第5章 函数

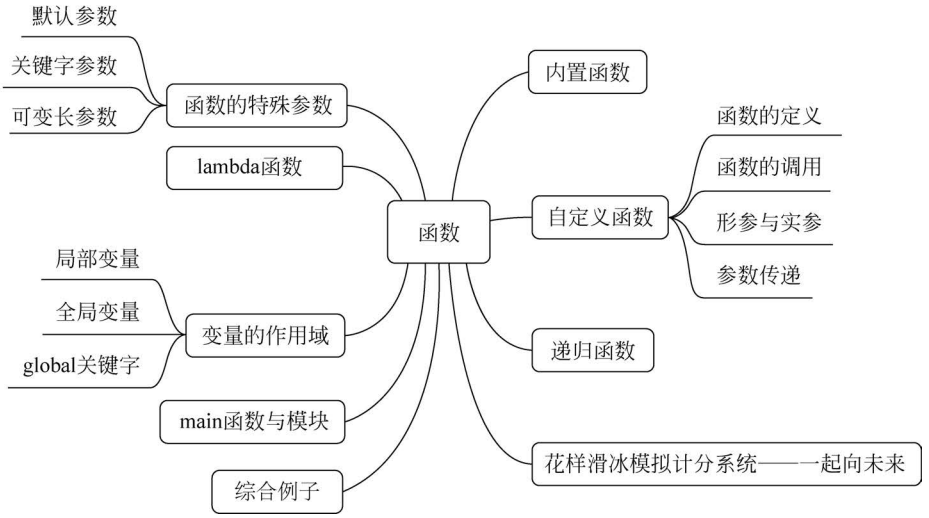
能力目标

【应知】 理解通过函数实现模块化编程思想。

【应会】 掌握函数定义和调用的方法，掌握函数参数传递机制，掌握默认参数、可变长参数和关键字参数，掌握局部变量和全局变量，掌握 lambda 函数和递归函数。

【难点】 函数的参数、lambda 函数和递归函数。

知识导图



在设计较复杂的程序时，一般采用自顶向下的方法，先将复杂问题划分为几个部分，再对各个部分进行细化，直到分解为能解决的子问题，每个子问题就变为独立的程序模块。每个模块是整个系统的一部分，完成一个单独的功能，这就是模块化编程。模块化编程的主要思想是将程序分解为逻辑上相对独立的模块，每个模块实现特定的功能，相互之间通过预先定义的接口进行交互。

为实现模块化，可使用函数封装模块的代码。函数可以将相关代码块组织在一起，通过定义函数接口与其他部分交互。函数有助于将大程序拆分为逻辑独立的模块，通过封装提高代码的复用性和可维护性，并提高程序的可读性。总之，使用函数可使代码更清晰、灵活、可管理，是模块化编程的基础。

本章主要内容包括 Python 内置函数和自定义函数，如何定义函数，如何调用函数，函

数的实参、形参，参数之间是如何传递的，函数的默认参数、可变长参数、关键字参数、局部变量和全局变量、lambda 函数、filter 函数、map 函数、递归函数、main 函数及模块。

5.1 内置函数

为方便用户使用，Python 提供了许多内置函数，如 print、abs、len、int、max 等。表 5.1 列出了常用的内置函数，内置函数是可以直接使用的函数。对于内置函数，需要掌握函数名、函数功能、函数参数和返回值。

表 5.1 常用的内置函数

函数类别	函数名	说　明	示　例	运行结果
输入输出	print(s)	输出字符串 s	>>> print("hello, Python")	hello,Python
	input()	获取用户输入内容	>>> name＝input(“enter your name：”)	
数值运算	abs(x)	返回参数 x 的绝对值	>>> abs(－3.4)	3.4
	divmod(a，b)	一个包含商和余数的元组：(a // b，a % b)	x,y＝divmoid(10,3)	(3,1)
	round(x[，n])	返回参数 x 的四舍五入值，参数 n 可选，表示保留 n 位小数，默认表示不保留小数位	>>> round(3.1415926) >>> round(3.1415926,2)	3 3.14
	pow(a,b)	返回参数 a 的 b 次幂	>>> power(3,2)	9
	sum()	对可迭代对象参数求和	>>> sum(1,2,3,4) >>> sum([1,2,3,4])	10 10
	min()	求最小值	>>> min(1,2,3,4) >>> min([1,2,3,4])	1 1
	max()	求最大值	>>> max(1,2,3,4) >>> max([1,2,3,4])	4 4
类型转换	int(a)	将参数 a 转换为整数并返回	>>> int(3.5)	3
	float(a)	根据参数 a 返回其对应的浮点数	>>> float(3)	3.0
	str(a)	返回参数 a 的字符串形式	>> str(35)	‘35’
数据结构	list(iterable)	接受一个可迭代的对象作为输入参数，将该对象中的元素转换为列表，再返回这个列表	>>> list((1,2))	[1,2]
	tuple(iterable)	接受一个可迭代的对象作为输入参数，将该对象中的元素转换为元组，再返回该元组	>>> tuple([1,2])	(1,2)
	dict(iterable)	通过迭代 iterable 对象，将键值对重新组装为字典	>>> dict([('a'，1)，('b'，2)]) >> dict(a＝1，b＝2)	{'a'：1，'b'：2} {'a'：1，'b'：2}
	set(iterable)	将可迭代对象转换为集合，元素不重复	>>> set([1,2,2,3])	{1,2,3}

续表

函数类别	函数名	说　明	示　例	运行结果

序列运算	range(start, stop[, step])	返回一个[start, stop)步长为step的列表,start和step的默认值分别为0和1	>>> range(5)
			>>> range(1,5)
			>>> range(1,5,2)
				[0,1,2,3,4]
				[1,2,3,4]
				[1,3]

	len(s)	返回对象s中元素的个数	>>> len([1,2,3,"h"])	4
	sorted(iterable, key = None, reverse=False)	对可迭代序列进行排序,返回排序后的序列	>>> sorted([5,2,3]) >>> sorted([5,2,3], reverse=True)	[2,3,5] [5,3,2]
	filter(function, iterable)	对iterable对象中的每个元素调用function函数进行判断,将函数返回True的元素过滤出来,生成一个新的迭代器返回	>>> list(filter(lambda x: x%2 == 0, [1,2,3,4]))	[2,4]

	zip(*iterables)	将可迭代对象中对应的元素打包为一个个元组,再返回由这些元组组成的zip对象	>>> n=[1,2]
			>>> s=["a","b"]
			>>> list(zip(n,s))
				[(1, 'a'), (2, 'b')]

	map(function, iterable, ...)	将function函数依次作用到iterable可迭代对象中的每个元素,并将结果作为新的map对象返回	>>> n=[1,2] >>> list(map(lambda x: x*3,n))	[3,6]
其他	id(object)	返回参数object的内存地址	y=id(6)	
	type(object)	返回参数object的数据类型	type(4)	<class 'int'>

5.2 自定义函数

5.2.1 函数定义

当内置函数无法满足需求时,需要自己创建函数,称为自定义函数,其语法格式为:

```
def func_name([arguments]):                # 函数定义头部,指定函数名和参数
    statement(s)                           # 函数体语句,实现功能
    [return expression]                    # 返回值表达式
```

函数由两部分组成:函数头和函数体。

函数头:包括def关键词、函数名称、参数列表。函数头以def开头,后跟函数名称和圆括号()。圆括号中是参数列表,多个参数之间用英文逗号隔开,可以没有参数。圆括号后面是一个冒号。

函数体:函数体包含函数执行的语句,使用缩进表示函数体代码块。函数体内部可使用return语句返回值。如果没有return语句,函数将不返回值。一个函数可有多个返回值。

函数定义涉及函数的名称、参数、返回值及函数体代码,是创建一个函数的必要元素。通过函数定义,构造一个可被重复调用的代码块,用于实现特定的功能。

【实例5.1】 函数定义示例。

```
def greet():          #定义一个输出欢迎信息的函数.函数名为 greet,没有参数
    print("Welcome to Python world!")
def c_f(c):           #定义一个将摄氏度向华氏度转换的函数,函数名为 c_f,有一个参数 c
    f = 32 + c * 1.8
    return f
```

该程序定义了两个函数。

greet 函数：无参数，打印一条问候语，没有返回值。

c_f 函数：有 1 个参数 c，实现摄氏度到华氏度的转换，通过 return 返回计算结果。

运行上述程序，发现该程序没有任何输出，因为程序中只进行了函数定义，未调用函数。定义仅表示创建了函数，但未实际执行。要实现函数功能，必须在其他地方调用预先定义的函数。

5.2.2 函数调用

函数调用是使用预先定义的函数完成某项任务的过程，函数调用会引发程序控制流从主程序转移到被调用函数的函数体，执行其中的语句，再返回主程序调用点并带回返回值。

函数调用的格式如下：

```
func_name(par1,par2,…)
```

其中：

func_name 为函数名，调用的函数名必须与定义的函数名完全一致。

函数调用中函数名后圆括号内的 par1,par2,…为函数实参，即从主程序向该函数传递的参数值。需要注意的是，函数调用时的参数个数和参数顺序必须与函数定义时一致。另外，即使该函数没有参数，调用时也要书写一对空的括号。

【实例 5.2】 调用【实例 5.1】中定义的函数示例。

```
greet()                       #调用函数 greet,该函数没有返回值
f_temperature = c_f(30)       #调用函数 c_f,将 30 的值传递给参数 c
print("30 摄氏度对应的华氏度为:",f_temperature)
print("20 摄氏度对应的华氏度为:",c_f(20))   #调用函数 c_f,将 20 的值传递给参数 c
```

运行结果如下：

```
Welcome to Python world!
30 摄氏度对应的华氏度为: 86.0
20 摄氏度对应的华氏度为: 68.0
```

实例 5.2 调用了实例 5.1 中定义的两个函数。

(1) 调用 greet 函数，该函数无返回值，直接调用即可执行其内部语句。

(2) 调用 c_f 函数，该函数有返回值。可将返回值赋给变量，如 f_temperature=c_f(30)，也可直接将函数调用作为表达式的一部分，如 print(c_f(20))。

所以对于无返回值的函数，直接调用即可。对于有返回值的函数，可通过赋值运算获得返回值，也可直接使用函数调用结果。

通过调用可重复使用函数内部的代码逻辑。正确调用函数是应用函数的关键。

【实例 5.3】　有多个返回值的函数示例。

```
def ave_names(dic1):
    scores = [values for values in dic1.values()]
    sum = 0
    for score in scores:
        sum = sum + score
    ave = sum/len(scores)
    names = []
    for name in dic1.keys():
        if dic1[name]< ave:
            names.append(name)
    return ave,names
scores = {"zhangsan":90,"lisi":79,"wangwu":56,"zhangliu":72}
average,names = ave_names(scores)
print("平均分:",average)
print("低于平均分的有:",names)
```

运行结果如下：

```
平均分: 74.25
低于平均分的有: ['wangwu', 'zhangliu']
```

该示例定义了 ave_names 函数，用于计算学生成绩字典的平均分和低于平均分同学的名字。函数参数是一个字典，该字典是以“姓名：分数”格式存储的成绩单。该函数首先计算所有分数的平均值 ave，再遍历所有名字，如果分数低于 ave，则将名字添加到 names 列表，通过 return 语句返回 ave 和 names 两个值。函数调用时，使用两个变量接收返回值：average，names＝ave_names(scores)，变量 average 接收返回的 ave，names 接收返回的 names。

该示例展示了函数可通过一个 return 语句返回多个值。调用时需用多个变量对应接收函数的多个返回值。

5.2.3　形式参数和实际参数

函数的参数有形式参数(简称形参)和实际参数(简称实参)之分，其中形参是指函数定义时函数名后括号中的参数，用于接收函数调用时传入的具体数据，其作用域为该函数局部。实参是函数调用时函数名后括号中的参数，用于向形参传递具体的值。在进行函数调用时参数传递的过程如图 5.1 所示。

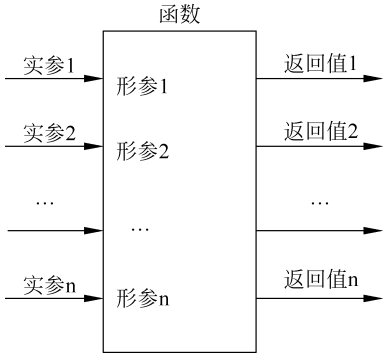


图 5.1　函数调用示意图

【实例 5.4】 形参和实参示例。

```
1    def add(a,b):#这里的 a 和 b 就是形参
2        return a+b
3    result=add(1,2)#这里的 1 和 2 是实参
4    print("1 + 2 =",result)
5    x=2
6    y=3
7    print(x,"+",y,"=",add(x,y))#这里的 x 和 y 是实参
```

运行结果如下：

```
1 + 2 = 3
2 + 3 = 5
```

实例 5.4 中定义了一个函数 add，有两个形参 a 和 b，一个返回值。第 3 行 result=add(1,2)是函数调用，这里的 1 和 2 是实参，1 的值传递给形参 a，2 的值传递给形参 b，add(1,2)得到的值赋给 result。

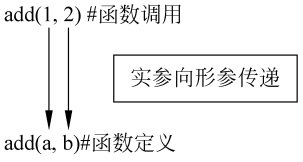


图 5.2 参数传递示意图

在函数调用时，实参列表按照形参列表的顺序依次向形参传递，如图 5.2 所示。

第 7 行也是函数调用，首先将实参 x 的值传递给形参 a，实参 y 的值传递给形参 b，计算得到 a、b 之和，再返回两者之和，通过 print 语句输出结果。

在函数调用时，实参和形参要一一对应，包括参数个数的对应、参数类型的对应，否则将导致错误。

假设仍然使用上面的 add 函数，如果进行如下函数调用：

```
>>> print(add(5))
```

则提示如下错误：

```
TypeError: add() missing 1 required positional argument: 'b'
```

如果进行如下函数调用：

```
>>> print(add(2,'hello'))
```

则提示如下错误：

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

5.2.4 参数传递

在 Python 中，函数调用的参数传递有两种方式：值传递和引用传递。

值传递：

- 形参作为函数的局部变量，与实参的值完全独立。
- 形参相当于实参的一个副本，函数内部对其操作不会影响实参。
- 适用于不可变类型，如数字、字符串等。

引用传递：

- 传入的参数是可变对象，如列表、字典等。

- 形参和实参引用同一对象，对形参的操作会改变实参。
- 实参传入的对象在函数中被修改，源对象也会改变。

区分传递方式，有助于理解函数对参数的影响范围。值传递保护实参安全，引用传递方便修改可变对象。

【实例 5.5】 参数传递示例。

```
def plus_one(x):                    #将数值形参 x 的值增加 1
    x = x + 1
def plus_two(lst):                  #将列表形参的每个元素值增加 2
    for i in range(len(lst)):
        lst[i] = lst[i] + 2
m = 5
list1 = [1,2,3]
print("执行增加函数之前:")
print("m:",m)
print("list1:",list1)
plus_one(m)
plus_two(list1)
print("执行增加函数之后:")
print("m:",m)
print("list1:",list1)
```

运行结果如下：

```
执行增加函数之前:
m: 5
list1: [1, 2, 3]
执行增加函数之后:
m: 5
list1: [3, 4, 5]
```

实例 5.5 中定义了两个函数，其中 plus_one 函数的参数是数值类型变量 x，其功能是将 x 的值增加 1。plus_two 函数的参数是一个列表型变量 lst，其功能是将 lst 中每个元素的值增加 2。

主程序中，先定义一个数值类型变量 m，值为 5；再定义一个列表类型变量 list1，值为[1,2,3]。在进行函数调用 plus_one(m)时，将 m 作为实参传递给形参 x，但在 plus_one 内部对 x 的改变不会影响外部变量 m。在进行函数调用 plus_two(list1)时，将 list1 作为实参传递给形参 lst，在 plus_two 内部对 lst 的改变会影响实参 list1。

因为数值类型属于不可变数据类型，采用值传递；而列表类型属于可变数据类型，采用引用传递。值传递时，形参不会影响实参；引用传递时，形参和实参引用同一对象，对形参的修改反映到实参上。

5.3 函数的特殊参数

5.3.1 默认参数

Python 中的函数支持设置参数的默认值，可使用赋值运算符“=”实现。定义函数时可

为参数设置默认值，如实例 5.6 所示。

```
1    def power(x, n=2):                  #默认参数 n 的值为 2
2        s = 1
3        while(n>0):
4            s = s * x
5            n = n-1
6        return s
7    y = power(5)                        #函数调用时，第二个参数 n 没有指定值，采用默认值 2
8    print(5,"**",2,"=",y)
9    print("4**3=",power(4,3))           #函数调用时，指定了第二个参数 n 的值，就采用指定的值
```

定义 power 函数时，指定了第二个参数 n 的值为 2。在进行函数调用时：

- 如果没有传递 n 的参数，该参数将取默认值 2，如第 7 行 power(5)所示。

- 如果调用时传递了 n 的参数，如第 9 行 power(4, 3)，则该参数取传入的实参值 3，而不是默认的 2。

参数默认值可简化函数的调用。调用者可仅传递变化的参数，不需要传递所有参数。默认参数只能出现在参数列表的最后，其后不能出现非默认参数，如实例 5.7 所示。

【实例 5.7】 默认参数的错误示例。

```
1    def power(x=2, n):
2        s = 1
3        while(n>0):
4            s = s * x
5            n = n-1
6        return s
7    y = power(5)
8    print(y)
```

运行结果如下：

```
def power(x=2, n):
                ^
SyntaxError: non-default argument follows default argument
```

5.3.2 关键字参数

在 Python 中，函数参数的传递默认按位置匹配，即调用函数时按照函数定义的参数顺序依次用位置对应的参数传递实际的值。例如，函数定义：def func_1(a, b, c, d)，函数调用：func_1(x, y, z, w)，则形成如图 5.3 所示的参数传递效果。

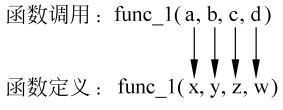


图 5.3 参数传递

可以看出，函数调用时实参按照形参的位置从左到右依次传递，实参的顺序必须与形参完全一致。一旦两者不一致，会产生异常运行结果。例如，求 2^3：

```
>>> print(pow(2,3))          #输出结果为 8，正确
>>> print(pow(3,2))          #输出结果为 9，错误
```

可以看出，按位置传参数易出现参数顺序不匹配问题。为解决此问题，Python 支持关

键字参数传递，其基本形式为“参数名 = 值”，从而明确指定参数对应的值，不再依赖参数的位置，如实例 5.8 所示。

【实例 5.8】 关键字参数示例。

```
1    def student_information(name,age,college):
2        print("name:",name)
3        print("age: ",age)
4        print("college:",college)
5    student_information("zhangsan",18,"computer science")          #按照位置传递参数
6    #第一个参数按照位置传递,后面两个按照参数名称传递
7    student_information("wangwu",college = "computer science",age = 18)
```

实例中定义了一个函数 student_information，含 3 个参数：name、age 和 college。

第 5 行代码 student_information("zhangsan", 18, "computer science")使用位置参数，将 3 个实参的值依次传递给 name、age 和 college。

第 7 行代码 student_information("wangwu", college="computer science", age=18)既使用了位置参数"wangwu"，也使用了关键字参数 college 和 age。位置参数满足从左到右依次传递的要求，关键字参数则通过“参数名=值”明确指派给对应的形参。所以“wangwu”传递给 name，“computer science”传递给 college，18 传递给 age。

注意，位置实参和关键字实参都是针对实参的，函数定义的参数顺序和名称不受影响。在函数定义中，形参还是与原来一样没有区别。只不过在函数调用时，如果是位置实参，则按照形参的顺序依次传递；如果是关键字参数，则根据关键字名字传递参数；若既有位置参数，又有关键字参数，则进行函数调用时，按照下列格式进行参数传递：

```
funcname([位置实参], [关键字实参])
```

注意应严格遵照此顺序，即位置参数在前，关键字实参在后。

所以 student_information(college="computer science",age=18,"wangwu")会出现下列错误提示：

```
student_information(college = "computer science",age = 18,"wangwu")
                                                                  ^
SyntaxError: positional argument follows keyword argument.
```

5.3.3 可变长参数

在定义函数时，通常预先定义函数需要的形参个数。但在某些情况下，无法确定函数调用时传入的参数个数，这时可使用可变长参数接收不定个数的参数。可变长参数通常以“*参数名”的形式定义，这类参数可接收调用时传入的不定长度的位置参数，并组织为一个元组。定义可变长参数的语法如下：

```
def func_name(formal_args, *args):
  pass
```

当定义包含可变长元组参数的函数时，普通形参 formal_args 在前，可变长元组参数 args 在后。普通形参可有多个，而可变长元组参数只能有一个。

在调用含可变长参数的函数时，要先匹配固定参数，剩余的位置参数会被 args 接收并

组成一个元组。

【实例 5.9】 可变长元组参数示例一。

```
1    def func(a, b, *c):
2      print("a:",a)
3      print("b:",b)
4      print("c:",c)
5    func(1,2,3,4,5)
6    func(1,2)
```

函数定义了 3 个参数 a、b、c，第 5 行函数调用时形参 a 接收了第一个实参的值 1，形参 b 接收了第二个实参的值 2，剩下所有实参的值 3、4、5 组成一个元组，全部传递给了 c；第 6 行函数调用时，形参 a 和 b 接收传递过来的 1 和 2，没有其他实参，那么形参 c 就是一个空元组。其参数传递如图 5.4 所示，运行结果如下：

图 5.4 可变长元组参数传递示意图

```
a: 1
b: 2
c: (3, 4, 5)
a: 1
b: 2
c: ()
```

【实例 5.10】 可变长元组参数示例二。

```
1    def lessThan(cutoffVal, *vals) :
2        #该函数的功能是将小于参数 cutoffVal 的所有数字返回
3        arr = []
4        for val in vals :
5            if val < cutoffVal:
6                arr.append(val)
7        return arr
8    average = 75
9    ar = lessThan(average,89,34,78,65,52)
10   print("小于",average,"的数字有:",ar)
11   average = 85
12   ar = lessThan(average,90,73,89,76,34,78,88,52)
13   print("小于",average,"的数字有:",ar)
```

运行结果如下：

```
小于 75 的数字有: [34, 65, 52]
小于 85 的数字有: [73, 76, 34, 78, 52]
```

lessThan 函数定义了两个参数 cutoffVal 和 * vals，cutoffVal 为普通参数，vals 为可变长元组参数，因为其前有 *，可以接收匹配 cutoffVal 之后的所有位置参数。

第 9 行函数调用 ar＝lessThan(average,89,34,78,65,52)时，首先将 average 传递给 cutoffVal，再将剩余的参数 89、34、78、65、52 作为一个元组全部传递给 vals。

Python 还提供了另一种可变长参数——可变长字典参数。可变长字典参数用在参数

名称前面加两个星号“**”表示，这种参数可接收函数调用时额外的关键字参数。

可变长字典参数的语法格式如下：

```
def func_name(formal_args, ** kwargs):
  pass
```

定义含可变长字典参数的函数时，普通形参 formal_args 在前，可变长字典参数 kwargs 在后，可变长字典形参只能有一个。

在函数调用时，会用实参依次匹配前面的普通参数 formal_args，之后剩下的所有格式如“关键字＝值”的实参将构成一个字典，传递给可变长字典参数 kwargs。

【实例 5.11】 可变长字典参数示例。学校新生开学注册时，其姓名和性别是必需信息，其他年龄、省份等信息是可选的，此时可使用可变长字典参数，即定义函数时，在参数名称前加 ** 。

```
1    #学生注册信息
2    def student(name,sex, ** others):      #others 前面 ** ,表明其可接收多余的字典参数
3          dic = {}
4          dic["name"] = name
5          dic["sex"] = sex
6          for k in others.keys():
7                dic[k] = others[k]
8          return dic
9    students = []
10   st1 = student("zhangsan","Male")       #没有多余的参数,所以此时 others 为空
11   st2 = student("lili","Female",age = 18,province = "jiangsu")   #多余的参数传入 others
12   students.append(st1)
13   students.append(st2)
14   for iterm in students:
15         print(iterm)
```

实例 5.11 中定义了一个 student 函数，该函数含 3 个参数：name、sex 和 others，前两个参数为普通参数，最后一个参数 others 前有 ** ，表明它是可变长字典参数，可接收匹配 name 和 sex 之后所有格式如“关键字＝值”的参数。

第 10 行 st1＝student("zhangsan","Male")，函数调用中只有两个实参，将“zhangsan”传递给形参 name，将“Male”传递给 sex，others 什么也没有接收到。

第 11 行 st2＝student("lili","Female",age＝18,province＝"jiangsu")，函数调用中有 4 个实参，首先“lili”传递给 name，“Female”传递给 sex，剩下的 age＝18，province＝"jiangsu"将构成一个字典，传递给 others。运行结果如下：

```
{'name': 'zhangsan', 'sex': 'Male'}
{'name': 'lili', 'sex': 'Female', 'age': 18, 'province': 'jiangsu'}
```

这种可变长字典参数实现了灵活的调用，可根据需要传递可选信息，使函数更具通用性。

总结来说，可变长字典参数用于接受函数调用中的额外关键字参数，是 Python 实现灵活参数调用的一种重要方式。

当然函数定义时既可包括可变长元组参数，也可包括可变长字典参数，其一般格式如下：

```
def func_name(formal_args, * args, ** kwargs):
  statements
  return expression
```

formal_args 代表一组普通参数，* args 代表一个可变长元组参数，** kwargs 代表一个可变长字典参数。注意，在函数定义时，必须是普通参数在前，可变长元组参数在后，可变长字典参数在可变长元组参数之后。def func_name(formal_args, ** kwargs, * args)、def func_name(* args, formal_args, ** kwargs)、def func_name(* args, ** kwargs, formal_args)等顺序都是错误的，必须采用普通参数、可变长元组参数、可变长字典参数的顺序。

在函数调用时，实参会优先匹配普通参数，如果二者个数相同，那么可变长参数将获得空的元组和字典；如果实参个数大于传统参数个数，且匹配完传统参数后多余的参数没有指定名称，那么将以元组的形式存放这些参数，如果指定了名称，则以字典的形式存放这些命名的参数。

【实例 5.12】 可变长元组参数和可变长字典参数综合示例。

```
1   def example(a, * args, ** kwargs):
2       print("a:",a)
3       print("args:",args)
4       print("kwargs:",kwargs)
5   example(1,2,3,4,5,6,7,8,name = 'Python',age = 30,)   #有可变长元组参数,也有可变长字典参数
6   print(" ********************** ")
7   example(4,5,6)              #有可变长元组参数,没有可变长字典参数
8   print(" ********************** ")
9   example(2,name = "Lili")    #有可变长字典参数,没有可变长元组参数
10  print(" ********************** ")
11  example(3)                  #只有普通参数,没有可变长元组参数,也没有可变长字典参数
```

实例 5.12 定义了一个包含一个普通参数、一个可变长元组参数、一个可变长字典参数的函数，函数的功能比较简单，输出这 3 部分的内容。函数调用时参数传递的具体细节如图 5.5 所示。

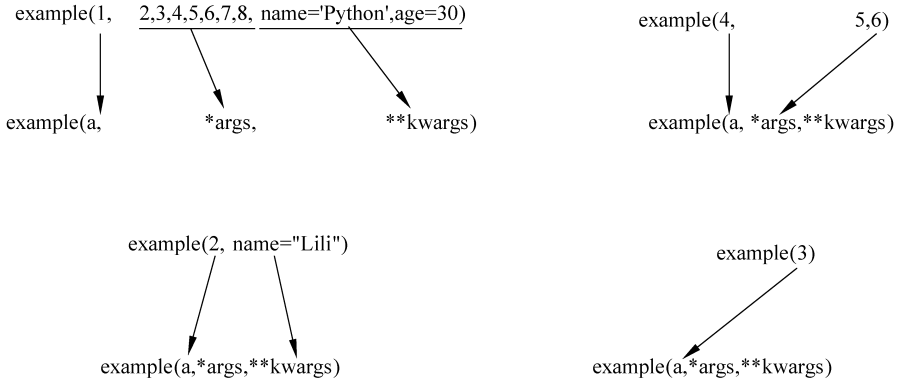


图 5.5 可变长参数传递示意图

运行结果如下：

```
a: 1
args: (2, 3, 4, 5, 6, 7, 8)
```

```
kwargs: {'name': 'Python', 'age': 30}
**********
a: 4
args: (5, 6)
kwargs: {}
**********
a: 2
args: ()
kwargs: {'name': 'Lili'}
**********
a: 3
args: ()
kwargs: {}
```

5.4 lambda 函数

如果一个函数在程序中只调用一次，那么可使用 lambda 函数。lambda 函数为匿名函数，即函数没有具体的名称，而用 def 创建的方法是有名称的。其语法格式如下：

```
lambda [arg1[, arg2, … argN]]: expression
```

lambda 是匿名函数的关键字，冒号前面为参数，是可选的，如果没有参数，则 lambda 冒号前为空。冒号后边为匿名函数的表达式，注意表达式只能占用一行。

【实例 5.13】 lambda 函数示例。

```
1   add = lambda a,b:a + b#将 lambda 函数赋值给一个变量，通过这个变量间接调用该 lambda 函数
2   print(add(3,4))
3   #还可这样使用:
4   print((lambda a,b:a + b)(3,4))
5   my_number = 5
6   is_even = (lambda x: x % 2 == 0)(my_number)
7   print(is_even)              # 输出 False
8   my_string = 'hello world'
9   upper_string = (lambda x: x.upper())(my_string)
10  print(upper_string)         # 输出 'HELLO WORLD'
```

运行结果如下：

```
7
7
False
HELLO WORLD
```

在实例 5.13 中，第 1 行 lambda a,b: a+b 中，匿名函数的形参为 a 和 b，表达式 a+b 为函数的返回值。注意，lambda 函数需在定义的同时调用该函数，而不能采用普通函数先定义再调用的方式。lambda 经常作为其他函数的参数，如 sort、filter、map 等。

filter 函数用于过滤序列中不符合条件的元素，返回符合条件的元素组成的新列表。其语法格式为：

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

其中，第一个参数为函数，第二个参数为序列，序列中的每个元素作为参数传递给函数进行判断，然后返回 True 或 False，最后将返回 True 的元素加入新列表。例如，

```
>>> fil = filter(lambda x: x>10,[1,11,2,45,7,6,13])
>>> print(list(fil))   #执行结果为:[11, 45, 13]
```

上述代码中 filter 函数的第二个参数是一个列表：[1,11,2,45,7,6,13]，第一个参数是 lambda 函数：lambda x：x＞10，filter 函数取出符合 lambda 函数列表中的所有元素，即过滤不符合 lambda 函数的列表元素，得到一个新的列表，所以其运行结果为[11，45，13]。

map 函数会根据提供的函数对指定序列进行映射，返回一个将 function 应用于 iterable 中每一项并输出其结果的迭代器，其语法格式为：

```
map(function, iterable, …)。
```

其中，第一个参数为函数，第二个参数为一个或多个序列。

【实例 5.14】 lambda 作为 map 函数参数示例。

```
1    mp1 = map(lambda x: x ** 2, [1, 2, 3, 4, 5])   # 使用 lambda 匿名函数作为参数
2    print(list(mp1))
3    mp2 = map(lambda x, y: x + y, [1, 3, 5, 7, 9], [2, 4, 6, 8, 10])   # 提供了两个列表,对相
4    # 同位置的列表数据进行相加
5    print(list(mp2))
6    mp3 = map(lambda x: x % 2 == 1, [1,3,2,4,1])
7    print(list(mp3))
```

运行结果如下：

```
[1, 4, 9, 16, 25]
[3, 7, 11, 15, 19]
[True, True, False, False, True]
```

在实例 5.14 中，第 1 行代码中 map 函数的第一个参数是 lambda 函数(lambda x：x ** 2)，第二个参数是一个列表(［1，2，3，4，5］)，利用 map 函数，对第二个参数(即列表中的每个元素)执行 lambda 函数，注意其返回值为 map 对象。第 2 行代码将 map 对象 mp1 转为 list，所以结果是：［1，4，9，16，25］。第 3 行代码 mp2＝map(lambda x，y：x ＋ y，［1，3，5，7，9］，［2，4，6，8，10］)中的第一个参数是 lambda 函数，第二和第三个参数分别为列表，其功能是对两个列表相应位置的元素执行 lambda 函数，所以其运行结果是：［3，7，11，15，19］。请读者自行分析第 5 行代码的功能。

5.5 变量的作用域

变量的作用域是指变量起作用的范围，即能够在多大范围内访问它。

【实例 5.15】 变量的作用域示例。

```
1    def my_func():
2        a = 10
3        print("a:{}".format(a))
4        print("b:{}".format(b))
5    b = 20
```

```
6    my_func()
7    print("b:{}".format(b))
8    print("a:{}".format(a))
```

运行结果如下：

```
a:10
b:20
b:20
Traceback (most recent call last):
  File "C:/python 教材/n5_14.py", line 8, in <module>
    print("a:{}".format(a))
NameError: name 'a' is not defined
```

在实例 5.15 中，my_func 函数内部定义了一个变量 a，在函数外部定义了一个变量 b。在 my_func 函数中可以访问函数内部定义的 a，也可以访问函数外部定义的 b。但是，在函数外部可以访问 b，但是不能访问 my_func 函数内部定义的变量 a。因为 my_func 中定义的变量 a 称为局部变量，只能在该函数范围内访问；在函数外部定义的变量 b 称为全局变量，在整个文件中都是可见的，都可以访问。

5.5.1　局部变量

局部变量是指在自定义函数内部定义的变量，其作用域为该函数内部。

【实例 5.16】　局部变量示例一。

```
1    def func1(x,y):
2        x1 = x
3        y1 = y
4        print("in func1, x1:{},y1:{},x:{},y:{}".format(x1,y1,x,y))
5    def func2():
6        x1 = 10
7        y1 = 20
8        print("in func2, x1:{},y1:{}".format(x1,y1))
9    func1(2,3)
10   func2()
```

运行结果如下：

```
in func1, x1:2,y1:3,x:2,y:3
in func2, x1:10,y1:20
```

此例在 func1 中定义了变量 x1 和 y1，它们是局部变量。在 func2 中也定义了 x1 和 y1，它们也是局部变量。在 func1 中访问的 x1 和 y1 是 func1 中定义的 x1 和 y1，在 func2 中访问的 x1 和 y1 是 func2 中定义的 x1 和 y1。由此可见，局部变量的作用域是其所在的函数，即只能在此函数内部访问。下面稍微修改一下此例。

【实例 5.17】　局部变量示例二。

```
1    def func1(x,y):
2        x1 = x
3        y1 = y
```

```
    print("in func1, x1:{},y1:{},x:{},y:{}".format(x1,y1,x,y))
    func2()   #在函数 func1 中调用函数 func2
def func2():
    x1 = 10
    y1 = 20
    print("in func2, x1:{},y1:{}".format(x1,y1))
func1(2,3)
```

此例中，func1 函数中调用了函数 func2，但是局部变量的作用域没有改变，即 func1 中定义的局部变量 x1 和 y2 的作用域仍然为 func1 内部，在 func2 中访问的 x1 和 y1 是 func2 中定义的局部变量 x1 和 y1。其运行结果如下：

```
in func1, x1:2,y1:3,x:2,y:3
in func2, x1:10,y1:20
```

5.5.2 全局变量

全局变量是指在函数外部定义的变量，其作用域为整个程序，即该程序中的所有函数都可以访问全局变量。

【实例 5.18】 全局变量示例。

```
z = 100                    #全局变量
def func1(x,y):
    x1 = x
    y1 = y
    print("in func1, x1:{},y1:{},x:{},y:{},z:{}".format(x1,y1,x,y,z))
def func2():
    x1 = 10
    y1 = 20
    print("in func2, x1:{},y1:{},z:{}".format(x1,y1,z))
func1(2,3)
func2()
print("z:{}".format(z))
```

此例中，在函数外部定义了一个全局变量 z，在函数 func1 和 func2 中均可访问此变量。运行结果如下：

```
in func1, x1:2,y1:3,x:2,y:3,z:100
in func2, x1:10,y1:20,z:100
z:100
```

【实例 5.19】 局部变量与全局变量同名示例。

```
z = 100                    #全局变量
def func1(x,y):
    x1 = x
    y1 = y
    z = 50                 #同名局部变量
    print("in func1, x1:{},y1:{},x:{},y:{},z:{}".format(x1,y1,x,y,z))
def func2():
    x1 = 10
```

```
 9          y1 = 20
10         print("in func2, x1:{},y1:{},z:{}".format(x1,y1,z))
11     func1(2,3)
12     func2()
13     print("z:{}".format(z))
```

运行结果如下：

```
in func1, x1:2,y1:3,x:2,y:3,z:50
in func2, x1:10,y1:20,z:100
z:100
```

实例 5.19 中，在函数外部定义了一个全局变量 z，在 func1 中也定义了一个变量 z，这里的 z 是在函数内部定义的，是一个局部变量，那么在 func1 中访问的 z 实际上是局部变量 z。

全局变量是在整个文件中声明，全局范围内都可以访问。局部变量是在某个函数中声明，只能在该函数内访问，如果试图在超出范围的地方访问，程序就会出错。如果在函数内部定义与某个全局变量一样名称的局部变量，那么在该函数内部访问此名称时，访问的是局部变量。无论在函数内如何改动此变量的值，只在函数内生效，对全局来说没有任何影响。这也侧面说明函数的局部变量优先级高于全局变量。

5.5.3　global 关键字

如果需要在函数体内修改全局变量的值，就要使用 global 关键字，即告诉 Python 编译器，此变量不是局部变量，而是全局变量。

【实例 5.20】　global 关键字示例。

```
1     num1 = 6
2     def fun1():
3          num1 = 2
4          print("函数内修改后 num1 = ",num1)
5     print("运行 func1 函数前 num1 = ",num1)
6     fun1()
7     print("运行 func1 函数后 num1 = ",num1)
```

运行结果如下：

```
运行 func1 函数前 num1 =  6
函数内修改后 num1 =  2
运行 func1 函数后 num1 =  6
```

实例 5.20 中声明了一个全局变量 num1＝6，第 3 行代码在函数 func1 内部修改变量 num1 的值，调用函数 func1 后第 7 行代码输出 num 的值，发现在函数外部 num 的值并没有变化。这是因为函数内部的 num1 是一个局部变量，对局部变量的任何修改不会影响同名的全局变量。如果要在函数内部修改全局变量的值，可用关键字 global 进行声明。下面修改这个程序。

【实例 5.21】　global 声明全局变量示例。

```
1     num1 = 6              #全局变量
2     def fun1():
```

```
3          global num1    #用 global 声明变量 num1,意味着在此函数内部访问的 num1 为全局变量
4          num1 = 2
5          print("func1 函数内修改后 num1 = ",num1)
6      print("运行 func1 函数前 num1 = ",num1)
7      fun1()
8      print("运行 func1 函数后 num1 = ",num1)
```

实例 5.21 的 func1 中使用 global 声明 num1，那么该函数内部访问的 num1 就是全局变量，所以对其进行修改也就是对全局变量进行修改。

运行结果如下：

```
运行 func1 函数前 num1 = 6
func1 函数内修改后 num1 = 2
运行 func1 函数后 num1 = 2
```

【实例 5.22】 全局变量的错误示例。

```
1      gcount = 10                    #全局变量
2      def global_test():
3          gcount * = 2               #试图访问全局变量
4          print (gcount)
5      print("在运行函数之前 gcount",gcount)
6      global_test()
7      print("在运行函数之后 gcount",gcount)
```

实例 5.22 是一个错误示例，因为在函数 global_test 中，第 3 行 gcount＝＊2 试图直接访问全局变量并进行运算，此时会提示错误“UnboundLocalError：local variable 'gcount' referenced before assignment”。所以，如果想要访问全局变量，可在函数内部使用关键字 global 进行声明。读者可自行修改这段代码，使之能够正确运行。

5.6 递归函数

递归是一个有趣的概念，可通过类似小明确定座位的例子理解递归的基本思想。

小明和同学们进电影院随便坐下看电影，小明想知道自己坐在电影院的第几排，他问自己前面一排的人是第几排，前面的人再问自己前面的人同样的问题……以此类推，类似问题被传递下去，这就是问题的“递”。当问题传到第一排的人时，他会给第二排的人一个确定的答案：1。然后第二排的人将其得到的答案 2 传给第三排的人，第三排的人再传给第四排的人，以此类推。假设小明最后从其前面的人处得到的答案为 9，那么他就可以确定自己坐在第 10 排。此过程中答案不断被构造出来并回传，称为“归”。

在此过程中，问题被一层一层地传递下去，而答案被一层一层地传递回来。这就是递归的基本思想：把一个大型复杂问题层层转化为一个与原问题相同但规模更小的问题，问题被拆解为子问题后，递归调用继续进行，直到子问题无须进一步递归即可解决为止。

如果函数中存在着调用函数本身的情况，这种现象叫作递归。如小明确定座位的例子，如果将确定座位定义为一个函数 ask_row_num，小明确定排次就是 ask_row_num(n)，前排的人确定排号就是 ask_row_num(n－1)，在 ask_row_num(n)中需要执行 ask_row_num(n－1)＋1，

这就是递归，因为在 ask_row_num(n)中调用了 ask_row_num(n－1)，同时还有一个终止往下调用的条件，就是第一排的人不能再问前排的人了，他需要告诉第二排的人，他所在的排号为 1。因此该函数可写为：

```
def ask_row_number(n):
    if n == 1:
        return 1
    else:
        return ask_row_number(n - 1) + 1
```

【实例 5.23】 请编写程序，使用递归实现阶乘。

分析：n!＝n＊(n－1)＊(n－2)＊…＊2＊1 可改写为：

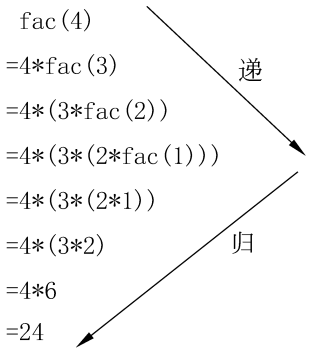
$$n!=\begin{cases}1 & n=1\\ n*(n-1)! & n>1\end{cases}$$

```
def fac(n):
    if n == 1:
        return 1
    else:
        return n * fac(n - 1)
m = 4
print(m,"!= ",fac(m))
```

运行结果如下：

```
4 != 24
```

递归的过程可分为“递”和“归”。下面是求 fac(4)的过程。



构成递归需具备以下两个条件。

1）基线条件

一个递归函数必须有一个基线条件，也就是递归的出口或终止条件。这是递归能够终止的条件，是最简单的情况。例如计算阶乘时，基线条件就是 n＝＝1，直接返回 1。

2）递归条件

递归函数需要递归条件以调用自身。在这个条件下，递归函数缩小问题的规模，为下一次递归调用简化问题。以阶乘为例，递归条件为：如果 n 不等于 1，就调用 fac(n－1)，以达到简化问题的目的。

由上述分析可得出递归函数的模板：

```
def recursion(n):
    if (终止条件):
        return ****
    else:
        recursion(n-1)
```

使用递归具有以下优点：递归函数使代码可读且易于理解，还可将复杂函数简化。

但递归也有其劣势，因为要进行多层函数调用，会消耗较大的堆栈空间和较长的函数调用时间。在调用深度达到1000后Python会停止函数调用。运行print(fac(1000))会出现以下错误提示：

```
RecursionError: maximum recursion depth exceeded in comparison
```

【实例5.24】 利用递归实现二分查找。

在计算机科学中，查找算法用于在数据结构中搜索需要的元素，常见的有顺序查找和二分查找。顺序查找从头到尾依次检查每个元素，直到找到目标元素，时间复杂度为$O(n)$。二分查找利用数据的有序性，时间复杂度可降至$O(\log n)$。假设数据序列nums是按照升序排序的，要查找的元素为target，二分查找的思路如下。

初始搜索的索引范围为0～len(nums)－1，令left＝0，right＝len(nums)－1

(1) 从中间位置mid＝(left＋right)/2开始比较。

(2) 如果nums[mid]＝＝target，则查找结束，返回mid。

(3) 如果nums[mid]＞target，则继续在左半区间查找，令right＝mid－1，查找范围为left～right。

(4) 如果nums[mid]＜target，则继续在右半区间查找，令left＝mid＋1，即查找范围为left～right。

(5) 如果查找范围为空，即left＞right，则查找不成功，返回－1。

由此可见，递归的递归条件和基线条件如下。

递归条件：当未找到目标时，缩小范围继续搜索左半区间或右半区间。

基线条件：找到目标值或查找范围为空。

以上介绍了二分查找的基本思路，下面给出其递归实现源代码。

```
def binary_search(nums, target,left,right):
    '''
    #在nums列表中的索引范围为left～righ中查找元素target,查找成功,返回target在
    #nums中的索引,不成功返回-1
    '''
    if left > right:                    #递归的终止条件1,查找空间为空,即查找失败
        return -1
    mid = (left + right) // 2
    if nums[mid] == target:             #递归的终止条件2,查找成功
        return mid
    elif nums[mid] > target:
        return binary_search(nums, target, left, mid-1)   #递归条件:左半区间查找
    else:
        return binary_search(nums, target, mid+1, right)  #递归条件:右半区间查找
```

```
15    nums = [0, 4, 5, 7, 9, 10, 12, 13, 14, 15]
16    right = len(nums) - 1
17    print(binary_search(nums,7,0,right))
```

5.7 main 函数与模块

在多数程序设计语言中，都有一个特殊的函数——main 函数，它在程序每次运行时自动执行，是程序执行的起点。但是在 Python 中，可以有 main 函数，也可以没有 main 函数，这是因为 Python 是一种解释型脚本语言，执行之前不需要将所有代码先编译为中间代码，程序运行时从程序的第一行开始，逐行进行翻译执行。所以，除了 def 后定义的函数之外的代码都会被认为是“main”函数中的内容，从上而下执行。但是，规范化的程序还是需要 main 函数的，如实例 5.25 所示。

【实例 5.25】 main 函数示例。

```
1    #test.py
2    def hello():
3        print("this is hello function")
4    def main():
5       hello()
6    if __name__ == "__main__":
7        main()
```

在实例 5.25 中，定义了两个函数 hello 和 main，在 main 函数中调用了 hello，第 6 行代码“if __name__==’__main__:”相当于一个标志，象征着程序主入口，程序就从第 6 行开始执行，是程序的入口，执行 main 函数。这是一个比较规范的 Python 程序代码，可以此为模板写程序。

函数是完成特定功能的一段程序，是可复用程序的最小组成单位，模块是在函数和类的基础上将一系列相关代码组织到一起的集合体。模块(module)实际上是一个 Python 文件，以.py 结尾，包含了 Python 对象定义和 Python 语句。通过模块可实现模块化程序设计，以更有逻辑地组织程序。在模块中可以定义函数、类和变量，模块还可以包含可执行的代码。

例如，在 greet.py 中定义一个 hello 函数，再在 test_module.py 中使用 hello 函数，可在文件前面通过 from greet import hello 导入模块。

【实例 5.26】 模块示例。

下面是 greet.py 文件，该文件中定义了一个函数 hello。

```
1    #gree.py
2    def hello():
3        print("this is hello function")
```

下面是 test_module.py 文件，该文件使用了 greet.py 文件中的 hello 函数。

```
1    #test_module.py
2    from greet import hello
```

```
def main():
    hello()
if __name__ == "__main__":
    main()
```

5.8 综合例子

【实例 5.27】 使用蒙特卡洛方法计算 π。其基本思想是：边长为 1 的正方形内有一个内切圆，随机扔一点在圆内的概率为 π/4。使系统随机生成 N 个横坐标和纵坐标都小于 1 的点，如果该点距离圆心的距离小于 1，那么说明该点落在了圆内。设落入圆内点的个数为 hits，那么 pi=(hits * 4)/(N * N)。

```
from random import random
def calPI(N = 100):
    hits = 0
    for i in range(1, N * N + 1):
        x, y = random(), random()              # 随机生成一个坐标
        dist = pow(x ** 2 + y ** 2, 0.5)       # 计算该坐标距离圆点的距离
        if dist <= 1.0:                        # 说明该点落在了圆内
            hits += 1
    pi = (hits * 4) / (N * N)
    return pi
m = 10
for i in range(0,4):   # 比较有 10 个点、100 个点、100 个点、1000 个点情况下的 pi 值
    n = m * pow(10,i)
    PI = calPI(n)
    print("{} points PI: {}".format(n,PI))
```

运行结果如下：

```
10 points PI: 3.12
100 points PI: 3.1424
1000 points PI: 3.144944
10000 points PI: 3.14190524
```

【实例 5.28】 小学生计算器。利用函数实现 100 以内的加减法、10 以内的乘除法功能。

```
import random
def compute():
    op = random.choice('+- * /')
    if op == '*':
        op1 = random.randint(0, 10)
        op2 = random.randint(0, 10)
        result = op1 * op2
    elif op == '/':
        op2 = random.randint(0, 10)
        m = random.randint(0, 10)
        op1 = op2 * m
        result = op1 / op2
```

```
        else:
            op1 = random.randint(0, 100)
            op2 = random.randint(0, 100)
            result = op1 + op2
            if op == '-':
                if op1 < op2:
                    op1, op2 = op2, op1
                result = op1 - op2
        print(op1, op, op2, '=', end = '')
        return result
count = 10
for i in range(10):
    answer = compute()
    yAnswer = int(input())
    if yAnswer == answer:
        count = count + 10
print("your score is :", count)
```

【实例 5.29】 利用函数输出 2～20 之间的所有素数。

```
import math
def prime(m):                          #判断 m 是否为素数,如果是,返回 1,否则返回 0
    i = 2
    while (i <= math.sqrt(m)):
        if m % i == 0:
            return 0
        i = i + 1
    return 1
print("2.20 之间的素数有:")
for m in range(2,20):
    if(prime(m) == 1):
        print(m," ",end = '')
```

运行结果如下：

```
2～20 之间的素数有:
2  3  5  7  11  13  17  19
```

【实例 5.30】 如果一个 3 位数等于其各位数字的立方和，则称这个数为水仙花数。例如：153＝13＋53＋33，因此 153 是一个水仙花数。利用函数求 1000 以内的水仙花数(3 位数)。

```
def Narcissistic(i):
    a = i//100
    b = (i-a*100)//10
    c = (i-a*100-b*10)
    if i == pow(a,3)+pow(b,3)+pow(c,3):
        return 1
    else:
        return 0
print("三位数的水仙花数有:")
for i in range(100,1000):
    if Narcissistic(i) == 1:
        print(i)
```

运行结果如下：

```
三位数的水仙花数有：
153
370
371
407
```

【实例 5.31】 猜数字。

本案例的任务：系统随机生成一个 1～100 之间的整数，让用户猜测该数字，如果用户猜的数据比答案大，提示太大了；反之则提示太小了，正确则提示用户猜测正确。

案例分析：根据案例需要实现的功能，可将任务分解为两个子任务，产生数字和用户猜数字，并通过两个函数实现。

源代码如下：

```
import random
def main():
    number = newNumber()          #1. 系统生成一个随机数并放到 number 中
guessNumber(number)               #2. 让用户猜测 number 的值
def newNumber():
    number = random.randint(1,101)
    return number
def guessNumber(number):
    yAnswer = int(input("enter a integer between 0 - 100:"))
    if (yAnswer > number):
        print("too big")
    elif yAnswer < number:
        print("too small")
    else:
        print("right")
if __name__ == '__main__':
    main()
```

其中定义了两个函数 newNumber 和 guessNumber，newNumber 函数的主要功能是产生一个 1～100 的随机整数，并将该数字返回。guessNumber 函数的功能是使用户输入一个整数，比较用户输入的数字和参数 number 的大小，并输出相应的提示信息。在主函数 main 中依次调用这两个函数。

上面的程序只给了用户一次机会，现在考虑给用户 10 次机会，因此程序中增加了一个函数 guessTime，该函数通过一个循环调用 guessNumber 函数，循环次数为 10。源代码如下：

```
import random
def main():
    number = newNumber()          #1. 系统生成一个随机数并放到 number 中
guessTime(number)                 #2. 让用户猜测 number 的值
def newNumber():
    number = random.randint(1,101)
    return number
def guessNumber(number):
```

```
        yAnswer = int(input("enter a integer between 0 - 100:"))
        if yAnswer > number:
            print("too big")
        elif yAnswer < number:
            print("too small")
        else:
            print("right")
    def guessTime(number):
        for i in range(10):
            guessNumber(number)
    if __name__ == '__main__':
        main()
```

此程序中，增加了一个函数 guessTime，该函数的功能是允许用户猜测 10 次。但是运行程序会发现，即使猜对了，程序仍然让用户继续猜测，所以还要对程序继续修改，使用户猜测正确时跳出循环。改进的关键在于 guessNumber 函数，如果 guessNumber 函数对于猜测结果仅给出提示信息，不返回任何值给 guessTime 函数的话，就无法控制退出循环，因此考虑修改 guessNumber 函数，如果猜对，除输出提示信息外，还返回 1；如果猜错，则提示太大或者太小，并返回 0。这样 guessTime 可根据 guessNumber 的返回值判断是否退出循环。因此程序修改如下：

```
import random
def main():
    number = newNumber()   #1.系统生成一个随机数并放到 number 中
times = 10   #2.给用户多次机会，机会次数存储到 times 变量中。让用户猜测数字
    guessTime(number,times)
def newNumber():
    number = random.randint(1,101)
    return number
def guessNumber(number):
    yAnswer = int(input("enter a integer between 0 - 100:"))
    if yAnswer > number:
        print("too big")
        return 0
    elif yAnswer < number:
        print("too small")
        return 0
    else:
        print("right")
        return 1
def guessTime(number,times):
    for i in range(times):
        if(guessNumber(number) == 1):
            print("你一共猜了",i + 1,"次")
            break
    if(i > times):
        print("sorry,你没有猜对，正确答案是",number)
if __name__ == '__main__':
    main()
```

从上述案例可以看出，为实现猜数字的任务，可将主任务分解为 3 个子任务：产生随机

数函数、判断猜测的数字是否正确的函数，以及让用户猜测多次的函数。这3个函数相互依赖，首先产生随机数newNumber，这是后面猜数字的基础，因为必须先有猜测的对象，才能让用户猜测。其次guessNumber让用户输入自己的猜测并判断是否正确，而目标是让用户猜测多次，因此再创建一个函数guessTime(number, times)，该函数调用times次guessNumber，从而实现用户最多可猜测times次，如果猜对，则退出循环。

5.9 花样滑冰模拟计分系统——一起向未来

5.9.1 思政导入

第24届冬季奥林匹克运动会简称2022年北京冬奥会，是由中国举办的国际性奥林匹克赛事，于2022年2月4日开幕，2月20日闭幕。此届冬奥会共设滑雪、滑冰、冰球、冰壶、雪车、雪橇和冬季两项7个大项，高山滑雪、自由式滑雪、单板滑雪、跳台滑雪、越野滑雪、北欧两项、短道速滑、速度滑冰、花样滑冰、冰壶、冰球、雪车、钢架雪车、雪橇和冬季两项15个分项及109个小项。

中国体育代表团总人数为387人，其中运动员176人，教练员、领队、科学医护人员等运动队工作人员164人，团部工作人员47人，创中国体育代表团历届冬奥会参赛规模之最。中国体育代表团完成了北京冬奥会全部7个大项、15个分项的“全项目参赛”任务，共获得9金4银2铜，位列奖牌榜第三，金牌数和奖牌数均创历史新高。其中，隋文静和韩聪以总成绩239.88分获得花样滑冰双人滑比赛金牌，实现了花滑双人滑全满贯的壮举。

花样滑冰起源于18世纪的英国，后相继在德国、美国、加拿大等欧美国家迅速开展。与其他竞技运动不同，花样滑冰是一项艺术与运动结合的体育项目，除了要掌握冰上技术，对运动员的艺术表现力有极高的要求。在音乐伴奏下，运动员在冰面上滑出各种图案、表演各种技巧和舞蹈动作，裁判员根据动作评分，决定名次。冬奥会花样滑冰包括4个项目：男子单人滑、女子单人滑、双人滑和冰舞，比赛均在室内进行。它要求在60米×30米的冰场上，运动员以40千米/时的速度完成各种高难度动作，同时还要用自己的艺术表演诠释背景音乐，感染裁判和观众。此项运动涵盖体育、艺术、音乐、舞蹈、服装设计、化妆……因此对运动员和教练技术以外的要求也非常高。

花样滑冰是比赛规则最复杂、评分难度最高的体育项目之一，评委需在高速运动且变化繁杂的动作中依据动作的类型、难度系数、完成情况、标准程度等给出精准的技术分，通过AI技术辅助评分难度也可见一斑。2022年1月21日，花样滑冰AI辅助评分系统1.0发布，这套辅助系统是根据中国花样滑冰运动员使用需求、场景应用需求打造的AI+虚拟现实解决方案，运用计算机视觉技术算法与深度学习，以对运动员的整体运动轨迹进行实时追踪，根据专业评分标准，对视频数据的人体骨骼、形体动作进行捕捉识别，从而实现稳定性可视化的比赛评判。

5.9.2 案例任务

由于花样滑冰的评分规则比较复杂，可对其进行简化，简化后的规则如下：花滑总分数

由技术水平分和节目内容分两部分构成。每位评委分别为每位选手打出技术水平分和节目内容分。裁判组对每位选手的技术水平分和节目内容分去掉最高分和最低分，再将其平均分相加，即得到该选手的综合分。

请编写程序模拟评委打分、裁判组汇总分数、显示分数。

5.9.3 案例分析与实现

首先比赛选手名单可用列表 skater_lst 存储，假定有 3 位选手，按照列表中选手的顺序依次进行表演，表演完毕后，评委(假定有 5 位)给出该选手的技术水平分 element_score 和节目内容分 component_score。定义一个字典 player_score，其键名为选手姓名，键值为列表类型，将每位评委打出的两部分分数组成一个元组，作为列表元素。打分完毕后，该字典有 3 个元素，每个元素的键名为选手姓名，键值是一个包含 5 个元素的列表，每个元素是一个形如“(element_score，component_score)”的元组。

裁判组对每位选手进行分数计算，用一个 compute_score 函数实现。该函数的参数为字典 play_score，功能为：对于该字典中的每个键名(即选手姓名)，按照规则计算其最终技术水平分、最终节目内容分和总分，并将 3 部分内容合在一起构建一个元组，附加到该选手对应键值(列表类型)的末尾。

显示分数环节用 show 函数实现，参数为字典 play_score。其功能为根据总分排序并输出选手的名次、姓名、总分、技术水平分和节目内容分。

源代码如下：

```
import random
import time
skater_lst = ["金博洋","羽生结弦","陈巍"]   #定义列表,保存选手姓名信息
player_score = {}
num_judge = 5                                #评委人数

def compute_score(player_score):
    '''
    player_score:{player_name:[(element_score1,component_score1),...]}
    '''
    for name in player_score.keys():
        score_list = player_score[name]
        for i in range(len( score_list)):#len(player_score[name])为评委个数

            #将 element_score 汇总起来
            element_scores = [score_list[j][0] for j in range(len(score_list))]
            component_scores = [score_list[j][1] for j in range(len(score_list))]
        #去掉最高分、最低分,求均值
        max_element = max(element_scores)
        max_component = max(component_scores)
        min_element = min(element_scores)
        min_component = min(component_scores)
        final_element = (sum(element_scores) - max_element - min_element)/(len(element_
scores) - 2)
        final_component = (sum(component_scores) - max_component - min_component)/(len
(component_scores) - 2)
```

```
            final_score = final_element + final_component
            tmp = (round(final_score,2),round(final_element,2),round(final_component,2))
            player_score[name].append(tmp)

    def show(player_score):
        '''
        player_score:{player_name:[(element_score1,component_score1),...(final_score,
    final_element,final_component)]}
        '''
        #将 player_score 中的姓名、总分、技术分、内容分存为列表,排序
        score_lst = []
        for name in player_score.keys():
            score = player_score[name][-1]
            tmp = [name]
            tmp.extend(score)
            score_lst.append(tmp)
        score_lst.sort(key = lambda x:x[1],reverse = True)
        print(" 名次   选手姓名      总 分    技术分       内容分 ")
        for i in range(len(score_lst)):
            print(f"{i+1:3} ",end = '')

            for j in range(len(score_lst[i])):
                 print("{:^9}".format(score_lst[i][j]),end = "")
            print()

    def main():
        print(" *** 欢迎使用花样滑冰模拟计分系统 *** ")
        for i in range(len(skater_lst)):
            print(f"欢迎欣赏{skater_lst[i]}的花样滑冰")
            player = skater_lst[i]
            player_score[player] = []
            #每个评委为该选手打分
            for j in range(num_judge):
                print(f"请第{j+1}位评委为{player}打分")
    #               element_score = eval(input("请输入技术分:"))
    #               component_score = eval(input("请输入内容分:"))
                element_score = round(random.uniform(50, 100),2)#为便于测试,由系统随机
                                                                #生成分数
                component_score = round(random.uniform(50, 100),2)
                tmp = (element_score,component_score)
                player_score[player].append(tmp)
        print("现在是裁判组汇总分数时间")
        compute_score(player_score)             #计算每个选手的总分
        show(player_score)                      #按照从高到低的顺序输出选手的得分信息
    if __name__ == "__main__":
         main()
```

5.9.4 总结和启示

从 2008 年到 2022 年，奥林匹克两度携手中国。“冰丝带”盈盈飘动，“雪如意”雄踞山巅，“冰之帆”御风而行，“雪飞天”长袖善舞……

“成功举办北京冬奥会、冬残奥会，不仅可以增强我们实现中华民族伟大复兴的信心，而且有利于展示我们国家和民族致力于推动构建人类命运共同体，阳光、富强、开放的良好形象，增进各国人民对中国的了解和认识。”正如习近平总书记所言，2022 年北京冬奥会不仅是一场体育盛会，更折射出中国推动构建人类命运共同体的价值追求，具有深远的世界意义。

武大靖、苏翊鸣、谷爱凌等运动健儿在冬奥赛场上奋力拼搏，勇创佳绩，可喜可贺！同时，人工智能也在冬奥会上绽放异彩，花样滑冰 AI 辅助评分系统的发布也让我们看到互联网、人工智能、大数据等技术在体育运动领域的飞速发展。

5.10 本章小结

本章主要介绍了函数的定义和调用、函数的参数、变量的作用域、lambda 函数、filter 函数、map 函数、递归函数及 main 函数与模块的使用。其中函数的参数是本章的难点之一，包括形参和实参，参数之间的传递、默认参数、可变长参数及关键字参数。通过具体实例的讲解，读者对其中的概念有更直观、更深刻的理解。最后通过几个综合例子锻炼读者的综合编程能力。

5.11 巩固训练

【训练 5.1】 给定一个正整数，编写程序计算有多少对质数的和等于输入的这个正整数，并输出结果。输入值小于 1000。

【训练 5.2】 编写函数 change(str)，其功能是对参数 str 进行大小写互换，即将字符串中的大写字母转为小写字母、小写字母转换为大写字母。

【训练 5.3】 编写函数 digit(num,k)，其功能为：求整数 num 第 k 位的值。

【训练 5.4】 编写递归函数 fibo(n)，其功能为：求第 n 个斐波那契数列的值，进而输出前 20 个斐波那契数列。

【训练 5.5】 编写一个函数 cacluate，可接收任意个数，返回一个元组。元组的第一个值为所有参数的平均值，第二个值为小于平均值的个数。

【训练 5.6】 模拟轮盘抽奖游戏。轮盘分为三部分：一等奖、二等奖和三等奖。轮盘随机转动，如果范围在[0,0.08)之间，代表一等奖；如果范围在[0.08,0.3)之间，代表二等奖；如果范围在[0，1.0)之间，代表三等奖。

【训练 5.7】 有一段英文：What is a function in Python? In Python, function is a group of related statements that perform a specific task. Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable. Furthermore, it avoids repetition and makes code reusable. A function definition consists of following components. Keyword def marks the start of function header. A function name to uniquely identify it. Function naming follows the same rules of writing identifiers in Python. Parameters (arguments)

through which we pass values to a function. They are optional. A colon (:) to mark the end of function header. Optional documentation string (docstring) to describe what the function does. One or more valid Python statements that make up the function body. Statements must have same indentation level (usually 4 spaces). An optional return statement to return a value from the function.

任务：1. 请统计该段英文有多少个单词，以及每个单词出现的次数。2. 如果不算 of、a、the 这 3 个单词，给出出现频率最高的 10 个单词，并给出其出现的次数。

【训练 5.8】 利用函数实现磅(lb)与千克(kg)的转换。用户可以输入千克，也可以输入磅，函数将根据用户的输入转换为磅或千克。

【训练 5.9】 一个数如果恰好等于其因子之和，该数就称为“完数”，例如 6＝1＋2＋3。编程找出 1000 以内的所有完数。

【训练 5.10】 利用递归函数调用方式，将用户输入的字符串以相反的顺序输出。