

第一章 快速入门

本章导读

Python是一种面向对象的解释型程序设计语言，其语法简洁，功能强大，适用诸多领域，且易于学习。

Python于1989年底由Guido van Rossum(吉多·范·罗苏姆)发明，1991年Python第一个公开版本发行，目前以Python3.X版本为主。Python源代码遵循 GPL(通用公众许可，General Public License的简写，即许可用户运行和复制软件自由，发行传播软件自由，获得软件源码自由，更改软件并将更改软件发行传播自由，通常称GPL软件为免费软件)。

Python常被昵称为**胶水语言(Glue Language)**，易于集成其他语言制作的各種模块(尤其是C/C++)。常见典型情景是用Python快速生成软件原型，然后对其中有特别要求部分，用更合适的语言改写。如3D游戏中的图形渲染模块，性能要求特别高，就可以用C/C++重写，而后封装为Python可以调用的扩展库。

Python扩展库丰富，已成为Python重要特征，涉及到方方面面，包括：数据分析、自然语言处理、多媒体处理、人工智能、大数据等等方面。学习Python，就能将各种模块应用到实际学习和工作之中成为，解决问题的重要工具。

本章是Python快速入门。学习本章，达到Python程序设计入门，掌握Python开发环境、程序设计步骤、基本语法等。

学习目标：

1. 掌握开发环境和开发步骤；
2. 理解直接量、间接量、变量等概念；
3. 理解数据类型等基本概念；
4. 掌握基本流程控制语句；
5. 掌握str、list、tuple等基本操作；
6. 掌握Python切片操作；
7. 掌握基本文件初步操作；
8. 掌握最常用扩展库的使用；
9. 初步掌握自定义函数以及lambda函数；

本章目录

- 第一节 初识Python
- 第二节 键盘输入
- 第三节 一名多量
- 第四节 认识切片
- 第五节 循环语句
- 第六节 选择语句
- 第七节 print()函数
- 第八节 字符串
- 第九节 文件读写
- 第十节 再说函数
- 第十一节 扩展库
- 第十二节 与Excel

第一节 初识Python

成功安装Python后，即可踏上Python学习之旅。在Windows平台左下角Windows标志上单击右键，在弹出的菜单中选择“运行”将弹出如图1-1所示的命令窗口(或者Windows+R)，键入cmd回车后，将弹出如图1-2所示的窗口，在光标后键入“python”即可进入Python交互式环境(如果不能出现>>>则表示Python安装存在问题)，开始练习Python命令。

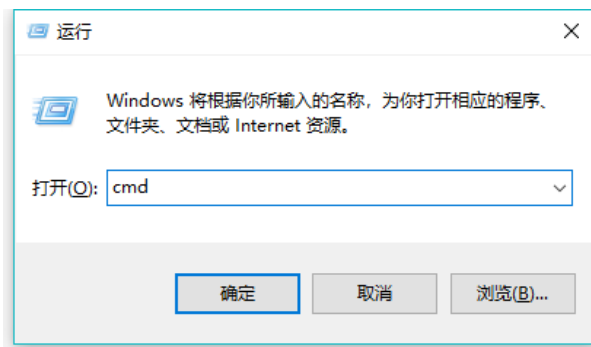


图1-1 进入Windows命令窗口

在>>>后, 可以键入Python的命令, 如`3+2`、`print("Hello,World!")`等。当命令输入完毕后, **按回车键 (键盘上的Enter键)**即可看到命令的执行结果, 如下图所示。**注意: 命令必须是英文半角符号**, 如`print()`命令后的英文, 则必须是英文引号, 中文引号将报错, 更多细节将逐步讲解。退出Python交互环境的命令是`exit()` (注意: 括号不能省略且必须是英文括号), 退出Windows命令窗口的命令是`exit`, 没有括号。当然, 点击右上角的“x”也可以。

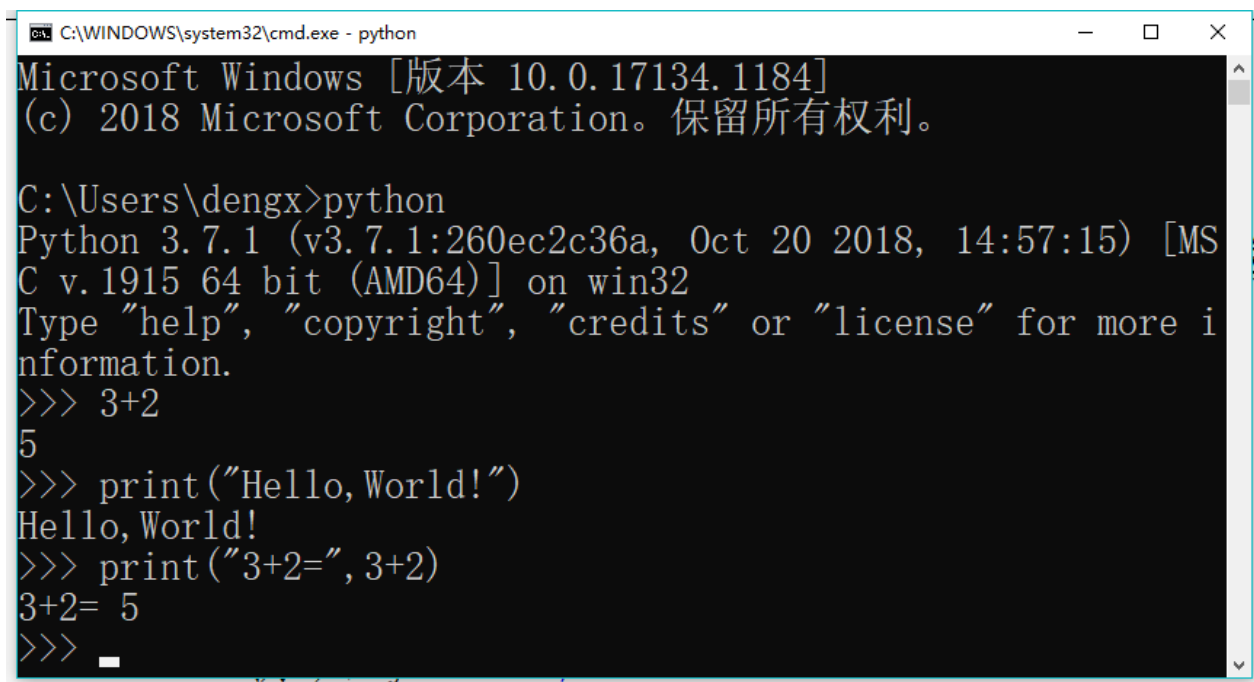


图1-2 Windows命令窗口

Python提供了集成的交互环境和编程环境。在开始菜单启动IDLE后, 如图1-3, 将弹出如图1-4所示的界面, 其使用方法与Windows命令窗口环境下的Python交互环境相一致。

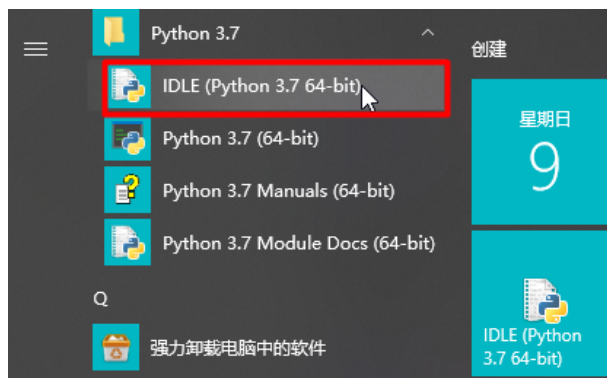


图1-3 启动IDLE

启动IDLE后, 如果没有出现如图1-4所示的>>>, 则表明IDLE没有安装成功, 请参考附件中的Python安装。

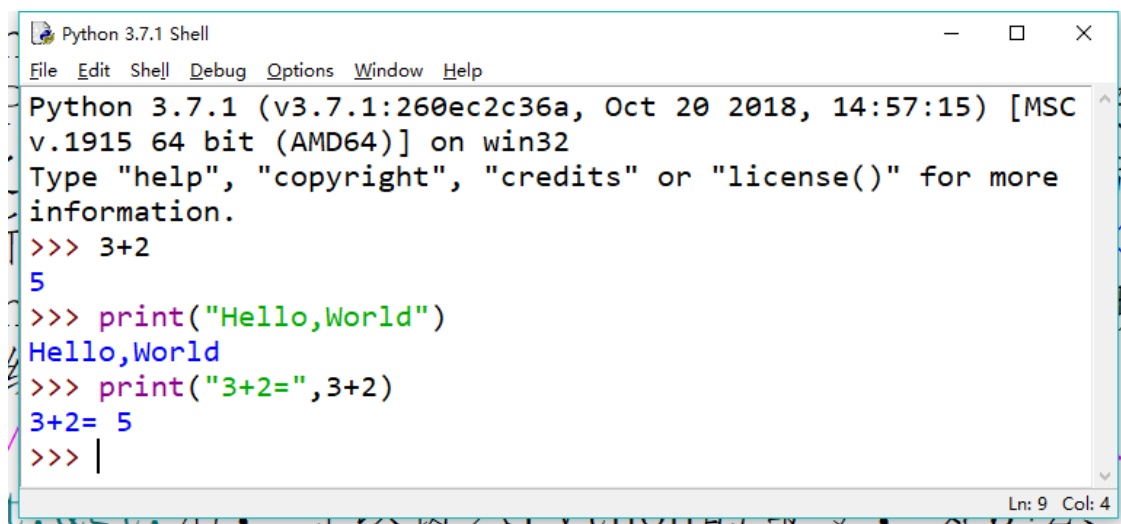


图1-4 Python的Shell界面

交互式环境利于执行短小命令，熟悉语法特征，但不能保存执行过的代码以便于重复执行或调试。将Python源代码有机地集中在一起，并保存文件之中，然后执行，就是Python程序。

撰写程序源代码，需要代码编辑器。一般说来，程序源代码可以用各种各样的文本编辑器编写（如：EditPlus、UltraEdit、Notepad++等），但专用软件更好。Python源码编辑，可以用Python自带的IDLE(Integrated Development and Learning Environment的简写，其意为集成开发与学习环境)，也可以用其他软件，如比较流行IDE软件(Integrated Development Environment，其意为集成开发环境)，如PyCharm等。

本书以Python自带的IDLE作为源码编辑器，启动后的界面如图1-5。选择图中的“File”菜单的“New File”或者按快捷键“Ctrl+N”即可进入代码编辑状态，如图1-6所示。键入如例程1-1或图1-6所示的源代码，选择“File”菜单下的“Save”，将源代码保存到指定的文件中。在本例中，程序被保存到C盘PythonProg文件夹下，命名为First.py。

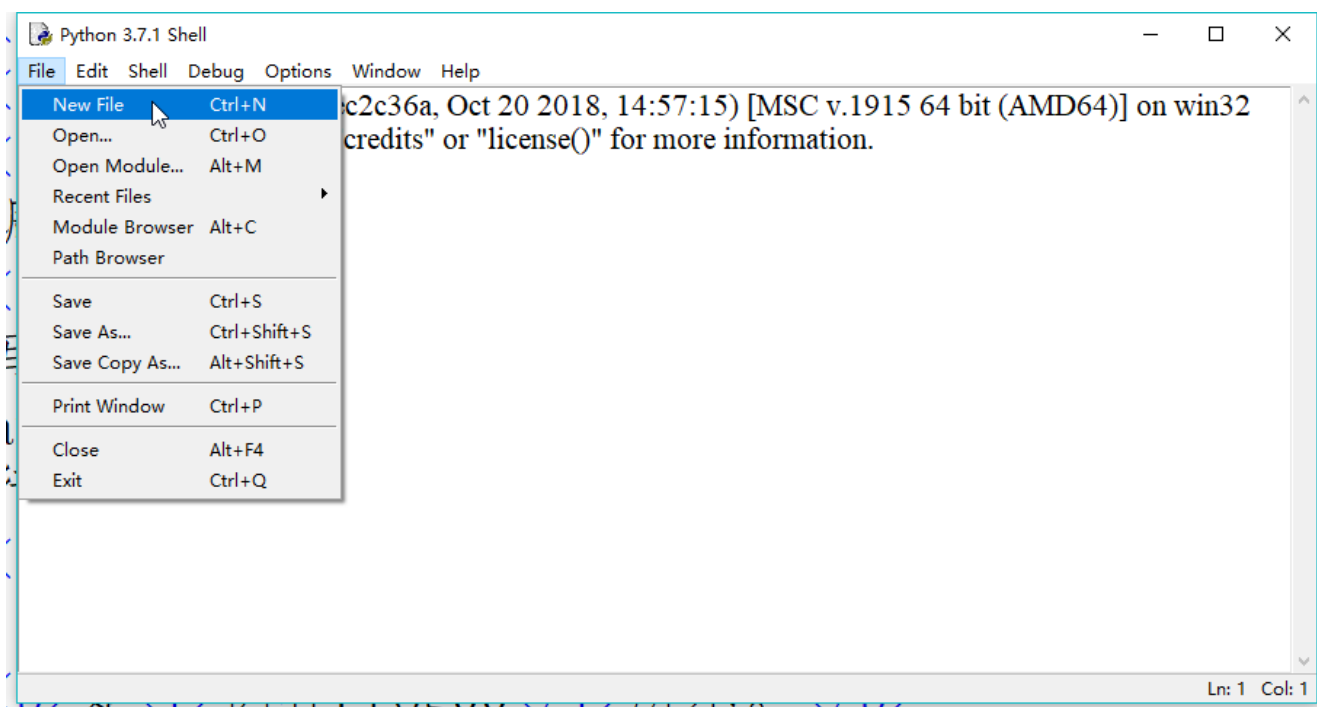


图1-5 Python自带源码编辑器IDLE启动界面

例程1-1

```

第1行  #我的第1个Python程序
第2行
第3行  print("Hello,World!") #输出：Hello,World!

```

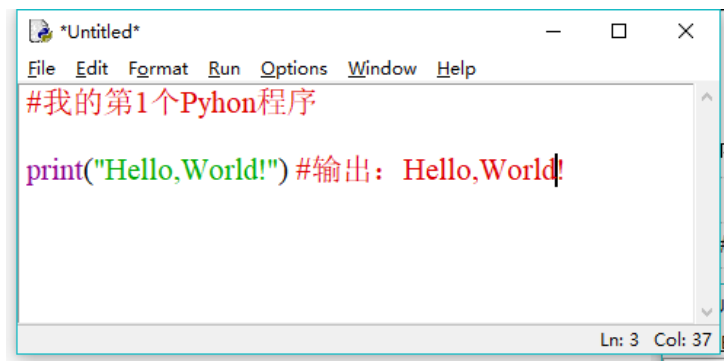


图1-6 Python自带源码编辑器IDLE代码编辑界面

源码编辑完毕后就是调试执行。选择菜单“Run”下的“Run Module”或直接按“F5”即可，如果尚未存盘将提示保存后执行，如果已存盘则直接执行，如图1-7所示，执行效果如图1-8图左所示。开发过程中，可以将撰写代码的窗口(图1-8图右)与观察执行效果的窗口(图1-8图左)并排，有利于程序调试，提升效率。在Windows中，可以用“Windows+键盘上的左或右箭头”实现。

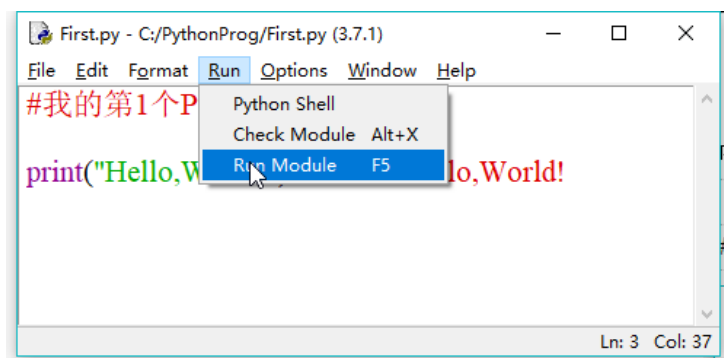


图1-7 IDLE中执行源码的菜单

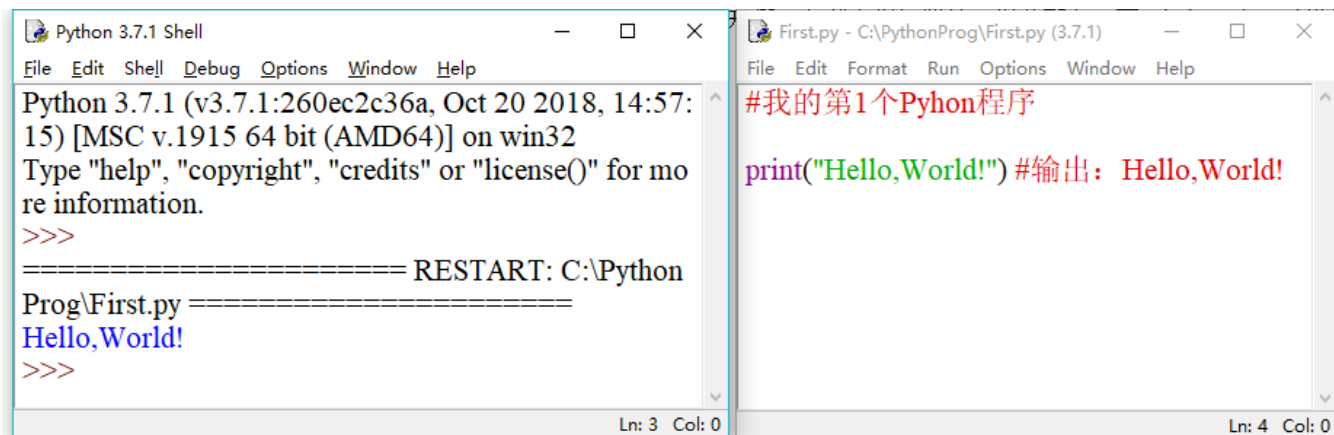


图1-8 例程1-1的执行效果

如果能输出“Hello,World!”则说明已准备好开发环境，已初步了解开发流程。如果不能输出“Hello,World!”则可能是多种原因，如开发环境尚未准备就位等，但最常见的原因是代码输入错误，如：

- **大小写错误：**Python是大小写敏感的语言，如例程中的print()功能是输出括号内的内容，写成Print()就是错误，注意P被大写了，而正确应该是小写；
- **括号错误：**print()后的括号必须是英文括号，但如果写成中文括号，则将导致错误，如图1-9所示；
- **引号错误：**一般说来在程序设计中，必须是英文引号且必须配对。相对于英语环境，可能在程序编写过程中忘记中英文切换而输入中文引号，将导致错误。另外，引号必须配对使用，即双引号开始，则必须配对双引号结束，如例程所示。在当前常见的环境中，中文英文切换按shift键即可；
- **注释错误：**注释对程序执行结果没有正面和负面影响，但有利于注解思路、提请注意、更快了解代码等，是良好的工程习惯。Python的注释以英文#开始，如果输入中文#将导致错误。

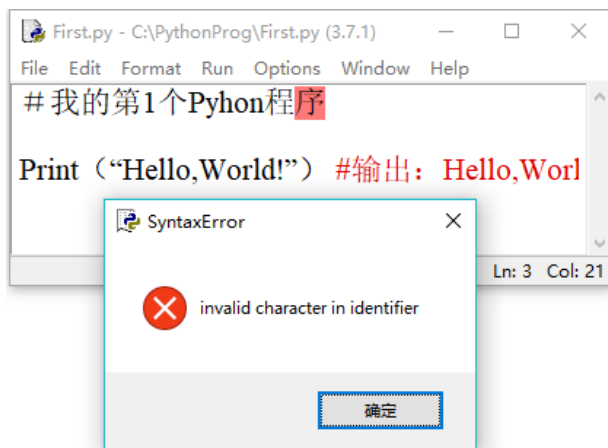


图1-9 例程1-1中的常见错误

IDLE不算完整功能的集成开发环境如PyCharm，但胜在简单实用且随Python安装而自带且基本功能已经具备。观察图1-6会发现不同内容用不同颜色表示，如：注释是红色，如果期望是注释但没有变色，则很可能是#输入错误；引号及引号内的内容是绿色等。另外，如果弹出如图1-9所示的对话框（invalid character in indentifer: 标识符中有错误字符），通常是因为英文符号误为中文符号。

上述错误，不必介怀，更不要想东想西，幻化出各种奇奇怪怪的说法，比如：智商高低、男女性别、文科理科生、大学生中学生等等。如果是IDLE不能成功启动则可能是Python安装有问题。由于Windows安装或运行后会有多种可能，难免导致安装失败，向老师助教或者有经验的人求助即可，“不好意思”等想法完全没有必要。如果IDLE能启动，但程序执行总有问题，很可能是上述错误，看看能不能对症下药，如果还是不能正确执行，尽快求助千万不要把问题积累。**积累问题是程序设计学习的大忌！些微小问题的积累可能就量变为拦路虎！**

“Hello,World!”程序虽然简单，但如果能正确运行，就表明环境已准备好，且初步掌握了某门程序设计语言的基本规则，很利于后续程序设计的学习；如果该程序不能正确执行，无需担心，及时解决即可。

Mac平台的IDLE

Mac安装Python请参考附件。当安装完毕后，进入python交互环境的步骤：单击启动台→单击其他→单击终端，将出现如图图1-10所示的界面。注意在命令行提示符后(\$后)，键入python3，而不是Windows下的Python。因为Mac平台自带Python，但是Python2.X的版本。新安装的Python一般都是Python3.X，且本书也是Python3.X为准。退出Python交互式环境的命令同样是exit()函数。

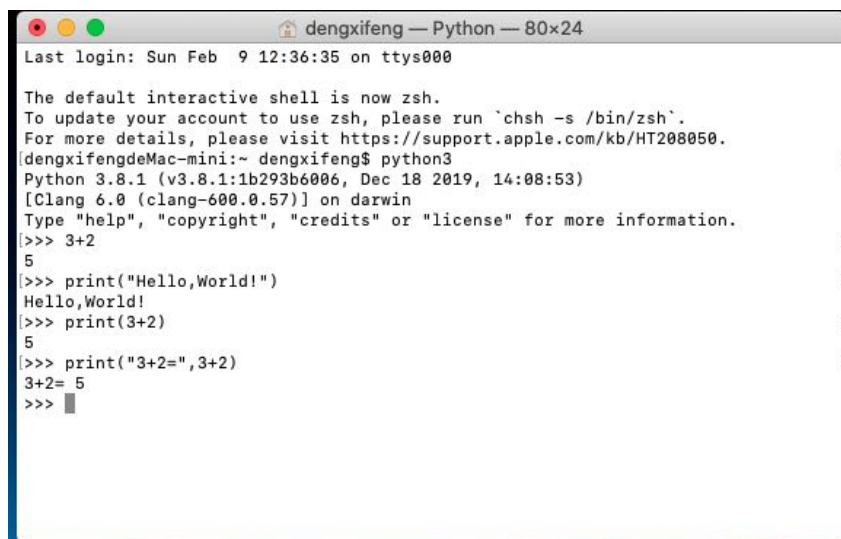


图1-10 Mac终端界面

在Mac平台启动IDLE的方法是idle3，而不是Windows平台的idle，原因与python3一样。启动IDLE后的界面如图1-11所示。进入IDLE后，Mac和Windows平台相差不大。

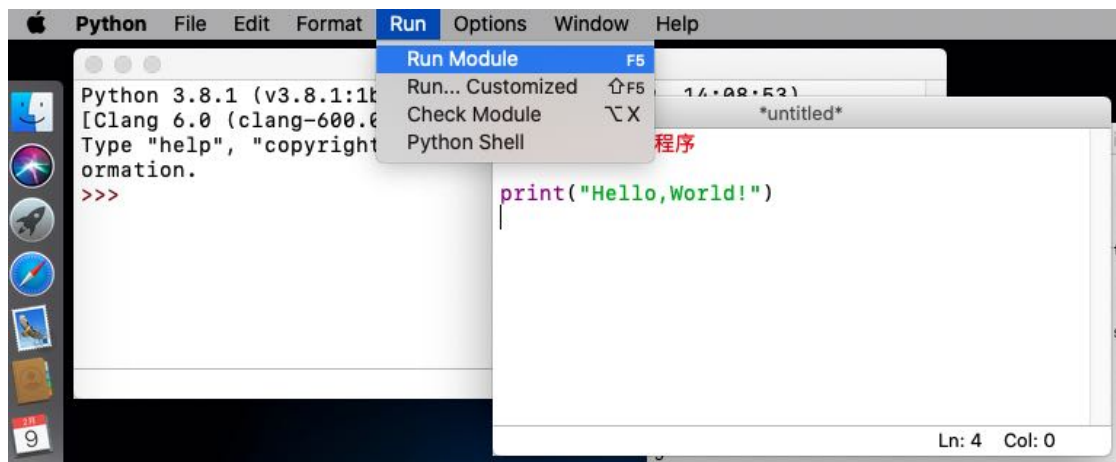


图1-11 Mac的IDLE

另外，和Windows平台不同，Mac平台大小写敏感。在Windows平台，输入Python或者python结果一样，但在Mac平台，则输入python3可以执行，但Python3则不会执行而报错。

继续认识“Hello, World”程序。在例程1-1第3行代码`print("Hello,World!")`中，`print()`称之为函数(中文函数对应的英文为function，与数学中的函数有颇多相似)，`print`是函数名称。和很多程序设计语言一样，函数名称后都必须跟随英文括号，括号内为参数(parameter)。`print()`的基本功能是输出参数所指定的内容，此处输出“Hello,World!”。例程1-2是`print()`函数的更多示例。

例程1-2

第1行	#print()更多示例
第2行	
第3行	<code>print(3+2)</code>
第4行	<code>print("3+2")</code>
第5行	<code>print("3+2=",3+2)</code>
第6行	
第7行	<code>a=3 #让a的值为3</code>
第8行	<code>b=2 #让b的值为2</code>
第9行	
第10行	<code>print(a+b)</code>
第11行	<code>print("a+b=",a+b)</code>
第12行	
第13行	<code>a="李白"</code>
第14行	<code>b="杜甫"</code>
第15行	
第16行	<code>print(a,b)</code>

```

5
3+2
3+2= 5
5
a+b= 5
李白 杜甫

```

图1-12是例程1-2的执行结果，从中可以看出，代码第3行输出结果为5。与其对照，第4行仍然原样输出为“3+2”。第4行相比第3行，多了一对英文引号。在很多程序设计语言中，**用引号界定的内容称之为字符串(string)**，当将字符串作为函数`print()`的参数时，将原样输出，不管里面是不是如3+2这样的加法式子。

第5行是第3、4行的组合，同时表明`print()`函数支持1个或者2个参数【注：`print()`函数实际能支持更多参数，以后会更详细介绍】。当参数个数多于1个时，参数和参数之间用英文逗号分隔，不仅适用于`print()`，其他函数同样如此。

图1-12 例程1-2执行结果

第10行执行后同样输出5，因为第7行和第8行的原因，代码中a代表3，b代表2，因此a+b的结果为5。a=3和数学中的“令a等于3”有相似的意义。

观察图1-12会发现自动分成多行，在Python中，**`print()`函数执行后默认自动换行，换到新的下一行，后续输入在此开始。**

`print(a+b)`和`print(3+2)`执行后都是输出5，但在`print(3+2)`中是直接两个值相加，而在`print(a+b)`中，是用a和b代表的值相加。在程序设计中，称3或者2、“3+2”以及第13行的“李白”和第14行的“杜甫”为**直接量**，而称a+b中的a和b为**间接量**，**间接量用名**

称（或称标识符）代表值。在直接量中，字符串直接量必须加上英文引号界定如："李白"、"杜甫"、"3+2="、"a+b="等等，而其他量则不能用引号界定，直接书写即可。对于间接量，不管是否代表字符串都不能用英文引号界定，如a="李白"相当于用a代表"李白"，虽然a代表的是一个字符串，但也不能加上引号，如果加上"a"则表示单个字符的字符串直接量。

在Python中，**间接量还可以称为变量(variable)**，即该名称代表的量可以发生改变，如例程1-3所示。在第3行a代表3，在第7行a代表5，即a所代表的值前后发生了变化，称之为变量名副其实。

例程1-3

```
第1行 #变量示例
第2行
第3行 a=3
第4行 b=4
第5行 print(a*a+b*b) #输出：25，*相当于数学中乘法符号
第6行
第7行 a=5
第8行 print(a*a+b*b) #输出：41
```

在例程1-3第3行a=3和第4行b=4这样类似的语句被称为**赋值语句(assignment statement)**，其功能是将等号右边的值或者运算的结果放到等号左边的变量名称中。例程1-4是赋值语句的更多示例。第6行是对两个变量赋以不同的值，第8行是多个变量赋以相同的值。

例程1-4

```
第1行 #赋值语句示例
第2行
第3行 a=3
第4行 b=a+4 #将等号右边a的值取出然后将运算结果赋值给b
第5行
第6行 a,b=3,4 #各自赋值。将3对应赋值给a，4对应赋值给b
第7行
第8行 a=b=c=5 #连等赋值。将a、b、c赋值为5
```

例程1-3第5行代码a*a+b*b中有两个**运算符+和***，每个运算有参与其运算的**操作数**，如在a*a中，*是运算符表示乘法，两个a分别是*运算符的**操作数**，这种由**运算符与操作数构成的序列，称之为表达式(expression)**。当表达式中有多个运算符时，要考虑**运算符计算的先后即运算符的优先级**，如同小学四则运算中的先乘除后加减。例程1-3第5行代码a*a+b*b中，先计算a*a然后b*b最后是加号运算。

加减乘除运算符在Python以及其他很多程序设计语言中，分别用+、-、*、/表示。除此以外，在Python中用%表示求余运算(求模)、**表示幂运算、//表示整除运算(两数相除的整数部分，不是四舍五入)，其中幂运算符具有最高优先级。例程1-5是运算符的相关示例。另外，Python还有其他运算符，将在相关章节讲解。

例程1-5

```
第1行 #运算符示例
第2行
第3行 a=5 #赋值语句
第4行 b=3
第5行
第6行 print(a**b) #输出：125，相当于a=5的三次方
第7行 print(a%b) #输出：2
第8行 print(a/b) #输出：1.6666666666666667
第9行 print(a//b) #输出：1
```

另外，+、*不仅用于数值计算，还能用于字符串运算，如例程1-6所示。+用于字符串拼接，即将多个字符串拼接为一个字符串；*用于字符串重复。

例程1-6

第1行	#+、*字符串运算符示例
第2行	
第3行	a="Yes"
第4行	b="No"
第5行	print(a+b) #输出: YesNo, 将a和b所代表的字符串拼接在一起
第6行	
第7行	print("李白"+"杜甫") #输出: 李白杜甫
第8行	
第9行	print("Ha"*3) #输出: HaHaHa, 重复三次

第二节
 键盘输入

前述例程都不能实现从键盘输入，没有输入就没有灵活。在Python中，print()函数是输出，input()函数是输入。例程1-7是input()函数的应用示例。运行时，当从键盘输入“李白”时，程序会通过print()函数输出刚才输入的内容。另外，input()函数的参数可以省略，即tangPoet=input()同样可以正确执行，只是没有了提示信息。良好的提示，对用户更加友好。

例程1-7

第1行	#input()函数应用示例1
第2行	
第3行	tangPoet=input("请输入唐代诗人的名字: ")
第4行	print(tangPoet)

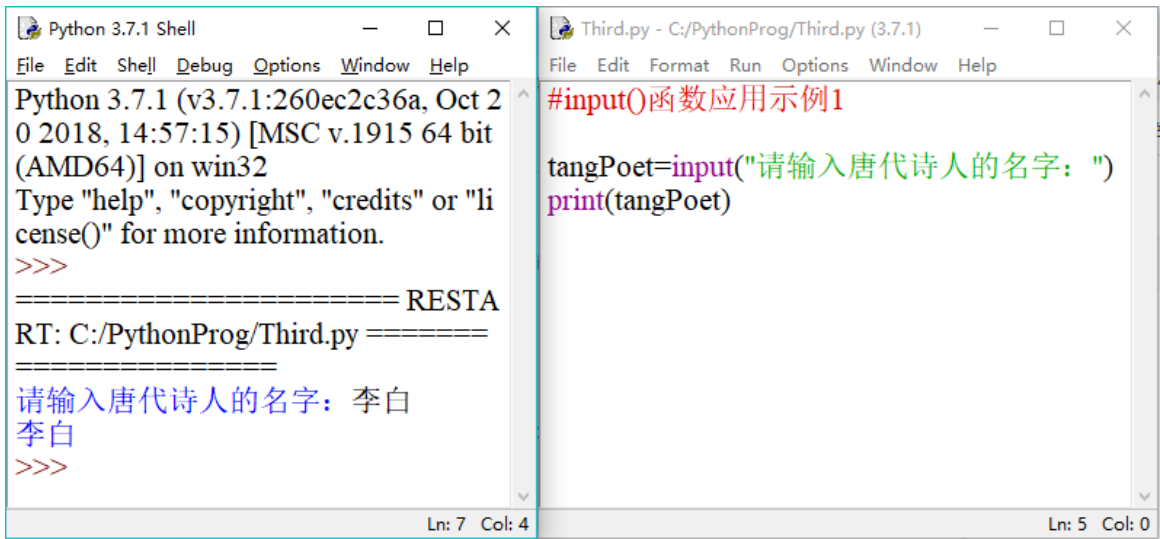


图1-13
 例程1-6执行结果

例程1-8是input()函数的又一示例，实现两个数相加，执行效果如图1-14所示。

例程1-8

第1行	#input()函数应用示例2
第2行	
第3行	firstNum=input("请输入第一个数: ") #假如输入: 3
第4行	secondNum=input("请输入第二个数: ") #假如输入: 2
第5行	print(firstNum+secondNum) #输出: 32
第6行	

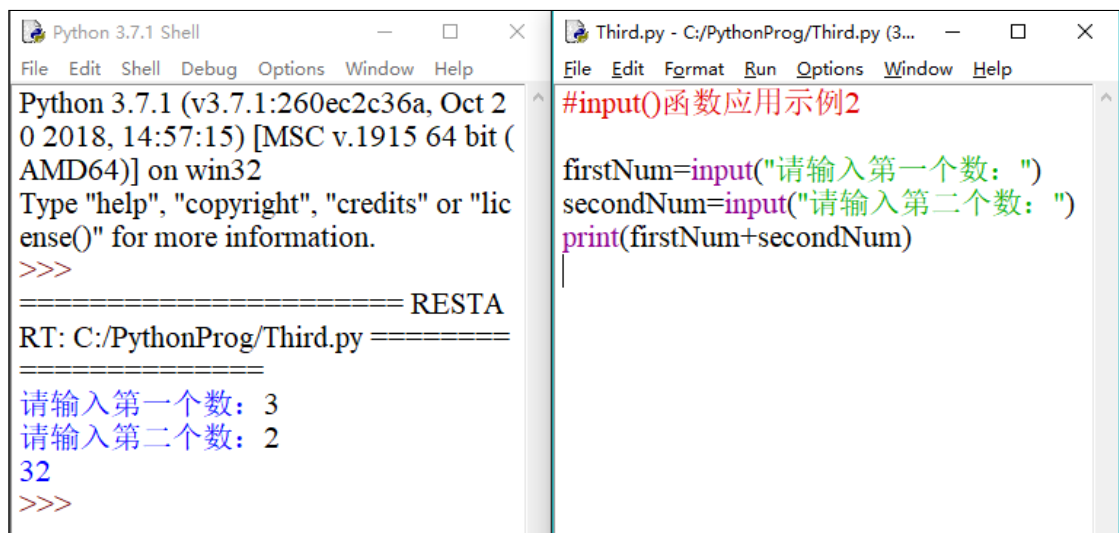


图1-14 例程1-7执行结果

从图中可以看出，并没有实现两个数相加，而是先后拼接，其原因是：input()函数只能输入字符串。在本程序相当于先后输入了"3"和"2"，然后用+拼接后输出32。对于input()函数，即便输入是数字，也只是由数字组成的字符串。如何解决呢？Python提供了int()函数用于将数字组成字符串转换为整数，如例程1-9所示，图1-15是其执行效果。

例程1-9

第1行 #input()函数应用示例2

第2行

第3行 numA=input("请输入第一个数: ")

第4行 numB=input("请输入第二个数: ")

第5行 firstNum=int(numA)

第6行 secondNum=int(numB)

第7行 print(firstNum+secondNum)

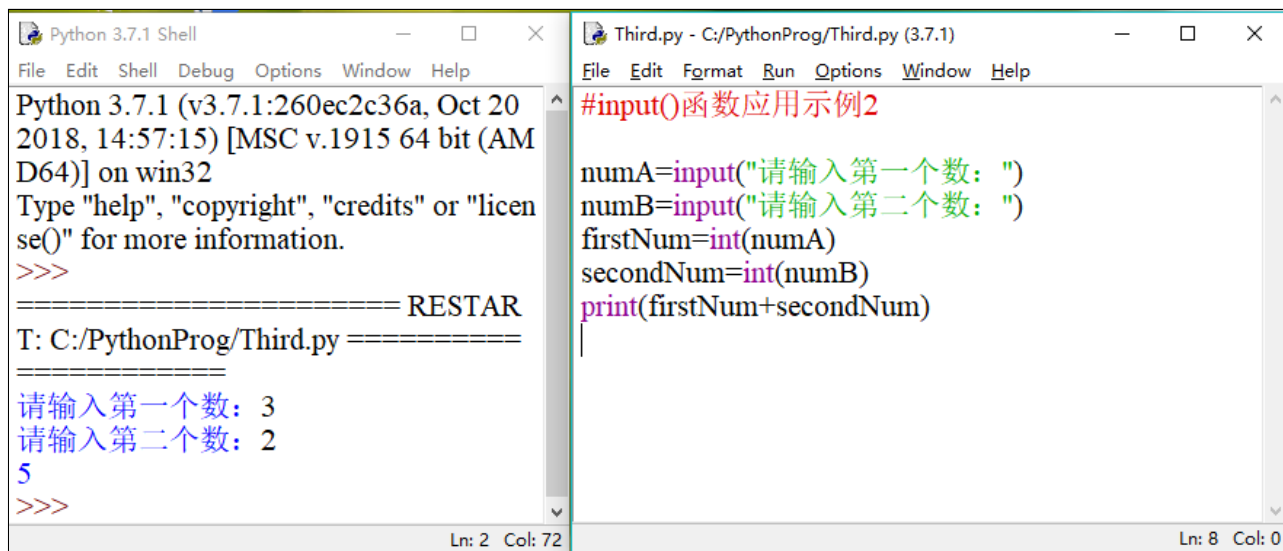


图1-15 例程1-9执行结果

在例程1-9中，第1个输入的值存入变量numA之中，第2个存入numB之中。第5行和第6行则将numA和numB分别通过函数int()转换为整数后存入变量firstNum和secondNum之中，第7行输出两数相加的和。从图1-15也可以看出，程序运行符合预期。输入其他整数，无需调整程序，运行即可，且结果正确。例程1-10是对例程1-9改进，减少了代码量。

例程1-10

第1行 #input()函数应用示例2

第2行

第3行 firstNum=int(input("请输入第一个数: "))

```
第4行 secondNum=int(input("请输入第二个数: "))
第5行
第6行 print(firstNum+secondNum)
```

在例程1-10中, int()函数内嵌入input()函数。当函数内嵌入函数时, 从内到外执行, 即先执行内层函数, 然后执行外层函数。内层函数的结果成为外层函数的参数或者说输入。

第三节 一名多量

前述例程都是一个变量代表一个值, 如果要代表某班50个学生数学的成绩, 难道需要50个变量? 能否类似数学的 X_i , 其中 i 的取值是1到50? 几乎所有程序设计语言都提供了类似能力, 如例程1-11所示。观察例程第3行, Math变量的等号后有6个数, 用英文方括号配对界定。这种形式在Python中被称为**list(列表)**。list能将多个数据组织在一起, Python称其为**组合数据类型 (compound data type)**。除list外, 还有其他类型, 如tuple(元组)、dict(字典)、set(集合)等, 将逐步讲解。

例程1-11

```
第1行 #一个变量代表多个值
第2行
第3行 Math=[78,98,89,67,76,99] #此处Math代表6个数,6个数与50个数, 形式上没有本质区别
第4行
第5行 print(Math[0]) #输出: 78, 获得编号0的值
第6行
第7行 print(Math) #输出: [78,98,89,67,76,99]
第8行
第9行 Math[4]=100 #改变编号为4的元素值
第10行 print(Math) #输出: [78, 98, 89, 67, 100, 99]
第11行
第12行 mathSum=Math[0]+Math[1]+Math[2]+Math[3]+Math[4]+Math[5]
第13行
第14行 print("成绩总和: ",mathSum) #输出: 成绩总和: 531
第15行
第16行 print("平均成绩: ",mathSum/6) #输出: 平均成绩: 88.5
```

例程1-11第5行Math[0]表示获得编号为0的数据成员。在Python中, 多个数据组成的序列, 其编号都是从0开始, 其他很多程序设计语言同样如此。第9行则Math[4]出现在等号左边, 可以对编号为4的成员赋值(修改其值)。对于list, 通过编号(也可以称为下标或索引)可以获得或者改写成员值。

对于list, 可以没有成员, 也可以通过函数增加成员, 如例程1-12所示。

例程1-12

```
第1行 #空list和增加成员
第2行
第3行 Math=[] #空list, 即没有成员的list
第4行
第5行 Math.append(99) #向Math增加一个成员
第6行 print(Math) #输出: [99]
第7行
第8行 Math.append(115) #向Math再增加一个成员
第9行 print(Math) #输出: [99,115]
第10行
第11行 Math.append(105) #向Math再增加一个成员
第12行 print(Math) #输出:[99,115,105]
```

在例程1-12的第3行代码中，`[]`表示空list即没有成员的list，没有第3行，执行第5行将导致错误。第3行相当于确定Math是list类型的间接量或变量。第5行代码append()函数用于向list变量增加成员。注意：append()也是函数，但是隶属于list的函数，因此前面必须有list类型的变量、间接量或直接量如Math等，如第5、8、11行所示。print()也是函数，可以认为是隶属于Python的函数，是Python的内置函数，前面不需要任何表示隶属关系的变量、间接量或直接量。

用input()函数，可以将多个从键盘输入的值存入Math变量，如例程1-13所示。但很明显，如果要输入很多值，采用例程所示方法，不是很好的方法，可采用本节后所讲述的循环语句予以很好解决。

例程1-13

```
第1行 #空list和增加成员
第2行
第3行 Math=[] #空list，即没有成员的list
第4行
第5行 nowScore=int(input("请输入数学成绩:"))
第6行 Math.append(nowScore) #向Math增加一个成员
第7行 print(Math)
第8行
第9行 nowScore=int(input("请输入数学成绩:"))
第10行 Math.append(nowScore) #向Math再增加一个成员
第11行 print(Math)
第12行
第13行 nowScore=int(input("请输入数学成绩:"))
第14行 Math.append(nowScore) #向Math再增加一个成员
第15行 print(Math)
```

list有排序函数sort()，很是常用，其使用方法如例程1-14所示，默认的排序规则是升序，即从小到大，如第5行的输出效果。如果需要从大到小，可在sort()函数中增加参数reverse，设置其值为True，即设置反序为真。

例程1-14

```
第1行 #list-排序函数sort()
第2行
第3行 Math=[78,98,89,67,76,99]
第4行 Math.sort()
第5行 print(Math) #输出: [67, 76, 78, 89, 98, 99]
第6行
第7行 Math.sort(reverse=True) #反序
第8行 print(Math) #输出: [99, 98, 89, 78, 76, 67]
```

在Python中，tuple与list相似，但其值不可更改，如例程1-15所示。比较第3、4行就会发现，list是方括号，tuple是圆括号。第5、6行是通过下标访问其成员，但由于tuple不可更改，因此第9行不能执行，对于list的第8行可以执行。空的tuple与空的list形式相同，都是没有成员，一方括号一圆括号的区分。但如果都只有一个成员，tuple必须多一个英文逗号。在本例程中，如果没有圆括号，相当于oneTuple的值为整数12，即相当于oneTuple=12。

tuple功能较少，数据不可修改，当数据量很大时，相较于list，速度更快。如果被处理数据无需修改，采用tuple是一个好习惯，防止误操作导致数据修改。

例程1-15

```
第1行 #认识Tuple
第2行
第3行 mathlist=[78,98,89,67,76,99] #这是list
第4行 mathTuple=(78,98,89,67,76,99) #这是tuple，由方变圆(方括号变圆括号)
第5行 print(mathlist[0]) #输出: 78
第6行 print(mathTuple[0]) #输出: 78
```

```

第7行
第8行 mathlist[0]=100
第9行 #mathTuple[0]=100 #错误! Tuple不能修改
第10行
第11行 nulllist=[] #空list
第12行 nullTuple=() #空tuple
第13行
第14行 onelist=[12] #只有一个成员的list
第15行 oneTuple=(12,) #只有一个成员tuple

```

在list中，可以嵌套list也可以嵌套tuple。当然，tuple中也可以嵌套tuple，也可以嵌套list(这种情况比较少见)。如例程1-16所示。在例程第6行listA[1][1]中，listA[1]是["李思",67]，然后对["李思",67]执行[1]运算，因此是67。例程第7行与之类似。listA[1]由于是list，因此可以被修改，而listB[1]是tuple，不可修改，因此第9行错误不能执行。第14-17行与之类似。

例程1-16

```

第1行 #list和tuple嵌套
第2行
第3行 listA=[["张珊",98],["李思",67],["王武",99]]
第4行 listB=[("张珊",98),("李思",67),("王武",99)]
第5行
第6行 print(listA[1][1]) #输出: 67
第7行 print(listB[1][1]) #输出: 67
第8行 listA[1][1]=100 #
第9行 #listB[1][1]=100 #错误! tuple不可修改
第10行
第11行 tupleA=(("张珊",98),("李思",67),("王武",99))
第12行 tupleB=(["张珊",98],["李思",67],["王武",99])
第13行
第14行 print(tupleA[1][1]) #输出: 67
第15行 print(tupleB[1][1]) #输出: 67
第16行 tupleB[1][1]=100
第17行 print(tupleB[1][1])

```

in和not in常用于list、tuple和str。in判断某个元素是否存在，not in与之相反，如例程1-17所示。代码第4行print(55 in numlist)的输出为False，因为numList中没有55存在；而第5行print(5 in numlist)的输出也为False，因为numList中没有5，虽然有54，但比较时每个成员与5进行比较，而不是成员的一部分。in和not in的结果只能是True或False，当存在其中时则为True，否则False。注意：True和False是直接量，又称为布尔值，且第1个字母必须大写，不能用引号界定。

例程1-17

```

第1行 #in和not in的使用
第2行
第3行 numlist=[12,34,54,54,32,67,98]
第4行 print(55 in numlist) #输出: False
第5行 print(5 in numlist) #输出: False
第6行
第7行 strHLM="众人见黛玉年貌虽小，其举止言谈不俗，身体面庞虽怯弱不胜，....."
第8行 print("黛玉" in strHLM) #输出: True
第9行 print("黛玉" not in strHLM) #输出: False

```

第四节 认识切片

切片(slice)用于从序列数据中截取其中一部分，适用于列表(list)、字符串(str)和元组(tuple)，如例程1-18所示。在例程第4行mylist[2:5]就是切片，表示从mylist中截取从编号为2开始直到5，但不包括5，相当于[2,5)，是左闭右开区间。

例程1-18

```
第1行 #切片(slice)示例
第2行
第3行 mylist=[14,17,2,6,7,100]
第4行 print(mylist[2:5]) #截取编号为2到5但不包括5，即编号为2、3、4
第5行 #输出： [2,6,7]
第6行
第7行 myStr="Crimes and Punishments"
第8行 print(myStr[-5:-1]) #输出:ment
```

例程1-18第8行代码myStr[-5:-1]表示从编号为-5开始截取到编号为-1仍然不包括-1，也仍然是左闭右开区间。在Python的序列数据类型中，最后一个元素的编号为-1，倒数第2个为-2，以此类推。因此，在例程中，编号-5是m，-1是s，由于不包括-1，因此是ment。

切片的语法是：[startIndex:endIndex:stepNum]，startIndex表示开始编号，endIndex是结束编号，stepNum是间隔，这3个参数都可以省略，但不可都省略。当省略startIndex时表示从编号0开始，当省略endIndex表示到最后一个位置且包含最后一个，当省略stepNum时默认为1。如例程1-19所示。

例程1-19

```
第1行 #切片(slice)示例
第2行
第3行 mylist=[1,2,3,4,5,6,7,8,9,10]
第4行 print(mylist[0:10:2]) #输出： [1, 3, 5, 7, 9]
第5行 print(mylist[:5]) #输出： [1, 2, 3, 4, 5]
第6行 print(mylist[5:]) #输出： [6, 7, 8, 9, 10]
第7行 print(mylist[-1:-10:-1]) #输出： [10, 9, 8, 7, 6, 5, 4, 3, 2]，不包括第1个
第8行 print(mylist[-1::-1]) #输出： [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]，包括第1个
第9行 print(mylist[:]) #输出： [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
第10行 print(mylist[::-1]) #输出： [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]，相当于反序输出
```

第五节 循环语句

例程1-11第12行将所有成员值加在一起，求得总和并将结果赋值给mathSum。第14、16行利用mathSum值输出总和和平均值。如果Math中有很多数据，成百上千上万怎么办呢？采用第12行方式不现实。例程1-20可以很好实现，图1-16是其执行效果。

78
98
89
67
76
99
Total Score: 507

图示中的前6项输出是第6行和第7行执行结果。在本例，for-in的功能是获得Math中每一个成员的值，然后存入到这个变量之中，每获得一个成员值执行一次第7行的print(i)，有多少个成员就执行多少次。当然，根据需要变量名i可以换成其他变量名称，如第10行对应位置为everyScore；同理Math也可以换成其他list（除了list之外，还可以是其他组合数据类型，如tuple、str、set、dict等）。

图1-16 例程1-20执行结果 在程序设计中，如果一个语句可能被多次执行，我们称其为循环（Loop）。for-in是Python中的一种循环方式。例程第10、11行同样为循环。在本循环中，Math中的每一个值被依次放入everyScore之中，每放入一次执行第11行语句一次。在循环之前的第9行，mathSum被设置为0。当第1次执行第11行语句时，等号右边的mathSum值为0，然后加上everyScore，即Math的第1个值78，执行后将结果赋值给mathSum，此时mathSum值变为78；当第2次执行第11行语句时，等号右边的mathSum值为78，然后加上everyScore，即Math的第2个值98，执行后将结果赋值给mathSum，此时mathSum值变为176。以此类推，最后结果为507。

根据Python的语法规则，for-in语句必须以英文冒号结尾，如例程第6行、第10行所示；被for-in控制的语句必须相对于for-in语句向后缩进，可采用1个或多个英文空格，也可以用键盘左边的TAB键缩进1个或多个(建议采用1个TAB)，但空格和TAB不可混用。如果被控制的语句是有多条，所有被控制语句必须采用相同方式缩进。

例程1-20

```

第1行 #for-in循环示例
第2行
第3行 Math=[78,98,89,67,76,99] #此处Math代表6个数,6个数与50个数，形式上没有本质区别
第4行
第5行 #输出每一个成绩
第6行 for i in Math:
第7行     print(i)
第8行
第9行 mathSum=0
第10行 for everyScore in Math:
第11行     mathSum=mathSum+everyScore
第12行
第13行 print("Total Score:",mathSum) #输出： 507， 注意：本次例程没有改变数据成员值

```

对于例程第11行，不可如数学一样等号两边减去mathSum，结果为0=everyScore。在程序设计中，其含义是先执行等号右边mathSum+everyScore，然后将结果存放入等号左边的变量。

例程1-21是对例程1-20成绩求和的部分的小改进。对比可以发现mathSum=mathSum+everyScore被改进为mathSum+=everyScore，其执行结果完全一样。+=也是Python运算符，称之为“累加运算符”，即以mathSum的初值0为基础，每次累加everyScore的值。除了+=累加运算符外，尚有-=（累减）、*=（累乘）、/=（累除）、//=（累整除）、%=（累求余）等，其用法与之相同。

例程1-21

```

第1行 #for-in循环示例
第2行
第3行 Math=[78,98,89,67,76,99] #此处Math代表6个数,6个数与50个数，形式上没有本质区别
第4行
第5行 mathSum=0
第6行 for everyScore in Math:
第7行     mathSum+=everyScore
第8行
第9行 print("Total Score:",mathSum) #输出： 507

```

例程1-22是求平均成绩，需要知道总成绩和人数。在例程中，每次循环人数（studCount）累加1。当循环结束时，人数和总成绩都能获得，也就可以计算出平均成绩。

另外，受第6行for-in循环控制的语句是第7、8两行，根据Python的语法规则，这两行必须采用相同的缩进方式，比如：都用1个TAB，或者都用1个或多个空格。

例程1-22

```

第1行 #for-in循环示例，求平均成绩
第2行
第3行 Math=[78,98,89,67,76,99] #此处Math代表6个数,6个数与50个数，形式上没有本质区别
第4行
第5行 mathSum,studCount=0,0
第6行 for everyScore in Math:
第7行     mathSum+=everyScore
第8行     studCount+=1
第9行
第10行 print("Average Score:",mathSum/studCount) #输出： 84.5

```


Python提供了获取成员数量的函数len(), 适用于包括字符串及包括list、tuple在内的组合数据类型, 例程1-22可修改如例程1-23所示。例程1-24是len()函数用于字符串的示例。

例程1-23

```
第1行 #len()函数示例
第2行
第3行 Math=[78,98,89,67,76,99] #此处Math代表6个数,6个数与50个数,形式上没有本质区别
第4行
第5行 mathSum=0
第6行 for everyScore in Math:
第7行     mathSum+=everyScore
第8行
第9行 studCount=len(Math) #获取名称为Math的list成员数量
第10行
第11行 print("Average Score:",mathSum/studCount) #输出: 84.5
```

例程1-24

```
第1行 #len()用于字符串示例
第2行
第3行 zenEng="Simple is better than complex" #Python之禅
第4行 zenChn="简单胜于复杂"
第5行
第6行 engCount=len(zenEng) #获得zenEng的长度,也就是字符串字符的个数
第7行 chnCount=len(zenChn) #获得zenChn的长度,也就是字符串字符的个数
第8行
第9行 print(engCount,chnCount) #输出: 29 6
```

例程1-25用于求1-10之和, 程序较为简单, numlist中有1-10共10个数, 第5行代码依次将1-10放入到变量i之中, 每放入一次执行第6行的累加, 第8行输出其结果。如果求1-100、1-1000、1-10000、100-10000怎么办呢? 注意观察例程1-26, 尤其是第4行与例程1-25第5行相比, 应该具有相同的作用才能得到相同结果。

例程1-25

```
第1行 #1-10之和
第2行
第3行 numlist=[1,2,3,4,5,6,7,8,9,10]
第4行 numSum=0
第5行 for i in numlist:
第6行     numSum+=i
第7行
第8行 print(numSum) #输出: 55
```

在Python中, **range()函数用于规律产生一系列整数**, 如range(1,11)产生1-10之间的每一个整数, 注意不包括最后一个数, 相当于[1,11), 是一个左闭右开区间。如果要求1-100之和, 将例程第4行的11修改为101即可, 求1-1000之和, 将11修改1001即可。

例程1-26

```
第1行 #1-10之和,range()应用示例
第2行
第3行 numSum=0
第4行 for i in range(1,11): #注意是到11, 不是10
第5行     numSum+=i
```

第6行

第7行 `print(numSum)` #输出: 55

```
输入起点数: 1
输入终点数: 10
55
```

```
>>>
```

```
=====
```

```
og/Sixth.py =====
```

```
输入起点数: 1
输入终点数: 100
5050
```

```
>>>
```

```
=====
```

```
og/Sixth.py =====
```

```
输入起点数: 2
输入终点数: 100
5049
```

例程1-26的缺点很明显, 如果要求某个范围内的数, 需要修改源程序, 不能直接从键盘上输入两个数求其范围内的和, 例程1-27是例程1-26的升级, 图1-17其执行结果。

从例程1-27可以看出, `range()`函数中的直接量1和11已经变成间接量或者变量, 这样就可以根据输入值的不同而发生改变, 适用于不同情况。在很多程序中, 直接量通常都可以改变为变量以适应不同的情况。

另外, 要特别注意 $n+1$, 因为是开区间, 最后一个数不会被计入。

`range()`的基本语法是`range(startNum,endNum[,stepNum])`, 其中`startNum`是起点数, `endNum`是终点数, `stepNum`是间隔数, `[,stepNum]`表示可以省略。当省略第3个参数时, 默认间隔为1, 可以设置为其他值, 如`range(1,100,2)`则表示从1开始的奇数, 到99为止。当只有一个参数如`range(5)`表示默认起点为0, 即相当于`range(0,5)`。`range(0)`相当于`range(0,0)`, 但根据规则不能包含0, 所以此时为空, 不会有成员。

另外, `range()`的`stepNum`可以为负数, 如`range(10,1,-1)`表示从10开始, 每次减少1个, 直到2为止, 同样不包括最后一个数1。如果省略`range(10,1,-1)`中的-1, 则同样为空。

图1-17 例程1-27执行结果

例程1-28是`range()`的又一示例。由于`range(1,15,2)`的间隔是2, 因此每次`i`值依次是1、3、5、7……, 体现在第4行, 则是*的重复次数依次是1、3、5、7……, 执行效果如图1-18所示。

例程1-27

第1行 #从m开始到n为止的连续整数之和,range()应用示例

第2行

第3行 `m=int(input("输入起点数: "))`

第4行 `n=int(input("输入终点数: "))`

第5行

第6行 `numSum=0`

第7行 `for i in range(m,n+1):` #注意: n要加上1, 开区间

第8行 `numSum+=i`

第9行

第10行 `print(numSum)`

```
*
***
*****
*****
*****
*****
*****
*****
*****
```

图1-18 例程1-28执行结果

例程1-28

第1行 #星号台阶, range()应用示例

第2行

第3行 `for i in range(1,15,2):`

第4行 `print("*"*i)` #根据i值决定重复*的次数

例程1-29利用循环实现录入指定人数的班级数学成绩(Math), 当然也适用于其他成绩。在例程第3行代码`Math=[]`申明Math为list类型。第5行执行后将录入班级人数, 第7行根据第5行的输入确定循环次数。注意, `range()`虽然不包括最后一个值, 此处为`studCount`, 但由于是从0开始, 所以数量仍然正确。第8行将输入的成绩存储到`nowScore`之中, 第9行将`nowScore`的值增加到名为Math的list之中。第11行输出Math全部成员。图1-19是执行结果, 输入的班级人数为5, 存储到变量`studCount`中。

```
请输入班级人数:5
请输入数学成绩:99
请输入数学成绩:97
请输入数学成绩:108
请输入数学成绩:110
请输入数学成绩:125
[99, 97, 108, 110, 125]
```

图1-19 例程1-29执行结果

例程1-29

第1行 #空list和增加成员

第2行

第3行 `Math=[]` #空list, 即没有成员的list

第4行

第5行 `studCount=int(input("请输入班级人数:"))` #输入班级人数

第6行

第7行	for everyMath in range(0,studCount):
第8行	nowScore=int(input("请输入数学成绩:"))
第9行	Math.append(nowScore)
第10行	
第11行	print(Math) #输出所有录入的数学成绩

第六节 选择语句

例程1-27正确执行的前提是起点数m小于终点数n。如果m大于n则结果为0。好的程序应该有一定宽容度，不宜太严苛，例程1-30是对例程1-27的升级。第6行是新知识点，if选择语句(Choice Statement)。**选择语句是指某些语句的执行需要满足一定条件，满足则执行，不满足则不执行。**如第7行，则必须满足m大于n这个条件，如果不满足该条件，则不执行，当满足该条件时则执行。在本程序中，如果起点数m大于终点数n，则两个数交换，n变成起点数，m变成终点数。

和for-in循环一样，**if语句必须以英文冒号结尾，且被控制的语句同样要采用缩进**，规则与for-in循环一样。

例程1-30

第1行	#从m开始到n为止的连续整数之和,range()应用示例
第2行	
第3行	m=int(input("输入起点数: "))
第4行	n=int(input("输入终点数: "))
第5行	
第6行	if m>n: #选择语句
第7行	m,n=n,m #m和n的值交换
第8行	
第9行	numSum=0
第10行	for i in range(m,n+1): #注意: n要加上1, 开区间
第11行	numSum+=i
第12行	
第13行	print(numSum)

例程1-31是例程1-27又一实现，**if语句有了else子句，其含义是当满足条件时执行if后的语句，否则执行else后的语句。**在本例，如果m>n则执行第7、8两行，否则执行第10、11行。注意else是与if同级的语句，且也必须以英文冒号结尾，其受控语句同样要缩进。注意观察第14行代码range(m,endNum,stepNum)。当stepNum为负数时，endNum要比n-1，如m=100，n=1，则计算的是100递减到1之间的所有整数的和，结束数比1小1，即为0，因此是n-1为endNum；如果m=1，n=100，则结束数是101，比n值大1。

选择语句的语法如下，其中类似m>n被称之为条件表达式。

if 条件表达式:

满足条件时执行的语句

else:

不满足条件时执行的语句

例程1-31

第1行	#从m开始到n为止的连续整数之和,range()应用示例
第2行	
第3行	m=int(input("输入起点数: "))
第4行	n=int(input("输入终点数: "))
第5行	
第6行	if m>n:
第7行	stepNum=-1
第8行	endNum=n-1 #如果步长是负数，则终点数比原数小1

```

第9行     else:
第10行     stepNum=1
第11行     endNum=n+1 #如果不长是正数，则总点数比原数大1
第12行
第13行     numSum=0
第14行     for i in range(m,endNum,stepNum): #注意：n要加上1，开区间
第15行         numSum+=i
第16行
第17行     print(numSum)

```

例程1-30和例程1-31代码m>n中的>是**关系运算符**中的一种，除了>（大于）外，还有>=（大于等于）、<（小于）、<=（小于等于）、==（相等）运算符，含义与数学中的运算符相同，但要注意<=等于不能写成≤，大于等于与之类似。尤其要注意=和==区别，前者用于赋值运算，用于将=右边的结果放到=左边的变量中，后者用于比较两侧表达式是否相等。**在if语句的条件表达式中不能出现=，但==可以出现，用于判断是否相等。**例程1-32是==的应用示例。第4行的代码n%5==0用于比较n除以5的余数与0相比是否相等，如果相等，则执行第5行否则执行第7行。

关系运算符的结果只能是True或False，与in和not in相同。当运算符成立时，结果为True，否则是False。

例程1-32

```

第1行     #==的使用
第2行
第3行     n=int(input("请输入一个整数："))
第4行     if n%5==0:
第5行         print(n,"能被5整除")
第6行     else:
第7行         print(n,"不能被5整除")

```

前述几个程序都是单条件判断，使用and和or则可以是多条件，如例程1-33所示。第5行判断数n能否同时被5和3整除，而第8行则是判断n能否被5或3中的一个整除。对于and，要求and前后的表达式满足条件，即前后表达式的结果均为True时，则结果为True否则False；对于or则只要满足其中一个即可，即前后表达式中有一个为True，结果为True，否则False。

例程1-33

```

第1行     #and和or的使用
第2行
第3行     n=int(input("请输入一个整数："))
第4行
第5行     if n%5==0 and n%3==0:
第6行         print(n,"能被5和3整除")
第7行
第8行     if n%5==0 or n%3==0:
第9行         print(n,"能被5或3整除")

```

在Python中，and和or被称为**逻辑运算符**。逻辑运算符共有3个，除了and和or外，还有not，相当于求反，观察例程1-34。对于True求反则为False，对于False求反则为True。

例程1-34

```

第1行     #and、or和not的使用
第2行
第3行     #列出能被3和5同时整除的数
第4行     for i in range(1,20):
第5行         if i%5==0 and i%3==0:
第6行             print(i)

```

第7行	#仅输出15
第8行	
第9行	for i in range(1,20):
第10行	if not(i%5==0 and i%3==0):
第11行	print(i)
第12行	#1-19, 除了15都会被列出
第13行	
第14行	for i in range(1,20):
第15行	if i%5==0 or i%3==0:
第16行	print(i)
第17行	#能被3或5整除的都将列出, 如3、5、6、9、10.....
第18行	
第19行	for i in range(1,20):
第20行	if not(i%5==0 or i%3==0):
第21行	print(i)
第22行	#列出不能被3和5整除的数, 如1、2、4、7、8、11.....

从例程1-34可以看出, for-in循环中可以有if选择语句, 可以成为循环嵌套选择。实际上循环可以嵌套循环, 也可以嵌套选择。同样, 选择可以嵌套选择, 也可以嵌套循环。

例程1-35能录入指定数量的数据(此处为数学成绩Math), 并能输出比平均成绩高的数据。第5行用于确定录入数据的数量, 第7-11行实现数据录入, 并获得录入数据的总和。第13行根据总数和数据数量求得平均值, 第16-18行则实现输出比平均值高的数据项。

例程1-35

```

请输入班级人数:8
请输入数学成绩:125
请输入数学成绩:98
请输入数学成绩:89
请输入数学成绩:87
请输入数学成绩:135
请输入数学成绩:69
请输入数学成绩:107
请输入数学成绩:80
125
135
107

```

图1-20 例程1-35执行结果

第1行	#列出比平均成绩高的成绩
第2行	
第3行	Math=[] #空list, 即没有成员的list
第4行	
第5行	studCount=int(input("请输入班级人数:")) #输入班级人数
第6行	
第7行	mathSum=0
第8行	for everyMath in range(0,studCount):
第9行	nowScore=int(input("请输入数学成绩:"))
第10行	mathSum+=nowScore #求得总分
第11行	Math.append(nowScore)
第12行	
第13行	avgScore=mathSum/studCount #求得平均分
第14行	
第15行	#输出比平均成绩高的成绩
第16行	for everyScore in Math:
第17行	if everyScore>avgScore: #如果比平均成绩高
第18行	print(everyScore)

第七节 print()函数

例程1-36的代码非常简单, 其执行效果如图1-21所示。从图中可以看出, 例程第3、4行输出内容是图片上部每个数值后换行的部分; 第6、7行是图片中部横向输出部分; 第9、10行输出是图片底部每个数值换行部分; 第12、13行为底部横向输出部分。为什么有如此区别呢? 比较代码可以看出, 差别由print()函数中end参数决定。

例程1-36

第1行	#更多认识print()函数
第2行	
第3行	for i in range(1,6): #1-10
第4行	print(i)
第5行	
第6行	for i in range(1,6):
第7行	print(i,end=",")
第8行	
第9行	for i in range(1,6):
第10行	print(i,end="\n")
第11行	
第12行	for i in range(1,6):
第13行	print(i,end="\t")

```

1
2
3
4
5
1,2,3,4,5,1
2
3
4
5
1      2      3      4      5

```

print()函数每次执行后默认换行，如需调整可通过end参数，即end参数用于设定print()函数执行后是否换行或紧接其他字符。如例程第7行所示，print()函数执行时将输出i值，输出后，紧接一个英文逗号而不是换行，下一次输出也是在逗号之后继续输出。

第9、10行执行后，为什么不是每个数字后跟随\n和n两个字符而换行呢？在Python以及C/C++/Java中，类似\n这种组合被称为**转义字符 (Escape Character)**，常用于表示常见但不能显示的字符，如\n(换行)、\t(制表符)。注意：**\n只是一个字符，而不是两个，因此print(len("\n"))将输出1，而不是2。**例程第13行执行后，相当于在每个输出数值后按下键盘左侧的TAB键。

图1-21 例程1-36执行结果

比较例程1-37第4、5行会发现第4行的每一项输出前都有一个空格，这是print()函数的默认。这种默认可以通过sep (separator的简写，分隔符) 参数进行调整，如例程第6行所示，sep被设置为空字符，即没有分隔符。第7行是设置\$为分隔符，执行后输出的每一项之前都有\$符号。

例程1-37

第1行	#更多认识print()函数
第2行	
第3行	a,b=10,20
第4行	print(a,"*",b,"=",a*b) #输出: 10 * 20 = 200
第5行	print("10*20=200") #输出: 10*20=200
第6行	print(a,"*",b,"=",a*b,sep="") #输出: 10*20=200
第7行	print(a,"*",b,"=",a*b,sep="\$") #输出: 10\$*\$20\$=\$200

例程1-37第6行的写法不直观友好，Python提供了其他多种解决方案，最新方案是f-string，如例程1-38所示第5行所示，和第4行相比，更加直观。在f-string中，花括号内是表达式，将计算出该表达式的值，替换所在位置。注意：**字符串前的f是必须**，如第6行所示，因为没有f则直接输出字符串内容。

例程1-38

第1行	#更多认识print()函数
第2行	
第3行	a,b=10,20
第4行	print(a,"*",b,"=",a*b,sep="") #输出: 10*20=200
第5行	print(f'{a}*{b}={a*b}') #输出: 10*20=200
第6行	print("{a}*{b}={a*b}") #输出: {a}*{b}={a*b}
第7行	
第8行	out1=f"a*b=a*b"
第9行	


```

out2=f"{a}*{b}={a*b}"
第10行 print(out1,out2) #输出: a*b=a*b 10*20=200
第11行 print("ABC"+out2+"XYZ") #输出: ABC10*20=200XYZ

```

观察例程第8、9行, 如果没有花括号, 就是普通字符串, 花括号是将被计算替换的位置, 所以第9行能达到预期。另外, f-string 就是字符串, 经过计算后形成的字符串。通过第11行能看出通过+运算符实现了字符串的拼接, 也从侧面证明out2为字符串。

第八节 字符串

字符串是程序设计语言最常用的数据类型。在Python可以用单引号、双引号、三单引号、三双引号创建, 如例程1-39所示。在单引号界定的字符串中可以有双引号; 在双引号界定的字符串中可以出现单引号; 在三单引号中, 可以出现双引号或单引号(不能是连续3个); 在三双引号中, 可以出现双引号(不能连续3个)或单引号。

例程1-39

```

第1行 #字符串
第2行 strA='Explicit is better than implicit.' #单引号
第3行 strB="Flat is better than nested." #双引号
第4行
第5行 #三引号, 3双引号
第6行 strC="""Beautiful is better than ugly.
第7行 Explicit is better than implicit.
第8行 Simple is better than complex.
第9行 Complex is better than complicated."""
第10行
第11行 #三引号, 3单引号
第12行 strD='''Beautiful is better than ugly.
第13行 Explicit is better than implicit.
第14行 Simple is better than complex.
第15行 Complex is better than complicated.'''
第16行
第17行 print(strA)
第18行 print(strB)
第19行 print(strC)
第20行 print(strD)

```

例程1-40(输出如图1-22所示)在三引号字符串中包含了学生成绩并用于求每个学生的平均成绩, 每行一个学生, 可以很多行, 此处作为示例仅包含了4行。第6行输出如图前4行所示, 之所以换行, 是因为在每行后有看不见但存在的\n换行符(或者说是之所以换行)。第8行字符串函数splitlines()用于将字符串按行切分为一个list类型数据以便于进一步处理, 每行是list的一个成员, 此处将切分结果存放在scoreRow之中, 从第9行输出可以看出, 已经变成list。

例程1-40

```

第1行 #字符串的函数
第2行 strScore="""张珊,98,89
第3行 李思,67,89
第4行 王武,99,85
第5行 刘柳,87,78"""
第6行 print(strScore)
第7行
第8行 scoreRow=strScore.splitlines() #按行切分
第9行 print(scoreRow)
第10行 #输出:['张珊,98,89','李思,67,89','王武,99,85','刘柳,87,78']

```

```

第11行
第12行 scorelist=[]
第13行 for everyData in scoreRow:
第14行     tmp=everyData.split(",") #在逗号处切分，形成list
第15行     print(tmp) #仅为了观察，可以删除
第16行     scoreTuple=(tmp[0],int(tmp[1]),int(tmp[2])) #将切分的结果转换后拼装为不可修改的tuple
第17行     scorelist.append(scoreTuple) #将切分转换后的结果添加到scoreList之中
第18行
第19行 print(scorelist) #观察转后的结果，符合预期
第20行 #输出： [('张珊',98,89),('李思',67,89),('王武',99,85),('刘柳',87,78)]
第21行
第22行 #输出每一个的平均值
第23行 for everyone in scorelist:
第24行     print(everyone[0],(everyone[1]+everyone[2])/2)

张珊,98,89
李思,67,89
王武,99,85
刘柳,87,78
['张珊',98,89,'李思',67,89,'王武',99,85,'刘柳',87,78]
['张珊','98','89']
['李思','67','89']
['王武','99','85']
['刘柳','87','78']
[('张珊',98,89),('李思',67,89),('王武',99,85),('刘柳',87,78)]
张珊 93.5
李思 78.0
王武 92.0
刘柳 82.5

```

图1-22 例程1-40执行结果

将数据切分到能方便处理的单元，将更好。例程第13行的循环，将对每个成绩行进行切分并进一步处理。例程中的split()用于对字符串按指定字符进行切分，此处用逗号切分。切分后的结果在第15行输出，可以发现切分后为list类型。第16行将切分后编号为1和2的成员转换为整数后与编号为0的成员一起构成tuple，第17行将其添加到scorelist之中，第19行输出转换后的结果，会发现姓名仍然是字符串，而两项成绩已经是数值。第23、24行输出每名学生对应的平均成绩。

例程1-41是字符串其他一些常用函数。count()用于统计字符串中指定字符串出现的次数，例程代码strTest.count("莲")即统计"莲"在strTest中出现的次数；find()函数用于查找字符串在指定字符串中出现的位置，如果没有找到，则返回值为-1，例程代码strTest.find("田田")则是查找"田田"在strTest中出现的位置；zfill()函数用于根据指定长度在字符串前填充0，一般用于数据项对齐，"123".zfill(5)构造5个字符的长度，如果长度不足则补充0，"123"长度为3，补充两个0不足到长度5；strip()用于清除字符串前后的空格，对字符串中间的空格不影响；upper()和lower()用于将字符串变成全部大写或全部小写；replace()函数用于字符串替换，即用新的字符替换字符串中原有某些字符，myStr.replace("计算机","电脑")即将myStr字符串中的"计算机"替换为"电脑。"

例程1-41

```

第1行 #字符串的函数
第2行 strTest="江南可采莲，莲叶何田田。鱼戏莲叶间。鱼戏莲叶东，鱼戏莲叶西，鱼戏莲叶南，鱼戏莲叶北。"
第3行 print(strTest.count("莲")) #输出： 7
第4行 print(strTest.find("田田")) #输出： 9
第5行
第6行 print("123".zfill(5)) #输出： 00123
第7行 print(" ABC ".strip())#输出： ABC
第8行
第9行 print("XyZ".upper()) #输出： XYZ

```

```

第10行 print("XyZ".lower()) #输出: xyz
第11行
第12行 myStr="计算机在中国越来越普遍, 改革开放以后, 计算机用户不断攀升。"
第13行 print(myStr.replace("计算机","电脑"))
第14行 #电脑在中国越来越普遍, 改革开放以后, 电脑用户不断攀升。
第15行
第16行 print(myStr) #输出: 计算机在中国越来越普遍, 改革开放以后, 计算机用户不断攀升
第17行
第18行 afterStr=myStr.replace("计算机","电脑")
第19行 print(afterStr)
第20行 #输出: 电脑在中国越来越普遍, 改革开放以后, 电脑用户不断攀升。
第21行
第22行 print(myStr)
第23行 #输出: 计算机在中国越来越普遍, 改革开放以后, 计算机用户不断攀升。

```

第13行执行后, 输出结果如第14行所示, 不过执行第16行会发现myStr并没有发生改变。字符串和tuple一样不可更改。replace()函数替换后的结果将形成新的字符串而不是原字符串, 如例程第18-23行所示。zfill()、lower()、upper()与之类似。

例程1-42是对装有str字符串成员排序。例程第2行是原始数据顺序, 第5行是排序后结果, 从中可以看出, 数字在前, 大写字母在其后, 小写字母跟随, 最后是汉字。每个字符在Python中都有编号, 可以用ord()函数(order的简写)获取, 排序就是以该编号为准, 如例程第7行所示。字符串排序时, 如果第1个字符相同, 则比较第2个字符, 以此类推。如果一个字符串是另外一个字符串的一部分, 则短字符串在前, 长在后。

例程1-42

```

第1行 #字符串的排序
第2行 strlist=["0123","012","ABCD","ABX","abcd","abc","9876","李白","杜甫","李商隐"]
第3行 strlist.sort()
第4行 print(strlist)
第5行 #输出: ['012','0123','9876','ABCD','ABX','abc','abcd','李商隐','李白','杜甫']
第6行
第7行 print(ord("0"),ord("1"),ord("A"),ord("a"),ord("李"),ord("杜"),sep=",")
第8行 #输出: 48,49,65,97,26446,26460
第9行
第10行 print(chr(97)) #输出: a
第11行
第12行 for i in range(ord("a"),ord("z")+1):
第13行     print(chr(i),end=" ")
第14行 #输出: abcdefghijklmnopqrstuvwxyz

```

字符在Python中采用Unicode编码。Unicode是对全球字符规定了唯一编码。英文字符包括数字、大写字母和小写字母以及一些控制字符, 采用与ASCII(American Standard Code for Information Interchange, 美国信息交换标准码)相同编号, 在该编码体系中, 数字0-9的编码依次顺序对应是48-57, A-B是65-90, a-z是97-122。基本汉字的范围编码界于19968(对应是一)-40869(对应是龠), 基本以笔划为序, 总计20902个汉字。函数chr()的功能是根据计算出整数对应的字符, 如第10行所示。第12、13行则是利用ord()和chr()生成a到z之间的全部字符。

第九节 文件读写

前列程序的运行结果都不能保存到文件, 当程序运行结束关闭计算机后, 结果也就完全消失。典型的程序设计语言都提供了写数据到文件使之持久存储, 或者从文件读取数据以供程序的能力。先来看看将数据写入到文件。例程1-43是将1-100之间所有能被3整除的数写入到文件3.txt之中, 图1-23是其文件用记事本软件打开后显示效果, 可以看出数据已经写入到了文件。

例程1-43

```

第1行 #数据写入到文件，将1-100中能被3整除的数写到文件3.txt之中。
第2行
第3行 myFile=open("3.txt","w") #打开文件用于写数据
第4行
第5行 for i in range(1,101):
第6行     if i%3==0:
第7行         print(i,file=myFile) #写入文件
第8行
第9行 myFile.close() #关闭文件，文件用完后一定要关闭

```

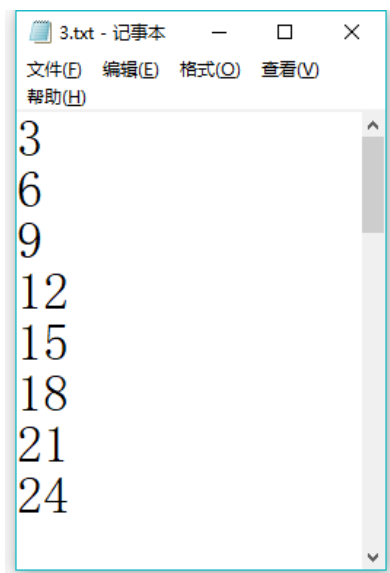


图1-23 例程1-43执行结果

与文件相关的操作，基本流程是三步骤：打开文件→操作文件→关闭文件。在例程1-43中，第3行是打开文件，第7行是操作文件，第9行是关闭文件。

打开文件时，第1个参数文件名称(包括文件夹等位置数据)，第2个参数是打开方式。在例程1-43中，3.txt是文件的名称，由于没有指明位置，因此该文件位于当前位置，即Python程序所在的位置。第2个参数可以是r或w，r表明程序将从文件读取数据，w则表示从文件读取数据到程序。例程1-43第3行打开文件的方式是w，因此是将程序运行的数据写入到文件。

第7行是熟悉的print()函数，一般说来print()是将程序运行结果输出到屏幕，但是当设置file参数时，则将输出到指定的文件。注意：file=后的不是文件名称，而是文件类型的变量名称，在例程中是myFile，由open()参数生成。当有了该参数后，程序将不向屏幕输出，而是输出到文件。要查看输出是否正确，需要用文本编辑器查看该文件，例如：记事本等。用于编写Python程序的IDLE也可以打开文本文件，选择“File”菜单下的“Open”，会弹出如图1-24的对话框，注意右下角，选择合适的文件类型，默认是打开Python程序文件，选择*.txt即可打开3.txt。另外，还要注意文件的位置，即文件在哪个文件夹下。

例程1-43第9行是关闭文件。注意close()函数是隶属文件变量的函数，因此必须在前面加上myFile或者类似的文件变量名。

注意：图1-23没有全部显示，仅仅是数据的一部分。

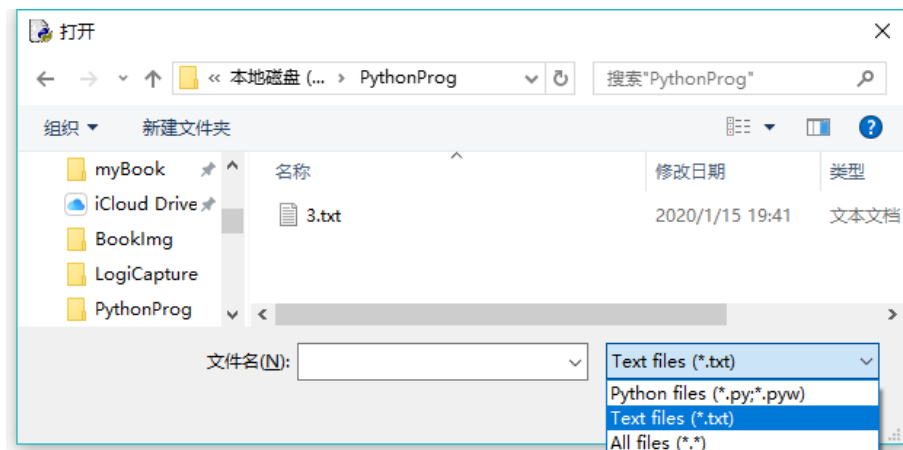


图1-24 在IDLE中打开文本文件

例程1-44是文件内容读取程序。第3行open()函数中的“r”参数表示文件打开用于读取。第5行代码中的myFile.read()用于读取文件全部内容，读出的内容为字符串。第9行用于输出字符串的前5个字符。但从图1-25可以看出仅3个字符，原因是每个字符后有换行的\n（注意：是1个字符），即内容实为3\n6\n9，9之后由于输出已经结束，所以没有\n。

```

3
6
9

```

图1-25 例程1-44执行效果

例程1-44

```

第1行 #从文件读取数据
第2行
第3行 myFile=open("3.txt","r") #打开文件用于读数据
第4行
第5行

```

```

第6行 dataFromFile=myFile.read() #从文件读取，读出内容是字符串
第7行
第8行 myFile.close() #关闭文件，文件用完后一定要关闭
第9行 print(dataFromFile[:5])

```

例程1-45是将读取的内容根据行，且分为list，此时就可以按list进一步处理。splitlines()函数是字符串的函数，用于将字符串按分行符进行切分。文本文件之所以能分行，其原因是在每行内容末尾有分行的控制符。

例程1-45

```

第1行 #从文件读取数据
第2行
第3行 myFile=open("3.txt","r") #打开文件用于读数据
第4行
第5行 dataFromFile=myFile.read() #从文件读取，读出内容是字符串
第6行
第7行 myFile.close() #关闭文件，文件用完后一定要关闭
第8行
第9行 datalist=dataFromFile.splitlines()
第10行 print(datalist)

```

['3', '6', '9', '12', '15', '18', '21', '24', '27', '30', '33', '36', '39', '42', '45', '48', '51', '54', '57', '60', '63', '66', '69', '72', '75', '78', '81', '84', '87', '90', '93', '96', '99']

图1-26 例程1-45执行效果

文件内容，本为整数，但现在仍然为字符串，带引号为证，如何转变呢？如例程1-46所示。从图1-27可以看出，list中的数字字符串已经变成整数，可以根据需要进一步处理。在例程第13行，list中每个成员通过int()函数变成整数。idx在每次循环中增加1，依次对应datalist中的每个成员。

例程1-46

```

第1行 #从文件读取数据
第2行
第3行 myFile=open("3.txt","r") #打开文件用于读数据
第4行
第5行 dataFromFile=myFile.read() #从文件读取，读出内容是字符串
第6行
第7行 myFile.close() #关闭文件，文件用完后一定要关闭
第8行
第9行 datalist=dataFromFile.splitlines()
第10行
第11行 idx=0
第12行 for i in datalist:
第13行     datalist[idx]=int(i)
第14行     idx+=1
第15行
第16行 print(datalist)

```

[3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 63, 66, 69, 72, 75, 78, 81, 84, 87, 90, 93, 96, 99]

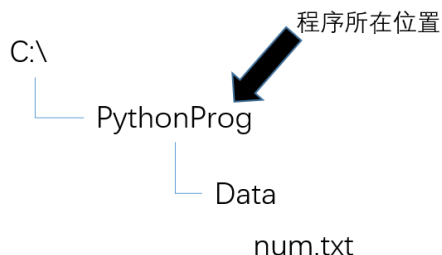
图1-27 例程1-46执行效果

例程1-47是读取含中文的文件。相对于全是英文字母或数字的文件，open()函数多了encoding参数，该参数有多个取值，暂都设置为utf-8，以后会详解本参数。本书提供的所有文件，都采用utf-8编码。

例程1-47

```
第1行 #读入中文内容
第2行
第3行 myFile=open("YueYangLouji.txt","r",encoding="utf-8") #读入《岳阳楼记》
第4行
第5行 content=myFile.read()
第6行 myFile.close()
第7行
第8行 print(content)
第9行
第10行 #统计句号出现次数
第11行 charCount=0
第12行 for everyone in content:
第13行     if everyone=="。":
第14行         charCount+=1
第15行
第16行 print(charCount)
```

前述打开的文件都没有文件夹或者目录，例程1-48第3行打开位于C盘PythonProg文件夹下Data文件夹下的num.txt文件，在Windows操作系统(包括DOS操作系统)下该文件可以表示为C:\PythonProg\Data\num.txt，其中C:表示盘符，第1个反斜杠(backslash)表示根文件夹(root folder，每个盘符有且仅有一个)，其他反斜杠是分隔符，用于分隔文件夹之间或文件夹与文件之间。



在Python中，字符串中的\表示开始转义字符，为了得到\，必须采用\\。另外，open("C:\\PythonProg\\Data\\num.txt","r")与open(r"C:\PythonProg\Data\num.txt","r")等价，注意字符串前的r，表示raw string(原始字符串)，有此标记Python将不对字符串转义。

文件定位有两种方式：绝对定位和相对定位。绝对定位从根文件夹开始，逐级定位到指定文件，而相对定位则是从当前文件夹开始。

C:\PythonProg\Data\num.txt是绝对定位，而open(r>Data\num.txt","r")中的Data\num.txt则是相对定位（注：程序位于C:\PythonProg之下，即当前文件夹是PythonProg，如左图1-28所示），表示从当前文件夹(此处为C:\PythonProg)开始

图1-28 例程1-48中num.txt文件所在位置

寻找到名为Data的子文件夹，然后在其中定位num.txt。open(r"\Data\num.txt","r")中\Data\num.txt中的\表示从当前文件夹(此处为C:\PythonProg)，Data位于当前文件夹之下。open(r"..PythonProg\Data\num.txt")中..\PythonProg\Data\num.txt的..表示父文件夹（在本例则定位到根文件夹），然后在其下寻找PythonProg然后Data然后num.txt。

例程1-48

```
第1行 #Python中的盘符、文件夹和文件
第2行
第3行 dataFile=open("C:\\PythonProg\\Data\\num.txt","r") #打开文件
第4行 dataFromFile=dataFile.read() #读取文件
第5行 dataFile.close() #关闭文件
第6行
第7行 data=dataFromFile.splitlines()#按行切分
第8行 dataCount=len(data)
第9行 for i in range(0,dataCount):
第10行     data[i]=int(data[i]) #将数据成员转换为整数
第11行
```


第12行	#求中位数，必须先排序
第13行	data.sort()
第14行	
第15行	#根据偶数个数和奇数个数分别处理
第16行	#如果是偶数，则对中间两个数求平均，
第17行	#如果是奇数，则中间数为中位数
第18行	
第19行	if dataCount%2==0:#偶数个数
第20行	firstSeatNo=dataCount//2-1 #第1个的编号
第21行	secondSeatNo=firstSeatNo+1 #第2个数的编号
第22行	median=(data[firstSeatNo]+data[secondSeatNo])/2
第23行	else: #奇数个数
第24行	seatNo=(dataCount-1)//2
第25行	median=data[seatNo]
第26行	
第27行	print(median) #输出中位数

第十节 再说函数

在Python中，可以将函数分为类外函数(outer-class function)和类中函数(inner-class function)，C++和Java等语言与之类似。类中函数依赖类而存在，如append()函数，依赖list类而存在，调用时通过list类的对象。不同的类有不同的函数，如append()依赖于list类，但str字符串类并没有该函数。类中函数调用时，必须有类的对象名称和英文句点表示隶属关系，而类外函数则没有，如例程1-49所示。

在例程中，append()是类中函数，调用时函数名称前必须有类(list)的对象名称Math或Poet和表示隶属关系运算符英文句点，如例程第5、9行所示；print()和len()函数是类外函数不是类中函数，因此调用时前面没有对象名称和表示隶属关系的英文句点。

例程1-49

第1行	#更多了解函数
第2行	
第3行	Math=[12,34]
第4行	print(len(Math)) #输出： 2
第5行	Math.append(56)
第6行	print(Math,len(Math)) #输出： [12, 34, 56] 3
第7行	
第8行	Poet=["李白","杜甫","王维","杜牧"]
第9行	Poet.append("白居易")
第10行	print(Poet) #输出： ['李白', '杜甫', '王维', '杜牧', '白居易']

在本章，先后学习了输入函数input()、输出函数print()、求成员数量的函数len()、转换为整数的函数int()、文件打开函数open()，这些都是类外函数；list的函数append()用于增加成员、字符串的函数splitlines()用于按行切分、文件的函数read()读取文件内容为字符串、文件的函数close()用于关闭已打开的文件，这些都是类中函数，其中append()属于list类，splitlines()隶属str类，read()和close()隶属文件类。

Python是面向对象程序设计(Object Oriented Programming，简称OOP)语言，类(class)、对象(object)、方法(method)等是其基本概念。list是类，在例程1-49中，Math和Poet是对象。类是抽象，对象是具象，如同马和白马黑马的关系，马是白马黑马的抽象，白马黑马是马的具象，对象是类的实例化。在Python中，list是类，Math和Poet是对象，是list的具象和实例化。方法是类中定义的函数，不同的类有不同的函数。append()是list类的方法，是其中定义的函数。

除了使用Python已有的类中函数和类外函数外，还可以根据需要自定义，例程1-50是类外函数的定义。**函数三要素是名称、参数和返回值(函数执行后结果)**。在例程中，def是自定义函数的关键字，表明将开始自定义函数，其结尾必须是英文冒号。ADD是函数名称，M和N是函数参数，表明调用该函数需要传入两个参数，如第8、10、12、14行所示。函数处理过程是简单的第5行，第6

行的return语句返回函数的处理结果，表明函数执行后的结果。如在第8行，ADD(3,2)执行后将得到结果5(经过第5行处理，然后由第6行返回处理后的结果)，然后用print()函数输出。

例程1-50

第1行	#自定义函数
第2行	
第3行	#M和N是函数的参数，ADD是函数的名称
第4行	def ADD(M,N): #英文冒号结尾是必须
第5行	Result=M+N
第6行	return Result #返回函数运行结果
第7行	
第8行	print(ADD(3,2)) #输出： 5
第9行	
第10行	ADD(3,2)
第11行	
第12行	print(ADD(3,2)+ADD(4,5)) #输出： 14
第13行	
第14行	GoGo=ADD(3,2)+3
第15行	print(GoGo) #输出： 8

第10行虽然调用了ADD()函数，但由于该函数的返回结果没有被其他函数接收(如第8行)、或赋值给其他变量(如第14行)、或成为表达式的一部分（如第12、14行），因此虽会被执行，但因为没有输出，不能直接观察到结果。

函数可以没有返回值，如例程1-51所示(执行效果如图1-29所示)，由于自定义printRow()函数没有return语句，因此不会有返回值。在自定义函数中，只要没有执行到return语句，就不会有返回值。没有返回值的函数，如果作为参数传递给print()函数，将输出None，即没有返回值的函数，其函数返回值为None，视为没有返回值。注意：自定义函数如执行print()函数，虽然有输出，但不是返回值，返回值一定是通过return语句返回的值。

例程1-51




图1-29 例程1-51执行效果

第1行	#自定义函数
第2行	
第3行	def printRow(N):
第4行	print(" *(10-N-1),"*"*N,sep="")
第5行	
第6行	for i in range(1,10,2):
第7行	printRow(i)
第8行	
第9行	print(printRow(3))

例程1-52涉及自定义类、类中函数以及对象等，例程第3-17行自定义一个名为cnID(中国身份证号)的类，第19、25行将根据cnID类分别生成对象名为myID和herID，在例程第21、22行和第27、28行分别调用类中函数getGender()和getBirthday()。

例程1-52

第1行	#类及其函数
第2行	
第3行	class cnID:
第4行	id="" #
第5行	
第6行	def __init__(self,ID): #初始化函数
第7行	self.id=ID
第8行	

```

第9行    def getGender(self):
第10行    genderNum=int(self.id[16]) #第17位为偶数是女性，奇数男性
第11行    if genderNum%2==0:
第12行        return "女"
第13行    else:
第14行        return "男"
第15行
第16行    def getBirthday(self):
第17行    return self.id[6:14]
第18行
第19行    myID=cnID("511725200010281575") #将自动调用__init__()函数
第20行
第21行    print(myID.getBirthday()) #输出：20001028
第22行    print(myID.getGender()) #输出：男
第23行    print(myID.id) #输出：511725200010281575
第24行
第25行    herID=cnID("110108199912280248") #将自动调用__init__()函数
第26行
第27行    print(herID.getBirthday()) #输出：19991228
第28行    print(herID.getGender()) #输出：女
第29行    print(herID.id) #输出：110108199912280248

```

cnID是类，myID和herID是根据cnID生成的对象，getGender()和getBirthday()是定义在cnID中的函数(也可以称为方法)，调用函数时，必须指明根据其类生成的对象，即myID或是herID。另外，第4行的id可以称为类的属性(property)，使用属性值时，没有括号，如例程第23、29行所示，这是和函数的不同。

类中函数的定义与类外函数大致相同，但通常第1个参数是self，表示自身，调用时由系统自动传递，不能包含该参数，因此虽然getGender()和getBirthday()虽然定义时都有1个self参数，但调用时没有参数。另外，__init__()是类的特殊函数(初始化函数)由Python约定，名称必须是init前后各有两个下横线，当类生成对象时，系统将自动调用该函数。

类中函数同样可以根据需要定义没有返回值的函数。有关类的更多信息，参加本书其他章节，本章更多是为了更好地理解各种类型的函数。

函数是程序片段，完成一定功能，通常带有入口和出口。入口即参数，供函数处理；出口即函数值或函数的返回值，返回供调用的程序继续处理。函数是模块化程序设计的基础，通常将大的任务分解成若干模块，每个模块用函数实现，降低程序设计的复杂性，使得程序易于维护和功能扩充，缩短开发周期。

例程1-53用于实现如图1-30所示的效果(不包括红框)，由于是两层循环，且内层循环嵌套有选择语句，对初学者有一些复杂。通过观察会发现，一共有9行，每行根据行的不同，输出不同内容。

```

1*1= 1
1*2= 2 2*2= 4
1*3= 3 2*3= 6 3*3= 9
1*4= 4 2*4= 8 3*4=12 4*4=16
1*5= 5 2*5=10 3*5=15 4*5=20 5*5=25
1*6= 6 2*6=12 3*6=18 4*6=24 5*6=30 6*6=36
1*7= 7 2*7=14 3*7=21 4*7=28 5*7=35 6*7=42 7*7=49
1*8= 8 2*8=16 3*8=24 4*8=32 5*8=40 6*8=48 7*8=56 8*8=64
1*9= 9 2*9=18 3*9=27 4*9=36 5*9=45 6*9=54 7*9=63 8*9=72 9*9=81

```

图1-30 例程1-53执行效果

例程1-53

```

第1行    #函数-模块化程序设计的基础
第2行
第3行

```

```

    for row in range(1,10):
第4行    for col in range(1,row+1):
第5行        if col*row<10:
第6行            print(f"{col}*{row}={row*col}",end=" ") #多一个空格, 为了对齐
第7行        else:
第8行            print(f"{col}*{row}={row*col}",end=" ")
第9行    print()

```

根据前述理解, 可以将代码函数化, 如例程1-54所示。这个程序能被执行, 虽然执行后似乎什么也没有干, 因为没有输出。例程第3行的pass是占位符, 没有实际作用, 但定义函数必须有函数体, 否则结构不完整。如果将pass修改为print(Hang), 将会输出1、2、3、.....、7、8、9。在函数定义中, 定义函数的参数称为形式参数(如第2行), 调用函数的参数称为实际参数(如第6行)。函数调用时, 实际参数值将传递给形式参数, 在本例, 将row的值传递给Hang, 由于循环执行9次, 因此printRow()将被执行9次, 依次传入1、2、3、.....、7、8、9。

例程1-54

```

第1行    #函数-模块化程序设计的基础
第2行    def printRow(Hang):
第3行        pass #pass占位符
第4行
第5行    for row in range(1,10):
第6行        printRow(row)

```

观察最终结果图, 会发现每行输出内容与行密切相关。第1行输出1个等式, 第2行输出2个等式、第3行输出3个等式, 以此类推, 可以将其理解为行内循环次数与所在行相同, 且等式由行和循环次数组成。基于此, 将代码修改如例程1-55所示, 输出如图1-31左侧标志为①的部分。从图中可以看出, 数据变化已经符合预期, 但格式(换行)仍然差别很大。

例程1-55

```

第1行    #函数-模块化程序设计的基础
第2行    def printRow(Hang):
第3行        for i in range(1,Hang+1):
第4行            print(f"{i}*{Hang}={i*Hang}")
第5行
第6行    for row in range(1,10):
第7行        printRow(row)

```

```

1*1=1      1*1=11*2=22*2=41*3=32*3=63*3=91*4=42*4=83*4=124*4=161*5=52*5=103*
1*2=2      5=154*5=205*5=251*6=62*6=123*6=184*6=245*6=306*6=361*7=72*7=143*7
2*2=4      =214*7=285*7=356*7=427*7=491*8=82*8=163*8=244*8=325*8=406*8=487*8
1*3=3      =568*8=641*9=92*9=183*9=274*9=365*9=456*9=547*9=638*9=729*9=81 ②
2*3=6      1*1=1
3*3=9      1*2=22*2=4
1*4=4      1*3=32*3=63*3=9
2*4=8      1*4=42*4=83*4=124*4=16 ③
3*4=12     1*5=52*5=103*5=154*5=205*5=25
4*4=16     1*6=62*6=123*6=184*6=245*6=306*6=36
1*5=5      1*7=72*7=143*7=214*7=285*7=356*7=427*7=49
①          1*8=82*8=163*8=244*8=325*8=406*8=487*8=568*8=64
           1*9=92*9=183*9=274*9=365*9=456*9=547*9=638*9=729*9=81

```

图1-31 程序演变

换行属于函数print()的功能, 其参数end能设置是否换行, 如果其值为\n则如同默认每次执行print()函数都将换行, 如果其值为""即空字符串, 则每次都不换行。图1-31第2部分的代码如例程1-56所示。

例程1-56

```

第1行 #函数-模块化程序设计的基础
第2行 def printRow(Hang):
第3行     for i in range(1,Hang+1):
第4行         print(f"{i}*{Hang}={i*Hang}",end="")
第5行
第6行     for row in range(1,10):
第7行         printRow(row)

```

每次输出都换行和都不换行，都不符合预期，应该是每行换行一次。在第7行后增加print()可以实现，其含义是当执行完毕printRow()函数后，执行print()可以实现换行，效果如同图1-31标志的第3部分。也可以在第4行增加print()，如例程1-57所示。

例程1-57

```

第1行 #函数-模块化程序设计的基础
第2行 def printRow(Hang):
第3行     for i in range(1,Hang+1):
第4行         print(f"{i}*{Hang}={i*Hang}",end="")
第5行         print()
第6行
第7行     for row in range(1,10):
第8行         printRow(row)

```

注意：如果例程1-57第5行与第4行对齐，则每次循环后都将换行，其效果如图1-31标志的第1部分，相当于和第4行一样，都受第3行循环控制。而如果和第3行对齐，则循环结束后采用执行第5行，基本达到预期。

密密麻麻，还不符合预期，可在{i*Hang}之后增加一个空格，执行后输出如图1-32所示，已基本符合预期，但会发现3*4=12与3*5=15没有对齐，仔细观察会发现是2*4=8与2*5=10一个是一位数，一个是两位数不同导致，在结果为一位数之前增加一个空格，即可实现对齐，即如果结果为一位数即增加一个空格，否则直接输出两位数，这个用选择语句即可实现，如例程1-58所示。

```

1*1=1
1*2=2 2*2=4
1*3=3 2*3=6 3*3=9
1*4=4 2*4=8 3*4=12 4*4=16
1*5=5 2*5=10 3*5=15 4*5=20 5*5=25
1*6=6 2*6=12 3*6=18 4*6=24 5*6=30 6*6=36
1*7=7 2*7=14 3*7=21 4*7=28 5*7=35 6*7=42 7*7=49
1*8=8 2*8=16 3*8=24 4*8=32 5*8=40 6*8=48 7*8=56 8*8=64
1*9=9 2*9=18 3*9=27 4*9=36 5*9=45 6*9=54 7*9=63 8*9=72 9*9=81

```

图1-32 增加空格后输出

例程1-58

```

第1行 #函数-模块化程序设计的基础
第2行 def printRow(Hang):
第3行     for i in range(1,Hang+1):
第4行         if i*Hang<10:
第5行             print(f"{i}*{Hang}={i*Hang} ",end="")
第6行         else:
第7行             print(f"{i}*{Hang}={i*Hang} ",end="")
第8行         print()
第9行
第10行     for row in range(1,10):
第11行         printRow(row)

```

例程1-58与例程1-53稍有不同。例程1-53用end参数控制每个等式后的空格，而例程1-58是在f-string中直接控制，均可。

从上述例程可以看出，函数的引入，将两层循环变成一层循环，且第一层循环仅控制行的变化，printRow()函数中仅控制行内循环，控制的粒度变小，编程的难度降低。将复杂任务分解为多个可控小模块，是一种思维训练。同时，函数之间的输入(参数)和输出(返回值)可以理解为任务间的关系。一个函数的输出，可以成为另外一个函数的输入。很多现实中的任务，也大多如此。这种思维不仅仅适用于程序设计，也常适用于现实工作。

图1-33是九九乘法表的又一形式，1*1= 1之所以能到左边，是因为前面有很多空格，图1-34是用-(减号)而不是空格，对比能更好理解。从图中可以看出，空格是逐行减少，每次减少7个，该效果实现如例程第3行所示。输出空格后由于不能换行，因此需要使用end参数。

```

                                1*1= 1
                                1*2= 2 2*2= 4
                                1*3= 3 2*3= 6 3*3= 9
                                1*4= 4 2*4= 8 3*4=12 4*4=16
                                1*5= 5 2*5=10 3*5=15 4*5=20 5*5=25
                                1*6= 6 2*6=12 3*6=18 4*6=24 5*6=30 6*6=36
                                1*7= 7 2*7=14 3*7=21 4*7=28 5*7=35 6*7=42 7*7=49
                                1*8= 8 2*8=16 3*8=24 4*8=32 5*8=40 6*8=48 7*8=56 8*8=64
                                1*9= 9 2*9=18 3*9=27 4*9=36 5*9=45 6*9=54 7*9=63 8*9=72 9*9=81

```

图1-33 九九乘法表又一形式

```

-----1*1= 1
-----1*2= 2 2*2= 4
-----1*3= 3 2*3= 6 3*3= 9
-----1*4= 4 2*4= 8 3*4=12 4*4=16
-----1*5= 5 2*5=10 3*5=15 4*5=20 5*5=25
-----1*6= 6 2*6=12 3*6=18 4*6=24 5*6=30 6*6=36
-----1*7= 7 2*7=14 3*7=21 4*7=28 5*7=35 6*7=42 7*7=49
-----1*8= 8 2*8=16 3*8=24 4*8=32 5*8=40 6*8=48 7*8=56 8*8=64
-----1*9= 9 2*9=18 3*9=27 4*9=36 5*9=45 6*9=54 7*9=63 8*9=72 9*9=81

```

图1-34 -(减号)替换空格后的显示效果

例程1-59

第1行	#函数-模块化程序设计的基础
第2行	def printRow(Hang):
第3行	print(" *(9-Hang)*7,end="")#每行前先输出一定数量的空格
第4行	for i in range(1,Hang+1):
第5行	if i*Hang<10:
第6行	print(f" {i}*{Hang}= {i*Hang} ",end="")
第7行	else:
第8行	print(f" {i}*{Hang}= {i*Hang} ",end="")
第9行	print()
第10行	
第11行	for row in range(1,10):
第12行	printRow(row)

函数应用场景很多，例程1-60是示例在list对象的sort()函数中应用。例程第2-5行前面已经学习过，第11行的sort()函数中有key参数，用于指定排序方法，其值为函数名称（注意：如此场景函数名称后一定不能有圆括号），其功能是将numlist的每个成员放入Mod10()运算，即numlist的每个成员值传递到函数参数N中进行计算，最后根据函数的运行结果排序。从例程第12行的输出可以看出，除以10后的余数分别是0、1、1、2、3、8、9，按升序排列。注意：key参数仅指定排序规则，不改变值本身。如同按身高排序，排序的依据是身高，排序的结果是人的先后顺序，不改变人本身。

例程1-60

第1行	#函数在sort()函数中的应用
第2行	numlist=[1,3,100,212,11,28,29]
第3行	numlist.sort()

第4行	
第5行	print(numlist)#输出: [1, 3, 11, 28, 29, 100, 212]
第6行	
第7行	def Mod10(N): #求除以10的余数
第8行	rtn=N%10
第9行	return rtn
第10行	
第11行	numlist.sort(key=Mod10)
第12行	print(numlist)#输出: [100, 1, 11, 212, 3, 28, 29]

例程1-61仍然是函数在排序函数sort()中的应用。传入AVG()函数的是每个scorelist成员，其为tuple，第3行利用该tuple编号为1和2的成员求的平均值。sort()用平均值作为排序依据，且设置降序(按默认的反序)。

例程1-61

第1行	#函数在sort()函数中的应用
第2行	def AVG(score): #求平均值
第3行	return (score[1]+score[2])/2
第4行	
第5行	scorelist=[("张珊",98,89),("李思",67,89),("王武",99,85),("刘柳",87,78)]
第6行	
第7行	scorelist.sort(key=AVG,reverse=True)
第8行	print(scorelist)
第9行	#输出: [('张珊',98,89),('王武',99,85),('刘柳',87,78),('李思',67,89)]

例程1-60和例程1-61的函数都比较简单，专门构造一个函数，似乎有些小题大做，更简洁的方式如例程1-62所示，其效果与例程1-61完全相同。代码第4行lambda x:(x[1]+x[2])/2的地位与例程1-61第7行的AVG完全相同，功能也相同，就是求得平均值。这种函数被称为**lambda函数**，又称**匿名函数**，不少程序设计语言都支持，如：C++、C#、Java等。

在Python的lambda函数中，可以有1个或多个参数，但没有括号，如在例程代码lambda x中，x是参数，冒号后的(x[1]+x[2])/2是返回值。Python语言规定，lambda函数的冒号后有且只能有一个表达式，该表达式的值就是函数的返回值。如果函数逻辑较为复杂不是一个表达式所能表达，则可以采用普通函数。

例程1-62

第1行	#lambda函数在sort()函数中的应用
第2行	scorelist=[("张珊",98,89),("李思",67,89),("王武",99,85),("刘柳",87,78)]
第3行	
第4行	scorelist.sort(key=lambda x:(x[1]+x[2])/2,reverse=True)
第5行	print(scorelist)
第6行	#输出: [('张珊',98,89),('王武',99,85),('刘柳',87,78),('李思',67,89)]

在Python程序设计中，lambda函数应用相当广泛，后续章节将能看到更加广泛的应用。

标识符及其命名

标识符(identifier)是指用来标识某个实体的符号，在程序设计语言中，是用于给变量、函数、类、对象等命名，以建立起名称与实体之间的关系。标识符通常由字母和数字以及其它字符构成，是计算机语言允许用作名字的有效字符序列。不同程序设计语言虽有不同规定，但大致相似。在Python，标识符必须遵守如下强制规则：

- 标识符以大写字母、小写字母或者下横线、汉字等开始；
- 除首字符外，其余字符可以由大写字母、小写字母、下划线、汉字以及0-9之间的字符组成；
- 大写字母和小写字母的不同组合代表的不同标识符，即大小写敏感；
- 不能使用关键字作为自定义名称标识符，Python关键字有False、None、True、and、as、assert、async、await、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；
- Python对标识符长度没有限制；

强制规则不能违背。除此以外，还建议遵守以下规范，不仅写程序当时明白，也能让编程人较长时间后能更好明白或有助于关系人快速理解。

- 使用有意义体现功能的名称，除循环外，不使用单字符标识符。如：setStudentID或者setID其功能浅显易懂；
- 使用便于阅读的字符序列，如getusername明显不如getUsername易读，前者需要断字，后者明显能区分单词组合；
- 遵守语言建议规范，很多语言都有自己约定俗成的规范；
- 在同一个代码单元内，不要让相同标识符承担不同的功能；
- 不要使用非ASCII字母命名，虽然Python支持汉字作为标识符的名称(注：Python2不支持汉字做标识符)；
- 服从开发组织制定的各种开发规范。

标识符命名有诸多约定俗成的方法，在演进中逐步形成了3种经典命名法，分别是：匈牙利命名法、骆驼命名法、帕斯卡命名法。

- **匈牙利命名法(Hungarian Notation)**。该命名法每个标识符开始若干表示类型的字符，主要用于微软产品。基本原则是：标识符=属性+类型+标识符本体描述。如i表示int，所有i开头的变量都表示int类型。s或str表示String、b表示布尔型、arr表示数组、fun表示函数、cls表示类、obj表示对象、p或者ptr表示指针、ref表示引用等。另外，还可以加上u表示无符号、l表示长型、s表示短型等。标识符的本体采用首字母大写法。如：strUserName、pstrUserName等。
- **骆驼命名法(CamelCase)**。正如它的名称所表示的那样，是指混合使用大小写字母来构成标识符的名称，利于快速理解标识符，首字母小写，其余每个单词首字母大写，如UserName、getUserName等。因为看上去像驼峰而得名。
- **帕斯卡命名法(PascalCase)**。即Pascal命名法，因为源自Pascal语言而流行，与骆驼命名法相似。其规则是所有单词首字母均大写，包括首字母，如UserName等。

第十一节 扩展库

Python扩展库非常丰富，可以说已经成为Python重要特色，也是Python得以快速流行的主要原因。扩展库又称扩展模块、扩展包等，其目的是扩展某个方面的功能，如数学模块(math)、随机模块(random)、网络模块(很多)等等。Python扩展库可以分为标准库和第三方库，标准库在安装Python自行随之安装，第三方库则需要单独安装。安装成功后与标准库用法相同。在代码中使用扩展库，需要使用关键字import，如例程1-63第4行所示。装入math库后，可以使用其中函数与属性，如pi是math的属性，floor()是math的函数等。在实际代码编写过程汇总，如果涉及到数学计算，可以先参看math是否已经提供。如已提供则直接使用，否则可通过自定义函数扩展。在交互式环境，可以先执行import math命令后执行help(math)查看相关帮助，其他库与之类似。

例程1-63

第1行	#扩展库的使用
第2行	
第3行	#装入math扩展库
第4行	import math
第5行	
第6行	print(math.pi)#输出：3.141592653589793，pi是math对象的属性
第7行	
第8行	valA=math.floor(-11.98)#math.floor()返回小于等于参数的最大整数
第9行	print(valA)#输出：-12
第10行	
第11行	valB=math.ceil(-11.98)#math.ceil()返回大于等于参数的最小整数
第12行	print(valB)#输出：-11
第13行	
第14行	valB=math.factorial(5)#求5的阶乘
第15行	print(valB)#输出：120
第16行	
第17行	#math.radians()是将角度转换为弧长，参数为角度
第18行	#math.cos()是求余弦，其参数会为弧长
第19行	print(math.cos(math.radians(60)))#0.5000000000000001
第20行	

```

第21行 print(math.sqrt(2))#求根号2的值，此处输出：1.4142135623730951
第22行
第23行 #eof

```

random(随机库)和time(时间库)也是很常用的库，例程1-64是random库的应用示例。random.random()将产生0-1之间的随机小数；random.uniform(m,n)则产生m和n之间的随机小数，m和n可以是整数、小数、正数或负数，m大于n或m小于n均可；random.randint(m,n)用于产生m和n之间的随机整数，要求m和n都是整数(可以是负整数)，且m小于n；random.randrange(startNum,endNum,stepNum)要求startNum、endNum和stepNum都是整数，与range()函数很相似；如仅有一个参数，默认startNum为0，参数值为endNum，如有两个参数，则第1个参数是startNum，第2个参数是endNum，stepNum默认为1；random.choice(Iterable)用于从Iterable中随机选择1个，Iterable是能for-in循环的类型，如：tuple、list、str等；random.sample(Iterable,n)表示从Iterable中随机选择n个成员，相当于抽样，同样Iterable代表能for-in循环的数据；random.shuffle(listData)用于对listData打乱顺序，listData代表list类型的数据，如例程1-64第19、20行所示。

例程1-64

```

第1行 #扩展库的使用
第2行
第3行 #装入random扩展库
第4行 import random
第5行
第6行 valA=random.random()#random.random()的功能是产生0-1之间的随机数
第7行 print("random.random()的使用：",valA)#输出：每次输出不一样
第8行
第9行 print("random.uniform(10,20)的使用：",random.uniform(10,20))#产生10-20见的随机小数
第10行 print("random.randint(10,20)的使用：",random.randint(10,20))#产生10-20见的随机整数
第11行
第12行 #从10开始，每次增加2，一直到50为止，然后从中随机选取一个
第13行 print("random.randrange(10,50,2)的使用：",random.randrange(10,50,2))
第14行
第15行 print("random.choice([1,3,4,1,7,9,11])的使用：",random.choice([1,3,4,1,7,9,11]))#随机选取其中一个
第16行
第17行 listA=[1,3,4,1,7,9,11]
第18行 print("listA原始序列",listA)
第19行 random.shuffle(listA)#乱序
第20行 print("random.shuffle()后listA的序列：",listA)#输出乱序后结果
第21行
第22行 print("random.sample()的使用：",random.sample([1,3,4,1,7,9,11],3))#随机抽样3个
第23行
第24行 #eof

```

random用途很多。例程1-65是假设仅有1张讲座票，但有很多学生学号，如何随机抽取学号。例程第9行利用random.choice()从studID中选取其中一个。如果是抽取多个，则可用random.sample()函数。

例程1-65

```

第1行 #抽讲座票1张
第2行
第3行 import random
第4行
第5行 #学号列表，可以很多，可通过文件读取形成
第6行 studID=("1800017856","1700098602","1700011744","1600012154",
第7行 "1900018407","1700098607","1800011002","1800011003")

```

```

第8行
第9行 hitID=random.choice(studID) #命中的学号
第10行 print(hitID) #输出结果，每次结果可能不一样

```

例程1-66是利用random库生成100万个随机整数(第6、7行)并排序，如果用input()函数键盘输入，将难以想象。第8行输出未排序的前50个数据，第10行利用list类对象xData的sort()函数排序然后在第11行输出排序后的前50个数据。

例程1-66

```

第1行 #随机往list中追加100万个随机整数，并排序
第2行
第3行 import random
第4行
第5行 xData=[] #空的list
第6行 for i in range(0,1000000):
第7行     xData.append(random.randint(0,1000000)) #追加随机整数
第8行 print(xData[:50]) #输出前50个数据
第9行
第10行 xData.sort() #排序
第11行 print(xData[:50]) #输出前50个数据

```

[1989715, 5874252, 5035039, 2777955, 81462, 4664991, 1453707, 3962577, 5532799, 7372510, 270731, 242025, 448379, 5810973, 7620466, 6793274, 7895691, 2494676, 6051943, 5357200, 9304021, 4609456, 1085417, 341266, 2220036, 8358997, 7245219, 3936478, 4899943, 9240902, 788734, 3129553, 6139813, 90397, 4961351, 5867636, 7389430, 993141, 5736138, 5205049, 4295422, 436908, 7686481, 2863019, 1241962, 7830900, 401481, 5947445, 776668, 9678826]

[14, 25, 32, 42, 47, 47, 62, 68, 80, 83, 90, 100, 142, 145, 170, 171, 195, 195, 205, 220, 226, 229, 234, 245, 248, 252, 260, 271, 280, 297, 306, 314, 325, 336, 362, 375, 377, 380, 381, 381, 387, 388, 415, 433, 468, 475, 488, 500, 504, 514]

图1-35 例程1-66执行效果

例程1-67是与时间相关的time库示例，其中time.time()函数(例程第6、7行)可以获得函数执行时刻的时间，注意是偏离1970年1月1日0时0分0秒后的秒数，相当于时间戳用以唯一标记某一时刻的时间；time.localtime(timestamp)可以获得时间戳的元组数据(如省略则默认当前时间)，其中0对应tm_year是年份，如例程第9、10、18行所示，其他则是1对应tm_mon(月份)、2对应tm_mday(天)、3对应tm_hour(小时)、4对应tm_min(分钟)、5对应tm_sec(秒数)、6对应tm_wday(周天，一周中的某一天，0代表周一)、7对应tm_yday(一年第N天，1到366之间)、8对应tm_isdst(是否夏令时，1表示夏令时、0表示非夏令时，-1表示状态未知，默认-1)；time.strftime(fmt,tupleTime)，fmt表示时间格式化数据，tupleTime是时间元素数据如time.localtime()函数的返回值，fmt如例程第14行所示以%加一个字符构成占位符，其中%Y表示4位数年份、%y表示2位数年份、%m表示月份、%d一月中的某一天、%H表示24小时制小时数、%I表示12小时制小时数、%M表示分钟数、%S表示秒、%a表示星期简化名称(如Mon、Fri)、%A本地完整星期名称(如Monday、Friday)、%b表示月份简化名称(如：Feb、Dec)、%B表示月份完整名称(如：February、December)、%c表示日期和时间表示、%j表示一年内的某一天、%p表示A.M.或P.M.、%U表示一年中的某星期数(注：星期天为开始)、%w表示星期几(注：星期天为星期的开始)、%W一年中的星期数(00-53)(注：星期一为开始)、%x表示相应日期、%X表示相应时间；time.strptime(strTime,fmt)函数用于将字符串strTime按ftm字符串指定格式转换为time数据，fmt格式定义与time.strftime(fmt,tupleTime)相同。

时长计算是较为常用的需求。例程第20到26行是涉及到计时相关的函数。time.time()能获得函数两次执行的时间戳，相减可获得时间，如例程第20、24行所示；time.perf_counter()和time.process_time()能获得更高精度，前者包含time.sleep(N)时间(例程第23行，相当于暂停N秒)，后者不包括。另外，time.process_time()是所在程序CPU执行时间，而time.perf_counter()是系统运行时间。由于基准点没有定义，因此time.perf_counter()或time.process_time()只有两次之差才有实际意义。

例程1-67

```

第1行 #time库示例
第2行
第3行 import time
第4行

```

```

第5行 #获取计算机时间，返回值为浮点数，是函数执行时距1970-1-1 00:00:00的秒数
第6行 nbTime=time.time()
第7行 print(nbTime) #输出:1581654196.1632285
第8行
第9行 localtime=time.localtime() #获得当地当时时间数据，包括年月日、时分秒、周几年日以及是否夏令时
第10行 print(localTime.tm_year,localTime.tm_mon,localTime.tm_mday) #输出： 2020 2 14
第11行
第12行 localtime2=time.localtime(nbTime) #time.localtime()可以有参数，无参则获得当前时间
第13行
第14行 print(time.strftime("%Y年%m月%d日 %H时%M分%S秒", time.localtime()))
第15行 #输出： 2020年02月14日 12时32分02秒
第16行
第17行 myTime=time.strptime("20190211123456", "%Y%m%d%H%M%S")
第18行 print(f"{myTime[0]}年{myTime[1]}月{myTime[2]}日")
第19行
第20行 print("开始计时1: ",time.time())
第21行 print("开始计时2: ",time.perf_counter())
第22行 print("开始计时3: ",time.process_time())
第23行 time.sleep(10)
第24行 print("结束计时1: ",time.time())
第25行 print("结束计时2: ",time.perf_counter())
第26行 print("结束计时3: ",time.process_time())

```

```

1581656919.3382452
2020 2 14
2020年02月14日 13时08分39秒
2019年2月11日
开始计时1: 1581656919.3920965
开始计时2: 0.392647866
开始计时3: 0.15625
结束计时1: 1581656929.4294424
结束计时2: 10.467900024
结束计时3: 0.171875

```

图1-36 例程1-67执行效果

例程1-68中的timeAFinish-timeAStart是计算追加1000万条数据到list类对象xData所需耗时，而timeBFinish-timeBStart是对1000万条数据排序所需耗时。从中也可以看出，如此规模数据，计算也非常快。

例程1-68

```

第1行 #time和random库联合应用示例
第2行
第3行 import random
第4行 import time
第5行
第6行 xData=[] #空的list
第7行
第8行 #数据追加时间
第9行 timeAStart=time.perf_counter()
第10行
第11行 for i in range(0,10000000):

```



```

第12行    xData.append(random.randint(0,10000000)) #追加随机整数
第13行
第14行    timeAFinish=time.perf_counter()
第15行    print("数据追加耗时: ",timeAFinish-timeAStart)
第16行
第17行    print(xData[:50]) #输出前50个数据
第18行
第19行    timeBStart=time.perf_counter()
第20行
第21行    xData.sort() #排序
第22行
第23行    timeBFinish=time.perf_counter()
第24行    print("数据排序耗时: ",timeBFinish-timeBStart)
第25行    print(xData[:50]) #输出前50个数据

```

数据追加耗时: 10.912099492

[358782, 6334782, 2271717, 3580161, 4151180, 5640277, 1856216, 2026315, 7162267, 8282479, 2670897, 8657263, 9479618, 5774265, 9669562, 1339585, 8192576, 7377013, 2448495, 8506420, 4691199, 2972368, 8946095, 9774789, 5741877, 5038441, 9162590, 7751147, 5640500, 1621594, 3146701, 2281172, 7300876, 3673142, 5401729, 8198340, 7062846, 9791109, 2222552, 6498833, 9726603, 9854899, 6674945, 7322723, 5937663, 8323580, 3834395, 8433762, 6509245, 9686112]

数据排序耗时: 4.0777833690000005

[0, 0, 0, 2, 2, 4, 6, 6, 7, 9, 9, 10, 14, 14, 15, 16, 18, 19, 20, 20, 21, 22, 22, 23, 23, 24, 24, 24, 24, 26, 26, 28, 28, 28, 31, 35, 37, 37, 40, 41, 41, 42, 42, 42, 43, 45, 47, 47, 48, 49]

图1-37 例程1-68执行效果

除了标准库外，Python还有大量需要安装才能使用的第三方库，安装后和标准库使用方法相同，将在后续章节涉猎。

第十二节 与Excel

Excel是常用的数据处理统计分析可视化效率工具(或者说是常用办公软件)。Python与Excel有多种交互方式，文件是其中最常用方式，二者结合，能发挥Python数据处理优势，同时也能发挥Excel简洁方便综合数据处理能力，例程1-69所处理的数据如图1-38所示（注：本例所涉数据来自国家统计局网站，数据量不大用Excel直接也很方便，此处更多是说明其一般处理流程），其产生的文件可在Excel中打开。

23523,23348,23008,22715,22558,22329,22287,22164,22259,24659
99357,99829,100260,100361,100469,100582,100403,100283,99938
16658,15831,15003,14386,13755,13161,12714,12288,11894,11307

图1-38 2018-1999年人口数据局部

例程1-69

```

第1行    #统计1999-2018年的人口数据，数据之间用逗号(,)分隔
第2行    #第1行：2018到2019年0-14岁人口数量，单位为万
第3行    #第2行：2018到2019年15-64岁人口数量，单位为万
第4行    #第3行：2018到2019年65岁以上人口数量，单位为万
第5行
第6行    populationFile=open("1999_2018.txt","r",encoding="utf-8")
第7行    populationData=populationFile.read()
第8行    populationFile.close()
第9行

```



```

第10行 populationData=populationData.splitlines()
第11行
第12行 pop0to14=populationData[0].split(",")
第13行 pop15to64=populationData[1].split(",")
第14行 pop65up=populationData[2].split(",")
第15行
第16行 wFile=open("population.txt","w",encoding="utf-8")
第17行 print("year,pop0to14,pop15to64,pop65up,rate",file=wFile) #写入标题行
第18行 for i in range(0,20):
第19行     #0-14占比
第20行     rate=int(pop0to14[i])/(int(pop0to14[i])+int(pop15to64[i])+int(pop65up[i]))
第21行     print(2018-i,int(pop0to14[i]),int(pop15to64[i]),int(pop65up[i]),rate,sep="," ,file=wFile)
第22行
第23行 wFile.close()

```

在例程1-69中，第6-8行分别是打开文件，读取数据并按行分解成list，然后关闭文件。第12到14行，分别将每行数据分别赋值给三个变量对应是0-14岁人口量（pop0to14）、15-64岁（pop15to64）、65岁及以上（pop65up），并按文件给定的逗号进行数据项切分。第16行是打开文件以备数据写入，第17行写入数据的标题。数据总计20项，因此循环范围是range(0,20)，第20行计算出0-14岁在总人口中的占比，第21行则将数据写入文件，注意理解2018-i的变化过程。

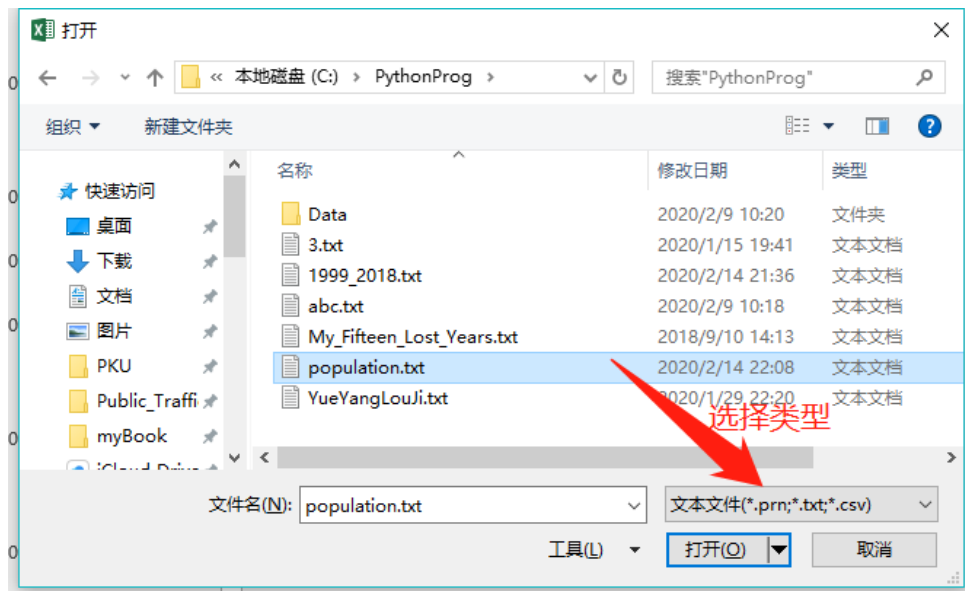


图1-39 在Excel中打开生成的文件

在Excel中打开由程序生成的文件。由于Python在此处生成文件类型是txt，而Excel默认是xls或xlsx，因此需要如图中所示，选择打开文件的类型。打开文件后，将进入如图1-40所示界面。如果数据有标题，请勾选，并根据文件的编码格式选择合适的类型。

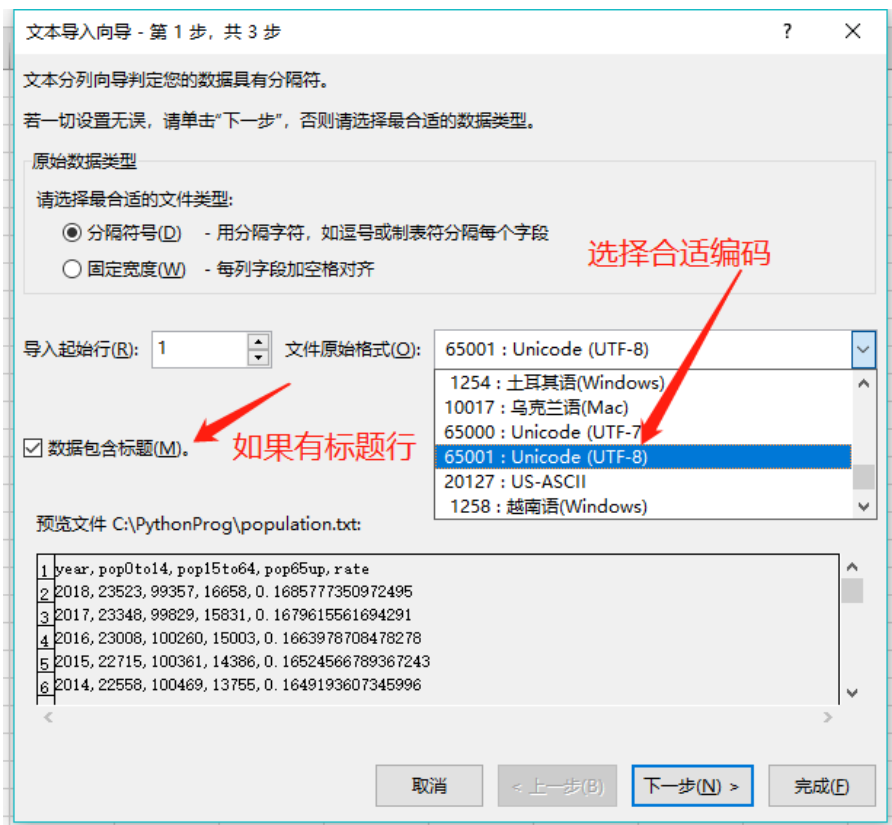


图1-40 选择文件格式及确定是否有标题行等

单击“下一步”后的界面是选择数据之间合适的分隔符, 如图1-41所示。当分隔符选择正确后, 将会发现数据已经按列分隔, 如图所示。

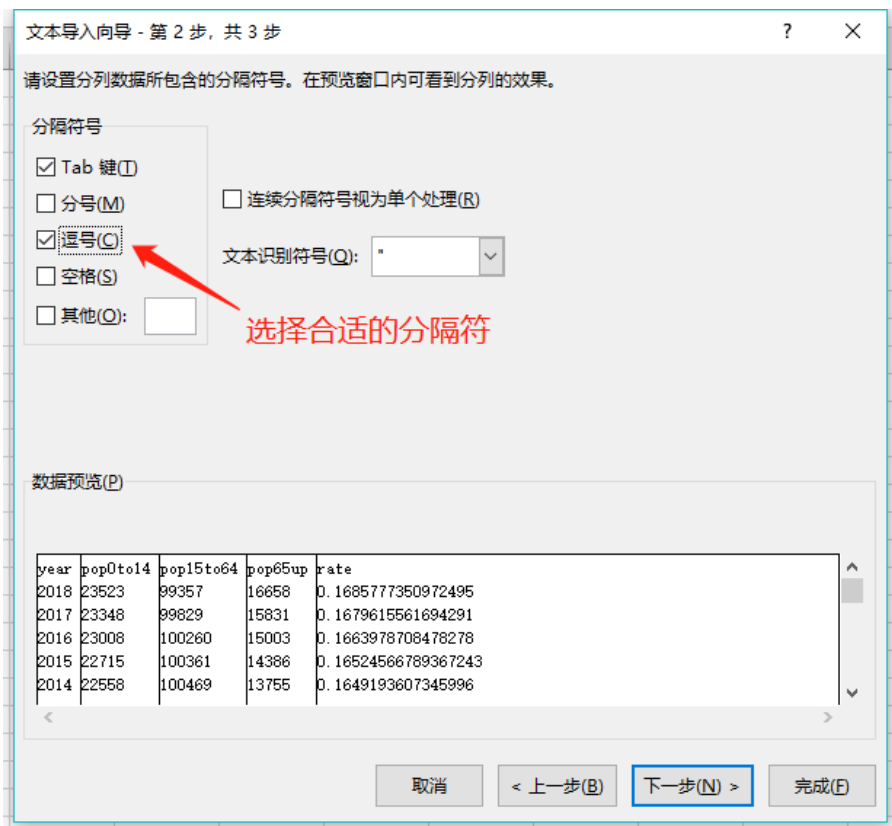


图1-41 设置数据列分隔符

单击“下一步”将进入设置各列数据格式, 也可以筛选某些列。操作时选择先选择某列, 然后设置对应列的格式。如果某列不设置, 则会默认是常规列, 将根据该列数据组成自动转换为相应类型。

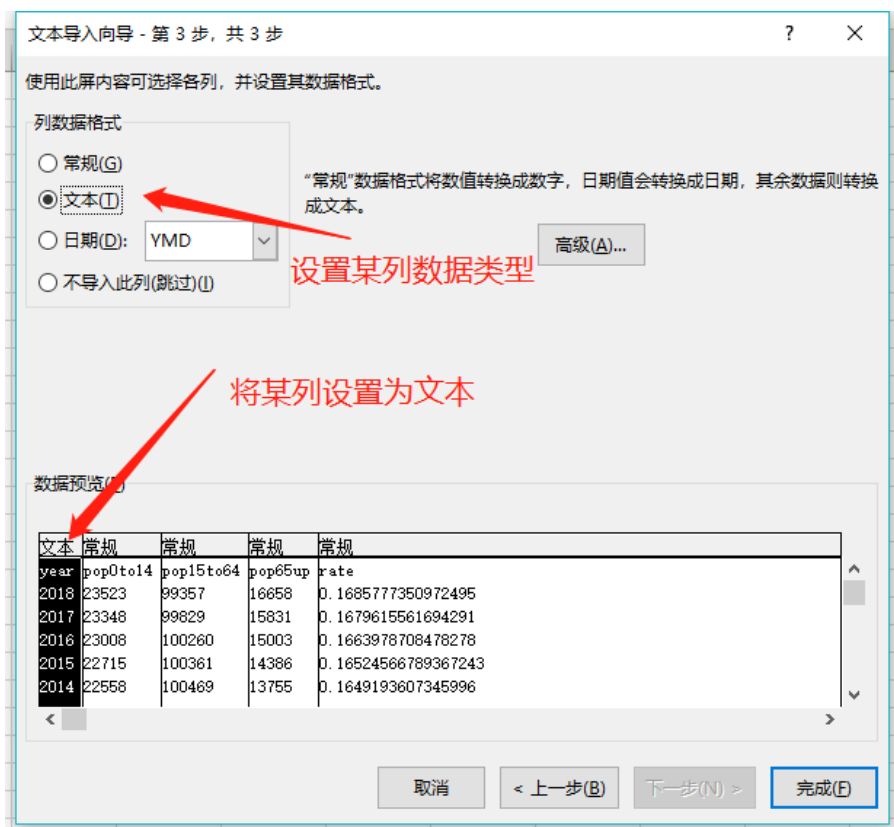


图1-42 设置数据列数据类型

单击“完成”后将出现如图1-43所示界面，数据已经成功导入到Excel之中。在Excel中，可以对数据进一步处理，包括数据排序、进一步计算以及生成统计图等。

文件 开始 插入 页面布局 公式 数据 审阅

剪贴板 复制 格式刷 剪贴板

等线 11 B I U 字体

B10 22259

	A	B	C	D	E
1	year	pop0to14	pop15to64	pop65up	rate
2	2018	23523	99357	16658	0.168577735
3	2017	23348	99829	15831	0.167961556
4	2016	23008	100260	15003	0.166397871
5	2015	22715	100361	14386	0.165245668
6	2014	22558	100469	13755	0.164919361
7	2013	22329	100582	13161	0.164096949
8	2012	22287	100403	12714	0.164596319
9	2011	22164	100283	12288	0.164500687
10	2010	22259	99938	11894	0.165999209
11	2009	24659	97484	11307	0.184780817
12	2008	25166	96680	10956	0.189500158
13	2007	25660	95833	10636	0.194204149
14	2006	25961	95068	10419	0.197500152
15	2005	26504	94197	10055	0.202698155
16	2004	27947	92184	9857	0.214996769
17	2003	28559	90976	9692	0.220998708
18	2002	28774	90302	9377	0.22400411
19	2001	28716	89849	9062	0.224999412
20	2000	29012	88910	8821	0.22890416
21	1999	31950	85157	8679	0.25400283

图1-43 数据导入到Excel之中

图1-44是按年份排序后，选择年份和0-14岁占比所做的折线图，从图中可以看出0-14岁占比越来越低，即便“二胎”政策后，变化也仍然不大，老龄化已成现实。

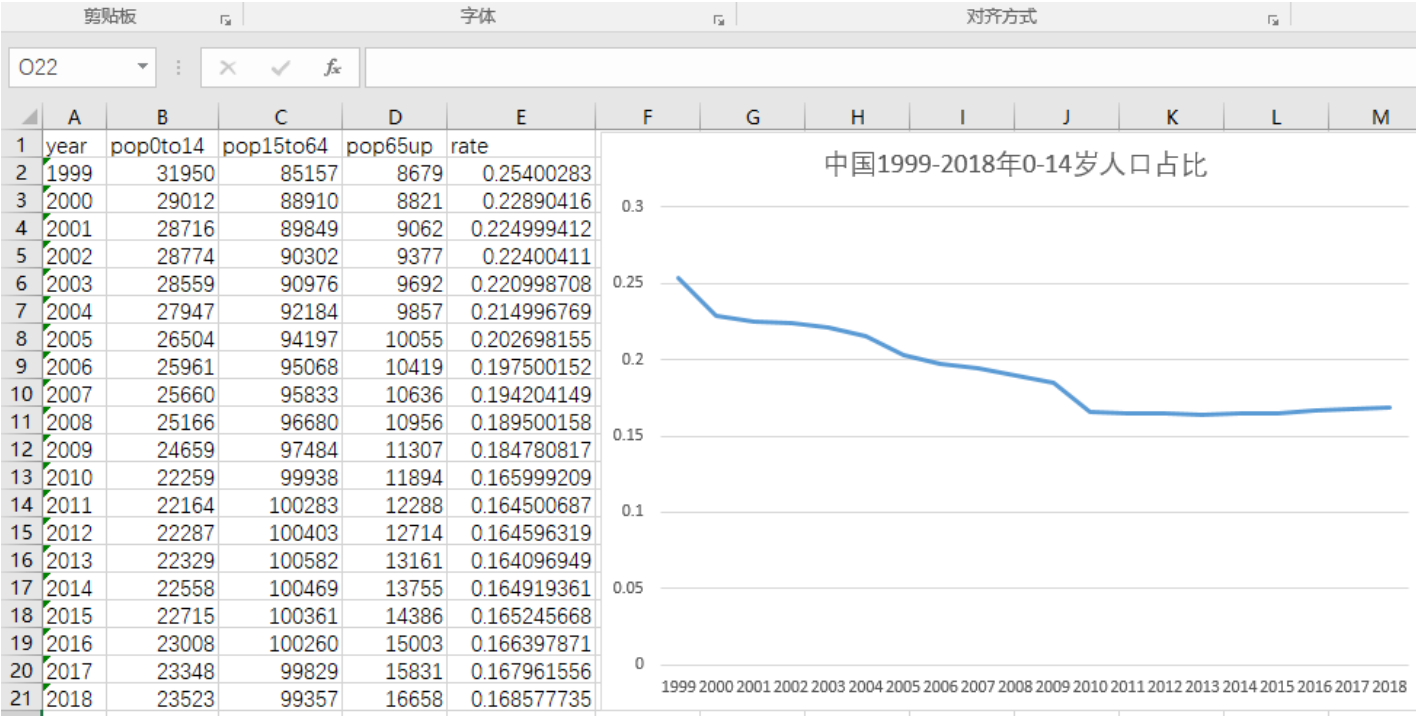


图1-44 对数据按列Year排序并选择年份和占比做折线图

在实际学习和工作中，利用Python和Excel，已能解决很多实际问题，是学习门槛不高易用实用的珠联璧合。