<1> python 语法

请说一下你对迭代器和生成器的区别？

答:(简单理解)
迭代器和生成器都是 Python 中特有的概念，迭代器可以看作是一个特殊的对象，每次调用该对象时会返回自身的下一个元素，从实现上来看，一个可迭代的对象必须是定义了__iter__()方法的对象，而一个迭代器必须是定义了__iter__()方法和 next()方法的对象。生成器的概念要比迭代器稍显复杂，因为生成器是能够返回一个迭代器的函数，其最大的作用是将输入对象返回为一个迭代器。Python 中使用了迭代的概念，是因为当需要循环遍历一个较大的对象时，传统的内存载入方式会消耗大量的内存，不如需要时读取一个元素的方式更为经济快捷。
（1）迭代器是一个抽象的概念，任何对象，如果它的类有 next 方法和 iter 方法返回自己本身都可以认为他是迭代器。对于 string、list、dict、tuple 等这类容器对象，使用 for 循环遍历是很方便的。在 for 语句对容器对象调用 iter()函数，(iter()是 python 的内置函数),iter()会返回一个定义了 next()方法的迭代器对象，它在容器中逐个访问容器内元素(next()也是 python 的内置函数),在没有后续元素时，next()会抛出一个 StopIteration 异常
（2）生成器（Generator）是创建迭代器的简单而强大的工具。它写起来就像是正规的函数，只是在需要返回数据的时候使用 yield 语句。每次 next()被调用时，生成器都会返回它脱离的位置（它记忆语句最后一次执行的位置和所有的数据值）．简单的说就是在函数的执行过程中，yield 语句会把你需要的值返回给调用生成器的地方，然后退出函数，下一次调用生成器函数的时候又从上次中断的地方开始执行，而生成器内的所有变量参数都会被保存下来供下一次使用。

区别: 生成器能做到迭代器能做的所有事,而且因为自动创建了__iter__()和 next()方法,生成器显得特别简洁,而且生成器也是高效的，使用生成器表达式取代列表解析可以同时节省内存。除了创建和保存程序状态的自动方法,当发生器终结时,还会自动抛出 StopIteration 异常

什么是线程安全？

答:
如果你的代码所在的进程中有多个线程在同时运行，而这些线程可能会同时运行这段代码。如果每次运行结果和单线程运行的结果是一样的，而且其他的变量的值也和预期的是一样的，就是线程安全的。
但有的时候多线程运行时,会同时竞争同一个共享资源并对他进行修改,这个时候由于线程之间竞争的问题就会造成修改错误,会产生多线程运行的结果和单线程的运行结果不一致的问题.

解决方法:
当多个线程几乎同时修改某一个共享数据的时候, 需要进行同步控制 线程同步能够保证多个线程安全访问竞争资源，最简单的同步机制是引入互斥锁。 互斥锁为资源引入一个状态：锁定/非锁定。 某个线程要更改共享数据时，先将其锁定，此时资源的状态为“锁定”，其他线程不能更改；直到该线程释放资源，将资源的状态变成“非锁定”，其他的线程才能再次锁定该资源。互斥锁保证了每次只有一个线程进行写入操作,

从而保证了多线程情况下数据的正确性。

socket 的 connect 方法工作原理是什么?

答:
网络编程 socket api 存在一批核心接口，而这一批核心接口就是几个看似简单的函数，尽管实际上这些函数没有一个是简单。connect 函数就是这些核心接口的一个函数，它完成主动连接的过程。connect 函数的功能是完成一个有连接协议的连接过程，对于 TCP 来说就是那个三路握手过程.它的函数原型：connect(address) 连接到 address 处的套接字。一般 address 的格式为元组（hostname,port），如果连接出错，返回 socket.error 错误。
connect 函数的功能:
connect 函数的功能可以用一句话来概括，就是完成面向连接的协议的连接过程，它是主要用于连接的。面向连接的协议如 TCP，在建立连接的时候总会有一方先发送数据，那么谁调用了 connect 谁就是先发送数据的一方。如此理解 connect 的参数就容易了，我必需指定数据发送的地址，这正好是 connect 的参数的作用.
参数元组（hostname,port）
指定数据发送的目的地，也就是服务器端的地址。这里服务器是针对 connect 说的，因为 connect 是主动连接的一方调用的，所以相应的要存在一个被连接的一方，被动连接的一方需要调用 listen 以接受 connect 的连接请求，如此被动连接的一方就是服务器了。与所有的 socket 网络接口一样，connect 总会在某个时候可能失败，此时它会返 socket.error,常见的错误有对方主机不可达或者超时错误，也可以是对方主机没有相应的进程在对应端口等待。

Python 中怎么简单的实现列表去重？

答:Python 中的列表去重方法有很多,下面有两种常用的简单方式
方法一:转换为集合数据类型, set(列表)

```
li = [1, 2, 3, 4, 1, 2]
s = set(li)
li = list(s)
print li
#[1, 2, 3, 4]
```

解释:列表中的元素可以重复,在集合中元素是不可以重复的,这是集合的特性.利用这个特性,我们可以把列表强制转化成集合,使得原来列表中重复的元素去重.但是注意这种方法有可能使得原有的列表中的数据顺序发生改变.

方法二:不改变原有列表顺序的方法

```
list2 = []
list1 = [1,2,3,2,2,2,4,6,5]
for i in list1:
    if i not in list2:
        list2.append(i)
list2
[1, 2, 3, 4, 6, 5]
```

解释:通过创建一个新的列表 list2 存贮去重后的数据,如以上代码显示,对 list1 中的数据依次进行 for 循环遍历.如果 list1 中的数据在 list2 中没有则添加到 list2 中,如果 list2 中有该数据则不添加,如此直到对 list1 中数据遍历完成就可以使得 list2 中存储保持原有顺序且不重复的数据.

python 中 yield 的用法?

答:
通常的 for...in...循环中，in 后面是一个数组，这个数组就是一个可迭代对象，类似的还有列表，字符串，文件。它可以是 mylist = [1, 2, 3]，也可以是 mylist = [x*x for x in range(3)]。 它的缺陷是所有数据都在内存中，如果有海量数据的话将会非常耗内存。
而生成器就不是,他不需要把所有的数据都存放在内层中,而是当使用到的时候再去获取.这是由于生成器是可以迭代的，生成器(generator)能够迭代的关键是它有一个 next()方法，工作原理就是通过重复调用 next()方法，不断的获取下一数据,而不是一次性把所有的数据都存放到内存中,这样重复操作直到捕获一个异常才会停止。比如 mygenerator = (x*x for x in range(3))，注意这里用到了() 这里相当于一个函数，它就不是数组，而上面的例子都是 [] 列表。
而带有 yield 的函数就跟我们举的例子 mygenerator = (x*x for x in range(3))一样,不再是一个普通函数，而是一个生成器 generator，可用于迭代，工作原理同上。 yield 是一个类似 return 的关键字，迭代一次遇到 yield 时就返回 yield 后面的值。重点是：下一次迭代时，从上一次迭代遇到的 yield 后面的代码开始执行。
简要理解：yield 就是 return 返回一个值，并且记住这个返回的位置，下次迭代就从这个位置后开始。

什么是面向对象编程?

面向对象的编程产生的原因：由于面向过程编程在构造系统时，无法解决重用，维护，扩展的问题，而且逻辑过于复杂，代码晦涩难懂，因此，人们开始想能不能让计算机直接模拟现实的环境，以人类解决问题的方法，思路，习惯和步骤来设计相应的应用程序。于是，面向对象的编程思想就产生了。
面向对象的编程的主要思想是把构成问题的各个事物分解成各个对象，建立对象的目的不是为了完成一个步骤，而是为了描述一个事物在解决问题的过程中经历的步骤和行为。对象作为程序的基本单位，将程序和数据封装其中，以提高程序的重用性，灵活性和可扩展性。
类是创建对象的模板，一个类可以创建多个对象。对象是类的实例化。 类是 抽象的，不占用存储空间,而对象是具体的，占用存储空间。 面向对象有三大特性：封装，继承，多态。

谈谈你对 GIL 锁对 python 多线程的影响？

答:

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。每个 CPU 在同一时间只能执行一个线程（在单核 CPU 下的多线程其实都只是并发，不是并行，并发和并行从宏观上来讲都是同时处理多路请求的概念。但并发和并行又有区别，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内发生。）

在 Python 多线程下，每个线程的执行方式：

1、获取 GIL

2、执行代码直到 sleep 或者是 python 虚拟机将其挂起。

3、释放 GIL

可见，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 锁存在下多线程的使用场景:

1.密集型科学运算(CPU 没有闲置)

由于 GIL 锁保证同一时刻只有一个线程可以使用 CPU,这使得多线程之间会对 GIL 锁进行争夺,争夺的过程中谁获取了 GIL 锁谁就可以使用 CPU,这种争夺状态会一直持续,这就造成了资源的浪费,浪费在了抢夺资源上,在这种情况下我们会发现如果只是用一个线程反而会避免这种资源的竞争,同时提高 CPU 的使用率,所以在科学计算这种没有 CPU 闲置的情况下我们倾向于使用单线程.

2.I/O 操作(有 CPU 闲置)

像爬虫,网络请求这类操作往往受到网速等原因的限制,会造成一个线程占用 CPU 的同时等待网络数据的获取,这个时候 GIL 锁会自动判定如果线程使用的 CPU 闲置他就会自动切换到另外一个线程,由此可以节省线程占用 CPU 阻塞等待网络请求的时间.所以在像网络操作这种有 CPU 闲置的情况下我们倾向使用多线程

python 是如何进行内存管理的？

答:(基本理解:足够面试使用)

一、垃圾回收：python 不像 C++，Java 等语言一样，他们可以不用事先声明变量类型而直接对变量进行赋值。对 Python 语言来讲，对象的类型和内存都是在运行时确定的。这也是为什么我们称 Python 语言为动态类型的原因（这里我们把动态类型可以简单的归结为对变量内存地址的分配是在运行时自动判断变量类型并对变量进行赋值）。

二、引用计数：Python 采用了类似 Windows 内核对象一样的方式来对内存进行管理。每一个对象，都维护这一个对指向该对对象的引用的计数。当变量被绑定在一个对象上的时候，该变量的引用计数就是 1，(还有另外一些情况也会导致变量引用计数的增加),系统会自动维护这些标签，并定时扫描，当某标签的引用计数变为 0 的时候，该对就会被回收。

(深入理解:加深对内存机制的理解)

三、内存池机制 Python 的内存机制如同金字塔一般，

第 0 层是 C 语言中的 malloc(c 语言中申请内存空间的方法)，free(C 语言中释放内存的方法)等内存分配和释放函数进行操作；

第 1 层和第 2 层是内存池，有 Python 的接口函数 PyMem_Malloc 函数实现，当对象小于 256K 时有该层直接分配内存；

第 3 层是最上层，也就是我们对 Python 对象的直接操作；

在 C 中如果频繁的调用 malloc 与 free 时,是会产生性能问题的.再加上频繁的分配与释放小块的内存会产生内存碎片. Python 在这里主要干的工作有:

如果请求分配的内存在 1~256 字节之间就使用自己的内存管理系统,否则直接使用 malloc.

这里还是会调用 malloc 分配内存,但每次会分配一块大小为 256k 的大块内存.

经由内存池登记的内存到最后还是会回收到内存池,并不会调用 C 的 free 释放掉.以便下次使用.对于简单的 Python 对象，例如数值、字符串，元组（tuple 不允许被更改）采用的是复制的方式，也就是说当将另一个变量 B 赋值给变量 A 时，虽然 A 和 B 的内存空间仍然相同，但当 A 的值发生变化时，会重新给 A 分配空间，A 和 B 的地址变得不再相同

请描述线程、进程、协程的优缺点

进程、线程和协程是三个在多任务处理中常听到的概念，三者各有区别又相互联系。

(1)进程

进程是一个程序在一个数据集中的一次动态执行过程，可以简单理解为“正在执行的程序”，它是 CPU 资源分配和调度的独立单位。 进程一般由程序、数据集、进程控制块三部分组成。我们编写的程序用来描述进程要完成哪些功能以及如何完成；数据集则是程序在执行过程中所需要使用的资源；进程控制块用来记录进程的外部特征，描述进程的执行变化过程，系统可以利用它来控制和管理进程，它是系统感知进程存在的唯一标志。

进程的缺点： 创建、撤销和切换的开销比较大。

(2)线程

线程是在进程之后发展出来的概念。 线程也叫轻量级进程，它是一个基本的 CPU 执行单元，也是程序执行过程中的最小单元，由线程 ID、程序计数器、寄存器集合和堆栈共同组成。一个进程可以包含多个线程。

线程的优点： 减小了程序并发执行时的开销，提高了操作系统的并发性能.

缺点的缺点： 线程没有自己的系统资源，只拥有在运行时必不可少的资源，但同一进程的各线程可以共享进程所拥有的系统资源，如果把进程比作一个车间，那么线程就好比是车间里面的工人。不过对于某些独占性资源存在锁机制，处理不当可能会产生“死锁”。

(3)协程

协程是一种用户态的轻量级线程，又称微线程，英文名 Coroutine，协程的调度完全由用户控制。人们通常将协程和子程序（函数）比较着理解。 子程序调用总是一个入口，一次返回，一旦退出即完成了子程序的执行。 协程的起始处是第一个入口点，在协程里，返回点之后是接下来的入口点。在 python 中，协程可以通过 yield 来调用其它协程。通过 yield 方式转移执行权的协程之间不是调用者与被调用者的关系，而是彼此对称、平等的，通过相互协作共同完成任务。其运行的大致流程如下：

第一步，协程 A 开始执行。

第二步，协程 A 执行到一半，进入暂停，通过 yield 命令将执行权转移到协程 B。

第三步，（一段时间后）协程 B 交还执行权。 第四步，协程 A 恢复执行。 1 2 3 4 协程的特点在于是一个线程执行，

与多线程相比，其优势体现在： 协程的执行效率非常高。因为子程序切换不是线程切换，而是由程序自身控制，因此，没有线程切换的开销，和多线程比，线程数量越多，协程的性能优势就越明显。 协程不需要多线程的锁机制。在协程中控制共享资源不加锁，只需要判断状态就好了。

提示:利用多核 CPU 最简单的方法是多进程+协程，既充分利用多核，又充分发挥协程的高效率，可获得极高的性能.

深拷贝和浅拷贝的区别

答: 通俗的说 浅拷贝只拷贝引用(也就是地址), 深拷贝就是拷贝具体的值, 重新开辟新的空间存储这些值。就像是浅拷贝就是你的影子,深拷贝是你的克隆人, 你没了影子也就没了,但是克隆人还活着。
深拷贝和浅拷贝有很多不同的情况,不同情况下都有不同的特点
1 拷贝单一的可变数据类型的时候 如 a = [1,2] 这种单一的列表
深拷贝和浅拷贝都是一样的,都会开辟新的一片新空间,存储数据,保证数据的独立性
2 拷贝复杂的有嵌套数据类型 如 a = [1,2],b = [3,4],c = [a,b]拷贝 c 的时候
d = copy.copy(c)
浅拷贝:只会给 d 开辟一片空间 存贮 c 中的数据,也就是存储 a,b 的引用, 而并不是开辟多个空间去存储 a 中 1,2 数据,b 中的 3,4 数据,这样以来 a 如果发生变化 d 也会跟着发生变化,没有办法保证数据的独立性.
d = copy.deepcopy(c)
深拷贝:会给 d 开辟多片空间,存储所有的数据,可以保证独立空间,保证数据的独立性
3 拷贝不可变数据类型 如元组这样的类型
深拷贝和浅拷贝都一样,因为不可变类型数据不会变化,所以不需要保证数据的独立性,直接引用这个数据就可以,而不会开辟新的空间.

提示:ptyhon 中大多情况都是浅拷贝

<2>Linux 基础和数据结构与算法

10 个常用的 Linux 命令？

a) 答案：略

find 和 grep 的区别？

grep 命令是一种强大的文本搜索工具，grep 搜索内容串可以是正则表达式，允许对文本文件进行模式查找。如果找到匹配模式， grep 打印包含模式的所有行。

find 通常用来在特定的目录下搜索符合条件的文件，也可以用来搜索特定用户属主的文件。

什么是阻塞？什么是非阻塞？

阻塞调用是指调用结果返回之前，当前线程会被挂起。函数只有在得到结果之后才会返回。有人也许会把阻塞调用和同步调用等同起来，实际上他是不同的。对于同步调用来说，很多时候当前线程还是激活的，只是从逻辑上当前函数没有返回而已。例如，我们在 CSocket 中调用 Receive 函数，如果缓冲区中没有数据，这个函数就会一直等待，直到有数据才返回。而此时，当前线程还会继续处理各种各样的消息。如果主窗口和调用函数在同一个线程中，除非你在特殊的界面操作函数中调用，其实主界面还是应该可以刷新。socket 接收数据的另外一个函数 recv 则是一个阻塞调用的例子。当 socket 工作在阻塞模式的时候，如果没有数据的情况下调用该函数，则当前线程就会被挂起，直到有数据为止。

非阻塞和阻塞的概念相对应，指在不能立刻得到结果之前，该函数不会阻塞当前线程，而会立刻返回。

描述数组、链表、队列、堆栈的区别？

数组与链表是数据存储方式的概念，数组在连续的空间中存储数据，而链表可以在非连续的空间中存储数据；

队列和堆栈是描述数据存取方式的概念，队列是先进先出，而堆栈是后进先出；队列和堆栈可以用数组来实现，也可以用链表实现。

你知道几种排序,讲一讲你最熟悉的一种?

排序方法	平均情况	最好情况	最坏情况	辅助空间	稳定性
冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	O(1)	稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	不稳定
插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	O(1)	稳定
希尔排序	$O(n \log n)$~ $O(n^2)$	$O(n^{1.3})$	$O(n^2)$	O(1)	不稳定
堆排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	O(1)	不稳定
归并排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	稳定
快速排序	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$~ $O(n)$	不稳定

如果有一个文本 temp.txt 中有且只有一个 1.8，此时我们想将其更换为 2.0，请在不适用编辑器的前提下实现该功能

```
sed –i s#1.8#2.0# temp.txt
```

如何使用两个栈结构实现一个队列

将数据压入一个栈，然后将其分别弹栈，并将弹栈数据压入另一个栈，然后另一个栈出栈

如何检查最近十分钟内变动过的文件

```
find / -mmin -10
```

在当前目录下查找大小小于 10m 的文件

```
find / -size -10M
```

已知二叉树的先序:0 1 3 7 8 4 2 5 6,中序:7 3 8 1 9 4 0 5 2 6，求后序遍历顺序 7 8 3 9 4 1 5 6 2 0