*1.完成《编程导论》练习题 2.5.2*

```python
def selection_sort(L):
    for i in range(len(L) - 1):
        _min = i
        for j in range(i + 1, len(L)):
            if L[j] < L[_min]:
                _min = j
        L[i], L[_min] = L[_min], L[i]
    return L


L = [12, 11, 3, 11, 6, 11, 12, 3, 11]
L_sorted = selection_sort(L)
ret = []
count = 0
init = L_sorted[0]
for i in L_sorted:
    if init == i:
        count += 1
    else:
        ret.append([init, count])
        init = i
        count = 1
ret.append([init, count])
print(ret)
```

Shell:

```
[[3, 2], [6, 1], [11, 4], [12, 2]]

Process finished with exit code 0
```

*2.完成《编程导论》老虎机游戏的练习题 2.5.4*

（1）

```
import random


money = 10
k = int(input())
while money > 0 and k > 0:
    money = money - 1
    r = random.random()
    if r < 0.6:
        prize = 0
    elif r < 0.8:
        prize = 1
    elif r < 0.9:
        prize = 2
    else:
        prize = 3
    money = money + prize
    print(k, ":The price is", prize, "The remander is", money)
    k -= 1
```

本金设置为 10

当 k=10 时

```
10
10 :The price is 2 The remander is 11
9 :The price is 1 The remander is 11
8 :The price is 0 The remander is 10
7 :The price is 3 The remander is 12
6 :The price is 1 The remander is 12
5 :The price is 1 The remander is 12
4 :The price is 1 The remander is 12
3 :The price is 2 The remander is 13
2 :The price is 1 The remander is 13
1 :The price is 0 The remander is 12
```

当 k=20 时

```
20
20 :The price is 0 The remander is 9
19 :The price is 0 The remander is 8
18 :The price is 0 The remander is 7
17 :The price is 0 The remander is 6
16 :The price is 0 The remander is 5
15 :The price is 1 The remander is 5
14 :The price is 2 The remander is 6
13 :The price is 0 The remander is 5
12 :The price is 0 The remander is 4
11 :The price is 0 The remander is 3
10 :The price is 0 The remander is 2
9 :The price is 2 The remander is 3
8 :The price is 0 The remander is 2
7 :The price is 1 The remander is 2
6 :The price is 0 The remander is 1
5 :The price is 0 The remander is 0
```

当 k=30 时

```
30
30 :The price is 0 The remander is 9
29 :The price is 3 The remander is 11
28 :The price is 0 The remander is 10
27 :The price is 1 The remander is 10
26 :The price is 1 The remander is 10
25 :The price is 0 The remander is 9
24 :The price is 0 The remander is 8
23 :The price is 1 The remander is 8
22 :The price is 0 The remander is 7
21 :The price is 0 The remander is 6
20 :The price is 1 The remander is 6
19 :The price is 0 The remander is 5
18 :The price is 0 The remander is 4
17 :The price is 3 The remander is 6
16 :The price is 1 The remander is 6
15 :The price is 0 The remander is 5
14 :The price is 2 The remander is 6
13 :The price is 0 The remander is 5
12 :The price is 0 The remander is 4
11 :The price is 1 The remander is 4
10 :The price is 0 The remander is 3
9 :The price is 0 The remander is 2
8 :The price is 0 The remander is 1
7 :The price is 0 The remander is 0
```

（2）

```python
import random


money = init_money = 10
loop = 100
win = 0
lose = 0
for i in range(loop):
    money = init_money
    while 0 < money <= init_money:
        money = money - 1
        r = random.random()
        if r < 0.6:
            prize = 0
        elif r < 0.8:
            prize = 1
        elif r < 0.9:
            prize = 2
        else:
            prize = 3
        money = money + prize
    if money > init_money:
        win += 1
    elif money == 0:
        lose += 1

print("win times:", win, ";lose times:", lose)
```

多次运行结果如下：

```
win times: 47 ;lose times: 53
```

```
win times: 48 ;lose times: 52
```

```
win times: 51 ;lose times: 49
```

```
win times: 50 ;lose times: 50
```

可以发现，基本在 50 左右浮动，即有一半的局数赢钱

***3.*** ***代数中有一个定律叫做结合律(Associativity),例如矩阵乘积就符合结合律，而没有交换律。结合律就是((A·B)·C)=(A·(B·C)) 4。所以当有3个矩阵乘积时，有此2种不同的计算“结合”方式。当有n个矩阵乘积，用F(n)表示有多少种不同的“结合”计算方式。很明显的F(2)=1,F(3)=2,F(4)=5。请完成以下任务：***

***a. 请写出递归式子对于任意正整数n的F(n)；***

***b. 写出Python程序计算出F(100)：***

***c. 请计算$\frac{F(n)}{F(n-1)}$的比例，以分析F(n)的增长是否随着n为指数倍的增加。***

<u>以下前提均为“完全添加括号”的“结合”方式</u>

a.

补充定义F(1)=1，那么对于任意正整数n都有：

$$F(n)=\sum_{i=1}^{n-1}F(i)\times F(n-i)$$

b.

```python
def Associativity(n):
    a = 0
    if n == 1:
        a = 1
    else:
        for i in range(1, n):
            a += (Associativity(i) * Associativity(n - i))
    return a


print(Associativity(100))
```

Shell:

```
227508830794229349661819540395688853956041682601541047340

Process finished with exit code 0
```

c.

由递归式可以得到当n≥1时的通项公式为：

$$F(n+1)=\frac{1}{n+1}\binom{2n}{n}=\frac{(2n)!}{(n+1)!\,n!}$$

可以知道当n≥2时满足：

$$\frac{F(n)}{F(n-1)}=\frac{2(2n-3)}{n}$$

即可以推得

$$\lim_{n\to\infty}\frac{F(n)}{F(n-1)}=4$$

说明F(n)的增长是随着n为指数倍增长。

***4. 请编写 Python 程序来计算未来的圆周率 π 值。计算 π 有许多方法此列举两种方式。请用 Python 实现这两种计算方式，要求能够达到 11 位有效数字精度。***

***a.*** $\frac{\pi}{2} = 1 + \frac{1}{3} + \frac{1}{3}\cdot\frac{2}{5} + \frac{1}{3}\cdot\frac{2}{5}\cdot\frac{3}{7} + \cdots$ ***【请尝试用时间复杂度为 O(n) 的算法实现】***

***b.*** $\pi = 16arctan\frac{1}{5} - 4arctan\frac{1}{239}$***，其中*** $arctanx = x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + \cdots$

***c. 请分析哪种方法拥有比较少的计算次数***

因为有现成的 π 值，似乎可以很轻松的知道 11 位有效数字精度的 π 就该为 3.1415926535（去尾后）。但假如这个时候我们在计算一个并不知道的常数，我们就不能凭借着猜测去确定循环的次数，需要通过一些数学的计算。
这些数学计算对于解答后面问题只是辅助作用，<u>代码会直接放在整个数学过程的最后</u>。
首先我们知道：

$$(2k+1)!! = \frac{(2k+1)!}{(2k)\big(2(k-1)\big)\ldots 2(1)} = \frac{(2k+1)!}{2^k k}$$

我们将原本的$\frac{\pi}{2}$的无穷级数式子改写为：

$$1 + \frac{1}{3} + \frac{1}{3}\cdot\frac{2}{5} + \frac{1}{3}\cdot\frac{2}{5}\cdot\frac{3}{7} + \cdots = \sum_{k=0}^{\infty}\frac{k!\,k!\,2^k}{(2k+1)!} = \sum_{k=0}^{\infty}\frac{2^k}{(2k+1)\binom{2k}{k}}$$

参考现有的“中心二项式系数倒数之和”文献（https://www.emis.de/journals/INTEGERS/papers/g27/g27.pdf）中所给的母函数（链接中的 Theorem 2.4），我们可以知道

$$A(t) = \mathcal{G}\left(\frac{4^k}{(2k+1)\binom{2k}{k}}\right) = \frac{1}{t}\sqrt{\frac{t}{1-t}}arctan\sqrt{\frac{t}{1-t}}$$

取$t = \frac{1}{2}$，就可以发现：

$$1 + \frac{1}{3} + \frac{1}{3}\cdot\frac{2}{5} + \frac{1}{3}\cdot\frac{2}{5}\cdot\frac{3}{7} + \cdots = 2arctan1 = \frac{\pi}{2}$$

由于数学知识限制，不知道接下来如何找到一个误差量，但是，通过穷举，我们也可以找到我们要计算多少项才能逼近保证有效数字为 11 位。
更具带有拉格朗日余项的麦克劳林公式，我们可以知道

$$arctanx = x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + \cdots + \frac{(arctan\theta x)^{(n+1)}}{(n+1)!}x^{n+1}, 0 < \theta < 1$$

现在对拉格朗日余项进行处理：

$$R(x) = \frac{(arctan\theta x)^{(n+1)}}{(n+1)!}x^{n+1}, 0 < \theta < 1$$

而对于 acrtanx 的 n 阶导数（https://www.emis.de/journals/AMEN/2010/090408-2.pdf）：

$$(arctanx)^{(n)} = \frac{(-1)^{n-1}(n-1)!}{(1+x^2)^{\frac{n}{2}}}\sin\left(narcsin\left(\frac{1}{\sqrt{1+x^2}}\right)\right), n = 1,2,3\ldots$$

放缩后，可以发现

$$|R(x)| \le \frac{n!\cdot x^{n+1}}{(1+x^2)^{\frac{n+1}{2}}(n+1)!} \le \frac{1}{(n+1)(\frac{1}{x^{n+1}}+1)}$$

当 x=1 时，余项为$\frac{1}{2(n+1)}$，要使得精度达到 11 位有效数字，需要展开到至少$5\times 10^{10}$项（这也为什么不用这个方法去算圆周率了）不过还好，计算机跑其实都不是很大的问题。
跑了许久后，再把结果去和上述级数和相减，直到误差小于$10^{-10}$。<u>**发现，需要计算 33 项可以符合。**</u>
而对于第二个算式，这个应该是第一个的加强版，只需要利用反三角的一些和公式

$$arctan\frac{a}{b}+arctan\frac{c}{d}=arctan\frac{bd+ac}{bd-ac}$$

就可以得到这个算式，我们依旧利用上述提到的拉格朗日余项，用同样的方式放缩后：

$$\left|R\left(\frac{1}{5}\right)\right|-\left|R\left(\frac{1}{239}\right)\right|\le\frac{1}{(n+1)(5^{n+1}+1)}-\frac{1}{(n+1)(239^{n+1}+1)}\le\frac{1}{(n+1)(5^{n+1}+1)}$$

用计算机去近似，发现**需要计算到 15 项可以符合**。

以下是代码部分：

a.

```
vari = 1
_pi = 0
loop = 34
for i in range(1, loop):
    _pi = _pi + vari
    vari = vari * (i / (2 * i + 1))
print(2 * _pi)
```

显然时间复杂度位 O(n)

Shell:

```
3.141592653519746

Process finished with exit code 0
```

b.

```
import math


def acrtan(x, loop_time):
    ret = 0
    for i in range(1, loop_time):
        if i % 4 == 1:
            ret += (math.pow(x, i) / i)
        elif i % 4 == 3:
            ret -= (math.pow(x, i) / i)
    return ret


loop = 16
_pi = 16 * acrtan(1 / 5, loop) - 4 * acrtan(1 / 239, loop)
print(_pi)
```

Shell:

```
3.1415926535886025

Process finished with exit code 0
```

c.

就循环次数而言，第二种方法总共循环了 30 次，第一种方法总共循环了 33 次，比较相当，但是要注意第一种方法采用的乘法运算，第二种方法引入了乘方运算，因此次数显然会比第一种方法多，故**第一种方法计算次数少**。

*5. 以下是一个找列表 L 的最小值和最大值的 Python 程序，用的是递归的方式。然而，有些代码出错变成乱码了，但幸运的是输出的结果保留了下来。请你根据输出结果，在乱码处(#烫烫烫)补上缺失的代码，并运行程序截图输出结果。*

```python
def find_mm(L):
    global depth
    # 烫烫烫
    print("Into", depth, L)
    if len(L) == 1:
        # 烫烫烫
        # 烫烫烫
        return L[0], L[0]
    if len(L) == 2:
        # 烫烫烫
        # 烫烫烫
        return min(L[0], L[1]), max(L[0], L[1])
    min1, max1 = find_mm(L[0:len(L) // 2])
    min2, max2 = find_mm(L[len(L) // 2:len(L)])
    # 烫烫烫
    # 烫烫烫
    # 烫烫烫


depth = 0
L = [9, 3, 1, 5, 2, 0, 7, 8, 10, 2]
print(find_mm(L))
```

```
Into 1 [9, 3, 1, 5, 2, 0, 7, 8, 10, 2]
Into 2 [9, 3, 1, 5, 2]
Into 3 [9, 3]
return to 2
Into 3 [1, 5, 2]
Into 4 [1]
return to 3
Into 4 [5, 2]
return to 3
return to 2
return to 1
Into 2 [0, 7, 8, 10, 2]
Into 3 [0, 7]
return to 2
Into 3 [8, 10, 2]
Into 4 [8]
return to 3
Into 4 [10, 2]
return to 3
return to 2
return to 1
return to 0
(0, 10)

Process finished with exit code 0
```

补充代码后：

```python
def find_mm(L):
    global depth
    depth += 1
    print("Into", depth, L)
    if len(L) == 1:
        depth -= 1
        print("return to", depth)
        return L[0], L[0]
    if len(L) == 2:
        depth -= 1
        print("return to", depth)
        return min(L[0], L[1]), max(L[0], L[1])
    min1, max1 = find_mm(L[0:len(L) // 2])
    min2, max2 = find_mm(L[len(L) // 2:len(L)])
    depth -= 1
    print("return to", depth)
    return min(min1, min2), max(max1, max2)


depth = 0
L = [9, 3, 1, 5, 2, 0, 7, 8, 10, 2]
print(find_mm(L))
```

Shell:

```
Into 1 [9, 3, 1, 5, 2, 0, 7, 8, 10, 2]
Into 2 [9, 3, 1, 5, 2]
Into 3 [9, 3]
return to 2
Into 3 [1, 5, 2]
Into 4 [1]
return to 3
Into 4 [5, 2]
return to 3
return to 2
return to 1
Into 2 [0, 7, 8, 10, 2]
Into 3 [0, 7]
return to 2
Into 3 [8, 10, 2]
Into 4 [8]
return to 3
Into 4 [10, 2]
return to 3
return to 2
return to 1
return to 0
(0, 10)

Process finished with exit code 0
```

*6. 大舅要贷款买房，知道你是华师大计算机专业的学生，认为你什么都懂，所以就请你为他解释并分析贷款的计算方式。你学了基本编程后自然功力大为增进，因此准备写个 Python 程序来回答大舅的疑问。这个题目包含三个层面：探索、编程及分析，我们要自行探索现在银行最通用的贷款方式“等额本息”的原理，我们要利用编程计算出还款金额、每年所付出本金和利息，等等；最后是分析利弊得失。完成这个题目后，你已经比你中学同学上商学院一年级的要厉害多了！（本题假设本金为 A，月利率为 r，贷款期限为 n 个月）*

*a. 使用”等额本息”方式，请推导并说明每月还款金额的公式，请自行到网上探索相关资料。*

*b. 编写 Python 程序，假设贷款的本金为 70 万，年利率 6%（月利率是它的 1/12），贷款 30 年，*

*i. 求出每月还款金额；*

*ii. 列出每年归还的本金及利息，以及全部交付的本金及利息；想想为什么要付这么多的利息啊？你大舅问你是不是前几年付的钱大部分都是付给利息啊？*

*c. （在题目 b 基础上）如果贷款期限为 20 年，利息是不是少了很多？*

*d. 另外一种贷款计算方式叫做“等额本金”，请到网上探索相关资料。以 20 年贷款 70 万，年利率 6%为例（为简单起见，以“月”为单位，不以“天”为单位）请编程计算出使用等额本金方式所支付的利息总数，以及列出前 12 个月每月所还的金额，并且请从利息的角度说明它与等额本息的差异，请分析两者（等额本息和等额本金）的优劣，尝试分析为什么现在银行都喜欢用等额本息方式来计算贷款？*

a.

设每个月还款 x 元，则

$$第一个月末还款 = A(1+r) - x$$

$$第二个月末还款 = (A(1+r) - x)(1+r) - x = A(1+r)^2 - x(1+(1+r))$$

$$第m个月末还款 = A(1+r)^m - x(\frac{(1+r)^m - 1}{r})$$

由于第 n 个月还款完成，因此：

$$A(1+r)^n - x\left(\frac{(1+r)^n - 1}{r}\right) = 0$$

故：

$$每月还款金额 = \frac{r \times (1+r)^n}{(1+r)^n - 1} \times A$$

每个月还的钱是相同的，因此称为等额本息。

b.

i.

```
import math


def equal_loan_payment(A, r, n):
    ret = ((r * math.pow(1 + r, n)) / (math.pow(1 + r, n) - 1)) * A
    return ret


principal = 700000
annual_rate = 0.06
limit = 30

print(equal_loan_payment(principal, annual_rate / 12, limit * 12))
```

Shell:

```
4196.853676069299

Process finished with exit code 0
```

每个月还款 4196.85 元

ii.

省略计算等额本息的函数定义

```python
principal = 700000
annual_rate = 0.06
limit = 30
pre_interest = 0
pre_principal = 0
total_interest = 0
total_principal = 0
pay_loan = equal_loan_payment(principal, annual_rate / 12, limit * 12)
for i in range(1, 31):
    for j in range(1, 13):
        pay_interest = principal * (annual_rate / 12)
        pre_interest += pay_interest
        pay_principal = pay_loan - pay_interest
        pre_principal += pay_principal
        principal -= pay_principal
    print(i, ": interest: %.2f principal: %.2f" % (pre_interest, pre_principal))
    total_interest += pre_interest
    total_principal += pre_principal
    pre_interest = 0
    pre_principal = 0
print("total interest: %.2f, total principal: %.2f" % (total_interest,total_principal))
```

Shell:

```
1 : interest: 41766.16 principal: 8596.08
2 : interest: 41235.97 principal: 9126.27
3 : interest: 40673.09 principal: 9689.16
4 : interest: 40075.48 principal: 10286.76
5 : interest: 39441.02 principal: 10921.23
6 : interest: 38767.42 principal: 11594.83
7 : interest: 38052.27 principal: 12309.97
8 : interest: 37293.02 principal: 13069.22
9 : interest: 36486.94 principal: 13875.30
10 : interest: 35631.14 principal: 14731.10
11 : interest: 34722.56 principal: 15639.68
12 : interest: 33757.94 principal: 16604.30
13 : interest: 32733.82 principal: 17628.42
14 : interest: 31646.54 principal: 18715.70
15 : interest: 30492.20 principal: 19870.05
16 : interest: 29266.65 principal: 21095.59
17 : interest: 27965.52 principal: 22396.72
```



```
16 : interest: 29266.65 principal: 21095.59
17 : interest: 27965.52 principal: 22396.72
```



```
18 : interest: 26584.14 principal: 23778.10
19 : interest: 25117.56 principal: 25244.68
20 : interest: 23560.53 principal: 26801.72
21 : interest: 21907.46 principal: 28454.79
22 : interest: 20152.43 principal: 30209.82
23 : interest: 18289.15 principal: 32073.09
24 : interest: 16310.95 principal: 34051.29
25 : interest: 14210.74 principal: 36151.50
26 : interest: 11981.00 principal: 38381.25
27 : interest: 9613.73 principal: 40748.52
28 : interest: 7100.45 principal: 43261.80
29 : interest: 4432.15 principal: 45930.09
30 : interest: 1599.29 principal: 48762.96
total interest: 810867.32, total principal: 700000.00
```

一方面由于还款时间长；另一方面由于这种方式一开始还的本金少，利息就会很多，导致大量的钱都用于还利息而不是本金。

前几年大多数的钱都用于还利息去了。

c.

```python
principal = 700000
annual_rate = 0.06
limit = 20
```

```
total_interest = 0
total_principal = 0
pay_loan = equal_loan_payment(principal, annual_rate / 12, limit * 12)
for i in range(1, 21):
    for j in range(1, 13):
        pay_interest = principal * (annual_rate / 12)
        total_interest += pay_interest
        pay_principal = pay_loan - pay_interest
        total_principal += pay_principal
        principal -= pay_principal
print("total interest: %.2f, total principal: %.2f" % (total_interest,
total_principal))
```

Shell:

```
total interest: 503604.18, total principal: 700000.00

Process finished with exit code 0
```

利息减少了很多。

d.

```
def equal_principal_payment(A, r, n, x):
    pay_interest = (A - (A / n) * x) * r
    pay_loan = (A / n) + pay_interest
    return pay_interest, pay_loan


principal = 700000
annual_rate = 0.06
limit = 20
total_interest = 0
for i in range(0, limit * 12):
    interest,pay_loan=equal_principal_payment(principal,annual_rate / 12,limit * 12, i)
    total_interest += interest
    print(i + 1, ":pay_loan: %.2f" %  pay_loan)
print("the total interest: %.2f" % total_interest)
```

Shell:

```
the total interest: 421750.00
```

利息总数为 421750.00 元；前 12 个月所还款位：

```
1 :pay_loan: 6416.67
2 :pay_loan: 6402.08
3 :pay_loan: 6387.50
4 :pay_loan: 6372.92
5 :pay_loan: 6358.33
6 :pay_loan: 6343.75
7 :pay_loan: 6329.17
8 :pay_loan: 6314.58
9 :pay_loan: 6300.00
10 :pay_loan: 6285.42
11 :pay_loan: 6270.83
12 :pay_loan: 6256.25
```

可以发现，等额本息每个月还款相同，一边付本金一边付利息，但是一开始利息较多，付的本金较少，导致整个还款过程利息较多，需要付的总利息更多；而等额本金一开始固定还每个月的本金外，还要付当个月的利息，使得利息更少，总利息更少。

等额本息的优点是每个月还款是确定的，风险较小；缺点是利息较高，总共需要还款的钱更多。

等额本金的优点是利息较低，总共需要还款的钱更少；缺点是每个月还款是先多后少的，一开始风险较高，不容易还上。

银行喜欢采用等额本息的原因是可以通过利息获得更多的收入，也可以凭借比等额本金更少的初月还款吸引贷款者。