

[Python]



# Python으로 배우는 소프트웨어 원리

## Chapter 08. 튜플과 딕셔너리

# 목차

1. 튜플
2. 딕셔너리

<https://github.com/dndx0r/BCPy>

01

튜플

# 01. 튜플

## [검색의 유용함]

- 튜플(tuple)은 리스트처럼 여러 개의 데이터를 저장할 수 있는 자료형이다.
- 튜플의 항목 값들은 생성 후로는 **변경 삭제가 불가능하다. (읽기 전용)**
- 튜플을 자주 검색하는 항목에 대해 **정렬**이 된 상태로 만들면 검색 속도를 향상시킬 수 있다.
- 튜플은 항목들의 위치에 따른 값의 의미를 일정하게 가져가는 방식으로 사용

#등급표

```
grades = ("A+", [100, 95]), ("A", [94, 90]), ("B+", [89, 85]), ("B", [84, 80]),  
         ("C+", [79, 75]), ("C", [74, 70]), ("D+", [69, 65]), ("D", [64, 60]),  
         ("F", [59, 0])
```

#점수표

```
members = (('choi', 93), ('han', 50), ('jung', 92), ('kang', 68), ('kim', 80),  
           ('lee', 90), ('moon', 65), ('na', 100), ('park', 75), ('song', 75))
```



그림 8-1 일상에서의 검색

# 01. 튜플

## I. 튜플의 개념과 생성

- 튜플은 리스트처럼 여러 개의 데이터를 저장할 수 있는 자료형
- 튜플은 리스트처럼 여러 개의 데이터 항목을 0부터 시작하는 정수형 **인덱스**를 []를 이용해 각 항목을 참조하거나 **슬라이싱**
- **소괄호 ( )**를 사용해서 튜플을 생성
- 인덱싱이나 슬라이싱은 리스트와 동일한 방법을 사용

```
튜플명 = (항목1, 항목2, 항목3, 항목4, 항목5, ... 항목n)
인덱스 → [0]    [1]    [2]    [3]    [4]    ... [n-1]
```

```
>>> a = (3, 1, 5, 9)      # 튜플의 생성
>>> a[2]                  # 인덱싱
5
>>> a[1:3]                # 슬라이싱
(1, 5)
```

- 튜플의 값은 **읽기 전용** 자료형으로 추가하거나 삭제, 변경하는 것이 불가능

# 01. 튜플

## I. 튜플의 개념과 생성

### 실습 8-1

### 튜플 생성하기

- ① 튜플은 소괄호를 이용해서 정의하지만, **소괄호**를 생략하고 값만 입력해도 됨

```
>>> a = (3, 1, 5, 9)
>>> b = 10, 20
>>> type(a); type(b)
<class 'tuple'>
<class 'tuple'>
```

- ② 값이 하나인 튜플 생성 시에는 반드시 값 뒤에逗를 사용해야

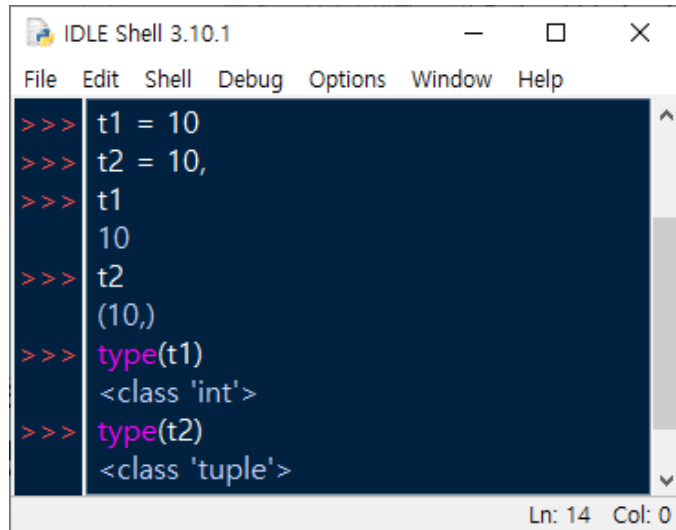
```
>>> x = 10                # 정수형 변수 생성
>>> y = 10,               # 항목이 하나인 튜플의 생성, y = (10)과 동일
>>> type(x); type(y)
<class 'int'>
<class 'tuple'>
```

```
>>> x = (3)               # ()는 묶음의 의미로 사용
>>> a = (3,)              # ()는 묶음의 의미, ,는 튜플 항목의 계속 의미로 해석
>>> b = 10,
>>> type(a); type(b); type(x);
```

# 01. 튜플

## I. 튜플의 개념과 생성

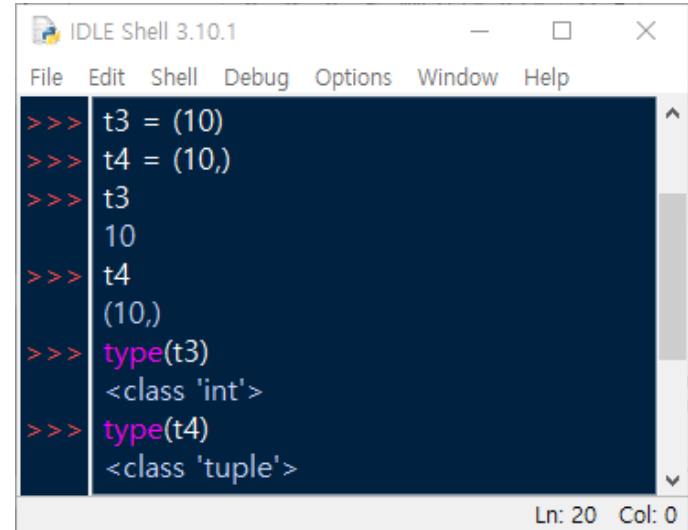
### [실습] 튜플 생성



```

IDLE Shell 3.10.1
File Edit Shell Debug Options Window Help
>>> t1 = 10
>>> t2 = 10,
>>> t1
10
>>> t2
(10,)
>>> type(t1)
<class 'int'>
>>> type(t2)
<class 'tuple'>
Ln: 14 Col: 0

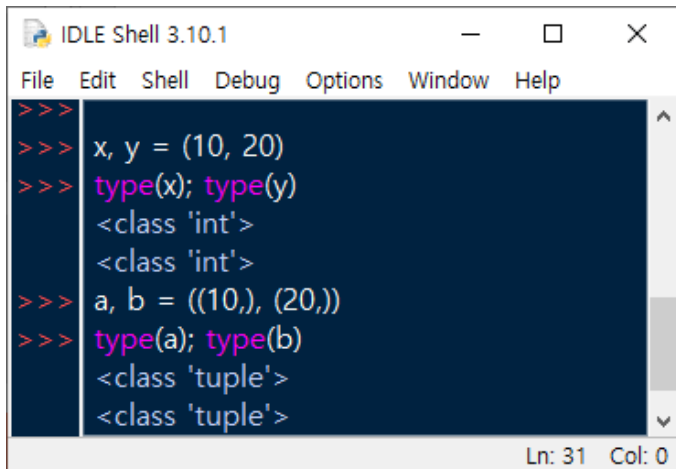
```



```

IDLE Shell 3.10.1
File Edit Shell Debug Options Window Help
>>> t3 = (10)
>>> t4 = (10,)
>>> t3
10
>>> t4
(10,)
>>> type(t3)
<class 'int'>
>>> type(t4)
<class 'tuple'>
Ln: 20 Col: 0

```



```

IDLE Shell 3.10.1
File Edit Shell Debug Options Window Help
>>>
>>> x, y = (10, 20)
>>> type(x); type(y)
<class 'int'>
<class 'int'>
>>> a, b = ((10,), (20,))
>>> type(a); type(b)
<class 'tuple'>
<class 'tuple'>
Ln: 31 Col: 0

```

# 01. 튜플

## I. 튜플의 개념과 생성

실습 8-1

튜플 생성하기

- ③ 튜플 생성 시에 리스트를 그대로 사용 가능, 반대로 튜플을 리스트 생성 시 사용 가능

```
>>> alist = [2, 4, 6, 8, 10]
>>> c = tuple(alist)          # 리스트를 튜플로 변환
>>> c
(2, 4, 6, 8, 10)
>>> blist = list(c)           # 튜플을 리스트로 변환
>>> blist
[2, 4, 6, 8, 10]
```

## 튜플 초기화

t\_tri = () #튜플 구조

```
t_tri = tuple(i for i in range(0, 15, 3))
print(t_tri)
```

- ④ range( ) 함수를 이용해서 규칙적인 간격을 갖는 정수로 구성된 튜플을 생성

```
>>> d = tuple(range(1, 10, 2))
>>> type(d)
<class 'tuple'>
>>> d
(1, 3, 5, 7, 9)
```



# 01. 튜플

## II. 튜플 사용법

### 실습 8-2

### 튜플 다루기

- ① 튜플의 특정 위치나 범위에 있는 항목을 참조하기 위한 인덱싱과 슬라이싱을 실행

```
>>> a = (3, 1, 5, 9)
>>> a[2]
5
>>> a[1:3]
(1, 5)
```



그림 8-2 튜플의 인덱싱과 슬라이싱

- ② '+' 연산자는 튜플 간의 결합, '\*' 연산자는 튜플의 반복을 의미

```
>>> b = 10, 20
>>> a + b
(3, 1, 5, 9, 10, 20)
>>> b * 3
(10, 20, 10, 20, 10, 20)
```



그림 8-3 튜플의 덧셈과 곱셈

# 01. 튜플

## II. 튜플 사용법

### 실습 8-2

### 튜플 다루기

- ③ for 반복문을 사용하면 튜플의 값을 하나씩 출력

```
>>> a = (3, 1, 5, 9)
>>> for x in a :
    print(x)
3
1
5
9
```

- ④ 리스트처럼 항목의 추가나 변경, 삭제는 실행할 수 없음 >> 변경 불가

```
>>> a.append(15)
Traceback (most recent call last):
  File "<pyshell#42>", line 1, in <module>
    a.append(15)
AttributeError: 'tuple' object has no attribute 'append'
```

# 01. 튜플

## II. 튜플 사용법

### 실습 8-2

### 튜플 다루기

- ⑤ 리스트에서와 같이 len()이나 index(), sum(), min()과 max() 함수들을 사용

```
>>> len(a)
4
>>> a.index(9)
3
>>> sum(a)
18
>>> a.sort()
```

Traceback (most recent call last):

```
File "<pyshell#8>", line 1, in <module>
    a.sort()
```

AttributeError: 'tuple' object has no attribute 'sort'

```
t_tri = (0, 3, 6, 9, 12) #튜플 구조
tmax = max(t_tri)
print(tmax)
```

# 01. 튜플

## II. 튜플 사용법

### 실습 8-3

### 튜플의 패킹과 언패킹

- ① **패킹**Packing은 하나의 변수에 여러 개의 데이터를 넣는 것, 즉 여러 개의 값을 튜플이나 리스트로 묶는 것을 의미

```
>>> atuple = 10, 20, 30, 40, 50      # 튜플 패킹  
>>> alist = ['A', 'B', 'C']          # 리스트 패킹
```

- ② **언패킹**Unpacking은 튜플이나 리스트의 각 항목을 여러 변수에 할당하는 것

```
>>> a, b, c, d, e = atuple             # 튜플 언패킹  
>>> print(a, b, c, d, e)  
10 20 30 40 50  
>>> x, y, z = alist                    # 리스트 언패킹  
>>> print(x, y, z)  
A B C
```

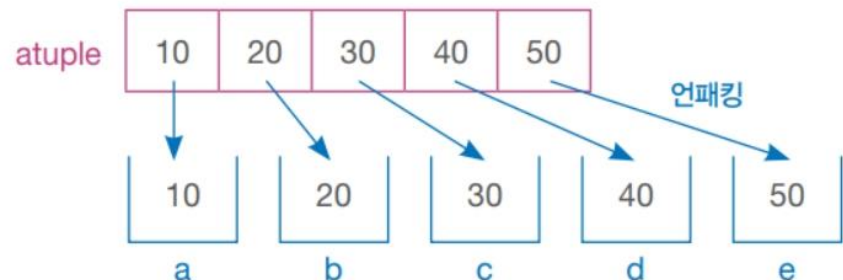


그림 8-4 언패킹의 개념

# 01. 튜플

## II. 튜플 사용법

### 실습 8-3

### 튜플의 패킹과 언패킹

- ③ 변수 이름 앞에 '\*' 기호를 붙이면 여러 개의 값을 갖는 리스트를 언패킹

```
>>> a, b, *c = atuple
>>> print(a, b, c)           # a와 b는 정수형, c는 리스트
10 20 [30, 40, 50]
```

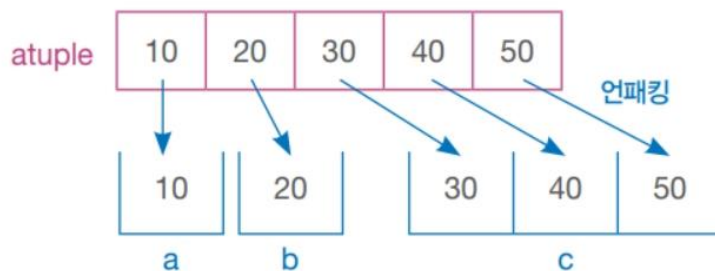


그림 8-5 리스트가 있는 언패킹 과정

# 01. 튜플

## III. 튜플의 활용

- 튜플 members에는 10명의 회원 정보가 있고, 각 항목은 회원 아이디와 점수로 구성된 2차원 튜플

```
# 회원 정보('아이디', 점수)
members = (('choi', 93), ('han', 50), ('jung', 92), ('kang', 68), ('kim', 80),
           ('lee', 90), ('moon', 65), ('na', 100), ('park', 75), ('song', 75))
```

실습 8-4

회원 가입 여부 확인하기

code08-04.py

- ① 튜플에는 여러 개의 데이터가 저장되어 있으므로 회원 아이디를 탐색하는 처리 과정에 'in' 연산이나 반복문이 사용

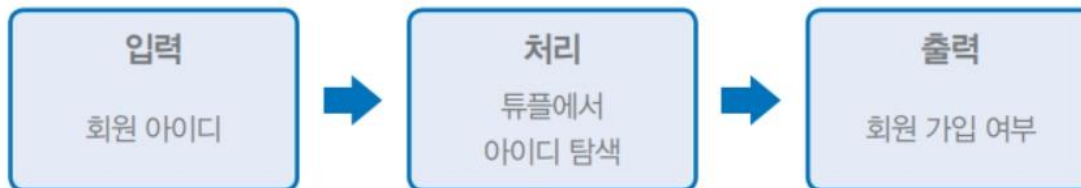


그림 8-6 회원 아이디 검색 후 가입 여부 판단 동작 순서

# 01. 튜플

## III. 튜플의 활용

실습 8-4

회원 가입 여부 확인하기

code08-04.py

② 아이디만 추출해서 리스트에 추가해 두고 'in'으로 찾을

```
01 members = (('choi', 93), ('han', 50), ('jung', 92), ('kang', 68), ('kim', 80),
02             ('lee', 90), ('moon', 65), ('na', 100), ('park', 75), ('song', 75))
03
04 search = input("검색할 아이디 입력 : ")
05
06 idList = []                                # 빈 리스트 정의
07 for x in members :
08     idList.append(x[0])                    # 튜플의 아이디(x[0])만 추출해서 리스트에 추가
09
10 if search in idList :
11     print("가입한 회원입니다.")
12 else :
13     print("회원이 아닙니다.")
```

③

검색할 아이디 입력 : hong  
회원이 아닙니다.

검색할 아이디 입력 : park  
가입한 회원입니다.

# 01. 튜플

## III. 튜플의 활용

### ■ [실습 8-4] "회원 가입 여부 확인"의 다른 해결 방법

```
# 회원 가입 여부 확인
members = (('choi', 93), ('han', 50), ('jung', 92), ('kang', 68), ('kim', 80),
           ('lee', 90), ('moon', 65), ('na', 100), ('park', 75), ('song', 75))

search = input("검색할 아이디 입력 : ")
result = ""
for x, y in members :
    if x == search:
        result = x

if len(result) > 0:
    print("%s는 가입한 회원입니다." %result)
else :
    print("%s는 회원이 아닙니다." %search)
```



# 01. 튜플

## III. 튜플의 활용

실습 8-5

만족도 점수가 가장 높은 회원 찾기

code08-05.py

- ① 앞서 살펴본 알고리즘에 따라 튜플에서 최고 점수를 가진 회원을 검색하는 프로그램을 작성

```
01 members = (('choi', 93), ('han', 50), ('jung', 92), ('kang', 68), ('kim', 80),
02             ('lee', 90), ('moon', 65), ('na', 100), ('park', 75), ('song', 75))
03 memberId = ''
04 num = 0
05
06 for x, y in members :           # x:아이디, y:점수
07     if y > num :                 # 점수 비교
08         memberId = x
09         num = y
10
11 print("만족도 점수가 가장 높은 회원은 %s, 점수는 %d입니다." % (memberId, num))
```

- ② 만족도 점수가 가장 높은 회원은 na, 점수는 100입니다.

# 01. 튜플

## III. 튜플의 활용

실습 8-5

만족도 점수가 가장 높은 회원 찾기

code08-05.py

### ■ 알고리즘 구성:

- 방법: 순서대로 큰 것을 찾으면서 더 큰 것이 있으면 보관하여 유지하는 방식
- 더 큰 점수num와 아이디id를 저장할 변수를 생성하여 초기화
- ✓ num 변수의 초기화 값은 가장 높은 값을 찾아야 하므로 가장 작은 값으로 초기화
- 반복문을 사용하여 현재 항목의 점수가 num보다 크면 id와 num을 현재 항목 값으로 변경

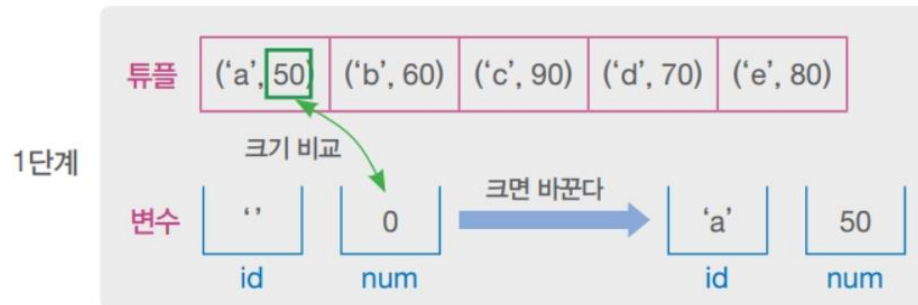


그림 8-7 변수의 초기 값과 첫 번째 항목 비교

# 01. 튜플

## III. 튜플의 활용

실습 8-5

만족도 점수가 가장 높은 회원 찾기

code08-05.py

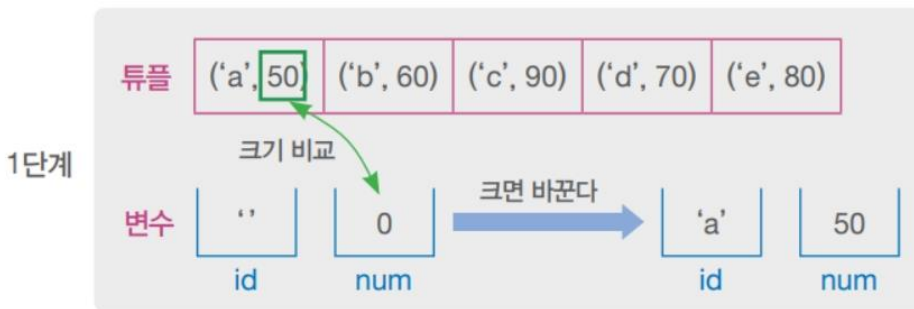


그림 8-7 변수의 초기 값과 첫 번째 항목 비교

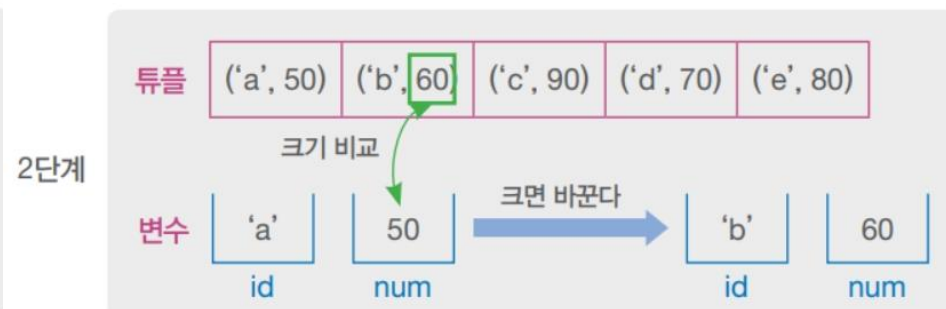


그림 8-8 변수와 두 번째 항목 비교

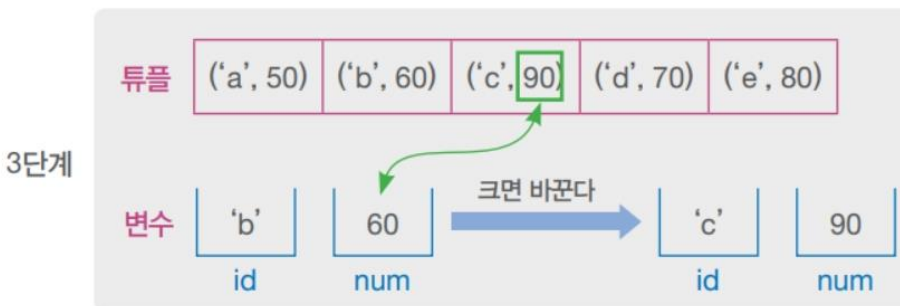


그림 8-9 변수와 세 번째 항목 비교

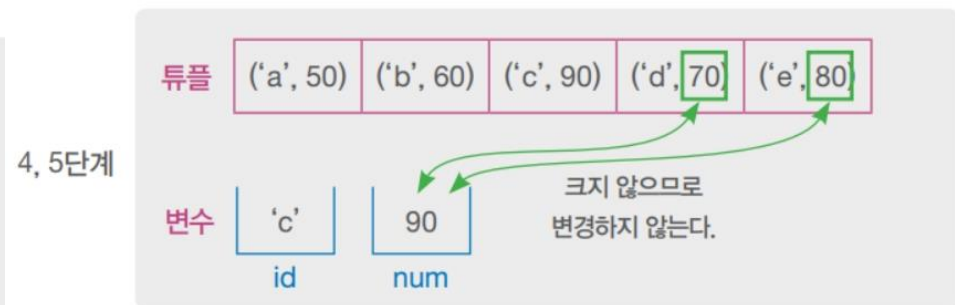


그림 8-10 변수와 나머지 항목 비교

# 01. 튜플

## III. 튜플의 활용

### ■ [실습 8-5] 최고점 회원 출력하기 >> 중복 점수도 처리

*## 튜플 응용 : [점수표] 가장 높은 점수를 취득한 학생의 id와 점수 출력하기(중복 고려)*

```
members = (('choi', 93), ('han', 50), ('jung', 92), ('kang', 68), ('kim', 80),  
           ('lee', 90), ('moon', 65), ('na', 100), ('park', 75), ('song', 75))
```

*#최고점 구하기*

```
smax = #점수만으로 리스트 구성 > 최대 값 찾기  
print(smax)
```

*#최고점 학생 찾기*

```
for id, scr in members:  
    if scr == smax:  
        print(id, scr)
```

# 01. 튜플

## III. 튜플의 활용

실습 8-6

특정 회원의 만족도 검색하기

code08-06.py

①

```
01 members = (('choi', 93), ('han', 50), ('jung', 92), ('kang', 68), ('kim', 80),
02             ('lee', 90), ('moon', 65), ('na', 100), ('park', 75), ('song', 75))
03
04 number = -1                # 만족도 점수의 초기화
05 search = input("아이디 입력 : ")
06 for x, y in members :
07     if search == x :        # 찾는 값이 있으면 점수를 저장하고 반복 종료
08         number = y
09         break
10
11 if number > -1 :
12     print(search, number)
13 else :
14     print("찾는 회원이 없습니다.")
```

# 01. 튜플

## III. 튜플의 활용


실습 8-6

특정 회원의 만족도 검색하기

code08-06.py

- ② **이진 탐색** BinarySearch을 사용하여 회원 아이디를 검색하는 프로그램에서는 탐색의 범위를 계속 줄여나가기 때문에 데이터 양이 증가하더라도 순차 탐색보다 더 효율적

```
01 members = (('choi', 93), ('han', 50), ('jung', 92), ('kang', 68), ('kim', 80),
02           ('lee', 90), ('moon', 65), ('na', 100), ('park', 75), ('song', 75))
03 search = input("아이디 입력 : ")
04 number = -1                                # 만족도 점수의 초기화
05 start = 0                                  # 범위의 시작과 마지막 설정
06 end = len(members)
07 mid = (start + end) // 2                    # 가운데 항목을 첫 번째 검색 위치로 설정
08
09 while start < end :
10     if search == members[mid][0] :          # 찾는 아이디가 있으면 점수를 저장하고 반복 종료
11         number = members[mid][1]
12         break
13     else :
14         if start == (end - 1) :              # 검색 범위를 줄일 수 없음(=찾는 값이 없음)
15             break
```



- ❖ 이진탐색(Binary Search)은 검색 대상이 검색하는 대상으로 정렬이 되어있을 때 적용 가능하다.

# 01. 튜플

## III. 튜플의 활용

실습 8-6

특정 회원의 만족도 검색하기

code08-06.py

- ② 이진 탐색을 사용하여 회원 아이디를 검색하는 프로그램에서는 탐색의 범위를 계속 줄여나가기 때문에 데이터 양이 증가하더라도 순차 탐색보다 더 효율적

```
16         elif search > members[mid][0] :
17             start = mid + 1                # 검색 범위를 뒤쪽 반으로 줄이기
18             mid = (start + end) // 2
19         else :
20             end = mid                      # 검색 범위를 앞쪽 반으로 줄이기
21             mid = (start + end) // 2
22
23     if number > -1 :
24         print(search, number)
25     else :
26         print("찾는 회원이 없습니다.")
```

③

회원 아이디 입력 : song  
song 75

회원 아이디 입력 : kim  
kim 80

회원 아이디 입력 : a  
찾는 회원이 없습니다.

# 01. 튜플

## [실습] 성적의 등급 판단 기준이 되는 **등급표** 튜플 생성

- ◆ 등급 체계는 5점 단위로 구분하여 등급표 튜플을 생성하시오.
- 각 항목의 첫번째는 등급으로 'A+', 'A', 'B+', 'B', 'C+', 'C', 'D+', 'D', 'F'로 구분한다.
- 각 항목의 두번째는 등급의 점수 범위로 두개 항목으로 구성된 리스트로 구성한다.
  - 두개 항목은 해당 등급의 최고점과 최저점으로 구성

```
grades = ("A+", [100, 95]), ("A", [94, 90]), ("B+", [89, 85]), ("B", [84, 80]),  
         ("C+", [79, 75]), ("C", [74, 70]), ("D+", [69, 65]), ("D", [64, 60]),  
         ("F", [59, 0]))
```

```
for grade, zone in grades:  
    print(grade, " > ", zone)
```

```
A+ > [100, 95]  
A > [94, 90]  
B+ > [89, 85]  
B > [84, 80]  
C+ > [79, 75]  
C > [74, 70]  
D+ > [69, 65]  
D > [64, 60]  
F > [59, 0]
```



# 01. 튜플

## [실습] 등급표 튜플 검색 (등급으로 찾기)

Ch08-GradeT00.py

◆ 등급을 입력 받아서 등급과 점수 범위를 출력하시오.

```
grades = (["A+", [100, 95]], ["A", [94, 90]], ["B+", [89, 85]], ["B", [84, 80]],  
          ["C+", [79, 75]], ["C", [74, 70]], ["D+", [69, 65]], ["D", [64, 60]],  
          ["F", [59, 0]])
```

#등급표 검색(등급으로 찾기)

```
ingrade = input("Grade 입력: ")
```

```
ingrade = ingrade.upper()    #대문자로 변환
```

```
for grade, zone in grades:    #zone은 리스트형
```

```
    if grade == ingrade:
```

```
        print("%s : %d~%d" %(grade, zone[1], zone[0]))
```

Grade 입력: **b+**

B+ : 85~89

# 01. 튜플

## [실습] 등급표 튜플 검색 (점수로 찾기)

Ch08-GradeT00.py

◆ 점수를 입력 받아서 점수에 해당되는 등급을 출력하시오.

```
grades = ([ "A+", [100, 95]], [ "A", [94, 90]], [ "B+", [89, 85]], [ "B", [84, 80]],  
          [ "C+", [79, 75]], [ "C", [74, 70]], [ "D+", [69, 65]], [ "D", [64, 60]],  
          [ "F", [59, 0]])
```

#등급표 검색(점수로 찾기)

```
inscore = int(input("Score 입력: "))
```

```
for grade, zone in grades:          #zone은 리스트형
```

```
    if inscore >= zone[1] and inscore <= zone[0]:
```

```
        print("%d : %s" %(inscore, grade))
```

Score 입력: 77

77 : C+

# 01. 튜플

## [과제-1] [실습 8-6] 학생 성적표 출력 (함수 호출로 등급 판단)

- ◆ 학생들의 점수에 따른 등급을 함께 표현하는 성적표를 출력하시오.
  - 학생들의 점수에 대한 등급을 판정해주는 함수를 만들어 해결하시오.

#등급표

```
grades = ("A+", [100, 95]), ("A", [94, 90]), ("B+", [89, 85]), ("B", [84, 80]),  
          ("C+", [79, 75]), ("C", [74, 70]), ("D+", [69, 65]), ("D", [64, 60]),  
          ("F", [59, 0]))
```

#점수표

```
members = (('choi', 93), ('han', 50), ('jung', 92), ('kang', 68), ('kim', 80),  
           ('lee', 90), ('moon', 65), ('na', 100), ('park', 75), ('song', 75))
```

#등급표 검색(점수로 찾기)

```
def find_grade(inscore):  
    :  
    :  
    return outgrade
```

#성적표 만들기

```
for id, score in members:  
    print("%10s : %3d [%2s]" % (id, score, find_grade(score)))
```

```
choi : 93 [ A]  
han : 50 [ F]  
jung : 92 [ A]  
kang : 68 [D+]  
kim : 80 [ B]  
lee : 90 [ A]  
moon : 65 [D+]  
na : 100 [A+]  
park : 75 [C+]  
song : 75 [C+]
```

# 01. 튜플

## III. 튜플의 활용

### ■ 항목을 변경할 수 있는 튜플 생성

- 튜플 항목을 리스트로 구성하면 튜플의 항목도 변경할 수 있다.

```
a = ([1], [2], [3])    #리스트를 항목으로 하는 튜플 생성
a
a[1] = [10]           #리스트 항목 변경
a[2] = 30
a
a.append([4])         #튜플에 리스트 항목 추가 불가
a
```

```
a = ([1], [2], [3])
a
([1], [2], [3])
a[1] = [10]
Traceback (most recent call last):
  File "<pyshell#101>", line 1, in <module>
    a[1] = [10]
TypeError: 'tuple' object does not support item assignment
a[1][0] = [20]
a
([1], [[20]], [3])
a.append([40])
Traceback (most recent call last):
  File "<pyshell#104>", line 1, in <module>
    a.append([40])
AttributeError: 'tuple' object has no attribute 'append'
```

# 01. 튜플

## [실습] 전화 지역번호표 튜플 생성

- ◆ 우리 나라의 전화번호의 지역별 지역번호표를 튜플로 생성하시오.
  - 한 항목에 복수 개의 데이터를 가질 수 있는 항목은 리스트로 구성한다.

#지역번호 만들기

```
zoneno = ([['010'], '휴대전화'],  
           ['02', ['서울특별시', '광명시', '과천시']], ['031', '경기도'], ['032', ['인천광역시', '부천시']], ['033', '강원도'],  
           ['041', '충청남도'], ['042', ['대전광역시', '계룡시']], ['043', '충청북도'], ['044', '세종특별자치시'],  
           ['051', '부산광역시'], ['052', '울산광역시'], ['053', ['대구광역시', '경산시']], ['054', '경상북도'],  
           ['055', '경상남도'], ['061', '전라남도'], ['062', '광주광역시'], ['063', '전라북도'], ['064', '제주특별자치도'])
```

```
zoneno[0][0] = ['010', '011', '016', '017', '018', '019'] #휴대전화 지역번호 항목 추가
```

#지역번호표 출력

```
for no, zone in zoneno:  
    print(no, "> ", zone)
```

```
['010', '011', '016', '017', '018', '019'] > 휴대전화  
02 > ['서울특별시', '광명시', '과천시']  
031 > 경기도  
032 > ['인천광역시', '부천시']  
033 > 강원도  
041 > 충청남도  
042 > ['대전광역시', '계룡시']  
043 > 충청북도  
044 > 세종특별자치시
```

# 01. 튜플

## [실습] 전화 지역번호표 튜플 검색 (지역번호로 검색)

Ch08-Zoneno00.py

◆ 지역번호를 입력 받아서 해당하는 지역을 출력하시오.

#지역번호 만들기

```
zoneno = ([['010'], '휴대전화'],
```

```
          ['02', ['서울특별시', '광명시', '과천시']], ['031', '경기도'], ['032', ['인천광역시', '부천시']], ['033', '강원도'],
```

```
          ['041', '충청남도'], ['042', ['대전광역시', '계룡시']], ['043', '충청북도'], ['044', '세종특별자치시'],
```

```
          ['051', '부산광역시'], ['052', '울산광역시'], ['053', ['대구광역시', '경산시']], ['054', '경상북도'],
```

```
          ['055', '경상남도'], ['061', '전라남도'], ['062', '광주광역시'], ['063', '전라북도'], ['064', '제주특별자치도'])
```

```
zoneno[0][0] = ['010', '011', '016', '017', '018', '019'] #휴대전화 지역번호 항목 추가
```

#지역번호 검색(번호로 찾기)

```
inno = input("전화 지역번호 입력: ")
```

```
for no, zone in zoneno:
```

```
    if type(no) is list: #항목 형식이 리스트인지 확인
```

```
        if inno in no: #리스트 항목에서 찾기
```

```
            print("%s > %s" %(inno, zone))
```

```
    elif no == inno: #일반 항목일 경우 일치 비교
```

```
        print("%s > %s" %(inno, zone))
```

전화 지역번호 입력: 032

032 > ['인천광역시', '부천시']

# 01. 튜플

## [실습] 전화 지역번호표 튜플 검색 (지역명으로 검색)

Ch08-Zoneno00.py

◆ 지역명을 입력 받아서 해당하는 지역번호를 출력하시오.

#지역번호 만들기

```
zoneno = ([['010'], '휴대전화'],  
          ['02', ['서울특별시', '광명시', '과천시']], ['031', '경기도'], ['032', ['인천광역시', '부천시']], ['033', '강원도'],  
          ['041', '충청남도'], ['042', ['대전광역시', '계룡시']], ['043', '충청북도'], ['044', '세종특별자치시'],  
          ['051', '부산광역시'], ['052', '울산광역시'], ['053', ['대구광역시', '경산시']], ['054', '경상북도'],  
          ['055', '경상남도'], ['061', '전라남도'], ['062', '광주광역시'], ['063', '전라북도'], ['064', '제주특별자치도'])  
zoneno[0][0] = ['010', '011', '016', '017', '018', '019'] #휴대전화 지역번호 항목 추가
```

#지역번호 검색(지역명으로 찾기)

```
inzone = input("전화번호 지역 입력: ")  
for no, zone in zoneno:  
    if type(zone) is list: #항목 형식이 리스트인지 확인  
        if inzone in zone : #리스트 항목에서 찾기  
            print("%s > %s" %(inno, zone))  
    elif zone == inzone: #일반 항목일 경우 일치 비교  
        print("%s > %s" %(inno, zone))
```

전화번호 지역 입력: 부천시  
032 > ['인천광역시', '부천시']

# 01. 튜플

## [과제-2] 전화 지역번호표 튜플 검색 (검색 작업 통합, 함수 호출 처리, 메뉴 제공)

- ◆ 지역 번호로 검색과 지역명으로 검색을 통합하여 검색 선택 메뉴를 제공하도록 수정하시오.
  - 선택 메뉴: [0] 종료 [1] 번호로 검색 [2] 지역명으로 검색
    - 메뉴 구성 시 ([메뉴 번호, 메뉴명]) 튜플 사용
  - 함수 호출: 지역 번호로 검색 : find\_no(), 지역명으로 검색 : find\_zone()

```
menuno = ([0, '종료'], [1, '번호로 검색'], [2, '지역명으로 검색']) #메뉴를 위한 튜플 데이터
```

```
while True:
    print("\n>> 번호로 선택: ", end="")
    for no, name in menuno:
        print("[%d] %s " %(no, name), end=' ')
    selnum = input("No?> ")

    #검색 작업
    if selnum == '0':
        break
    elif selnum == '1':
        inno = input("전화 지역번호 입력: ")
        find_no(inno) #지역번호 검색(번호로 찾기)
    elif selnum == '2':
        inzone = input("전화번호 지역명 입력: ")
        find_zone(inzone) #지역번호 검색(지역명으로 찾기)
```

```
>>번호로 선택: [0] 종료      [1] 번호로 검색      [2] 지역명으로 검색
No?> 1
>>전화 지역번호 입력: 033
033 > 강원도

>>번호로 선택: [0] 종료      [1] 번호로 검색      [2] 지역명으로 검색
No?> 2
>>전화번호 지역 입력: 부천시
033 > ['인천광역시', '부천시']

>>번호로 선택: [0] 종료      [1] 번호로 검색      [2] 지역명으로 검색
No?> 0
```



# 01. 튜플

## [과제-2 도전] 전화 지역번호표 튜플 검색 (검색 작업 통합, 함수 호출 처리, 메뉴 제공)

- ◆ 지역 번호로 검색과 지역명으로 검색을 통합하여 검색 선택 메뉴를 제공하도록 수정하시오.
  - 선택 메뉴: [0] 종료 [1] 번호로 검색 [2] 지역명으로 검색
    - 메뉴 구성 시 ([메뉴 번호, 메뉴명]) 튜플 사용
  - 함수 호출: 지역 번호로 검색 : find\_no(), 지역명으로 검색 : find\_zone()
  - **[2] 지역명으로 검색 시 부분 일치 검색이 되도록 Upgrade**
    - '부'로 검색하면 '부천시'와 '부산광역시'가 함께 검색 되도록 함

**menuno** = ([0, '종료'], [1, '번호로 검색'], [2, '지역명으로 검색']) #메뉴를 위한 튜플 데이터

while True:

```
print("\n>> 번호로 선택: ", end='')
for no, name in menuno:
    print("[%d] %s " %(no, name), end=' ')
selnum = input("No?> ")
```

#검색 작업

```
if selnum == '0':
    break
```

```
elif selnum == '1':
    inno = input("전화 지역번호 입력: ")
    find_no(inno) #지역번호 검색(번호로 찾기)
```

```
elif selnum == '2':
    inzone = input("전화번호 지역명 입력: ")
    find_zone(inzone) #지역번호 검색(지역명으로 찾기)
```

```
>>번호로 선택: [0] 종료    [1] 번호로 검색    [2] 지역명으로 검색
No?> 2
>>전화번호 지역명 입력: 부
032 > ['인천광역시', '부천시']
051 > 부산광역시
```

02

딕셔너리

## 02. 딕셔너리

### I. 딕셔너리 개념과 생성

- 딕셔너리 Dictionary는 리스트나 튜플처럼 여러 개의 데이터를 처리
- 리스트처럼 딕셔너리를 생성해놓고 항목을 **추가, 삭제 가능**
- 딕셔너리는 중괄호({ })안에 **키**와 **값**을 묶어서 하나의 항목으로 저장 (키 중복 불가)
- 인덱스 index 대신 **키 Key**를 사용해서 대응되는 값을 사용
- **콜론(:)** 기호로 키와 값을 연결하고, 각 항목 사이에는 리스트나 튜플처럼 쉼표(,)를 사용

딕셔너리명 = {키1:값1, 키2:값2, ... 키n:값n}

- 키에 대응되는 값은 단수 개의 값이나 복수 개를 갖는 리스트나 튜플도 사용 가능

```
>>> adic = {'a':90, 'b':70, 'c':60, 'd':70}
```

adic

'a'	90
'b'	70
'c'	60
'd'	70

키 : 값

```
>>> bdic = {1:'a', 'b':'bin', 3:[10, 20]}
```

bdic

1	'a'
'b'	'bin'
3	[10,20]

키 : 값

그림 8-17 딕셔너리 구조

## 02. 딕셔너리

### I. 딕셔너리 개념과 생성

- 딕셔너리는 리스트나 튜플처럼 순서 번호를 인덱스로 사용하지 않고, 키를 통해 값을 가져옴
- 키를 통한 대응 값 검색 또는 존재 확인 용도로 사용

```
>>> adic['b']  
70
```

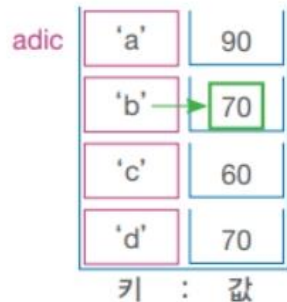
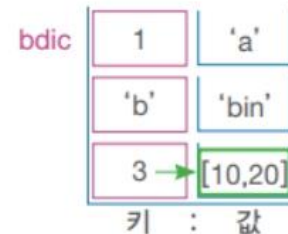


그림 8-18 딕셔너리의 키를 이용한 값 참조

```
>>> bdic[3]  
[10, 20]
```



```
adic = {'a': 90, 'a': 80, 'a': 70} #딕셔너리 구조  
for keys, values in adic.items():  
    print(keys, values)
```

## 02. 딕셔너리

### I. 딕셔너리 개념과 생성

#### 실습 8-7

#### 딕셔너리 생성과 값 참조하기

- ① 딕셔너리를 정의할 때 항목을 저장해도 되고, 빈 딕셔너리를 만들고 나서 필요할 때 항목을 추가

```
>>> adic = {'a':90, 'b':70, 'c':60, 'd':70}
>>> bdic = {}                                # 빈 딕셔너리 생성
>>> type(adic); type(bdic)
<class 'dict'>
<class 'dict'>
```

✓ **TIP** 빈 딕셔너리를 생성할 때 'bdic = **dict()**' 문장 사용 가능

- ② 키를 이용해서 값을 참조, 잘못된 키를 사용하면 오류 메시지가 출력

```
>>> adic['b']
70
>>> adic['f']
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    adic['f']
KeyError: 'f'
```

```
adic = {'a':90, 'b':70, 'c':60, 'd':70}
adic
{'a': 90, 'b': 70, 'c': 60, 'd': 70}
'e' in adic
False
adic.get('b')
70
adic.get('e')
```

## 02. 딕셔너리

### II. 딕셔너리 사용법

#### 실습 8-8

딕셔너리의 값 수정, 항목 추가, 삭제하기

- ① 딕셔너리 항목을 참조할 때처럼 키를 이용해서 값을 수정할 수 있음

```
>>> adic['b'] = 80
>>> adic
{'a': 90, 'b': 80, 'c': 60, 'd': 70}
```

- ② 값을 수정하는 문장에서 만약 기존에 없는 키를 사용하면 새 항목이 추가

```
>>> adic['f'] = 100
>>> adic
{'a': 90, 'b': 80, 'c': 60, 'd': 70, 'f': 100}
```

새 항목 추가

- ③ 항목을 삭제할 때는 리스트나 튜플처럼 del( ) 함수를 사용

```
>>> del adic['f'] # del(adic['f'])와 동일
>>> adic
{'a': 90, 'b': 80, 'c': 60, 'd': 70}
```

✓ **TIP** 모든 항목을 삭제하려면 딕셔너리명.clear()를 사용

## 02. 딕셔너리

### II. 딕셔너리 사용법

- 메소드로 검색

- ① `get()` 메소드로 키를 전달하여 검색, 존재하면 대응 값 반환,  
대응 값이 없으면 에러 없이 반환 값 없음

```
>>> adic = {'a': 90, 'b': 80, 'c': 60, 'd': 70}
>>> 'b' in adic
True
>>> adic.get('b')
80
>>> 'f' in adic
False
>>> adic.get('f')           # 키가 없을 때 오류가 발생하지 않고, 반환값이 없음
```

- ② `keys()`, `values()`, `items()` 메소드를 사용하여 키, 값 또는 항목 검색, **결과는 리스트**

```
>>> adic.keys()             # 키만 가져오기
dict_keys(['a', 'b', 'c', 'd'])
>>> adic.values()          # 값만 가져오기
dict_values([90, 80, 60, 70])
>>> adic.items()           # 키와 값을 묶어서 가져오기
dict_items([('a', 90), ('b', 80), ('c', 60), ('d', 70)])
```

## 02. 딕셔너리

### II. 딕셔너리 사용법

#### 여기서 잠깐 for 반복문으로 딕셔너리 출력하기

- 딕셔너리의 모든 키와 값을 리스트형으로 가져올 수 있으므로, for 반복문을 사용하면 하나씩 출력할 수 있음

```
>>> for x in adic.keys():
```

```
    print(x)
```

a

b

c

d

키 리스트

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

```
    print(x)
```

90

80

60

70

값 리스트

```
>>> for x in adic.items():
```

```
    print(x)
```

'a', 90

'b', 80

'c', 60

'd', 70

항목 리스트



## 02. 딕셔너리

### [실습] 딕셔너리 탐색

Ch08-10Ex.py

#### ◆ 등급(grade) 딕셔너리 초기화

```
grades = {"A+": [100, 95], "A": 90, "B+": 85, "B": 80, "C+": 75,  
          "C": 70, "D+": 65, "D": 60, "F": 0}
```

#### ❖ 키 탐색

```
ingrade = input("등급 입력: ")  
if ingrade in grades.keys():  
    print(grades[ingrade])
```

#### ❖ 값 탐색

```
inscore = int(input("점수 입력: "))  
if inscore in grades.values():  
    print("점수가 존재함.")
```

#### ❖ 항목 탐색

```
inscore = int(input("점수 입력: "))  
for x, y in grades.items():  
    if inscore == y:  
        print(x, grades[x])
```

```
'A' in grades  
True  
90 in grades  
False  
90 in grades.values()  
True  
100 in grades.values()  
False  
[100, 95] in grades.values()  
True
```

## 02. 딕셔너리

### II. 딕셔너리 사용법

#### 실습 8-9

#### 딕셔너리의 탐색과 추출, 정렬하기

- ③ 만약 딕셔너리에 있는 항목을 정렬하고 싶다면 내장 함수 `sorted()`를 사용, 딕셔너리의 키 값이 동일한 데이터형이어야 가능하고 **정렬 결과는 리스트형이 됨**

```
>>> s = sorted(adic.items()) # 오름차순 정렬
>>> s
[('a', 90), ('b', 80), ('c', 60), ('d', 70)]
```

- ④ 정렬된 결과를 딕셔너리로 변환하려면 `dict()`를 사용

```
>>> s = dict(sorted(adic.items(), reverse=True)) # 내림차순 정렬
>>> s
{'d': 70, 'c': 60, 'b': 80, 'a': 90}
```

딕셔너리로 변환

```
adic
{'a': 90, 'b': 70, 'c': 60, 'd': 70}
adic = dict(sorted(adic.items(), reverse=True))
adic
{'d': 70, 'c': 60, 'b': 70, 'a': 90}
```

## 02. 딕셔너리

### III. 딕셔너리의 활용

실습 8-10

딕셔너리로 전체 학생의 평균 점수 구하기

code08-10.py

- ① 딕셔너리의 값을 모두 추출하고, 반복문으로 합계를 먼저 계산, 반복이 종료되면 평균을 계산해서 출력하는 순서로 코드를 만들어 저장

```
01 members = {'choi':93, 'han':50, 'jung':92, 'kang':68, 'kim':80,  
02           'lee':90, 'moon':65, 'na':100, 'park':75, 'song':75}  
03  
04 total = 0  
05 for x in members.values():      # 항목의 값을 모두 추출하고, x에 하나씩 대입하는 반복문  
06     total += x  
07  
08 print("회원 점수 평균 =", total / len(members))
```

- ② 저장한 프로그램을 실행시켜 결과를 확인

회원 점수 평균 = 78.8

## 02. 딕셔너리

### III. 딕셔너리의 활용

실습 8-11

딕셔너리로 도서 검색 프로그램 만들기

code08-11.py

- 도서명을 입력하면 해당 도서의 가격을 알려주고, 종료 조건(0)을 입력하면 프로그램을 종료
  - 만약 도서명을 잘못 입력하면, 오류가 발생하지 않도록 처리하고 안내 메시지를 보여줌
- ① 무한 반복을 위한 while 문장 안에서 입력과 검색, 출력이 모두 이루어지는 구조

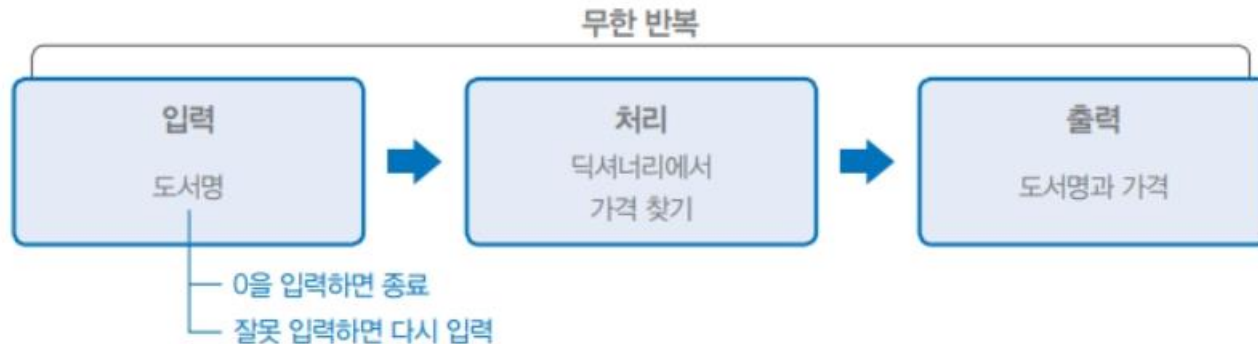


그림 8-19 도서 검색 프로그램 실행 순서

## 02. 딕셔너리

### III. 딕셔너리의 활용

실습 8-11

딕셔너리로 도서 검색 프로그램 만들기

code08-11.py

- ② 동작 순서에 따라 다음과 같이 프로그램을 만들고 저장

```
01 books = {'여행의 이유':13500, '소년이로':13000, '희랍인 조르바':9000,  
02         '세 여자':14000, '아픔이 길이 되려면':18000}  
03 print("검색 가능 도서 :", list(books.keys()))      # 도서명을 모두 출력  
04 print('-'*35)  
05  
06 while True :  
07     bookName = input("도서명 입력(검색 종료는 0) : ")  
08     if bookName in books :  
09         print(bookName, "=", books.get(bookName), "원\n")  
10     elif bookName == '0' :  
11         print("프로그램을 종료합니다.")  
12         break  
13     else :  
14         print("검색 가능한 도서가 아닙니다.\n")
```

## 02. 딕셔너리

### III. 딕셔너리의 활용

실습 8-12

모스 부호 사용하기

code08-12.py

- 모스 부호는 짧은 전류(.)와 긴 전류(-)를 조합하여 알파벳과 숫자를 표기하는 신호 체계
- 알파벳 단어를 입력하면 해당하는 모스 부호를 출력하는 프로그램

A	B	C	D	E
. _ .	_ . .	_ . .	_ . .	. .
F	G	H	I	J
. . . .	_ . .	. . . .	. .	. _ . _
K	L	M	N	O
_ . _	. _ . .	_ _	_ .	_ _ _
P	Q	R	S	T
. _ . _	_ . _ .	. . .	. . .	_
U	V	W	X	Y
. . _	. . . _	. _ . _	_ . . .	_ . _
MORSE		Z	CODE	
		_ . . .		

S O S

그림 8-20 모스부호 체계

## 02. 딕셔너리

### III. 딕셔너리의 활용

실습 8-12

모스 부호 사용하기

code08-12.py

- ① 알파벳 문자를 키로 하고, 모스 부호는 값으로 설정

```
01 code = {'A': '.-.', 'B': '-....', 'C': '-.-.', 'D': '-...', 'E': '.', 'F': '..-.', 'G': '---.',  
02         'H': '....', 'I': '...', 'J': '---', 'K': '-.-.', 'L': '.-..', 'M': '---', 'N': '-.',  
03         'O': '---', 'P': '.-.-.', 'Q': '--.-', 'R': '.-..', 'S': '...', 'T': '-.', 'U': '..-',  
04         'V': '...-', 'W': '.-.-', 'X': '-.-.', 'Y': '-.-.', 'Z': '---.'}
```

- ② 단어를 구성하는 각 문자를 키로 사용하고, 해당하는 값을 찾아 출력, 키가 아닌 문자가 입력되는 경우에는 'None'을 출력

```
05 word = input("모스 부호로 표시할 단어(알파벳 대문자) : ")  
06 word = word.upper() #대문자로 변환 #소문자 변환은 lower()  
07 for ch in word :  
08     if ch in code :  
09         print(code.get(ch), end = " ") # 부호마다 뒤에 빈칸 추가  
10     else :  
11         print("None", end = " ")
```

- ③ 저장한 프로그램을 실행시켜 결과를 확인

```
모스 부호로 표시할 단어(알파벳 대문자) : SOS  
... --- ...
```

## 02. 딕셔너리

### III. 딕셔너리의 활용

여기서 잠깐

문자열의 구조

- 문자열(string)은 하나 이상의 문자를 순서대로 저장하는 데이터형으로, 리스트처럼 인덱싱할 수 있어서 for 문에도 동일하게 사용

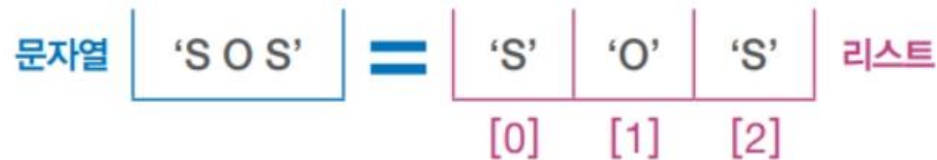


그림 8-21 문자열과 리스트 비교



## 02. 딕셔너리

### [실습] 카페 메뉴표를 딕셔너리로 구성하기 (소분류 메뉴표 생성) Ch08-Menu00.py

- ◆ 아래의 카페 메뉴를 참조하여 소분류 메뉴 딕셔너리(menu)를 구성하시오.
  - Key는 대분류 메뉴명으로 하고, 키에 대응되는 값은 [[메뉴명, 단가(천단위)], [, ... ]]로 구성

#(소분류) 메뉴표 만들기

menu = {'COFFEE': [['에스프레소', 3.0], ['아메리카노', 3.0], ['카페라떼', 4.0], ['카프치노', 4.0]],

'LATTE': [['말차라떼', 4.0], ['초코라떼', 4.0], ['카페라떼', 4.0]],

'TEA': [['청귤차', 4.0], ['자몽차', 4.0],

'ADE': [['자몽에이드', 4.5], ['레몬에이드', 4.5],

'JUICE': [['망고', 4.5], ['바나나', 4.5],

'SMOOTHIE': [['청귤스무디', 4.5], ['요거트스무디', 4.5],

'MILK TEA': [['흑당밀크티', 4.5], ['달고나밀크티', 4.5]]

COFFEE		LATTE		TEA		MILK TEA	
에스프레소	3.0	말차라떼	4.0	청귤차	4.0	흑당밀크티	4.5
아메리카노	3.0	초코라떼	4.0	자몽차	4.0	달고나밀크티	4.5
카페라떼	4.0	카페라떼	4.0	레몬차	4.0		
카프치노	4.0			카모마일	4.5		
ADE		JUICE		SMOOTHIE			
자몽에이드	4.5	망고	4.5	청귤스무디	4.5		
레몬에이드	4.5	바나나	4.5	요거트스무디	4.5		
청포도에이드	4.5	딸기	4.5				
		키위	4.5				

#[출력] 소분류 메뉴 딕셔너리

```
print("\n[[ 소분류 메뉴 ]])")
```

```
for title, mlist in menu.items():
```

```
    print(title, " : ", mlist)
```

[[ 소분류 메뉴 ]]

COFFEE : [['에스프레소', 3.0], ['아메리카노', 3.0], ['카페라떼', 4.0], ['카프치노', 4.0]]

LATTE : [['말차라떼', 4.0], ['초코라떼', 4.0], ['카페라떼', 4.0]]

TEA : [['청귤차', 4.0], ['자몽차', 4.0], ['레몬차', 4.0], ['카모마일', 4.5]]

ADE : [['자몽에이드', 4.5], ['레몬에이드', 4.5], ['청포도에이드', 4.5]]

JUICE : [['망고', 4.5], ['바나나', 4.5], ['딸기', 4.5], ['키위', 4.5]]

SMOOTHIE : [['청귤스무디', 4.5], ['요거트스무디', 4.5]]

MILK TEA : [['흑당밀크티', 4.5], ['달고나밀크티', 4.5]]

## 02. 딕셔너리

### [실습] 카페 메뉴표를 딕셔너리로 구성하기 (소분류 메뉴표 출력) Ch08-Menu01.py

◆ 소분류 메뉴표 출력하시오.

- 소분류 메뉴 딕셔너리(menu)를 참조하여 대분류 단위로 출력

```
#[출력] 소분류 메뉴
print("[[ 메뉴 ]]")
for title, mlist in menu.items():
    print("## %s ##" %title)
    for mmenu, price in mlist:
        print("%-10s%5.1f" %(mmenu, price))
    print()
```

```
[[ 메뉴 ]]
## COFFEE ##
에스프레소      3.0
아메리카노      3.0
카페라떼        4.0
카프치노        4.0

## LATTE ##
말차라떼        4.0
초코라떼        4.0
카페라떼        4.0

## TEA ##
청굴차          4.0
자몽차          4.0
레몬차          4.0
카모마일        4.5
```

## 02. 딕셔너리

### [실습] 카페 메뉴표를 딕셔너리로 구성하기 (대분류 메뉴표 생성) Ch08-Menu02.py

◆ 대분류 메뉴 딕셔너리(menugroup)를 구성하시오.

- 이미 구성된 소분류 메뉴 딕셔너리(menu)를 참조하여 구성
- Key는 0부터 시작하는 순서번호로 하고, 키에 대응되는 값은 대분류 메뉴명으로 구성

#대분류 메뉴표 딕셔너리 생성

```
menugroup = {}
```

```
no = 0
```

```
for title in menu.keys():
```

```
    menugroup[no] = title    #딕셔너리에 항목 추가
```

```
    no += 1
```

#[출력] 대분류 메뉴표 딕셔너리

```
print("[[ 대분류 메뉴 ]])")
```

```
for no, title in menugroup.items():
```

```
    print("%d : %-10s" %(no, title))
```

[[ 대분류 메뉴 ]]

0 : COFFEE

1 : LATTE

2 : TEA

3 : ADE

4 : JUICE

5 : SMOOTHIE

6 : MILK TEA

## 02. 딕셔너리

### [실습] 카페 메뉴표를 딕셔너리로 구성하기 (대분류 메뉴 선택) Ch08-Menu03.py

◆ 대분류 메뉴표 출력하여주고, 그 중 하나를 선택하는 코드를 완성하시오.

- 키보드로 대분류 메뉴의 키Key를 입력 받아 처리
- 입력 받은 키로 소분류 메뉴 딕셔너리(menu)를 참조하여 해당 리스트를 출력

```
#[주문] 대분류 메뉴 선택
print("\n>> [선택] 대분류 메뉴 <<")
for no, title in menugroup.items():
    print("[%d] %s" %(no, title)) #대분류 메뉴 출력

mcnt = len(menugroup)
nogroup = -1
while nogroup < 0 or nogroup >= mcnt:
    print('-'*30)
    nogroup = int(input(">>메뉴 그룹(번호) 선택: "))
inkey = menugroup.get(nogroup) #소분류 메뉴(menu) 키 찾기
mlist = menu.get(inkey) # 소분류 메뉴(menu) 키로 소분류 메뉴 리스트 구성(찾기)
print(mlist) # 소분류 메뉴 출력
```

```
>> [선택] 대분류 메뉴 <<
[0] COFFEE
[1] LATTE
[2] TEA
[3] ADE
[4] JUICE
[5] SMOOTHIE
[6] MILK TEA
-----
>>메뉴 그룹(번호) 선택: 1
[['말차라떼', 4.0], ['초코라떼', 4.0], ['카페라떼', 4.0]]
```

## 02. 딕셔너리

### [실습] 카페 메뉴표를 딕셔너리로 구성하기 (소분류 메뉴 선택) Ch08-Menu04.py

- ◆ 선택된 대분류 메뉴의 소분류 메뉴표 출력하여주고, 그 중 하나를 선택하는 코드를 완성하시오.
  - 선택된 대분류 메뉴로 소분류 메뉴 리스트(mlist)를 출력, 이때 순서번호도 함께 출력
  - 소분류 메뉴의 순서번호를 키보드로 입력 받아 메뉴 선택, 선택된 메뉴명 출력

```
#[주문] 소분류 메뉴 선택
print("\n>> [%s] 소분류 메뉴 <<" %inkey)
seq = 0
for mmenu, price in mlist:          #선택 메뉴 출력
    print("[%d] %-10s\t%.5f" %(seq, mmenu, price))
    seq += 1                        #순서번호 생성

mcnt = len(mlist)                   #선택 가능한 메뉴 수
nomenu = -1                         #메뉴 선택번호
while nomenu < 0 or nomenu >= mcnt:
    print('-'*30)
    nomenu = int(input(">>메뉴(번호) 선택: "))
print("> %s를 선택하셨습니다." %mlist[nomenu][0])
```

```
>> [선택] 대분류 메뉴 <<
[0] COFFEE
[1] LATTE
[2] TEA
[3] ADE
[4] JUICE
[5] SMOOTHIE
[6] MILK TEA
-----
>>메뉴 그룹(번호) 선택: 1

>> [LATTE] 소분류 메뉴 <<
[0] 말차라떼          4.0
[1] 초코라떼          4.0
[2] 카페라떼          4.0
-----
>>메뉴(번호) 선택: 2
> 카페라떼를 선택하셨습니다.
```

## 02. 딕셔너리

### [실습] 카페 메뉴표를 딕셔너리로 구성하기 (주문내역 리스트 생성) Ch08-Menu05.py

- ◆ 선택된 소분류 메뉴에 대한 주문 수량을 입력받아 주문내역 리스트(selmlist)를 구성하시오.
  - 주문 리스트(selmlist)는 [메뉴명, 단가, 주문수량, 금액]로 구성
  - 금액은 (단가\*주문수량)으로 계산, 구성된 주문 리스트(selmlist)를 출력

```
selmlist = []    #주문내역 리스트 생성
```

```
#[주문] 주문내역 리스트 생성
```

```
inqty = 0    #주문 수량
```

```
while inqty <= 0:
```

```
    inqty = int(input(">>몇 잔을 원하십니까? "))
```

```
print("> [%s]를 [%d]잔 선택하셨습니다." %(mlist[nomenu][0], inqty))
```

```
totprice = int(mlist[nomenu][1]*inqty*1000)    #주문내역 리스트에 추가
```

```
selmlist.append([mlist[nomenu][0], mlist[nomenu][1], inqty, totprice])
```

```
print(selmlist)
```

```
-----  
>>메뉴 그룹(번호) 선택: 1
```

```
>> [LATTE] 소분류 메뉴 <<
```

```
[0] 말차라떼          4.0
```

```
[1] 초코라떼          4.0
```

```
[2] 카페라떼          4.0  
-----
```

```
>>메뉴(번호) 선택: 2
```

```
> 카페라떼를 선택하셨습니다.
```

```
>>몇 잔을 원하십니까? 3
```

```
> [카페라떼]를 [3]잔 선택하셨습니다.
```

```
[['카페라떼', 4.0, 3, 12000.0]]
```

## 02. 딕셔너리

### [실습] 카페 메뉴표를 딕셔너리로 구성하기 (계속 주문 처리)

Ch08-Menu06.py

- ◆ 메뉴 선택을 계속 할 수 있도록 코드를 수정하고, 마지막에 주문 내역(selmlist)을 출력하시오.
  - 주문 종료는 키보드로 'x'나 'X' 값을 입력받아 처리, 기타 키는 주문 계속으로 처리
  - 최종 주문내역(selmlist)은 주문한 (메뉴명, 단가, 주문 수량, 금액)을 나열
  - 마지막에 주문 합계를 (총 주문 잔 수, 총 청구 금액)으로 출력

```
#[주문] 주문 반복 처리
```

```
insel = "      #키보드 입력(계속 여부)
```

```
selmlist = []  #주문내역 리스트 생성
```

```
while True:
```

```
    #[주문] 대분류 메뉴 선택
```

```
    print("\n>> [선택] 대분류 메뉴 <<")
```

```
    #중간 생략#
```

```
    insel = input(">>계속 주문을 하시겠습니까? [종료: x, 계속: Enter] ")
```

```
    if insel == 'x' or insel == 'X':
```

```
        break
```

```
[2] 레몬차          4.0
```

```
[3] 카모마일        4.5
```

```
-----
```

```
>>메뉴(번호) 선택: 3
```

```
> 카모마일을 선택하셨습니다.
```

```
>>몇 잔을 원하십니까? 4
```

```
> [카모마일]를 [4]잔 선택하셨습니다.
```

```
>>계속 주문을 하시겠습니까? [종료: x, 계속: Enter] x
```

## 02. 딕셔너리

### [과제-3] 카페 메뉴표를 딕셔너리로 구성하기 (주문 내역서 출력)

- ◆ 메뉴 선택을 계속 할 수 있도록 코드를 수정하고, 마지막에 주문 내역(selmlist)을 출력하시오.
  - 주문 종료는 키보드로 'x'나 'X' 값을 입력받아 처리, 기타 키는 주문 계속으로 처리
  - 최종 주문내역(selmlist)은 주문한 (메뉴명, 단가, 주문 수량, 금액)을 나열
  - 마지막에 주문 합계를 (총 주문 잔 수, 총 청구 금액)으로 출력

#[주문] 주문 반복 처리

inse1 = " #키보드 입력(계속 여부)

selm1ist = [] #주문내역 리스트 생성

while True:

#[주문] 대분류 메뉴 선택

print("\n>> [선택] 대분류 메뉴 <<")

#중간 생략#

inse1 = input(">>계속 주문을 하시겠습니까? [종료: x, 계속: Enter]

if inse1 == 'x' or inse1 == 'X':

break

#[출력] 주문 결과

[2] 레몬차 4.0  
[3] 카모마일 4.5

>>메뉴(번호) 선택: 3

> 카모마일을 선택하셨습니다.

>>몇 잔을 원하십니까? 4

> [카모마일]를 [4]잔 선택하셨습니다.

>>계속 주문을 하시겠습니까? [종료: x, 계속: Enter] x

===== >> 주문 결과 << =====

카페라떼 4000 3 12000

카모마일 4500 4 18000

주문 합계 : [7]잔 30000원



## 02. 딕셔너리

### [과제-3 도전] 카페 메뉴표를 딕셔너리로 구성하기 (주문 내역 항목 취소)

- ◆ 마지막에 주문내역(selmlist)이 확정되기 전에 주문내역 항목을 삭제할 수 있도록 Upgrade하시오.
  - 항목 번호를 출력해주고, 해당 번호를 선택하면 주문 리스트에서 해당 항목만 제거

```
#[주문] 주문 반복 처리
insel = "          #키보드 입력(계속 여부)
selmlist = []      #주문내역 리스트 생성
while True:
    #[주문] 대분류 메뉴 선택
    print("\n>> [선택] 대분류 메뉴 <<")

    #중간 생략#

    insel = input(">>계속 주문을 하시겠습니까? [종료: x, 계속: Enter] ")
    if insel == 'x' or insel == 'X':
        break

#주문 항목 선택 취소

#[출력] 최종 주문 결과
```

```
>>계속 주문을 하시겠습니까? [종료: x, 계속: Enter] x
===== >> 주문 결과 << =====
[0] 요거트스무디      1
[1] 청포도에이드      2
[2] 키위              3
-----
>>삭제할 메뉴(번호) 선택(없으면 Enter): 1
> 청포도에이드를 취소하겠습니다.
>>정말 취소하시겠습니까? [아니오: n, 예: Enter]
===== >> 주문 결과 << =====
[0] 요거트스무디      1
[1] 키위              3
-----
>>삭제할 메뉴(번호) 선택(없으면 Enter):
===== >> 주문 결과 << =====
요거트스무디      4500   1   4500
키위              4500   3   13500
-----
주문 합계 :           [4]잔   18000원
```

## 03. 리스트, 튜플, 딕셔너리의 심화 내용

### ■ 세트(Set)

- 세트Set는 키만 모아 놓은 딕셔너리의 특수한 형태
- 딕셔너리의 키는 중복되면 안 되므로 세트에 들어 있는 값은 항상 유일
- 세트를 생성하려면 딕셔너리처럼 중괄호 {} 사용하지만 : 없이 값을 입력
- 중복된 키는 자동으로 하나만 남음

```
mySet1 = {1, 2, 3, 3, 3, 4}  
mySet1
```

출력 결과

```
{1, 2, 3, 4}
```

- 판매된 물품의 전체 수량이 아닌 종류만 파악하고 싶을 때

```
salesList = ['삼각김밥', '바나나', '도시락', '삼각김밥', '삼각김밥', '도시락', '삼각김밥']  
set(salesList)
```

출력 결과

```
{'도시락', '바나나', '삼각김밥'}
```

## 03. 리스트, 튜플, 딕셔너리의 심화 내용

### ■ 세트(Set)

- 두 세트 사이의 교집합, 합집합, 차집합, 대칭(양방향) 차집합을 구할 때

```
mySet1 = {1, 2, 3, 4, 5}
mySet2 = {4, 5, 6, 7}
mySet1 & mySet2      # 교집합
mySet1 | mySet2      # 합집합
mySet1 - mySet2      # 차집합
mySet1 ^ mySet2      # 대칭 차집합
```

#### 출력 결과

```
{4, 5}
{1, 2, 3, 4, 5, 6, 7}
{1, 2, 3}
{1, 2, 3, 6, 7}
```

- 연산자 &, |, -, ^ 대신 함수를 사용

```
mySet1.intersection(mySet2)      # 교집합
mySet1.union(mySet2)              # 합집합
mySet1.difference(mySet2)         # 차집합
mySet1.symmetric_difference(mySet2) # 대칭 차집합
```

## 03. 리스트, 튜플, 딕셔너리의 심화 내용

### ■ zip()

- zip() 함수는 동시에 여러 리스트에 접근하여 같은 인덱스끼리 매칭시켜준다.

```
foods = ['떡볶이', '짜장면', '라면', '피자', '맥주', '치킨', '삼겹살']  
sides = ['오텍', '단무지', '김치']  
for food, side in zip(foods, sides):  
    print(food, ' --> ', side)
```

#### 출력 결과

```
떡볶이 --> 오텍  
짜장면 --> 단무지  
라면 --> 김치
```

## 03. 리스트, 튜플, 딕셔너리의 심화 내용

### ■ zip()

- zip() 함수는 두 리스트를 튜플이나 딕셔너리로 짝지을 때 zip() 함수 사용

```
foods = ['떡볶이', '짜장면', '라면', '피자', '맥주', '치킨', '삼겹살']  
sides = ['오뎅', '단무지', '김치']  
tupList = list(zip(foods, sides))  
dic = dict(zip(foods, sides))  
tupList  
dic
```

#### 출력 결과

```
[('떡볶이', '오뎅'), ('짜장면', '단무지'), ('라면', '김치')]  
{'떡볶이': '오뎅', '짜장면': '단무지', '라면': '김치'}
```

# Thank You !

[Python]