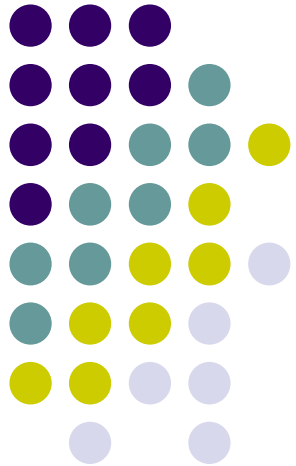
上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

第9章 模拟与并发

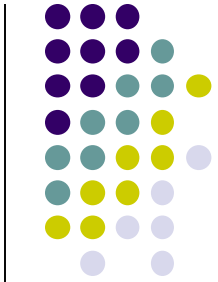
Simulation and Concurrence

申丽萍

lpshen@sjtu.edu.cn

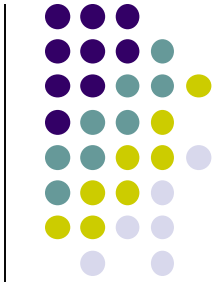


第9章 模拟与并发



- 模拟
- 设计
  - 自顶向下
  - 原型法
- 并行计算
  - 进程与线程
  - 多线程编程

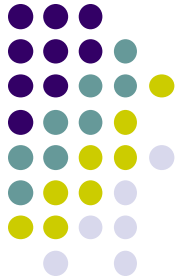
模拟



- 我们目前掌握的工具已经足以解决一些有意思的问题.
  - "有意思"是指:如果不设计实现计算机算法,该问题是很难或不可能解决的.
- 模拟:用计算机为实际问题建模,从而提供非如此不能获得的信息.
  - 这是解决实际问题的强大技术,每天都在应用:
    - 天气预报
    - 设计飞机和宇宙飞船
    - 制作电影特效
    - .....
- 例：模拟混沌现象
  - demo_simu_chaos

```
def main():
    x = input("Enter a number between 0 and 1: ")
    for i in range(10):
        x = 3.9 * x * (1 - x)
        print x
```

随机问题的模拟 - 随机数



- 计算机的计算根据指令运行，是确定的。
- 如何用确定性的计算机模拟非确定性?
  - 用函数生成随机数(实际上是伪随机数).
    - 从种子值开始,计算出一个“随机”数;
    - 如果还需要,就用上一个随机数反馈给生成函数,生成下一个随机数.
    - 这类模拟也称为Monte Carlo蒙特卡罗算法
  - Python库random提供了一些伪随机数生成函数:
    - `randrange(start,end+1,step)`:生成指定范围内的整数
    - `random()`:生成[0,1)间的一个浮点数
    - 随机数生成次数越多，其随机性越明显。

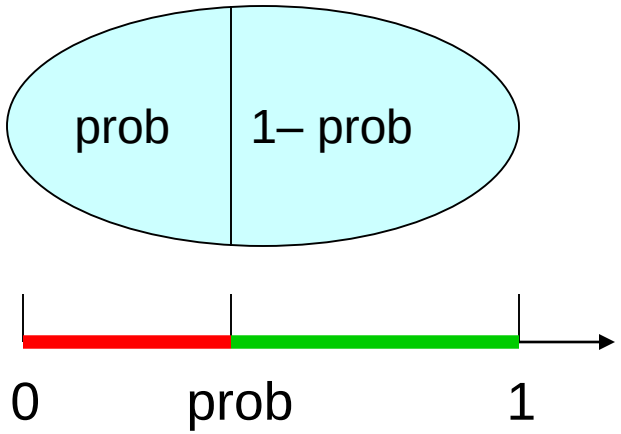
用random模拟乒乓球输赢

- 设发球方获胜概率是prob
- 程序中显然需要这样的代码:

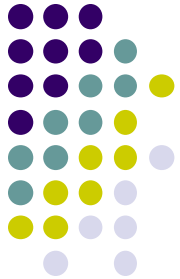
```
if 发球者胜了本回合:
    score = score + 1
```

  - 并且要使该条件为真的情况占prob
- 用random函数模拟:

```
if random() < prob:
    score = score + 1
```



一个模拟问题:乒乓球比赛

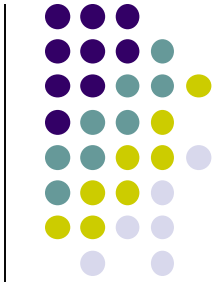


- 世界乒乓球的难题:中国队永居第一
  - 为了增强乒乓球运动的吸引力，通过制定规则来削弱中国队的绝对优势.
- 解决方法: 编程模拟乒乓球,通过模拟不同规则对高水平球员的不利影响程度。
  - 扩大乒乓球直径
  - 每一局比分从21分改为11分
  - 三局两胜 vs. 五局三胜
  - ……

程序规格说明

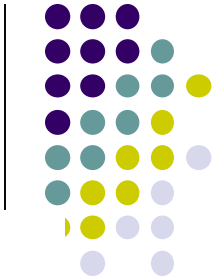
- 球技水平:用球员作为发球方时的获胜概率来模拟.
- 程序规格说明
  - 输入:
    - 两个球员的水平: 球员获胜概率p和1-p
    - 模拟比赛局数：n
    - 比赛规则：21分或11分
  - 输出: 两球员各自的获胜局数及比分

第9章 模拟与并发

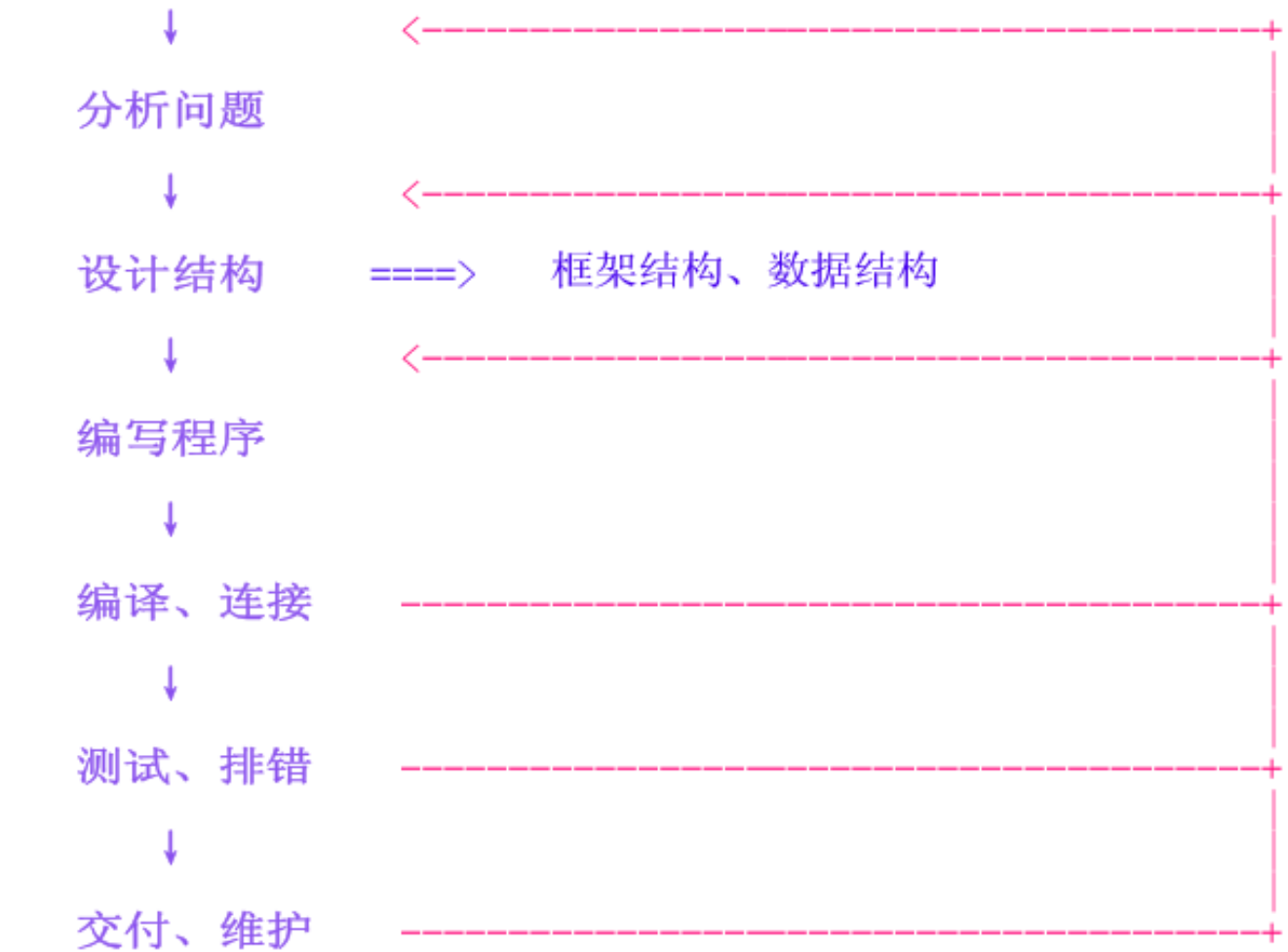


- 模拟
- 设计
  - 自顶向下
  - 原型法
- 并行计算
  - 进程与线程
  - 多线程编程

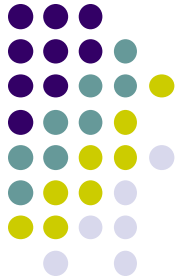
设计



程序设计的整个流程：

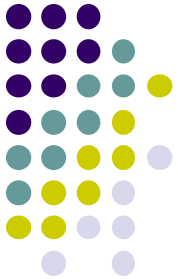


自顶向下设计



- 对复杂问题常采用自顶向下设计:
  - 将对一般问题的解决方案用若干个较小问题来表达.
  - 再对较小问题用同样的方法分解.
  - 直至小问题很容易求解.
  - 将所有小问题的解合并,就得到大问题的解.
- 抽象
  - 在设计的每一层,接口指明了需要下一层的哪些细节;其他可暂时忽略.
  - 抽象:确定某事物的重要特性并忽略其他细节的过程. 抽象是基本的设计工具.
  - 自顶向下设计的过程可视为发现有用的抽象的系统化方法.

顶层设计



- 基本算法:

  介绍程序功能

  取得输入: probA, n,rule

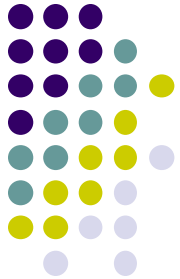
  利用probA根据rule模拟n局比赛

  输出结果报告

- 基本程序

```
def main():
    printIntro()
    prob, n,rule = getInputs()
    winsA, winsB = simNGames(n, prob,rule)
    printSummary(winsA, winsB)
```

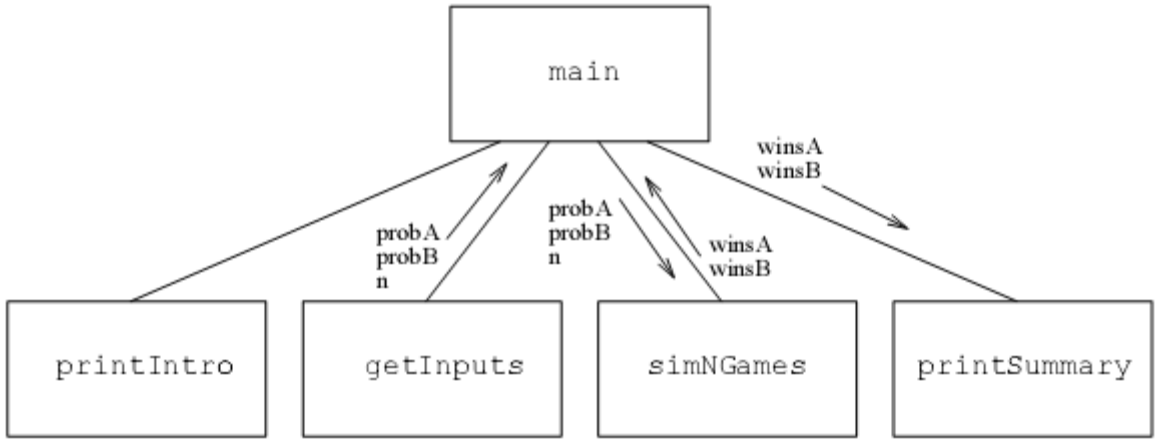
分离关注



- *Separation of Concerns* (SoC): 将计算机程序分解成不同部分,各部分功能重叠越少越好.
  - 一个关注是指程序中的一个兴趣点或焦点.
- 好处:
  - 允许多人独立开发系统的不同部分
  - 便于重用
  - 确保系统可维护性
  - 易于增加新功能
  - 使系统易理解
  - ……

结构图

- 模拟乒乓球的程序被分成了四个关注:
  - 为每个关注定义了函数的接口(*interface*)或称特征(*signature*).
    - 即函数名,参数,返回值的信息
  - 高层设计时只须关心函数的接口,而非函数的实现.
- 用结构图(或称模块层次图)表示:



第二层设计

- 设计simNGames()

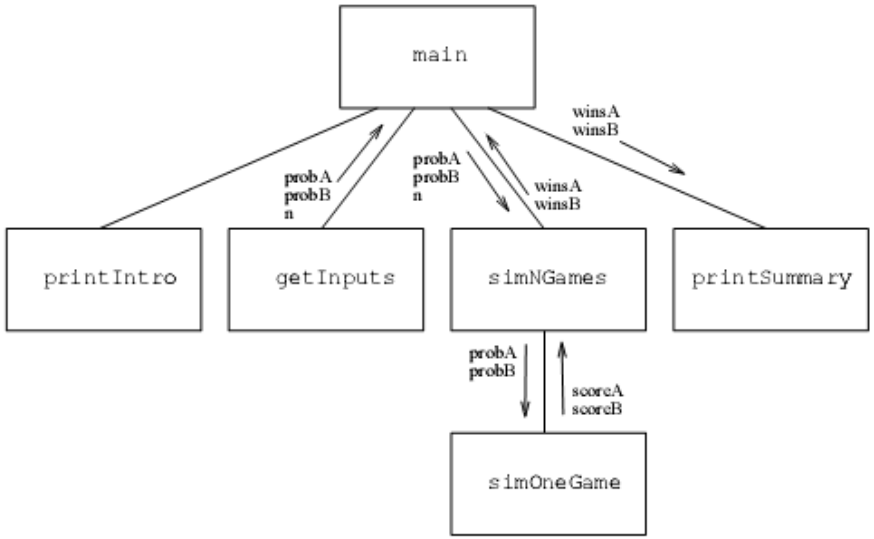
winsA和winsB初始化为0

循环n次

模拟一局

if playerA胜: winsA加1

else: winsB加1

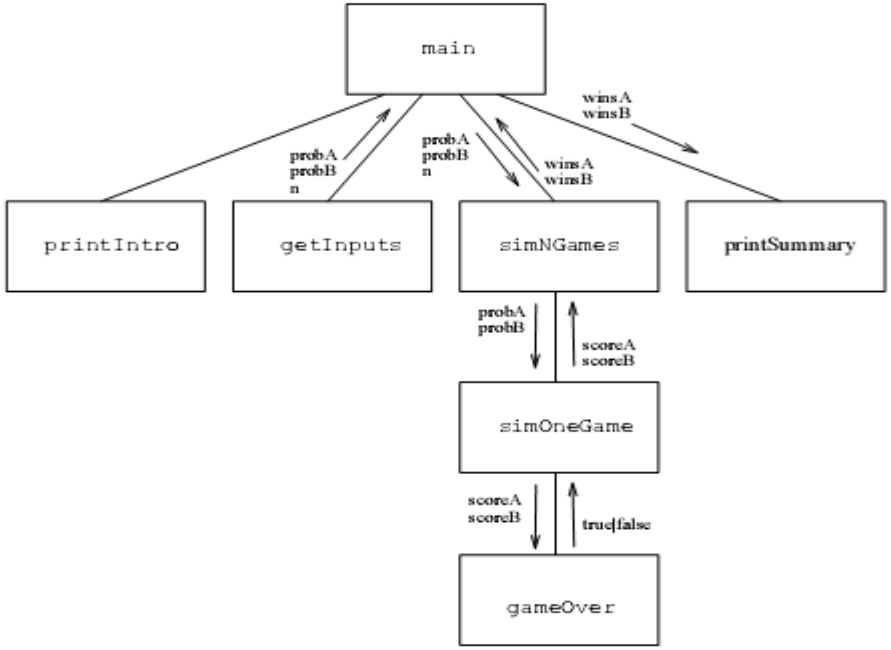



```
def simNGames(n, prob,rule):
    winsA = winsB = 0
    for i in range(n):
        scoreA, scoreB = simOneGame(prob,rule)
    if scoreA > scoreB: winsA = winsA + 1
    else: winsB = winsB + 1
    return winsA, winsB
```

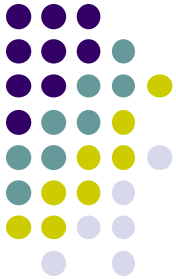
第三层设计

- 结束条件用函数gameOver

```
def simOneGame(prob, rule):
    scoreA = 0
    scoreB = 0
    while not gameOver(scoreA,
    scoreB,rule):
        if random() < probA:
            scoreA = scoreA + 1
        else:
            scoreB = scoreB + 1
    return scoreA, scoreB
```

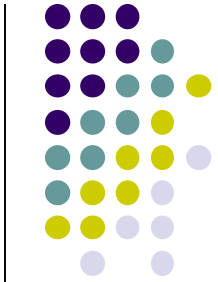


设计过程小结



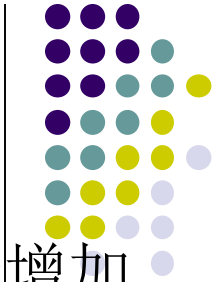
- 自顶向下,逐步求精
  - 将算法表达为一系列较小问题
  - 为每个小问题设计一个(函数)接口
  - 用各小问题的接口细化算法
  - 对各小问题重复此过程
- 自底向上实现
  - 从结构图的底层开始实现,逐级向上.
  - 每完成一个模块,进行单元测试.
  - 这也是分离关注,使debug更容易

第9章 模拟与并发



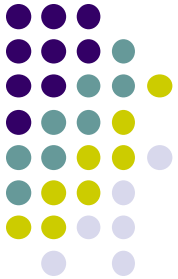
- 模拟
- 设计
  - 自顶向下
  - 原型法
- 并行计算
  - 进程与线程
  - 多线程编程

原型设计方法



- 原型法(*prototyping*):从程序的一个简单版本开始,逐步增加功能,直至完全满足程序规格.
  - 初始的简单版本称为原型(*prototype*).
- 原型技术导致螺旋式开发过程:
  - 确认用户需求
  - 原型的设计,实现,测试，向用户展示，获得反馈
  - 新功能的设计,实现,测试，向用户展示，获得反馈
  - ……直到用户满意为止
- 两种原型法
  - 产品核心原型法：核心产品
  - 快速原型法：界面设计
- 适合情况:对程序功能不熟悉,难以按自顶向下设计方法给出完整设计.

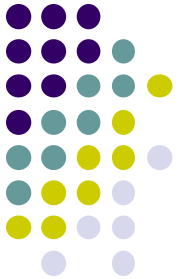
例:乒乓球模拟程序的原型



- **simOneGame()**
  - 固定水平五五开
  - 固定比赛规则：谁先得21分者胜

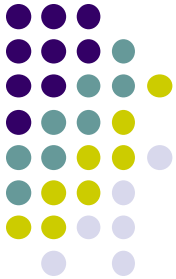
```
from random import random
def simOneGame():
    scoreA = 0
    scoreB = 0
    while scoreA!=21 and scoreB!=21:
        if random() < .5:
                scoreA = scoreA + 1
        else:
               scoreB = scoreB + 1

    print scoreA, scoreB
```

例:对原型的扩展



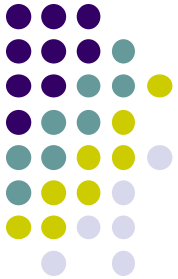
- 增加一个参数:选手的技术水平
- 增加比赛规则以判断一局的胜负
- 完成多局比赛,统计各人获胜局数
- 增加交互式输入,格式化输出

第9章 模拟与并发



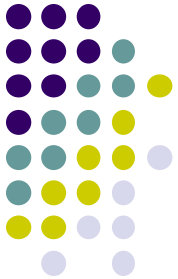
- 模拟
- 原型法
- 并行计算
  - 进程与线程
  - 多线程编程

程序的执行



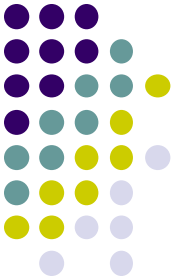
- Von Neumann体系结构:程序(指令序列)和数据都存储在内存中
- CPU根据程序计数器PC(或称指令指针IP)的内容, 取出当前指令执行,然后PC被赋予下一条要执行的指令的地址。即CPU串行执行指令。
- 虽然也存在指令级并行技术,但我们讨论的是程序级的并行.

顺序(或串行)执行



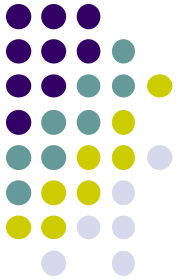
- CPU执行一个程序时总是从该程序的第一条指令开始,不间断地一直到执行到最后一条指令.
- 只有一个程序结束,才会去执行下一个程序.
- CPU每次由一个程序独占.只要前一个程序还没有结束,下一个程序就不能使用CPU.
- 缺点:系统资源的利用率不高.
  - 计算机系统中有许多资源.当一个程序在使用某个资源时,其他资源是空闲的.如果允许其他程序使用空闲资源,就能提高系统资源的利用率.
  - 例如,存储器直接访问DMA

并发执行



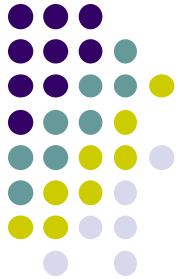
- 计算机程序的执行是由操作系统控制的.现代操作系统都支持所谓"多道程序"或"多任务",即允许多个程序"同时"执行.
- "同时":在只有一个CPU的情况下,是不可能有真正的多个程序"同时"运行的,因为CPU在任一时刻只能执行一条指令!
  - 分时使用CPU,即CPU在多个程序之间切换.
- 这种多个相互独立的程序交叉执行的方式称为并发或并行执行.
  - 函数并行(functional parallelism)
  - 数据并行(data parallelism)
- 多处理器/多核处理器上能真正并行.

进程



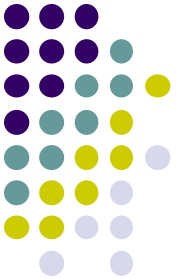
- 进程,是指程序的一次执行而形成的实体.程序一旦执行,即创建一个进程.
- 进程的构成:程序代码+进程状态信息(上下文,包括程序数据的当前值,当前执行点等)等.
- 程序与进程:不同程序的执行对应不同的进程;同一个程序多次执行也创建多个进程(相同程序代码+不同上下文).
- 多进程:
  - OS调度进程，上下文切换代价高.
  - 进程之间不共享地址空间，很难共享信息。

线程



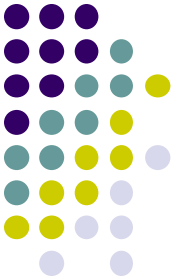
- 线程,是指程序(进程)中的一段代码,它构成程序中一个相对独立的执行流.
  - 字面意义:线程是程序内部的一个"线索”
- 线程是一个程序内部的多任务机制.
  - 一个程序中可以有多个执行"线索"
- 线程是OS调度的最小单位。
- 多线程:
  - 线程间共享地址空间（代码和上下文）
  - 切换代价小
  - 容易通信

进程 vs 线程



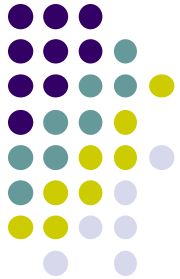
- 多个进程一般是相互独立的,而多线程是同一进程的一个多个执行流;
- 进程带有独立的状态信息,而一个进程内的多个线程则共享状态,内存和其他资源
- 进程有独立的地址空间,而同一进程的多个线程共享地址空间
- 进程间通信(IPC)较麻烦,而线程之间可通过共享内存容易地通信

多线程编程



- 线程原是OS中的概念,是系统的工具,用于系统的功能
- 线程现在已成为用户程序设计的工具
- 应用在需要并行执行的场合:
  - 科学应用:更快地计算;
  - 解决阻塞:用户输入;在长时间计算期间进行显示服务;多媒体,动画等.
- 多线程会导致竞态问题(race problem)
  - 各线程相互独立，并发执行没有确定次序，是非确定性计算模型。
  - 12306网站只剩一张票，两人同时订票，到底先给谁？

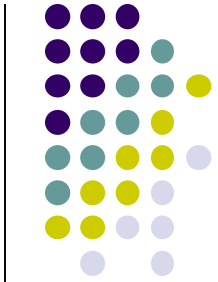
Python多线程编程 – thread模块



- 利用thread模块: import thread
- 线程的创建和启动:

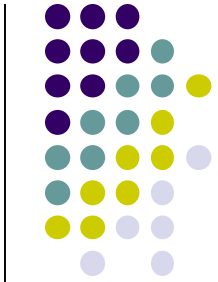
  thread.start_new_thread (*function*, *args*)
  - 创建一个新线程并立即返回;新线程启动,向函数*function*传递参量元组*args*.
- 线程执行的代码:即*function*
- 线程终止:当函数*function*返回。
- 例demo_multThread0.py和demo_multThread1.py

Python多线程编程 - threading模块



- threading模块: import threading
  - 面向对象
  - 提供Thread类来支持线程化程序设计.
- 创建线程对象后,通过调用它的start()方法来启动线程; start()在一个独立的控制流中调用run()方法.
  - 创建线程,并向Thread类的构造器传递线程任务代码
    t = Thread(target=func,args=(...))
  - 然后启动线程　t.start()
- 有两种方法指定线程的活动:
  - 向Thread类的构造器传递一个可调用对象
    - 例demo_multThread2.py和demo_multThread3.py
  - 在Thread类的新子类中重定义run()方法.

Python多线程编程 - threading模块



- 定义Thread的子类并重定义run()

```
class myThread(threading.Thread):
    def __init__(self, threadName, delay):
        threading.Thread.__init__(self)
        ......
    def run(self): ......
```

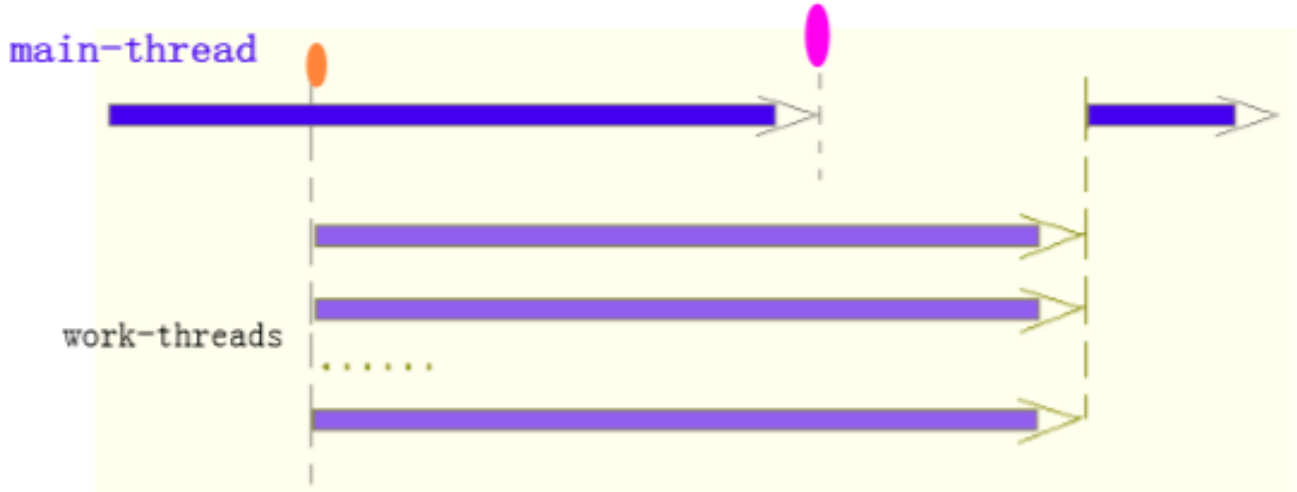
- 创建并启动线程

```
t = myThread(...)
t.start()
```

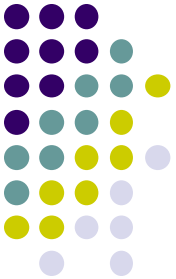
- 例： demo_multThread4.py

Python多线程编程 - threading模块

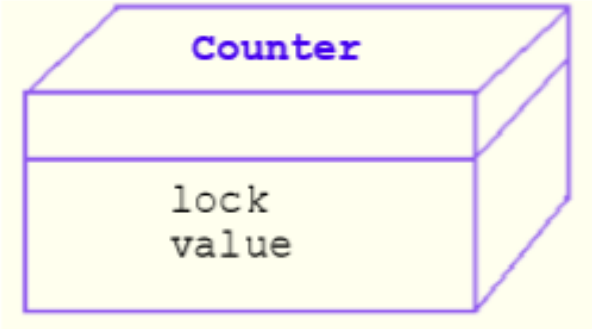
- Thread.join([timeout])：调用线程挂起，直到被join的线程终止。
- 如在main()中执行 for item in threads: item.join()，则时序图如下图。
- 例：demo_multThread5.py



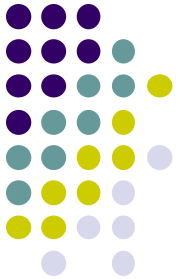
Python多线程编程 - threading模块



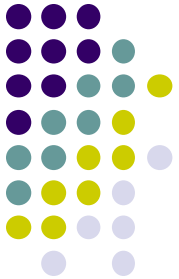
- Thread.Lock()：对全局变量的数据加锁
- 例：demo_multThread6.py



Final



- 设计与实现一个小游戏，如：
  - 扫雷游戏、迷宫游戏、数独
  - tic tac toe（三连棋游戏）、彩色五子棋
  - 小青蛙过河、俄罗斯方块、红心大战
  - ……
- 注意从ftp://public.sjtu.edu.cn/ct/assignments 下载详细final文档
- 上机时间：12月29日 16：00~17：40
- 上机地点：电院4号楼311机房
- 截止日期：1月8日



End