CPU - 计算机的核心，承担计算机的所有任务。

操作系统 - 计算机的管理者，负责资源的管理和分配、任务的调度。

程序 - 运行在系统上具有某种功能的软件

进程 - 一个程序在一个数据集上的一次动态执行过程。

* 程序：用来描述进程要完成哪些功能以及如何完成
* 数据集：程序在执行过程中所需要使用的资源
* 进程控制快：记录进程的外部特征，描述进程的执行变化过程，系统可利用它来控制和管理进程，是系统感知进程存在的唯一标志
* 使用线程可以把占据长时间的程序中的任务放到后台处理
* 用户界面可以更加吸引人，如弹出的进度条
* 程序的运行速度可能加快
* 在一些等待的任务实现上可以释放一些珍贵的资源如内存占用等
* 线程可以被抢占（中断）。
* 在其他线程正在运行时，线程可以暂时搁置（也称为睡眠） -- 这就是线程的退让。

**进程(process)和线程(thread)的区别：**

* 一个程序就是一个进程，而一个程序中的多个任务则被称为线程。
* 一个线程只能属于一个进程，而一个进程可以有多个线程，但至少有一个线程。
* 同一进程的所有线程共享该进程的所有资源。但是每个线程拥有自己的栈段，栈段又叫运行时段，用来存放所有局部变量和临时变量。
* 如果把上课的过程比作进程，那么每个学生就是一个线程，他们共享教室，即线程共享进程的内存空间。每一个时刻，只能一个学生问老师问题，老师回答完毕，轮到下一个。即线程在一个时间片内占有cpu。

线程在执行过程中与进程还是有区别的。每个独立的线程有一个程序运行的入口、顺序执行序列和程序的出口。但是线程不能够独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。

每个线程都有他自己的一组**CPU寄存器**，称为线程的**上下文**，该上下文反映了线程上次运行该线程的CPU寄存器的状态。

线程之间是**随机调度**

**指令指针和堆栈指针寄存器**是线程上下文中两个最重要的寄存器，线程总是在进程得到上下文中运行的，这些地址都用于标志拥有线程的进程地址空间中的内存。

Python中使用线程有两种方式：函数或者用类来包装线程对象。

函数式：调用thread模块中的start\_new\_thread()函数来产生新线程。语法如下: thread.start\_new\_thread ( function, args[, kwargs] )

time.sleep(t)推迟调用线程的运行，可通过参数指秒数，表示进程的挂起的时间

# time.ctime()把一个按秒计算的浮点数转化为time.asctime()的形式;

time time()返回当前时间的时间戳

**线程模块**

1. threading.currentThread(): 返回当前的线程变量
2. threading.enumerate(): 返回一个包含正在运行的线程的list。
3. threading.activeCount(): 返回正在运行的线程数量

**使用Threading模块创建线程** - 直接从threading.Thread继承，然后重写\_\_init\_\_方法和run方法

threading.RLock允许在同一个线程中被多次acquire。 而Lock却不允许这种情况。如果使用RLock, 那么acquire和release必须成对出现，即调用了n次acquire，必须调用n次的release才能真正释放所占用的琐。

**协程**存在的意义：对于多线程应用，CPU通过切片的方式来切换线程间的执行，线程切换时需要耗时（保存状态，下次继续）。协程，则只使用一个线程，在一个线程中规定某个代码块执行顺序。