Python 프로그래밍

마지막 수정일 : 2016.11.30 by girin

참고자료

http://www.slideshare.net/ssuserd8e91f/python-if-loopfunction □ □□ □□

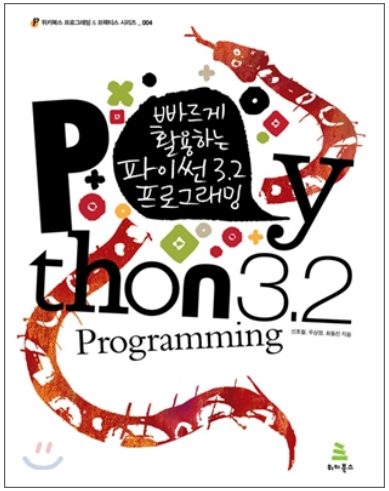
https://github.com/wikibook/python32 □□□□

□□ □□□ □□□ □□□

https://google.github.io/styleguide/pyguide.html

Python IDE □□□□

https://www.jetbrains.com/pycharm/



http://creativeworks.tistory.com/category/Programming/Python%20Tutorials

파이썬 (Python) 이란?

Gudi Van Rossum

- 1991년 귀도 반 로섬(Gudi Van Rossum)이 발표
- 플랫폼 독립적인 인터프리터 언어며,
- 객체 지향적, 동적 타이핑 언어
- 처음 C언어로 구현되었음

알아두면 좋은 이야기

- 귀도 반 로섬은 1989년 크리스마스에 할 일이 딱히없어 파이썬을 개발하였다고 함
- "파이썬" 이라는 이름은 코메디 프로그램 "Monty Python's Flying Circus" 에서 유래
- 파이썬의 원래 의미는 그리스 신화에 나오는 거대한 뱀
- 귀도는 구글에 근무했고 현재 DropBox에서 근무 중
- 귀도는 파이썬 개발자를 찾는 헤드헌터로 부터 취업 제안 메일을 받은 적이 있음

파이썬 (Python)

파이썬의 로고

두 마리의 뱀이 겹쳐 있는 모습

파이썬의 특징 1

플랫폼(?)

독립적인(?)

인터프리터(?) 언어

파이썬의 특징

플랫폼 = OS

: 윈도우, 리눅스, 안드로이드, 맥OS, iOS 등 프로그램이
실행되는 운영 체제를 플랫폼이라고 함

독립적인 = 관계없는, 상관없는

: OS에 상관없이 한번 프로그램을 작성하면 사용가능

인터프리터 = 통역기를 사용하는 언어...

: 소스코드를 바로 실행할 수 있게 지원하는 프로그램 실행 방법

원말인지 모르면 넘어가자. 괜찮다.

파이썬의 특징 2

- 가독성
  - 간결하고 가독성이 좋습니다.
  - 코드블럭을 들여쓰기(indentation)로 구분.
- 풍부한 라이브러리
  - 매우 광범위한 라이브러리가 기본으로 포함되어 있음.
  - 외부 라이브러리가 풍부하며 확장이 쉬움.
- 접착성
  - 쉽게 라이브러리를 추가할 수가 있습니다.
  - 파이썬에서 C로 구현된 부분을 사용할 수도 있으며,
    C에서 파이썬을 사용할 수도 있습니다.
- 무료
  - 파이썬은 파이썬 소프트웨어 재단(Python Software Foundation)에서 관리하고 있음.
  - 무료와 다름없는 Python Software Foundation License.
- 유니코드
  - 파이썬에서의 문자열들은 모두 유니코드입니다.
  - 우리가 한글, 한자 등을 표현하려고 특별한 노력을 할 필요가 없습니다.
- 동적타이핑
  - 런타임 시에 타입 체크를 하는 동적타이핑을 지원하며,
  - 메모리 관리 자동으로 합니다.

파이썬의 특징 3

객체 지향적(?)

동적(?) 타이핑 언어

파이썬의 특징 3

객체 지향적 언어

: 실행 순서가 아닌 단위 모듈(객체) 중심으로 프로그램을 작성
하나의 객체는 어떤 목적을 달성하기 위한 행동(**method**)와
데이터(**attribute**)를 가지고 있음

동적 타이핑 언어

: 프로그램이 실행하는 시점 프로그램이 가지고 있는 데이터에
대한 타입을 검사함

원말인지 모르면 넘어가자. 괜찮다.

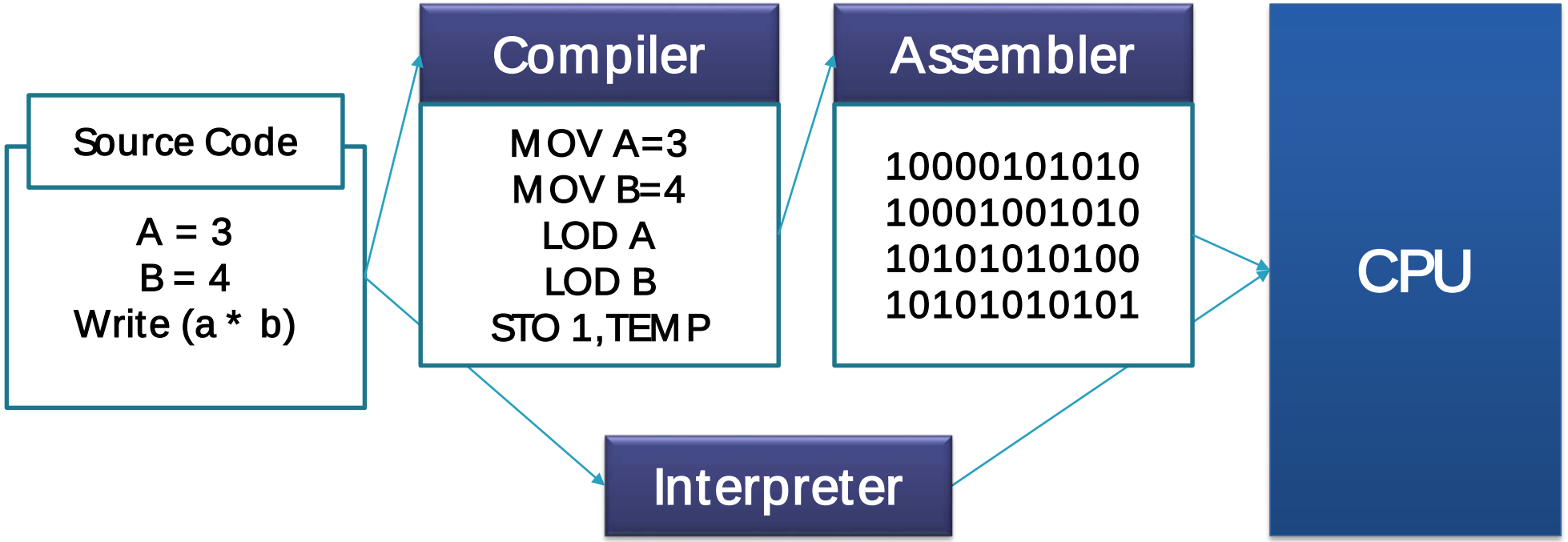
컴파일러언어 vs 인터프리터 언어

컴파일러		인터프리터
소스코드를 기계어로 먼저 번역하고 해당 플랫폼에 최적화되어 프로그램을 실행	**작동방식**	별도의 번역과정 없이 소스코드를 한줄 한줄 분석하여 컴퓨터가 처리할 수 있도록 함
실행속도가 빠름 한번의 많은 기억장소 필요	**장점** **단점**	간단히 작성, 메모리가 적게 필요 실행속도가 느림
C, 자바, C++, C#	**주요 언어**	파이썬, 스칼라

※ 최근 인터프리터 언어도 실행속도를 올리기 위해 사용자가 모르게 컴파일한 다음 번역없이 실행

소프트웨어(프로그램)의 동작과정

사람이 알 수 있는 고급언어를 기계가 알수 있는 저급언어로 변환



※ 파이썬은 처음에 컴파일러 언어인 C로 작성되었다.
당연히 실행시 Assembler와 같은 기계어 변환 과정을 거친다

Why Python?

왜 파이썬인가 ?

인간 지향적인 간단한 문법

JAVA vs Python

JAVA

변수선언 여부

```
int    myCounter = 0;
String myString = String.valueOf(myCounter);
if (myString.equals("0")) ...
```

PYTHON

```
myCounter = 0
myString = str(myCounter)
if myString == "0": ...
```

JAVA

```
// print the integers from 1 to 9
for (int i = 1; i < 10; i++)
{
    System.out.println(i);
}
```

Loop 선언 형태

Brace 여부

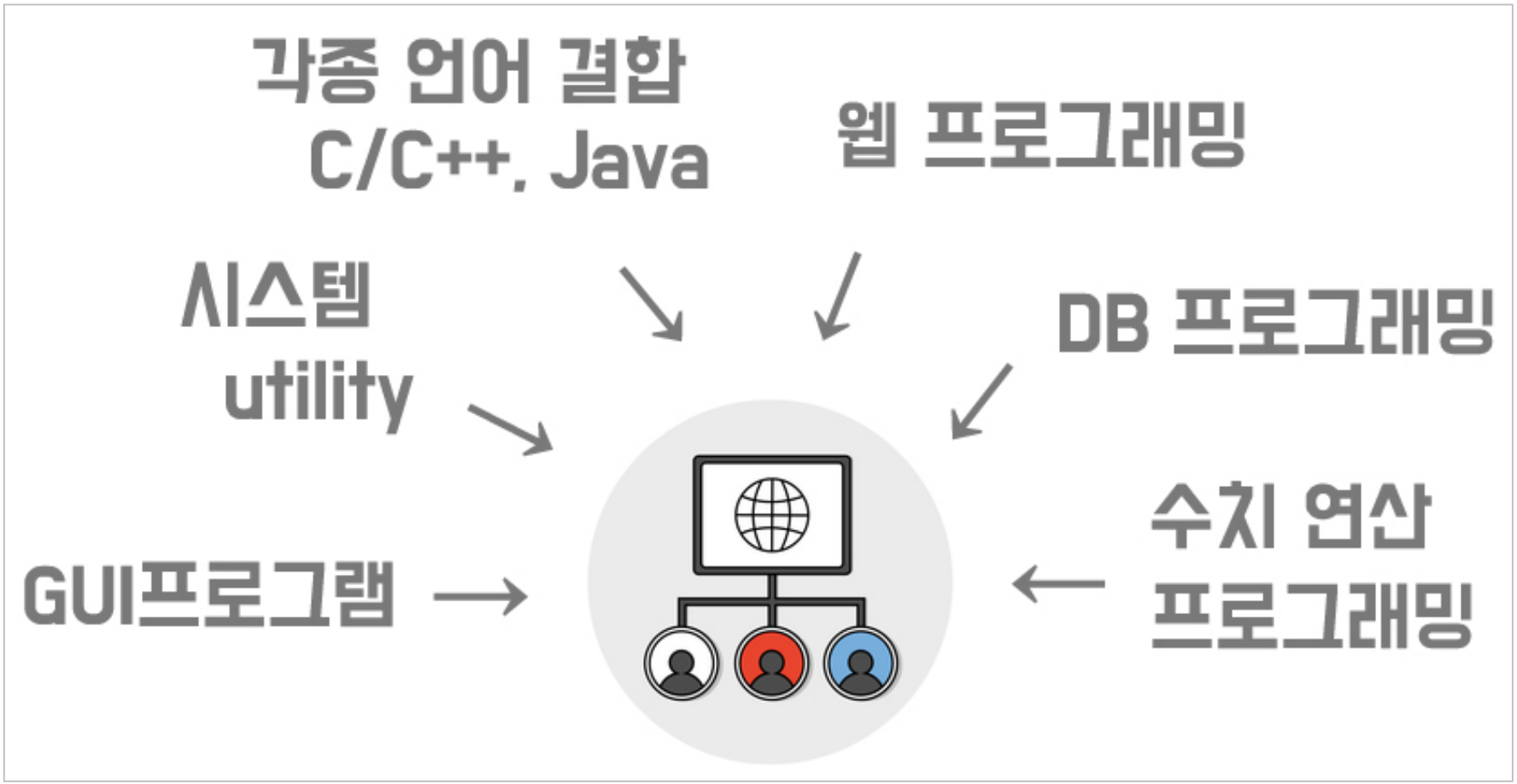
PYTHON

```
 print the integers from 1 to 9
for i in range(1,10):
    print i
```

프로그래밍을 몰라도 해석이 가능한 문법의 언어

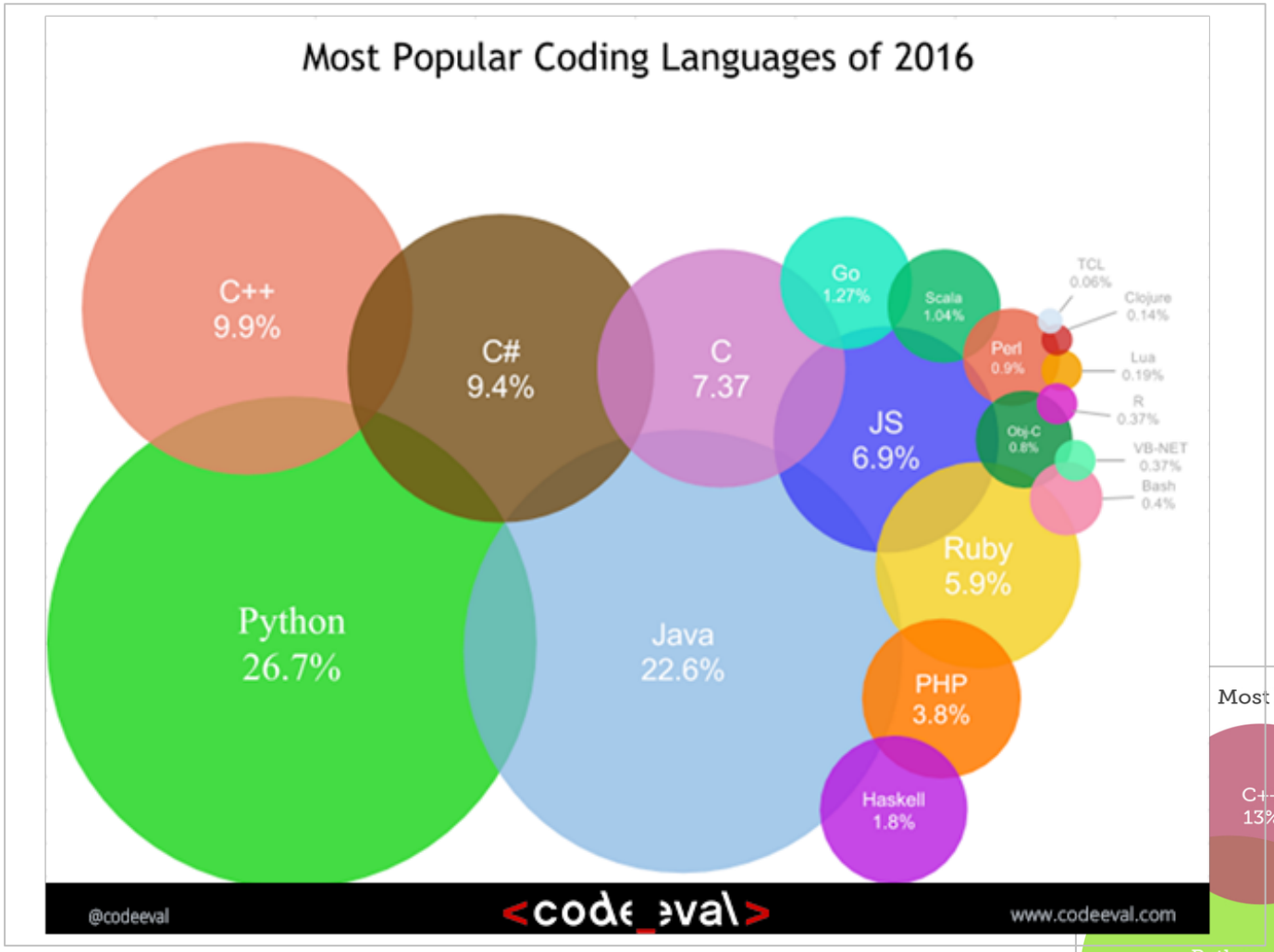
왜 파이썬인가 ?

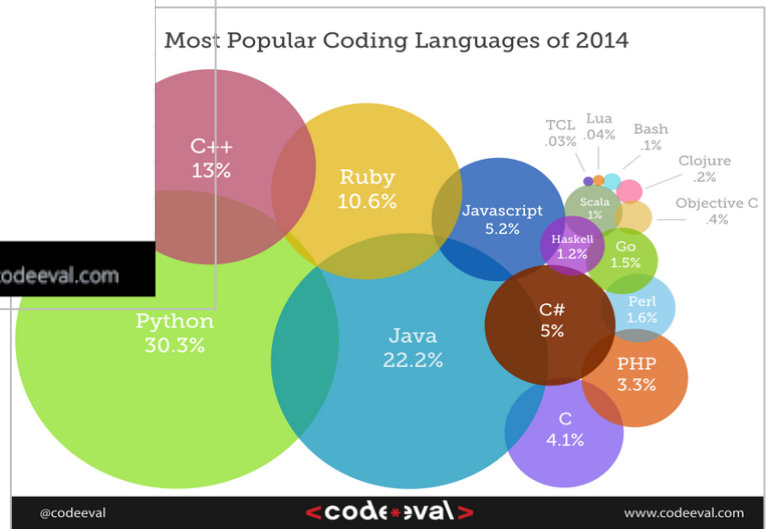
다양한 라이브러리,넓은 활용범위



http://blog.naver.com/hbi2hyunjung/220611246872

가장 인기있고 대중화된 Python





파이썬의 종류

- Cpython : C로 짜여진 파이썬 . 우리가 쓰는 파이썬 인터프리터가 바로 이 CPython 이다 . 바꿔 말하면 파이썬은 C로 짜여졌다 라고 할 수도
- Cython : □□□□ □□□□ □□□(superset)□□. □□ □□ □□□□ □□□□ □□□ □□□□ □□□□ □□□ 을 하면 되지
- Jython : □□□□□ □□□□ □□□
- IronPython : □□□□ □□□□ □□□
- PyPy : □□□□□ □□□ □□□ □□□□□

파이썬 프로젝트

□□□ □□□□ 400□□ □□□ □□□□
http://pythonhackers.com/open-source/

- □□ □□□ □□□□ □□□□ □□□□□□ □□□□□,
  □□□ □□□ □□ □□ □□□□□.
  - BitTorrent, MoinMoin, Scons, Trac, Yum
  - CherryPy, Django
  - GIMP, Maya, Paint Shop Pro
  - Youtube.com, Google Groups, Google Maps, Gmail

파이썬 2.x 와 3.x 차이

print가 함수로 변경.

2.x style :
```
>>> print "welcome to", "python3k“
welcome to python3k
```
3.0 style :
```
>>> print("welcome to", "python3k")
welcome to python3k
```

long 자료형이 없어지고 int 로 통일 .

- 2.x style :
  ```
  >>> type(2**31)
  <type 'long'>
  >>> sys.maxint
  2147483647
  ```
- 3.0 style :
  ```
  >>> type(2**31)
  <class 'int'>
  >>> type(2**40)
  <class 'int'>
  ```

- ‘int / int’의 결과가 float 으로 처리 .

  2.x style :
  ```
  >>> 3/2
  1
  ```
  3.0 style :
  ```
  >>> 3/2
  1.5
  >>> type(2/2)
  <class 'float'>
  ```
- String, Unicode □□ □□.

  2.x style :
  ```
  >>> type('가')
  <type 'str'>
  >>> type(u'가')
  <type 'unicode'>
  ```
  3.0 style :
  ```
  >>> type('가')
  <class 'str'>
  >>> type('가'.encode('cp949'))
  <class 'bytes'>
  ```

소스코드 인코딩

- □□□□□□ # □□□ □□□□ □□□□□.
- □□□ □□□ □□,
  □□□□ □□□□ □□□ □□,
  □□□□ □□□□ □□□□ □□□ □□□□□.
  - # coding: latin-1
  - # -*- coding: utf-8 -*-

기타 파이썬 문법

- 한 라인에 여러 구문이 올 경우에는 세미콜론 (;)을 □ □ □ □ □ □ □ .
  - >>> a = 1; b = 2
- 들여쓰기 (indentation)이 중요하지만 , 문장이 아직 안 끝난 경우에는 들여쓰기를 안 해도 문법오류가 나지 □ □ □ □ .
  - >>> a = (1 +
    [TAB]2 +
    3 +
    [TAB][S][S]4)
    >>> a
    10

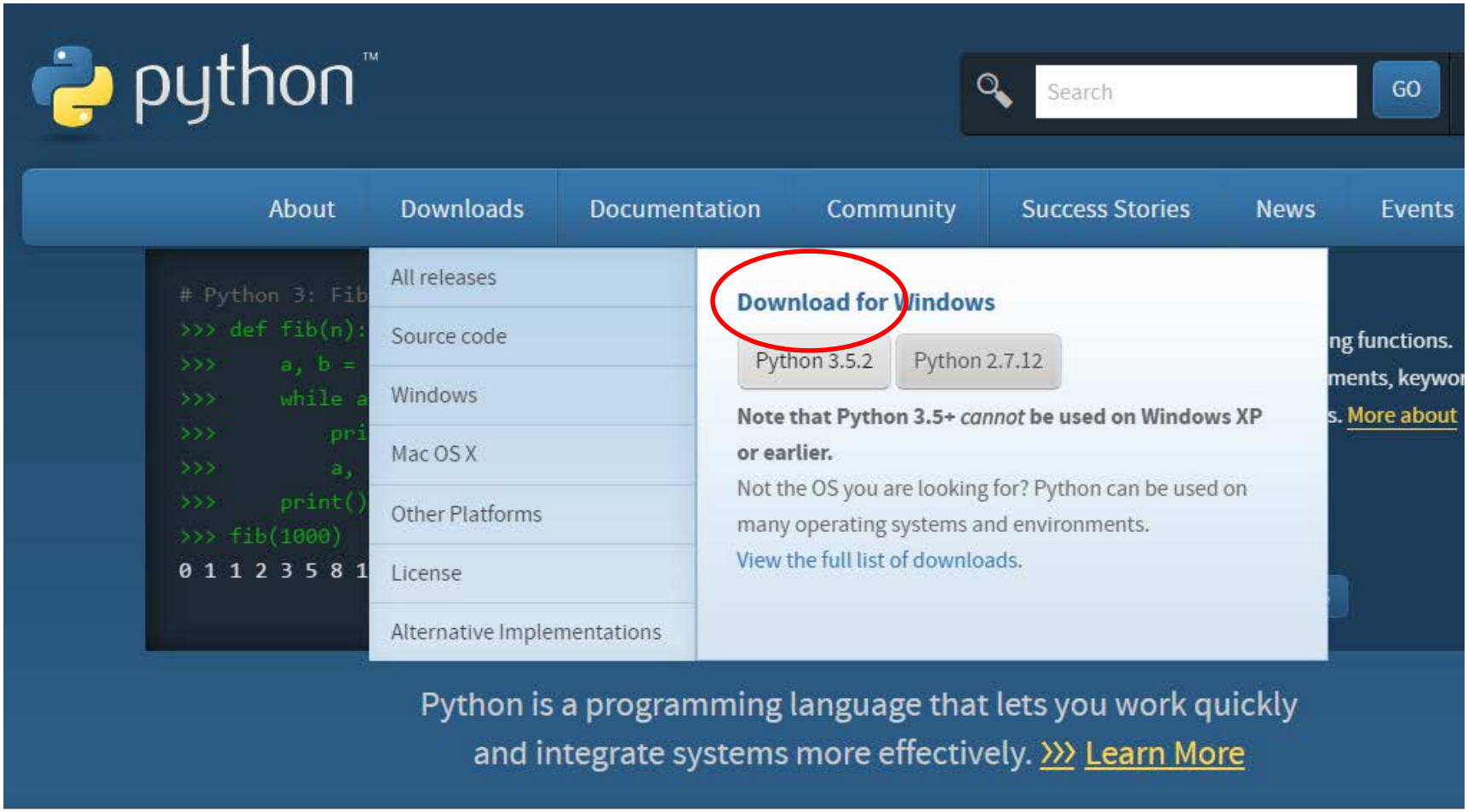
파이썬 버전 2.x를 3.ㅌ 로 변환하기

- 파이썬 2.x 과 파이썬 3 과는 호환이 되지 않는 다는 것은 큰 단점입니다만 , 2.x 에서 3 로 변경하는 툴을 제공합니다 .
  - C:\ Python30\ Tools\ Scripts>2to3.py -w test.py

파이썬 시작하기

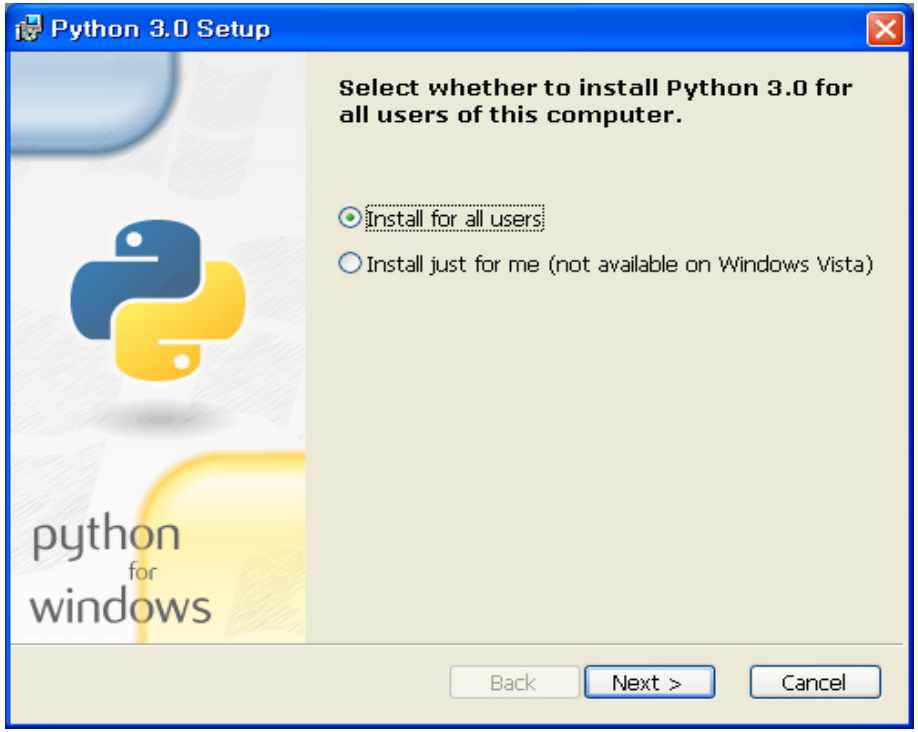
Python 설치

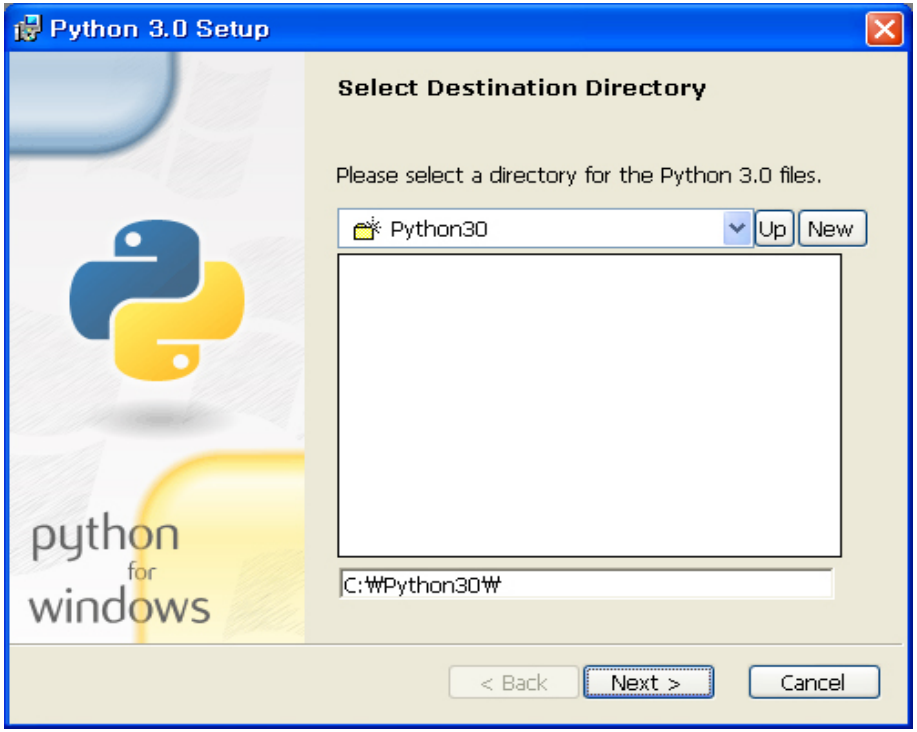
https://www.python.org 에서 다운로드 설치
설치할 때 **Path** 자동으로 등록되도록 체크



설치 및 개발환경

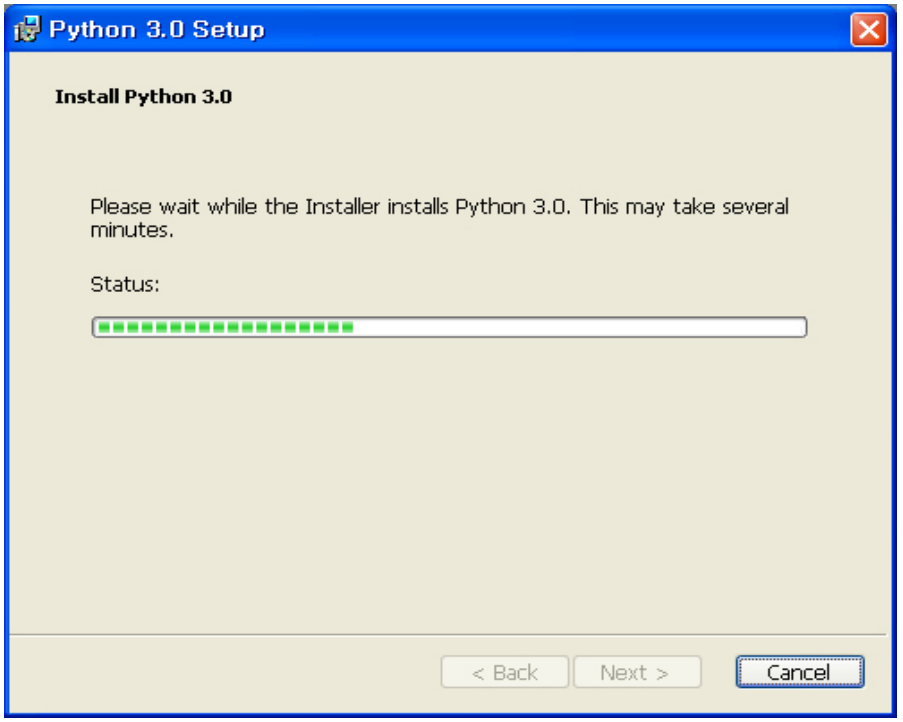
- http://python.org/download





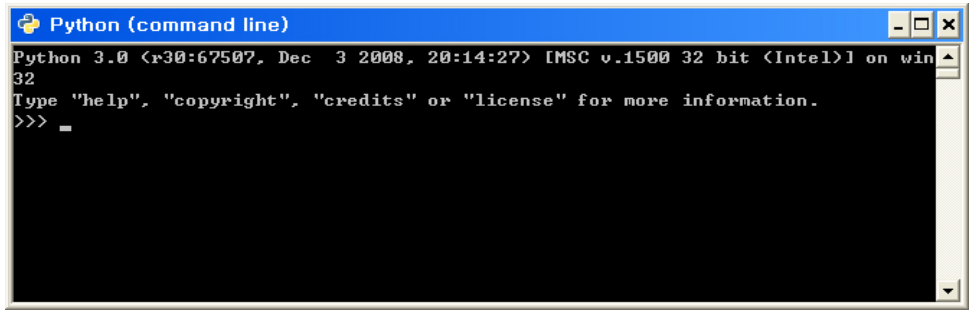
설치 및 개발환경

Python 3.0 Setup
Customize Python 3.0
Select the way you want features to be installed. Click on the icons in the tree below to change the way features will be installed.
Python
Register Extensions
Tcl/Tk
Documentation
Utility Scripts
Test suite
Python Interpreter and Libraries
This feature requires 19MB on your hard drive. It has 5 of 5 subfeatures selected. The subfeatures require 25MB on your hard drive.
python for windows
Disk Usage
Advanced
< Back
Next >
Cancel

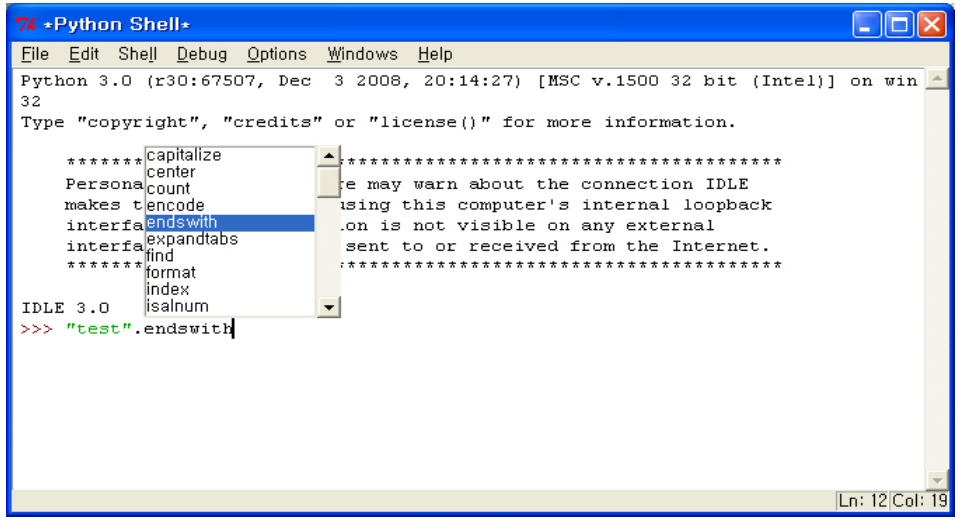

Python 3.0 Setup
Install Python 3.0
Please wait while the Installer installs Python 3.0. This may take several minutes.
Status:
< Back
Next >
Cancel

설치 및 개발환경

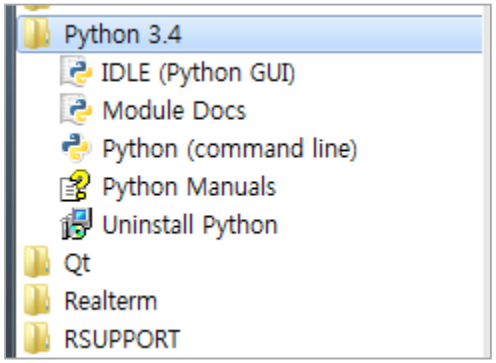
- Python command line

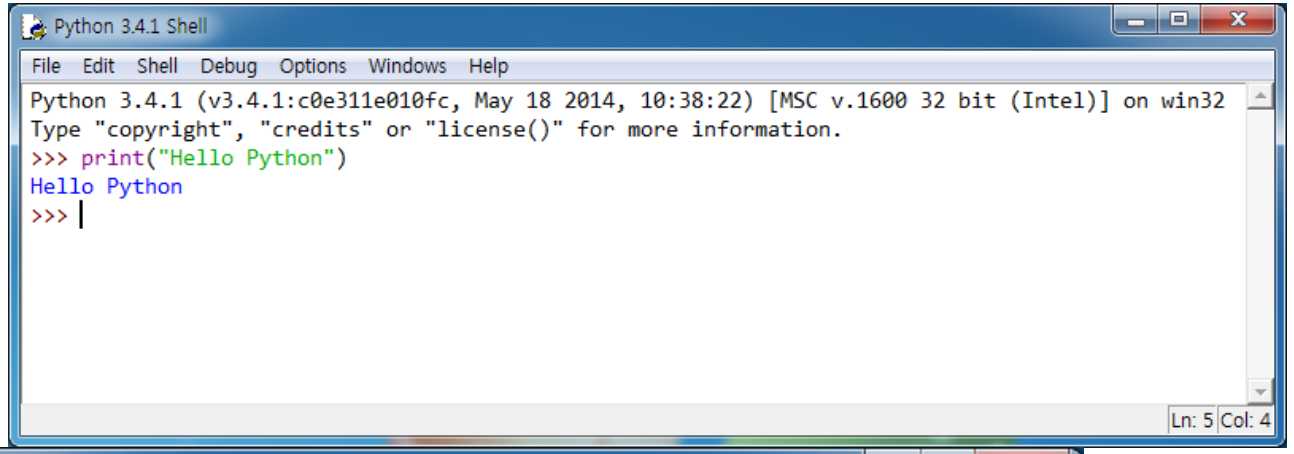


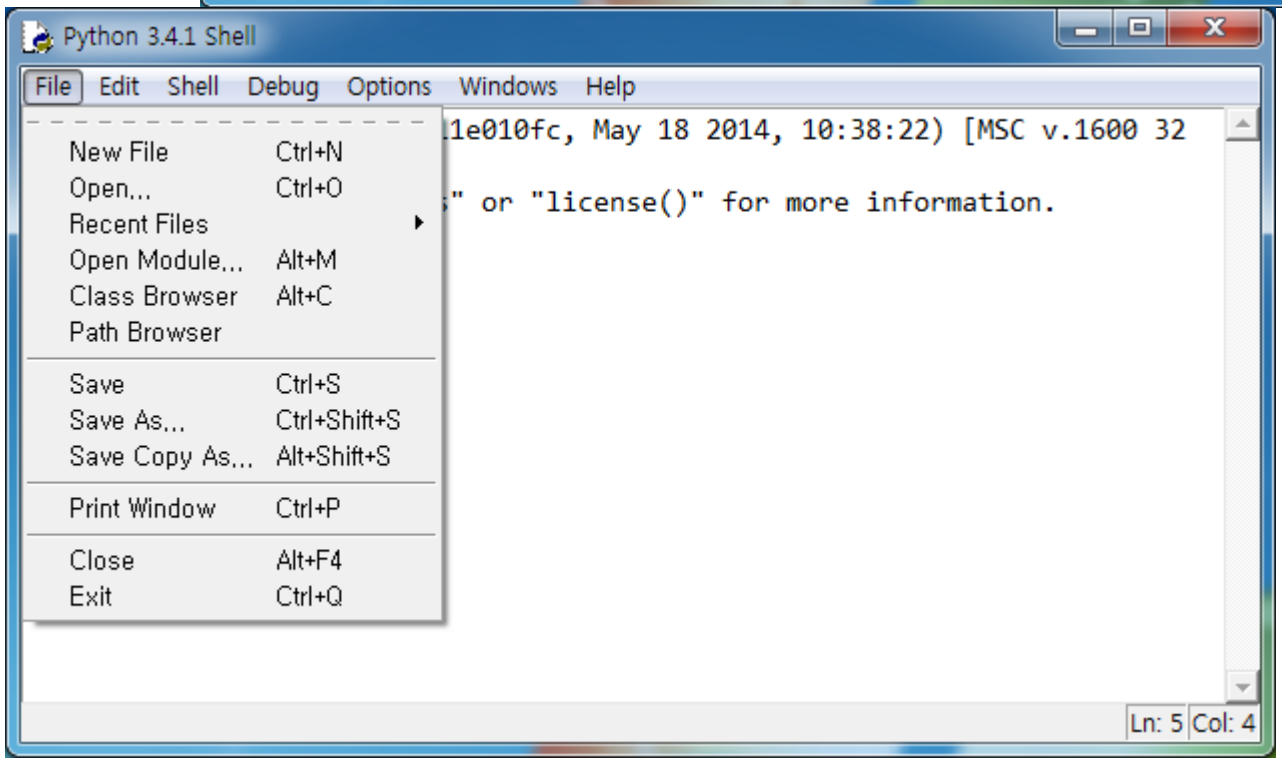
IDLE



Python IDLE 실행



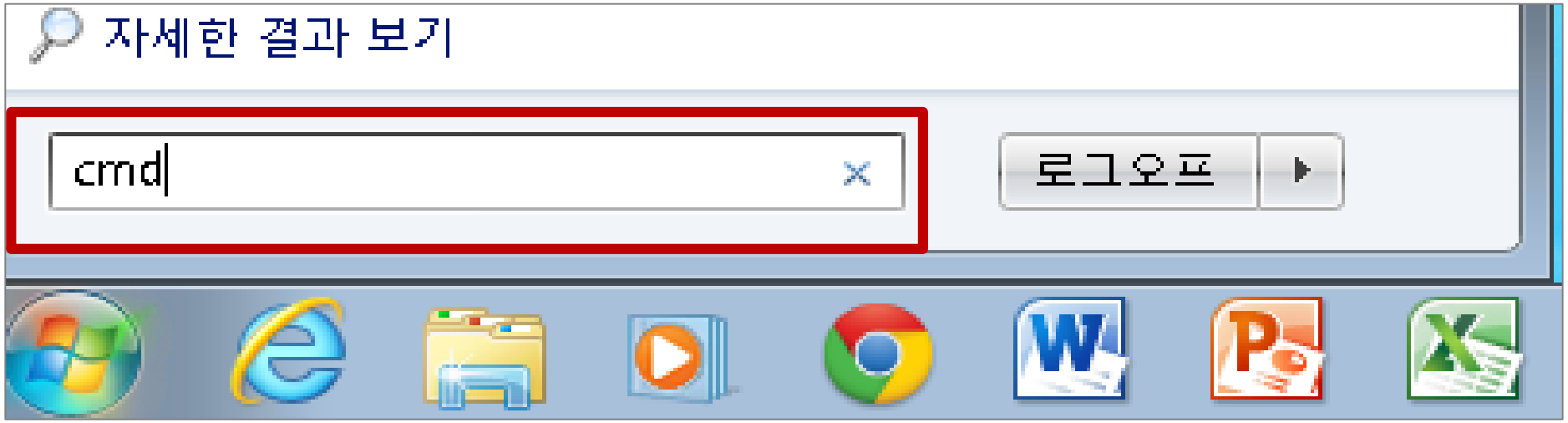




Python 실행

Windows에서도 CLI 환경에서 개발

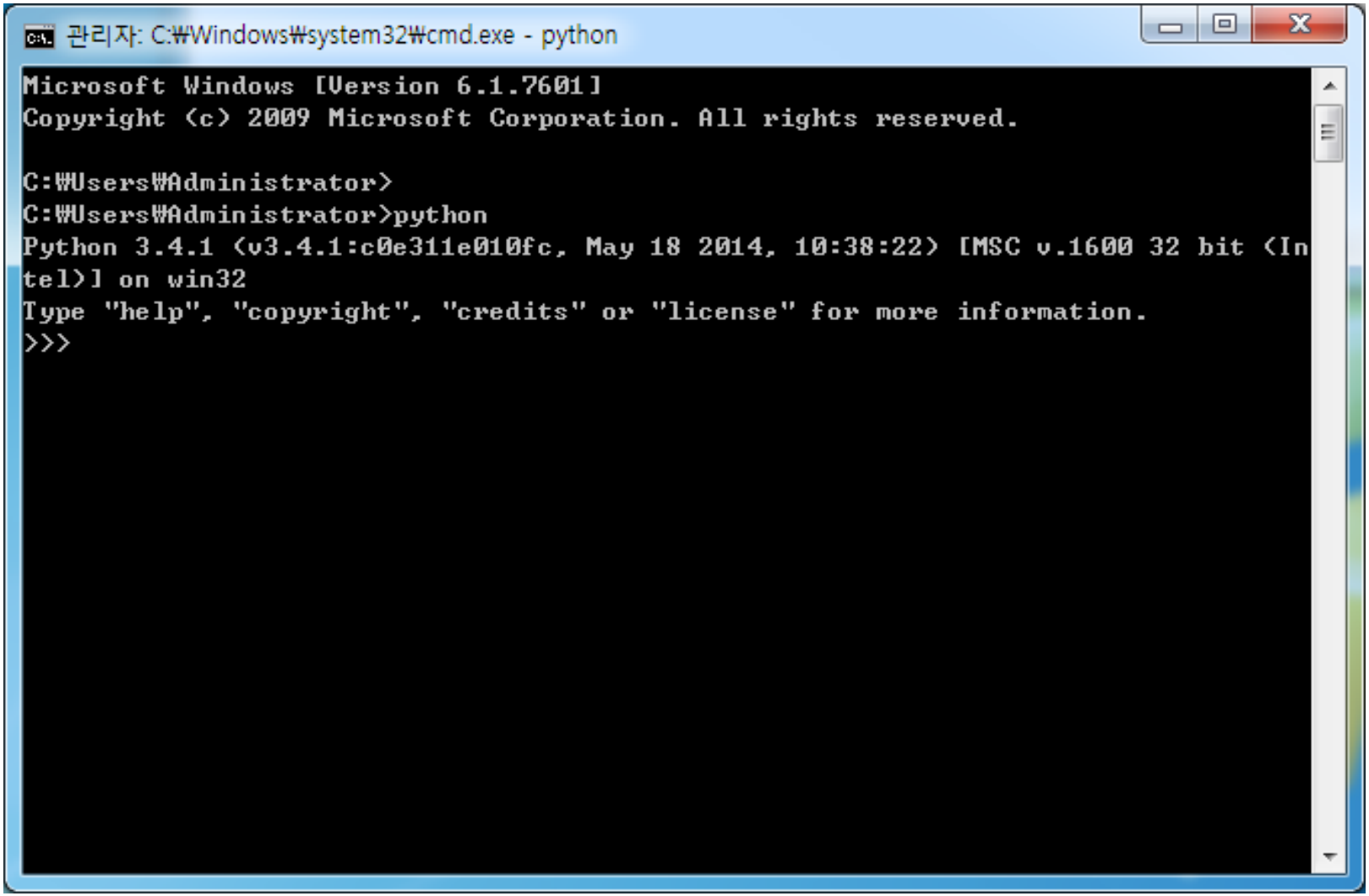
실행창에 cmd 입력 (하단명령어 입력)



Command Line Interface (CLI)

Graphic User Interface (GUI) 와 달리 **Text**를 사용하여 컴퓨터에 명령을 입력하는 인터페이스 체계

CLI 환경에서 “python” 파이썬 실행



CLI 환경

GUI환경의 명령들도 CLI를 통해 호출 가능

예) 작업관리자 호출 : **taskmgr**

PATH가 설정되어야 어디서나 작동 가능

노트패드 실행: **notepad**
익스플로러 실행: **"c:\ Program Files\ Internet Explorer\ iexplore"**

실행시 실행인자(argument) 값 사용 가능

종료: **shutdown /s**, 재시작: **shutdown /r**
같은 명령어지만 인자값(옵션 또는 인수)에 따라 기능이 다름

여러 명령어를 동시에 또는 자동화하여 실행 가능

예) **autoexec.bat**

CLI 상에서 주요 명령어

명령어	수행내용
dir	해당 폴더의 파일리스트 보여주기
mkdir python	"python" 이라는 이름으로 새폴더 만들기
copy abc.txt python\	"abc.txt" 파일을 "python" 폴더로 복사
cd python	"python" 폴더로 이동
move abc.txt ..\	"abc.txt" 파일을 부모폴더로 이동 (복사후 삭제)
cd..	부모 폴더로 이동
del abc.txt	"abc.txt" 파일 삭제
rmdir python	"python" 폴더 삭제
echo	Echo “Hello World” > hello.txt

코딩 컨벤션

코딩 컨벤션

- 프로그래밍은 공동작업이 많기 때문에 프로그램을 작성하는 가이드라인 언어마다 정해져 있음
- 자신만의 방식으로만 프로그램을 작성할 경우, 다른 사용자의 이해가 떨어짐
- 컴퓨터가 이해할 수 있는 코드는 어느 바보나 다 짤 수 있다. 좋은 프로그래머는 사람이 이해할 수 있는 코드를 짠다. - 마틴 파울러 –
- 이를 위해 기본적인 코딩 작성법을 코딩 컨벤션이라 함

Python 코딩 컨벤션

- 들여쓰기는 공백 4칸을 권장
- 한 줄은 최대 79자까지
- 클래스 내의 메소드 정의는 1줄씩 띄어쓰기
- 불필요한 공백은 피함 (([])와 (())안, (,), (:)과 (;) 앞
- = 연산자는 붙여씀
- 주석은 항상 갱신, 불필요한 주석은 삭제
- 소문자 L, 대문자 O, 대문자 I는 변수명으로 사용하지말것 (가독성 문제)
- 함수명은 소문자로 구성, 필요하면 밑줄로 나눔

모든 프로그래밍 언어
C, C++, Java, Python, R, 자바스크립트, C#…

변수

여러 가지 다른 값들을 나타내는 일종의 상징

byte char int long float double

연산

산술연산자(+, -, /, * , ^ , %)
비교연산자(=, >, <, >=, <=, ==, !=)
비트연산자(&, |, ^ , ~, <<, >>)
논리연산자(&&, ||, !) 증감 연산자(a++, a--, ++a, --a)

제어

switch 문, while 반복문, if 문

함수

영어 function에서 온 말인데 이 뜻은 "기능"을 뜻한다.
즉, 하나의 기능을 구현하는 부분을 따로 떼어 구현하는 것
int sum(a, b);

Python 변수

변수(Variable)

임시로 혹은 영구적으로 값을 기억해두기 위한 저장소
값을 담아줄 뿐만 아니라 어떠한 값에 이름표를 붙이는 것
변수가 담고 있는 값은 항상 고정되어 있는게 아니라 변할수 있다

- 문자, 숫자, 밑줄(_)로 구성됩니다.
  숫자는 처음에 나올 수 없습니다.
- 대소문자를 구분합니다.
- 예약어 사용 불가.

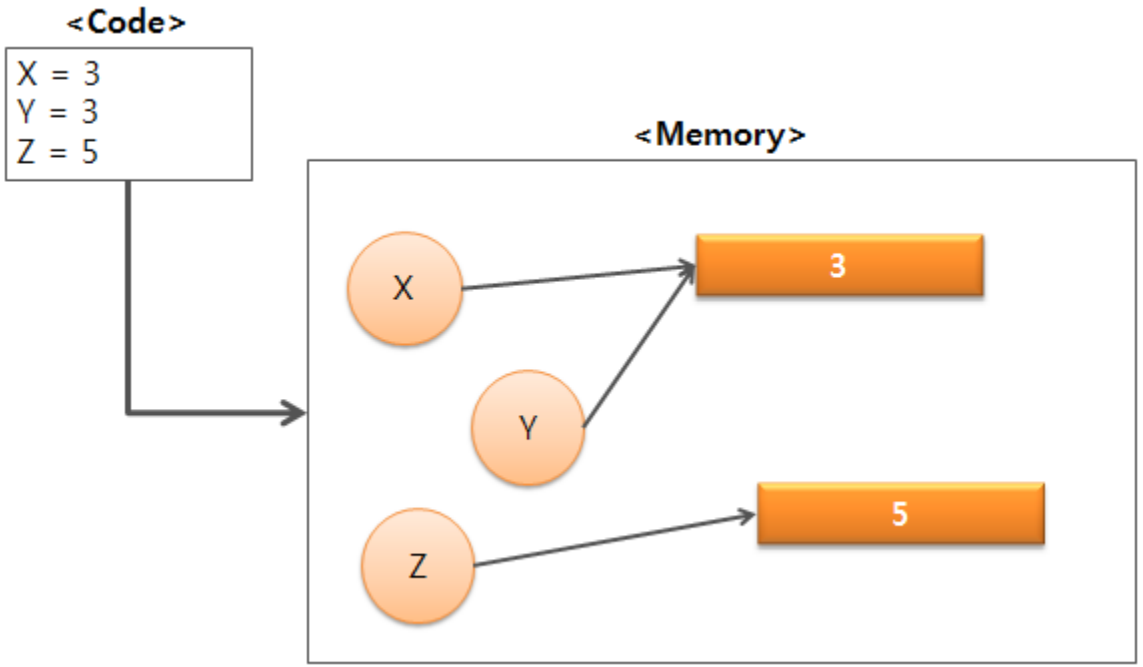
```
4pple = 10  # 숫자가 가장 처음에 등장했다 . 이 경우에는 구문 에러가 발생한다 .
admin = 50 # □□□□□ admin□□ □□□ □□□□□.
Apple = 60  # □□□□□ Apple□□ □□□ □□□□□.
apple = 70  # □□□□□ □□□□ □□□ □□ □□ □□□ Apple□□ □□□ □□□□.
print(admin, Apple, apple) # 세 변수의 값을 모두 출력한다 . 출력된 결과는 50 60 70 이다 .
```

- 예약어, KeyWorld 사용 불가.

False, None, True, and, as, assert, 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

파이썬의 모든 자료형은 객체

```
>>> a = 1
>>> b = "python"
>>> c = [1,2,3]
```




```
>>> X = 3
>>> Y = 3
>>> Z = 5
>>> print(id(X), id(Y), id(3), id(Z), id(5))
```

```
507098848 507098848 507098848 507098912 507098912
```

수치자료형(Numeric Data Type)

정수(Integer)

```
>>> A = 5
>>> B = 10
>>> print(A, B)
```

```
>>> print(0b10, 0o10, 0x10)
2 8 16
```

```
>>> bin(202020)
'0b110001010100100100'
>>> oct(20114)
'0o47222'
>>> hex(401000)
'0x61e68'
```

실수(Float)

```
>>> A = 3.14
>>> B = 5.1402020319
>>> print(A, B)
3.14 5.1402020319
```

```
>>> 0.23193202032E8
23193202.032
>>> 1412232.22E-10
0.000141223222
```

복소수(Complex)

```
>>> x = 7 - 3j
>>> type(x)
<class 'complex'>
>>> x.imag
-3.0
>>> x.real
7.0
>>> x.conjugate()
(7+3j)
```

정수형 상수 (int)

- 10진수, 16진수, 8진수, 2진수 변환 가능

```
a= 32                                  # a에 정수 32할당
print a, hex(a), oct(a), bin(a)        # 10진수, 16진수, 8진수, 이진수 값 출력
32 0x20 040 0b100000
```

- 정수간의 연산은 정수, 정수-실수 계산은 실수로 연산

```
>>> print 10 + 10
20
>>> print 10 + 10.0
20.0
```

실수형 상수 (float)

```
>>> a = 1.2                                  # a에 정수 1.2 할당
>>> b = 2.0                                  # b에 정수 2.0 할당
>>> print a. is_integer(), b.is_integer()        # a와 b가 정수형인지 확인
False True                                        # a는 정수형 불가, b는 가능
>>> import math              # math 라이브러리 호출
>>>  print round(a), math.ceil(a), math.floor(a)
1.0 2.0 1.0                                  # a의 반올림, 올림, 내림 계산 수행
```

복소수형 상수(complex)

- 실수부와 허수부로 분리, 허수부는 j를 붙임

```
>>> a = 1 + 1j                          # a에 1 + i 할당
>>> b = 1 - 1j                          # b에 1 - i 할당
>>> print (a+b), (a-b), (a* b)          # a+b , a-b, a* b 출력
(2+0j) 2j (2+0j)
>>> print a.real, a.imag, a.conjugate() # a의 실수, 허수, 켤레복소수 출력
1.0 1.0 1-1j
>>> a = 1
>>> b = -1
>>> complex (a,b)                       # a는 실수, b는 허수로 복소수 생성
(1-1j)
```

모듈 math 를 활용한 연산

- 복잡한 수학 연산을 위해 기존 만들어진 모듈을 사용

```
import math                    # math 모듈 호출
print math.pow(2,10)           # 지수연산
print math.sqrt(121)           # 제곱근 연산
print math.log(12)             # 자연로그 연산
Print math.log10(100)          # 상용로그 연산
```

```
1024.0
11.0
2.48490664979
2.0
```

문자열
(string)

문자열(string)

- 시퀀스 자료형으로 문자형 **data**를 메모리에 저장
- 영문자 한글자는 **1byte**의 메모리공간을 사용

```
import sys                          # sys 모듈을 호출
print sys.getsizeof("a"), sys.getsizeof("ab"), sys.getsizeof("abc")
25 26 27                            # “a”, “ab”, “abc”의 각 메모리 사이즈 출력
```

- **string**은 **1byte** 크기로 한글자씩 메모리 공간이 할당됨

```
a = “abcde”
```

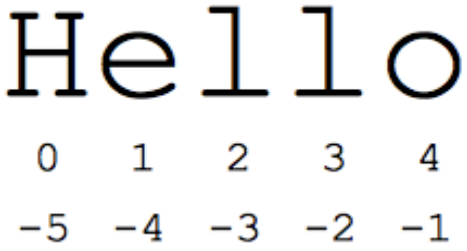
a	0100 1001
b	0100 1010
c	0100 1011
d	0100 1100
e	0100 1101

인덱싱(Indexing)

- 문자열의 각 문자는 개별 주소(offset)를 가짐,
  이 주소를 사용해 할당된 값을 가져오는 것이 인덱싱

```
a = "abcde“
print a[0], a[4]          # a 번수의 0번째, 4번째 주소에 있는 값
print a[-1], a[-5]        # a 변수의 오른쪽에서 1번째, 5번째 주소에 있는 값
```

- 각 문자의 오프셋은
  왼쪽에선 0부터
  오른쪽에선 -1부터 시작함



Source: http://goo.gl/XjPSpP

슬라이싱(slicing)

- 문자열의 주소값을 기반으로 문자열의 부분값을 반환

```
a = "Gachon Unversity"
print a[0:6], " AND ", a[-9:] # a 번수의 0부터 5까지, -9부터 끝까지
print a[:]              # a변수의 처음부터 끝까지
print a[-50:50]     # 범위를 넘어갈 경우 자동으로 최대 범위를 지정
print a[::2], " AND ", a[::-1]            # 2칸 단위로, 역으로 슬라이싱
```

문자열 연산 및 포함여부 검사

- 덧셈과 뺄셈 연산 가능, in 명령으로 포함여부 체크

```
a = "Gachon"
b = "University“
a + " " + b                    # 덧셈으로 a와 b 변수 연결하기
 ' Gachon University '
A * 2 + ″  ″ + b * 2
 ' GachonGachon UniversityUniversity‘      # 곱하기로 반복 연산 가능
if 'U' in a: print a # ‘U‘가 a에 포함되었는지 확인
else: print b
```

다양한 문자열 표현

문자열 선언은 큰따옴표(“”)나 작은 따옴표 (‘)를 활용

It’s OK 이라는 문자열은 어떻게 표현할까?

① a = ‘It\ ’ ok.’

\ ’는 문자열 구분자가 아닌 출력문자로 처리

② a = “It’s ok.” #

큰따옴표로 문자열 선언후 작은 따옴표는 출력문자로 사용

두줄 이상은 어떻게 저장할까?

① \ n # \ n은 줄바꿈을 의미하는 특수 문자

② 큰따옴표 또는 작은 따옴표 세번연속 사용

예) a= “”” It’Ok
I’m Happy.
See you. “””

특수문자(escape character)

문자열을 표시할때 백슬래시 \ 를 사용하여 키보드로 표시하기 어려운 문자들을 표현함

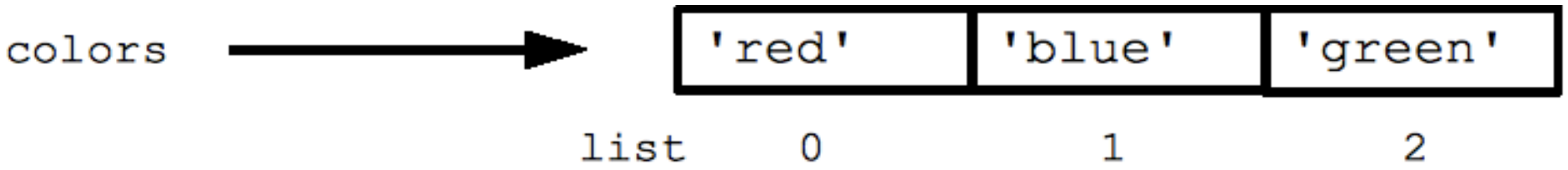
문자	설명	문자	설명
\ [Enter]	□□ □□ □□□□ □□	\ n	□□□□
\ \	\ 문자 자체	\ t	TAB 키
\ `	` 문자	\ e	ESC 키
\ "	" □□		
\ b	백 스페이스		

리스트 자료형
(list)

리스트 (list)

- 문자형과 같은 시퀀스 자료형, 여러 데이터들의 집합
  리스트명 = [요소1, 요소2, 요소3, ...]

```
colors = ['red', 'blue', 'green']
 print colors[0]    ## red
 print colors[2]    ## green
 print len(colors)  ## 3
```



Source: http://goo.gl/q4VvB1, http://goo.gl/JMbHm0

리스트의 연산

- 인덱싱, 슬라이싱, 연산등을 동일하게 사용

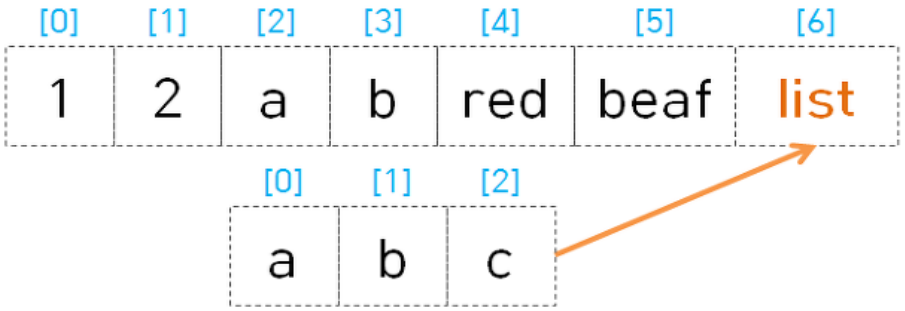
```
color = ['red', 'blue', 'green']
color2 = ['orange', 'black', 'white']
print color + color2          # 두 리스트 합치기
len(color) # 리스트 길이
color[0] = 'yellow' # 0번째 리스트의 값을 변경
print color *  2      # color 리스트 2회 반복
'blue' in color2      # 문자열 'blue'가 color2 존재 여부 반환
total_color = color + color2
for each_color in total_color # total_color에 입력된 문자열 하나씩 순회
    print each_color
```

리스트 요소 추가와 삭제

```
color.append("white")       # 리스트에 “white” 추가
color.extend(["black","purple"]) # 리스트에 새로운 리스트 추가
color.insert(0,"orange")    # 0번째 주소에 “orange” 추가
print color
['orange', 'yellow', 'blue', 'green', 'white', 'black', 'purple']
color.remove("white")       # 리스트에 “white” 삭제
del color[0]                # 0번째 주소 리스트 객체 삭제
print color
['yellow', 'blue', 'green', 'black', 'purple']
```

리스트의 특징

```
a = [“color”, 1, 0.2]          # 다양한 데이터 타입 입력 가능
color = ['yellow', 'blue', 'green', 'black', 'purple']
a[0] = color                   # 리스트 안에 리스트도 입력 가능
print a
[['yellow', 'blue', 'green', 'black', 'purple'], 1, 0.20000000000000001]
```

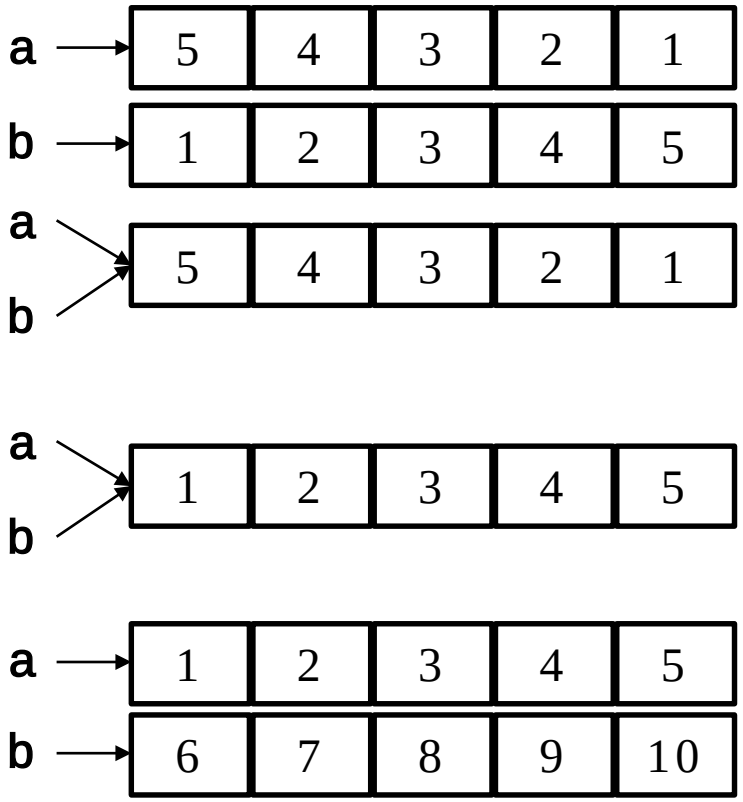


□□ □□□□ □□□ □□

Source: http://goo.gl/FApwnw

리스트 함수들

```
a = [5,3,4,1,2]
a.index(2)              # 정수 2가 있는 주소값
a.count(5)              # 리스트내에 5의 개수
a.sort()                # 리스트 a 정렬
a.reverse()             # 리스트 a 역정렬
b = sorted(a)           # 정렬된 a를 b 변수에 할당
```



리스트의 메모리 저장 방식

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

b = [1, 2, 3, 4, 5]

b = a

print b # [5, 4, 3, 2, 1]

a.sort()

print b # [1, 2, 3, 4, 5]

b = [6,7,8,9,10]

print a, b

[1, 2, 3, 4, 5] [6, 7, 8, 9, 10]
```

“=“ 의 의미는 같다가 아닌 메모리 주소에 해당 값을 할당 (연결) 한다는 의미

패킹과 언패킹(packing and unpacking)

- 패킹: 한 변수에 여러 개의 데이터를 넣는 것
- 언패킹: 한 변수의 데이터를 각각의 변수로 반환하는 것

```
t = [1, 2, 3]              #  1,2,3을 변수 t에 패킹
a , b , c = t              #  t에 있는 값 1, 2, 3 을 변수 a, b, c에 언패킹
Print t, a, b, c           # [1, 2, 3] 1 2 3
```

이차원 리스트

- 리스트 안에 리스트를 만들어 행렬(Matrix) 생성 가능

```
kor_score = [49,79,20,100,80]
math_score = [43,59,85,30, 90]
eng_score = [49,79,48,60,100]
midterm_score = [kor_score, math_score, eng_score]
print midterm_score[0][2]
```

	A	B	C	D	E
국어점수	49	79	20	100	80
수학점수	43	59	85	30	90
영어점수	49	79	48	60	100

이차원 리스트 순환

- 이전 페이지 코드를 활용하여 사람별 평균을 구하라

```
student_score = [0,0,0,0,0]
i = 0
for subject in midterm_score:
    for score in subject:
        student_score[i] += score       # 각 학생마다 개별로 교과 점수를 저장
        i += 1                          # 학생 index 구분
    i = 0                               # 과목이 바뀔때 학생 인덱스 초기화
Else:
    a, b, c, d, e = student_score     # 학생별 점수를 unpacking
    student_average = [a/3,b/3,c/3,d/3,e/3] #
    print student_average
```

성적 입력 프로그램 연습

동적으로 학생수와 교과목 개수를 받아서
학생별 성적 평균과 교과목 성적 평균을
계산하는 프로그램을 구하라

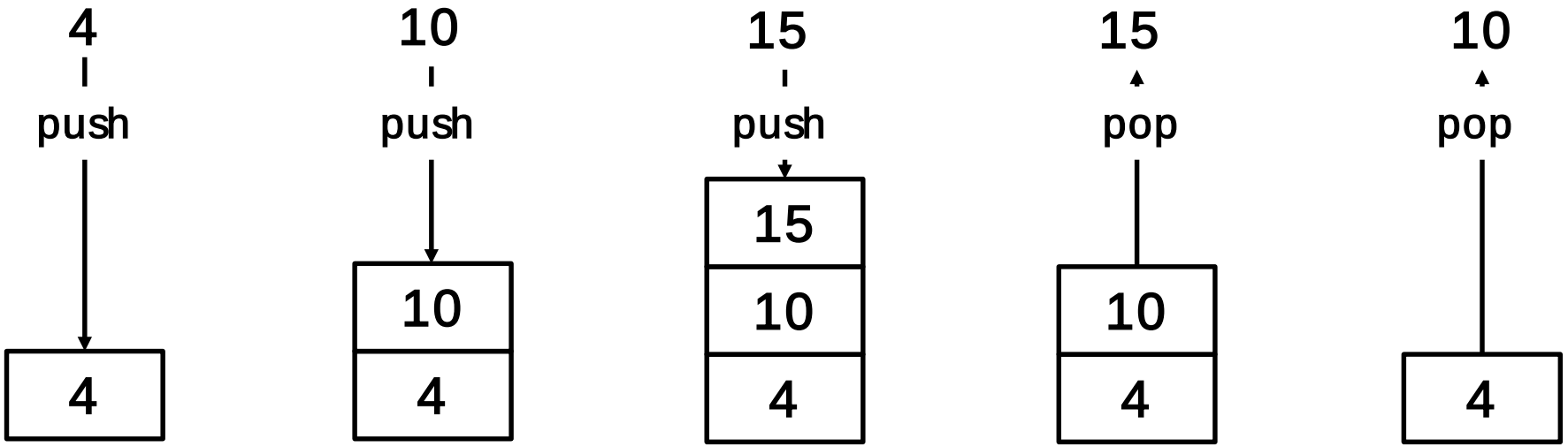
데이터 구조
(stack, queue, tuple, set, dictionary)

데이터 구조(Data Structure)

- 메모리상에 데이터를 효율적으로 관리하는 방법
- 검색, 저장 등의 작업에서 효율을 고려하여
  메모리 사용량과 실행시간 등을 최소화 함
- 파이썬 에서는 리스트, 튜플, 집합(Set),
  사전(dictionary) 등의 기본 데이터 구조를 제공함

스택(Stack)

- 나중에 넣은 데이터를 먼저 반환하도록 설계된 메모리 구조로 **Last In First Out (LIFO)**로 구현됨
- **Data**의 입력을 **Push**, 출력을 **Pop**이라고 함



큐(Queue)

- 먼저 넣은 데이터를 먼저 반환하도록 설계된 메모리 구조로 **First In First Out**으로 구현됨
- **Stack**과 반대되는 개념

PUT(A)	PUT(B)	PUT(C)	GET()	PUT(D)	GET()
				D	D
		C	C	C	C
	B	B	B	B	
A	A	A			

Source: http://goo.gl/2SioRS

파이썬에서 큐(Queue) 구현

- 파이썬은 리스트를 사용하여 큐 구조를 활용
- **put를 append(), get을 pop(0)를 사용**

```
a = [1,2,3,4,5]
a.append(10)
a.append(20)
a.pop(0) # 1 출력
a.pop(0) # 2 출력
```

튜플 (tuple)

- 리스트와 동일한 개념이나 값의 변경이 불가능함
- 선언시 “[ ]” 가 아닌 “( )”를 사용
- 리스트의 연산, 인덱싱, 슬라이싱 등을 동일하게 사용

```
t = (1,2,3)
print t +t , t * 2          # (1, 2, 3, 1, 2, 3) (1, 2, 3, 1, 2, 3)
len(t)                      # 3
t[1] = 5            # Error 발생
```

집합 (set)

- 값을 순서없이 저장, 중복 불허 하는 자료형
- set 객체 선언을 이용하여 객체 생성

```
s = set([1,2,3])            #  set 함수를 사용 1,2,3을 집합 객체 생성
s.add(1)                    #  1를 추가, 중복불허로 추가 되지 않음
s.remove(1)                 #  1 삭제
s.update([1,4,5,6,7])       #  [1,4,5,6,7] 추가
s.discard(3)                #  3 삭제
s.clear()                   # 모든 원소 삭제
```

집합 연산

- 수학에서 활용하는 다양한 집합연산 가능

```
s1 = set([1,2,3,4,5])
s2 = set([3,4,5,6,7])
s1.union(s2)                    #  s1 과 s2의 합집합
s1 | s2                         # set([1, 2, 3, 4, 5, 6, 7])
s1.intersection(s2)             #  s1 과 s2의 교집합
s1 & s2                         # set([3, 4, 5])
s1.difference(s2)               # s1 과 s2의 차집합
s1 - s2                         # set([1, 2])
```

사전 (Dictionary)

- **key와 value를 매핑하여 key로 value를 검색**
- **다른 언어에서는 HashTable 이라는 용어를 사용**
- **{Key1:Value1, Key2:Value2, Key3:Value3 ...} 형태**

student_info = {20140012:'Sungchul', 20140059:'Jiyong',20140058:'JaeHong'}

```
student_info[20140012]
student_info[20140012] = 'Janhyeok'
student_info[20140012]
student_info[20140039] = 'wonchul‘
student_info
```

Key	Value
20140012	Janhyeok
20140059	Jiyong
20140058	JaeHong
20140039	wonchul

데이터 형 확인하기

type() 함수는 변수의 데이터형을 확인하는 함수

```
>>> a=int(10.3)          # a는 정수형으로 10.3을 할당
>>> b=float(10.3)        # b는 실수형으로 10.3을 할당
>>> c=str(10.3)          # c는 문자열로으로 10.3을 할당
>>> type(a)              # a의 타입을 출력
<type 'int'>
>>> type(b)              # b의 타입을 출력
<type 'float'>
>>> type(c)              # c의 타입을 출력
<type 'str'>
```

str() 함수는 숫자 값을 문자 값으로 변환함

Python 연산

수학연산 과 비교연산

연산자	설명
+	덧셈
-	뺄셈
*	곱셈
/	나눗셈
//	나눗셈의 몫
%	모듈로 (나눗셈의 나머지)
**	지수 연산자
+var	단항 덧셈
-var	단항 뺄셈

연산자	설명
>	큼
<	작음
>=	크거나 같음
<=	작거나 같음
!=	같지 않음
==	같음

비트연산, 쉬프트(이동) 연산, 논리(Bool)연산

연산자	설명
&	논리곱 연산자로서 비트가 두 항에 모두 나타나는 경우 비트를 결과에 복사함
|	논리합 연산자로서 비트가 두 항 중 어느 곳에 나타나는 경우 비트를 결과에 복사함
^	배타적 논리합 연산자로서 어느 한 쪽의 항에만 비트가 존재할 경우 비트를 결과에 복사함
~	부정 연산자로서 비트를 뒤집어서 각 비트에 대하여 정확히 반대를 반환함

x<<n	왼쪽으로 이동 (숫자 x에 2를 n번 곱한 것과 동등함)
x>>n	오른쪽으로 이동 (숫자 x를 2로 n번 나눈 것과 동등함)

조건부	논리
and	x가 거짓으로 False 반환하고, y를 평가하여 결과값을 반환
or	x가 참으로 판명되면 True 반환하고, y를 평가하여 결과값을 반환
not	not x 평가에 있어서, x가 거짓이라는 것은 x의 반대를 의미함

연산자 우선순위

가장 높은 것부터 가장 낮은 것까지의 연산자의 우선순위	이름
+var, -var, ~var	단항 연산자
* *	제곱 연산자
* , /, //, %	곱셈, 나눗셈, 몫, 나머지
+, -	덧셈, 뺄셈
<<, >>	왼쪽 및 오른쪽 이동
&	논리곱
^	배타적 논리합
|	논리합
<, >, <=. >= , <>	비교 연산자
==, != , is, is not, in, not in	동등 및 포함
and, or, not	여부 판단 조건

Python 제어

콘솔입출력

콘솔창 입출력 1

raw_input() 함수는 콘솔창에서 문자열을 입력받음

프로그램 정지효과

raw_input.py

```
print "Enter your name:"
somebody = input()                    # 콘솔창에서 입력한 값을 somebody에 저장
print "Hi", somebody, "How are you today?"
```

실행

```
$ python raw_input.py                 # 코드 실행
Enter your name:
Sungchul Choi                         # 콘솔 창에서 이름 입력
Hi Sungchul Choi How are you today?
```

콘솔창 입출력 2

콤마(,) 사용할 경우 print 문이 연결됨

```
print "Hello World!",                    # , 사용
print "Hello Again!!!“

Hello World! Hello Again!!!              # 실행시 두 문장이 열결되서 출력됨
```

숫자 입력 받기

```
fahrenheit  = float(input())  # 입력시 바로 형변환 하기
print Fahrenheit
 python float_input.py
30                                       # 실행시 30을 입력하면
30.0                                     # 30.0으로 형변환하여 출력
```

화씨변환기 연습

아래와 같이 출력되는 프로그램을 만드시오

```
본 프로그램은 섭씨를 화씨로로 변환해주는 프로그램입니다
변환하고 싶은 섭씨  온도를 입력해 주세요:
32.2                    # 사용자 입력
섭씨온도 : 32.2
화씨온도 : 89.96
```

※ 한글입력을 위해서 다음행을 첫줄에 삽입

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

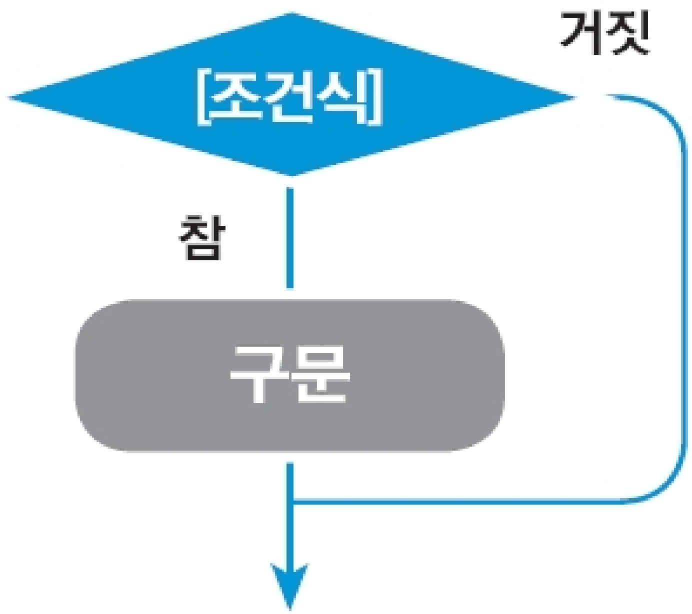
섭씨 온도 변환 공식은: ((9/5) * 섭씨온도) + 32

화씨변환기 답

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

print "본 프로그램은 섭씨를 화씨로로 변환해주는 프로그램입니다"
print "변환하고 싶은 섭씨  온도를 입력해 주세요: "
celcius = float(input())
fah = ( (9.0/5.0) * celcius ) + 32
print "섭씨온도 :", celcius
print "화씨온도 :", fah
```

제어

- if, elif, else
- 조건식의 참/거짓 판단
- 단축 평가
- while 문
- for 문
- break, continue, else
- 리스트 내장
- 제어문과 관련된 유용한 내장 함수



If 조건문

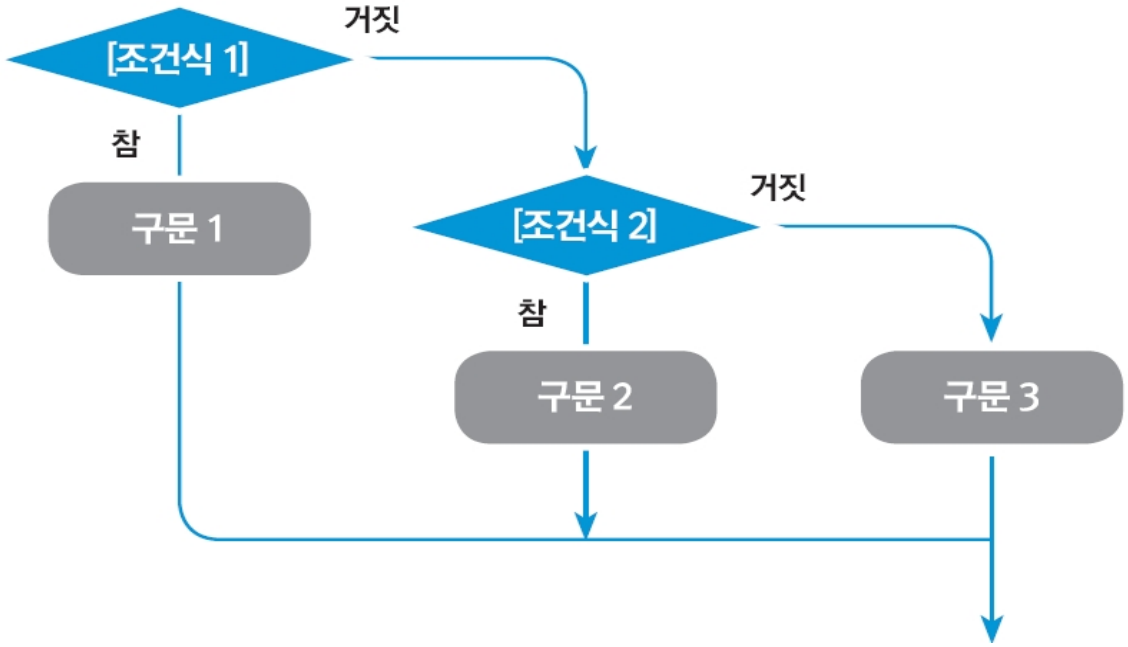
거짓
[조건식]
참
구문

if <□ □ □> **:**
<□ □>

```
>>> value=10
>>> if value > 5 :
...    print(“value is bigger then 5”)
```

- 조건식을 □ □ □ □ □ □ □ □ □ □ □ □ □ □
- 2 개 이상의 구문은 들여쓰기로 블록을 지정
  - □ □ □ □ □
  - 들여쓰기의 정도는 파일 전체를 통틀어 일치해야 함

83



If ~ elif ~ else 조건문

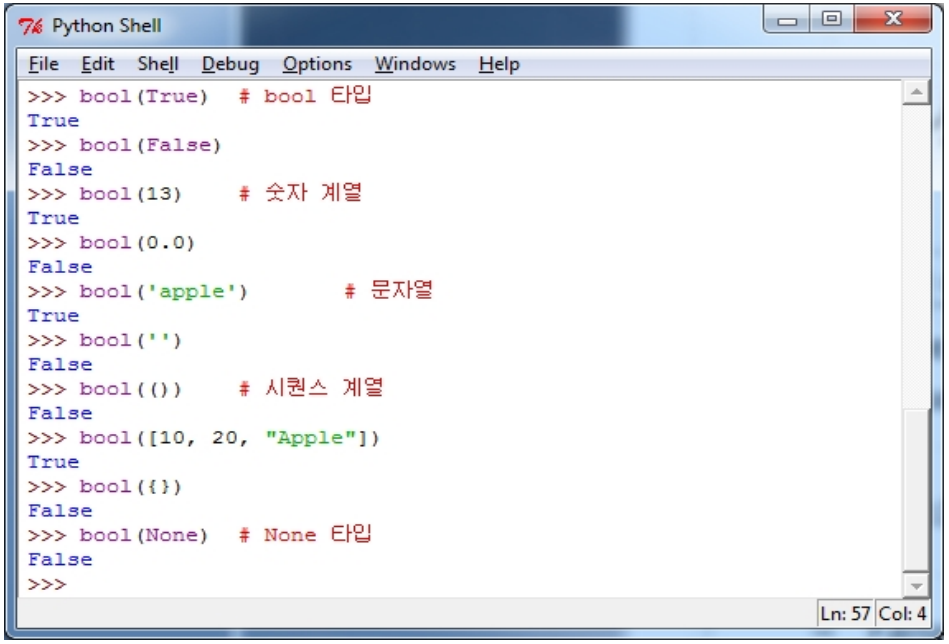
```
if <조건식 1> :
        <구문 1>
elif <조건식 2> :
        <구문 2>
else:
        <구문 3>
```

- elif
  - 2개 □□□ □□□ □□□□ □□
  - if는 가장 처음에만 사용할 수 있는 반면에 elif는 필요한 만큼 사용 가능 ,
- else
  - □□□ □□□□ □□□□ □□ □□
  - □□ □□□□□ □□ □□

If ~ elif ~ else 조건문 예제

```
# -*- coding: cp949 -*-
score = int(input(('Input Score: ')))  # 사용자로부터 정수값을 입력 받습니다
if 90 <= score <= 100:
        grade = "A"
elif 80 <= score < 90:
        grade = "B"
elif 70 <= score < 80:
        grade = "C"
elif 60 <= score < 70:
        grade = "D"
else:
        grade = "F"

print ("Grade is " + grade)
```

- 파이썬의 □ □ □ □ □ □ □
  - 70 <= score < 80
  - grade >= 70 and grade < 80



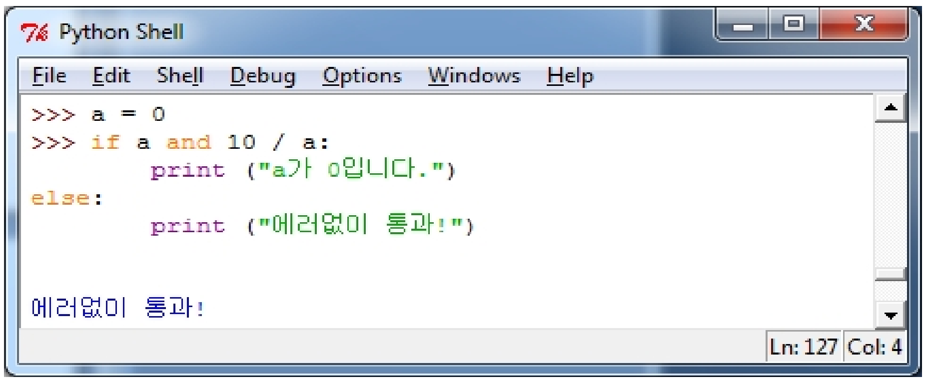
조건식의 참 거짓 판단

- 기본적으로 자료형의 bool 값과 동일 (2 장 참조)
  - True 로 판명 : 10 > 0
  - False 로 판명 : 5 > 10
- False
  - 0, 0.0, (), [], {} “(빈 문자열), None 인 경우
- True
  - False 인 경우를 제 외 한 값이 할당된 경우

and / or

- 2개 □□□ □□□ □□□ □□ □□□
  - □□ □□□□□ □□□□□□ □□
  - **and, &**
    - score > 70 **and** score <= 80
    - score > 70 **&** score <= 80
  - **or, |**
    - math > 80 **or** english > 80
    - math > 80 **|** english > 80

- 하지만 모든 경우에 'and'와 '&', 'or'와 '|'가 동일하게 수행
  □□ □□ □□!!



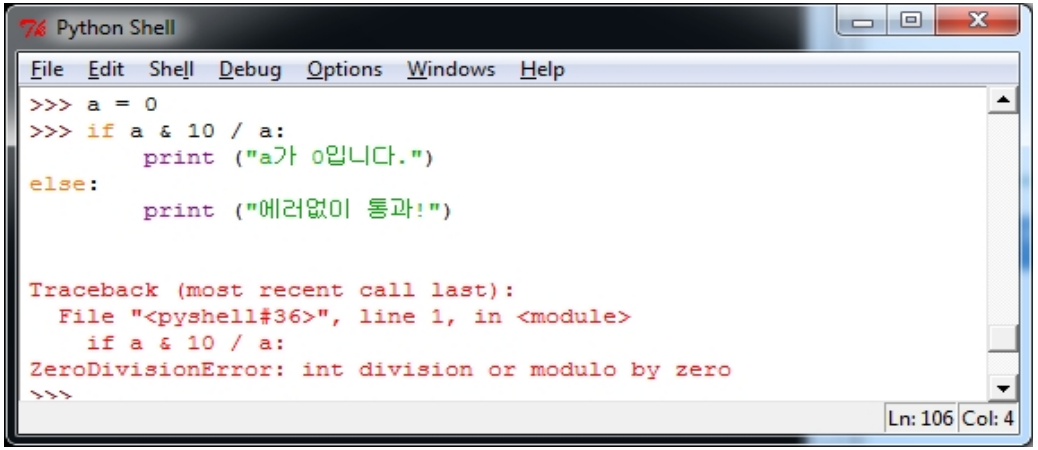
단축평가

- 단축 □ □ □?
  - 조건식 전체를 판단하지 않고 순차적으로 진행하다 식 전체가 자명한 □ □, □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
- 'and'와 'or'는 단축 평가로 수행되도록 보장
  - x and y: x가 False인 경우, y값은 평가하지 않음
  - x or y: x가 True인 경우, y값은 평가하지 않음

Python Shell
File Edit Shell Debug Options Windows Help
Ln: 127 Col: 4

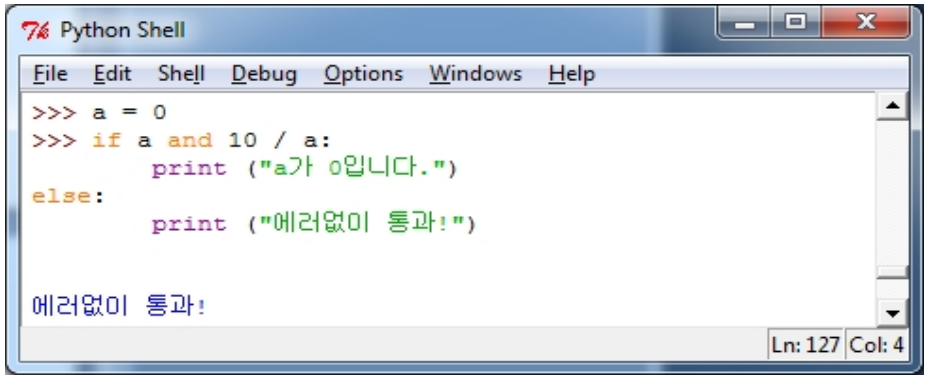
```
>>> a = 0
>>> if a and 10 / a:
	print ("a가 0입니다.")
else:
	print ("에러없이 통과!")


에러없이 통과!
```

88



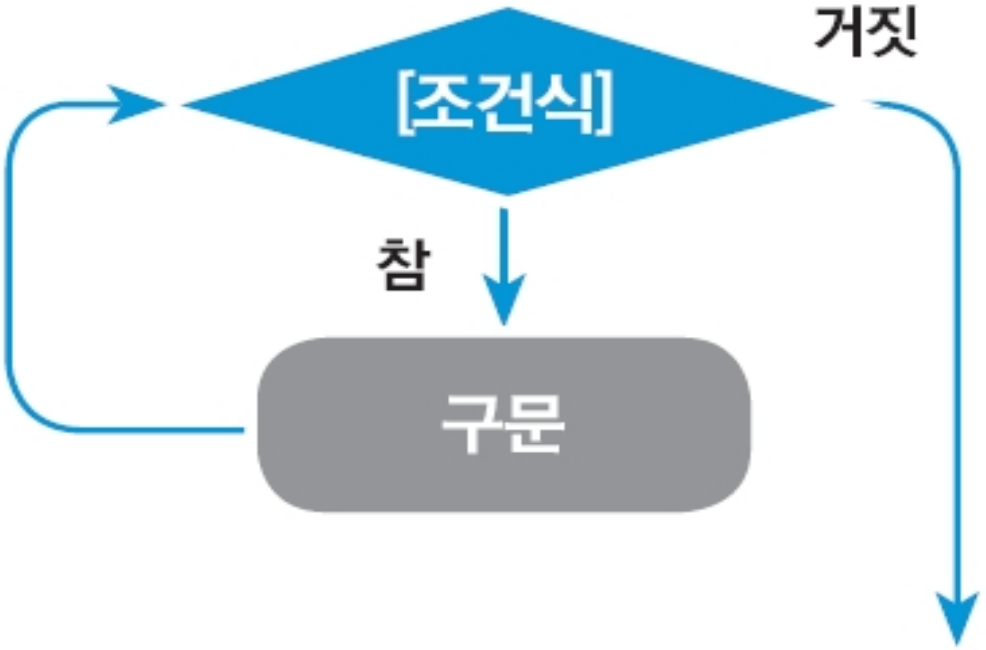
단축평가

- a 가 0 인 경우 조건식 'a & 10/a'는 거짓
  - 10 / a 에 의하여 ZeroDivisionError 가 발생



단축평가의 장점

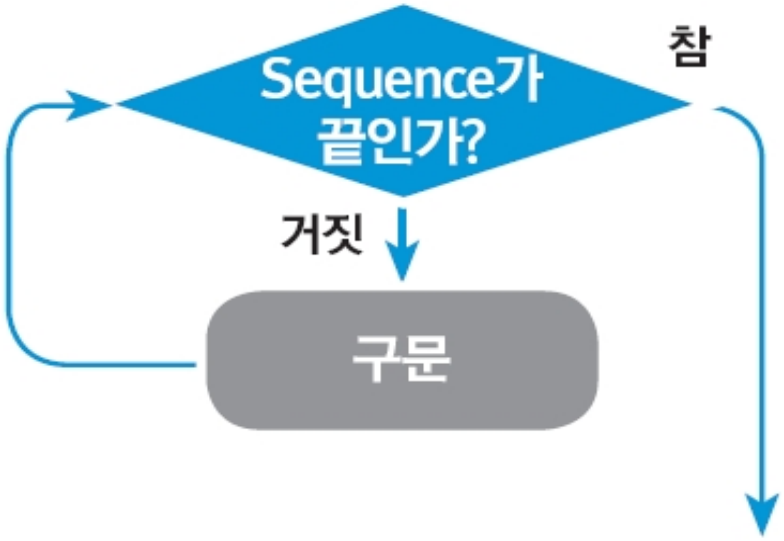
- 조건식의 결과가 결정되는 시점 이후로 추가적인 판별 연
  □□ □□□□ □□ □□□ □□ □□
- Run time error □□□ try~except □□□ □□ □□□
  □□ □□□ □□ □□



while 반복문

while <□ □ □ > :
<□ □ >

- 조건식이 참 (True)인 동안 내부 구문을 반복 수행
  - □□□□ □□□ □□□□ □□□ □□ □□
  - □□□ □□ □□ □□ □□ □□□□ □□□
  - 조건식이 거짓 (False)이면 while 문 구조를 벗어남

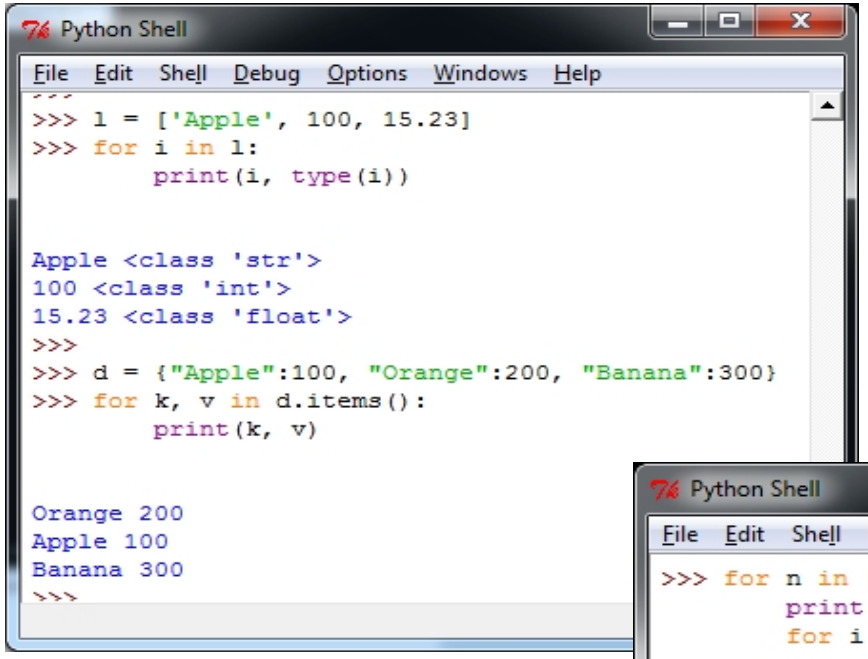


for 반복문

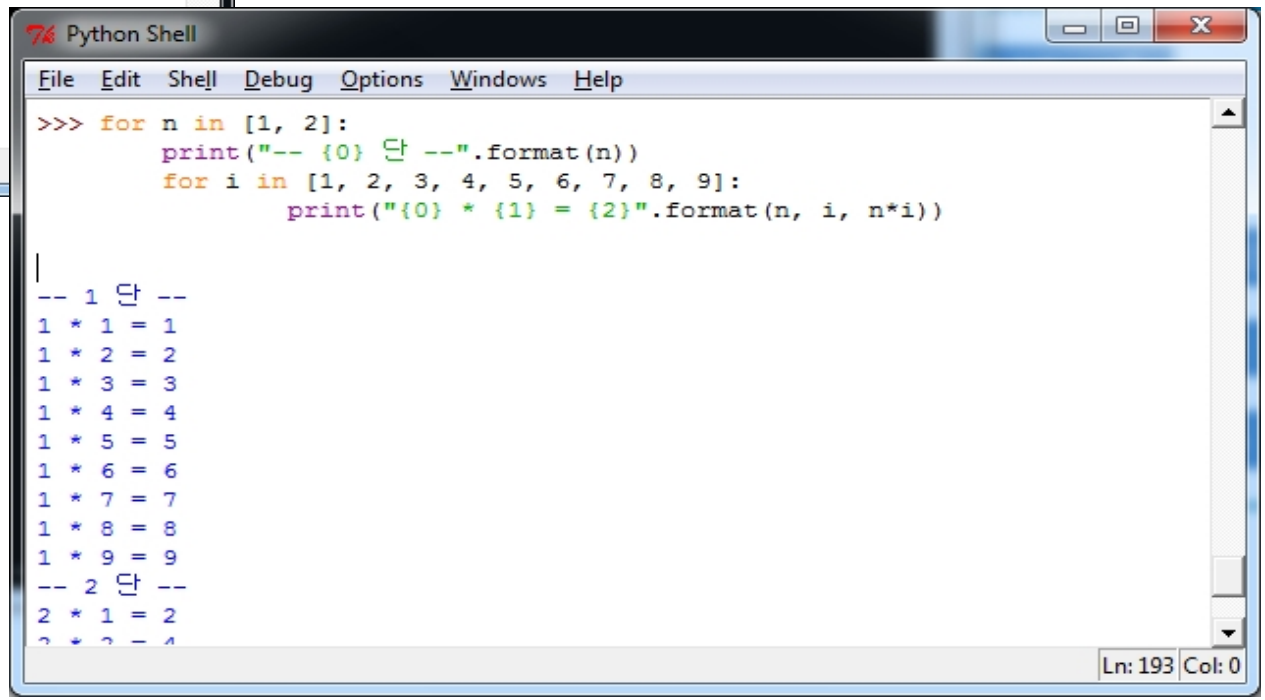
for <□ □ □ I> in <□ □ □ □ □ □ S> :
<□ □ >

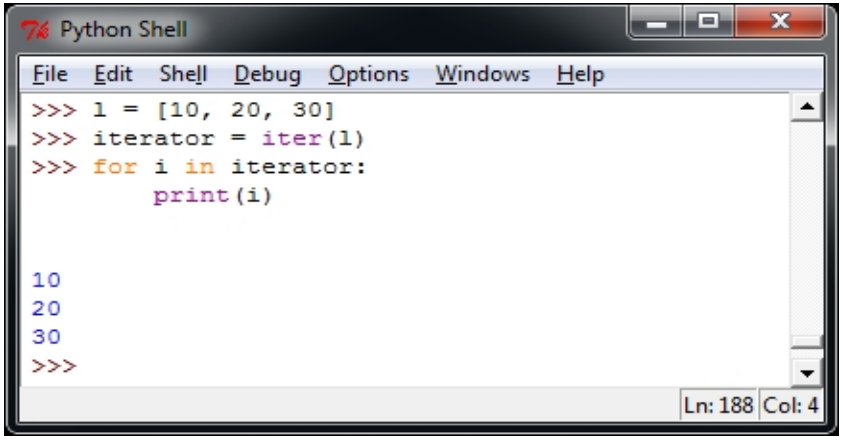
- 시퀀스형 □ □ □ □ □ □ □ □ □ □
  - ‘시퀀스형 객체 S’의 각 아이템을 ‘아이템 I’에 할당
  - 할당된 아이템 I를 가지고 구문을 수행
  - 모든 아이템을 순회하거나 break을 만나면 for문이 종료

for 반복문 예제



- 반복문은 2개 이상 중첩해서 사용 가능



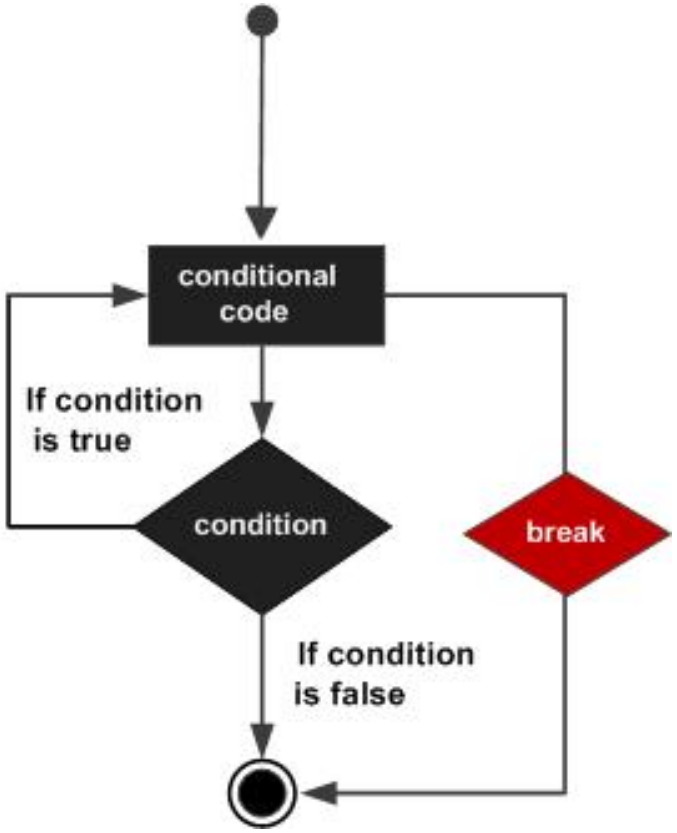


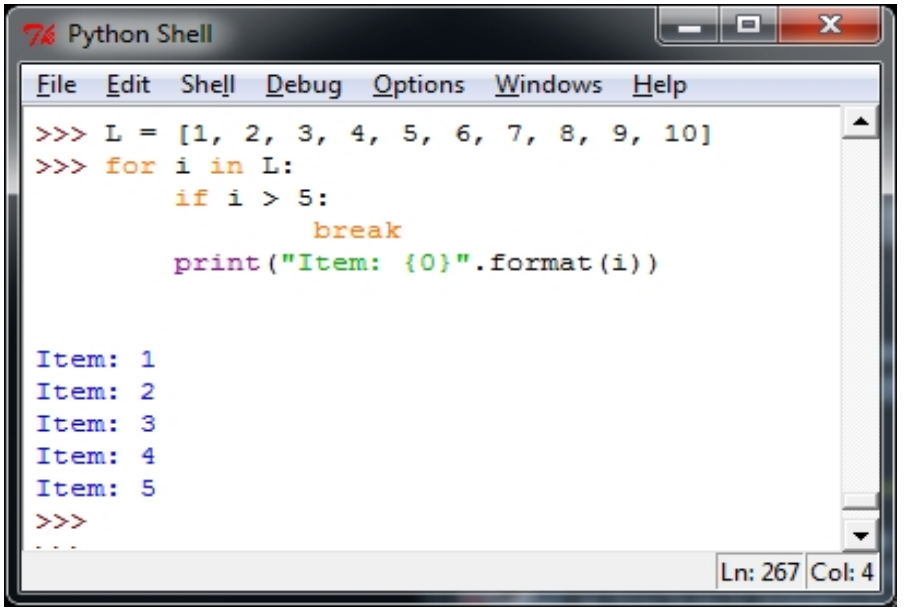
for 반복문 대상 자료

- for문에서 □ □ □ □ □ □ □ □
  - □ □ □, □ □ □, □ □, □ □
  - 이터레이터 (iterator), 제너레이터 객체 (3 장 참조)

break 중지문

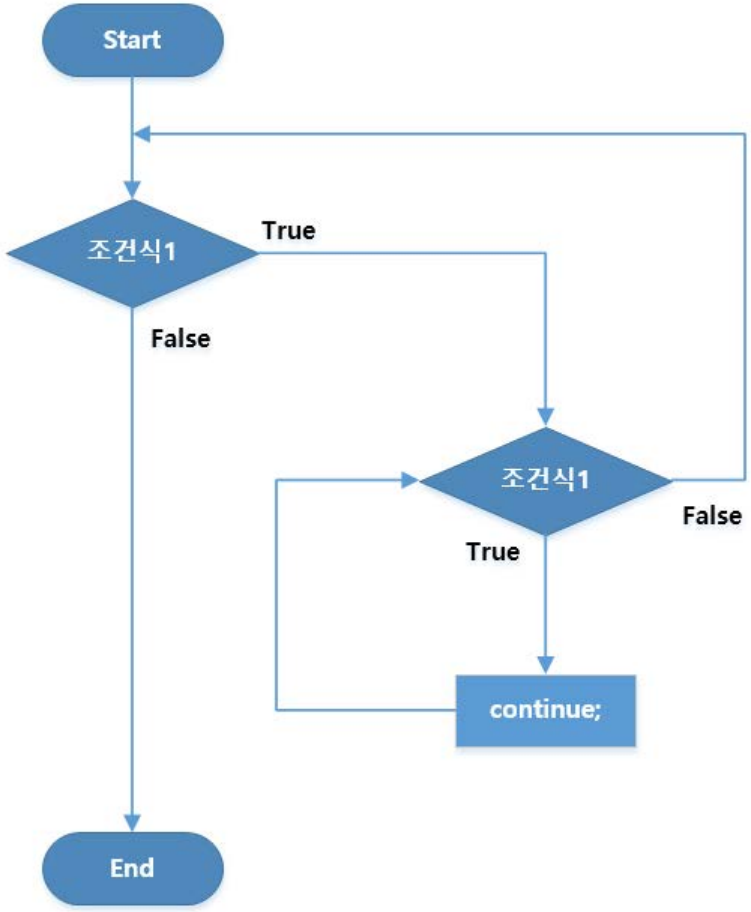
- break을 □□□ □□□ □□ □□□ □□□

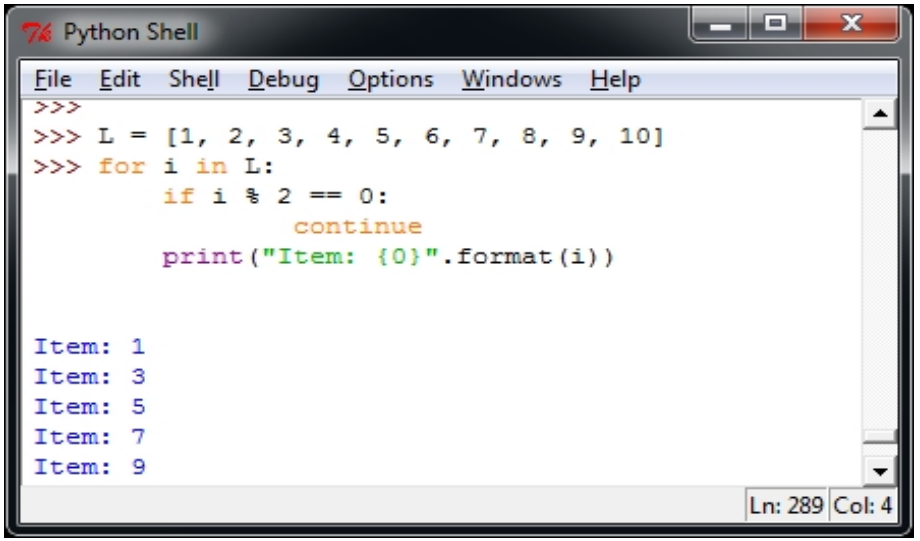




continue 문

- continue 이후 □□□ □□ □□□ □□□□ □□, □□ □□□□ □□□□ □□ □□□ □□ □□□□ □□





Else 문

- 반복문 수행도중 break로 인하여 중간에 종료되지 않고 끝가지 수행되었을 때, else 블록이 수행
  - else □□□ □□□□ □□


```
# -*- coding: cp949 -*-
L = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for i in L:
    if i % 2 == 0:
        continue
    print("Item: {0}".format(i))
else:
    print("Exit without break.")
print("Always this is printed")   # 외부 루프 문장
```



```
Item: 1
Item: 3
Item: 5
Item: 7
Item: 9
Exit without break.
Always this is printed
```

  - else □□□ □□□□ □□ □□


```
# -*- coding: cp949 -*-
L = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for i in L:
    if i > 5:
        break
    print("Item: {0}".format(i))
else:
    print("Exit without break.")
print("Always this is printed")   # 외부 루프 문장
```



```
Item: 1
Item: 2
Item: 3
Item: 4
Item: 5
Always this is printed
```

97

Pass 구문

- 아무 □□ □□ □□□□.

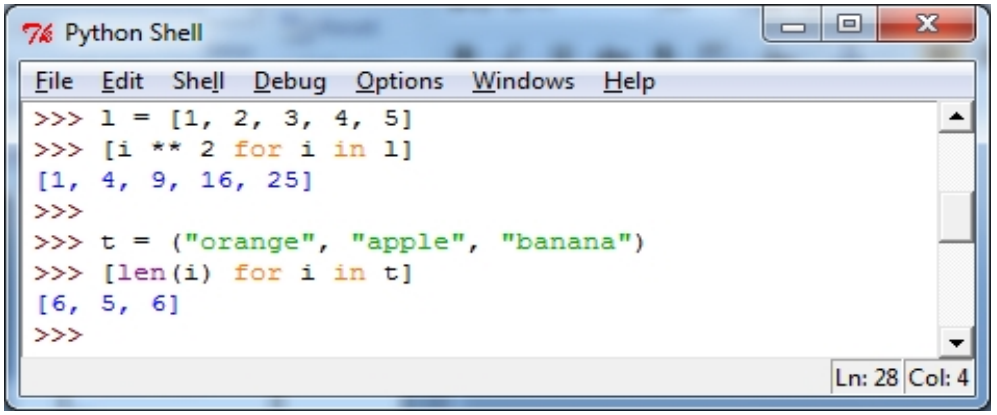
```
>>> while True:
        pass
```

- 아무것도 하지 않는 함수 , 모듈 , 클래스를 만들어야 할 경우가 있는데 . 이 때 pass가 사용될 수 있다 .
- □□ □□□ □□□, □□, □□□□ □□ □□□ □□ □□ □□

```
>>>def sample():
        pass

>>>sample()
>>>class pre_class():
        pass
>>>
```

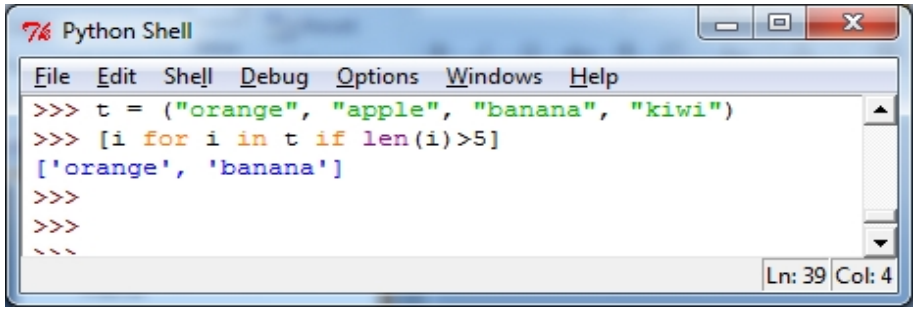
리스트 내장 반복문

- [<표현식> for <아이템> in <시퀀스 객체> (if <조건식>)]
  - □□ □□□ □□□ □□□□ □□□□ □□□ □□□ □□□ □□□ □□□ □□

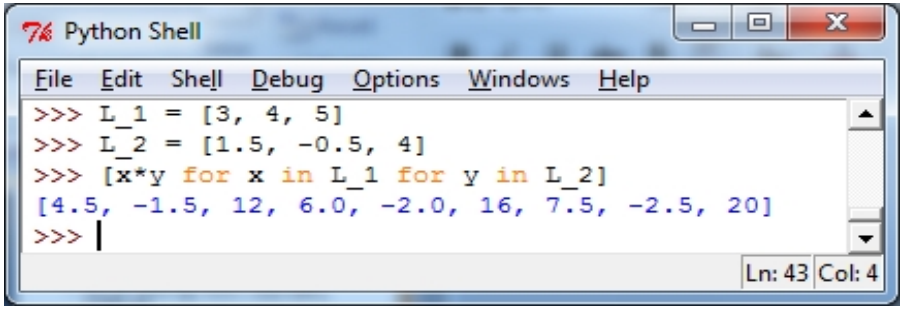


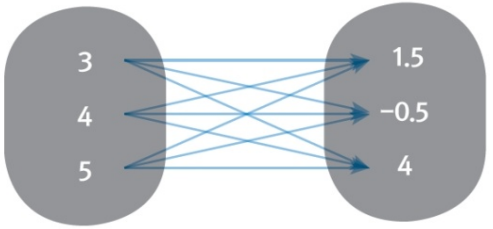
리스트 내장 반복문

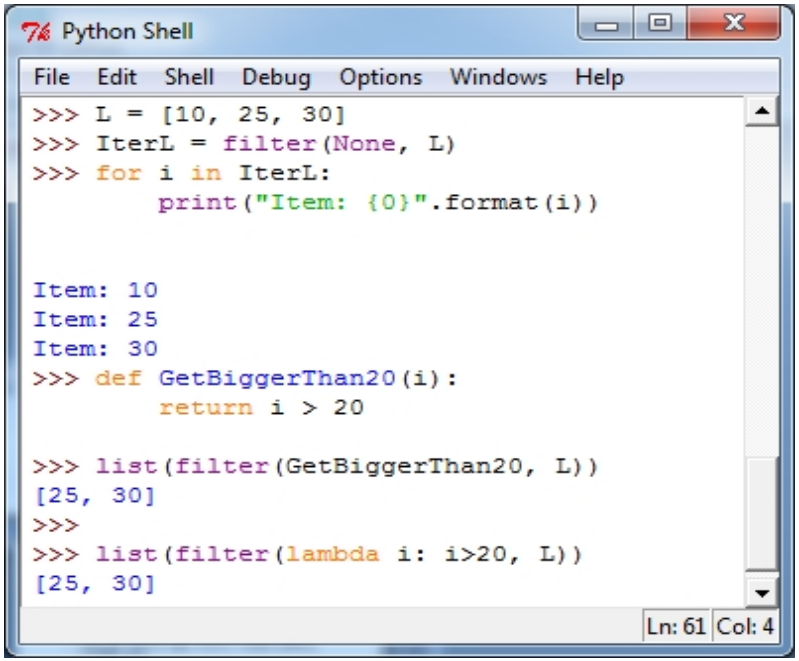
- [<표현식> for <아이템> in <시퀀스 객체> (if <조건식>)]
  - □□□□ □□□□ □□ □□□□ □□□ □□□□ □□□□ □□



- □□ □□□□ 2□□ □□

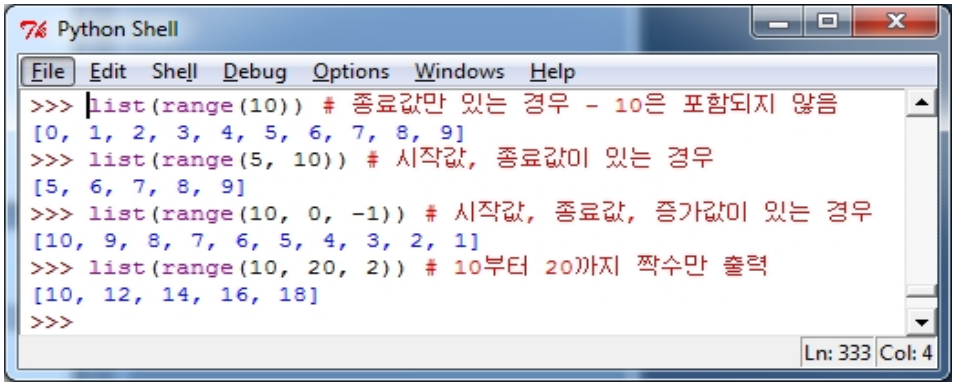






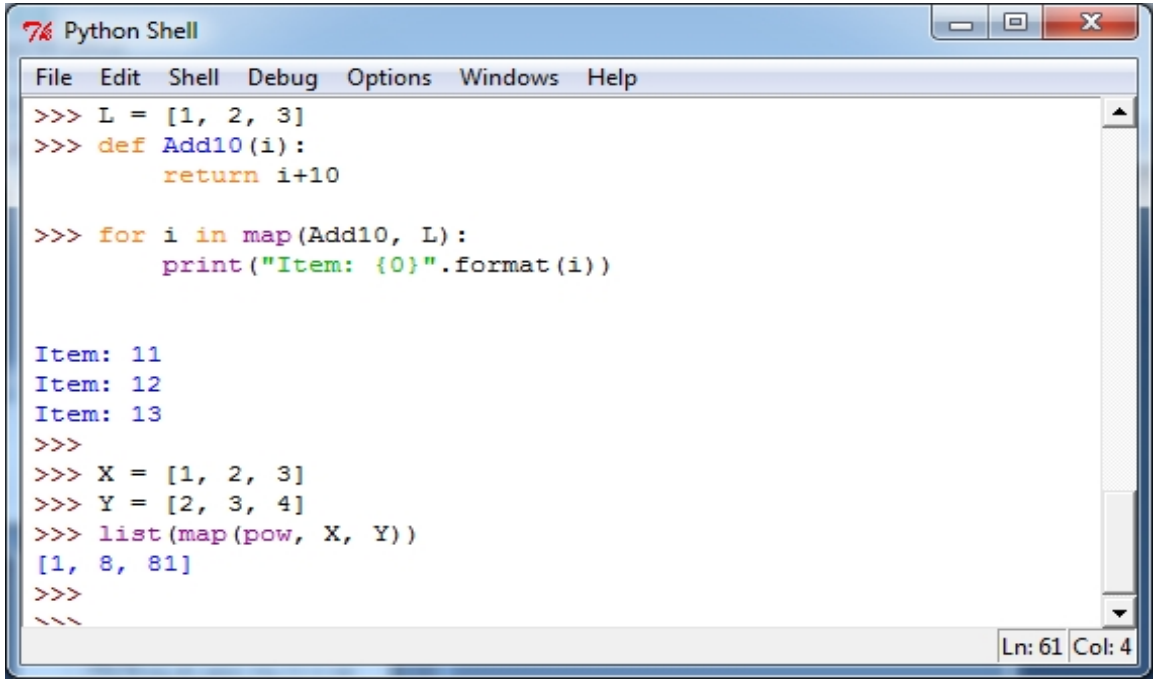
제어에 유용한 함수

- **filter**(<function>|None, 시퀀스 □ □)
  - □
  - None 이 오는 경우 필터링하지 않음



제어에 유용한 함수

- **range**(['□ □ □'], '□ □ □'[, '□ □ □'])
  - □
  - 시작값과 증가값은 생략 가능하며 , 이때는 각 0, 1 이 할당



제어에 유용한 함수

- map(<function>, 시퀀스 □ □, …)
  - 시퀀스 객체를 순회하며 function 의 연산을 수행
  - □□□ □□□□□ □□□ □□□ □□

Python 함수

형 변환 함수

함수	설명
chr(value)	정수를 문자로 변환
complex(real [,imag])	복소수를 만듦
dict(sequence)	주어진 (키, 값) 튜플의 순서로부터 사전을 만듦
eval(string)	문자열을 평가하여 수학적 계산에 유용한 개체를 반환. 주의: 이 함수는 올바로 사용하지 않을 경우 보안 위험을 초래할 수 있으므로 매우 조심하여야 함
float(value)	숫자를 부동소수점수로 변환
frozenset(set)	집합을 동결집합으로 변환
hex(value)	정수로부터 16진수를 나타내는 문자열로 변환
int(value [, base])	정수로 변환하며 base 문자열이 주어진 경우에는 사용함
list(sequence)	주어진 순서를 목록으로 변환
long(value [, base])	long(긴 정수)형으로 변환하며 base 문자열이 주어진 경우에는 사용함
oct(value)	정수로부터 8진수를 나타내는 문자열로 변환
ord(value)	문자를 가리키는 정수값으로 변환
repr(value)	개체를 표현 문자열로 변환. 표현식을 역따옴표로 둘러싼 것과 같음(`x + y`). 개체에 대하여 출력가능하며 평가 가능한 문자열을 반환
set(sequence)	sequence를 집합으로 변환
str(value)	개체를 문자열로 변환. value에 대하여 출력가능한 문자열을 반환 하나, 평가할 수는 없음
tuple(sequence)	주어진 sequence를 튜플로 변환
unichr(value)	정수를 유니코드 문자로 변환

내장 수학 함수

기능	□□
abs(var)	□□□
pow(x, y)	** □□□ □□□ □□ □□
pow(x,y,modulo)	□□ □□-□□□ (x ** y) % modulo
round(var[, n])	10-n 또는 (10**-n)의 반올림한 근사치를 반환 . n 의 기본값은 0
divmod(x, y)	□□□□ □□ □□□□ □□□□ □□□ □□

문자열 함수 (1/2)

함수명	기능
len(a)	□□□□ □□ □□□ □□
a.upper()	□□□□ □□
a.lower()	□□□□ □□
a.capitalize()	첫 문자를 대문자로 변환
a.titile()	□□□□□ □□ 띄워쓰기후 첫 글자만 대문자
a.count('abc')	문자열 a 에 'abc' 가 들어간 횟수 반환
a.find('abc') a.rfind('abc')	문자열 a 에 'abc' 가 들어간 위치 (오프셋) 반환
a.startswith('ab'c)	문자열 a 는 'abc' 로 시작하는 문자열여부 반환
a.endswith('abc')	문자열 a 는 'abc' 로 끝나는 문자열여부 반환

문자열 함수 (2/2)

함수명	기능
a.strip()	□□ □□□ □□
a.rstrip()	□□□ □□□ □□
a.lstrip()	□□ □□□ □□
a.split()	□□□ □□□□ □□ □□□□ □□
a.split('abc')	abc를 기준으로 나눠 리스트로 반환
a.isdigit()	□□□□ □□□□ □□ □□
a.islower()	□□□□ □□□□□ □□ □□
a.isupper()	□□□□ □□□□□ □□ □□

함수(Function)

- 함수의정의
- return
- 인수전달
- 스코핑 룰
- 함수 인수
- Lambda 함수
- 재귀적 함수 호출
- Pass
- __doc__속성과 help 함수
- 이터레이터
- 제네레이터

함수 정의

- 함수는 여러 개의 문장(statement)을 하나로 묶어 줌
- 이미 정의 되어 있는 함수를 사용하거나 필요한 함수를 정의함
- 한 번 혹은 여러 번 호출 될 수 있으며 함수 종료 시 결과값을 전달.
- 프로그램을 구조적, 논리적으로 만들어 준다.
- 함수의 선언은 def로 시작하고 콜론(:)으로 끝낸다.
- 함수의 시작과 끝은 코드의 들여쓰기로 구분
- 시작과 끝을 명시해 줄 필요가 없다.
- 헤더(header)파일, 인터페이스(interface)/구현(implementation)같은 부분으로 나누지 않음

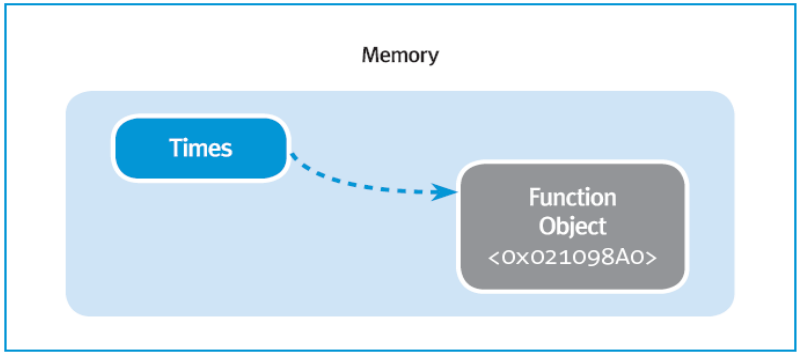
함수 선언 방법

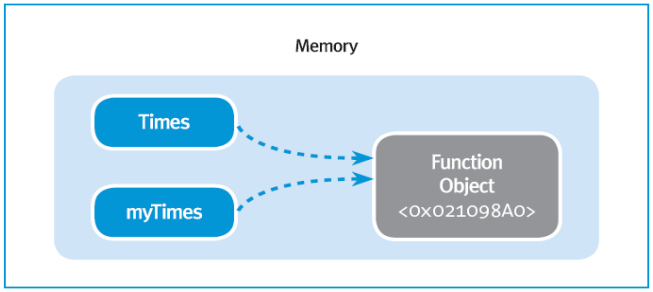
- 함수 □□ □□

```
def <함수명>(인수1, 인수2, ...인수N):
    <구문>
    return <반환값>
```

- □□□ □□ □□□ □□
  - □□ □□ 2□□ □□□□ □□ □□ □□ □□□□.

```
def Times(a, b):
    return a*b

>>> Times(10, 10)
100
```





함수 선언시 일어나는 일

- 메모리에 □□ □□□ □□□□.
- □□ □□□ □□□□ □□□□□ □□□□.

- □□ □□□□□ □□□ □□□ □□□□ □□.
  - 함수 레퍼런스는 다른 변수에 할당 할 수 있다 .

```
>>> myTimes = Times
>>> r = myTimes(10, 10)
>>> r
100
```

return 반환문

- 함수를 □□□□□ □□□ □□□ □□□□ □□.
- return은 어떠한 객체로 돌려줄 수 있다 .
  - 여러 개의 값을 튜플로 묶어서 값을 전달 할 수 있음

```
>>> def swap(x, y):
        return y, x

>>> swap(1, 2)
(2,1)
```

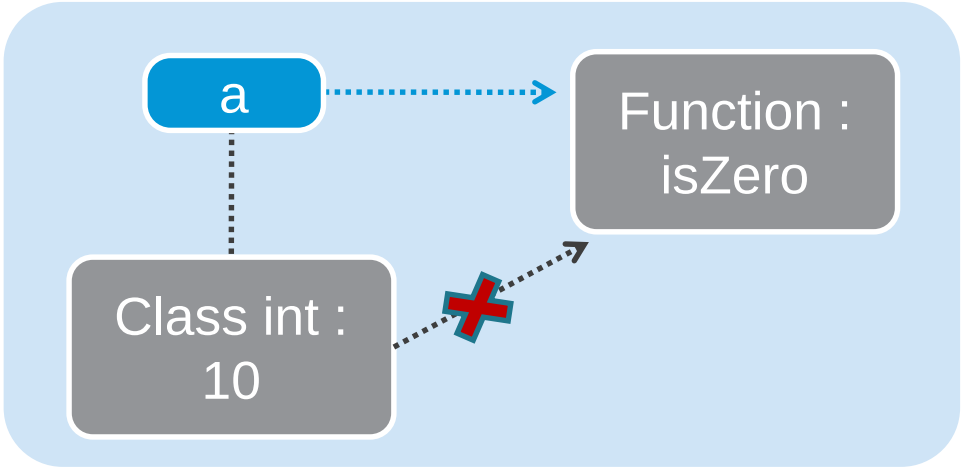
- return을 사용하지 않거나 return만 적을 때도 함수가 종료
  - 리턴값으로 None을 리턴

```
>>> def setValue(newValue):
        x = newValue      ← 반환 값이 없는 경우

>>> retval = setValue(10)

>>> print(retval)
None
```

파이썬 함수에서 인수전달 1

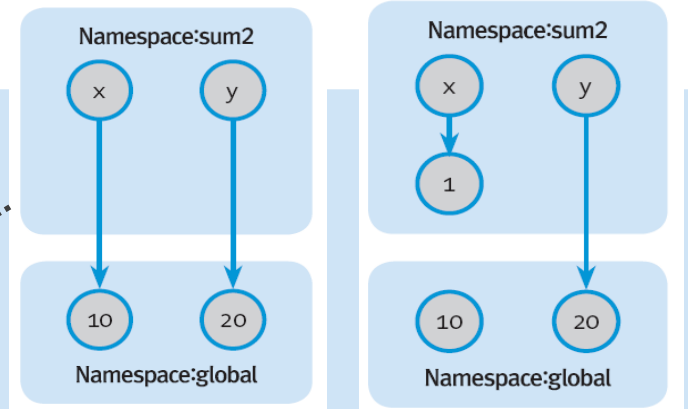
- 파이썬에서 □□ □□□ □□□□□ □□□ □□
  - □□□ □□□ □□□ □□ □□□ □□□□

```
>>> a = 10
>>> def isZero(arg1):
        return arg1 == 0
>>> isZero(a)
False
```



파이썬 함수에서 인수전달 2

- 호출자가 □□□□ □□□ □□□ □□ □□□ □□
  - □□□□ □□ (mutable)
  - □□□ □□ (immutable)
- □□ □□□ □□ □□




```
>>> a = 10
>>> b = 20
>>> def sum1(x, y):
        return x + y
>>> sum1(a, b)
30
>>> x = 10
>>> def sum2(x, y):
        x = 1        ← 이 부분에서 값이 1인 객체가 생성되고 x에 레퍼런스를 할당합니다
        return x + y
>>> sum2(x, b)      ← x를 인수로 넣어 줍니다
21
>>> x               ← 함수 내부에서 변경한 사항이 외부에 영향을 미치지 않습니다
10
```

파이썬 함수에서 인수전달 3

- 변경가능한 □ □ □ □ □ □ □ □ .

```
>>> def change(x):
        x[0] = 'H'          ← list x의 첫 번째 아이템을 H로 바꿉니다
>>> wordlist = ['J', 'A', 'M']
>>> change(wordlist)
>>> wordlist
['H', 'A', 'M']          ← change가 호출자의 객체에게 영향을 미칩니다
```

```
>>> def change(x):
        x = x[:]            ← 입력받은 인수를 모두 x에 복사합니다
        x[0] = 'H'          ← list x의 첫 번째 아이템을 H로 바꿉니다
        return None
>>> wordlist = ['J', 'A', 'M']
>>> change(wordlist)
>>> wordlist
['J', 'A', 'M']          ← change가 호출자의 객체에 영향을 미치지 않습니다
```

범위 규칙 (Scoping Rule)

- 이름공간 (Name Space)
  - □□□ □□□ □□□□ □□ □□
  - □□ □□□ □□□□, □□ □□(Local scope)
  - □□ □□ □□, □□ □□(Global scope)
  - □□□ □□□□ □□□ □□□ □□ □□, □□ □□ (Built-in Scope)
- LGB □□
  - 변수 이름을 찾을 때 Local Scope -> Global Scope -> Built-in Scope □□□ □□□.

```
>>> x = 1
>>> def func(a):
        return a + x
>>> func(1)
2
>>> def func2(a):
        x = 2
        return a +
>>> func2(1)
3
```

- 지역 영역에서 전역 영역의 이름을 접근할 때 global을 이용

파이썬 함수에서 인수 모드 1

- 기본 인수 값
  - 함수를 호출 할 때 인수를 지정해 주지 않아도 기본 값이 할당되도록 □□ □□.

```
>>> def Times(a = 10, b = 20):
        return a * b
>>> Times()
200
>>> Times(5)
100
```

- □□□ □□
  - □□ □□□□ □□ □□□□ □□
  - 변수의 이름으로 특정 인수를 전달할 수 있다 .

```
>>> def connectURI(server, port):
        str = "http://" + server + ":" + port
        return str
>>> connectURI("test.com", "8080")
'http://test.com:8080'
>>> connectURI(port="8080", server="test.com")
'http://test.com:8080'
```

파이썬 함수에서 인수 모드 2

- 가변인수 □ □ □
  - □
  - * 를 사용하며 인수는 튜플 형식으로 전달 됨 .


```
>>> def union2(*ar):
        res = []
        for item in ar:
            for x in item:
                if not x in res:
                    res.append(x)
        return res

>>> union2("HAM", "EGG", "SPAM")
['H', 'A', 'M', 'E', 'G', 'S', 'P']
```

- □ □ □ □ □ □ □ □ □ □ □ □ .
  - ** 를 사용하면 정의되지 않은 인수를 사전형식으로 전달


```
>>> def userURIBuilder(server, port, **user):
        str = "http://" + server + ":" + port + "/?"
        for key in user.keys():
            str += key + "=" + user[key] + "&"
        return str

>>> userURIBuilder("test.com", "8080", id='userid', passwd='1234')
'http://test.com:8080/?passwd=1234&id=userid&'

>>> userURIBuilder("test.com", "8080", id='userid', passwd='1234', name='mike',
age='20')
'http://test.com:8080/?passwd=1234&age=20&id=userid&name=mike&'
```

119

람다 (Lambda) 함수

- 이름이 없는 1 줄 짜리 함수

```
lambda 인수 : <구문>
```

- 한 줄의 간단한 함수가 필요한 경우
- □ □ □ □ □ □ □ □ □ □ □ □ □
- 함수를 인수로 넘겨 줄 때

```
>>> g = lambda x, y : x * y
>>> g(2, 3)
6
>>> (lambda x: x * x)(3)
9
```

재귀적(Recursive) 함수 호출

- 함수 □□□□ □□ □□□ □□ □□ □□ □□
  - 변수를 조금씩 변경하면서 연속적으로 반복된 연산을 할 때 유용함

```
>>> def factorial(x):
        if x == 1:
            return 1
        return x * factorial(x - 1)
>> factorial(10)
3628800
```

__doc__ 속성과 help 함수

- help 함수를 이용해 함수의 설명을 볼 수 있다 .

```
>>> help(print)
```

- 사용자가 만든 함수도 help 를 사용해 설명을 볼 수 있다 .

```
>>> def plus(a, b):
        return a + b

>>> help(plus)
Help on function plus in module __main__:
plus(a, b)
```

- [□ □ □ □ .

```
>>>plus.__doc__ = "return the sum of parameter a, b "
>>> help(plus)
Help on function plus in module __main__:
plus(a, b)
    return the sum of parameter a, b
```

반복자(Iterator) – 홑개, 순회변수

- 순회가능한 객체의 요소를 순서대로 접근 할 수 있는 객체
  - □□ □□□□ □□□ □□ □□
  - 이터레이터 안의 __next__() 를 이용해 순회 가능한 객체의 요소를 하나씩 접근 할 수 있다 .

```
>>> s = 'abc'
>>> it = iter(s)
>>> it
<iterator object at 0x00A1DB50>
>>> next(it)
'a'
>>> next(it)
'b'
>>> it.__next__()
'c'
>>> next(it)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
    next(it)
StopIteration
```

생성기 (Generator)

- return 대신 yield □□ □□□ □□□ □□ □□□ □□□
  체 값을 호출자에 넘겨줌
  - 값을 넘겨준 후 함수 객체는 그대로 유지
  - 함수의 상태를 그대로 유지하고 다시 호출 할 수 있기 때문에
    순회가능한 객체를 만들 때 매우 편리함 .

```
>>> def reverse(data):
        for index in range(len(data) - 1, -1, -1):
            yield data[index]

>>> for char in reverse('golf'):
        print(char)

f

l

o

g
```

Generator 예제

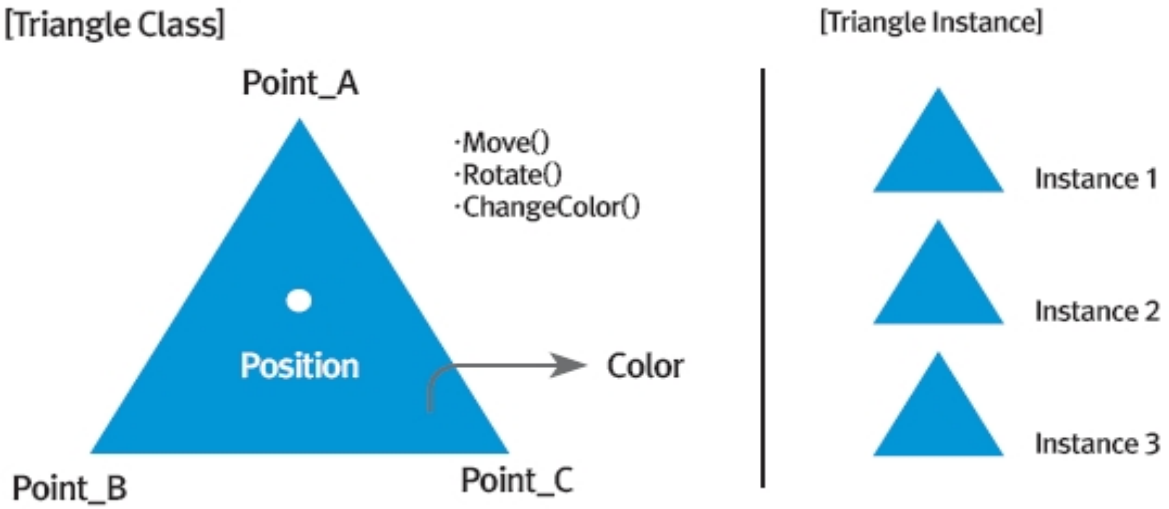
```
>>> def abc():
        data = "abc"
        for char in data:
            yield char


>>> abc
<function abc at 0x0205EB70>
>>> abc()
<generator object abc at 0x02061A30>

>>> it = iter(abc())
>>> next(it)
 'a'
>>> next(it)
 'b'
>>> next(it)
 'c'
```

클래스

- 클래스란?
- 클래스 기본
- 이름 공간
- 클래스와 인스턴스 관계
- 생성자와 소멸자
- 메서드 확장
- 연산자 중복
- 상속

클래스란 ?

- 데이터와 □



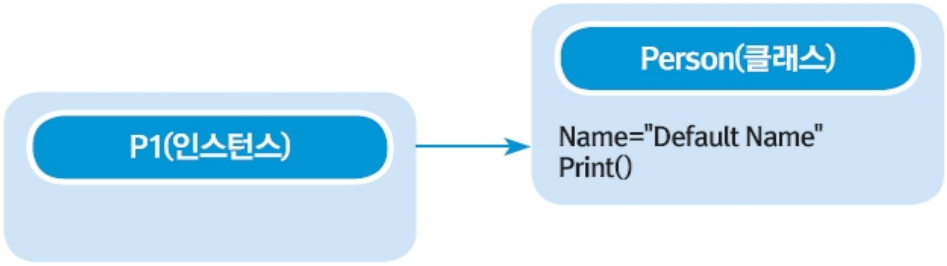
- □ □ □ (Method)
- □ □ □ □ (Instance)
- □ □ □ □ (Information Hiding)
- □ □ □ (Abstraction)
  - □ □ □ □ □
  - □ □ □ □ □

클래스 기본

- 클래스와 □ □ □ □

```
class Person:  # 클래스 정의
    Name = "Default Name" # 멤버 변수

    def Print(self): # 멤버 메소드
        print("My Name is {0}".format(self.Name))

p1 = Person() # 인스턴스 객체 생성

p1.Print() # 멤버 변수값을 출력
```

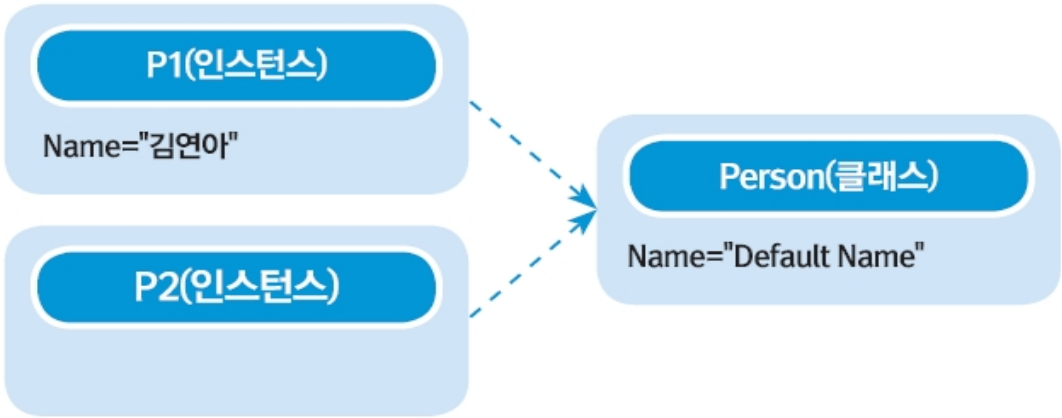
- □ □ □ □ □ □ □ □ □ □ □ □



이름 공간 (name space)

- 검색 □ □
  - □ □ □ □ □ □ □ □ -> □ □ □ □ □ □ □ □ -> □ □ □ □
  - □ □ □ □

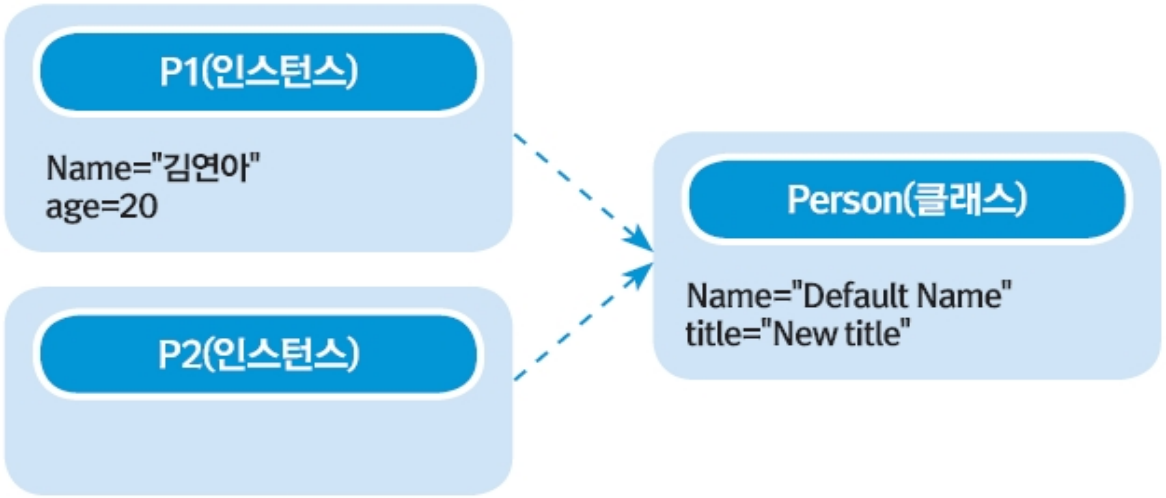
```
>>> class Person:       # 클래스 정의
        name = "Default Name"


>>> p1 = Person()       # 인스턴스 객체 생성
>>> p2 = Person()
>>> print("p1's name: ", p1.name) # 각 객체의 name 속성 출력
p1's name:  Default Name
>>> print("p2's name: ", p2.name)
p2's name:  Default Name
>>> p1.name = "김연아"  # p1 인스턴스의 'name' 속성을 변경
>>> print("p1's name: ", p1.name)
p1's name:  김연아
>>> print("p2's name: ", p2.name)
p2's name:  Default Name
```



이름공간 계속

- 클래스와 □ □ □ □ □ □ □ □ □ □ □ □
  - □□ □□

```
>>> Person.title = "New title" # 클래스 객체에 새로운 멤버 변수 title 추가
>>> p1.age = 20 # p1 객체에만 age 멤버 변수를 추가
>>> print("p1's title: ", p1.age)
p1's title:  20
>>> print("p2's title: ", p2.age)
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    print("p2's title: ", p2.age)
AttributeError: 'Person' object has no attribute 'age'
```



클래스와 인스턴스의 관계

- isinstance(인스턴스 □□, □□□ □□)
  - □□□□ □□□ □□ □□□□□□ □□□□□□ □□
  - □□ □□□ □□ □□
  - □□ □□

```
>>> class Person:
        pass

>>> class Bird:
        pass

>>> class Student(Person):
        pass

>>> p, s = Person(), Student()
>>> print("p is instance of Person: ", isinstance(p, Person))
p is instance of Person:  True
>>> print("s is instance of Person: ", isinstance(s, Person))
s is instance of Person:  True
>>> print("p is instance of object: ", isinstance(p, object))
p is instance of object:  True
>>> print("p is instance of Bird: ", isinstance(p, Bird))
p is instance of Bird:  False
```

생성자와 소멸자

- 생성자
  - □□□ □□□ □□□ □□
  - 인스턴스 객체가 생성될 때 자동으로 호출
  - __init__()
- □□□
  - □□□ □□ □□□ □□
  - 인스턴스 객체의 참조 카운터가 '0' 이 될 때 호출
  - __del__()
- □□ □□

```
class MyClass:
        def __init__(self, value): # 생성자 메소드
                self.Value = value
                print("Class is created! Value = ", value)

        def __del__(self): # 소멸자 메소드
                print("Class is deleted!")

def foo():
        d = MyClass(10) # 함수 foo 블록안에서만 인스턴스 객체 d가 존재

foo()
```

메서드 확장

- 정적(static) □ □ □
  - □ □ □ □ □ □ □ □ □ □ □ □, □ □ □ □ □ □
    직접 호출할 수 있는 메소드
    - □ □ □ □ □ □ □ □ □ □ □ self □ □ □ □ □ □ □ □ □ □
  - □ □
    - <□ □ □ □ □ □ □ □> = staticmethod(□ □ □ □ □ □ □ □ □ □ □ □)

- □ □ □ (class) □ □ □
  - 클래스 영역의 데이터에 직접 접근할 수 있는 메소드
    - 암시적으로 첫 인자로 클래스 객체가 전달
  - □ □
    - <□ □ □ □ □ □ □ □> = classmethod(□ □ □ □ □ □ □ □ □ □ □ □)

연산자 중복

- 연산자 □□□□
  - □□□ □□ □□□□ □□□ □□□□ □□ □□□ □□□ □□□
    □□□□□ □□□
  - 연산자 중복을 위하여 두 개의 밑줄 문자가 앞뒤로 있는
    □□□□ □□ □□□
  - □□ □□


```
class GString:
    def __init__(self, init = None):
        self.content = init

    def __sub__(self, str): # '-' 연산자 중복 정의
        for i in str:
            self.content = self.content.replace(i, '')
        return GString(self.content)

    def Remove(self, str):
        return self.__sub__(str)
```

134

연산자 중복 계속

- 수치 □ □ □

메소드	연산자	사용 예
__add__(self, other)	+ (이항)	A + B, A += B
__sub__(self, other)	- (이항)	A - B, A -= B
__mul__(self, other)	*	A * B, A *= B
__truediv__(self, other)	/	A / B, A /= B (3 이상 지원, 그 이하는 버전에서는 __div__가 사용)
__floordiv__(self, other)	//	A // B, A //= B
__mod__(self, other)	%	A % B, A %= B
__divmod__(self, other)	divmod()	divmod(A, B)
__pow__(self, other[, modulo])	pow(), **	pow(A, B), A ** B
__lshift__(self, other	<<	A << B, A <<= B

– 그 외 다른 메소드는 교제 102 ~ 108 페이지 참조

상속

- 상속이란
  - 부모 클래스의 모든 속성(데이터, 메소드)를 자식 클래스로 물려줌
  - 클래스의 공통된 속성을 부모 클래스에 정의
  - 하위 클래스에서는 특화된 메소드와 데이터를 정의

- 장점
  - 각 클래스마다 동일한 코드가 작성되는 것을 방지
  - 부모 클래스에 공통된 속성을 두어 코드의 유지보수가 용이
  - 각 개별 클래스에 특화된 기능을 공통된 인터페이스로 접근 가능

상속

- 예제 □ □

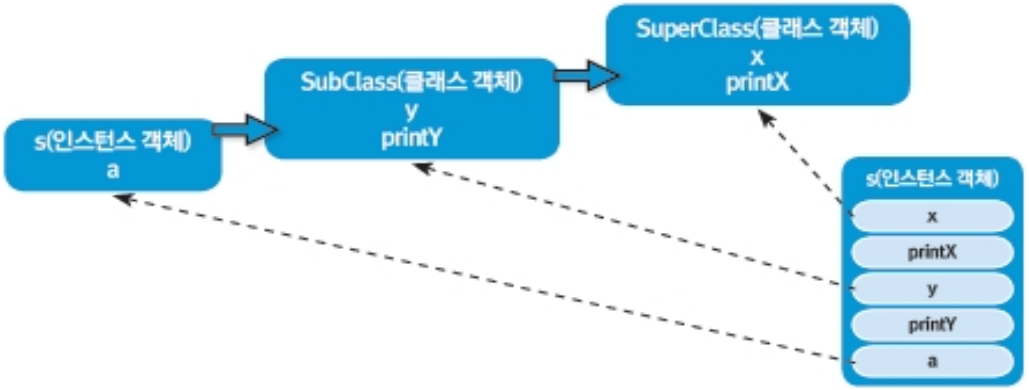
```
class Person:
    def __init__(self, name, phoneNumber):
        self.Name = name
        self.PhoneNumber = phoneNumber

class Student(Person):
    def __init__(self, name, phoneNumber, subject, studentID):
        self.Name = name
        self.PhoneNumber = phoneNumber
        self.Subject = subject
        self.StudentID = studentID
```

  - 그 외 다른 메소드는 교제 102 ~ 108 페이지 참조

상속

- 다중 □ □
  - 2 개 이상의 클래스를 상속받는 경우
  - 두 클래스의 모든 속성 (변수와 메소드)을 전달 받음
  - □ □ □ □

```
# -*- coding: cp949 -*-
class Tiger:
    def Jump(self):
        print("호랑이처럼 멀리 점프하기")

class Lion:
    def Bite(self):
        print("사자처럼 한입에 꿀꺽하기")

class Liger(Tiger, Lion): # 다중 상속
    def Play(self):
        print("라이거만의 사육사와 재미있게 놀기")
```



상속

- 클래스 □ □ □ □ □ □ □
  □ □ □ □ □ □ □
  -> □ □ □ □ □ □ □ □ □ □ □ □ □ □(□ □ □ □ □ □
  -> □ □ □ □ □ □ □)
  -> □ □ □ □

  – □ □ □ □

```
class SuperClass: # 부모 클래스
    x = 10
    def printX(self):
        print(self.x)

class SubClass(SuperClass): # 자식 클래스
    y = 20
    def printY(self):
        print(self.y)

s = SubClass()
s.a = 30
```

  – 그 외 다른 메소드는 교제 102 ~ 108 페이지 참조

모듈

- 모듈 사용하기
- 모듈 만들기
- 모듈의 경로
- 모듈 임포트
- 모듈 임포트 파헤치기
- 유용한 팁
- 패키지

모듈 사용하기

- 여러 코드를 한데 묶어 다른 곳에서 재사용할 수 있는 코드 모음

- □□ □□□ 3.0 □□□□□ □□ 200□□ □□ □□□ □□
  - □□□(string), □□(date), □□(time), □□□(decimal), □□(random)
  - □□(file), os, sqlite3, sys, xml, email, http □□

- 간단하게 모듈을 사용 할 수 있음

- □□□ □□□□ □□

  - 코드의 재 사용성

  - 코드를 이름공간으로 구분하고 관리 할 수 있음 .

  - 복잡하고 어려운 기능을 포함하는 프로그램을 간단하게 만들 수 있다 .

모듈 import

- import : 모듈을 □ □ □ □ □ □ □ □ □ □ □ □ □ □

```
>>> import math
>>> math.pow(2, 10)
1024.0
>>> math.pi
3.1415926535897931
```

- math 모듈은 삼각함수 , 제곱근 , 로그함수 등 수학과 관련된 기능이 들어 있는 □ □ □ □.
- dir() 함수를 이용해 모듈에 어떠한 함수 혹은 데이터가 들어 있는지 알 수 있다.

```
>>>import math
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh',
'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh',
'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor',
'fmod', 'frexp', 'fsum', 'gamma', 'hypot', 'isfinite', 'isinf', 'isnan', 'ldexp',
'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'pi', 'pow', 'radians', 'sin',
'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
>>>
```

간단한 FTP 프로그램

- FTP로 □□□ □□□ □□ □□□□ □□□□ □□□□.

```
>>> from ftplib import FTP
>>> ftp = FTP('ftp.cwi.nl')
>>> ftp.login()
'230 Login successful.'
>>> ftp.retrlines('LIST')
drwxrwsr-x 2 ftp ftp 512 Jan 15 2001 DIENST
drwxr-xr-x 2 ftp ftp 512 Nov 16 2004 incoming
-rw-r--r-- 1 ftp ftp 1810 Jul 05 2004 incoming.readme
lrwxrwxrwx 1 ftp ftp 1 Nov 14 2004 people -> .
dr-xr-xr-x 75 ftp ftp 1536 Feb 04 07:16 pub
drwxrwsr-x 10 ftp ftp 512 Nov 09 2001 sigchi
-r--r--r-- 1 ftp ftp 2195 May 30 1995 welcome.msg
'226 Directory send OK.'
>>>ftp.quit()
'221 Goodbye.'
```

모듈 만들기

- 사용자가 직접 모듈을 만들 수 있다 .
- 큰 프로젝트의 경우 모듈 단위로 일을 진행 하기도 함 .
- □ □ □ □ □ □ □ □ <□ □ □ □ >.py □ □ □ □ □ □ □ .

Simpleset 모듈 만들기

- 텍스트 에디터를 이용해 교집합 , 차집합 , 합집합 함수를 만 □ □ □.

- simpleset.py 이름으로 저장하고 simpleset.py를 파이썬 □ □ □ □ □ □ □ □ □ □ □ □ □ □ □.

- import □ □ □ □ □ □ simpleset □ □ □ □ □ □ □ □ □.

```
>>> import simpleset
```

- □ □ □ □ □ □

```
>>> setA = [1, 3, 7, 10]
>>> setB = [2, 3, 4, 9]
>>> simpleset.union(setA, setB)
[1, 3, 7, 10, 2, 4, 9]
```

모듈의 경로

- 모듈을 임포트 했을 때 모듈의 위치를 검색하는 경로
  - sys.path 에 저장되어 있는 디렉토리를 검색한다 .
- 모듈의 경로 밖의 모듈은 임포트 할 수 없음
- □□ □□ □□ □□.
  - □□□□□ □□□ □□□□
  - PYTHONPATH □□ □□□ □□□ □□
  - □□ □□□□□ □□□□

모듈 import

- 모듈 □□ □□□□□□ (□□, □□□)□□ □□□□□□ □□□□ □□ □□.
- Import □□□ □□□□□□ □□ □□ □□.
  - 함수 , 제어문 내부에서도 import 를 할 수 있다 .
- Import <*모듈이름*>
  - 기본적인 임포트 방법
  - *모듈.이름* 형식으로 모듈 안의 데이터나 함수를 사용 할 수 있다.
  - 모듈은 임포트 하는 방법은 *import 모듈이름* 방법 이외에도 다른 방법이 있다.

모듈 import 방법

- from <모듈> import <□ □ □ □ □>

```
>>> from simpleset import union
>>> union([1, 2, 3], [3], [3, 4])
[1, 2, 3, 4]
```

- from <□ □> import *

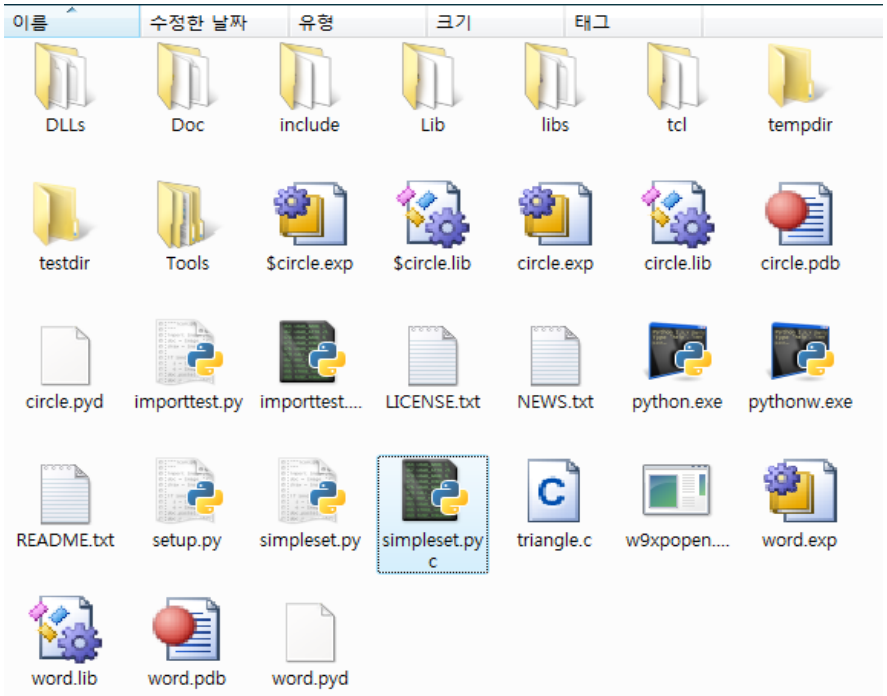
- import <□ □> as <□ □>
  - <□ □> □ □ □ <□ □>□ □ □ □ □ □ □ □ □ □ □ □.

모듈 import 파헤치기

- 임포트를 할 때 , 해당 모듈의 바이트 코드가 있으면 이를 임포트 한다 .
- □□□ □□□ □□ □□ □□□ □□□ □□□□.
- □□□ □□□□□ □□□□ □□ □□□ □□□□ □□□□□□ □□□ □□ □□□□ □□□ □□□ □□□□ □□□.

바이트 코드

- 일종의 □□ □□
  - □□□ □□□□ □□□ □□□ □□
  - □□□ □□□ □□ □□□ :
    □□□ □□□□□(Interpreting) □□ □□ □□ □□□
    □□ □□
  - □□□ □□□ □□□ :
    □□□ □□□□□ □□ □□□ □□□ □□
  - □□□ □□□ □□□ □□



모듈이 메모리에 로딩될 때

- 모듈의 코드가 실행 됨

```
# -*- coding: cp949 -*-
print("test module")
defaultvalue = 1
def printDefaultValue():
    print(defaultvalue)
```

- 처음 임포트 할 때 “print” 구문이 실행된다 .

- 한 번 메모리에 로딩된 모듈은 끝나기 전 까지 변하지 않는다 .

유용한 팁

- 모듈이 직접 실행 혹은 다른 곳에서 임포트 되었는지를 구분 해 줄 수 있는 __name__ □□□□□
  - 모듈이 임포트 되었을 때 __name__은 모듈 자기 자신의 이름
  - 모듈이 직접 실행 되었을 때 __name__은 "__main__"

```
# -*- coding: cp949 -*-
print("my module")


if __name__ == '__main__':
    print("모듈을 직접 실행하셨네요")
else:
    print("임포트 하셨네요")
```

패키지

- 모듈의 □ □
  - 파이썬의 모듈 이름공간을 구조화 하는 한 방법
  - 파이썬 내장 라이브러리 중 XML 패키지의 디렉터리 구조

```
|--- __init__.py
    +--- dom
          +--- __init__.py
          +--- domreg.py
          +--- expatbuilder.py
          …
    +--- etree
          +--- __init__.py
          +--- cElementTree.py
          …
    +--- parsers
          +--- __init__.py
          +--- expat.py
```

예외처리

- 예외(Exception)란?
- 구문 에러
- 예외의 몇가지 종류
- 내장예외 클래스 계층구조
- 주요 내장 예외
- 예외 처리
- raise 구문
- 사용자 정의 예외
- assert 구문

예외(Exception) 란 ?

- 프로그램의 □□ □□□ □□□□ □□ □□□□ □□□
  - 처리를 하지 않는 예 외 에 대하여 자동으로 에러 (Error) 가 발생하고 □□□□□ □□

```
>>> a = [10, 20, 30]
>>> a[3]   # 리스트의 크기를 넘어서는 인덱스 참조
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    a[3]   # 리스트의 크기를 넘어서는 인덱스 참조
IndexError: list index out of range
```

  - 처리되지 않은 예 외 (Unhandled Exception)
    - '0'□□ □□□ □□
    - □□□ □□ □□□ □□□ □□□ □□□□ □□ □□
    - □□□ □□□□ □□□□ □□□ □□□ □□

구문에러

- 오타, □ □ □ □ □ □ □ □ □ □
- □
  □

```
>>> print("{0}, {1}.format(10, 20))
  File "<stdin>", line 1
    print("{0}, {1}.format(10, 20))
                                  ^
SyntaxError: EOL while scanning string literal
>>> if a > 5 print('a is bigger than 5')
  File "<stdin>", line 1
    if a > 5 print('a is bigger than 5')
                 ^
SyntaxError: invalid syntax
>>>
```

예외의 종류 1

- NameError
  - 선언하지 □ □ □ □ 'a'□ □ □

```
>>> print(a)
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    print(a)
NameError: name 'a' is not defined
```

- ZeroDivisionError
  - '0'□ □ □ □

```
>>> 10 / 0
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    10 / 0
ZeroDivisionError: int division or modulo by zero
```

예외의 종류 2

- IndexError
  - 리스트의 □□ □□□ □□□□ □□

```
>>> a = [10, 20, 30]
>>> a[3]
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    a[3]
IndexError: list index out of range
```

- TypeError
  - □□□□ □□ □□(□□□ □□□□ □□)

```
>>> a = 'Apple'
>>> 10 / a
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    10 / a
TypeError: unsupported operand type(s) for /: 'int' and 'str'
```

내장 예외 클래스 계층구조

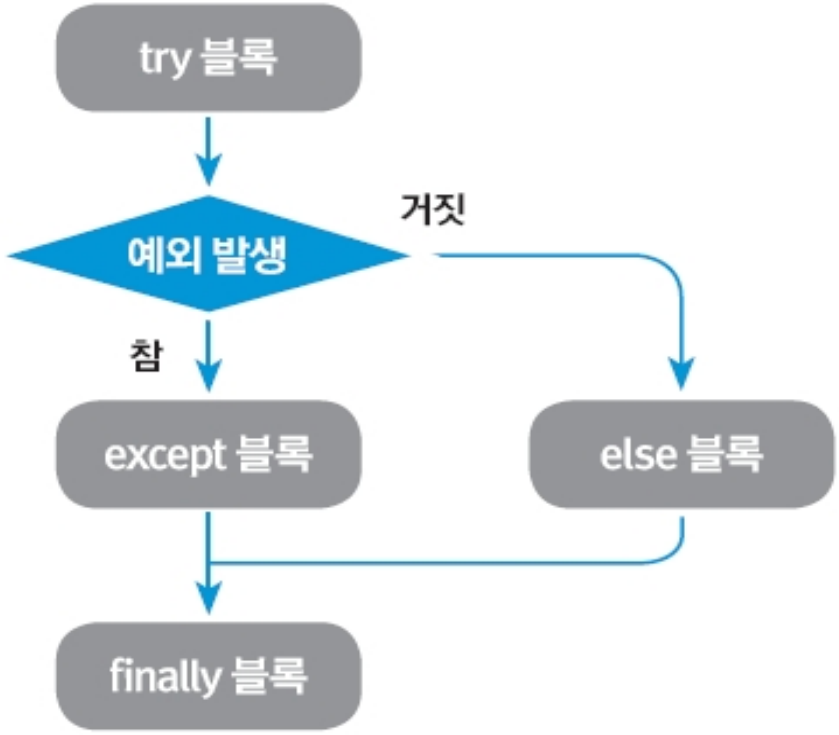
- 내장 □ □ □ exceptions □ □ □ □ □ □ □
  - □ □ □ □ □ □ □ □ □ □ □ □ □ □
  - □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EnvironmentError
      |    +-- IOError
      |    +-- OSError
      |         +-- WindowsError (Windows)
      |         +-- VMSError (VMS)
      +-- EOFError
      +-- ImportError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
      +-- ReferenceError
      +-- RuntimeError
      |    +-- NotImplementedError
      +-- SyntaxError
      |    +-- IndentationError
      |         +-- TabError
      +-- SystemError
      +-- TypeError
      +-- ValueError
      |    +-- UnicodeError
      |         +-- UnicodeDecodeError
      |         +-- UnicodeEncodeError
      |         +-- UnicodeTranslateError
      +-- Warning
           +-- DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
           +-- ImportWarning
           +-- UnicodeWarning
           +-- BytesWarning
```

주요 내장 에러

클래스 이름	내용
Exception	모든 내장 예외의 기본 클래스 - 사용자 정의 예외를 작성시 활용
ArithmeticError	수치 연산 예외의 기본 클래스
LookupError	시퀀스 관련 예외의 기본 클래스
EnvironmentError	파이썬 외부 에러의 기본 클래스

- 참고 □□
  - □□□ □□□□ □□□ 3, 148~149 □□□
  - □□□ □□ □□, Built-in Exceptions □□



예외처리 - 1

- try 구문

```
try:
    <예 외 발생가능성이 있는 문장          >
except <예 외 종류  > :
    <예 외 처리 문장    >
except (예 외  1, 예 외  2):
    <예 외 처리 문장    >
except 예 외  as 인자 :
    <예 외 처리 문장    >
else:
    <예 외 가 발생하지 않은 경우        , 수행할 문장    >
finally:
    <예 외 발생 유무에 상관없이        try 블록 이후 수행할 문장        >
```

예외처리 - 2

- try ~ except □ □

```
def divide(a, b):
    return a / b

try:
    c = divide(5, 0)
except:
    print('Exception is occured!!')
```

  - □□ □□

```
Exception is occured!!
```

예외처리 - 3

- 예제 7-3-1 (p151)

```
# -*- coding: cp949 -*-
def divide(a, b):
    return a / b

try:
    c = divide(5, 'string')
except ZeroDivisionError:
    print('두번째 인자는 0이면 안됩니다.')
except TypeError:
    print('모든 인수는 숫자이어야 합니다.')
except:
    print('음~ 무슨 에러인지 모르겠어요!!')
```

– □□ □□

```
>>>
모든 인수는 숫자이어야 합니다.
```

예외처리 - 4

- 예제 7-3-3 (p153)

```
# -*- coding: cp949 -*-
def divide(a, b):
    return a / b

try:
    c = divide(5, 2)
except ZeroDivisionError:
    print('두번째 인자는 0이면 안됩니다.')
except TypeError:
    print('모든 인수는 숫자이여야 합니다.')
except:
    print('ZeroDivisionError, TypeError를 제외한 다른 에러')
else:# 예외가 발생하지 않는 경우
    print('Result: {0}'.format(c))
finally: # 예외 발생 유무와 상관없이 수행
    print('항상 finally 블록은 수행됩니다.')
```

```
Result: 2.5
항상 finally 블록은 수행됩니다.
```

예외처리 - 5

- 예제 7-3-4 (p154)

```
# -*- coding: cp949 -*-
def divide(a, b):
    return a / b

try:
    c = divide(5, "af")
except TypeError as e: # 전달되는 예외 인스턴스 객체를 e로 받아서 사용
    print('에러: ', e.args[0])
except Exception:
    print('음~ 무슨 에러인지 모르겠어요!!')
```

- □□ □□

```
에러:  unsupported operand type(s) for /: 'int' and 'str'
```

예외처리 - 6

- 예제 7-3-5 (p154)

```
# -*- coding: cp949 -*-
def divide(a, b):
    return a / b

try:
    c = divide(5, 0)
except (ZeroDivisionError, OverflowError, FloatingPointError):
    # 명시된 에러를 모두 처리
    print('수치 연산 관련 에러입니다.')
except TypeError:
    print('모든 인수는 숫자이어야 합니다.')
except Exception:
    print('음~ 무슨 에러인지 모르겠어요!!')
```

– □□ □□

수치 연산 관련 에러입니다.

예외처리 - 7

- 예제 7-3-6 (p155)

```
# -*- coding: cp949 -*-
def divide(a, b):
    return a / b

try:
    c = divide(5, 0)
except ArithmeticError:
    #상위 클래스를 처리시 하위 모든
    # 클래스도 이 부분에서 처리
    print('수치 연산 관련 에러입니다.')
except TypeError:
    print('모든 인수는 숫자이어야 합니다.')
except Exception:
    print('음~ 무슨 에러인지 모르겠어요!!')
```

예외처리 - 8

- try ~ finally

  try:

  <예 외 발생 가능성이 있는 문장 >

  finally:

  <예 외 와 관계없이 , 항상 수행되어야 할 문장 >

  – 위와 같은 구조인 경우 , finally는 생략 불가

  – □□ 7-3-7

```
# -*- coding: cp949 -*-
FilePath = './test.txt'

try:
    f = open(FilePath, 'r')
    try:
        data = f.read(128)
        print(data)
    finally:
        f.close()
except IOError:
    print("Fail to open {0} file".format(FilePath))
```

raise 구문

- 명시적으로 □ □ □ □
- raise □ □ □ □
  - raise [Exception]
  - raise [Exception(data)]
  - raise

- □ □ 7-4-1

```
# -*- coding: cp949 -*-
def RaiseErrorFunc():
    raise NameError # 내장 예외인 NameError를 발생

try:
    RaiseErrorFunc()
except:
    print("NameError is Catched")
```

사용자 정의 예외 - 1

- 내장 예 외 만으로 부족한 경우 , 개발자가 직접 예 외 를 정의
  □□ □□ □□
  - Exception 클래스나 그 하위 클래스를 상속받아서 구현
  - □□□□ □□□ □□ □□□ □□□□ □□□
    □□ □□□□ □□

- □□ 7-5-1

```
class NegativeDivisionError(Exception): # 사용자 정의 예외 정의
    def __init__(self, value):
        self.value = value
```

사용자 정의 예외 - 2

- 예제 7-5-1

```
def PositiveDivide(a, b):
    if(b < 0):
        raise NegativeDivisionError(b)
    return a / b

try:
    ret = PositiveDivide(10, -3)
    print('10 / 3 = {0}'.format(ret))
except NegativeDivisionError as e: # 사용자 정의 예외인 경우
    print('Error - Second argument of PositiveDivide is ', e.value)
except ZeroDivisionError as e: # '0'으로 나누는 경우
    print('Error - ', e.args[0])
except: # 그 외 모든 예외의 경우
    print(e.args)
```

- □□ □□

```
Error - Second argument of PositiveDivide is  -3
```

assert 구문 - 1

- 표현식
  - Assert <조건식>, <관련 데이터>
- 인자로 받은 조건식이 거짓인 경우 , AssertionError 가 발생
- □□□□□□ □□□, □□ □□ □□□□□ □□
- __debug__ 가 True 인 경우만 assert 구문 활성화
  - 명령 프롬프트에서 최적화 옵션 (-O) 을 설정하면 __debug__ 는 False 로 설정됨
  - □□ □□□ □□

```
if __debug__:
    if not <조건식>:
        raise AssertionError(<관련 데이터>)
```

assert 구문 - 2

- 예제 7-6-1

```
# -*- coding: cp949 -*-
def foo(x): # 받은 인자의 type이 정수형인지 검사
    assert type(x) == int, "Input value must be integer"
    return x * 10

ret = foo("a") # AssertionError가 발생
print(ret)
```

- □□ □□

```
C:\Python31>python.exe 7-6-1.py
Traceback (most recent call last):
  File "7-6-1.py", line 6, in <module>
    ret = foo("a") # AssertionError가 발생
  File "7-6-1.py", line 3, in foo
    assert type(x) == int, "Input value must be integer"
AssertionError: Input value must be integer

C:\Python31>python.exe -O 7-6-1.py
aaaaaaaaaa
```

입 출력

출력

- □□□□ □□□ □□ print() □□□ □□□□□. □□□ □□ 2.x 때는 print 가 함수가 아니었지만 , 3.0 에서는 함수로 □□□□□□.
- 즉 , 다음과 같이 함수처럼 괄호 안에 출력할 인자를 적으면 □□□.

```
>>> print(1)
1
>>> print('hi, guyz')
hi, guyz
```

- Print □□□ □□□□□ □□□ □□ □□□(sep), 끝라인 (end), 출력 (file)을 지정해 줄 수 있습니다 . 아래 예제와 같이 file 을 이용해서 출력을 표준오류 (standard error)로 □□□□□ □□□ □□ □□ □□□□.

```
>>> print("welcome to","python", sep="~", end="!", file=sys.stderr)
welcome to~python!
```

포맷팅(Formatting)

- Print만 가지고는 출력이 좀 불편하다고 느끼실 것입니다. 그러나 format() 메소드을 사용하면 문자열을 그 이상으로 자유롭게 다룰 수가 있습니다.

- 문자열 내에서 어떤 값이 들어가길 원하는 곳은 {}로 표시를 합니다. {} 안의 값들은 숫자로 표현할 수 있으며, format 인자들의 인덱스를 사용합니다.

- 아래 예제를 보면, {0}는 첫 번째 인자인 "apple"을 나타내고 {1}은 두 번째 인자인 "red"를 나타냅니다.
  - >>> print("{0} is {1}".format("apple", "red"))
    apple is red

포맷팅(Formatting)

- { } 안의 값을 지정할 때는 다음과 같이 format 의 인자로 키 (key) 와 값 (value) 을 주어 위와 동일한 결과를 얻을 수 있습니다 .
  - >>> print("{item} is {color}".format(item="apple", color="red"))
    apple is red

- 이번에는 dictionary 를 입력으로 받는 경우를 살펴보도록 하겠습니다 .
  - >>> dic = {"item":"apple", "color":"red"}
  - >>> print("{0[item]} is {0[color]}".format(dic))
    apple is red

포맷팅(Formatting)

- 앞에서 {0[item]}의 0[ ]을 달고 다니느라 매우 불편하셨을 텐데요 , ** 기호를 사용하면 dictionary를 입력으로 받은 것으로 판단하고 인자를 하나 만 받게 됩니다 . 그러므로 아래와 같이 불필요한 index는 생략할 수 있습니다 .
  - >>> print("{item} is {color}".format(**dic))
    apple is red

포맷팅(Formatting)

- ! 기호를 사용해서 문자열 변환을 사용할 수 있습니다 .
  - >>> print("{item!s} is {color!s}".format(**dic))
    apple is red
  - >>> print("{item!r} is {color!r}".format(**dic))
    'apple' is 'red'
  - >>> print("{item!a} is {color!a}".format(**dic))
    'apple' is 'red'
- !s, !r, !a 는 각각 str(), repr(), ascii() 를 실행한 결과와 동일하다고 보시면 됩 □ □ .

포맷팅(Formatting)

- 이번에는 “:” 기호를 이용하여 보다 정교하게 정렬 , 폭 , 부호 , 공백처리 , 소수점 , 타입 등을 지정할 수 있는 법을 알아보겠습니다 .
  - >>> print("{0:$>5}".format(10))
    $$$10
- “{0" 까지는 첫 번째 인자인 ‘10’을 표시하라는 뜻인 것을 알 수 있습니다 . 그런데 ":" 이후의 기호들은 모두 처음 보는 것들이네요 . 눈치가 빠르신 분들은 이미 파악하셨겠지만 , "$" 는 공백을 채우는 문자를 "$" 로 채우라는 의미이며 , ">" 는 오른쪽 정렬을 , "5" 는 전체 자리수가 5 자리라는 것을 나타냅니다 .

포맷팅(Formatting)

- 정렬에 사용되는 기호는 ">", "<", "^", "=" 이 있습니다 .
  ">"는 오른쪽 기준 ,
  "<"는 왼쪽 기준 ,
  "^"는 가운데 기준을 나타냅니다 .
  "="는 부호와 상관이 있는데 , "="가 사용되면 공백문자들 앞에 부호가 표시됩니다 . 사용되지 않으면 공백문자들 뒤 , 즉 , 숫자 바로 앞에 부호가 표시됩니다 .
- 부호를 나타내는 기호는 "+", "-", " "이 있습니다 .
  "+"는 플러스 부호를 나타내라는 뜻이며 ,
  "-"는 마이너스 값만 마이너스 부호를 나타내라는 것을 의미하며 ,
  " "는 마이너스 값에는 마이너스 부호를 나타내고 플러스일 때는 공백을 표시하 □□ □□□□.

포맷팅(Formatting)

- 진수를 □□□ □□□ □□ □□□□.
  "b"는 이진수를 , "d"는 십진수를 , "x"는 16 진수를 , "o"는 8 진수를 나타내며 "c"는 문자열을 출력합니다 .
  - >>> print("{0:b}".format(10))
    1010
  - >>> print("{0:o}".format(10))
    12
  - >>> print("{0:c}".format(65))
- 또한 "#"를 사용하면 #x는 16 진수 #o는 8 진수 , #b는 2 진수로 표시 됩니다 .
  - >>> print("{0:#x}, {0:#o}, {0:#b}".format(10))
    0xa, 0o12, 0b1010

포맷팅(Formatting)

- 정수 이 외 에 실수에 대한 변환도 제공되며 , "e" 는 지수표현을 , "f" 는 일반적인 실수 표현을 , "% " 는 퍼센트 표현을 의미합니다 .
  - >>> print("{0:e}".format(4 / 3))
    1.333333e+00
  - >>> print("{0:f}".format(4 / 3))
    1.333333
  - >>> print("{0:% }".format(4 / 3))
    133.333333%
- 실수에서는 다음과 같이 소수점 몇 번째 자리까지 표현할 것인지를 지정할 수 있습니다 . 아래 예제에서는 소수점 3 번째 자리까지만 출력되는 것을 확인할 수 □ □ □ □ .
  - >>> print("{0:.3f}".format(4 / 3))
    1.333

입력

- 사용자로부터 입력은 다음과 같이 input() 함수를 이용해서 받을 수 있습니다 .
- input 의 입력인자로는 화면에 출력할 프롬프트 (prompt) 를 줄 수 있으며 , 생략 □□□□, □□□□□□ □□□ □□□ □□□□□.

  – >>> a = input('insert any keys :')
    insert any keys : *test*
  – >>> print(a)
    test

파일 입출력

- 파일로의 입출력은 앞에서 본 바와 같이 print() 함수의 file 입력인자를 이용할 □□ □□□, □□ □□□ □□□ □□ □□□□ □□ □□□□ open() □□□ 통해서 파일을 연 후 파일 전용 함수들을 이용해서 작업하는 것이 일반적입니다 .
  - >>> f = open('test.txt', 'w')
- Open 함수의 마지막 인자는 파일을 열 때의 속성을 의미하며 , 다음 속성들의 □□□□ □□□ □□□□□.
  - r : 읽기 모드 (디폴트)
  - w : 쓰기 모드
  - a : 쓰기 + 이어쓰기 모드
  - + : 읽기 + 쓰기 모드
  - b : 바이너리 모드
  - t : 텍스트 모드 (디폴트)

파일 입출력

- 파일로부터 읽고 쓰기 위해서 파일로부터 모든 데이터를 읽는 read() 함수와 문자열을 쓰는 write() 함수가 제공됩니다 . 또한 파일을 열고 할 일을 모두 완료했을 경우 파일객체를 닫아주는 close() 함수가 있습니다 .
  - >>> f = open('test.txt', 'w')
  - >>> f.write('plow deep\ nwhile sluggards sleep')
    31
  - >>> f.close()

파일 입출력

- 이번에는 이 파일이 제대로 쓰였는지 읽어 보도록 하겠습니다 .
  - >>> f = open('test.txt')
  - >>> f.read()
    'plow deep\ nwhile sluggards sleep'
  - >>> f.close()
  - >>> f.closed
    True

- 텍스트 모드에서는 일반 문자열과 같이 encoding 이 적용되기 때문에 , 바이너리 데이터 (binary data) 를 다룰 때에는 오류가 발생하게 됩니다 . 그러므로 바이 □□ □□□□ □□ □□□ □□□ □□□□ □□□ □□□□ □□□.

파일 입출력

- 파일 □□□ □□ □□□
  - readline() 함수는 호출할 때 마다 한 줄씩 읽은 문자열을 반환하며 ,
    readlines() 함수는 파일의 모든 내용을 줄 단위로 잘라서 리스트를 반환합니다 .
  - tell() □□□ □□ □□□□ □□□□ □□ □□□ □□□ □□□□,
    seek() □□□ □□□□ □□□ □□□ □□□□ □□□□□.
- With ~ as □□
  -
    ```
    >>> with open('test.txt') as f:
    [TAB]print( f.readlines() )
    [TAB]print( f.closed )
    ```
  -
    ```
    ['plow deep\  n', 'while sluggards sleep']
    False
    ```
  -
    ```
    >>> f.closed
    True
    ```

188

Pickle

- 그런데 문자열의 경우에는 배운 방법을 사용하여 쉽게 다룰 수 있지만 , 리스트나 클래스 등을 저장할 때는 어떻게 해야 할까요 ?
- 내용을 모두 분해해서 파일에 저장한 후 , 다시 읽어 들일 때는 구분해서 다시 값을 설정해야 할까요 ?
- 파이썬에는 이러한 일들을 쉽게 할 수 있도록 도와주는 pickle 이라는 모듈이 있□□□.

Pickle

- 다음과 □□ □□□□ □□□ □□□□□.
  - >>> colors = ['red', 'green', 'black']

- 아래와 같이 pickle 모듈의 dump 함수를 사용하면 , colors 를 다음과 같이 쉽게 파일에 저장할 수 있습니다 .
  - >>> import pickle
  - >>> f = open('colors', 'wb')
  - >>> pickle.dump(colors, f)
  - >>> f.close()

Pickle

- 이번에는 파일로부터 다시 읽어보겠습니다 . 우선 colors를 삭제한 후 , load 함 □□ □□□□ □□□□ □□ □□□□.
  - >>> del colors
  - >>> f = open('colors', 'rb')
  - >>> colors = pickle.load(f)
  - >>> f.close()
  - >>> colors
    ['red', 'green', 'black']

- pickle 로 파일에 쓰거나 읽을 때는 반드시 바이너리 모드로 파일을 열어야 합니다 .

Pickle

- pickle로 저장할 수 있는 대상은 파이썬 객체라면 거의 모두 가능합니다 .
  □□ □□□□ □□□□ □□ □□□ □□ □□□□ □□□ □□□ □□□
  pickle 이 가능합니다 .
  - >>> class test:
    [TAB]var = None
  - >>> a = test()
  - >>> a.var = 'Test'
  - >>> f = open('test', 'wb')
  - >>> pickle.dump(a, f)
  - >>> f.close()
  - >>> f = open('test', 'rb')
  - >>> b = pickle.load(f)
  - >>> f.close()
  - >>> b.var
    'Test'

C/C++ 연동

- 간단한 확장 모듈 예제
- 모듈 초기화
- 모듈 빌드
- 파이썬/C API
- 에러 처리
- 확장 타입
- ctypes

확장 모듈이 필요한 이유

- 새로운 □□ □□ □□□ □□

  - 확장모듈을 이용하여 파이썬에서 C/C++ 라이브러리 함수 혹은 시스템 콜 가능 .

- □□□□ □□□□□□ □□

  - □□ □□□□ □□□ Up

  - 잘하는 것 (문자열처리 , 자료형 , 인터페이스)은 파이썬 , 연산처리는 C/C++ 코드

- □□ □□

  - 파이썬 코드를 C/C++ 모듈이나 C 확장형으로 구현 시 , 작성한 파이썬 코드 (*.py) 들의 □□□□ □□□ □□□□ □□ □□□□

  - □□□□ □□□□□ □□ □□ □□, □□□ □□□□□ □□□

간단한 확장모듈 예제

- 헤더 □□ python.h
- spam 이라는 확장 모듈을 만들면서 파이썬과 C의 연동을 알아보겠습니다 .
  - □□ □□ □□□□ □□□ □□□ □□
  - 파이썬에서 spam 모듈이 아래와 같이 사용 될 것임

```
>>> import spam
>>> strcnt = spam.strlen("test");
>>> strcnt
4
```

확장 모듈 spam 만들기

- Spammodule.c 파일 □ □ □ □
  - 전통적으로 '모듈이름' + module.c 와 같은 형식으로 파일이름 지정
- 첫 줄에 "python.h"를 포함 시킨다 .
  - 헤더파일을 포함할 때 python.h 를 제일 먼저 포함해야 함 .

```
#include <python.h>
```

strlen 함수 만들기

- 파이썬에서 spam.strlen 을 호출 했을 때 실행되는 C 코드 만들기

```
static PyObject *
spam_strlen(PyObject *self, PyObject *args)
{
    char* str;
    int len;
    if (!PyArg_ParseTuple(args, "s", &str))
        return NULL;
    len = strlen(str);
    return Py_BuildValue("i", len);
}
```

- PyObject, PyArg_ParseTuple, Py_BuildValue

모듈 초기화

- 모듈을 임포트 할 때 파이썬 내부에서 다음과 같은 작업이 이루어진다 .
  - 1□□ : □□□ □□□.
  - 2□□ : □□□ □□□ □□.
  - 3 단계 : 지역 이름공간 (Local Namespace)에 이름을 정의한다 .

- □□ □□□ □□□ 2,3 □□□ □□ □□□ □□.

Py_InitModule() 함수

- 모듈을 □ □ □ □ □ □ Py_InitModule() □ □ PyModule_Create() □ □ □ □ □

```
static PyMethodDef SpamMethods[] = {
    {"strlen", spam_strlen, METH_VARARGS, "count a string length."},
    {NULL, NULL, 0, NULL}        ← 배열의 끝을 나타냅니다.
};
static struct PyModuleDef spammodule = {
    PyModuleDef_HEAD_INIT,
    "spam",      ← 모듈 이름
    "It is test module.",    ← 모듈 설명을 적는 부분, 모듈의 __doc__에 저장됩니다
    -1, SpamMethods
};
PyMODINIT_FUNC
PyInit_spam(void)
{
    return PyModule_Create(&spammodule);
}
```

확장 모듈 빌드

- 작성한 □□□□ spammodule.c □□□ □□.

```
from distutils.core import setup, Extension
spam_mod = Extension('spam', sources = ['spammodule.c'])
setup(name = "spam",
    version = "1.0",
    description = "A sample extension module",
    ext_modules = [spam_mod],
)
```

- Distutils 를 이용해서 빌드

```
$>python setup.py install
```

위의 명령은 리눅스에서는 gcc 를 호출하며 , 윈도에서는 visual C++ 컴파일러가 구동되면서 빌드 후 파이썬 디렉토리의 Lib/site-packages 디렉토리에 mylib.so 또는 mylib.pyd 파□□ □□□□.

확장 모듈 spam 테스트

- 생성된 spam.pyd 를 모듈 검색이 가능한 경로에 복사
- □ □ □ □ □ □ □

```
>>> import spam          ← spam
>>> spam.strlen("hello world")
11
```

파이썬 C/API

- 파이썬에서 C□ □ □ □ □ □ □ □ C/API□ □ □ □ □.
  - PyArg_ParseTuple() □ □
  - Py_BuildValue() □ □

PyArg_ParseTuple( ) 함수

- 파이썬에서 C로 전달되는 인수를 C 함수에 맞게 변경
  - □□ □□ (Prototype)

```
int PyArg_ParseTuple(PyObject *arg, char *format,...);
```

- □□□□ □□ □□

파이썬 객체 타입	기호	C 자료형
int	i	int
unicode object	s	const char *
float	f	float
int	l	long
bytes object	y	const char *
unicode object	u	Py_UNICODE
object	O, S	PyObject

Py_BuildValue( ) 함수

- Py_BuildValue는 C의 자료형을 파이썬의 자료형으로 변환
  - □ □ □ □ (Prototype)


```
PyObject *Py_BuildValue(char *format, ...);
```

- □ □ □ □


```
Py_BuildValue("")                           ←— None
Py_BuildValue("i", 123)                     ←— 123
Py_BuildValue("iii", 123, 456, 789)         ←— (123, 456, 789)
Py_BuildValue("s", "hello")                 ←— 'hello'
Py_BuildValue("y", "hello")                 ←— b'hello'
```

204

예외처리

- 파이썬 □□□□ □□□ □□□ □□□□□ □□□□□.
- C □□ □□□□□ □□□ □□□ □□□ □□□ □□□ □□□ □□.
  - 에러가 발생했을 때 에러의 상황을 파이썬에 직접 알려 줘야 한다 .
- 파이썬 C/API 함수들은 특별히 에 외 처리를 하지 않아도 된다 .

```
>>> import spam
>>> spam.strlen(1)                ← 문자열 대신 숫자를 입력합니다.
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: argument 1 must be string, not int
```

에러가 발생했을 때

- 에러가 발생했을 때
  - □□ □□□ □□□□□ □□□
  - NULL을 리턴한다 .

- PyErr_SetString() □□
  - □□□ □□□□ □□ □□□□ □□.
  - □□ □□

```
void PyErr_SetString(PyObject *type, const char *message)
```

  - type 은 에러의 종류를 입력
    - 분수의 제수가 0 일 때 PyExc_ZeroDivisionError
  - message 는 에러에 대한 설명

확장타입

- C를 □□□ □□ □□□ □□
  - 파이썬이 제공하는 패턴을 이용해 타입을 만들면 비교적 쉽게 확장 타입을 만들 수 있다 .
- PyObject_HEAD
  - □□ □□ □□□□□ □□ □□□, □□ □□□ □□□□ □□□□ □□□
- PyTypeObject
  - 파이썬 include 디렉터리의 object.h 에 선언
  - □□ □□ □□□ □□□ □□□ □□□□ □□ □□□
  - 모든 항목을 다 채울 필요는 없고 사용하는 기능만 사용하면 된다 .

ctypes

- 파이썬에서 외 부 라이브러리를 간편하게 사용할 수 있는 내장 모듈
  - C의 데이터 타입이나 , DLL 혹은 shared library의 함수들을 직접 사용할 수 있습니다 .

- ctypes □ □ □ □ □ □ □ □ □ System function □ □ □ □ □ □

```
>>> import ctypes
>>> print(hex(ctypes.windll.kernel32.GetModuleHandleA(None)))
0x1d000000
```

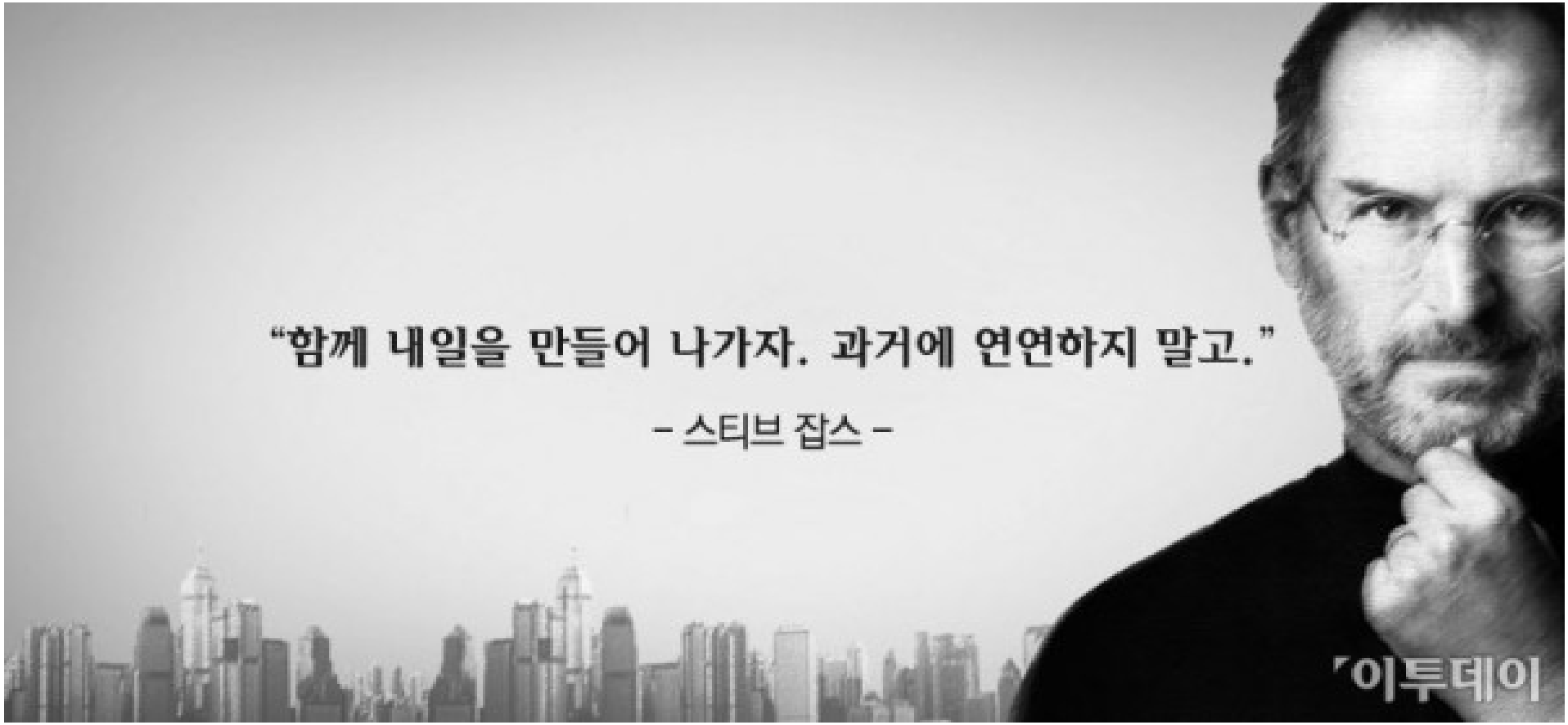
types 로 C 데이터 타입 사용하기

- ctypes가 □ □ □ □ C □ □ □ □ □ □ □

ctypes 타입	C 타입	파이선 타입
c_short	Short	int
c_ushort	unsigned short	int
c_int	Int	int
c_uint	unsigned int	int
c_long	Long	int

- ctypes □ □ □ □ □ □ □ □ □ □ □ □

```
>>> from ctypes import *
>>> i = c_int(10)
>>> print("ctypes : ", i ,", value : ", i.value )
ctypes : c_long(10), value : 10
```


"함께 내일을 만들어 나가자. 과거에 연연하지 말고."
– 스티브 잡스 –
「이투데이