1.简答题

1. 如何交换两个变量值

方法1,：

a, b = b, a

方法2：使用加法操作：

a+=b;

b=a-b;

a -= b;

方法3：使用异或操作：

a^=b;

b^=a;

a^=b;

2. 去掉list中的重复元素，old\_list = [1,1,1,3,4]

方法1,使用set函数，set(list)

old\_list = [1,1,1,3,4]

new\_list = list(set(old\_list))

方法2，使用字典函数，

>>> old\_list = [1,1,1,3,4]

>>> a={}

>>>a=a.fromkeys(old\_list)

>>>b=list(a.keys())

>>>b

方法3, 编程用sort进行排序，然后从最后一个元素开始判断

old\_list = [1,1,1,3,4]

old\_list.sort()

last = old\_list [-1]

for i in range(len(old\_list)-2,-1,-1):

if last== old\_list [i]:

del old\_list [i]

else:last= old\_list [i]

print(old\_list)

3. 翻转一个字符串，s = ‘ abcde ’

>>> s = 'abcdef'

>>> s[::-1]

1. 简单的步长为-1, 即字符串的翻转(常用)

# -\*- coding: utf-8 -\*-

string = 'abcdef'

def string\_reverse1(string):

return string[::-1]

print(string\_reverse1(string))

2. 交换前后字母的位置;

# -\*- coding: utf-8 -\*-

string = 'abcdef'

def string\_reverse2(string):

t = list(string)

l = len(t)

for i,j in zip(range(l-1, 0, -1), range(l//2)):

t[i], t[j] = t[j], t[i]

return "".join(t)

print(string\_reverse2(string))

3. 递归的方式, 每次输出一个字符;

# -\*- coding: utf-8 -\*-

string = 'abcdef'

def string\_reverse3(string):

if len(string) <= 1:

return string

return string\_reverse3(string[1:]) + string[0]

print(string\_reverse3(string))

4. 双端队列, 使用extendleft()函数;

# -\*- coding: utf-8 -\*-

string = 'abcdef'

from collections import deque

def string\_reverse4(string):

d = deque()

d.extendleft(string)

return ''.join(d)

print(string\_reverse4(string))

5. 使用for循环, 从左至右输出;

# -\*- coding: utf-8 -\*-

string = 'abcdef'

def string\_reverse5(string):

#return ''.join(string[len(string) - i] for i in range(1, len(string)+1))

return ''.join(string[i] for i in range(len(string)-1, -1, -1))

print(string\_reverse5(string))

4. 用两个元素之间有对应关系的list构造一个dict

names = [‘ jianpx’, ’yue’]

ages = [23, 40]

names = [‘jianpx’,’yue’]

ages = [23,40]

m = dict(zip(names,ages))

5. 将数量较多的字符串相连，如何效率较高，为什么

fruits = [‘apple’, ’banana’]

方式1.使用加号（+）进行连接，字符串拼接直接进行相加就可以，变量直接相加，不是变量就要用引号引起来，数字是要转换为字符串才能够进行相加的，不能把数字直接相加。 如fruits[0]+fruit[1]

方式2.使用%进行连接，把变量统一进行添加，这样避免了使用加号的情况，能够让代码更加简短

方式3.使用单引号或者双引号的方式连接，首先进行定义，把我们需要的格式进行定义

以上三种方法应视情况而定其效率问题

6. 调试python代码的方法有哪些？

调试工具：

1.IDE调试，比如:IDLE, Eclipse+Pydev都可以设置断点调试。

2.pdb模块也可以做调试。

3.PyChecker是一个python代码的静态分析工具，它可以帮助查python代码的bug, 会对代码的复杂度和格式提出警告

4.Pylint是另外一个工具可以进行coding standard检查。

调试方法：

1. 直接print打印输出

2. Assert（断言），表达式n != 0应该是True，否则，后面的代码就会出错。

3. 把print替换为logging是第3种方式，和assert比，logging不会抛出错误

4. 启动Python的调试器pdb，让程序以单步方式运行，可以随时查看运行状态

5. pdb.set\_trace()这个方法也是用pdb，但是不需要单步执行，我们只需要import pdb，然后，在可能出错的地方放一个pdb.set\_trace()

7. 什么是GIL？

GIL(Global Interpreter Lock)全局解释器锁，简单地说就是:

每一个interpreter进程,只能同时仅有一个线程来执行, 获得相关的锁, 存取相关的资源.

那么很容易就会发现,如果一个interpreter进程只能有一个线程来执行,

多线程的并发则成为不可能, 即使这几个线程之间不存在资源的竞争.

从理论上讲,我们要尽可能地使程序更加并行, 能够充分利用多核的功能.

GIL无疑就是一把全局排他锁。Python代码的执行由Python 虚拟机(也叫解释器主循环，CPython版本)来控制，Python 在设计之初就考虑到要在解释器的主循环中，同时只有一个线程在执行，即在任意时刻，只有一个线程在解释器中运行。对Python 虚拟机的访问由全局解释器锁（GIL）来控制，正是这个锁能保证同一时刻只有一个线程在运行。

8. 是否遇到过python的模块间循环引用的问题，如何避免它？

这是代码结构设计的问题，模块依赖和类依赖

如果老是觉得碰到循环引用，很可能是模块的分界线划错地方了。可能是把应该在一起的东西硬拆开了，可能是某些职责放错地方了，可能是应该抽象的东西没抽象

总之微观代码规范可能并不能帮到太多，重要的是更宏观的划分模块的经验技巧，推荐uml，脑图，白板等等图形化的工具先梳理清楚整个系统的总体结构和职责分工

采取办法，从设计模式上来规避这个问题，比如:

1. 使用 “\_\_all\_\_” 白名单开放接口

2. 尽量避免 import

9. Python里面如何拷贝一个对象？以及各种拷贝方式的区别

方式1.赋值（=），就是创建了对象的一个新的引用，修改其中任意一个变量都会影响到另一个。

方式2.浅拷贝：创建一个新的对象，但它包含的是对原始对象中包含项的引用（如果用引用的方式修改其中一个对象，另外一个也会修改改变）{1,完全切片方法；2，工厂函数，如list()；3，copy模块的copy()函数}

方式3.深拷贝：创建一个新的对象，并且递归的复制它所包含的对象（修改其中一个，另外一个不会改变）{copy模块的deep.deepcopy()函数}

10. 如何用Python的re模块来进行查询和替换一个文本字符串（举例说明）？

可以使用re模块中的sub()函数或者subn()函数来进行查询和替换，

格式：sub(replacement,string[,count=0]) (replacement 是被替换成的文本，string是被替换成的文本，count是一个可选参数，指最大被替换的数量)

11. 单引号，双引号，三引号的区别

单引号和双引号是等效的，如果要换行，需要符号(\),三引号则可以直接换行，并且可以包含注释

如果要表示Let’s go 这个字符串

单引号：s4 = ‘Let\’s go’

双引号：s5 = “Let’s go”

s6 = ‘I realylike“python”!’

这就是单引号和双引号都可以表示字符串的原因了

12. 什么是lambda函数？他有什么好处？

Python有两种函数,一种是def定义，一种是lambda函数()

当程序代码很短，且该函数只使用一次，为了程序的简洁，及节省变量内存占用空间，引入了匿名函数这个概念：

lambda 表达式，通常是在需要一个函数，但是又不想费神去命名一个函数的场合下使用，也就是指匿名函数

lambda函数：首要用途是指点短小的回调函数

lambda [arguments]:expression

>>> a=lambdax,y:x+y

>>>a(3,11)

2. 编程题

1. Python里如何反序的迭代一个序列

如果是一个list, 方案是：

list.reverse()

try:

for x in list:

“do something with x”

finally:

list.reverse()

如果不是list, 最通用但是稍慢的解决方案是：

for i in range(len(sequence)-1, -1, -1):

x = sequence[i]

2. 如何用Python来进行查询和替换一个文本字符串

可以使用re模块中的sub()函数或者subn()函数来进行查询和替换，

格式：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）

>>> import re

>>>p=re.compile(‘blue|white|red’)

>>>print(p.sub(‘colour’,'blue socks and red shoes’))

colour socks and colourshoes

>>>print(p.sub(‘colour’,'blue socks and red shoes’,count=1))

colour socks and redshoes

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

3. 有二维的list对象alist，假定其中的所有元素都具有相同的长度，写一段程序根据元素的的第二个元素排序

lists = [['ab','cd'],['ef','gh']]

def sort\_lists(lists, sord, idx):

if sord == 'desc':

lists.sort(key=lambda x:x[idx], reverse=True)

else:

lists.sort(key=lambda x:x[idx])

return lists

sort\_lists(lists,'desc',1)

print(lists)

4. 写一个装饰器decorator,décor(),只运行foo()方法前添加打印“do sth before foo()..”，在运行foo()方法后添加打印“do sth alter foo()..”

def decor(fn):

def foo(): # 此时foo为闭包,因为foo调用了fn参数

print("运行方法前：")

fn()

print("运行方法后：")

return foo

@decor

def decorator():

print("do sth before foo()..")

def decorator2():

print("do sth alter foo()..")

decorator()

decorator2()

5. 设计一个单链表，实现逆置方法

逆置方法一般分为两种：递归反转单链表，循环反转单链表