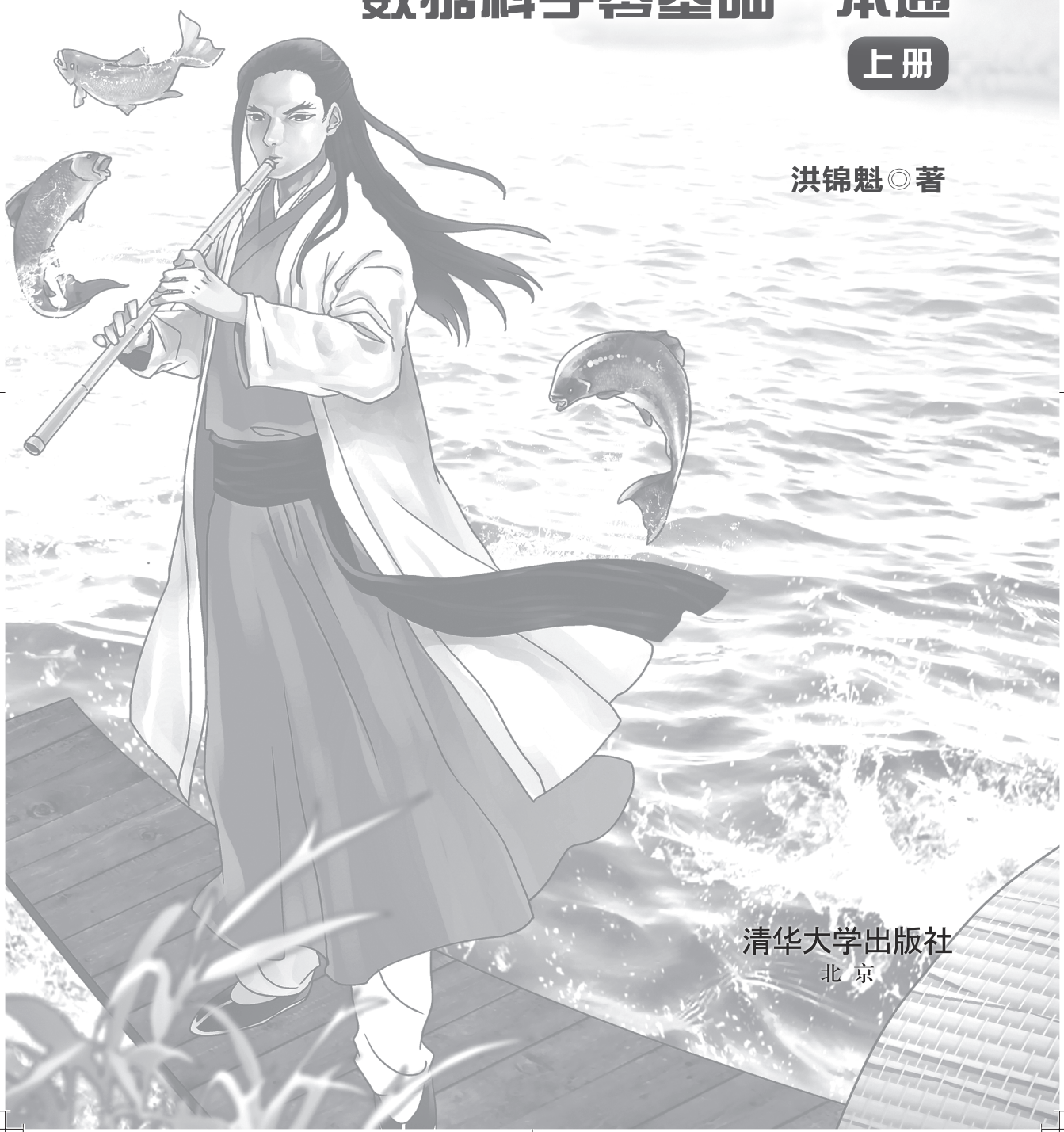
Python

数据科学零基础一本通

上册



洪锦魁◎著

清华大学出版社
北 京

内 容 简 介

这是一本专为没有编程基础的读者编写的 Python 入门书籍，全书包含 800 多个程序实例及 200 多道实践习题，一步一步详细讲解 Python 语法的基础知识，同时也将应用范围拓展至图形界面设计、影像处理、图表绘制、文字识别、词云、股市资料摘取与图表制作、线性代数、基础统计以及与数据科学相关的 Numpy、Scipy、Pandas。Python 是一门非常灵活的编程语言，本书特色在于对 Python 的基础知识与应用辅以大量实例进行讲解，读者可以通过这些程序实例事半功倍地学会 Python。

图书在版编目（CIP）数据

Python数据科学零基础一本通 / 洪锦魁著. —北京：清华大学出版社，2020.2
ISBN 978-7-302-54539-2

Ⅰ. ①P… Ⅱ. ①洪… Ⅲ. ①软件工具—程序设计 Ⅳ. ①TP311.561

中国版本图书馆 CIP 数据核字（2019）第 290377 号

责任编辑： 杜 杨 薛 阳
封面设计： 杨玉兰
责任校对： 徐俊伟
责任印制：

出版发行： 清华大学出版社
网 址： http://www.tup.com.cn，http://www.wqbook.com
地 址： 北京清华大学学研大厦 A 座 **邮 编：** 100084
社 总 机： 010-62770175 **邮 购：** 010-62786544
投稿与读者服务： 010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：
经 销： 全国新华书店
开 本： 170mm×240mm **印 张：** 48.75 **字 数：** 1278 千字
版 次： 2020 年 4 月第 1 版 **印 次：** 2020 年 4 月第 1 次印刷
定 价： 129.00 元（上、下册）

产品编号：085277-01

序

多次与教育界的朋友相聚，谈到计算机语言的发展趋势时，大家一致认为 Python 是当今最重要的计算机语言。许多知名公司，例如 Google、Facebook 等皆已将 Python 列为必备计算机语言。许多人想学 Python，市面上的书也不少，但书中对 Python 语法的讲解并不完整，造成读者学习上的障碍，读者读完一本 Python 书籍，仍然看不懂专家写的 Python 程序。因此，笔者决定撰写一本用丰富、实用、有趣的实例完整且深入讲解 Python 语法的入门书籍。

Python 以简洁著名，语法非常灵活，同时拥有丰富、实用的模块。本书除了以实例解说 Python 语法，还会穿插讲解各种模块，以帮助读者更灵活地掌握 Python。此外，笔者也尝试在书中穿插基本的科学、数学、统计与人工智能的基础知识，帮助读者为进一步的学习打下扎实的基础。

本书包含 800 多个程序实例，搭配 400 多个模块，并辅以 200 多道实践习题，细致讲解 Python 语法。本书也会说明下列知识与应用：

- 人工智能基础知识；
- Python 彩蛋；
- 从 bytes 数据、编码（encode）、译码（decode）说起，到精通列表（list）、元组（tuple）、字典（dict）、集合（set）；
- 从小型列表、元组、字典到大型数据资料的建立；
- 生成式（generator）建立 Python 数据结构；
- 在坐标轴内计算任意两点之间的距离，同时解说与人工智能的关联；
- 用经纬度计算地球任意两座城市之间的距离，学习取得地球任意位置的经纬度；
- 用莱布尼茨公式、尼拉卡莎级数、蒙特卡罗模拟计算圆周率；
- 讲解基础函数观念，也深入到嵌套、closure、lambda、Decorator 等高阶应用；
- 对 map( ) 和 reduce( ) 进行完整解说，并进一步配合 lambda 解说高级应用；
- 建立类别的同时深入讲解装饰器 @property、@classmethod、@staticmethod 与类别特殊属性与方法；

- 设计与应用自己设计的模块、活用外部模块（module）；
- 赌场骗局；
- 自己设计加密与解密程序；
- Python 的输入与输出；
- 文件压缩与解压缩；
- 程序除错与异常处理；
- 文件读取与目录管理；
- 剪贴板应用；
- 正则表达式；
- 递归式观念与碎形 Fractal；
- 图像处理与文字辨识，更进一步说明计算机储存图像的方法；
- 基本与进阶 QR code 制作；
- 词云（Word Cloud）设计；
- GUI 设计：设计小计算器；
- 动画与游戏；
- matplotlib 中英文图表绘制；
- 说明 CSV 和 JSON 文件；
- 股市数据读取与图表制作；
- Python 解线性代数；
- Python 解联立方程式；
- Python 执行数据分析；
- 科学计算与数据分析 Numpy、Scipy、Pandas。

笔者编写过许多计算机领域的著作，本书将沿袭笔者以往著作的特色，程序实例丰富。相信读者通过学习本书内容，一定可以快速精通 Python。笔者虽力求完美，但是书中不足与疏漏在所难免，请不吝指正。

洪锦魁

2019.10.31

目　录

01

第 1 章

基本概念

本章摘要

1-1 认识 Python

Python 是一种**直译式**（Interpreted Language）、**面向对象**（Object Oriented Language）的程序语言，它拥有完整的函数库，可以协助用户轻松地完成许多常见的工作。

直译式语言是指，**直译器**（Interpretor）会将程序代码一句一句直接执行，不需要经过**编译**（Compile）动作，将语言先转换成**机器码**，再予以执行。目前 Python 的直译器是 CPython，这是由 C 语言编写的一个直译程序，与 Python 一样目前由 Python 基金会管理使用。

Python 也算是一种动态的高级语言，具有**垃圾回收**（garbage collection）功能。**垃圾回收**是指程序在执行时，直译程序会主动收回不再需要的动态内存空间，将内存集中管理，这种机制可以减轻程序设计师的负担，当然也就减少了程序设计师犯错的机会。

由于 Python **开放源码**（Open Source），每个人皆可免费使用或为它贡献，除了它本身有许多内建的**套件**（package）或称**模块**（module）外，许多公司也为它开发了更多的**套件**，促使它的功能可以持续扩充，因此 Python 目前已经是全球最热门的程序语言之一。

1-2 Python 的起源

Python 的最初设计者是**吉多・范罗姆苏**（Guido van Rossum），他是荷兰人，1956 年出生于荷兰哈勒姆，1982 年毕业于阿姆斯特丹大学的数学和计算机系，并获得硕士学位。

吉多・范罗姆苏在 1996 年为一本 O'Reilly 出版社作者 Mark Lutz 所著的 *Programming Python* 的序言中表示：**6 年前，1989 年我想在圣诞节期间思考设计一种程序语言打发时间，当时我正在构思一个新的脚本（script）语言的解释器，它是 ABC 语言的后代，期待这个程序语言对 UNIX C 的程序语言设计师会有吸引力。基于我是蒙提派森飞行马戏团（Monty Python's Flying Circus）的疯**

狂爱好者，所以就以 Python 为这个程序命名。

在一些 Python 的文件或有些书封面喜欢用蟒蛇代表 Python，但是从吉多 • 范罗姆苏的上述序言可知，Python 灵感的来源是**马戏团名称**而非**蟒蛇**。

1999 年，他向美国国防高级研究计划局（Defense Advanced Research Projects Agency，DARPA）提出 Computer Programming for Everybody 的研发经费申请，并提出了下列 Python 的目标。

（1）这是一个简单直觉式的程序语言，可以和主要程序语言一样强大。

（2）这是开放源码（Open Source）的程序语言，每个人皆可自由使用与贡献。

（3）程序代码像英语一样容易理解与使用。

（4）可在短期间内开发一些常用功能。

现在上述目标都已经实现了，Python 已经与 C/C++、Java 一样成为程序设计师必备的程序语言，然而它却比 C/C++ 和 Java 更容易学习。

目前，Python 语言是由 Python 软件基金会（www.python.org）管理，有关新版软件下载的相关信息可以在这个基金会取得，可参考附录 A。

1-3 Python 语言发展史

在 1991 年 Python 正式诞生时，当时的操作系统平台是 Mac。尽管吉多 • 范罗姆苏坦言 Python 是构思于 ABC 语言，但是 ABC 语言并没有成功。吉多 • 范罗姆苏本人认为 ABC 语言并不是一个开放的程序语言，是其失败的主要原因。因此，在 Python 的推广中，他避开了这个错误，将 Python 推向开放式系统，因而获得了巨大的成功。

1. Python 2.0 发布

2000 年 10 月 16 日，Python 2.0 正式发布，主要是增加了**垃圾回收**的功能，同时支持 Unicode。

Unicode 是一种适合多语系的编码规则，主要是使用可变长度字节方式存储字符，以节省内存空间。例如，对于英文字母而言是使用 1 字节（byte）空间存储即可，对于含有附加符号的希腊文、拉丁文或阿拉伯文等则用 2 字节空间存储，中文则是以 3 字节空间存储，只有极少数的平面辅助文字需要 4 字节空间存储。也就是说，这种编码规则已经包含全球所有语言的字符了，所以采用这种编码方式设计程序时，其他语系的程序只要支持 Unicode 编码即可显示。例如，法国人即使使用法文版的程序，也可以正常显示中文。

2. Python 3.0 发布

2008 年 12 月 3 日，Python 3.0 正式发布。一般程序语言的发展会考虑到兼容特性，但是 Python 3 在开发时为了不受先前 2.x 版本的束缚，因此没有考虑兼容特性，所以许多早期版本开发的程序是无法在 Python 3.x 版上执行的。

不过为了解决这个问题，尽管发布了 Python 3.x 版本，后来又陆续将 3.x 版的特性移植到 Python 2.6/2.7x 版上，所以现在进入 Python 基金会网站时，可以发现有 2.7x 版和 3.7x 版的软件可以下载。

笔者经验提醒：有一些早期开发的冒险游戏软件只支持 Python 2.7x 版，目前尚未支持 Python 3.7x 版。不过相信这些软件未来也将朝向支持 Python 3.7x 版的路迈进。

Python 基金会提醒：Python 2.7x 已经被确定为最后一个 Python 2.x 的版本，目前暂定基金会对此版本的支持到 2020 年。

笔者在撰写此书时，所有程序都是以 Python 3.x 版作为主要依据的。

1-4 Python 的应用范围

尽管 Python 是一个非常适合初学者学习的程序语言，在国外有许多儿童程序语言教学也是以 Python 为工具，然而它却是一个功能强大的程序语言，下列是它的部分应用。

（1）设计动画游戏。

（2）支持图形用户接口（Graphical User Interface, GUI）开发。

（3）数据库开发与设计动态网页。

（4）科学计算与大数据分析。

（5）人工智能与机器学习。

（6）Google、Yahoo!、YouTube、NASA、Dropbox（文件分享服务）、Reddit（社交网站）在内部都大量使用 Python 作为开发工具。

（7）网络爬虫、黑客攻防。

目前，Google 搜索引擎、纽约股票交易所、NASA 航天行动的关键任务执行，都是使用 Python 语言。

1-5 静态语言与动态语言

变量（variable）是一个语言的核心，由变量的设置可以知道这个程序所要完成的工作。

有些程序语言的**变量**在使用前需要先声明它的数据类型，这样**编译程序**（compile）会在内存内预留空间给这个变量。这个变量的数据类型经过声明后，未来无法再改变它的数据类型，这类的程序语言称为**静态语言**（static language），例如，C、C++、Java 等。声明变量可以协助计算机捕捉可能的错误，同时也可以让程序执行速度更快，但是程序设计师需要花更多的时间编写程序与思考程序的规划。

有些程序语言的变量在使用前不必声明它的数据类型，这样可以用比较少的程序代码完成更多工作，增加程序设计的便利性，这类程序在执行前不必经过**编译**（compile）过程，而是使用**直译器**（interpreter）直接**直译**（interpret）与**执行**（execute），这类的程序语言称为**动态语言**（dynamic language），有时也可称这类语言是**文字码语言**（scripting language），例如，Python、Perl、Ruby。动态语言执行速度比经过编译后的静态语言执行速度慢，所以有相当长的时间动态语言只适合进行短小程序的设计，或是将它作为准备数据供静态语言处理，在这种状况下也有人将这种动态语言称为**胶水码**（glue code）。后来随着软件技术的进步，直译器执行速度越来越快，已经可以用它执行复杂的工作了。如果读者懂 Java、C、C++，将会发现，Python 相较于这些语言除了便利性，程序设计效率已经远远超过这些语言了，这也是 Python 成为目前最热门程序语言的原因。

使用 Python 语言时可以直接在提示信息下（>>>）输入程序代码执行工作（可参考 1-7 节），也可以将程序代码存储成文件然后再执行（可参考 1-9 节）。

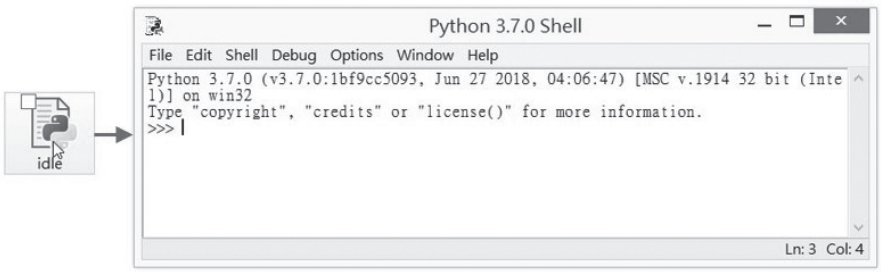
1-6 跨平台的程序语言

Python 是一种跨平台的程序语言，主要的操作系统，如 Windows、Mac OS、UNIX、Linux 等，都可以安装和使用。

跨平台的程序语言意味着，用户可以在某一个平台上使用 Python 设计一个程序，未来这个程序也可以在其他平台上顺利运行。

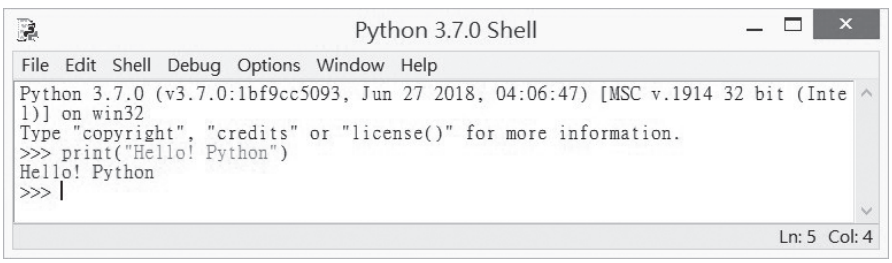
1-7 系统的安装与执行

有关安装 Python 的步骤请参考附录 A。下面将以 Python 3.7x 版为例进行说明。双击附录 A 中所建的在 Windows 桌面上的 idle 图标，将看到下列 Python Shell 窗口。



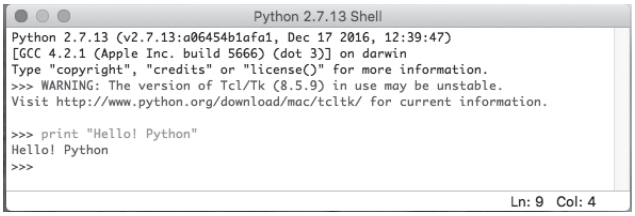
图中 >>> 符号是提示信息，可以在此输入 Python 指令。

程序实例 ch1_1.py：使用 print( ) 函数，输出字符串。

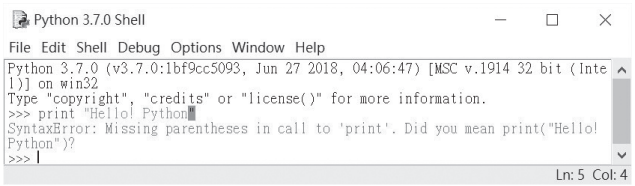


1-8 Python 2 与 Python 3 不相容的验证

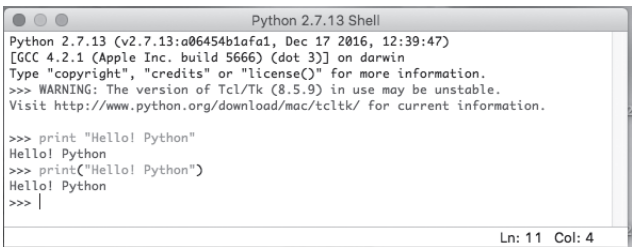
下面是早期在 Python 2 上执行输出字符串的 print 用法。



如果相同的输出方式应用在 Python 3 中将出现错误。



出现错误的原因是，在 Python 3 中 print( ) 已经是一个函数。不过在 1-3 节中也提过，Python 基金会后来陆续将 3.x 版的特性移植到 Python 2.6/2.7x 版上，所以如果在 Python 2.6/2.7x 版本上使用 print( ) 函数，将可以得到正确的输出。

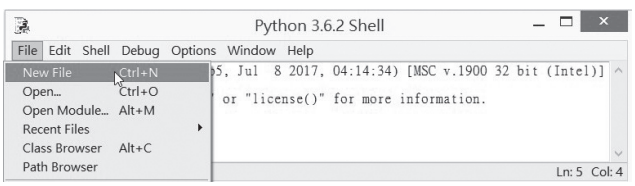


1-9 文件的建立、存储、执行与打开

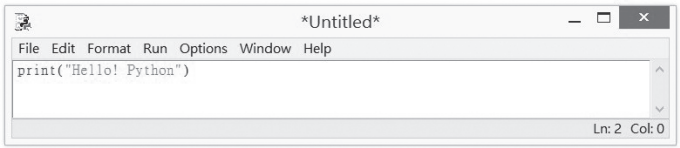
如果设计一个程序每次均要在 Python Shell 窗口环境重新输入命令的话，将是一件麻烦的事，所以程序设计时，可以将所设计的程序保存在文件内是一件重要的事。

1-9-1 文件的建立

在 Python Shell 窗口中可以执行 File → New File 命令，建立一个空白的 Python 文件。



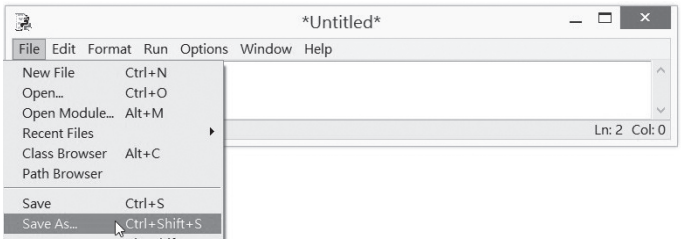
然后可以建立一个 Untitled 窗口，窗口内容是空白，下面是笔者在空白文件内输入一条命令的实例。



如果想要执行上述文件，需要先存储上述文件。

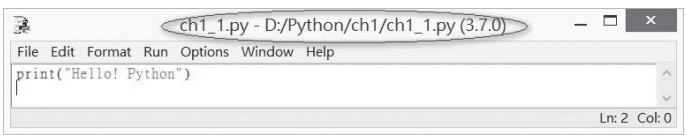
1-9-2　文件的存储

可以执行 File → Save As 命令存储文件。



然后将看到**另存新文件**对话框，此例将文件存储在 D:/Python/ch1 文件夹，文件名是 ch1_1（Python 的扩展名是 py），可以得到下列结果。

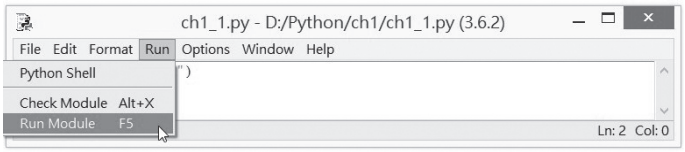
单击**“保存”**按钮。



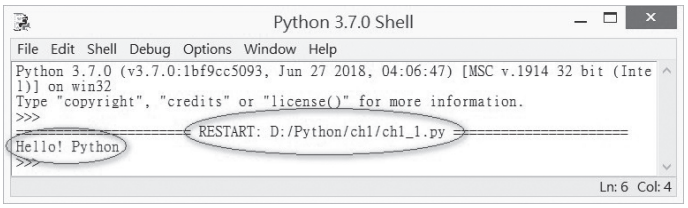
原标题 Untitled 已经改为 ch1_1.py 了。

1-9-3　文件的执行

执行 Run → Run Module 命令，就可以正式执行先前所建的 ch1_1.py 文件。



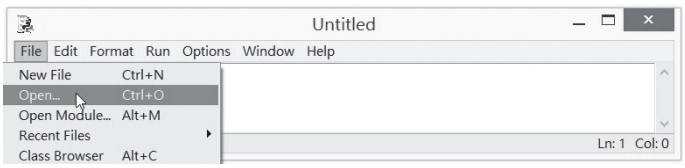
执行后，在原先的 Python Shell 窗口中可以看到执行结果。



学习到此，恭喜你已经成功地建立了一个 Python 文件，同时执行成功了。

1-9-4 打开文件

假设已经离开 ch1_1.py 文件，未来想要打开这个程序文件，可以执行 File → Open 命令。



然后会出现“打开文件”对话框，选择要打开的文件即可。

1-10 程序注释

程序注释的主要功能是让程序可读性更高，更容易了解。在企业工作中，一个实用的程序可以很轻易超过几千或上万行，此时可能需设计好几个月，给程序加上程序注释，可方便自己或他人了解程序内容。

1-10-1 注释符号

不论是使用 Python Shell 直译器或是 Python 程序文件中，“#”符号右边的文字，都称为程序注释，Python 语言的直译器会忽略此符号右边的文字。可参考下列实例。

实例 1：在 Python Shell 窗口注释的应用 1，注释可以放在程序语句的右边。

```
>>> print("Python数据科学零基础一本通")   # 打印本书名称
Python数据科学零基础一本通
>>>
```

实例 2：在 Python Shell 窗口注释的应用 2，注释可以放在程序语句的最左边。

```
>>> # 打印本书名称
>>> print("Python数据科学零基础一本通")
Python数据科学零基础一本通
>>>
```

程序实例 ch1_2.py：重新设计 ch1_1.py，为程序增加注释。

```
1  # ch1_2.py
2  print("Hello! Python")   # 打印字符串
```

注：Python 程序左边是没有行号的，上述行号是笔者为了读者阅读方便加上去的。

1-10-2　三个单引号或双引号

如果要进行大段落的注释，可以用三个单引号或双引号将注释文字包起来。

程序实例 ch1_3.py：以三个单引号当作注释。

```
1  '''
2  程序实例ch1_3.py
3  作者:洪锦魁
4  使用三个单引号当作注释
5  '''
6  print("Hello! Python")   # 打印字符串
```

上述前 5 行是程序注释。

程序实例 ch1_4.py：以三个双引号当作注释。

```
1  """
2  程序实例ch1_4.py
3  作者:洪锦魁
4  使用三个双引号当作注释
5  """
6  print("Hello! Python")   # 打印字符串
```

上述前 5 行是程序注释。

1-11　Python 彩蛋

Python 核心程序开发人员在软件内部设计了两个彩蛋，一个是搞笑网站，一个是经典名句又称 Python 之禅。这是在其他软件中没有见过的，非常有趣。

1. Python 之禅

在 Python Shell 环境下输入“import this”即可看到经典名句，其实这些经典名句也代表着研读 Python 的意境。

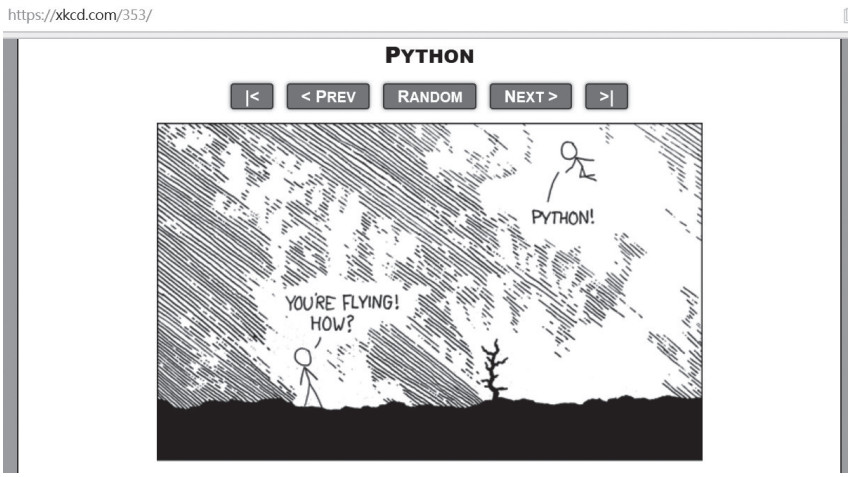
```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

2. Python 搞笑网站

可以在 Python Shell 环境下输入“import antigravity”即可连接下列网址，读者可以欣赏有关 Python 的趣味内容。

https://xkcd.com/353/



习题

设计程序可以输出下列 3 行数据。

就读学校

年级

姓名

```
========== RESTART: D:/Python/ex/ex1_1.py ==========
明志科技大学
一年级
洪锦魁
>>>
```

02

第 2 章

认识变量与基本数学运算

本章摘要

本章将从基本数学运算开始，一步一步讲解变量的使用与命名，接着介绍 Python 的算术运算。

2-1 用 Python 做计算

假设读者到麦当劳打工，一小时可以获得 120 元，如果想计算一天工作 8 小时，可以获得多少工资，可以用计算器执行 120×8，然后得到执行结果。在 Python Shell 中，可以使用下列方式计算。

```
>>> 120 * 8
960
>>>
```

如果一年实际工作天数是 300 天，可以用下列方式计算一年所得。

```
>>> 120 * 8 * 300
288000
>>>
>>>
```

如果读者一个月的花费是 9000 元，可以用下列方式计算一年可以存储多少钱。

```
>>> 9000 * 12
108000
>>> 288000 - 108000
180000
>>>
```

上述先计算一年的花费，再用一年的收入减去一年的花费，可以得到所存储的金额。本章将一步一步推导应如何以程序思想，处理一般的运算问题。

2-2 认识变量

2-2-1 基本概念

变量是一个暂时存储数据的地方，对于 2-1 节的内容而言，如果时薪从 120 元调整到 125 元，想要重新计算一年可以存储多少钱，将发现所有的计算需要重新开始。为了解决这个问题，可以考虑将时薪设为一个变量，未来如果调整薪资，直接更改变量内容即可。

在 Python 中可以用“=”设置变量的内容，在这个实例中，建立了一个变量 x，然后用下列方式设置时薪。

```
>>> x = 120
>>>
```

如果想要用 Python 列出时薪，可以使用 print( ) 函数。

```
>>> print(x)
120
>>>
```

如果时薪从 120 元调整到 125 元，那么可以用下列方式表达。

```
>>> x = 125
>>> print(x)
125
>>>
```

注　在 Python Shell 环境，也可以直接输入变量名称，即可获得执行结果。

```
>>> x = 125
>>> x
125
>>>
```

一个程序中是可以使用多个变量的，如果想计算一天工作 8 小时，一年工作 300 天，可以赚多少钱，假设用变量 y 表示一年工作所赚的钱，可以用下列方式计算。

```
>>> x = 125
>>> y = x * 8 * 300
>>> print(y)
300000
>>>
```

如果每个月花费是 9000 元，使用变量 z 表示每个月的花费，可以用下列方式计算每年的花费，使用 a 表示每年的花费。

```
>>> z = 9000
>>> a = z * 12
>>> print(a)
108000
>>>
```

如果想计算每年可以存储多少钱，使用 b 表示每年所存储的钱，可以使用下列方式计算。

```
>>> x = 125
>>> y = x * 8 * 300
>>> z = 9000
>>> a = z * 12
>>> b = y - a
>>> print(b)
192000
>>>
```

上述语句顺利地使用 Python Shell 计算了每年可以存储多少钱，可是上述使用 Python Shell 做运算潜藏的最大问题是，只要过了一段时间，我们可能忘记当初所有设置的变量是代表什么意义。因此在设计程序时，如果可以为变量取个有意义的名称，未来看到程序时，可以比较容易记得。下列是笔者重新设计的变量名称。

时薪：hourly_salary，用此变量代替 x，即每小时的薪资。

年薪：annual_salary，用此变量代替 y，即一年工作所赚的钱。

月支出：monthly_fee，用此变量代替 z，即每个月的花费。

年支出：annual_fee，用此变量代替 a，即每年的花费。

年存储：annual_savings，用此变量代替 b，即每年所存储的钱。

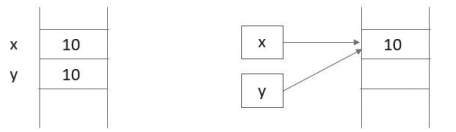
如果现在使用上述变量重新设计程序，可以得到下列结果。

```
>>> hourly_salary = 125
>>> annual_salary = hourly_salary * 8 * 300
>>> monthly_fee = 9000
>>> annual_fee = monthly_fee * 12
>>> annual_savings = annual_salary - annual_fee
>>> print(annual_savings)
192000
>>>
```

相信经过上述说明，读者应该了解变量的基本意义了。

2-2-2 认识变量的地址

Python 是一种动态语言，它处理变量的过程与一般静态语言不同。对于静态语言而言，例如 C、C++，当声明变量时内存就会预留空间存储此变量的内容，例如，若声明与定义 x=10, y=10 时，内存内容如下方左图所示。



静态语言, 例如:C　　动态语言Python相对引用观念

对于 Python 而言，变量所使用的是参照（reference）地址的观念，设置一个变量 x 等于 10 时，Python 会在内存某个地址存储 10，此时我们建立的变量 x 好像是一个标志（tags），标志内容是存储 10 的内存地址。如果有另一个变量 y 也是 10，则变量 y 的标志内容也是存储 10 的内存地址，如上方右图所示。

使用 Python 可以使用 id( ) 函数获得变量的地址，可参考下列语法。

实例：列出变量的地址，相同内容的变量会有相同的地址。

```
>>> x = 10
>>> y = 10
>>> z = 20
>>> id(x)
1614727440
>>> id(y)
1614727440
>>> id(z)
1614727600
```

2-3 认识程序的意义

延续上一节的实例，如果时薪改变、工作天数改变或每个月的花费改变，所有输入与运算都要重新开始，而且每次都要重新输入程序代码，这是一件很费劲的事，同时很可能会输入错误，为了解决这个问题，可以使用 Python Shell 打开一个文件，将上述运算存储在文件内，这个文件就是所谓的**程序**。未来有需要时，再打开重新运算即可。

程序实例 ch2_1.py：使用程序计算每年可以存储多少钱，下面是整个程序设计。

```
# ch2_1.py
hourly_salary = 125
annual_salary = hourly_salary * 8 * 300
monthly_fee = 9000
annual_fee = monthly_fee * 12
annual_savings = annual_salary - annual_fee
print(annual_savings)
```

执行结果

```
====================== RESTART: D:\Python\ch2\ch2_1.py ======================
192000
```

未来时薪改变、工作天数改变或每个月的花费改变时，只要适度修改变量内容，就可以获得正确的执行结果。

2-4 认识注释的意义

程序 ch2_1.py 中尽管已经为变量设置了有意义的名称，但时间一久，常常还是会忘记各个指令的内涵。所以笔者建议，设计程序时，应适度地为程序代码加上注释。在 1-10 节已经讲解了注释的方法，下面将直接以实例说明。

程序实例 ch2_2.py：重新设计程序 ch2_1.py，为程序代码加上注释。

```
# ch2_2.py
hourly_salary = 125                              # 设置时薪
annual_salary = hourly_salary * 8 * 300          # 计算年薪
monthly_fee = 9000                               # 设置每月花费
annual_fee = monthly_fee * 12                    # 计算每年花费
annual_savings = annual_salary - annual_fee      # 计算每年存储金额
print(annual_savings)                            # 列出每年存储金额
```

执行结果 与 ch2_1.py 相同。

相信经过上述注释后，即使再过 10 年，只要一看到程序也可轻松了解整个程序的意义。

2-5 Python 变量与其他程序语言的差异

许多程序语言变量在使用前需要先声明，Python 对于变量的使用则是可以在需要时，再直接设置使用。有些程序语言在声明变量时，需要设置变量的数据类型，Python 则不需要设置，它会针对变量值的内容自行设置数据类型。

2-6 变量的命名原则

Python 对于变量的命名，在使用时有一些规则要遵守，否则会造成程序错误。

（1）必须由英文字母、_（下画线）或中文开头，建议使用英文字母。

（2）变量名称只能由英文字母、数字、_（下画线）或中文组成。

（3）英文字母大小写是敏感的，例如，Name 与 name 被视为不同变量名称。

（4）Python **系统保留字**（或称**关键词**）不可当作变量名称，会让程序产生错误，Python 内建**函数名称**不建议当作变量名称。

注：虽然变量名称可以用中文，不过笔者不建议使用中文，将来可能会有兼容性的问题。

下列是不可当作变量名称的 Python **系统保留字**。

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

下列是不建议当作变量名称的 Python 系统**内建函数**，若是不小心将系统内建函数名称当作变量，程序本身不会错误，但是原函数功能会丧失。

abs( )	all( )	any( )	apply( )	basestring( )
bin( )	bool( )	buffer( )	bytearray( )	callable( )
chr( )	classmethod( )	cmp( )	coerce( )	compile( )
complex( )	delattr( )	dict( )	dir( )	divmod( )
enumerate( )	eval( )	execfile( )	file( )	filter( )
float( )	format( )	frozenset( )	getattr( )	globals( )
hasattr( )	hash( )	help( )	hex( )	id( )
input( )	int( )	intern( )	isinstance( )	issubclass( )
iter( )	len( )	list( )	locals( )	long( )
map( )	max( )	memoryview( )	min( )	next( )
object( )	oct( )	open( )	ord( )	pow( )
print( )	property( )	range( )	raw_input( )	reduce( )
reload( )	repr( )	reversed( )	round( )	set( )
setattr( )	slice( )	sorted( )	staticmethod( )	str( )
sum( )	super( )	tuple( )	type( )	unichr( )
unicode( )	vars( )	xrange( )	zip( )	_import( )

实例 1：下列是一些不合法的变量名称。

```
sum,1          # 变量不可有 ","
3y             # 变量不可由阿拉伯数字开头
x$2            # 变量不可有 "$" 符号
and            # 这是系统保留字不可当作变量名称
```

实例 2：下列是一些合法的变量名称。

```
SUM
_fg
x5
总和
```

实例 3：下列 3 个代表不同的变量。

```
SUM
Sum
sum
```

2-7 基本数学运算

2-7-1 四则运算

Python 的四则运算是指加（+）、减（-）、乘（*）和除（/）。

实例 1：下列是加法与减法运算实例。

```
>>> x = 5 + 6            # 将5加6设置给变量x
>>> print(x)
11
>>> y = x - 10           # 将x减10设置给变量y
>>> print(y)
1
>>>
```

实例 2：乘法与除法运算实例。

```
>>> x = 5 * 9            # 将5乘9设置给变量x
>>> print(x)
45
>>> y = 9 / 5            # 将9除以5设置给变量y
>>> print(y)
1.8
>>>
```

2-7-2 余数和整除

余数（mod）所使用的符号是“%”，可计算出除法运算中的余数。整除所使用的符号是“//”，是指除法运算中只保留整数部分。

实例：余数和整除运算实例。

```
>>> x = 9 % 5            # 将9除以5的余数设置给变量x
>>> print(x)
4
>>> y = 9 // 2           # 将9除以2的整数结果设置给变量y
>>> print(y)
4
>>>
```

其实在程序设计中求余数是非常有用的，例如，如果要判断数字是奇数或偶数可以用 %，例如

"num % 2"，如果 num 是奇数，所得结果是 1；如果 num 是偶数，所得结果是 0。当读者学会更多指令后，笔者会做更多的应用说明。

2-7-3 次方

次方的符号是“**”。

实例：平方、次方的运算实例。

```
>>> x = 3 ** 2          # 将3的平方设置给变量x
>>> print(x)
9
>>> y = 3 ** 3          # 将3的3次方设置给变量y
>>> print(y)
27
>>>
```

2-7-4 Python 语言控制运算的优先级

Python 语言碰上计算式同时出现在一个指令内时，除了括号“( )”内部运算最优先外，其余计算优先次序如下。

（1）次方；

（2）乘法、除法、求余数（%）、求整数（//），彼此依照出现顺序运算；

（3）加法、减法，彼此依照出现顺序运算。

实例：Python 语言控制运算的优先级的应用。

```
>>> x = ( 5 + 6 ) * 8 - 2
>>> print(x)
86
>>> y = 5 + 6 * 8 - 2
>>> print(y)
51
>>> z = 2 * 3**3 * 2
>>> print(z)
108
```

2-8 指派运算符

常见的指派运算符如下。

运算符	实例	说明
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b
//=	a //= b	a = a // b
**=	a **= b	a = a ** b

实例：指派运算符的实例说明。

```
>>> x = 10
>>> x += 5
>>> print(x)
15
>>> x = 10
>>> x -= 5
>>> print(x)
5
>>> x = 10
>>> x *= 5
>>> print(x)
50
>>> x = 10
>>> x /= 5
>>> print(x)
2.0
>>> x = 10
>>> x %= 5
>>> print(x)
0
>>> x = 10
>>> x //= 5
>>> print(x)
2
>>> x = 10
>>> x **= 5
>>> print(x)
100000
>>>
```

2-9 Python 等号的多重指定使用

使用 Python 时，可以一次设置多个变量等于某一数值。

实例 1：设置多个变量等于某一数值的应用。

```
>>> x = y = z = 10
>>> print(x)
10
>>> print(y)
10
>>> print(z)
10
>>>
```

Python 也允许多个变量同时指定不同的数值。

实例 2：设置多个变量，每个变量有不同值。

```
>>> x, y, z = 10, 20, 30
>>> print(x, y, z)
10 20 30
>>>
```

当执行上述多重设置变量值后，甚至可以执行更改变量内容。

实例 3：将两个变量内容交换。

```
>>> x, y = 10, 20
>>> print(x, y)
10 20
>>> x, y = y, x
>>> print(x, y)
20 10
>>>
```

上述原先 x, y 分别设为 10, 20，但是经过多重设置后变为 20, 10。其实可以使用多重指定更灵活地应用 Python，在 2-7-2 节有求商和余数的实例，可以使用 divmod( ) 函数一次获得商和余数，可参考下列实例。

```
>>> x = 9 // 5          # 将9除以5的整数给变量x
>>> print(x)
1
>>> y = 9 % 5           # 将9除以5的余数给变量y
>>> print(y)
4
>>> z = divmod(9,5)     # 一次获得商与余数
>>> print(z)
(1, 4)
>>> x,y = z
>>> print(x)
1
>>> print(y)
4
>>>
```

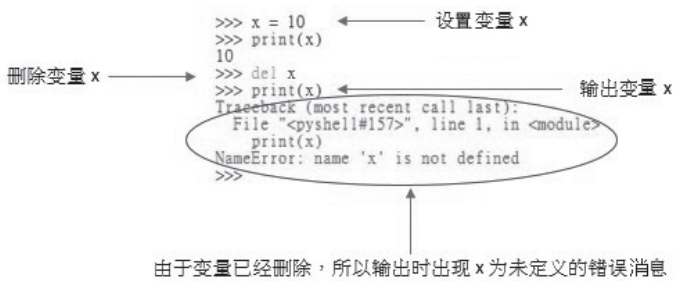
上述使用了 divmod（9,5）方法一次获得了元组值（1,4），第 8 章会介绍元组，然后使用多重指定将此元组（1,4）分别设置给 x 和 y 变量。

2-10 删除变量

程序设计时，如果某个变量不再需要，可以使用 del 指令将此变量删除，相当于可以收回原变量所占的内存空间，以节省内存空间。删除变量的格式如下：

```
del 变量名称
```

实例：验证变量名称回收后，将无法再使用。此例中尝试输出已删除的变量，然后程序出现错误消息。



2-11 Python 的断行

2-11-1 一行有多个语句

在 Python 中允许一行有多个，彼此用“;”隔开即可，尽管 Python 有提供此功能，不过笔者不鼓励如此撰写程序代码。

程序实例 ch2_3.py：一行有多个语句的实例。

```
# ch2_3.py
x = 10
print(x)
y = 20;print(y)          # 一行有两个语句,不过不鼓励这种写法
```

执行结果

```
====================== RESTART: D:\Python\ch2\ch2_3.py ======================
10
20
```

2-11-2　将一个语句分成多行

在设计大型程序时，常会碰上一个语句很长，需要分成两行或更多行撰写，此时可以在语句后面加上“\”符号，Python 解释器会将下一行的语句视为这一行的语句。特别注意，在“\”符号右边不可以加上任何符号或文字，即使是注释符号也是不允许的。

另外，也可以在语句内使用小括号，如果使用小括号，就可以在语句右边加上注释符号。

程序实例 ch2_4.py：将一个语句分成多行的应用。

```
# ch2_4.py
a = b = c = 10
x = a + b + c + 12
print(x)
# 续行方法1
y = a +\
    b +\
    c +\
    12
print(y)
# 续行方法2
z = ( a +      # 此处可以加上注释
      b +
      c +
      12 )
print(z)
```

执行结果

```
====================== RESTART: D:\Python\ch2\ch2_4.py ======================
42
42
42
```

2-12　专题——复利计算 / 计算圆面积与圆周长

2-12-1　银行存款复利的计算

程序实例 ch2_5.py：银行存款复利的计算。假设目前银行年利率是 1.5%，复利公式如下：

本金和 = 本金 ×(1 + 年利率)^{n} # n 是年

现有一笔 5 万元存款，请计算 5 年后的本金和。

```
# ch2_5.py
money = 50000 * ( 1 + 0.015 ) ** 5
print("本金和是")
print(money)
```

执行结果

```
========== RESTART: D:\Python\ch2\ch2_5.py ==========
本金和是
53864.20019421873
```

2-12-2 计算圆面积与周长

程序实例 ch2_6.py：假设圆半径是 5cm，圆面积与圆周长计算公式分别如下：

圆面积 = PI×r×r # PI = 3.14159, r 是半径

圆周长 = 2×PI×r

```
# ch2_6.py
PI = 3.14159
r = 5
print("圆面积:单位是平方厘米")
area = PI * r * r
print(area)
circurference = 2 * PI * r
print("圆周长:单位是厘米")
print(circurference)
```

执行结果

```
========== RESTART: D:\Python\ch2\ch2_6.py ==========
圆面积:单位是平方厘米
78.53975
圆周长:单位是厘米
31.4159
>>>
```

在程序语言的设计中，有一个概念是**常量**（named constant），这种常量是不可更改内容的。上述计算圆面积或圆周长所使用的 PI 是圆周率，这是一个固定的值，由于 Python 语言没有提供此**常量**（names constant）的语法，上述程序笔者用大写 PI 当作**常量**的变量，这是一种习惯，未来读者可以用这种方式处理固定内容的变量。

习题

1. 请重新设计 ch2_1.py，将打工时薪改为 150 元。(2-1 ~ 2-3 节)

```
========== RESTART: D:/Python/ex/ex2_1.py ==========
每年存款金额
252000
```

2. 重新设计 ch2_5.py，假设是单利率，5 年期间可以领多少利息？（2-5 ~ 2-7 节）

```
========== RESTART: D:/Python/ex/ex2_2.py ==========
利息总和
3750.0
```

3. 重新设计 ch2_5.py，假设期初本金是 100 000 元，年利率是 2%，这是复利计算，请问 10 年后本金总和是多少？（2-5 ~ 2-12 节）

```
========== RESTART: D:\Python\ex\ex2_3.py ==========
10年后本金和
121899.44199947573
```

4. 一个幼儿园买了 100 个苹果给学生当营养午餐，学生人数是 23 人，每个人午餐可以吃一个，请问这些苹果可以吃几天？第几天苹果会不够供应？同时列出缺少了几个。（2-5 ~ 2-12 节）

```
========== RESTART: D:/Python/ex/ex2_4.py ==========
苹果可以吃的天数
4
第几天产生苹果供应不足
5
不足数量
15
```

5. 地球和月球的距离是 384 400 千米，假设火箭飞行速度是每分钟 400 千米，请问从地球飞到月球需要多少分钟？（2-5 ~ 2-12 节）

```
========== RESTART: D:/Python/ex/ex2_5.py ==========
地球到月球所需分钟总数
961.0
```

6. 假设圆柱半径是 20 厘米，高度是 30 厘米，请计算此圆柱的体积。圆柱体积计算公式是：圆面积 × 圆柱高度。（2-5 ~ 2-12 节）

```
========== RESTART: D:/Python/ex/ex2_6.py ==========
圆柱体积:单位是立方厘米
37699.08
```

7. 圆周率 PI 是一个数学常数，常常使用希腊字母表示，它的物理意义是圆的周长和直径的比率。历史上第一个无穷级数公式称为**莱布尼茨**公式，它的计算公式如下：（2-5 ~ 2-12 节）

$$PI = 4\times\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}+\cdots\right)$$

请分别计算下列级数的执行结果。

（1）$PI = 4\times\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}\right)$

（2）$PI = 4\times\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}\right)$

（3）$PI = 4\times\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}+\frac{1}{13}\right)$

注　上述级数如果要收敛到我们熟知的 3.14159 需要相当长的级数计算。

```
========== RESTART: D:\Python\ex\ex2_7.py ==========
PI的值4 * (1 - 1/3 + 1/5 - 1/7 + 1/9)
3.3396825396825403
PI的值4 * (1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11)
2.9760461760461765
PI的值4 * (1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 + 1/13)
3.2837384837384844
>>>
```

莱布尼茨（Leibniz，1646—1716）是德国人，在世界数学舞台上占有一席之地，他本人另一个职业是律师，许多数学公式都是他在各大城市通勤期间完成的。数学历史上有一个两派说法的无解公案，有人认为他是微积分的发明人，也有人认为发明人是牛顿（Newton）。

8. 尼拉卡莎级数也是应用于计算圆周率 PI 的级数，此级数收敛的速度比莱布尼茨级数更好，更适合于用来计算 PI，它的计算公式如下：

$$PI=3+\frac{4}{2\times3\times4}-\frac{4}{4\times5\times6}+\frac{4}{6\times7\times8}-\cdots$$

请分别计算下列级数的执行结果。

（a） $PI=3+\frac{4}{2\times3\times4}-\frac{4}{4\times5\times6}+\frac{4}{6\times7\times8}-\cdots$

（b） $PI=3+\frac{4}{2\times3\times4}-\frac{4}{4\times5\times6}+\frac{4}{6\times7\times8}-\frac{4}{8\times9\times10}\cdots$

```
========== RESTART: D:/Python/ex/ex2_8.py ==========
PI的值3 + 4/(2*3*4) - 4/(4*5*6) + 4/(6*7*8)
3.145238095238095
PI的值3 + 4/(2*3*4) - 4/(4*5*6) + 4/(6*7*8) - 4/(8*9*10)
3.1396825396825396
```

03

第 3 章

Python 的基本数据类型

本章摘要

Python 的基本数据类型有下列几种。

（1）**数值**数据类型（numeric type）：常见的数值数据又可分成**整数**（int）、**浮点数**（float）、**复数**（complex number）（**不常用**所以不在本书讨论范围）。

（2）**布尔值**（Boolean）数据类型：也可归为数值数据类型。

（3）**文字序列**类型（text sequence type）：也就是字符串（string）数据类型。

（4）**字符组**（bytes，有的书称**字节**）数据类型：这是二进制的数据类型，长度是 8 位。

（5）**序列**类型（sequence type）：list（第 6 章说明）、tuple（第 8 章说明）。

（6）**对映**类型（mapping type）：dict（第 9 章说明）。

（7）**集合**类型（set type）：集合 set（第 10 章说明）、冻结集合 frozenset。

3-1 type( ) 函数

在正式介绍 Python 的数据类型前，笔者想介绍一下 type( ) 函数，这个函数可以列出变量的数据类型类别。这个函数在读者未来进入 Python 实战时非常重要，因为变量在使用前不需要声明，同时在程序设计过程中变量的数据类型会改变，我们常常需要使用此函数判断目前的变量数据类型。或是在进阶 Python 应用中，会调用一些方法（method），这些方法会返回一些数据，可以使用 type( ) 获得所返回的数据类型。

程序实例 ch3_1.py：列出数值变量的数据类型。

```
1  # ch3_1.py
2  x = 10
3  y = x / 3
4  print(x)
5  print(type(x))
6  print(y)
7  print(type(y))
```

执行结果

```
========== RESTART: D:/Python/ch3/ch3_1.py ==========
10
<class 'int'>
3.3333333333333335
<class 'float'>
```

从上述执行结果可以看到，变量 x 的内容是 10，数据类型是整数（int）。变量 y 的内容是 3.33…3，数据类型是浮点数（float）。下一节会说明为何是这样。

3-2 数值数据类型

3-2-1 整数 int

整数的英文是 integer，在计算机程序语言中一般用 int 表示。如果读者学过其他计算机语言，

在介绍整数时一定会告诉你，该计算机语言使用了多少空间存储整数，所以设计程序时整数的大小必须是在某一区间，否则会有**溢位**（overflow）造成数据不正确。例如，如果存储整数的空间是 32 位，则整数大小为 -2 147 483 648 ～ 2 147 483 647。在 Python 2.x 版时代，整数被限制在 32 位，另外还有长整数 long，空间大小是 64 位，所以可以存储的数值更大，达到 -9 223 372 036 854 775 808 ~ 9 223 372 036 854 775 807。在 Python 3 中已经将整数可以存储空间大小的限制取消了，所以没有 long 了，也就是说 int 可以是任意大小的数值。

英文 googol 是指自然数 10^{100}，计算机是用 $1e^{100}$ 显示，这是 1938 年美国数学家**爱德华・卡斯纳**（Edward Kasner）9 岁的侄子**米尔顿・西罗蒂**（Milton Sirotta）所创造的。下列是笔者尝试使用整数 int 显示此 googol 值。

```
>>> googol = 10 ** 100
>>> googol
10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000
```

3-2-2　浮点数

浮点数的英文是 float，既然整数大小没有限制，浮点数大小当然也没有限制。在 Python 语言中，带有小数点的数字称为**浮点数**。例如：

```
x = 10.3
```

表示 x 是浮点数。

3-2-3　基本数值数据的使用

Python 在声明变量时可以不用设置这个变量的数据类型，未来如果这个变量内容是放整数，这个变量就是整数（int）数据类型，如果这个变量内容是放浮点数，这个变量就是浮点数数据类型。整数与浮点数最大的区别是，整数不含小数点，浮点数含小数点。

程序实例 ch3_2.py：测试浮点数。

```
1  # ch3_2.py
2  x = 10.0
3  print(x)
4  print(type(x))
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_2.py ======================
10.0
<class 'float'>
```

在程序实例 ch3_1.py 中，x 变量的值是“10”，列出 x 变量是**整数变量**，在这个实例中，x 变量的值是“10.0”，列出 x 变量是**浮点数变量**。

3-2-4　整数与浮点数的运算

Python 程序设计时不相同的数据类型也可以执行运算，程序设计时常常会发生整数与浮点数之

间的数据运算，Python 具有简单的自动转换能力，在计算时会将整数转换为浮点数再执行运算。

程序实例 ch3_3.py：不同数据类型的运算。

```
# ch3_3.py
x = 10
y = x + 5.5
print(x)
print(type(x))
print(y)
print(type(y))
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_3.py ======================
10
<class 'int'>
15.5
<class 'float'>
```

上述变量 y，由于是整数与浮点数的加法，所以结果是浮点数。此外，某一个变量如果是整数，但是如果最后所存储的值是浮点数，Python 也会将此变量转成浮点数。

程序实例 ch3_4.py：整数转换成浮点数的应用。

```
# ch3_4.py
x = 10
print(x)
print(type(x))      # 加法前列出x数据类型
x = x + 5.5
print(x)
print(type(x))      # 加法后列出x数据类型
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_4.py ======================
10
<class 'int'>
15.5
<class 'float'>
```

原先变量 x 所存储的值是整数，所以列出的是整数。后来存储了浮点数，所以列出的是浮点数。

3-2-5 二进制整数与函数 bin( )

可以用二进制方式代表整数，Python 中定义凡是以 0b 开头的数字，代表这是二进制的整数。

bin( ) 函数可以将一般整数数字转换为二进制。

程序实例 ch3_5.py：将十进制数值与二进制数值互转的应用。

```
# ch3_5.py
x = 0b1101          # 这是二进制整数
print(x)            # 列出十进制的结果
y = 13              # 这是十进制整数
print(bin(y))       # 列出转换成二进制的结果
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_5.py ======================
13
0b1101
```

3-2-6　八进制整数与函数 oct( )

可以用八进制方式代表整数，Python 中定义凡是以 0o 开头的数字，代表这是八进制的整数。

oct( ) 函数可以将一般数字转换为八进制。

程序实例 ch3_6.py：将十进制数值与八进制数值互转的应用。

```
1  # ch3_6.py
2  x = 0o57              # 这是八进制整数
3  print(x)              # 列出十进制的结果
4  y = 47                # 这是十进制整数
5  print(oct(y))         # 列出转换成八进制的结果
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_6.py ======================
47
0o57
```

3-2-7　十六进制整数与函数 hex( )

可以用十六进制方式代表整数，Python 中定义凡是以 0x 开头的数字，代表这是十六进制的整数。

hex( ) 函数可以将一般数字转换为十六进制。

程序实例 ch3_7.py：将十进制数值与八进制数值互转的应用。

```
1  # ch3_7.py
2  x = 0x5D              # 这是十六进制整数
3  print(x)              # 列出十进制的结果
4  y = 93                # 这是十进制整数
5  print(hex(y))         # 列出转换成十六进制的结果
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_7.py ======================
93
0x5d
```

3-2-8　强制数据类型的转换

有时候设计程序时，可以自行强制使用下列函数，转换变量的数据类型。

int( )：将数据类型强制转换为整数。

float( )：将数据类型强制转换为浮点数。

程序实例 ch3_8.py：将浮点数强制转换为整数的运算。

```
1  # ch3_8.py
2  x = 10.5
3  print(x)
4  print(type(x))       # 加法前列出x数据类型
5  y = int(x) + 5
6  print(y)
7  print(type(y))       # 加法后列出y数据类型
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_8.py ======================
10.5
<class 'float'>
15
<class 'int'>
```

程序实例 ch3_9.py：将整数强制转换为浮点数的运算。

```
1  # ch3_9.py
2  x = 10
3  print(x)
4  print(type(x))       # 加法前列出x数据类型
5  y = float(x) + 10
6  print(y)
7  print(type(y))       # 加法后列出y数据类型
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_9.py ======================
10
<class 'int'>
20.0
<class 'float'>
```

3-2-9 数值运算常用的函数

下列是数值运算时常用的函数。

abs( )：计算绝对值。

pow (x,y)：返回 x 的 y 次方。

round( )：这是采用运算法则的 Bankers Rounding 概念，如果处理位数左边是**奇数则使用四舍五入**，如果处理位数左边是**偶数则使用五舍六入**，例如，round(1.5)=2，round(2.5)=2。

处理小数时，第 2 个参数代表取到小数第几位，1 代表取到小数第 1 位。根据保留小数位的后两位，采用 "50" 舍去，"51" 进位，例如，round(2.15,1)=2.1，round(2.25,1)=2.2，round(2.151,1)=2.2，round(2.251,1)=2.3。

程序实例 ch3_10.py：abs( )、pow( )、round( )、round(x,n）函数的应用。

```
# ch3_10.py
x = -10
print("以下輸出abs()函數的应用")
print(x)            # 輸出x变量
print(abs(x))       # 輸出abs(x)
x = 5
y = 3
print("以下輸出pow()函數的应用")
print(pow(x, y))    # 輸出pow(x,y)
x = 47.5
print("以下輸出round(x)函數的应用")
print(x)            # 輸出x变量
print(round(x))     # 輸出round(x)
x = 48.5
print(x)            # 輸出x变量
print(round(x))     # 輸出round(x)
x = 49.5
print(x)            # 輸出x变量
print(round(x))     # 輸出round(x)
print("以下輸出round(x,n)函數的应用")
x = 2.15
print(x)            # 輸出x变量
print(round(x,1))   # 輸出round(x,1)
x = 2.25
print(x)            # 輸出x变量
print(round(x,1))   # 輸出round(x,1)
x = 2.151
print(x)            # 輸出x变量
print(round(x,1))   # 輸出round(x,1)
x = 2.251
print(x)            # 輸出x变量
print(round(x,1))   # 輸出round(x,1)
```

执行结果

```
========== RESTART: D:\Python\ch3\ch3_10.py ==========
以下輸出abs()函數的应用
-10
10
以下輸出pow()函數的应用
125
以下輸出round(x)函數的应用
47.5
48
48.5
48
49.5
50
以下輸出round(x,n)函數的应用
2.15
2.1
2.25
2.2
2.151
2.2
2.251
2.3
```

需留意的是，使用上述 abs( )、pow( ) 或 round( ) 函数，尽管可以得到运算结果，但是原先变量的值是没有改变的。

3-2-10　科学记数法

科学记数的概念如下，将一个数字转换成下列数学式：

$a \times 10^n$

a 是浮点数，例如，123456 可以表示为 1.23456 ×10^5，以 10 为基底数我们用 E 或 e 表示，指数部分则转为一般数字，然后省略“×”符号，最后表达式如下：

```
1.23456E+5
```

或

```
1.23456e+5
```

如果是碰上小于 1 的数值，则 E 或 e 右边是负值“-”。例如，0.000123 转成科学记数法，最后表达式如下：

```
1.23E-4
```

或

```
1.23e-4
```

下列是示范输出。

```
>>> x = 1.23456E+5
>>> x
123456.0
>>> y = 1.23e-4
>>> y
0.000123
```

4-2-2 节和 4-2-3 节会介绍将一般数值转成科学记数法输出的方式，以及格式化输出方式。

3-3 布尔值数据类型

Python 的布尔值（Boolean）数据类型的值有两种，True（真）或 False（伪）。它的数据类型代号是 bool。布尔值一般应用在程序流程的控制中，特别是在条件表达式中，程序可以根据这个布尔值判断应该如何执行下一步工作。

程序实例 ch3_11.py：列出布尔值 True 与布尔值 False 的数据类型。

```
1  # ch3_11.py
2  x = True
3  print(x)
4  print(type(x))      # 列出x数据类型
5  y = False
6  print(y)
7  print(type(y))      # 列出y数据类型
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_11.py ======================
True
<class 'bool'>
False
<class 'bool'>
```

如果将布尔值数据类型强制转换成整数，如果原值是 True，将得到 1；如果原值是 False，将得到 0。

程序实例 ch3_12.py：将布尔值强制转换为整数，同时列出转换的结果。

```
1  # ch3_12.py
2  x = True
3  print(int(x))
4  print(type(x))      # 列出x数据类型
5  y = False
6  print(int(y))
7  print(type(y))      # 列出y数据类型
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_12.py ======================
1
<class 'bool'>
0
<class 'bool'>
```

在本章一开始说过，有时候也可以将布尔值当作数值数据，因为 True 会被视为 1，False 会被视为 0，可以参考下列实例。

程序实例 ch3_13.py：将布尔值与整数值相加，并观察最后变量数据类型，可以发现，最后的变量数据类型是整数值。

```
1  # ch3_13.py
2  xt = True
3  x = 1 + xt
4  print(x)
5  print(type(x))      # 列出x数据类型
6
7  yt = False
8  y = 1 + yt
9  print(y)
10 print(type(y))      # 列出y数据类型
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_13.py ======================
2
<class 'int'>
1
<class 'int'>
```

此外，在程序设计中 False 值不一定是要经过条件判断是 False，才可以得到 False，下列情况也会被视为 False。

布尔值 False
整数 0
浮点数 0.0
空字符串 ''
空列表 [ ]
空元组 ( )
空字典 { }
空集合 set( )
None
至于其他的都会被视为 True。

3-4 字符串数据类型

字符串（string）数据是指**两个单引号**（‘）之间**或**是**两个双引号**（“）之间任意个数字元符号的数据，它的数据类型代号是 str。在英文字符串的使用中常会发生某字中间有单引号的情况，其实这是文字的一部分，如下所示：

```
This is James's ball
```

如果用单引号去处理上述字符串将产生错误，如下所示：

```
>>> x = 'This is James's ball'
SyntaxError: invalid syntax
>>>
```

碰到这种情况，可以用双引号解决，如下所示：

```
>>> x = "This is James's ball"
>>> print(x)
This is James's ball
>>>
```

程序实例 ch3_14.py：使用单引号与双引号设置与输出字符串数据的应用。

```
1  # ch3_14.py
2  x = "DeepStone means Deep Learning"    # 双引号设置字符串
3  print(x)
4  print(type(x))                         # 列出x字符串数据类型
5  y = '深石数字 - 深度学习滴水穿石'       # 单引号设置字符串
6  print(y)
7  print(type(y))                         # 列出y字符串数据类型
```

执行结果

```
====================== RESTART: D:\Python\ch3\ch3_14.py ======================
DeepStone means Deep Learning
<class 'str'>
深石数字 - 深度学习滴水穿石
<class 'str'>
```

3-4-1 字符串的连接

数学的运算符“+”，可以进行两个字符串相加的操作，产生新的字符串。

程序实例 ch3_15.py：字符串连接的应用。

```
1  # ch3_15.py
2  num1 = 222
3  num2 = 333
4  num3 = num1 + num2
5  print("以下是数值相加")
6  print(num3)
7  numstr1 = "222"
8  numstr2 = "333"
9  numstr3 = numstr1 + numstr2
10 print("以下是由数值组成的字符串相加")
11 print(numstr3)
12 numstr4 = numstr1 + " " + numstr2
13 print("以下是由数值组成的字符串相加，同时中间加上一空格")
14 print(numstr4)
15 str1 = "DeepStone "
16 str2 = "Deep Learning"
17 str3 = str1 + str2
18 print("以下是一般字符串相加")
19 print(str3)
```

执行结果

```
====================== RESTART: D:\Python\ch3\ch3_15.py ======================
以下是数值相加
555
以下是由数值组成的字符串相加
222333
以下是由数值组成的字符串相加，同时中间加上一空格
222 333
以下是一般字符串相加
DeepStone Deep Learning
```

3-4-2　处理多于一行的字符串

程序设计时如果字符串长度多于一行，可以使用三个单引号（或是三个双引号）将字符串括起来即可。

程序实例 ch3_16.py：使用三个单引号处理多于一行的字符串。

```
1  # ch3_16.py
2  str1 = '''Silicon Stone Education is an unbiased organization
3  concentrated on bridging the gap ... '''
4  print(str1)
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_16.py ======================
Silicon Stone Education is an unbiased organization
concentrated on bridging the gap ...
```

读者可以留意第 2 行 Silicon 左边的 3 个单引号和第 3 行末端的 3 个单引号，另外，上述第 2 行若是少了 "str1 = "，3 个单引号间的跨行字符串就变成了程序的注释。

3-4-3　转义字符

在字符串使用中，如果字符串内有一些特殊字符，例如单引号、双引号等，必须在此特殊字符前加上“\”（反斜杠），才可正常使用，这种含有“\”符号的字符称**为转义字符**（Escape Character）。

转义字符	Hex 值	意义	转义字符	Hex 值	意义
\'	27	单引号	\n	0A	换行
\"	22	双引号	\o		八进制表示
\\	5C	反斜杠	\r	0D	光标移至最左位置
\a	07	响铃	\x		十六进制表示
\b	08	Backspace 键	\t	09	Tab 键
\f	0C	换页	\v	0B	垂直定位

字符串使用中特别是碰到字符串含有单引号时，如果是使用单引号定义这个字符串，必须要使用此**转义字符**，才可以顺利显示，可参考 ch3_17.py 的第 3 行。如果是使用双引号定义字符串，则可以不必使用**转义字符**，可参考 ch3_17.py 的第 6 行。

程序实例 ch3_17.py：转义字符的应用，这个程序第 9 行增加了“\t”字符，所以“can't”跳到下一个 Tab 键位置输出。同时有“\n”字符，这是换行符号，所以“loving”跳到下一行输出。

```
1  # ch3_17.py
2  #以下输出使用单引号设置的字符串,需使用\'
3  str1 = 'I can\'t stop loving you.'
4  print(str1)
5  #以下输出使用双引号设置的字符串,不需使用\'
6  str2 = "I can't stop loving you."
7  print(str2)
8  #以下输出有\t和\n字符
9  str3 = "I \tcan't stop \nloving you."
10 print(str3)
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_17.py ======================
I can't stop loving you.
I can't stop loving you.
I       can't stop
loving you.
```

3-4-4 str( ) 函数

str( ) 函数有如下几个用法。

（1）设置空字符串。

```
>>> x = str()                    # 设置空字符串
>>> x
''
>>> print(x)

>>>
```

（2）设置字符串。

```
>>> x = str('ABC')
>>> x
'ABC'
```

（3）强制将数值数据转换为字符串数据。

```
>>> x = 123
>>> y = str(x)
>>> y
'123'
```

程序实例 ch3_18.py：使用 str( ) 函数将数值数据强制转换为字符串的应用。

```
1  # ch3_18.py
2  num1 = 222
3  num2 = 333
4  num3 = num1 + num2
5  print("这是数值相加")
6  print(num3)
7  str1 = str(num1) + str(num2)
8  print("强制转换为字符串相加")
9  print(str1)
```

执行结果

```
====================== RESTART: D:\Python\ch3\ch3_18.py ======================
这是数值相加
555
强制转换为字符串相加
222333
```

上述字符串相加，读者可以想成是字符串连接，执行结果是一个字符串，所以上述执行结果 555 是数值数据，222333 则是一个字符串。

3-4-5　将字符串转换为整数

int( ) 函数可以将字符串转为整数，在未来的程序设计中也常会发生将字符串转换为整数数据，下面将直接以实例做说明。**注**：如果数字是非数字字符组成，会产生错误。

程序实例 ch3_19.py：将字符串数据转换为整数数据的应用。

```
1  # ch3_19.py
2  x1 = "22"
3  x2 = "33"
4  x3 = x1 + x2
5  print(x3)              # 打印字符串相加
6  x4 = int(x1) + int(x2)
7  print(x4)              # 打印整数相加
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_19.py ======================
2233
55
```

上述执行结果 55 是数值数据，2233 则是一个字符串。

3-4-6　字符串与整数相乘产生字符串复制效果

在 Python 中允许将字符串与整数相乘，结果是字符串将重复该整数的次数。

程序实例 ch3_20.py：字符串与整数相乘的应用。

```
1  # ch3_20.py
2  x1 = "A"
3  x2 = x1 * 10
4  print(x2)              # 打印字符串乘以整数
5  x3 = "ABC"
6  x4 = x3 * 5
7  print(x4)              # 打印字符串乘以整数
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_20.py ======================
AAAAAAAAAA
ABCABCABCABCABC
```

3-4-7 聪明地使用字符串加法和换行字符 \n

有时在设计程序时，想将字符串分行输出，可以使用字符串加法功能，在加法过程中加上换行字符“\n”即可产生字符串分行输出的结果。

程序实例 ch3_21.py：将数据分行输出的应用。

```
# ch3_21.py
str1 = "洪锦魁著作"
str2 = "HTML5+CSS3王者归来"
str3 = "Python数据科学零基础一本通"
str4 = str1 + "\n" + str2 + "\n" + str3
print(str4)
```

执行结果

```
========== RESTART: D:\Python\ch3\ch3_21.py ==========
洪锦魁著作
HTML5+CSS3王者归来
Python数据科学零基础一本通
```

3-4-8 字符串前加 r

在使用 Python 时，如果在字符串前加上 r，可以防止转义字符被转义，可参考 3-4-3 节的转义字符表，相当于可以取消转义字符的功能。

程序实例 ch3_22.py：字符串前加上 r 的应用。

```
# ch3_22.py
str1 = "Hello!\nPython"
print("不含r字符的输出")
print(str1)
str2 = r"Hello!\nPython"
print("含r字符的输出")
print(str2)
```

执行结果

```
========== RESTART: D:\Python\ch3\ch3_22.py ==========
不含r字符的输出
Hello!
Python
含r字符的输出
Hello!\nPython
```

3-5 字符串与字符

在 Python 中没有所谓的字符（character）数据，如果字符串含一个字符，我们称这是含一个字符的字符串。

3-5-1　ASCII 码

计算器内部最小的存储单位是**位**（bit），这个**位**只能存储 0 或 1。一个英文字符在计算器中是被存储成 8 个位的一连串 0 或 1 中，存储这个英文字符的编码称为 ASCII（American Standard Code for Information Interchange，美国信息交换标准程序代码），有关 ASCII 码的内容可以参考附录 E。

在这个 ASCII 表中由于是用 8 位定义一个字符，所以使用了 0 ～ 127 定义了 128 个字符，在这 128 个字符中有 33 个字符是无法显示的控制字符，其他则是可以显示的字符。不过有一些应用程序扩充了功能，让部分控制字符可以显示，例如，扑克牌花色、笑脸等。至于其他可显示字符有一些符号，例如 +、−、=、0 ～ 9、A ～ Z 或 a ～ z 等。这些符号每一个都有一个编码，我们称这个编码是 ASCII 码。

可以使用下列函数执行数据的转换。

chr (x)：返回函数 x 值的 ASCII 或 Unicode 字符。

例如，从 ASCII 表可知，字符 a 的 ASCII 码值是 97，可以使用下列方式打印出此字符。

```
>>> x = 97
>>> print(chr(x))
a
```

英文小写与英文大写的码值相差 32，可参考下列实例。

```
>>> x = 97
>>> x -= 32
>>> print(chr(x))
A
```

3-5-2　Unicode 码

计算机是美国发明的，因此 ASCII 码对于英语系国家的确很好用，但是地球是一个多种族的社会，存在几百种语言与文字，ASCII 所能容纳的字符是有限的，只要随便一个不同语系的外来词，例如café，含重音字符就无法显示了，更何况有几万中文字或其他语系文字。为了让全球语系的用户可以彼此用计算机沟通，因此有了 Unicode 码。

Unicode 码的基本精神是，世上所有的文字都有一个码值，可以参考下列网页：

http://www.unicode.org/charts

目前，Unicode 内定义了超过 11 万的文字，它的定义方式是以“\u”开头后面有 4 个十六进制的数字，所以是从“\u0000”至“\uFFFF”。在上述网页中可以看到不同语系表，其中，East Asian Scripts 字段可以看到 CJK（Chinese，Japanese，Korean），在这里可以看到汉字的 Unicode 码值表，CJK 统一汉字的编码为 4E00 ～ 9FBB。

在 Unicode 编码中，前 128 个码值是保留给 ASCII 码使用，所以对于原先存在 ASCII 码中的英文大小写、标点符号等，是可以正常在 Unicode 码中使用的，Unicode 编码中经常用的是 ord( ) 函数。

ord (x)：可以返回函数字符参数 x 的 Unicode 码值，如果是中文字也可返回 Unicode 码值。如果是英文字符，Unicode 码值与 ASCII 码值是一样的。有了这个函数，可以很轻易地获得字符的 Unicode 码值。

程序实例 ch3_23.py：这个程序首先会将整数 97 转换成英文字符‘a’，然后将字符‘a’转换成 Unicode 码值，最后将中文字‘魁’转成 Unicode 码值。

```
# ch3_23.py
x1 = 97
x2 = chr(x1)
print(x2)                # 输出数值97的字符
x3 = ord(x2)
print(x3)                # 输出字符x3的Unicode(十进制)码值
x4 = '魁'
print(hex(ord(x4)))      # 输出字符'魁'的Unicode(十六进制)码值
```

执行结果

```
====================== RESTART: D:/Python/ch3/ch3_23.py ======================
a
97
0x9b41
```

3-5-3 utf-8 编码

utf-8 是针对 Unicode 字符集的可变长度编码方式，这是 Internet 目前所遵循的编码方式，在这种编码方式下，utf-8 使用 1 ～ 4 个 byte 表示一个字符，这种编码方式会根据不同的字符变化编码长度。

- ASCII 使用 utf-8 编码规则

对于 ASCII 字符而言，基本上它使用 1 个 byte 存储 ASCII 字符，utf-8 的编码方式是 byte 的第一个位是 0，其他 7 个位则是此字符的 ASCII 码值。

- 中文字的 utf-8 编码规则

对于需要 n 个 byte 编码的 Unicode 汉字字符而言，例如需要 3 个 byte 编码的汉字，第一个 byte 的前 n(3) 位皆设为 1，n+1(4) 设为 0。后面第 2 和第 3 个 byte 的前 2 位是 10，其他没有说明的二进制全部是此汉字字符的 Unicode 码。依照此规则，可以得到汉字的 utf-8 编码规则如下：

```
1110xxxx 10xxxxxx 10xxxxxx          # xx 就是要填入的 Unicode 码
```

例如，从 ch3_23.py 的执行结果可知“魁”的 Unicode 码值是 0x9b41，如果转成二进制方式则如下所示：

```
10011011 01000001
```

我们可以用下列更细的方式，将“魁”的 Unicode 码值填入 xx 内。

utf-8中文编码规则	1	1	1	0	x	x	x	x	1	0	x	x	x	x	x	x	1	0	x	x	x	x	x	x
魁的Unicode编码					1	0	0	1			1	0	1	1	0	1			0	0	0	0	0	1
魁的utf-8编码	1	1	1	0	1	0	0	1	1	0	1	0	1	1	0	1	1	0	0	0	0	0	0	1

从上图可以得到“魁”的 utf-8 编码结果是 0xe9ad81，3-6-1 节的实例 2 也可以验证这个结果。

3-6 bytes 数据

使用 Python 处理一般字符串数据时，可以很放心地使用 Unicode 字符串 str 数据类型，至于 Python 内部如何处理可以不用理会，这些事情 Python 的直译程序会处理。

但是有一天需要与外界沟通或交换数据时，特别是我们使用中文，如果不懂中文字符串与 bytes 数据的转换，所获得的数据将会是乱码。例如，设计电子邮件的接收程序，所接收的可能是 bytes 数据，这时必须学会将 bytes 数据转成 Unicode 字符串，否则会有乱码产生。或是有一天你要设计供中国人使用的网络聊天室，必须设计将使用者所传送的 Unicode 中文字符串转成 bytes 数据传上聊天室，然后也要设计将网络接收的 bytes 数据转成 Unicode 中文字符串，这个聊天室才可以顺畅使用。

bytes 数据格式是在字符串前加上 b，例如，下列是“魁”的 bytes 数据。

```
b'\xe9\xad\x81'
```

如果是英文字符串的 bytes 数据格式，相对单纯地会显示原始的字符，例如，下列是字符串“abc”的 bytes 数据。

```
b'abc'
```

3-6-1　Unicode 字符串转成 bytes 数据

将 Unicode 字符串转成 bytes 数据称为**编码**（encode），所使用的是 encode( ) 函数，这个方法的参数是指出编码的方法，可以参考下列表格。

编码	说明
'ascii'	标准 7 位的 ASCII 编码
'utf-8'	Unicode 可变长度编码，这也是最常使用的编码
'cp-1252'	一般英文 Windows 操作系统编码
'cp950'	繁体中文 Windows 操作系统编码
'unicode-escape'	Unicode 的常数格式，\uxxxx 或 \Uxxxxxxxx

如果 Unicode 字符串是英文则转成 bytes 数据相对容易，因为对于 utf-8 格式编码，Unicode 也是用一个 byte 存储每个字符串的字符。

实例 1：英文 Unicode 字符串数据转成 bytes 数据。

假设有一个字符串 string，内容是‘abc’，可以使用下列方法设置，同时检查此字符串的长度。

```
>>> string = 'abc'
>>> len(string)
3
```

下面将 Unicode 字符串 string 用 utf-8 编码格式转成 bytes 数据，然后列出 bytes 数据的长度、数据类型，以及 bytes 数据的内容。

```
>>> stringBytes = string.encode('utf-8')
>>> len(stringBytes)
3
>>> type(stringBytes)
<class 'bytes'>
>>> stringBytes
b'abc'
```

实例 2：中文 Unicode 字符串数据转成 bytes 数据。

假设有一个字符串 name，内容是‘洪锦魁’，可以使用下列方法设置，同时检查此字符串的长度。

```
>>> name = '洪锦魁'
>>> len(name)
3
```

下面将 Unicode 字符串 name 用 utf-8 编码格式转成 bytes 数据，然后列出 bytes 数据的长度、数据类型，以及 bytes 数据的内容。

```
>>> nameBytes = name.encode('utf-8')
>>> len(nameBytes)
9
>>> type(nameBytes)
<class 'bytes'>
>>> nameBytes
b'\xe6\xb4\xaa\xe9\x94\xa6\xe9\xad\x81'
```

由上述数据可以得到原来 Unicode 字符串用了 3byte 存储一个中文字，所以 3 个中文字获得了 bytes 的数据长度是 9。

3-6-2 bytes 数据转成 Unicode 字符串

对于一个专业的 Python 程序设计师而言，常常需要从网络取得数据，所取得的是 bytes 数据，这时需要将此数据转成 Unicode 字符串，将 bytes 数据转成 Unicode 字符串可以称为**译码**，所使用的是 decode( ) 函数，这个方法的参数是指出编码的方法，与 encode( ) 函数相同。

实例 1：bytes 数据转成 Unicode 字符串数据。

```
>>> stringUcode = stringBytes.decode('utf-8')
>>> len(stringUcode)
3
>>> stringUcode
'abc'
```

实例 2：bytes 数据转成 Unicode 字符串数据。

下面是将 nameBytes 数据使用 utf-8 编码格式转成 Unicode 字符串的方法，同时列出字符串长度和字符串内容。

```
>>> nameUcode = nameBytes.decode('utf-8')
>>> len(nameUcode)
3
>>> nameUcode
'洪锦魁'
```

3-7 专题——地球到月球时间计算 / 计算坐标轴两点之间的距离

3-7-1 计算地球到月球所需时间

马赫是音速的单位，主要是为了纪念奥地利科学家**恩斯特·马赫**（Ernst Mach）而命名，一马赫就是一倍音速，它的速度大约是每小时 1225 千米。

程序实例 ch3_24.py：从地球到月球约 384 400 千米，假设火箭的速度是一马赫，设计一个程序计算需要多少天、多少小时才可抵达月球。这个程序省略分钟数。

```
1  # ch3_24.py
2  dist = 384400                    # 地球到月亮距离
3  speed = 1225                     # 马赫速度每小时1225千米
4  total_hours = dist // speed      # 计算小时数
5  days = total_hours // 24         # 商 = 计算天数
6  hours = total_hours % 24         # 余数 = 计算小时数
7  print("总共需要天数")
8  print(days)
9  print("小时数")
10 print(hours)
```

执行结果

```
========== RESTART: D:\Python\ch3\ch3_24.py ==========
总共需要天数
13
小时数
1
```

由于尚未介绍完整的格式化程序输出，所以使用上述方式输出，第 4 章会改良上述程序。Python 之所以可以成为当今最流行的程序语言，主要是它有丰富的函数库与方法，上述求商（第 5 行）和余数（第 6 行），在 2-9 节中介绍了 divmod( ) 函数，其实可以用 divmod( ) 函数一次取得商和余数，如下：

```
商，余数 = divmod(被除数，除数)                # 函数方法
days, hours = divmod(total_hours, 24)      # 本程序应用方式
```

程序实例 ch3_25.py：使用 divmod( ) 函数重新设计 ch3_24.py。

```
1  # ch3_25.py
2  dist = 384400                          # 地球到月亮距离
3  speed = 1225                           # 马赫速度每小时1225千米
4  total_hours = dist // speed            # 计算小时数
5  days, hours = divmod(total_hours, 24)  # 商和余数
6  print("总共需要天数")
7  print(days)
8  print("小时数")
9  print(hours)
```

执行结果　与 ch3_24.py 相同。

3-7-2　计算坐标轴两个点之间的距离

有两个点坐标分别是（x1, y1）、（x2, y2），这两个点的距离计算公式如下。

$$\sqrt{(x1-x2)^2+(y1-y2)^2}$$

可以将上述公式转成下列计算机数学表达式。

```
dist = ( (x1 - x2)² + (y1 - y2)² ) ** 0.5        # ** 0.5 相当于开根号
```

在人工智能的应用中，常用点坐标代表某一个对象的**特征**（feature），计算两个点之间的距离，相当于可以了解物体间的相似程度。距离越短代表相似度越高，距离越长代表相似度越低。

程序实例 ch3_26.py：有两个点坐标分别是 (1, 8）与 (3, 10)，请计算这两个点之间的距离。

```
1  # ch3_26.py
2  x1 = 1
3  y1 = 8
4  x2 = 3
5  y2 = 10
6  dist = ((x1 - x2) ** 2 + ((y1 - y2) ** 2)) ** 0.5
7  print("两点的距离是")
8  print(dist)
```

执行结果

```
========== RESTART: D:\Python\ch3\ch3_26.py ==========
两点的距离是
2.8284271247461903
```

习题

1. 假设 a 是 10，b 是 18，c 是 5，请计算下列执行结果，取整数结果。(3-2 节)

（a）s = a + b – c　　（b）s = 2 * a + 3 – c　　（c）s = b * c + 20 / b

（d）s = a % c * b + 10　　（e）s = a ** c – a * b * c

```
========== RESTART: D:\Python\ex\ex3_1.py ==========
13
18
42.5
10
99600
```

2. 请重新设计第 2 章习题 2，请使用 int( ) 函数，以整数列出本金和。(3-2 节)

```
========== RESTART: D:\Python\ex\ex3_2.py ==========
本金和
106120
```

3. 请重新设计第 2 章习题 2，使用 round( ) 函数，以整数列出本金和。(3-2 节)

```
========== RESTART: D:\Python\ex\ex3_3.py ==========
本金和
106121
```

4. 地球和月球的距离是 384 400 千米，假设火箭飞行速度是每分钟 250 千米，请问从地球飞到月球需要多少天、多少小时、多少分钟，请舍去秒钟。(3-2 节)

```
========== RESTART: D:/Python/ex/ex3_4.py ==========
天总数
1
小时数
25
分钟数
37
```

5. 请列出你自己名字十进制的 Unicode 码值。(3-5 节)

```
========== RESTART: D:/Python/ex/ex3_5.py ==========
洪
27946
锦
38182
魁
39745
```

6. 请列出你自己名字十六进制的 Unicode 码值。(3-5 节)

```
====================== RESTART: D:/Python/ex/ex3_6.py ======================
洪
0x6d2a
锦
0x9526
魁
0x9b41
```

7. 请将 Unicode 字符串“Python 王者归来”转成 bytes 数据，然后输出 bytes 数据。(3-6 节)

```
====================== RESTART: D:/Python/ex/ex3_7.py ======================
Unicode字符串内容
Python王者归来
bytes数据内容
b'Python\xe7\x8e\x8b\xe8\x80\x85\xe5\xbd\x92\xe6\x9d\xa5'
将bytes数据转回Unicode字符串
Python王者归来
```

8. 重新设计 ch3_25.py，需计算至分钟与秒钟数。(3-7 节)

```
====================== RESTART: D:/Python/ex/ex3_8.py ======================
313.7959183673469
总共需要天数
13.0
小时数
1.7959183673469283
分钟数
47
秒钟数
45
```

9. 请修改 ch3_26.py，计算这两个点坐标（1, 8）与（3, 10）距坐标原点（0, 0）的距离。

```
====================== RESTART: D:/Python/ex/ex3_9.py ======================
坐标(1, 8)点与坐标原点(0, 0)的距离是
8.06225774829855
坐标(3, 10)点与坐标原点(0, 0)的距离是
10.44030650891055
```