

2016 벤처창업 웹프로그래밍 1

파이썬 Part.

#3

강사 이진석

- 어썸마트 CTO
- Microsoft Azure MVP
- AskDjango Operator
- <https://festi.kr/about/>



<파이썬. Part> 강의 목표

- 파이썬으로 코딩함에 있어서, 두려움을 없애자.



질문 채널

- Slack 을 통한 커뮤니케이션
 - 일종의 그룹 채팅 서비스
 - 초대장 받기 : <http://goo.gl/LsCvAq>
 - 등록이 되시면, #snu-web 채널로 들어오세요.
- 실습에 사용된 소스코드는 Github 저장소를 통해 확인하실 수 있습니다.
- Slack 앱은 웹 뿐만 아니라, iOS/Android 앱들도 제공하고 있습니다.
 - <https://slack.com/downloads/>

네이버 검색은 맛집검색에서만 !!!

- 개발자로 검색은 구글링 & 영문검색
 - 개발자로 찾을 때에는 공식문서부터 !!!
 - 그리고, 소스코드 !!!
 - 최신을 반영함.
 - 프로그램은 소스코드 대로 동작할 수 밖에 없음.

NAVER

신사역 오빠랑

검색

통합검색 블로그 카페 지식iN 이미지 동영상 어학사전 뉴스 더보기

정렬 기간 영역 옵션유지 꺼짐 켜짐 상세검색

블로그

신사역 맛집 로꼬로꼬조개찜 오빠랑 함께한 올 여름 보양식... 2016.08.16. | ↗
신선한 해산물 가득, 그리고 닭한마리 완전 최고 ㅋㅋㅋ 신사역 회식 하기에도 강추 퇴근 후 소주 한잔 하기에도 강추 오빠랑 데이트도 강추 신사역 맛집 중 당연최고!... The Truewoman Show blog.naver.com/um... | 약도 Smart Editor 3.0

신사역 맛집 오빠랑 98한우 2015.05.24. | ↗
신사역 맛집 오빠랑 98한우 윤코미에요 :) 오랜만에 먹방 포스팅! 그간 먹방도 좀 뜻했죠? 정말정말 만족했던 맛집, 미루어두었던 신사역 맛집 오빠랑 98한우... 윤코미 jjjinggaon.blog.me/220367960452 | 블로그 내 검색

오빠랑 신사역 가로수길 떡볶이맛집 다녀옴▶빌라드 스파이시 2015.07.17. | ↗
며칠전 오빠랑 신사역 가로수길 떡볶이맛집 다녀왔어요:) 이름은 빌라드 스파이시!!! 전에 고속터미널갔을 때 파미에스테이션점은 지나가며 보기만 했는데 가로수길... 네뜨르 핸이 매력적인 일상... blog.naver.com/haein... | 블로그 내 검색 | 약도

신사역 맛집 : 오빠랑 신사 용참치 3번째 방문 2015.04.09. | ↗
먹구요 :) 신사역맛집 용참치 오빠랑 세번 째 방문인데 약간 메뉴가 달라졌더라구요! 원래는 매운탕도 나왔었는데 조림이나 그런류로 바뀌어서 나왔어요 :) 이건... 뷔스타 룽이 뷔티놀이 :-) blog.naver.com/lsov... | 블로그 내 검색 | 약도

*신사역 맛집 오빠랑 먹기 좋은 '한성 돈까스' 2013.12.24. | ↗
내가 제목에 신사역 맛집 오빠랑이라고 쓴 이유는!!! 내 블로그 친절히 지켜봐주신분들은 아시겠지만, 컴퓨터에 사진은 많은데 올리는 식당은 다 맛있는집 위주로... 1인칭주인공시점 blog.naver.com/pyc90... | 블로그 내 검색 | 약도

소스코드

- 대개의 유명한 프로젝트는 매뉴얼이 잘 되어있습니다.
 - 하지만, 종종 매뉴얼이 부족한 프로젝트도 있습니다.
 - 이럴 땐, **소스코드가 최고의 매뉴얼입니다.**
- 웹프레임워크 Django : 매뉴얼, 소스코드
- 데이터 분석 라이브러리 Pandas
- Microsoft 의 여러 오픈소스 프로젝트

코드 오류가 발생하면 ???

- 아래 코드를 실행하려하는데,

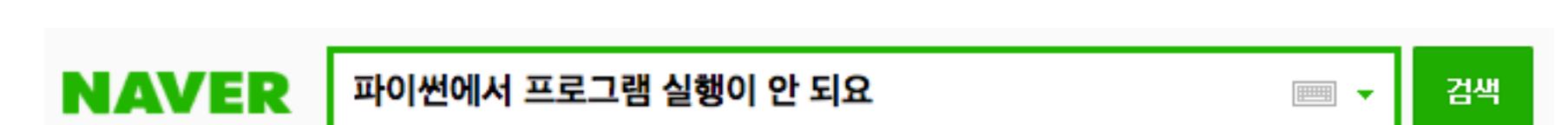
```
print("hello world")
```

- 다음과 같은 오류가 뜹니다. 어떻게 할까요?

```
$ python test1.py
File "test1.py", line 1
    print("hello world")
               ^
SyntaxError: EOL while scanning string literal
```

내 마음을 알아줘.

여기서 이러시면 안 됩니다.



[전체](#) [동영상](#) [이미지](#) [뉴스](#) [지도](#) [더보기 ▾](#) [검색 도구](#)

검색결과 약 30,600개 (0.59초)

관련검색: [파이썬 외부 프로그램 실행](#) [파이썬 다른 프로그램 실행](#) [파이썬 프로그램 실행](#) [파이썬 실행방법](#) [파이썬 실행시간](#)

01-5 파이썬 둘러보기 - 점프 투 파이썬 - WikiDocs

<https://wikidocs.net/9> ▾

파이썬에서 def는 함수를 만들 때 사용하는 예약어이다. 위의 예제는 sum 파일을 저장하면 자동으로 파이썬 프로그램이 실행 된다. 실행 결과는 참이 됩니다. 따라서 무한루프에 빠지게 되죠 ... 이전엔 들여쓰기(indent)가 안 되어서 삽질했습니다.

시작하기 — Introduction to Programming with Python - OpenTechSchool

https://opentechschool.github.io/python-beginners/ko/getting_started.html ▾

파이썬을 아직 못 구했다면, 최신 공식 설치 꾸러미를 다음에서 찾을 수 있어요: ... 윈도우즈에서는, 파이썬`을 경로에 추가하여 다른 프로그램이 찾게 할 수 있어요. 이렇게 하려면 설치 ... Ctrl+B 단축키로 작업중인 파이썬 파일을 바로 실행하게 해주세요.

윈도우에서 파이썬을 설치하고 실행해보자. - 혼자서도 재미있는 자기 ...

selfc.tistory.com/27 ▾

2011. 1. 19. - 다운 받은 python-3.1.3.msi를 실행하면 다음 화면이 나타납니다. ... 시작 > 프로그램 > Python 3.1 이 만 들어진 것을 확인할 수 있고 안을 보면 다음 ...

[Python101] 000. 파이썬 실행하는 법 - Wireframe - 숨은 아직 20대

soooprime.com/wp/archives/2897 ▾

2012. 4. 21. - 파이썬을 그냥 실행되면 “대화형 쉘” 형태로 실행이 된다. ... 하지만 실제로 “재사용 가능한 프로그램”을 만드려면 코드를 작성하여 저장하고 이를 실행할 ... 파이썬 스크립트를 작성했다면, 이는 터미널에서 다음과 같이 실행할 수 있다.

NAVER

파이썬에서 프로그램 실행이 안 되요

통합검색

블로그

카페

지식iN

이미지

동영상

어학사전

뉴스

더보기

정렬

기간

영역

옵션유지 | 상세검색

검색어제안

파이썬에서 프로그램 실행이 안될때로 검색하시겠습니까? [파이썬에서 프로그램 실행이 안될때 검색결과 보기](#)

지식iN

[지식iN에 물어보기](#)

Q <내공 100>.PYC 실행하는법(파이썬 실행파일) 2015.02.13.

제가 금번에 써야 할 프로그램이 있는데 그게 확장자가 .pyc/.py 이렇게 되어있더라구요 여러 가지가 있구요 풀더도 있고, 처음에 월실행해 아울질 몰라서 찾아보다가 파이썬파일이라는...

A 여러 가지 가능성이 있습니다만, 파일 안에 에러가 있거나, 파일이 콘솔로 실행하는 포로그램일 경우가 있습니다. 확인 방법은, 먼저 그 파일이 있는 상위...

컴퓨터통신 > 소프트웨어 > 웹브라우저 | 1 · 0

Q 파이썬 서버 실행문제 답변부탁드려요! 2015.01.13.

제가 몇달전에 채팅 서버 구현하면서 파이썬과 자바 스크립트를... http를 실행을 하면 어디서든 채팅이 가능한 그런 프로그램이었어요. 다시 공부하려고 구현을...

A serve_forever() [/code] 만약 파이썬을 사용한 cgi가 필요하면: [code] # -*- coding:utf-8 -*- import os, sys from http.server import...

컴퓨터통신 > 웹사이트 제작, 운영 > 웹 서버구축, 호스팅 | 1 · 0

Q 프로그램 실행이안되요. 2013.09.17.

어느날부터 백신프로그램도 실행이 안되고, 게임도 안열리고(리그오브레전드 할 때 처음 리그오브레전드 표시는 뜨는데 로그인 창이 안열리고 그냥 꺼짐) 프로그램들도 많이...

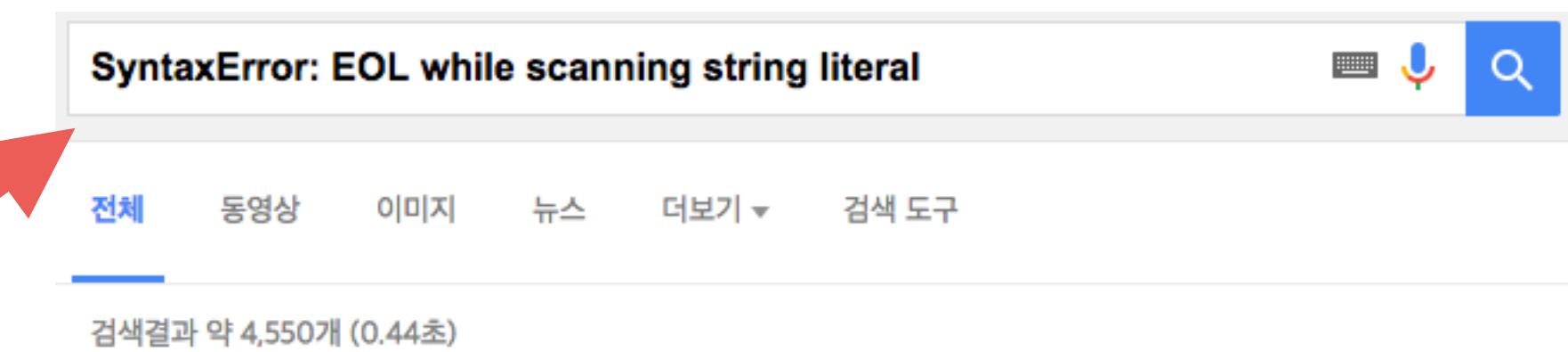
A 아래 답변을 참조해서 백신들이 구동이 되게 조치한 다음에 안 되는 프로그램들은... 혹시라도 정상 모드에서는 하우스콜 런처조차도 실행이 안 될 시에는 안전모드...

파워지식iN 답변 나만큼만답변해봐 | 컴퓨터통신 > 보안 > 바이러스 | 1 · 0

[지식iN 더보기](#)

검색은 오류내용으로 정확히 영문 검색

```
$ python test1.py
File "test1.py", line 1
    print("hello world)
          ^
SyntaxError: EOL while scanning string literal
```



관련검색: python syntaxerror eol while scanning string literal eol while scanning string literal python
eol while scanning string literal found newline while scanning string literal
pcc s 02021 found newline while scanning string literal

- [python: SyntaxError: EOL while scanning string literal - Stack Overflow](#)
stackoverflow.com/.../python-syntaxerror-eol-while-scanning-stri... ▾ 이 페이지 번역하기
2010. 8. 24. - I have the above mentioned error in s1="some very long string....." ... you are not putting a "
before the end of the line. use """" a very long string ...
- [python: SyntaxError: EOL while scanning string literal - Stack Overflow](#)
stackoverflow.com/.../python-syntaxerror-eol-while-scanning-stri... ▾ 이 페이지 번역하기
I had this problem - I eventually worked out that the reason was that I'd included \ characters in the string. If
you have any of these, "escape" them with \\ and it ...

- [What's an EOL error and how do I fix it? | Codecademy](#)
[https://www.codecademy.com/en/.../52c0b0977c82ca009d00603... ▾](https://www.codecademy.com/en/.../52c0b0977c82ca009d00603...) 이 페이지 번역하기
2016. 6. 2. - When I try to submit my code I get an error pointing to the end of line one "SyntaxError: EOL
while scanning string literal". What does that mean? What should be ...
- Stuck on Conditionals 2016년 6월 11일
- Error when writing sentence over 2 lines 2016년 6월 9일
- Sending a ridiculous error 2016년 6월 3일
- 5/9 SyntaxError: EOL while scanning string literal 2016년 6월 2일

- [\[PDF\] CSE117 프로그래밍기초 강의노트 11. 문자열, 수, 변수, 입출력 Strings ...](#)
[plasse.hanyang.ac.kr/class/cse117/2013/Notes/1.pdf ▾](plasse.hanyang.ac.kr/class/cse117/2013/Notes/1.pdf)
- SyntaxError: EOL while scanning string literal. 같은 모양의 따옴표로 둘러싸지 ... 오류syntax error가 발생했다는
오류메시지와 함께 비정상적으로 계산을 종료한다.

파이썬 (Python, 피팅)

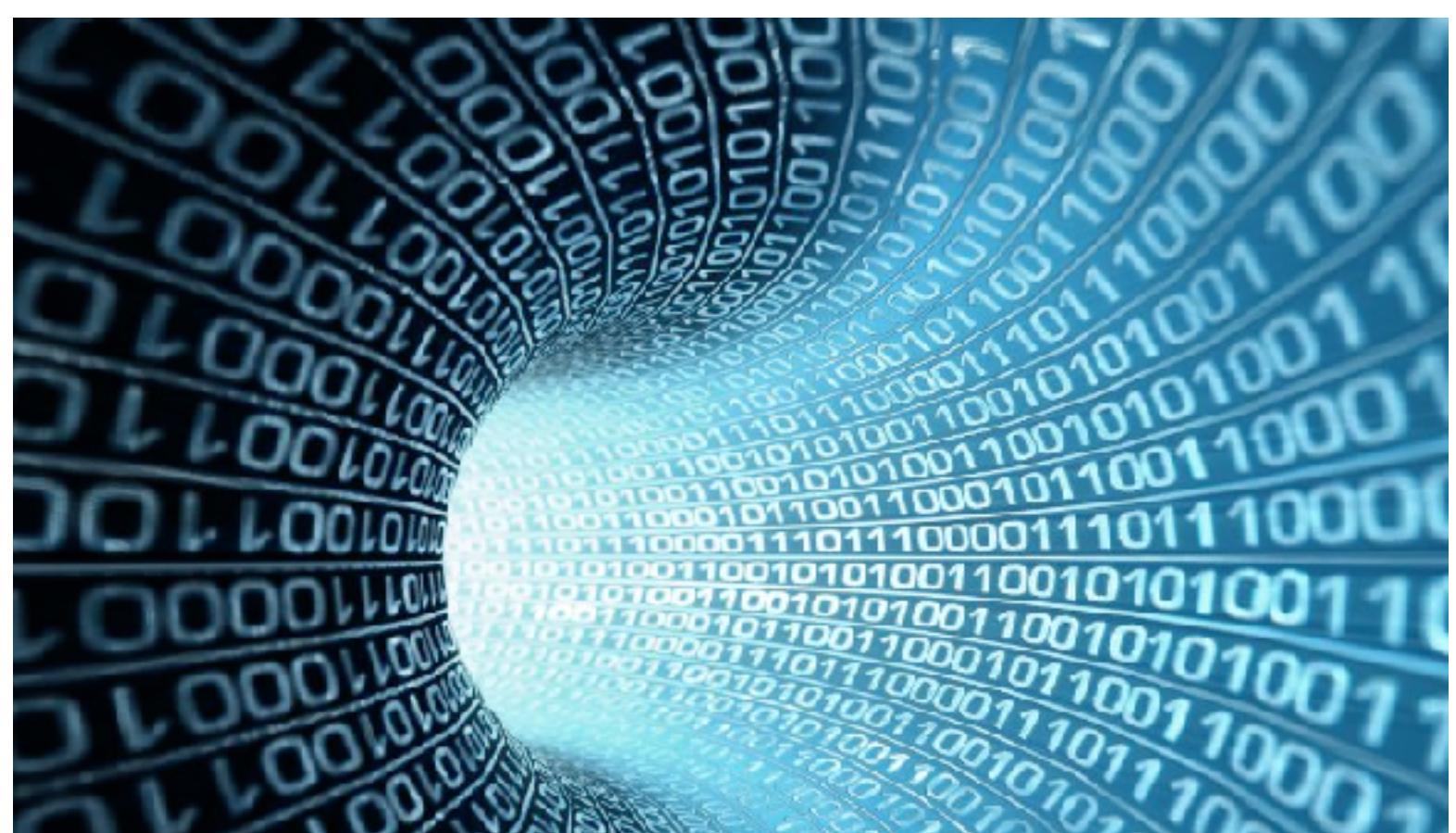
먼저 받아둡시다.

- Anaconda Python 3.5, 64비트 다운로드
 - <https://www.continuum.io/downloads>
- USB 디스크
 - 맥 : Anaconda3-4.1.1-MacOSX-x86_64.pkg
 - 윈도우 64비트 : Anaconda3-4.1.1-Windows-x86_64.exe
 - 윈도우 32비트 : Anaconda3-4.1.1-Windows-x86.exe



프로그래밍 언어 (Programming Language)

- 다양한 "사람의 언어"
 - 한국어, 영어, 중국어, 일본어, 라틴어, 프랑스어, 스페인어, 아랍어, 러시아어, etc.
 - 다양한 "프로그래밍 언어"
 - 컴퓨터에게 일을 시키기 위한 (명령하기 위한) 언어



사람 ————— 컴퓨터

자미로 보는 언어 랭킹 (전 분야)

- <http://www.tiobe.com/tiobe-index/>

- 절대적인 순위는 아닙니다. 산업군마다 많이 쓰이는 언어가 다릅니다.

Sep 2016	Sep 2015	Change	Programming Language	Ratings	Change
1	1		Java	18.236%	-1.33%
2	2		C	10.955%	-4.67%
3	3		C++	6.657%	-0.13%
4	4		C#	5.493%	+0.58%
5	5		Python	4.302%	+0.64%
6	7	▲	JavaScript	2.929%	+0.59%
7	6	▼	PHP	2.847%	+0.32%
8	11	▲	Assembly language	2.417%	+0.61%
9	8	▼	Visual Basic .NET	2.343%	+0.28%
10	9	▼	Perl	2.333%	+0.43%

분야별 많이 쓰이는 언어 (메인 언어)

- 스타트업에서 웹 서비스 개발을 하고 싶어요 : Python or RubyOnRails
- 데이터 분석을 하고 싶어요. : Python or R
- 웹 프론트엔드 개발을 하고 싶어요 : html, css, javascript, etc.
- 큰 회사에서 웹 서비스 개발을 하고 싶어요. : Java
- Android 앱을 개발하고 싶어요. : Java
- iOS 앱을 개발하고 싶어요. : Objective-C, Python

파이썬 (Python)

- 1991년 구도 반 로섬 (Guido van Rossum) 이 발표한 프로그래밍 언어
 - Google (2005년 ~ 2012년), Dropbox (2013년 ~)
- 파이썬 소프트웨어 재단 (Python Software Foundation)에서 관리
- 범용 프로그래밍 언어
- 다양한 플랫폼 지원 (윈도우, 맥, 리눅스, etc)
 - 그렇지만, **리눅스를 가장 잘 지원합니다.**
- 풍부한 라이브러리 : 요즘 데이터분석에서 강세 (Pandas 라이브러리)
- 대학을 비롯한 여러 교육 기관, 연구 기관 및 산업계에서 이용이 증가



파이썬으로 할 수 있는 것들

- 윈도우/맥/리눅스에서 모두 구동 : 맥/리눅스에선 기본 설치
- 시스템 유ти리티
- GUI 프로그램 : PyQt, wxPython, Kivy 등
- 네트워크 애플리케이션 : Twisted, Tornado 등
- 웹 프로그래밍 : Django, Flask, Pyramid, CherryPy, Bottle, Google App Engine, Pylons, web2py 등
- 안드로이드 앱 개발 : Kivy (서비스용은 아닙니다.), PyBee
- 게임 : Pygame, Pandas3D 등
- 데이터베이스
- 수치연산 : Matplotlib, numpy, pandas 등
- C/C++/Java/닷넷 과의 결합
- 임베딩 스크립트 엔진 : ArcGIS, AutoDesks (Maya, MotinoBuilder, Softimage), Blender, MineCraft
- 각종 자동화 : 3D Max 스크립팅, MS 오피스 자동화, 포토샵 자동화 등
- UI 자동화 : 마우스 자동 클릭, 키보드 자동입력 등
- 못하는 게 없 ...
- 참고 : http://en.wikipedia.org/wiki/List_of_Python_software

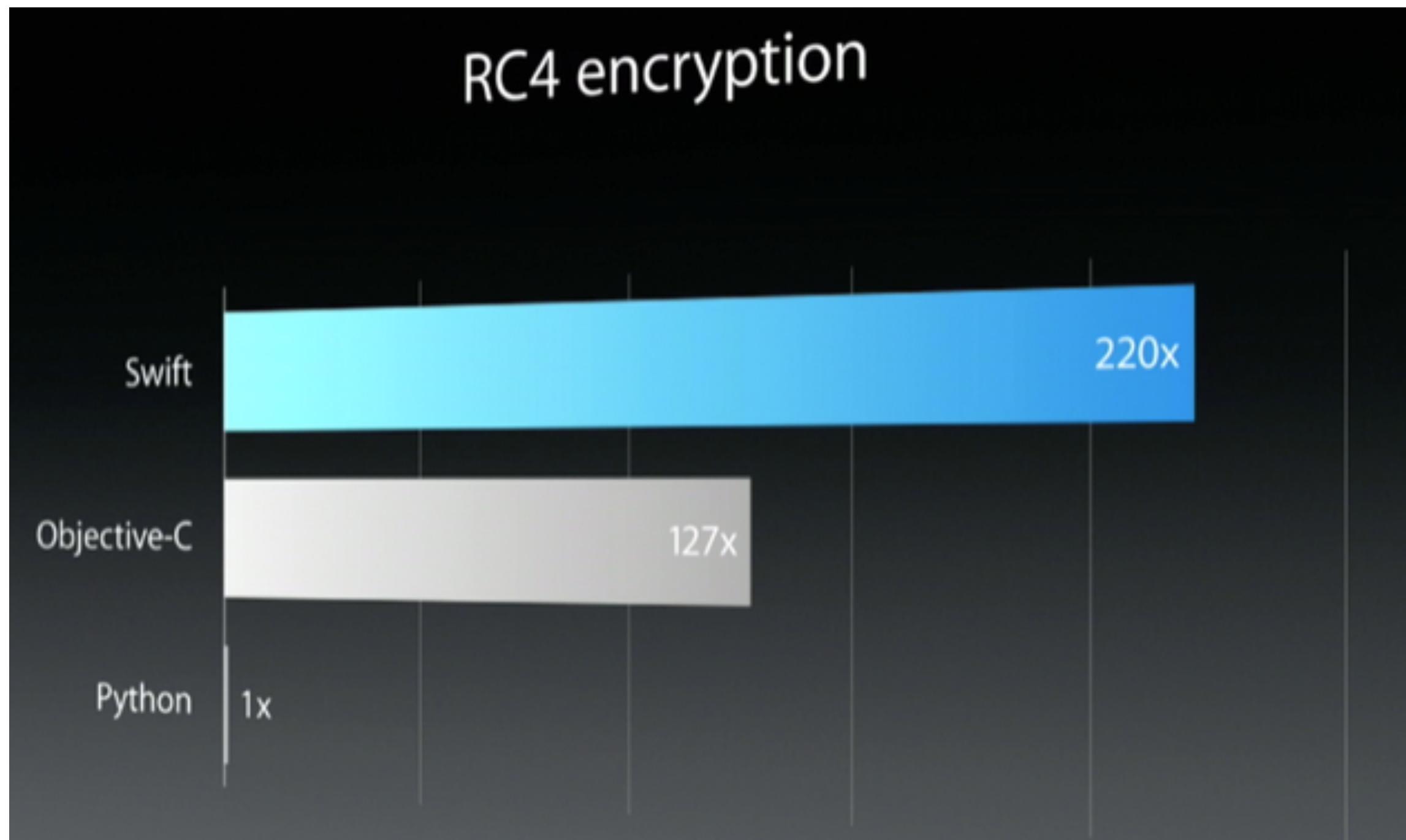
Impossible

- Android/iOS Native 앱 개발은 불가
- 네이티브 앱
 - OS 벤더에서 제공하는 SDK (Software Developer Kit)로 만드는 앱
 - Android : Java by Google
 - iOS : Objective-C, Swift by Apple
- 파이썬의 동적인 특성으로 인해, 지원 불가.
- PyBee라는 프로젝트에서 시도 중. 아직 극초반.



파이썬은 느린가요?

- 파이썬의 강점은 속도에 있지 않다.
- 하지만, 그리 느리지도 않다.
- 쉽고 간결하고 강력하고 GLUE, 빠른 개발속도
- **시간이 가장 큰 비용**이다. (10배 이상의 시간을 아낄 수도)



2014년 6월 WWDC

파이썬의 선 (The Zen of Python)

- 명시가 암시보다 좋다.
- 가독성은 중요하다.
- <https://www.python.org/dev/peps/pep-0020/>
- 번역 : <http://kenial.tistory.com/903>

import this

```
$ python -c "import this"
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

다양한 파이썬 구현체

- CPython : C 언어로 만들어진 파이썬
- Jython : Java 로 만들어진 파이썬
- IronPython : Microsoft .NET으로 만들어진 파이썬
- PyPy : 파이썬으로 만들어진 파이썬
- CLPython, Cython, Parrot, Psyco, Stackless Python, Unladen Swallow, etc.



파이썬은 3.4 이상을 쓰세요.

- 파이썬은 현재 버전 2.X 와 버전 3.X 가 공존합니다.
 - 2.7 은 3.X 를 위한 이사 대비 버전
 - 2.7 은 더 이상의 기능 업데이트도 없으며, 2020년까지 시한부 생명 (출처)

파이썬 스타일 가이드

- PEP 0008 - Style Guide for Python Code
 - PEP : Python Enhancements Proposals
 - 일관성있는 코딩 스타일을 위한 코딩 스타일 제안
 - 주요 스타일
 - 들여쓰기는 공백 4칸 (구글은 공백 2칸)
 - 파이썬 코딩 컨벤션 by spoqa

말도 안되는 문법

```
#include <stdio.h>

void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}

int main() {
    int x = 1;
    int y = 2;
    swap(&x, &y);
    printf("x = %d, y=%d\n", x, y);
    return 0;
}
```

```
x = 1
y = 2
y, x = x, y
print("x = {}, y = {}".format(x, y))
```

파이썬 tuple 문법



Anaconda Python 설치

다양한 파이썬 배포판

- 공식 배포판 : <https://www.python.org/downloads/>
- **Anaconda Python** : <https://www.continuum.io/downloads>
 - 특히나 윈도우에서 사용하기 편하도록, 필수 라이브러리들이 팩키징되어있습니다.
 - 윈도우에서는 이 배포판을 설치하는 것이 **정신건강**에 좋아요.
 - 왜냐하면, 윈도우에는 컴파일러가 기본 설치가 안 되어있기 때문이죠.
 - 파이썬 라이브러리들은 C로 작성된 라이브러리들이 많아요. 특히나 과학기술/데이터분석 팩키지들
- Enthought Canopy : <https://www.enthought.com/products/canopy/>

설치

- 조금 전에 다운받은 Anaconda Python 을 설치해주세요.
- 주의) 윈도우 : 필히 "환경변수 PATH 에 추가" 옵션 체크

CLI 띄우기

- CLI (Command Line Interface) 는 운영체제마다 서로 다른 CLI 인터페이스를 제공합니다.
 - 비교) GUI (Graphic User Interface)
- 윈도우
 - 명령 프롬프트 : 실행창 띄워서 cmd 엔터
 - 파워쉘
- 맥
 - 터미널
 - Iterm2

CLI를 통해 파이썬 버전을 확인해보세요.

- 윈도우
 - 쉘> python --version
- 맥
 - 쉘> python3 --version

소스코드 편집기

Sublime Text 3

소스코드 편집기

- 워드 프로세스
 - 문서마다 내용과 서식을 직접 입력/지정
 - MS 워드, 한글, etc.
- 소스코드 편집기
 - 문서의 내용만을 입력하고, 서식 (글자의 색상, 스타일)은 자동 지정됨.
 - **Sublime Text 3 (추천)**, Atom, Visual Studio Code, VIM, Emacs, etc.
 - 에디터는 편한 걸 쓰세요

Sublime Text 3 설치

- 파이썬3 으로 만들어진 소스코드 편집기
- <https://www.sublimetext.com/3>
- 고정폭 폰트 D2Coding 설치
 - 가변폭 폰트는 코드보기 불편해요.
 - 다운받아서, 설치파일을 더블클릭해서 설치



Package Manager 설치

- <https://packagecontrol.io/installation>

- Sublime Text 3

- 상단 메뉴 : View > Show Console

- 하단 입력에 우측 코드를 복사&붙여넣기

- 설치가 진행됩니다.

- 설치 중에 경고창이 1개 뜨는 것은 무시하셔도 됩니다.



Package Control

INSTALLATION

Simple

The simplest method of installation is through the Sublime Text console. The console is accessed via the `ctrl+`` shortcut or the `View > Show Console` menu. Once open, paste the appropriate Python code for your version of Sublime Text into the console.

아래 코드를

클립보드에 복사

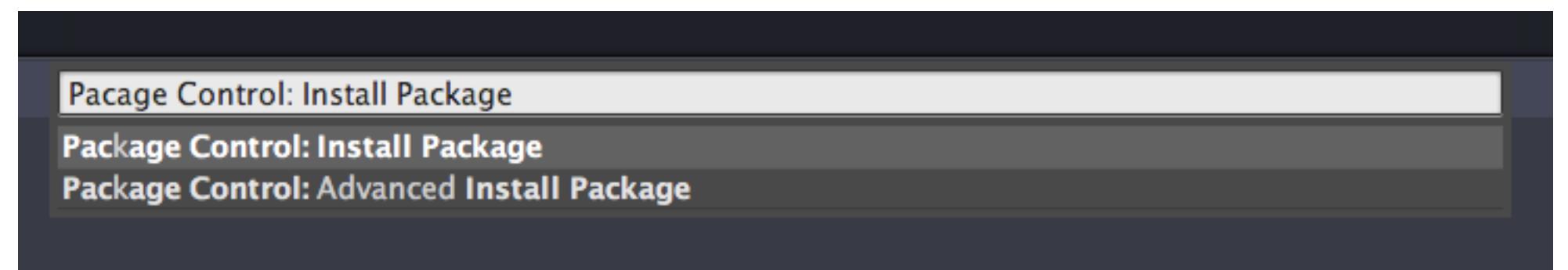
SUBLIME TEXT 3

SUBLIME TEXT 2

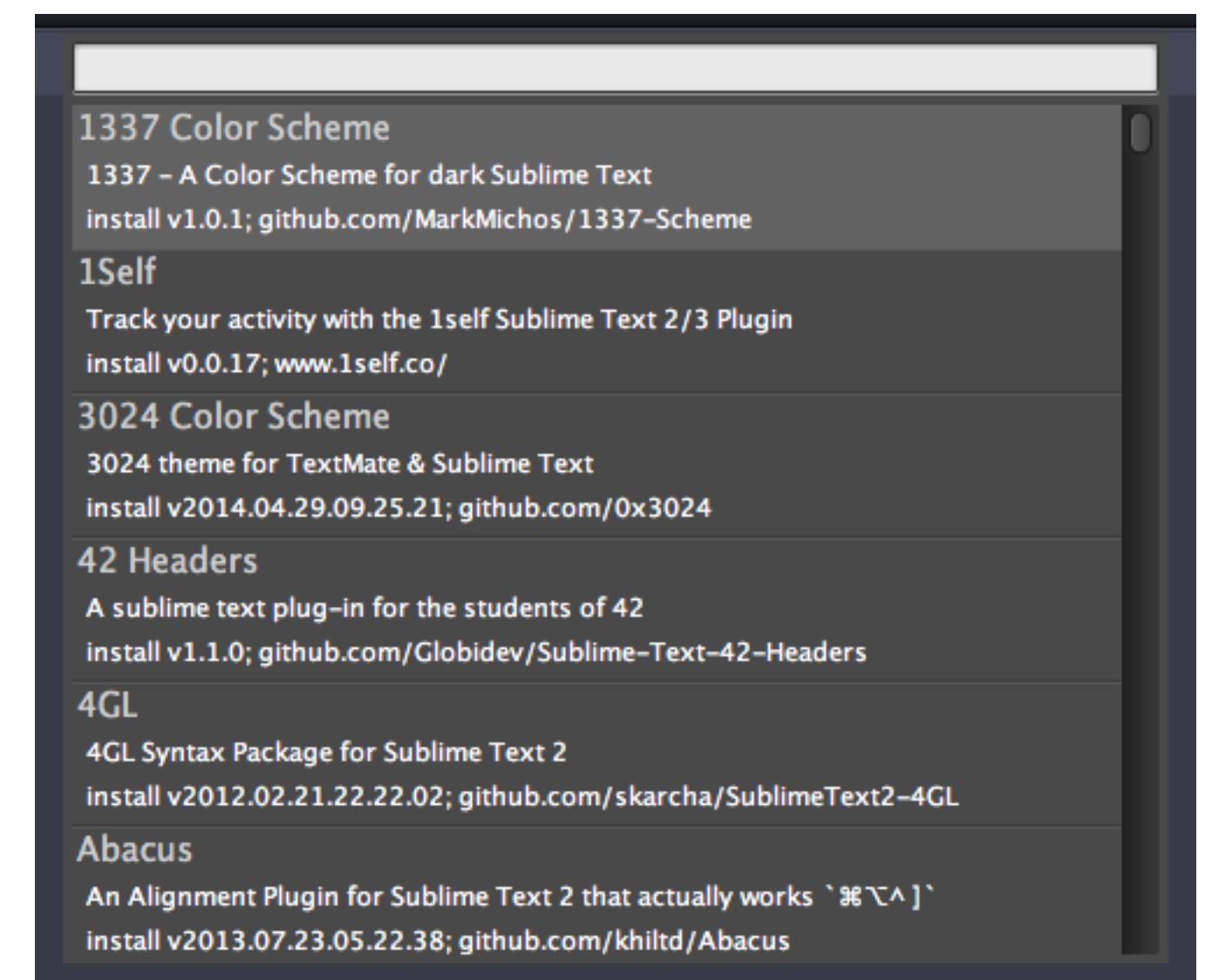
```
import urllib.request,os,hashlib; h =
'261dd1222b4693ce6d4f85f9c827ac06' +
'6d5ab8ebdd020086947172a8a1356bb6'; pf =
'Package
Control.sublime-package'; ipp =
sublime.installed_packages_path();
urllib.request.install_opener(
urllib.request.build_opener(
urllib.request.ProxyHandler())); by =
urllib.request.urlopen('http://packagecontrol.io/' +
pf.replace(' ', '%20')).read(); dh =
hashlib.sha256(by).hexdigest(); print('Error
validating download (got %s instead of %s), please try
manual install' % (dh, h)) if dh != h else
open(os.path.join(ipp, pf), 'wb').write(by)
```

Package Manager 를 통한 플러그인 설치

- 상단 메뉴 Tools > Command Palette 선택



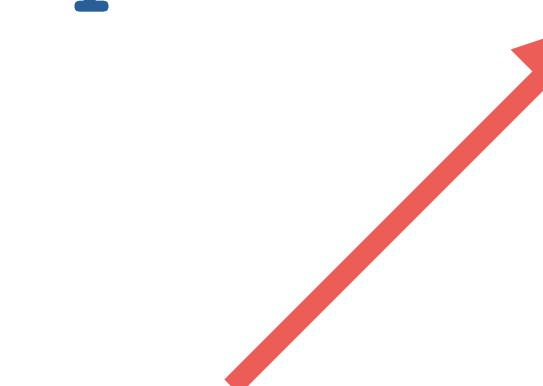
- 팝업창에서 Package Control: Install Package 입력 <Enter>
- 약간의 로딩 시간 후에, 플러그인 선택 창이 뜹니다. 플러그인을 선택하면, 해당 플러그인이 설치됩니다.
- 추천 라이브러리
 - colors sublime 플러그인 설치 : 테마 플러그인
 - SideBarEnhancements 설치 : 사이드바 메뉴 강화



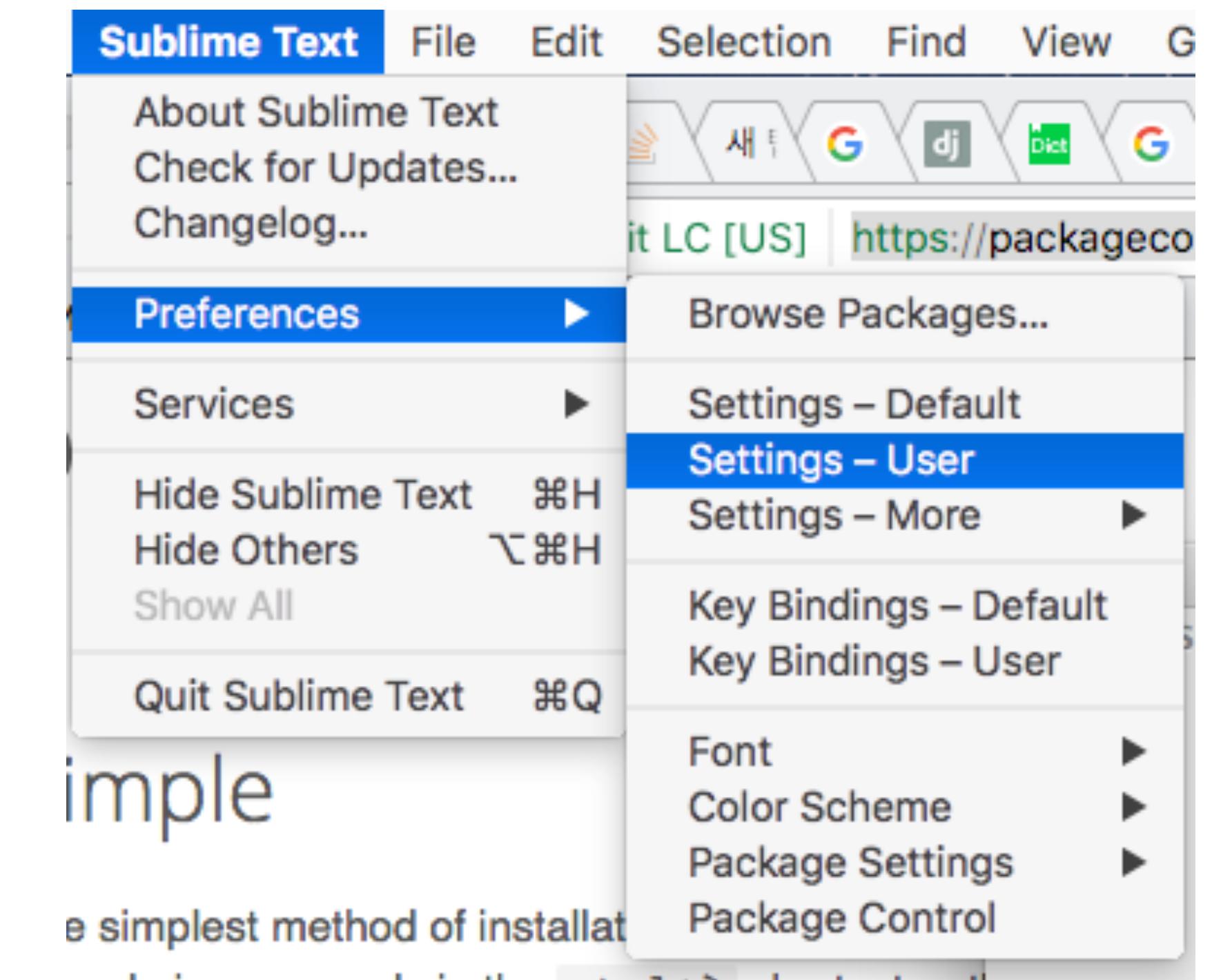
Sublime Text 3 설정

- 상단 메뉴 Preferences > Settings - User 선택하여, 아래 설정을 입력합니다. (json 포맷)
 - <https://github.com/askdjango/snu-web-2016-09#sublime-text-3-추천-설정>

```
{
  "font_face": "D2Coding",
  "highlight_line": true,
  "highlight_modified_tabs": true,
  "tab_size": 4,
  "translate_tabs_to_spaces": true,
  "trim_trailing_whitespace_on_save": true,
  "word_wrap": true
}
```



끝에 콤마(,) 가 없음에 유의해주세요.



Sublime Text Tip

- 파일을 하나씩 열지 마시고, 프로젝트를 디렉토리 통채로 여세요.
- 한 화면에 소스파일 여러 개 보기
 - View -> Layout
- 파일 쉽게 찾기
 - Tool -> Goto Anything ...

소스코드 경로는

- 소스코드를 두는 경로는 바탕화면이나 내문서 경로를 쓰지마세요.
- 윈도우
 - c:\dev
- 맥
 - ~/dev

파이썬에서 코드 실행하는 방법. 3가지

- 1) Interactive Shell에서 한땀 한땀 실행하기
 - 쉘> python
 - >>> print(sum(range(101)))
- 2) 파이썬에 코드 직접 넣기
 - 쉘> python -c "print(sum(range(101)))"
- 3) 소스파일로부터 한 번에 실행하기
 - 쉘> python filename.py



Lift is short, use **Python**.

2016 벤처창업 웹프로그래밍 1

파이썬 Part.

#4

News : Sublime Text 3 Build 3124

- 2016년 9월 22일 (목) 업데이트
- 주요 변경내역
 - Package Control 을 메뉴를 통해 편하게 설치 가능
 - Settings 를 Default 와 Users 를 같이 보여줍니다.
- 출처 : <https://www.sublimetext.com/blog/articles/sublime-text-3-build-3124>

오늘 다룰 주제

- 데이터 타입/변수
- 연산자
- 제어구조
- 인코딩/디코딩, 유니코드



변수/데이터타입

변수 (Variables) (1)

- 프로그램이 실행되면서, 필요한 데이터가 저장된 공간
- 프로그램에서 데이터를 처리하기 위해서는, 필히 변수에 값을 저장해서 처리해야합니다. 이때 효율성을 높이기 위해, 적절한 크기/용도의 변수에 값을 담아서 처리해야합니다.
- 변수에 값을 저장하는 문법
 - >>> name = "nabi"
>>> print(name)
nabi
 - >>> message = "Python is best!"
>>> print(message)
Python is best!
- 하나의 소프트웨어가 동작하면서, 로직에 따라 수많은 새로운 변수가 생겨나고 변경되며 삭제됩니다.

```
>>> name = "nabi"  
>>> print(name)  
nabi  
>>> message = "Python is best!"  
>>> print(message)  
Python is best!
```

데이터 타입 (Data Types) (1)

- 변수는 하나의 데이터를 담아두는 공간
- 그릇 개념으로 이해할 수 있습니다.
- **효율성**을 위해, 그릇도 목적 (크기/용도)에 따라 다양한 그릇이 필요합니다.
- 왜냐하면, 하드웨어 자원은 **유한**하기 때문입니다.

모두 물을 담아두고 쓰는 용기이지만 ~



데이터 타입 (Data Types) (2)

- 숫자형 (Numeric Type) : 숫자를 표현하는 데이터형
 - int 타입 : 정수형 데이터 타입
 - float 타입 : 실수형 데이터 타입
- 사칙연산 (+, -, *, /), 몫 (//), 나머지 (%), 지수 승 (**) 을 지원
- >>> 3
3
>>> 10 // 3
3
>>> 10 % 3
1
>>> 10 ** 3
1000
>>> myvar = 10 // 3
>>> print(myvar)
3
- 아주 큰 수도 파이썬을 통해 계산할 수 있습니다.

```
% python
Python 3.5.1 (default, Feb 24 2016, 12:24:01)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on darw
Type "help", "copyright", "credits" or "license" for more information
>>> 2 ** 1000
1995063116880758384883742162683585083823496831886192454852008949
3239578557300467937945267652465512660598955205500869181933115425
8065225096643874441046759871626985453222868538161694315775629640
2556564953060314268023496940033593431665145929777327966577560617
615734588139257095379769119277800826957735674441230620187578363
6147305207431992259611796250130992860241708340807605932320161268
1040584584156607204496286701651506192063100418642227590867090057
9827074044731623767608466130337787060398034131971334936546227005
8368837838027643282775172273657572744784112294389733810861607423
9052104758088261582396198577012240704433058307586903931960460340
7407752739863482984983407569379556466386218745694992790165721037
2987812389473928699540834346158807043959118985815145779177143619
2561490458290499246102863008153558330813010198767585623434353895
7553764067268986362410374914109667185570507590981002467898801782
7636741877711055384225739499110186468219696581651485130494222369
6855163986053460229787155751794738524636944692308789426594821700
4173363116569360311222499379699992267817323580231118626445752991
5623486556440536962622018963571028812361567512543338303270029097
3018530484711381831540732405331903846208403642176370391155063978
4482985184615097048880272747215746881315947504097321150804981904
```

데이터 타입 (Data Types) (3) - 논리형

- 논리형 (Boolean Type) : 참/거짓을 표현
 - bool 타입
 - 참은 True 로 표현하고, 거짓은 False 로 표현
- >>> 10 > 3
True
>>> 10 < 3
False
- 비교 연산자 (<, <=, >, >=, ==, !=, is, is not) 에서는 논리형 값으로 계산된다.
- 논리 연산자
 - x or y : >>> (10 < 3) or True
 - x and y : >>> (10 > 3) and False
 - not x : >>> not (10 < 3)

데이터 타입 (Data Types) (4) - 문자열형

- 문자열형 (String Type)

```
name1 = "Chinseok" | name1 = 'Chinseok'
```

- 문자열을 홀따옴표 (') 로 감싸거나, 쌍따옴표 (") 로 감싸면, 파이썬이 문자열로 인식합니다.
- 홀(쌍)따옴표 1개로 감싼 문자열 안에 홀(쌍)따옴표를 문자열로서 처리하고 싶은 경우, 해당 홀(쌍)따옴표를 ESCAPE 처리해주면 됩니다. `like I\'m the queen'`
- 파이썬은 여러 줄 문자열 문법을 지원합니다. 홀따옴표 3개 (""") 혹은 쌍따옴표 3개 ("""") 로 감싸는 문법입니다.
- 참고 : [코드#1](#), [코드#2](#)

```
lyrics1 = "The snow glows white on the mountain tonight"
"Now a footprint to be seen"
"A kingdom of isolation"
"and it looks like I'm the queen"
```

데이터 타입 (Data Types) (5) - 문자열 형식 지정자

```
>>> '{0}, {1}, {2}'.format('a', 'b', 'c')
'a, b, c'
>>> '{}, {}, {}'.format('a', 'b', 'c')  # 3.1+ only
'a, b, c'
>>> '{2}, {1}, {0}'.format('a', 'b', 'c')
'c, b, a'
>>> '{2}, {1}, {0}'.format(*'abc')        # unpacking argument sequence
'c, b, a'
>>> '{0}{1}{0}'.format('abra', 'cad')    # arguments' indices can be repeated
'abracadabra'
```

```
>>> 'Coordinates: {latitude}, {longitude}'.format(latitude='37.24N', longitude='-115.81W')
'Coordinates: 37.24N, -115.81W'
>>> coord = {'latitude': '37.24N', 'longitude': '-115.81W'}
>>> 'Coordinates: {latitude}, {longitude}'.format(**coord)
'Coordinates: 37.24N, -115.81W'
```

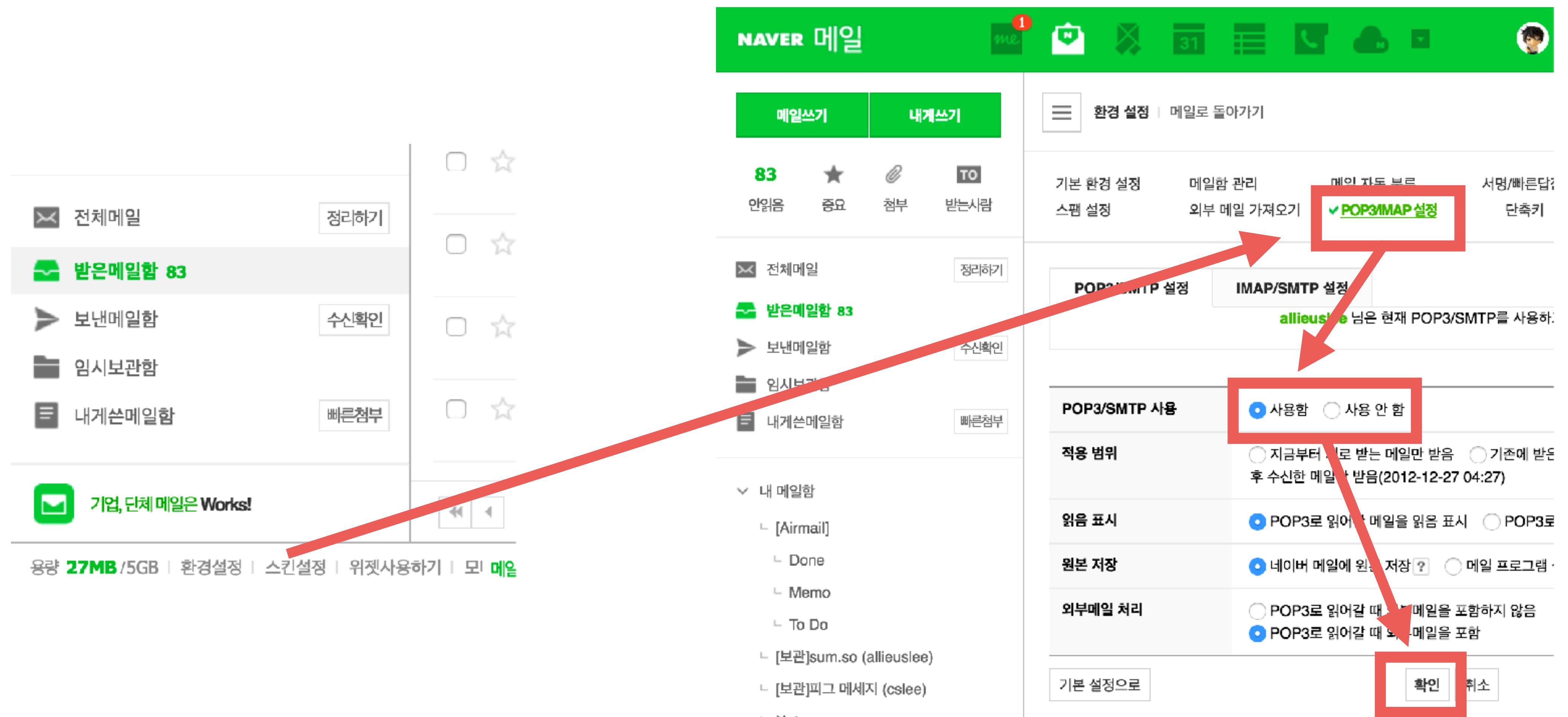
- 더 많은 내용 참고 : <https://docs.python.org/3/library/string.html>

나에게 보내는 편지

네이버 메일, smtp 설정

- SMTP : Simple Mail Transfer Protocol
- SMTP 서버명 : smtp.naver.com
- SMTP 포트 : 465, 보안연결 (SSL) 필요
- 아이디 : 네이버 아이디
- 비밀번호 : 네이버 비밀번호
- 공식문서 : <https://docs.python.org/3/library/smtplib.html>

네이버 SMTP 계정 설정



암호 입력받기

- `input()` 를 통해서 입력을 받을 때에는 입력 내용이 노출이 됩니다.
- 이때는 `getpass.getpass()` 를 통해 암호를 입력받아보세요.

```
from getpass import getpass  
  
password = getpass('Enter password : ')  
  
print(password)
```

네이버 SMTP 를 통해 나에게 메일 쓰기

```
import smtplib
from getpass import getpass

sender = 'allieuslee@naver.com'
receiver = 'allieuslee@gmail.com'
password = getpass('Enter Password : ')
subject = '나에게 쓰는 편지'
message = '''blah blah blah
blah blah blah ~~~
blah blah blah ~~~
blah blah blah ~~~'''

server = smtplib.SMTP_SSL('smtp.naver.com', 465)
server.login(sender, password)

body = '''To: {}
From: {}
Subject: {}

{}'''.format(receiver, sender, subject, message)

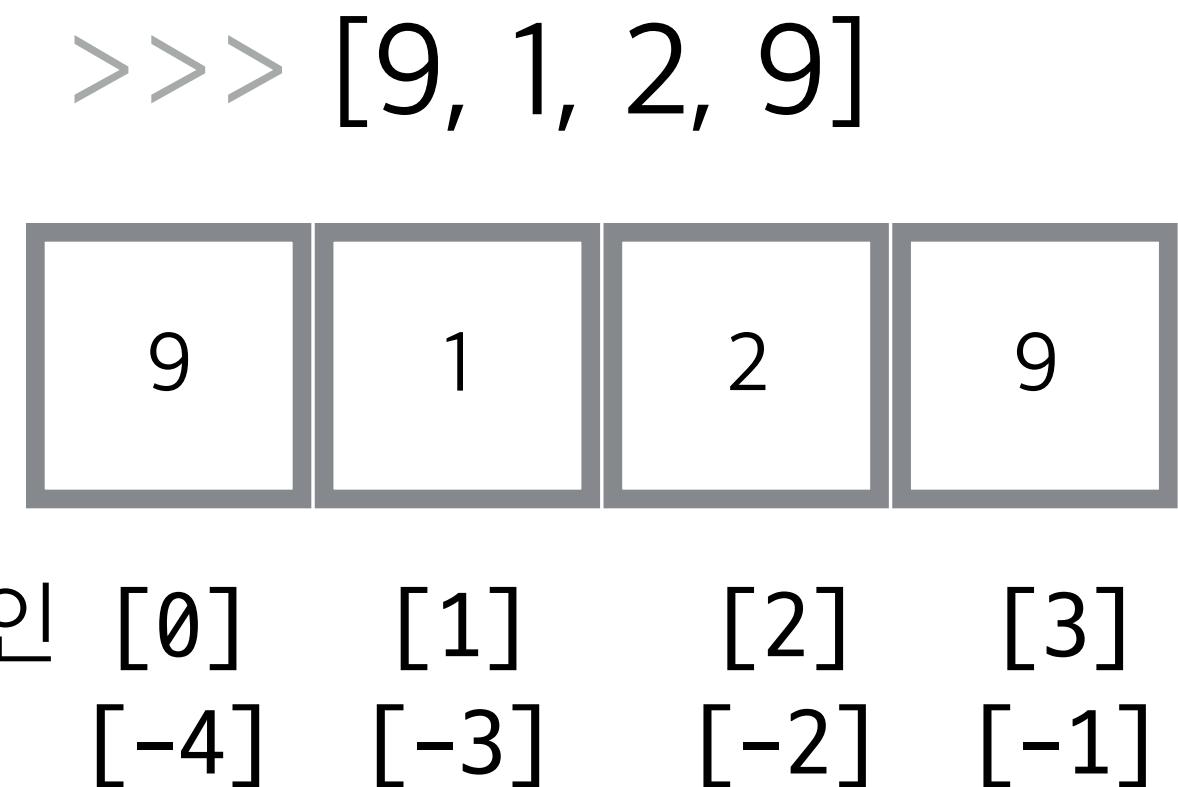
server.sendmail(sender, [receiver], body.encode('utf8'))
server.quit()

print('sended!')
```

기본 자료구조

리스트 형 (List Type) (1)

- 공식문서 : <https://docs.python.org/3/tutorial/datastructures.html#more-on-lists>
 - 여러 값을 순차적으로 저장하는 용도
 - Syntax : 각 값을 쉼표 (,) 로 구분하며, 대괄호 ([]) 로 감싸줍니다.
- ```
>>> numbers = [1, 3, 5, 7, 9, 11]
>>> for n in numbers:
 print(n)
```
- 저장된 데이터는 색인 (index) 을 지원한다.
    - 0 부터 시작하여 1씩 증가
    - 혹은, 끝에서부터 역으로 -1 부터 1씩 감소
  - 다른 언어의 배열 (Array) 과 유사하지만, 다르다.
  - 한 List 에 서로 다른 데이터타입의 값을 넣을 수 있다. 하지만, 가급적 같은 타입으로 맞춰주는 것이 보다 알기 쉬운 코드를 작성하는 방법.



# 리스트 형 (List Type) (2)

- 기본 예시

```
>>> numbers = [1, 3, 5, 7] # 리스트 선언

>>> print(numbers[0]) # 색인 [0]의 값 출력
>>> print(numbers[3]) # 색인 [3]의 값 출력
>>> print(numbers[-3]) # 색인 [-3]의 값 출력
>>> print(numbers[-1]) # 색인 [-1]의 값 출력
>>> print(len(numbers)) # 리스트의 길이 출력

>>> for i in numbers: # for 루프를 통해 List 내 모든 값을 순회
 print(i)
```

- 범위 밖의 색인을 참조할 경우, IndexError 가 발생

```
>>> numbers = [1, 3, 5, 7]
>>> print(numbers[10])
Traceback (most recent call last):
 File "t.py", line 4, in <module>
 print(numbers[10])
IndexError: list index out of range
```

# 리스트 형 (List Type) (3)

- 데이터 변경

```
>>> numbers = [1, 3, 5, 7, 5]
>>> numbers[0] = 10 # 지정 색인의 값을 변경
>>> numbers.append(9) # 끝에 값을 추가
>>> numbers.pop(3) # 특정 색인값을 가져옴과 동시에 제거
>>> numbers.remove(5) # 특정 값을 1회 제거
>>> numbers.insert(1, 11) # 특정 색인 위치에 값을 추가
```

- 값을 잘라내기 (Slice) Syntax

```
>>> numbers = [1, 3, 5, 7, 5]
>>> print(numbers[1:]) # 유형1) 리스트[시작인덱스:]
[3, 5, 7, 5]
>>> print(numbers[1:3]) # 유형2) 리스트[시작인덱스:끝인덱스] - 시작인덱스 이상, 끝인덱스 미만
[3, 5]
>>> print(numbers[1:3:2]) # 유형3) 리스트[시작인덱스:끝인덱스:인덱스증가량]
[3]
```

- 리스트 합치기

```
>>> numbers1 = [1, 3, 5, 7]
>>> numbers2 = [2, 4, 6, 8]
>>> print(numbers1 + numbers2)
[1, 3, 5, 7, 2, 4, 6, 8]
```

- List Comprehension

```
>>> numbers1 = [1, 3, 5, 7]
>>> numbers2 = [2, 4, 6, 8]
>>> print([i + j for (i, j) in zip(numbers1, numbers2)])
```

# 튜플 형 (Tuple Type)

- 공식 문서 : <https://docs.python.org/3/tutorial/datastructures.html#tuples-and-sequences>
- 대괄호가 아닌 소괄호로 선언
- List Type 과 유사하지만, 변경 불가능 (Immutable)

```
>>> numbers = (1, 3, 5, 7, 5)
>>> numbers[0] = 10 # 지정 색인의 값을 변경
Traceback (most recent call last):
 File "t.py", line 2, in <module>
 numbers[0] = 10 # 지정 색인의 값을 변경
TypeError: 'tuple' object does not support item assignment
```

```
>>> numbers.append(9) # 끝에 값을 추가
Traceback (most recent call last):
 File "t.py", line 3, in <module>
 numbers.append(9) # 끝에 값을 추가
AttributeError: 'tuple' object has no attribute 'append'
```

# 튜플 형 (Tuple Type)

- 소괄호는 때에 따라, 우선순위 연산자 혹은 튜플로 쓰인다.

```
>>> number1 = (1 + 3) # 우선순위
>>> number2 = (1 + 3,) # 튜플
>>> number3 = (3) # 우선순위
>>> number4 = (3,) # 튜플
```

- Packing / Unpacking

```
>>> numbers = (1, 2, 3) # packing
>>> v1, v2, v3 = numbers # unpacking
```

# 집합 형 (Set Type)

- 공식 문서 : <https://docs.python.org/3/tutorial/datastructures.html#sets>
- 중복을 허용하지 않는 데이터의 집합
- List/Tuple 과 다르게 순서가 없다.
- in 연산자를 통해, 지정값의 유무를 확인
- 예시

```
>>> set_numbers = { 1, 3, 4, 5, 1, 4, 3, 1 }
>>> print(set_numbers)
>>> print(5 in set_numbers)
```

# 집합 형 (Set Type)

- 수학의 합집합, 교집합, 차집합, 여집합을 지원

```
>>> set_numbers1 = { 1, 3, 4, 5, 1, 4, 3, 1 }
>>> set_numbers2 = { 11, 13, 14, 15, 11, 14, 13, 11 }
```

```
>>> print(set_numbers1 - set_numbers2) # 차집합
>>> print(set_numbers1 | set_numbers2) # 합집합
>>> print(set_numbers1 & set_numbers2) # 교집합
>>> print(set_numbers1 ^ set_numbers2) # 여집합
```

# 사전 형 (Dictionary Type)

- 공식 문서 : <https://docs.python.org/3/tutorial/datastructures.html#dictionaries>
- 전체 항목이 정렬되지 않은 키 (Key) 와 값 (Value) 의 쌍으로 구성된 집합

# 사전 형 (Dictionary Type)

```
>>> dict_values = { 'blue': 10, 'yellow': 3, 'red': 7 }

>>> print(dict_values['blue']) # 지정 Key 의 Value 를 조회
10
>>> print(dict_values['magenta']) # 등록되지 않은 Key 에 접근할 경우, KeyError 발생
Traceback (most recent call last):
 File "t.py", line 5, in <module>
 print(dict_values['magenta']) # 등록되지 않은 Key 에 접근할 경우, KeyError 발생
KeyError: 'magenta'

>>> dict_values['blue'] = 20 # 지정 Key 의 값을 변경
>>> dict_values['black'] = 30 # 새로운 Key:Value 를 등록

>>> del dict_values['black'] # 지정 Key:Value 를 제거
>>> del dict_values['magenta'] # 등록되지 않은 Key 에 접근하여, KeyError 발생
Traceback (most recent call last):
 File "t.py", line 8, in <module>
 del dict_values['magenta'] # 등록되지 않은 Key 에 접근하여, KeyError 발생
KeyError: 'magenta'
```

# 사전 형 (Dictionary Type)

- in 연산자로 지정 Key 의 등록여부를 확인

```
>>> dict_values = { 'blue': 10, 'yellow': 3, 'red': 7 }
```

```
>>> print('magenta' in dict_values)
```

```
False
```

```
>>> print('blue' in dict_values)
```

```
True
```

- for 루프 순회

```
>>> dict_values = { 'blue': 10, 'yellow': 3, 'red': 7 }
```

```
>>> for key in dict_values:
```

```
 print(key)
```

```
red
```

```
yellow
```

```
blue
```

```
>>> for (key, value) in dict_values.items():
```

```
 print(key, value)
```

```
red 7
```

```
yellow 3
```

```
blue 10
```

블록문, 들여쓰기, 주석

# 블록문과 들여쓰기 (1)

- 블록문 (Block Statement)
  - 연속된 코드의 묶음
  - 코드는 다수의 블록문으로 구성되며, 하나의 블록문 안에 다수의 블록문을 가질 수 있습니다.
  - 다른 언어에서는 중괄호 ( {} ) 로서 블록을 구분하지만, 파이썬에서는 **들여쓰기** (Indent) 를 통해 블록을 구분합니다.
  - 들여쓰기는 Tabs 또는 Spaces 로 입력 ([Github 코드 저장소에서의 Tabs/Spaces 사용 통계](#))
    - 파이썬 언어 특징 중에서 가장 **호불호**가 갈리는 기능
    - 코드의 가독성 증대

# 블록문과 들여쓰기 (2)

- C언어

```
#include<stdio.h>

int main() {
 int max = 10;
 int result = 0;
 for(int i=0; i<=max; i++) {
 result = result + i;
 }
 printf("result = %d\n", result);
 return 0;
}
```

- PHP

```
<?
$max = 10;
$result = 0;
for ($i=0; $i<=$max; $i++) {
 $result = $result + $i;
}
echo "result = $result";
?>
```

들여쓰기가  
강제됩니다.

- JavaScript (node.js)

```
var max = 10;
var result = 0;
for(var i=0; i<=max; i++) {
 result = result + i;
}
console.log("result = %s", result);
```

- Python

```
max = 10;
result = 0;
for i in range(max+1):
 result = result + i
print("result = %d" % result)
```

# 블록문과 들여쓰기 (3)

- 파이썬에서 들여쓰기를 지키지 않는다면 IndentationError 가 발생합니다.

```
max = 10;
result = 0;
for i in range(max+1):
 result = result + i
print("result = %d" % result)
```

```
% python sample04-block-statement.py
File "sample04-block-statement.py", line 4
 result = result + i
 ^
IndentationError: expected an indented block
```

- 하나의 들여쓰기는 Python Style Guide #Indentation에 따라, **공백 4칸**을 권장하고 있습니다.
- Tab 과 Space 는 엄연히 다른 글자입니다. 섞어서 쓰다보면, IndentationError 가 발생할 수 있습니다. 필히 하나로 통일 ~ !!!

탭 입력  
공백4칸 입력

```
max = 10;
result = 0;
for i in range(max+1):
 result = result + i
 print("current result %s" % result)
 print("result = %d" % result)
```

```
% python sample04-block-statement.py
File "sample04-block-statement.py", line 5
 print("current result %s" % result)
 ^
IndentationError: unindent does not match any outer indentation level
```

# 블록문과 들여쓰기 (4)

- 우리는 Sublime Text 3 에 이미 관련 설정을 한 적이 있습니다.

```
{
 "font_face": "D2Coding",
 "highlight_line": true,
 "highlight_modified_tabs": true,
 "tab_size": 4,
 "translate_tabs_to_spaces": true,
 "trim_trailing_white_space_on_save": true,
 "word_wrap": true
}
```

- 위 설정으로 말이암마, Sublime Text 3 에서 Tab 을 입력하면, "공백 4칸" 으로 자동변환되어 입력됩니다.

# 주석 (Comments) (1)

- 소스코드를 설명하기 위한 목적으로 씁니다.
- 주석으로 쓴 부분은 실행되지 않습니다.
- 파이썬에서의 주석 문법은 "1줄 주석" 문법만 지원합니다.
  - "여러 줄 주석" 문법은 "문자열"로서 표기합니다.

# 주석 (Comments) (2) - 언어별 주석 차이

- C언어

```
#include<stdio.h>

/* 여러줄 주석
두번째 줄
세번째 줄
네번째 줄 */

// 한 줄 주석

int main() {
 int max = 10;
 int result = 0;
 for(int i=0; i<=max; i++) {
 result = result + i;
 }
 printf("result = %d\n", result);
 return 0;
}
```

- PHP

```
<?
/* 여러줄 주석
두번째 줄
세번째 줄
네번째 줄 */

// 한 줄 주석

$max = 10;
$result = 0;
for ($i=0; $i<=$max; $i++) {
 $result = $result + $i;
}
echo "result = $result";
?>
```

- JavaScript (node.js)

```
/* 여러줄 주석
두번째 줄
세번째 줄
네번째 줄 */

// 한 줄 주석

var max = 10;
var result = 0;
for(var i=0; i<=max; i++) {
 result = result + i;
}
console.log("result = %s", result);
```

- Python

```
'''여러줄 주석
두번째 줄
세번째 줄
네번째 줄'''

한 줄 주석

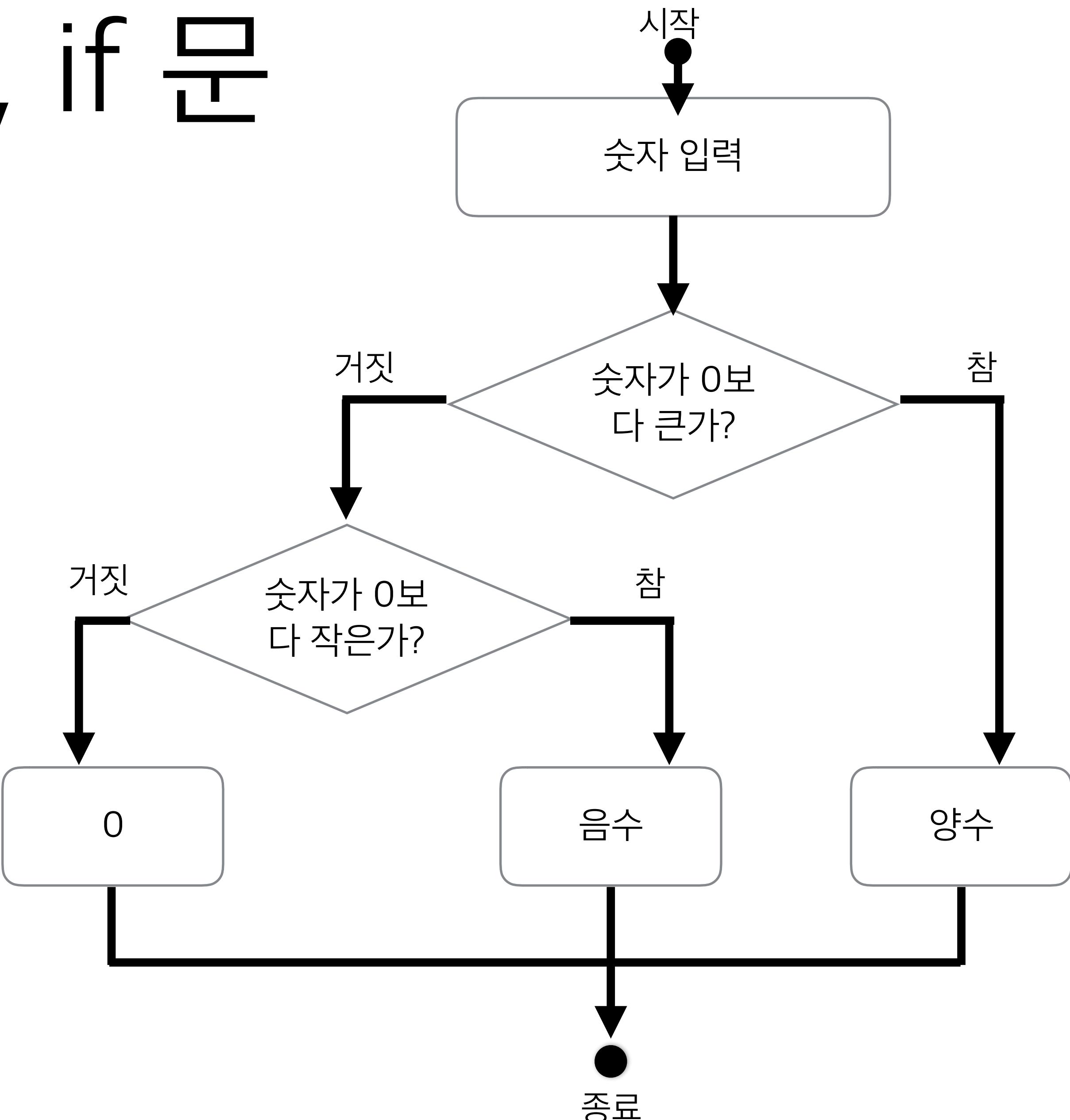
'또 다른 한 줄 주석'

max = 10;
result = 0;
for i in range(max+1):
 result = result + i
print("result = %d" % result)
```

# 제어구조

# 제어구조 (1) - 조건문, if 문

- 조건문 : 특정 조건에 따라, 분기를 하기 위함
- Syntax
  - if 조건:**  
위 조건에 부합될 때 실행할 코드 블록
  - elif 두번째조건:**  
두번째조건에 부합될 때 실행할 코드 블록
  - elif 세번째조건:**  
세번째조건에 부합될 때 실행할 코드 블록
  - elif 네번째조건:**  
네번째조건에 부합될 때 실행할 코드 블록
  - else:**  
위 조건에 모두 맞지 않을 때 실행할 코드 블록
- 한 **if statement**에서
  - if / else 는 1회만 쓸 수 있으나,
  - elif 는 원하는 조건의 수만큼 쓸 수 있습니다.



# 제어구조 (2) - 어떤 차이가 있을까요?

```
number = int(input('Enter number : '))

if number > 0:
 print('%d is positive number.' % number)
elif number < 0:
 print('%d is negative number.' % number)
else:
 print('%d is zero.' % number)
```

```
% python sample06-if-statement.py
Enter number : 10
10 is positive number.
```

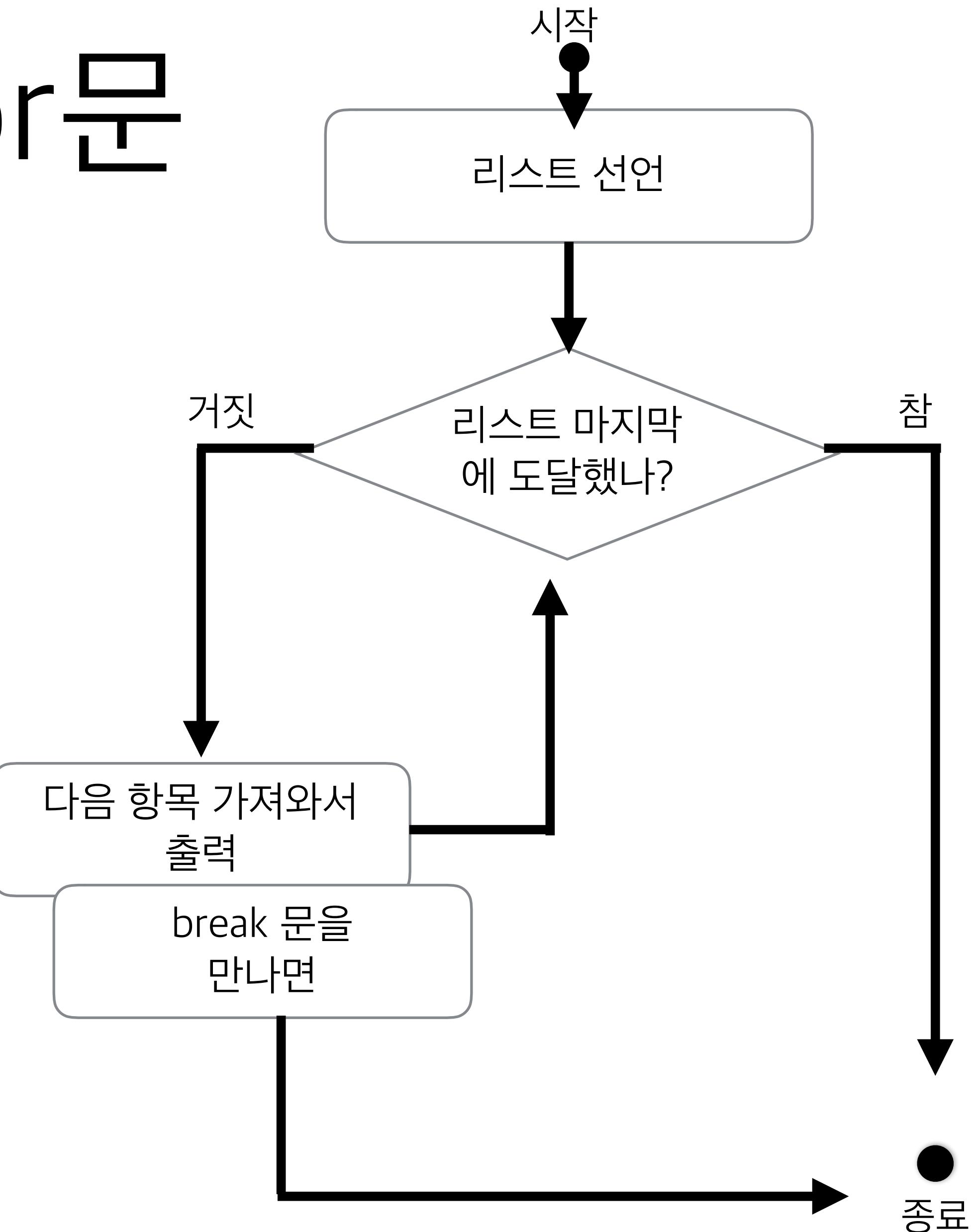
```
number = int(input('Enter number : '))

if number > 0:
 print('%d is positive number.' % number)
if number < 0:
 print('%d is negative number.' % number)
else:
 print('%d is zero.' % number)
```

```
% python sample06-if-statement-02.py
Enter number : 10
10 is positive number.
10 is zero.
```

# 제어구조 (3) - 반복문, for문

- 반복문 : 특정 조건이 만족하는 동안에, 해당 block statement 을 반복 실행
  - for** 변수 **in** 리스트:  
매 항목마다 수행할 코드 블록
  - for 내에서 break 문을 만나면
    - 해당 반복문 종료



# range 내장함수

```
for i in range(1, 10):
 print(i)
```

- range 내장함수
  - range(stop) : 0부터 stop 미만의 범위에서 1 씩 증가시킨 값으로 리스트를 구성
  - range(start, stop[, step]) : start 값 이상, stop 값 미만의 범위에서 step 씩 증가시킨 값으로 리스트를 구성
- 참고 : [파이썬 공식문서](#)

코드 문법을 표기할 때  
[]로 표기하는 것은  
옵션 인자를 뜻합니다.

# 제어구조 (4) - for 문, 예시

```
for i in range(10, 13):
 print(i)
```

```
print('----')
```

```
for i in range(10, 13):
 print(i * i)
```

```
print('----')
```

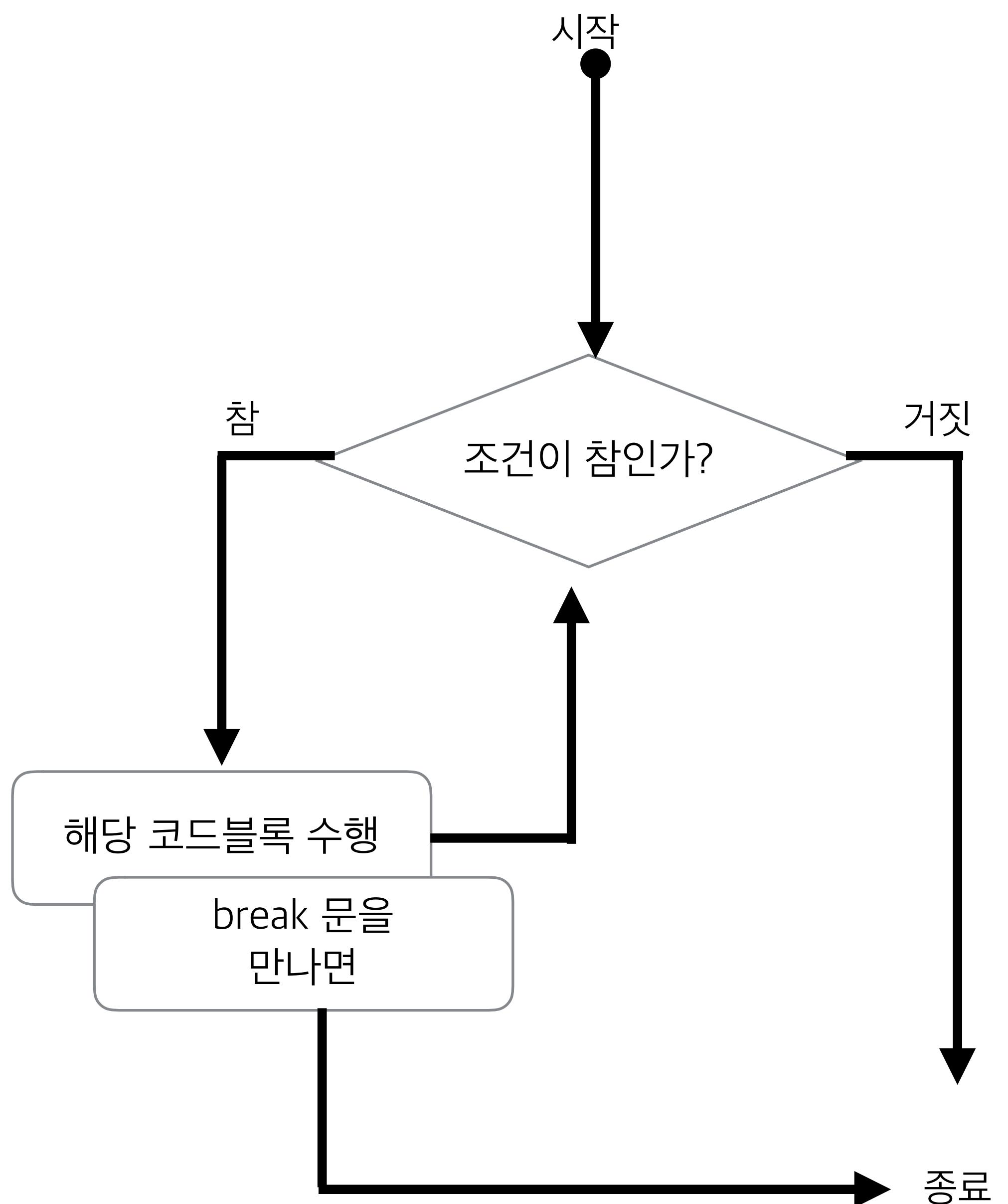
```
for i in range(10, 13):
 print(i ** 2)
```

```
print('----')
```

```
for i in [10, 20, 30]:
 print(i ** 2)
```

# 제어구조 (5) - while

- while 조건:  
    매 항목마다 수행할 코드 블록
- **while** 내에서 break 문을 만나면
  - 해당 반복문 종료



# 제어구조 (6) – while 예시

```
i = 10
while i < 13:
 print(i)
 i += 1
```

```
print('----')
```

```
i = 10
while i < 13:
 print(i * i)
 i += 1
```

```
print('----')
```

```
i = 10
while i < 13:
 print(i ** 2)
 i += 1
```

# 제어구조 (7) - 무한 루프

- <반복문 조건> 이 항상 True 일 때

```
i = 10
while i < 13:
 print(i)
 i -= 1
```

- for 에서는 itertools.count 함수를 통해 가능

```
from itertools import count
```

```
for i in count(1):
 print(i)
```

# 인코딩/디코딩, 유니코드

# 바이트 (byte) 와 비트 (bit)

- 데이터의 크기를 나타내는 단위
- 1 Byte = 8 Bit
  - 총 256개의 데이터를 표현 가능

# ascii 코드

- 영문 알파벳을 사용하는 대표적인 문자 인코딩
- 33개의 출력불가능한 제어문자들과 공백을 비롯한 95개의 출력가능한 문자들
- 참고 : [위키백과] 미국정보교환표준부호

```
>>> import string
>>> string.printable
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\'()*+,./:;<=>?@[\\]^_`{|}~ \t\n\r\x0b\x0c'
>>> string.whitespace
' \t\n\r\x0b\x0c'
```

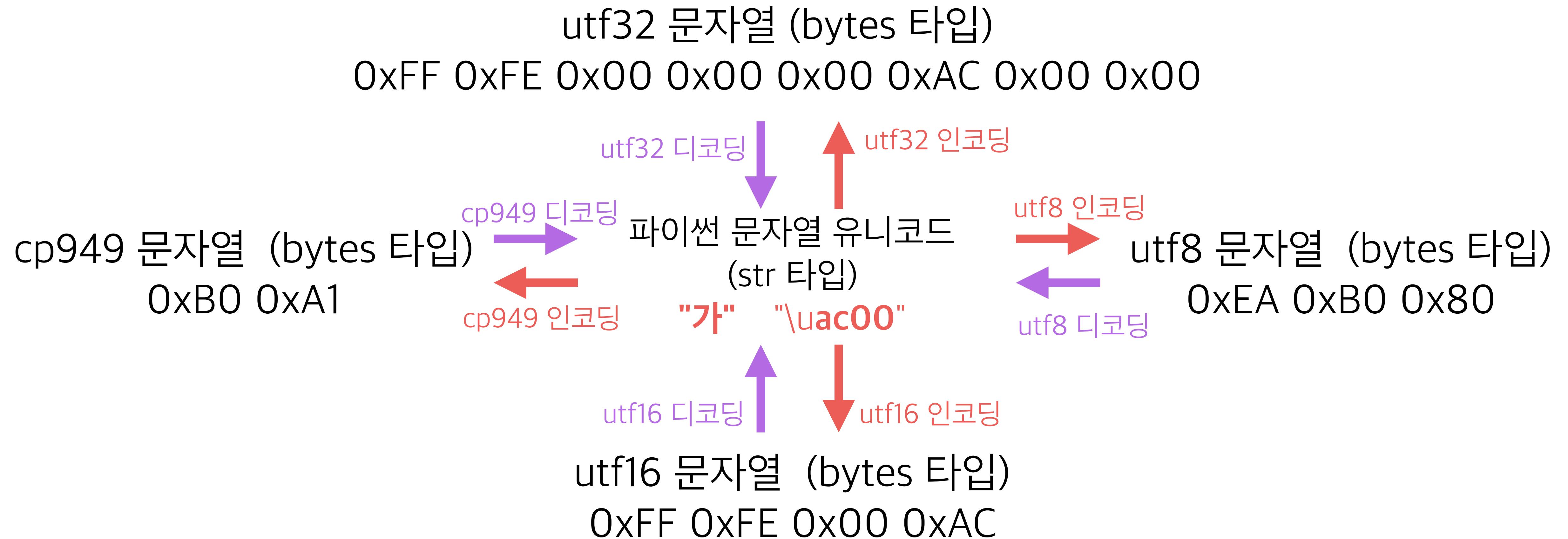
# 문자 인코딩

- 문자나 기호들의 집합을 부호화 (인코딩) 하는 방법
- 인코딩의 2가지 의미
  - 변환하는 방법 : ascii, cp949, utf8, utf16, utf32, etc
  - 변환하는 행위
- 하나의 동영상을 avi, mp4, mkv 등으로 변환 (인코딩) 할 수 있듯이, 하나의 문자열도 다양한 인코딩 (cp949, utf8, utf16, utf32 등) 으로 변환할 수 있습니다.
- 각 인코딩마다 표현가능한 범위가 다릅니다.
- 참고 : [\[위키백과\] 문자 인코딩](#)

# 인코딩 & 디코딩 (1)

- 일반적인 "인코딩 (Encoding)"의 의미 : 어떠한 값을 특정 룰에 맞춰서 변환
  - 디코딩 : 역변환
- 파이썬 문자열 (str)에서의 인코딩 (cp949, utf8, utf16, etc)
  - 하나의 문자를 하나의 숫자로서 표현하는 다양한 Mapping Rule
  - 해당 Mapping Rule에 맞춰서, 변환하는 것

# 인코딩 & 디코딩 (2)



# 인코딩 & 디코딩 (3)

```
unicode_ga = "가"
print(unicode_ga)

utf8_ga = unicode_ga.encode("utf8")
print(utf8_ga)

cp949_ga = utf8_ga.decode("utf8").encode("cp949")
print(cp949_ga)

utf16_ga = cp949_ga.decode("cp949").encode("utf16")
print(utf16_ga)
```

```
$ python d.py
가
b'\xea\xb0\x80'
b'\xb0\xa1'
b'\xff\xfe\x00\xac'
```

# 인코딩 & 디코딩 (4)

- 파이썬 코드 안에서는 모두 유니코드로 처리한다.
- 파이썬 밖과 문자열 데이터를 주고 받을 때에는
  - 줄 때에는 필히 특정 인코딩으로 인코딩 (str -> bytes) 한 후에 전송을 하고.
  - 받을 때에는 그 즉시 특정 인코딩으로 디코딩 (bytes -> str)한다.
- 어떤 경우에 ?
  - 문자열을 파일에 저장할 때
  - 네트워크로 문자열을 전송할 때

# 파일에 저장하고 읽어오기

- 파일을 열 때, 5가지 모드
  - r (read) : 파일 읽기, w (write) : 파일 생성, a (append) : 기존 파일 추가
  - t (text) : 문자열 모드, b (binary) : 바이너리

# 문자열/바이너리 모드

- 문자열 모드 t
  - 지정된 인코딩으로 자동 인코딩/디코딩 수행
- 바이너리 모드 b
  - 직접 인코딩/디코딩을 수행해야함.

```
file = open("filename1.txt", "wt", encoding="utf8")
file.write('가')
file.close()
```

```
file = open("filename2.txt", "wb")
file.write('가'.encode('utf8'))
file.close()
```

# 소스파일을 저장할 때에는 UTF8 ~ !!!

```
cp949_sourcecode.py *
1 # -*- coding: utf-8 -*-
2
```

- 각 파이썬 소스코드마다 <소스코드 인코딩> 을 지정하실 수 있습니다. 미지정시에는 utf8 로 처리됩니다. (파이썬3 기준)
- 다른 소스코드 편집기를 쓰실 때에는, 소스코드 파일 인코딩을 필히 utf8 로 맞춰주세요.
  - 다른 인코딩과 섞어 쓰지 마세요. 그것이 정신건강에 좋아요.
  - 요즘 대다수 소스코드 편집기들의 기본 소스코드 인코딩은 utf8 입니다.
- 소스코드 인코딩을 파이썬에게 잘못 알려주면, 다음과 같이 오류가 뜹니다.
  - 이는 파이썬이 코드실행을 위해, 소스코드를 읽어들여 디코딩을 수행할 때, 예상했던 인코딩과는 다른 문자열을 만났기 때문입니다.

```
$ python cp949_sourcecode.py
 File "cp949_sourcecode.py", line 3
 SyntaxError: (unicode error) 'utf-8' codec can't decode byte 0xb3 in position 0: invalid start byte
```
- 이는 주석이라고 해서 예외가 없습니다. 파일내 모든 문자열에 대해서 해당됩니다.
- 참고 : [UTF8, 유니코드에 대한 오해](#)

# 과제

- 특정 수를 입력받아서, 구구단을 출력하는 프로그램을 작성

```
number = int(input("구구단 : "))
```

- 오른쪽 다이아몬드를 출력하는 코드의 2가지 버전 작성

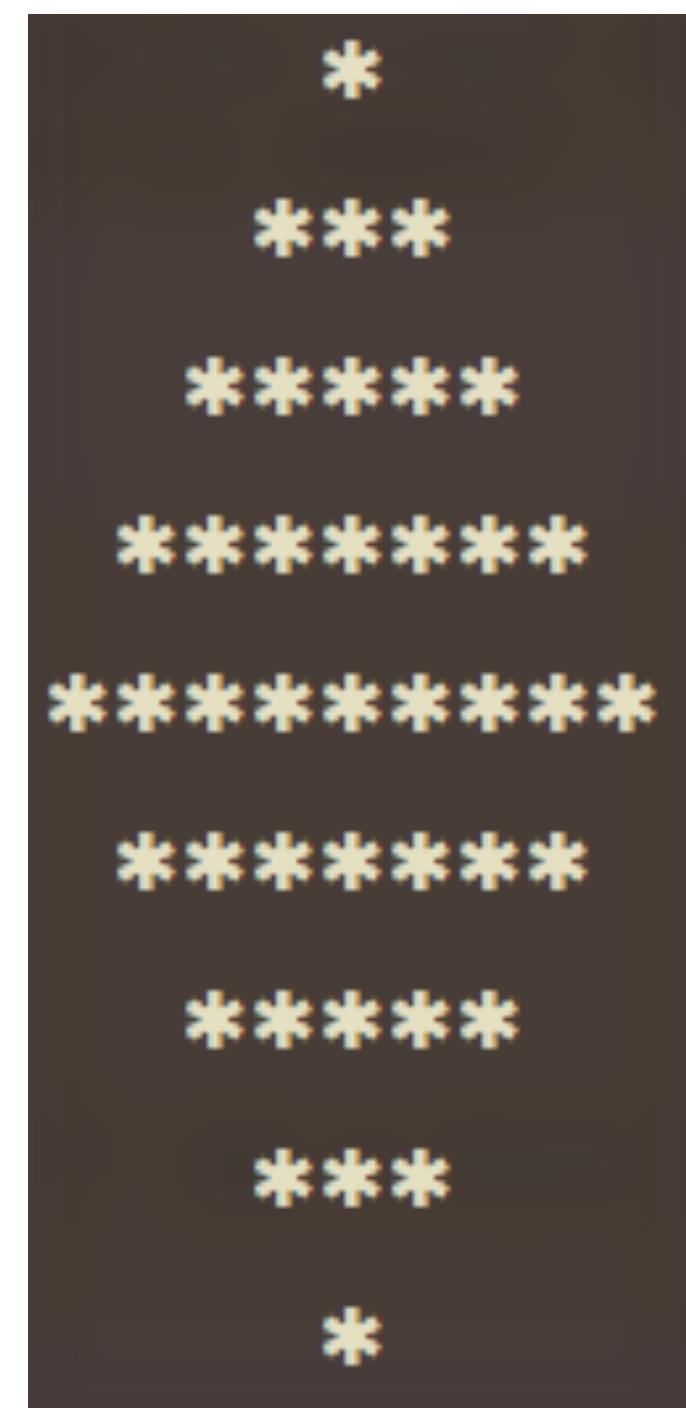
- 형식 지정자를 사용하지 않은 버전, 형식 지정자를 사용한 버전

- 한 문자열에서 지정 단어의 횟수를 카운트하는 프로그램을 작성

```
>>> message = '노랑 빨강 녹색 파랑 노랑 빨강 노랑 빨강'
>>> print(message.split()) # white space로 기준으로 새로운 List를 생성
['노랑', '빨강', '녹색', '파랑', '노랑', '빨강', '노랑', '빨강']
```

- 성적 계산 프로그램 완성하기 - 뼈대 코드

- 과제는 학점에 반영됩니다.



# 과제 이메일 제출 양식

- 제출방법 : ask@festi.kr로 제출
- [2016 벤처창업웹프로그래밍1] 전공, 학번, 이름
  - 첨부로 하실 때에는 압축은 하지 마시고, 개별 .py 소스코드 파일로 제출해주세요.



Lift is short, use **Python**.

# 2016 벤처창업 웹프로그래밍 1

파이썬 Part.

#5

# 오늘 다룰 주제

- Jupyter Notebook
- 함수 (Function)
- 변수의 유효범위 (Scope)
- 인자 (Arguments)
  - 위치 인자 (Positional Arguments), 키워드 인자 (Keyword Arguments)
- Packing, Unpacking



# Jupyter Notebook

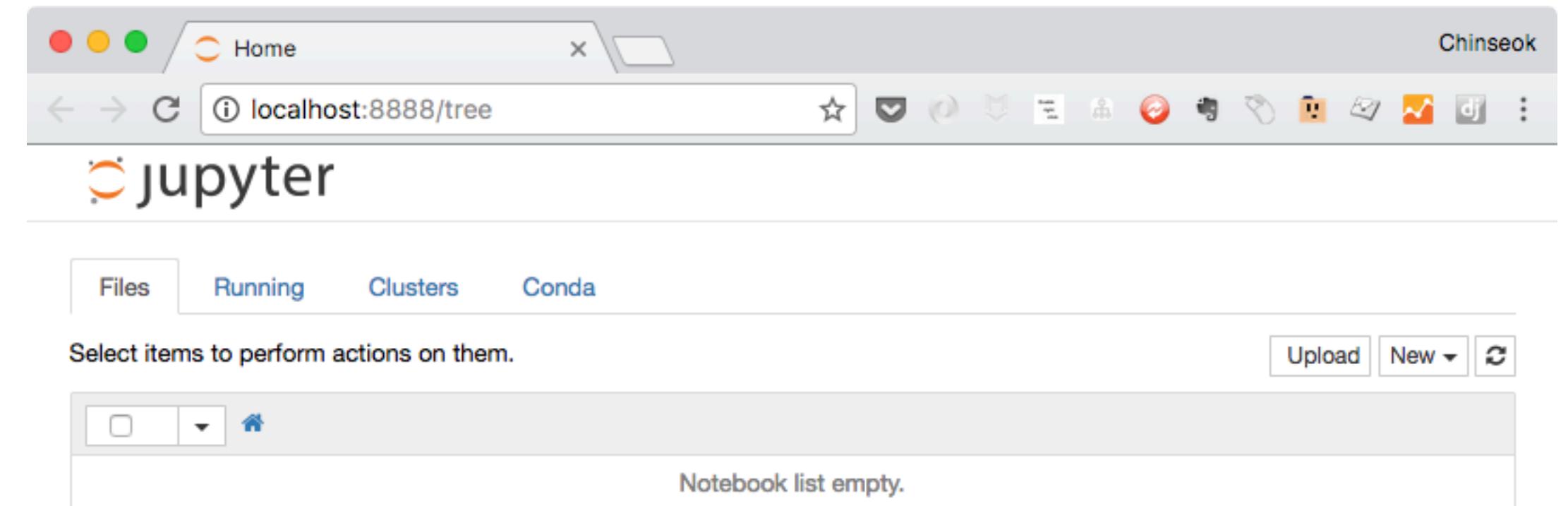
# Jupyter Notebook

- 웹 브라우저를 통해 쓸 수 있는 파이썬 인터랙티브 쉘
- 터미널은 단순 글자만 표현할 수 있는 데 반해, Jupyter Notebook 은 웹브라우저가 표현할 수 있는 모든 것 (차트, 그래픽, 비디오, 오디오 등) 을 표현할 수 있습니다.
- 하나의 Notebook 에 다양한 타입 (Code, Markdown, Raw) 의 다수 Cell 로 구성
- 처음에는 IPython Notebook 이라는 이름으로 파이썬만 지원했으나, Jupyter Notebook 으로 이름을 변경하여 다른 언어 (Ruby, Julia, CSharp 등) 을 지원하기 시작
- 설치) 이미 Anaconda Python 에 포함되어있습니다.
  - 쉘> pip3 install "ipython[notebook]"

# Jupyter Notebook 서버 실행하기

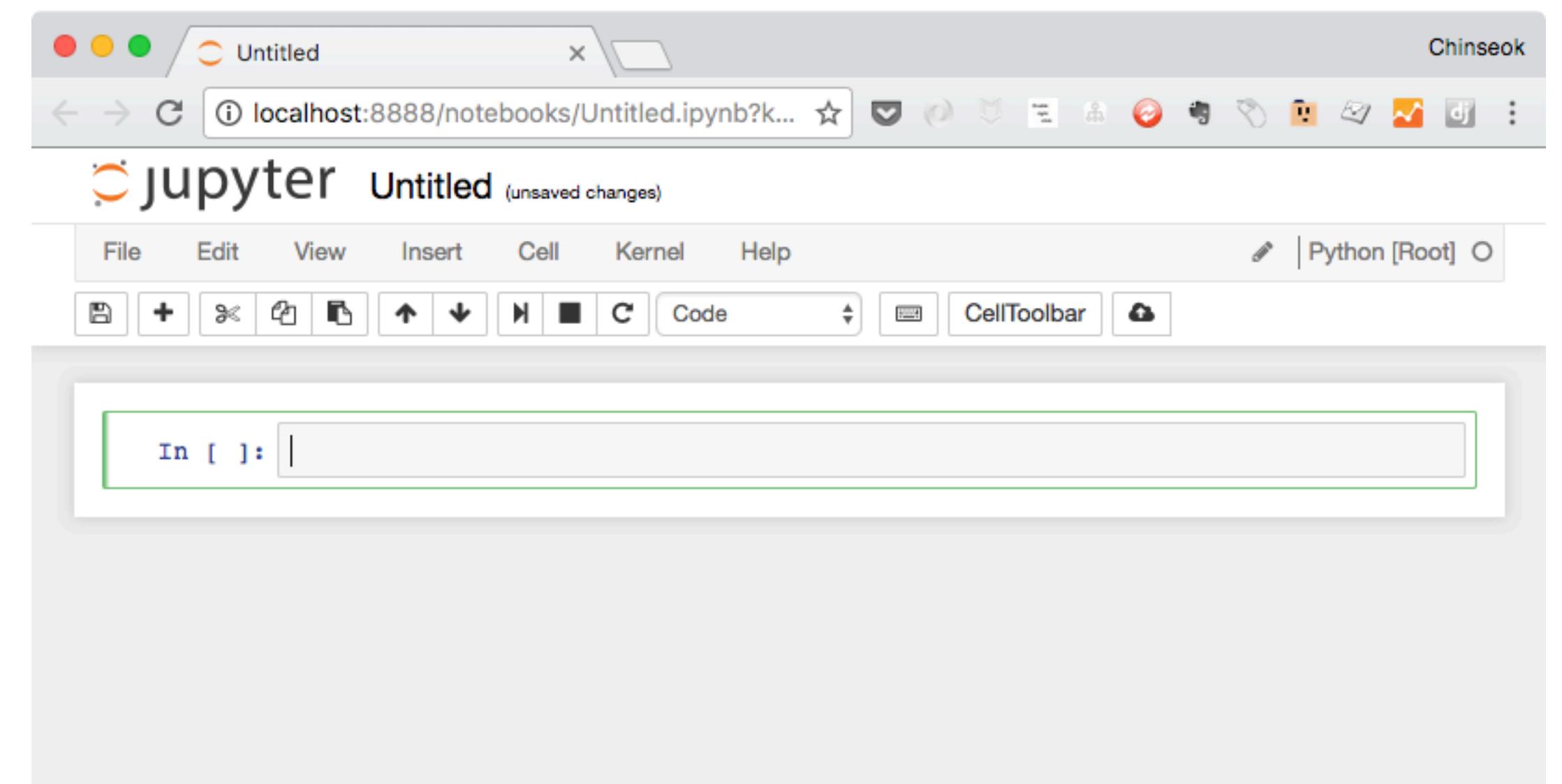
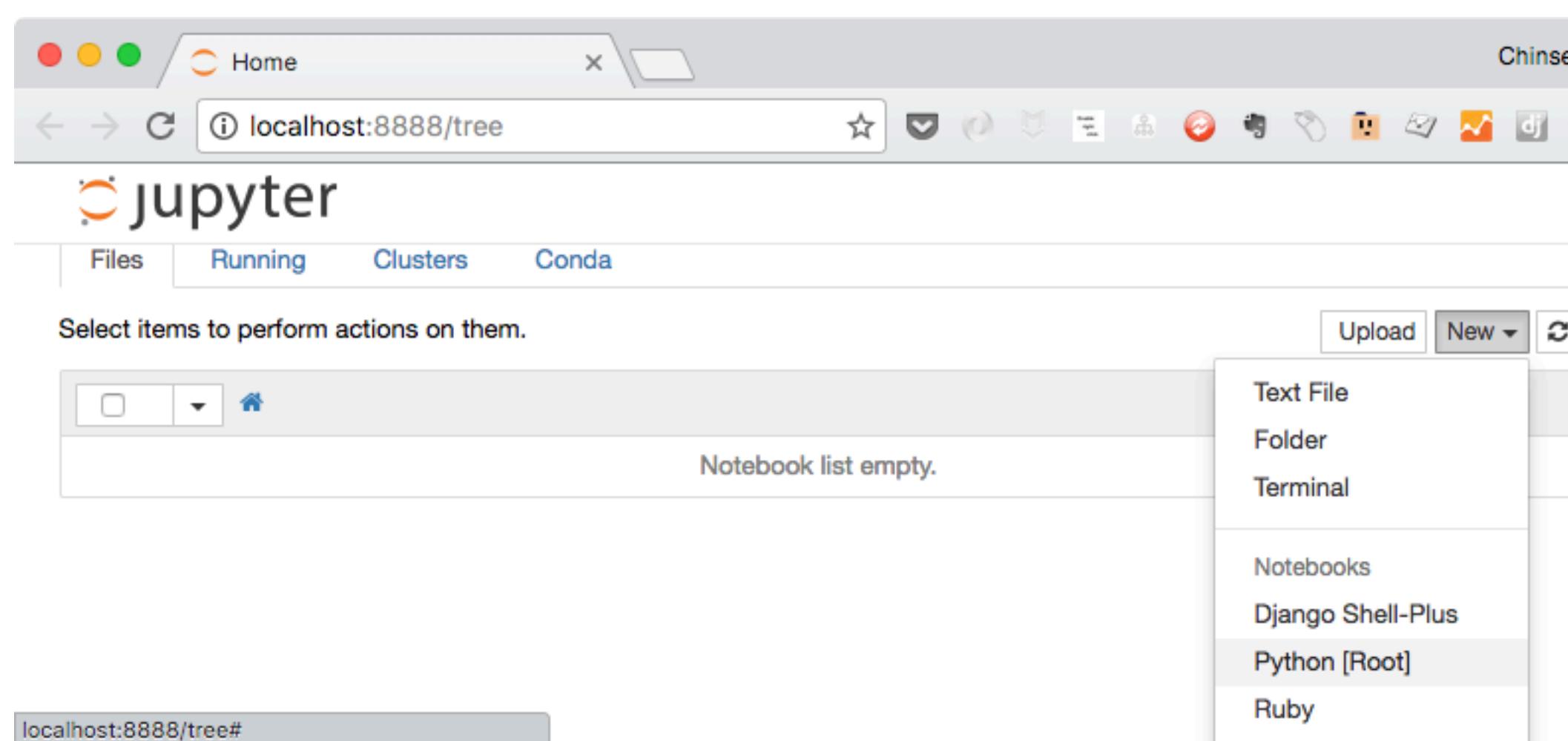
- 로컬에 항상 Jupyter Notebook **서버**가 실행되고 있어야 됩니다.
- 명령프롬프트/터미널을 실행시키시고, 프로젝트 경로로 이동하신 후에, jupyter notebook 명령을 입력하시어, jupyter 서버를 실행시켜 주세요.
- 서버정지는 Ctrl-C

```
$ jupyter notebook
[W 11:18:25.242 NotebookApp] Unrecognized JSON config file version, assuming version 1
[I 11:18:26.098 NotebookApp] [nb_conda_kernels] enabled, 3 kernels found
[I 11:18:26.541 NotebookApp] [nb_conda] enabled
[I 11:18:26.658 NotebookApp] [nb_anacondacloud] enabled
[I 11:18:26.853 NotebookApp] ✓ nbpresent HTML export ENABLED
[W 11:18:26.854 NotebookApp] ✗ nbpresent PDF export DISABLED: No module named 'nbbrowserpdf'
[I 11:18:26.861 NotebookApp] Serving notebooks from local directory: /Users/allieus/askdjango/nabi-201610
[I 11:18:26.861 NotebookApp] 0 active kernels
[I 11:18:26.861 NotebookApp] The Jupyter Notebook is running at: http://localhost:8888/
[I 11:18:26.861 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
```



# 새 Jupyter Notebook 만들기

- 우측 메뉴에서 New -> Python



# 도움말 보기

- 도움말 가능 대상 : 함수, 클래스, 모듈, 팩키지 등
- 도움말 보는 방법
  - ipython쉘> **help**(대상) # python 쉘에서도 가능
  - ipython쉘> 대상? # ipython 쉘 혹은 Jupyter Notebook에서 가능

# 단축키 (명령모드)

- 진입하는 법 : ESC 키
- Enter : 편집모드 진입
- Cell 실행
  - Ctrl-Enter : Cell 실행
  - Shift-Enter : Cell 실행 및 아래 Cell 선택
  - Alt-Enter : Cell 실행, 아래에 새 Cell 추가
- Cell 타입 변환
  - Y : code 타입으로 변환
  - M : markdown 타입으로 변환
  - R : Raw 타입으로 변환

# 단축키 (수정 모드)

- 진입하는 법 : Enter 키
- Tab : 코드 자동완성 혹은 들여쓰기
- ESC 혹은 Ctrl-M : 커맨드 모드로 전환
- Ctrl-/ : 현재 선택된 줄에대해 주석 설정/해제 Toggle

# Markdown 문법

- Plain Text 파일
- 존 그루버 (John Gruber) 가 2004년에 개발되어, 다양한 확장문법이 존재
- Markdown 문서를 html, pdf, slide, html 등으로 변환이 가능
- Markdown Cells : 공식문서
- Marp : Markdown Presentation Writer (맥 전용)

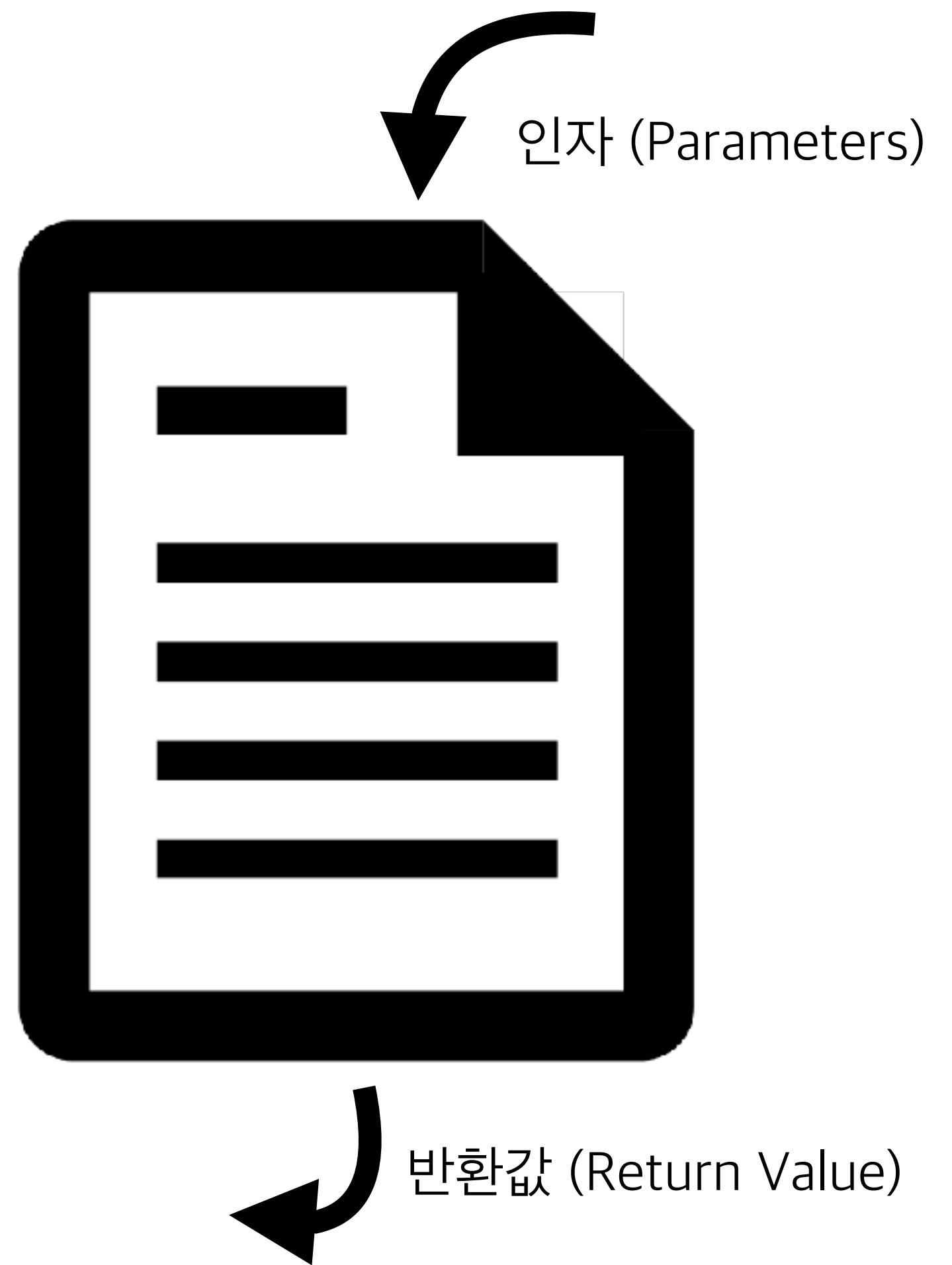
# 알림

- 앞으로 모든 과제는 jupyter notebook 파일인 ipynb 파일로 제출하세요.

# 함수 (Functions)

# 함수 (Functions)

- 함수는 <작업 지시서> 와 같은 개념
  - 코드 중복을 제거하기 위한 목적
- 함수의 구성
  - 1개의 **함수명** (필수) : 작업의 이름
  - 0개 이상의 **인자값** (옵션) : 작업에 필요한 정보
  - 1개의 **반환값** (옵션) : 작업의 결과를 하나 돌려받습니다.
- 코드의 중복을 제거하기 위해서 가장 필요한 문법
- 빌트인 함수 (Builtin Functions) : print, range 등



# 함수 (Functions) - 함수의 유형

- 인자 값, 반환 값 없는 함수

```
def myfn1():
 answer = input('Who are you? ')
 print(answer)
 return None

print(myfn1())
```

- 인자 값은 있지만, 반환 값은 없는 함수

```
def myfn2(a, b, c):
 result = a + b + c
 print('result = {}'.format(result))

print(myfn2(1, 2, 3))
```

- 인자 값과 반환 값이 모두 있는 함수

```
def myfn3(a, b, c):
 result = a + b + c
 return result

print(myfn3(1, 2, 3))
```

- Tip : 반환값이 없는 함수는 None 을 리턴한 것으로 처리됩니다.

# 파이썬에서는 함수명만으로 함수를 구분합니다.

- 앞선 함수와 같은 이름의 함수를 정의하면, 기존 함수를 덮어쓰게 됩니다.

```
def fn():
 print('called fn1()')

fn()

def fn():
 print('called fn2()')

fn()

def fn():
 print('called fn3()')

fn()
```

실행결과> python class-20161005/functions\_1.py  
called fn1()  
called fn2()  
called fn3()

# 함수를 만들어봅시다.

- 다양한 다이아몬드를 출력하는 함수

```
def print_diamond(size):
 print('TODO: {} 크기의 다이아몬드를 출력해봅시다.'.format(size))
```

```
print_diamond(3)
```

```
..
```

다음을 출력하세요.

```
*

 *
 ..
 ..
```

```
print_diamond(5)
```

```
..
```

다음을 출력하세요.

```
*

 *
 ..
 ..
```

# 변수의 유효범위 (Scope)

- 변수가 선언되어, 해당 변수가 영향을 미치는 영역
- 지역 변수 (Local Variable) : 함수 안에서 선언되어, 함수 내에서만 활용이 가능한 변수
- 전역 변수 (Global Variable) : 함수 밖에서 선언되어, 함수 안에서도 활용이 가능한 변수
  - 코드의 가독성을 해치므로, **절대 권장하지 않는** 방법
  - global 키워드가 필요
- 변수의 값이 변경되는 경우라면, 그 유효범위를 최소화하여 지역변수를 사용하는 것이 버그 발생확률을 획기적으로 낮출 수 있습니다.

# 인자 (Arguments)

- 함수가 실행되는데에 필요한 0개 이상의 변수 목록
- Positional Arguments : 인자의 **위치**에 기반한 인자

```
def fn_with_positional_arguments(name, age):
 print("당신의 이름은 {}이며, 나이는 {}입니다.".format(name, age))
```

- Keyword Arguments : 인자의 **이름**에 기반한 인자
  - 디폴트 인자와 같이 쓰입니다. 디폴트 인자값이 지정된 인자에 한해, 함수 호출 시에 해당 인자를 지정하지 않으면, 디폴트 인자값으로 값이 자동지정되어 함수가 호출됩니다.

```
def fn_with_keyword_arguments(name="", age=0):
 print("당신의 이름은 {}이며, 나이는 {}입니다.".format(name, age))
```

# 가변인자 (1)

- 인자의 갯수를 제한하지 않고, 다수의 인자를 받을 수 있습니다.
- 리스트/튜플을 통해, 다수의 값을 받는 방법

```
def fn1(colors):
 for color in colors:
 print(color)

fn1(['white', 'yellow', 'black'])
```

- 인자를 넘길 때, 리스트/튜플로 넘기지 않아도, **packing** 되어 **튜플**로 대입

```
def fn2(*colors):
 for color in colors:
 print(color)

fn2('white', 'yellow', 'black')
```

- 인자를 넘길 때, 리스트/튜플을 **unpacking** 하여 넘길 수 있습니다.

```
colors = ['white', 'yellow', 'black']
fn2(*colors)
```

# 가변인자 (2)

- 인자의 갯수를 제한하지 않고, 다수의 인자를 받을 수 있습니다.
- 사전 (Dictionary Type) 을 통해, 다수의 값을 받는 방법

```
def fn1(colors, scores):
 for color in colors:
 print(color)

 for key, score in scores.items():
 print(key, score)

fn1(['white', 'yellow', 'black'], {'apple': 10, 'orange': 5})
```

- 인자를 넘길 때, 사전으로 넘기지 않아도, **packing** 되어 **사전**으로 대입

```
def fn2(*colors, **scores):
 for color in colors:
 print(color)

 for key, score in scores.items():
 print(key, score)

fn2('white', 'yellow', 'black', apple=10, orange=5)
```

- 인자를 넘길 때, 사전을 **unpacking** 하여 넘길 수 있습니다.

```
colors = ['white', 'yellow', 'black']
scores = {'apple': 10, 'orange': 5}
fn2(*colors, **scores)
```

# collections.OrderedDict

- 공식문서 : <https://docs.python.org/3/library/collections.html#collections.OrderedDict>
- 대입된 순서를 저장하는 것을 제외하고는, 사전타입과 같습니다.

```
from collections import OrderedDict
```

```
mydict = OrderedDict()
mydict['a'] = 1
mydict['b'] = 2
mydict['c'] = 3
mydict['d'] = 4
mydict['e'] = 5

for key, value in mydict.items():
 print(key, value)
```

# collections.defaultdict

- 지정 키값이 존재하지 않을 때, 적용할 디폴트값을 함수로 지정할 수 있습니다. 함수의 리턴값을 디폴트값으로 씁니다.

```
message = '노랑 빨강 녹색 파랑 노랑 빨강 노랑 빨강'
```

```
from collections import defaultdict
```

```
word_count = defaultdict(int)
for word in message.split():
 word_count[word] += 1

print(word_count)
```

# 클래스 (Class)

# 클래스 (Class)

- 커스텀 <데이터 타입>으로서, 관련된 다수의 변수와 함수의 묶음으로 구성
  - ex) 사람 (Person) 클래스, 책 (Book) 클래스, 옷 (Cloth) 클래스
- Tip : 파이썬에서 함수명은 snake\_case, 클래스명은 CamelCase로 작성합니다.

# 클래스 (Class)

- 클래스 타입의 변수 = 인스턴스 (Instance)
- 인스턴스를 생성 = 지정 클래스 타입의 변수를 새로이 생성
  - **함수를 호출하듯이**, 클래스를 호출합니다.
    - ex) 인스턴스 = **클래스이름()**
  - 클래스가 호출이 될 때, 클래스 내 **\_init\_** 함수 (앞/뒤로 언더바 2개) 가 자동으로 호출됩니다. 이 함수를 **생성자 (Constructor)** 라 부릅니다. 클래스를 초기화하기 위한 목적으로 쓰입니다.
- 해당 변수 생성에 필요한 값이 있으면, "클래스 호출 (인스턴스 생성)" 시에 지정해줍니다.
  - ex) 인스턴스 = **클래스이름(인자1, 인자2)**
  - 클래스 내 변수/함수를 멤버변수/멤버함수라고 부릅니다. 혹은 인스턴스 변수 / 인스턴스 함수라 부르기도 합니다.

# 커스텀 클래스 - Person 클래스 (1)

- 여러 사람 (Person) 의 리스트를 관리하려 합니다.
  - 이때, Person 클래스를 만들어서, 보다 효율적으로 처리해보려 합니다.
- 한 사람에 대한 정보 중에 어떤 정보가 필요할까요? (설계의 문제)
  - 우리는 이름/나이/지역 정보만 저장하여, 함수를 통해 처리하려 합니다.
  - 그리고, 그 이름/나이/지역을 처리할 함수도 정의해봅시다.

# 커스텀 클래스 - Person 클래스 (2)

```
class Person(object):
 def __init__(self, name, age, region):
 self.name = name
 self.age = age
 self.region = region
```

```
Person 타입의 변수 tom을 생성
tom = Person('Tom', 10, 'Seoul')
print(tom.name)
print(tom.age)
print(tom.region)
```

```
Person 타입의 변수 jamie를 생성
jamie = Person('Jamie', 20, 'Pusan')
print(jamie.name)
print(jamie.age)
print(jamie.region)
```

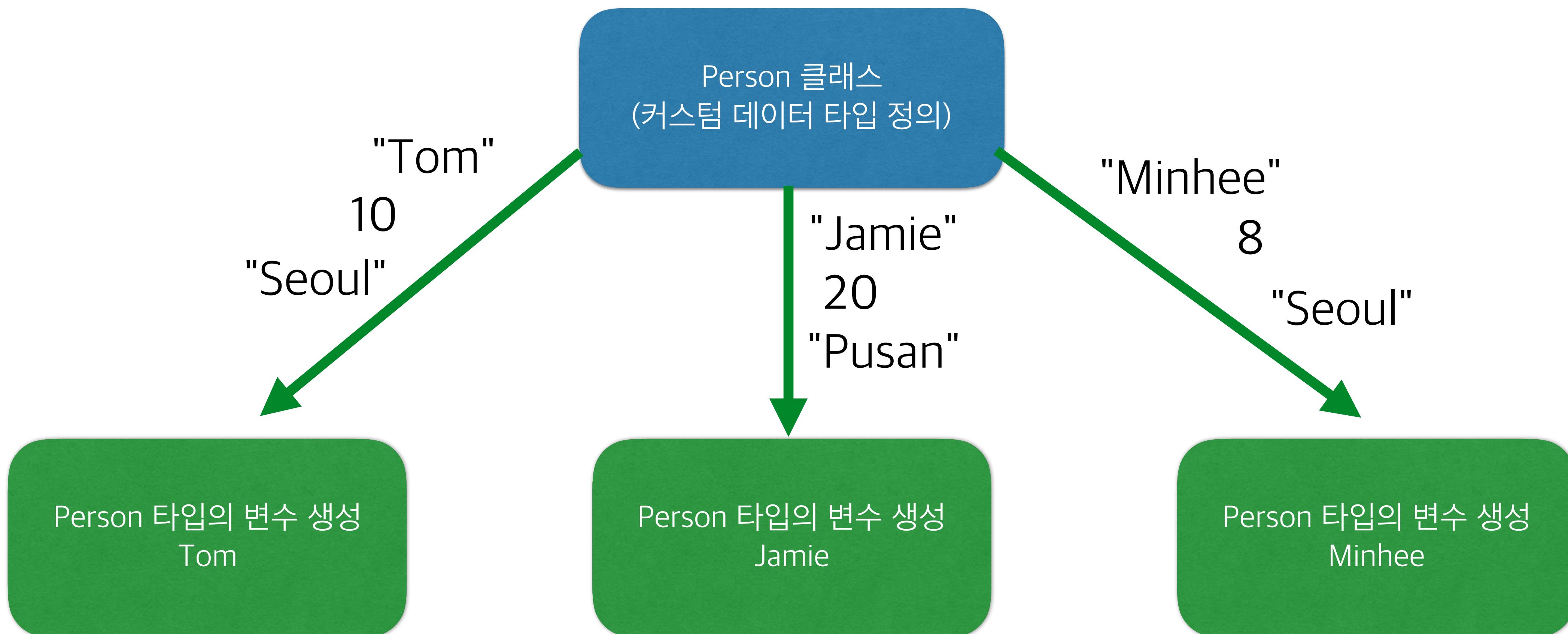
```
Person 타입의 변수 minhee를 생성
minhee = Person('Minhee', 8, 'Seoul')
print(minhee.name)
print(minhee.age)
print(minhee.region)
```

클래스 내, 함수의 첫번째 인자는 **self**로 지정합니다.  
이 인자에 직접 값을 넣지 않습니다. 함수 호출 시에 파이썬에서  
자동으로 대입합니다.

region 변수의 값을 해당 인스턴스 내 region 변수 (self.region)로 값  
을 복사합니다.

- 사람 (Person)에는 이름 (name), 나이 (age),  
지역 (region) 정보를 담아서 처리해보려 합니  
다.
- 각각의 멤버변수/멤버함수는 각 **인스턴스 내에  
서만 유효**하며, 서로 **격리된 공간**을 가집니다.

# 커스텀 클래스 - Person 클래스 (3)



# 커스텀 클래스 - Person 클래스 (4)

```

class Person(object):
 def __init__(self, name, age, region):
 self.name = name
 self.age = age
 self.region = region

 def say_hello(self):
 print('Hello, {} (from {})'.format(self.name, self.region))

 def move_to(self, region):
 self.region = region

tom = Person('Tom', 10, 'Seoul')
tom.say_hello()
tom.move_to('Pusan')
tom.say_hello()

```

Person 을 구성하는 데에 꼭 필요한 값들을  
`__init__` 함수를 통해서 받습니다.

한 Person 타입의 변수가 살아있는 동안에 그 값을  
 유지시킬려면, `self` 를 통해서 멤버변수로서 등록을 시켜야 합니다.

멤버함수에서는 `self` 를 통해, 멤버변수에 접근할 수 있습니다.  
`print('Hello, {} (from {})'.format(self.name, self.region))`

인자가 없는 멤버함수는 `self` 인자만을 취합니다.  
 이는 해당 함수를 호출 시에 파이썬에서 알아서 그 값을 채워줍니다.  
 우리가 직접 지정할 필요가 없습니다.

# 상속 (Inheritance)

- 코드 중복을 최소화하기 위한 목적으로 등장
- 클래스 간에 상속관계에 놓이게 되면, 부모 클래스 / 자식 클래스 관계가 성립
- 우리는 부모님의 자식이며, 누군가의 부모가 될 것입니다.

# 상속을 알기 전에는 ~~~

```
class Doctor:
 def __init__(self, name):
 self.name = name

 def run(self):
 print('뛰니다.')

 def eat(self, food):
 print('{}을 먹습니다.'.format(food))

 def sleep(self):
 print('잠을 잡니다.')

 def study(self):
 print('열심히 공부합니다.')

 def research(self):
 print('열심히 연구합니다.')
```

```
class Programmer:
 def __init__(self, name):
 self.name = name

 def run(self):
 print('뛰니다.')

 def eat(self, food):
 print('{}을 먹습니다.'.format(food))

 def sleep(self):
 print('잠을 잡니다.')

 def study(self):
 print('열심히 공부합니다.')

 def coding(self):
 print('열심히 코딩합니다.')
```

```
class Programmer:
 def __init__(self, name):
 self.name = name

 def run(self):
 print('뛰니다.')

 def eat(self, food):
 print('{}을 먹습니다.'.format(food))

 def sleep(self):
 print('잠을 잡니다.')

 def study(self):
 print('열심히 공부합니다.')

 def design(self):
 print('열심히 디자인합니다.')
```

# 상속을 알고 나면 ~ !!!

```

class Person(object):
 def __init__(self, name):
 self.name = name

 def run(self):
 print('뛰니다.')

 def eat(self, food):
 print('{}을 먹습니다.'.format(food))

 def sleep(self):
 print('잠을 잡니다.')

 def study(self, target):
 print('{}을 열심히 공부합니다.'.format(target))

```

Doctor, Programmer, Designer 클래스의  
**같은 동작을 하는 중복코드**를 Person 클래스에 정리하고,  
각 클래스는 Person 클래스를 상속받도록 합니다.  
동작은 이전과 동일합니다.

```

class Doctor(Person):
 def research(self):
 print('열심히 연구합니다.')

class Programmer(Person):
 def coding(self):
 print('열심히 개발합니다.')

class Designer(Person):
 def design(self):
 print('열심히 디자인을 합니다.')

```

# 모듈 (Modules) 과 팩키지 (Packages)

# 모듈 (Modules) 과 팩키지 (Packages)

- 파이썬 모듈 : 파이썬 소스코드 파일
- 파이썬 팩키지 : 파이썬 소스코드가 있는 디렉토리
  - 디렉토리에 `_init_.py` 파일이 있어야, <파이썬 팩키지>로서 인식을 합니다.
  - 없으면 `ImportError` 예외가 발생합니다.

# 다른 파이썬 소스코드의 함수/클래스 가져오기 (1)

- 다른 파이썬 소스코드의 함수/클래스 등을 지금의 소스코드로 가져올 수 (import) 있습니다.
- 다른 파이썬 소스코드를 가져온다 (Import) 는 것
  - Import 시점에서 해당 코드를 실행한다는 의미
  - 해당 소스코드의 클래스/함수를 현재 코드에서 쓸 수 있다는 의미
- 코드 예)
  - import 팩키지내함수경로
  - import 모듈내함수경로
  - from 팩키지경로 import 함수경로
  - from 모듈경로 import 함수경로

# 다른 파일의 소스코드의 함수/클래스 가져오기 (2)

- myimport1.py

```
import myimport2

def mysum(x, y):
 return x + y

if __name__ == '__main__':
 print(mysum(1, 2))
 print(myimport2.mymultiply(1, 2))
```

- myimport2.py

```
def mymultiply(x, y):
 return x * y
```

- 실행결과

- 셸> python myimport1.py

3

2

# 파이썬 소스코드 내 `_name_` 과 `_file_` 의 역할 (1)

- `_name_` : `_name_` 가 있는 파이썬 소스코드 파일의 파일명
  - ex) `hello_world.py` 의 경우 : `_name_` 은 "hello\_world"
- `_file_` : `_file_` 이 있는 파이썬 소스코드 파일의 경로명
  - ex) `c:\dev\hello_world.py`
- 단, **최초 진입** 소스코드일 경우 `_name_` 은 "`_main_`" 이 되지만, `_file_` 은 그대로 유지 됩니다.
  - 최초 진입 소스코드를 지정할 수 있습니다.

# 파이썬 소스코드 내 `_name_` 과 `_file_` 의 역할 (2)

- 다음 소스코드가 있습니다.

```
print('current _name_ is {}'.format(_name_))
print('current _file_ is {}'.format(_file_))
```

- 현재 경로에 name\_test.py 가 있을 경우

- 쉘> python3 name\_test.py

```
current _name_ is _main_
```

```
current _file_ is name_test.py
```

- 쉘> python3

```
>>> import name_test
```

```
current _name_ is name_test
```

```
current _file_ is /Users/allieus/dev/askdjango/snu-web-201609/name_test.py
```

# 파이썬 소스코드 내 \_\_name\_\_ 과 \_\_file\_\_ 의 역할 (3)

- 다음 소스코드가 있습니다.

```
print('current __name__ is {}'.format(__name__))
print('current __file__ is {}'.format(__file__))
```

- 현재 경로에 a/b/c/name\_test.py 가 있을 경우

- 쉘> python3 a/b/c/name\_test.py ← a/b/c/name\_test.py 로부터 파이썬 실행

```
current __name__ is __main__
current __file__ is a/b/c/ame_test.py
```

- 쉘> python3 ←

```
>>> import a.b.c.name_test
current __name__ is name_test
current __file__ is /Users/allieus/dev/askdjango/snu-web-201609/a/b/c/name_test.py
```

따로 파이썬을 실행하고나서, a/b/c/name\_test.py 를 가져옴.

- 주의 : 다음 3개 파일이 각 파이썬 팩키지 내에 존재해야만 import 가 가능합니다.

- a/\_\_init\_\_.py, a/b/\_\_init\_\_.py, a/b/c/\_\_init\_\_.py

- 파이썬 3.3 이상에서는 \_\_init\_\_.py 가 없어도 import 가 가능하지만, 호환성 보장과 가독성을 위해 \_\_init\_\_.py 파일을 꼭 넣어주세요.

# 파이썬 소스코드 내 `__name__` 과 `__file__` 의 역할 (4)

- 특정 파이썬 소스코드 내에서 `if __name__ == '__main__'` 을 쓰는 이유
  - 다른 소스코드에 의해서 Import 되었을 때가 아니라, 직접 실행되었을 때에만 실행할 코드가 필요할 경우

# 파이썬 소스코드 내 \_\_name\_\_ 과 \_\_file\_\_ 의 역할 (5)

- myimport1.py

```
import myimport2

def mysum(x, y):
 return x + y

print(mysum(1, 2))
print(myimport2.mymultiply(1, 2))
```

- myimport2.py

```
import myimport1

def mymultiply(x, y):
 return x * y

print(mymultiply(1, 2))
print(myimport1.mysum(1, 2))
```

- myimport1.py

```
import myimport2

def mysum(x, y):
 return x + y

if __name__ == '__main__':
 print(mysum(1, 2))
 print(myimport2.mymultiply(1, 2))
```

- myimport2.py

```
import myimport1
```

```
def mymultiply(x, y):
 return x * y
```

```
if __name__ == '__main__':
 print(mymultiply(1, 2))
 print(myimport1.mysum(1, 2))
```



Lift is short, use **Python**.

# 2016 벤처창업 웹프로그래밍 1

파이썬 Part.

#6

# 모두 잘 하셨어요. :D

- 기대 이상으로 모두모두 잘 해주셨습니다. ;D



# 간단 Tip

- 리턴하지 않은 함수는 None 을 리턴합니다.

# 함수 인자 디폴트값을 지정할 때, list/dict 를 쓰지마세요.

- 함수 fn1, fn2 의 인자 args 는 fn1, fn2 와 같은 block 에 위치합니다.
  - 즉 함수 호출되었을 때 지정된 디폴트값으로 초기화되지 않으며, 함수 호출이 끝나도 메모리 해제되지 않습니다.

```
def fn1(args=[]):
 args.append(1)
 print(args)

fn1()
fn1()
fn1()
```

실행결과  
 [1]  
 [1, 1]  
 [1, 1, 1]

```
def fn2(args=None):
 if args is None:
 args = [] # 빈 리스트로 새로이 변수 생성
 args.append(1)
 print(args)
```

fn2()  
 fn2()  
 fn2()

실행결과  
 [1]  
 [1]  
 [1]

# 부동 소수점과 고정 소수점

- 부동 소수점 (Floating Point) 형식 : float 타입
- 고정 소수점 (Fixed Point) : decimal.Decimal 타입
  - 정밀도가 필요한 연산에 사용

```
0.15 + 0.15 + 0.15
0.44999999999999996
```

```
from decimal import Decimal

Decimal('0.15') + Decimal('0.15') + Decimal('0.15')
Decimal('0.45')
```

# 표준 입출력

- 표준 입출력의 종류
  - 파이썬에서 `input()` 함수로 표준입력을 받습니다. 대개 키보드로 입력합니다.
- 표준 입력
  - 다음을 실행해서, `input.txt` 파일의 내용을 `myscript.py` 실행 시에 표준입력으로 들어갑니다.
    - 쉘/명령프롬프트> `python myscript.py < input.txt`
- 표준 출력
  - 혹은 다음을 실행해서, `myscript.py` 의 출력을 `output.txt` 파일에 즉시 저장할 수 있습니다. 파일이 없으면 생성되고, 기존에 있는 파일이면 **덮어쓰기**가 됩니다. `open` 에서의 `w` 모드와 같습니다.
    - 쉘/명령프롬프트> `python myscript.py > output.txt`
- 표준 에러
  - 혹은 다음을 실행해서, `myscript.py` 의 출력을 `output.txt` 파일에 **추가**할 수 있습니다. 파일이 없으면 생성되고, 기존에 있는 파일이면 추가가 됩니다. `open` 에서의 `w` 모드와 같습니다.
    - 쉘/명령프롬프트> `python myscript.py >> output.txt`

# 내장함수 enumerate

- 공식문서
- enumerate(iterable, start=0)

```
In [31]: mylist = ['a', 'b', 'c', 'd', 'e']
In [32]: i = 0
In [33]: for ch in mylist:
...: print(i, ch)
...: i += 1
...:
0 a
1 b
2 c
3 d
4 e

In [34]: for (i, ch) in enumerate(mylist):
...: print(i, ch)
...:
0 a
1 b
2 c
3 d
4 e

In [35]: for (i, ch) in enumerate(mylist, 1):
...: print(i, ch)
...:
1 a
2 b
3 c
4 d
```

# 파이썬을 파이썬답게 쓰기

# Import 경로

- import sys  
sys.path
- 파이썬에서 Import 를 수행할 때, sys.path 경로 상의 파이썬 모듈/팩키지를 읽어옵니다.
- sys.path 상의 빈 경로는 파이썬이 실행된 현재 디렉토리를 뜻합니다.
- sys.path 는 List 로서 앞에 위치할 수록, 우선순위를 가집니다.
- sys.path 는 List 이기 때문에, 경로를 자유롭게 추가/수정/삭제할 수 있습니다. 하지만 경로를 직접 수정하는 것은, 관리성이 나빠지기 때문에 권장하는 방법은 아닙니다.

# range 와 xrange (1)

- 파이썬 2에서는 따로 range, xrange 가 구분
- 파이썬 3에서는 range 가 제거되고, xrange 가 range 로 변경됨
- 차이
  - range : 가용값들을 미리 생성하고 시작하느냐 (+ 메모리 공간)
  - xrange : 값의 범위만 정해두고, 값을 그때 그때 생성해내느냐 (+ CPU 연산)

# range 와 xrange (2)

- 파이썬2에서 range 와 xrange 코드의 처리방식의 차이

```
-*- coding: utf-8 -*-
import datetime
```

```
print(datetime.datetime.now())
```

```
for i in range(30000000):
 print(i)
 break
```

본 range() 함수가 준비되는 데에, 약 8초

```
print(datetime.datetime.now())
```

```
for i in xrange(30000000):
 print(i)
 break
```

본 xrange() 함수가 준비되는 데에, 약 0.00001초

```
print(datetime.datetime.now())
```

- 쉘> python2 generator\_2\_for\_python2.py  
2016-07-11 08:11:06.301121  
0  
2016-07-11 08:11:14.016743  
0  
2016-07-11 08:11:14.016799

# range 와 xrange (3)

```
def myrange(start, end, step):
 mylist = []
 while start < end:
 mylist.append(start)
 start += step
 return mylist
```

```
def myxrange(start, end, step): # 코루틴 생성
 while start < end:
 yield start # generator 문법
 start += step
```

```
if __name__ == '__main__':
 for i in myrange(0, 10, 2):
 print(i)

 print('----')

 for i in myxrange(0, 10, 2):
 print(i)
```

# 코루틴 (Co-Routine)

- Sub Routine : 함수
  - 진입점이 하나
  - 부모/자식관계가 성립. 순차적으로 진행
  - 매 호출시마다, Routine 내 context 가 초기화
- Co Routine : 코루틴
  - 진입점이 여럿
  - 병렬적으로 진행
  - 여러번 호출이 되어도, Routine 내 Context 가 유지
  - 파이썬에서는 **Generator** 문법으로 간편히 구현

호출코드

```
def sub_routine():
 i = 10
 i += 1
 print(i)
 i += 1
 print(i)
 i += 1
 print(i)
```

호출코드

```
def co_routine():
 generator = co_routine()
 print(next(generator))
 print(next(generator))
 print(next(generator))
 print(next(generator))
```

호출코드

```
i = 10
i += 1
yield(i)
i += 1
yield(i)
i += 1
yield(i)
i += 1
yield(i)
```

**StopIteration 예외 발생 !!!**

# Generator 문법 (1)

- 공식문서 : <https://wiki.python.org/moin/Generators>
- 연속된 (Sequence) 값들을 생산해내는 함수
- 일반 함수에서 **return 문** 대신에 **yield 키워드**가 쓰여지면, Generator 가 됩니다.
- Generator 에서는 return 문은 쓰이지 않습니다.
- yield 한 값들이 순차적으로 생산됩니다.
- 끝에 도달하면 (추가 yield 가 없으면), StopIteration 예외 자동 발생합니다.
  - for 루프는 StopIteration 예외를 자동으로 처리합니다.
  - 관련 PEP문서

# Generator 문법 (2)

- 예시코드

```
def to_3():
 yield 0
 yield 1
 yield 2
 yield 3

generator = to_3()
print(generator)

for i in generator:
 print(i)
```

- 실행결과

- 셸> python generator\_1.py  
<generator object to\_3 at 0x1054ccba0>  
0  
1  
2  
3

# Generator 를 tuple/list 로 변환

- 조금 전 to\_3() 제너레이터를 다음과 같이 tuple/list 를 반환도록 할 수 있습니다.
  - >>> numbers = list(to\_3())
  - >>> numbers = tuple(to\_3())

# tuple/list/generator 를 dict 으로 변환

- dict 은 key, value 의 쌍으로 이뤄집니다.

- >>> mylist = [['a', 1], ['b', 2]]  
>>> dict(mylist)  
{'a': 1, 'b': 2}

- key 가 중복이 되면, 하나의 마지막 key/value 가 남습니다.

- >>> mylist = [['a', 1], ['b', 2], ['a', 3]]  
>>> dict(mylist)  
{'a': 3, 'b': 2}

- key, value 쌍이 맞지 않을 경우, ValueError 가 발생합니다.

- >>> mylist = [['a', 1], ['b', 2], ['a']]  
>>> dict(mylist)

ValueError: dictionary update sequence element #2 has length 1; 2 is required

# Generator 를 활용한 피보나치 수열

- 코드 #1

```
def fib(n):
 x, y, counter = 0, 1, 0
 while True:
 if counter > n:
 break
 yield x
 x, y = y, x + y
 counter += 1

for x in fib(10):
 print(x, end=' ')
```

- 코드 #2

```
def fib2():
 x, y = 0, 1
 while True:
 yield x
 x, y = y, x + y

counter = 0
for x in fib2():
 print(x, end=' ')
 counter += 1
 if counter > 10:
 break
```

# 순회 가능한 (Iterable) 객체

- set, list, dict, tuple, string, generator 는 모두 순회 가능한 객체
- Custom 클래스에 대해서도 `_iter_` 멤버함수를 구현하여, 순회가능토록 만들 수 있습니다.

```
print('# str : ', end=' ')
for ch in "hello world":
 print(ch, end=' ')
print()
```

```
print('# list : ', end=' ')
for i in [1, 2, 3]:
 print(i, end=' ')
print()
```

```
print('# tuple : ', end=' ')
for i in (1, 2, 3):
 print(i, end=' ')
print()
```

```
print('# set : ', end=' ')
for i in {1, 2, 1, 2, 1, 2}:
 print(i, end=' ')
print()
```

```
print('# dict (keys) : ', end=' ')
for key in {'a': 1, 'b': 2}:
 print(key, end=' ')
print()
```

```
print('# dict (keys) : ', end=' ')
for key in {'a': 1, 'b': 2}.keys():
 print(key, end=' ')
print()

print('# dict (values) : ', end=' ')
for value in {'a': 1, 'b': 2}.values():
 print(value, end=' ')
print()

print('# dict (items) : ', end=' ')
for key, value in {'a': 1, 'b': 2}.items():
 print(key, value, end=' ')
print()

print('# Generator : ', end=' ')
def to_3():
 yield 1
 yield 2
 yield 3

for i in to_3():
 print(i, end=' ')
print()
```

# 호출가능한 (Callable) 객체 (1)

- 함수를 **호출**해서, 리턴값을 취한다.
- 클래스를 **호출**해서, 인스턴스를 생성한다.
- 클래스의 인스턴스로 **호출가능토록** 만들 수 있습니다.

```
class Calculator(object):
 def __init__(self, base):
 self.base = base

calculator = Calculator(10)
print(calculator(1, 2))
```

```
$ python callable_object.py
Traceback (most recent call last):
 File "callable_object.py", line 7, in <module>
 print(calculator(1, 2))
TypeError: 'Calculator' object is not callable
```

오류메세지가 "인스턴스입니다." 가 아니라 "**호출불가합니다.**"

# 호출가능한 (Callable) 객체 (2)

- 인스턴스를 호출가능도록 만들려면, `_call_` 멤버함수를 구현합니다.

```
class Calculator(object):
 def __init__(self, base):
 self.base = base
```

```
def __call__(self, x, y):
 return self.base + x + y
```

```
calculator = Calculator(10)
print(calculator(1, 2))
```

```
$ python callable_object.py
13
```

# 익명 함수 (Anonymous Function)

- 파이썬에서는 lambda 식을 통해 익명함수를 생성
- return 문은 쓰지 않아도, 마지막 값을 리턴값으로 처리
- 대개 인자로 1줄 함수를 지정할 때, 많이 쓰인다.
- 일반 함수와 인자처리도 동일하게 처리된다. (Positional Arguments, Keyword Arguments)

```
def mysum1(x, y):
 return x + y

mysum2 = lambda x, y: x + y

print(mysum1(1, 2))
print(mysum2(1, 2))

mysum3 = lambda *args: sum(args)
print(mysum3(1, 2, 3, 4, 5, 6, 7))
```

# 1급(First Class) 함수, 고차(High Order) 함수

- 1급 함수의 조건

- 변수에 담을 수 있다.

```
def mysum(x, y):
 return x + y
```

- 인자(Parameter)로 전달할 수 있다.

```
def myfn(x, y, callback):
 return callback(x, y)

print(myfn(1, 2, mysum))
```

- 반환값(Return Value)으로 전달할 수 있다.

- 런타임 생성이 가능하다.

- 익명(Anonymous) 생성이 가능하다.

- 고차함수

- 다른 함수를 인자로 받거나, 그 결과로 함수를 반환하는 함수

```
def generate_sum_fn():
 def fn(x, y):
 return x + y
 return fn
```

```
fn2 = generate_sum_fn()
print(fn2(10, 20))
```

```
fn3 = lambda x, y: x + y
print(fn3(100, 200))
```

# 파이썬3, 빌트인 함수 활용 (1)

- 공식문서 : <https://docs.python.org/3/library/functions.html>
- sorted : sorted(iterable[, **key**][, reverse])
  - key : 정렬기준값을 계산할 함수를 지정. 값에 기반해 오름차순/내림차순 정렬. 새로이 정렬된 **리스트**를 리턴
- filter : filter(**function**, iterable)
  - function : 필터링할 함수를 지정. True 를 리턴한 항목으로만 구성된 **Generator** 를 리턴
- map : map(**function**, iterable, ...)
  - function : 값을 변환할 함수를 지정. 리턴값으로 구성된 새로운 **Generator** 를 리턴

# 파이썬3, 빌트인 함수 활용 (2)

- max

- max(iterable, \*[, **key**, default])

- iterable에서 key 함수를 거친 결과값 중에 가장 큰 결과값의 원래값을 반환

- default : iterable 이 비었을 경우, 이 값을 반환

- max(arg1, arg2, \*args[, **key**])

- arg1, args2, ... 중에서 key 함수를 거친 결과값 중에 가장 큰 결과값의 원래값을 반환

- min

- min(iterable, \*[, **key**, default])

- min(arg1, arg2, \*args[, **key**])

# 파이썬3, 빌트인 함수 활용 (3)

```
mylist = list(range(10))

print(sorted(mylist, key=lambda i: i%3, reverse=False))

print(list(filter(lambda i: i%3, mylist)))

print(list(map(lambda i: i*2, mylist)))

print(list(map(lambda i: i*2, filter(lambda i: i%3, mylist)))))

print(max(-10, 1, 2, 3, key=lambda i: abs(i)))
print(min(-10, 1, 2, 3, key=lambda i: abs(i)))

print(max(range(-10, 20), key=lambda i: abs(i)))
print(min(range(-10, 20), key=lambda i: abs(i)))
```

# 임의 기준으로 정렬하기 (1)

- 파이썬 공식문서 : <https://docs.python.org/3/howto/sorting.html>
- List 에는 sort 정렬함수 가 제공
  - 리스트를 재정렬
  - 리스트.sort(key=None, reverse=False)
    - key : 정렬기준을 만들 함수를 지정
    - reverse : 내림차순 정렬 여부
- sorted 내장함수
  - 인자로 제공된 **Iterable 변수 (tuple, list, dict, etc)** 내 순서를 변경하지 않고, 새로운 리스트를 생성

# 임의 기준으로 정렬하기 (2)

- 리스트.sort() 예시

```
>>> mylist = list(range(20))
>>> print(mylist)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> mylist.sort(key=lambda x: x%3)
>>> print(mylist)
[0, 3, 6, 9, 12, 15, 18, 1, 4, 7, 10, 13, 16, 19, 2, 5, 8, 11, 14, 17]
```

- sorted 예시

```
>>> mylist = list(range(20))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> print(mylist)
>>> mylist2 = sorted(mylist, key=lambda x: x%3)
>>> print(mylist)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> print(mylist2)
[0, 3, 6, 9, 12, 15, 18, 1, 4, 7, 10, 13, 16, 19, 2, 5, 8, 11, 14, 17]
```

# 대소 비교

```
>>> print(0 < 1)
True
>>> print('a' < 'b')
True
>>> print([9] < [0])
False
>>> print([0, 9] < [9])
True
```

문자열은 같은 위치의 문자끼리만 비교대상  
앞의 자리일수록 우선순위가 높습니다.

tuple/list 는 같은 자리수끼리만 비교대상  
앞의 자리일수록 우선순위가 높습니다.

# Quiz) 다음 코드를 완성하세요.

- Person 클래스 작성 (이름, 나이를 인자로 받습니다.)
- 나이 역순으로 정렬하기

# Quiz) 다수 기준으로 정렬하기

- 다음 리스트를 다음 기준으로 정렬해보세요.

- 1차 기준 : 자릿수

- 2차 기준 : 1의 자리 숫자

```
>>> mylist = [10, 11, 9, 20, 12, 313, 211, 121]
>>> # Quiz 코드 : sorted(...)
>>> print(mylist)
[9, 10, 20, 11, 12, 211, 121, 313]
```

- 힌트

- 숫자 11은 str(11)를 통해 문자열로 변환할 수 있습니다.
  - 문자열의 길이는 len(문자열)을 통해 알 수 있습니다.

# Comprehension (List, Dict, Set) (1)

- 순회가능한 객체를 조작하여, 필터링/새로운 리스트/사전/집합을 만들 수 있는 아주 간편한 방법
- 하지만, 과한 적용은 금물. 간단한 케이스에 대해서만 써주세요.
- List Comprehension
  - [ 표현식 **for** 변수 **in** 순회가능한객체 ]
  - [ 표현식 **for** 변수 **in** 순회가능한객체 **if** 필터링조건 ]
- Dict Comprehension
  - { Key:표현식 **for** 변수 **in** 순회가능한객체 }
  - { Key:표현식 **for** 변수 **in** 순회가능한객체 **if** 필터링조건 }
- Set Comprehension
  - { 표현식 **for** 변수 **in** 순회가능한객체 }
  - { 표현식 **for** 변수 **in** 순회가능한객체 **if** 필터링조건 }

# Comprehension (List, Dict, Set) (2)

```
val_list_comprehension1 = [i**2 for i in range(10)]
print(val_list_comprehension1)
```

```
mylist1 = []
for i in range(10):
 mylist1.append(i**2)
print(mylist1)
```

---

```
val_list_comprehension2 = [i**2 for i in range(10) if i % 2 == 0]
print(val_list_comprehension2)
```

```
mylist2 = []
for i in range(10):
 if i % 2 == 0:
 mylist2.append(i**2)
print(mylist2)
```

```
val_dict_comprehension = {i:i**2 for i in range(10)}
print(val_dict_comprehension)
```

```
mydict = {}
for i in range(10):
 if i % 2 == 0:
 mydict[i] = i**2
print(mydict)
```

---

```
val_set_comprehension = {i%5 for i in range(100)}
print(val_set_comprehension)
```

```
myset = set()
for i in range(10):
 if i % 2 == 0:
 myset.add(i%5)
print(myset)
```

# 제너레이터 (Generator) 표현식

- `[i**2 for i in range(1000000)]` # list comprehension
  - 한 번에 List 를 생성
- `(i**2 for i in range(1000000))` # generator expression
  - 값이 필요할 때마다, 값을 생성하여 **yield**
- `list((i**2 for i in range(1000000)))` 는 줄여서,
  - `list(i**2 for i in range(1000000))` 로 쓸 수 있습니다.

# 장식자 (Decorator) (1)

- 어떤 함수의 기능을 감싸는 (Wrapping) 용도의 함수

```
def verbose(fn):
 def wrap(x, y):
 print('begin function')
 ret = fn(x, y)
 print('end function')
 return ret
 return wrap

def mysum(x, y):
 return x + y

print(mysum(1, 2))
print('----')

mysum = verbose(mysum)
print(mysum(1, 2))
```

- (잠깐) 1급 함수 : 함수를 동적으로 생성할 수 있으며, 반환값으로 전달할 수 있다.

# 장식자 (Decorator) (2)

```
def mysum(x, y):
 return x + y
mysum = verbose(mysum)
```

```
@verbose
def mysum(x, y):
 return x + y
```

# 장식자 (Decorator) (3)

```
import time
```

```
def memoize(fn):
 cached = {}
 def wrap(x, y):
 key = (x, y)
 if key not in cached:
 cached[key] = fn(x, y)
 return cached[key]
 return wrap
```

```
@memoize
def myfn(x, y):
 time.sleep(1)
 return x + y
```

```
print(myfn(1, 2))
print(myfn(1, 2))
print(myfn(1, 2))
```

# 과제

- Book 클래스를 만들고, Book 리스트를 다양한 기준으로 정렬 (Sort) 하는 예를 3개 이상 만들어보세요.
- Book 클래스 멤버변수 : 책 제목, 책값, 저자, 출간년도, 판매부수
- ex) 책 제목 단어 갯수로 내림차순 정렬
- 인자를 받는 장식자 base 를 구현해보세요.

```
@base(10)
def sum(x, y):
 return x + y

print(sum(1, 2))
```

# 과제 이메일 제출 양식

- 제출방법 : ask@festi.kr로 제출
- [2016 벤처창업웹프로그래밍1] 전공, 학번, 이름
- Jupyter Notebook 파일로 제출



Life is short, use **Python**.

# 2016 벤처창업 웹프로그래밍 1

파이썬 Part.

#7

# 재귀 함수 (Recursion Function)

- 자기 자신을 호출하는 함수
- 파이썬에서는 최대 함수 호출 depth 제한
- 위키피디아

재귀함수를 쓰지 않은 구현

```
def factorial(n):
 result = 1
 for i in range(n, 1, -1):
 result *= i
 return result
```

```
factorial(10)
3628800
```

```
def factorial(n):
 if n <= 1:
 return 1
 else:
 return n * factorial(n-1)

factorial(10)
3628800
```

```
In [1]: def fn():
...: return fn()
...:

In [2]: fn()

RecursionError
<ipython-input-2-fd7064640434> in <module>()
----> 1 fn()

<ipython-input-1-0375f6d41d8b> in fn()
 1 def fn():
----> 2 return fn()

... last 1 frames repeated, from the frame below ...

<ipython-input-1-0375f6d41d8b> in fn()
 1 def fn():
----> 2 return fn()

RecursionError: maximum recursion depth exceeded
```

```
In [3]: import sys

In [4]: sys.getrecursionlimit()
Out[4]: 1000
```

# File-like object

- 오리 타이핑 (Duck Typing)
- 파일-object와 같은 인터페이스를 제공하면 파일-object처럼 취급한다.
  - 순회
  - read : 입력의 끝까지 모두 읽어서, 그 값을 반환
  - readline : 1줄만 읽어서, 그 값을 반환
  - write : 인자로 지정한 값을 파일에 쓰기
- 표준입력 (sys.stdin), 표준출력 (sys.stdout), 표준에러 (sys.stderr)
- 메모리 기반의 File Object : io.BytesIO, io.StringIO

# 파이썬 코드에서 인자받기

- 코드가 시작될 때, 명령행으로 인자 받기 (일반적)

```
import sys
print(sys.argv)
```

- 코드가 시작되고 나서, 표준입력으로 인자받기

- 방법1) 내장함수 input 을 활용
- 방법2) sys.stdin로부터 읽기 (입력의 끝은 Ctrl-D)

- 한번에 여러 줄의 표준입력을 받을 때 유용

```
import sys
lines = [line.strip() for line in sys.stdin]
print(lines)
```

```
→ ~ vi test_argv.py
→ ~ python test_argv.py
['test_argv.py']
→ ~ python test_argv.py -a
['test_argv.py', '-a']
→ ~ python test_argv.py --name=10
['test_argv.py', '--name=10']
→ ~ python test_argv.py --count=10
['test_argv.py', '--count=10']
```

→ ~ python test\_argv.py

```
3
1 0
5
4 2
1 2 3 4
6 0
1 1 9 1 1 1
```

입력의 끝으로서  
Ctrl-D 입력 포인트



['3', '1 0', '5', '4 2', '1 2 3 4', '6 0', '1 1 9 1 1 1']

# 파이썬 코드에서 인자받기 (연습문제)

- 다수 Book 클래스 정보 (책 제목, 책 가격, 저자, 출간년도, 판매부수 등) 를 sys.stdin 을 통해 받아보기

# 자료구조와 알고리즘

- 자료구조 (Data Structures) : 어떠한 데이터 뭉치를 효율적으로 저장/관리하는 구조
- 알고리즘 (Algorithms) : 어떠한 문제를 효율적으로 해결하는 방법
- 각종 자료구조/알고리즘 책 리뷰 by 한상곤 님

# Algorithm Online Judge

- [librewiki] 알고리즘 온라인 저지 사이트 소개
- 코딩도장 : <http://codingdojang.com>
- Algospot : <https://algospot.com>
- Baekjoon Online Judge : <https://www.acmicpc.net>
  - 메뉴) 문제 추천

# 기본 자료구조 몇 가지

- 큐 (Queue) : 선입선출, 우선 순위가 같은 작업 예약, 콜센터 고객 대기
  - 우선순위 큐 (Priority Queue)
- 스택 (Stack) : 후입선출, 함수 호출 구현, 웹 브라우저 방문기록, Undo 기능
- 그래프 (Graph) : 운송 네트워크 표현 (비행노선, 고속도로망), 종속 표현 (예시)
  - 트리 (Tree) : 나무와 유사한 계층적 구조를 가진 자료구조, **순환을 갖지 않음**. 의사결정 트리, 딕토리  
구조
- 해쉬 (Hash) : 두 해시 값이 다르다면 원래의 데이터도 다르다는 것
  - hashlib : md5, sha1, sha224, sha256, sha384, sha512

# queue.Queue

- 선입선출 자료구조 ([공식문서](#))
- Queue Objects
  - Queue.qsize(), Queue.emtpy(), Queue.full(), Queue.put(item), Queue.get(block=False)
- list 를 통해 **유사하게** 컨셉 구현

```
class Queue(list):
 def qsize(self):
 return len(self)

 def empty(self):
 return not bool(self)

 def full(self):
 return not self.empty()

 def put(self, item):
 self.append(item)

 def get(self):
 return self.pop(0)
```

# queue.PriorityQueue

- 우선순위에 기반한 선입선출 자료구조 (공식문서)
- get 시에 대소비교에서 작은 Item 부터 획득
- list 를 통해 **유사하게** 컨셉 구현

```
class PriorityQueue(list):
 def qsize(self):
 return len(self)

 def empty(self):
 return not bool(self)

 def full(self):
 return not self.empty()

 def put(self, item):
 self.append(item)
 self.sort()

 def get(self):
 return self.pop(0)
```

# queue.PriorityQueue (연습문제)

- 프린터 쿠 (출처: Baekjoon Online Judge)
  - 프린터 기기는 인쇄명령을 받은 "순서대로", 먼저 요청된 것을 먼저 인쇄합니다. 여러 개가 쌓인다면 Queue 자료구조에 쌓여서 FIFO (First In First Out)에 따라 인쇄가 시작됩니다. 이때 중요도가 높은 문서부터 인쇄를 해야합니다.
  - 예를 들어 Queue에 4개의 문서 (A B C D)가 있고, 중요도가 (2 1 4 3)이라면, (C D A B) 순으로 인쇄가 이뤄져야 합니다.
  - 이를 파이썬 코드로 표현해보세요.

# collections.deque

- 큐와 스택에 대한 범용 구현 (공식문서)

```
>>> from collections import deque
>>> queue = deque(["Eric", "John", "Michael"])
>>> queue.append("Terry") # Terry arrives
>>> queue.append("Graham") # Graham arrives
>>> queue.popleft() # The first to arrive now leaves
'Eric'
>>> queue.popleft() # The second to arrive now leaves
'John'
>>> queue # Remaining queue in order of arrival
deque(['Michael', 'Terry', 'Graham'])
```

# Graph, 운송 네트워크 표현

- <https://www.python.org/doc/essays/graphs/>

```
graph = {'A': ['B', 'C'],
 'B': ['C', 'D'],
 'C': ['D'],
 'D': ['C'],
 'E': ['F'],
 'F': ['C']}
```

# 정렬 (Sorting)

- 일정한 순서대로 나열하는 알고리즘 (위키피디아, 나무위키)
- 공식문서 : Sorting HOW TO - 파이썬에서의 Sort 라이브러리 활용법
  - list 의 sort 멤버함수와 sorted 내장함수
  - 정렬기준을 생성하는 함수 지정 가능
  - 몇 가지 정렬 알고리즘 체험 : Quick Sort, Bubble Sort

# K-리그, 이동국 선수의 개인 기록

```
import requests
from bs4 import BeautifulSoup
from itertools import groupby

def get_records(player_id):
 url = 'http://www.kleague.com/KOR_2016/record/data_5.asp'
 params = {'player': player_id}

 html = requests.get(url, params=params).text
 soup = BeautifulSoup(html, 'html.parser')

 header = [th_tag.text for th_tag in soup.select('.table1-col thead th')]

 rows = []
 for tr_tag in soup.select('.table1-col tbody tr'):
 cols = [td_tag.text for td_tag in tr_tag.select('td')]
 if rows:
 diff_size = len(rows[-1]) - len(cols)
 if diff_size > 0:
 cols = rows[-1][:diff_size] + cols
 rows.append(cols)

 records = [dict(zip(header, row)) for row in rows]
 return records
```

정렬순서 안에서  
그룹핑

```
if __name__ == '__main__':
 records = get_records('19980443') # 이동국

 print('출전수 : ', len(records))
 print('총득점 : ', sum(int(record['득점']) for record in records))

 print('대회별 득점')

 group_list = []
 for criteria, group in groupby(records, lambda record: record['대회']):
 total_score = sum(int(record['득점']) for record in group)
 print(criteria, total_score)
 group_list.append((criteria, total_score))

 group_list.sort(key=lambda row: row[1], reverse=True)
 print('Top3 대회 :', group_list[:3])

 print('Home/Away 별 득점')
 records.sort(key=lambda record: record['H'])

 for criteria, group in groupby(records, lambda record: record['H']):
 total_score = sum(int(record['득점']) for record in group)
 print(criteria, total_score)
```

# 과제

# 과제 이메일 제출 양식

- 제출방법 : ask@festi.kr로 제출
- [2016 벤처창업웹프로그래밍1] 전공, 학번, 이름
- Jupyter Notebook 파일로 제출



Life is short, use **Python**.

# 2016 벤처창업 웹프로그래밍 1

파이썬 Part.

#8

# MRO (Method Resolution Order)

- 파이선의 클래스 탐색순서는 MRO 를 따릅니다.
  - Class.\_\_mro\_\_ 를 통해 확인 가능
- MRO 가 꼬이도록 클래스를 설계할 수는 없습니다.
  - TypeError: Cannot create a consistent method resolution order (MRO)

```
class A:
 pass

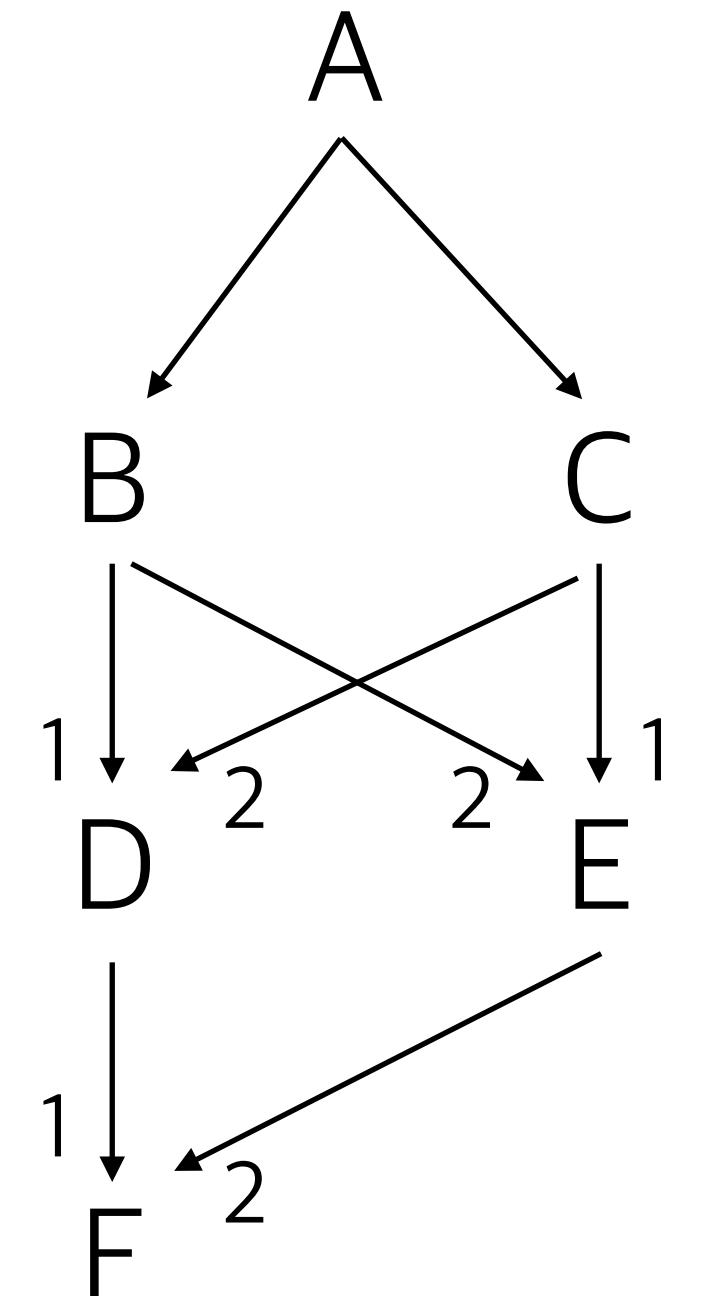
class B(A):
 pass

class C(A):
 pass

class D(B, C):
 pass

class E(C, B):
 pass

class F(D, E):
 pass
```



# 부모의 함수 호출

- 내장함수 super 를 통해 부모의 함수 호출

  - D().fn() 의 실행결과로서 A, C, B, D 가 출력

- super 호출 시에 MRO 에 기반하여 호출

## 파이썬3 문법

```
class A:
 def fn(self):
 print('A')
```

```
class B(A):
 def fn(self):
 super().fn()
 print('B')
```

```
class C(A):
 def fn(self):
 super().fn()
 print('C')
```

```
class D(B, C):
 def fn(self):
 super().fn()
 print('D')
```

## 파이썬2 호환문법

```
class A:
 def fn(self):
 print('A')
```

```
class B(A):
 def fn(self):
 super(B, self).fn()
 print('B')
```

```
class C(A):
 def fn(self):
 super(C, self).fn()
 print('C')
```

```
class D(B, C):
 def fn(self):
 super(D, self).fn()
 print('D')
```

D().fn()

# 예외 (Exceptions)

# 예외 (Exceptions)

- 소스코드 실행 중에 에러가 발생하는 경우
- 빌트인 예외
  - Exception : 최상위 예외

# 흔히 만나는 예외

- StopIteration : object.\_\_next\_\_() 내에서 발생. Iterator 내에서 더 이상 생산할 Item이 없을 때 발생
- AttributeError : attribute 참조 실패 혹은 설정이 실패한 경우에 발생
- ImportError : 지정 모듈/팩키지를 Import 하지 못한 경우에 발생
  - 파이썬은 sys.path 리스트 상에 명시된 경로순서대로 모듈/팩키지를 탐색합니다.
- IndexError : 범위 밖의 인덱스 참조시에 발생
- KeyError : 존재하지 않는 Key에 접근시에 발생
- NameError : local/global name 을 찾지못한 경우
- TypeError : 부적절한 연산/함수를 적용했을 때 발생, ex) 1 + '1'
- ValueError : 부적절한 값을 발견했을 때 발생, ex) int('a')
- NotImplementedError : 오버라이딩이 되지 않은 abstract 멤버함수가 호출되었을 때 발생
- IndentationError : 소스코드 내에 부적절한 들여쓰기가 적용되어있을 경우 발생

# 참고 : Import 경로

- import sys  
sys.path
- 파이썬에서 Import 를 수행할 때, sys.path 경로 상의 파이썬 모듈/팩키지를 읽어옵니다.
- sys.path 상의 빈 경로는 파이썬이 실행된 현재 디렉토리를 뜻합니다.
- sys.path 는 List 로서 앞에 위치할 수록, 우선순위를 가집니다.
- sys.path 는 List 이기 때문에, 경로를 자유롭게 추가/수정/삭제할 수 있습니다. 하지만 경로를 직접 수정하는 것은, 관리성이 나빠지기 때문에 권장하는 방법은 아닙니다.

# 예외 처리 (try, except, else, finally)

- tuple로 예외를 다수 지정할 수 있습니다.
- as를 통해 예외 인스턴스를 획득 가능
- else : 예외가 발생하지 않았을 때 호출되는 블럭
- finally : 예외 발생 유무에 상관없이 호출되는 블럭

```

try:
 1/0
except ValueError:
 print('ValueError 가 발생했어요.')
except (ValueError, TypeError):
 print('ValueError/TypeError 중에 하나가 발생')
#except ZeroDivisionError:
print('0으로 나누지마세요.')
except ZeroDivisionError as e:
 print('0으로 나누지마세요. : {}'.format(e))
else:
 print('예외가 발생하지 않았어요.')
finally:
 print('마무리!!!')

```

# 사용자 예외 정의

- 좀 더 디테일한 예외 처리를 위해, 기존 예외를 상속받아서 사용자 예외를 정의할 수 있습니다.

```
class TooBigNumberException(Exception):
 def __init__(self, value):
 self.value = value

 def __str__():
 return 'too big number {}'.format(self.value)

i = 10
if i > 5:
 raise TooBigNumberException(3)
```

# 과제 & 기말고사 예상

- 1이상 100000이하의 수를 랜덤 100회 생성하여, 1의 자리수값으로 그룹을 지어, 많은 숫자가 포함된 그룹부터 TOP3 출력
  - 랜덤값 획득 코드 예) import random; number = random.randint(1, 10000)
- Callable Object 만들기 : 함수 버전, 클래스 버전
- 장식자 (Decorator) : 함수 버전, 클래스 버전
- Generator 만들기 : 무한 팩토리얼, 무한 랜덤값
- cp949 인코딩의 CSV 파일 생성하기
- filter/sort/map, comparison
- List/Dict/Set Comprehensionss
- 마지막 시간 내용

# 과제 이메일 제출 양식

- 제출방법 : ask@festi.kr로 제출
- [2016 벤처창업웹프로그래밍1] 전공, 학번, 이름
- Jupyter Notebook 파일로 제출



Life is short, use **Python**.

# 2016 벤처창업 웹프로그래밍 1

파이썬 Part.

#9

# 과제 & 기말고사 예상

- 10이상 100000이하의 수를 랜덤 100회 생성하여, 1의 자리수값으로 그룹을 지어, 많은 숫자가 포함된 그룹부터 TOP3 출력
  - 랜덤값 획득 코드 예) `import random; number = random.randint(1, 10000)`
- Callable Object 만들기 : 함수 버전, 클래스 버전
- 장식자 (Decorator) : 함수 버전, 클래스 버전
- Generator 만들기 : 무한 팩토리얼, 무한 랜덤값
- cp949 인코딩의 CSV 파일 생성하기
- filter/sort/map, comparison
- List/Dict/Set Comprehensionss
- 정규표현식 작성
- 테스트 작성

# 정규 표현식 (Regular Expression)

# 정규 표현식

- 문자열의 패턴, 규칙, Rule 을 정의하는 방법
- 이를 통해, 문자열 검색이나 치환작업을 간편하게 처리할 수 있습니다.
- <https://docs.python.org/3/library/re.html>
  - re.match : 문자열 전체매칭
  - re.search : 문자열 부분매칭
  - re.sub : 지정 패턴의 문자열을 다른 문자열로 변경
- [Regular Expression HOWTO](#)

# 한 글자를 나타내는 패턴

- 숫자 1글자 : [0123456789] 또는 [0-9] 또는 \d
- 알파벳 소문자 1글자 : [abcdefghijklmnopqrstuvwxyz] 혹은 [a-z]
- 알파벳 대문자 1글자 : [ABCDEFGHIJKLMNOPQRSTUVWXYZ] 혹은 [A-Z]
- 알파벳 대/소문자 1글자 : [a-zA-Z]
- 화이트 스페이스 : [ \t\n\r\f\v] 혹은 \s
- 16진수 1글자 : [0-9a-fA-F]
- 문자열의 시작을 표시 : ^
- 문자열의 끝을 표시 : \$
- 한글 1글자 : "ㄱ-ㅎ"

# 반복 횟수 지정

- 숫자 0회 또는 1회 : \d?
- 숫자 0회 이상 : \d\*
- 숫자 1회 이상 : \d+
- 숫자 2글자 : \d{2}
- 숫자 3글자 이상, 5글자 이하 : \d{3,5} # 띄워쓰기 하나에도 민감합니다.

# 정규 표현식, 예시

- 휴대폰 번호인지 검사
- 이메일인지 검사
- 아이디에 f\*\*\*k 이 포함되어있는지 검사
- 입력받은 이름이 김씨이며, 이름이 "석"으로 끝나는지 체크해보자.
- 입력받은 문자열 내에서 숫자만 다 뽑아보자.
- 특정 문자열 내에서 "김"씨 성을 가진 이름을 다 뽑아보자.

# Quiz : 휴대폰번호 체크

- 번호는 010, 016, 017, 018, 019로 시작할 수 있습니다.
- 총 휴대폰 번호 자릿수는 하이픈을 제외하고, 10자리 혹은 11자리가 될 수 있습니다.
- 하이픈은 포함될 수도, 포함되지 않을 수도 있습니다.

테스트

# 테스트

- 작성한 코드를 테스트해야합니다.
- 누가 테스트하죠 ??? - 나 아니면 누군가. 혹은 단지 운에 맡깁니다.
  - 이를 테스트 라이브러리를 통해, 시간을 절약하고, 마음의 안식을 얻을 수 있습니다.
  - 변경에 대한 자신감 : 이거 하나 바꾸면, 소스코드가 제대로 동작안하는 건 아닐까???
- 어디까지 테스트해야 하나요?
  - 메인 로직을 검증
  - 많은 혹은 과도한 테스트 루틴은 유지보수 비용을 높입니다. 적절하게 해야죠.
- 테스트는 얼마나 자주 실행하나요?
  - 자주 실행하면 좋습니다.

# 좋은 테스트란?

- 한 번에 하나의 테스트
- 명확한 실패
- 빠른 테스트
- 중복되지 않기
- 자동화
- 독립적 (다른 것의 영향을 받지 않기)

# 단위 테스트와 기능 테스트

- Unit Test (단위 테스트)
  - 함수 단위
- Function Test (기능 테스트)
  - 요구사항 단위
  - Fixture 를 사용

# pytest

- 공식문서 : <http://doc.pytest.org/>
- 파이썬 테스트 강추 라이브러리 (unittest라는 파이썬 표준 라이브러리가 있지만)
- 설치> pip3 install pytest
- 초간단 사용방법
  - 특정 프로그램을 위한 디렉토리 생성 후, 코드 구현
  - 파일명/함수명/클래스명을 pytest 네이밍 룰에 따라 명명
    - 해당 루틴 내에서 assert 참거짓표현식을 통해, 테스트 검증
  - 테스트 수행 : 쉘> pytest

```
def func(x):
 return x + 1
```

```
def test_answer():
 assert func(3) == 5
```

# pytest 네이밍 룰

- 테스트를 수행할 파일명은 test\_ 로 시작도록 함.
- 소스코드내 테스트를 수행할 ...
  - 클래스 이름은 test\_ 로 시작
  - 클래스 이름은 Test 로 시작해야 하며, 멤버함수 이름은 test\_ 로 시작

# pytest 연습 (함수 기반)

- 인자로 받은 숫자를 모두 더하는 mysum 테스트
- 장식자 base\_number 테스트

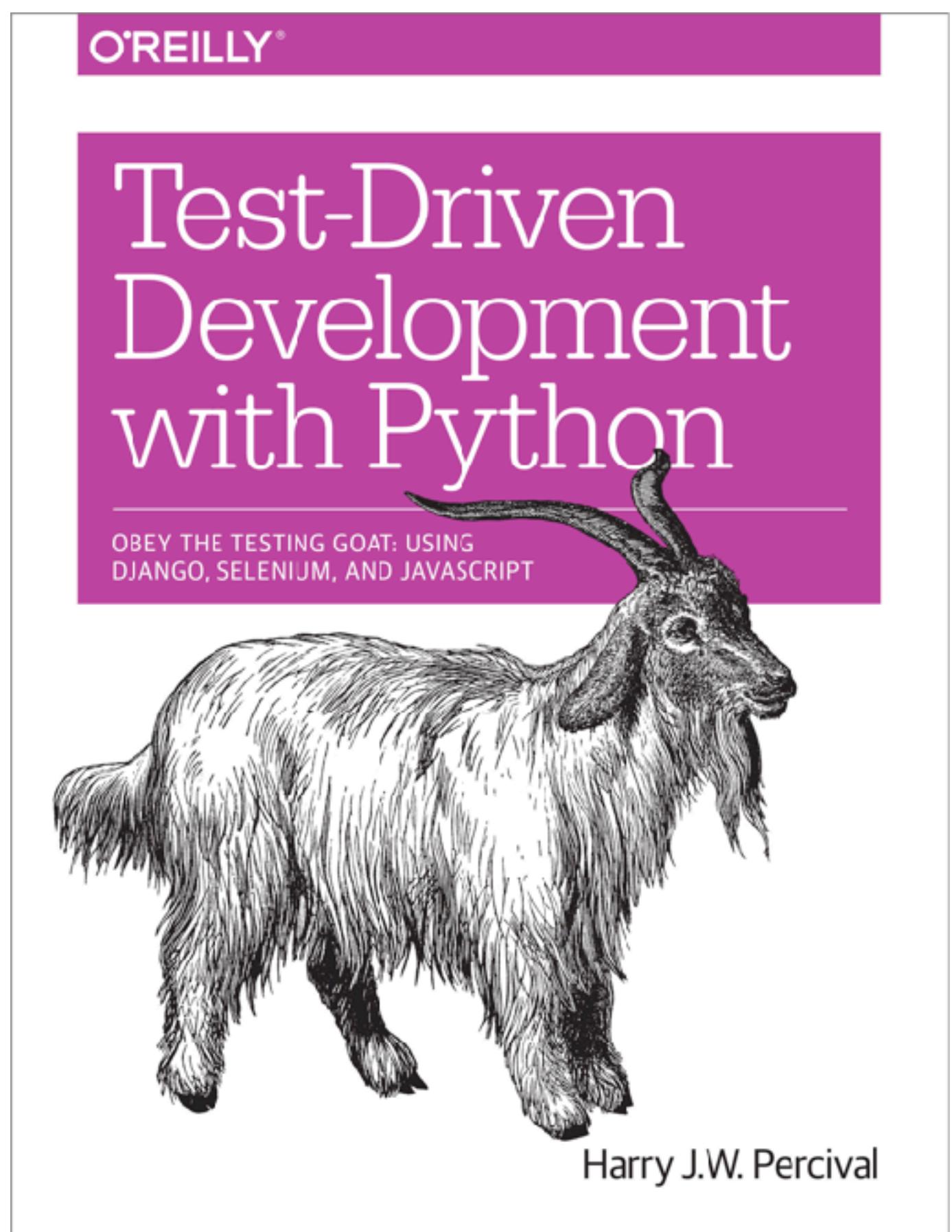
# pytest-html

- 소스코드 저장소 : <https://github.com/pytest-dev/pytest-html>
- 설치> pip<sup>3</sup> install pytest-html
- 사용법
  - 쉘> pytest --html=생성할파일경로.html

| Environment                                                                                                                                                                                                                                        |                                 |          |  |  |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------|----------|--|--|
| Platform                                                                                                                                                                                                                                           | Darwin-16.1.0-x86_64-i386-64bit |          |  |  |
| Python                                                                                                                                                                                                                                             | 3.5.2                           |          |  |  |
| Summary                                                                                                                                                                                                                                            |                                 |          |  |  |
| 2 tests ran in 0.02 seconds.<br><br>(Un)check the boxes to filter the results.                                                                                                                                                                     |                                 |          |  |  |
| <input checked="" type="checkbox"/> 2 passed, <input type="checkbox"/> 0 skipped, <input type="checkbox"/> 0 failed, <input type="checkbox"/> 0 errors, <input type="checkbox"/> 0 expected failures, <input type="checkbox"/> 0 unexpected passes |                                 |          |  |  |
| Results                                                                                                                                                                                                                                            |                                 |          |  |  |
| <a href="#">Show all details</a> / <a href="#">Hide all details</a>                                                                                                                                                                                |                                 |          |  |  |
| Result                                                                                                                                                                                                                                             | Test                            | Duration |  |  |
| Passed ( <a href="#">show details</a> )                                                                                                                                                                                                            | test.py::test_answer            | 0.00     |  |  |
| Passed ( <a href="#">show details</a> )                                                                                                                                                                                                            | test.py::test_base_number       | 0.00     |  |  |

# 참고

- The Hitchhiker's Guide to Python : Testing Your Code
- Pycon KR 2015 세션
  - Python and test
  - Python 테스트 시작하기
- Test-Driven Development with Python
- Getting started with pytest





Life is short, use **Python**.