**尚硅谷Python 学科面试宝典**

如答案有误或其他问题，请通过以下方式联系更正，谢谢

WECHAT：Kyasing

GITHUB：<http://github.com/jiaxingzhao>

[1. Python基础 33](#_Toc525304390)

[1.1. 基础语法 33](#_Toc525304391)

[1.1.1. 输入与输出 33](#_Toc525304392)

[1) 代码中要修改不可变数据会出现什么问题? 抛出什么异常? 33](#_Toc525304393)

[2) a=1,b=2,不用中间变量交换a 和b 的值？ 33](#_Toc525304394)

[3) print 调用Python 中底层的什么方法? 34](#_Toc525304395)

[4) 在Python中如何使用多进制数字？ 34](#_Toc525304396)

[5) 简述你对input()函数的理解? 34](#_Toc525304397)

[1.1.2. 条件与循环 34](#_Toc525304398)

[1) range 和xrange 的区别？ 34](#_Toc525304399)

[2) 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？ 34](#_Toc525304400)

[3) 以下Python 程序的输出？ 35](#_Toc525304401)

[1.1.3. 文件操作 35](#_Toc525304402)

[1) 4G 内存怎么读取一个5G 的数据？ 35](#_Toc525304403)

[2) 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 35](#_Toc525304404)

[3) read、readline 和readlines 的区别? 36](#_Toc525304405)

[1.1.4. 异常 36](#_Toc525304406)

[1) 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？ 36](#_Toc525304407)

[2) 介绍一下except 的作用和用法？ 36](#_Toc525304408)

[1.1.5. 模块与包 36](#_Toc525304409)

[1) 常用的Python 标准库都有哪些？ 36](#_Toc525304410)

[2) 赋值、浅拷贝和深拷贝的区别？ 37](#_Toc525304411)

[3) Python 里面如何生成随机数？ 37](#_Toc525304412)

[4) 输入某年某月某日，判断这一天是这一年的第几天？(可以用Python 标准库) 37](#_Toc525304413)

[5) 说明一下os.path 和sys.path 分别代表什么？ 38](#_Toc525304414)

[6) Python 中的os 模块常见方法？ 38](#_Toc525304415)

[7) Python 的sys 模块常用方法？ 38](#_Toc525304416)

[8) unittest 是什么？ 39](#_Toc525304417)

[9) 模块和包是什么？ 39](#_Toc525304418)

[1.1.6. Python 特性 39](#_Toc525304419)

[1) Python 是强语言类型还是弱语言类型？ 39](#_Toc525304420)

[2) 谈一下什么是解释性语言，什么是编译性语言? 39](#_Toc525304421)

[3) Python 中有日志吗?怎么使用? 40](#_Toc525304422)

[4) Python 是如何进行类型转换的？ 40](#_Toc525304423)

[5) Python2 与Python3 的区别？ 40](#_Toc525304424)

[6) 关于Python 程序的运行方面，有什么手段能提升性能？ 40](#_Toc525304425)

[7) Python 中的作用域？ 40](#_Toc525304426)

[8) 什么是Python？ 40](#_Toc525304427)

[9) 什么是Python 自省？ 41](#_Toc525304428)

[10) 什么是Python 的命名空间？ 41](#_Toc525304429)

[11) 你所遵循的代码规范是什么？请举例说明其要求？ 41](#_Toc525304430)

[12) 解释一下 Python 中的三元运算子 41](#_Toc525304431)

[13) “猴子补丁”（monkey patching）指的是什么？ 41](#_Toc525304432)

[1.1.7. Linux 42](#_Toc525304433)

[1) Linux 的基本命令（怎么区分一个文件还是文件夹） 42](#_Toc525304434)

[2) 日志以什么格式，存放在哪里？ 42](#_Toc525304435)

[3) Linux 查看某个服务的端口? 42](#_Toc525304436)

[4) ubuntu 系统如何设置开机自启动一个程序? 42](#_Toc525304437)

[5) 在linux 中find 和grep 的区别 42](#_Toc525304438)

[6) Linux 重定向命令有哪些？有什么区别？ 43](#_Toc525304439)

[7) 软连接和硬链接的区别？ 43](#_Toc525304440)

[8) 10 个常用的Linux 命令？ 43](#_Toc525304441)

[9) Linux 关机命令有哪些？ 43](#_Toc525304442)

[10) 使用Linux命令查询file1中空行所在的行号 44](#_Toc525304443)

[11) Linux查看内存、磁盘存储、io 读写、端口占用、进程等命令 44](#_Toc525304444)

[12) unix 进程间通信方式 44](#_Toc525304445)

[1.1.8. Git命令 45](#_Toc525304446)

[1) git 合并文件有冲突，如何处理? 45](#_Toc525304447)

[2) git 中 merge 和 rebase命令 的区别 45](#_Toc525304448)

[3) reset 与 rebase, pull 与 fetch 的区别 45](#_Toc525304449)

[4) git如何实现v1.0 、v2.0 等版本的管理？ 45](#_Toc525304450)

[5) 什么是gitlab？ 45](#_Toc525304451)

[6) git中 ".gitignore"文件的作用? 45](#_Toc525304452)

[7) 什么是敏捷开发？ 45](#_Toc525304453)

[1.2. 数据类型 46](#_Toc525304454)

[1.2.1. 字典（ Dict ） 46](#_Toc525304455)

[1) 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？ 46](#_Toc525304456)

[2) 怎样获取字典中所有键的列表？ 46](#_Toc525304457)

[3) 说一下字典和json 的区别？ 46](#_Toc525304458)

[4) 存入字典里的数据有没有先后排序？ 46](#_Toc525304459)

[5) 字典推导式？ 46](#_Toc525304460)

[6) 字典有哪些常用的内置方法？ 46](#_Toc525304461)

[7) 对于字典aaa={“a”:10, “b”:34, “c”:11, “d”:22}，写出程序对调key和value，产生新的字典 47](#_Toc525304462)

[1.2.2. 字符串 （ String ） 47](#_Toc525304463)

[1) 怎样将字符串转换大小写？ 47](#_Toc525304464)

[2) 如何理解Python 中字符串中的\字符？ 47](#_Toc525304465)

[3) 请反转字符串“aStr”? 47](#_Toc525304466)

[4) 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 47](#_Toc525304467)

[5) 请按alist 中元素的age 由大到小排序 48](#_Toc525304468)

[6) 字符串格式化 % 和 format的区别 48](#_Toc525304469)

[7) 给定一个字符串作为输入，将其中的数字用数字前近邻的字母序列进行替换，字母序列重复的次数等于该数字，最后将字符串输出： 输入：”af1an2ti3” 输出：”afanantititi” 48](#_Toc525304470)

[8) 写尽可能多的str对象的方法 49](#_Toc525304471)

[9) 怎么移除一个字符串中的前导空格？ 50](#_Toc525304472)

[10) 如何查询和替换一个文本中的字符串 50](#_Toc525304473)

[1.2.3. 列表 （ List ） 50](#_Toc525304474)

[1) 如何以就地操作方式打乱一个列表的元素？ 50](#_Toc525304475)

[2) 下面代码的输出结果将是什么？ 50](#_Toc525304476)

[3) 写一个列表生成式，产生一个公差为11 的等差数列 50](#_Toc525304477)

[4) 请写出一段Python 代码实现删除一个list 里面的重复元素? 50](#_Toc525304478)

[5) 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 51](#_Toc525304479)

[6) 有如下数组list = range(10)我想取以下几个数组，应该如何切片？ 51](#_Toc525304480)

[7) 下面这段代码的输出结果是什么？请解释？ 51](#_Toc525304481)

[8) 将以下3 个函数按照执行效率高低排序 52](#_Toc525304482)

[9) 获取1~100 被6 整除的偶数？ 52](#_Toc525304483)

[1.2.4. 元祖 （ Tuple ） 53](#_Toc525304484)

[1) 现有两元祖 (('a'),('b'),('c'),('d') ) ,请使用Python中的匿名函数生成列表 [ {'a':'c'},{'b':'d'}] 53](#_Toc525304485)

[2) 在Python中数组和元组的区别是什么？ 53](#_Toc525304486)

[1.2.5. 集合 （ Set ） 53](#_Toc525304487)

[1) 快速去除列表中的重复元素 53](#_Toc525304488)

[1.3. 网络协议 53](#_Toc525304489)

[1) URL的格式是什么？ 53](#_Toc525304490)

[2) http协议头中，keep-alive字段有什么作用？ 53](#_Toc525304491)

[3) 请求头里都可以得到什么内容？ 54](#_Toc525304492)

[4) 简述TCP 和UDP 的区别以及优缺点? 54](#_Toc525304493)

[5) 描述用浏览器访问www.baidu.com 的过程 54](#_Toc525304494)

[6) POST和GET请求的区别? 55](#_Toc525304495)

[7) HTTP与HTTPS协议有什么区别？ 55](#_Toc525304496)

[8) cookie 和session 的区别？ 56](#_Toc525304497)

[9) ARP协议 56](#_Toc525304498)

[10) 三次握手，四次挥手 57](#_Toc525304499)

[11) 代理和反向代理 57](#_Toc525304500)

[2. Python高级 57](#_Toc525304501)

[2.1. 元类 57](#_Toc525304502)

[1) Python 中类方法、类实例方法、静态方法有何区别？ 57](#_Toc525304503)

[2.2. 内存管理与垃圾回收机制 57](#_Toc525304504)

[1) Python 的内存管理机制及调优手段？ 57](#_Toc525304505)

[2) 内存泄露是什么？如何避免？ 58](#_Toc525304506)

[2.3. 函数 59](#_Toc525304507)

[2.3.1. 函数参数 59](#_Toc525304508)

[1) 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？ 59](#_Toc525304509)

[2) 对缺省参数的理解？ 59](#_Toc525304510)

[3) 为什么函数名字可以当做参数用? 59](#_Toc525304511)

[4) Python 中pass 语句的作用是什么？ 59](#_Toc525304512)

[5) 有这样一段代码，print c 会输出什么，为什么？ 59](#_Toc525304513)

[2.3.2. 内建函数 60](#_Toc525304514)

[1) map 函数和reduce 函数？ 60](#_Toc525304515)

[2) 递归函数停止的条件？ 60](#_Toc525304516)

[3) 回调函数，如何通信的? 60](#_Toc525304517)

[4) 解释 Python 中的 help() 和 dir() 函数 60](#_Toc525304518)

[5) Python 主要的内置数据类型都有哪些？ print dir( ‘a ’) 的输出？ 61](#_Toc525304519)

[6) print(list(map(lambda x: x \* x, [y for y in range(3)])))的输出？ 61](#_Toc525304520)

[7) hasattr() getattr() setattr() 函数使用详解？ 61](#_Toc525304521)

[8) 一句话解决阶乘函数？ 61](#_Toc525304522)

[2.3.3. Lambda 61](#_Toc525304523)

[1) 什么是lambda 函数？ 有什么好处？ 61](#_Toc525304524)

[2) 下面这段代码的输出结果将是什么？请解释 61](#_Toc525304525)

[2.4. 设计模式 62](#_Toc525304526)

[2.4.1. 单例 62](#_Toc525304527)

[1) 全局变量和单例模式有什么区别 62](#_Toc525304528)

[2) 请手写一个单例 62](#_Toc525304529)

[3) 单例模式的应用场景有哪些？ 63](#_Toc525304530)

[2.4.2. 装饰器 63](#_Toc525304531)

[1) 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？ 63](#_Toc525304532)

[2) 解释一下什么是闭包? 手写一个闭包函数 63](#_Toc525304533)

[3) 函数装饰器有什么作用？ 63](#_Toc525304534)

[2.4.3. 生成器 63](#_Toc525304535)

[1) 生成器、迭代器的区别？ 63](#_Toc525304536)

[2) X 是什么类型？ 64](#_Toc525304537)

[3) 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? 64](#_Toc525304538)

[4) Python 中yield 的用法？ 64](#_Toc525304539)

[5) 解释生成器（generator）与函数的不同 64](#_Toc525304540)

[2.5. 面向对象 64](#_Toc525304541)

[2.5.1. 类 64](#_Toc525304542)

[1) 解释一下 Python 中的继承 64](#_Toc525304543)

[2) 请写一个Python类，要求包含public/private方法及变量的声明，同时定义一个静态方法 65](#_Toc525304544)

[2.5.2. 对象 65](#_Toc525304545)

[1) 单下划线和双下划线在python的区别？ 65](#_Toc525304546)

[2) Python 中的可变对象和不可变对象？ 65](#_Toc525304547)

[3) Python 中is 和==的区别？ 66](#_Toc525304548)

[4) Python 的魔法方法 66](#_Toc525304549)

[5) 面向对象中怎么实现只读属性? 66](#_Toc525304550)

[6) 在类的方法定义中，' self ' 是怎样的一个参数 66](#_Toc525304551)

[7) 谈谈你对面向对象的理解？ 66](#_Toc525304552)

[8) 如何判断一个Python对象的类型？ 67](#_Toc525304553)

[2.6. 正则表达式 67](#_Toc525304554)

[1) Python 里match 与search 的区别？ 67](#_Toc525304555)

[2) Python 字符串查找和替换？ 67](#_Toc525304556)

[3) 用Python 匹配HTML g tag 的时候，<.\*> 和<.\*?> 有什么区别？ 67](#_Toc525304557)

[2.7. 系统编程 67](#_Toc525304558)

[1) select、poll、epoll各有哪些区别？ 67](#_Toc525304559)

[2) B/S和C/S的区别 68](#_Toc525304560)

[3) 在Python中如何实现多线程？ 68](#_Toc525304561)

[4) 谈谈你对多进程，多线程，以及协程的理解，项目是否用？ 68](#_Toc525304562)

[5) 进程池、线程池、连接池 69](#_Toc525304563)

[6) 什么是多线程竞争？ 69](#_Toc525304564)

[7) 解释一下什么是锁，有哪几种锁? 69](#_Toc525304565)

[8) 什么是死锁？ 69](#_Toc525304566)

[9) 什么是线程安全，什么是互斥锁？ 70](#_Toc525304567)

[10) 说说下面几个概念：同步，异步，阻塞，非阻塞? 70](#_Toc525304568)

[11) 什么是僵尸进程和孤儿进程？怎么避免僵尸进程? 70](#_Toc525304569)

[12) Python 中的进程与线程的使用场景? 70](#_Toc525304570)

[13) 线程是并发还是并行，进程是并发还是并行？ 71](#_Toc525304571)

[14) 并行（parallel）和并发（concurrency）？ 71](#_Toc525304572)

[15) 利用python执行shell脚本并取得返回结果 71](#_Toc525304573)

[16) IO 密集型和CPU 密集型区别？ 71](#_Toc525304574)

[2.8. 网络编程 71](#_Toc525304575)

[1) 怎么实现强行关闭客户端和服务器之间的连接? 71](#_Toc525304576)

[2) C10K问题 71](#_Toc525304577)

[3) 简述浏览器通过WSGI 请求动态资源的过程? 71](#_Toc525304578)

[2.9. Python编程题 72](#_Toc525304579)

[1) 有这样一个文本文件，他的路径为baseDir，他的名字test.txt，要求with方式打开，并打印每一行文本，并要求文件路径考虑跨平台问题 72](#_Toc525304580)

[2) 有两个磁盘文件A和B，各存放一行字母，要求把这两个文件中的信息合并（按字母顺序排列），输出到一个新文件C中 72](#_Toc525304581)

[3) 下面是一个单线程的代码，请改写成多线程的 72](#_Toc525304582)

[4) 根据描述，写出接口调用的请求代码 73](#_Toc525304583)

[3. 前端 74](#_Toc525304584)

[3.1. HTML 74](#_Toc525304585)

[1) 浏览器渲染原理解析 74](#_Toc525304586)

[2) iframe有那些缺点？ 74](#_Toc525304587)

[3) Label的作用是什么？是怎么用的？ 74](#_Toc525304588)

[4) HTML5的form如何关闭自动完成功能？ 74](#_Toc525304589)

[5) title与h1的区别、b与strong的区别、i与em的区别？ 74](#_Toc525304590)

[6) 每个HTML文件里开头都有个很重要的东西，Doctype，知道这是干什么的吗？ 75](#_Toc525304591)

[7) div+css的布局较table布局有什么优点？ 75](#_Toc525304592)

[8) img的alt与title有何异同？ strong与em的异同？ 75](#_Toc525304593)

[9) 知道的网页制作会用到的图片格式有哪些？ 75](#_Toc525304594)

[10) 在css/js代码上线之后开发人员经常会优化性能，从用户刷新网页开始，一次js请求一般情况下有哪些地方会有缓存处理？ 76](#_Toc525304595)

[11) 你如何理解HTML结构的语义化？ 76](#_Toc525304596)

[12) HTML5 Canvas 元素有什么用？ 76](#_Toc525304597)

[13) 如何在 HTML5 页面中嵌入音频? 76](#_Toc525304598)

[14) 如何在 HTML5 页面中嵌入视频？ 76](#_Toc525304599)

[3.2. CSS 76](#_Toc525304600)

[1) 水平垂直居中的方法 76](#_Toc525304601)

[2) 清除浮动元素的方法和各自的优缺点 77](#_Toc525304602)

[3) css实现自适应正方形 77](#_Toc525304603)

[4) position的值 77](#_Toc525304604)

[5) CSS优先级算法如何计算？ 78](#_Toc525304605)

[6) display有哪些值？说明他们的作用 78](#_Toc525304606)

[7) 为什么要初始化CSS样式 78](#_Toc525304607)

[8) position跟display、margin collapse、overflow、float这些特性相互叠加后会怎么样？ 78](#_Toc525304608)

[9) CSS优化、提高性能的方法有哪些？ 79](#_Toc525304609)

[10) display:inline-block 什么时候会显示间隙？ 79](#_Toc525304610)

[11) 什么是外边距重叠？重叠的结果是什么？ 79](#_Toc525304611)

[12) px和em的区别 79](#_Toc525304612)

[13) 行内元素和块级元素的具体区别是什么？行内元素的padding和margin可设置吗？ 79](#_Toc525304613)

[14) 知道css有个content属性吗？有什么作用？有什么应用？ 80](#_Toc525304614)

[3.3. JavaScript 80](#_Toc525304615)

[1) eval是做什么的？ 80](#_Toc525304616)

[2) 什么是window对象? 什么是document对象? 80](#_Toc525304617)

[3) null，undefined 的区别？ 80](#_Toc525304618)

[4) 事件是？IE与火狐的事件机制有什么区别？ 如何阻止冒泡？ 81](#_Toc525304619)

[5) JSON 的了解？ 81](#_Toc525304620)

[6) JS延迟加载的方式有哪些？ 82](#_Toc525304621)

[7) javascript的typeof返回哪些数据类型 82](#_Toc525304622)

[8) 例举3种强制类型转换和2种隐式类型转换? 82](#_Toc525304623)

[9) 数组方法pop() push() unshift() shift() 82](#_Toc525304624)

[10) 事件绑定和普通事件有什么区别？ 83](#_Toc525304625)

[11) IE和DOM事件流的区别 83](#_Toc525304626)

[12) window.onload 和document ready的区别 84](#_Toc525304627)

[13) ”==”和“===”的不同 84](#_Toc525304628)

[14) JavaScript是一门什么样的语言，它有哪些特点？ 84](#_Toc525304629)

[15) JavaScript的数据类型都有什么？ 84](#_Toc525304630)

[16) foo = foo||bar ，这行代码是什么意思？为什么要这样写？ 84](#_Toc525304631)

[17) 把两个数组合并，并删除第二个元素。 85](#_Toc525304632)

[18) 简述创建函数的几种方式 85](#_Toc525304633)

[19) Javascript如何实现继承？ 85](#_Toc525304634)

[20) 哪些操作会造成内存泄漏？ 85](#_Toc525304635)

[21) javascript 中的垃圾回收机制？ 86](#_Toc525304636)

[22) 如何在HTML中添加事件，几种方法？ 86](#_Toc525304637)

[23) 你如何优化自己的代码？ 86](#_Toc525304638)

[24) 简述readyonly与disabled的区别 86](#_Toc525304639)

[25) document.write和 innerHTML的区别 86](#_Toc525304640)

[26) 谈谈对this的理解 86](#_Toc525304641)

[3.4. jQuery 87](#_Toc525304642)

[1) jquery中如何将数组转化为json字符串，然后再转化回来？ 87](#_Toc525304643)

[2) 针对 jQuery 的优化方法？ 87](#_Toc525304644)

[3) JQuery一个对象可以同时绑定多个事件，这是如何实现的？ 87](#_Toc525304645)

[4) bind(), live(), delegate()的区别 88](#_Toc525304646)

[5) jQuery框架中$.ajax()的常用参数有哪些？写一个post请求并带有发送数据和返回数据的样例 88](#_Toc525304647)

[3.5. vue.js 88](#_Toc525304648)

[1) vue 双向绑定底层实现原理 88](#_Toc525304649)

[2) vue 虚拟DOM和react 虚拟DOM的区别 89](#_Toc525304650)

[3) vue组件通信 89](#_Toc525304651)

[4) 你如何评价vue 89](#_Toc525304652)

[5) 说说你对MVVM的理解 89](#_Toc525304653)

[6) 为什么选择vue 90](#_Toc525304654)

[7) 谈谈你对组件的理解 90](#_Toc525304655)

[4. 数据库 90](#_Toc525304656)

[4.1. MySQL 90](#_Toc525304657)

[1) Python 中操作MySQL 步骤 90](#_Toc525304658)

[2) 使用的MySQL版本号是多少？ 91](#_Toc525304659)

[3) SQL 的select 语句完整的执行顺序 91](#_Toc525304660)

[4) 说一下MySQL 数据库存储的原理？ 92](#_Toc525304661)

[5) 事务的特性？ 92](#_Toc525304662)

[6) 数据库索引？ 93](#_Toc525304663)

[7) 乐观锁与悲观锁 93](#_Toc525304664)

[8) 数据库怎么优化查询效率？ 94](#_Toc525304665)

[9) MySQL 集群的优缺点？ 94](#_Toc525304666)

[10) 你用的MySQL 是哪个引擎，各引擎之间有什么区别？ 94](#_Toc525304667)

[11) 数据库的优化？ 95](#_Toc525304668)

[12) SQL的优化 95](#_Toc525304669)

[13) MySQL 数据库如何分区、分表？ 95](#_Toc525304670)

[14) 如何对查询命令进行优化？ 96](#_Toc525304671)

[15) SQL 注入是如何产生的，如何防止？ 97](#_Toc525304672)

[16) 数据库范式 97](#_Toc525304673)

[17) NoSQL 和关系数据库的区别？ 97](#_Toc525304674)

[18) MySQL 数据库中怎么实现分页？ 98](#_Toc525304675)

[19) SQL 语句怎么看效率？ 98](#_Toc525304676)

[20) 优化数据库？提高数据库的性能？ 98](#_Toc525304677)

[21) 提取数据库中倒数10 条数据？ 99](#_Toc525304678)

[22) 数据库负载均衡 99](#_Toc525304679)

[23) 存储过程和函数的区别? 99](#_Toc525304680)

[24) 关系型数据库的事务隔离级别有哪些，分别有什么影响？ 100](#_Toc525304681)

[25) MVCC是什么？ 100](#_Toc525304682)

[26) 数据库表增加了新字段更新问题 101](#_Toc525304683)

[27) 有一个数据库表为student，字段有name（姓名），age（年龄），sex（性别），请按照以下要求编写SQL语句： 101](#_Toc525304684)

[28) 列出几种常见的关系型数据库和非关系型数据库？ 101](#_Toc525304685)

[4.2. Redis 102](#_Toc525304686)

[1) Redis是什么？ 102](#_Toc525304687)

[2) Redis可以用来做什么？ 102](#_Toc525304688)

[3) Redis的优缺点是什么？ 102](#_Toc525304689)

[4) 为什么redis需要把所有数据放到内存中？ 102](#_Toc525304690)

[5) Redis 中list 底层实现有哪几种？有什么区别？ 102](#_Toc525304691)

[6) Redis五大数据结构，及在项目中的应用 103](#_Toc525304692)

[7) 怎样解决数据库高并发的问题？ 103](#_Toc525304693)

[8) Redis 集群实现？ 103](#_Toc525304694)

[9) Redis 的并发竞争问题怎么解决？ 104](#_Toc525304695)

[10) Redis 和MongoDB 的优缺点? 104](#_Toc525304696)

[11) Redis 基本类型、相关方法 104](#_Toc525304697)

[12) LPOP和BLPOP的区别? 105](#_Toc525304698)

[13) Redis的有一些包含SCAN关键字的命令是干嘛的? SCAN返回的数据量是固定的吗? 105](#_Toc525304699)

[14) Redis中的Lua有没有使用过? 可以用来做什么? 为什么可以这么用? 105](#_Toc525304700)

[15) Redis的Pipeline是用来干什么的? 105](#_Toc525304701)

[16) Redis中List, HashTable都⽤用到了ZipList, 为什么会选择它? 105](#_Toc525304702)

[17) Redis 的事务？ 105](#_Toc525304703)

[18) Redis 的使用场景有哪些？ 106](#_Toc525304704)

[19) Redis 默认端口，默认过期时间，Value 最多可以容纳的数据长度？ 106](#_Toc525304705)

[20) redis是否可以做持久化？ 106](#_Toc525304706)

[21) redis的过期策略 106](#_Toc525304707)

[22) redis的分布式锁实现 107](#_Toc525304708)

[23) 发布者与订阅者、消息队列 107](#_Toc525304709)

[24) Redis 有多少个库？ 108](#_Toc525304710)

[25) Redis缓存击穿、缓存雪崩、缓存重建 108](#_Toc525304711)

[26) 秒杀场景如何解决？ 108](#_Toc525304712)

[4.3. MongoDB 109](#_Toc525304713)

[1) MongoDB是什么？ 109](#_Toc525304714)

[2) 数据库的一些基本操作命令（列举一些常用命令即可）？ 110](#_Toc525304715)

[3) Python 中调用mongo 数据库的包叫什么？ 110](#_Toc525304716)

[4) MongoDB 成为优秀的NoSQL 数据库的原因是什么? 110](#_Toc525304717)

[5) 分析器在MongoDB 中的作用是什么? 110](#_Toc525304718)

[6) 怎么查看MongoDB 正在使用的链接? 111](#_Toc525304719)

[7) MySQL 与MongoDB 本质之间最基本的差别是什么 111](#_Toc525304720)

[8) 使用MongoDB 的优点 111](#_Toc525304721)

[9) MongoDB支持存储过程吗？如果支持的话，怎么用？ 111](#_Toc525304722)

[5. 爬虫 111](#_Toc525304723)

[5.1. 常用库与模块 111](#_Toc525304724)

[1) User-Agent如何判断浏览器类型，火狐怎么显示，Chrome怎么显示 111](#_Toc525304725)

[2) 列举您使用过的Python 网络爬虫所用到的网络数据包? 112](#_Toc525304726)

[3) 列举您使用过的Python 网络爬虫所用到的解析数据包? 112](#_Toc525304727)

[4) 爬取数据后使用哪个数据库存储数据的，为什么？ 112](#_Toc525304728)

[5) 你用过的爬虫框架或者模块有哪些？谈谈他们的区别或者优缺点？ 113](#_Toc525304729)

[6) 写爬虫是用多进程好？还是多线程好？ 为什么？ 114](#_Toc525304730)

[7) 常见的反爬虫和应对方法？ 114](#_Toc525304731)

[8) 验证码的解决? 115](#_Toc525304732)

[9) 写一个邮箱地址的正则表达式？ 115](#_Toc525304733)

[10) 编写过哪些爬虫中间件？ 115](#_Toc525304734)

[11) “极验”滑动验证码如何破解？ 116](#_Toc525304735)

[12) 爬的那些内容数据量有多大，多久爬一次，爬下来的数据是怎么存储？ 116](#_Toc525304736)

[13) cookie 过期的处理问题？ 116](#_Toc525304737)

[14) HTTPS 有什么优点和缺点? 116](#_Toc525304738)

[15) HTTPS 是如何实现安全传输数据的? 117](#_Toc525304739)

[16) TTL，MSL，RTT？ 117](#_Toc525304740)

[17) 谈一谈你对Selenium 和PhantomJS 了解? 118](#_Toc525304741)

[18) 代理IP 里的“透明”“匿名”“高匿”分别是指？ 118](#_Toc525304742)

[19) requests 返回的content 和text 的区别？ 118](#_Toc525304743)

[20) robots 协议 119](#_Toc525304744)

[21) 为什么requests 请求需要带上header？ 119](#_Toc525304745)

[22) dumps、loads 与dump、load 的区别？ 119](#_Toc525304746)

[23) requests 使用小技巧 119](#_Toc525304747)

[24) IP 存放在哪里？怎么维护IP？对于封了多个ip 的，怎么判定IP 没被封？ 120](#_Toc525304748)

[25) 怎么获取加密的数据？ 120](#_Toc525304749)

[26) 假如每天爬取量在5、6 万条数据，一般开几个线程，每个线程ip 需要加锁限定吗？ 120](#_Toc525304750)

[27) 怎么监控爬虫的状态 120](#_Toc525304751)

[5.2. Scrapy 121](#_Toc525304752)

[1) 描述下scrapy 框架运行的机制？ 121](#_Toc525304753)

[2) 谈谈你对Scrapy 的理解？ 121](#_Toc525304754)

[3) 怎么样让Scrapy框架发送一个post 请求（具体写出来） 121](#_Toc525304755)

[4) 怎么判断网站是否更新？ 122](#_Toc525304756)

[5) 图片、视频爬取怎么绕过防盗连接，或者说怎么获取正确的链接地址？ 122](#_Toc525304757)

[6) 爬虫向数据库存数据开始和结束都会发一条消息，是scrapy 哪个模块实现的？ 122](#_Toc525304758)

[7) 怎么设置深度爬取? 122](#_Toc525304759)

[8) 爬取下来的数据如何去重，说一下具体的算法依据 122](#_Toc525304760)

[5.3. Scrapy-Redis 122](#_Toc525304761)

[1) scrapy 和scrapy-redis 有什么区别？为什么选择redis 数据库？ 122](#_Toc525304762)

[2) 分布式爬虫主要解决什么问题？ 123](#_Toc525304763)

[3) 什么是分布式存储？ 123](#_Toc525304764)

[4) 你所知道的分布式爬虫方案有哪些？ 123](#_Toc525304765)

[5) 在爬取的时候遇到某些内容字段缺失怎么判断及处理？ 124](#_Toc525304766)

[6. WEB框架 124](#_Toc525304767)

[6.1. Django 124](#_Toc525304768)

[1) 你对 Django 的认识？ 124](#_Toc525304769)

[2) Django HTTP 请求的处理流程? 125](#_Toc525304770)

[3) CSRF是什么？ 如何防御？ 126](#_Toc525304771)

[4) HttpRequest 和HttpResponse 是什么?干嘛用的？ 127](#_Toc525304772)

[5) session 和cookie的区别是什么？为什么session是安全的？ 128](#_Toc525304773)

[6) Django和Tornado的关系？ 129](#_Toc525304774)

[7) Django 中models 利用ORM 对Mysql 进行查表的语句（多个语句）？ 129](#_Toc525304775)

[8) Celery 分布式任务队列？ 131](#_Toc525304776)

[9) django 中间件的使用？ 132](#_Toc525304777)

[10) 简述Django 下的（内建的）缓存机制? 132](#_Toc525304778)

[11) Django使用redis 做缓存 133](#_Toc525304779)

[12) Django MVC 和MVT 133](#_Toc525304780)

[13) Django的继承有哪些？ 134](#_Toc525304781)

[14) Django如何在Model保存前做一定的固定操作，比如写一条日志 134](#_Toc525304782)

[15) 请简要概括下Django测试工具 134](#_Toc525304783)

[16) 解释下django-debug-toolbar的使用 134](#_Toc525304784)

[17) Django中如何连接多个数据库？ 135](#_Toc525304785)

[18) 什么是RESTful API，谈谈你的理解？ 135](#_Toc525304786)

[19) RESTful API设计规范 135](#_Toc525304787)

[20) wsgi和uwsgi是什么？ 136](#_Toc525304788)

[6.2. Flask 136](#_Toc525304789)

[1) 手写一个Flask的HelloWorld 136](#_Toc525304790)

[2) Flask与Django框架有什么区别？ 136](#_Toc525304791)

[3) Flask框架依赖的组件是什么？ 137](#_Toc525304792)

[4) Flask蓝图的作用？ 137](#_Toc525304793)

[5) 列举使用过的Flask第三方组件？ 137](#_Toc525304794)

[6) 简述Flask上下文管理流程? 137](#_Toc525304795)

[7) Flask中g对象的作用？ 137](#_Toc525304796)

[8) Flask 中请求钩子的理解和应用？ 138](#_Toc525304797)

[9) Flask中blinker是什么？ 138](#_Toc525304798)

[10) Flask 项目中如何实现session 信息的写入？ 138](#_Toc525304799)

[11) Flask中的session是什么时候创建，什么时候销毁的？ 138](#_Toc525304800)

[12) flask中一共有几个LocalStack和Local对象 138](#_Toc525304801)

[6.3. Tornado 138](#_Toc525304802)

[1) 简述Tornado框架的特点。 138](#_Toc525304803)

[2) Tornado中静态文件是如何处理的？ 如： <link href="{{static\_url("commons.css")}}" rel="stylesheet" /> 139](#_Toc525304804)

[3) Tornado操作MySQL使用的模块？ 139](#_Toc525304805)

[4) Tornado操作redis使用的模块？ 139](#_Toc525304806)

[5) 简述Tornado框架的适用场景？ 139](#_Toc525304807)

[6.4. Nginx 139](#_Toc525304808)

[1) 什么是Nginx？ 139](#_Toc525304809)

[2) 为什么要用Nginx？ 139](#_Toc525304810)

[3) 为什么Nginx性能这么高？ 140](#_Toc525304811)

[4) 为什么不使用多线程？ 140](#_Toc525304812)

[5) Nginx是如何处理一个请求的呢？ 140](#_Toc525304813)

[6) 正向代理 140](#_Toc525304814)

[7) 反向代理 140](#_Toc525304815)

[8) 动态资源，静态资源分离 141](#_Toc525304816)

[9) 为什么要做动、静分离？ 141](#_Toc525304817)

[10) 负载均衡 141](#_Toc525304818)

[7. Shell编程 141](#_Toc525304819)

[1) Shell脚本里如何检查一个文件是否存在？如果不存在该如何处理？ 141](#_Toc525304820)

[2) Shell脚本里如何检查一个变量是否为空？ 141](#_Toc525304821)

[3) 用shell写一个脚本，对文本中无序的一列数字排序 142](#_Toc525304822)

[4) 请用shell脚本写出查找当前文件夹（/home）下所有的文本文件内容中包含有字符”a”的文件名称 142](#_Toc525304823)

[8. 测试 142](#_Toc525304824)

[1) 禅道和qc 的区别？ 142](#_Toc525304825)

[2) 编写测试计划的目的是 142](#_Toc525304826)

[3) 测试人员在软件开发过程中的任务是什么 142](#_Toc525304827)

[4) 您以往的工作中，一条软件缺陷（或者叫Bug）记录都包含了哪些内容？如何提交高质量的软件缺陷（Bug）记录？ 143](#_Toc525304828)

[5) 简述黑盒测试和白盒测试的优缺点 143](#_Toc525304829)

[6) 简述常用的Bug 管理或者用例管理工具,并且描述其中一个工作流程。 143](#_Toc525304830)

[7) 请列出你所知道的软件测试种类，至少5 项 143](#_Toc525304831)

[8) Alpha 测试与Beta 测试的区别是什么？ 144](#_Toc525304832)

[9) 举例说明什么是Bug？一个bug report 应包含什么关键字？ 144](#_Toc525304833)

[9. 数据结构与算法 144](#_Toc525304834)

[9.1. 理论知识 144](#_Toc525304835)

[1) 算法的特征？ 144](#_Toc525304836)

[2) 基础的数据结构有哪些？ 144](#_Toc525304837)

[3) 基本的算法有哪些，怎么评价一个算法的好坏？ 144](#_Toc525304838)

[4) 深度优先遍历和广度优先遍历的区别？ 144](#_Toc525304839)

[9.2. 字符串 145](#_Toc525304840)

[1) 替换空格 145](#_Toc525304841)

[2) 字符流中第一个不重复的字符 145](#_Toc525304842)

[3) 左旋转字符串 146](#_Toc525304843)

[4) 翻转单词顺序列 146](#_Toc525304844)

[9.3. 数组 147](#_Toc525304845)

[1) 二维数组中的查找 147](#_Toc525304846)

[2) 构建乘积数组 147](#_Toc525304847)

[3) 调整数组顺序使奇数位于偶数前面 148](#_Toc525304848)

[4) 顺时针打印矩阵 148](#_Toc525304849)

[5) 数组中的逆序对 149](#_Toc525304850)

[6) 数字在排序数组中出现的次数 150](#_Toc525304851)

[9.4. 链表 150](#_Toc525304852)

[1) 从尾到头打印链表 150](#_Toc525304853)

[2) 链表中环的入口结点 151](#_Toc525304854)

[3) 删除链表中重复的结点 151](#_Toc525304855)

[4) 链表中倒数第k个结点 152](#_Toc525304856)

[5) 反转链表 152](#_Toc525304857)

[6) 合并两个排序的链表 153](#_Toc525304858)

[7) 复杂链表的复制 153](#_Toc525304859)

[8) 链表成对调换 154](#_Toc525304860)

[9) 两个链表的第一个公共结点 154](#_Toc525304861)

[9.5. 树 155](#_Toc525304862)

[1) 重建二叉树 155](#_Toc525304863)

[2) 二叉树的下一个结点 155](#_Toc525304864)

[3) 对称的二叉树 156](#_Toc525304865)

[4) 按之字形顺序打印二叉树 156](#_Toc525304866)

[5) 把二叉树打印成多行 157](#_Toc525304867)

[6) 序列化二叉树 157](#_Toc525304868)

[7) 二叉搜索树的第k个结点 158](#_Toc525304869)

[8) 树的子结构 158](#_Toc525304870)

[9) 从上往下打印二叉树 159](#_Toc525304871)

[10) 二叉搜索树的后序遍历序列 160](#_Toc525304872)

[11) 求二叉树最大深度 160](#_Toc525304873)

[12) 广度遍历与深度遍历二叉树 160](#_Toc525304874)

[13) 二叉树的前序、中序、后序遍历 161](#_Toc525304875)

[14) 二叉树中和为某一值的路径 162](#_Toc525304876)

[9.6. 栈和队列 162](#_Toc525304877)

[1) 实现一个简单的栈结构 162](#_Toc525304878)

[2) 用两个栈实现队列 163](#_Toc525304879)

[3) 栈的压入、弹出序列 163](#_Toc525304880)

[9.7. 查找和排序 164](#_Toc525304881)

[1) 旋转数组的最小数字 164](#_Toc525304882)

[9.8. 递归和循环 164](#_Toc525304883)

[1) 斐波那契数列 164](#_Toc525304884)

[2) 跳台阶 165](#_Toc525304885)

[3) 变态跳台阶 165](#_Toc525304886)

[4) 矩形覆盖 165](#_Toc525304887)

[9.9. 位运算 165](#_Toc525304888)

[1) 二进制中1的个数 165](#_Toc525304889)

[2) 不用加减乘除做加法 166](#_Toc525304890)

[9.10. 时间效率 166](#_Toc525304891)

[1) 数组中出现次数超过一半的数字 166](#_Toc525304892)

[2) 最小的K个数 166](#_Toc525304893)

[3) 连续子数组的最大和 167](#_Toc525304894)

[4) 把数组排成最小的数 167](#_Toc525304895)

[5) 二分查找 167](#_Toc525304896)

[6) 丑数 168](#_Toc525304897)

[9.11. 其它编程题 168](#_Toc525304898)

[1) 写一个算法判断一个英文单词的所有字母是否全都不同（不区分大小写）。 168](#_Toc525304899)

[2) 使用函数实现一个字符串的全排列 169](#_Toc525304900)

[3) 找零问题 169](#_Toc525304901)

[4) 统计一篇英文文章中单词个数 170](#_Toc525304902)

[5) 给你A,B两个文件，各存放50亿条URL，每条URL占用64字节，内存限制是16G，让你找出A,B文件共同的URL。如果是三个乃至n个文件呢？ 170](#_Toc525304903)

[9.12. 排序算法 171](#_Toc525304904)

[1) 插入排序 171](#_Toc525304905)

[2) 冒泡排序 171](#_Toc525304906)

[3) 快速排序 172](#_Toc525304907)

[4) 归并排序 174](#_Toc525304908)

[5) 堆排序 175](#_Toc525304909)

[10. 项目相关问题 175](#_Toc525304910)

[10.1. 博客系统 175](#_Toc525304911)

[1) 对于这个的架构能够阐述下吗？ 175](#_Toc525304912)

[2) 在项目中担任的角色？实现的模块有哪些？ 176](#_Toc525304913)

[3) 项目的后台管理是怎样实现的？ 176](#_Toc525304914)

[4) 你部署过项目没？怎样部署的？ 176](#_Toc525304915)

[5) 这个平台的缓存你了解吗？是怎样实现的？ 177](#_Toc525304916)

[6) 在这个项目中遇到过什么问题？怎么解决的？ 177](#_Toc525304917)

[10.2. 数据分析平台 177](#_Toc525304918)

[1) 这平台的数据来源能够阐述一下吗？ 177](#_Toc525304919)

[2) django是不支持非关系型数据库的，怎么解决这个问题的？遇到过什么问题？ 177](#_Toc525304920)

[3) 这个平台的主要业务有什么？ 178](#_Toc525304921)

[4) 你主要负责什么模块？ 178](#_Toc525304922)

[5) 在开发过程中遇到过哪些问题？怎么解决的？ 179](#_Toc525304923)

[6) 爬虫每天能够抓取多少的数据？ 179](#_Toc525304924)

[10.3. 电商网站项目 179](#_Toc525304925)

[1) 说说你最近做的这个项目的背景，简单的介绍一下你这个项目? 179](#_Toc525304926)

[2) 项目开发所需人员结构 180](#_Toc525304927)

[3) 你承担这个项目的哪些核心模块? 180](#_Toc525304928)

[4) 负责模块功能实现步骤 181](#_Toc525304929)

[5) 商品管理模块 181](#_Toc525304930)

[6) 你们的系统中有多少SKU？ 有多少SPU？ 181](#_Toc525304931)

[7) 项目中哪些功能模块涉及了大数据量访问?你是如何解决的? 182](#_Toc525304932)

[8) 你做完这个项目后有什么收获? 182](#_Toc525304933)

[9) 如何编写图片上传功能？ 183](#_Toc525304934)

[10) 如何在后台转发请求？ 183](#_Toc525304935)

[11) 你们这个项目中订单 ID 是怎么生成的?我们公司最近打算做一个电商项目，如果让你设计这块，你会考虑哪些问题? 183](#_Toc525304936)

[12) 你们订单系统中用户提交订单的时候验证价格和验证库存是怎么做的？ 183](#_Toc525304937)

[13) 如何解决并发问题的? 183](#_Toc525304938)

[14) 单点登录系统 184](#_Toc525304939)

[15) 你们生产环境的服务器有多少台?(重点以web服务器为主) 184](#_Toc525304940)

[16) 数据备份是怎么做的?有没有做读写分离? 184](#_Toc525304941)

[17) 你们项目中用户从下单到支付成功大概需要多长时间？是否出现过响应过慢的问题？如何解决的？ 184](#_Toc525304942)

[18) 你们使用什么做支付的?如果使用支付宝做支付，请求超时了怎么处理? 184](#_Toc525304943)

[19) 付款成功后支付宝会有数据返回吗?如果付款后支付宝没有返回，或者返回超时了，但是钱又已经扣了，你怎么办? 184](#_Toc525304944)

[20) 你们怎么做退款功能的，要多长时间才能把钱退回给用户? 184](#_Toc525304945)

[21) 不同域名的网站如何实现用户信息共享 184](#_Toc525304946)

[22) 点一个链接访问到一个页面，这个页面上既有静态数据，又有动态数据(需要查数据库的)，打开这个页面的时候就是很慢但是也能打开。怎么解决这个问题，怎么优化?(静态化) 185](#_Toc525304947)

[23) 如果用户一直向购物车添加商品怎么办?并且他添加一次你查询一次数据库?互联网上用户那么多，这样会对数据库造成很大压力。 185](#_Toc525304948)

[24) 你们项目中服务器集群数据不一致问题有什么解决方案 186](#_Toc525304949)

[25) 你们电商网站日活多少？你们电商网站中搜索系统用的es节点有几个？文件服务器有几个？ 186](#_Toc525304950)

[26) 你说你用了 redis 缓存，你 redis 存的是什么格式的数据，是怎么存的? 186](#_Toc525304951)

[27) 解释一下 Django 和 Tornado 的关系、差别 186](#_Toc525304952)

[28) 用户在浏览器中输入一个网址，到 Django 后台捕捉到请求其中的过程？ 186](#_Toc525304953)

[29) 解释下 Django 的 MVC 模式，其中那一部分充当的是 controller 的部分？ 187](#_Toc525304954)

[30) 你在上家公司做开发的时候bug率是多少？ 187](#_Toc525304955)

[31) 项目中分布式事务怎么做的？ 187](#_Toc525304956)

[32) 其它问题 187](#_Toc525304957)

[11. 企业面试题 187](#_Toc525304958)

[11.1. 阿凡题 187](#_Toc525304959)

[1) 对于aaa=[1,2,3,4,5,6,7,8,9,0]分别写出以下操作对应的输出： 187](#_Toc525304960)

[2) 请写出函数countchr，统计出字符串中每个字符的个数，输出为字典格式。如有字符串aaa=“我们不是阿凡提，我们是阿凡题”，输出格式为：{“我”:2, “们”:2, “不”:1……(略)} 187](#_Toc525304961)

[3) 对于字典aaa={“a”:10, “b”:34, “c”:11, “d”:22}，写出程序对调key和value，产生新的字典 188](#_Toc525304962)

[4) 通过正则匹配下列字符串aaa中的url地址，输入：aaa='"http://www.baidu.com"><a href=“http://www.afanti100.com” target=“\_blank”>阿凡题</a>' 188](#_Toc525304963)

[5) 请说明HTTP请求中的GET和POST请求的区别 188](#_Toc525304964)

[6) 给定一个字符串作为输入，将其中的数字用数字前近邻的字母序列进行替换，字母序列重复的次数等于该数字，最后将字符串输出： 输入：”af1an2ti3” 输出：”afanantititi” 189](#_Toc525304965)

[7) 根据描述，写出接口调用的请求代码： 189](#_Toc525304966)

[11.2. 得瑞领新 190](#_Toc525304967)

[1) 列举出Python自带的主要数据类型并举例说明 190](#_Toc525304968)

[2) Python里，正则表达式search()与match()的区别 191](#_Toc525304969)

[3) Python里，copy，deepcopy和引用的区别 191](#_Toc525304970)

[4) 简要描述Python里的进程，线程，通信方式及各自优缺点 191](#_Toc525304971)

[5) GIL锁的作用是什么，有何局限性？ 192](#_Toc525304972)

[6) 解释生成器（generator）与函数的不同 192](#_Toc525304973)

[7) 请写一个Python类，要求包含public/private方法及变量的声明，同时定义一个静态方法 192](#_Toc525304974)

[8) 编写程序，将数组中的重复值去除（仅保留第一个），要求保留原数组顺序，比如[3,2,1,4,2,6,1]，输出[3,2,1,4,6] 192](#_Toc525304975)

[9) 编写程序，输入一个数值的列表，如[1,8,3,4,9]，对每个数逐一进行平方加一计算后求列表的平均值，输出为该平均值，要求使用map/reduce方式实现 193](#_Toc525304976)

[10) 有一个数据库表为student，字段有name（姓名），age（年龄），sex（性别），请按照以下要求编写SQL语句： 193](#_Toc525304977)

[11.3. 精图天下外派向前金服 193](#_Toc525304978)

[1) Python是如何进行内存管理的 193](#_Toc525304979)

[2) 字符串格式化 % 和 format的区别 194](#_Toc525304980)

[3) 写出以下两个print的值 194](#_Toc525304981)

[4) 以下的代码的输出将是什么？写出你的答案： 194](#_Toc525304982)

[5) Python中的lambda是什么？ 195](#_Toc525304983)

[6) 如何查询和替换一个文本中的字符串 195](#_Toc525304984)

[7) 列出几种常见的关系型数据库和非关系型数据库？ 195](#_Toc525304985)

[8) 常见的反爬虫机制有哪些？应对措施请举例 195](#_Toc525304986)

[9) <div class=“house-chat-phone”><p><span>电话</span></p></div>提取这段代码中的手机号，采用2中以上的方式 196](#_Toc525304987)

[11.4. 乐飞天下 196](#_Toc525304988)

[1) 如何判断一个Python对象的类型？ 196](#_Toc525304989)

[2) Python里面如何生成随机数？ 196](#_Toc525304990)

[3) 请写出匹配IP的Python正则表达式 197](#_Toc525304991)

[4) 写尽可能多的str对象的方法 197](#_Toc525304992)

[5) 全局变量与局部变量的区别，如何在function里面给一个全局变量赋值？ 198](#_Toc525304993)

[6) tuple和list的区别，有两个list，b1=[1,2,3]，b2=[2,3,4]，写出合并代码 198](#_Toc525304994)

[7) 请写出一段Python代码实现删除一个list里面的重复元素，l = [1,1,2,3,4,5,4] 198](#_Toc525304995)

[8) 写出如下list的交集与差集代码，b1=[1,2,3]，b2=[2,3,4] 199](#_Toc525304996)

[9) 请写出一段Python代码实现list里排序a=[1,2,4,2,4,5,7,10,5,5,7,8,9,0,3] 199](#_Toc525304997)

[10) D=[x\*x for x in range(1,10)], 请写出D的输出结果 200](#_Toc525304998)

[11) 说说用过的几种爬虫框架及他们的优缺点，用过哪些解析html 的库 200](#_Toc525304999)

[12) 谈一下对于对线程的理解，对于cpu 密集性IO 密集性怎样使用线程，说说线程池，线程锁的用法，有么有用过multiprocessing 或者concurrrent.futures? 200](#_Toc525305000)

[13) 解析HTML： <a href=”http://www.51smart.com” class=”lefei”>lftx123</a>，将”http://www.51smart.com” 和 “lftx123” 全部提取出来（使用库和正则均可） 200](#_Toc525305001)

[14) 常见的HTTP状态： 404、302、301、500、403分别是什么意思？ 201](#_Toc525305002)

[15) @classmethod、@staticmethod有什么区别？什么情况下用@classmethod，什么情况下用@staticmethod？ 201](#_Toc525305003)

[11.5. 英赛克 201](#_Toc525305004)

[1) 请介绍一下 Python是如何进行内存管理的? 201](#_Toc525305005)

[2) 请写一段 Python代码,说明自定义装饰器的使用。 202](#_Toc525305006)

[3) 请介绍一下你常用的 Python Web程序部署方式? 202](#_Toc525305007)

[4) Django的Model有哪几种继承形式,有何不同? 202](#_Toc525305008)

[5) 如何提高页面的响应速度 203](#_Toc525305009)

[6) Django中如何在 Model保存前做一些固定的操作,比如写日志? 203](#_Toc525305010)

[7) 请从你开发过的 Python Web项目中,选一个进行介绍,用框图的形式画出软件架构和主要功能模块即可,面试阶段进行详细说明。 203](#_Toc525305011)

[11.6. 掌阅科技 204](#_Toc525305012)

[1) 描述对super、pass、yield、lambda关键字修饰的理解 204](#_Toc525305013)

[2) 请大致描述一下Python的GIL机制，以及Python中多线程和多进程的区别 204](#_Toc525305014)

[3) Python是如何进行内存管理的，以及大致描述一下Python的GC机制 205](#_Toc525305015)

[4) 请分别描述一下类装饰器和函数修饰器的实现过程及应用场景 206](#_Toc525305016)

[5) 请用两个队列来实现一个栈（给出伪代码即可） 206](#_Toc525305017)

[6) 实现一个Singleton单例类，要求遵循基本语言编程规范 206](#_Toc525305018)

[7) 请实现一个简单的socket编程，要求： 207](#_Toc525305019)

[8) 请为掌阅设计一个并发处理KEY-VALUE引擎，要求每条请求的数据小于16k，数据总量为1T，QPS为5000/s，要求： 207](#_Toc525305020)

[11.7. 云趣网络 207](#_Toc525305021)

[1) 请简述你对Web开发中MVC模型的理解 207](#_Toc525305022)

[2) 请举出你所知道的Python Web开发框架 208](#_Toc525305023)

[3) 请写出一段Python遍历某个目录文件的代码 208](#_Toc525305024)

[4) \_xxx、\_\_xxx、\_\_xxx\_\_三种形式的变量的区别是什么 208](#_Toc525305025)

[5) Python的列表生成式有什么优点？ 209](#_Toc525305026)

[6) yield是什么？ 209](#_Toc525305027)

[7) Python的数据类型有哪些？ 209](#_Toc525305028)

[8) 写出一段代码打开一个名为user.log的文件，并写入hello world 209](#_Toc525305029)

[9) Python生成多进程/线程的方式有哪些？（可通过代码举例） 209](#_Toc525305030)

[10) Web开发中的跨域指的是什么？ 210](#_Toc525305031)

[11) SQL语句中REPLACE into user(uid, username) VALUES(“0”, “maze”)代表什么？ 210](#_Toc525305032)

[12) 如何杀死Linux中含有example字符的进程 210](#_Toc525305033)

[13) 请说出RSA加密与RSA签名的区别 210](#_Toc525305034)

[14) 说说Python装饰器的作用及其适用场景 210](#_Toc525305035)

[11.8. 什么值得买 211](#_Toc525305036)

[1) 描述一下比较熟悉的爬虫框架 211](#_Toc525305037)

[2) 定义一个函数，反转一个链表（输入：链表，输出：链表） 211](#_Toc525305038)

[3) 请描述下TCP/IP协议、HTTP协议、HTTPS协议、DNS服务，以及彼此之间的关系 211](#_Toc525305039)

[4) 请写出4种排序算法名称，以及对应的时间复杂度 212](#_Toc525305040)

[5) 请描述下cut、awk、tail、grep的作用和简单语句示例 212](#_Toc525305041)

[6) 请写出http协议头中各参数的名称，以及对应的作用 213](#_Toc525305042)

[7) 请写出下面代码的输出值 213](#_Toc525305043)

[8) 灯亮与灯灭：100 个灯泡，起始全是灭的。执行 100 次步骤：第i次步骤，对i，2\*i，3\*i 。。的灯全部切换一下开关（灭的开等，亮的关灯）最后有多少灯是亮的? 213](#_Toc525305044)

[11.9. 数据堂 213](#_Toc525305045)

[1) 当变量x为何值的时候，以下三个表达式在内存层面的操作不同？有什么不同？ 213](#_Toc525305046)

[2) 假设变量d是一个字典，要判断元素e是否在该字典中，分别说明以下两个表达式的时间复杂度 214](#_Toc525305047)

[3) 请编写函数delete\_by\_index(l, index\_list)，使得下面的脚本能正确执行。（其中l是一个非空列表，index\_list是l中要删除的元素的索引组成的有序列表，要求原地删除） 214](#_Toc525305048)

[4) 请编写函数group(), 使得下面的脚本可以正确运行。（其中l是一个字符串组成的列表，返回一个列表的列表，使得每个子列表里的字符串包含相同的首字符） 215](#_Toc525305049)

[11.10. 微讯联创 215](#_Toc525305050)

[1) Python中pass语句的作用是什么？ 215](#_Toc525305051)

[2) dict的items()方法与iteritems()方法的不同？ 215](#_Toc525305052)

[3) 写出尽可能多的str对象的方法，并解释其作用 215](#_Toc525305053)

[4) 说明os,sys模块的不同，并列举常用的模块方法？ 216](#_Toc525305054)

[5) 下面代码的输出是什么？请解释你的答案 217](#_Toc525305055)

[6) 有两个升序排列的listA和listB，尽可能高效的方法将这两个list合并为一个新的升序排列的listC 218](#_Toc525305056)

[7) 现代x86 CPU的内页尺寸是？ 218](#_Toc525305057)

[8) RISC是什么意思？ 218](#_Toc525305058)

[9) 变量name为用户输入的用户名，下列代码有什么问题？ 219](#_Toc525305059)

[11.11. 企名片 219](#_Toc525305060)

[1) 简述一次完整的http通信过程，常用请求方式，常见响应状态码，http的无状态性，cookies这些概念是什么？ 219](#_Toc525305061)

[2) 说说进程、线程和锁的关系 219](#_Toc525305062)

[3) MySQL操作： 219](#_Toc525305063)

[4) a=[1,2,”hello”,[1,2]] 220](#_Toc525305064)

[5) \*args and \*\*kwargs的区别 220](#_Toc525305065)

[6) 写一个匹配email地址的正则表达式 220](#_Toc525305066)

[7) 常见反爬虫措施有哪些？一般怎么克服？ 221](#_Toc525305067)

[8) 编写爬虫的常用模块或者框架有哪些?请说明一个爬虫的行为步骤 221](#_Toc525305068)

[9) 排序算法有哪些用Python 写一种排序算法 222](#_Toc525305069)

[11.12. 凯奇谷 222](#_Toc525305070)

[1) 请写出三种不依赖中间变量的方法交换两个变量a和b的值 222](#_Toc525305071)

[2) 写出下列程序的打印结果： 222](#_Toc525305072)

[3) 请给出尽可能多的方案，将一个字符串反转（如”hello world” -> “dlrow olleh”） 223](#_Toc525305073)

[4) 在一个长度为n的列表中搜索一个元素，平均需要进行\_ N \_次比较。 223](#_Toc525305074)

[5) 将一个数字列表按\*绝对值\*大小排序（如[3,1,-4,-2] -> [1,-2,3,-4]） 223](#_Toc525305075)

[6) 有函数定义如下： 224](#_Toc525305076)

[7) 构造一个栈类（后进先出），可以在初始化时指定大小，并实现push、pop、is\_empty、is\_full四种方法，使用示例如下： 224](#_Toc525305077)

[8) 写一个抽卡函数，按照配置的权重分布随机抽取一张卡牌。 225](#_Toc525305078)

[9) 现有Tornado日志文件log，每行数据示例如下： 225](#_Toc525305079)

[10) 动态语言的一大优势是可以在代码运行时动态修改其方法，实现hot patch，请写出可以在下列注释处执行的一段代码，使得inst.add方法改为返回x+y+1，实现如下效果： 226](#_Toc525305080)

[11) 请选出下面你了解的Python关键词 227](#_Toc525305081)

[11.13. 猎豹移动 227](#_Toc525305082)

[1) 请尽可能列举Python列表的成员方法，并给出以下列表操作的答案： 227](#_Toc525305083)

[2) 用Python实现统计一篇英文文章内每个单词的出现频率，并返回频率最高的前10个单词及其出现次数，并解答以下问题（标点符号可省略）： 227](#_Toc525305084)

[3) 简述Python GIL概念，以及它对Python多线程的影响？编写一个多线程抓取网页的程序，并阐述用多线程抓取程序是否比单线程有提升，并解释原因 228](#_Toc525305085)

[4) 用Python编写一个线程安全的单例模式实现 228](#_Toc525305086)

[5) 请回答以下问题 229](#_Toc525305087)

[6) Python里面如何拷贝一个对象？（赋值，浅拷贝，深拷贝的区别） 229](#_Toc525305088)

[7) 介绍一下except的用法和作用？ 229](#_Toc525305089)

[8) Python中pass语句的作用是什么？介绍一下Python下range()函数的用法？ 230](#_Toc525305090)

[9) 如何用Python来进行查询和替换一个文本字符串？ 230](#_Toc525305091)

[10) Python里面match()和search()的区别？ 230](#_Toc525305092)

[11) 用Python匹配HTML tag的时候，<.\*>和<.\*?>有什么区别？ 231](#_Toc525305093)

[12) 有没有一个工具可以帮助查找Python的bug和进行静态的代码分析？ 231](#_Toc525305094)

[13) 单引号，双引号，三引号的区别 231](#_Toc525305095)

[11.14. 东方金信 231](#_Toc525305096)

[1) 下列哪个语句在Python中是非法的？ 231](#_Toc525305097)

[2) 关于Python内存管理，下列说法错误的是： 231](#_Toc525305098)

[3) 下面哪个不是Python合法的标识符： 231](#_Toc525305099)

[4) 下列哪种说法是错误的 232](#_Toc525305100)

[5) 下列表达式在py2.x中的值为True的是 232](#_Toc525305101)

[6) Python不支持的数据类型有 232](#_Toc525305102)

[7) 关于Python中的复数，下列说法错误的是： 232](#_Toc525305103)

[8) 关于字符串下列说法错误的是 232](#_Toc525305104)

[9) 以下不能创建一个字典的语句是 232](#_Toc525305105)

[10) 下列Python语句正确的是 232](#_Toc525305106)

[11.15. 中创信测 232](#_Toc525305107)

[1) 有如下的一段代码： 232](#_Toc525305108)

[2) 问题： 为了让下面这段代码运行，需要增加哪些代码？ 233](#_Toc525305109)

[3) 下面这段代码输出什么？ 234](#_Toc525305110)

[4) 下面这段代码输出什么？ 234](#_Toc525305111)

[5) 下面这段代码输出什么？ 235](#_Toc525305112)

[6) 如何快速交换下面两个变量值 235](#_Toc525305113)

[7) 如下的代码： 235](#_Toc525305114)

[8) 一个包里有三个模块，mod1.py , mod2.py , mod3.py , 但使用form demopack import \* 导入模块时，如何保证只有mod1, mod3被导入了？ 236](#_Toc525305115)

[9) 如何限制一个class声明的属性？ 236](#_Toc525305116)

[10) 写一个闭包函数，接收整数参数n，返回一个函数，函数的功能是把函数的参数和n相乘并把结果返回。 236](#_Toc525305117)

[11) 解释下面的代码慢在哪？ 237](#_Toc525305118)

[11.16. 文思海辉外派中国移动 237](#_Toc525305119)

[1) 软连接和硬连接的区别？ 237](#_Toc525305120)

[2) 深拷贝和浅拷贝的区别？ 237](#_Toc525305121)

[3) Redis的应用场景？ 237](#_Toc525305122)

[4) 如何处理高并发？ 238](#_Toc525305123)

[5) 对堆和栈了解吗？ 238](#_Toc525305124)

[6) 用Linux命令替换文件中字符串？ 239](#_Toc525305125)

[7) 用Linux命令动态查看Log日志 239](#_Toc525305126)

[8) Nginx崩溃了怎么办？ 了解一下Keepalived架构 239](#_Toc525305127)

[11.17. 深圳库尚 239](#_Toc525305128)

[1) 简单描述\*args and \*\*kwagrs区别和应用场景 239](#_Toc525305129)

[2) 编写代码获取昨天的日期， 格式为：2018-01-03 239](#_Toc525305130)

[3) 请写出以下程序的运行结果： 240](#_Toc525305131)

[4) 用Python抓取一个网页（例如：www.qq.com），用正则匹配出页面所有的链接 240](#_Toc525305132)

[5) MySQL有哪几种存储引擎，分别有什么区别？ 240](#_Toc525305133)

[6) MySQL怎么支持emoji表情？ 241](#_Toc525305134)

[7) 有如下表结构： 241](#_Toc525305135)

[8) 简述Redis如何实现分布式锁 242](#_Toc525305136)

[9) Linux查看系统端口占用情况用什么命令？ 242](#_Toc525305137)

[10) 写一个Linux命令查找/data目录下中的所有Python文件中包含eval字符串的行 242](#_Toc525305138)

[11.18. 博派通达 242](#_Toc525305139)

[1) 请列举你所知道的Python代码检测工具及它们之间的区别 243](#_Toc525305140)

[2) 请简述你对单元测试的理解并列举Python单元测试相关工具的库 243](#_Toc525305141)

[3) 请给出下面代码片段的输出并阐述涉及的Python相关机制 243](#_Toc525305142)

[4) 请给出下面代码片段的输出 243](#_Toc525305143)

[5) func()参考第四题 244](#_Toc525305144)

[6) 请简述标准库中functools.wraps的作用 244](#_Toc525305145)

[7) 请给出下面代码片段的输出 244](#_Toc525305146)

[8) 请给出下面代码片段的输出 245](#_Toc525305147)

[9) 第八题的单例实现需要改进的地方 245](#_Toc525305148)

[10) 简述Python 垃圾回收机制和如何解决循环引用 246](#_Toc525305149)

[11) 请列举出常见的MySQL 存储引擎 247](#_Toc525305150)

[12) InnoDB 有哪些特性? 247](#_Toc525305151)

[11.19. 上海理想信息 248](#_Toc525305152)

[1) 在Python中，list、tuple、dict、set有什么区别，主要应用在什么样的场景？ 248](#_Toc525305153)

[2) 简要介绍下python里面的装饰器，列举几个常见的内置装饰器。 248](#_Toc525305154)

[3) 静态方法、类方法、实例方法的区别 248](#_Toc525305155)

[4) python的常见的内建函数有哪些 249](#_Toc525305156)

[5) \_\_call\_\_、\_\_init\_\_、\_\_new\_\_都是在什么时候触发？ 249](#_Toc525305157)

[6) 下面代码的输出是什么？ 249](#_Toc525305158)

[7) 生成100内的偶数列表 250](#_Toc525305159)

[8) 将列表内元素，根据位数合并成字典 250](#_Toc525305160)

[9) 合并两个有序列表 250](#_Toc525305161)

[10) 请用尽量简洁的方法将二维数组转换成一维数组 251](#_Toc525305162)

[11) lst = [{“a”:3}, {“b”:1}, {“c”:5}] 251](#_Toc525305163)

[12) 将列表按下列规则排序，补全代码 251](#_Toc525305164)

[13) 下面的代码输出的是什么，为什么会是这个结果？ 251](#_Toc525305165)

[14) 请阅读下面的代码： 252](#_Toc525305166)

[15) 下面代码的输出是什么，为什么？如何修改可以让代码有期望的输出？ 252](#_Toc525305167)

[16) HTTP请求中的GET和POST有什么区别？Cookies和Sessions有什么区别？ 252](#_Toc525305168)

[17) Python常见的任务队列有哪些？ 253](#_Toc525305169)

[18) Python的垃圾回收机制是什么 254](#_Toc525305170)

[19) 简要介绍Django的验证机制，如扩展auth\_user表，添加额外的字段？ 254](#_Toc525305171)

[11.20. Zingfront智线 5 254](#_Toc525305172)

[1) 我们定义 个字符 的超集为: 若字符 A包含字符 B中的所有字符，且字符 A中某 字符 a 的数 于a在字符 B中的数 ，那么A为B的”超集”。 254](#_Toc525305173)

[2) 有一个MxN 的棋盘,每个格子上有一个字母,现在按照下面的规则来构造单词：可以从任意一个位置开始,接着向该位置相邻的8 个位置中选取下一个(不能选取已经选取过的位置)，如此循环，构成一个字母序列，生成一个单词。 254](#_Toc525305174)

[3) 王者农药新模式——“智慧王者”， 提供5个英雄， 假设各自血量和攻击力如下： 255](#_Toc525305175)

[4) 给定.一组数字，.一组有9个数字，将这9个数字填写到3\*3的九宫格内；使得横，竖，斜对.角.一条线上的三个数字之和相等；如果.无解则打印.无解； 256](#_Toc525305176)

[5) 给定形如下⾯面的矩阵， 256](#_Toc525305177)

[6) 设计⼀一个股票模拟交易易系统。假设我们有⼀一个很⽜牛叉的AI系统，已经预测到未来⼀一段时间内给定股票的价格，以数组来表示，它的第i个元素是⼀一⽀支给定的股票在第i天的价格。 257](#_Toc525305178)

[11.21. 全品教育 257](#_Toc525305179)

[1) Tuple 和list 区别？ 257](#_Toc525305180)

[2) 这两个参数\*args \*\*kwargs 是什么意思? 257](#_Toc525305181)

[3) Python 里面如何实现tuple 和list 的转换? 257](#_Toc525305182)

[4) Python 里面range 和xrange 的区别? 258](#_Toc525305183)

[5) Python 里面classmethod 和staticmethod 的区别? 258](#_Toc525305184)

[6) 如何反向输出序列比如[2,6,5,3],输出为[3,5,6,2] 258](#_Toc525305185)

[7) Python 里面实现删除重复的元素? 258](#_Toc525305186)

[8) Python 里面copy 和deepcopy 的区别? 259](#_Toc525305187)

[9) Python 里面的search 和match 的区别? 259](#_Toc525305188)

[10) Python 代码如何得到列表的交集和差集？ 259](#_Toc525305189)

[11) 请写出一段代码求出1 到100 的和？ 260](#_Toc525305190)

[12) Python 中正则表达式提取出字符串中的数字 260](#_Toc525305191)

[11.22. 首信科技 260](#_Toc525305192)

[1) Python 中list、tuple、dict、set 有什么区别，主要应用在什么样的场景？ 260](#_Toc525305193)

[2) Python 中静态函数、类函数、成员函数的区别？并写一个示例？ 260](#_Toc525305194)

[3) 用Python 语言写一个函数，输入一个字符串，返回倒序结果？ 261](#_Toc525305195)

[4) 介绍一下Python 的异常处理机制和自己开发过程中的体会? 261](#_Toc525305196)

[5) jQuery 库中$()是什么？网上有5 个<div>元素，如何使用jQuery 来选择它们？ 262](#_Toc525305197)

[6) Django 中使用memcached 作为缓存的具体方法？有缺点说明？ 262](#_Toc525305198)

[11.23. 号外科技 262](#_Toc525305199)

[1) 单引号、双引号、三引号的区别？ 262](#_Toc525305200)

[2) 如何在一个function 里面设置一个全局变量？ 262](#_Toc525305201)

[3) 描述yield 使用场景? 262](#_Toc525305202)

[4) 生成1~10 之间的整数？ 263](#_Toc525305203)

[5) Python 如何生成缩略图？ 263](#_Toc525305204)

[6) 列出比较熟悉的爬虫框架，并简要说明？ 263](#_Toc525305205)

[7) 列举常见的反爬技术，并给出应对方案？ 263](#_Toc525305206)

[8) 网络协议http 和https 区别？ 264](#_Toc525305207)

[9) 什么是cookie，session 有什么区别？ 264](#_Toc525305208)

[10) MySQL中MyISAM与InnoDB的区别？ 265](#_Toc525305209)

[11.24. 妙计旅行 265](#_Toc525305210)

[1) Python 主要的内置数据类型有哪些？ 265](#_Toc525305211)

[2) List能否完全取代tuple？ 265](#_Toc525305212)

[3) print(dir('a'))输出的是什么？ 266](#_Toc525305213)

[4) 给定两个list，A 和B，找出相同元素和不同元素？ 266](#_Toc525305214)

[5) 请反转字符串？ 266](#_Toc525305215)

[6) 交换变量A, B 的值？ 267](#_Toc525305216)

[7) 用select 语句输出每个城市中心距离市中心大于20km 酒店数？ 267](#_Toc525305217)

[8) 给定一个有序列表，请输出要插入值k 所在的索引位置？ 267](#_Toc525305218)

[9) 正则表达式贪婪与非贪婪模式的区别？ 267](#_Toc525305219)

[10) 写出开头匹配字母和下划线，末尾是数字的正则表达式？ 267](#_Toc525305220)

[11) 请说明HTTP 状态码的用途，请说明常见的状态码机器意义？ 267](#_Toc525305221)

[12) 当输入http://www.baidu.com 时，返回页面的过程中发生了什么？ 268](#_Toc525305222)

[13) 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。 268](#_Toc525305223)

[14) 关系型数据库中，表和表之间有左连接，内连接，外连接，分别解释下他们的含义和区别？ 268](#_Toc525305224)

[15) 如何定时启动你的爬虫项目？ 268](#_Toc525305225)

[16) 什么是Scrapy-Redis 中的指纹,是如何去重的？ 269](#_Toc525305226)

[17) 代码优化从哪些方面考虑？有什么想法？ 269](#_Toc525305227)

[18) 写出以下程序的输出： 269](#_Toc525305228)

[19) 用一个SQL查询country属于‘CN’的所有酒店，并输出HotelId、Name、City；MySQL有哪几个常见引擎，有什么差异 270](#_Toc525305229)

[20) 编程题1：有一个3G大小的文件，文件每行一个String，内容为酒店的id和一个图片的名字，使用‘\t’分割 272](#_Toc525305230)

[21) 编程题2：Given a string containing juest the characters ‘(‘, ‘)’, ‘{‘, ‘}’ , ‘[‘ and ‘]’, determine if the input string is valid , The brackets must close in the correct order, ‘()’ and ‘()[][()]’ are all valid but ‘(]’ and ‘([)]’ are not. 273](#_Toc525305231)

[22) re.findall(r’a.\*b’, ‘aabab’)写出该正则的输出并说明为什么 273](#_Toc525305232)

[23) 请用xpath找出html页面内id=’test\_xxx’且class=’demo’节点下的所有图片地址 273](#_Toc525305233)

[24) 指出下面程序存在的问题 274](#_Toc525305234)

[25) Django 项目的优化（web 通用） 274](#_Toc525305235)

[11.25. 360 274](#_Toc525305236)

[1) 请拿出B表中的accd，(A表中和B表中的一样的数据) 274](#_Toc525305237)

[2) a = “abbbccc”，用正则匹配为abccc,不管有多少b，就出现一次？ 274](#_Toc525305238)

[3) Xpath使用的什么库？ 274](#_Toc525305239)

[4) py2 和py3 的区别？ 275](#_Toc525305240)

[5) Redis 里面list 内容的长度？ 275](#_Toc525305241)

[6) 多线程交互，访问数据，如果访问到了就不访问了，怎么避免重读？ 275](#_Toc525305242)

[7) MySQL怎么限制IP 访问？ 275](#_Toc525305243)

[8) 带参数的装饰器? 275](#_Toc525305244)

[11.26. 成安科技 276](#_Toc525305245)

[1) Python 中使用%与.format 格式化文本 276](#_Toc525305246)

[2) Python 的logging 模块常用的几个等级？ 277](#_Toc525305247)

[3) 在HTTP1.1 中常见的状态码有哪些，如何设置状态码？ 277](#_Toc525305248)

[4) 用Python 写一段快速排序算法。 277](#_Toc525305249)

[11.27. 上海金台灯 277](#_Toc525305250)

[1) 什么是lambda 函数，它有什么好处？ 277](#_Toc525305251)

[2) 什么是Python 的list and dict comprehensions（列表推导式和字典推导式） 278](#_Toc525305252)

[3) Python 里面如何实现tuple 和list 的转换？ 278](#_Toc525305253)

[4) Python 里面如何拷贝一个对象？ 278](#_Toc525305254)

[5) 写一段except 的函数 278](#_Toc525305255)

[6) Python 里面pass 语句的作用是什么？ 279](#_Toc525305256)

[7) Python 中range()函数的用法 279](#_Toc525305257)

[8) Python re 模块匹配HTML tag 的的时候，<.\*>和<.?>有什么区别？ 279](#_Toc525305258)

[9) Python 程序中中文乱码如何解决？ 279](#_Toc525305259)

[10) Python 的传参是传值还是传址？ 279](#_Toc525305260)

[11) with 语句的作用,写一段代码？ 279](#_Toc525305261)

[11.28. 芸品绿 280](#_Toc525305262)

[1) Python常用的数据结构的类型及其特性？ 280](#_Toc525305263)

[2) 已知： A = {1,2,3} B = {1,2,4} 280](#_Toc525305264)

[3) 已知： AList = [1,2,3] BSet = {1,2,3} 280](#_Toc525305265)

[4) ADist = {“a”:0，”b”:1, “c”:2}判断”h” 是否是ADist的key？（写出具体的实现过程） 281](#_Toc525305266)

[5) Python的进程、线程、协程有什么区别？ 281](#_Toc525305267)

[6) python2中字符串编码有几种，与Python3有什么区别？字节和字符编码分别是什么意思？ 282](#_Toc525305268)

[7) 爬虫过程中碰到js加载后的数据，除了常规的模拟浏览器比如phantomjs之外，还有什么好的应对方法？ 282](#_Toc525305269)

[11.29. 慧贸天下 282](#_Toc525305270)

[1) 请写出一段Python代码，实现删除一个list里面的重复元素，方法不限，可以写多种实现思路。 282](#_Toc525305271)

[2) 给x,y分别赋值为2,3，如何取出'f2'的值，值是多少？ 283](#_Toc525305272)

[3) 请写出如下代码相应的输出结果： 283](#_Toc525305273)

[4) 补充缺失的代码： 284](#_Toc525305274)

[5) Python字典按照key如何排序，请写出你掌握的排序代码或思路（方法可以多种） 284](#_Toc525305275)

[6) 请写出下面代码的输出结果是什么，并做响应的解释说明。 284](#_Toc525305276)

[11.30. 微软小冰 8 285](#_Toc525305277)

[1) 完成一个函数：def Add(num1, num2):其中，两个输入都是数字，都是字符串（如：“12345678986543210123456789”），要求计算两个数字的和，返回一个字符串，不能使用内置函数，如int，long等。例如，输入两个数字是：“1000000000000000”和“-1”，返回“999999999999999”。 285](#_Toc525305278)

[2) 给定一个数组nums，然后对其排序，使得排序结果满足nums[0] < nums[1] > nums[2] < nums[3]…。 例如给定数组nums=[1,2,3,4,5,6,7,8,9],其中一个满足条件的结果是1<6>2<7>3<8>4<9>5.给出一个结果即可（可能无解）。最优解法是O(n)时间复杂度和O(1)空间复杂度。 285](#_Toc525305279)

[3) 写一个函数，输入是两个int数组A和B。要求从A和B中分别取出一个数，使他们的和为20。打印出所有的组合。要求数字在数组中的位置和数字本身。比如输入为 A = [18, 2, 7, 8, 3], B = [17, 1, 19]，输出为 3 (A4) + 17 (B0) = 20，表示A的第4个元素是3，B的第0个元素是17 285](#_Toc525305280)

[4) 写一个函数，输入一个随机的01序列，打印出这个序列中最长的01交替出现的序列的起始位置和结束位置。例如：输入“0001010101101”，输出起始位置2, 结束位置10 285](#_Toc525305281)

[5) 逆转一个单链表。 285](#_Toc525305282)

[6) 实现一个类，用于表示一个大小无限制的整数，要求可以用任何进制的形式初始化，并提供方法打印成任何进制的形式，并实现加减法（或四则运算），实现repr 285](#_Toc525305283)

[7) 给一个数n，给出可以由多少种方法用正整数来累加成该数。例如给一个数3，那么一共有三种：1+1+1，1+2，3 285](#_Toc525305284)

[8) 将两个有序整数序列合并为一个大的有序整数序列，输出，然后反转，再输出。自己实现，不能调用sort方法和reverse方法 285](#_Toc525305285)

[11.31. 超盟数据 285](#_Toc525305286)

[1) 请写出以下代码的运行结果 285](#_Toc525305287)

[2) 请写出以下代码运行结果 286](#_Toc525305288)

[3) 请实现一个def func\_1(n)函数，要求： 1. 利用递归方法， 2.输出n! 的计算结果 286](#_Toc525305289)

[4) 请实现一个func\_timeout的装饰器，当函数调用时间超过50ms时阻断函数继续执行并抛出异常 286](#_Toc525305290)

[5) 请简单描述以下Memcache与Redis的区别都有哪些 287](#_Toc525305291)

[6) 请简单描述一下SQL连接查询的几种方式 287](#_Toc525305292)

[7) （加分）请实现一个简单的缓存结构，支持get与set操作，并分别列出两操作的时间复杂度 288](#_Toc525305293)

[8) （加分）下图是一组连续高度不一宽度为1的墙壁，可使用数组 [ 1, 3, 4, 3, 28, 10, 9, 19, 22, 3, 34, 55, 43, 21 ] 表示，当下雨时雨水会填满墙之间的空间，请使用python实现求出下图被水填满区域面积的相关逻辑 289](#_Toc525305294)

[9) （加分）10x10的二维矩形世界中，每个方格居住着一个活着的（可用x表示）或死亡的（可用 - 表示）细胞。死亡的细胞相邻方格中活细胞数量为3个时细胞将会复活，当相邻方格活着的细胞数量超过3个，这个细胞会因为资源匮乏而在下一代死去；相反，如果周围活细胞少于2个，这个细胞会因太孤单而死去，请用代码实现并打印出第一代至第十代细胞的存活分布情况。 289](#_Toc525305295)

[11.32. 信达悦和 289](#_Toc525305296)

[1) 给出以下列表操作的答案： a = [1, 2, 3, 4, 5] , a[::2] = ? , a[-2:] = ? 289](#_Toc525305297)

[2) 单引号，双引号，三引号的区别，分别阐述三种引号用的场景和区别 289](#_Toc525305298)

[3) 什么是lambda函数，有什么好处？ 290](#_Toc525305299)

[4) 补充缺失的代码 290](#_Toc525305300)

[5) 阅读下面的代码，写出A0，A1至An的最终值 290](#_Toc525305301)

[6) Python和多线程（mutil-threading）这是个好主意吗？列举一些让Python代码以并行方式运行的方法 291](#_Toc525305302)

[7) 下面代码会输出什么？ 291](#_Toc525305303)

[8) 这两个参数是什么意思： \*args， \*\*kwargs？ 我们为什么要使用它们？ 291](#_Toc525305304)

[9) 下面这些是什么意思： @classmethod，@staticmethod，@property？ 292](#_Toc525305305)

[10) 简要描述Python的垃圾回收机制（garbage collection） 292](#_Toc525305306)

[11) Python如何实现单例模式，保证线程安全 292](#_Toc525305307)

[12) 将下面的函数按照执行效率高低排序，它们都接受由0至1之间的数字构成的列表作为输入，这个列表可以很长。一个输入列表的示例如下： [ random.random() for i in range(100000)] ，你如何证明自己的答案是正确的。 293](#_Toc525305308)

[13) Python里面如何拷贝一个对象？ （ 赋值，浅拷贝，深拷贝的区别 ） 293](#_Toc525305309)

[14) 写一个函数，输入一个字符串，返回倒序排列的结果，输入： string\_reverse(‘abcdef’) 返回：’fedcba’，写出你能想到的多种方法 293](#_Toc525305310)

[11.33. Weego旅游 294](#_Toc525305311)

[1) Python里面如何拷贝一个对象 294](#_Toc525305312)

[2) 描述Python的垃圾回收机制 294](#_Toc525305313)

[3) Python的装饰器的实现及应用 294](#_Toc525305314)

[4) 什么是鸭子类型？ 295](#_Toc525305315)

[5) 简述Python中的容器、可迭代对象、迭代器、生成器 295](#_Toc525305316)

[6) Python多线程、多进程和协程的区别 295](#_Toc525305317)

[7) 简述Python中list、tuple、set、dict的特性和区别 296](#_Toc525305318)

[8) 使用Python实现单例模式 296](#_Toc525305319)

[11.34. 课堂在线 296](#_Toc525305320)

[1) 去除列表中的重复元素 297](#_Toc525305321)

[2) 字符串和列表的转换 297](#_Toc525305322)

[3) xrange和range的区别 297](#_Toc525305323)

[4) 将下面的函数改写成lambda形式 297](#_Toc525305324)

[5) Cookie和Session的区别 297](#_Toc525305325)

[6) 实现一个二分查找 297](#_Toc525305326)

[7) Django中Q的使用 298](#_Toc525305327)

[8) 简述Django中间件处理过程 298](#_Toc525305328)

[9) 写一个装饰器用来计算函数的执行时间 298](#_Toc525305329)

[11.35. 思百达 299](#_Toc525305330)

[1) 下列哪个语句在Python中是非法的？ 299](#_Toc525305331)

[2) 以下不能创建一个字典的语句是 299](#_Toc525305332)

[3) a与b定义如下，下列哪个是正确的？ 299](#_Toc525305333)

[4) 下列代码执行结果是？ 299](#_Toc525305334)

[5) 下面代码运行后，a、b、c、d四个变量的值，描述错误的是？ 299](#_Toc525305335)

[6) 有如下函数定义，执行结果正确的是？ 299](#_Toc525305336)

[7) 下面代码运行结果是？ 300](#_Toc525305337)

[1) 请写出以下代码的输出： 301](#_Toc525305338)

[2) 下面代码的执行结果是 301](#_Toc525305339)

[3) Python实现：斐波那契数列即著名的兔子数列：1、1、2、3、5、8、13、21、34 301](#_Toc525305340)

[4) 给定三角形的坐标，用Python如何求三角形的面积？ 301](#_Toc525305341)

[5) Python实现快速排序算法 302](#_Toc525305342)

[11.36. 感易智能 302](#_Toc525305343)

[1) 请设计一个数组类，数组的大小由用户输入，完成三个函数： 302](#_Toc525305344)

[2) 现有一颗二叉树如下，请使用迭代写出先序遍历算法. 303](#_Toc525305345)

[3) 请写出二叉树的“Z字型”遍历算法（迭代/递归都可以） 303](#_Toc525305346)

[4) 输入两个整数序列，第一个序列表示栈的压入顺序，请判断第二个序列是否为该栈的弹出顺序。假设压入栈的所有数字均不相等。例如序列1, 2, 3, 4, 5 是某栈的压入顺序，序列4, 5, 3, 2, 1是该压栈序列对应的一个弹出序列，但4, 3, 5, 1, 2就不可能是该压栈序列的弹出序列。（注意：这两个序列的长度是相等的） 304](#_Toc525305347)

[5) Django如何连接两个DB数据库？ 305](#_Toc525305348)

[6) Python dumps 能转换哪些类型？ 305](#_Toc525305349)

[7) range与xrange有什么区别？ 305](#_Toc525305350)

[8) Linux如何查看内存情况？ 305](#_Toc525305351)

[9) Django代码修改后Celery是否会自动重启？ 305](#_Toc525305352)

[10) 多个Django实例需要搭配多少个Celery？ 305](#_Toc525305353)

[11) Django ORM的save() 有什么参数？ 306](#_Toc525305354)

[11.37. 汇游科技 ( 智力测试 ) 306](#_Toc525305355)

[1) 五个答案中哪一个是最好的类比？\_\_A\_\_ 306](#_Toc525305356)

[2) 找出与众不同的一个\_\_\_C\_\_\_ 306](#_Toc525305357)

[3) 全班学生排成一行，从左数和从右数小明都是第15名，问：全班共有学生\_\_C\_\_人。 306](#_Toc525305358)

[4) 一个立方体的六面，分别写着A、B、C、D、E、F六个字母，根据下面四张图，推测B的对面是E字母。 306](#_Toc525305359)

[5) 找出与“确信”意义相同或者意义最相近的词\_\_\_E\_\_\_ 306](#_Toc525305360)

[6) 如果所有的甲都是乙，没有一个乙是丙，那么，一定没有一个丙是甲，这句话是\_\_A\_\_\_ 306](#_Toc525305361)

[7) 找出下列数字中特殊的一个\_\_15\_\_ 307](#_Toc525305362)

[8) 小明比小强大，小红比小明小，下列陈述中哪一句最正确\_\_\_D\_\_\_ 307](#_Toc525305363)

[9) 五个答案中哪一个是最好的类比？ 307](#_Toc525305364)

[10) 小明有12枚硬币，共3角6分钱，其中有5枚硬币是一样的，那么这五枚硬币一定是\_\_C\_\_ 307](#_Toc525305365)

[11) 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\_\_\_ 307](#_Toc525305366)

[12) 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 307](#_Toc525305367)

[13) 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\_\_ 307](#_Toc525305368)

[14) 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\_\_C\_\_趟。 307](#_Toc525305369)

[15) 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。 307](#_Toc525305370)

[16) 找出与众不同的一个\_\_\_B\_\_\_\_ 307](#_Toc525305371)

[17) 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厂，火车的机车、车厢、守车共长\_\_\_C\_\_米。 307](#_Toc525305372)

[18) 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（ 月 ）巴 307](#_Toc525305373)

[19) 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。 307](#_Toc525305374)

[20) 数数有\_\_D\_\_个三角形 307](#_Toc525305375)

[11.38. 汇游科技 308](#_Toc525305376)

[1) 写程序得到两个列表list的交集和差集 308](#_Toc525305377)

[2) 写出一个单例类 308](#_Toc525305378)

[3) 用Python正则匹配HTML tag的时候，<.\*>和<.\*?>用什么区别 308](#_Toc525305379)

[4) 下面的代码会输出什么： 309](#_Toc525305380)

[5) 平衡点：比如int[] numbers = {1, 3, 5, 7, 8, 25, 4, 24}；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 309](#_Toc525305381)

[6) 将语句SQL查询用Django的ORM进行查询：SELECT \* FROM order WHERE id BETWEEN 20 AND 100 AND ( num <= ‘20’ or num >= ‘30’ ); 309](#_Toc525305382)

[7) Django常见的线上部署方式有哪几种 309](#_Toc525305383)

[8) Django对数据查询结果排序怎么做，降序怎么做 309](#_Toc525305384)

[9) MySQL中varchar与char的区别以及varchar(50)中50代表的涵义 310](#_Toc525305385)

[10) 请简述项目中优化SQL语句执行效率的方法 310](#_Toc525305386)

[11) git pull 和 git fetch 的区别 310](#_Toc525305387)

[12) git merge 和 git rebase 的区别 310](#_Toc525305388)

[13) 如何回滚代码到指定分支 311](#_Toc525305389)

[14) 在CentOS 7.2 中， 用一句话将所有的test.py进程全部杀死 311](#_Toc525305390)

[15) 在CentOS 7.2 中， 如何查看程序执行所消耗的CPU、内存等硬件资源 311](#_Toc525305391)

[11.39. 蓝汛 312](#_Toc525305392)

[1) 游程编码 是一种无损数据压缩编码。通过记录字母连续重复次数的方式，节约存储空间。举例“BBCCCCABDDD”，可编码为“2B4CAB3D” ( 如果某个字符仅连续出现一次则要省略前面的数字1 ) 312](#_Toc525305393)

[2) 去除多余的嵌套，假设有一个列表，其中每个元素可能为一个整数，或者为一个列表，需要你将多余的嵌套去掉，将所有的整数保存至一个新的列表中。例如[ 1, [ 2, [ 3 ], [ 4, 5 ]], [ 6, 7 ]] , 去除嵌套后为 [ 1, 2, 3, 4, 5, 6, 7 ]。 提示： type函数可以获取一个对象的类型，如 type(5) == int , type([1,2]) == list 312](#_Toc525305394)

[3) 归并排序，假设有多个已经排序后的列表，你需要将它们归并为一个有序列表。例如 [[ 3, 4, 5 ], [ 1, 2, 3 ], [ 4, 5 ]], 归并后为 [ 1, 2, 3, 3, 4, 4, 5 ]，要求：不能使用内置的sort函数 313](#_Toc525305395)

[11.40. OKAY新东方教育研究院 314](#_Toc525305396)

[1) int、float、str、tuple、list、set、dict，前面这些类型哪些可以做dict的key，哪些可以做dict的value？ 314](#_Toc525305397)

[2) 请写出一段Python代码实现删除一个list里面的重复元素 314](#_Toc525305398)

[3) 写出结果为 [ 1, 4, 9, 16, 25, ... , 100 ] 的列表推导式。 314](#_Toc525305399)

[4) 写出结果为 { 1:1, 2:4, 3:9, 4:16, 5:25, ..., 10:100 } 的字典推导式。 314](#_Toc525305400)

[5) 下面的代码会输出什么： 314](#_Toc525305401)

[6) a = [ 1, 2, 3, 4, 5 ], a[::2] = ? , a[-2:] = ? 315](#_Toc525305402)

[7) 写出print的结果 315](#_Toc525305403)

[8) List = [ -2, 1, 3, -6 ], 如何实现以绝对值大小从小到大将List中内容排序。 315](#_Toc525305404)

[11.41. 上海梅赛德斯奔驰 315](#_Toc525305405)

[1) Given an array of integers and an integer, return all pairs in the array which sum up tp the given integer. 315](#_Toc525305406)

[2) Use a single array to implement three stacks. Please use Python. 316](#_Toc525305407)

[3) Database skills. 317](#_Toc525305408)

[11.42. 互影科技 318](#_Toc525305409)

[1) 十进制数值17.5625的二进制表示是： 318](#_Toc525305410)

[2) 将下列存储设备按访问速度从低到高排序：内存、磁盘HDD、SSD、寄存器 318](#_Toc525305411)

[3) 列举5个以上常见的UNIX/LINUX操作命令，再列举3个以上常见的系统调用 318](#_Toc525305412)

[4) 有两个相距2000km的机房A和B。现有100GB的数据需要通过一条FTP连接在100秒内从A传到B。网络信号传输速度为2e5km/s，假设机房间宽带足够高，那么A节点的发送缓冲区最小可以设置为\_\_\_。 318](#_Toc525305413)

[5) 在请求页式存储管理中，一个程序页面走向为3、4、2、1、4、5、3、4、5、1、2，采用LRU算法。设分配给该程序的存储块数为3，则在该访问中发生的缺页次数是\_\_8次\_\_。 318](#_Toc525305414)

[6) 数据表有1e6个元素，如果仅需要求出其中最大的10个元素，采用\_\_堆排序\_\_算法比较好。 318](#_Toc525305415)

[11.43. 好巧网 318](#_Toc525305416)

[1) （简答题）下列代码的输出 318](#_Toc525305417)

[2) （简答题）下列代码的输出 319](#_Toc525305418)

[3) （简答题）下面代码会输出什么 319](#_Toc525305419)

[4) （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？ 320](#_Toc525305420)

[5) （简答题）下面代码会输出什么 320](#_Toc525305421)

[6) （简答题）下面代码会输出什么 320](#_Toc525305422)

[7) （编码提）实现二分查找：在一个排序数组中找一个数，返回该数出现的任意位置，如果数值不存在，则返回-1，例如在数组[ 1, 3, 5, 7, 9 ] 查找元素“3”， 返回位置“1” 320](#_Toc525305423)

[11.44. 思沃创智 321](#_Toc525305424)

[1) 简述Python 中is 和= =的区别？ 321](#_Toc525305425)

[2) 简述read,readline 和readlines 的区别？ 321](#_Toc525305426)

[3) 举例说明创建字典的至少两种方法 322](#_Toc525305427)

[4) Python 代码中(\*args, \*\*kwargs)是什么意思 322](#_Toc525305428)

[5) 简述Python 里面search 和match 的区别？ 322](#_Toc525305429)

[6) 一个函数接受文件夹的名称作为输入参数，返回该文件夹中所有文件的全路径。请补充缺失的代码。 322](#_Toc525305430)

[7) 阅读下面的代码，写出A0，A1至An的最终值 323](#_Toc525305431)

[8) Python 代码实现:删除一个list 里面的重复元素 323](#_Toc525305432)

[9) 创建一个装饰器把下面函数输出的字符串首字母大写。 324](#_Toc525305433)

[10) 请将下面这段话翻译成英文： 324](#_Toc525305434)

[11.45. 保险师 324](#_Toc525305435)

[1) 下列哪个表达式在Python中是非法的？ 324](#_Toc525305436)

[2) python my.py v1 v2 命令运行脚本，通过from sys import argv 如何获得v2的参数值？ 324](#_Toc525305437)

[3) 如何解释下面的执行结果？ 324](#_Toc525305438)

[4) 下列代码执行结果是什么？ 324](#_Toc525305439)

[5) 下列哪种类型是Python的映射类型？ 325](#_Toc525305440)

[6) 下述字符串格式化语法正确的是？ 325](#_Toc525305441)

[7) 在Python 2.7 中，下列哪种是Unicode编码的书写方式？ 325](#_Toc525305442)

[8) 下列代码的运行结果是？ 325](#_Toc525305443)

[9) 下列代码运行结果是？ 325](#_Toc525305444)

[10) 下列哪种不是Python元组的定义方式？ 325](#_Toc525305445)

[11) a与b定义如下，下列哪个是正确的？ 325](#_Toc525305446)

[12) 下列对协程的理解错误的是？ 325](#_Toc525305447)

[13) 下列哪种函数参数定义不合法？ 325](#_Toc525305448)

[14) 下列代码执行结果是？ 325](#_Toc525305449)

[15) 一段代码定义如下，下列调用结果正确的是？ 326](#_Toc525305450)

[16) 下面代码运行结果？ 326](#_Toc525305451)

[17) 下面代码运行后，a、b、c、d四个变量的值，描述错误的是？ 326](#_Toc525305452)

[18) 有如下函数定义，执行结果正确的是？ 326](#_Toc525305453)

[19) 有如下类定义，下列描述错误的是？ 327](#_Toc525305454)

[20) 下列代码运行结果是？ 327](#_Toc525305455)

[21) Python中函数是对象，描述正确的是？ 327](#_Toc525305456)

[22) 若 a = range(100) ，以下哪些操作是合法的？ 327](#_Toc525305457)

[23) 若 a = ( 1, 2, 3 )，下列哪些操作是合法的？ 327](#_Toc525305458)

[24) Python中单下划线\_foo与双下划线\_\_foo与\_\_foo\_\_的成员，下列说法正确的是？ 327](#_Toc525305459)

[25) \_\_new\_\_和\_\_init\_\_的区别，说法正确的是？ 327](#_Toc525305460)

[26) 列表（list）和元组（tuple）的区别 327](#_Toc525305461)

[27) 标准库re正则表达式match和search区别 328](#_Toc525305462)

[28) 什么是装饰器，如何使用装饰器 328](#_Toc525305463)

[29) 在类的方法定义中“self”是怎样的一个参数 328](#_Toc525305464)

[30) 说明Cpython的内存管理机制 328](#_Toc525305465)

[11.46. 中科院信工所 329](#_Toc525305466)

[1) 关于进程和线程，下面说法不正确的是（ ）。 329](#_Toc525305467)

[2) 用二维表数据来表示实体与实体之间的联系的数据模型称为（ ）。 329](#_Toc525305468)

[3) 下列协议中不是电子邮件协议的是（ ）。 329](#_Toc525305469)

[4) “当多个事物并发执行时，任一事务的更新操作直到其成功提交的整个过程，对其他事务都是不可见的”，这一性质通常被称为事务的（ ）。 329](#_Toc525305470)

[5) 引用调用方式下进行的函数调用，是将（ ）。 329](#_Toc525305471)

[6) 编译程序对高级语言进行编译的过程中，要不断收集、记录和使用源程序中的一些相关符号的类型和特征信息，并将去存入（ ）中。 329](#_Toc525305472)

[7) 假如某计算机同中的资源R的可用数为6，系统中有三个进行竞争R，且每个进程都需要i个R，该系统可能会发生死锁的最小值是（ ）。若信号量S当前的值为2， 则R的可用数和等待R的进程数分别为（ ）。 329](#_Toc525305473)

[8) 二叉树如右图所示，若进行顺序存储（即用一维数组元素的存储该二叉树中的结点且通过下标反应结点的关系，例如，对于下标为i的结点，其左孩子的下标为2i、右孩子的下标为2i+1），则该数组的大小至少为（ ）；若采用三叉链表存储该二叉树（各个节点包括结点数据、父结点指针、左孩子指针、右孩子指针），则该链表的所有结点中空指针的数目为（ ）。 330](#_Toc525305474)

[9) 实现二分查找（折半查找）时，要求查找表（ ）。 330](#_Toc525305475)

[10) IP地址块155.32.80.192/26包含了（ ）个主机地址，以下IP中不属于这个网络地址的是（ ）。 330](#_Toc525305476)

[11) 32位和64位机器中long型数据分别占（ ）个字节，同时32位和64位机器的指针占（ ）字节。 330](#_Toc525305477)

[12) 有如下一段代码 330](#_Toc525305478)

[13) 两个线程 330](#_Toc525305479)

[14) 下列关于Django的ORM叙述正确的是（ ）。 331](#_Toc525305480)

[15) 下面代码运行后，a、b、c、d四个变量的值，描述错误的是（ ）。 331](#_Toc525305481)

[16) 关于以下程序段，输出是（ ）。 331](#_Toc525305482)

[17) 解释型语言的特性有什么（ ）。 331](#_Toc525305483)

[18) python读取文件内容并且生成迭代器，以下做法正确的是（ ）。 331](#_Toc525305484)

[19) 请问经过下列编码的字符串的解码顺序是（ ）。 331](#_Toc525305485)

[20) python异常处理机制错误的是（ ）。 332](#_Toc525305486)

[21) python my.py v1 v2命令运行脚本，通过from sys import argv 如何获得v2 的参数值（ ）。 332](#_Toc525305487)

[22) 在python2中，下列哪种函数参数定义不合法（ ）。 332](#_Toc525305488)

[23) 下面list1、list2、list3的输出值分别是（ ）。 332](#_Toc525305489)

[24) 下列对协程的理解错误的是（ ）。 332](#_Toc525305490)

[25) 关于Python内存管理错误的是（ ）。 332](#_Toc525305491)

[26) 在开发WEB应用程序的时候，HTTP请求消息使用GET或POST方法以便在WEB上传数据，下列关于GET和POST方法描述正确的是（ ）。 333](#_Toc525305492)

[27) 下列说法正确的是（ ）。 333](#_Toc525305493)

[28) 下列关于Django的model继承形式正确的是（ ）。 333](#_Toc525305494)

[29) 下列关于Python装饰器说法错误的是（ ）。 333](#_Toc525305495)

[30) 下列python知识错误的是（ ）。 333](#_Toc525305496)

[11.47. 美景听听 333](#_Toc525305497)

[1) Python中如何进行内存管理的？ 333](#_Toc525305498)

[2) 如何用Python来进行查询和替换一个文本字符串？ 334](#_Toc525305499)

[3) Python中元组和列表有和不同？ 334](#_Toc525305500)

[4) “猴子补丁”指的是什么？适用场景是什么？ 334](#_Toc525305501)

[5) 说说decorator的用法和它的应用场景？请写一个装饰器 335](#_Toc525305502)

[6) 什么是Python的命名空间？ 335](#_Toc525305503)

[7) 如何在Python中拷贝一个对象？ 335](#_Toc525305504)

[8) Python的模块间循环引用的问题，如何避免它？ 335](#_Toc525305505)

[9) 如何使用上下文管理器？它的好处是什么？ 336](#_Toc525305506)

[10) Python如何实现单例模式？ 336](#_Toc525305507)

[11) 什么是协程？与进程和线程有何区别？ 336](#_Toc525305508)

[12) 说明os，sys模块不同，并列举常用的模块方法？ 337](#_Toc525305509)

[13) 列举一些让Python代码以并行方式运行的方法 338](#_Toc525305510)

[14) def multipliers(): 338](#_Toc525305511)

[15) def extendList(val, list=[]): 338](#_Toc525305512)

[16) class Parent(object): 338](#_Toc525305513)

[11.48. 中国电信 339](#_Toc525305514)

[1) www页面访问的大致过程如下： 339](#_Toc525305515)

[2) 当n=5时，下列函数的返回值是： （ ） 339](#_Toc525305516)

[3) 静态变量通常存储在进程哪个区？ （ ） 339](#_Toc525305517)

[4) 下列程序的输出是： （ ） 339](#_Toc525305518)

[5) IP地址131.153.12.71是一个（ ）类IP地址。 339](#_Toc525305519)

[6) S市A，B共有两个区，人口比例为3：5，据历史统计A区的犯罪率为0.01%，B区为0.015%，现有一起新案件发生在S市，那么案件发生在A区的可能性有多大？ 339](#_Toc525305520)

[7) 假设某计算机具有 1MB的内存(目前使用的计算机往往具有 64MB 以上内存),并按字节编址 ,为了能存取该内存各地址的内容,其地址寄存器至少需要二进制\_\_\_\_位.为使4 字节组成的字段从存储器中一次读出,要求存放存储器中的字边界对齐,一个字节的地址码应\_\_\_\_\_若存储器周期为 200ns,且每个周期可访问 4 个字节,则该存储器带宽为\_\_\_\_\_bit/s 假如程序员所用的地址为\_\_\_\_\_\_,而真正访问内存的地址称为\_\_\_\_\_\_\_ 340](#_Toc525305521)

[8) 下面代码会输出什么： 340](#_Toc525305522)

[9) 什么lambda函数？它有什么好处？ 340](#_Toc525305523)

[10) Python里面match()和search()的区别？ 340](#_Toc525305524)

[11) 线程与进程的区别？各有什么特点？ 341](#_Toc525305525)

[12) 进程间IPC（Inter-Process Communication）有哪些？ 341](#_Toc525305526)

[13) TCP/IP协议网络体系分层？每层的协议有哪些？ 342](#_Toc525305527)

[14) Python中，range与xrange的区别？哪个性能更高？ 342](#_Toc525305528)

[11.49. 库尚信息 342](#_Toc525305529)

[15) 写出下面程序的输出结果： 342](#_Toc525305530)

[16) 请排序如下列表数据，按age从小到大排序 343](#_Toc525305531)

[17) 请用Python实现基本数据结构栈（只需实现指定的方法即可） 343](#_Toc525305532)

[18) 请用Python实现一个函数重试装饰器 344](#_Toc525305533)

[19) 有如下表结构： 345](#_Toc525305534)

[20) 有如下表数据和SQL语句，请指出哪些查询能用上索引？ 346](#_Toc525305535)

[11.50. 一起科技 346](#_Toc525305536)

[1) 类变量和实例变量的区别？ 346](#_Toc525305537)

[2) Python新式类和旧式类的区别？ 347](#_Toc525305538)

[3) 描述下GIL线程全局锁。 347](#_Toc525305539)

[4) Python里的拷贝？ 347](#_Toc525305540)

[5) 描述下操作系统的死锁 347](#_Toc525305541)

[6) 描述MyISAM和InnoDB的区别？ 347](#_Toc525305542)

[7) 描述下数据库的事务。 348](#_Toc525305543)

[8) 描述下HTTP中POST和GET区别？ 348](#_Toc525305544)

[9) 编程实现链表成对调换，如：1 -> 2 -> 3 -> 4 -> 转换成 2 -> 1 -> 4 -> 3 。 349](#_Toc525305545)

[10) 编程实现单例模式。 349](#_Toc525305546)

[11.51. 智慧星光 350](#_Toc525305547)

[1) 请查看下列语句是否有错误，如果无错误，请写出该语句执行结果；如果有错误，请说明原因 350](#_Toc525305548)

[2) 如下代码段： 350](#_Toc525305549)

[3) 请写出下面代码的输出结果 351](#_Toc525305550)

[4) 请写出下面代码的输出结果 351](#_Toc525305551)

[5) 现有字典d = {'a': 26, 'g': 20, 'e': 22, 'c': 24, 'd': 23, 'f': 21, 'b': 25} , 请按照字段中的value字段进行排序 352](#_Toc525305552)

[6) 列出你知道的排序算法，并用代码实现其中一种： 352](#_Toc525305553)

[7) 请列出HTTP常用的请求报头及其功能意义 352](#_Toc525305554)

[8) 请简述常见的反爬虫和对应的解决方法 353](#_Toc525305555)

[9) 什么是Global Interpreter Lock（GIL）， 为什么引入GIL，并简要描述其利弊 353](#_Toc525305556)

[11.52. 芸品绿 354](#_Toc525305557)

[1) 选出以下表达式表述正确的选项： 354](#_Toc525305558)

[2) 写出由tupleA和tupleB得到res的具体实现过程 354](#_Toc525305559)

[3) Python代码获取命令行参数 354](#_Toc525305560)

[4) 已知ip=’192.168.0.100’ 代码实现提取ip的各部分并写入列表 354](#_Toc525305561)

[5) 已知 AList = [‘a’, ‘b’, ‘c’] , 将Alist转化为 ‘a,b,c’ 的实现过程 354](#_Toc525305562)

[6) 已知 StrA = “1234A6FASKKSJJLSKWLM<SJKL90” 355](#_Toc525305563)

[7) 已知 Alist = [ 1, 2, 3, 1, 3, 1, 2, 1, 3 ]，如何根据Alist得到[ 1, 2, 3 ] 355](#_Toc525305564)

[8) 编写一个函数，这个函数接受一个文件夹名称作为参数，显示文件夹中文件的路径，以及其中包含文件夹中文件的路径 355](#_Toc525305565)

[9) 1000以内的完美数（如果一个数恰好等于它的因子之和，则称该数为完美数） 355](#_Toc525305566)

[10) 有A.txt 和 B.txt两个文件，使用多进程分别读取这两个文件 356](#_Toc525305567)

[11) 输入两个列表，alist，blist，依次顺序比较两个list中的元素，如果alist的元素大于blist的元素，返回alist；如果alist的元素小于blist的元素，返回blist；如果两个list的所有元素都相等，返回alist，否则返回blist 356](#_Toc525305568)

[12) 一个大小为100G的文件etl\_log.txt，要读取文件中的内容，写出具体过程代码。 357](#_Toc525305569)

[13) 操作表tableA，根据字段age进行分组，查找出每个分组中有多少条数据，并将每个分组按照所包含的数量按照由多到少进行排序。 357](#_Toc525305570)

[14) 表tableB中的数据如下所示，其中在SEX中1表示男性，2表示女性： 357](#_Toc525305571)

[11.53. 车势科技 357](#_Toc525305572)

[1) 请编写算法，将整型转换为货币类型，如1234567转换成“$1,234,567”。 357](#_Toc525305573)

[2) 使用正则表达式编写程序，将字符串“<a href=’https://www.autoforce.net’>北京车势科技官网</a><a href=’https://dms.autoforce.net’>车势科技dms系统</a>”将其中的https://www.autoforce.net和https://dms.autoforce.net匹配出来 358](#_Toc525305574)

[3) 已知列表a=[{“k”:1,”v”:2}, {“k”:12, “v”:22}, {“k”:13, “v”:32}]，请用内置方法，写出列表按照键“k”的倒序排列的代码。 358](#_Toc525305575)

[4) MySQL操作： 358](#_Toc525305576)

[11.54. BOSS直聘 359](#_Toc525305577)

[1) 简述TCP断开握手的过程 359](#_Toc525305578)

[2) 简述进程、线程、协程的含义 360](#_Toc525305579)

[3) 简述静态方法、类方法、实例方法的区别？ 360](#_Toc525305580)

[4) dict的items()方法与iteritems()方法的不同？ 361](#_Toc525305581)

[5) 请简述魔法函数\_\_new\_\_、\_\_call\_\_、\_\_repr\_\_、\_\_str\_\_、\_\_setattr\_\_、\_\_getattr\_\_的含义是什么？ 361](#_Toc525305582)

[6) Django中如何读取和保存session，整个session的运行机制是什么？ 361](#_Toc525305583)

[7) Python判断变量类型时，判断一个变量的类型时为什么不适用type()，而是使用isinstance()方法 361](#_Toc525305584)

[8) 写一个函数，输入一个字符串，返回倒序排列的结果，如：string\_reverse(‘abcdef’)，返回：‘fedcba’ 362](#_Toc525305585)

[9) 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： 362](#_Toc525305586)

[10) nginx日志格式如下，tail显示http状态码不为200的整行日志 363](#_Toc525305587)

[11) MySQL中delete from 表名 和 turncate table 表名 的区别 363](#_Toc525305588)

[12) 在MySQL中，要生成下列结果，SQL语句如何实现？ 363](#_Toc525305589)

[11.55. 射雕 364](#_Toc525305590)

[1) 请分别用一句话解释下列各Linux命令的用途： 364](#_Toc525305591)

[2) 有一个目录logs，下面有多级的子目录，每个子目录下都有一些以“.log”结尾的日志文件，每个日志文件的内容和格式如下（以空格分隔各项，url内无空格）： 364](#_Toc525305592)

[3) 已知素数的定义为：大于1且只能被1和自身整除的整数。试编写一个函数，该函数接收一个大于1的整数n，输出小于n的全部素数 365](#_Toc525305593)

[4) 单例模式（Singleton） 365](#_Toc525305594)

[5) 什么是线程安全，请写出一个线程安全的计数器 365](#_Toc525305595)

[11.56. 观止创想 366](#_Toc525305596)

[1) 写出你使用语言的helloword程序 366](#_Toc525305597)

[2) 将当前时间格式化为“2014-02-03 11:12:02” 366](#_Toc525305598)

[3) 变量和常量的区别。声明一个整型常量，一个字符串常量 366](#_Toc525305599)

[4) 如何获取字符串”hello, world”的长度？ 366](#_Toc525305600)

[5) 将数组的每个值大于5的元素加10，写出代码： 366](#_Toc525305601)

[6) 整型和字符串类型如何相互转换？ 367](#_Toc525305602)

[7) 函数如何返回多个值？ 367](#_Toc525305603)

[8) 如何抛出和捕获异常？ 367](#_Toc525305604)

[9) 若使用js，则获取指定元素，及其子元素的文本。 367](#_Toc525305605)

[10) 任选一种设计模式（Factory， Command， MVC， Strategy， Singleton）并写出代码 367](#_Toc525305606)

[11) 写出发起http请求的例子 368](#_Toc525305607)

[12) python中使用lambda或函数变量，js/as使用closure，java和php提供函数，实现字符串的反转，譬如：“abcdefg”反转为“gfedcba” 368](#_Toc525305608)

[13) 编写函数，获取字符串中最小的数字，譬如：字符串“a9b82jd1d”，结果：整型1 369](#_Toc525305609)

[11.57. 小药药 369](#_Toc525305610)

[1) 简述数组和元组之间的区别是什么？ 369](#_Toc525305611)

[2) a = [ 1, 2, 4, 2, 4, 5, 6, 7, 8, 9, 0 ], 编写一段代码，去除重复元素 369](#_Toc525305612)

[3) 描述浅拷贝和深拷贝区别及应用场景： 370](#_Toc525305613)

[4) 请用Scrapy的xpath提取出“太原生活君”： 370](#_Toc525305614)

[5) 简述页面从输入域名到显示元素的完整解析流程，如何查看每个阶段的执行时间，常用优化方法 370](#_Toc525305615)

[11.58. 维朗科技 371](#_Toc525305616)

[1) 请写一个函数funA，接受一个list作为参数，返回值为该list中所有元素之和 371](#_Toc525305617)

[2) 请说出list和tuple的异同点 371](#_Toc525305618)

[3) range与xrange有什么不同 371](#_Toc525305619)

[4) 请解释下什么叫做装饰器？有何用途，请用代码说明 371](#_Toc525305620)

[5) 请解释Python中staticmethod、classmethod 372](#_Toc525305621)

[6) 请说出这几个函数的用途：map、filter、reduce、zip 372](#_Toc525305622)

[7) 请列举出你最常用到的python标准库模块，并说明该模块的主要功能 372](#_Toc525305623)

[8) 请写一个脚本：count\_word.py，接受一个文本文件名book.txt作为参数，输出文件中一共有多少个英文单词 373](#_Toc525305624)

[11.59. 造数科技 373](#_Toc525305625)

[1) What’s the introspection? iIIustrate it with examples. 373](#_Toc525305626)

[2) What’s the GIL? 373](#_Toc525305627)

[3) Briefly explain the differences of List and Tuple. 373](#_Toc525305628)

[4) Exchange the Keys and Values of a Dictionary with one line code and give the application circumstances. Exp. {“foo”:”bar”, “index”:1 } => {“bar”:”foo”, 1:”index” } 374](#_Toc525305629)

[5) What do these mean to you: @classmethod , @staticmethod, @property ? 374](#_Toc525305630)

[6) Birefly explain the differences of TCP and UDP? 374](#_Toc525305631)

[7) Birefly explain HTTP cookies and the way it works. 374](#_Toc525305632)

[8) Birefly explain HTTPS and how to sniff the message over HTTPS. 375](#_Toc525305633)

[9) Birefly explain the network TCP/IP five-layer software model. 375](#_Toc525305634)

[10) List all HTTP code you know and explain each of them. 375](#_Toc525305635)

[11) Implement an algorithm to reverse a linked list. 376](#_Toc525305636)

[12) Implement an algorithm to detect the existence of a circle in a linked list. 376](#_Toc525305637)

[11.60. 其他公司面试题 377](#_Toc525305638)

[1) B/S和C/S的区别 377](#_Toc525305639)

[2) 有两个磁盘文件A和B，各存放一行字母，要求把这两个文件中的信息合并（按字母顺序排列），输出到一个新文件C中。 377](#_Toc525305640)

[3) 实现用户登录，要求输错第一次则提醒”输错三次账户将被锁定”，输错第二次则提醒”已错误输入两次！”输错第三次则提示”当日拒绝再次登录” 378](#_Toc525305641)

[4) 加入有如下成绩表A： 378](#_Toc525305642)

[5) 下面哪个函数能够在Linux环境下创建一个子进程： 378](#_Toc525305643)

[6) 已知x=43, ch=“A”, y=1, 则表达式( x>y and ch<“B” and y )的值是： 378](#_Toc525305644)

[7) 下列表达式中返回为True的是： 379](#_Toc525305645)

[8) 以下代码的输出是什么？ 379](#_Toc525305646)

[9) Python中有三目运算符吗？如果有请写出简单例子 379](#_Toc525305647)

[10) 请使用一行代码实现1到100之间偶数计算，并将所有偶数保存到列表中 379](#_Toc525305648)

[11) 下面的哪个语句会无限循环下去： 379](#_Toc525305649)

[12) robots协议是什么？ 379](#_Toc525305650)

[13) 平衡点问题： 380](#_Toc525305651)

[12. Python知识点汇总（ 命令版 ） 380](#_Toc525305652)

[12.1. 类型和运算 380](#_Toc525305653)

[12.2. 语法和语句 388](#_Toc525305654)

[12.3. 函数语法规则 391](#_Toc525305655)

[12.4. 函数例子 395](#_Toc525305656)

[12.5. 模块moudle 397](#_Toc525305657)

[12.6. 类与面向对象 399](#_Toc525305658)

[12.7. 类的高级话题 402](#_Toc525305659)

[12.8. 异常相关 406](#_Toc525305660)

[12.9. Unicode和字节字符串 409](#_Toc525305661)

[12.10. 其他 411](#_Toc525305662)

# Python基础

## 基础语法

### 输入与输出

#### 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

#### a=1,b=2,不用中间变量交换a 和b 的值？

方法一：  
1. a = a+b  
2. b = a-b  
3. a = a-b  
方法二：  
1. a = a^b  
2. b =b^a  
3. a = a^b  
方法三：  
1. 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。

#### 在Python中如何使用多进制数字？

二进制数字由0和1组成，我们使用 0b 或 0B 前缀表示二进制数。  
>>> int(0b1010)  
10  
  
使用bin()函数将一个数字转换为它的二进制形式。  
>>> bin(0xf)  
‘0b1111’

>>> bin(10)

'0b1010'  
  
八进制数由数字 0-7 组成，用前缀 0o 或 0O 表示 8 进制数。  
>>> oct(8)  
‘0o10’  
  
十六进数由数字 0-15 组成，用前缀 0x 或者 0X 表示 16 进制数。  
>>> hex(16)  
‘0x10’

#### 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。

### 条件与循环

#### 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

再考虑以下代码，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}，{**'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，当后续操作修改a[**'num'**]的值的时候，l 中的值也会跟着改变，相当于浅拷贝。

#### 以下Python 程序的输出？

for i in range(5, 0, -1):

print(i)

5 4 3 2 1

### 文件操作

#### 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。  
  
详细链接： <https://blog.csdn.net/mxgsgtc/article/details/12048919>

#### 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？

**def** get\_lines():  
 l = []  
 **with** open(**'file.txt'**，**'rb'**) **as** f:  
 data = f.readlines(60000)  
 l.append(data)  
 **yield** l  
   
要考虑到的问题有：  
内存只有4G 无法一次性读入10G 的文件，需要分批读入。分批读入数据要记录每次读入数据的位置。分批每次读入数据的大小，太小就会在读取操作上花费过多时间。

#### 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。

### 异常

#### 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？

**except**: *#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

### 模块与包

#### 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。  
常用的科学计算库（如Numpy，Scipy，Pandas)。

#### 赋值、浅拷贝和深拷贝的区别？

一、赋值  
在Python 中，对象的赋值就是简单的对象引用。  
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。  
  
  
二、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。  
  
  
三、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。  
  
  
四、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。

#### 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？(可以用Python 标准库)

**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).days + 1

#### 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

#### 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 特性

#### 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?

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)

#### Python2 与Python3 的区别？

Python2 和 Python3的区别体现在如下几个方面：   
1. print的改变   
2. 编码的改变   
3. True和False的改变   
4. nonlocal关键字   
5. 迭代器的改变  
  
详细链接：<https://blog.csdn.net/t46414704152abc/article/details/79089451>

#### 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。  
  
详细链接： <http://www.woola.net/detail/2016-08-28-python-object-introspection.html>

#### 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。  
  
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。  
内置变量: 小写，两个前导下划线和两个后置下划线\_\_class\_\_  
  
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。  
  
避免通用名称， 诸如list, dict, sequence 或者element 这样的名称应该避免。

#### 解释一下 Python 中的三元运算子

[on true] **if** [expression] **else** [on false]  
如果表达式为True，就执行[on true]中的语句。否则，就执行[on false]中的语句。

#### “猴子补丁”（monkey patching）指的是什么？

“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。即在运行期间动态修改一个类或模块。

**class** A:  
 **def** func(self):  
 print(**'Hi'**)  
  
**def** monkey(self):  
 print(**'Hi, monkey'**)  
  
A.func = monkey  
  
a = A()  
a.func()  
  
>>> Hi, monkey

### Linux

#### Linux 的基本命令（怎么区分一个文件还是文件夹）

ls -F 在显示名称的时候会在文件夹后添加**'/'**，在文件后面加**'\*'**。

#### 日志以什么格式，存放在哪里？

日志以文本可以存储在**'/var/log/'**目录下后缀名为.log。

#### Linux 查看某个服务的端口?

netstat -anp | grep service\_name

#### ubuntu 系统如何设置开机自启动一个程序?

直接修改/etc/rc0.d ~ /etc/rc6.d 和/etc/rcS.d 文件夹的内容，添加需启动的程序，S 开头的表示启动，K 开头的表示不启动。

#### 在linux 中find 和grep 的区别

Linux 系统中grep 命令是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹配的行打印出来。grep 全称是Global Regular Expression Print，表示全局正则表达式版本，它的使用权限是所有用户。  
linux 下的find：  
功能：在目录结构中搜索文件，并执行指定的操作。此命令提供了相当多的查找条件，功能很强大。  
语法：find 起始目录寻找条件操作  
说明：find 命令从指定的起始目录开始，递归地搜索其各个子目录，查找满足寻找条件的文件并对之采取相关的操作。  
简单点说说，grep 是查找匹配条件的行，find 是搜索匹配条件的文件。

#### Linux 重定向命令有哪些？有什么区别？

command > filename 把标准输出重定向到一个新文件中

command >> filename 把标准输出重定向到一个文件中(追加)

command 1 > fielname 把标准输出重定向到一个文件中

command > filename 2>&1 把标准输出和标准错误一起重定向到一个文件中

command 2 > filename 把标准错误重定向到一个文件中

command 2 >> filename 把标准输出重定向到一个文件中(追加)

command >> filename 2>&1 把标准输出和标准错误一起重定向到一个文件中(追加)

command < filename >filename2 把command命令以filename文件作为标准输入，以filename2文件为标准输出

command < filename 把command命令以filename文件作为标准输入

command << delimiter 把从标准输入中读入，直至遇到delimiter分界符

command <&m 把文件描述符m作为标准输入

command >&m 把标准输出重定向到文件描述符m中

command <&- 把关闭标准输入

#### 软连接和硬链接的区别？

软连接类似Windows 的快捷方式，当删除源文件时，那么软链接也失效了。硬链接可以理解为源文件的一个别名，多个别名所代表的是同一个文件。当rm 一个文件的时候，那么此文件的硬链接数减1，当硬链接数为0 的时候，文件被删除。

#### 10 个常用的Linux 命令？

pwd 显示工作路径  
ls 查看目录中的文件  
cd /home 进入**'/ home'** 目录**'**cd .. 返回上一级目录  
cd ../.. 返回上两级目录  
mkdir dir1 创建一个叫做**'dir1'** 的目录**'**rm -f file1 删除一个叫做**'file1'** 的文件**'，-f 参数，忽略不存在的文件，从不给出提示。**rmdir dir1 删除一个叫做**'dir1'** 的目录**'**groupadd group\_name 创建一个新用户组  
groupdel group\_name 删除一个用户组  
tar -cvf archive.tar file1 创建一个非压缩的tarball  
chmod 777 filename 修改文件权限  
lsof -i:8000 查看8000端口的占用

#### Linux 关机命令有哪些？

reboot 重新启动操作系统  
shutdown –r now 重新启动操作系统，shutdown 会给别的用户提示  
shutdown -h now 立刻关机，其中now 相当于时间为0 的状态  
shutdown -h 20:25 系统在今天的20:25 会关机  
shutdown -h +10 系统再过十分钟后自动关机  
init 0 关机  
init 6 重启

#### 使用Linux命令查询file1中空行所在的行号

以aa.txt举例：  
方法1：sed -n **'/[a-zA-Z0-9@#$%^&\*]/!='** aa.txt  
方法2：grep -n ^$ aa.txt  
方法3：awk **'/^$/{print NR}'** aa.txt  
方法4：sed -n **'/^$/='** aa.txt

#### Linux查看内存、磁盘存储、io 读写、端口占用、进程等命令

1、查看内存：top  
2、查看磁盘存储情况：df -h  
3、查看磁盘IO读写情况：iotop（需要安装一下：yum install iotop）、iotop -o（直接查看输出比较高的磁盘读写程序）  
4、查看端口占用情况：netstat -tunlp | grep 端口号  
5、 查看进程：ps aux

#### unix 进程间通信方式

1. 管道（Pipe）：管道可用于具有亲缘关系进程间的通信，允许一个进程和另一个与它有共同祖先的进程之间进行通信。  
2. 命名管道（named pipe）：命名管道克服了管道没有名字的限制，因此，除具有管道所具有的功能外，它还允许无亲缘关系进程间的通信。命名管道在文件系统中有对应的文件名。命名管道通过命令mkfifo或系统调用mkfifo来创建。  
3. 信号（Signal）：信号是比较复杂的通信方式，用于通知接受进程有某种事件发生，除了用于进程间通信外，进程还可以发送信号给进程本身；linux除了支持Unix早期信号语义函数sigal外，还支持语义符合Posix.1标准的信号函数sigaction（实际上，该函数是基于BSD的，BSD为了实现可靠信号机制，又能够统一对外接口，用sigaction函数重新实现了signal函数）。  
4. 消息（Message）队列：消息队列是消息的链接表，包括Posix消息队列system V消息队列。有足够权限的进程可以向队列中添加消息，被赋予读权限的进程则可以读走队列中的消息。消息队列克服了信号承载信息量少，管道只能承载无格式字节流以及缓冲区大小受限等缺  
5. 共享内存：使得多个进程可以访问同一块内存空间，是最快的可用IPC形式。是针对其他通信机制运行效率较低而设计的。往往与其它通信机制，如信号量结合使用，来达到进程间的同步及互斥。  
6. 内存映射（mapped memory）：内存映射允许任何多个进程间通信，每一个使用该机制的进程通过把一个共享的文件映射到自己的进程地址空间来实现它。  
7. 信号量（semaphore）：主要作为进程间以及同一进程不同线程之间的同步手段。  
8. 套接口（Socket）：更为一般的进程间通信机制，可用于不同机器之间的进程间通信。起初是由Unix系统的BSD分支开发出来的，但现在一般可以移植到其它类Unix系统上：Linux和System V的变种都支持套接字。

### Git命令

#### git 合并文件有冲突，如何处理?

1、git merge 冲突了，根据提示找到冲突的文件，解决冲突如果文件有冲突，那么会有类似的标记  
2、修改完之后，执行git add 冲突文件名  
3、git commit 注意:没有-m 选项进去类似于vim 的操作界面，把conflict 相关的行删除掉直接push 就可以了，因为刚刚已经执行过相关merge 操作了。

#### git 中 merge 和 rebase命令 的区别

merge：会将不同分支的提交合并成一个新的节点，之前的提交分开显示，注重历史信息、可以看出每个分支信息，基于时间点 ， 遇到冲突，手动解决，再次提交  
  
rebase：将两个分支的提交结果融合成线性，不会产生新的节点，注重开发过程，遇到冲突，手动解决，继续操作

#### reset 与 rebase, pull 与 fetch 的区别

git reset 不修改commit相关的东西，只会去修改.git目录下的东西。  
git rebase 会试图修改你已经commit的东西，比如覆盖commit的历史等，但是不能使用rebase来修改已经push过的内容，容易出现兼容性问题。rebase还可以来解决内容的冲突，解决两个人修改了同一份内容，然后失败的问题。  
git pull pull=fetch+merge,  
使用git fetch是取回远端更新，不会对本地执行merge操作，不会去动你的本地的内容。

#### git如何实现v1.0 、v2.0 等版本的管理？

在命令行中，使用“git tag –a tagname –m “comment”可以快速创建一个标签。需要注意，命令行创建的标签只存在本地Git库中，还需要使用Git push –tags指令发布到TFS服务器的Git库中

#### 什么是gitlab？

gitlab是公司自己搭建的项目代码管理平台

#### git中 ".gitignore"文件的作用?

设置哪些文件不需要添加到版本管理中 （比如Python的.pyc文件和一些包含密码的配置文件等）

#### 什么是敏捷开发？

敏捷开发：是一种以人为核心、迭代、循序渐进的开发方式。  
  
它并不是一门技术，而是一种开发方式，也就是一种软件开发的流程。它会指导我们用规定的环节去一步一步完成项目的开发。因为它采用的是迭代式开发，所以这种开发方式的主要驱动核心是人

## 数据类型

### 字典（ Dict ）

#### 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])

#### 怎样获取字典中所有键的列表？

使用 keys() 获取字典中的所有键  
  
>>> mydict={**'a'**:1,**'b'**:2,**'c'**:3,**'e'**:5}  
>>> mydict.keys()  
  
dict\_keys([**'a'**, **'b'**, **'c'**, **'e'**])

#### 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。

#### 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}

#### 字典有哪些常用的内置方法？

dict.fromkeys(seq [,value])  
创建一个新字典，序列seq中元素作为字典的键，value（可选）作为字典所有键对应的初始值。  
  
dict.get(key[, default=**None**])  
返回指定键的值，如果值不在字典中返回default值(默认为none)  
  
key **in** dict  
如果键在字典dict里返回true，否则返回false。key -- 在字典中要查找的键  
  
dict.items()  
以列表返回可遍历的(键, 值) 元组数组。  
  
dict.keys()  
以列表返回一个字典所有的键。  
  
dict.values()  
以列表返回字典中的所有值。

#### 对于字典aaa={“a”:10, “b”:34, “c”:11, “d”:22}，写出程序对调key和value，产生新的字典

**def** swap\_key\_value(old\_dict):  
 new\_dict = {key:value **for** value,key **in** old\_dict.items()}  
 **return** new\_dict

### 字符串 （ String ）

#### 怎样将字符串转换大小写？

**'THIS IS STRING'**.lower() ==>> **'this is string'** *# 转换为小写***'this is string'**.upper() ==>> **'THIS IS STRING'** *# 转换为大写*

#### 如何理解Python 中字符串中的\字符？

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

#### 请反转字符串“aStr”?

print(**'aStr'**[::-1])

#### 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.split(**'|'**):  
 key, value = iterms.split(**':'**)  
 dict1[key] = value  
 **return** dict1

#### 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(alist):  
 **return** sorted(alist,key=**lambda** x:x[**'age'**],reverse=**True**)

#### 字符串格式化 % 和 format的区别

一、%s用法（%s的用法是写多少个，后面就要传多少个）  
例1：  
>>> **"my name is %s and I am %d years old"** %(**"xiaoming"**,18)  
**'my name is xiaoming and I am 18 years old'**二、format用法（基本语法是通过{}和:来代替%。format函数可以接受不限个参数，位置可以不按顺序）  
例3：*#设置指定位置，按默认顺序*>>> **"{} {}"**.format(**"hello"**,**"world"**)  
**'hello world'**例4：*# 设置指定位置*>>> **"{0} {1}"**.format(**"hello"**, **"world"**)  
**'hello world'**

#### 给定一个字符串作为输入，将其中的数字用数字前近邻的字母序列进行替换，字母序列重复的次数等于该数字，最后将字符串输出： 输入：”af1an2ti3” 输出：”afanantititi”

**def** print\_string(string):  
 cur = times = res = **''** sum\_times = **False  
 for** i **in** range(len(string)):  
 **if** string[i].isalpha():  
 sum\_times = **False  
 if** string[i].isdigit():  
 times += string[i]  
 sum\_times = **True  
 if** i != len(string) - 1:  
 **continue  
 else**:  
 sum\_times = **False  
 if not** sum\_times **and** times:  
 res += cur \* int(times)  
 cur = times = **''** sum\_times = **False** cur += string[i]  
 **return** res

#### 写尽可能多的str对象的方法

常见方法：  
capitalize(...)  
 S.capitalize() -> string 返回字符串第一个字符大写后的值。  
  
center(...)  
 S.center(width[, fillchar]) -> string  
 返回一个给定width的字符串，并且原字符串集中在该字符串的中间，其它部分使用指定的字符字符填充(默认是一个空格)  
  
count(...)  
 S.count(sub[, start[, end]]) -> int  
 返回子串sub在S中出现的次数。可选参数start、end解释为切片符号。  
  
decode(...)  
 S.decode([encoding[,errors]]) -> object  
 使用给定的编码encoding解码S。解码错误时，错误errors可以设定不同的错误处理方案，默认是“UnicodeDecodeError”，其他可能的值是“ignore”和“replace”以及任何其他编解码器注册过的名字。register\_error是能够处理的“UnicodeDecodeErrors”。  
  
encode(...)  
 S.encode([encoding[,errors]]) -> object  
 使用给定的编码encoding编码。编码出错时，错误errors可以设定不同的错误处理方案，默认是“strict”意味着错误上升到“UnicodeEncodeError”，其他可能的值是“ignore”,“replace”和“xmlcharrefreplace”以及其他编解码器注册过的名字。register\_error是能够处理的“UnicodeDecodeErrors”。  
  
endswith(...)  
 S.endswith(suffix[, start[, end]]) -> bool  
 如果S以指定的suffix结束，返回True,否则假。  
 用可选的参数start,在那个位置开始测试S。  
 用可选的参数end,在那个位置停止比较。  
 suffix也可以尝试一个字符串的元组。  
  
详细链接：https://blog.csdn.net/qq\_31792281/article/details/70176710

#### 怎么移除一个字符串中的前导空格？

字符串中的前导空格就是出现在字符串中第一个非空格字符前的空格。我们使用方法lstrip()可以将它从字符串中移除。  
如果我们想去除后缀空格，就用rstrip()方法。

#### 如何查询和替换一个文本中的字符串

可以使用re模块中的sub()函数或者subn()函数来进行查询和替换，  
格式：sub(replacement, string[,count=0])（replacement是被替换成的文本，string是需要被替换的文本，count是一个可选参数，指最大被替换的数量）  
subn()方法执行的效果跟sub()一样，不过它会返回一个二维数组，包括替换后的新的字符串和总共替换的数量

### 列表 （ List ）

#### 如何以就地操作方式打乱一个列表的元素？

为了达到这个目的，我们从random模块中导入shuffle()函数。  
  
>>> **from** random **import** shuffle  
>>> shuffle(mylist)  
>>> mylist  
运行结果：  
  
[3, 4, 8, 0, 5, 7, 6, 2, 1]

#### 下面代码的输出结果将是什么？

list = ['a', 'b', 'c', 'd', 'e']

print list[10:]

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)])

#### 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]

[1:]  
[1:7]  
[3:7]  
[-1]  
[1::2]

#### 下面这段代码的输出结果是什么？请解释？

def extendlist(val, list=[]):

list.append(val)

return list

list1 = extendlist(10)

list2 = extendlist(123, [])

list3 = extendlist('a')

print("list1 = %s" %list1)

print("list2 = %s" %list2)

print("list3 = %s" %list3)

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(lIn):

l1 = sorted(lIn) # O(N)

l2 = [i for i in l1 if i<0.5] # O(N)

return [i\*i for i in l2] # O(N-M)

def f2(lIn):

l1 = [i for i in lIn if i<0.5] # O(N)

l2 = sorted(l1) # O(N-M)

return [i\*i for i in l2] # O(N-M)

def f3(lIn):

l1 = [i\*i for i in lIn] # O(N)

l2 = sorted(l1) # O(N)

return [i for i in l1 if i<(0.5\*0.5)] # O(N)

f2() f1() f3()

时间复杂度，排序和遍历都为N， N-M肯定是优于N的

#### 获取1~100 被6 整除的偶数？

**def** A():  
 alist = []  
 **for** i **in** range(1，100):  
 **if** i % 6 == 0:  
 alist.append(i)  
 print(alist)

### 元祖 （ Tuple ）

#### 现有两元祖 (('a'),('b'),('c'),('d') ) ,请使用Python中的匿名函数生成列表 [ {'a':'c'},{'b':'d'}]

data = ((**'a'**),(**'b'**),(**'c'**),(**'d'**) )  
v = list(map(**lambda** x,y:{x:y},data[0:2],data[2:4]))  
*# data[0:2] = ('a'),('b')  
# data[2:4] = ('c'),('d')*

#### 在Python中数组和元组的区别是什么？

在python中，数组就是列表，列表是可变的，而元组是不可变的

### 集合 （ Set ）

#### 快速去除列表中的重复元素

a = [11,22,33,33,44,22,55]  
  
set(a)  
{11, 22, 33, 44, 55}

## 网络协议

#### URL的格式是什么？

Uniform Resource Locator / 统一资源定位符  
协议://用户名:密码@子域名.域名.顶级域名:端口号/目录/文件名.文件后缀?参数=值*#标志*拓展：  
URL又分为绝对URL和相对URL：  
绝对URL（absolute URL）显示文件的完整路径，这意味着绝对URL本身所在的位置与被引用的实际文件的位置无关  
相对URL（relative URL）以包含URL本身的文件夹的位置为参考点，描述目标文件夹的位置。

#### http协议头中，keep-alive字段有什么作用？

HTTP协议采用“请求-应答”模式，当使用普通模式，即非KeepAlive模式时，每个请求/应答客户和服务器都要新建一个连接，完成 之后立即断开连接（HTTP协议为无连接的协议）；  
当使用Keep-Alive模式（又称持久连接、连接重用）时，Keep-Alive功能使客户端到服 务器端的连接持续有效，当出现对服务器的后继请求时，Keep-Alive功能避免了建立或者重新建立连接。  
通过使用keep-alive机制，可以减少tcp连接建立次数，也意味着可以减少TIME\_WAIT状态连接，以此提高性能和提高httpd服务器的吞吐率(更少的tcp连接意味着更少的系统内核调用,socket的accept()和close()调用)。

#### 请求头里都可以得到什么内容？

请求头：   
Accept: text/html,image/\*(浏览器可以接收的类型)   
Accept-Charset: ISO-8859-1(浏览器可以接收的编码类型)   
Accept-Encoding: gzip,compress(浏览器可以接收压缩编码类型)   
Accept-Language: en-us,zh-cn(浏览器可以接收的语言和国家类型)   
Host: www.it315.org:80(浏览器请求的主机和端口)   
If-Modified-Since: Tue, 11 Jul 2000 18:23:51 GMT(某个页面缓存时间)   
Referer: http://www.it315.org/index.jsp(请求来自于哪个页面)   
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)(浏览器相关信息)   
Cookie：(浏览器暂存服务器发送的信息)   
Connection: close(1.0)/Keep-Alive(1.1)(HTTP请求的版本的特点)   
Date: Tue, 11 Jul 2000 18:23:51 GMT(请求网站的时间)

#### 简述TCP 和UDP 的区别以及优缺点?

UDP 是面向无连接的通讯协议，UDP 数据包括目的端口号和源端口号信息。  
优点：UDP 速度快、操作简单、要求系统资源较少，由于通讯不需要连接，可以实现广播发送  
缺点：UDP 传送数据前并不与对方建立连接，对接收到的数据也不发送确认信号，发送端不知道数据是否会正确接收，也不重复发送，不可靠。  
  
TCP 是面向连接的通讯协议，通过三次握手建立连接，通讯完成时四次挥手  
优点：TCP 在数据传递时，有确认、窗口、重传、阻塞等控制机制，能保证数据正确性，较为可靠。  
缺点：TCP 相对于UDP 速度慢一点，要求系统资源较多。

#### 描述用浏览器访问www.baidu.com 的过程

先要解析出baidu.com 对应的ip 地址  
. 要先使用arp 获取默认网关的mac 地址  
. 组织数据发送给默认网关(ip 还是dns 服务器的ip，但是mac 地址是默认网关的mac 地址)  
. 默认网关拥有转发数据的能力，把数据转发给路由器  
. 路由器根据自己的路由协议，来选择一个合适的较快的路径转发数据给目的网关  
. 目的网关(dns 服务器所在的网关)，把数据转发给dns 服务器  
. dns 服务器查询解析出baidu.com 对应的ip 地址，并原路返回请求这个域名的client  
  
得到了baidu.com 对应的ip 地址之后，会发送tcp 的3 次握手，进行连接  
. 使用http 协议发送请求数据给web 服务器  
. web 服务器收到数据请求之后，通过查询自己的服务器得到相应的结果，原路返回给浏览器。  
. 浏览器接收到数据之后通过浏览器自己的渲染功能来显示这个网页。  
. 浏览器关闭tcp 连接，即4 次挥手结束，完成整个访问过程

#### POST和GET请求的区别?

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。  
  
传输数据的大小：  
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。  
  
  
安全性：POST 的安全性比GET 的高。

效率：GET 比POST 效率高。  
  
POST 请求的过程：  
1.浏览器请求tcp 连接（第一次握手）  
2.服务器答应进行tcp 连接（第二次握手）  
3.浏览器确认，并发送post 请求头（第三次握手，这个报文比较小，所以http 会在此时进行第一次数据发送）  
4.服务器返回100 **continue** 响应  
5.浏览器开始发送数据  
6.服务器返回200 ok 响应  
  
GET 请求的过程：  
1.浏览器请求tcp 连接（第一次握手）  
2.服务器答应进行tcp 连接（第二次握手）  
3.浏览器确认，并发送get 请求头和数据（第三次握手，这个报文比较小，所以http 会在此时进行第一次数据发送）  
4.服务器返回200 OK 响应

#### HTTP与HTTPS协议有什么区别？

http协议是超文本传输协议，被用于在web浏览器和网站服务器之间传递信息。http协议工作是以明文方式发送内容，不提供任何形式的数据加密，而这也是很容易被黑客利用的地方，如果黑客截取了web浏览器和网站服务器之间的传输信息，就可以直接读懂其中的信息，因此http协议不适合传输一些重要的、敏感的信息，比如信用卡密码及支付验证码等。

安全套接字层https协议就是为了解决http协议的这一安全缺陷而出生的，为了数据传输的安全，https在http的基础上加入了ssl协议，ssl依靠证书来验证服务器的身份，为浏览器和服务器之间的通信加密，这样的话即使黑客借去了发送过程中的信息，也无法破解读懂它，我们网站及用户的信息便得到了最大的安全保障。  
  
https协议需要到ca申请证书，一般免费证书很少，需要费用。  
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  
http和https使用的是完全不同的连接方式用的端口也不一样,前者是80,后者是443。  
http的连接很简单,是无状态的， HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议 要比http协议安全

#### cookie 和session 的区别？

1:cookie数据存放在客户的浏览器上(客户端),session数据放在服务器上.  
 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,如果主要考虑到安全应当使用session  
 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,如果主要考虑到减轻服务器性能方面,应当使用cookie  
  
2:单个cookie在客户端的限制是4K,就是说一个站点在客户端存放的cookie不能4k  
 将登陆信息等重要信息存放为session;其他信息如果需要保留,可以放在 cookie 中  
  
3:session不能区分路径,同一个用户在访问一个网站期间,所有的session在任何一个地方都可以访问到.而cookie中如果设置了路径参数,那么同一个网站中不同路径下的cookie互相是访问不到的.cookie只能是子路径访问父路径设置的cookie

#### ARP协议

ARP (Address Resolution Protocol) 是个地址解析协议。最直白的说法是：在IP以太网中，当一个上层协议要发包时，有了该节点的IP地址，ARP就能提供该节点的MAC地址。  
为什么要有ARP？  
OSI 模式把网络工作分为七层，彼此不直接打交道，只通过接口(layre interface). IP地址在第三层, MAC地址在第二层。  
协议在发生数据包时，首先要封装第三层 （IP地址）和第二层 （MAC地址）的报头, 但协议只知道目的节点的IP地址，不知道其物理地址，又不能跨第二、三层，所以得用ARP的服务。  
详细说明：  
Ø 在网络通讯时，源主机的应用程序知道目的主机的IP地址和端口号，却不知道目的主机的硬件地址，而数据包首先是被网卡接收到再去处理上层协议的，如果接收到的数据包的硬件地址与本机不符，则直接丢弃。因此在通讯前必须获得目的主机的硬件地址。ARP协议就起到这个作用  
Ø 当一台主机把以太网数据帧发送到位于同一局域网上的另一台主机时，是根据 48位的以太网地址来确定目的接口的，设备驱动程序从不检查 IP数据报中的目的IP地址。ARP（地址解析）模块的功能为这两种不同的地址形式提供映射：32位的 IP地址和 48位的以太网地址  
  
  
详细链接：  
http://blog.csdn.net/tigerjibo/article/details/7351992  
https://www.cnblogs.com/luckyxiaoxuan/p/3395527.html

#### 三次握手，四次挥手

1. 客户端通过向服务器端发送一个SYN来创建一个主动打开，作为三路握手的一部分。客户端把这段连接的序号设定为随机数 A。  
2. 服务器端应当为一个合法的SYN回送一个SYN/ACK。ACK 的确认码应为 A+1，SYN/ACK 包本身又有一个随机序号 B。  
3. 最后，客户端再发送一个ACK。当服务端受到这个ACK的时候，就完成了三路握手，并进入了连接创建状态。此时包序号被设定为收到的确认号 A+1，而响应则为 B+1。  
  
  
客户端或服务器均可主动发起挥手动作，在socket编程中，任何一方执行close()操作即可产生挥手操作。  
（1）客户端A发送一个FIN，用来关闭客户A到服务器B的数据传送。  
（2）服务器B收到这个FIN，它发回一个ACK，确认序号为收到序号加1。和SYN一样，一个FIN将占用一个序号。  
（3）服务器B关闭与客户端A的连接，发送一个FIN给客户端A。  
（4）客户端A发回ACK报文确认，并将确认序号设置为收到序号加1。

#### 代理和反向代理

正向代理： 用浏览器访问时，被残忍的block，于是你可以在国外搭建一台代理服务器，让代理帮我去请求google.com，代理把请求返回的相应结构再返回给我。  
反向代理：反向代理服务器会帮我们把请求转发到真实的服务器那里去。Nginx就是性能非常好的反向代理服务器，用来做负载均衡。 正向代理的对象是客户端，反向代理的对象是服务端

# Python高级

## 元类

#### 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。

## 内存管理与垃圾回收机制

#### 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。  
  
  
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。

## 函数

### 函数参数

#### 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

[10]  
对于字符串、数字，传递是相应的值。

### 内建函数

#### 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。

#### 解释 Python 中的 help() 和 dir() 函数

Help() 函数是一个内置函数，用于查看函数或模块用途的详细说明：  
  
>>> **import** copy  
>>> help(copy.copy)  
  
  
Dir() 函数也是 Python 内置函数，dir() 函数不带参数时，返回当前范围内的变量、方法和定义的类型列表；带参数时，返回参数的属性、方法列表。  
  
>>> dir(copy.copy)

#### 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；

#### print(list(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。  
  
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()。  
  
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

#### 一句话解决阶乘函数？

*# 仅针对python2 ， Python3中没有reduce函数*reduce(**lambda** x,y: x\*y, range(1,n+1))

### Lambda

#### 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。  
  
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)]

## 设计模式

### 单例

#### 全局变量和单例模式有什么区别

（1）全局变量是对一个对象的静态引用，全局变量确实可以提供单例模式实现的全局访问功能，但是它并不能保证应用程序只有一个实例；编码规范也明确的指出应该要少使用全局变量，因为过多的使用全局变量会造成代码难读；全局变量并不能实现继承。  
  
（2）单例模式虽然在继承上不能很好的处理，但是还是可以实现继承的；单例模式在类中保存了它的唯实例这个类，可以保证只能创建一个实例，同时它还提供了一个访问该唯一实例的全局访问点。

#### 请手写一个单例

*# 使用\_\_new\_\_方法***class** Singleton(object):  
 **def** \_\_new\_\_(cls, \*args, \*\*kw):  
 **if not** hasattr(cls, **'\_instance'**):  
 orig = super(Singleton, cls)  
 cls.\_instance = orig.\_\_new\_\_(cls, \*args, \*\*kw)  
 **return** cls.\_instance  
  
*# 装饰器版本***def** singleton(cls, \*args, \*\*kw):  
 instances = {}  
 **def** getinstance():  
 **if** cls **not in** instances:  
 instances[cls] = cls(\*args, \*\*kw)  
 **return** instances[cls]  
 **return** getinstance

#### 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

### 装饰器

#### 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。

#### 解释一下什么是闭包? 手写一个闭包函数

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。

**def** outer(n):  
 **def** inner(num):  
 **return** n \* num  
 **return** inner

#### 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

#### 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 range(10))

X 是generator生成器类型。

#### 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。

#### 解释生成器（generator）与函数的不同

生成器和函数的主要区别在于函数 **return** a value，生成器 **yield** a value同时标记或记忆point of the **yield** 以便于在下次调用时从标记点恢复执行。 **yield** 使函数转换成生成器，而生成器反过来又返回迭代器。

## 面向对象

### 类

#### 解释一下 Python 中的继承

当一个类继承自另一个类，它就被称为一个子类/派生类，继承自父类/基类/超类。它会继承/获取所有类成员（属性和方法）。  
  
继承能让我们重新使用代码，也能更容易的创建和维护应用。Python 支持如下种类的继承：  
  
单继承：一个类继承自单个基类  
多继承：一个类继承自多个基类  
多级继承：一个类继承自单个基类，后者则继承自另一个基类  
分层继承：多个类继承自单个基类  
混合继承：两种或多种类型继承的混合

#### 请写一个Python类，要求包含public/private方法及变量的声明，同时定义一个静态方法

**class** PythonClass(object):  
 variable = **'python'  
  
 def** public\_method(self):  
 print(**'i am public'**)  
  
 **def** \_\_private(self):  
 print(**'i am private'**)  
  
 @staticmethod  
 **def** static():  
 print(**'i am static'**)

### 对象

#### 单下划线和双下划线在python的区别？

\_xxx 不能用’**from** module **import** \*’导入  
  
\_\_xxx\_\_ 系统定义名字  
  
\_\_xxx 类中的私有变量名  
  
  
以单下划线开头（\_foo）的代表不能直接访问的类属性，需通过类提供的接口进行访问，不能用“**from** xxx **import** \*”而导入；以双下划线开头的（\_\_foo）代表类的私有成员，意思是只有类对象自己能访问，连子类对象也不能访问到这个数据；以双下划线开头和结尾的（\_\_foo\_\_）代表python里特殊方法专用的标识，如 \_\_init\_\_（）代表类的构造函数

#### 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\_\_ 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?

**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

#### 在类的方法定义中，' self ' 是怎样的一个参数

在类的代码（函数）中，需要访问当前的实例中的变量和函数的，即，访问Instance中的：   
对应的变量（property)：Instance.ProperyNam，去读取之前的值和写入新的值   
调用对应函数（function）：Instance.function()，即执行对应的动作   
此处的Instance本身就是self。   
Python中的self等价于C++中的self指针和Java、C*#中的this参数。*

#### 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。

#### 如何判断一个Python对象的类型？

type(1) # int  
type(**'1'**) # str

## 正则表达式

#### 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'beijing'**，**'beijing.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'beijing'**，**'bj'**，**'beijing.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“<>”。

## 系统编程

#### select、poll、epoll各有哪些区别？

Select的3个缺点：  
1. 连接数受限  
2. 查找配对速度慢  
3. 数据由内核拷贝到用户态  
Poll 只是改变连接数的限制，后面的缺点没有改变，  
Epoll 在poll基础上改了select的3个缺点：  
1. 文件描述符数量没有上限；  
2. Epoll只是轮询就绪的文件描述符，不是全部轮询全部的，效率会快很多  
3. Epoll使用mmap(文件对象映射，别映射到多个内存页上)，改变了原来的拷贝模式

#### B/S和C/S的区别

1．硬件环境不同:   
C/S 一般建立在专用的网络上, 小范围里的网络环境, 局域网之间再通过专门服务器提供连接和数据交换服务.   
B/S 建立在广域网之上的, 不必是专门的网络硬件环境,例与电话上网, 租用设备. 信息自己管理. 有比C/S更强的适应范围, 一般只要有操作系统和浏览器就行   
2．对安全要求不同   
C/S 对服务端、客户端都安全都要考虑。  
B/S 因没有客户端，所以只注重服务端安全即可。  
3．系统维护不同   
系统维护是软件生存周期中,开销大, -------重要   
C/S 程序由于整体性, 必须整体考察, 处理出现的问题以及系统升级. 升级难. 可能是再做一个全新的系统   
B/S 构件组成,方面构件个别的更换,实现系统的无缝升级. 系统维护开销减到最小.用户从网上自己下载安装就可以实现升级.   
4．用户接口不同   
C/S 多是建立的Window平台上,表现方法有限,对程序员普遍要求较高   
B/S 建立在浏览器上, 通过WEB服务或其他公共可识别描述语言可跨平台，使用更灵活。不仅可应用在Window平台上，还可应用于unix/Linux等平台。  
5．信息流不同   
C/S 程序一般是典型的中央集权的机械式处理, 交互性相对低   
B/S 信息流向可变化, B-B B-C B-G等信息、流向的变化, 更像交易中心。

#### 在Python中如何实现多线程？

一个线程就是一个轻量级进程，多线程能让我们一次执行多个线程。我们都知道，Python 是多线程语言，其内置有多线程工具包。  
Python 中的 GIL（全局解释器锁）确保一次执行单个线程。一个线程保存 GIL 并在将其传递给下个线程之前执行一些操作，这会让我们产生并行运行的错觉。但实际上，只是线程在 CPU 上轮流运行。当然，所有的传递会增加程序执行的内存压力。

#### 谈谈你对多进程，多线程，以及协程的理解，项目是否用？

进程：一个运行的程序（代码）就是一个进程，没有运行的代码叫程序，进程是系统资源分配的最小单位，进程拥有自己独立的内存空间，所以进程间数据不共享，开销大。

线程： 调度执行的最小单位，也叫执行路径，不能独立存在，依赖进程存在一个进程至少有一个线程，叫主线程，而多个线程共享内存(数据共享，共享全局变量)，从而极大地提高了程序的运行效率。

协程：是一种用户态的轻量级线程，协程的调度完全由用户控制。协程拥有自己的寄存器上下文和栈。协程调度切换时，将寄存器上下文和栈保存到其他地方，在切回来的时候，恢复先前保存的寄存器上下文和栈，直接操作栈则基本没有内核切换的开销，可以不加锁的访问全局变量，所以上下文的切换非常快。

#### 进程池、线程池、连接池

池的概念  
  
池可以分为多种，常见的有内存池、进程池、线程池和连接池。  
由于服务器的硬件资源“充裕”，那么提高服务器性能的一个很直接的方法就是以空间换时间，即“浪费”服务器的硬件资源，以换取其运行效率。这就是池的概念。池是一组资源的集合，这组资源在服务器启动之初就完全被创建并初始化，这称为静态资源分配。当服务器进入正是运行阶段，即开始处理客户请求的时候，如果它需要相关的资源，就可以直接从池中获取，无需动态分配。很显然，直接从池中取得所需资源比动态分配资源的速度要快得多，因为分配系统资源的系统调用都是很耗时的。当服务器处理完一个客户连接后，可以把相关的资源放回池中，无需执行系统调用来释放资源。从最终效果来看，池相当于服务器管理系统资源的应用设施，它避免了服务器对内核的频繁访问。  
详细链接：  
https://blog.csdn.net/walkerkalr/article/details/37729323  
https://blog.csdn.net/sjyttkl/article/details/75577178

#### 什么是多线程竞争？

线程是非独立的，同一个进程里线程是数据共享的，当各个线程访问数据资源时会出现竞争状态即：  
数据几乎同步会被多个线程占用，造成数据混乱，即所谓的线程不安全那么怎么解决多线程竞争问题？-- 锁。  
锁的好处：  
确保了某段关键代码(共享数据资源)只能由一个线程从头到尾完整地执行能解决多线程资源竞争下的原子操作问题。  
锁的坏处：  
阻止了多线程并发执行，包含锁的某段代码实际上只能以单线程模式执行，效率就大大地下降了锁的致命问题：死锁。

#### 解释一下什么是锁，有哪几种锁?

锁(Lock)是Python 提供的对线程控制的对象。有互斥锁、可重入锁、死锁。

#### 什么是死锁？

若干子线程在系统资源竞争时，都在等待对方对某部分资源解除占用状态，结果是谁也不愿先解锁，互相干等着，程序无法执行下去，这就是死锁。

原因:  
1. 竞争资源  
2. 程序推进顺序不当

必要条件:  
1. 互斥条件  
2. 请求和保持条件  
3. 不剥夺条件  
4. 环路等待条件

处理死锁基本方法:  
1. 预防死锁(摒弃除1以外的条件)  
2. 避免死锁(银行家算法)  
3. 检测死锁(资源分配图)  
4. 解除死锁  
1. 剥夺资源  
2. 撤销进程

#### 什么是线程安全，什么是互斥锁？

每个对象都对应于一个可称为**" 互斥锁"** 的标记，这个标记用来保证在任一时刻，只能有一个线程访问该对象。  
同一个进程中的多线程之间是共享系统资源的，多个线程同时对一个对象进行操作，一个线程操作尚未结束，另一个线程已经对其进行操作，导致最终结果出现错误，此时需要对被操作对象添加互斥锁，保证每个线程对该对象的操作都得到正确的结果。

#### 说说下面几个概念：同步，异步，阻塞，非阻塞?

同步：多个任务之间有先后顺序执行，一个执行完下个才能执行。  
异步：多个任务之间没有先后顺序，可以同时执行有时候一个任务可能要在必要的时候获取另一个同时执行的任务的结果，这个就叫回调！  
阻塞：如果卡住了调用者，调用者不能继续往下执行，就是说调用者阻塞了。  
非阻塞：如果不会卡住，可以继续执行，就是说非阻塞的。  
同步异步相对于多任务而言，阻塞非阻塞相对于代码执行而言。

#### 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。  
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，同一个进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。

#### 利用python执行shell脚本并取得返回结果

**import** subprocess  
result = subprocess.getoutput(**'dir'**)

#### 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。

## 网络编程

#### 怎么实现强行关闭客户端和服务器之间的连接?

在socket 通信过程中不断循环检测一个全局变量(开关标记变量)，一旦标记变量变为关闭，则调用socket 的close 方法，循环结束，从而达到关闭连接的目的。

#### C10K问题

所谓c10k问题，指的是服务器同时支持成千上万个客户端的问题，也就是concurrent 10 000 connection（这也是c10k这个名字的由来）。  
推荐: http://www.kegel.com/c10k.html

#### 简述浏览器通过WSGI 请求动态资源的过程?

1.发送http 请求动态资源给web 服务器  
2.web 服务器收到请求后通过WSGI 调用一个属性给应用程序框架  
3.应用程序框架通过引用WSGI 调用web 服务器的方法，设置返回的状态和头信息。  
4.调用后返回，此时web 服务器保存了刚刚设置的信息  
5.应用程序框架查询数据库，生成动态页面的body 的信息  
6.把生成的body 信息返回给web 服务器  
7.web 服务器吧数据返回给浏览器

## Python编程题

#### 有这样一个文本文件，他的路径为baseDir，他的名字test.txt，要求with方式打开，并打印每一行文本，并要求文件路径考虑跨平台问题

**import** os  
baseDir = **r'C:\Users\haora'**filename = **'test.txt'**file\_path = os.path.join(baseDir,filename)  
**with** open(file\_path,**'r'**) **as** f:  
 **for** line **in** f.readlines():  
 print(line)

#### 有两个磁盘文件A和B，各存放一行字母，要求把这两个文件中的信息合并（按字母顺序排列），输出到一个新文件C中

lis\_A = open(**'a.txt'**).readlines()  
lis\_B = open(**'b.txt'**).readlines()  
  
lis\_C = zip(lis\_A,lis\_B)  
  
file\_C = open(**'c.txt'**, **'w'**)  
**for** txt **in** lis\_C:  
 s = **''**.join(txt).replace(**'\n'**, **''**)  
 file\_C.write(**''**.join(sorted(s)) + **'\n'**)  
file\_C.close()

#### 下面是一个单线程的代码，请改写成多线程的

start = "http://google.com"

queue = [start]

visited = {start}

while queue:

url = queue.pop(0)

print(url)

for next\_url in extract\_url(url):

if next\_url not in visited:

queue.append(next\_url)

visited.add(next\_url)

**from** concurrent.futures **import** ThreadPoolExecutor  
  
start = **"http://google.com"**queue = [start]  
visited = {start}  
pool = ThreadPoolExecutor(10)  
**def** func(url):  
 **for** next\_url **in** extract\_url(url):  
 **if** next\_url **not in** visited:  
 queue.append(next\_url)  
 visited.add(next\_url)  
**while** queue:  
 url = queue.pop(0)  
 pool.submit(func,url)  
pool.shutdown(wait=**True**)

#### 根据描述，写出接口调用的请求代码

服务请求地址：http://api.atguigu.com/ocr/

请求格式（POST/form-data）:

{

“user\_name”: “co\_demo\_xxx” #用户名

“submit\_key”: “xUOFzdMJMeiI/A==“ #验证信息

“pic”: File Object #上传图片文件

}

图片存储在本地：/root/img/test.jpg

请调用此服务，将本地图片POST到服务器

**import** requests  
  
**def** post\_file(user\_name, submit\_key, file\_path):  
 url = **'http://api.atguigu.com/ocr/'** file\_obj = open(file\_path,**'rb'**).read()  
 params = {  
 **'user\_name'**: user\_name, *# 用户名* **'submit\_key'**: submit\_key, *# 验证信息* **'pic'**: file\_obj, *#上传图片文件* }  
 res = requests.post(url, files=params)  
 **return** res  
  
post\_file(**'co\_demo\_xxx'**, **'xUOFzdMJMeiI/A=='**, **'/root/img/test.jpg'**)

# 前端

## HTML

#### 浏览器渲染原理解析

1、首先渲染引擎下载HTML，解析生成DOM Tree  
2、遇到css标签或JS脚本标签就新起线程去下载他们，并继续构建DOM。（其中css是异步下载同步执行）浏览器引擎通过 DOM Tree 和 CSS Rule Tree 构建 Rendering Tree  
3、 通过 CSS Rule Tree 匹配 DOM Tree 进行定位坐标和大小，这个过程称为 Flow 或 Layout 。  
4、最终通过调用Native GUI 的 API 绘制网页画面的过程称为 Paint 。  
当用户在浏览网页时进行交互或通过 js 脚本改变页面结构时，以上的部分操作有可能重复运行，此过程称为 Repaint 或 Reflow。 重排是指dom树发生结构变化后，需要重新构建dom结构。 重绘是指dom节点样式改变，重新绘制。 重排一定会带来重绘，重绘不一定有重排。  
如何减少浏览器重排：将需要多次重排的元素，position属性设为absolute或fixed，这样此元素就脱离了文档流，它的变化不会影响到其他元素。

#### iframe有那些缺点？

1. iframe会阻塞主页面的Onload事件；  
2. 搜索引擎的检索程序无法解读这种页面，不利于SEO;  
3. iframe和主页面共享连接池，而浏览器对相同域的连接有限制，所以会影响页面的并行加载。  
4. 使用iframe之前需要考虑这两个缺点。如果需要使用iframe，最好是通过javascript  
5. 动态给iframe添加src属性值，这样可以绕开以上两个问题。

#### Label的作用是什么？是怎么用的？

label标签来定义表单控制间的关系, 当用户选择该标签时，浏览器会自动将焦点转到和标签相关的表单控件上。  
<label **for**=**"Name"**>Number:</label>  
<input type=**'text'** name=**"Name"** id=**"Name"**/>  
<label> Date: <input type=**"text"** name=**"B"**/></label>

#### HTML5的form如何关闭自动完成功能？

给不想要提示的 form 或某个 input 设置为 autocomplete=off。

#### title与h1的区别、b与strong的区别、i与em的区别？

title属性没有明确意义只表示是个标题，H1则表示层次明确的标题，对页面信息的抓取也有很大的影响；  
strong是标明重点内容，有语气加强的含义，使用阅读设备阅读网络时： <strong> 会重读，而 <B> 是展示强调内容。  
i内容展示为斜体，me表示强调的文本；  
Physical Style Elements -- 自然样式标签  
b, i, u, s, pre  
Semantic Style Elements -- 语义样式标签  
strong, em, ins, **del**, code  
应该准确使用语义样式标签, 但不能滥用, 如果不能确定时首选使用自然样式标签。

#### 每个HTML文件里开头都有个很重要的东西，Doctype，知道这是干什么的吗？

文档声明。  
<!DOCTYPE> 声明位于文档中的最前面的位置，处于 <html> 标签之前。此标签可告知浏览器文档使用哪种 HTML 或 XHTML 规范。（重点：告诉浏览器按照何种规范解析页面）IE下如不书写文档声明会使用怪异模式解析网页导致一系列CSS兼容性问题。

#### div+css的布局较table布局有什么优点？

正常场景一般都适用div + CSS布局，  
优点：  
1.结构与样式分离  
2.代码语义性好  
3.更符合HTML标准规范  
4.SEO友好  
  
Table布局的适用场景：  
某种原因不方便加载外部CSS的场景，例如邮件正文，此时用table布局可以在无css情况下保持页面布局正常。

#### img的alt与title有何异同？ strong与em的异同？

a:alt(alt text):为不能显示图像、窗体或applets的用户代理（UA），alt属性用来指定替换文字。替换文字的语言由lang属性指定。(在IE浏览器下会在没有title时把alt当成 tool tip显示)  
title(tool tip):该属性为设置该属性的元素提供建议性的信息。  
em:表现为斜体，语义表示强调  
strong:表现为粗体，语义为强烈语气，强调程度超过em

#### 知道的网页制作会用到的图片格式有哪些？

png-8，png-24，jpeg，gif，svg。  
但是上面的那些都不是面试官想要的最后答案。面试官希望听到是Webp。（是否有关注新技术，新鲜事物）科普一下Webp：WebP格式，谷歌（google）开发的一种旨在加快图片加载速度的图片格式。图片压缩体积大约只有JPEG的2/3，并能节省大量的服务器带宽资源和数据空间。Facebook Ebay等知名网站已经开始测试并使用WebP格式。  
在质量相同的情况下，WebP格式图像的体积要比JPEG格式图像小40%

#### 在css/js代码上线之后开发人员经常会优化性能，从用户刷新网页开始，一次js请求一般情况下有哪些地方会有缓存处理？

DNS缓存，CDN缓存，浏览器缓存，服务器缓存。

#### 你如何理解HTML结构的语义化？

HTML结构语义化：  
更符合W3C统一的规范标准，是技术趋势。  
没有样式时浏览器的默认样式也能让页面结构很清晰。  
对功能障碍用户友好。屏幕阅读器（如果访客有视障）会完全根据你的标记来“读”你的网页。  
对其他非主流终端设备友好。例如机顶盒、PDA、各种移动终端。  
对SEO友好。

#### HTML5 Canvas 元素有什么用？

Canvas 元素用于在网页上绘制图形，该元素标签强大之处在于可以直接在 HTML 上进行图形操作。

#### 如何在 HTML5 页面中嵌入音频?

HTML 5 包含嵌入音频文件的标准方式，支持的格式包括 MP3、Wav 和 Ogg：  
<audio controls>  
 <source src=**"jamshed.mp3"** type=**"audio/mpeg"**>  
 Your browser does**'nt support audio embedding feature.**</audio>

#### 如何在 HTML5 页面中嵌入视频？

和音频一样，HTML5 定义了嵌入视频的标准方法，支持的格式包括：MP4、WebM 和 Ogg：  
<video width=**"450"** height=**"340"** controls>  
 <source src=**"jamshed.mp4"** type=**"video/mp4"**>  
 Your browser does**'nt support video embedding feature.**</video>

## CSS

#### 水平垂直居中的方法

行内布局  
line-height + text-align vertical-align + text-align  
  
块布局  
position absolute + margin auto position absolute + negative margin  
position absolute + translate(-50 %, -50 %)  
  
父容器子容器不确定宽高的块级元素，做上下居中  
  
1.flex  
2.tabel  
3.absolute + transform水平垂直居中  
4.webkit-box

详细链接: <https://github.com/hawx1993/tech-blog/issues/12>

#### 清除浮动元素的方法和各自的优缺点

清除浮动，实际上是清除父元素的高度塌陷。因为子元素脱离了父元素的文档流，所以，父元素失去了高度，导致了塌陷。要解决这个问题，就是让父元素具有高度。  
  
浮动元素的特性： 在正常布局中位于该浮动元素之下的内容，此时会围绕着浮动元素，填满其右侧的空间。浮动到右侧的元素，其他内容将从左侧环绕它（浮动元素影响的不仅是自己，它会影响周围的元素对其进行环绕。float仍会占据其位置，position: absolute不占用页面空间  
会有重叠问题 ）  
  
1.在浮动元素末尾添加空标签清除浮动  
clear: both （缺点：增加无意义标签）  
<div style=**"clear:both;"**></div>  
  
2.给父元素设置overflow: auto属性  
  
3.after伪元素

#### css实现自适应正方形

方案一：CSS3 vw 单位  
方案二：设置垂直方向的padding撑开容器  
方案三：利用伪元素的 margin(padding)-top 撑开容器

#### position的值

1. absolute :生成绝对定位的元素， 相对于最近一级的 定位不是 static 的父元素来进行定位。  
2. fixed （老IE不支持）生成绝对定位的元素，通常相对于浏览器窗口或 frame 进行定位。  
3. relative 生成相对定位的元素，相对于其在普通流中的位置进行定位。  
4. static 默认值。没有定位，元素出现在正常的流中  
5. sticky 生成粘性定位的元素，容器的位置根据正常文档流计算得出

#### CSS优先级算法如何计算？

\*优先级就近原则，同权重情况下样式定义最近者为准;  
\*载入样式以最后载入的定位为准;  
  
优先级为:  
同权重: 内联样式表（标签内部） > 嵌入样式表（当前文件中） > 外部样式表（外部文件中）。  
!important > id > **class** > tag  
  
important比内联优先级高

#### display有哪些值？说明他们的作用

block 块类型。默认宽度为父元素宽度，可设置宽高，换行显示。  
none 缺省值。象行内元素类型一样显示。  
inline 行内元素类型。默认宽度为内容宽度，不可设置宽高，同行显示。  
inline-block 默认宽度为内容宽度，可以设置宽高，同行显示。  
list-item 象块类型元素一样显示，并添加样式列表标记。  
table 此元素会作为块级表格来显示。  
inherit 规定应该从父元素继承 display 属性的值。

#### 为什么要初始化CSS样式

- 因为浏览器的兼容问题，不同浏览器对有些标签的默认值是不同的，如果没对CSS初始化往往会出现浏览器之间的页面显示差异。  
  
- 当然，初始化样式会对SEO有一定的影响，但鱼和熊掌不可兼得，但力求影响最小的情况下初始化。  
  
最简单的初始化方法： \*{padding: 0;margin: 0;} （强烈不建议）

#### position跟display、margin collapse、overflow、float这些特性相互叠加后会怎么样？

如果元素的display为none,那么元素不被渲染,position,float不起作用,如果元素拥有position:absolute;或者position:fixed;属性那么元素将为绝对定位,float不起作用.如果元素float属性不是none,元素会脱离文档流,根据float属性值来显示.有浮动,绝对定位,inline-block属性的元素,margin不会和垂直方向上的其他元素margin折叠.

#### CSS优化、提高性能的方法有哪些？

关键选择器（key selector）。选择器的最后面的部分为关键选择器（即用来匹配目标元素的部分）；  
如果规则拥有 ID 选择器作为其关键选择器，则不要为规则增加标签。过滤掉无关的规则（这样样式系统就不会浪费时间去匹配它们了）；  
提取项目的通用公有样式，增强可复用性，按模块编写组件；增强项目的协同开发性、可维护性和可扩展性;  
使用预处理工具或构建工具（gulp对css进行语法检查、自动补前缀、打包压缩、自动优雅降级）；

#### display:inline-block 什么时候会显示间隙？

移除空格、使用margin负值、使用font-size: 0、letter-spacing、word-spacing

#### 什么是外边距重叠？重叠的结果是什么？

外边距重叠就是margin-collapse。  
在CSS当中，相邻的两个盒子（可能是兄弟关系也可能是祖先关系）的外边距可以结合成一个单独的外边距。这种合并外边距的方式被称为折叠，并且因而所结合成的外边距称为折叠外边距。  
折叠结果遵循下列计算规则：  
两个相邻的外边距都是正数时，折叠结果是它们两者之间较大的值。  
两个相邻的外边距都是负数时，折叠结果是两者绝对值的较大值。  
两个外边距一正一负时，折叠结果是两者的相加的和。

#### px和em的区别

px和em都是长度单位，区别是：  
px值固定，容易计算。  
em值不固定，是相对单位，其相对应父级元素的字体大小会调整

#### 行内元素和块级元素的具体区别是什么？行内元素的padding和margin可设置吗？

块级元素(block)特性：  
总是独占一行，表现为另起一行开始，而且其后的元素也必须另起一行显示;  
宽度(width)、高度(height)、内边距(padding)和外边距(margin)都可控制;  
内联元素(inline)特性：  
和相邻的内联元素在同一行;  
宽度(width)、高度(height)、内边距的top/bottom(padding-top/padding-bottom)和外边距的top/bottom(margin-top/margin-bottom)都不可改变（也就是padding和margin的left和right是可以设置的），就是里面文字或图片的大小。  
那么问题来了，浏览器还有默认的天生inline-block元素（拥有内在尺寸，可设置高宽，但不会自动换行），有哪些？  
答案：<input> 、<img> 、<button> 、<textarea> 、<label>

#### 知道css有个content属性吗？有什么作用？有什么应用？

css的content属性专门应用在 before/after 伪元素上，用来插入生成内容。最常见的应用是利用伪类清除浮动。  
after伪元素通过 content 在元素的后面生成了内容为一个点的块级素，再利用clear:both清除浮动。  
那么问题继续还有，知道css计数器（序列数字字符自动递增）吗？如何通过css content属性实现css计数器？  
答案：css计数器是通过设置counter-reset 、counter-increment 两个属性 、及 counter()/counters()一个方法配合after / before 伪类实现。

## JavaScript

#### eval是做什么的？

它的功能是把对应的字符串解析成JS代码并运行；  
应该避免使用eval，不安全，非常耗性能（2次，一次解析成js语句，一次执行）。  
由JSON字符串转换为JSON对象的时候可以用eval，var obj =eval(**'('**+ str +**')'**);

#### 什么是window对象? 什么是document对象?

window对象是指浏览器打开的窗口。  
document对象是Documentd对象（HTML 文档对象）的一个只读引用，window对象的一个属性。

#### null，undefined 的区别？

null 表示一个对象是“没有值”的值，也就是值为“空”；  
undefined 表示一个变量声明了没有初始化(赋值)；  
undefined不是一个有效的JSON，而null是；  
undefined的类型(typeof)是undefined；  
null的类型(typeof)是object；  
  
Javascript将未赋值的变量默认值设为undefined；  
Javascript从来不会将变量设为null。它是用来让程序员表明某个用var声明的变量时没有值的。  
typeof undefined  
// **"undefined"**undefined: 是一个表示**"无"**的原始值或者说表示**"缺少值"**，就是此处应该有一个值，但是还没有定义。当尝试读取时会返回undefined；  
例如变量被声明了，但没有赋值时，就等于undefined  
typeof null  
// **"object"**null: 是一个对象(空对象, 没有任何属性和方法)；  
例如作为函数的参数，表示该函数的参数不是对象；  
  
注意：  
在验证null时，一定要使用　 == = ，因为 == 无法分别  
null  
和　undefined  
null == undefined // true  
null == = undefined // false  
再来一个例子：  
null  
Q：有张三这个人么？  
A：有！  
Q：张三有房子么？  
A：没有！  
  
undefined  
Q：有张三这个人么？  
A：有！  
Q: 张三有多少岁？  
A: 不知道（没有被告诉）

#### 事件是？IE与火狐的事件机制有什么区别？ 如何阻止冒泡？

1. 我们在网页中的某个操作（有的操作对应多个事件）。例如：当我们点击一个按钮就会产生一个事件。是可以被 JavaScript 侦测到的行为。  
2. 事件处理机制：I.E.是事件冒泡、Firefox同时支持两种事件模型，也就是：捕获型事件和冒泡型事件；  
3. ev.stopPropagation();（旧ie的方法 ev.cancelBubble = true;）

#### JSON 的了解？

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。  
它是基于JavaScript的一个子集。数据格式简单, 易于读写, 占用带宽小  
如：{**"age"**: **"12"**, **"name"**: **"back"**}  
  
JSON字符串转换为JSON对象:  
var obj = eval(**'('** + str + **')'**);  
var obj = str.parseJSON();  
var obj = JSON.parse(str);  
  
JSON对象转换为JSON字符串：  
var last = obj.toJSONString();  
var last = JSON.stringify(obj);

#### JS延迟加载的方式有哪些？

1. defer和async
2. 动态创建DOM方式（用得最多）
3. 按需异步载入js

#### javascript的typeof返回哪些数据类型

alert(typeof[1, 2]);  
// object  
  
alert(typeof **'guigu'**);   
// string  
  
var i = true;  
alert(typeof i);   
// boolean  
  
alert(typeof 1);   
// number  
  
var a;  
alert(typeof a);   
// undefined  
  
function a(){;};  
alert(typeof a)   
// function

#### 例举3种强制类型转换和2种隐式类型转换?

强制（parseInt(),parseFloat(),Number()）  
隐式（== ,!!）

#### 数组方法pop() push() unshift() shift()

push()尾部添加 pop()尾部删除  
unshift()头部添加 shift()头部删除  
map() : 遍历数组中的元素, 返回一个新数组(包含回调函数返回的数据)  
filter():遍历数组中的元素, 返回一个新数组(包含回调函数返回true的元素)

#### 事件绑定和普通事件有什么区别？

普通添加事件的方法：  
var btn = document.getElementById(**"hello"**);  
btn.onclick = function(){  
 alert(1);  
}  
  
btn.onclick = function(){  
 alert(2);  
}  
  
执行上面的代码只会alert 2  
  
  
  
事件绑定方式添加事件：  
var btn = document.getElementById(**"hello"**);  
btn.addEventListener(**"click"**, function(){  
 alert(1);  
}, false);  
  
btn.addEventListener(**"click"**, function(){  
 alert(2);  
}, false);  
  
执行上面的代码会先alert 1 再 alert 2  
普通添加事件的方法不支持添加多个事件，最下面的事件会覆盖上面的，而事件绑定（addEventListener）方式添加事件可以添加多个。  
addEventListener不兼容低版本IE  
普通事件无法取消  
addEventLisntener还支持事件冒泡 + 事件捕获

#### IE和DOM事件流的区别

1.执行顺序不一样、  
2.参数不一样  
3.事件加不加on  
4.this指向问题

#### window.onload 和document ready的区别

window.onload 是在dom文档树加载完和所有文件加载完之后执行一个函数document.ready原生中没有这个方法，jquery中有 $().ready(function),在dom文档树加载完之后执行一个函数（注意，这里面的文档树加载完不代表全部文件加载完）。  
$(document).ready要比window.onload先执行  
window.onload只能出来一次，$(document).ready可以出现多次

#### ”==”和“===”的不同

前者会自动转换类型  
后者不会

#### JavaScript是一门什么样的语言，它有哪些特点？

javaScript一种直译式脚本语言，是一种动态类型、弱类型、基于原型的语言，内置支持类型。它的解释器被称为JavaScript引擎，为浏览器的一部分，广泛用于客户端的脚本语言，最早是在HTML网页上使用，用来给HTML网页增加动态功能。JavaScript兼容于ECMA标准，因此也称为ECMAScript。  
基本特点  
1. 是一种解释性脚本语言（代码不进行预编译）。  
2. 主要用来向HTML（标准通用标记语言下的一个应用）页面添加交互行为。  
3. 可以直接嵌入HTML页面，但写成单独的js文件有利于结构和行为的分离。  
跨平台特性，在绝大多数浏览器的支持下，可以在多种平台下运行（如Windows、Linux、Mac、Android、iOS等）。

#### JavaScript的数据类型都有什么？

基本数据类型：String,boolean,Number,Undefined, Null  
引用数据类型：Object, Array, Function  
那么问题来了，如何判断某变量是否为数组数据类型？  
方法一.判断其是否具有“数组性质”，如slice()方法。可自己给该变量定义slice方法，故有时会失效  
方法二.obj instanceof Array 在某些IE版本中不正确  
方法三.方法一二皆有漏洞，在ECMA Script5中定义了新方法Array.isArray(), 保证其兼容性，最好的方法如下：  
 **if**(typeof Array.isArray===**"undefined"**){  
 Array.isArray = function(arg){  
 **return** Object.prototype.toString.call(arg)===**"[object Array]"** };  
 }

#### foo = foo||bar ，这行代码是什么意思？为什么要这样写？

如果foo存在，值不变，否则把bar的值赋给foo。  
短路表达式：作为”&&”和”||”操作符的操作数表达式，这些表达式在进行求值时，只要最终的结果已经可以确定是真或假，求值过程便告终止，这称之为短路求值。

#### 把两个数组合并，并删除第二个元素。

var array1 = [**'a'**,**'b'**,**'c'**];  
var bArray = [**'d'**,**'e'**,**'f'**];  
var cArray = array1.concat(bArray);  
cArray.splice(1,1);

#### 简述创建函数的几种方式

第一种（函数声明）：  
function sum1(num1,num2){  
 **return** num1+num2;  
}  
  
第二种（函数表达式）：  
var sum2 = function(num1,num2){  
 **return** num1+num2;  
}  
  
第三种（函数对象方式）：  
var sum3 = new Function(**"num1"**,**"num2"**,**"return num1+num2"**);

#### Javascript如何实现继承？

1.构造继承法  
2.原型继承法  
3.实例继承法

#### 哪些操作会造成内存泄漏？

内存泄漏指任何对象在您不再拥有或需要它之后仍然存在。  
垃圾回收器定期扫描对象，并计算引用了每个对象的其他对象的数量。如果一个对象的引用数量为 0（没有其他对象引用过该对象），或对该对象的惟一引用是循环的，那么该对象的内存即可回收。  
1. setTimeout 的第一个参数使用字符串而非函数的话，会引发内存泄漏。  
2. 闭包  
3. 控制台日志  
4. 循环（在两个对象彼此引用且彼此保留时，就会产生一个循环）

#### javascript 中的垃圾回收机制？

在Javascript中，如果一个对象不再被引用，那么这个对象就会被GC回收。如果两个对象互相引用，而不再 被第3者所引用，那么这两个互相引用的对象也会被回收。因为函数a被b引用，b又被a外的c引用，这就是为什么 函数a执行后不会被回收的原因。

#### 如何在HTML中添加事件，几种方法？

1、标签之中直接添加 onclick=**"fun()"**;  
2、JS添加 Eobj.onclick = method;  
3、绑定事件 IE： obj.attachEvent(**'onclick'**, method)；  
 FF: obj.addEventListener(**'click'**, method, false);

#### 你如何优化自己的代码？

代码重用  
避免全局变量（命名空间，封闭空间，模块化mvc..）  
拆分函数避免函数过于臃肿  
注释

#### 简述readyonly与disabled的区别

ReadOnly和Disabled的作用是使用户不能够更改表单域中的内容.  
但是二者还是有着一些区别的：  
1、Readonly只针对input(text/password)和textarea有效，而disabled对于所有的表单元素有效，包括select,radio,checkbox,button等。  
2、在表单元素使用了disabled后，我们将表单以POST或者GET的方式提交的话，这个元素的值不会被传递出去，而readonly会将该值传递出去

#### document.write和 innerHTML的区别

document.write只能重绘整个页面  
innerHTML可以重绘页面的一部分

#### 谈谈对this的理解

this总是指向函数的直接调用者（而非间接调用者）；  
如果有new关键字，this指向new出来的那个对象；  
在事件中，this指向触发这个事件的对象，特殊的是，IE中的attachEvent中的this总是指向全局对象Window；

## jQuery

#### jquery中如何将数组转化为json字符串，然后再转化回来？

jQuery中没有提供这个功能，所以你需要先编写两个jQuery的扩展：  
  
$.fn.stringifyArray = function(array){  
 **return** JSON.stringify(array)  
}  
  
$.fn.parseArray = function(array){  
 **return** JSON.parse(array)  
}  
  
然后调用：  
$(**""**).stringifyArray(array)

#### 针对 jQuery 的优化方法？

\*基于Class的选择性的性能相对于Id选择器开销很大，因为需遍历所有DOM元素。  
  
\*频繁操作的DOM，先缓存起来再操作。用Jquery的链式调用更好。  
比如：var  
str =$(**"a"**).attr(**"href"**);  
  
\* **for** (var i = size; i < arr.length; i++)  
{}  
**for** 循环每一次循环都查找了数组(arr) 的.length 属性，在开始循环的时候设置一个变量来存储这个数字，可以让循环跑得更快：  
**for** (var i = size, length = arr.length; i < length; i++) {}

#### JQuery一个对象可以同时绑定多个事件，这是如何实现的？

\*多个事件同一个函数：  
$(**"div"**).on(**"click mouseover"**, function()  
{});  
\*多个事件不同函数  
$(**"div"**).on({  
 click: function(){},  
 mouseover: function()  
{}  
});

#### bind(), live(), delegate()的区别

bind： 绑定事件，对新添加的事件不起作用，方法用于将一个处理程序附加到每个匹配元素的事件上并返回jQuery对象。  
live： 方法将一个事件处理程序附加到与当前选择器匹配的所有元素（包含现有的或将来添加的）的指定事件上并返回jQuery对象。  
delegate： 方法基于一组特定的根元素将处理程序附加到匹配选择器的所有元素（现有的或将来的）的一个或多个事件上。

#### jQuery框架中$.ajax()的常用参数有哪些？写一个post请求并带有发送数据和返回数据的样例

async是否异步  
url请求地址  
contentType发送信息至服务器时内容编码类型  
data发送到服务器的数据  
dataType预期服务器返回的数据类型  
type请求类型  
success请求成功回调函数  
error请求失败回调函数  
  
$.ajax({  
 url: **"/jquery/test1.txt"**,  
 type: **'post'**,  
 data: {  
 id: 1  
 },  
 success: function(data) {  
 alert(data);  
}  
}

## vue.js

#### vue 双向绑定底层实现原理

vue.js  
采用数据劫持的方式，结合发布者 - 订阅者模式，通过Object.defineProperty()  
来劫持各个属性的setter，getter以监听属性的变动，在数据变动时发布消息给订阅者，触发相应的监听回调：  
  
https://github.com/hawx1993/tech-blog/issues/11

#### vue 虚拟DOM和react 虚拟DOM的区别

在渲染过程中，会跟踪每一个组件的依赖关系，不需要重新渲染整个组件树。而对于React而言，每当应用的状态被改变时，全部子组件都会重新渲染。 在React应用中，当某个组件的状态发生变化时，它会以该组件为根，重新渲染整个组件子树。 如要避免不必要的子组件的重新渲染，你需要在所有可能的地方使用PureComponent，或是手动实现shouldComponentUpdate方法在React中，数据流是自上而下单向的从父节点传递到子节点，所以组件是简单且容易把握的，子组件只需要从父节点提供的props中获取数据并渲染即可。如果顶层组件的某个prop改变了，React会递归地向下遍历整棵组件树，重新渲染所有使用这个属性的组件。

#### vue组件通信

非父子组件间通信，Vue有提供Vuex，以状态共享方式来实现同信，对于这一点，应该注意考虑平衡，从整体设计角度去考量，确保引入她的必要。  
  
父传子: this.$refs.xxx  
子传父: this.$parent.xxx  
  
还可以通过$emit方法出发一个消息，然后$on接收这个消息

#### 你如何评价vue

框架能够让我们跑的更快，但只有了解原生的JS才能让我们走的更远。  
  
vue专注于MVVM中的viewModel层，通过双向数据绑定，把view层和Model层连接了起来。核心是用数据来驱动DOM。这种把directive和component混在一起的设计有一个非常大的问题，它导致了很多开发者滥用Directive（指令），出现了到处都是指令的情况。  
  
优点：   
1. 不需要setState，直接修改数据就能刷新页面，而且不需要react的shouldComponentUpdate就能实现最高效的渲染路径。   
2. 渐进式的开发模式，模版方式->组件方式->路由整合->数据流整合->服务器渲染。上手的曲线更加平滑简单，而且不像react一上来就是组件全家桶  
3. v - model给开发后台管理系统带来极大的便利，反观用react开发后台就是个杯具  
4. html，css与js比react更优雅地结合在一个文件上。  
  
缺点：指令太多，自带模板扩展不方便； 组件的属性传递没有react的直观和明显

#### 说说你对MVVM的理解

Model层代表数据模型，可以在Model中定义数据修改和操作业务逻辑； view代表UI组件。负责将数据模型转换成UI展现出来ViewModel是一个同步View和Model的对象  
用户操作view层，view数据变化会同步到Model，Model数据变化会立即反应到view中。viewModel通过双向数据绑定把view层和Model层连接了起来

#### 为什么选择vue

reactjs的全家桶方式，实在太过强势，而自己定义的JSX规范，揉和在JS的组件框架里，导致如果后期发生页面改版工作，工作量将会巨大。  
  
vue的核心：数据绑定和视图组件。  
  
Vue的数据驱动：数据改变驱动了视图的自动更新，传统的做法你得手动改变DOM来改变视图，vuejs只需要改变数据，就会自动改变视图，一个字：爽。再也不用你去操心DOM的更新了，这就是MVVM思想的实现。  
  
视图组件化：把整一个网页的拆分成一个个区块，每个区块我们可以看作成一个组件。网页由多个组件拼接或者嵌套组成

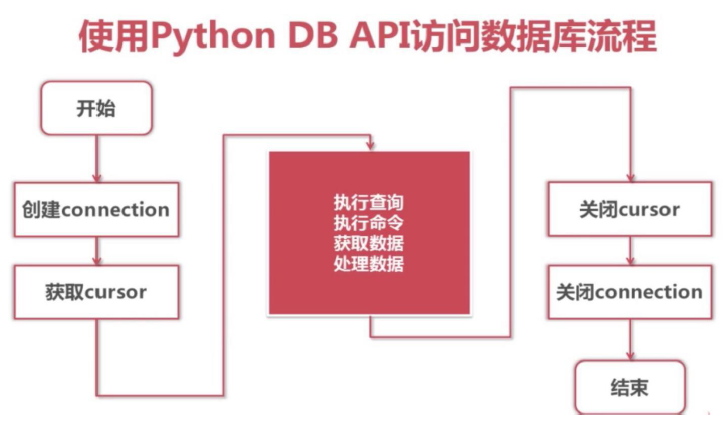
#### 谈谈你对组件的理解

一个组件应该有以下特征：  
  
1. 可组合（Composeable）：一个组件易于和其它组件一起使用，或者嵌套在另一个组件内部。如果一个组件内部创建了另一个组件，那么说父组件拥有（own）它创建的子组件，通过这个特性，一个复杂的  
2. UI可以拆分成多个简单的UI组件；  
3. 可重用（Reusable）：每个组件都是具有独立功能的，它可以被使用在多个UI场景；  
4. 可维护（Maintainable）：每个小的组件仅仅包含自身的逻辑，更容易被理解和维护；  
5. 可测试（Testable）：因为每个组件都是独立的，那么对于各个组件分别测试显然要比对于整个UI进行测试容易的多。

# 数据库

## MySQL

#### Python 中操作MySQL 步骤



1. 安装MySQL-Python包  
2. 创建Connection  
3. 获取cursor游标  
4. 执行查询、执行命令、获取数据、处理数据  
5. 关闭cursor游标  
6. 关闭Connection

#### 使用的MySQL版本号是多少？

MySQL常用版本号为5.6 / 5.7 / 5.8  
最新的MySQL版本号为8.0  
  
拓展： MySQL 4 - 5.6版本区别  
http://www.jb51.net/article/121638.htm  
拓展： MySQL 8.0 新特性  
http://www.jb51.net/article/137123.htm

#### SQL 的select 语句完整的执行顺序

1、from子句组装来自不同数据源的数据；   
2、where子句基于指定的条件对记录行进行筛选；   
3、group by子句将数据划分为多个分组；   
4、使用聚集函数进行计算；   
5、使用having子句筛选分组；   
6、计算所有的表达式；   
7、select 的字段；  
8、使用order by对结果集进行排序。

SQL 语言不同于其他编程语言的最明显特征是处理代码的顺序。在大多数据库语言中，代码按编码顺序被处理。但在SQL 语句中，第一个被处理的子句式FROM，而不是第一出现的SELECT。

SQL 查询处理的步骤序号：

(1) FROM <left\_table>

(2) <join\_type> JOIN <right\_table>

(3) ON <join\_condition>

(4) WHERE <where\_condition>

(5) GROUP BY <group\_by\_list>

(6) WITH {CUBE | ROLLUP}

(7) HAVING <having\_condition>

(8) SELECT

(9) DISTINCT

(9) ORDER BY <order\_by\_list>

(10) <TOP\_specification> <select\_list>

1、FROM：对FROM 子句中的前两个表执行笛卡尔积(交叉联接)，生成虚拟表VT1。

2、ON：对VT1 应用ON 筛选器，只有那些使为真才被插入到TV2。

3、OUTER (JOIN):如果指定了OUTER JOIN(相对于CROSS JOIN 或INNER JOIN)，保留表中未找到匹配的行将作为外部行添加到VT2，生成TV3。如果FROM 子句包含两个以上的表，则对上一个联接生成的结果表和下一个表重复执行步骤1 到步骤3，直到处理完所有的表位置。

4、WHERE：对TV3 应用WHERE 筛选器，只有使为true 的行才插入TV4。

5、GROUP BY：按GROUP BY 子句中的列列表对TV4 中的行进行分组，生成TV5。

6、CUTE|ROLLUP：把超组插入VT5，生成VT6。

7、HAVING：对VT6 应用HAVING 筛选器，只有使为true 的组插入到VT7。

8、SELECT：处理SELECT 列表，产生VT8。

9、DISTINCT：将重复的行从VT8 中删除，产品VT9。

10、ORDER BY：将VT9 中的行按ORDER BY 子句中的列列表顺序，生成一个游标(VC10)。

11、TOP：从VC10 的开始处选择指定数量或比例的行，生成表TV11，并返回给调用者。

#### 说一下MySQL 数据库存储的原理？

储存过程是一个可编程的函数，它在数据库中创建并保存。它可以有 SQL 语句和一些特殊的控制结构组成。当希望在不同的应用程序或平台上执行相同的函数，或者封装特定功能时，存储过程是非常有用的。数据库中的存储过程可以看做是对编程中面向对象方法的模拟。它允许控制数据的访问方式。 存储过程通常有以下优点:  
1)存储过程能实现较快的执行速度。  
2)存储过程允许标准组件是编程。  
3)存储过程可以用流控制语句编写，有很强的灵活性，可 以完成复杂的判断和较复杂的运算。  
4)存储过程可被作为一种安全机制来充分利用。  
5)存储过程能过减少网络流量

#### 事务的特性？

1．原子性(Atomicity):事务中的全部操作在数据库 中是不可分割的，要么全部完成，要么均不执行。  
2．一致性(Consistency):几个并行执行的事务，其执行 结果必须与按某一顺序串行执行的结果相一致。  
3．隔离性(Isolation):事务的执行不受其他事务的干扰， 事务执行的中间结果对其他事务必须是透明的。  
4、持久性(Durability):对于任意已 交事务，系统必须 保证该事务对数据库的改变不被丢失，即使数据库出

#### 数据库索引？

MYSQL数据库索引类型包括普通索引，唯一索引，主键索引与组合索引  
  
（1）普通索引  
这是最基本的MySQL数据库索引，它没有任何限制。  
CREATE INDEX indexName ON mytable(username(length));  
  
  
（2）唯一索引  
它与前面的普通索引类似，不同的就是：MySQL数据库索引列的值必须唯一，但允许有空值。如果是组合索引，则列值的组合必须唯一。  
创建索引  
CREATE UNIQUE INDEX indexName ON mytable(username(length))  
  
  
（3）主键索引  
它是一种特殊的唯一索引，不允许有空值。  
CREATE TABLE mytable( ID INT NOT NULL, username VARCHAR(16) NOT NULL, PRIMARY KEY(ID) );  
  
  
（4）组合索引  
为了形象地对比单列索引和组合索引，为表添加多个字段  
CREATE TABLE mytable( ID INT NOT NULL, username VARCHAR(16) NOT NULL, city VARCHAR(50) NOT NULL, age INT NOT NULL );

#### 乐观锁与悲观锁

悲观锁：假定会发生并发冲突，屏蔽一切可能违反数据完整性的操作。它指的是对数据被外界（包括本系统当前的其他事务，以及来自外部系统的事务处理）的修改持保守态度，因此，在整个数据处理过程中，将数据处于锁定状态。悲观锁的实现，往往依靠数据库提供的锁机制（也只有数据库层提供的锁机制才能真正保证数据访问的排他性，否则，即使在本系统中实现了加锁机制，也无法保证外部系统不会修改数据）。

乐观锁：假设不会发生并发冲突，只在提交操作时检查是否违反数据完整性。乐观锁假设认为数据一般情况下不会造成冲突，所以在数据进行提交更新的时候，才会正式对数据的冲突与否进行检测，如果发现冲突了，则让返回用户错误的信息，让用户决定如何去做。

详细链接：<http://blog.csdn.net/jirongzi_cs2011/article/details/43120145>

#### 数据库怎么优化查询效率？

1.储存引擎选择:如果数据表需要事务处理，应该考虑使用 InnoDB，因为它完全符合ACID 特性。如果不需要事务处理，使用默认存储引擎MyISAM 是比较明智的  
2.分表分库，主从  
3.对查询进行优化，要尽量避免全表扫 ，首先应考虑在 where 及 order by 涉及的列上建立索引  
4.应尽量避免在 where 子句中对字段进行 null 值判断， 否则将导致引擎放弃使用索引而进行全表扫  
5.应尽量避免在 where 子句中使用 != 或 <> 操作符， 否则将引擎放弃使用索引而进行全表扫  
6.应尽量避免在 where 子句中使用 **or** 来连接条件，如果 一个字段有索引，一个字段没有索引，将导致引擎放弃使用索 引而进行全表扫  
7.Update语句，如果只更改1、2 个字段，不要Update全部字段，否则频繁调用会引起明显的性能消耗，同时带来大 量日志

#### MySQL 集群的优缺点？

优点  
a) 99.999%的高可用性   
b) 快速的自动失效切换   
c)灵活的分布式体系结构，没有单点故障   
d)高吞吐量和低延迟   
e)可扩展性强，支持在线扩容  
  
缺点  
a)存在很多限制，比如：不支持外键   
b)部署、管理、配置很复杂   
c)占用磁盘空间大，内存大   
d)备份和恢复不方便   
e)重启的时候，数据节点将数据load到内存需要很长时间

#### 你用的MySQL 是哪个引擎，各引擎之间有什么区别？

主要 MyISAM 与 InnoDB 两个引擎，其主要区别如下：  
  
一、InnoDB 支持事务，MyISAM 不支持，这一点是非常之重要。事务是一种高级的处理方式，如在一些列增删改中只要哪个出错还可以回滚还原，而 MyISAM就不可以了；  
二、MyISAM 适合查询以及插入为主的应用，InnoDB 适合频繁修改以及涉及到安全性较高的应用；  
三、InnoDB 支持外键，MyISAM 不支持；  
四、MyISAM 是默认引擎，InnoDB 需要指定；  
五、InnoDB 不支持 FULLTEXT 类型的索引；  
六、InnoDB 中不保存表的行数，如 select count(\*) **from** table 时，InnoDB；需要扫描一遍整个表来计算有多少行，但是 MyISAM 只要简单的读出保存好的行数即可。注意的是，当 count(\*)语句包含 where 条件时 MyISAM 也需要扫描整个表；  
七、对于自增长的字段，InnoDB 中必须包含只有该字段的索引，但是在 MyISAM表中可以和其他字段一起建立联合索引；  
八、清空整个表时，InnoDB 是一行一行的删除，效率非常慢。MyISAM 则会重建表；  
九、InnoDB 支持行锁（某些情况下还是锁整表，如 update table set a=1 where user like **'%lee%'**

#### 数据库的优化？

1. 优化索引、SQL 语句、分析慢查询；  
2. 设计表的时候严格根据数据库的设计范式来设计数据库；  
3. 使用缓存，把经常访问到的数据而且不需要经常变化的数据放在缓存中，能节约磁盘IO；  
4. 优化硬件；采用SSD，使用磁盘队列技术(RAID0,RAID1,RDID5)等；  
5. 采用MySQL 内部自带的表分区技术，把数据分层不同的文件，能够提高磁盘的读取效率；  
6. 垂直分表；把一些不经常读的数据放在一张表里，节约磁盘I/O；  
7. 主从分离读写；采用主从复制把数据库的读操作和写入操作分离开来；  
8. 分库分表分机器（数据量特别大），主要的的原理就是数据路由；  
9. 选择合适的表引擎，参数上的优化；  
10. 进行架构级别的缓存，静态化和分布式；  
11. 不采用全文索引；  
12. 采用更快的存储方式，例如 NoSQL存储经常访问的数据

#### SQL的优化

1、SELECT子句中避免使用 \*， 尽量应该根据业务需求按字段进行查询  
2、尽量多使用COMMIT如对大数据量的分段批量提交释放了资源，减轻了服务器压力  
3、在写sql语句的话，尽量保持每次查询的sql语句字段用大写，因为oracle总是先解析sql语句，把小写的字母转换成大写的再执行  
4、用UNION - ALL替换UNION，因为UNION - ALL不会过滤重复数据，所执行效率要快于UNION, 并且UNION可以自动排序，而UNION - ALL不会  
5、避免在索引列上使用计算和函数, 这样索引就不能使用

#### MySQL 数据库如何分区、分表？

什么是分表？  
分表是将一个大表按照一定的规则分解成多张具有独立存储空间的实体表，我们可以称为子表，每个表都对应三个文件，MYD数据文件，.MYI索引文件，.frm表结构文件。这些子表可以分布在同一块磁盘上，也可以在不同的机器上。app读写的时候根据事先定义好的规则得到对应的子表名，然后去操作它。  
  
什么是分区？  
分区和分表相似，都是按照规则分解表。不同在于分表将大表分解为若干个独立的实体表，而分区是将数据分段划分在多个位置存放，可以是同一块磁盘也可以在不同的机器。分区后，表面上还是一张表，但数据散列到多个位置了。app读写的时候操作的还是大表名字，db自动去组织分区的数据。  
  
分表的几种方式：  
1、mysql集群  
它并不是分表，但起到了和分表相同的作用。集群可分担数据库的操作次数，将任务分担到多台数据库上。集群可以读写分离，减少读写压力。从而提升数据库性能。  
  
2、自定义规则分表  
大表可以按照业务的规则来分解为多个子表。通常为以下几种类型，也可自己定义规则。  
Range（范围）–这种模式允许将数据划分不同范围。例如可以将一个表通过年份划分成若干个分区。  
Hash（哈希）–这中模式允许通过对表的一个或多个列的Hash Key进行计算，最后通过这个Hash码不同数值对应的数据区域进行分区。例如可以建立一个对表主键进行分区的表。  
Key（键值）-上面Hash模式的一种延伸，这里的Hash Key是MySQL系统产生的。  
List（预定义列表）–这种模式允许系统通过预定义的列表的值来对数据进行分割。  
Composite（复合模式） –以上模式的组合使用  
  
3、利用merge存储引擎来实现分表  
merge分表，分为主表和子表，主表类似于一个壳子，逻辑上封装了子表，实际上数据都是存储在子表中的。  
我们可以通过主表插入和查询数据，如果清楚分表规律，也可以直接操作子表。

#### 如何对查询命令进行优化？

1.对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索  
2.应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：  
 select id **from** t where num **is** null  
 可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：  
 select id **from** t where num=0  
3.应尽量避免在 where 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描。  
4.应尽量避免在 where 子句中使用 **or** 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描，如：  
 select id **from** t where num=10 **or** num=20  
 可以这样查询：  
 select id **from** t where num=10  
 union all  
 select id **from** t where num=20  
5.**in** 和 **not in** 也要慎用，否则会导致全表扫描，如：  
 select id **from** t where num **in**(1,2,3)  
 对于连续的数值，能用 between 就不要用 **in** 了：  
 select id **from** t where num between 1 **and** 3  
6.下面的查询也将导致全表扫描：  
 select id **from** t where name like **'%abc%'** 若要提高效率，可以考虑全文检索。  
  
详细链接： <https://www.cnblogs.com/Mryang-blog-cn/p/SQLYOUHUA.html>

#### SQL 注入是如何产生的，如何防止？

1.sql注入产生的原因：   
程序开发过程中不注意书写规范，对sql语句和关键字未进行过滤，导致客户端可以通过全局变量get或者post提交sql语句到服务器端正常运行;  
  
2.防止过滤：   
1).过滤掉一些常见的数据库关键字：select、insert、update、delete、and等;或者通过系统函数addslashes(需要过滤的内容)来进行过滤；   
2).在PHP配置文件中register\_global=off;(设置为关闭状态)作用是将注册全局变量关闭掉；   
3).sql语句书写的时候尽量不要忽略小引号和单引号；   
4).提高数据库命名技巧，对于一些重要字段根据程序特点命名，取不易猜到的；   
5).对于常用的方法加以封装，避免直接暴露sql语句；   
6).开启安全模式，safe\_mode=on;   
7).打开magic\_quotes\_gpc=off,默认是关闭的，它打开后自动把用户提交的sql语句进行转换（加上\转义），这对防止sql注入有很大作用；因此开启magic\_quotes\_gpc=on   
8).控制错误信息：关闭错误提示信息，将错误信息写入系统日志文件；   
9).使用mysqli和pdo进行处理；

#### 数据库范式

第一范式（1NF）：属性不可分。比如某些数据库系统中需要用到“地址”这个属性，本来直接将“地址”属性设计成一个数据库表的字段就行。但是如果系统经常会访问“地址”属性中的“城市”部分，那么就非要将“地址”这个属性重新拆分为省份、城市、详细地址等多个部分进行存储，这样在对地址中某一部分操作的时候将非常方便。

第二范式（2NF）：符合1NF，并且，非主属性完全依赖于码（也就是说在一个数据库表中，一个表中只能保存一种数据，不可以把多种数据保存在同一张数据库表中）。

第三范式（3NF）：符合2NF，并且，消除传递依赖（每一列数据都和主键直接相关，而不能间接相关）。

#### NoSQL 和关系数据库的区别？

非关系型数据库：非关系型数据库产品是传统关系型数据库的功能阉割版本，通过减少用不到或很少用的功能，来大幅度提高产品性能。  
非关系型数据库严格上不是一种数据库，应该是一种数据结构化存储方法的集合。  
关系型数据库：是指采用了关系模型来组织数据的数据库。  
关系模型指的就是二维表格模型，而一个关系型数据库就是由二维表及其之间的联系所组成的一个数据组织。  
可以用SQL语句方便的在一个表以及多个表之间做非常复杂的数据查询。对于安全性能很高的数据访问要求可以实现

#### MySQL 数据库中怎么实现分页？

一：分页需求：  
客户端通过传递start(页码)，limit(每页显示的条数)两个参数去分页查询数据库表中的数据，那我们知道MySql数据库提供了分页的函数limit m,n，但是该函数的用法和我们的需求不一样，所以就需要我们根据实际情况去改写适合我们自己的分页语句，具体的分析如下：  
  
比如：  
  
查询第1条到第10条的数据的sql是：select \* **from** table limit 0,10; ->对应我们的需求就是查询第一页的数据：select \* **from** table limit (1-1)\*10,10;  
  
查询第10条到第20条的数据的sql是：select \* **from** table limit 10,20; ->对应我们的需求就是查询第二页的数据：select \* **from** table limit (2-1)\*10,10;  
  
查询第20条到第30条的数据的sql是：select \* **from** table limit 20,30; ->对应我们的需求就是查询第三页的数据：select \* **from** table limit (3-1)\*10,10;  
  
  
二：通过上面的分析，可以得出符合我们自己需求的分页sql格式是：select \* **from** table limit (start-1)\*limit,limit; 其中start是页码，limit是每页显示的条数。

#### SQL 语句怎么看效率？

Explain命令在解决数据库性能上是第一推荐使用命令，大部分的性能问题可以通过此命令来简单的解决，Explain可以用来查看 SQL 语句的执行效 果，可以帮助选择更好的索引和优化查询语句，写出更好的优化语句。  
Explain语法：explain select … **from** … [where ...]  
  
例如：explain select \* **from** news;

#### 优化数据库？提高数据库的性能？

一、硬件调整性能  
1、最有可能影响性能的是磁盘和网络吞吐量，解决办法就是扩大虚拟内存，以此来保证有足够扩充的空间；   
2、把数据库服务器上的不必要的服务关闭；   
3、把数据库服务器和主域服务器分开；   
4、把SQL数据库服务器的吞吐量调整到最大；   
5、 在具有一个以上处理器的机器上运行SQL。

二、调整数据库结构  
 若对某些特定的表进行查询频率高的，可以适当建立索引，建立索引时，按照where选择条件建立索引，尽量为整型建立为有且只有一个簇集索引，数据在物理上按顺序在数据也上，缩短了查找范围，在查询经常食用的全部列建立非簇集索引，能最大的覆盖查询，但是索引不能太多，执行insert，update，delete时需要维护这些索引，所以相对的将会增加数据库开销；避免在索引中有太多的索引键；避免使用大型数据类型的列为索引；保证每个索引键值有少数行。

三、使用存储过程  
 应用程序的实现过程中，能够采用存储过程实现的数据库操作则尽量通过存储过程来实现，，因为存储过程是存放在数据库服务器上的一次性被设计、编码、测试，并被再次使用，需要执行该任务的应用可以简单的执行存续过程，并且只返回结果集或者数值，这样不仅可以使程序模块化，同事提高了响应速度，减少网络流量，并且通过输入茶树接收输入，使得在应用中完成逻辑的一致性实现。

四、应用程序结构和算法   
 建立查询条件索引仅仅是提高速度的前提条件，响应速度的提高还依赖对索引的使用，因为人们在使用SQL时往往会陷入一个误区，太关注所得的结果是否正确，特别是对数据量不是特别大的数据库操作时，是否建立索引和使用索引的好坏对程序的响应速度并不大，因此程序员在书写程序时就会忽略不同的实现方法之间可能存在的性能差异，这种性能差异在数据量特别的时或者大型的复杂的数据库环境中尤为突出。在工作实践中发现，不良的SQL往往来自不恰当的索引设计、不充分的连接条件和不可优化的where子句。

#### 提取数据库中倒数10 条数据？

1. 数据库记录倒序排序，然后用limit 10子句获取10条：  
SELECT title AS TITLE FROM news ORDER BY id DESC LIMIT 10;  
  
2. 取得最大记录的id，然后在WHERE子句减去10  
SELECT temp.title **as** 标题 FROM news AS temp WHERE news\_id > (SELECT MAX(id) FROM news) - 10;

#### 数据库负载均衡

1. 基于数据库连接的负载均衡：例如总共有100个数据库连接，50个连接登录到数据库机器A，另外50个连接登录到数据库机器B，这样每个连接中接下来的所有请求全都是发往同一台数据库机器的。  
  
2.基于批处理请求的负载均衡：在建立数据库连接的时候，会同时与每台数据库服务器建立连接，之后针对客户端的每次请求，都会根据负载均衡算法，独立地选出某个数据库节点来执行这个请求。此种思路符合数据库长时间连接的特征，不存在上面所述的基于连接的负载均衡方法的缺点。

#### 存储过程和函数的区别?

存储过程和函数存在以下几个区别：  
  
1）一般来说，存储过程实现的功能要复杂一点，而函数的实现的功能针对性比较强。存储过程，功能强大，可以执行包括修改表等一系列数据库操作；用户定义函数不能用于执行一组修改全局数据库状态的操作。  
  
2）对于存储过程来说可以返回参数，如记录集，而函数只能返回值或者表对象。函数只能返回一个变量；而存储过程可以返回多个。存储过程的参数可以有IN,OUT,INOUT三种类型，而函数只能有IN类~~存储过程声明时不需要返回类型，而函数声明时需要描述返回类型，且函数体中必须包含一个有效的RETURN语句。  
  
3）存储过程，可以使用非确定函数，不允许在用户定义函数主体中内置非确定函数。  
  
4）存储过程一般是作为一个独立的部分来执行（ CALL 语句执行），而函数可以作为查询语句的一个部分来调用（SELECT调用）。 SQL语句中不可用存储过程，而可以使用函数

#### 关系型数据库的事务隔离级别有哪些，分别有什么影响？

1、未提交读(Read Uncommitted)  
直译就是**"读未提交"**, 意思就是即使一个更新语句没有提交, 但是别的事务可以读到这个改变.这是很不安全的。允许任务读取数据库中未提交的数据更改，也称为脏读。  
2、提交读(Read Committed)  
直译就是**"读提交"**, 可防止脏读，意思就是语句提交以后即执行了COMMIT以后别的事务就能读到这个改变.只能读取到已经提交的数据。Oracle等多数数据库默认都是该级别  
3、可重复读(Repeatable Read)：  
直译就是**"可以重复读"**, 这是说在同一个事务里面先后执行同一个查询语句的时候, 得到的结果是一样的.在同一个事务内的查询都是事务开始时刻一致的，InnoDB默认级别。在SQL标准中，该隔离级别消除了不可重复读，但是还存在幻象读  
4、串行读(Serializable)  
直译就是**"序列化"**, 意思是说这个事务执行的时候不允许别的事务并发执行.完全串行化的读，每次读都需要获得表级共享锁，读写相互都会阻塞  
  
四，隔离级别对事务并发的控制  
下表是各隔离级别对各种异常的控制能力:

丢失更新 脏读 非重复读 二类丢失更新(覆盖更新) 幻像读

未提交读 Y Y Y Y Y

提交读 N N Y Y Y

可重复读 N N N N Y

串行读 N N N N N

#### MVCC是什么？

Multi-Version Concurrency Control 多版本并发控制. 它的目标是在保证数据一致性的前提下，提供一种高并发的访问性能。在MVCC协议中，每个用户在连接数据库时看到的是一个具有一致性状态的镜像，每个事务在提交到数据库之前对其他用户均是不可见的。当事务需要更新数据时，不会直接覆盖以前的数据，而是生成一个新的版本的数据，因此一条数据会有多个版本存储，但是同一时刻只有最新的版本号是有效的。因此，读的时候就可以保证总是以当前时刻的版本的数据可以被读到，不论这条数据后来是否被修改或删除。  
MySQL的innodb引擎是如何实现MVCC的  
innodb会为每一行添加两个字段，分别表示该行创建的版本和删除的版本，填入的是事务的版本号，这个版本号随着事务的创建不断递增。在repeated read的隔离级别（事务的隔离级别请看这篇文章）下，具体各种数据库操作的实现：  
• select：满足以下两个条件innodb会返回该行数据：  
o 该行的创建版本号小于等于当前版本号，用于保证在select操作之前所有的操作已经执行落地。  
o 该行的删除版本号大于当前版本或者为空。删除版本号大于当前版本意味着有一个并发事务将该行删除了。  
• insert：将新插入的行的创建版本号设置为当前系统的版本号。  
• delete：将要删除的行的删除版本号设置为当前系统的版本号。  
• update：不执行原地update，而是转换成insert + delete。将旧行的删除版本号设置为当前版本号，并将新行insert同时设置创建版本号为当前版本号。  
其中，写操作（insert、delete和update）执行时，需要将系统版本号递增。  
 由于旧数据并不真正的删除，所以必须对这些数据进行清理，innodb会开启一个后台线程执行清理工作，具体的规则是将删除版本号小于当前系统版本的行删除，这个过程叫做purge。  
通过MVCC很好的实现了事务的隔离性，可以达到repeated read级别，要实现serializable还必须加锁。

#### 数据库表增加了新字段更新问题

MySQL增加了新字段后，如果设置了字段默认值，则原先所有数据在该字段的值更新为默认值。  
就像在更新MySQL字段后，在Django中执行Migrate操作必须要对新字段设置默认值一样

#### 有一个数据库表为student，字段有name（姓名），age（年龄），sex（性别），请按照以下要求编写SQL语句：

* 1. 增加一条学生记录（Jackson，19，Male）
  2. 查询年龄在18-20之间的所有学生
  3. 删除名为Jackson的学生记录

insert into student(name,age,sex) values(**'Jackson'**,19,**'Male'**);  
  
select \* **from** student where age between 18 **and** 20;  
  
delete **from** student where name=**'Jackson'**;

#### 列出几种常见的关系型数据库和非关系型数据库？

常见的关系型数据库：  
Oracle、DB2、PostgreSQL、Microsoft SQL Server、Microsoft Access、MySQL  
  
常见非关系模型数据库：  
列模型：Hbase  
键值对模型：redis, MemcacheDB  
文档类模型：mongoDB

## Redis

#### Redis是什么？

通常而言目前的数据库分类有几种，包括 SQL/NOSQL,，关系数据库，键值数据库等等，分类的标准也不以，Redis 本质上也是一种键值数据库的，但它在保持键值数据库简单快捷特点的同时，又吸收了部分关系数据库的优点。从而使它的位置处于关系数据库和键值数 据库之间。Redis 不仅能保存Strings 类型的数据，还能保存Lists 类型（有序）和Sets 类型（无序）的数据，而且还能完成排序（SORT） 等高级功能，在实现INCR，SETNX 等功能的时候，保证了其操作的原子性，除此以外，还支持主从复制等功能。

#### Redis可以用来做什么？

通常局限点来说，Redis 也以消息队列的形式存在，作为内嵌的List 存在，满足实时的高并发需求。而通常在一个电商类型的数据处理过程之中，有关商品，热销，推荐排序的队列，通常存放在Redis 之中，期间也包扩Storm 对于Redis 列表的读取和更新。

#### Redis的优缺点是什么？

优点:

1. 性能极高 – Redis 能支持超过 100K+ 每秒的读写频率。
2. 丰富的数据类型 – Redis 支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets数据类型操作。
3. 原子 – Redis 的所有操作都是原子性的，同时Redis 还支持对几个操作全并后的原子性执行。
4. 丰富的特性 – Redis 还支持 publish/subscribe, 通知, key 过期等等特性。

缺点:  
1. 是数据库容量受到物理内存的限制,不能用作海量数据的高性能读写,因此Redis 适合的场景主要局限在较小数据量的高性能操作和运算上。

#### 为什么redis需要把所有数据放到内存中？

Redis为了达到最快的读写速度将数据都读到内存中，并通过异步的方式将数据写入磁盘。所以redis具有快速和数据持久化的特征。如果不将数据放在内存中，磁盘I/O速度为严重影响redis的性能。在内存越来越便宜的今天，redis将会越来越受欢迎。  
如果设置了最大使用的内存，则数据已有记录数达到内存限值后不能继续插入新值。

#### Redis 中list 底层实现有哪几种？有什么区别？

列表对象的编码可以是ziplist或者linkedlist。  
  
ziplist是一种压缩链表，它的好处是更能节省内存空间，因为它所存储的内容都是在连续的内存区域当中的。当列表对象元素不大，每个元素也不大的时候，就采用ziplist存储。但当数据量过大时就ziplist就不是那么好用了。因为为了保证他存储内容在内存中的连续性，插入的复杂度是O(N)，即每次插入都会重新进行realloc。如下图所示，对象结构中ptr所指向的就是一个ziplist。整个ziplist只需要malloc一次，它们在内存中是一块连续的区域。  
  
linkedlist是一种双向链表。它的结构比较简单，节点中存放pre和next两个指针，还有节点相关的信息。当每增加一个node的时候，就需要重新malloc一块内存。

#### Redis五大数据结构，及在项目中的应用

|  |  |  |
| --- | --- | --- |
| 结构 | | 项目中的应用 |
| String | 字符串 | 比如说当一个ip地址访问网站超过了预定的次数，可以禁止访问，则这个预定次数就可以使用String来存储 |
| List | 列表 | 实现最新消息信息排列展示【消息队列】 |
| Set | 集合 | 特殊之处在于set是可以**自动排重**的。比如在微博应用中，每个人的好友存在一个集合（set）中，这样求两个人的共同好友的操作，可能就只需要用求交集命令即可。 |
| Hash | 散列 | 存储用户信息：key(用户ID) + field(属性标签) 操作对应属性数据了，既不需要重复存储数据，也不会带来序列化和并发修改控制的问题。很好的解决了问题 |
| Zset | 有序集合 | 以某个条件为权重，比如按顶的次数排序。  需要精准设定过期时间的应用  使用sorted set的设置过期时间的时间戳，那么就可以简单地通过过期时间排序，定时清除过期数据。 |

#### 怎样解决数据库高并发的问题？

1.在客户端将连接进行池化，同时对客户端读写Redis操作采用内部锁synchronized。  
2.服务器角度，利用setnx变向实现锁机制。执行关键读写时添加内部锁

#### Redis 集群实现？

Redis集群是一个distribute、fault-tolerant的Redis实现，主要设计目标是达到线性可扩展性、可用性、数据一致性。  
  
线性拓展:  
官方推荐最大的节点数量为1000，由于Cluster架构中无Proxy层，Master与Slave之间使用异步replication。  
  
数据一致性:  
客户端容忍一定程度的数据丢失，集群尽可能保存Client write操作的数据，保证数据一致性。  
  
可用性:  
Redis集群通过partition来提供一定程度的可用性，当集群中的一部分节点失效或者无法进行通讯时，集群仍可以继续提供服务。

#### Redis 的并发竞争问题怎么解决？

1. 利用redis自带的incr命令，具体用法: http://doc.redisfans.com/string/incr.html  
2. 可以使用独占锁的方式，类似操作系统的mutex机制。  
3. 使用乐观锁的方式进行解决（成本较低，非阻塞，性能较高）  
4. 利用redis的setnx实现内置的锁。

#### Redis 和MongoDB 的优缺点?

MongoDB 和 Redis 都是 NoSQL，采用结构型数据存储。二者在使用场景中，存在一定的区别，这也主要由于二者在内存映射的处理过程，持久化的处理方法不同。MongoDB 建议集群部署，更多的考虑到集群方案，Redis 更偏重于进程顺序写入， 虽然支持集群，也仅限于主-从模式.  
  
Redis 优点:  
1 读写性能优异  
2 支持数据持久化，支持 AOF 和 RDB 两种持久化方式  
3 支持主从复制，主机会自动将数据同步到从机，可以进 行读写分离。  
4 数据结构丰富:除了支持string类型的 value外还支 持 string、hash、set、sortedset、list 等数据结构。  
缺点:  
1 Redis 不具备自动容错和恢复功能，主机从机的宕机都 会导致前端部分读写请求失败，需要等待机器重启或者手动切换前端的 IP 才能恢复。  
2 主机宕机，宕机前有部分数据未能及时同步到从机，切换IP后还会引入数据不一致的问题，降低了系统的可用性。

Redis 较难支持在线扩容，在集群容量达到上限时在线 扩容会变得很复杂。为避免这一问题，运维人员在系统上线时 必须确保有足够的空间，这对资源造成了很大的浪费.避免这一问题，运维人员在系统上线时 必须确保有足够的空间，这对资源造成了很大的浪费。  
  
mongodb的优缺点：  
优点:弱一致性(最终一致)，更能保证用户的访问速度 文档结构的存储方式，能够更便捷的获取数 频)内置 GridFS，高效存储二进制大对象 (比如照片和视支持复制集、主备、互为主备、自动分片等特性 动态查询 全索引支持,扩展到内部对象和内嵌数组  
缺点:不支持事务MongoDB 占用空间过大，维护工具不够成熟

#### Redis 基本类型、相关方法

Redis支持五种数据类型:string(字符串)、hash(哈希)、list(列表)、set (集合)及zset(sorted set:有序集合)。  
String是Redis最为常用的一种数据类型，String的数据结构为key/value类型，String 可以包含任何数据。常用命令: set,get,decr,incr,mget 等

#### LPOP和BLPOP的区别?

BLPOP 是阻塞式列表的弹出命令。 它是命令 LPOP 的阻塞版本，这是因为当给定列表内没有任何元素可供弹出的时候， 连接将被 BLPOP 命令阻塞。 当给定多个 key 参数时，按参数 key 的先后顺序依次检查各个列表，弹出第一个非空列表的头元素。

#### Redis的有一些包含SCAN关键字的命令是干嘛的? SCAN返回的数据量是固定的吗?

SCAN 命令及其相关的 SSCAN 命令、 HSCAN 命令和 ZSCAN 命令都用于增量地迭代，它们每次执行都只会返回少量元素。  
SCAN返回的数据量不固定。

#### Redis中的Lua有没有使用过? 可以用来做什么? 为什么可以这么用?

使用过，可以用来解决秒杀业务的高并发请求。  
（1）Lua可以将复杂或多步的redis操作，写成脚本一次性提交给redis执行，提升性能  
（2）Lua脚本类似于redis事务有一定原子性，不会被其他命令插队，可以完成一些redis事务操作

#### Redis的Pipeline是用来干什么的?

pipeline（管道）可以一次性发送多条命令并在执行完后一次性将结果返回，pipeline通过减少客户端与redis的通信次数来实现降低往返延时时间。

#### Redis中List, HashTable都⽤用到了ZipList, 为什么会选择它?

redis初始创建hash表,有序集合，链表时, 存储结构采用一种ziplist的存储结构, 这种结构内存排列更紧密, 能提高访存性能。

#### Redis 的事务？

redis中事务的实现特征  
  
1. 在事务中的所有命令都将会被串行化的顺序执行，在事务执行期间，redis不会再为其他客户端的请求提供热河服务，从未保证了事务中的所有命令被原子的执行

2. 和关系型数据库中的事务相比，在redis事务中如果有某一条命令执行失败，其后的命令仍然会被继续执行

3. 我们可以通过multi命令开启一个事务，在该语句知乎执行的命令都将被视为事务之内的操作。最后可以通过执行exec/discard命令来提交/回滚该事务内的所有操作

4. 在事务开启之前，如果客户端与服务器之间出现通讯故障并导致网络断开，其后所有待执行的语句都将不会被服务器执行。然而如果网络中断事件是发生在客户端执行exec命令之后，那么该事务中的所欲命令都会被服务器执行

5. 当使用Append-Only模式时，redis会通过调用系统函数write将该事务内的所欲写操作在本次调用中全部写入磁盘。然而如果在写入的过程中出现系统崩溃，如电源故障的宕机，那么此时也许只有部分数据被写入到磁盘，而另一部分数据已经丢失。redis服务器会在重启时执行一系列必要的检测，一旦发现类似问题，就会立即退出并给出相应的错误提示。此时，我们就要充分利用redis工具包中提供的redis-check-aof工具。该工具可以帮助我们定位到数据不一致的错误，并将已经写入的部分数据进行回滚。修复之后我们就可以再次重启服务器

#### Redis 的使用场景有哪些？

-1.取最新 N 个数据的操作  
-2.排行榜应用,取 TOP N 操作  
-3.需要精准设定过期时间的应用  
-4.计数器应用  
-5.uniq 操作,获取某段时间所有数据排重值  
-6.Pub/Sub 构建实时消息系统  
-7.构建队列系统  
-8.缓存

#### Redis 默认端口，默认过期时间，Value 最多可以容纳的数据长度？

1. 6379  
2. 默认不过期  
3. 在Redis中字符串类型的Value最多可以容纳的数据长度是512M

#### redis是否可以做持久化？

RDB持久化   
　　-每隔一段时间对redis进行一次持久化（基于时间点快照的方式，复用方式进行数据持久化）  
　　-效率较高，数据不完整，安全性不高  
  
AOF持久化  
　　-把所有命令保存起来，如果想到重新生成到redis，那么就要把命令重新执行一次。  
　　-效率相对较低，安全性较高

RDB与AOF混合持久化：

- Redis 4.0新功能，在每次重写AOF文件时，检查是否有RDB文件，并将RDB文件信息写入AOF文件头，这样在恢复时，首先恢复RDB文件，剩余的信息由AOF文件负责恢复，可以同时兼顾到RDB文件的恢复效率及AOF文件的恢复完整度

#### redis的过期策略

voltile-lru： 从已设置过期时间的数据集（server.db[i].expires）中挑选最近频率最少数据淘汰  
volatile-ttl： 从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰  
volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰  
  
allkeys-lru： 从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰  
allkeys-random： 从数据集（server.db[i].dict）中任意选择数据淘汰  
no-enviction（驱逐）：禁止驱逐数据

#### redis的分布式锁实现

单台redis时，可以使用watch  
对多台redis进行操作时，加锁并设置超时时间，保证在此期间只有你一个人对redis操作，  
  
----流程  
- 写值｛key:**"sfdfff"**｝加锁并设置超时时间  
- 超过一半的redis实例设置成功，就表示加锁完成。  
- 解锁 执行lua脚本，用key检测每个redis中是否有次key，有则删除  
  
- 使用：安装redlock-py   
 **from** redlock **import** Redlock  
　　Redlock算法  
 dlm = Redlock(  
 [  
 {**"host"**: **"localhost"**, **"port"**: 6379, **"db"**: 0},  
 {**"host"**: **"localhost"**, **"port"**: 6379, **"db"**: 0},  
 {**"host"**: **"localhost"**, **"port"**: 6379, **"db"**: 0},  
 ]  
 )  
  
 *# 加锁，acquire* my\_lock = dlm.lock(**"my\_resource\_name"**,10000)  
 **if** my\_lock:  
 *# J进行操作  
 # 解锁,release* dlm.unlock(my\_lock)  
 **else**:  
 print(**'获取锁失败'**)

#### 发布者与订阅者、消息队列

发布订阅：只要发布者发布任务则所有订阅者都会接受到此任务。  
  
消息队列：队列中放一个任务，则只有一个进程取任务  
  
发布订阅和简单的消息队列区别在于，发布订阅会将消息发送给所有的订阅者，而消息队列中的数据被消费一次便消失。  
  
所以，Redis实现发布和订阅时，会为每一个订阅者创建一个队列，而发布者发布消息时，会将消息放置在所有相关队列中。

#### Redis 有多少个库？

当redis 服务器初始化时，会预先分配 16 个数据库（该数量可以通过配置文件配置），所有数据库保存到结构 redisServer 的一个成员 redisServer.db 数组中。当我们选择数据库 select number 时，程序直接通过 redisServer.db[number] 来切换数据库。有时候当程序需要知道自己是在哪个数据库时，直接读取 redisDb.id 即可。

#### Redis缓存击穿、缓存雪崩、缓存重建

缓存击穿：  
 当一个连接访问缓存数据库中不存在的数据时，会直接通过缓存数据库到后端数据库中查找数据，但如果有大量连接在查询一个不存在的数据，就会有大量连接直接访问到后端数据库，给后端服务器造成巨大压力，可能导致后台服务器崩溃的情况。 解决方案：当查询一个不存在的缓存数据时，访问数据库，如果发现后端数据库也不存在这个文件，将这个查询的key在缓存中保存为None。下次再有连接请求这个key时直接从缓存中返回。  
  
缓存雪崩：  
 缓存数据一般会设置过期时间，当所有的缓存数据几乎在同一时间过期时，会有大量数据直接访问到后端数据库，导致后端数据库瞬间承受访问压力过大，可能导致后端数据库崩溃。并且当数据库重启后，仍会有大量连接访问到数据库，可能导致再次崩溃。 解决方案：采用缓存重建或缓存预热。  
  
缓存重建：  
 Redis支持持久化操作，比如缓存之类不重要的数据可以使用RDB持久化，可以恢复大部分缓存内容到缓存数据库汇总，减少直接访问后端数据库的次数。  
  
缓存预热：  
 系统上线时手动加载一部分数据到缓存数据库，同样也是为了减少缓存数据库中没有数据，导致连接直接穿透到后端数据库的情况。

#### 秒杀场景如何解决？

使用Redis队列，设定队列最大数量，有新的订单直接加入队列，当超过最大数量时直接返回产品暂时售空，请稍后尝试，期间依次处理Redis队列中的订单请求，如果订单均成功，提示剩余用户秒杀结束，这样也可以一定程度的避免超卖情况。具体介绍看链接。  
  
详细链接： <https://blog.csdn.net/zhanjianshinian/article/details/53342730>

## MongoDB

#### MongoDB是什么？

MongoDB 是一个面向文档的数据库系统。使用C++编写，不支持SQL，但有自己功能强大的查询语法。  
  
MongoDB 使用BSON 作为数据存储和传输的格式。BSON 是一种类似JSON 的二进制序列化文档，支持嵌套对象和数组。MongoDB 很像MySQL，document 对应MySQL 的row，collection 对应MySQL 的table

应用场景：  
a) 网站数据：mongo 非常适合实时的插入，更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性。  
b) 缓存：由于性能很高，mongo 也适合作为信息基础设施的缓存层。在系统重启之后，由mongo 搭建的持久化缓存可以避免下层的数据源过载。  
c) 大尺寸、低价值的数据：使用传统的关系数据库存储一些数据时可能会比较贵，在此之前，很多程序员往往会选择传统的文件进行存储。  
d) 高伸缩性的场景：mongo 非常适合由数十或者数百台服务器组成的数据库。  
e) 用于对象及JSON 数据的存储：mongo 的BSON 数据格式非常适合文档格式化的存储及查询。  
f) 重要数据：mysql，一般数据：mongodb，临时数据：memcache  
g) 对于关系数据表而言，mongodb 是提供了一个更快速的视图view；而对于PHP 程序而言，mongodb 可以作为一个持久化的数组来使用，并且这个持久化的数组还可以支持排序、条件、限制等功能。  
h) 将mongodb 代替mysql 的部分功能，主要一个思考点就是：把mongodb 当作mysql的一个view（视图），view 是将表数据整合成业务数据的关键。比如说对原始数据进行报表，那么就要先把原始数据统计后生成view，在对view 进行查询和报表。

不适合的场景：  
a) 高度事物性的系统：例如银行或会计系统。传统的关系型数据库目前还是更适用于需要大量原子性复杂事务的应用程序。  
b) 传统的商业智能应用：针对特定问题的BI 数据库会对产生高度优化的查询方式。对于此类应用，数据仓库可能是更合适的选择。  
c) 需要SQL 的问题  
d) 重要数据，关系数据

优点:

1）弱一致性（最终一致），更能保证用户的访问速度  
2）文档结构的存储方式，能够更便捷的获取数  
3）内置GridFS，高效存储二进制大对象(比如照片和视频)  
4）支持复制集、主备、互为主备、自动分片等特性  
5）动态查询  
6）全索引支持,扩展到内部对象和内嵌数组

缺点:

1）不支持事务  
2）MongoDB 占用空间过大,维护工具不够成熟

#### 数据库的一些基本操作命令（列举一些常用命令即可）？

1、Help查看命令提示  
help  
db.help();  
db.yourColl.help();  
db.youColl.find().help();  
rs.help();  
  
2、切换/创建数据库  
use yourDB;   
当创建一个集合(table)的时候会自动创建当前数据库  
  
3、查询所有数据库  
show dbs;  
  
4、删除当前使用数据库  
db.dropDatabase();

#### Python 中调用mongo 数据库的包叫什么？

Pymongo

#### MongoDB 成为优秀的NoSQL 数据库的原因是什么?

以下特点使得MongoDB成为最好的NoSQL数据库：  
1. 面向文件的,丰富的数据模型  
2. 容易扩展  
3. 不牺牲速度  
4. 便捷的管理  
5. 丰富的查询语言

#### 分析器在MongoDB 中的作用是什么?

MongoDB中包括了一个可以显示数据库中每个操作性能特点的数据库分析器。通过这个分析器你可以找到比预期慢的查询(或写操作);利用这一信息，比如，可以确定是否需要添加索引。

#### 怎么查看MongoDB 正在使用的链接?

db.\_adminCommand(**"connPoolStats"**);

#### MySQL 与MongoDB 本质之间最基本的差别是什么

MySQL和MongoDB两者都是免费开源的数据库。MySQL和MongoDB有许多基本差别包括数据的表示(data representation)，查询，关系，事务，schema的设计和定义，标准化(normalization)，速度和性能。  
  
通过比较MySQL和MongoDB，实际上我们是在比较关系型和非关系型数据库，即数据存储结构不同。

#### 使用MongoDB 的优点

优点  
1. 面向文档存储(类JSON数据模式简单而强大)  
2. 动态查询  
3. 全索引支持,扩展到内部对象和内嵌数组  
4. 查询记录分析  
5. 快速,就地更新  
6. 高效存储二进制大对象 (比如照片和视频)  
7. 复制和故障切换支持  
8. Auto- Sharding自动分片支持云级扩展性  
9. MapReduce 支持复杂聚合  
  
缺点  
1. 不支持事务（进行开发时需要注意，哪些功能需要使用数据库提供的事务支持）  
2. MongoDB占用空间过大 （不过这个确定对于目前快速下跌的硬盘价格来说，也不算什么缺点了）  
3. MongoDB没有如MySQL那样成熟的维护工具，这对于开发和IT运营都是个值得注意的地方  
4. 在32位系统上，不支持大于2.5G的数据(很多操作系统都已经抛弃了32位版本，所以这个也算不上什么缺点了，3.4版本已经放弃支持32 位 x86平台)

#### MongoDB支持存储过程吗？如果支持的话，怎么用？

MongoDB支持存储过程，它是javascript写的，保存在db.system.js表中。

# 爬虫

## 常用库与模块

#### User-Agent如何判断浏览器类型，火狐怎么显示，Chrome怎么显示

一般为Mozilla开头 + 设备识别号 + 浏览器核心版本号  
切记！ 所有User-Agent 均为Mozilla开头；Mozilla是Netscape的吉祥物，也是Netscape Navigator浏览器使用的内部开发代号。由于Netscape早期的影响力，直到今天，所有浏览器包括IE，向Web服务器报告自己的浏览器标识的 时候，都以 “Mozilla”开头，表明自己是Mozilla兼容的。  
  
chrome  
Mozilla/5.0 (Windows NT 5.2) AppleWebKit/534.30 (KHTML, like Gecko) Chrome/12.0.742.122 Safari/534.30  
  
Firefox  
Mozilla/5.0 (Windows NT 5.1; rv:5.0) Gecko/20100101 Firefox/5.0

#### 列举您使用过的Python 网络爬虫所用到的网络数据包?

requests、urllib、urllib2

#### 列举您使用过的Python 网络爬虫所用到的解析数据包?

BeautifulSoup、Xpath、lxml

#### 爬取数据后使用哪个数据库存储数据的，为什么？

MongoDB 是使用比较多的数据库，这里以MongoDB 为例，大家需要结合自己真实开发环境

回答。

原因：

1）与关系型数据库相比，MongoDB 的优点如下。

①弱一致性（最终一致），更能保证用户的访问速度

举例来说，在传统的关系型数据库中，一个COUNT 类型的操作会锁定数据集，这样可以保证得到“当前”情况下的较精确值。这在某些情况下，例如通过ATM 查看账户信息的时候很重要，但对于Wordnik 来说，数据是不断更新和增长的，这种“较精确”的保证几乎没有任何意义，反而会产生很大的延迟。他们需要的是一个“大约”的数字以及更快的处理速度。

但某些情况下MongoDB 会锁住数据库。如果此时正有数百个请求，则它们会堆积起来，造成许多问题。我们使用了下面的优化方式来避免锁定。

每次更新前，我们会先查询记录。查询操作会将对象放入内存，于是更新则会尽可能的迅速。在主/从部署方案中，从节点可以使用“-pretouch”参数运行，这也可以得到相同的效果。

使用多个mongod 进程。我们根据访问模式将数据库拆分成多个进程。

②文档结构的存储方式，能够更便捷的获取数据。

对于一个层级式的数据结构来说，如果要将这样的数据使用扁平式的，表状的结构来保存数据，这无论是在查询还是获取数据时都十分困难。

③内置GridFS，支持大容量的存储。

GridFS 是一个出色的分布式文件系统，可以支持海量的数据存储。内置了GridFS 了MongoDB，能够满足对大数据集的快速范围查询。

④内置Sharding。

提供基于Range 的Auto Sharding 机制：一个collection 可按照记录的范围，分成若干个段，切分到不同的Shard 上。Shards 可以和复制结合，配合Replica sets 能够实现Sharding+fail-over，不同的Shard 之间可以负载均衡。查询是对客户端是透明的。客户端执行查询，统计，MapReduce等操作，这些会被MongoDB 自动路由到后端的数据节点。这让我们关注于自己的业务，适当的时候可以无痛的升级。MongoDB 的Sharding 设计能力较大可支持约20 petabytes，足以支撑一般

应用。

这可以保证MongoDB 运行在便宜的PC 服务器集群上。PC 集群扩充起来非常方便并且成本很低，避免了“sharding”操作的复杂性和成本。

⑤第三方支持丰富。(这是与其他的NoSQL 相比，MongoDB 也具有的优势)

现在网络上的很多NoSQL 开源数据库完全属于社区型的，没有官方支持，给使用者带来了很大的风险。而开源文档数据库MongoDB 背后有商业公司10gen 为其提供供商业培训和支持。

而且MongoDB 社区非常活跃，很多开发框架都迅速提供了对MongDB 的支持。不少知名大公司和网站也在生产环境中使用MongoDB，越来越多的创新型企业转而使用MongoDB 作为和Django，RoR 来搭配的技术方案。

⑥性能优越

在使用场合下，千万级别的文档对象，近10G 的数据，对有索引的ID 的查询不会比mysql 慢，而对非索引字段的查询，则是全面胜出。mysql 实际无法胜任大数据量下任意字段的查询，而mongodb 的查询性能实在让我惊讶。写入性能同样很令人满意，同样写入百万级别的数据，mongodb 比我以前试用过的couchdb 要快得多，基本10 分钟以下可以解决。补上一句，观察过程中mongodb 都远算不上是CPU 杀手。

2)Mongodb 与redis 相比较

①mongodb 文件存储是BSON 格式类似JSON，或自定义的二进制格式。

mongodb 与redis 性能都很依赖内存的大小，mongodb 有丰富的数据表达、索引；最类似于关系数据库，支持丰富的查询语言，redis 数据丰富，较少的IO ，这方面mongodb 优势明显。

②mongodb 不支持事物，靠客户端自身保证，redis 支持事物，比较弱，仅能保证事物中的操作按顺序执行，这方面redis 优于mongodb。

③mongodb 对海量数据的访问效率提升，redis 较小数据量的性能及运算,这方面mongodb性能优于redis. monbgodb 有mapredurce 功能，提供数据分析，redis 没有，这方面mongodb优于redis 。

#### 你用过的爬虫框架或者模块有哪些？谈谈他们的区别或者优缺点？

Python自带：urllib，urllib2  
第三方：requests  
框架：Scrapy  
  
  
urllib2 ：urllib2.urlopen可以接受一个Request对象或者url，（在接受Request对象时候，并以此可以来设置一个URL 的headers），urllib.urlopen只接收一个url  
urllib : 有urlencode,urllib2没有，因此总是urllib，urllib2常会一起使用的原因  
scrapy : 是封装起来的框架，他包含了下载器，解析器，日志及异常处理，基于多线程， twisted的方式处理，对于固定单个网站的爬取开发，有优势，但是对于多网站爬取 100个网站，并发及分布式处理方面，不够灵活，不便调整与括展。  
request : 是一个HTTP库， 它只是用来，进行请求，对于HTTP请求，他是一个强大的库，下载，解析全部自己处理，灵活性更高，高并发与分布式部署也非常灵活，对于功能可以更好实现.  
  
  
Scrapy优缺点：  
优点：  
1. scrapy 是异步的  
2. 采取可读性更强的xpath代替正则  
3. 强大的统计和log系统  
4. 同时在不同的url上爬行  
5. 支持shell方式，方便独立调试  
6. 写middleware,方便写一些统一的过滤器  
7. 通过管道的方式存入数据库  
  
缺点：基于python的爬虫框架，扩展性比较差  
1. 基于twisted框架，运行中的exception是不会干掉reactor，并且异步框架出错后是不会停掉其他任务的，数据出错后难以察觉。

#### 写爬虫是用多进程好？还是多线程好？ 为什么？

IO密集型代码(文件处理、网络爬虫等)，多线程能够有效提升效率(单线程下有IO操作会进行IO等待，造成不必要的时间浪费，而开启多线程能在线程A等待时，自动切换到线程B，可以不浪费CPU的资源，从而能提升程序执行效率)。在实际的数据采集过程中，既考虑网速和响应的问题，也需要考虑自身机器的硬件情况，来设置多进程或多线程

#### 常见的反爬虫和应对方法？

1. 通过headers反爬虫  
2. 基于用户行为的发爬虫：  
 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  
3. 动态网页反爬虫  
 例如：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  
4. 对部分数据进行加密处理的  
 例如：我们要抓的数据部分能够抓到，另外的部分加密处理了，是乱码

应对策略：  
 1.对于基本网页的抓取可以自定义headers,添加headers的数据，代理来解决  
 2. 有些网站的数据抓取必须进行模拟登陆才能抓取到完整的数据，所以要进行模拟登陆。  
 3. 对于限制抓取频率的，可以设置抓取的频率降低一些，  
 4. 对于限制ip抓取的可以使用多个代理ip进行抓取，轮询使用代理  
 5. 针对动态网页的可以使用selenium+phantomjs进行抓取，但是比较慢，所以也可以使用查找接口的方式进行抓取。  
 6. 对部分数据进行加密的，可以使用selenium进行截图，饭后使用python自带的 pytesseract库进行识别，但是比较慢最直接的方法是找到加密的方法进行逆向推理。

#### 验证码的解决?

1.输入式验证码  
解决思路：这种是最简单的一种，只要识别出里面的内容，然后填入到输入框中即可。这种识别技术叫OCR，这里我们推荐使用Python的第三方库，tesserocr。对于没有什么背影影响的验证码如图

2.滑动式验证码  
解决思路：对于这种验证码就比较复杂一点，但也是有相应的办法。我们直接想到的就是模拟人去拖动验证码的行为，点击按钮，然后看到了缺口的位置，最后把拼图拖到缺口位置处完成验证。

3.点击式的图文验证  
图文验证：通过文字提醒用户点击图中相同字的位置进行验证。  
图标选择： 给出一组图片，按要求点击其中一张或者多张。借用万物识别的难度阻挡机器。  
这两种原理相似，只不过是一个是给出文字，点击图片中的文字，一个是给出图片，点出内容相同的图片。  
这两种没有特别好的方法，只能借助第三方识别接口来识别出相同的内容，推荐一个超级鹰，把验证码发过去，会返回相应的点击坐标。

#### 写一个邮箱地址的正则表达式？

re.match(**r'^[0-9a-zA-Z\_]{0,19}@[0-9a-zA-Z]{1,13}\.[com,cn,net]{1,3}$'**,text)

#### 编写过哪些爬虫中间件？

下载器中间件(Downloader middlewares)  
下载器中间件是在引擎及下载器之间的特定钩子(specific hook)，处理Downloader传递给引擎的response（也包括引擎传递给下载器的Request）。 其提供了一个简便的机制，通过插入自定义代码来扩展Scrapy功能。  
  
一句话总结就是：处理下载请求部分  
  
Spider中间件(Spider middlewares)  
Spider中间件是在引擎及Spider之间的特定钩子(specific hook)，处理spider的输入(response)和输出(items及requests)。 其提供了一个简便的机制，通过插入自定义代码来扩展Scrapy功能。  
  
一句话总结就是：处理解析部分

#### “极验”滑动验证码如何破解？

*# 步骤一:点击按钮，弹出没有缺口的图片  
# 步骤二：获取步骤一的图片  
# 步骤三：点击滑动按钮，弹出带缺口的图片  
# 步骤四：获取带缺口的图片  
# 步骤五：对比两张图片的所有RBG像素点，得到不一样像素点的x值，即要移动的距离  
# 步骤六：模拟人的行为习惯（先匀加速拖动后匀减速拖动），把需要拖动的总距离分成一段一段小的轨迹  
# 步骤七：按照轨迹拖动，完全验证  
# 步骤八：完成登录*详细链接： <https://blog.csdn.net/lmw1239225096/article/details/79099238>

#### 爬的那些内容数据量有多大，多久爬一次，爬下来的数据是怎么存储？

1. 内容数据量一般在几K到几十K之间（关闭图片显示的情况），如果爬取图片，大概在几百K以上  
2. 根据网站限制不同设置的抓取频率不同，一般3-5秒发送一次抓取请求，需要更新的内容，每天重新抓取一次  
3. 有规律的数据比如房产数据，招聘数据，可以直接存入mysql， 但是全网爬虫，比如企业信息等，需要存入mongodb

京东整站的数据大约在1 亿左右，爬下来的数据存入数据库，mysql 数据库中如果有重复的url 建议去重存入数据库，可以考虑引用外键。评分，评论如果做增量，Redis 中url 去重，评分和评论建议建立一张新表用id 做关联。

多久爬一次这个问题要根据公司的要求去处理，不一定是每天都爬。

Mongo 建立唯一索引键（id）可以做数据重复前提是数据量不大2 台电脑几百万的情况数据库需要做分片（数据库要设计合理）。

例：租房的网站数据量每天大概是几十万条，每周固定爬取。

#### cookie 过期的处理问题？

使用selenium。  
  
步骤1、 采用selenium自动登录获取cookie，保存到文件;  
步骤2、 读取cookie，比较cookie的有效期，若过期则再次执行步骤1；  
步骤3、 在请求其他网页时，填入cookie，实现登录状态的保持。

#### HTTPS 有什么优点和缺点?

优点  
SEO方面：谷歌曾在2014年8月份调整搜索引擎算法，并称“比起同等HTTP网站，采用HTTPS加密的网站在搜索结果中的排名将会更高”。  
安全性：尽管HTTPS并非绝对安全，掌握根证书的机构、掌握加密算法的组织同样可以进行中间人形式的攻击，但HTTPS仍是现行架构下最安全的解决方案，主要有以下几个好处：  
1. 使用HTTPS协议可认证用户和服务器，确保数据发送到正确的客户机和服务器;  
2. HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，要比http协议安全，可防止数据在传输过程中不被窃取、改变，确保数据的完整性;  
3. HTTPS是现行架构下最安全的解决方案，虽然不是绝对安全，但它大幅增加了中间人攻击的成本。  
  
  
缺点  
SEO方面：据ACM CoNEXT数据显示，使用HTTPS协议会使页面的加载时间延长近50%，增加10%到20%的耗电，此外，HTTPS协议还会影响缓存，增加数据开销和功耗，甚至已有安全措施也会受到影响也会因此而受到影响。  
经济方面：SSL证书需要钱，功能越强大的证书费用越高，个人网站、小网站没有必要一般不会用;  
HTTPS连接缓存不如HTTP高效，大流量网站如非必要也不会采用，流量成本太高;  
HTTPS连接服务器端资源占用高很多，支持访客稍多的网站需要投入更大的成本，如果全部采用HTTPS，基于大部分计算资源闲置的假设的VPS的平均成本会上去;  
HTTPS协议握手阶段比较费时，对网站的相应速度有负面影响，如非必要，没有理由牺牲用户体验。

#### HTTPS 是如何实现安全传输数据的?

1. 客户端（通常是浏览器）先向服务器发出加密通信的请求  
2. 服务器收到请求,然后响应  
3. 客户端收到证书之后会首先会进行验证  
4. 服务器收到使用公钥加密的内容，在服务器端使用私钥解密之后获得随机数pre-master secret，然后根据radom1、radom2、pre-master secret通过一定的算法得出session Key和MAC算法秘钥，作为后面交互过程中使用对称秘钥。同时客户端也会使用radom1、radom2、pre-master secret，和同样的算法生成session Key和MAC算法的秘钥。  
5. 然后再后续的交互中就使用session Key和MAC算法的秘钥对传输的内容进行加密和解密。  
  
详细链接： https://www.jianshu.com/p/b0b6b88fe9fe

#### TTL，MSL，RTT？

MSL：  
Maximum Segment Lifetime缩写，译为“报文最大生存时间”。它指的是任何报文在网络上存在的最长时间，超过这个时间的报文将会被丢弃。标准规范中规定MSL为2分钟，实际应用中常用的是30s，1min和2min等。  
  
  
RTT：  
Round-Trip Time缩写，简单理解的话，RTT指的是客户端到服务器往返所花费的时间。其实RTT的定义是一个很小的分组(这里的很小也就是说对于发送方来说“开始发送”和“发送完”是同一个时刻。换句话说这个分组是没有长度的，传输时延可以忽略不计)，从客户端发送到接收端再返回客户端的时间。  
  
  
TTL：  
Time To Time缩写，译为“生存时间”。这个生存时间是由源主机设置（IP头里有一个TTL域）初始值但不是存的具体时间，而是存储了一个IP数据报可以经过最大路由数，每经过一个处理它的路由器此值就减1，当此值为0则数据报将被丢弃，同时发送ICMP报文通知源主机。

#### 谈一谈你对Selenium 和PhantomJS 了解?

Selenium是一个Web的自动化测试工具，最初是为了网站自动化测试开发的，Selenium测试直接运行在浏览器中，就像真正的用户在操作一样。支持的浏览器包括IE（7, 8, 9, 10, 11），Mozilla Firefox，Safari，Google Chrome，Opera等。也包括PhantomJS。  
  
Selenium可以根据我们的指令，让浏览器自动加载页面，获取需要的数据，甚至是页面截屏。  
  
  
PhantomJS 是一个基于Webkit的“无界面”(headless)浏览器，它会把网站加载到内存并执行页面上的 JavaScript，因为不会展示图形界面，所以运行起来比完整的浏览器要高效。  
  
如果我们把 Selenium 和 PhantomJS 结合在一起，就可以运行一个非常强大的网络爬虫了，这个爬虫可以处理 JavaScrip、Cookie、headers，以及任何我们真实用户需要做的事情。

#### 代理IP 里的“透明”“匿名”“高匿”分别是指？

透明代理的意思是客户端根本不需要知道有代理服务器的存在，但是它传送的仍然是真实的IP。你要想隐藏的话，不要用这个。

普通匿名代理能隐藏客户机的真实IP，但会改变我们的请求信息，服务器端有可能会认为我们使用了代理。不过使用此种代理时，虽然被访问的网站不能知道你的ip地址，但仍然可以知道你在使用代理，当然某些能够侦测ip的网页仍然可以查到你的ip。

高匿名代理不改变客户机的请求，这样在服务器看来就像有个真正的客户浏览器在访问它，这时客户的真实IP是隐藏的，服务器端不会认为我们使用了代理。

#### requests 返回的content 和text 的区别？

text 返回的是unicode 型的数据，一般是在网页的header中定义的编码形式。  
content返回的是bytes，二级制型的数据。  
  
也就是说你如果想要提取文本就用text  
但是如果你想要提取图片、文件，就要用到content

#### robots 协议

Robots协议（也称为爬虫协议、爬虫规则、机器人协议等）也就是robots.txt，网站通过robots协议告诉搜索引擎哪些页面可以抓取，哪些页面不能抓取。  
Robots协议是网站国际互联网界通行的道德规范，其目的是保护网站数据和敏感信息、确保用户个人信息和隐私不被侵犯。因其不是命令，故需要搜索引擎自觉遵守。

#### 为什么requests 请求需要带上header？

模拟浏览器，欺骗服务器，获取和浏览器一致的内容

#### dumps、loads 与dump、load 的区别？

json.dumps : dict转成str 一个是将字典转换为字符串   
json.loads: str转成dict 一个是将字符串转换为字典   
json.dump 是将python数据保存成json   
json.load 是读取json数据

#### requests 使用小技巧

1：保持请求之间的Cookies，我们可以这样做。  
**import** requests  
session = requests.Session()  
session.get(login\_url) *# 可以保持登录态*2：请求时，会加上headers，一般我们会写成这样  
session.get(url, params, headers=headers)  
唯一不便的是之后的代码每次都需要这么写，代码显得臃肿，所以我们可以这样:  
  
*#在构造函数中，这样设置是全局的。  
# 设置请求头*s = requests.Session()  
s.headers = {**'balabala'**}  
  
*# 移除服务器验证*s.verify = **False***# 设置代理*s.proxies={**'aa'**}  
  
*# 如果后续headers有改变，再次赋值就可以了。*s.get(url, params, headers=new\_headers)  
  
3：默认requests请求失败后不会重试，但是我们跑case时难免遇到一些网络,服务重启，外部原因导致case失败，我们可以在Session实例上附加HTTPAdapaters 参数，增加失败重试次数。  
request\_retry = requests.adapatrs.HTTPAdapaters(max\_retries=3）  
  
session.mount(**'https://'**,request\_retry)   
session.mount(**'http://'**,request\_retry)  
*# 这样，之后的请求，若失败，重试3次。*4：重定向   
*# 网络请求中可能会遇到重定向，我们需要一次处理一个请求，可以把重定向禁止。*session.post(url,data,allow\_redirects=**False**)  
  
5： post请求提交json格式数据时(请求头为:{“Content-Type”: “application/json”})，一般先要把python对象转换成json对象。可能很多时候是这样写：  
session.post(url, data=json.dumps(data))  
其实post有一个默认参数json，可以直接简写成：  
session.post(url, json=data)

#### IP 存放在哪里？怎么维护IP？对于封了多个ip 的，怎么判定IP 没被封？

1. IP存放于代理IP池文件  
2. 不能过度使用一个IP地址，可以先测试网站封IP的频率，在临近封禁时更换IP地址  
3. 使用requests访问网站首页，返回status\_code 200就是没有被封禁

#### 怎么获取加密的数据？

1. 手动获取网站JS加密方式和流程  
2. 编写代码将数据进行加解密后处理

#### 假如每天爬取量在5、6 万条数据，一般开几个线程，每个线程ip 需要加锁限定吗？

1. 一般开5-10个线程，具体根据网速和爬取内容决定  
2. 线程IP不需要加锁限定，这种量级的爬取肯定会被认为是机器操作，需要定期更换IP地址

#### 怎么监控爬虫的状态

在 scrapy 运行的过程中，新建一个 middleware ，当有 request 经过 middleware 的时候，将当前 crawler 的状态保存到 redis ，这样在 redis 中就有实时的爬虫状态信息了，前端一直 ajax 获取 redis 里面的信息放到前端渲染出来就行了。

## Scrapy

#### 描述下scrapy 框架运行的机制？

从start\_urls里获取第一批url并发送请求，请求由引擎交给调度器入请求队列，获取完毕后，调度器将请求队列里的请求交给下载器去获取请求对应的响应资源，并将响应交给自己编写的解析方法做提取处理：

1. 如果提取出需要的数据，则交给管道文件处理；

2. 如果提取出url，则继续执行之前的步骤（发送url请求，并由引擎将请求交给调度器入队列...)，直到请求队列里没有请求，程序结束。

#### 谈谈你对Scrapy 的理解？

scrapy 是一个为了爬取网站数据，提取结构性数据而编写的应用框架，我们只需要实现少量代码，就能够快速的抓取到数据内容。Scrapy 使用了 Twisted异步网络框架来处理网络通讯，可以加快我们的下载速度，不用自己去实现异步框架，并且包含了各种中间件接口，可以灵活的完成各种需求。

scrapy 框架的工作流程：

1.首先Spiders（爬虫）将需要发送请求的url(requests)经ScrapyEngine（引擎）交给Scheduler（调度器）。

2.Scheduler（排序，入队）处理后，经ScrapyEngine，DownloaderMiddlewares(可选，主要有User\_Agent， Proxy 代理)交给Downloader。

3.Downloader 向互联网发送请求，并接收下载响应（response）。将响应（response）经ScrapyEngine，SpiderMiddlewares(可选)交给Spiders。

4.Spiders 处理response，提取数据并将数据经ScrapyEngine 交给ItemPipeline 保存（可以是本

地，可以是数据库）。提取url 重新经ScrapyEngine 交给Scheduler 进行下一个循环。直到无Url

请求程序停止结束。

#### 怎么样让Scrapy框架发送一个post 请求（具体写出来）

**class** mySpider(scrapy.Spider):  
 *# start\_urls = ["http://www.example.com/"]* **def** start\_requests(self):  
 url = **'http://www.renren.com/PLogin.do'** *# FormRequest 是Scrapy发送POST请求的方法* **yield** scrapy.FormRequest(  
 url=url,  
 formdata={**"email"**: **"xxx"**, **"password"**: **"xxxxx"**},  
 callback=self.parse\_page  
 )  
  
 **def** parse\_page(self, response):  
 *# do something*

#### 怎么判断网站是否更新？

将网页中需要爬取的所有数据直接进行哈希计算，因为哈希在计算时，相同的值输入必定相同，所以直接判断输出值，就可以判断出网页是否进行了更新

使用MD5 数字签名：

每次下载网页时，把服务器返回的数据流ResponseStream 先放在内存缓冲区，然后对ResponseStream 生成MD5 数字签名S1，下次下载同样生成签名S2，比较S2 和S1，如果相同，则页面没有更新，否则网页就有更新。

#### 图片、视频爬取怎么绕过防盗连接，或者说怎么获取正确的链接地址？

一般盗链判断是对访问资源页面的来源地址进行判断，即判断是否为从本站访问  
然而一般网站都是针对Header中是否有Referer请求头来判断  
所以我们在访问外链地址时，可以先获取其页面地址，然后修改请求中的referer信息，以解决盗链问题

#### 爬虫向数据库存数据开始和结束都会发一条消息，是scrapy 哪个模块实现的？

Scrapy的信号模块，比如spider\_opened,spider\_closed,item\_dropped等等。我们可以利用这些信号来做一些初始化的工作和数据收集统计。比如spider\_opened信号就方便了我们在爬虫启动的时候，创建到数据库的连接等，item\_dropped的信号就方便我们去收集在爬虫工作的期间，收集被丢弃的item的数量等等。

#### 怎么设置深度爬取?

通过在settings.py中设置DEPTH\_LIMIT的值可以限制爬取深度，这个深度是与start\_urls中定义url的相对值。也就是相对url的深度。例如定义url为：http://www.domz.com/game/,DEPTH\_LIMIT=1那么限制爬取的只能是此url下一级的网页。深度大于设置值的将被ignore。

#### 爬取下来的数据如何去重，说一下具体的算法依据

1.通过 MD5 生成电子指纹来判断页面是否改变  
2.nutch 去重。nutch 中 digest 是对采集的每一个网页内容的 32 位哈希值，如果两个网页内容完全一样，它们的 digest 值肯定会一样。

3. 使用Scrapy-Redis + BloomFilter 去重，当数据量达到亿（甚至十亿、百亿）数量级时，内存有限，必须用“位”来去重，才能够满足需求。Bloomfilter就是将去重对象映射到几个内存“位”，通过几个位的 0/1值来判断一个对象是否已经存在。

## Scrapy-Redis

#### scrapy 和scrapy-redis 有什么区别？为什么选择redis 数据库？

1) scrapy是一个Python爬虫框架，爬取效率极高，具有高度定制性，但是不支持分布式。而scrapy-redis一套基于redis数据库、运行在scrapy框架之上的组件，可以让scrapy支持分布式策略，Slaver端共享Master端redis数据库里的item队列、请求队列和请求指纹集合。  
  
2) 为什么选择redis数据库，因为redis支持主从同步，而且数据都是缓存在内存中的，所以基于redis的分布式爬虫，对请求和数据的高频读取效率非常高。

#### 分布式爬虫主要解决什么问题？

1) ip  
2) 带宽  
3）cpu  
4）io

#### 什么是分布式存储？

传统定义：分布式存储系统是大量PC 服务器通过Internet 互联，对外提供一个整体的服务。

分布式存储系统具有以下的几个特性：

可扩展：分布式存储系统可以扩展到几百台甚至几千台这样的一个集群规模，系统的整体性能线性增长。

低成本：分布式存储系统的自动容错、自动负载均衡的特性，允许分布式存储系统可以构建在低成本的服务器上。另外，线性的扩展能力也使得增加、减少服务器的成本低，实现分布式存储系统的自动运维。

高性能：无论是针对单台服务器，还是针对整个分布式的存储集群，都要求分布式存储系统具备高性能。

易用：分布式存储系统需要对外提供方便易用的接口，另外，也需要具备完善的监控、运维工具，并且可以方便的与其他的系统进行集成。分布式存储系统的挑战主要在于数据和状态信息的持久化，要求在自动迁移、自动容错和并发读写的过程中，保证数据的一致性。

容错：可以快速检测到服务器故障，并自动的将在故障服务器上的数据进行迁移。

负载均衡：新增的服务器在集群中保障负载均衡？数据迁移过程中保障不影响现有的服务。

事务与并发控制：实现分布式事务。

易用性：设计对外接口，使得设计的系统易于使用。

#### 你所知道的分布式爬虫方案有哪些？

1.Slaver 端从Master 端拿任务（Request/url/ID）进行数据抓取，在抓取数据的同时也生成新任务，并将任务抛给Master。Master 端只有一个Redis 数据库，负责对Slaver 提交的任务进行去重、加入待爬队列。

优点： scrapy-redis 默认使用的就是这种策略，我们实现起来很简单，因为任务调度等工作scrapy-redis 都已经帮我们做好了，我们只需要继承RedisSpider、指定redis\_key 就行了。

缺点： scrapy-redis 调度的任务是Request 对象，里面信息量比较大（不仅包含url，还有callback 函数、headers 等信息），导致的结果就是会降低爬虫速度、而且会占用Redis 大量的存储空间。当然我们可以重写方法实现调度url 或者用户ID。

2.Master 端跑一个程序去生成任务（Request/url/ID）。Master 端负责的是生产任务，并把任务去重、加入到待爬队列。Slaver 只管从Master 端拿任务去爬。

优点： 将生成任务和抓取数据分开，分工明确，减少了Master 和Slaver 之间的数据交流；Master 端生成任务还有一个好处就是：可以很方便地重写判重策略（当数据量大时优化判重的性能和速度还是很重要的）。

缺点： 像QQ 或者新浪微博这种网站，发送一个请求，返回的内容里面可能包含几十个待爬的用户ID，即几十个新爬虫任务。但有些网站一个请求只能得到一两个新任务，并且返回的内容里也包含爬虫要抓取的目标信息，如果将生成任务和抓取任务分开反而会降低爬虫抓取效率。毕竟带宽也是爬虫的一个瓶颈问题，我们要秉着发送尽量少的请求为原则，同时也是为了减轻网站服务器的压力，要做一只有道德的Crawler。所以，视情况而定。

3.Master 中只有一个集合，它只有查询的作用。Slaver 在遇到新任务时询问Master 此任务是否已爬，如果未爬则加入Slaver 自己的待爬队列中，Master 把此任务记为已爬。它和策略一比较像，但明显比策略一简单。策略一的简单是因为有scrapy-redis 实现了scheduler 中间件，它并不适用于非scrapy 框架的爬虫。

优点： 实现简单，非scrapy 框架的爬虫也适用。Master 端压力比较小，Master 与Slaver 的数据交流也不大。

缺点：“健壮性”不够，需要另外定时保存待爬队列以实现“断点续爬”功能。各Slaver的待爬任务不通用。

如果把Slaver 比作工人，把Master 比作工头。策略一就是工人遇到新任务都上报给工头，需要干活的时候就去工头那里领任务；策略二就是工头去找新任务，工人只管从工头那里领任务干活；策略三就是工人遇到新任务时询问工头此任务是否有人做了，没有的话工人就将此任务加到自己的“行程表”。

#### 在爬取的时候遇到某些内容字段缺失怎么判断及处理？

在爬取数据时，对于没有数据的内容可以给与空值或给与一个默认值，具体根据爬取内容而定  
例如爬取一条文字信息，就可以给与空值  
如果爬取的数据要进行统计计算，就要根据实际情况，给与0值或1值

# WEB框架

## Django

#### 你对 Django 的认识？

Django 是走大而全的方向，它最出名的是其全自动化的管 理后台:只需要使用起 ORM，做简单的对象定义，它就能自动生成数据库结构、以及全功能的管理后台。

优点：  
超高的开发效率。  
适用的是中小型的网站，或者是作为大型网站快速，实现产品雏形的工具。  
彻底的将代码、样式分离; Django 从根本上杜绝在模板中进行编码、处理数据的可能。

缺点：  
其性能扩展有限;  
采用 Django 的项目，在流量达到一定规模后，都需要对其进行重构，才能满足性能的要求。  
Django 内置的 ORM 跟框架内的其他模块耦合程度高。

#### Django HTTP 请求的处理流程?

Django 和其他Web 框架的HTTP 处理的流程大致相同，Django 处理一个Request 的过程是首先通过中间件，然后再通过默认的URL 方式进行的。我们可以在Middleware 这个地方把所有Request 拦截住，用我们自己的方式完成处理以后直接返回Response。

1. 加载配置  
Django 的配置都在“Project/settings.py” 中定义，可以是Django 的配置，也可以是自定义的配置，并且都通过django.conf.settings 访问，非常方便。

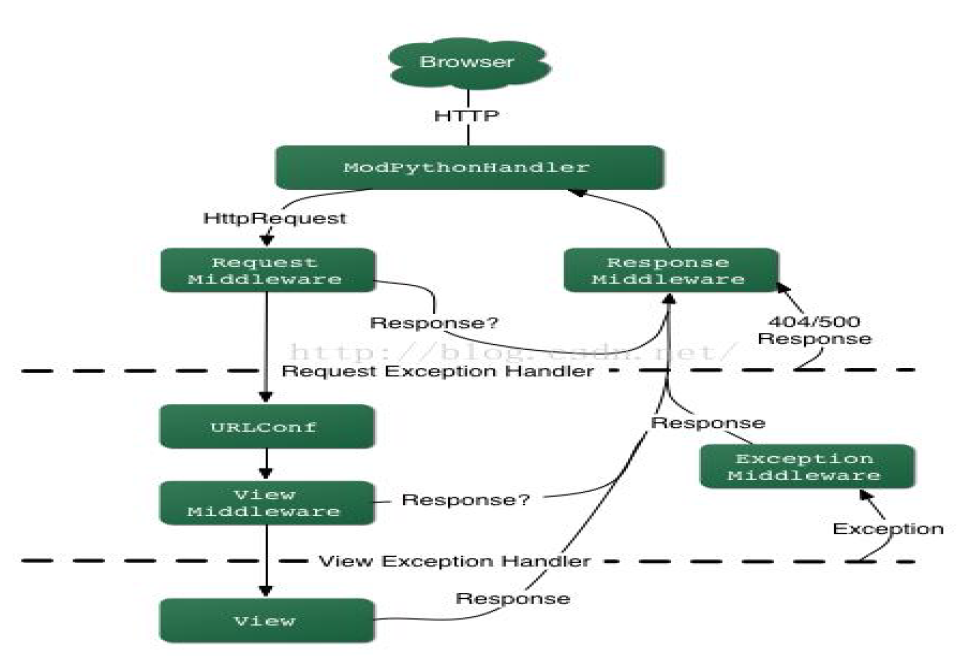
2. 启动  
最核心动作的是通过django.core.management.commands.runfcgi 的Command 来启动，它运行django.core.servers.fastcgi 中的runfastcgi，runfastcgi 使用了flup 的WSGIServer 来启动fastcgi 。而WSGIServer 中携带了django.core.handlers.wsgi 的WSGIHandler 类的一个实例，通过WSGIHandler 来处理由Web 服务器（比如Apache，Lighttpd 等）传过来的请求，此时才是真正进入Django 的世界。

3. 处理Request  
当有HTTP 请求来时，WSGIHandler 就开始工作了，它从BaseHandler 继承而来。WSGIHandler 为每个请求创建一个WSGIRequest 实例，而WSGIRequest 是从http.HttpRequest 继承而来。接下来就开始创建Response 了。

4. 创建Response  
BaseHandler 的get\_response 方法就是根据request 创建response，而具体生成response 的动作就是执行urls.py 中对应的view 函数了，这也是Django 可以处理“友好URL ”的关键步骤，每个这样的函数都要返回一个Response 实例。此时一般的做法是通过loader 加载template 并生成页面内容，其中重要的就是通过ORM 技术从数据库中取出数据，并渲染到Template 中，从而生成具体的页面了。

5. 处理Response  
Django 返回Response 给flup，flup 就取出Response 的内容返回给Web 服务器，由后者返回给浏览器。

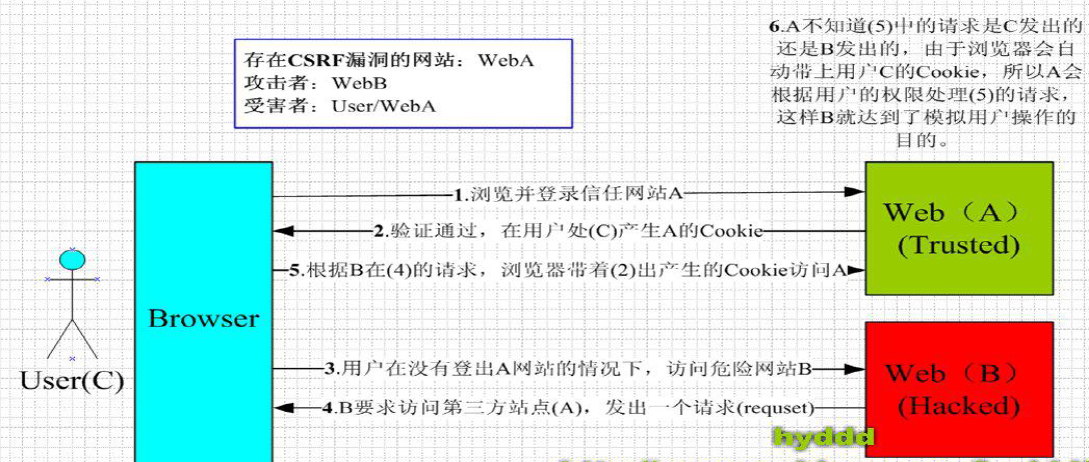
总之，Django 在fastcgi 中主要做了两件事：处理Request 和创建Response，而它们对应的核心就是“ urls 分析”、“模板技术”和“ ORM 技术”。



如图所示，一个HTTP 请求，首先被转化成一个HttpRequest 对象，然后该对象被传递给  
Request 中间件处理，如果该中间件返回了Response，则直接传递给Response 中间件做收尾处理。  
否则的话Request 中间件将访问URL 配置，确定哪个view 来处理，在确定了哪个view 要执行，但是还没有执行该view 的时候，系统会把request 传递给view 中间件处理器进行处理，如果该中间件返回了Response，那么该Response 直接被传递给Response 中间件进行后续处理，否则将执行确定的view 函数处理并返回Response，在这个过程中如果引发了异常并抛出，会被Exception中间件处理器进行处理。

#### CSRF是什么？ 如何防御？

CSRF(Cross-site request forgery跨站请求伪造，也被称成为“one click attack”或者session riding，通常缩写为CSRF或者XSRF，是一种对网站的恶意利用。



如何防御？  
 在服务端进行防御（目前主流的防御方法）。  
 1. 验证http referer 字段。  
 2. 在请求地址中添加token并验证  
 3. 在http中自定义属性并验证  
  
Django的CSRF防护机制  
 django 第一次响应来自某个客户端的请求时，会在服务器端随机生成一个 token，把这个 token 放在 cookie 里。然后每次 POST 请求都会带上这个 token，这样就能避免被 CSRF 攻击。  
  
Django中避免CSRF的方法  
1. 首先，最基本的原则是：GET 请求不要用有副作用。也就是说任何处理 GET 请求的代码对资源的访问都一定要是“只读“的  
2. 要启用 django.middleware.csrf.CsrfViewMiddleware 这个中间件  
3. 再次，在所有的 POST 表单元素时，需要加上一个 {% csrf\_token %} tag  
4. 在渲染模块时，使用 RequestContext。RequestContext 会处理 csrf\_token 这个 tag, 从而自动为表单添加一个名为 csrfmiddlewaretoken 的 input

#### HttpRequest 和HttpResponse 是什么?干嘛用的？

HttpRequest 是django 接受用户发送多来的请求报文后，将报文封装到HttpRequest 对象中去。  
HttpResponse 返回的是一个应答的数据报文。render 内部已经封装好了HttpResponse 类。  
视图的第一个参数必须是HttpRequest 对象，两点原因：表面上说，他是处理web 请求的，所以必须是请求对象，根本上说，他是基于请求的一种web 框架，所以，必须是请求对象。因为view 处理的是一个request 对象，请求的所有属性我们都可以根据对象属性的查看方法来获取具体的信息：格式：request.属性  
request.path 请求页面的路径，不包含域名  
request.get\_full\_path 获取带参数的路径  
request.method 页面的请求方式  
request.GET GET 请求方式的数据  
request.POST POST 请求方式的数据  
request.COOKIES 获取cookie  
request.session 获取session  
request.FILES 上传图片（请求页面有enctype=**"multipart/form-data"**属性时FILES 才有数据。  
？a=10 的键和值时怎么产生的，键是开发人员在编写代码时确定下来的，值时根据数据生成或者用户填写的，总之是不确定的。  
403 错误：表示资源不可用，服务器理解客户的请求，但是拒绝处理它，通常由于服务器上文件和目录的权限设置导致的web 访问错误。如何解决：1、把中间件注释。2、在表单内部添加{% scrf\_token %}request.GET.get()取值时如果一键多值情况，get 是覆盖的方式获取的。getlist（）可以获取多个值。在一个有键无值的情况下，该键名c 的值返回空。有键无值：c: getlist 返回的是列表，空列表在无键无值也没有默认值的情况下，返回的是None 无键无值：e:**None**

HttpResponse 常见属性：  
content： 表示返回的内容  
charset: 表示response 采用的编码字符集，默认是utf-8  
status\_code:返回的HTTP 响应状态码3XX 是对请求继续进一步处理，常见的是重定向。

常见方法：  
init:创建httpResponse 对象完成返回内容的初始化  
set\_cookie：设置Cookie 信息：格式：set\_cookies(**'key'**,**'value'**,max\_age=**None**,expires=**None**)  
max\_age 是一个整数，表示指定秒数后过期，expires 指定过期时间，默认两个星期后过期。  
write 向响应体中写数据  
应答对象：  
方式一：render(request,**"index.html"**) 返回一个模板render(request,**"index.html"**, context) 返回一个携带动态数据的页面  
方式二：render\_to\_response(**"index.html"**) 返回一个模板页面  
方式三：redirect(**"/"**) 重定向  
方式四：HttpResponseRdeirect(**"/"**) 实现页面跳转功能  
方式五：HttpResponse（**"atguigu"**)在返回到额页面中添加字符串内容  
方式六：HttpResponseJson() 返回的页面中添加字符串内容。  
JsonResponse 创建对象时候接收字典作为参数，返回的对象是一个json 对象。能接收Json 格式数据的场景，都需要使用view 的JsonResponse 对象返回一个json 格式数据  
ajax 的使用场景，页面局部刷新功能。ajax 接收Json 格式的数据。  
在返回的应答报文中，可以看到JsonResponse 应答的content-Type 内容是application/json

ajax 实现网页局部刷新功能：ajax 的get()方法获取请求数据ajax 的each()方法遍历输出这些数据

#### session 和cookie的区别是什么？为什么session是安全的？

Cookie 是保存在客户端的键值对，主要包括：名字、值、过期时间、路径和域。路径和域组成cookie的作用范围，过期时间若不设置，有效期就是到浏览器的关闭，关闭浏览器cookie失效。一般保存在内存中。若是设置了过期时间，cookie存储在硬盘上，有效期是设置的时间，不会随着浏览器的关闭而失效，在本地内存中能够修改。所以重要的信息不应该保存在本地的cookie中。

Session机制是服务端的机制，服务端以散列表的形式存储，当程序需要为某个客户端的请求创建一个session时，服务器首先检查这个客户端的请求里是否已包含了一个session标识（称为session id），如果已包含则说明以前已经为此客户端创建过session，服务器就按照session id把这个session检索出来使用（检索不到，会新建一个），如果客户端请求不包含session id，则为此客户端创建一个session并且生成一个与此session相关联的session id，session id的值应该是一个既不会重复，又不容易被找到规律以仿造的字符串，这个session id将被在本次响应中返回给客户端保存。保存这个session id的方式可以采用cookie，这样在交互过程中浏览器可以自动的按照规则把这个标识发送给服务器。一般这个cookie的名字都是类似于SEEESIONID。但cookie可以被人为的禁止，则必须有其他机制以便在cookie被禁止时仍然能够把session id传递回服务器。经常被使用的一种技术叫做URL重写，就是把session id直接附加在URL路径的后面。还有一种技术叫做表单隐藏字段。就是服务器会自动修改表单，添加一个隐藏字段，以便在表单提交时能够把session id传递回服务器。

主要区别：  
 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  
 5. 所以个人建议：将登陆信息等重要信息存放为SESSION，其他信息如果需要保留，可以放在COOKIE

#### Django和Tornado的关系？

都是python 的web开发框架，django 是一站式（全栈）的开发框架，比较大，开发比较快，适合中小型的压力不大的项目。  
tornado是一个epoll异步非阻塞的开发框架,支持长连接，速度比较快，异步加载比较快，适合定制功能比较独特的web网站。

#### Django 中models 利用ORM 对Mysql 进行查表的语句（多个语句）？

字段查询

all():返回模型类对应表格中的所有数据。

get():返回表格中满足条件的一条数据，如果查到多条数据，则抛异常：MultipleObjectsReturned，查询不到数据，

则抛异常：DoesNotExist。

filter():参数写查询条件，返回满足条件QuerySet 集合数据。

条件格式：  
\*\*模型类属性名\*\*\_\_条件名=值  
注意：此处是模型类属性名，不是表中的字段名  
关于filter 具体案例如下：  
判等exact。  
1. BookInfo.object.filter(id=1)  
2. BookInfo.object.filter(id\_\_exact=1)此处的\_\_exact 可以省略模糊查询like  
例：查询书名包含**'传'**的图书。contains  
1. contains BookInfo.objects.filter(btitle\_\_contains=’传’)

空查询where 字段名isnull  
1. BookInfo.objects.filter(btitle\_\_isnull=**False**)

范围查询where id **in** (1，3，5)  
1. BookInfo.objects.filter(id\_\_in=[1，3，5])

比较查询gt lt(less than) gte(equal) lte  
1. BookInfo.objects.filter(id\_\_gte=3)

日期查询  
1. BookInfo.objects.filter(bpub\_date\_\_year = 1980)  
2. BookInfo.objects.filter(bpub\_date\_\_gt = date(1980，1，1))

exclude:返回不满足条件的数据。  
3. BookInfo.objects.exclude(id=3)

F 对象  
作用：用于类属性之间的比较条件。  
1. **from** django.db.models **import** F  
2. 例：where bread > bcomment BookInfo.objects.filter(bread\_\_gt =F(‘bcomment’))  
3. 例：BookInfo.objects.filter(bread\_\_gt=F(‘bcomment’)\*2)

Q 对象  
作用：用于查询时的逻辑条件。可以对Q 对象进行&|~操作。  
1. **from** django.db.models **import** Q  
2. BookInfo.objects.filter(id\_\_gt=3， bread\_\_gt=30)  
3. BooInfo.objects.filter(Q(id\_\_gt=3) & Q(bread\_\_gt=3))  
4. 例：BookInfo.objects.filter(Q(id\_\_gt=3) | Q(bread\_\_gt=30))  
5. 例：BookInfo.objects.filter(~Q(id=3))

order\_by 返回QuerySet  
作用：对查询结果进行排序。  
1. 例： BookInfo.objects.all().order\_by(**'id'**)  
2. 例： BookInfo.objects.all().order\_by(**'-id'**)  
3. 例：BookInfo.objects.filter(id\_\_gt=3).order\_by(**'-bread'**)

聚合函数  
作用：对查询结果进行聚合操作。  
1. sum count max min avg

aggregate：调用这个函数来使用聚合。  
1. **from** django.db.models **import** Sum，Count，Max，Min，Avg  
2. 例：BookInfo.objects.aggregate(Count(**'id'**))

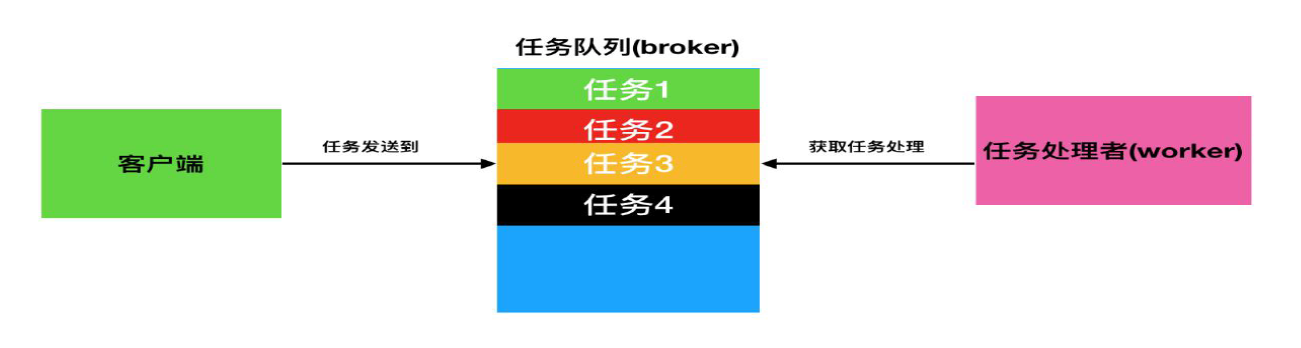
{**'id\_\_count'**: 5} 注意返回值类型及键名  
1. 例：BookInfo.objects.aggregate(Sum(‘bread’))  
{‘bread\_\_sum’:120} 注意返回值类型及键名

count 函数  
作用：统计满足条件数据的数目。  
例：统计所有图书的数目。  
1. BookInfo.objects.all().count()  
例：统计id 大于3 的所有图书的数目。  
1. BookInfo.objects.filter(id\_\_gt = 3).count()

模型类关系  
一对多关系  
例：图书类-英雄类  
models.ForeignKey() 定义在多的类中。  
2）多对多关系  
例：新闻类-新闻类型类  
models.ManyToManyField() 定义在哪个类中都可以。  
3）一对一关系  
例：员工基本信息类-员工详细信息类  
models.OneToOneField() 定义在哪个类中都可以。

#### Celery 分布式任务队列？

情景：用户发起request，并等待response 返回。在本些views 中，可能需要执行一段耗时的程序，那么用户就会等待很长时间，造成不好的用户体验，比如发送邮件、手机验证码等。  
使用celery 后，情况就不一样了。解决：将耗时的程序放到celery 中执行。  
将多个耗时的任务添加到队列queue 中，也就是用redis 实现broker 中间人，然后用多个worker 去监听队列里的任务去执行。



任务task：就是一个Python 函数。  
队列queue：将需要执行的任务加入到队列中。  
工人worker：在一个新进程中，负责执行队列中的任务。  
代理人broker：负责调度，在布置环境中使用redis。

#### django 中间件的使用？

Django 在中间件中预置了六个方法，这六个方法的区别在于不同的阶段执行，对输入或输出进行干预，方法如下：  
1.初始化：无需任何参数，服务器响应第一个请求的时候调用一次，用于确定是否启用当前中间件。  
**def** \_\_init\_\_():  
 **pass**2.处理请求前：在每个请求上调用，返回None 或HttpResponse 对象。  
**def** process\_request(request):  
 **pass**3.处理视图前：在每个请求上调用，返回None 或HttpResponse 对象。  
**def** process\_view(request, view\_func, view\_args, view\_kwargs):  
 **pass**4.处理模板响应前：在每个请求上调用，返回实现了render 方法的响应对象。  
**def** process\_template\_response(request, response):  
 **pass**5.处理响应后：所有响应返回浏览器之前被调用，在每个请求上调用，返回HttpResponse 对象。  
**def** process\_response(request, response):  
 **pass**6.异常处理：当视图抛出异常时调用，在每个请求上调用，返回一个HttpResponse 对象。  
**def** process\_exception(request,exception):  
 **pass**

#### 简述Django 下的（内建的）缓存机制?

一个动态网站的基本权衡点就是，它是动态的。每次用户请求页面，服务器会重新计算。从开销处理的角度来看，这比你读取一个现成的标准文件的代价要昂贵的多。  
这就是需要缓存的地方。  
Django 自带了一个健壮的缓存系统来保存动态页面这样避免对于每次请求都重新计算。方便起见，Django 提供了不同级别的缓存粒度：可以缓存特定视图的输出、可以仅仅缓存那些很难生产出来的部分、或者可以缓存整个网站Django 也能很好的配合那些“下游”缓存， 比如Squid 和基于浏览器的缓存。这里有一些缓存不必要直接去控制但是可以提供线索， (via HTTPheaders)关于网站哪些部分需要缓存和如何缓存。  
设置缓存：  
缓存系统需要一些设置才能使用。也就是说，你必须告诉他你要把数据缓存在哪里- 是数据库中，文件系统或者直接在内存中。这个决定很重要，因为它会影响你的缓存性能，是的，一些缓存类型要比其他的缓存类型更快速。  
你的缓存配置是通过setting 文件的CACHES 配置来实现的。这里有CACHES 所有可配置的变量值。

#### Django使用redis 做缓存

要安装redis **for** django的插件Django\_redis, redis\_cache, 然后再在setting中配置  
  
1. **'django.middleware.cache.UpdateCacheMiddleware'** *# 必须放在第一个位置*2. **'django.middleware.cache.FetchFromCacheMiddleware'** *# 必须放在最后一个位置*如果要使用要在views.py中导入  
From django.core.cache **import** cache  
  
Ps：django其他的缓存使用方法大同小异  
1. 使用memcache缓存（django官方推荐）  
2. 数据库的缓存  
3. 本地内存的缓存（默认的）  
4. 文件系统的缓存  
5. 虚拟缓存（用于开发）  
6. 自定义后端缓存

#### Django MVC 和MVT

MVC模式  
model模型：负责数据从数据库中存入取出。  
view视图：将数据格式化后呈现给用户。  
controller控制器：根据浏览器的url地址访问“模型”获取数据，并调用“视图”显示这些数据。  
  
MVT模式  
m表示model，负责与数据库交互  
v表示view，是核心，负责接收请求、获取数据、返回结果  
t表示template，负责呈现内容到浏览器

#### Django的继承有哪些？

1..模板继承使用： {% extends ‘base.html’ %}  
 模板继承允许你建立一个基本的”骨架”模板, 它包含你所有最常用的站点元素并定义了一些可以被子模板覆盖的block.这个base.html 就是父模板， 在子模板中你能够重载、添加、或者保留block的内容。  
  
2. 模型继承（models的继承）：  
 主要包含三种:抽象基类继承、多表继承、代理继承  
 1. Meta的继承  
 创建抽象基类的时候，Django 会将你在基类中所声明的有效的 Meta 内嵌类做为一个属性。如果子类没有声明它自己的 Meta 内嵌类，它就会继承父类的 Meta 。子类的 Meta 也可以直接继承父类的 Meta 内嵌类，对其进行扩展。  
  
 2. 多表继承。  
 如果你想从现有的Model继承并让每个Model都有自己的数据表，那么使用多重表继承Multi-table inheritance。使用这种继承方式时，同一层级下的每个子 model 都是一个真正意义上完整的 model 。每个子 model 都有专属的数据表，都可以查询和创建数据表。继承关系在子 model 和它的每个父类之间都添加一个链接 (通过一个自动创建的 OneToOneField 来实现)。  
  
 3. 代理模型  
 如果你只想在 model 中修改 Python-level 级的行为，而不涉及字段改变。比如：更改默认的 manager ，或是添加一个新方法。 代理 model (Proxy models) 适用于这种场合。代理 model 继承方式要做的：为原始 model 创建一个代理(proxy)。你可以创建，删除，更新代理 model 的实例，而且所有的数据都可以象使用原始 model 一样被保存。不同之处在于：你可以在代理 model 中改变默认的排序设置和默认的 manager ，更不会对原始 model 产生影响。声明代理 model 和声明普通 model 没有什么不同。设置Meta 内置类中 proxy 的值为 **True**，就完成了对代理 model 的声明。

#### Django如何在Model保存前做一定的固定操作，比如写一条日志

利用Django的Model的Signal Dispatcher, 通django.db.models.signals.pre\_save()方法，在事件发生前，发射触发信号。

#### 请简要概括下Django测试工具

Django测试框架非常简单,首选方法是使用python标准库中的unittest模块。  
Django的单元测试使用python的unittest模块，这个模块使用基于类的方法来定义测试。类名为django.test.TestCase,继承于python的unittest.TestCase。  
  
如果测试是基于数据库访问的(读取、查询Model)，一定要用django.test.TestCase建立测试类，而不要用unittest.TestCase。

#### 解释下django-debug-toolbar的使用

使用django开发站点时，可以使用django-debug-toolbar来进行调试。在settings.py中添加**'**debug\_toolbar.middleware.DebugToolbarMiddleware**'**到项目的MIDDLEWARE\_CLASSES 内

#### Django中如何连接多个数据库？

连接多个数据库是Django1.2版本后新特性  
  
详细配置：  
https://code.ziqiangxuetang.com/django/django-multi-database.html

#### 什么是RESTful API，谈谈你的理解？

REST:Representational State Transfer 的缩写，翻译：“具象状态传输”。一般解释为“表现层状态转换”。  
REST 是设计风格而不是标准。是指客户端和服务器的交互形式。我们需要关注的重点是如何设计REST 风格的网络接口。  
REST 的特点：  
1.具象的。一般指表现层，要表现的对象就是资源。比如，客户端访问服务器，获取的数据就是资源。比如文字、图片、音视频等。  
2.表现：资源的表现形式。txt 格式、html 格式、json 格式、jpg 格式等。浏览器通过URL 确定资源的位置，但是需要在HTTP 请求头中，用Accept 和Content-Type 字段指定，这两个字段是对资源表现的描述。  
3.状态转换：客户端和服务器交互的过程。在这个过程中，一定会有数据和状态的转化，这种转化叫做状态转换。其中，GET 表示获取资源，POST 表示新建资源，PUT 表示更新资源，DELETE 表示删除资源。HTTP 协议中最常用的就是这四种操作方式。  
RESTful 架构：  
1.每个URL 代表一种资源；  
2.客户端和服务器之间，传递这种资源的某种表现层；  
3.客户端通过四个http 动词，对服务器资源进行操作，实现表现层状态转换。

#### RESTful API设计规范

URI规范  
不要用大写  
单词间使用下划线**'\_'**不使用动词，资源要使用名词复数形式，如：user、rooms、tickets  
层级 >= 三层，则使用**'?'**带参数  
users/1/address/2/citys (bad)

s/citys?users=1&address=2; (good)  
  
Method  
GET：查询资源  
POST：创建资源  
PUT/PATCH  
PUT：全量更新资源（提供改变后的完整资源）  
PATCH：局部更新资源（仅提供改变的属性）  
DELETE：删除资源  
  
详细链接： https://www.cnblogs.com/EasonJim/p/7770711.html

#### wsgi和uwsgi是什么？

WSGI  
 Web服务器网关接口（Python Web Server Gateway Interface，缩写为WSGI）是为Python语言定义的Web服务器和Web应用程序或框架之间的一种简单而通用的接口. WSGI将 web 组件分为三类： web服务器，web中间件,web应用程序， wsgi基本处理模式为 ： WSGI Server -> (WSGI Middleware)\* -> WSGI Application 。

uWSGI  
 uWSGI是一个Web服务器，它实现了WSGI协议、uwsgi、http等协议。uWSGI，既不用wsgi协议也不用FastCGI协议，而是自创了一个uwsgi的协议，uwsgi协议是一个uWSGI服务器自有的协议，它用于定义传输信息的类型（type of information），每一个uwsgi packet前4byte为传输信息类型描述，它与WSGI相比是两样东西。

## Flask

#### 手写一个Flask的HelloWorld

**from** flask **import** Flask  
  
app = Flask(\_\_name\_\_)  
  
@app.route(**'/'**)  
**def** index():  
 **return 'Hello world!'  
  
if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 app.run()

#### Flask与Django框架有什么区别？

Django功能大而全，Flask只包含基本的配置 Django的一站式解决的思路，能让开发者不用在开发之前就在选择应用的基础设施上花费大量时间。Django有模板，表单，路由，认证，基本的数据库管理等等内建功能。与之相反，Flask只是一个内核，默认依赖于两个外部库： Jinja2 模板引擎和 Werkzeug WSGI 工具集，其他很多功能都是以扩展的形式进行嵌入使用。  
  
Flask 比 Django 更灵活 用Flask来构建应用之前，选择组件的时候会给开发者带来更多的灵活性 ，可能有的应用场景不适合使用一个标准的ORM(Object-Relational Mapping 对象关联映射)，或者需要与不同的工作流和模板系统交互。

#### Flask框架依赖的组件是什么？

wsgi方面主要依赖于werkzeug，html模板方面主要依赖于jinja2

#### Flask蓝图的作用？

蓝图：用于实现单个应用的视图、模板、静态文件的集合。  
  
蓝图就是模块化处理的类。  
  
简单来说，蓝图就是一个存储操作路由映射方法的容器，主要用来实现客户端请求和URL相互关联的功能。 在Flask中，使用蓝图可以帮助我们实现模块化应用的功能。

#### 列举使用过的Flask第三方组件？

Flask-Mail  
Flask-SQLAlchemy  
Flask-Bootstrap  
Flask-WTForm  
Flask-Script  
Flask-Migrate

#### 简述Flask上下文管理流程?

简单来说，falsk上下文管理可以分为三个阶段：  
1、请求进来时，将请求相关的数据放入上下问管理中  
2、在视图函数中，要去上下文管理中取值  
  
详细点来说：  
1、请求刚进来，将request，session封装在RequestContext类中，app，g封装在AppContext类中，并通过LocalStack将requestcontext和appcontext放入Local类中  
2、视图函数中，通过localproxy--->偏函数--->localstack--->local取值  
3、请求相应时，先执行save.session()再各自执行pop(),将local中的数据清除

#### Flask中g对象的作用？

g：**global**1. g对象是专门用来保存用户的数据的。   
2. g对象在一次请求中的所有的代码的地方，都是可以使用的。

g 相当于一次请求的全局变量，当请求进来时将g和current\_app封装为一个APPContext类，在通过LocalStack将Appcontext放入Local中，取值时通过偏函数，LocalStack、loca　l中取值，响应时将local中的g数据删除：

#### Flask 中请求钩子的理解和应用？

请求钩子是通过装饰器的形式实现的，支持以下四种：  
1，before\_first\_request 在处理第一个请求前运行  
2，before\_request:在每次请求前运行  
3，after\_request:如果没有未处理的异常抛出，在每次请求后运行  
4，teardown\_request:即使有未处理的异常抛出，在每次请求后运行

#### Flask中blinker是什么？

Flask框架中的信号基于blinker，其主要就是让开发者可是在flask请求过程中定制一些用户行为。简单来说就是flask在列表里面，预留了几个空列表，在里面存东西。信号通过发送通知来帮助你解耦应用。简言之，信号允许某个发送者通知接收者有事情发生了

#### Flask 项目中如何实现session 信息的写入？

Flask 中有三个session：  
第一个：数据库中的session，例如:db.session.add()  
第二个：在flask\_session 扩展中的session，使用：**from** flask\_session importSession，使用第三方扩展的session 可以把信息存储在服务器中，客户端浏览器中只存储sessionid。  
第三个：flask 自带的session，是一个请求上下文， 使用：**from** flask **import** session。自带的session 把信息加密后都存储在客户端的浏览器cookie 中。

#### Flask中的session是什么时候创建，什么时候销毁的？

当请求进来时，会将requset和session封装为一个RequestContext对象，通过LocalStack将RequestContext放入到Local对象中，因为请求第一次来session是空值，所以执行open\_session,给session（uuid4()）赋值，再通过视图函数处理，请求响应时执行save.session,将签名session写入cookie中，再讲Local中的数值pop掉。

#### flask中一共有几个LocalStack和Local对象

两个LocalStack，两个Local  
request、session共同用一个LocalStack和Local  
g、app共同用一个Localstack和Local

## Tornado

#### 简述Tornado框架的特点。

特点：  
作为Web框架，是一个轻量级的Web框架，其拥有异步非阻塞IO的处理方式。  
作为Web服务器，Tornado有较为出色的抗负载能力，官方用nginx反向代理的方式部署Tornado和其它Python web应用框架进行对比，结果最大浏览量超过第二名近40%。  
  
性能：  
Tornado有着优异的性能。它试图解决C10k问题，即处理大于或等于一万的并发。

#### Tornado中静态文件是如何处理的？ 如： <link href="{{static\_url("commons.css")}}" rel="stylesheet" />

static\_url()自动去配置的路径下找commons.css文件

#### Tornado操作MySQL使用的模块？

torndb、mysqldb

#### Tornado操作redis使用的模块？

Tornado-redis

#### 简述Tornado框架的适用场景？

web聊天室，在线投票，处理高并发任务

## Nginx

#### 什么是Nginx？

Nginx是一个高性能的HTTP和反向代理服务器，及电子邮件（IMAP/POP3）代理服务器，同时也是一个非常高效的反向代理、负载平衡。

#### 为什么要用Nginx？

优点：  
跨平台、配置简单非阻塞、高并发连接：处理2-3万并发连接数，官方监测能支持5万并发

内存消耗小：开启10个nginx才占150M内存，Nginx采取了分阶段资源分配技术Nginx处理静态文件好,耗费内存少

内置的健康检查功能：如果有一个服务器宕机，会做一个健康检查，再发送的请求就不会发送到宕机的服务器了。重新将请求提交到其他的节点上。

节省宽带：支持GZIP压缩，可以添加浏览器本地缓存

稳定性高：宕机的概率非常小

master/worker结构：一个master进程，生成一个或者多个worker进程

接收用户请求是异步的：浏览器将请求发送到nginx服务器，它先将用户请求全部接收下来，再一次性发送给后端web服务器，极大减轻了web服务器的压力一边接收web服务器的返回数据，一边发送给浏览器客户端网络依赖性比较低，只要ping通就可以负载均衡可以有多台nginx服务器

事件驱动：通信机制采用epoll模型

#### 为什么Nginx性能这么高？

得益于它的事件处理机制：  
异步非阻塞事件处理机制：运用了epoll模型，提供了一个队列，排队解决

#### 为什么不使用多线程？

Apache: 创建多个进程或线程，而每个进程或线程都会为其分配cpu和内存（线程要比进程小的多，所以worker支持比perfork高的并发），并发过大会榨干服务器资源。  
Nginx: 采用单线程来异步非阻塞处理请求（管理员可以配置Nginx主进程的工作进程的数量）(epoll)，不会为每个请求分配cpu和内存资源，节省了大量资源，同时也减少了大量的CPU的上下文切换。所以才使得Nginx支持更高的并发。

#### Nginx是如何处理一个请求的呢？

首先，nginx在启动时，会解析配置文件，得到需要监听的端口与ip地址，然后在nginx的master进程里面先初始化好这个监控的socket，再进行listen然后再fork出多个子进程出来, 子进程会竞争accept新的连接。  
此时，客户端就可以向nginx发起连接了。当客户端与nginx进行三次握手，与nginx建立好一个连接后，某一个子进程会accept成功，然后创建nginx对连接的封装，即ngx\_connection\_t结构体  
接着，根据事件调用相应的事件处理模块，如http模块与客户端进行数据的交换。  
最后，nginx或客户端来主动关掉连接，到此，一个连接就寿终正寝了

#### 正向代理

一个位于客户端和原始服务器(origin server)之间的服务器，为了从原始服务器取得内容，客户端向代理发送一个请求并指定目标(原始服务器)，然后代理向原始服务器转交请求并将获得的内容返回给客户端。客户端才能使用正向代理  
正向代理总结就一句话：代理端代理的是客户端

#### 反向代理

反向代理（Reverse Proxy）方式是指以代理服务器来接受internet上的连接请求，然后将请求，发给内部网络上的服务器  
并将从服务器上得到的结果返回给internet上请求连接的客户端，此时代理服务器对外就表现为一个反向代理服务器  
反向代理总结就一句话：代理端代理的是服务端

#### 动态资源，静态资源分离

动态资源、静态资源分离是让动态网站里的动态网页根据一定规则把不变的资源和经常变的资源区分开来，动静资源做好了拆分以后，我们就可以根据静态资源的特点将其做缓存操作，这就是网站静态化处理的核心思路  
动态资源、静态资源分离简单的概括是：动态文件与静态文件的分离

#### 为什么要做动、静分离？

在我们的软件开发中，有些请求是需要后台处理的（如：.jsp,.do等等），有些请求是不需要经过后台处理的（如：css、html、jpg、js等等文件）  
这些不需要经过后台处理的文件称为静态文件，否则动态文件。因此我们后台处理忽略静态文件。这会有人又说那我后台忽略静态文件不就完了吗， 当然这是可以的，但是这样后台的请求次数就明显增多了。在我们对资源的响应速度有要求的时候，我们应该使用这种动静分离的策略去解决动、静分离将网站静态资源（HTML，JavaScript，CSS，img等文件）与后台应用分开部署，提高用户访问静态代码的速度，降低对后台应用访问  
这里我们将静态资源放到nginx中，动态资源转发到tomcat服务器中

#### 负载均衡

负载均衡即是代理服务器将接收的请求均衡的分发到各服务器中  
负载均衡主要解决网络拥塞问题，提高服务器响应速度，服务就近提供，达到更好的访问质量，减少后台服务器大并发压力。

# Shell编程

#### Shell脚本里如何检查一个文件是否存在？如果不存在该如何处理？

**if** [ -f file.txt ]; then  
 echo **"文件存在!"  
else** echo **"文件不存在!"**fi

#### Shell脚本里如何检查一个变量是否为空？

*#!/bin/sh  
#filename: test.sh*para1=  
**if** [ ! -n $para1 ]; then  
 echo **"IS NULL"  
else** echo **"NOT NULL"**fi

#### 用shell写一个脚本，对文本中无序的一列数字排序

sort -n a|awk **'{a+=$0;print $0}END{print "SUM="a}'**

#### 请用shell脚本写出查找当前文件夹（/home）下所有的文本文件内容中包含有字符”a”的文件名称

sudo grep -rR **"atguigu"** /etc | cut -d **":"** -f 1 | tee /home/atguigu/output</span>

# 测试

#### 禅道和qc 的区别？

都是缺陷管理工具。  
A.QC  
作为缺陷管理工具，QC 在缺陷管理方面，做的相对完善。提bug 页面：填写内容可以根据测试需求，不断修改添加新的字段；以我上一家公司为例，在提bug 过程中，有一下几个必填项：  
Bug 状态（new、fixed、closed 等）、发现人员、缺陷发现阶段(测试阶段、上现阶段等)、缺陷来源（测试人员给出的bug 定位）、Bug 分类（功能、性能等问题）、测试阶段（单元测试、集成测试、系统测试等）、归属需求、缺陷回归次数、优先级、分配给，这些必填项再加上bug 标题和操作描述、上传附件，使很多疑问都变得清晰。  
缺陷查看页面：可以根据自己需要选择要呈现的字段，相对人性化可操作，每个显示的字段都可以进行筛选，使研发人员很快能定位到属于自己的bug，再根据bug 状态、优先级进行筛选，使未完结的bug 能有序并无遗漏地完成修改；页面还有注释功能，研发人员能写出针对本问题的各种感想，方便完善而又人性化。

B.禅道（开源版）  
禅道涉及面非常广，但是在缺陷管理这方面，与老牌的QC 还是略逊一筹。提bug 页面：页面是非常清晰整洁的web 页面，但是需要填写的字段，并没有完全覆盖开发和测试人员的全部需求。页面字段：产品模块（对应QC 中的项目）、所属项目（对应QC 中的需求）、影响版本（bug 所属版本？）、当前指派（修改bug 的人员）、bug 标题、重现步骤、相关需求（页面标注了这个字段，但是什么也没有显示，并且没有可填写的位置）、相关任务、类型/严重。

#### 编写测试计划的目的是

1. 使测试工作顺利进行  
2. 使项目参与人员沟通更舒畅  
3. 使测试工作更加系统化

#### 测试人员在软件开发过程中的任务是什么

1. 寻找Bug；  
2. 避免软件开发过程中的缺陷；  
3. 衡量软件的品质；  
4. 关注用户的需求。  
5. 总的目标是：确保软件的质量。

#### 您以往的工作中，一条软件缺陷（或者叫Bug）记录都包含了哪些内容？如何提交高质量的软件缺陷（Bug）记录？

一条Bug 记录最基本应包含：编号、Bug 所属模块、Bug 描述、Bug 级别、发现日期、发现人、修改日期、修改人、修改方法、回归结果等等；要有效的发现Bug 需参考需求以及详细设计等前期文档设计出高效的测试用例，然后严格执行测试用例，对发现的问题要充分确认肯定，然后再向外发布如此才能提高提交Bug 的质量。

#### 简述黑盒测试和白盒测试的优缺点

※ 黑盒测试的优点有：  
1）比较简单，不需要了解程序内部的代码及实现；  
2）与软件的内部实现无关；  
3）从用户角度出发，能很容易的知道用户会用到哪些功能，会遇到哪些问题；  
4）基于软件开发文档，所以也能知道软件实现了文档中的哪些功能；  
5）在做软件自动化测试时较为方便。

※ 黑盒测试的缺点有：  
1）不可能覆盖所有的代码，覆盖率较低，大概只能达到总代码量的30%；  
2）自动化测试的复用性较低。

※ 白盒测试的优点有：  
1）帮助软件测试人员增大代码的覆盖率，提高代码的质量，发现代码中隐藏的问题。

※ 白盒测试的缺点有：  
1）程序运行会有很多不同的路径，不可能测试所有的运行路径；测试基于代码，只能测试开发人员做的对不对，而不能知道设计的正确与否，可能会漏掉一些功能需求；系统庞大时，测试开销会非常大。

#### 简述常用的Bug 管理或者用例管理工具,并且描述其中一个工作流程。

常用：testlink，QC，mantis，禅道，TAPD，JIRA 。  
TAPD：产品创建(需求，计划，模块)-->项目创建（PM 排期、任务分解）-->研发(编码、单元测试等)-->测试(测试计划，用例，执行，bug，报告等)。

#### 请列出你所知道的软件测试种类，至少5 项

单元测试，集成测试，系统测试，验收测试。  
系统测试包含：功能测试，性能测试，压力测试，兼容性测试，健壮性测试，冒烟测试，文档测试。

#### Alpha 测试与Beta 测试的区别是什么？

Alpha 主要是模拟用户的操作和用户的环境。  
Beta 主要验证测试，准备进入发布阶段，Beta 测试是一种验收测试。

#### 举例说明什么是Bug？一个bug report 应包含什么关键字？

比如聊天中，点击发送按钮后，无法发送消息。  
标题，模块，严重程度，bug 类型，版本号，可否重现，描述，附件，日志等等。

# 数据结构与算法

## 理论知识

#### 算法的特征？

1）有穷性： 一个算法必须保证执行有限步骤之后结束；  
2）确切性： 算法的每一步骤必须有确切的定义；  
3）输入：一个算法有0 个或多个输入，以刻画运算对象的初始情况，所谓0 个输入是指算法本身给出了初始条件；  
4）输出：一个算法有一个或多个输出，以反映对输入数据加工后的结果。没有输出的算法是毫无意义的；  
5）可行性： 算法原则上能够精确地运行，而且人们用笔和纸做有限次数运算后即可完成。

#### 基础的数据结构有哪些？

基本的算法有: 排序算法(冒泡排序，插入排序， 快速排序， 归并排序)， 查找(二分查找)，搜索（（DFS）深度优先搜索，（BFS）广度优先搜索)，（Dijkstra 算法），动态规划算法，分类（朴素贝叶斯分类算法等）。

#### 基本的算法有哪些，怎么评价一个算法的好坏？

基本的算法有: 排序算法(简单排序， 快速排序， 归并排序)， 搜索(二分搜索)， 对于其他基本数据结构， 栈， 队列，树，都有一些基本的操作。

评价算法的好坏一般有两种: 时间复杂度和空间复杂度。  
时间复杂度：同样的输入规模(问题规模)话费多少时间。  
空间复杂度：同样的输入规模花费多少空间(主要是内存)。  
以上两点越小越好。  
稳定性：不会引文输入的不同而导致不稳定的情况发生。  
算法的思路是否简单：越简单越容易实现的越好。

#### 深度优先遍历和广度优先遍历的区别？

1） 二叉树的深度优先遍历的非递归的通用做法是采用栈，广度优先遍历的非递归的通用做法是采用队列。  
2） 深度优先遍历：对每一个可能的分支路径深入到不能再深入为止，而且每个结点只能访问一次。  
要特别注意的是，二叉树的深度优先遍历比较特殊，可以细分为先序遍历、中序遍历、后序遍历。具体说明如下：  
先序遍历：对任一子树，先访问根，然后遍历其左子树，最后遍历其右子树。  
中序遍历：对任一子树，先遍历其左子树，然后访问根，最后遍历其右子树。  
后序遍历：对任一子树，先遍历其左子树，然后遍历其右子树，最后访问根。  
广度优先遍历：又叫层次遍历，从上往下对每一层依次访问，在每一层中，从左往右（也可以从右往左）访问结点，访问完一层就进入下一层，直到没有结点可以访问为止。  
3）深度优先搜素算法：不全部保留结点，占用空间少；有回溯操作(即有入栈、出栈操作)，运行速度慢。  
广度优先搜索算法：保留全部结点，占用空间大； 无回溯操作(即无入栈、出栈操作)，运行速度快。通常，深度优先搜索法不全部保留结点，扩展完的结点从数据库中弹出删去，这样，一般在数据库中存储的结点数就是深度值，因此它占用空间较少。所以，当搜索树的结点较多，用其它方法易产生内存溢出时，深度优先搜索不失为一种有效的求解方法。  
广度优先搜索算法，一般需存储产生的所有结点，占用的存储空间要比深度优先搜索大得多，因此，程序设计中，必须考虑溢出和节省内存空间的问题。但广度优先搜索法一般无回溯操作，即入栈和出栈的操作，所以运行速度比深度优先搜索要快些。

## 字符串

#### 替换空格

请实现一个函数，将一个字符串中的空格替换成“%20”。例如，当字符串为We Are Happy.则经过替换之后的字符串为We%20Are%20Happy。

**class** Solution:  
 *# s 源字符串* **def** replaceSpace(self, s):  
 *# write code here* **if not** s:  
 **return ''** s = s.replace(**' '**,**'%20'**)  
 **return** s

#### 字符流中第一个不重复的字符

请实现一个函数用来找出字符流中第一个只出现一次的字符。例如，当从字符流中只读出前两个字符"go"时，第一个只出现一次的字符是"g"。当从该字符流中读出前六个字符“google"时，第一个只出现一次的字符是"l"。

**def** first\_not\_repeating\_char(string):  
 **if not** string:  
 **return** temp = dict([(x, 0) **for** x **in** range(26)])  
 **for** s **in** string:  
 *# ord('a') == 97* temp[ord(s) - 97] += 1  
 **for** s **in** string:  
 **if** temp[ord(s) - 97] == 1:  
 res = s  
 **break  
 return** res

#### 左旋转字符串

汇编语言中有一种移位指令叫做循环左移（ROL），现在有个简单的任务，就是用字符串模拟这个指令的运算结果。对于一个给定的字符序列S，请你把其循环左移K位后的序列输出。例如，字符序列S=”abcXYZdef”,要求输出循环左移3位后的结果，即“XYZdefabc”。是不是很简单？OK，搞定它！

*# 左旋转字符串***def** left\_rotate\_string(data, n):  
 **if not** data:  
 **return  
 if** len(data) > 0 **and** n < len(data) **and** n > 0:  
 firstStart = 0  
 firstEnd = n - 1  
 secondStart = n  
 secondEnd = len(data) - 1  
  
 data = list(data)  
 reverse\_string(data, firstStart, firstEnd)  
 reverse\_string(data, secondStart, secondEnd)  
 reverse\_string(data, firstStart, secondEnd)  
 **return ''**.join(data)

#### 翻转单词顺序列

最近来了一个新员工Fish，每天早晨总是会拿着一本英文杂志，写些句子在本子上。同事Cat对Fish写的内容颇感兴趣，有一天他向Fish借来翻看，但却读不懂它的意思。例如，“student. a am I”。后来才意识到，这家伙原来把句子单词的顺序翻转了，正确的句子应该是“I am a student.”。Cat对一一的翻转这些单词顺序可不在行，你能帮助他么？

**def** reverse\_string(data, start, end):  
 **while** start < end:  
 data[start], data[end] = data[end], data[start]  
 start += 1  
 end -= 1  
  
**def** reverse\_sentence(data):  
 **if not** data:  
 **return** start, end = 0, len(data) - 1  
 data = list(data)  
 reverse\_string(data, start, end)  
  
 start = end = 0  
 **while** end < len(data):  
 **if** data[end] == **' ' or** end == len(data) - 1:  
 ends = end **if** end == len(data) - 1 **else** end - 1  
 reverse\_string(data, start, ends)  
 start = end = end + 1  
 **else**:  
 end += 1  
 **return ''**.join(data)

## 数组

#### 二维数组中的查找

在一个二维数组中，每一行都按照从左到右递增的顺序排序，每一列都按照从上到下递增的顺序排序。请完成一个函数，输入这样的一个二维数组和一个整数，判断数组中是否含有该整数。

**class** Solution:  
 *# array 二维列表* **def** Find(self, target, array):  
 *# write code here* **if not** array:  
 **return False** col = len(array[0]) - 1  
 cur\_col = 0  
 cur\_row = len(array) - 1  
 **while** cur\_col <= col **and** cur\_row >= 0:  
 **if** array[cur\_row][cur\_col] == target:  
 **return True  
 if** array[cur\_row][cur\_col] > target:  
 cur\_row -= 1  
 **if** array[cur\_row][cur\_col] < target:  
 cur\_col += 1  
 **return False**

#### 构建乘积数组

给定一个数组A[0,1,...,n-1],请构建一个数组B[0,1,...,n-1],其中B中的元素B[i]=A[0]\*A[1]\*...\*A[i-1]\*A[i+1]\*...\*A[n-1]。不能使用除法。

**def** multiply(arrayA):  
 head = [1]  
 tail = [1]  
 **for** i **in** range(len(arrayA) - 1):  
 head.append(arrayA[i] \* head[i])  
 tail.append(arrayA[-i - 1] \* tail[i])  
 **return** [head[j] \* tail[-j - 1] **for** j **in** range(len(head))]

#### 调整数组顺序使奇数位于偶数前面

输入一个整数数组，实现一个函数来调整该数组中数字的顺序，使得所有的奇数位于数组的前半部分，所有的偶数位于位于数组的后半部分，并保证奇数和奇数，偶数和偶数之间的相对位置不变。

**def** reorder\_odd\_even(arr, func):  
 **if not** arr **or** len(arr) <= 1:  
 **return** begin = 0  
 end = len(arr) - 1  
 **while** begin < end:  
 **while** begin < end **and not** func(arr[begin]):  
 begin += 1  
 **while** begin < end **and** func(arr[end]):  
 end -= 1  
 **if** begin < end:  
 arr[begin], arr[end] = arr[end], arr[begin]

#### 顺时针打印矩阵

输入一个矩阵，按照从外向里以顺时针的顺序依次打印出每一个数字，例如，如果输入如下矩阵： 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 则依次打印出数字1,2,3,4,8,12,16,15,14,13,9,5,6,7,11,10.

*# 解决方法1***def** print\_matrix(matrix):  
 row1 = 0  
 col1 = 0  
 row2 = len(matrix) - 1  
 col2 = len(matrix[0]) - 1  
 **while** row1 <= row2 **and** col1 <= col2:  
 printEdge(matrix, row1, col1, row2, col2)  
 row1 += 1  
 col1 += 1  
 row2 -= 1  
 col2 -= 1  
  
  
**def** printEdge(matrix, row1, col1, row2, col2):  
 **if** row1 == row2:  
 **for** i **in** col2:  
 print(col1[i])  
 **elif** col1 == col2:  
 **for** i **in** row2:  
 print(row1[i])  
 **else**:  
 curC = col1  
 curR = row1  
 **while** curC != col2:  
 print(matrix[row1][curC])  
 curC += 1  
 **while** curR != row2:  
 print(matrix[curR][col2])  
 curR += 1  
 **while** curC != col1:  
 print(matrix[row2][curC])  
 curC -= 1  
 **while** curR != row1:  
 print(matrix[curR][col1])  
 curR -= 1

#### 数组中的逆序对

在数组中的两个数字，如果前面一个数字大于后面的数字，则这两个数字组成一个逆序对。输入一个数组,求出这个数组中的逆序对的总数P。并将P对1000000007取模的结果输出。 即输出P%1000000007

**class** Solution:  
 **def** InversePairs(self, data):  
 self.count = 0  
  
 **def** MergeSort(lists):  
 **if** len(lists) <= 1:  
 **return** lists  
 num = len(lists) >> 1  
 left = MergeSort(lists[:num])  
 right = MergeSort(lists[num:])  
 R = L = 0  
 LL, LR = len(left), len(right)  
 result = []  
 **while** L < LL **and** R < LR:  
 **if** left[L] < right[R]:  
 result.append(left[L])  
 L += 1  
 **else**:  
 result.append(right[R])  
 R += 1  
 self.count += LL - L  
 result += right[R:]  
 result += left[L:]  
 **return** result  
 MergeSort(data)  
 **return** self.count

#### 数字在排序数组中出现的次数

统计一个数字在排序数组中出现的次数。

**def** find\_nums\_appear\_once(data):  
 **if not** data **or** len(data) < 2:  
 **return** res = 0  
 **for** num **in** data:  
 res ^= num  
 index = first = second = 0  
 **while** res & 1 == 0:  
 res = res >> 1  
 index += 1  
 **for** num **in** data:  
 **if** (num >> index) & 1:  
 first ^= num  
 **else**:  
 second ^= num  
 **return** (first, second)

## 链表

#### 从尾到头打印链表

输入一个链表，按链表值从尾到头的顺序返回一个ArrayList。

**class** Solution:  
 *# 返回从尾部到头部的列表值序列，例如[1,2,3]* **def** printListFromTailToHead(self, listNode):  
 *# write code here* **if not** listNode:  
 **return** []  
 **return** self.printListFromTailToHead(listNode.next) + [listNode.val]

#### 链表中环的入口结点

给一个链表，若其中包含环，请找出该链表的环的入口结点，否则，输出null。

**def** entry\_node\_of\_loop(head):  
 **if not** head:  
 **return** pAhead = head.next.next  
 pBehind = head.next  
 **while** pAhead != pBehind:  
 pAhead = pAhead.next.next  
 pBehind = pBehind.next  
 pAhead = head  
 **while** pAhead != pBehind:  
 pAhead = pAhead.next  
 pBehind = pBehind.next  
 **return** pAhead

#### 删除链表中重复的结点

在一个排序的链表中，存在重复的结点，请删除该链表中重复的结点，重复的结点不保留，返回链表头指针。 例如，链表1->2->3->3->4->4->5 处理后为 1->2->5

**class** Solution:  
 **def** deleteDuplication(self, pHead):  
 *# write code here* **if not** pHead:  
 **return** head = ListNode(**None**)  
 head.next = pHead  
 p = head  
 **while** pHead:  
 **while** pHead.next **and** pHead.next.val == pHead.val:  
 pHead = pHead.next  
 **if** p.next != pHead:  
 pHead = pHead.next  
 p.next = pHead  
 **else**:  
 p = pHead  
 pHead = pHead.next  
 **return** head.next

#### 链表中倒数第k个结点

输入一个链表，输出该链表中倒数第k个结点。

**def** find\_Kth\_to\_tail(head, k):  
 **if not** head **or not** k:  
 **return False** pAhead = pBehind = head  
 **for** i **in** range(k - 1):  
 **if** pAhead.next:  
 pAhead = pAhead.next  
 **else**:  
 **return False  
 while** pAhead.next:  
 pAhead = pAhead.next  
 pBehind = pBehind.next  
 **return** pBehind

#### 反转链表

输入一个链表，反转链表后，输出新链表的表头。

*# 非递归方式***def** reverse\_list(head):  
 **if not** head:  
 **return  
 if** head **and** head.next == **None**:  
 **return** head  
 prev = **None  
 while** head:  
 next = head.next  
 **if** next == **None**:  
 rhead = head  
 head.next = prev  
 prev = head  
 head = next  
 **return** rhead  
  
*# 递归方式***def** reverse\_list1(head):  
 **if not** head **or** head.next == **None**:  
 **return** head  
 **else**:  
 newhead = reverse\_list1(head.next)  
 head.next.next = head  
 head.next = **None  
 return** newhead

#### 合并两个排序的链表

输入两个单调递增的链表，输出两个链表合成后的链表，当然我们需要合成后的链表满足单调不减规则。

**def** merge(head1, head2):  
 **if** head1 == **None**:  
 **return** head2  
 **if** head2 == **None**:  
 **return** head1  
 **if** head1.data < head2.data:  
 mHead = head1  
 mHead.next = merge(head1.next, head2)  
 **else**:  
 mHead = head2  
 mHead.next = merge(head1, head2.next)  
 **return** mHead

#### 复杂链表的复制

输入一个复杂链表（每个节点中有节点值，以及两个指针，一个指向下一个节点，另一个特殊指针指向任意一个节点），返回结果为复制后复杂链表的head。（注意，输出结果中请不要返回参数中的节点引用，否则判题程序会直接返回空）

**def** clone\_nodes(head):  
 **while** head:  
 cloned = head  
 head, head.next = cloned.next, cloned  
  
  
**def** clone\_siblings(head):  
 **while** head:  
 cloned = head.next  
 **if** head.sibling:  
 cloned.sibling = head.sibling.next  
 head = cloned.next  
  
  
**def** reconnect\_nodes(head):  
 cloned\_head = cloned = head.next  
 head.next = cloned.next  
 head = head.next  
  
 **while** head:  
 cloned.next = head.next  
 cloned = cloned.next  
 head.next = cloned.next  
 head = head.next  
  
 **return** cloned\_head  
  
  
**def** clone(head):  
 **if not** head:  
 **return** clone(head)  
 clone\_siblings(head)  
 **return** reconnect\_nodes(head)

#### 链表成对调换

给定一个链表，结构如下： 1 --> 2 --> 3 --> 4 ，处理后输出 2 --> 1 --> 4 --> 3

**class** Solution:  
 **def** swapPairs(self, head):  
 **if** head **and** head.next:  
 next = head.next  
 head.next = self.swapPairs(next.next)  
 next.next = head  
 **return** next  
 **return** head

#### 两个链表的第一个公共结点

输入两个链表，找出它们的第一个公共结点。

**def** find\_first\_common\_node(pHead1, pHead2):  
 **if not** pHead1 **or not** pHead2:  
 **return None** pa = pHead1  
 pb = pHead2  
 **while**(pa != pb):  
 pa = pHead2 **if** pa **is None else** pa.next  
 pb = pHead1 **if** pb **is None else** pb.next  
 **return** pa

## 树

#### 重建二叉树

输入某二叉树的前序遍历和中序遍历的结果，请重建出该二叉树。假设输入的前序遍历和中序遍历的结果中都不含重复的数字。例如输入前序遍历序列{1,2,4,7,3,5,6,8}和中序遍历序列{4,7,2,1,5,3,8,6}，则重建二叉树并返回。

**class** Solution:  
 *# 返回构造的TreeNode根节点* **def** reConstructBinaryTree(self, pre, tin):  
 *# write code here* **if not** pre **or not** tin:  
 **return None** root = TreeNode(pre.pop(0))  
 index = tin.index(root.val)  
 root.left = self.reConstructBinaryTree(pre, tin[:index])  
 root.right = self.reConstructBinaryTree(pre, tin[index+1:])  
 **return** root

#### 二叉树的下一个结点

给定一个二叉树和其中的一个结点，请找出中序遍历顺序的下一个结点并且返回。注意，树中的结点不仅包含左右子结点，同时包含指向父结点的指针。

**class** Solution:  
 **def** GetNext(self, pNode):  
 *# write code here* dummy = pNode  
 **while** dummy.next:  
 dummy = dummy.next  
 self.result = []  
 self.midTraversal(dummy)  
 **return** self.result[self.result.index(pNode) + 1] **if** self.result.index(pNode) != len(self.result) - 1 **else None  
  
 def** midTraversal(self, root):  
 **if not** root: **return** self.midTraversal(root.left)  
 self.result.append(root)  
 self.midTraversal(root.right)

#### 对称的二叉树

请实现一个函数，用来判断一颗二叉树是不是对称的。注意，如果一个二叉树同此二叉树的镜像是同样的，定义其为对称的。

**def** Symmetrical(root1, root2):  
 **if not** root1 **and not** root2:  
 **return True  
 if** root1.data != root2.data:  
 **return False  
 return** Symmetrical(root1.left, root2.right) **and** Symmetrical(root1.right, root2.left)  
  
  
**def** isSymmetrical(root):  
 **return** Symmetrical(root, root)

#### 按之字形顺序打印二叉树

请实现一个函数按照之字形打印二叉树，即第一行按照从左到右的顺序打印，第二层按照从右至左的顺序打印，第三行按照从左到右的顺序打印，其他行以此类推。

**class** Solution:  
    **def** Print(self, pRoot):  
        **if not** pRoot:  
            **return** []  
        nodeStack=[pRoot]  
        result=[]  
        **while** nodeStack:  
            res = []  
            nextStack=[]  
            **for** i **in** nodeStack:  
                res.append(i.val)  
                **if** i.left:  
                    nextStack.append(i.left)  
                **if** i.right:  
                    nextStack.append(i.right)  
            nodeStack=nextStack  
            result.append(res)  
        returnResult=[]  
        **for** i,v **in** enumerate(result):  
            **if** i%2==0:  
                returnResult.append(v)  
            **else**:  
                returnResult.append(v[::-1])  
        **return** returnResult

#### 把二叉树打印成多行

从上到下按层打印二叉树，同一层结点从左至右输出。每一层输出一行。

**class** Solution:  
    *# 返回二维列表[[1,2],[4,5]]*    **def** Print(self, pRoot):  
        **if not** pRoot:  
            **return** []  
        nodeStack = [pRoot]  
        result = []  
        **while** nodeStack:  
            res = []  
            nextStack = []  
            **for** i **in** nodeStack:  
                res.append(i.val)  
                **if** i.left:  
                    nextStack.append(i.left)  
                **if** i.right:  
                    nextStack.append(i.right)  
            nodeStack = nextStack  
            result.append(res)  
        **return** result

#### 序列化二叉树

请实现两个函数，分别用来序列化和反序列化二叉树

*# 序列化二叉树***def** serialize(root):  
 **if not** root:  
 print(**'$,'**, end=**''**)  
 **return** print(root.data, end=**','**)  
 serialize(root.left)  
 serialize(root.right)  
  
*# 反序列化二叉树***def** deserialize(serialize):  
 **if** serialize:  
 serialize = iter(serialize.split(**','**))  
  
 **def** create\_tree():  
 data = next(serialize)  
 **if** data == **'$'**:  
 **return None** node = BSTnode(data=data)  
 node.left = create\_tree()  
 node.right = create\_tree()  
 **return** node  
 **return** create\_tree()

#### 二叉搜索树的第k个结点

给定一颗二叉搜索树，请找出其中的第k小的结点。例如， 5 / \ 3 7 /\ /\ 2 4 6 8 中，按结点数值大小顺序第三个结点的值为4。

**def** k\_node(root, k):  
 **if not** root **or not** k:  
 **return** res = []  
  
 **def** dfs(node):  
 **if** len(res) >= k **or not** node:  
 **return** dfs(node.left)  
 res.append(node)  
 dfs(node.right)  
 dfs(root)  
 **if** len(res) < k:  
 **return  
 return** res[k - 1]

#### 树的子结构

输入两棵二叉树A，B，判断B是不是A的子结构。（ps：我们约定空树不是任意一个树的子结构）

**def** equal(num1, num2):  
 **if** (num1 - num2) > -0.0000001 **and** (num1 - num2) < 0.0000001:  
 **return True  
 else**:  
 **return False  
  
  
def** does\_tree1\_have\_tree2(root1, root2):  
 **if not** root2:  
 **return True  
 if not** root1:  
 **return False  
 if** equal(root1.data, root2.data):  
 **return False  
 return** does\_tree1\_have\_tree2(root1.left, root2.left) **and** does\_tree1\_have\_tree2(root1.right, root2.right)  
  
  
**def** has\_subtree(root1, root2):  
 result = **False  
 if** root1 **and** root2:  
 **if** equal(root1.data, root2.data):  
 result = does\_tree1\_have\_tree2(root1, root2)  
 **if not** result:  
 result = has\_subtree(root1.left, root2)  
 **if not** result:  
 result = has\_subtree(root1.right, root2)  
 **return** result

#### 从上往下打印二叉树

从上往下打印出二叉树的每个节点，同层节点从左至右打印。

**def** print\_tree(root):  
 **if not** root:  
 **return** stack = []  
 stack.append(root)  
 **while** stack:  
 root = stack.pop(0)  
 print(root.data)  
 **if** root.left:  
 stack.append(root.left)  
 **if** root.right:  
 stack.append(root.right)

#### 二叉搜索树的后序遍历序列

输入一个整数数组，判断该数组是不是某二叉搜索树的后序遍历的结果。如果是则输出Yes,否则输出No。假设输入的数组的任意两个数字都互不相同。

**def** verify\_squence\_of\_bst(sequence):  
 **if** len(sequence) == 0:  
 **return False** root = sequence[-1]  
 i = 0  
 **for** node **in** sequence[:-1]:  
 **if** node > root:  
 **break** i += 1  
 **for** node **in** sequence[i:-1]:  
 **if** node < root:  
 **return False** left = **True  
 if** i > 0:  
 left = verify\_squence\_of\_bst(sequence[:i])  
 right = **True  
 if** i < len(sequence) - 2:  
 right = verify\_squence\_of\_bst(sequence[i:-1])  
 **return** left **and** right

#### 求二叉树最大深度

给定一个根节点root，求出二叉树最大深度

**class** Solution:  
 **def** maxDepth(self, root):  
 **if not** root:  
 **return** 0  
 **return** max(self.maxDepth(root.left), self.maxDepth(root.right)) + 1

#### 广度遍历与深度遍历二叉树

**class** Solution:  
 *# 广度遍历* **def** lookup(self, root):  
 **if not** root:  
 **return None** stack = [root]  
 **while** stack:  
 current = stack.pop(0)  
 print(current.val)  
 **if** current.left:  
 stack.append(current.left)  
 **if** current.right:  
 stack.append(current.right)  
  
 *# 深度遍历* **def** deep(self, root):  
 **if not** root:  
 **return None** print(root.val)  
 self.deep(root.left)  
 self.deep(root.right)

#### 二叉树的前序、中序、后序遍历

**class** TreeNode:  
 **def** \_\_init\_\_(self, x):  
 self.val = x  
 self.left = **None** self.right = **None  
  
def** preorder(root):  
 **if** root:  
 print(root.val)  
 preorder(root.left)  
 preorder(root.right)  
  
**def** inorder(root):  
 **if** root:  
 inorder(root.left)  
 print(root.val)  
 inorder(root.right)  
  
**def** postorder(root):  
 **if** root:  
 postorder(root.left)  
 postorder(root.right)  
 print(root.val)

#### 二叉树中和为某一值的路径

输入一颗二叉树和一个整数，打印出二叉树中结点值的和为输入整数的所有路径。路径定义为从树的根结点开始往下一直到叶结点所经过的结点形成一条路径。

**def** find\_path(root, target):  
 **if not** root:  
 **return** []  
 all\_path = []  
  
 **def** find\_path\_main(root, path=[], currentSum=0):  
 currentSum += root.data  
 path.append(root)  
 isLeaf = **not** root.left **and not** root.right  
 **if** currentSum == target **and** isLeaf:  
 all\_path.append([x.data **for** x **in** path])  
 **if** currentSum < target:  
 **if** root.left:  
 find\_path\_main(root.left, path, currentSum)  
 **if** root.right:  
 find\_path\_main(root.right, path, currentSum)  
 path.pop()  
  
 find\_path\_main(root)  
 **return** all\_path

## 栈和队列

#### 实现一个简单的栈结构

**class** Stack(object):  
 **def** \_\_init\_\_(self):  
 self.items = []  
  
 **def** \_\_iter\_\_(self):  
 **return** self.items.\_\_iter\_\_()  
  
 **def** pop(self):  
 **return** self.items.pop()  
  
 **def** top(self):  
 **if** len(self.items) > 0:  
 **return** self.items[len(self.items) - 1]  
  
 **def** push(self, item):  
 self.items.append(item)  
  
 **def** empty(self):  
 self.items = []  
  
 **def** size(self):  
 **return** len(self.items)

#### 用两个栈实现队列

用两个栈来实现一个队列，完成队列的Push和Pop操作。 队列中的元素为int类型。

**class** Solution:  
 **def** \_\_init\_\_(self):  
 self.in\_stack = []  
 self.out\_stack = []  
  
 **def** push(self, node):  
 self.in\_stack.append(node)  
 *# write code here* **def** pop(self):  
 **if** self.out\_stack:  
 **return** self.out\_stack.pop()  
 **else**:  
 **while** self.in\_stack:  
 self.out\_stack.append(self.in\_stack.pop())  
 **return** self.out\_stack.pop()

#### 栈的压入、弹出序列

输入两个整数序列，第一个序列表示栈的压入顺序，请判断第二个序列是否为该栈的弹出顺序。假设压入栈的所有数字均不相等。例如序列1,2,3,4,5是某栈的压入顺序，序列4,5,3,2,1是该压栈序列对应的一个弹出序列，但4,3,5,1,2就不可能是该压栈序列的弹出序列。（注意：这两个序列的长度是相等的）

**def** is\_pop\_order(pPush, pPOP):  
 stack = []  
 **while** pPOP:  
 **if** pPush **and** pPush[0] == pPOP[0]:  
 pPush.pop(0)  
 pPOP.pop(0)  
 **elif** stack **and** stack[-1] == pPOP[0]:  
 stack.pop()  
 pPOP.pop(0)  
 **elif** pPush:  
 stack.append(pPush.pop(0))  
 **else**:  
 **return False  
 return True**

## 查找和排序

#### 旋转数组的最小数字

把一个数组最开始的若干个元素搬到数组的末尾，我们称之为数组的旋转。 输入一个非减排序的数组的一个旋转，输出旋转数组的最小元素。 例如数组{3,4,5,1,2}为{1,2,3,4,5}的一个旋转，该数组的最小值为1。 NOTE：给出的所有元素都大于0，若数组大小为0，请返回0。

**class** Solution:  
 **def** minNumberInRotateArray(self, rotateArray):  
 *# write code here* **if not** rotateArray:  
 **return** 0  
 index = rotateArray[0]  
 **while** index >= rotateArray[-1]:  
 rotateArray.insert(0, rotateArray[-1])  
 rotateArray.pop()  
 **return** rotateArray[0]

## 递归和循环

#### 斐波那契数列

大家都知道斐波那契数列，现在要求输入一个整数n，请你输出斐波那契数列的第n项。

n<=39

**class** Solution:  
 **def** Fibonacci(self, n):  
 *# write code here* **if** n > 39:  
 **return** 0  
 res = [0,1]  
 **while** len(res) <= n:  
 res.append(res[-1] + res[-2])  
 **return** res[n]

#### 跳台阶

一只青蛙一次可以跳上1级台阶，也可以跳上2级。求该青蛙跳上一个n级的台阶总共有多少种跳法（先后次序不同算不同的结果）。

**class** Solution:  
 **def** jumpFloor(self, number):  
 *# write code here* res = [0,1,2]  
 **while** len(res) <= number:  
 res.append(res[-1] + res[-2])  
 **return** res[number]

#### 变态跳台阶

一只青蛙一次可以跳上1级台阶，也可以跳上2级……它也可以跳上n级。求该青蛙跳上一个n级的台阶总共有多少种跳法。

**class** Solution:  
 **def** jumpFloorII(self, number):  
 *# write code here* **return** 1<<(number-1)

#### 矩形覆盖

我们可以用2\*1的小矩形横着或者竖着去覆盖更大的矩形。请问用n个2\*1的小矩形无重叠地覆盖一个2\*n的大矩形，总共有多少种方法？

**class** Solution:  
 **def** rectCover(self, number):  
 *# write code here* res = [0,1,2]  
 **while** len(res) <= number:  
 res.append(res[-1] + res[-2])

## 位运算

#### 二进制中1的个数

输入一个整数，输出该数二进制表示中1的个数。其中负数用补码表示。

**class** Solution:  
 **def** NumberOf1(self, n):  
 *# write code here* **return** bin(n).count(**'1'**) **if** n >= 0 **else** bin(2\*\*32 + n).count(**'1'**)

#### 不用加减乘除做加法

写一个函数，求两个整数之和，要求在函数体内不得使用+、-、\*、/四则运算符号。

**def** add\_num(num1, num2):  
 **if not** num1 **or not** num2:  
 **return  
 return** sum([num1, num2])

## 时间效率

#### 数组中出现次数超过一半的数字

数组中有一个数字出现的次数超过数组长度的一半，请找出这个数字。例如输入一个长度为9的数组{1,2,3,2,2,2,5,4,2}。由于数字2在数组中出现了5次，超过数组长度的一半，因此输出2。如果不存在则输出0。

**def** more\_than\_half\_num(nums):  
 **if not** nums:  
 **return 0**

result, times = nums[0], 1  
 **for** i **in** range(1, len(nums)):  
 **if** times == 0:  
 result = nums[i]  
 times = 1  
 **else**:  
 times += 1 **if** nums[i] == result **else** -1  
 **return** result **if** nums.count(result) > len(nums) / 2 **else 0**

#### 最小的K个数

输入n个整数，找出其中最小的K个数。例如输入4,5,1,6,2,7,3,8这8个数字，则最小的4个数字是1,2,3,4,。

*# 使用堆***import** heapq  
  
  
**def** get\_least\_numbers(nums, k):  
 heaps = []  
 ret = []  
 **for** num **in** nums:  
 heapq.heappush(heaps, num)  
 **if** k > len(heaps):  
 **return** []  
 **for** i **in** range(k):  
 ret.append(heapq.heappop(heaps))  
 **return** ret

#### 连续子数组的最大和

HZ偶尔会拿些专业问题来忽悠那些非计算机专业的同学。今天测试组开完会后,他又发话了:在古老的一维模式识别中,常常需要计算连续子向量的最大和,当向量全为正数的时候,问题很好解决。但是,如果向量中包含负数,是否应该包含某个负数,并期望旁边的正数会弥补它呢？例如:{6,-3,-2,7,-15,1,2,2},连续子向量的最大和为8(从第0个开始,到第3个为止)。你会不会被他忽悠住？(子向量的长度至少是1)

**class** Solution:  
 **def** FindGreatestSumOfSubArray(self, array):  
 *# write code here* **if not** array:  
 **return  
 if** len(array) == 1:  
 **return** array[0]  
 current = c\_sum = array[0]  
 **for** i **in** range(1, len(array)):  
 c\_sum = max(c\_sum + array[i], array[i])  
 current = max(current, c\_sum)  
 **return** current

#### 把数组排成最小的数

输入一个正整数数组，把数组里所有数字拼接起来排成一个数，打印能拼接出的所有数字中最小的一个。例如输入数组{3，32，321}，则打印出这三个数字能排成的最小数字为321323。

**def** print\_min\_number(numbers):  
 **if not** numbers:  
 **return  
 from** functools **import** cmp\_to\_key  
 key = cmp\_to\_key(**lambda** x, y: int(x + y) - int(y + x))  
 res = **''**.join(sorted(map(str, numbers), key=key)).lstrip(**'0'**)  
 **return** res **or '0'**

#### 二分查找

**def** binarySearch(arr, target):  
 **if not** arr:  
 **return False** low, high = 0, len(arr) - 1  
 **while** low < high:  
 mid = (low + high) // 2  
 **if** arr[mid] > target:  
 high = mid  
 **elif** arr[mid] < target:  
 low = mid + 1  
 **else**:  
 **return** mid  
 **return** low **if** arr[low] == target **else False**

#### 丑数

把只包含质因子2、3和5的数称作丑数（Ugly Number）。例如6、8都是丑数，但14不是，因为它包含因子7。 习惯上我们把1当做是第一个丑数。求按从小到大的顺序的第N个丑数。

**def** get\_ugly\_number(index):  
 **if not** index:  
 **return** 0  
 ugly\_number = [1] \* index  
 next\_index = 1  
 m2 = m3 = m5 = 1  
 **while** next\_index < index:  
 ugly\_number[next\_index] = min(m2 \* 2, m3 \* 3, m5 \* 5)  
 **while** m2 \* 2 <= ugly\_number[next\_index]:  
 m2 += 1  
 **while** m3 \* 3 <= ugly\_number[next\_index]:  
 m3 += 1  
 **while** m5 \* 5 <= ugly\_number[next\_index]:  
 m5 += 1  
 next\_index += 1  
 **return** ugly\_number[next\_index - 1]

## 其它编程题

#### 写一个算法判断一个英文单词的所有字母是否全都不同（不区分大小写）。

**def** is\_single(string):  
 **if not** string:  
 **return False** res\_dict = {}  
 **for** i **in** string:  
 **if** i **not in** res\_dict:  
 res\_dict[i] = 1  
 **else**:  
 **return False  
 return True**

#### 使用函数实现一个字符串的全排列

**import** itertools  
  
**def** all\_permutations(string):  
 **if not** string:  
 **return None  
 for** i **in** itertools.permutations(string,len(string)):  
 print(**''**.join(i))

#### 找零问题

*# need\_change 为需要找零的金额，  
# currency\_list 为该国货币的面值列表，  
# num\_list 为需要找零的最少货币数目, num\_list的长度至少为(need\_change+1)  
# used\_list 为需要找零的最少货币数目, 长度与num\_list相同***def** giveChange(need\_change, currency\_list, num\_list, used\_list):  
 **for** change **in** range(need\_change+1): *#从0开始计算最少需要的货币数* **for** currency **in** currency\_list: *#遍历每一种货币* **if** (change-currency >= 0) **and** (num\_list[change-currency]+1<=num\_list[change]): *#计算最少货币需求数* num\_list[change] = num\_list[change-currency] + 1  
 used\_list[change] = currency *#记录消耗的货币* **return***# 返回需要的货币***def** showChange(need\_change, used\_list):  
 give\_list = []  
 **while** need\_change > 0:  
 give\_list.append(used\_list[need\_change])  
 need\_change -= used\_list[need\_change]  
 give\_list.sort() *#排序* **return** give\_list  
  
**def** main():  
 need\_change = 64 *#需要找零的钱数* currency\_list = [1,5,10,21,25] *# 该国的货币面值列表* num\_list = list(range(need\_change+1)) *#初始化num\_list为0到need\_change,共(need\_change+1)个数* used\_list = list(range(need\_change+1)) *#初始化used\_list为0到need\_change,共(need\_change+1)个数* giveChange(need\_change, currency\_list, num\_list, used\_list)  
 print(**"%d 需要 %d 个货币来找零"**%(need\_change, num\_list[need\_change]))  
 give\_list = showChange(need\_change, used\_list)  
 print(give\_list)  
  
main()

#### 统计一篇英文文章中单词个数

**def** getText():  
 txt=open(**"article.txt"**,**"r"**).read()  
 txt=txt.lower()  
 **for** ch **in '~!@#$%^&\*()\_+"{}[]|?.<>?'**:  
  
 txt=txt.replace(ch,**""**)  
 **return** txt  
hamletTxt=getText()  
words=hamletTxt.split()  
counts={}  
**for** word **in** words:  
 counts[word]=counts.get(word,0)+1  
items=list(counts.items())  
items.sort(key=**lambda** x:x[1],reverse=**True**)  
**for** i **in** range (10): *###这里就让它输出10个看看得了* word,count=items[i]  
 print(**"{0:<10}{1:>5}"**.format(word,count))

#### 给你A,B两个文件，各存放50亿条URL，每条URL占用64字节，内存限制是16G，让你找出A,B文件共同的URL。如果是三个乃至n个文件呢？

分析：

50亿条64字节长度的URL总共需要内存空间：

50亿 \* 64 字节 = 3200亿字节 ÷ 1024 =3.18亿KB ÷ 1024 ≈ 31万MB ≈ 298GB

总共需要298GB内存才可以存放50亿条URL地址， 需求量太大，成本太高，直接考虑布隆过滤器

计算：

按照失误率万分之一设计布隆过滤器

P 为预期失误率 即0.0001

n 为需要在布隆过滤器中存放的URL数量

m 为需要多少个二进制位 （bit）

k 为需要多少个哈希函数

公式计算保留小数点后两位

计算后得到:

m为95937500000 bit 即约为11.19 GB， 内存限制为16GB

k 为13.24 但哈希函数必为整数，向上取整为14个哈希函数

通过计算后的内存空间（16GB = 137438953472 bit）和哈希函数数量（14）重新计算实际失误率P

P=〖( 1-e^(-(n\*k)/m))〗^k = 〖( 1-0.49)〗^14 = 0.000003

得到实际失误率约为百万分之三

## 排序算法

#### 插入排序

描述  
插入排序的基本操作就是将一个数据插入到已经排好序的有序数据中，从而得到一个新的、个数加一的有序数据，算法适用于少量数据的排序，时间复杂度为O(n^2)。是稳定的排序方法。插入算法把要排序的数组分成两部分：第一部分包含了这个数组的所有元素，但将最后一个元素除外（让数组多一个空间才有插入的位置），而第二部分就只包含这一个元素（即待插入元素）。在第一部分排序完成后，再将这个最后元素插入到已排好序的第一部分中。  
  
代码实现  
*# 插入排序***def** insert\_sort(lists):  
 *# 将list的长度赋值给count* count = len(lists)  
 *# 从1开始遍历list* **for** i **in** range(1, count):  
 *# 将当前遍历的项的值赋值给key* key = lists[i]  
 *# 将i-1的值赋值给j* j = i - 1  
 *# 使用while循环，当j>=0时执行，即j的位置在list中执行* **while** j >= 0:  
 *# 第一次进入时，因i=1，j=0，j为列表第一位，i为列表第二位，如果第一位大于第二位，让这两个数值交换位置，* **if** lists[j] > key:  
 lists[j + 1], lists[j] = lists[j], key  
 *#交换位置后让j-=1，等于是交换后的数值再与其前一位做判断* j -= 1  
 *# 将最后的lists返回出去* **return** lists

#### 冒泡排序

描述  
它重复地走访过要排序的数列，一次比较两个元素，如果他们的顺序错误就把他们交换过来。走访数列的工作是重复地进行直到没有再需要交换，也就是说该数列已经排序完成。  
  
代码实现  
*# 冒泡排序***def** bubble\_sort(lists):  
 *# 将lists列表长度赋值给count* count = len(lists)  
 *# 遍历lists中每一个位置* **for** i **in** range(0, count):  
 *# 遍历当前项的下一项* **for** j **in** range(i + 1, count):  
 *# 如果当前项的值小于下一项的值，则交换位置* **if** lists[i] > lists[j]:  
 lists[i], lists[j] = lists[j], lists[i]  
 *# 将排序后的列表返回* **return** lists

#### 快速排序

描述  
通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。  
  
代码实现  
  
  
*# 第一种实现方式***def** quicksort(arr, left, right):  
 *# 只有left < right 排序* **if** left < right:  
 pivot\_index = partition(arr, left, right)  
 quicksort(arr, left, pivot\_index - 1)  
 quicksort(arr, pivot\_index + 1, right)  
  
  
**def** partition(arr, left, right):  
 *"""找到基准位置, 并返回"""* pivot\_index = left  
 pivot = arr[left]  
  
 **for** i **in** range(left + 1, right + 1):  
 **if** arr[i] < pivot:  
 *# 如果此处索引的值小于基准值, 基准值的位置后移一位  
 # 并将后移一位的值和这个值交换, 让基准位置及之前的始终小于基准值* pivot\_index += 1  
 **if** pivot\_index != i:  
 arr[pivot\_index], arr[i] = arr[i], arr[pivot\_index]  
 *# 将基准值移动到正确的位置* arr[left], arr[pivot\_index] = arr[pivot\_index], arr[left]  
 **return** pivot\_index  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 arr = [1, 3, 2, 4, 5, 7, 6, 8]  
 print(arr)  
 quicksort(arr, 0, len(arr) - 1)  
 print(arr)  
  
  
*# 第二种实现方式***def** quick\_sort(list):  
 **if not** list:  
 **return** []  
 pivot = list[0]  
 less = [x **for** x **in** list[1:] **if** x <= pivot]  
 bigger = [x **for** x **in** list[1:] **if** x >= pivot]  
 **return** quick\_sort(less) + [pivot] + quick\_sort(bigger)  
  
  
a = [1, 3, 2, 4, 5, 7, 6, 8]  
print(quick\_sort(a))  
  
  
*# 第三种实现方式***def** quick\_sort(list):  
 **return** [] **if** list == [] **else** quick\_sort([x **for** x **in** list[1:] **if** x <= list[0]]) \  
 + [list[0]] + quick\_sort([x **for** x **in** list[1:] **if** x >= list[0]])  
  
  
a = [1, 3, 2, 4, 5, 7, 6, 8]  
print(quick\_sort(a))

#### 归并排序

描述  
归并排序是建立在归并操作上的一种有效的排序算法, 该算法是采用分治法（Divide **and** Conquer）的一个非常典型的应用。将已有序的子序列合并，得到完全有序的序列；即先使每个子序列有序，再使子序列段间有序。若将两个有序表合并成一个有序表，称为二路归并。  
  
归并过程为：比较a[i]和a[j]的大小，若a[i]≤a[j]，则将第一个有序表中的元素a[i]复制到r[k]中，并令i和k分别加上1；否则将第二个有序表中的元素a[j]复制到r[k]中，并令j和k分别加上1，如此循环下去，直到其中一个有序表取完，然后再将另一个有序表中剩余的元素复制到r中从下标k到下标t的单元。归并排序的算法我们通常用递归实现，先把待排序区间[s, t]以中点二分，接着把左边子区间排序，再把右边子区间排序，最后把左区间和右区间用一次归并操作合并成有序的区间[s, t]。  
  
代码实现  
  
**def** merge(left, right):  
 i, j = 0, 0  
 result = []  
 **while** i < len(left) **and** j < len(right):  
 **if** left[i] <= right[j]:  
 result.append(left[i])  
 i += 1  
 **else**:  
 result.append(right[j])  
 j += 1  
 result += left[i:]  
 result += right[j:]  
 **return** result  
  
  
**def** merge\_sort(lists):  
 *# 归并排序* **if** len(lists) <= 1:  
 **return** lists  
 num = len(lists) / 2  
 left = merge\_sort(lists[:num])  
 right = merge\_sort(lists[num:])  
 **return** merge(left, right)

#### 堆排序

描述  
堆排序(Heapsort)  
是指利用堆积树（堆）这种数据结构所设计的一种排序算法，它是选择排序的一种。可以利用数组的特点快速定位指定索引的元素。堆分为大根堆和小根堆，是完全二叉树。大根堆的要求是每个节点的值都不大于其父节点的值，即A[PARENT[i]] >= A[i]。在数组的非降序排序中，需要使用的就是大根堆，因为根据大根堆的要求可知，最大的值一定在堆顶。  
  
代码实现  
  
  
**def** adjust\_heap(lists, i, size):  
 lchild = 2 \* i + 1  
 rchild = 2 \* i + 2  
 max = i  
 **if** i < size / 2:  
 **if** lchild < size **and** lists[lchild] > lists[max]:  
 max = lchild  
 **if** rchild < size **and** lists[rchild] > lists[max]:  
 max = rchild  
 **if** max != i:  
 lists[max], lists[i] = lists[i], lists[max]  
 adjust\_heap(lists, max, size)  
  
  
**def** build\_heap(lists, size):  
 **for** i **in** range(0, (size / 2))[::-1]:  
 adjust\_heap(lists, i, size)  
  
  
**def** heap\_sort(lists):  
 size = len(lists)  
 build\_heap(lists, size)  
 **for** i **in** range(0, size)[::-1]:  
 lists[0], lists[i] = lists[i], lists[0]  
 adjust\_heap(lists, 0, i)

# 项目相关问题

## 博客系统

#### 对于这个的架构能够阐述下吗？

这个系统是配套的数据分析平台的沟通平台，为实现用户和管理员的沟通做的， 主要是获取用户的信息和发表的评论的内容，然后针对部分的评论进行回复，便于增强和客户的关系。主要使用到django, mysql, bootstrap对于评论的回复还使用到富文本编辑器（kindeditor），和富文本的图片上传等

#### 在项目中担任的角色？实现的模块有哪些？

主要是模型类的设计，还有用户评论的内容展示和跟帖的用户的展示

#### 项目的后台管理是怎样实现的？

主要就是django自带的后台管理功能，不过自己定制了写功能，比如在管理中的排序方式等

#### 你部署过项目没？怎样部署的？

这个可以根据自己情况灵活回答，假如掌握的不好可以回答没有部署过，一般都是组长或者专门人员进行部署的。如果回答部署，就说一下啊部署的方法：  
部署使用的是： Nginx + uWSGI + redis + mysql  
分为两步走：  
1.uWSGI + 项目能够实现  
可能会问到WSGI / UWSGI / uWSGI的区别，要区分好网管、协议、服务器  
在项目的根目录下创建配置文件进行配置。  
  
然后运行配置文件，查看项目能否正常启动，能够启动说明这一步没问题，进行下一步，如果有问题主要查看配置文件的编写是否正确（前提是项目能够正常的运行）  
2.  
配置nginx  
在 / etc / nginx / conf.d / 下创建test.conf  
server  
{  
 listen 80; *# 监听80端口* access\_log / var / log / nginx / access.log; *# 日志文件位置* error\_log / var / log / nginx / error.log;  
 location / { *# 访问/时* include uwsgi\_params; *# 加载uwsgi模块* proxy\_pass http://127.0.0.1:8000; *# 将连接转到该IP* }  
 error\_page 500 502 503 504 / 50x.html;  
 location = /50x.html  
 {  
 root html;  
 }  
 location/static / {  
 alias /var/www/static/; *# 访问/static静态文件时直接访问到,目录要由读写的权限*}  
然后修改uWSGI的配置文件  
  
接下来把项目下的所有静态文件导入到服务器指定的目录中， 如果有视频或者图片较多，并且不会轻易修改，可以使用第三方的CDN, 比如七牛云等

#### 这个平台的缓存你了解吗？是怎样实现的？

可能会问cookie和session 的区别。要注意复习下  
这个缓存主要是两部分： 用户的缓存和常用页面和数据的缓存  
1. 对于用户信息的缓存使用的是django自带的sessionMiddle的缓存，只要在setting的Middleware中启用并分给权重值就可以了  
2.对于不经常改动的页面和数据的缓存使用的是redis的缓存，这个就要安装django-redis并且在Middleware中进行配置和自己写一些缓存的信息。

#### 在这个项目中遇到过什么问题？怎么解决的？

可以结合在开发中遇到的路径问题或者查询数据的问题进行回答

## 数据分析平台

#### 这平台的数据来源能够阐述一下吗？

这个平台的数据来源是京东商城的手机，要自己抓取数据源写入到mongodb中，通过scrapy\_redis实现成分布式，并通过engine和 scheduler实现url的去重;为避免反爬虫，使用中间件UserAgentMiddleware、ProxyMiddleware, 并限制抓取的频率download\_delay=2.

#### django是不支持非关系型数据库的，怎么解决这个问题的？遇到过什么问题？

Django支持的数据库有：SQlite3, Oracle, Mysql、PostgreSql, 并不支持非关系型数据库，所以为了解决这个问题使用的是Mongoengine, 一个第三方的mongodb的引擎。当时有几个备选的方案：  
 1. django-nonrel, 能够支持ORM,支持部分的Nosql数据库, 包括mongodb,  
但是缺点实在太多， 比如对于ORM的许多特性会抛出NotImplemented（异常对象的error）的异常;另外不能使用多表继承，只能从anstract类继承。  
 2. pymongo, 也能够实现,数据库的查询增加，删除等操作，但是要自己写很多的Mongodb的查询语法，容易出粗，并且不能使用Django的亮点ORM了，后期维护难度较大。  
Mongoengine, 支持ORM, 使用起来和django一样， 都可以定义模型类，也有自己的字段类型，比如stringFiled相当于MysqL的CharField等等， 同时也可以支持模型类的继承。

#### 这个平台的主要业务有什么？

* 1. 京东商城的数据抓取

1. 和前端对接，沟通要抓取的数据字段都有哪些， 封装的数据结构是怎样的json。  
2. 对京东商城的整站的入口进行分析，找到主入口和要抓取数据的API进行数据的抓取和解析。例如价格的api的查找和拼接，对特定参数的去除验证； 商品评论的接口的参数的确认和自定义，实现不同类型的数据的抓取和解析。  
3. 使用中间件规避常规的反爬虫措施：设置代理的Ip池， User-Agent; 对于抓取的频率进行限制， 避免抓取过快被封号。  
4. 对于存储的数据通过分布式写入到Mongodb 中，使用scrapy-redis实现分布式。  
5. 对于url去重是通过调度器中redis的set实现。对于url使用爬虫+request+fingerPinter的hashlib加密实现去重，如果调度器的队列里已经有请的url 的指纹就放过不再请求，如果没有就添加进去。

* 1. 分析展示平台的呈现

1.使用Mongoengine定义模型类和已经存在的mongodb数据库进行对接。  
2.怎么使用非关系型数据的。  
主要是在settings.py中进行配置  
3.对于数据的展示使用的是什么？Matlab吗？  
使用的是eacharts, 对于matlab在学习中，会在后续的业务中结合numpy和pandas使用.  
4.用户的登陆注册模块实现。  
是使用的django - engine的自带的mongo - auth，实现的，能够很好的使用django的User模型的函数  
5.日志系统的实现  
日志主要是使用logginge模块，然后在配置文件中指定存储位置和存储的格式  
6.缓存的实现  
缓存有两种方案：  
1.redis做的缓存，缓存的常用的不经常变化的数据和缓存了部分的模板  
2.基于mongoengine缓存的session, 主要是在配置文件中进行配置。  
SESSION\_ENGINE = **'django\_mongoengine.sessions'**7.商品销量top10的柱状图展示。分别点击每个商品的链接实现商品的购买。这个功能能够发现用户对哪些手机最感兴趣  
8.对商品的评论详情的饼图的呈现。能够直观的看到用户对此商品的直观感受，对于生产厂商能够提供后续改进的需求，能够对卖家平台铺货有直观的指导  
9.对同一款手机的不同颜色进行柱状图展示，能够直观的看到同款手机不同颜色的销量情况。  
10.对手机厂商的销量进行统计， 能够直观看到小米和华为的销量最大  
11.在抓取数据过程中有没有遇到编码问题？怎么解决的？  
使用的是python3.5但是也遇到编码问题了， 主要是特殊字符的， 比如笑脸的或者ascii码的，对于笑脸的就直接剔除掉了， 对于ascii的进行encode转换为unicode或者utf - 8.

#### 你主要负责什么模块？

主要负责数据源的采集也就是爬虫这块，还有就是用户的登陆注册模块的实现（可以结合自己比较熟悉的模块来说）

#### 在开发过程中遇到过哪些问题？怎么解决的？

1. 关于分布式的实现和url的去重  
 分布式是基于scrapy-redis做的，原来使用的也是基于redis的去重，但是实在内存做的，后来发现数据量大后并不理想，就使用了bloom filter实现了（结合url去重来说）  
2. 关于非关系型数据库的实现。三种方式中mongengine引擎的确定，主要是耗时间。

#### 爬虫每天能够抓取多少的数据？

每天能够入库的数据10万，主要是考虑到反爬虫抓取的频率设置的很低

## 电商网站项目

#### 说说你最近做的这个项目的背景，简单的介绍一下你这个项目?

1. 背景

电商项目的背景一般是由市场推动的，比如行业竞争或者经营方式的改变(营销理念)。竞争的形态也发生了巨大的变化，从以产品、价格为主的竞争转向以服务为主的竞争，服务成为主导竞争格局的重要因素。渠道作为企业完成客户沟通、产品/服务交换过程以及实现价值、产生效益的重要载体，发挥了采集、传达 客户和竞争对手等市场信息，为买卖双方供便利，协调供需矛盾，为客户供合适的产品与服务，向客户传递产品/服务信息，实现营销/服务目标等重要的功能。 XXX商城之前主要以实体店为主，进行批发与零售。业务也相对比较传统，为了升业务绩效，增强客户满意度和粘性，另一方面，也为基于互联网的商务模式创新奠定基础。针对上述行业环境变化和业务战 略目标，xxx商城网上终端预约销售基础上，即将启动网上商城建设项目，用于建立网上终端、营销案在线销售及相关辅助功能，包含商品管理、订单管理、类目管理、客户管理、合作商管理、客服管理、购物平台、内容管理等，很大程度上分担了人工的压力，对高客户服务效率和客户满意度能够起到较好的作用。 基于此，XXX 公司 出建设网上商城建设项目工程。

1. 项目介绍

xxx 商城项目打造的是“社区+电商”的模式，用户不只是在社区中有自己的圈子，还可以将电商加入到社区中，整个电商网站实现的功能非常之多，采用分布式的架构设计，包括后台管理、前台系统、订单系统、 单点登录系统、搜索系统、会员系统等。  
该项目是自己公司的产品，我们公司负责整个网站的运营，属于 B2C 平台;  
系统的用途，主要是 供 B2C 的平台，其中自营商品也有商家入住，类似天猫。  
系统架构，采用分布式的系统架构，其中前台系统和单点登录系统采用了集群的方式部署。  
如果使用Django实现，则可以使用服务端渲染技术，并且使用Django自带的后台管理系统。  
如果使用Tornado实现，则可以前后端分离，使用Tornado来实现Restful接口，接口对跨域进行支持。  
消息队列使用Celery。  
部署方面，采用了Nginx+uwsgi+Django的模式，其中nginx的作用一方面是做反向代理、负载均衡、另一方面是做图片等静态资源的服务器。

1. 系统介绍

后台管理系统:管理商品、订单、类目、商品规格属性、用户管理以及内容发布等功能。  
前台系统:用户可以在前台系统中进行注册、登录、浏览商品、首页、下单等操作。  
会员系统:用户可以在该系统中查询已下的订单、收藏的商品、我的优惠券、团购等信息。  
订单系统: 供下单、查询订单、修改订单状态、定时处理订单。  
搜索系统: 供商品的搜索功能。  
会员登录系统:为多个系统之间 供用户登录凭证以及查询登录用户的信息。

1. 提供的功能

商品管理模块:其中包括品牌管理，属性管理商品录入/上下架管理，商品添加审核，静态页面发布，使用Django Admin模块实现。  
订单模块：创建订单功能、查看订单列表、根据订单id查询订单的详细信息、订单修改、订单取消、订单状态、订单评价等功能的实现。  
商品前台首页:其中主要负责首页商品列表筛选，首页上动态展示筛选条件，点击每一个筛选条件下面的商品列表要做联动  
单品页面:采用缓存来实现页面快速加载，展示商品详情信息和商品购买，降低服务器负载。  
交订单页面: 交用户的订单信息, 处理并发问题。  
个人中心，包括用户的登录，个人信息的管理，收货地址的管理，用户所下的订单的管理 购物车:把购物车的信息存在 cookie 里面管理

#### 项目开发所需人员结构

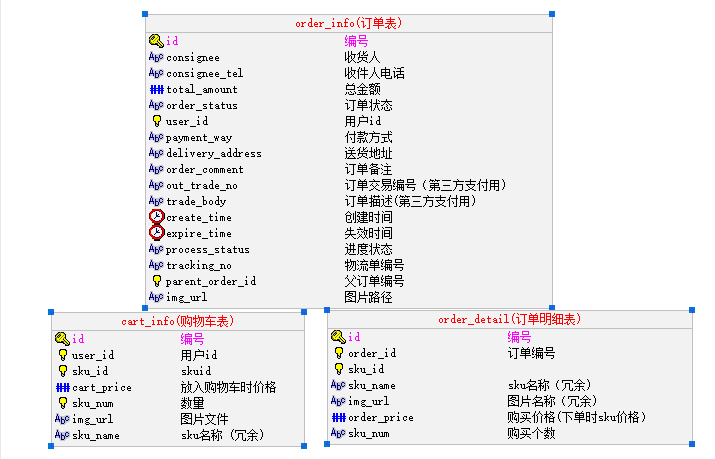
产品经理：3人，确定需求以及给出产品原型图。  
项目经理：1人，项目管理。  
前端团队：5人，根据产品经理给出的原型制作静态页面。  
后端团队：20人，实现产品功能。  
测试团队：5人，测试所有的功能。  
运维团队：3人，项目的发布以及维护。

#### 你承担这个项目的哪些核心模块?

在项目中主要负责相关系统的开发，主要有:  
后台管理系统，主要实现商品管理、商品规格参数管理、订单管理、会员管理等、CMS(内容管理系统) 等，并且提供了跨域支持;  
前台系统，主要是面向用户访问，使用 Requests 和后台系统接口做交互，并且该系统在部署上采用 集群的方式;  
会员登录系统，主要是 供集中用户登录凭证的集中解决方案， 供和用户信息相关的接口，比如说 用户注册、查询等接口。  
订单系统，主要是 供和订单相关的业务接口，在订单系统了做了严格的数据校验以及高并发写的支持(这里可以说使用队列实现)，并且使用了 Celery 定时任务实现对订单的定时扫描，比如说关闭超时未付款的订单;  
搜索系统，主要是 供商品的搜索，采用Whoosh实现，采用了 MQ 机制保证了商品数据 可以及时同步到 Whoosh 中;  
会员系统，主要是维护用户的信息，已购买订单、优惠券、系统消息、修改密码、绑定手机等功能; 7) 缓存，主要是用 Redis 实现，并且对 Redis 做了集群来保证 Redis 服务的高可用。  
支付系统，主要是负责订单的支付、对账等功能，主要是对接了支付宝的接口;

#### 负责模块功能实现步骤

购物车 + 订单 + 表设计  
1.分析购物车数据应该放到哪里：redis + mysql  
 a) 登录情况：redis + mysql  
 b) 非登录情况：cookie中  
 c) 下订单的时候合并购物车  
2.订单是从购物车中得来  
 a) 在购物车中通过复选框的形式来确定要下单的商品。  
 b) 被选中的商品在缓存中存储一份 - ---即下订单的商品。



#### 商品管理模块

主要掌握的核心是品牌的添加 图片服务器的搭建， 上传图片到图片服务器 表单的验证，离开焦点的验证，点击完成时的验证，后台服务器的 ajax 验证，表单规范 防止表单的二次提交 编辑时不需要修改品牌名，区别 readOnly 和 disabled 删除时的二次确认。

#### 你们的系统中有多少SKU？ 有多少SPU？

1. SKU(Stock Keeping Unit)库存量单元 --- SKU是商品下的一个分类属性（商品下一个颜色或者尺码）  
2. SPU(Standard Product Unit)标准化产品单元 --- SPU是一个商品编码  
一个SKU可以对应多个SPU，简单的说： SPU就是一个iPhone6s, SKU就是银色iPhone6s、粉色iPhone6s  
  
根据电商的业务类型不同SKU在1万-100万之间，SPU的值=SKU/5

#### 项目中哪些功能模块涉及了大数据量访问?你是如何解决的?

系统前台是互联网上的用户访问的，会有大量用户来访问。 假定有 1w 个人打开你的网站来订商品，问你如何解决并发问题(可扩展到任何高并发网站要考虑的并发读写问题) 问题，1w 个人来访问，商品没出去前要保证大家都能看到有商品，不可能一个人在看到商品的时候别人就 不能看了。到底谁能抢到，那得看这个人的“运气”(网络快慢等) 其次考虑的问题，并发，1w 个人同时点击购买，到底谁能成交?总共只有一张商品。  
可以使用redis队列来解决这个问题，谁在队列的最前面，谁成交。

#### 你做完这个项目后有什么收获?

首先，在数据库方面，我现在是真正地体会到数据库的设计真的是一个程序或软件设计的重要和根基。因为数据库怎么设计，直接影响到一个程序或软件的功能的实现方法、性能和维护。由于我做的模块是要对数据库的数据进行计算和操作的，所以我对数据库的设计对程序的影响是深有体会，就是因为我们的数  
据库设计得不好，搞得我在对数据库中的数据进行获取和计算利润、总金时，非常困难，而且运行效率低，时间和空间的复杂也高，而且维护起来很困难，过了不久，即使自己有注释，但是也要认真地看自己的代码才能明白自己当初的想法和做法。加上师兄的解说，让我对数据库的重要的认识更深一层，数据库的设计真的是重中之重。  
  
其次，就是分工的问题。虽然这次的项目我们没有在四人选出一个组长，但是，由于我跟其他人都比较熟，也有他们的号码，然后我就像一个小组长一样，也是我对他们进行了分工。俗话也说，分工合作，分好了工，才能合作。但是这次项目，我们的分工却非常糟糕，我们在分工之前分好了模块，每个模块实现什么功能，每个人负责哪些模块。本以为我们的分工是明确的，后来才发现，我们的分工是那么的一踏糊涂，一些功能上紧密相连的模块分给了两个人来完成，使两个人都感到迷惘，不知道自己要做什么，因为两个人做的东西差不多。我做的，他也在做，那我是否要继续做下去?总是有这样的疑问。从而导致了重复工作，浪费时间和精力，并打击了队员的激情，因为自己辛辛苦苦写的代码，最后可能没有派上用场。我也知道，没有一点经验的我犯这样的错是在所难免，我也不过多地怪责自己，吸取这次的教训就好。分工也是一门学问。  
  
再者，就是命名规范的问题。可能我们以前都是自己一个人在写代码，写的代码都是给自己看的，所以我们都没有注意到这个问题。就像师兄说的那样，我们的代码看上去很上难看很不舒服，也不知道我们的变量是什么类型的，也不知道是要来做什么的。但是我觉得我们这一组人的代码都写得比较好看，每个人  
的代码都有注释和分隔，就是没有一个统一的规范，每个人都人自己的一个命名规则和习惯，也不能见名知义。还有就是没有定义好一些公共的部分，使每个人都有一个自己的“公共部分”，从而在拼起来时，第一件事，就是改名字。而这些都应该是在项目一开始，还没开始写代码时应该做的。  
  
然后，我自己在计算时，竟然太大意算错了利润，这不能只一句我不小心就敷衍过去，也是我的责任，而且这也是我们的项目的核心部分，以后在做完一个模块后，一定要测试多次，不能过于随便地用一个数据测试一下，能成功就算了，要用可能出现的所有情况去测试程序，让所有的代码都有运行过一次，确认无误。  
  
最后，也是我比较喜欢的东西，就是大家一起为了一个问题去讨论和去交流。因为我觉得，无论是谁，他能想的东西都是有限的，别人总会想到一些自己想不到的地方。跟他人讨论和交流能知道别人的想法、了解别人是怎样想一个问题的，对于同样的问题自己又是怎样想的，是别人的想法好，还是自己的想法好， 好在什么地方。因为我发现问题的能力比较欠缺，所以我也总是喜欢别人问我问题，也喜欢跟别人去讨论一个问题，因为他们帮我发现了我自己没有发现的问题。在这次项目中，我跟植荣的讨论就最多了，很多时候都是不可开交的那种，不过我觉得他总是能够想到很多我想不到的东西，他想的东西也比我深入很多， 虽然很多时候我们好像闹得很僵，但是我们还是很要好的! 嘻嘻!而且在以后的学习和做项目的过程中，我们遇到的问题可能会多很多，复杂很多，我们一个人也不能解决，或者是没有想法，但是懂得与他人讨论与交流就不怕这个问题，总有人的想法会给我们带来一片新天地。相信我能做得更好。  
  
还有就是做项目时要抓准客户的要求，不要自以为是，自己觉得这样好，那样好就把客户的需求改变，项目就是项目，就要根据客户的要求来完成。

#### 如何编写图片上传功能？

使用CDN

#### 如何在后台转发请求？

使用requests

#### 你们这个项目中订单 ID 是怎么生成的?我们公司最近打算做一个电商项目，如果让你设计这块，你会考虑哪些问题?

生成订单ID的目的是为了使订单不重复，本系统订单ID生成规则: \  
用户ID + 当前系统的时间戳  
  
设计的时候我会考虑:  
  
订单ID不能重复  
订单ID尽可能的短(占用存储空间少，实际使用方便，客服相关)  
订单ID要求是全数字(客服)

#### 你们订单系统中用户提交订单的时候验证价格和验证库存是怎么做的？

验价：下单的时候，再跟购物车的价格进行匹配。看是否一致！  
验库存：调用库存系统，库存系统需要【商品id，商品数据量】进行查询，如果够返回1，否则返回0。

#### 如何解决并发问题的?

水桶原理, 主要对于最短板的模块进行弥补.  
  
前台首页 静态化 缓存 服务器的集群  
后台系统 服务器集群 缓存 读写分离  
订单模块 使用MQ进行流量限制.

#### 单点登录系统

单独搭建用户信息中心的服务

请参考：http://cnodejs.org/topic/55f6e69904556da7553d20dd

#### 你们生产环境的服务器有多少台?(重点以web服务器为主)

面试前要数好，一般是十几到二十台。(用在哪里？这是重点) Nginx至少2台 Django至少3台以上(每个模块需要撘集群至少3台) 数据库至少2台 Redis至少1台

#### 数据备份是怎么做的?有没有做读写分离?

主从(一主多从，主要是备份主)，每天备份，备份的文件不要放到数据库服务器上，可以FTP。要检查有效否。读写分离自己查一下，分库分表做过。

#### 你们项目中用户从下单到支付成功大概需要多长时间？是否出现过响应过慢的问题？如何解决的？

1-3秒，响应过慢是因为下单过程中要做的业务比较多，解决办法就是找出非核心的业务操作，例如发送通知，记录日志等等。这些操作可以利用消息队列做成异步的方式处理。

#### 你们使用什么做支付的?如果使用支付宝做支付，请求超时了怎么处理?

1，重试，一般三次，每次重试都要停顿一会，比如，以第一次停顿 1 秒，第二次停顿 2 秒，第三次停顿 3 秒; 2，给订单标识付款异常状态，并且发出警告(邮件、短信)给相关人员。 3，写个定时任务，定时处理异常状态的订单。(一般在凌晨比较空闲的时间)

#### 付款成功后支付宝会有数据返回吗?如果付款后支付宝没有返回，或者返回超时了，但是钱又已经扣了，你怎么办?

1，我们请求了支付宝，但是没有接受到响应，我们就认为该订单没有支付成功，并且将订单标识为异常状态;  
2，使用定时任务处理; 3，做一个对账的任务，实时处理异常状态的订单。

#### 你们怎么做退款功能的，要多长时间才能把钱退回给用户?

用户申请退款后，经过客服审核通过会将退款请求交到支付宝，具体到账时间要看支付宝的处理。

#### 不同域名的网站如何实现用户信息共享

更换电脑，必须登录才能看到之前购物车的商品。 跨域cookie同步方案: 场景:有时一个公司可能有多个不同域名的网站，比如sina.com和weibo.cn, 比如taobao.com和tmall.com。 这些网站背后很多是同一套会员体系。由于http协议规定  
cookie是跟着域名走的，这时就需要在不同的域名下同步登陆状态，避免出现用户体验上出现需要二次登陆验证的情况。 假设下面这样一个场景: 用户在bbb.com上已经登陆，现在要去aaa.com  
上玩，在aaa.com域名下暂未登录。需要访问的aaa.com / resource.html资源需要登录才能访问。两个网站是同一套会员体系，同一个公司的。这是要让用户体验上做到用户在aaa.com上玩也能识别出登录状态。  
  
解释如下: 第一步:用户向aaa.com发起get请求，获取resource.html资源，aaa.com发现用户未登录，返回302状态和外部重定向url: j.bbb.com?target = www.aaa.com / resource.html，注意j.bbb.com子域名上部署的应用可以认为是专门用了跨域同步。 第二步: 用户根据重定向url，访问j.bbb.com?target = www.aaa.com / resource.html，由于在bbb.com上已经登录，所以bbb.com上能拿到从client端传递过来cookie信息。子域j.bbb.com上的应用负责将cookie读取出来，并作为参数再次重定向到p.aaa.com?tartet = www.aaa.com / resource.html & sessionid = xxx & loginId = xxx & ......第三步:用户根据第二步重定向url，访问p.aaa.com。p.aaa.com子域名上的应用专门负责根据请求参数里的参数对，往aaa.com域写入cookie，并重定向到用户第一步请求的url。 第四步: 经过前三步，已经完成了再aaa.com域名下同步bbb.com的登录状态，用户再次请求aaa.com / resource.html, 这是就能成功访问了。

#### 点一个链接访问到一个页面，这个页面上既有静态数据，又有动态数据(需要查数据库的)，打开这个页面的时候就是很慢但是也能打开。怎么解决这个问题，怎么优化?(静态化)

如果要静态页面的话 那就得用ajax异步,通过js操作异步刷新表单,通过 js 对返回结果 组装成 html。 缓存、动态页面静态化: 所谓缓存, 是指将那些经常重复的操作结果暂时存放起来, 在以后的执行过程中, 只要使用前面的暂存结果即可. 那么在我们开发 Web 网站的过程中, 到底有多少工作可以采用用缓存呢? 或者说, 我们可以在哪些地方使 用缓存呢? 首先, 最好的情况是客户端不发送任何请求直接就能获得数据, 这种情况下, 用于缓存的数据保存在客户 端浏览器的缓存中. 其次, 在具有代理服务器的网络环境中, 代理服务器可以针对那些经常访问的网页制作缓存, 当局域网中 第一台主机请求了某个网页并返回结果后, 局域网中的第二台主机再请求同一个网页, 这时代理服务器会 直接返回上一次缓存的结果, 并不会向网络中的 IIS 服务器发送请求, 例如: 现在的连接电信和网通线路的 加速器等. 但是代理服务器通常有自己专门的管理软件和管理系统, 作为网站开发人员对代理服务器的控制能力有限. 再次, 前面也说过, 当用户将请求地址发送到 IIS 服务器时, IIS 服务器会根据请求地址选择不同的行为, 如: 对于\*.aspx 页面会走应用程序管道, 而对\*.html、.jpg 等资源会直接返回资源, 那么我们可以把那些频繁访 问的页面做成.html 文件, 这样用户请求\*.html, 将不用再走应用程序管道, 因此会 升效率. 例如: 网站首 页、或某些突发新闻或突发事件等, 可以考虑做成静态网页. 就拿”天气预报的发布页面”打比方: 天气预报 的发布页面的访问用户非常多, 我们可以考虑将发布页做成静态的\*.html网页, 之后在整个网站程序启动时,在 Gboabl.asax 的 Application\_Start 事件处理器中, 创建子线程以实现每 3 个小时重新获取数据生成新的天 气发布页面内容. 之后的 asp.net 的处理流程, 作为程序员我们是无法干涉的. 直到启动 HttpApplication 管道后, 我们才可以 通过 Global.asax 或 IHttpModule 来控制请求处理过程, 在应用程序管道中适合做整页或用户控件的缓存. 如: 缓存热门页面, 我们可以自动缓存整个网站中访问量超过一定数值(阀值)的页面, 其中为了减小IO操作, 将缓存的页面放在内容中.

#### 如果用户一直向购物车添加商品怎么办?并且他添加一次你查询一次数据库?互联网上用户那么多，这样会对数据库造成很大压力。

首先我们使用 redis 作为一个缓冲的内存数据库作为存储用户购物车的信息,当用户登陆以后,将 redis 的信息写入关系型数据库.第二当用户是登录状态的情况下,也是使用 redis 作为缓存,当用户做出关闭当前页 面操作的时候,将 redis 的内容统一写入到数据库之中.为了 高 redis 的运行效率我没还可以使用 redis 中的 hash 结构.来具体的找到某个字段,做小范围的修改.也可以减轻 redis 的压力.

#### 你们项目中服务器集群数据不一致问题有什么解决方案

使用工具：定期时间进行匹配：如果不匹配，运维通过MySQL修数据

#### 你们电商网站日活多少？你们电商网站中搜索系统用的es节点有几个？文件服务器有几个？

日活3000-10000  
ES 数据量千万及以下的话，两台服务即可。  
文件服务器 2T以内，两台即可

#### 你说你用了 redis 缓存，你 redis 存的是什么格式的数据，是怎么存的?

熟悉几种常用的redis数据结构。

#### 解释一下 Django 和 Tornado 的关系、差别

Django源自一个在线新闻 Web站点，于 2005 年以开源的形式被释放出来。  
Django 框架的核心组件有：  
用于创建模型的对象关系映射为最终用户设计的完美管理界面一流的 URL 设计设计者友好的模板语言缓存系统等等  
它鼓励快速开发,并遵循MVC设计。Django遵守 BSD版权，最新发行版本是Django 1.4，于2012年03月23日发布.Django的主要目的是简便、快速的开发数据库驱动的网站。它强调代码复用,多个组件可以很方便的以“插件”形式服务于整个框架，Django有许多功能强大的第三方插件，你甚至可以很方便的开发出自己的工具包。这使得Django具有很强的可扩展性。它还强调快速开发和DRY(Do Not RepeatYourself)原则。  
Tornado是 FriendFeed使用的可扩展的非阻塞式 web 服务器及其相关工具的开源版本。这个 Web 框架看起来有些像 web.py 或者 Google 的 webapp，不过为了能有效利用非阻塞式服务器环境，这个 Web 框架还包含了一些相关的有用工具和优化。  
Tornado 和现在的主流 Web 服务器框架（包括大多数Python 的框架）有着明显的区别：它是非阻塞式服务器，而且速度相当快。得利于其 非阻塞的方式和对epoll的运用，Tornado 每秒可以处理数以千计的连接，这意味着对于实时 Web服务来说，Tornado 是一个理想的 Web 框架。我们开发这个 Web 服务器的主要目的就是为了处理 FriendFeed 的实时功能 ——在 FriendFeed 的应用里每一个活动用户都会保持着一个服务器连接。（关于如何扩容 服务器，以处理数以千计的客户端的连接的问题。

#### 用户在浏览器中输入一个网址，到 Django 后台捕捉到请求其中的过程？

浏览器先检查缓存，如果有缓存，就从缓存中获得资源文件并加载，如果木有缓存，则执行下一步。 进行 DNS 域名解析，将域名解析成 ip 地址。 与 ip 地址对影的服务器进行 TCP 连接。 接着经历 TCP 3次握手过程。 一旦连接建立后，开始发送 Http 请求。 服务器获得 Http 请求后，将该请求打包成 HttpRequest 对象。 接着检查 Request 中是否需要 Django 中间件的方法，如果没有则执行下一步。 判断 Request 中的各种信息，诸如 user\_agent、GET/POST 等，并在 URLconf 中进行匹配路由到对应的 views 视图函数中。 返回一个 Response 对象，并调用相应的 views 视图函数。 最后返回一个 Http 相应，并加载页面。

#### 解释下 Django 的 MVC 模式，其中那一部分充当的是 controller 的部分？

Django是一个MTV模式的框架, MTV三个部分如下，  
模型(Model)，数据存取层：处理与数据相关的所有事务，即如何存取、如何验证有效性、包含哪些行为以及数据之间的关系等。 模板(Template)，表现层：处理与表现相关的决定，即如何在页面或其他类型文档中进行显示。 视图(View)，业务逻辑层：存取模型及调取恰当模板的相关逻辑。模型与模板之间的桥梁。 而其中作为controller的部分是Django的URLconf。  
  
它获取用户在地址栏中输入的URL并将其路由到views模块对应的各个函数，并调用他们。实现了相应的视图函数路由到相应界面的映射功能。

#### 你在上家公司做开发的时候bug率是多少？

一般由 QA进行统计，程序员可以不了解。

#### 项目中分布式事务怎么做的？

尽量跨模块的分布式事务，如果有我们利用消息队列传递数据，如果失败消息队列会有重试机制，如果还是没成功，会有定时任务根据状态的停滞时间进行扫描，进行补偿。如果还是失败，会发送人工处理。

#### 其它问题

简单的介绍下你在公司的项目，不管是不是后端相关的，主要是要体现出你干了什么； 你在项目中遇到最难的部分是什么，你是怎么解决的； 你看过django的admin源码么；你如何理解开源； MVC / MTV； 缓存怎么用； 中间件是干嘛的； CSRF是什么，django是如何避免的；XSS呢； 如果你来设计login，简单的说一下思路； session和cookie的联系与区别；session为什么说是安全的； uWSGI和Nginx的作用。

# Python知识点汇总（ 命令版 ）

Python文件版本：



## 类型和运算

*#-- 寻求帮助：* dir(obj) *# 简单的列出对象obj所包含的方法名称，返回一个字符串列表* help(obj.func) *# 查询obj.func的具体介绍和用法  
   
#-- 测试类型的三种方法，推荐第三种* **if** type(L) == type([]): print(**"L is list"**)  
 **if** type(L) == list: print(**"L is list"**)  
 **if** isinstance(L, list): print(**"L is list"**)  
   
*#-- Python数据类型：哈希类型、不可哈希类型  
 # 哈希类型，即在原地不能改变的变量类型，不可变类型。可利用hash函数查看其hash值，也可以作为字典的key* **"数字类型：int, float, decimal.Decimal, fractions.Fraction, complex"  
 "字符串类型：str, bytes"  
 "元组：tuple"  
 "冻结集合：frozenset"  
 "布尔类型：True, False"  
 "None"** *# 不可hash类型：原地可变类型：list、dict和set。它们不可以作为字典的key。  
   
#-- 数字常量* 1234, -1234, 0, 999999999 *# 整数* 1.23, 1., 3.14e-10, 4E210, 4.0e+210 *# 浮点数* 0o177, 0x9ff, 0X9FF, 0b101010 *# 八进制、十六进制、二进制数字* 3+4j, 3.0+4.0j, 3J *# 复数常量，也可以用complex(real, image)来创建* hex(I), oct(I), bin(I) *# 将十进制数转化为十六进制、八进制、二进制表示的“字符串”* int(str, base) *# 将字符串转化为整数，base为进制数  
 # 2.x中，有两种整数类型：一般整数（32位）和长整数（无穷精度）。可以用l或L结尾，迫使一般整数成为长整数* float(**'inf'**), float(**'-inf'**), float(**'nan'**) *# 无穷大, 无穷小, 非数  
   
#-- 数字的表达式操作符* **yield** x *# 生成器函数发送协议* **lambda** args: expression *# 生成匿名函数* x **if** y **else** z *# 三元选择表达式* x **and** y, x **or** y, **not** x *# 逻辑与、逻辑或、逻辑非* x **in** y, x **not in** y *# 成员对象测试* x **is** y, x **is not** y *# 对象实体测试* x<y, x<=y, x>y, x>=y, x==y, x!=y *# 大小比较，集合子集或超集值相等性操作符* 1 < a < 3 *# Python中允许连续比较* x|y, x&y, x^y *# 位或、位与、位异或* x<<y, x>>y *# 位操作：x左移、右移y位* +, -, \*, /, //, %, \*\* *# 真除法、floor除法：返回不大于真除法结果的整数值、取余、幂运算* -x, +x, ~x *# 一元减法、识别、按位求补（取反）* x[i], x[i:j:k], x(……) *# 索引、分片、调用* int(3.14), float(3) *# 强制类型转换  
   
#-- 整数可以利用bit\_length函数测试所占的位数* a = 1; a.bit\_length() *# 1* a = 1024; a.bit\_length() *# 11  
   
#-- repr和str显示格式的区别* **"""  
 repr格式：默认的交互模式回显，产生的结果看起来它们就像是代码。  
 str格式：打印语句，转化成一种对用户更加友好的格式。  
 """***#-- 数字相关的模块  
 # math模块  
 # Decimal模块：小数模块* **import** decimal  
 **from** decimal **import** Decimal  
 Decimal(**"0.01"**) + Decimal(**"0.02"**) *# 返回Decimal("0.03")* decimal.getcontext().prec = 4 *# 设置全局精度为4 即小数点后边4位  
 # Fraction模块：分数模块* **from** fractions **import** Fraction  
 x = Fraction(4, 6) *# 分数类型 4/6* x = Fraction(**"0.25"**) *# 分数类型 1/4 接收字符串类型的参数  
   
#-- 集合set* **"""  
 set是一个无序不重复元素集, 基本功能包括关系测试和消除重复元素。  
 set支持union(联合), intersection(交), difference(差)和sysmmetric difference(对称差集)等数学运算。  
 set支持x in set, len(set), for x in set。  
 set不记录元素位置或者插入点, 因此不支持indexing, slicing, 或其它类序列的操作  
 """** s = set([3,5,9,10]) *# 创建一个数值集合，返回{3, 5, 9, 10}* t = set(**"Hello"**) *# 创建一个唯一字符的集合返回{}* a = t | s t.union(s) *# t 和 s的并集* b = t & s t.intersection(s) *# t 和 s的交集* c = t – s t.difference(s) *# 求差集（项在t中, 但不在s中）* d = t ^ s t.symmetric\_difference(s) *# 对称差集（项在t或s中, 但不会同时出现在二者中）* t.add(**'x'**) t.remove(**'H'**) *# 增加/删除一个item* t.update([10,37,42]) *# 利用[......]更新s集合* x **in** s, x **not in** s *# 集合中是否存在某个值* s.issubset(t) s.issuperset(t) s.copy() s.discard(x) s.clear()  
 {x\*\*2 **for** x **in** [1, 2, 3, 4]} *# 集合解析，结果：{16, 1, 4, 9}* {x **for** x **in 'spam'**} *# 集合解析，结果：{'a', 'p', 's', 'm'}  
   
#-- 集合frozenset，不可变对象* **"""  
 set是可变对象，即不存在hash值，不能作为字典的键值。同样的还有list、tuple等  
 frozenset是不可变对象，即存在hash值，可作为字典的键值  
 frozenset对象没有add、remove等方法，但有union/intersection/difference等方法  
 """** a = set([1, 2, 3])  
 b = set()  
 b.add(a) *# error: set是不可哈希类型* b.add(frozenset(a)) *# ok，将set变为frozenset，可哈希  
   
#-- 布尔类型bool* type(**True**) *# 返回<class 'bool'>* isinstance(**False**, int) *# bool类型属于整形，所以返回True* **True** == 1, **True is** 1 *# 输出(True, False)  
   
#-- 动态类型简介* **"""  
 变量名通过引用，指向对象。  
 Python中的“类型”属于对象，而不是变量，每个对象都包含有头部信息，比如"类型标示符" "引用计数器"等  
 """** *#共享引用及在原处修改：对于可变对象，要注意尽量不要共享引用！  
 #共享引用和相等测试：* L = [1], M = [1], L **is** M *# 返回False* L = M = [1, 2, 3], L **is** M *# 返回True，共享引用  
 #增强赋值和共享引用：普通+号会生成新的对象，而增强赋值+=会在原处修改* L = M = [1, 2]  
 L = L + [3, 4] *# L = [1, 2, 3, 4], M = [1, 2]* L += [3, 4] *# L = [1, 2, 3, 4], M = [1, 2, 3, 4]  
   
#-- 常见字符串常量和表达式* S = **''** *# 空字符串* S = **"spam’s"** *# 双引号和单引号相同* S = **"s\np\ta\x00m"** *# 转义字符* S = **"""spam"""** *# 三重引号字符串，一般用于函数说明* S = **r'\temp'** *# Raw字符串，不会进行转义，抑制转义* S = **b'Spam'** *# Python3中的字节字符串* S = **u'spam'** *# Python2.6中的Unicode字符串* s1+s2, s1\*3, s[i], s[i:j], len(s) *# 字符串操作* **'a %s parrot'** % **'kind'** *# 字符串格式化表达式* **'a {0} parrot'**.format(**'kind'**) *# 字符串格式化方法* **for** x **in** s: print(x) *# 字符串迭代，成员关系* [x\*2 **for** x **in** s] *# 字符串列表解析* **','**.join([**'a'**, **'b'**, **'c'**]) *# 字符串输出，结果：a,b,c  
   
#-- 内置str处理函数：* str.upper() str.lower() str.swapcase() str.capitalize() str.title() *# 全部大写，全部小写、大小写转换，首字母大写，每个单词的首字母都大写* str.ljust(width) *# 获取固定长度，右对齐，左边不够用空格补齐* str.rjust(width) *# 获取固定长度，左对齐，右边不够用空格补齐* str.center(width) *# 获取固定长度，中间对齐，两边不够用空格补齐* str.zfill(width) *# 获取固定长度，右对齐，左边不足用0补齐* str.find(**'t'**,start,end) *# 查找字符串，可以指定起始及结束位置搜索* str.rfind(**'t'**) *# 从右边开始查找字符串* str.count(**'t'**) *# 查找字符串出现的次数  
 #上面所有方法都可用index代替，不同的是使用index查找不到会抛异常，而find返回-1* str.replace(**'old'**,**'new'**) *# 替换函数，替换old为new，参数中可以指定maxReplaceTimes，即替换指定次数的old为new* str.strip() str.lstrip() str.rstrip() str.strip(**'d'**) str.lstrip(**'d'**) str.rstrip(**'d'**)  
 str.startswith(**'start'**) *# 是否以start开头* str.endswith(**'end'**) *# 是否以end结尾* str.isalnum() str.isalpha() str.isdigit() str.islower() str.isupper() *# 判断字符串是否全为字符、数字、大写、小写  
   
#-- 三重引号编写多行字符串块，并且在代码折行处嵌入换行字符\n* mantra = **"""hello world  
 hello python  
 hello my friend"""** *# mantra为"""hello world \n hello python \n hello my friend"""  
   
#-- 索引和分片：* S[0], S[len(S) – 1], S[-1] *# 索引* S[1:3], S[1:], S[:-1], S[1:10:2] *# 分片，第三个参数指定步长  
   
#-- 字符串转换工具：* int(**'42'**), str(42) *# 返回(42, '42')* float(**'4.13'**), str(4.13) *# 返回(4.13, '4.13')* ord(**'s'**), chr(115) *# 返回(115, 's')* int(**'1001'**, 2) *# 将字符串作为二进制数字，转化为数字，返回13* bin(13), oct(13), hex(13) *# 将整数转化为二进制/八进制/十六进制字符串，返回('1001', '0o15', '0xd')  
   
#-- 另类字符串连接* name = **"wang" "hong"** *#单行，name = "wanghong"* name = **"wang"** \  
 **"hong"** *#多行，name = "wanghong"  
   
#-- Python中的字符串格式化实现1--字符串格式化表达式* **"""  
 基于C语言的'print'模型，并且在大多数的现有的语言中使用。  
 通用结构：%[(name)][flag][width].[precision]typecode  
 """  
 "this is %d %s bird"** % (1, **'dead'**) *# 一般的格式化表达式* **"%s---%s---%s"** % (42, 3.14, [1, 2, 3]) *# 字符串输出：'42---3.14---[1, 2, 3]'* **"%d...%6d...%-6d...%06d"** % (1234, 1234, 1234, 1234) *# 对齐方式及填充："1234... 1234...1234 ...001234"* x = 1.23456789  
 **"%e | %f | %g"** % (x, x, x) *# 对齐方式："1.234568e+00 | 1.234568 | 1.23457"* **"%6.2f\*%-6.2f\*%06.2f\*%+6.2f"** % (x, x, x, x) *# 对齐方式：' 1.23\*1.23 \*001.23\* +1.23'* **"%(name1)d---%(name2)s"** % {**"name1"**:23, **"name2"**:**"value2"**} *# 基于字典的格式化表达式* **"%(name)s is %(age)d"** % vars() *# vars()函数调用返回一个字典，包含了所有本函数调用时存在的变量  
   
#-- Python中的字符串格式化实现2--字符串格式化调用方法  
 # 普通调用* **"{0}, {1} and {2}"**.format(**'spam'**, **'ham'**, **'eggs'**) *# 基于位置的调用* **"{motto} and {pork}"**.format(motto = **'spam'**, pork = **'ham'**) *# 基于Key的调用* **"{motto} and {0}"**.format(ham, motto = **'spam'**) *# 混合调用  
 # 添加键 属性 偏移量 (import sys)* **"my {1[spam]} runs {0.platform}"**.format(sys, {**'spam'**:**'laptop'**}) *# 基于位置的键和属性* **"{config[spam]} {sys.platform}"**.format(sys = sys, config = {**'spam'**:**'laptop'**}) *# 基于Key的键和属性* **"first = {0[0]}, second = {0[1]}"**.format([**'A'**, **'B'**, **'C'**]) *# 基于位置的偏移量  
 # 具体格式化* **"{0:e}, {1:.3e}, {2:g}"**.format(3.14159, 3.14159, 3.14159) *# 输出'3.141590e+00, 3.142e+00, 3.14159'* **"{fieldname:format\_spec}"**.format(......)  
 *# 说明:* **"""  
 fieldname是指定参数的一个数字或关键字, 后边可跟可选的".name"或"[index]"成分引用  
 format\_spec ::= [[fill]align][sign][#][0][width][,][.precision][type]  
 fill ::= <any character> #填充字符  
 align ::= "<" | ">" | "=" | "^" #对齐方式  
 sign ::= "+" | "-" | " " #符号说明  
 width ::= integer #字符串宽度  
 precision ::= integer #浮点数精度  
 type ::= "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" | "n" | "o" | "s" | "x" | "X" | "%"  
 """** *# 例子:* **'={0:10} = {1:10}'**.format(**'spam'**, 123.456) *# 输出'=spam = 123.456'* **'={0:>10}='**.format(**'test'**) *# 输出'= test='* **'={0:<10}='**.format(**'test'**) *# 输出'=test ='* **'={0:^10}='**.format(**'test'**) *# 输出'= test ='* **'{0:X}, {1:o}, {2:b}'**.format(255, 255, 255) *# 输出'FF, 377, 11111111'* **'My name is {0:{1}}.'**.format(**'Fred'**, 8) *# 输出'My name is Fred .' 动态指定参数  
   
#-- 常用列表常量和操作* L = [[1, 2], **'string'**, {}] *# 嵌套列表* L = list(**'spam'**) *# 列表初始化* L = list(range(0, 4)) *# 列表初始化* list(map(ord, **'spam'**)) *# 列表解析* len(L) *# 求列表长度* L.count(value) *# 求列表中某个值的个数* L.append(obj) *# 向列表的尾部添加数据，比如append(2)，添加元素2* L.insert(index, obj) *# 向列表的指定index位置添加数据，index及其之后的数据后移* L.extend(interable) *# 通过添加iterable中的元素来扩展列表，比如extend([2])，添加元素2，注意和append的区别* L.index(value, [start, [stop]]) *# 返回列表中值value的第一个索引* L.pop([index]) *# 删除并返回index处的元素，默认为删除并返回最后一个元素* L.remove(value) *# 删除列表中的value值，只删除第一次出现的value的值* L.reverse() *# 反转列表* L.sort(cmp=**None**, key=**None**, reverse=**False**) *# 排序列表* a = [1, 2, 3], b = a[10:] *# 注意，这里不会引发IndexError异常，只会返回一个空的列表[]* a = [], a += [1] *# 这里实在原有列表的基础上进行操作，即列表的id没有改变* a = [], a = a + [1] *# 这里最后的a要构建一个新的列表，即a的id发生了变化  
   
#-- 用切片来删除序列的某一段* a = [1, 2, 3, 4, 5, 6, 7]  
 a[1:4] = [] *# a = [1, 5, 6, 7]* a = [0, 1, 2, 3, 4, 5, 6, 7]  
 **del** a[::2] *# 去除偶数项(偶数索引的)，a = [1, 3, 5, 7]  
   
#-- 常用字典常量和操作* D = {}  
 D = {**'spam'**:2, **'tol'**:{**'ham'**:1}} *# 嵌套字典* D = dict.fromkeys([**'s'**, **'d'**], 8) *# {'d': 8, 's': 8}* D = dict(name = **'tom'**, age = 12) *# {'age': 12, 'name': 'tom'}* D = dict([(**'name'**, **'tom'**), (**'age'**, 12)]) *# {'age': 12, 'name': 'tom'}* D = dict(zip([**'name'**, **'age'**], [**'tom'**, 12])) *# {'age': 12, 'name': 'tom'}* D.keys() D.values() D.items() *# 字典键、值以及键值对* D.get(key, default) *# get函数* D.update(D\_other) *# 合并字典，如果存在相同的键值，D\_other的数据会覆盖掉D的数据* D.pop(key, [D]) *# 删除字典中键值为key的项，返回键值为key的值，如果不存在，返回默认值D，否则异常* D.popitem() *# pop字典中的一项（一个键值对）* D.setdefault(k[, d]) *# 设置D中某一项的默认值。如果k存在，则返回D[k]，否则设置D[k]=d，同时返回D[k]。* **del** D *# 删除字典* **del** D[**'key'**] *# 删除字典的某一项* **if** key **in** D: **if** key **not in** D: *# 测试字典键是否存在  
 # 字典注意事项：（1）对新索引赋值会添加一项（2）字典键不一定非得是字符串，也可以为任何的不可变对象  
   
#-- 字典解析* D = {k:8 **for** k **in** [**'s'**, **'d'**]} *# {'d': 8, 's': 8}* D = {k:v **for** (k, v) **in** zip([**'name'**, **'age'**], [**'tom'**, 12])}  
   
*#-- 字典的特殊方法\_\_missing\_\_：当查找找不到key时，会执行该方法* **class** Dict(dict):  
 **def** \_\_missing\_\_(self, key):  
 self[key] = []  
 **return** self[key]  
 dct = Dict()  
 dct[**"foo"**].append(1) *# 这有点类似于collections.defalutdict* dct[**"foo"**] *# [1]  
   
#-- 元组和列表的唯一区别在于元组是不可变对象，列表时可变对象* a = [1, 2, 3] *# a[1] = 0, OK* a = (1, 2, 3) *# a[1] = 0, Error* a = ([1, 2]) *# a[0][1] = 0, OK* a = [(1, 2)] *# a[0][1] = 0, Error  
   
#-- 元组的特殊语法: 逗号和圆括号* D = (12) *# 此时D为一个整数 即D = 12* D = (12, ) *# 此时D为一个元组 即D = (12, )  
   
#-- 文件基本操作* output = open(**r'C:\spam'**, **'w'**) *# 打开输出文件，用于写* input = open(**'data'**, **'r'**) *# 打开输入文件，用于读。打开的方式可以为'w', 'r', 'a', 'wb', 'rb', 'ab'等* fp.read([size]) *# size为读取的长度，以byte为单位* fp.readline([size]) *# 读一行，如果定义了size，有可能返回的只是一行的一部分* fp.readlines([size]) *# 把文件每一行作为一个list的一个成员，并返回这个list。其实它的内部是通过循环调用readline()来实现的。如果提供size参数，size是表示读取内容的总长。* fp.readable() *# 是否可读* fp.write(str) *# 把str写到文件中，write()并不会在str后加上一个换行符* fp.writelines(seq) *# 把seq的内容全部写到文件中(多行一次性写入)* fp.writeable() *# 是否可写* fp.close() *# 关闭文件。* fp.flush() *# 把缓冲区的内容写入硬盘* fp.fileno() *# 返回一个长整型的”文件标签“* fp.isatty() *# 文件是否是一个终端设备文件（unix系统中的）* fp.tell() *# 返回文件操作标记的当前位置，以文件的开头为原点* fp.next() *# 返回下一行，并将文件操作标记位移到下一行。把一个file用于for … in file这样的语句时，就是调用next()函数来实现遍历的。* fp.seek(offset[,whence]) *# 将文件打操作标记移到offset的位置。whence可以为0表示从头开始计算，1表示以当前位置为原点计算。2表示以文件末尾为原点进行计算。* fp.seekable() *# 是否可以seek* fp.truncate([size]) *# 把文件裁成规定的大小，默认的是裁到当前文件操作标记的位置。* **for** line **in** open(**'data'**):   
 print(line) *# 使用for语句，比较适用于打开比较大的文件* open(**'f.txt'**, encoding = **'latin-1'**) *# Python3.x Unicode文本文件* open(**'f.bin'**, **'rb'**) *# Python3.x 二进制bytes文件  
 # 文件对象还有相应的属性：buffer closed encoding errors line\_buffering name newlines等  
   
#-- 其他  
 # Python中的真假值含义：1. 数字如果非零，则为真，0为假。 2. 其他对象如果非空，则为真  
 # 通常意义下的类型分类：1. 数字、序列、映射。 2. 可变类型和不可变类型*

## 语法和语句

*#-- 赋值语句的形式* spam = **'spam'** *# 基本形式* spam, ham = **'spam'**, **'ham'** *# 元组赋值形式* [spam, ham] = [**'s'**, **'h'**] *# 列表赋值形式* a, b, c, d = **'abcd'** *# 序列赋值形式* a, \*b, c = **'spam'** *# 序列解包形式（Python3.x中才有）* spam = ham = **'no'** *# 多目标赋值运算，涉及到共享引用* spam += 42 *# 增强赋值，涉及到共享引用  
   
#-- 序列赋值 序列解包* [a, b, c] = (1, 2, 3) *# a = 1, b = 2, c = 3* a, b, c, d = **"spam"** *# a = 's', b = 'p'* a, b, c = range(3) *# a = 0, b = 1* a, \*b = [1, 2, 3, 4] *# a = 1, b = [2, 3, 4]* \*a, b = [1, 2, 3, 4] *# a = [1, 2, 3], b = 4* a, \*b, c = [1, 2, 3, 4] *# a = 1, b = [2, 3], c = 4  
 # 带有\*时 会优先匹配\*之外的变量 如* a, \*b, c = [1, 2] *# a = 1, c = 2, b = []  
   
#-- print函数原型* print(value, ..., sep=**' '**, end=**'\n'**, file=sys.stdout, flush=**False**)  
 *# 流的重定向* print(**'hello world'**) *# 等于sys.stdout.write('hello world')* temp = sys.stdout *# 原有流的保存* sys.stdout = open(**'log.log'**, **'a'**) *# 流的重定向* print(**'hello world'**) *# 写入到文件log.log* sys.stdout.close()  
 sys.stdout = temp *# 原有流的复原  
   
#-- Python中and或or总是返回对象(左边的对象或右边的对象) 且具有短路求值的特性* 1 **or** 2 **or** 3 *# 返回 1* 1 **and** 2 **and** 3 *# 返回 3  
   
#-- if/else三元表达符（if语句在行内）* A = 1 **if** X **else** 2  
 A = 1 **if** X **else** (2 **if** Y **else** 3)  
 *# 也可以使用and-or语句（一条语句实现多个if-else）* result = (a > 20 **and "big than 20" or** a > 10 **and "big than 10" or** a > 5 **and "big than 5"**)  
   
*#-- Python的while语句或者for语句可以带else语句 当然也可以带continue/break/pass语句* **while** a > 1:  
 ......  
 **else**:  
 ......  
 *# else语句会在循环结束后执行，除非在循环中执行了break，同样的还有for语句* **for** i **in** range(5):  
 ......  
 **else**:  
 ......  
   
*#-- for循环的元组赋值* **for** (a, b) **in** [(1, 2), (3, 4)]: *# 最简单的赋值* **for** ((a, b), c) **in** [((1, 2), 3), ((4, 5), 6)]: *# 自动解包赋值* **for** ((a, b), c) **in** [((1, 2), 3), (**"XY"**, 6)]: *# 自动解包 a = X, b = Y, c = 6* **for** (a, \*b) **in** [(1, 2, 3), (4, 5, 6)]: *# 自动解包赋值  
   
#-- 列表解析语法* M = [[1,2,3], [4,5,6], [7,8,9]]  
 res = [sum(row) **for** row **in** M] *# G = [6, 15, 24] 一般的列表解析 生成一个列表* res = [c \* 2 **for** c **in 'spam'**] *# ['ss', 'pp', 'aa', 'mm']* res = [a \* b **for** a **in** [1, 2] **for** b **in** [4, 5]] *# 多解析过程 返回[4, 5, 8, 10]* res = [a **for** a **in** [1, 2, 3] **if** a < 2] *# 带判断条件的解析过程* res = [a **if** a > 0 **else** 0 **for** a **in** [-1, 0, 1]] *# 带判断条件的高级解析过程  
 # 两个列表同时解析：使用zip函数* **for** teama, teamb **in** zip([**"Packers"**, **"49ers"**], [**"Ravens"**, **"Patriots"**]):  
 print(teama + **" vs. "** + teamb)  
 *# 带索引的列表解析：使用enumerate函数* **for** index, team **in** enumerate([**"Packers"**, **"49ers"**, **"Ravens"**, **"Patriots"**]):  
 print(index, team) *# 输出0, Packers \n 1, 49ers \n ......  
   
#-- 生成器表达式* G = (sum(row) **for** row **in** M) *# 使用小括号可以创建所需结果的生成器generator object* next(G), next(G), next(G) *# 输出(6, 15, 24)* G = {sum(row) **for** row **in** M} *# G = {6, 15, 24} 解析语法还可以生成集合和字典* G = {i:sum(M[i]) **for** i **in** range(3)} *# G = {0: 6, 1: 15, 2: 24}  
   
#-- 文档字符串:出现在Module的开端以及其中函数或类的开端 使用三重引号字符串* **"""  
 module document  
 """  
 def** func():  
 *"""  
 function document  
 """* print()  
 **class** Employee:  
 *"""  
 class document  
 """* print()  
 print(func.\_\_doc\_\_) *# 输出函数文档字符串* print(Employee.\_\_doc\_\_) *# 输出类的文档字符串  
   
#-- 命名惯例:* **"""  
 以单一下划线开头的变量名(\_X)不会被from module import\*等语句导入  
 前后有两个下划线的变量名(\_\_X\_\_)是系统定义的变量名，对解释器有特殊意义  
 以两个下划线开头但不以下划线结尾的变量名(\_\_X)是类的本地(私有)变量  
 """***#-- 列表解析 in成员关系测试 map sorted zip enumerate内置函数等都使用了迭代协议* **'first line' in** open(**'test.txt'**) *# in测试 返回True或False* list(map(str.upper, open(**'t'**))) *# map内置函数* sorted(iter([2, 5, 8, 3, 1])) *# sorted内置函数* list(zip([1, 2], [3, 4])) *# zip内置函数 [(1, 3), (2, 4)]  
   
#-- del语句: 手动删除某个变量* **del** X  
   
*#-- 获取列表的子表的方法:* x = [1,2,3,4,5,6]  
 x[:3] *# 前3个[1,2,3]* x[1:5] *# 中间4个[2,3,4,5]* x[-3:] *# 最后3个[4,5,6]* x[::2] *# 奇数项[1,3,5]* x[1::2] *# 偶数项[2,4,6]  
   
#-- 手动迭代：iter和next* L = [1, 2]  
 I = iter(L) *# I为L的迭代器* I.next() *# 返回1* I.next() *# 返回2* I.next() *# Error:StopIteration  
   
#-- Python中的可迭代对象* **"""  
 1.range迭代器  
 2.map、zip和filter迭代器  
 3.字典视图迭代器：D.keys()), D.items()等  
 4.文件类型  
 """**

## 函数语法规则

*#-- 函数相关的语句和表达式* myfunc(**'spam'**) *# 函数调用* **def** myfunc(): *# 函数定义* **return None** *# 函数返回值* **global** a *# 全局变量* **nonlocal** x *# 在函数或其他作用域中使用外层（非全局）变量* **yield** x *# 生成器函数返回* **lambda** *# 匿名函数  
   
#-- Python函数变量名解析:LEGB原则，即:* **"""  
 local(functin) --> encloseing function locals --> global(module) --> build-in(python)  
 说明:以下边的函数maker为例 则相对于action而言 X为Local N为Encloseing  
 """***#-- 嵌套函数举例:工厂函数* **def** maker(N):  
 **def** action(X):  
 **return** X \*\* N  
 **return** action  
 f = maker(2) *# pass 2 to N* f(3) *# 9, pass 3 to X  
   
#-- 嵌套函数举例:lambda实例* **def** maker(N):  
 action = (**lambda** X: X\*\*N)  
 **return** action  
 f = maker(2) *# pass 2 to N* f(3) *# 9, pass 3 to X  
   
#-- nonlocal和global语句的区别  
 # nonlocal应用于一个嵌套的函数的作用域中的一个名称 例如:* start = 100  
 **def** tester(start):  
 **def** nested(label):  
 **nonlocal** start *# 指定start为tester函数内的local变量 而不是global变量start* print(label, start)  
 start += 3  
 **return** nested  
 *# global为全局的变量 即def之外的变量* **def** tester(start):  
 **def** nested(label):  
 **global** start *# 指定start为global变量start* print(label, start)  
 start += 3  
 **return** nested   
   
*#-- 函数参数，不可变参数通过“值”传递，可变参数通过“引用”传递* **def** f(a, b, c): print(a, b, c)  
 f(1, 2, 3) *# 参数位置匹配* f(1, c = 3, b = 2) *# 参数关键字匹配* **def** f(a, b = 1, c = 2): print(a, b, c)  
 f(1) *# 默认参数匹配* f(1, 2) *# 默认参数匹配* f(a = 1, c = 3) *# 关键字参数和默认参数的混合  
 # Keyword-Only参数:出现在\*args之后 必须用关键字进行匹配* **def** keyOnly(a, \*b, c): print(**''**) *# c就为keyword-only匹配 必须使用关键字c = value匹配* **def** keyOnly(a, \*, b, c): ...... *# b c为keyword-only匹配 必须使用关键字匹配* **def** keyOnly(a, \*, b = 1): ...... *# b有默认值 或者省略 或者使用关键字参数b = value  
   
#-- 可变参数匹配: \* 和 \*\** **def** f(\*args): print(args) *# 在元组中收集不匹配的位置参数* f(1, 2, 3) *# 输出(1, 2, 3)* **def** f(\*\*args): print(args) *# 在字典中收集不匹配的关键字参数* f(a = 1, b = 2) *# 输出{'a':1, 'b':2}* **def** f(a, \*b \*\*c): print(a, b, c) *# 两者混合使用* f(1, 2, 3, x = 4, y = 5) *# 输出1, (2, 3), {'x':4, 'y':5}  
   
#-- 函数调用时的参数解包: \* 和 \*\* 分别解包元组和字典* func(1, \*(2, 3)) <==> func(1, 2, 3)  
 func(1, \*\*{**'c'**:3, **'b'**:2}) <==> func(1, b = 2, c = 3)  
 func(1, \*(2, 3), \*\*{**'c'**:3, **'b'**:2}) <==> func(1, 2, 3, b = 2, c = 3)  
   
*#-- 函数属性:(自己定义的)函数可以添加属性* **def** func():.....  
 func.count = 1 *# 自定义函数添加属性* print.count = 1 *# Error 内置函数不可以添加属性  
   
#-- 函数注解: 编写在def头部行 主要用于说明参数范围、参数类型、返回值类型等* **def** func(a:**'spam'**, b:(1, 10), c:float) -> int :  
 print(a, b, c)  
 func.\_\_annotations\_\_ *# {'c':<class 'float'>, 'b':(1, 10), 'a':'spam', 'return':<class 'int'>}  
 # 编写注解的同时 还是可以使用函数默认值 并且注解的位置位于=号的前边* **def** func(a:**'spam'**=**'a'**, b:(1, 10)=2, c:float=3) -> int :  
 print(a, b, c)  
   
*#-- 匿名函数:lambda* f = **lambda** x, y, z : x + y + z *# 普通匿名函数，使用方法f(1, 2, 3)* f = **lambda** x = 1, y = 1: x + y *# 带默认参数的lambda函数* **def** action(x): *# 嵌套lambda函数* **return** (**lambda** y : x + y)  
 f = **lambda**: a **if** xxx() **else** b *# 无参数的lambda函数，使用方法f()  
   
#-- lambda函数与map filter reduce函数的结合* list(map((**lambda** x: x + 1), [1, 2, 3])) *# [2, 3, 4]* list(filter((**lambda** x: x > 0), range(-4, 5))) *# [1, 2, 3, 4]* functools.reduce((**lambda** x, y: x + y), [1, 2, 3]) *# 6* functools.reduce((**lambda** x, y: x \* y), [2, 3, 4]) *# 24  
   
#-- 生成器函数:yield VS return* **def** gensquare(N):  
 **for** i **in** range(N):  
 **yield** i\*\* 2 *# 状态挂起 可以恢复到此时的状态* **for** i **in** gensquare(5): *# 使用方法* print(i, end = **' '**) *# [0, 1, 4, 9, 16]* x = gensquare(2) *# x是一个生成对象* next(x) *# 等同于x.\_\_next\_\_() 返回0* next(x) *# 等同于x.\_\_next\_\_() 返回1* next(x) *# 等同于x.\_\_next\_\_() 抛出异常StopIteration  
   
#-- 生成器表达式:小括号进行列表解析* G = (x \*\* 2 **for** x **in** range(3)) *# 使用小括号可以创建所需结果的生成器generator object* next(G), next(G), next(G) *# 和上述中的生成器函数的返回值一致  
 #（1）生成器(生成器函数/生成器表达式)是单个迭代对象* G = (x \*\* 2 **for** x **in** range(4))  
 I1 = iter(G) *# 这里实际上iter(G) = G* next(I1) *# 输出0* next(G) *# 输出1* next(I1) *# 输出4  
 #（2）生成器不保留迭代后的结果* gen = (i **for** i **in** range(4))  
 2 **in** gen *# 返回True* 3 **in** gen *# 返回True* 1 **in** gen *# 返回False，其实检测2的时候，1已经就不在生成器中了，即1已经被迭代过了，同理2、3也不在了  
   
#-- 本地变量是静态检测的* X = 22 *# 全局变量X的声明和定义* **def** test():  
 print(X) *# 如果没有下一语句 则该句合法 打印全局变量X* X = 88 *# 这一语句使得上一语句非法 因为它使得X变成了本地变量 上一句变成了打印一个未定义的本地变量(局部变量)* **if False**: *# 即使这样的语句 也会把print语句视为非法语句 因为:* X = 88 *# Python会无视if语句而仍然声明了局部变量X* **def** test(): *# 改进* **global** X *# 声明变量X为全局变量* print(X) *# 打印全局变量X* X = 88 *# 改变全局变量X  
   
#-- 函数的默认值是在函数定义的时候实例化的 而不是在调用的时候 例子:* **def** foo(numbers=[]): *# 这里的[]是可变的* numbers.append(9)   
 print(numbers)  
 foo() *# first time, like before, [9]* foo() *# second time, not like before, [9, 9]* foo() *# third time, not like before too, [9, 9, 9]  
 # 改进:* **def** foo(numbers=**None**):  
 **if** numbers **is None**: numbers = []  
 numbers.append(9)  
 print(numbers)  
 *# 另外一个例子 参数的默认值为不可变的:* **def** foo(count=0): *# 这里的0是数字, 是不可变的* count += 1  
 print(count)  
 foo() *# 输出1* foo() *# 还是输出1* foo(3) *# 输出4* foo() *# 还是输出1*

## 函数例子

**"""数学运算类"""**abs(x) *# 求绝对值，参数可以是整型，也可以是复数，若参数是复数，则返回复数的模*complex([real[, imag]]) *# 创建一个复数*divmod(a, b) *# 分别取商和余数，注意：整型、浮点型都可以*float([x]) *# 将一个字符串或数转换为浮点数。如果无参数将返回0.0*int([x[, base]]) *# 将一个字符串或浮点数转换为int类型，base表示进制*long([x[, base]]) *# 将一个字符串或浮点数转换为long类型*pow(x, y) *# 返回x的y次幂*range([start], stop[, step]) *# 产生一个序列，默认从0开始*round(x[, n]) *# 四舍五入*sum(iterable[, start]) *# 对集合求和*oct(x) *# 将一个数字转化为8进制字符串*hex(x) *# 将一个数字转换为16进制字符串*chr(i) *# 返回给定int类型对应的ASCII字符*unichr(i) *# 返回给定int类型的unicode*ord(c) *# 返回ASCII字符对应的整数*bin(x) *# 将整数x转换为二进制字符串*bool([x]) *# 将x转换为Boolean类型***"""集合类操作"""**basestring() *# str和unicode的超类，不能直接调用，可以用作isinstance判断*format(value [, format\_spec]) *# 格式化输出字符串，格式化的参数顺序从0开始，如“I am {0},I like {1}”*enumerate(sequence[, start=0]) *# 返回一个可枚举的对象，注意它有第二个参数*iter(obj[, sentinel]) *# 生成一个对象的迭代器，第二个参数表示分隔符*max(iterable[, args...][key]) *# 返回集合中的最大值*min(iterable[, args...][key]) *# 返回集合中的最小值*dict([arg]) *# 创建数据字典*list([iterable]) *# 将一个集合类转换为另外一个集合类*set() *# set对象实例化*frozenset([iterable]) *# 产生一个不可变的set*tuple([iterable]) *# 生成一个tuple类型*str([object]) *# 转换为string类型*sorted(iterable[, cmp[, key[, reverse]]]) *# 集合排序* L = [(**'b'**,2),(**'a'**,1),(**'c'**,3),(**'d'**,4)]  
 sorted(L, key=**lambda** x: x[1]), reverse=**True**) *# 使用Key参数和reverse参数* sorted(L, key=**lambda** x: (x[0], x[1])) *# 使用key参数进行多条件排序，即如果x[0]相同，则比较x[1]***"""逻辑判断"""**all(iterable) *# 集合中的元素都为真的时候为真，特别的，若为空串返回为True*any(iterable) *# 集合中的元素有一个为真的时候为真，特别的，若为空串返回为False*cmp(x, y) *# 如果x < y ,返回负数；x == y, 返回0；x > y,返回正数***"""IO操作"""**file(filename [, mode [, bufsize]]) *# file类型的构造函数。*input([prompt]) *# 获取用户输入，推荐使用raw\_input，因为该函数将不会捕获用户的错误输入*raw\_input([prompt]) *# 设置输入，输入都是作为字符串处理*open(name[, mode[, buffering]]) *# 打开文件，与file有什么不同？推荐使用open***"""其他"""**callable(object) *# 检查对象object是否可调用*classmethod(func) *# 用来说明这个func是个类方法*staticmethod(func) *# 用来说明这个func为静态方法*dir([object]) *# 不带参数时，返回当前范围内的变量、方法和定义的类型列表；带参数时，返回参数的属性、方法列表。*help(obj) *# 返回obj的帮助信息*eval(expression) *# 计算表达式expression的值，并返回*exec(str) *# 将str作为Python语句执行*execfile(filename) *# 用法类似exec()，不同的是execfile的参数filename为文件名，而exec的参数为字符串。*filter(function, iterable) *# 构造一个序列，等价于[item for item in iterable if function(item)]，function返回值为True或False的函数* list(filter(bool, range(-3, 4)))*# 返回[-3, -2, -1, 1, 2, 3], 没有0*hasattr(object, name) *# 判断对象object是否包含名为name的特性*getattr(object, name [, defalut]) *# 获取一个类的属性*setattr(object, name, value) *# 设置属性值*delattr(object, name) *# 删除object对象名为name的属性*globals() *# 返回一个描述当前全局符号表的字典*hash(object) *# 如果对象object为哈希表类型，返回对象object的哈希值*id(object) *# 返回对象的唯一标识，一串数字*isinstance(object, classinfo) *# 判断object是否是class的实例* isinstance(1, int) *# 判断是不是int类型* isinstance(1, (int, float)) *# isinstance的第二个参数接受一个元组类型*issubclass(**class**, classinfo) *# 判断class是否为classinfo的子类*locals() *# 返回当前的变量列表*map(function, iterable, ...) *# 遍历每个元素，执行function操作* list(map(abs, range(-3, 4))) *# 返回[3, 2, 1, 0, 1, 2, 3]*next(iterator[, default]) *# 类似于iterator.next()*property([fget[, fset[, fdel[, doc]]]]) *# 属性访问的包装类，设置后可以通过c.x=value等来访问setter和getter*reduce(function, iterable[, initializer]) *# 合并操作，从第一个开始是前两个参数，然后是前两个的结果与第三个合并进行处理，以此类推* **def** add(x,y):**return** x + y   
 reduce(add, range(1, 11)) *# 返回55 (注:1+2+3+4+5+6+7+8+9+10 = 55)* reduce(add, range(1, 11), 20) *# 返回75*reload(module) *# 重新加载模块*repr(object) *# 将一个对象变幻为可打印的格式*slice(start, stop[, step]) *# 产生分片对象*type(object) *# 返回该object的类型*vars([object]) *# 返回对象的变量名、变量值得字典* a = Class(); *# Class为一个空类* a.name = **'qi'**, a.age = 9  
 vars(a) *# {'name':'qi', 'age':9}*zip([iterable, ...]) *# 返回对应数组* list(zip([1, 2, 3], [4, 5, 6])) *# [(1, 4), (2, 5), (3, 6)]* a = [1, 2, 3], b = [**"a"**, **"b"**, **"c"**]  
 z = zip(a, b) *# 压缩：[(1, "a"), (2, "b"), (3, "c")]* zip(\*z) *# 解压缩：[(1, 2, 3), ("a", "b", "c")]*unicode(string, encoding, errors) *# 将字符串string转化为unicode形式，string为encoded string。*

## 模块moudle

*#-- Python模块搜索路径:* **"""  
 (1)程序的主目录 (2)PYTHONPATH目录 (3)标准链接库目录 (4)任何.pth文件的内容  
 """***#-- 查看全部的模块搜索路径* **import** sys  
 sys.path  
   
*#-- 模块的使用代码* **import** module1, module2 *# 导入module1 使用module1.printer()* **from** module1 **import** printer *# 导入module1中的printer变量 使用printer()* **from** module1 imoprt \* *# 导入module1中的全部变量 使用不必添加module1前缀  
   
#-- 重载模块reload: 这是一个内置函数 而不是一条语句* **from** imp **import** reload  
 reload(module)  
   
*#-- 模块的包导入:使用点号(.)而不是路径(dir1\dir2)进行导入* **import** dir1.dir2.mod *# d导入包(目录)dir1中的包dir2中的mod模块 此时dir1必须在Python可搜索路径中* **from** dir1.dir2.mod **import** \* *# from语法的包导入  
   
#-- \_\_init\_\_.py包文件:每个导入的包中都应该包含这么一个文件* **"""  
 该文件可以为空  
 首次进行包导入时 该文件会自动执行  
 高级功能:在该文件中使用\_\_all\_\_列表来定义包(目录)以from\*的形式导入时 需要导入什么  
 """***#-- 包相对导入:使用点号(.) 只能使用from语句* **from** . **import** spam *# 导入当前目录下的spam模块（错误: 当前目录下的模块, 直接导入即可）* **from** .spam **import** name *# 导入当前目录下的spam模块的name属性（错误: 当前目录下的模块, 直接导入即可，不用加.）* **from** .. **import** spam *# 导入当前目录的父目录下的spam模块  
   
#-- 包相对导入与普通导入的区别* **from** string **import** \* *# 这里导入的string模块为sys.path路径上的 而不是本目录下的string模块(如果存在也不是)* **from** .string **import** \* *# 这里导入的string模块为本目录下的(不存在则导入失败) 而不是sys.path路径上的  
   
#-- 模块数据隐藏:最小化from\*的破坏* \_X *# 变量名前加下划线可以防止from\*导入时该变量名被复制出去* \_\_all\_\_ = [**'x'**, **'x1'**, **'x2'**] *# 使用\_\_all\_\_列表指定from\*时复制出去的变量名(变量名在列表中为字符串形式)  
   
#-- 可以使用\_\_name\_\_进行模块的单元测试:当模块为顶层执行文件时值为'\_\_main\_\_' 当模块被导入时为模块名* **if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 doSomething  
 *# 模块属性中还有其他属性，例如：* \_\_doc\_\_ *# 模块的说明文档* \_\_file\_\_ *# 模块文件的文件名，包括全路径* \_\_name\_\_ *# 主文件或者被导入文件* \_\_package\_\_ *# 模块所在的包  
   
#-- import语句from语句的as扩展* **import** modulename **as** name  
 **from** modulename **import** attrname **as** name  
   
*#-- 得到模块属性的几种方法 假设为了得到name属性的值* M.name  
 M.\_\_dict\_\_[**'name'**]  
 sys.modules[**'M'**].name  
 getattr(M, **'name'**)

## 类与面向对象

*#-- 最普通的类* **class** C1(C2, C3):  
 spam = 42 *# 数据属性* **def** \_\_init\_\_(self, name): *# 函数属性:构造函数* self.name = name  
 **def** \_\_del\_\_(self): *# 函数属性:析构函数* print(**"goodbey "**, self.name)   
 I1 = C1(**'bob'**)  
   
*#-- Python的类没有基于参数的函数重载* **class** FirstClass:  
 **def** test(self, string):  
 print(string)  
 **def** test(self): *# 此时类中只有一个test函数 即后者test(self) 它覆盖掉前者带参数的test函数* print(**"hello world"**)  
   
*#-- 子类扩展超类: 尽量调用超类的方法* **class** Manager(Person):  
 **def** giveRaise(self, percent, bonus = .10):  
 self.pay = int(self.pay\*(1 + percent + bonus)) *# 不好的方式 复制粘贴超类代码* Person.giveRaise(self, percent + bonus) *# 好的方式 尽量调用超类方法  
   
#-- 类内省工具* bob = Person(**'bob'**)  
 bob.\_\_class\_\_ *# <class 'Person'>* bob.\_\_class\_\_.\_\_name\_\_ *# 'Person'* bob.\_\_dict\_\_ *# {'pay':0, 'name':'bob', 'job':'Manager'}  
   
#-- 返回1中 数据属性spam是属于类 而不是对象* I1 = C1(**'bob'**); I2 = C2(**'tom'**) *# 此时I1和I2的spam都为42 但是都是返回的C1的spam属性* C1.spam = 24 *# 此时I1和I2的spam都为24* I1.spam = 3 *# 此时I1新增自有属性spam 值为2 I2和C1的spam还都为24  
   
#-- 类方法调用的两种方式* instance.method(arg...)  
 **class**.method(instance, arg...)  
   
*#-- 抽象超类的实现方法  
 # (1)某个函数中调用未定义的函数 子类中定义该函数* **def** delegate(self):  
 self.action() *# 本类中不定义action函数 所以使用delegate函数时就会出错  
 # (2)定义action函数 但是返回异常* **def** action(self):  
 **raise** NotImplementedError(**"action must be defined"**)  
 *# (3)上述的两种方法还都可以定义实例对象 实际上可以利用@装饰器语法生成不能定义的抽象超类* **from** abc **import** ABCMeta, abstractmethod  
 **class** Super(metaclass = ABCMeta):  
 @abstractmethod  
 **def** action(self): **pass** x = Super() *# 返回 TypeError: Can't instantiate abstract class Super with abstract methods action  
   
#-- # OOP和继承: "is - a"的关系* **class** A(B):  
 **pass** a = A()  
 isinstance(a, B) *# 返回True, A是B的子类 a也是B的一种  
 # OOP和组合: "has- a"的关系* **pass** *# OOP和委托: "包装"对象 在Python中委托通常是以"\_\_getattr\_\_"钩子方法实现的, 这个方法会拦截对不存在属性的读取  
 # 包装类(或者称为代理类)可以使用\_\_getattr\_\_把任意读取转发给被包装的对象* **class** wrapper:  
 **def** \_\_init\_\_(self, object):  
 self.wrapped = object  
 **def** \_\_getattr(self, attrname):  
 print(**'Trace: '**, attrname)  
 **return** getattr(self.wrapped, attrname)  
 *# 注:这里使用getattr(X, N)内置函数以变量名字符串N从包装对象X中取出属性 类似于X.\_\_dict\_\_[N]* x = wrapper([1, 2, 3])  
 x.append(4) *# 返回 "Trace: append" [1, 2, 3, 4]* x = wrapper({**'a'**:1, **'b'**:2})  
 list(x.keys()) *# 返回 "Trace: keys" ['a', 'b']  
   
#-- 类的伪私有属性:使用\_\_attr* **class** C1:  
 **def** \_\_init\_\_(self, name):  
 self.\_\_name = name *# 此时类的\_\_name属性为伪私有属性 原理 它会自动变成self.\_C1\_\_name = name* **def** \_\_str\_\_(self):  
 **return 'self.name = %s'** % self.\_\_name  
 I = C1(**'tom'**)  
 print(I) *# 返回 self.name = tom* I.\_\_name = **'jeey'** *# 这里无法访问 \_\_name为伪私有属性* I.\_C1\_\_name = **'jeey'** *# 这里可以修改成功 self.name = jeey  
   
#-- 类方法是对象:无绑定类方法对象 / 绑定实例方法对象* **class** Spam:  
 **def** doit(self, message):  
 print(message)  
 **def** selfless(message)  
 print(message)  
 obj = Spam()  
 x = obj.doit *# 类的绑定方法对象 实例 + 函数* x(**'hello world'**)  
 x = Spam.doit *# 类的无绑定方法对象 类名 + 函数* x(obj, **'hello world'**)  
 x = Spam.selfless *# 类的无绑定方法是函数 在3.0之前无效* x(**'hello world'**)  
   
*#-- 获取对象信息: 属性和方法* a = MyObject()  
 dir(a) *# 使用dir函数* hasattr(a, **'x'**) *# 测试是否有x属性或方法 即a.x是否已经存在* setattr(a, **'y'**, 19) *# 设置属性或方法 等同于a.y = 19* getattr(a, **'z'**, 0) *# 获取属性或方法 如果属性不存在 则返回默认值0  
 #这里有个小技巧，setattr可以设置一个不能访问到的属性，即只能用getattr获取* setattr(a, **"can't touch"**, 100) *# 这里的属性名带有空格，不能直接访问* getattr(a, **"can't touch"**, 0) *# 但是可以用getattr获取  
   
#-- 为类动态绑定属性或方法: MethodType方法  
 # 一般创建了一个class的实例后, 可以给该实例绑定任何属性和方法, 这就是动态语言的灵活性* **class** Student(object):  
 **pass** s = Student()  
 s.name = **'Michael'** *# 动态给实例绑定一个属性* **def** set\_age(self, age): *# 定义一个函数作为实例方法* self.age = age  
 **from** types **import** MethodType  
 s.set\_age = MethodType(set\_age, s) *# 给实例绑定一个方法 类的其他实例不受此影响* s.set\_age(25) *# 调用实例方法* Student.set\_age = MethodType(set\_age, Student) *# 为类绑定一个方法 类的所有实例都拥有该方法*

## 类的高级话题

*#-- 多重继承: "混合类", 搜索方式"从下到上 从左到右 广度优先"* **class** A(B, C):  
 **pass***#-- 类的继承和子类的初始化  
 # 1.子类定义了\_\_init\_\_方法时，若未显示调用基类\_\_init\_\_方法，python不会帮你调用。  
 # 2.子类未定义\_\_init\_\_方法时，python会自动帮你调用首个基类的\_\_init\_\_方法，注意是首个。  
 # 3.子类显示调用基类的初始化函数：* **class** FooParent(object):  
 **def** \_\_init\_\_(self, a):  
 self.parent = **'I\'m the Parent.'** print(**'Parent:a='** + str(a))  
 **def** bar(self, message):  
 print(message + **' from Parent'**)  
 **class** FooChild(FooParent):  
 **def** \_\_init\_\_(self, a):  
 FooParent.\_\_init\_\_(self, a)  
 print(**'Child:a='** + str(a))  
 **def** bar(self, message):  
 FooParent.bar(self, message)  
 print(message + **' from Child'**)  
 fooChild = FooChild(10)  
 fooChild.bar(**'HelloWorld'**)  
   
*#-- #实例方法 / 静态方法 / 类方法* **class** Methods:  
 **def** imeth(self, x): print(self, x) *# 实例方法：传入的是实例和数据，操作的是实例的属性* **def** smeth(x): print(x) *# 静态方法：只传入数据 不传入实例，操作的是类的属性而不是实例的属性* **def** cmeth(cls, x): print(cls, x) *# 类方法：传入的是类对象和数据* smeth = staticmethod(smeth) *# 调用内置函数，也可以使用@staticmethod* cmeth = classmethod(cmeth) *# 调用内置函数，也可以使用@classmethod* obj = Methods()  
 obj.imeth(1) *# 实例方法调用 <\_\_main\_\_.Methods object...> 1* Methods.imeth(obj, 2) *# <\_\_main\_\_.Methods object...> 2* Methods.smeth(3) *# 静态方法调用 3* obj.smeth(4) *# 这里可以使用实例进行调用* Methods.cmeth(5) *# 类方法调用 <class '\_\_main\_\_.Methods'> 5* obj.cmeth(6) *# <class '\_\_main\_\_.Methods'> 6  
   
#-- 函数装饰器:是它后边的函数的运行时的声明 由@符号以及后边紧跟的"元函数"(metafunction)组成* @staticmethod  
 **def** smeth(x): print(x)  
 *# 等同于:* **def** smeth(x): print(x)  
 smeth = staticmethod(smeth)  
 *# 同理* @classmethod  
 **def** cmeth(cls, x): print(x)  
 *# 等同于* **def** cmeth(cls, x): print(x)  
 cmeth = classmethod(cmeth)  
   
*#-- 类修饰器:是它后边的类的运行时的声明 由@符号以及后边紧跟的"元函数"(metafunction)组成* **def** decorator(aClass):.....  
 @decorator  
 **class** C:....  
 *# 等同于:* **class** C:....  
 C = decorator(C)  
   
*#-- 限制class属性: \_\_slots\_\_属性* **class** Student:  
 \_\_slots\_\_ = (**'name'**, **'age'**) *# 限制Student及其实例只能拥有name和age属性  
 # \_\_slots\_\_属性只对当前类起作用, 对其子类不起作用  
 # \_\_slots\_\_属性能够节省内存  
 # \_\_slots\_\_属性可以为列表list，或者元组tuple  
   
#-- 类属性高级话题: @property  
 # 假设定义了一个类:C，该类必须继承自object类，有一私有变量\_x* **class** C(object):  
 **def** \_\_init\_\_(self):  
 self.\_\_x = **None** *# 第一种使用属性的方法* **def** getx(self):  
 **return** self.\_\_x  
 **def** setx(self, value):  
 self.\_\_x = value  
 **def** delx(self):  
 **del** self.\_\_x  
 x = property(getx, setx, delx, **''**)  
 *# property函数原型为property(fget=None,fset=None,fdel=None,doc=None)  
 # 使用* c = C()  
 c.x = 100 *# 自动调用setx方法* y = c.x *# 自动调用getx方法* **del** c.x *# 自动调用delx方法  
 # 第二种方法使用属性的方法* @property  
 **def** x(self):  
 **return** self.\_\_x  
 @x.setter  
 **def** x(self, value):  
 self.\_\_x = value  
 @x.deleter  
 **def** x(self):  
 **del** self.\_\_x  
 *# 使用* c = C()  
 c.x = 100 *# 自动调用setter方法* y = c.x *# 自动调用x方法* **del** c.x *# 自动调用deleter方法  
   
#-- 定制类: 重写类的方法  
 # (1)\_\_str\_\_方法、\_\_repr\_\_方法: 定制类的输出字符串  
 # (2)\_\_iter\_\_方法、next方法: 定制类的可迭代性* **class** Fib(object):  
 **def** \_\_init\_\_(self):  
 self.a, self.b = 0, 1 *# 初始化两个计数器a，b* **def** \_\_iter\_\_(self):  
 **return** self *# 实例本身就是迭代对象，故返回自己* **def** next(self):  
 self.a, self.b = self.b, self.a + self.b  
 **if** self.a > 100000: *# 退出循环的条件* **raise** StopIteration()  
 **return** self.a *# 返回下一个值* **for** n **in** Fib():  
 print(n) *# 使用  
 # (3)\_\_getitem\_\_方法、\_\_setitem\_\_方法: 定制类的下标操作[] 或者切片操作slice* **class** Indexer(object):  
 **def** \_\_init\_\_(self):  
 self.data = {}  
 **def** \_\_getitem\_\_(self, n): *# 定义getitem方法* print(**'getitem:'**, n)   
 **return** self.data[n]  
 **def** \_\_setitem\_\_(self, key, value): *# 定义setitem方法* print(**'setitem:key = {0}, value = {1}'**.format(key, value))  
 self.data[key] = value  
 test = Indexer()  
 test[0] = 1; test[3] = **'3'** *# 调用setitem方法* print(test[0]) *# 调用getitem方法  
 # (4)\_\_getattr\_\_方法: 定制类的属性操作* **class** Student(object):  
 **def** \_\_getattr\_\_(self, attr): *# 定义当获取类的属性时的返回值* **if** attr==**'age'**:  
 **return** 25 *# 当获取age属性时返回25* **raise** AttributeError(**'object has no attribute: %s'** % attr)  
 *# 注意: 只有当属性不存在时 才会调用该方法 且该方法默认返回None 需要在函数最后引发异常* s = Student()  
 s.age *# s中age属性不存在 故调用\_\_getattr\_\_方法 返回25  
 # (5)\_\_call\_\_方法: 定制类的'可调用'性* **class** Student(object):  
 **def** \_\_call\_\_(self): *# 也可以带参数* print(**'Calling......'**)  
 s = Student()  
 s() *# s变成了可调用的 也可以带参数* callable(s) *# 测试s的可调用性 返回True  
 # (6)\_\_len\_\_方法：求类的长度* **def** \_\_len\_\_(self):  
 **return** len(self.data)  
   
*#-- 动态创建类type()  
 # 一般创建类 需要在代码中提前定义* **class** Hello(object):  
 **def** hello(self, name=**'world'**):  
 print(**'Hello, %s.'** % name)  
 h = Hello()  
 h.hello() *# Hello, world* type(Hello) *# Hello是一个type类型 返回<class 'type'>* type(h) *# h是一个Hello类型 返回<class 'Hello'>  
 # 动态类型语言中 类可以动态创建 type函数可用于创建新类型* **def** fn(self, name=**'world'**): *# 先定义函数* print(**'Hello, %s.'** % name)  
 Hello = type(**'Hello'**, (object,), dict(hello=fn))   
 *# 创建Hello类 type原型: type(name, bases, dict)* h = Hello() *# 此时的h和上边的h一致*

## 异常相关

*#-- #捕获异常:* **try**:  
 **except**: *# 捕获所有的异常 等同于except Exception:* **except** name: *# 捕获指定的异常* **except** name, value: *# 捕获指定的异常和额外的数据(实例)* **except** (name1, name2):  
 **except** (name1, name2), value:  
 **except** name4 **as** X:  
 **else**: *# 如果没有发生异常* **finally**: *# 总会执行的部分  
 # 引发异常: raise子句(raise IndexError)* **raise** <instance> *# raise instance of a class, raise IndexError()* **raise** <**class**> *# make and raise instance of a class, raise IndexError* **raise** *# reraise the most recent exception  
   
#-- Python3.x中的异常链: raise exception from otherException* **except** Exception **as** X:  
 **raise** IndexError(**'Bad'**) **from** X  
   
*#-- assert子句: assert <test>, <data>* **assert** x < 0, **'x must be negative'***#-- with/as环境管理器:作为常见的try/finally用法模式的替代方案* **with** expression [**as** variable], expression [**as** variable]:  
 *# 例子:* **with** open(**'test.txt'**) **as** myfile:  
 **for** line **in** myfile: print(line)  
 *# 等同于:* myfile = open(**'test.txt'**)  
 **try**:  
 **for** line **in** myfile: print(line)  
 **finally**:  
 myfile.close()  
   
*#-- 用户自定义异常: class Bad(Exception):.....* **"""  
 Exception超类 / except基类即可捕获到其所有子类  
 Exception超类有默认的打印消息和状态 当然也可以定制打印显示:  
 """  
 class** MyBad(Exception):  
 **def** \_\_str\_\_(self):  
 **return '定制的打印消息'  
 try**:  
 MyBad()  
 **except** MyBad **as** x:  
 print(x)  
   
*#-- 用户定制异常数据* **class** FormatError(Exception):  
 **def** \_\_init\_\_(self, line ,file):  
 self.line = line  
 self.file = file  
 **try**:  
 **raise** FormatError(42, **'test.py'**)  
 **except** FormatError **as** X:  
 print(**'Error at '**, X.file, X.line)  
 *# 用户定制异常行为(方法):以记录日志为例* **class** FormatError(Exception):  
 logfile = **'formaterror.txt'  
 def** \_\_init\_\_(self, line ,file):  
 self.line = line  
 self.file = file  
 **def** logger(self):  
 open(self.logfile, **'a'**).write(**'Error at '**, self.file, self.line)  
 **try**:  
 **raise** FormatError(42, **'test.py'**)  
 **except** FormatError **as** X:  
 X.logger()  
   
*#-- 关于sys.exc\_info:允许一个异常处理器获取对最近引发的异常的访问* **try**:  
 ......  
 **except**:  
 *# 此时sys.exc\_info()返回一个元组(type, value, traceback)  
 # type:正在处理的异常的异常类型  
 # value:引发的异常的实例  
 # traceback:堆栈信息  
   
#-- 异常层次* BaseException  
 +-- SystemExit  
 +-- KeyboardInterrupt  
 +-- GeneratorExit  
 +-- Exception  
 +-- StopIteration  
 +-- ArithmeticError  
 +-- AssertionError  
 +-- AttributeError  
 +-- BufferError  
 +-- EOFError  
 +-- ImportError  
 +-- LookupError  
 +-- MemoryError  
 +-- NameError  
 +-- OSError  
 +-- ReferenceError  
 +-- RuntimeError  
 +-- SyntaxError  
 +-- SystemError  
 +-- TypeError  
 +-- ValueError  
 +-- Warning

## Unicode和字节字符串

*#-- Python的字符串类型* **"""Python2.x"""** *# 1.str表示8位文本和二进制数据  
 # 2.unicode表示宽字符Unicode文本* **"""Python3.x"""** *# 1.str表示Unicode文本（8位或者更宽）  
 # 2.bytes表示不可变的二进制数据  
 # 3.bytearray是一种可变的bytes类型  
   
#-- 字符编码方法* **"""ASCII"""** *# 一个字节，只包含英文字符，0到127，共128个字符，利用函数可以进行字符和数字的相互转换* ord(**'a'**) *# 字符a的ASCII码为97，所以这里返回97* chr(97) *# 和上边的过程相反，返回字符'a'* **"""Latin-1"""** *# 一个字节，包含特殊字符，0到255，共256个字符，相当于对ASCII码的扩展* chr(196) *# 返回一个特殊字符：Ä* **"""Unicode"""** *# 宽字符，一个字符包含多个字节，一般用于亚洲的字符集，比如中文有好几万字* **"""UTF-8"""** *# 可变字节数，小于128的字符表示为单个字节，128到0X7FF之间的代码转换为两个字节，0X7FF以上的代码转换为3或4个字节  
 # 注意：可以看出来，ASCII码是Latin-1和UTF-8的一个子集  
 # 注意：utf-8是unicode的一种实现方式，unicode、gbk、gb2312是编码字符集  
   
#-- 查看Python中的字符串编码名称，查看系统的编码* **import** encodings  
 help(encoding)  
 **import** sys  
 sys.platform *# 'win64'* sys.getdefaultencoding() *# 'utf-8'* sys.getdefaultencoding() *# 返回当前系统平台的编码类型* sys.getsizeof(object) *# 返回object占有的bytes的大小  
   
#-- 源文件字符集编码声明: 添加注释来指定想要的编码形式 从而改变默认值 注释必须出现在脚本的第一行或者第二行* **"""说明：其实这里只会检查#和coding:utf-8，其余的字符都是为了美观加上的"""** *# \_\*\_ coding: utf-8 \_\*\_  
 # coding = utf-8  
   
#-- #编码: 字符串 --> 原始字节 #解码: 原始字节 --> 字符串  
   
#-- Python3.x中的字符串应用* s = **'...'** *# 构建一个str对象，不可变对象* b = **b'...'** *# 构建一个bytes对象，不可变对象* s[0], b[0] *# 返回('.', 113)* s[1:], b[1:] *# 返回('..', b'..')* B = **B"""  
 xxxx  
 yyyy  
 """** *# B = b'\nxxxx\nyyyy\n'  
 # 编码，将str字符串转化为其raw bytes形式：* str.encode(encoding = **'utf-8'**, errors = **'strict'**)  
 bytes(str, encoding)  
 *# 编码例子：* S = **'egg'** S.encode() *# b'egg'* bytes(S, encoding = **'ascii'**) *# b'egg'  
 # 解码，将raw bytes字符串转化为str形式：* bytes.decode(encoding = **'utf-8'**, errors = **'strict'**)  
 str(bytes\_or\_buffer[, encoding[, errors]])  
 *# 解码例子：* B = **b'spam'** B.decode() *# 'spam'* str(B) *# "b'spam'"，不带编码的str调用，结果为打印该bytes对象* str(B, encoding = **'ascii'**)*# 'spam'，带编码的str调用，结果为转化该bytes对象  
   
#-- Python2.x的编码问题* u = **u'汉'** print repr(u) *# u'\xba\xba'* s = u.encode(**'UTF-8'**)  
 print repr(s) *# '\xc2\xba\xc2\xba'* u2 = s.decode(**'UTF-8'**)  
 print repr(u2) *# u'\xba\xba'  
 # 对unicode进行解码是错误的* s2 = u.decode(**'UTF-8'**) *# UnicodeEncodeError: 'ascii' codec can't encode characters in position 0-1: ordinal not in range(128)  
 # 同样，对str进行编码也是错误的* u2 = s.encode(**'UTF-8'**) *# UnicodeDecodeError: 'ascii' codec can't decode byte 0xc2 in position 0: ordinal not in range(128)  
   
#-- bytes对象* B = **b'abc'** B = bytes(**'abc'**, **'ascii'**)  
 B = bytes([97, 98, 99])  
 B = **'abc'**.encode()  
 *# bytes对象的方法调用基本和str类型一致 但:B[0]返回的是ASCII码值97, 而不是b'a'  
   
#-- #文本文件: 根据Unicode编码来解释文件内容，要么是平台的默认编码，要么是指定的编码类型  
 # 二进制文件：表示字节值的整数的一个序列 open('bin.txt', 'rb')  
   
#-- Unicode文件* s = **'A\xc4B\xe8C'** *# s = 'A?BèC' len(s) = 5  
 #手动编码* l = s.encode(**'latin-1'**) *# l = b'A\xc4B\xe8C' len(l) = 5* u = s.encode(**'utf-8'**) *# u = b'A\xc3\x84B\xc3\xa8C' len(u) = 7  
 #文件输出编码* open(**'latindata'**, **'w'**, encoding = **'latin-1'**).write(s)  
 l = open(**'latindata'**, **'rb'**).read() *# l = b'A\xc4B\xe8C' len(l) = 5* open(**'uft8data'**, **'w'**, encoding = **'utf-8'**).write(s)  
 u = open(**'uft8data'**, **'rb'**).read() *# u = b'A\xc3\x84B\xc3\xa8C' len(u) = 7  
 #文件输入编码* s = open(**'latindata'**, **'r'**, encoding = **'latin-1'**).read() *# s = 'A?BèC' len(s) = 5* s = open(**'latindata'**, **'rb'**).read().decode(**'latin-1'**) *# s = 'A?BèC' len(s) = 5* s = open(**'utf8data'**, **'r'**, encoding = **'utf-8'**).read() *# s = 'A?BèC' len(s) = 5* s = open(**'utf8data'**, **'rb'**).read().decode(**'utf-8'**) *# s = 'A?BèC' len(s) = 5*

## 其他

*#-- 60个字符解决FizzBuzz:* **"""写一个程序, 打印数字1到100, 3的倍数打印“Fizz”来替换这个数, 5的倍数打印“Buzz”, 既是3又是5的倍数的打印“FizzBuzz”"""  
 for** x **in** range(101):  
 print(**"fizz"**[x%3\*4::]+**"buzz"**[x%5\*4::] **or** x) *# 解释:最主要用到列表(字符串)的子表  
   
#-- Python实现任意深度的赋值 例如a[0] = 'value1'; a[1][2] = 'value2'; a[3][4][5] = 'value3'* **class** MyDict(dict):  
 **def** \_\_setitem\_\_(self, key, value): *# 该函数不做任何改动 这里只是为了输出* print(**'setitem:'**, key, value, self)  
 super().\_\_setitem\_\_(key, value)  
 **def** \_\_getitem\_\_(self, item): *# 主要技巧在该函数* print(**'getitem:'**, item, self) *# 输出信息  
 # 基本思路: a[1][2]赋值时 需要先取出a[1] 然后给a[1]的[2]赋值* **if** item **not in** self: *# 如果a[1]不存在 则需要新建一个dict 并使得a[1] = dict* temp = MyDict() *# 新建的dict: temp* super().\_\_setitem\_\_(item, temp) *# 赋值a[1] = temp* **return** temp *# 返回temp 使得temp[2] = value有效* **return** super().\_\_getitem\_\_(item) *# 如果a[1]存在 则直接返回a[1]  
 # 例子:* test = MyDict()  
 test[0] = **'test'** print(test[0])  
 test[1][2] = **'test1'** print(test[1][2])  
 test[1][3] = **'test2'** print(test[1][3])  
   
*#-- Python中的多维数组* lists = [0] \* 3 *# 扩展list，结果为[0, 0, 0]* lists = [[]] \* 3 *# 多维数组，结果为[[], [], []]，但有问题，往下看* lists[0].append(3) *# 期望看到的结果[[3], [], []]，实际结果[[3], [3], [3]]，原因：list\*n操作，是浅拷贝，如何避免？往下看* lists = [[] **for** i **in** range(3)] *# 多维数组，结果为[[], [], []]* lists[0].append(3) *# 结果为[[3], [], []]* lists[1].append(6) *# 结果为[[3], [6], []]* lists[2].append(9) *# 结果为[[3], [6], [9]]* lists = [[[] **for** j **in** range(4)] **for** i **in** range(3)]  
 lists *# 3行4列，且每一个元素为[]*