

# Python

담당 : 이도연

교육기간 : 2019.5.18~19.6.8

# class

```
1 class MyClass:
2     pass
```

```
1 def __init__(self, width, height):
2     self.width = width
3     self.height = height
4
5     # private 변수 __area
6     self.__area = width * height
7
8     # private 메서드
9     def __internalRun(self):
10         pass
```

```
1 class Rectangle:
2     count = 0 # 클래스 변수
3
4     # 초기자(initializer)
5     def __init__(self, width, height):
6         # self.* : 인스턴스변수
7         self.width = width
8         self.height = height
9         Rectangle.count += 1
10
11     # 메서드
12     def calcArea(self):
13         area = self.width * self.height
14         return area
```

```
>>> class Person:
        pass

>>> p =Person()
>>> print(p)
<__main__.Person object at 0x02122BF0>
```

```
1  class Service:
2      ~~~~~
3      ~~~~~
4      ~~~~~
5      ~~~~~
6
7  ~~~~~
8  ~~~~~
9  ~~~~~
```

```
1  class Service:
2      secret = "이순신은 해군이다."
3      def sum(self, x, y ):
4          result = x +y
5          print("%s + %s = %s입니다." % (x,y,result))
6
7  myService = Service()
8  myService.sum(2,3)
9  print(myService.secret)
```

```
1  class Service:
2      secret ="이순신은 해군이다."
3      def setName(self, name):
4          self.name = name
5
6      def sum(self, x, y ):
7          result = x +y
8          print("%s님 %s + %s = %s입니다." % (self.name, x, y, result))
9
10 myService = Service()
11 myService.setName("happy DoYeon")
12 myService.sum(3,5)
```

```

1  class Calculator:
2      def __init__(self):
3          self.result = 0
4
5      def plus(self, num):
6          self.result += num
7          return self.result
8
9  cal1 = Calculator()
10 cal2 = Calculator()
11
12 print(cal1.plus(3))
13 print(cal1.plus(4))
14 print(cal2.plus(3))
15 print(cal2.plus(7))

```

```

1  class Rectangle:
2      count = 0 # 클래스 변수
3
4      def __init__(self, width, height):
5          self.width = width
6          self.height = height
7          Rectangle.count += 1
8
9      # 인스턴스 메서드
10     def calcArea(self):
11         area = self.width * self.height
12         return area
13
14     # 정적 메서드
15     @staticmethod
16     def isSquare(rectWidth, rectHeight):
17         return rectWidth == rectHeight
18
19     # 클래스 메서드
20     @classmethod
21     def printCount(cls):
22         print(cls.count)
23
24
25     # 테스트
26     square = Rectangle.isSquare(5, 5)
27     print(square) # True
28
29     rect1 = Rectangle(5, 5)
30     rect2 = Rectangle(2, 5)
31     rect1.printCount() # 2

```

```

1  def __add__(self, other):
2      obj = Rectangle(self.width + other.width, self.height + other.height)
3      return obj
4
5  # 사용 예
6  r1 = Rectangle(10, 5)
7  r2 = Rectangle(20, 15)
8  r3 = r1 + r2  # __add__()가 호출됨

```

```

1  # 객체 생성
2  r = Rectangle(2, 3)
3
4  # 메서드 호출
5  area = r.calcArea()
6  print("area = ", area)
7
8  # 인스턴스 변수 액세스
9  r.width = 10
10 print("width = ", r.width)
11
12 # 클래스 변수 액세스
13 print(Rectangle.count)
14 print(r.count)

```

```

1  r = Rectangle(2, 3)
2
3  Rectangle.count = 50
4  r.count = 10  # count 인스턴스 변수가 새로 생성됨
5
6  print(r.count, Rectangle.count)  # 10 50 출력

```

```

1 class Animal:
2     def __init__(self, name):
3         self.name = name
4     def move(self):
5         print("move")
6     def speak(self):
7         pass
8
9 class Dog (Animal):
10     def speak(self):
11         print("bark")
12
13 class Duck (Animal):
14     def speak(self):
15         print("quack")

```

```

1 dog = Dog("doggy") # 부모클래스의 생성자
2 n = dog.name # 부모클래스의 인스턴스변수
3 dog.move() # 부모클래스의 메서드
4 dog.speak() # 파생클래스의 멤버

```

```

1 animals = [Dog('doggy'), Duck('duck')]
2
3 for a in animals:
4     a.speak()

```

# `__init__()` - 초기화 메소드

```
class Person:
    def __init__(self, name):
        self.name = name
    def say_hi(self):
        print 'Hello, my name is', self.name

p = Person('Swaroop')
p.say_hi()
# The previous 2 lines can also be written as
# Person('Swaroop').say_hi()
```

---

실행 결과:

```
$ python oop_init.py
Hello, my name is Swaroop
```

---



# 상속(Inheritance)

Class 자손 클래스명(부모 클래스명)

```
1  class HousePark:
2      lastname='박'      # 클래스변수
3
4      def __init__(self, name):  # 초기값 주기, 생성자 함수라고도 함
5          self.fullName = self.lastname + name
6
7      def travel(self, where):
8          print("%s, %s여행을 가다." %(self.fullName, where))
9
10 # house1 = HousePark("해피도연")
11 # house1.travel("제주도")
12
13 class HouseLee(HousePark):
14     lastname = "Lee"
15
16 mySub = HouseLee("자손")
17 mySub.travel("방콕")
```

```

class SchoolMember:
    '''Represents any school member.'''
    def __init__(self, name, age):
        self.name = name
        self.age = age
        print '(Initialized SchoolMember: {})'.format(self.name)

    def tell(self):
        '''Tell my details.'''
        print 'Name:"{}" Age:"{}"'.format(self.name, self.age),

class Teacher(SchoolMember):
    '''Represents a teacher.'''
    def __init__(self, name, age, salary):
        SchoolMember.__init__(self, name, age)
        self.salary = salary
        print '(Initialized Teacher: {})'.format(self.name)

    def tell(self):
        SchoolMember.tell(self)
        print 'Salary: "{:d}"'.format(self.salary)

class Student(SchoolMember):
    '''Represents a student.'''
    def __init__(self, name, age, marks):
        SchoolMember.__init__(self, name, age)
        self.marks = marks
        print '(Initialized Student: {})'.format(self.name)

    def tell(self):
        SchoolMember.tell(self)
        print 'Marks: "{:d}"'.format(self.marks)

t = Teacher('Mrs. Shrividya', 40, 30000)

```

```
s = Student('Swaroop', 25, 75)
```

```

# prints a blank line
print

```

```

members = [t, s]
for member in members:
    # Works for both Teachers and Students
    member.tell()

```

---

### 실행 결과:

```

$ python oop_subclass.py
(Initialized SchoolMember: Mrs. Shrividya)
(Initialized Teacher: Mrs. Shrividya)
(Initialized SchoolMember: Swaroop)
(Initialized Student: Swaroop)

Name:"Mrs. Shrividya" Age:"40" Salary: "30000"
Name:"Swaroop" Age:"25" Marks: "75"

```

# 상속과 다형성

```
1 class Animal:
2     def __init__(self, name):
3         self.name = name
4     def move(self):
5         print("move")
6     def speak(self):
7         pass
8
9 class Dog (Animal):
10     def speak(self):
11         print("bark")
12
13 class Duck (Animal):
14     def speak(self):
15         print("quack")
```

```
1 dog = Dog("doggy") # 부모클래스의 생성자
2 n = dog.name # 부모클래스의 인스턴스변수
3 dog.move() # 부모클래스의 메서드
4 dog.speak() # 파생클래스의 멤버
```

```
1 animals = [Dog('doggy'), Duck('duck')]
2
3 for a in animals:
4     a.speak()
```

# Overriding

```
1  class HousePark:
2      lastname='박'      # 클래스변수
3
4      def __init__(self, name):  # 초기값 주기, 생성자 함수라고도 함
5          self.fullName = self.lastname + name
6
7      def travel(self, where):
8          print("%s, %s여행을 가다." %(self.fullName, where))
9
10 class HouseKang(HousePark):
11     lastname = "Kang"
12
13     def travel(self, where, day):
14         print("%s, %s여행 %d 갑니다." %(self.fullName, where, day))
15
16 mySub = HouseKang("강자손")
17 mySub.travel("파타야", 5)
```

# Operation Overloading

```
1  class HousePark:
2      lastname='박'      # 클래스변수
3      def __init__(self, name):  # 초기값 주기, 생성자 함수라고도 함
4          self.fullName = self.lastname + name
5      def travel(self, where):
6          print("%s, %s여행을 가다." %(self.fullName, where))
7      def love(self, other) :
8          print("%s, %s 사랑에 빠졌네" %(self.fullName, other.fullName))
9      def __add__(self, other):
10         print("%s, %s 결혼했네. " % (self.fullName, other.fullName))
11
12  class HouseKang(HousePark):
13      lastname = "Kang"
14      def travel(self, where, day):
15          print("%s, %s여행 %d 갑니다." %(self.fullName, where, day))
16
17  myParent = HousePark("도연")
18  mySub = HouseKang("자손")
19
20  myParent.love(mySub)
21  myParent + mySub
```

# 모듈

```
import 모듈1[, 모듈2[, ... 모듈N]
```

```
1 | import math
2 | n = math.factorial(5)
```

```
1 | # factorial() 함수를 f()로 사용 가능
2 | from math import factorial as f
3 | n = f(5) / f(3)
```

```
1 | # 여러 함수를 import
2 | from math import (factorial,acos)
3 | n = factorial(3) + acos(1)
4 |
5 | # 모든 함수를 import
6 | from math import *
7 | n = sqrt(5) + fabs(-12.5)
```

```
>>> import sys
>>> sys.path
['', 'C:\\Python35\\Lib\\idlelib', 'C:\\Python35\\python35.zip', 'C:\\Python35\\DLLs', 'C:\\Python35\\lib', 'C:\\Python35', 'C:\\Python35\\lib\\site-packages']
>>> sys.path[0]
''
```

현재 검색경로를 표시함

첫번째는 빈 문자열로 현재 디렉토리를 가리킴

```
>>> sys.path.append('C:\\PySrc')
>>>
```

새 폴더를 검색경로에 추가함

# 모듈 작성

```
1 # mylib.py
2 def add(a, b):
3     return a + b
4
5 def subtract(a, b):
6     return a - b
```



```
1 # exec.py
2 from mylib import *
3
4 i = add(10, 20)
5 i = subtract(20, 5)
```

```
1 # run.py
2 import sys
3 def openurl(url):
4     #..본문생략..
5     print(url)
6
7 if __name__ == '__main__':
8     openurl(sys.argv[1])
```



```
$ python3.5 run.py google.com
google.com
```

```
$ python3.5
>>> from run import *
>>> openurl('google.com')
```

```
1  def hap(a, b):  
2      return a + b  
3  def safe_hap(a, b):  
4      if type(a) != type(b) :  
5          print("타입이 달라서 더할 수 없습니다.")  
6          return  
7      else :  
8          result = hap(a, b)  
9          return result
```

```
>>> from module import *  
>>> safe_hap(3, 'a')  
타입이 달라서 더할 수 없습니다.  
>>> import math  
>>> n = math.factorial(5)  
>>> print(n)  
120  
>>> hap(2,3)  
5  
>>> safe_hap(4, 5)  
9
```



# 표준 라이브러리 - sys 모듈

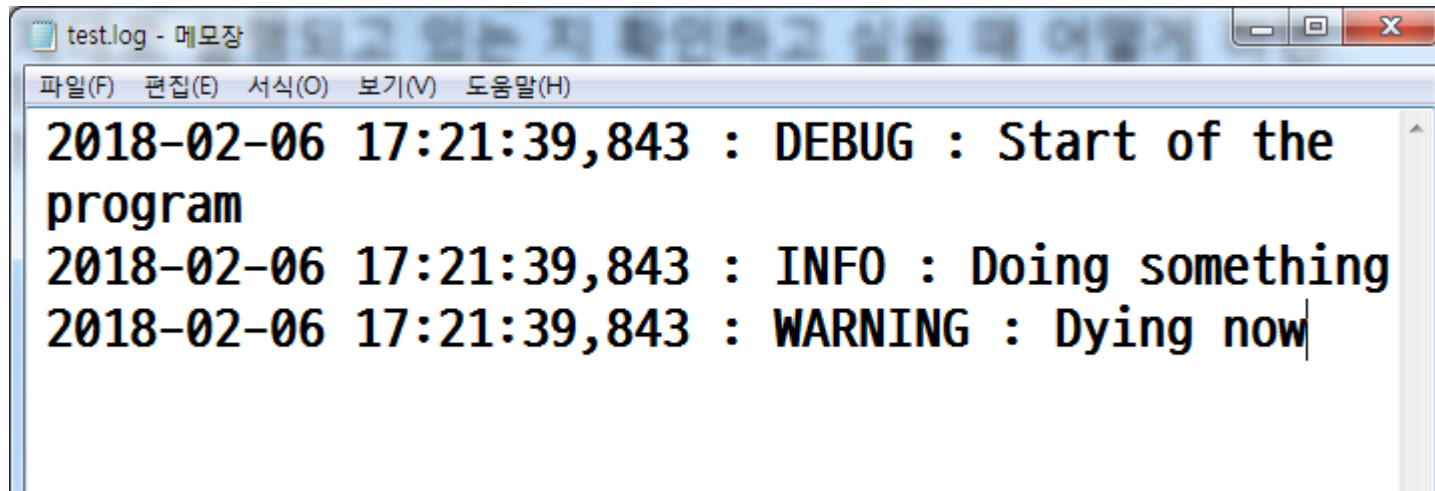
- sys

- ✓ sys 모듈에는 시스템의 기능을 다루는 여러 함수들이 들어 있습니다.  
예를 들어 sys.argv 리스트에는 명령줄 인수들이 들어 있습니다.

```
>>> import sys
>>> sys.version_info
sys.version_info(major=3, minor=6, micro=4, releaselevel='final', serial=0)
>>> sys.version_info.major == 2
False
```

# logging 모듈

- 디버그를 할 때 중간에 변수들의 내용 등을 출력하고 싶거나 혹은 실행시 중요한 메시지를 어딘가에 저장해 두게 하여 프로그램이 제대로 실행되고 있는 지 확인하고 싶을 때 어떻게 하면 좋을까요? 어떻게 이러한 메시지들을 "어딘가에 저장해" 둘 수 있을까요? 이를 위해 logging 모듈을 사용합니다.



```
test.log - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
2018-02-06 17:21:39,843 : DEBUG : Start of the
program
2018-02-06 17:21:39,843 : INFO : Doing something
2018-02-06 17:21:39,843 : WARNING : Dying now
```

```
1  import os, platform, logging
2  if platform.platform().startswith('Windows'):
3      logging_file = os.path.join(os.getenv('HOMEDRIVE'),
4      os.getenv('HOMEPATH'), 'test.log')
5  else:
6      logging_file = os.path.join(os.getenv('HOME'), 'test.log')
7      print ("Logging to", logging_file)
8  logging.basicConfig(
9      level=logging.DEBUG,
10     format='%(asctime)s : %(levelname)s : %(message)s',
11     filename = logging_file,
12     filemode = 'w',
13 )
14 logging.debug("Start of the program")
15 logging.info("Doing something")
16 logging.warning("Dying now")
```

# > > python module - 표준라이브러리

- <https://docs.python.org/2/library/pdb.html>
- <https://docs.python.org/2/library/argparse.html>
- <https://docs.python.org/2/library/re.html>
- <https://docs.python.org/2/library/>
- <https://docs.python.org/3.7/>
- <https://docs.python.org/3.7/library/exceptions.html>

## >> 예외처리

```
1  try:
2      문장1
3      문장2
4  except:
5      예외처리
6  finally:
7      마지막에 항상 수행
```

```
1  def calc(values):
2      sum = None
3      try:
4          sum = values[0] + values[1] + values[2]
5      except (IndexError, ValueError):
6          print('오류발생')
7
8      print(sum)
```

```
1  def calc(values):
2      sum = None
3      # try...except...else
4      try:
5          sum = values[0] + values[1] + values[2]
6      except IndexError as err:
7          print('인덱스에러')
8      except Exception as err:
9          print(str(err))
10     else:
11         print('에러없음')
12     finally:
13         print(sum)
14
15     # 테스트
16     calc([1, 2, 3, 6]) # 출력: 에러없음 6
17     calc([1, 2])      # 출력: 인덱스에러 None
```

## >> 에러 무시와 에러 생성

```
1  # pass 를 사용한 예
2  try:
3      check()
4  except FileNotFoundError:
5      pass
6
7  # raise 를 사용한 예
8  if total < 0:
9      raise Exception('Total Error')
```

## >> 파일 에러 처리 예제

```
1  try:
2      fp = open("test.txt", "r")
3      try:
4          lines = fp.readlines()
5          print(lines)
6      finally:
7          fp.close()
8  except IOError:
9      print('파일에러')
```

```
1  with open('test.txt', 'r') as fp:
2      lines = fp.readlines()
3      print(lines)
```

## >> 예외 설명

예외	설명
BaseException	모든 다른 예외들의 근원(root)이 되는 예외
GeneratorExit	반복의 종료를 위한 것으로서 generators의 close() 메소드에 의하여 일어남
KeyboardInterrupt	끼어들기 키에 의하여 일어남
SystemExit	프로그램 종료
Exception	모든 non-exiting 예외의 근원
StopIteration	반복(iteration)의 종료를 위하여 발생
StandardError	모든 내장 예외의 기본 클래스
ArithmeticError	모든 수치연산 관련 예외의 기본이 됨
FloatingPointError	부동소수 연산이 실패할 때 발생
OverflowError	너무 큰 수치 연산
ZeroDivisionError	나눗셈 또는 나머지 연산 수행시 0으로 나눌 경우
AssertionError	단정문이 실패할 때 발생
AttributeError	속성의 참조 또는 할당 실패
EnvironmentError	파이썬의 외부적인 요인에 의한 오류



IOError	입출력 연산 오류
OSError	os 모듈에서 발생하는 오류
EOFError	파일의 끝을 지나쳐서 input() 또는 raw_input()을 시도
ImportError	들여올(import) 모듈이나 이름을 찾지 못함
LookupError	IndexError와 KeyError의 기본 클래스
IndexError	순서 색인이 범위를 벗어남
KeyError	존재하지 않는 사상(사전)의 키를 참조
MemoryError	기억장소 소진
NameError	지역 또는 전역의 이름을 찾는 데 실패
UnboundLocalError	할당되지 않은 지역 변수를 참조
ReferenceError	쓰레기 수거된 개체에 대한 접근을 시도
RuntimeError	모든 경우에 대한 쓸모 없는 오류
NotImplementedError	기능이 구현되지 않은 경우에 발생
SyntaxError	구문분석기가 구문 오류를 만남
IndentationError	구문분석기가 들여쓰기 문제를 만남
TabError	탭과 스페이스를 잘못 섞어 씀
SystemError	치명적이지 않은 번역기 오류
TypeError	연산자 또는 함수에 부적당한 형이 전달됨
ValueError	TypeError 또는 다른 명확한 오류에 해당하지 않는 인자 오류
Warning	모든 경고의 기본

## >> 경고 분류

경고	설명
Warning	최상위 경고 클래스
UserWarning	사용자 정의 경고
DeprecationWarning	사용금지된(deprecated) 기능에 대한 경고
SyntaxWarning	구문 문제
RuntimeWarning	실행시간 문제
FutureWarning	앞으로의 릴리스에서 변경될 예정인 특정 기능에 대한 경고

# >> 경고 함수

함수	설명
<code>warn(message[, category[, stacklevel]])</code>	경고를 발행. 매개변수는 메시지 문자열, 선택적인 경고 카테고리, 그리고 호출하는 함수 혹은 함수 자체 중 어느 스택 프레임으로부터 경고가 발생하였는지를 알려주는 선택적인 스택 레벨을 포함한다.
<code>warn_explicit(message, category, filename, lineno[, module[, registry]])</code>	이것은 보다 세부적인 경고 메시지를 제공하며 <code>category</code> 를 필수 파라미터로 만든다. <code>filename</code> , <code>lineno</code> 및 <code>module</code> 은 경고가 어디에 위치하는지 알려준다. <code>registry</code> 는 현재 활성화된 모든 경고 필터를 나타낸다.
<code>showwarning(message, category, filename, lineno[, file])</code>	경고를 파일에 기록할 수 있도록 해준다.
<code>formatwarning(message, category, filename, lineno)</code>	경고에 대한 서식화된 문자열을 생성한다.
<code>simplefilter(action[, category[, lineno[, append]]])</code>	경고 필터에 대한 정렬된 목록에 간단한 엔트리를 삽입한다. <code>simplefilter</code> 는 카테고리나 행 번호가 일치하는 한 어떤 모듈의 어떤 메시지도 항상 <code>match</code> 하므로 정규표현식을 필요로 하지 않는다. 아래에서 설명할 <code>filterwarnings()</code> 는 정규표현식을 사용하여 경고를 <code>match</code> 한다.
<code>resetwarnings()</code>	모든 경고 필터를 리셋.
<code>filterwarnings(action[, message[, category[, module[, lineno[, append]]]])</code>	경고 필터 목록에 엔트리를 추가한다. 경고 필터는 경고의 동작을 수정할 수 있도록 해준다. 경고 필터의 <code>action</code> 은 표 7-4에 나열된 것 중 하나가 될 수 있고, <code>message</code> 는 정규표현식, <code>category</code> 는 발행될 경고의 종류, <code>module</code> 은 정규표현식이 될 수 있고, <code>lineno</code> 는 모든 행에 대하여 일치하는 행 번호이고, <code>append</code> 는 필터가 모든 필터의 목록에 추가되어야 할 지를 규정한다.

BaseException	모든 예외의 최상위 예외
└ SystemExit	프로그램을 종료하는 명령이 실행되었을 때
└ KeyboardInterrupt	Control-C 키가 입력되었을 때
└ Exception	대부분의 예외의 상위 예외
└ ArithmeticError	수의 연산과 관련된 문제
└ ZeroDivisionError	수를 0으로 나누려 할 때
└ AssertionError	assert 문에 의해 발생
└ AttributeError	(모듈·클래스·인스턴스에서) 잘못된 속성을 가리킬 때
└ EOFError	(파일에서) 읽어들일 데이터가 더이상 없을 때
└ ImportError	모듈을 임포트할 수 없을 때
└ ModuleNotFoundError	임포트할 모듈을 찾을 수 없을 때
└ LookupError	잘못된 인덱스·키로 인덱싱할 때
└ IndexError	(시퀀스에서) 잘못된 인덱스로 인덱싱할 때
└ KeyError	(매핑에서) 잘못된 키로 인덱싱할 때
└ NameError	잘못된 이름(변수)을 가리킬 때
└ OSError	운영 체제의 동작과 관련된 다양한 문제
└ ChildProcessError	하위 프로세스(프로그램이 실행한 외부 프로그램)에서 오류 발생
└ FileExistsError	이미 존재하는 파일·디렉토리를 새로 생성하려 할 때
└ FileNotFoundError	존재하지 않는 파일·디렉토리에 접근하려 할 때
└ IsADirectoryError	파일을 위한 명령을 디렉토리에 실행할 때
└ NotADirectoryError	디렉토리를 위한 명령을 파일에 실행할 때
└ PermissionError	명령을 실행할 권한이 없을 때
└ TimeoutError	명령의 수행 시간이 기준을 초과했을 때
└ RuntimeError	다른 분류에 속하지 않는 실행시간 오류
└ NotImplementedError	내용 없는 메서드가 호출되었을 때
└ RecursionError	함수의 재귀 호출 단계가 허용한 한계를 초과했을 때
└ SyntaxError	구문 오류
└ IndentationError	들여쓰기가 잘못되었을 때
└ TabError	들여쓰기에 탭과 스페이스를 번갈아가며 사용했을 때
└ TypeError	연산·함수가 계산할 데이터의 유형이 잘못되었을 때
└ ValueError	연산·함수가 계산할 데이터의 값이 잘못되었을 때
└ UnicodeError	유니코드와 관련된 오류
└ Warning	심각한 오류는 아니나 주의가 필요한 사항에 관한 경고

## > > 필터 역할

필터	역할
'always'	경고 메시지를 항상 출력
'default'	경고가 발생하는 곳마다 한 번 씩 출력
'error'	경고를 예외로 변환
'ignore'	경고를 무시
'module'	경고가 발생하는 모듈마다 한 번 씩 경고
'once'	단 한 번만 경고

# 경고에 대한 예외를 일으키기 위한 간단한 경고 필터를 설정

```
>>> warnings.simplefilter('error', UserWarning)
>>> warnings.warn('This will be raised as an exception')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/Applications/Jython/jython2.5.1rc2/Lib/warnings.py", line 63, in warn
    warn_explicit(message, category, filename, lineno, module, registry,
  File "/Applications/Jython/jython2.5.1rc2/Lib/warnings.py", line 104, in warn_explicit
    raise message
UserWarning: This will be raised as an exception
```

# resetwarnings()를 사용하여 모든 활성화된 필터를 끄기

```
>>> warnings.resetwarnings()
>>> warnings.warn('This will not be raised as an exception')
__main__:1: UserWarning: This will not be raised as an exception
```

# 경고를 거르기 위하여 정규 표현식을 사용

# 여기서는, "one"이라는 단어를 포함하는 모든 경고를 무시

```
>>> warnings.filterwarnings('ignore', '.*one*.')
>>> warnings.warn('This is warning number zero')
__main__:1: UserWarning: This is warning number zero
>>> warnings.warn('This is warning number one')
>>> warnings.warn('This is warning number two')
__main__:1: UserWarning: This is warning number two
>>>
```

# 내장 함수

<code>abs()</code>	<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>
<code>all()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>any()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>ascii()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>bin()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bool()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>breakpoint()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	

# >> 외장 함수

## 1. sys

1. 명령 행에서 인수 전달하기 - sys.argv
2. 강제로 스크립트 종료하기 - sys.exit
3. 자신이 만든 모듈 불러와 사용하기 - sys.path

## 2. pickle

## 3. os

1. 내 시스템의 환경 변수값을 알고 싶을 때 - os.environ
2. 디렉터리 위치 변경하기 - os.chdir
3. 디렉터리 위치 리턴받기 - os.getcwd
4. 시스템 명령어 호출하기 - os.system
5. 실행한 시스템 명령어의 결과값 리턴받기 - os.popen
6. 기타 유용한 os 관련 함수

## 4. shutil

1. 파일 복사하기 - shutil.copy(src, dst)

## 5. glob

1. 디렉터리에 있는 파일들을 리스트로 만들기 - glob(pathname)

## 6. tempfile

## 7. time

1. time.time
2. time.localtime
3. time.asctime
4. time.ctime
5. time.strftime
6. time.sleep

## 8. calendar

1. calendar.weekday
2. calendar.monthrange

## 9. random

## 10. webbrowser

## 11. namedtuple

## 12. defaultdict