목차

01

튜플

01. 튜플

[검색의 유용함]

- 길 찾기, 점심 메뉴 선정, 날씨 확인 등 검색은 시시각각 일어나고 있다.
- 튜플(tuple)은 리스트처럼 여러 개의 데이터를 저장할 수 있는 자료형이다.
- 변경이 불가능하여 반복문을 활용한 검색 작업에 사용하기 좋은 구조이다.

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 저장할 수 있고, 0부터 시작하는 정수형 인덱스를 이용해 각 항목을 참조하거나 슬라이싱
- 리스트에서 사용한 대괄호([ ]) 대신 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)
```

- **튜플은 리스트와 달리 항목을 추가하거나 삭제, 변경하는 것이 불가능**

01. 튜플

I. 튜플의 개념과 생성

실습 8-1 튜플 생성하기

① 튜플은 소괄호를 이용해서 정의하지만, 소괄호를 생략하고 값만 입력해도 됨

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

② 리스트를 튜플로 바꾸거나, 반대로 튜플을 리스트로 바꿀 수도 있음

```
>>> alist = [2, 4, 6, 8, 10]
>>> c = tuple(alist)          # 리스트를 튜플로 변환
>>> c
(2, 4, 6, 8, 10)
>>> blist = list(b)           # 튜플을 리스트로 변환
>>> blist
[10, 20]
```

01. 튜플

I. 튜플의 개념과 생성

실습 8-1 튜플 생성하기

③ range( ) 함수를 이용해서 규칙적인 간격을 갖는 정수로 구성된 튜플을 만들 수도 있음

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

01. 튜플

I. 튜플의 개념과 생성

여기서 잠깐 **콤마를 이용한 튜플 생성**

- 다음 코드는 각각 변수와 튜플을 정의하는 문장으로, 저장되는 데이터는 하나의 값이지만 완전히 다른 자료 구조로 변환

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

01. 튜플

II. 튜플 사용법

실습 8-2 튜플 다루기

① 튜플의 특정 위치나 범위에 있는 항목을 참조하기 위한 인덱싱과 슬라이싱을 실

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

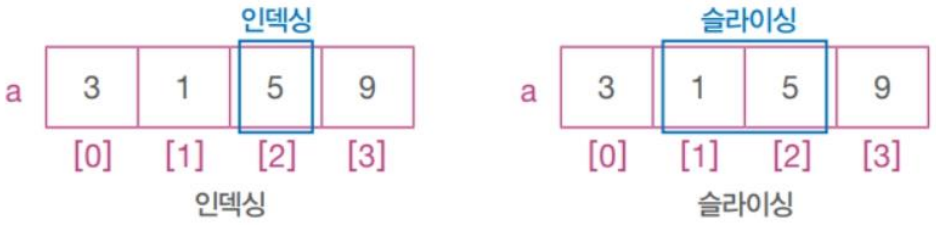


그림 8-2 튜플의 인덱싱과 슬라이싱

② 튜플을 피연산자로 하는 덧셈(+)과 곱셈(*) 연산식을 만들고 결과를 확인

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

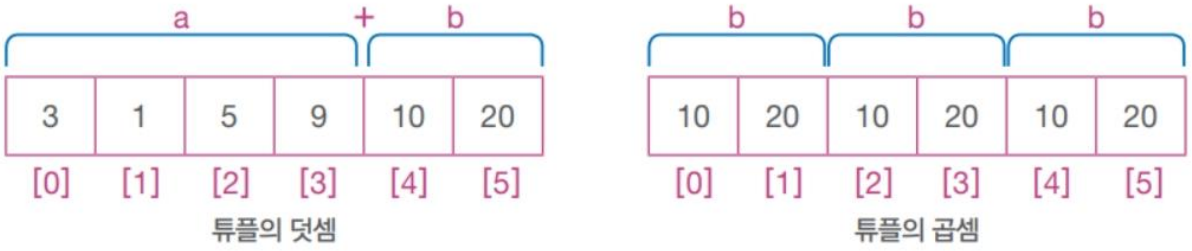


그림 8-3 튜플의 덧셈과 곱셈

II. 튜플 사용법

실습 8-2 튜플 다루기

③ for 반복문을 사용하면 튜플의 값을 하나씩 출력

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

④ 리스트처럼 항목의 추가나 변경, 삭제는 실행할 수 없음

```
>>> 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 튜플의 길이인 len()이나 항목의 위치인 index(), 합계인 sum(), 최소 값인 min() 과 최대 값인 max() 등 리스트에서 소개한 함수들을 튜플에도 사용할 수 있음

```
>>> 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'
```

II. 튜플 사용법

실습 8-3 튜플의 패킹과 언패킹

① 패킹은 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 언패킹은 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
```

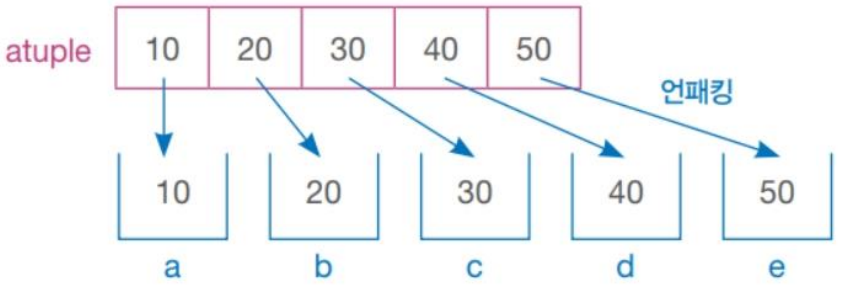


그림 8-4 언패킹의 개념

II. 튜플 사용법

실습 8-3 튜플의 패킹과 언패킹

③ 언패킹할 때 변수 이름 앞에 '*' 기호를 붙이면 여러 개의 값을 갖는 리스트

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

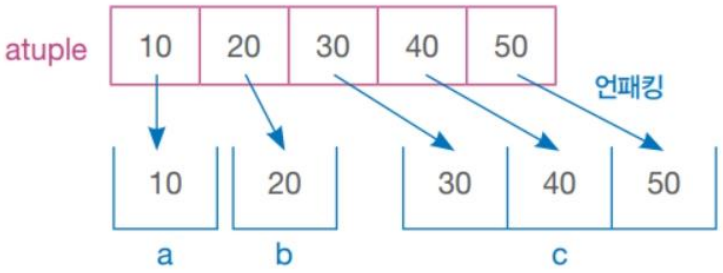


그림 8-5 리스트가 있는 언패킹 과정

III. 파일 처리 시스템의 문제점

- 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 있으 므로 회원 아이디를 탐색하는 처리 과정에 'in' 연산이나 반복문이 사용

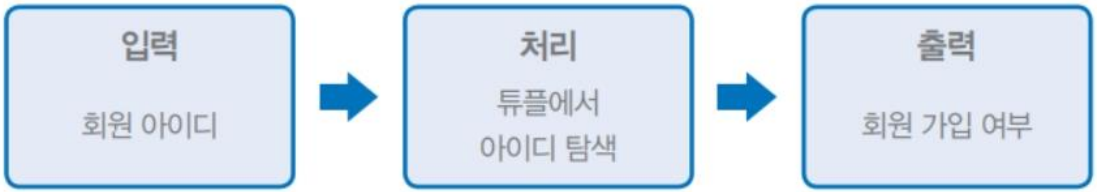


그림 8-6 회원 아이디 검색 후 가입 여부 판단 동작 순서

III. 파일 처리 시스템의 문제점

실습 8-4 회원 가입 여부 확인하기 code08-04.py

② 아이디만 추출해서 리스트에 추가해 두고 'in'으로 찾음

```
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.
```

III. 파일 처리 시스템의 문제점

여기서 잠깐 **for 반복문을 변형하여 프로그램 간단하게 만들기**

- 'in' 연산에 for 반복문을 내포시켜 다음과 같이 코드를 간단하게 변경 가능

```
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("검색할 아이디 입력 : ")
04 if search in (x[0] for x in members) :
05     print("가입한 회원입니다.")
06 else :
07     print("회원이 아닙니다.")
```

아이디만 추출해서 목록을 생성

III. 파일 처리 시스템의 문제점

실습 8-5 만족도 점수가 가장 높은 회원 찾기 code08-05.py

- 알고리즘 규칙
  - 1단계 : 튜플의 첫 번째 항목은 점수(50)가 변수 num의 초기 값(0)보다 크기 때문에 id와 num 변수를 첫 항목의 값으로 수정

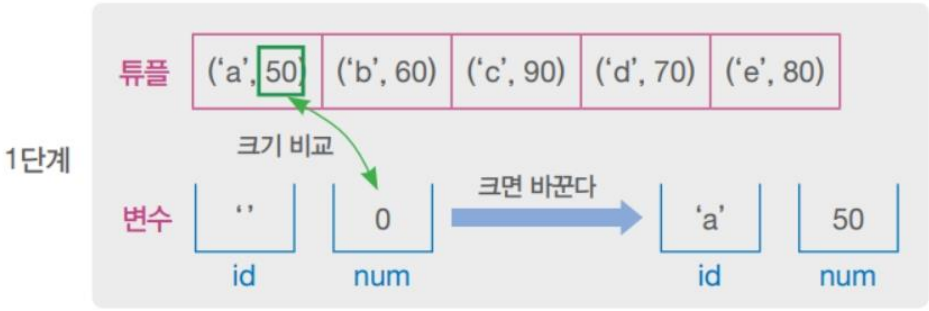


그림 8-7 변수의 초기 값과 첫 번째 항목 비교

  - 2단계 : 튜플의 두 번째 항목 역시 점수(60)가 변수 num(50)보다 더 크므로, 변수 num의 값을 다시 수정

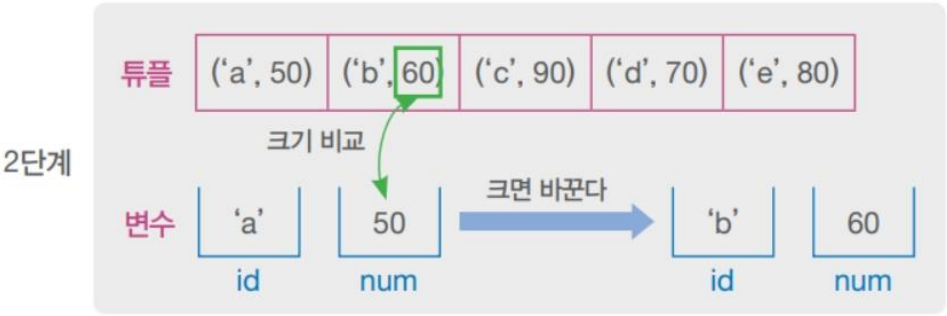


그림 8-8 변수와 두 번째 항목 비교

III. 파일 처리 시스템의 문제점

실습 8-5 만족도 점수가 가장 높은 회원 찾기 code08-05.py

- 알고리즘 규칙
  - 3단계 : 세 번째 항목도 점수(90)가 변수 num(60)보다 더 큰 값이라, 변수 num의 값을 세 번째 항목으로 수정

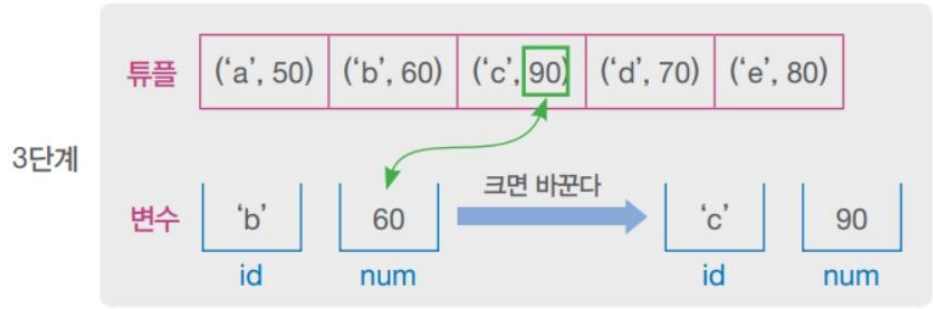


그림 8-9 변수와 세 번째 항목 비교

  - 4, 5단계 : 네 번째와 다섯 번째 항목은 점수가 변수 num보다 작아서 변경할 필요가 없음

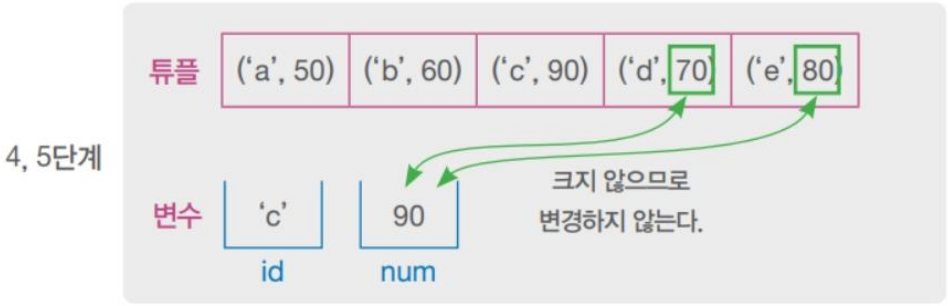


그림 8-10 변수와 나머지 항목 비교

III. 파일 처리 시스템의 문제점

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.
```

III. 파일 처리 시스템의 문제점

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.")
```

III. 파일 처리 시스템의 문제점

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

```
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
```

III. 파일 처리 시스템의 문제점

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.
```

02

딕셔너리

02. 딕셔너리

I. 딕셔너리 개념과 생성

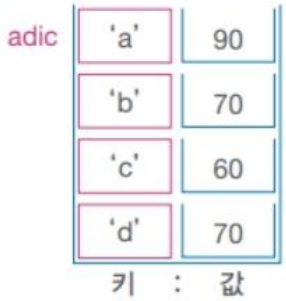
- 리스트나 튜플처럼 여러 개의 데이터를 처리할 수 있는 딕셔너리는 중괄호({ })안에 키와 값을 묶어서 하나의 항목으로 저장
- 콜론(:) 기호로 키와 값을 연결하고, 각 항목 사이에는 리스트나 튜플처럼 쉼표(,)를 사용해서 구분

```
딕셔너리명 = {키1:값1, 키2:값2, ... 키n:값n}
```

- 키와 값은 사용자가 지정하는 것으로, 정수나 문자열 같은 대부분의 데이터형, 리스트나 튜플, 변수도 사용 가능

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

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



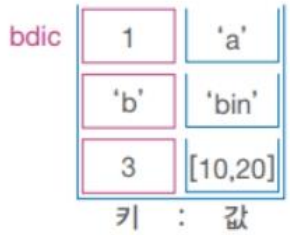


그림 8-17 딕셔너리 구조

02. 딕셔너리

I. 딕셔너리 개념과 생성

- 딕셔너리는 리스트나 튜플처럼 순서 번호를 인덱스로 사용하지 않고, 키를 통해 값을 가져옴

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

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

그림 8-18 딕셔너리의 키를 이용한 값 참조

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'
```

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

실습 8-9 딕셔너리의 탐색과 추출, 정렬하기

① 딕셔너리를 참조할 때 잘못된 키를 사용하면 오류가 발생하므로 in 연산자로 키가 있는지 먼 저 탐색하거나, get( ) 메소드를 이용하여 처리

```
>>> 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 값을 수정하는 문장에서 만약 기존에 없는 키를 사용하면 새 항목이 추가

```
>>> 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

실습 8-9 딕셔너리의 탐색과 추출, 정렬하기

③ 만약 딕셔너리에 있는 항목을 정 렬하고 싶다면 내장 함수 sorted( )를 사용, 정렬하려는 딕셔너리의 키 값이 동일한 데이 터형이어야 정렬 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

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

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
```

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, 출 력이 모두 이루어지는 구조

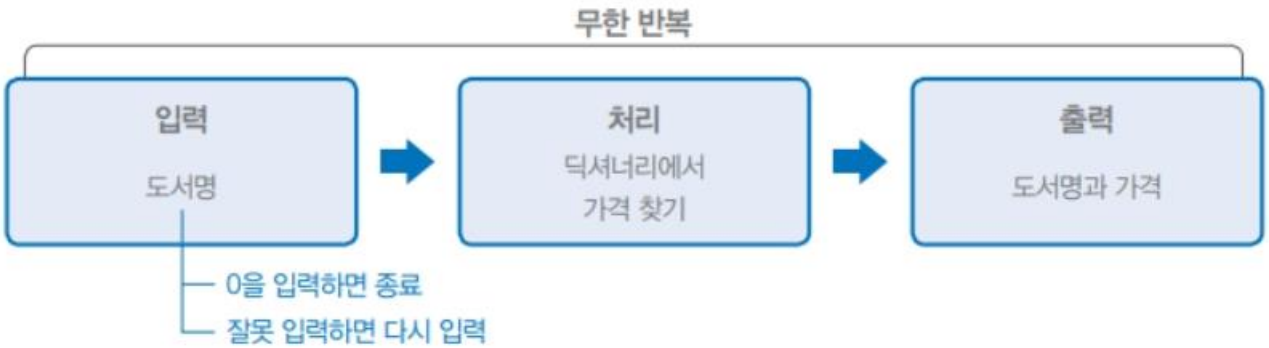


그림 8-19 도서 검색 프로그램 실행 순서

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")
```

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

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
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
... --- ...
```

III. 딕셔너리의 활용

여기서 잠깐 **문자열의 구조**

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

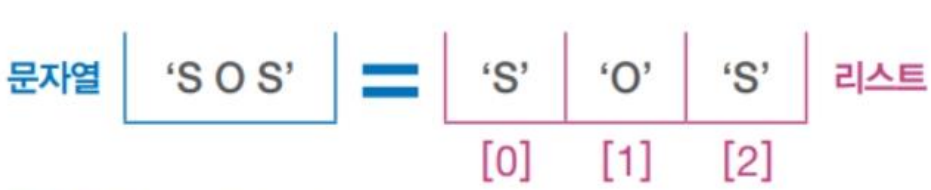


그림 8-21 문자열과 리스트 비교