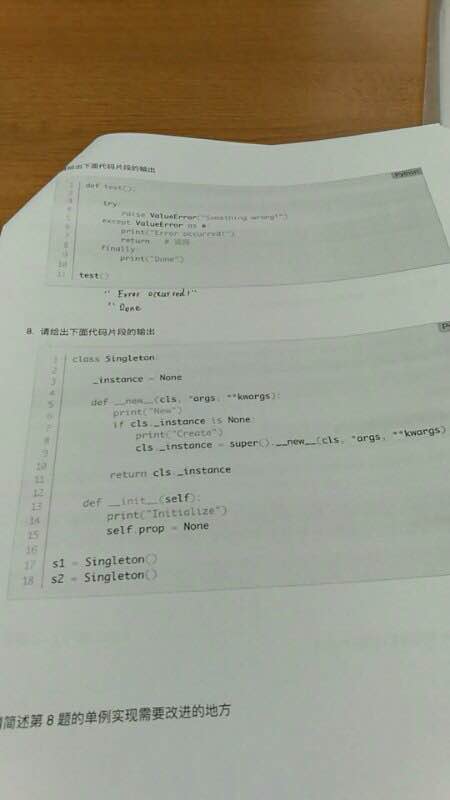


1:B 2:D 3:B 4:D 5:A 6:? 7: A 8: B 9:ABC 10: C



第八题：

输出结果：

None

Create

Initialize

None

Initialize

需要改进的地方：

class Singleton (object):

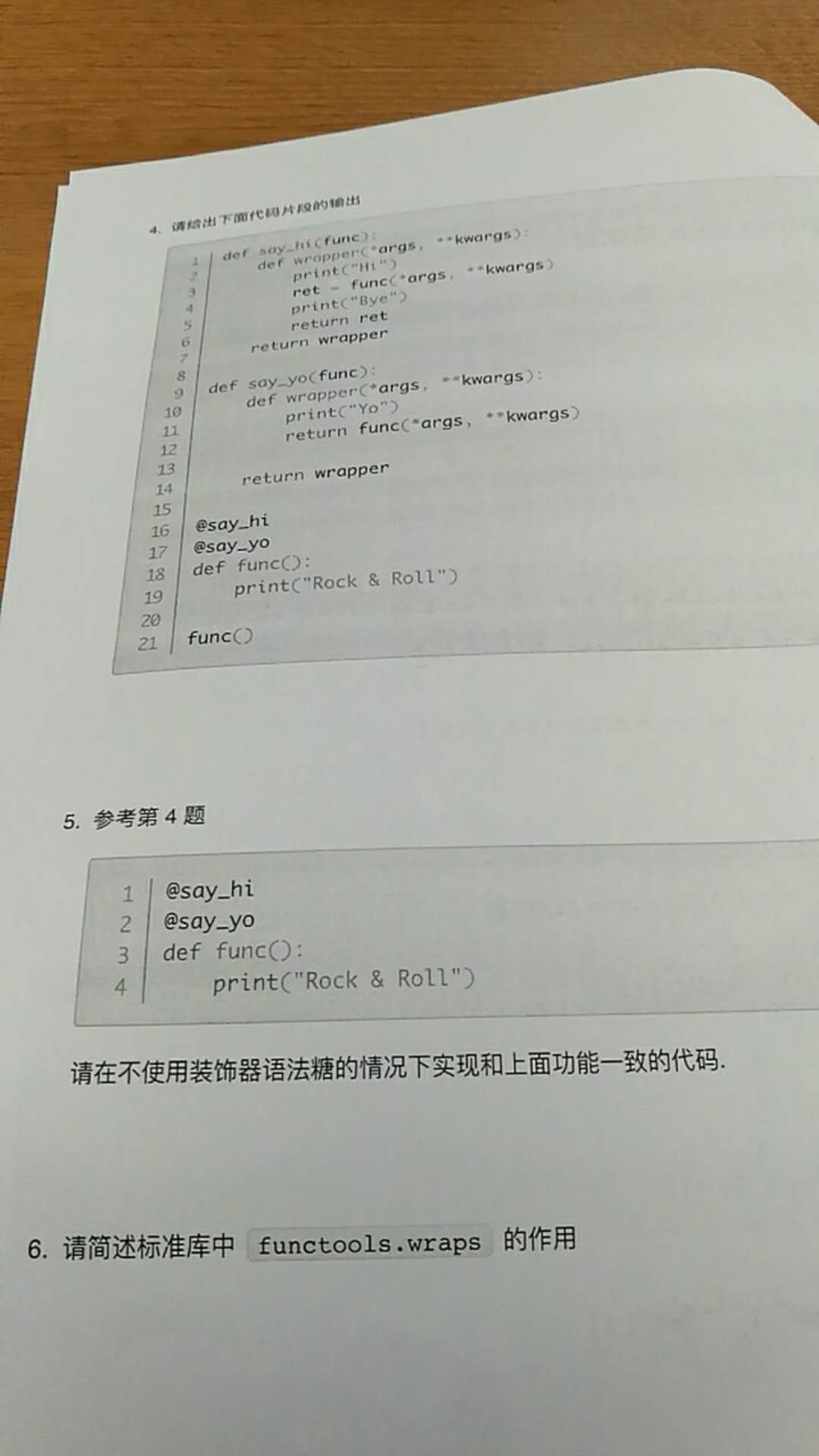
def \_\_new\_\_(cls, \*args, \*\*kwargs):

if not hasattr(cls, ‘\_instance’):

orig = super(Singleton, cls)

cls.\_instance = orig.\_\_new\_\_(cls, \*args, \*\*kwargs)

return cls.\_instance



第四题：

输出结果；

Hi

yo

Rock & Roll

Bye

第五题：

def func\_dec(func):

def wrapper():

print('hi')

print('yo')

func()

print('bye')

return wrapper

def func():

print('Rock & Roll')

func\_super = func\_dec(func)

func\_super()

第六题：

Python 中使用装饰器对在运行期对函数进行一些外部功能的扩展。但是在使用过程中,由于装饰器的加入导致解释器认为函数本身发生了改变,在某些情况下——比如测试时——会导致一些问题。Python 通过 functool.wraps 为我们解决了这个问题:在编写装饰器时,在实现前加入 @functools.wraps(func) 可以保证装饰器不会对被装饰函数造成影响。