

파이썬 프로그래밍

10주차 강의자료

나재호

복합 자료형 활용

01 리스트와 함수

02 리스트와 반복문

03 리스트의 반복적 접근 예제: 인구조사

04 이중 리스트

05 딕셔너리와 반복문

06 딕셔너리와 함수

이 단원을 마치고 나면

- 리스트를 함수와 같이 사용할 수 있고 그 특징을 이해할 수 있다.
- 반복문을 이용해 리스트를 반복적으로 접근할 수 있다.
- 이중 리스트의 개념을 이해하고 반복적으로 접근할 수 있다.
- 딕셔너리를 반복문과 같이 사용할 수 있다.
- 딕셔너리를 함수와 같이 사용할 수 있다.



01 | 리스트와 함수

전달할 양이 많다면?

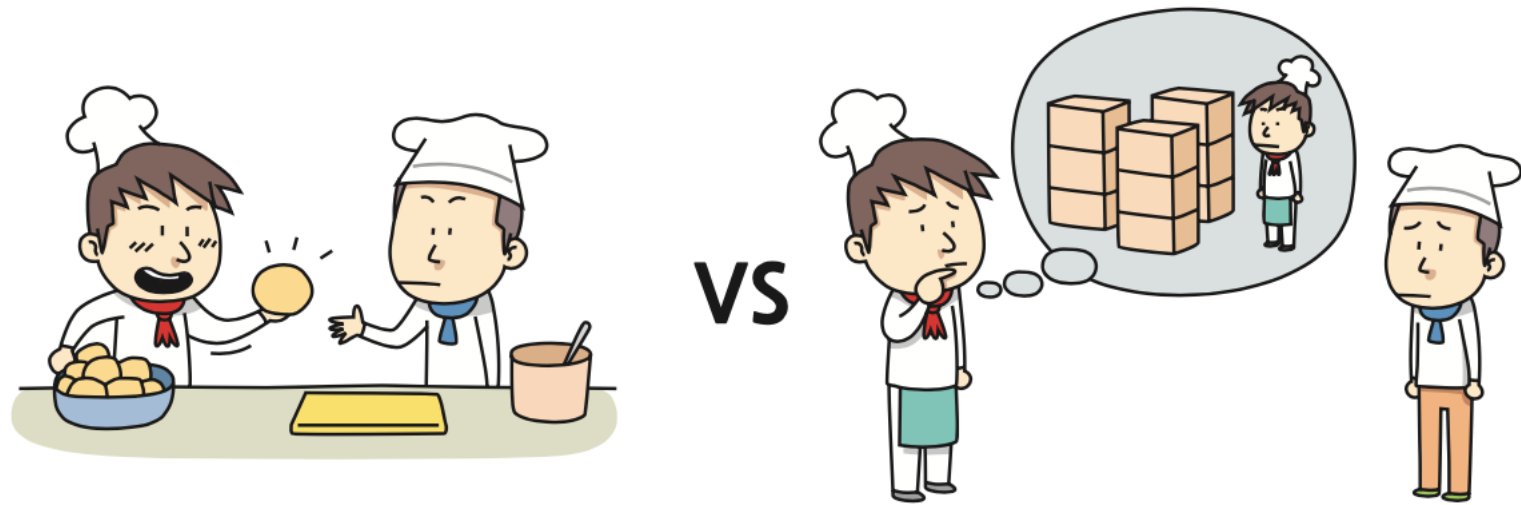


그림 10.2 전달할 양이 많다면?

값 대신 위치를 전달하자!

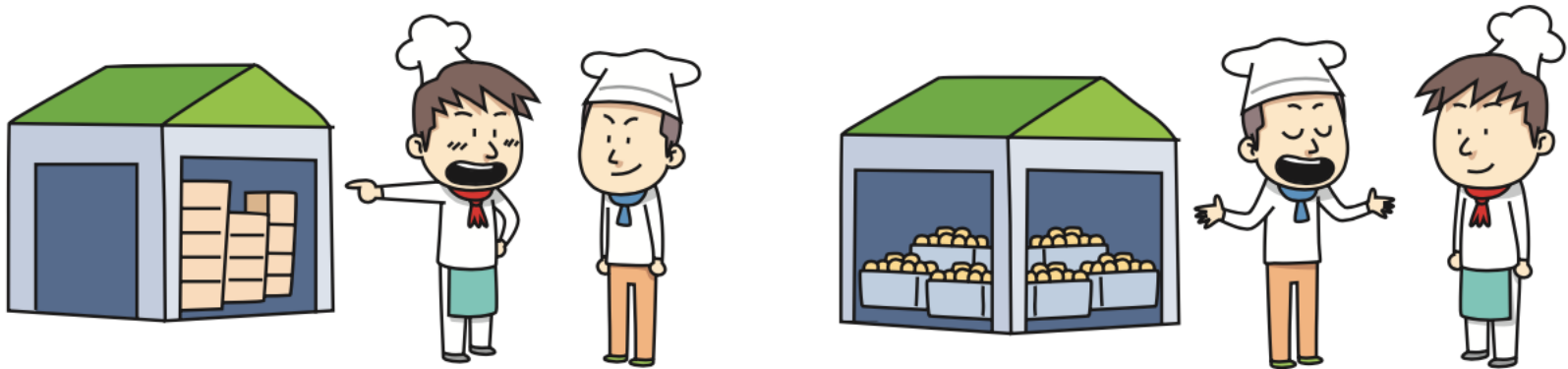


그림 10.3 위치를 전달하자

리스트를 매개변수로 받는 함수

- ◆ 인수로 리스트 지정 시 리스트 객체에 대한 아이디어가 매개변수로 전달
 - 함수에게 효율적으로 리스트 전달 가능
 - 매개변수를 통해 인수인 리스트를 조작하는 효과

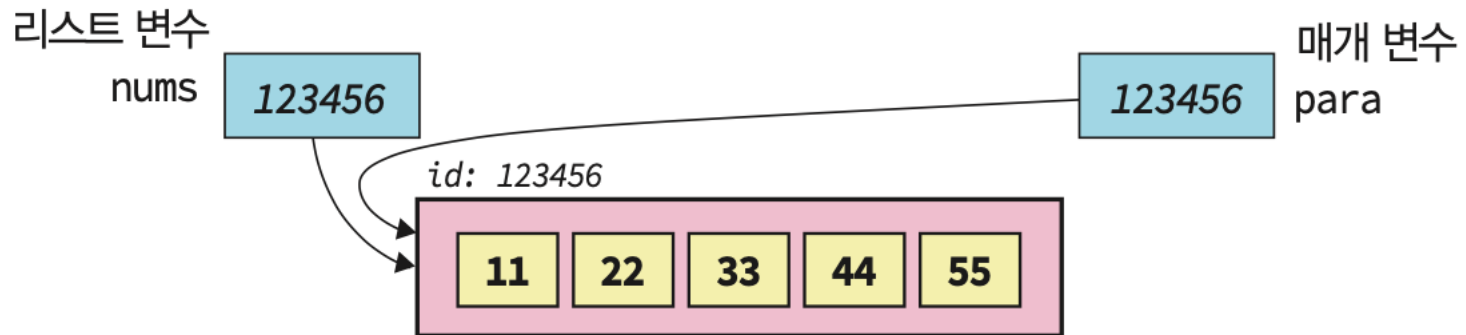


그림 10.1 인수인 리스트를 전달받는 매개변수

함수에 전달된 리스트

- ◆ 함수 내부에서 매개변수를 이용해 실인수인 리스트를 참조할 뿐만 아니라 수정도 가능

코드 10-1 함수에 전달된 리스트

```
01 def input_list(lst) :  
02     name = input('이름? ')  
03     num = input('번호? ')  
04  
05     lst.append(name)  
06     lst.append(num)  
07  
08 # 주 프로그램부  
09 userinfo = []  
10 input_list(userinfo)  
11 print(f'{userinfo[1]}번 {userinfo[0]}')
```

실행 결과

이름? 이찬수

번호? 123

123번 이찬수

코드 10-1의 실행

Python 3.6
([known limitations](#))

```
1 def input_list(lst) :  
2     name = input('이름? ')  
3     num = input('번호? ')  
4  
5     lst.append(name)  
6     lst.append(num)  
7  
8 # 주 프로그램부  
9 userinfo = []  
10 input_list(userinfo)  
11 print(f'{userinfo[1]}번 {userinfo[0]}')
```

[Edit this code](#)

→ line that just executed
→ next line to execute

Step 9 of 10

Print output (drag lower right corner to resize)

이름? 이찬수
번호? 123

Frames

Global frame

- input_list
- userinfo

Objects

- function input_list(lst)
- list
 - 0 "이찬수"
 - 1 "123"

input_list

lst	
name	"이찬수"
num	"123"
Return value	None

(1) 매개변수 리스트에 요소 추가

그림 10.5 매개변수에 접근 및 복귀

리스트를 인수로 갖는 함수

- ◆ 인수로 지정한 리스트의 아이디어가 매개변수에 복사됨
 - 리스트의 각 요소값이 매개변수에 복사되는 것이 아님
- ◆ 함수 내부에서 매개변수로 실인수인 리스트에 접근 가능
 - 리스트의 각 요소값 참조
 - 리스트의 특정 요소를 수정, 삭제, 추가 가능
 - 가변 객체이므로

리스트를 반환하는 함수

- ◆ 리스트 객체에 대한 아이디를 호출측으로 반환
 - 요소값 복사 없이 좀 더 효율적으로 리스트를 전달
 - 한 함수가 한 번에 여러 개의 값을 반환하는 효과



그림 10.6 아예 구입까지 부탁해보자!

리스트를 반환하는 함수의 예

코드 10-2 리스트를 반환하는 함수

```
01 def input_list() :  
02     name = input('이름? ')  
03     num = input('번호? ')  
04  
05     lst = []  
06     lst.append(name)  
07     lst.append(num)  
08     return lst  
09  
10 # 주 프로그램부  
11 userinfo = input_list()  
12 print(f'{userinfo[1]}번 {userinfo[0]}')
```

▶ 실행 결과

이름? 이찬수

번호? 123

123번 이찬수

코드 10-2의 실행

Python 3.6
(known limitations)

```
1 def input_list() :  
2     name = input('이름? ')  
3     num = input('번호? ')  
4  
5     lst = []  
6     lst.append(name)  
7     lst.append(num)  
8     return lst  
9  
10 # 주 프로그램부  
11 userinfo = input_list()  
→ 12 print(f'{userinfo[1]}번 {userinfo[0]}')
```

[Edit this code](#)

→ line that just executed
→ next line to execute

Done running (11 steps)

Print output (drag lower right corner to resize)

```
이름? 이찬수  
번호? 123  
123번 이찬수
```

Frames

Global frame

- input_list
- userinfo

Objects

- function input_list()
- list
 - 0 "이찬수"
 - 1 "123"

그림 10.8 함수에서 반환된 리스트의 사용

연습문제 10.1

- ◆ 리스트 뿐 아니라 튜플로도 한꺼번에 반환 가능
- ◆ 생년월일을 입력받아 만 60번째 생일을 출력하는 프로그램

```
# 사용자 정의 함수부
```

```
def get_date() :  
    y = int(input('연도? '))  
    m = int(input('월? '))  
    d = int(input('일? '))  
    return (y, m, d)
```

```
# 주 프로그램부
```

```
print('당신의 생일을 입력하세요:')  
bday = get_date()  
print(f'당신의 만 60번째 생일은 {bday[0]+60}년 {bday[1]}월 {bday[2]}일입니다.')
```



02 | 리스트와 반복문

리스트와 for 문

- ◆ 유한한 개수의 데이터들이 순서대로 저장된 리스트는 주로 for 문을 이용해 반복

```
for e in 리스트 :  
    리스트_요소값_e를_이용한_문장
```

- 반복 범위 자리에 리스트를 직접 기술 가능
- 매 루프 주기마다 리스트 시작 요소부터 한 요소씩 순서대로 변수 e로 가져와 주어진 문장 반복
 - 변수 e는 다른 이름 사용 가능

리스트의 각 요소값에 대해 반복 처리 예

- ◆ 리스트의 모든 요소에 대해 시작부터 끝까지 순서대로 접근하고자 할 때 유용

#

코드 10-3

리스트와 for 문

```
01 msgs = ['단출하게', '단아하게', '당당하게']
02 for e in msgs :
03     print(f'{e}/', end='')
```



실행 결과

단출하게/단아하게/당당하게/

인덱스를 이용한 반복

- ◆ 리스트의 일부 요소들에 대해서만 반복하거나, 역순으로 접근하고자 할 때 사용

```
for i in range(len(리스트)) :  
    리스트[i]를_이용한_문장
```

- range()를 통해 매 루프 주기마다 리스트에 대한 인덱스 자동 생성
 - 변수명 i는 다른 이름으로 지정 가능
- 전체 요소 반복 시에는 len() 함수를 이용해 리스트의 길이 자동 확인 가능

인덱스를 이용한 반복이 갖는 특징

- ◆ 리스트의 부분만 반복하고자 하는 경우에는 유일한 방법임
 - 전체 요소에 대해 반복시 코드 10-3보다 복잡해 보임

코드 10-4 인덱스를 이용한 리스트 반복

```
01 msgs = ['단출하게', '단아하게', '당당하게']
02 for i in range(len(msgs)) :
03     print(f'{msgs[i]}/', end='')
```

▶ 실행 결과

단출하게/단아하게/당당하게/

마지막 요소부터 역순으로 접근 예

- ◆ 접근하고자 하는 요소의 인덱스를 역순으로 생성
 - `len(msgs)-1` 부터 `-1` 전까지 `-1`씩 증가(`1`씩 감소)

#

코드 10-5

인덱스를 이용해 리스트를 역순으로 반복 접근

```
01 msgs = ['단출하게', '단아하게', '당당하게']
02 for i in range(len(msgs) - 1, -1, -1) :
03     print(f'{msgs[i]}/', end='')
```



실행 결과

당당하게/단아하게/단출하게/



03 |

리스트의 반복적 접근

예제: 인구조사

실습예제 개요

◆ 다음과 같이 인구조사에 사용하는 프로그램을 작성하라

- 지정된 한 빌라에 대해서만 조사함
- 이 빌라는 4층이며, 각 층에 한 가구만 거주함
- 인구조사원은 1층부터 4층까지 각 세대를 방문해 각 가구의 거주 인원수를 입력 받음
- 입력된 정보는 가구별로 분리해 유지해야 함
- 입력을 마치면 각 가구별 거주인원수와 총 거주인원수를 출력

인구조사 프로그램의 실행 예

▶ 실행 결과

1층의 거주인원수는? 3 Enter↵
2층의 거주인원수는? 4 Enter↵
3층의 거주인원수는? 3 Enter↵
4층의 거주인원수는? 2 Enter↵
1층의 거주인원수는 3명입니다.
2층의 거주인원수는 4명입니다.
3층의 거주인원수는 3명입니다.
4층의 거주인원수는 2명입니다.
총 거주인원수는 12명입니다.



그림 10.9 인구 조사

1단계: 주 프로그램부 (설계)

- ◆ 단계별 세분화 전략에 따라 설계
 - 전체 문제를 큰 덩어리로 나눠 의사코드로 표현

코드 10-6 인구조사 프로그램의 의사 코드

01	# 1~4층의 거주인원수를 입력받음
02	# 입력받은 1~4층의 거주인원수를 출력
03	# total = 입력받은 1~4층의 거주인원수의 총합을 구함
04	# 총 거주인원수(total) 출력

1단계: 주 프로그램부 (구현)

◆ 복잡할 수 있는 각 부분은 함수로 구성

#

코드 10-7

인구조사 프로그램의 주 프로그램부 구현

```
01  # 주 프로그램부
02  # 1~4층의 거주인원수를 입력받음
03  population = input_num_of_population()
04
05  # 입력받은 1~4층의 거주인원수를 출력
06  show_num_of_population(population)
07
08  # total = 입력받은 1~4층의 거주인원수의 총합을 구함
09  total = get_total(population)
10
11  # 총 거주인원수(total) 출력
12  print(f'총 거주인원수는 {total}명입니다.')
```


2단계: 각 층의 거주인원수 입력 함수 설계

◆ 이 함수에 맡겨진 요구사항

- 1) 1~4층의 각 세대의 거주인원수를 입력받음
- 2) 입력받은 각 층의 거주인원수를 호출측으로 반환

- 그저 입력만 받으면 되는 것이 아니라
- 입력값을 이후 프로그램의 다른 부분에서 사용할 수 있도록 유지해야 하고,
- 이 함수가 끝난 이후에도 사용할 수 있도록 반환

2단계: 각 층의 거주인원수 입력 함수 설계

◆ 첫 번째 시도

- 개별적인 변수에 입력

코드 10-8 입력 함수의 설계 v1

```
01  def input_num_of_population() :  
02      # nPeople1 = 1층의 거주인원수를 입력받음  
03      # nPeople2 = 2층의 거주인원수를 입력받음  
04      # nPeople3 = 3층의 거주인원수를 입력받음  
05      # nPeople4 = 4층의 거주인원수를 입력받음  
06  
07      # 입력받은 거주인원수를 호출측으로 반환  
08      return (nPeople1, nPeople2, nPeople3, nPeople4)
```

- 비슷한 문장들을 일일이 반복해 가며 나열하는 것은 효율적이지 못함

2단계: 각 층의 거주인원수 입력 함수 설계

◆ 두 번째 시도

- 효율적인 작성을 위해 반복문을 적용
- 1~4까지 혹은 4번 반복하면 되는 구조 → for 문 이용

코드 10-9 입력 함수의 설계 v2 (오류)

```
01 def input_num_of_population() :  
02     # for f를 1부터 4까지 변화하면서 :  
03     # nPeople = f층의 거주인원수를 입력받음  
04  
05     # 입력받은 거주인원수를 호출측으로 반환  
06     return (nPeople1, nPeople2, nPeople3, nPeople4)
```

- 변수 이름은 그 자체로 고유한 식별자로 연산식으로 조합 불가 → 같은 변수에만 저장되는 문제 발생

2단계: 각 층의 거주인원수 입력 함수 설계

◆ 최종

- 반복문 내에서 서로 다른 저장 공간에 입력 받기 위해 리스트 사용
 - 반복문과 리스트는 밀접하게 사용됨

#

코드 10-10

입력 함수의 설계 v3

```
01 def input_num_of_population() :  
02     # 리스트 nPeople 선언  
03     # for f를 1부터 4까지 변화하면서  
04         # nPeople[f - 1] = f층의 거주인원수를 입력받음  
05  
06     # 입력받은 거주인원수를 호출측으로 반환  
07     return nPeople
```

2단계: 각 층의 거주인원수 입력 함수 구현

◆ 1안

- 매 반복 주기마다 1부터 4까지 변화해가며 변수 f에 층수 전달
 - 리스트는 0번 요소부터 존재하므로 리스트 인덱싱 시 $f - 1$ 로 기술해야 함에 유의

#

코드 10-11

입력 함수의 구현 v1

```
01 def input_num_of_population() :
02     nPeople = [0, 0, 0, 0]
03     for f in range(1, 5) :
04         nPeople[f - 1] = int(input(f'{f}층의 거주인원수는? '))
05
06     # 입력받은 거주인원수를 호출측으로 반환
07     return nPeople
```

2단계: 각 층의 거주인원수 입력 함수 구현

◆ 2안

- 좀 더 간단 명료한 표현을 위해 반복문의 인덱스를 리스트의 인덱스에 맞춤
 - f의 범위를 0~3으로 지정, 프롬프트의 층수를 $f + 1$ 로 기술
 - 반복횟수는 `len()`을 이용해 좀 더 유연하게 처리 가능

#

코드 10-12

입력 함수의 구현 v2

```
01 def input_num_of_population() :
02     nPeople = [0, 0, 0, 0]
03     for f in range(4) :      # range(len(nPeople))
04         nPeople[f] = int(input(f'{f + 1}층의 거주인원수는? '))
05
06     # 입력받은 거주인원수를 호출측으로 반환
07     return nPeople
```

2단계: 각 층의 거주인원수 입력 함수 구현

◆ 3안

- 2안의 경우 리스트의 요소값을 일일이 초기화하는 번거로움
- 공백 리스트 준비 후 입력 시 리스트에 요소 추가

#

코드 10-13

입력 함수의 구현 v3

```
01 def input_num_of_population() :
02     nPeople = []
03     for f in range(4) :
04         n = int(input(f'{f + 1}층의 거주인원수는? '))
05         nPeople.append(n)
06
07     # 입력받은 거주인원수를 호출측으로 반환
08     return nPeople
```

3단계: 각 층의 거주인원수 출력 함수 구현

◆ 요구사항

- 매개변수로 전달받은 리스트에 대해 각 요소값을 적절한 형식으로 층수와 함께 출력

◆ 1안

- 매개변수로 전달된 리스트의 길이를 구하고, 인덱스를 이용해 각 요소에 반복 접근

#

코드 10-14

각 층의 거주인원수를 출력하는 함수의 구현 v1

```
01 def show_num_of_population(p) :  
02     cnt = len(p)  
03     for i in range(cnt) :  
04         print(f'{i + 1}층의 거주인원수는 {p[i]}명입니다.')
```


3단계: 각 층의 거주인원수 출력 함수 구현

◆ 2안

- 리스트의 모든 요소에 접근하므로 리스트에 대해 직접 반복 처리 가능

#

코드 10-15

각 층의 거주인원수를 출력하는 함수의 구현 v2

```
01 def show_num_of_population(p) :  
02     f = 1  
03     for n in p :  
04         print(f'{f}층의 거주인원수는 {n}명입니다.')  
05         f += 1
```

- 프롬프트에 층 수 출력을 위해 별도로 변수 f를 둠
 - 초기화는 반복문 시작 전에 1번만 수행되어야 함에 주의

4단계: 총 거주인원수 계산 함수 설계/구현

◆ 요구사항

- 매개변수로 받은 리스트의 모든 요소를 구해 반환

◆ 1안

#

코드 10-16

리스트의 모든 요소의 합계를 구하는 함수의 설계: v1

```
01 def get_total(lst) :  
02     total = lst[0] + lst[1] + lst[2] + lst[3]  
03     return total
```

- 요소의 개수가 많으면 이렇게 나열하는 것은 번거로움
- 매개변수로 전달받은 리스트의 요소 개수가 4개가 아닌 경우에는 적절히 처리하지 못함
- 좀 더 일반화된 형태로 작성할 필요

4단계: 총 거주인원수 계산 함수 설계/구현

◆ 2안

- 매개변수로 전달받은 리스트의 각 요소에 대해 누적합 산출을 위해 반복문 이용할 수 있도록 반복 대상 결정
- +는 이항 연산자

```
total = ( ( ( lst[0] + lst[1] ) + lst[2] ) + lst[3] )
```

#

코드 10-17

리스트의 모든 요소의 합계를 구하는 함수의 설계: v2-1

```
01 def get_total(lst) :  
02     total = lst[0] + lst[1]  
03     total += lst[2]  
04     total += lst[3]  
05  
06     return total
```

4단계: 총 거주인원수 계산 함수 설계/구현

- ◆ 가능하면 좀 더 넓은 범위를 반복 할 수 있도록 수정

#

코드 10-19

리스트의 모든 요소의 합계를 구하는 함수의 설계: v2-3

```
01 def get_total(lst) :  
02     total = 0  
03     total += lst[0]  
04     total += lst[1]  
05     total += lst[2]  
06     total += lst[3]  
07  
08     return total
```

4단계: 총 거주인원수 계산 함수 설계/구현

- ◆ 반복처리하는 문장들을 반복문으로 구성
 - 간결한 구성
 - 어떤 길이의 리스트를 전달받더라도 함수의 수정 없이 처리 가능

#

코드 10-20

리스트의 모든 요소의 합계를 구하는 함수의 구현: v3

```
01 def get_total(lst) :  
02     total = 0  
03     for n in lst :  
04         total += n  
05  
06     return total
```

참고: 리스트의 총합 구하기

- ◆ 리스트의 총합을 구하는 일은 자주 사용됨
 - 내장 함수 `sum()` 제공
 - `total = sum(population)`
- ◆ 해결 가능한 간단한 작업을 직접 구현해보며 프로그래밍에 대한 개념과 문법에 대한 이해 도모

최종 프로그램 (전체 코드)

코드 10-21 인구조사 프로그램 (최종)

```
01 # 사용자 정의 함수부
02 # 코드 10-13
03 def input_num_of_population() :
04     nPeople = []
05     for f in range(4) :
06         n = int(input(f'{f + 1}층의 거주인원수는? '))
07         nPeople.append(n)
08
09     # 입력받은 거주인원수를 호출측으로 반환
10     return nPeople
11
12 # 코드 10-14
13 def show_num_of_population(p) :
14     cnt = len(p)
15     for i in range(cnt) :
16         print(f'{i + 1}층의 거주인원수는 {p[i]}명입니다.')
17
18 # 코드 10-20
19 def get_total(lst) :
20     total = 0
21     for n in lst :
22         total += n
23
24     return total
```

```
25
26 # 주 프로그램부 : 코드 10-7
27 # 1~4층의 거주인원수를 입력받음
28 population = input_num_of_population()
29
30 # 입력받은 1~4층의 거주인원수를 출력
31 show_num_of_population(population)
32
33 # total = 입력받은 1~4층의 거주인원수의 총합을 구함
34 total = get_total(population)
35
36 # 총 거주인원수(total) 출력
37 print(f'총 거주인원수는 {total}명입니다.')
```



04 | 이중 리스트

이중 리스트

- ◆ 리스트의 요소가 또다른 리스트로 중첩된 경우
 - 2개의 [] 첨자 연산으로 각 요소에 접근

코드 10-22 이중 리스트

```
01 scores = [  
02     [ '이찬수', 95, 85 ],  
03     [ '홍길동', 90, 80 ]  
04 ]  
05  
06 print(scores)  
07 print(scores[0])  
08 print(scores[0][0])
```

▶ 실행 결과

```
[['이찬수', 95, 85], ['홍길동', 90, 80]]  
['이찬수', 95, 85]  
이찬수
```

이중 리스트의 구조 및 요소 선택식

- ◆ 첫 번째 인덱스는 '그룹의 인덱스',
두 번째 인덱스는 '그룹 내 요소에 대한 인덱스'

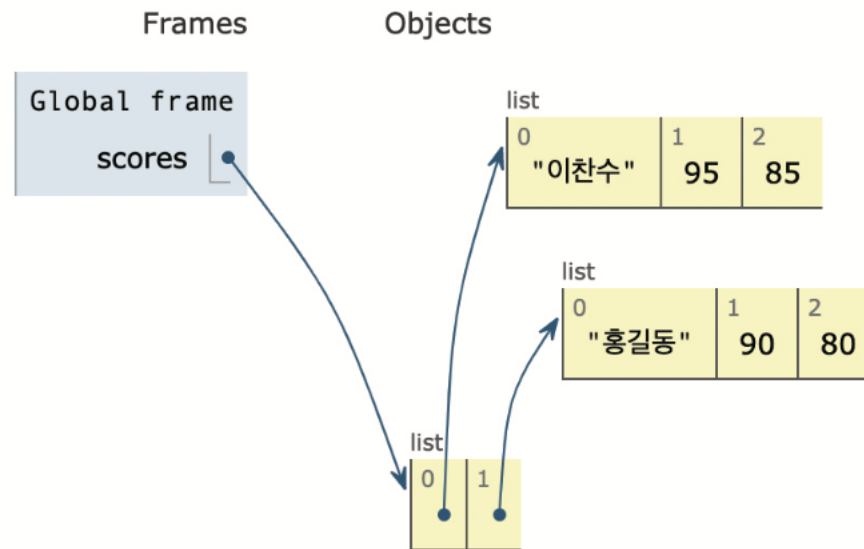


그림 10.10 파이썬 튜터에서 살펴본 이중 리스트의 구조

이중 리스트와 반복문

◆ 중첩된 반복문으로 각 요소에 순차적으로 접근

#

코드 10-23

이중 리스트의 반복적 접근

```
01 scores = [  
02     [ '이찬수', 95, 85 ],  
03     [ '홍길동', 90, 80 ]  
04 ]  
05  
06 for g in scores:  
07     for e in g:  
08         print(e, end='/')  
09     print()
```



실행 결과

```
이찬수/95/85/  
홍길동/90/80/
```

#

코드 10-24

이중 리스트의 반복적 접근

```
06 for gi in range(len(scores)):  
07     for ei in range(len(scores[gi])):  
08         print(scores[gi][ei], end='/')  
09     print()
```



05

딕셔너리와 반복문

딕셔너리와 반복문

- ◆ 딕셔너리는 키와 값의 순서쌍 형태로 데이터 저장하는 구조
 - 딕셔너리의 각 요소에 접근하기 위해서는 각 요소값과 연관된 키 값을 이용해야 함
 - 순서를 저장하지 않음
 - 반복문과 함께 루핑(looping)하기 위해 다음 메서드 제공

```
딕셔너리.keys()    # 딕셔너리 내의 키들의 목록을 반환  
딕셔너리.values()  # 딕셔너리 내의 값들의 목록을 반환  
딕셔너리.items()   # 딕셔너리 내의 키-값 쌍들의 목록을 반환
```

딕셔너리의 루핑

◆ 저장 순서는 유지되지 않음에 유의

코드 10-25 딕셔너리의 각 요소에 반복적으로 접근하기

```
01 d = {  
02     'python': '파이썬',  
03     'basic': '기초',  
04     'programming': '프로그래밍'  
05 }  
06  
07 for key in d.keys():  
08     print(key, d[key])
```

▶ 실행 결과

```
python 파이썬  
basic 기초  
programming 프로그래밍
```

코드 10-26 딕셔너리의 각 요소에 반복적으로 접근하기

```
07 for key in d:  
08     print(key, d[key])
```



06 | 딕셔너리와 함수

복합 자료형과 함수

- ◆ 복합 자료형을 함수의 인수로 사용하면
 - 매개변수로 전달된 아이디를 통해 인수 영역의 데이터 객체에 접근(수정) 가능
 - 가변 객체
- ◆ 함수 안에서 선언된 복합 자료형을 반환하면
 - 호출측으로 아이디가 전달되어 계속 사용 가능
 - 데이터 객체는 함수 바깥에서도 유효하게 사용 가능
 - 객체의 아이디를 저장하는 변수는 지역적인 특징을 가짐

예제: 딕셔너리를 안전하게 처리하는 함수

- ◆ 딕셔너리의 요소 접근 시 [] 연산 자주 사용
 - 유효하지 않은 키 값 사용시 오류 발생



- 궁극적으로는 예외 처리 메커니즘으로 처리



- 각 요소 접근시마다 키값의 유효성 검사
 - 사용자 정의 함수를 이용해 처리

딕셔너리의 요소 접근 함수

- ◆ 유효한 키인 경우에만 값을 반환하도록 작성
 - 매개변수
 - 딕셔너리(dic), 접근할 요소의 키(key)
 - 반환값
 - 유효한 키의 경우에는 그에 대한 요소값을 반환
 - 유효하지 않은 키의 경우에는 None 반환

#

코드 10-27

딕셔너리의 요소 접근 함수 구성의 예

```
01 def dict_get(dic, key) :  
02     if key in dic :  
03         return dic[key]  
04     else :  
05         return None
```

요소 접근 함수의 이용 예

코드 10-27 딕셔너리의 요소 접근 함수 구성의 예 (cont)

```
07 d = { 'python': '파이썬', 'basic': '기초', 'programming': '프로그래밍' }
08
09 res = dict_get(d, 'python')
10 if res != None :
11     print(res)
12 else :
13     print('오류: 잘못된 키')
14
15 res = dict_get(d, 0)
16 if res != None :
17     print(res)
18 else :
19     print('오류: 잘못된 키')
```

```
01 def dict_get(dic, key) :
02     if key in dic :
03         return dic[key]
04     else :
05         return None
```

실행 결과

파이썬
오류: 잘못된 키

딕셔너리에 요소 추가 함수

- ◆ 중복되지 않은 키인 경우에만 추가하도록 작성
 - 매개변수
 - 딕셔너리(dic), 추가할 데이터의 키(key)와 값(value)
 - 반환값
 - 추가 여부 참/거짓으로 반환

코드 10-28

```
01 def dict_append(dic, key, value) :  
02     if key in dic :  
03         return False  
04  
05     dic[key] = value  
06     return True
```

요소 추가 함수의 이용 예

코드 10-28 (cont)

```
08 d = { 'python': '파이썬', 'basic': '기초', 'programming': '프로그래밍' }
09
10 if dict_append(d, 'PYTHON', '파이썬') :
11     print('추가 성공')
12 else :
13     print('추가 실패')
14
15 if dict_append(d, 'basic', '베이직') :
16     print('두 번째 추가 성공')
17 else :
18     print('두 번째 추가 실패')
19
20 print(d)
```

```
01 def dict_append(dic, key, value) :
02     if key in dic :
03         return False
04
05     dic[key] = value
06     return True
```

▶ 실행 결과

추가 성공
두 번째 추가 실패
{'python': '파이썬', 'basic': '기초', 'programming': '프로그래밍',
'PYTHON': '파이썬'}

딕셔너리의 요소 삭제 함수

- ◆ 유효한 키인 경우에만 값을 삭제하도록 작성
 - 매개변수
 - 딕셔너리(dic), 삭제할 요소의 키(key)
 - 반환값
 - 삭제 처리 여부를 반환

#

코드 10-29

딕셔너리의 요소 삭제 함수 구성의 예

```
01 def dict_delete(dic, key) :  
02     if key in dic :  
03         del dic[key]  
04         return True  
05     else :  
06         return False
```

요소 삭제 함수의 이용 예

#

코드 10-29

딕셔너리의 요소 삭제 함수 구성의 예 (cont)

```
08 d = { 'python': '파이썬', 'basic': '기초', 'programming': '프로그래밍' }
09 if dict_delete(d, 'basic') :
10     print('삭제 성공')
11 else :
12     print('삭제 실패')
13
14 print(d)
```

```
01 def dict_delete(dic, key) :
02     if key in dic :
03         del dic[key]
04         return True
05     else :
06         return False
```



실행 결과

삭제 성공

```
{'python': '파이썬', 'programming': '프로그래밍'}
```

연습문제 10.2

◆ 사용자로부터 입력 받은 수 중에 가장 큰 수 찾기

- 다음 역할을 수행하는 find_max() 함수 정의
 - 임의의 길이를 갖는 리스트를 입력받아,
반복문과 리스트 인덱싱을 이용해 최대값을 찾아, 이 값을 반환
 - 파이썬의 내장 함수 max()와 동일하게 작동
- 5개의 정수를 입력받아, 위에서 정의한 함수를 이용해,
그 중 가장 큰 값을 찾는 프로그램 완성
- 실행 화면

정수 입력: **10**

정수 입력: **-5**

정수 입력: **7**

정수 입력: **4**

정수 입력: **205**

가장 큰 정수는 **205**입니다.

연습문제 10.3

◆ 연습문제 9.1을 학생 수를 유연하게 처리 가능하도록 수정

- 사용자로부터 학생들의 점수를 입력받아, 하나의 리스트에 저장하여 반환하는 함수 `input_scores()` 작성. 음수가 입력되면 반복을 마치며, 이 음수는 리스트에 저장 안함.
- 인수로 전달된 리스트에 저장된 점수들의 평균을 구해 반환하는 `get_average()` 작성
- 인수로 전달된 리스트의 각 학생의 점수를 출력하는 `show_scores()` 작성
- 위 함수들을 이용해 오른쪽 실행 결과처럼 나오는 주 프로그램부 완성.
평균은 소수점 아래 첫째 자리까지 출력

[점수 입력]

#1? 95

#2? 100

#3? 24

#4? 72

#5? 1

#6? -1

[점수 출력]

개인점수: 95 100 24 72 1

평균: 58.4

연습문제 10.4

- ◆ 연습문제 10.3 실행 후, 특정 점수를 받은 학생의 번호 확인
 - 하나의 리스트(lst)와 정수(n)을 매개변수로 전달받아, 그 리스트상에 n이 나타나는 인덱스를 반환하는 함수 search() 작성.
n이 나타나지 않으면 이 함수는 None 반환
 - 실행 화면
 - 구현을 간단히 하기 위해 동일 점수를 받은 학생은 없다 가정
 - search() 함수 대신 리스트의 메서드를 이용하는 방식으로도 구현
- [점수 입력]

#1? 95
#2? 82
#3? 100
#4? 90
#5? 97
#6? -1

[점수 출력]

개인점수: 95 82 100 90 97
평균: 92.8

[검색]

찾고자 하는 점수는? 100
100점은 3번 학생의 점수입니다.

[검색]

찾고자 하는 점수는? 0
0점을 받은 학생은 없습니다.

개념 확인 과제(연습문제 10.6)

- ◆ 연습 문제 9.4(쇼핑몰 장바구니)를 함수를 이용해 재구성
 - 호출시마다 인수로 전달된 딕셔너리 shopping_bag에 사용자로부터 입력받은 상품명과 수량을 저장하는 buy() 작성
 - 장바구니에 들어 있는 전체 상품을 출력하는 show() 작성
 - 장바구니에서 확인하고자 하는 상품을 입력받아 이 상품의 수량을 출력하는 find() 작성
(장바구니 안에 없으면 해당 상품이 없다고 출력)
 - 주 프로그램 부

```
shopping_bag = {}  
while True:  
    if buy(shopping_bag) == False:  
        break  
show(shopping_bag)  
find(shopping_bag)
```

개념 확인 과제(연습문제 10.6)

◆ 연습 문제 9.4(쇼핑몰 장바구니)를 함수를 이용해 재구성

- 실행 결과 (상품명에 엔터 치면 구입 종료)

[구입]

상품명? 샌드위치

수량은? 10

장바구니에 샌드위치 10개가 담겼습니다.

[구입]

상품명? 주스

수량은? 5

장바구니에 주스 5개가 담겼습니다.

[구입]

상품명?

>>> 장바구니 보기: {'샌드위치': 10, '주스': 5}

[검색]

장바구니에서 확인하고자 하는 상품은? 과자

장바구니에 과자는 (는) 없습니다.

◆ 제출 기한 및 방법

- 다음 수업시간 전까지
본인의 repository에
hw8.py를 업로드

[검색]

장바구니에서 확인하고자 하는 상품은? 샌드위치
샌드위치 (는) 10개 담겨 있습니다.