>>import random

>>a = [1,2,3,4,5,6,7,8,9]

>>random.shuffle(a)

>>a

[9, 1, 3, 4, 6, 8, 7, 5, 2]

<https://blog.csdn.net/qikaihuting/article/details/78224690>

Python代码:加上算法描述

Fisher–Yates shuffle算法

<https://blog.csdn.net/qikaihuting/article/details/78224690>

现在很多玩牌游戏，在玩家面前，先需要进行随机洗牌，保证发到各玩家手里的牌给的都是均等的。洗牌算法是将原来的数组进行打散，使原数组的某个数在打散后的数组中的每个位置上等概率的出现。即每个牌的抽取概率都要求是1/n，且不能重复抽取。

规则：

（1）每次随机从原始数据列表中，抽取一个，放入新列表；

（2）为了避免重复抽取某一个数据，要求抽取出该数据后，在原始数据列表中被删除。

（3）原始数据列表列表数据抽完，即完成了所有抽取任务。

import random

cards=[i for i in range(9)] #产生0到8顺序数

print(cards)

i=0

news=[] #接受产生的随机数

while cards: #列表里有数值,就循环

n=len(cards) #取被取值列表的长度

if n==1: #当最后一个数值时

news.append(cards[0])#可以直接放到新列表尾部

break #跳出循环

i=random.randint(0,n-1) #在实际长列表里随机产生一个顺序号i

news.append(cards[i]) #把取得的数放入接受取值列表里

cards[i]=cards[-1] #把原先列表的尾部数放到i处

del cards[-1] #删除尾部一个值

print(news) #打印随机取值的结果

可以继续改进，不用删除，而是改为控制n-=1

Knuth 和 Durstenfeld  在Fisher 等人的基础上对算法进行了改进，在原始数组上对数字进行交互，省去了额外O(n)的空间。该算法的基本思想和 Fisher 类似，每次从未处理的数据中随机取出一个数字，然后把该数字放在数组的尾部，即数组尾部存放的是已经处理过的数字。

import random

cards=[i for i in range(9)] #产生0到8顺序数

print(cards)

n=len(cards) #取列表的长度

i=n-1 #i指向列表最后一个元素

while i>=0: #从列表最大范围,缩小到0

p=random.randint(0,i) #在实际长列表里随机产生一个顺序号p

cards.append(cards[p]) #把该顺序号指向的元素抽取,放到列表尾部

del cards[p] #删除该位置的元素

i-=1 #缩小抽取范围

print(cards) #打印随机取值的结果

（1）采用del()函数删除当前元素，当列表元素很多时，删除元素后的所有元素dou 得往前挪1位，在内存里是需要消耗时间的，会造成运行性能下降。

当需要考虑被抽取数据不固定时(随时要增加),则可以用Inside-Out Algorithm

换牌思路：

基本思思是从前向后扫描数据，把位置i的数据随机插入到前i个（包括第i个）位置中（假设为k），这个操作是在新数组中进行，然后把原始数据中位置k的数字替换新数组位置i的数字。其实效果相当于新数组中位置k和位置i的数字进行交互。

import random

import copy

cards=[i for i in range(9)] #产生0到8顺序数

print(cards)

i=0

n=len(cards) #取被取值列表的长度

news=copy.deepcopy(cards)

while i<n:

p=random.randint(0,i)

news[i]=news[p]

news[p]=cards[i]

i+=1

print(news)