Python面试题

代码中要修改不可变数据会出现什么问题？抛出什么异常

代码不会正常运行，抛出 TypeError 异常

单引号、双引号、三引号的区别

这几个符号都是可以表示字符串的，如果是表示一行，则用单引号或者双引号表示，它们的区别是 如果内容里有"符号，并且你用双引号表示的话则需要转义字符，而单引号则不需要三单引号和三双引号也是表示字符串， 并且可以表示多行，遵循的是所见即所得的原则。 另外，三双引号和三单引号可以作为多行注释来用，单行注释用#号。

a=1,b=2,不用中间变量交换 a 和 b 的值

方法一： a = a+b b = a-b a = a-b

方法二： a = a^b b =b^a a = a^b

方法三： a,b = b,a

print 调用 Python 中底层的什么方法

print 方法默认调用 sys.stdout.write 方法，即往控制台打印字符串。

Python中append和extend的区别

list.append(object) 向列表中添加一个对象object
list.extend(sequence) 把一个序列seq的内容添加到列表中

```
1
2
3
4
5
```

使用append的时候，是将new_media看作一个对象，整体打包添加到music_media对象中。

```
1
2
3
4
5
```

使用extend的时候，是将new_media看作一个序列，将这个序列和music_media序列合并，并放在其后面。

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
```

结果为：

1 1 1 #继承自父类的类属性 x，所以都一样，指向同一块内存地址。

1 2 1 #更改 Child1，Child1 的 x 指向了新的内存地址。

3 2 3 #更改 Parent，Parent 的 x 指向了新的内存地址

简述你对input()函数的理解

在Python3中，input()获取用户输入，不论用户输入的是什么，获取到的都是字符串类型的。

在Python2中有 raw_input()和input(), raw_input()和Python3中的 input()作用是一样的，

input()输入的是什么数据类型的，获取到的就是什么数据类型的。

阅读下面的代码，写出 A0，A1 至 An 的最终值

```
A0 = dict(zip(('a', 'b', 'c', 'd', 'e'), (1, 2, 3, 4, 5)))
A1 = range(10)
A2 = [i for i in A1 if i in A0] A3 = [A0[s] for s in A0]
A4 = [i for i in A1 if i in A3] A5 = {i:i*i for i in A1}
A6 = [[i, i*i] for i in A1]
答:
A0 = {'a': 1, 'c': 3, 'b': 2, 'e': 5, 'd': 4} A1 = [0, 1, 2, 3, 4, 5, 6, 7
A2 = []
A3 = [1, 3, 2, 5, 4] A4 = [1, 2, 3, 4, 5]
A5 = {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81} A6
49], [8, 64] [9, 81]]
```

range 和 xrange 的区别

两者用法相同，不同的是 range 返回的结果是一个列表，而 xrange 的结果是一个生成器，前者是直接开辟一块内存空间来保存列表，后者是边循环边使用，只有使用时才会开辟内存空间，所以当列表很长时，使用xrange性能要比 range好

考虑以下 Python 代码，如果运行结束，命令行中的运行结果是什么

```
l = []
for i in xrange(10):
l.append({‘num’:i}) print l 考虑以下代码，运行结束后的结果是什么 l = []
a = {'num':0}
for i in xrange(10):
a['num']= i l.append(a)
print l 以上两段代码的运行结果是否相同，如果不相同，原因是什么？ 上方代码的结果：
[{‘num’:0}, {‘num’:1}, {‘num’:2}, {‘num’:3}, {‘num’:4}, {‘num’:5}, {‘num’:6},
{‘num’:7}, {‘num’:8}, {‘num’:9}] 下方代码结果： [{‘num’:9}, {‘num’:9}, {‘num’:9}, {
{‘num’:9}, {‘num’:9}, {‘num’:9}]
原因是：字典是可变对象，在下方的 l.append(a)的操作中是把字典 a 的引用传到列表 l 中，当后 续
以下 Python 程序的输出
for i in range(5, 0, -1):
print(i)
答：5 4 3 2 1
```

4G 内存怎么读取一个5G的数据

方法一： 可以通过生成器，分多次读取，每次读取数量相对少的数据（比如 500MB）进行处理，处理结束后在读取后面的 500MB 的数据。

方法二： 可以通过 linux命令 split 切割成小文件，然后再对数据进行处理，此方法效率比较高。可以按照行数切割，可以按照文件大小切割。

现在考虑有一个 jsonline 格式的文件 file.txt 大小约为10K，之前处理文件的代码如下所示

```
def get_lines():
l = []
with open(‘file.txt’, ‘rb’) as f:
for eachline in f:
l.append(eachline)
return l
if name
== ‘ main ’:
for e in get_lines():
process(e) #处理每一行数据
```

现在要处理一个大小为 10G 的文件，但是内存只有4G，如果在只修改 get_lines 函数而其他代 码保持不变的情况下，应该如何实现？需要考虑的问题都有哪些

read、readline 和 readlines 的区别?

read:读取整个文件

readline：读取下一行，使用生成器方法

readlines：读取整个文件到一个迭代器以供我们遍历

补充缺失的代码

```
def print_directory_contents(sPath): """
这个函数接收文件夹的名称作为输入参数
返回该文件夹中文件的路径 以及其包含文件夹中文件的路径 """
# 补充代码
------------代码如下-------------------- import os
for sChild in os.listdir(sPath):
sChildPath = os.path.join(sPath, sChild)
if os.path.isdir(sChildPath):
print_directory_contents(sChildPath)
else:
print(sChildPath)
```

在except中return后还会不会执行finally中的代码？怎么抛出自定义异常?

会继续处理finally中的代码；用raise方法可以抛出自定义异常

介绍一下except的作用和用法

xcept:#捕获所有异常

except:<异常名>:#捕获指定异常 except:<异常名1，异常名2):捕获异常1或者异常2 except:<异常名>,<数据>:捕获指定异常及其附加的数据 except:<异常名1,异常名2>:<数据>:捕获异常名1或者异常名2,及附加的数据

常用的Python标准库都有哪些

os操作系统，time时间，random随机，pymysql连接数据库，threading线程，multiprocessing 进程，queue队列 第三方库：django和flask也是第三方库，requests，virtualenv，selenium，scrapy，xadmin， celery，re，hashlib，md5。

赋值、浅拷贝和深拷贝的区别

赋值

在 Python 中，对象的赋值就是简单的对象引用，这点和 C++不同，如下所示

a=[1,2,"hello",['python',‘C++’]

b = a

在上述情况下,a和b是一样的,他们指向同一片内存,b不过是a的别名,a是引用

我们可以使用 b is a 去判断，返回 True，表明他们地址相同，内容相同，也可以使用 id()函数来查 看两个列表的地址是否相同。 赋值操作(包括对象作为参数、返回值)不会开辟新的内存空间，它只是复制了对象的引用。也就是 说除了 b 这个名字之外，没有其他的内存开销。修改了 a，也就影响了 b，同理，修改了 b，也就影响 了 a。

浅拷贝(shallow copy)

浅拷贝会创建新对象，其内容非原对象本身的引用，而是原对象内第一层对象的引用。 浅拷贝有三种形式:切片操作、工厂函数、copy 模块中的 copy 函数。

比如上述的列表 a；

切片操作：b = a[:] 或者 b = [x for x in a]； 工厂函数：b = list(a)；

copy 函数：b = copy.copy(a)；

浅拷贝产生的列表 b 不再是列表 a 了，使用 is 判断可以发现他们不是同一个对象，使用 id 查看，他 们也不指向同一片内存空间。但是当我们使用 id(x) for x in a 和 id(x) for x in b 来查看 a 和 b 中元 素的地址时，可以看到二者包含的元素的地址是相同的。

在这种情况下，列表 a 和 b 是不同的对象，修改列表 b 理论上不会影响到列表 a。 但是要注意的是，浅拷贝之所以称之为浅拷贝，是它仅仅只拷贝了一层，在列表 a 中有一个嵌套的 list， 如果我们修改了它，情况就不一样了。

比如：a[3].append('java')。查看列表 b，会发现列表 b 也发生了变化，这是因为，我们修改了嵌 套的 list，修改外层元素，会修改它的引用，让它们指向别的位置，修改嵌套列表中的元素，列表的地址并未 发生变化，指向的都是用一个位置。

深拷贝(deep copy)

深拷贝只有一种形式，copy 模块中的 deepcopy()函数。 深拷贝和浅拷贝对应，深拷贝拷贝了对象的所有元素，包括多层嵌套的元素。因此，它的时间和空 间开销要高。

同样的对列表 a，如果使用 b = copy.deepcopy(a)，再修改列表 b 将不会影响到列表 a，即使嵌 套的 列表具有更深的层次，也不会产生任何影响，因为深拷贝拷贝出来的对象根本就是一个全新的对象， 不再与原 来的对象有任何的关联。

拷贝的注意点？ 对于非容器类型，如数字、字符，以及其他的“原子”类型，没有拷贝一说，产生的都是原对象的 引用。 如果元组变量值包含原子类型对象，即使采用了深拷贝，也只能得到浅拷贝。

init 和 new 的区别

init 在对象创建后，对对象进行初始化。

new 是在对象创建之前创建一个对象，并将该对象返回给 init。

Python 里面如何生成随机数

在 Python 中用于生成随机数的模块是 random，在使用前需要 import. 如下例子可以酌情列 举：

random.random()：生成一个 0-1 之间的随机浮点数； random.uniform(a, b)：生成[a,b]之间的浮点数； random.randint(a, b)：生成[a,b]之间的整数；

random.randrange(a, b, step)：在指定的集合[a,b)中，以 step 为基数随机取一个数；

random.choice(sequence)：从特定序列中随机取一个元素，这里的序列可以是字符串，列表， 元组等

输入某年某月某日，判断这一天是这一年的第几天

```
import datetime def dayofyear():
year = input("请输入年份: ")
month = input("请输入月份: ")
day = input("请输入天: ")
date1 = datetime.date(year=int(year), month=int(month), day=int(day))
date2 = datetime.date(year=int(year), month=1, day=1)
return (date1 - date2 + 1).days
打乱一个排好序的list对象alist
import random
random.shuffle(alist)
```

说明一下 os.path 和 sys.path 分别代表什么

os.path 主要是用于对系统路径文件的操作

sys.path 主要是对 Python 解释器的系统环境参数的操作（动态的改变 Python 解释器搜索路径）

Python 中的 os 模块常见方法

os.remove()删除文件

os.rename()重命名文件 os.walk()生成目录树下的所有文件名 os.chdir()改变目录 os.mkdir/makedirs 创建目录/多层目录

os.rmdir/removedirs 删除目录/多层目录 os.listdir()列出指定目录的文件 os.getcwd()取得当前工作目录 os.chmod()改变目录权限 os.path.basename()去掉目录路径，返回文件名 os.path.dirname()去掉文件名，返回目录路径 os.path.join()将分离的各部分组合成一个路径名

os.path.split()返回（dirname(),basename())元组 os.path.splitext()(返回 filename,extension)元组

os.path.getatime\ctime\mtime 分别返回最近访问、创建、修改时间 os.path.getsize()返回文件大小

os.path.exists()是否存在 os.path.isabs()是否为绝对路径 os.path.isdir()是否为目录 os.path.isfile()是否为文件

Python 的 sys 模块常用方法

sys.argv 命令行参数 List，第一个元素是程序本身路径 sys.modules.keys() 返回所有已经导入的模块列表 sys.exc_info() 获取当前正在处理的异常类,exc_type、exc_value、exc_traceback 当前 处理的异常详细信息sys.exit(n) 退出程序，正常退出时 exit(0) sys.hexversion 获取 Python 解释程序的版本值，16 进制格式如：0x020403F0 sys.version 获取 Python 解释程序的版本信息 sys.maxint 最大的 Int 值 sys.maxunicode 最大的 Unicode 值 sys.modules 返回系统导入的模 块字段，key 是模块名，value 是模块 sys.path 返回模块的搜索路径，初始化时使用 PYTHONPATH 环 境变量的值 sys.platform 返回操作系统平台名称sys.stdout 标准输出 sys.stdin 标准输入 sys.stderr 错误输出 sys.exc_clear() 用来清除当前线程所出现的当前的或最近的错误信息 sys.exec_prefix 返回平台独立的 python 文件安装的位置 sys.byteorder 本地字节规则的指示器， big-endian 平台的值是'big',little-endian 平台的值是 'little' sys.copyright 记录
python 版权相关的东西 sys.api_version 解释器的 C 的 API 版本 sys.version_info 元组则提 供一个更简单的方法来使你的程序具备 Python 版本要求功能

Python 中 list、tuple、dict、set 有什么区别，主要应用在什么样的场景？并用 for 语句遍历

区别：
1、list、tuple 是有序列表；dict、set 是无序列表；
2、list 元素可变、tuple 元素不可变；
3、dict 和 set 的 key 值不可变，唯一性；
4、set 只有 key 没有 value；
5、set 的用途：去重、并集、交集等；
6、list、tuple：+、*、索引、切片、检查成员等；
7、dict 查询效率高，但是消耗内存多；list、tuple 查询效率低、但是消耗内存少 应用场景：
list,：简单的数据集合,可以使用索引; tuple：把一些数据当做一个整体去使用,不能修改; dict：使用键值和值进行关联的数据; set：数据只出现一次,只关心数据是否出现，不关心其位置; 列表遍历：

```
a_list = [1, 2, 3, 4, 5]
for num in a_list:
print(num,end=' ')
元组遍历:
a_turple = (1, 2, 3, 4, 5)
for num in a_turple:
print(num,end=" ")
遍历字典:
dict = {“name”:”xiaoming”,”sex”:”man”}
for key in dict.key():
for value in dict.value() for item in dict.items() print key
print value print item
Set 遍历:
s = set(['Adam', 'Lisa', 'Bart'])
for name in s:
print name
Lisa Adam Bart
```

Python 中静态函数、类函数、成员函数的区别？并写一个示例

定义：

静态函数(@staticmethod)：即静态方法,主要处理与这个类的逻辑关联, 如验证数据； 类函数(@classmethod)：即类方法, 更关注于从类中调用方法, 而不是在实例中调用方法, 如构造 重载；

成员函数:：实例的方法, 只能通过实例进行调用。

用 Python 语言写一个函数，输入一个字符串，返回倒序结果

```
def test ()
strA = raw_input("请输入需要翻转的字符串: ")
order = []
for i in strA:
order.append(i)
order.reverse() #将列表反转
print ''.join(order) #将 list 转换成字符串
test()
```

unittest 是什么

在 Python 中，unittest 是 Python 中的单元测试框架。它拥有支持共享搭建、自动测试、在测试中暂停代码、将不同测试迭代成一组，等的功能

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。

Python 是强语言类型还是弱语言类型

Python 是强类型的动态脚本语言。 强类型：不允许不同类型相加。 动态：不使用显示数据类型声明，且确定一个变量的类型是在第一次给它赋值的时候。 脚本语言：一般也是解释型语言，运行代码只需要一个解释器，不需要编译。

谈一下什么是解释性语言，什么是编译性语言

计算机不能直接理解高级语言，只能直接理解机器语言，所以必须要把高级语言翻译成机器语言， 计算机才能执行高级语言编写的程序。 解释性语言在运行程序的时候才会进行翻译。 编译型语言写的程序在执行之前， 需要一个专门的编译过程，把程序编译成机器语言（可执行文件）。

Python 中有日志吗?怎么使用

有日志Python 自带 logging 模块，调用 logging.basicConfig()方法，配置需要的日志等级和相应的参数， Python 解释器会按照配置的参数生成相应的日志

Python 是如何进行类型转换的

内建函数封装了各种转换函数，可以使用目标类型关键字强制类型转换，进制之间的转换可以用 int(‘str’，base=’n’)将特定进制的字符串转换为十进制，再用相应的进制转换函数将十进制转换 为目标 进制。
可以使用内置函数直接转换的有： list----->tuple tuple(list) tuple----->list list(tuple)

Python 主要的内置数据类型有哪些

Python 主要的内置数据类型有：str，int，float，tuple，list，dict，set。

print(dir('a'))输出的是什么

会打印出字符型的所有的内置方法
1. ['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
'__getslice__', '__gt__', '__hash__', '__init__', '__le__', '__len__', '__lt__',
'__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', '_formatter_field_name_split', '_formatter_parser',
'capitalize', 'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs',
'find', 'format', 'index', 'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace',
'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition', 'replace',
'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper',
'zfill']

Python2 与 Python3 的区别

核心类差异

Python3 对 Unicode 字符的原生支持。

Python2 中使用 ASCII 码作为默认编码方式导致 string 有两种类型 str 和 unicode，Python3 只 支持 unicode 的 string。Python2 和 Python3 字节和字符对应关系为：

Python3 采用的是绝对路径的方式进行 import

Python2 中相对路径的 import 会导致标准库导入变得困难（想象一下，同一目录下有 file.py，如 何同时导入这个文件和标准库 file）。Python3 中这一点将被修改，如果还需要导入同一目录的文件必须使 用绝对路径，否则只能使用相关导入的方式来进行导入 Python2中存在老式类和新式类的区别，Python3统一采用新式类。新式类声明要求继承object， 必须用新式类应用多重继承

Python3 使用更加严格的缩进。Python2 的缩进机制中，1 个 tab 和 8 个 space 是等价的，所 以在缩进中可以同时允许 tab 和 space 在代码中共存。这种等价机制会导致部分 IDE 使用存在问题。Python3 中 1 个 tab 只能找另外一个 tab 替代，因此 tab 和 space 共存会导致报错：TabError: inconsistent use of tabs and spaces in indentation

废弃类差异

print 语句被 Python3 废弃，统一使用 print 函数

exec 语句被 python3 废弃，统一使用 exec 函数

execfile 语句被 Python3 废弃，推荐使用 exec(open("./filename").read())

不相等操作符"<>"被 Python3 废弃，统一使用"!=" long 整数类型被 Python3 废弃，统一使用 int

xrange函数被 Python3 废弃，统一使用 range，Python3 中 range 的机制也进行修改并提高了大数据 集生成效率

Python3 中这些方法再不再返回 list 对象：dictionary 关联的 keys()、values()、items()，

zip()， map()，filter()，但是可以通过 list 强行转换：

```
mydict={"a":1,"b":2,"c":3}
mydict.keys() #<built-in method keys of dict object at 0x000000000040B4C8>
list(mydict.keys()) #['a', 'c', 'b']
```

迭代器 iterator 的 next()函数被 Python3 废弃，统一使用 next(iterator)

raw_input 函数被 Python3 废弃，统一使用 input 函数 字典变量的 has_key 函数被 Python 废弃，统一使用 in 关键词

file 函数被 Python3 废弃，统一使用 open 来处理文件，可以通过 io.IOBase 检查文件类型

apply 函数被 Python3 废弃

异常 StandardError 被 Python3 废弃，统一使用 Exception

修改类差异 浮点数除法操作符“/”和“//”的区别 “ / ”：

Python2：若为两个整形数进行运算，结果为整形，但若两个数中有一个为浮点数，则结果为 浮点数；

Python3:为真除法，运算结果不再根据参加运算的数的类型。

“//”： Python2：返回小于除法运算结果的最大整数；从类型上讲，与"/"运算符返回类型逻辑一致。

Python3：和 Python2 运算结果一样。

异常抛出和捕捉机制区别

Python2

raise IOError, "file error" #抛出异常

except NameError, err: #捕捉异常

Python3

raise IOError("file error") #抛出异常

except NameError as err: #捕捉异常

for 循环中变量值区别

Python2，for 循环会修改外部相同名称变量的值

i = 1

print （'comprehension: ', [i for i in range(5)])

print （'after: i =', i） #i=4

Python3，for 循环不会修改外部相同名称变量的值

i = 1

print （'comprehension: ', [i for i in range(5)])

print （'after: i =', i） #i=1 round 函数返回值区别

Python2，round 函数返回 float 类型值 isinstance(round(15.5),int) #True Python3，round 函数返回 int 类型值 isinstance(round(15.5),float) #True

比较操作符区别

Python2 中任意两个对象都可以比较

11 < 'test' #True

Python3 中只有同一数据类型的对象可以比较

11 < 'test' # TypeError: unorderable types: int() < str()

第三方工具包差异

我们在 pip 官方下载源 pypi 搜索 Python2.7 和 Python3.5 的第三方工具包数可以发现， Python2.7 版本对应的第三方工具类目数量是 28523,Python3.5 版本的数量是 12457，这两个版本在 第三方工具 包支持数量差距相当大。 我们从数据分析的应用角度列举了常见实用的第三方工具包（如下表），并分析这些工具包在 Python2.7 和 Python3.5 的支持情工具安装问题windows 环境 Python2 无法安装 mysqlclient。Python3 无法安装 MySQL-python、flup、functools32、 Gooey、Pywin32、 webencodings。 matplotlib 在 python3 环境中安装 报错：The following required packages can not be built:freetype, png。需要手动下载安 装源码包安装解决。 scipy 在 Python3 环境中安装报错， numpy.distutils.system_info.NotFoundError，需要自己手 工下载对应的安装包，依赖 numpy,pandas 必须严格根据 python 版本、操作系统、64 位与否。运行 matplotlib 后发现基础包 numpy+mkl 安装失败，需要自己下载，国内暂无下载源 centos 环境下 Python2 无法安装 mysql- python 和 mysqlclient 包，报错：EnvironmentError: mysql_config not found，解决方案是安装 mysql-devel 包解决。使用 matplotlib 报错：no module named _tkinter， 安装 Tkinter、tk-devel、tc-devel 解决。 pywin32 也无法在 centos 环境下安装

关于 Python 程序的运行方面，有什么手段能提升性能

1、使用多进程，充分利用机器的多核性能 2、对于性能影响较大的部分代码，可以使用 C 或 C++编写 3、对于 IO 阻塞造成的性能影响，可以使用 IO 多路复用来解决 4、尽量使用 Python 的内建函数 5、尽量使用局部变量

Python 中的作用域

Python 中，一个变量的作用域总是由在代码中被赋值的地方所决定。当 Python 遇到一个变量的话 它会按照这的顺序进行搜索： 本地作用域(Local)--->当前作用域被嵌入的本地作用域(Enclosing locals)--->全局/模块作用域 (Global)--->内置作用域(Built-in)。

什么是 Python

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，可以加入其他语 言的对比。 Python 是一种解释型语言，Python 在代码运行之前不需要解释。 Python 是动态类型语言，在声明变量 时，不需要说明变量的类型。Python 适合面向对象的编程，因为它支持通过组合与继承的方式定义类。在 Python 语言中，函数是第一类对象。 Python 代码编写快，但是运行速度比编译型语言通常要慢。Python 用途广泛，常被用走"胶水语言"，可帮助其他语言和组件改善运行状况。使用 Python，程序员可以专注于算 法和数据结构的设计，而不用处理底层的细节

什么是Python 自省

Python 自省是 Python 具有的一种能力，使程序员面向对象的语言所写的程序在运行时,能够获得对象的类 Python 型。Python 是一种解释型语言，为程序员提供了极大的灵活性和控制力。

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。

你所遵循的代码规范是什么？请举例说明其要求

PEP8 规范。变量 常量：大写加下划线 USER_CONSTANT。
私有变量：小写和一个前导下划线_private_value。 Python 中不存在私有变量一说，若是遇到需要保护的变量，使用小写和一个前导下 划线。但这只是 程序员之间的一个约定，用于警告说明这是一个私有变量，外部类不要去访问它。但实际上， 外部类还 是可以访问到这个变量。
内置变量：小写，两个前导下划线和两个后置下划线 class 两个前导下划线会导致变量在解释期间被更名。这是为了避免内置变量和其他变量产生冲突。用户 定义的变量 要严格避免这种风格。以免导致混乱。
函数和方法 总体而言应该使用，小写和下划线。但有些比较老的 库使用的是混合大小写，即首单词小写，之后 每个单词第一个字母大写，其余小写。但现在，小写和下划线已 成为规范。 私有方法：小写和一个前导下划线 这里和私有变量一样，并不是真正的私有访问权限。 同时也应该注意一般函数不要使用两个前导下 划线(当遇到两个前导下划线时，Python 的名称改编特性将发 挥作用)。 特殊方法： 小写和两个前导下划线，两个后置下划线 这种风格只应用于特殊函数，比如操作 符重载等。
函数参数：小写和下划线，缺省值等号两边无空格 类 类总是使用驼峰格式命名，即所有 单词首字母大写其余字母小写。类名应该简明，精确，并足以从 中理解类所完成的工作。常见的一个方法是使 用表示其类型或者特性的后缀，例如: SQLEngine，MimeTypes 对于基类而言，可以使用一个 Base 或者 Abstract 前缀 BaseCookie， AbstractGroup 模块和包 除特殊模块init之外，模块名称都使用不带下划线的小写字母。 若是它们实现一个协议，那么通常使用 lib 为后缀，例如: import smtplib import os import sys关于参数 不要用断言来实现静态类型检测。断言可以用于检查参 数，但不应仅仅是进行静态类型检测。 Python 是动态类型语言，静态类型检测违背了其设计思想。断言应 该用于避免函数不被毫无意义的调用。不要滥用 *args 和 **kwargs。*args 和 **kwargs 参数可能会 破坏函数的健壮性。它们使签 名变得模糊，而且代码常常开始在不应该的地方构建小的参数解析器。其他使 用 has 或 is 前缀命名布尔元素 is_connect = True has_member = False 用复数形式命 名序列 members = ['user_1', 'user_2'] 用显式名称命名字典 person_address ={'user_1':'10 road WD', 'user_2' : '20 street huafu'} 避免通用名称 诸如 list, dict, sequence 或者 element 这样的名称应该避免。 避免现有名称 诸如 os, sys 这种系统已经存在的名称 应该避免。一些数字
一行列数：PEP 8 规定为 79 列。根据自己的情况，比如不要超过满屏时编辑 器的显示列数。
一个函数：不要超过 30 行代码, 即可显示在一个屏幕类，可以不使用垂直游标即可 看到整个函数。
一个类：不要超过 200 行代码，不要有超过 10 个方法。一个模块 不要超过 500 行。验证脚本 可以安装一个 pep8 脚本用于验证你的代码风格是否符合 PEP8。

请介绍一下字典

dict:字典，字典是一组键(key)和值(value)的组合，通过键(key)进行查找，没有顺序， 使用大括号”{}”; 应用场景： dict，使用键和值进行关联的数据;
现有字典 d={‘a’:24，‘g’:52，‘i’:12，‘k’:33}请按字典中的
value 值进行排序
sorted(d.items()，key = lambda x:x[1])

说一下字典和 json 的区别

字典是一种数据结构，json 是一种数据的表现形式，字典的 key 值只要是能 hash 的就行，json 的必须是字符串。

什么是可变、不可变类型

可变不可变指的是内存中的值是否可以被改变，不可变类型指的是对象所在内存块里面的值不可以 改变，有数值、字符串、元组；可变类型则是可以改变，主要有列表、字典。

存入字典里的数据有没有先后排序

存入的数据不会自动排序，可以使用 sort 函数对字典进行排序

字典推导式

```
d = {key: value for (key, value) in iterable}
```

如何将字典 D={‘Adam’: 95, ‘Lisa’: 85, ‘Bart’: 59}中的值’Adam’删除

```
del D['Adam']
```

请按照如下格式 K：V 打印出字典

```
for k,v in D.items():
print(k,":",v)
```

字符串

str:字符串是 Python 中最常用的数据类型。我们可以使用引号('或")来创建字符串。

String = “{1},{0}”; string = string.format(“Hello”, “Python”),请问将 string 打印出来为 Python,Hello

如何理解 Python 中字符串中的\字符

有三种不同的含义： 转义字符 路径名中用来连接路径名 编写太长代码手动软换行

将字符串"k:1|k1:2|k2:3|k3:4"，处理成 Python 字典：{k:1, k1:2, ... } # 字 典里的 K 作为字符串处理

```
str1 = "k:1|k1:2|k2:3|k3:4" def str2dict(str1): dict1 = {} for iterms in str1.spl
```

value return dict1

请判断一个字符串是否以 er 结尾

```
endswith
```

请将“#teacher#”两侧的#去掉

```
str = "#tea#"
b = str.replace("#","").strip()
```

请使用 map 函数将[1,2,3,4]处理成[1,0,1,0]

```
def f(x):
if x%2 == 0:
return 0 else:
return 1
b = map(f,[1,2,3,4])
print(list(b))
```

请按alist 中元素的age由大到小排序

alist [{'name':'a'，'age':20}，{'name':'b'，'age':30}，{'name':'c'，'age':25}] def sort_by_age(list1): return sorted(alist，key=lambda x:x['age']，reverse=True)

列表

list:是 Python 中使用最频繁的数据类型，在其他语言中通常叫做数组，通过索引进行查找，使用方括号”[]”, 列表是有序的集合。应用场景：定义列表使用 [ ] 定义，数据之间使用 “，”分割列表的索引从 0 开 始：索引就是数据在列表中的位置编号，索引又可以被称为下标。【注意】：从列表中取值时,如果超出索引 范围,程序会产生异常。IndexError: list index out of range

列表的常用操作

name_list = ["zhangsan", "lisi", "wangwu", "zhaoliu"]增加 列表名.insert(index, 数据)：在指定位置插入数据(位置前有空元素会补位） # 往列表 name_list 下标为 0 的地方插入数据 In [3]: name_list.insert(0, "Sasuke") In [4]: name_list Out[4]: ['Sasuke','zhangsan', 'lisi', 'wangwu', 'zhaoliu'] # 现有的列表下标是 0–4，如果我们要在下标是6 的地方插入数据，那个会自动插入到下标为 5 的地方，也就是# 插入到最后 In [5]: name_list.insert(6, "Tom") In [6]: name_list Out[6]: ['Sasuke', 'zhangsan', 'lisi', 'wangwu', 'zhaoliu', 'Tom'] 列表名.append(数据)：在列表的末尾追加数据(最常用的 方法) In [7]: name_list.append("Python") In [8]: name_list Out[8]: ['Sasuke', 'zhangsan', 'lisi', 'wangwu', 'zhaoliu', 'Tom', 'Python'] 列 表.extend(Iterable)：将可迭代对象中的元素追加到列表。 # 有两个列表 a 和 b a.extend(b) 会将 b 中的元素追加到列表 a 中 In [10]: a = [11, 22, 33] In [11]: b = [44, 55, 66] In [12]: a.extend(b) In [13]: a Out[13]: [11, 22, 33, 44, 55, 66]# 有列 表 c 和 字符串 d c.extend(d) 会将字符串 d 中的每个字符拆开作为元素插入到列表 c In [14]: c = ['j', 'a', 'v', 'a'] In [15]: d = "python" In [16]: c.extend(d) In [17]: c Out[17]: ['j', 'a', 'v', 'a', 'p', 'y', 't', 'h', 'o', 'n'] 取值和修改 取值：列表名[index]： 根据下标来取值。 In [19]: name_list = ["zhangsan",

"lisi", "wangwu", "zhaoliu"] In [20]: name_list[0] Out[20]: 'zhangsan' In [21]: name_list[3] Out[21]: 'zhaoliu' 修改：列表名[index] = 数据：修改指定索引的数据 In [22]: name_list[0] = "Sasuke" In [23]: name_list Out[23]: ['Sasuke','lisi', 'wangwu', 'zhaoliu']
删除 d0el 列表名[index]：删除指定索引的数据。 In [25]: name_list = ["zhangsan", "lisi", "wangwu", "zhaoliu"]# 删除索引是 1 的数据 In [26]: del name_list[1] In [27]: name_list Out[27]: ['zhangsan', 'wangwu', 'zhaoliu'] 列表名.remove(数 据)：删除第一个出现的指定数据 In [30]: name_list = ["zhangsan", "lisi", "wangwu", "zhaoliu", "lisi"]# 删除 第一次出现的 lisi 的数据 In [31]: name_list.remove("lisi") In [32]: name_list Out[32]: ['zhangsan', 'wangwu', 'zhaoliu', 'lisi'] 列表 名.pop()：删除末尾的数据,返回值: 返回被删除的元素 In [33]: name_list = ["zhangsan", "lisi", "wangwu", "zhaoliu"]# 删除最后一个元素 zhaoliu 并将元素 zhaoliu 返回 In [34]: name_list.pop() Out[34]: 'zhaoliu' In [35]: name_list Out[35]: ['zhangsan', 'lisi', 'wangwu'] 列表名.pop(index)：删除指定索引的数据，返回被删除的元素
In [36]: name_list = ["zhangsan", "lisi", "wangwu", "zhaoliu"]# 删除索引为 1 的数据 lisi In [37]: name_list.pop(1) Out[37]: 'lisi' In [38]: name_list Out[38]: ['zhangsan', 'wangwu', 'zhaoliu'] 列表名.clear()：清空整个列表的元素 In [40]: name_list = ["zhangsan", "lisi", "wangwu", "zhaoliu"] In [41]: name_list.clear() In [42]: name_list Out[42]: []
排序 列表名.sort()：升序排序 从小到大 In [43]: a = [33, 44, 22, 66, 11] In [44]: a.sort() In [45]: a Out[45]: [11, 22, 33, 44, 66] 列表 名.sort(reverse=True)：降序排序 从大到小。 In [46]: a = [33, 44, 22, 66, 11] In [47]: a.sort(reverse=True) In [48]: a Out[48]: [66, 44, 33, 22, 11] 列表 名.reverse()：列表逆序、反转 In [50]: a = [11, 22, 33, 44, 55] In [51]: a.reverse() In [52]: a Out[52]: [55, 44, 33, 22, 11]
统计相关 len(列表名)：得到列表的长度。 In [53]: a = [11, 22, 33, 44, 55] In [54]: len(a) Out[54]: 5 列表名.count(数据)：数据在列表中出现的次数。 In [56]: a = [11, 22, 11, 33, 11] In [57]: a.count(11) Out[57]: 3 列表名.index(数据)：数据 在列表中首次出现时的索引，没有查到会报错。 In [59]: a = [11, 22, 33, 44, 22] In [60]: a.index(22) Out[60]: 1 if 数据 in 列表： 判断列表中是否包含某元素 a = [11, 22, 33, 44 ,55] if 33 in a: print("找到了....")

循环遍历

使用 while 循环：

```
a = [11, 22, 33, 44, 55]
i = 0
while i < len(a):
    print(a[i])
    i += 1
```

使用 for 循环：

```
a = [11, 22, 33, 44, 55]
for i in a:
    print(i)
```

定义 A=[1,2,3,4],使用列表生成式[i*i for i in A]生成列表为

[1, 4, 9, 16]

如何将 L1 = [1,2,3,4],L2 = [6,7,8,9];使用列表内置函数变成

L1=[1,2,3,4,5,6,7,8,9]

L1.extend(L2)

给定一个有序列表，请输出要插入值 k 所在的索引位置

```
def index(list, key)
if key < list[0]:
position = 0 elif key >list[-1]:
position = len(list)-1 else:
for index in range(list):
if key>list[index] and list[index]>key:
position = index
```

给定两个list，A 和B，找出相同元素和不同元素

A、B 中相同元素：print(set(A)&set(B)) A、B 中不同元素：print(set(A)^set(B))

请反转字符串

```
new_str = old_str[::-1]
```

交换变量 a,b 的值

a,b = b,a

请使用 filter 函数将[1,2,3,4]处理成[2,4]

```
def f(x):
if x%2 == 0:
return x
b = filter(f,[1,2,3,4])
print(list(b))
```

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。

写一个列表生成式，产生一个公差为 11 的等差数列

```
print([x*11 for x in range(10)]
```

```
给定两个列表，怎么找出他们相同的元素和不同的元素
list1 = [1, 2, 3]list2 = [3, 4, 5]set1 = set(list1) set2 = set(list2)
print(set1&set2) print(set1^set2)
```

请写出一段 Python 代码实现删除一个 list 里面的重复元素

比较容易记忆的是用内置的 set：l1 = ['b', 'c', 'd', 'b', 'c', 'a', 'a'] l2 = list(set(l1)) print l2如果想要保持他们原来的排序：用 list 类的 sort 方法：l1 = ['b', 'c', 'd', 'b', 'c', 'a', 'a'] l2 = list(set(l1)) l2.sort(key=l1.index) print l2也可以这样写：l1 = ['b', 'c', 'd', 'b', 'c', 'a', 'a'] l2 = sorted(set(l1), key=l1.index) print l2也可以用遍历：l1 = ['b', 'c', 'd', 'b', 'c', 'a', 'a']l2 = []for i in l1: if not i in l2: l2.append(i) print l2

给定两个 list A ,B，请用找出 A ,B 中相同的元素，A ,B 中不同的元素

A、B 中相同元素：print(set(A)&set(B))A、B 中不同元素：print(set(A)^set(B))

有如下数组 list = range(10)我想取以下几个数组，应该如何切片

[1, 2, 3, 4, 5, 6, 7, 8, 9][1, 2, 3, 4, 5, 6][3, 4, 5, 6] [9] [1, 3, 5, 7, 9]答：[1:] [1:7] [3:7] [-1] [1::2]

下面这段代码的输出结果是什么

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)输出结果：list1 = [10, 'a']list2 = [123] list3 = [10, 'a']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

将以下 3 个函数按照执行效率高低排序

def f1(lln): l1 = sorted(lln) l2 = [i for i in l1 if i<0.5] return [i*i for i in l2] def f2(lln): l1 = [i for i in l1 if i<0.5] l2 = sorted(l1)

return [i*i for i in l2] def f3(lln): l1 = [i*i for i in lln] l2 = sorted(l1) return [i for i in l1 if i<(0.5*0.5)] 按执行效率从高到低排列：f2、f1 和 f3。要证明这个答案是正确的，你应该知道如何分析自己代码的性能。Python 中有一个很好的程序分析包， 可以满足这个需求import random import cProfile lln = [random.random() for i in
range(100000)]cProfile.run('f1(lln)')
cProfile.run('f2(lln)')cProfile.run('f3(lln)')

获取 1~100 被 6 整除的偶数?

```
1 def A(): alist = [] for i in range(1, 100): if i % 6 == 0: 5. alist.append(i) 6.
2 last_num = alist[-3:] 7. print(last_num)
```

元祖

tuple:元组，元组将多样的对象集合到一起，不能修改，通过索引进行查找，使用括号”()”; 应用场景：把
一些数据当做一个整体去使用，不能修改；

如何让元祖内部可变

元祖变成列表
定义 A=(“a”, “b”, “c”, “d”),执行 delA[2]后的结果为异常

集合

set:set 集合，在 Python 中的书写方式的{}，集合与之前列表、元组类似，可以存储多个数据，但 是这
些数据是不重复的。集合对象还支持 union(联合), intersection(交), difference(差)和 sysmmetric_difference(对称差集)等数学运算.快速去除列表中的重复元素In [4]: a =
[11,22,33,33,44,22,55] In [5]: set(a) Out[5]: {11, 22, 33, 44, 55}交集：共有的部分 In [7]: a =
{11,22,33,44,55} In [8]: b = {22,44,55,66,77} In [9]: a&b Out[9]:
{22, 44, 55}并集：总共的部分In [11]: a = {11,22,33,44,55}In [12]: b =
{22,44,55,66,77} In [13]: a | bOut[13]: {11, 22, 33, 44, 55, 66, 77}差集：另一个集 合中没有的部分In [15]: a = {11,22,33,44,55} In [16]: b = {22,44,55,66,77} In [17]: b – a Out[17]: {66, 77}对称差集(在 a 或 b 中，但不会同时出现在二者中)In [19]: a =
{11,22,33,44,55} In [20]: b = {22,44,55,66,77} In [21]: a ^ b Out[21]: {11, 33,66, 77}

Python 中类方法、类实例方法、静态方法有何区别

类方法：是类对象的方法，在定义时需要在上方使用“@classmethod”进行装饰，形参为 cls， 表示类对象，类对象和实例对象都可调用；类实例方法：是类实例化对象的方法，只有实例对象可以调用，形参为 self，指代对象本身；静态方法：是一个任意函数，在其上方使用“@staticmethod”进行装饰，可以用对象 直接调用， 静态方法实际上跟该类没有太大关系

Python 中如何动态获取和设置对象的属性

if hasattr(Parent，'x'): print(getattr(Parent，'x')) setattr(Parent，'x'，3)

print(getattr(Parent，'x'))

Python 的内存管理机制及调优手段

内存管理机制：引用计数、垃圾回收、内存池。

引用计数：

引用计数是一种非常高效的内存管理手段， 当一个 Python 对象被引用时其引用计数增加 1， 当其不再被一个变量引用时则计数减 1. 当引用计数等于 0 时对象被删除。

垃圾回收 ：

1. 引用计数 引用计数也是一种垃圾收集机制，而且也是一种最直观，最简单的垃圾收集技术。当 Python 的某 个对象的引用计数降为 0 时，说明没有任何引用指向该对象，该对象就成为要被回收的垃圾了。比如 某个新建对象，它被分配给某个引用，对象的引用计数变为 1。如果引用被删除，对象的引用计数为 0， 那 么该对象就可以被垃圾回收。不过如果出现循环引用的话，引用计数机制就不再起有效的作用了

2. 标记清除 如果两个对象的引用计数都为 1，但是仅仅存在他们之间的循环引用，那么这两个对象都是需要 被 回收的，也就是说，它们的引用计数虽然表现为非 0，但实际上有效的引用计数为 0。所以先将循环引 用 摘掉，就会得出这两个对象的有效计数。

3. 分代回收 从前面“标记-清除”这样的垃圾收集机制来看，这种垃圾收集机制所带来的额外操作实际上与系 统 中总的内存块的数量是相关的，当需要回收的内存块越多时，垃圾检测带来的额外操作就越多，而垃圾 回 收带来的额外操作就越少；反之，当需回收的内存块越少时，垃圾检测就将比垃圾回收带来更少的额外操作 举个例子： 当某些内存块 M 经过了 3 次垃圾收集的清洗之后还存活时，我们就将内存块 M 划到一个集合

A 中去，而新分配的内存都划分到集合 B 中去。当垃圾收集开始工作时，大多数情况都只对集合 B 进 行垃 圾回收，而对集合 A 进行垃圾回收要隔相当长一段时间后才进行，这就使得垃圾收集机制需要处 理的内存少 了，效率自然就提高了。在这个过程中，集合 B 中的某些内存块由于存活时间长而会被转 移到集合 A 中， 当然，集合 A 中实际上也存在一些垃圾，这些垃圾的回收会因为这种分代的机制而 被延迟。 内存池： 1. Python 的内存机制呈现金字塔形状，-1，-2 层主要有操作系统进行操作； 2. 第 0 层是 C 中的 malloc，free 等内存分配和释放函数进行操作； 3. 第 1 层和第 2 层是内存池，有 Python 的接口函 数 PyMem_Malloc 函数实现，当对象小于 256K 时有该层直接分配内存； 4. 第 3 层是最上层，也就是 我们对 Python 对象的直接操作； Python 在运行期间会大量地执行 malloc 和 free 的操作，频繁地 在用户态和核心态之间进行切 换，这将严重影响 Python 的执行效率。为了加速 Python 的执行效率， Python 引入了一个内存池 机制，用于管理对小块内存的申请和释放。 Python 内部默认的小块内存与大块 内存的分界点定在 256 个字节，当申请的内存小于 256 字节 时，PyObject_Malloc 会在内存池中申请 内存；当申请的内存大于 256 字节时，PyObject_Malloc 的 行为将蜕化为 malloc 的行为。当然，通 过修改 Python 源代码，我们可以改变这个默认值，从而改 变 Python 的默认内存管理行为。

调优手段（了解）

1.手动垃圾回收

2.调高垃圾回收阈值

3.避免循环引用（手动解循环引用和使用弱引用）

内存泄露是什么？如何避免

指由于疏忽或错误造成程序未能释放已经不再使用的内存的情况。内存泄漏并非指内存在物理上的消失，而是应用程序分配某段内存后，由于设计错误，失去了对该段内存的控制，因而造成了内存的浪费。导致程序运行 速度减慢甚至系统崩溃等严重后果。 有del() 函数的对象间的循环引用是导致内存泄漏的主凶。 不 使用一个对象时使用:del object 来删除一个对象的引用计数就可以有效防止内存泄漏问题。 通过Python 扩展模块 gc 来查看不能回收的对象的详细信息。 可以通过 sys.getrefcount(obj) 来获取对 象的引用计数，并根据返回值是否为0 来判断是否内存泄漏

Python 函数调用的时候参数的传递方式是值传递还是引用传递

Python 的参数传递有：位置参数、默认参数、可变参数、关键字参数。函数的传值到底是值传递还是引用传递，要分情况： 不可变参数用值传递：像整数和字符串这样的不可变对象，是通过拷贝进行传递的，因为你无 论如何都不可能在原处改变 不可变对象 可变参数是引用传递的： 比如像列表，字典这样的对象是通过引用 传递、和 C 语言里面的用指针传递数组很相似，可变对象 能在函数内部改变。

对缺省参数的理解

缺省参数指在调用函数的时候没有传入参数的情况下，调用默认的参数，在调用函数的同时赋值时，所传入的参数会替代默认参数。*args 是不定长参数，他可以表示输入参数是不确定的，可以是任意多个。

**kwargs 是关键字参数，赋值的时候是以键 = 值的方式，参数是可以任意多对在定义函数的时候 不确定 会有多少参数会传入时，就可以使用两个参数。

为什么函数名字可以当做参数用

Python 中一切皆对象，函数名是函数在内存中的空间，也是一个对象

Python 中 pass 语句的作用是什么

在编写代码时只写框架思路，具体实现还未编写就可以用 pass 进行占位，使程序不报错，不会进行任何操作

有这样一段代码，print c 会输出什么，为什么

a = 10 b = 20 c = [a]a = 15答：10 对于字符串、数字，传递是相应的值

交换两个变量的值

a,b = b,a

map 函数和 reduce 函数?

从参数方面来讲：map()包含两个参数，第一个参数是一个函数，第二个是序列（列表 或元组）。其中，函数（即 map 的第一个参数位置的函数）可以接收一个或多个参数。 reduce()第一个参数是函数，第 二个是序列（列表或元组）。但是，其函数必须接收两个参数。从对传进去的数值作用来讲：map()是将传 入的函数依次作用到序列的每个元素，每个元素都是独自被函数“作用”一次 。 reduce()是将传人的函数作 用在序列的第一个元素得到结果后，把这个结果继续与下一个元素作用 （累积计算）。

递归函数停止的条件

递归的终止条件一般定义在递归函数内部，在递归调用前要做一个条件判断，根据判断的结果选择 是继续调用自身，还是 return;返回终止递归。 终止的条件： 1. 判断递归的次数是否达到某一限定值 2. 判断运算 的结果是否达到某个范围等，根据设计的目的来选择

回调函数，如何通信的

回调函数是把函数的指针(地址)作为参数传递给另一个函数，将整个函数当作一个对象，赋值给调 用的函 数。

Python 主要的内置数据类型都有哪些？ print dir(‘a ’) 的输出

内建类型：布尔类型、数字、字符串、列表、元组、字典、集合； 输出字符串‘a’的内建方法；

map(lambda x:x*x，[y for y in range(3)])的输出

[0， 1， 4]

hasattr() getattr() setattr() 函数使用详解

hasattr(object, name)函数： 判断一个对象里面是否有 name 属性或者 name 方法，返回 bool 值，有 name 属性(方法)返回 True， 否则返回 False。 注意：name 要使用引号括起来class function_demo(object): name = 'demo' def run(self): return "hello function" functiondemo = function_demo() res = hasattr(functiondemo,
'name') #判断对象是否有 name 属性，True res = hasattr(functiondemo, "run") #判断 对象是否有 run 方法，True res = hasattr(functiondemo, "age") #判断对象是否有 age 属性，Falsw print(res) getattr(object, name[,default]) 函数：获取对象 object 的属性或者方法，如果存在则打印出来，如果不存在，打印默认值，默认值可选。 注意：如果返回的是对象的方法，则打印结果是：方法的内存地 址，如果需要运行这个方法，可以在后 面添加括号()。1functiondemo = function_demo() getattr(functiondemo, 'name') #获取 name 属性，存在就打印出来--- demo
getattr(functiondemo, "run") #获取 run 方法，存在打印出 方法的内存地址---<bound method function_demo.run of < main .function_demo object at 0x10244f320>> getattr(functiondemo, "age") #获取不存在的属性，报错如下： Traceback (most recent
call last): File
"/Users/liuhuiling/Desktop/MT_code/OpAPIDemo/conf/OPCommUtil.py", line 39, in <module> res = getattr(functiondemo, "age") AttributeError: 'function_demo' object has no attribute 'age' getattr(functiondemo, "age", 18) #获取不存在的属性，返 回一个默认值 setattr(object,name,values)函数：给对象的属性赋值，若属性不存在，先创建再赋值class function_demo(object): name = 'demo' def run(self): return "hello function" functiondemo = function_demo() res = hasattr(functiondemo, 'age') # 判断 age 属性是否存在，False print(res) setattr(functiondemo, 'age', 18) #对 age 属性进行赋值，无返回值 res1 = hasattr(functiondemo, 'age') #再次判断属性是否存在，True

综合使用：class function_demo(object): name = 'demo' def run(self):

return "hello function" functiondemo = function_demo() res = hasattr(functiondemo, 'addr') # 先判断是否存在 if res: addr = getattr(functiondemo, 'addr') print(addr)else: addr = getattr(functiondemo, 'addr', setattr(functiondemo, 'addr', '北京首都'))
#addr = getattr(functiondemo, 'addr', '美国纽约') print(addr)

请使用 reduce 函数计算 100 的阶乘

from functools import reduce sum=reduce(lambda x,y:x*y,range(1,101))
print(sum)

一句话解决阶乘函数

reduce(lambda x,y: x*y, range(1,n+1))

什么是 lambda 函数? 有什么好处

lambda 函数是一个可以接收任意多个参数(包括可选参数)并且返回单个表达式值的函数1、lambda 函数比
较轻便，即用即仍，很适合需要完成一项功能，但是此功能只在此一处使用， 连名字都很随意的情况下；
2、匿名函数，一般用来给 filter， map 这样的函数式编程服务; 3、作为回调函数，传递给某些应用，比 如消息处理

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()]上面代码输出的结果是[6， 6， 6， 6] (不是我们想的[0， 2， 4，
6])。 你如何修改上面的 multipliers 的定义产生想要的结果? 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)]

什么是 lambda 函数? 它有什么好处? 写一个匿名函数求两个数的和

lambda 函数是匿名函数；使用 lambda 函数能创建小型匿名函数。这种函数得名于省略了用 def 声明函
数的标准步骤；f = lambda x，y:x+y print(f(2017，2018))

请手写一个单例(设计模式)

class A(object):

instance = None def new (cls, *args, **kwargs):
if cls. instance is None: cls. instance = object. new (cls)
return cls. instance else: return cls. Instance

单例模式的应用场景有哪些

单例模式应用的场景一般发现在以下条件下： (1）资源共享的情况下，避免由于资源操作时导致的性能或损耗等。如日志文件，应用配置。 (2）控制资源的情况下，方便资源之间的互相通信。如线程池等。 1.网站 的计数器 2.应用配置 3.多线程池 4. 数据库配置，数据库连接池 5.应用程序的日志应用

对装饰器的理解 ，并写出一个计时器记录方法执行性能的装饰器

装饰器本质上是一个 Python 函数，它可以让其他函数在不需要做任何代码变动的前提下增加额外 功能，装 饰器的返回值也是一个函数对象import time def timeit(func): def wrapper(): start = time.clock() func() end =time.clock() print 'used:', end – start
return wrapper @timeit def foo(): print 'in foo()'foo()

解释一下什么是闭包

在函数内部再定义一个函数，并且这个函数用到了外边函数的变量，那么将这个函数以及用到的一些变量称之为闭包

函数装饰器有什么作用

装饰器本质上是一个 Python 函数，它可以在让其他函数在不需要做任何代码的变动的前提下增加额外的功能。装饰器的返回值也是一个函数的对象，它经常用于有切面需求的场景。 比如：插入日志、性能测试、事 务处理、缓存、 权限的校验等场景 有了装饰器就可以抽离出大量的与函数功能本身无关的雷同代码并发并继 续使用

生成器、迭代器的区别

迭代器是一个更抽象的概念，任何对象，如果它的类有 next 方法和 iter 方法返回自己本身，对于 string、list、 dict、tuple 等这类容器对象，使用 for 循环遍历是很方便的。在后台 for 语句对容 器对象调用 iter()函数，iter() 是 python 的内置函数。iter()会返回一个定义了 next()方法的迭 代器对象，它在容器中逐个访问容器内元素，next() 也是 python 的内置函数。在没有后续元素时， next()会抛出一个 StopIteration 异常。 生成器（Generator）是创建迭代器的简单而强大的工具。它 们写起来就像是正规的函数，只是在需要返回数 据的时候使用 yield 语句。每次 next()被调用时，生成 器会返回它脱离的位置（它记忆语句最后一次执行的位置 和所有的数据值） 区别：生成器能做到迭代器能做 的所有事,而且因为自动创建了 iter()和 next()方法,生成器显得特别简洁,而且 生成器也是高效的，使 用生成器表达式取代列表解析可以同时节省内存。除了创建和保存程序状态的自动方法,当 发生器终结时,还 会自动抛出 StopIteration 异常

X 是什么类型

X = (for i in ramg(10)) X 是 generator 类型

请尝试用“一行代码”实现将 1-N 的整数列表以 3 为单位分组，比如 1-100 分组后为?

print([[x for x in range(1，100)][i:i+3] for i in range(0，len(list_a)，3)])

Python 中 yield 的用法

yield 就是保存当前程序执行状态。你用 for 循环的时候，每次取一个元素的时候就会计算一次。用 yield 的函数 叫 generator，和 iterator 一样，它的好处是不用一次计算所有元素，而是用一次算一次，可以节省很多空间。generator 每次计算需要上一次计算结果，所以用 yield，否则一 return，上次计算结果就没了。>>> def createGenerator(): mylist = range(3) for i in mylist: yield i*i >>> mygenerator = createGenerator() # create a generator >>> print(mygenerator) # mygenerator is an object!<generator object createGenerator at 0xb7555c34> >>> for i in mygenerator: print(i) 0 1 4

Python 中的可变对象和不可变对象

不可变对象，该对象所指向的内存中的值不能被改变。当改变某个变量时候，由于其所指的值不能被改变，相当于把原来的值复制一份后再改变，这会开辟一个新的地址，变量再指向这个新的地址。 可变对象，该对象 所指向的内存中的值可以被改变。变量（准确的说是引用）改变后，实际上是其所指的值直接 发生改变，并没 有发生复制行为，也没有开辟新的出地址，通俗点说就是原地改变。

Python 中，数值类型（int 和 float）、字符串 str、元组 tuple 都是不可变类型。而列表 list、字典 dict、集合 set 是可变类型

Python 中 is 和==的区别

is 判断的是 a 对象是否就是 b 对象，是通过 id 来判断的。==判断的是 a 对象的值是否和 b 对象的值相等，是通过 value 来判断的

Python 的魔法方法

魔法方法就是可以给你的类增加魔力的特殊方法，如果你的对象实现 （重载）了这些方法中的某一个，那么这个 方法就会在特殊的情况下被 Python 所调用，你可以定义自己想要的行为，而这一切都是自动发生的。 它们经常是 两个下划线包围来命名的（比如 init ， lt ），Python 的魔法方法是非常强大的，

所以了解其使用方法也变得尤为重要！ init构造器，当一个实例被创建的时候初始化的方法。但是它并不是实例化调用的第一个方法。 new 才是实例化对象调用的第一个方法，它只取下cls参数，并把其他参数传给init 。 new 很少使用，但是也有它适合的场景，尤其是当类继承自一个像元组或者字符串这样不经常改变的类型的时候。call允许一个类的实例像函数一样被调用 。

getitem 定义获取容器中指定元素的行为，相当于 self[key] 。

getattr 定义当用户试图访问一个不存在属性的时候的行为 。

setattr 个属性被访问的时候的行为定义当一个属性被设置的时候的行为 。

getattribute 定义当一个属性被访问的时候的行为

面向对象中怎么实现只读属性?

将对象私有化，通过共有方法提供一个读取数据的接口class person: def init (self,x):

self. age = 10; def age(self): return self. age; t = person(22)#

t. age = 100 print(t.age())最好的方法class MyCls(object):

weight = 50

@property #以访问属性的方式来访问 weight 方法 def weight(self): return
self. weight if name
== ' main ': obj = MyCls() print(obj.weight)
obj.weight = 12 Traceback (most recent call last): 50 File "C:/PythonTest/test.py", line 11, in
<module>obj.weight = 12 AttributeError: can't set attribute

谈谈你对面向对象的理解

面向对象是相对于面向过程而言的。面向过程语言是一种基于功能分析的、以算法为中心的程序设计方法；而
面 向对象是一种基于结构分析的、以数据为中心的程序设计思想。在面向对象语言中有一个有很重要东西，叫 做类。 面向对象有三大特性：封装、继承、多态

Python 里match 与search 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

Python 字符串查找和替换

re.findall(r’目的字符串’，’原有字符串’）#查询
re.findall(r'cast'，'itcast.cn')[0] re.sub(r‘要替换原字符’，’要替换新字符’，’原始字符串’）
re.sub(r'cast'，'heima'，'itcast.cn')

用 Python 匹配 HTML g tag 的时候，<.> 和 <.?> 有什么区别

<.*>是贪婪匹配，会从第一个“<”开始匹配，直到最后一个“>”中间所有的字符都会匹配到，中间可能会包含
“<>”。 <.*?>是非贪婪匹配，从第一个“<”开始往后，遇到第一个“>”结束匹配，这中间的字符串都会匹配 到，但是 不会有“<>”

请写出下列正则关键字的含义

进程总结

进程：程序运行在操作系统上的一个实例，就称之为进程。进程需要相应的系统资源：内存、时间片、pid。
创建进程：1.首先要导入 multiprocessing 中的 Process； 2.创建一个 Process 对象；
3. 创建Process 对象时，可以传递参数；p = Process(target=XXX, args=(元组,) , kwargs={key:value}) 2target = XXX 指定的任务函数,不用加() args=(元组,) , kwargs={key:value}给任务函数传递的参数
4.使用 start()启动进程；

5.结束进程Process 语法结构：Process([group[, target [, name [, args [, kwargs]]]]])target：如果传递了函数的引用，可以让这个子进程 就执行函数中的代码 args：给 target 指定的函数传递的参数，以元组的形式进行传递 kwargs：给 target 指定的函数传递参数，以字典的形式进行传递 name：给进程设定一个名字，可以省略 group：指 定进程组，大多数情况下用不到 Process 创建的实例对象的常用方法有：start()：启动子进程实例(创建 子进程) is_alive()：判断进程子进程是否还在活着 join(timeout)：是否等待子进程执行结束，或者等 待多少秒terminate()：不管任务是否完成，立即终止子进程 Process 创建的实例对象的常用属性： name：当前进程的别名，默认为 Process-N,N 为从 1 开始递增的整数pid：当前进程的 pid(进程号)

给子进程指定函数传递参数 Demo：import osfrom multiprocessing import Process import time 3def pro_func(name, age, **kwargs): for i in range(5): print("子进程正 在运行中,name=%s, age=%d, pid=%d" %(name, age, os.getpid())) print(kwargs)

time.sleep(0.2) if __name

== '__main__': # 创建 Process 对象 p =

Process(target=pro_func, args=('小明',18), kwargs={'m': 20}) # 启动进程 p.start() time.sleep(1) # 1 秒钟之后，立刻结束子进程 p.terminate() p.join()注意：进程间 不共享全局变量

进程之间的通信-Queue

在初始化 Queue()对象时，(例如 q=Queue()，若在括号中没有指定最大可接受的消息数量，或数 量为负 值时，那么就代表可接受的消息数量没有上限-直到内存的尽头) Queue.qsize()：返回当前队列包含的消息 数量。 Queue.empty()：如果队列为空，返回 True,反之 False。 Queue.full()：如果队列满了，返 回 True，反之 False。 Queue.get([block[,timeout]])：获取队列中的一条消息，然后将其从队列 中移除，block 默认值为 True。如果 block 使用默认值，且没有设置 timeout（单位秒），消息列队如 果为空，此时程序将被阻塞 （停在读取状态），直到从消息列队读到消息为止，如果设置了 timeout，则会 等待 timeout 秒，若还 没读取到任何消息，则抛出"Queue.Empty"异常； 如果 block 值为 False， 消息列队如果为空，则会立刻抛出"Queue.Empty"异常；Queue.get_nowait()：相当 Queue.get(False)；Queue.put(item,[block[, timeout]])：将 item 消息写入队列，block 默 认值为 True；如果 block 使用默认值，且没有设置 timeout（单位秒），消息列队如果已经没有空间可 写入，此 时程序将被阻塞（停在写入状态），直到从消息列队腾出空间为止，如果设置了 timeout，则会等 待 timeout 秒，若还没空间，则抛出"Queue.Full"异常； 如果 block 值为 False，消息列队如果没 有空间可写入，则会立刻抛出"Queue.Full"异常； Queue.put_nowait(item)：相当 Queue.put(item, False)；

进程间通信 Demofrom multiprocessing import Process, Queueimport os, time, random # 写数据进程执行的代码:def write(q): for value in ['A', 'B', 'C']: print('Put %s to queue...' % value) q.put(value) time.sleep(random.random()) # 读数据 进程执行的代码:def read(q): while True: if not q.empty():

value = q.get(True) print('Get %s from queue.' % value)

time.sleep(random.random()) else: breakif

__name__ =='__main__': # 父进程创建 Queue，并传给各个子进程： q = Queue()

pw = Process(target=write, args=(q,)) pr = Process(target=read, args=(q,))
# 启动子进程 pw，写入: pw.start() # 等待 pw 结束: pw.join() # 启动子进程 pr，读 取: pr.start()
pr.join() # pr 进程里是死循环，无法等待其结束，只能强行终止:
print('') print('所有数据都写入并且读完')
进程池 Pool# -*- coding:utf-8 -*- from multiprocessing import Poolimport os, time,
randomdef worker(msg): t_start = time.time() print("%s 开始执行,进程号为%d"
% (msg,os.getpid())) # random.random()随机生成 0~1 之间的浮点数
time.sleep(random.random()*2) t_stop = time.time() print(msg,"执行完毕，耗 时%0.2f" %
(t_stop-t_start)) po = Pool(3) # 定义一个进程池，最大进程数 3for i in range(0,10): #
Pool().apply_async(要调用的目标,(传递给目标的参数元祖,)) # 每次循 环将会用空闲出来的子进程去
调用目标 po.apply_async(worker,(i,)) print("---- start----") po.close() # 关闭进程池，关闭后 po
不再接收新的请求 po.join() # 等 待 po 中所有子进程执行完成，必须放在 close 语句之后 print("--
-----end-----")
multiprocessing.Pool 常用函数解析：apply_async(func[, args[, kwds]]) ：使用非阻塞方式
调用 func（并行执行，堵塞方式必须等待 上一个进程退出才能执行下一个进程），args 为传递给
func 的参数列表，kwds 为传递给 func 的关键字参数列表； close()：关闭 Pool，使其不再接受新的
任 务；terminate()：不管任务是否完成，立即终止； join()：主进程阻塞，等待子进程的退出， 必须
在 close 或 terminate 之后使用；
进程池中使用 Queue：如果要使用 Pool 创建进程，就需要使用 multiprocessing.Manager()中的
Queue()，而不是 multiprocessing.Queue()，否则会得到一条如下的错误信息： RuntimeError:
Queue objects should only be shared between processes through inheritancefrom
multiprocessing import Manager,Poolimport os,time,random def reader(q): print("reader 启动
(%s),父进程为(%s)" % (os.getpid(), os.getppid())) for i in range(q.qsize()): print("reader 从 Queue
获取到消息：%s" % q.get(True))def writer(q): print("writer 启动(%s),父进程为(%s)" %
(os.getpid(), os.getppid())) for i in "itcast": q.put(i) if name == " main ": print("(%s) start"
% os.getpid()) q = Manager().Queue() # 使用 Manager 中的 Queue po = Pool()
po.apply_async(writer, (q,)) time.sleep(1) # 先让上面的任务向 Queue 存入数据，然后 再让下面的
任务开始从中取数据 po.apply_async(reader, (q,)) po.close() po.join() print("(%s) End" %
os.getpid())

什么是僵尸进程和孤儿进程？怎么避免僵尸进程

孤儿进程：父进程退出，子进程还在运行的这些子进程都是孤儿进程，孤儿进程将被 init 进程(进 程号
为

1) 所收养，并由 init 进程对它们完成状态收集工作。 僵尸进程：进程使用 fork 创建子进程，如果子进 程退出，而父进程并没有调用 wait 或 waitpid 获 取子进程的状态信息，那么子进程的进程描述符仍然保 存在系统中的这些进程是僵尸进程。
2) 避免僵尸进程的方法： 1.fork 两次用孙子进程去完成子进程的任务；
2.用 wait()函数使父进程阻塞； 3.使用信号量，在 signal handler 中调用 waitpid，这样父进程不 用阻塞

Python 中的进程与线程的使用场景

多进程适合在 CPU 密集型操作(cpu 操作指令比较多，如位数多的浮点运算)。 多线程适合在 IO 密集型操
作(读写数据操作较多的，比如爬虫)。

线程是并发还是并行，进程是并发还是并行

线程是并发，进程是并行； 进程之间相互独立，是系统分配资源的最小单位，同一个线程中的所有线程共享资源
并行（parallel）和并发（concurrency）
并行：同一时刻多个任务同时在运行。
并发：在同一时间间隔内多个任务都在运行，但是并不会在同一时刻同时运行，存在交替执行的情况
实现并行的库有：multiprocessing
实现并发的库有：threading
程序需要执行较多的读写、请求和回复任务的需要大量的 IO 操作，IO 密集型操作使用并发更好。
CPU 运算量大的程序程序，使用并行会更好

IO 密集型和 CPU 密集型区别

IO 密集型：系统运作，大部分的状况是 CPU 在等 I/O (硬盘/内存)的读/写。
CPU 密集型：大部份时间用来做计算、逻辑判断等 CPU 动作的程序称之 CPU 密集型
a = “abbbccc”，用正则匹配为 abccc,不管有多少 b，就出现一次

1. 思路：不管有多少个 b 替换成一个 2. re.sub(r'b+', 'b', a)

带参数的装饰器

带定长参数的装饰器

```
def new_func(func):

def wrappedfun(username,passwd):
```

```
if username == 'root' and passwd == '123456789':
print('通过认证！') print('开始执行附加功能') return func()
else: print('用户名或密码错误') return

return wrappedfun

@new_func

def orign():
print('开始执行函数')

orign('root','123456789')
```

带不定长参数的装饰器

```
def new_func(func):

def wrappedfun(*parts):

if parts:
counts = len(parts) print('本系统包含 ', end='') for part in parts:
print(part, ' ', end='') print('等', counts, '部分') return func()
else: print('用户名或密码错误') return func()

return wrappedfun

@new_func

def orign(): print('开始执行函数') orign('硬件', '软件', '用户数据')
```

同时带不定长、关键字参数的装饰器

```
def new_func(func):

def wrappedfun(*args,**kwargs):

if args:
counts = len(args) print('本系统包含 ',end='') for arg in args:
print(arg,' ',end='') print('等',counts,'部分') if kwargs:
for k in kwargs: v= kwargs[k] print(k,'为： ',v)


return func()

else:

if kwargs:

for kwarg in kwargs:
print(kwarg) k,v = kwarg print(k,'为： ',v)

return func()

return wrappedfun

@new_func

def orign(): print('开始执行函数') orign('硬件','软件','用户数据',总用户数=5,系统版本='CentOS 7.4')
```

正则表达式贪婪与非贪婪模式的区别

在形式上非贪婪模式有一个“？”作为该部分的结束标志。 在功能上贪婪模式是尽可能多的匹配当前正则表达
式，可能会包含好几个满足正则表达式的字符串， 非贪婪模式，在满足所有正则表达式的情况下尽可能少的匹 配当前正则表达式

写出开头匹配字母和下划线，末尾是数字的正则表达式

^[A-Za-z]_.*\d$

正则表达式操作

1. 匹配手机号
分析：
（1）手机号位数为 11 位；
（2）开头为 1，第二位为 3 或 4 或 5 或 7 或 8;
表达式为：/^[1][3,4,5,7,8][0-9]{9}$/； 。
2. 请匹配出变量 A ='json({"Adam":95,"Lisa":85,"Bart":59})'中的 json 字符串

1. A = 'json({"Adam":95,"Lisa":85,"Bart":59})'

2. b = re.search(r'json.*?({.*?}).*',A,re.S)

3. print(b.group(1))
怎么过滤评论中的表情

1. co = re.compile(u'[\uD800-\uDBFF][\uDC00-\uDFFF]')

2. co.sub(‘’,text)

Python 中 pass 语句的作用是什么

在编写代码时只写框架思路，具体实现还未编写就可以用 pass 进行占位，使程序不报错，不会进行任何操
作。

尽可能写出多的 str 方法

strina.catalizeO

strino.center(width)

–

–width

strina.. ωunt(str bea =0.

e d=len(inall

str

sgtringbegend

strino.decode(encodin 'UTF–S'

encodingstring .

ValueErrorerrors m

'ignore' !IJGreplace'

strino.encode(encodina ='UTF−8'. ωdingstring .

!1

strina.endswh(obL bea=O

d= n(strinall

s1l' nO.expandtabs(tabsize=81

''/

ValueErer rrors'ignore'

'replace'

lf!ojbegend Jj!!.Jl

EobjT .lil!

Fal .

4

lf! stringtabtab

8.

.

;

Etrina.finds! tr. b =

d=len(strinall

i

rina.index(bea=o.

d=len! rinall

Q

:

p
strina.isdeci 10

gg.isd l

i2ngbegend

'

–1
find(J –strstring–

string –E

True'J.i1tJ .\i

tring– . @ '

True

p

tJ .\iFalse . –

Estring .31–9!.JliTrueFal .

tringTrueFalse .

4

.

;

string Jlti tJeCJJlt!True . !

False

I

i
strinQ.liu t(width)

L
.lstri strinQ.maketrans(intab.outtabll
J

string –

( )True.False

stringq { }

–

Bwidth

string .

nng y

maketrans()

_

...

.

$str aJ. :A

Q
strin a.oartition(st 7

.

str .J aJ.

nd(s)plO –

!I! string–3 '

{ing _preT st ng_post_' J.string

.A.

Y
rin g_pre_str == string

strino.r lace(strl.str2. Je! string5trl .lil\ str2num J!!

nu =strino.countlstrl\\

num.

d nd() .

beQend=len(strinoll

strino.rind l str.

beQ=Oend= len(strino))

index()

strino.riustlwidth\ width

stripartítion partition()

string

strstringnum num=stricrountls!r 1 num @

strino.501itlin

7fV \r\nn' ' –
keepend5False 3
TrueJj!tJfj A

Strin .s!artsw (obi bjHTrue

beQ=O.end=len(strino))

.begend

生成一个斐波那契数列

def fib(times):

n = 0

a,b = 0,1

while n<times:

print(b)

a,b = b,a+b n+=1
return 'done

说明一下 os.path 和 sys.path 分别代表什么

os.path 主要是用于用户对系统路径文件的操作
sys.path 主要用户对 Python 解释器的系统环境参数的操作。

什么是 lambda 函数？ 有什么好处

lambda 函数是一个可以接收任意多个参数(包括可选参数)并且返回单个表达式值的函数。
1、lambda 函数比较轻便，即用即仍，很适合需要完成一项功能，但是此功能只在此一处使用， 连名字都很 随意的情况下；
2、匿名函数，一般用来给 filter， map 这样的函数式编程服务; 3、作为回调函数，传递给某些应用，比 如消息处理。

有一个多层嵌套列表A=[1,2,[3.4["434",[...]]]]请写一段代码

遍历A中的每一个元素并打印出来。

思路：就是有几个嵌套链表就用几个 for 循环进行迭代,然后对最后一个结果进行打印

```
a= ["a","b",["1","3",["4","haha"]]]

for b in a :

for c in b :

for d in c :

print(d)
```

现在需要从一个简单的登陆网站获取信息，请使用 Python 写出简要的登陆函数的具 体实现？（登录信息只包含用户

名，密码）

session = requests.session()

response = session.get(url,headers)

如何在一个 function 里面设置一个全局变量?

Global 声明。

描述 yield 使用场景

生成器。 当有多个返回值时，用 return 全部一起返回了，需要单个逐一返回时可以用 yield。

生成 1~10 之间的整数

for i in range(1,10)
生成器：(i for i in range(1,10))

Python 如何生成缩略图

import os import glob
from PIL import Image

def thumbnail_pic(path): a=glob.glob(r'./*.jpg') for x in a:
name=os.path.join(path,x) im=Image.open(name) im.thumbnail((80,80))
print(im.format,im.size,im.mode) im.save(name,'JPEG') print('Done!')

```
if __name__ == '__main__':
path='.' thumbnail_pic(path)
```

代码优化从哪些方面考虑？有什么想法

1.优化算法时间复杂度。
2.减少冗余数据。
3.合理使用 copy 与 deepcopy。
4.使用 dict 或 set 查找元素。
5.合理使用生成器（generator）和 yield。 6.优化循环。
7.优化包含多个判断表达式的顺序。
8.使用 join 合并迭代器中的字符串。
9.选择合适的格式化字符方式。
10 不借助中间变量交换两个变量的值。
11.使用 if is。
12.使用级联比较 x < y < z。
13.while 1 比 while True 更快。
14.使用**而不是 pow。
15.使用 cProfile, cStringIO 和 cPickle 等用 c 实现相同功能（分别对应 profile, StringIO, pickle） 的包。
16.使用最佳的反序列化方式。
17.使用 C 扩展(Extension)。
18.并行编程。
19.终级大杀器：PyPy。
20.使用性能分析工具