第1章

基础知识

目前虽然专门从事软件开发的人员很多，教学、科研、商务中的许多业务，可以委托这些专业人员去完成，但有时受限于客观条件，需要我们自己处理和分析学习和工作中的数据。使用本行业专业化的软件虽然能够满足要求，但这些软件庞大而昂贵；用大众化软件如Excel却又不能满足要求，此时非软件开发人员就需要小露一手，自己编写程序。

C++、Java等编程语言虽然功能强大，但对于非软件专业的人士，编写代码的工作量很大，在较短的时间内上手有一定的困难。Matlab虽然在科学计算、数据分析、通信、机器学习、图像处理等方面提供了简单易用的工具包，但软件价格昂贵。Python是由Guido van Rossum在1989年年底出于某种娱乐目的而开发的，其语言基础是ABC。ABC语言功能强大，专门为非专业程序员而设计，因而Python上手容易，学习成本低。

1.1 软件的安装

登录网站https://www.Python.org，根据个人计算机操作系统选择下载并安装Python。Python有2.X和3.X版本，考虑到2.X版本将来不再更新，建议安装3.X版本。安装成功后，Python自带一个集成开发环境IDLE。要注意，低于Python 3.7的版本不能直接识别中文命名的文件和文件夹。除Python的IDLE外，还可以安装其他编程器，如Anaconda、Pycharm等。Anaconda是一个包含180多个科学包及其依赖项的发行版本，下载的网址https://www.anaconda.com/download/。Anaconda自带有conda、NumPy、SciPy、ipython notebook等。Pycharm软件下载的网址https://www.jetbrains.com/pycharm/，安装完成后，如果计算机上安装了不同版本的Python，需要为Pycharm配置Python版本及库文件。方法是单击Pycharm，选择File→Settings→Project Interpreter菜单命令，设置指定版本的Python。

Pycharm是Python最专业的编程软件，但运行时计算机耗费的资源较大。本书仅以Windows下Python自带的IDLE(Integrated Development and Learning Environment，集成开发与学习环境)为例，介绍Python。

1.2 管理Python相关的扩展库

要安装Python的库文件，需要以管理员的身份在Windows的命令提示符窗口中执行pip命令。pip命令在Python软件安装的文件夹中，如D:\Python\Python3.7.1\Scripts。另

外，更新 pip 命令时需要使用 Python.exe，该命令在 Python 安装的文件夹中，命令窗口中每次运行这些命令时都需要转到相应的文件夹下，比较麻烦。简单的方法是在 Windows 中设置环境变量。下面以 Python 安装在 D:\Python\Python3.7.1 为例，说明环境变量的设置。

在 Windows 桌面右击“此电脑”，依次选择“属性”→“高级系统设置”→“环境变量”，查看在 administrator 的用户变量下有无 Path 变量，如果没有就单击“新建”按钮；否则单击“编辑”按钮。“新建”时在“变量名”中输入 Path、“变量值”中输入 D:\Python\Python3.7.1\Scripts; D:\Python\Python3.7.1\。编辑 Path 变量时，在已经存在的变量值后增加 D:\Python\Python3.7.1\Scripts; D:\Python\Python3.7.1\，变量值间用半角分号(;)分隔。

在 Windows 的搜索框中，输入 cmd，出现命令提示符，右击，选择“以管理员身份运行”，进入“命令提示符”窗口。输入命令 pip list，将列出已经安装的库。如果有新版本的 pip 命令，执行时会出现 WARNING: You are using pip version 19.3.1; however, version 20.0.2 is available.You should consider upgrading via the 'Python -m pip install --upgrade pip' command 的提示，要求输入：Python -m pip install --upgrade pip 完成更新。

(1) 联网情况下。在 Windows 命令提示符下执行 pip，可以完成 Python 扩展库的安装、升级、卸载等。下面以 NumPy 为例说明。

安装 NumPy：

```
pip install numpy
```

升级 NumPy：

```
pip install --upgrade numpy
```

卸载 NumPy：

```
pip uninstall numpy
```

在某些情况下，必须安装指定的版本才能保证各模块间相互协调，如 aircv.1.4.6 须安装 OpenCV 3.4.2.16 才能使用，可用 pip install opencv-python==3.4.2.16 指定 OpenCV 的版本。

由于 OpenCV 3.x 将 SIFT 等算法整合到 xfeatures2d 集合，而 xfeatures2d 在 opencv-contrib 中，故在 OpenCV 中要使用 SIFT 等算法，必须用 pip install opencv-contrib-Python 安装 OpenCV。

(2) 在联网的情况下，先从网站 https://www.lfd.uci.edu/~gohlke/Pythonlibs/上将需要安装库的 wdl 下载到当地某个文件夹。图 1-1 是下载 OpenCV 的界面，根据自己 Windows 系统，选择下载的文件。下载的链接中 amd64 表示 64 位，win 表示 Windows 系统，4.2.0 表示版本号。

以下载 opencv_Python4.2.0cp38cp38win_amd64.whl 到 c:\Python_wdl 文件夹为例，安装时需要在 Windows 的“命令提示符”窗口中执行：

```
pip install c:\Python_wdl\opencv_Python4.2.0cp38cp38win_amd64.whl
```

有时从第三方网站库上下载的安装文件中有 setup.py，安装过程如下。

① 解压下载的文件。

opencv_python-4.2.0-cp38-cp38-win_amd64.whl
opencv_python-4.2.0-cp38-cp38-win32.whl
opencv_python-4.2.0-cp37-cp37m-win_amd64.whl
opencv_python-4.2.0-cp37-cp37m-win32.whl
opencv_python-4.2.0-cp36-cp36m-win_amd64.whl
opencv_python-4.2.0-cp36-cp36m-win32.whl
opencv_python-4.1.2-cp38-cp38-win_amd64.whl
opencv_python-4.1.2-cp38-cp38-win32.whl
opencv_python-4.1.2-cp37-cp37m-win_amd64.whl
opencv_python-4.1.2-cp37-cp37m-win32.whl
opencv_python-4.1.2-cp36-cp36m-win_amd64.whl
opencv_python-4.1.2-cp36-cp36m-win32.whl
opencv_python-4.1.2-cp35-cp35m-win_amd64.whl

图 1-1　下载 OpenCV 不同版本库

② 在 Windows 的“命令提示符”窗口中，通过 cd 命令进入有 setup.py 的文件夹。

③ 执行 Python setup.py build 命令。

④ 执行 Python setup.py install 命令。

某些情况下如网络不稳定，在线通过 pip install 安装库文件时会出现 request time out 提示，表示安装不成功，可以通过在线下载安装文件，然后以离线方式安装下载的库文件，一般都能安装成功。

1.3　使用 IDLE

安装 Python 后 IDLE 会自动安装，非商业开发 IDLE 即可满足要求。该环境下的交互方式，可直接测试一些程序片段，在图 1-2 中，“>>>”是 IDLE 的命令提示符，由系统自动提供。

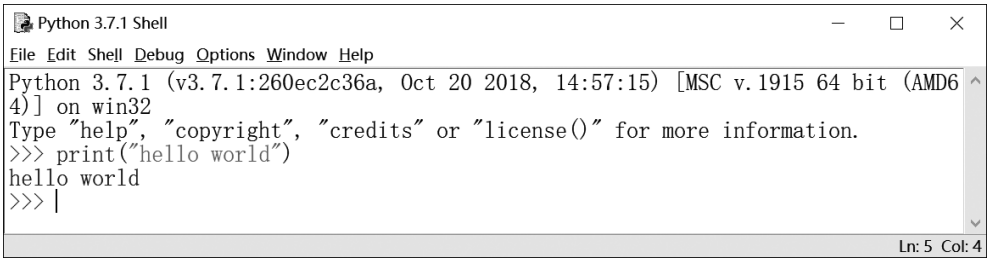

图 1-2　交互式输入代码

判断某个库(如 NumPy)是否安装成功，可在图 1-2 中输入 import numpy as np 命令，如果安装不成功或者该库没有安装，IDLE 会出现运行错误的提示。

图 1-2 中选择 File→New File 菜单命令，按图 1-3 所示编辑 Python 代码，将文件保存为 hello.py，执行 Run→Run Module 菜单命令，运行结果如图 1-4 所示。

每个 Python 的脚本在运行时都有一个__name__(name 左右是两条下画线)属性。如

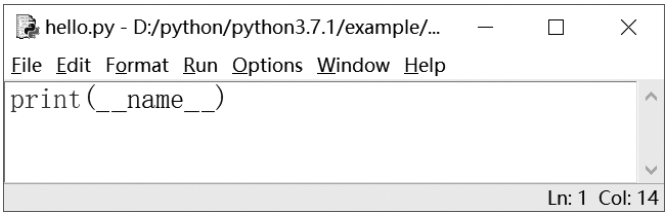

图 1-3　在 IDLE 中编写 Python 程序代码

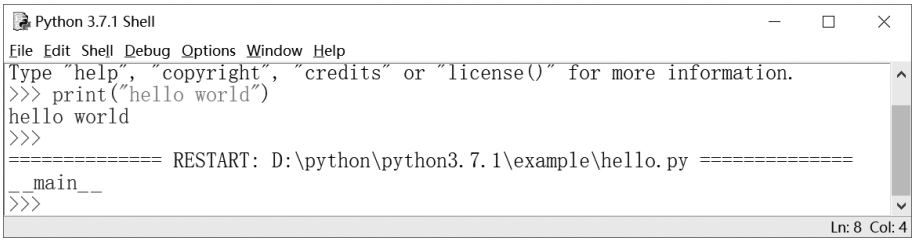

图 1-4　运行脚本输出__name__属性

果脚本直接独立地运行，__name__的属性返回值为__main__；如果脚本是作为模块被导入，则其属性被设置为模块名。交互环境中输入 import hello 命令，导入 hello.py 模块，输入结果是 hello。利用__name__属性的这一特性，可控制 Python 程序的运行方式，如果希望 hello.py 程序只能以模块导入的方式运行，可以将其代码更改为：

```
if __name__=="__main__":
    print("本程序只能以模块导入的方式运行")
else:
    print(__name__)
```

代码中 if...else...是判断语句。注意其语句后面必须有冒号(:)。Python 语句块是按照空格识别的，IDLE 默认的空格是 4 个，输入 if __name__==="__main__":后回车，下一条语句自动缩进 4 个空格后输入下一条语句。如果 if 下有多条语句，缩进的空格必须相同，除非里面又嵌套了别的判断、循环等语句块。Python 编程时代码不能随意缩格，空格有特殊的意义，这点与 C、C#、Java 等语言大不相同！代码中==是比较运算符，比较其左右内容是否相同，不同于赋值运算符=。

有时 Python 程序在命令行中运行，运行时有可能输入参数，此时需要用到 sys.argv。将以下代码输入后保存为 argv.py。

```
import sys
if __name__=='__main__':
    print("输入的参数有{}个，它们是：{}".format(len(sys.argv),sys.argv))
else:
print("不能以模块方式运行")
```

代码中 format()函数是字符串格式化输出的函数，{}是点位符，分别用 format()中指定的数据填入到对应的点位符中，len()用于测试字符串、列表、元组等的长度。

在 Windows 的 command 命令行中，将当前文件夹转到 argv.py 所在的文件夹，然后按

图 1-5 所示分别运行以下 3 条命令，观察输出的结果。

```
Python argv.py
Python argv.py "x"
Python argv.py "x" "y"
```

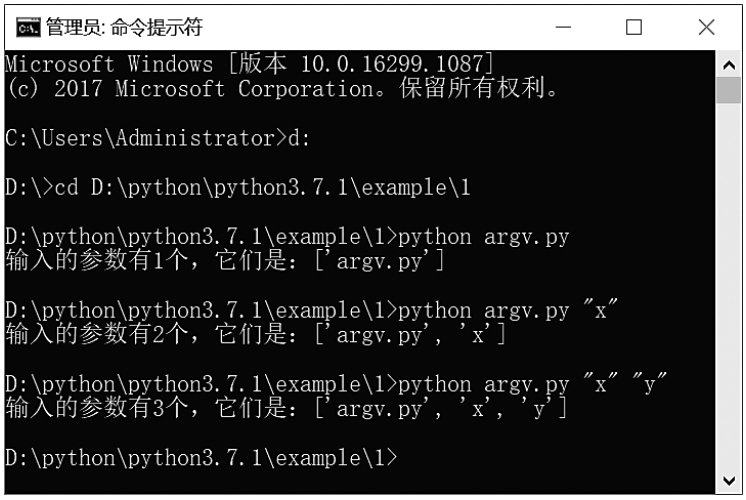

图 1-5　sys.argv 应用示例

图 1-5 中，sys.argv 指的是命令窗口中 Python 命令后的参数列表，如命令窗口中运行：python argv.py "x"，其中 sys.argv 是指['argv.py', 'x']，要想得到'x'，可以取 sys.argv[1:]（:是切片）。

1.4 模　　块

每个.py 文件本身就是一个模块，要使用该文件中的功能，只需要在文件中导入该.py 文件就可以了。

1.4.1 将整个模块导入

格式：

```
import 模块名 [as 别名]
```

导入后需要在要使用的对象前加上前缀，即以“模块名.对象名”的方式访问，如：

```
>>>import math
>>>math.sin(0.5)
```

有时导入的模块名太长书写不方便，可以使用 as 简化，如：

```
>>>import numpy as np
>>>a=np.array((1,2,3,4,5))
```

1.4.2 从某个模块中导入某个函数

格式：

```
from 模块名 import 对象名 [as 别名]
```

此种方式下导入模块中的函数后，使用函数时不能再加模块名，如 math.sin(30)：

```
>>>from math import sin
>>>sin(0.5)
```

从某个模块中导入多个函数的格式为 from module_name import func1, func2, func3，如：

```
>>>from math import sin,cos
>>>sin(30)+cos(30)
```

将某个模块中的全部函数导入的格式为 from module_name import *，如开一个 GUI 窗口：

```
>>>from tkinter import *
>>>win=Tk()
>>>win.mainloop()
```

1.4.3 使用软件包管理模块

当软件较大时，一般由多人编写，如果甲和乙两人编写的文件都命名为 hello.py，需要将两个 hello.py 放在不同的文件夹中。即使设置了搜索路径，使用 import hello 也只能找到一个 hello.py，解决方法是建立两个文件夹（如 a 和 b），将两个 hello.py 分别放在两个文件夹中，并且 a 和 b 每个文件夹中再放置一个__init__.py 文件（init 前后各两条下画线），表明该文件夹是一个软件包，暂时可先保持__init__.py 中的内容为空。假设当前所在位置与文件夹 a 和 b 平行，则：

```
>>>from a import hello # 导入 a 文件夹下的 hello.py
>>>from b import hello # 导入 b 文件夹下的 hello.py
```

在 Python 中，用 # 表示注释。导入 hello.py 也可以写成：

```
import a.hello 和 import b.hello
```

1.5 数据类型和变量

1.5.1 数据类型

Python 的数据类型有数值、字符串、元组、列表、字典、集合等。

1. 数值类型

数值类型包括 int、float、bool、complex（复数），如 10、3.5、False、2+3j，分别表示整型、

浮点型、布尔型、复数型。布尔型取值为 True 或 False。表示复数的格式是 a+bj，其中 a 是实部，b 是虚部，虚部后面必须有后缀 j 或 J。

```
>>>x=2+3j
>>>print("x 的实部是{},虚部是{},共轭复数是{}".format(x.real,x.imag,x.conjugate
()))
```

直接写一个整数值如 10，默认是十进制整数。若要表示一个二进制数字，则在数字前置 0b 或 0B；若要表示一个八进制整数，则在数字前置 0o 或 0O，之后接上数字 1～7；若要表示一个十六进制整数，则以 0x 或 0X 开头，后接 1～9 及 A～F，如 0b10、0XF、0o13。

2. 字符类型

字符类型需要将字符用单引号或者双引号括起来，如：

```
>>>"hello world"# 输出'hello world',"hello world"与'hello world'相同
```

编程时常常需要将字符串按照一定的格式显示或输出，如要求 10/3 只保留两位小数，输出字符串长度为 6：

```
>>>x=10/3
>>>'%6.2f'%x  #输出'  3.33'
```

对字符串 x 格式化，语法格式为：

```
'%[-][+][0][m][.n]格式字符'%x
```

格式中第 1 个%表示格式开始；第 2 个%是格式运算符；－表示左对齐；＋表示正数时加上＋号；0 表示空白用 0 补齐；m 指定最小宽度；.n 是保留的小数点位数。格式化的控制符号见表 1-1。

表 1-1　常用的格式化的控制符号

控制符号	说　明	控制符号	说　明
%%	输出字符%	%b	二进制整数
%d	十进制整数	%o	八进制整数
%f,%F	十进制浮点数	%x,%X	十六进制整数
%e,%E	底为 e 或 E 的指数	%g	指数 e 或浮点数
%s	字符串，以 str()显示	%G	指数 E 或浮点数
%r	字符串，以 repr()显示	%c	单个字符

下面是格式化输出的示例：

```
>>>a=10
>>>b=3
>>>'%-5d除以%-5d是:%8.3f'%(a,b,a/b)     #输出'10   除以 3    是:   3.333'
```

如果输出的格式较复杂，上面的方式可读性较差，为此可以使用 format()函数：

```
>>>'{}除以{}是:{}'.format(a,b,a/b)       #输出'10 除以 3 是:3.3333333333333335'
```

{}是占位符,{n}中没有指定 n 时,用 format()中给定参数先后顺序填入占位符;如果指定了 n,则按照 format()中参数(顺序为 0,1,2,…,n)填入占位符。在占位符中,可以使用上面字符串格式化的格式,如:

```
>>>'{0:d}除以{1:d}是:{2:.3f}'.format(a,b,a/b) #输出'10 除以 3 是:3.333'
>>>'{1:d}除以{0:d}是:{2:.3f}'.format(b,a,a/b) #输出'10 除以 3 是:3.333'
>>>"{0:}的二进制数为: {0:b} ".format(80)        #输出'80 的二进制数为: 1010000'
>>>"{0:}的二进制数为: {0:#b} ".format(80)       #输出'80 的二进制数为: 0b1010000'
```

字符串中 0、1、2 表示参数的顺序,输出时与 format()中的参数相对应。参数的顺序可以用指定名称的方式,如下面的 a、b、c,这种情况下需要在 format()中为这些参数赋值:

```
>>>'{a:d}除以{b:d}是:{c:.3f}'.format(a=10,b=3,c=a/b)     #输出'10 除以 3 是:3.333'
```

使用 format()时如果要指定对齐方式,可以使用“<”(表示左对齐)和“>”(表示右对齐),默认是右对齐,如果数据位数不足,指定宽度时要补上指定的字符,可以使用“^”指定:

```
>>'{0:*^5d}除以{1:!^6d}是:{2:6.3f}'.format(a,b,a/b)     # 输出'*10**除以!!3!!!
                                                        #是: 3.333'
```

3. 转义字符

有些字符,如回车换行符等不方便放在字符串中,就需要以转义字符的形式输入这些字符。转义字符由一个\开头,后面接着需要添加的字符,如双引号转义字符是\"。下面是\'转义字符的效果:

```
>>>speak="同学问:\"如何才能学好 Python? \""
>>>speak #输出'同学问:"如何才能学好 Python?"'
```

不使用转义字符,不能在双引号中再加上双引号。常用的转义字符见表 1-2。

表 1-2 转义字符

转义字符	说 明	转义字符	说 明
\n	换行	\t	制表位
\'	单引号	\\	反斜杠
\"	双引号		

可以在字符串的开始处加上 r,使其成为原始字符串,此时会忽略字符串中所有的转义字符。比较下面两个输出的字符串:

```
>>>print("say hello to Peter\'s mother")      #输出 say hello to Peter's mother
>>>print(r"say hello to Peter\'s mother")     #输出 say hello to Peter\'s mother
```

以 r 开始的字符串,将字符串中的转义字符直接以原样输出,这在正则表达式中经常会用到。

1.5.2 变量

Python 中变量不用预先声明,变量区分大小写,每个变量使用前必须被赋值。

```
>>>a=1                #给一个变量赋值
>>>x=y=z=1            #同时给多个变量赋相同的值
>>>a,b=1,"hello"      #同时为两个变量赋不同的值
>>>a, b, c, d=10, 2.5, True, 1+2j
                      #a,b,c,d 的数据类型分别为 int,float,bool,complex
>>>c="hello"*3
>>>d="hello"+" world"
>>>a="I lovePython"
>>>print(a[0])        #输出 I
>>>print(a[0:6])      #切片操作,从 0 个字符截取到第 6 个字符(不包括第 6 个),输出: I love
>>>len(a[0:6])        #测字符串长度,输出: 6
```

Python 属于动态类型的程序设计语言,变量本身没有数据类型的信息,变量始终是个引用到该值的名称,赋值运算只是改变了变量引用的对象。

```
>>>x=10
>>>y=x
>>>print(id(10),id(x),id(y)) #输出 140732857234544 140732857234544 140732857234544
```

上述语句中对象 10 赋给 x,又将 x 引用的对象赋值给 y,利用 id()获取引用对象的地址,可见 x、y 引用的都是对象 10 的地址。

```
>>>x=10
>>>y=20
>>>print(id(x),id(y))          #输出 140732857234544 140732857234864
```

将 y 引用到另一个对象 20 后,可以看到 x、y 引用的地址就不一样了。

```
>>>x=x+10
>>>id(x)                       #输出 140732857234864
```

x 原来引用到对象 10 的地址,通过执行 x=x+10 后,创建了新的对象,x 引用到对象 20 的地址。

```
>>>x=[10,20,30]
>>>y=x
>>>x[0]=40
>>>y                           #输出[40, 20, 30]
```

y=x 赋值语句使 y 引用的地址与 x 相同,x[0]=40 修改了 x 引用地址上存放的数据,故输出的 y 值也会发生改变。当变量不再需要时,可以使用 del 删除。

```
>>>z=10
>>>del z
```

程序中有时要判断变量赋值后的数据类型,可以使用 Python 内部函数 isinstance()。

```
>>>n=input("输入 n:")
```

输入 n:5

```
>>>isinstance(n,int)           #输出 False
>>>isinstance(n,str)           #输出 True
```

isinstance(x,type)中 x 是要判断的变量,type 是数据类型,如 str、int、float、tuple、list、dict、set 分别是字符串、整型、浮点型、元组、列表、字典、集合。也可以使用 type()函数判断数据类型。

```
>>>x=10
>>>type(x)==int                #输出 True
```

虽然 isinstance 和 type 都可以判断数据类型,但在判断继承关系时,二者存在差异:类 B 继承于类 A,但 type 却判定 B()不是类型 A。

```
class A:
    pass
class B(A):
    pass
print(isinstance(A(), A))        #返回 True
print(type(A())==A)              #返回 True
print(isinstance(B(), A))        #返回 True
print(type(B())==A)              #返回 False
```

1.5.3 运算符

1. 赋值运算符

使用“=”为某个变量赋值,Python 支持连续赋值,如:

```
>>>a="北京"
>>>b=c=d=100
>>>a,b=1,5
```

2. 比较运算符

比较运算符用于比较两个值的大小,比较的结果是 True 或 False。比较运算符包括>、>=、<、<=、==、!=、is(判断两个变量所引用的对象是否相同)、is not(判断两个变量所引用的对象是否不相同)。

```
>>>x=[10,20,30]
>>>y=[10,20,30]
>>>x==y                          #输出 True
>>>x is y                        #输出 False
>>>id(x)                         #输出 2136931039112
>>>id(y)                         #输出 2136930785032
```

通过 id(x)和 id(y)的值可以看出 x 和 y 指向的地址不同,故 x 和 y 是两个不同的变量,x is y 返回 False。下面代码中,10 与 z 指向的地址相同,故 z is 10,返回 True。

```
>>>id(10)                        #输出 140732857234544
>>>z=10
>>>z is 10                       #输出 True
```

x==y 时,逐一比对 x、y 两个列表中的元素是否相同,如果全部相同则返回 True;否则返回 False。