1. 核心算法实现
程序实现了两种策略：
? 随机策略：每个囚犯随机打开50个盒子寻找自己的编号
? 循环策略：囚犯从自己编号的盒子开始，根据盒内纸条跳转到对应盒子
2. 关键函数
? generate_boxes(n)：生成随机排列的盒子
? random_strategy(boxes, prisoner, k)：随机策略实现
? cycle_strategy(boxes, prisoner, k)：循环策略实现
? simulate_single_trial(n, k, strategy)：单次试验模拟
? get_cycle_lengths(boxes)：获取排列的循环长度分布
? calculate_theoretical_success(n, k)：计算循环策略的理论成功率
3. 分析功能
程序提供以下分析：
? 两种策略的成功率对比
? 最大循环长度分布直方图
? 成功/失败试验的循环长度对比
? 参数变化分析（成功率随K/N比例变化）
? 理论成功率与实际成功率对比
4. 理论背景
循环策略的成功率与排列的循环分解有关。当且仅当排列中没有长度大于K的循环时，所有囚犯都能在K步内找到自己的编号。对于N=100, K=50，理论成功率约为：
1 - (1/51 + 1/52 + ... + 1/100) ≈ 0.3118
5. 性能优化
程序通过以下方式优化性能：
? 循环策略中直接分析循环分解，避免模拟每个囚犯的搜索过程
? 使用向量化操作和高效数据结构
? 批量处理多次试验
复杂度分析
? 时间复杂度：
o 随机策略：O(T * N * K)
o 循环策略：O(T * N)（通过循环分解优化）
? 空间复杂度：O(N)（存储盒子排列和访问标记）