클래스(Class)

클래스 정의

```
class cls_name :
    block
class cls_name( parent_cls_names ) :
    block
```

객체 생성

```
obj_name = cls_name( arguments )
```

클래스 변수

정의

```
var = init_value
self.var = value
```

설정

```
obj_name.var = value
self.var = value
```

멤버 함수

```
def func_name( self, arguments ) :
    block
```

클래스 블록 내에 정의
반드시 self(객체 자신)를포함해야 함.
함수 실행 : self 는 생략해도 됨

```
obj_name.func_name( arguments )
__init__( self, arguments )
__del__( self )
__add__( self, other )
...
```

```
class HousePark :
    firstname = "박"
    def __init__( self, name ) :
        self.fullname = self.firstname + name
    def travel( self,  where ) :
        print("%s, %s여행." % (self.fullname, where))
    def love( self, other ) :
        print("%s, %s 사랑" % (self.fullname, other.fullname) )
    def __add__( self, other ) :
        print("%s, %s 결혼" % (self.fullname, other.fullname) )
    def __del__( self ) :
        print("%s 죽네" % self.fullname)

class HouseKim( HousePark ):
    firstname = "김"
    def travel( self, where, day ) :
        print("%s, %s여행 %d일" % (self.fullname, where, day) )

pey = HousePark("응용")
julliet = HouseKim("줄리엣")
pey.love(julliet)
pey + julliet
```

모듈(Module)

모듈 로드

```
import mod_name
from mod_name import mod_func_name,...
    from mod_name import *
reload( mode_name )
```

모듈 참조

```
mod_name.var
mod_name.mod_func_name
```

메인 모듈 여부 판단

```
if __name__=="__main__" :
    block
```

모듈 경로 설정

```
import sys
sys.path.append( module_path )
```

```
# mod2.py
PI = 3.141592
class Math:
    def solv(self, r):
        return PI * (r ** 2)
    def sum(a, b):
        return a+b
if __name__=="__main__":
    print(PI a = Math())
    print(a.solv(2))
    print(sum(PI, 4.4))
```

```
# modtest.py
import mod2
result = mod2.sum(3, 4)
print(result)
```

```
# config.py
a='doo'
```

```
# reload_test.py
import config
import imp
f = open("config.py", 'w')
f.write("a = 'foo'")
f.close()
imp.reload(config)
c = config.a
print(c)
```

예외 처리(Exception)

예외처리

```
try:
    block
except :
    block

try:
    block
except error :
    block

try:
    block
except error as var :
    block
```

```
try:
    4 / 0
except ZeroDivisionError as e :
    print(e)

try:
    f = open("나없는파일.txt", 'r')
except IOError
    print("쓰기모드로 파일을 엽니다.")
    f = open("나없는파일.txt", 'w')
```

내장함수

함수	설명
abs	절대값
apply	함수 적용
chr	아스키코드값을 받아 문자출력
cmp	객체 비교
dir	객체의 클래스 변수/함수 나열
divmod	몫과 나머지. (div,mod)
eval	문자열을 프로그램으로 인식 계산
execfile	파이썬 파일 실행
filter	리스트 필터링. filter(func, list)
hex	16진수를 표시하는 문자열로 변환
id	객체 아이디
input	사용자 입력 . input(prompt)
int	정수로 변환
isinstance	클래스의 객체 여부 판단
lambda	람다 함수
len	리스트 길이
list	리스트로 변환

함수	설명
long	정수로 변환
map	리스트에 함수 적용 리턴
max	최대값
min	최소값
oct	8진수 문자열로 변환
open	파일 열기
ord	아스키값 리턴
pow	제곱승
range	해당 범위의 리스트를 생성
raw_input	사용자 입력
reduce	리스트의 모든 값을 함수에 적용
reload	모듈 리로드
repr	객체를 문자열로 변환
str	객체를 문자열로 변환,eval 사용x
tuple	튜플로 변환
type	타입을 리턴

파이썬 날개달기

클래스, 모듈, 예외 처리

클라스 (class)

- 키워드 class, 식별자 class명, delimiter :
- class 내 method에 있는 첫번째 인자와 속성에는 self가 붙음

클래스는 도대체 왜 필요한가?
두 개의 계산기가 필요한 경우
클래스를 이용한 계산기
클래스의 구조
사칙연산 클래스 1/2/3/4
__init__ 메서드 1/2
클래스의 상속
메서드 오버라이딩
연산자 오버로딩 1/2

1 파이썬 프로그래밍의 핵심, 클래스

클래스는 도대체 왜 필요한가?(한 개의 계산기만 필요한 경우)

```
result = 0

def adder(num):
    global result
    result += num
    return result
```

이전값을 저장했다가 더한 후 return
print(adder(3)) - 3 return
print(adder(4)) - 7 return

1 파이썬 프로그래밍의 핵심, 클래스

두 개의 계산기가 필요한 경우

```
result1 = 0
result2 = 0

def adder1(num):
    global result1
    result1 += num
    return result1

def adder2(num):
    global result2
    result2 += num
    return result2
```

print(adder1(3)) - 3
print(adder1(4)) - 7
print(adder2(3)) - 3
print(adder2(7)) - 10

1 파이썬 프로그래밍의 핵심, 클래스

클래스를 이용한 계산기 (class로 한 개, object 두 개)

```
class Calculator:                    키워드 class, 식별자 class명, delimiter :
    def __init__(self):
        self.result = 0

    def adder(self, num):            print(cal1.add(3)) - 3
        self.result += num           print(cal1.add(4)) - 7
        return self.result           print(cal2.add(3)) - 3
                                     print(cal2.add(7)) - 10
cal1 = Calculator()
cal2 = Calculator()
```

1 파이썬 프로그래밍의 핵심, 클래스

클래스의 구조

```
class 클래스이름[(상속 클래스명)]:  키워드 class, 식별자 class명, delimiter :
    <클래스 변수 1>      delimiter (, 식별자 class명, delimiter )
    <클래스 변수 2>          class내 변수 – method 밖에서는 self가 없음
                             method 내에서는 self가 있음
    ...
    def 클래스함수1(self[, 인수1, 인수2,,,]):
        <수행할 문장 1>      class내 함수 – 만드시 self가 첫번째 인자
        <수행할 문장 2>
        ...
    def 클래스함수2(self[, 인수1, 인수2,,,]):
        <수행할 문장1>
        <수행할 문장2>
        ...
```

1 파이썬 프로그래밍의 핵심, 클래스

사칙연산 클래스 1

```
>>> class FourCal:      } 키워드 class, 식별자 class명, delimiter :
...     pass            } 가장 간단한 class
...
>>>
```

```
>>> a = FourCal()       식별자 class명, delimiter (, delimiter )
>>> type(a)                  a는 객체(object)
<class '__main__.FourCal'>   a는 FourCal의 인스턴스(instance)
```

1 파이썬 프로그래밍의 핵심, 클래스

사칙연산 클래스 2 (method에 인자 전달)

```
>>> a.setdata(4, 2)      FourCal.setdata(a, 4, 2)
```

```
>>> class FourCal:
...     def setdata(self, first, second):   } method
...         self.first = first
...         self.second = second
...                                  a.first = first
>>>                                  a.second = second
```

1 파이썬 프로그래밍의 핵심, 클래스

사칙연산 클래스 3

```
>>> a = FourCal()
>>> a.setdata(4, 2)       FourCal.setdata(a, 4, 2)
>>> print(a.sum())
6
```

1 파이썬 프로그래밍의 핵심, 클래스

사칙연산 클래스 4

```
>>> class FourCal:
...     def setdata(self, first, second):
...         self.first = first
...         self.second = second
...     def sum(self):
...         result = self.first + self.second
...         return result
...
>>>
```

1 파이썬 프로그래밍의 핵심, 클래스

__init__ 메서드로 초깃값을 설정 1

```
>>> class HousePark:
...     lastname = "박"
...     def __init__(self, name):
...         self.fullname = self.lastname + name
...     def travel(self, where):
...         print("%s, %s여행을 가다." %
(self.fullname, where))
...
>>>
```

1 파이썬 프로그래밍의 핵심, 클래스

__init__ 메서드로 초깃값을 설정 2

```
>>> pey = HousePark()
TypeError: __init__() takes exactly 2 arguments (1 given)
```

```
>>> pey = HousePark("응용")
```

1 파이썬 프로그래밍의 핵심, 클래스

클래스의 상속 부모 class 명

```
>>> class HouseKim(HousePark):
...     lastname = "김"
...
>>>
```

```
>>> juliet = HouseKim("줄리엣")
>>> juliet.travel("독도")
김줄리엣, 독도여행을 가다.
```

1 파이썬 프로그래밍의 핵심, 클래스

메서드 오버라이딩

```
>>> class HouseKim(HousePark):
...     lastname = "김"
...     def travel(self, where, day):
...         print("%s, %s여행 %d일 가네." % (self.fullname, where, day))
...
>>>
```

```
>>> juliet = HouseKim("줄리엣")
>>> juliet.travel("독도", 3)
김줄리엣, 독도여행 3일 가네.
```

1 파이썬 프로그래밍의 핵심, 클래스

연산자 오버로딩 1

```
>>> pey = HousePark("응용")
>>> juliet = HouseKim("줄리엣")
>>> pey + juliet
박응용, 김줄리엣 결혼했네
>>>
```

연산자 오버로딩
연산자 오버로딩이란 연산자(+, -, *, /, , ,)등을 인스턴스끼리 사용
인스턴스끼리 연산자 기호를 사용 : pey + juliet

1 파이썬 프로그래밍의 핵심, 클래스

연산자 오버로딩 2

```
class HousePark:
    lastname = "박"
    def __init__(self, name):
        self.fullname = self.lastname + name
    def travel(self, where):
        print("%s, %s여행을 가다." % (self.fullname, where))
    def __add__(self, other):
        print("%s, %s 결혼했네" % (self.fullname, other.fullname))
```

모듈 (module)

- 복합구문 여러 개의 구문으로 이루어짐
- 조건문의 참/거짓에 따라서 실행되는 문장이 달라지는 구문
- 키워드 IF, 조건식 참거짓, : delimiter, 들여쓰기 else elif

모듈 만들고 불러 보기
from 모듈이름 import 모듈함수
sys.path.append
PYTHONPATH 환경 변수 사용하기

2 모듈

모듈 만들고 불러 보기

```
# C:\Python\Mymodules\mod1.py
def sum(a, b):
    return a + b
```

```
cd C:\Python\Mymodules
```

```
>>> import mod1
>>> print(mod1.sum(3,4))
7
```

2 모듈

from 모듈이름 import 모듈함수

```
>>> from mod1 import sum
>>> print(sum(3, 4))
7
```

2 모듈

sys.path.append

set PYTHONPATH=C:₩xxx
다른 디렉토리에 있는 파일을 module로 사용

```
>>> import sys
>>> sys.path
['', 'C:\\Windows\\SYSTEM32\\python35.zip',
'c:\\Python35\\DLLs',
'c:\\Python35\\lib', 'c:\\Python35',
'c:\\Python35\\lib\\site-packages']
```

```
sys.path.append("C:/Python/Mymodules")
```

2 모듈

PYTHONPATH 환경 변수 사용하기

```
C:\Users\home>set PYTHONPATH=C:\Python\Mymodules
C:\Users\home>python
Python 3.5.1 (v3.5.1:37a07cee5969, Dec 6 2015, 01:54:25) [MSC v.1900 64 bit (AM...
Type "help", "copyright", "credits" or "license" for more information.
>>> import mod1
```

패키지 (package)

- 복합구문 여러 개의 구문으로 이루어짐
- 조건문의 참/거짓에 따라서 실행되는 문장이 달라지는 구문
- 키워드 IF, 조건식 참거짓, : delimiter, 들여쓰기 else elif

가상의 game 패키지 예
테스트를 위해 패키지 만들기
패키지 안의 함수 실행하기 1/2
__all__
relative 패키지

3 패키지

가상의 game 패키지 예

```
game/
    __init__.py
    sound/
        __init__.py
        echo.py
        wav.py
    graphic/
        __init__.py
        screen.py
        render.py
    play/
        __init__.py
        run.py
        test.py
```

3 패키지

테스트를 위해 패키지 만들기

```
C:/Python/game/__init__.py
C:/Python/game/sound/__init__.py
C:/Python/game/sound/echo.py
C:/Python/game/graphic/__init__.py
C:/Python/game/graphic/render.py
```

```
# echo.py
def echo_test():
    print ("echo")
```

```
# render.py
def render_test():
    print ("render")
```

3 패키지

패키지 안의 함수 실행하기 1

```
C:\> set PYTHONPATH=C:/Python
C:\> python
Python 3.5.1 (v3.5.1:37a07cee5969, Dec 6 2015, 01:54:25) [MSC v.1900 64 bit (AM...
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

3 패키지

패키지 안의 함수 실행하기 2

```
>>> import game.sound.echo
>>> game.sound.echo.echo_test()
echo
```

```
>>> from game.sound import echo
>>> echo.echo_test()
echo
```

```
>>> from game.sound.echo import echo_test
>>> echo_test()
echo
```

3 패키지

__all__

```
>>> from game.sound import *
>>> echo.echo_test()
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
NameError: name 'echo' is not defined
```

```
# C:/Python/game/sound/__init__.py
__all__ = ['echo']
```

3 패키지

relative 패키지

```
# render.py
from ..sound.echo import echo_test

def render_test():
    print ("render")
    echo_test()
```

예외처리 (exception)

- 복합구문 여러 개의 구문으로 이루어짐
- 조건문의 참/거짓에 따라서 실행되는 문장이 달라지는 구문
- 키워드 IF, 조건식 참거짓, : delimiter, 들여쓰기 else elif

오류는 어떤 때 발생하는가?
try, except문
try .. else
try .. finally
오류 회피하기
오류 일부러 발생시키기

4 예외처리

오류는 어떤 때 발생하는가?

```
>>> f = open("나없는파일", 'r')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: ' 나없는파일'
```

4 예외처리

try, except문

```
try:
    ...
except [발생 오류[as 오류 메시지 변수]]:
    ...
```

```
try:
    4 / 0
except ZeroDivisionError as e:
    print(e)
```

4 예외처리

try .. else

```
try:
    f = open('foo.txt', 'r')
except FileNotFoundError as e:
    print(str(e))
else:
    data = f.read()
    f.close()
```

4 예외처리

try .. finally

```
f = open('foo.txt', 'w')
try:
    # 무언가를 수행한다.
finally:
    f.close()
```

4 예외처리

오류 회피하기

```
try:
    f = open("나없는파일", 'r')
except FileNotFoundError:
    pass
```

4 예외처리

오류 일부러 발생시키기

```
class Bird:
    def fly(self):
        raise NotImplementedError
```

```
class Eagle(Bird):
    def fly(self):
        print("very fast")

eagle = Eagle()
eagle.fly()
```

내장 함수 (Built-in Functions)

5 내장함수

abs

```
>>> abs(3)
3
>>> abs(-3)
3
>>> abs(-1.2)
1.2
```

5 내장함수

all

```
>>> all([1, 2, 3])
True
>>> all([1, 2, 3, 0])
False
```

false

반복 가능한 자료형의 요소가 전부 참 - 참
하나라도 거짓 - 거짓

리스트, 튜플, 문자열, 딕셔너리, 집합

5 내장함수

any

```
>>> any([1, 2, 3, 0])
True
>>> any([0, ""])
False
```

반복 가능한 자료형의 요소가 하나라도 참 - 참
전부 거짓 - 거짓

5 내장함수

chr (ASCII -> 문자)

```
>>> chr(97)
'a'
>>> chr(48)
'0'
```

a = 97
b = chr(a)
'a'
c = odr(b)

5 내장함수

dir (객체의 모든 속성과 method 출력)

```
>>> dir([1, 2, 3])
['append', 'count', 'extend', 'index', 'insert', 'pop',...]
>>> dir({'1':'a'})
['clear', 'copy', 'get', 'has_key', 'items', 'keys',...]
```

5 내장함수

divmod(a, b) a/b 몫과 나머지 튜플로 출력

```
>>> divmod(7, 3)
(2, 1)
>>> divmod(1.3, 0.2)
(6.0, 0.09999999999999978)
```

5 내장함수

enumerate

```
>>> for i, name in enumerate(['body', 'foo',
'bar']):
...     print(i, name)
...
0 body
1 foo
2 bar
```

순서가 있는 자료형을 입력 받아
인덱스값을 포함하는 객체 return (0부터 시작)

5 내장함수

eval(expression)

```
>>> eval('1+2')
3
>>> eval("'hi' + 'a'")
'hia'
>>> eval('divmod(4, 3)')
(1, 1)
```

실행가능한 문자열(식)을 입력받아
문자열을 실행한 결과 return

5 내장함수

filter

```
def positive(l):
	result = []
	for i in l:
		if i>0:
			result.append(i)
	return
print(positive([1, -3, 2, 0, -5, 6])
```

```
def positive(x):
    return x > 0
print(list(filter(positive, [1, -3, 2, 0, -5, 6])))
```

return이 참인 것만 걸러내 준다

```
>>> print(list(filter(lambda x: x > 0, [1, -3, 2, 0, -5, 6])))
```

5 내장함수

hex(x) (정수 -> 16진수)

```
>>> hex(234)
'0xea'
>>> hex(3)
'0x3'
```

5 내장함수

id(object) (고유주소 반환)

```
>>> a = 3
>>> id(3)
135072304
>>> id(a)
135072304
>>> b = a
>>> id(b)
135072304
```

5 내장함수

input([prompt]) (사용자 입력을 받는 함수)

```
>>> a = input()
hi
>>> a
'hi'
>>> b = input("Enter: ")
Enter: hi
```

```
>>> b
'hi'
```

5 내장함수

int 문자열->숫자(정수), 실수->정수

```
>>> int('3')
3
>>> int(3.4)
3
```

```
>>> int('11', 2)  # type(2진수)
3
>>> int('1A', 16) # type(16진수)
26
```

5 내장함수

isinstance(object, class)

```
>>> class Person: pass
...
>>> a = Person()
>>> isinstance(a, Person)  # 인스턴스(object)가 해당 class인지 검사
True
```

```
>>> b = 3
>>> isinstance(b, Person)
False
```

5 내장함수

lambda

```
>>> sum = lambda a, b: a+b
>>> sum(3,4)
7
```

```
>>> myList = [lambda a,b:a+b, lambda a,b:a*b]
>>> myList
[at 0x811eb2c>, at 0x811eb64>]
```

```
myList[0](1, 2) -> 3 (1+2)
myList[1](1, 2) -> 2 (1*2)
```

5 내장함수

len(s) (s의 길이)

```
>>> len("python")
6
>>> len([1,2,3])
3
>>> len((1, 'a'))
2
```

5 내장함수

list(s) (반복 가능한 자료-> list로 변환)

```
>>> list("python")
['p', 'y', 't', 'h', 'o', 'n']
>>> list((1,2,3))
[1, 2, 3]
```

```
>>> a = [1, 2, 3]
>>> b = list(a)
>>> b
[1, 2, 3]
```

5 내장함수

map(f, iterable)

```
>>> def two_times(x): return x*2
```

```
>>> list(map(two_times, [1, 2, 3, 4]))
[2, 4, 6, 8]
```

```
>>> list(map(lambda a: a*2, [1, 2, 3, 4]))
[2, 4, 6, 8]
```

입력받은 자료형의 각 요소가
함수 f에 대해 수행된 결과를 묶어서 리턴하는 함수

5 내장함수

max(iterable) (요소 중 최대값 출력)

```
>>> max([1, 2, 3])
3
>>> max("python")
'y'
```

min(iterable) (요소 중 최소값 출력)

```
>>> min([1, 2, 3])
1
>>> min("python")
'h'
```

5 내장함수

oct(x) (정수x -> 8진수)

```
>>> oct(34)
'0o42'
>>> oct(12345)
'0o30071'
```

5 내장함수

open

```
>>> f = open("binary_file", "rb")
```

mode	설명
w	쓰기 모드로 파일 열기
r	읽기 모드로 파일 열기
a	추가 모드로 파일 열기
b	바이너리 모드로 파일 열기

5 내장함수

ord(c) (문자->코드값(정수))

```
>>> ord('a')
97
>>> ord('0')
48
```

5 내장함수

pow(x, y)

```
>>> pow(2, 4)
16
>>> pow(3, 3)
27
```

05-5 내장함수

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

```
>>> list(range(5))
[0, 1, 2, 3, 4]
```

stop (step = 1), (start = 0)

```
>>> list(range(5, 10))
[5, 6, 7, 8, 9]
```

start stop (step = 1)

```
>>> list(range(1, 10, 2))
[1, 3, 5, 7, 9]
>>> list(range(0, -10, -1))
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
```

5 내장함수

sorted(iterable) (정렬 후 list로 변환)

```
>>> sorted([3, 1, 2])
[1, 2, 3]
>>> sorted(['a', 'c', 'b'])
['a', 'b', 'c']
>>> sorted("zero")
['e', 'o', 'r', 'z']
>>> sorted((3, 2, 1))
[1, 2, 3]
```

5 내장함수

str(object)

```
>>> str(3)
'3'
>>> str('hi')
'hi'
>>> str('hi'.upper())
'HI'
```

5 내장함수

tuple(iterable)

```
>>> tuple("abc")
('a', 'b', 'c')
>>> tuple([1, 2, 3])
(1, 2, 3)
>>> tuple((1, 2, 3))
(1, 2, 3)
```

5 내장함수

type(object)

```
>>> type("abc")
<class 'str'>
>>> type([ ])
<class 'list'>
>>> type(open("test", 'w'))
<class '_io.TextIOWrapper'>
```

5 내장함수

zip

```
>>> list(zip([1, 2, 3], [4, 5, 6]))
[(1, 4), (2, 5), (3, 6)]
>>> list(zip([1, 2, 3], [4, 5, 6], [7, 8, 9]))
[(1, 4, 7), (2, 5, 8), (3, 6, 9)]
>>> list(zip("abc", "def"))
[('a', 'd'), ('b', 'e'), ('c', 'f')]
```

동일한 개수로 이루어진 자료형을 묶어주는 역할을 하는 함수

외장함수 (external function)

6 외장함수

sys.argv (명령행에서 인수 전달하기)

```
# argv_test.py
import sys
print(sys.argv)
```

```
C:/Python/Mymodules>python argv_test.py you need python
['argv_test.py', 'you', 'need', 'python']
```

sys.exit() -> 강제로 script 종료하기
sys.path -> 자신이 만든 모듈 불러와 사용하기

6 외장함수

pickle 객체형태 그대로 유지하면서 파일에서 불러옴

```
>>> import pickle
>>> f = open("test.txt", 'wb')
>>> data = {1: 'python', 2: 'you need'}
>>> pickle.dump(data, f)
>>> f.close()
```

```
>>> import pickle
>>> f = open("test.txt", 'rb')
>>> data = pickle.load(f)
>>> print(data)
{2:'you need', 1:'python'}
```

6 외장함수

os (환경변수나 디렉토리, 파일 등의 os 자원 제어)

```
>>> import os
>>> os.environ          # 내 시스템 환경 변수
environ({'PROGRAMFILES': 'C:\\Program Files', 'APPDATA': … 생략 …})
>>> os.chdir("C:\WINDOWS") # 디렉토리 위치 변경
>>> os.getcwd()         # 디렉토리 위치 알기
'C:\WINDOWS'
>>> os.system("dir")    # 시스템 명령어 호출하기
>>> f = os.popen("dir") # 실행한 시스템 명령어 결과값 리턴받기
>>> print(f.read())
```

6 외장함수

os (환경변수나 디렉토리, 파일 등의 os 자원 제어)

```
os.mkdir(디렉토리)   : 디렉토리 생성
os.rmdir(디렉토리)   : 디렉토리 삭제, 비어있어야 가능
os.unlink(파일)      : 파일을 지운다
os.rename(src, dst) : 이름을 바꿈
```

6 외장함수

shutil

```
>>> import shutil
>>> shutil.copy("src.txt", "dst.txt")
```

6 외장함수

glob (특정 디렉토리에 있는 파일 모두 보여줌-> list로 출력)

```
>>> import glob
>>> glob.glob("C:/Python/q*")
['C:\Python\quiz.py', 'C:\Python\quiz.py.bak']
>>>
```

6 외장함수

tempfile (파일을 임시로 만들어 사용)

```
>>> import tempfile
>>> filename = tempfile.mktemp() # 파일 생성 후 이름 무작위로 생성
>>> filename
'C:\WINDOWS\TEMP\~-275151-0'
```

```
>>> import tempfile
>>> f = tempfile.TemporaryFile() # 파일 생성 후 파일 객체 리턴
>>> f.close()
```

6 외장함수

time 1

```
>>> import time
>>> time.time()       # utc를 이용하여 현재 시간을 실수로 반환
988458015.73417199
```

```
>>> time.localtime(time.time())   # 실수값 -> 년도, 월, 일, 시, 분 초로 표시
time.struct_time(tm_year=2013, tm_mon=5, tm_mday=21,
    tm_hour=16, tm_min=48, tm_sec=42, tm_wday=1,
    tm_yday=141, tm_isdst=0)
```

6 외장함수

time 2

```
>>> time.asctime(time.localtime(time.time()))
'Sat Apr 28 20:50:20 2001'
```

```
>>> time.ctime()
'Sat Apr 28 20:56:31 2001'
```

```
>>> import time
>>> time.strftime('%x', time.localtime(time.time()))
'05/01/01'
>>> time.strftime('%c', time.localtime(time.time()))
'05/01/01 17:22:21'
```

6 외장함수

calendar

```
>>> calendar.weekday(2015, 12, 31) # 요일 표시
3
```

```
>>> calendar.monthrange(2015,12) # 달의 일수 표시
(1, 31)
```

6 외장함수

random

```
>>> import random
>>> random.random()          # 0.0~1.0 사이 난수 표시
0.53840103305098674
```

```
>>> random.randint(1, 10)    # 1~10 사이 정수 난수로 표시
6
```

```
>>> data = [1, 2, 3, 4, 5]
>>> random.shuffle(data)     # list에 있는 값을 섞음
>>> data
[5, 1, 3, 4, 2]
```

6 외장함수

webbrowser

```
>>> import webbrowser
>>> webbrowser.open("http://google.com")
```

```
>>> webbrowser.open_new("http://google.com")
```

6 외장함수

threading

```
import threading
import time
def say(msg):
    while True:
        time.sleep(1)
        print(msg)
for msg in ['you', 'need', 'python']:
    t = threading.Thread(target=say, args=(msg,))  # thread 생성
    t.daemon = True
    t.start()
for i in range(100):
    time.sleep(0.1)
    print(i)
```