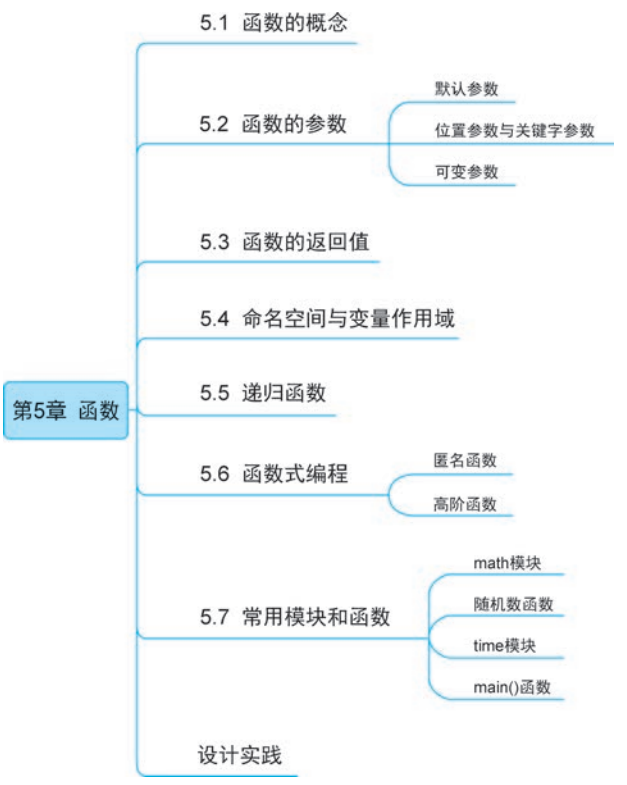
第5章 函　数

CHAPTER 5

章节导图



学习目标

（1）理解函数的定义与相关概念；

（2）掌握函数的调用、参数传递和返回值；

（3）掌握变量作用域、LEGB 原则；

（4）掌握递归函数的编程方法；

（5）掌握匿名函数、函数式编程的概念及用法；

（6）掌握 math 函数、日期函数、随机数函数的使用。


视频讲解

5.1 函数的概念

编程的过程中，经常需要实现重复的功能，解决的最好办法是利用函数。比如前面实现图形打印时，就可以将其设计为函数实现，把打印的行数设置为参数，这样想打印多少行，设置对应的参数就可以了。

简单地说，函数是能够实现特定功能的一组代码，它能够提高代码的模块化和重复利用率。借助于函数可以快速实现一定的功能，使得设计代码结构更加清晰。print()、id()、type()、range()、int()等都是 Python 内置的函数。

下面例子中的代码实现了输入限定行数的三角图形打印功能：

```
#Example5.1 三角形打印
def triangle(n):                          # 打印 n 行三角形
    """打印 n 行三角形图案."""             # 文档字符串,用于函数说明
    for i in range(1,n+1):
        for j in range(i):
            print("*",end=" ")
        print("\r")
triangle(5)                               # 函数调用
```

Python 中，一般采用 def 关键字定义函数，后跟函数名与括号内的参数列表，函数语句从下一行开始，且必须缩进。格式如下：

```
def 函数名(参数列表):
    函数体
```

函数定义之后，就有了一段具有特定功能的代码。该代码并不会自动执行，要想让这些代码能够执行，需要调用函数。调用函数的方式很简单，通过“函数名()”即可完成。

函数的内容称作函数体，是实现函数功能的主体。函数体的整体都要遵循缩进规则。函数体的第一行语句可以使用文档字符串(docstring)，用于函数说明。文档字符串也要缩进，并用三引号引起来。利用文档字符串可以自动生成在线文档或打印版文档，在代码中加入文档字符串是 Python 开发的好习惯。

函数定义后，Python 解释器把函数名与函数对象关联在一起，把函数名指向的对象作为用户自定义函数，用户可以使用其他名称指向同一个函数对象，并访问该函数。下面的例子中定义了变量 t 指向 triangle 函数对象，也可以访问函数。

```
>>> triangle
<function triangle at 0x0000025AB3BC3E20>
>>> t = triangle
```

```
>>> t(3)
 *
 * *
 * * *
>>>
```

上面定义的函数没有返回值语句。可以在函数中使用 return 语句返回一个值给调用方，没有指定返回值的函数相当于返回 None。

比如在第 1 章引用的斐波那契数列函数的例子，使用 return 语句将结果返回给调用方 f100，f100 就具有了返回变量的类型和数值，如下所示：

```
#Example5.2 斐波那契函数
def fib2(n):
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)
        a, b = b, a+b
    return result
f100 = fib2(100)
print(f100)
```

执行结果如下：

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

在定义与使用函数时需要注意以下内容：

(1) 函数名要符合命名规范，在命名规则上与变量名一致，一般用小写单词组成。

(2) 参数列表可以没有参数，也可以有多个参数，参数之间用逗号隔开。

(3) 函数定义不能与其他函数重名，如重名，前一个会被覆盖。

(4) return 语句返回函数的值。

(5) return 语句不带表达式参数时，返回 None。函数执行完毕退出也返回 None。

5.2 函数的参数


视频讲解

通常，将定义函数时设置的参数称为形式参数(简称形参)，将调用函数时传入的参数称为实际参数(简称实参)。如 5.1 节中第一个例子中，函数定义 triangle(n)中的参数 n 为形式参数，函数调用语句 triangle(5)中的 5 为实际参数。函数调用语句执行时，实参的值赋给形参。

函数的参数是函数内部与外部交流的纽带，起传递数据的作用，函数调用者可以通过函数参数把函数内部需要的数据从外部传递过去。

下面的例子中为实现计算数字累加和的函数。函数 acc_value 有一个参数 number。调

用函数时，需要输入一个自然数进去，数据传递的过程称为参数传递。示例代码如下：

```
#Example5.3 参数传递
def acc_value(number):                    #number 为参数
    return sum(x for x in range(1,number + 1))
print("1～10 的累加和 = ",acc_value(10))
```

Python 中，函数定义支持可变数量的参数，函数的定义与调用既要考虑参数的数量，也要考虑参数的位置及顺序。函数参数传递属于函数的核心内容。下面分默认参数、位置参数和关键字参数、可变参数三部分进行讲解。


视频讲解

5.2.1 默认参数

函数调用时，通常都涉及参数的传递问题。可以在定义函数的时候为参数指定默认值，这样在调用函数时，可以使用比定义时更少的参数。默认参数可以看作可选参数，例如：

```
#Example5.4 默认参数 1
def person_info(name, age = 35 ):                    # age 为默认参数
    print("Name:{0},Age:{1}".format(name,age))
person_info(name = "Rio")                            # 调用时，未包含默认参数
person_info(age = 20,name = "Rio" )                  # 给出所有参数
```

该例中，age 为默认参数，具有默认值 35，在调用时如果没有该参数的传递，可以取默认值 35；如果给出该参数，则取给定的数值。

需要注意的是：

(1) 参数可以按照默认顺序调用，也可以按照参数名调用。

(2) 带有默认值的参数一定要位于参数列表的最右边，否则程序会报错。

(3) 形参的类型与传入的实参类型一致，函数内部处理时需要注意。

(4) 默认值如果是一个变量，那么其默认值是在函数定义时的定义域内取值。参见下面的代码：

```
#Example5.5 默认参数 2
i = 5
def f(arg = i):
    print(arg)
i = 6
f()
```

这段代码最后的输出结果为 5，程序中的 i 在函数调用之前虽然已经是 6，但在函数定义时 i 的默认值已经取 5，这一默认值就不变了。

(5) 默认值只计算一次。默认值为列表、字典或类实例等可变对象时，会产生与该规则不同的结果。例如，下面的函数会累积后续调用时传递的参数：

```
#Example5.6 默认参数 3
def f(a, L=[]):                    # 参数 L 的默认值为[],空列表
    L.append(a)
    return L
print(f(1))
print(f(2))
```

输出结果如下,结果发生了数据的累积,后续的调用受到前面调用的影响。

```
[1]
[1, 2]
```

如果不想在后续调用之间共享默认值,应以如下方式编写函数,对默认参数加以判断:

```
#Example5.7 默认参数 4
def f(a, L=None):
    if L is None:
        L = []
    L.append(a)
    return L
```

5.2.2 位置参数与关键字参数


视频讲解

Python 中函数参数传递可以分为位置参数传递和关键字参数传递。

1. 位置参数

函数参数列表中的参数有先后顺序,在调用时多个实参依次按顺序传递给对应的形参,实参和形参的数量与顺序一一对应,否则会抛出异常。

下面给出了一个位置参数调用的示例:

```
#Example5.8 未知参数调用
def pos_func(a,b=2,c=3):                    # b,c 为默认参数
    return a,b,c
print(pos_func(1,2,3))
```

该程序中,实参 1 传给了形参 a,实参 2 传递给形参 b,实参 3 传递给形参 c,参数是按照顺序传入的,程序执行的结果如下:

```
(1, 2, 3)
```

2. 关键字参数

在调用函数时,通过参数的名字明确指定给哪个形参传递什么实参,形式为 kwarg=value,这时实参的顺序可以和形参不对应,不影响传递的最终结果。示例如下:

```
#Example5.9 关键字参数
def pos_func(a,b=2,c=3):                    # b,c 为默认参数
```

```
    return a,b,c
print(pos_func(3,c = 5,b = 9))                    # 默认参数也可以按照名字进行参数传递
```

执行结果如下：

```
(3, 9, 5)
```

函数调用时，关键字参数必须跟在位置参数后面。所有传递的关键字参数都必须匹配一个函数接收的参数，关键字参数的顺序并不重要，但不能对同一个参数多次赋值。下面是一些错误的函数参数调用代码示例：

```
pos_func()                    #a 是必选参数，未给出
pos_func(a = 1,2)             #关键字参数，后面不能再使用位置参数
pos_func(2,a = 1)             #参数重复赋值
pos_func(d = 1)               #d 参数不存在
```


视频讲解

5.2.3 可变参数

在 Python 函数中，有时可能需要处理比最初声明还要多的参数，此时可以采用可变参数的形式定义，只需要在参数的前面加符号 * 或 **，如 * args 或 ** kwargs。这些参数叫作可变参数，其中 * args 和 ** kwargs 的区别如下：

（1） * args 会接收所有未命名的变量参数，并放入元组对象，该元组可以包含形参列表之外的所有位置参数，* args 一般称作可变参数。

（2） ** kwargs 会接收以键-值对形式传入的参数，并放入字典对象，该字典包含了函数中已定义形参对应之外的所有关键字参数，并允许通过变量名匹配该字典，** kwargs 一般称作可变关键字参数。

（3）普通关键字参数与 * args 或 ** kwargs 同时使用时，普通关键字参数放在最左边。

（4） * args 参数或者 ** kwargs 参数分别只能出现一次，否则会报错。

若函数没有接收到任何数据，则参数 * args 和 ** kwargs 为空，即它们为空元组或空字典。

【例 5-1】 利用 * args 可变参数编写函数，计算给定的所有整数之和。

```
#Example5.10 计算给定所有数值的和
def number_multi( * num):
    result = 0
    for n in num:
        result += n
    return result
print(number_multi(1,2,3))
```

上述程序中，* num 参数为可变参数，可以接收数量不定的输入参数，这些参数传入函数内部会被看成是 tuple 类型的变量，也就是说函数内部的 num 可以按照 tuple 类型进行

处理，上述程序的执行结果为 6。

【例 5-2】 请设计一个函数 average，用于计算多个数据的平均值。

该函数调用格式示例如下：

```
average(2,3,4)                          # 输出结果为 3.0
average(1,2,3,4,5,6,7,8,9,10)           #输出结果为 5.5
```

参考代码如下：

```
#Example5.11 average 函数
def average( * numbers):
    sum = 0
    for n in numbers:
        sum += n
    return sum/len(numbers)
print("平均值 1 = ",average(2,3,4))
print("平均值 2 = ",average(1,2,3,4,5,6,7,8,9,10))
```

该程序中，* numbers 为可变参数，用以接收数量不定的输入数据，对于输入数据可以按照元组进行遍历，从而计算平均值。

形参为 ** kwargs 形式时，接收一个字典数据。例如，可以定义下面的函数，该函数接收关键字参数 ** kwargs，函数内部有一个字典结构 student_info，其中的键-值对存储了一些默认的键和值，通过调用该函数可以将默认值修改为指定的数值，示例程序如下：

```
#Example5.12 关键字参数
def keyword_test( **kwargs ):
    student_info = {
            '姓名': '某某',
            '性别': '男',
            '年龄': 18, }
    student_info.update(kwargs)
    print(student_info)
keyword_test( 姓名 = '李四', 年龄 = 20 )
```

执行结果如下：

```
{'姓名': '李四', '性别': '男', '年龄': 20}
```

如果关键字参数和可变参数同时存在，则普通参数放在最左边，可变参数放在最右边，如下所示：

```
#Example5.13 同时有可变参数与关键字参数
def arg_test(kw1, * args, **kwargs):
    print("kw1:",kw1)                   # 显示传入的 kw1 参数
    print("args:",args)                 # 显示传入的 args 参数
    print("kwargs:",kwargs)             # 显示传入的 kwargs 参数
arg_test(1,["abc", 1, 2], {"name":"zhang", "age": 20})
```

该程序的执行代码如下：

```
kw1: 1
args: (['abc', 1, 2], {'name': 'zhang', 'age': 20})          # 字典数据也传入了 args
kwargs: {}                                                    # kwargs 参数未传入数据
```

由以上例子可以看出，可变参数 * args 和 ** kwargs 虽然可以一块使用，但是放在后面的 ** kwargs 参数获得的实参数据将为空数据，原因是 * args 参数为可变参数，可以接收可变参数，传入的多个参数都作为元组数据输入 * args 参数了，** kwargs 就只能传入空数据。

定义函数的形式参数可以使用 * 和 ** 表达式，同样实际参数也可以使用 * 和 ** 表达式。如果将 * args 和 ** kwargs 作为实参使用，那么运算符 * 和 ** 可以分别实现拆分可迭代对象和字典对象的功能。

调用函数时，实际参数传递数据使用运算符 * 拆分可迭代对象，而形式参数会以位置参数接收这些可迭代对象的元素。下面看一个示例：

```
#Example5.14 使用运算符 * 拆分可迭代对象
def test(a,b,c,d,e):
    print('number:',a,b,c,d,e)
num = (33,44,55)
test(11,22, * num)
```

执行结果如下：

```
number: 11 22 33 44 55                    # num 实参被拆分
```

由以上例子可以看出，* 运算符将实际参数拆分之后，分别传递给形式参数 c、d、e。

同样地，调用 test()函数时传入一个字典，可以使用 ** 运算符对该字典进行拆分。下面看一个示例：

```
#Example5.15 使用运算符 ** 拆分字典对象
def test(a,b,c,d,e):
    print('number:',a,b,c,d,e)
num = {'c':33,'d':44,'e':55}
test(11,22, ** num)
```

执行结果如下：

```
number: 11 22 33 44 55                    # num 实参被拆分
```

由以上例子看出，** 运算符拆分字典时，键要作为形式参数的名称，用来接收字典对象的值。

5.3 函数的返回值


视频讲解

所谓“返回值”，就是程序中函数完成一件事情后反馈给调用者的结果，函数的返回值是使用 return 语句得到的。

如下面的函数：

```
#Example5.16 函数返回值
def add_test(a, b):
    s = a+b
    return s
x = add_test(1,2)
```

return 用于返回结果数据给调用者 x。如果函数没有返回值，return 可以忽略不写；如果返回值只有一个，则直接返回原类型；如果返回值是多个，则返回的结果为元组。

如下函数返回多个值：

```
#Example5.17 函数有多个返回值
def test():
    name = "zhangsan"
    age = 18
    return name,age                    # 也可以写成 return (name,age)
print(test())
```

该程序执行结果如下，为一元组对象。如写成 return (name,age)，执行效果相同，也是返回元组对象。

```
('zhangsan', 18)
```

5.4 命名空间与变量作用域


视频讲解

命名空间与作用域是程序设计中的基础概念。了解定义的变量的命名空间和作用域，有助于减少代码中隐藏问题的发生。

1. 命名空间

命名空间(Namespace)，也称为名字空间，提供了从名字到对象的映射关系。不同的命名空间是相互独立的，不同命名空间的变量即使名字相同也没有关系。

Python 的命名空间有 built-in、global 和 local 三种。built-in 称为内置命名空间，是 python 自带的内建命名空间，任何模块均可以访问，存放内置的函数和异常；Global 称为全局命名空间，在模块加载时定义，记录模块的变量、函数、类等；Local 称为局部命名空间，是指在函数或类内部所拥有的命名空间。

不同命名空间根据其创建及退出的机制不同而具有不同的生命周期。

2. 变量作用域

变量作用域是指能够访问该变量的代码范围，决定变量可以被引用的命名空间。函数、类和模块内部会产生新的作用域，变量作用域与其在代码中的位置有关。在 Python 程序中访问一个变量，会从内到外依次访问所有命名空间，否则会产生未定义的错误。变量作用域决定了不同程序可以访问的变量范围。

在 Python 中访问一个变量，需要遵循 LEGB 规则进行顺序搜索。LEGB 也称为 Python 中的四种变量作用域，这四种作用域的含义如下：

(1) L(local)：局部作用域，如函数、方法或类内部定义的变量。

(2) E(enclosing)：外层嵌套函数区域，常见的是上一层函数作用域。

(3) G(global)：全局作用域，就是模块级别定义的变量。

(4) B(built-in)：系统内建作用域，是 Python 解释器系统自带变量，比如 int、str 等。

变量的作用域可以通过下面的代码进行简单示例：

```
#Example5.18 变量作用域
g_count = 1                    # 全局作用域 G
def outer():
    o_count = 2                # 函数外的函数中 E
    def inner():
        i_count = 6            # 局部作用域 L
```

3. 全局变量与局部变量

在程序中，变量有全局变量和局部变量之分。全局变量在模块内，但位于函数之外，这类变量拥有全局作用域。全局变量可以在整个程序范围内访问。如果出现全局变量和局部变量名字相同的情况，则在函数中访问的是局部变量。

所谓局部变量，就是在函数内部定义的变量，这类变量只在定义它的函数中有效，一旦函数结束就会失效。下面的程序中，函数 sum 内部的 total 变量和函数外部的 total 变量分别为局部变量和全局变量，二者互不影响，所以最后执行的结果为 30 和 0。

```
#Example5.19 全局变量与局部变量
total = 0                      # total 为全局变量
def sum(arg1, arg2):
    total = arg1 + arg2        # 此处的 total 为局部变量
    print(total)
    return total
sum(10, 20)
print(total)
```

4. global 和 nonlocal 关键字

当在程序中，局部空间需要使用外层空间定义的变量，或者外层空间需使用局部空间定义的变量，这时就需要使用 global 和 nonlocal 关键字。

global 关键字用来在函数或其他局部作用域中使用全局变量，在使用的地方用 global 声明该变量是在函数外定义的全局变量。

下面的程序是对 global 全局变量的使用进行的说明：

```
#Example5.20 global 关键字
gcount = 0                  #gcount 为全局变量
def global_test():
    global gcount           #将 gcount 声明为全局变量，说明该变量是在函数外定义
    gcount += 1
    print(gcount)
global_test()
```

nonlocal 关键字用来在函数或其他作用域中使用外层（非全局）定义的变量，nonlocal 可以当存在函数嵌套，内层函数需要访问外层函数中定义的变量时使用，下面的例子是对该关键字进行的说明：

```
#Example5.21 nonlocal 关键字
def test():
    count = 1                   #局部变量
    def func_in():
        nonlocal count          #声明 count 为上一层变量
        count = 12
    func_in()
    print(count)
test()
```

上述代码的执行结果为 12，如果去掉 nolocal 语句，则输出为 1。

5.5 递归函数


视频讲解

递归函数是实现递归算法的函数，通过直接或者间接调用函数自身求解问题。递归算法的实质是将一个问题不断分解为规模缩小的子问题，然后通过递归调用方法表示问题的解。数学上常见的阶乘、斐波纳契数列、二叉树等计算问题都可以使用递归求解。

首先看一下使用递归函数计算 n 的阶乘的例子，该函数的实现代码如下：

```
#Example5.22 利用递归计算 n 的阶乘
def fac(n):
    if n == 1:
        return 1
    return n * fac(n - 1)                   #递归调用本身
print(fac(4))
```

上述代码输出结果为 24。为了理解递归程序的执行原理，看清楚递归调用的过程，可以将上述程序修改一下，在程序中设置打印语句把每一步的计算过程打印出来，示例代码

如下：

```
#Example5.23 利用递归计算 n 的阶乘
def fac(n):
    if n == 1:
        print("fac(%d)返回 1" %n)                    # 打印返回值
        return 1
    else:
        print("计算 %d*fac(%d-1)" %(n,n))            # 打印正在计算的阶乘
        f = n*fac(n-1)                               # 递归调用
        print("fac(%d)返回 %d" %(n,f))               # 打印已经完成的返回值
        return f
fac(4)
```

运行结果如下：

```
计算 4*fac(4-1)
计算 3*fac(3-1)
计算 2*fac(2-1)
fac(1)返回 1
fac(2)返回 2
fac(3)返回 6
fac(4)返回 24
```

从运行结果可以看出，递归调用是逐层调用的，如图 5-1 所示。递归的过程可看成是将问题逐层向下转化成下一级的计算问题，也就是说一直要找到能够得到计算结果的层级。本例中递归过程为：fac(4)→fac(3)→fac(2)→fac(1)，fac(1)可以直接计算出结果，fac(1)也是本例的递归出口，之后便可以进行反向求值过程，将各层级自底向上都计算出结果，从而实现递归调用的过程。

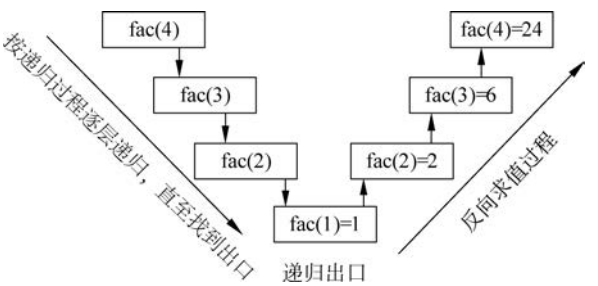


图 5-1　递归调用过程

由以上例子可以看出，用递归函数时，需要找到递归关系和递归出口，递归关系需要建立起步骤 n 与步骤(n－1)的数学关系式，递归出口需要找到最底层的初始值。本例中的递归关系和递归出口为：

(1) 递归关系：fac(n)＝fac(n－1)＊n。

(2) 递归出口：fac(1)＝1。

理解了这些关系，就很容易完成递归函数的编程了。为了巩固递归函数的编程，下面举一个类似的简单例子进行说明。

【例 5-3】 利用递归函数计算 1～n 的数字之和，并调用该函数计算 1～10 的数字之和。

本例可以仿照前面利用递归函数实现阶乘的方法，假设要实现的函数名为 summ(n)，首先要找出递归关系和递归出口，即：

(1) 递归关系：summ(n)＝summ(n－1)＋n。

(2) 递归出口：summ(1)＝1。

这样就将很容易将该递归函数写出，如下所示：

```
#Example5.24 利用递归计算 1～n 的数字和
def summ(n):
    if n == 1:
        return 1
    return n + summ(n-1)                    # 递归调用
print(summ(10))
```

【例 5-4】 利用递归函数实现斐波那契数列。

在 1.4.1 节利用 while 循环实现了斐波那契数列 fib(n)，输出小于 n 的斐波那契数列。斐波那契数列为：0，1，1，2，3，5，8，13，21，34，55，…，首先可以递归方法求出第 n 个斐波那契数，假设计算第 n 个斐波那契数的函数名为 fib_num(n)，则递归关系和递归出口可以表述如下：

(1) 递归关系：fib_num(n) ＝ fib_num(n－1) ＋ fib_num(n－2)。

(2) 递归出口：fib_num(0) ＝ 0，fib_num(1) ＝ 1。

基于上述分析，可以写出 fib_num(n)的递归实现，如下所示：

```
#Example5.25 递归计算第 n 个斐波那契数
def fib_num(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib_num(n-2)+fib_num(n-1)                    # 递归调用
print(fib_num(10))
```

上述程序的最后一条语句调用该 fib_num()函数计算第十个斐波那契数，运行结果为 55。将以上程序稍做改变就可以计算出 n 以内的斐波那契数列，示例代码如下：

```
#Example5.26 递归计算小于 m 的斐波那契数列
def fib_seq(m):
    def fib_num(n):
        if n == 0:
            return 0
```

```
        elif n == 1:
            return 1
        else:
            return fib_num(n-2) + fib_num(n-1)                    # 递归调用
    fibo_list = []
    i = 0
    while fib_num(i) < m:
        fibo_list.append(fib_num(i))
        i += 1
    return fibo_list
print(fib_seq(200))
```

该程序利用 while 循环，将计算得到的每个斐波那契数写入 fibo_list 列表，最后运行结果为：

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
```

5.6 函数式编程

Python 支持函数式编程，允许把函数本身作为参数传入另一个函数，还允许返回一个函数。函数式编程的主要思想是把运算过程尽量写成一系列嵌套的函数调用。函数式编程可以使代码更加简洁，易于理解。

函数式编程将一个问题分解成一系列函数。在函数式程序里，输入会流经一系列函数，每个函数接收输入并输出结果。函数式编程使得程序更加模块化，函数更加明确，更易于编写，也更易于阅读和检查错误，易于程序调试及测试。前面章节中讲到的迭代器、推导式、生成器等都属于函数式编程的范畴，本节是对这些内容的补充。


视频讲解

5.6.1 匿名函数

匿名函数就是没有名称且不使用 def 语句定义的函数。匿名函数需要使用 lambda 关键字声明，主要格式如下：

```
lambda 参数: 表达式
```

匿名函数的简单示例如下：

```
lambda a, b: a + b
```

该匿名函数返回两个参数的和，a 和 b 是两个参数，a+b 就是该函数要实现的功能表达式，参数与表示式之间为冒号。使用 lambda 关键字可以创建简单的匿名函数。

匿名函数只能有一个表达式，表达式的值就是返回值，不需要 return 语句。关键字 lambda 表示匿名函数，lambda 表达式只能包含一个表达式，不允许包含选择、循环等语法

结构。匿名函数不能共享给其他程序使用，只能实现简单的功能。

【例 5-5】 利用 lambda 表达式编写匿名函数，计算给定两个数的平方和，并调用该函数计算 6 和 8 的平方和。

参考代码如下：

```
#Example5.27 lambda 匿名函数
sqr_sum = lambda x,y: x ** 2 + y ** 2
print(sqr_sum(6,8))
```

lambda 匿名函数可以用在函数中作为 return 表达式的返回函数，如下述示例所示。返回的是一个匿名函数，f 即为该函数。匿名函数内部的 n 已经取值为 42，参数 x 依次取值为 0、1，即 f(0)的结果实现的是 0＋42，f(1)实现的是 1＋42。示例代码如下：

```
>>> def make_incrementor(n):
... return lambda x: x + n
...
>>> f = make_incrementor(42)
>>> f(0)
42
>>> f(1)
43
```

lambda 匿名函数使用时，还可以把匿名函数用作传递的实参，如下述示例程序所示，该段代码实现了将 pairs 按照对象元素中元组的第一个元素进行升序排列。

```
>>> pairs = [(1, 'one'), (3, 'three'), (2, 'two'), (4, 'four')]
>>> pairs.sort(key = lambda pair: pair[0])
>>> pairs
[(1, 'one'), (2, 'two'), (3, 'three'), (4, 'four')]
>>>
```

5.6.2 高阶函数


视频讲解

Python 中，函数与其他数据类型(如整数类型)处于平等地位，可以将函数赋值给变量，也可以将其作为参数传入其他函数，将它们存储在其他数据结构(如字典)中，并将它们作为其他函数的返回值。

Python 中，高阶函数是指可以接收另一个函数作为参数的函数。python 里内置了一些高阶函数，其中比较常用的有 map()、reduce()、filter()、sorted()等。sorted()函数在第 4 章已经讲过，下面重点讲解其他几个函数。

1. map()函数

Python 中，map()函数会根据提供的函数对指定的序列做映射。map()函数的定义如下：

```
map(f, iterA, iterB, ...)
```

其中，f 是函数的名称，后面的参数 iterA、iterB 等支持可迭代(iterable)的对象。map 函数将传入的函数依次作用到序列的每个元素，返回一个遍历序列的迭代器。

map()函数的示例代码如下。该程序的 f 函数采用 lambda 匿名函数对 x 求 3 次方，后面跟的可迭代对象是一个列表，将该匿名函数依次作用到列表中的每个元素，实现了对列表的每个元素求 3 次方，并将结果作为新的序列返回。

```
>>> list1 = list(map(lambda x : x ** 3,[1,2,3,4]))                    # map 函数映射
>>> print(list1)
[1, 8, 27, 64]                                                        # 返回列表
>>>
```

【例 5-6】 下面列表 list1 的元素由数字构成，请使用 map()函数将列表中的每个元素转化为字符串。

```
list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

参考代码如下：

```
>>> list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list2 = list(map(str, list1))
>>> print(list2)
['1', '2', '3', '4', '5', '6', '7', '8', '9']
>>>
```

【例 5-7】 写出下面语句完成的功能。

```
list(map(lambda x, y: (x + y, x - y), [1,3,5], [2,4,6]))
```

参考答案：将所给两个列表完成匿名函数表达式要求的对应元素相加及相减，组成新的元素，返回新的列表，最后执行的结果为：[(3，－1)，(7，－1)，(11，－1)]。

【例 5-8】 利用 map()函数将下面列表中的每一个元素加 2。

```
list1 = [1,2,3,4,5]
```

参考代码如下：

```
>>> list1 = [1,2,3,4,5]
>>> func  =  lambda x:x + 2
>>> list2  =  map(func, list1)
>>> print(list(list2))
[3, 4, 5, 6, 7]
>>>
```

2. reduce()函数

reduce()函数可以对可迭代对象的所有元素按函数执行累积操作，返回累积计算所得

到的唯一结果。该函数的格式如下：

```
reduce(func, iter, [initial_value])
```

其中，func 必须是一个接收两个元素并返回一个值的函数。reduce()函数接收迭代器返回的前两个元素 A 和 B 并计算 func(A,B)，然后请求第三个元素 C，计算 func(func(A,B),C)，如此继续直到遍历整个可迭代对象；如果提供了初值(initial value)，该初值会被用作计算的起始值，也就是先计算 func(initial_value,A)。

reduce()函数属于 functools 模块提供的高阶函数，在 Python 2 中为内置函数，在 Python 3 中需要导入 functools 模块，导入代码如下：

```
from functools import reduce
```

之后，就可以与 map()等函数一样使用了。

使用时需要注意，reduce()函数传入的 function 参数不能为 None，如果输入的可迭代对象不返回任何值，会抛出 TypeError 异常。

下面是使用 reduce()函数的几个示例程序：

```
>>> from functools import reduce                    # 导入 functools 模块
>>> def mul_2(x,y):
... return x * y
...
>>> product = reduce(mul_2, [1, 2, 3], 1)           # 实现累积乘法
>>> print(product)
6
>>> s1 = reduce((lambda x,y:x + y), [1,2,3,4])      # 实现累积加法
>>> print("s1 = ",s1)
s1 = 10
>>> s2 = reduce((lambda x,y:x + y), [1,2,3,4], 90)  # 有初始值
>>> print(s2)
100
>>> s3 = reduce((lambda x,y:x/y), [1,2,4,5])        # 实现累积除法
>>> print(s3)
0.025
>>>
```

【例 5-9】 利用 reduce()函数实现 10 的阶乘运算。

参考代码如下：

```
>>> from functools import reduce
>>> def multiply(x,y):
... return x * y
...
>>> print(reduce(multiply,range(1,11)))
3628800
>>>
```

3. filter()函数

filter()函数可以对指定序列执行过滤操作，过滤掉某些不需要的数据，保留有用的数据，格式如下：

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

其中，参数 function 可以是函数名或者 None，参数 iterable 为一可迭代数据。如果 function 是函数名，则将 iterable 数据里的每一个元素作为函数的参数进行计算，将返回 True 的数据筛选出来。

例如，在下面的例子中，filter()函数的 function 参数为 None，直接筛选值为 True 的数据。

```
>>> list(filter(None,[11,1,2,0,0,0,False,True]))          # 筛选函数为 None
[11, 1, 2, True]
>>>
```

在下面的例子中，filter()函数的 function 参数为匿名函数实现数据除以 2 并取余，相当于筛选序列中的奇数数据。

```
>>> list(filter(lambda x:x % 2, range(1,11)))          # 筛选函数为匿名函数
[1, 3, 5, 7, 9]
>>>
```

【例 5-10】 将下面给出的列表 list1 中的负数和 0 过滤掉，得到一个只有正整数的数组，并将得到的结果存入列表 list2。

```
list1 = [ - 1,0,1, - 2,3]
```

参考代码如下：

```
>>> list1 = [ - 1,0,1, - 2,3]
>>> list2 = list(filter(lambda x:x > 0, list1))
>>> list2
[1, 3]
>>>
```

5.7 常用模块和函数

Python 3.11 中内置了约 70 个内置函数，这些函数在 Python 官方文档中有详细的解释。这些函数涵盖了数学运算、类型转换、序列操作、变量操作、文件操作等多种类型，在前面的内容中有很多已经讲过，比如 print()、int()、type()、id()、max()等，本节补充讲解几个第三方模块及需要用到第三方模块的常用函数。

5.7.1 math 模块


视频讲解

math 模块是 Python 的标准模块，提供了三角函数、对数函数、幂函数、角度转换、双曲函数等常用的数学运算，提供了对 C 标准定义的数学函数的访问。除非另有明确说明，math 模块提供函数的所有返回值均为浮点数。

要使用 math 模块方法，首先要导入该模块，如下所示：

```
import math
```

导入后，即可以使用其中的数学函数相关的方法。math 模块中有常量和函数，常量主要包括两个，即 math.pi 和 math.e。

(1) 圆周率 π。π 表示圆周率，是一个常量，在 math 库中表示为 math.pi，取值 3.141592…，精确到可用精度。

(2) 自然常数 e。e 表示自然常数，也是一个常量，属于无理数，在数学中也是常用的一个常量数据，在 math 库中表示为 math.e，取值为 2.718281…，精确到可用精度。

π 和 e 的使用示例如下：

```
>>> import math
>>> PI = math.pi                    # 圆周率 π
>>> E = math.e                      # 自然数 e
>>> print("PI = ",PI)
PI = 3.141592653589793
>>> print("E ",E)
E 2.718281828459045
```

下面给出一些 math 模块提供的常用函数，如表 5-1 所示。

表 5-1 math 常用函数

函 数	说 明
math.exp(x)	返回 e 的 x 次方，其中 e = 2.718281…，是自然对数的基数
math.sqrt(x)	返回 x 的平方根
math.sin(x)	返回 x 弧度的正弦值
math.cos(x)	返回 x 弧度的余弦值
math.tan(x)	返回 x 弧度的正切值
math.degrees(x)	将角度 x 从弧度转换为度数
math.radians(x)	将角度 x 从度数转换为弧度
math.log(x[,base])	如果使用一个参数，返回 x 的自然对数(底为 e)。如果有两个参数，则返回给定 base 的对数 x，计算为 log(x)/log(base)
math.log2(x)	返回 x 以 2 为底的对数，通常比 log(x,2) 更准确
math.log10(x)	返回 x 以 10 为底的对数，通常比 log(x,10) 更准确

示例代码如下：

```
>>> import math
>>> PI = math.pi                    # 圆周率 π
>>> print(math.sin(PI/6))           # sin(π/6)
0.49999999999999994
>>> print(math.exp(2))              # e 的 2 次方
7.38905609893065
>>> print(math.sqrt(100))           # 100 的平方根
10.0
>>> print(math.log(100, 10))        # 100 以 10 为底的对数
2.0
>>> print(math.log2(256))           # 256 以 2 为底的对数
8.0
>>> print(math.log10(100))          # 100 以 10 为底的对数
2.0
>>>
```

从以上 math 模块函数运算结果来看，结果都是浮点数，且有些结果存在无限近似，比如 math.sin(PI/6)，即 sin(30)＝0.5，但是实际输出结果为 0.49999999999999994，是一个无限近似的结果。math 模块提供的数学函数很多，使用时可以查询 Python 官方文档。


视频讲解

5.7.2 随机数函数

程序中经常会用到随机数，Python 中的 random 模块可以用于生成随机数。借助于该模块，可以方便地生成随机整数和小数、从序列中随机抽取数据等。random 模块中的主要随机函数的用法如表 5-2 所示。

表 5-2 随机数函数功能

方　法	说　明
random.random()	用于生成一个 0～1 的随机浮点数，0≤n<1.0
random.uniform(a,b)	返回 a,b 之间的随机浮点数，范围是[a,b]还是[b,a]取决于四舍五入后的结果，a 不一定比 b 小
random.randint(a,b)	返回 a,b 之间的整数，范围为[a,b]。注意：传入参数必须是整数，a 一定要比 b 小
random.randrange([start], stop[,step])	返回给定区间内的随机整数，可以设置 step，默认值为 1
random.choice(sequence)	从序列类型变量中随机获取一个元素
random.shuffle(x[,random])	用于将列表中的元素打乱顺序
random.sample(sequence,k)	从指定序列中随机获取 k 个元素作为一个片段返回，不会修改原有序列

random 模块不是内置模块，使用时需要在程序开始前导入。下面是随机数生成的示例：

```
>>> import random
>>> print(random.random())                    # 生成 0～1 的随机数
```

```
0.603362655084515
>>> print(random.uniform(50,100))              # 生成 50～100 的随机浮点数
75.33137414537545
>>> random: 85.75230473615628
>>> print(random.randint(12,20))               # 生成 12～20 之间的整数
19
>>> p = ["Python","is", "simple"]              # 将 p 打乱顺序
>>> random.shuffle(p)
>>> print(p)
['Python', 'simple', 'is']
>>> list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]     #从 list 中随机抽取五个元素
>>> print(random.sample(list, 5) )
[6, 5, 2, 4, 8]
>>>
```

5.7.3 time 模块


视频讲解

Python 中常见的处理时间与日期相关的模块主要有 time 模块、datetime 模块和 calendar 模块。time 模块是处理时间的模块，如获取时间戳、格式化日期等；datetime 模块是 date 和 time 的结合体，可以处理日期和时间相关的操作；calendar 模块是日历相关的模块，主要用于处理年历/月历等日历相关的操作。

time 模块、datetime 模块和 calendar 模块都不是 Python 的内置模块，都需要先导入再使用，如下所示：

```
>>> import time
>>> import datetime
>>> import calendar
>>>
```

这三个模块都内置了许多属性与方法，在使用时可以借助于 dir()或者 help()函数对模块所带的属性与方法进行查询，比如可以利用 dir()函数列出 time 模块的方法或属性，如下所示：

```
>>> import time
>>> dir(time)
['_STRUCT_TM_ITEMS', '__doc__', '__loader__', '__name__', '__package__', '__spec__', 'altzone',
'asctime', 'ctime', 'daylight', 'get_clock_info', 'gmtime', 'localtime', 'mktime', 'monotonic',
'monotonic_ns', 'perf_counter', 'perf_counter_ns', 'process_time', 'process_time_ns', 'sleep',
'strftime', 'strptime', 'struct_time', 'thread_time', 'thread_time_ns', 'time', 'time_ns',
'timezone', 'tzname']
>>>
```

在时间模块中用得比较多的是时间元组(struct_time)，该元组共有九个元素，含义如表 5-3 所示。

表 5-3 struct_time 对象的元素

索 引	属 性	值	功 能
0	tm_year	四位数，如 2022	年份
1	tm_mon	range [1,12]	月份
2	tm_mday	range [1,31]	月份中的日期
3	tm_hour	range [0,23]	小时
4	tm_min	range [0,59]	分钟
5	tm_sec	range [0,61]	秒
6	tm_wday	range [0,6]，周一为 0	一周中的第几天
7	tm_yday	range [1,366]	一年中的第几天
8	tm_isdst	0,1 或 −1	是否夏令时，1 为是 0 为否
N/A	tm_zone	时区名称的缩写	
N/A	tm_gmtoff	以 s 为单位，UTC(世界标准时间)以东偏离	

time.localtime([secs])可以返回当前时间元组 struct_time。time.asctime([t])函数接收时间元组并返回一个可读的时间形式的 24 个字符的字符串，包含年、月、日、星期、时、分、秒。如下所示：

```
#Example5.28 时间模块
import time
print(time.localtime())                    # 本地时区时间
print(time.asctime())                      # 可读形式时间
```

执行结果如下：

```
time.struct_time(tm_year = 2022, tm_mon = 3, tm_mday = 27, tm_hour = 10, tm_min = 40, tm_sec =
2, tm_wday = 6, tm_yday = 86, tm_isdst = 0)
Sun Mar 27 10:40:02 2022
```

time.strftime(format,[t])可以将 struct_time 对象转换为 format 格式化的可读形式输出，如果参数 t 未给出，则取当前时间。比如将本地当前时间转换为“年、月、日、时、分、秒”的形式输出，代码如下所示：

```
#Example5.29 本地时间格式化输出
import time
t = time.localtime()
print(time.strftime("%Y-%m-%d %H:%M:%S"))
```

执行结果如下：

```
2022-03-27 10:58:39
```

time.sleep(secs)是线程相关的时间函数，可以推迟线程的运行时间，通过参数 sec 指

定秒数，表示线程暂停执行的时间。在程序中可以用作延时程序，比如实现 5s 的延时，代码如下：

```
#Example5.30 延时
import time
t1 = time.localtime()
print(time.strftime("开始时间:"" %H:%M:%S"))
time.sleep(5)                    # 延时 5s
print(time.strftime("结束时间:"" %H:%M:%S"))
```

执行结果如下：

```
开始时间:11:06:02
结束时间:11:06:07
```

【例 5-11】 设计一个倒计数程序，实现倒数 10 个数，每秒倒数一个，数到 1 时显示“开始!”

设计分析：本例中需要一个倒数序列，可以用循环生成，而利用序列的逆序功能将更加简便。另外每次输出的文本都输出在相同的位置，需要控制输出之后不换行，而且每次要回到行首，这就需要用到“\r”回车符。利用回车符控制输出回到行首，time.sleep()函数实现 1 秒延时，本程序就可以方便地实现了。示例代码如下：

```
#Example5.31 倒计数程序设计
import time
for i in range(10,0,-1):                    # 序列为[10、9、…、2、1]
    print('\r',i, end = '')                 # \r 控制回到行首，但不换行
    time.sleep(1)                           # 延时 1s
print("\n 开始")
```

执行结果如下：

```
1
开始
```

【例 5-12】 设计一个进度条，前面显示变化的数字，从 0 一直变化到 100%，随着安装进度的进行后面显示-，每变化 5%增加一个-，所有进度完成后在进度条的末尾显示“OK!”。可以用时间模拟安装进程，每 0.1s 完成 1%进度条，示例如下：

```
50% ----------
100% ----------OK!
```

参考代码如下：

```
#Example5.32 进度条设计
import time
nums_sign = 20                  # -的数量
```

```
count = 5                    # 计数
for i in range(101):
    disp_string = "\r" + str(i) + "% " + "-" * int(i/5)
    print(disp_string,end = "")
    time.sleep(0.1)
    count += 1
print("OK!")
```

需要注意的是,尽管所有平台都可以使用 time 相关模块,但由于模块中定义的大多数函数的实现都需要调用其所在平台的 C 语言库的同名函数,因此这些函数的语义可能会有所变化,有些函数可能无法使用或执行方式存在差异,需要在使用时查阅对应平台的相关文档;另外 Idle 平台对于回车符"\r"的支持不理想,需要借助 PyCharm、Anaconda 等平台。


视频讲解

5.7.4 main()函数

利用 Python 开发的代码文件可以单独执行,也可以作为模块在其他文件中调用执行。在 Python 中,引入了一个变量__name__,当前文件被调用时,__name__的值为文件名,而当编写的 py 文件直接运行时,__name__的值为“__main__”。也就是说,当该 python 脚本被作为模块(module)导入(import)时,其中的 main()函数将不会被执行。

在编写可能会被其他程序调用的 Python 程序时,利用该特性可以编写只有本程序独立执行时的程序代码,用于某些功能测试或验证。示例代码如下:

```
#Example5.33 main()函数使用
print('Hello World!')
print('__name__变量的值为:', __name__)
def main():
    print('本程序单独运行')
if __name__ == '__main__':          #判断是否单独执行
    main()
```

程序运行结果为:

```
Hello World!
__name__变量的值为: __main__
本程序单独运行
```

本程序作为 module 被其他程序 import 执行时,__name__的值为文件名,main()函数中的语句将不再执行,也就是说“本程序单独运行”这条信息将不会显示。

设计实践


视频讲解

1. 四则运算

编写一个 Python 程序,生成“加减乘除”四则运算的练习,并能判断结果是否正确。程

序可以选择进行哪种运算，根据输入的数据判断运算结果是否正确，最后给出正确性统计。

2. 图案绘制

在 python 中，边循环边计算的机制称为生成器，当用到数据的时候再生成，这样可以节约空间，提高效率。试编写程序，利用生成器得到如图 5-2 所示的图案，其中 n 作为输入参数，代表图案的行数。

```
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
[1, 6, 15, 20, 15, 6, 1]
[1, 7, 21, 35, 35, 21, 7, 1]
[1, 8, 28, 56, 70, 56, 28, 8, 1]
[1, 9, 36, 84, 126, 126, 84, 36, 9, 1]
```

图 5-2 n=10 时的图案


视频讲解

3. 因数分解

在数学中，因数分解又称素因数分解，是指把一个正整数写成几个约数的乘积。例如，对于 45 这个数，可以将其分解成 3×3×5。根据算术基本定理，这样的分解结果应该是独一无二的。


视频讲解

4. 杨辉三角

杨辉三角是二项式系数在三角形中的一种几何排列，最早由中国南宋数学家杨辉 1261 年提出，其数据如图 5-3 所示。杨辉三角第 n 行的数字有 n 项，每行数字左右对称，由 1 开始逐渐变大，每个数等于它上方两数之和。试用递归函数方法打印出 n 行的杨辉三角图形。


视频讲解

```
n=1                    1
n=2                  1   1
n=3                1   2   1
n=4              1   3   3   1
n=5            1   4   6   4   1
n=6          1   5   10  10  5   1
n=7        1   6   15  20  15  6   1
```

图 5-3 杨辉三角

本章小结

本章主要介绍了函数的相关知识，包括函数的概念、函数的参数、函数的返回值、变量的作用域、递归函数、函数式编程及常用模块和函数。

函数的概念部分介绍了如何定义函数和调用函数；函数的参数部分主要介绍参数的传递方式，包括默认参数传递、位置参数与关键字参数传递及可变参数传递；函数的返回值部分介绍了如何将函数结果传递给调用者；变量的作用域部分介绍了访问变量要遵循的 LEGB 原则并且不同变量的作用域也不同；递归函数部分介绍了如何用函数去实现递归算法；函数式编程介绍了匿名函数和高阶函数（包括 map()、reduce()、filter()函数）；常用模块和函数部分介绍了 math 模块、随机数函数、time 模块、main()模块，其中时间相关模块的内容比较多，限于篇幅，本节只讲了入门内容，具体可以查阅 Python 开发文档或 Python 帮助。

本章内容可帮助读者充分利用这些函数，在实际应用中更好地提高应用的模块化，提升工作效率。

本章习题

一、填空题

1. 函数体的第一行语句可以使用_____，用于函数说明，利用文档字符串可以自动生成在线文档或打印版文档。

2. 函数定义之后，Python 解释器把函数名与_____关联在一起，把函数名指向的对象作为用户自定义函数。

3. 函数的_____是函数内部与外部交流的纽带，起传递数据的作用。

4. “* name”形式的可变参数会接收所有未命名的变量参数，并放入_____对象中，该元组可以包含形参列表之外的所有位置参数。

5. 函数中 return 用于返回数据给调用者，如果返回值是多个，则返回结果的数据类型为_____。

6. Python 中的作用域分四种情况，简称 LEGB 规则，其中的 L 含义是_____。

7. 如果想在函数中修改全局变量，需要在变量的前面加上_____关键字。

8. 下述程序执行后，执行的结果为_____：_____。

```
def test(a,b,c,d,e):
    print('number:',a,b,c,d,e)
num = {'c':3,'d':4,'e':5}
test(1,2, ** num)
```

9. 用递归函数进行功能实现时，需要找到递归关系和递归出口，递归关系需要建立起 n 与_____步骤的数学关系式。

10. Python 支持函数式编程，允许把函数本身作为_____传入另一个函数，还允许返回一个函数。

11. _____就是没有名称的函数，不再使用 def 语句定义的函数。

12. 一个函数可以接收另一个函数作为参数，这种函数就称为_____ 。

13. _____函数可以对可迭代对象的所有元素按函数执行一个累积操作，返回累积计算所得到的唯一结果。

14. 生成 1～10 的随机整数，使用的语句为_____。

15. 时间模块中用得比较多的是_____，该元组共有九个元素，包含年、月、日、时、分、秒等信息。

二、选择题

1. Python 中一般采用(　　)关键字定义函数，后跟函数名与括号内的参数列表。

A. func　　B. def　　C. proc　　D. define

2. 定义函数时函数名后面的一对小括号中给出的参数称为(　　)。

A. 实参　　B. 形参　　C. 类型参数　　D. 名字参数

3. 可变参数传入函数后，会被看作是(　　)类型的变量。

A. 字典　　B. 元组　　C. 集合　　D. 列表

4. 将一个函数的运算结果返回函数调用方，应使用(　　)语句。

A. break　　B. return　　C. print　　D. continue

5. 执行表达式 len(range(1,10))，执行后的结果为(　　)。

A. 9　　B. 10　　C. 45　　D. 55

6. 下列高阶函数中，(　　)函数可以接收两个参数，一个是函数，一个是 Iterable，它将传入的函数依次作用到序列的每个元素，并把结果作为新的 Iterator 返回。

A. map()　　B. reduce()　　C. filter()　　D. sorted()

7. 下列高阶函数中，(　　)函数可以用于过滤序列，过滤掉不符合条件的元素，返回符合条件的元素组成新列表。

A. map()　　B. reduce()　　C. filter()　　D. sorted()

8. 使用(　　)关键字声明匿名函数。

A. def　　B. function　　C. lambda　　D. procedure

三、判断题

1. 函数中使用 return 语句返回结果数据给调用者，不论返回值有多少个，都返回一个元组数据。(　　)

2. 函数中带有默认值的参数位置没有要求，主要参数的书写就可以。(　　)

3. 函数的返回值只能有 1 个。(　　)

4. global 和 nonlocal 关键字含义类似，在使用时可以互换，不会影响执行的功能。(　　)

5. 匿名函数不需要 return 语句，可以包含多个表达式，表达式的值就是返回值。(　　)

四、简答题

1. 什么是函数？函数定义与使用时需要注意的问题有哪些？

2. 什么是默认参数？简述默认参数使用时的注意事项。

3. 什么是位置参数传递和关键字参数传递？二者有何区别？

4. 什么是可变参数？可变参数前面加 * 和加 ** 有何区别？

5. 什么是变量的作用域？简述 LEGB 的含义。

6. 简述什么是函数式编程，有何作用。

五、编程题

1. 请编写递归编写函数 fac(n)，实现整数 n 的阶乘计算：n! = 1 * 2 * 3 *… * n，并调用该函数，计算 10～20 的阶乘。

2. 编写程序，利用 map()函数，将列表元素中的英文字符都转换成小写。例如：输入['Qingdao','DALIAN','xian']；输出'qingdao','dalian','xian'。

3. 回文数是一个左右对称的整数，比如 131，1357531。请编写函数，判断用户输入的整数是否为回文数。

4. 定义一个可实现四则运算的函数 cal()，要求如下：

（1）函数共包含三个参数：num1、num2 和 operator，其中参数 num1 和 num2 接收数据，而 operator 的默认值为＋，且只接收＋、－、＊、/中的任一运算符；

（2）函数中根据相应的运算符执行相应的运算，并返回计算后的结果；

（3）执行除法运算时，num2 的值不能为 0。

5. 汉诺塔问题是经典的心理学实验研究问题之一。相传在古印度圣庙中，有一种被称为汉诺塔（Hanoi）的游戏。该游戏是在一块铜板装置上，有三根杆（编号为 A、B、C），在 A 杆自下而上、由大到小按顺序放置 64 个金盘，如图 5-4 所示。游戏的目标：把 A 杆上的金盘全部移到 C 杆上，并仍保持原有顺序叠好。操作规则：每次只能移动一个盘子，且在移动过程中三根杆上都始终保持大盘在下、小盘在上，操作过程中盘子可以置于 A、B、C 任一杆上。试利用函数编写程序解决该问题。

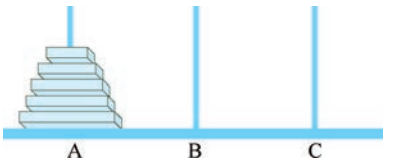


图 5-4　汉诺塔