第 2 章

Python 编程基础

学习指引

本章重点学习 Python 中的一些语法元素结构和运算符，和学习其他语言一样，Python 也有自己的一些语法规则。作为开发人员，我们要遵循这些规则，开发起来才更加高效。Python 语言采用严格的“缩进”来表明程序的格式框架。缩进指每一行代码开始前的空白区域，用来表示代码之间的包含和层次关系。

重点导读

- 程序设计语言基础。
- 了解 Python 语言基础知识。
- 掌握 Python 程序开发环境的建立。
- 熟悉 Python 解释器与集成开发环境的使用。
- 熟悉程序运行流程。

2.1 编程基础知识

软件是按照需求事先设计并按照指定顺序执行的数据和指令的序列集合，是计算机系统中与硬件相互依存的部分。按功能划分软件可分为：系统软件和应用软件。系统软件是指用于控制计算机运行、管理计算机的各种资源，并为应用软件提供支持和服务的一类软件，如操作系统、数据库管理系统、设备驱动程序等；应用软件是指以实现某一专门的应用目的或特定服务而开发的计算机软件，如办公软件、视频软件、游戏以及财务管理软件等。

2.1.1 软件开发流程

软件开发流程即软件设计思路和方法实现的一般过程。一个软件的开发的完整过程，始于软件开发计划，止于软件运营维护，其中还包括设计软件的功能和实现的算法和方法、软件的总体结构设计和模块设计、编程和调试、程序联调和测试以及编写、提交程序等。

2.1.2 程序的运行流程

软件的运行过程就是模拟人类解决问题的思路、方法和手段并通过编译以计算机能够识别的形式告诉计算机，使得计算机能够根据人的指令一步一步去工作，完成某种特定的任务。这种运算交流的过程就是软件运行流程。程序运行通常是数据运算的过程，数据运算包括三个重要要素：输入数据（获取数据）、处理数据和输出数据，如图 2-1 所示。

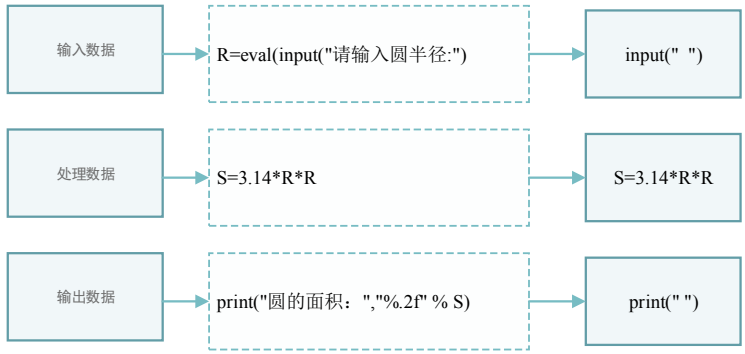

图 2-1 数据处理三要素

下面是一个非常简单的 Python 计算圆面积的程序。

【例 2-1】输入圆半径求圆面积。

```
R=eval(input("请输入圆半径:"))          #运行程序提示“请输入圆半径：”
S=3.14*R*R                              #将圆的半径值输入圆的面积公式中并计算
print("圆的面积：","%.2f" % S)          #输出圆的面积并保留两位小数
```

程序运行流程中比较简单的有数据存取，加减乘除，逻辑运算，复杂的向量运算等。如果将各种运算叠加起来，就可以实现各种复杂的运算功能。各种游戏都是从最基本的简单运算开始一步一步到复杂运算来实现的。

1.输入数据

输入数据（Input）是一个程序的开始。程序要处理的数据有多种来源，形成了多种输入方式，包括文件输入、网络输入、控制台输入、交互界面输出、随机数据输入、内部参数输入等。

2.处理数据

处理数据（Process）是程序对输入数据进行计算产生输出结果的过程。计算问题的处理方法统称为“算法”，它是程序最重要的组成部分。可以说，算法是一个程序的灵魂。

3.输出数据

输出数据（Output）是程序展示运算成果的方式。程序的输出方式包括：控制台输出、图形输出、文件输出、网络输出、操作系统内部变量输出等。

2.2 Python 程序元素构成

用 Python 编写的程序与其他编程语言一样，也有自己的基本结构和写法规范。

【例 2-2】程序元素构成。

```
num=int(input("输入一个数字："))
if num%2==0:      #判断该数字能否整除 2,成立则执行下面的语句,否则则执行对应外层 else 后的语句
  if num%5==0:    #如果该数字能整除 2,二次判断能否整除 5
     print ("你输入的数字可以整除 2 和 5")
  else:
     print ("你输入的数字可以整除 2,但不能整除 5")
else:
  if num%5==0:    #如果该数字不能整除 2,则判断能否整除 5,成立则执行下面的语句,否则执行 else 后的语句
     print ("你输入的数字可以整除 5,但不能整除 2")
  else:
    print ("你输入的数字不能整除 2 和 5")
```

在这一段 Python 程序代码中包括：注释、缩进、变量、赋值语句、输入输出语句、程序分支语句等程序元素。

2.3 Python 基本语法元素

Python 基本语法包括程序层次结构、代码注释、换行与并行、变量与保留字、字符串、程序分支语句、赋值语句和数据输入与输出等元素。下面依次介绍一下这几种元素，学习它们的使用方法和在使用过程中应该要注意哪些。

2.3.1 程序层次结构

习惯了 C 语言、C++之类的程序结构，初学 Python 者经常会被莫名奇妙的缩进错误给整迷糊，Python 必须使用正确的缩进格式。在 Python 里不能用大括号“{ }”来表示语句块，也不能用开始或结束标志符来表示，而是靠缩进来表示程序的层次结构，“缩进”不仅是为了让程序结构好看。

空白（缩进）在 Python 中是非常重要的。缩进是指每一行代码前端的空白区域，用来识别代码之间的包含和层次关系。这意味着同一层次的语句必须有相同的缩进。每一组这样的语句称为一个块。借用“缩进”的方式会使程序层次结构非常清晰，便于代码阅读。

在 Python 代码编写过程中，缩进可以通过按 Tab 键或使用多个空格（通常是 4 个空格）来实现。例如如下的一段 Python 程序代码。

【例 2-3】程序层次结构。

```
num=int(input("输入一个数字："))
if num%2==0:
   if num%5==0:                              #单层代码缩进
      print ("你输入的数字可以整除 2 和 5")      #多层代码缩进（缩进嵌套）
   else:
      print ("你输入的数字可以整除 2,但不能整除 5")
else:
   if num%5==0:
      print ("你输入的数字可以整除 5,但不能整除 2")
   else:
      print  ("你输入的数字不能整除 2 和 5")
```

在该段代码中可以发现，除第 1、2、7 行代码外都存在缩进，不需要缩进的代码顶行编写，不留空白

（缩进）。其中，第 3 行代码采用单层代码缩进，第 4 行用到了多层代码缩进（嵌套缩进）。通过缩进可以很清楚地分清哪个 if 与 else 是相匹配的条件判断。通常来说，在代码中判断、循环、函数以及类等语法形式使用缩进形式来标识代码间的包含关系，能更清晰地传达语义。但是，如果是非常简单的语句不表达包含关系，就不需要使用缩进了。

```
    print ("Hello World!")
    print ("I like Python.")
```

值得注意的是，处于同一级别的代码缩进量和缩进的符号（Tab 键或空格）要保持一致，这样才能保持嵌套的层次关系清晰正确。否则，由于缩进的方式不一致可能导致嵌套错误，甚至会影响程序的正确运行。另外，在 Python 的代码缩进中最好采用空格的方式，每一层向右缩进 4 个空格，通常不建议采用 Tab 键，更不能两种混合使用。

另外，现在有一些 Python 辅助开发工具可以自定义，按一次 Tab 键生成 4 个空格的代码缩进。还有一些工具可以自动实现代码缩进，这些都可以给程序编写带来极大的方便。

2.3.2 代码注释

在大多数编程语言中，注释都是一项很有用处的功能。注释是程序员在程序代码中添加的一行或多行说明信息，在编程中是很重要的部分。由于注释不是程序的组成部分，所以注释是不被计算机执行的。但是可以让程序代码更易于被其他程序员阅读，它能告诉你这段代码是干什么用的，提示代码的可读性。由于注释不被程序所执行，可以借用注释来删除或跳过一部分暂时不需要执行的代码。例如，在如下代码中，第 1 行就是一个注释，会被编译或者解释器略去，是不被计算机执行的。

【例 2-4】代码注释。

```
    #下面将打印出语句"Hello World!"
    print ("Hello World!")
```

Python 语言有两种使用注释的方法：单行注释和多行注释。单行注释是在每一行的前面输入“#”号，“#”号后面的内容都会被 Python 解释器忽略，如下所示。

```
    #这条是注释
    #这条还是注释
    #这条也是注释
```

多行注释是使用三个单引号（'''）来添加多行注释，如下所示。

```
    '''
    这条是注释
    这条还是注释
    这条也是注释吆
    '''
```

1. 注释的意义

在程序中编写注释的目的是表明代码要做什么，以及是如何做。在项目开发期间，程序员可能对程序如何工作及原理了如指掌，但过一段时间后，部分细节问题可能会被遗忘。当然没注释的程序是可以花费时间重新研究代码来确定各个部分的工作原理，这势必会浪费很多时间和精力。但如果通过编写注释，以清晰的自然描述语言对程序解决方案进行阐述，可节省很多时间和精力。

现在编写项目程序，大多是团队合作，可能是跨部门程序员也可能是跨公司的程序员，甚至是跨国的程序员在一起开发一个项目。清晰规范的程序重要，清晰简洁的程序注释也同样重要，这样才能被别的程序员看懂程序，程序才能相互更好地融合在一起，更利于团队项目的开发和合作。

要想成为专业的程序员，或与其他程序员有良好的合作，就必须编写有意义的注释，训练有素的程序员，都希望代码中包含注释，因此在程序中添加描述性的语言注释，是新手最值得养成的习惯和素养之一。

2. 注释的主要用途

程序注释在程序开发中的用途主要表现在如下几个方面。

（1）标注软件作者及版权信息。

在每个程序源代码文件的开始前增加注释，如标记、编写代码的作者、日期、用途、版权声明等信息。根据注释内容可采用单行或多行注释。

```
# ------------------------------------------------------------------------------------------
# 版权所有：
# 软件名称：
# 软件版本：
# 软件功能描述：
# Author：
# 时间：
'''
================JUOOCKEJI_LICENSE================
￥%012101210106006701260032009700940075006300590 1
270127003240990103010500320089009800320096009001 21
012500650088 01! &1040107004587640094011000930098 01
060071009701050106010&*￥#~036006500960088012200 77
012700580&%@76200710078007101030058005700610103 00
98011800%#
'''
// 其他注释信息：
# ------------------------------------------------------------------------------------------
```

（2）注释代码原理和用途。

在程序关键代码附近增加注释，解释核心代码的用处、原理及注意事项，增加程序的可读性。由于程序本身已经表达了功能意图，为了不影响程序阅读连贯性，程序中的注释一般采用单行注释，标记在关键行与关键代码同行。对于一段关键代码，可以在附近选择一个多行注释，或者多个单行注释，给出代码设计原理等信息。

（3）辅助程序调试。

在调试程序时，可以通过单行或多行注释，临时去掉一行或多行与当前调试无关的代码，辅助程序员找到程序发生问题的可能位置。

2.3.3 换行与并行

在 Python 程序编写过程中，有时会遇到两行代码放在同一行更易懂或者一行中过长的代码为了结构清晰易懂不适合放到同一行中。下面将探讨在 Python 中如何处理代码换行与并行的问题。

1. 代码换行

在 Python 编程中一般是一行写完所有代码，如果遇到一行写不完需要换行的情况，也允许采用代码换行的方式将一行代码分成多行编写。有如下 4 种方法供选择。

【例 2-5】代码换行。

（1）在该行代码末尾加上续行符“\”。

```
print ("2019 年\
      我在学习\
      Python!")
输出结果：2019 年 我在学习 Python!
```

（2）语句中包含()、{}、[]时分行不需要加换行符。

```
print ('2019 年'
      '我在学习'
      'Python!')
输出结果：2019 年我在学习 Python!
```

（3）采用三个单引号“''”。

```
print ('''2019 年'''
       '''我在学习'''
       '''Python!''')
输出结果：2019 年我在学习 Python!
```

（4）采用三个双引号“"""”。

```
print ("""2019 年"""
       """我在学习"""
       """Python!""")
输出结果：2019 年我在学习 Python!
```

2. 代码并行

在 Python 代码缩进语句块中如果只有一条语句，将下句代码直接写在“:”语句后面也是正确的。

【例 2-6】代码并行。

```
num=int(input("输入一个数字："))
if num%2==0:
    if num%5==0:                                    #该行不允许并到上一行
       print("你输入的数字可以整除 2 和 5")          #该行允许并到上一行
    else:
       print("你输入的数字可以整除 2,但不能整除 5")  #该行允许并到上一行
else:
    if num%5==0:                                    #该行不允许并到上一行
       print("你输入的数字可以整除 5,但不能整除 2")  #该行允许并到上一行
    else:
       print("你输入的数字不能整除 2 和 5")          #该行允许并到上一行
```

在上述程序代码的第 03 行和 08 行代码是不被允许并行到上行代码“:”语句后面的。因为第 03 行和 08 行代码后还包含一个判断语句块，不是独立的一条语句。其他代码并行后结果如下：

```
num=int(input("输入一个数字："))
if num%2==0:
    if num%5==0:print ("你输入的数字可以整除 2 和 5")
    else:print ("你输入的数字可以整除 2,但不能整除 5")
else:
    if num%5==0:print ("你输入的数字可以整除 5,但不能整除 2")
    else:print  ("你输入的数字不能整除 2 和 5")
```

在 Python 代码中除了可以将“:”语句单独一条语句并行，也可以将“;”后的语句进行并行，并支持连续的并行。

```
num1 =5;
num2 =6;    #该行允许并到上一行
```

```
print(num1+num2);
输出结果：11
```

在上述程序代码中的第 02 行代码允许并行到上行代码“;”语句的后面。并行结果如下：

```
num1 =5;num2 =6;
print(num1+num2);    #该行也允许并到上一行
输出结果：11
```

在上述程序代码中的第 02 行也允许并行到上行代码“;”语句的后面。并行结果如下：

```
num1 =5;num2 =6;print(num1+num2);
输出结果：11
```

注意：在 C、Java、PHP 等语言的每一条语句最后加个分号，是语法要求。但是对于 Python 语言，分号是可加可不加的，因为 Python 是靠换行来区分代码语句的，这里建议最好还是不加分号。

2.3.4 变量与保留字

在 Python 程序中是通过“变量”来存储和标识具体数据值的，数据的调用和操作是通过变量的名称。这就需要给程序“变量”元素关联一个标识符（命名），并保证其唯一性。在 Python 中对“变量”命名时，需要遵守一些命名规则。违反这些规则将可能引发程序错误。请牢记下述有关变量命名的规则。

（1）变量名只允许包含字母（a～z，A～Z）、数字和下画线。变量名可以以字母或下画线开头，但第一个字符不能是数字。例如，可将变量命名为 username 或者 userName2，但不能将其命名为 2userName。

（2）变量名不允许包含空格，但可使用下画线来分隔其中的单词。例如，变量名命名为 user_name 是可行的，但命名为 user name 是不被允许的，会引发错误。

（3）在 Python 程序中对大小写是敏感的。例如，username 和 userName 是不同的变量名。

（4）变量命名应既简短又具有描述性。例如，name 比 n 好，user_name 比 u_n 好。

（5）慎用小写字母 l 和大写字母 O，因为它们可能被人错看成数字 1 和 0。另外，字母 p 的大小写也应慎用，不易区分。

（6）不要使用 Python 程序已经保留用于特殊用途的 Python 关键字和函数名作为变量名，如 print、if、for（如下所示）。

fals	none	true	and	as
assert	break	class	continue	def
del	elif	else	except	finally
for	from	global	if	import
in	is	lambda	nonlocal	not
or	pass	raise	return	try
while	with	yield		

“保留字”指在高级程序语言中已经被定义过的字，不允许使用者再将这些字作为变量名或常量名使用。

注意：编写 Python 程序过程中，建议使用小写的 Python 变量名。在变量名中使用大写字母虽然不会导致错误，但避免使用大写字母，这样可以更利于程序代码的阅读。

2.3.5 字符串

字符串表示的是文本，通常是指要展示给别人的或者是想要从程序里“输出”的一小段字符。在 Python 中可以对文本通过双引号（""）或者单引号（''）标注来识别出字符串来。例如：

```
name = "python"或 userage = '18'
```

在本例中写的是 userage = '18'，所以 userage 就是一个字符串。但是如果写的是 userage = 18（没有引号），那么 userage 就不是一个字符串了，而变成了一个整数。

另外，在 Python 中，可以使用字符串操作符“+”（加号）实现两个字符串的连接操作。例如，字符串"python"+" is good！"和"python"+" is good！"与'python is good'所表达的字符串的值是相同的。

2.3.6 程序分支语句

在 Python 中采用 if-elif-else 描述多分支结构，是对上级 if 判断条件语句为真值情况的二次判断，甚至多次判断。语句格式如下：

```
if  <条件表达式 1>:
    语句块 1
    if <条件表达式 2>:
    语句块 2
    elif <条件表达式 3>:
     语句块 3
    else
     语句块 4
elif <条件表达式 4>:
    语句块 5
else:
    语句块 6
```

在 2.1 节的程序范例中，首先程序对所输入的数字进行与 2 整除结果判断（if num%2==0），如果条件满足再进行与 5 整除结果判断（if num%5==0），最后程序根据这两项判断条件是否成立（为真）的情况，给出所输入数字被 2 或 5 整除情况的字符串信息。

2.3.7 赋值语句

在前面的程序中运用了一条 num=int(input("输入一个数字："))语句，其中的“=”在 Python 中表示“赋值”，包含“=”的语句在 Python 中称为赋值语句。“=”是一个赋值符号，表示将“=”右边的值赋给“=”左侧的变量，在本语句中表示将“=”右侧获取到的输入数字赋给左侧的 num 变量。“=”赋值符号和数学中的“=”号的含义是不一样的。

另外，在 Python 中还有一种是同步赋值语句，该语句可以同时对多个变量赋值（先运算右侧 N 个表达式，然后同时将表达式结果赋给左侧）语法如下：

```
<变量 1>,…,<变量 N>=<表达式 1>,…,<表达式 N>
```

例如：交换变量 x 和 y

如果采用单个赋值，需要 3 行语句：

```
>>>z = x
>>>x = y
>>>y = z
```

在本例中即通过一个临时变量 z 先缓存下 x 的原始值，然后将 y 值（交换）赋给 x，最后将 x 的原始值再通过 z（交换）赋值给 y，完成变量 x 和 y 值的交换操作。

如果采用同步赋值语句的方式，不需要借用临时变量缓存数值，仅需要一行代码即可：

```
>>>x, y = y, x
```

同步赋值语句可以让赋值过程变得更便捷，减少变量的使用，使赋值语句更简洁易懂，提高程序的可读性。

另外，在 Python 程序中，赋值语句 x = y 和 y = x 的含义是不同的。例如：

```
>>>x = 3
>>>y = 9
>>>x = y
>>>print ("x 的值是：",x)
>>>print ("y 的值是：",y)
```

注意：上述代码需要一行一行地输入和执行，否则会报语法错误。

在本例中，虽然 x 的初始值是 3（在第 1 行中赋值的），但在第 03 行 x=y 的赋值语句中又把 y 的值（9）赋值给了 x，现在 x 的值已经由最初的 3 变成了 9。y 的值没有被重新赋值保持不变。所以程序执行输出的数值均为 9，如下所示。

```
x 的值是：9
y 的值是：9
```

接下来，将范例中第 03 行赋值语句修改为 y = x。范例如下：

```
>>>x = 3
>>>y = 9
>>>y = x
>>>print ("x 的值是：",x)
>>>print ("y 的值是：",y)
```

在数学运算中通常 x = y 和 y = x 有着相同的含义，然而在程序中它们的含义却发生了变化。在第 03 行通过 y=x 的赋值语句中把 x 的值（3）赋值给了 y，现在 y 的值已经由最初的 9 变成了 3。x 的值没有被重新赋值保持不变。所以程序执行输出的数值均为 3，如下所示。

```
x 的值是：3
y 的值是：3
```

2.3.8 数据输入与输出

在 Python 编程中是通过 Python 内置的 input()函数和 print()函数实现数据的输入读取和输出显示信息的。下面将学习 Python 数据的输入与输出。

1. input()函数

input()函数可以让程序暂停运行，等待用户输入数据信息。程序在获取用户输入的信息后，Python 将其存储在一个变量中，以方便后面程序的使用。

在 2.1 节的程序范例中的第 01 行就用到了 input()函数。

```
num=int(input("输入一个数字："))
```

input()函数接受一个参数，即要向用户显示的提示或说明，让用户知道该做什么。在这个范例中，Python 运行到第 01 行代码时，用户将看到提示“输入一个数字：”。程序等待用户输入数字，当用户完成数字的输入并按 Enter 键后程序才继续运行。用户所输入数字存储在变量 num 中。

input()输入函数的语法如下：

```
<变量>=input(<提示性文字>)
```

在 Python 3.X 中，input()函数获得的用户输入均以字符串形式保存在变量中，参见如下范例代码。

【例 2-7】input()输入函数。

```
>>> input_string=input("请输入：")
请输入：我在学习 Python
>>> print (input_string)
我在学习 Python
>>> input_string=input("请输入：")
请输入：2018
>>> print (input_string)
2018
>>>
```

无论用户输入的是数字还是字符，input()函数都统一按照字符串的类型输出显示。在例 2-7 的第 06 行输入 2018 时，input()函数以字符的形式输出。

2. print()函数

print()函数向用户或者屏幕上输出指定的字符信息。在 print()函数的括号中加上字符串，就可以向屏幕上输出指定的文字。例如输出“hello，world”，用代码实现如下。

```
>>> print(hello, world)
```

print()函数也可以接受多个字符串，用逗号“,”隔开，就可以连成一串输出：

```
>>> print("hello,world ","Python ","是一门优秀的编程语言！")
hello, world Python 是一门优秀的编程语言！
```

print()会依次打印输出每个字符串，遇到逗号“,”会输出一个空格，因此，输出的字符串是就是这样拼起来的。

print()输出函数的语法如下：

```
print(value, …, sep=' ', end='\n', file=sys.stdout, flush=False)
```

- 参数 sep 是实现分隔符，例如多个参数输出时想要输出中间的分隔字符；
- 参数 end 是输出结束时的字符，默认是换行符\n；
- 参数 file 是定义流输出的文件，可以是标准的系统输出 sys.stdout，也可以重定义为别的文件；
- 参数 flush 是判断是否立即把内容输出到流文件，不做缓存（这里是 sys.stdout，也就是默认的显示器）。

print()输出函数中的 sep、end、file、flush 参数是 4 个可选参数。具体应用方法如下。

（1）sep 参数：在输出字符串之间插入指定字符串，默认是空格，范例代码如下。

【例 2-8】print()输出函数中的 sep 参数。

```
>>> print("a","b","c",sep="$$")   #将默认空格分隔符修改为"$$"
a$$b$$c
```

（2）end 参数：在 print 输出语句的结尾加上指定字符串，默认是换行（\n），范例代码如下。

【例 2-9】print()输出函数中的 end 参数。

```
>>> print("a","b","c", end="；")   #将默认空格分隔符修改为"；"
a b c；
```

注意：print 默认是换行，即输出语句后自动切换到下一行，对于 Python 3.X 来说，如果要实现输出不换行的功能，可以设置 end=" "（Python 2 可以在 print 语句之后加“,”实现不换行的功能）。

（3）file 参数：指定文本将要发送到的文件、标准流或者其他类似文件的对象，默认是 sys.stdout，范例代码如下：

【例 2-10】print()输出函数中的 file 参数。

```
>>> print(1,2,3,sep='-',end='; \n',file=open('print.txt','a')) #执行 4 次 print()函数
1-2-3;
1-2-3;
1-2-3;
1-2-3;
```

在本例中，file=open('print.txt','a')设置了输出文件路径，'a'设置了打开文件的方式是添加模式，所以字符串会加在文件末尾，不会重写文件。其中，sep='-'参数设置了字符写入时的分隔符（-）；end='；\n'参数设置了字符写完后的结尾符号（；）及换行（\n）。另外，执行该函数会在 Python 软件根目录中新建一个 print.txt 文本文件用于写入本例指定文本，如图 2-2 所示。

（4）flush 参数：flush 参数值为 True 或者 False，默认为 False，表示是否立刻将输出语句输入到参数 file 指向的对象中（默认是 sys.stdout），范例代码如下。

【例 2-11】print()输出函数中的 flush 参数。

```
>>>f = open('print.txt','w')
>>>print('python',file=f)    #将 python 字符文本输入到 print.txt 文本文件中
```

可以看到 print.txt 文件这时为空，只有执行 f.close()之后才将内容写进文件，如图 2-3 所示。

图 2-2 print.txt 文本文件

图 2-3 print.txt 空文本文件

在这里将 file=f 参数值修改为 True，则立刻就可以看到指定文件的输出函数内容，如图 2-4 所示。

```
>>> f = open('print.txt','w')
>>> print('python',file=f,flush=True)
```

flush 参数的功能在客户端脚本中几乎用不上，多用于服务器端。例如，在线 Web 即时聊天页面会实时显示聊天的内容，其实后台是在一直向服务器请求数据的，正常情况下是请求完毕之后才会输出相应的请求内容，但是因为是即时聊天，就需要一有信息响应就立即返回，在这里 flush 也就起作用了。

图 2-4 输出函数立即写入 print.txt 文件

2.4 就业面试技巧与解析

面试官问：什么是 PEP8？

应聘者：PEP8 是一个编程规范，是可以使程序代码整洁美观，更具可读性的建议。