훈련교사 : 전 은 석
email : euns_jun@naver.com

❖ 학습 내용

- Python 프로그래밍 개요
- 개발환경 구축
- Python 식별자, 데이터형
- Python 연산자

❖ 스크립트 언어

◆ 스크립트 언어란?

- 스크립트(scripts)의 사전적 의미 - 연극의 대사등이 적혀 있는 스크립트(대본)에서 유래
- 소스코드 = 스크립트
- 연기자가 스크립트를 보고 연기 하듯 컴퓨터가 스크립트를 읽어 수행
- a special run-time environment = 수행시간 환경 = 인터프리터(interpreter : 해석기)
- 해석과 수행을 한다.
- 라인 단위로 해석하여 수행한다.

◆ 스크립트 언어의 특징

- 일반적으로 스크립트 언어는 매우 빠르게 배우고 작성하기 위해 고안되었으며,
  짧은 소스 코드 파일이나 REPL(Read–eval–print-loop)로 상호작용하는데 적합하다.
  일반적으로 상대적으로 단순한 구문과 의미를 내포한다.
  즉, 보통 "스크립트"(스크립트 언어로 작성된 코드)는 시작에서 끝날 때까지 실행되며,
  명확한 엔트리 포인트 (예: main함수)가 없다.

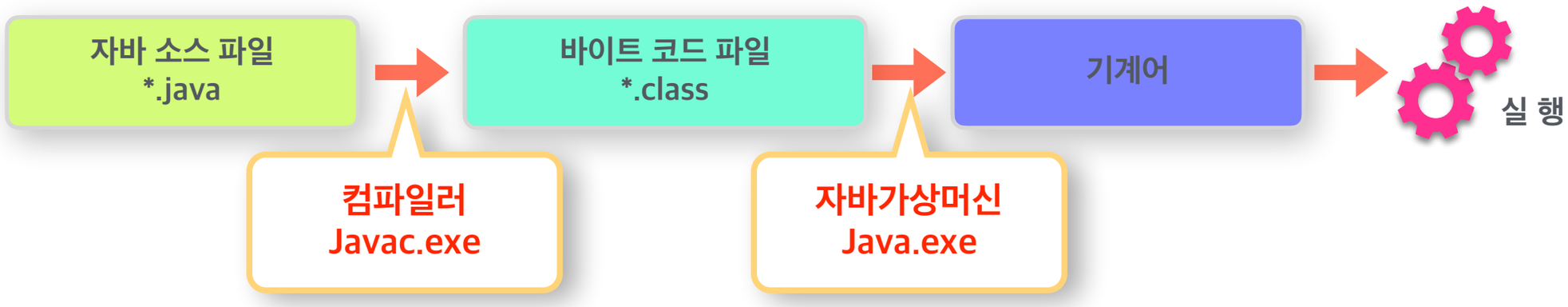
- 위키피디아 참조 -

❖ 스크립트 언어

◆ 컴파일 언어와 스크립트 언어와의 비교

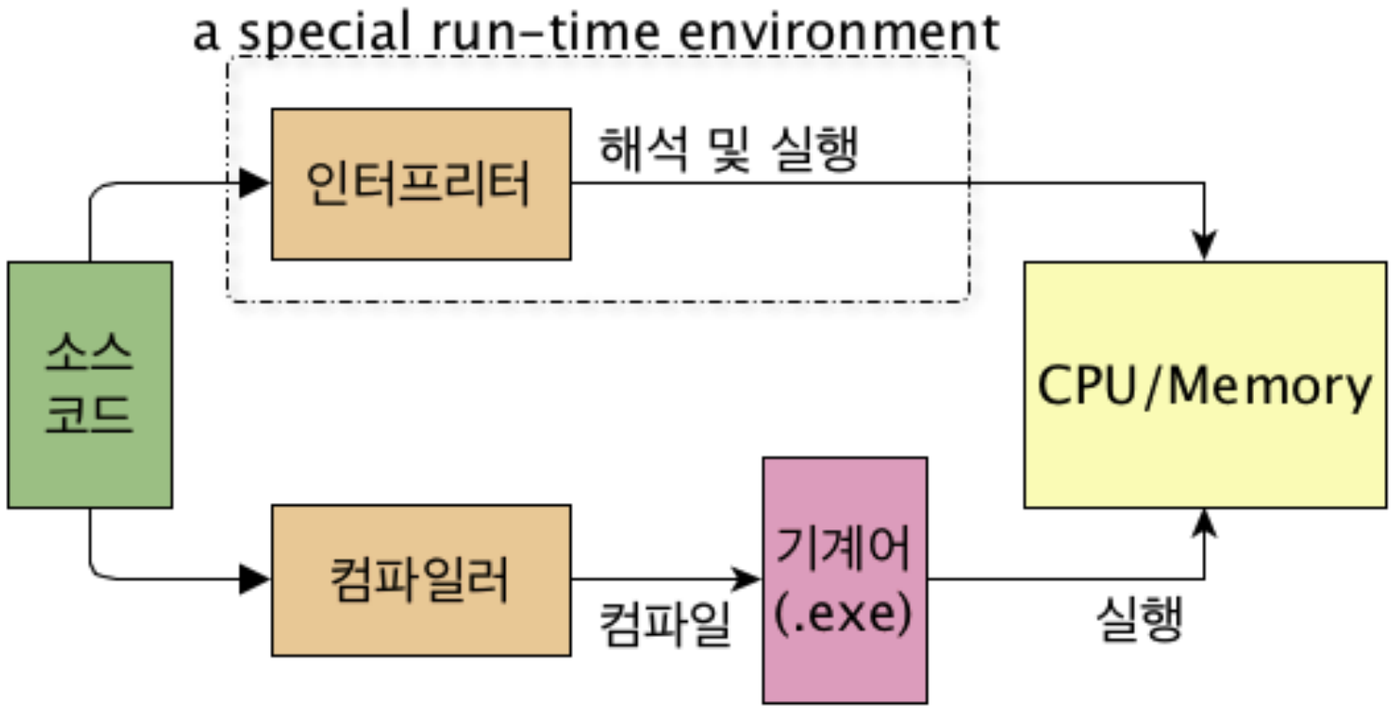
컴파일 언어	스크립트 언어
컴파일러에 의해 기계어로 번역된 채로 실행 → 수정이 빈번하게 발생할 경우 수정 후 다시 컴파일 필요	수정이 빈번하게 발생하는 경우 소스코드를 한 줄 한 줄 읽어 바로 실행하는 방식
사이즈가 큰 프로그램은 컴파일 시간이 길어진다. 간단한 수정에도 오랜 기간의 컴파일 시간이 요구된다.	스크립트 소스코드를 컴파일 방식에 의해 중간 코드(Bytecode)로 우선 만들고, 인터프리터 방식으로 해석하여 수행하는 방법도 종종 활용
자원을 많이 요구하고 컴파일 시간이 소요된다.	

❖ 자바 소스 코드 실행 과정

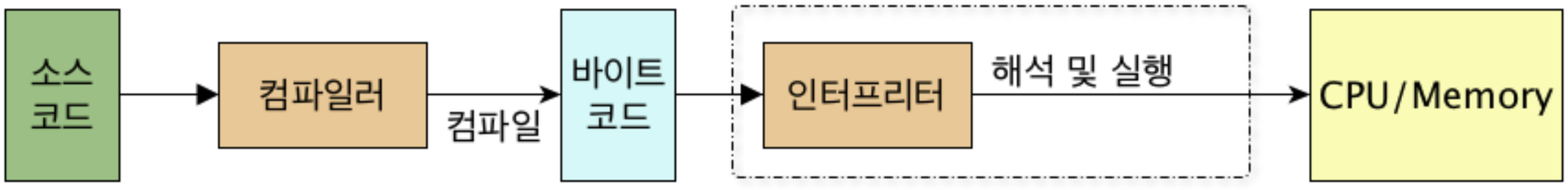


❖ 스크립트 언어

◆ 소스코드 실행 과정



❖ 컴파일 방식과 인터프리터방식을 모두 활용한 실행 과정



* 참고

파이썬의 경우 두가지 방법 모두 활용 가능

❖ 스크립트 언어

◆ 스크립트 언어의 장점

- 개발 시간이 단축된다.
- 소스 코드 수정이 빠르고 간단하게 이루어진다.

→ 소스 코드 수정이 빈번할 경우 스크립트 언어가 용이

◆ 스크립트 언어의 단점

- 중간 코드를 만드는 것은 간단하지만 그것을 실제로 실행시키는 것은 많은 작업이 필요
- 실행 시간이 오래 걸린다

→ 실행 시간을 좀 더 단축하고자 수치분석이 많이 요구됨

❖ 스크립트 언어

◆ 스크립트 언어의 종류

- JavaScript : 브라우저에서 웹페이지를 동적으로 만들어 준다.
- ActionScript : flash 개발 시 사용
- Perl
- PHP : 동적인 웹 페이지 구현 시 많이 사용
- Lua, Ruby : 최근에 개발된 스크립트 언어
- Python

❖ 스크립트 언어

◆ 스크립트 언어 사용을 피해야 되는 경우

1. 리소스에 민감한 작업들, 특히 속도가 중요한 요소일 때(정렬, 해쉬 등)
2. 강력한 산술 연산 작업들, 특히 임의의 정밀도 연산
3. 플랫폼 간 이식성이 필요할 때
4. 구조적 프로그래밍이 필요한 복잡한 애플리케이션
   (변수의 타입 체크나 함수 프로토 타입 등이 필요할 때)
5. 업무에 아주 중요하거나 회사의 미래가 걸렸다는 확신이 드는 애플리케이션
6. 보안 상 중요해서 시스템 무결성을 보장하기 위해 외부의 침입이나
   크래킹, 파괴 등을 막아야 할 필요가 있을 때
7. 서로 의존적인 관계에 있는 여러 컴포넌트로 이루어진 프로젝트
8. 과도한 파일 연산이 필요할 때
9. 다차원 배열이 필요할 때
10. 링크드 리스트나 트리 같은 데이터 구조가 필요할 때
11. 그래픽이나 GUI를 만들고 변경하는 등의 작업이 필요할 때
12. 시스템 하드웨어에 직접 접근해야 할 때
13. 포트나 소켓 I/O가 필요할 때
14. 예전에 쓰던 코드를 사용하는 라이브러리나 인터페이스를 써야 할 필요가 있을 때
15. 독점적이고 소스공개를 안 하는 애플리케이션을 만들어야 할 때(오픈소스 스크립트 언어의 경우)

- wikipedia 인용 - (출처: 김태용의 리눅스 쉘 프로그래밍)

❖ 파이썬 언어의 유래 및 파이썬 언어의 특징

◆ 파이썬의 사전적 의미

Python의 사전적 의미는 독이 없는 뱀으로 먹이를 몸으로 감아서 압사시키는 큰 뱀,
또는 그리스 신화에 나오는 악마

◆ 파이썬의 유래

프로그래밍 언어로서 Python 이름의 유래는 1970년대에 영국 BBC에서 방영한 "Monty Python's Flying Circus" 방영 프로그램에서 유래되었다. 이 프로그램은 매우 어색하고 이상한 행동을 보이는 사람들을 쇼형태로 방영하여 많은 인기를 누렸다.

이 언어를 만든 Guido van Rossum이 이 프로그램의 대단한 팬이었기 때문이었다.

1989년 12월 크리스마스 휴가를 보내고 있던 Guido는 휴가기간 동안 무엇인가 재미있게 할 수 있는 무언가를 찾고 있었다.

연휴라서 연구실이 닫힌 상황에서 집에서 컴퓨터를 가지고 예전부터 생각하고 있었던 인터프리터(interpreter)를 만들어 보기로 결심하였다.

즉, Guido가 무료한 한때를 보내기 위하여 만들기 시작했던 언어가 지금 많은 사람들이 사용하는 Python 언어가 되었다.

훈련교사 : 전 은 석
email : euns_jun@naver.com

❖ 파이썬 언어의 유래 및 파이썬 언어의 특징

◆ 파이썬의 출발

파이썬의 기반은 80년대 초반에 Guido가 동료들과 함께 개발을 하였던 ABC라는 언어이다.

80년대에 ABC라는 언어는 최고의 교육용 언어로 알려졌었고 전문 프로그래머가 아닌 사람이 배우기 편하게 만들어진 아주 훌륭한 언어였다.

그럼에도 불구하고 전문 프로그래머들에게는 거의 받아 들여지지도 않았고 잘 사용되지도 않았다.

이는 프로그래밍이 대중화가 많이 되지 않은 상황에서 전문 프로그래머들에게 외면 받았을 때에 나올 수 있는 결과라고 볼 수 있다.

Guido는 파이썬으로 실제 문제를 더욱더 쉽게 해결할 수 있도록 설계하였으며 C언어나 유닉스 쉘을 대신해서 좀 더 편하게 사용할 수 있는 언어를 개발하는 데 초점을 맞추었다.

◆ 파이썬의 필요성

- 가장 중요한 대답: "생산성이 높기 때문"
- 먼저 개발하라! 그리고 나서 성능을 개선하라.

＊개발을 먼저 할 시 장점 : 협업 시 비교하여 성능을 높이기 용이함

❖ 파이썬 언어의 유래 및 파이썬 언어의 특징

◆ 파이썬의 특징

- 대화 기능의 인터프리터 언어
- •. 동적인 데이터 타입 결정 지원

*대화 기능 → 마치 컴퓨터와 개발자가 대화하는듯한 느낌

```
def add(a,b):
    return a+b
print(add(1,2))
print(add('abc', 'def'))
print(add([1, 2, 3], [4, 5, 6]))
```

```
3
abcdef
[1, 2, 3, 4, 5, 6]
```

def → add 함수를 만드는 키워드

print() → add 함수 수행 값을 반환한 것을 출력하는 것

문자열을 더하면 문자들이 합쳐짐

리스트 + 리스트 = 하나의 리스트

함수의 a와 b는 타입이 정해져 있지 X

타입은 실제로 값이 변수에 할당되는 순간에 정해짐

❖ 파이썬 언어의 유래 및 파이썬 언어의 특징

◆ 파이썬의 특징

- 플랫폼 독립적 언어
- 개발 기간 단축에 촛점을 둔 언어
- 간단하고 쉬운 문법
- 고수준의 내장 객체 자료형 제공

```
a=[12, 'abcde', 4+2j, [3,4,5]]
a.append('add')
print(a)
```

[12, 'abcde', (4+2j), [3, 4, 5], 'add']

- 윈도우에서 개발한 파이썬 소스코드를 맥과 리눅스에서 수행 가능
- C언어의 경우 소스코드를 컴파일을 다시 진행해야 함
- 이미 OS에서 인터프리터가 설치가 되어 있기 때문
- a 변수 → 리스트
- 리스트 : 서로 다른 객체들을 원소로 갖고, 순서가 부여된 자료구조형
- 리스트 : 첫 번째, 두 번째 인덱싱을 할 수 있음

❖ 파이썬 언어의 유래 및 파이썬 언어의 특징

◆ 파이썬의 특징

- 메모리 자동 관리
- 쉬운 유지 보수
- 많은 수의 라이브러리 제공
- 짧아지는 코드
- 높은 확장성

- C언어 또는 Java → 100라인, 파이썬 → 5~10라인
- 코딩 시 필요한 모듈 끌어와 사용 가능
- 직접 만든 모듈도 다른 사람에게 제공 가능

❖ 파이썬 언어의 유래 및 파이썬 언어의 특징

◆ 파이썬의 활용처

- 시스템 유틸리티
- GUI - wxpython, tkinter
- 게임 프로그래밍 - 파이썬 게임엔진: PyOpenGL PySDL PyGame Kivy PyOgre Panda3D Cocos2D PySoy
- 웹 프로그래밍 - django 프레임워크
- 수치 프로그래밍 - numpy 및 pandas 모듈 - networks 모듈
- 데이터베이스 프로그래밍
- 기타

- 시스템 유틸리티: OS가 지원해야 하는 명령어
- 리눅스 → ls, DOS → dir, copy, move

❖ 파이썬 언어의 유래 및 파이썬 언어의 특징

◆ 파이썬의 장점

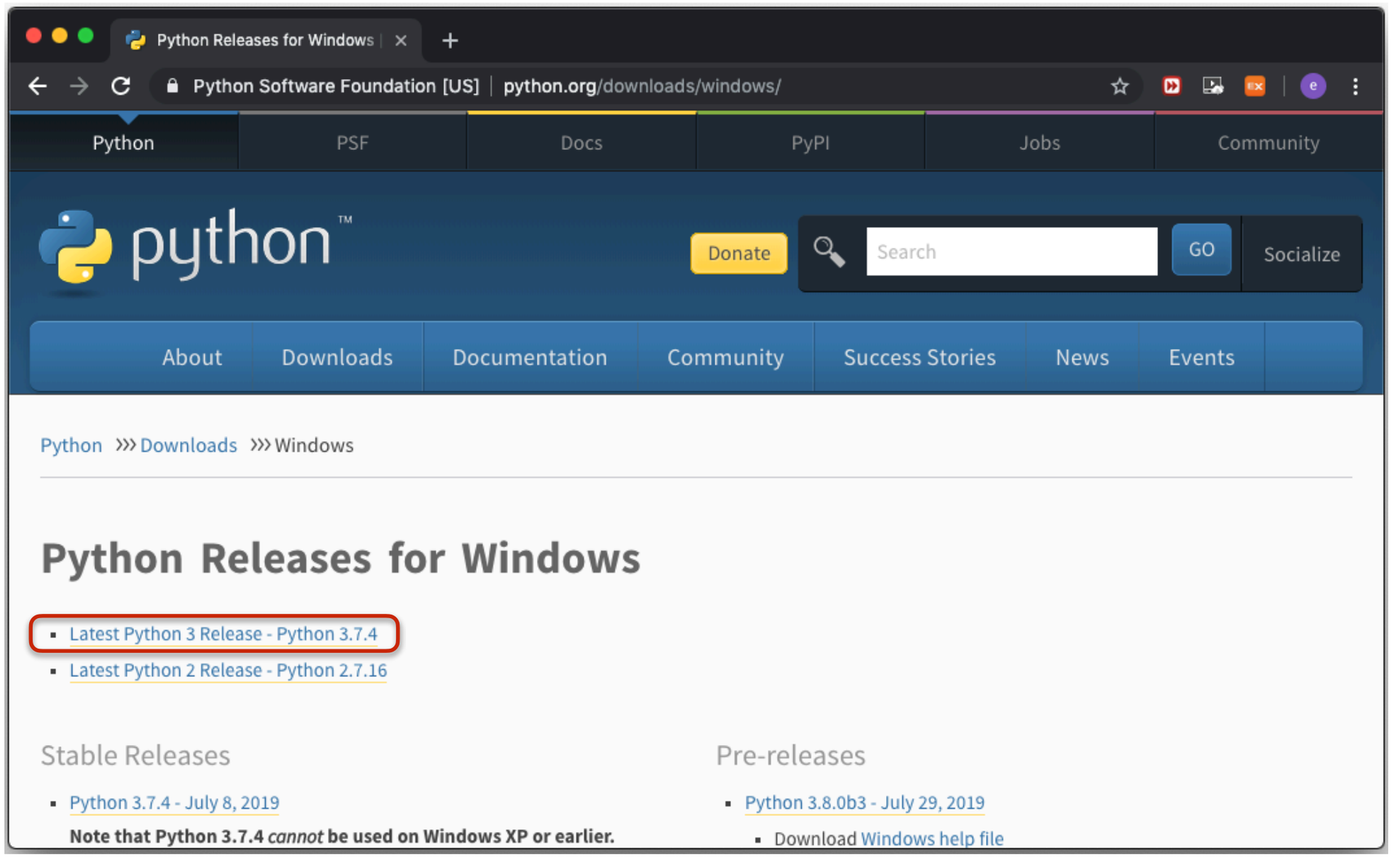
- Guido가 생각했던 Python 문법적 특징은 들여쓰기를 철저하게 지키도록 언어를 설계했다는 점이다.
- 이는 코드의 가독성을 현격히 높여준다.
- C 언어에서처럼 {} 등의 괄호를 넣지 않기 때문에 프로그램을 좀더 깔끔하게 만들어준다.
- 파이썬 코드는 재사용하기가 쉽다.
- 코드의 분석이 쉽기 때문에 다른 사람이 작성한 코드를 받아서 작업하는 사람들이 훨씬 더 작업을 편하게 해준다.

- 파이썬은 들여쓰기를 안 할 경우 error 발생 → 들여쓰기 강제

❖ 파이썬 개발환경 구축

◆ 파이썬의 다운로드

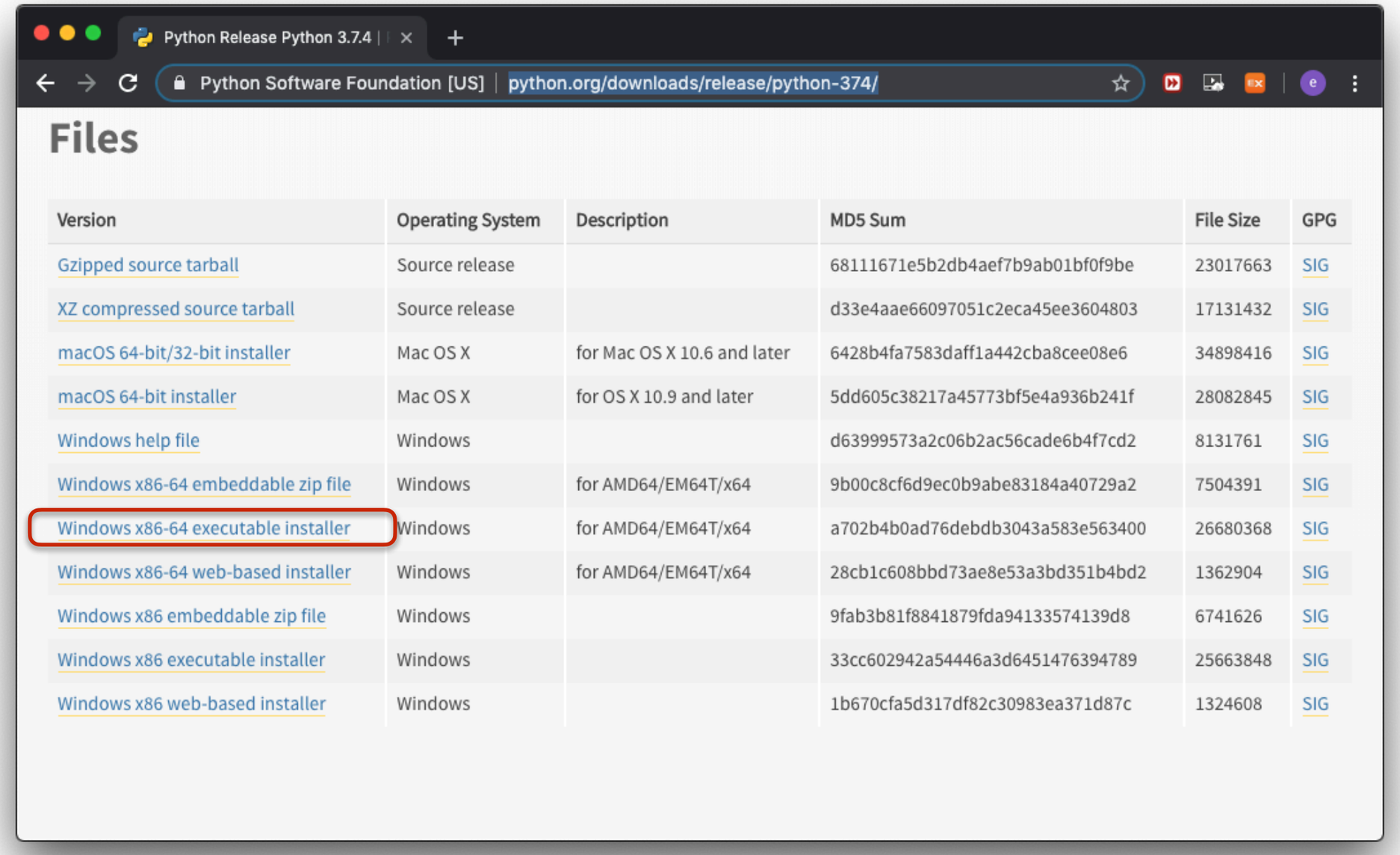
- https://www.python.org/downloads/windows/
  에 접속해서 자신의 환경에 맞는 파일을 다운 받는다.
- 우리의 경우 윈도우즈용 환경에 맞는 Python 3.7.4 클릭



❖ 파이썬 개발환경 구축

◆ 파이썬의 다운로드

- https://www.python.org/downloads/release/python-374/
  페이지에서 이동한 후 아래쪽으로 스크롤해서 Files 섹션에서
- Windows x86-64 executable installer 링크를 클릭해서 다운 받는다.



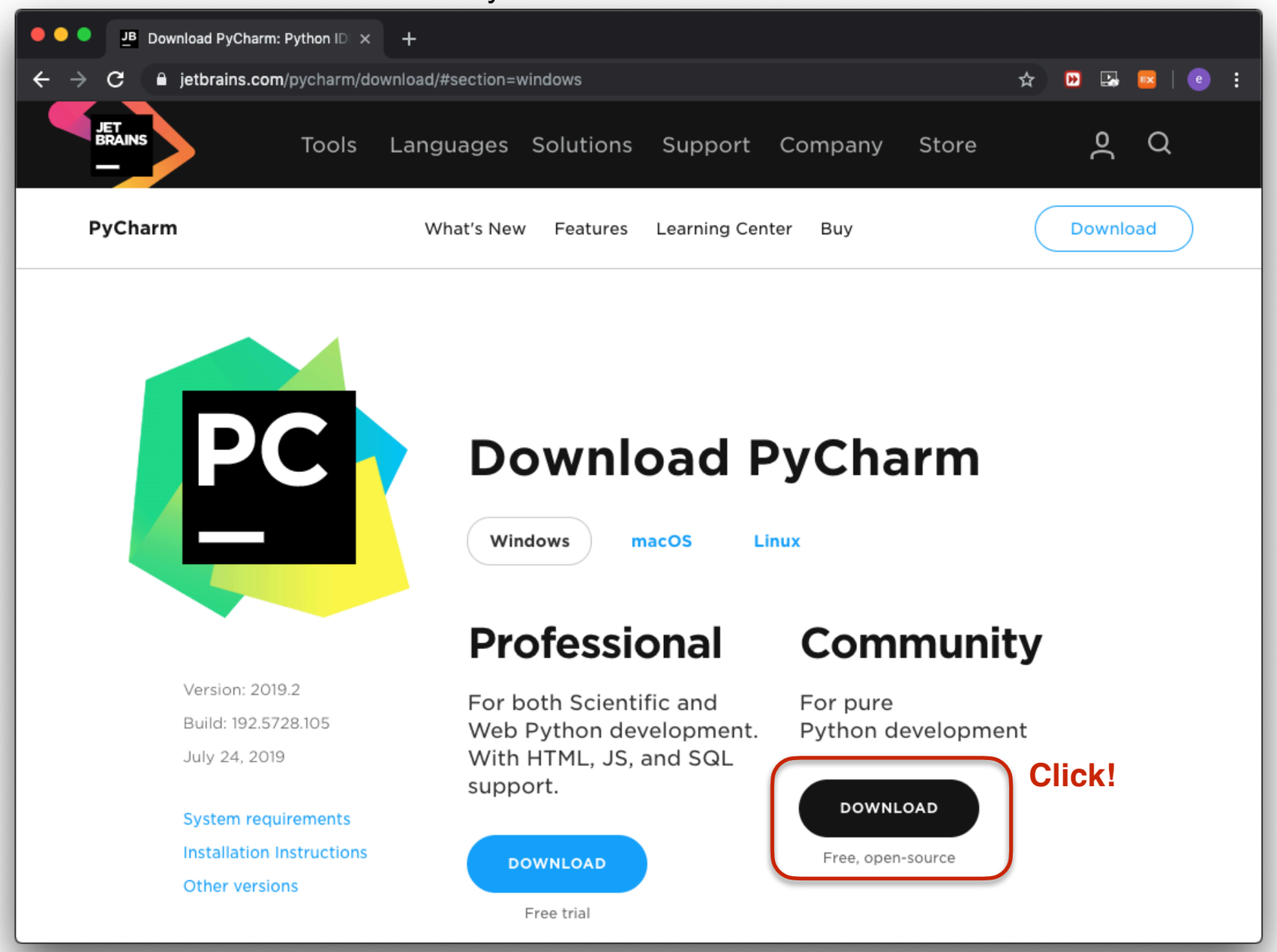
Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		68111671e5b2db4aef7b9ab01bf0f9be	23017663	SIG
XZ compressed source tarball	Source release		d33e4aae66097051c2eca45ee3604803	17131432	SIG
macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	6428b4fa7583daff1a442cba8cee08e6	34898416	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	5dd605c38217a45773bf5e4a936b241f	28082845	SIG
Windows help file	Windows		d63999573a2c06b2ac56cade6b4f7cd2	8131761	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	9b00c8cf6d9ec0b9abe83184a40729a2	7504391	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	a702b4b0ad76debdb3043a583e563400	26680368	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	28cb1c608bbd73ae8e53a3bd351b4bd2	1362904	SIG
Windows x86 embeddable zip file	Windows		9fab3b81f8841879fda94133574139d8	6741626	SIG
Windows x86 executable installer	Windows		33cc602942a54446a3d6451476394789	25663848	SIG
Windows x86 web-based installer	Windows		1b670cfa5d317df82c30983ea371d87c	1324608	SIG

✻ 다운로드 받은 파일을 실행한다.

❖ 파이썬 개발환경 구축

◆ PyCharm 다운로드

- https://www.jetbrains.com/pycharm/download/#section=windows
  페이지에서 이동한 후 Community 를 클릭한다.



* 다운로드 받은 파일을 실행해서 설치한다.

❖ 파이썬 테스트

◆ IDLE에서 간단한 파이썬 예제 실행하기

- 파이썬이 설치되어 있는 경로에서

  ./Lib/idlelib/idle.pyw

  파일을 찾아서 실행한다.(또는 시작메뉴에서 IDLE(Python 3.7 64-bit) 아이콘 클릭)

```
print(4 + 5)
print('Hello World!')
```

```
9
Hello World!
```

❖ 파이썬 테스트

◆ 파이썬 버전 알아보기

```
import sys
print(sys.version)
print()
print(sys.version_info)
```

```
>>> print(sys.version)
3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 03:03:55)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)]
>>> print()

>>> print(sys.version_info)
sys.version_info(major=3, minor=6, micro=5, releaselevel='final', serial=0)
>>>
```

❖ 파이썬 테스트

◆ 파일로 저장하여 실행하기

- 다음 경로에

  C:\PyData\cal.py

  파일을 만든다.

```
# file: cal.py
import calendar
calendar.prmonth(2019, 8)
```

∗ prmonth : print month의 약자

- CMD 창을 실행시킨 후 파일이 저장된 경로로 이동해서 실행한다.

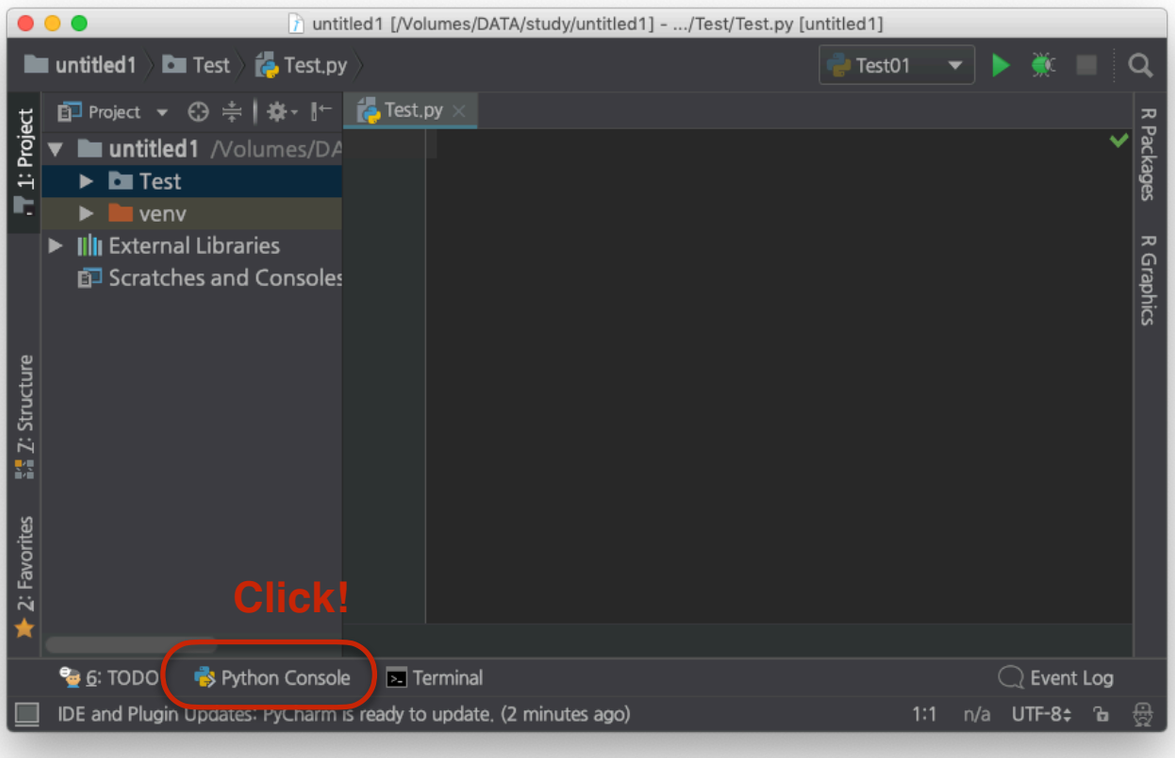
```
cd c:\PyData
python cal.py
```

```
    August 2019
Mo Tu We Th Fr Sa Su
                1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

❖ 파이썬 테스트

◆ 대화식 모드에서 저장된 파일 실행하기

- PyCharm 을 실행한다.
- 아래쪽 탭 중
  Python Console
  을 클릭한다.




```
execfile('c:\\PyData\\cal.py')
```

```
    August 2019
Mo Tu We Th Fr Sa Su
                1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

훈련교사 : 전 은 석
email : euns_jun@naver.com

파이썬 기본 문형

훈련교사 : 전 은 석
email : euns_jun@naver.com

❖ 파이썬 예약어 및 내장함수

◆ 예약어

- 예약어 (또는 키워드)
  - 파이썬에서 이미 문법적인 용도로 사용되고 있기 때문에 변수명 등의 식별자로 사용하면 안 되는 단어들

- 예약어는 Reserved(예약된) Words 또는 키워드라고 함
- 파이썬에서 이미 사용되고 있는(용도가 예약된) 단어들
- 이미 문법적인 용도로 사용 → 신택스(Syntax)
- 코딩할 때 활용할 예약어를 변수에 활용하면 안됨
- 예약어를 변수에 활용 → 에러는 없으나 고유 기능은 사라짐

❖ 파이썬 예약어 및 내장함수

◆ 예약어의 종류 알아보는 방법

- 파이썬의 예약어 종류를 알아보는 방법은?
- keyword 모듈을 불러옴(import)
- keyword 모듈이 지원하는 kwlist를 출력(print())
- 이 때 print()도 예약어이므로 다른 식별자로 사용하면 안됨
- print()만 실행 → 한 줄을 띄워줌
- len → 특별한 모듈 추가 없이 사용할 수 있는 내장 함수
- keyword 모듈에 있는 kwlist에 몇 개의 단어가 있는지 알아봄
- 프로그램을 수행하면 콘솔에 수행결과가 나타남
- 수행결과는 문자열이 나열된 리스트 형태
- 리스트 안에 나열된 문자들 → 예약어
- 예약어의 개수는 31개(len 함수의 결과)
- 파이썬의 버전에 따라 예약어 종류 달라짐
- 파이썬 버전 3.6 에서는 33개 예약어 지원함
- 앞서 알아본 len이 내장 함수(Built-in Function)
- import keyword에서 keyword가 모듈
- 모듈을 불러올 때는 import(내장 함수) 사용
- 현재 프로그램에 사용된 예약어 → import, print
- keyword 모듈은 기본 인터프리터 환경에 추가 안됨
- 파이썬을 설치하면 수많은 모듈이 설치됨
- import keyword → 그 중 keyword 모듈을 사용하겠다
- 모듈 안에 있는 함수는 모듈을 import 해야 사용 가능

❖ 파이썬 예약어 및 내장함수

◆ 예약어의 종류 알아보기

➡ ex 1]

```
import keyword
print(keyword.kwlist)
print()
print(len(keyword.kwlist))
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del',
'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal',
'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

35
```

* python 3.7.4 기준 35개의 예약어

❖ 파이썬 예약어 및 내장함수

◆ 내장 함수

- 별도의 모듈(Module)의 추가 없이 기본적으로 제공되는 함수들

- 참고사이트
- 내장(Built-in) 함수: http://docs.python.org/3/library/functions.html
- 대표적인 내장 함수
- abs, max, min, pow, chr, str, range, type, …
- abs(x) : 수치형 자료 x에 대해 x의 절대값을 반환하는 함수

모듈 : 파이썬 파일 하나를 의미한다

- 내장 함수는 모듈 추가 없이 활용 가능
- 파이썬 공식 홈페이지의 내장 함수 안내 페이지
- abs → 수치형 자료를 절대값으로 반환하는 내장 함수

➡ ex 1]

```
print(abs(3))
print(abs(-3))
```

```
3
3
```

❖ 파이썬 예약어 및 내장함수

◆ 내장 함수

- max(s)
  - 시퀀스 자료형(문자열, 리스트, 튜플)을 입력받아 그 자료가 지닌 원소 중 최대값을 반환하는 함수

* max → 주어진 자료 중 최대값을 반환하는 내장 함수

➡ ex 1]

```
print(max(1,2))
print(max([1, 2, 3]))
print(max("python"))
```

```
2
3
y
```

* 문자열 중 아스키 코드값이 가장 큰 문자 반환

❖ 파이썬 예약어 및 내장함수

◆ 내장 함수

- min(s)
  - 시퀀스 자료형(문자열, 리스트, 튜플)을 입력받아 그 자료가 지닌 원소 중 최소값을 반환하는 함수

➡ ex 1]

```
print(min(1,2))
print(min([1, 2, 3]))
print(min("python"))
```

```
1
1
h
```

➡ 설 명]

- print(min(1, 2)) → 1과 2 중 더 작은 1을 반환
- print(min("python")) → 문자열 중 아스키 코드값이 가장 작은 문자 반환

❖ 파이썬 예약어 및 내장함수

◆ 내장 함수

- pow(x, y)
  - 수치형 자료형 x, y에 대해 x의 y승을 반환하는 함수

  - pow(a, b) → a의 b승 값을 반환

➡ ex 1]

```
print pow(2,4)
print pow(3, 3)
print pow(2, -1)
```

```
16
27
0.5
```

➡ 설 명]

- print pow(2, 4) → 2의 4승, 즉, 16을 출력
- print pow(3, 3) → 3의 3승, 즉, 27을 출력
- print pow(2, -1) → 2의 -1승, 즉, 0.5를 출력

❖ 파이썬 예약어 및 내장함수

◆ 내장 함수

- chr(i)
  - 정수 형태의 ASCII코드 값을 입력으로 받아 그에 해당하는 문자를 반환하는 함수
  - 인수 i의 범위: 0부터 255까지

  - chr() → 아스키 코드값을 문자로 변환해 주는 함수

➡ ex 1]

```
print(chr(97))
print(chr(65))
print(chr(48))
```

```
a
A
0
```

➡ 설 명]

- print(chr(97)) → 아스키 코드값이 97인 문자 → a
- print(chr(65)) → 아스키 코드값이 65인 문자 → A
- print(chr(48)) → 아스키 코드값이 48인 문자 → 0
- 이 때 0은 숫자 0이 아니라 문자로써의 0

❖ 파이썬 예약어 및 내장함수

◆ 내장 함수

- str(object)
  - 임의의 객체 object에 대해 해당 객체를 표현하는 문자열을 반환하는 함수

  - str(a) → a를 출력(print)하면 어떻게 나타나는지를 반환

➡ ex 1]

```
print(str(3))
print(str([1, 2]))
```

```
3
[1, 2]
```

➡ 설 명]

- str(3) → 3이라는 객체를 출력했을 때 나타나는 결과 → 3
- str([1,2]) → [1, 2]를 출력했을 때 나타나는 결과 → [1, 2]
- str() → 해당 객체를 표현하는 문자열로 반환해주는 함수

❖ 파이썬 예약어 및 내장함수

◆ 내장 함수

- range([start,]stop[,step])
  - 수치형 자료형으로 start, stop, step 등을 입력받아 해당 범위에 해당하는 정수를 리스트로 반환하는 함수
- 인수가 하나(stop)인 경우
  - 0부터 stop-1까지의 정수 리스트를 반환한다.
- 인수가 두 개(start, stop)인 경우
  - start부터 stop-1까지의 정수 리스트를 반환한다
- 인수가 세 개(start, stop, step)인 경우
  - start부터 stop-1까지의 정수를 반환하되 각 정수 사이의 거리가 step인 것들만 반환한다.

➡ ex 1]

```
print(range(10))
print(range(3, 10))
print(range(3, 10, 2))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[3, 4, 5, 6, 7, 8, 9]
[3, 5, 7, 9]
```

* 2.x.x 버젼에서 실행

```
range(0, 10)
range(3, 10)
range(3, 10, 2)
```

* 3.x.x 버젼에서 실행

➡ 설 명]

- str(3) → 3이라는 객체를 출력했을 때 나타나는 결과 → 3
- str([1,2]) → [1, 2]를 출력했을 때 나타나는 결과 → [1, 2]
- str() → 해당 객체를 표현하는 문자열로 반환해주는 함수

❖ 파이썬 예약어 및 내장함수

◆ 내장 함수

- range([start,]stop[,step])

➡ 설 명 1] - python 2.x.x 환경에서 실행

- range(10) → 0부터 10-1까지의 정수 리스트를 반환
- range(3, 10) → 3부터 10-1까지의 정수 리스트를 반환
- range(3, 10, 2) → 3부터 10-1까지 2개씩 건너 띈 정수 리스트를 반환
- range(3, 10, 2) → [3, 5, 7, 9]
- 5 – 3 = 2(step) → 앞, 뒤의 차이가 2씩 나게 함
- 7 – 5 = 2(step) → 앞, 뒤의 차이가 2씩 나게 함
- 9 – 7 = 2(step) → 앞, 뒤의 차이가 2씩 나게 함
- range 함수는 각각 원소의 차이가 step값만큼 나도록 할 수 있음
- range([start], stop, [step]) → start와 step은 생략 가능
- range(10) → stop만 10으로 기술한 것
- range(10) → 10-1까지의 정수 리스트를 반환
- start가 생략되었을 경우 0부터 시작
- range(a, b, c) → a부터 b-1까지 c씩 차이 나게 정수 리스트를 반환

➡ 설 명 2]

- 3.x.x 환경에서 실행하는 경우 변수에 객체(함수)자체를 저장하게 되므로 range() 자체를 반환한다.

❖ 파이썬 예약어 및 내장함수

◆ 내장 함수

- type(a)
  - a의 자료형을 반환하는 함수

➡ ex 1]

```
print(type(-1))
print(type('abc'))
print(type([1, 2, 3]))
```

```
<class 'int'>
<class 'str'>
<class 'list'>
```

➡ 설 명]

- print(type(-1)) → -1의 자료형을 반환 → int(정수형)
- print(type('abc')) → 'abc'의 자료형을 반환 → str(문자형)
- print(type([1, 2, 3])) → [1, 2, 3]의 자료형을 반환 → list(리스트)
- type() 함수를 사용하여 객체의 자료형을 알 수 있음

❖ 파이썬 식별자와 변수 사용

◆ 식별자 만드는 법

- 파이썬 식별자는 변수, 함수, 모듈, 클래스 또는 객체를 식별하는데 사용되는 이름이다.
- 식별자의 조건

  - 대소문자 구별함
  - 식별자는 문자 A-Z 또는 a-z와 언더바(_)로 시작할 수 있다.
  - 식별자 첫 시작을 제외하고 식별자 내에 숫자(0~9)를 사용할 수 있다.
  - 특수문자 @, $, %등은 식별자에 올 수 없다.
  - 예를 들어 다음과 같은 것은 식별자가 될 수 없음 : 1abc, @file, %x

- 파이썬 식별자: 변수, 함수, 모듈, 클래스, 객체 식별에 사용되는 이름
- 가장 흔히 쓰는 식별자 → 변수 이름
- 프로그래밍 하면서 식별자를 정하게 됨
- 식별자 정할 때의 조건 1. 대소문자를 구별함
- 식별자를 정할 때 a(소문자)와 A(대문자)는 다름
- 변수 a와 A가 다르게 인식됨
- 조건 2. 식별자는 문자 또는 언더바(_)로 시작함
- aaa, AAAA, _aaa, _AAAA 모두 식별자로 사용 가능
- 조건 3. 식별자는 맨 처음을 제외하고 숫자 포함 가능
- _aaa2, _a2aa 모두 식별자로 사용 가능
- 그러나 2_a2aa는 식별자로 사용 불가
- 조건 4. 특수 문자는 식별자로 사용 불가
- 식별자로 사용 불가 → 1abc(숫자가 맨 앞), @file, %x(특수 문자 포함)

❖ 파이썬 식별자와 변수 사용

◆ 변수명 만들 때 조심할 점

- 예약어, 내장함수, 모듈 이름을 변수명으로 만드는 일이 없도록 할 것

■ 변수명 만들 때 예약어, 내장 함수, 모듈 이름으로 만들지 않도록 주의
■ 예약어, 내장 함수, 모듈 이름은 이미 활용되고 있기 때문

➡ ex 1]

```
print(str(12345))
```

```
12345
```

■ str은 주어진 객체를 출력해주는 함수

➡ ex 2]

```
str = 'abc'
print(str(12345))
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object is not callable
```

■ 본래 str 함수가 가지고 있던 기능을 잃어버림

❖ 파이썬 식별자와 변수 사용

◆ 변수의 생성 및 사용

- 파이썬에서 변수가 생성되는 시점은 해당 변수에 임의의 값이 할당될 때이다.

➡ ex 1]

```
a = 1
print(a)
```

```
1
```

- 파이썬과 다른 언어의 큰 차이점 → 변수 생성 시 타입(type)을 적지 않음
- 변수에 값이 할당될 때 변수의 타입이 정해짐
- type 함수를 통해 a의 타입을 확인해보면 int(정수)로 나타남
- 이 때 a의 타입은 값이 할당이 되었을 때 int로 정해진 것
- 만약 a에 실수형(1.0)을 할당하면 타입은 float(실수)으로 나타남
- 동일한 a라도 할당된 값에 따라 타입이 바뀜
- 즉, 변수의 타입을 처음부터 정할 수 없음
- 이를 파이썬의 동적 특성, 동적 변수 할당이라 함

❖ 파이썬 식별자와 변수 사용

◆ 변수의 생성 및 사용

- 변수의 생성 없이 곧바로 사용할 수 없다.

➡ ex 2]

```
print(b)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'b' is not defined
```

➡ 설 명]

- 변수 b 생성 않고 바로 print b → 에러 발생
- 변수 b가 생성되지 않았기 때문에 이해할 수 없음
- 반드시 변수에 값을 할당한 후 사용해야 함

❖ 파이썬 식별자와 변수 사용

◆ 변수의 삭제

- del 이라는 예약어 사용

➡ ex 1]

```
b = 2
print(b)
del b
print(b)
```

```
2
----------------------------------------------------------------------------------------------------
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'b' is not defined
```

➡ 설 명]

- del → 변수를 삭제 할 때 사용
- b에 2를 할당한 후 print b → 2가 정상적으로 출력됨
- del b → b라는 변수를 삭제
- 그 후 print b → 에러 발생

❖ 파이썬 기초 문형

◆ 주석문 (단일행 주석)

➡ ex 1]

```
#이것은 주석입니다.
import sys #이것도 주석입니다.
```

➡ 설 명]

- 주석문: #(샵) 뒤에 오는 문장
- 주석문은 첫 번째 줄에 올 수도 있고
- 일반적인 파이썬문 뒤에 주석문이 올 수도 있음
- 한 줄을 모두 차지하거나 명령문 뒤에 달렸거나 상관없이
- #(샵) 기호와 함께 사용되면 주석

❖ 파이썬 기초 문형

◆ 주석문 (다중행(Multi Line) 주석)

➡ ex]

```
"""
이것은 주석입니다.
import sys
이것도 주석입니다.
"""
```

➡ 설 명]

- 여러줄을 주석으로 처리하고자 할 때 따옴표 세개를 사용해서 표현한다.

""" 내용 """
or
''' 내용 '''

❖ 파이썬 기초 문형

◆ 연속라인

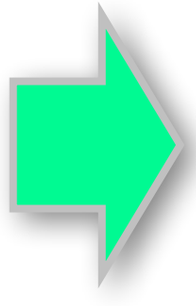
➡ ex]

```
a = 1
b = 3
if (a == 1) and ＼
    (b == 3) :
    print('connected lines')
```

```
connected lines
```

➡ 설 명]

- a가 1이고 b가 3이면 'connected lines'라는 문구를 출력하는 예제
- 이 때 if부터 3까지는 한 줄이 되어야 함
- ＼(백슬래시) → 나눠진 줄을 한 줄로 인식하도록 함
- ＼(백슬래시)는 코딩이 길어져 한 화면에 나타나지 않을 때 사용



```
a = 1
b = 3
if (a == 1) and (b == 3) :
    print('connected lines')
```

훈련교사 : 전 은 석
email : euns_jun@naver.com

❖ 파이썬 기초 문형

◆ 할당문

➡ ex 1]

```
a = 1
b = a
```

➡ 설 명 1]

- 할당문 → 등호(=)를 사용한 연산
- a = 1 → a에 1을 할당
- b = a → b에 a를 할당 → a가 1이므로 b도 1
- 그러므로 a, b 모두 1을 할당

- 주의! 등호 왼쪽에 표현식이 오면 안됨(변수가 와야 함)

➡ ex 2]

```
1 + 3 = a
```

```
  File "<stdin>", line 1
SyntaxError: can't assign to operator
```

➡ 설 명 2]

- 표현식(1+3)은 등호 왼쪽에 올 수 없음

❖ 파이썬 기초 문형

◆ 할당문

➡ ex 3]

```
a = 1
a = a + 1
print(a)
```

```
2
```

➡ 설 명 3]

- 표현식은 등호 오른쪽에 올 수 있음
- 이 때 오른쪽의 표현식이 평가됨
- a = a + 1 → 현재 a가 1이므로 1 + 1, 즉, 2 → a = 2

❖ 파이썬 기초 문형

◆ 할당문

➡ ex 4]

```
c, d = 3, 4
print(c, d)

x = y = z = 0
print(x, y, z)

e = 3.5; f = 5.6
print(e, f)
```

```
3 4
0 0 0
3.5 5.6
```

➡ 설 명 4]

- c, d = 3, 4 → c와 d에 각각 3과 4를 할당
- x = y = z = 0 → 맨 오른쪽부터 이해해야 함
- z = 0 → z에 0을 할당
- y = z → z의 값을 y에 할당 → y = 0
- x = y → y의 값을 x에 할당 → x = 0
- print(x, y, z) → 0 0 0
- 세미콜론(;): 하나의 문장이 끝났음을 의미
- 두 문장을 한 줄에 이어 쓸 때 세미콜론(;)을 사용
- 세미콜론(;)은 프로그램 가독성 문제로 자주 활용되지 않음

❖ 파이썬 기초 문형

◆ 할당문

➡ ex 4]

```
c, d = 3, 4
print(c, d)

x = y = z = 0
print(x, y, z)

e = 3.5; f = 5.6
print(e, f)
```

```
3 4
0 0 0
3.5 5.6
```

➡ 설 명 4]

- c, d = 3, 4 → c와 d에 각각 3과 4를 할당
- x = y = z = 0 → 맨 오른쪽부터 이해해야 함
- z = 0 → z에 0을 할당
- y = z → z의 값을 y에 할당 → y = 0
- x = y → y의 값을 x에 할당 → x = 0
- print(x, y, z) → 0 0 0
- 세미콜론(;): 하나의 문장이 끝났음을 의미
- 두 문장을 한 줄에 이어 쓸 때 세미콜론(;)을 사용
- 세미콜론(;)은 프로그램 가독성 문제로 자주 활용되지 않음

❖ 파이썬 기초 문형

◆ 할당문

➡ ex 4]

```
c, d = 3, 4
print(c, d)

x = y = z = 0
print(x, y, z)

e = 3.5; f = 5.6
print(e, f)
```

```
3 4
0 0 0
3.5 5.6
```

➡ 설 명 4]

- c, d = 3, 4 → c와 d에 각각 3과 4를 할당
- x = y = z = 0 → 맨 오른쪽부터 이해해야 함
- z = 0 → z에 0을 할당
- y = z → z의 값을 y에 할당 → y = 0
- x = y → y의 값을 x에 할당 → x = 0
- print(x, y, z) → 0 0 0
- 세미콜론(;): 하나의 문장이 끝났음을 의미
- 두 문장을 한 줄에 이어 쓸 때 세미콜론(;)을 사용
- 세미콜론(;)은 프로그램 가독성 문제로 자주 활용되지 않음

훈련교사 : 전 은 석
email : euns_jun@naver.com

❖ 파이썬 기초 문형

◆ 할당문

- 두 변수의 값을 swap하는 방법

➡ ex 1]

```
e = 3.5; f = 5.6
e, f = f, e
print(e, f)
```

```
5.6 3.5
```

➡ 설 명 1]

- 스왑(swap): 값을 바꾸는 것을 의미
- e, f = f, e → e에 f를 할당, f에 e를 할당 → 값을 스왑하는 문장
- print로 결과 확인하면 두 값이 바뀐 것을 알 수 있음

- 아래에서 b = c + d는 식(Expression)이 아니라 문(Statement)이기 때문에 a에 할당될 수 없다.

➡ ex 2]

```
a = (b = c + d)
```

```
File "<stdin>", line 1
  a = (b = c + d)
       ^
SyntaxError: invalid syntax
```

➡ 설 명 2]

- a = (b = c + d) → 잘못된 식
- (b = c + d) → 식이 아니라 문
- 문 자체가 a에 할당될 수 없음

❖ 파이썬 기초 문형

◆ 확장 할당문

➡ ex 1]

```
a = 1
a += 4
print(a)
```

```
5
```

➡ 설 명 1]

- 확장 할당문: +=, -=, *=, /=, %=, **=, //=
- a += 4는 a = a + 4와 동일
- a += 4 → a에 4를 더한 후 다시 a에 할당

➡ ex 2]

```
a = 10
a *= 2+3
print(a)
```

```
50
```

➡ 설 명 2]

- a *= 2 + 3 → a에 2 + 3을 곱한 후 다시 a에 할당(2 + 3을 먼저 수행)
- 2 + 3 = 5, 그 다음 10 * 5 = 50
- a에 50을 할당

❖ 파이썬 기초 문형

◆ 객체와 할당

- 객체의 변수는 해당 객체의 레퍼런스를 지니고 있음
- a = 1이라는 Statement에서 a는 이름
- 1은 객체이며 a 변수는 1이라는 객체를 가리킨다.
  - 즉, a 변수는 1 객체의 레퍼런스를 지니고 있음

- a = 1 → a는 변수 이름, 1은 객체
- a = 1 → a라는 변수가 어딘가에 있는 객체 1을 가리키는 것을 의미
- 가리킨다 → 레퍼런스를 지니고 있다

훈련교사 : 전 은 석
email : euns_jun@naver.com

❖ 파이썬 기초 문형

◆ 객체와 할당

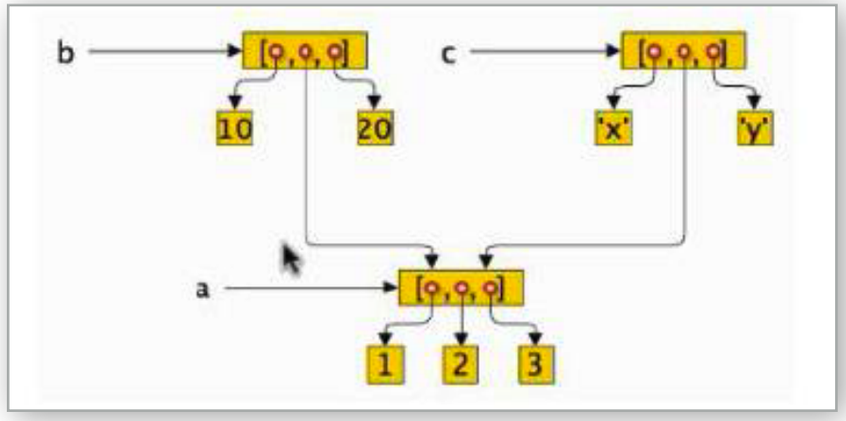
➡ ex 1]

```
a = [1, 2, 3]
b = [10, a, 20]
c = ['x', a, 'y']
print(a)
print(b)
print(c)
```

```
[1, 2, 3]
[10, [1, 2, 3], 20]
['x', [1, 2, 3], 'y']
```

➡ 설 명 1]

- a = [1, 2, 3] → 어딘가에 있는 1, 2, 3을 가리키는 형태
- 리스트 내부에는 1, 2, 3을 가리키는 레퍼런스 값이 존재
- 1, 2, 3을 가리키는 레퍼런스가 존재하는 리스트를 a가 가리킴
- b도 어딘가에 있는 10, 20을 가리키는 레퍼런스 값이 존재
- 리스트 안에 있는 a는 [1, 2, 3]을 가리키는 형태
- c가 어떤 리스트를 가리키고 있음
- 리스트는 x, y라는 문자를 가리킴
- 리스트 안에 있는 a는 [1, 2, 3]을 가리키는 형태
- 파이썬에서는 모든 것이 다 객체
- 식별자 변수는 객체를 가리키는 형태
- a = [1, 2, 3] → print a → [1, 2, 3]
- b = [10, a, 20] → a가 문자가 아닌 리스트[1, 2, 3]으로 나타남
- c = ['x', a, 'y'] → a가 문자가 아닌 리스트[1, 2, 3]으로 나타남
- a = [1, 2, 3]에서 1은 인덱스 0, 2는 인덱스 1, 3은 인덱스 2



❖ 파이썬 기초 문형

◆ 객체와 할당

➡ ex 2]

```
a[1] = 1000

print(a)
print(b)
print(c)
```

```
[1, 1000, 3]
[10, [1, 1000, 3], 20]
['x', [1, 1000, 3], 'y']
```

➡ 설 명 2]

a[1] = 1000 → 인덱스 1에 1000을 넣음
인덱스 1에 1000을 넣음 → 2 대신 1000을 넣음
리스트 a를 가리키고 있는 b, c 모두 변경됨
b에 a가 연결되어 있기 때문에 a가 변하면 b도 변함
마찬가지로 c에 a가 연결되어 있기 때문에 a가 변하면 c도 변함

❖ 파이썬 콘솔 입출력

◆ 콘솔 입력

- 콘솔(Console)
  - 윈도우에서는 Command창, 리눅스/맥에서는 Terminal창
  - 각 IDE(예, PyCharm)에서는 별도의 콘솔 창이 제공됨(Python Console)

- 콘솔(Console): 윈도우의 커맨드 창, 리눅스/맥의 터미널 창을 의미
- 콘솔창: 내용을 입력 또는 출력하는 창
- 프로그램 작성 후 실행하면 콘솔창에 내용이 출력됨

- Python 2.x 버전과 3.x 버전의 콘솔 입력함수가 변경이 되었으므로 버전 별로 알아본다.

훈련교사 : 전 은 석
email : euns_jun@naver.com

❖ 파이썬 콘솔 입출력

◆ 콘솔 입력(Python 2.x)

- raw_input() : 입력한 값을 문자열로 반환해주는 함수.
- input() : 입력한 데이터를 평가하여(연산을 해서..) 그 결과를 반환해준다.

➡ ex 1]

```
name = raw_input('name : ')
```

```
name : 홍길동
```

➡ 설 명 1]

- raw_input() → 콘솔창에서 문자열을 입력받는 내장 함수
- name = raw_input('name : ') → 일단 콘솔창에 name : 나타남
- 콘솔창에 나타난 name : 뒤에 홍길동 입력 → name에 홍길동 할당됨

➡ ex 2]

```
print name
```

```
홍길동
```

➡ 설 명 2]

- 그 후 print name → name에 할당된 홍길동이 출력됨
- raw_input() → 콘솔창에서 문자열을 입력받는 내장 함수
- raw_input('텍스트') → 콘솔창에 텍스트 나타남
- raw_input() → 콘솔창에 1도 입력 가능
- 이 때 1은 숫자 1이 아닌 문자로써의 1
- 파이썬에서는 캐릭터와 문자열을 구분하지 않고 문자열로 취급
- 따라서 문자열을 입력받는 raw_input() 함수임에도 1이 입·출력된 것

❖ 파이썬 콘솔 입출력

◆ 콘솔 입력(Python 2.x)

- int() : 문자열을 정수로 반환해주는 함수.

➡ ex 3]

```
k = int(raw_input('int : '))
print k
```

```
int : 12
12
```

➡ 설 명 3]

- int(raw_input()) → 콘솔창에 입력된 값을 int 내장 함수에 넣음
- int() → 문자열을 숫자(정수)로 바꿈
- 이런 경우 raw_input() 값을 숫자 형태로 받아야 함
- raw_input('int:') → 콘솔창에 int: 나타남
- 콘솔창에 숫자 10 입력 후 엔터 → print k 실행 → 숫자 10 출력
- print k + 10 → k가 문자가 아닌 숫자이므로 연산이 가능함
- k는 int 내장 함수가 반환한 숫자 10이 들어있으므로 연산 가능
- raw_input() → 문자열을 입력받음 / int() → 문자열을 정수로 반환함

❖ 파이썬 콘솔 입출력

◆ 콘솔 입력(Python 2.x)

- input() : 정수, 실수, Expression(표현식 또는 연산식) 입력 내장함수

➡ ex 4]

```
i = input('int : ')
print i
print type(i)
```

```
int : 45
45
<type 'int'>
```

➡ 설 명 4]

- 이전 예제와 같이 input() 내장 함수를 활용
- 반환값의 타입은 입력값에 따라 평가 후 결정된다.

➡ ex 5]

```
k = input('expr : ')
print k
```

```
expr : 30 + 50
80
```

➡ 설 명 5]

- input() → 문자열 입력받는 raw_input()과 달리 연산식까지 입력 가능

❖ 파이썬 콘솔 입출력

◆ 콘솔 입력(Python 3.x)

- input() : Python 3.x 에서의 input() 는 2.x 에서의 raw_input()과 비슷하다.
- 3.x 에서 raw_input() 은 존재하지 않는다.
- 반환값은 문자열 처리된다.

➡ ex 1]

```
l = input('int : ')
print(l)
print(type(l))
```

```
int : 45
'45'
<class 'str'>
```

➡ 설 명 1]

- 2.x 버전 예제와 같이 input() 내장 함수를 활용
- 반환값의 타입은 문자열로 결정된다.

➡ ex 2]

```
no1 = int(input('no1 : '))
no2 = int(input('no2 : '))
print(no1 + no2)
print(type(no1), type(no2))
```

```
no1 : 30
no2 : 50
80
<class 'int'> <class 'int'>
```

➡ 설 명 2]

- 입력받은 값을 정수로 변환해야 할 경우 int() 사용해서 처리한다.

❖ 파이썬 콘솔 입출력

◆ 콘솔 출력

- print 화면에 자료를 출력하는 보편적인 statement
- 여러 자료를 한꺼번에 출력할 때에는 콤마(,)를 사용

➡ ex 1]

```
print(4 + 5, 4 - 2)
```

```
9 2
```

➡ 설 명 1]

- 콘솔창에 단순한 숫자가 아닌 식을 입력
- 콘솔창에서 식 입력 후 엔터 → 식이 계산된 결과 출력되어 변수에 할당
- 콘솔 출력은 print 예약어(Keyword)를 활용하여 가능
- print(4 + 5, 4 – 2) → 9 2
- print() 수행 시 콤마(,) → 한 칸 띄어주는 역할
- 콤마(,) → 여러 자료를 한 번에 출력할 때 결과 사이를 한 칸 씩 띄어줌

❖ 파이썬 콘솔 입출력

◆ 콘솔 출력

- 세미콜론(;) 또는 콤마(,)는 순차적으로 입력된 각 statement를 분리함

➡ ex 2]

```
print(1);print(2)
```

```
1
2
```

➡ 설 명 2]

- 세미콜론(;) → 각 문장의 줄을 분리
- 파이썬에서 한 문장은 기본적으로 한 줄을 차지
- 세미콜론(;) → 한 줄에 있는 두 문장을 두 줄에 나타나게 함
- 기본적으로 print는 한 줄을 바꾸어 줌, 즉, 개행이 됨

- 콤마(,)는 2.x 버전의 경우 줄바꿈 하지 않는다.

❖ 파이썬 콘솔 입출력

◆ 콘솔 출력

• + 연산자는 숫자와 문자열에 대한 연산을 지원하지 않는다.

➡ ex 3]

```
print(12 + 'spam')
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

➡ 설 명 3]

- 더하기(+) 연산자는 숫자와 문자를 합쳐주지 않음
- 파이썬은 수치형 자료와 문자형 자료를 합할 수 없음

➡ ex 4]

```
print(str(12) + 'spam')
```

```
12spam
```

➡ 설 명 4]

- print(12 + ‘spam’) → print(‘12’ + ‘spam’) (숫자를 문자로 수정)
- print(12 + ‘spam’) → print(str(12) + ‘spam’) (str() 함수 사용)
- str() → 숫자 12를 문자 12로 바꿔주는 내장 함수

훈련교사 : 전 은 석
email : euns_jun@naver.com

파이썬 자료형
(Data Type)

훈련교사 : 전 은 석
email : euns_jun@naver.com

❖ 자료형

◆ 자료형의 정의

- Data Type
- 데이터 또는 자료의 형식
- 데이터의 분류 체계
- 동적 형식(Dynamic type) : 프로그램 실행 시 작동
  - 동적 형식 언어(Danamic typed language)에서 사용
  - 파이썬의 자료형
- 정적 형식(Static type) : 프로그램 실행 전 작동
  - 정적 형식 언어(Static typed language)에서 사용

➡ **ex 1]**

```
print(type(1234))
```

```
<class 'int'>
```

➡ **설 명 1]**

- type(a) : 데이터의 자료형을 반환해주는 함수
- a 에 변수를 입력해서 변수의 자료형을 알 수 있다.
  - 변수 대신 데이터를 위 코드처럼 직접 입력해서 사용할 수도 있다.

❖ 수치 자료형

◆ 종류

- 정수(int), 실수(float), 복소수(complex)

◆ 정수형(int - integer(정수) 형식)

➡ ex 1]

```
a = 23    # 10진 정수
b = 0o23  # 8진 정수
c = 0x23  # 16진 정수
d = 0b10111 #2진  정수
print(type(a), type(b), type(c), type(d))
print(a, b, c, d)
```

```
<class 'int'> <class 'int'> <class 'int'> <class 'int'>
23 19 35 23
```

➡ 설 명 1]

- 정수형 상수: 소수 영역이 없는 수
- a=23 → a에 23이라는 정수형 상수 할당(23은 10진 정수)
- 숫자 앞에 숫자 0과 알파벳 o를 붙이면(0o) 8진 정수
- 숫자 앞에 0x(숫자 0, 문자 x)를 붙이면 16진 정수
- a, b, c의 타입 → int(정수형 상수)
- print a, b, c → a, b, c에 할당된 정수가 출력됨
- 10진 정수는 그대로 출력
- 8진·16진 정수는 10진 정수로 값이 바뀌어 출력

훈련교사 : 전 은 석
email : euns_jun@naver.com

❖ 수치 자료형

◆ 정수형(int - integer(정수) 형식)

* 정수표현 접두사

- 0x - 16진수 (Hexadecimal Number)
- 0b - 2진수 (Binary Number)
- 0o - 8진수 (Octal Number)

* 진수 변환 함수

- hex() - 16진수 (Hexadecimal Number)
- bin() - 2진수 (Binary Number)
- oct() - 8진수 (Octal Number)

* 정수의 사칙 연산자 (연산자 : 연산에 사용하는 기호)

- + : 더하기
- - : 빼기
- * : 곱하기
- / : 나누기
- % : 나눈 나머지 구하기
- // : 나눈 몫 구하기
- ** : 제곱

훈련교사 : 전 은 석
email : euns_jun@naver.com

❖ 수치 자료형

◆ 최대 정수

➡ ex 1]

```
import sys
print(sys.maxsize)
```

```
9223372036854775807
```

➡ 설 명 1]

- sys 모듈의 maxsize 를 확인하면 알 수 있음

- 3.x 버전에서는 long 타입이 없이 int 타입으로 처리한다.

➡ ex 2]

```
import sys
no1 = sys.maxsize
no2 = sys.maxsize + 1
print(no1)
print(no2)
print(type(no1))
print(type(no2))
```

```
9223372036854775807
9223372036854775808
<class 'int'>
<class 'int'>
```

➡ 설 명 2]

- sys.maxsize 에 1을 더한 no2의 타입도 <class 'int'> 로 출력이 된다.
- 결론 : 정수형 타입은 int 타입으로 사용 하면 된다.

훈련교사 : 전 은 석
email : euns_jun@naver.com

❖ 수치 자료형

◆ 실수형(float - Real Number(실수) 형식)

- Python 은 실수를 지원하기 위해 부동 소수형을 제공
  - 부동 소수형 : 소수점을 이동시켜서 소수를 표현하는 자료형
- 부동소수형은 **8바이트(Byte)만을 이용**해서 수를 표현(한정된 수만 표현 가능)
  - 디지털 방식으로 수소를 표현해야 하므로 **정밀도에 한계**가 있다.

➡ ex 1]

```
a = 1.2
b = 3.5e3
c = -0.2e-4
print(type(a), type(b), type(c))
print(a, b, c)
```

```
<class 'float'> <class 'float'> <class 'float'>
1.2 3500.0 -2e-05
```

➡ 설 명 1]

- 실수형 상수: 소수 영역이 있는 수
- 3.5e3=3.5*10^3=3500(e3 → 10^3 = 1000)
- 실수형 상수는 3500이 아닌 3500.0
- e-4 → 10^{-4} = 1/10000
- -0.2e-4 = -0.2*(1/10000)=-2*(1/100000)=-2e-5
- a, b, c의 타입 → float(실수형 상수)
- print(a, b, c) → 각 표현 방식에 따라 출력됨
- 3.5e3 → 3500.0
- -0.2e-4 → -2e-5
- print로 인한 변화는 str(repr) 내장 함수에 의한 것

- 파이썬에서 정수는
  컴퓨터의 성능이 지원하는 한
  모든 정수를 다룰 수 있다.
- 부동 소수점은
  정밀도와 범위에 한계가 있다.

❖ 수치 자료형

◆ 복소수형(complex - Complex Number(복소수) 형식)

- 복소수 : 실수부(ex. 10)와 허수부(ex. 20j) 로 구성
- 연산은 실수부와 허수부가 각각 계산된다.

➡ ex 1]

```
a = 10 + 20j
print(a)
b = 10 + 5j
print(a + b)
```

```
(10+20j)
(20+25j)
```

➡ 설 명 1]

- 변수 a와 b의 실수부와 허수부가 각각 계산된다.

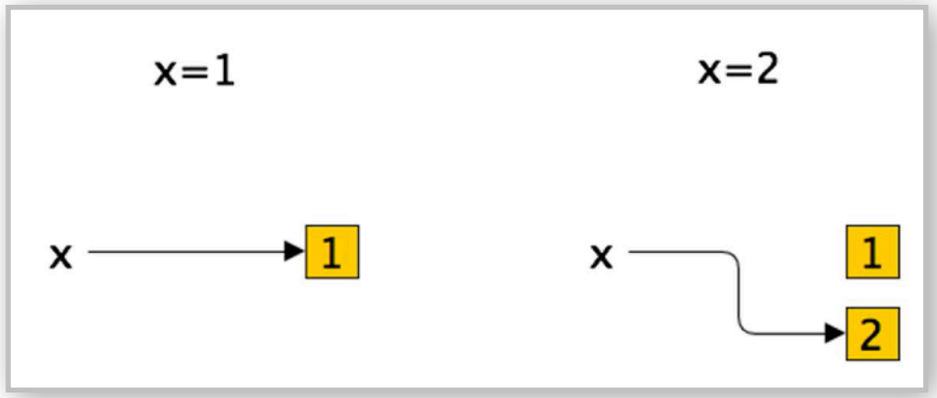
❖ 수치 자료형

◆ 수치 자료형의 치환

➡ ex 1]

```
x = 1
x = 2
```

➡ 설 명 1]



- x=1로 입력한 후 x=2로 치환할 때
- x=1 입력 → x가 1을 가리키도록 하는 주소(레퍼런스) 생김
- 그 후 x=2 입력 → x가 2를 가리키도록 하는 주소(레퍼런스) 생김
- 그리고 기존에 1을 가리키던 주소는 없어짐
- 이 때 1은 쓰레기(garbage)가 되어 수집 가능

❖ 수치 자료형

◆ 수치 연산과 관련된 내장 함수

➡ ex 1]

```
print(abs(-3))
print(round(3.14))
print(int(3.141592))
print(int(-3.1415))
print(float(5))
print(complex(3.4, 5))
print(complex(6))
```

```
3
3
3
-3
5.0
(3.4+5j)
(6+0j)
```

➡ 설 명 1]

- print(abs(숫자)) → 숫자의 절대값이 출력됨
- print(round(실수)) → 입력된 실수를 소수 첫자리에서 반올림해줌
- print(int(숫자)) → 숫자의 정수형이 출력됨
- -3.9999가 -4에 가깝지만 정수형은 -3
- print(float(숫자)) → 숫자의 실수형이 출력됨
- print(complex(a, b)) → (a+bj)와 같이 복소수형이 출력됨
- print(complex(a)) → (a+0j)와 같이 실수부에만 숫자 나타남

❖ 수치 자료형

◆ 수치 연산과 관련된 내장 함수

➡ ex 2]

```
print(divmod(5, 2))
print(pow(2, 3))
print(pow(2.3, 3.5))
```

```
(2, 1)
8
18.4521691056
```

➡ 설 명 2]

- print(divmod(a, b)) → (a를 b로 나눈 몫, a를 b로 나눈 나머지)
- print(pow(a, b)) → a의 b승 값이 출력됨
- pow() 함수는 실수(소수 부분이 있는 수)도 입력이 가능함

❖ 수치 자료형

◆ math 모듈의 수치 연산 함수

➡ ex 1]

```
import math
print(math.pi)
print(math.e)

print(math.floor(math.pi))
print(math.trunc(math.pi))

# 1.0 라디안에 대한 사인 값
print(math.sin(1.0))

print(math.sqrt(2)) # 제곱근
```

```
3.14159265359
2.71828182846

3
3

0.841470984808

1.41421356237
```

➡ 설 명 1]

- **math** 모듈은 **수학적으로 정의된 변수, 함수** 지원됨
- math.pi → 파이 값
- math.e → 지수 값
- math.floor() / math.trunc() → 소수점 이하 버림
- math.sin() → 싸인 값
- math.sqrt() → 제곱근 값

❖ 수치 자료형

◆ math 모듈의 수치 연산 함수

➡ ex 2]

```
r = 5.0                                                  # 반지름
a = math.pi * r * r                                      # 면적
degree = 60.0
rad = math.pi * degree / 180.0                           # 각도를 라디안으로 변환
print(math.sin(rad), math.cos(rad), math.tan(rad))       #sin, cos, tan
```

```
0.866025403784 0.5 1.73205080757
```

➡ 설 명 2]

- 반지름이 5인 원 a의 면적을 구하는 공식
- 각도가 60도일 때 라디안 값을 구하는 공식
- 그 후 라디안 값을 싸인, 코싸인, 탄젠트 값으로 출력
- math 모듈로 여러 가지 수치 연산 활용 가능

❖ 문자열 자료형

- 문자열은 변경 불가능한 자료형이다.(Immutable Type)

◆ 형식

- ''
- ""

➡ ex 1]

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

```
Hello World!
Hello World!
```

➡ 설 명 1]

- 문자열을 만들 때에는 단일 따옴표 또는 이중 따옴표를 사용
- 두 따옴표는 사용 시 차이 없으므로 선택적으로 사용 가능
  → Hello “World” 를 출력하려면 단일 따옴표로 문자열 생성
  → Hello 'World' 를 출력하려면 이중 따옴표로 문자열 생성

❖ 문자열 자료형

◆ Multi Line 형식

- ''' '''
- """ """

➡ ex 1]

```
multiline = '''
To be, or not to be
that is the question
'''

print(multiline)

multiline2 = """
To be, or not to be
that is the question
"""
print(multiline2)
```

```
To be, or not to be
that is the question

To be, or not to be
that is the question
```

➡ 설 명 1]

- 여러 줄의 문자열을 만들 때에는 ’’’ ’’’ 또는 ”””” ”””” 사용
- 첫 번째 ’’’ 뒤에 개행 문자가 있음
- 문자열 첫 번째 줄 바꿈이 된 이유는 개행 문자 때문
- 마지막 ’’’ 앞에 개행 문자가 있음
- 문자열 마지막 줄 바꿈이 된 이유도 개행 문자 때문
- 이중 따옴표 세 개로 만든 여러 줄의 문자열
- 마찬가지로 개행 문자가 처음과 끝에 있음

❖ 문자열 자료형

◆ 인덱싱

➡ ex 1]

```
s = "Hello world!"
print(s[0])
print(s[1])
print(s[-1])
print(s[-2])
```

```
H
e
!
d
```

➡ 설 명 1]

- 문자열은 시퀀스(순차자료)형으로 인덱스가 붙음
-

문자	H	e	l	l	o	(공 백)	w	o	r	l	d	!
인덱스	0	1	2	3	4	5	6	7	8	9	10	11

- 인덱싱: 꺽쇠괄호([] : 대괄호) 안에 인덱스 번호 입력
- print(s[0]) → 문자열 s에서 인덱스 0인 문자(H) 출력
- print(s[1]) → 문자열 s에서 인덱스 1인 문자(e) 출력
- print(s[11]) → 문자열 s에서 인덱스 11인 문자(!) 출력
- 인덱스 앞에 '–(마이너스)' 붙이면 뒤에서부터 셈

❖ 문자열 자료형

◆ 슬라이싱

• 문자열[시작인덱스(포함):끝낼 인덱스(제외): 스텝]

➡ ex 1]

```
s = "Hello world!"
print(s[1:3])
print(s[0:5])
```

```
el
Hello
```

➡ 설 명 1]

- 슬라이싱: 꺽쇠괄호([]) 안에 인덱스 번호와 콜론(:) 입력
- [시작할 인덱스(포함) : 끝낼 인덱스(제외) : 스텝]
- S[1:3] → 인덱스 1인 문자(포함)부터 3인 문자(제외)까지
- 슬라이싱이란 자르기
- [1:3] → 인덱스 1(e)부터 인덱스 3 앞(l)까지 자르기
- 인덱스 3(두 번째 l)은 제외, 인덱스 2(첫 번째 l)까지만 출력
- [0:5] → 인덱스 0(H)부터 인덱스 5 앞(o)까지 자르기

❖ 문자열 자료형

◆ 슬라이싱

➜ ex 2]

```
s = 'Hello'
print(s[1:])
print(s[:3])
print(s[:])
```

```
ello
Hel
Hello
```

➜ 설 명 2]

- 콜론(:)만 있고 값이 없을 경우 → 기본값으로 사용
- [1:] → 인덱스 1(H)부터 마지막(자료형의 크기)까지
- 자료형의 크기는 5 → [1:]는 [1:5]와 같음
- [1:5] → 인덱스 1(e)부터 인덱스 5앞(o)까지 자르기
- [:3] → 처음(인덱스 0)부터 인덱스 3앞(l)까지
- [:] → 전체 문자열(슬라이싱 하지 않음)

❖ 문자열 자료형

◆ 슬라이싱

➡ ex 3]

```
s = 'abcd'
print(s[::2])
print(s[::-1])
```

```
ac
dcba
```

➡ 설 명 3]

- [::값] → 해당 값으로 스텝
- H(인덱스 1)와 e(인덱스 2)의 차이는 1 ← 이 차이가 스텝
- 기본적인 인덱스의 차이(스텝)는 1
- 스텝 2 → 인덱스 차이가 2씩 나도록 슬라이싱
- [::2] → 처음부터 마지막까지 인덱스 차이가 2씩 나도록 슬라이싱
- 인덱스 0(a)을 가져온 다음 인덱스 2(c)를 가져옴
- 인덱스 2(c)를 가져온 다음 인덱스 4를 가져옴(없으므로 끝)
- 스텝의 값만큼 문자의 인덱스 차이가 나도록 슬라이싱
- 스텝의 값이 마이너스 → 인덱스의 차이가 마이너스
- 문자열을 거꾸로 가져오게 됨
- 2(c의 인덱스) - 3(d의 인덱스) = -1
- 1(b의 인덱스) - 2(c의 인덱스) = -1
- **[::-1] → 문자열을 거꾸로 뒤집어(역순으로) 가져옴**

❖ 문자열 자료형

◆ 슬라이싱

➡ ex 4]

```
s = 'Hello World'
s[0] = 'h'
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

➡ 설 명 4]

- **문자열 자료형은 내부 내용 변경 불가능**
- s[0] = h → 문자열 s의 인덱스 0을 h로 변경한다는 의미
- s[0] = h → 수행해보면 에러 발생
- 문자열 내의 문자를 바꿀 수 없으므로 주의

❖ 문자열 자료형

◆ 슬라이싱

➜ ex 5]

```
s = 'Hello World'
s = 'h' + s[1:]
s
```

'hello World'

➜ 설 명 5]

- 문자열을 바꾸기 위해서는 슬라이싱을 활용
- ‘h’ + s[1:] → 문자 h와 인덱스 1부터 슬라이싱한 결과를 합함
- ‘h’ + s[1:] = h + ello World = hello World
- 문자열의 기존 내용이 바뀐 것이 아님
- 새로운 문자열이 구성되어 할당된 것
- 따라서 기존 문자열은 쓰레기(garbage)

❖ 문자열 자료형

◆ 문자열 연산

➡ ex 1]

```
print('Hello' + '' + 'World')
print('Hello' * 3)
print('-' * 10)
```

```
HelloWorld
HelloHelloHello
----------
```

➡ 설 명 1]

- 문자열 + 문자열 → 순서 그대로 두 문자열 연결
- print('Hello' + ''(공백없음) + 'World') → HelloWorld
- 문자열 * 숫자 → 숫자만큼 문자열 반복
- print('Hello' * 3) → HelloHelloHello (Hello 3번 반복)
- print('-' * 10) → -가 10번 반복되어 나타남

◆ 문자열 길이

➡ ex 2]

```
s = 'Hello World'
print(len(s))
```

```
11
```

➡ 설 명 2]

- len(s) → s의 문자열 길이를 의미

❖ 문자열 자료형

◆ 문자열내 포함 관계 여부

➡ ex 1]

```
s = 'Hello World'
print('World' in s)
print('World' not in s)
```

```
True
False
```

➡ 설 명 1]

- **'A' in B → B 안에 A가 있다(결과는 True or False)**
- print('World' in s) → 문자열 s 안에 World가 있다 → True
- 'A' not in B → B 안에 A가 없다(결과는 True or False)
- print('World!' in s) → 문자열 s 안에 World!가 있다 → False
- print(' ' in s) → 문자열 s 안에 공백이 있다 → True

훈련교사 : 전 은 석
email : euns_jun@naver.com

파이썬 연산자

❖ 기본 연산자(Basic Operators)

- 연산자 : 연산을 수행할 때 사용하는 기호

◆ 종류

- 산술 연산자 (Arithmetic Operators)
- 비교 (즉, 관계형) 연산자 (Comparison (i.e., Relational) Operators)
- 할당 연산자 (Assignment Operators)
- 논리 연산자 (Logical Operators)
- 비트 연산자 (Bitwise Operators)
- 구성원 연산자 (Membership Operators)
- 식별 연산자 (Identity Operators)

❖ 기본 연산자(Basic Operators)

◆ 산술 연산자 (Arithmetic Operators)

- 수학적인 계산에 사용하는 연산자(기호)

✻ a = 10, b = 20, c = 3 이라 가정한다.

Operator	Description	Example
+	더하기	a + b = 30
-	빼기	a - b = -10
*	곱하기	a * b = 200
/	나누기	b / a = 2.0
%	나머지	b % a = 0
**	제곱	a ** c = 1000
//	몫	a // c = 3

❖ 기본 연산자(Basic Operators)

◆ 비교 연산자 (Comparison Operators)

- 두개의 피연산자를 비교할 때 사용하는 연산자

✻ a = 10, b = 20 이라 가정한다.

Operator	Description	Example
==	값이 동일하다	(a == b) → false
!=	값이 동일하지 않다	(a != b) → true
>	왼쪽 값이 오른쪽 값보다 크다	(a > b) → false
<	왼쪽 값이 오른쪽 값보다 작다	(a < b) → true
>=	왼쪽 값이 오른쪽 값보다 크거나 동일하다	(a >= b) → false
<=	왼쪽 값이 오른쪽 값보다 작거나 동일하다	(a <= b) → true

❖ 기본 연산자(Basic Operators)

◆ 할당 연산자 (Assignment Operators)

- 변수에 데이터를 입력할 때 사용하는 연산자(기호)

✻ a = 10, b = 20 이라 가정한다.

Operator	Description	Example
=	왼쪽 변수에 오른쪽 값을 할당한다	c = a + b → c = a + b
+=	왼쪽 변수에 오른쪽 값을 더하고 결과를 왼쪽변수에 할당	c += a → c = c + a
-=	왼쪽 변수에서 오른쪽 값을 빼고 결과를 왼쪽변수에 할당	c -= a → c = c - a
*=	왼쪽 변수에 오른쪽 값을 곱하고 결과를 왼쪽변수에 할당	c *= a → c = c * a
/=	왼쪽 변수에서 오른쪽 값을 나누고 결과를 왼쪽변수에 할당	c /= a → c = c / a
%=	왼쪽 변수에서 오른쪽 값을 나눈 나머지의 결과를 왼쪽변수에 할당	c %= a → c = c % a
**=	왼쪽 변수에 오른쪽 값만큼 제곱을 하고 결과를 왼쪽변수에 할당	c **= a → c = c ** a
//=	왼쪽 변수에서 오른쪽 값을 나눈 몫의 결과를 왼쪽변수에 할당	c //= a → c = c // a

❖ 기본 연산자(Basic Operators)

◆ 비트 연산자 (Bitwise Operators)

- 비트 연산에 사용하는 연산자(기호)

Operator	Description	Example
&	AND 연산. 둘다 참일때만 만족	(a & b) = 12 → 0000 1100
|	OR 연산. 둘 중 하나만 참이여도 만족	(a | b) = 61 → 0011 1101
^	XOR 연산. 둘 중 하나만 참일 때 만족	(a ^ b) = 49 → 0011 0001
~	보수 연산(not 연산).	(~a) = -61 → 1100 0011
<<	왼쪽 시프트 연산자. 변수의 값을 왼쪽으로 지정된 비트 수 만큼 이동	a << 2 = 240 → 1111 0000
>>	오른쪽 시프트 연산자. 변수의 값을 오른쪽으로 지정된 비트 수 만큼 이동	a >> 2 = 15 → 0000 1111

✻ a = 60, b = 13 이라 가정한다. a = 0011 1100 b = 0000 1101

❖ 기본 연산자(Basic Operators)

◆ 삼항 연산자 (Ternary Operators)

✤ 형식

```
참인경우 값  if  조건  else  거짓인경우 값
```

- 연산 대상의 개수에 따라 연산자를 분리하면 단항 연산자, 이항 연산자, 삼항 연산자로 분리 합니다.
- 단항 연산자는 부호(+, -), not 등이 있으며 +, -, *, / .... 등 대부분의 연산자가 이항 연산자 입니다.
- 삼항 연산자는 1개가 존재 합니다.

➡ ex 1]

```
year = 2000;
isLeapYear = year%400==0 or year%4==0 and year%100
print('{0}년은 "{1}"입니다.'.format(year,'윤년' if isLeapYear else '평년'))
```

```
2000년은 "윤년"입니다.
```

❖ 기본 연산자(Basic Operators)

◆ 논리 연산자 (Logical Operators)

✻ a = True, b = False 이라 가정한다.

Operator	Description	Example
and	논리 AND 연산. 둘다 참일때만 참	(a and b) = False
or	논리 OR 연산. 둘 중 하나만 참이여도 참	(a or b) = True
not	논리 NOT 연산. 논리 상태를 반전	not(a and b) = True

◆ 멤버 연산자 (Membership Operators)

✻ a = 10, b = 10, list = {1, 2, 3, 4, 5} 이라 가정한다.

Operator	Description	Example
in	list 내에 포함되어 있으면 참	(a in list) = False
not in	list 내에 포함되어 있지 않으면 참	(b not in list) = True

◆ 식별 연산자 (Identity Operators)

- 두 개체의 메모리 위치를 비교한다.

✻ a = 10, b = 10, list = {1, 2, 3, 4, 5} 이라 가정한다.

Operator	Description	Example
is	개체메모리 위치나 값이 같다면 참	(a is b) = True
is not	개체메모리 위치나 값이 같지 않다면 참	(a is not b) = False

❖ 기본 연산자(Basic Operators)

◆ 연산자 우선순위(Operators Precedence)

Operator	Description
**	지수
~ + -	Ccomplement, 단항 플러스와 마이너스 (마지막 두의 메서드 이름은 + @이며, - @)
* / % //	곱하기, 나누기, 나머지, 몫
+ -	덧셈과 뺄셈
>> <<	좌우 비트 시프트
&	비트 'AND'
^ |	비트 전용 'OR'와 정기적 인 'OR'
<= < > >=	비교 연산자
<> == !=	평등 연산자(동등비교 연산자)
%= /= //= -= += *= **=	할당 연산자
is is not	식별 연산자
in not in	맴버 연산자
not or and	논리 연산자