**Python（黄）**

**1网络编程**

**01-基本概念**

1. 网络通信概述（说段子，没什么意义）
2. IP地址（信息极少，没什么意义）

Ip地址 e.g 196.168.1.1 ;目的：用来标记网络上的一台主机。

1. linux，windows查看网卡信息

ifconfig适用于linux,unix,mac；ipconfig适用于windows(进入cmd命令行模式输入此命令)

对于windows系统可以直接控制面板>网络和 Internet>网络和共享中心>更改适配器设置>点击相应网卡并查看详细信息来实现。

Plus: linux 命令模式下ctrl +a 光标定位至首位，ctrl + e定位至末尾，编辑模式下输入O在该行前一行插入空白行，输入o在该行后一行插入空白行。

Linux中用Sudo ipconfig ens40(XXX网卡名) up/down 来打开或关闭网卡；windows中直接点击相应的网卡>禁用。

1. ip地址的分类-ipv4和ipv6介绍



以上是ipv4的格式xxxxxxxx. xxxxxxxx. xxxxxxxx. xxxxxxxx;四个部分每个部分的取值为0-255。

可以分成两段：网络号和主机号。

A,B,C三类IP地址的主机号分别占据3段，2段，1段。

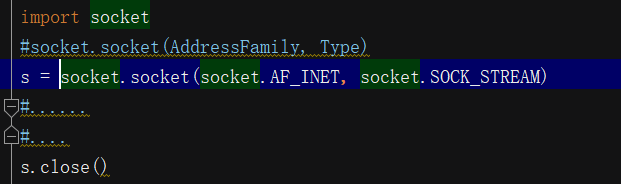
另外注意主机号的有些数值是不能取的，如C类地址主机号的0,255（首末）两个数值不能取。

1. 端口（没什么实质内容）
2. 端口分类：知名端口(well know ports)，动态端口(dynamic ports)

程序运行之后会产生进程（在任务管理器中看到的就是进程），每个进程会分配到一个端口号。

端口号一共有65535个0-1024是知名端口，有其约定的特别用途，其余的动态端口，端口号可以随便用。

07-socket介绍



Python创建套接字，图中是创建UDP的套接字。

**02-UDP**

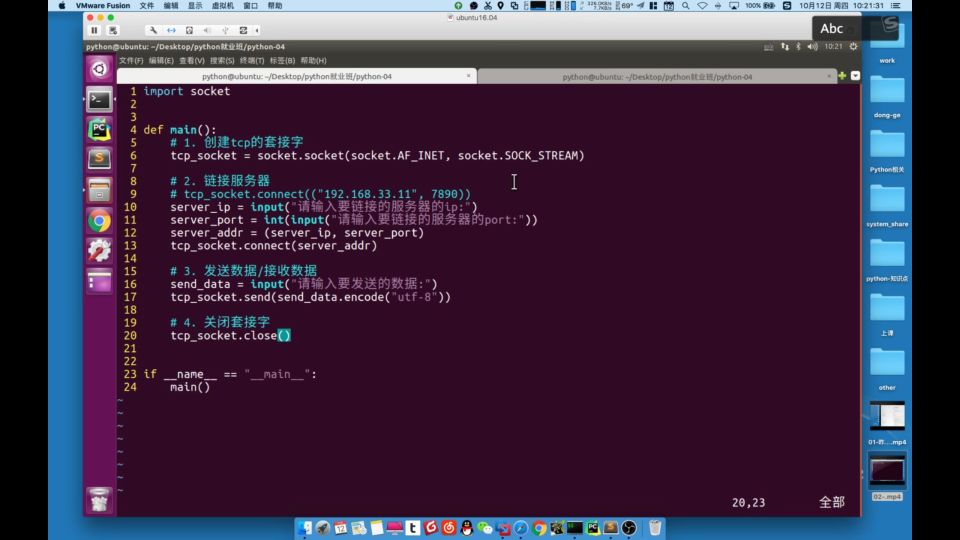
**03-tcp客户端**

Tcp(传输控制协议)

Tcp通信需要经过创建连接，数据传输，终止连接三个步骤。是面向连接的，双方必须建立连接才能进行数据的传输。

Tcp和udp的区别通俗来说前者类似于打电话后者类似于写信，前者如果对方没收到信息仍然有补救措施保证信息传输到目的地，后者则无法弥补（TCP在通信之前需要建立连接，对方收到信息之后会有反馈应答，让发送方知道信息是否顺利传输，比如我们用迅雷下载电影能同时看到下载速度和上传速度一样，这个很小的上传速度就是TCP协议中要求接收方有所反馈。）

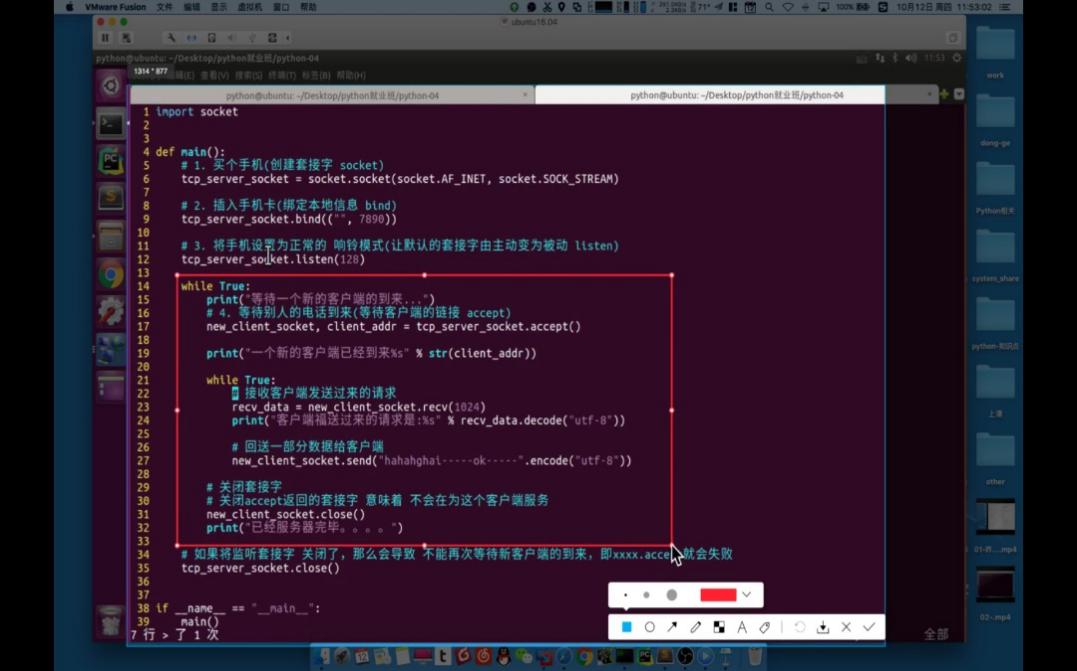
02-（重点）tcp客户端

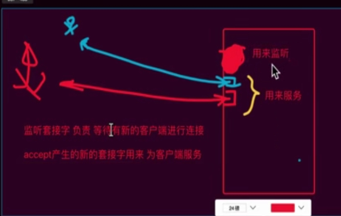


一切尽在图中。

**04-TCP服务器**

1. 创建套接字
2. Bind绑定ip和port
3. Listen使套接字变为可以被动链接
4. Accept等待客户端连接
5. recv/send接收发送数据





注意listen和recv的参数是一个数字，recv的内容需要用decode解码再输出，否则只是binary；accept这一步需要拆包，然后使用新的套接字来和客户端进行交流，原先的tcp\_server\_socket用来监听，注意如果没有服务器连接客户端那么accept这里就会阻塞，如同input函数没有收到输入值；str(client\_addr),将元组转成字符串；我们这里的第一个while True的代码块是为了循环为多个客户端服务，第二个while True是为一个客户端循环服务，所以上述截图可以循环为多个服务器进行多次服务。

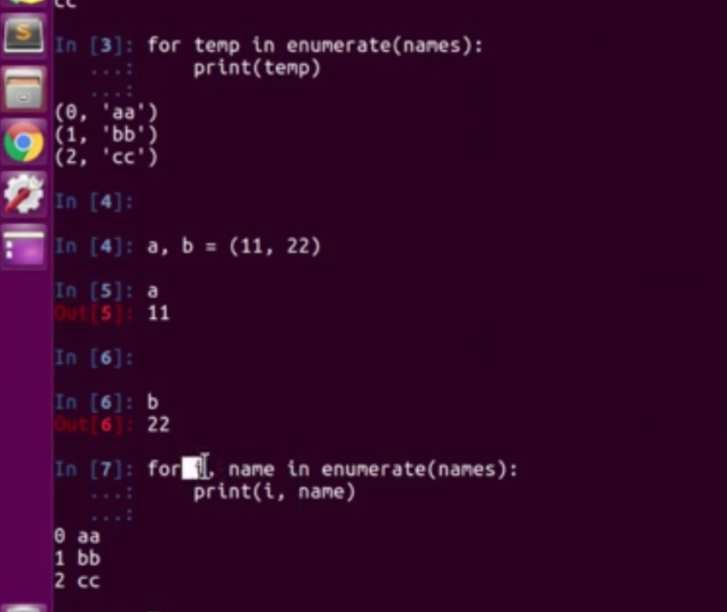


这里我们在第二个代码块中加入了if, else语句。注意recv在没收到客户端数据的时候也是处于堵塞状态的，可以通过给服务器发数据来解堵塞，如果客户端的套接字直接关闭的话recv也会解堵塞并且接收到一个空值,(if 0; if []/();None,这些条件都是不满足的，返回值为False),这样程序就跳到else来终止循环。

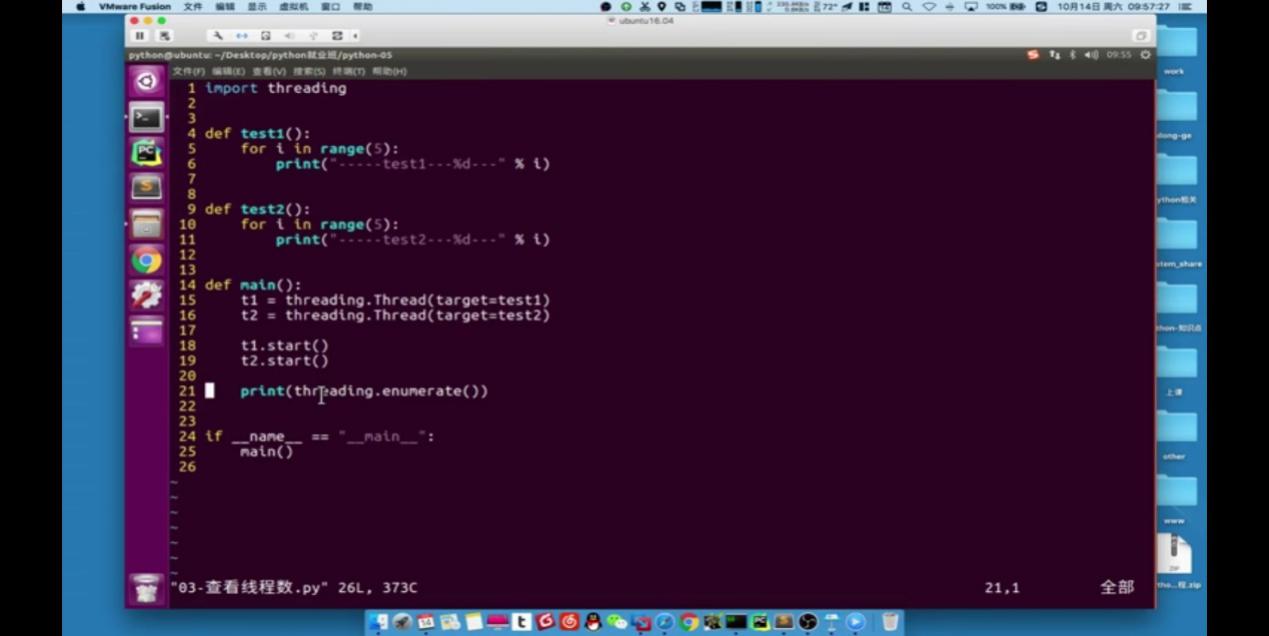
05-tcp下载文件（待总结）

**2多任务**

1. **多线程**
2. 多任务介绍,以及threading的基本使用（简单介绍,内容较少）
3. Thread创建线程，完成多任务（实质内容较少）
4. 查看正在运行的线程、主线程等待子线程先结束（重点）

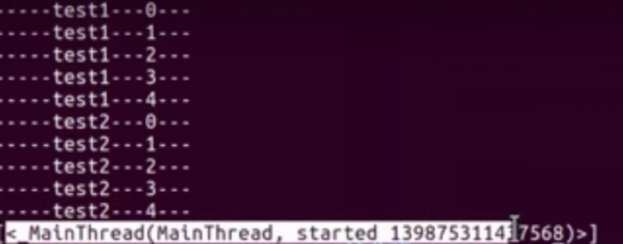


首先了解一下enumerate,可以将列表中的元素转成元组。



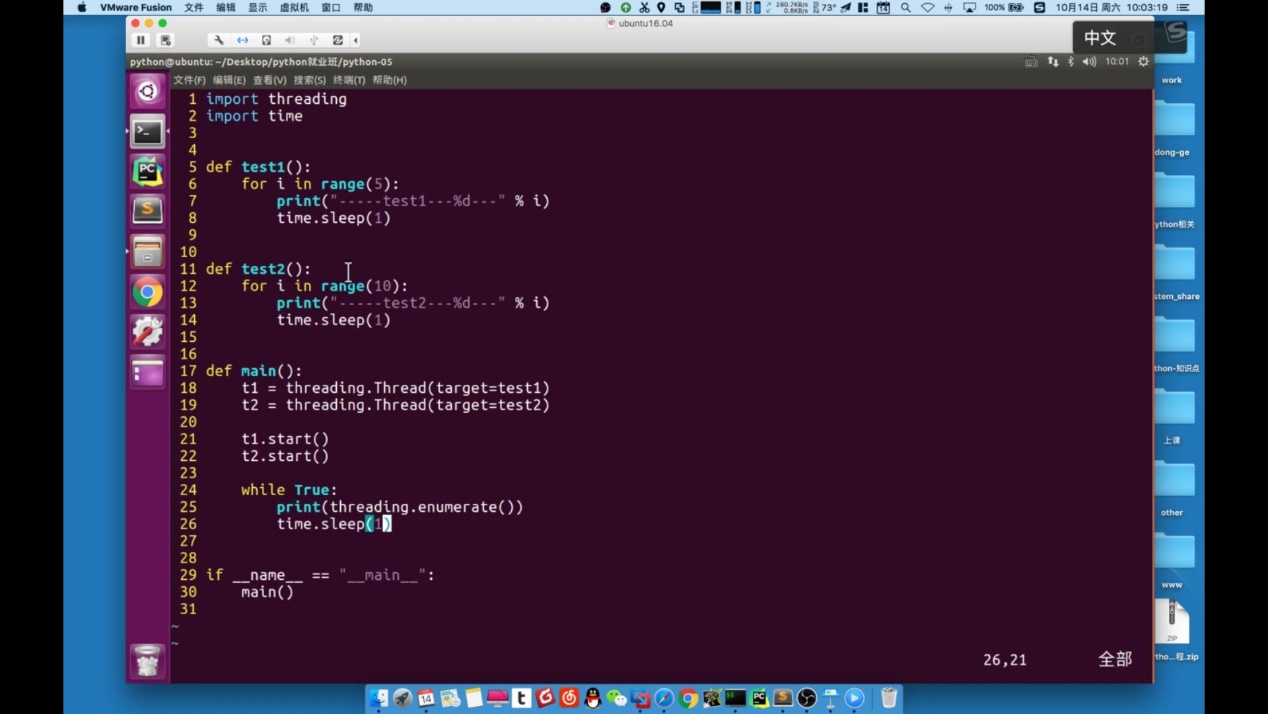
Plus:这里的enumerate()可以以list形式返回当前的线程。

运行结果：

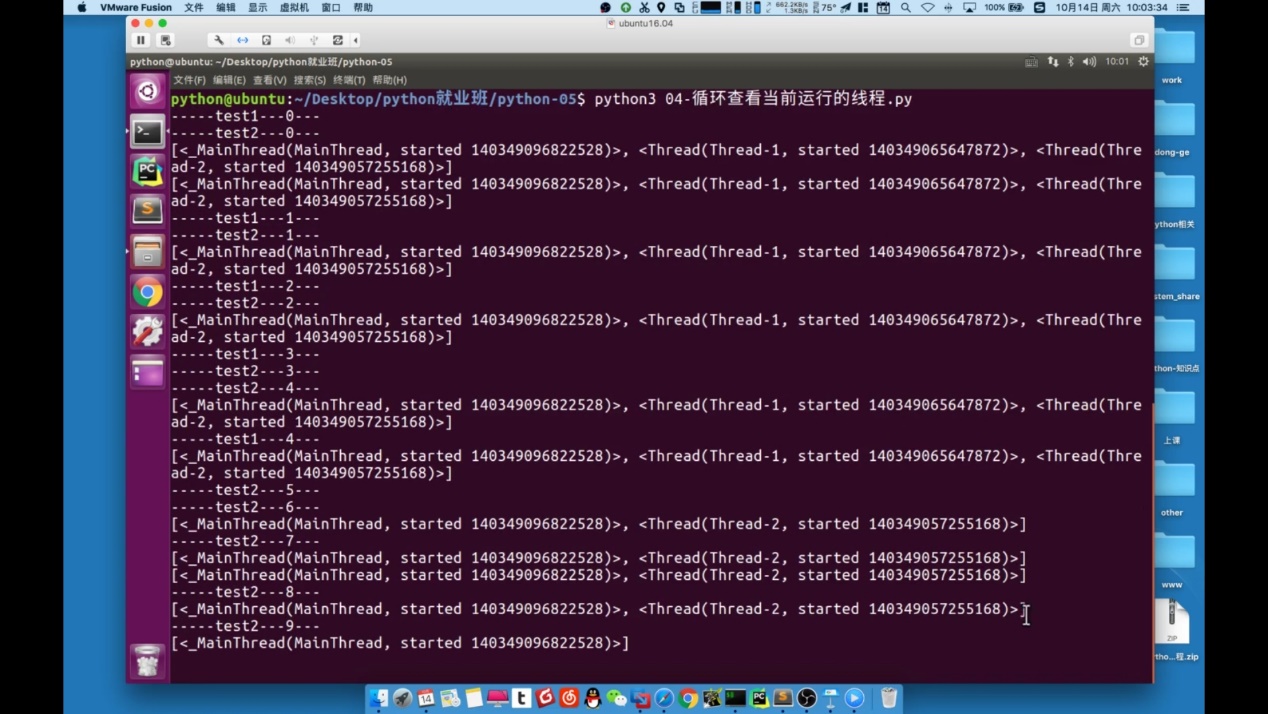


解析：这里一共三个线程，即除了主线程还有两个子线程，操作系统调度线程是没有先后顺序的，即他们可以视为平权的。由于子线程没有sleep函数这个耗时操作，一个子线程被调度后瞬间运行结束，所以会看到test1和test2扎堆出现而非交错出现。

补充一个例子：

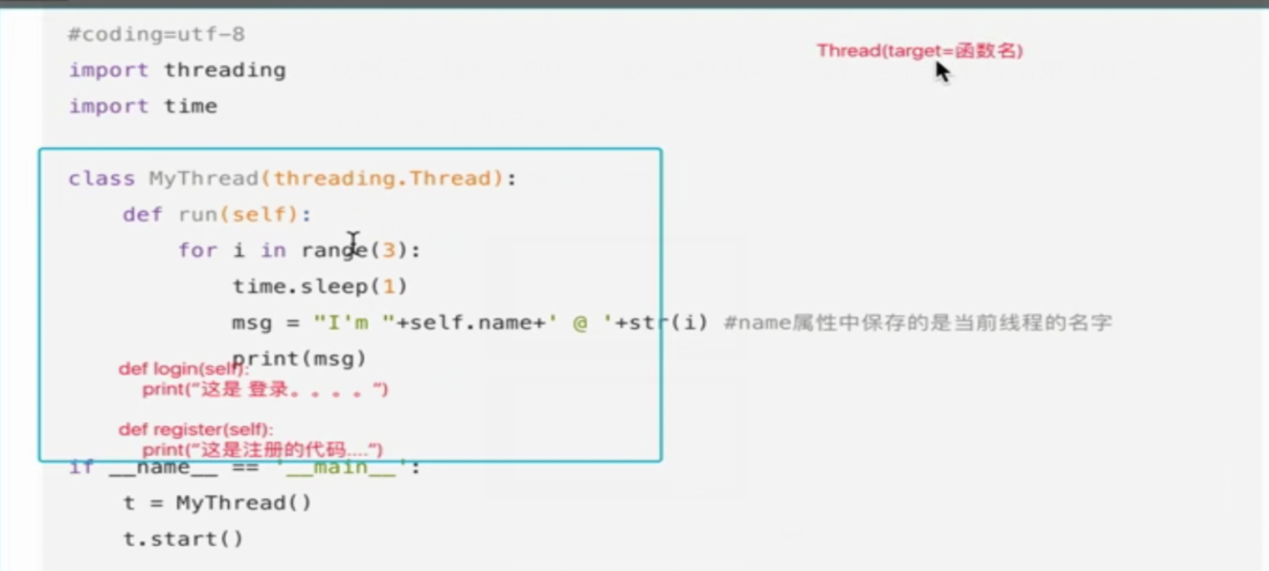


运行结果（局部）：



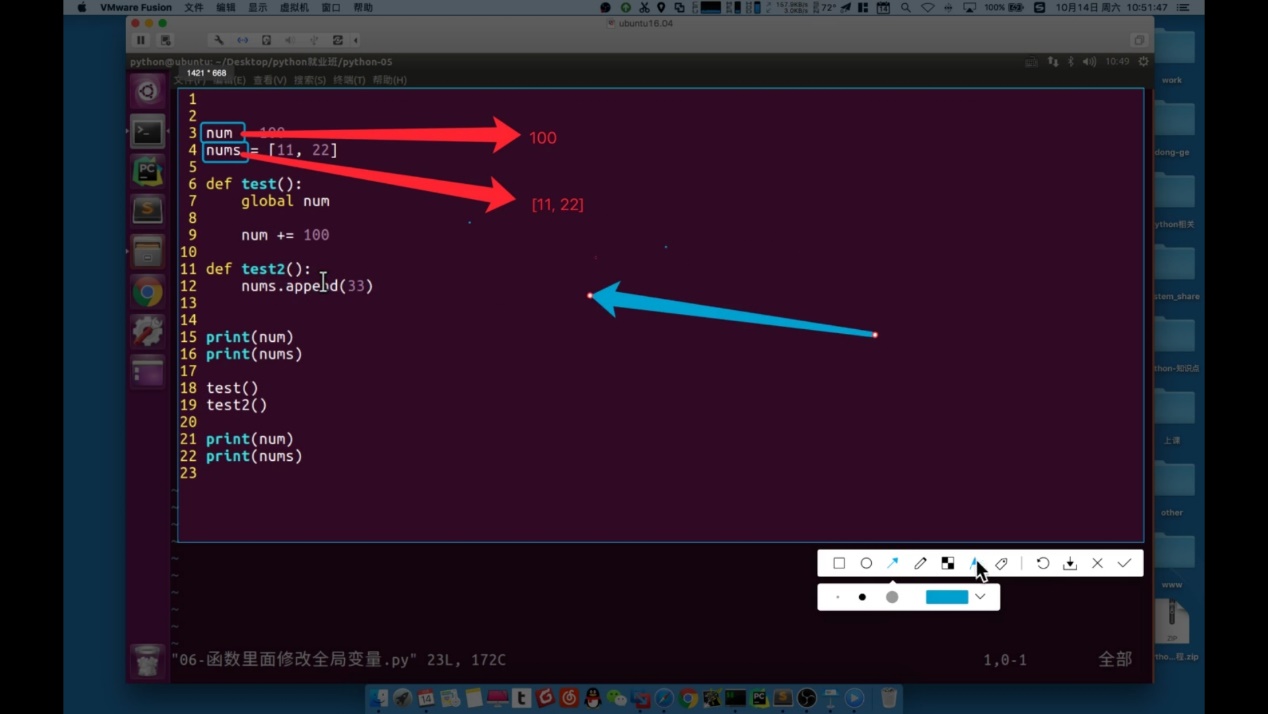
这里一共3个线程，两个子线程以及一个主线程，一开始操作系统随机调度三个线程（rt,1 2 主；主 1 2；主 1 2；主 1 2；2 主1）；之后test1的线程运行结束死掉了，剩下主线程和test2子线程；最后只剩下主线程。Plus:主线程会默认等子线程全部结束再结束，因为主线程一死子线程全部要死。

1. 通过继承Thread类完成创建过程



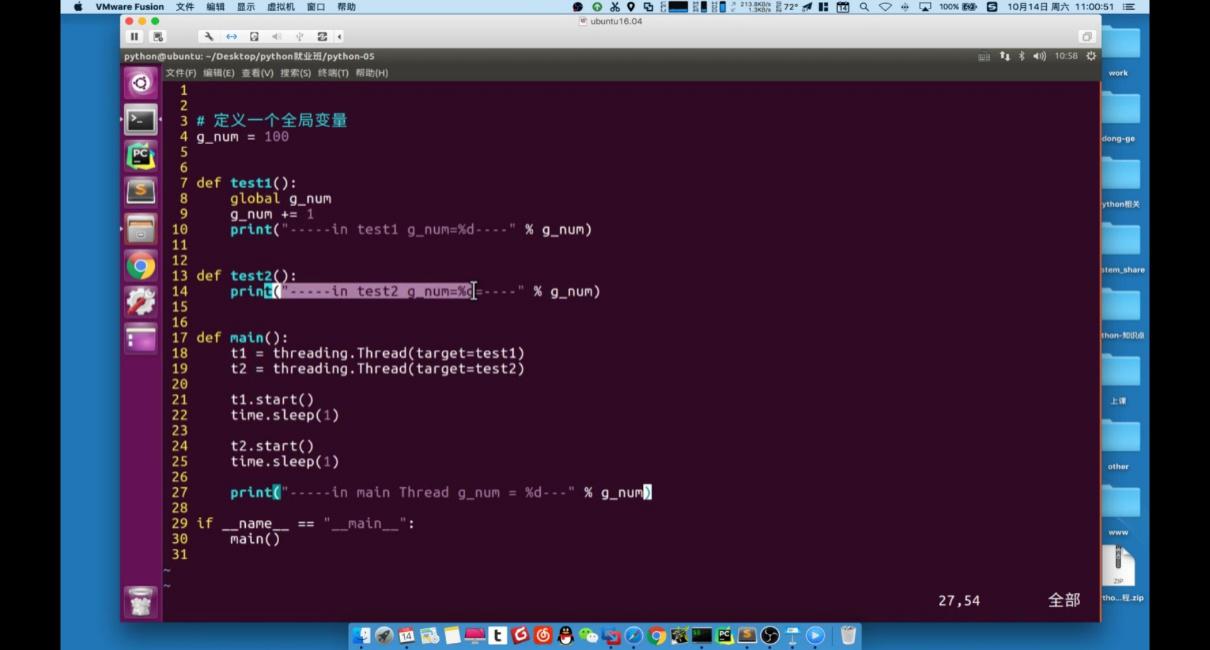
MyThread通过继承并定义run方法（若不定义run函数程序无法运行）来实现多线程，**这里t.start()会自动调用run函数，不需要把函数作参数传进去。我们可以把需要执行的函数放入run函数的代码块中来运行多个函数**

1. 略
2. 多线程共享全局变量



首先提一下关于全局变量的修改，如果我们在函数中直接改变变量名所指向的数据的值，比如截图中test2函数中的情况，那么不需要加global； 如果我们是把变量名的指向给改变了如截图中test函数，那么就需要加global。

下面这张截图很好的体现了多线程的全部变量共享

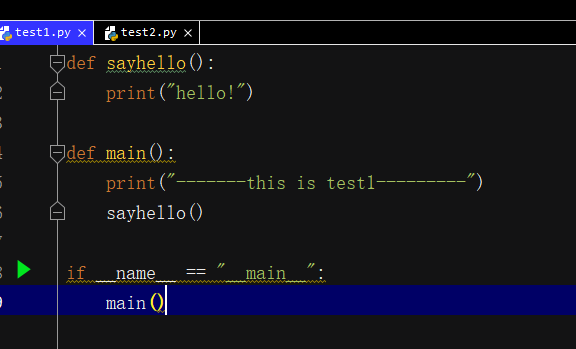
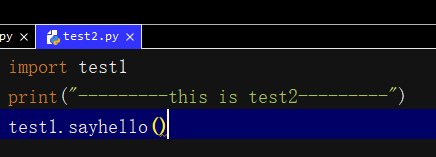


运行结果：



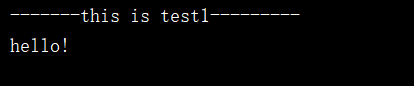
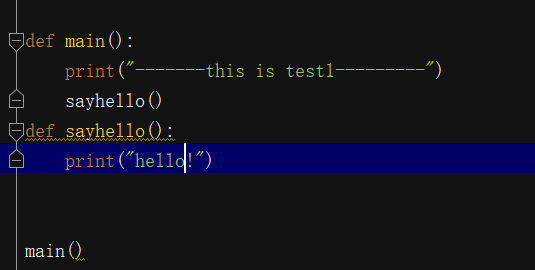
可以看到2个子线程以及一个主线程的g\_num值都为101。

Plus:关于语法细节

这里test2运行结果显而易见，但如果把左图中test1的if条件去掉（直接写main()），那么test2在导入test1时会直接把test1运行一遍，if \_\_name\_\_ … …条件就是为了让当前模块被导入时，其下的代码块不会直接被运行。

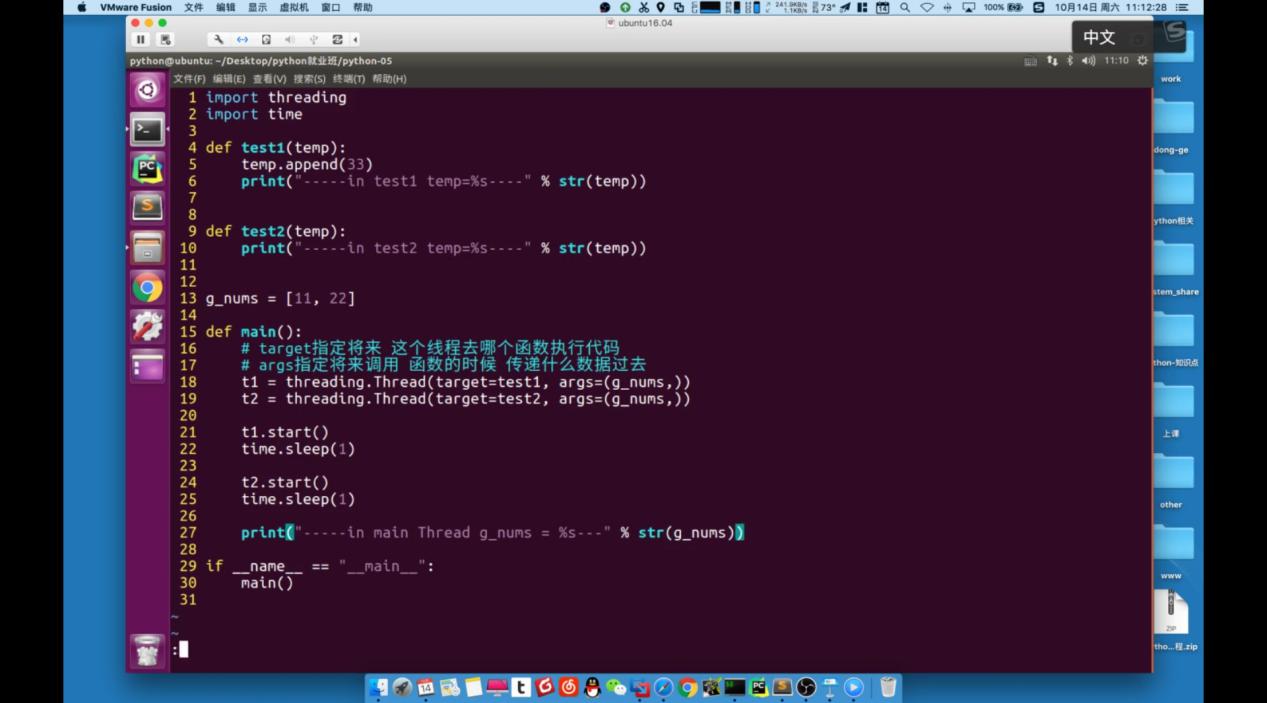
另外注意：



在函数定义之前调用，如果这个调用语句也是在其他函数定义之中（或类定义）嵌套的，那么仍然可以正常运行（如果调用语句在外面显然运行不了）。因为这里在运行main()时上面的定义已经跑过一遍了，电脑已经知道了这两个函数的定义了。

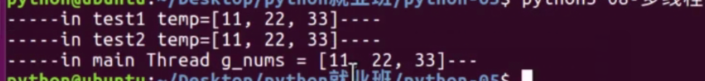
07-多线程共享全局变量-args参数

之前的例子中函数并没有参数，如果函数里面有参数的情况下我们启用多线程那就需要用args传递参数：

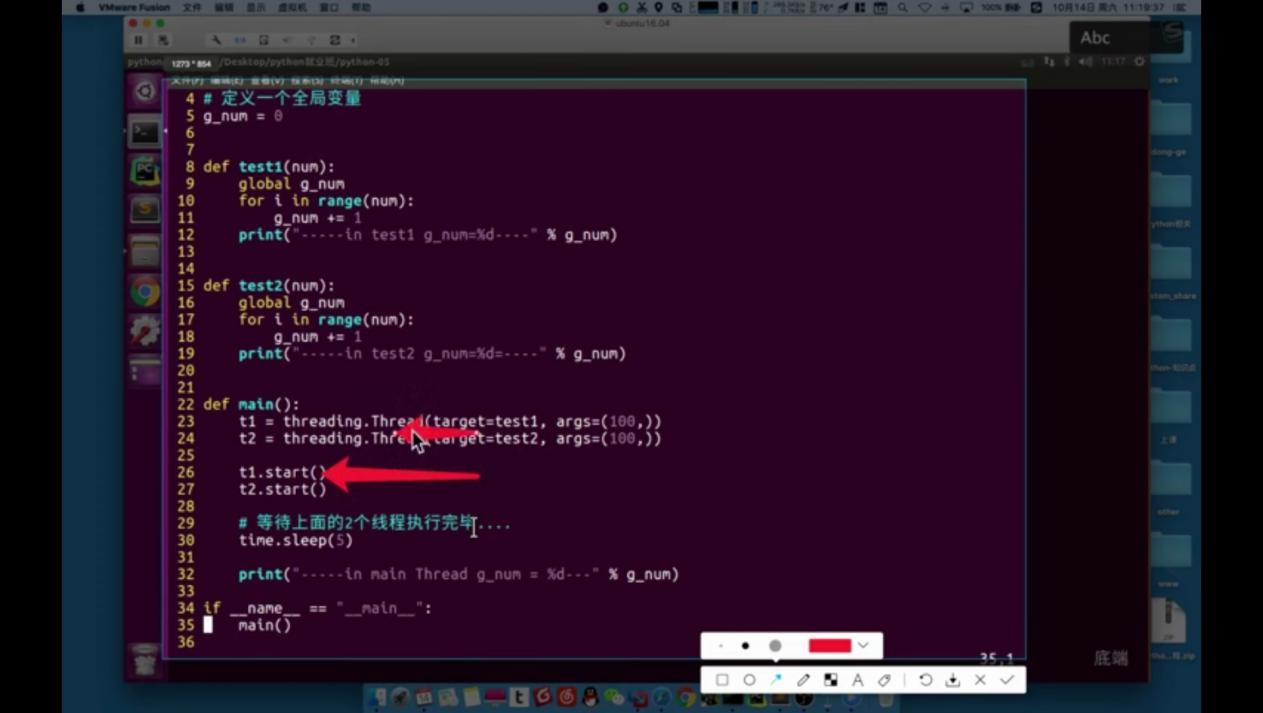


在Thread方法里面用args给函数传递实参，即g\_num的值传递给tmep。注意args后接tuple，括号内如果只有一个值是要加逗号的。

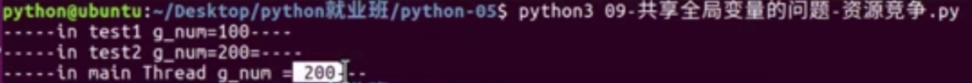
运行结果：



08-共享全局变量的问题

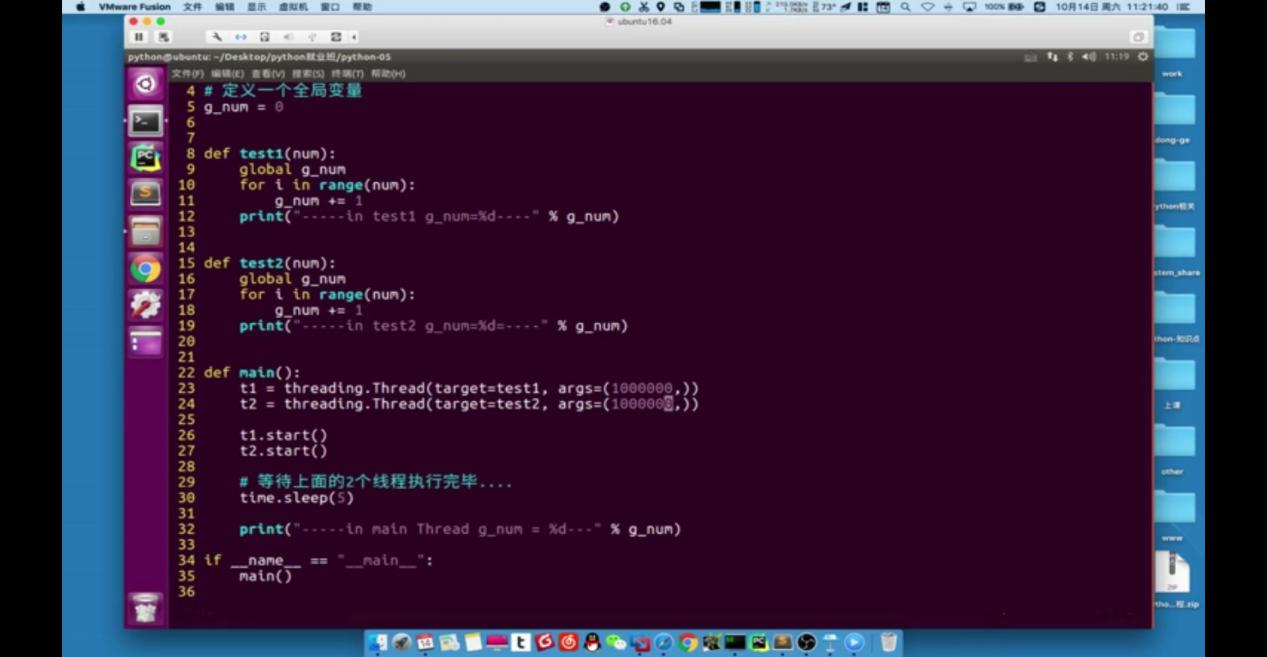


输出结果：

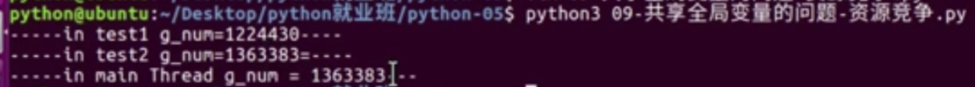


意料之中，步多做解释

如果把100改成1000000：



输出结果：



这里输出结果并不是2000000.

原因：



程序在运行时,线程中的for循环可以视为三步，第3步是把值存到变量中。在运行过程中可能会出现操作系统切换线程的时候值还没存进去的情况，这样使得最终得到的结果小于2000000.至于num = 100的情况那是因为线程需要的时间太短，瞬间100次加完并且存入了变量值（1，2两部交替执行，执行n个轮回之后执行第三步,循环往复）。

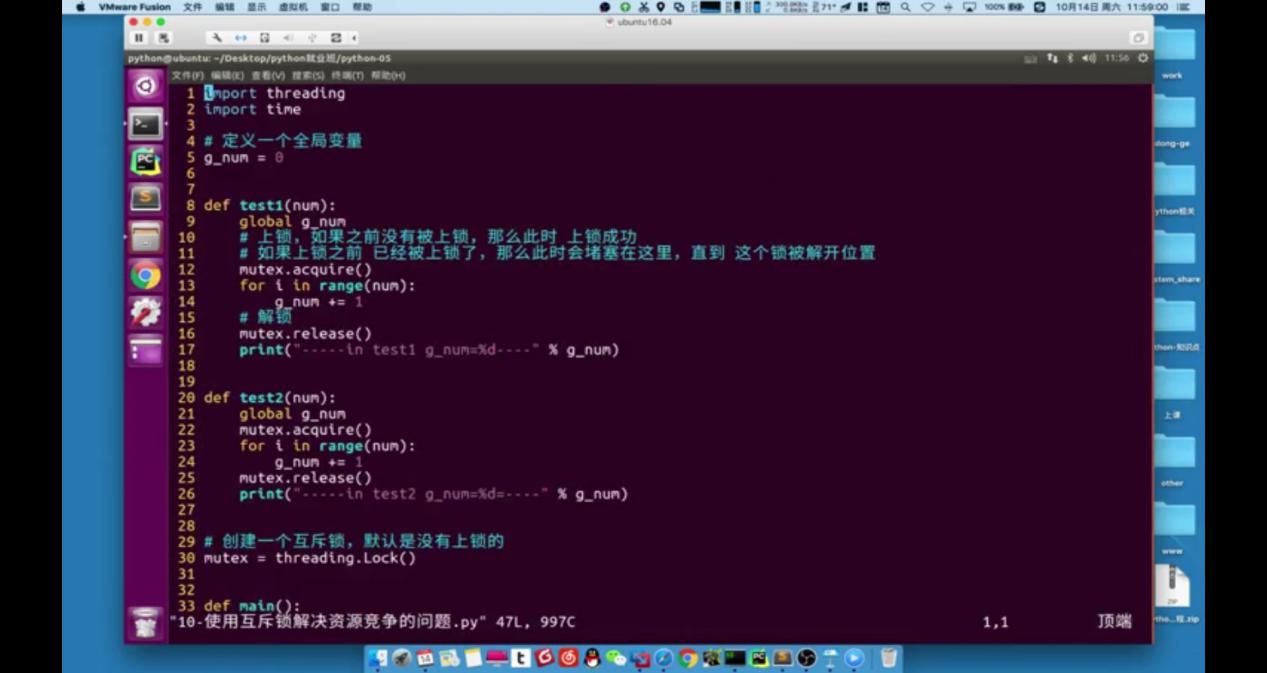
09-同步概念、互斥锁解决资源竞争

上面的例子就是资源竞争的现象，如果要解决就需要引入互斥锁。

Mutex = threading.Lock() #创建互斥锁

Mutex.acquire() #上锁

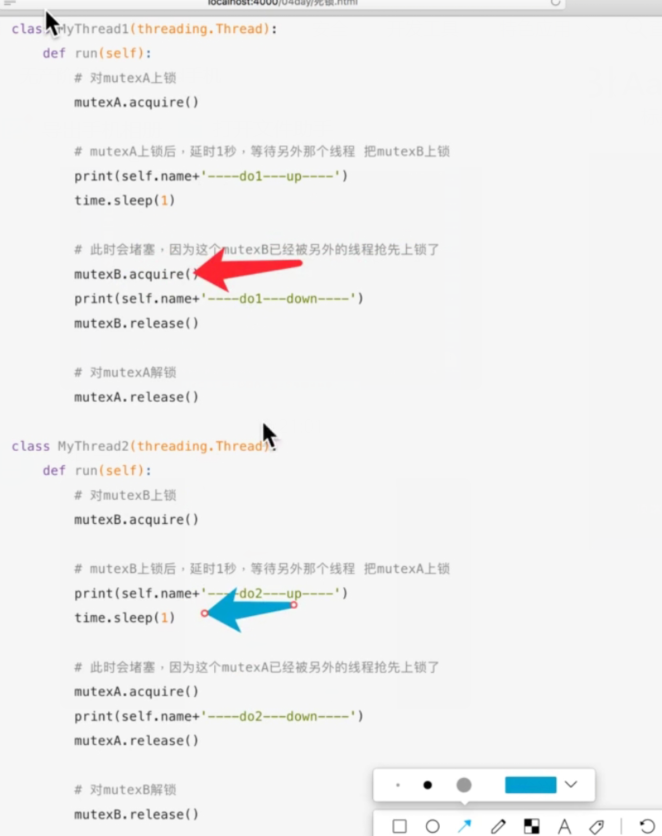
Mutex.release() #解锁



把出现资源竞争的地方上锁，问题就解决了。

输出结果是100000和2000000.

1. 死锁、银行家算法



上面的截图展示的就是思索，程序将一直卡死在这里。

在设计代码的时候要避免。

1. 案例：多线程版udp聊天器（带看）
2. **进程**

01-线程知识点复习（略）

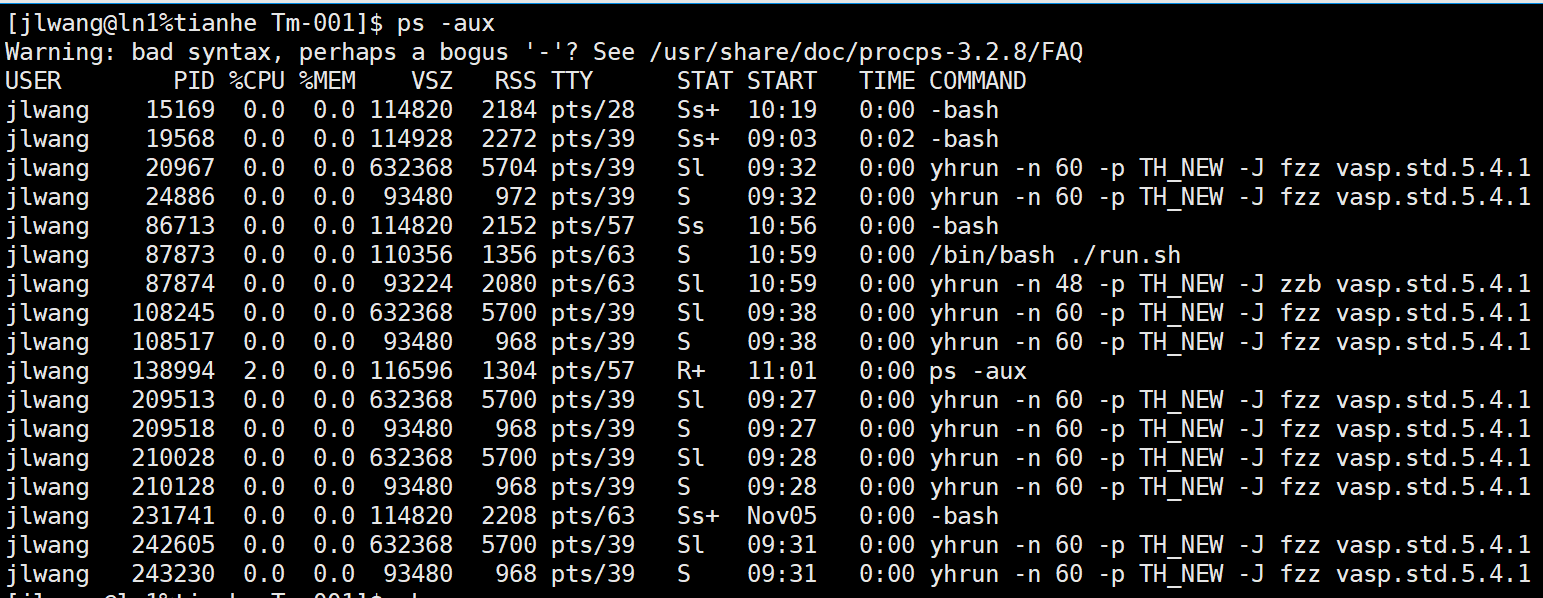
02-进程、程序的概念

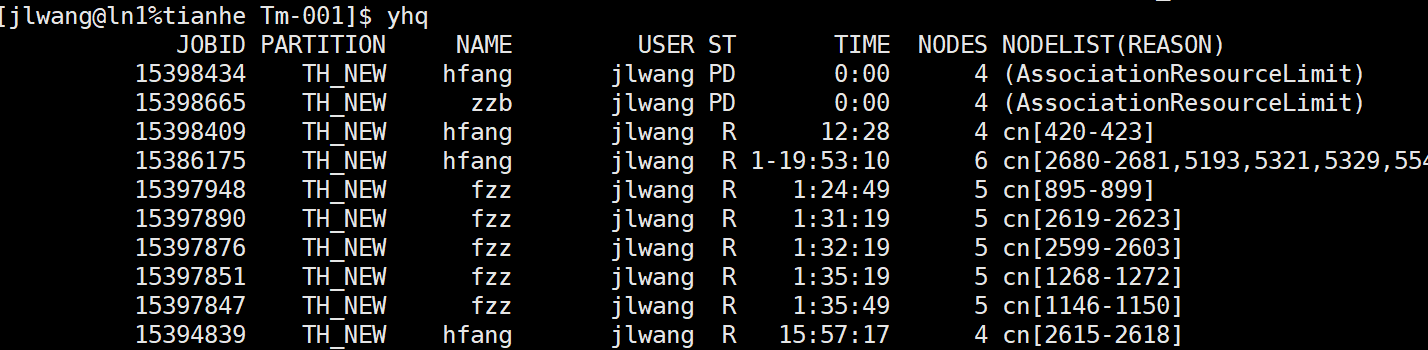
我们打开windows中任务栏，其中每条信息都对应着一个进程。程序是二进制的exe文件，是静态的，运行起来后叫进程，是活的，需要占用资源（比如它可以调用鼠标键盘摄像头，进行内存读写），电脑上打开n次qq客户端，会产生n个qq的进程：



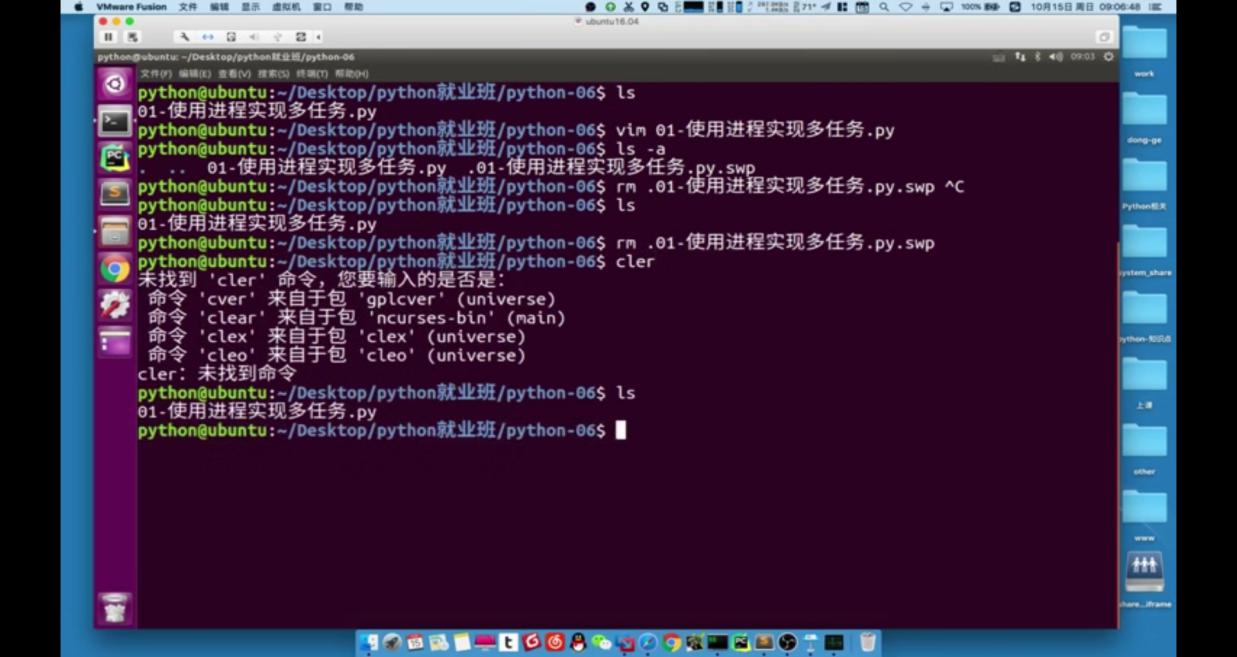
03-使用process完成多线程

插入一点,linux中查看进程 ：ps -aux

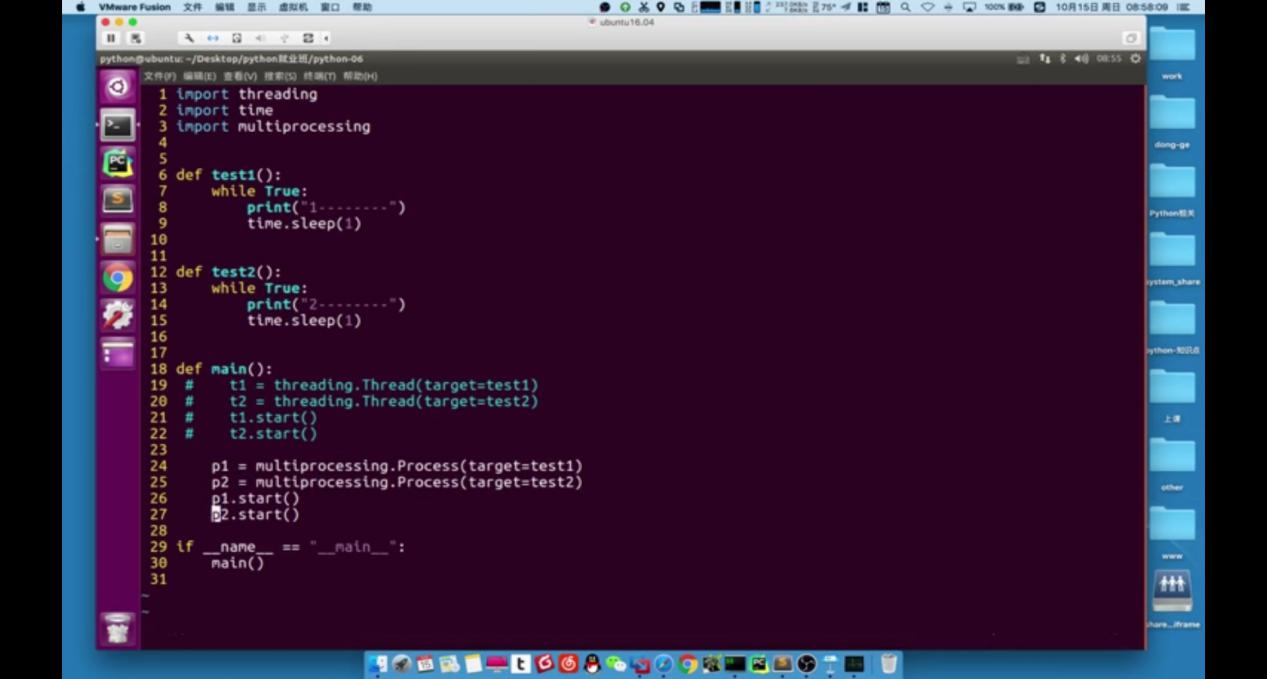




可以通过kill + 进程号（PID）来杀死进程，注意PID和任务号不是一回事。

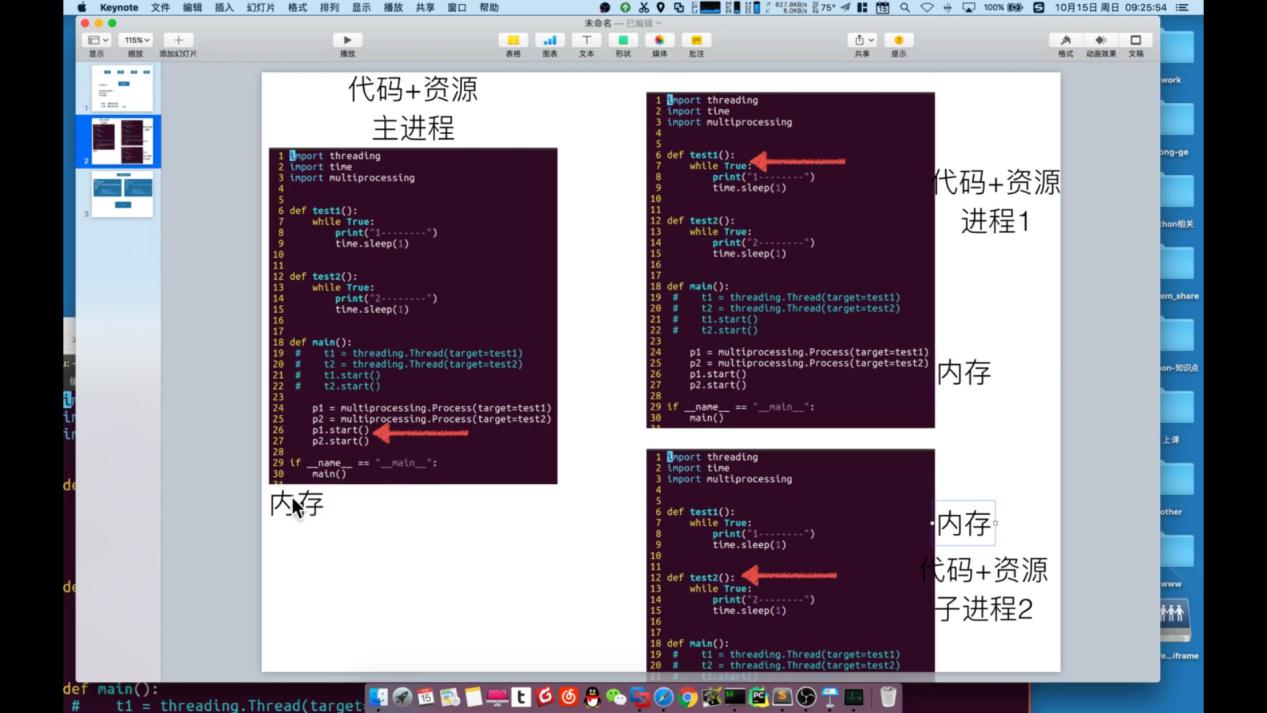


补充一个常见的linux系统的问题。在编辑模式下没有保存就异常退出了，当再次打开文件时会出现一个选择界面，操作之后才能进入文件，这是因为非正常退出使系统在文件所在目录下产生了额外的文件（以.开头，这类名字的文件是隐藏文件，需要用ls -a才能看到，删除之，便可以直接打开文件）。



上图是多进程的例子，导入multiprocessing模块的Process方法实现多线程。结果和之前的threading.Thread（多线程）是一样的。

04-使用Process完成多进程-补充

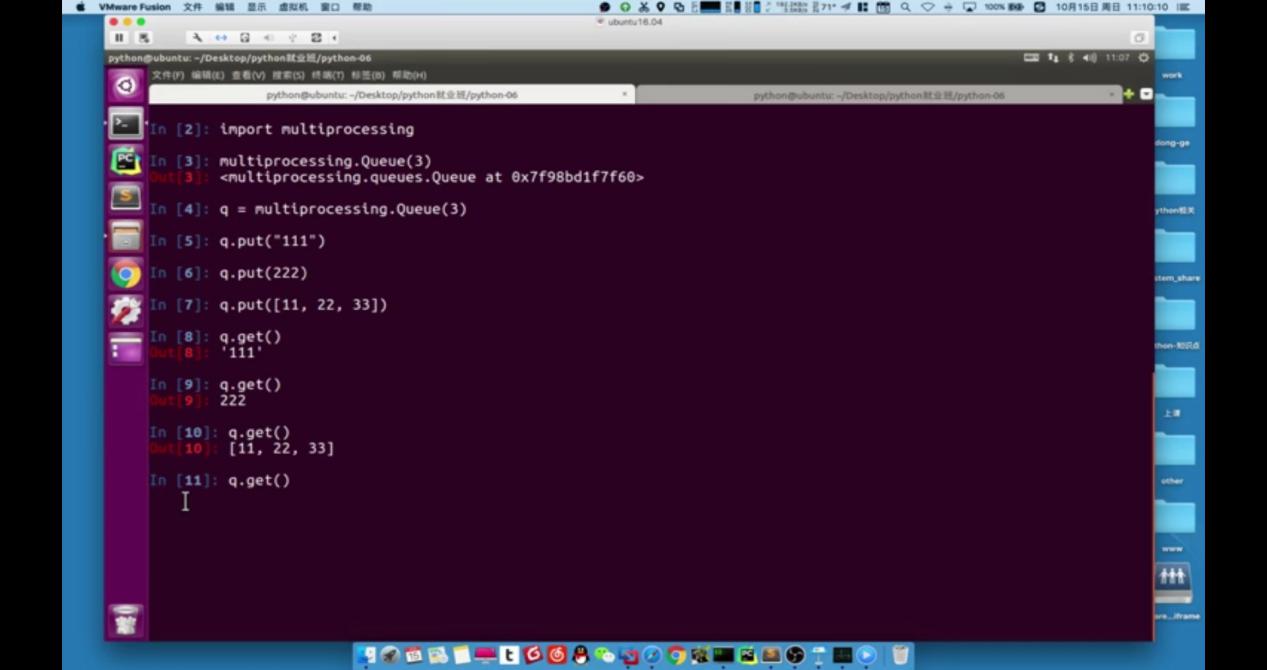


多线程中的子进程相当于需要复制父进程的代码，占据同样的资源和内存，所以比多线程浪费资源。

05-进程、线程的区别

现有进程再有线程，进程是资源分配的单位，线程是操作系统调用的单位，一个进程中起码有一个线程（主线程）。进程之间是相互独立的，线程之间共享全局变量。

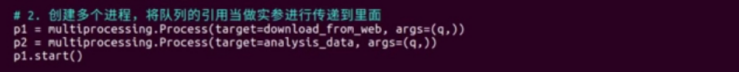
06-通过队列完成进程间通信

进程间可以通过网络来通信，也可以通过队列开辟出一个公用的内存空间。

**栈是先进后出，队列queue是先进先出。**

队列满了之后再放入数据会堵塞，队列空了后再取也会堵塞。





（一行截图截不下来）

\*\*\* 可见创建多线程和多进程的写法非常类似并且二者都可以通过args来传参

这个程序要注意在主程序中，创建多进程之前就要创建队列。之后在创建多进程的代码中把队列当成参数传递到子进程中。

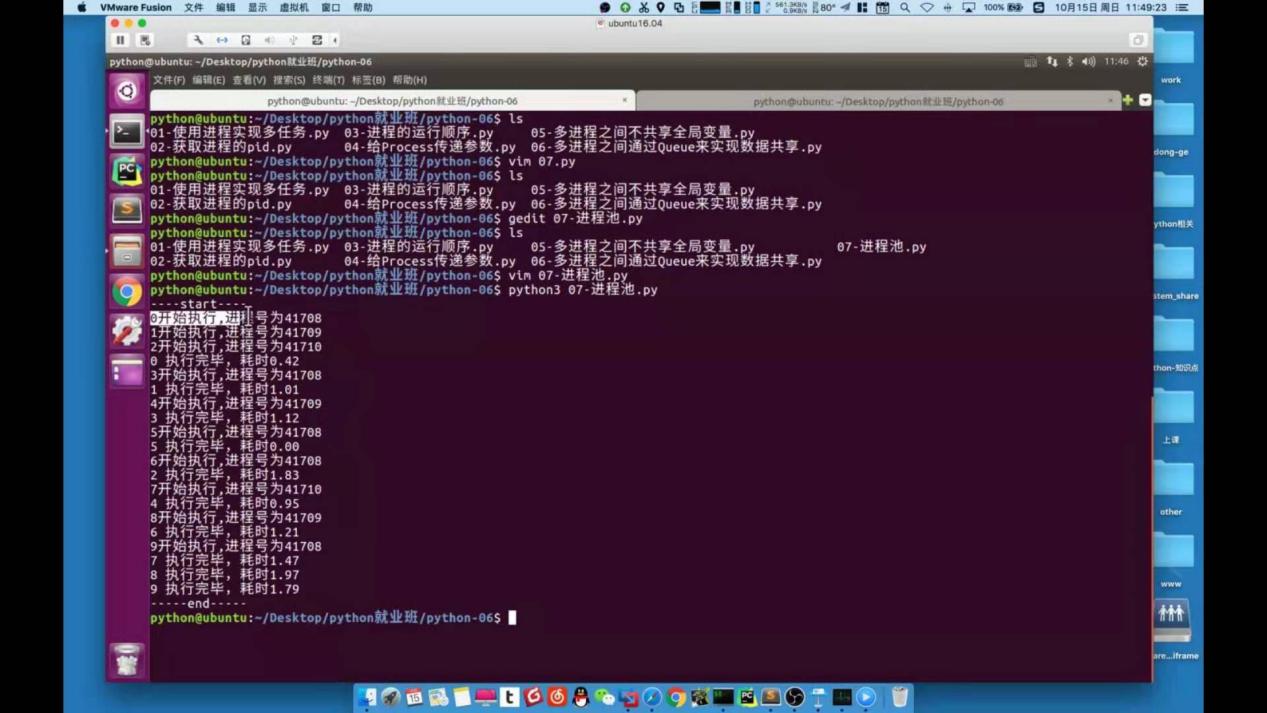
07-进程池概述

当需要创建的子进程数量不多时，可以直接利用multiprocessing中的Process动态生成多个进程，但如果是上百或者上千个目标，手动的去创建进程的工作量巨大，此时就可以用模块中的Pool方法。

08-进程池的创建



运行结果：



Pool(3)意味着进程池能同时让三个进程处于运行状态，其他多余的进程在着三个进程运行结束之后补位进去（可以看到结果中进程号三个一轮回）。

注意一个细节，用Process创建进程的程序中主进程会默认等到子进程结束再结束，而用进程池主进程不会等待子进程，可能会导致程序死了子进程还没结束，这里的po.join()就是让主进程等子进程（注意这句话要放在关闭进程池之后）。

**\*\*\* 线程(或进程)中的主线程(进程)会在等待子线程(进程)结束之后再结束主线程,但是若子进程是由进程池创建的则主进程不会等待子进程(在关闭进程后调用join()方法可以解决这个问题)**

**\*\*\* threading --- Thread Lock**

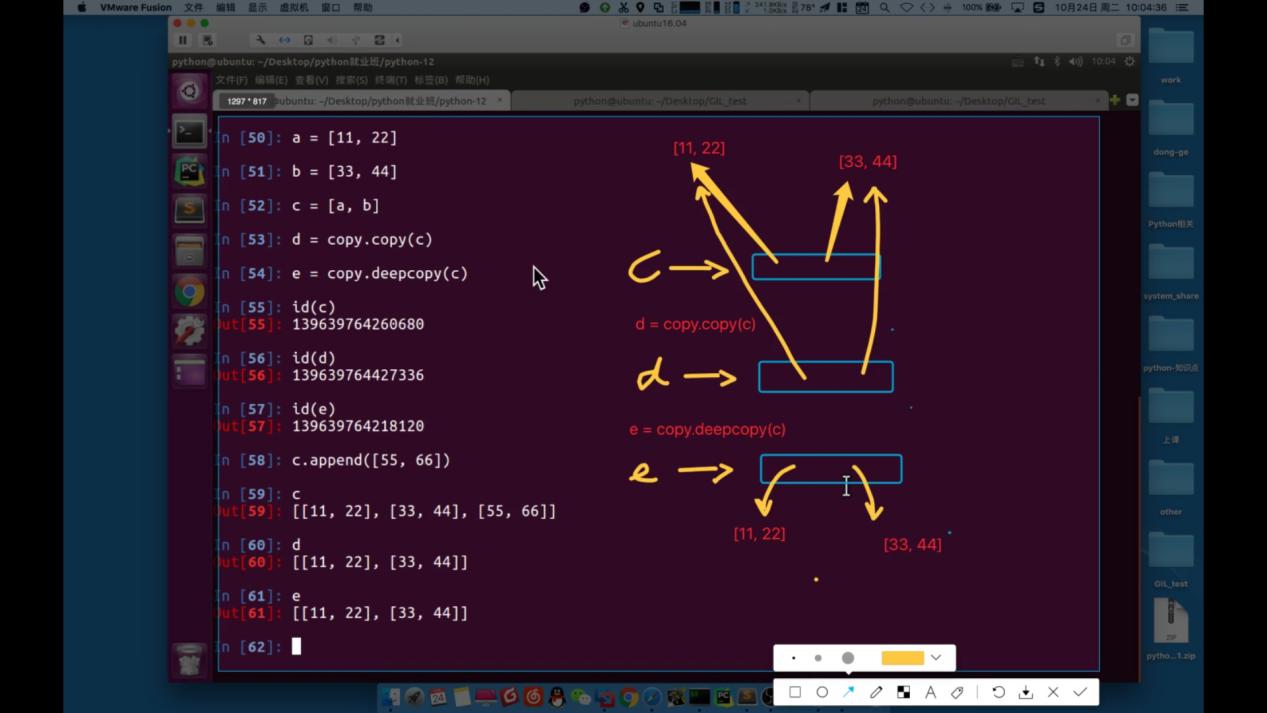
**\*\*\* multiprocessing --- Process Pool Queue**

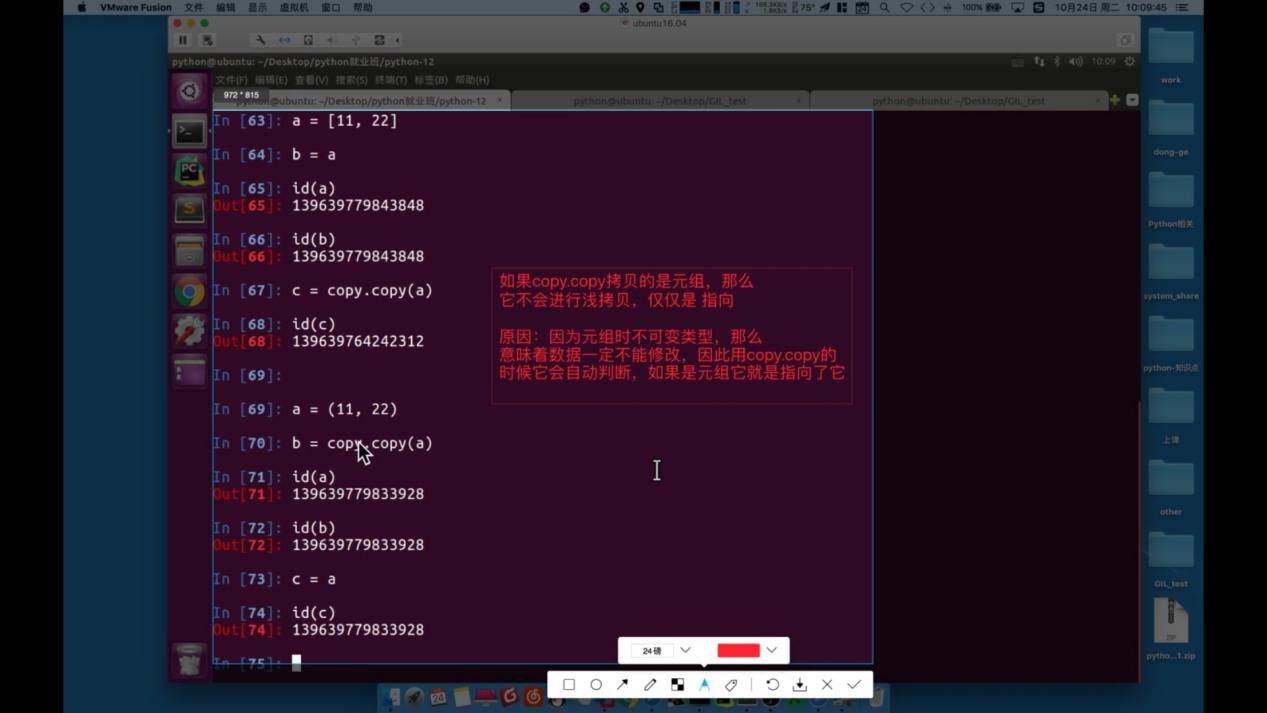
**----------------------python高级语法----------------------**

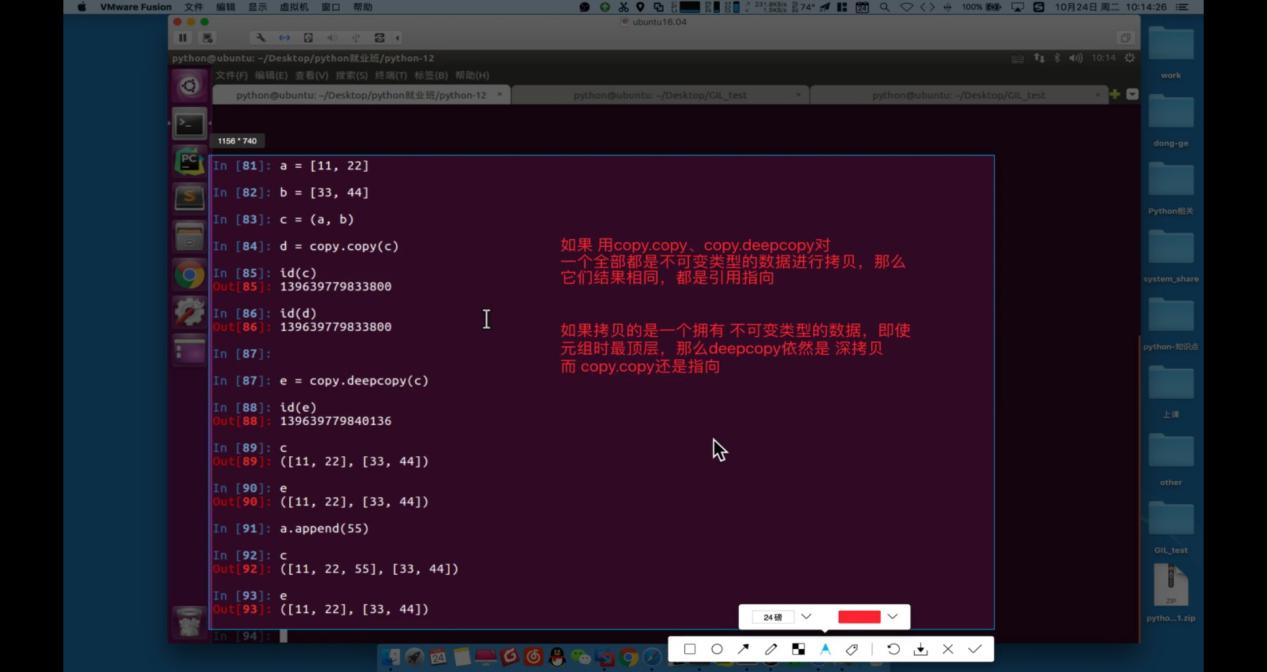
**关于GIL锁：**

1. GIL(全局解释器锁)和python没关系，仅仅是由于历史原因在Cpython解释器，难以移出GIL。
2. **每个线程在执行时都需要获取GIL**，保证同一时刻只有一个线程可以执行代码。
3. 线程释放GIL锁的情况：在IO操作等可能引起阻塞的system call之前可以暂时释放GIL，并在执行结束后重新获取GIL，python3.x使用计时器(执行时间达到阈值后释放GIL)
4. python的多进程和多线程是可以真正利用CPU的多核资源的(并行)
5. \*\* jpython解释器则不存在GIL锁的问题，其多线程是并行，然而通常下载的Python默认用的是Cpython解释器(官网推荐)。
6. \*\* 对于计算密集型代码用多线程并无优势，但对于IO密集型代码用多线程比较高效. 计算密集型：多进程；IO密集型：多线程(or 携程)
7. \*\* 如果想要让python的多线程称为真正的并行，除了更换解释器之外还可以在python代码中的线程中引入其它语言的代码(python中调用C C++ Java javascript等是没问题。

**关于浅拷贝和深拷贝**：

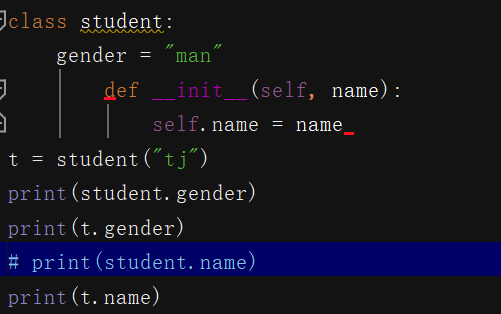






不赘述，看图。另外注意list的切片操作和字典的copy方法(如a.copy(),a为字典)都属于浅拷贝。

**关于类和私有变量：**

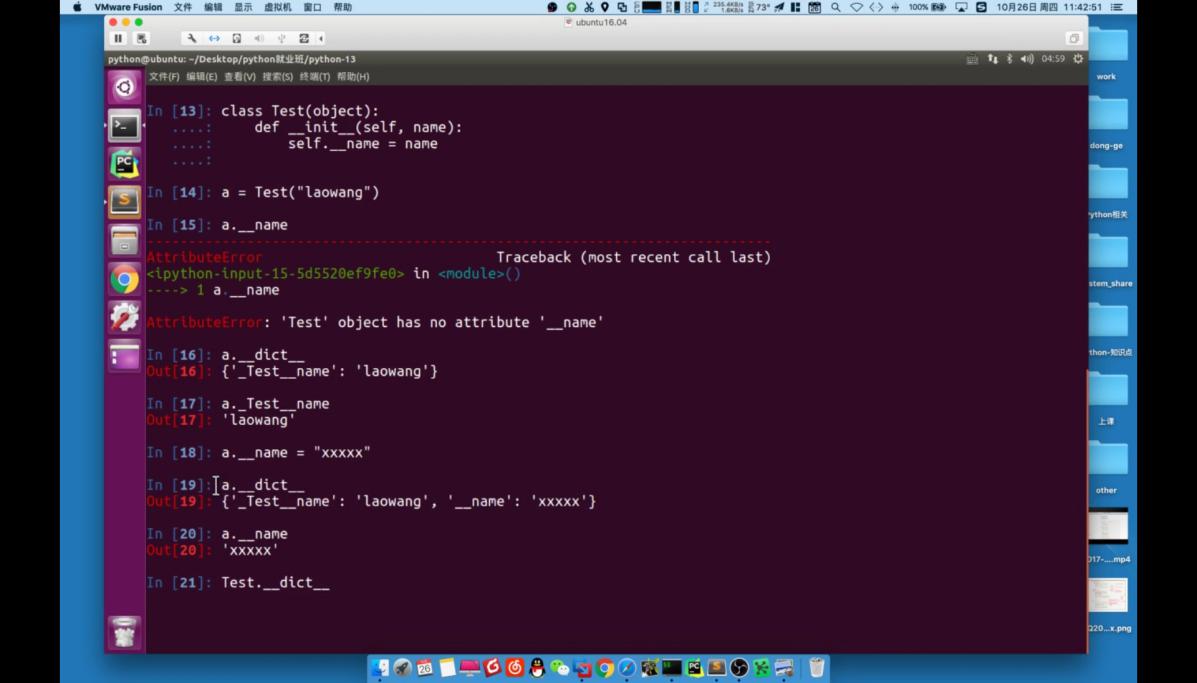
 

类定义中没用self的变量可以被类和实例访问到，用self声名的变量只能被实例访问，所以截图中第三个print语句会出错。 实际运用中若希望各个实例可以获得不同的属性值，那么用self声名，反之直接声名。

用双下划线和但下划线命名的变量为私有变量

\_x : import a后欲用其中的变量或函数需要写a.xxx的形式，from a import \*则可以让代码中免写模块名，相当于把a中所有对象都导入，然而并不能导入以下划线开头的变量(但是以其它形式导入是完全没问题的) plus: \*同样无法导入双下划线开头的变量：\_\_x，\_\_x\_\_,**即只要是以下划线开头的变量都无法通过这种方法导入**，**但是注意若变量在类定义中，则是可以导入这个类，其使用也完全不受影响**。 **注意\_\_x\_\_并不是私有变量，在子类可以使用，也可以在类定义外通过 .\_\_x\_\_获取 （已测试）。**

\_\_x: 如图，\_\_name经过了命名重整化，无法在类定义之外通过a.name获得，如果我们直接在外部写上对a.name(类定义中没提到的变量在其外部直接写是没问题的)的赋值语句就相当于增加了另一个变量，不会对它产生影响，但是我们可以通过其他方式获取到(如图中的\_\_Test\_\_name)



03.04-再议封装、继承、多态(没东西，浪费时间)

04方法解析顺序表-06私有属性和名字重整、魔法属性和方法、上下文管理器 未看

**-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\***

**12-关于git和github**

**\*\* git是做版本控制的，目前可能接触不多或者接触不到**

**创建一个版本库：**

创建文件夹之后用 git init命令创建一个空仓库，之后git会借助这个空仓库来管理这个目录下的代码， git add code.txt + git commit -m “说明信息” 来 创建版本 git log可以查看创建日志 (先登录，再操作) **\* 注意-m只是说明信息，两次提交的版本说明信息可以相同，但是最后提交的版本序列号是肯定不同的.**

**版本回退与撤销回退**：

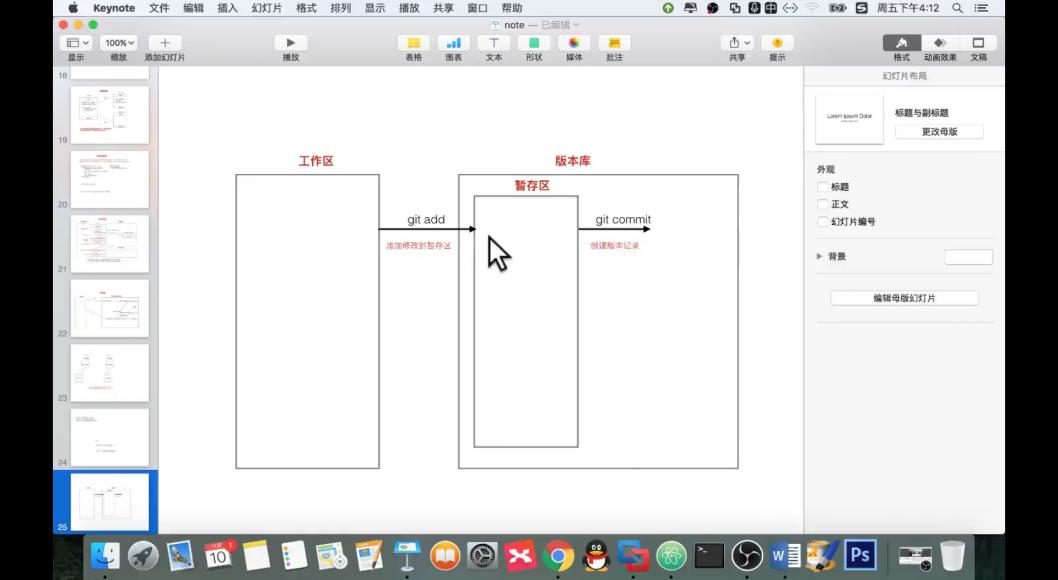
git reset --hard HEAD^ 退回到上一级版本(HEAD^^是上上一级版本依此类推， 也可以用HEAD~2表示法，代表上上一个版本)

想撤销回退，e.g.撤销版本2到1的回退，若知道版本2的序列号仍然可通过：git reset --hard 序列号实现 此外，git reflog可以查看之前的操作信息，其中也包含序列号信息，所以此命令也可以用于撤销版本回退

**暂存区**

工作区有一个隐藏目录.git, 这个不属于工作区，而是git的版本库，里面存了很多东西，其中最重要的就是称为stage(或称index)的暂存区 ，git add是添加修改到暂存区，git commit是创建版本记录 可以add多次之后一次性commit: git add code1.txt code2.txt

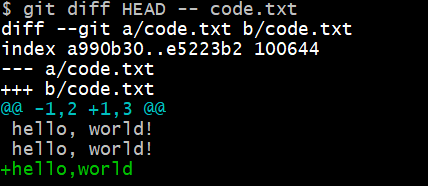
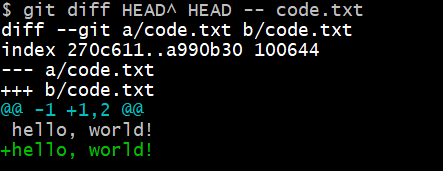
+ git commit 将缓存区的所有修改提交到分支。 如下图：



**撤销工作区改动**：git checkout -- file (**这里的改动实际上是相对于之前的add动作而言的，e.g. 新建一个code.txt被add后再被summit作为version1, 之后修改code.txt再add但未summit,此时输入git status显示处于to be commited状态，之后，修改code.txt为最初始的状态，git status发现处于修改状态 我们这里的checkout就是回到add时的状态**）

**撤销暂存区的改动**：git reset HEAD file 将暂存区的修改撤销掉(就是暂存区的文件拿回到工作区)，重新放回工作区 ，之后再用git checkout -- file将其彻底还原.

**对比文件的不同**：1.对比工作区和版本库中的文件(left) 2对比两个版本中的文件(right)

**删除文件**： 可以通过之前提到的撤销工作区的改动来实现对删除文件的恢复。

**\* 新建一个文件但未add,那么删除此文件将无法恢复，因为没有添加到缓存区经历的file仓库不会跟踪。**

**关于分支：**

\*\* 上述的HEAD实际上指向master(主分支)，master指向提交的版本。每次提交，master分支都会变长。

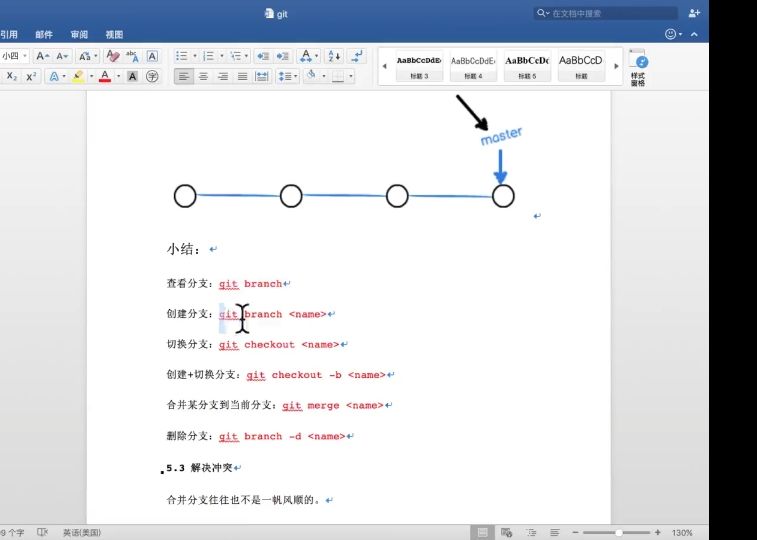
当我们创建新的分支，例如dev时，git新建了一个指针叫dev, 指向master相同的提交，再把HEAD指向dev,就表示当前分支在dev上 (**同一个文件在不同分支中当然可以不相同**)

关于分支的命令见下图：**git branch** (查看分支) **git branch** 分支名(新建分支)

**git checkout** 分支名(切换分支) **git checkout -b** 分支名(新建分支并切换到此分支)

**git merge** 分支名(合并分支) **git branch -d** 分支名(删除分支)

Plus: **git log --pretty=oneline可以将log精简显示，git log --graph --pretty=oneline可以显示出分支图.**



**在两个分支上都有新的提交并且编辑的时同一个文件那么merge的时候会起冲突而失败，需要手动解决后再add,commit.**

**02-github分支管理/03-git分支管理策略未看，暂时不看意义不大**

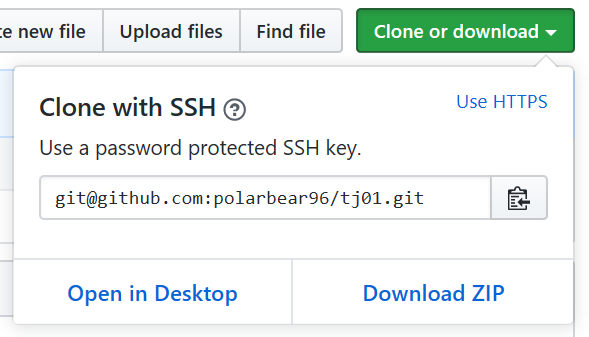
**03-Github的使用**

，**git clone 项目地址**，

**1.将项目克隆到本地**，若克隆失败添加以下两个命令后载克隆。

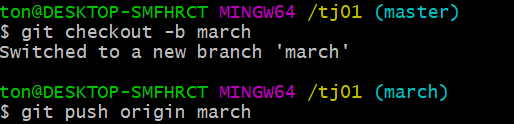
**（1）eval “$ssh-aqent -s” （2）ssh-add**

项目地址指的是：

(注意不要选Use HTTPS)

1. **远程推送git push origin 本地分支名)**

e.g:



创建一个本地分支march并切换到它，之后远程push到github



（效果）

\*\* 这是在之前的clone的基础上操作的，操作的是clone下的文件夹，否则这里的origin将无法使用，需先定义origin是什么：e.g:

这里切换到别的文件夹，仍想上传skt1分支到之前的仓库，需要向截图中一样指定origin指代的仓库，之后再push.

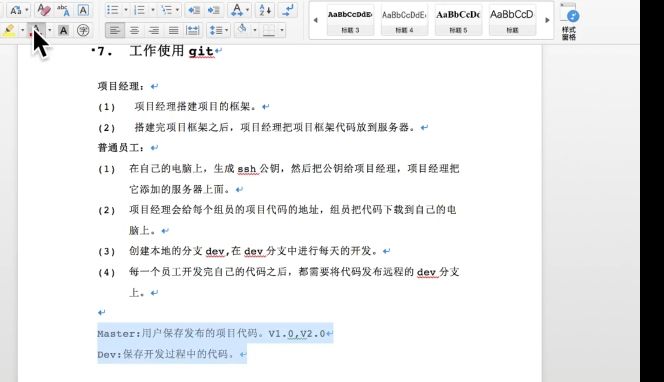
**3.拉取代码**

**git pull origin 分支名** 将远程分支上的代码下载并合并到本地所在分支，和push类似

**4.跟踪远程分支**

**git branch --set-upstream-to=origin/本地分支名 远程分支名** 让本地分支跟踪远程分支

\*\*\*工作中git的运用(了解一下)：



**观看完毕**

**-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\***

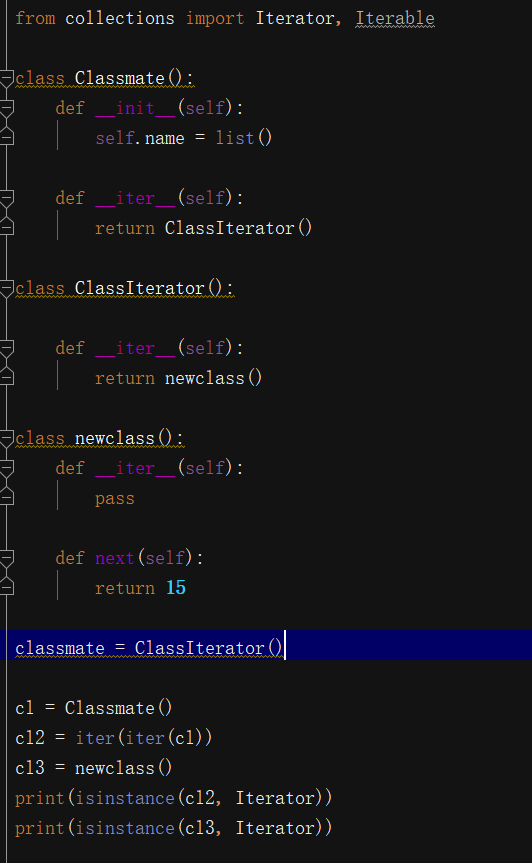
PYTHON

协程

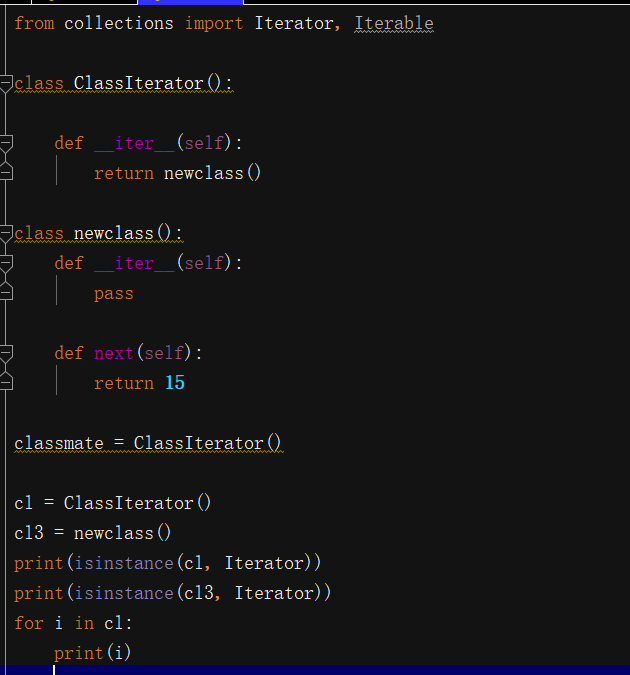
1. 迭代器

只要一个类有\_\_iter\_\_方法那么它就是可迭代对象(即isinstance(xxx, Iterable)的返回值为True)

如果一个类既有\_\_iter\_\_ 也有\_\_next\_\_方法，那么他就是迭代器(isinstance(xxx, Iterator)的返回值为True)。这两个定义非常简单。

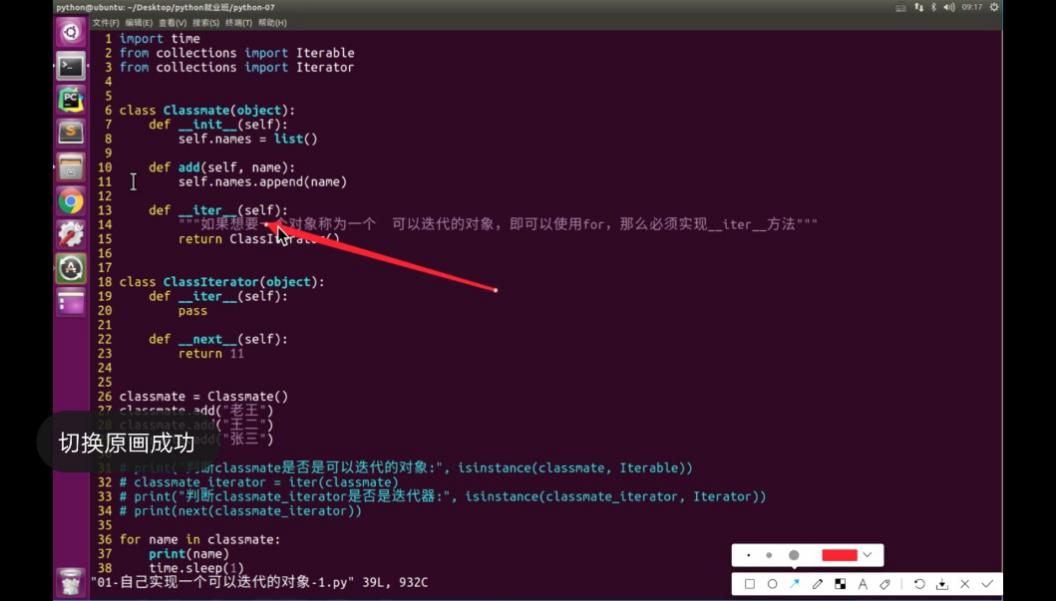


上图的运行结果会输出两个True. Iter(XXX)可以调用XXX这个实例对象的\_\_iter\_\_方法，这样cl3和cl2是迭代器咯。

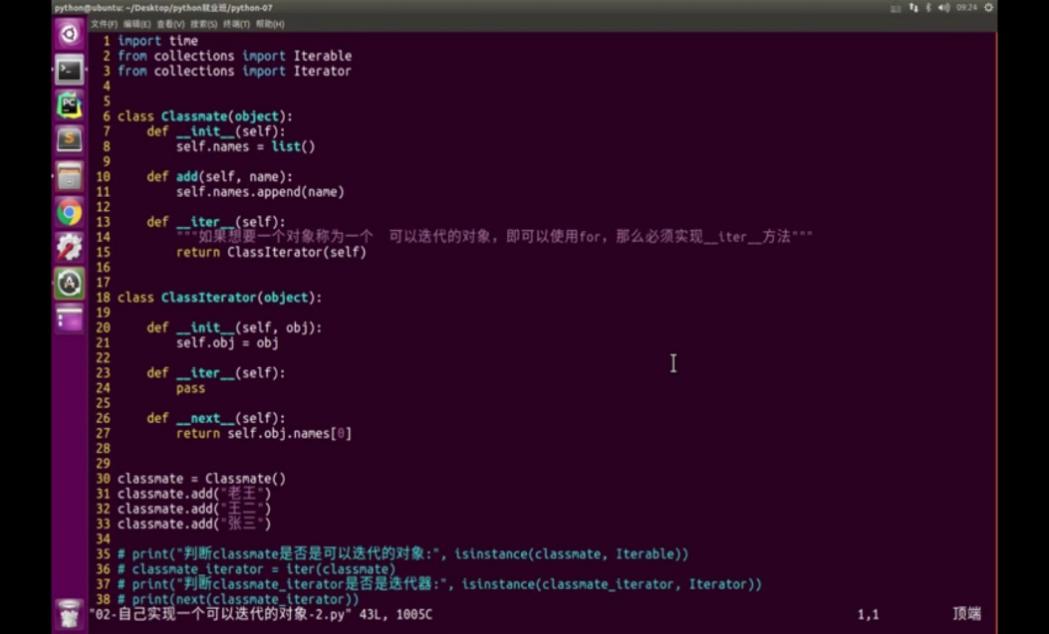


但是需要注意的，如上图，for循环作用于cl是没问题的，若改为cl3则会抛出错误。

如果让实例可以被for遍历的话那么1.这个类必须有\_\_iter\_\_方法并且 2.这个类的\_\_iter\_\_方法返回一个实例，该实例对应的类定义里面既有\_\_iter\_\_方法也有\_\_next\_\_方法。（其他情况一律不行）



如上图，for循环会不断的返回11.



我们如果想要把老王、王二、张三被for循环遍历到，就像上面这样写，让两个类勾连在一起（了解一下）。

以上理解起来比较绕，目前可以不用深究，耗时而无甚意义

协程以下部分略（不适合看）

**3 web服务器**

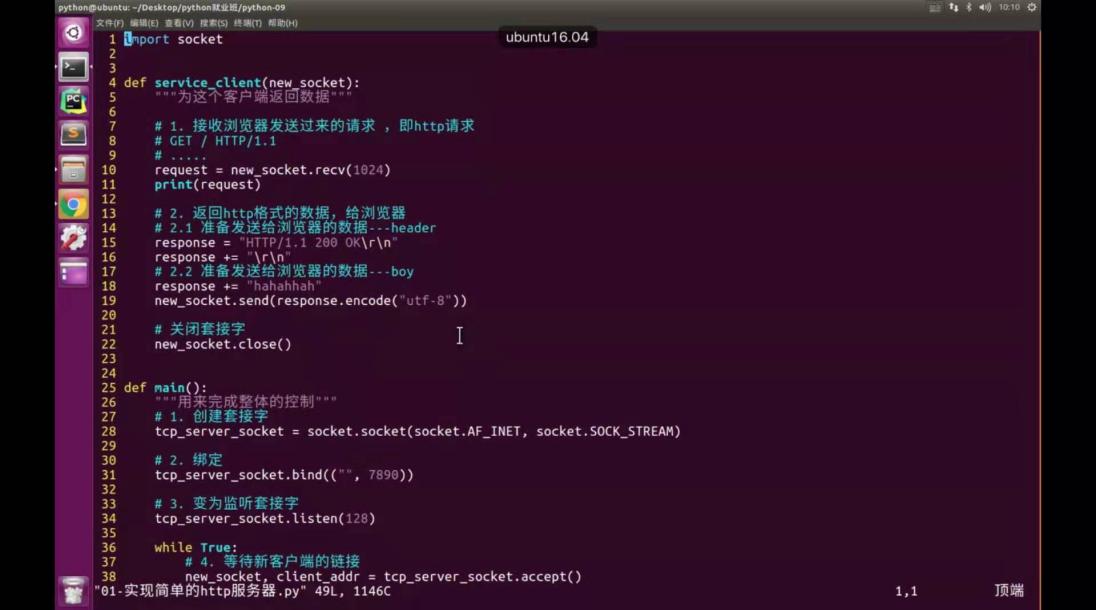
1. **http协议(基于TCP协议的超文本传输协议)**
2. 此阶段的基础知识（无意义）

http协议是对于浏览器和服务器（网站）之间的协议

1. http协议的通俗讲解（我不知道该记录什么）
2. 通过网络调试助手充当http服务器来验证http协议（同02，不知该记什么）

**03-简单web服务器实现**

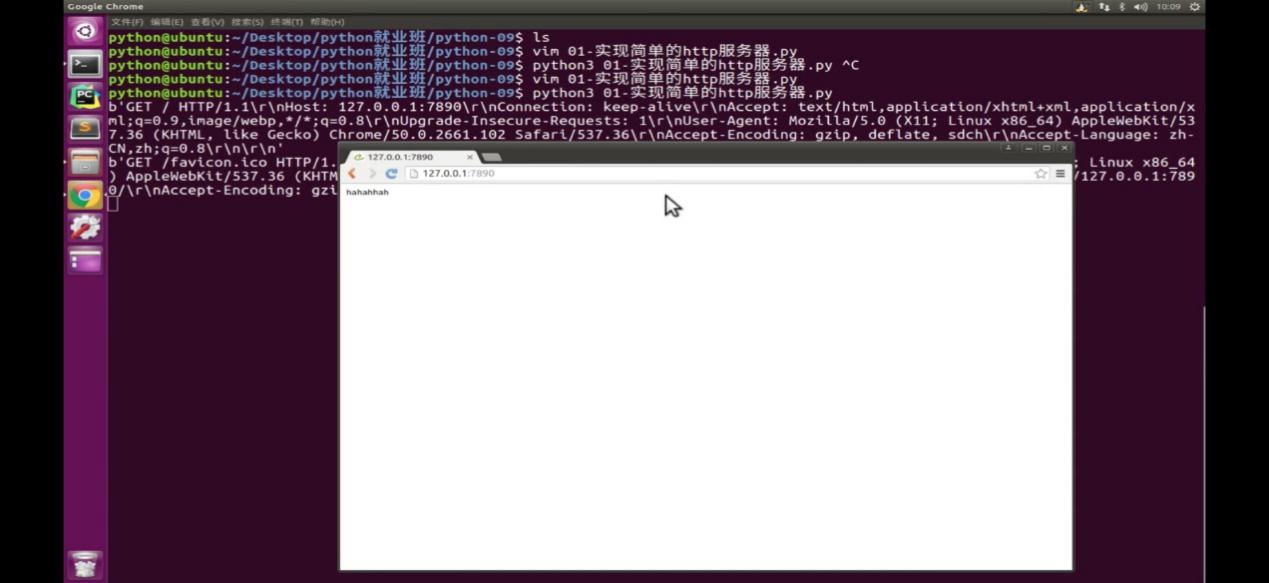
1. 返回固定页面的http服务器

****



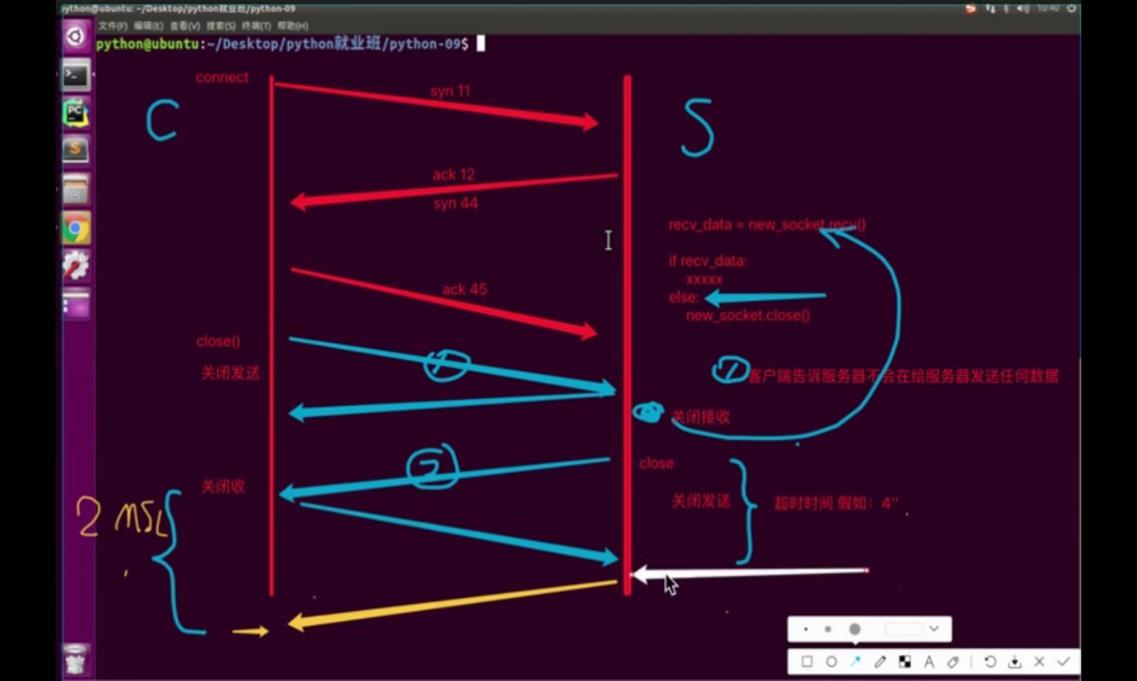
上图是用python写的http服务器，用chrome浏览器作为客户端给服务器发送数据，在代码中将其打印出来并回送给浏览器数据。

首先运行服务器，在chorme中输入服务器ip和port建立和服务器的连接，建立之后服务器将立刻收到消息并且将其打印出来：



在实际中这一串信息就是告诉服务器一些重要信息比如该浏览器的版本和遵从的HTTP协议之类，然后服务器会根据协议发送浏览器可以解析的message（注意换行符是\r\n才会被浏览器解析出来。）。

1. tcp3次握手、4次挥手

****

上面第一张图展示了TCP的三次握手和四次挥手：客户端连接服务器时对应着三次握手；二者的连接断开对应着四次挥手。 三次握手：1客户端向服务器发送请求，2服务器应答客户端的请求，同时向客户端发送请求 3 客户端应答服务器的请求 （syn表示请求，ack表示应答） 四次握手有些不好理解，知道一下，客户端和服务器关闭套接字时有超时时间，并且先发送关闭请求的需要等待较长时间才能重复使用自己的端口，所以一般都是客户端先关闭因为相比于服务器客户端是不绑定端口的（随机），关闭后再打开端口会变化，所以不受影响。（图中四次挥手那里的箭头是操作系统发送的，目的是关闭套接字，即使套接字的接发功能被关闭了对此也不影响。）

1. tcp三次握手、四次挥手强调（对02重点内容的复述）

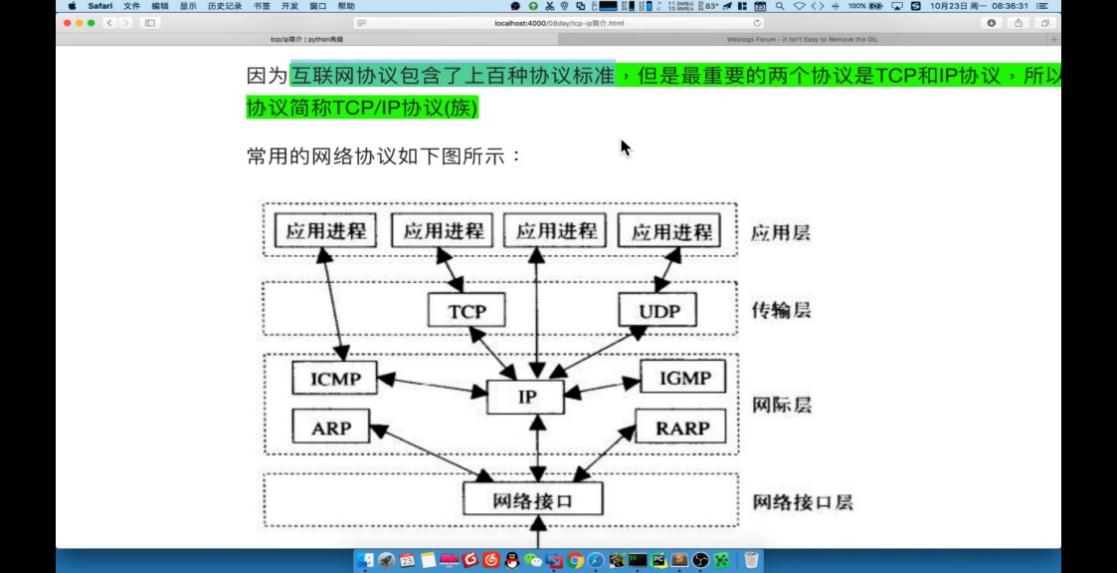
tips: 关于tcp协议不要深挖，差不多理解就行了

04/05（案例-待看）

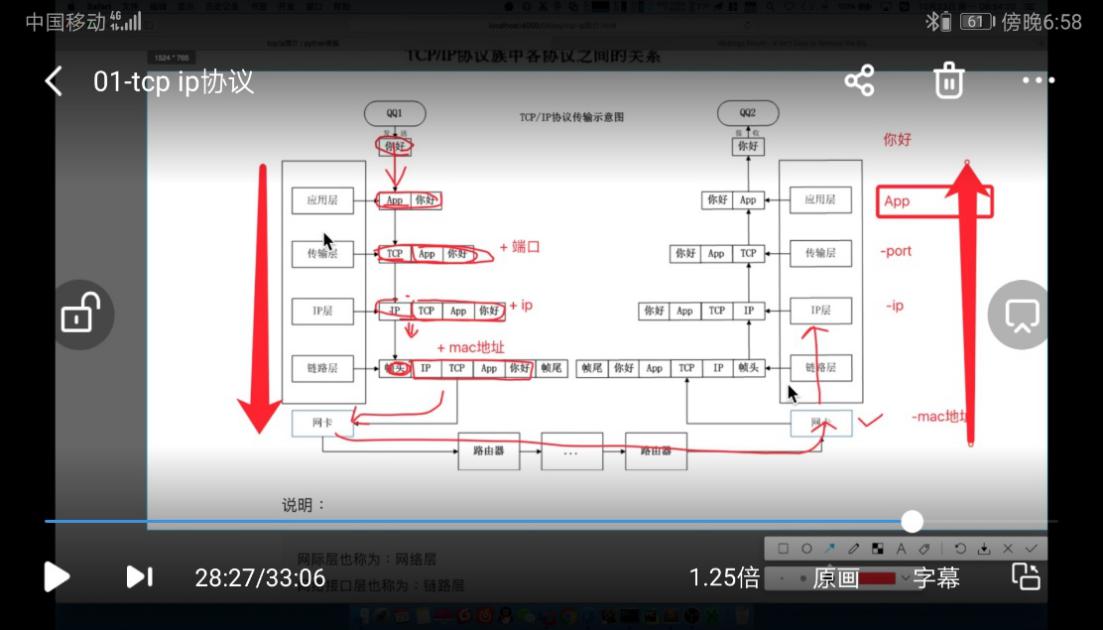
**04-（未看）**

**05-网络通信**

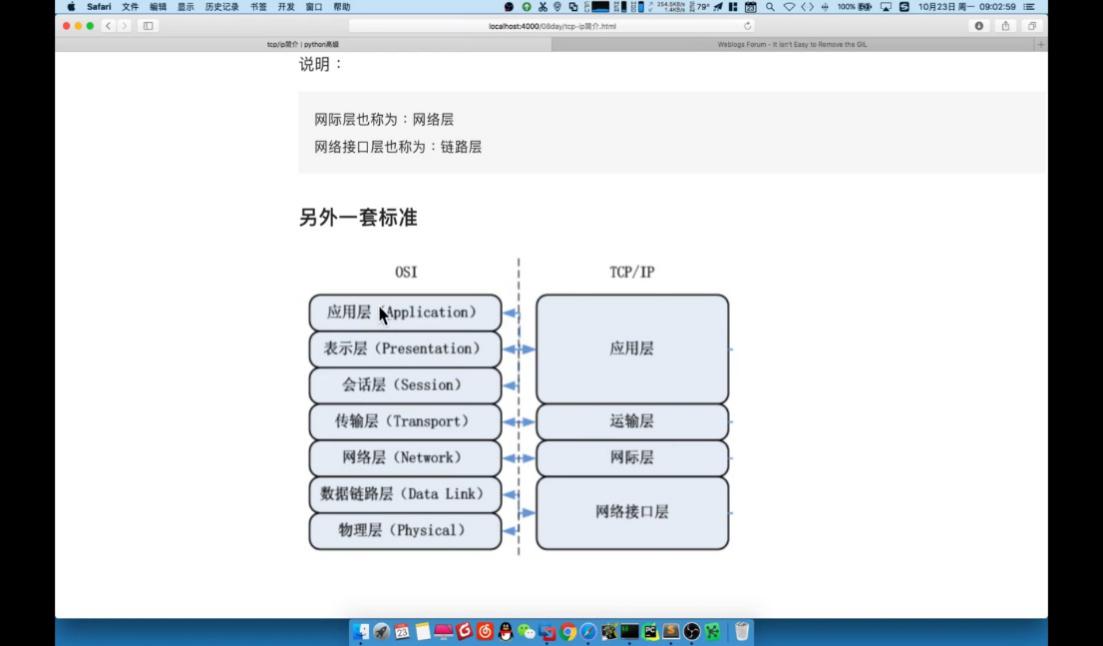
1. TCP ip协议



首先TCP ip协议是一个协议族，如上图，包含很多协议，只是核心的部分是TCP 协议和IP协议多以称之为TCP ip协议。其中网际层又称网络层，网络接口层又称链路层。HTTP其实只是应用(浏览器)层的协议，不同的应用比如QQ,微信，迅雷等都有自己的应用层协议，级别和HTTP等同。



上图是网络中两台电脑的QQ发送消息的全过程：在传到网络的过程中每往上传一层都会加一个东西，之后从网络上到目的ip的过程中每往下走一层都会丢下东西，直至只剩下原始信息给目的ip。 Mac地址是标志设备的序列号，手机里贴的入网许可就是。



OSI模型是个理论模型，可以说没什么实际意义。

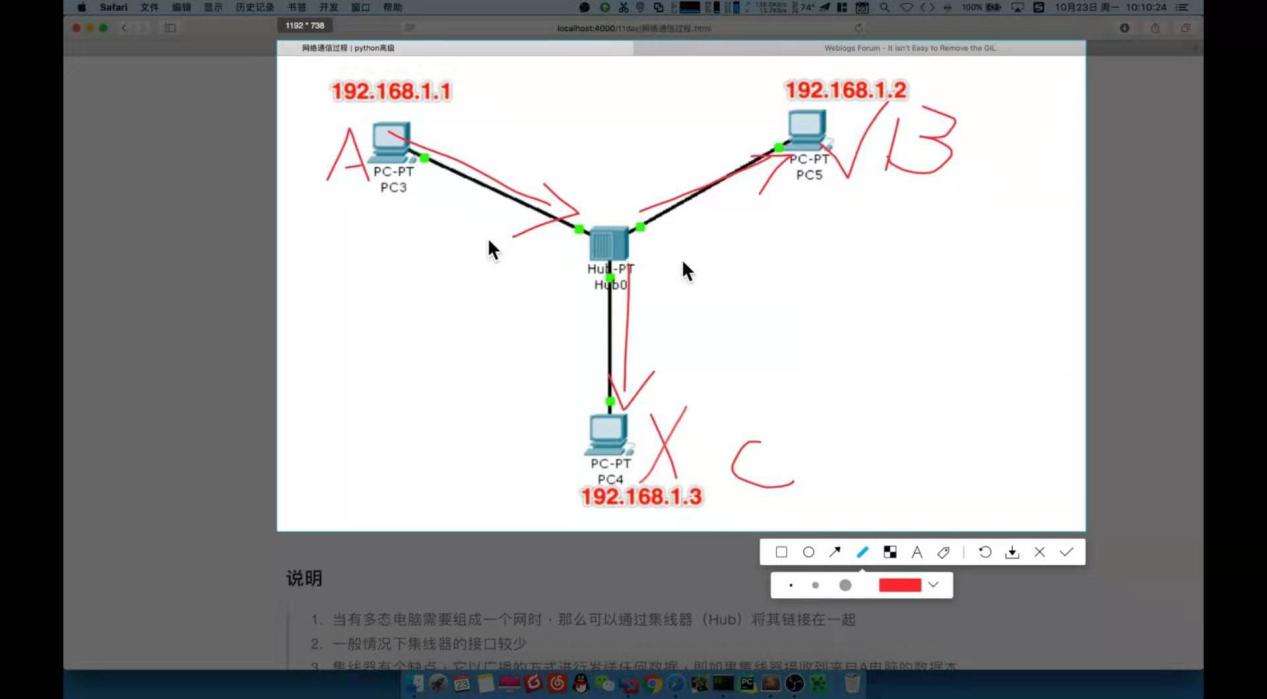
02/03（略）

04-2台电脑通信、网络掩码（简单，略）

05-集线器(hub)、交换机组网(switch)、arp获取mac地址等

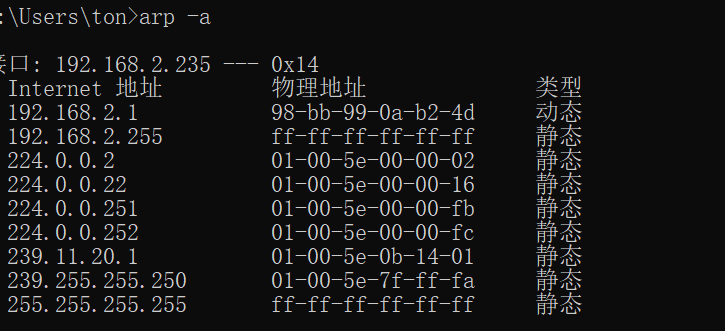
网络上的集线器基本已经被交换机取代了，但是usb的hub(usb扩充口)还是很常见的。

hub的缺点就是收到数据后只能以广播的形式发送走：

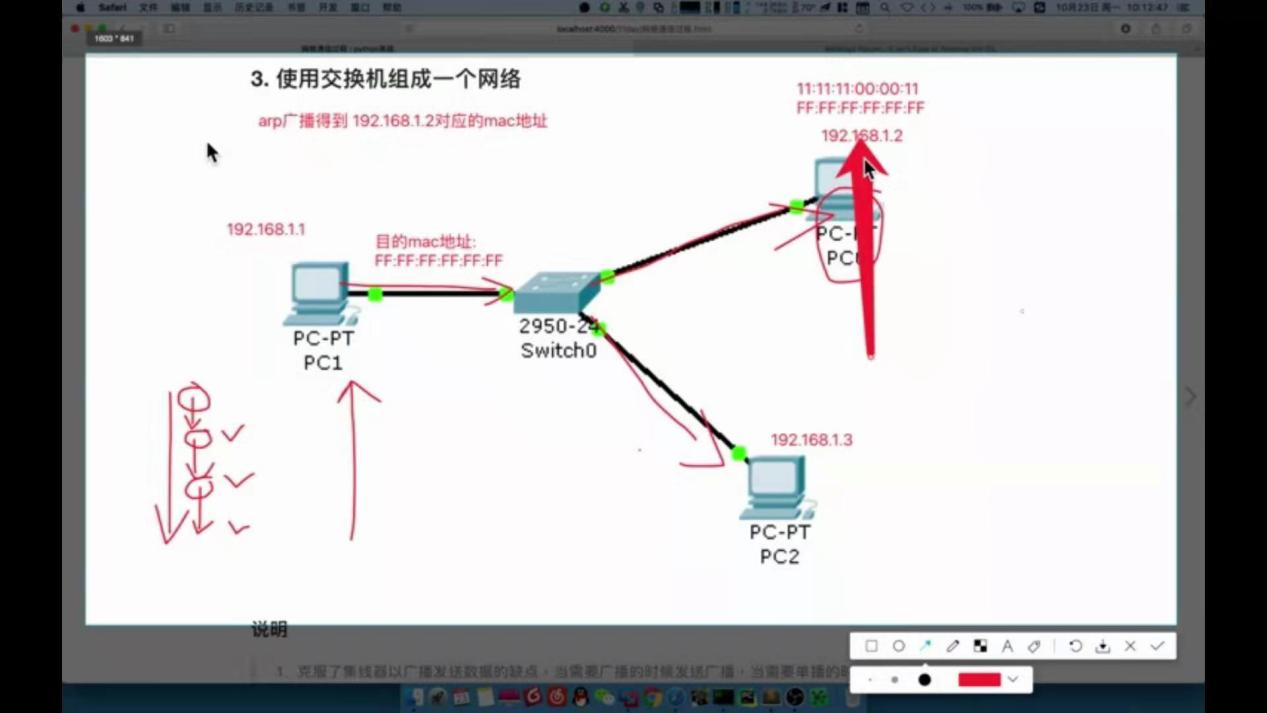


这样在A给B发送数据的时候到达C的路径也被占用了，这段时间C无法进行通信；如果网络中电脑数量增加的话整个网络的传输效率会进一步降低。

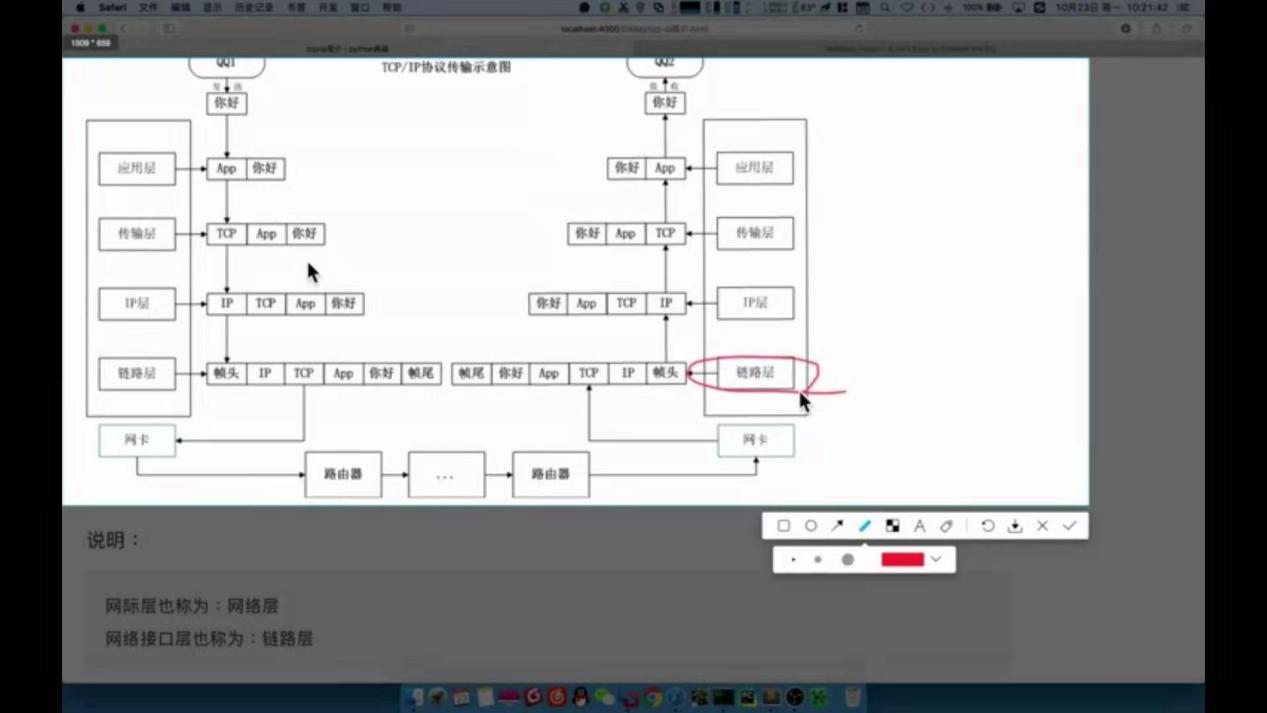
关于交换机：



以上是电脑里缓存的别人的地址（物理地址就是mac地址）。两台电脑通信会先在缓存区查找是否有对方地址。

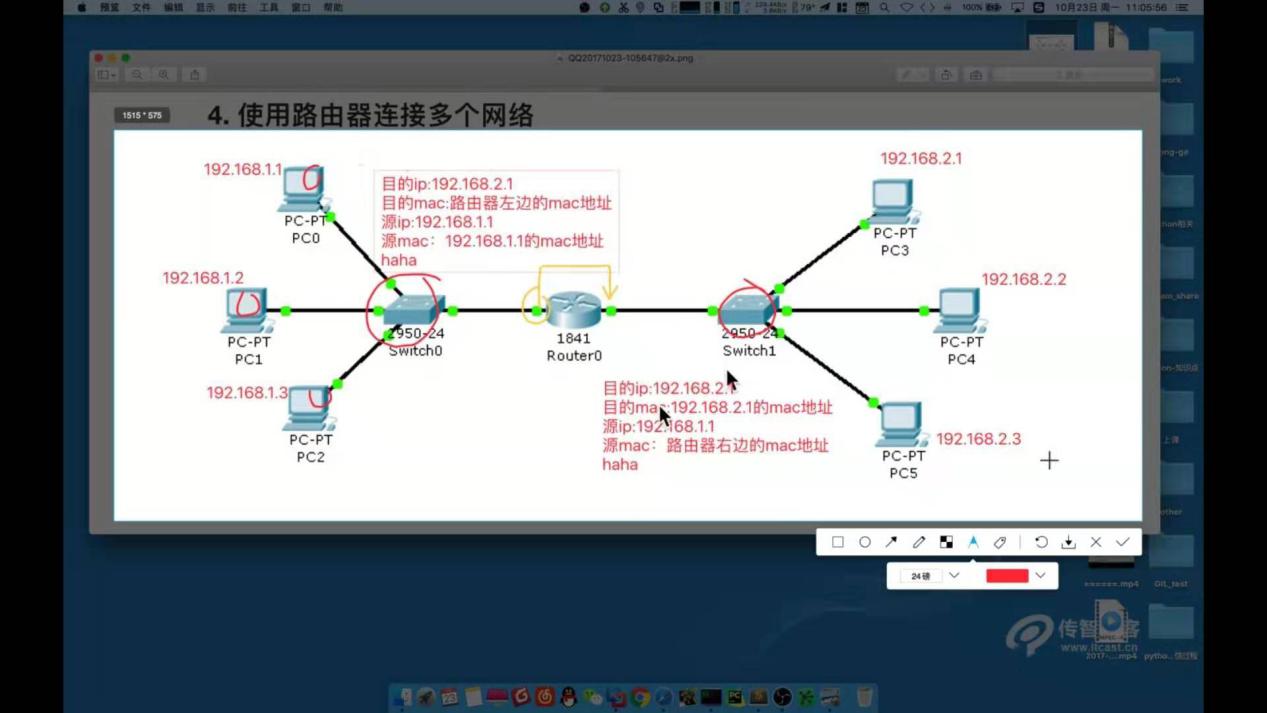


如果两台电脑第一次通信（电脑缓存区没有对方地址），则会通过ARP协议获得对方的MAC。图中192.168.1.1的电脑和192.168.1.2通信，MAC地址填写六组FF来进行广播（所有MAC都能收到），收到后192.168.1.2会回送信息（里面包含其MAC地址），之后二者通信直接单播就可以了。



（这张图插入一下，比较重要）

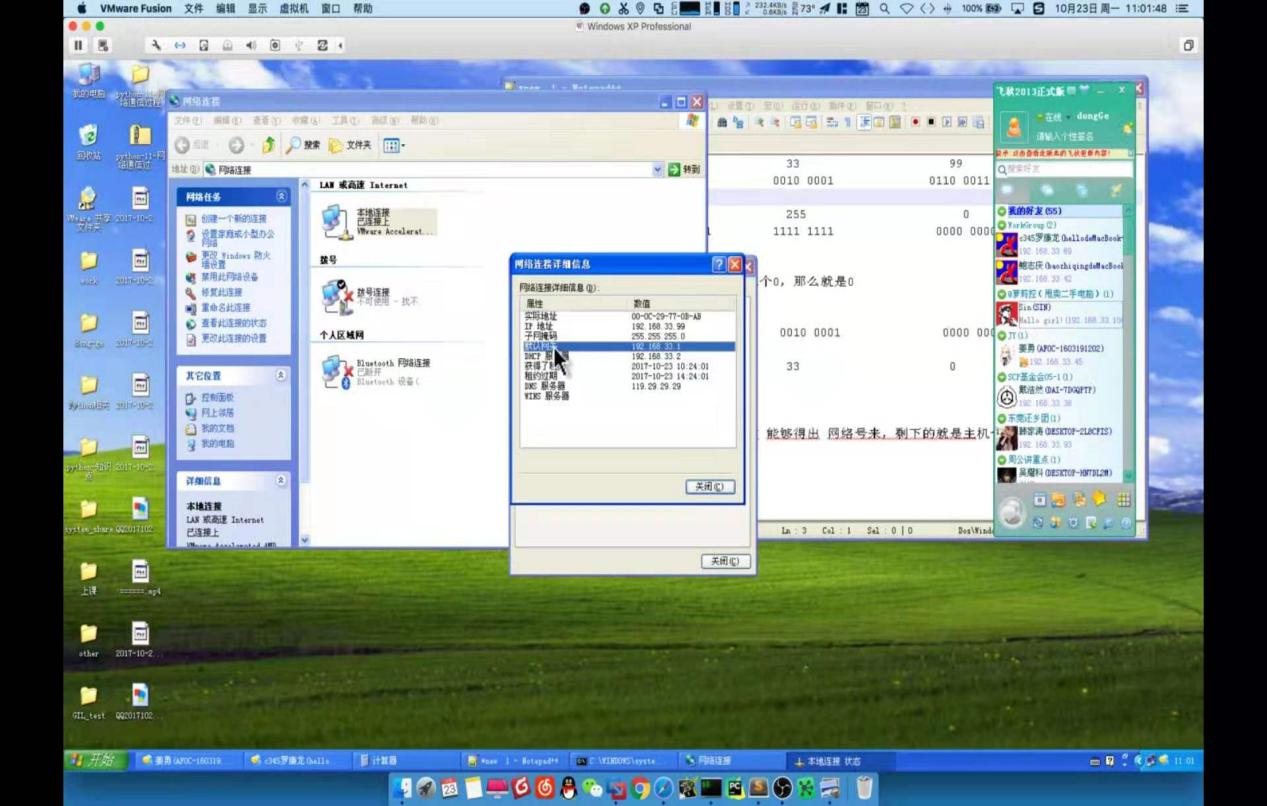
06-路由器链接多个网络、默认网关



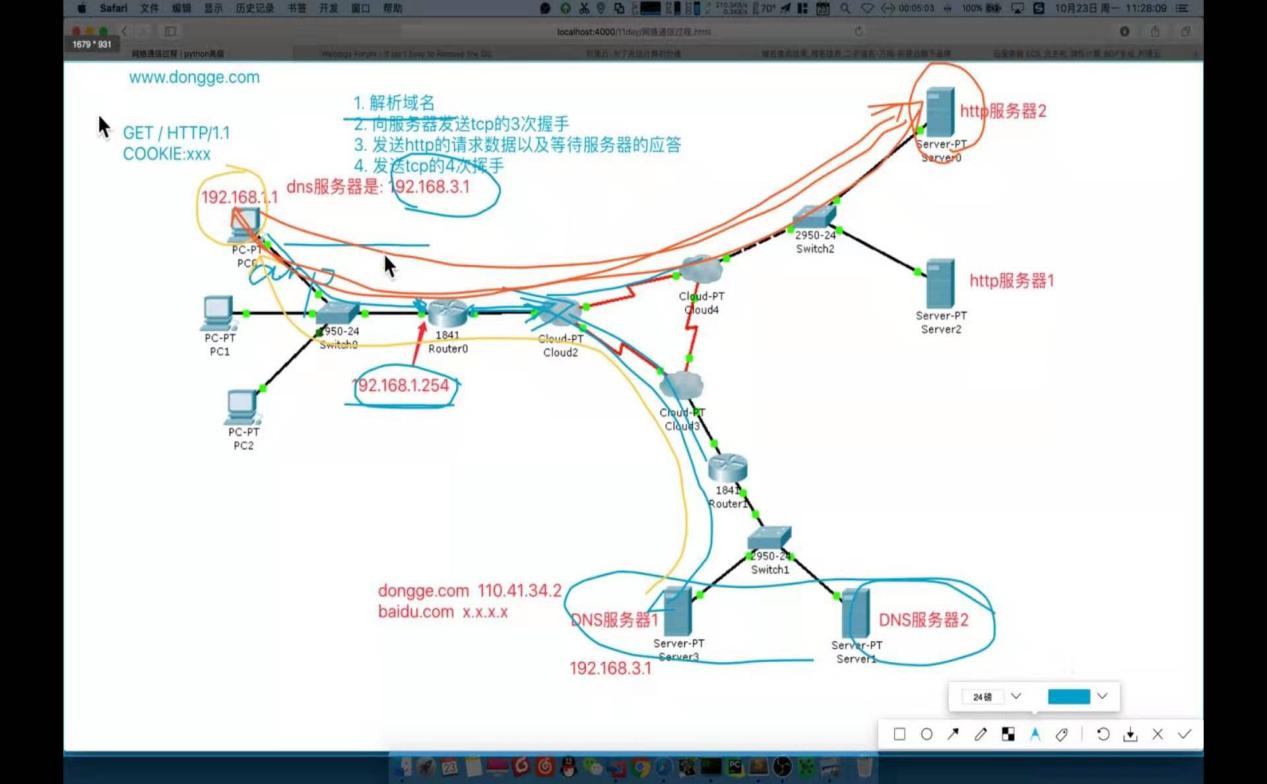
路由器连接两个子网络，有多个mac地址。图中两个网络进行通信，整个过程中源ip和目的ip是不变的，源mac和目的mac在变化（如上图文本框所示）。

Plus: 若同一网络的两台主机进行通信，直接通过交换机即可，但如果是不同网络中的主机则必须设置默认网关（路由器就是网关的一种）。

设置默认网关：



07-浏览器访问服务器的过程



上图是浏览器访问服务器的流程图，电脑通过向网关向DNS服务器(电脑有默认的DNS服务器地址) 发送http服务器的域名，DNS服务器将域名解析为IP地址发送回电脑，这样电脑便可以连接到http服务器。之后再进行TCP三次握手... ...