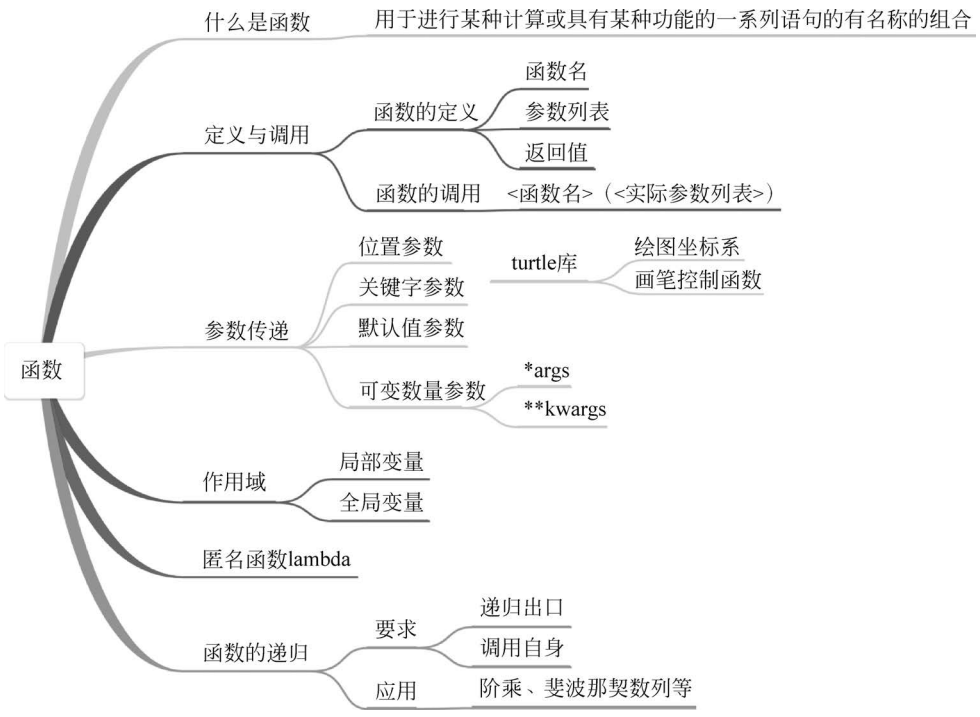
第5章

函　数

【本章导读】

数学中的函数 y＝f(x)可以实现某种数据运算功能，例如 y＝sin(x)用来计算自变量 x 的正弦值 y。程序设计中也有函数的概念。程序中的函数是可以实现某个特定功能的小程序块(block)。每当程序需要实现该特定功能时，只需要调用事先写好的函数，不必每次重复编写相同功能的代码。当需要改变函数功能时，只需要修改函数中的代码，则程序中所有调用该函数的地方都会同步修改。

【本章主要内容】



5.1 函数的定义和调用

Python 程序不需要 main()函数就可以运行，对于一些简单的、小规模的程序，通常不需要定义函数就可以实现所有功能。但当问题复杂性提高后，若把所有代码都写在一起，其编码实现、阅读和维护将会变得非常困难。针对复杂问题的程序设计，一般的方法是先把问题分解为若干子问题，再将每个子问题编写成一个函数，以降低编程难度，提高程序的可读性、可重用性。

在程序设计中，函数是指用于进行某种计算或具有某种功能的一系列语句的有名称的组合。定义函数时，需要明确指定函数名称、可接受的参数以及实现函数功能的程序语句。完成函数定义后，可以通过函数名称调用该函数。例如，系统内置函数 input()、print()等，这些函数把输入、输出等功能语句封装，以函数的形式提供给用户使用。用户在用到这些功能时，不需要再重复编写代码来实现，直接通过函数的调用和参数的传递来实现相关的功能。

在实际的程序设计过程中，有很多操作是完全相同或非常相似的，可以由一段代码来实现。在需要这个功能的地方复制该代码段就可以实现功能的复制。但从程序设计的角度讲，这样直接复制代码段并不明智。大量的重复代码不仅会增加程序的代码行数，也会使程序的逻辑变得更加复杂。解决这个问题的一个有效的方法是设计函数。将可能需要反复执行的代码封装为函数，在需要执行该功能的地方调用该函数，可以实现代码的复用。应用函数的方法也可以保证代码的一致性，对函数的修改可以同时作用到所有调用该函数的位置。

5.1.1 函数的定义

Python 程序中函数的使用要遵循先定义后调用的规则。也就是说函数的调用必须位于函数定义之后，一般的做法是将函数的定义放在程序的开头部分。

函数的定义形式如下。

```
def  <函数名>  (<参数>):
    <函数体>
    return  <返回值列表>
```

(1) def 是关键字，是英文单词 define 的缩写，def 后的空格接函数名，函数名可以是任何有效的 Python 标识符。圆括号和冒号“:”是语法的一部分，不能省略。

(2) 参数是调用该函数时传递给它的值，又称为形式参数(简称形参)，可以有零个、一个或多个。当参数个数为 0 时表明函数体内的代码无须外部传入参数就可以执行，当传递多个参数时参数之间用逗号隔开。

(3) 函数体是函数每次被调用时执行的一组语句，由一行或多行语句组成，通过执行语句来实现函数所定义的特定功能。函数体相对于 def 关键字要缩进 4 个空格。

(4) 函数定义时希望可以将函数的处理结果返回给调用处进行更进一步的处理，此时可使用 return 语句向外提供该函数的处理结果。函数的返回值语句由 return 关键词开头，返回值没有类型和个数限制。当返回值为多个时，这些值会被作为一个元组中的元素。多

个返回值之间逗号隔开。函数没有返回值语句时，例如，用 print()在函数中输出处理结果，或者利用绘图语句直接绘制图形，此时函数返回 None 值。

5.1.2　函数的调用

程序中定义的函数只有在被调用时才运行。定义好的函数可以通过名字来进行调用，Python 函数调用的一般形式如下。

```
<函数名>(<实际参数列表>)
```

调用函数时，实际参数列表中给出要传入函数内部的参数，这类参数称为实际参数，简称实参。调用时传入的参数必须具有确定的值，这些值会被传递给函数定义中的形参，相当于一个赋值的过程。

【例 5.1】 定义求阶乘的函数 fact()，调用 fact()函数，计算并输出组合式 C_{10}^{2} 的值。

【分析】 数学中的组合式 C_n^m 是从 n 个元素中不重复地选取 m 个元素的一个组合，计算公式为 $C_n^m=\frac{n!}{m!(n-m)!}$。从公式可以看出，计算组合数需要多次计算阶乘，因此自定义一个求阶乘的函数 fact()，然后在计算组合数的过程中多次调用求阶乘的函数。fact()函数需要一个参数，求得的阶乘值通过 return 返回。

程序代码如下：

```
def fact(n):
    s = 1
    for i in range(1,n + 1):
        s * = i
    return s
# 分别调用 fact()函数,求 10、2 和 8 的阶乘
a = fact(10)
b = fact(2)
c = fact(10 - 2)
print('C(10,2) = ',a//(b * c))
```

代码运行结果：

```
C(10,2) = 45
```

程序说明：

(1) 该程序运行从 a=fact(10)开始，由于调用了 fact()函数，因此转向执行 fact()函数的定义部分，并将实参 10 传递给参数 n，n 的值为 10。

(2) 在函数调用过程中，实参 10 传递给形参 n，经过 for 循环计算得到 s=3 628 800，通过 return 将 s 的值返回，并赋值给 a，数据传递如图 5.1 所示。

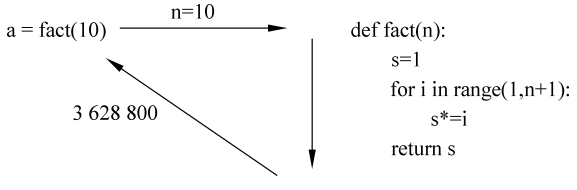


图 5.1　函数的调用过程

(3) 同样的调用过程,计算 b=fact(2)和 c=fact(10-2)。使用函数降低了编程难度,同时可以多次复用相似代码,提高了代码效率。

5.1.3 文档注释

在代码中添加注释能够方便查看代码功能,提高代码可读性。文档注释是 Python 独有的注释方式,用三双引号括起来的注释语句作为函数里的第一条语句,注释内容可以通过对象的__doc__成员被自动提取,并且被 pydoc 所用。文档注释的内容主要包括该函数的功能、可接受参数的个数和数据类型、返回值的个数和类型等。例如:

```
def add(x,y):
    """
    Calculate the sum of two numbers.
    The numbers could be integer or float.
    """
    return x + y
print(add.__doc__)
```

代码运行结果:

```
Calculate the sum of two numbers.
The numbers could be integer or float.
```

5.2 函数参数

如果是有参函数调用,系统将实参传递给被调函数的形参。参数类型主要有以下 4 种,分别为位置参数、关键字参数、默认值参数和可变数量参数。

5.2.1 位置参数

在函数调用过程中,实参按顺序传递给对应的形参,这种形参也称为位置参数。

【例 5.2】 函数的参数。

```
def introduce(name,city,hobby):
    print(f"我的名字是{name},来自{city},爱好是{hobby}。")
introduce("张浩","北京","篮球")
```

在定义 introduce()函数时形参 name、city、hobby 没有值,当调用 introduce()函数时,将"张浩""北京""篮球"分别依次传递给 name、city 和 hobby 参数,输出结果如下:

```
我的名字是张浩,来自北京,爱好是篮球。
```

5.2.2 关键字参数

在函数调用时,提供实参对应的形参名称,根据每个参数的名称传递参数。例如:

```
introduce(city = "北京",name = "张浩",hobby = "篮球")
```

此时，实参的顺序已不再重要，因为实参与形参的对应靠关键字进行区分。这种实参被称为关键字参数。这样做可以增强程序的易读性，参数书写顺序也更加灵活；缺点是增加了函数调用时的代码书写量。

实际上，位置参数和关键字参数还可以混着用，但一定要把位置参数置于关键字参数之前，否则，编译器无法明确知道除关键字以外的参数出现的顺序。

```
introduce("张浩",city = "北京",hobby = "篮球")
```

下面的代码会报 Positional argument after keyword argument 错误。

```
introduce(name = "张浩","北京",hobby = "篮球")
```

5.2.3 默认值参数

在定义函数时，可以使用形如 hobby＝"篮球"的方式给形参赋予默认值。在函数调用时，如果该参数得到传入值，按传入值进行计算，否则将使用默认值。这有助于使函数更灵活，因为不必总是提供所有参数的值。例如：

```
def introduce(name,city,hobby = "篮球"):
    print(f"我的名字是{name},来自{city},爱好是{hobby}。")
introduce("张浩","北京") #默认值参数可以不传值
introduce("李菲","烟台","唱歌")
```

输出结果如下：

```
我的名字是张浩,来自北京,爱好是篮球。
我的名字是李菲,来自烟台,爱好是唱歌。
```

默认值参数可以有多个，例如：

```
def introduce(name, city = "北京", hobby = "篮球"):
    print(f"我的名字是{name},来自{city},爱好是{hobby}。")
```

当函数的参数有多个时，默认值参数必须在后面。以下代码运行时会报 SyntaxError：non-default argument follows default argument 的错误。

```
def introduce(name,hobby = "篮球",city):        #默认值参数应该在参数列表最后面
    print(f"我的名字是{name},来自{city},爱好是{hobby}。")
```

5.2.4 可变数量参数

有时，可能希望函数接受可变数量的参数，而不确定参数的数量。在 Python 中，可以使用 * args 和 ** kwargs 来实现这一点。

1. * args

* args 用于传递非关键字可变数量参数，它们以元组的形式传递给函数。

【例 5.3】 可变数量参数。

```
def add( * args):
    result = 0
    for num in args:
        result += num
    return result
s = add(1, 2, 3, 4, 5)
print(s)
```

代码运行结果：

```
15
```

程序说明：

(1) * args 允许传递任意数量的参数，并将它们收集到一个元组中。

(2) 带星号的参数如果不是放在最后，需要使用形参的名称指定后续参数。下面代码中，实参 10 赋值给 x，调用函数 star()中的实参 z 值为 9 与形参 z 对应，实参中剩余的元素组成元组(20,30,40,500)赋值给形参 * y。

```
def star(x, * y,z):
    print(x,y,z)
star(10,20,30,40,50,z = 9)
```

代码运行结果：

```
10 (20, 30, 40, 50) 9
```

2. ** kwargs

** kwargs 用于传递关键字可变数量的参数，它们以字典的形式传递给函数。例如：

```
def person_info(n, ** kwargs):
    print("{: - ^10}".format(n))
    for key, value in kwargs.items():
        print(f"{key}: {value}")
person_info(1,name = "Alice", age = 30, city = "New York")
```

代码运行结果：

```
---- 1 -----
name: Alice
age: 30
city: New York
```

其中，实参 1 赋值给形参 n，** kwargs 接收多个关键字参数，将它们收集到一个字典中。这些可变数量参数使函数能够处理各种不同参数数量的情况，从而提高了函数的灵活性。

5.3 变量的作用域

变量的作用域就是指变量的有效范围。变量按照作用范围分为两类：全局变量和局部变量。全局变量是在函数外部定义的变量，作用范围是从定义点到程序结束。局部变量为

函数内部定义的变量，包含在 def 定义的语句块中，只有在函数内部起作用，当退出函数时变量将被释放。

【例 5.4】 局部变量与全局变量的使用。

```
def func():
    m = 66
    print("m inside func:",m)
    print("n inside func:",n)
m = 1
n = 2
func()
print("m outside func:", m)
print("n outside func:", n)
```

代码运行结果：

```
m inside func: 66
n inside func: 2
m outside func: 1
n outside func: 2
```

程序说明：

(1) m=1 在全局作用域中添加了一个名为 m 的变量并赋值 1；n=2 在全局作用域中添加了一个名为 n 的变量并赋值 2，这两个变量都被称为全局变量。

(2) 函数内部的 m=66 在函数的局部作用域中添加了一个变量并赋值 66，这个变量被称为局部变量。输出结果证明，局部变量的 m 被赋值完全不影响全局变量 m 的值，两者是相互独立的。

(3) 在函数内部访问某个名字时，会先在局部作用域中查找，如果有，则使用局部作用域中的那一个；如果没有，则尝试在全局作用域中找，如本例中的 print("n inside func:", n)。如果全局作用域中仍找不到，则会报错。

(4) 函数的形参也属于局部作用域。

(5) 局部变量依赖创建该变量的函数是否处于活动的状态，函数调用时创建，函数调用结束后销毁该变量并释放内存。例如：

```
def f():
    x = 1 #函数内部定义的局部变量,函数外部不能访问
    print(x)
f()
print(x) #NameError: name 'x' is not defined
```

x 是在函数 f()内部创建的局部变量，只能在函数内部访问，在函数外部访问该变量时会触发 NameError 异常。

在函数内部定义的变量，除非特别声明为全局变量，否则均默认为局部变量。当需要在函数体内声明一个可以在函数体外访问的全局变量时，可以使用 global 关键字来声明变量的作用域为全局。global 的作用就是把局部变量提升为全局变量。例如，将例 5.4 的代码修改如下：

```
def func():
    m = 66
    global  n                    #声明 n 为全局变量
    n = 77                       #函数内定义全局变量会屏蔽函数外的同名变量
    print("m inside func:",m)
    print("n inside func:",n)
m = 1
n = 2
func()
print("m outside func:",m)
print("n outside func:",n)       #调用 func()函数后,n 的值为 77
```

代码运行结果：

```
m inside func: 66
n inside func: 77
m outside func: 1
n outside func: 77
```

在函数的内部，通过 global n 将 n 提升为全局变量，语句 n＝77 声明了一个新的变量，屏蔽了函数外的同名变量。在函数调用后，再次访问变量 n 时，访问的是最近在函数内部声明的新全局变量 n，其值为 77。

当全局变量值为列表等可变数据类型，函数内部需要修改变量值时，不需要使用 global 关键字进行声明，直接可以使用。这是因为列表等可变数据类型的值的修改是在内存中进行的，只有显式声明才会重新创建对象。

【例 5.5】 局部变量与全局变量的使用。

```
ls = ["hello","world"]
def f(a):
    ls.append(a)
    return
f("good")
print(ls)
```

代码运行结果：

```
['hello', 'world', 'good']
```

程序说明：在本例中，首先创建一个列表 ls，ls 为全局变量。在函数 f()内部没有创建 ls，但是把 ls 当作列表类型使用，追加元素值"good"。这种情况下，函数内部的 ls 就等同于全局变量 ls。调用该函数，将字符串"good"增加到 ls 中，输出结果表示列表 ls 在函数 f()的内部从两个元素增加为三个元素。

5.4 匿名函数

匿名函数是一个没有函数名字的临时使用的函数，在 Python 中使用 lambda 关键字创建匿名函数，通常是在函数式编程中直接使用 lambda 函数。

lambda 函数形式如下。

```
<函数名> = lambda  <参数列表>:<表达式>
```

lambda 函数可以等价替换成 def 定义的函数。冒号“:”之前可以有 0 个或多个参数。上面的语句等价于下面的函数定义。

```
def  <函数名>  (参数列表):
    return  <表达式>
```

Python 提供了很多函数式编程的特性，如 sorted()、map()、reduce()、filter()等，这些函数都支持函数作为参数，lambda 函数可以应用在函数式编程中。

sorted()函数的用法如下：

```
sorted(iterable, * ,key = None,reverse = False)
```

其中，sorted()函数中的 key 可以接收函数，包括自定义函数、内置函数和 lambda 函数等，并以函数返回值为排序依据。

【例 5.6】 lambda 函数的使用。

```
ls = [ - 5,6, - 9,8,1]
#按整数数值升序排序
print(sorted(ls))
#按列表各元素的平方升序排序
print(sorted(ls,key = lambda x:x ** 2))
ls = ['hello', 'i', 'abcd', 'am', 'abc']
#按字符串升序排序
print(sorted(ls))
#按字符长度升序排序
print(sorted(ls,key = lambda x:len(x)))
```

代码运行结果：

```
[ - 9, - 5, 1, 6, 8]
[1, - 5, 6, 8, - 9]
['abc', 'abcd', 'am', 'hello', 'i']
['i', 'am', 'abc', 'abcd', 'hello']
```

注意：lambda 函数不需要 return 来返回值，表达式本身的计算结果就是函数的返回值。lambda 的主体是一个表达式，而不是代码块，仅能在表达式中封装有限的逻辑，不允许包含其他复杂的语句，最多只能用于类似条件表达式这样的三元运算。

filter()是一个内置函数，用于从可迭代对象中筛选符合条件的元素，返回一个迭代器。它是函数式编程中常用的工具，它的核心作用是根据指定的条件过滤数据。其用法如下：

```
filter(function, iterable)
```

(1) function：过滤条件的函数。若为 None，则直接使用元素的真值(Truth Value)过滤。

(2) iterable：待处理的可迭代对象(如列表、元组、字符串等)。

(3) filter()函数返回的是一个迭代器,也可以用 * 解包,也可以用 list()函数转换为列表输出。

filter()函数与 lambda 函数结合,可以用一行代码实现把列表[8,3,2,7,9]中的奇数过滤出来的功能。

```
print( list(filter(lambda x:x % 2, [8,3,2,7,9])) )
```

输出结果:

```
[3, 7, 9]
```

5.5 函数的递归

递归(recursion)是一种直接或间接调用函数自身的算法,其实质是把问题分解成规模缩小的同类子问题,然后递归调用来表示问题的解。

例如,数学中:

$$5!=5\times4\times3\times2\times1=5\times(4\times3\times2\times1)=5\times4!$$

可以被抽象成:

$$\begin{cases}1 & n=1\\ n!=n(n-1)! & n>1\end{cases}$$

求 4 的阶乘与求 5 的阶乘是相同性质的问题,其区别仅在于问题的规模不同(即参数大小不同)。如果我们定义了一个函数 fact(n)可以求出 n 的阶乘,那么理论上,fact(n-1)可以求出 n-1 的阶乘。

阶乘的例子揭示了递归的两个关键特征。

(1) 存在一个或多个基例,基例不需要再次递归,它是确定的表达式。例如,当 n=1,n!=1 是已知的值,这就是一种基例,与其他值不存在递归的关系。

(2) 计算过程中存在递归链条,如 n!和(n-1)!构成了递归的链条。所有递归链要以一个或多个基例结尾。

递归也叫数学归纳法。

证明当 n 取第一个值 n_1 时,命题成立。

假设当 $n=n_k$ 时,命题成立,证明当 $n=n_{k+1}$ 时命题也成立,递归是数学归纳法思想的编程体现。

在数学中,通过函数自身来定义的函数称为递归函数。如阶乘、斐波那契数列等问题,用递归函数来解决,可以用较少的代码完成。

【例 5.7】 定义递归函数,求 n!。

```
def fact(n):
    print(f"fact ({n}) is called.")
    if n == 1:
        return 1
    return n * fact(n - 1)
print("5!= ", fact(5))
```

代码运行结果：

```
fact (5) is called.
fact (4) is called.
fact (3) is called.
fact (2) is called.
fact (1) is called.
5!= 120
```

程序说明：

(1) fact()函数内部调用了 fact()函数自身，这是一个递归函数。

(2) 如果 n=1，则直接返回结果 1。此处，称 n=1 为递归的终止条件或基例，这个终止条件保证了对于合法的 n 值，这个递归函数一定会运行结束。

(3) fact(5)的执行过程可以这样理解：为了求 5 的阶乘，函数调用函数自身求 4 的阶乘，为了求 4 的阶乘，函数调用自身求 3 的阶乘……函数调用自身求 1 的阶乘，1 的阶乘满足终止条件，返回结果 1。得到了 1 的阶乘，fact(2)通过 n=2 * 1=2 得到了 2 的阶乘并返回。得到了 2 的阶乘，fact(3)通过 n=3 * 2=6 得到了 3 的阶乘并返回……然后得到了 4 的阶乘为 24，fact(5)通过 n=5 * 24=120 得到 5 的阶乘，并返回给外部调用者。

(4) 在 fact(1)函数被执行时，整个解释器内实际上有 5 个 fact()函数正在执行，分别是 fact(5)→fact(4)→fact(3)→fact(2)→fact(1)。fact(1)执行完毕，返回值到 fact(2)，fact(2)得到 fact(1)的返回值，计算后返回给 fact(3)……最终 fact(5)在得到 fact(4)的返回值后，再将计算结果返回给外部调用者。

有关递归的实现，需要注意以下几点。

(1) 递归本身是一个函数，需要使用函数定义的方式描述。

(2) 递归的实现需要函数与分支语句。

(3) 函数内部采用分支语句对输入参数进行判断。

(4) 递归函数必须设置一个出口(终止条件)，即不能无限递归。递归深度同时受操作系统栈的深度限制，不同系统环境下支持的最大递归深度不同。在 64 位 Windows 10 环境下，最大递归深度约为 3900 次。

【例 5.8】 汉诺塔。

这是来自印度的古老的传说。有一个地方有三根柱子，如图 5.2 所示，在最左侧的柱子上放了一组圆盘，圆盘有大有小，需要将这组圆盘所形成的塔形状移到三根柱子的最右侧的柱子上，但是在移动的过程中需要有一些规则来约束，即小的圆盘永远放在大的圆盘上面。

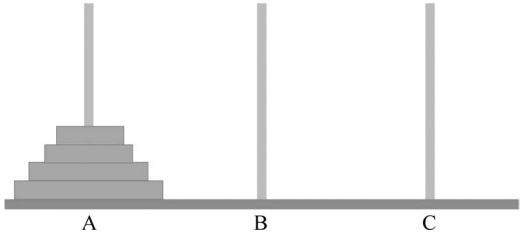


图 5.2　汉诺塔

假设最左侧只有两个圆盘，小的放在大的上面，那么如何将这两个圆盘移到最右侧的柱

子上呢？

（1）将小的圆盘移到中间的圆柱。

（2）将大的圆盘移到最右侧的圆柱上。

（3）将中间小的圆盘移到最右侧的圆柱上。

这样就实现了这种塔状的圆盘从左移到右侧，那如何通过递归来实现汉诺塔呢？

汉诺塔的实现过程中，对于给定数量的圆盘，从最左侧移到最右侧需要多少步骤？如何搬运？

初始状态，所有圆盘都放在 A 上面，经过中间柱子 B，最后到达 C。

定义一个函数：hanoi(n,src,dst,mid)，共有 4 个参数，第一个参数 n 是圆盘数量，第二个参数 src 是源柱子，第三个参数 dst 是目标柱子，第四个参数 mid 是过渡柱子。

```
count = 0
def hanoi(n,src,dst,mid):
    global count
    if n == 1:
        print("{}:{}->{}".format(1,src,dst))
        count += 1
    else:
        hanoi(n-1,src,mid,dst)
        print("{}:{}->{}".format(n,src,dst))
        count += 1
        hanoi(n-1,mid,dst,src)
```

如果当前圆盘只有一个，很容易从源位置移到目标位置，用 print()和 format()输出移动过程。

（1）输出当前圆盘尺寸，是最小圆盘？还是哪一个尺寸的圆盘？

（2）输出从哪个源位置到哪个目标？

为了计算每一次移动圆盘的步骤，首先定义一个全局变量 count，因为递归函数本身也是函数，所以内部的变量是局部变量，每次调用时会被清零；每移动一次就增加 1。

基例：n=1 时，打印移动步骤并计数。

那么递归的链条呢？

假设 n 个圆盘从 A 到 C 的移动步骤如下。

（1）将 A 上 n−1 个圆盘从 A 移到 B。

（2）将 A 中剩下的一个圆盘从 A 移到 C。

（3）将 B 上的 n−1 个圆盘从 B 移到 C。

那么 n−1 个圆盘如何移动呢？

递归的问题只关心递归的链条。当圆盘数量为 n 时，如何拆解为当前 n 与当前 n−1 之间的关系？至于 n−1 暂时先不管。

把 n 个圆盘从 A 移到 C，递归链条：hanoi(n−1,src,mid,dst)。只需：

（1）将 n−1 个圆盘从 A(src)移到 B(mid)，移动过程使用 C(dst)作为过渡。

（2）将剩余的最后 1 个圆盘从 A 移到 C。

（3）将 n−1 个圆盘从中间 B(mid)移到 C(dst)，过程中使用 A(src)作为过渡。所以，通过对递归链条的描述，就能完成汉诺塔的定义。

再看一下函数代码，其实我们并没有一步一步地拆解移动汉诺塔的步骤，只是将递归的链条和递归的基例弄清楚，通过函数加分支结构表达出来，就可以去执行这段程序，并且会告知整个运行的具体结果。

只有三个圆盘的移动过程：假设做一个简单的汉诺塔，只有三个圆盘，从柱子 A 移动到 C，中间过渡的柱子为 B。

```
count = 0
def hanoi(n,src,dst,mid):
    …
hanoi(3,"A","C","B")
print(count)
```

【分析】

(1) 1:A→C　＃将 1＃盘从 A 移到 C

(2) 2:A→B　＃将 2＃盘从 A 移到 B

(3) 1:C→B　＃将 1＃盘从 C 移到 B，通过(1)～(3)三步将 1＃、2＃两个盘从 A 移到 B

(4) 3:A→C　＃将 3＃盘从 A 移到 C

(5) 1:B→A　＃第(5)和(6)两步将 B 中两个盘从 B 移到 C

(6) 2:B→C

(7) 1:A→C

实现汉诺塔的完整代码：

```
count = 0
def hanoi(n,src,dst,mid):
    global count
    if n == 1:
        print("{}:{}->{}".format(1,src,dst))
        count += 1
    else:
        hanoi(n-1,src,mid,dst)
        print("{}:{}->{}".format(n,src,dst))
        count += 1
        hanoi(n-1,mid,dst,src)
n = int(input("Input n:"))
hanoi(n,"A","C","B")
print(count)
```

代码运行结果：

```
Input n:3
1:A->C
2:A->B
1:C->B
3:A->C
1:B->A
2:B->C
1:A->C
7
```

只有用抽象的方式理解了递归链条的表达关系，才能自如地运用递归的方式来解决所面对的计算问题。

5.6 Python 内置函数

Python 解释器提供了内置函数可以直接使用，通过 dir(__builtins__)命令查看所有的内置函数和内置常量名，其中，大写字母开头的为内置常量名。

```
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BaseExceptionGroup',
'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'ChildProcessError',
'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetError',
'DeprecationWarning', 'EOFError', 'Ellipsis', 'EncodingWarning', 'EnvironmentError', 'Exception',
'ExceptionGroup', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError',
'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError',
'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt',
'LookupError', 'MemoryError', 'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError',
'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning',
'PermissionError', 'ProcessLookupError', 'RecursionError', 'ReferenceError', 'ResourceWarning',
'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'SyntaxError',
'SyntaxWarning','SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError',
'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError',
'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError',
'__build_class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__',
'__spec__', 'abs', 'aiter', 'all', 'anext', 'any', 'ascii', 'bin', 'bool', 'breakpoint', 'bytearray',
'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr',
'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec', 'exit', 'filter', 'float', 'format', 'frozenset',
'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance',
'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next',
'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'repr', 'reversed',
'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type',
'vars', 'zip']
```

Python 3.9 提供了 69 个内置函数，如下所示。

abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__
complex()	hasattr()	max()	round()	

这些函数中的大部分会在本书的各章节出现，本节只介绍几个常用函数。

1. id(object)

id()函数返回括号中对象的内存地址，一个对象的 id 值在解释器中就代表它在内存中的首地址。用 is 判断两个对象是否相同时，依据的就是这个 id 值是否相同。对于字符串、整数等类型，变量的 id 是随着值的改变而改变的。列表等复合类型的对象 id 唯一且不变，但在不重合的生命周期里，可能会出现相同的 id 值。

2. zip(seq[,seq,…])

调用 zip()函数时，可把两个或多个序列中的相应项合并在一起，返回由这些元素组成的可迭代对象，在处理完最短序列中的所有项后就停止。注意，zip()函数不能直接输出组合后的数据，可以通过 list()函数转换为列表再输出。例如：

```
>>> lst1 = [1,2,3]
>>> lst2 = [4,5,6]
>>> lt = zip(lst1,lst2)
>>> print(lt)
< zip object at 0x000002002E1FFE80 >
>>> print(list(lt))
[(1, 4), (2, 5), (3, 6)]
```

3. filter(function, iterable)

调用 filter()函数时，把函数 function 作用于序列 iterable 中的每个元素，然后根据函数返回值是 True 或 False 判断保留还是丢弃该元素，保留返回真值的所有项，过滤掉返回假值的所有项，最后返回一个迭代器对象。如果要转换为列表，则可以使用 list()函数来转换。例如：

```
def is_odd(n):
    return n % 2 == 1
tmplist = filter(is_odd, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
print(list(tmplist))
```

代码运行结果：

```
[1, 3, 5, 7, 9]
```

filter()函数中的 function 也可以为匿名函数。

4. map(function, iterable, …)

将传入的函数 function 作用到序列中的每个元素，并将结果组成新的迭代器对象。例如：

```
def f(x):
     return x * x
lt = map(f, [1, 2, 3, 4, 5, 6, 7, 8, 9])
lst = list(lt)
print(lst)
```

代码运行结果：

```
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

微实践——兔子繁殖问题

斐波那契在《计算之书》中提出了一个有趣的兔子问题：如果有一对小兔，每个月都生下一对小兔，而所生下的每一对小兔在出生后的第三个月也都生下一对小兔，那么，由一对兔子开始，满一年时一共可以繁殖成多少对兔子？

第一个月小兔子没有繁殖能力，所以还是一对；两个月后，生下一对小兔，总共有两对；三个月以后，老兔子又生下一对小兔，因为小兔子还没有繁殖能力，所以一共是三对；……以此类推，可以列出表 5.1 所示的情形。

表 5.1　兔子繁殖

经过月数	0	1	2	3	4	5	6	7	8	9	10	11	12
总体对数	0	1	1	2	3	5	8	13	21	34	55	89	144

表中数字 1,1,2,3,5,8……构成了一个数列。这个数列有一个十分明显的特点：前面相邻两项之和，构成了后一项。斐波那契对上述规律进行总结和形式化，得到关于 n 个月后兔子数量的通项公式如下。

$$F(n)=\begin{cases}1 & n=1\\ 1 & n=2\\ F(n-1)+F(n-2) & n\geqslant 3\end{cases}$$

程序代码如下：

```
def fib(n):
    if n<=2:
        return  1
    #最近两项的值,a 为前前项,b 为前项
    a,b = 1,1
    for x in range(3,n+1):
        s = a+b                 #新值: 前两项之和
        a,b = b,s               #a=b,b=s
    return s
print(fib(10))
```

代码运行结果：

```
55
```

斐波那契数列也可以用递归方法来实现。当 n=1 或 n=2 时，F(n)的值为 1。其他情况下，使用递归，调用自身，得到 F(n−1)和 F(n−2)的和，作为 F(n)的值。代码如下：

```
def fib1(n):
    if n==1 or n==2:
        return 1
    else:
```

```
        return fib1(n - 1) + fib(n - 2)
print(fib1(10))
```

数学之美——斐波那契数列的奇妙

（1）自然界中的斐波那契数列。

如果数一朵花的花瓣数，通常会发现总数是斐波那契数列中的一个数。百合花为 3 瓣，梅花为 5 瓣，飞燕草为 8 瓣，万寿菊为 13 瓣，向日葵为 21 或 34 瓣，雏菊有 34、55 和 89 三个数目的花瓣。

观察向日葵中心的种子，会发现它们的种子以两组螺旋的形式排列，一组顺时针转动，另一组逆时针转动，如图 5.3 所示。而这两组螺旋的数量，恰好是相邻的斐波那契数。这种排列方式帮助向日葵以最有效的方式填满空间，使每颗种子都能获得充足的光照。

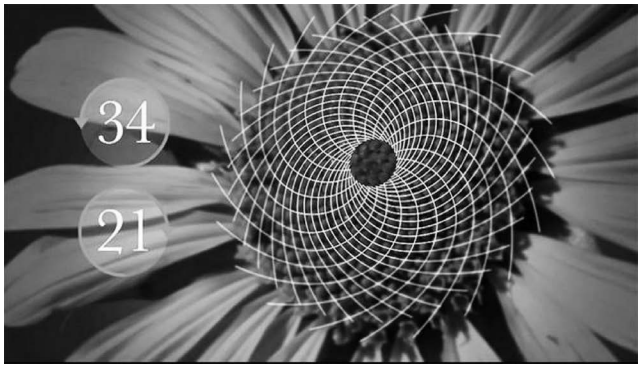


图 5.3　向日葵

（2）黄金分割。

黄金分割是指将整体一分为二，较大部分与整体部分的比值等于较小部分与较大部分的比值，其比值约为 0.618。这个比例被公认为是最能引起美感的比例，因此被称为黄金分割。计算黄金分割最简单的方法，是计算斐波那契数列 1,1,2,3,5,8,13,21,……自第二位起相邻两数之比，即 2/3,3/5,5/8,8/13,13/21……的近似值。

黄金分割具有严格的比例性、艺术性、和谐性，蕴藏着丰富的美学价值，被认为是建筑和艺术中最理想的比例，如图 5.4 所示。

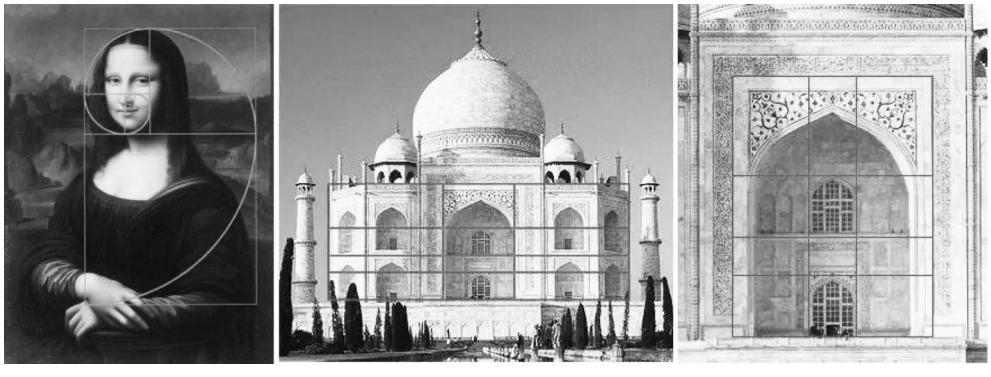


图 5.4　黄金分割

5.7 turtle 库的应用

turtle 库是 Python 语言的标准库之一，属于入门级的图形绘制函数库。其绘图原理为：有一只海龟在窗体画布上游走，走过的轨迹形成了绘制的图形。海龟由程序控制，可以自由改变轨迹的颜色、方向和宽度。

5.7.1 绘图坐标体系

1. turtle 库的导入

使用 turtle 库进行绘图时，需要导入 turtle 库，并使用相关函数。

```
import turtle
```

此时，程序可以调用库中的所有函数。

```
turtle.函数名(函数参数)
```

2. 坐标系

画布就是 turtle 绘图窗口，可以设置它的大小和初始位置。例如，用如下代码设置绘图窗口的大小和位置，如图 5.5 所示。

```
turtle.setup(width, height, startx, starty)
```

width：窗口宽度，如果值是整数，则表示像素值；如果值是小数，则表示窗口宽度与屏幕的比例。

height：窗口高度，如果值是整数，则表示像素值；如果值是小数，则表示窗口高度与屏幕的比例。

startx：窗口左侧与屏幕左侧的像素距离，如果值是 None，则窗口位于屏幕水平中央。

starty：窗口顶部与屏幕顶部的像素距离，如果值是 None，则窗口位于屏幕垂直中央。

以绘图窗体中心为原点，水平向右为 X 轴正方向，垂直向上为 Y 轴正方向，建立平面直角坐标系，这就是 turtle 空间坐标体系。在初始状态时，海龟位于 turtle 空间坐标系原点的位置上，行进方向为水平右方。turtle 坐标系示意图如图 5.6 所示。

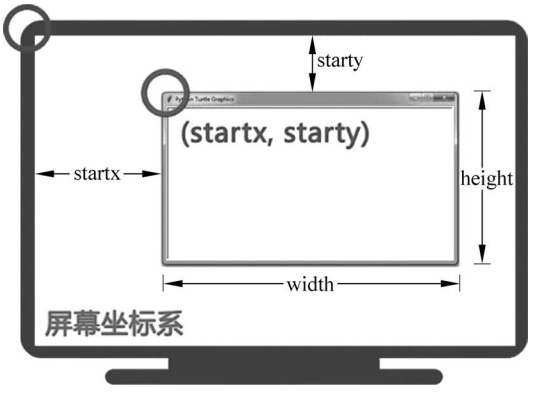


图 5.5 turtle.setup()函数 4 个参数的含义

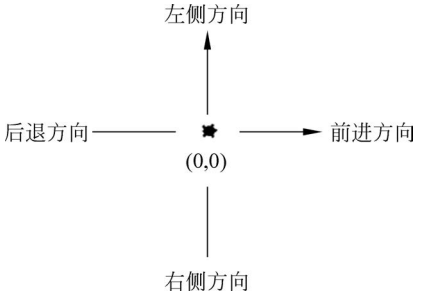


图 5.6 turtle 坐标系示意图

5.7.2 画笔控制函数

turtle 中的画笔(即小海龟)可以通过一组函数来控制。

1. 形状绘制函数

turtle 通过一组函数控制画笔的行进动作：前进、后退、转向等，进而绘制形状，如表 5.2 所示。

表 5.2 画笔运动函数

函 数	说 明
forward(d)或 fd(d)	向当前画笔方向移动 d 像素
backward(d)或 bk(d)	向当前画笔相反方向移动 d 像素
right(degree)	degree 为角度值，向右旋转 degree 度
left(degree)	degree 为角度值，向左旋转 degree 度
setheading(degree)或 seth(degree)	将画笔的朝向设置为指定角度
goto(x,y)	将画笔移动到坐标点为(x,y)的位置
circle(r, degree)	绘制一个指定半径 r、角度 degree 的弧形
speed(s)	设置画笔绘制的速度为 s，s 为[0,10]的整数

turtle 库中的角度坐标体系，以正东向为绝对 0°，即小海龟初始爬行方向。角度坐标体系是绝对方向体系，与小海龟爬行的当前方向无关，可以用于改变小海龟前进方向，如图 5.7 所示。seth(degree)函数中 degree 就是绝对方向角度值；而 right(degree)和 left(degree)函数是相对于海龟的前进方向的右转和左转，degree 是相对方向角度值。

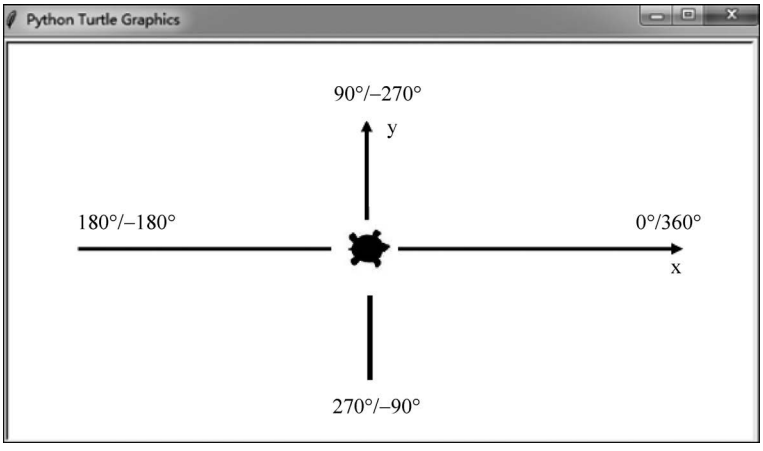


图 5.7 turtle 库的角度坐标系

2. 画笔状态函数

海龟就是一支画笔，可以通过函数设置画笔宽度、颜色、填充等。常用画笔状态函数如表 5.3 所示。

表 5.3 常用画笔状态函数

函　数	说　明
penup()或 pu()	画笔抬起,意味着当它移动时没有线条会被画出来
pendown()或 pd()	画笔落下,默认绘制
pencolor(* args)	设置画笔颜色,参数可以是颜色字符串或 RGB 的值
pensize(width)	设置画笔宽度为 width
color(color1,color2)	设置画笔颜色 color1 和填充颜色 color2,颜色可以是表示颜色的字符串,例如,"red"、"blue"、"purple"等。
begin_fill()	准备开始填充图形
end_fill()	填充完成
hideturtle()	隐藏画笔的 turtle 形状
showturtle()	显示画笔的 turtle 形状
done()	启动事件循环,必须是图形程序中的最后一条语句

【例 5.9】 编写程序,利用 turtle 库绘制一个如图 5.8 所示的红色五角星。

【分析】 五角星每个角和每条边的形状完全相同,边的夹角都是 36°,可以使用循环控制绘制。画笔首先前进一定距离,然后向右旋转 144°,再前进,再旋转,如此循环,直到完成五条边的绘制,从而形成五角星的外形。

图 5.8 五角星效果图

程序代码:

```
import turtle
turtle.color("red")              #设置画笔颜色为红色
turtle.pensize(3)                #设置画笔宽度为 3
for i in range(5):               #循环绘制五角星
    turtle.forward(100)          #向前移动 100 像素
    turtle.right(144)            #向右旋转 144 度
turtle.hideturtle()              #隐藏画笔
turtle.done()                    #显示绘制窗口
```

程序说明:本例中画笔先向前移动 100 像素,再右转 144°,也可以采用其他顺序进行绘制。试着修改画笔的颜色和宽度以及画笔的运动顺序,体会其他方法绘制的过程。

【例 5.10】 绘制如图 5.9 所示的正方形彩色螺旋线。

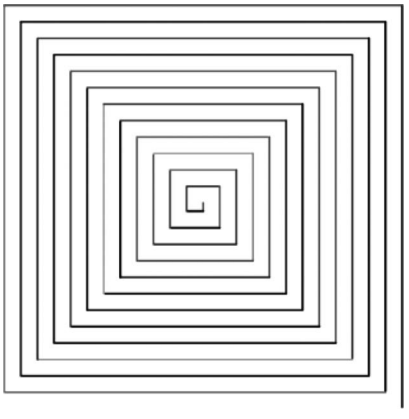


图 5.9 正方形螺旋线效果图

【分析】 在绘制四边形螺旋线时，可以通过控制四条边的长度和旋转角度来实现螺旋线的效果。具体方法：沿着当前方向画一条边，再旋转 90°，改变方向；重复上述步骤，直到达到所需的螺旋线形状，注意绘制的边长长度需逐渐增加。

程序代码如下：

```
import turtle
import random
length = 5
times = 50
angle = 90
for i in range(times):
    r,g,b = random.random(),random.random(),random.random()
    turtle.pencolor(r,g,b)
    turtle.forward(length * i)
    turtle.right(angle)
turtle.hideturtle()
turtle.done()
```

程序说明：

(1) pencolor()函数的参数除了颜色字符串以外，也可以是颜色对应的 RGB 数值(红，绿，蓝)，RGB 每色取值范围 0～255 的整数或 0～1 的小数，例如，(0.1，0.2，0.5)。

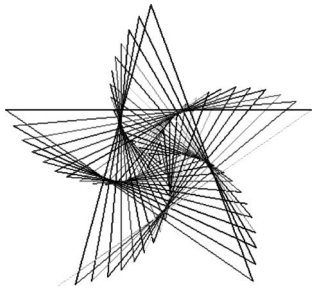
(2) 程序中导入 random 库，利用 random()函数随机生成三个小数 r、g、b，使得每条线段的颜色都是随机生成的。

将上述程序用函数实现，接收参数为每次转角的角度、边长每次递增的长度以及线段迭代的次数。

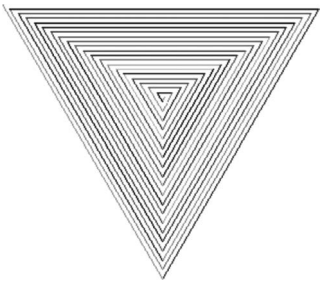
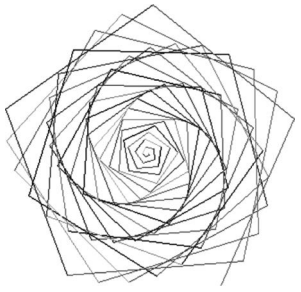
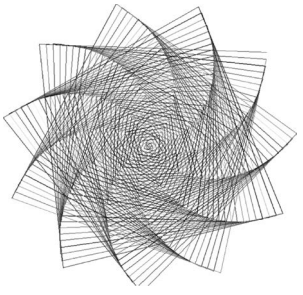
```
import turtle
import random
def drawgraphic(angle,delta = 5,times = 60):
    for i in range(times):
        r,g,b = random.random(),random.random(), random.random()
        turtle.pencolor(r,g ,b)
        turtle.forward(delta * i)
        turtle.right(angle)
    turtle.hideturtle()
    turtle.done()
```

调用函数，通过调整输入的 3 个参数得到有趣的图形，如表 5.4 所示。

表 5.4　调整参数得到有趣的图形

参　数	图　形
drawgraphic(145,5,74)	

续表

参　数	图　形
drawgraphic(120,5,60)	
drawgraphic(74.5,2,104)	
drawgraphic(98,1.5,350)	

【例 5.11】 绘制如图 5.10 所示的同心圆。

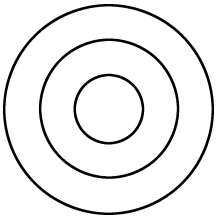


图 5.10　同心圆

【分析】 circle(r, degree)函数的 degree 参数省略时,绘制半径为 r 的圆。

程序代码如下:

```
import turtle as t
def DrawCctCircle(n):
    t.penup()
    t.goto(0, -n)
    t.pendown()
```

```
        t.circle(n)
    for i in range(20,80,20):
        DrawCctCircle(i)
    t.hideturtle()
    t.done()
```

程序说明：

(1) 在 import turtle as t 语句中，t 是 turtle 库的别名。这意味着在 Python 中，可以通过 t 来调用 turtle 库中的所有函数和类，而不需要每次都使用完整的库名 turtle。

(2) 自定义函数 DrawCctCircle(n)的功能：先抬起画笔，将画笔定位在(0，－n)点，然后落下画笔，绘制以 n 为半径的圆。

(3) 调用 DrawCctCircle()函数，分别绘制半径为 20、40、60 的圆，实现同心圆的效果。

本章小结

本章学习了函数的定义与调用方法，还学习了实参和形参以及 Python 中传递参数的不同方式以及利用 turtle 库绘制图形。

(1) 函数是一段具有特定功能的、可重用的代码。函数是功能的抽象。一般来说，每个函数表达特定的功能。函数是带名字的代码块，用于完成具体的工作。要执行函数定义的特定任务，可调用该函数。

(2) 匿名函数是一个没有函数名字的临时使用的小函数，用 lambda 创建的匿名函数经常被用作函数的参数传递，例如，作为排序关键字。

(3) 递归是指在函数的定义中使用函数自身的方法，把规模大的问题转换为规模小的相似的子问题来解决。递归可求解的问题都可以用循环求解。

(4) turtle 库是 Python 语言的标准库之一，属于入门级的图形绘制函数库。其绘图原理为：有一只海龟在窗体画布上游走，走过的轨迹形成了绘制的图形。

习题

一、思考题

1. 使用函数的优点有哪些？
2. 如何定义带有默认值参数的函数？
3. 如何定义带有可变数量参数的函数？
4. 假如 return 语句同时返回 3 个值，返回值是什么数据类型？
5. 参数的位置传递和关键字参数各有什么优缺点？
6. 什么是全局变量和局部变量？
7. 下面代码的输出结果是什么？

```
f = lambda x,y:y+x
print(f(10,10))
```

8. 下列代码的输出结果是什么？

```
def f2(a):
    if a > 33:
        return True
li = [11, 22, 33, 44, 55]
res = filter(f2, li)
print(list(res))
```

9. 下列代码的输出结果是什么？

```
def f(x, y = 0, z = 0):
    print(x,y,z)
f(1,3)
```

10. 下列代码绘制的图形是什么？

```
import turtle
def drawLine(draw):
    turtle.pendown() if draw else turtle.penup()
    turtle.fd(50)
    turtle.right(90)
drawLine(True)
drawLine(True)
drawLine(True)
drawLine(True)
```

二、编程题

1. 实现 isOdd()函数，参数为整数，如果整数为奇数，则返回 True，否则返回 False。

2. 实现 isNum()函数，参数为一个字符串，如果这个字符串属于整数、浮点数或复数的表示，则返回 True，否则返回 False。

3. 实现 multi()函数，参数个数不限，返回所有参数的乘积。

4. 将素数的判定代码定义为一个函数 is_prime(n)，接收传入的实参整数 n，如果 n 是素数，则返回 True，否则返回 False。在主函数中输入整数 m，通过调用 is_prime(n)函数，输出 2～m 中所有的素数。

5. 编写一个四则运算的程序，要求加、减、乘、除各定义为一个函数来实现。

6. 一只青蛙每次可以跳上 1 级台阶或 2 级台阶，跳上 10 级台阶总共有多少种跳法？利用递归方法实现。

7. 编写程序，利用 turtle 库绘制不同形状的图形。