Python面试题

1. 基础

|  |
| --- |
| **什么是Python？（为什么选择学Python？）**  1. Python是一种解释型语言。这就是说，与C语言和C的衍生语言不同，Python代码在运行之前不需要编译。其他解释型语言还包括PHP和Ruby。  2. Python是动态类型语言，指的是你在声明变量时，不需要说明变量的类型。你可以直接编写类似x=111和x="I'm a string"这样的代码，程序不会报错。  3. Python非常适合面向对象的编程（OOP），因为它支持通过组合（composition）与继承（inheritance）的方式定义类（class）。Python中没有访问说明符（access specifier，类似C++中的public和private），这么设计的依据是“大家都是成年人了”。  4. 在Python语言中，函数是第一类对象（first-class objects）。这指的是它们可以被指定给变量，函数既能返回函数类型，也可以接受函数作为输入。类（class）也是第一类对象。  5. Python代码编写快，但是运行速度比编译语言通常要慢。好在Python允许加入基于C语言编写的扩展，因此我们能够优化代码，消除瓶颈，这点通常是可以实现的。numpy就是一个很好地例子，它的运行速度真的非常快，因为很多算术运算其实并不是通过Python实现的。  6. Python用途非常广泛——网络应用，自动化，科学建模，大数据应用，等等。它也常被用作“胶水语言”，帮助其他语言和组件改善运行状况。  7. Python让困难的事情变得容易，因此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。  8. 优雅，简单，开发效率高，第三方包多，高级语言，可移植，可嵌入，胶水语言，脚本语言 |
| **解释型编译型的区别**  1. 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。  解释型语言在运行程序的时候才会进行翻译，边翻译边运行。  2. 编译型语言在程序运行之前，需要一个专门的编译过程，把程序编译成机器语言，会生成一个编译文件，直接运行该编译文件即可。若源代码没有修改，再次运行程序时，无需再次编译。  3. 编译型：一次性将所有程序编译成二进制文件，优点是运行速度快，缺点是开发效率低，不能跨平台（C、C++）  4. 解释型：当程序执行时，一行一行的解释，优点时开发效率高，跨平台，缺点是运行速度慢（python、php） |
| **Python解释器有哪些？** CPython 当我们从Python官方网站下载并安装好Python 2.7后，我们就直接获得了一个官方版本的解释器：CPython。这个解释器是用C语言开发的，所以叫CPython。在命令行下运行python就是启动CPython解释器。CPython是使用最广的Python解释器。教程的所有代码也都在CPython下执行。 IPython IPython是基于CPython之上的一个交互式解释器，也就是说，IPython只是在交互方式上有所增强，但是执行Python代码的功能和CPython是完全一样的。好比很多国产浏览器虽然外观不同，但内核其实都是调用了IE。  CPython用>>>作为提示符，而IPython用In [序号]:作为提示符。 PyPy PyPy是另一个Python解释器，它的目标是执行速度。PyPy采用[JIT技术](http://en.wikipedia.org/wiki/Just-in-time_compilation" \t "_blank)，对Python代码进行动态编译（注意不是解释），所以可以显著提高Python代码的执行速度。  绝大部分Python代码都可以在PyPy下运行，但是PyPy和CPython有一些是不同的，这就导致相同的Python代码在两种解释器下执行可能会有不同的结果。如果你的代码要放到PyPy下执行，就需要了解[PyPy和CPython的不同点](http://pypy.readthedocs.org/en/latest/cpython_differences.html" \t "_blank)。 Jython Jython是运行在Java平台上的Python解释器，可以直接把Python代码编译成Java字节码执行。 IronPython IronPython和Jython类似，只不过IronPython是运行在微软.Net平台上的Python解释器，可以直接把Python代码编译成.Net的字节码。 |
| **关于Python程序的运行，有什么手段能提升性能。**  使用多进程，充分利用机器的多核性能。  对于性能意向比较大的部分代码，可以使用c或c++编写。  对于IO阻塞造成的性能影响，可以使用IO多路复用来解决。  尽量使用Python的内置函数。  尽量使用局部变量。 |
| **尽可能多的写出列表的去重方案，另写出去重的同时保持顺序不变的方案**  # 第一种  def dedupe(items):  seen = set()  for item in items:  if item not in seen:  yield item  seen.add(item)  a = [1, 5, 2, 1, 9, 1, 5, 10]  print(list(dedupe(a)))  # 第二种  a\_list = [1, 4, 4, 2, 1, 7]  b\_list = []  for a in a\_list:  if a not in b\_list:  b\_list.append(a)  print(b\_list) |
| **将[{1:”a”},{2:”b”}]转换为[{‘value’:’a’, ‘key’:1},{‘value’:’b’, ‘key’:2}]**  [{‘key’:key,’value’:value} for key,value in dict.items()] |
| **如何构造有序字典？**  import collections  # 通过OrderedDict类创建的字典是有序的  dic = collections.OrderedDict() |
| **在collections库中有哪些常用的集合，简单说明一下它们的作用？**  **namedtuple(给元组中的元素命名)**  **Counter(统计字典中的关键字的值出现的次数)**  **OrderedDict(使传入的key值按照传入的顺序排列)**  **deque(构造一个双端的队列， 一端进入 一端出，最大长度受限，可以节省空间)** |
| **数组类型list, set, tuple, dict有什么区别，哪两个类型可以相互做等价交换。例如list\_a = list(tuple(list\_a))**  dict1 = {'name': 'zhangsan', 'age': 20}  dict1 = dict(tuple(dict1.items()))  print(dict1) |
| **写入一个函数,输入一个字符串,返回逆序排列的结果,如:string\_reverse(“abcdef”),返回”fedcba”**  def string\_reverse (a):  return a[::-1] |
| **简述Python参数传递机制（值传递（对应不可变类型） 址传递（对应可变类型））?**  python不允许程序员选择采用传值还是传引用.Python参数传递采用的肯定是“传对象引用”的方式。这种方式相当于传值和传引用的一种综合。如果函数收到的是一个可变对象（比如字典或者列表）的引用，就能修改对象的原始值－－相当于通过“传引用”来传递对象。如果函数收到的是一个不可变对象（比如数字、字符或者元组）的引用，就不能直接修改原始对象－－相当于通过“传值'来传递对象 |
| **如何安装Python环境?**  1. 从Python官网下载对应的python版本的可执行文件/源码进行安装  2. 下载Anaconda集成环境安装 |
| **集成环境的优缺点**  优点:自带很多包,省去了安装过程,使用更简单  缺点:因为带有大量的包,所以使用PyCharm创建项目的时候加载速度比较慢 |
| **如何在Windows/ Ubuntu系统中安装双环境（同时安装python2和python3）？**  Windows安装标准环境：从python官网下载对应安装包双击安装  Ubuntu安装标准环境：apt-get install python  如果已经安装了Anaconda python3,终端执行命令即可安装python2  conda create --name python27 python=2.7.12 |
| **为什么使用虚拟环境？**  每个项目可能需要使用不同版本的第三方库，为了使环境互相独立，可以考虑使用虚拟环境 |
| **如何创建指定python版本的虚拟环境?**  pip install virtualenv 安装虚拟环境  cd my\_project\_dir 进入项目  virtualenv venv 创建虚拟环境  virtualenv -p /usr/bin/python2.7 venv 创建指定python版本的虚拟环境  source venv/bin/activate 激活环境  . venv/bin/deactivate 取消激活环境 |
| **你使用的Python编程工具是什么?**  Pycharm\Sublime\VSCode\nodepad++.... |
| **什么是面向对象(Java,PHP,JS,Python)？什么是面向过程（C）？**   1. 面向过程”是一种是事件为中心的编程思想。就是分析出解决问题所需的步骤，然后用函数把这写步骤实现，并按顺序调用。(简单) 2. ”面向对象“是以“对象”为中心的编程思想。（万物皆对象）   强类型语言（strong） 和 弱类型语言（weak）  强类型：定义变量的时候必须指定类型，例如 int a = 10 C / C++ / C# / Java  弱类型：定义变量的时候不需要指定类型，由语言自己根据数值自行判断 age = 10  Python语言在运行时runtime 为强类型语言 |
| **面向对象的特点?**  封装 继承 多态 |
| **什么是多态和多态性？**  多态：同一种事物的多种形态，动物分为人类，猪类（在定义角度） 多态性：一种调用方式，不同的执行效果（多态性） |
| **继承的特点有哪些？**  子类拥有父类的属性和方法  子类拥有自己的属性和方法  子类可以重写父类的方法 |
| **type的用法有哪些?**   1. 查看对象的类型 2. 动态创建类   参数1:类名 类型字符串  参数2:元组 父类  参数3:字典 类的属性  注意事项:  type的第2个参数，元组中是父类的名字，而不是字符串  添加的属性是类属性，并不是实例属性  print(type('zhangsan'))  People = type('People', (object,), {'name': 'zhangsan', 'age': 20})  a = People ()  print(a.name) |
| **什么是元类?**  1.元类简单来说就是类的类  2.type就是创建所有类的元类 |
| **type为什么小写而不大写?**  我认为是因为统一命名规范 |
| **什么是高阶函数**  就是把函数当成参数传递的一种函数  def add(x, y, func):  return func(x) + func(y)  print(add(-2, 14, abs)) |
| **\_\_slots\_\_有什么用?**  1.限定用户能动态添加哪些属性  2.仅对当前类实例起作用，对继承的子类是不起作用的  class People(object):  \_\_slots\_\_ = ['name']  p = People()  p.name = 'zhangsan' *# 可以添加*  p.age = 20 *# 不能添加，会报错*  print(p.name, p.age) |
| **Import的基本用法**  1.Import 包名/函数/类/变量……  2.From 模块名 import 包名/函数/类/变量……  注意:最好不用import \*,最好用什么导入什么 |
| **循环导入怎么解决?**  效果:A导入B,B导入A,结果提示'module' object has no attribute  解决:可以引入一个新的文件为C,C中同时包含A和B中的需要引入的代码,A和B只需要引入C即可  注意:最好把公共的/用处频繁的函数单独放到一个工具类中,可以防止循环导入的发生 |
| **循环引用怎么解决?**  效果:A引用B,B引用A,结果是A和B都释放不掉(导致内存泄漏：明明内存看起来够用，但是实际提示不够)  解决:A或B用完对方以后手动del即可 |
| **什么叫耦合度?**  耦合度就是两个模块/文件/代码的关联性(依赖性)  注意:耦合度越大越不好,例如我需要引入一个功能,但是需要导入5个文件 |
| **什么是命名空间?**  同一个路径下能否创建两个同名文件?不能  命名空间说白了就是文件夹路径 |
| **同步和异步的区别?**  所谓同步，就是发出一个功能调用时，在没有得到结果之前，该调用就不返回或继续执行后续操作  异步与同步相对，当一个异步过程调用发出后，调用者在没有得到结果之前，就可以继续执行后续操作。当这个调用完成后，一般通过状态、通知和回调来通知调用者。 |
| **串行和并行和并发的区别?**  串行类似于同步，两个或多个时间按照顺序发生  并行是指两个或者多个事件在同一时刻发生；而并发是指两个或多个事件在同一时间间隔发生  并发是同时处理(dealing)很多的事情，并行是同时做(doing)很多的事情。不同，但也相关。一个是关于代码结构，一个是关于代码执行。并发为可能的（不是必须的）并行问题提供了一种解决方案 |
| **局部变量和全局变量的生命周期(作用域)?**  局部变量:在函数内定义的变量都是局部变量,从定义该变量开始,到离开该变量所在函数  全局变量:在函数外定义的变量都是全部变量,或者在函数内定义的变量但是用global来修饰也属于全局变量,它的生命周期从定义该变量开始,到整个文件结束  注意:如果使用不当,可能会出现局部变量覆盖全局变量的问题  尽可能使用局部变量替代全局变量，便于维护，提高性能并节省内存  .  在这里,外部的a是全局变量,内部的a是局部变量,两个变量同名则会产生覆盖.如果不想被覆盖,需要在函数内指定a是全局变量 |
| **写出下面的运行结果**  a = 10  **def** func(a):  a = 20  func(a)  print(a) **# 10**  a = []  **def** func(a):  a.append(20)  func(a)  print(a) **# [20]** |
| **有如下两个类，请问A.name是什么**  **class** A:  name = **'a'**  **class** B(A):  **pass**  b = B()  b.name = **'b'**  print(A.name)  **a** |
| **Python2和Python3的区别？**  **性**能：Py3.0运行 pystone benchmark的速度比Py2.5慢30%  **编码：**Py3.X源码文件默认使用utf-8编码  **语法：**去除了<>，全部改用!=，关键词加入as 和with，还有True,False,None  **字符串和字节串：**现在字符串只有str一种类型，但它跟2.x版本的unicode几乎一样  **数据类型：**Py3.X去除了long类型，现在只有一种整型——int，但它的行为就像2.X版本的long |
| **什么是赋值?什么是深拷贝?浅拷贝?区别是?**  赋值:例如a=b,将b的内容赋值给a,这时a和b指向同一块内存空间,因此不管是改a还是改b,都会影响内容  浅拷贝:a=copy.copy(b),拷贝b的对象给a,b中的第一层数据会被拷贝，其它层不拷贝。  深拷贝:a=copy.deepcopy(b) 拷贝b的对象给a,而且b里面引入的其它对象也会拷贝 |
| **列举您使用过的python中的编码方式**  UTF-8，ASCII，GBK，gb18030 |
| **Ascii、unicode、utf8、gbk的区别**  ascii 是最早美国用的标准信息交换码，把所有的字母的大小写，各种符号用 二进制来表示，共有256中，加入些拉丁文等字符，1bytes代表一个字符，  Unicode是为了统一世界各国语言的不用，统一用2个bytes代表一个字符，可以表达2\*\*16=65556个，称为万国语言，特点：速度快，但浪费空间，  可以用在内存处理中，兼容了utf-8，gbk，ASCII，  utf-8 为了改变Unicode的这种缺点，规定1个英文字符用1个字节表示，1个中文字符用3个字节表示，特点；节省空间，速度慢，用在硬盘数据传输，网络数据传输，相比硬盘和网络速度，体现不出来的，  gbk  是中文的字符编码，用2个字节代表一个字符， |
| **编码格式之间怎么转换？**  python3-str2  encode 负责字符到字节的编码转换。默认使用 UTF-8 编码准换。  decode 负责字节到字符的解码转换，通用使用 UTF-8 编码格式进行转换。 |
| **python3.5语言中enumerate的意思是**  对于一个可迭代的（iterable）/可遍历的对象（如列表、字符串），enumerate将其组成一个索引序列，利用它可以同时获得索引和值  Enumerate常用于在for循环中得到元素的索引值 |
| 八进制转二进制  143 ---- 001 100 011  152 ---- 001 101 010  375 ---- 011 111 101  726 ---- 111 010 110  二进制转八进制  110 001 101 ----- 615  000 111 000 010 ----- 0702  二进制转十六进制 8421 1-9 ABCDEF  0110 0100 1100 ----- 64C  0110 0110 0001 ----- 661  八进制转十六进制  476 ----- 0001 00 11 1 110 ----- 13E  二进制转10进制  1100011001 ---- 2^0\*1 + 2^1\*0 + 2^2\*0 + 2^3 \* 1 + 2^4\*1 + 2^5\*0 + 2^6\*0 + 2^7\*0 + 2^8 \*1 + 2^9\*1  10进制转二进制  15 ---- 1111  17 ---- 10001  35 ---- 100011 |
| 二进制转8进制/10进制/16进制  十进制转2进制/8进制/18进制  例子:001 1001  转10进制: 2^0+2^3+2^4 = 25  转8进制:421按位乘以011 拼接 421按位乘以001 = 31  转16进制:8421按位乘以0001 拼接 8421按位乘以 1001 = 19  例子:十进制的35  转2进制:除以2求余数,从下网上读取余数即可 结果是100011  转8进制:先转2进制,再转8进制  转16进制:先转2进制,再转16进制 |
| **如何将1314这个十进制数转换为二进制数?请写出思路和步骤**  十进制数除以2求余数，当余数为0时将余数从下往上写即可。 |
| **如何查看中央处理器,内存,硬盘的使用情况？**  psutil是一个跨平台库，能够轻松实现获取系统运行的进程和系统利用率(CPU，内存，磁盘，网络等)信息，主要应用于系统监控，分析和限制系统资源及进程的管理  <https://blog.csdn.net/linux_hacher/article/details/72628208?locationNum=6&fps=1>  <https://www.cnblogs.com/Missowalker/p/7921888.html> |
| **用4,9,2,7四个数字，可以使用+、-、\*和/，每个数字使用一次，使表达式的结果为24，表达式是**  9+7+4\*2 |
| 阅读以下Python程序  for i in range(5,0,-1):    print(i)  请在下面写出打印结果  5 4 3 2 1 |
| **Python中如何使用异常捕获**  Try:  可能出错的代码  Except 错误类型1 as e:  Print(e)  Except 错误类型2 as 2:  Print(2)  Else:  没有以上错误  Finally:  不管有没有错误都走这里  raise assert这俩词和异常有关 |
| **常用的错误异常类型有哪些？**  TypeError, IOError, IndexError, ValueError… |
| **数据分析用过哪些常用的库**  Numpy np.array()  Pandas pd.DataFrame pd. series  Matplotlib 饼状图 柱状图 折线图… |
| **如何处理缺失数据？**  1.忽略元组  当缺少类别标签时通常这样做（假定挖掘任务涉及分类时），除非元组有多个属性缺失值，否则该方法不是很有效。当每个属性缺少值的百分比变化很大时，它的性能特别差。  2.人工填写缺失值  一般该方法很费时，并且当数据集很大，缺少很多值时，该方法可能行不通。  3.使用一个全局常量填充缺失值  将缺失的属性值用同一个常数(如“Unknown”或 负无穷)替换。如果缺失值都用“unknown”替换，则挖掘程序可能会认为它们形成一个有趣的概念，因为它们都具有相同的值“unknown”。因此，虽然该方法很简单，但是它十分不可靠。  4.使用与给定元组属同一类的所有样本的属性均值  例如：将顾客按照credit\_risk分类，则使用具有相同信用度的给定元组的顾客的平均收入替换income中的缺失值。  5.使用最可能的值填充缺失值  可以用回归、使用贝叶斯形式化的基于推理的工具或决策树归纳确定。例如，利用数据集中其他顾客的属性，可以构造一颗决策树来预测income的缺失值。 |
| **将df1和df2进行行合并，并选出c列大于等于9的数据。**  **import** pandas **as** pd  df1 = pd.DataFrame(  data=[[1, 2, 3], [4, 5, 6], [7, 8, 9]],  index=[1, 2, 3],  columns=[**'a'**, **'b'**, **'c'**]  )  df2 = pd.DataFrame(  data=[[11, 22, 33], [44, 55, 66], [77, 88, 99]],  index=[4, 5, 6],  columns=[**'a'**, **'b'**, **'c'**]  ) |
| **构造一个等差数列，数列长度为30，最小值1，最大值5**  import numpy as np  a = np.linspace(1, 5, 30)  print(a) |
| **== 和 is的区别?**  A = 10  B = 10  Print(a == b)  Print(a is b)  ==比较的是值  Is比较的是内存地址 |
| **Js中==和===的区别？**  "==" 只要求值相等; "===" 要求值和类型都相等 |
| **请写出以下操作的命令**  在当前用户根目录下创建文件夹f mkdir -p ~/f  在f下创建文件a和c touch ~/f/a ~/f/c  将a重命名为d并移动到上一级目录 mv ~/f/a ~/d  将b复制到上一级目录并写入hello world cp ~/f/b ~/b echo ‘hello world’ > ~/b  将f的权限设置为最高权限 chmod -R 777 ~/f |
| **xrange和range区别(针对python2说的)**  1.xrang返回的是生成器,range返回是列表  2.xrange性能比range快很多  3.尤其是返回很大的时候。尽量用xrange吧，除非你是要返回一个列表.  Python3 已经 把 range和xrange合并为range |
| **字典、列表查询时的时间复杂度是怎样的？**  列表是序列，可以理解为数据结构中的数组，字典可以理解为数据结构中的hashmap，python中list对象的存储结构采用的是线性表，因此其查询复杂度为O(n),而dict对象的存储结构采用的是散列表(hash表)，其在最优情况下查询复杂度为O(1)。dict的占用内存稍比list大，会在1.5倍左右。 |
| **什么是lambda函数？它有什么好处？另外python在函数编程方面提供了些什么函数和语法？**  l**ambda是Python中的匿名函数。它语法简单，简化代码，不会产生命名冲突，污染命名空间。Python提供了map，reduce，filter等函数方法，提供了装饰器，闭包等语法** |
| 如下两行代码：  **for** \_func **in** [**lambda**: x\*2 **for** x **in** range(2)]:  print(\_func())  结果  2  2 |
| **什么是闭包?(闭包的特点)**   1. 函数里面套函数 2. 外部函数的返回值是里面的内部函数 3. 内部函数必须使用外部函数的 变量   注意:以上三点都满足的才叫闭包 |
| **闭包的优缺点?**  1.闭包似优化了变量，原来需要类对象完成的工作，闭包也可以完成  2.由于闭包引用了外部函数的局部变量，则外部函数的局部变量没有及时释放，占用内存 |
| **什么是抓包?(用过哪些抓包工具)(为什么抓包)**   1. 抓包就是通过工具获取某些请求和请求中的数据 2. 抓包工具用过: 花瓶 Wireshark fiddler 3. 如果想获取手机端或者PC端的相关数据,可以通过抓包来实现   注意: 抓取Https的包,一般都是乱码, 可以通过fiddler尝试抓取 |
| **如何抓取手机端数据？**   * + - 1. 和pc端连接同一个网络       2. 配置手机端的代理，ip填写pc的ip，端口填写8888   打开花瓶，访问手机端网页/app即可 |
| **Python 用过哪些装饰器?**  @classmethod @staticmethod @property @csrf\_exempt @app-route |
| **写一个装饰器函数，打印函数的执行时间**  **import** time  **from** functools **import** wraps  **def** print\_use\_time(func):  @wraps(func)  **def** \_func(\*args, \*\*kwargs):  start = time.time()  result = func(\*args, \*\*kwargs)  end = time.time()  print(**f"函数耗时:{end-start}"**)  **return** result  **return** \_func  @print\_use\_time  **def** send\_request(url, method, \*\*kwargs):  print(**"发送请求开始"**)  time.sleep(2)  print(**"发送请求结束"**) |
| **装饰器的基本用法有哪些?**  http://www.cnblogs.com/alamZ/articles/7007945.html |
| **详细说说tuple、list、dict的用法，它们的特点？**  **tuple 元祖，固定长度不可变的顺序容器，访问效率高，适合存储一些常量数据，可以作为字典的键使用。**  **list 列表，是长度可变有序的数据存储容器。可以通过下标索引取到相应的数据。**  **dict 字典，长度可变的hash字典容器。存储的方式为键值对，可以通过相应的键获取相应的值，key支持多种类型。**  **备注:不可变类型可以充当字典的key,例如:字符串,元组等。** |
| **运行如下代码会出现什么情况？为什么？**  t = (2, 4, [6, 8])  t[-1].append(7)  print(t[-1])  t[-1] += [9]  print(t[-1])  [6, 8, 7]  TypeError: 'tuple' object does not support item assignment |
| **如下代码**  ***# demo1***  **ll = [1, 2, 3]**  **ll = ll + [1]**  ***# demo2***  **ll = [1, 2, 3]**  **ll += [1]**  ***# demo1和demo2有何不同***  Demo1中相当于对ll重新赋值，得到的ll是新的列表对象  Demo2中相当于修改自己的内容，得到的ll还是自己 |
| **说说python中装饰器、迭代器的用法；描述下dict 的 items() 方法与 iteritems() 方法的不同；**  **装饰器是指对函数执行过程，做一些扩展，甚至可以更改本身函数的执行**  **迭代器是指遵循迭代器协议的对象，这类对象在被for循环时，每次迭代生成下一个项，不用一开始就生成整个列表**  **在python3中 不存在iteritems，items方法返回可迭代对象**  **在python2中 items()返回[(key,value)]的列表对象，iteritems()返回迭代器对象，iteritems()循环时不可以增删dict的内容** |
| **你用过哪些迭代器(Iterator)?(什么是迭代器?)**  我大致用过以下两类可迭代类型:  一类是集合数据类型，如 list 、 tuple 、 dict 、 set 、 str 等；  一类是 generator ，包括生成器和带 yield 的generator function。  可以通过iter()函数将上述类型转换成迭代器  备注:可迭代的 iterable 迭代器 Iterator   1. 凡是能用for遍历的都是可迭代的 2. 凡是能用next()函数都是迭代器 3. 集合类型可以通过iter()函数生成迭代器 |
| **讲讲对unicode, gbk, utf-8等的理解，python2.x是如何处理编码问题？**  **unicode编码：为了解决各个国家的语言的存储，引进的unicode码，包括UCS2,UCS4,UTF-8,UTF-7等**  **gbk：和unicode不一样的编码方式 ，常用的为 gbk-2312**  **utf-8: UTF-8 使用 1-4 个字节来存储单个字符，应该是目前最流行的字符集。Linux 默认字符集就是UTF-8。既解决了大多数语言的编码方式，又避免了简单字符存储对空间的浪费**  **python 2.x 需要在文件上方声明文件的编码方式# -\*- coding: utf8 -\*-，可以通过str.decode/str.encode处理字符串的文本编码**  **保证 python 编辑器的编码正确，或使用 u’中文’保证被转换成 unicode 编码，推荐使用sys.setdefaultencoding('utf-8')来保证我们的编码**  **备注:python3默认是utf-8,几乎不存在编码问题** |
| **关于 python 程序的运行性能方面，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**  **6、多线程本身也能提升其性能,可以避免项目整体卡顿.但是python没有真正意义多线程,因为存在GIL(全局解释锁)**  **备注:python中用过哪些内建函数:如字典的,字符串,列表的** |
| **说出你对GIL的理解?**  GIL的全称是Global Interpreter Lock(全局解释器锁)，来源是python设计之初的考虑，为了数据安全所做的决定。某个线程想要执行，必须先拿到GIL，我们可以把GIL看作是“通行证”，并且在一个python进程中，GIL只有一个。拿不到通行证的线程，就不允许进入CPU执行。GIL只在cpython中才有，因为cpython调用的是c语言的原生线程，所以他不能直接操作cpu，只能利用GIL保证同一时间只能有一个线程拿到数据。而在pypy和jpython中是没有GIL的。 |
| **list 对象 alist [{'name':'a','age':20},{'name':'b','age':30},{'name':'c','age':25}]， 请按 alist 中元素的age 由大到小排序；**  **方案1：**  def sort\_by\_age(list1):  return sorted(alist,key=lambda x:x['age'],reverse=True)  **方案2：**（性能比匿名函数快）  from operator import itemgetter  dict1 = [  {'name': 'zhangsan', 'age': 20},  {'name': 'lisi', 'age': 30},  {'name': 'wangwu', 'age': 25},  {'name': 'zhangfei', 'age': 12},  {'name': 'zhaoliu', 'age': 42},  ]  sort\_by\_age = sorted(dict1, key=itemgetter('age'))  print(sort\_by\_age) |
| **两个 list 对象 alist ['a','b','c','d','e','f'], blist ['x','y','z','d','e','f']，请用简洁的方法合并这两个 list，并且 list 里面的元素不能重复；** 备注:列表拼接尽量不要用加号,因为会生成一个新列表,推荐使用extend alist = ['a','b','c','d','e','f']  blist = ['x','y','z','d','e','f']  **方案1**  def merge\_list(\*args):  s = set()  for i in args:  s = s.union(i)  return list(s)  **方案2**  alist.extend(blist)  print list(set(alist)) |
| **打乱一个排好序的 list 对象 alist；**  import random  random.shuffle(alist) |
| os.path和sys.path的区别？ os.path是module，包含了各种处理长文件名(路径名)的函数。  sys.path是由目录名构成的列表，Python 从中查找扩展模块( Python 源模块, 编译模块,或者二进制扩展). 启动 Python 时,这个列表从根据内建规则,PYTHONPATH 环境变量的内容, 以及注册表( Windows 系统)等进行初始化. |
| **将字符串："k:1|k1:2|k2:3|k3:4"，处理成 python 字典：{k:1, k1:2, ... }**  **方案1**  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  **方案2**  import json  str1 = "k:1|k1:2|k2:3|k3:4"  str1 = str1.replace('|', '","')  str1 = str1.replace(':', '":"')  str2 = '{"'+str1+'"}'  print str2  print json.loads(str2) |
| **使用json.dumps将dict转为json字符串时，如果包含中文需要注意什么？**  import json  dic = {'name': '张三'}  result = json.dumps(dic, ensure\_ascii=False)  print(result) |
| **url编码和解码函数是什么？**  from urllib import parse  str1 = "name=张三"  # 编码  str2 = parse.quote(str1)  print(str2)  # 解码  str3 = parse.unquote(str2)  print(str3) |
| **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 |
| **解释生成器(generator)与函数的不同，并实现和使用简单generator？**  **生成器和函数的主要区别在于函数 return avalue，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。** |
| **什么叫做生成器?你们用过哪些生成器**  这种一边循环一边计算的机制，称为生成器：generator  **1.列表生成器 将列表推导式[]改成()**  **2.通过yield可以做生成器**  **生成器的特点：**  **1.节约内存**  **2.迭代到下一次的调用时，所使用的参数都是第一次所保留下的，即是说，在整个所有函数调用的参数都是第一次所调用时保留的，而不是新创建的** |
| **dict 的 items() 方法与 iteritems() 方法的不同？**  **items方法将所有的字典以列表方式返回，其中项在返回时没有特殊的顺序；**  **iteritems方法有相似的作用，但是返回一个迭代器对象** |
| **Python中单下划线和双下划线的区别？**  xx: 公有变量  \_x: 单前置下划线,私有化属性或方法，from somemodule import \*禁止导入,类对象和子类可以访问  \_\_xx：双前置下划线,避免与子类中的属性命名冲突，无法在外部直接访问(名字重整所以访问不到)  \_\_xx\_\_:双前后下划线,用户名字空间的魔法对象或属性。例如:\_\_init\_\_ , \_\_ 不要自己发明这样的名字  xx\_:单后置下划线,用于避免与Python关键词的冲突 |
| **列表推到式和字典推到式怎么写？**  字典推导式：d = {key: value for (key, value) in iterable}  列表推到式：d = [a for a in iterable]  生成器推导集合式：d = (a for a in iterable)  集合推导式：d = {a for a in iterable} |
| **\*args and \*\*kwargs**  这两个是python中的可变参数。\*args表示任何多个无名参数，它是一个tuple；\*\*kwargs表示关键字参数，它是一个dict。如果同时使用\*args和\*\*kwargs时， \*args参数必须在\*\*kwargs前 |
| **Python中重载**  Python中没有真正的重载。因为重载有两个特点，一是可变参数类型，二是可变参数个数。其中python是弱类型语言，可以接收任意类型参数，而且python的函数不能重名，因此没有重载。但可以通过使用\*args和\*\*kwargs来模拟重载。 |
| **\_\_new\_\_和\_\_init\_\_和\_\_call\_\_()和\_\_del\_\_()和\_\_dict\_\_()的区别(最新)**  1.\_\_new\_\_是一个静态方法,而\_\_init\_\_是一个实例方法.  2.\_\_new\_\_方法会返回一个创建的实例,而\_\_init\_\_什么都不返回.  3.只有在\_\_new\_\_返回一个cls的实例时后面的\_\_init\_\_才能被调用.  4.当创建一个新实例时调用\_\_new\_\_,初始化一个实例时用\_\_init\_\_. |
| **简述old style class 和 new style class**  新式类继承自object或者其他新式类。新式类可以使用super函数，而且比老式类多了一些特殊方法，比如\_\_new\_\_方法。  \_\_new\_\_方法是一个静态方法，你不需要通过staticmethod来特意指定，python解释器会自动判断。当你执行C(\*args, \*\*kwargs)时，python的内部操作是先执行C.\_\_new\_\_(C,\*args,\*\*kwargs)，比如new方法返回的结果叫ret，python会判断ret是否是C的实例，如果是C的实例就执行C.\_\_init\_\_(ret,\*args,\*\*kwargs)，通过init方法来初始化实例。 |
| **Python中的作用域的搜索形式是什么？**  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） |
| **read,readline和readlines的区别是什么？**  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 |
| >>>seq = [1, 2, 3, 4]  >>>seq[:2]\_\_\_\_\_[1,2]\_\_\_\_\_\_\_\_\_\_\_  >>>seq[-2:]\_\_\_\_[3,4]\_\_\_\_\_\_\_\_\_\_\_  >>>seq[10:]\_\_\_\_[ ]\_\_\_\_\_\_\_\_\_\_\_\_  >>>seq[::-1]\_\_\_[4,3,2,1]\_\_\_\_\_\_\_\_  >>>seq[:]\_\_\_\_\_\_[1,2,3,4]\_\_\_\_\_\_\_\_  >>> id(seq[:]) == id(seq)  \_\_\_\_\_\_\_False\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| **如下代码**  **S = ‘abcdefg’**  **S1 = ‘abcedfg’**  **id(s)是否等于id(s1) 为什么？**  相等。在Python中，整数和短小的字符，Python都会缓存这些对象，以便重复使用。当我们创建多个相同字符串时，实际上是让所有这些引用指向同一个对象。为了检验两个引用指向同一个对象，我们可以用is关键字。is用于判断两个引用所指的对象是否相同。 |
| **优化以下程序**  **result = []for x in range(10):**  **result.append(x \*\* 2)**  **print(result)**  **优化后**  >>> result = [x\*\*2 for x in range(10)]  >>> print result  [0, 1, 4, 9, 16, 25, 36, 49, 64, 81] |
| **Pass的作用是什么？**  pass语句什么也不做，一般作为占位符或者创建占位程序，pass语句不会执行任何操作。 |
| **Python是如何进行类型转换的？**  Python提供了将变量或值从一种类型转换成另一种类型的内置函数。比如int函数能够将符合数学格式数字型字符串转换成整数。否则，返回错误信息。 |
| **你在编程期间遵守的编码规范是什么？**  1.PEP8: 是一种编码规范，是一系列关于如何让Python代码更加具有可读性的编码建议。  2.公司有自己的一套规范 |
| **PEP8的细节是？**  每一级缩进使用4个空格  空格是首选的缩进方式  所有行限制的最大字符数为79  顶层函数和类的定义，前后用两个空行隔开  类里的方法定义用一个空行隔开  。。。 |
| **版本控制怎么做？多人协作开发怎么做？项目版本迭代怎么做？都是一类问题**  使用GIT或者SVN。Svn基本已经被淘汰了，我们公司用的是GIT，仓库搭建在码云上，一般都是私有仓库，使用的可视化工具是小乌龟。SVN之前也用过一次，基本命令都差不多。 |
| **GIT和SVN的区别是什么？**  最核心的区别Git是分布式的，而Svn不是分布的。  Git的内容的完整性要优于SVN: GIT的内容存储使用的是SHA-1哈希算法。  Git和svn分支不同  <https://www.cnblogs.com/mtl-key/p/6902627.html> |
| **GIT常用命令有哪些？**  git add添加修改文件  git commit 提交修改到本地仓  git merge 合并分支  git pull拉取远程仓库版本  git push 推送本地仓到远程仓  git log 查看commit历史记录  **git仓库在哪？**  码云/github/公司内部服务器  **Git的可视化工具**  [Tortoise](http://www.baidu.com/link?url=2zGpFPjmM06AtWdFd80iO3gXIbCDktc1-FjDrQdNW6tCmwaVC8SsjWXpQdeY8X1gLkAK7K5L8no_YXGT2iIgEkX-jgH3CBlZDRMkpMB1gEG" \t "_blank)Git 小乌龟 |
| **GIT出现代码冲突怎么解决？**  打开冲突文件  删除特殊符号 <== === ==>  保留你想保留的代码，删除你想删除的代码  重新提交即可 |
| **PyCharm如何使用GIT对项目进行版本控制？**  <https://jingyan.baidu.com/article/f96699bbe6d685894e3c1b81.html> |
| **def f(x, l=[]):**  **for i in range(x):**  **l.append(i\*i)**  **print(l)**  f(2) # [0, 1]  f(3, [3, 2, 1]) # [3,2,1,0,1,4]  f(3) # [0,1,0,1,4] |
| **“猴子补丁”（monkey patching）指的是什么？这种做法好吗？**  “猴子补丁”就是指，在函数或对象已经定义之后，再去改变它们的行为。  举个例子：  http://mmbiz.qpic.cn/mmbiz_png/IP70Vic417DPribb6Gt7agibibMS6TWsBMybibWSB72kz1icV0QorgoJ6HRq3g9rkm8k3MAD0OQCBwqDl4JRbJt9blIA/640?wx_fmt=png&wxfrom=5&wx_lazy=1  大部分情况下，这是种很不好的做法 - 因为函数在代码库中的行为最好是都保持一致。打“猴子补丁”的原因可能是为了测试。mock包对实现这个目的很有帮助。 |
| **什么是序列化（pickling）和反序列化（unpickling）?**  Pickle模块允许我们将Python对象转换成一个string表示的信息，并且可以使用dump函数将其保存到一个文件中去，这样的过程称为序列化。而使用文件中保存的对象信息重构Python对象的过程称为反序列化。 |
| **Python是如何被解释执行的？**  Python是一种解释型语言，它可以直接从源代码运行程序。程序员编写的源代码首先转变成一种中间语言代码，然后再被转换成能够直接执行的机器语言代码。 |
| **能够找到程序中的bug进行静态分析的工具有哪些？**  PyChecker是一种能够发现Python源代码中的bug并对其中的代码风格和复杂度进行警告的静态分析工具。另外一种工具是Pylint，它能够验证模块是否满足编码标准。 |
| **下列代码的输出结果是**  **A0 = dict(zip(('a','b','c','d','e'),(1,2,3,4,5)))**  **A1 = range(10)**  **A2 = [i for i in A1 if i in A0]**  **A3 = [A0[s] for s in A0]**  **A4 = [i for i in A1 if i in A3]**  **A5 = {i:i\*i for i in A1}**  **A6 = [[i,i\*i] for i in A1]**  {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}  range(0, 10)  [ ]  [1, 2, 3, 4, 5]  [1, 2, 3, 4, 5]  {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}  [[0, 0], [1, 1], [2, 4], [3, 9], [4, 16], [5, 25], [6, 36], [7, 49], [8, 64], [9, 81]] |
| **Python提供哪些内置类型？**  可变类型：列表、集合、字典  不可变类型：字符串、元组、数字 |

1. 爬虫、网络、HTML

|  |
| --- |
| **您使用过的python网络爬虫所用到的网络请求包**  aiohttp、grequests、requests、urllib、urllib2、httplib2、urllib3 |
| **列举您使用过的python网络爬虫所用到的解析数据包**  BeautifulSoup、Xpath、lxml、pyquery、css、正则 |
| **请描述以下cookies，sessionStorage和localStorage的区别** |
| **介绍以下标准的CSS盒子模型** |
| **列举你熟知的块级元素与行内元素** |
| **Css选择符有哪些？哪些属性可以继承** |
| **Display有哪些值？说明他们的作用。** |
| **什么是window对象？什么是document对象？** |
| **列出比较熟悉的爬虫框架(其中pyspider要能说出用法)**  Scrapy、pyspider |
| **写出在网络爬虫爬取数据的过程中，遇到的防爬虫问题的解决方案**  通过headers反爬虫：解决策略，伪造headers  基于用户行为反爬虫：动态变化去爬取数据，模拟普通用户的行为  通过动态更改代理ip来反爬虫  基于动态页面的反爬虫：跟踪服务器发送的ajax请求，模拟ajax请求 |
| **常用的代理ip的网站有哪些?**  西刺代理 芝麻代理 |
| **代理池（cookie池）怎么维护？** |
| **post、get有什么区别？**  从参数位置,长度,安全性,使用场景四点来回答  1. 根据HTTP规范，GET一般用于获取/查询资源信息。而POST一般用于更新资源信息，例如表单提交。  2. get是在url中传递数据，数据放在请求头中。 post是在请求体中传递数据  3. get传送的数据量较小，只能在请求头上发送数据。post传送的数据量较大，一般被默认为不受限制。  4. Post比GET相对安全，但是两者都不安全。  注意：根据HTTP协议，没有规定GET和POST的以上区别，只是按照平常写法而总结的结论 |
| **Epoll，select的区别？边缘触发，水平触发区别？**  区别（epoll相对select优点）主要有三：  1.select的句柄数目受限，在linux/posix\_types.h头文件有这样的声明：#define \_\_FD\_SETSIZE    1024  表示select最多同时监听1024个fd。而epoll没有，它的限制是最大的打开文件句柄数目。  2.epoll的最大好处是不会随着FD的数目增长而降低效率，在selec中采用轮询处理，其中的数据结构类似一个数组的数据结构，而epoll 是维护一个队列，直接看队列是不是空就可以了。epoll只会对"活跃"的socket进行操作---这是因为在内核实现中epoll是根据每个fd上面 的callback函数实现的。那么，只有"活跃"的socket才会主动的去调用 callback函数（把这个句柄加入队列），其他idle状态句柄则不会，在这点上，epoll实现了一个"伪"AIO。但是如果绝大部分的I/O都是 “活跃的”，每个I/O端口使用率很高的话，epoll效率不一定比select高（可能是要维护队列复杂）。  3.使用mmap加速内核与用户空间的消息传递。无论是select,poll还是epoll都需要内核把FD消息通知给用户空间，如何避免不必要的内存拷贝就很重要，在这点上，epoll是通过内核于用户空间mmap同一块内存实现的。  在linux的IO多路复用中有水平触发,边缘触发两种模式,这两种模式的区别如下:  **水平触发**:如果文件描述符已经就绪可以非阻塞的执行IO操作了,此时会触发通知.允许在任意时刻重复检测IO的状态.select,poll就属于水平触发.  **边缘触发**:如果文件描述符自上次状态改变后有新的IO活动到来,此时会触发通知.在收到一个IO事件通知后要尽可能多的执行IO操作,因为如果在一次通知中没有执行完IO那么就需要等到下一次新的IO活动到来才能获取到就绪的描述符.信号驱动式IO就属于边缘触发. |
| **http报文组成结构？**  请求报文：请求行、请求头、空行、请求数据  响应报文：响应行、响应头、空行、响应数据 |
| **解释下Http请求头和常见响应状态码**  Accept:指浏览器或其他客户可以接爱的MIME文件格式。可以根据它判断并返回适当的文件格式。  Accept-Charset：指出浏览器可以接受的字符编码。英文浏览器的默认值是ISO-8859-1.  Accept-Language：指出浏览器可以接受的语言种类，如en或en-us，指英语。  Accept-Encoding：指出浏览器可以接受的编码方式。编码方式不同于文件格式，它是为了压缩文件并加速文件传递速度。浏览器在接收到Web响应之后先解码，然后再检查文件格式。  Cache-Control：设置关于请求被代理服务器存储的相关选项。一般用不到。  Connection：用来告诉服务器是否可以维持固定的HTTP连接。HTTP/1.1使用Keep-Alive为默认值，这样，当浏览器需要多个文件时(比如一个HTML文件和相关的图形文件)，不需要每次都建立连接。  Content-Type：用来表名request的内容类型。可以用HttpServletRequest的getContentType方法取得。  Cookie：浏览器用这个属性向服务器发送Cookie。Cookie是在浏览器中寄存的小型数据体，它可以记载和服务器相关的用户信息，也可以用来实现会话功能。  状态代码有三位数字组成，第一个数字定义了响应的类别，且有五种可能取值：  1xx：指示信息--表示请求已接收，继续处理  2xx：成功--表示请求已被成功接收、理解、接受  3xx：重定向--要完成请求必须进行更进一步的操作  4xx：客户端错误--请求有语法错误或请求无法实现  5xx：服务器端错误--服务器未能实现合法的请求  常见状态代码、状态描述、说明：  200 OK //客户端请求成功  400 Bad Request //客户端请求有语法错误，不能被服务器所理解  401 Unauthorized //请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用  403 Forbidden //服务器收到请求，但是拒绝提供服务  404 Not Found //请求资源不存在，eg：输入了错误的URL  500 Internal Server Error //服务器发生不可预期的错误  503 Server Unavailable //服务器当前不能处理客户端的请求，一段时间后可能恢复正常 |
| **简述JS中this的特点**  this含义是明确且具体的，即指向当前对象。一般在编译期绑定。而在javascript中，this是动态绑定的，它可以是全局对象、当前对象或者任意对象，这完全取决于函数的调用方式，并且在严格模式下也有不同，这就导致了this具备了多重含义，可以使得javascript更灵活的使用。 |
| **Ajax依赖于什么东西？如何发送get、post请求？**  依赖于jquery，可以直接引入网址，或者下载文件引入路径  $.ajax({url,data,type,datatype,success,error,}) |
| **前台向后台发数据用什么?后台返回数据格式是什么?**  Ajax  Json和xml |
| **http、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，这样的话即使黑客截取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 |
| **http协议特点有哪些?**  1.支持C/S模式,客户端/服务器模式 2.简单快速 3..灵活 4.无状态 5.无连接 |
| **域名和IP之间有什么关系，如何查看某个域名对应的所有IP？**  国际互联网(Internet)上有成千百万台主机（host），为了区分这些主机，人们给每台主机都分配了一个专门的“地址”作为标识，称为IP地址  由于IP地址全是些的数字，为了便于用户记忆，Internet上引进了域名服务系统DNS（Domain Name System）。  当您键入某个域名的时候，这个信息首先到达提供此域名解析的服务器上，再将此域名解析为相应网站的IP地址。完成这一任务的过程就称为域名解析。  1.ping 2.nslookup 3.使用站长工具等 |
| **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)。 |
| **列举几个常用的dom解析项目、插件**  xpath、lxml、xml、libxml2  前端操作dom使用js或jquery  Jquery是对js进行封装,用法上更简单,效率js更快.(国外用js更多,国内用jquery)  前台向后台传输数据:ajax,它依赖jquery |
| **请写出实现如下图所示的页面布局的HTML和CSS代码，使用div布局，不能使用table。**    **<!DOCTYPE html>**  **<html lang="en">**  **<head>**  **<meta charset="UTF-8">**  **<title>Title</title>**  **<style>**  **div** {  **border**: 2**px solid red**;  **text-align**: **center**;  }  **#out\_div** {  **display**: **flex**;  **flex-direction**: **column**;  }  **#in\_div** {  **display**: **flex**;  **flex-direction**: **row**;  }  **#center\_div1** {  **flex**: 1;  }  **#center\_div2** {  **flex**: 2;  }  **#center\_div3** {  **flex**: 1;  }  </style>  </head>  <body>  <div id="out\_div">  <div>1</div>  <div id="in\_div">  <div id="center\_div1">2</div>  <div id="center\_div2">3</div>  <div id="center\_div3">4</div>  </div>  </div>  </body>  </html> |
| **如何提高爬虫效率**  1.爬取方面，利用异步io。  2.处理方面，利用消息队列做生产者消费者模型  3.多线程  4.多进程  5.分布式  6.提升网速  7.部署高配置服务器中 |
| **有一个html文本字符串，让我取出<a href="提示我这个链接地址">sflkj</a>这个a标签里面的href的链接地址？**  **方案1**  import re  Pattern = re.compile(r'href="(.\*?)"')  print Pattern.findall('<a href="提示我这个链接地址">sflkj</a>')[0]  **方案2**  from lxml import etree  content = '<a href="提示我这个链接地址">sflkj</a>'  root = etree.fromstring(content)  print root.attrib['href']  **方案3**  from bs4 import BeautifulSoup  text = "<a href='提示我这个链接地址'>sflkj</a>"  the\_html = BeautifulSoup(text,features='lxml')  print(the\_html.find('a').attrs['href']) |
| **什么是Socket？什么是心跳包？** |
| **写出基于TCP的Socker服务端代码？** |
| **写出基于UDP的Socker服务端代码？** |
| **TCP和UDP的区别？**  **TCP是可靠的、面向连接的、尽力传输的协议，而UDP是不可靠的、面向非连接的、不尽力传输的协议。但是不可靠不代表它没有用，UDP有自己的应用场景，语音和视频几乎都在使用UDP协议，它的不可靠只是相对于 TCP来说的，但是它的好处就是效率，高效在某些场景要比可靠性重要。**  TCP（Transmission Control Protocol，传输控制协议）是面向连接的协议，也就是说，在收发数据前，必须和对方建立可靠的连接。一个TCP连接必须要经过三次“对话”才能建立起来，其中的过程非常复杂，只简单的描述下这三次对话的简单过程：主机A向主机B发出连接请求数据包：“我想给你发数据，可以吗？”，这是第一次对话；主机B向主机A发送同意连接和要求同步（同步就是两台主机一个在发送，一个在接收，协调工作）的数据包：“可以，你什么时候发？”，这是第二次对话；主机A再发出一个数据包确认主机B的要求同步：“我现在就发，你接着吧！”，这是第三次对话。三次“对话”的目的是使数据包的发送和接收同步，经过三次“对话”之后，主机A才向主机B正式发送数据。  UDP（User Data Protocol，用户数据报协议）  　　（1）UDP是一个非连接的协议，传输数据之前源端和终端不建立连接，当它想传送时就简单地去抓取来自应用程序的数据，并尽可能快地把它扔到网络上。在发送端，UDP传送数据的速度仅仅是受应用程序生成数据的速度、计算机的能力和传输带宽的限制；在接收端，UDP把每个消息段放在队列中，应用程序每次从队列中读一个消息段。  　　（2）由于传输数据不建立连接，因此也就不需要维护连接状态，包括收发状态等，因此一台服务机可同时向多个客户机传输相同的消息。  　　（3）UDP信息包的标题很短，只有8个字节，相对于TCP的20个字节信息包的额外开销很小。  　　（4）吞吐量不受拥挤控制算法的调节，只受应用软件生成数据的速率、传输带宽、源端和终端主机性能的限制。  　　（5）UDP使用尽最大努力交付，即不保证可靠交付，因此主机不需要维持复杂的链接状态表（这里面有许多参数）。  　　（6）UDP是面向报文的。发送方的UDP对应用程序交下来的报文，在添加首部后就向下交付给IP层。既不拆分，也不合并，而是保留这些报文的边界，因此，应用程序需要选择合适的报文大小。  我们经常使用“ping”命令来测试两台主机之间TCP/IP通信是否正常，其实“ping”命令的原理就是向对方主机发送UDP数据包，然后对方主机确认收到数据包，如果数据包是否到达的消息及时反馈回来，那么网络就是通的。  **TCP三次握手过程：**  　　1 主机A通过向主机B 发送一个含有同步序列号的标志位的数据段给主机B ,向主机B 请求建立连接,通过这个数据段，主机A告诉主机B 两件事:我想要和你通信;你可以用哪个序列号作为起始数据段来回应我.  　　2 主机B 收到主机A的请求后,用一个带有确认应答(ACK)和同步序列号(SYN)标志位的数据段响应主机A,也告诉主机A两件事:我已经收到你的请求了,你可以传输数据了;你要用哪佧序列号作为起始数据段来回应我  　　3 主机A收到这个数据段后,再发送一个确认应答,确认已收到主机B 的数据段:"我已收到回复,我现在要开始传输实际数据了这样3次握手就完成了,主机A和主机B 就可以传输数据了.  **三次握手的特点：**  　　没有应用层的数据  　　SYN这个标志位只有在TCP建产连接时才会被置1  　　握手完成后SYN标志位被置0  **为什么TCP建立连接要进行3次握手,而断开连接要进行4次：**  　　1 当主机A完成数据传输后,将控制位FIN置1,提出停止TCP连接的请求  　　2  主机B收到FIN后对其作出响应,确认这一方向上的TCP连接将关闭,将ACK置1  　　3 由B 端再提出反方向的关闭请求,将FIN置1  　　4 主机A对主机B的请求进行确认,将ACK置1,双方向的关闭结束.  　　由TCP的三次握手和四次断开可以看出,TCP使用面向连接的通信方式,大大提高了数据通信的可靠性,使发送数据端和接收端在数据正式传输前就有了交互,为数据正式传输打下了可靠的基础。 |
| **TCP三次握手的时候IP层和Mac层在做什么？** |
| **爬虫框架scrapy的工作流程？**  a、spider解析下载器下下来的response，返回item或是links  b、item或者link经过spidermiddleware的process\_spider\_out( )方法，交给engine  c、engine将item交给item pipeline ，将links交给调度器  d、在调度器中，先将requests对象利用scrapy内置的指纹函数生成一个指纹对象  e、如果requests对象中的don't filter参数设置为False，并且该requests对象的指纹不在信息指纹的队列中，那么就把该request对象放到优先级队列中  f、从优先级队列中获取request对象，交给engine  g、engine将request对象交给下载器下载，期间会通过downloadmiddleware的process\_request方法  h、下载器完成下载，获得response对象，将该对象交给engine，期间会经过downloadmiddleware的process\_response（ ）方法  i、engine将获得的response对象交给spider进行解析，期间会经过spidermiddleware的process\_spider\_input()的方法  j、从a开始循环 |
| **多线程和多进程有什么区别？**  **线程：**  　　线程是操作系统能够进行运算调度的最小单位（程序执行流的最小单元）。它被包含在进程之中，是进程中的实际运作单位。一条线程指的是进程中一个单一顺序的控制流，一个进程中可以并发多个线程，每条线程并行执行不同的任务。  　　一个标准的线程由线程ID，当前指令指针(PC)，寄存器集合和堆栈组成。另外，线程是进程中的一个实体，是被系统独立调度和分派的基本单位，线程自己不拥有系统资源，只拥有一点儿在运行中必不可少的资源，但它可与同属一个进程的其它线程共享进程所拥有的全部资源。一个线程可以创建和撤消另一个线程，同一进程中的多个线程之间可以并发执行。由于线程之间的相互制约，致使线程在运行中呈现出间断性。线程也有就绪、阻塞和运行三种基本状态。就绪状态是指线程具备运行的所有条件，逻辑上可以运行，在等待处理机;运行状态是指线程占有处理机正在运行;阻塞状态是指线程在等待一个事件(如某个信号量)，逻辑上不可执行。每一个程序都至少有一个线程，若程序只有一个线程，那就是程序本身。  　　线程是程序中**一个单一的顺序控制流程**。**进程内一个相对独立的、可调度的执行单元**，是系统独立调度和分派CPU的基本单位指运行中的程序的调度单位。在单个程序中同时运行多个线程完成不同的工作，称为多线程。  **进程：**  进程由至少一个线程组成。进程是计算机中的程序关于某数据集合上的一次运行活动，是系统进行资源分配和调度的基本单位，是操作系统结构的基础。在早期面向进程设计的计算机结构中，进程是程序的基本执行实体;在当代面向线程设计的计算机结构中，进程是线程的容器。程序是指令、数据及其组织形式的描述，进程是程序的实体。里面包含对各种资源的调用，内存的管理，网络接口的调用等。。。对各种资源管理的集合 就可以称为 进程  **区别：**  （1）线程共享内存空间；进程的内存是独立的  （2）同一个进程的线程之间可以直接交流；两个进程想通信，必须通过一个中间代理来实现  （3）创建新线程很简单； 创建新进程需要对其父进程进行一次克隆  （4）一个线程可以控制和操作同一进程里的其他线程；但是进程只能操作子进程  （5）改变主线程（如优先权），可能会影响其它线程；改变父进程，不影响子进程 |
| **写一个多线程程序，调用子程序输出pid后，随机sleep1到10秒后退出。** |
| **爬取的内容怎么去重？**  **利用数据库，根据信息的特定字段去重。**  **根据数据和url去去重;**  **根据scrapy自带的去重;**  **还可以用md5对内容加密后再比较；**  **使用布隆过滤去重；** |
| **用正则匹配手机号码(笔试题考了)**  “^((13[0-9])|(14[5|7])|(15([0-3]|[5-9]))|(18[0,5-9]))\\d{8}$” |
| **有一个文本，以#或@对其进行分割。例如“123#456@abccccc@d”,分割结果为[“123”, “456”, “abccccc”, “d”]**  **import** re  content = **"123#456@abccccc@d"**  pattern = re.compile(**r"@|#"**)  result = pattern.split(content)  print(result) |
| **写一个正则，匹配文本(a+b)\*(a1-b1)/2中所有变量(a, b, a1, b1)**  **import** re  content = **"(a+b)\*(a1-b1)/2"**  pattern = re.compile(**r"[a-zA-Z]+\d\*"**)  pattern = re.compile(**r"[a-z]\w\*"**)  result = pattern.findall(content)  print(result) |
| **re模块中match和search方法的不同？**  **match() 函数只检查 RE 是否在字符串开始处匹配，而search() 则是扫描整个字符串。** |
| **session和cookie的联系与区别；session为什么说是安全的；**   |  | Cookie | Session | | --- | --- | --- | | 储存位置 | 客户端 | 服务器端 | | 目的 | 跟踪会话，也可以保存用户偏好设置或者保存用户名密码等 | 跟踪会话 | | 安全性 | 不安全 | 安全 |   Session依赖于Cookie |
| **Cookie有哪些属性和属性的作用(刚考过)**  name字段为一个cookie的名称。  value字段为一个cookie的值。  domain字段为可以访问此cookie的域名  path字段为可以访问此cookie的页面路径。  Size字段 此cookie大小。  http字段  cookie的httponly属性。若此属性为true，则只有在http请求头中会带有此cookie的信息，而不能通过document.cookie来访问此cookie。  secure 字段 设置是否只能通过https来传递此条cookie |
| **有没有做过单点登录?怎么实现?**  简称SSO,如果在网站的某一个子域名下登陆过,那么再访问其它域名就不需要登录了.  用户登陆过以后将数据存储到cookie中,但是要修改cookie的域名字段,设置为 .域名  <https://cnodejs.org/topic/55f6e69904556da7553d20dd> |
| **为h1标签定义样式表:字体红色,14像素大小,内边距4像素**  <h1 style=’color:red;font-size:14px;padding:4px’>helloworld</h1>  <style>  H1 {  Color:green;  Font-size:14px;  Padding:4px;  }  </style>  创建一个\*\*\*.css文件,里面写成  H1 {  Color:blue;  Font-size:14px;  Padding:4px;  } |
| **robots协议是什么？**  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。 |
| **什么是协程？**  协程可以认为是一个微线程,就是只有一个线程,对一个线程进行分解,在一个线程中规定某些任务执行的顺序，当有些任务不需要使用cpu操作时(IO),可以使用协程 |
| **urllib和urllib2的区别**  1.urllib提供urlencode方法用来GET查询字符串的产生，而urllib2没有。这是为何urllib常和urllib2一起使用的原因。  2.urllib2可以接受一个Request类的实例来设置URL请求的headers，urllib仅可以接受URL。这意味着，你不可以伪装你的User Agent字符串等。 |
| **HTTP1.0和HTTP1.1的区别？**  1.请求头Host字段,一个服务器多个网站 2.长链接  3.文件断点续传 4.身份认证,状态管理,Cache缓存 |
| **如何实现大型文件的断点下载/断点续传**  1.发送请求获取文件总大小  2.发送下载请求的时候在请求头部添加一个range字段，用来表示下载文件的范围  3.下载过程当中不断把数据拼接到文件后  4.如果已经暂停，如何继续下载？  5.通过os获取文件已经下载的大小，继续通过range往后下载 |
| **正则的贪婪匹配和非贪婪匹配**  贪婪匹配：尽可能多的匹配数据  非贪婪匹配：尽可能少的匹配数据 |
| **解释Python中basestring,str和unicode的关系**  basestring是str和unicode的超类（父类），是抽象类（The basestring type cannot be instantiated），不能被调用和实例化，但可以用来判断一个对象是否为str或者unicode的实例，isinstance(obj, basestring)等价于isinstance(obj, (str, unicode))；  python2.3版本以后引入该函数，兼容python2.3以后python2各版本。但是python3舍弃了该函数，所以该函数不能在python3中使用。 |
| **From \_\_future\_\_ import division这条语句有什么作用?**  导入python未来支持的语言特征division(精确除法)，当我们没有在程序中导入该特征时，"/"操作符执行的是截断除法(Truncating Division),当我们导入精确除法之后，"/"执行的是精确除法。 |
| **解释下Http协议**  HTTP是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。  HTTP协议的主要特点可概括如下：  1.支持客户/服务器模式。  2.简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于HTTP协议简单，使得HTTP服务器的程序规模小，因而通信速度很快。  3.灵活：HTTP允许传输任意类型的数据对象。正在传输的类型由Content-Type加以标记。  4.无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。  5.无状态：HTTP协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。 |
| **爬取数据，进行市场调研和商业分析** 知乎：爬取优质答案，为你筛选出各话题下最优质的内容。  淘宝、京东：抓取商品、评论及销量数据，对各种商品及用户的消费场景进行分析。  安居客、链家：抓取房产买卖及租售信息，分析房价变化趋势、做不同区域的房价分析。  拉勾网、智联：爬取各类职位信息，分析各行业人才需求情况及薪资水平。  雪球网：抓取雪球高回报用户的行为，对股票进行分析和预测。 |
| **作为机器学习、数据挖掘的原始数据** 比如你要做一个推荐系统，那么你可以去爬取更多维度的数据，做出更好的模型。  比如你要做图像识别，你可以先去爬取大量的图片作为训练集进行训练。 |
| **爬取优质的资源：图片、文本、视频** 爬取知乎钓鱼贴\图片网站，获得福利图片。  爬取微信公众号文章，分析新媒体内容运营策略。  这些事情，原本我们也是可以手动完成的，但如果是单纯地复制粘贴，非常耗费时间，比如你想获取100万行的数据，大约需忘寝废食重复工作两年。而爬虫可以在一天之内帮你完成，而且完全不需要任何干预。  对于小白来说，爬虫可能是一件非常复杂、技术门槛很高的事情。比如有的人认为学爬虫必须精通 Python，然后哼哧哼哧系统学习 Python 的每个知识点，很久之后发现仍然爬不了数据；有的人则认为先要掌握网页的知识，遂开始 HTML\CSS，结果入了前端的坑，瘁……  但掌握正确的方法，在短时间内做到能够爬取主流网站的数据，其实非常容易实现。但建议你从一开始就要有一个具体的目标，你要爬取哪个网站的哪些数据，达到什么量级。  在目标的驱动下，你的学习才会更加精准和高效。那些所有你认为必须的前置知识，都是可以在完成目标的过程中学到的。这里给你一条平滑的、零基础快速入门的学习路径。 |

1. 数据库

|  |
| --- |
| **目前主流的大型数据库名称有哪些？至少三个**  MySQL，Oracle，SQLServer |
| **简述触发器的作用？**  什么是触发器：  触发器是一种特殊的存储过程，它在试图更改触发器所保护的数据时自动执行。  触发器有什么作用：  1. 安全性。可以基于数据库的值使用户具有操作数据库的某种权利。  2. 审计。可以跟踪用户对数据库的操作。  3. 实现复杂的非标准的数据库相关完整性规则。触发器可以对数据库中相关的表进行连环更新。触发器能够拒绝或回退那些破坏相关完整性的变化，取消试图进行数据更新的事务。当插入一个与其主健不匹配的外部键时，这种触发器会起作用。  4. 同步实时地复制表中的数据。  5. 自动计算数据值，如果数据的值达到了一定的要求，则进行特定的处理。  触发器弊端：  1、如果需要变动整个数据集而数据集数据量又较大时，触发器效果会非常低  2、对于批量操作并不适合使用触发器 使用触发器实现的业务逻辑在出现问题时很难进行定位，特别是设计到多个触发器的情况 协同开发时，写业务层代码如果不清楚数据库 触发器的细节，容易搞不清到底触发了那些触发器 大量使用触发器会导致代码结构容易被打乱，阅读源码困难 |
| **是否创建过历史表，简述历史表的作用** |
| **关系型数据库的ACID指的是什么?**  1、A (Atomicity) 原子性  原子性很容易理解，也就是说事务里的所有操作要么全部做完，要么都不做，事务成功的条件是事务里的所有操作都成功，只要有一个操作失败，整个事务就失败，需要回滚。  比如银行转账，从A账户转100元至B账户，分为两个步骤：1）从A账户取100元；2）存入100元至B账户。这两步要么一起完成，要么一起不完成，如果只完成第一步，第二步失败，钱会莫名其妙少了100元。  2、C (Consistency) 一致性  一致性也比较容易理解，也就是说数据库要一直处于一致的状态，事务的运行不会改变数据库原本的一致性约束。  例如现有完整性约束a+b=10，如果一个事务改变了a，那么必须得改变b，使得事务结束后依然满足a+b=10，否则事务失败。  3、I (Isolation) 独立性  所谓的独立性是指并发的事务之间不会互相影响，如果一个事务要访问的数据正在被另外一个事务修改，只要另外一个事务未提交，它所访问的数据就不受未提交事务的影响。  比如现有有个交易是从A账户转100元至B账户，在这个交易还未完成的情况下，如果此时B查询自己的账户，是看不到新增加的100元的。  4、D (Durability) 持久性  持久性是指一旦事务提交后，它所做的修改将会永久的保存在数据库上，即使出现宕机也不会丢失。 |
| 有表List，并有字段A、B、C，类型都是整数。表中有如下几条记录：   |  |  |  | | --- | --- | --- | | A | B | C | | 2 | 7 | 9 | | 5 | 6 | 4 | | 3 | 11 | 9 |   现在对该表一次完成以下操作：  查询出B和C列的值，要求按B列升序排列  Select B,C from List order by B AES;  写出一条新的记录，值为{7,9,8}  Insert into List values (7,9,8);  查询C列，要求消除重复的值，按降序排列  Select distinct(C) from List order by C DESC; |
| **数据库视图的作用**  1. 数据库视图隐藏了数据的复杂性。  2. 数据库视图有利于控制用户对表中某些列的访问。  3. 数据库视图使用户查询变得简单。  备注:是个虚拟表,本身不存在,但是可以使用.  Mysql:1.基本sql(结构化查询语言)语句2.索引,3.事务,4.视图5.性能优化 |
| **列出几种常见的关系型数据库和非关系型数据库？**  SQL: Oracle、Mysql、SQLServer、DB2  NoSQL: Redis MongoDB Cassandra |
| **Python如何操作Mongodb、mysql?**  使用pymongo/pymysql |
| **关系型数据库和非关系型数据库的区别?**   |  |  |  |  | | --- | --- | --- | --- | | 数据库类型 | 特性 | 优点 | 缺点 | | 关系型数据库  SQLite、Oracle、mysql | 1、关系型数据库，是指采用了关系模型来组织数据的数据库；  2、关系型数据库的最大特点就是事务的一致性；  3、简单来说，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。 | 1、容易理解：二维表结构是非常贴近逻辑世界一个概念，关系模型相对网状、层次等其他模型来说更容易理解；  2、使用方便：通用的SQL语言使得操作关系型数据库非常方便；  3、易于维护：丰富的完整性(实体完整性、参照完整性和用户定义的完整性)大大减低了数据冗余和数据不一致的概率；  4、支持SQL，可用于复杂的查询。 | 1、为了维护一致性所付出的巨大代价就是其读写性能比较差；  2、固定的表结构；  3、高并发读写需求；  4、海量数据的高效率读写； | | 非关系型数据库  MongoDb、redis、HBase | 1、使用键值对存储数据；  2、分布式；  3、一般不支持ACID特性；  4、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。 | 1、无需经过sql层的解析，读写性能很高；  2、基于键值对，数据没有耦合性，容易扩展；  3、存储数据的格式：nosql的存储格式是key,value形式、文档形式、图片形式等等，文档形式、图片形式等等，而关系型数据库则只支持基础类型。 | 1、不提供sql支持，学习和使用成本较高；  2、无事务处理，附加功能bi和报表等支持也不好； | |
| **设教学数据库Education有三个关系表：**  **学生关系S（SNO,SNAME,AGE,SEX,SDEPT）;学习关系SC（SNO, CNO, GRADE）；课程关系C（CNO, CNAME, CDEPT）**  **备注：SNO学号，SNAME姓名，AGE年龄，SEX性别，SDEPT专业，CNO课程号，CNAME课程名称，GRADE年级，CDEPT课程所属专业**  **使用sql语句解释下列问题：**   1. **检索计算机专业的全体学生的学号、姓名和性别，按照学号降序排序** 2. **检索学习课程号为C2的学生学号与姓名** 3. **检索选修课程为DS的学生学号与姓名** 4. **查询学号大于100的学生学号与姓名** 5. **检索学生姓名中包含华的学生学号与姓名** 6. **检索不学C2课的学生姓名和年龄** 7. **检索学习全部课程的学生姓名** 8. **统计一共有多少名学生** 9. **查询最大学号的同学姓名与年龄** 10. **插入课程关系：课程号-100，课程名称-物理，所属专业-物理** 11. **更新10记录专业为理科** 12. **删除11记录更新后的记录** 13. **删除课程关系表C** |
| **MySQL锁有几种?**  1） 表级锁：开销小，加锁快；不会出现死锁；锁定粒度大，发生锁冲突的概率最高，并发度最低。  2） 行级锁：开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低，并发度也最高。  3） 页面锁：开销和加锁时间界于表锁和行锁之间；会出现死锁；锁定粒度界于表锁和行锁之间，并发度一般。  三种锁各有各的特点，若仅从锁的角度来说，表级锁更适合于以查询为主，只有少量按索引条件更新数据的应用，如WEB应用；行级锁更适合于有大量按索引条件并发更新少量不同数据，同时又有并发查询的应用，如一些在线事务处理（OLTP）系统。 |
| **一个表A有一列time代表他的插入时间，查询最新插入的10条记录**  select \* from 表名 order by time desc limit 10; |
| **一个表A有一列time代表他的插入时间，查询这张表每天插入的记录数量**  SELECT COUNT (\*) FROM 表名 GROUP BY TO\_CHAR (time,'YYYY-MM-DD')； |
| **异步编程中的锁有几种？** |
| **死锁是怎么产生的?**  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  3.检测死锁(资源分配图)  4.解除死锁  4.1剥夺资源  4.2撤销进程 |
| 如何分区、分表；  <https://blog.csdn.net/zhanglong_longlong/article/details/52670141> |
| MySQL的char varchar text的区别；  数据的检索效率是：char > varchar > text  空间占用方面，就要具体情况具体分析了。  char：存储定长数据很方便，CHAR字段上的索引效率级高，必须在括号里定义长度，可以有默认值，比如定义char(10)，那么不论你存储的数据是否达到了10个字节，都要占去10个字节的空间（自动用空格填充），且在检索的时候后面的空格会隐藏掉，所以检索出来的数据需要记得用什么trim之类的函数去过滤空格。  varchar：存储变长数据，但存储效率没有CHAR高，必须在括号里定义长度，可以有默认值。保存数据的时候，不进行空格自动填充，而且如果数据存在空格时，当值保存和检索时尾部的空格仍会保留。另外，varchar类型的实际长度是它的值的实际长度+1，这一个字节用于保存实际使用了多大的长度。  text：存储可变长度的非Unicode数据，最大长度为2^31-1个字符。text列不能有默认值，存储或检索过程中，不存在大小写转换，后面如果指定长度，不会报错误，但是这个长度是不起作用的，意思就是你插入数据的时候，超过你指定的长度还是可以正常插入。 |
| 如何防止sql注入攻击？  1）execute()函数本身有接受sql语句参数位的，可以通过python自身的函数处理sql注入问题  2）python模块MySQLdb自带针对mysql的字符转义函数escape\_string，可以对字符串转义 |
| 了解join么，有几种，有何区别？  left join：左表(A)的记录将会全部表示出来,而右表(B)只会显示符合搜索条件的记录  right join：是以右表(B)为基础的,A表不足的地方用NULL填充.  Inner join：两个表都符合才显示 |
| 数据库的索引的工作原理是什么？你用过哪些索引？  索引是对记录按照多个字段进行排序的一种方式。对表中的某个字段建立索引会创建另一种数据结构，其中保存着字段的值，每个值又指向与它相关的记录。这种索引的数据结构是经过排序的，因而可以对其执行二分查找。  非唯一索引、唯一索引、全局索引、局部索引、全文索引......  数据库索引的优缺点：  第一，通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。  　　第二，可以大大加快数据的检索速度，这也是创建索引的最主要的原因。  　　第三，可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。  　　第四，在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。  第五，通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。  第一，创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。  第二，索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间，如果要建立聚簇索引，那么需要的空间就会更大。  第三，当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度。 |
| 数据库索引的使用场景？  1. 数据量较大，且查询频繁的字段，可以考虑创建索引提高查询效率,创建索引要额外占用磁盘空间，建立的索引太多可能导致磁盘空间不足。因此，在建立索引时，一定要慎重选择正确的字段。  2. 数据量小，或增删改频繁，则不推荐使用索引 |
| redis有几种常用存储类型  字符串、集合、有序集合、列表、哈希 |
| Redis的安装流程?(windows和ubuntu的安装) |
| Redis的应用场景?  数据缓存,主从,数据备份,分布式 |
| Redis作为数据缓存服务器是怎么体现的?  因为MySQL的访问速度不是特别好,如果访问量特别巨大,数据库可能会崩溃,所以使用缓存服务器来减轻数据库压力  当需要获取数据的时候,先从缓存中获取,如果存在则直接返回,如果不存在,再从数据库中查找,找到以后缓存一份然后再返回  一般数据更改不频繁的会放入缓存中,例如商品信息,验证码,浏览量 |
| 如何保证MySQL的数据更改以后缓存也会相应更新?  在更新数据的时候手动更新(删除)缓存 |
| Redis和memcache的区别?  1. Redis中，并不是所有的数据都一直存储在内存中的，这是和Memcache相比一个最大的区别。  2. Redis在很多方面具备数据库的特征，或者说就是一个数据库系统，而Memcache只是简单的K/V缓存。  3. 他们的扩展都需要做集群；实现方式：master-slave、Hash。  4. 在100k以上的数据中，Memcache性能要高于Redis。  5. 如果要说内存使用效率，使用简单的key-value存储的话，Memcached的内存利用率更高，而如果Redis采用hash结构来做key-value存储，由于其组合式的压缩，其内存6. 利用率会高于Memcache。当然，这和你的应用场景和数据特性有关。  7. 如果你对数据持久化和数据同步有所要求，那么推荐你选择Redis，因为这两个特性Memcache都不具备。即使你只是希望在升级或者重启系统后缓存数据不会丢失，选择Redis也是明智的。  8. Redis和Memcache在写入性能上面差别不大，读取性能上面尤其是批量读取性能上面Memcache更强  Memcache      Memcache可以利用多核优势，单实例吞吐量极高，可以达到几十万QPS,适用于最大程度扛量      只支持简单的key/value数据结构，不像Redis可以支持丰富的数据类型。      无法进行持久化，数据不能备份，只能用于缓存使用，且重启后数据全部丢失   Redis      支持多种数据结构，如string,list,dict,set,zset,hyperloglog      单线程请求，所有命令串行执行，并发情况下不需要考虑数据一致性问题。      支持持久化操作，可以进行aof及rdb数据持久化到磁盘，从而进行数据备份或数据恢复等操作，较好的防止数据丢失的手段。      支持通过Replication进行数据复制，通过master-slave机制，可以实时进行数据的同步复制，支持多级复制和增量复制.      支持pub/sub消息订阅机制，可以用来进行消息订阅与通知。      支持简单的事务需求，但业界使用场景很少，并不成熟 |
| redis如何持久化?两种方式有什么区别?   Redis DataBase(简称RDB)  执行机制：快照，直接将databases中的key-value的二进制形式存储在了rdb文件中  优点：性能较高（因为是快照，且执行频率比aof低，而且rdb文件中直接存储的是key-values的二进制形式，对于恢复数据也快）  使用单独子进程来进行持久化，主进程不会进行任何IO操作，保证了redis的高性能  缺点：在save配置条件之间若发生宕机，此间的数据会丢失  RDB是间隔一段时间进行持久化，如果持久化之间redis发生故障，会发生数据丢失。所以这种方式更适合数据要求不严谨的时候   Append-only file (简称AOF)  执行机制：将对数据的每一条修改命令追加到aof文件  优点：数据不容易丢失  可以保持更高的数据完整性，如果设置追加file的时间是1s，如果redis发生故障，最多会丢失1s的数据；且如果日志写入不完整支持redis-check-aof来进行日志修复；AOF文件没被rewrite之前（文件过大时会对命令进行合并重写），可以删除其中的某些命令（比如误操作的flushall）  缺点：性能较低（每一条修改操作都要追加到aof文件，执行频率较RDB要高，而且aof文件中存储的是命令，对于恢复数据来讲需要逐行执行命令，所以恢复慢）  AOF文件比RDB文件大，且恢复速度慢。 |
| BTree索引和hash索引的区别？  1. Hash 索引仅仅能满足"=","IN"和"<=>"查询，不能使用范围查询。  2. Hash 索引无法被用来避免数据的排序操作。  3. Hash 索引不能利用部分索引键查询。  4. Hash 索引在任何时候都不能避免表扫描。  5. Hash 索引遇到大量Hash值相等的情况后性能并不一定就会比B-Tree索引高。 |
| 常用的数据库存储引擎有哪些？什么区别？  MyISAM 适合于一些需要大量查询的应用，但其对于有大量写操作并不是很好。甚至你只是需要update一个字段，整个表都会被锁起来，而别的进程，就算是读进程都无法操作直到读操作完成。另外，MyISAM 对于 SELECT COUNT(\*) 这类的计算是超快无比的。  InnoDB 的趋势会是一个非常复杂的存储引擎，对于一些小的应用，它会比 MyISAM 还慢。他是它支持“行锁”，于是在写操作比较多的时候，会更优秀。并且，他还支持更多的高级应用，比如：事务。 |

1. Django

<http://www.cnblogs.com/LiCheng-/p/6920900.html> 基本覆盖所有django知识点

<https://www.jianshu.com/p/38c4dd6d8e28?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation> django优秀资源

|  |
| --- |
| 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。** |
| 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。** |
| 简要说明Django中Context与RequestContext的区别Django的模板渲染中，Context可以用来传递数据，一个Context是一系列变量和值的集合，它和Python的字典有点相似。from django.template import Template,Contextt = Template('My name is {{ name }}.')c = Context({'name':'Jacky'})t.render(c)在settings.py中有一个与RequestContext密切相关的配置项为TEMPLATE\_CONTEXT\_PROCESSORS。这些Processors都会被RequestContext顺序调用，往当前Context中放入一些预定义变量。当使用render\_to\_response方法时，RequestContext应作为其第三个参数传入，如：from django.shortcuts import render\_to\_response, RequestContext　　#...　　return render\_to\_response('template.html', particular\_data\_dictionary, context\_instance=RequestContext(request)) |
| http协议是一种无状态的协议，如何在django编写的web应用中记住客户的状态，比如如何记住某个用户的购物车中都有哪些商品。写出几个你认为django中与此相关的模块，并简要解释其实现原理。 |
| Template机制是django中的一个重要组成部分，如果让您来设计和实现django中现有的template机制和功能（包括tag和filter等），您的设计和实现思路是怎样的？请简要描述一下。 |
| Django的请求的生命周期 |
| 如何调试django 1. 使用logging记录异常错误信息  2. 使用断点调试工具  3. 使用django-debug-toolbar插件  4. 使用try except异常捕获 |
| Django的template的注释是什么样子？ {# 单行注释  #}  {% comment %} 多行注释 {% endcomment %} |
| Django怎么弄并发？1. 采用nginx做负载2. 去掉自增主键原因很简单,因为自增主键的存在写库存在抢锁, 可以利用全局id生成器提前生成id直接写入数据库3. 换成异步任务去写库如果数据只是存在mysql中做备份，建议使用异步的方式写入库，先把数据写到缓存下发给用户，之后在利用后台异步任务一点点的写入，例如聊天系统可以这样干换成更高效的框架或者语言4. 可以试试tornado, 如果tornado依然无法满足，可以尝试使用golango，毕竟golang是以高并发著称, 而且是编译语言，而且基于它的web框架也很容易上手，性能很可观，例如Iris======================================django毫无疑问，用原生django的server做处理的表现是最烂的，在10000次请求的情况下brokenpipe的几率极高，只有1400次请求被处理，成功率只有14%，我也懒得继续测下去了。django + nginx这次搭上了nginx做反向代理，也使的脆弱的django服务器的情况有所缓解，但成功率仍然不高（10000次请求中有3684个请求被处理）。uwsgi + nginxuwsgi是性能极高的一个由C编写的服务器，它使用uwsgi协议，这次让它配合nginx处理django的request，参数为4进程+2线程，性能立即直线上升，处理请求的成功率也基本在90%左右，不过我在测试时遇到了一个坑，就是uwsgi在处理请求的时候发送了队列溢出的问题，因为当前测试设置的并发数为每秒1000次并发，而uwsgi的处理队列容量默认为100，导致处理请求的时间加长，而这个问题则可以通过修改somaxcon的大小解决，总的来说，使用uwsgi+nginx是一个理想的选择。gunicorn + nginxgunicorn跟uwsgi类似，也是一个高性能的http服务器，它由ruby的unicorn项目移植，是由python编写的，它的配置简单，而且可以灵活地搭配其他网络库，部署十分方便，在测试数据中可以看到，用这种配置运行django能在短时间内就能处理大量的并发请求，成功率在90%左右。gunicorn + nginx + gevent前面说的几种环境，看似不错，但我们需要追求完美！由于gunicorn是同步（sync）单线程模型的，有的时候它不免会发生一些阻塞问题，这时候我们为gunicorn加上-k gevent参数来用gevent做处理接口，这就比较靠谱地处理了阻塞问题，从数据中可以看到，gunicorn + nginx + gevent的模式不仅拥有100%的处理成功率，而且时间也在很短之内完成，是5组测试数据当中的性能最好的。 |
| Django的form主要具有哪些功能？1. 自动生成HTML表单元素2. 检查表单数据的合法性3. 如果验证错误，重新显示表单（数据不会重置）4. 数据类型转换（字符类型的数据转换成相应的Python类型） |
| Tornado的ioloop知道是什么吗？ |
| 多继承的顺序python中使用多继承，会涉及到查找顺序（MRO）、重复调用（钻石继承，也叫菱形继承问题）等MROMRO即method resolution order，用于判断子类调用的属性来自于哪个父类。在Python2.3之前，MRO是基于深度优先算法的，自2.3开始使用C3算法，定义类时需要继承object，这样的类称为新式类，否则为旧式类从图中可以看出，旧式类查找属性时是深度优先搜索，新式类则是广度优先搜索 http://segmentfault.com/img/bVq1gN |
| Django的权限认证怎么做 |
| 解释下Django使用redis缓存服务器 为了能在Django中使用redis，还需要安装redis for Django的插件。然后在Django的settings中配置了。现在连接和配置都已经完成了，接下来是一个简单的例子：   |  | | --- | | from django.conf import settings  from django.core.cache import cache  #read cache user id  def read\_from\_cache(self, user\_name):      key = 'user\_id\_of\_'+user\_name      value = cache.get(key)      if value == None:          data = None      else:          data = json.loads(value)      return data  #write cache user id  def write\_to\_cache(self, user\_name):      key = 'user\_id\_of\_'+user\_name      cache.set(key,json.dumps(user\_name),settings.NEVER\_REDIS\_TIMEOUT) | |
| Django中用过哪些中间件？  SessionMiddleware  CsrfViewMiddleware  AuthenticationMiddleware |
| 如何自定义中间件？  **from** django.utils.deprecation **import** MiddlewareMixin  **class** MyCustomMiddleware(MiddlewareMixin):  **def** process\_request(self, request):  print(**"这里写具体处理请求的逻辑"**)  **def** process\_response(self, request, response):  **return** response  然后在settings中配置中间件，一般放在SessionMiddleware之后 |
| Django中自定义过哪些中间件？  用于过滤UserAgent/ip/Cookie  限定ip访问次数的中间件  进行前后台安全验证的中间件 |
| Django中用过哪些装饰器？   1. @csrf\_exempt 关闭当前函数的csrf安全检测   登录验证 @login\_required  指定请求方法 @require\_http\_methods(["GET", "POST"])   1. 自定义:自定义登录验证的装饰器 |
| 访问django的一个网页，需要经过哪些文件？  urls.py -> middleware -> views.py -> templates --> \*\*\*.html |
| 如何使用ajax向django发送POST请求？  第一种  <script>  $(".eq").on("click",function () {  $.ajax({  url:"/eq/",  type:"POST",  data:{  csrfmiddlewaretoken: {{ csrf\_token }}, //必须写在模板中，才会被渲染  a: $(".a").val(),  b: $(".b").val()  },  success:function (data) {  $(".c").html(data);  }  })  })  </script>  第二种  <script src="/static/jquery.cookie.js"></script> //必须先引入它  <script>  $("#btn").on("click",function () {  $.ajax({  url:"/lianxi/",  type:"POST",  headers:{"X-CSRFToken":$.cookie('csrftoken')},  data:$("#f1").serialize()  })  })  </script> |
| 如何部署django项目到Linux？ |
| uWSGI和Nginx的作用  uWSGI是一个Web服务器，它实现了WSGI协议、uwsgi、http等协议。  Nginx中HttpUwsgiModule的作用是与uWSGI服务器进行交换。WSGI是一种Web服务器网关接口。它是一个Web服务器（如nginx，uWSGI等服务器）与web应用（如用Flask框架写的程序）通信的一种规范。 |
| 常见的后台攻击方式有哪些？  Sql注入  Xss跨站脚本攻击 控制数据格式，过滤标签  Csrf表单伪造 使用验证码， 给表单添加csrf\_token字段  上传木马文件 只能上传图片或视频文件 |
| 网站用户密码保存(用过哪些加密方式？)  1.明文保存  2.明文hash后保存,如md5  3.MD5+Salt方式,这个salt可以随机  4.知乎使用了Bcrypy(好像)加密 |
| 什么是RestfulAPI？Django如何实现Rest接口开发？ |
| 什么是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 |

1. Linux

|  |
| --- |
| 您最熟悉的Unix环境是\_\_\_.Unix下查询环境变量的命令是\_,查询脚本定时任务的命令是\_\_\_  AIX,env,crontab |
| 将当前目录下所有文件打包成a.tar（写命令） |
| 列出当前目录下所有文件与目录们包括隐藏文件（写命令） |
| 创建a目录，并将b文件移动到a目录下（写命令） |
| 删除一个目录b，包括里面所有文件（写命令） |
| 查看当前linux系统版本号（写命令） |
| 查找当前目录，包括子目录下文件名为a的文件（写命令） |
| 给当前目录添加其他组用户读权限，包括子目录下的所有文件（写命令） |
| 请列出vi编辑器常用的命令：向下，向上，进入输入模式，退到命令行，保存退出，删除当前行，跳到最后一行 |
| Linux通过终端命令查看磁盘剩余空间  df -h |
| 强行杀死名字为server.py的所有进程 |
| 查找/home下面文件内容包含jack关键词且后缀为.txt的所有文件 |
| 让一个名字为server.py的python脚本在后台运行 |
| 找出端口号为8888对应的程序名称和路径 |
| Unix进程间通信方式有哪些? |
| apache和nginx的区别 nginx 相对 apache 的优点：  1.轻量级，同样起web 服务，比apache 占用更少的内存及资源  2.抗并发，nginx 处理请求是异步非阻塞的，支持更多的并发连接，而apache 则是阻塞型的，在高并发下nginx 能保持低资源低消耗高性能  3.配置简洁  4.高度模块化的设计，编写模块相对简单  5.社区活跃  apache 相对nginx 的优点：  1.rewrite ，比nginx 的rewrite 强大  2.模块超多，基本想到的都可以找到  3.少bug ，nginx 的bug 相对较多  4.超稳定 |

1. 数据结构

|  |
| --- |
| 请举出三种常用的排序算法  冒泡、选择、快速 |
| 使用JS实现快排 |
| 快排和归并排序实现原理(笔试面试都考过) |
| 列出2种稳定的排序算法 |
| 请列举常用的数据结构有哪些？  栈 队列 链表 树 顺序表 |
| 简单实现一个stack(栈结构,特点:先入后出 FILO:First In Last Out) 说明：栈的一些典型功能:push(入栈),pop(出栈),top(栈顶),empty(栈为空),size(栈的大小)必须在代码中体现出来，写错没关系 class Stack:  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) |
| 简述linux部署主从的流程 |
| 简述linux部署scrapy项目的流程 |
| 简述scrapy分布式的实现流程 |
| scrapy如何和celery结合? |
| Celery的用法? |
| 结巴分词的用法?缺点? |
| 请写出linux中重启nginx服务器的命令 |

1. 其他

|  |
| --- |
| 对你最有影响的或是您认为最有价值的软件方面的几本书是  Python 核心编程,利用Python进行数据分析,Python CookBook,Python数据分析与挖掘实战 |
| 内存泄露是什么？如何避免？  指由于疏忽或错误造成程序未能释放已经不再使用的内存的情况。内存泄漏并非指内存在物理上的消失，而是应用程序分配某段内存后，由于设计错误，失去了对该段内存的控制，因而造成了内存的浪费。  导致程序运行速度减慢甚至系统崩溃等严重后果。有 \_\_del\_\_() 函数的对象间的循环引用是导致内存泄漏的主凶。不使用一个对象时使用:del object 来删除一个对象的引用计数就可以有效防止内存泄漏问题。  通过 Python 扩展模块 gc 来查看不能回收的对象的详细信息。  可以通过 sys.getrefcount(obj) 来获取对象的引用计数，并根据返回值是否为 0 来判断是否内存泄漏。  Python 默认内存处理机制为GC(垃圾回收)  例如：  class People(object):  def \_\_init\_\_(self):  self.dog = None  print 'people init'  def \_\_del\_\_(self):  print 'people del'  class Dog(object):  def \_\_init\_\_(self):  self.p = None  print 'Dog init'  def \_\_del\_\_(self):  print 'Dog del'  for i in range(0,10000):  p1 = People()  d1 = Dog()  p1.dog = d1  d1.p = p1 |
| Python的内存管理机制  1.引用计数：通过引用计数来保持对内存中的变量跟踪，Python内部记录中所有在使用对象各有多少个引用。  Python中有个内部跟踪变量叫做引用计数器，每个变量有多少个引用，简称引用计数。当对象被创建时就创建了一个引用计数。  当某个对象的引用计数为0时，对象就不在需要，就列入了垃圾回收队列。  引用计数增加：1.对象被创建：x=4;2.另外的别人被创建：y=x;3.被作为参数传递给函数：foo(x);4.作为容器对象的一个元素：a=[1,x,'33'];  引用计数减少时：1.一个本地引用离开了它的作用域。比如上面的foo(x)函数结束时，x指向的对象引用减1;  2.对象的别名被显式的销毁：del x ；或者del y;  3.对象的一个别名被赋值给其他对象：x=789  4.对象从一个窗口对象中移除：myList.remove(x)  5.窗口对象本身被销毁：del myList，或者窗口对象本身离开了作用域。  2.垃圾回收  1.引用计数： 每个对象中都有ob-refcnt来做引用计数。当一个对象...,ob-refcnt就会增加，当引用的对象删除，那么ob-refcnt就会减少当ob-refcnt为零，就会释放该对象的内存空间  2.标记清除： 解决循环引用的问题。先按需分配，等到没有空闲内存的时候，从寄存器和程序栈上的引用出发，遍历所有对象和引用把所有能访问的打标记，最后将没有标记的对象释放掉  3.分代技术： 提高效率，提高垃圾回收的效率，按照存活时间，分成不同的集合。将内存块按照其存活时间划分为不同的集合。每个集合就称为一个“代”，垃圾回收的频率随代的存活时间增大而减小。。Python默认定义分代对象集合，引用数越大，对象的存活时间越长  例如：  当某些内存块M经过了3次垃圾收集的清洗之后还存活时，我们就将内存块M划到一个集合A中去，而新分配的内存都划分到集合B中去。当垃圾收集开始工作时，大多数情况都只对集合B进行垃圾收集，而对集合A进行垃圾回收要隔相当一段长时间才进行，这就使得垃圾收集机制需要处理的内存少了，效率自然就提高了。在这个过程中，集合B中的某些内存块由于存活时间长而会被转移到集合A中，当然，集合A中实际上也存在一些垃圾，这些垃圾的回收会因为这种分代的机制而被延迟。  3.内存池机制  在Python中，大多数申请的都是小块的内存，会执行大量的malloc和free操作。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，即Pymalloc机制。  它将不用的内存放到内存池而不是返回给操作系统。  1. 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。  2. 对于Python对象，如整数，浮点数和List，都有其独立的私有内存池，对象间不共享他们的内存池。也就是说如果你分配又释放了大量的整数，用于缓存这些整数的内存就不能再分配给浮点数。  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 时由该层直接分配内存。  4.第3层是最上层， 也就是我们队python对象的直接操作。  Python在运行期间会大量执行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：  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） |
| 对内存中栈和堆的了解？  内存结构:1.堆区,栈区,全局区,文字常量区,程序代码区.  　　栈：  什么是栈，它是你的电脑内存的一个特别区域，它用来存储被每一个function（包括mian（）方法）创建的临时变量。栈是FILO，就是先进后出原则的结构体，它密切的被CPU管理和充分利用。每次function声明一个新的变量，它就会被“推”到栈中。然后每次一个function退出时，所有关于这个函数中定义的变量都会被释放（换句话说就是删除）。一旦栈中的变量释放，这块区域就会变成可用的，提供给其他栈中的变量。  用栈存储变量的好处是，内存是被你管理的。你不用手动的创建内存，不用当你不在需要它的时候手动释放内存。另外，由于CPU组织栈内存很高效。读出和写入栈变量是很快的。        理解栈的关键是理解概念，当一个function退出时，所有它的变量都会从栈中弹出,以后都会永远消失。因此栈中的变量本质是局部的。这和我们原来理解为变量作用域或者本地或者全局变量是相关的。在C中，一个公共的bug 是从你程序中的一个function外尝试访问一个在栈中的这个function的变量（在该function已经退出后）。       关于栈的另一个特点我们应该记住，就是存储在栈中的变量的大小有限制。而堆上创建变量不用考虑。       总结栈：        a、栈的伸长和伸缩就是函数压入或者推出局部变量。        b、我们不用自己去管理内存，变量创建和释放都是自动的。        c、栈中的变量只有在函数创建运行时存在。  　　堆：  堆也是我们的计算机内存中的一个区域，但是他不是自动管理的。而且也不是被CPU密切的管理着。它是一片更加自由的内存区域（很大）。要想在堆上创建内存，我们必须使用malloc() 或者calloc（），他们都是C语言编译的。一旦你在堆上分配内存，当你不在需要的时候你必须用free()去销毁。如果你不销毁或者销毁失败，你的程序就会有内存泄露。换句话说就是堆内存会一直在，其他进程无法使用。       不像栈，堆没有变量大小的限制（除了你电脑的物理限制条件外）。堆内存读出和写入都比较慢，因为它必须使用指针图访问堆内存。  栈和堆的优缺点：(背下来)       　 栈：            a、快速访问。            b、没有必要明确的创建分类变量，因为它是自动管理的。            c、空间被CPU高效地管理着，内存不会变成碎片。            d、只有局部变量            e、受限于栈大小(取决于操作系统)            f、变量不能调整大小。        堆：            a、变量可以被全局访问            b、没有内存大小限制            c、（相对）访问比较慢            d、没有高效地使用空间，随着块内存的创建和销毁，内存可能会变成碎片。            e、你必须管理内存（变量的创建和销毁你必须要负责）            f、变量大小可以用realloc( )调整 |
| 递归中如果没有终止条件会怎样？  概念：递归算法是一种直接或者间接的调用自身算法的过程。在计算机编写程序中，递归算法对解决一大类问题是十分有效的。  特点：  　　①递归就是在过程或者函数里调用自身。  　　②在使用递归策略时，必须有一个明确的递归条件，称为递归出口。  　　③递归算法解题通常显得很简洁，但递归算法解题的效率较低。所以一般不倡导使用递归算法设计程序。  　　④在递归调用的过程当中系统的每一层的返回点、局部变量等开辟了栈来存储。递归函数次数过多容易造成栈溢出等。 所以一般不倡导用递归算法设计程序  　　递归如果没有终止条件会导致递归调用成为死循环而不能正常结束，并且会造成栈溢出 |
| 如何解决递归的栈溢出问题？  使用尾递归进行优化，但Python中没有尾递归，好在有大牛写过尾递归的装饰你，具体代码不记得了。 |
| 递归计算n的阶乘 |
| 设计一个算法，计算出n阶乘中尾部零的个数 |
| 回文数是指正读倒读都一样的整数，例如121,1221等。请设计 ，找出100000以内的回文素数（既是回文数又是素数（质数）） |
| 递归计算1到100的阶乘的和  **def** sum\_factorial(x):  **if** x == 1:  **return** 1  **def** f(x):  **if** x == 1:  **return** 1  **return** f(x - 1) \* x  **return** sum\_factorial(x - 1) + f(x)  print(sum\_factorial(3)) |
| 写一个简单递归函数显示C盘下所有文件及文件夹名称  **import** os  **def** gci(filepath):  *#遍历filepath下所有文件，包括子目录*  files = os.listdir(filepath)  **for** fi **in** files:  fi\_d = os.path.join(filepath, fi)  **if** os.path.isdir(fi\_d):  gci(fi\_d)  **else**:  print(os.path.join(filepath, fi\_d)) |
| **计算n的阶乘，不要使用递归**  def factorial(n):  x = 1  for i in range(1, n+1):  x = x \* i  return x |
| Python 是如何进行内存管理的？python 的程序会内存泄露吗？说说有没有什么方面防止或检测内存泄露？ 备注:iOS使用的内存管理机制是MRC(手动引用计数管理)和ARC(自动引用计数管理). Python GC主要使用 引用计数 来跟踪和回收垃圾。在引用计数的基础上，通过“标记-清除”解决容器对象可能产生的循环引用问题。通过以空间换时间的方法提高垃圾回收效率.  1、引用计数： 每个对象中都有ob-refcnt来做引用计数。当一个对象被创建,ob-refcnt就会增加，当引用的对象删除，那么ob-refcnt就会减少,当ob-refcnt为零，就会释放该对象的内存空间  2、标记清除： 解决循环引用的问题。先按需分配，等到没有空闲内存的时候，从寄存器和程序栈上的引用出发，遍历所有对象和引用把所有能访问的打标记，最后将没有标记的对象释放掉  3、分代技术： 提高效率，提高垃圾回收的效率，按照存活时间，分成不同的集合。将内存块按照其存活时间划分为不同的集合。每个集合就称为一个“代”，垃圾回收的频率随代的存活时间增大而减小。。Python默认定义分代对象集合，引用计数越大，对象的存活时间越长.  Python也会内存泄露，Python本身的垃圾回收机制无法回收重写了\_\_del\_\_的循环引用的对象  程序员管理好每个python对象的引用，尽量在不需要使用对象的时候，断开所有引用  尽量少通过循环引用组织数据，可以改用weakref叫做弱引用或者用id之类的句柄访问对象  通过gc模块的接口可以检查出每次垃圾回收有哪些对象不能自动处理，再逐个逐个处理 |
| 深度遍历和广度遍历算法如何实现？  **def** deep\_search(root):  print(root.value)  **if** root.left\_child:  deep\_search(root.left\_child)  **if** root.right\_child:  deep\_search(root.right\_child)  **def** wide\_search(root):  queue = [root]  **while** queue:  node = queue.pop(0)  print(node.value)  **if** node.left\_child:  queue.append(node.left\_child)  **if** node.right\_child:  queue.append(node.right\_child) |
| Python如何实现单例模式？  使用\_\_new\_\_实现  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 使用共享属性实现 Class Borg(object):      \_state = {}      def \_\_new\_\_(cls, \*args, \*\*kw):          ob = super(Borg, cls).\_\_new\_\_(cls, \*args, \*\*kw)          ob.\_\_dict\_\_ = cls.\_state          return ob    class MyClass2(Borg):      a = 1 使用装饰器实现 def singleton(cls, \*args, \*\*kw):      instances = {}      def getinstanc  e():          if cls not in instances:              instances[cls] = cls(\*args, \*\*kw)          return instances[cls]      return getinstance    @singleton  class MyClass:    ... |
| 微信公众号二次开发的配置流程？ |
| 订阅号和公众号的区别是什么？ |

爬虫上机操作

|  |
| --- |
| 杭州市江干区下沙天城东路创意大厦B608  <http://online.watertest.com.cn/>  # -\*- coding:utf-8 -\*-  from selenium import webdriver  browser = webdriver.PhantomJS()  browser.get("http://online.watertest.com.cn/")  print browser.page\_source  map\_list = browser.find\_elements\_by\_xpath("//map")  try:      for mp in map\_list:          map\_id = mp.get\_attribute("id")          map\_name = mp.get\_attribute("name")          area\_list = mp.find\_elements\_by\_xpath("area")          for area in area\_list:              # print area.get\_attribute("alt")              onmousedown = area.get\_attribute("onmousedown")              browser.execute\_script(onmousedown.encode("utf-8"))              # print browser.find\_elements\_by\_xpath("//table[@class='table1 style\_2']/tr/td//text()")              print '监测站名称:', browser.find\_elements\_by\_xpath("//strong")[0].text              # 测量时间 print browser.find\_elements\_by\_xpath("//strong")[1].text              # 项目 print browser.find\_elements\_by\_xpath("//strong")[2].text              # 测量值 print browser.find\_elements\_by\_xpath("//strong")[3].text              # 水质类别print browser.find\_elements\_by\_xpath("//strong")[4].text              # Ⅲ类标准 print browser.find\_elements\_by\_xpath("//strong")[5].text              print '时间', browser.find\_elements\_by\_xpath("//strong")[6].text              # pH print browser.find\_elements\_by\_xpath("//strong")[7].text              print 'pH测量值', browser.find\_elements\_by\_xpath("//strong")[8].text              # '水质类别1' print  browser.find\_elements\_by\_xpath("//strong")[9].text              print browser.find\_elements\_by\_xpath("//strong")[10].text              print browser.find\_elements\_by\_xpath("//strong")[11].text              print browser.find\_elements\_by\_xpath("//strong")[12].text              print browser.find\_elements\_by\_xpath("//strong")[13].text              print browser.find\_elements\_by\_xpath("//strong")[14].text              print browser.find\_elements\_by\_xpath("//strong")[15].text              print browser.find\_elements\_by\_xpath("//strong")[16].text              print "\*"\*20              break  except Exception as e:      print e      browser.quit() |
| 写程序获取一个字符串中的数字字符,并按照数组形式输出,例如  输入:dgfhfgh25433bhku2894fgdhdy675gfh  输出:[25433,2894,675]  import re  content = "dgfhgh2543bhku2894fgdhdy675gfh"  pattern = re.compile("\d{1,}")  content\_list = pattern.findall(content)  print content\_list |
| 单向链表复制：一个链表的节点数据结构如下所示，补全deep\_clone方法。  输入：链表头结点  输出：新链表的头节点。要求新链表是旧链表的完全复制，不能复用旧链表的节点对象  提示：链表的尾节点满足条件 next\_node is None  Class Node(object):  Def \_\_init\_\_(self, int\_val, next\_node):  Self.int\_val = int\_val  Self.next\_node = next\_node  Def deep\_clone(head\_node):  Pass  def deep\_clone(head\_node):  node\_list = []  while head\_node != None:  node = Node(head\_node.int\_val, None)  node\_list.append(node)  head\_node = head\_node.next\_node  for idx, node in enumerate(node\_list):  if idx+1 == len(node\_list):  break  node.next\_node = node\_list[idx+1]  return node\_list[0] |
| Scrapy的中间件/pipeline的底层代码 |
| Django的中间件底层代码 |
| 你知道哪些算法?(证明你是计算机专业,各种算法都要说) |
| 爬虫测试网址:全国企业统一信用网 <http://gsxt.hnaic.gov.cn/notice/> |
| 爬虫怎么更新数据?是全部更新还是更新一部分?(重要)  定时爬取全部网页，如果有网页更新就更新一部分。  可以通过对网页源码md5加密后存储起来，更新前判断一下md5字段是否一致 |
| https://www.urlteam.org/2016/05/merchants-guide-to-the-galaxy%E7%AC%94%E8%AF%95%E9%A2%98%E8%A7%A3%E6%9E%90/ |
| 用python写一个词频统计小工具,统计英语文档中所有单词的词频,输出每个单词在文档中出现次数,按出现次数从多到少排序,每列一个单词,出现次数相同的单词按字典排序,并输出文档中的单词总数.  输入为文本文件,输出结果也请写入文本文件.  请在程序中加上必要的使用说明.  输出样例:  With:12  Me:11  My:11  import sys  import codecs  # 统计有哪些单词  word = []  # 统计单词和单词的个数  counter = {}  # 读取a.txt文件,获得句柄handle  with codecs.open('a.txt') as handle:      # 通过handle读取文档所有行,返回列表,获得每一行的内容line      for line in handle.readlines():          # 去掉行左右的空格          line = line.strip()          # 如果行的内容长度为0表示为空行,忽略不计          if len(line) == 0:              continue          # 每一行按照空格分割获得单词列表          for w in line.split():              # 如果单词没有统计过              if not w in word:                  # 把单词加到列表中                  word.append(w)              # 如果单词没有统计过个数              if not w in counter:                  # 默认个数为0                  counter[w] = 0              else:                  # 如果统计过个数,就在原有基础上加1                  counter[w] += 1  # 对字典按照value排序,降序排列  counter\_list = sorted(counter.items(), key=lambda x: x[1], reverse=True)  # 打开空白文件result.txt准备写入数据  with codecs.open("result.txt", "w", encoding="utf-8") as handle:      # 遍历列表的单词和个数      for word, count in counter\_list:          # 按照a:b的形式拼接字符串          info = "{}:{}\n".format(word, str(count))          # 写入文件          handle.write(info) |
| 已知如下序列：[(1, {“date”: “2015-3-1”}, “jack”), (2, {“date”: “2015-3-2”}, “tom”)…],写程序根据date先后排序  a\_list = [(1, {"date": "2015-3-1"}, "jack"), (2, {"date": "2015-3-2"}, "tom"), (3, {"date": "2015-2-2"}, "tom")]  print sorted(a\_list, key=lambda a:a[1]['date']) |
| import sys  import codecs  reload(sys)  sys.setdefaultencoding("utf-8")  counter = {}  with codecs.open('a.txt') as handle:  for line in handle.readlines():  line = line.strip()  if len(line) == 0:  continue  line = line.split(" ")  key = line[0]  value = line[1]  if not key in counter:  counter[key] = {"value\_count": 0, "value\_list": [], "values\_count": {value:1}}  else:  if not value in counter[key]['value\_list']:  counter[key]['value\_list'].append(value)  counter[key]['value\_count'] += 1  if not value in counter[key]['values\_count'].keys():  counter[key]['values\_count'][value] = 1  else:  counter[key]['values\_count'][value] += 1  print counter  for key, value in counter.items():  key += ":"  for item in value['values\_count'].items():  key += ",".join(map(str, item))  key += ";"  print key[:-1] |
| 已知一段二进制数据为TCP协议头部信息,写程序解析包含的各字段内容.TCP Header格式如下 [http://www.rfc-editor.org/rfc/rfc793.txt (3](http://www.rfc-editor.org/rfc/rfc793.txt%20(3).1 Header Format) |
| 请编写一个Python程序,可以获取任意一个京东商品页面的价格和库存信息,测试链接可以使用<https://item.id.com/5089225.html>, 作为命令行参数传入,该程序返回对应的价格和库存(是否在售). |
| 用一行Python代码对给定的任意整数数列,按照如下规则排序:  1.非负数在前,负数在后;  2.非负数部分按照从小到大排序;  3.负数部分按从大到小排序;  例如:foo = [-5, 8, 0, 4, 9, -4, -20, -2, 8, 2, -4]  排序后:foo = [0, 2, 4, 8, 8, 9, -2, -4, -4, -5, -20] |
| 简述with语句的作用和用法? |
| 判断101-200之间有多少个质数，并输出所有素数？ |
| 给出一个m行n列的网格地图，每个位置为0或1，0表示海水，1表示陆地，找出四周都环水的陆地坐标与个数 |
| 说说decotator的用法和场景? |
| Iterable与Generators的区别有哪些?return与yield有哪些不同? |
|  |
| 广州创品知识产产权服务有限公司爬虫工程师测试题.字段要求:商品名称,产品链接,价格,销售,店铺名称,店铺链接,地址,电话等.测试数据地址如下:  <https://s.1688.com/selloffer/-74D0F4.html?spm=b26110380.sw1688.0.0.JEPvj9&featurePair=973%3A125200612&cps=1&earseDirect=false&n=y&province=%B9%E3%B6%AB&filt=y&city=%D6%D0%C9%BD&uniqfield=pic_tag_id>  from selenium import webdriver  import time,codecs,hashlib,requests,sys  from lxml import etree  from openpyxl import Workbook  reload(sys)  sys.setdefaultencoding("utf-8")  wb = Workbook()  ws = wb.active  ws.append([u'商品名称', u'商品地址', u'商品价格', u'商品销量', u'商店名称', u'商店网址', u'商店地址', u'商店手机']) # 设置表头  start\_urls = 'https://s.1688.com/selloffer/-74D0F4.html?spm=b26110380.sw1688.0.0.JEPvj9&featurePair=973%3A125200612&cps=1&earseDirect=false&n=y&province=%B9%E3%B6%AB&filt=y&city=%D6%D0%C9%BD&uniqfield=pic\_tag\_id'  try:  browser = webdriver.Chrome()  browser.get(start\_urls)  time.sleep(1)  browser.find\_element\_by\_class\_name("s-overlay-close").click()  while True:  js ="document.getElementsByClassName('fui-next')[0].scrollIntoViewIfNeeded();"  browser.execute\_script(js)  time.sleep(1)  browser.execute\_script(js)  time.sleep(1)  browser.execute\_script(js)  next\_page = browser.find\_element\_by\_link\_text(u"下一页")  page\_source = browser.page\_source  file\_name = hashlib.md5(browser.current\_url).hexdigest()  with codecs.open(file\_name+".html", "w", encoding="utf-8") as handle:  handle.write(browser.page\_source)  contents = etree.HTML(browser.page\_source)  div\_list = contents.xpath("//ul[@id='sm-offer-list']/li/div[@class='imgofferresult-mainBlock']")  for div in div\_list:  goods\_name\_a = div.xpath("div[contains(@class, 's-widget-offershopwindowtitle')]/a")[0]  goods\_name = goods\_name\_a.xpath("@title")[0]  goods\_url = goods\_name\_a.xpath("@href")[0]  goods\_price = div.xpath("div[contains(@class, 's-widget-offershopwindowprice')]/span[contains(@class, 'sm-offer-priceNum')]")  if goods\_price:  goods\_price = goods\_price[0].get("title", "")  else:  goods\_price = ""  goods\_sale = div.xpath("div[contains(@class, 's-widget-offershopwindowprice')]/span[contains(@class, 'sm-offer-trade')]")  if goods\_sale:  goods\_sale = goods\_sale[0].get("title", "")  else:  goods\_sale = u""  goods\_shop\_a = div.xpath("div[contains(@class, 's-widget-offershopwindowcompanyinfo')]/a[contains(@class, 'sm-offer-companyName')]")  goods\_shop\_name = goods\_shop\_a[0].xpath("@title")  if goods\_shop\_name:  goods\_shop\_name = goods\_shop\_name[0]  else:  goods\_shop\_name = u""  goods\_shop\_url = goods\_shop\_a[0].xpath("@href")  if goods\_shop\_url:  goods\_shop\_url = goods\_shop\_url[0]  else:  goods\_shop\_url = u""  shop = requests.get(goods\_shop\_url).text  shop = etree.HTML(shop)  shop = shop.xpath("//div[@class='top-nav-bar-box']/ul/li[@data-page-name='contactinfo']/a/@href")  if shop:  shop\_url = shop[0]  shop = requests.get(shop\_url).text  shop = etree.HTML(shop)  goods\_shop\_phone = shop.xpath("//div[@class='contcat-desc']/dl[1]/dd/text()")  if goods\_shop\_phone:  goods\_shop\_phone = goods\_shop\_phone[0]  else:  goods\_shop\_phone = u""  goods\_shop\_address = shop.xpath("//div[@class='contcat-desc']/dl/dd[@class='address']/text()")  if goods\_shop\_address:  goods\_shop\_address = goods\_shop\_address[0]  else:  goods\_shop\_address = u""  else:  goods\_shop\_phone = ""  goods\_shop\_address = ""  print goods\_name, goods\_url, goods\_price, goods\_sale, goods\_shop\_name, goods\_shop\_url, goods\_shop\_address, goods\_shop\_phone  line = [goods\_name,  goods\_url,  goods\_price,  goods\_sale,  goods\_shop\_name,  goods\_shop\_url,  goods\_shop\_address,  goods\_shop\_phone]  line = map(str, line)  line = map(str.strip, line)  ws.append(line)  wb.save('goods.xlsx')  next\_page.click()  except Exception as e:  print e  browser.quit()  time.sleep(3)  browser.quit() |
| 带我飞科技:关于Python爬虫技术互动题  抓去源:香港快运  网站:http://www.hkexpress.com/en-hk  APP:HKEXPRESS  需要获取的数据:单程机票,具体看图红色框框.注意:考虑如何尽量避免被封  动态网站的爬取方式  考虑翻墙爬取 使用vpn |
| Python+selenium+phantomjs的缺陷是  爬取速度比较慢，就算使用多线程/多进程优化，也容易出现问题  Phantomjs已经停止更新，chrome无头模式可以替代phantomjs  定位网页元素的前提是，元素必须存在，需要通过代码滚动或者其他操作，否则会直接报错  使用scrapy-splash可以爬取动态网页，需要熟练使用docker命令‘ |
| N个结点的无向图，最多几条边？  2 – 1 3 – 3 4 – 6 5 – 10 6 – 15 7 – 21 n -- (n-1)\*n/2 |
| Linux文件夹权限是551的形式为-r-xr-x—x |
| SQL语句的alter的作用是： |
| 由于IP质量的问题，导致爬取成功率较低，如何提高成功率？  购买付费代理ip并搭建代理ip池  采用分布式爬虫  可以考虑爬取其他平台的数据，例如手机端 |
| 简单描述，用户在浏览器中输入百度首页回车之后的网络通信流程  浏览器在此时充当客户端，点击跳转以后  通过DNS将网址解析成对应的IP地址  根据网址等信息浏览器自动封装一个请求发给百度服务器  百度服务器收到请求，以响应的形式返回给浏览器  浏览器收到数据解析并展示 |
| Ubuntu在线安装包的命令是  Pip:python  Wget:下载  brew :mac  apt-get:ubuntu  yum:centos |
| 输入一段英文，单词间以空格隔开，统计不同单词的出现次数  words = [ 'look', 'into', 'my', 'eyes', 'look', 'into', 'my', 'eyes', 'the', 'eyes', 'the', 'eyes', 'the', 'eyes', 'not', 'around', 'the', 'eyes', "don't", 'look', 'around', 'the', 'eyes', 'look', 'into', 'my', 'eyes', "you're", 'under' ]  from collections import Counter  word\_counts = Counter(words)  # 出现频率最高的 3 个单词  top\_three = word\_counts.most\_common(3)  print(top\_three) |
| <http://www.qyfgj.cn/newys/user_kfs.aspx>  爬取下方的下一页 |
| *微博数据循环下拉再解析*  *# -\*- coding:utf-8 -\*-*  import time  from lxml import etree  from selenium import webdriver  url = 'https://weibo.com/ttarticle/p/show?id=2310474178360870779439'  browser = webdriver.PhantomJS()  try:  browser.get(url)  time.sleep(1)  content = browser.page\_source  detail\_content = etree.HTML(content)  print(content)  title = detail\_content.xpath("//div[@class='title']/text()")  print(title)  except:  print("报错了")  finally:  browser.quit() |
| *假如一万个新闻让你用同一套爬虫系统 爬取某些关键信息 能做吗 怎么做* |
| *进程之间数据共享 除了管道还有什么方式* |
| *分布式爬虫 当奴隶端由于未知原因卡死 怎么能检测到 怎么分辨是项目挂掉还是阻塞等*  *这种问题之前没碰见过，但是可以按照这个思路执行：每个奴隶端用一个变量记录爬虫爬取的数据个数，以心跳包的方式定时向master汇报。如果有人没发，表示掉线，如果连续发了好几次数值都比较小，表示卡死* |
| *爬取app 公众号 的难点有哪些 怎么做数据反向编译和脱壳* |
| scrapy 和 selenium 结合爬取数据 怎么做 |
|  |
| 写出几种拼接字符串的方法  str1 = **'hello'**  str2 = **'world'**  str3 = str1 + str2  str3 = **"%s%s"**%(str1, str2)  str3 = **"{}{}"**.format(str1, str2)  str3 = f**"{str1}{str2}"** |
| 扫描指定文件夹下所有txt文件，查看文件中是否带有root这个关键字，有就打印该文件文件名   * + - 1. 根据路径遍历文件夹       2. 文件读写操作       3. 输出语句   **import** os  paths = os.listdir(**"./"**)  **for** path **in** paths:  **if** os.path.isdir(path):  print(**"文件夹"**, path)  **else**:  **with** open(path) **as** f:  **if "django" in** f.read():  print(**"文件"**, path) |
| 用列表表达式生成一个1990-2020年中闰年的列表。闰年：能被4整除但不能被100整除的年份，或者能被400整除的年份  years = [year **for** year **in** range(1990, 2021) **if** (year%4==0 **and** year%100!=0) **or** year%400==0]  print(years) |
| Python3.7新特性 |
| Python3支持几种不同的数据类型？ |
| 下列哪些是python3的内置函数  Bool format strcpy printf |
| Python3中dict转换成json类型对应的是  Array list object string |
| Mysql权限表分别是  1 user  2 host  3 db  4 tables\_priv  5 columns\_priv  6 procs\_priv |
| 如何获取表内所有字段对象的名称和类型\_\_\_\_\_\_\_ |
| Python读写excel文件的有\_3\_\_种模块，任意写出两种你最常用的模块：\_\_\_\_\_\_\_ 和 \_\_\_\_\_\_\_\_  用xlrd和xlwt进行excel读写；  用openpyxl进行excel读写；  用pandas进行excel读写； |
| 请用python3将out.xlsx文件的A1:D24区域的内容存为png格式的图片  import os  import win32com.client as win32  root\_path = os.path.dirname(os.path.abspath(\_\_file\_\_))  excel = win32.Dispatch('Excel.Application')  wb = excel.Workbooks.Open(os.path.join(root\_path, 'a.xlsx'))  ws = wb.Worksheets('Sheet1')  ws.Range('A1:D24').CopyPicture()  手动ctrl+v即可变为图片  可以考虑使用python操作剪切板 |
| 给你一个列表[‘红’, ‘黄’, ‘蓝’, [1, 2, 3], [4, 5, 6], [7, 8, 9]]，请复制一份，要求两份列表相互完全独立  **import** copy  list1 = [**'1'**, **'2'**, **'3'**, [4, 5, 6], [7, 8, 9]]  list2 = copy.deepcopy(list1) |
| 在linux平台上（centos6），运行python脚本查看服务状态。例如：脚本名字为status.py.执行python3 status,py httpd查看httpd服务的状态。提示：centos6下查看服务命令为：service <service-name> status  Python文件如何执行终端命令 os.system  Python文件如何接收终端参数 args |
| 写出以下代码的输出结果  fruit\_basket = [**'apple'**, **'banana'**, **'pear'**]  **for** fruit **in** fruit\_basket:  **if** fruit == **'peach'**:  print(**"peacg is in the basket"**)  **break**  **else**:  print(**"{} is in the basket"**.format(fruit))  **else**:  print(**"thank you!"**) |
| 画出一个你常用的web服务器的体系结构，并尽可能的描述该体系结构的了解 |
| 列举出三个你知道的设计模式，并简述你的理解 |
| 用纯js方式查找id为id\_checkcode\_img的元素，并设置它的宽度为150 |
| 描述下soap的交互流程以及实现原理，并列举python中自己曾用过的插件 |
| 对于一下函数哪些传参是正确的？（）多选题  Def f(a, b, c=’1’, d=’2’, \*args, \*\*kwargs):  Pass  A f(1,2,)  B f(b=1, a=2)  C f(a=1, 2, c=1)  D f(1,2, d=3, f=4)  E f(\*(1, 2), \*\*{‘c’: 1, ‘d’:2, ‘a’: 5})  F f(\*(1,), \*\*{‘c’: 1, ‘d’: 2, ‘b’: 5}) |
| 一行代码，通过filter和lambda函数输出以下列表索引为奇数对应的元素  List\_a = [12, 213, 22, 2, 2, 2, 22, 2, 2, 32] |
| 写出下面代码分别输出什么  Class A(object):  X = 1  Class B(A):  Pass  Class C(A):  Pass  Print(A.x, B.x, C.x)  B.x = 2  Print(A.x, B.x, C.x)  A.x = 3  Print(A.x, B.x, C.x)  1 1 1  1 2 1  3 2 3 |
| 数据库sql语句去重 |
| 限时15分钟，适用于1年以上python开发者  **from** functools **import** reduce  **from** math **import** ceil, sqrt  N = 10  print(0 == **False**) *# True*  print(0 **is False**) *# False*  print(0 == .0) *# True*  print(len([i **for** i **in** [**None**, **True**, **False**, -1, 0, 1, **''**, list(), tuple(), set(), dict(), [[]]] **if** i])) *# 4*  print(len(**'\t\nabc\t123\t\r\n'**.strip())) *# 7*  print(sum((range(N)[2: -2: 2]))) *# 12*  print(eval(**"-"**.join(map(**lambda** x: str(x\*\*2), range(N))))) *# -285*  print(len([list(range(N))[i: i+3] **for** i **in** range(0, N, 3)])) *# 4*  print([**None**, 0.0, **False**, 0, 1, **True**, 1.0].index(**True**)) *# 4*  print([**None**, 0.0, **False**, 0, 1, **True**, 1.0].index(0)) *# 1*  print(list(range(N)).sort(reverse=**True**)) *# None*  print({i: i **for** i **in** range(0, N)}.get(0, N)) *# 0*  print(reduce(**lambda** x, y: x+y, map(**lambda** x, y: x\*\*y, range(4), range(N)))) *# 33*  print(sum(filter(**lambda** x: N <= x\*sqrt(x) <= 2\*N, range(N)))) *# 18*  print(max({ceil(sqrt(x)) **for** x **in** range(2\*N)}))*# 5*  print(N.\_\_class\_\_.\_\_name\_\_) *# int*  print(len(**'你好\tutf8\n'**.encode(**'utf-8'**))) *# 12*  print(len(**'你好\tgb2312\n'**.encode(**'gb2312'**)))*# 12*  print(sqrt(N \*\* 2 // 2) // 1 % 2) *# 1.0*  print(getattr(N, **'bit\_length'**)()) *# 4* |
| A有一些石子，A扔掉一颗并把剩余数量的一半给了B，B扔掉一颗并把剩余数量的一半给C，C扔掉一颗后剩下四颗，A最初拥有几颗石子？ |
| 修一段路，A单独完成需要3天，AB两人共同完成需要2天，那么B单独完成需要几天？ |
| 有一家四口人的年龄和是147岁，爷爷比爸爸大38岁，妈妈比儿子大27岁，爷爷的年龄是儿子与妈妈年龄之和的2倍，问爸爸和妈妈的年龄相差多少岁？ |
| 有长短两只蜡烛，相同时间燃烧长度相同，它们的长度之和为56厘米，将它们同时点燃一段时间后，长蜡烛同短蜡烛点燃前一样长，这时短蜡烛的长度又恰好是长蜡烛的2/3。点燃前两只蜡烛长度只差为几厘米？ |
| 有四个数，其中每三个数的和分别为：0、1、2、3，这四个数分别是？ |
| 某家具厂用方木料90方，五合板600方，准备加工成书桌和书橱出售。已知生产每张书桌需要方木料0.1方，五合板2方，生产每个书橱需要方木料0.2方，五合板1方，出售一张方桌可获利80元，出售一个书橱可获利润120元。怎样安排生产可使所得利润最大？ |
| 已知某一个月份有五个星期二，而且其中的第二个星期天是13号，请问这个月的第一个星期一是几号？ |
| 市场上有三种投资产品的收益率依次为20%、5%和2%三档，单纯从收益上来看，这三者最有可能是哪几项（政府债券、股票、存款储蓄） |
| 2017年1月1日，小李在某银行存储定期为一年的存款100万元（一年期1.5%，活期收益率0.35%），2017年7月1日，他将钱取出后全部用于购买某银行半年期理财产品（年华收益率为4%）。到2017年12月31日，他合计获得多少利息？ |
| 有一杯水，甲乙两人要分了喝。一个公平的办法是：一个人分好水后，让另一个人先选。。则对于N个人分一杯水喝 的情况，采用什么方法能保证公平？ |
| 北京时间2000-01-02 03：04：05对应的UTC时间是？ |
| A类私有网络的广播地址是？ |
| 2字节无符号整型的表述范围是？ |
| SQL查询语句中分组和排序分别用哪两个关键词？ |
| 一颗含1023个节点的完美二叉树中相距最远的两节点间距是？ |
| 请写出3种常见的数据加密算法（非散列算法） |
| 请写出3种常见的浏览器内核 |
| 请写出3种常见的数据无损压缩算法 |
| 请写出3种常见的进程间通信方式  信号、管道、消息队列、共享内存 |
| 介绍一下getrefcount |
| http和TCP的区别 |
| Css的基本语句构成是 |
| 什么是盒子模型 |
| 什么是CDN |
| 什么是跨域 |
| 了解npm,node.js吗，简要描述 |
| MVC,MVP,MVT,MVVM分别是什么 |
| Var a=[]; a[0]=0;a[1]=1;a[4]=4;  请问a.length的值是多少？a[3]的输出结果是什么？ |
| typeof(null),typeof(undefined), typeof(Nan)的结果是 |
| 写出下列输出的值  Console.log(undefined || 1)  Console.log(null || NaN)  Console.log(0 && 1)  Console.log(0 && 1 || 0) |
| 适用于折半查找的表的存储方式\_\_顺序方式存储\_\_\_\_\_\_\_及元素排列要求为\_\_元素有序\_\_\_\_\_ |
| 查看某一个对象的所有属性和方法的命令是\_\_dir(对象名成)\_\_\_\_\_\_ |
| Linux查看cpu占用命令是：  https://blog.csdn.net/albenxie/article/details/72885951 |
| Linux查看80端口是否被占用，是哪个进行占用的？  netstat -anp |grep 80 |
| 在一个二维数组中（每个一维数组的长度相同），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。  **def** search(list1, value):  **for** list2 **in** list1:  **if** value **in** list2:  print(**"找到了"**)  **return True**  **return False** |
| **用两个栈实现队列，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.stack1 = []**  **self.stack2 = []**  **def push(self, node):**  **# write code here**  **self.stack1.append(node)**  **def pop(self):**  **# return xx**  **if self.stack2:**  **return self.stack2.pop()**  **elif not self.stack1:**  **return None**  **else:**  **while self.stack1:**  **self.stack2.append(self.stack1.pop())**  **return self.stack2.pop()** |
| **给定一个二叉树，找出其最大深度。**  **二叉树的深度为根节点到最远叶子节点的最长路径上的节点数。**  **class Solution:**  **def maxDepth(self, root):**  **if not root:#递归边界**  **return 0**  **else:**  **l=1+self.maxDepth(root.left)#递归**  **r=1+self.maxDepth(root.right)**  **return max(l,r)** |
| **Python内置的sorted函数采用的排序算法是？**  C:\Users\mac\Documents\Tencent Files\864071694\Image\Group\M{IJ{NB8DCHID~O$8PZ_F]L.png |
| 请问如何修改以下Python代码，使得下面的代码调用类A的show方法？  **class** A(object):  **def** show(self):  print(**"base show"**)  **class** B(A):  **def** show(self):  print(**"derived show"**)  obj = B()  obj.\_\_class\_\_ = A  obj.show() |
| **请问如何修改以下Python代码，使得代码能够运行？**  **class** A(object):  **def** \_\_init\_\_(self, a, b):  self.\_\_a = a  self.\_\_b = b  **def** myprint(self):  print(**f"a={self.\_\_a}, b={self.\_\_b}"**)  **def** \_\_call\_\_(self, num):  print(**f"call:{num+self.\_\_a}"**)  a1 = A(10, 20)  a1.myprint()  a1(80) |
| **下面这段代码的输出是什么？**  **class** B(object):  **def** fn(self):  print(**"B fn"**)  **def** \_\_init\_\_(self):  print(**"B init"**)  **class** A(object):  **def** fn(self):  print(**"A fn"**)  **def** \_\_new\_\_(cls, a):  print(**f"New {a}"**)  **if** a > 10:  **return** super(A, cls).\_\_new\_\_(cls)  **return** B()  **def** \_\_init\_\_(self, a):  print(**f"Init {a}"**)  a1 = A(5)  a1.fn()  a2 = A(20)  a2.fn()  **New 5**  **B init**  **B fn**  **New 20**  **Init 20**  **A fn** |
| **下面这段代码输出什么?**  ls = [1, 2, 3, 4]  list1 = [i **for** i **in** ls **if** i > 2]  print(list1)  list2 = [i\*2 **for** i **in** ls **if** i > 2]  print(list2)  dic1 = {x: x\*\*2 **for** x **in** (2, 4, 6)}  print(dic1)  dic2 = {x: **'item'** + str(x\*\*2) **for** x **in** (2, 4, 6)}  print(dic2)  set1 = {x **for** x **in 'hello world' if** x **not in 'low level'**}  print(set1)  **[3, 4]**  **[6, 8]**  **{2: 4, 4: 16, 6: 36}**  **{2: 'item4', 4: 'item16', 6: 'item36'}**  **{'r', 'h', 'd'}** |
| **下面这段代码输出什么?**  num = 9  **def** f1():  num = 20  **def** f2():  print(num)  f2()  f1()  f2()  **9**  **9** |
| **如何使用一行代码交换两个变量值？**  a = 8  b = 9  a, b = b, a |
| **如何添加代码，使得没有定义的方法都调用mydefault方法？**  **class** A(object):  **def** \_\_init\_\_(self, a, b):  self.a1 = a  self.b1 = b  print(**"init"**)  **def** mydefault(self):  print(**"default"**)  **def** \_\_getattr\_\_(self, item):  **return** self.mydefault  a1 = A(10, 20)  a1.fn1()  a1.fn2()  a1.fn3() |
| **一个包里有三个模块，mod1.py , mod2.py , mod3.py ，但使用 from demopack import \* 导入模块时，如何保证只有 mod1 、 mod3 被导入了。**  **在包中增加 \_\_init\_\_.py 文件，并在文件中增加： \_\_all\_\_ = [‘mod1’, ‘mod3’]** |
| **写一个函数，接收整数参数 n ，返回一个函数，函数返回n和参数的积。**  **def** mulby(num):  **def** gn(val):  **return** num \* val  **return** gn  zw = mulby(7)  print(zw(9)) |
| **请问下面的代码有什么隐患？（Python2中）**  **def** strtest1(num):  str = **'first'**  **for** i **in** range(num):  str += **'X'**  **return** str  **由于变量str是个不可变对象，每次迭代，python都会生成新的str对象来存储新的字符串，num越大，创建的str对象越多，内存消耗越大。** |
| **Odoo的标准组件有哪些？** |
| **Odoo常见的视图有哪些？** |
| **A = [2, 3, 4, 5, 6], 请用方法输出 b = [5, 6]** |
| **A = [‘a’, ‘b’, ‘c’], b = [1, 2] 请用方法输出 d = [[‘a’, 1], [‘b’, 2]]** |
| **Dict的items()方法与iteritems()方法的不同？** |
| **解释一个WSGI和FasCGI的关系？** |
| **介绍一下Python中webbrowser的用法** |
| **简述下class，function，module的命名空间** |
| **写一个签名函数，对一个dict格式的数据做md5签名** |
| **A = {‘2’: ‘a’, 2: ‘b’, “2”: ‘c’}, b=json.loads(json.dumps(a)),则b的值为{‘2’: ‘b’}** |
| 简述队列和堆栈的区别？  堆  堆中主要存放用new构造的对象和数组  优势：可以动态的分配内存的大小，生存期也不必事先告诉编译器，因为它是在运行时动态分配内存的。  缺点：由于要在运行时动态分配内存，存取速度比较慢。  栈  栈中主要存放一些基本类型的变量和对象引用类型。  优势：存取速度比较快，仅次于寄存器，栈数据可以共享。  缺点：栈中的数据大小和生存周期必须是确定的，缺乏灵活性。  队列  是设计程序中常用的一种数据结构，采用“先进先出”的存储结构，类似于队列。数据元素只能从队尾进入，从队首取出。在此队列中，数据元素可以随意增减，但是数据元素的次序不会更改。每次都是取出队首的元素，后面的元素会整体向前移动一位。队列便利数据的速度要快的多 |
| 进程中堆和栈的区别  栈和堆的区别：(背下来)  栈：   a、快速访问。   b、没有必要明确的创建分类变量，因为它是自动管理的。   c、空间被CPU高效地管理着，内存不会变成碎片。   d、只有局部变量   e、受限于栈大小(取决于操作系统)   f、变量不能调整大小。  堆：   a、变量可以被全局访问   b、没有内存大小限制   c、（相对）访问比较慢   d、没有高效地使用空间，随着块内存的创建和销毁，内存可能会变成碎片。   e、你必须管理内存（变量的创建和销毁你必须要负责）   f、变量大小可以用realloc()调整 |
| **悲观锁和乐观锁的区别？使用时应该如何选择？** |
| **简述一下Linux命令的用途：ps, top, grep, du, df**  ps命令是最基本同时也是非常强大的进程查看命令，使用该命令可以确定有哪些进程正在运行和运行的状态、进程是否结束、进程有没有僵死、哪些进程占用了过多的资源等等，总之大部分信息都是可以通过执行该命令得到的。  top命令可以实时动态地查看系统的整体运行情况，是一个综合了多方信息监测系统性能和运行信息的实用工具。通过top命令所提供的互动式界面，用热键可以管理。  grep（global search regular expression(RE) and print out the line，全面搜索正则表达式并把行打印出来）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。  du命令也是查看使用空间的，但是与df命令不同的是Linux du命令是对文件和目录磁盘使用的空间的查看，还是和df命令有一些区别的。  df命令用于显示磁盘分区上的可使用的磁盘空间。默认显示单位为KB。可以利用该命令来获取硬盘被占用了多少空间，目前还剩下多少空间等信息。 |
| **如何在linux下压缩和解压缩文件夹？**  tar -zcvf 打包后生成的文件名全路径 要打包的目录  tar -xf 要解压的压缩包路径 |
| **TCP/IP协议栈有哪几层？路由器工作在哪一层？**  应用层FTP SMTP HTTP ...  传输层TCP UDP  IP网络层IP ICMP IGMP  网络接口层ARP RARP以太网令牌环FDDI ...  路由器工作在IP网络层 |
| **软件的生命周期**  软件生命周期又称为软件生存周期或系统开发生命周期，是软件的产生直到报废的生命周期，周期内有问题定义、可行性分析、总体描述、系统设计、编码、调试和测试、验收与运行、维护升级到废弃等阶段，这种按时间分程的思想方法是软件工程中的一种思想原则，即按部就班、逐步推进，每个阶段都要有定义、工作、审查、形成文档以供交流或备查，以提高软件的质量。但随着新的面向对象的设计方法和技术的成熟，软件生命周期设计方法的指导意义正在逐步减少。 生命周期的每一个周期都有确定的任务，并产生一定规格的文档（资料），提交给下一个周期作为继续工作的依据。按照软件的生命周期，软件的开发不再只单单强调“编码”，而是概括了软件开发的全过程。软件工程要求每一周期工作的开始只能必须是建立在前一个周期结果“正确”前提上的延续；因此，每一周期都是按“活动 ── 结果 ── 审核 ── 再活动 ── 直至结果正确”循环往复进展的。 |
| **什么叫设计模式？简述之前用过的设计模式？**  设计模式（Design pattern）代表了最佳的实践，通常被有经验的面向对象的软件开发人员所采用。设计模式是软件开发人员在软件开发过程中面临的一般问题的解决方案。这些解决方案是众多软件开发人员经过相当长的一段时间的试验和错误总结出来的。  设计模式是一套被反复使用的、多数人知晓的、经过分类编目的、代码设计经验的总结。使用设计模式是为了重用代码、让代码更容易被他人理解、保证代码可靠性。  用过的设计模式：  1. 单例模式，保证整个程序在运行期间某一个类的实例对象只存在一个，不会在内存中创建多个对象。  2. 装饰器模式，在后台开发中，可以利用装饰器给某一个视图函数添加一个额外的功能模块，并且django中也自带了很多的装饰器用于用户的认证登录等功能。  3. MVC设计模式，后台框架flask就是采用MVC的结构来组织整个项目结构的。  4. MTV模式，后台框架django采用的是MTV结构组织整个项目结构。 |
| **什么叫观察者模式？该设计模式主要用于解决什么问题？**  观察者模式（又被称为发布-订阅（Publish/Subscribe）模式，属于行为型模式的一种，它定义了一种一对多的依赖关系，让多个观察者对象同时监听某一个主题对象。这个主题对象在状态变化时，会通知所有的观察者对象，使他们能够自动更新自己。  解决什么问题？  1.关联行为场景，需要注意的是，关联行为是可拆分的，而不是“组合”关系。  2.事件多级触发场景。  3.跨系统的消息交换场景，如消息队列、事件总线的处理机制。 |
| **什么是时间复杂度？空间复杂度？**  时间复杂度是指执行算法所需要的计算工作量；  而空间复杂度是指执行这个算法所需要的内存空间。  （算法的复杂性体现在运行该算法时的计算机所需资源的多少上，计算机资源最重要的是时间和空间（即寄存器）资源，因此复杂度分为时间和空间复杂度）。  简单来说，时间复杂度指的是语句执行次数，空间复杂度指的是算法所占的存储空间 |
| **对以下算法时间复杂度进行排序**  **O(2^n)**  **O(n^100)**  **O(n^2)**  **O(n^n)**  **结果是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |
| **现有数组A[N]，含有N个数字，写一个算法时间复杂度为O(N)的函数来计算数组B[N]，其中B[i] = A[1]\*A[2]\*…A[i+1]\*…\*A[N-1]\*A[N](从A【1】到A[N]除了A[i]的乘法积)，算法过程不能使用除法。** |
| **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?**  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  l = []  **for** i **in** range(10):  l.append({**'num'**: i})  print(l)  l = []  a = {**'num'**: 0}  **for** i **in** range(10):  a[**'num'**] = i  l.append(a)  print(l)  **两段代码结果不相同**  **结果1：[{'num': 0}, {'num': 1}, {'num': 2}, {'num': 3}, {'num': 4}, {'num': 5}, {'num': 6}, {'num': 7}, {'num': 8}, {'num': 9}]**  **结果2：[{'num': 9}, {'num': 9}, {'num': 9}, {'num': 9}, {'num': 9}, {'num': 9}, {'num': 9}, {'num': 9}, {'num': 9}, {'num': 9}]**  **结果1的原因是：每次循环创建一个新字典，字典之间互不影响**  **结果2的原因：每次循环使用的是同一个字典，修改的只是字典内部的数值** |
| **请尝试用一行代码实现随机数数列表以3为单位分组**  **n = 10**  **print([[x for x in range(1, n+1)][i:i+3] for i in range(0, n, 3)])** |
| **写一个斐波那契数列，简单地说，起始两项为0和1，此后的项分别为它的前两项之和。**  def fibo(n):  x, y = 0, 1  while(n):  x,y,n = y, x+y, n - 1  return x  def fibo(n):  def fib\_iter(n,x, y):  if n == 0:  return x  else:  return fib\_iter(n-1, y, x+y)  return fib\_iter(n, 0, 1) |
| **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？**  **要考虑的问题：**  **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。**  **要考虑每次读入数据的大小，太小就会在读取操作上花费过多的时间。**  **修改后的代码：**  **def get\_lines():**  **l=[]**  **with open('file.txt','rb') as f:**  **data=f.readlines(6000)**  **l.append(data)**  **yield l**  **书中给的代码并不能循环迭代，修改如下。**  **def get\_lines():**  **# l=[]**  **flag=True**  **with open('file.txt','rb') as f:**  **while flag:**  **data=f.readlines(1000)**  **# l.append(data)**  **if data:**  **yield data**  **else:**  **flag=False** |
| 4G内存如何读取一个5G的数据？  方法一：  可以通过生成器，分多次读取，每次读取比较少量的数据进行处理，处理结束后在一次读取后面的数据。  方法二：  可以通过Linux命令split切割成小文件，然后对数据进行处理，此方法效率比较高。可以按照行数切割，也可以按照文件大小切割。 |
| 补充缺失的代码。  def print\_directory\_contents(sPath):  '''  这个函数接收文件夹的名称作为输入参数。  返回该文件夹中文件的路径  以及其包含文件夹中文件的路径  '''  #请补充代码  完整代码如下，  #首先导入os库  import os  def print\_directory\_contents(sPath):  #os.listdir() 方法用于返回指定的文件夹包含的文件或文件夹的名字的列表。  #只支持在 Unix, Windows 下使用。  for sChild in os.listdir(sPath):  #os.path.join(path1,path2,...)用于路径拼接文件路径  sChildPath=os.path.join(sPath,sChild)  #os.path.isdir()函数判断是否为文件夹，是文件夹返回True.  if os.path.isdir(sChildPath):  print\_directory\_contents(sChildPath)  else:  print(sChildPath) |
| 在except中return后还会执行finally中的代码吗？如何抛出自定义异常？  return之后依然执行finally中的代码。  用raise方法可以抛出自定义异常。 |
| 题目：请写一个函数，函数的输入是一个string类型的英文句子，函数的输出也是一个string类型的应为句子：是把输入的句子里面的单词按逆序输出，但是每个单词按内部的字符按顺序输出  示例：  输入： I like you, but just I like you.  输出： .you like I just but, you like I  注意事项：  标点符号当作一个单词处理  **import** re  **def** parse(content):  **if not** isinstance(content, str):  **raise** TypeError(**"不是一个字符串"**)  pattern = re.split(**r'([ ,.])'**, content)  content = **""**.join(pattern[::-1])  **return** content  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  content = **"i like you, but just i like you."**  result = parse(content)  print(**f"输出结果为：{result}"**) |
| **class** Rect(object):  **def** \_\_init\_\_(self, left, right, top, bottom):  self.left = left  self.right = right  self.top = top  self.bottom = bottom  **def** area(self):  **return** abs((self.right-self.left)\*(self.top-self.bottom))  **def** is\_chongdie(r1, r2):  **return not** (r1.left > r2.right **or** r1.top > r2.bottom **or** r2.left > r1.right **or** r2.top > r1.bottom)  M1 = [(), (), (), (), ()]  m1\_left = min([x **for** x, y **in** M1])  m1\_right = max([x **for** x, y **in** M1])  m1\_top = max([x **for** x, y **in** M1])  m1\_bottom = min([x **for** x, y **in** M1])  m1 = Rect(m1\_left, m1\_right, m1\_top, m1\_bottom)  M2 = [(), (), (), (), ()]  m2\_left = min([x **for** x, y **in** M2])  m2\_right = max([x **for** x, y **in** M2])  m2\_top = max([x **for** x, y **in** M2])  m2\_bottom = min([x **for** x, y **in** M2])  m2 = Rect(m2\_left, m2\_right, m2\_top, m2\_bottom)  **if** is\_chongdie(m1, m2):  area\_left = max(m1.left, m2.left)  area\_right = min(m1.right, m2.right)  area\_top = min(m1.top, m2.top)  area\_bottom = max(m1.bottom, m2.bottom)  area\_width = area\_right-area\_left  area\_height = area\_top - area\_bottom  area = abs(area\_width \* area\_height)  **if** area == 0:  print(**"重叠一个点或一条线"**)  **elif** area == m1.area() **or** area == m2.area():  print(**"完全重合"**)  **else**:  print(**"重叠一部分"**)  **else**:  print(**"不重叠"**) |

1. 笔试题

|  |
| --- |
| 一个整数，大于0，不用循环和本地变量，按照n,2n,4n,8n的顺序递增，当值大于5000时，把值按照指定顺序输出来。例如：n=1237，则输出为：1237，2474，4948，9896，9896，4948，2474，1237，（上海爱福窝云技术有限公司） |
| 将一个键盘输入的数字转化成中文输出（例如：输入1234567，输出一百二拾三万四千五百六拾七）（上海爱福窝云技术有限公司） |
| 判断101-200之间有多少个素数，并输出所有素数（上海爱福窝云技术有限公司） |
| 给定一个字符串，输出本字符串中只出现一次并且最靠前的那个字符的位置？比如abcccddeeefa 则是b，输出2（上海爱福窝云技术有限公司） |
| 写一个函数找出一个整数数组中，第二大的数（上海爱福窝云技术有限公司） |
| 设计一个3人斗地主的发牌算法（54张牌发给3个人，剩3张）（上海爱福窝云技术有限公司） |
| 简述QQ登录过程（上海爱福窝云技术有限公司）  qq登陆过程：1、首先QQ客户端向服务器发送一个请求登录令牌的数据包.服务器返回登录令牌.这个令牌是在服务器端生成的；2、在QQ客户端得到登录令牌之后.就会向服务器发送一个包含登录信息的登录请求.要求登录.3、服务端验证客户端的登录信息是否与服务器上保存的登录信息进行比较.匹配的话就向客户端返回一个登录成功的数据包.不匹配返回登录失败. |
| 输入一个字符串，打印出该字符串中字符的所有排列（上海爱福窝云技术有限公司） |
| 设计三张表，表结构如下(live800数据工程师笔试题)  Student表  Id name sex age department  Course表  Id name credit  Sc表  Id stu\_id course\_id score  完成以下问题  1． 把course表中id为3的课程的credit修改为5  2． 在以上3个表中查询选的课程的学分为5，并且成绩大于70的学生的学号、姓名  3 . 统计所有学生的学分总数 |
| 数据库日志的作用是什么？(live800数据工程师笔试题) |
| 请说出ArrayList，Vector，LinkedList的存储性能和特性(live800数据工程师笔试题) |
| 下列输出结果是  #include <stdio.h>  int main(void) {  int i = 0, s = 0;  do {  if (i % 2 == 0) {  i ++;  continue;  }  i ++;  s = s + i;  } while (i < 7);  printf("%d",s);  return 0;  }  12 |
| Python中合并两个字典，key值相同则保留最大值，不同key值保留(live800数据工程师笔试题)  dica = {**'a'**: 1, **'b'**: 2, **'c'**: 3, **'d'**: 4, **'f'**: **'hello'**}  dicb = {**'b'**: 3, **'d'**: 5, **'e'**: 7, **'m'**: 9, **'k'**: **'world'**}  **for** keya, valuea **in** dica.items():  **if** keya **in** dicb:  **if** valuea > dicb[keya]:  dicb[keya] = valuea  **else**:  dicb[keya] = valuea  print(dicb) |
| 使用python创建一个文件名中含有日期的文件(live800数据工程师笔试题)  from datetime import datetime  open(datetime.now().date().isoformat()+'.txt', 'w').close() |
| 有表List，并有字段A,B,C，类型都是整数，表中有如下几条记录  A B C  2 7 9  5 6 4  3 11 9  现在对该表依次完成以下操作  1.查询出B和C列的值，要求按B列升序排列  Select b,c from list order by b aes;  2.写出一条新的记录，值为(7，8，9)  Insert into list values (7, 8, 9);  3.查询C列，要求消除重复的值，按降序排列  Select distinct(c) from list order by c desc; |
| 写出下列表达式的值  print(bool(**'False'**)) *# True*  print(~~~5) *# -6*  print(0.1 + 0.2 == 0.3) *# False*  print([i\*i **for** i **in** range(5) **if** i%2 == 0]) *# [0, 4, 16]*  print((0, 0) == 0, 0) *# False, 0*  print(0, 0 == (0, 0)) *# 0, False* |
| 写出下列代码的执行结果  x = 1  x, y = 2, x  print(y) # 1  person = {**'nice'**: 1991, **'alice'**: 1992, **'kict'**: 1993, **'nice'**: 1995, **'bob'**: 1992}  print(person[**'alice'**], person[**'nice'**])  # 1992 1995  **class** Cat(object):  age = 5  color = **'gray'**  **def** \_\_init\_\_(self, color):  self.color = color  Cat.age += 1  cat1 = Cat(**'black'**)  cat2 = Cat(**'white'**)  print(cat1.age, cat2.age, Cat.age)  cat1.color = **'red'**  print(cat1.color, cat1.color, Cat.color)  # 7 7 7  # red red gray  **def** func():  **return** [**lambda** x : i\*x **for** i **in** range(5)]  print([m(2) **for** m **in** func()])  # [8, 8, 8, 8, 8] |
| 简答题  1.简述del和pop()的区别  pop(index)方法是对可变序列中元素下标进行检索删除，返回删除值  del(list[index])方法是对可变序列中元素下边进行检索删除，不返回删除值  2.python中怎么实现private、project  \_\_foo\_\_: 定义的是特殊方法，一般是系统定义名字 ，类似 \_\_init\_\_() 之类的。  \_foo: 以单下划线开头的表示的是 protected 类型的变量，即保护类型只能允许其本身与子类进行访问，不能用于 from module import \*  \_\_foo: 双下划线的表示的是私有类型(private)的变量, 只能是允许这个类本身进行访问了。  foo:就是public方法 |
| [小明去附近的水果店买橙子，水果商贩只提供整袋购买，有每袋6个和每袋8个的包装（包装不可拆分）。可是小明只想购买恰好n个橙子，并且尽量少的袋数方便携带。如果不能购买恰好n个橙子，小明将不会购买。请根据此实现一个程序，要求： 输入一个整数n，表示小明想要购买n（1≤n≤100）个橙子 输出一个整数表示最少需要购买的袋数，如果不能买恰好n个橙子则输出-1 例如，输入20，输出3。](https://www.cnblogs.com/yikuan-919/p/9331063.html) |
| 有一个n阶楼梯，一次一步或者两步，只能进不能退，定义一个函数算算有多少种爬法？ |
| 表达式3<4<5是哪个表达式的缩写 |
| 获取python解释器版本的办法是  python -v |
| 如果模块是被导入的,\_\_name\_\_的值是\_\_ 模块名称\_\_\_\_;如果模块是被直接执行的,\_\_name\_\_的值是\_\_\_\_\_main\_\_\_\_\_\_ |
| Python的内存管理中，为了追踪内存中的对象，使用了\_\_引用计数\_\_\_\_\_\_ |
| 实现函数，判断一个字符串是不是另外一个字符串的变形（包含的字符及每个字符数量相同，仅排列不同）。例如good和dogo应返回True，car和cdr应返回False |
| 查找给定目标的区间范围  给定一个包含n个整数的排序数组，找出给定目标值target的起始和结束位置。如果目标值不在数组中则返回[-1, -1]  样例  给出[5, 7, 7, 8, 8, 10]和目标值target=8，返回[3, 4]  挑战  时间复杂度O(logn) |
| 查找滑动窗口最大值的区间  描述  给出一个可能包含重复的整数数组，和一个大小为k的滑动窗口，从左到右在数组中滑动这个窗口，找到数组中每个窗口内的最大值。  样例  给出数组[1, 2, 7, 7, 8], 滑动窗口大小为k=3，返回[7, 7, 8]  解释  最开始窗口的状态如下[[1, 2, 7], 7, 8]，最大值为7  然后窗口向右移动一位  [1, [2, 7, 7], 8]，最大值为7  最后窗口再向右移动一位  [1, 2, [7, 7, 8]]，最大值为8  挑战  O(n)时间，O(k)的额外空间 |
| SQL查询：查找各个部门的最高工资    结果如下    SELECT  deprtment.`name` AS `department`,  employee.`name` AS `employee`,  max(employee.salary) AS `salary`  FROM  employee  RIGHT JOIN deprtment ON deprtment.id=employee.departmentId  GROUP BY  employee.departmentId; |
| X,y均取值于列表[1, 2, 3, 4, 5]，生成x为键，y为值的字典，其中x为奇数，y为偶数。  list1 = [1, 2, 3, 4, 5]  dict1 = {(x, y) **for** x **in** list1 **if** x % 2 == 1 **for** y **in** list1 **if** y % 2 == 0}  print(dict1) |
| 某学校进行分班考试，现有150个学生考试结果，请将前50名放入A班，中间50名放B班，最后50名放C班。倾斜一个存储过程实现。 |
| 写一个分数类并实现加减乘除运算  **class** Faction(object):  **def** \_\_init\_\_(self, fz=0, fm=1):  self.\_\_fz = fz  **if** fm == 0:  **raise** ValueError(**'分母不能为0'**)  **else**:  self.\_\_fm = fm  @property  **def** fz(self):  **return** self.\_\_fz  @property  **def** fm(self):  **return** self.\_\_fm  **def** chang\_format(self):  min\_value = min(self.\_\_fz, self.\_\_fm)  max\_value = max(self.\_\_fz, self.\_\_fm)  **while** max\_value % min\_value != 0:  temp = max\_value % min\_value  max\_value = min\_value  min\_value = temp  self.\_\_fz //= min\_value  self.\_\_fm //= min\_value  **if** self.\_\_fm == 1:  str = **"{fz}"**.format(fz=self.\_\_fz)  **else**:  str = **"{fz}/{fm}"**.format(fz=self.\_\_fz, fm=self.\_\_fm)  **return** str  **def** \_\_add\_\_(self, other):  fz = self.fz \* other.fm + self.fm \* other.fz  fm = self.fm \* other.fm  **return** Faction(fz, fm).chang\_format()  **def** \_\_sub\_\_(self, other):  fz = self.fz \* other.fm - self.fm \* other.fz  fm = self.fm \* other.fm  **return** Faction(fz, fm).chang\_format()  **def** \_\_mul\_\_(self, other):  fz = self.fz \* other.fz  fm = self.fm \* other.fm  **return** Faction(fz, fm).chang\_format()  **def** \_\_truediv\_\_(self, other):  fz = self.fz \* other.fm  fm = self.fm \* other.fz  **return** Faction(fz, fm).chang\_format()  a = Faction(2, 3)  b = Faction(4, 3)  print(a/b) |
| 现在有三种不同重量的标准砝码1g, 3g, 8g,请问可以称出多少不同物品的重量？在进行称量时，要称的东西与已知的标准砝码可以任意的放在天平的两盘之一，另外，每种砝码都只有一个，而且不准赋值（基本的组合能力）  11种 |
| 有两只乌龟一起赛跑，甲龟到达12米终点线时，乙龟才跑到9米，为了让两只乌龟同时到达重点，甲龟需要从起跑线后退多少米，两只乌龟都是匀速跑步（基本的比例能力）  4米 |
| 一位先生要到10层楼的第8层去办事，不巧正赶上停电，电梯无法使用，智能步行上楼。如果他从第一层爬到第四层需要48秒，请问，以同样的速度从第一层走到第八层需要多少秒？（细节的严谨性）  112秒 1-4层只用爬3层，每层16秒。1-8层需要爬7层 |
| 阿聪和阿傻到公园玩，他俩想买一瓶可乐喝，阿聪差1元，阿傻差一分。他俩的钱合起来钱还是不够，请问一瓶可乐多少钱？（简单的排除法）  1.1元以上 |
| 2的100次方，其个位数是（基本的归纳综合能力）  6 n%4=1则是2 n%4=2则是4 n%4=3则是8 n%4=0 则是6 |
| 在一次登山活动中，同学们每小时走2里路，登上山顶后，按原路返回，每小时走6里路，求登山的平均速度（里/时）  4里/时 = （2+6）/ 2 |
| 有99个硬币，甲乙轮流拿，一次最少拿一个最多拿四个，谁拿到最后一个就算赢，现在甲先拿，先拿多少个可以确保甲一定赢  99%5=4，先拿4个，剩下的对方取多少你就取5-n即可 |
| 袋中有n只球，收球规则是“每次拿出袋中球的一半后再放进去一只”，算取一次。取了832次后袋中剩下2只球，那么你知道在开始时袋中有多少只球吗？  2只 |
| 54张牌，三个人打，大小王同时在一个人手上的概率是多少？  1/3 |
| 一个细菌一分钟由一个分裂成两个，两分钟后分裂成四个，把一个这样的细菌放在一个瓶子里，一小时后瓶子被充满，现将两个这种细菌放在瓶子里，多久瓶子被充满？  59分钟 |
| 代码题：同时投掷三个骰子，可能得到的最小点数为3，最大点数为18，请使用第三方库绘制一个1000次同时投掷三个骰子得出的点数的直方图 |
| **class** Man(object):  **def** run(self):  print(**"run Man run!"**)  **def** stop(self):  print(**"stop Man stop!"**)  **def** pause(self):  **raise** Exception(**'Not Implemented'**)  **class** Forrest(Man):  **def** run(self):  super(Forrest, self).run()  print(**'run Forrest run!'**)  **class** Jenny(Man):  **def** run(self):  super(Jenny, self).run()  print(**'run Jenny run!'**)  **def** stop(self):  super(Jenny, self).stop()  print(**'stop Jenny stop!'**)  **class** LittleForrest(Forrest, Jenny):  **def** run(self):  super(LittleForrest, self).run()  print(**'run LittleForrest run!'**)  **def** stop(self):  super(LittleForrest, self).stop()  print(**'stop LittleForrest stop!'**)  **def** pause(self):  print(**'wait LittleForrest wait!'**)  **class** E(Forrest, Jenny):  **pass**  man = Man()  forrest = Forrest()  jenny = Jenny()  littleForrest = LittleForrest()  e = E()  *# 下面的输出语句是什么？*  man.run()  forrest.run()  jenny.run()  littleForrest.run()  e.run()  man.stop()  forrest.stop()  jenny.stop()  littleForrest.stop()  e.stop()  man.pause()  forrest.pause()  jenny.pause()  littleForrest.pause()  e.pause()  1.  run Man run!  2.  run Man run!  run Forrest run!  3.  run Man run!  run Jenny run!  4.  run Man run!  run Jenny run!  run Forrest run!  run LittleForrest run!  5.  run Man run!  run Jenny run!  run Forrest run!  6.  stop Man stop!  7.  stop Man stop!  8.  stop Man stop!  stop Jenny stop!  9.  stop Man stop!  stop Jenny stop!  stop LittleForrest stop!  10  stop Man stop!  stop Jenny stop!  11  Exception: Not Implemented  12  Exception: Not Implemented  13  Exception: Not Implemented  14  wait LittleForrest wait!  15  Exception: Not Implemented |
| Given an array of size n, find the majority element, The majority element is the element that appears more than [ n/2] times.  You may assume that the array is non-empty and the majority element always exist in the array.  class Solution:  # @param {integer[]} nums  # @return {integer}  def majorityElement(self, nums):  lenth=len(nums)  index=0  count=1  for i in range(lenth):  if nums[index]==nums[i]:  count+=1  else:  count-=1  if count==0:  index=i  count+=1  return nums[index] |
| Given an unsorted array of integers, find the length of the longest consecutive elements sequence. 给定一个未排序的整数数组，找出最长连续序列的长度。  Given [100, 4, 200, 1, 3, 2]  The longest consecutive elements sequence is [1, 2, 3, 4]. Return its length: 4 |
| 从下面的字典中查找出值为数字的字符串，并显示出对应字典的键名和该值  a = {  **'var1'**: [**'key6'**, **'data2'**, **'dss6'**],  **'var2'**: [**'key2'**, **'data1'**, **'dss3'**],  **'var3'**: [**'dddw'**, **'3234'**, **'bbsss'**],  }  result\_dict = {}  **for** key, value **in** a.items():  **for** num **in** value:  **if** num.isdigit():  **if** key **in** result\_dict:  result\_dict[key].append(value)  **else**:  result\_dict[key] = [num]  print(result\_dict) |
| A = [**'var1'**, **'var2'**, **'var3'**, **'var4'**],B = [**'value1'**, **None**, **'value3'**, **None**]，把列表B中的None替换成“”，然后把列表A与B对应位置的值生成为要给字典键值对，A为健，B为相对应的值。执行结果为：C = {**'var4'**: **''**,**'var1'**: **'value1'**,**'var3'**: **'value3'**, **'var2'**: **''**}  C = {}  **for** key, value **in** zip(A, B):  C[key] = value **if** value **else ""**  print(C) |
| 请写出应用IOCP的工作流程 |
| 请画出你曾经使用的一个网络服务器架构，并说明各部分功能。 |
| 对于以下函数哪些传参是正确的？  Def f(a,b,c=’1’, d=’2’, \*args, \*\*kwargs):  Pass  A: f(1, 2,)  B: f(b=1, a=2)  C: f(a=1, 2, c=1)  D: f(1, 2, d=3, f=4)  E: f(\*(1, 2), \*\*{‘c’:1, ‘d’: 2, ‘a’: 5})  F: f(\*(1, ), \*\*{‘c’: 1, ‘d’: 2, ‘b’: 5}) |
| 深圳市掌世界网络科技有限公司  对mygame表中game，userid相同的记录进行去重，表记录如下：  Id game userid  1 a 111111  3 a 111111  5 b 222222  6 b 222222  7 c 222222  8 d 111111  9 a 111111  去重后结果如下  Id game userid  6 b 222222  7 c 222222  8 d 111111  9 a 111111  表结构如下  CREATE TABLE `mygame` {  `id` int(11) not null auto\_increment,  `game` varchar(50) default null,  `userid` int(11) default null,  `create\_datetime` datetime default null,  primary key(`id`)  } ENGINE=INNODB AUTO\_INCREMENT=6 DEFAULT CHARSET=utf8; |
| 编写要给函数将ip地址转化为主机名称，主机名称为ip每段不足3位前面补零 “10.10.10.1” –》 “010010010001”  **def** change\_ip(ip):  ip\_list = []  ip\_parts = ip.split(**"."**)  **for** part **in** ip\_parts:  part = int(part)  part = **"%.3d"** % part  ip\_list.append(part)  **return ""**.join(ip\_list)  ip = **'192.168.1.1'**  print(change\_ip(ip)) |
| **以下描述错误的是(百乐润信息技术深圳有限公司)**  **import** threading  **import** time  **def** func(i, j, k):  **while True**:  print(i\*j\*k)  i += 1  time.sleep(1.0)  t = threading.Thread(name=**"test"**, target=func(), args=(10, 12, 7), daemon=**True**)   1. 这个线程被指定了一个名字“test” 2. 如果不主动执行t.join(), 线程t不会阻塞其他线程 3. 函数func是线程t的线程函数 4. 线程没有被启动 |
| 对如下代码描述正确的是**(百乐润信息技术深圳有限公司)**  v1 = 100.0  **def** pow(x, y):  **return** x\*\*(y+1)  **def** func(x, y):  **from** math **import** pow  v1 = pow(x, y)  v1 += 1  **return**  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  func(2, 2)  print(v1)   1. 最终打印输出的是5.0 2. 不能再函数中执行from math import pow 3. 最终打印输出的是9.0 4. **最终的打印输出是100.0** |
| 对如下代码描述正确的是**(百乐润信息技术深圳有限公司)**  **def** f(\*args):  print(args)    **def** g(\*\*kwargs):  print(kwargs)   1. 函数f的调用方式是f(‘a’=1, ‘b’=2) 2. **函数g的调用方式是g(‘a’=1, ‘b’=2)** 3. **函数f的调用方式是f(1, 2)** 4. 函数g的调用方式是g(1,2) |
| 创建一个生成器应该使用关键字**(百乐润信息技术深圳有限公司)**   1. Yield 2. With 3. Generator 4. Iterator |
| 下列描述错误的是**(百乐润信息技术深圳有限公司)**   1. 上下文管理器用关键字with定义 2. 支持上下文管理器的对象必须定义\_\_enter\_\_()和\_\_exit\_\_()函数清理上下文 3. With代码段内部的语句执行正常完成时，能够自动调用\_\_exit\_\_()函数清理上下文 4. With代码段内部的语句产生异常时，不能自动调用\_\_exit\_\_()函数，需要自行捕捉异常并清理上下文 |
| 关于如下代码，描述错误的时**(百乐润信息技术深圳有限公司)**  **import** types  **def** foo(x, y):  **return** x+1, y+1   1. 表达式isinstance(foo, types.MethodType)的值位True 2. 表达式callable(foo)的值为True 3. 可以调用foo函数为a, b = foo(1, 2) 4. 可以调用foo函数为c = foo(1, 2) |
| 关于lambda表达式，描述错误的是：**(百乐润信息技术深圳有限公司)**   1. Lambda表达式可以赋值给一个变量，也可以作为返回值 2. Lambda表达式不需要return，直接返回表达式的值 3. Lambda表达式支持多个表达式，以；分割 4. Lambda表达式是一个匿名函数 |
| 关于python的类，正确的说法是**(百乐润信息技术深圳有限公司)**   1. 类定义不是对象 2. Type()函数可以返回一个类定义 3. Python的类不支持多重继承 4. 自定义类的\_\_init\_\_函数会在执行完自定义代码后自动调用父类的\_\_init\_\_函数 |
| 在python3环境，下面代码的执行结果是**(百乐润信息技术深圳有限公司)**  **def** foo(x, y):  **return "{:.02f}"**.format(x/y), **"{:.02f}"**.format(x//y), round(x+y-0.5)  print(foo(8, 3))  ('2.67', '2.00', 10) |
| 关于异常处理，下面哪个说法是错误的**(百乐润信息技术深圳有限公司)**   1. 表达式except Exception, err:是python3环境下的异常捕捉方式 2. Finally子句如果存在，不管是否发生异常都会被执行 3. 在一个try代码段后面，可以跟随多个except自句捕捉不同的异常 4. Else子句如果存在，只会在前方的try代码段不发生异常时才会被执行 |
| 如下代码的执行结果是**(百乐润信息技术深圳有限公司)**  a = 100  b = 10  **def** foo(x, y, m, n):  **global** a, b  **for** item **in** y:  **if** item **in** x:  x[item] += 1  a = x[**"a"**]  b = x[**"b"**]  m = x[**"a"**] + 1  n = x[**"b"**] + 1  **return**  c = {**"a"**: a, **"b"**: b}  d = [**"a"**, **"b"**, **"c"**, **"d"**]  foo(c, d, a, b)  print(a, b, c)  101 11 {'a': 101, 'b': 11} |
| 如果不想修改一个函数的定义，但是又想在运行时动态增加函数的特性，可以使用**(百乐润信息技术深圳有限公司)**   1. Generator 2. Decorator 3. Iterator 4. Functools, partial |
| 关于select/poll/epoll，下面说法错误的时**(百乐润信息技术深圳有限公司)**   1. 通常select只能够监控1024个文件句柄 2. Poll解决了select的句柄数限制 3. Select/poll在监控事件发生后，都需要遍历所有监控的文件句柄 4. Epoll通过将文件句柄列表从用户空间拷贝到内核空间来提高效率 |
| 关于多进程模块multiprocessing的说法，错误的是**(百乐润信息技术深圳有限公司)**   1. Multiprocessing提供了与threading相似的进程同步机制，比如lock，semaphore 2. 计算密集型任务并发，使用多线程不能提高效率，需要多进程 3. 多进程间的数据交换可以使用哦multiprocessing提高的pipe，queue 4. 多进程间的数据交换不能像多线程一样使用共享变量 |
| 关于python常用的内建模块，描述错误的是**(百乐润信息技术深圳有限公司)**   1. Numpy提供了多维数组及其基本操作 2. Hashlib提供了常见的摘要算法 3. Datetime提供了处理日期/时间的操作 4. Csv提供了csv文件的读/写/解析操作 |
| 不是python下的web后台开发框架式**(百乐润信息技术深圳有限公司)**   1. Flask 2. Vue 3. Django 4. Tornado |
| 关于async/wait说法错误的是**(百乐润信息技术深圳有限公司)**   1. 可以用来实现协程 2. Await可以接受一个普通生成器或awaitable对象 3. Await作用与yield from类似 4. [Async作用与@asyncio.coroutine](mailto:Async作用与@asyncio.coroutine)类似 |
| 关于枚举类，说法错误的是**(百乐润信息技术深圳有限公司)**   1. 可以通过[]访问枚举成员 2. 可以通过 . 访问枚举成员 3. 枚举类的值成员的value属性，默认从0开始赋值 4. 可以用@unique保证没有重复定义的枚举 |
| 为python提供了科学计算和图像处理的常用第三方库是**(百乐润信息技术深圳有限公司)**   1. Pandas，opencv 2. Scipy，sk-learn 3. Scipy,opencv 4. Pandas,sk-learn |
| 一共五条赛道，25个人赛跑，只记录名次不记录时间，最少几次比赛可以确定他们的第三名（简要说明计算方法或思路）**(百乐润信息技术深圳有限公司)** |
| 随机选一个碗拿出一颗糖，发现是水果糖，那么这颗糖来自碗#1的概率是多少？（简要说明计算方法或思路）**(百乐润信息技术深圳有限公司)** |
| 0x1234567以BigEndian字节序在内存中的存储结果是(帕斯亚软件测试题)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| (帕斯亚软件测试题)  #include <stdio.h>  void main(void) {  int a[5] = {1, 2, 3, 4, 5};  int\* ptr = (int\*)(&a+1);  printf("%d, %d", \*(a+1), \*(ptr-1));  return;  }  输出结果是\_\_\_\_\_\_2, 5\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| 观察如下代码  #include <iostream>  using namespace std;  class A {  inline virtual void print() {  cout << "A" << endl;  }  public:  A() {print();}  ~A() {print();}  };  class B: public A {  inline void print() { cout << "B" << endl;}  public:  B() {print();}  virtual ~B() {print();}  };  class C: public B {  inline void print() {cout << "C" << endl;}  public:  C() {print();}  ~C() {print();}  };  int main() {  A\* a = new C();  B\* b = new C();  C\* c = new C();  cout << "-----------------" << endl;  delete a;  delete b;  delete c;  return 0;  }  输出结果是  A  B  C  A  B  C  A  B  C  -----------------  A  C  B  A  C  B  A |
| 写出float x 与 “零值” 比较的if语句？ |
| **import** copy  a = [1, 2, 3, 4, [**'a'**, **'b'**]] *# 原始对象*  b = a *# 赋值，传对象的引用*  c = copy.copy(a) *# 对象拷贝，浅拷贝*  d = copy.deepcopy(a) *# 对象拷贝，深拷贝*  a.append(5) *# 修改对象a*  a[4].append(**'c'**) *# 修改对象a中的['a', 'b']数组对象*  print(**'a ='**, a)  print(**'b ='**, b)  print(**'c ='**, c)  print(**'d ='**, d)  a = [1, 2, 3, 4, ['a', 'b', 'c'], 5]  b = [1, 2, 3, 4, ['a', 'b', 'c'], 5]  c = [1, 2, 3, 4, ['a', 'b', 'c']]  d = [1, 2, 3, 4, ['a', 'b']] |