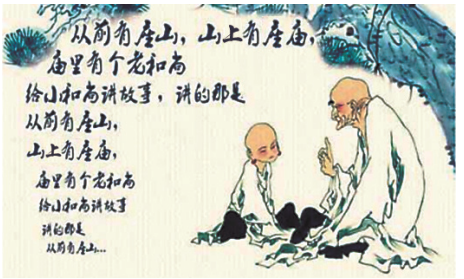

图 3-1　递归的故事

大家一定听过一个如图 3-1 所示这样的故事：从前有座山，山上有座庙，庙里有个老和尚，老和尚在给小和尚讲故事，讲的是什么呢？从前……然后一直这样不断讲下去。我们发现：在故事中，不断提到了同样的故事。

就像一个人站在装满镜子的房间中，看到的影像就是递归的结果。

生活中很多事物拥有一种奇特的结构，就是在一个结构中，蕴含着一个与自身相似的另一个的结构，如此往复，如常见的花菜、雷雨过后的闪电、漫天飞舞的雪花、贝壳身上的螺旋图案。小至各种植物的结构及形态，遍布人体全身纵横交错的血管，大到天空中聚散不定的白云、连绵起伏的群山。递归贯穿于各种文化，贯穿于科学、数学和艺术之中，如图 3-2 所示。

图 3-2　生活中的递归结构

快来学习神奇的递归，它能够以非常简洁的逻辑解决重要的问题！

3.1　什么是递归

一个事物由这个事物本身所构建，那么在理解这个事物的时候，就需要先理解事物的构成，于是就回到理解这个事物本身，从而再次需要理解这个事物的构成……这个不断循环理解的过程就形成了递归。

Python 中的函数作为一种代码封装，可以被其他程序调用，当然，也可以被自身函数内部的代码所调用。这种函数定义中调用函数自身的方式称为递归。

“老师，一个函数可以调用另一个函数，那么函数能够调用自己本身吗？”

“可以的，函数自己调用自己，就形成了递归。”

“老师，您能给我举个递归的例子吗？”

“嗯，我们通过汉诺塔游戏，来感受递归的强大吧！”

汉诺塔 (Tower of Hanoi) 来源于古印度的传说。在印度北部贝拿勒斯的圣庙里，一块黄铜板上插着三根宝石针，在其中一根针上从下到上地穿好了由大到小的 64 片金片，这就是所谓的汉诺塔。不论白天黑夜，总有一个僧侣在按照下面的法则移动这些金片：一次只移动一片，不管在哪根针上，小片必须在大片上面，如图 3-3 所示。

图 3-3　汉诺塔游戏及规则

那么多盘子，移动的过程真的非常复杂！经计算，64 片金片从穿好的那根针上完全移动到另一根针上，需要移动 18 446 744 073 709 551 615 次，就算每秒移动一次，完成也需要 5845.54 亿年以上。但是这么复杂的问题，用递归完成起来却非常简洁，十几行代码就足够啦！

“老师，能不能讲个简单的例子，让我感受一下递归呢？”

【例 3-1】 求 1~n 的和。

数学上有一个经典的递归例子叫求 1~n 的和：

$$f(n)=1+2+3+\cdots+n$$

为了实现这个程序，可以通过一个简单的循环求和去计算。实际上，1~n 的累加和也可以给出另一种表达式：

$$f(n)=f(n-1)+n$$

“这个函数这样写有什么问题？”

“f (1) = f (0)+1，f (0) = f (–1)+0，…如果一直这样根本没办法得到正确答案。”

递归函数需要有边界：即什么时候递归可以停止。例如，设当 n==0 时，f(0)=0，这样递归就可以停止。如果缺少递归的边界，则会出现如图 3-4 所示的错误。

```
def f(n):
    return n + f(n-1)
s = f(10)
print("1 到 10 的和为 :{}".format(s))
```

```
    return n + f(n-1)
  [Previous line repeated 996 more times]
RecursionError: maximum recursion depth exceeded
```

图 3-4　递归错误

重新修改程序，加入边界后的程序如下。

```
def f(n):
    if n>0:
        return n + f(n-1)
    else:
        return 0
s = f(10)
print("1 到 10 的和为 :{}".format(s))
```

通过这个例子，可以总结出递归的两个关键特征。

（1）递归函数存在一个或多个边界条件，满足边界不需要再次递归。

（2）所有递归链要以一个或多个边界条件结尾。

【问题 3-1】　递归函数如果没有边界条件，会出现什么?

【问题 3-2】　递归函数的代码复杂吗?

【问题 3-3】　下列关于递归函数的叙述中，正确的是（　　）。

A. 函数间接地调用自身实现的也是递归

B. 递归函数包含一个循环结构
C. 递归函数不用边界条件也能正确实现
D. 递归函数代码复杂，难以实现

3.2　简单的递归实现

“小萌，你在想什么呢？”

“嗯…，我在想，既然自然界普遍存在递归现象，那么递归是否可以帮助我们解决生活中遇到的问题呢？……”

【例 3-2】 有趣的兔子问题（见图 3-5）。

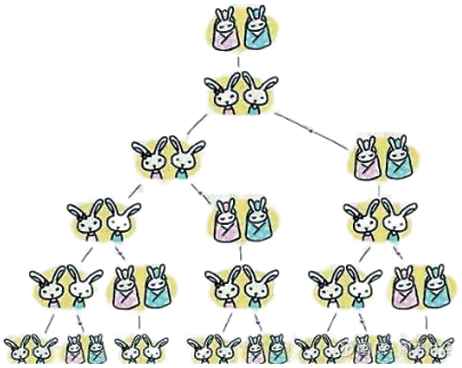

图 3-5　兔子问题

有个人想知道，一年之内一对兔子能繁殖多少对？于是就筑了一道围墙把一对兔子关在里面。已知一对兔子每个月可以生一对小兔子，而一对兔子从出生后第 3 个月起每月生一对小兔子。假如一年内没有发生死亡现象，那么，一对兔子一年内（12 个月）能繁殖成多少对？（兔子的规律为数列：1，1，2，3，5，8，13，21，…）

其实这就是斐波那契数列：一个数列当前项等于前两项之和。斐波那契数

列（Fibonacci sequence），又称黄金分割数列，因数学家莱昂纳多·斐波那契（Leonardo Fibonacci）以兔子繁殖为例子而引入，故又称为“兔子数列”。

现在我们来帮帮小萌和小帅，利用递归完成斐波那契数列吧！

```
def fib(n):
    if n in [1,2]:
        return 1
    else:
        return fib(n-1)+fib(n-2)
n = eval(input(" 请输入你要查询的月份："))
print("{} 个月兔子有 {} 对 ".format(n,fib(n)))
```

斐波那契数列在自然科学的其他分支有许多应用。例如，树木的生长，由于新生的枝条，往往需要一段“休息”时间，供自身生长，而后才能萌发新枝，所以，一株树苗在一段间隔（例如一年）以后长出一条新枝；第二年新枝“休息”，老枝依旧萌发；此后，老枝与“休息”过一年的枝同时萌发，当年生的新枝则次年“休息”。这样，一株树木各个年份的枝 Y 数，如图 3-6 所示，便构成斐波那契数列。这个规律，就是生物学上著名的“鲁德维格定律”。

图 3-6　花椰菜的螺旋

另外，观察野玫瑰、大波斯菊、金凤花、百合花、蝴蝶花的花瓣，可以发现它们的花瓣数目都是斐波那契数列中的某个值：3，5，8，13，21，…

【例 3-3】　斐波那契螺旋线。

斐波那契螺旋线也称为“黄金螺旋线”，是根据斐波那契数列画出来的螺旋曲线，这种形状在自然界中无处不在。该原理和黄金比例紧密相连，用后一项除以前一项，比例会越来越接近 1.618 : 1。常见于各种摄影构图、设计理念、建筑物当中，自然界中也有很多如贝类的螺旋轮廓线、银河漩涡等天然的“黄

金螺旋”，如图 3-7 和图 3-8 所示。

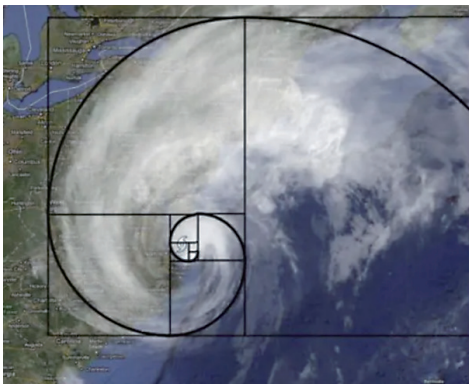

图 3-7　自然界中的斐波那契螺旋线

图 3-8　建筑中的斐波那契螺旋线

可以将斐波那契螺旋线抽象成如图 3-9 所示的原理图。

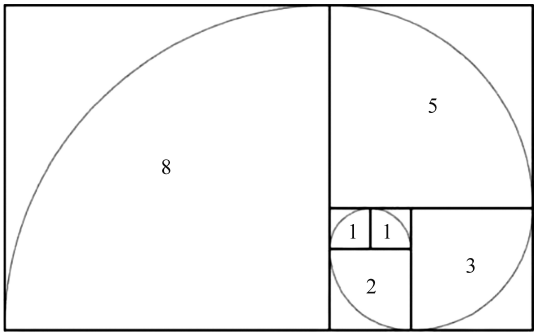

图 3-9　斐波那契螺旋线原理图

如何通过 Python 来绘制斐波那契的图形和螺旋线呢？下面的规律是编程实现的关键所在。

（1）正方形的边长是逐渐增加的斐波那契数列。

（2）螺旋线由诸多四分之一圆弧构成，圆弧的半径仍是斐波那契数列，且圆弧的绘制起点与正方形绘制起点相同，在绘制一个正方形后，会回到该正方形的起点和方向，再绘制四分之一圆弧，圆弧绘制后的终点正好是下一个正方形的起点。

将各个正方形和圆弧连接起来，即可完成斐波那契数列的图形和螺旋线。根据分析完成下面的程序：

```
from turtle import *
from random import *
def draw_f(n):
```

```
        for i in range(4):
            fd(n*10)
            left(90)
        circle(n*10,90)
    def fib(n):
        if n==1 or n==2:
            return 1
        else:
            return fib(n-1)+fib(n-2)
    n = int(input(" 请输入斐波那契项数 n(n>=1):"))
    hideturtle()
    speed(0)
    pencolor('black')
    pensize(2)
    for i in range(1,n+1):
        n = fib(i)
        r,g,b = random(),random(),random()
        fillcolor(r,g,b)
        begin_fill()
        draw_f(n)
        end_fill()
    done()
```

运行结果如图 3-10 所示。

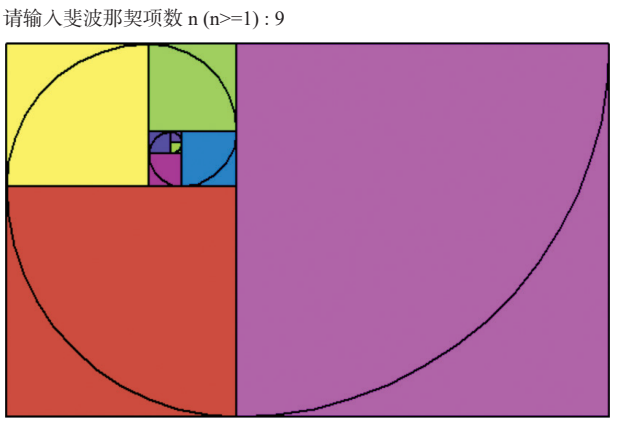

图 3-10　斐波那契数列的图形和螺旋线

【问题 3-4】　如果将例 3-3 程序中的语句：“for i in range(1,n+1)”，写成“for i in range(n)”行不行？为什么？

【问题 3-5】　使用递归完成输入一个整数 n，求 n 阶乘的功能。请在程序的横线上填写适当的语句，实现该功能。

```
def jc(n):
    if n in [0,1]:
        return 1
    else:
        ________
n = eval(input(" 请输入一个大于或等于 0 的整数 :"))
print("{} 的阶乘值为 {}".format(n,jc(n)))
```

A. `n*jc(n-1)`

B. `return n!`

C. `return n*jc(n-1)`

D. `return n*(n-1)`

“在本单元中，我们掌握了递归的使用方法，能够在生活的实际问题中使用递归解决问题，并绘制了美丽的黄金图形和黄金螺旋线。

在后续的课程中，我们还将进一步发现问题，使用递归来解决问题、利用递归来解决很复杂的事，会变得容易很多。”

习　题

1. 下列关于递归函数的叙述中，正确的是（　　）。
   A. 递归函数应该有边界条件以保证函数正确性
   B. 递归函数必须用函数名作为返回值
   C. 递归函数的实现通常比非递归函数复杂
   D. 递归函数中必须包含循环结构

2. 下列关于递归函数的叙述中，正确的是（　　）。
   A. 递归函数的边界条件只能有一个
   B. 递归函数的运行效率比较高
   C. 递归函数的边界条件决定递归调用的深度
   D. 递归函数在时间上消耗大，对计算机内存消耗小

3. 下列关于递归函数的叙述中，正确的是（　　）。
   A. 递归函数的本质是通过自己直接或间接地调用自身实现的
   B. 递归函数占用计算机内存资源较少
   C. 递归函数无须边界条件也能正确实现
   D. 递归函数思路简单，代码复杂难实现

4. 下列关于递归函数的叙述中，不正确的是（　　）。
   A. 递归函数都可以用非递归的方法实现
   B. 递归函数使用不当容易导致程序发生内存溢出错误
   C. 递归函数边界条件可以有一个或多个
   D. 递归函数的返回值必须是函数名

5. 下列关于递归函数的叙述中，不正确的是（　　）。
   A. 正确的递归函数必须有边界条件
   B. 递归函数使用不当容易导致程序发生内存溢出错误
   C. 递归函数一般比非递归实现需要更多的代码
   D. 递归函数的边界条件可以有一个或多个

6. 函数定义代码如下，有关函数调用及返回结果的叙述，正确的是（　　）。

```
def demo(n):
    if n <= 0:
        return 1
    elif n in (3,5):
        return 8
    else:
        return n+demo(n-3)
```

A. demo (8) 的返回值为 18
B. demo (6) 的返回值为 16
C. demo (5) 的返回值为 8
D. demo (4) 的返回值为 5

7. 函数定义代码如下，有关函数调用及返回结果的叙述，正确的是（　　）。

```
def demo(n):
    if n in (1,3,5):
        return 3
    elif n in (2,4,6):
        return 4
    else:
        return n + demo(n-2)
```

A. demo (10) 的返回值为 21
B. demo (9) 的返回值为 19
C. demo (8) 的返回值为 11
D. demo (7) 的返回值为 8

8. 函数定义代码如下，有关函数调用及返回结果的叙述，正确的是（　　）。

```
def demo(n):
    if n in (1, 4):
        return 3
    elif n in (2, 3):
```

```
            return 4
        else:
            return n + demo(n-2)
```

A. demo (10) 的返回值为 28
B. demo (9) 的返回值为 24
C. demo (8) 的返回值为 17
D. demo (7) 的返回值为 15

9. 函数定义代码如下，有关函数调用及返回结果的叙述，正确的是 (　　)。

```
def demo(n):
    if n in (1, 5):
        return 3
    elif n in (2,6):
        return 4
    else:
        return n + demo(n-3)
```

A. demo (10) 的返回值为 24
B. demo (9) 的返回值为 15
C. demo (8) 的返回值为 12
D. demo (7) 的返回值为 16

10. 函数定义代码如下，有关函数调用及返回结果的叙述，正确的是 (　　)。

```
def demo(n):
    if n < 3:
        return 5
    elif n in (2, 5, 6):
        return 7
    else:
        return n + demo(n-1)
```

A. demo (8) 的返回值为 21

B. demo (6) 的返回值为 5
C. demo (4) 的返回值为 12
D. demo (2) 的返回值为 7

11. 使用自定义函数实现求解数字序列中指定位置的值的功能。

有一个数字序列：1，2，2，4，8，32，256 ，8192，2 097 152，…，其第 1 项和第 2 项的值分别为 1 和 2，从第 3 项开始，每一项的值都是其前两项数值的乘积。例如，第 6 项的值为第 4 项的值 4 和第 5 项的值 8 的乘积，即 32。

具体要求如下。

（1）函数名称为：mu2。

（2）mu2 函数有 1 个参数，该参数为自然数，参数名称不限。

（3）mu2 函数有 1 个返回值，该返回值为非负整数，返回值为数字序列中函数参数对应项的数值。

函数调用举例说明：

mu2(6) 的返回值为 32。

12. 使用自定义函数实现求解如下问题。

有一人带着苹果外出售卖，每经过一个村庄，卖出手边苹果的一半加一个。他经过了 m 个村子后，手里还剩余 2 个苹果，求解出发时，他一共带了多少个苹果。

例如，他经过了 3 个村庄，还剩余 2 个苹果，那么他出发时，带的苹果个数为 30 个。

具体要求如下。

（1）函数名称为：apple。

（2）apple 函数有 1 个参数，为自然数，参数名称不限，代表经过的村庄个数。

（3）apple 函数有 1 个返回值，为非负整数。该返回值为商人刚出门时携带的苹果个数。

函数调用举例说明：

apple(3) 的返回值为 30。

13. 使用自定义函数实现如下功能。

有一个数字序列：1，2，3，4，4，6，7，10，11，16，18，26，…。其第 1，2，3，4 项的值分别为 1，2，3，4，从第 5 项开始，每一项的值都是其左侧隔位第

2 个和第 4 个数据的和。例如，第 5 项的值为第 3 项的值 3 和第 1 项的值 1 的和，即 4；第 6 项的值为第 4 项的值 4 和第 2 项的值 2 的和，即 6。

具体要求如下。

（1）函数名称为：add2。

（2）add2 函数有 1 个参数，为自然数，参数名称不限。

（3）add2 函数有 1 个返回值，为非负整数。该返回值表示数字序列中函数参数对应项的值。

函数调用举例说明：

add2 (8) 的返回值为 10。