第 5 章

CHAPTER 5

函数和模块

程序的函数是指将相关的代码块组织在一起，形成一个独立的、可重复使用的代码段。这个代码段可以被其他程序调用，以实现特定的功能或操作。

在编程语言中，函数通常被定义为一个具有特定名称和参数列表的代码块，并在需要时调用。函数可以接收输入参数，并返回一个结果。通过函数，可以将复杂的程序逻辑分解为更小的、易于管理的代码块，提高代码的可读性和可维护性。同时，函数也可以提高代码的复用性，避免重复编写相同的代码。

函数有许多好处，主要包括以下几点。

(1) 提高代码的可读性和可维护性：函数能将大块的代码划分为小块，每个函数只负责完成一个特定的任务，使代码更加清晰、易于理解和修改。

(2) 避免重复代码和提高复用性：如果有一段功能需要在多个地方使用，则只需在函数中实现一次，就可以在需要的地方调用该函数。这样能减少代码的重复性，提高代码的复用性。

(3) 便于修改和维护：通过将代码划分为多个函数，可以方便地对某一部分代码进行修改或扩展，而不会影响到整个程序。同时，由于函数具有独立性，可以单独测试和调试，进一步提高了代码的可维护性。

(4) 实现代码模块化：通过将代码分解为多个函数，可以使代码更加模块化，各个模块之间的耦合度降低，有利于代码的管理和维护。高内聚、低耦合。

(5) 有利于社会分工：大型项目可以进行分解，每个人只编写属于自己的函数，编写好的函数也可以被其他程序员使用，从而提高了工作效率。

因此，使用 Python 函数可以使代码更加清晰、易于理解、可维护、可重用，并且有利于代码的管理、维护和社会分工。

在 Python 语言中，已经内置了许多系统自带的函数，如输出函数 print()、输入函数 input()等，只是在使用时调用它们，不用关心具体的实现细节。也可以定义自己的函数以实现具体的业务功能。


5min

5.1 函数的定义

定义函数使用关键字 def，后跟函数名与括号内的形参列表。函数语句从下一行开始，并且必须缩进。

Python 函数的基本格式如下：

```
def 函数名称( 参数 1, 参数 2, …… ):
    """文档字符串"""
    函数体
    return 值
```

其中，

(1) def 是 Python 的关键字，用于定义函数。这一行的最后以英文冒号(:)结束。

(2) 函数名称，用户自己定义的名称，这个名字最好能够体现函数的功能。

(3) (参数 1, 参数 2,……)，函数的参数，可以定义一个或多个参数。参数是可选的。

(4) """文档字符串"""，函数的文档字符串(docstring)，它是可选的，用于描述函数的作用和用法。另外也可以自动生成函数的说明文档。

(5) 函数体，包含了实现函数功能的 Python 代码。它们相对于函数的名称要缩进 4 个英文字符。

(6) 函数的返回值。这是可选的。如果函数执行完毕后不需要返回任何值，则可以省略这一行。

函数作为一个语句块，不会直接自动执行，只能通过函数的名称调用里面的语句块，有参数的函数要传入对应的参数值。


6min

5.2 函数的实现

实现基本的函数，代码如下：

```
def show():                              #第 1 行
    print("这是一个简单的函数例子") #第 2 行

show()                                   #第 4 行
```

这段代码定义并调用了一个名为 show 的函数，该函数的功能是打印出“这是一个简单的函数例子”。

上述代码的含义如下。

(1) def show()这一行定义了一个名为 show 的函数。def 是 Python 中用于定义函数的关键字。

(2) print("这是一个简单的函数例子") 这一行是 show 函数的内容。当这个函数被调

用时，它会执行这一行代码并输出“这是一个简单的函数例子”。

(3) show()这一行调用了上面定义的 show 函数。当运行这段代码时，它会输出“这是一个简单的函数例子”。

函数的执行过程如下：

第 1 步，执行第 4 行，通过函数的名称来调用第 1 行的函数。

第 2 步，执行第 1 行，在第 1 行中，如果有参数，则接收传过来的参数的值。

第 3 步，调用第 2 行，执行函数里面的内容。在屏幕上输出：这是一个简单的函数例子。

【示例 5-1】 函数在主程序中可以被多次调用，代码如下：

```
#第 5 章 5.1 函数的调用
def say_hello():
    '''
    函数的功能为输出两行 hello
    '''
    for i in range(1,3):
        print("hello")

print("start")
say_hello()
print("…")
say_hello()
print("end")

#输出的结果如下
start
hello
hello
…
hello
hello
end
```

这段代码首先定义了一个名为 say_hello 的函数，该函数的功能是打印两行 hello，因此，主程序首先输出字符串 start，然后调用函数输出两行 hello。输出字符串“…”，再次调用函数输出两行 hello。最后，输出字符串 end。

也可以定义带有参数的函数，在进行调用时，要注意调用的函数中要有对应的变量或值。

【示例 5-2】 输入书本的数量和价格作为参数，通过函数的调用得到总的钱数，代码如下：

```
#第 5 章 5.2 输入书本的数量和价格,计算总钱数
def calculate_total_price(num_books, price_per_book):      #第 1 行
    """
    计算总钱数
    """
```

```
    total_price = num_books * price_per_book          #第5行
    print(f"书的数量为{num_books}本,每本书的价格为{price_per_book}元,"
    "总钱数为{total_price}元。")                        #第7行

nums = int(input("请输入书的数量(单位本):"))          #第9行
price = float(input("请输入每本书的价格(单位元):"))  #第10行
calculate_total_price(nums,price)                     #第11行

#输出的结果如下
请输入书的数量(单位本):100
请输入每本书的价格(单位元):59.5
书的数量为100本,每本书的价格为59.5元,总钱数为5950.0元。
```

程序的执行顺序如下：

第1步，执行第9行，得到书的数量，赋值给变量nums。

第2步，执行第10行，得到书的价格，赋值给变量price。

第3步，执行第11行，在第11行通过名称调用第1行的calculate_total_price()函数。

第4步，执行第1行，其中，第11行的nums的值传递给参数num_books，price的值传递给参数price_per_book。

第5步，执行函数体中的语句，包括第5行到第7行。通过函数计算总价，即书本数量乘以每本书的价格，并返回结果。在这个示例中，我们假设每本书的价格为59.5元，购买了100本书，因此总价为5950.0元。

【示例5-3】 对任意两个整数求和并返回结果，实现了带有参数并且有返回值的函数调用，代码如下：

```
#第5章 5.3 对任意两个整数求和并返回结果
def add_numbers(num1, num2):                          #第1行
    """
    对两个整数求和,并返回结果
    """
    sum_of_numbers = num1 + num2                      #第5行
    return sum_of_numbers                             #第6行

m = int(input("请输入一个整数:"))                     #第8行
n = int(input("请输入一个整数:"))                     #第9行
#调用函数并打印结果
result = add_numbers(m, n)                            #第11行
print(result)

#输出的结果如下
请输入一个整数:3
请输入一个整数:4
7
```

它的执行顺序如图5-1所示。

第1步，执行第8行和第9行，程序代码实现提示用户输入数字，并分别赋值给变量m

和 n。

第 2 步，执行第 11 行，调用第 1 行的 add_numbers()函数，分别将变量 m 的值传给参数 num1，将变量 n 的值传给参数 num2。

第 3 步，执行第 1～6 行，运行 add_numbers ()函数，在函数内部，将这两个参数的值相加，并将结果赋值给变量 sum_of_numbers，并返回这个结果。

第 4 步，当 add_numbers()函数执行完成后，将变量 sum_of_numbers 的值赋值给变量 result。

第 5 步，打印输出 result 的值。

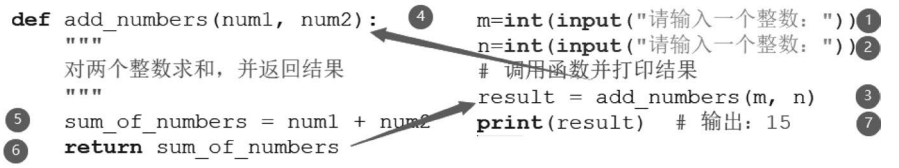

图 5-1　函数调用的执行顺序

例如，如果用户输入的数字分别是 3 和 4，程序则会输出 7。这意味着程序将用户输入的两个数字相加，得到结果 7，然后通过 add_numbers 函数返回这个结果。主程序变量 result 接收了函数关键字 return 后变量 sum_of_numbers 返回的值。

【示例 5-4】 使用有返回值的函数，对房屋租赁行业中仓储成本进行计算，代码如下：

```
#第 5 章 5.4 计算仓储成本
def calculate_storage_cost(area, rent):
    '''
    根据仓库面积和租金计算存储成本
    '''
    cost = area * rent
    return cost

area = 500          #仓库面积(平方米)
rent = 10           #租金(元/平方米/月)
result = calculate_storage_cost(area, rent)
print(result)       #输出存储成本(元/月)

#输出的结果如下
5000
```

在上述程序中，已知仓库面积和租金，通过调用函数，就可以计算出仓储成本。

在实际业务中，不同区域的仓库面积和租金成本可能不一样，需要进行处理和计算。可以将计算仓储成本的逻辑封装在一个函数中，并在需要的地方调用它以获取结果。

斐波那契数列(Fibonacci Sequence)，又称黄金分割数列，因意大利数学家莱昂纳多·斐波那契(Leonardo Fibonacci)以兔子繁殖为例子而引入，故又称“兔子数列”，其数值第 1 项和第 2 项的值为 0 和 1，第 3 项的值为前两项之和。数列的值依次为 0、1、1、2、3、5、8、13、

21、34…。

【示例 5-5】 函数名为 fib，接收一个参数 n，然后打印出 Fibonacci 数列中小于 n 的所有数，代码如下：

```
#第 5 章 5.5 斐波那契数列的输出
def fib(n):
    """打印到 n 的斐波那契数列"""
    a, b = 0, 1
    while a < n:
        print(a, end = ' ')
        a, b = b, a + b
    print()

fib(100)

#输出的结果如下
0 1 1 2 3 5 8 13 21 34 55 89
```

在函数内部，首先定义了两个变量 a 和 b，分别初始化为 0 和 1。这两个变量用于存储 Fibonacci 数列中的连续两个数字，然后有一个 while 循环，条件是 a 小于 n。在循环体内，首先打印出 a 的值，然后更新 a 和 b 的值。a 的新值是 b，而 b 的新值是 a 和 b 的和。这样，每次循环迭代都会打印出 Fibonacci 数列中的下一个数字。当循环结束时会打印一个换行符，这样如果调用 fib(100)，则将会在控制台上打印出 Fibonacci 数列中所有小于 100 的数字，每个数字一行。

【示例 5-6】 在医院的住院过程中，治疗方式可能有手术或吃药。在计算费用时会考虑住院天数和是否有保险，在下面的示例中模拟了在医院通过调用函数来计算医疗费用，代码如下：

```
#第 5 章 5.6 模拟医院通过调用函数来计算医疗费用
def calculate_medical_cost(treatment, days, is_insured):
    """
计算一个病人的医疗费用
Treatment: 治疗方式; days: 住院天数; is_insured: 是否有保险
    """
    #根据治疗方式、住院天数和是否有保险计算医疗费用
    if treatment == "surgery":          #外科手术
        cost = 1000
    elif treatment == "medicine":       #吃药
        cost = 500
    else:
        cost = 300
    if days > 7:
        cost += days * 110 + (days - 7) * 100
    else:
        cost += days * 110
    if is_insured:
```

```
        cost *= 0.8
    return cost

treatment = "surgery"                    #治疗方式
days = 8                                 #住院天数
is_insured = True                        #是否有保险
result = calculate_medical_cost(treatment, days, is_insured)
print(f"治疗的总费用为{result}元")#输出医疗费用(元)

#输出的结果如下
治疗的总费用为1584.0元
```

在上面的示例中，首先传递3个参数，然后通过判断这3个参数的类型来计算出住院的总费用。实际的住院费用更加复杂，但计算过程是类似的。

【示例5-7】 通过函数的调用实现求圆的面积，代码如下：

```
#第5章 5.7 求圆的面积
def calculate_circle_area(radius):
    """
        已知半径,求圆的面积
    """
    pi = 3.14159
    area = pi * radius ** 2
    return area

radius = 5
result = calculate_circle_area(radius)
print(result)    #输出 78.53975
```

在上面的示例中，传入参数半径，通过调用函数得出圆的面积。

5.3　函数中变量的作用域

在Python语言中，函数中的变量有作用域，它们的使用范围不同。函数中的变量可以分为全局变量和局部变量。

全局变量是在函数之外定义的变量，它们可以在整个程序中访问，包括在函数内部。当在函数内部需要使用全局变量时，需要在函数内部使用global关键字声明。局部变量是在函数内部定义的变量，它们只能在该函数内部访问。

全局变量常常被用来存储配置参数，这些参数可能在程序的许多地方需要使用。全局变量也可以在不同函数或方法之间共享状态。这在一些需要跨多个函数跟踪状态的情况下很有用，如用户会话、计数器等。

局部变量在函数内部进行临时计算时使用，这些变量只在函数执行期间存在，函数返回后就会被销毁，这样可以避免不必要的内存使用。另外局部变量是实现封装和数据隐藏的重要手段。通过将数据限制在函数内部，只通过函数的接口(输入和输出)与外界交互，可以

隐藏实现细节，提高代码的可维护性和安全性。

【示例 5-8】 局部变量只能在函数的内部使用，代码如下：

```
#第 5 章 5.8 局部变量的使用
#定义另一个函数,在函数内部定义局部变量 y
def my_function():
    #在函数内部定义局部变量 y
    y = 20
    print(y)


#调用函数 another_function,输出局部变量 y 的值
another_function()    #输出:20
#print(y)             #出错:执行时,报 NameError: name 'y' is not defined 异常
```

在上面的代码中，在函数 my_function 中定义了一个变量 y，这个变量只能在函数的内部使用，当进行函数调用时，在函数内部打印输出 y 的值。如果在函数外部调用，则会出错。

【示例 5-9】 全局变量和局部变量的区别，代码如下：

```
#第 5 章 5.9 全局变量和局部变量的区别
def fun(x):
    global y          #全局变量 y
    print(x,y)        #输出: 100 200
    x = 3
    y = 4
    print(x,y)        #输出: 3 4

x = 100
y = 200
fun(x)                #将外边变量 x 的值 100 传入函数中
print(x,y)            #输出: 100 4

#输出的结果如下
100 200
3 4
100 4
```

程序执行的顺序如下：

(1) 主程序中的变量 x 被赋值为 100，变量 y 被赋值为 200。

(2) 当调用函数 fun()时，函数内的局部变量 x 接收主程序变量 x 的值为 100。由于变量 y 是全局变量，与主程序中的 y 是同一个变量，因此第 1 次输出为 100 200。

(3) 在函数内部，局部变量 x 的值变为 3，y 的值变为 4，因此第 2 次输出为 3　4。

(4) 当函数执行完毕，在执行最后一行程序代码时，由于主程序变量 x 的值为 100，变量 y 是全局变量，函数内部值的修改会影响外部变量的值，所以输出的结果为 100　4。

5.4　函数之间的调用

11min

在 Python 语言中，一个函数可以调用其他函数，函数之间可以通过调用关系相互关联，从而实现代码的模块化和可重用性，如图 5-2 所示。

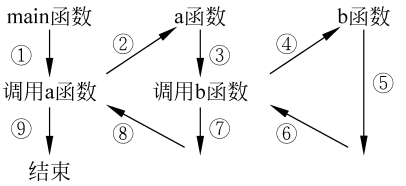

图 5-2　函数之间的调用

在图 5-2 中，主程序调用 a 函数，a 函数调用 b 函数，b 函数执行完成后，依次按返回顺序执行，最后执行主程序剩下的程序代码。

【示例 5-10】 一个电信业务示例，展示如何在一个函数中调用另一个函数，代码如下：

```
#第 5 章 5.10 函数之间的调用
#定义一个函数,用于计算电话费
def calculate_fee(duration, rate):
    return duration * rate

#定义一个函数,用于计算套餐费
def calculate_package_fee(package_type, duration):
    if package_type == 'basic':
        return duration * 0.5
    elif package_type == 'VIP':
        return duration * 0.1
    else:
        return duration

#定义一个函数,用于计算总费用
def calculate_total_fee(phone_number, package_type, duration):
    phone_fee = calculate_fee(duration, 0.1)                        #计算电话费
    package_fee = calculate_package_fee(package_type, duration)     #计算套餐费
    total_fee = phone_fee + package_fee                             #计算总费用
    return total_fee

#调用 calculate_total_fee 函数
phone_number = '1234567890'                                         #电话号码
package_type = 'basic'                                              #套餐类型
duration = 300                                                      #通话时长为 300min
total_fee = calculate_total_fee(phone_number, package_type, duration)
print("总费用为:", total_fee)

#输出的结果如下
总费用为: 180.0
```

在这个示例中，定义了 3 个函数：calculate_fee、calculate_package_fee 和 calculate_total_fee。calculate_fee 函数用于计算电话费，它接受通话时长和费率作为参数，并返回通话费用。calculate_package_fee 函数用于计算套餐费，它接受套餐类型和通话时长作为参数，并返回套餐费用。calculate_total_fee 函数用于计算总费用，它接受电话号码、套餐类型和通话时长作为参数，并调用 calculate_fee 和 calculate_package_fee 函数来计算电话费和套餐费，并将它们相加，从而得到总费用。最后，我们调用 calculate_total_fee 函数来计算总费用，并将结果打印出来。

在这个例子中，通过调用不同的函数来计算电话费、套餐费和总费用，从而实现了电信的计费业务。


4min

5.5 默认值参数

为参数指定默认值为非常有用的方式。在调用函数时，有些参数可以使用默认值，而不用传入对应的值。

【示例 5-11】 定义一个函数，传入姓名、年龄和总费用共 3 个参数，并输出相关信息，代码如下：

```
#第 5 章 5.11 函数中默认值参数的使用
def fun(name,age = 18,cost = 100):
    print(f"姓名为{name},年龄为{age},总费用为{cost}")

fun("Tom")              #第 1 次调用函数 fun,参数 name 接收值
fun("Jack",23)          #第 2 次调用函数 fun,参数 name、age 接收值
fun("Mike",17,9999)     #第 3 次调用函数 fun,参数 name、age、cost 接收值

#输出的结果如下
姓名为 Tom,年龄为 18,总费用为 100
姓名为 Jack,年龄为 23,总费用为 100
姓名为 Mike,年龄为 17,总费用为 9999
```

在上面的示例中，定义了一个函数 fun，里面有 3 个参数，其中 name 为必选参数，age 和 cost 为可选参数。实现的功能为打印输出这 3 个变量的信息。在第 1 次调用时，只是对 name 变量传入值 Tom。在第 2 次调用时，对变量 name 和 age 进行赋值。第 3 次调用时，则对这 3 个变量进行了传值。

由于变量 age 和变量 cost 已经有了默认值，所以在调用函数 fun 传入参数时，可以忽略这两个参数的调用。变量 name 由于没有默认值，所以必须放在第 1 个位置，并需要传入相应的值。

对于可选择参数，也可以对指定参数进行传值，而忽略其他参数。如执行以下语句：

```
fun("李明",cost = 500)
```

对 name 参数和 cost 参数指定了具体的值，而忽略了其他参数。输出的结果如下：

```
姓名为李明,年龄为 18,总费用为 500
```

5.6　接收未知数量的参数

在 Python 中，使用 * args 和 ** kwargs 来在函数定义中接收任意数量的位置参数和关键字参数。它们适用于参数的个数未知。

（1）* args：* args 是一个用于接收任意数量的位置参数的语法。这些参数会被收集为一个元组，可以在函数内部进行迭代或者直接使用。

【示例 5-12】 以元组形式接收未知数量的参数，代码如下：

```
#第 5 章 5.12 以元组形式接收未知数量的参数
def print_numbers( * args):
    for num in args:
        print(num,end = " ")

print_numbers(1, 2, 3, 4, 5)

#输出的结果如下
1 2 3 4 5
```

在这个例子中，* args 接收了 5 个参数，以元组的形式保存，并在函数内部通过循环打印出每个参数的值。

（2）** kwargs：** kwargs 是一个用于接收任意数量的关键字参数的语法。这些参数会被收集为一个字典，可以在函数内部通过键访问。

【示例 5-13】 以字典形式接收未知数量的参数，代码如下：

```
#第 5 章 5.13 以字典形式接收未知数量的参数
def print_words( ** kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

print_words(apple = "red", banana = "yellow", cherry = "red")

#输出的结果如下
apple: red
banana: yellow
cherry: red
```

在这个例子中，通过调用 print_words 函数传递了 3 个关键字参数（apple＝"red"，banana＝"yellow"，cherry＝"red"），其中参数 ** kwargs 接收了这 3 个键-值对，并以字典的形式保存。在函数的内部通过循环打印输出每个键-值对。


7min

5.7 递归函数

递归调用是一种特殊的嵌套调用，是某个函数调用自己或者调用其他函数后再次调用自己。递归函数必须有一个结束条件，否则会导致无限递归，从而大量占用计算机资源，直至程序崩溃。

在 Python 中，递归函数的实现方式如下：

(1) 定义函数，在函数内部调用自身。

(2) 定义递归的结束条件，当满足该条件时，函数不再递归调用，而是直接返回结果。

(3) 在函数内部，将问题分解为更小的子问题，然后递归调用自身来解决这些子问题。

(4) 将子问题的结果合并，得到原问题的解。

【示例 5-14】 使用递归函数来计算阶乘，代码如下：

```
#第 5 章 5.14 递归函数的使用
def factorial(n):
    '''
    求 n 的阶乘，通项公式为 n!= 1×2×3×…×n
    '''
    if n == 1:                          #递归结束条件
        return 1
    else:
        return n * factorial(n-1)       #递归调用

print(factorial(5))                     #输出:120
```

在这个例子中，函数 factorial 接受一个数字 n 作为参数，并返回 n 的阶乘。如果 n 等于 1，则函数返回 1，否则函数返回 n 乘以 factorial(n－1)的结果。这个调用 factorial(n－1)的过程会一直递归下去，直到 n 等于 1 为止。

例如，如果我们调用 factorial(5)，则需要依次计算出 factorial(4)、factorial(3)、factorial(2)和 factorial(1)的值，并将它们的结果相乘。在计算 factorial(1)时，函数会返回 1，然后依次向上计算出 factorial(2)、factorial(3)、factorial(4)的值。最终 factorial(5)的结果为 120。


3min

5.8 lambda 表达式

在 Python 语言中，lambda 是一个非常有用的特性，它用于创建匿名函数，也就是没有明确名称的函数。lambda 函数通常用于短小的函数定义，而不需要使用 def 关键字创建一个正式的函数。对于只有一行语句的函数，用 lambda 表达式可能更为直观。

【示例 5-15】 定义一个函数，实现对 3 个数求和，代码如下：

```
#第 5 章 5.15 一般函数的使用
def add1(a,b,c):
    '''对 3 个数求和'''
    return a + b + c

result1 = add1(1,2,3)
print(result1)

#输出的结果如下
6
```

上面代码定义了一个名为 add1 的函数，传入了 3 个参数 a、b、c，并对它们求和，通过调用这个函数，得到了这 3 个数的和，并打印输出。

类似地，可以用 lambda 表达式来表示：

```
#lambda 表达式的实现
add2 = lambda x,y,z:x + y + z
print(add2(2,3,4))

#输出的结果如下
9
```

在上面的这段代码中，函数的名称为 add2，传入的 3 个参数分别为 x、y、z，然后求和。

【示例 5-16】 使用 lambda 作为另一个函数的参数，代码如下：

```
#第 5 章 5.16 使用 lambda 作为另一个函数的参数
def apply_func(func, value):                #第 1 行
    return func(value)                      #第 2 行

result = apply_func(lambda x: x ** 2, 4)    #第 4 行
print(result) #输出:16                      #第 5 行
```

在上面的示例中，apply_func 有两个参数 func 和 value。当执行到第 4 行时，数字 4 将被赋值给 apply_func 函数中的参数 value，而 func 则是一个 lambda 表达式。也就是 value 的值将被传给变量 x，计算 x 的平方，最后的结果为 16。

【示例 5-17】 把匿名函数用作传递的实参，实现对列表的排序，代码如下：

```
pairs = [ (2, 'two'), (3, 'three'), (1, 'one'), (4, 'four')]
pairs.sort(key = lambda pair: pair[1])
print(pairs)
#输出的结果如下
[(4, 'four'), (1, 'one'), (3, 'three'), (2, 'two')]
```

这段 Python 代码实现了对一个元组列表进行排序。

pairs 是一个列表，包含 4 个元组：(2, 'two')，(3, 'three')，(1, 'one')和(4, 'four')。

pairs.sort(key＝lambda pair：pair[1])这行代码会对这个列表进行排序。这里的 key

参数是一个函数，它决定按哪种方式进行排序。在这个例子中，lambda pair：pair[1] 是一个匿名函数，它接受一个元组作为输入（在这里是 pair），然后返回这个元组的第 2 个元素（pair[1]），因此，这行代码会对列表中的元组按照每个元组的第 2 个元素（也就是字符串的字母顺序）进行排序。

最后，print(pairs) 这行代码会打印出排序后的列表。


8min

5.9 模块的使用

在 Python 中，模块（module）是一个包含 Python 代码的文件，通常以.py 结尾。模块可以让我们将相关的函数、类和变量组织在一起。一旦编写了一个模块，就可以从解释器或其他模块加载它。在不同的模块中定义不同的功能，这有助于隔离关注点，使代码更加清晰和易于理解，以及更加灵活和易于扩展。每个模块可以由不同的开发人员开发和测试，这也有助于提高代码的质量和开发效率。通过模块，将代码分解为更小的、可管理的部分，并且可以在需要的地方重复使用这些代码，使代码更加简洁、易于阅读和维护，提高了代码的可维护性和重用性。

例如有一个 calc.py 文件，里面包含 4 个函数，代码如下：

```
#第 5 章 5.17 calc.py 文件，包含 4 个函数
#实现两个数相加
def add(a,b):
    return a + b

#实现两个数相减
def substract(a,b):
    return a - b

#实现两个数相乘
def multiply(a,b):
    return a * b

#实现两个数相除
def divide(a,b):
    return a/b
```

可以在同一个位置新建另一个 Python 文件，在这个文件中可以调用上述文件中的函数，代码如下：

```
import calc
print(calc.add(3,5)) 输出:8
```

上述代码的执行流程和含义如下：

（1）使用 import 关键字导入名为"calc"的模块，其中 calc 为另一个 Python 文件的文件名，里面包含了要调用的函数。

(2) 通过 calc 前缀加点的方式调用 calc 模块中的 add 函数,并将 3 和 5 作为参数传递给该函数。在函数内部实现了加法运算,通过输出语句打印 add 函数的返回值。

这允许编写一段代码,然后可以在任何需要的地方调用它。当处理较大的项目并希望在不同的模块之间分离业务问题时,这很有帮助。

也可以为这个模块加一个别名,通过别名来调用模块中的内容,代码如下:

```
import calc as ca    #ca 为模块 calc 的别名
print(ca.add(3,5)) 输出:8
```

这种调用方式的缺点在于,通过关键字 import 会导入这个模块的所有内容,并且每次调用时必须在调用函数的前面加上模块的名称。

可以采用另一种导入方式来调用模块中的部分函数,代码如下:

```
from calc import add,multiply
print(add(4,6))
print(multiply(3,9))

#输出的结果如下
10
27
```

这段代码的含义是从 calc 模块中导入 add 和 multiply 两个函数。调用求和 add 函数,并传入 4 和 6 作为参数。打印 add 函数的返回值,结果为 10。调用乘法 multiply 函数,并传入 3 和 9 作为参数。打印 multiply 函数的返回值,结果为 27。

通过 from 关键字调用这个模块中的多个函数。这种调用方式只调用需要的函数,提高了效率。

类似地,也可以调用导入模块中的所有内容,代码如下:

```
from calc import *
print(substract(10,3))
print(divide(36,9))
```

上述代码使用星号 * 来导入模块中的所有内容。一般情况下,不建议从模块或包内导入所有内容。因为,这种方式可能会导入一批未知的名称,来覆盖已经定义的名称,这种操作经常让代码变得难以理解。

通过导入模块,可以轻松地使用其中定义的函数和变量,而不需要重新编写这些代码。模块是 Python 中组织、重用和管理代码的重要工具,提供了一种将相关的代码组织在一起的方式,使代码更加模块化和更易维护。模块还可以用于共享代码,使其他开发者可以轻松地重用和扩展已有的功能。通过模块,可以提高开发效率、减少代码冗余,并轻松地扩展和重用代码。

5.10　__main__的使用

__main__是最高层级代码运行所在环境的名称。“最高层级代码”即用户指定最先启动运行的 Python 模块。它被称为“最高层级”是因为它将导入程序所需的所有其他模块。

有时“最高层级代码”也被称为应用的入口点。

一个模块可以通过检查自己的 __name__，来发现它是否在顶层环境中运行。这是允许在模块没有从导入语句中初始化的情况下有条件地执行代码的一个常见的语句：

```
if __name__ == '__main__':
    #执行在导入语句中未被初始化的语句
```

如果源程序作为主程序被执行，则 Python 解释器将内置变量__name__的值设置为“__main__”。如果文件是从其他模块中导入的，则变量__name__的值将为这个模块的名称，代码如下：

```
#第 5 章 5.18 最高层级代码中内置变量__name__的使用
print("start")
#定义一个 main 函数
def main():
    print("Hello World")

#使用指定的变量 __name__
if __name__ == "__main__":
    main ()

#输出的结果如下
start
Hello World
```

在上面的程序中，定义了一条输出语句和一个 main()函数，由于程序顺序执行，所以先输出 start，由于是在最高层级代码中运行，__name__的值为__main__，所以满足 if 条件，此时输出为 Hello World。

5.11 包的含义

在 Python 中，包(Package)是一个包含 Python 模块的文件夹。它是一种组织和管理 Python 代码的方式，可以避免命名冲突，并提供代码的可重用性。

包提供了一种方式来结构化大型的代码库，可以通过导入语句访问其他模块的代码。

在 Python 中，可以使用 import 语句来导入包和模块。要创建和使用包，可以按照以下步骤进行：

(1) 创建一个文件夹，作为包的根文件夹。

(2) 在该文件夹下创建一个名为__init__.py 的文件，该文件可以为空，也可以包含一些初始化代码或定义。

(3) 在该文件夹下创建其他的 Python 模块文件，这些文件可以被视为包的一部分。

(4) 使用 import 语句导入包或模块。

假设有一个名为 abc 的包，其中包含一个名为 calc 的模块（calc 模块的内容同上），包的工程结构如图 5-3 所示。

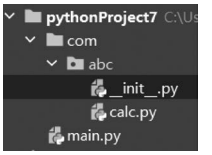

图 5-3　包的工程结构

在工程中，包含 com 文件夹，com 文件夹中包含包名 abc，包名 abc 中包含初始化文件__init__.py 和模块文件 calc.py。com 和 abc 中可以包含多个模块文件。

在进行调用时，要写出完整的模块路径名，示例代码如下：

```
from com.abc.calc import add    #从 com.abc.calc 包中导入 add 函数
print(add(3, 6)) #输出结果:9
```

上述代码表示从 com.abc.calc 包中导入 add 加法函数，返回结果并打印输出。

包的作用是将相关的模块组织在一起，使程序更加模块化和更易扩展。包内可以再有模块，也可以有类或者函数，包中也可以包含子包。

在使用包时，需要注意包的初始化文件__init__.py 的作用。该文件可以包含包的初始化代码、定义和其他模块的引用。当使用 import 语句导入包时，Python 会首先运行包的初始化文件，因此，可以在该文件中定义全局变量、函数或类等，以便在包的其他模块中使用。

在较复杂的项目中，通常会包含很多包和模块来完成不同的业务需求，它们之间可以相互调用。

5.12　第三方包和模块的安装

Python 语言标准库自带有许多内置的模块可供我们使用。Python 语言的强大之处在于它有很多开源的第三方包来供我们使用，第三方包的索引网址为 https://pypi.org/。可以从中查找、安装和发布 Python 软件包。当正在开发自己的项目时，很有可能会从中找到一个合适的特定任务包，能有效地使用它来满足我们的业务需求。

第三方包的安装方式有多种，可以从 pypi 官网或其他渠道下载 whl 格式的文件，下载后直接运行即可完成安装，也可以采用源码方式安装。最常用的是使用包管理器 pip 来下载和安装。

在 Python 语言中用于包管理的标准工具为 pip，它提供了一种简单和标准的方法来下载和安装第三方包。在使用 pip 命令下载第三方包时，它还能够处理包的依赖关系，并确保正确安装所需的依赖项。pip 是 Python 社区中最常用和被广泛接受的包管理工具，是下载和使用第三方包的首选工具。

使用 pip 命令下载第三方包的操作步骤如下：

（1）打开命令行终端或命令提示符。直接输入 pip 命令可以得到 pip 的帮助信息，如图 5-4 所示。

（2）使用以下命令来安装第三方包或模块：

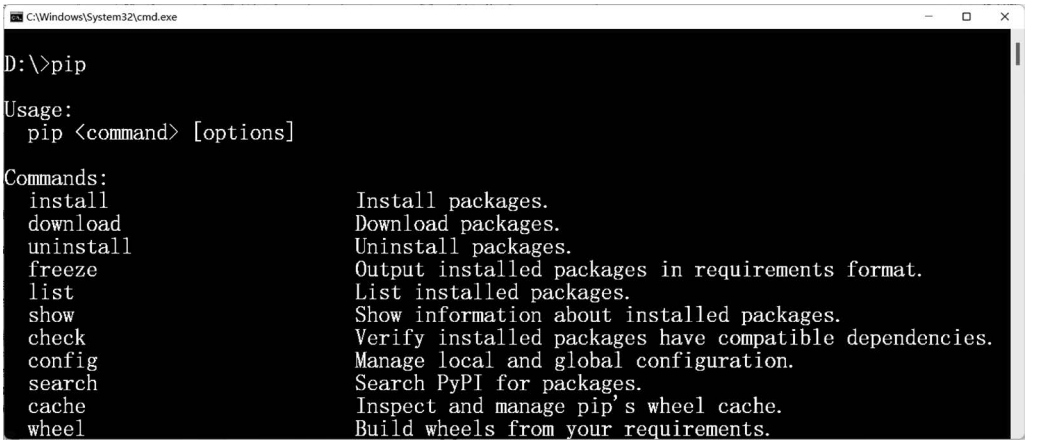

图 5-4　pip 帮助信息

```
pip install package_name
```

其中，package_name 是要安装的第三方包或模块的名称。

例如，要安装名为 requests 的第三方包，可以运行的命令如下：

```
pip install requests
```

这时，pip 将自动下载并安装所需的包和依赖项，如图 5-5 所示。

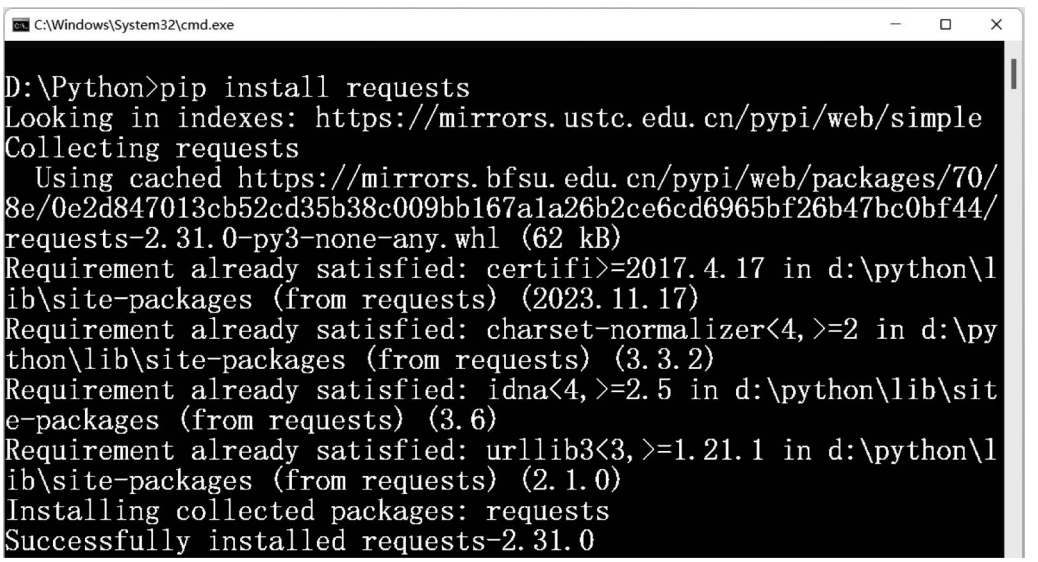

图 5-5　下载和安装 requests 包

(3) 安装完成后，可以在 Python 代码中导入并使用该包或模块。

例如，如果安装了 requests 包，则可以在代码中使用 import 关键字导入并使用它：

```
import requests

#使用 requests 进行 HTTP 请求等操作
```

(4) 也可以使用 pip 命令来卸载第三方的包或模块,命令的格式如下:

```
pip uninstall some_package
```

其中,some_package 为要卸载的第三方包的名称。

由于直接下载国外的软件速度比较慢,推荐使用国内镜像网站来下载第三方的包。

5.13　实训作业

(1) 编写一个函数,接收一个正整数参数,返回该数的阶乘。

(2) 编写一个函数,接收一个公司名称参数和公司员工姓名参数(员工数量不定)。