第5章 chapter 5

Python 语言基础

Python 被以一种简洁易读的方式设计，可以用少量的代码高效率地编写程序。Python 能够在 Windows、MacOS 和 Linux/UNIX 等系统兼容运行，同时支持广泛的应用程序开发，在 Web 开发、游戏制作、嵌入式开发、数据分析和深度学习等方面都得到了广泛应用，成为一大主流计算机语言。

5.1 Python 简介及其开发环境

5.1.1 Python 简介

Python 由荷兰人吉多·范罗苏姆(Guido van Rossum)开发，1991 年第一个版本公开发行，2002 年发布了 2.0 系列，2008 年发布了 3.0 系列版本，2020 年结束对 Python 2.0 版本的支持。

Python 语言具有如下特点。

(1) 简单易学。Python 有相对较少的关键字，结构简单，具有明确定义的语法，学习起来更加简单。

(2) 免费、开源。Python 是自由/开放源码软件之一，源代码被公开在网络上，任何人都可以无偿使用，同时使用者可以自由地发布这个软件的副本，阅读它的源代码并对它进行修改。

(3) 易于阅读。Python 代码具有更清晰的定义形式，具有伪代码的特质，可以让开发者在开发 Python 程序时专注于解决问题，而不是纠结于语言本身。

(4) 解释性。编译型语言(如 C、C++)在执行时需要经过编译，生成机器码后才能执行。Python 直接由解释器执行，但是这并不意味着 Python 丢掉了编译，实际上 Python 是字节编译的，可以生成一种近似机器语言的中间形式。因为纯粹的解释型语言通常比编译型语言运行慢，Python 通过这种方式不仅改善了性能，还同时保持了解释型语言的优点。之所以说 Python 是解释型语言，只是其在开发过程中没有显式地调用编译操作，表现出更多解释型的特性，事实上，编译是存在的。

(5) 模块自信。Python 的强大体现在“模块自信”上，因为 Python 不仅有很强大的自有模块，还有海量的第三方模块，并且很多开发者还在不断贡献自己开发的新模块。

(6) 面向对象。所谓面向对象是指将程序功能模块化,并通过调用这些模块来实现最终的程序功能的思维方式。通过这种方式,各功能模块的独立性得到了保障,从而提高开发效率和模块再利用率,增强程序的可维护性和稳定性。Python 设计之初就已经是一门面向对象语言,可以进行面向对象编程。

5.1.2 Python 环境搭建

本书主要以 Windows 系统为例讲解 Python 环境的搭建和 Python 程序的运行。在环境搭建之前,首先打开命令行终端窗口输入 python,按 Enter 键查看自己的计算机是否具备 Python 运行环境。如果出现类似图 5.1 的界面,可以显示 Python 的版本号,则说明具备 Python 环境,如果没有,则需要进行环境搭建。

图 5.1 Python 命令行窗口

用 Python 语言进行程序开发,既可选择 Python 自带的集成开发环境(IDLE),也可以选择使用 PyCharm、Notepad++等作为开发环境。本书选择 Python 自带的 IDLE 作为开发环境,所有示例使用 Python 3.7.5 进行程序的开发和演示。

1. Python 解释器安装

计算机执行 Python 程序时,需要将 Python 代码翻译为计算机可识别的机器指令语言,做翻译工作的是 Python 解释器。可以在 Python 官网下载 Python 解释器。

安装 Python 解释器需要如下几个步骤。

(1) Python 下载。打开 Python 官网主页 https://www.python.org/,根据自己的操作系统版本选择相应的 Python 版本,下载即可。本书以 64 位 Python 3.7.5 版本的可执行安装程序进行安装演示。

(2) 在 Windows 平台安装 Python。双击下载包,进入 Python 安装向导,如图 5.2 所示。首先建议选择第二项自定义安装(Customize installation),可以自行设定安装的路径;然后选择最下面的 Add Python 3.7 to PATH 选项。此选项的功能是把 Python 的安装路径添加到系统路径下面,选中这个选项,安装后直接在命令行终端窗口输入 python,按 Enter 键就可直接调用 python.exe,避免安装后还要手动配置 Python 的环境变量。

安装好 Python 后,可以打开命令行终端窗口输入 python,按 Enter 键,查看自己的计算机是否已经具备 Python 运行环境。

图 5.2 Python 安装向导

2. Python 的开发环境

Python 解释器安装包将在系统中安装一组与 Python 开发和运行相关的程序，其中最重要的两个是 Python 命令行和 Python 集成开发环境(IDLE)。

在 IDLE 下运行 Python 程序有两种方式：交互式和文件式。交互式指 Python 解释器即时响应用户输入的每条代码，给出输出结果，这种方式适合单条语法的练习。文件式是指用户将 Python 程序写在一个或多个文件中，然后启动 Python 解释器批量执行文件中的代码，文件式是编程的主要方式。

1) 启动 IDLE

安装 Python 后，可以从“开始”菜单→“所有程序”→Python 3.7→IDLE，来启动 IDLE。

启动 IDLE 后，即可打开 Python Shell 窗口，通过它可以在交互式模式下执行 Python 命令。

2) IDLE 交互式运行方法

在交互式界面可以进行简单的交互式程序的输入和执行，如图 5.3 所示。

说明：>>>是交互式下的提示符，表示可以在它后面输入要执行的语句。本书中示例带有>>>符号的代码是指在 IDLE 的交互式环境下运行的代码，不带此提示符的代码表示是以文件方式运行的。

3) IDLE 文件式运行方法

打开 IDLE，按快捷键 Ctrl+N 打开一个新窗口，或者在菜单中选择 File→New File 命令，打开新窗口。这个窗口可以进行代码编辑，在其中输入 Python 代码，保存为.py 文件，如图 5.4 所示。按快捷键 F5 或者在菜单中选择 Run→Run Module 命令，运行该程序，程序运行结果显示在交互式界面，如图 5.5 所示。

IDLE 是一个简单有效的集成开发环境，无论交互式还是文件式，都有助于快速编写

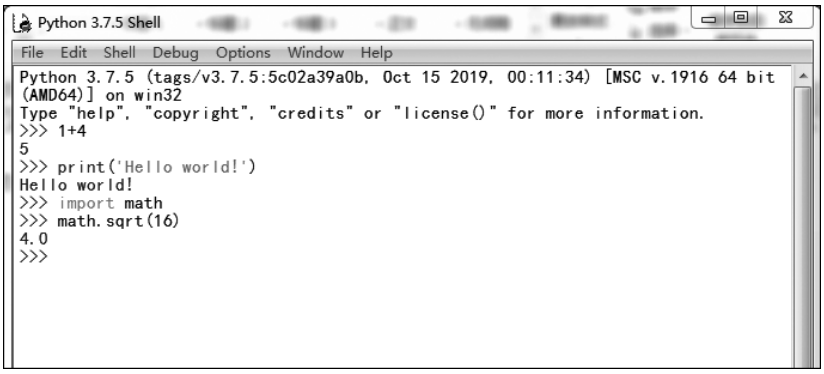

图 5.3 IDLE 交互式界面

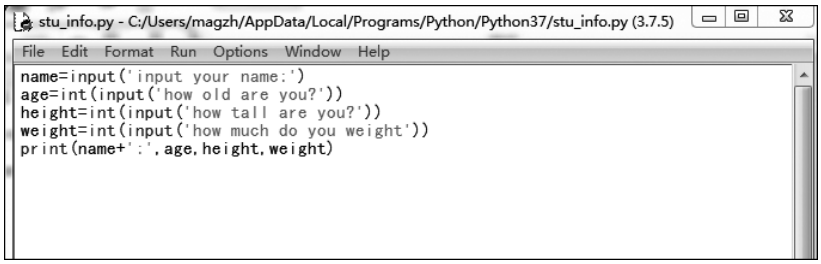

图 5.4 IDLE 文件式界面

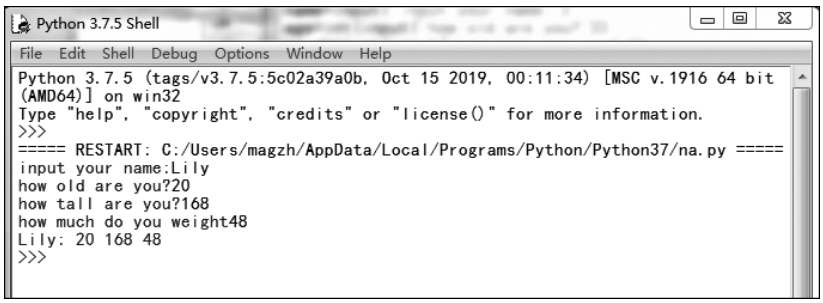

图 5.5 程序运行结果显示界面

和调试代码，是小规模 Python 软件项目的主要编写工具，可以实现语法加亮、段落缩进、基本文本编辑、Tab 键控制和调试程序等基本功能。

5.2 Python 语言基础知识

5.2.1 Python 的程序要素

1. 标识符

在 Python 中会遇到很多名称，如变量名、函数名和模块名等，这些名称从技术上来

说都称为标识符。在Python中,标识符的构成需要满足如下规则。

(1) 每个标识符必须以字母或下画线("_")开头,后跟字母、汉字、数字或下画线的任意组合。根据该规则,x、age、stu_id、_x3都是Python中合法的标识符,而3x因以数字开头,不是合法的标识符。

(2) 标识符区分大小写。对于Python来说,stu_id、Stu_id、Stu_ID是不同的名称。

(3) 自定义的名称不能与Python的"保留字"相同。Python 3.x版本有33个保留字,如表5.1所示。Python的保留字与其他标识符一样,也对大小写敏感。例如,def是保留字,但是Def不是,Python允许将Def定义为变量名使用。

表5.1 Python 3.x的保留字列表

保 留 字			
False	def	if	raise
None	del	import	return
True	elif	in	try
and	else	is	while
as	except	lambda	with
assert	finally	nonlocal	yield
break	for	not	
class	from	or	
continue	global	pass	

在大多数情况下,编程者可以自由选择符合规则的任何名称,但是需要尽量做到选择的名称能够描述被命名的事物。

另外,Python还有相当多的内置函数,例如我们会经常用到的print函数,虽然在技术上可以将这些函数名称标识符用于其他目的,但是一旦将print重新定义,它就无法打印信息,也会给其他读到代码的用户带来困扰,所以要尽量避免。

2. 表达式

产生或计算新数据值的程序代码片段称为表达式,它是编程语言中最基本的程序结构。

表达式包含值和操作符,并且总是可以求值。在Python Shell中输入表达式时,Shell会计算表达式并打印出结果。

```
>>>2+2
4
```

没有操作符的单个值也认为是一个表达式,求值的结果就是它自身。

```
>>>2
```

```
2
>>> "hello"
'hello'
```

一个变量也可作为表达式。例如下面语句，首先变量 x 被赋值为 5，然后要求 Python 对表达式 x 求值，作为响应，Python Shell 打印出 x 的值 5。

```
>>>x=5
>>>x
5
```

多个简单的表达式和操作符可以组合成复杂的表达式。对于数字，Python 提供了一组标准的数学运算。表 5.2 列出了 Python 的所有数学操作符。

表 5.2　数学操作符

操　作　符	操　　作	例　　子	值
**	指数	3**3	27
%	取余	10%3	1
//	整除	10//3	3
/	除法	10/4	2.5
*	乘法	3*3	9
－	减法	5－3	2
＋	加法	5＋3	8

以下是两个复杂表达式的示例。

```
3.9*5+(100-36)/4
((x1-x2)/2*n)+(y/k**3)
```

3. 给变量赋值

如果要保存一个表达式的值并方便以后引用，可以将其赋值给一个变量。变量的命名规则要符合标识符的命名规则。

创建一个变量很简单，只需要取一个名字，然后给它赋予一个值，赋值时不需要指定变量的数据类型。

变量赋值包括 3 种形式：简单赋值、赋值输入和同时赋值。

1）简单赋值

简单赋值语句具有以下形式：

```
<variable>=<expr>
```

这里 variable 是一个标识符，也称为变量名，expr 是一个表达式。赋值的语义是：求解右侧表达式的值，然后将该值与左侧命名的变量相关联。

在交互式环境下，赋值语句示例如下。

```
>>>age=20
>>>x=5
>>>x=x+3
>>> name="Lily"
```

给变量第一次赋值可称为对变量初始化，例如上例中 x＝5，即把变量 x 的初始值设置为 5。此后可以在表达式中引用变量的值，如上例中的 x＝x＋3 就是引用了 x 的当前值，计算的结果重新赋值给变量 x。

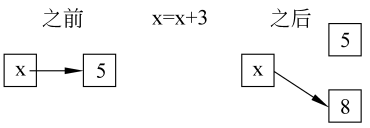

图 5.6 变量 x 赋值展示效果图

Python 中对变量 x 赋值就像把一个便利贴放在值上，并注明“这是 x”。当对变量重新赋值时，变量只需切换到引用新值。图 5.6 给出上面示例中给变量 x 两次赋值的展示效果。

如果一个值不再被任何变量引用，Python 会自动从内存中将其清除，以便腾出空间用于存放其他新值。

2）赋值输入

输入语句的目的是从用户那里获取一些信息，并存储到变量中。在 Python 中，输入是用一个赋值语句结合一个内置函数 input 实现的。

输入语句的形式取决于你希望从用户那里获取的数据类型，对于文本类型的数据，语句形式如下。

```
<variable>=input(<prompt>)
```

这里的 prompt 是一个字符串表达式，用于提示用户输入，示例如下。

```
>>> name=input("Enter your name:")
Enter your name:Lily
>>> name
'Lily'
```

计算机执行上面的第一条语句时，打印出提示语“Enter your name:”，然后解释器暂停，等待用户输入，用户输入字符串'Lily'后，该值被赋给变量 name。

如果用户希望输入的是一个数字，则需要将 input 输入的文本数据通过 eval 或 int、float 函数转换为数字，语句形式如下。

```
<variable>=eval(input(<prompt>))        #去掉外层引号，执行表达式
<variable>=int(input(<prompt>))         #将数字字符串转换为整数
<variable>=float(input(<prompt>))       #将数字字符串转换为浮点数
```

示例如下。

```
>>> input('Enter your age:')
Enter your age:21
```

```
'21'
>>> eval(input('Enter your age:'))
Enter your age:21
21
>>> int(input('Enter your score:'))
Enter your score:89
89
>>> float(input('Enter your score:'))
Enter your score:89.6
89.6
```

在 input 语句中使用 int 而不是 eval，可以确保用户只能输入有效的整数。

3）同时赋值

Python 中有一个赋值语句的替代形式，允许同时计算几个值。例如：

```
sum, diff=x+y, x-y
```

这种形式称为“同时赋值”。语义上，告诉 Python 首先对右侧的所有表达式求值，然后将这些值赋给左侧相应的变量，这里 sum 得到 x 和 y 的和，diff 得到 x 和 y 的差，上面的语句等价于以下两个语句：

```
sum=x+y
diff=x-y
```

这种形式使得在 Python 中交换两个变量的值变得非常容易，可以通过下面的语句实现：

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

4. 输出语句

Python 中使用内置函数 print 在屏幕上打印信息。

使用 print 语句打印表达式的值，所有提供的表达式都从左到右求值，然后将结果值以从左到右的方式显示在输出行，示例如下。

```
>>> print(2+3)
5
>>> print(2,3,2+3)
2 3 5
>>> print('2 和 3 的和为',2+3)
2 和 3 的和为 5
>>> x=4
>>> y=5
>>> print(x,y)
4 5
```

Python 中 print()函数默认以换行符结尾，即执行一次 print()，则自动换行。以下代码是通过 for 循环输出列表元素。

```
ls=[1,2,3,4,5]
for i in ls:
    print(i)
```

程序运行结果如下。

```
1
2
3
4
5
```

实际应用中如果有特殊要求，例如希望列表元素在一行输出，Python3 的 print()函数增加了一个 end 参数，可通过该参数调整输出格式。

上面代码可以改写如下。

```
ls=[1,2,3,4,5]
for i in ls:
    print(i,end=' ')
```

end=''表示 print 语句结束时以空格结尾，这段代码的执行结果如下。

```
1 2 3 4 5
```

关于输出语句的其他用法，例如格式化输出，会在后续的学习中逐步学到。

5. 缩进

Python 与其他编程语言最大的不同是用缩进来区分代码块，缩进的空格数量不限，但是同一代码块缩进必须一致，一般缩进以一个 Tab 键为准。

Python 的分支结构中程序缩进示例如下。

```
if True:
    print('True')                    #缩进
else:
    print('False')                   #缩进
```

6. 注释

注释起到解释或说明的作用，一般写在程序的开头或语句的后面，在程序运行时，Python 会忽略注释。

Python 中单行注释以＃开头，后面的文字直到行尾都是注释，示例如下。

```
print("Hello world1!")               #这是注释
```

多行注释可以使用多个＃号，也可用 3 个单引号或 3 个双引号，示例如下。

```
#这是注释
```

```
#print('Hello world1!')
#print('Hello world2!')
print('Hello world3!')
'''
print('Hello world4!')
print('Hello world5!')
print('Hello world6!')
'''
```

上面多行注释示例代码运行的结果如下。

```
Hello world3!
```

前面的 2 个和后面的 3 个 print 语句因为被注释，所以跳过不执行。

5.2.2 Python 的数据类型

表达式是值和操作符的组合，每个表达式都可以通过求值得到某个值，而每个值都属于一种数据类型。Python 中通过对变量赋值访问不同数据类型的对象。

Python 中基本的数据类型有数字类型、字符串类型、列表、元组、字典和集合等，可以使用 type()函数查看变量和常量的数据类型，示例如下。

```
>>>a=25
>>>print(type(a))
<class 'int'>
>>> b,c,d=2.5,'HELLO',True
>>> print(type(b),type(c),type(d))
<class 'float'><class 'str'><class 'bool'>
```

其中，<class 'int'>表示数据是整数类型，<class 'float'><class 'str'><class 'bool'>分别是浮点数、字符串和布尔类型。

1. 数字类型

表示数字或数值的数据为数字类型，Python 中提供 3 种数字类型：整数(int)、浮点数(float)和复数(complex)，分别对应数学中的整数、实数和复数。

1) 整数类型

整数类型与数学中的整数一致，共有 4 种进制表示：十进制、二进制、八进制和十六进制，默认情况下采用十进制。整数类型的 4 种进制如表 5.3 所示。

表 5.3 整数类型的 4 种进制

进　制	引导符号	示　例
十进制	无	由字符 0～9 组成，如 1010，79
二进制	0b 或 0B	由字符 0～1 组成，如 0b1010，0B101