#Review Chapter 1

#变量 - 变量无需声明，可直接使用

print('-' \* 80)

a = 100

b = 'Python'

c = 3.14

print('a is : ', a, ', b is : ' , b, ', c is : ',c)

print('-' \* 80)

#字符串

#字符串可以包含任意字符，可用单引号也可用双引号

v\_str1 = "Harry is studying Python, keep going."

print(v\_str1)

v\_str2 = 'If you want to change , just do it. And you should never stop.'

print(v\_str2)

print('-' \* 80)

#字符串使用“+”拼接，拼接非字符串需要使用str()或者repr()函数转换后再拼接

v\_str3 = "Hello , who are you ?\t"

v\_str4 = "Hello, I am Harry.\t"

v\_str5 = "How old are you ? \t"

v\_age = 36

v\_str6 = "I am " + str(v\_age) + ' years old.'

print(v\_str3 + v\_str4)

print(v\_str5 + v\_str6)

print('-' \* 80)

#长字符串

v\_str7 = '''

Long long ago , there was a boy,

His name is Jack.

He liked to study.

...

'''

print(v\_str7)

print('-' \* 80)

v\_rstr = r'This is a resource string, \t will not be a table'

print(v\_rstr)

print('-' \* 80)

#转义字符

v\_str = "abc\bdef\nghi\rjkl\tmno\"opqr\'stu\\vwxys"

print(v\_str)

print('-' \* 80)

#字符串转字节串

v\_bytes = b'IamHarry'

print(type(v\_bytes))

print('bytes array\'s length is : ', len(v\_bytes))

for b in v\_bytes:

print(b)

print('Character : ' + str(b) + '\t')

print('Character is : %r' %b)

print('-' \* 80)

#字符串常用函数

#索引计算字符串

v\_str = "abcdefghijklmnopqistuvwxyz"

print(v\_str[3:6])

print(v\_str[8])

print(v\_str[-8:-1])

print(v\_str[1:10:3])

print('-' \* 80)

#in判断是否包含某个字符

print('a is in the string : ', 'a' in v\_str)

print('2 is in the string : ', '2' in v\_str)

print('-' \* 80)

#len()字符串长度

print('v\_str length is : ', len(v\_str))

print('-' \* 80)

#max/min函数

print("Max character : ", max(v\_str))

print("Min character : ", min(v\_str))

print('-' \* 80)

#title()首字母大写

print(v\_str.title())

print('-' \* 80)

#upper()全部大写

print(v\_str.upper())

print('-' \* 80)

#lower()全部小写

print(v\_str.lower())

print('-' \* 80)

#strip()去除空格

print(v\_str.strip())

print('-' \* 80)

#startswith()是否以某个字符开头

print(v\_str.startswith('b'))

print('-' \* 80)

#endswith()是否以某个字符结尾

print(v\_str.endswith('z'))

print('-' \* 80)

#find()查找字符，存在返回index，不存在返回-1

v\_index = v\_str.find('g')

print('Index is : ', v\_index)

v\_index = v\_str.find('8')

print('Index is : ', v\_index)

print('-' \* 80)

#replace()替换字符

v\_new\_str = v\_str.replace('abc', '123')

print('New string is : ', v\_new\_str)

print('-' \* 80)

#split()分割字符串

v\_str = 'a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q'

v\_array = v\_str.split(',')

print(v\_array)

print('-' \* 80)

#join()连接字符串

v\_str = '-'.join(v\_array)

print('Now the string is : ', v\_str)

print('-' \* 80)

#运算符

x = 20

y = 7

print(x + y)

print(x - y)

print(x \* y)

print(x / y)

print(x % y)

print(x // y)

print(x \*\* 2)

print('-' \* 80)

x += 100

print(r'x+=100 :', x)

x -= 10

print(r'x-=10 :', x)

x \*= 2

print(r'x\*=2 :', x)

x /= 4

print(r'x/=4 :', x)

x //= 5

print(r'x//=5 :', x)

x %= 4

print(r'x%=4 :', x)

x \*\*= 3

print(r'x\*\*=3 :', x)

print('-' \* 80)

x , y = 20, 30

print('x : ', x, ' y : ', y)

print(r'x > y : ', x > y)

print(r'x >= y : ', x >= y)

print(r'x < y : ', x < y)

print(r'x <= y : ', x > y)

print(r'x == y : ', x == y)

print(r'x != y : ', x != y)

z = x

print('z : ', z)

print(r'z is x : ', z is x)

print(r'z is not x : ', z is not x)

b1,b2 = True, False

print('b1 : ', b1, ' b2 : ', b2)

print('b1 and b2 : ', b1 and b2)

print('b1 or b2 : ', b1 or b2)

print('not b1 : ', not b1)

print('not b2 : ', not b2)

print('-' \* 80)

#三目运算符

#表达式：True\_Statements if expression else Else\_Statements，支持嵌套

print('I am OK!' if b2 else print('I am OK indeed!') if b1 else print('I am not OK!'))

print('-' \* 80)

#Review Chapter 2

#列表、元组(两者的区别就是列表元素可以修改，元组元素不可修改)

print('' \* 100)

v\_list = [1,2,3,'Java',3.14,"Python"]

v\_tuple = (10,20,30,'Harry',40,50)

print('List : ', v\_list)

print('Tuple : ', v\_tuple)

v\_list = list(range(1,10))

v\_tuple = tuple(range(5,15))

print('' \* 100)

print('Now the list is : ', v\_list)

print('Now the tuple is : ', v\_tuple)

print('' \* 100)

#通过索引访问列表及元组

for i in range(len(v\_list)):

print('Index is : ', i, ' element is : ', v\_list[i])

print('-' \* 100)

#某个下标对应的值

print('List index 4 , element is : ', v\_list[4])

print('Tuple index 4 , element is : ', v\_tuple[4])

print('-' \* 100)

#包含前面的下标值，不包含后面的下标值

print('List elements 3 - 5 : ', v\_list[3:5])

print('Tuple Elements 3 - 5 : ', v\_tuple[3:5])

print('-' \* 100)

#带有步长的列表取值范围

print('List elements 1 - 10 , every two : ', v\_list[1:10:2])

print('Tuple elements 1 - 10 , every two : ', v\_tuple[1:10:2])

print('-' \* 100)

#列表、元组相加 - 只是元素的总和

#列表只能和列表相加、元组只能和元组相加。如果列表加元组，需要使用list函数将元组转换为列表后再相加

v\_list1 = [1,2,3]

v\_list2 = ['a',b'','c']

v\_list\_sum = v\_list1 + v\_list2

print('List 1 : ', v\_list1)

print('List 2 : ', v\_list2)

print('List sum : ', v\_list\_sum)

print('-' \* 100)

#列表相乘，只是元素重复对应的倍数，字符串也支持翻倍

print('List1 : ', v\_list1)

print('List1 \* 3 : ', v\_list1 \* 3)

v\_str = 'abc'

print('String : ', v\_str)

print('String \* 3 : ', v\_str \* 3)

#序列相关函数

print('-' \* 100)

#len()序列长度

print('List 1 : ', v\_list1)

print('List1 length : ', len(v\_list1))

#max/min函数

print('Max element of list1', max(v\_list1))

print('Min element of list1', min(v\_list1))

print('-' \* 100)

#封包，将多个值付给一个变量，Python会自动将多个值封装成元组，这种功能称之为封包

v\_params = 1,2,3,'Python',100

print('Param type is : ', type(v\_params))

#解包，把一个元组赋给多个变量时，Python会自动将元组元素自动赋值给各个变量，这种功能称之为解包

print('-' \* 100)

v\_params = (1,2,3,'C#')

a,b,c,d = v\_params

print('a is : ',a,'\t b is : ',b,'\t c is : ',c, '\t d is : ', d)

print('-' \* 100)

#多个值赋给多个变量:本质就是先封包后解包

a,b,c,d = 1,2,3,'Java'

print('a is : ',a,'\t b is : ',b,'\t c is : ',c, '\t d is : ', d)

print('-' \* 100)

#列表操作方法

#追加元素 - append方法

v\_list = [1,2,3]

print('List is : ', v\_list)

v\_list.append('.NET')

print('After append one element : ', v\_list)

print('-' \* 100)

#extend()追加另一列表/元组所以的元素到当前列表

v\_list.extend(list(range(4,8)))

print('After extend a list : ', v\_list)

v\_list.extend(tuple(range(8,10)))

print('After extend a tuple : ', v\_list)

print('-' \* 100)

#insert()将元素插入到指定的位置

v\_list.insert(5, 'Python')

print('Insert one element of index 5 : ', v\_list)

print('-' \* 100)

#删除元素- 方式一：del Python专门执行删除的语句，可以删除列表元素、字典元素以及变量

print('Before delete the list is :', v\_list)

del v\_list[3]

print('First delete (delete index 3 element) : ', v\_list)

del v\_list[1:3]

print('Second delete (delete [1 - 3] index elements) : ', v\_list)

del v\_list[2:6:2]

print('Second delete (delete [2 - 6] step 2 elements) : ', v\_list)

print('-' \* 100)

#remove()方法，不是根据下标删除元素，而是删除第一个匹配的元素

print('Before remove the list is :', v\_list)

v\_list.remove(7)

print('Remove element 7 : ', v\_list)

print('-' \* 100)

#元素赋值也可实现列表的增加和删除

v\_list = list(range(1,10))

print('The origin list is : ', v\_list)

#指定索引插入一个值，列表新增一个元素

v\_list[3] = 'Java'

print(v\_list)

#指定索引索引域插入N个值，列表新增N个元素

v\_list[4:6] = ['Python','C#','JavaScript']

print(v\_list)

#指定索引索引域插入1个值，列表较少N个元素

v\_list[0:3] = [100]

print(v\_list)

#字符串会被当成数组处理

v\_list[5:7] = 'Harry'

print(v\_list)

print('-' \* 100)

#列表方法

#统计元素个数count()

print('element r count : ', v\_list.count('r'))

print('element 200 count : ', v\_list.count(200))

print('-' \* 100)

#定位元素坐标, 返回第一个匹配元素的坐标

print('element r index : ', v\_list.index('r'))

print('-' \* 100)

#弹出元素pop()，弹出最后一个元素

v\_list.pop()

print('After pop the list is : ', v\_list)

print('-' \* 100)

#翻转列表

v\_list.reverse()

print('Now the list is : ', v\_list)

print('-' \* 100)

#排序 - 此方法仅限同种类型的元素

v\_list = [1,100,20,15,200,43,45,190]

print('Before sort : ', v\_list)

v\_list.sort()

print('After sort asc: ', v\_list)

v\_list.reverse()

print('desc order : ', v\_list)

print('-' \* 100)

#字典 - 字典创建方式

#方式一：{}直接创建

v\_dict1 = {'Python':100,'Java':200,'Groovy':300}

print('Use {} to create a dictionary : ', v\_dict1)

print('-' \* 100)

#方式二：使用dict构造器构筑，传值方式有二：1：传入多个列表或者元组（列表、元组的元素个数为2）2：传入关键字，关键字不允许使用表达式

v\_dict2 = dict([['A',1000],['B',2000],['C',3000],['D',4000]])

print('Use dict to create a dictionary 1 : ', v\_dict2)

v\_dict3 = dict([('OK',200),('ERROR',400),('NOTFOUND',404)])

print('Use dict to create a dictionary 2 : ', v\_dict3)

v\_dict4 = dict((('A',200),('B',400),('C',404)))

print('Use dict to create a dictionary 3 : ', v\_dict4)

v\_dict5 = dict((['AA',200],['BB',400],['CC',404]))

print('Use dict to create a dictionary 4 : ', v\_dict5)

v\_dict6 = dict(AAA=100,BBB='JACK',CCC=3.14)

print('Use dict to create a dictionary 5: ', v\_dict6)

print('-' \* 100)

#字典访问 方式一：类型列表下标访问，方式二：使用get方法

print('Use [] way (v\_dict1[\'Python\']): ', v\_dict1['Python'])

print('Use get() way ( v\_dict1.get(\'Python\')): ', v\_dict1.get('Python'))

print('-' \* 100)

#更改元素,存在key，则修改value值，不存在，则新增元素

print('Before update : ', v\_dict2)

v\_dict2['B'] = 'Change'

v\_dict2['E'] = 500

print('After update : ', v\_dict2)

print('-' \* 100)

#删除字典元素 del

print('Before delete : ', v\_dict2)

del v\_dict2['C']

print('After delete : ', v\_dict2)

print('-' \* 100)

#in/not in 判断是否存在某个元素 : 根据key进行判断

print('A' in v\_dict2)

print('C' in v\_dict2)

print('-' \* 100)

#清空字典

print('Before clear : ', v\_dict3)

v\_dict3.clear()

print('After clear : ', v\_dict3)

print('-' \* 100)

#update : 存在key，更新value值，否则新增key-value , 参数形式和dict构筑字典方式的参数一致

print('Before update : ', v\_dict4)

v\_dict4.update((('A',900),))

print('After update 1: ', v\_dict4)

v\_dict4.update([('D',100),('E',200),('F',300)])

print('After update 2: ', v\_dict4)

v\_dict4.update(E=800,H=600)

print('After update 3: ', v\_dict4)

print('-' \* 100)

#遍历

print('Dictiionary 5 : ', v\_dict5)

print('-' \* 100)

#遍历item

for k, v in v\_dict5.items():

print('Key is : \t', k, '\tvalue is :\t', v)

#遍历key

print('-' \* 100)

for k in v\_dict5.keys():

print('Key is : \t', k, '\tvalue is :\t', v\_dict5[k])

print('-' \* 100)

#遍历value

for v in v\_dict5.values():

print('Value : \t', v)

print('-' \* 100)

#setdefault方法，获取字段元素值，如果存在key，则返回对应的value，如果不存在，则返回默认值，同时字典新增一个key-value

print('Dictiionary 6 : ', v\_dict6)

v1 = v\_dict6.setdefault('NF', 404)

v2 = v\_dict6.setdefault('BBB', 500)

print('Value 1 is : \t', v1)

print('Value 2 is : \t', v2)

print('Dictiionary 6 : ', v\_dict6)

print('-' \* 100)

#fromkeys，使用序列构筑字典,默认值为None，也可指定默认值

v\_dict7 = dict.fromkeys(['K1','K2','K3','K4'])

v\_dict8 = dict.fromkeys(('KEY1','KEY2','KEY3'), 100)

print('Dictiionary create by list : ', v\_dict7)

print('Dictiionary create by tuple : ', v\_dict8)

print('-' \* 100)

#格式化字符串 ， 根据key匹配， 元组根据位置匹配

v\_str = "The book's name is : %s, and the price is : %10.2f"

print(v\_str %('Crazy Java', 134))

v\_str = "The book's name is : %(name)s, and the price is : %(price)10.2f"

print(v\_str %{'price':138, 'name':'Crazy Linux'})

print('-' \* 100)

#Review Chapter 3

#if

print('-' \*80)

age = 25

if age > 18:

print('成年人!')

else:

print('未成年!')

#if elif else

print('-' \*80)

name = 'Jack'

if name == 'Harry':

print('I am Harry.')

elif name == 'Jack':

print('I am Jack')

else:

print('I am someone else!')

print('-' \*80)

#三目运算符

grade = input('Input your grade (A,B,C) : ')

score = 100 if grade == 'A' else 90 if grade == 'B' else 80

print('Score : ', score)

print('-' \*80)

v\_list = []

if v\_list:

print('List is not empty!')

else:

print('List is empty!')

print('-' \* 80)

#pass : 占位，不做操作

if v\_list:

print('list is ok')

else:

pass

print('-' \* 80)

#while

v\_list = list(range(1,25))

i = 0

while i < len(v\_list):

print('Index is : ', i, '\tvalue is : ', v\_list[i])

i += 1

print('-' \* 80)

v\_dict = {'OK':200,'ERROR':400,'NOTFOUND':404,'SYSERROR':500}

print('Keys type : ', type(v\_dict.keys()))

keys = list(v\_dict.keys())

i = 0

while i < len(keys):

print('Key is : ', keys[i], '\tvalue is : ', v\_dict.get(keys[i]))

i += 1

print('-' \* 80)

#for in

for e in v\_list:

print('Element : ', e)

print('-' \* 80)

for key, value in v\_dict.items():

print('Key is : ', key, '\tvalue is : ', value)

print('-' \* 80)

#列表推导式

new\_list = [str(i) + '-' + str(i) for i in list(range(1,6))]

print('New list is : ', new\_list)

print('-' \* 80)

#break continue return

for i in list(range(1,20)):

if i == 15:

break

else:

print('i is : ', i)

print('-' \* 80)

for i in list(range(1,20)):

if i % 2 == 0:

continue

else:

print('i is : ', i)

print('-' \* 80)

#Review Chapter 4

#函数

print('-' \* 80)

def hi():

print('Hello , I am Harry')

hi()

print('-' \* 80)

#带有参数的函数

def paramFunction(name,age):

print('Name is : ', name, '\tage is : ', age)

#通过位置方式传入

paramFunction('Harry', 36)

print('-' \* 80)

#通过关键字传入

paramFunction(age=36,name='ChengGuoqian')

print('-' \* 80)

#带有默认值的函数，带有默认值的参数要放在不带默认值的参数后面

def defaultParamFun(age,name='Jack'):

print('Name is : ', name, '\tage is : ', age)

defaultParamFun(36)

print('-' \* 80)

#如果省略第一个参数，后面的参数需要使用关键字进行传入

def defaultParamFun2(name='ChengGuoqian',age=36):

print('Name is : ', name, '\tage is : ', age)

defaultParamFun2('Harry')

defaultParamFun2('Sam',24)

defaultParamFun2(age=25)

print('-' \* 80)

#参数收集

def multiParam1(amount, \*books):

print('Amount is : ', amount)

print('Books : ', books)

for book in books:

print('Book name : ', book)

multiParam1(100, 'Python','Java','Vue')

print('-' \* 80)

#如果普通参数收集放到了其他形参前面，形参只能通过关键字传参方式传入

def multiParam2(\*books, amount):

print('Amount is : ', amount)

print('Books : ', books)

for book in books:

print('Book name : ', book)

#以下调用会报错

#multiParam2('Python','Java','Vue'， 200)

#正确的调用方式如下

multiParam2('Python','Java','Vue',amount=200)

print('-' \* 80)

#前面加两个星号，表示通过关键字进行参数收集，参数会按照字典进行处理

def multiParam3(amount,\*books,\*\*scores):

print('Amount is : ', amount)

print('Books is : ', books)

print('Scores is : ', scores)

multiParam3(100,'Java','C#','.NET',english=120,math=120,chinese=100)

print('-' \* 80)

def multiParam4(\*names, message,\*\*scores):

print('Names is : ', names)

print('Message is : ', message)

print('Scores is : ', scores)

multiParam4('Harry','Jack','Tom',message='I am OK!',english=120,math=120,chinese=100)

print('-' \* 80)

#反向参数收集

params = (100, 200)

dictParam = {'a':300,'b':400}

def reParam(a,b):

print('a is : ',a, '\tb is : ', b)

reParam(\*params)

reParam(\*\*dictParam)

print('-' \* 80)

# 变量作用域

a = 1000

b = 2000

print(globals())

print('-' \* 80)

#变量覆盖

def info():

#使用globals函数获取全局参数字典，根据key获取值，此方式不会更改全局变量的值

print(globals()['a'])

a = 'local'

print('Local variable a is : ',a)

info()

print('Global variable a is : ',a)

def info2():

#使用global声明，此时使用的全局变量，此方式会更改全局变量a的值

global a

a = 'local'

print('Local variable a is : ',a)

print('-' \* 80)

info2()

print('Global variable a is : ',a)

info()

print('Global variable a is : ',a)

print('-' \* 80)

#内部函数

def outer(p):

print('outer function')

a = 'OK'

print('outer a is : ', a)

print('outer p is : ', p)

print('-' \* 50)

def inner(pi):

nonlocal a, p

for i in range(5):

print('inner for : ', i, '\ta is : ', a, '\tout p is : ', p, '\tinner p is : ', pi)

a += str(i)

p += str(i)

inner('inParam')

print('final a is : ', a)

print('final p is : ', p)

outer('outerParam')

print('-' \* 80)

#Review Chapter 5

#创建类

class User:

def \_\_init\_\_(self, name='Jack',pwd='123456'):

self.name = name

self.pwd = pwd

def info(self):

print('Name : %s , password : %s' %(self.name, self.pwd))

print('-' \* 80)

user = User()

user.info()

print('-' \* 80)

user = User('Sam')

user.info()

print('-' \* 80)

user = User('Harry', '12345678')

user.info()

print('-' \* 80)

user = User(pwd='4566789')

user.info()

print('-' \* 80)

user = User(pwd='526341',name='Tom')

user.info()

#修改实例变量

print('-' \* 80)

user = User(pwd='526341',name='User1')

user.info()

user.name = 'User2'

user.info()

#新增变量

user.address = 'SD YT'

print(user.address)

#删除变量

del user.address

#以下输出报错

#print(user.address)

print('-' \* 80)

#新增方法一：直接新增

def newFunc(self):

print('This is a new Method...')

user.hi = newFunc

#新增方法不会自动绑定self实例，需要手动绑定

user.hi(user)

print('-' \* 80)

#新增方式二：使用MethodType将方法包装成实例方法

def newFunc2(self):

print('This is another new method : ', self.name)

from types import MethodType

user.hello = MethodType(newFunc2, user)

user.hello()

print('-' \* 80)

#类中的方法调用其他方法时，self不可省略

class Dog:

def run(self):

self.jump()

print('Dog is running...')

def jump(self):

print('Dog is jumping...')

dog = Dog()

dog.run()

print('-' \* 80)

#self可以作为实例方法的返回值，把self参数作为实例方法的返回值，则可以多次连续调用该方法

class Plant:

def \_\_init\_\_(self, height=2):

self.height = height

def grow(self):

self.height += 2

return self

plant = Plant(1.5)

plant.grow().grow().grow()

print('Now the plant height is : ', plant.height,'(m)')

print('-' \* 80)

#类调用实例方法

class Role:

def info(self):

print('Role info ...')

r = Role()

r.info()

#类调用实例方法，此时就变成了‘未绑定方法’，self参数必须手动传入

Role.info(r)

print('-' \* 80)

#类方法 - 1. @classmethod， 2. 方法的第一个参数定义为cls

class Tiger:

@classmethod

def info(cls):

print('class method...')

print(cls)

Tiger.info()

#对象调用类方法等同于类调用，同样也会自动绑定

t = Tiger()

t.info()

print('-' \* 80)

#静态方法 - @staticmethod 方法参数没有要求

class Pig:

@staticmethod

def eat(food):

print('The pig is eating : ', food)

p = Pig()

p.eat('grass')

print('-' \* 80)

#函数装饰器

def foo(fn):

print('foo function')

print(fn)

print('PythonFunction')

@foo

def bar():

print('bar function')

print(bar)

print('-' \* 80)

foo(bar)

print('-' \* 80)

#函数装饰器与AOP

def aop(fn):

print('AOP function ...')

def aspect(\*args):

if args[0] < 100:

return 0

print('Arguments are : ',args[0], args[1])

for arg in args:

print('Argument is : ', arg)

print('Before the function execute ...')

fn(\*args)

print('After the function execute ...')

return 1

return aspect

@aop

def fun(a, b):

print('a is : ', a)

print('b is : ', b)

fun(10, 200)

print('-' \* 80)

#类变量与实例变量

class ClassA:

v1 = 'classVariable1'

def \_\_init\_\_(self, name='Harry',age=36):

self.name = name

self.age = age

print('Old class variable : ', ClassA.v1)

#新增类变量

ClassA.v2 = 'classVariable2'

print('New class variable : ', ClassA.v2)

#实例只能访问类变量，不能修改类变量

c = ClassA('Tom', 25)

print('Variable 1 is : ' , c.v1)

print('Variable 2 is : ' , c.v2)

#如果试图通过实例增加类变量，结果就是增加了实例变量

c.v1 = 'AddVarialbe'

print("Variable is : ",c.v1)

print('Class Variable v1 is : ', ClassA.v1)

print(ClassA)

print('-' \* 80)

#合成属性property

class Rectangle:

def \_\_init\_\_(self, width, height):

self.width = width

self.height = height

def getarea(self):

return self.width \* self.height

def getsize(self):

return self.width, self.height

def setsize(self, size):

self.width = size[0]

self.height = size[1]

area = property(fget=getarea,doc='get the rectangle area')

size = property(fget=getsize,fset=setsize,doc='set and get rectangle size')

r = Rectangle(100, 200)

print('Rectangle area is : ', r.area)

print('Rectangle size is : ', r.size)

print('-' \* 80)

class SystemUser:

def \_\_init\_\_(self,name='None'):

if isinstance(name, str) and 4 <= len(name) <= 8:

self.\_\_name = name

else:

self.\_\_name = 'None'

def setName(self, name):

if isinstance(name, str) and 4 <= len(name) <= 8:

self.\_\_name = name

else:

self.\_\_name = '用户名无效!'

def getName(self):

return self.\_\_name

name = property(fget=getName, fset=setName)

su = SystemUser()

print("User name is : ", su.name)

su.setName('Jack')

print(su.getName())

print('Now the name is : ', su.name)

print('-' \* 80)

#类的继承及复写父类方法

class Fruit:

def info(self):

print('The fruit is good for your health.')

class Apple(Fruit):

def info(self):

print('I am an apple , and I am much helpful for your health.')

apple = Apple()

apple.info()

print('-' \* 80)

class Employee:

def work(self):

print('work hard 996...')

class Manager(Employee):

def work(self):

print('I am a manager , and I must work even harder ...')

def relax(self):

#self.work()

#调用父类的work方法

#方法一：

#Employee.work(self)

#方法二

super().work()

print('Even I am on vacation, I still have to work.')

employee = Employee()

employee.work()

manager = Manager()

manager.relax()

print('-' \* 80)

#调用父类的构造方法，方式和调用父类方法一样，两种方式

class Work:

def \_\_init\_\_(self, salary):

self.salary = salary \* 2

class Leader(Work):

def \_\_init\_\_(self, salary, position):

super().\_\_init\_\_(salary)

self.position = position

leader = Leader(8000, 'PartLeader')

print('Leader position : ', leader.position)

print('Leader salary : ',leader.salary)

print('-' \* 80)