新起点电脑教程

第 2 章

基础语法知识介绍

本章要点

- 独有的缩进规则
- 注释
- 标识符和关键字
- 变量
- 输入和输出
- 字符串
- 数字类型

本章主要内容

语法知识是任何一门开发语言的核心内容，Python 语言也不例外。在本章的内容中，将详细介绍 Python 语言的基本语法知识，主要包括缩进规则、注释、输入和输出等内容，为读者步入本书后面知识的学习打下基础。

2.1 独有的缩进规则

Python 语言要求编写的代码最好全部使用缩进来分层(块)。代码缩进一般用在函数定义、类的定义以及一些控制语句中。一般来说，行尾的冒号“:”表示下一行代码缩进的开始，即使没有使用括号、分号、大括号等进行语句(块)的分隔，通过缩进分层也可以使结构非常清晰。

↑扫码看视频

Python 语言规定，缩进只使用空白实现，必须使用 4 个空格来表示每级缩进。使用 Tab 字符和其他数目的空格虽然都可以编译通过，但不符合编码规范。支持 Tab 字符和其他数目的空格仅仅是为了兼容旧版的 Python 程序和某些有问题的编辑器。要确保使用一致数量的缩进空格，否则编写的程序将显示错误。请看下面的实例代码，演示了缩进 Python 代码的过程。

实例 2-1：在 Python 程序中使用缩进
源文件路径：daima\2\2-1

实例文件 suojin.py 的具体实现代码如下所示。

```
# if True 是一个固定语句，后面的总是被执行
if True:
    print("Hello girl!")      #缩进 4 个空白的占位
else:                         #与 if 对齐
    print("Hello boy!")       #缩进 4 个空白的占位
```

```
=======
Hello girl!
>>>
```

图 2-1 执行后的效果

在上述代码中，使用了 4 个空白的缩进格式，执行后的效果如图 2-1 所示。

再看如下所示的代码，使用了不同的缩进方式。

```
if True:
    print("Hello girl!")
else:
    print("Hello boy!")
print("end")                  #改正时只需将这行代码前面的空格全部删除即可
```

在上述代码中，实现缩进的方式不一致，有的是通过 tab 键实现的，有的是通过空白实现的，这是 Python 语法规则所不允许的，所以执行后会出错，如图 2-2 所示。

```
>>> if True:
    print("Hello girl!")
else:
    print("Hello boy!")
  print("end")

SyntaxError: unindent does not match any outer indentation level
```

图 2-2 出错

2.2　注　　释

在计算机语言中，注释可以帮助阅读程序，通常用于概括算法、确认变量的用途或者阐明难以理解的代码段。注释并不会增加可执行程序的大小，编译器会忽略所有注释。

↑扫码看视频

在 Python 程序中有两种类型的注释，分别是单行注释和多行注释。

(1) 单行注释

单行注释是指只在一行中显示注释内容，Python 中的单行注释以#开头，具体语法格式如下：

```
# 这是一个注释
```

例如下面的代码：

```
# 下面代码的功能是输出：Hello, World!
print("Hello, World!")
```

(2) 多行注释

多行注释也被称为成对注释，是从 C 语言继承过来的，这类注释的标记是成对出现的。在 Python 程序中，有两种实现多行注释的方法。

- 第一种：用三个单引号“'''”将注释括起来。
- 第二种：用三个双引号“"""”将注释括起来。

下面是用三个单引号创建的多行注释：

```
#!/usr/bin/python3
'''
这是多行注释，用三个单引号
这是多行注释，用三个单引号
这是多行注释，用三个单引号
'''
print("Hello, World!")
```

下面是用三个双引号创建的多行注释：

```
#!/usr/bin/python3
"""
这是多行注释，用三个双引号
这是多行注释，用三个双引号
这是多行注释，用三个双引号
"""
print("Hello, World!")
```

当使用上述多行(成对)注释时，编译器把放入注释对(3 个双引号或 3 个单引号)之间的内容作为注释。任何允许有制表符、空格或换行符的地方都允许放注释对。注释对可以跨越程序的多行，但不是一定要如此。当注释跨越多行时，最好能直观地指明每一行都是注释的一部分。

实例 2-2：使用注释
源文件路径：daima\2\2-2

实例文件 zhushi.py 的具体实现代码如下所示。

```
'''
print("我在注释里")              #这部分是注释
print ("我还在注释里")           #这部分是注释
'''
print("我在注释的外面")
```

```
=======
我在注释的外面
>>>
```

图 2-3 执行后的效果

在上述代码中，虽然前两个 print 语句是 Python 格式的代码，但是因为在注释标记内，所以执行后不会显示任何效果。此实例执行后的效果如图 2-3 所示。

知识精讲

在 Python 程序中通常混用上述两种注释形式。太多的注释混入程序代码可能会使代码难以理解，通常最好是将一个注释块放在所解释代码的上方。当改变代码时，注释应与代码保持一致。程序员即使知道系统其他形式的文档已经过期，但还是会信任注释，认为它会是正确的。错误的注释比没有注释更糟，因为它会误导后来者。

2.3 标识符和关键字

标识符和关键字都是具有某种意义的标记和称谓，就像人的外号一样。在本书前面的演示代码中，已经使用了大量的标识符和关键字。例如代码中的分号、单引号、双引号等就是标识符，而代码中的 if、print 等就是关键字。

↑扫码看视频

Python 语言的标识符使用规则和 C 语言类似，具体说明如下所示。

- 第一个字符必须是字母或下划线(_)。
- 剩下的字符可以是字母和数字或下划线。
- 大小写敏感。

- 标识符不能以数字开头；除了下划线，其他的符号都不允许在开头使用。处理下划线最简单的方法是把它们当成字母字符。大小写敏感意味着标识符 foo 不同于 Foo，而这两者也不同于 FOO。
- 在 Python 3.x 中，非 ASCII 标识符也是合法的。

关键字是 Python 系统保留使用的标识符，也就是说只有 Python 系统才能使用，程序员不能使用这样的标识符。关键字是 Python 中的特殊保留字，开发者不能把它们用作任何标识符名称。Python 的标准库提供了一个 keyword module，可以输出当前版本的所有关键字，执行后的效果如下所示。

```
>>> import keyword    #导入名为 keyword 的内置标准库
>>> keyword.kwlist    # kwlist 能够列出所有内置的关键字
['False', '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']
```

2.4 变　　量

变量是指其值在程序的执行过程中可以发生变化的量。变量是计算机内存中的一块区域，变量可以存储规定范围内的值，而且值可以改变。基于变量的数据类型，解释器会分配指定内存，并决定什么数据可以被存储在内存中。常量是一块只读的内存区域，一旦被初始化就不能被改变。

↑扫码看视频

2.4.1 声明变量

Python 语言中的变量不需要声明，变量的赋值操作即是变量的声明和定义的过程。在内存中创建的每个变量都包括变量的标识、名称和数据等信息。

实例 2-3：输出变量的值
源文件路径：daima\2\2-3

实例文件 bianliang.py 的具体实现代码如下所示。

```
x = 1                                #变量赋值定义一个变量 x
print(id(x))                         #打印变量 x 的标识
print(x+5)                           #使用变量
print("=========华丽的分割线=========")
x = 2                                #量赋值定义一个变量 x
print(id(x))                         #此时的变量 x 已经是一个新的变量
print(x+5)                           #名称相同，但是使用的是新的变量 x
```

```
x = 'hello python   '          #将变量赋值定义为一个文本字符串
print(id(x))                   #函数 id()的功能是返回对象的"身份证号"
print(x)                       #现在会输出文本字符串
```

在上述代码中对变量 x 进行了 3 次赋值，首先给变量 x 赋值为 1，然后又重新给变量 x 赋值为 2，然后又赋值变量 x 的值为“hello python”。在 Python 程序中，一次新的赋值将创建一个新的变量。即使变量的名称相同，变量的标识也不相同。执行后的效果如图 2-4 所示。

```
1909637696
6
=========华丽的分割线=========
1909637728
7
2317816604400
hello python
>>>
```

图 2-4　执行后的效果

智慧锦囊

值得读者注意的是，上述代码中的 id()是 Python 中的一个内置函数，功能是返回对象的“身份证号(内存地址)”，值唯一且不变，但在不重合的生命周期里，可能会出现相同的 id 值。print(id(x))的功能是返回变量 x 的内存地址。

2.4.2　局部变量

在 Python 程序中，局部变量是只能在函数或者代码块内使用的变量，函数或者代码块一旦结束，局部变量的生命周期也将结束。例如在下面的实例中，演示了局部变量只能在定义的函数或者代码块内使用的过程。

实例 2-4：使用局部变量
源文件路径：daima\2\2-4

在文件 file01.py 中定义了函数 fun()，在函数 fun()中定义了一个局部变量 local_var，并赋值为 100。局部变量 local_var 只在函数 fun()内有效，只能被函数 fun()访问。即使是在文件 file01.py 中定义的函数 fun2()也不能使用 local_var。

```
#fileName:file01
def fun():
    local_var = 100                #定义一个局部变量
    print(local_var)               #这行代码可以成功运行，打印输出变量值 100
def fun2():
    zero = local_var - 100         #在函数 fun2()中使用局部变量 local_var 是非法的
    print("get zero : %d"%zero)
fun()
#fun2()
print("local_var -1 = %d"%(local_var - 1)) #file01 中使用局部变量(不可以)
```

执行上述代码后会出错，执行效果如图 2-5 所示。

```
C:\Users\apple\AppData\Local\Programs\Python\Python36\python.exe H:/daima/2/2-2/file01.py
100
Traceback (most recent call last):
  File "H:/daima/2/2-2/file01.py", line 10, in <module>
    print("local_var -1 = %d"%(local_var - 1)) #file01中使用局部变量(不可以)
NameError: name 'local_var' is not defined
```

图 2-5　执行效果

而在另外一个实例文件 file02.py 中，即使使用 import 语句导入了上面文件 file01.py 中的功能，但是因为变量 local_var 是一个局部变量，所以不能在文件 file02.py 中生效。实例文件 file02.py 的具体实现代码如下所示。

```
import file01
file01.fun()
print(local_var)
```

2.4.3　全局变量

在 Python 程序中，在函数之外定义的变量叫作全局变量。全局变量能够被不同的函数、类或文件所共享，可以被文件内的任何函数和外部文件所访问。例如在下面的实例中，演示了使用全局变量的过程。

实例 2-5：使用全局变量
源文件路径：daima\2\2-5

实例文件 quan.py 的具体实现代码如下所示。

```
g_num1 = 1                      #定义全局变量
g_num2 = 2                      #定义全局变量
def add_num():
    global g_num1               #引用全局变量
    g_num1 = 3                  #修改全局变量的值
    result = g_num1 + 1
    print("result : %d"%result)
def sub_num():
    global g_num2               #使用 global 关键字
    g_num2 = 5
    result = g_num2 - 3
    print("result : %d"%result)
①add_num()
②sub_num()
③print("g_num1:%d "%g_num1)
④print("g_num2:%d "%g_num2)
```

在上述代码中，在函数外部分别定义了两个全局变量 g_num1 和 g_num2，并分别设置初始值为 1 和 2。在函数 add_num()内部使用了全局变量 g_num1，在使用时用到了关键字 global。

① 在调用函数 add_num()时，result 为局部变量，执行后会输出“4”。

② 在调用函数 sub_num()时，result 为局部变量，执行后会输出“2”。

③ 在执行 add_num()函数时，使用关键字 global 改变了全局变量 g_num1 的值，执行后会输出“3”。

④ 在执行 sub_num()函数时，使用关键字 global 改变了全局变量 g_num2 的值，执行后会输出“5”。

实例文件 quan.py 的执行效果如图 2-6 所示。

```
result : 4
result : 2
g_num1:3
g_num2:5
```

图 2-6 执行效果

2.5 输入和输出

Python 程序必须通过输入和输出才能实现用户和电脑的交互，才能实现软件程序的具体功能。对于所有的软件程序来说，输入和输出是用户与程序进行交互的主要途径，通过输入程序能够获取运行所需的原始数据，通过输出程序能够将数据的处理结果输出，让开发者了解程序的运行结果。

↑扫码看视频

2.5.1 实现输入功能

要想在 Python 程序中实现输入功能，就必须调用其内置函数 input()，其语法格式如下所示：

```
input([prompt])
```

其中的参数 prompt 是可选的，意思是既可以使用，也可以不使用。参数 prompt 用来提供用户输入的提示信息字符串。当用户输入程序所需要的数据时，就会以字符串的形式返回。也就是说，函数 input 不管你输入的是什么，最终返回的都是字符串。如果需要输入数值，则必须经过类型转换处理。

实例 2-6：使用函数 input()实现输入功能
源文件路径：daima\2\2-6

实例文件 input.py 的具体实现代码如下所示。

```
name = input('亲，请输入你的名字：')
```

在上述代码中，函数 input()的可选参数是“亲，请输入你的名字：”，这个可选参数的作用是提示你输入名字，这样用户就会知道将要输入的是什么数据，否则用户看不到相关提示，可能认为程序正在运行，而一直等待运行结果。执行后将在界面中显示“亲，请输入你的名字：”，之后等待用户的输入。当用户输入名字“西门吹雪”并按 Enter 键时，程序就接收了用户的输入。之后，用户输入变量名 name，就会显示变量所引用的对象——用户输入的名字“西门吹雪”。最终的执行效果如图 2-7 所示。

```
========
亲，请输入你的名字：西门吹雪
>>> name
'西门吹雪'
>>>
```

图 2-7　执行效果

2.5.2　实现输出功能

输出就是显示执行结果，这个功能是通过函数 print()实现的，在本书前面的实例中已经多次用到了这个函数。使用 print 加上字符串，就可以在屏幕上输出指定的文字。比如输出“hello, world”，用下面的代码即可实现：

```
>>> print 'hello, world'
```

在 Python 程序中，函数 print()的语法格式如下所示。

```
print (value,…,sep='',end='\n')  #此处只是展示了部分参数
```

各个参数的具体说明如下所示。

- value：用户要输出的信息，后面的省略号表示可以有多个要输出的信息。
- sep：多个要输出信息之间的分隔符，其默认值为一个空格。
- end：print()函数中所有输出信息之后添加的符号，默认值为换行符。

在 Python 程序中，print 中也可以同时使用多个字符串，使用逗号“,”隔开，就可以连成一串输出，例如下面的代码：

```
>>> print 'The quick brown fox', 'jumps over', 'the lazy dog'
The quick brown fox jumps over the lazy dog
```

这样 print 会依次打印每个字符串，遇到逗号“,”时就会输出一个空格。另外，print 也可以打印整数或计算结果，例如下面的演示代码。

```
>>> print 300
300
>>> print 100 + 200
300
```

由此可见，我们可以把计算 100 + 200 的结果打印得更漂亮一点，例如下面的演示代码。

```
>>> print '100 + 200 =', 100 + 200
100 + 200 = 300
```

读者需要注意的是，对于“100 + 200”来说，Python 解释器自动计算出结果 300，但是，“100 + 200 =”是字符串而非数学公式，Python 把它视为字符串，需要我们自行解释上述打印结果。

实例 2-7：使用函数 print()输出结果
源文件路径：daima\2\2-7

实例文件 shuchu.py 的具体实现代码如下所示。

```
print('a','b','c')                    #正常打印输出
print('a','b','c',sep=',')            #将分隔符改为","
print('a','b','c',end=';')            #将分隔符改为";"
print('a','b','c')                    #正常输出
print('peace',22)                     #逗号进行分隔
```

在上述代码中使用了 5 条语句，调用了 5 次 print()函数。其中第 2 条语句将分隔符改为“,”，第 3 条语句将分隔符改为“;”。第 5 条语句演示了逗号的作用，这说明在使用 print 时可以在语句中添加多个表达式，每个表达式用逗号分隔。当使用逗号分隔进行输出时，print 语句会在每个输出项后面自动添加一个空格。不管是字符串还是其他类型，最终都将转化为字符串进行打印。

所以执行代码后，第 1 行为默认的输出，数据之间以空格分开，结束后添加了一个换行符；第 2 行输出的数据项之间以逗号分开；第 3 行输出结束后添加了分号，所以和第 4 条语句的输出放在了同一行中；第 5 行用逗号进行了分隔，执行后将 peace 和 22 显示在一行中。执行效果如图 2-8 所示。

```
=======
a b c
a,b,c
a b c;a b c
peace 22
>>>
```

图 2-8 执行效果

2.6 字 符 串

在 Python 程序中，字符串类型 str 是最常用的数据类型。我们可以使用引号(单引号或双引号)来创建字符串。创建 Python 字符串的方法非常简单，只要为变量分配一个值即可。

↑扫码看视频

2.6.1 实现字符串

例如在下面的代码中，“Hello World!”和“Python R”都属于字符串。

```
var1 = 'Hello World!'          #字符串类型变量
var2 = "Python R"              #字符串类型变量
```

在 Python 程序中，字符串通常由单引号(')、双引号(")、三个单引号或三个双引号包围的一串字符组成。当然这里说的单引号和双引号都是英文字符符号。

(1) 单引号字符串与双引号字符串本质上是相同的。但当字符串内含有单引号时，如果用单引号字符串就会导致无法区分字符串内的单引号与字符串标志的单引号，就要使用转义字符串，如果用双引号字符串就可以在字符串中直接书写单引号。例如：

```
'abc"dd"ef'
"'acc'd'12"
```

(2) 三引号字符串可以由多行组成(单引号或双引号字符串则不行)，当需要使用大段多行的字符串时就可以使用它。例如：

```
'''
这就是字符串
'''
```

在 Python 程序中，字符串中的字符可以包含数字、字母、中文字符、特殊符号，以及一些不可见的控制字符，如换行符、制表符等。例如下面列出的都是合法的字符串：

```
'abc'
'123'
"ab12"
"大家"
'''123abc'''
"""abc123"""
```

2.6.2　访问字符串中的值

在 Python 程序中，字符串还可以通过序号(序号从 0 开始)来取出其中的某个字符，例如'abcde.[1] '取得的值是'b'。

实例 2-8：访问字符串中的值
源文件路径：daima\2\2-8

实例文件 fangwen.py 的具体实现代码如下所示。

```
var1 = 'Hello World!'           #定义第一个字符串
var2 = "Python Toppr"           #定义第二个字符串
print ("var1[0]", var1[0])      #截取第一个字符串中的第一个字符
print ("var2[1:5]", var2[1:5]) #截取第二个字符串中的第 2～5 个字符
```

在上述代码中，使用方括号截取了字符串 var1 和 var2 的值，执行效果如图 2-9 所示。

```
========
var1[0] H
var2[1:5] ytho
>>>
```

图 2-9　执行效果

2.6.3　更新字符串

在 Python 程序中，开发者可以对已存在的字符串进行修改，并赋值给另一个变量。

实例 2-9：修改字符串中的某个值
源文件路径：daima\2\2-9

实例文件 gengxin.py 的具体实现代码如下所示。

```
var1 = 'Hello World!'   #定义一个字符串
print ("原来是：",var1)  #输出字符串原来的值
#截取字符串中的前 6 个字符
print ("下面开始更新字符串：", var1[:6] + 'Boy!')
```

通过上述代码，将字符串中的 World 修改为 Boy。执行后的效果如图 2-10 所示。

```
原来是： Hello World!
下面开始更新字符串： Hello Boy!
>>>
```

图 2-10　执行后的效果

2.6.4　转义字符

在 Python 程序中，当需要在字符中使用特殊字符时，需要用到用反斜杠“\”表示的转义字符。Python 中常用的转义字符的具体说明如表 2-1 所示。

表 2-1　Python 中常用的转义字符

转义字符	描　述
\(在行尾时)	续行符
\\	反斜杠符号
\'	单引号
\"	双引号
\a	响铃
\b	退格(Backspace)
\e	转义
\000	空
\n	换行
\v	纵向制表符
\t	横向制表符
\r	回车
\f	换页
\oyy	八进制数，yy 代表的字符，例如“\o12”代表换行
\xyy	十六进制数，yy 代表的字符，例如“\x0a”代表换行
\other	其他的字符以普通格式输出

有时我们并不想让上面的转义字符生效，而只是想显示字符串原来的意思，这时就要用 r 和 R 来定义原始字符串。如果想在字符串中输出反斜杠“\”，就需要使用“\\”实现。

实例 2-10：使用转义字符
源文件路径：daima\2\2-10

实例文件 zhuanyi.py 的具体实现代码如下所示。

```
print ("你们好\n 我们好") #普通换行
print ("来吧\\小宝贝")    #显示一个反斜杠
print ("我爱\'美女\'")    #显示单引号
print (r'\t\r')      # r 的功能是显示原始数据，也就是不用转义的
```

在上述代码中，第 1 行使用转义字符“\n”实现了换行，第 2 行使用转义字符“\\”显示一个反斜杠，第 3 行使用两个转义字符“\'”显示了两个单引号，第 4 行使用 r 显示了原始字符串，这个功能也可以使用 R 实现。执行后的效果如图 2-11 所示。

```
========
你们好
我们好
来吧\小宝贝
我爱'美女'
\t\r
>>>
```

图 2-11　执行后的效果

2.6.5　格式化字符串

Python 语言支持格式化字符串的输出功能，虽然这样可能会用到非常复杂的表达式，但是在大多数情况下，只需要将一个值插入一个字符串格式符“%”中即可。在 Python 程序中，字符串格式化的使用方式与 C 语言中的函数 sprintf 类似，常用的字符串格式化符号如表 2-2 所示。

表 2-2　Python 字符串格式化符号

符　号	描　述
%c	格式化字符及其 ASCII 码
%s	格式化字符串
%d	格式化整数
%u	格式化无符号整型
%o	格式化无符号八进制数
%x	格式化无符号十六进制数
%X	格式化无符号十六进制数(大写)
%f	格式化浮点数，可指定小数点后的精度
%e	用科学计数法格式化浮点数
%E	作用同%e，用科学计数法格式化浮点数
%g	%f 和%e 的简写
%G	%f 和%E 的简写
%p	用十六进制数格式化变量的地址

实例 2-11：格式化处理字符串
源文件路径：daima\2\2-11

实例文件 geshihua.py 的具体实现代码如下所示。

```
#%s 是格式化字符串
#%d 是格式化整数
print ("我的名字是%s，今年已经%d 岁了!" % ('西门吹雪', 33))
```

在上述代码中用到%s 和%d 两个格式化字符，执行后的效果如图 2-12 所示。

```
========
我的名字是西门吹雪，今年已经33岁了!
>>>
```

图 2-12　执行后的效果

2.7 数 字 类 型

在 Python 程序中，数字类型 Number 用于存储数值。数据类型是不允许改变的，这就意味着如果改变 Number 数据类型的值，将重新分配内存空间。从 Python 3 开始，只支持 int、float、bool、complex(复数)共计四种数字类型，删除了 Python 2 中的 Long(长整数)类型。

↑扫码看视频

2.7.1 整型 int

整型就是整数，包括正整数、负整数和零，不带小数点。在 Python 语言中，整数的取值范围是很大的。Python 中的整数还可以以几种不同的进制进行书写。0+“进制标志”+数字代表不同进制的数。现实中有如下 4 种常用的进制标志。

- 00[00]数字：表示八进制整数，例如：0024、0024。
- 0x[0X]数字：表示十六进制整数，例如：0x3F、0X3F。
- 0b[0B]数字：表示二进制整数，例如：0b101、0B101。
- 不带进制标志：表示十进制整数。

整型的最大功能是实现数学运算，例如下面的演示过程。

```
>>> 5 + 4        # 加法
9
>>> 4.3 - 2      # 减法
2.3
>>> 3 * 7        # 乘法
21
>>> 2 / 4        # 除法，得到一个浮点数
0.5
>>> 2 // 4       # 除法，得到一个整数
0
>>> 17 % 3       # 取余
2
>>> 2 ** 5       # 乘方
32
```

2.7.2 浮点型

浮点型 float 由整数部分与小数部分组成，浮点型也可以使用科学计数法表示(2.5e2 = $2.5 \times 10^2 = 250$)。整型在计算机中肯定是不够用的，因此就出现了浮点型数据，浮点数据用

来表示 Python 中的浮点数，浮点类型数据表示有小数部分的数字。当按照科学记数法表示时，一个浮点数的小数点位置是可变的，比如，1.23×10^9 和 12.3×10^8 是相等的。浮点数可以用数学写法，如 1.23，3.14，-9.01，等等。但是对于很大或很小的浮点数，就必须用科学计数法表示，把 10 用 e 替代，1.23×10^9 就是 1.23e9，或者 12.3e8，0.000012 可以写成 1.2e-5，等等。

整数和浮点数在计算机内部存储的方式是不同的，整数运算永远是精确的(除法也是精确的)，而浮点数运算则可能会有四舍五入的误差。更加详细地说，Python 语言的浮点数有如下两种表示形式。

(1) 十进制数形式：这种形式就是平常简单的浮点数，例如 5.12，512.0，.512。浮点数必须包含一个小数点，否则会被当成 int 类型处理。

(2) 科学计数法形式：例如 5.12e2(即 5.12×10^2)，5.12E2(也是 5.12×10^2)。必须指出的是，只有浮点类型的数值才可以使用科学计数形式表示。例如 51200 是一个 int 类型的值，而 512E2 则是浮点型的值。

2.7.3　布尔型

布尔类型是一种表示逻辑值的简单类型，它的值只能是真或假这两个值中的一个。布尔型是所有的诸如 a<b 这样的关系运算的返回类型。在 Python 语言中，布尔型的取值只有 True 和 False 两个，请注意大小写，分别用于表示逻辑上的“真”或“假”，其值分别是数字 1 和 0。布尔类型在 if、for 等控制语句的条件表达式中比较常见，例如 if 条件控制语句、while 循环控制语句、do 循环控制语句和 for 循环控制语句。

在 Python 程序中，可以直接用 True、False 表示布尔值(请注意大小写)，也可以通过布尔运算计算出来，例如：

```
>>> True
True
>>> False
False
>>> 3 > 2        #数字 3 确实大于数字 2
True
>>> 3 > 5        #数字 3 大于数字 5?
False
```

布尔值可以用 and、or 和 not 进行运算。其中 and 运算是与运算，只有所有值都为 True，and 运算结果才是 True，例如下面的演示过程。

```
>>> True and True       #两个都为 True
True
>>> True and False      #一个是 True，一个是 False
False
>>> False and False     #两个都是 False
False
```

而 or 运算是或运算，只要其中有一个值为 True，or 运算结果就是 True，例如：

```
>>> True or True            #两个都为 True
True
```

```
>>> True or False              #一个是 True，一个是 False
True
>>> False or False             #两个都是 False
False
```

而 not 运算是非运算，它是一个单目运算符，把 True 变成 False，False 变成 True，例如：

```
>>> not True
False
>>> not False
True
```

在 Python 程序中，布尔值经常被用在条件判断应用中，例如：

```
age=12;                        #设置 age 的值是 12
if age >= 18:
    print("adult")             #如果 age 的值大于等于 18 则打印输出 adult
else:
    print ("teenager")         #如果 age 的值不大于等于 18 则打印输出 teenager
```

2.7.4 复数型

在 Python 程序中，复数型即 complex 型，由实数部分和虚数部分构成，可以用 a + bj 或者 complex(a,b)表示，复数的实部 a 和虚部 b 都是浮点型。表 2-3 演示了 int 型、float 型和 complex 型的对比。

表 2-3 int 型、float 型和 complex 型的对比

int 型	float 型	complex 型
10	0.0	3.14j
100	15.20	45.j
-786	-21.9	9.322e-36j
080	32.3+e18	.876j
-0490	-90.	-.6545+0j
-0x260	-32.54e100	3e+26j
0x69	70.2-e12	4.53e-7j

在 Python 程序中，使用内置的函数 type()可以查询变量所属的对象类型。

实例 2-12：获取并显示各个变量的类型
源文件路径：daima\2\2-12

实例文件 leixing.py 的具体实现代码如下所示。

```
#注意下面的代码中的赋值方式
#将 a 赋值为整数 20
#将 b 赋值为浮点数 5.5
#将 c 赋值为布尔数 True
#将 d 赋值为复数 4+3j
a, b, c, d = 20, 5.5, True, 4+3j
```

```
print(type(a), type(b), type(c), type(d))
```

执行代码后将分别显示 4 个变量 a、b、c、d 的数据类型，执行效果如图 2-13 所示。

```
========
<class 'int'> <class 'float'> <class 'bool'> <class 'complex'>
>>>
```

图 2-13　执行效果

智慧锦囊

- Python 可以同时为多个变量赋值，例如 a, b = 1, 2，表示 a 的值是 1，b 的值是 2。
- 一个变量可以通过赋值指向不同类型的对象。
- 数值的除法“/”总是返回一个浮点数，要想获取整数，需要使用“//”操作符。
- 在进行混合计算时，Python 会把整型转换成为浮点数。

2.8　实践案例与上机指导

通过本章的学习，读者基本可以掌握 Python 语言的基础语法知识。其实 Python 语言的基础语法知识还有很多，这需要读者通过课外渠道来加深学习。下面通过练习操作，以达到巩固学习、拓展提高的目的。

↑扫码看视频

2.8.1　多个变量同时进行赋值

Python 语言支持对多个变量同时进行赋值，请看下面的实例。

实例 2-13：同时赋值多个变量
源文件路径：daima\2\2-13

实例文件 tongshi.py 的具体实现代码如下所示。

```
a = (1,2,3)                          #定义一个元组
x,y,z = a                            #把序列的值分别赋予 x、y、z
print("a : %d, b: %d, z:%d"%(x,y,z)) #打印结果
```

```
========
a : 1, b: 2, z:3
>>>
```

图 2-14　执行效果

在上述代码中，对变量 x、y、z 同时进行了赋值，最后分别输出了变量 a、b、z 的值。执行效果如图 2-14 所示。

2.8.2 使用字符串处理函数

Python 语言中提供了很多对字符串进行操作的函数，其中最为常用的字符串处理函数如表 2-4 所示。

表 2-4 常用的字符串处理函数

字符串处理函数	描 述
string.capitalize()	将字符串的第一个字母大写
string.count()	获得字符串中某一子字符串的数目
string.find()	获得字符串中某一子字符串的起始位置，无则返回-1
string.isalnum()	检测字符串是否仅包含 0～9，A～Z，a～z
string.isalpha()	检测字符串是否仅包含 A～Z，a～z
string.isdigit()	检测字符串是否仅包含数字
string.islower()	检测字符串是否均为小写字母
string.isspace()	检测字符串中所有字符是否均为空白字符
string.istitle()	检测字符串中的单词是否为首字母大写
string.isupper()	检测字符串是否均为大写字母
string.join()	连接字符串
string.lower()	将字符串全部转换为小写
string.split()	分割字符串
string.swapcase()	将字符串中的大写字母转换为小写，小写字母转换为大写
string.title()	将字符串中的单词首字母大写
string.upper()	将字符串中的全部字母转换为大写
len(string)	获取字符串长度

实例 2-14：使用字符串处理函数
源文件路径：daima\2\2-14

实例文件 hanshu.py 的具体实现代码如下所示。

```
mystr = 'I love you!.'                          #定义的原始字符串
print('source string is:',mystr)                #显示原始字符串
print('swapcase demo\t',mystr.swapcase())       #大小写字母转换
print('upper demo\t',mystr.upper())             #全部转换为大写
print('lower demo\t',mystr.lower())             #全部转换为小写
print('title demo\t',mystr.title())             #将字符串中的单词首字母大写
print('istitle demo\t',mystr.istitle())         #检测是否为首字母大写
print('islower demo\t',mystr.islower())         #检测字符串是否均为小写字母
print('capitalize demo\t',mystr.capitalize())   #将字符串的第一个字母大写
print('find demo\t',mystr.find('u'))            #获得字符串中字符"u"的起始位置
print('count demo\t',mystr.count('a'))          #获得字符串中字符"a"的数目
```

```
print('split demo\t',mystr.split(' '))        #使用单引号分隔字符串，以空格为界
print('join demo\t',' '.join('abcde'))        #连接字符串
print('len demo\t',len(mystr))                #获取字符串长度
```

在上述代码中，从第 3 行开始，每行都调用了一个字符串处理函数，并打印输出了处理结果。执行效果如图 2-15 所示。

```
-------
source string is: I love you!.
swapcase demo     i LOVE YOU!.
upper demo        I LOVE YOU!.
lower demo        i love you!.
title demo        I Love You!.
istitle demo      False
islower demo      False
capitalize demo   I love you!.
find demo         9
count demo        0
split demo        ['I', 'love', 'you!.']
join demo         a b c d e
len demo          12
>>>
```

图 2-15　执行效果

2.9　思考与练习

本章详细讲解了 Python 语言的基础语法知识，循序渐进地讲解了缩进、注释、标识符和关键字、变量、输入和输出、字符串和数字类型等内容。在讲解过程中，通过具体实例介绍了使用这些知识点的方法。通过本章的学习，读者应该了解 Python 语言的基础语法的知识，掌握它们的使用方法和技巧。

一、选择题

(1) 声明全局变量的关键字是(　　)。

A. function　　B. global　　C. all

(2) 下面不是布尔型值的是(　　)。

A. True　　B. False　　C. 2

二、判断对错

(1) 局部变量只有在局部变量被创建的函数内有效。(　　)

(2) 布尔类型是一种表示逻辑值的简单类型，它的值只能是真或假这两个值中的一个。(　　)

三、上机练习

(1) 编写一个程序，展示不同进制整数在 Python 中的使用。

(2) 编写一个程序，要求在变量中使用数学分隔符。

新起点电脑教程

第 3 章

运算符和表达式

本章要点

- 运算符和表达式
- 算术运算符和算术表达式
- 比较运算符和比较表达式
- 赋值运算符和赋值表达式
- 位运算符和位表达式
- 逻辑运算符和逻辑表达式
- 成员运算符和成员表达式

本章主要内容

在 Python 语言中，即使有了变量和字符串，也不能进行日常的程序处理工作，还必须使用某种方式将变量、字符串的关系表示出来，此时运算符和表达式便应运而生。运算符和表达式的作用是，为变量建立一种组合联系，实现对变量的处理，以实现现实中某个项目需求的某一个具体功能。本章将详细介绍 Python 语言中运算符和表达式的基本知识，为读者步入本书后面知识的学习打下坚实的基础。