Why python?

1. python的使用：①算法，数据分析②爬虫③算法④web后端⑤。。。

①AI和ML，人工智能，机器学习②数据分析和算法领域③爬虫④web后端开发⑤自动化测试，运维，辅助脚本

运行环境：windows,Linux,MacOS,www

**编程规范：函数名，变量名使用下划线隔开的方式命名；类——大驼峰。**

**方法：先写框架然后再进行细节化：**

**if num == 1: #框架**

**pass**

**if num == 2:**

**pass**

**...**

2.Python3：有很多地方与python2不兼容，比如<>不等号，开发者普遍不使用这个符号，因此，python3删除，

3.学习：

①python3的常用语法，

②面向对象编程思维，

③运用模块进行编程，

④游戏编程，

⑤计算机仿真：对真实世界的建模

1. shell一个通过键入文本与程序交互的途径，eg:windows的dos窗口，linux的命令窗口......IDLE,python的shell

Alt + N:回到上一条命令

Alt + P：回到下一条命令

ctrl + A：到命令行第一个字符处, ctrl + E

1. python中变量的命名，类 -> 大驼峰，变量函数等,小写，下划线连接

eat\_apple()

1. 不像其他语言（C++, java 等等），就拿java举个例子：我们都知道在java里面方法后面是用{}来区分是否属于一个方法的，但在python里面方法体并不使用{}来区分，python的是用缩进来识别语法逻辑块的(i.e. if, while, for, def 等）。同一个if逻辑块下面的语句必须要有相同的缩进，否则python解析器怎么知道if块控制哪些语句？对吧，在python中，所有的逻辑代码块也就是一个方法中的代码，都必须使用相同的缩进来标识区分是同一个方法，否则编译会报错。
2. **deb文件安装：**谷歌浏览器比其他浏览器运行起来流畅多了啊，直接去官网下载deb文件,deb文件的安装： sudo dkpg -i xxxx.deb

16.\\ ——输出反斜杠

输出特殊符号：‘’，“”时，print(“Let\’s go!”) #加\ 输出

输出一次输出多个转移字符的话：三引号

“””abc\nabc\abc\n””” #三层反斜杠

1. 比较运算符的优先级 > 逻辑运算符

Eg: 3>2 and 1<2 = (3>2) and (1<2)

1. 条件表达式,唯一的一个三目运算符，

：small = x if x < y else y

等于这种：small = x < y ? x : y;

1. **assert(断言)————使得程序崩溃，自爆，用于调试程序中，逻辑正常时，正常运行，出错时，程序自动崩溃，报：asserterror错误。从而避免后面的编译出错...**

**抛出异常**

**异常处理**

1. python中的for循环和其他语言的不太一样，它可以自动调用迭代器。

for xxx in yyy:

zzz

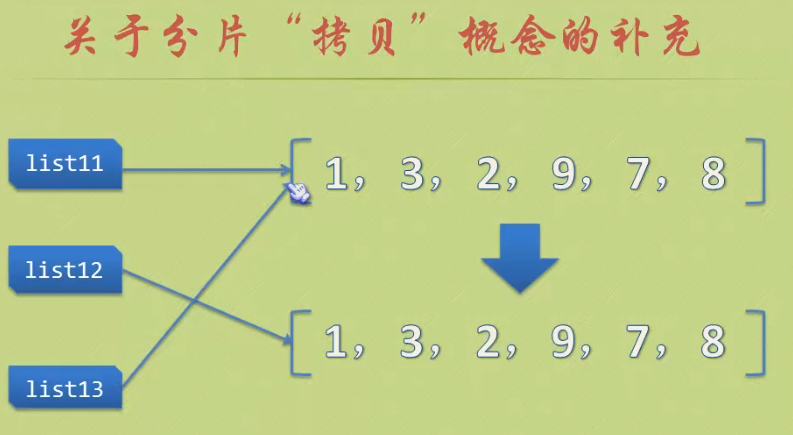
1. list（列表）————打了激素的数组，数组进阶版，可以存放不同类型的数据

Member = [1, 1.23, ‘float’]

Member.append(1)

Member.extend([“abc”, ”def”]) #使用别的列表来扩展这个列表

23.



**NOTE：拷贝，Python中的赋值，仅仅是多了一个指向原有数组内存空间的标签，而不会再创建一个列表，开辟一个新的空间。因此，当你对新的列表进行操作的时候原有列表也发生了变化。这个地方，很容易导致程序出错，所以在复制列表的时候，尽量使用过分片（slice）这个语句: list2 = list1[ : ] #这样就会产生一个新的和原来一模一样的列表**

Eg:

list1 = [1, 2, 3, 4]

list3 = list1

list3.reverse() #list1也发生了变化

1. tuple(元组)

和列表类似，但是不能随意修改其中的值。区分元组的标志：有无逗号。

8 \* (8) #返回64

8 \* (8, ) #返回 （8，8，8，8，8，8，8，8）

1. str的一些列操作：



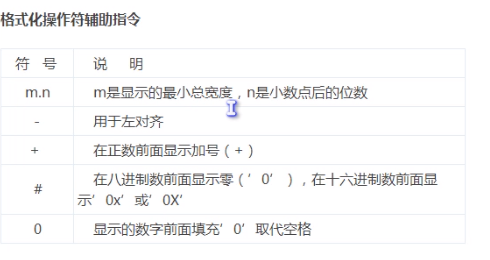
1. 格式化字符串：即按照固定的格式输出字符串，类似于C中的printf()函数，不过没有逗号

：print(‘%c + %c = %c’ %(97, 98, 99)) #没有逗号，否则报错

‘%d + %d = %d’ % (4, 5, 4+5) # %d



输出格式：（格式化）’%m.nf’ % 1.24 #浮点型，默认保留小数点后六位，补零



**26.列表 元组 字符串**

**27.函数 对象 模块————重中之重**

形参（parameter）

实参（argument）

一般来说，系统的特殊属性都是：以双下划线开始，双下划线结束——**魔法方法**

eg : MyFirstFunction.\_\_doc\_\_

1. 关键字参数和默认参数：

手机参数，即可变参数，参数个数不确定，是可以变得。生成机制，将传入的参数用一个元组打包起来。

eg:

def SaySome(\*params) : #参数个数任意

Print(‘参数个数：’ ， len(params))

Print(‘第二个参数是：’， params[1])

1. 函数 与 过程

因为python数据类型不确定，所以返回值也不确定，甚至可以返回list(列表)。不像其他语言一样，返回值类型固定。

1. 局部变量与全局变量

当全局变量被定义好时，不要试图在函数内部修改它的值，只能访问它的值，因为如果试图修改全局变量时，python会创建一个名字与全局变量一模一样的局部变量，但是与全局变量互不影响，因为内存空间不一致。所以，**全局变量只能在函数内部访问！！！**

1. **内嵌函数和闭包**

①面向对象

②面向过程

③函数式编程方式：重要语法结构，闭包 python

Eg1:

def FunX(x):

def FunY(y):

return x \* y

return FunY #FunX(5)(8) = 40

函数FunY在FunX外部的其他地方，都不能被调用,否则报错

i = FunX(5)

Type(i) #class ‘function’,返回值为函数类型，即对象

误区：

eg2:

def FunX(x):

x = 5

def FunY():

x \*= x #这里会报错，x对于FunY来说‘相当于’一个全局变量，在这 #个函数内部是不可以进行修改的，解决办法和全局变量差不多

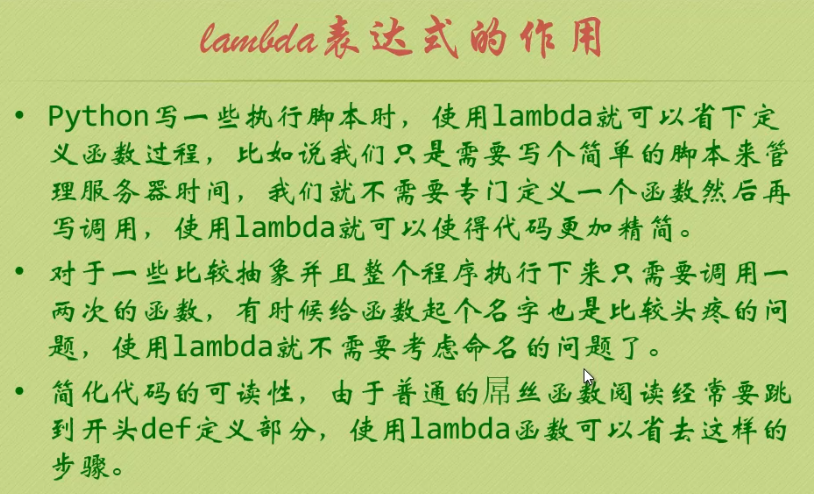
#在x \*= x前面加： nonlocal x 类比 global x

#pyhon3才有这种解决方法，python2没有

return x

return FunY

1. lambda表达式



lambda表达式的用处：

传统方法：

def g(x):

return 2 \* x + 1

a = g(1)

lambda方法:

g = lambda x : 2 \* x + 1 # ’:’左侧，输入参数，右侧，返回值

g(5) # return 11

两个参数时：

add = lambda x, y : x + y

add(1, 2)

1. bif:

filter:

map() #不是地图的意思，是映射

34递归：**普通程序员使用迭代 天才程序员使用递归**

汉诺塔游戏，树结构。谢尔并司机三角形

斐波那契数列：1，1，2，3，5，8，13，21，34，55，89，144

前一个数字与后一个数字的比例越来越接近0.618：1（黄金分割比），因此，算法上如果想得到黄金分割比通常使用斐波那契数列来代替。迭代次数如果太久的话，会非常影响工作效率，速度会非常慢。因此，**在迭代次数少的时候可以使用递归，稍大就不适合了**

1. 字典：

字典不是序列类型，是**映射类型**；

list(列表)，tuple(元组),str(string,字符串)都是序列类型；

eg:dict1 = dict(((1, ‘ont’), (2, ‘two’), (3, ‘three’), (4, ‘four’), (5, ‘five’)))

1. dict1.clear() #删除函数，return ,dict1 = {}

最好使用clear命令，不要用：dict1 = {},因为如果：

dict1 = {‘1’ : ‘one’, ‘2’ : ‘two’, ‘3’ : ’three’}

dict2 = dict1 #相当于dict2也指向这个字典

dict1 = {} #相当于dict1指向了一个新的空的字典

dict2 #dict2没有改变，因此，最好使用dict1.clear()，彻底删除

1. dir(dict) #列出dict所有可能的用法，

赋值号：浅拷贝

dict.copy() #赋值一个一模一样的字典，新的空间

eg:

a = {1:’one’, 2:’two’, 3:’three’}

b = a #浅拷贝,id(b) = id(a)

c = a.copy() #深拷贝，新开辟一块内存，id(c) != id(a)

1. range函数，包头不含尾的...

range(10) = range(0, 9) #0 ~ 9

1. 集合（set）：**唯一，自动清理重复元素，无序，无法索引**

定义方法：①花括号{}；②set()

num1 = {1, 1, 2, 2, 3, 3}

num1.add(4)

num1.remove(4)

frozen——不可变集合

1. 文件的输入与输出：





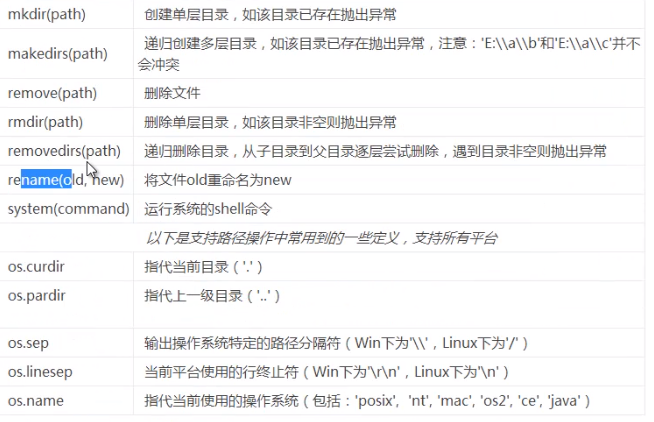
有一个很重要的问题，f.read()等指令都是从‘书签’的位置开始读取。如果使用了：f.read()读取了所有的内容以后，书签已经指向了最后一个空字符（字符串结束标志），再使用f.read()，等命令，只能读出 ‘ ’（空）。所以使用f。指令前，最好先使用f.tell() 看看书签在哪儿。

f.tell() #返回书签，即读取到哪里了

f.seed(0, 0) #将书签移动到字符串第一个字符位置——起始位置

40. os module

import os



**POSIX表**示可移植操作系统接口（Portable Operating System Interface of UNIX，缩写为 POSIX ），POSIX标准定义了操作系统应该为应用程序提供的接口标准。

windows: [\\,](\\\\,) \为转义字符， ’\\’即输出一个反下划线 ’\’

linux: /

‘ . ’ : 当前目录

‘ .. ’ : 上一级目录

POSIX标准意在期望获得源代码级别的软件可移植性。换句话说，为一个POSIX兼容的操作系统编写的程序，应该可以在任何其它的POSIX操作系统（即使是来自另一个厂商）上编译执行。

1. module —— os.path





import os.path

**getatime(),getctime() create time,getmtime() modify time ,返回的都是浮点类型的数据，如果想要查看这个时间要引入时间模块。**

import time

gmtime() #世界标准时间

localtime() #得到当地时间

os.path.getatime(‘/home/qz/Desktop/python/test.txt’) #返回的数据类型为浮点型

time.localtime(os.path.getatime(‘/home/qz/Desktop/python/test.txt’)) #得到北京时间

os.path.ismount() #判断是否是**挂载点**，return:True or False

eg:

windows:

os.path.ismoutn(‘E:\\teset’) # False

os.path.ismount(‘E:\\’) # True ,C , D, E , ...

eg:

Linux:

os.path.ismount(‘ /home/qz/’) #False

os.path.ismount(‘/’) # True

绝对路径

相对路径

1. 一个将列表，字典，类等对象转换成为二进制的模块————**pickle**

import pickle #可以以二进制保存数据pickle.dump()以及以二进制打开数据pickle.load()

import os

os.getcwd #读取文件的路径，创建文件的保存路径，查看当前路径

my\_list = [1, 2.34, 'abc', [1, 2, 3]] #创建一个新的列表

pickle\_file = open(‘my\_pickle.pkl’, ‘wb’) #open函数内部没有my\_pickle.pkl时，会自动创建 #一个，打开方式，写二进制，这时候在该路径下 #已经可以查看到my\_pickle.pkl文件存在

pickle.dump(my\_list, pickle\_file) #二进制方式写入文件中

pickle\_file.close() #关闭该文档

pickle\_f = open(“my\_pickle.pkl”, ‘rb’) #读二进制方式

list2 = pickle.load(pickle\_f)

pickle\_f.close()

print(list2)

**pickle的使用：**

字典等很大的数据，尽量不要存放在函数中，应该存在外部，使用pickle存放，到时候直接调用就可以。

eg:天气查询系统（一个脚本）：

1. Exception————python的异常处理机制

常见错误（ERROR）



常见错误

1. 报错处理：try except

eg1:

try :

f = open(‘1.txt’)

except OSError as reason: #将reason 转成str类型输出

print(“出错啦, 原因是：” + str(reason))

eg2:

try: #被检测的程序

except: #出错后执行的程序

finally: #无论如何都会执行的程序

引发异常的raise语句：

raise ZeroDivisionError #会引发一个ZeroDivisionError异常

raise ZeroDivisionError(‘除数不能为零’

1. 当while中有break语句打断时，else不会执行。

当while中没有被break语句打断时，else就会执行。

while:

else: #如果while中的循环全部都执行完了，没有跳出，则执行else语句

eg1:

try:

int('123') #ValueError

except (ValueError) as reason: #reason is't str type

print('A error happened:' + str(reason))

else:

print('No error!')

except中的语句没有执行，因此执行else语句中的内容。

eg2:eg3:

for i in range(3):

if(i == 3):

break

print(i)

else:

print('execute Else T\_T') ##not 'break',execut 'else'

eg4:

for i in range(3):

if(i == 2):

break

print(i)

else:

print('execute Else T\_T') ##execute, 'break',No 'else'

1. **GUI**

GUI(Graphical User Interface)——图形用户界面编程,按钮等......

import easygui #easygui模块

easygui.msgbox(‘a’)

1. **对象—————python，无处不对象**

**对象 = 属性（静态，变量） + 方法（函数）**

**属性名用名词； 方法名用动词**

eg:

class Turtle: ## 类，通过类来定义对象

## shu xing

color = 'green'

weight = '10'

gender = 'male'

## fang fa

def climb(self):

print('pa')

def run(self):

print('pao')

def eat(self):

print('chi')

创建对象，也叫做类的实例化。

tt = Turtle()

**继承 + 封装 + 多态**

**继承：子类自动继承父类的数据和方法的机制。**

**封装在内部，**

**多态：不同对象对同一方法响应不同的行动。**

eg:继承

class MyList(list): #ji cheng list

pass

list1 = MyList() # a object

list1.append(1) #继承了list类的append()方法

list1 #return [1]

eg:多态

class A():

def fun():

print(“我是小A”)

class B():

def fun():

print(“我是小B”)

a = A()

b = B()

a.fun() #

b.fun() #fun函数来说，名称相同，但是输出内容不一致，即**多态**

**47.python 类的魔法方法——在类被创建的时候即存在，类似于c++的构造函数，可以通过重写实现一些功能。**

**我们首先得从\_\_new\_\_(cls[,...])的参数说说起，\_\_new\_\_方法的第一个参数是这个类，而其余的参数会在调用成功后全部传递给\_\_init\_\_方法初始化，这一下子就看出了谁是老子谁是小子的关系。(我们比较两个方法的参数，可以发现\_\_new\_\_方法是传入类(cls)，而\_\_init\_\_方法传入类的实例化对象(self)，而有意思的是，\_\_new\_\_方法返回的值就是一个实例化对象（ps:如果\_\_new\_\_方法返回None，则\_\_init\_\_方法不会被执行，并且返回值只能调用父类中的\_\_new\_\_方法，而不能调用毫无关系的类的\_\_new\_\_方法）。我们可以这么理解它们之间的关系，\_\_new\_\_是开辟疆域的大将军，而\_\_init\_\_是在这片疆域上辛勤劳作的小老百姓，只有\_\_new\_\_执行完后，开辟好疆域后，\_\_init\_\_才能工作。)**

\_\_init\_\_

\_\_del\_\_

eg:

class A(object):

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

print('\_\_init\_\_')

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

print('\_\_new\_\_')

return super().\_\_new\_\_(cls)

def \_\_del\_\_(self):

print('\_\_del\_\_')

1. 私有变量：（private)——————’\_\_name’，python中，双下划线表示私有变量。私有成员只能在类的内部被调用。

访问私有成员的方法：

def getName(self)

return \_\_name

a = Student()

1. getName()
2. 继承：

class 类名（父类名）：

如果父类的方法和子类的方法名称一致，则调用子类的方法，即，以自身为主，父类为辅。

1. 父类方法与子类方法一致时，默认执行的是子类的方法（子类优先，比如重写构造函数，父类的构造函数就不执行了），如果，想让他们都执行，解决方法：

①调用未绑定的父类方法；Fish.\_\_init\_\_(self)

②super函数；super().\_\_init\_\_(self)

1. 多重继承：

eg:

class Baes1:

def fool1(self):

print(‘a’)

class Base2:

def fool2(self)

print(‘b’)

class C(Base1, Base2): #**多重继承，慎用，可能会导致不可预知的BUG**

pass

c = C()

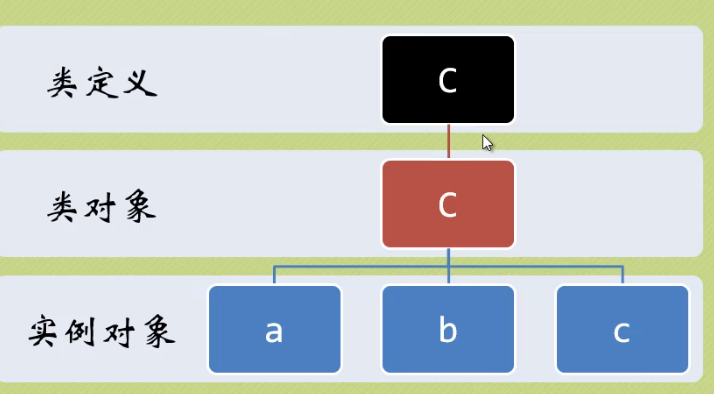
c.foo1() #基类Base1和Base2的方法都可以调用

c.fool2()

51.组合

组合可以把几个纵向没有关系的类，进行组合，形成一个新的类。

1. **类定义，类对象，实例对象**



step1:定义类 -> step2:类的实例化，得到对象 -> step3:对对象进行操作

52.static只有在程序退出的时候才会被释放,变量永久存在

issubclass(A, object) #object 是所有类的基类,创建即存在

1. property()

通过定义方法的方法来对属性进行操作，将方法定义好，封装在property内部，实现用户不变的情况下，修改属性的方法。

**BIF = Built in Function(内置函数)**

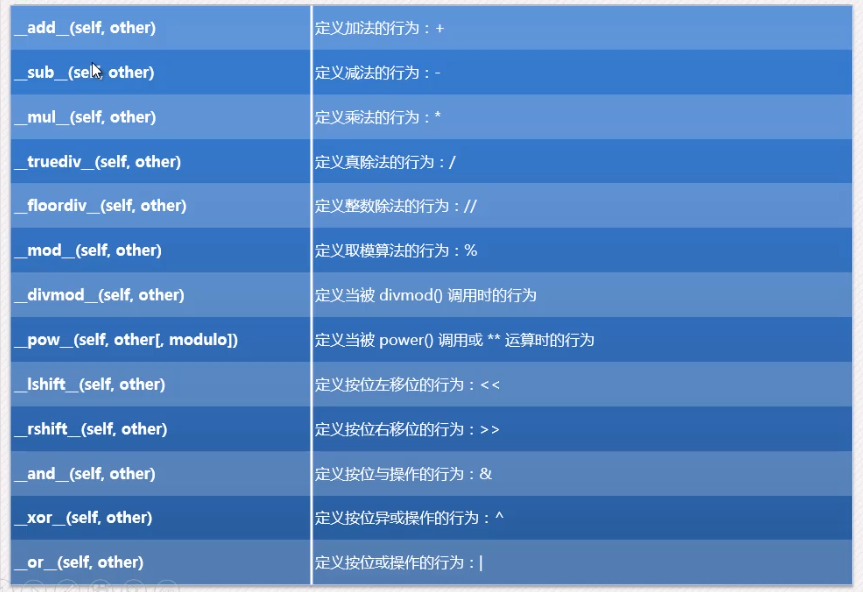
53 难点：P42，P43，P44,https://fishc.com.cn/forum.php?mod=forumdisplay&fid=243&filter=typeid&typeid=403

\_\_new\_\_ #构造函数，类被创建马上执行

\_\_del\_\_ #析构函数（垃圾回收机制）

所有指向这个对象的标签都被删除以后，才会执行这个垃圾回收机制。

54.类的算术运算(实际上就是C++中的运算符的重载)





eg反运算符：

class NewInt(int):

def \_\_radd\_\_(self, other): #这样新定义的NewInt类型有了radd()方法

return int.\_\_radd\_\_(self, other)

a = NewInt(5) #

1 + a #本来会报错，因为1是个整型，没有+这个方法，但是这里a是一个新的类型，上面写了radd方法，就可以执行a的radd方法，所以不会报错。当是减法等与运算顺序有关的运算符的时候，要注意self和other的顺序。

a是一个NewInt类型的变量，该变量有int类型的所有方法，唯独它的\_\_radd\_\_方法被重写了，

55.



import time as t

t.localtime()

**通过其他专门的模块可以实现精度更高的定时器。后续如果要对这个定时器进行升级的话可以试试。**

1. 属性访问的魔法方法：这四个函数可能会导致死循环，要认真检查。



访问顺序： \_\_getattribue\_\_()(该属性存在时) -> \_\_getattr\_\_() （属性不存在时）

**没有继承的父类时，默认为object类，还是有父类的（object）**，只是没有写。

class C(object): #对象类,object

def \_\_getattr\_\_(self, name):

print(‘a’)

return super().\_\_getattr(name) #super()函数，目的是不影响\_\_getattr()\_\_的原 #始功能

**在构造函数\_\_init(self)\_\_中写入形参时，在类的实例的时候就必须进行赋初值，负责报错，而其他的就不用\_\_getattr(self, name)，定义对象的时候不用传参。**

不能有空整行，为什么？？？

eg:计算面积：

默认为输入长和宽，如果为’square’属性赋值的话，默认触发”\_\_setattr\_\_”，为square赋值，

只要有赋值操作就会触发\_\_setattr\_\_,所以会报：’Recursion Error’，解决方法有两个：

①super().\_\_setattr\_\_(name, value)

②self.\_\_dict\_\_[name] = value

1. 属性访问，使用修改BIF的方法：**使用 ’super().\_\_getattribute\_\_’ 的原因是为了使\_\_getattr\_\_的功能同样生效。Note，例如：如果不加super().\_\_delattr\_\_(name),执行del c.x，只是打印出了内容，而没有执行del c.x需要的操作，继续输入 c.x, 还是会返回c.x的值，说明：c.x的属性还没有被删除。**

class C:

def \_\_getattribute\_\_(self, name): #属性在类内部存在时，getattribute没有找到时，调用getattr

print(‘getattribute’)

return super().\_\_getattribute\_\_(name)

def \_\_getattr\_\_(self, name):

print(‘getattr’)

super().\_\_getattr\_\_(name):

def \_\_setattr\_\_(self, name, value):

print(‘setattr’)

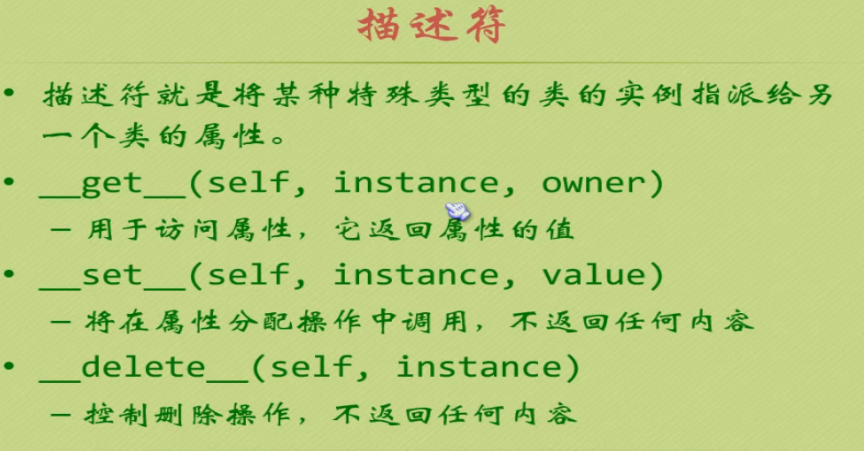
super().\_\_setattr\_\_(self, name)

def \_\_delattr\_\_(self, name):

print(‘delattr’)

super().\_\_delattr\_\_(self, name)

57.描述符（discriptor，描述符类）;就是将某种特殊类的实例，指派给另一个类的属性。



**input()输入的都是字符串。**

eg:

class MyDescriptor: #描述符类

def \_\_get\_\_(self, instance, owner):

print(‘getting’,self, instance, owner) #

def \_\_set\_\_(self, instance, value):

print(‘setting’,instance, value)

def \_\_delete\_\_(self, self, instance):

print(‘deleting’, self, instance)

class Test:

x = MyDescriptor()

t = Test() #Test（）类的实例

t.x

t.x = ‘abc’

del t.x #**使用和property()函数的使用几乎一致—>可以书写自己的property函数**

58.书写自己的property()函数，本质还是描述符类，

eg；**书写自己的property函数**

class MyProperty:

def \_\_init\_\_(self, fget, fset, gdel):

self.fget = fget

self.fset = fset

self.fdel = fdel

def \_\_get\_\_(self, instance, owner): #**只有get类函数需要返回值，要return**

return self.fget(instance)

def \_\_set\_\_(self, instance, value):

self.fset(instance, value)

def \_\_delete\_\_(self, instance)：

self.fdel(instance)

class Test:

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

self.x = None #attribute x

def getX(self):

return self.xd

def setX(self, value)

self.x = value

def delX(self, value)

del self.x

y = Myproperty(getX, setX, delX,)

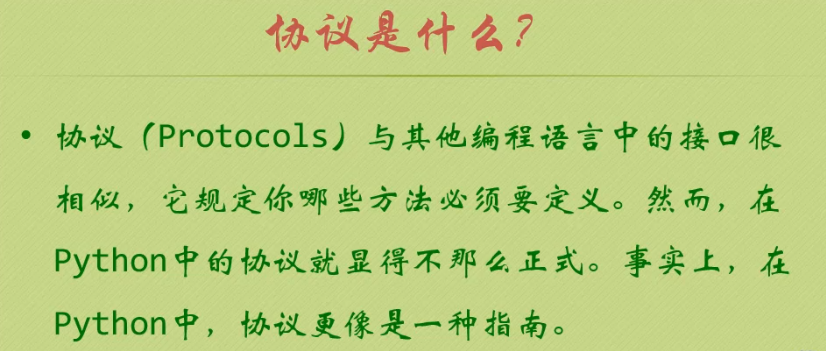
test = Test()

test.y #get

test.y = ‘I love you’

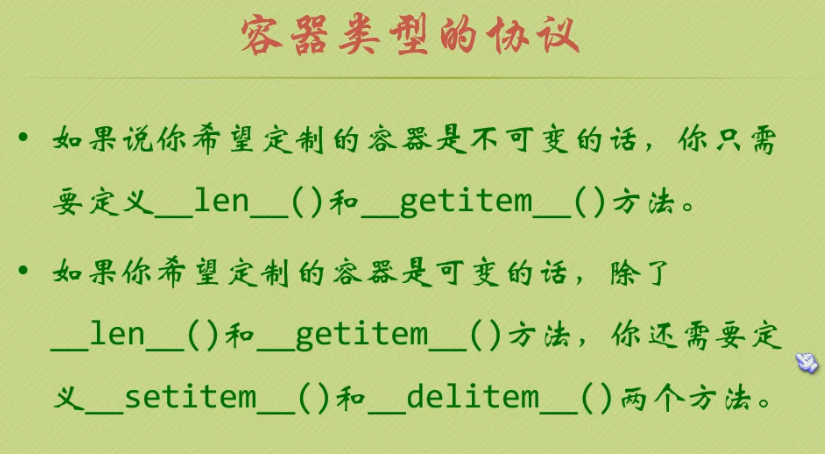
del test.y

1. **定制序列:**



**定制容器;**

鸭子：如果一只动物它走路像鸭子，吃东西像鸭子，做事也像鸭子，那我们就叫它鸭子。——> 同理可得，“容器”





**列表推导式**:list = [x for x in range[10]]

eg；定制序列,\_\_len\_\_, \_\_getitem\_\_

class MyList:

def \_\_init\_\_(self, \*parms) #\*表示参数的个数不确定

self.keys = [i for i in parms]

self.values = {}.fromkeys(range(len(self.keys)), ‘default’)

def \_\_len\_\_(self):

return len(self.keys)

def \_\_getitem\_\_(self, key):

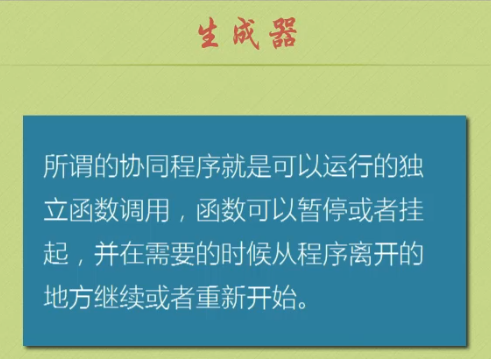
return self.values[key]

1. 迭代器：

iter() #获取xxx的迭代器

next()

1. 生成器——**一种特殊的迭代器**



yield,

1. **web spider(爬虫)eg:谷歌，百度等搜索引擎实现方法，每天有大量的爬虫在网上，将检索到的数据存入到自己的数据库中，再按照搜索热度进行排序，这样就可以在你搜索的时候将相关链接展现在我们面前了。**

import urllib #通过python访问互联网