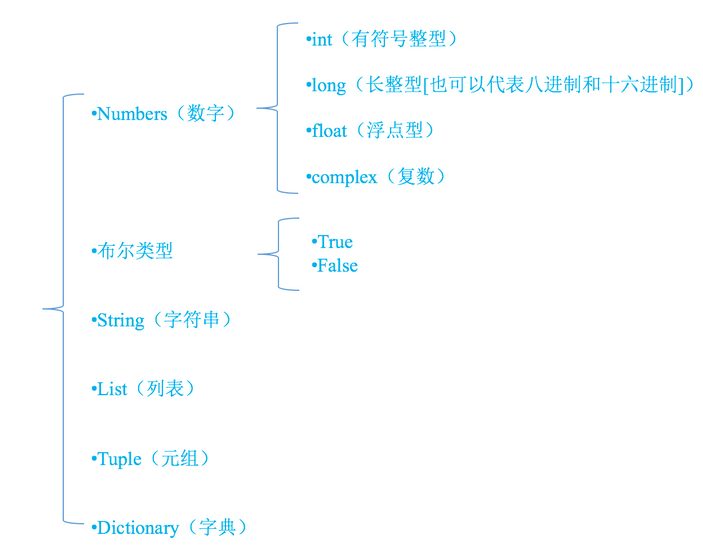
如果直接在程序中用到了中文，如果直接运行输出，程序会出错。解决的办法为：在程序的开头写入： #coding=utf-8或者# -\*- coding:utf-8 -\*-





键盘输入的数据的方法是采用 raw\_input() 函数

raw\_input()的小括号中放入的是，提示信息，用来在获取数据之前给用户的一个简单提示

raw\_input()在从键盘获取了数据以后，会存放到等号右边的变量中

raw\_input()会把用户输入的任何值都作为字符串来对待

input()函数与raw\_input()类似，但其接受的输入必须是表达式。

python3版本中没有raw\_input()函数，只有input()，并且 python3中的input与python2中的raw\_input()功能一样

for..in语句是另一个循环语句，它迭代一个对象的序列

for (int i = 0; i < 5; i++) 等价于 for i in range(0,5)。

if score>=90 and score<=100:

print('本次考试，等级为A')

elif score>=80 and score<90: //elif必须和if一起使用

print('本次考试，等级为B')

elif score>=70 and score<80:

print('本次考试，等级为C')

可以和else一起使用

if 性别为男性:

输出男性的特征

elif 性别为女性:

输出女性的特征

else:

第三种性别的特征

print(\*objects, sep=' ', end='\n', file=sys.stdout, flush=False)

print输出默认换行，在出入后面加上end=' '可不换行

continue的作用：用来结束本次循环，紧接着执行下一次的循环

break/continue只能用在循环中，只对最近的一层循环起作用，除此以外不能单独使用

双引号或者单引号中的数据，就是字符串

列表list[] 其中元素类型可以不同，元素也可以是另一个list

列表是个可变的有序表，可以使用append追加元素到末尾classmates.append('Adam')

可以用insert插入到指定位置classmates.insert(1, 'Jack')

使用pop删除list末尾的元素classmates.pop() 删除指定位置classmates.pop(i)

替换可以直接赋值 classmates[1] = 'Sarah'

元祖 tuple

Python的元组与列表类似，不同之处在于元组的元素不能修改，不能删除。

元组使用小括号，列表使用方括号。列表和元组中元素可以是不同类型的

testList = [1, 'a']

aTuple = ('et',77,99.9)

1个元素的tuple定义时必须加一个逗号,，来消除歧义 t = (1,)

字典：info = {'name':'班长', 'id':100, 'sex':'f', 'address':'地球亚洲中国北京'}

字典和列表一样，也能够存储多个数据

列表中找某个元素时，是根据下标进行的

字典中找某个元素时，是根据'名字'（就是冒号:前面的那个值，例如上面代码中的'name'、'id'、'sex'）

字典的每个元素由2部分组成，键:值。例如 'name':'班长' ,'name'为键，'班长'为值

不确定字典中是否存在某个键而又想获取其值时，可以使用get方法

字典元素可修改，通过key找到，即可修改，修改时如果不存在这个值，就会添加新的元素

删除字典：del info['name'] del info

清空字典：info.clear()

切片 语法：[起始:结束:步长]

切片是指对操作的对象截取其中一部分的操作。字符串、列表、元组都支持切片操作。

选取的区间属于左闭右开型，即从"起始"位开始，到"结束"位的前一位结束

name = 'abcdef'

print(name[0:3]) # 取 下标0~2 的字符

print(name[2:]) # 取 下标为2开始到最后的字符

print(name[1:-1]) # 取 下标为1开始到最后第2个之间的字符

遍历

通过for…in…的语法结构可以遍历字符串、列表、元组、字典

引用

在python中，值是靠引用来传递来的。

我们可以用id()来判断两个变量是否为同一个值的引用 id(a)

定义函数

def 函数名(): //使用函数名调用

代码

如果在函数中修改全局变量，那么就需要使用global进行声明后再重新赋值，否则出错

如果全局变量的名字和局部变量的名字相同，那么使用的是局部变量的

在函数中不使用global声明全局变量时不能修改全局变量的本质是不能修改全局变量的指向，即不能将全局变量指向新的数据。

对于不可变类型的全局变量来说，因其指向的数据不能修改，所以不使用global时无法修改全局变量。

对于可变类型的全局变量来说，因其指向的数据可以修改，所以不使用global时也可修改全局变量。

函数参数

缺省参数

调用函数时，缺省参数的值如果没有传入，则被认为是默认值。带有默认值的参数一定要位于参数列表的最后面。

def information( name, age = 35 ):

不定长参数

需要一个函数能处理比当初声明时更多的参数，不定长参数声明时不会命名

def functionname([formal\_args,] \*args, \*\*kwargs):

"函数\_文档字符串"

function\_suite

return [expression]

引用传参

Python中函数参数是引用传递（注意不是值传递）。对于不可变类型，因变量不能修改，所以运算不会影响到变量自身；而对于可变类型来说，函数体中的运算有可能会更改传入的参数变量。

匿名函数 lambda [arg1 [,arg2,.....argn]]:expression

lambda关键词能创建小型匿名函数

文件

打开和关闭

使用open函数，可以打开一个已经存在的文件，或者创建一个新文件

open(文件名，访问模式)

close( )

# 新建一个文件，文件名为:test.txt

f = open('test.txt', 'w')

# 关闭这个文件

f.close()



2. 文件的读写

使用write()可以完成向文件写入数据

f = open('test.txt', 'w')

f.write('hello world, i am here!')

f.close()

read(num)可以从文件中读取数据，num表示要从文件中读取的数据的长度（单位是字节）

f = open('test.txt', 'r')

content = f.read(5)

print(content)

readlines可以按照行的方式把整个文件中的内容进行一次性读取，并且返回的是一个列表，其中每一行的数据为一个元素

文件备份

定位读写

读写文件的过程中，使用tell()来获取当前的位置，使用seek()定位到另外一个位置进行操作

seek(offset, from)有2个参数

offset:偏移量

from:方向

0:表示文件开头

1:表示当前位置

2:表示文件末尾

文件重命名

import os

os.rename("毕业论文.txt", "毕业论文-最终版.txt")

删除文件

import os

os.remove("毕业论文.txt")

文件夹操作

import os

os.mkdir("张三") #创建文件夹

os.getcwd() #获取当前目录

os.chdir("../") #改变默认目录

os.listdir("./") #获取目录列表

os.rmdir("张三") #删除文件夹

**面向对象编程(Object Oriented Programming-OOP)**

面向对象编程的2个非常重要的概念：类和对象

对象是面向对象编程的核心，可以是直接使用的，在使用对象的过程中，为了将具有共同特征和行为的一组对象抽象定义，提出了另外一个新的概念——类，一个类可以找到多个对象

类就相当于制造飞机时的图纸，用它来进行创建的飞机就相当于对象

类就是创建对象的模板

类的构成

类(Class) 由3个部分构成

类的名称:类名

**类的属性:一组数据**

**类的方法:允许对进行操作的方法 (行为)**

定义类

class Car: #car为类名

def getCarInfo(self):

print('车轮子个数:%d, 颜色%s'%(self.wheelNum, self.color))

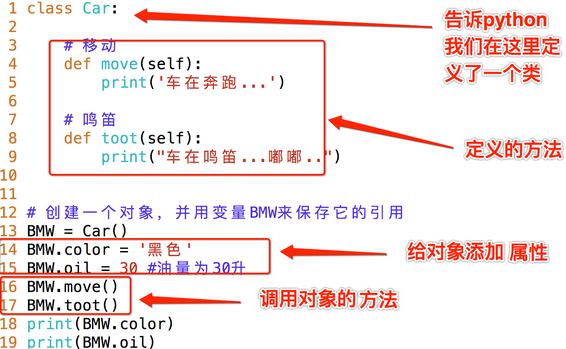
def move(self):

print("车正在移动...")

定义类时有2种：新式类和经典类，上面的Car为经典类，如果是Car(object)则为新式类

创建对象

可以根据已经定义的类去创建出一个个对象：对象名 = 类名()



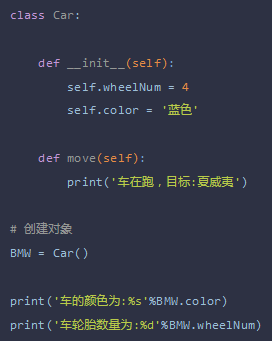
\_\_init\_\_()方法

def 类名:

#初始化函数，用来完成一些默认的设定,只要创建类的一个实例,就会运行这个方法

def \_\_init\_\_():

pass



\_\_init\_\_()方法，在创建一个对象时默认被调用，不需要手动调用

\_\_init\_\_(self)中，默认有1个参数名字为self，如果在创建对象时传递了2个实参，那么\_\_init\_\_(self)中除了self作为第一个形参外还需要2个形参，例如\_\_init\_\_(self,x,y)

\_\_init\_\_(self)中的self参数，不需要开发者传递，python解释器会自动把当前的对象引用传递进去

\_\_str\_\_()方法

python中方法名如果是\_\_xxxx\_\_()的，那么就有特殊的功能，因此叫做“魔法”方法

print输出对象的时候，只要自己定义了\_\_str\_\_(self)方法，那么就会打印从在这个方法中return的数据

def \_\_str\_\_(self):

msg = "嘿。。。我的颜色是" + self.color + "我有" + int(self.wheelNum) + "个轮胎..."

return msg

self

所谓的self，可以理解为自己

可以把self当做C++中类里面的this指针一样理解，就是对象自身的意思

某个对象调用其方法时，python解释器会把这个对象作为第一个参数传递给self，所以开发者只需要传递后面的参数即可

隐藏数据

修改对象的属性可以直接通过对象名修改，也可以通过方法间接修改，直接修改改的次数较多，间接只需要调用一次

保护对象的属性

将属性定义为私有属性，添加一个可以调用的方法，供调用

以属性命名方式来区分，如果在属性名前面加了2个下划线'\_\_'，则表明该属性是私有属性，否则为公有属性（方法也是一样，方法名前面加了2个下划线的话表示该方法是私有的，否则为公有的）。

删除对象

创建对象后，python默认调用\_\_init\_\_()方法；删除对象时，python默认调用\_\_del\_\_()方法

当有1个变量保存了对象的引用时，此对象的引用计数就会加1

当使用del删除变量指向的对象时，如果对象的引用计数不是 1，比如3，那么此时只会让这个引用计数减1，即变为2，当再次调用del时，变为1，如果再调用1次del，此时会真的把对象进行删除

继承

1. 单继承

继承描述的是事物之间的所属关系，子类没有定义\_\_init\_\_方法，但是父类有，在子类继承父类的时候这个方法就被继承了，所以只要创建对象，就默认执行了那个继承过来的\_\_init\_\_方法

子类在继承的时候，在定义类时，小括号()中为父类的名字

父类的属性、方法，会被继承给子类

私有的属性，不能通过对象直接访问，但是可以通过方法访问

私有的方法，不能通过对象直接访问

私有的属性、方法，不会被子类继承，也不能被访问

一般情况下，私有的属性、方法都是不对外公布的，往往用来做内部的事情，起到安全的作用

1. 多继承

在定义子类时，小括号()中为父类的名字 class C(A,B):

如果父类A和父类B中，有一个同名的方法，那么通过子类去调用的时候有调用顺序

print(C.\_\_mro\_\_) #可以查看C类的对象搜索方法时的先后顺序

重写与调用父类方法

重写就是子类中，有一个和父类相同名字的方法，在子类中的方法会覆盖掉父类中同名的方法，如果你的父类方法的功能不能满足你的需求，你可以在子类重写你父类的方法

调用 super().\_\_init\_\_(参数)

调用父类的\_\_init\_\_方法1(python2) Cat.\_\_init\_\_(self,参数)

调用父类的\_\_init\_\_方法2 super(子类名称,self).\_\_init\_\_(参数)

调用父类的\_\_init\_\_方法3 super().\_\_init\_\_(参数)

多态

定义时的类型和运行时的类型不一样，此时就成为多态

类属性、实例属性（对象属性）

类属性就是类对象所拥有的属性，它被所有类对象的实例对象所共有，在内存中只存在一个副本，这个和C++中类的静态成员变量有点类似。对于公有的类属性，在类外可以通过类对象和实例对象访问

class People(object):

name = 'Tom' #公有的类属性

\_\_age = 12 #私有的类属性

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

self.name = 'xiaowang' #实例属性

self.age = 20 #实例属性

p = People()

p.age =12 #实例属性

print(p.name) #正确

print(People.name) #正确

print(p.\_\_age) #错误，不能在类外通过实例对象访问私有的类属性

print(People.\_\_age) #错误，不能在类外通过类对象访问私有的类属性

print(p.address) #正确

print(p.name) #正确

print(p.age) #正确

print(People.address) #正确

print(People.name) #错误

print(People.age) #错误

p.name=’eeeww ’ #通过实例（对象）修改类属性

del p.name #删除实例属性

如果需要在类外修改类属性，必须通过类对象去引用然后进行修改。如果通过实例对象去引用，会产生一个同名的实例属性，这种方式修改的是实例属性，不会影响到类属性，并且之后如果通过实例对象去引用该名称的属性，实例属性会强制屏蔽掉类属性，即引用的是实例属性，除非删除了该实例属性。

静态方法和类方法

普通方法可以通过self访问实例属性,类方法可以通过cls访问类属性,静态方法不可以访问

1. 类方法

是类对象所拥有的方法，需要用修饰器@classmethod来标识其为类方法，对于类方法，第一个参数必须是类对象，一般以cls作为第一个参数，能够通过实例对象和类对象去访问。

类方法还有一个用途就是可以对类属性进行修改

1. 静态方法

需要通过修饰器@staticmethod来进行修饰，静态方法不需要多定义参数

类方法的第一个参数是类对象cls，那么通过cls引用的必定是类对象的属性和方法；

而实例方法的第一个参数是实例对象self，那么通过self引用的可能是类属性、也有可能是实例属性（这个需要具体分析），不过在存在相同名称的类属性和实例属性的情况下，实例属性优先级更高。

静态方法中不需要额外定义参数，因此在静态方法中引用类属性的话，必须通过类对象来引用

工厂模式

1. 简单工厂模式
2. 工厂方法模式

定义了一个创建对象的接口(可以理解为函数)，但由子类决定要实例化的类是哪一个，工厂方法模式让类的实例化推迟到子类，抽象的CarStore提供了一个创建对象的方法createCar，也叫作工厂方法。

\_\_new\_\_方法

class A(object):

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

def \_\_new\_\_(cls):

\_\_new\_\_至少要有一个参数cls，代表要实例化的类，此参数在实例化时由Python解释器自动提供

\_\_new\_\_必须要有返回值，返回实例化出来的实例，可以return父类\_\_new\_\_出来的实例，或者直接是object的\_\_new\_\_出来的实例

\_\_init\_\_有一个参数self，就是这个\_\_new\_\_返回的实例，\_\_init\_\_在\_\_new\_\_的基础上可以完成一些其它初始化的动作，\_\_init\_\_不需要返回值

单例模式

确保某一个类只有一个实例，而且自行实例化并向整个系统提供这个实例，这个类称为单例类，单例模式是一种对象创建型模式。

模块

import关键字引入模块

调用模块中的函数格式 模块名.函数名 print math.sqrt(2)

如果只用到模块中某个函数，只需要引入该函数

两个模块含有相同名称函数的时候，后面一次引入会覆盖前一次引入。

from 模块名 import 函数名1,函数名2....

把一个模块的所有内容全都导入到当前的命名空间 from modname import \*

定位模块

导入一个模块，Python解析器对模块位置的搜索顺序是：

1. 当前目录
2. 如果不在当前目录，Python则搜索在shell变量PYTHONPATH下的每个目录。
3. 如果都找不到，Python会察看默认路径。UNIX下，默认路径一般为/usr/local/lib/python/
4. 模块搜索路径存储在system模块的sys.path变量中。变量里包含当前目录，PYTHONPATH和由安装过程决定的默认目录。

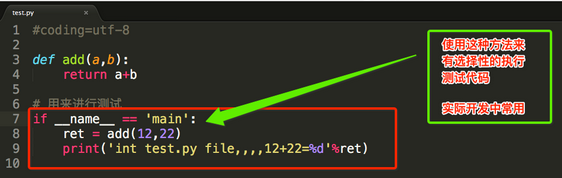
模块制作

在Python中，每个Python文件都可以作为一个模块，模块的名字就是文件的名字。

调用自己定义的模块

import 文件名，之后通过文件名.函数名调用

测试模块，为了让模块能够在项目中达到想要的效果，自行在py文件中添加一些测试信息



模块中的\_\_all\_\_

如果一个文件中有\_\_all\_\_变量，那么也就意味着这个变量中的元素，不会被from xxx import \*时导入

包

包将有联系的模块组织在一起，即放到同一个文件夹下，并且在这个文件夹创建一个名字为\_\_init\_\_.py 文件，那么这个文件夹就称之为包

有效避免模块名称冲突问题，让应用组织结构更加清晰



\_\_init\_\_.py 控制着包的导入行为，为空时仅仅是把这个包导入，不会导入包中的模块。文件中，定义一个\_\_all\_\_变量，它控制着 from 包名 import \*时导入的模块