#decorator

# runtime add the behavious in the base class

from functools import wraps

def timmer\_def(func):

@wraps(func)

def wrapper(\*args):

print('Timmer calling from {}'.format(func.\_\_name\_\_))

return func(\*args)

return wrapper

def LogInfo(func):

@wraps(func)

def wrapper(\*args):

print('Log func calling from {}'.format(func.\_\_name\_\_))

return func(\*args)

return wrapper

@timmer\_def #callFunc = decoratorfun(baseMethod)

@LogInfo

def baseMethod():

print('base func')

@timmer\_def

def baseAnotherMethod(x, y):

print('base func {0}, {1}'.format(x, y))

baseMethod()

baseAnotherMethod(3, 4)