

# 응용프로그래밍

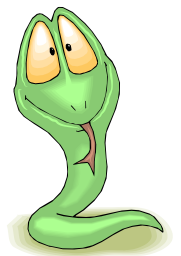
python

2021-2

- 복합 자료형
  - ◆ Dictionary(딕셔너리)
  - ◆ Tuple(튜플)

python

Dictionary(딕셔너리)



# Dictionary 명령어

- `get( )`
- `update( )`
- `popitem( )`
- `del`
- `clear( )`
- membership test (in 키워드)
- `len( )`
- `keys( )`
- `values( )`
- `items( )`
- `dict( )`
- `zip( )`

# Dictionary (딕셔너리)

## ▪ Dictionary는 정렬되지 않은 **key: value** 쌍의 집합임

↳ **value**는 **key**를 통해 식별됨

- 일종의 '문자열로 색인화된 배열'임

↳ **value**는 모든 데이터 유형을, **key**는 거의 모든 데이터 유형을 가질 수 있음

- **Key**값으로 '리스트'와 '딕셔너리'를 사용할 수 없음

↳ '**값이 중복**'되면, '가장 뒤에 있는 값'만 사용함

↳ **중괄호 { }**를 이용해서 정의됨

- cf. **대괄호 [ ]**: 리스트, **소괄호 ( )**: 튜플

- '**숫자(정수, 실수, 복소수)**' 가능  
- '**문자열(strings)**' 가능  
- '**튜플(tuple)**' 가능

```
mydict = { }
```

```
mydict = {name: 'Kim', year: 1999}
```

key

콜론

value

# value 가져오기

- **딕셔너리의 이름**을 쓰고 **keyname**을 지정하면, 지정된 키에 해당하는 **value**를 반환함

**dictionary** [keyname]

```
>>> mydict = { 'name': 'Kim', 'year': 1999 }
```

```
>>> mydict ['name']
```

'Kim'

```
>>> mydict ['year']
```

1999

# value 가져오기 : **get( )**

- **get( )**은 지정된 keyname에 해당하는 value를 반환하는 또 다른 하나의 기능임.

**dictionary.get(keyname, value)**

♦ **keyname** : 필수사항

↳ 일치하는 keyname이 없으면, "지정된 값"을 반환함

♦ **value** : 선택사항

```
>>> mydict = {'name': 'Kim', 'year': 1999}
>>> mydict.get('name')
'Kim'      # mydict['name'] 과 동일한 결과
>>> mydict.get('namw')
아무 것도 출력되지 않음
>>> mydict.get('namw', 'The name is not in the dictionary.')
'The name is not in the dictionary.'
```

# update( ) (1/2)

(1) 딕셔너리에 'key:value' 쌍이 없는 경우

↳ 딕셔너리의 마지막 위치에 'key:value' 쌍을 추가함

**dictionary.update({key:value})**

'key:value' 쌍  
It's iterable

```
>>> mydict = {'name': 'Kim', 'year': 1999}
>>> mydict.update({'hobby': 'collection'})
#{'name': 'Kim', 'year': 1999, 'hobby': 'collection'}
>>> mydict.update({'score': 98})
#{'name': 'Kim', 'year': 1999, 'hobby': 'collection', 'score': 98}
>>> mydict['color'] = 'red'
#{'name': 'Kim', 'year': 1999, 'hobby': 'collection', 'score': 98, 'color': 'red'}
```



# update( ) (2/2)

(2) 딕셔너리에 'key:value' 쌍이 있는 경우

↳ 딕셔너리에 동일한 key를 찾아 해당 value를 변경함

```
#{'name': 'Kim', 'year': 1999, 'hobby': 'collection', 'score': 98, 'color': 'red'}
```

```
>>> mydict['name'] = 'John'
```

```
#{'name': 'John', 'year': 1999 ....}
```

```
>>> mydict['year'] = 2000
```

```
#{'name': 'John', 'year': 2000 ....}
```

```
>>> mydict.update({'color': 'blue'})
```

```
#{ ..., 'score': 98, 'color': 'blue'}
```

# popitem( ), del, clear( )

- 디렉터리에서 "항목을 제거하는 몇가지 명령어" 사용하기
  - ◆ popitem( ), del, clear( )

```
>>> mydict = {'name': 'John', 'year': 2000, 'hobby': 'collection', 'score': 98, 'color': 'blue'}
```

```
>>> mydict.popitem( )
```

맨 뒤의 항목을 제거  
(**'color', 'blue'**)

```
# {'name': 'John', 'year': 2000, 'hobby': 'collection', 'score': 98}
```

```
>>> del mydict['year']
```

year 항목을 제거

```
# {'name': 'John', 'hobby': 'collection', 'score': 98}
```

```
>>> mydict.clear( )
```

전체 항목을 모두 제거

```
# { }
```

```
>>> del mydict
```

# no longer exists

# membership test, len( )

- **in 키워드**는 '지정된 key가 사전에 있는지 여부'를 테스트함(membership test)
- **len( )** 은 '딕셔너리의 길이(length)'를 반환함

## 'key' in dictionary

```
>>> dictionary = { '한국': '서울', '베트남': '하노이', '일본': '도쿄' }
>>> '한국' in dictionary      # membership test
True
>>> '중국' in dictionary
False
>>> '서울' in dictionary • • •
False
>>> len(dictionary)          # length
3
```

### ▶ 주의 사항

"key" 항목이 아닌 "value" 항목을 쓰면  
↳ 존재해도 **False**가 리턴됨!

# Example & Solution

- 'level' dictionary를 작성하고 다음 질문에 답하시오(4문제).

```
>>> level = {'low':1, 'medium':5}
```

(1) 'level' dictionary에서 'medium' key의 value를 반환하시오.

```
>>> level['medium'] or level.get('medium') # 5
```

(2) 'level' dictionary가 'low' key를 가지고 있는지 확인하시오.

```
>>> 'low' in level # True
```

(3) 'level' dictionary에 {'high': 10} 쌍을 추가하시오.

```
>>> level['high'] = 10 or level.update({'high':10})  
# {'low': 1, 'medium': 5, 'high': 10}
```

(4) 'level' dictionary 요소 중 'low' key를 삭제하시오.

```
>>> del level['low'] or level.pop('low') ▶ pop('low') : low 항목이 제거됨  
# {'medium': 5, 'high': 10}
```

# Exercise1 & Solution

- 'x' dictionary를 작성하고, key인 10에 해당하는 value를 출력하는 코드를 쓰시오.

```
>>> x = {10:'Hello', 'world':30}
```

```
>>> print(x[10])
```

```
Hello
```

- 다음 y 값을 출력하는 코드를 쓰시오.

```
>>> y = len({10:0, 20:1, 30:2, 40:3, 50:4, 60:7})
```

```
>>> print(y)
```

```
6
```

# keys( ), values( ), items( )

- **keys( )** : 딕셔너리에 있는 **모든 키(keys)**를 '**리스트**' 형태로 반환함
- **values( )** : 딕셔너리에 있는 **모든 값(values)**을 '**리스트**' 형태로 반환함
- **items( )** : 딕셔너리에 있는 **모든 키-값(keys-values)** 쌍을 '**튜플**'로 묶어 '**리스트**' 형태로 반환함

```
>>> capitaldict = {'한국': '서울', '베트남': '하노이', '일본': '도쿄'}
```

```
>>> capitaldict.keys( )
```

```
# dict_keys( ['한국', '베트남', '일본'] )
```

```
>>> capitaldict.values( )
```

```
# dict_values( ['서울', '하노이', '도쿄'] )
```

```
>>> capitaldict.items( )
```

```
dict_items( [('한국', '서울'), ('베트남', '하노이'), ('일본', '도쿄')] )
```

# 딕셔너리에서 문장 사용하기(1/3)

## ▪ `keys( )`를 사용하여 딕셔너리 반복하기

```
>>> capitaldict = {'한국': '서울', '베트남': '하노이', '일본': '도쿄'}
```

```
>>> for x in capitaldict.keys():  
    print(x)
```

"keys( )"는 생략 가능

**Result>**

한국  
베트남  
일본

```
>>> for x in capitaldict:  
    print(x)
```

# 딕셔너리에서 문장 사용하기(2/3)

## ▪ values( )를 사용하여 딕셔너리 반복하기

```
>>> capitaldict = {'한국': '서울', '베트남': '하노이', '일본': '도쿄'}
```

```
>>> for x in capitaldict.values():
```

```
    print(x, end='')
```

▶ 출력의 끝을 지정

↳ 끝을 공백(' ')으로 지정

**Result>**

서울 < 하노이 < 도쿄



# 딕셔너리에서 문장 사용하기(3/3)

- **items( )**을 사용하여 딕셔너리 반복하기

```
>>> capitaldict = {'한국': '서울', '베트남': '하노이', '일본': '도쿄'}  
  
>>> for x, y in capitaldict.items():  
    print(x, y)
```

**Result>**

한국 서울  
베트남 하노이  
일본 도쿄

# Example & Solution

- 딕셔너리의 key와 value를 사용하여 결과와 같이 출력하는 코드를 작성하시오.  
(반복문 **for**)

```
>>> capitaldict = {'한국': '서울', '베트남': '하노이', '일본': '도쿄'}  
  
>>> for x, y in capitaldict.items():  
    print('{0}의 수도는 {1}입니다.'.format(x, y))
```

## Result>

```
한국의 수도는 서울입니다.  
베트남의 수도는 하노이입니다.  
일본의 수도는 도쿄입니다.
```

- ▶ 문자열 포매팅(string formatting)
  - ↳ 문자열 포매팅은 중괄호({ })안에 포매팅을 지정하고 .format( )에 있는 값을 넣는다.  
(파이썬 변수와 기본 자료형 - print( )함수)

- Example: 'model'이 딕셔너리에 있으면 '해당 값(value)'을 업데이트 하시오.

```
>>> cardict = {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
>>> if 'model' in cardict:
    print("Yes, 'model' is in the car dictionary")
    cardict['model'] = 'BMW'
    print(cardict)
```

## Result>

```
Yes, 'model' is in the car dictionary
{'brand': 'Ford', 'model': 'BMW', 'year': 1964}
```

# Note: 반복문 안에서 'key-value 쌍'은 삭제 불가

- if "value == 20", 딕셔너리에서 key-value 쌍을 삭제하시오.

```
>>> x = {'a': 10, 'b': 20, 'c': 30, 'd': 40, 'e': 50}
>>> for key, value in x.items():
    if value == 20:
        del x[key]
    print(x)
```

▶ "delete" 명령을 사용하여  
딕셔너리의 "{key:value}" 쌍을  
반복문 안에서 삭제할 수 없다.  
→ '다음 페이지'의 "표현식" 사용

Traceback (most recent call last):

File "<pyshell#39>", line 1, in <module>

for key, value in x.items():

**RuntimeError:** dictionary changed size during iteration

- *for* 문과 *if* 문을 사용하여 **표현식** 만들기

③ key, value 쌍으로  
딕셔너리 만들기

① x 딕셔너리로부터  
key, value 전달하기

② 20이 아닌 값

④ **x** ← {key:value for key,value in x.items() if value != 20}

```
>>> x = {'a': 10, 'b': 20, 'c': 30, 'd': 40, 'e': 50}
>>> x = {key: value for key, value in x.items() if value != 20}
>>> x
{'a': 10, 'b': 20, 'c': 30, 'd': 40, 'e': 50}
```

# dict( )

▪ **dict( )**는 딕셔너리를 생성함.

♦ '리스트'나 '튜플'을 "딕셔너리"로 변환할 수 있음.

```
>>> li = [ ('name', 'Alice'), ('year', 1999) ]
```

```
>>> info = dict(li)
```

```
#{'name': 'Alice', 'year': 1999}
```

```
>>> li = ( ('name', 'Alice'), ('year', 1999) )
```

```
>>> info = dict(li)
```

```
#{'name': 'Alice', 'year': 1999}
```

# zip( )

- **zip( )**은 '두 리스트의 요소'를 '쌍(pair)으로 묶는 딕셔너리'를 생성함.

```
>>> title = ['name', 'age', 'birthday']  
>>> value = ['Johnson', 20, '1998-04-15']
```

```
>>> info = dict(zip(title, value))  
# { 'name': 'Johnson', 'age': 20, 'birthday': '1998-04-15' }
```

## [참고사항]

▶ **list( zip(title, value) )**

→ # [ 'name': 'Johnson', 'age': 20, 'birthday': '1998-04-15' ]

▶ **tuple( zip(title, value) )**

→ # ( 'name': 'Johnson', 'age': 20, 'birthday': '1998-04-15' )

# Homework 1

❖ '빈 리스트(list) 두 개'를 만들고, '사용자로부터 요소를 입력'받아 '딕셔너리로 변환'하는 코드를 작성하시오.

◆ 빈 리스트 `name`과 `score`를 만드시오.

```
name = [ ]  
score = [ ]
```

◆ 사용자의 이름과 점수를 계속해서 입력 받으시오(`while True`: ).

◆ "Enter"를 치면, 입력을 종료합니다.

```
if n == "":  
    break
```

◆ 입력된 이름과 점수를 모두 출력하시오.

Result>

이름을 입력하시오 : 홍길동

점수를 입력하시오 : 80

이름을 입력하시오 : 이기자

점수를 입력하시오 : 90

이름을 입력하시오 : 박상진

점수를 입력하시오 : 100

이름을 입력하시오 :

점수를 입력하시오 :

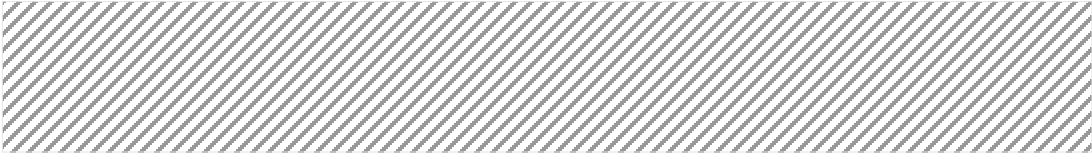
{'홍길동': '80', '이기자': '90', '박상진': '100'}

"Enter"



# Homework 1 : Solution

```
name = []  
score = []  
while True:
```

```
      
    if n == "":  
        break
```

```
print(result)
```

Result>

이름을 입력하시오 : 홍길동

점수를 입력하시오 : 80

이름을 입력하시오 : 이기자

점수를 입력하시오 : 90

이름을 입력하시오 : 박상진

점수를 입력하시오 : 100

이름을 입력하시오 :

점수를 입력하시오 :

{'홍길동': '80', '이기자': '90', '박상진': '100'}

python

Tuple(튜플)



# Tuple 명령어

- index
- slice
- concatenation
- repetition
- len( )
- membership test (in 키워드)
- zip( )

# Tuple(튜플)

- 튜플은 한번 생성되면 그 안에 있는 내용을 수정할 수 없는 정렬된 집합(collection)

↳ 다른 말로, 읽기 전용 배열(read-only array)이라고도 함

↳ cf. "리스트(list)"는 수정 가능

실수로 변경되는 상황을  
방지할 수 있는 "자료구조"

- 튜플은 '동일한 유형의 데이터'나 '혼합 유형의 데이터'를 가질 수 있음

↳ 숫자(정수, 실수, 복소수), 문자(string), 리스트(list) 모두 가능

↳ 튜플 안의 튜플(중첩)도 가능

- 튜플은 소괄호( )를 이용해서 정의함

↳ 소괄호( )는 생략 가능

↳ cf. '대괄호 [ ]'는 "리스트(list)"를 정의하는데 사용

# Tuple: 생성 예 (1/2)

```
>>> # 빈 튜플(empty tuple)
```

```
>>> my_data = ( )
```

```
>>> print(my_data)
```

```
# ( )
```

```
>>> # 여러 개의 문자열을 가지는 튜플(tuple of strings)
```

```
>>> my_data = ("hi", "hello", "bye")
```

```
>>> print(my_data)
```

```
# ('hi', 'hello', 'bye')
```

```
>>> # 정수, 실수, 문자열을 가지는 튜플(tuple of int, float, string)
```

```
>>> my_data2 = (1, 2.8, "Hello World")
```

```
>>> print(my_data2)
```

```
# (1, 2.8, 'Hello World')
```

# Tuple: 생성 예 (2/2)

```
>>> # 문자열과 리스트를 가지는 튜플(tuple of string and list)
>>> my_data3 = ("Book", [1, 2, 3])
>>> print(my_data3)
```

```
# ('Book', [1, 2, 3])
```

```
>>> # "다른 튜플"을 가지는 튜플(tuples inside another tuple)
>>> # └─ 중첩된 튜플(nested tuple)
>>> my_data4 = ((2, 3, 4), (1, 2, "hi"))
>>> print(my_data4)
```

```
# ((2, 3, 4), (1, 2, 'hi'))
```

```
>>> # '단 하나의 항목'만을 가지는 튜플(tuple with only a single item)
>>> my_data = (99, )
```

```
# 만약 99 뒤에 콤마(,)가 없으면
```

```
# my_data는 '정수형 변수'로 취급됨에 주의
```

- "튜플"은 값을 변경할 수 없기 때문에 '**append**'나 '**remove**' 명령어 사용 불가

```
>>> tu = (12, 54, 37, ' bar ')
```

```
>>> tu.append(50)
```

Traceback (most recent call last):

File "<pyshell#42>", line 1, in <module>

tu.append(50)

**AttributeError: 'tuple' object has no attribute 'append'**

```
>>> tu.remove(12)
```

Traceback (most recent call last):

File "<pyshell#48>", line 1, in <module>

tu.remove(12)

**AttributeError: 'tuple' object has no attribute 'remove'**

- "리스트"와 동일하게 'indexing'과 'slicing' 가능

```
>>> tu = (12, 54, 37, ' bar ')  
>>> tu[1]                                # index  
54  
>>> tu[1:3]                              # slice  
(54, 37)  
>>> tu[2:]                               # slice  
(37, 'bar')
```



# concatenation, repetition

```
>>> tu = (12, 54, 37, ' bar ')
```

```
>>> p = (' best ', 70)
```

```
>>> tu+p
```

# concatenation

```
(12, 54, 37, 'bar', 'best', 70)
```

```
>>> tu*3
```

# repetition

```
(12, 54, 37, 'bar', 12, 54, 37, 'bar', 12, 54, 37, 'bar')
```

# 음수 인덱싱(Negative indexing)

- "리스트"와 동일하게 '음수 인덱싱' 가능
  - ↳ 튜플 엘리먼트(element)의 '끝'부터 접근이 가능한 인덱싱
  - ↳ **-1** : '마지막 엘리먼트'에 접근
  - ↳ **-2** : '마지막에서 두번째 엘리먼트'에 접근 등등

```
>>> my_data = (1, 2, "Kevin", 8.9)
```

```
>>> print(my_data[-1])
```

```
# 8.9
```

```
>>> print(my_data[-2])
```

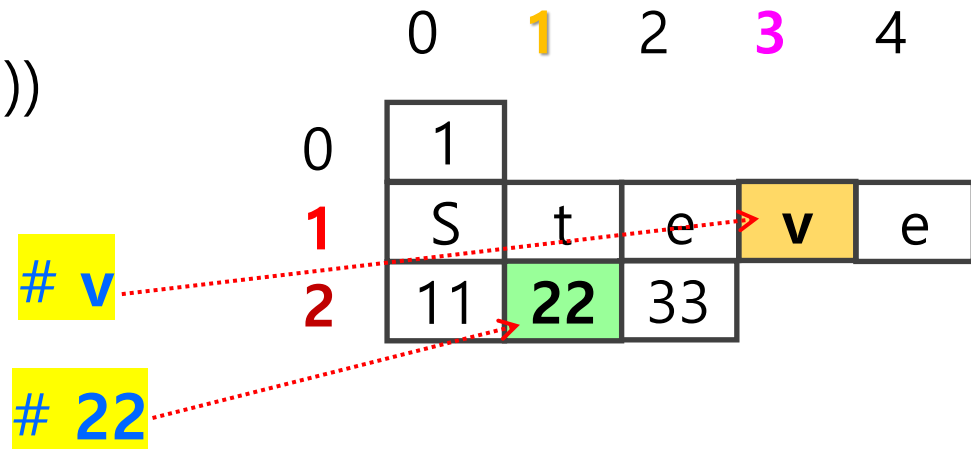
```
# Kevin
```

- '**이중 인덱스**'는 '**중첩 튜플**'의 **엘리먼트의 접근**'에 사용
  - ▶ 첫 번째 인덱스는 "행(외측) 튜플의 엘리먼트"를 나타내고,  
두 번째 인덱스는 "열(내측) 튜플의 엘리먼트"를 나타냄

```
>>> my_data = (1, "Steve", (11, 22, 33))
```

```
>>> print(my_data[1][3])
```

```
>>> print(my_data[2][1])
```



# len( ), membership test

- len( )은 '튜플의 길이(length)'를 반환함
- in 키워드는 '지정된 요소가 튜플에 있는지 여부'를 테스트함(membership test)

```
>>> tu = (12, 54, 37, 'bar')
```

```
>>> len(tu)                                # length
```

```
4
```

```
>>> 'bar' in tu                             # membership test
```

```
True
```

```
>>> 13 in tu                               # membership test
```

```
False
```

# 중첩 튜플의 길이

- 중첩 튜플의 길이는 '모든 개별 튜플의 개수'를 의미

```
>>> # nested tuple(중첩 튜플)
>>> my_data4 = ((2, 3, 4), (1, 2, "hi"))
>>> len(my_data4)
```

# 2

# zip( )

- '두 개의 튜플 엘리먼트'를 '서로 쌍(pair)'으로 묶어, "새로운 튜플"을 생성

```
>>> a = ("John", "Charles", "Mike")
```

```
>>> b = ("Jenny", "Christy", "Monica", "Vicky")
```

```
>>> x = zip(a, b)
```

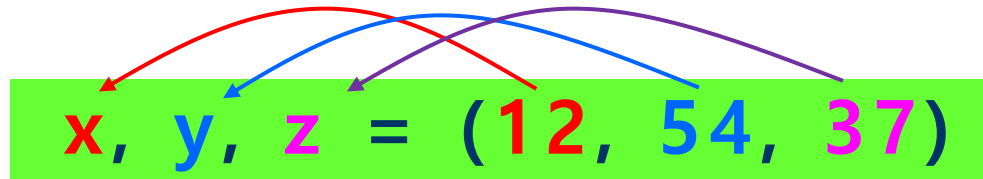
```
>>> print(tuple(x))
```

```
(('John', 'Jenny'), ('Charles', 'Christy'), ('Mike', 'Monica'))
```

"쌍"을 이루지 못하는  
엘리먼트 "Vicky"는 무시됨

```
# [참조] print(x) ► < zip object at 0x04369238 >
```

- ◆ "언패킹"은 '튜플에 있는 엘리먼트들'을 '각각의 변수에 순서대로 대입'하는 연산



```
>>> tu = (12, 54, 37, 'bar')
```

```
>>> x, y, z, s = tu
```

```
>>> x
```

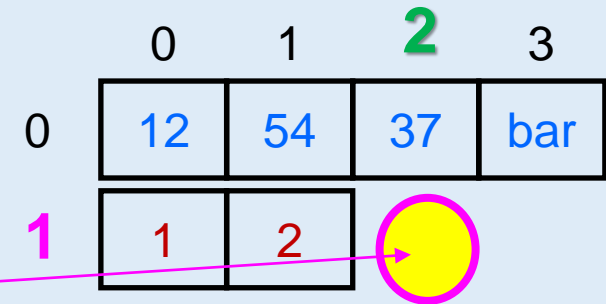
```
12
```

```
>>> s
```

```
'bar'
```

# Example

```
>>> tu = (12, 54, 37, 'bar')
>>> ple = tu, (1,2)
>>> print(ple)           # ((12, 54, 37, 'bar'), (1, 2))
>>> ple[0][0]            # 12
>>> ple[0][3]            # 'bar'
>>> ple[1][0]            # 1
>>> ple[1:]              # ((1, 2), )
>>> ple[1][2]
```



Traceback (most recent call last):

File "<pyshell#20>", line 1, in <module>

ple[1][2]

**IndexError:** tuple index out of range



# 형 변환 (튜플 ↔ 리스트)

```
>>> st = 10, 20, 30
```

```
# (10, 20, 30)
```

```
# 튜플
```

```
>>> li = list(st)
```

```
# [10, 20, 30]
```

```
# 리스트
```

```
>>> tu = (10, 20, 30)
```

```
# (10, 20, 30)
```

```
# 튜플
```

```
>>> li = list(tu)
```

```
# [10, 20, 30]
```

```
# 리스트
```

```
>>> li.append(40)
```

```
# [10, 20, 30, 40]
```

```
# 리스트
```

```
>>> tu = tuple(li)
```

```
# (10, 20, 30, 40)
```

```
# 튜플
```

Q&A