实验练习03 参考答案

1. 递归阶乘函数设计

```
def recursion(n):
    if n == 1:
        return 1
    else:
        return n * recursion(n - 1)


test_1 = recursion(5)
print("5!=%d" % test_1)
test_2 = recursion(6)
print("6!=%d" % test_2)
```

2. 计算27的立方根

```
import math
result = math.pow(27,1.0/3)
print(result)
```

3. 分治法寻找假币

对于某一次迭代，硬币数可能为奇数或者偶数。

对于偶数个硬币，寻找中间元素下标，然后统计左右“天平”的重量，然后对较轻的一侧进入递归迭代；

对于奇数个硬币，寻找中间元素下标，同样统计左右“天平”重量，如果左右重量一致，则中间元素就是假币。

迭代终止条件因硬币个数奇偶不同分为两种情况（该题解决方法不唯一，可有其他解决思路）。

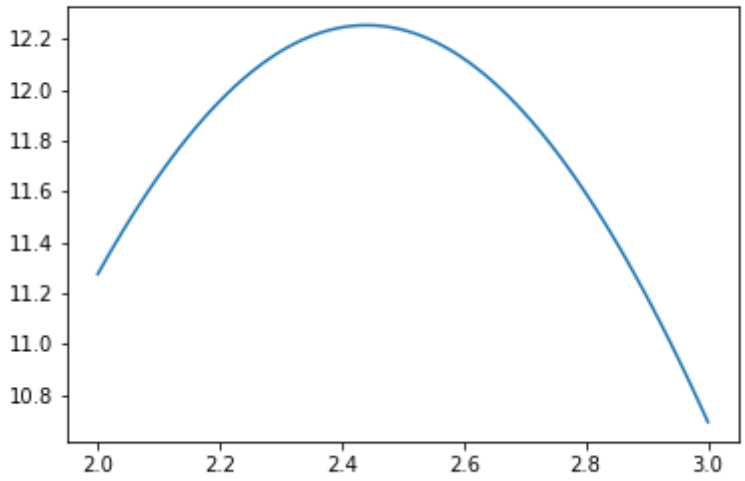
```
def find_incorrect_coin(coins, start_index, end_index):
    if end_index == start_index:
        return end_index
    if end_index - start_index == 1:
        return end_index if coins[end_index] < coins[start_index] else
start_index
    if (end_index - start_index) % 2 == 1:  # 偶数个硬币
        mid_index = (end_index + start_index) // 2
        left_weight = sum(coins[start_index:mid_index + 1])
        right_weight = sum(coins[mid_index + 1:end_index + 1])
        if left_weight < right_weight:
            return find_incorrect_coin(coins, start_index, mid_index)
        else:
            return find_incorrect_coin(coins, mid_index + 1, end_index)
    else:  # 奇数个硬币
        mid_index = (end_index + start_index) // 2
        left_weight = sum(coins[start_index:mid_index])
```

```
        right_weight = sum(coins[mid_index + 1:end_index + 1])
        if left_weight < right_weight:
            return find_incorrect_coin(coins, start_index, mid_index - 1)
        elif left_weight > right_weight:
            return find_incorrect_coin(coins, mid_index + 1, end_index)
        else:
            return mid_index


coins = [2,2,2,2,2,2,1,2,2,2,2,2,2,2,2]
result = find_incorrect_coin(coins, 0, len(coins) - 1)
print(result)
```

4. 蒙特卡洛法求积分

预估函数在[2,3]区间内的最值，大约在12到13之间。




```
import random
import math

S = (3.0 - 2.0) * 13
N = 10000000
C = 0
for i in range(N):
    x = random.uniform(2.0, 3.0)
    y = random.uniform(0.0, 13.0)
    if y <= x **2 + 4 * x * math.sin(x):
        C += 1
I = C * S / N
print(I)
```

蒙特卡洛法的中心思想就是**目标区域内随机点数量与总随机点数量之比近似于目标区域面积与矩形面积之比**

实际求得的近似值与生成的随机数、循环迭代次数都有关系

理论上矩形的纵坐标长度可以取很大的值，只要迭代次数足够多，也可以取得近似值，但为了计算效率，同时保证随机点尽可能落在函数图像下方，我们的纵坐标上界尽量与函数在指定区间的最大值相近。