

파이썬

파이썬이란...

- 1991년 귀도 반 로섬(Guido van Rossum)이 발표.
- 인터프리터 언어
- 사용자 층이 점점 넓어지는 중.
- 외국에서 많이 사용중 교육및 실무용으로
- 구글
- 파일 동기화 서비스인 드롭박스(Dropbox)
- 쉽고 빠르게 웹 개발을 할 수 있도록 도와주는 프레임워크인 장고(Django)
- 임베디드 분유
- 통계분야

파이썬의 특징

- 가독성
 - 간결하고 가독성이 좋습니다.
 - 코드블럭을 들여쓰기(indentation)로 구분.
- 풍부한 라이브러리
- 인터프리터 언어이다
- 별도의 변수 선언이 없더라도 사용이 가능하다
- 접착성(C언어와의 결합이 손쉬움)
- 무료
- 유니코드
- 동적타이핑 (런타임시에 동적 타이핑)과 자동으로 메모리 관리

파이썬의 종류

- Cpython : C 언어로 작성된 기본 파이썬
- Jython : 자바로 구현된 파이썬, 자바 가상머신에서 운영된다.
- IronPython : Net과 Mono용 c#으로 구현
- PyPy : 파이썬으로 구성된 파이썬

파이썬의 사용분야

- 시스템 유틸리티 제작
- GUI 프로그래밍
- C/C++와의 결합
- 웹프로그래밍
- 수치연산프로그래밍
- 데이터베이스 프로그래밍
- 데이터분석이나 사물 인터넷
- 크롤링
- 딥러닝 (Tensorflow)
- 웹 프로그래밍(Django)
- 아직 모바일이나 시스템 언어로서의 사용은 불가능

파이썬이 쓰이는 프로젝트들

- 알게 모르게 파이썬이 사용되는 프로젝트들이 많습디다만, 유명한 것들만 예를 들어 보겠습니다.
 - BitTorrent, MoinMoin, Scons, Trac, Yum
 - CherryPy, Django
 - GIMP, Maya, Paint Shop Pro
 - Youtube.com, Google Groups, Google Maps, Gmail

2.x와 3.x의 차이

- print가 함수로 변경.
- long 자료형이 없어지고 int로 통일.
 - 2.x style :

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

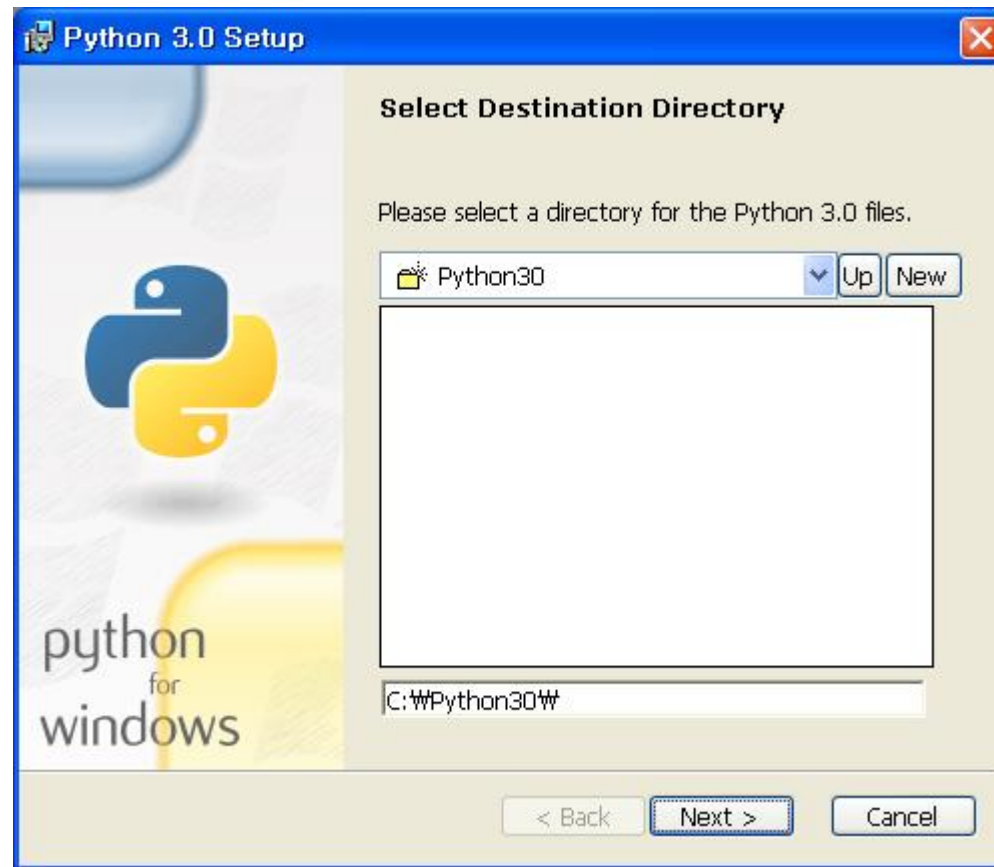
```
>>> type(2**31)
<class 'int'>
>>> type(2**40)
<class 'int'>
```
- 'int / int'의 결과가 float으로 처리.
- String, Unicode 체계 변경.

설치 및 개발환경 1

- <http://python.org/download>



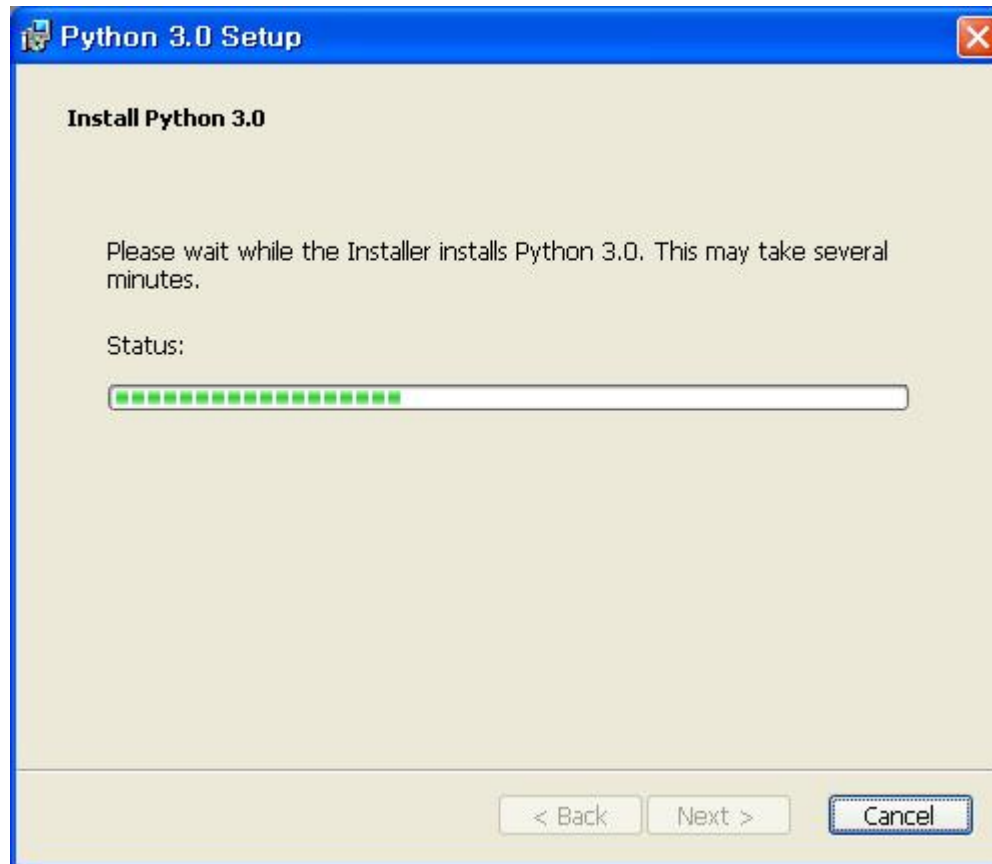
설치 및 개발환경 2



설치 및 개발환경 3



설치 및 개발환경 4

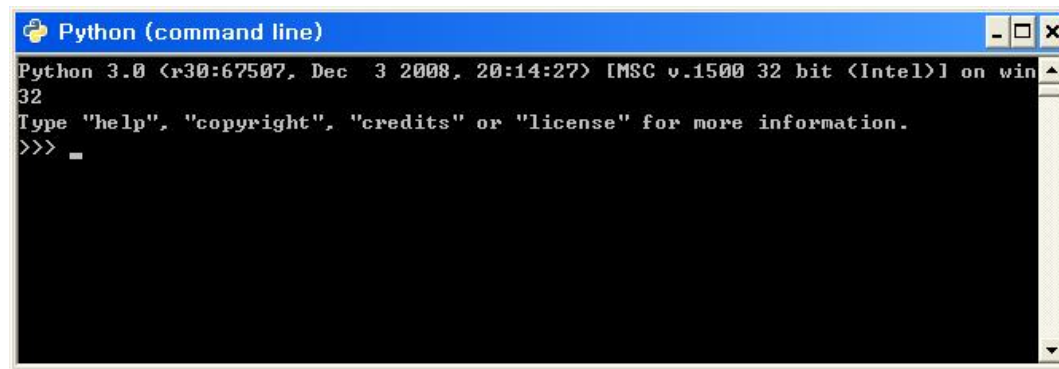


설치 및 개발환경 5



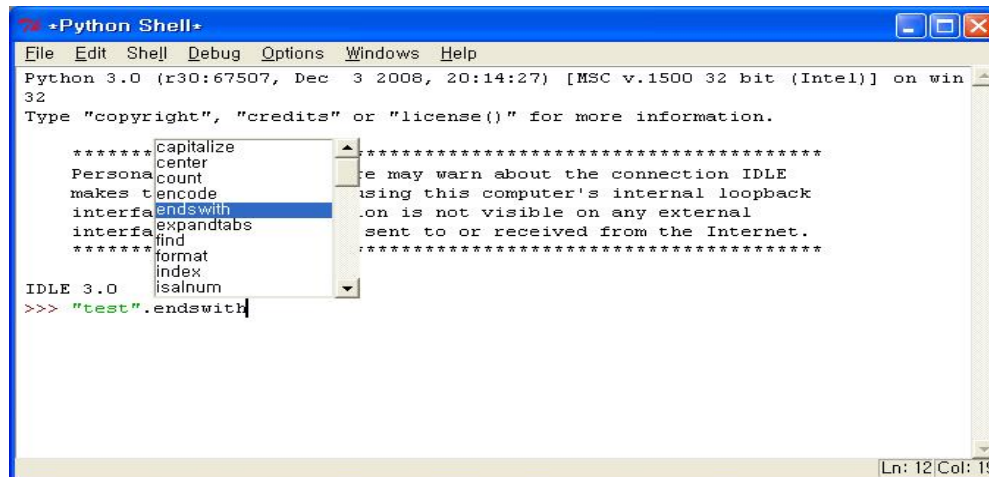
설치 및 개발환경 6

- Python command line



A screenshot of a Windows command prompt window titled "Python (command line)". The window shows the Python 3.0 version information: "Python 3.0 (r30:67507, Dec 3 2008, 20:14:27) [MSC v.1500 32 bit (Intel)] on win32". Below this, it says "Type 'help', 'copyright', 'credits' or 'license' for more information." and the prompt ">>>" is followed by a cursor.

- IDLE



A screenshot of the Python Shell window in IDLE. The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The text area shows the same Python 3.0 version information as the command line window. Below this, it says "Type 'copyright', 'credits' or 'license()' for more information." and the prompt ">>>" is followed by the text "test.endswith". A context menu is open over the text, showing a list of string methods: "capitalize", "center", "count", "encode", "endswith", "expandtabs", "find", "format", "index", and "isalnum". The "endswith" method is highlighted. At the bottom right of the window, it says "Ln: 12 Col: 19".

Hello World

- ```
>>> print("hello world")
hello world
```

# 들여쓰기

- 들여쓰기(indentation)는 파이썬 문법의 가장 큰 특징입니다.
  - 블록을 사용하지 않고 들여쓰기 내쓰기로 각 코드 블록을 구분해 냅니다
  - 가독성을 높이지만, 오류가 일어나지 않도록 조심해야 합니다.
  - 코드블럭1
    - [TAB]코드블럭2
    - 코드블럭3
    - [TAB]코드블럭4

# 소스코드 인코딩

- 파이썬에서는 # 이후는 주석으로 인식합니다.
- 그러나 다음과 같이,  
소스코드 부분에서 사용될 경우,  
소스코드 인코딩을 지정하는 용도로 사용됩니다.

- # coding: latin-1
- # -\*- coding: utf-8 -\*-
- 여러줄 한번에 주석처리하기

"""

이 사이에 코드를 둔다

"""



# 기타 파이썬 문법

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

## 2.X를 3으로 변경하기

- 파이썬 2.x과 파이썬 3과는 호환이 되지 않는다는 것은 큰 단점입니다만, 2.x에서 3로 변경하는 툴을 제공합니다.
  - `C:\Python30\Tools\Scripts>2to3.py -w test.py`

# 개발툴

- Edit plus, note ++, 그밖의 편집기
- Eclipse 에 파이썬 플러그인 설치후 사용 가능
- Visual studio 도 파이썬 개발 도구로 무료버전 express 지원중

# 이클립스와 파이썬

- 자바를 먼저 설치한다
- (자바가 설치되지 않으면 이클립스가 가동되지 않음)
- <http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html>

# 이클립스와 파이썬

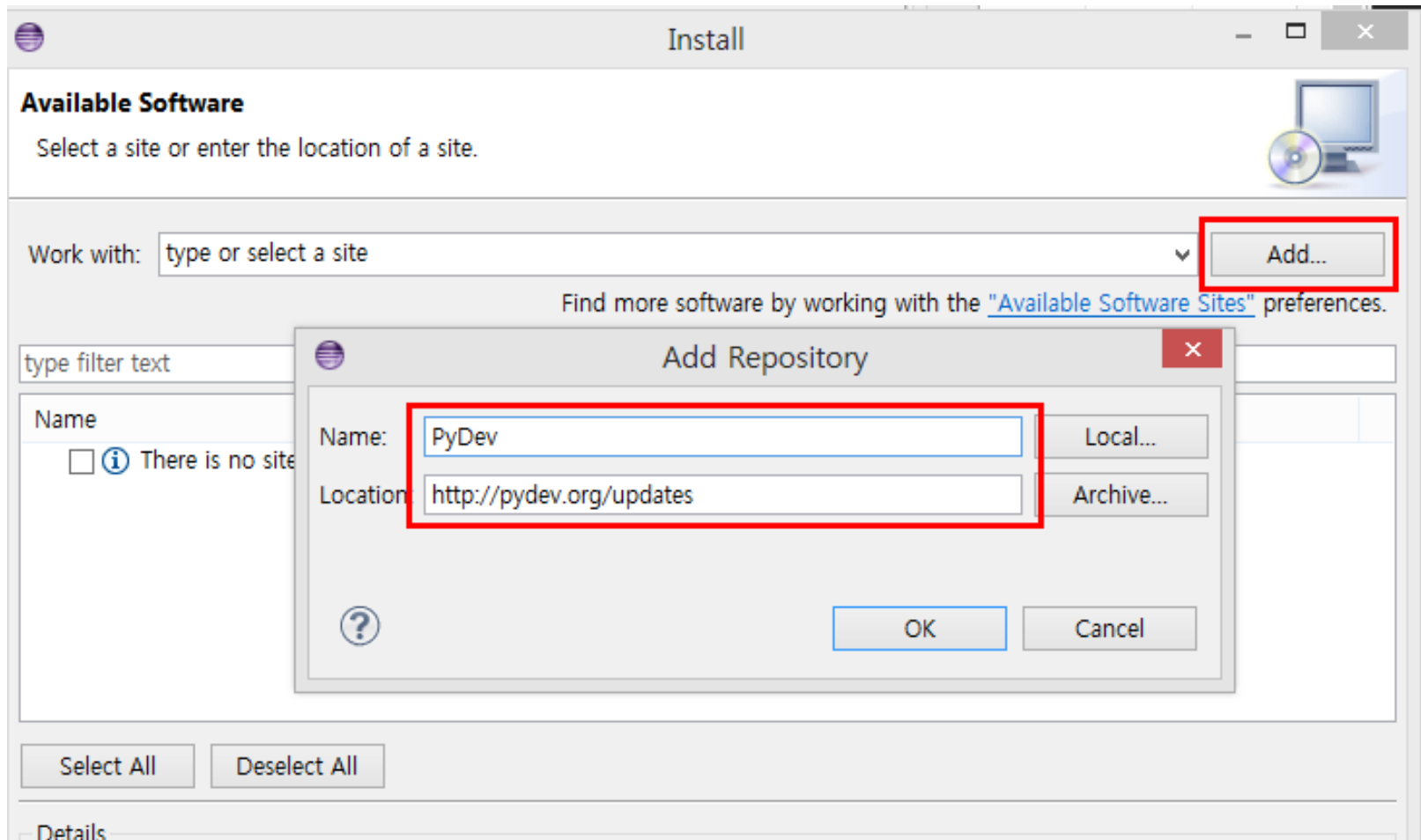
- 이클립스 다운받기
- <http://eclipse.org/downloads/>
- 가장 최신버전을 다운받는다
- 압축만 풀면 설치 완료
  - 설치버전으로 변경되었음
  - 컴퓨터 버전에 맞추어서 다운받는다
  - (32비트용과 64비트용이 있다)
  - C:\Users\계정\workspace\java-mars

# 이클립스와 파이썬

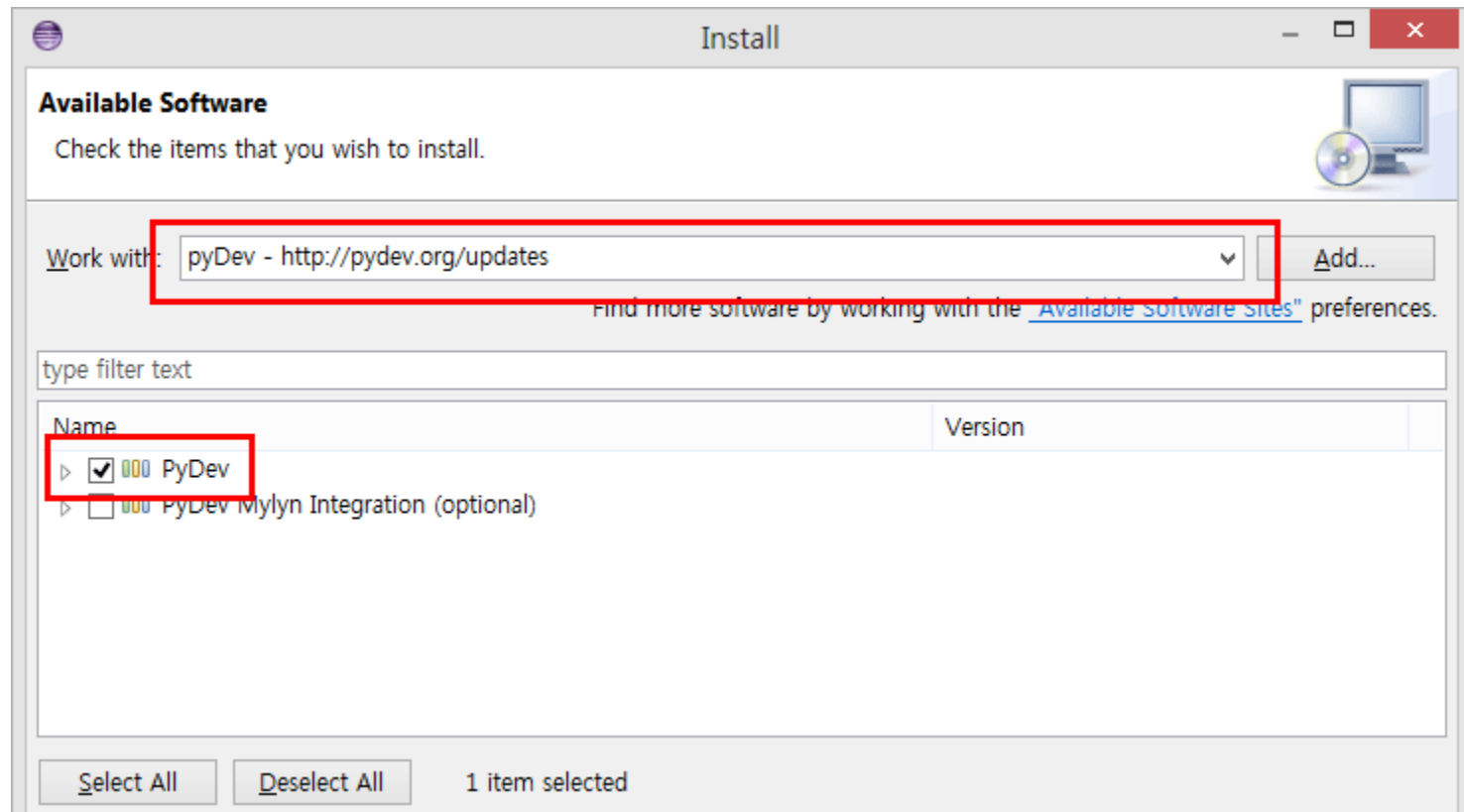
- **PyDev 플러그인 설치**

- 이클립스에서 파이썬 작업이 가능하도록 하는 플러그인을 설치한다
- 이클립스의 메뉴에서 "Help > Install New Software..." 를 선택하세요.
- 새로운 버전의 이클립스에서는 이 방법이 아니고, Eclipse 안에 있는 Market Place를 활용해서 보다 편하게 설치가 가능합니다.
- 파이썬은 기본 파이썬 설치후 필요한 라이브러리들을 pip 라는 설치 프로그램을 이용해 간단히 설치가 가능합니다. 필요한 라이브러리를 계속 하나씩 설치 하기 힘들때는 Anaconda 라는 패키지를 이용해 한번에 설치가 가능합니다
- 아나콘다 경로
- <https://www.continuum.io/downloads>(64비트가 더 안정적으로 설치됩니다)

# 이클립스와 파이썬



# 이클립스와 파이썬

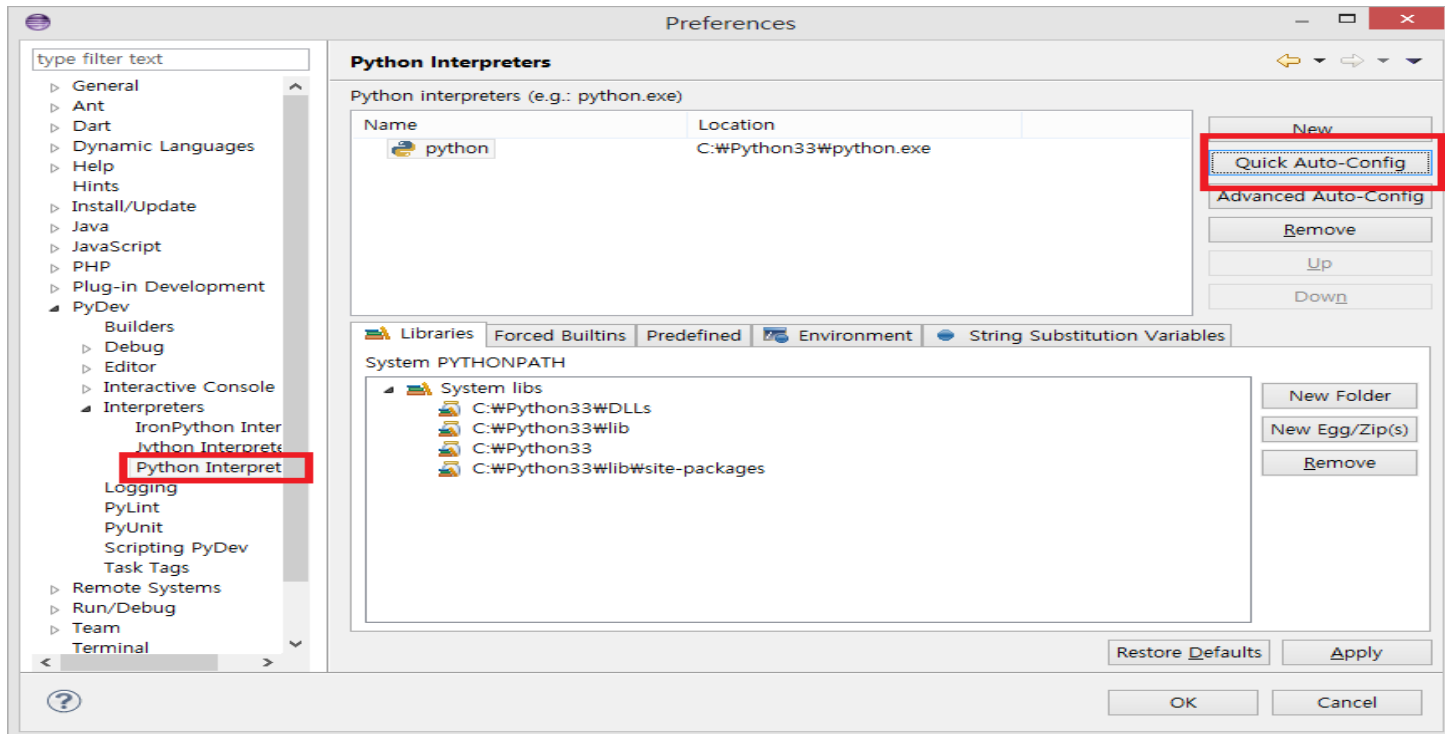




# 이클립스와 파이썬

## 인터프리터 설정하기

이클립스에서 'Windows > Preferences' 를 열어주세요.



# 이클립스와 파이썬

- 보안때문에 안될경우 아래 경로에서
- <https://sourceforge.net/projects/pydev/files/pydev/PyDev%205.1.1/>
- 파일을 직접 다운받아 압축을 푼다
- 두개의 폴더가 생성되는데 이 폴더 두개를 복사해서 이클립스 폴더에 붙여넣기 한다
- 설치버전의 경우에 다음 폴더에 놓는다

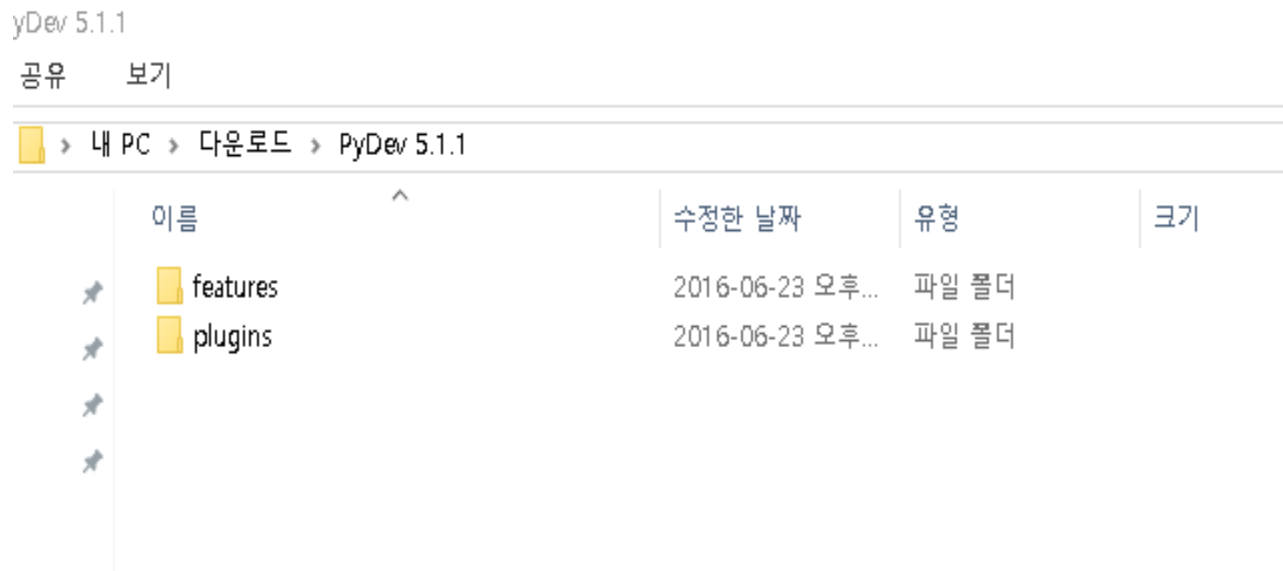
C:\Users\계정\p2\pool

# 이클립스와 파이썬

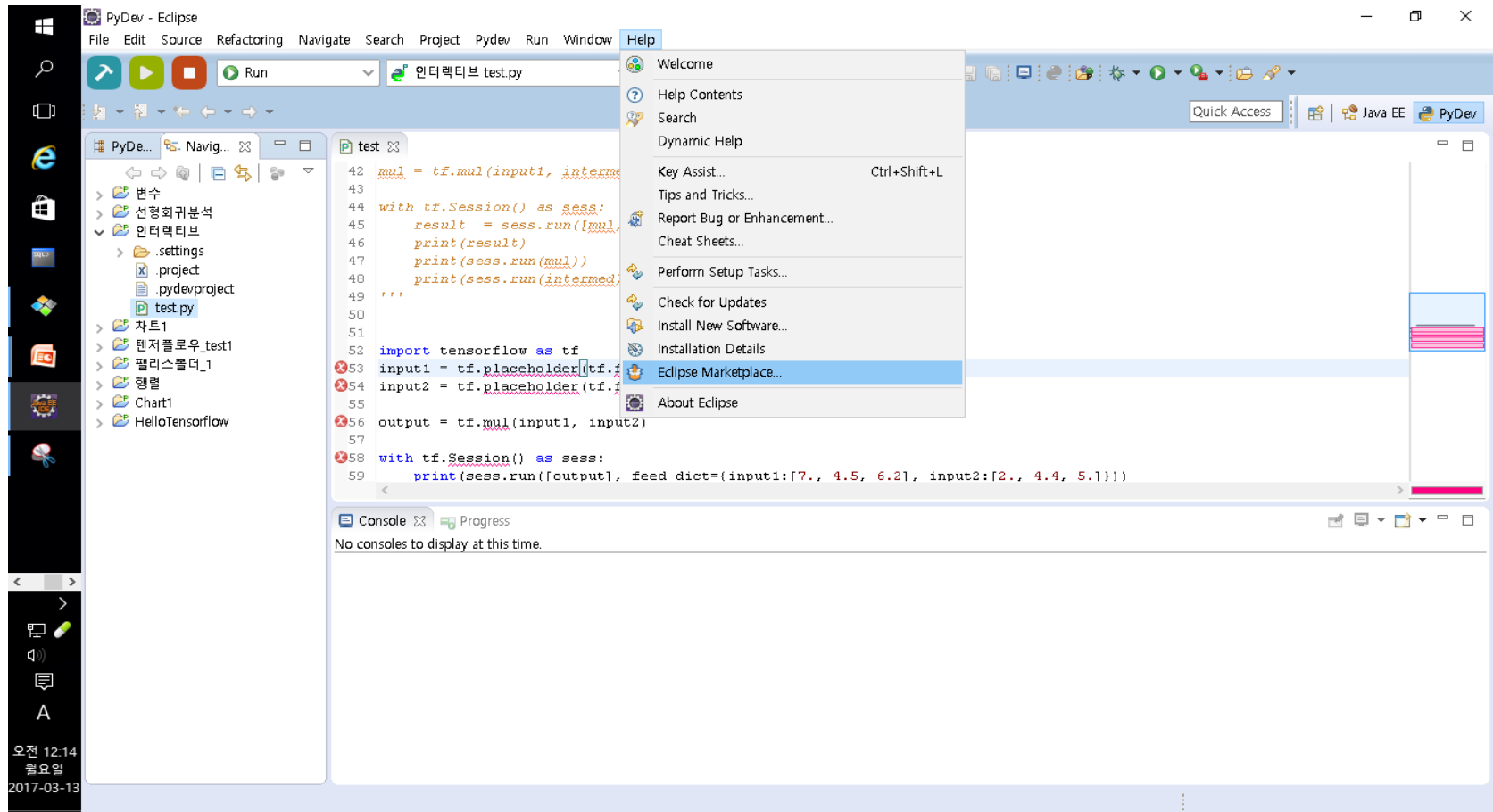
- New > Project > PyDev Project 프로젝트 생성
- New > File > Hello.py 를 추가한다
- `print("Hello")`
- 실행

# 이클립스와 파이썬

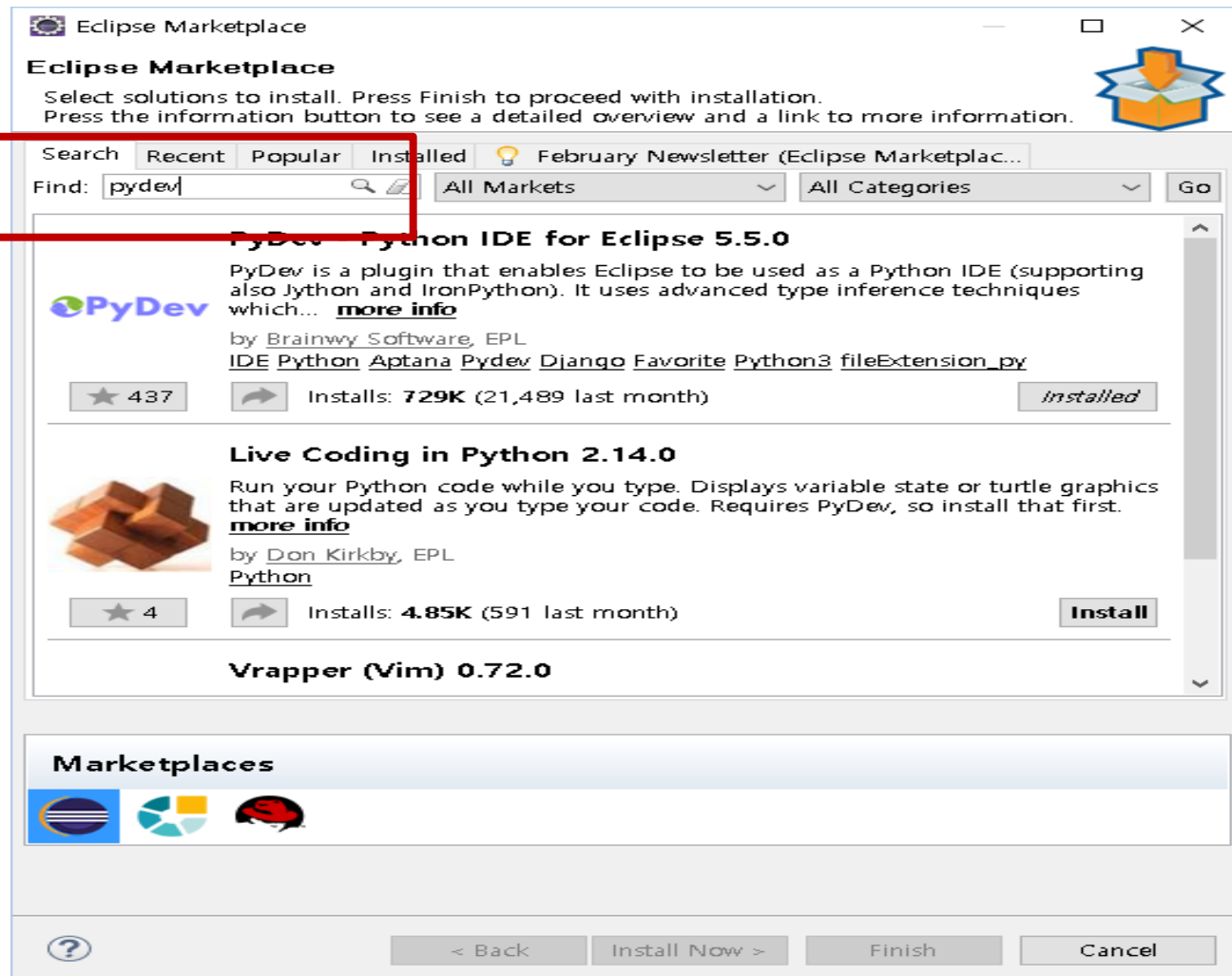
- 그림



# 이클립스 Market Place



# 이클립스 Market Place



# Visual studio

- **Visual Studio Express 2013 Update 3 for Windows Desktop 설치**
- 먼저 아래의 사이트로 이동을 합니다.
- <http://www.visualstudio.com/downloads/download-visual-studio-vs#d-express-windows-desktop>
- PTVS를 설치하기 위하여 해당 홈페이지로 이동을 합니다.
- <http://pytools.codeplex.com/>
- 참고 사이트
- <https://wikidocs.net/1033>

# 자료형 및 연산자



# 변수명

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

ex.

```
>>> friend = 10
>>> Friend = 1
>>> friend
10
>>> Friend
1
```

- 인터프리터 언어라 별도로 선언을 하지 않는다

# 수치형

- int
  - `>>> 10, 0x10, 0o10, 0b10`  
`(10, 16, 8, 2)`
- float
  - `>>> type(3.14), type(314e-2)`  
`(<class 'float'>, <class 'float'>)`
- Complex(복소수형)
  - `>>> x=3-4j`  
`>>> type(x), x.imag, x.real, x.conjugate()`  
`(<class 'complex'>, -4.0, 3.0, (3+4j))`
- 연산자
  - `+, -, *, /, //(몫 연산자), %, **(거듭제곱), =`

# 사칙연산연습

```
>>> a=10; b=3;
>>> print(a+b);
13
>>> print(a-b);
7
>>> print(a*b);
30
>>> print(a/b);
3.3333333333333335
>>> print(a//b);
3
>>> print(a%b);
1
>>> print(a**b);
1000
```

# 문자

- 'string', "string",  
"""줄바꿈도  
그대로 적용됩니다"""
- 문자열을 취급할 경우에  
' 나 " 모두 사용할 수 있  
습니다
- Escape 문자모두

| Escape 문자<br>사용예 | 의미            |
|------------------|---------------|
| \n               | 개행(줄바꿈)       |
| \t               | 탭             |
| \r               | 캐리지 리턴        |
| \0               | 널(Null)       |
| \\               | 문자 '\'        |
| \'               | 단일<br>인용부호(') |
| \"               | 이중<br>인용부호(") |

문자

- **+, \* 연산자**
  - >>> 'py' 'thon'  
'python'
  - >>> 'py' \* 3  
'pypypy'
- **인덱싱 & 슬라이싱**
  - >>> 'python'[0]  
'p'  
>>> 'python'[5]  
'n'  
>>> 'python'[1:4]  
'yth'  
>>> 'python'[-2:]  
'on'

[시작위치:끝위치]

|    |    |    |    |    |    |   |
|----|----|----|----|----|----|---|
| p  | y  | t  | h  | o  | n  |   |
| 0  | 1  | 2  | 3  | 4  | 5  | 6 |
| -6 | -5 | -4 | -3 | -2 | -1 |   |

# 유니코드

- 모든 문자열(string)이 기본적으로 유니코드입니다.
- 유니코드 이외의 인코딩이 있는 문자열은 bytes로 표현됩니다.
  - ```
>>> type('가')  
<class 'str'>  
>>> '가'.encode('utf-8')  
b'\xea\x80'  
>>> type('가'.encode('utf-8'))  
<class 'bytes'>
```

입력 연습

- 입력은 input 함수를 사용합니다. 입력값은 항상 String 타입입니다.
- Input("프롬프트");

```
name = input("your name ? ")  
address = input("your address?")  
print(name + "'s address is " + address)
```

리스트

- 리스트는 쉽게 값들의 나열이라고 생각하시면 됩니다.
- 파이썬은 배열대신 list 라는 타입을 제공합니다. 이 타입은 배열과 linked list 중간쯤 되는 타입이라고 생각하시면 됩니다
- 마치 집합처럼 여러 개의 데이터를 초기화 시켜서 객체를 생성할 수도 있고 메서드들의 호출(append, insert, extend, +)등을 통해 데이터를 추가하거나 확장할 수 있습니다
- 또한, 인덱싱과 슬라이싱도 가능합니다.
 - ```
colors = ['red', 'green', 'gold']
print(colors)
['red', 'green', 'gold']
```



# 리스트

- Index를 이용해 어디에 원하는 값이 있는지 확인 가능합니다.
  - ```
>>> colors.index('red')  
0
```
- Count를 이용해 원하는 값의 개수를 알 수 있으며, pop을 이용해 값을 뽑아낼 수도 있습니다.
 - ```
>>> colors
['red', 'black', 'green', 'gold', 'blue', 'white', 'gray', 'red']
>>> colors.count('red')
2
>>> colors.pop()
'red'
>>> colors
['red', 'black', 'green', 'gold', 'blue', 'white', 'gray']
```
- Remove로 값을 삭제하거나, sort로 정렬을 할 수도 있습니다.

# 리스트 사용 예

```
#-*- coding: utf-8 -*-
colors=['red', 'green', 'blue']; #객체 생성하기
print(colors); #맨뒤에 데이터 추가하기
colors.append('yellow');
print(colors);
colors.insert(1, 'magenta'); #데이터를 중간에 끼워넣기
print(colors);
colors.extend(['white', 'black']); #list에 다른 list 를 추가하기 반드시 list
 만 extend 가능하다
print(colors);
colors += ['violet']; #append처럼 동작함
print(colors);
print(colors.index('red', 0, 5)) #위치 찾기 - 시작위치, 마지막 위치
print(colors.index('black', 0, len(colors))) #위치 찾기
print(colors.count('red')) #현재 데이터 개수 알아내기
print(colors.count('빨강')) #현재 데이터 개수 알아내기
```

# 세트

- 세트(set)는 수학시간에 배운 집합과 동일합니다.
- 세트는 리스트와 마찬가지로 값들의 모임이며, 순서가 없습니다.
- 제공되는 메소드는 리스트와 유사하며, 추가적으로 교집합(intersection)과 합집합(union)이 제공됩니다.
  - ```
>>> a = {1, 2, 3}
>>> b = {3, 4, 5}
>>> a.union(b)           # 합집합
{1, 2, 3, 4, 5}
>>> a.intersection(b)   # 교집합
{3}
```

튜플

- 튜플(tuple)은 리스트와 유사하나, 읽기전용입니다.
- 읽기 전용인 만큼 제공되는 함수도 리스트에 비해 적지만, 속도는 그만큼 빠릅니다.
- 튜플에서 제공되는 메소드는 count, index 정도입니다.
- 튜플을 이용하면 'C'와 같은 언어에서는 변수가 하나 더 필요한 swap 예제를 다음과 같이 간단하게 해결할 수도 있습니다.
 - ```
>>> a, b = 1, 2
>>> print(a, b)
1 2
>>> a, b = b, a
>>> print(a, b)
2 1
```

# 딕셔너리

- 딕셔너리는 키와 값의 쌍으로 이루어져 있으며, 다음과 같이 정의할 수 있습니다.
  - ```
>>> d = dict(a=1, b=3, c=5)
>>> d
{'a': 1, 'c': 5, 'b': 3}
```
- 새로운 값의 추가나 변경은, 다음과 같이 간단하게 새로운 키와 값을 할당하면 됩니다.
 - ```
>>> color
{'apple': 'red', 'banana': 'yellow'}
>>> color["cherry"] = "red"
>>> color
{'cherry': 'red', 'apple': 'red', 'banana': 'yellow'}
>>> color["apple"] = "green"
>>> color
{'cherry': 'red', 'apple': 'green', 'banana': 'yellow'}
```

# 딕셔너리

- 딕셔너리의 내용을 얻기 위해서는 다음과 같이 items(), keys(), values() 메소드를 사용하면 됩니다. items()는 딕셔너리의 모든 키와 값을 튜플로 묶어서 반환하며, keys()는 키만을, values()는 값만을 반환합니다.
  - ```
>>> for k, v in color.items():  
    [TAB]print(k, v)  
cherry red  
apple green  
banana yellow
```
- 삭제는 del을 이용할 수도 있으며 clear를 이용해 한번에 삭제할 수도 있습니다.
 - ```
>>> color
{'cherry': 'red', 'apple': 'green', 'banana': 'yellow'}
>>> del color['cherry']
>>> color
{'apple': 'green', 'banana': 'yellow'}
>>> color.clear()
>>> color
{}
```

# 부울

- 부울(bool)은 참과 거짓을 나타내는 자료형으로, 가능한 값은 True와 False 뿐입니다.
- 주로 부울은 부울 값들 간의 논리연산이나, 수치들간의 비교연산의 결과로 사용됩니다.
- 비교연산자의 종류는 '크다(>)', '작다(<)', '같다(==)', '다르다(!=)', '같거나 크다(>=)', '같거나 작다(<=)' 가 있습니다.
- 논리연산자는 'and(&)', 'or(|)', 'not' 이 있습니다.  
'and'는 두 값이 모두 참이어야만 참을 반환하고, 'or'는 둘 중 하나의 값만 참이면 참을 반환합니다. 또한 'not'은 반대의 값을 반환합니다.  
논리연산자 역시 비교연산자와 함께 제어문의 조건에서 주로 사용됩니다.

# 함수



# 함수

- 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.

- A. 함수의정의
- B. return
- C. 인수전달
- D. 스코핑 룰
- E. 함수 인수
- F. Lambda 함수
- G. 재귀적 함수 호출
- H. Pass
- I. \_\_doc\_\_속성과 help 함수
- J. 이터레이터
- K. 제네레이터

# 함수의 정의

- 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

- 함수 선언 문법

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

- 간단한 함수 선언해 보기

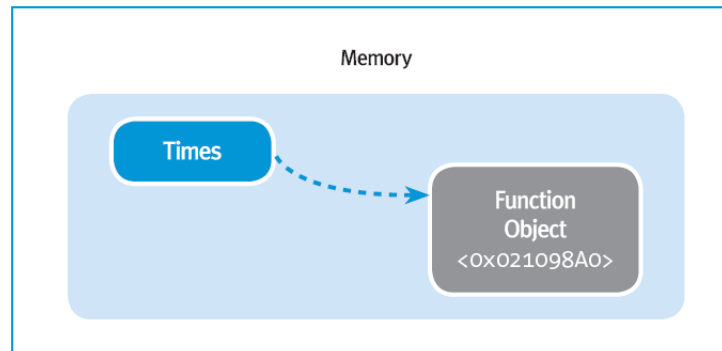
- 입력 받은 2개의 인수를 서로 곱한 값을 리턴한다.

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

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

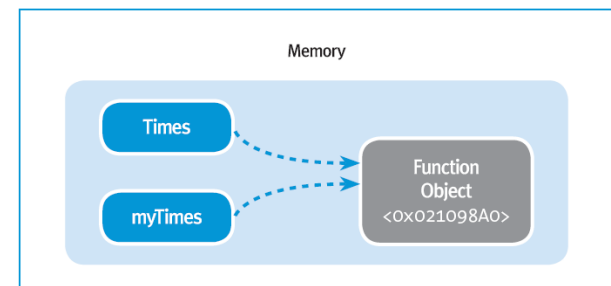
# 함수의 선언

- 메모리에 함수 객체가 생성된다.
- 함수 객체를 가리키는 레퍼런스가 생성된다.



- 함수 레퍼런스를 통해서 함수를 사용하게 된다.
  - 함수 레퍼런스는 다른 변수에 할당 할 수 있다.

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



# return

- 함수를 종료시키고 호출한 곳으로 돌아가게 한다.
- 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

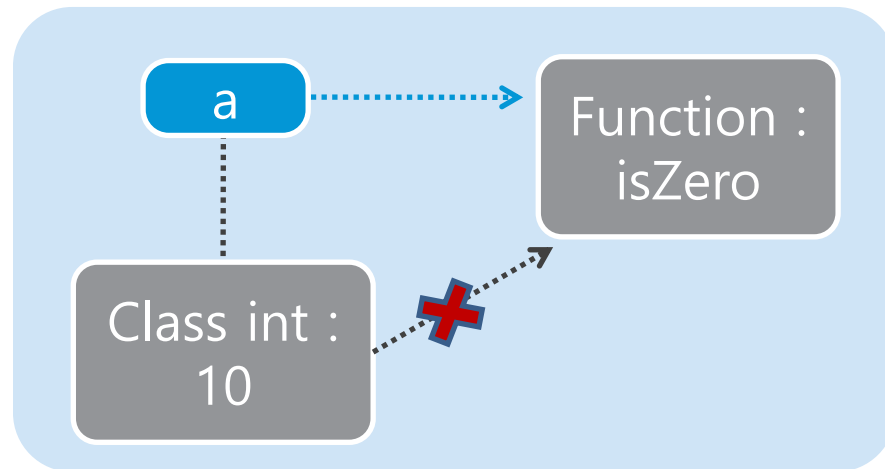
>>> retval = setValue(10)

>>> print(retval)
None
```

# 파이썬 함수에서 인수 전달-1

- 파이썬에서 함수 인수는 레퍼런스를 이용해 전달
  - 함수의 인수는 호출자 내부 객체의 레퍼런스

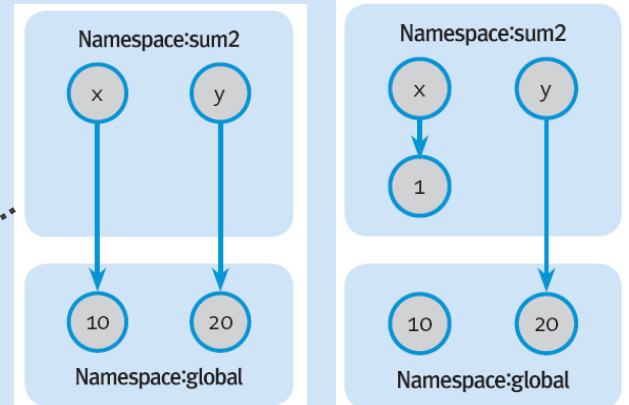
```
>>> a = 10
>>> def isZero(arg1):
>>> return arg1 == 0
>>> isZero(a)
False
```



# 함수에서 인수 전달-2

- 호출자가 전달하는 변수의 타입에 따라 다르게 처리
  - 변경가능 변수 (mutable)
  - 불가능 변수 (immutable)
- 변경 불가능 변수 예제

```
>>> a = 10
>>> b = 20
>>> def sum1(x, y):
>>> return x + y
>>> sum1(a, b)
30
>>> x = 10
>>> def sum2(x, y):
>>> x = 1
>>> return x + y
>>> sum2(x, b)
21
>>> x
10
```



이 부분에서 값이 1인 객체가 생성되고 x에 레퍼런스를 할당합니다

x를 인수로 넣어 줍니다

함수 내부에서 변경한 사항이 외부에 영향을 미치지 않습니다

# 함수에서 인수 전달-3

- 변경가능한 변수를 인수로 전달(리스트나 튜플).

```
>>> 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
```



# 스코핑 룰(Scoping rule)

- 이름공간 (Name Space)
  - 변수의 이름이 저장되어 있는 장소
  - 함수 내부의 이름공간, 지역 영역(Local scope)
  - 함수 밖의 영역, 전역 영역(Global scope)
  - 파이썬 자체에서 정의한 내용에 대한 영역, 내장 영역 (Built-in Scope)
- LGB 규칙
  - 변수 이름을 찾을 때 Local Scope -> Global Scope -> Built-in Scope 순서로 찾는다.

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

함수 내 지역 영역에 해당 이름이 없기  
전역 영역에서 같은 이름을 찾아서 시

함수 내 지역 영역에 x라는 이름이 등록됨

- 지역 영역에서 전역 영역의 이름을 접근할 때 global을 이용

# 함수에서 인수 모드 - 1

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

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

>>> Times()
200

>>> Times(5) ← a에만 5가 할당됩니다
100
```

- 키워드 인수
  - 인수 이름으로 값을 전달하는 방식
  - 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'
```

# 함수에서 인수 모드 - 1

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

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

>>> Times()
200

>>> Times(5) ← a에만 5가 할당됩니다
100
```

# 함수에서 인수 모드 - 2

- 정의되지 않은 인수 처리하기.
  - \*\*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&'
```

# 람다(lambda) 함수

- 이름이 없는 1줄 짜리 함수

```
lambda 인수 : <구문>
```

- 한 줄의 간단한 함수가 필요한 경우
- 프로그램의 가독성을 위해서
- 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 내부에서 자기 자신을 계속 호출 하는 방법
  - 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
```

# pass 구문(statement)

- 아무 일도 하지 않습니다.

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

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

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

>>> sample()
>>>
>>>
```

# \_\_doc\_\_ 속성과 help 함수

- help 함수를 이용해 함수의 설명을 볼 수 있다.

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

- 사용자가 만든 함수도 help를 사용해 설명을 볼 수 있다.

```
>>> def plus(a, b): ← 간단한 함수를 생성합니다
 return a + b

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



# `__doc__` 속성과 `help` 함수

- 조금 더 자세한 설명을 추가 하려면 `__doc__` 속성을 이용한다.

```
>>> 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
```

# 이터레이터 (Iterator)

- 순회가능한 객체의 요소를 순서대로 접근 할 수 있는 객체
  - 내부 반복문을 관리해 주는 객체
  - 이터레이터 안의 `__next__()`를 이용해 순회 가능한 객체의 요소를 하나씩 접근 할 수 있다.

```
>>> s = 'abc'
>>> it = iter(s) ← iter 함수는 순회 가능한 객체에서 이터레이터를 가
>>> it ← 이터레이터 객체입니다
<iterator object at 0x00A1DB50>
>>> next(it) ←
'a' next 함수는 이터레이터가 가리키는 값을 리
>>> next(it) 턴하고 다음 요소를 가리키게 합니다.
'b'
>>> it.__next__() ← 이런 식으로 직접 __next__() 메소드를 실행 시
'c'
>>> next(it)
Traceback (most recent call last):
 File "<stdin>", line 1, in ?
 next(it)
StopIteration
```

요소의 끝 부분에서 `__next__` 키면 `StopIteration` 예외가

# 제네레이터 (Generator)

- return 대신 yield라는 구문을 이용해 함수 객체를 유지한 채 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
```

# 제네레이터 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'
```

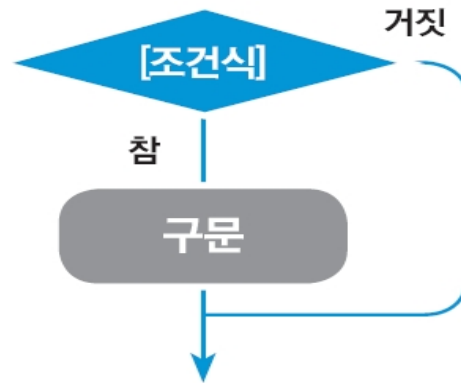
# 제어구조

# 목 차

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

# if 문

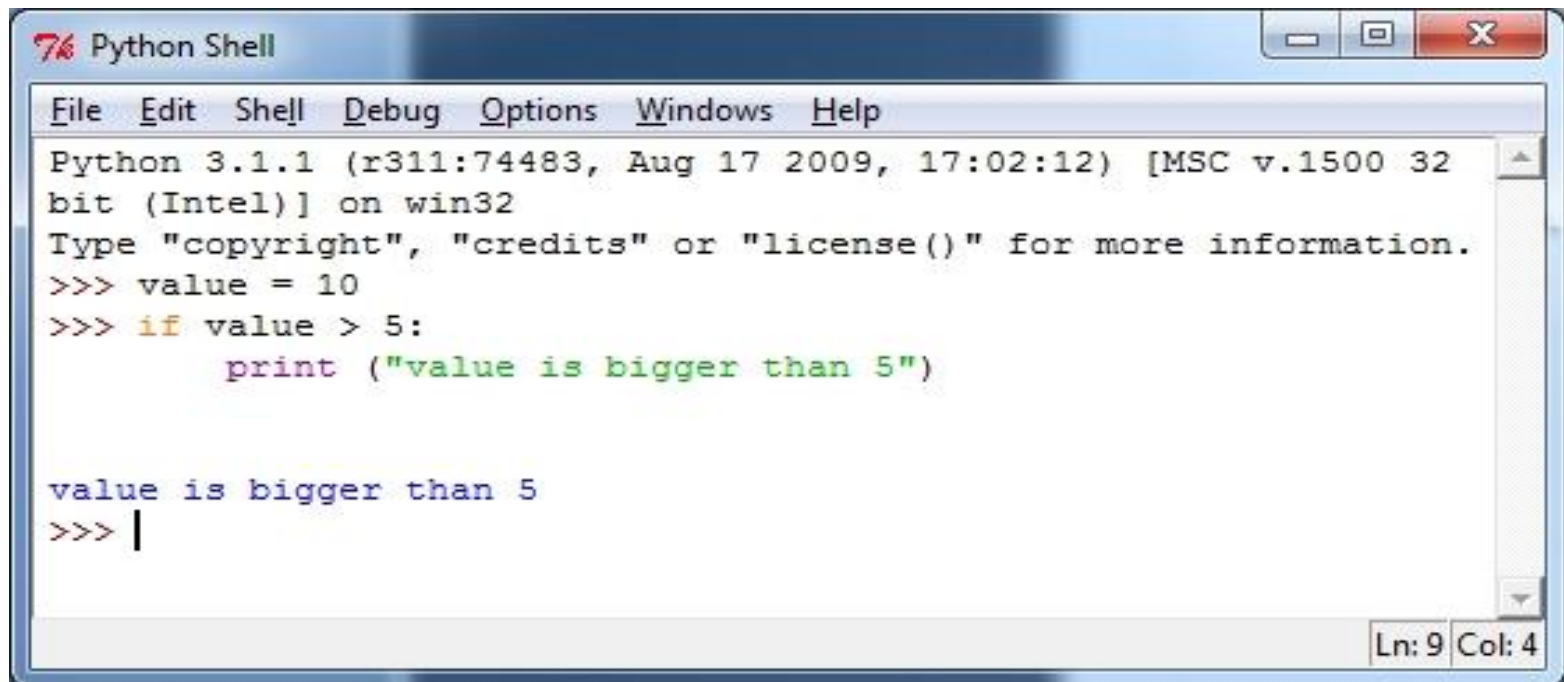
**if** <조건식> :  
    <구문>



- 조건식을 평가하고 참인 경우만 구문이 수행
- 2개 이상의 구문은 들여쓰기로 블록을 지정
  - 함수와 동일
  - 들여쓰기의 정도는 파일 전체를 통틀어 일치해야 함

# if 문

- 예제 1



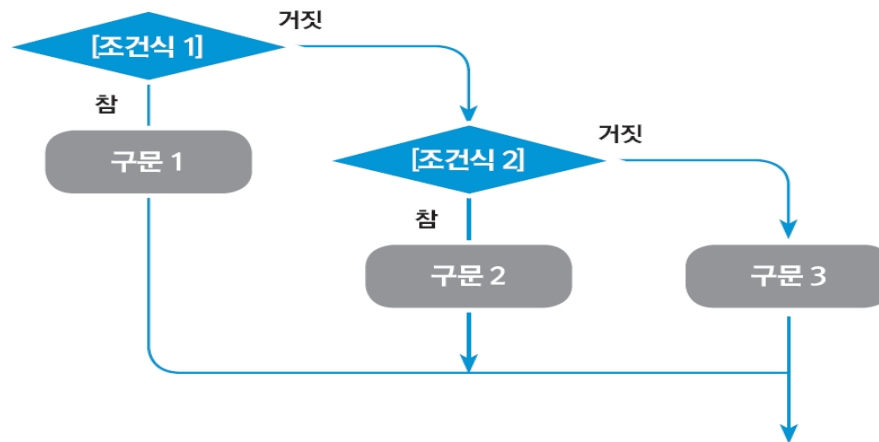
The screenshot shows a 'Python Shell' window with a menu bar (File, Edit, Shell, Debug, Options, Windows, Help) and a status bar (Ln: 9 Col: 4). The text inside the window is as follows:

```
Python 3.1.1 (r311:74483, Aug 17 2009, 17:02:12) [MSC v.1500 32
bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> value = 10
>>> if value > 5:
 print ("value is bigger than 5")

value is bigger than 5
>>> |
```



# elif, else



- **elif**
  - 2개 이상의 조건을 처리하는 경우
  - if는 가장 처음에만 사용할 수 있는 반면에, elif는 필요한 만큼 사용 가능
- **else**
  - 어떠한 조건에도 해당하지 않는 경우
  - 가장 마지막에만 사용 가능

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

# elif, else

- 예제 2

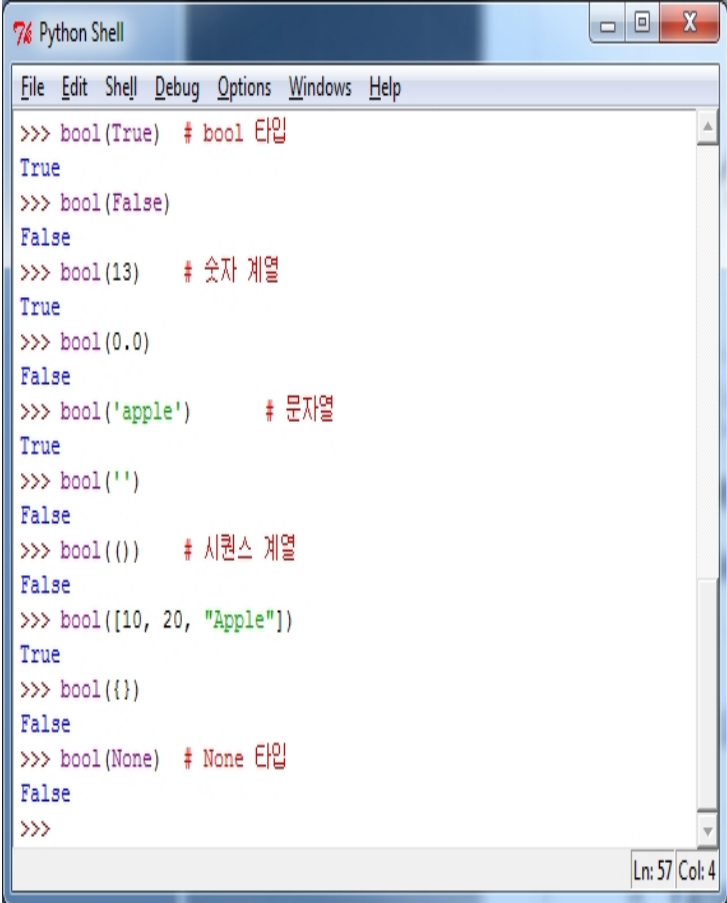
```
-*- coding: cp949 -*-
score = int(input('Input Score: ')) # 사용자로부터 정수값을 입력 받습니다
if 90 <= score <= 100:
 grade = "A"
elif 80 <= score < 90:
 grade = "B"
elif 70 <= score < 80:
 grade = "C"
elif 60 <= score < 70:
 grade = "D"
else:
 grade = "F"

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

- 파이썬의 조건식 표현 방법
  - 70 <= score < 80
  - grade >= 70 and grade < 80

# 조건식의 참/거짓 판단

- 기본적으로 자료형의 bool 값과 동일(2장 참조)
  - True로 판명:  $10 > 0$
  - False로 판명:  $5 > 10$
- False
  - 0, 0.0, (), [], {} "(빈 문자열), None인 경우
- True
  - False인 경우를 제외한 값이 할당된 경우



```
Python Shell
File Edit Shell Debug Options Windows Help
>>> bool(True) # bool 타입
True
>>> bool(False)
False
>>> bool(13) # 숫자 계열
True
>>> bool(0.0)
False
>>> bool('apple') # 문자열
True
>>> bool('')
False
>>> bool(()) # 시퀀스 계열
False
>>> bool([10, 20, "Apple"])
True
>>> bool({})
False
>>> bool(None) # None 타입
False
>>>
```

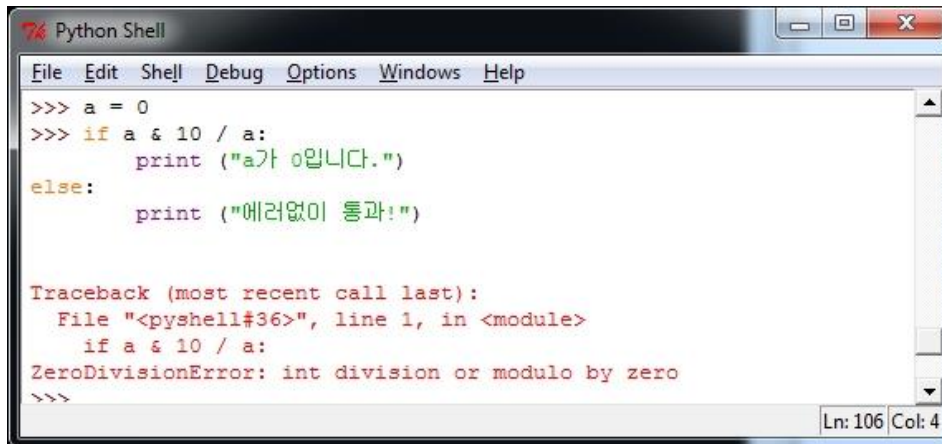
Ln: 57 Col: 4

# and / or(단축)

- 2개 이상의 논리식 판별을 위한 연산자
  - 식의 왼쪽에서 오른쪽으로 판별
  - **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 수행되는 것은 아님!!

# 단축 평가

- a가 0인 경우 조건식 'a & 10/a'는 거짓
  - 10 / a에 의하여 ZeroDivisionError가 발생



The screenshot shows a Python Shell window with a menu bar (File, Edit, Shell, Debug, Options, Windows, Help). The code entered is:

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

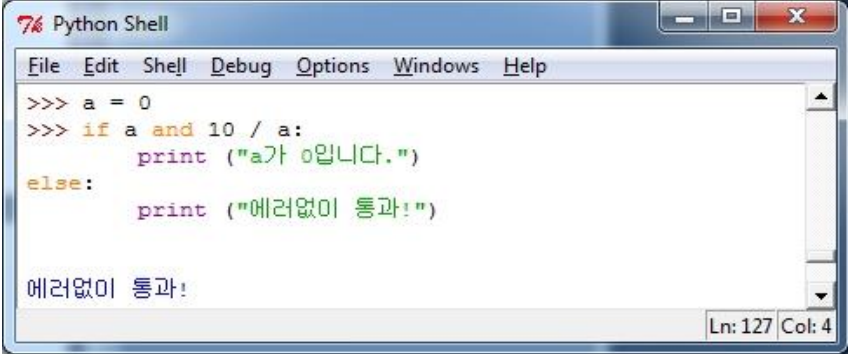
A traceback error message is displayed:

```
Traceback (most recent call last):
 File "<pyshell#36>", line 1, in <module>
 if a & 10 / a:
ZeroDivisionError: int division or modulo by zero
>>>
```

The status bar at the bottom right indicates "Ln: 106 Col: 4".

# 단축 평가

- 단축 평가란?
  - 조건식 전체를 판단하지 않고 순차적으로 진행하다 식 전체가 자명한 경우, 더이상 수식을 평가하지 않는 방법
- '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



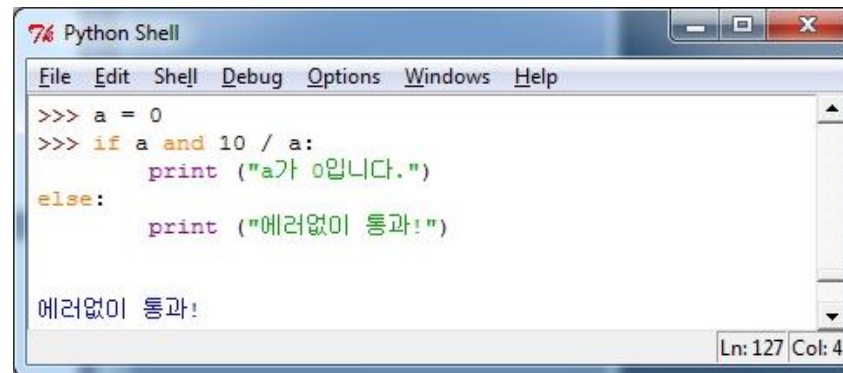
The screenshot shows a Python Shell window with a menu bar (File, Edit, Shell, Debug, Options, Windows, Help). The code entered is:

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

The output shown is "에러없이 통과!". The status bar at the bottom right indicates "Ln: 127 Col: 4".

# 단축 평가의 장점

- 조건식의 결과가 결정되는 시점 이후로 추가적인 판별 연산을 수행하지 않기 때문에 속도 향상
- Run time error 발생을 try ~ except 구문이 아닌 논리식으로 사전에 차단 가능



```
Python Shell
File Edit Shell Debug Options Windows Help
>>> a = 0
>>> if a and 10 / a:
>>> print ("a가 0입니다.")
else:
>>> print ("에러없이 통과!")

에러없이 통과!
Ln: 127 Col: 4
```

# while 문

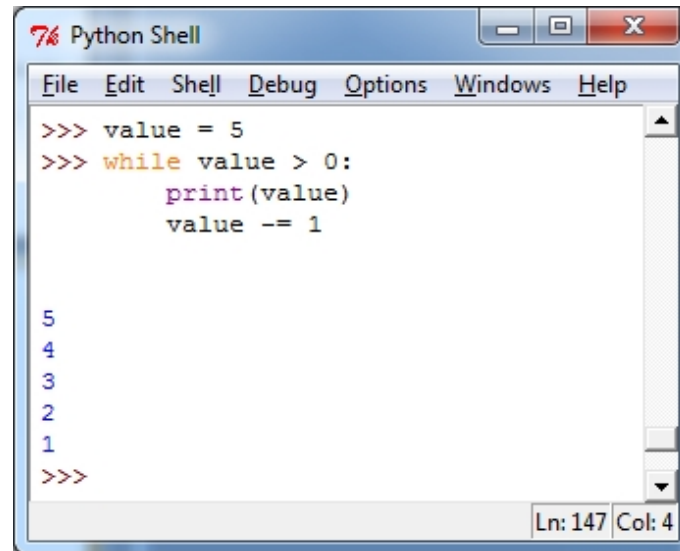
**while** <조건식> :  
    <구문>

- 조건식이 참(True)인 동안 내부 구문을 반복 수행
  - 조건식은 구문이 수행되기 이전에 우선 평가
  - 구문을 모두 수행 이후 다시 조건식을 재평가
  - 조건식이 거짓(False)이면 while 문 구조를 벗어남



# while 문

- 예제 3

A screenshot of a Python Shell window titled 'Python Shell'. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Windows', and 'Help'. The main text area contains the following code:

```
>>> value = 5
>>> while value > 0:
 print(value)
 value -= 1
```

The output of the code is displayed below the code:

```
5
4
3
2
1
>>>
```

The status bar at the bottom right shows 'Ln: 147 Col: 4'.

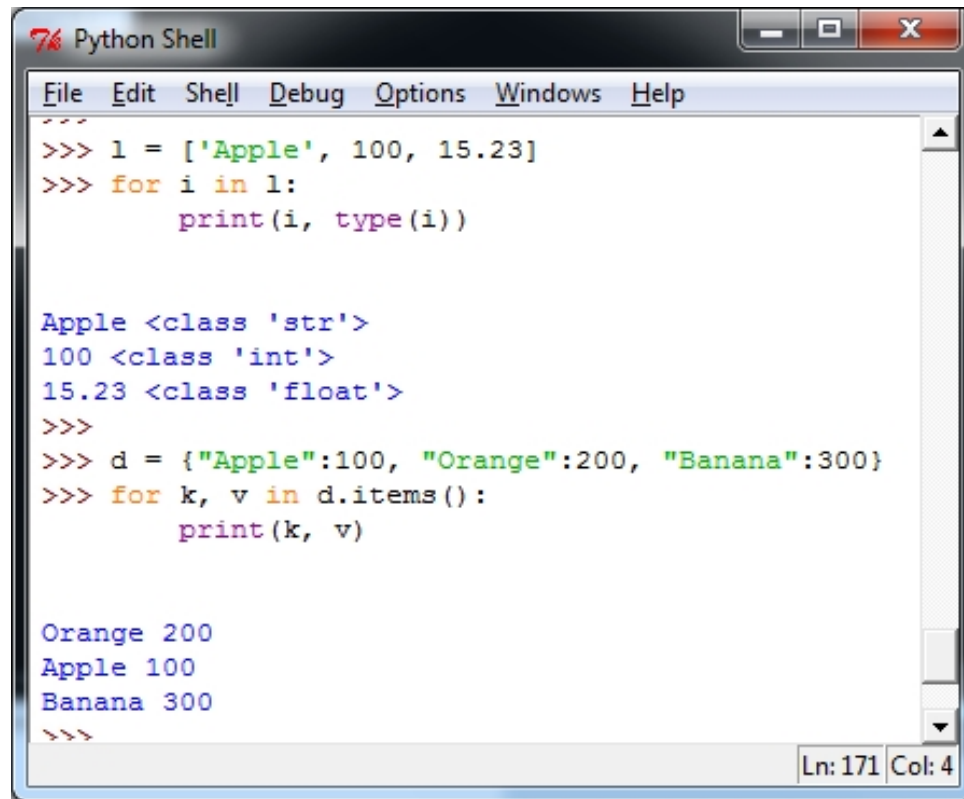
# for문

**for** <아이템 I> **in** <시퀀스형 객체 S> :  
    <구문>

- 시퀀스형 객체를 순차적으로 순회
  - '시퀀스형 객체 S'의 각 아이템을 '아이템 I'에 할당
  - 할당된 아이템 I를 가지고 구문을 수행
  - 모든 아이템을 순회하거나 break를 만나면 for문이 종료

# for 문

- 예제 4



```
Python Shell
File Edit Shell Debug Options Windows Help
>>> l = ['Apple', 100, 15.23]
>>> for i in l:
>>> print(i, type(i))

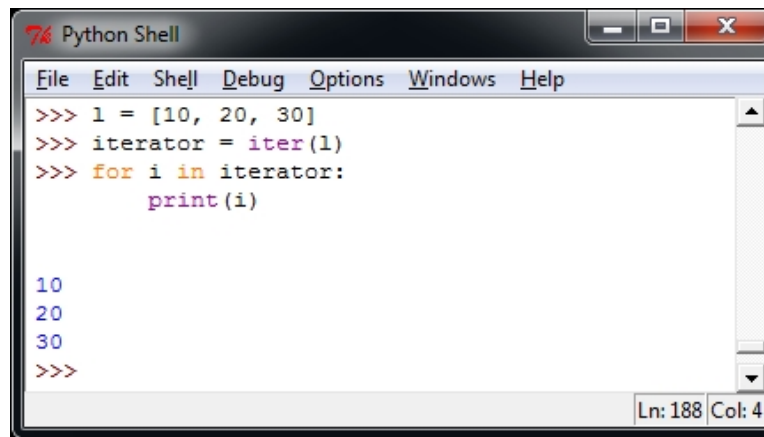
Apple <class 'str'>
100 <class 'int'>
15.23 <class 'float'>
>>>
>>> d = {"Apple":100, "Orange":200, "Banana":300}
>>> for k, v in d.items():
>>> print(k, v)

Orange 200
Apple 100
Banana 300
>>>
```

Ln: 171 Col: 4

# for 문

- for문에서 사용할수 있는 자료
  - 문자열, 리스트, 튜플, 사전
  - 이터레이터(iterator), 제너레이터 객체(3장 참조)

A screenshot of a Python Shell window. The window has a title bar that says "Python Shell" and a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area contains the following code:

```
>>> l = [10, 20, 30]
>>> iterator = iter(l)
>>> for i in iterator:
>>> print(i)
```

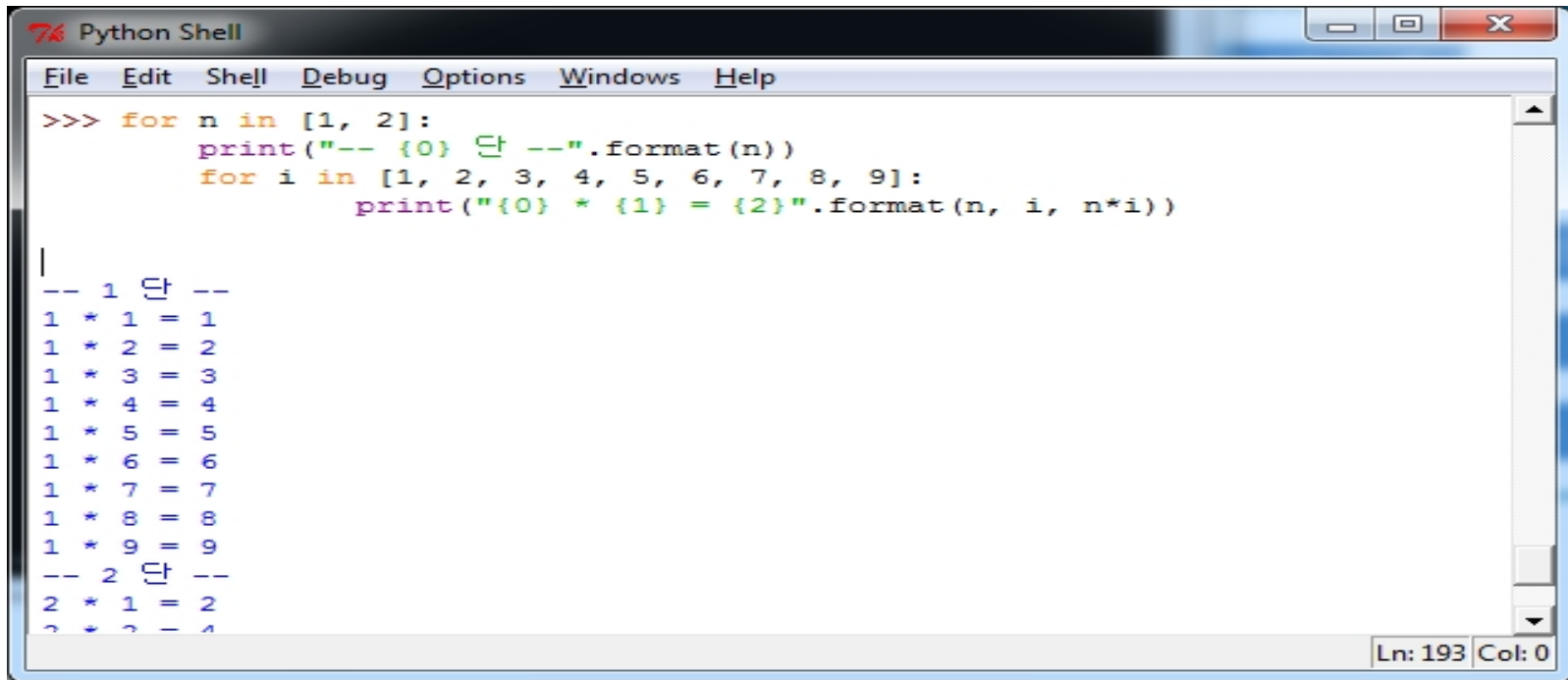
The output of the code is displayed below the input lines:

```
10
20
30
>>>
```

The status bar at the bottom right of the window shows "Ln: 188 Col: 4".

# for 문

- 반복문은 2개 이상 중첩해서 사용 가능

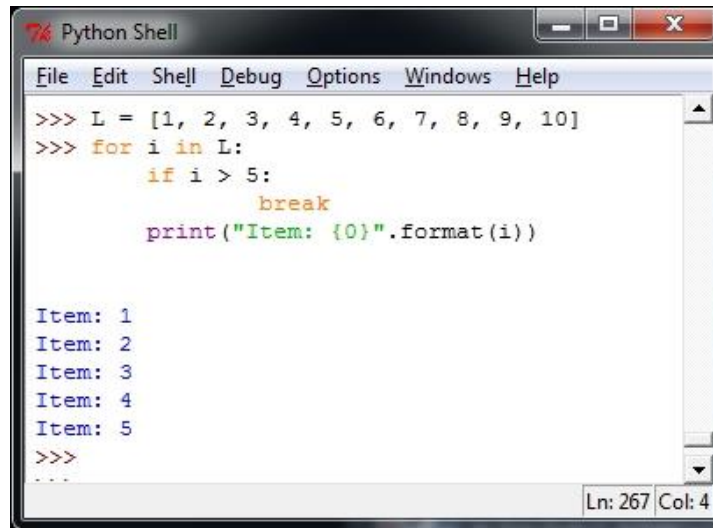


```
Python Shell
File Edit Shell Debug Options Windows Help
>>> for n in [1, 2]:
 print("-- {0} 단 --".format(n))
 for i in [1, 2, 3, 4, 5, 6, 7, 8, 9]:
 print("{0} * {1} = {2}".format(n, i, n*i))

-- 1 단 --
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9
-- 2 단 --
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
Ln: 193 Col: 0
```

# break

- break을 만나면 반복문 내부 블록을 벗어남

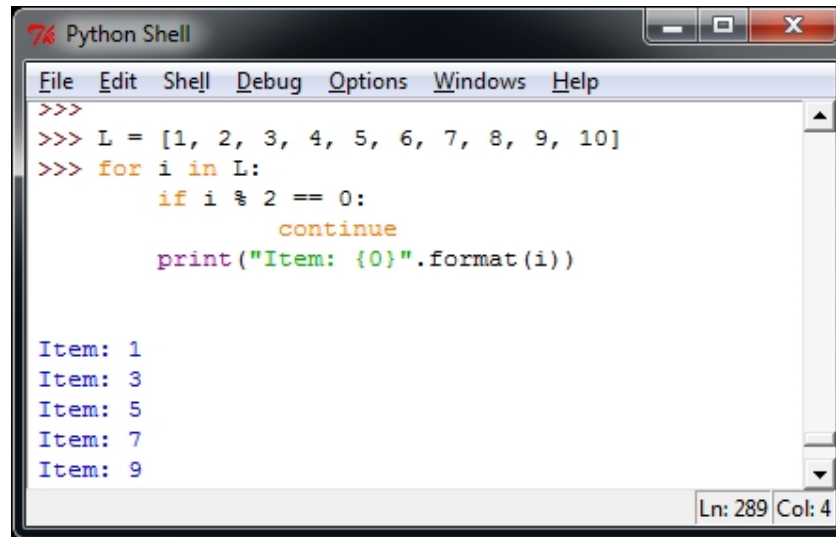


```
Python Shell
File Edit Shell Debug Options Windows Help
>>> L = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> for i in L:
>>> if i > 5:
>>> break
>>> print("Item: {}".format(i))
Item: 1
Item: 2
Item: 3
Item: 4
Item: 5
>>>
....
Ln: 267 Col: 4
```

The screenshot shows a Python Shell window with a menu bar (File, Edit, Shell, Debug, Options, Windows, Help). The code being executed is a for loop over a list L = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. Inside the loop, there is an if statement that checks if i > 5. If true, it executes a break statement, which exits the loop. The output shows the items 1 through 5 being printed, followed by the prompt >>> and a line of dots indicating the loop has ended. The status bar at the bottom right shows 'Ln: 267 Col: 4'.

# continue

- `continue` 이후 반복문 내부 블록을 수행하지 않고, 다음 아이টে임을 선택하여 내부 블록의 시작 지점으로 이동



```
Python Shell
File Edit Shell Debug Options Windows Help
>>>
>>> L = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> for i in L:
>>> if i % 2 == 0:
>>> continue
>>> print("Item: {}".format(i))

Item: 1
Item: 3
Item: 5
Item: 7
Item: 9
Ln: 289 Col: 4
```

# else

- 반복문 수행도중 break로 인하여 중간에 종료되지 않고 끝까지 수행되었을 때, else 블록이 수행
  - else 블록이 수행되는 경우

```
-*- 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 블록이 수행되지 않는 경우

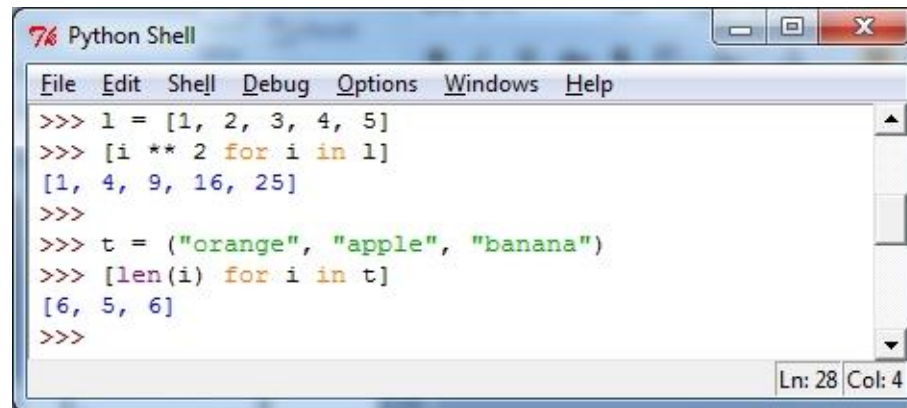
```
-*- coding: cp949 -*-
L = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for i in L:
 if i > 5:
 break
 print("Item: {0}".format(i))
else:
 print("Exit without break.")
print("Always this is printed") # 외부 루프 문장
```

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



# 리스트 내장

- [**<표현식>** **for** **<아이템>** **in** **<시퀀스 객체>** (**if** **<조건식>**)]
  - 기존 시퀀스 객체를 이용하여 추가적인 연산을 통하여 새로운 리스트 객체를 생성



```
Python Shell
File Edit Shell Debug Options Windows Help
>>> l = [1, 2, 3, 4, 5]
>>> [i ** 2 for i in l]
[1, 4, 9, 16, 25]
>>>
>>> t = ("orange", "apple", "banana")
>>> [len(i) for i in t]
[6, 5, 6]
>>>
Ln: 28 Col: 4
```

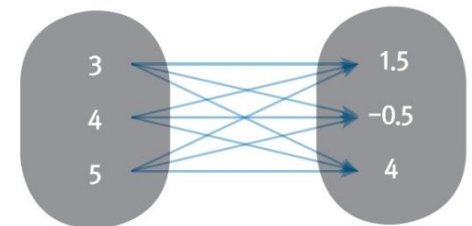
# 리스트 내장

- [**<표현식>** **for** **<아이템>** **in** **<시퀀스 객체>** (**if** **<조건식>**)]
  - 조건식을 이용하여 원본 객체에서 조건을 만족하는 아이템만 선별

```
Python Shell
File Edit Shell Debug Options Windows Help
>>> t = ("orange", "apple", "banana", "kiwi")
>>> [i for i in t if len(i)>5]
['orange', 'banana']
>>>
>>>
>>>
```

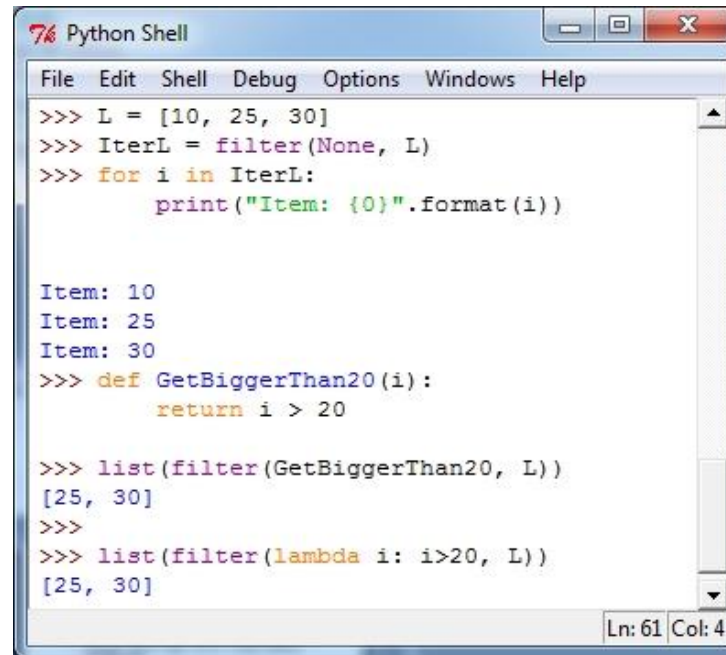
- 원본 리스트가 2개인 경우

```
Python Shell
File Edit Shell Debug Options Windows Help
>>> L_1 = [3, 4, 5]
>>> L_2 = [1.5, -0.5, 4]
>>> [x*y for x in L_1 for y in L_2]
[4.5, -1.5, 12, 6.0, -2.0, 16, 7.5, -2.5, 20]
>>> |
```



# 제어문 관련 유용 함수

- **filter**(<function>|None, 시퀀스 객체)
  - 함수의 결과 값이 참인 시퀀스 객체의 이터레이터를 반환
  - None이 오는 경우 필터링하지 않음

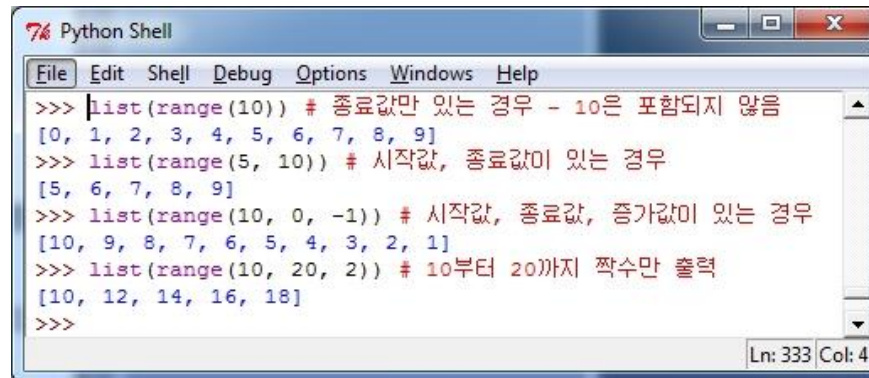


```
Python Shell
File Edit Shell Debug Options Windows Help
>>> L = [10, 25, 30]
>>> IterL = filter(None, L)
>>> for i in IterL:
>>> print("Item: {}".format(i))

Item: 10
Item: 25
Item: 30
>>> def GetBiggerThan20(i):
>>> return i > 20
>>> list(filter(GetBiggerThan20, L))
[25, 30]
>>>
>>> list(filter(lambda i: i>20, L))
[25, 30]
```

# 제어문 관련 유용 함수

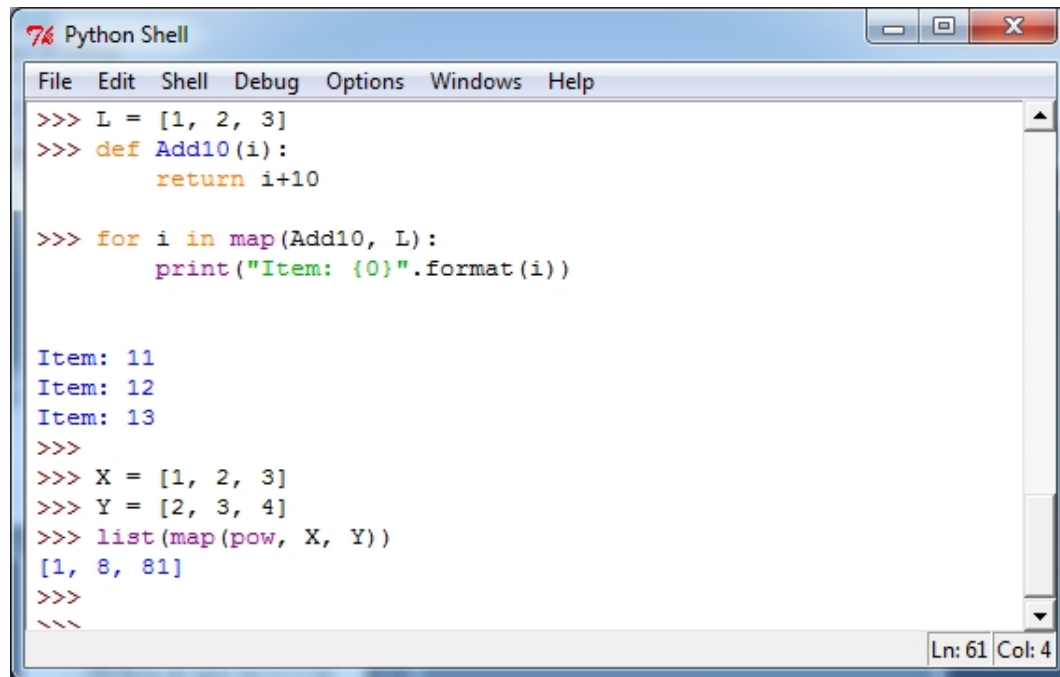
- **range**(['시작값', '종료값', '증가값'])
  - 수열을 순회하는 이터레이터 객체를 반환
  - 시작값과 증가값은 생략 가능하며, 이때는 각 0, 1이 할당

A screenshot of a Python Shell window titled 'Python Shell'. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Windows', and 'Help'. The shell contains several lines of Python code and their outputs, demonstrating the range function with different arguments. The status bar at the bottom right shows 'Ln: 333 Col: 4'.

```
>>> list(range(10)) # 종료값만 있는 경우 - 10은 포함되지 않음
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(5, 10)) # 시작값, 종료값이 있는 경우
[5, 6, 7, 8, 9]
>>> list(range(10, 0, -1)) # 시작값, 종료값, 증가값이 있는 경우
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
>>> list(range(10, 20, 2)) # 10부터 20까지 짝수만 출력
[10, 12, 14, 16, 18]
>>>
```

# 제어문 관련 유용 함수

- **map(<function>, 시퀀스 객체, ...)**
  - 시퀀스 객체를 순회하며 function의 연산을 수행
  - 함수의 인자수만큼 시퀀스 객체를 전달



```
Python Shell
File Edit Shell Debug Options Windows Help
>>> L = [1, 2, 3]
>>> def Add10(i):
>>> return i+10
>>> for i in map(Add10, L):
>>> print("Item: {0}".format(i))

Item: 11
Item: 12
Item: 13
>>>
>>> X = [1, 2, 3]
>>> Y = [2, 3, 4]
>>> list(map(pow, X, Y))
[1, 8, 81]
>>>
>>>
```

Ln: 61 Col: 4

# 배열

- #1차원배열

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

```
b = [0]*10 # [None]*10메모리 할당하기
```

```
for i in range(0, len(a)): #len(배열), len(list) 항목의 개수
 b[i] = a[i]
```

```
print(b)
```

## 2차원 배열

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

```
b = [[None]*3]*3
```

## 메모리 할당이 잘못 이루어져 결과안나옴

```
print("#####")
```

```
for i in range(0, len(a)):
```

```
 for j in range(0, len(a[i])):
```

```
 b[i][j] = a[i][j]
```

```
 print(b[i][j], a[i][j])
```

```
print(b)
```

## 2차원 배열

```
b = []
for i in range(3):
 b.append([None]*3)

for i in range(0, len(a)):
 for j in range(0, len(a[i])):
 b[i][j] = a[i][j]
b[0][0]=10
print(a)
print(b)
```



# 정 리

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

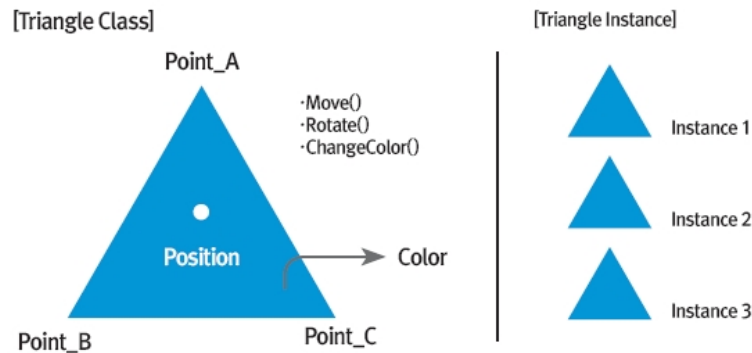
# 클래스

# 목 차

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

# 클래스란?

- 데이터와 데이터를 변형하는 함수를 같은 공간으로 작성



- 메서드(Method)
- 인스턴스(Instance)
- 정보 은닉(Information Hiding)
- 추상화(Abstraction)
  - 부모클래스
  - 자식 클래스

# 클래스 기본

- 클래스와 인스턴스

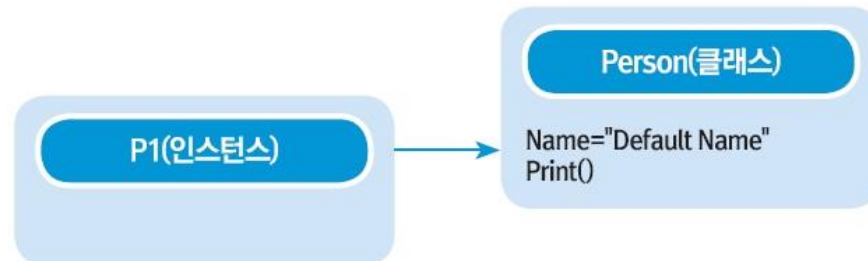
```
class Person: # 클래스 정의
 Name = "Default Name" # 멤버 변수

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

p1 = Person() # 인스턴스 객체 생성

p1.Print() # 멤버 변수값을 출력
```

- 클래스와 인스턴스 이름 공간



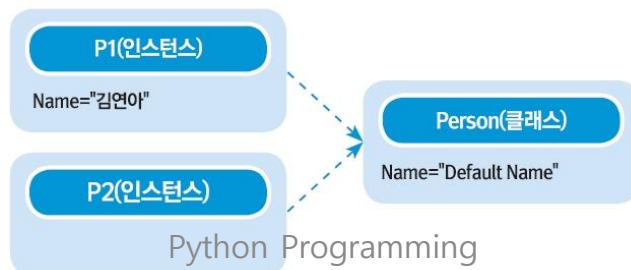
# 이름 공간

- 검색 순서

- 인스턴스 객체 영역 -> 클래스 객체 영역 -> 전역 영역
- 예제 코드

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

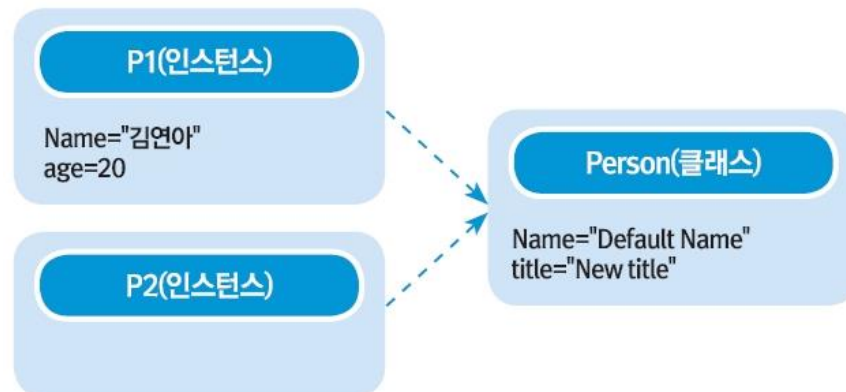
>>> p1 = Person() # 인스턴스 객체 생성
>>> p2 = Person()
>>> print("p1's name: ", p1.name) # 각 객체의 name 속성 출력
p1's name: Default Name
>>> print("p2's name: ", p2.name)
p2's name: Default Name
>>> p1.name = "김연아" # p1 인스턴스의 'name' 속성을 변경
>>> print("p1's name: ", p1.name)
p1's name: 김연아
>>> print("p2's name: ",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



# 이름 공간 – con'd

- 클래스와 인스턴스에 멤버 데이터 추가
- 파이썬에서는 클래스객체와 인스턴스 객체에 동적으로 멤버를 추가할 수 있다
- 모든 멤버의 접근 권한은 기본이 public 이다
  - 예제 코드

```
>>> 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 인스턴스 관계

- isinstance(인스턴스 객체, 클래스 객체)
  - 인스턴스 객체가 어떤 클래스로부터 생성되었는지 확인
  - 불린 형태로 결과 반환
  - 예제 코드

```
>>> 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
  - 생성시 초기화 작업을 수행
  - 인스턴스 객체가 생성될 때 자동으로 호출
  - `__init__()` : 하나만 만들 수 있다
  - 생성자를 다양하게 사용하려면 생성자 매개변수에 None 기본값을 주거나 리스트 형태(\*args) 또는 튜플 형태(\*\*args)로 전달해야 한다
- 소멸자
  - 소멸시 종료 작업을 수행
  - 인스턴스 객체의 참조 카운터가 '0'이 될 때 호출
  - `__del__()`

- 예제 코드

```
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) 메서드
  - 인스턴스 객체를 통하지 않고, 클래스를 통해 직접 호출할 수 있는 메소드
    - 인스턴스 객체를 참조하는 self 인자가 필요하지 않음
  - 형식
    - <호출할 메소드 이름> = staticmethod(클래스내 정의한 메소드 이름)
- 클래스(class) 메서드
  - 클래스 영역의 데이터에 직접 접근할 수 있는 메소드
    - 암시적으로 첫 인자로 클래스 객체가 전달
  - 형식
    - <호출할 메소드 이름> = classmethod(클래스내 정의한 메소드 이름)

# 연산자 중복

- 연산자 중복이란

- 사용자 정의 객체에서 필요한 연산자를 내장 타입과 형태와 동작이 유사하도록 재정의
- 연산자 중복을 위하여 두 개의 밑줄 문자가 앞뒤로 있는 메소드를 미리 정의함

- 예제 코드

```
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)
```

# 연산자 중복 – con'd

- 수치 연산자

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

# 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 코드

```
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
```

- 클래스 간의 관계 확인
  - 상속 관계인 두 클래스 간의 관계를 확인
    - issubclass(자식 클래스, 부모 클래스)

# 상속

- 다중 상속

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

- 예제 코드

```
-*- 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 상속과 이름 공간

인스턴스 객체 영역

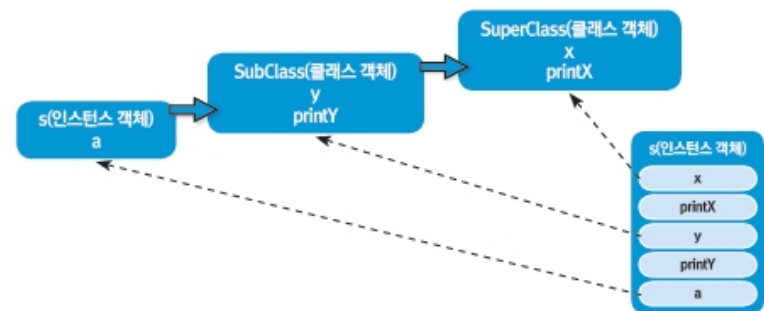
- > 클래스 객체간 상속을 통한 영역(자식 클래스 영역)
- > 부모 클래스 영역)
- > 전역 영역

- 예제 코드

```
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
```





# 모듈

# 모듈

- 여러 코드를 한데 묶어 다른 곳에서 재사용할 수 있는 코드 모음
  - 모듈 사용하기
  - 모듈 만들기
  - 모듈의 경로
  - 모듈 임포트
  - 모듈 임포트 파헤치기
  - 유용한 팁
  - 패키지

# 모듈 사용하기

- 현재 파이썬 3.0 버전에서는 대략 200개가 넘는 모듈을 지원
  - 문자열(string), 날짜(date), 시간(time), 십진법(decimal), 랜덤(random)
  - 파일(file), os, sqlite3, sys, xml, email, http 등등
- 간단하게 모듈을 사용 할 수 있음
- 모듈을 사용하는 이유
  - 코드의 재 사용성
  - 코드를 이름공간으로 구분하고 관리 할 수 있음.
  - 복잡하고 어려운 기능을 포함하는 프로그램을 간단하게 만들 수 있다.

# 모듈 import

- import : 모듈을 현재 이름공간으로 가져오는 역할

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

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

```
>>> dir(math)
['__doc__', '__name__', '__package__', 'acos',
'acosh', 'asin', 'asinh', 'atan', 'atan2',
'atanh', 'ceil', 'copysign', 'cos', 'cosh',
...
```

# 간단한 FTP 프로그램

- FTP로 서버에 접근해 파일 리스트를 가져오는 프로그램.

```
ftplib import FTP
>>> f>>> from tp = 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.
- 모듈은 일반적으로 <모듈이름>.py 으로 지정합니다.

# simpleset 모듈 만들기

- 텍스트 에디터를 이용해 교집합, 차집합, 합집합 함수를 만듭시다.
- simpleset.py 이름으로 저장하고 simpleset.py를 파이썬 라이브러리 디렉터리에 옮깁니다.
- import 명령을 이용해 simpleset 모듈을 가지고 옵니다.  
    >>> import simpleset
- 합집합 구하기  
    >>> 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
- 모듈 경로 탐색 순서.
  - 프로그램이 실행된 디렉터리
  - `PYTHONPATH` 환경 변수에 등록된 위치
  - 표준 라이브러리 디렉터리



# 모듈 импорт

- 모듈 안의 어트리뷰트 (함수, 데이터)들을 사용하려면 Imports를 해야 한다.
- Import 구문은 어디에서나 사용 가능 하다.
  - 함수, 제어문 내부에서도 import를 할 수 있다.
- Import <모듈이름>
  - 기본적인 импорт 방법
  - *모듈.이름* 형식으로 모듈 안의 데이터나 함수를 사용 할 수 있다.
  - 모듈은 импорт 하는 방법은 *import 모듈이름* 방법 이외에도 다른 방법이 있다.

# 모듈 импорт 방법

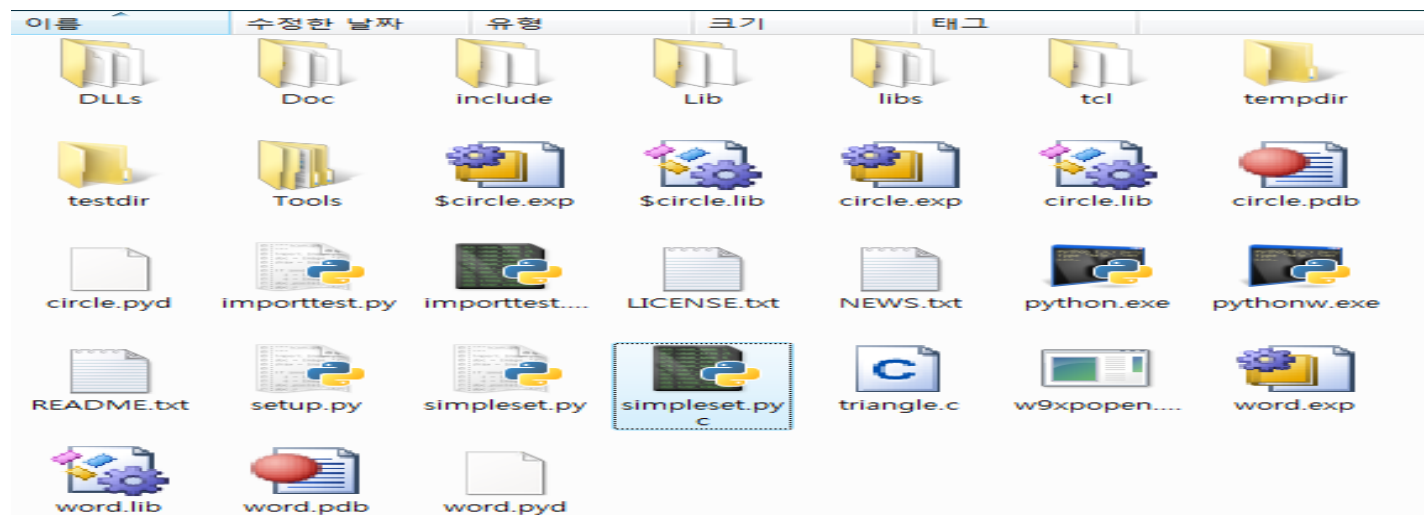
- from <모듈> import <어트리뷰트>  
    >>> from simpleset import union  
    >>> union([1, 2, 3], [3], [3, 4])  
    [1, 2, 3, 4]
- from <모듈> import \*
- import <모듈> as <별칭>
  - <모듈> 이름을 <별칭>으로 변경하여 임포트합니다.

# 모듈 импорт 파헤치기

- 임포트를 할 때, 해당 모듈의 바이트 코드가 있으면 이를 импорт 한다.
- 모듈을 импорт 하면 해당 모듈의 코드가 실행된다.
- 모듈이 импорт되면 메모리에 모듈 코드가 로딩되면 프로그램이나 파이썬 인터프리터가 끝나기 전까지 변경되지 않는다.

# 바이트 코드

- 일종의 중간 파일
  - 모듈의 임포트를 빠르게 해주는 역할
  - 바이트 코드가 이미 있으면 :  
모듈을 인터프리팅(Interpreting) 하지 않고 바로 바이트 코드 로딩
  - 바이트 코드가 없으면 :  
모듈을 인터프리팅 해서 바이트 코드를 생성
  - 바이트 코드가 생성된 모습



# 모듈이 메모리에 로딩 될 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__` 어트리뷰트
  - 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 실행하셨습니다")
else:
 print("임포트 하셨습니다")
```

# 패키지

- 모듈의 모음
  - 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
```

# Excel 제어하기

- xlwt
- OpenPyXL - 무난하다  
(<http://openpyxl.readthedocs.io/en/default/tutorial.html>)
- XlsxWriter - 엑셀 파일을 쓰는데 사용된다 읽기는 별도로 해야 한다  
<http://xlsxwriter.readthedocs.io/tutorial01.html>
- PyExcelerate
- win32 com client - anaconda패키지 설치시 64비트에서 제대로 동작하지 않는다. 차후로 개선될거라 본다



# Win32 com client 사용

- <https://sourceforge.net/projects/pywin32/> 사이트에서 버전에 맞추어 다운을 받아 설치해야 한다
- 64비트, python 3.6은 아직 지원안됨
- 예제)

```
from win32com.client import Dispatch
excel = Dispatch("Excel.Application")
wb = excel.Workbooks.Open("c:/Temp/Book1.xlsx")
sh = wb.Sheets[0]
sh.Cells(2,1).Value = 'pywin32'

print(wb.name, sh.name, sh.cells(1,2).value)

wb.save
wb.close
excel.Application.Quit()
del excel
```

# OpenPyXL 사용예

- <http://openpyxl.readthedocs.io/en/default/tutorial.html>
- 예)

```
#-*- coding: utf-8 -*-
from openpyxl import Workbook
wb = Workbook() #workbook 객체를 가져온다
ws = wb.active #현재 활성화된 시트 정보를 가져온다
ws['A1'] = 100 #A1셀에 값 100을 입력한다

ws.append([1, 2, 3]) #행에 데이터 추가하기

ws['A2'] = "Hello"

wb.save("sample.xlsx") #저장하기
```

# OpenPyXL 사용예

- 데이터 읽기

```
from openpyxl import load_workbook
wb = load_workbook(filename='sample.xlsx',
read_only=True)
ws = wb['Sheet']

for row in ws.rows:
 for cell in row:
 print(cell.value)
```

# Numpy 사용법

- Python 언어에서 기본으로 지원하지 않는 array (배열) 혹은 matrix (행렬) 의 계산을 쉽게 할 수 있다.
- 기계학습에서 많이 사용되는 해석학 및 선형대수학에 관련된 수식들을 Python 위에서 쉽게 프로그래밍 할 수 있다.

# Scipy

- Python의 라이브러리와 동시에 계산과학을 위한 시스템을 총칭.
- Numpy, Matplotlib (그래프를 그리는 데에 주로 사용됨) 외에도 IPython (Web에서 작동하는 Python의 인터랙티브 셸) 및 Pandas (데이터 저장 및 분석을 위한 라이브러리) 패키지들로 이루어져 있음.

# Matplotlib 예

```
import numpy as np
import pylab as pl
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]
pl.plot(x, y)
pl.show()
```

