

## 2. 파이썬 프로그래밍

# 스마트시스템캡스톤디자인

수요일 12-3시  
미정관 406호

Prof. Hae Won Byun

# Curriculum

- 9월 04일 : 과목소개 및 주요사항
- 9월 11일 : 파이썬 프로그래밍
- 9월 18일 : 파이썬 프로그래밍
- 9월 25일 : 라즈베리파이 기본 개념, 라즈비안 운영체제 설치
- 10월 02일 : LED 제어
- 10월 09일 : 한글날
- 10월 16일 : 모터 제어
- 10월 23일 : 카메라 사용
- 10월 30일 : 중간고사
- 11월 06일 : SPI 인터페이스 (프로젝트 계획서 제출)
- 11월 13일 : I2C 인터페이스
- 11월 20일 : 라즈베리파이 프로젝트
- 11월 27일 : 라즈베리파이 프로젝트
- 12월 04일 : 라즈베리파이 프로젝트
- 12월 11일 : 라즈베리파이 프로젝트
- 12월 18일 : 라즈베리파이 프로젝트 발표



# 3장

## 파이썬 기본 지식

## 3장 파이썬 기본 지식

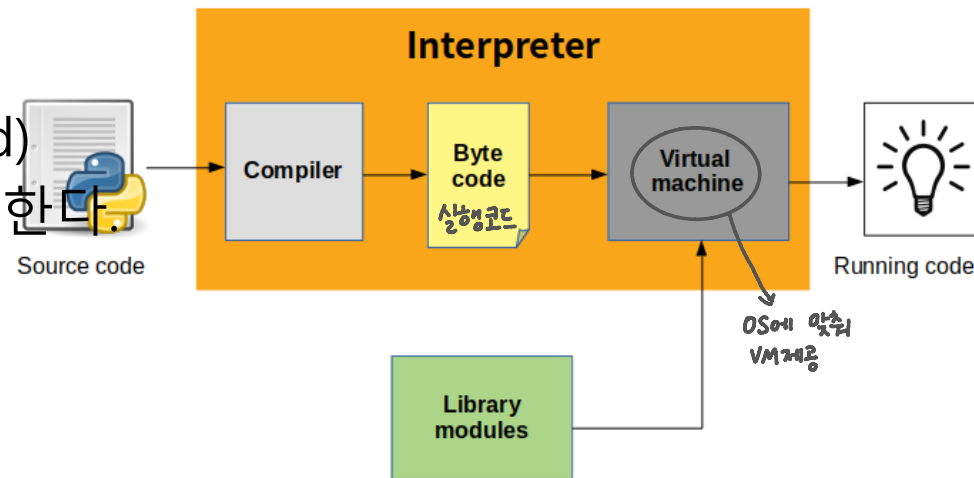
1. 파이썬이란?
2. Hello World 프로그램 작성하기
3. 프로그래밍 기본 지식
4. 기본 자료형 - 다양한 데이터 다루기
5. 상황에 따른 처리
6. 파일 조작
7. 함수 만들기
8. 모듈
9. GUI 프로그램

## 1. 파이썬이란?

---

# 파이썬 개요

- 파이썬
  - 1991년 프로그래머 'Guido vanRossum'이 발표
  - 고급 프로그래밍 언어
- 플랫폼 독립적 (가상머신, virtual machine)
  - 서로 다른 플랫폼에서 실행할 수 있도록 파이썬 가상머신 제공
  - 동작 플랫폼 : 윈도우, 리눅스, 유닉스, 맥, 라즈비안 라즈베리파이 OS
- 인터프리터식
  - 소스 코드를 바로 실행하는 컴퓨터 프로그램 (중간 코드로 변환 후 바로 실행) 가계어가 아닌 중간코드로 빠르게 변환
- 객체지향 프로그래밍
- 동적타이핑 (dynamically typed)
  - 실행 시간에 자료형을 검사한다. C언어 → 컴파일시간

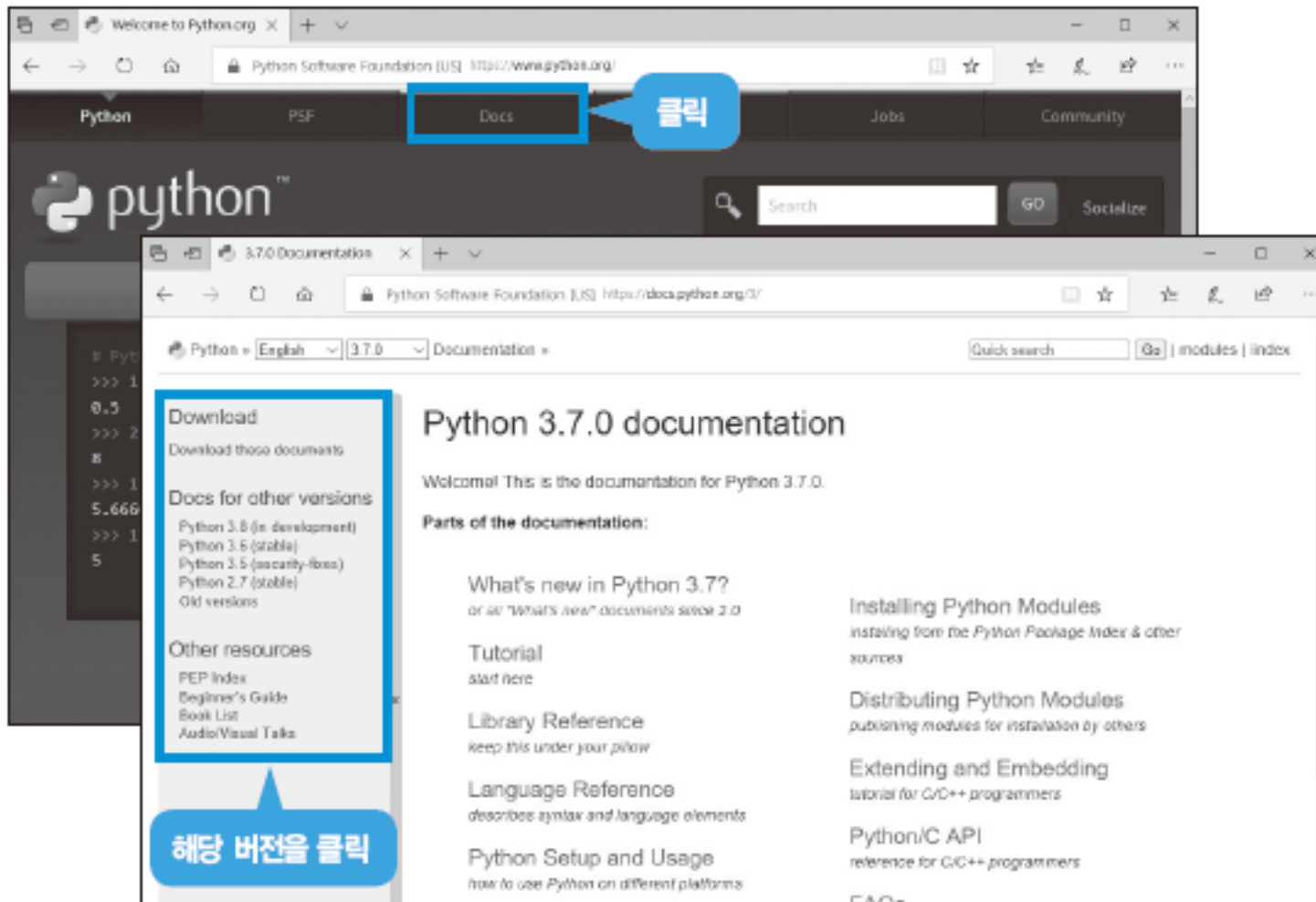




# 파이썬 다운로드

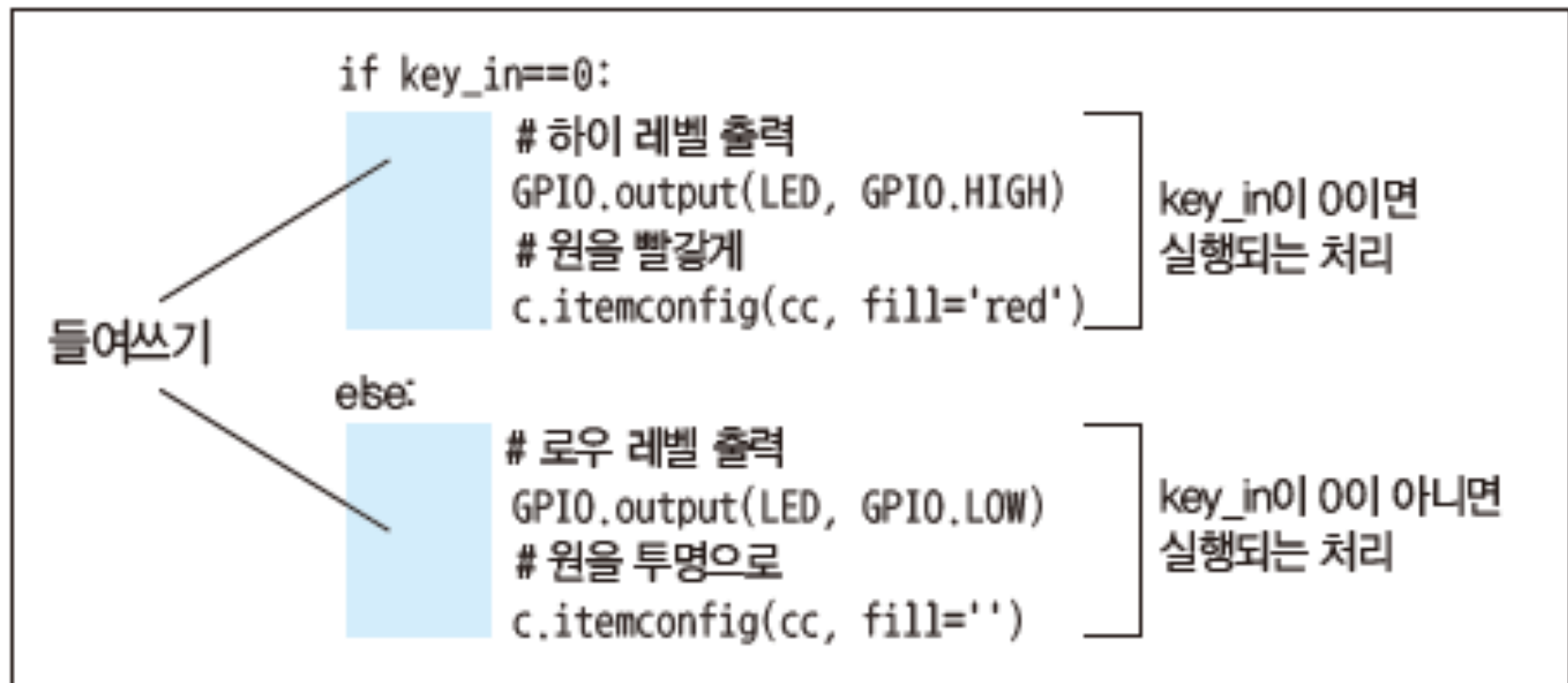
path 설정 관련 옵션 클릭

▼ 그림 3-2 파이썬 사이트(<https://www.python.org>)



## 파이썬 특징

- **들여쓰기로 블록 구조를 표현함**
    - 파이썬은 함수 정의나 if 문 같은 구문 범위를 **들여쓰기로 표현함**
    - vs. 다른 프로그래밍 언어는 중괄호({})를 주로 사용
- ▼ 그림 3-3 파이썬 들여쓰기





## 2. Hello World 프로그램 작성하기

---

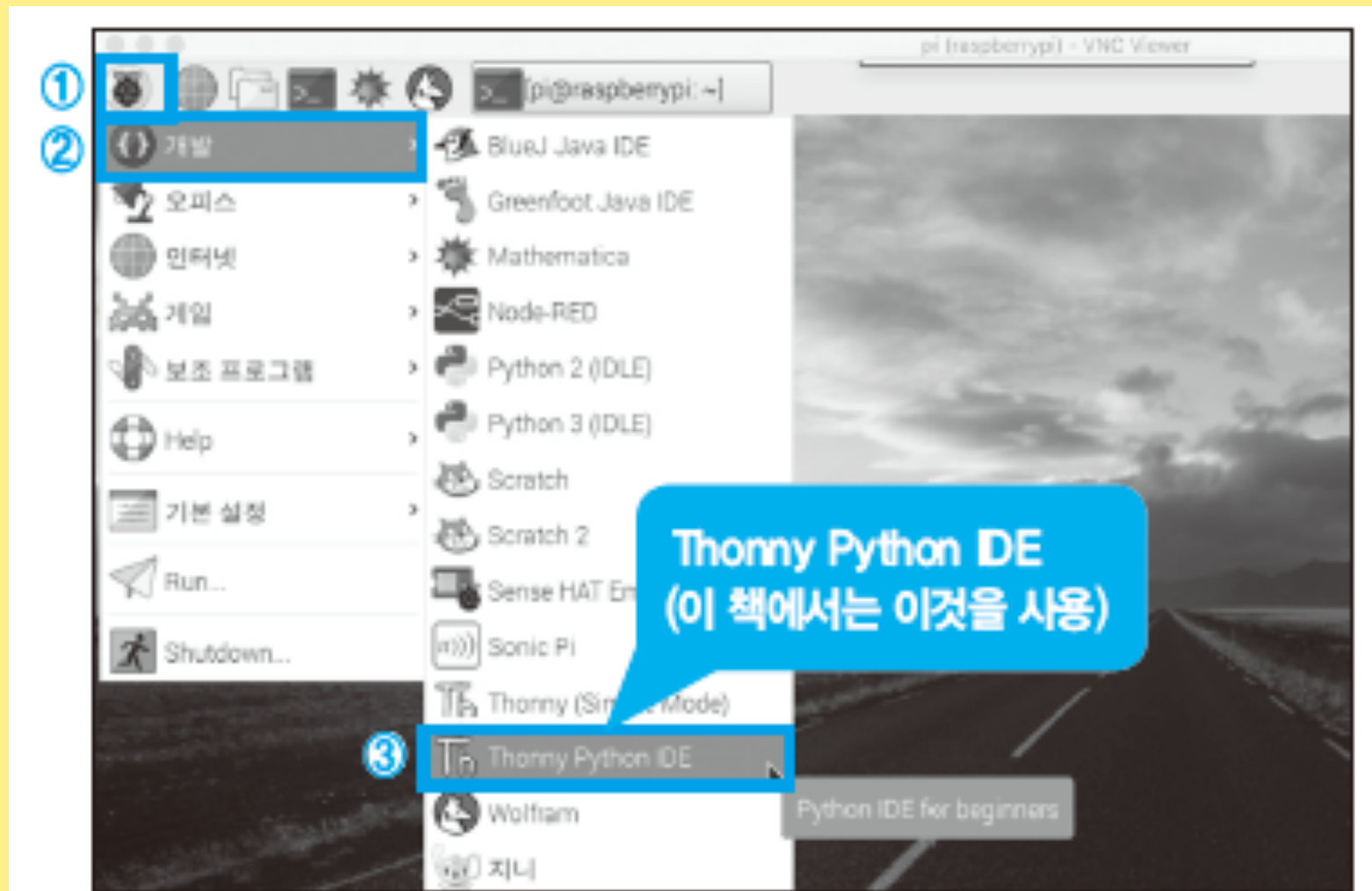
# 1. IDE 사용 방법

### ● IDE 사용 방법


- 대부분의 프로그래밍 언어에는 프로그램 작성과 실행을 돕는 다양한 기능을 가진 편리한 애플리케이션이 있음
- 애플리케이션을 IDE(Integrated Development Environment, 통합 개발 환경)라고 부름
- 텍스트 에디터로도 프로그램을 작성할 수 있지만 IDE를 사용하면 프로그램 개발 작업 전반인 '프로그램 작성 → 실행 → 디버그'를 원활하게 진행할 수있음
- 파이썬으로 프로그램을 작성하는 데 사용할 수 있는 IDE는 라즈비안에도 설치되어 있음
- 우선 IDE를 실행하는 방법은 데스크톱 왼쪽 위에 있는 Menu 아이콘을 클릭
- 프로그래밍 > Thonny Python IDE를 선택함
- Thonny Python IDE 아이콘이 데스크톱에 있으면 그 아이콘을 클릭해도 됨

# 1. IDE 사용 방법

### ▼ 그림 3-4 Thonny Python IDE 실행하기



## 2. 대화형 세션으로 프로그램 실행하기

- 대화형 세션으로 프로그램 실행하기
  - Thonny Python IDE를 실행
  - Python Shell(입력한 파이썬 명령어를 실행하고 결과를 표시하는 사용자 인터페이스 프로그램) 창이 열림
  - 이 창에 표시되는 프롬프트( >>>) 뒤에 파이썬 프로그램을  성하고 키를 누르면 프로그램이 실행
  - 파이썬에서는 이를 대화형 세션이라고 함

### ▼ 그림 3-5 print 문

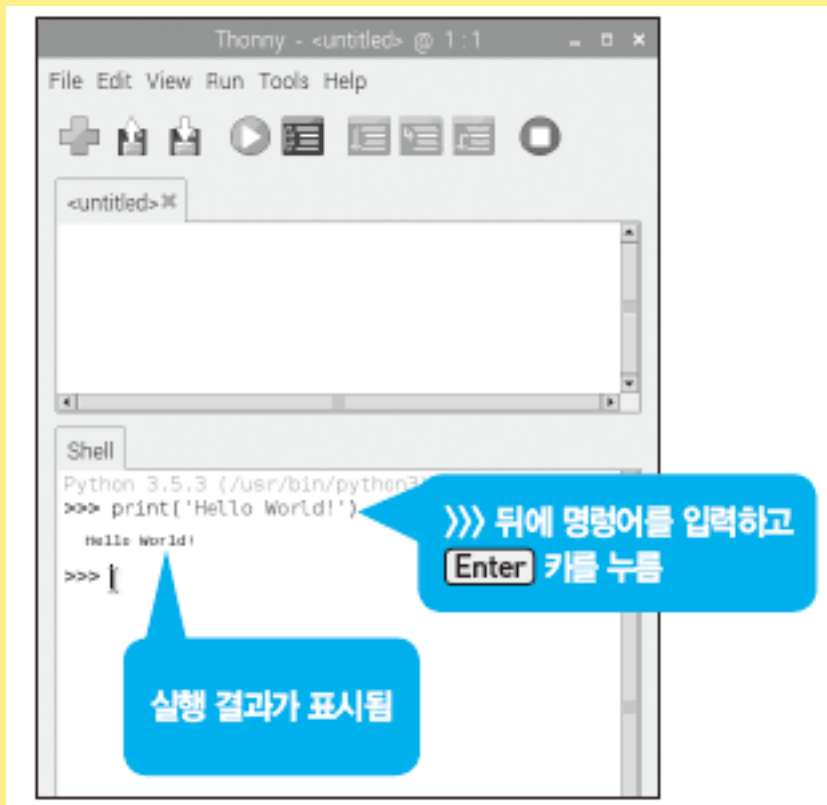
```
print('문자열')
```



작은따옴표(') 또는 큰따옴표(")로 감쌘

## 2. 대화형 세션으로 프로그램 실행하기

### ▼ 그림 3-6 대화형 세션으로 print 문을 실행



## 3. 프로그램을 파일로 저장하기

### ● 프로그램을 파일로 저장하기

- 앞에서는 대화형 세션으로 프로그램을 실행했는데 이 방법은 실행할 때마다 프로그램을 작성해야 함
- 여러 번 실행하는 프로그램이나 복잡한 프로그램에는 어울리지 않음
- IDE 에디터로 작성한 프로그램을 저장해 두면 프로그램을 몇 번이고 반복해서 실행
- 나중에 변경하거나 추가할 때 편함
- IDE로 새 파일을 만들어 프로그램을 작성하려면 툴바에 있는 + 를 클릭
- 여기서 작성한 프로그램을 파일로 저장할 수 있음



## 3. 프로그램을 파일로 저장하기

### ▼ 그림 3-7 IDE로 새 파일을 만들어 프로그램 작성하기



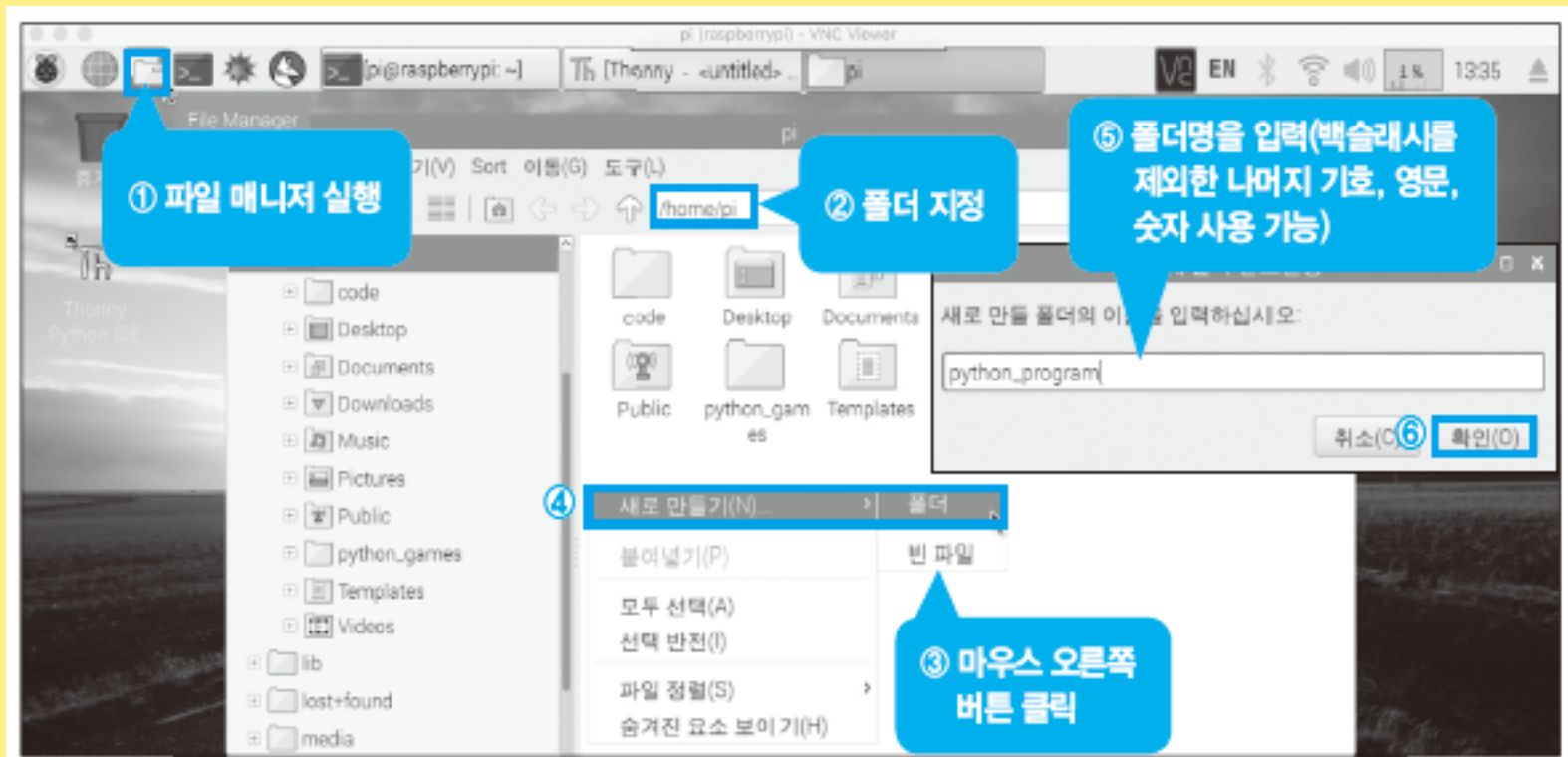
## 3. 프로그램을 파일로 저장하기

### ● 프로그램을 파일로 저장하기

- 현재 라즈비안의 IDE는 파일을 저장할 때 새 폴더를 작성할 수 없음
- 파일 매니저로 /home/pi 아래에 미리 python\_program 폴더를 만듦
- 여기에 프로그램 파일을 저장함
- 파일 매니저를 실행하려면 작업 표시줄의 파일 매니저 아이콘을 클릭

## 3. 프로그램을 파일로 저장하기

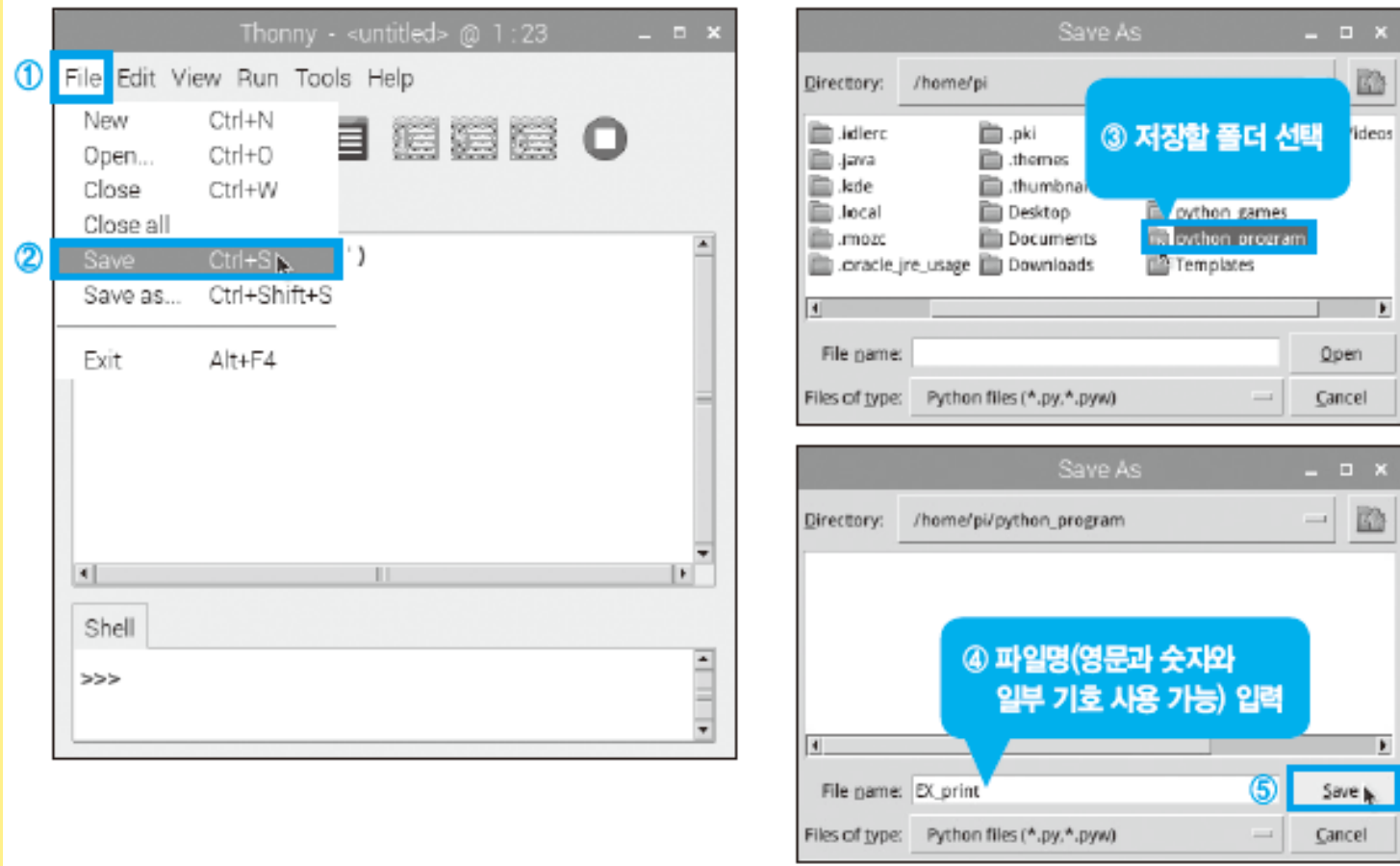
### ▼ 그림 3-8 폴더 작성



↳ Linux 화면

## 3. 프로그램을 파일로 저장하기

### ▼ 그림 3-9 IDE에서 파일 저장



## 3. 프로그램을 파일로 저장하기

### ● 프로그램을 파일로 저장하기

- 폴더명과 파일명에는 영문, 숫자, 백슬래시(\)를 제외한 나머지 기호만 사용할 수 있음
- 프로그램을 실행하려면 프로그램을 읽어들이는 상태에서 F5 키를 누르거나 IDE에서 Run > Run Module을 선택
- Run current script를 클릭해도 됨
- 실행 결과는 셸 창에 표시됨
- 프로그램 내용을 변경해서 다른 프로그램으로 저장하고 싶다면 File 메뉴에서 Save As를 클릭

## 3. 프로그램을 파일로 저장하기

### ▼ 그림 3-10 IDE에서 프로그램 실행하기





# 파이썬 통합개발환경(IDE)

- IDLE

- 프로그램 또는 파일 검색으로 찾아서 실행

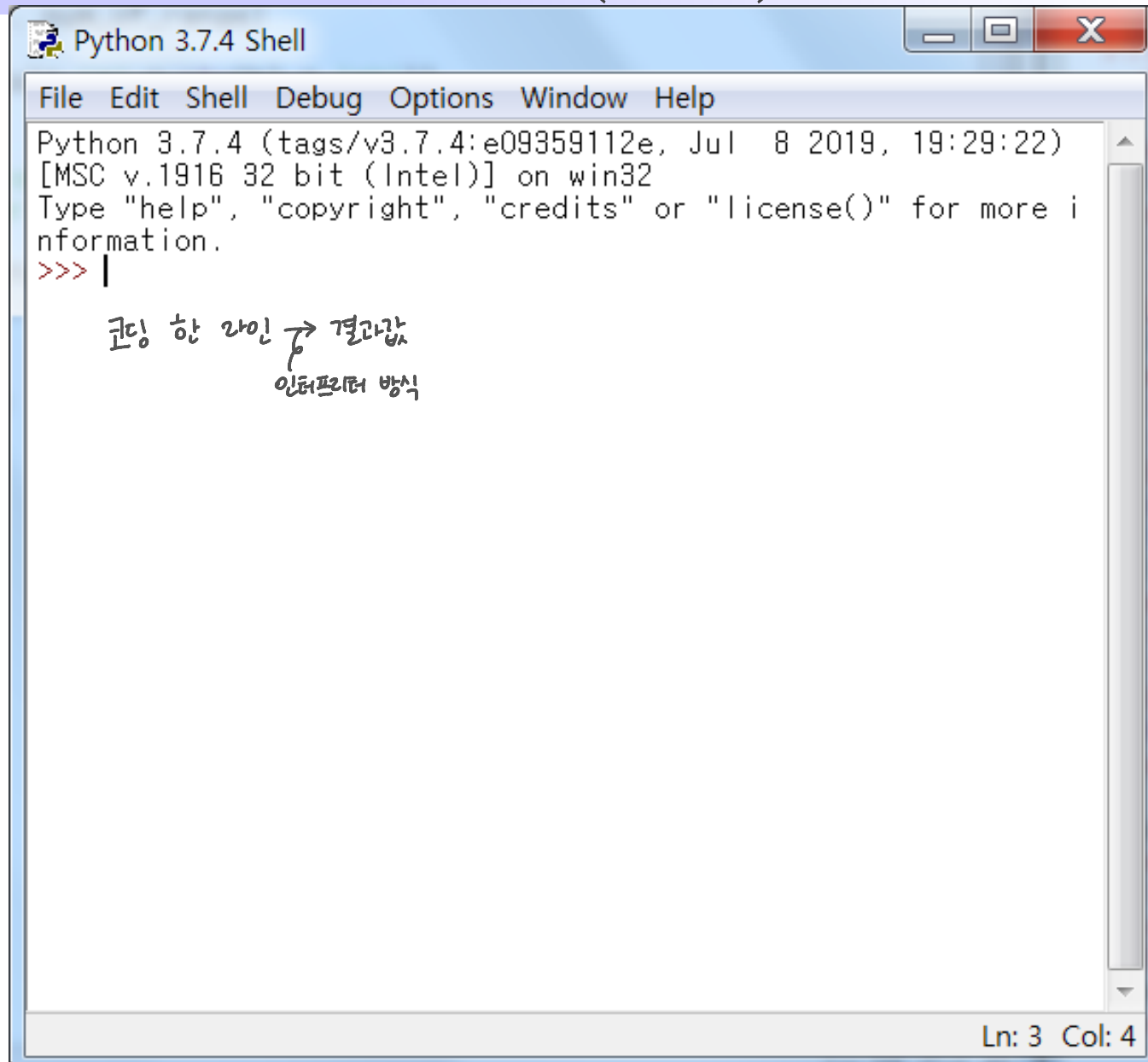
- 프로그래밍

- Python Shell

- **.py 파일** 만 들어서 실행

- File > New File

코드가 길어질 경우 →



# 파이썬 통합개발환경(IDE)

Python 3.7.4 Shell

File Edit Shell Debug Options Window Help

Python 3.7.4 (tags/v3.7.4:e09359112e, [MSC v.1916 32 bit (Intel)] on win32  
Type "help", "copyright", "credits" or  
nformation.  
>>> |

a.py - C:/Users/hyewonByun/AppData/Local/Progra...

File Edit Format Run Options Window Help

```
import time
import random

num_of_times = 5
game_time = 25
num_of_range = 100

start_time = time.time()

for i in range(num_of_times):
    a = random.randint(1, num_of_range)
    b = random.randint(1, num_of_range)
    c = a + b
    ans = int(input(str(a) + '+' + str(b) + '=>'))

    if ans != c :
        print('Sorry, wrong answer')
        print('The answer is ' + str(c))
        break
    elif time.time() - start_time > game_time :
        print('Time Out')
        break
    else :
        print('Bingo !')
else:
    print('Complete')
print('end of program')
|
```

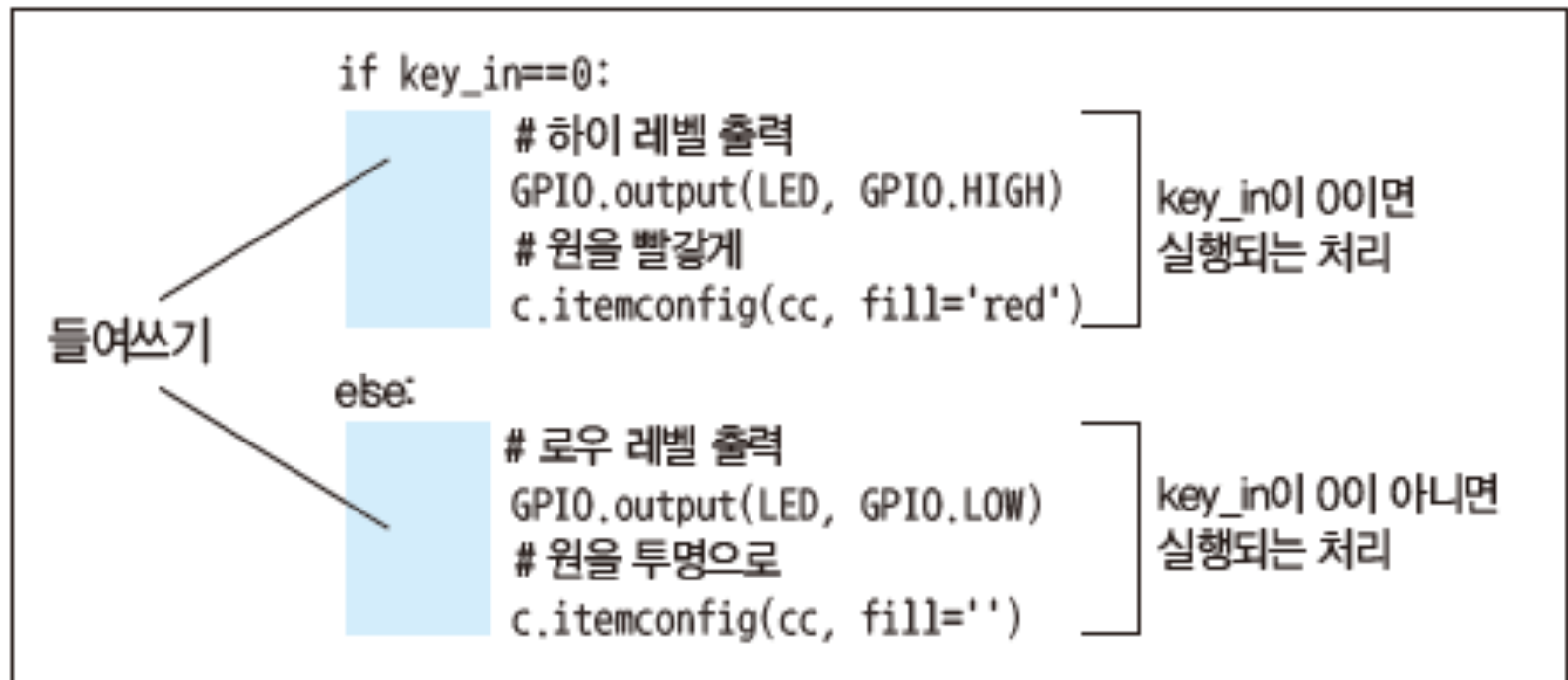
Ctrl-S : 파일 저장  
F5 : 실행

.PY 파일을 실행한  
실행결과가  
Shell에  
나타남

Ln: 28 Col: 0

## 4. 파이썬 프로그램 작성 규칙

- 파일명은 영문, 숫자, 기호만 사용
- 공백
  - 문장 시작 부분에 Spacebar 키로 공백을 입력하면 안 됨
  - 문법에서 들여쓰기가 필요한 부분은 별개임



## 4. 파이썬 프로그램 작성 규칙

### ● 주석

- 파이썬은 #으로 시작하는 라인을 주석으로 처리
- 파이썬 3.x는 표준 문자 코드가 UTF-8이므로 주석에 한글 사용 가능
- 파이썬 2.x는 표준 문자 코드가 아스키(ASCII)이므로 그대로는 사용할 수 없음
  - 파이썬 2.x에서도 문제없이 동작하게 하려면 프로그램 첫 부분에 # coding:utf-8을 입력

#### ▼ 그림 3-11 한글 주석 넣기

```
# coding:utf-8
```

```
# Hello World!라고 표시하기
```



파이썬 2.x 대응을 위해 UTF-8 지정  
#부터 다음 줄의 앞까지가 주석이 됨

```
print('Hello World!') → python interpreter가 하는 것이  
                          많아 "아" " " " " 상관 X
```

## 4. 파이썬 프로그램 작성 규칙

- 긴 줄 하나를 여러 줄로 작성

- 프로그램을 작성하다가 줄이 너무 길어지면 백슬래시( `\` )를 입력하고 줄 바꿈을 하면 됨

한 줄은 아무 곳에도 들어갈 수 있음

#으로 시작하는 줄은 주석 처리

```
# coding:utf-8
```

UTF-8 지정

```
# 중간에 백슬래시(\)를 넣어서 줄 바꾸기
```

```
print \
```

```
('Happy Python')
```

프로그램 중간에 백슬래시를  
넣으면 줄 바꿈이 가능

```
print ('Happy Python!')
```

문장 구성 요소 사이에는 공백을 여러 칸 넣을 수 있음

문장이 시작되는 부분에는 공백은 넣을 수 없음

원칙적으로 프로그램은  
위에서부터 순서대로 실행됨

실행 결과

```
Happy Python
Happy Python!
```

## 5. 파이썬 문법 오류

### ● 오류 메시지

- 함수의 괄호가 빠졌을 때

▼ 예제 print 문의 괄호가 없음

```
>>> print'Python'
File "<pyshell>", line 1 ← 오류가 발생한 줄
    print 'Python'      ← 틀린 부분
        ^
SyntaxError: Missing parentheses in call to 'print'
```

- 줄이 시작되는 부분에 공백이 있을 때

▼ 예제 print 문 앞에 공백이 있음

```
>>>  print 'Python'

File "<pyshell>", line 1
    print('Python')
    ^
SyntaxError: unexpected indent
```



## 5. 파이썬 문법 오류

- 대화형 세션의 오류 메시지

- 특수문자가 있을 때

- ▼ 예제 print 문 뒤에 특수문자가 있음

```
>>> print ( 'Python' )  
File "<pyshell>", line 1  
    print ( 'Python' )  
          ^
```

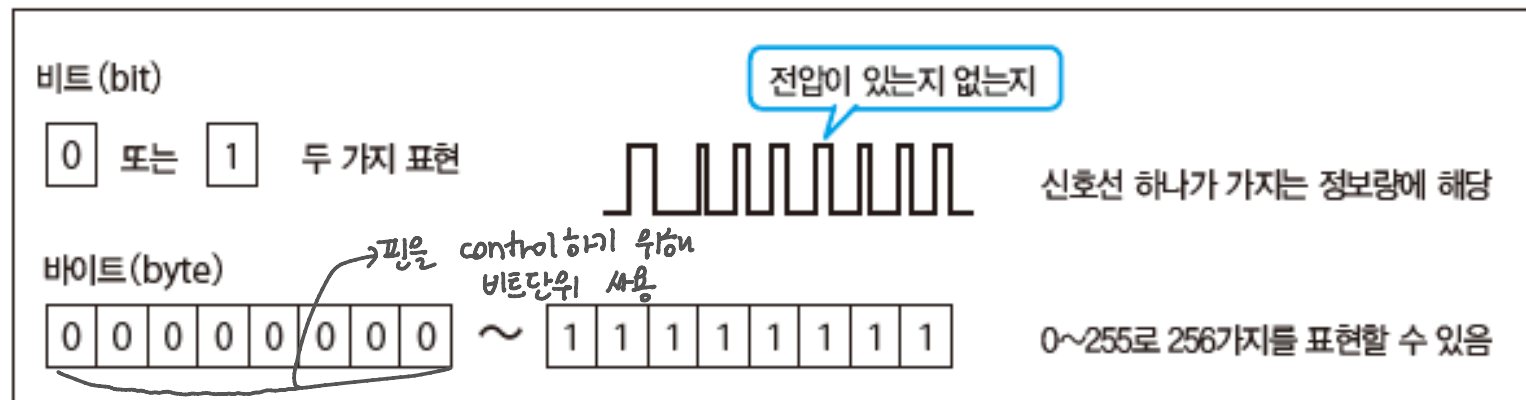
SyntaxError: `invalid character in identifier`

### 3. 프로그래밍 기본 지식

---

# 1. 컴퓨터에서 다루는 데이터 단위와 자료형

- 데이터 단위와 프로그램에서 사용하는 자료형
    - 16비트, 32비트, 64비트 등 프로그램을 실행하는 컴퓨터의 정해진 비트 너비(범위)에 따라 **int** 형의 크기가 결정됨 int 크기가 달라짐
    - 라즈베리 파이에서 파이썬을 실행하면 **int** 형의 크기는 32비트가 됨
    - 파이썬은 프로그램 실행 중에 값이 커져서 원래 크기를 초과하면 **자동으로 비트 범위를 확장**하므로 자료형의 크기를 신경 써야 할 일이 거의 없음
- ▼ 그림 3-16 컴퓨터에서 다루는 단위



# 1. 컴퓨터에서 다루는 데이터 단위와 자료형

▼ 그림 3-17 2진수, 16진수, 10진수 대응표

비트	각	가독성
2진수	16진수	10진수
0	0	0
1	1	1
10	2	2
11	3	3
100	4	4
101	5	5
110	6	6
111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15
10000	10	16

2진수와 16진수의  
자릿수가 똑같이 올라감



# 1. 컴퓨터에서 다루는 데이터 단위와 자료형

## ● 2진수, 10진수, 16진수

- 프로그램을 작성할 때는 일반적인 숫자를 10진수로 표기함
- 하드웨어 상태나 각종 비트를 의미하는 데이터를 16진수로 표기하는 등 그 숫자가 의미하는 대상에 따라 구분해서 사용
- 4장에서는 라즈베리 파이와 센서 IC 사이의 통신을 위해 센서 IC를 만든 곳에서 미리 정해 둔 주소나 명령어를 프로그램에서 사용
- 각종 부품 데이터 시트(부품 사양 등이 적혀 있는 서류)에는 주소나 명령어 등이 보통 16진수로 작성되어 있어서 16진수로 쓰는 것이 알아보기 쉽고 편함

## 2. 논리 연산

### ● 논리 연산 → 2진수나 16진수로 논리연산 사용

- 논리 연산이란 컴퓨터 세상에서 자주 쓰는 연산으로 '참'과 '거짓' 조건을 **논리곱(AND)**, **논리합(OR)**, **부정(NOT)**으로 연산
- 프로그래밍 언어에서는 참을 **True**, 거짓을 **False**로 표현함
- True 또는 False 값을 갖는 자료형을 불(Boolean)이라고 함

▼ 첫 문자만 대문자 그림 3-18 진릿값 표

논리곱(AND)

조건 A	조건 B	출력
True	True	True
True	False	False
False	True	False
False	False	False

논리합(OR)

조건 A	조건 B	출력
True	True	True
True	False	True
False	True	True
False	False	False

부정(NOT)

조건 A	출력
True	False
False	True



## 2. 논리 연산

### ● 논리 연산

- a가 True고 b가 False일 때 **논리곱 and**를 쓰면 False가 됨

>>> a = True      동적타이핑이므로 변수선언을 미리 할 필요 x  
 >>> b = False  
 >>> a and b  
 False

shell  
코딩 방식

- a가 True고 b가 False일 때 b에 **부정 not**을 붙이고 논리곱 and를 사용하면 True가 됨

>>> a = True  
 >>> b = False  
 >>> a and not b  
 True      파이썬이 제공하는 키워드 (소문자)

## 2. 논리 연산

- 논리 연산

- a가 True고 b가 False일 때 **논리합 or**를 사용하면 True가 됨

```
>>> a = True
>>> b = False
>>> a or b
True
```

- a가 True고 b가 False고 c가 True일 때 **논리곱 and와 논리합 or**

```
>>> a = True
>>> b = False
>>> c = True
>>> a and b and c
False
>>> a or b or c
True
```

## 3. 문자 정보 처리

### ● ASCII 코드

- 영미권에서 표준으로 사용하는 문자 코드는 미국에서 정한 **아스키(ASCII, American Standard Code for Information Interchange)**임
- 아스키는 영미권에서 고안한 것으로 숫자, 기호, 줄 바꿈 같은 제어 문자와 알파벳, 이렇게 **128개의 코드**로 이루어져 있음
- 아스키는 모두 1바이트만 사용하므로 **'1바이트 문자'**라고도 부름

### ● 한글 문자 표현

- 한국에서는 한글을 다루기 위해 문자 코드로 **2바이트 조합형, 2바이트 완성형, EUC-KR, CP949** 등을 사용함
- 한글은 알파벳보다 다뤄야 하는 글자 수가 많아서 1바이트로는 부족함
- 길이가 **2바이트** 이상인 수를 사용함
- 이런 문자를 1바이트 문자와 비교해서 **'멀티바이트 문자'**라고도 부름

## 3. 문자 정보 처리

### ● 한글 문자 처리

- 문자 코드 : 모든 언어에서 사용하는 문자를 같은 문자 코드 체계로 표시하기 위해 **유니코드(Unicode)**라는 문자 코드가 고안됨
- 인코딩 방식 : 그 중에서도 **UTF-8**은 유니코드 한 문자를 **8비트 단위로 끊어서 표현**하는 방식
- 라즈비안은 표준 문자 코드가 UTF-8이므로 한글을 사용할 때 UTF-8을 사용

```
# coding:utf-8
# Hello World!라고 표시하기
```

```
print('Hello World!')
```



파이썬 2.x 대응을 위해 UTF-8 지정  
#부터 다음 줄의 앞까지가 주석이 됨

## 4. 함수, 메서드, 모듈 사용 방법

- 함수

- 함수는 어떤 단위로 묶어서 나눌 수 있는 처리를 하나의 블록으로 만든 것

- 메서드

- 객체 지향 프로그래밍에서 객체에 실행하는 조작
- 객체 = 데이터 + 메서드

- 모듈 → `import`

- 모듈보다 큰 개념으로서 라이브러리와 유사

## 4. 함수, 메서드, 모듈 사용 방법

### ● 함수

- 파이썬에서는 라이브러리 등을 따로 불러오지 않고도 바로 함수 형태로 이용할 수 있는 것을 **내장 함수**라고 함

▼ 예제 **내장 함수 int()**를 사용함

*parameter를 int로 바꿔줌*

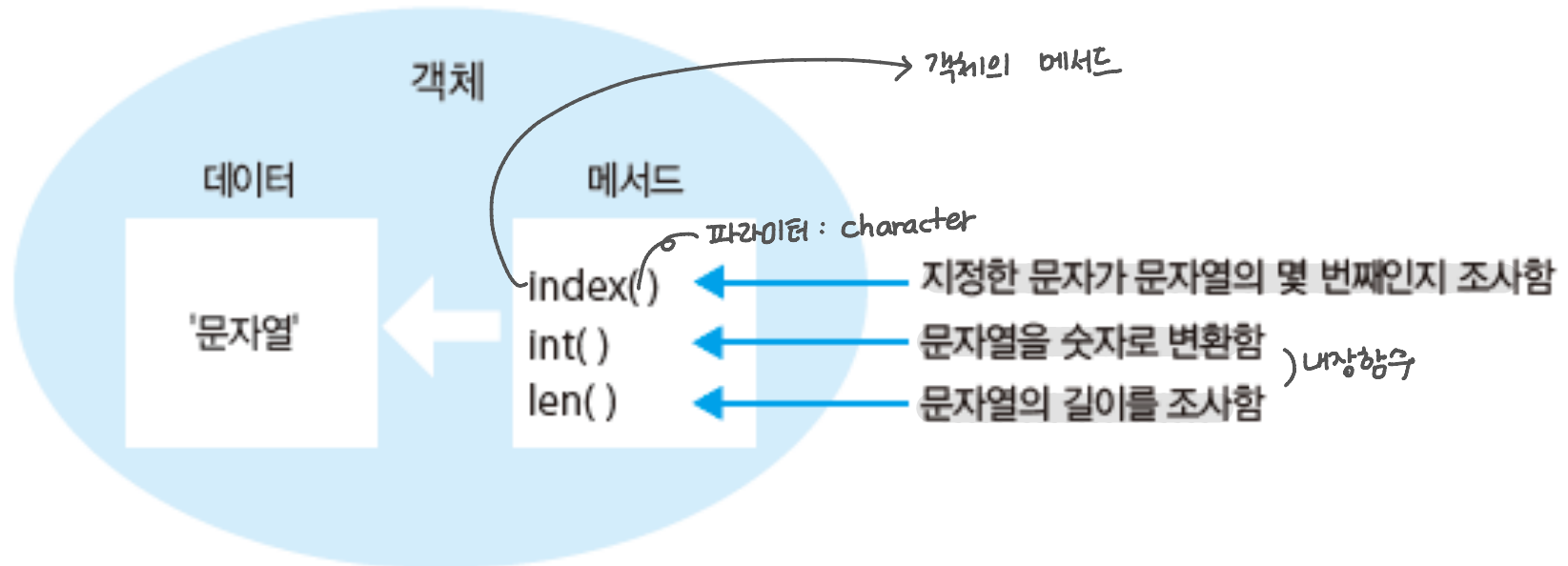
```
>>> int('7')
7
>>> int()
0
```

## 4. 함수, 메서드, 모듈 사용 방법

### ● 메서드

- 객체 지향 프로그래밍은 프로그램이나 데이터를 단계(절차)별로 관리하는 것이 아니라 프로그램에서 다루는 **대상별로** 관리하는 방법임
- 객체는 정보(데이터 또는 프로퍼티라고도 함)와 메서드로 구성

▼ 그림 3-22 객체 구조



## 4. 함수, 메서드, 모듈 사용 방법

### ● 메서드 사용

- 프로그램에서 내장 메서드를 사용하려면 메서드를 사용할 객체와 메서드명 사이에 마침표(.)를 쓰면 됨

```
객체.메서드()
```

### ● 메서드 예제

- index()는 지정한 문자가 몇 번째에 있는지 확인하는 메서드
- 첫 번째 줄에서 문자열 'Python'을 변수 letters에 할당
- 두 번째 줄에서 메서드 index()의 문자 'y'를 인수로 지정
- 파이썬 인덱스는 0부터 시작하므로 첫 글자 'p'의 위치는 0, 'y' 위

```
>>> letters = 'Python' → 문자열 객체로 해석
```

```
>>> letters.index('y')
```

```
1
```



## 4. 함수, 메서드, 모듈 사용 방법

- **모듈**  $\approx$  library

- 모듈은 함수나 메서드보다 큰 단위
- 예) **math 모듈**: 수학에서 사용하는 함수나 상수 포함되어 있음
- 파이썬에서 모듈 기능을 사용하려면 프로그램 시작 부분에서 **사용하려는 모듈을 импорт(import)해야 함**
- 자신이 만든 프로그램을 모듈로 만들어서 다른 프로그램에서 **사용** 할 수도 있음

```
import 모듈명
```

```
모듈.함수()
```

```
>>> import math
>>> math.sqrt(4)
2.0
```

## 4. 함수, 메서드, 모듈 사용 방법

### ▼ 표 3-2 중요한 모듈

모듈명	기능	소개하는 자료형, 메서드, 함수와 작성법
math	수학 관련 모듈 <import 문> import math	math.sqrt(a): a의 제곱근을 돌려줌 math.sin(a): a의 사인 값을 돌려줌 math.radians(a): a의 라디안을 돌려줌 math.pi: 원주율 3.141592... (상수)
sys	운영체제, 시스템 관련 모듈 <import 문> import sys	sys.maxint(): 파이썬의 정수형이 지원하는 정수 최댓값(상수)
random	난수 관련 모듈. 난수란 예측할 수 없는 임의의 값으로 게임이나 뽑기, 암호 생성 등에 사용함 <import 문> import random	random.randint(a, b): a와 b 사이에 있는 임의의 값을 돌려줌

## 4. 함수, 메서드, 모듈 사용 방법

모듈명	기능	소개하는 자료형, 메서드, 함수와 작성법
time	<p>시각 관련 모듈</p> <p>〈import 문〉</p> <pre>import time</pre>	<p>time.asctime(): 인수를 지정하지 않으면 현재 시각을 돌려줌</p> <p>time.sleep(): 인수로 지정한 초만큼 처리를 정지함</p> <p>time.time(): 에포크(epoch, 신기원) 시간에서 경과한 시간을 초로 돌려줌. 인수는 필요없지만 괄호()는 생략할 수 없음. 에포크 시간은 time.gmtime(0)으로 얻을 수 있음</p>
Tkinter	<p>GUI(Graphical User Interface)를 제어하는 모듈</p> <p>〈import 문〉</p> <pre>import Tkinter</pre>	<p>자세한 내용은 227쪽 '9.1 Tkinter 라이브러리'를 참조</p>

## 4. 함수, 메서드, 모듈 사용 방법

모듈명	기능	소개하는 자료형, 메서드, 함수와 작성법
<b>RPi.GPIO</b> 하드웨어 제어	라즈베리 파이의 GPIO 포트를 제어하는 모듈. 라즈베리 파이에서만 사용할 수 있는 모듈. 이 책에서는 임포트할 때 별칭으로 GPIO를 지정해 사용함 <import 문> <code>import RPi.GPIO as GPIO</code>	자세한 내용은 4장을 참조
<b>collections</b> (deque 형)	내장 자료형을 대신하는 특수 자료형의 모 음. 필요한 자료형을 임포트해서 사용함 <import 문> <code>from collections import deque</code>	deque 형(double-ended queue, 디큐) 은 메서드 rotate(n)를 사용할 수 있음. 자세 한 건 4장을 참조

## 4. 기본 자료형 – 다양한 데이터 다루기

---

# 1. 파이썬의 내장 자료형

▼ 표 3-3 파이썬의 주요 내장 자료형과 작성법

내장 자료형	설명	작성법	불변/가변
정수	소수가 없는 숫자	<code>a = 10, a = -10</code>	불변
부동소수	소수가 있는 숫자	<code>a = 10.5</code>	불변
불	True 또는 False 값을 가진 자료형	<code>a = True</code>	불변
문자열	문자 데이터를 순서대로 나열한 데이터. 문자가 1개뿐이어도 문자열이라고 부름	<code>a = 'Python'</code> 문자열의 0번째(첫 글자) 문자는 <code>a[0]</code> 으로 지정	불변
리스트	숫자나 문자열 등의 요소를 순서대로 나열한 구조체	<code>a = ['p', 'y', 't', 'h', 'o', 'n']</code> 리스트 <code>a</code> 의 0번째(첫 글자) 요소는 <code>a[0]</code> 으로 지정	가변
튜플	리스트와 비슷하지만 불변성 때문에 바꿔 쓸 수 없음	<code>a = ('p', 'y', 't', 'h', 'o', 'n')</code> 리스트 <code>a</code> 의 0번째(첫 글자) 요소는 <code>a[0]</code> 으로 지정	불변
딕셔너리	각 데이터에 이름(키)이 있는 데이터 구조	<code>a = {'happy': '(^^)', 'sad': '(ToT)'}</code> 딕셔너리 <code>a</code> 의 키 <code>happy</code> 에 대응하는 값은 <code>a['happy']</code> 로 지정	가변

## 2. 변수

### ● 변수에 할당하기

- C나 자바 언어에서 변수 사용하려면 변수 타입 **미리 선언**
- 컴파일(프로그램을 기계어로 번역하는 처리)할 때 이런 정보가 필요
- 파이썬은 변수를 **미리 선언하지 않음.**
- 변수에 데이터를 할당할 때 할당한 데이터에 따라 변수명과 자료형이 정해지기 때문임

#### ▼ 예제 변수 할당과 계산

```
>>> a = 2
>>> b = 3
>>> a + b
5
```

```
>>> a = 5    → 변수 a가 정수형이 됨
>>> a
5
```

```
>>> a = 'abc' → 변수 a가 문자열형이 됨
>>> a
'abc'
```

## 2. 변수

### ● 변수 이름 규칙

- 대문자와 소문자를 구별함 (ex) ABC와 abc는 서로 다른 변수명
- 변수명은 숫자로 시작할 수 없음 (ex) 01\_data 변수명 사용 불가
- 파이썬 예약어는 사용할 수 없음 (IDE에서 주황색 표시)

#### 파이썬의 예약어

and	as	assert	break	class	continue	def	del
elif	else	except	exec	finally	for	from	global
if	import	in	is	lambda	not	or	pass
print	raise	return	while	with	yield		



## 3. 정수형, 부동소수형, 불형

### ● 정수형

- 정수형 데이터는 10진수 외에도 2진수, 8진수, 16진수로 표기 가능

표기법	접두어	사용할 수 있는 문자
10진수(Decimal)	0 외의 숫자	0~9
2진수(Binary)	0b 또는 0B	0, 1
8진수(Octal)	0o 또는 0O*	0~7
16진수(Hexadecimal)	0x 또는 0X	0~9, a~f(또는 A~F)

\* 0(숫자)과 알파벳 소문자 o 또는 대문자 O

▼ 예제 16진수 값 할당과 내장 함수 hex()

```
>>> a = 0x55
```

```
>>> a
```

(85)

→ 16진수 55는 10진수 85

```
>>> hex(a) → a의 값을 hex로 변경
```

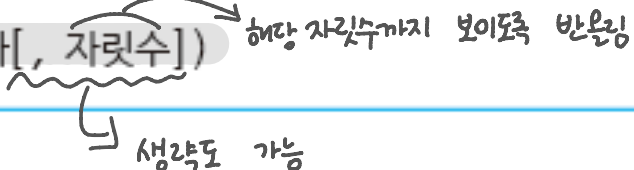
```
'0x55'
```

→ 항상 10진수로 보여줌

## 3. 정수형, 부동소수형, 불형

### ● 부동소수형

- 내장 함수 `round()` : 부동소수의 소수점 이하 반올림
- 반올림할 숫자와 자릿수를 인수로 작성함
- 자릿수를 생략하면 가장 가까운 정수가 반환 값이 되어 표시

`round(반올림할 숫자[, 자릿수])` 

#### ▼ 예제 `round()`로 소수를 반올림

```
>>> a = 1.23456
>>> round(a)
1
>>> round(a, 1)
1.2
>>> round(a, 3)
1.235
```

## 4. 숫자 값 계산

### ● 산술 연산자

▼ 표 3-5 파이썬의 산술 연산자(우선순위가 낮은 것부터)

우선순위 ↓

연산자	의미
$x + y$	x와 y를 더함(덧셈)
$x - y$	x에서 y를 뺌(뺄셈)
$x * y$	x와 y를 곱함(곱셈)
$x / y$	x를 y로 나눔(나눗셈. 정수끼리 계산하면 소수점 이하는 버림)
$x \% y$	나눗셈의 나머지를 구함
$x ** y$	제곱을 구함. $x^y$ (x의 y승)을 의미함

우선순위 ↑

## 4. 숫자 값 계산

- 산술 연산자

- 나눗셈은 나누는 양쪽이 모두 정수면 소수점 이하를 버림
- 한쪽이라도 소수면 소수점 이하까지 계산

▼ 예제 덧셈( $3+2=5$ )

```
>>> 3 + 2  
5
```

▼ 예제 뺄셈( $2-3=-1$ )

```
>>> 2 - 3  
-1
```

▼ 예제 곱셈( $2\times 3=6$ )

```
>>> 2 * 3  
6
```

## 4. 숫자 값 계산

### ▼ 예제 덧셈( $3+2=5$ )

```
>>> 3 / 2
```

```
1          → 정수끼리 계산하면 연산 결과도 정수
```

```
>>> 3.0 / 2
```

```
1.5        → 한쪽이 소수면 연산 결과도 소수
```

```
>>> 3 % 2
```

```
1          → 나머지를 구함
```

### ▼ 예제 제곱(2의 3승)

```
>>> 2 ** 3
```

```
8
```

## 4. 숫자 값 계산

- 산술 연산자

- 연산 우선순위를 변경하려면 미리 실행하고 싶은 계산을 괄호()로 둘러싸야 함

▼ 예제 괄호에 따라 우선순위 변경

```
>>> a = (1 + 1) * 2
```

```
>>> a
```

4

```
>>> a = 1 + 1 * 2
```

```
>>> a
```

3



## 4. 숫자 값 계산

- 복합 할당 연산자

- 복합 할당 연산자란 연산과 할당을 하나로 합친 것임

▼ 표 3-6 파이썬의 복합 할당 연산자

연산식	의미
$x += y$	$x + y$ 의 결과를 $x$ 에 할당
$x -= y$	$x - y$ 의 결과를 $x$ 에 할당
$x *= y$	$x * y$ 의 결과를 $x$ 에 할당
$x /= y$	$x \div y$ 의 결과를 $x$ 에 할당
$x %= y$	$x \div y$ 의 나머지를 $x$ 에 할당
$x \gg= y$	$x$ 를 $y$ 비트만큼 오른쪽으로 시프트*해서 $x$ 에 할당 $\div 2$
$x \ll= y$	$x$ 를 $y$ 비트만큼 왼쪽으로 시프트해서 $x$ 에 할당 $\times 2$

## 4. 숫자 값 계산

- 복합 할당 연산자

- 변수에 복합 할당 연산자를 사용할 때는 **그 변수에 값이 미리 할당되어 있어야** 오류가 발생하지 않음

▼ 예제 값이 할당되어 있지 않은 변수에는 복합 할당 연산자를 사용할 수 없음

```
>>> z += 1
```

```
Traceback (most recent call last):
```

```
File "<pyshell>", line 1, in <module>
```

```
z += 1
```

```
NameError: name 'z' is not defined
```

→ 변수 **z**에는 아무것도 할당되어 있지 않아서  
복합 할당 연산자로 연산할 수 없음



## 4. 숫자 값 계산

### ● 비트 연산자

▼ 표 3-7 파이썬의 비트 연산자

연산식	의미
$x   y$	x와 y의 비트 단위 논리합(OR)
$x \& y$	x와 y의 비트 단위 논리곱(AND)
$x \wedge y$	x와 y의 비트 단위 배타적 논리합(EOR*)
$\sim x$	x의 비트 단위 반전(NOT)
$x \gg y$	x를 y비트만큼 오른쪽으로 시프트, 빈 비트에는 0을 넣음
$x \ll y$	x를 y비트만큼 왼쪽으로 시프트, 빈 비트에는 0을 넣음

\* 배타적 논리합(EOR, Exclusive OR)이란 둘 중 어느 한쪽만 참일 때 참이 되는 논리 연산입니다.

## 4. 숫자 값 계산

- 비트 연산자

- 내장 함수 `bin()`: 2진수 문자열로 변경

▼ 예제 비트 단위 논리합

```
>>> a = 0b100
>>> a = a | 0b110
>>> bin(a)
'0b110'
```

100 or 110  
→ 비트의 한쪽이라도 1이면 1이 됨

▼ 예제 비트 단위 논리곱

```
>>> a = 0b110
>>> a = a & 0b101
>>> bin(a)
'0b100'
```

110 and 101  
→ 비트의 한쪽이라도 0이면 0이 됨

## 4. 숫자 값 계산

### ▼ 예제 비트 단위 배타적 논리합

```
>>> a = 0b110
>>> a = a ^ 0b011
>>> bin(a)
'0b101'
```

→  $110 \text{ xor } 011$   
→ 비트가 서로 같으면 0, 서로 다르면 1이 됨

### ▼ 예제 1비트만큼 오른쪽으로 시프트

```
>>> a = 0b100
>>> a = a >> 1
>>> bin(a)
'0b10'
```

→ 이동한 결과로 세 번째 비트에 0이 들어와서 0b010이 되지만,  
앞에 있는 0은 숫자로는 의미가 없으므로 표시되지 않음

### ▼ 예제 1비트만큼 왼쪽으로 시프트

```
>>> a = 0b001
>>> a = a << 1
>>> bin(a)
'0b10'
```

→ 이동한 결과로 첫 번째 비트에 0이 들어감

## 4. 숫자 값 계산

- 비트 연산자 라즈베리파이
  - GPIO에서 읽은 데이터를 가공할 때 비트 연산 사용
  - 라즈베리 파이뿐만 아니라 하드웨어를 다루는 프로그램에서는 비트 연산을 자주 사용함

▼ 예제 8비트 데이터와 2비트 데이터를 더해서 10비트 데이터를 작성

```
>>> a = 0b11111111
```

```
>>> b = 0b11
```

```
>>> b = b << 8    → 변수 b를 8비트 시프트
```

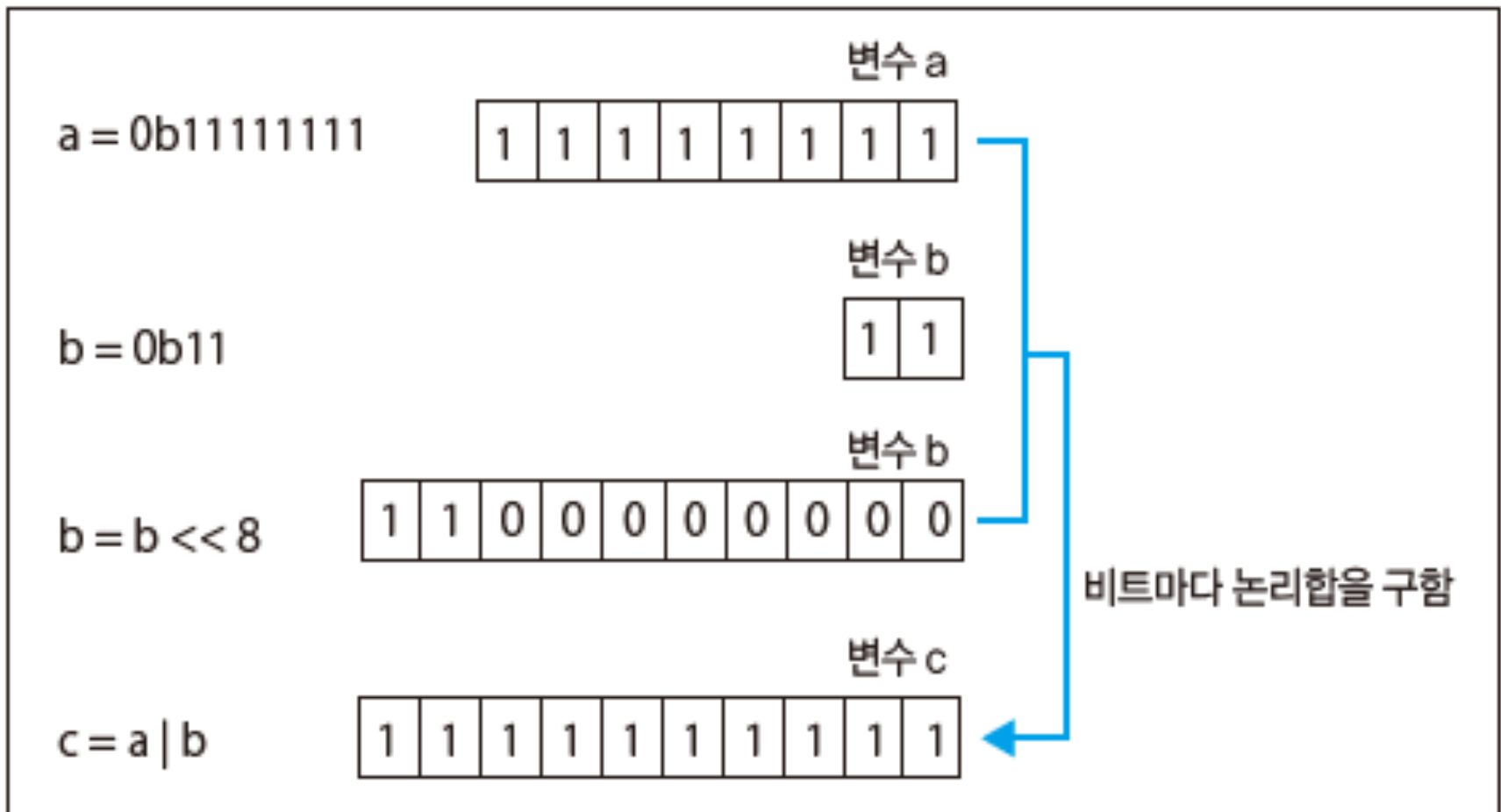
```
>>> c = a | b
```

```
>>> bin(c)
```

```
'0b1111111111'    → 최상위 2비트는 변수 b에서 온 데이터고, 하위 8비트가 변수 a에서 온 데이터
```

## 4. 숫자 값 계산

▼ 그림 3-28 비트 연산에 의한 데이터 가공 예제



## 5. 문자열

- 문자열
  - 문자열이란 하나 이상의 문자가 나열된 데이터임
  - 문자열 표시 : 작은따옴표(')나 큰따옴표(") 사용, 둘 다 사용 가능

▼ 예제 변수 a에 문자열 Python 할당하기

```
>>> a = 'Python'
>>> print(a)
Python
```

## 5. 문자열

- 변수에 문자열 할당하기
  - 작은따옴표로 감싼 문자열 안에 작은따옴표가 섞여 있으면 거기서 문자열이 끝나는 것으로 인식해서 오류가 발생함
  - 다른 따옴표라면 오류가 발생하지 않음
  - 즉, 작은따옴표로 감싼 문자열 안에 큰따옴표가 포함되어 있다면 오류가 발생하지 않음

▼ 예제 작은따옴표로 감싼 문자열에 큰따옴표가 포함되어 있음

```
>>> a = 'Happy "Python"'
>>> print(a)
Happy "Python"
```

## 5. 문자열

시퀀스 형태로 저장. 시퀀스형으로 문자열 사용

- 문자열을 인덱스로 지정하기

- 문자열처럼 요소들이 일정한 순서대로 나열된 데이터형을 파이썬에서는 **시퀀스형**이라 함
- 인덱스로 요소에 접근
- 파이썬 시퀀스형 : 문자열, 리스트, 튜플

▼ 그림 3-29 시퀀스형의 인덱스





## 5. 문자열

- 문자열을 인덱스로 지정하기

- 인덱스는 앞에서 세면 첫 번째 요소부터 0, 1, 2, ...
- 인덱스는 뒤에서 세면 마지막 요소부터 -1, -2, -3, ...
- 인덱스 번호는 대괄호([])로 표현

▼ 예제 변수 a에 문자열 Python을 할당하고 0번째와 마지막 문자 지정하기

```
>>> a = 'Python'
>>> a[0]
'P'
>>> a[-1]
'n'
```

## 5. 문자열

- 문자열을 인덱스로 지정하기
  - 정수형 숫자는 자릿수가 여러 개더라도 인덱스로 값을 추출할 수 없음

▼ 예제 정수형에 인덱스를 지정하면 오류가 발생함

```
>>> a = 1234
```

```
>>> a[1]
```

```
Traceback (most recent call last):
```

```
  File "<pyshell#54>", line 1, in <module>
```

```
    a[1]
```

```
TypeError: 'int' object has no attribute '__getitem__'
```

## 5. 문자열

### ● 여러 문자를 인덱스로 지정하기

- 파이썬 시퀀스형은 인덱스를 지정할 때 정수 x와 y를 콜론(:)으로 이어서 [x:y]처럼 작성하면 인덱스 x부터 y-1까지 범위에 있는 요소들을 한 번에 지정할 수 있음
- 이때 인덱스 y가 가리키는 요소는 포함되지 않으므로 주의함
- 이런 인덱스 지정 방법을 슬라이스라고 함

#### ▼ 예제 문자열 요소 0번째부터 1번째까지 지정하기

```
>>> a = '1234' → 문자열
>>> a[0:2]
'12'
```

#### ▼ 예제 시작 위치를 생략하면 0번째부터 지정한

```
>>> a[:2]
'12'
```

#### ▼ 예제 종료 위치를 생략하면 마지막까지 지정한

```
>>> a[2:] → 문자열 끝까지 (-1 x)
'34'
```

## 5. 문자열

- 따옴표(')와 백슬래시(\)를 문자열로 만들기
  - 문자열 내부에 문자열을 감싸는 따옴표와 같은 따옴표가 포함되면 파이썬은 거기서 문자열이 끝났다고 해석함
  - 이를 피하려면 이스케이프 시퀀스를 사용하거나 3중 따옴표로 감싸야 함

## 5. 문자열

- **이스케이프 시퀀스**를 사용하기
  - 따옴표 앞에 **백슬래시(w)**를 넣으면 문자열로 감싼 따옴표와 같은 따옴표를 문자열 안에 작성할 수 있음
  - 표 3-8에 정리한 것처럼 백슬래시 자체와 줄 바꿈 코드 등도 이스케이프 시퀀스로 작성할 수 있음

▼ 표 3-8 주요 이스케이프 시퀀스

이스케이프 시퀀스	의미
\\	백슬래시를 표시함
\'	작은따옴표를 표시함
\"	큰따옴표를 표시함
\n	줄 바꿈

## 5. 문자열

- 이스케이프 시퀀스를 사용하기

- ▼ 예제 작은따옴표로 감싼 문자열에 작은따옴표가 포함되면 오류가 발생

```
>>> a = 'Happy 'Python'  
SyntaxError: invalid syntax → 오류가 발생
```

- ▼ 예제 이스케이프 시퀀스로 문자열 안에 작은따옴표 작성하기

```
>>> a = 'Happy \'Python\''  
>>> print(a)  
Happy 'Python'
```

- ▼ 예제 문자열에 이스케이프 시퀀스로 줄 바꿈 코드를 넣기

```
>>> a = 'Happy \n Python'  
>>> print(a)  
Happy  
Python ←
```

## 5. 문자열

### ● 3중 따옴표로 감싸기

- 문자열을 3중 작은따옴표로 감싸면 도중에 줄 바꿈이나 따옴표가 포함되더라도 모두 그대로 문자열로 다룰 수 있음

▼ 예제 따옴표와 줄 바꿈이 포함된 문자열을 3중 따옴표로 감싸기

```
>>> a = ''' Happy  
Python' '''  
>>> print(a)  
Happy  
'Python'
```

## 5. 문자열

### ● 문자열 일부 교체하기

- 문자열 자료형의 내장 메서드인 `replace()`를 사용하면 문자열 일부를 지정한 문자열로 교체할 수 있음

`replace('교체하기 전의 문자열', '새롭게 넣을 문자열')`

객체(문자열).`replace(...)`

▼ 예제 변수 a에 문자열 'Python'을 할당하고 P를 p로 교체한 다음 다시 할당하기

```
>>> a = 'Python'
>>> a = a.replace('P', 'p')
>>> a
'python'
```



## 5. 문자열

- 연산자로 문자열 연결/반복하기

- + 연산자와 \* 연산자로 문자열을 연결하거나 반복할 수 있음

- ▼ 예제 문자열 연결하기

```
>>> a = 'Python'
>>> b = ' in Raspberry Pi'
>>> a + b
'Python in Raspberry Pi'
```

- ▼ 예제 문자열 반복하기

```
>>> a = 'Python'
>>> a * 3
'PythonPythonPython'
```

## 5. 문자열

- 글자 수 세기

- 문자열에 포함된 글자 수를 알아내려면 내장 함수 **len()** 사용

↘ 객체의 메서드가 X

```
len(문자열)
```

▼ 예제 문자열의 글자 수를 알아내기

```
>>> a = 'Python'
>>> len(a)
6
```

## 5. 문자열

- 특정 문자열이 포함되어 있는지 조사하기

- `in` 연산자
- `find()` 메서드

- `in` 연산자 사용하기

찾고 싶은 문자열 `in` 문자열 전체

▼ 예제 문자열 'Python'에 'Py' 또는 'pi'가 포함되어 있는지 확인하기

```
>>> a = 'Python'
>>> 'Py' in a → 대소문자 구분 x
True
>>> 'pi' in a
False
```

## 5. 문자열

- find() 메서드 사용하기

문자열 전체.find(찾고 싶은 문자열[, 시작 인덱스][, 종료 인덱스])

- 시작 인덱스와 종료 인덱스는 생략할 수 있음
- 문자열을 찾으면 찾은 문자열의 **인덱스를 반환**하고, 문자열을 찾지 못하면 **-1 반환**
- 찾고 싶은 문자열이 문자열 안에 여러 번 포함되어 있으면 **최초로 찾은 위치의 인덱스**를 돌려줌

▼ 예제 문자열 안에 'py'가 있는지 확인하기

```
>>> a = 'Python in Raspberry Pi'
>>> a.find('Py', 0, 3)      → 0번째와 3번째 사이에 'Py'가 있는지 확인
0
>>> a.find('Py', 2)         → 2번째와 마지막 사이에 'Py'가 있는지 확인
-1
```

## 6. 리스트

- 리스트 가변

- 리스트는 여러 숫자나 문자열 등의 요소를 일정한 순서로 나열한 구조의 자료형
- 리스트의 요소로 **각종 객체를 저장할 수도 있음**

[요소1, 요소2, 요소3, 요소4, 요소5, ...]

▼ 예제 변수 a에 리스트 1, 2, 3, 4를 할당하고 0번째와 마지막 문자를 지정하기

```
>>> a = [1, 2, 3, 4]
>>> a[0]
1
>>> a[-1]
4
```

## 6. 리스트

- 변수에 리스트를 할당해서 인덱스로 지정하기
  - 하나의 리스트에 서로 다른 자료형이 섞여 있어도 됨

▼ 예제 문자열, 정수, 부동소수가 담긴 리스트

```
>>> a = ['Python', 1234, 1.234]
>>> a[0]
'Python'
>>> a[1]
1234
>>> a[2]
1.234
```

## 6. 리스트

- 변수에 리스트를 할당해서 인덱스로 지정하기
  - **리스트에는 또 다른 리스트를 저장할 수도 있음**
  - 리스트에 들어 있는 또 다른 리스트의 요소를 인덱스로 지정하려면 다음과 같이 작성

▼ 예제 리스트의 3번째 요소인 자식 리스트의 첫(0번째) 요소를 지정하기

```
리스트[부모 리스트의 인덱스][자식 리스트의 인덱스]
```

▼ 예제 리스트의 3번째 요소인 자식 리스트의 첫(0번째) 요소를 지정하기

```
>>> a = [1, 2, 3, [4, 5, 6]]
```

```
>>> a[3][0]
```

```
4
```

## 6. 리스트

- 변수에 리스트를 할당해서 인덱스로 지정하기
  - 리스트는 **가변 데이터**이므로 **각 요소의 값을 자유롭게 변경**  
튜플은 불가능

▼ 예제 인덱스로 지정한 리스트의 요소 교체하기

```
>>> a = [1, 2, 3, 4]
>>> a[0] = 0
>>> a
[0, 2, 3, 4]
```



## 6. 리스트

- 여러 요소를 인덱스로 지정하기

[a:b]

- 문자열과 마찬가지로 리스트도 슬라이스를 통해 여러 요소를 한번에 지정할 수 있음

▼ 예제 리스트의 요소 0번째부터 1번째까지 지정하기

```
>>> a = [1, 2, 3, 4]
>>> a[0:2]
[1, 2]
```

▼ 예제 시작 위치를 생략하면 0번째부터 지정한 것이 됨

```
>>> a[:2]
[1, 2]
```

▼ 예제 종료 위치를 생략하면 마지막까지 지정한 것이 됨

```
>>> a[2:]
[3, 4]
```

## 6. 리스트

### ● 여러 요소를 인덱스로 지정하기

- 슬라이스를 사용하면 리스트의 여러 요소를 한 번에 교체
- 이때, 할당할 요소도 리스트 형식으로 작성

▼ 예제 0번째부터 1번째까지 요소를 다른 요소로 교체하기

```
>>> a = [1, 2, 3, 4]
>>> a[0:2] = [0, 0]
>>> a
[0, 0, 3, 4]
```

- 슬라이스로 지정한 요소 개수보다 할당한 리스트의 요소 개수가 많으면 그만큼 원래 리스트에 들어 있는 요소 개수가 늘어남

▼ 예제 0번째 요소부터 1번째 요소까지를 다른 요소로 교체하면서 요소 추가하기

```
>>> a = [1, 2, 3, 4]
>>> a[0:2] = [0, 0, 0]
>>> a
[0, 0, 0, 3, 4] → 리스트 늘어남
```

## 6. 리스트

### ● 연산자로 리스트 연결/반복하기

- 문자열과 마찬가지로 + 연산자와 \* 연산자로 리스트를 연결하거나 반복할 수 있음

#### ▼ 예제 리스트 연결하기

```
>>> a = ['(^o^)', '(^_^)']
>>> b = ['(;o;)', '(T^T)']
>>> a + b
['(^o^)', '(^_^)', '(;o;)', '(T^T)']
```

#### ▼ 예제 리스트를 3번 반복하기

```
>>> a = ['(^o^)', '(^_^)']
>>> a * 3
['(^o^)', '(^_^)', '(^o^)', '(^_^)', '(^o^)', '(^_^)']
```

## 6. 리스트

### ● 리스트에 요소 추가하기

- **append() 메서드**로 리스트 끝에 요소 한 개 추가하기
- 인수로 지정한 문자열이나 숫자를 리스트 끝에 추가

▼ 예제 리스트 끝에 요소 추가하기

```
>>> a = [1, 2, 3, 4]
>>> a.append(5)
>>> a
[1, 2, 3, 4, 5]
```

- **extend() 메서드** 사용해서 문자열이나 숫자를 리스트 끝에 추가
- append() 메서드와 다른 점은 **요소 여러 개 추가하기**

▼ 예제 리스트 끝에 요소 3개 추가하기

```
>>> a = [1, 2, 3, 4]
>>> a.extend([5, 6, 7])
>>> a
[1, 2, 3, 4, 5, 6, 7]
```

## 6. 리스트

내장함수 / 메서드 표준 정리하기

### ● 리스트의 요소 제거하기

- **del** 문으로 지정한 요소 제거하기

▼ 예제 del 문으로 리스트에 있는 요소 제거하기

```
>>> a = [1, 2, 3, 4]
>>> del a[0]
>>> a
[2, 3, 4]
```

↳ 슬라이스 사용가능

- **pop()** 메서드로 리스트의 마지막 요소 제거하기

▼ 예제 리스트의 마지막 요소 제거하기

```
>>> a = [1, 2, 3, 4]
>>> a.pop()
4
>>> a
[1, 2, 3]
```

## 6. 리스트

- 리스트의 요소를 역순으로 재배치하기

- `reverse()` 메서드를 사용해서 리스트의 요소를 역순으로 재배치

▼ 예제 리스트 a의 요소를 역순으로 나열하기

```
>>> a = ['(>_<)', '(+o+)', '(*_*)']  
>>> a.reverse()  
>>> a  
['(*_*)', '(+o+)', '(>_<)']
```

## 6. 리스트

- 리스트의 요소를 지정한 순서로 재배치하기

ASCII = 숫자 → 영어 → 기호

- 리스트를 특정 순서로 재배치하려면 `sort()` 메서드 사용

▼ 예제 리스트 요소 재배치하기(오름차순)

```
>>> a = ['3', '2', 'B', '1', 'a']  
>>> a.sort()  
>>> a  
['1', '2', '3', 'B', 'a']
```

- 반대로 재배치하려면 `reverse = True`를 인수로 지정함

▼ 예제 리스트 요소 재배치하기(내림차순)

```
>>> a.sort(reverse = True)  
>>> a  
['a', 'B', '3', '2', '1']
```

## 7. 튜플

- 튜플

- 시퀀스형 - 리스트, 문자열
- 튜플은 리스트와 비슷한 자료형이지만 불변이므로 일부 요소만 변경하거나 제거할 수 없음 → 전체 수정 or 삭제 가능
- 튜플에도 다양한 객체를 요소로 저장할 수 있음
- 전체를 소괄호(())로 둘러쌘

(요소1, 요소2, 요소3, 요소4, 요소5, ...)

▼ 예제 변수 a에 튜플을 할당하고 0번째와 마지막 요소 지정하기

```
>>> a = ('P', 'y', 't', 'h', 'o', 'n')
>>> a[0]
'P'
>>> a[-1]
'n'
```



## 7. 튜플

- 변수에 튜플을 할당하고 인덱스 지정하기

- 다음 예제에서는 문자열, 정수, 부동소수 이렇게 서로 다른 자료형의 데이터를 하나의 튜플에 저장함

▼ 예제 문자열, 정수, 부동소수가 담긴 튜플

```
>>> a = ('Python', 1234, 1.234)
>>> a[0]
'Python'
>>> a[1]
1234
>>> a[2]
1.234
```

▼ 예제 튜플에 튜플을 저장하기

```
>>> a = (1, 2, 3, (4, 5, 6))
>>> a[3][0]
4
```

## 7. 튜플

- 변수에 튜플을 할당하고 인덱스 지정하기

- 다음과 같이 여러 개의 변수에 튜플을 할당하면 각 변수에 튜플의 요소가 할당됨

▼ 예제 요소가 3개인 튜플을 세 변수에 할당하기

```
>>> a = (1, 2, 3)
>>> x, y, z = a → 요소 하나씩 할당가능
>>> x
1
>>> y
2
>>> z
3
```

## 7. 튜플

### ● 여러 요소를 인덱스로 지정하기

- 튜플도 슬라이스로 여러 요소를 지정할 수 있음
- 지정하는 방법은 문자열이나 리스트와 같음

▼ 예제 튜플의 0번째부터 1번째까지 요소 지정하기

```
>>> a = (1, 2, 3, 4)
>>> a[0:2]
(1, 2)
```

▼ 예제 시작 위치를 생략하면 0번째부터 지정한 것이 됨

```
>>> a[:2]
(1, 2)
```

▼ 예제 종료 위치를 생략하면 마지막까지 지정한 것이 됨

```
>>> a[2:]
(3, 4)
```

## 7. 튜플

### ● 연산자로 튜플 연결/반복하기

- 문자열이나 리스트처럼 + 연산자와 \* 연산자로 튜플을 연결하거나 반복할 수 있음

#### ▼ 예제 튜플 연결하기

```
>>> a = ('(^o^)', '^_^')
>>> b = ('(;o;)', '(T^T)')
>>> a + b
('(^o^)', '^_^', '(;o;)', '(T^T)')
```

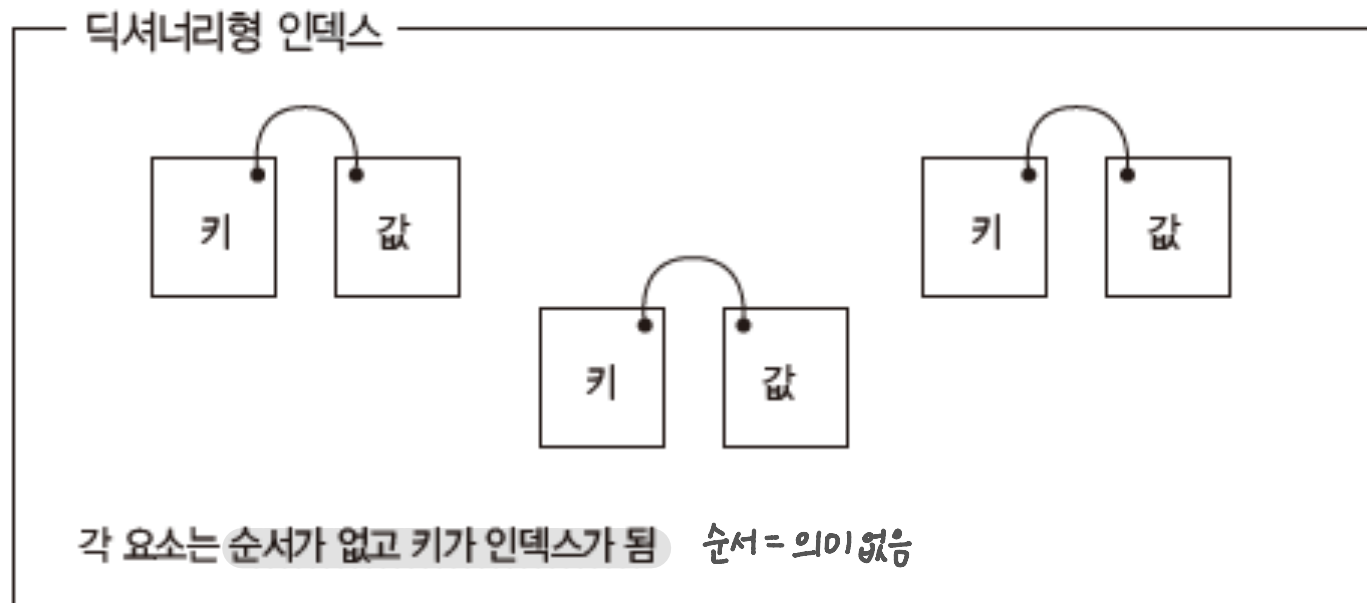
#### ▼ 예제 튜플을 3번 반복하기

```
>>> a = ('(^o^)', '^_^')
>>> a * 3
('(^o^)', '^_^', '^_^', '^_^', '^_^', '^_^')
```

## 8. 딕셔너리

- 딕셔너리 자료형

- 딕셔너리는 (키, 값) 쌍으로 이루어진 자료형임
- 값을 지정할 때는 짝으로 지정한 키를 사용함 → 인덱싱: 키 사용
- 딕셔너리의 값으로는 각종 객체를 지정할 수 있지만 키로는 문자열이나 숫자처럼 변하지 않는 자료형만 지정할 수 있음



## 8. 딕셔너리

- 변수에 딕셔너리를 할당해서 인덱스 지정하기
  - 딕셔너리를 정의할 때는 키와 값을 콜론(:)으로 묶음
  - 키와 값 쌍을 쉼표(,)로 구분해서 전체를 중괄호({})로 감쌘

```
{키1:값1, 키2:값2, 키3:값3, ...}
```

- 키가 이모티콘을 설명하는 문자열이고 값이 이모티콘인 딕셔너리를 정의

▼ 예제 변수 a에 딕셔너리를 할당하고 키로 값을 지정하기

```
>>> a = {'happy': '^_^', 'sad': '(ToT)'}  
>>> a['happy']  
'^_^'  
>>> a['sad']  
'(ToT)'
```

## 8. 딕셔너리

- 변수에 딕셔너리를 할당해서 인덱스 지정하기
  - 키와 값은 자료형이 서로 달라도 됨
    - ▼ 예제 키가 숫자고 값이 문자열인 딕셔너리

```
>>> a = {1:'(^_^)', 2:'(ToT)'}  
>>> a[1]  
'(^_^)'
```

- 딕셔너리에는 또 다른 딕셔너리를 저장할 수도 있음
- 이때 딕셔너리 안에 있는 딕셔너리의 값은 다음과 같이 지정함

```
딕셔너리명[부모 딕셔너리의 키][딕셔너리의 키]
```

## 8. 딕셔너리

- 변수에 딕셔너리를 할당해서 인덱스 지정하기
  - 다음은 자식이 된 딕셔너리를 지정하는 예제
    - ▼ 예제 자식 딕셔너리의 값 지정하기

```
>>> a = {'happy':{1:'(^_^)', 2:'(^o^)'}, 'sad':{1:'(ToT)', 2:'(T-T)'}}  
>>> a['sad'][1]  
'(ToT)'
```

- 딕셔너리는 **가변 데이터**이므로 정의한 다음에도 각 값을 바꿀 수 있음
  - ▼ 예제 인덱스로 지정한 값 바꾸기

```
>>> a = {'happy':'(^_^)', 'sad':'(ToT)'}  
>>> a['happy'] = '^o^'  
>>> a['happy']  
'(^o^)'
```



## 8. 딕셔너리

### ● 딕셔너리에 요소 추가하기

- 딕셔너리에 키와 값을 추가하려면 새로운 키를 지정해서 값을 할당하면 됨
- 딕셔너리는 시퀀스형이 아니므로 요소는 무작위로 나열됨

#### ▼ 예제 딕셔너리에 요소 추가하기

```
>>> a = {'happy': '^_^', 'sad': '(ToT)'}  
>>> a['sleepy'] = '(_-_)'  
>>> a  
{'happy': '^_^', 'sad': '(ToT)', 'sleepy': '(_-_)'}
```

## 8. 덕셔너리

## ● 디렉셔리의 요소 제거하기

- **del 문**으로 지정한 키워드 값 제거하기

▼ 예제 del 문으로 키와 값 제거하기

```
>>> a = {'happy': '^_^', 'sad': '(ToT)'}
>>> del a['sad']
>>> a
{'happy': '^_^'}
```

- **pop()** 메서드로 키와 값 제거하기

### ▼ 예제 pop() 메서드로 키와 값 제거하기

```
>>> a = {'happy': '^_^', 'sad': '(ToT)'}
>>> a.pop('happy') → 요소들이 순서가 없기 때문에
                     키값을 파라미터로 지정해야 함
'^_^'
>>> a
{'sad': '(ToT)'}
```

## 9. 다른 자료형으로 변환하기

### ● 자료형 변환하기

- 문자열인 숫자를 계산하고 싶을 때
- 어떤 함수의 인수로 특정 자료형의 데이터를 넘겨야 할 때

▼ 표 3-9 자료형을 변환하는 함수

내장 함수	내용	인수로 지정 가능한 자료형
<code>float()</code>	부동소수형으로 변환	정수, 문자열
<code>int()</code>	정수형으로 변환	부동소수, 문자열
<code>str()</code>	문자열로 변환	숫자, 리스트, 튜플 등의 객체
<code>list()</code>	리스트로 변환	문자열, 튜플 등
<code>tuple()</code>	튜플로 변환	문자열, 리스트 등

▼ 예제 `float()`으로 정수형을 부동소수형으로 변환하기

```
>>> a = 1
>>> b = float(a)
>>> b
1.0
```

## 9. 다른 자료형으로 변환하기

- 자료형 변환하기

- int()로 부동소수형을 정수형으로 변환하면 소수점 이하는 버려짐

▼ 예제 int()로 부동소수형을 정수형으로 변환하기

```
>>> a = 1.0  
>>> b = int(a)  
>>> b  
1
```

▼ 예제 str()로 정수형을 문자열로 변환하기

```
>>> a = 1.234  
>>> str(a)  
'1.234'
```

## 9. 다른 자료형으로 변환하기

- 자료형 변환하기

- 다음은 문자열을 정수형으로 변환하는 예제임

- ▼ 예제 int()로 문자열을 정수형으로 변환하기

```
>>> a = '1234'  
>>> int(a)  
1234
```

- 문자열로 된 숫자를 연산하려면 연산하기 전에 문자열을 숫자 값으로 변환해야 함

- ▼ 예제 문자열을 숫자 값으로 변환해서 계산하고, 숫자 값을 다시 문자열로 되돌려서 변수 c에 할당하기

```
>>> a = '3'  
>>> b = 5  
>>> c = str(int(a) + b)  
>>> c  
'8'
```

## 9. 다른 자료형으로 변환하기

- 자료형 변환하기

- 다음으로 list()와 tuple()을 사용해서 튜플을 리스트로 변환함
- 리스트를 튜플로 변환함

▼ 예제 튜플을 리스트로 변환해 요소를 변경하고, 리스트를 다시 튜플로 되돌리기

```
>>> a = (1,2,3)
>>> b = list(a)
>>> b[0] = 0
>>> a = tuple(b)
>>> a
(0, 2, 3)
```

## 9. 다른 자료형으로 변환하기

- 변수나 데이터가 어떤 자료형인지 확인하기

- 내장 함수인 `isinstance()`를 사용하면 변수나 데이터가 특정 자료형인지 확인할 수 있음

`isinstance(객체, 자료형의 명칭)`

▼ 표 3-10 `isinstance()`의 두 번째 인수로 지정할 수 있는 자료형

자료형	명칭
정수형	<code>int</code>
부동소수형	<code>float</code>
문자열	<code>str</code>
리스트	<code>list</code>
튜플	<code>tuple</code>
딕셔너리	<code>dict</code>

▼ 예제 변수가 `float` 형인지 `int` 형인지 확인하기

```
>>> a = 1.0
>>> isinstance(a, float)
True
>>> isinstance(a, int)
False
```

# 10. 문자열 서식 지정

시험X

## ● 문자열 서식 지정

- 변수를 지정한 문자열 형식에 따라 표현
- 문자열 서식 지정을 하려면 **% 연산자**를 사용하거나 **format() 메서드**를 사용함

▼ 예제 소수점 이하 자릿수를 두 자리로 지정해서 표시하기

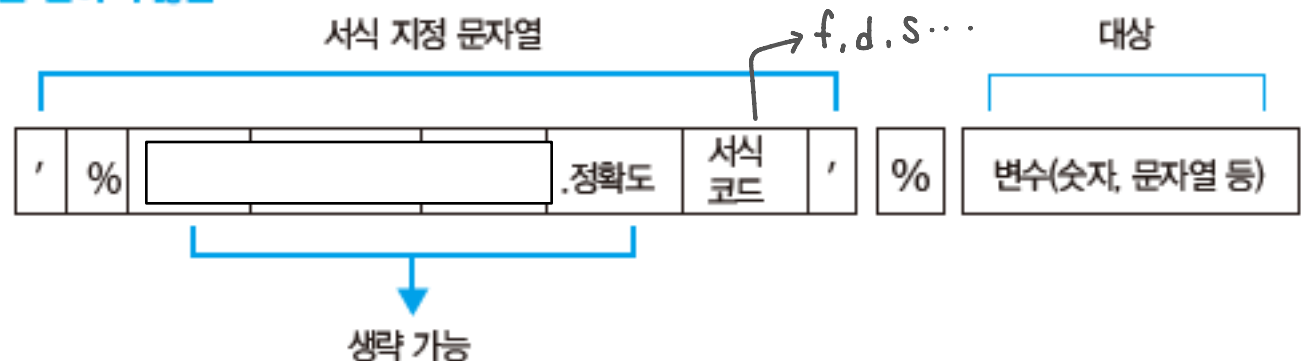
```
>>> a = 10.1234
```

```
>>> print('%0.2f' % a)
```

10.12 → 소수점 세 번째 자리 이하는 버림

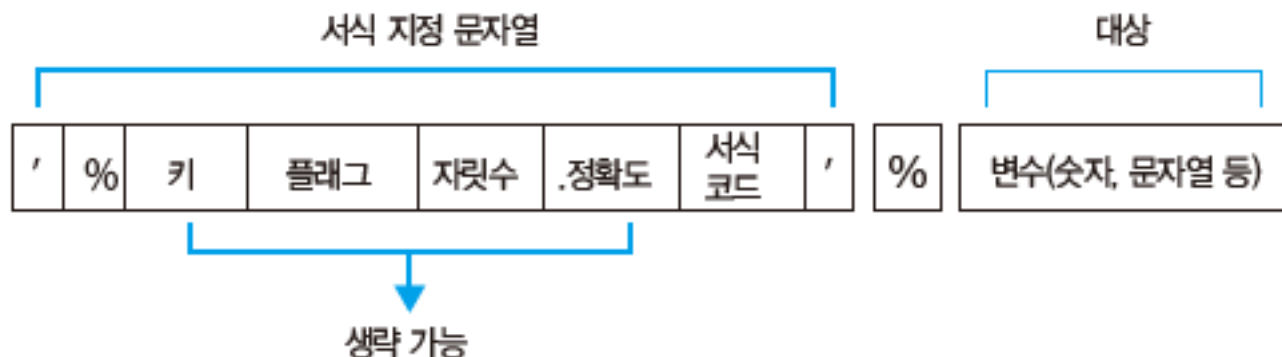
```
>>> a
```

10.1234 → 변수 a의 값은 변하지 않음





## 10. 문자열 서식 지정



### ● 키

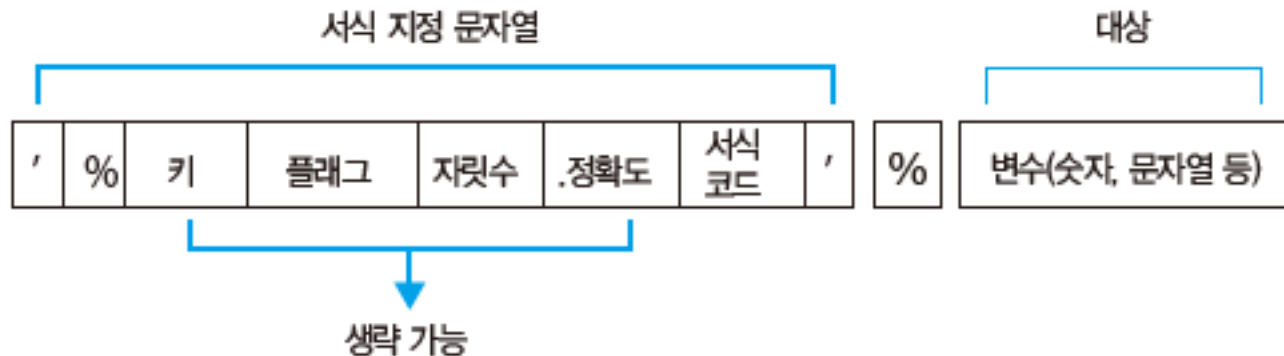
- 서식 지정 대상이 딕셔너리일 때 (키)처럼 작성하면 해당 값을 지정할 수 있음

### ● 플래그

▼ 표 3-11 % 연산자에서 사용할 수 있는 플래그

플래그	내용
0	대상이 숫자면 빈 자리를 0으로 채움
-	왼쪽 정렬
(공백)	양수 앞에 공백 넣기
+	양수 앞에 + 넣기
#	서식 코드 옵션(표 3-12 참조)

## 10. 문자열 서식 지정



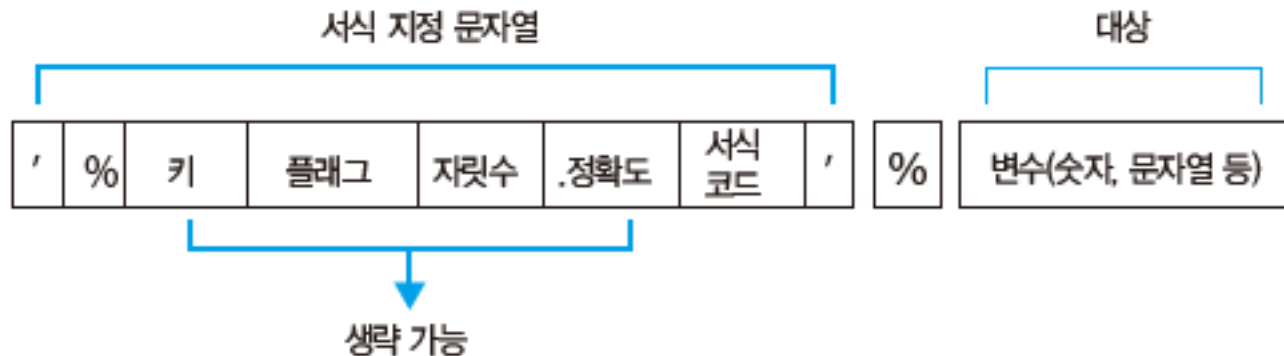
- 자릿수

- 정수 부분 전체 자릿수
- 지정한 자릿수가 6인데 표시할 문자열이나 숫자의 자릿수가 4라면 왼쪽에서부터 두 자리를 공백으로 채움

- 정확도

- 정수로 소수점 이하의 자릿수를 지정할 수 있음

## 10. 문자열 서식 지정



### ● 서식 코드

▼ 표 3-12 주요 서식 코드

기호	의미
s	대상 객체를 문자열로 처리
c	숫자를 대응하는 문자열로 변환
d	숫자를 부호가 있는 10진수로 표시
o	숫자를 부호가 있는 8진수로 표시 플래그에 #을 지정하면 앞부분에 0이 붙음
x	숫자를 부호가 있는 16진수로 표시 플래그에 #을 지정하면 앞부분에 0이 붙음
f	숫자를 고정소수로 표시 플래그에 #을 지정하면 정확도가 0이더라도 소수가 표시됨

## 10. 문자열 서식 지정

- 서식 코드

- ▼ 예제 변수 a, b를 변수 msg에 문자열로 할당하기

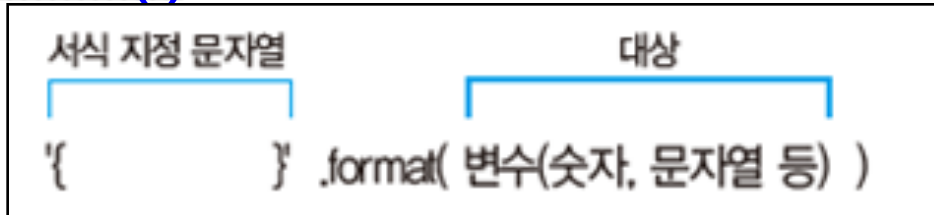
```
>>> a = 7
>>> b = 'Python'
>>> msg = '%s wonders of the %s' % (a, b)   → 여러 변수를 할당하려면 괄호로 감싸기
>>> msg
'7 wonders of the Python'
>>> a
7      → 변수 a의 값은 변하지 않음
```

- ▼ 예제 소수점 이하 자릿수를 두 자리로 지정해서 표시하기

```
>>> a = 10.1234
>>> print('%.2f' % a)
10.12      → 소수점 세 번째 자리 아하는 버림
>>> a
10.1234    → 변수 a의 값은 변하지 않음
```

## 10. 문자열 서식 지정

- **format() 메서드 사용하기**



▼ 예제 format() 메서드로 여러 변수 넣기

```
>>> a = "Raspberry Pi"
>>> b = "Python"
>>> c = "in"
>>> '{1} {2} {0}'.format(a, b, c)
'Python in Raspberry Pi'
```

▼ 예제 소수점 이하의 자릿수를 지정해서 표시하기

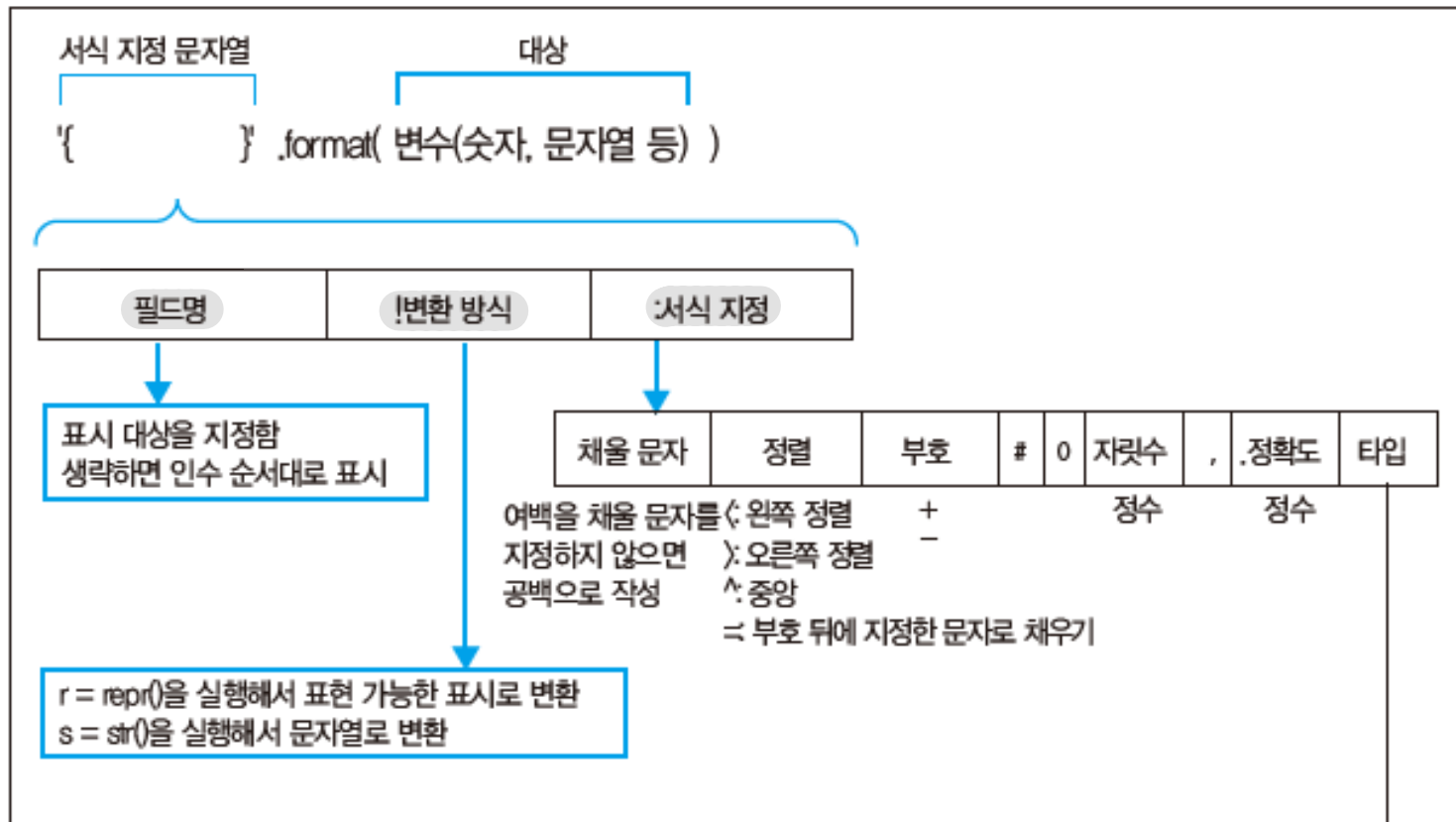
```
>>> a = 1.2345
>>> b = 4.4444
>>> 'a = {:.2f}, b = {:.3f}'.format(a, b)
'a = 1.23, b = 4.444'
```

변수들의 순서를 지정할 때

→ 지정한 자릿수 아래를 버린 값이 표시되지만 변수 a, b 값은 변하지 않음

# 10. 문자열 서식 지정

## ▼ 그림 3-32 format() 메서드



## 10. 문자열 서식 지정

b: 정수를 2진수로 표시	e: 숫자를 지정한 지수로 표시. 지수를 뜻하는 e는 소문자
o: 정수를 8진수로 표시	E: 숫자를 지정한 지수로 표시. 지수를 뜻하는 E는 대문자
d: 정수를 10진수로 표시	f: 숫자를 고정소수로 표시
n: 숫자에 구분 문자를 삽입	F: 숫자를 고정소수로 표시. 알파벳은 대문자
x: 숫자를 16진수로 표시	g: 정확도에 1 이상의 숫자를 지정하면 지정한 정확도에 따라 숫자를 올림 처리해서 자릿수에 따라 고정소수 또는 지수로 표시
X: 숫자를 16진수로 표시, 알파벳은 대문자	G: g와 같으나 알파벳을 대문자로 표시
c: 숫자에 대응하는 유니코드를 표시	%: 숫자를 100배 해서 고정소수 f로 표기

## 11. 한글 문자 처리법

- 한글 문자 처리법
  - 파이썬 3은 표준 문자 코드가 UTF-8임
  - 파이썬 2에서 필요했던 인코딩 지정이 필요 없음

### ▼ 코드 3-1 korean.py

```
# 문자열
a = '라즈베리'
print(a)

# 리스트
b = ['딸기', '복숭아', '사과']

print(b[0])
print(b[1])
print(b[2])

# 딕셔너리
c = {'기쁨': '(*>_<*)', '슬픔': '(T_T)'}
print(c['기쁨'])
print(c['슬픔'])
```

### ▼ 실행결과

```
라즈베리
딸기
복숭아
사과
(*>_<*)
(T_T)
```



## 12. 키보드에서 입력받는 방법

- 키보드에서 입력받는 방법

- 내장 함수 `input()`을 사용하면 키보드로 입력한 것을 문자열로 받을 수 있음

▼ 코드 3-2 sample\_input.py

```
name = input('input your name >')  
print('Hello! ' + name + ' (*^_^)')
```

```
>>>  
input your name >eunmi  
Hello! eunmi (*^_^)
```

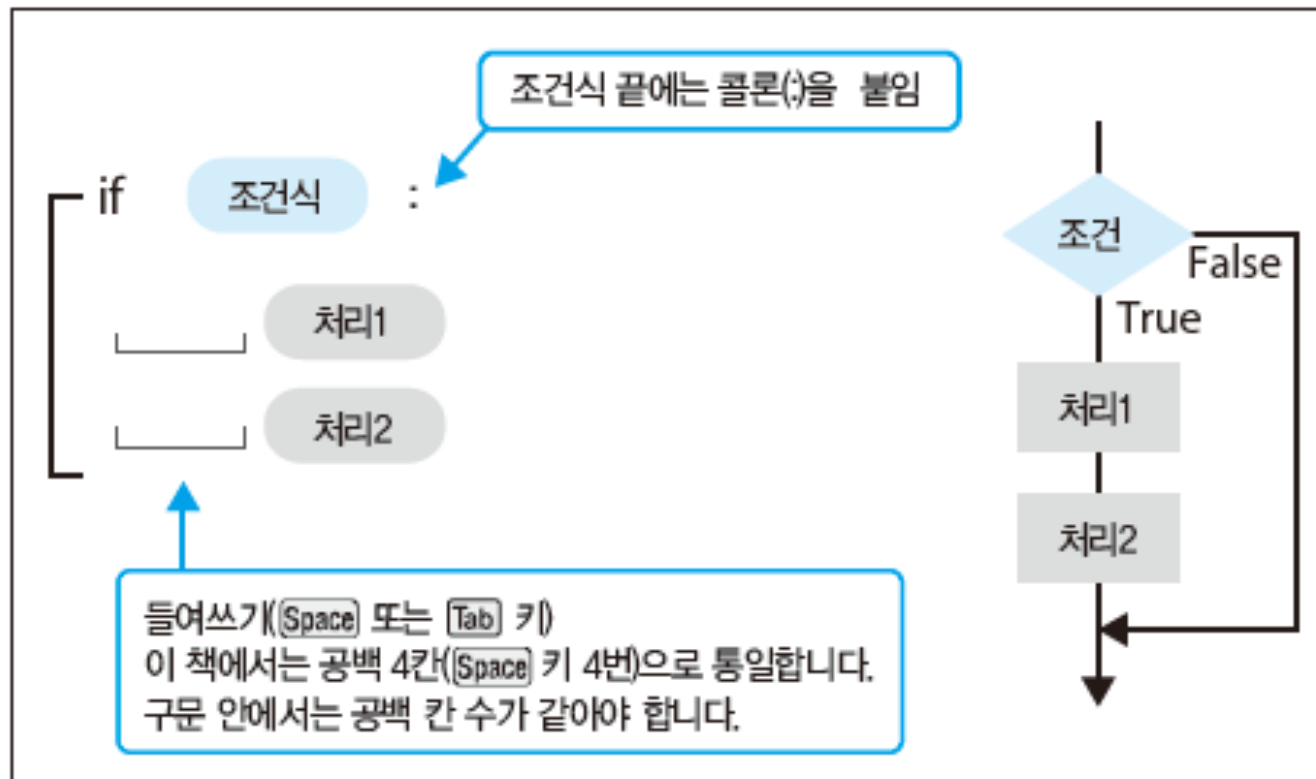
## 5. 상황에 따른 처리

# 1. 조건에 따라 처리 나눠 보기(if 문)

## ● if 조건문

- if 조건식 뒤에 콜론(:)을 붙이고
- 처리문 앞에는 반드시 들여쓰기를 해야 함

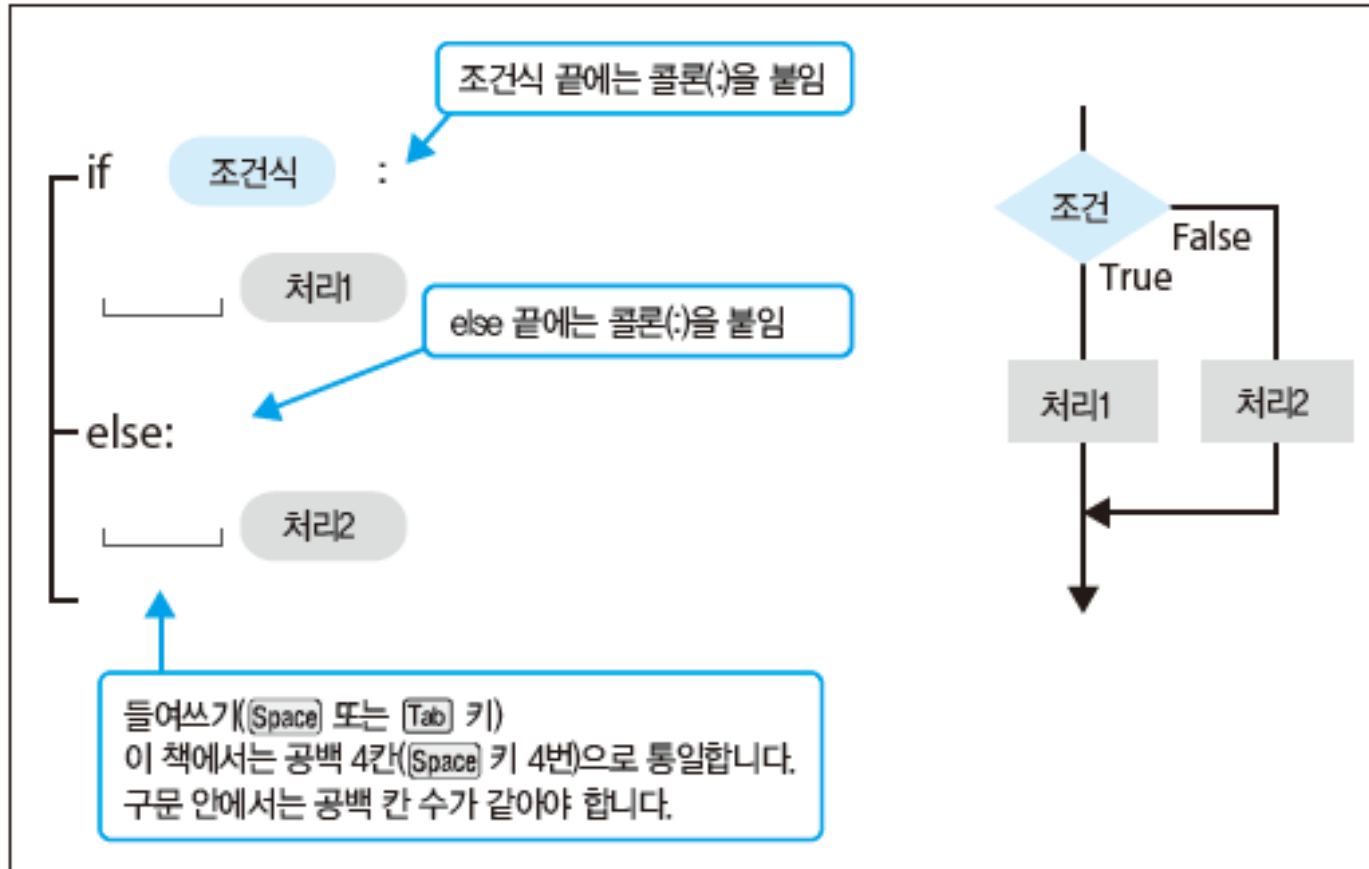
▼ 그림 3-35 if문 사용법



# 1. 조건에 따라 처리 나눠 보기(if 문)

## ● if else 조건문

- if 문에서 조건식을 만족하지 않을 때는 else에 속한 처리를 실행함
- ▼ 그림 3-36 else 사용법



# 1. 조건에 따라 처리 나눠 보기(if 문)

## ● if else 조건문 예제

▼ 예제 변수 a가 True면 문자열 True를 돌려주고 False면 문자열 False를 돌려줌

```
>>> a = 0
>>> if a:                → 줄 바꿈 ①
    print('True')        → 줄 바꿈 ②
else:                    → 들여쓰기를 지우고 else:를 입력한 후 줄 바꿈 ③
    print('False')       → 줄 바꿈 ④
                        → 줄 바꿈 ⑤
False                   → 실행 결과가 표시됨 ⑥
```

▼ 예제 not 연산자를 붙이면 조건이 반전됨

```
>>> a = 0
>>> if not a:
    print('True')
else:
    print('False')

True
```

# 1. 조건에 따라 처리 나눠 보기(if 문)

## ● if else 조건문 예제

▼ 예제 변수 a가 정수면 문자열 True 출력

```
>>> a = 0
>>> if isinstance(a, int):
    print('True')
else:
    print('False')
```

True

▼ 예제 변수 a 값이 5면 문자열 True 출력

```
>>> a = 5
>>> if a == 5:
    print('True')
else:
    print('False')
```

True

# 1. 조건에 따라 처리 나눠 보기(if 문)

- 조건식 비교 연산자

- 값을 비교하는 연산자를 비교 연산자라고 함

▼ 표 3-13 파이썬의 비교 연산자

비교 연산자	설명
<code>x == y</code>	x와 y가 같으면 True, 그렇지 않으면 False
<code>x != y</code>	x와 y가 같지 않으면 True, 같으면 False
<code>x &lt; y</code>	x가 y보다 작으면 True, 크거나 같으면 False
<code>x &gt; y</code>	x가 y보다 크면 True, 작거나 같으면 False
<code>x &lt;= y</code>	x가 y보다 작거나 같으면 True, 크면 False
<code>x &gt;= y</code>	x가 y보다 크거나 같으면 True, 작으면 False
<code>x in y</code>	x가 y의 요소(문자열, 리스트, 튜플 등)면 True, 요소가 아니면 False
<code>x not in y</code>	x in y의 반대. x가 y에 포함되어 있으면 False, 포함되어 있지 않으면 True

# 1. 조건에 따라 처리 나눠 보기(if 문)

- 조건식 비교 연산자

▼ 예제 != 사용(~가 아니면)

```
>>> a = 5
>>> if a != 4:      → 변수 a가 4가 아니면 True
    print('True')
else:
    print('False')

True
```



# 1. 조건에 따라 처리 나눠 보기(if 문)

- 조건식 비교 연산자

- ▼ 예제 < 사용(~보다 작으면)

```
>>> a = 5
>>> if a < 6:    → 변수 a가 6보다 작으면 True
    print('True')
else:
    print('False')

True
```

- ▼ 예제 <= 사용(~ 이하면)

```
>>> a = 5
>>> if a <= 5:   → 변수 a가 5보다 작거나 같으면 True
    print('True')
else:
    print('False')

True
```

# 1. 조건에 따라 처리 나눠 보기(if 문)

- 조건식 비교 연산자

- ▼ 예제 ~가 포함되어 있으면

```
>>> a = ['(^-^)', '(^o^)', '(^_^)']
>>> if '(^-^)' in a:      → 지정한 문자열이 포함되어 있으면 True
    print('True')
else:
    print('False')

True
>>> if '(;_;) ' in a:     → 지정한 문자열이 포함되어 있지 않으면 False
    print('True')
else:
    print('False')

False
```

# 1. 조건에 따라 처리 나눠 보기(if 문)

- if 문 처리 범위
  - if 문의 처리문은 들여쓰기에 따라 어디까지가 if 문의 처리문인지 정해짐
  - 처리문 앞에 들여쓰기가 없으면 if 문 밖에 있다고 처리되어 if 문과 상관없이 늘 실행됨

▼ 코드 3-3 sample\_if.py

```
a = input('input a number>')  
if a == '5':  
    print('your number is 5')  
print('end of program')
```

# 1. 조건에 따라 처리 나눠 보기(if 문)

- 여러 조건식 판단하기

- 논리 연산자를 이용해 if 문의 조건식에 조건을 여러 개 넘겨보자

▼ 예제 논리곱

```
>>> a = 5
>>> if a > 3 and a < 5:    → 변수 a가 3보다 크고 5보다 작으면 True 출력
    print('True')
else:
    print('False')          → 두 조건 중 하나라도 만족하지 않으면 False 출력

False                        → 5보다 작지 않으므로 False
```

▼ 예제 논리합

```
>>> a = 5
>>> if a > 3 or a < 5:    → 변수 a가 3보다 크거나 5보다 작으면 True 출력
    print('True')
else:
    print('False')          → 두 조건 중 하나라도 만족하면 True 출력, 둘 다 만족하지 않으면 False 출력

True                        → 5보다 작진 않지만 3보다는 크므로 True
```

# 1. 조건에 따라 처리 나눠 보기(if 문)

- 조건식 우선순위
  - and와 or 중에서는 and가 우선순위가 높음.

## ▼ 예제 and와 or의 우선순위 1

```
>>> a = 2
>>> if a > 1 or a == 0 and a == 1:
    print('True')
else:
    print('False')
```

만저처리

True

## ▼ 예제 and와 or의 우선순위 2

```
>>> a = 2
>>> if (a > 1 or a == 0) and a == 1:
    print('True')
else:
    print('False')
```

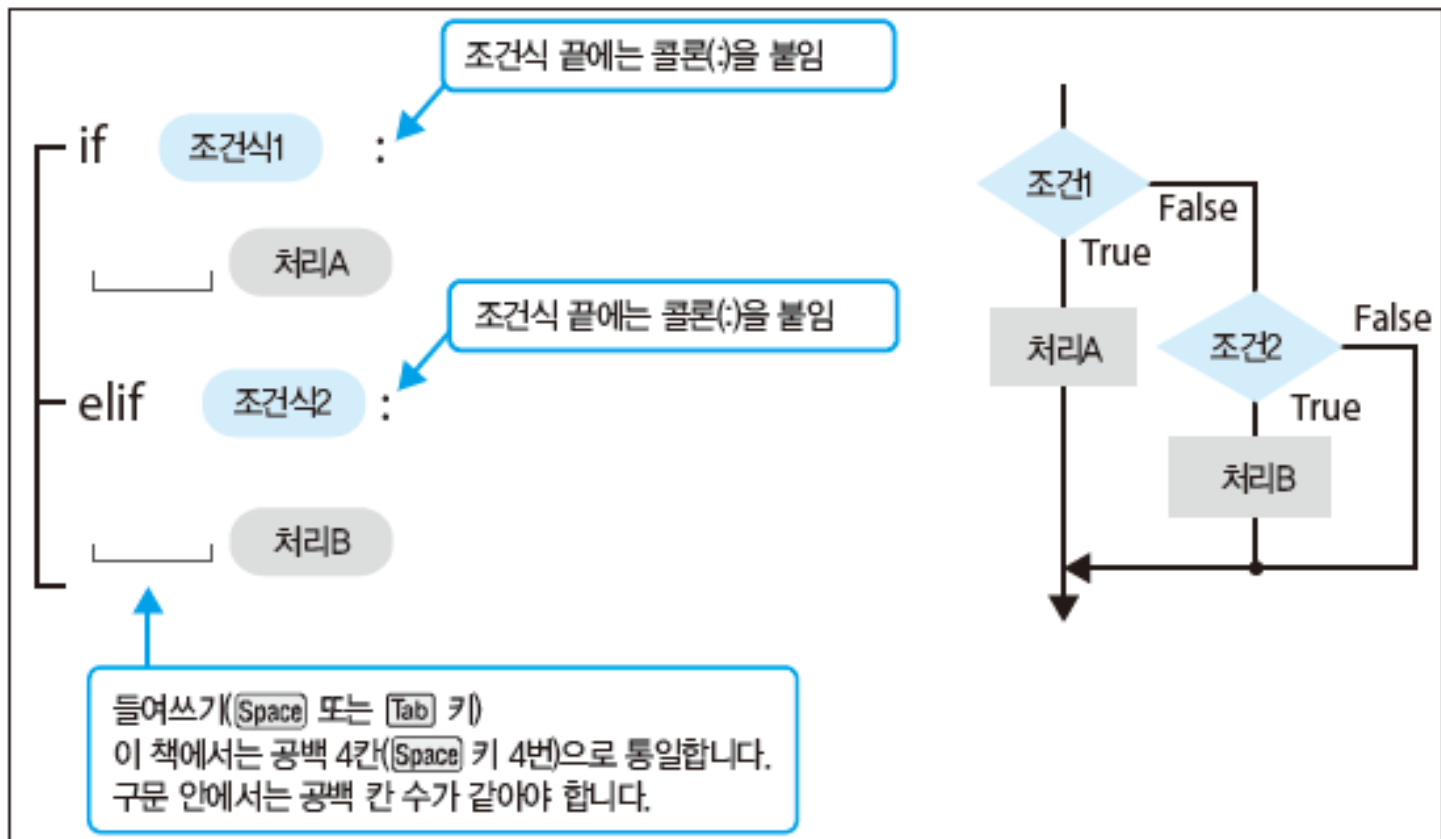
False

# 1. 조건에 따라 처리 나눠 보기(if 문)

- **elif**로 조건식을 여러 개 작성하기

- else와 if를 결합한 elif를 사용하면 if 문의 조건식에 해당하지 않을 때 새 조건식을 추가해서 처리할 수 있음

▼ 그림 3-39 elif 사용법



# 1. 조건에 따라 처리 나눠 보기(if 문)

- elif로 조건식을 여러 개 작성하기
  - 하나의 if 문에 elif를 여러 개 작성할 수도 있고 else와 조합해서 사용할 수도 있음

▼ 코드 3-4 sample\_if\_elif\_else.py

```
a = input('input a number>')
a = int(a)
if a < 5:
    print('your number is smaller than 5')
elif a < 10:
    print('your number is smaller than 10')
else:
    print('your number is greater than or equal to 10')

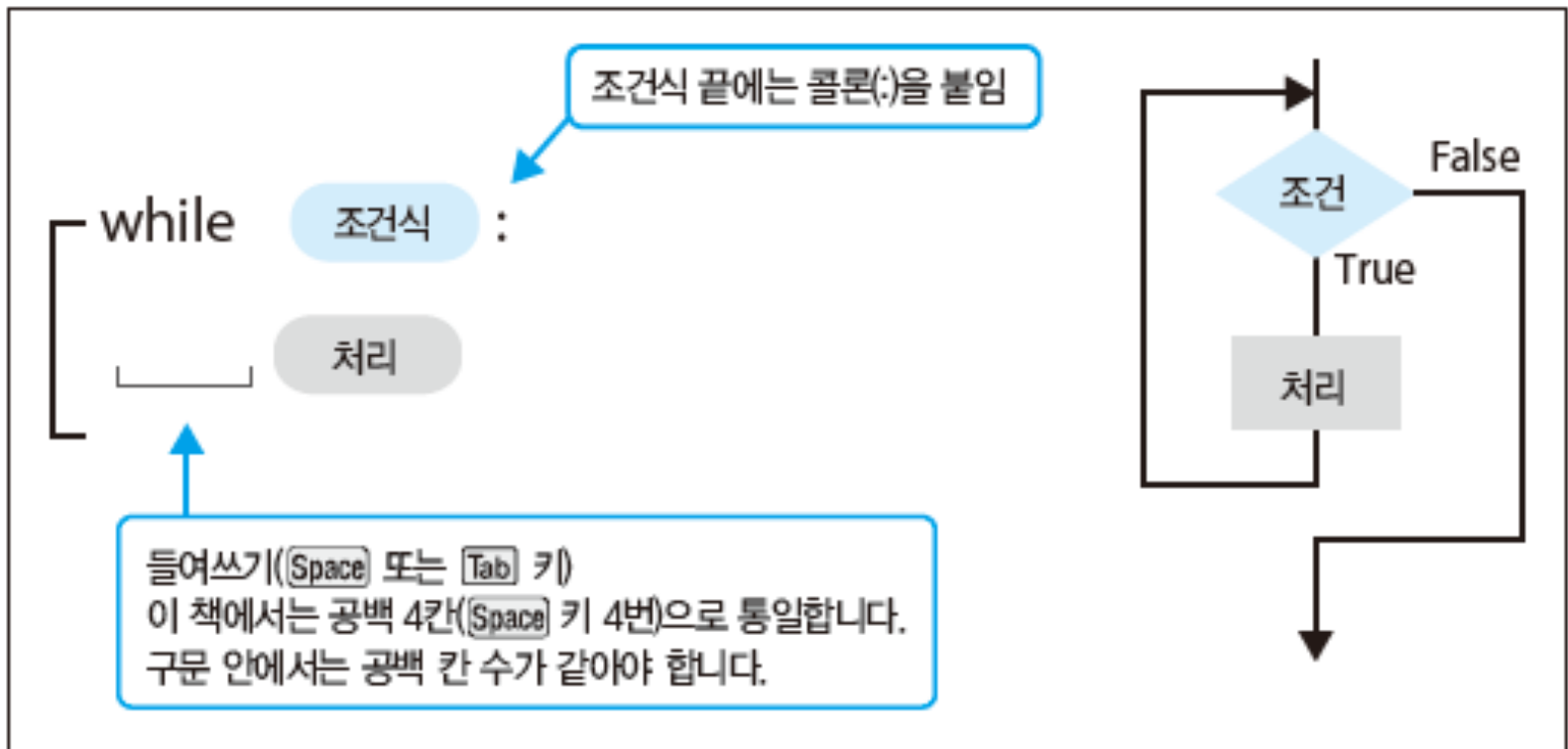
print('end of program')
```

## 2. 처리 반복하기(while 문)

- While 반복문

- while 문은 조건식이 성립하는 동안(조건식이 True인 동안) 처리를 반복하는 구문임

▼ 그림 3-41 while 문 사용법





## 2. 처리 반복하기(while 문)

▼ 코드 3-5 sample\_while.py

```
a = 0
while a < 3:
    a = a + 1
    print('(-.-)zzZ')

print('end of program')
```

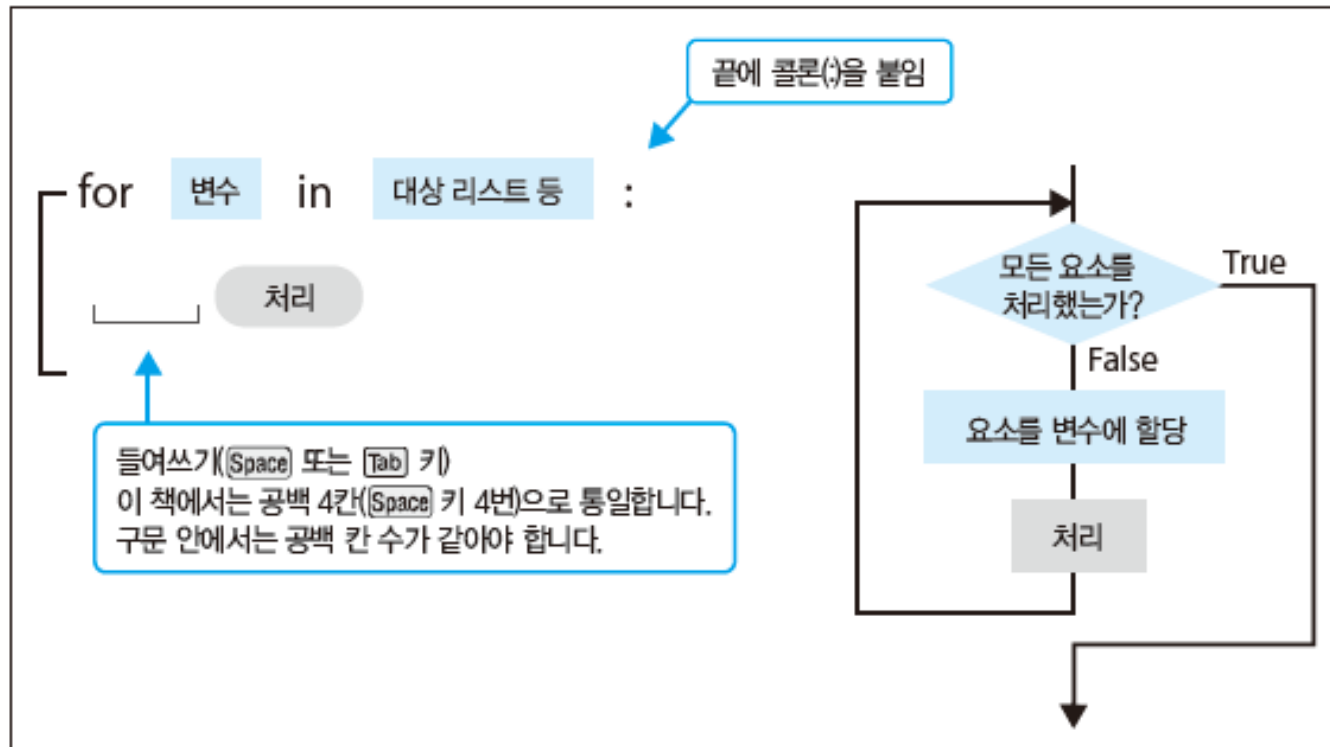
실행 화면

```
>>>
(-.-)zzZ
(-.-)zzZ
(-.-)zzZ
end of program
```

## 3. 시퀀스 자료형의 처리 반복하기(for ~ in 문)

- 시퀀스 자료형의 처리 반복하기(for ~ in 문)
  - for ~ in 문은 문자열, 리스트, 튜플 등 일정한 순서대로 나열된 시퀀스형 객체를 요소의 처음부터 끝까지 하나씩 꺼내서 순서대로 처리할 때 사용함

▼ 그림 3-43 for~in 문 사용법



## 3. 시퀀스 자료형의 처리 반복하기(for ~ in 문)

- 시퀀스 자료형의 처리 반복하기(for ~ in 문)
    - 리스트의 각 요소를 하나씩 표시하고 마지막으로 end of program을 출력하는 예제
- ▼ 코드 3-6 sample\_for\_01.py

```
for i in ['(^o^)', '(^-^)', '(-.-)zzZ']:  
    print(i)  
  
print('end of program')
```

실행 화면

```
>>>  
(^o^)  
(^-^)  
(-.-)zzZ  
end of program
```

## 3. 시퀀스 자료형의 처리 반복하기(for ~ in 문)

- 시퀀스 자료형의 처리 반복하기(for ~ in 문)
  - range()는 for 문과 조합해서 자주 사용하는 내장 함수임
  - range() 함수는 등차수열 리스트를 생성하는 함수로 for 문과 조합해서 사용하면 다음과 같이 동작함

range(x)	→ 0부터 $x-1$ 까지 리스트를 작성해서 그것으로 처리를 반복합니다.
range(x, y)	→ x부터 y-1까지 리스트를 작성해서 그것으로 처리를 반복합니다.
range(x, y, z)	→ x부터 y-1 범위에서 z 단위로 리스트를 작성해서 그것으로 처리를 반복합니다.

### ▼ 코드 3-7 sample\_for\_02.py

```
for i in range(3):  
    print('(-.-)zzZ')  
  
print('end of program')
```

## 3. 시퀀스 자료형의 처리 반복하기(for ~ in 문)

### ▼ 코드 3-8 sample\_for\_03.py

```
a = ('P', 'y', 't', 'h', 'o', 'n')  
for i in range(0, len(a), 2):  
    print(a[i])  
print('end of program')
```

↗ 0 ~ len(a)-1 까지 2의 단위로  
0 2 4 ...

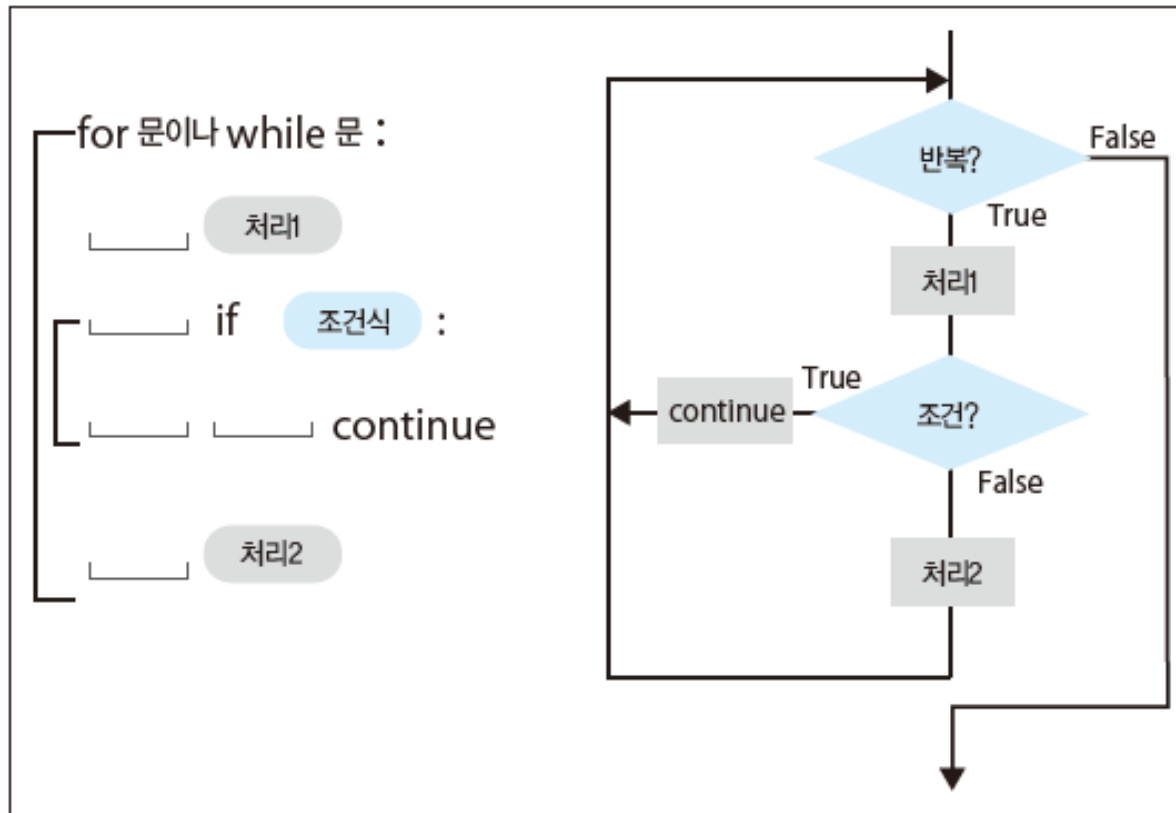
### ▼ 실행화면

```
>>>  
P  
t  
o  
end of program
```

## 4. 반복 처리 종류

- **if ~ continue** 문과 **if ~ break** 문으로 반복 흐름 바꾸기
  - continue는 반복 처리 도중에 그 이후의 처리를 멈추고 다음 반복 처리로 넘어가게 함

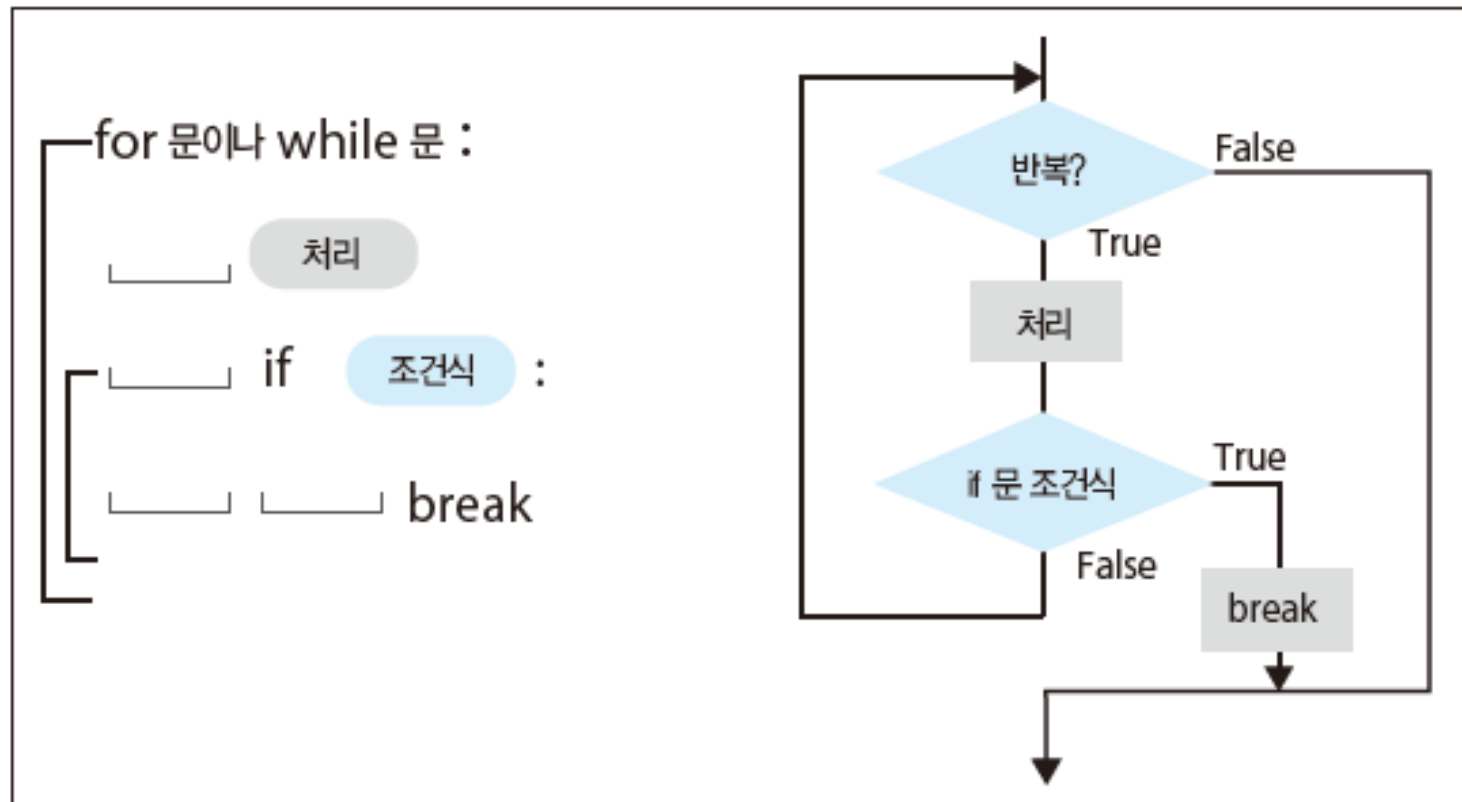
▼ 그림 3-46 if~continue 문의 반복 처리 흐름



## 4. 반복 처리 종류

- **if ~ continue** 문과 **if ~ break** 문으로 반복 흐름 바꾸기
  - break를 사용하면 for 문이나 while 문의 반복 처리에서 즉시 빠져나옴

▼ 그림 3-47 if~break 문의 반복 처리 흐름



## 4. 반복 처리 종류

- **if ~ continue** 문과 **if ~ break** 문으로 반복 흐름 바꾸기
  - 다음은 continue와 break를 사용한 예제
    - ▼ 코드 3-9 sample\_continue\_break.py

```
while True:
    a = input('input a number>')
    a = int(a)
```

```
    if a <= 7:
        continue
    if a > 7:
        break
```

```
print('end of program')
```

실행 화면

```
>>>
Input a number>5
Input a number>6
Input a number>7
Input a number>8
end of program
```

← 7 이하를 입력하면 키보드 입력 대기를 반복

← 8 이상을 입력하면 반복에서 즉시 빠져나와 프로그램을 종료



## 4. 반복 처리 종류

- **if ~ continue** 문과 **if ~ break** 문으로 반복 흐름 바꾸기

▼ 코드 3-10 sample\_break\_else.py

```
while True:
    a = input('input a number>')
    a = int(a)

    if a >= 7:
        break

    else:
        print('the number is under 7')

print('end of program')
```

실행 화면

The diagram illustrates the execution of the program. It shows a sequence of inputs and outputs. The first two iterations show the user entering 5 and 6, both of which are less than 7, leading to the output 'the number is under 7'. A blue bracket groups these two iterations, with an arrow pointing to the text '7 미만을 입력하면 키보드 입력 대기를 반복' (Repeat keyboard input waiting if 7 or less is entered). The third iteration shows the user entering 7, which is greater than or equal to 7, leading to the output 'end of program'. An arrow points from the text '7 이상을 입력하면 반복에서 즉시 빠져나와 프로그램을 종료' (Immediately exit the loop and end the program if 7 or more is entered) to the 'end of program' output.

```
input a number>5
the number is under 7
input a number>6
the number is under 7
input a number>7
end of program
```

7 미만을 입력하면 키보드 입력 대기를 반복

7 이상을 입력하면 반복에서 즉시 빠져나와 프로그램을 종료

## 4. 반복 처리 종류

### ● 예제 프로그램: 숫자 맞추기 퀴즈

▼ 코드 3-11 sample\_number\_quiz.py

```
import random

answer = random.randint(1, 10) 1~9 중 랜덤
message = 'guess a number (1 ~ 10)'

while True:
    number = int(input(message + '>'))
    # 입력한 값을 number로 받아옴

    if number < answer:
        message = 'guess a bigger number b(_ _ )'
    elif number > answer:
        message = 'guess a smaller number q(_ _ )'
    else: number == answer
        print('Bingo! (b>_^)b')
        break

print('>_^>end of program<(^_^<')
```

실행 화면

```
>>>
guess a number (1 ~ 10)>6
guess a bigger number b(_ _)>7
guess a bigger number b(_ _)>8
Bingo! (b>_^)b
(>_^>)end of program<(^_^<)
```

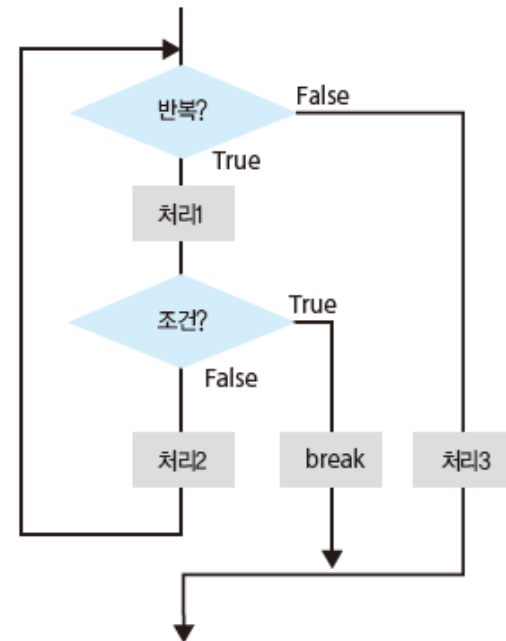
## 4. 반복 처리 종류

- **else로 반복문이 끝났을 때 실행할 처리 추가하기**
  - while 문이나 for 문에서 else를 사용하면 반복 처리가 끝난 다음에 실행할 처리를 작성
  - 도중에 break로 중단하지 않고 반복문이 끝났을 때만 실행하고 싶은 처리를 else에 작성 ▼ 그림 3-52 반복 처리 중 else 흐름

while문 안에 있는 else

for 문 또는 while 문 :

```
_____ 처리1
_____ if 조건식 :
_____ break
_____ else :
_____ 처리2
else :
_____ 처리3
```



## 4. 반복 처리 종류

- else로 반복문이 끝났을 때 실행할 처리 추가하기

▼ 코드 3-12 sample\_calculate\_quiz.py

```
import time
import random

num_of_times = 5
game_time = 25
num_of_range = 100

start_time = time.time()

for i in range(num_of_times):
    a = random.randint(1, num_of_range)
    b = random.randint(1, num_of_range)
    c = a + b
    ans = int(input(str(a) + '+' + str(b) + '=> '))
```

→ 5번반복

> 1~100 사이 랜덤int

## 4. 반복 처리 종류

```
if ans != c:
    print('Sorry, wrong answer. (*x*)')
    print('The answer is ' + str(c))
    break

elif time.time() - start_time > game_time:
    print('Time out! (><)')
    break

else:
    print('Bingo!! (^^)b')
else:
    print('Complete!')

print('end of program')
```

→ 타임아웃

## 4. 반복 처리 종류

실행 화면(제한 시간을 넘겼을 때)

>>>

1+85=> 86

Bingo!! (^^)b

79+43=> 122

Bingo!! (^^)b

11+71=> 82

Bingo!! (^^)b

17+90=> 107

Bingo!! (^^)b

49+31=> 80

Time out! (><)

end of program



정답은 맞혔지만 제한 시간을 넘겼을 때

## 4. 반복 처리 종류

실행 화면(정답을 모두 맞혔을 때)

```
>>>
11+7=> 18
Bingo!! (^^)b
45+99=> 144
Bingo!! (^^)b
56+20=> 76
Bingo!! (^^)b
32+37=> 69
Bingo!! (^^)b
94+40=> 134
Bingo!! (^^)b
Complete!
end of program
```

← 제한 시간 안에 정답을 모두 맞혔을 때



Thank you !