딕셔너리 x에서 마지막 키-값 쌍인 'd': 40을 삭제함

파이썬 3.6

```
>>> x = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
>>> x.popitem()
('d', 40)
>>> x
{'a': 10, 'b': 20, 'c': 30}
```

- 참고로 파이썬 3.5와 그 이하 버전에서 popitem 메서드를 사용하면 임의의 키-값을 삭제하므로 매번 삭제하는 키-값 쌍이 달라짐

파이썬 3.5

```
>>> x = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
>>> x.popitem()    # 파이썬 3.5 이하에서는 매번 삭제하는 키-값 쌍이 달라짐
('a', 10)
>>> x
{'b': 20, 'c': 30, 'd': 40}
```

25.1 딕셔너리 조작하기

딕셔너리의 모든 키-값 쌍을 삭제하기

- clear()는 딕셔너리의 모든 키-값 쌍을 삭제함
- 다음은 딕셔너리 x의 모든 키-값 쌍을 삭제하여 빈 딕셔너리 {}가 됨

```
>>> x = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
>>> x.clear()
>>> x
{}
```

25.1 딕셔너리 조작하기

딕셔너리에서 키의 값을 가져오기

- get(키)는 딕셔너리에서 특정 키의 값을 가져옴

```
>>> x = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
>>> x.get('a')
10
```

- get(키, 기본값)처럼 기본값을 지정하면 딕셔너리에 키가 있을 때는 해당 키의 값을 반환하지만 키가 없을 때는 기본값을 반환함

```
>>> x.get('z', 0)
0
```

25.1 딕셔너리 조작하기

딕셔너리에서 키-값 쌍을 모두 가져오기

- 딕셔너리는 키와 값을 가져오는 다양한 메서드를 제공함

- items: 키-값 쌍을 모두 가져옴
- keys: 키를 모두 가져옴
- values: 값을 모두 가져옴

- 다음과 같이 items()는 딕셔너리의 키-값 쌍을 모두 가져옴

```
>>> x = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
>>> x.items()
dict_items([('a', 10), ('b', 20), ('c', 30), ('d', 40)])
```

- keys()는 키를 모두 가져옴

```
>>> x.keys()
dict_keys(['a', 'b', 'c', 'd'])
```

- values()는 값을 모두 가져옴

```
>>> x.values()
dict_values([10, 20, 30, 40])
```

25.1 딕셔너리 조작하기

리스트와 튜플로 딕셔너리 만들기

- dict.fromkeys(키리스트)는 키 리스트로 딕셔너리를 생성하며 값은 모두 None으로 저장함

```
>>> keys = ['a', 'b', 'c', 'd']
>>> x = dict.fromkeys(keys)
>>> x
{'a': None, 'b': None, 'c': None, 'd': None}
```

- dict.fromkeys(키리스트, 값)처럼 키 리스트와 값을 지정하면 해당 값이 키의 값으로 저장됨

```
>>> y = dict.fromkeys(keys, 100)
>>> y
{'a': 100, 'b': 100, 'c': 100, 'd': 100}
```

25.2 반복문으로 딕셔너리의 키-값 쌍을 모두 출력하기

반복문으로 딕셔너리의 키-값 쌍을 모두 출력하기

```
>>> x = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
>>> for i in x:
...     print(i, end=' ')
...
a b c d
```

- for i in x:처럼 for 반복문에 딕셔너리를 지정한 뒤에 print로 변수 i를 출력해보면 값은 출력되지 않고 키만 출력됨

```
for 키, 값 in 딕셔너리.items():
    반복할 코드
```

25.2 반복문으로 딕셔너리의 키-값 쌍을 모두 출력하기

반복문으로 딕셔너리의 키-값 쌍을 모두 출력하기

- 다음은 for로 리스트 a의 모든 키와 값을 출력함

```
>>> x = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
>>> for key, value in x.items():
...     print(key, value)
...
a 10
b 20
c 30
d 40
```

```
for key, value in {'a': 10, 'b': 20, 'c': 30, 'd': 40}.items():
    print(key, value)
```

25.2 반복문으로 딕셔너리의 키-값 쌍을 모두 출력하기

딕셔너리의 키만 출력하기

- 지금까지 items로 키와 값을 함께 가져왔는데, 키만 가져오거나 값만 가져오면서 반복할 수도 있음

```
items: 키-값 쌍을 모두 가져옴
keys: 키를 모두 가져옴
values: 값을 모두 가져옴
```

- 먼저 for 반복문에서 keys로 키를 가져오면서 반복해보겠음

```
>>> x = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
>>> for key in x.keys():
...     print(key, end=' ')
...
a b c d
```

25.2 반복문으로 딕셔너리의 키-값 쌍을 모두 출력하기

딕셔너리의 값만 출력하기

- for 반복문에서 values를 사용하면 값만 가져오면서 반복할 수 있음

```
>>> x = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
>>> for value in x.values():
...     print(value, end=' ')
...
10 20 30 40
```

25.3 딕셔너리 표현식 사용하기

딕셔너리 표현식 사용하기

- 리스트와 마찬가지로 딕셔너리도 for 반복문과 if 조건문을 사용하여 딕셔너리를 생성할 수 있음

- {키: 값 for 키, 값 in 딕셔너리}
- dict({키: 값 for 키, 값 in 딕셔너리})

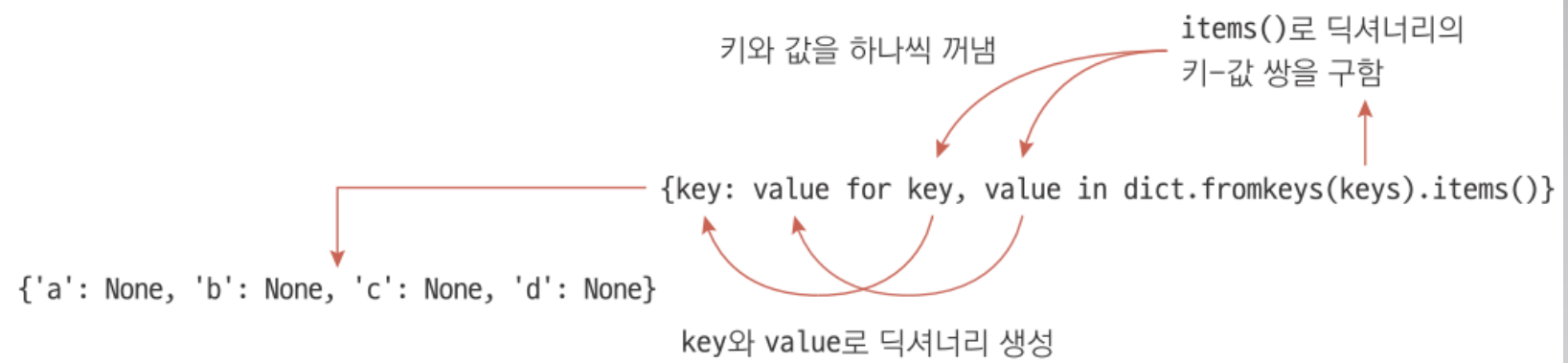
```
>>> keys = ['a', 'b', 'c', 'd']
>>> x = {key: value for key, value in dict.fromkeys(keys).items()}
>>> x
{'a': None, 'b': None, 'c': None, 'd': None}
```

- 딕셔너리 표현식을 사용할 때는 for in 다음에 딕셔너리를 지정하고 items를 사용함
- 키, 값을 가져온 뒤에는 키: 값 형식으로 변수나 값을 배치하여 딕셔너리를 생성됨

```
x = {key: value for key, value in dict.fromkeys(keys).items()}
```

25.3 딕셔너리 표현식 사용하기

▼ 그림 25-1 딕셔너리 표현식의 동작 순서



25.3 딕셔너리 표현식 사용하기

딕셔너리 표현식 사용하기

- 다음과 같이 keys로 키만 가져온 뒤 특정 값을 넣거나, values로 값을 가져온 뒤 값을 키로 사용할 수도 있음

```
>>> {key: 0 for key in dict.fromkeys(['a', 'b', 'c', 'd']).keys()}              # 키만 가져옴
{'a': 0, 'b': 0, 'c': 0, 'd': 0}
>>> {value: 0 for value in {'a': 10, 'b': 20, 'c': 30, 'd': 40}.values()}      # 값을 키로 사용
{10: 0, 20: 0, 30: 0, 40: 0}
```

- 키와 값의 자리를 바꾸는 등 여러 가지로 응용할 수 있음

```
>>> {value: key for key, value in {'a': 10, 'b': 20, 'c': 30, 'd': 40}.items()}  # 키-값 자리를 바꿈
{10: 'a', 20: 'b', 30: 'c', 40: 'd'}
```

25.3 딕셔너리 표현식 사용하기

딕셔너리 표현식에서 if 조건문 사용하기

- 딕셔너리 표현식은 딕셔너리에서 특정 값을 찾아서 삭제할 때 유용함
- 딕셔너리는 특정 키를 삭제하는 pop 메서드만 제공할 뿐 특정 값을 삭제하는 메서드는 제공하지 않음
- 간단하게 for 반복문으로 반복하면서 del로 삭제하는 방식을 떠올릴 수 있음

dict_del_by_value_error.py

```
x = {'a': 10, 'b': 20, 'c': 30, 'd': 40}

for key, value in x.items():
    if value == 20:    # 값이 20이면
        del x[key]     # 키-값 쌍 삭제

print(x)
```

실행 결과

```
Traceback (most recent call last):
  File "C:\project\dict_del_by_value_error.py", line 3, in <module>
    for key, value in x.items():
RuntimeError: dictionary changed size during iteration
```

25.3 딕셔너리 표현식 사용하기

딕셔너리 표현식에서 if 조건문 사용하기

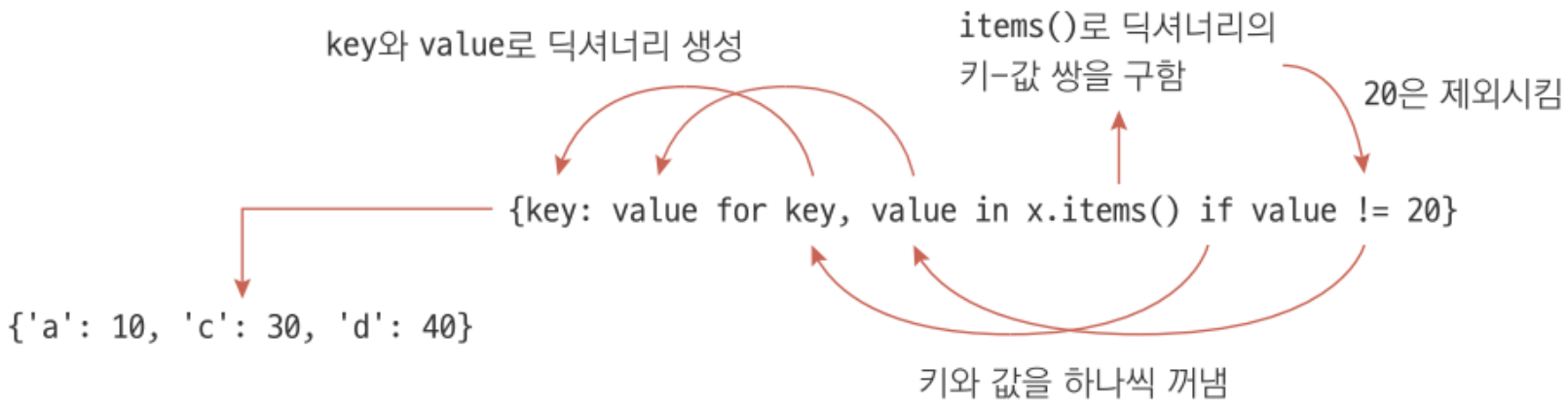
- 별 문제 없이 잘 삭제될 것 같지만 반복 도중에 딕셔너리의 크기가 바뀌었다는 에러가 발생함
- 딕셔너리는 for 반복문으로 반복하면서 키-값 쌍을 삭제하면 안 됨
- 딕셔너리 표현식에서 if 조건문을 사용하여 삭제할 값을 제외하면 됨

- {키: 값 for 키, 값 in 딕셔너리 if 조건식}
- dict({키: 값 for 키, 값 in 딕셔너리 if 조건식})

```
>>> x = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
>>> x = {key: value for key, value in x.items() if value != 20}
>>> x
{'a': 10, 'c': 30, 'd': 40}
```

25.3 딕셔너리 표현식 사용하기

▼ 그림 25-2 딕셔너리 표현식에서 if 조건문 사용하기



25.4 딕셔너리 안에서 딕셔너리 사용하기

딕셔너리 안에서 딕셔너리 사용하기

- 딕셔너리 = {키1: {키A: 값A}, 키2: {키B: 값B}}

- 예를 들어 지구형 행성의 반지름, 질량, 공전주기를 딕셔너리로 표현해보자

dict_dict.py

```
terrestrial_planet = {
    'Mercury': {
        'mean_radius': 2439.7,
        'mass': 3.3022E+23,
        'orbital_period': 87.969
    },
    'Venus': {
        'mean_radius': 6051.8,
        'mass': 4.8676E+24,
        'orbital_period': 224.70069,
    },
    'Earth': {
        'mean_radius': 6371.0,
        'mass': 5.97219E+24,
        'orbital_period': 365.25641,
    },
    'Mars': {
        'mean_radius': 3389.5,
        'mass': 6.4185E+23,
        'orbital_period': 686.9600,
    }
}

print(terrestrial_planet['Venus']['mean_radius'])    # 6051.8
```

실행 결과

```
6051.8
```

25.4 딕셔너리 안에서 딕셔너리 사용하기

딕셔너리 안에서 딕셔너리 사용하기

- 중첩 딕셔너리는 계층형 데이터를 저장할 때 유용함

- 딕셔너리[키][키]
- 딕셔너리[키][키] = 값

- 여기서는 딕셔너리가 두 단계로 구성되어 있으므로 대괄호를 두 번 사용함
- 금성(Venus)의 반지름(mean radius)를 출력하려면 다음과 같이 먼저 'Venus'를 찾아가고 다시 'mean_radius'의 값을 가져오면 됨

```
print(terrestrial_planet['Venus']['mean_radius'])    # 6051.8
```

25.5 딕셔너리의 할당과 복사

딕셔너리의 할당과 복사

- 먼저 딕셔너리를 만든 뒤 다른 변수에 할당함

```
>>> x = {'a': 0, 'b': 0, 'c': 0, 'd': 0}
>>> y = x
```

- 변수 이름만 다를 뿐 딕셔너리 x와 y는 같은 객체임

```
>>> x is y
True
```

- x와 y는 같으므로 y['a'] = 99와 같이 키 'a'의 값을 변경하면 딕셔너리 x와 y에 모두 반영됨

```
>>> y['a'] = 99
>>> x
{'a': 99, 'b': 0, 'c': 0, 'd': 0}
>>> y
{'a': 99, 'b': 0, 'c': 0, 'd': 0}
```

25.5 딕셔너리의 할당과 복사

딕셔너리의 할당과 복사

- 딕셔너리 x와 y를 완전히 두 개로 만들려면 copy 메서드로 모든 키-값 쌍을 복사해야 함

```
>>> x = {'a': 0, 'b': 0, 'c': 0, 'd': 0}
>>> y = x.copy()
```

```
>>> x is y
False
>>> x == y
True
```

- 딕셔너리 x와 y는 별개이므로 한쪽의 값을 변경해도 다른 딕셔너리에 영향을 미치지 않음
- 다음과 같이 딕셔너리 y에서 키 'a'의 값을 변경하면 딕셔너리 x는 그대로이고 딕셔너리 y만 바뀜

```
>>> y['a'] = 99
>>> x
{'a': 0, 'b': 0, 'c': 0, 'd': 0}
>>> y
{'a': 99, 'b': 0, 'c': 0, 'd': 0}
```

25.5 딕셔너리의 할당과 복사

중첩 딕셔너리의 할당과 복사 알아보기

- 다음과 같이 중첩 딕셔너리를 만든 뒤 copy 메서드로 복사함

```
>>> x = {'a': {'python': '2.7'}, 'b': {'python': '3.6'}}
>>> y = x.copy()
```

- 이제 y['a']['python'] = '2.7.15'와 같이 y의 값을 변경해보면 x와 y에 모두 반영됨

```
>>> y['a']['python'] = '2.7.15'
>>> x
{'a': {'python': '2.7.15'}, 'b': {'python': '3.6'}}
>>> y
{'a': {'python': '2.7.15'}, 'b': {'python': '3.6'}}
```

25.5 딕셔너리의 할당과 복사

중첩 딕셔너리의 할당과 복사 알아보기

- 중첩 딕셔너리를 완전히 복사하려면 copy 메서드 대신 copy 모듈의 deepcopy 함수를 사용해야 함

```
>>> x = {'a': {'python': '2.7'}, 'b': {'python': '3.6'}}
>>> import copy              # copy 모듈을 가져옴
>>> y = copy.deepcopy(x)     # copy.deepcopy 함수를 사용하여 깊은 복사
>>> y['a']['python'] = '2.7.15'
>>> x
{'a': {'python': '2.7'}, 'b': {'python': '3.6'}}
>>> y
{'a': {'python': '2.7.15'}, 'b': {'python': '3.6'}}
```

- 딕셔너리 y의 값을 변경해도 딕셔너리 x에는 영향을 미치지 않음
- copy.deepcopy 함수는 중첩된 딕셔너리에 들어있는 모든 딕셔너리를 복사하는 깊은 복사(deep copy)를 해줌