# 面试题

1. 考虑如下python 代码，如果运行结束，命令行中的运行结果是什么？

if \_\_"name"\_\_ == "\_\_main\_\_":

I = []

for i in xrange(10):

I.append({"num":I})

print I

再考虑以下代码，运行结束后的结果是什么？

if \_\_"name"\_\_ == "\_\_main\_\_":

I = []

a = {"num":0}

for iin xrange(10)

a[num] = i

I.append(a)

print I

两端代码的运行结果是否相问，如果不相同，原因是什么？

答：否，一个value是列表，另一个是循环变量

1. 设想有一个计算集群，其中包括大量规格不同的计算节点（假没分为4核8g，8核16g,16核32g 三个不同种类)现在我们要分配大量计算任务在该集群上（假设远远超过集群同时可以提供的最大算力）每个任务占用资源各不相同（假没分为：单核2g，2核4g,4核8g 三个不同任务）每个任务的运行时长也各不相同，请设计一个任务调度器

使该集群可以在尽可能短的时间内完成所有任务，要求写出整体设计

路、所要使用的技术/工具以及调度策略

1. a = [] a = 1 a = 1

def fun(a): def fun(a): def fun():

a.append(1) a = 2 a = 2

fun(a) fun(a) print a

print a print a fun(a)

print a

解：[1] 1 2

1

1. class Person(object):

name = ‘’

weight = 50

list = [4,5,6]

p1 = Person()

p2 = Person()

Person.name = ‘Robot’

p1.list.append(1)

p2.weight = 99

print(p1.name,p1.weight,p1.list) “Robot” 50 [4, 5, 6, 1]

print(p2.name,p2.weight,p2.list) “Robot” 99 [4, 5, 6, 1]

print(Person.name,Person.weight,Person.list) “Robot” 50 [4, 5, 6, 1]

1. def f(a,b=[]):

for i in range(a):

b.append(i\*i)

print(b)

f(2)

[0]

[0, 1]

f(3,[1,2])

[1, 2, 0]

[1, 2, 0, 1]

[1, 2, 0, 1, 4]

f(3)

[0, 1, 0]

[0, 1, 0, 1]

[0, 1, 0, 1, 4]

1. python提供了对内存的垃圾收集机制，它将不用的内存释放到？

Python提供了对内存的垃圾收集机制，但是它将不用的内存放到内存池而不是返回给操作系统。

1. python 多线程

threading用于提供线程相关的操作，线程是应用程序中工作的最小单元。python当前版本的多线程库没有实现优先级、线程组，线程也不能被停止、暂停、恢复、中断。

threading模块提供的类：

　　Thread, Lock, Rlock, Condition, [Bounded]Semaphore, Event, Timer, local。

threading 模块提供的常用方法：

　　threading.currentThread(): 返回当前的线程变量。

　　threading.enumerate(): 返回一个包含正在运行的线程的list。正在运行指线程启动后、结束前，不包括启动前和终止后的线程。

　　threading.activeCount(): 返回正在运行的线程数量，与len(threading.enumerate())有相同的结果。

threading 模块提供的常量：

　　threading.TIMEOUT\_MAX 设置threading全局超时时间。

1. 长度为N的整形数组，找出乘积最大的两个数.（伪代码，时间复杂度）
2. 简述Session的原理，并说明session和cookies的区别。

会话（Session）跟踪是Web程序中常用的技术，用来跟踪用户的整个会话。常用的会话跟踪技术是Cookie与Session。Cookie通过在客户端记录信息确定用户身份，Session通过在服务器端记录信息确定用户身份。

1、cookie数据存放在客户的浏览器上，

session数据放在服务器上

2、cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗

考虑到安全应当使用session

3、session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能

考虑到减轻服务器性能方面，应当使用COOKIE

1. 求两个日期的差数，例如2007-2-5 2007-3-6 的日期差数

import datetime

def days(str1,str2):

    date1=datetime.datetime.strptime(str1[0:10],"%Y-%m-%d")

    date2=datetime.datetime.strptime(str2[0:10],"%Y-%m-%d")

    num=(date1-date2).days

    return num

1. 如何将服务器/home/xxx/a.txt拷贝到本地

scp root@×××.×××.×××.×××:/home/xxx/a.txt

1. 如何判断一个字符串是否以 “abc” 开头，以“abc”结尾。包含“abc”

用正则r’^abc.+abc.+abc$匹配’

1. 简述使用过的shell命令及功能

将当前目录下包含jack串的文件中，jack字符串替换为tom

sed -i "s/jack/tom/g" `grep "jack" -rl ./`

将某个文件中的jack字符串替换为tom

sed -i "s/jack/tom/g" test.txt

ort file1 对file1里面的内容进行排序

sort -u file1 去重排序

sort -r file1 降序排列

1. 如何获取文件“/home/user/xxxx/admin.html” 的文件名，文件类型

filepath,tempname = os.path.split(“/home/user/xxxx/admin.html”)

filename,fileclass = os.path.splittext(tempname)

其中filename就是文件名，fileclass就是文件类型

1. 现有一文件nba.txt内如入下

The 6-foot-6 guard dealt with hand pain during the final weeks of the season,ultimately sitting out the final 12 games of 2017-18.

Booker,who signed a five-year maximum extension during the offseason,is coming off his third NBA season in which he averaged 24.9 points,4.7 assists and 4.5 rebounds per game.The former 13th overall pick won the JBL Three-Point Contest during the 2018 NBA All-Star Weekend and recently participated in the USA Men's National Team minicamp in Las Vegas.

The Suns were hoping to take a step forward behind Booker and the offseason additions of No.1 overall pick Deandre Ayton,No.10 overall pick Mikal Bridges and free agent signee Trevor Ariza.Phoenix has not made the playoffs since 2010.

请统计文件中的每个单词所在的行数（Line number)，以及在该行的起始位置。

1. http POST和GET区别

（1）.GET请求，请求的数据会附加在URL之后，以?分割URL和传输数据，多个参数用&连接。URL的编码格式采用的是ASCII编码，而不是uniclde，即是说所有的非ASCII字符都要编码之后再传输。

POST请求：POST请求会把请求的数据放置在HTTP请求包的包体中。上面的item=bandsaw就是实际的传输数据。

因此，GET请求的数据会暴露在地址栏中，而POST请求则不会。

（2）.在HTTP规范中，没有对URL的长度和传输的数据大小进行限制。但是在实际开发过程中，对于GET，特定的浏览器和服务器对URL的长度有限制。因此，在使用GET请求时，传输数据会受到URL长度的限制。

对于POST，由于不是URL传值，理论上是不会受限制的，但是实际上各个服务器会规定对POST提交数据大小进行限制，Apache、IIS都有各自的配置。

1. .POST的安全性比GET的高。这里的安全是指真正的安全，而不同于上面GET提到的安全方法中的安全，上面提到的安全仅仅是不修改服务器的数据。比如，在进行登录操作，通过GET请求，用户名和密码都会暴露再URL上，因为登录页面有可能被浏览器缓存以及其他人查看浏览器的历史记录的原因，此时的用户名和密码就很容易被他人拿到了。除此之外，GET请求提交的数据还可能会造成Cross-site request frogery攻击。
2. 什么是http无状态协议，怎么解决？

http协议无状态中的【状态】到底指的是什么？！

先来看这句话的另外两个概念：（标准的http协议是无状态的，无连接的）

标准的http协议指的是不包括cookies, session，application的http协议，他们都不属于标准协议，虽然各种网络应用提供商，实现语言、web容器等，都默认支持它

无连接指的是什么

每一个访问都是无连接，服务器挨个处理访问队列里的访问，处理完一个就关闭连接，这事儿就完了，然后处理下一个新的

无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接

两种用于保持HTTP连接状态的技术就应运而生了，一个是Cookie，而另一个则是Session

1. http和https的区别和优缺点。

1. http 的URL 以http:// 开头，https以https:// 开头。

2. http 标准端口是80 ，https是443。

3.https 协议需要到ca申请证书，http不需要。

4.http 是超文本传输协议，信息是明文传输，https 则是具有安全性的ssl加密传输协议。

5.http 的连接很简单,是无状态的，https协议是由SSL+[http协议](https://www.baidu.com/s?wd=http%E5%8D%8F%E8%AE%AE&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)构建的可进行加密传输、身份认证的网络协议 要比[http协议](https://www.baidu.com/s?wd=http%E5%8D%8F%E8%AE%AE&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)安全。

优点：

1.  通过证书可以更信任服务器。

2.  更安全，防篡改。

缺点：

1.  https 需要证书。

2.  因为对传输进行加密，会一定程度增加cpu消耗。

3.  由于https 要还密钥和确认加密算法的需要，所以首次建立连接会慢一些。

4.  带宽消耗会增加。

1. Linux列出监听80端口的进程。
2. 请用python语言实现如下功能：

给定一个数组，其中该数组中的每个元素均为字符串，删除该数组中的空白字符串。

def delete\_list(lt):

a = list(filter(lambda x: x != "",lt)) #引用对象发生了变化，需要返回值

lt = list(a)

return lt #返回lt

lt = ["",1,2,3,""]

print(delete\_list(lt))

1. 什么是生成器（Generators）？适用的场景是什么？

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

1. 什么是异步调用？并请用一段简单的代码（不限语言）实现异步调用。

一个可以无需[等待](https://baike.baidu.com/item/%E7%AD%89%E5%BE%85/10246019)被调用函数的返回值就让操作继续进行的方法

异步(async)是相对于同步(sync)而言的，很好理解。

同步就是一件事一件事的执行。只有前一个任务执行完毕，才能执行后一个任务

from threading import Thread

from time import sleep

def async(f):

def wrapper(\*args, \*\*kwargs):

thr = Thread(target = f, args = args, kwargs = kwargs)

thr.start()

return wrapper

@async

def A():

sleep(20)

print "a function"

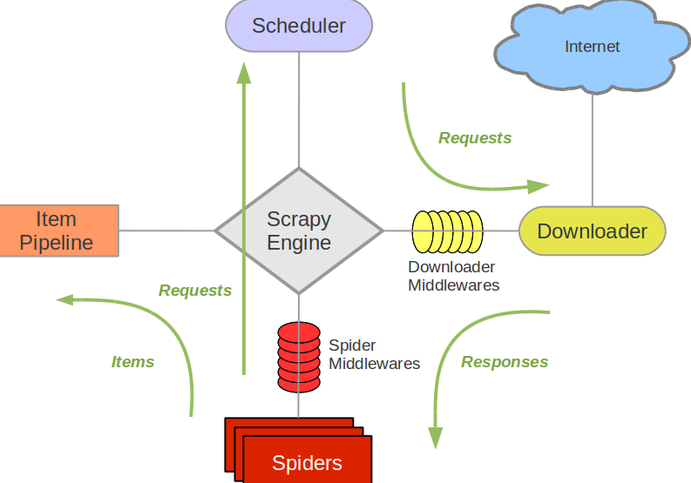
def B():

print "b function"

A()

B()

1. 简述Scrapy的框架（可做图说明），及各模块的基本功能。如果没有使用过Scrapy框架，那么挑选一个你使用过的框架进行说明。



Scrapy Engine(引擎): 负责Spider、ItemPipeline、Downloader、Scheduler中间的通讯，信号、数据传递等。

Scheduler(调度器): 它负责接受引擎发送过来的Request请求，并按照一定的方式进行整理排列，入队，当引擎需要时，交还给引擎。

Downloader（下载器）：负责下载Scrapy Engine(引擎)发送的所有Requests请求，并将其获取到的Responses交还给Scrapy Engine(引擎)，由引擎交给Spider来处理，

Spider（爬虫）：它负责处理所有Responses,从中分析提取数据，获取Item字段需要的数据，并将需要跟进的URL提交给引擎，再次进入Scheduler(调度器)，

Item Pipeline(管道)：它负责处理Spider中获取到的Item，并进行进行后期处理（详细分析、过滤、存储等）的地方.

Downloader Middlewares（下载中间件）：你可以当作是一个可以自定义扩展下载功能的组件。

Spider Middlewares（Spider中间件）：你可以理解为是一个可以自定扩展和操作引擎和Spider中间通信的功能组件（比如进入Spider的Responses;和从Spider出去的Requests）

1. 请实现如下功能（不限语言）：给定一个无序的数值序列，找出其中第N大的数值。

a = [123,13,13,312,123,3425,233]

b = sorted(a)

max\_N = b[-N]

1. Python 不支持的数据类型有（A）

A、char B、int C、float D、list

1. 关于Python中的复数，下列说法错误的是（B）

A、表示复数的语法是real+imagej

B、实部和虚部都是浮点数

C、虚部必须后缀j，且必须是小写

D、方法conjugate返回复数的共轭复数

1. 关于字符串下列说法错误的是（A）

A、字符应该视为长度为1的字符串

B、字符串以\0标志字符串的结束

C、既可以用单引号，也可以用双引号创建字符串

D、在三引号字符串中可以包含换行回车等特殊字符

1. 一下不能创建一个字典的语句是（D）

A、dict1={} B、dict2={3:5}

C、dict3={[1,2,3]:’uestc’}

D、dict4={(1,2,3):’uestc’}

1. Python是如何进行内存管理的

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会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。

1. \_\_new\_\_和\_\_init\_\_的区别

\_\_new\_\_:创建对象时调用，会返回当前对象的一个实例

\_\_init\_\_:创建完对象后调用，对当前对象的一些实例初始化，无返回值

1. Python中单下划线和双下划线的区别

什么是工作区，什么是暂存区？各自的撤销命令是什么？

两头下划线：Python类内置成员专用，区别用户自定义成员

单下划线：类的普通成员

双下划线：解析器自动转换为：\_类名\_\_成员名，代替原有成员，访问需要在原有成员名字前加上\_类名。

1.1工作区（Working Directory）

指的是在PC中能看得到的创建的一个管理仓库的目录，比如我的GitMyWarehouse文件夹就是一个工作区

1.2版本库（Repository）

工作区有一个隐藏目录.git，这个不算工作区，而是Git的版本库。

Git的版本库里存了很多东西，其中最重要的就是称为stage（或者叫index）的暂存区，还有Git为我们自动创建的第一个分支master，以及指向master的一个指针叫HEAD。

1. Git stash 命令的作用？如何使用？

使用Git过程中，经常遇到代码写到一半，突然一个bug需要紧急修复，或者有另一个新的任务需要优先完成，此时写到一半的代码怎么办？答案是使用git stash命令。

如果想要保存当前的修改，可以使用git stash或git stash save将修改储藏起来，保存在栈上。

如果想要查看储藏的东西，可以使用git stash list。

如果想要恢复一个储藏，可以使用的命令有git stash apply 和

git stash pop，区别在于后者恢复储藏后会把储藏从栈上移除掉，而前者不会，也就是list时还能不能看到记录的区别~

如果想要清除某个储藏，可以使用git stash drop stash@{x}

如果想清除全部，可以使用git stash clear

创建一个新分支并恢复储藏内容，然后继续工作：git stash branch <branchname>

1. 工作中实践过那些Git工作流，简单描述一下实际应用中的工作流程？

it工作流你可以理解为工作中团队成员遵守的一种代码管理方案，在Git中有以下几种工作流方案作为方案指导：

集中式工作流

功能开发工作流

Gitflow工作流

Forking工作流

下面针对性说下每个工作流可能使用到的场景和适用性：

集中式工作流

集中式工作流 | center

这种工作方式跟svn类似，它只有一个master分支，开发者会先把远程的仓库克隆到本地，之后的修改和提交都在本地操作，直到在某个合适的时间点将本地的代码合入到远程master。这种工作流比较适合小团队，因为小团队可能不会太多的协作和合流的动作。

1. 你有过失败的经历吗？你有实施过个人项目吗？
2. 是否有阅读源码的习惯？看过那些开源项目？你从中学到了什么？
3. 设计一个日志收集系统，要求无论重启，宕机都不能丢失数据，数据接收要求采用异步方式，不能阻塞其他服务
4. Python自带的数据结构分为可变和不可变的，下列选项中，不可变的是（B）。

A 数组 B 集合 C 字典 D集合

1. 下列哪个语句在Python中是非法的（B）。

A x=y=z=1 B x=(y=z+1) C x,y=y,x D x+=y

1. 关于Python内存管理，下列说法错误的是（B）

A 变量不必事先声明

B 变量无须先创建和赋值而直接使用

C 变量无须指定类型

D 可以使用del释放资源

1. 下面哪个不是Python合法的标识符（B）

A int32 B 40XL C self D name

1. Django常见的线上部署方式有哪几种

nginx + django（http方式）

nginx + uwsgi（http方式）

nginx + uwsgi（socket方式）

1. Django对数据查询结果排序怎么做，降序怎么做

在你的 Django 应用中，你或许希望根据某字段的值对检索结果排序，比如说，按字母顺序。 那么，使用 order\_by() 这个方法就可以搞定了。

1. MySQL中varchar和char的区别以及varchar（50）中50代表的含义

CHAR列的长度固定为创建表时声明的长度。长度可以为从0到255的任何值。当保存CHAR值时，在它们的右边填充空格以达到指定的长度。当检索到CHAR值时，尾部的空格被删除掉。在存储或检索过程中不进行大小写转换。

VARCHAR列中的值为可变长字符串。长度可以指定为0到65535之间的值。VARCHAR的最大有效长度由最大行大小和使用的字符集确定。在MySQL

4.1之前的版本，VARCHAR(50)的“50”指的是50字节（bytes）。如果存放UTF8汉字时，那么最多只能存放16个（每个汉字3字节）。从MySQL

4.1版本开始，VARCHAR(50)的“50”指的是50字符（character），无论存放的是数字、字母还是UTF8汉字（每个汉字3字节），都可以存放50个。

CHAR和VARCHAR类型声明的长度表示保存的最大字符数。例如，CHAR(30)可以占用30个字符。对于MyISAM表，推荐CHAR类型；对于InnoDB表，推荐VARCHAR类型。另外，在进行检索的时候，若列值的尾部含有空格，则CHAR列会删除其尾部的空格，而VARCHAR则会保留空格

1. 请简述项目中优化sql语句执行的效率的方法

（1）选择最有效率的表名顺序

（2）WHERE子句中的连接顺序

（3）SELECT子句中避免使用‘\*’

（4）用Where子句替换HAVING子句

（5）通过内部函数提高SQL效率

（6）避免在索引列上使用计算。

（7）提高GROUP BY 语句的效率, 可以通过将不需要的记录在GROUP BY 之前过滤掉。

1. git pull 和git fetch的区别

commit-id：在每次本地工作完成后，都会做一个git commit 操作来保存当前工作到本地的repo， 此时会产生一个commit-id，这是一个能唯一标识一个版本的序列号。 在使用git push后，这个序列号还会同步到远程仓库。

有了以上的概念再来说说git fetch

git fetch：这将更新git remote 中所有的远程仓库所包含分支的最新commit-id, 将其记录到.git/FETCH\_HEAD文件中

git fetch更新远程仓库的方式如下：

git fetch origin master:tmp

//在本地新建一个temp分支，并将远程origin仓库的master分支代码下载到本地temp分支

git diff tmp

//来比较本地代码与刚刚从远程下载下来的代码的区别

git merge tmp

//合并temp分支到本地的master分支

git branch -d temp

//如果不想保留temp分支 可以用这步删除12345678

（1）如果直接使用git fetch，则步骤如下：

创建并更新本 地远程分支。即创建并更新origin/xxx 分支，拉取代码到origin/xxx分支上。

在FETCH\_HEAD中设定当前分支-origin/当前分支对应，如直接到时候git merge就可以将origin/abc合并到abc分支上。

（2）git fetch origin

只是手动指定了要fetch的remote。在不指定分支时通常默认为master

（3）git fetch origin dev

指定远程remote和FETCH\_HEAD，并且只拉取该分支的提交。

git pull : 首先，基于本地的FETCH\_HEAD记录，比对本地的FETCH\_HEAD记录与远程仓库的版本号，然后git fetch 获得当前指向的远程分支的后续版本的数据，然后再利用git merge将其与本地的当前分支合并。所以可以认为git pull是git fetch和git merge两个步骤的结合。

git pull的用法如下：

git pull <远程主机名> <远程分支名>:<本地分支名>

//取回远程主机某个分支的更新，再与本地的指定分支合并。12

因此，与git pull相比git fetch相当于是从远程获取最新版本到本地，但不会自动merge。如果需要有选择的合并git fetch是更好的选择。效果相同时git pull将更为快捷。

1. Git merge 和git rebase 的区别

merge 与 rebase 的区别

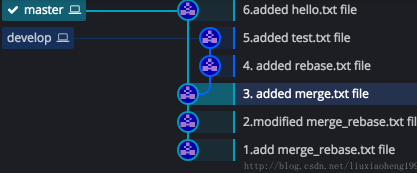
merge

现在假设我们有一个主分支 master 及一个开发分支 deve，仓库历史就像这样：

初始仓库历史

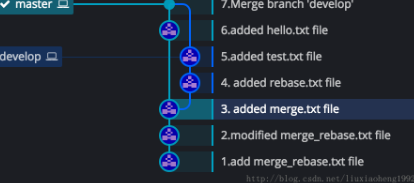
现在如果在 master 分支上 git merge deve：Git 会自动根据两个分支的共同祖先即 e381a81 这个 commit 和两个分支的最新提交即 8ab7cff 和 696398a 进行一个三方合并，然后将合并中修改的内容生成一个新的 commit，即下图的 78941cb

merge 合并图



rebase

rebase 是什么情况呢？还是一个初始的仓库历史图：



rebase初始仓库历史

如果是在 master 分支上 git rebase deve：Git 会从两个分支的共同祖先 3311ba0 开始提取 master 分支（当前所在分支）上的修改，即 85841be、a016f64 与 e53ec51，再将 master 分支指向 deve 的最新提交（目标分支）即 35b6708 处，然后将刚刚提取的修改依次应用到这个最新提交后面。操作会舍弃 master 分支上提取的 commit，同时不会像 merge 一样生成一个合并修改内容的 commit，相当于把 master 分支（当前所在分支）上的修改在 deve 分支（目标分支）上原样复制了一遍

1. 如何回滚代码到指定的分支

这个是重点要说的内容，过程比本地回滚要复杂

应用场景：自动部署系统发布后发现问题，需要回滚到某一个commit，再重新发布

原理：先将本地分支退回到某个commit，删除远程分支，再重新push本地分支

操作步骤：

1、git checkout the\_branch

2、git pull

3、git branch the\_branch\_backup //备份一下这个分支当前的情况

4、git reset --hard the\_commit\_id //把the\_branch本地回滚到the\_commit\_id

5、git push origin :the\_branch //删除远程 the\_branch

6、git push origin the\_branch //用回滚后的本地分支重新建立远程分支

7、git push origin :the\_branch\_backup //如果前面都成功了，删除这个备份分支

如果使用了gerrit做远程代码中心库和code review平台，需要确保操作git的用户具备分支的push权限，并且选择了 Force Push选项（在push权限设置里有这个选项）

另外，gerrit中心库是个bare库，将HEAD默认指向了master，因此master分支是不能进行删除操作的，最好不要选择删除master分支的策略，换用其他分支。如果一定要这样做，可以考虑到gerrit服务器上修改HEAD指针。。。不建议这样搞

1. 在CentOS7.2中，用一句话将所有的test.py进程全部杀死

killall -9 test.py

1. 用python完成两个数据的横向合并，并以Frame1为主表，并计算总成绩（数学+语文）

Frame1:

|  |  |
| --- | --- |
| 学号 | 数学成绩 |
| ST01 | 57 |
| ST02 | 78 |
| ST03 | 65 |
| ST04 | 87 |
| ST05 | 91 |
| ST06 | 77 |

|  |  |  |
| --- | --- | --- |
| 学号 | 数学成绩 | 英语成绩 |
| ST02 | 58 | 77 |
| ST05 | 85 | 69 |
| ST01 | 74 | 80 |
| ST03 | 82 | 71 |
| ST04 | 89 | 88 |

Frame2:

Import\_\_\_\_\_\_\_\_\_

df=\_\_\_\_\_\_\_\_(Frame,Frame2,on=’学号’,\_\_\_\_\_\_\_\_\_\_\_\_\_)

df\_new=df.replace(np.nan，0） #或用df\_new=df.fillna(0)

df\_new[‘总成绩’]=df\_new[‘数学成绩’]+df\_new[‘语文成绩’]+df\_new[‘英语成绩’]

1. class Job(models.Model):

Name = models.Charfield(max\_length=50)

type = models.foreignkey(JobType,related\_name=’jobs’)

city = models.Charfield(max\_length=30)

experience = models.Charfield(max\_length=15)

cducation = models.Charfield(max\_length=10)

salary = models.Charfield(max\_length=15)

public\_time = models.DateTimeField()

def \_\_str\_\_(self):  
 return self.name

class JobType(models.Model):

name = models.Charfield(max\_length=10)

description = models.Charfield(max\_length=100,null=True,blank=True)

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

Return self.name

写出Django查询语句，找出city以‘北京’或者‘上海’开头的Job:

查询JobType的name为‘PyThon’之外的所有Job

1. Django restfull framework中view有哪几种，以及他们的关系是？各自有什么应用场景

1. django View

  首先，我们使用django自带的view，获取一个课程的列表：

# drf是通过json的格式进行数据交互的，所以这里也返回json数据

import json

from django.views.generic.base import View

from django.core import serializers

from django.http import HttpResponse,JsonResponse

from .models import Course

class CourseListView(View):

def get(self, request):

"""

通过django的view实现课程列表页

"""

courses = Course.objects.all()[:10]

json\_data = serializers.serialize('json', courses)

json\_data = json.loads(json\_data)

return JsonResponse(json\_data, safe=False)

2. APIView

接下来，我们用APIView来实现

from rest\_framework.views import APIView

from rest\_framework.response import Response

# 这个serializers是在其他文件自定义的，这里对这个不进行过多介绍

from .serializers import CourseSerializer

class CourseListView(APIView):

def get(self, request, format=None):

"""

通过APIView实现课程列表页

"""

courses = Course.objects.all()

serializer = CourseSerializer(courses, many=True)

return Response(serializer.data)

  在APIView这个例子中，调用了drf本身的serializer以及Response方法。

  APIView对django本身的View进行封装，从上述的代码，这样分析，两者的差别看起来不是很大，但实际中APIView做了很多东西，它定义了很多属性与方法，举几个例子

# 这三个是常用的属性

authentication\_classes : 用户登录认证方式，session或者token等等

permission\_classes : 权限设置，是否需要登录等

throttle\_classes : 限速设置，对用户进行一定的访问次数限制等等。

  到这里，可能还不能体现drf通过view的强大之处，那么接下来的GenericAPIView就展示了它强大的功能。

3. GenericAPIView

from rest\_framework import mixins

from rest\_framework import generics

class CourseListView(mixins.ListModelMixin, generics.GenericAPIView):

"""

课程列表页

"""

queryset = Course.objects.all()

serialize\_class = CourseSerializer

def get(self, request, \*args, \*\*kwargs):

# list方法是存在于mixins中的，同理，create等等也是

# GenericAPIView没有这些方法！

return self.list(request, \*args, \*\*kwargs)

在这个例子中，继承了mixins中的ListModelMixin，在get( )方法中，调用了它的list( )方法，list方法会返回queryset的json数据。这里对mixins不进行过多的介绍。

GenericAPIView对APIView再次封装，实现了强大功能：

加入queryset属性，可以直接设置这个属性，不必再将实例化的courses，再次传给seriliazer,系统会自动检测到。除此之外，可以重载get\_queryset()，这样就不必设置’queryset=\*’，这样就变得更加灵活，可以进行完全的自定义。

加入serializer\_class属性与实现get\_serializer\_class()方法。两者的存在一个即可，通过这个，在返回时，不必去指定某个serilizer

设置过滤器模板：filter\_backends

设置分页模板：pagination\_class

加入 lookup\_field=”pk”，以及实现了get\_object方法: 这个用得场景不多，但十分重要。它们两者的关系同1，要么设置属性，要么重载方法。它们的功能在于获取某一个实例时，指定传进来的后缀是什么。

  举个例子，获取具体的某个课程，假设传进来的ulr为：http://127.0.0.1:8000/course/1/，系统会默认这个1指的是course的id。那么，现在面临一个问题，假设我定义了一个用户收藏的model，我想要知道我id为1的课程是否收藏了，我传进来的url为：http://127.0.0.1:8000/userfav/1/,系统会默认获取userfav的id=1的实例，这个逻辑明显是错的，我们需要获取course的id=1的收藏记录，所以我们就需要用到这个属性或者重载这个方法 lookup\_field=”course\_id”.

  在generics除了GenericAPIView还包括了其他几个View: CreateAPIView、ListAPIView、RetrieveAPIView、ListCreateAPIView···等等，其实他们都只是继承了相应一个或多个mixins和GenericAPIView，这样，有什么好处？我们看一下同样一个例子的代码：

class CourseListView(ListAPIView):

"""

课程列表页

"""

queryset = Course.objects.all()

serialize\_class = CourseSerializer

4.GenericViewSet

GenericAPIView的不足之处

  既然GenericAPIView以及它相关的View已经完成了许许多多的功能，那么还要ViewSet干嘛！

  首先，我们思考一个问题，同样上面的例子，我们在功能上，要获取课程的列表，也要获取某个课程的具体信息。那么怎么实现，按照GenericAPIView，我们可以这样实现：

class CourseView(ListAPIView，RetrieveAPIView):

＃　只需要在上面的基础上，再继承RetrieveAPIView就ok了。

queryset = Course.objects.all()

serialize\_class = CourseSerializer

  但这样实现有一个问题，关于serialize\_class，显然，当获取课程列表时，只需要传回去所有课程的简要信息，如课程名字，老师，封面等等，但当获取课程的具体信息，我们还要将他们的章节以及相关下载资料（很明显，章节是另外一个model，有一个外键指向course），这些信息会很多，在获取课程列表，将这些传回去显然是不理智的。那么，还需要再定义一个CourseDetailSerializer，在get /courses/的时候，使用CourseSerializer，在get /courses/id/的时候，使用CourseDetailSerializer。

1. Django配置文件中，对于中间件的执行顺序

Settings.MIDDLEWARE\_CLASSES={

“Middleware1”,

“Middleware2”,

“Middleware3”,}s

1. 请问下面代码中与数据库发生了几次交互？发生再那几步？

q = Entry.objects.filter(headline\_\_startwith=’What’)

q = q.filter(pub\_dete\_\_lte=datetime.date.today())

q = q.exclude(body\_text\_\_\_icontains=’food’)

print(q)

1. 请问再使用统一redis服务且db相同的情况下，下面代码会输出什么？

form django.core.cache import cache # django对redis的封装

Import redis # 原生redis

cache.set(“cae9ad331b9206a1b5594813b509e1003”,”test”,timeout=10)

print redis.get(“cae9ad331b9206a1b5594813b509e1003”)

1. 使用Flask sqlalchemy时如何正确的设置autocommit？

SQLALCHEMY\_DATABASE\_URI = ('mysql+pymysql://username:passwrod@ip:prot/database?charset=utf8&autocommit=true')

1. 面向对象编程中有组合和继承的方法实现新的类，假设我们手头只有“栈”类，请用“组合”的方式使用“栈”（LIFO）来实现“队列”（FIFO），完成以下代码（如果认为不需要添加代码可以不写）。

#!/usr/bin/env python

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

#stack 类实现了栈，并提供pop和push方法。

import stack

Class Queue(object):

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

self.s1 = stack()

self.s2 = stack()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ #请在此处添加代码

def pop(self):

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ #请在此处添加代码

def push(self,e):

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ #请在此处添加代码

1. 试编写算法，不使用额外的变量，交换两个数。

a,b = b,a

1. 试实现快速排序，语言不限

void QuickSort(int\* array,int left,int right)

{

assert(array);

if(left >= right)//表示已经完成一个组

{

return;

}

int index = PartSort(array,left,right);//枢轴的位置

QuickSort(array,left,index - 1);

QuickSort(array,index + 1,right);

}

1. 什么是lambda函数？它有什么好处？（有python经验者回答，其他人跳过）

lambda函数是匿名函数；使用lambda函数能够创建小型匿名函数。这种函数得名于省略了用def声明函数的标准步骤

1. 试描述下面程序的运行结果（写在右边空白处）

from vcScript import \*

def OnSignal(signal):

Pass

def OnRun():

servo = comp.findBehaviour(‘Servo’)

While True:

servo.moveJoint(0,800.0)

delay(5)

servo.moveJoint(0,0.0)

def JointValues(signal):

jointX = comp.Joint\_X

jointY = comp.Joint\_Y

Print”Joint X is %s and Joint Y is %s” % (jointX,jointY)

comp = getComponent()

signal = comp.findBehaviour(‘BooleanSignal’)

signal.OnValueChange = JointValues

1. 常用的js框架有哪些？

Vue.js

angular.Js

React

1. Session和Cookie的区别？

cookie 和session 的区别：

1、cookie数据存放在客户的浏览器上，session数据放在服务器上。

2、cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗

考虑到安全应当使用session。

3、session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能

考虑到减轻服务器性能方面，应当使用COOKIE。

4、单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。

1. 常用的Mysql表引擎及其特点？

数据库的存储引擎是对表进行某种设置，存储引擎决定该表的数据存储方式、数据更新方式、数据查询性能以及是否支持索引、事务处理性能等。MySQL常用的几种引擎有：MyISAM、InnoDB、MEMORY、CSV、BLACKHOLE、ARCHIVE、PERFORMANCE\_SCHEMA、Merge等

MyISAM

MyIASM是MySQL默认的引擎，使用一种表格锁定的机制来优化多个并发的读写操作。需要经常运行OPTIMIZE TABLE命令，来恢复被更新机制所浪费的空间，否则碎片也会随之增加，最终影响数据访问性能。

优点：1、MyISAM表是独立于操作系统的，这说明可以轻松地将其从Windows服务器移植到Linux服务器。2、MyISAM存储引擎在查询大量数据时非常迅速，这是它最突出的优点。3、另外进行大批量插入操作时执行速度也比较快。4、单独保存了表的行数。

缺点：1、MyISAM表没有提供对数据库事务的支持。2、不支持行级锁和外键。3、不适合用于经常UPDATE（更新）的表，效率低。

InnoDB

优点：1、提供事物支持。2、提供系统崩溃修复能力。3、支持多版本并发控制（即MVCC Multi-Version Concurrency Control）的行级锁，由于锁粒度小，写操作和更新操作并发高、速度快。适合更新频繁的表。4、支持自增长列。5、支持外键。6、适合于大容量数据库系统，支持自动灾难恢复

1. Char和Vachar的区别？哪个更占磁盘空间？

char和varchar类型声明长度表示用户想保存的最大字符数，其中char(M)定义的列的长度为固定的，M的取值可以0-255之间，当保存char值时，在它们的右边填充空格以达到指定的长度。当检索到char值时，尾部的空格被删除掉（如下图）。在存储或检索过程中不进行大小写转换。char存储定长数据很方便，char字段上的索引效率很高。

          varchar(M)定义的列的长度是可变长度字符串，在MySQL5.0以上的版本中，varchar的数据类型长度支持到了65535，因为起始位和结束位占去了3个字节，所以其整体最大长度为65532字节（varchar的最大有效长度由最大行大小和使用的字符集确定）。

          除此之外，与char比，varchar值保存时只保存需要的字符数，另加一个字节来记录长度（长度超过255时需要2个字节）。

         同样在char和varchar尾部加空格，检索时char类型后的被删掉，而varchar类型的保存。

1. Mysql字段定义如下Float(6,2)，其表示的整数和小数位各是多少？

整数4位，小数2位

1. Mysql中innoDB的索引结构是什么？为什么不使用MD5、uuid作为主键？

既然我们需要全局唯一的id，为什么不用GUID（又成为UUID）呢？它能保证任何地点、任何地方都不会有重复的数据出现。但这也是问题所在，GUID本身太大了，在MySQL中进行索引非常慢。我们为了保证MySQL能够快速响应，把所有需要查询的数据都建立了索引，并且我们只通过索引检索，因此索引的大小和类型非常关键，一定要适合快速索引。如果我们不能把所有的索引放进内存，那就很难保证数据库能够快速响应。

1. 是否遇到过数据库执行语句发生死锁？产生的原因？解决办法？

**死锁的第一种情况**

一个用户A 访问表A(锁住了表A),然后又访问表B；另一个用户B 访问表B(锁住了表B)，然后企图访问表A；这时用户A由于用户B已经锁住表B，它必须等待用户B释放表B才能继续，同样用户B要等用户A释放表A才能继续，这就死锁就产生了。

解决方法：

这种死锁比较常见，是由于程序的BUG产生的，除了调整的程序的逻辑没有其它的办法。仔细分析程序的逻辑，对于数据库的多表操作时，尽量按照相同的顺序进 行处理，尽量避免同时锁定两个资源，如操作A和B两张表时，总是按先A后B的顺序处理， 必须同时锁定两个资源时，要保证在任何时刻都应该按照相同的顺序来锁定资源。

**死锁的第二种情况**

用户A查询一条纪录，然后修改该条纪录；这时用户B修改该条纪录，这时用户A的事务里锁的性质由查询的共享锁企图上升到独占锁，而用户B里的独占锁由于A 有共享锁存在所以必须等A释放掉共享锁，而A由于B的独占锁而无法上升的独占锁也就不可能释放共享锁，于是出现了死锁。这种死锁比较隐蔽，但在稍大点的项 目中经常发生。如在某项目中，页面上的按钮点击后，没有使按钮立刻失效，使得用户会多次快速点击同一按钮，这样同一段代码对数据库同一条记录进行多次操 作，很容易就出现这种死锁的情况。

解决方法：

1、对于按钮等控件，点击后使其立刻失效，不让用户重复点击，避免对同时对同一条记录操作。  
2、使用乐观锁进行控制。乐观锁大多是基于数据版本（Version）记录机制实现。即为数据增加一个版本标识，在基于数据库表的版本解决方案中，一般是 通过为数据库表增加一个“version”字段来实现。读取出数据时，将此版本号一同读出，之后更新时，对此版本号加一。此时，将提交数据的版本数据与数 据库表对应记录的当前版本信息进行比对，如果提交的数据版本号大于数据库表当前版本号，则予以更新，否则认为是过期数据。乐观锁机制避免了长事务中的数据 库加锁开销（用户A和用户B操作过程中，都没有对数据库数据加锁），大大提升了大并发量下的系统整体性能表现。Hibernate 在其数据访问引擎中内置了乐观锁实现。需要注意的是，由于乐观锁机制是在我们的系统中实现，来自外部系统的用户更新操作不受我们系统的控制，因此可能会造 成脏数据被更新到数据库中。  
3、使用悲观锁进行控制。悲观锁大多数情况下依靠数据库的锁机制实现，如Oracle的Select … for update语句，以保证操作最大程度的独占性。但随之而来的就是数据库性能的大量开销，特别是对长事务而言，这样的开销往往无法承受。如一个金融系统， 当某个操作员读取用户的数据，并在读出的用户数据的基础上进行修改时（如更改用户账户余额），如果采用悲观锁机制，也就意味着整个操作过程中（从操作员读 出数据、开始修改直至提交修改结果的全过程，甚至还包括操作员中途去煮咖啡的时间），数据库记录始终处于加锁状态，可以想见，如果面对成百上千个并发，这 样的情况将导致灾难性的后果。所以，采用悲观锁进行控制时一定要考虑清楚。

**死锁的第三种情况**

如果在事务中执行了一条不满足条件的update语句，则执行全表扫描，把行级锁上升为表级锁，多个这样的事务执行后，就很容易产生死锁和阻塞。类似的情 况还有当表中的数据量非常庞大而索引建的过少或不合适的时候，使得经常发生全表扫描，最终应用系统会越来越慢，最终发生阻塞或死锁。

解决方法：

SQL语句中不要使用太复杂的关联多表的查询；使用“执行计划”对SQL语句进行分析，对于有全表扫描的SQL语句，建立相应的索引进行优化

1. 常用的git命令有哪些，具体含义是什么?

一 设置配置信息

config 配置有三个级别， --global 全局配置；--system 系统配置； --local 存储库配置，使用 --配置文件 可以定位到文件

1.1 设置配置信息  (config只需要配置一次)

git config --global user.name  "baicun"      # "baicun"就是你的用户名

git config --global user.name "javaman\_baicun@126.com        # "javaman\_baicun@126.com"就是你的用户名

1.2 查看配置

git config --system --list  # 查看系统配置

git config --global --list    # 查看全局配置

git config --local --list      # 查看存储库配置,注意需要切换目录到你的仓库下，在执行查看命令

git config  # 查看 Git 信息，得到帮助 ,见下图。对于我们使用具体命令很重要

.3 修改配置

git config --global --replace-all user.name "javaman\_baicun@126.com"   # "javaman\_baicun@126.com"是你修改后的名字

1.4 删除配置

首先新增一个配置项：

git config --local --add alibaba.ss "beijing"

开始删除

$ git config --local --unset alibaba.ss    # 删除配置项 alibaba.ss

二 代码管理

2.1 上传文件

2.1.1 先GitHub 上创建仓库

创建完成以后，保留远程仓库URL：https://github.com/baicun/city.git

2.1.2 本地目录初始化

git init     # git 初始化

初始化以后，会在我们的目录下生成一个 .git 文件。默认是隐藏的，可以取消隐藏查看。该目录下就存放着 --local 的配置信息

通过命令查看该目录下状态

git status     # 查看状态

可以看到，文件目录下的文本文档显示红色的，说明没有加到我们的版本控制中

2.1.3 添加版本控制

git add beijing.txt    # 添加一个文件，如果是多个，可以使用 "." 或 "\*"

git commit -m "first commit"     #  -m：message,后面跟提交的说明，也就是日后的日志信息

2.1.5 提交到远程仓库

远程仓库建立别名,方便日后传输

 git remote add city https://github.com/baicun/city.git

说明：remote 远程 ；city 别名，自己定义，默认是 origin；https://github.com/baicun/city.git   远程仓库的地址

上传至远程仓库

git push -u city master

说明： city  上面我们起的别名；master 本地仓库名称

此时，可以在我们的GitHub上查看了。

说明它变了，不是我们之前的那个 beijing.txt 了，

1. 实现二分査找

int bsearch(int array[], int low, int high, int target)  
{  
 if (low > high) return -1;  
   
 int mid = (low + high)/2;  
 if (array[mid]> target)  
 return binarysearch(array, low, mid -1, target);  
 if (array[mid]< target)  
 return binarysearch(array, mid+1, high, target);  
   
 //if (midValue == target)  
 return mid;  
}

1. 实现任意-种排序算法（快排，堆排，冒泡等〉，并评估其时间复杂度.

每次从待排序的数据中选取最小（最大）的一个元素，存放到序列的起始位置，知道全部排完

[1,3,5,6,8,9,2,3,5,2,1]

#选择排序

#[1,2,3]

def selectSort(list):

for i in range(len(list)-1,0,-1):

maxone = 0

for j in range(1,i+1):

if list[j] > list[maxone]:

maxone = j

temp = list[i]

list[i] = list[maxone]

list[maxone] = temp

return list

list = [33,44,55,43,23,3,4,5,6,7,8,777,3]

print(selectSort(list))

1. -棵二叉树，逐层打印每一层节点?例如下图,打印结果为2,1,3,5,6,7

2

/ \

1 3

/ \ \

5 6 7

1. 用户登陆时，通过http协议JI传用户名和密码，服务器校验正确后，返回是否登陆成功的结果。在此过程中，如何尽可能保证用户数据安全性？

答案：在关系到用户隐私的时候，要时刻遵循两个原则：

1、任何应用程序都不能在本地存储与安全相关的用户信息

2、任何应用程序在向服务器传递数据的时候，都不能直接传递与安全相关的用户信息。

要想让用户信息安全，就必须对其进行加密，让别人即便是拿到了安全信息，摆在眼前的也是一串乱码，没有半点用处

MD5是一种常用的加密方法，它是一种散列函数，利用MD5对用户信息进行加密，会增加用户信息安全性。【self.passwordText.text nd5String】;

1. Server?系统部署在多台机器上，运行期间，会产生大量的用户操作日志及错误日志。如 何设计，能便于后续对用户操作日志的査找及统计？以及错误日志的及时报警？

答案：1.  尽可能完整。 每一条错误日志都完整描述了： 什么场景下发生了什么错误， 什么原因（或者哪些可能原因）， 如何解决（或解决提示）；

2.  尽可能具体。 比如 NC 资源不足， 究竟具体指什么资源不足， 是否可以通过程序直接指明； 通用错误，比如 VM NOT EXIST ， 要指明在什么场景下发生的，可能便于后续统计的工作。

3.  尽可能直接。 最理想的错误日志应该让人在第一直觉下能够知道是什么原因导致，该怎么去解决，而不是还要通过若干步骤去查找真正的原因。

4.  将已有经验集成直接到系统中。 所有已经解决过的问题及经验都要尽可能以友好的方式集成到系统中，给新进人员更好的提示，而不是埋藏在其他地方。

5.  排版要整洁有序， 格式统一化规范化。 密密麻麻、随笔式的日志看着就揪心， 相当不友好， 也不便于排查问题。

6.  采用多个关键字唯一标识请求，突出显示关键字： 时间、实体标识（比如vmname）、操作名称。

1. 以下代码是否有误？如果无误请直接写出运行结果，如果有误请改正之后在写运行结果

Def gen():

X=0

While True:

X = yield x\*x

a = gen()

for i in range(5):

print(a.send(i))

1. 解释一下魔法方法的作用，一级他们在什么情况下会被触发？
2. \_\_call\_\_：会在 调用非静态方法时 触发

(2)\_\_iter\_\_：会在迭代的时候触发

(3)\_\_enter\_\_：。当with语句在开始运行时，会在上下文管理器对象上调用 \_\_enter\_\_ 方法。

(4)\_\_exit\_\_：with语句运行结束后，会在上下文管理器对象上调用 \_\_exit\_\_ 方法

1. python 标准库中，在多个进程间共享数据的方法有哪些？他们是 process-safe的 什么玩意儿？

有界函数是包在一个类中，并且只有当类被实例化之后才能使用的函数，它的界就是这个实例。我们常常把这些函数称为类方法。例如以self为参数的类方法。

无界函数可以是没有被包在类中的函数，也可以是类中的静态方法，它们跟类是独立的。如类中的静态方法，它即使在某个类中被定义，但是不能访问类中的参数和其他方法。

python多进程的机制应该是把每个进程要调用的方法和传入的参数（如上面例子中的ProcessWorker）编译然后打包，然后复制到每个进程中执行。如果输入的是一个有界函数，那么它的参数应该是它所属的类（包括参数和方法），但是这是无法获得的，而且类属性和方法可能会有坑，导致难以打包。所以python限定了多进程要调用的函数不能是类方法。

我们要把多进程调用的函数放到类外面，或者变成静态函数。但是静态函数的话不能被所属的类的方法调用（self.ProcessWorker的形式），需要在外部调用，如mc=MyClass(),mc.ProcessWorker来调用，或者MyClass().ProcessWorker来调用。

1. 常用的I/O多路复用(I/O multiplexing)技术有哪几种？请阐明他们各自的特点及 。

关于I/O多路复用(又被称为“事件驱动”)，首先要理解的是，操作系统为你提供了一个功能，当你的某个socket可读或者可写的时候，它可以给你一个通知。

这样当配合非阻塞的socket使用时，只有当系统通知我哪个描述符可读了，我才去执行read操作，可以保证每次read都能读到有效数据而不做纯返回-1和EAGAIN的无用功。写操作类似。操作系统的这个功能通过select/poll/epoll/kqueue之类的系统调用函数来使用，这些函数都可以同时监视多个描述符的读写就绪状况，这样，多个描述符的I/O操作都能在一个线程内并发交替地顺序完成，这就叫I/O多路复用，这里的“复用”指的是复用同一个线程。

以select和tcp socket为例，所谓可读事件，具体的说是指以下事件：

1 socket内核接收缓冲区中的可用字节数大于或等于其低水位SO\_RCVLOWAT;2 socket通信的对方关闭了连接，这个时候在缓冲区里有个文件结束符EOF，此时读操作将返回0；

3 监听socket的backlog队列有已经完成三次握手的连接请求，可以调用accept；

4 socket上有未处理的错误，此时可以用getsockopt来读取和清除该错误。所谓可写事件，则是指：

1 socket的内核发送缓冲区的可用字节数大于或等于其低水位SO\_SNDLOWAIT；

2 socket的写端被关闭，继续写会收到SIGPIPE信号；

3 非阻塞模式下，connect返回之后，发起连接成功或失败；

4 socket上有未处理的错误，此时可以用getsockopt来读取和清除该错误。

1. int，float，str,tuple,list,set,dict前面这些类型那些可以做dict的key,哪些可以做dict的value？

答：str，int，float,tuple可以当成dict的key

int，float，str,tuple,list,set,dic可以当成value

1. 请写出一段Python代码实现删除一个list里面的重复元素

ids = [1,2,3,3,4,2,3,4,5,6,1]

news\_ids = []

for id in ids:

if id not in news\_ids:

news\_ids.append(id)

print news\_ids

或用set

ids = [1,4,3,3,4,2,3,4,5,6,1]

ids = list(set(ids))

1. 写出结果为【1,4,9,16,25，...，100】的列表推导式

答：[x\*x for x in range(1，100)]

1. 写出结果为【1:1,2:4，3:9,4:16,5:25，...，10:100】的字典推导式
2. 下面代码会输出什么？

def f(x,l=[]):

for i in range(x) :

l.append(i\*i)

print l

f(2)

f(3,[3,2,1])

f(3)

1. a = [1,2,3,4,5],a[::2]=? a[-2:] = ?
2. 写出print结果

class Person:

name=[]

p1 = Person()

p2 = Person()

p1.name.append(1)

print p1.name

print p2.name

1. 用Python匹配HtmlHTML tag的时候， <.\*>和<.\*?>有什么区别？

答：<.\*>这种匹配称作贪心匹配 <.\*?>称作非贪心匹配

1. 列举至少5个PEP8标准。

答：一、代码编排

1、缩进。4个空格的缩进，不使用Tap,更不能混合使用Tap和空格

2、每行最大长度79，换行可以使用反斜杠，最好使用圆括号。换行点要在操作符的后边敲回车

二、文档编排

1、不要在一句import中多个库，比如import 欧式，sys

三、空格的使用

总体原则，避免不必要的空格

四、注释

1、总体原则，错误的注释不如没有注释，当一段代码发生变化时，第一件事就是要修改注释！

2、注释必须使用英文，最好是完整的句子，首字母要大写的。

3、避免无谓的注释

五、命名规范

类的方法第一个参数必须是self，而静态方法第一个参数必须是cls

遵循python的变量的命名规范。

六、编码建议

1、编码中考虑是他python实现的效率等问题，比如运算符‘+’在Jpython中效率却非常低，采用。join()的方式

2、字符串不要以空格收尾

3、使用startwith() and endswith() 代替切片进行序列前缀或后缀的检查

4、使用isinstance()比较对象的类型：

5、判断序列空与不空，if not 条件,优于if 条件

1. 举例说明，Python中使用三元运算符的方式。

答：为真时结果 if 判断条件为真 else判断条件为假时的结果   
x = 5 if y=1 else 6   
就是说y=1则x=5，否则x=6

1. 距离说明Python中字典更新的几种方式。

答：1.创建字典

>>> phonebook={'Alice':'2897','Alan':'0987','Jery':'6754'}

2.dict函数

>>> items=[('name','Gumby'),('age',42)]>>> d=dict(items)>>> d

{'age': 42, 'name': 'Gumby'}>>> d['name']'Gumby'

>>> d=dict(name='July',age=24)>>> d

{'age': 24, 'name': 'July'}

还有cooy，clear，deepcopy

1. Nginx服务器有什么作用，如果需要将不同的请求路径转发不同的…需要怎样实现？

同一Nginx服务器，是指某台安装了Nginx软件的电脑，使用某个端口，比如默认的web端口作为服务器，对外来各种不同的访问只要是指向该台电脑的该端口，就认为是同一Nginx服务器；来自不同的域名请求，这个好理解，就比如 www.xxx.com，www.yyy.cn，www.kkk.net 等等就是不同的域名请求；那怎么实现设定的目标呢？其实就是把各个不同请求的域名通过IP等方式设置指向同一个Nginx服务器，这样就这个Nginx服务就可用接受来自不同域名的请求了，因Nginx有对应转发功能，然后通过合理的服务器配置，即可实现对来自不同的域名的请求经过同一Nignx服务按合理的配置转发给不同的集群服务器，

1. 说明一下在工作中遇到的需要对sql语句进行优化的场景，以…优化的？

由于SQL优化起来比较复杂,并且还会受环境限制,在开发过程中,写SQL必须必须要遵循以下几点的原则：

　　1.ORACLE采用自下而上的顺序解析WHERE子句,根据这个原理,表之间的连接必须写在其他WHERE条件之前, 那些可以过滤掉最大数量记录的条件必须写在WHERE子句的末尾.

　　例如:

　　(低效)

　　SELECT … FROM EMP E WHERE SAL > 50000 AND JOB = ‘MANAGER’ AND 25 < (SELECT COUNT(\*) FROM EMP WHERE MGR=E.EMPNO);

　　(高效)

　　SELECT … FROM EMP E WHERE 25 < (SELECT COUNT(\*) FROM EMP WHERE MGR=E.EMPNO) AND SAL > 50000 AND JOB = ‘MANAGER’;

　　2.SELECT子句中避免使用’\*’

　　当在SELECT子句中列出所有的COLUMN时,使用动态SQL列引用 ‘\*’ 是一个方便的方法.可是,这是一个非常低效的方法. 实际上,ORACLE在解析的过程中, 会将’\*’ 依次转换成所有的列名, 这个工作是通过查询数据字典完成的, 这意味着将耗费更多的时间.

　　3.使用表的别名(Alias)

　　当在SQL语句中连接多个表时, 请使用表的别名并把别名前缀于每个Column上.这样一来,就可以减少解析的时间并减少那些由Column歧义引起的语法错误.

注：Column歧义指的是由于SQL中不同的表具有相同的Column名,当SQL语句中出现这个Column时,SQL解析器无法判断这个Column的归属。

1. 距离说明从哪些角度去对postgresql数据库进行优化，常…数据库配置项

答：通过优化文件系统，提高磁盘的I/O读写速度。通过优化表结构、增加索引、优化查询语句是系统响应速度更快。  
PostgreSQL性能优化主要包括查询速度优化、更新速度优化、数据库结构优化

1. odoo表单视图中对新创建数据的Manyone字段做筛选的…oddo表单视图中动态隐藏字段的方式（可选）。

答：input type="hidden" name="test" >提交表单到后台可以直接从表单form["test"]中取到值，

1. 说明一下bootstrap框架中界面布局的方式（可选）

答：1、固定宽度布局：为网页设置一个固定的宽度，通常以px做为长度单位，常见于PC端网页。

2、流式布局：为网页设置一个相对的宽度，通常以百分比做为长度单位。

3、栅格化布局：将网页宽度人为的划分成均等的长度，然后排版布局时则以这些均等的长度做为度量单位，通常利用百分比做为长度单位来划分成均等的长度。

4、响应式布局：通过检测设备信息，决定网页布局方式，即用户如果采用不同的设备访问同一个网页，有可能会看到不一样的内容，一般情况下是检测设备屏幕的宽度来实现。

1. Best Time to Buy and Sell Stock

Say you have an array for which the ith element is the price of a given stock on day i,

If you where only permitted to complete at most one transaction (ie, buy one and sell one share od the stock).design an aigorithm to find the maximum profit.

Example 1:

Input:[7, 1, 5, 3, 6, 4]

Output:5

Max.difference = 6-1 =5(not 7-1 = 6, as sellin price needs to be larger than buying price)

Example 2:

Input:[7, 1, 4, 3, 1]

Output:0

In this case, no transaction is done i.e.max profit = 0

Implement Stack using Queue

Implement the following operations of a stack using queues

push(x) --Push element x onto stack.

pop() -- Removes the element on top of the stac.

top() -- Get the top element.

empty() -- Return whether the stack is empty.

Notes:

You must use only standard operations of a queue -- which means only push to back,peek/pop form front,size,and is empty operations are valid.

Depending on your language,queue may not be supported natively,You may simulate a queue by using a list or deque(double-ended queue),as long as you use only standard operations of a queue

You may assume that all operations are valid (for example, no pop or top operations will be called on an empty stack).

1. 请写出用Python正则表达式，使assert成立

Import re

url = ‘http://www.google.com/search?q=’

p = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

m = re.search(p,url)

assert(m.group(1) == ‘google.com’)

1. 请写出下面html片段中3个li标签的css selector和xpath

<ul>

<li id=”a”></li>

<li name=’b’></li>

<li></li>

<div></div>

</ul>

1. 给定一个list A，实现一个function shuffle 将A中元素的顺序随机打乱

Example:

>>> A = range(5)

>>> shuffle(A)

>>> print(A)

[3,1,2,0,4]

1. 请实现一个function删除URL中的端口号

Example:

>>> url = ‘https://127.0.0.1:8000/path?query=q#fragment’

>>>remove\_port(url)

‘http://127.0.0.1/path?query=q#fragment’

1. 给定一个长度为n的list A，将它尽可能的平均分成k份，使得每份的个数之差不超过1

Example：

Input:

A = range(12)

k = 5

output:

[0,1,2],[3,4,5],[6,7],[8,9],[10,11]

1. 有2个字符串列表a和b，每个字符串是用逗号分隔的一些字符

a = [

‘a,1’,

‘b,3,22’,

‘c,3,4’,

]

b = [

‘a,2’,

‘b,1’,

‘d,2’,

]

按每个字符串的第一个值，合并a和b到c

c = [

‘a,1,2’,

‘b,3,22,1’,

‘c,3,4’,

‘d,2’,

]

1. Python的装饰器是什么，它的原理和作用分别是什么？

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。

功能：1.引入日志；2.函数执行时间统计；3.执行函数前预备处理；4.执行函数后清理功能；5.权限校验；6.缓存

1. range and xrange？两者有什么不同

Python 2.x 中：

range 函数说明：range([start,] stop[, step])，根据start与stop指定的范围以及step设定的步长，生成一个序列。

xrange 函数说明：用法与range完全相同，所不同的是生成的不是一个数组，而是一个生成器。

python 3.x中：

已经将xrange函数删除，range函数同Python2.x 中的xrange函数用法相同

1. 什么是装饰器？Functools.wraps这个装饰器的作用？

答：装饰器是一个函数，其主要用途是包装另一个函数或类。这种包装的首要目的是透明地修改或增强被包装对象的行为。

1. 有用过with statement吗？它的好处是什么？

答：**ith**语句的作用是**通过某种方式简化异常处理**，它是所谓的上下文管理器的一种

用法举例如下：

with open('output.txt', 'w') as f:

f.write('Hi there!')

当你要成对执行两个相关的操作的时候，这样就很方便，以上便是经典例子，**with语句会在嵌套的代码执行之后，自动关闭文件**。这种做法的还有另一个优势就是，**无论嵌套的代码是以何种方式结束的，它都关闭文件**。如果在嵌套的代码中发生异常，它能够在外部exception handler catch异常前关闭文件。如果嵌套代码有return/continue/break语句，它同样能够关闭文件。

1. 对比一下dict中items与iteritems

字典的items方法作用：是可以将字典中的所有项，以列表方式返回。因为字典是无序的，所以用items方法返回字典的所有项，也是没有顺序的。

字典的iteritems方法作用：与items方法相比作用大致相同，只是它的返回值不是列表，而是一个迭代器。

1. 以下为and/or的几个例子：

a)(1) ‘a’ and ‘b’ and ‘c’

‘c’

b)(2) ‘’ or [] or {}

{}

c)(3)>>>a=”first”

i.>>>b=”second”

ii.>>>0 and a or b

d)写出他们各自的输出

‘second’

1. （简答题）下列代码的输出

a=1

def fun(a):

a=2

fun(a)

b=[]

def fun(b):

b.append(1)

fun(b)

print(a,b)

1 [1]

1. （简答题）下列代码的输出

import copy

a=[1,2,3,4,[‘a’,’b’]]

b=a

c=copy.copy(a)

d=copy.deepcopy(a)

a.append(5)

a[4].append(‘c’)

print(‘a=’,a)

print(‘b=’,b)

print(‘c=’,c)

print(‘d=’,d)

a= [1, 2, 3, 4, ['a', 'b', 'c'], 5]

b= [1, 2, 3, 4, ['a', 'b', 'c'], 5]

c= [1, 2, 3, 4, ['a', 'b', 'c']]

d= [1, 2, 3, 4, ['a', 'b']]

1. （单选题）一个栈的入栈序列式ABCDE则不可能的出栈顺序是：C

A.DECBA B. DCEBA C. ECDBA D. ABCDE

1. （单选题）对进程和线程的描述，以下哪一个是正确的？D

A.父进程的所有线程共享相同的地址空间，父进程的所有子进程共享相同的地址空间。

B.改变进程里面的主线程的状态会影响其他线程的行为，改变父进程的状态不会影响其 他进程。

C.多线程会引发死锁，而多进程不会。

D.以上都不对

1. （单选题）计算机操作系统出现死锁的原因是什么？D

A.资源数大大少于进程数，或进程同时申请的资源数大大超过资源总数

B.有多个封锁的进程同时存在

C.一个进程进入死循环

D.若干进程因竞争资源而无休止的等待着其他进程释放已占有的资源

1. 以下不能创建一个字典的语句是（C）
2. dic1={}
3. dict2={3:5}
4. dic3=dict([2,5]:[3,4])
5. dic4=dict(([1,2],[3,4]))
6. 下列哪种说法是错误的 （C）
7. 除字典类型外，所有标准对象均可以用于布尔测试
8. 空字符串的布尔值是False
9. 空列表对象的布尔值是False

值为0的任何数字对象的布尔值是False

1. b=1 print(b.\_\_dict\_\_) 结果输出的是（c）
2. ｛‘b’:1｝
3. [1]

c. 抛异常

1. 请编写一段程序，输出101-200间的所有素数，并统计个数语言使用Python）

提醒：判断素数的方法：用一个数分别去除2到sqrt（该数），如果能被整除，则表示此数不能素数

1. 编写一个函数，用于生成随机密码，入参为随机密码的长度，出参为生成的随机密码，要求生成的随机密码必须同时包含大写字母，小写字母，数字.
2. 经过破译敌人密码，已经知道了“香蕉苹果大鸭梨”的意思是“星期三秘密进攻”，“苹果甘蔗水蜜桃”的意思是“执行秘密计划”，“广柑香蕉西红柿”的意思是“星期三的胜利属于我们”，那么“大鸭梨”的意思是\_\_\_\_c\_\_\_

A秘密B。星期三C、进攻D、执行E、计划

1. 一本书的价格降低了50%，现在，如果按原价出售，提高了百分之几\_\_d\_\_\_\_\_

A、25%B、50%C、75%D、100%E、200%

1. 五个答案中哪一个是最好的类比?水对于龙头相当于电对于\_\_b\_\_\_

A、光线B、开关C、电话D、危险E、电线

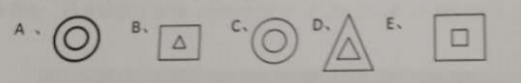
1. 打满水缸要11桶水，王林每次只能提2桶水，要打满水缸他需要走\_\_6\_\_\_趟

A、5 B.5.5 C、6 D、6.5 E、7

1. 小张、小李、小王、小刘共买苹果144个，小张买的苹果比小李多10个，比小王多26个，比小刘多32个。小张买了\_\_\_c\_\_个苹果

A、73 B、63 C、53 D、43 E、27

1. 找出与众不同的一个\_\_b\_\_\_



1. 火车守车(车尾)长6.4米，机车的长度等于守车的长加上半节车厢的长，车厢长度等于守车长加上机车长，火车的机车、车厢、守车共长\_\_\_a\_\_米

A、25.6 B、36 C、51.2 D、64.4 E、76.2

1. 在括号中填一个字，使这个字与括号外面的字分别组成两个字：古( )巴
2. 哥哥今年15岁，他的年龄是妹妹年龄的3倍，当哥哥的年龄是妹妹年龄的2倍时，哥哥\_\_b\_\_\_岁

A、18 B、20 C、24 D、26 E、30

1. 数数有\_\_\_\_\_个三角形。

A、5 B、7 C、9 D、11 E、13

1. 请用python代码实现生产者消费者模型（仅限使用python标准库提供的组件，勿使用第三方库）

答：产者消费之模型就是,比如一个包子铺,中的顾客吃包子,和厨师做包子,  
不可能是将包子一块做出来,在给顾客吃,但是单线程只能这麽做,  
所以用多线程来执行,厨师一边做包子,顾客一边吃包子,  
当顾客少时,厨师做的包子就放在一个容器中,等着顾客来吃,  
当顾客多的时候,就从容器中先取出来给顾客吃,厨师继续做包子  
用队列来模拟这个容器  
  
1

# 当做完一个包子后就要给顾客发送一个信号,表示已经做完,让他们吃包子

[IMG_256](https://www.cnblogs.com/xiaokang01/p/javascript:void(0);)

import threading, time, queue

q = queue.Queue()def Produce(name):

count = 0 # conut表示做的包子总个数

while count < 10:

print('厨师%s在做包子中...'%name)

time.sleep(2)

q.put(count) # 容器中添加包子

# 当做完一个包子后就要给顾客发送一个信号,表示已经做完,让他们吃包子

print('produce%s已经做好了第%s个包子'%(name, count))

count += 1

print('oking...')def Consumer(name):

count = 0 # count表示包子被吃的总个数

while count < 10:

time.sleep(2) # 排队去取包子,

if not q.empty(): # 如果存在

data = q.get() # 取包子, 吃包子

print('\033[32;1mConsumer %s已经把第%s个包子吃了...\033[0m' %(name, data))

else:

print('包子被吃完了...')

count += 1if \_\_name\_\_ == '\_\_main\_\_':

p1 = threading.Thread(target=Produce, args=('A君',))

c1 = threading.Thread(target=Consumer, args=('B君',))

c2 = threading.Thread(target=Consumer, args=('C君',))

c3 = threading.Thread(target=Consumer, args=('D君',))

p1.start()

c1.start()

c2.start()

c3.start()

[IMG_257](https://www.cnblogs.com/xiaokang01/p/javascript:void(0);)

2这里就是 当顾客吃完了然后给生产者发送一个信号  
当生产者就接收到信号时,继续做包子

[IMG_258](https://www.cnblogs.com/xiaokang01/p/javascript:void(0);)

import threading, time, queue

q = queue.Queue()

def Produce(name):

count = 0 # conut表示做的包子总个数

while count < 10:

print('厨师%s在做包子中...'%name)

time.sleep(2)

q.put(count) # 容器中添加包子

print('produce%s已经做好了第%s个包子'%(name, count))

count += 1

# q.task\_done() # 当做完一个包子后就要给顾客发送一个信号,表示已经做完,让他们吃包子

q.join() #等待接收信号,

print('ok...')def Consumer(name):

count = 0 # count表示包子被吃的总个数

while count < 10:

time.sleep(2)

# print('waiting...')

# q.join()

data = q.get() # 取包子, 吃包子

print('%seating...'%name)

time.sleep(4) # 吃包子用了4s然后给厨师发送一个信号 q.task\_done()

print('\033[32;1mConsumer %s已经把第%s个包子吃了...\033[0m' % (name, data))

# print('包子被吃完了...')

count += 1if \_\_name\_\_ == '\_\_main\_\_':

p1 = threading.Thread(target=Produce, args=('A君',))

c1 = threading.Thread(target=Consumer, args=('B君',))

c2 = threading.Thread(target=Consumer, args=('C君',))

c3 = threading.Thread(target=Consumer, args=('D君',))

p1.start()

c1.start()

c2.start()

c3.start()

[IMG_259](https://www.cnblogs.com/xiaokang01/p/javascript:void(0);)

IMG_260 利用同步对象event模拟信号

1. 请编写一个开平方函数，其输入为一个double类型非负值，输出为输入值开方的结果，要求保留小数点后6位。

**def sqrt\_opt**(*a*)**:** b **=** math.sqrt(*a*)  
 print(**'%.6f'%**b)

1. （选做题）：现有不限数量的面额分别为1元，5元，10元，20元，50元，100元的货币；如果要用这些货币组合成x元，求一共有多少种组合方式？请说明这类问题在算法中被叫做什么问题，并用代码实现。

答：现有6种面额的纸币用来组合成给定的x元金额。那么可以大致推出这个等式  
sum 表示给定的金额  
{x1, x2, x3, x4, x5, x6}分别表示1元，5元，10元，20元，50元、100元的*张数*

sum=x1∗1+x2∗5+x3∗10+x4∗20+x5∗50+x6∗100sum = x1 \* 1 + x2 \* 5 + x3 \* 10 + x4 \* 20 + x5 \* 50 + x6 \* 100*sum*=*x*1∗1+*x*2∗5+*x*3∗10+*x*4∗20+*x*5∗50+*x*6∗100

如此看来其实就是求解满足这个等式的 {x1, x2, x3, x4, x5, x6} 的所有可能的个数。  
可以通过循环来依次确定每种面额的纸币有多少张，最终来判断，不同张数的组合最终是否等于x元。  
于是有了如下代码：

publicclassDemo1{

*/\*\**

*\* @param x 商品金额*

*\*/*

publicstaticvoidtest1(int *x*){

int *sum* =0;

*//符合条件的组合次数*

int *count* =0;

*//循环次数*

int *times* =0;

*//硬币面额*

int[] *a* ={1,5,10,20,50,100};

for(int *i* =0; *i* <= *x* / *a*[5]; *i*++){

*//100元可能出现的张数*

for(int *j* =0; *j* <= *x* / *a*[4]; *j*++){

*//50元可能出现的张数*

for(int *k* =0; *k* <= *x* / *a*[3]; *k*++){

*//20元可能出现的张数*

for(int *l* =0; *l* <= *x* / *a*[2]; *l*++){

*//10元可能出现的张数*

for(int *m* =0; *m* <= *x* / *a*[1]; *m*++){

*//5元可能出现的张数*

*//for(int n=0;n<x/1;n++){//这步循环可省略*

int *n* = *x* -(*i* \* *a*[5]+ *j* \* *a*[4]+ *k* \* *a*[3]+ *l* \* *a*[2]+ *m* \* *a*[1]);

*sum* = *i* \* *a*[5]+ *j* \* *a*[4]+ *k* \* *a*[3]+ *l* \* *a*[2]+ *m* \* *a*[1]+ *n* \* *a*[0];

*times*++;

if(*sum* == *x* && *n* >=0){

*count*++;

}

*//}*

}

}

}

}

}

*System*.*out*.println("循环次数："+ *times*);

*System*.*out*.println("组合数："+ *count*);

}

publicstaticvoidmain(*String*[] *args*){

long *startTime* = *System*.currentTimeMillis();

*//指定200元的金额*

test1(200);

long *endTime* = *System*.currentTimeMillis();

*System*.*out*.println("执行时间："+(*endTime* - *startTime*)+"ms");

}

}

* 1

执行结果如下：

*循环次数：142065*

*组合数：3274*

*执行时间：13ms*

1. 下面代码输出什么

class Parent（object）：

x=1

class Child1（object）：

pass

class Child2（object）：

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

1. 下面代码输出什么

def multipliers():

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

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

1. 实现二分查找

在一个排序数组中找一个数，返回该数的位置，如果不存在则返回-1，例如数组【1,3,5,7,9】查找元素“3”，返回位置1.

答：*# 递归*

*def* *binary\_search(lst, value, lo, hi):*

*if* *lo > hi:*

*return* *-1*

*half* *=* *(lo* *+* *hi)/2*

*if* *lst[half]* *==* *value:*

*return* *half*

*elif* *lst[half] > value:*

*return* *binary\_search(lst, value, lo, half-1)*

*else:*

*return* *binary\_search(lst, value, half+1, hi)*

1. python的import有几种定义方式？import的搜索路径是什么？

答：.函数功能用于动态的导入模块，主要用于反射或者延迟加载模块

2.\_\_import\_\_(module) 相当于 import module

\_\_import\_\_导入的模块,搜索的路径首先在sys.path列表中.

例如:

在目录/home/ts/Downloads/test/TEST/a/b/c下有index.py文件

import sys

#添加路径

sys.path.append('/home/ts/Downloads/test/TEST/a/b/c')

#将自定义的文件导入

cfile = \_\_import\_\_('index')

dir(cfile) : 查看index.py文件的内容

---------------------

1. python装饰器的概念，写法，用法？

答：简单来说，可以把装饰器理解为一个包装函数的函数，它一般将传入的函数或者是类做一定的处理，返回修改之后的对象．所以，我们能够在不修改原函数的基础上，在执行原函数前后执行别的代码．比较常用的场景有日志插入，事务处理等．

**装饰器**

最简单的函数，返回两个数的和

[?](https://www.jb51.net/article/134782.htm)

|  |  |
| --- | --- |
| 1  2  3 | ***def*** *calc\_add(a, b):*  ***return*** *a* ***+*** *b*  *calc\_add(1, 2)* |

但是现在又有新的需求，计算求和操作耗时，很简单，求和前获取一下时间，求和后再获取一次，求差即可

[?](https://www.jb51.net/article/134782.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | ***import*** *datetime*  ***def*** *calc\_add(a, b):*  *start\_time* ***=*** *datetime.datetime.now()*  *result* ***=*** *a* ***+*** *b*  *end\_tiem* ***=*** *datetime.datetime.now()*  ***print*** *"result:", result, "used:", (end\_tiem* ***-*** *start\_time).microseconds, "μs"*  ***return*** *result*  *calc\_add(1, 2)* |

现在呢，函数*calc\_diff(a, b)*，计算a-b，也想计算减法操作的时间差，很好办，把那段代码复制过去．但是假如我们现在想编的是一个数学函数库，各种函数都想计算其执行耗时，总不能一个一个复制代码，想个更好的办法．

我们知道，在Python中函数也是被视为对象的，可以作为参数传递，那么假如把计算耗时的独立为一个单独的函数*calc\_spend\_time()*，然后把需要计算耗时的函数例如calc\_add的引用传递给它，在calc\_spend\_time中调用calc\_add，这样所有的需要计算耗时的函数都不用修改自己的代码了．

[?](https://www.jb51.net/article/134782.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | ***def*** *calc\_spend\_time(func,* ***\*****args,* ***\*\*****kargs):*  *start\_time* ***=*** *datetime.datetime.now()*  *result* ***=*** *func(****\*****args,* ***\*\*****kargs)*  *end\_tiem* ***=*** *datetime.datetime.now()*  ***print*** *"result:", result, "used:", (end\_tiem* ***-*** *start\_time).microseconds, "μs"*  ***def*** *calc\_add(a, b):*  ***return*** *a* ***+*** *b*  *calc\_spend\_time(calc\_add, 1, 1)*  *# calc\_spend\_time(calc\_add, a=1, b=2)* |

看起来也不错，负责计算的函数不用更改，只需调用的时候作为参数传给计算时间差的函数．但就是这，调用的时候形式变了，不再是clac(1, 2)，而是calc\_spend\_time(clac\_add, 1, 2)，万一calc\_add大规模被调用，那么还得一处一处找，然后修改过来，还是很麻烦．如果想不修改代码，就得使*clac()*和*calc\_spend\_time(clac)*效果一样，那么可以在*calc\_spend\_time()*里把传入的clac包装一下，然后返回包装后的新的函数，再把返回的包装好的函数赋给clac，那么calc()的效果就和上例*calc\_spend\_time(calc())*效果一样．

[?](https://www.jb51.net/article/134782.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | ***import*** *datetime*  ***def*** *calc\_spend\_time(func):*  ***def*** *new\_func(a, b):*  *start\_time* ***=*** *datetime.datetime.now()*  *result* ***=*** *func(a, b)*  *end\_tiem* ***=*** *datetime.datetime.now()*  *print* *"result:", result, "used:", (end\_tiem* ***-*** *start\_time).microseconds, "μs"*  ***return*** *new\_func*  ***def*** *calc\_add(a, b):*  ***return*** *a* ***+*** *b*  *calc\_add* ***=*** *calc\_spend\_time(calc\_add)*  *calc\_add(1, 2)* |

1. 简述select，poll，epoll的作用和优缺点？

答：select优点

1）select()的可移植性更好，在某些Unix系统上不支持poll()

2）select() 对于超时值提供了更好的精度：微秒，而poll是毫秒。

select缺点

1） 单个进程可监视的fd数量被限制。

2） 需要维护一个用来存放大量fd的数据结构，这样会使得用户空间和内核空间在传递该结构时复制开销大。

3） 对fd进行扫描时是线性扫描。fd剧增后，IO效率较低，因为每次调用都对fd进行线性扫描遍历，所以随着fd的增加会造成遍历速度慢的性能问题

4）select() 函数的超时参数在返回时也是未定义的，考虑到可移植性，每次在超时之后在下一次进入到select之前都需要重新设置超时参数。

poll

　　poll与select不同，通过一个pollfd数组向内核传递需要关注的事件，故没有描述符个数的限制，

　　pollfd中的events字段和revents分别用于标示关注的事件和发生的事件，故pollfd数组只需要被初始化一次。

　　poll的实现机制与select类似，其对应内核中的sys\_poll，只不过poll向内核传递pollfd数组，然后对pollfd中的每个描述符进行poll，相比处理fdset来说，poll效率更高。

　　poll返回后，需要对pollfd中的每个元素检查其revents值，来得指事件是否发生。

poll优点

1）poll() 不要求开发者计算最大文件描述符加一的大小。

2）poll() 在应付大数目的文件描述符的时候相比于select速度更快

3）它没有最大连接数的限制，原因是它是基于链表来存储的。

poll缺点

1）大量的fd的数组被整体复制于用户态和内核地址空间之间，而不管这样的复制是不是有意义。

2）与select一样，poll返回后，需要轮询pollfd来获取就绪的描述符

epoll

　　epoll是Linux下多路复用IO接口select/poll的增强版本。

　　它能显著减少程序在大量并发连接中只有少量活跃的情况下的系统CPU利用率，因为它不会复用文件描述符集合来传递结果而迫使开发者每次等待事件之前都必须重新准备要被侦听的文件描述符集合。

　　另一点原因就是获取事件的时候，它无须遍历整个被侦听的描述符集，只要遍历那些被内核IO事件异步唤醒而加入Ready队列的描述符集合就行了。

　　epoll除了提供select/poll 那种IO事件的电平触发（Level Triggered）外，还提供了边沿触发（Edge Triggered），这就使得用户空间程序有可能缓存IO状态，减少epoll\_wait/epoll\_pwait的调用，提高应用程序效率。

底层实现

　　epoll在底层实现了自己的高速缓存区，并且建立了一个红黑树用于存放socket，另外维护了一个链表用来存放准备就绪的事件。

工作过程：

　　执行epoll\_ create时，创建了红黑树和就绪链表，执行epoll\_ ctl时，如果增加socket句柄，则检查在红黑树中是否存在，存在立即返回，不存在则添加到树干上，然后向内核注册回调函数，用于当中断事件来临时向准备就绪链表中插入数据。执行epoll\_wait时立刻返回准备就绪链表里的数据即可。

epoll优点

1）支持一个进程打开大数目的socket描述符(FD)

2）IO效率不随FD数目增加而线性下降

3）使用mmap加速内核与用户空间的消息传递。

总结

（1）select，poll实现需要自己不断轮询所有fd集合，直到设备就绪，期间可能要睡眠和唤醒多次交替。

　　而epoll其实也需要调用 epoll\_ wait不断轮询就绪链表，期间也可能多次睡眠和唤醒交替，但是它是设备就绪时，调用回调函数，把就绪fd放入就绪链表中，并唤醒在 epoll\_wait中进入睡眠的进程。

　　虽然都要睡眠和交替，但是select和poll在“醒着”的时候要遍历整个fd集合，而epoll在“醒着”的 时候只要判断一下就绪链表是否为空就行了，这节省了大量的CPU时间，这就是回调机制带来的性能提升。

（2）select，poll每次调用都要把fd集合从用户态往内核态拷贝一次，并且要把current往设备等待队列中挂一次，而epoll只要一次拷贝，而且把current往等待队列上挂也只挂一次（在epoll\_wait的开始，注意这里的等待队列并不是设备等待队列，只是一个epoll内 部定义的等待队列），这也能节省不少的开销。

1. 有3个对象，分别为 a=['1','2','3','4'], b='1234', c={'1':1,'2':2,'3':3,'4':4},

用大O记法，写出以下实现的时间复杂度。

1. '1' not in a

O(f(n))

1. '2' in b

O(f(n))

1. '3' in c

O(f(1))

1. 请尝试用“一行代码”实现随机数数列表以3为单位分组，比如1-100分组后是为：

[[2,6,12],[45,43,2], ... ,[71]], 现在考虑给定任意（N）个元素的随机数数列，如何实现以3为单位分组。（提示：列表解析）

1. 现在考虑有一个jsonline格式的文件 file.txt 大小约为 10k，JAMY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():

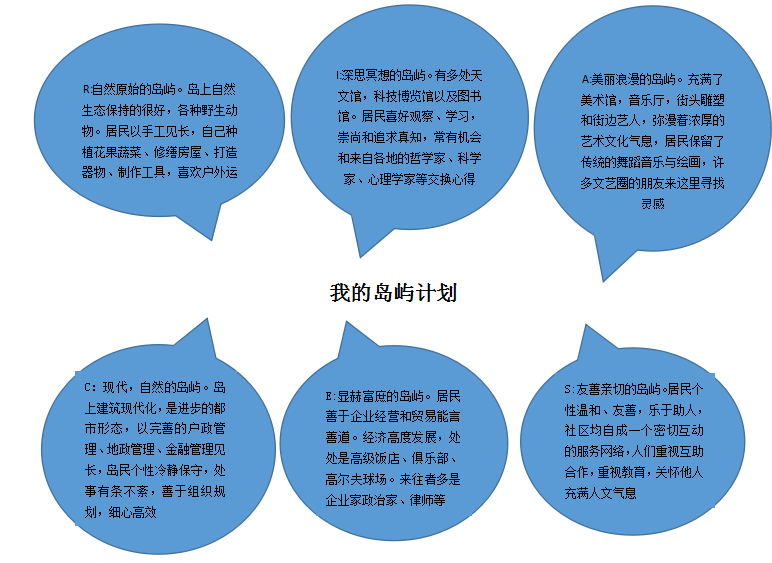
process(e) #处理每一行数据

现在JAMY被交付一个大小为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（提示：生成器）。需要考虑的问题 都有哪些。

**测试**

假设在你去度假的途中，你所乘坐的轮船突然发生了意外故障，必须紧急靠岸。这时候，轮船正好处于下列6个岛屿中间。幸运的是，你可以凭借自己的兴趣选择一个岛屿靠岸。

要知道，这些岛屿只能通过轮船与外界联系，由于天气原因，今后至少半年内船只都无法出航，而且你还要等待境外的轮船运送人员和器材前来维修你所乘坐的轮船。因此一旦靠岸，你可能需要在这个岛屿上呆很长一段时间（至少一年）。请按一，二，三的顺序挑出3个岛屿。



1. 给定数组a,调整期中元素的顺序，使得奇数在前偶数在后。

样例：a=[1,2,3,4],返回[1,3,2,4]

**def func**(*arr*)**:** a1 **=** []  
 a2 **=** []  
 **for** a **in** *arr***:  
 if** a**%**2**==**0**:** a2.append(a)  
 **else:** a1.append(a)  
 **return** a1 **+** a2

1. 给定两个版本号v1、v2,如v1>v2, 返回1, v1<v2,返回-1, 否则返回0。

样例：v1=”0.1.1”, v2=”0.1.2”, 返回-1; v1=”1.2”, v2=”1.1.13”, 返回1。

思路：按照点对字符串进行截取，把截取内容转成数值，放入列表，把列表从0开始往后比对

1. 考虑有一个jsonline格式的文件file.txt大小约为10K，JAMY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\_\_”:

get\_lines():

1. Python内部字符串编码方式？

ASCII, UTF-8, unicode

1. Python下有哪些并发方案？

多进程 多线程 协成

1. 常用的异步任务框架有哪些？

Celery tornordo

1. 递归读取目录下的文件列表，并打印到屏幕？

import os

#遍历文件夹

def iter\_files(rootDir):

#遍历根目录

for root,dirs,files in os.walk(rootDir):

for file in files:

file\_name = os.path.join(root,file)

print(file\_name)

for dirname in dirs:

#递归调用自身,只改变目录名称

iter\_files(dirname)

1. 下面这些是什么意思：@classmethod.@staticmethod.@property？

@staticmethod或@classmethod或@property的语法糖。本质上，它们都是函数装饰器，只不过通常被用来修饰类成员函数而已。

1. “猴子补丁”（monkey patching）指的是什么？这种做法好吗？

猴子补丁”就是指，在函数或对象已经定义之后，再去改变它们的行为。

举个例子：

import datetime

datetime.datetime.now = lambda: datetime.datetime(2018, 8, 29)

大部分情况下，这是种很不好的做法 - 因为函数在代码库中的行为最好是都保持一致。打“猴子补丁”的原因可能是为了测试。mock包对实现这个目的很有帮助。

为什么提这个问题？

答对这个问题说明你对单元测试的方法有一定了解。你如果提到要避免“猴子补丁”，可以说明你不是那种喜欢花里胡哨代码的程序员（公司里就有这种人，跟他们共事真是糟糕透了），而是更注重可维护性。还记得KISS原则码？答对这个问题还说明你明白一些Python底层运作的方式，函数实际是如何存储、调用等等。

另外：如果你没读过mock模块的话，真的值得花时间读一读。这个模块非常有用。

1. Linux下常用的进程管理工具有哪些？

htop - Linux下强大的进程管理工具

1. 常用的Web框架有哪些？

Django flask tornado

1. 写出A0，A1，….An 的值

A0 = dict（zip（（‘a’ , ’b’ , ’c’ , ‘d’ , ‘e’）,(1、2、3、4、)））

A1 = range（10）

A2 = sorteb（[ i for i in A1 if i in A0] ）

A3 = sorteb（ [ 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 ]

name 'sorteb' is not defined

1. 写出下面code 的输出：

def f （x，l = [ ] ）:

for i in range (x) :

l . append( i \* i )

print ( l )

f (2)

f(3,[3,2,1])

f(3)

1. 写一个能匹配的邮箱格式的Python正则。

**[0-9a-zA-Z\_]{0,19}@163.com**

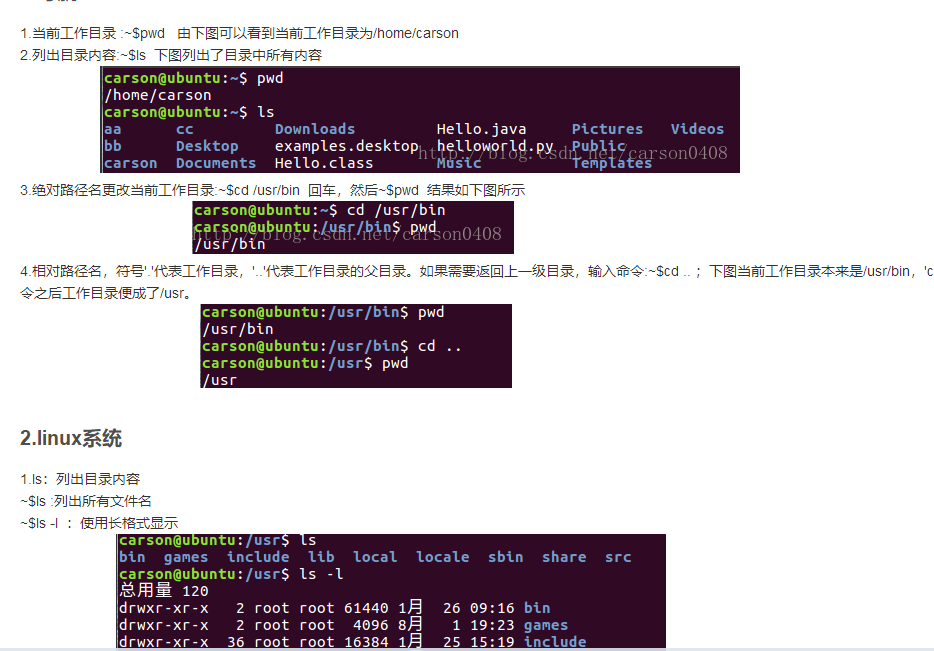
1. Python函数中的\*args和\*\*kwargs分表表示什么意思？

首先，解释星号的作用，一个星号\*的作用是将tuple或者list中的元素进行unpack，分开传入，作为多个参数；两个星号\*\*的作用是把dict类型的数据作为参数传入。kwargs是keyword argument的缩写，args就是argument。我们知道，在Python中有两种参数，一种叫位置参数（positional argument），一种叫关键词参数（keyword argument），关键词参数只需要用 keyword = somekey 的方法即可传参，而位置参数只能由参数位置决定。这也就决定了位置参数一定要在前面，否则关键词参数数量的变化（比如有些kwargs有默认值因此没有传参或者在后面传参的），都会使得位置无法判断。因此常见的也是\*args 在 \*\*kwargs 前面

1. Python类中的@staticmethod和@classmethod分别表示什么意思？

静态方法和类方法

1. 列举几个Linux的常用命令，并举例说明他们各自的用途。



1. 考试以下python代码，如果运行结束，命令行中的运行结果是？

If\_\_name\_\_ == ’\_\_main\_\_’:

l = []

for i in xrnage(10):

l.append({‘num’:i})

print l

再考虑以下代码，运行结束后的结果是什么

if \_\_name\_\_ == “\_\_main\_\_”:

l = []

a = {‘num’:0}

for x in xrange(10) :

a[‘num’] = x

l.append(a)

print(l)

以上两段代码的运行结果是否相同 如果不同 原因是什么

1. sql中有一个表

|  |  |  |  |
| --- | --- | --- | --- |
| 品牌（branch） | 类别（category） | 售价（price） | 数量（number） |
| Nike | 鞋 | 300 | 2 |
| Nike | 衣服 | 299 | 1 |
| Adidas | 鞋 | 520 | 1 |
| … |  |  |  |

使用sql语句统计出以下的数据：

销售排名前5位的品牌

select branch from sql where 1 order by number DESC limit 5

平均销售价格最高的类别

select category from sql where 1 order by price DESC limit 1

1. python什么是全局锁

我们所说的Python全局解释锁（GIL）简单来说就是一个互斥体（或者说锁），这样的机制只允许一个线程来控制Python解释器。

这就意味着在任何一个时间点只有一个线程处于执行状态。GIL对执行单线程任务的程序员们来说并没什么显著影响，但是它成为了计算密集型（CPU-bound）和多线程任务的性能瓶颈。

由于GIL即使在拥有多个CPU核的多线程框架下都只允许一次运行一个线程，所以在Python众多功能中其声誉可谓是“臭名昭著”。

1. python中的多线程有何优缺点

多进程：

　　优点：可以用多核

　　缺点：开销大

多线程：

　　优点：开销小

缺点：不能使用多核

1. 假设我们需要对某个社交网站（譬如微博）进行数据采集工作，请结合以下各方面对采集准备进行描述。

1）采集内容研究

需要采集什么内容由什么决定，如何判断哪些数据比较有价值

采集内容由需求决定 和结果直接相关的更有价值

2）采集接口研究

数据接口格式的研究方式有哪些，需要获取哪些数据接口

可以通过chrome的network标签查看数据的接口

需要获取目标数据的接口

3）如何准备采集资源

如何考虑账号资源、代理资源在采集工作中的重要程度，以及如何准备这些资源

一般可以自己申请或者从一些已有的脱裤数据中获取

4）采集架构的设计

采用什么样的架构来开展数据采集工作

一般都是多线程的架构，一边多线程获取一般多线程解析，另外可以分布式爬虫爬取

1. 在python中，list，tuple，dict，set有什么区别，主要应用在什么样的场景？

list是有序集合 内容可变

tuple内容不可变

dict存储的是对应关系

set是另一种包含无序不重复元素的数据结构，就如同dict中的key一样，只不过其相对于dict只有key，而没有与key相对应的value

1. 请写出打印结果

x=[0,1]

i=0

i,x[i] = 1,2

print(x)

打印结果x:

[0,2]

1. 请用自己的算法，按升序合并如下两个list，并去除重复的元素：

List1 = [2,3,8,4,9,5,6]

List2 = [5,6,10,17,11,2]

list3 **=** List1 **+** List2  
  
list3 **=** sorted(list3)  
ls **=** []  
  
**for** l **in** list3**:  
 if** l **in** ls**:  
 continue  
 else:** ls.append(l)  
  
print(ls)

1. 关系型数据库中，表与表之间有左链接，内连接，外链接。分别解释下他们的含义及区别？

left join （左连接）：返回包括左表中的所有记录和右表中连接字段相等的记录。

　　right join （右连接）：返回包括右表中的所有记录和左表中连接字段相等的记录。

　　inner join （等值连接）：只返回两个表中连接字段相等的行。

　　full join （全外连接）：返回左右表中所有的记录和左右表中连接字段相等的记录。

1. 题目描述：现有两个hive表，表结构分别如下：

|  |  |
| --- | --- |
| 表名 | 字段 |
| t\_user:用户基础信息表 | uld:用户id  name:姓名  city:用户归属城市  age:年龄 |
| t\_pay\_log:消费记录表 | uld:用户id  ctime:消费时间  money:消费金额 |

问题1：统计最近一周消费金额分别在0-100,101-500，大于500的用户数量，用户平均年龄

问题2：统计历史累计充值消费总金额top10用户的uid，姓名，年龄，累计充值总金额

问题3：统计每个城市内，最近一周充值金额top10用户的uid，姓名，年龄，累计充值总金额

1. 某hive数据表tmp\_tag内只有一个字段，字段名为tag，按行存放用户画像标签数据如下，请使用hive sql对该字段进行拆分：
2. cron后台常驻程序（daemon）用于：

A.负责文件在网络中的共享

B.管理打印子系统

C.跟踪管理系统信息和错误

D.管理系统日常任务的调度

1. 哪些统计方法同来度量数据的集中趋势？

A.平均值和正态分布

B.平均值，中位数和众数

C.众数，Alpha和极差

D.标准擦，极差和平均值

E.中位数，极差和正态分布

1. 皮尔森（Pearson）相关性捕捉了两个变量之间的线性依赖关系，而斯皮尔曼（Spearman）相关性捕捉的是两个变量之间的单词相关关系。

A.正确

B.错误

1. 如何用python来进行查询和替换一个文本字符串

正则

1. 遇到过得反爬虫策略以及解决办法

1）.通过Headers反爬虫

从用户请求的Headers反爬虫是最常见的反爬虫策略。很多网站都会对Headers的User-Agent进行检测，还有一部分网站会对Referer进行检测（一些资源网站的防盗链就是检测Referer）。如果遇到了这类反爬虫机制，可以直接在爬虫中添加Headers，将浏览器的User-Agent复制到爬虫的Headers中；或者将Referer值修改为目标网站域名。对于检测Headers的反爬虫，在爬虫中修改或者添加Headers就能很好的绕过。

2）.基于用户行为反爬虫

还有一部分网站是通过检测用户行为，例如同一IP短时间内多次访问同一页面，或者同一账户短时间内多次进行相同操作。

大多数网站都是前一种情况，对于这种情况，使用IP代理就可以解决。可以专门写一个爬虫，爬取网上公开的代理ip，检测后全部保存起来。这样的代理ip爬虫经常会用到，最好自己准备一个。有了大量代理ip后可以每请求几次更换一个ip，这在requests或者urllib2中很容易做到，这样就能很容易的绕过第一种反爬虫。

对于第二种情况，可以在每次请求后随机间隔几秒再进行下一次请求。有些有逻辑漏洞的网站，可以通过请求几次，退出登录，重新登录，继续请求来绕过同一账号短时间内不能多次进行相同请求的限制。

3）.动态页面的反爬虫

上述的几种情况大多都是出现在静态页面，还有一部分网站，我们需要爬取的数据是通过ajax请求得到，或者通过JavaScript生成的。首先用Fiddler对网络请求进行分析。如果能够找到ajax请求，也能分析出具体的参数和响应的具体含义，我们就能采用上面的方法，直接利用requests或者urllib2模拟ajax请求，对响应的json进行分析得到需要的数据。

能够直接模拟ajax请求获取数据固然是极好的，但是有些网站把ajax请求的所有参数全部加密了。我们根本没办法构造自己所需要的数据的请求。这种情况下就用selenium+phantomJS，调用浏览器内核，并利用phantomJS执行js来模拟人为操作以及触发页面中的js脚本。从填写表单到点击按钮再到滚动页面，全部都可以模拟，不考虑具体的请求和响应过程，只是完完整整的把人浏览页面获取数据的过程模拟一遍。

用这套框架几乎能绕过大多数的反爬虫，因为它不是在伪装成浏览器来获取数据（上述的通过添加 Headers一定程度上就是为了伪装成浏览器），它本身就是浏览器，phantomJS就是一个没有界面的浏览器，只是操控这个浏览器的不是人。利selenium+phantomJS能干很多事情，例如识别点触式（12306）或者滑动式的验证码，对页面表单进行暴力破解等。

1. 列举网络爬虫所用到的网络数据包,解析包

1、fiddle抓包

2、chrome浏览器network

3、火狐浏览器network

1. 你个人工作中遇到哪些问题及解决方法

多用户访问造成拥堵

使用缓存解决

1. 有一个列表a,里面的元素都是整数类型,求重复次数最多的和第二多的元素.
2. 有两个列表里面存的都是整数a和b

1) 合并a的前3个元素和b的后3个元素,并按照当前排列顺序倒序输出

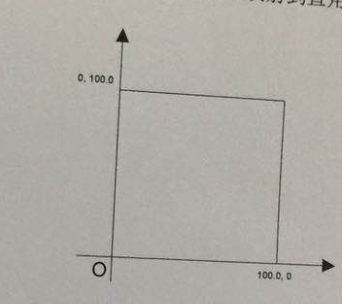
2)求a和b的交集和并集

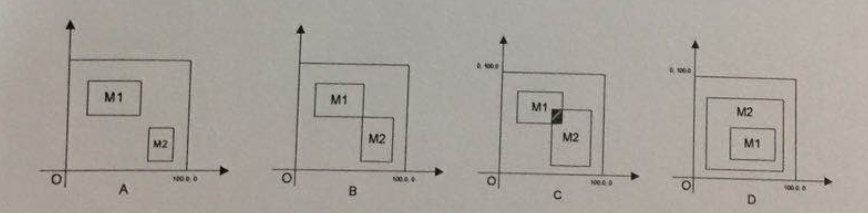
3)假设a和b的并集为c,求出结果中最长连续片段

例:[.1,2,3,5,6,10,11,12,13,22,33] --> [10,11,12,13]

1. 在python中，使用列表（x1,y1）,(x2,y2)……(xn,yn)表示一个多边形，比如一个矩形可以表示为[(1.0,1.0),(2.0,1.0),(2.0,2.0),(1.0,2.0),(1.0,1.0)](列表中有五个点，首尾两个点重合，相邻点之间使用直线互连)。如N边形，列表内点的个数是N+1。

现在有长X宽为100.0X100.0的一块大矩形区域映射到直角坐标系上如下所示：



现考虑在此矩形区域之中存在两块小矩形区域M1和M2，根据M1和M2的重叠情况 分为如下A,B,C和D四种情况：

问题1：如何编程实现判断两个矩形是否不重叠，部分重叠活着完全重叠，如果重叠计 算重叠区域的面积大小。

假设

M2[,

M2[,

1. 写程序得到两个列表list的交集和差集
2. 写出一个单例类

class Singleton(object):

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

# 关键在于这，每一次实例化的时候，我们都只会返回这同一个instance对象

if not hasattr(cls, 'instance'):

cls.instance = super(Singleton, cls).\_\_new\_\_(cls)

return cls.instance

obj1 = Singleton()

obj2 = Singleton()

obj1.attr1 = 'value1'

print obj1.attr1, obj2.attr1

print obj1 is obj2

1. 用Python正则匹配HTML tag的时候，<.\*>和<.\*?>有什么区别

第一个使用贪婪模式 第二个是非贪婪模式

1. 下面的代码会输出什么：

def f(x,l=[]) :

for i in range(x):

append(i\*i)

print(l)

f(2)

f(3,[3,2,1])

f(3)

1. 平衡点：比如int[] numbers={1,3,5,7,8,25,4,20};25前面的总和为24,25后面的总和也是24,25这个点就是平衡点；假如一个数组中的元素，其前面的部分等于后面的部分，那么这个点的位序就是平衡点
2. 数组和元祖之间的区别？

数组内容是可以被修改的，而元组内容是只读的。另外，元组可以被哈希，比如作为字典的关键字。

1. 介绍一下Scrapy的数据处理流程
2. 引擎打开一个域名，时蜘蛛处理这个域名，并让蜘蛛获取第一个爬取的URL。
3. 引擎从蜘蛛那获取第一个需要爬取的URL，然后作为请求在调度中进行调度。
4. 引擎从调度那获取接下来进行爬取的页面。
5. 调度将下一个爬取的URL返回给引擎，引擎将他们通过下载中间件发送到下载器。
6. 当网页被下载器下载完成以后，响应内容通过下载中间件被发送到引擎。
7. 引擎收到下载器的响应并将它通过蜘蛛中间件发送到蜘蛛进行处理。
8. 蜘蛛处理响应并返回爬取到的项目，然后给引擎发送新的请求。
9. 引擎将抓取到的项目项目管道，并向调度发送请求。
10. 有一篇英文文章保存在a.txt中，请用python实现统计这篇文章内每个单词的出现频率，并返回出现频率最高的前10个单词及其出现次数（只考虑空格，标点符号可忽略。）

import re  
   
with open('1.txt', 'r') as f:  
    dictResult = {}  
   
    # 每次读取一行的文件  
    while True:  
        line = f.readline()  
        if not line:  
            break  
        listMatch = re.findall('\w+', line.lower()) # 转为小写字母  
    # 计数  
        for eachLetter in listMatch:  
            eachLetterCount = len(re.findall(eachLetter, line.lower()))  
            dictResult[eachLetter] = dictResult.get(eachLetter, 0) + eachLetterCount  
   
    # 从大到小排序  
    result = sorted(dictResult.items(), key=lambda d: d[1], reverse=True)[:3]  
    for each in result:  
        print(each)

1. 将下列网址提取出域名：

http://www.interoem.com/messageinfo.asp?id=35

http://3995503.com/class/class09/news\_show.asp?id=14

http://lib.wzmc.edu.cn/news/onews.asp?id=739

http://www.zy-ls.com/alfx.asp?newsid=337&id=6

http://fincm.com/newslist.asp?id=415

import urllib

print "--"\*40  
for url in urls:  
    proto, rest = urllib.splittype(url)  
    res, rest = urllib.splithost(rest)  
    print "unkonw" if not res else res

1. 给定一个数组，构建二叉树，并且按层次打印这个二叉树。

#include<iostream>

#include<malloc.h>

#include<queue>

#include<list>

using namespace std;

struct node

{

char c;

node \*lchild,\*rchild;

};

char pre[100],mid[100];

void build(node\* &t,int start1,int end1,int start2,int end2)

{

int i=start2;

while(pre[start1]!=mid[i])

i=i+1;

t=(node\*)malloc(sizeof(node));

t->c=pre[start1];

if(i==start2)

t->lchild=NULL;

else build(t->lchild,start1+1,start1+i-start2,start2,i-1);

if(i==end2)

t->rchild=NULL;

else build(t->rchild,start1+i-start2+1,end1,i+1,end2);

}

list<node\*> que;

void visit(node \*t)

{

que.push\_back(t);

while(!que.empty())

{

node \*temp=que.front();

cout<<temp->c;

if(temp->lchild!=NULL)

que.push\_back(temp->lchild);

if(temp->rchild!=NULL)

que.push\_back(temp->rchild);

que.pop\_front();

}

cout<<endl;

}

void last(node \*t)

{

if(t==NULL)

return;

if(t->lchild!=NULL)

last(t->lchild);

if(t->rchild!=NULL)

last(t->rchild);

cout<<t->c;

}

int main()

{

node \*tree;

int length;

while(1==1)

{

cout<<"pre"<<endl;

cin>>pre;

cout<<"mid"<<endl;

cin>>mid;

length=strlen(pre);

build(tree,0,length-1,0,length-1);

//last(tree);

if(!que.empty())

que.clear();

visit(tree);

}

return 0;

}

1. 用SQL语句写出：C评级的用户中，1期30+逾期率（按照金额算及按笔算，提示：两个表先连接）（填空题，每空5分，共20分）

Select sum(duedate\_30)/sum(principal) as ‘金额金额逾期率’

Sun(case when duedate\_30>0\_\_\_\_\_\_)/count(1) as ‘笔数逾期率’ from （select a user\_id ,a.principal,isnull(a.duedate\_30,\_\_\_\_\_\_)as duedate\_30,b.credit\_level from risk.vintage a\_\_\_\_\_\_risk.credit b \_\_\_\_\_\_\_\_\_\_\_\_where b.credit\_level = ‘C’）

1. 用python读入C盘路径下的test.csv文件，命名为df,再导出到D盘的output.csv,需要输入\_\_\_\_\_\_csv\_\_\_\_\_\_模块，读入时用\_\_\_\_\_\_read()\_\_\_\_\_\_\_\_语句，导出时用to\_csv语句。
2. 用python提取身份证号码idn中的生日信息，idn是字符串类型，例如idn=’310125199102181564’，生日截取为’19910218’，并转化为’1991-02-18’的格式，请选出正确的选项 B

(A) pd.concat(idn[6:10],’-’,idn[10:12],’-’,idn[12:14])

(B) idn[6:10]+’-’+idn[10:12]+’-’+idn[12:14]

(C) pd.concat(idn.ix[6:10],’-’,idn.ix[10:12],’-’,idn.ix[12:14])

(D) idn.ix[6:10]+’-’+idn.ix[10:12]+’-’+idn.ix[12:14]

1. 用python编写函数evaluate(x),实现的功能为 数值型变量x,范围为[0,100],如果x在[0,60]，返回不及格；如果x在[60,80]内，则返回‘合格’，如果x在[80,90]之前，则返回’良好’；如果x在[90,100]内，如果则返回优秀，超出范围，返回’错误’

\_\_\_\_\_\_\_\_\_\_\_

If(x>=0)&(x<60):

return (‘不合格’)

\_\_\_(x>=60)&(x<80)

return (‘合格’)

\_\_\_(x>=80)&(x<90)

return (‘良好’)

\_\_\_(x>=90)&(x<=100)

return (‘优秀’)

\_\_\_\_\_\_\_\_\_\_\_\_\_

return(‘错误’)

1. 简述：Django Models 多条件查询 以及Q/F查询

**1、传参数**

models.UserInfo.objects.filter(id=3,name='alex')

**2、传字典**

需要注意的是,传入字典时,字典前面需要加**\*\*** ，记为字典

dic = {'id':123,'name':'alex'} models.UserInfo.objects.filter(\*\*dic)

所以我们可以在在捕捉用户输入后,将输入构造成字典,然后将字典当做参数传入查询

**3、传Q对象,构造搜索条件**

a、在 filter() 等函式中关键字参数彼此之间都是 **"AND" 关系**。如果你要执行更复杂的查询(比如，  
实现筛选条件的 **OR 关系**)，可以使用 Q 对象。  
b、Q对象包括 AND 关系 和OR 关系  
c、Q 对象可以用 & 和 | 运算符进行连接。当某个操作连接两个 Q 对象时，**就会产生一个新的等**  
**价的 Q 对象。**

如：下面这段语句就产生了一个 Q ，这是用 "OR" 关系连接

Q(question\_\_startswith='Who') | Q(question\_\_startswith='What')

d、每种查询函式(比如 filter(), exclude(), get()) 除了能接收关键字参数以外，也能以位置参数的  
形式接受一个或多个 Q 对象。如果你给查询函式传递了多个 Q 对象，那么它们彼此间都是  
**"AND" 关系。**例如：

Poll.objects.get( Q(question\_\_startswith='Who'), Q(pub\_date=date(2005, 5, 2)) | Q(pub\_date=date(2005, 5, 6)) )

**e: filter() 等函数 可以接受 Q对象和条件参数，但Q对象必须放在 条件参数前面**

**4、Q对象使用实例：**

首先还是需要导入模块

from django.db.models import Q

**传入条件进行查询:**

q1 = Q() q1.connector = 'OR' #连接方式 q1.children.append(('id', 1)) q1.children.append(('id', 2)) q1.children.append(('id', 3)) models.Tb1.objects.filter(q1)

合并条件进行查询:

con = Q() q1 = Q() q1.connector = 'OR' q1.children.append(('id', 1)) q1.children.append(('id', 2)) q1.children.append(('id', 3)) q2 = Q() q2.connector = 'OR' q2.children.append(('status', '在线')) con.add(q1, 'AND') con.add(q2, 'AND') models.Tb1.objects.filter(con)

F对象

可以使用模型的字段A与字段B进行比较，如果A写在了等号的左边，则B出现在等号的右边，需要通过F对象构造

list.filter(bread\_\_gte=F('bcommet'))

django支持对F()对象使用算数运算

list.filter(bread\_\_gte=F('bcommet') \* 2)

F()对象中还可以写作“模型类\_\_列名”进行关联查询

list.filter(isDelete=F('heroinfo\_\_isDelete'))

对于date/time字段，可与timedelta()进行运算

list.filter(bpub\_date\_\_lt=F('bpub\_date') + timedelta(days=1))

Q对象

过滤器的方法中关键字参数查询，会合并为And进行

需要进行or查询，使用Q()对象

Q对象(django.db.models.Q)用于封装一组关键字参数，这些关键字参数与“比较运算符”中的相同

from django.db.models import Q list.filter(Q(pk\_ \_lt=6))

Q对象可以使用&（and）、|（or）操作符组合起来

当操作符应用在两个Q对象时，会产生一个新的Q对象

list.filter(pk\_ \_lt=6).filter(bcommet\_ \_gt=10) list.filter(Q(pk\_ \_lt=6) | Q(bcommet\_ \_gt=10))

使用~（not）操作符在Q对象前表示取反

list.filter(~Q(pk\_\_lt=6))

可以使用&|~结合括号进行分组，构造做生意复杂的Q对象

过滤器函数可以传递一个或多个Q对象作为位置参数，如果有多个Q对象，这些参数的逻辑为and

过滤器函数可以混合使用Q对象和关键字参数，所有参数都将and在一起，Q对象必须位于关键字参数的前面

1. 用列表函数写出快排或其他方式

def quick\_sort(array, left, right):

if left >= right:

return

low = left

high = right

key = array[low]

while left < right:

while left < right and array[right] > key:

right -= 1

array[left] = array[right]

while left < right and array[left] <= key:

left += 1

array[right] = array[left]

array[right] = key

quick\_sort(array, low, left - 1)

quick\_sort(array, left + 1, high)

1. 写一个函数要求实现：输入一个日期，返回......（没有啦！）
2. 面向对象编程中有组合和继承的方法实现新的类，假设我们手头上只有栈类，请用组合方式使用‘栈’（LIFO）来实现‘队列’（FIFO），完成以下代码。（如果认为不需要添加任何代码可以不写）

#stack 类实现了栈，并提供pop和push方法。

Import stack

class Queue(object):

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

self.s1 = stack()

self.s2 = stack()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#在此处添加代码

def pop(self):

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#在此处添加代码

def push(self,e):

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#在此处添加代码

1. django的model中，我想设计一个Person的公共类，但又不想为这个类创建表。怎么做，假设有两个字段。

Name = models.CharField(max\_length=50)

Age = models.PositiveIntergerField()

请写出模型 class Persion

1. 10240转成16进制

2800

1. 给定一个数组arr，在O(n)时间内，求第二大的数

def FindSecMax(arr){

int count = data.length;

int maxnumber = data[0];

int sec\_max = Integer.MIN\_VALUE;//最小负整数

for (int i=1;i<count;i++){//循环遍历一遍

if (data[i]>maxnumber){//当前数组元素的值是比最大的数大

sec\_max=maxnumber;//更新第二大的数值为先前的最大值

maxnumber = data[i];//更新先前的最大值为当前数组元素的值

}

else{

if (data[i]>sec\_max){//当前数组元素的值是比最大值小则与第二大值比较

sec\_max = data[i];//如果当前数组元素的值比第二大值大，则更新第二大值

}

}

}

return sec\_max;

}

1. 有一张表Email,写一个SQL语句，找出email重复的行

|  |  |
| --- | --- |
| Id | Email |
| 1 | [abc@gmail.com](mailto:abc@gmail.com) |
| 2 | [abcd@gmail.com](mailto:abcd@gmail.com) |
| 3 | [abc@gmail.com](mailto:abc@gmail.com) |

1. 在python中，list，tuple，dict，set有什么特点

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | 可变性 | 存在形式 | 可重复性 | 有序性 | 其他特点 |
| list | 可修改元素 | 值的形式[1,2] | 值可以重复 | 有序 | 相比dict拥有占用内存小的特点，常用于堆栈等的处理 |
| tuple | 不可以对元素进行修改，但元素自身可以变化，例如元素为list时 | 值的形式(1,2) | 值可以重复 | 有序 | 本身不可变，相对比较稳定 |
| dict | 键不可变，值可变 | 键值对形式{1:1,2:2} | 键不可以重复，值可以重复 | 无序 | 符合用内存转换思想，常用于查找 |
| set | 可以修改元素 | 值的形式{1,2,3,4,5,6} | 值不可以重复 | 无序 | 常用于判断值是否存在 |

1. a=set(‘abcde’)

b=set(‘defg’)

1）求a中不包含在集合b中的元素

2）求集合a,b中都包含的元素

3）求集合a,b所有的元素

4）不同时包含于a,b的元素

(1)

test1=a.difference(b)#计算a与b的差集

print(test1)

(2)

test2=a.intersection(b)#计算a,b的交集test2

print(test2)

(3)

test3=a.union(b)

print(test3)

(4)

test4=a^b

test4

1. python使用符号\_\_#\_单行注释,多行注释使用\_\_\_’’’ ’’’\_\_符号.
2. 目前linux下常用的专用于python包的安装管理工具\_pip\_.

所有的Python对象都有三个属性: \_\_身份,类型和值

1. Python序列类型包括\_\_\_列表\_\_\_ , \_\_\_\_元组\_\_\_\_,\_\_\_\_字符串\_\_\_\_三种.

\_\_\_\_\_字典\_\_\_\_是python中唯一的映射类型.

1. Python提供了两个对象身份比较操作符\_\_\_\_\_和\_\_\_\_\_来测设两个变量是否指向同一个对象,对可以通过内建的函数\_\_\_\_\_\_\_来测试对象的身份.
2. 每一个python对象使用内置属性\_\_\_\_\_\_\_来存储所有的实例属性,该属性的数据类型是\_\_\_\_\_.
3. 设S= ‘abcdefg’, 则s[3]值是\_\_d\_\_\_,s[3:5]值是\_\_de\_\_\_\_.

在类定义中,python使用函数修饰符\_staticmethod\_表示静态方法,修饰符\_\_\_\_表示属性

1. 判断题

1.python是把源代码编译成汇编代码在执行的 ( 错 )

2.python 程序内存管理有解释器负责 ( 对 )

3.print函数打印对象实际上调用的是repr函数显示对象 ( 对 )

4.python以{}符号作为语句快的划分 ( 错 )

5.python对于函数内没有处理的异常会交由上层函数处理 ( 对 )

6.python不允许动态创建对象属性 ( 对 )

1. python支持的数据类型 ( D )

A. char

B. int

C. float

D. list

1. 关于Python内存管理,下列说法错误的是 ( B )

A.变量不必事先声明

B.变量无需先创建和赋值而直接使用

C.变量无需指定类型

D.可以使用del释放资源

1. 下列语句正确的是 ( C )

A.Max = x>y?x:y

B.If(x>y) print(x)

C.Min = x if x < y else y

1. 关于字符串下列说法错误的是 ( A )

A. 字符应该视为长度为1的字符串

B. 字符串以\0标志字符串的结束

c. 既可以用但引号,也可以用双引号创建字符串

1. 下面代码的输出

Class Singleton:

\_instance = None

def \_\_new\_\_(cls,\*args,\*\*kwargs):

print(“New”)

If cls.\_instance is None:

print(“Create”)

cls.\_instance = super().\_\_new\_\_(cls,\*args,\*\*kwargs)

return cls.\_instance

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

print(“Initialize”)

self.prop = None

Singleton()

Singleton()

单例实现需要改进的地方

1. （简答题）下列代码的输出

a = 1

def fun(a):

a = 2

fun(a)

b = [ ]

def fun(b):

b.append(1)

fun(b)

print a,b

答案：1 [1]

1. （简答题）下列代码的输出

import copy

a = [1,2,3,4,['a','b']]

b = a

c = copy.copy(a)

d = copy.deepcopy(a)

a.append(S)

a[4].append('c')

print 'a = ' . a

print 'b = ' . b

print 'c = ' . c

print 'd = ' . d

1. （简答题）下面代码会输出什么

def f (x,l=[ ]):

for i in range(x):

l.append(i\*i)

print l

f(2)

[0]

[0, 1]

f(3,[3,2,1])

[3, 2, 1, 0]

[3, 2, 1, 0, 1]

[3, 2, 1, 0, 1, 4]

f(3)

[0, 1, 0]

[0, 1, 0, 1]

[0, 1, 0, 1, 4]

1. （简答题）这两个参数是什么意思：\*args，\*\*kwargs？我们为什么要使用它们？

如果我们不确定要往函数中传入多少个参数，或者我们想往函数中以列表和元组的形式传参数时，那就使要用\*args；   
如果我们不知道要往函数中传入多少个关键词参数，或者想传入字典的值作为关键词参数时，那就要使用\*\*kwargs

1. SQL测试:

数据库有2张表

表1: risk.credit为额度记录表,字段为:用户id;记录时间;模型评分;信用额度;拒绝理由;信用评级

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| userid | Inserttime | Creditscore | Credit\_amount | Audit\_reason | Credit\_level |
| 203568 | 2017-05-01 15::31 | 620 | 10000 |  | C |
| 358966 | 2017-05-02 16:16 | 650 | 12000 |  | C |
| 158793 | 2017-05-22 09:21 | 680 | 18000 |  | B |
| 358769 | 2017-05-02 21:05 | 580 | 0 | 381 | D |
| 158736 | 2017-05-02 01:37 | 610 | 0 | 380 | C |
| 458932 | 2017-05-01 19:49 | 720 | 20000 |  | A |
| 514893 | 2017-05-02 17:23 | 630 | 0 | 484 | C |
| 289731 | 2017-05-21 14:02 | 700 | 0 | 373 | A |

表2:risk vintage为逾期表,字段分别为:用户id;成交时间;成交金额;1期5+逾期金额;1期10+逾期金额;1期30+逾期金额

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| User\_id | Auditing\_date | Principal | Duedate\_5 | Duedate\_10 | Duedate\_30 |
| 158793 | 2017-05-22 09:21 | 16000 | null | null | null |
| 458932 | 2017-05-01 19:49 | 15000 | null | null | null |
| 203568 | 2017-05-01 15::31 | 8000 | null | 900 | 900 |
| 358721 | 2017-10-01 21:56 | 9000 | null | null | 950 |
| 514893 | 2017-05-02 17:23 | 12000 | 1000 | 1000 | 1000 |

1)用sql语句筛选出:’2017-05-01’以后的拒绝理由的类型(不重复)(5分) C

1. Select distinct audit\_reason from risk credit where inserttime>=’ 2017-05-01’
2. Select distinct audit\_reason from risk credit having inserttime>=’ 2017-05-01’
3. Select unique audit\_reason from risk credit where inserttime>=’ 2017-05-01’
4. Select unique audit\_reason from risk credit having inserttime>=’ 2017-05-01’

2)计算有额度用户中,每个评级的平均额度,以下sql内建函数或语句肯定用不到或有错 误的是: D

(A) avg内建函数

(B) case when … end语句

(C) group by语句

(D) sumif内建函数