**多任务**

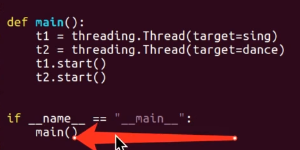
并行：多任务执行

并发：假的多任务，还是有一定的时间差

线程：创建一个主线程，主线程可以调用其他的子线程将这些子线程执行，并且能够同时调用。

Thread 创建一个线程，给target传递的参数就是子线程

多个实例对象进行调用目标函数，那么就实现了多任务。

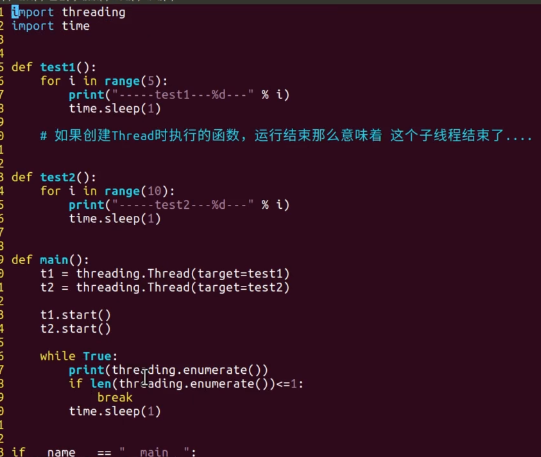


主线程会一直在等待相应的子线程完成任务，完成之后，主线程会关闭，程序结束。

在vim模式中w 跳过一个单词，B返回一个单词。

线程的执行是没有先后顺序的，没有先调用和后调用的固定顺序，但是主线程一定是最后一个执行结束的。

但是如果先要让一个先执行，那么可以将一个延时执行，另外一个后执行，休眠一段时间。



子线程从开始进行调用的时候，start的时候执行，程序执行完就结束掉线程。

使用第三方库：threading：

第三方库用api：

Threading.Enumerate 可以查看当前的线程的信息。

创建线程通过对thread类的继承完成：

一定要在其中定义run方法，继承的方法



**二.继承一种类，类里面是什么自动执行什么。**

**一定要是run方法才可以，上面就是相应的线程方法。**

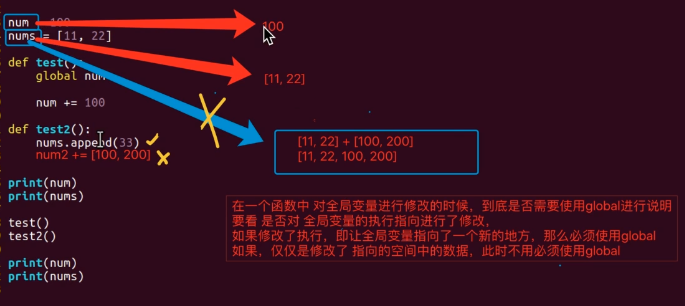
**将所有的方法放在一个类里面，start会自动调用run方法，run调用其他的方法。**

多个线程之间共享全局变量：

在局部变量变成全局变量，

能够修改里面的数据可以不使用global,

如果使用一个新的变量那么需要使用global，对全局变量的指向进行了修改，指向一个新的地方不需要使用global。



需要判断数据是否是可变的，可变的才能修改数据。

如果做了多线程程序，子线程和子线程之间共享全局变量。

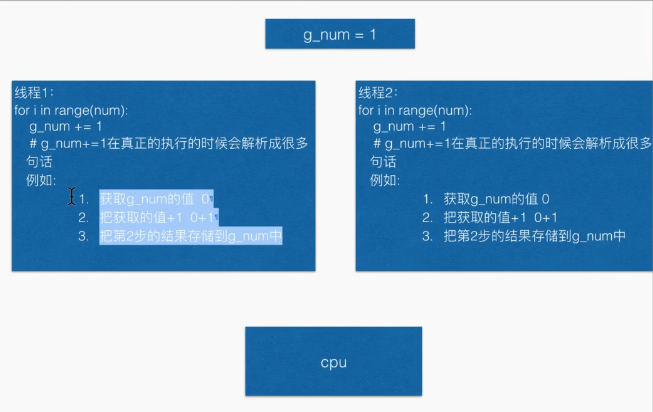
Args 和 kwarg -------------arg传递元组

target指定将来这个线程执行什么代码，args指定将来调用函数的时候，传递什么数据。

Threading.thread(target, args=())传递的参数一定是元组

**资源竞争问题：**  
共享全局变量引起的问题，如下图：

数据越大，资源越多执行起来就会出现更多的竞争情况，因为需要执行的时间。



为了解决问题：

即同步，预定先后时间运行，一个线程先做完后，第二个线程再进行

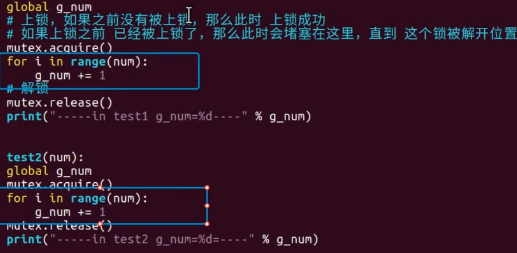
工具：**互斥锁**，多个线程同时修改一个共享数据的时候，需要使用同步控制，能够保证多个线程安全访问同一个数据。

在一个线程进行数据修改的时候，首先将线程锁定，步骤：

同一把锁使用的时候，另外的进程/线程等着。

**具体的使用：**

将资源竞争的时候使用锁，程序执行的时候使用锁确定程序的先后执行顺序，直到其中一个线程执行完毕。



预计可能出现的bug:

一个加一次，另外一个加一次，所以造成两个可能都加了总次数。

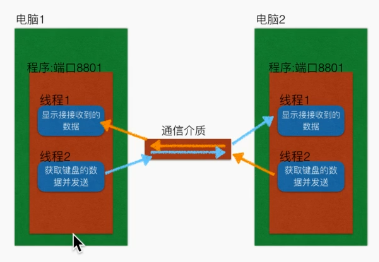
**死锁现象：**

**即两个线程之间在互相等待资源完成，使用银行家算法**

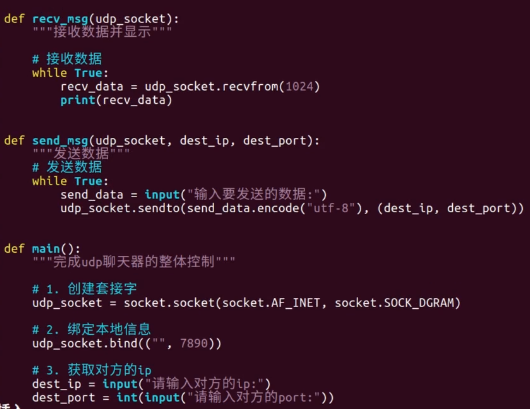
**或者添加超时时间，如果在一定的时间内线程没有完成，那么自动进行解锁。**

**银行家算法：重点首先满足一个最容易达成的条件。**

**约定好解锁时间，即设定超时时间；**

**多任务类型的聊天器：**

一个套接字就可以完成收发功能，因为套接字一般都是全双工类型





闯将线程执行相应的功能，因为接受只需要套接字，而发送需要目标的ip 和 端口号，所以需要传递两个参数。

线程只要调用threading模块就可以了

共享相同的资源时，可能会出现资源竞争，为了解决这个问题需要使用互斥锁，但相应也会出现死锁的情况---解决方案使用超时时间进行设定解决问题；

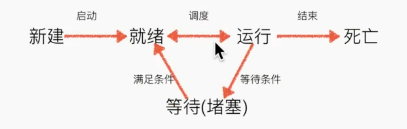
**进程：实现多任务的方案**

**二级制文件可以直接进行执行，exe形式文件。**

**程序运行的时候就变成了进程，可以多次进行执行程序得到多个进程。 静态 动态**

**进程有资格可以调用资源，即可以从电脑中调用资源。**

**进程的生命周期：**

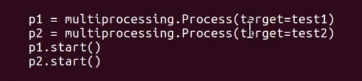


使用process完成多进程，也是为了完成多任务。

**V normal i # V是将全部代码选中**

**建立新的进程需要使用的函数：**

**导入multiprocessing 模块**



Ps 可以查看相应的进程状态， -aux 详细数据的查看。

Kill + pid（进程的代号） 可以杀死一个进程

如果在没有保存的时候直接杀死进程，造成文件内容丢失

进程数会占据相应的空间，造成内存的拥堵。

进程和线程的对比：  
进程首先出现，然后再有的线程

每个进程执行总的任务，分配给下面的线程，多个线程又来完成进程的任务，子进程又帮助父进程完成任务。

进程之间是独立的，线程的数据是共享的。

因此多任务之间的进程传递数据需要使用一个通信手段。

进程之间的通信，互相传递数据，socket就是一种进程通信。

因此需要使用queue(队列)进行进程之间的通信，如果使用文件的话，需要从硬盘中读取数据造成效率比较低，所以在内存中开一个queue是较好的一种方法。

栈和队列的区别：

栈和队列的存储方式不同，先进先出----队列 ，

先进后出栈----电梯直升型对比----另外一种数据的存储方式

使用方法：



使用put向队列中放数据，使用get从中取出数据。

get只能从中取一次，取走就不会再有数据了。

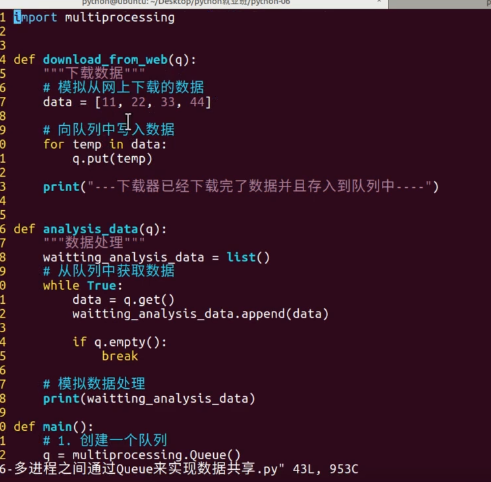
1. get\_nowwait 不等待直接取，如果没有数据程序崩溃

.fll能判断队列中的数据是不是满了。

目的：使用queue进行解耦，为了减少程序之间的耦合度。

注：D+b 从当前光标向前删除一个单词。linux编程中使用的操作。

主进程管理子进程，子进程完成任务，使用队列进行子进程之间通信



以上就是简单的在同一台电脑进行数据，可扩展。

进程池：  
重复利用进程里面的数据，解决上面只能取一次的问题

使用Multiprocessing模块使用Pool方法，导入就可以使用

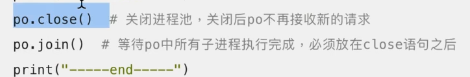
简而言之，进程池就是进程任务的优化：



最大进程是一次只能运行多少个进程

同样可以将参数传递到其中。

当任务结束后，进程池关闭：



进程池中的进程不会等待子进程完成再推出

向左按键+.不停的向左----------linux进行另外的一种操作。

案列：文件夹拷贝器

Os.\_\_file\_\_ 模块查看命令-----------Linux

\*过滤所有文件

Os.listdir (“./文件的路径”)

子进程拷贝数据放入队列，主进程将数据取出。

在进程池中使用的Que和一个进程之间的通信创建的que不同。

注意：代码需要一次次进行修改和功能强化

重要的是功能之间的配合，和文件之间的关系。

协程来实现多任务：生成器，迭代器

yield---和return的区别

迭代：是访问集合元素的一种方式，迭代器是一个可以记住遍历的位置的对象，迭代器对象从集合的第一个元素开始访问，直到所有的元素被访问结束。

在原来的基础上添加物品------简单来说就是这样

使用import iterable 这个类就是可以进行迭代



判断是否可以进行迭代

如果想要使用一个对象成为可以迭代的对象，需要使用

\_\_iter\_\_方法。

iter会引用一个函数，那个函数需要包含\_\_iter\_\_和\_\_next\_\_

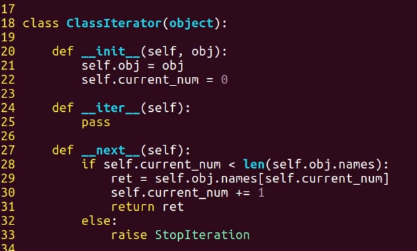
那个这个被引用的函数就是迭代器。

而这个包含iter1的函数就是所谓的可迭代对象。

**对这个类可以进行相应的改造和提升**

**可以让\_\_iter\_\_调取本身，同时将本身包含上\_\_next\_\_函数**

**那么这个类就是可迭代的，并且也成为了迭代器。**



**Raise StopIteration 会使得停止迭代**

**修改版：**

这个classmate就是一个迭代器

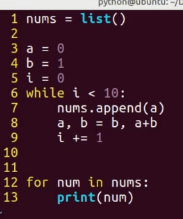
**迭代器的应用**

**保存数据，但是保证不需要太大的空间**

迭代器做的更多的是数列的存储方式

斐波那契数列使用迭代器：

1. 普通方法：



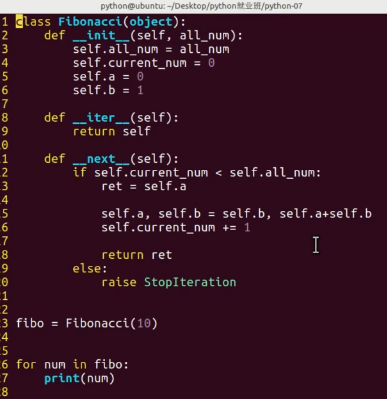
需要生成一个列表来保存数据，占据了大量的空间

修改全部变量的快速操作-------linux

1. 使用迭代器迭代的方式：  
   需要在next中设定计算方式

使用迭代器的话，不需要占据空间

生成的时候，就进行调取

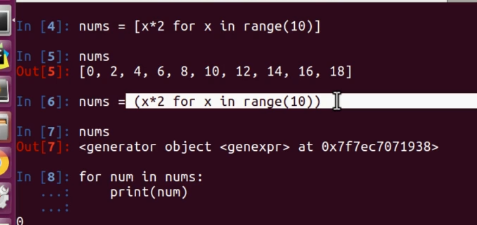


转换对象也是进行的迭代，列表和元组进行转换的时候使用的也是迭代，如列表-------元组

从列表中取数值，将其保存在元组中。

生成器：一种特殊的迭代器

1. 创建方式，将列表创建一个中括号换成小括号



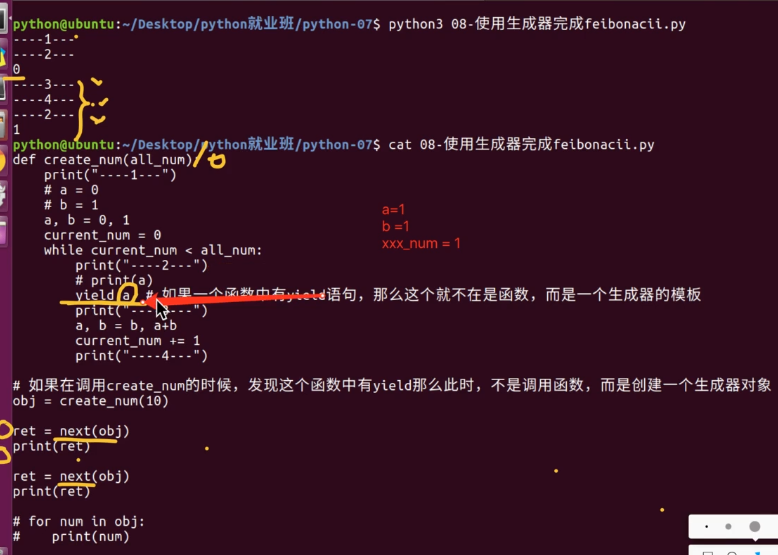
用的极少

1. 第二种创建方式：  
   加上一个yield，节省空间-------标志yield

调用函数的时候，这个就成了一个调用生成器对象

yield后面的值直接返回给函数，并且**暂停**

**下次会继续从上次暂停的地方调用。**



核心：一个函数中有yield 那么这个函数就成为了生成器

可以创建多个生成器对象，多个对象不影响。

**yield暂停，return停止重开，这是两个之间的区别。**

其中生成器的除了 next的另外一种方式



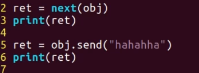
就像图中一样，next可以生成一个新的生成器

运行的一种新的方式：

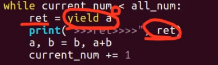
Next不能传参，但是send可以进行传参

xx.value 类创建的对象其中的return显示相应的值

使用编写技巧是相同的如下图所示：

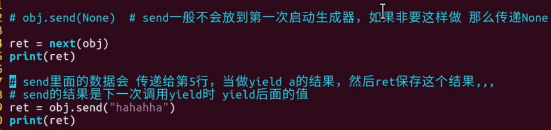
其中send(可以将相应的参数传入生成器中)

**obj是类生成的对象 send传递的值就是yield 生成的值**



可以在yield下面调用函数

使用价值：可以将新的值直接传进去。



send可以将至传入其中

使用yield 完成多任务：**重点**

yield作为一个暂停的值，

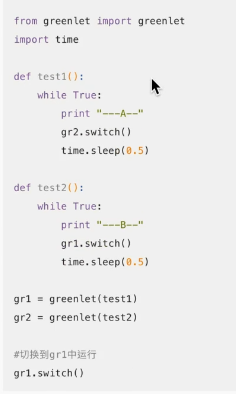
并行：一起执行

并发：假的并发，一个执行和另外一个交替执行。

进程：资源浪费 线程：占的资源较少 携程：占的资源最少

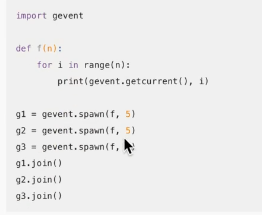
**使用grennlet gevent完成多任务**

grennlet替换yield



对yield进行封装好，不需要在函数中使用yield

gevent使用的最多，异步并发库



后面的5为传递的参数

遇到了延时操作就会自动运行下一个任务，等到任务延时完成就有回到原函数。

将线程的堵塞延时时间，用来执行另外一个线程。



所有的gevent可以更新一个time

重构

使用补丁，不需要替换自动进行替换，使用好的库

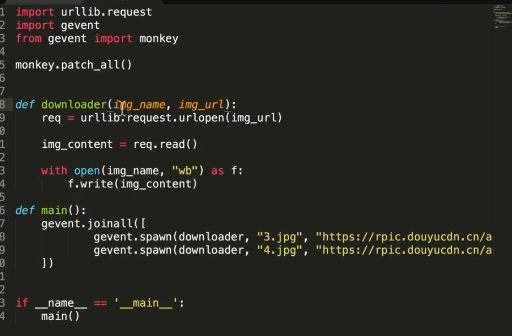


使用这个自己替换，使用gevent执行。



将创建的携程全部加入joinall中，那么就不需要join

并发下载器：



进程占的资源多，但是比较稳定

携程节省资源,不稳定；