2) 1에서 저장한 파일을 읽어와 점수, 등수를 파일로 출력하는 프로그램을 작성하시오. 단, Score는 5칸(%5d), Rank는 각 4칸(%4d)으로 파일(test2.txt)로 출력한다.

```
f = open(r'c:\test1.txt', "r")
score = f.readlines()
f.close()

score_int = [ int(n) for n in score ]
print(score_int)

rank = []
max_number = 0
for i in range(5):
    max_number = -1
    for i in score_int:
        if max_number == 0:
            max_number = i

        if max_number < i:
            max_number = i

    rank.append(max_number)
    score_int.remove(max_number)

print(rank)

f = open(r'c:\test2.txt', 'w')
pos = 1
for line in rank:
    f.write("%5d %4d\n" % (line, pos))
    pos += 1
f.close()
```

파일에 새로운 내용 이어쓰기

파일의 내용을 이어서 쓰려면 이어쓰기(a)모드로 파일을 설정한다. 같은 이름의 파일이 존재하면 그 파일의 내용을 이어 쓴다. 같은 파일의 이름이 없으면 새로 파일을 만든다.

```
f = open("c:\song.txt", "a")
f.write("뒤에 이어서 쓴 애국가 입니다.")
f.close()
```

with 와 open()

with 키워드와 Open()를 같이 쓰면 with 키워드의 실행 블록(들여쓰는 부분)이 다 실행 되면 자동으로 close() 함수를 호출하여 편리하다.

```
with open("c:\song.txt", "r") as f:
    read_file = f.read()
```

객체 지향과 절차 지향 프로그래밍

절차지향 : 코드를 순서대로(위 아래) 작성하여 로직을 구성하는 프로그래밍 방법, 프로젝트가 커질수록 참여 인원이 많아질 수록 관리가 어렵다.

객체지향 : 사물, 사람 등 대상을 프로그래밍으로 모델링하는 프로그래밍 방법 대상에 대해 데이터(키, 무게 등)와 기능 혹은 행동(노래를 부른다, 걸어 다닌다) 으로 나누어 사물을 모델링

객체 : 사물, 사람, 동물 등 세상에 있는 대상 강아지

데이터 : 강아지의 나이, 색상, 견종 행동(기능) : 사료를 먹는다, 짖는다.

데이터 >> 변수 행동(기능) >> 함수

클래스 : 사물의 데이터와 행동을 변수와 함수로 지정하여 만들어 구성한 설계도

Quiz) 보조베터리, 에어콘, 택배보관함

를 대상으로 데이터와 행동 5개씩 선정하기

보조베터리

데이터 : 충전량, 단자수, 모델명 행동(기능): 충전하기, 충전 잔량확인하기, 과충전방지하기

에어콘

데이터 : 색상, 설치 타입, 냉방 평수 행동(기능) : 냉방하기, 먼지거르기, 온도출력하기

택배보관함

데이터 : 비밀번호, 보관 가격, 보관 위치 행동(기능) : 물건찾기, 문자알림, 도난알림

클래스 만드는 문법

class 클래스이름: 변수1 변수2

```
def 함수이름(self,매개변수):
    함수 실행 부분

def 함수이름2(self,매개변수):
    함수 실행 부분
```

```
class Dog:
    age = 1
    color = "gold"
    dog_type = "골든 리트리버"

    def eat(self, food):
      print("강아지가 %s를 먹고 있습니다." % food)

    def bark(self):
      print(self.dog_type + "강아지가 멍멍 합니다")
```

클래스로 부터 객체 만드는 명령

클래스는 대상을 표현한 설계도 이므로 프로그래밍에서 실체로 활용할 수 없고, 클래스로 객체를 만들어야 프로그래밍에 활용할 수 있다.

객체를 저장할 변수 = 클래스이름([인수,])

```
dog = Dog()
```

클래스의 변수 읽어오기, 함수 실행하기

객체 변수.(클래스)변수 객체 변수.함수()

```
print(dog.age)
dog.eat("껌")
```

다음과 같은 클래스를 만드세요.

대상 : 사과장수 데이터 : 돈(money), 사과 수(apple), 사과 가격(price) 행동 : 가격 출력하기(알려주기), 사과 팔기(갯수)

대상 : 사과구매자 데이터 : 돈(money), 사과 수(apple), 사과 가격(price) 행동 : 가격 물어보기, 사과 구입하기(갯수)

```
class AppleSeller:
    money = 10000
    apple = 100
    price = 500

    def printPrice(self):
        print("%d 원입니다." % self.price)

    def sellApple(self, ea):
        if self.apple >= ea:
            self.apple = self.apple - ea
            self.money = self.money + (ea * self.price)
        else:
            print("물량이 부족하여 판매할 수 없습니다.")

seller = AppleSeller()
seller.printPrice()
seller.sellApple(50)
print(seller.money)
print(seller.apple)

class AppleBuyer:
    money = 10000
    apple = 100
    price = 500

    def askPrice(self):
        print("가격이 얼마인가요?")

    def buyApple(self, ea, price):
        if self.money >= (ea * price):
            self.apple = self.apple + ea
            self.money = self.money - (ea * price)
        else:
            print("돈이 부족하여 판매할 수 없습니다.")
```

객체지향 프로그래밍은?

객체간에 데이터를 주고 받는 형태로 전체적인 처리 로직을 프로그래밍하는 방법

```
buyer = AppleBuyer()
seller = AppleSeller()

buyer.askPrice()
```

```
seller.printPrice()
buyer.buyApple(10, buyer.price)
seller.sellApple(10)
print(buyer.apple)
print(seller.apple)

seller2 = AppleSeller()
seller2.apple = 3000

print(seller.apple)
print(seller2.apple)
```

init 함수

객체 생성과 동시에 클래스의 변수를 초기화 하고 싶은 경우에 활용한다. def **init**(self, 초기값1, 초기값2 ...): 클래스의변수1 = 초기값1 클래스의변수2 = 초기값2

```
class AppleSeller2:
    money = 10000
    apple = 100
    price = 500

    def __init__(self, a_money, a_apple, a_price):
        self.money = a_money
        self.apple = a_apple
        self.price = a_price

    def printPrice(self):
        print("%d 원입니다." % self.price)

    def sellApple(self, ea):
        if self.apple >= ea:
            self.apple = self.apple - ea
            self.money = self.money + (ea * self.price)
        else:
            print("물량이 부족하여 판매할 수 없습니다.")

seller = AppleSeller()
seller2 = AppleSeller2(30000, 200, 100)
print(seller.price)
print(seller2.price)
seller2.banana = 10
print(seller2.banana)
```

클래스 상속

클래스들의 공통적인 함수, 변수 등을 그대로 가져와서 새로운 클래스를 작성할 수 있는 방법

부모 클래스 : 공통적인 함수&변수를 가지고 있는 클래스 자식 클래스 : 부모 클래스로 부터 만들어진 새로운 클래스

자식 클래스를 만들려면

class 클래스이름(부모클래스의 이름): 클래스의 내용

```
class Car:
    color = "sliver"

    def run(self):
        print("움직입니다.")


class Truck(Car):
    def carry(self, pack):
        print("%s 을(를) 적재하였습니다" % pack)


t = Truck()
t.carry("과일")
print(t.color)
t.run()
```

Car 클래스를 상속하여 오토바이 클래스를 만들어주세요. 오토바이 클래스는 tire 변수, 경적소리를 내는 sound() 함수를 만드세요.

```
class Motorcycle(Car):
    tire = 2

    def sound(self):
        print("부르르르르릉")

m = Motorcycle()
m.sound()
```

재정의(오버라이딩, overiding)

자식 클래스는 부모 클래스의 변수 함수와 같은 이름의 변수 함수를 새로 만들수 있다

```
class Truck2(Car):
    color = "blue"

    def carry(self, pack):
        print("%s 을(를) 적재하였습니다" % pack)

    def run(self):
        print("우다다타탕")

class Motorcycle2(Car):
    tire = 2
    color = "red"

    def run(self):
        print("위이이이잉")

    def sound(self):
        print("부르르르르릉")


t2 = Truck2()
m2 = Motorcycle2()

t.run()
t2.run()
m2.run()
```

Person 클래스를 만들어서 AppleSeller3, AppleBuyer3 클래스를 만드세요.

상속할 내용

money = 10000 apple = 100 price = 500

printStat() >> 객체가 가진 돈, 사과, 가격을 출력하는 함수

```
class Person:
    money = 0
    apple = 0
    price = 0
```

```
    def __init__(self, aMoney, aApple, aPrice):
        self.money = aMoney
        self.apple = aApple
        self.price = aPrice

    def printStat(self):
        print("%s %s %s\n" % ("돈", "사과", "가격"))
        print("%d %d %d\n" % (self.money, self.apple, self.price))

class AppleBuyer3(Person):
    place = ""

    def __init__(self, aMoney, aApple, aPrice, aPlace):
        self.money = aMoney
        self.apple = aApple
        self.price = aPrice
        self.place = aPlace

    def askPrice(self):
      print("가격이 얼마인가요?")

    def buyApple(self, ea, price):
      if self.money >= (ea * price):
        self.apple = self.apple + ea
        self.money = self.money - (ea * price)
      else:
        print("돈이 부족하여 판매할 수 없습니다.")


buyer3 = AppleBuyer3(10000, 200, 100, "서울")
buyer3.printStat()

class AppleSeller3(Person):
    place = ""


    def __init__(self, aMoney, aApple, aPrice, aPlace):
        self.money = aMoney
        self.apple = aApple
        self.price = aPrice
        self.place = aPlace

    def printPrice(self):
        print("%d 원입니다." % self.price)

    def sellApple(self, ea):
        if self.apple >= ea:
            self.apple = self.apple - ea
            self.money = self.money + (ea * self.price)
```

```
        else:
            print("물량이 부족하여 판매할 수 없습니다.")

seller3 = AppleSeller3(2000, 30, 100,"부산")
seller3.printStat()
```

연산자 오버로딩(overloading)

클래스의 객체간에 연산자의 실행 방식을 정의할 수 있다. 각 연산자 마다 실행 방식을 정의할 수 있는 함수가 정해져 있다.

함수 이름 : 연산자 **add** : + **sub** : - **mul** : * **truediv** : / **mod**: %

```
class AppleSeller4(Person):
    place = ""

    def __add__(self, other):
        print("두 사과장수의 사과 갯수는 %d 개입니다." % (self.apple + other.apple))

    def __init__(self, aMoney, aApple, aPrice, aPlace):
        self.money = aMoney
        self.apple = aApple
        self.price = aPrice
        self.place = aPlace

    def printPrice(self):
        print("%d 원입니다." % self.price)

    def sellApple(self, ea):
        if self.apple >= ea:
            self.apple = self.apple - ea
            self.money = self.money + (ea * self.price)
        else:
            print("물량이 부족하여 판매할 수 없습니다.")

seller4 = AppleSeller4(10000, 100, 100, "서울")
seller5 = AppleSeller4(20000, 200, 500, "서울")

seller4 + seller5
```

quiz)

곱하기연산자(*)를 활용하여 두 사과장수가 합병했을 때 돈, 사과, 가격을 합쳐서 출력해주세요.

참고) 함수 이름 : 연산자 **add** : + **sub** : - **mul** : * **truediv** : / **mod**: %

```
class AppleSeller5(Person):
    place = ""
    def __mul__(self, other):
        print("두 사과장수가 합병하면 사과 : %d,  돈 : %d, 가격 : %d" %
              ((self.apple+other.apple),(self.money+other.money),
(self.price+other.price)))


    def __add__(self, other):
        print("두 사과장수의 사과 갯수는 %d 개입니다." % (self.apple + other.apple))

    def __init__(self, aMoney, aApple, aPrice, aPlace):
        self.money = aMoney
        self.apple = aApple
        self.price = aPrice
        self.place = aPlace

    def printPrice(self):
        print("%d 원입니다." % self.price)

    def sellApple(self, ea):
        if self.apple >= ea:
            self.apple = self.apple - ea
            self.money = self.money + (ea * self.price)
        else:
            print("물량이 부족하여 판매할 수 없습니다.")

seller4 = AppleSeller5(10000, 100, 100, "서울")
seller5 = AppleSeller5(20000, 200, 500, "서울")

seller4 * seller5
```