

# 파이썬 프로그래밍

9주차 강의자료

나재호

CHAPTER  
**09**

# 복합 자료형 소개

- 01** 복합 자료형이란?
- 02** 리스트
- 03** 리스트의 동작
- 04** 튜플
- 05** 딕셔너리

## 이 단원을 마치고 나면

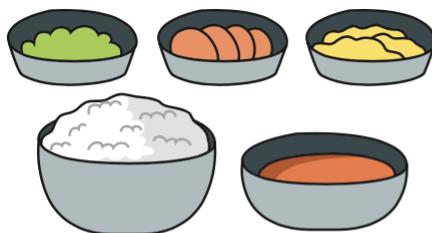
- 다수의 데이터를 한 덩어리로 묶어 저장하는 복합 자료형의 개념과 특징을 알 수 있다.
- 리스트를 선언하고 각종 연산자와 메서드를 이용해 데이터를 조작할 수 있다.
- 튜플을 선언하고 각종 연산자와 메서드를 이용해 데이터를 참조할 수 있다.
- 딕셔너리를 선언하고 각종 연산자와 메서드를 이용해 데이터를 조작할 수 있다.



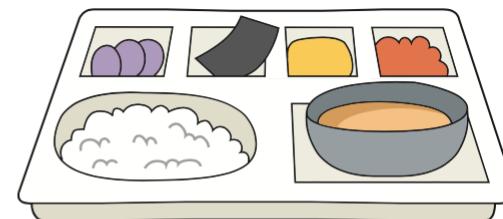
# 01 | 복합 자료형이란

# 복합 자료형(compound data type)

- ◆ 여러 데이터를 한 덩어리로 모아서 유지
  - 하나의 이름으로 다수의 데이터에 접근 가능
  - 연관된 데이터들을 일관된 방법으로 저장하고 접근
- ◆ 왜 모으나?
  - 좀 더 간단 명료하고 일관되게 접근 가능
  - 반복문과 결합하여 반복적인 처리 용이



(1) 일반 그릇



(2) 식판

그림 9.1 단순 변수와 복합 자료형 변수의 비유



## 02 | 리스트

# 일상에서의 리스트 (List)

## ◆ 순서대로 나열된 연관된 정보들의 목록

- 주소록
- 작업 목록
- 장비 목록
- 탑승자 목록 등



그림 9.4 각 학과 별 탑승자 목록

# 리스트 사용시 이점

- ◆ 연관된 것끼리 묶어 하나의 이름으로 관리할 수 있음
  - 예: 주스들, 슬러시들
- ◆ 개별 항목을 식별하지 않더라도 순번으로 접근 가능
  - 예: 왼쪽 첫 번째 주스

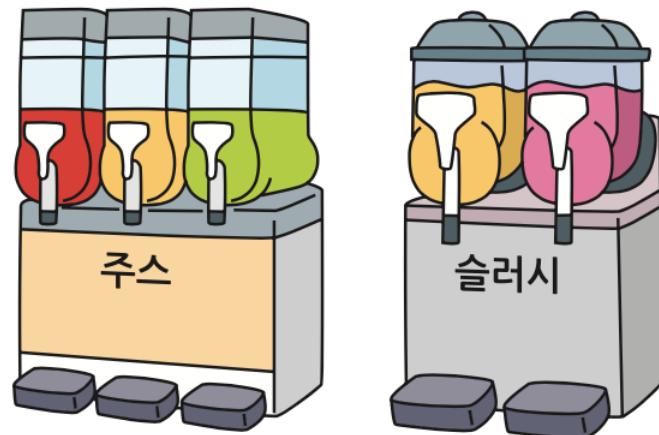
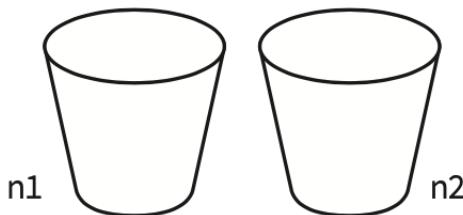


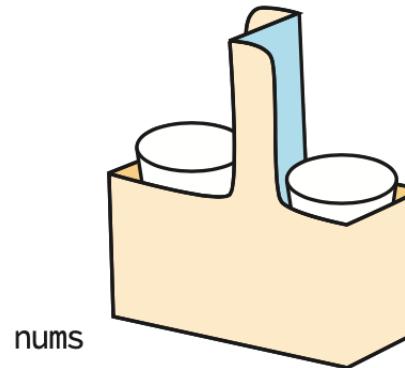
그림 9.3 음료수 디스펜서

# 프로그래밍에서의 리스트

- ◆ 여러 데이터를 **하나로 묶어 순번에 따라 저장**
  - 연관된 다수의 데이터를 하나의 이름으로 접근
  - 저장 순서를 유지
  - 순번에 따라 각 데이터에 접근
  - 반복문과 함께 사용하여 효율적으로 데이터에 접근



(1) 개별적인 두 변수 *n1*과 *n2*



(2) 두 요소를 갖는 하나의 리스트 *nums*

그림 9.2 단순 변수와 복합 자료형 변수의 비유

# 파이썬의 리스트

---

- ◆ 다수의 요소들을 순번에 따라 한 덩어리로 묶은 자료구조
  - 요소(element)
    - 리스트에 포함된 개별 데이터
  - 문자열과 비슷하게 리스트 자체적으로 각종 기능을 제공하는 메서드를 가짐
    - 프로그래밍시 편리하고 효율적으로 리스트에 대해 다양한 작업 처리 가능

# 리스트의 생성과 초기화

- ◆ 변수와 마찬가지로 리스트를 사용하기 위해서는 먼저 초기값 할당을 통해 선언해야 함
- ◆ 선언시 **[ ]** 안에 0개 이상의 요소들을 순서대로 콤마로 구분하여 나열

```
리스트_이름 = [값1, 값2, ..., 값n]      # n개의 요소를 갖는 리스트 선언문
```

- 각 요소값들을 초기값으로 갖는 리스트 생성

# 같은 자료형의 요소를 갖는 리스트

- ◆ 저장 순서는 임의로 변경되지 않음
  - 빈 리스트 생성 가능

# 코드 9-1

기본적인 리스트 선언 예

```
01 list1 = [11, 22, 33, 44, 55]
02 list2 = ['단출하게', '단아하게', '당당하게']
03 list3 = []
04 print(list1)
05 print(list2)
06 print(list3)
```



실행 결과

```
[11, 22, 33, 44, 55]
['단출하게', '단아하게', '당당하게']
[]
```

# 다른 자료형의 요소를 갖는 리스트

- ◆ 모든 요소의 자료형이 일치할 필요 없음
  - 요소로 또다른 리스트도 포함 가능

# 코드 9-2

## 보다 복잡한 구성의 리스트 선언

```
01 list4 = [11, '홍길동', 19, 180.5]      # 번호 / 이름 / 나이 / 키  
02 list5 = [11, '홍길동', [10, 10, 92, 100]] # 번호 / 이름 / 점수들  
03 print(list4)  
04 print(list5)
```



### 실행 결과

```
[11, '홍길동', 19, 180.5]  
[11, '홍길동', [10, 10, 92, 100]]
```

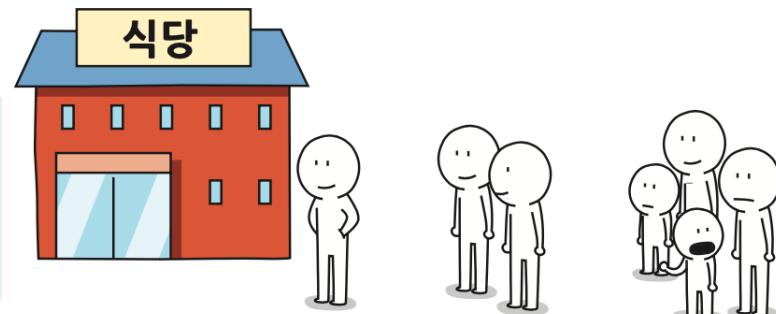


그림 9.5 식당 대기줄

# 리스트의 내부 구조

## ◆ 복합 자료형인 리스트는 가변 객체

- 여러 칸을 가지고 있는 그릇이되,  
그 칸의 개수가 가변적일 수 있는 그릇에 비유
  - 리스트 변수에 저장된 데이터 객체에 대한 아이디 변경 가능
  - 리스트 변수가 가리키는 리스트 객체 자신도 수정 가능

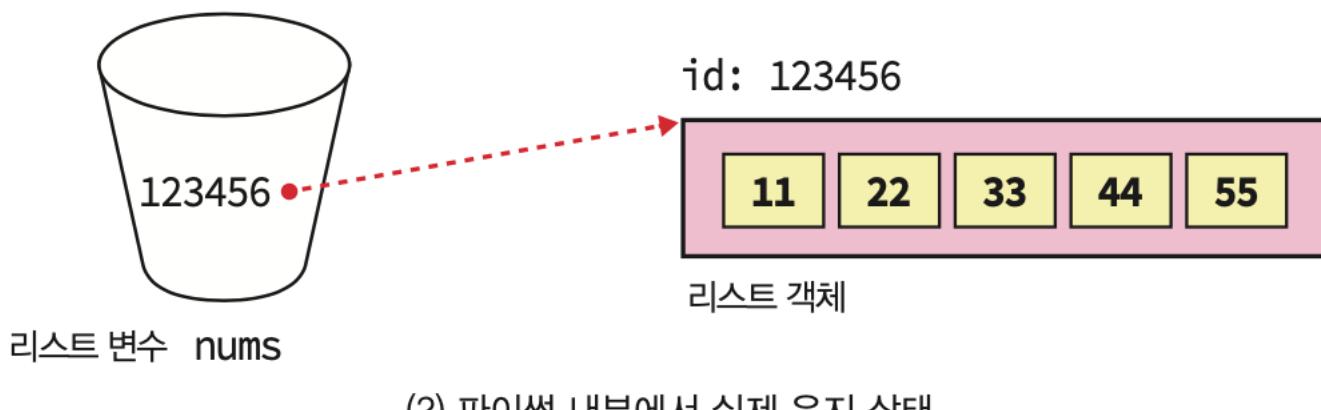


그림 9.6 리스트의 내부 구조

# 파이썬 컴퓨터 사이트

## ◆ <https://pythontutor.com>

- 파이썬 프로그램 실행 시 현재 실행 문장과 그에 따른 각 변수의 변화를 가시적으로 보여줌

The screenshot shows a web browser window for 'Python Tutor: Learn Python, Java' at 'pythontutor.com'. The main heading is 'Learn Python, JavaScript, C, C++, and Java'. Below it, a sub-headline says 'This tool helps you learn Python, JavaScript, C, C++, and Java programming by [visualizing code execution](#). You can use it to debug your homework assignments and as a supplement to online coding tutorials.' A call-to-action button 'Start coding now in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)' has the 'Python' link highlighted with a red dashed box. Below this, a statistic 'Over 15 million people in more than 180 countries' have used the tool. On the left, a code editor shows Python 3.6 code for 'listSum' with a cursor at line 5. On the right, a diagram titled 'Frames' and 'Objects' shows the execution environment. It includes a 'Global frame' with variables 'listSum' and 'myList', and three nested frames for the recursive calls, each containing a 'tuple' object with two elements: (0, 1), (0, 1), and (0, 1, None).

```
1 def listSum(numbers):
2     if not numbers:
3         return 0
4     else:
5         (f, rest) = numbers
```

Frames Objects

Global frame

listSum myList

function listSum(numbers)

tuple tuple tuple

0 1 0 1 0 1 None

# 파이썬 컴퓨터로 리스트 내부 구조 분석

## # 코드 9-3 변수와 리스트의 비교

```
01 n = 10  
02  
03 nums = [11, 22, 33]
```

Write code in Python 3.6

```
1 n = 10  
2  
3 nums = [11, 22, 33]|
```

Visualize Execution      Live Programming Mode

hide exited frames [default]   inline primitives, don't nest objects [default]   draw pointers as arrows [default]

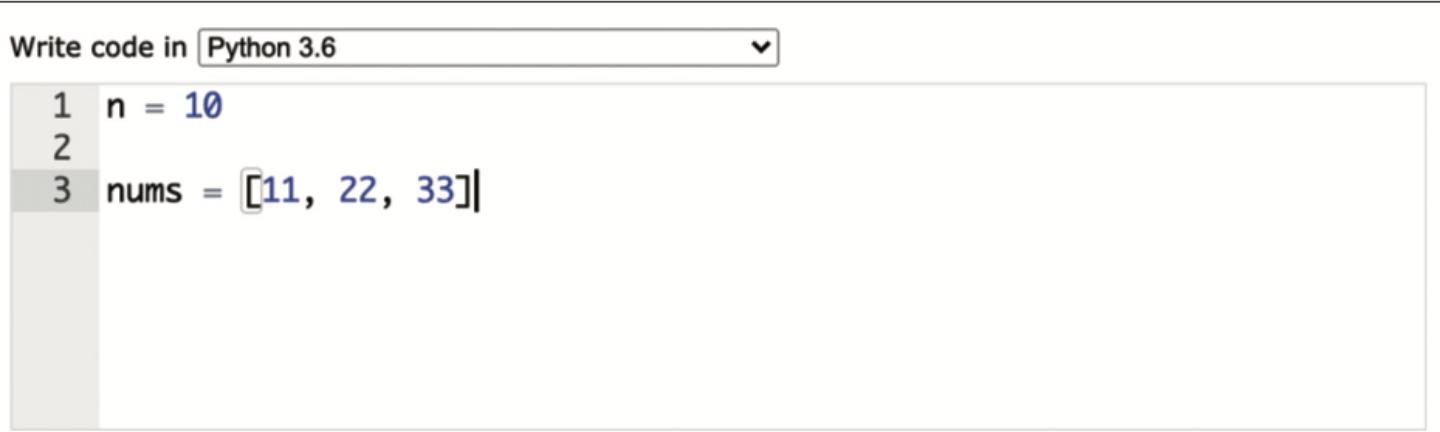


그림 9.8 파이썬튜터 사이트에서 코드 9-3 작성

# 가변 객체인 리스트의 구성

- ◆ 변수는 데이터 객체에 대한 아이디를 가짐
  - 참조값(객체의 아이디)
  - 요소값들(객체의 내용)
    - 불변 객체와 달리 가변 객체는 요소값이 변경될 수 있음



그림 9.11 파이썬 튜터 사이트에서의 [시작화된 실행]: 두 번째 문장 실행 후

# 리스트의 요소 인덱스

- ◆ 리스트의 각 요소는 순번에 따라 식별됨
  - 리스트에 요소 할당시 인덱스 0부터 양의 정수로 배정됨에 유의
  - 음의 정수 인덱스로는 리스트의 끝요소부터 식별하는데 사용 가능

```
list1 = [11, 22, 33, 44, 55]
```

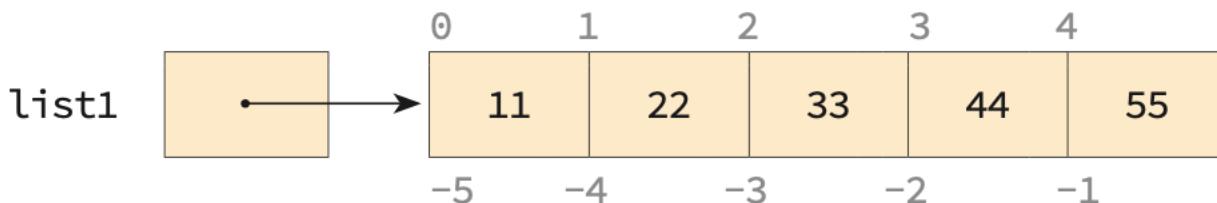


그림 9.12 리스트의 구성

# 리스트 요소의 선택: 인덱싱

- ◆ 유효한 순번(인덱스)에 따라 각 요소 데이터를 개별적으로 선택해 접근하는 연산

리스트이름[접근할\_요소의\_인덱스] # 해당 인덱스의 요소값을 결과로 가짐

- 일반 단순 변수 처럼 조작 가능
  - L-값으로 사용 가능

# 코드 9-4

리스트 인덱싱

```
01 list4 = [11, '홍길동', 19, 180.5]      # 번호 / 이름 / 나이 / 키  
02 print(list4[1], '님의 키는', list4[-1], 'cm입니다.')
```

▶ 실행 결과

홍길동 님의 키는 180.5 cm입니다.

# 주의: 리스트 선언과 인덱싱

---

- ◆ 리스트 선언문에서 [ ] 안에는
  - 각 요소의 초기값들을 나열
  - `names = [ '이찬수', '홍길동', '일지매' ]`
  
- ◆ 리스트 인덱싱에서 [ ] 안에는
  - 접근할 요소의 인덱스를 기술
  - `gildong = names[1]`

# 리스트 슬라이싱

- ◆ 범위를 이용해 리스트를 구성하는 부분 요소들에 대한 리스트를 결과로 얻는 연산

```
리스트이름[시작인덱스:끝인덱스] # 시작인덱스 ~ (끝인덱스 - 1) 사이의  
# 부분 리스트를 결과값으로 가짐
```

- '시작인덱스'의 요소부터 '끝인덱스 - 1'까지의 요소를 갖는 부분 리스트를 결과값으로 가짐
  - 시작인덱스 생략시 0번부터 슬라이싱
  - 끝인덱스의 표현은 생략시 마지막까지 슬라이싱

# 리스트 슬라이싱의 예

# 코드 9-5

## 리스트 슬라이싱

```
01 list1 = [11, 22, 33, 44, 55]  
02 sub_list = list1[1:3]  
03 print(sub_list)          # [22, 33]이 출력  
04 print(sub_list[0])       # 22가 출력
```



실행 결과

```
[22, 33]  
22
```

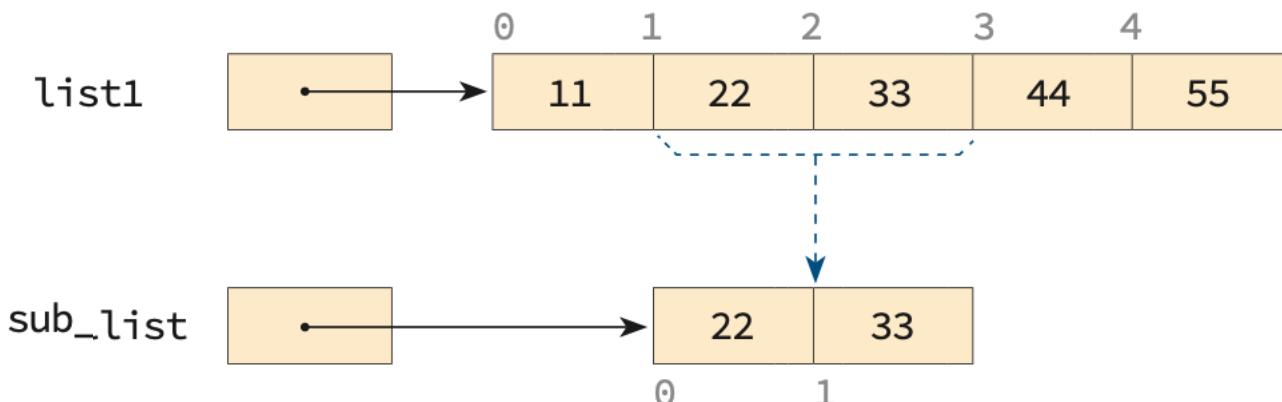


그림 9.13 리스트 슬라이싱의 예

# 리스트의 요소 수정

- ◆ 가변 객체인 리스트는 그 자신이 갖고 있는 요소 데이터 변경이 가능

`리스트이름[수정할요소인덱스] = 교체데이터`

# 특정 한 요소를 대체

`리스트이름[시작인덱스: 끝인덱스+1] = [교체데이터들]` # 지정된 범위의 요소들을 대체

- 리스트 인덱싱 이용
  - 한 요소의 값을 대체하는 효과를 가짐
- 리스트 슬라이싱 이용
  - 지정한 범위의 요소들을 한 번에 대체하는 효과를 가짐

# 리스트 요소 수정 예

## ◆ 인덱싱/슬라이싱 연산으로 수정할 요소 선택

#

코드 9-6

리스트 요소 수정

```
01  nums = [11, 22, 33, 44, 55]
02
03  nums[0] = -1
04  print(nums)    # [-1, 22, 33, 44, 55]
05
06  nums[1:3] = [100, 200, 300]
07  print(nums)    # [-1, 100, 200, 300, 44, 55]
```



실행 결과

[-1, 22, 33, 44, 55]

[-1, 100, 200, 300, 44, 55]

# 리스트 요소 수정시 주의

- ◆ 리스트 인덱싱을 이용해 특정 요소를 다른 리스트로 대체시 원하지 않는 결과 초래 가능
  - 한 요소는 그에 대한 하나의 데이터로 대체하자!
    - 인덱싱으로는  $1 : 1$  대체

# 코드 9-7

## 리스트 요소 수정시 주의사항

```
01  nums = [11, 22, 33, 44, 55]  
02  
03  nums[1] = [100, 200, 300]  
04  print(nums)
```



### 실행 결과

```
[11, [100, 200, 300], 33, 44, 55]
```

# 리스트 요소 수정시 주의

- ◆ 한 요소를 여러 요소로 대체하고자 하는 경우  
리스트 슬라이싱 이용
  - 한 요소만을 갖는 부분 리스트를 다른 리스트로 대체
    - 슬라이싱으로는 1 : 다 대체

#

코드 9-8

한 요소를 다수의 요소로 대체할 때의 주의사항

```
01 nums = [11, 22, 33, 44, 55]  
02  
03 nums[1:2] = [100, 200, 300]  
04 print(nums)
```



실행 결과

```
[11, 100, 200, 300, 33, 44, 55]
```

# 리스트의 대입 연산

- ◆ 모든 변수는 데이터 객체에 대한 아이디를 가짐
  - 대입 연산으로 변수가 갖는 아이디가 복사됨

#

코드 9-9

리스트 대입 연산(얕은 복사)

```
01  nums1 = [11, 22, 33]
02  nums2 = []
03
04  nums2 = nums1      # 리스트의 대입 연산은 어떤 특징이 있나?
05
06  nums2[0] = 0
07  print(nums1)      # 예상과 같은 결과가 출력되는가?
08  print(nums2)
```



실행 결과

```
[0, 22, 33]
[0, 22, 33]
```

# 얕은 복사(shallow copy)

- ◆ 대입 연산은 두 변수가 같은 객체에 대한 아이디를 갖게 함
  - 즉, 엄밀히 말해 리스트가 복사되었다고 할 수 없음!

Python 3.6  
(known limitations)

```
1 nums1 = [11, 22, 33]
2 nums2 = []
3
4 nums2 = nums1
5
6 nums2[0] = 0
7 print(nums1)
8 print(nums2)
```

[Edit this code](#)

line that just executed  
next line to execute

<< First < Prev Next > >> Step 4 of 6

The screenshot shows the Python Tutor interface. On the left, there's a code editor with the above Python code. On the right, there's a visualization of the state. A 'Frames' panel shows two frames: 'Global frame' containing variables 'nums1' and 'nums2'. An 'Objects' panel shows a single list object with elements 0, 11, 22, and 33. Arrows from both 'nums1' and 'nums2' in the Global frame point to the same list object in the Objects panel, illustrating that they both reference the same memory location.

Print output (drag lower right corner to resize)

Frames Objects

Global frame

nums1

nums2

list

0	11	22	33
---	----	----	----

(2) 리스트의 복사 : 복사 전/후 상태 비교

# 깊은 복사(deep copy)

- ◆ 기존의 리스트와 똑같은 요소들을 갖는,  
별도의 복사본 리스트를 생성
  - 단순 할당 대신 리스트 슬라이싱 이용

# 코드 9-10

리스트 대입 연산(깊은 복사)

```
01 nums1 = [11, 22, 33]
02 nums2 = []
03
04 nums2 = nums1[:]
05
06 nums2[0] = 0
07 print(nums1)
08 print(nums2)
```



실행 결과

```
[11, 22, 33]
[0, 22, 33]
```

# 깊은 복사된 리스트

- ◆ 리스트에 대한 변수값이 아니라 변수가 가리키는 리스트 객체의 복사본 생성

Python 3.6  
(known limitations)

```
1 nums1 = [11, 22, 33]
2 nums2 = []
3
4 nums2 = nums1[:]
5
6 nums2[0] = 0
7 print(nums1)
8 print(nums2)
```

[Edit this code](#)

▶ line that just executed  
→ next line to execute

<< First < Prev Next > Last >>  
Step 4 of 6

Print output (drag lower right corner to resize)

Frames Objects

Global frame

list	0	1	2	3	33
nums1	11	22			
nums2	11	22			

list

0	1	2	3	33
11	22			

그림 9.15 리스트의 깊은 복사

# 리스트의 비교 연산

- ◆ 단지 아이디 비교가 아닌 리스트에 속한 각 요소 값들을 순서대로 비교

연산자 기호	리스트 연산에서의 의미
<code>==</code>	두 리스트에 포함된 각 요소들이 순서에 따라 모두 같은가? (일치)
<code>!=</code>	두 리스트에 포함된 각 요소들이 순서에 따라 같지 않은 경우가 하나라도 있는가? (불일치)
<code>&gt;</code>	두 리스트를 각 요소 순서에 따라 비교하되, 첫 번째로 다른 요소의 왼쪽 리스트 요소 값이 오른쪽 리스트 요소보다 더 큰가? (초과)
<code>&gt;=</code>	두 리스트를 각 요소 순서에 따라 비교하되, 첫 번째로 다른 요소의 왼쪽 리스트 요소 값이 오른쪽 리스트 요소보다 더 크거나 같은가? (이상)
<code>&lt;</code>	두 리스트를 각 요소 순서에 따라 비교하되, 첫 번째로 다른 요소의 왼쪽 리스트 요소 값이 오른쪽 리스트 요소보다 더 작은가? (미만)
<code>&lt;=</code>	두 리스트를 각 요소 순서에 따라 비교하되, 첫 번째로 다른 요소의 왼쪽 리스트 요소 값이 오른쪽 리스트 요소보다 더 작거나 같은가? (이하)

# 리스트 비교 연산의 예

- ◆ 리스트 앞쪽 요소에 비교 우선순위가 부여됨

#

코드 9-11

리스트의 비교 연산의 예

```
01 nums1 = [11, 22, 33]
02
03 nums2 = nums1[:]
04 print(nums1 == nums2)      # True
05
06 nums2[1] = 0              # [11, 0, 33]
07 print(nums1 == nums2)      # False
08 print(nums1 < nums2)       # False
09 print(nums1 > nums2)       # True
```

# 리스트의 연결 연산

```
리스트1 + 리스트2      # 두 리스트를 연결한 하나의 새로운 리스트를 결과로 가짐  
리스트 * 정수          # 리스트를 정수 번 반복한 하나의 새로운 리스트를 결과로 가짐
```

- ◆ 두 리스트에 대해 + 연산시
  - 두 리스트를 순서대로 하나로 연결한 리스트를 결과값으로 가짐
- ◆ 한 리스트에 정수를 \* 연산시
  - 대상 리스트의 요소들을 정수 번 반복한 리스트를 결과값으로 가짐

# 리스트의 연결 연산의 예

#

코드 9-12

리스트의 연결 연산

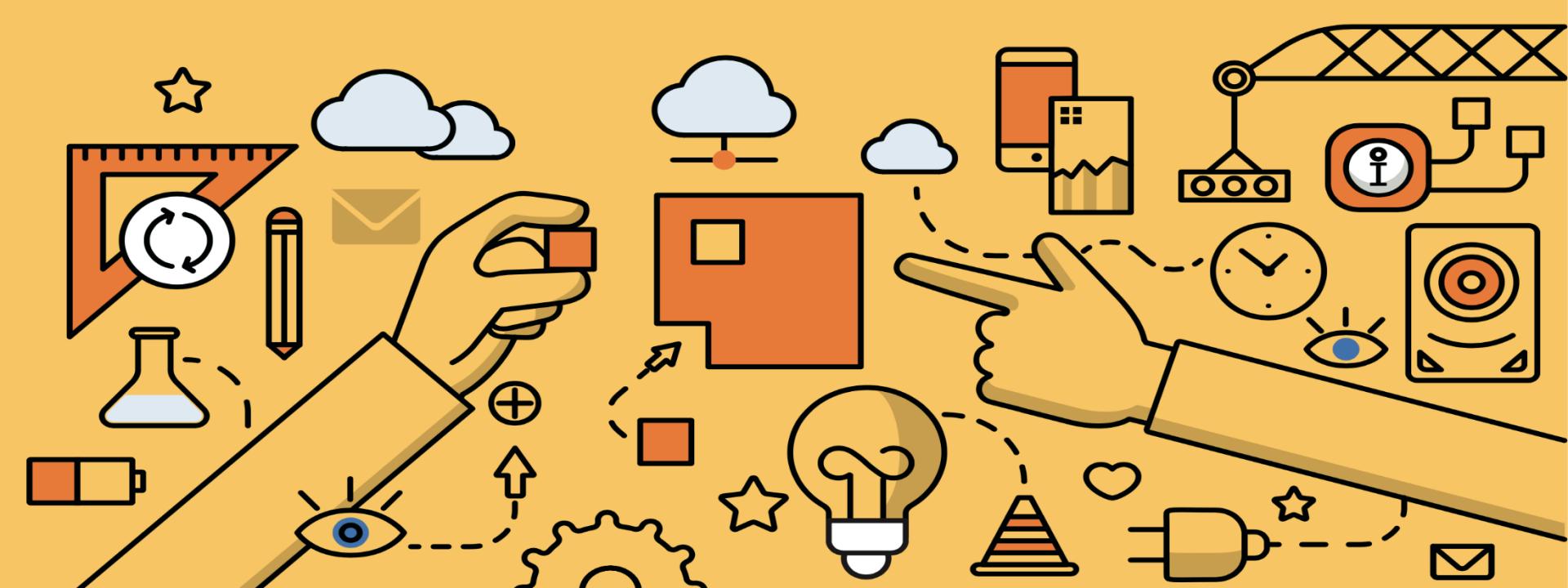
```
01  nums1 = [11, 22, 33]
02  nums2 = [44, 55]
03
04  nums3 = nums1 + nums2      # 리스트의 연결 연산
05  print(nums3)
06
07  nums4 = nums1 * 2          # 리스트의 반복 연산
08  print(nums4)
```



실행 결과

```
[11, 22, 33, 44, 55]
```

```
[11, 22, 33, 11, 22, 33]
```



## 03 | **리스트의 동작**

# 리스트의 길이

## ◆ 리스트에 포함된 요소 개수 확인

`len(리스트이름)` # 인수로 지정한 리스트의 요소 개수를 반환

# 코드 9-13

리스트 길이(요소 개수) 확인

```
01 nums = [11, 22, 33, 44, 55]  
02  
03 n = len(nums)      # 리스트 nums의 요소 개수가 저장됨  
04 print(n)
```



실행 결과

5

# 리스트에 요소 추가/삽입

- ◆ 리스트는 가변 객체이므로 대상 리스트의 요소를 직접 변경함!
  - 리스트 객체가 제공하는 메서드 이용

```
리스트이름.append(값)
```

# 리스트의 마지막 요소에 값 추가

- 인수로 지정한 데이터를 리스트 마지막 요소로 추가

```
리스트이름.insert(삽입위치, 값)
```

# 리스트의 삽입위치에 값을 끼워넣음

- 리스트의 삽입위치에 요소를 끼워 넣음
- 삽입위치에 있는 기존 요소와 그 이후 요소들은 한 칸씩 뒤로 밀림

# 리스트 요소 추가 예

# 코드 9-14

## 리스트에 요소 추가

```
01 nums = [11, 22, 33]
02
03 nums.append(55)          # nums의 맨 마지막 요소로 55 추가
04 print(nums)
05
06 nums.insert(3, 44)       # nums의 3번 요소 자리에 44 삽입
07 print(nums)
08
09 nums.insert(-1, 50)      # nums의 마지막 요소 자리에 50 삽입
10 print(nums)
```



실행 결과

[11, 22, 33, 55]

[11, 22, 33, 44, 55]

[11, 22, 33, 44, 50, 55]

# 리스트 요소의 삭제

**del** 리스트이름[삭제할\_요소\_인덱스]

**del** 리스트이름[삭제할\_요소의\_시작\_인덱스 : 삭제할\_요소의\_마지막\_다음\_인덱스]

- 리스트 인덱싱이나 슬라이싱을 이용해 삭제할 요소의 선택식 지정

리스트이름.*remove*(삭제할\_값)

- 리스트에서 요소값과 일치하는 첫 번째 요소 삭제

# 리스트 요소 삭제 예

# 코드 9-15

## 리스트 요소 삭제

```
01 nums = [11, 22, 33, 0, 11, 22, 33]
02
03 del nums[1]          # nums의 1번 요소인 22 삭제
04 print(nums)
05
06 del nums[2:4]        # nums의 2번부터 4번 전까지의 요소 [0, 11] 삭제
07 print(nums)
08
09 nums.remove(33)      # nums의 시작에서 처음 만나는 33 삭제
10 print(nums)
```



실행 결과

```
[11, 33, 0, 11, 22, 33]
[11, 33, 22, 33]
[11, 22, 33]
```

# 리스트 슬라이싱과 메서드로 삭제하기

```
리스트이름[시작인덱스:끝인덱스] = []      # 시작인덱스~(끝인덱스-1) 요소 삭제
```

- '시작인덱스'의 요소부터 '끝인덱스 - 1'까지의 요소를 공백 리스트로 대체
- `리스트이름[:]` = []
  - 전체 요소 삭제 효과

```
리스트이름.clear()                      # 모든 요소 삭제
```

- 대상 리스트의 전체 요소를 삭제

# 여러 요소의 삭제 예

#

코드 9-16

리스트의 여러 요소 삭제

```
01  nums = [11, 22, 33, 44, 55]
02
03  nums[1:4] = []      # nums의 1번~3번 요소 삭제
04  print(nums)        # [11, 55] 출력
05
06  nums.clear()       # nums의 모든 요소 삭제
07  print(nums)        # [] 출력
```



실행 결과

```
[11, 55]
[]
```

# 리스트의 위치 반환과 요소 개수 세기

## ◆ 위치 반환

```
리스트.index(값)    # 리스트 내에 인수값과 일치하는 첫 번째 요소의 인덱스 반환
```

- 리스트 상에 존재하는 특정 요소의 인덱스 확인

## ◆ 값이 일치하는 요소 개수 세기

```
리스트.count(값)    # 리스트 내에 인수값과 일치하는 요소의 개수 반환
```

- 일치하는 요소가 없는 경우 0 반환

# 리스트 요소 위치 및 개수 확인 예

#

코드 9-17

리스트 요소 위치 및 개수 확인

```
01  nums = [11, 22, 33, 44, 33]
02
03  print(nums.index(33))      # 첨자 2 출력
04  print(nums.count(33))      # 개수 2 출력
```



실행 결과

2

2

# 존재하지 않는 요소의 검색

- ◆ `index()` 메서드의 인수로 전달된 값이 리스트 내에 존재하지 않는 경우 `ValueError` 오류 발생



실행 결과

```
>>> nums = [11, 22, 33, 44, 33]
>>> nums.index(55)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    nums.index(55)
ValueError: 55 is not in list
```

- `index()` 메서드 호출 전에 조건문을 통해 검색값이 리스트 내에 존재하는지 먼저 확인 필요
  - 궁극적으로는 예외처리를 이용

# 리스트 내용 검사

## ◆ 리스트 내에 특정 데이터가 포함되는지 확인

값 `in` 리스트

# 리스트 상에 값이 존재하는지 참/거짓으로 판별

값 `not in` 리스트

# 리스트 상에 값이 존재하지 않는지를 참/거짓으로 판별

- `count()` 메서드의 결과값으로 유추할 수도 있음
- if 값 `in` 리스트: # 이 경우에만 `index()` 메소드 호출

# 코드 9-18

리스트 내에 특정 값 포함 여부 확인

```
01 nums = [11, 22, 33, 44, 55]  
02 print(22 in nums)          # nums의 요소에 22가 존재하는가?  
03 print(66 not in nums)      # nums의 요소에 66이 존재하지 않는가?
```



실행 결과

True  
True

# 리스트의 다양한 메서드들

---

- ◆ 이외에도 리스트 객체는 다양한 메서드를 가짐
  - 리스트에 대해 다양한 처리를 손쉽게 작성 가능
  - 파이썬 공식 사이트 문서 참고
    - <https://docs.python.org/ko/3/tutorial/datastructures.html>
- ◆ 다만, 이 강의에서는 프로그래밍에 대한 기본 개념을 익히는 데 초점을 둠
  - 오히려 메서드로 제공하는 기능들을 직접 구현해 보자



## 04 | 튜플

# 튜플(Tuple)

- ◆ 리스트와 마찬가지로 순서에 따라 데이터를 저장하는 복합 자료형
  - 요소 인덱스는 리스트와 마찬가지로 0부터 시작
- ◆ 다만 리스트와 달리 한 번 정의된 이후에는 그 내용을 바꿀 수 없음
  - 튜플의 요소값을 **수정/삭제/추가 불가**
  - 리스트에 비해 상대적으로 빠르게 동작

# 튜플의 생성

- ◆ 리스트와 마찬가지로 초기값 할당을 통해 먼저 선언한 후 사용
  - () 안에 1개 이상의 값들을 콤마로 구분하여 나열

```
튜플이름 = (값1, 값2, ..., 값n)      # n의 요소를 갖는 튜플의 선언  
튜플이름 = 값1, 값2, ..., 값n
```

- 간단한 생성을 위해 초기값을 감싸는 괄호는 생략 가능

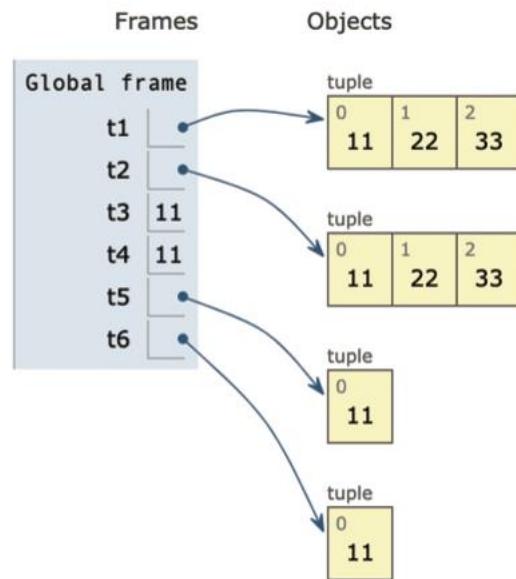
# 단일 요소만 갖는 튜플 선언 시 유의점

#

코드 9-19

튜플의 선언

```
01 t1 = (11, 22, 33)
02 t2 = 11, 22, 33
03
04 t3 = (11)
05 t4 = 11
06
07 t5 = (11,)
08 t6 = 11,
```



- 단일 요소만 갖는 경우 요소값 다음에 콤마 기술 필요

# 튜플의 사용

- ◆ 인덱싱, 슬라이싱, 연결 연산 등 리스트와 같은 방식으로 접근 가능
  - 단, 요소값 추가, 수정, 삭제 불가

#	코드 9-20	튜플의 사용	실행 결과
01	t1 = (11, 22, 33, 44, 55)		11
02	print(t1[0])		(22, 33)
03	print(t1[1:3])		(11, 22, 33, 44, 55, 66, 77)
04			(66, 77, 66, 77, 66, 77)
05	t2 = (66, 77)		
06	t3 = t1 + t2      # (11, 22, 33, 44, 55)와 (66, 77)을 연결		
07	print(t3)		
08			
09	t4 = t2 * 3      # (66, 77)을 3번 반복 연결		
10	print(t4)		

# 튜플은 요소 수정 불가



## 실행 결과

요소  
추가  
불가

```
>>> t1 = 10,  
>>> t1.append(20)  
Traceback (most recent call last):  
  File "<pyshell#1>", line 1, in <module>  
    t1.append(20)  
AttributeError: 'tuple' object has no attribute 'append'
```

요소  
수정  
불가

```
>>> t1[0] = 20  
Traceback (most recent call last):  
  File "<pyshell#2>", line 1, in <module>  
    t1[0] = 20  
TypeError: 'tuple' object does not support item assignment
```

# 튜플은 요소 삭제 불가



## 실행 결과

요소  
삭제  
불가

```
>>> del t1[0]
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    del t1[0]
TypeError: 'tuple' object doesn't support item deletion
```

튜플  
삭제  
가능

```
>>> del t1
>>> print(t1)
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    t1
NameError: name 't1' is not defined
```

- 튜플 자체는 삭제 가능



## 05 | 드셔너리

# 딕셔너리 (Dictionary)

- ◆ 키(key)와 값(value)의 순서쌍 형태로 데이터를 저장하는 복합자료형
  - 저장 순서를 유지하지 않음
    - 전체 데이터 검색없이 좀 더 빠르게 데이터에 접근 가능
  - 순번 대신 키 값에 의해 인덱싱 되어 해당 데이터의 값에 접근

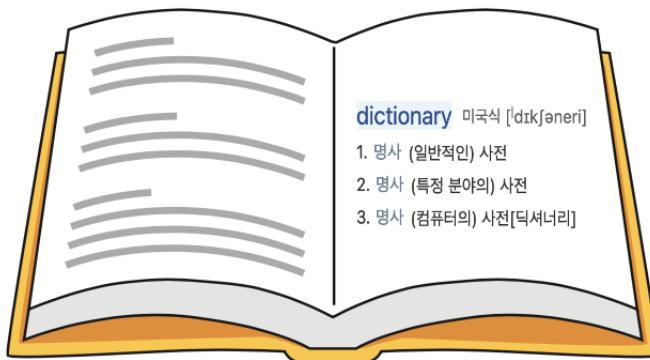


그림 9.18 대표적인 딕셔너리 구조를 갖는 사전

# 딕셔너리의 생성

- ◆ 선언 시 {} 안에 0개 이상의 키:값의 쌍들을 콤마로 구분하여 나열

```
딕셔너리이름 = {키1:값1, 키2:값2, ..., 키n:값n} # n개 요소를 갖는 딕셔너리
```

- 키
  - 다양한 자료형 가능
  - 딕셔너리 내의 각 데이터에 대한 식별자
    - 한 딕셔너리 내에서는 중복을 허용하지 않음
    - 중복된 키 값 사용시 '임의의' 하나를 제외한 나머지는 무시됨

## # 코드 9-23

잘못된 딕셔너리 선언의 예: 중복된 키값

```
01 err = { 'python':'파이썬', 'python':'파이선~' }
```

# 딕셔너리의 예

# 코드 9-22

딕셔너리의 선언 예

```
01 d = { 'python':'파이썬', 'basic':'기초', 'programming':'프로그래밍' }  
02 print(d)
```



실행 결과

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

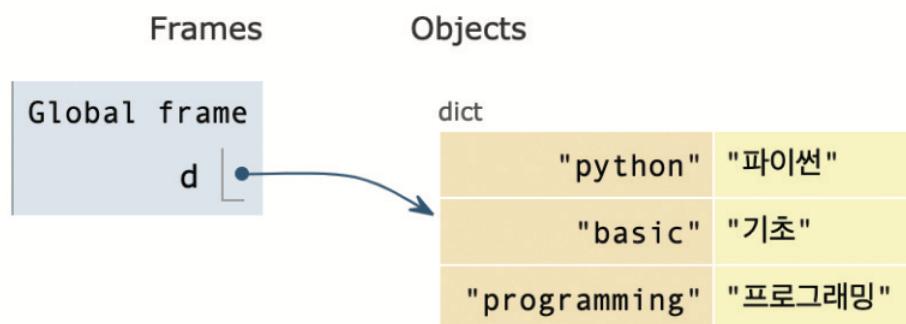


그림 9.19 딕셔너리의 선언

# 딕셔너리의 요소 접근(1)

## ◆ [ ] 첨자 연산으로 요소에 접근하기

딕셔너리 이름 [ 접근할\_요소의\_키 ]

- 첨자로 요소 번호가 아닌 접근하고자 하는 데이터에 대한 키 값을 전달
  - 유효하지 않은 키 사용시 KeyError 발생
- 연산의 결과로 해당 키로 맵핑되어 저장된 데이터 값을 결과값으로 가짐

# 딕셔너리의 요소 접근(1)

## ◆ 반드시 유효한 키 값 지정 필요

# 코드 9-24

첨자 연산을 통한 딕셔너리의 요소 접근 예

```
01 d = { 'python':'파이썬', 'basic':'기초', 'programming':'프로그래밍' }
02 print(d['python'])    # 'python'이란 키로 저장된 데이터 '파이썬' 출력
03
04 print(d[0])          # 0이란 키로 저장된 데이터는 없으므로 오류 발생
```



실행 결과

파이썬

Traceback (most recent call last):

```
  File "C:\pyWork\runme.py", line 4, in <module>
    print(d[0])      # 0이란 키로 저장된 데이터는 없으므로 오류 발생
  KeyError: 0
```

# 딕셔너리의 요소 접근(2)

## ◆ get() 메서드로 요소에 접근하기

```
딕셔너리이름.get(접근할_요소의_키)
```

- 유효하지 않은 키 사용시 아무 것도 반환되지 않음
  - 오류 발생하지 않음

#

코드 9-25

get() 메서드를 이용한 딕셔너리의 요소 접근 예

```
01 d = { 'python':'파이썬', 'basic':'기초', 'programming':'프로그래밍' }  
02 print(d.get('python'))      # '파이썬' 출력  
03  
04 res = d.get(0)              # 반환되는 값이 없음  
05 print(res)                 # None 출력
```



실행 결과

파이썬  
None

# 키의 유효성 검사

- ◆ 유효한 키인지의 여부를 True/False로 반환

키 **in** 딕셔너리를

# 코드 9-26

in 연산과 첨자 연산을 이용한 딕셔너리의 요소 접근 예

```
01 d = { 'python':'파이썬', 'basic':'기초', 'programming':'프로그래밍' }  
02  
03 key = 0  
04 if key in d :  
05     print(d[key])  
06 else :  
07     print(f'오류: 유효하지 않은 키 {key}')
```



실행 결과

오류: 유효하지 않은 키 0

# 딕셔너리에 요소 추가

- ◆ [ ] 첨자 연산을 이용해 새 키에 대한 값 추가

딕셔너리의 [새로\_추가할\_요소의\_키] = 값

## # 코드 9-27 딕셔너리의 요소 추가

```
01 d = { 'python':'파이썬', 'basic':'기초', 'programming':'프로그래밍' }  
02 d['PYTHON'] = '파이썬'  
03 print(d)
```



### 실행 결과

```
{'python': '파이썬', 'basic': '기초', 'programming': '프로그래밍',  
'PYTHON': '파이썬'}
```

# 딕셔너리의 요소 수정

- ◆ [ ] 첨자 연산을 이용해 기존 키에 대한 값 수정

딕셔너리 이름[수정할\_요소의\_키] = 새로운\_값

## # 코드 9-28 딕셔너리의 요소 수정

```
01 d = { 'python':'파이썬', 'basic':'기초', 'programming':'프로그래밍' }  
02 d['python'] = 'PYTHON'  
03 print(d)
```



### 실행 결과

```
{'python': 'PYTHON', 'basic': '기초', 'programming': '프로그래밍'}
```

# 딕셔너리의 요소 삭제

- ◆ del 문과 [] 첨자 연산을 이용해 기존 키를 갖는 요소 삭제

```
del 딕셔너리이름[삭제할_요소의_키]
```

#

코드 9-29

딕셔너리의 특정 요소 삭제

```
01 d = { 'python':'파이썬', 'basic':'기초', 'programming':'프로그래밍' }  
02 del d['basic']  
03 print(d)
```



실행 결과

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

# 딕셔너리의 전체 요소 삭제

- ◆ `clear()` 메서드는 빈 딕셔너리로 만듦

```
딕셔너리이름.clear()      # 모든 요소 삭제
```

#

코드 9-30

딕셔너리의 모든 요소 한 번에 삭제

```
01 d = {'python':'파이썬', 'basic':'기초', 'programming':'프로그래밍'}  
02 d.clear()  
03 print(d)
```



실행 결과

```
{}
```

# 복합 자료형의 데이터 생성시 사용되는 괄호

---

- ◆ ()
  - 튜플 만들 때 소괄호 사용
- ◆ []
  - 리스트 만들 때 대괄호 사용
- ◆ {}
  - 딕셔너리 만들 때 중괄호 사용

# 문자열, 튜플, 리스트, 딕셔너리 비교

	기본 자료형	복합 자료형		
	문자열	튜플	리스트	딕셔너리
선언	$s = '값'$ $s2 = "값"$	$t = (\text{값}1, \text{값}2)$ $t2 = \text{값}1, \text{값}2$	$l = [\text{값}1, \text{값}2]$	$d = \{\text{키}1:\text{값}1, \text{키}2:\text{값}2\}$
요소 접근	$s[\text{인덱스}]$	$t[\text{인덱스}]$	$l[\text{인덱스}]$	$d[\text{키}1]$ $d.get(\text{키}1)$
요소 수정	불가	불가	$l[\text{인덱스}] = \text{수정값}$	$d[\text{키}1] = \text{수정값}$
요소 추가	불가	불가	$l.append(\text{새값})$ $l.insert(\text{인덱스}, \text{새값})$	$d[\text{새키}] = \text{새값}$
요소 삭제	불가	불가	$del l[\text{인덱스}]$ $del l[\text{시작}: \text{끝}+1]$ $l.remove(\text{삭제할값})$	$del d[\text{키}1]$
전체 삭제	불가	불가	$l.clear()$	$d.clear()$
개수 확인	$\text{len}(s)$	$\text{len}(t)$	$\text{len}(l)$	$\text{len}(d)$

# 연습문제 9.1

---

## ◆ 세 학생의 점수를 입력 받아, 이 점수와 평균을 출력

- 사용자로부터 점수 3개를 입력 받아 리스트 scores에 저장 (append() 이용. For문을 사용하면 편하나, 사용하지 않아도 무방)
- 리스트에 저장된 점수들의 평균을 구함
- 리스트에 저장된 점수들과 함께 평균을 소수점 첫째자리까지 출력
- 실행 화면

[점수 입력]

#1? 90

#2? 85

#3? 100

[점수 출력]

입력 점수: 90 85 100

평균: 91.7

## 연습문제 9.2

---

- ◆ 앞 문제에, 특정 점수를 받은 학생의 번호 확인 기능 추가
  - 학생의 번호는 입력순서와 같고, 동일 점수 받은 학생은 없다 가정
  - index() 메소드를 호출하여 구현 가능
  - 실행 결과

[점수 입력]

#1? 88

#2? 24

#3? 55

[점수 출력]

입력 점수: 88 24 55

평균: 55.7

[검색]

찾고자 하는 점수는? 88

88점은 1번 학생의 점수입니다.

# 연습문제 9.3

## ◆ 쇼핑몰 장바구니 시뮬레이션 프로그램 작성

- shopping\_bag = [] #장바구니  
while True:  
    # item = 상품 이름을 입력받기  
    # item이 빈 문자열이면 루프 빠져 나가기  
    # item을 장바구니에 추가  
    # 장바구니의 모든 상품 이름 출력 (리스트 이름 사용)

- 실행 결과 [구입]  
    상품명? 치약  
    장바구니에 치약가(이) 담겼습니다.

상품명? 칫솔  
장바구니에 칫솔가(이) 담겼습니다.

상품명?

>>> 장바구니 보기: ['치약', '칫솔']

# 개념 확인 과제 (연습문제 9.4)

## ◆ 연습문제 9.3에 수량 저장/출력 기능 추가

- 딕셔너리를 사용하여 수량을 저장
- 장바구니 내 상품을 검색하여 해당 상품의 수량 출력
- 실행 결과

[구입]

상품명? 사과  
수량은? 10

장바구니에 사과 10개가 담겼습니다.

상품명? 바나나

수량은? 2

장바구니에 바나나 2개가 담겼습니다.

상품명?

>>> 장바구니 보기: {'사과': 10, '바나나': 2}

[검색]

장바구니에서 확인하고자 하는 상품은? 바나나  
바나나은(는) 2개 담겨 있습니다.

## ◆ 제출 기한 및 방법

- 다음 수업시간 전까지 본인의 repository에 hw7.py을 업로드