Python

1. 可变对象与不可变对象 该对象内存中的值可否被改变 不可变：tuple string int float bool 可变： list dict set
2. tuple, list, dict区别 list 可以通过pop删除指定位置的元素，insert 到指定位置。
   tuple 指向的位置不可变，但是如果指向的不是基本类型，其内部元素可以被该变。
   Dict get 方法更加保险， dict的 key必须为不可变对象.
   set和dict类似，区别是不可重复
   dict.keys() 和 dict的区别就是keys() 会把key都转换位list, 而后者则是hash. 后者会快。
   Dict对key进行hash,再进行一次hash查找， 速度略微比set慢,
   dict.items() 以返还可遍历的键值元组.
   列表表达式 在for循环后面的if是一个筛选条件，for前面if必须有else,必须计算出一个结果. for循环可以两层生成全排列， 不可像map一样并行计算讲个list [x + y for x in 'abc' for y in 'xyz'] → [ax ay az bx by bz cx cy cz]
   字典合并，放法1 dict1.update(dict2), 方法2 dict3 = {**dict1,**dict2} 后者更快，出现相同值均为后者覆盖前者.
   字典生成式 {k ：v for k,v in dict1.items() if v > 15}
   append 和 extend 的区别，extend是在列表后面拼接列表，维度一致，append则是在list后面拼接元素[1,2,[3,4]]
3. iterables, generator,yield的区别
   iterable 为可迭代对象，list,tuple,dict, str 等，只要可以返还一个迭代器__iter__方法
   generator 把列表生成式的[]改为()为第一种方法，第二种方法是yield. 惰性加载可以节省内存，用到时再计算
4. python 装饰器

map() 函数可以接受一个函数和一个list，并把函数作用于list上，同时也可以接受多个list一并计算

zip()函数接受一系列可迭代对象做参数，不同对象中的相应元素打包为一个元素，如果长度不等取最短

reduce()函数 与map类似，不同是做累计运算，从头至尾,且reduce只能接受一个list, lambda函数有两个变量

lambda表达式：冒号前为参数，冒号后为运算式 lambda x,y:(x**y,x+y)运算结果为返还值

不定参（*args，**kw）含义 *args->可变参数，传入函数为一个tuple，如果直接传入一个list,则是一整个list传入，如果是*list，则是按照顺序传入 **kw 关键字参数，带名字的参数集合，为dict格式，传入用**dict

修饰符， 不改变原函数功能从而增加功能，因为wrapper函数会覆写原函数的名字，所以有依赖函数签名的代码执行就会出错 需要@functools.wraps(func)，有返还值的函数需要在wrapper里面返还，如果需要计时等在执行函数后做的工作，则可以先 a = func() ->dosomething -> return a

5. 深拷贝和浅拷贝 潜拷贝只拷贝内存，修改不可变对象需要开辟新空间，修改可变对象需要空间，深拷贝完全拷贝一个副本，完全不一样
6. 内存回收机制 引用计数法 任何对象核心就是PyObject， 内部有引用计数器（ob_redcnt） 对象被创建，引用，作为参数，作为元素储存会让计数器+1，对象别名被删除，赋予新对象，离开作用域，容器被销毁或者从容器中剔除 标记清除 （追踪回收）class person()... class dog()... p = person() d = dog() p.pet =dog dog.master = p del p,dog 策略：遍历所有对象，如果对象可达，则标记可达，再循环，如果没有被标记为可达，则删除

分代回收 总共有三代 0 - 1 - 2， 新创建的为0， 在第一次回收幸存对象会被移到老一代中去，如果老一代内存回收机制触发则新一代也会触发

7. 线程和进程

进程是资源分配最小的单位，线程是CPU调度最小的单位， 一个进程可以包含多个线程，进程之间数据共享较为困难，进程可以拓展到多机器，进程消耗更多的资源 线程是进程内部的子任务，每个进程至少有一个线程。

进程的优点：封闭性和可再现性，缺点是切换进程消耗比线程要耗费，进程之间无法共享内存，通讯难度大。线程的缺点 频繁调度会造成资源浪费

多进程的优点 每个进程互相独立，不影响主程序的稳定性，缺点是多进程调度开销大，逻辑控制复杂。

多线程的优点 逻辑控制方法简单，共享内存和变量，缺点是与主程序公用内存，单个线程的崩溃会影响主程序的稳定性 python 多进程适合在CPU密集型操作，多线程适合IO密集，线程是并发，进程是并行

Python 的 GIL 是全局解释器锁，导致一个进程永远只能执行一个线程 执行线程的方式 → 获取 GIL， 执行代码到sleep或者虚拟机将其挂起，释放GIL

python Thread类有两个模块 _thread低级模块和threading高级模块, 线程的5个生命周期，新建，就绪，运行，阻塞，死亡， 启动线程用start方法而不是run方法，只能对新建线程用start方法 线程在如下情况会被阻塞 线程调用sleep方法，调用了I/O 试图获得锁对象，等待通知Notify 其他程序调用了join()方法后，调用线程将被阻塞直到被调用的执行完毕 后台线程 deamon thread 称为守护线程，后台线程的特征是前台线程死亡后，后台线程会自动死亡， 线程可以睡眠(sleep)

线程同步 两个线程同时对一个对象进行操作的时候需要锁，lock 和 Rlock， lock和Rlock的区别是可否对一个对象重复上锁，lock需要等锁释放再次上锁，Rlock则需要锁与释放数目匹配 Rlockl支持在同一线程中多次请求同一资源，直到所有线程被release，其他线程才可获得资源 Rlock 使用场景 一个线程用带锁的方法，该方法又调用了另外一个带锁的方法，即可直接调用，而无需重新获得锁

用上下文管理器加锁，with lock：--> 更加安全

死锁：若干子线程等待对方某部分资源解除占用-> A,B均需要两个锁才能运行，若A持有a,B持有b，A等待B释放，B等待A释放 死锁通常在同一线程内，嵌套获取同一个锁，多个线程，不按顺序获取多个锁

死锁解决方案 ：设置锁的超时时间，避免多次锁定，锁排序: 用上下文管理器，强制用升序来往每个线程获取锁，为每个锁分配一个唯一的ID (内存) 银行家算法

死锁的检测,恢复，防止：https://zhuanlan.zhihu.com/p/61221667

https://blog.csdn.net/Mikeoperfect/article/details/78574887

上下文管理器 确保一些系统资源可以被正确的占用和释放 方法 __exit()__ __enter()__ 一开始加载exit()备用，执行上下文管理器的__enter()__方法 将enter方法的返还值加载到as对象中 执行with 内的代码块 执行_exit()_ 实现方法：用修饰符 from contextlib import context manager @context manager

condition 线程通信 wait() 线程挂起，收到notify唤起，notify_all唤起所有等待线程,其余功能与锁类似

queue 提供三种阻塞队列，FIFO， LIFO，priorityqueue 分几种操作：queue.put 向队列中添加元素，如果队列已满则队列阻塞，queue.get 获取元素，如果队列为空则阻塞

event 简单的线程通信机制，一个线程发出event,其他线程可以被event激发， set() 唤醒所有处于等待中的线程，wait()阻塞当前线程,与condition的区别是没有锁
线程池 启动新的线程成本高昂，线程池可以提升性能，尤其是需要大量生命周期短的线程的时候，线程池还可以控制系统中总线程的数量避免崩溃 使用线程池、进程池把当前任务提交给线程池计算即可, done()会返还是否完成，result()则会返还结果，不过会阻塞当前线程 解决方案是add_done_callback() 代替result()
wait()和sleep的区别 wait释放锁，直到被唤醒，sleep 则不释放锁
yield暂停当前正在执行的线程对象，并将其重新变为可运行状态，以允许具有相同优先级的其他线程获得运行机会
同步，异步 同步方法调用一旦开始，调用者必须等到方法返回后，才能继续后续的行为， 异步方法更像是一个消息传递，一旦开始，方法调用就会立即返回，调用者就可以继续后续的操作
并发，并行 **并行性**是指两个或多个事件在同一时刻发生。而**并发性**是指连个或多个事件在同一时间间隔内发生。
死锁：概念：死锁是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去。例子：有a，b两个锁，A，B两个线程，均需要获取a，b两个锁。若A持有a，B持有b，则A，B会无限等待。
饥饿：概念：某一个或者多个线程因为种种原因无法获得所需要的资源，导致一直无法执行例子： 1它的线程优先级可能太低2某一个线程一直占着关键资源不放，导致其他需要这个资源的线程无法正常执行，但起码可能有资源终于被释放，自己执行的一天。不会彻底死。
活锁：概念：如果线程的智力不够，且都秉承着 “谦让” 的原则，主动将资源释放给他人使用，那么就会出现资源不断在两个线程中跳动，而没有一个线程可以同时拿到所有的资源而正常执行
多进程：multiprocessing.process创建新进程，pid返还进程的ID，ppid返还父进程id
进程之间两种通信方式， queue和pipe，queue与进程的queue类似，pipe返还两个端口，接到两个进程之间进行通信 进程池 减少启动进程的资源
协程： 称微线程，效率高，没有线程切换的开销，不需要锁机制，python 通过generator实现协程
asyncio 异步库，消息循环，eventloop， 执行的协程添加进eventloop中即可
锁的优化：
减少锁的持有时间，减少锁的粒度，将大对象(经常被访问到的对象)拆分成小对象，降低锁竞争，权重低的锁才有机会
锁分离，读写锁分为读锁和写锁，锁粗化，用完需要锁的部分立即释放

8.面向对象编程：

1. **封装、继承、多态**
    a. **封装**：
    b. 封装是指将对象的实现细节隐藏起来，然后通过公共的方法来向外暴露出该对象的功能，使用封装不仅仅安全，更可以简化操作
    c. **继承**：
    d. 继承是面向对象实现软件复用的重要手段，当子类继承父类后，子类是一种特殊的父类，能直接或间接获得父类里的成员
    e. **多态**：
    f. 定义：多态简而言之就是同一个行为具有多个不同表现形式或形态的能力
    g. 多态的条件：1）继承；2）重写；3）向上转型

h. 用处：可以屏蔽不同子类对象之间的实现差异，从而写出通用的代码达到通用编程
i. 参考：https://zhuanlan.zhihu.com/p/66736303

2. **面向过程 vs 面向对象**
   a. 看到知乎上有一句有意思的话：
   b. 你的程序要完成一个任务，相当于讲一个故事。
   c. 面向过程：编年体；面向对象：纪传体。
   d. 而对于复杂的程序/宏大的故事，事实都证明了，面向对象/纪传是更合理的表述方法。
3. **重写 vs 重载**
   a. **重写（Override）**是指子类对父类方法的一种重写
   b. **重载（Overload）**表示同一个类中可以有多个名称相同的方法，但这些方法的参数列表各不相同
   c. 问：函数的返回值不同可以构成重载吗？为什么？
   d. 答：不可以，因为 Java 中调用函数并不需要强制赋值，如直接调用f（），此时若有两个函数public boolean f（）和public int f（），编译器无法判断调用的哪个

数据结构与算法
100个香蕉 猴子搬运，50<100-3x /100-3x>50,
64匹马8条跑到，最快的4匹：https://blog.csdn.net/jodie123456/article/details/101264113

链表反转，链表两两反转，链表k个元素反转，合并两个有序的链表，删除链表中倒数第n个元素(一遍循环) 快慢指针相差N和步长，寻找链表中是否有环(快慢指针)，寻找链表中环的位置(d = len+x, 2d = len+r+x --> len = r-x), 寻找链表中点(快慢指针，快的速度是慢节点两倍),判断链表是否相交(无环)(判断两个链表的末尾是否相等，两个相交链表焦点后的元素共享)判断两个节点是否相交(有环)找到两个链表的环内节点(判断是否为环快慢节点相交处)，判断其中一个内环节点在不在第二个链表上, 查找两个链表的公共节点(两个指针遍历一遍后第二个指针等于第一个链表头，第一个指针等于第二个链表头)，两个栈实现队列（进队第一个栈压入，出队第二个栈弹出，如果第二个栈为空，pop出第一个栈的所有元素), LRU cache,一致性hash, 链表排序(递归与非递归)

二分法查找,第一个比target大/最后一个比target小的元素, 寻找数组中第一个/最后一个target, 寻找插入位置，反转数组的二分法查找(判断是左还是右有序)，第k小的两数之差,寻找数组中的峰值数组

二叉树/红黑树: 前序遍历，中序遍历，后序遍历 （三种递归和非递归方法）层次遍历，之字形打印，最近公共祖先，反转二叉树(递归与队列)，查找第k大的元素(反向中序遍历(右中左))，第K层的节点树，树的结构是否相同，返还和为某一值的路径，返还所有路径, 返还中序遍历的下一个节点(有父节点)，序列化二叉树,判断二叉树是否对称, 寻找二叉树的最长路径(深度遍历)

图: dinic算法，二分图最大流匹配(二分图中所有边为1，所有起止点的边为1),课程表(拓扑排序), 寻找通路(广度遍历)， 网络延迟：dijkstra’s algorithm 堆优化

哈希相关：变位词组，最佳直线,稀疏相似度(dic1[num] 记录含有num的所有doc的id,对每一个id数组，两两元素记录到dic2[i, j]，即为docs[i]与docs[j]的交集大小)

并查集：二维数组记录好友关系，一维数组记录并查集

动态规划: 一次编辑，抢劫问题 dp[i+1] = max(dp[i],dp[i-1]+num), 堆箱子(寻找能支撑第I号箱子的K号箱子),有效快递序列数目(有序插入 2i-1，无序插入 (2i-1)*(i-1)), 正则表达式匹配，背包问题：f[i,j] = max(f[i-1,j-Wi]+Pi,f[i-1,j]) f[i,j]表示在前i件物品中选择若干件放在承重为 j 的背包中，可以取得的最大价值 Pi表示第i件物品的价值。决策：为了背包中物品总价值最大化，第 i件物品应该放入背包中?, 斐波那契数，完全平方数

数组：3sum，最长单调递增数列，锯齿状数组的最小操作次数,子数组最大平均数,分割数组(左边所有元素小于右边)，缺失的最小正整数(标记负数)，0矩阵，是否存在环形数组(快慢指针加剪枝)，匹配子序列的单词数，和为k的n数之和(回溯加剪枝)，和为K 的最少斐波那契数(贪心)，统计全部为1的正方形子矩阵，找到所有数组中消失的数字(标记负数)，快照数组(哈希),数组中第K大的数（快速选择），接水问题,矩阵最小路径和，三角形最小路径和(空间优化自顶向下)
数对和(数组排序，双指针),两次买卖股票(动态规划), 数组的全排列（回溯加剪枝，如果前一位相同的数字还未使用，则本位提前停止), 两个数组最长的公共数组（动态规划，或者哈希，求数组中每个长度为len的哈希值进行匹配, 数组中最长的连续数列,数组中最大的乘积(动态规划)，矩阵中最大的矩形(动态规划)，判断是否互为字符串重写(位运算)，三个数最大乘积(最大三个和最小俩个数),数组中重复的数(负数标记),计算岛屿的最大面积(深度优先(递归，栈),广度优先,标记每个访问过的格子，每个格子只访问一次)，二维网格搜索单词(与搜索岛屿类似，每个格子标记先标记一次，查找失败释放标记，每个格子允许被二次访问),找出数组主要元素(占据过半，摩尔投票法加二部验证),T9键盘 哈希，字母与数字(标记数组，查找最长相同数组)

字符串相关：压缩字符串，回复空格(动态规划), 计算器(栈，如果+-，下一个数进栈，*/ 出栈计算, 移除无效括号(栈),最长单词(动态规划)，最长回文子串(动态规划 p(i,j) = p(i+1,j-1)), 带括号的计算器(后缀树)

快速平方：y = x^n/2 n为偶数，x^n = y^2 n为奇数 x^n = y^2*x , 牛顿法开根号，等概率随机选取

排序 归并排序(递归与否)，冒泡排序及其优化,快排及其优化

堆(heap) 二叉树和数组两种表示，数组就是对堆的每一层进行全排列,堆是完全二叉树，插入插入到最后一个位置，并且与父节点比较，如果大于父节点则交换，直到小于父节点。删除只能删除根节点，并拿最后一个元素替换根节点，与子节点中较小的，并且小于本节点的元素进行交换

红黑树 根节点为黑，叶子节点为黑，红色节点的两个子节点为黑色，任意一个节点到每个叶子节点都会经过相同数量的黑节点

左旋影响右子树结构，右旋影响左子树结构

- https://www.jianshu.com/p/e136ec79235c
- https://juejin.im/entry/6844903454767317005

B，B+树:

- https://juejin.im/entry/6844903613915987975
- https://blog.csdn.net/login_sonata/article/details/75268075
- https://www.cnblogs.com/nullzx/p/8729425.html

计算机网络

网络的协议层数模型：

OSI七层网络模型	Linux TCP/IP四层概念模型	对应网络协议
应用层（Application）	应用层	TFTP, FTP, NFS, WAIS
表示层（Presentation）		Telnet, Rlogin, SNMP, Gopher
会话层（Session）		SMTP, DNS
传输层（Transport）	传输层	TCP, UDP
网络层（Network）	网络层	IP, ICMP, ARP, RARP, AKP, UUCP
数据链路层（Data Link）	网络接口	FDDI, Ethernet, Arpanet, PDN, SLIP, PPP
物理层（Physical）		IEEE 802.1A, IEEE 802.2到IEEE 802.11

Http 状态码：

a. 1XX信息提示
   i. 100继续
   ii. 101切换协议
b. 2XX表示成功
   i. 200：确定。客户端请求已成功
c. 3XX表示重定向
   i. 300多种选择，也即是针对这个请求服务器可以做出多种操作
d. 4XX表示请求出错
   i. 400表示服务器不理解语法
   ii. 404表示服务器找不到请求页面
e. 5XX表示服务器处理请求出错
   i. 500表示服务器内部出错

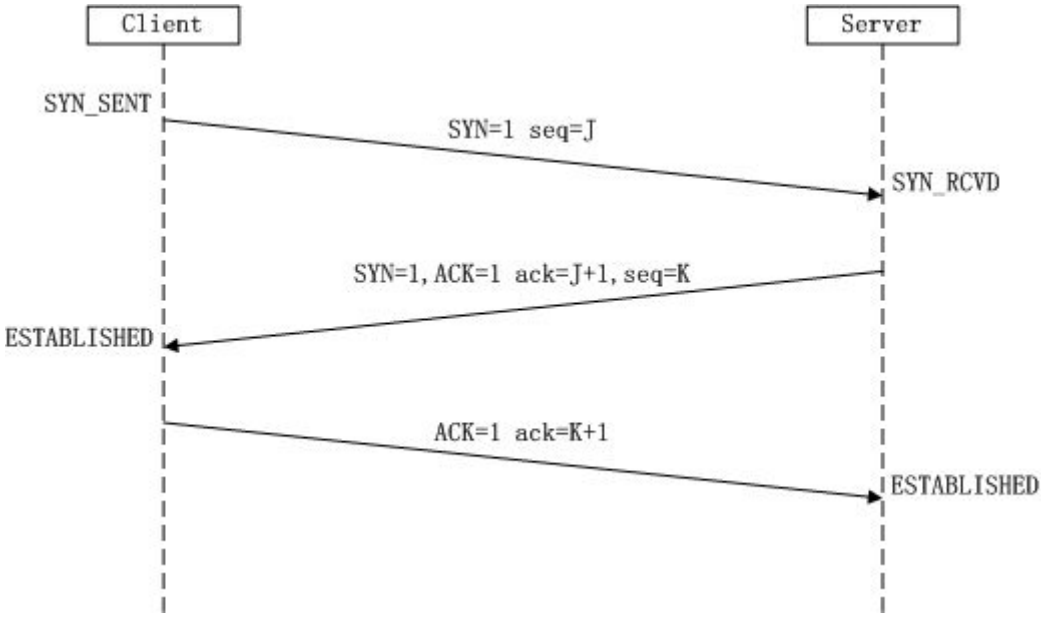
输入URL的过程：

a. 浏览器对用户输入的网址做初步的格式化检查
b. 用http还是https协议（默认http）
c. 客户端 - 本地DNS服务器 - 根DNS服务器 - com DNS服务器 - 具体网站

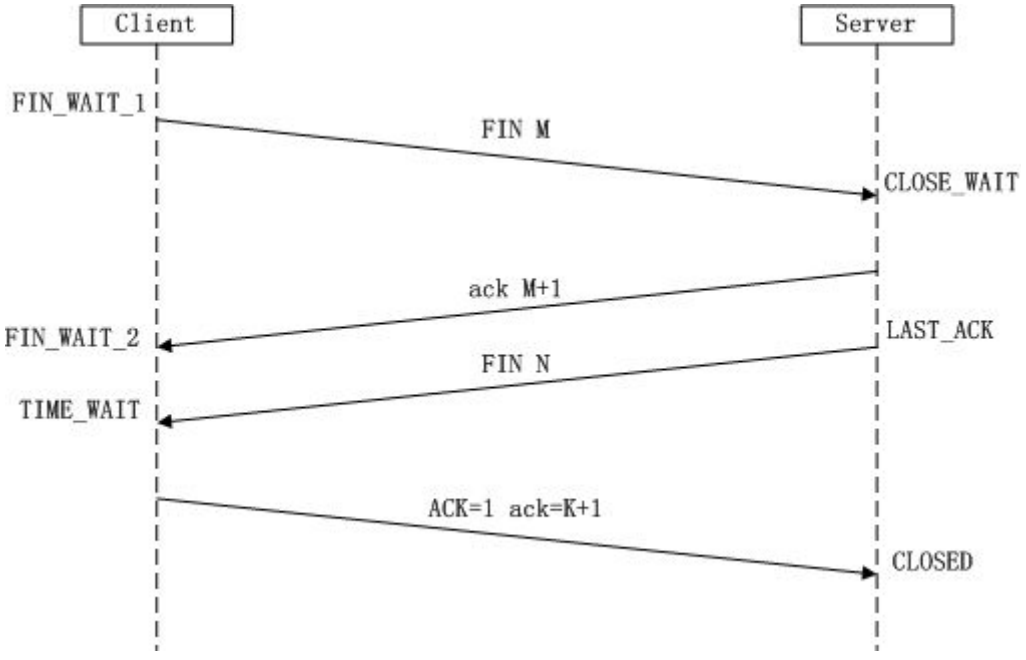
d. 建立TCP连接：**三次握手**
e. 发送http请求
f. 服务器处理请求（调度资源，业务逻辑处理，返回结果）
g. 响应：对应的比如HTTP状态码
h. 关闭TCP连接：**四次挥手**
i. 浏览器解析HTTP，CSS，JS，图片等资源
j. 浏览器渲染布局

TCP， UDP 的区别

Tcp 三次握手，四次挥手**三次握手**



四次挥手



问题：为什么建立连接是三次握手，而关闭连接却是四次挥手呢？

这是因为服务端在LISTEN状态下，收到建立连接请求的SYN报文后，把ACK和SYN放在一个报文里发送给客户端。

而**关闭连接时**，当收到对方的FIN报文时，**仅仅表示对方不再发送数据了但是还能接收数据**，己方也未必全部数据都发送给对方了，所以己方可以立即close，也可以发送一些数据给对方后，再发送FIN报文给对方来表示同意现在关闭连接，因此，己方ACK和FIN一般都会分开发送

数据库

索引：

- https://cloud.tencent.com/developer/article/10049121
- https://www.infoq.cn/article/OJKWYykjoyc2YGB0Sj2c
- https://juejin.im/post/6844903909899632654#%E8%81%9A%E7%B0%87%E7%B4%A2%E5%BC%95
- https://blog.csdn.net/suifeng3051/article/details/52669644
- 联合索引及最左前缀原理
- 为较长的字符串使用前缀索引
- 主键外键一定要建索引
- 对 where,on,group by,order by 中出现的列使用索引
- 尽量扩展索引，不要新建索引
- 不要过多创建索引：会增加对表增删改的时间

锁

- https://juejin.im/entry/6844903650897182733
- https://blog.csdn.net/qq_35246620/article/details/69943011
- 适用：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；而行级锁则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
- https://blog.csdn.net/C_J33/article/details/79487941

隔离级别

http://www.zsythink.net/archives/1233/

Redis

https://juejin.im/post/6844903592998993928

Web 开发

Cookie和Session的区别:

Cookie在客户端，session在服务端。

https://juejin.im/entry/6844903434366222350

安全攻击的手段：

https://blog.csdn.net/zoomla188/article/details/74908314/

Django：

https://www.jianshu.com/p/724233387ba3