第5章 函　数

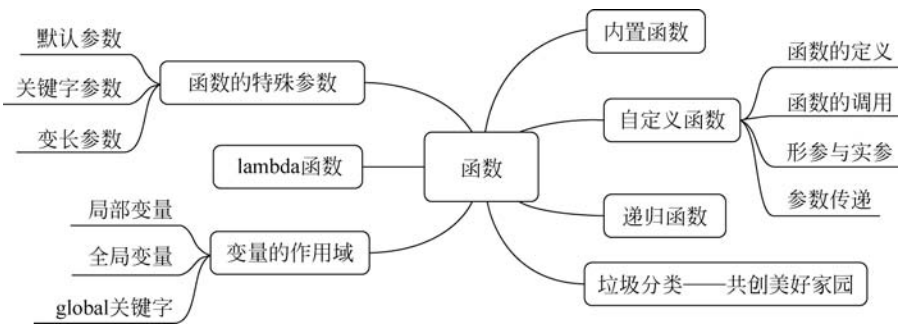
能力目标

【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，掌握函数参数传递的机制，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，将复杂问题先划分为几个部分，各个部分再进行细化，直到分解为能够较容易解决的子问题为止，每个子问题就变成独立的程序模块。每个模块构成整个算法的一部分，并完成一个单独的功能。使用模块可以使整个算法更简单、更有系统性，并且能够减少错误。由于每个模块只完成一个单独的任务，程序设计人员可以逐个模块地进行开发，设计出相应算法，所有模块的算法都开发完成后，一个复杂问题就得到了完全的解决。

利用函数，不仅可以实现程序的模块化，使得程序设计更加简单和直观，从而提高程序的易读性和可维护性，而且还可以把程序中经常用到的一些计算或操作编写成通用函数，以供随时调用。

本章的主要内容包括 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以及 lambda 函数和递归函数。

5.1 内置函数

为了方便用户，Python 提供了许多内置函数，例如 print、abs、len、int、max 等。表 5.1 列出了部分内置函数，内置函数是可以直接使用的函数。对于内置函数，需要了解函数的输入参数、返回值、函数的功能和具体的函数名。

表 5.1 常用的内置函数

函数名	函数功能	示 例
print(s)	输出字符串 s	print("hello,Python")
raw_input()	从用户键盘捕获字符，可以有参数，也可以没有参数，有参数时，参数作为提示语句显示	yname = raw_input() y_name = raw_input("enter your name:")
type(a)	返回参数 a 的类型	type(4)
int(a)	将参数 a 转化为整数并返回	y = int(3.5)
float(a)	根据参数 a 返回其对应的浮点数	y = float(3)
str(a)	返回参数 a 的字符串形式	y = str(35)
id(a)	返回参数 a 的内存地址	y = id(6)
pow(a,b)	返回参数 a 的 b 次幂	y = power(3,4)
abs(x)	返回参数 x 的绝对值。参数可以是实数也可以是复数，若参数是复数，则返回复数的模	y = abs(-3.4)
complex([r[, i]])	创建一个实部为 r，虚部为 i 的复数，其中参数 i 可选	y = complex(3,5)
divmod(a, b)	返回 a/b 的商和余数	x,y = divmoid(10,3)
range([b], e[, s])	生一个从 b 开始，到 e 结尾，步长为 s 的序列，默认从 0 开始，步长为 1	range(1,10,2)
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)
sum()	对可迭代对象参数求和	y = sum(1,2,3,4)

5.2 自定义函数

视频讲解

5.2.1 自定义函数的定义

当内置函数无法满足需求时，就需要自己创建函数，称为自定义函数，其语法格式为：

```
def func_name(arg1, arg2, arg3, ..., argN):
    statement(s)
    [return expression]
```

其中,函数的定义包括以下几个部分。

函数定义以 def 关键词开头,后接函数标识符名称即函数名和小括号( )。小括号中放的是函数的参数,也称为形式参数或者形参,若有多个参数,则参数之间用逗号隔开。函数可以有参数也可以没有参数,但是必须有小括号。小括号后面紧跟一个冒号,一定记得冒号不能省略。

函数体由一些语句构成,需要注意的是,函数体需要缩进。缩进块内的内容是函数的主体,没有被缩进的部分是不属于该函数的。

可以通过 return expression 将表达式的值返回给调用程序。当然函数可以没有返回值,这时在函数体内部就没有 return 语句。有时函数会有多个返回值。

总之函数的定义给出了函数的名称,指定了函数的参数和函数体,并且指出了有无返回值,有几个返回值。

【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 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 c_f,有一个参数 c
    f = 32 + c * 1.8
    return f
```

实例 5.1 定义了两个函数,分别是 greet 函数和 c_f 函数,其中 greet 函数没有参数,函数体是一条输出语句,没有返回值。c_f 函数有一个参数 c,在函数体中实现将摄氏度 c 转换为华氏度 f,并通过 return f 将计算出的华氏度 f 返回给调用函数。

运行上面的程序,可以发现该程序没有任何输出,因为这里只是进行了函数定义,函数的定义并不会使得函数执行。要运行该函数,必须通过函数调用。

5.2.2 自定义函数的调用

函数调用是指将一组特定的数据传递给被调用函数,然后启动函数体的执行,最后返回到主程序中的调用点并带回返回值的过程。调用方法如下:

```
func_name(par1,par2,…)
```

其中:

(1) func_name 为函数名,和函数定义时的函数名必须保持一致;

(2) 函数调用中函数名后边的小括号内的 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,函数定义中有多少个形参,调用时就需要传入多少个值,且顺序必须和函数定义时保持一致。另外,即使该函数没有参数,函数名后的小括号也不能省略。

【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 传递给形参 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 传递给形参 c
```

运行结果：

```
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 5.1 中定义了一个没有返回值的函数 greet，在该函数定义中没有 return 语句，调用这样的函数时的基本格式为“函数名(实参列表)”，正如实例 5.2 中的第 1 行代码所示。有返回值的函数，在函数定义中应有 return 语句。实例 5.1 中定义了一个 c_f 函数，该函数有一个返回值。在进行函数调用时可以出现在赋值表达式的右边，将函数的返回值赋值给一个变量，如实例 5.2 中第 2 行代码：f_temperature＝c_f(30)。也可以出现在表达式中，此时先计算出函数的返回值，然后参与运算，如实例 5.2 中第 4 行。

【实例 5.3】 有多个返回值的函数示例。

```
1  def ave_names(dic1):
2      scores = [values for values in dic1.values()]
3      sum = 0
4      for score in scores:
5          sum = sum + score
6      ave = sum/len(scores)
7      names = []
8      for name in dic1.keys():
9          if dic1[name]< ave:
10             names.append(name)
11     return ave,names
12 scores = {"zhangsan": 90,"lisi": 79,"wangwu": 56,"zhangliu": 72}
13 average,names = ave_name(scores)
14 print("平均分: ",average)
15 print("低于评价分的有: ",names)
```

运行结果：

```
平均分: 74.25
低于评价分的有: ['wangwu', 'zhangliu']
```

函数 ave_name 的参数是一个字典，该字典是以“姓名：分数”格式存储的成绩单，函数功能是获取平均分和低于平均分的同学名字。可以看到，在函数体的最后一行有“return ave,names”，即该函数有两个返回值。所以在进行函数调用时，需要用两个变量来接收返回的值。第 13 行代码：average,names＝ave_names(scores)，这样实现变量 average 接收返回的 ave,names 接收返回的 names。


视频讲解

5.2.3 形式参数和实际参数

函数的参数有形参和实参之分，其中形参指的是函数定义时函数名后面括号里的参数，多个形参用逗号“,”分隔，这些形参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。实参是在函数调用时函数名后小括号中的参数,用于给形参传递具体的值。在进行函数调用时参数传递的过程如图 5.1 所示。

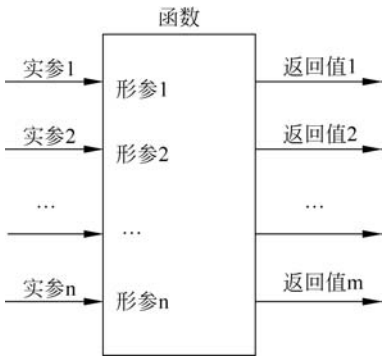


图 5.1　函数调用示意图

【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:

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

在实例 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)得到的值赋值给 result。

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 和 b 之和,然后返回两者之和,在 print 语句中,将结果输出。

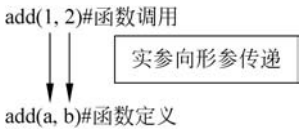


图 5.2　参数传递示意图

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,否则会导致错误。

假设仍然使用上面的 add 函数,如果进行如下的函数调用:

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

则会提示如下错误:

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

如果采用如下的函数调用(参数类型不对应):

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

则会提示如下错误:

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


视频讲解

5.2.4 参数传递

函数调用时,实参向形参传递,传递的方式有两种:值传递和引用传递。在值传递过程中,形式参数作为被调函数的局部变量处理,即在堆栈中开辟了内存空间来存放由实参传递过来的值,从而成为了实参的一个副本。不管在函数中对这个形参如何操作,实际参数值本身不会受到任何影响。如果传入的参数对象是可变对象——列表或字典,则是引用传递。可变对象参数在函数体内被修改,那么源对象也会被修改。

【实例 5.5】 参数传递示例。

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

运行结果:

```
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的,函数的功能是将该参数的值增加 1。plus_two 函数的参数是一个列表,其功能是将列表中的每个元素值增加 2。在进行函数调用 plus_one(m)时,m 的值传递给了参数 x,在函数体中将 x 的值增加了 1,但是对形参 x 的改变并没有影响到实参 m,所以调用函数之前和之后 m 的值都是 5,没有任何改变。在进行函数调用 plus_two(list1)时,实参 list1 的值传递给了形参 list,在

函数体中将形参 list 中的每个元素值都增加 2,之后输出 list1 的值,发现实参 list1 的值发生了改变——每个元素的值都增加了 2。

当参数是数值类型、字符串类型和布尔类型、元组时,形参的改变不会影响实参,这样的数据类型称为不可变数据类型。当参数是列表、字典时,形参的改变意味着实参的改变,这样的数据类型称为可变数据类型。

5.3 函数特殊参数

视频讲解

5.3.1 默认参数

函数参数可以在 Python 中具有默认值,在定义函数时使用赋值运算符“=”为形式参数提供默认值。调用时如果不给此参数传递实参值,则会使用默认值;如果给默认参数传递了实参值,则使用传入的实参值。

【实例 5.6】 默认参数示例。

```
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
```

运行结果:

```
5 ** 2 = 25
4 ** 3 = 64
```

上面的例子中定义了一个 power 函数,这个函数有两个参数:x 和 n。其中 n 是默认参数,设置的默认值是 2,y=power(5),只传入了 x 的值,并没有传入 n 的值,所以 n 采用默认值 2,因此其结果为 25。在 power(4,3)中,4 传给了 x,3 传给了 n,结果就是 64。

默认参数只能出现在参数列表的最后,其后面不能出现非默认参数。

【实例 5.7】 默认参数的错误实例。

```
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：

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

5.3.2 关键字参数


视频讲解

前面学习的参数传递，是按照位置来传递的，例如，如果函数的定义是 func_1(a，b，c，d)这样的形式，那么在函数调用时 func_1(x，y，z，w)就会从左到右依次匹配，形成如图 5.3 所示的参数传递效果。

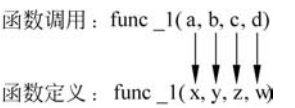


图 5.3 参数传递

可以看出，在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，会产生意想不到的运行结果。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
```

可见，按位置传递时，实参顺序不能有误，而关键字实参对于实参的位置没有要求。关键字参数在函数调用时实参采用“形参名＝实参”的格式，明确将实参值传递给某个形参，实现一种显式的参数匹配效果，从而摆脱位置的约束。

【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  student_information("wangwu",college = "computer science",age = 18)   #第一个参数按照
   #位置传递,后面两个按照参数名称传递
```

实例 5.8 中定义了一个函数，该函数有 3 个参数，第 5 行代码函数调用：student_information("zhangsan",18,"computer science")，将这 3 个实参的值依次传递给 name、age、college。第 6 行函数调用 student_information("wangwu",college＝"computer science",age＝18)，既有位置参数又有关键字参数，首先位置参数依次传递给对应位置的形参，即"wangwu"传递给 name，然后按照关键字名称传递参数，即将"computer science"传递给 college，将 18 传递给 age。

需要注意的是，位置参数和关键字参数都是针对实参的，在函数定义中，形参还是和原来一样。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，则根据关键字名字来传递参数；若既有位置参数又有关键字参数，则在进行函数调用时，按照下面的格式进行参数传递：

```
funcname([位置参数],[关键字参数])
```

注意这个前后顺序是严格的，即仍然位置参数在前，关键字参数在后。

所以 student_information(college="computer science",age=18,"wangwu")就会出现下面的错误提示：

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



视频讲解

5.3.3 可变长度参数

到目前为止，我们要定义一个函数时，必须要预先定义这个函数需要多少个参数(或者说可以接收多少个参数)。一般情况下这是没问题的，但是有时无法知道参数个数或者个数不确定，这时可以用带 * 的参数来接收可变数量参数。这个可变的参数称为元组变长参数。

定义变长元组参数的一个格式为：

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

当定义包含元组变长参数的函数时，普通形参 formal_args 在前，元组变长参数 args 在后。普通形参可以有多个，而元组变长参数只能有一个。

在函数调用时，会将实参依次匹配前面的普通参数 formal_args，之后剩下的所有实参将构成一个元组传递给元组变长参数 args。

【实例 5.9】 元组变长参数示例一。

```
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)
```

函数定义时有 3 个参数 a、b、c，第 5 行函数调用 func(1,2,3,4,5)，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，剩下的所有实参的值 3、4、5 组成一个元组全部传递给了 c，第 6 行函数调用 func(1,2)，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。其运行结果如下：

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

图 5.4 可变参数传递示意图

【实例 5.10】 元组变长参数示例二。

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

运行结果：

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

需要注意的是，函数定义有两个参数 cutoffVal 和 * vals，第 9 行在函数调用 ar＝lessThan(average,89,34,78,65,52)时，首先将 average 传递给 cutoffVal，然后将剩余的参数 89、34、78、65、52 作为一个元组全部传递给了 vals。

Python 还提供了另外一种变长参数——字典变长参数。字典变长参数的表示形式为在参数名称前面加两个星号“**”，函数调用时会将溢出的关键字实参全部接收到字典变长参数中。

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

定义包含字典变长参数的函数时，普通形参 formal_args 在前，变长字典参数 kwargs 在后。普通形参可以有多个，而变长字典形参只能有一个。

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

【实例 5.11】 变长字典参数示例。在学校新生开学注册时，其姓名和性别是必要的，其他的比如年龄、省份等一些信息是可选的，此时可以使用字典变长参数，即在函数定义时，在参数名称前面加上“**”。

```
#学生注册信息
def student(name,sex, ** others): #others 前面 **,表明它可以接收多余的字典参数
    dic = {}
    dic["name"] = name
    dic["sex"] = sex
    for k in others.keys():
        dic[k] = others[k]
```

```
    return dic
students = []
st1 = student("zhangsan","Male")#没有多余的参数,所以此时 others 为空
st2 = student("lili","Female",age = 18,province = "jiangsu")#
students.append(st1)
students.append(st2)
for iterm in students:
    print(iterm)
```

实例 5.11 中定义了一个 student 函数,该函数有 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,因为其前面有 ** ,所以它是一个字典变长参数,它将可以接收匹配完 name 和 sex 之后的所有的格式如“关键字=值”的参数。在第 10 行 st1=student("zhangsan","Male")中,函数调用时只有两个实参,将"zhangsan"传递给形参 name,将"Male"传递给 sex,那么 others 就什么都没有接收到。在第 11 行 st2=student("lili","Female",age=18,province="jiangsu")中,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"将其构成一个字典传递给 others。

运行结果:

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

当然在函数定义时可以既包括元组变长参数,也包括字典变长参数,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 代表一个元组变长参数,** kwargs 代表一个字典变长参数。需要注意的是,在函数定义的时候,必须是普通参数在前,元组变长参数在后,字典变长参数在元组变长参数的后面。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、元组变长参数、字典变长参数这样的顺序来定义函数。

在函数调用的时候,实参会优先匹配普通参数,如果二者个数相同,那么元组变长参数将获得空的元组和字典;如果实参的个数大于传统参数的个数,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】 元组变长参数和字典变长参数综合示例。

```
def exmaple(a, * args, ** kwargs):
    print("a: ",a)
    print("args: ",args)
```

```
    print("kwargs: ",kwargs)
example(1,2,3,4,5,6,7,8,name = 'Python',age = 30,)   #有元组变长参数,也有字典变长
#参数
print("**********")
example(4,5,6)            #有元组变长参数,没有字典变长参数
print("**********")
example(2,name = "Lili")  #有字典变长参数,没有元组变长参数
print("**********")
example(3)                #只有普通参数,没有元组变长参数,也没有字典变长参数
```

实例 5.12 定义了一个包含一个普通参数、一个元组变长参数、一个字典变长参数的函数,函数的功能比较简单,将这 3 部分的内容输出。函数调用时参数传递的具体细节如图 5.5 所示。

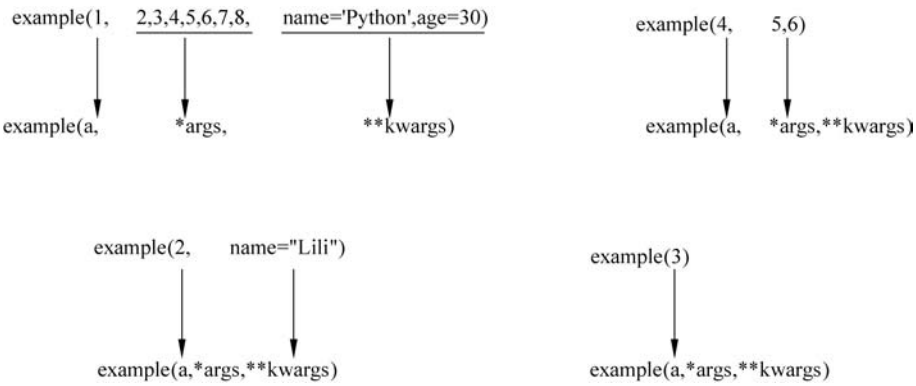


图 5.5　可变参数传递示意图

运行结果:

```
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: {}
```

在定义函数时形参的顺序为:普通形参、默认参数、元组变长参数、字典变长参数,例如 def func_name(a,b,c=3, * args, ** kwargs)。

函数调用时实参由位置实参和关键字实参组成,并且位置实参在前,关键字实参在后。实参向形参传递时应遵循以下基本规则:

(1) 与有无默认值无关,位置实参永远按位置传递给 * args 或 ** kwargs 之前对应的形参。

(2) 多余的位置实参传入 * args。

(3) 关键字实参则匹配剩下的普通形参。

(4) 多余的关键字实参则传入 ** kwargs。

当没有 * args 时,位置实参不能多于限定位置形参和普通形参的总量;当没有 ** kwargs 时,关键字参数必须在普通形参和限定关键字形参中存在;除 * args 和 ** kwargs 外,所有没有默认值的形参都必须匹配到值。同一形参不能被匹配两次。

视频讲解

5.4 lambda 函数

如果有一个函数,在程序中只被调用一次,那么可以使用 lambda 函数,lambda 函数是匿名函数,即函数没有具体的名称,而用 def 创建的方法是有名称的。其语法如下:

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

lambda 是匿名函数的关键字,冒号前面是参数,是可选的,如果没有参数,则 lambda 冒号前面就没有。冒号后边是匿名函数的函数表达式,注意表达式只能占用一行。

【实例 5.13】 lambda 函数。

```
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  #还可以这样使用:
4  print((lambda a,b: a + b)(3,4))
```

运行结果:

```
7
7
```

“lambda a,b: a+b”这个匿名函数的形参是 a 和 b,表达式 a+b 是函数的返回值。注意,lambda 函数需要定义的同时调用该函数,而不能采用普通函数那样先定义、然后再调用的使用方式。

5.5 变量的作用域

变量的作用域指变量起作用的范围,即能够在多大范围能够访问到它。

【实例 5.14】 变量作用域示例。

```
def my_func():
    a = 10
    print("a: {}".format(a))
    print("b: {}".format(b))
b = 20
my_func()
print("b: {}".format(b))
print("a: {}".format(a))
```

运行结果：

```
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 5.14 中，myfunc 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 myfunc 函数中可以访问函数内部定义的 a，也可以访问在函数外部定义的 b。在函数外部可以访问 b，但是不能访问 myfunc 函数中定义的变量 a。因为 myfunc 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，在整个文件都是可以访问的。

5.5.1 局部变量


视频讲解

局部变量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 5.15】 局部变量示例一。

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

运行结果：

```
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它所在的函数,即只能在此函数中访问。下面稍微修改一下这个例子。

【实例 5.16】 局部变量示例二。

```
def func1(x,y):
    x1 = x
    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)
```

在此例中,func1 函数中调用了函数 func2,但是局部变量的作用域并没有改变,即 func1 中定义的局部变量 x1 和 y1 的作用域仍然在 func1 中,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
```

视频讲解

5.5.2 全局变量

全局变量指的是在函数外部定义的变量,其作用域是整个程序,即在该程序中的所有函数都可以访问全局变量。

【实例 5.17】 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,函数外部定义了一个全局变量 z,在函数 func1 和 func2 中都可以访问此变量。

运行结果：

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

【实例 5.18】 局部变量与全局变量同名示例。

```
1   z = 100                 #全局变量
2   def func1(x,y):
3       x1 = x
4       y1 = y
5       z = 50              #同名局部变量
6       print("in func1, x1: {},y1: {},x: {},y: {},z: {}".format(x1,y1,x,y,z))
7   def func2():
8       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：

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

实例 5.18 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 z 实际上访问的是局部变量 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的，只能在该函数中访问使用它，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这个名称时，访问的是局部变量。无论在函数内怎样改动这个变量的值，只有在函数内生效，对全局来说是没有任何影响的。这也可以侧面说明函数内定义的局部变量优先级高于全局变量。

5.5.3 global 关键字


视频讲解

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，使用 global 关键字就是告诉 Python 编译器这个变量不是局部变量而是全局变量。

【实例 5.19】 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：

```
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 5.19 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 num1 的值，发现在函数外部 num1 的值并没有发生改变。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都不会影响同名的全局变量。如果想要在函数内部修改全局变量的值，可以用关键字 global 进行声明。下面修改这个程序。

【实例 5.20】 global 声明全局变量示例二。

```
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到的 num1 是
                             # 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 5.20 的 func1 中使用 global 声明 num1，那么在该函数内部访问的 num1 就是全局变量 num1，所以对它的修改也就是对全局变量的修改。

运行结果：

```
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 5.21】 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 5.21 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，可在函数内部用关键字 global 声明一下。读者可以自行修改这段代码，使之能够正确运行。


视频讲解

5.6 递归函数

递归，是一种可以根据其自身来定义问题的编程技术，递归通过将问题逐步分解为与原始问题类似但规模更小的子问题来解决，即将一个复杂问题简化并最终转化为简单问题，而简单问题的逐一解决，就反过来解决了整个问题。

考虑阶乘这个例子。n!＝n＊(n－1)＊(n－2)…＊2＊1。还可以改写为：

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

【实例 5.22】 下面采用递归函数的形式实现阶乘的计算。

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

运行结果：

```
4 != 24
```

构成递归需具备以下两个条件：

(1) 子问题须与原始问题为同样的问题，且更为简单。例如，n 的阶乘可以转化为求 n－1 的阶乘。

(2) 不能无限制的调用本身，必须有个出口，化简为非递归状况处理。例如，当 n＝1 时，n 的阶乘等于 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；递归函数可以将复杂函数修改为更简单的函数。

但递归也有它的劣势，因为它要进行多层函数调用，会消耗较大的堆栈空间和函数调用时间。Python 在调用深度达到 1000 后会停止函数调用。运行 print(fac(1000))则会出现以下错误提示：

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

【实例 5.23】 利用递归实现文件目录的显示。

```
import os
def print_files(path):
    lsdir = os.listdir(path)
    dirs = [i for i in lsdir if os.path.isdir(os.path.join(path, i))]
    if dirs:
        for i in dirs:
            print_files(os.path.join(path, i))
    files = [i for i in lsdir if os.path.isfile(os.path.join(path,i))]
    for f in files:
       print (os.path.join(path, f))
path = "C:/Python 教材"
print_files(path)
```

上面的例子中使用到了 os 库,所以需要导入该库: import os。

os.listdir(path)的功能是列出参数 path 所指定的目录下所有文件,返回值是一个 list,将其下的所有文件和子目录的名字以字符串的形式放入一个列表中。

os.path.isdir(path)判断参数指定的是否是目录。

os.path.isfile(path)判断是否是文件。

os.path.join(path1,path2)用于连接路径 path1 和路径 path2。

5.7 综 合 例 子

【实例 5.24】 蒙特卡洛方法计算 π。其基本思想是:边长为 1 的正方形内有一内切圆,随机扔一点在圆内的概率为 π/4。让系统随机生成 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。统计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)       #计算该坐标距离圆心的距离
        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):    #比较在有 10 个点、100 个点、100 个点、1000 个点情况下 pi 的值
    n = m * pow(10,i)
    PI = calPI(n)
    print("{} points PI: {}".format(n,PI))
```

运行结果：

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

【实例 5.25】 小学生计算器。利用函数实现能够求 100 以内的加减法、10 以内的乘除法。

```
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 5.26】 利用函数实现将 2～20 的所有素数输出。

```
import math
def prime(m):      # 判断 m 是否是素数,如果是返回 1,不是返回 0
    i = 2
    while (i <= math.sqrt(m)):
        if m % i == 0:
            return 0
        i = i + 1
```

```
    return 1
print("2～20 的素数有: ")
for m in range(2,20):
    if(prime(m) == 1):
        print(m," ",end = '')
```

运行结果:

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

【实例 5.27】 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,$153 = 1^3 + 5^3 + 3^3$,因此 153 就是一个水仙花数。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("3 位数的水仙花数有: ")
for i in range(100,1000):
    if Narcissistic(i) == 1:
        print(i)
```

运行结果:

```
3 位数的水仙花数有:
153
370
371
407
```

【实例 5.28】 猜数字。

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

案例分析:根据案例需要实现的功能,可以将任务分解为两个子任务:产生数字和用户猜数字,并用两个函数实现。

代码如下:

```
import random
def main():
    #1.系统生成一个随机数并放到 number 中
    number = newNumber()
```

```
    #2.让用户猜测 number 的值到底是几
    guessNumber(number)
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()
```

这里定义了两个函数 newNumber()和 guessNumber(),其中 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()函数的功能是让用户输入一个整数,比较用户输入的数字和参数的大小,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。另外大家可能会注意到,这里用到了 main 函数,关于 main 函数大家可以查找相关资料进行了解。

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(),该函数用一个循环来调用 guessNumber()函数,循环次数为 10。程序代码如下:

```
import random
def main():
    #1.系统生成一个随机数并放到 number 中
    number = newNumber()
    #2.让用户猜测 number 的值到底是几
    guessTime(number)
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()
```

在这个程序中,我们又增加了一个函数 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,所以还需要对程序继续修改,实现当用户猜测正确的时候,跳出循环。改进的关键在于 guessNumber()函数,因为我们的基本思路是当猜测正确的时候能够退出循环,如果 guessNumber()函数中对于猜对猜错仅仅给出提示信息,不返回任何值给调用它的函数 guessTime(),就无法控制退出循环,因此应考虑修改 guessNumber()函数,如果猜对,除了输出提示信息外,还返回 1;如果猜错,提示太大或者太小,并且返回 0。这样可以在 guessTime()中根据 guessNumber()的返回值来判断是否退出循环。因此程序修改如下:

```
1  import random
2  def main():
3      #1.系统生成一个随机数并放到 number 中
4      number = newNumber()
5      #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
6      times = 10
7      guessTime(number,times)
8  def newNumber():
9      number = random.randint(1,101)
10     return number
11 def guessNumber(number):
12     yAnswer = int(input("enter a integer between 0 - 100: "))
13     if yAnswer > number:
14         print("too big")
15         return 0
16     elif yAnswer < number:
17         print("too small")
18         return 0
19     else:
20         print("right")
21         return 1
22 def guessTime(number,times):
23     for i in range(times):
24         if(guessNumber(number) == 1):
25             print("你一共猜了",i + 1,"次")
26             break
27     if(i > times):
28         print("sorry,你没有猜对,正确答案是",number)
29 if __name__ == '__main__':
30     main()
```

从上面的案例可以看出,为了实现猜数字的任务,我们将主任务分解为 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;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,这个函数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。接着是 guessNumber()让用户输入自己的猜测并判断是否正确,而我们的目标是让用户猜测多次,因此又建立了一个函数 guessTime(number,times),该函数调用 times 次 guessNumber(),从而实现让用户最多可以猜测 times 次,如果猜对则退出循环。

5.8 垃圾分类——共创美好家园


视频讲解

5.8.1 案例背景

随着人们生活水平的不断提高，对美好生活的向往也越发迫切，这不仅仅意味着物质上的富足，也包括优美宜居的生态环境和绿色文明的生活方式。但是随着经济的发展和消费水平大幅提高，我国垃圾产生量迅速增长，不仅造成资源浪费，也使环境隐患日益突出，成为经济社会持续健康发展的制约因素、人民群众反映强烈的突出问题。

垃圾分类关系着人民群众的生活环境，关系着人民群众的身体健康，关系着生态资源的节约，实现可持续发展。遵循减量化、资源化、无害化原则，实施垃圾分类处理，引导人们形成绿色发展方式和生活方式，可以有效改善城乡环境，促进资源回收利用，也有利于国民素质提升、社会文明进步。

“实行垃圾分类，关系广大人民群众生活环境，关系节约使用资源，也是社会文明水平的一个重要体现。”习近平总书记对垃圾分类工作做出重要指示，深刻指出垃圾分类的重要意义，明确提出推行垃圾分类的具体要求，为我们进一步做好垃圾分类工作指明了方向，对于动员全社会共同为推动绿色发展、建设美丽中国贡献智慧和力量，具有十分重要的意义。

垃圾分类不是易事，需要加强科学管理、形成长效机制、推动习惯养成。这几年，垃圾分类的顶层设计不断完善、推进力度持续加强，由点到面、逐步推开，成效初显。从 2020 年开始，全国地级及以上城市全面启动生活垃圾分类工作，垃圾分类取得积极进展。但也要看到，总体上，我国垃圾分类覆盖范围还很有限，垃圾分类收运和处置设施依然存在短板，群众对垃圾分类的思想认识仍有不足。进一步做好垃圾分类工作，就要按照习近平总书记的重要指示，加强引导、因地制宜、持续推进，把工作做细做实，持之以恒抓下去。

中国的生活垃圾分类回收还处于起步阶段，生活垃圾一般可分为四大类：可回收垃圾、厨余垃圾、有害垃圾和其他垃圾。目前存在着分类不细，缺乏相关的法律法规监督等问题，因此我们通过生活垃圾分类回收实现资源的再利用和可持续发展还有很长的路要走。

“不积跬步，无以至千里。”推动形成绿色发展方式和生活方式，是发展观的一场深刻革命。大家携起手来，共同把“垃圾分类”的事情办实做好，就能让良好生态环境成为人民幸福生活的增长点、成为经济社会持续健康发展的支撑点、成为展现我国良好形象的发力点，让中华大地天更蓝、山更绿、水更清、环境更优美。

5.8.2 案例任务

生活垃圾一般可分为四大类：可回收垃圾、厨余垃圾、有害垃圾和其他垃圾，对应 4 个不同颜色的垃圾桶。本案例的任务是根据用户所要丢弃的垃圾，告诉用户这是什么类型的垃圾，需要放入到哪个颜色的垃圾桶。

5.8.3 案例分析与实现

分析：因为有 4 种类型的垃圾，所以可以定义 3 个全局列表变量，用来存储可回收垃圾、厨余垃圾、有害垃圾，其他不在这 3 个列表中的就都属于其他垃圾。根据用户输入的垃

圾判断属于哪一类,然后执行对应的操作。

参考代码如下:

```
#垃圾分类
waste_recycle = ["paper","glasses","bottle","plastic"]        #可回收垃圾
waste_kitchen = ["peel","vegetable", "leftover", "flower"]    #厨余垃圾
waste_harmful = ["battery","light tube","daily chemical"]     #有害垃圾
def handle_harmful():
    print("This is harmful waste and please put it into the red trash can ")
def handle_kitchen():
    print("This is kitchen waste and please put it into the green trash can ")
def handle_recycle():
    print("This is recycle waste and please put it into the blue trash can ")
def handle_others():
    print("This is other waste and please put it into the yellow trash can ")
def do_with_waste(waste):
    if waste in waste_harmful:
        handle_harmful()
    elif waste in waste_kitchen:
        handle_kitchen()
    elif waste in waste_recycle:
        handle_recycle()
    else:
        handle_others()
waste = input("enter waste name:")
do_with_waste(waste)
```

5.8.4 总结和启示

随着互联网、人工智能、大数据等技术飞速发展,科技在我们日常生活中的作用越来越大。如何将其应用到垃圾分类中是我们重点研究的问题。在垃圾收集环节,基于人工智能的垃圾分类主要出现了监督分类型智能垃圾桶和自动分类垃圾桶。监督分类型智能垃圾桶通过机器视觉判断垃圾分类正确性,用人脸识别等功能将居民信用与垃圾分类行为挂钩,以此督促居民正确投放。自动分类垃圾桶则是用户只需投入垃圾,由智能垃圾桶通过传感器、摄像头、AI 图像识别算法来自动进行垃圾分类,然后通过垃圾桶内部特殊的机械结构将垃圾传输到不同的垃圾桶。

本案例只是垃圾分类的一个简单示例,大家可以结合人工智能领域中的机器视觉来实现基于图像的垃圾分类功能。

5.9 本章小结

本章主要介绍函数的定义和调用、函数的参数、变量的作用域、lambda 函数、递归函数。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、可变长参数以及关键字参数。通过具体的实例讲解,使得读者对其中的概念有了更为直观和深刻的理解。最后通过几个综合例子锻炼读者的综合编程能力。

5.10 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)，其功能是求整数 num 的第 k 位的值。

【训练 5.4】 编写递归函数 fibo(n)，其功能是求第 n 个斐波那契数列的值，进而实现将前 20 个斐波那契数列输出。

【训练 5.5】 编写一个函数 cacluate，可以接收任意多个数，返回的是一个元组。元组的第一个值为所有参数的平均值，第二个值是小于平均值的个数。

【训练 5.6】 模拟轮盘抽奖游戏。轮盘分为 3 部分：一等奖、二等奖和三等奖；轮盘转的时候是随机的，如果范围为[0,0.08)，代表一等奖；如果范围为[0.08,0.3)，代表二等奖；如果范围为[0.3，1.0)，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，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，并给出它们出现的次数。

【训练 5.8】 磅(lb)和千克(kg)的转换。利用函数实现磅和千克的转换。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成磅或者千克。

【训练 5.9】 一个数如果恰好等于它的真因子(即除了自身以外的约数)之和，这个数就称为“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，将用户所输入的字符串以相反的顺序输出。