**1、下面这段代码的输出结果是什么？请解释。**

def extendList(val, list=[]): list.append(val) return list

list1 = extendList(10) list2 = extendList(123,[]) list3 = extendList('a')

print "list1 = %s" % list1 print "list2 = %s" % list2 print "list3 = %s" % list3

怎样修改extendList的定义能够产生以下预期的行为?

上面代码输出结果将是：

list1 = [10, 'a'] list2 = [123] list3 = [10, 'a']

很多人都会误认为list1=[10]，list3=[‘a’],因为他们以为每次extendList被调用时，列表参数的默认值都将被设置为[].但实际上的情况是，新的默认列表只在函数被定义的那一刻创建一次。

当extendList被没有指定特定参数list调用时，这组list的值随后将被使用。这是因为带有默认参数的表达式在函数被定义的时候被计算，不是在调用的时候被计算。因此list1和list3是在同一个默认列表上进行操作（计算）的。而list2是在一个分离的列表上进行操作（计算）的。（通过传递一个自有的空列表作为列表参数的数值）。

extendList的定义可以作如下修改。

尽管，创建一个新的列表，没有特定的列表参数。

下面这段代码可能能够产生想要的结果。

def extendList(val, list=None): if list is None: list = [] list.append(val) return list

通过上面的修改，输出结果将变成：

list1 = [10] list2 = [123] list3 = ['a']

**2、下面这段代码的输出结果将是什么？请解释。**

def multipliers(): return [lambda x : i \* x for i in range(4)]

print [m(2) for m in multipliers()]

你如何修改上面的multipliers的定义产生想要的结果？

上面代码输出的结果是[6, 6, 6, 6] (不是我们想的[0, 2, 4, 6])。

上述问题产生的原因是Python闭包的延迟绑定。这意味着内部函数被调用时，参数的值在闭包内进行查找。因此，当任何由multipliers()返回的函数被调用时，i的值将在附近的范围进行查找。那时，不管返回的函数是否被调用，for循环已经完成，i被赋予了最终的值3。

因此，每次返回的函数乘以传递过来的值3，因为上段代码传过来的值是2，它们最终返回的都是6(3\*2)。碰巧的是，《The Hitchhiker’s Guide to Python》也指出，在与lambdas函数相关也有一个被广泛被误解的知识点，不过跟这个case不一样。由lambda表达式创造的函数没有什么特殊的地方，它其实是和def创造的函数式一样的。

下面是解决这一问题的一些方法。

一种解决方法就是用Python生成器。

def multipliers(): for i in range(4): yield lambda x : i \* x

另外一个解决方案就是创造一个闭包，利用默认函数立即绑定。

def multipliers(): return [lambda x, i=i : i \* x for i in range(4)]

还有种替代的方案是，使用偏函数：

from functools import partial from operator import mul

def multipliers(): return [partial(mul, i) for i in range(4)]

**3、下面这段代码的输出结果将是什么？请解释。**

class Parent(object): x = 1

class Child1(Parent): pass

class Child2(Parent): pass

print Parent.x, Child1.x, Child2.x Child1.x = 2 print Parent.x, Child1.x, Child2.x Parent.x = 3 print Parent.x, Child1.x, Child2.x

输出结果将是：

1 1 1 1 2 1 3 2 3

让很多人困惑或惊讶的是最后一行输出为什么是3 2 3 而不是 3 2 1.为什么在改变parent.x的同时也改变了child2.x的值？但与此同时没有改变Child1.x的值？

此答案的关键是，在Python中，类变量在内部是以字典的形式进行传递。

如果一个变量名没有在当前类下的字典中发现。则在更高级的类（如它的父类）中尽心搜索直到引用的变量名被找到。（如果引用变量名在自身类和更高级类中没有找到，将会引发一个属性错误。）

因此,在父类中设定x = 1,让变量x类(带有值1)能够在其类和其子类中被引用到。这就是为什么第一个打印语句输出结果是1 1 1

因此，如果它的任何一个子类被覆写了值（例如说，当我们执行语句Child1.x = 2）,这个值只在子类中进行了修改。这就是为什么第二个打印语句输出结果是1 2 1

最终，如果这个值在父类中进行了修改，（例如说，当我们执行语句Parent.x = 3）,这个改变将会影响那些还没有覆写子类的值（在这个例子中就是Child2）这就是为什么第三打印语句输出结果是3 2 3

**4、下面这段代码在Python2下输出结果将是什么？请解释。**

def div1(x,y): print "%s/%s = %s" % (x, y, x/y)

def div2(x,y): print "%s//%s = %s" % (x, y, x//y)

div1(5,2) div1(5.,2) div2(5,2) div2(5.,2.)

在Python3下结果会有怎样的不同？（当然，假设上述打印语句被转换成Python3的语法）

在Python2中，上述代码输出将是

5/2 = 2 5.0/2 = 2.5 5//2 = 2 5.0//2.0 = 2.0

默认情况下，Python 2 自动执行整形计算如果两者都是整数。因此,5/2 结果是2，而5./2结果是2.5

注意，在Python2中，你可以通过增加以下引用来覆写这个行为。

fromfutureimportdivision

同时要注意的是，//操作符将总是执行整形除法，不管操作符的类型。这就是为什么即使在Python 2中5.0//2.0的结果是2.0。然而在Python3中，没有此类特性，

例如，在两端都是整形的情况下，它不会执行整形除法

因此，在Python3中，将会是如下结果：

5/2 = 2.5 5.0/2 = 2.5 5//2 = 2 5.0//2.0 = 2.0

注： 在 Python 3 中，/ 操作符是做浮点除法，而 // 是做整除（即商没有余数，比如 10 // 3 其结果就为 3，余数会被截除掉，而 (-7) // 3 的结果却是 -3。这个算法与其它很多编程语言不一样，需要注意，它们的整除运算会向0的方向取值。而在 Python 2 中，/ 就是整除，即和 Python 3 中的 // 操作符一样）

**5、下面代码的输出结果将是什么？**

list = ['a', 'b', 'c', 'd', 'e'] print list[10:]

下面的代码将输出[],不会产生IndexError错误。就像所期望的那样，尝试用超出成员的个数的index来获取某个列表的成员。

例如，尝试获取list[10]和之后的成员，会导致IndexError.

然而，尝试获取列表的切片，开始的index超过了成员个数不会产生IndexError,而是仅仅返回一个空列表。

这成为特别让人恶心的疑难杂症，因为运行的时候没有错误产生，导致bug很难被追踪到。

**6、考虑下列代码片段**：

list = [ [ ] ] \* 5 list # output? list[0].append(10) list # output? list[1].append(20) list # output? list.append(30) list # output?

2,4,6,8行将输出什么结果？试解释。

输出的结果如下：

[[], [], [], [], []] [[10], [10], [10], [10], [10]] [[10, 20], [10, 20], [10, 20], [10, 20], [10, 20]] [[10, 20], [10, 20], [10, 20], [10, 20], [10, 20], 30]

解释如下：

第一行的输出结果直觉上很容易理解，例如 list = [ [ ] ] \* 5 就是简单的创造了5个空列表。然而，理解表达式list=[ [ ] ] \* 5的关键一点是它不是创造一个包含五个独立列表的列表，而是它是一个创建了包含对同一个列表五次引用的列表。只有了解了这一点，我们才能更好的理解接下来的输出结果。

list[0].append(10) 将10附加在第一个列表上。

但由于所有5个列表是引用的同一个列表，所以这个结果将是：

[[10],[10],[10],[10],[10]]

同理，list[1].append(20)将20附加在第二个列表上。但同样由于5个列表是引用的同一个列表，所以输出结果现在是：

[[10,20],[10,20],[10,20],[10,20],[10,20]]

作为对比， list.append(30)是将整个新的元素附加在外列表上，因此产生的结果是： [[10, 20], [10, 20], [10, 20], [10, 20], [10, 20], 30].

**7、Given a list of N numbers。**

给定一个含有N个数字的列表。

使用单一的列表生成式来产生一个新的列表，该列表只包含满足以下条件的值：

(a)偶数值

(b)元素为原始列表中偶数切片。

例如，如果list[2]包含的值是偶数。那么这个值应该被包含在新的列表当中。因为这个数字同时在原始列表的偶数序列（2为偶数）上。然而，如果list[3]包含一个偶数，

那个数字不应该被包含在新的列表当中，因为它在原始列表的奇数序列上。

对此问题的简单解决方法如下：

[xforxinlist[::2]ifx%2==0]

例如，给定列表如下：

list=[1,3,5,8,10,13,18,36,78]

列表生成式[x for x in list[::2] if x%2 == 0] 的结果是，

[10,18,78]

这个表达式工作的步骤是，第一步取出偶数切片的数字，

第二步剔除其中所有奇数。

**8、给定以下字典的子类，下面的代码能够运行么？为什么？**

class DefaultDict(dict): def \_\_missing\_\_(self, key): return []

d = DefaultDict() d['florp'] = 127

能够运行。

当key缺失时，执行DefaultDict类，字典的实例将自动实例化这个数列。

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

### **1、Python如何实现单例模式？**

Python有两种方式可以实现单例模式，下面两个例子使用了不同的方式实现单例模式：

1.

classSingleton(type): def\_\_init\_\_(cls,name,bases,dict): super(Singleton,cls).\_\_init\_\_(name,bases,dict) cls.instance=None def\_\_call\_\_(cls,\*args,\*\*kw): ifcls.instanceisNone: cls.instance=super(Singleton,cls).\_\_call\_\_(\*args,\*\*kw) returncls.instance classMyClass(object): \_\_metaclass\_\_=Singleton printMyClass() printMyClass()

1. 使用decorator来实现单例模式

defsingleton(cls): instances={} defgetinstance(): ifclsnotininstances: instances[cls]=cls() returninstances[cls] returngetinstance @singleton classMyClass: …

### **2：什么是lambda函数？**

Python允许你定义一种单行的小函数。定义lambda函数的形式如下：labmda 参数：表达式lambda函数默认返回表达式的值。你也可以将其赋值给一个变量。lambda函数可以接受任意个参数，包括可选参数，但是表达式只有一个：

>>>g=lambdax,y:x\*y >>>g(3,4) 12 >>>g=lambdax,y=0,z=0:x+y+z >>>g(1) 1 >>>g(3,4,7) 14

也能够直接使用lambda函数，不把它赋值给变量：

>>>(lambdax,y=0,z=0:x+y+z)(3,5,6) 14

如果你的函数非常简单，只有一个表达式，不包含命令，可以考虑lambda函数。否则，你还是定义函数才对，毕竟函数没有这么多限制。

### **3：Python是如何进行类型转换的？**

Python提供了将变量或值从一种类型转换成另一种类型的内置函数。int函数能够将符合数学格式数字型字符串转换成整数。否则，返回错误信息。

>>>int(”34″) 34 >>>int(”1234ab”)#不能转换成整数 ValueError:invalidliteralforint():1234ab

函数int也能够把浮点数转换成整数，但浮点数的小数部分被截去。

>>>int(34.1234) 34 >>>int(-2.46) -2

函数°oat将整数和字符串转换成浮点数：

>>>float(”12″) 12.0 >>>float(”1.111111″) 1.111111

函数str将数字转换成字符：

>>>str(98) ‘98′ >>>str(”76.765″) ‘76.765′

整数1和浮点数1.0在python中是不同的。虽然它们的值相等的，但却属于不同的类型。这两个数在计算机的存储形式也是不一样。

### **4：Python如何定义一个函数**

函数的定义形式如下：

def<name>(arg1,arg2,…argN): <statements>

函数的名字也必须以字母开头，可以包括下划线“ ”,但不能把Python的关键字定义成函数的名字。函数内的语句数量是任意的，每个语句至少有一个空格的缩进，以表示此语句属于这个函数的。缩进结束的地方，函数自然结束。 下面定义了一个两个数相加的函数：

>>>defadd(p1,p2): printp1,“+”,p2,“=”,p1+p2 >>>add(1,2) 1+2=3

函数的目的是把一些复杂的操作隐藏，来简化程序的结构，使其容易阅读。函数在调用前，必须先定义。也可以在一个函数内部定义函数，内部函数只有在外部函数调用时才能够被执行。程序调用函数时，转到函数内部执行函数内部的语句，函数执行完毕后，返回到它离开程序的地方，执行程序的下一条语句。

### **5：Python是如何进行内存管理的？**

Python的内存管理是由Python得解释器负责的，开发人员可以从内存管理事务中解放出来，致力于应用程序的开发，这样就使得开发的程序错误更少，程序更健壮，开发周期更短

### **6：如何反序的迭代一个序列？**

how do I iterate over a sequence in reverse order

如果是一个list, 最快的解决方案是：

list.reverse() try: forxinlist: “dosomethingwithx” finally: list.reverse()

如果不是list, 最通用但是稍慢的解决方案是：

foriinrange(len(sequence)-1,-1,-1): x=sequence[i] <dosomethingwithx>

### **7：Python里面如何实现tuple和list的转换？**

函数tuple(seq)可以把所有可迭代的(iterable)序列转换成一个tuple, 元素不变，排序也不变。 例如，tuple([1,2,3])返回(1,2,3), tuple(’abc’)返回(’a’.’b',’c').如果参数已经是一个tuple的话，函数不做任何拷贝而直接返回原来的对象，所以在不确定对象是不是tuple的时候来调用tuple()函数也不是很耗费的。 函数list(seq)可以把所有的序列和可迭代的对象转换成一个list,元素不变，排序也不变。 例如 list([1,2,3])返回(1,2,3), list(’abc’)返回['a', 'b', 'c']。如果参数是一个list, 她会像set[:]一样做一个拷贝

### **8：Python面试题：请写出一段Python代码实现删除一个list里面的重复元素**

可以先把list重新排序，然后从list的最后开始扫描，代码如下：

ifList: List.sort() last=List[-1] foriinrange(len(List)-2,-1,-1): iflast==List[i]:delList[i] else:last=List[i]

### **9：Python文件操作的面试题**

1. 如何用Python删除一个文件？ 使用os.remove(filename)或者os.unlink(filename);
2. Python如何copy一个文件？ shutil模块有一个copyfile函数可以实现文件拷贝

### **10：Python里面如何生成随机数？**

标准库random实现了一个随机数生成器，实例代码如下：

importrandom random.random()

它会返回一个随机的0和1之间的浮点数

### **11：如何用Python来发送邮件？**

可以使用smtplib标准库。 以下代码可以在支持SMTP监听器的服务器上执行。

importsys,smtplib fromaddr=raw\_input(”From:“) toaddrs=raw\_input(”To:“).split(’,') print“Entermessage,endwith^D:” msg=” while1: line=sys.stdin.readline() ifnotline: break msg=msg+line #发送邮件部分 server=smtplib.SMTP(’localhost’) server.sendmail(fromaddr,toaddrs,msg) server.quit()

### **12：Python里面如何拷贝一个对象？**

一般来说可以使用copy.copy()方法或者copy.deepcopy()方法，几乎所有的对象都可以被拷贝 一些对象可以更容易的拷贝，Dictionaries有一个copy方法：

newdict=olddict.copy()

### **13：有没有一个工具可以帮助查找python的bug和进行静态的代码分析？**

有，PyChecker是一个python代码的静态分析工具，它可以帮助查找python代码的bug, 会对代码的复杂度和格式提出警告 Pylint是另外一个工具可以进行coding standard检查。

### **14：如何在一个function里面设置一个全局的变量？**

解决方法是在function的开始插入一个global声明：

deff() globalx

### **15：用Python匹配HTML tag的时候，<.>和<.?>有什么区别？**

当重复匹配一个正则表达式时候， 例如<.\*>, 当程序执行匹配的时候，会返回最大的匹配值 例如：

importre s=‘<html><head><title>Title</title>’ print(re.match(’<.\*>’,s).group())

会返回一个匹配而不是 而

importre s=‘<html><head><title>Title</title>’ print(re.match(’<.\*?>’,s).group())

则会返回 <.>这种匹配称作贪心匹配 <.?>称作非贪心匹配

### **16：Python里面search()和match()的区别？**

match（）函数只检测RE是不是在string的开始位置匹配， search()会扫描整个string查找匹配, 也就是说match（）只有在0位置匹配成功的话才有返回，如果不是开始位置匹配成功的话，match()就返回none 例如：

print(re.match(’super’,’superstition’).span())

会返回(0, 5) 而

print(re.match(’super’,‘insuperable’))

则返回None search()会扫描整个字符串并返回第一个成功的匹配 例如：

print(re.search(’super’,’superstition’).span())

返回(0, 5)

print(re.search(’super’,‘insuperable’).span())

返回(2, 7)

### **17：如何用Python来进行查询和替换一个文本字符串？**

可以使用sub()方法来进行查询和替换，sub方法的格式为：sub(replacement, string[, count=0]) replacement是被替换成的文本 string是需要被替换的文本 count是一个可选参数，指最大被替换的数量 例子：

importre p=re.compile(’(blue|white|red)’) print(p.sub(’colour’,'bluesocksandredshoes’)) print(p.sub(’colour’,'bluesocksandredshoes’,count=1))

输出：

coloursocksandcolourshoes coloursocksandredshoes

subn()方法执行的效果跟sub()一样，不过它会返回一个二维数组，包括替换后的新的字符串和总共替换的数量 例如：

importre p=re.compile(’(blue|white|red)’) print(p.subn(’colour’,'bluesocksandredshoes’)) print(p.subn(’colour’,'bluesocksandredshoes’,count=1))

输出

(’coloursocksandcolourshoes’,2) (’coloursocksandredshoes’,1)

### **18：介绍一下except的用法和作用？**

Python的except用来捕获所有异常， 因为Python里面的每次错误都会抛出 一个异常，所以每个程序的错误都被当作一个运行时错误。 一下是使用except的一个例子：

try: foo=opne(”file”)#open被错写为opne except: sys.exit(”couldnotopenfile!”)

因为这个错误是由于open被拼写成opne而造成的，然后被except捕获，所以debug程序的时候很容易不知道出了什么问题 下面这个例子更好点：

try: foo=opne(”file”)#这时候except只捕获IOError exceptIOError: sys.exit(”couldnotopenfile”)

### **19：Python中pass语句的作用是什么？**

pass语句什么也不做，一般作为占位符或者创建占位程序，pass语句不会执行任何操作，比如：

whileFalse: pass

pass通常用来创建一个最简单的类：

classMyEmptyClass: pass

pass在软件设计阶段也经常用来作为TODO，提醒实现相应的实现，比如：

definitlog(\*args): pass#pleaseimplementthis

### **20：介绍一下Python下range()函数的用法？**

如果需要迭代一个数字序列的话，可以使用range()函数，range()函数可以生成等差级数。 如例：

foriinrange(5) print(i)

这段代码将输出0, 1, 2, 3, 4五个数字 range(10)会产生10个值， 也可以让range()从另外一个数字开始，或者定义一个不同的增量，甚至是负数增量 range(5, 10)从5到9的五个数字 range(0, 10, 3) 增量为三， 包括0,3,6,9四个数字 range(-10, -100, -30) 增量为-30， 包括-10, -40, -70 可以一起使用range()和len()来迭代一个索引序列 例如：

a=['Nina','Jim','Rainman','Hello'] foriinrange(len(a)): print(i,a[i])

### **21：有两个序列a,b，大小都为n,序列元素的值任意整形数，**

无序；要求：通过交换a,b中的元素，使[序列a元素的和]与[序列b元素的和]之间的差最小。 将两序列合并为一个序列，并排序，为序列Source

拿出最大元素Big，次大的元素Small

在余下的序列S[:-2]进行平分，得到序列max，min

将Small加到max序列，将Big加大min序列，重新计算新序列和，和大的为max，小的为min。

Python代码

defmean(sorted\_list): ifnotsorted\_list: return(([],[])) big=sorted\_list[-1] small=sorted\_list[-2] big\_list,small\_list=mean(sorted\_list[:-2]) big\_list.append(small) small\_list.append(big) big\_list\_sum=sum(big\_list) small\_list\_sum=sum(small\_list) ifbig\_list\_sum>small\_list\_sum: return((big\_list,small\_list)) else: return((small\_list,big\_list)) tests=[[1,2,3,4,5,6,700,800], [10001,10000,100,90,50,1], range(1,11), [12312,12311,232,210,30,29,3,2,1,1] ] forlintests: l.sort() print print“SourceList:”,l l1,l2=mean(l) print“ResultList:”,l1,l2 print“Distance:”,abs(sum(l1)-sum(l2)) print‘-\*’\*40

输出结果

SourceList:[1,2,3,4,5,6,700,800] ResultList:[1,4,5,800][2,3,6,700] Distance:99 -\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\* SourceList:[1,50,90,100,10000,10001] ResultList:[50,90,10000][1,100,10001] Distance:38 -\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\* SourceList:[1,2,3,4,5,6,7,8,9,10] ResultList:[2,3,6,7,10][1,4,5,8,9] Distance:1 -\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\* SourceList:[1,1,2,3,29,30,210,232,12311,12312] ResultList:[1,3,29,232,12311][1,2,30,210,12312] Distance:21 -\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*

**1.如何反向迭代一个序列**

#如果是一个list,最快的方法使用reverse tempList = [1,2,3,4] tempList.reverse() for x in tempList: print x

#如果不是list,需要手动重排 templist = (1,2,3,4) for i in range(len(templist)-1,-1,-1): print templist[i]

## **2.如何查询和替换一个文本中的字符串**

#最简单的方法使用replace() tempstr = "hello you hello python are you ok" print tempstr.replace("you","python")

#还可以使用正则,有个sub() tempstr = "hello you hello python are you ok" import re rex = r'(hello|Use)' print re.sub(rex,"Bye",tempstr)

## **3.使用python实现单例模式**

#方法一:可以使用\_\_new\_\_方法 #在\_\_new\_\_方法中把类实例绑定到类变量\_instance上，如果cls.\_instance为None表示该类还没有实例化过，实例化该类并返回。如果cls\_instance不为None表示该类已实例化，直接返回cls\_instance class SingleTon(object): def \_\_new\_\_(cls,\*args,\*\*kwargs): if not hasattr(cls,'\_instance'): cls.\_instance = object.\_\_new\_\_(cls,\*args,\*\*kwargs) return cls.\_instance class TestClass(SingleTon): a = 1

test1 = TestClass() test2 = TestClass() print test1.a,test2.a

test1.a=2 print test1.a,test2.a

print id(test1),id(test2)

#方法二:使用装饰器,建立过实例的就放到instances里面,下次建立的时候先检查里面有没有 def SingleTon(cls,\*args,\*\*kwargs): instances = {} print instances def \_singleton(): if cls not in instances: instances[cls] = cls(\*args,\*\*kwargs) print instances return instances[cls] return \_singleton

@SingleTon class LastClass(object): a = 1 test1 = LastClass() print test1.a test2 = LastClass() print test2.a

#方法三:使用\_\_metaclass\_\_(元类)

class SignalTon(type): def \_\_init\_\_(cls,name,bases,dict): super(SignalTon, cls).\_\_init\_\_(name,bases,dict) cls.\_instance = None

def \_\_call\_\_(cls, \*args, \*\*kwargs): if cls.\_instance is None: cls.\_instance = super(SignalTon,cls).\_\_call\_\_(\*args,\*\*kwargs) return cls.\_instance

class TestClass(object): \_\_metaclass\_\_ = SignalTon

test1 = TestClass() test2 = TestClass()

test1.a = 2 print test1.a,test2.a print id(test1),id(test2)

#方法四:共享属性 所谓单例就是所有的引用（实例，对象）拥有相同的属性和方法，同一个类的实例天生都会有相同的方法，那我们只需要保证同一个类所产生的实例都具有相同的属性。所有实例共享属性最简单直接的方法就是共享\_\_dict\_\_属性指向。

class SingleTon(object): \_state = {} def \_\_new\_\_(cls, \*args, \*\*kwargs): obj = object.\_\_new\_\_(cls,\*args,\*\*kwargs) obj.\_\_dict\_\_ = cls.\_state return obj

class TestClass(SingleTon): a = 1

test1 = TestClass() test2 = TestClass() print test1.a,test2.a test1.a = 2 print test1.a,test2.a print id(test1),id(test2) #方法五:使用同一个模版 #写在mysingleton.py中 class My\_Singleton(object): def foo(self): pass

my\_singleton = My\_Singleton()

#写在要使用这个实例的py文件里面,在不同的引用的地方都引用相同的实例,以此实现单例模式 from mysingleton import my\_singleton my\_singleton.foo()

## **4.重新实现str.strip()**

def rightStrip(tempStr,splitStr): endindex = tempStr.rfind(splitStr) while endindex != -1 and endindex == len(tempStr) - 1: tempStr = tempStr[:endindex] endindex = tempStr.rfind(splitStr) return tempStr

def leftStrip(tempStr,splitStr): startindex = tempStr.find(splitStr) while startindex == 0: tempStr = tempStr[startindex+1:] startindex = tempStr.find(splitStr) return tempStr

str = " H " print str print leftStrip(str,' ') print rightStrip(str,' ') #输出 H H H

## **5.super的原理**

#阅读下面的代码，它的输出结果是什么？ class A(object): def \_\_init\_\_(self): print "enter A" super(A, self).\_\_init\_\_() # new print "leave A"

class B(object): def \_\_init\_\_(self): print "enter B" super(B, self).\_\_init\_\_() # new print "leave B"

class C(A): def \_\_init\_\_(self): print "enter C" super(C, self).\_\_init\_\_() print "leave C"

class D(A): def \_\_init\_\_(self): print "enter D" super(D, self).\_\_init\_\_() print "leave D" class E(B, C): def \_\_init\_\_(self): print "enter E" super(E, self).\_\_init\_\_() # change print "leave E"

class F(E, D): def \_\_init\_\_(self): print "enter F" super(F, self).\_\_init\_\_() # change print "leave F"

#输出

enter F enter E enter B enter C enter D enter A leave A leave D leave C leave B leave E leave F

## **6.闭包**

常用的装饰器就是闭包的一种

def make\_adder(addend): def adder(addend): return addend+addend return adder

P1 = make\_adder(5) P2= make\_adder(4)

print p1(10) #输出15 print p2(10) #输出14

闭包（Closure）是词法闭包（Lexical Closure）的简称，是引用了自由变量的函数。这个被引用的自由变量将和这个函数一同存在，即使已经离开了创造它的环境也不例外

## **7.给列表中的字典排序**

list 对象 alist [{“name”:”a”,”age”:20},{“name”:”b”,”age”:30},{“name”:”c”,”age”:25}]按照 age 从大到小排序

alist = [{"name":"a","age":20},{"name":"b","age":30},{"name":"c","age":25}] alist.sort(key=lambda:x:-x.get("age")) print alist

## **8.合并两个列表排除重复元素**

用简洁的方法合并alist = [‘a’,’b’,’c’,’d’,’e’,’f’] blist = [‘x’,’y’,’z’,’e’,’f’]并且元素不能重复

alist = ['a','b','c','d','e','f'] blist = ['x','y','z','e','f'] def merge\_list(\*args): s = set() for i in args: s = s.union(i) print(s) return s

merge\_list(alist,blist)

## **9.打乱一个排好序的列表**

from random import shuffle alist = range(10) print(alist) shuffle(alist) print(alist)

## **10.简单的实现一个栈结构 stack**

class Stack(object): def \_\_init\_\_(self): self.value = []

def push(self,x): self.value.append(x)

def pop(self): self.value.pop()

stack = Stack()

stack.push(1) stack.push(2) stack.push(3) print(stack.value) stack.pop() print(stack.value)

## **11.输入一个日期,返回时一年中的哪一天**

from datetime import datetime def which\_day(year,month,day): return (datetime(year,month,day)-datetime(year,1,1)).days+1

print(which\_day(2017,1,15))

## **12.把字符串”k1:1|k2:2|k3:3”处理成 python 字典的形式:{k1:1,k2:2,k3:3}**

def string\_to\_dict(string): d = {} for kv in string.split("|"): k,v = kv.split(":") if v.isdigit(): v=int(v) d[k]=v return d

print(string\_to\_dict("k1:1|k2:2|k3:3"))

## **13.判断输入的值是否在矩阵之中(杨氏矩阵)**

在一个二维数组之中,每一行都按照从走到右递增的顺序排序,每一列到按照从上到下的顺序排序.请完成一个函数,输入这样的一个二维手术和一个整数,判断数组中是否含有该整数

#处理数组矩阵 arr = [[1,4,7,10,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]] def get\_num(num,data=None): while data: if num > data[0][-1]: del data[0] elif num<data[0][-1]: data = list(zip(\*data)) del data[-1] data = list(zip(\*data)) else: return True data.clear() return False print (get\_num(18,arr))

不处理数组矩阵 使用 step-wise 线性搜索

def getvalue(data,value): m = len(data)-1 n = len(data[0])-1 r = 0 c = n while c>=0 and r<=m: if value == data[r][c]: return True elif value>data[r][c]: r = r+1 else: c = c-1 return False

## **14.获取最大公约数(欧几里得算法)**

a= 25 b=15

def max\_common(a,b): while b: a,b=b,a%b return a

## **15.求两个数的最小公倍数(公式法)**

两个数的乘积等于这两个数的 最大公约数与最小公倍数的积

a=25 b=15 def min\_common(a,b): c= a\*b while b: a,b=b,a%b return c//a

**16.获取中位数**

如果总数个数是奇数，按从小到大的顺序，取中间的那个数；如果总数个数是偶数个的话，按从小到大的顺序，取中间那两个数的平均数。

#计算中位数 def mediannum(num): listnum = [num[i] for i in range(len(num))] listnum.sort() lnum = len(num) if lnum % 2 == 1: i = int((lnum + 1) / 2)-1 return listnum[i] else: i = int(lnum / 2)-1 return (listnum[i] + listnum[i + 1]) / 2

def medin(data): data.sort() half = len(data)//2 return (data[half]+data[~half])/2 l = [1,3,4,53,2,46,8,42,82] print (median(l))

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Python 是一种解释型，交互式，面向对象的高级编程语言。和别的一些使用标点符号的语言不同，Python使用了大量的英语单词作为关键字，因而具有很好的可读性。而且跟其他编程语言相比，它有更少的语法结构。

* Python 是一种解释性语言：这意味着你的程序在执行之前不需要编译，而是由解释器在运行时处理。这个特点跟PERL，PHP很像。
* Python是可交互的：这意味着你可以使用一个Python终端在写程序时和解释器直接交互。
* Python是面向对象的：它支持面向对象风格和将代码封装成类的各种技术。
* Python非常适合编程初级人员：Python非常容易入门而且应用领域也非常广泛，从简单的文字处理到网络浏览和游戏开发。

**1) 什么是Python？使用Python有什么好处？**

Python是一种编程语言，它有对象，模块，线程，异常处理和自动内存管理。它简洁，简单，方便，容易扩展，有许多自带的数据结构，而且它开源。

**2) 什么是PEP8？**

PEP8 是一个编程规范，一些关于如何让你的程序更具有可读性的建议。

**3) 什么是pickling和unpickling？**

Pickle模块读入任何Python对象，将它们转换成字符串，然后使用dump函数将其转储到一个文件中——这个过程叫做pickling。反之从存储的字符串文件中提取原始Python对象的过程叫做unpickling。

**4) Python 是如何被解释的？**

Python是一种解释性语言。Python的源代码可以直接运行。Python解释器会将源代码转换成中间语言，之后再翻译成机器码再执行。

**5) Python是怎样管理内存的？**

Python的内存管理是由私有heap空间管理的。所有的Python对象和数据结构都在一个私有heap中。程序员没有访问该heap的权限，只有解释器才能对它进行操作。

为Python的heap空间分配内存是由Python的内存管理模块进行的。其核心API提供一些访问该模块的方法供程序员使用。

Python有自带的垃圾回收系统，它回收并释放没有被使用的内存让它们能够被其他程序使用。

**6) 有哪些工具可以帮助debug或做静态分析？**

PyChecker 是一个静态分析工具，它报告源代码中的错误并且会报告错误类型和复杂度。Pylint是检验模块是否达到代码标准的另一个工具。

**7) 什么是Python装饰器？**

Python装饰器是Python中的特有变动，可以使修改函数变得更容易。

**8) 数组和元组之间的区别是什么？**

数组和元组之间的区别是数组内容是可以被修改的而元组内容是只读的。元组可以被哈希比如作为字典的关键字。

**9) 参数按值传递和引用传递是怎样实现的？**

Python中的一切都是类，所有的变量都是一个对象的引用。引用的值是由函数确定的，因此无法被改变。但是如果一个对象是可以被修改的你可以改动对象。

**10) 字典推导式和列表推导式是什么？**

他们是可以轻松创建字典和列表的语法结构。

**11) Python都有那些自带的数据结构？**

Python自带的数据结构分为可变的和不可变的。

可变的有：

* 数组
* 集合
* 字典

不可变的有：

* 字符串
* 元组
* 数

**12) 什么是Python的命名空间？**

在 Python 中，所有的名字都存在于一个空间中，它们在该空间中存在和被操作——这就是命名空间。它就好像一个盒子，每一个变量名字都对应装着一个对象。当查询变量的时候，会从该盒子里面寻找相应的对象。

**13) Python 中的lambda 是什么？**

这是一个常被用于代码中的单个表达式的匿名函数。

**14) 为什么 lambda 没有语句？**

匿名函数lambda没有语句的原因是它被用于在代码被执行的时候构建新的函数对象并且返回。

**15) Python中的pass是什么？**

Pass是一个在Python中不会被执行的语句。在复杂语句中，如果一个地方需要暂时被留白，它常常被用于占位符。

**16) Python中什么是遍历器？**

遍历器用于遍历一组元素，比如列表这样的容器。、

**17) Python中的unittest是什么？**

在Python中，unittest是Python中的单元测试框架。它支持共享搭建，自动测试，在测试中暂停代码，将不同测试迭代成一组等等功能。

**18) 在Python中什么是slicing？**

Slicing是一种在有序的对象类型中（数组，元组，字符串）节选某一段的语法。

**19）在Python中什么是生成器？**

生成器是实现迭代器的一种机制。它功能的实现依赖于yield表达式，除此之外它跟普通的函数没有两样。

**20) Python 中docstring什么？**

在 Python 中文档字符串被称为docstring，它被用于在Python中为函数，模块和类注释生成文档。

**21) 在Python中如何拷贝一个对象？**

如果要在Python中拷贝一个对象，大多时候你可以用copy.copy () 或者copy.deepcopy()。并不是所有的对象都可以被拷贝。

**22) Python中的负索引是什么？**

Python中的序列索引可以是正也可以是负。如果是正索引，0是序列中的第一个索引，1是第二个索引。如果是负索引，（-1）是最后一个索引而（-2）是倒数第二个索引。

**23) 如何将一个数字转换成一个字符串？**

你可以使用自带函数str()将一个数字转换为字符串。如果你想要八进制或者十六进制数，可以用oct() 或 hex()。

**24) Xrange和range的区别是什么？**

Xrange返回一个xrange对象，而range返回一个数组。不管那个范围多大，Xrange使用同样的内存。

**25) Python中的模块和包是什么？**

在Python中，模块是搭建程序的一种方式。每一个Python代码文件都是一个模块，并可以引用其他的模块比如对象和属性。

一个包含许多Python代码的文件夹是一个包。一个包可以包含模块和子文件夹。