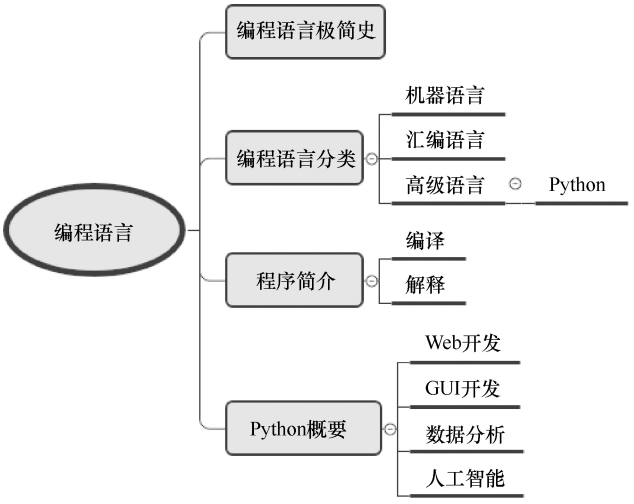
第 1 章　编程语言

使用计算机，离不开软件，软件都是用编程语言开发的。本书从现在开始，就要向读者讲述编程语言的那些事儿。当然，现在本星球上的编程语言比较多，但它们并非杂乱无章。本章将简要介绍一些相关的基本知识，并最终确定本书要学习的语言 Python。

知识技能导图



1.1　编程语言简史

Programming Language，即“编程语言”或者“程序设计语言”。这种语言不同于汉语、英语等语言。后者是随着人类文化发展而演化的语言，称为“自然语言”。而编程语言是“人造”的，属于“人工语言”（或“人造语言”），是用来定义计算机程序的形式语言。

世界上第一台电子数字计算设备是 1937 年设计的“阿塔纳索夫-贝瑞计算机”（Atanasoff–Berry Computer，通常简称 ABC 计算机）。当然，ABC 计算机并不能进行编程，它能做的就是求解线性方程组，也不是冯·诺伊曼结构的。20 世纪 40 年代以后，逐渐发展出来的电子计算机都是冯·诺伊曼结构的，并延续至今。

相对于计算机的发展，编程语言出现得更早。从 19 世纪初起，“程序”就被用在提花织机、音乐盒和钢琴等机器上。只是到后来，随着电子计算机的飞速发展，“软件”已经成为不可或缺的组成部分，“编程语言”才与电子计算机密切绑定在一起。

现在，人类所使用的编程语言有多少种？

难以统计！

在《维基百科》上列出了目前已知的编程语言（https://en.wikipedia.org/wiki/List_of_

programming_languages）。为什么需要这么多编程语言呢？比较有说服力的回答可能是“不同的语言解决不同的问题”，以及“开发者有自己的喜好”。不管什么理由，现实就是人类创造了多种多样的编程语言。

所以，在下述“编程语言极简史”中只能选择所谓的“主流语言”了。

- 1950 年以前是编程语言的“史前”年代。虽然已经有了用“打孔卡”方式编程（见图 1-1-1）的记载，但并没有被广泛采用。

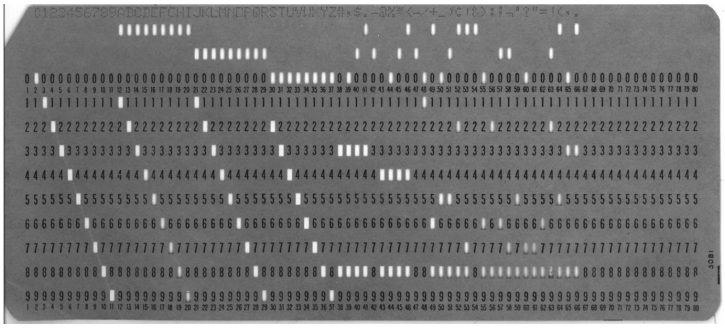


图 1-1-1　80 列、矩形孔的标准 IBM 打孔卡（源自《维基百科》网站）

- 1957 年，Fortran 诞生，它是世界上第一个被正式采用并流传至今的高级编程语言。发明者是 John Warner Backus，此处应当献上敬意和崇拜（以下列出的各项语言发明者，亦或该语言发明团队的负责人、主要设计者，为了简便，统一称为“发明者”，并且都要献上敬意和崇拜）。
- 1958 年，LISP 诞生。发明者 John McCarthy。
- 1964 年，BASIC 诞生。发明者 John G.Kemeny 和 Thomas E.Kurtz。
- 1970 年，Pascal 诞生。发明者 Niklaus Emil Wirth。此外，他还是 Algol W、Modula、Oberon、Euler 等语言的发明者。
- 1972 年，C 诞生。发明者 Dennis Ritchie 和 Ken Thompson。
- 1983 年，C++诞生。发明者 Bjarne Stroustrup。
- 1986 年，Objective-C 诞生。发明者 Tom Love 和 Brad Cox。
- 1987 年，Perl 诞生。发明者 Larry Wall。
- 1991 年，本书的主角 Python 诞生。发明者 Guido van Rossum。有打油诗赞到：Python 诞生，天降大任，开源开放，简洁优雅，独步天下，人工智能，“唯我不败”。请牢记这个值得纪念的年份和“仁慈的独裁者”（BDFL）。
- 1993 年，Ruby 诞生。发明者松本行弘。
- 1995 年，Java 诞生。发明者 James Gosling。
- 1995 年，JavaScript 诞生。发明者 Brendan Eich。注意，JavaScript 与 Java 在名字上和语法上虽然相似，但它们是两种完全不同的编程语言。
- 1995 年，PHP 诞生。发明者 Rasmus Lerdorf。
- 2001 年，C#诞生。发明者 Microsoft 公司。
- 2009 年，Go 诞生。发明者 Robert Griesemer、Rob Pike、Ken Thompson。
- 2011 年，Rust 诞生。发明者 Graydon Hoare。
- 2014 年，Swift 诞生。发明者 Chris Lattner。

图 1-1-2 是一些编程语言的拟人化。

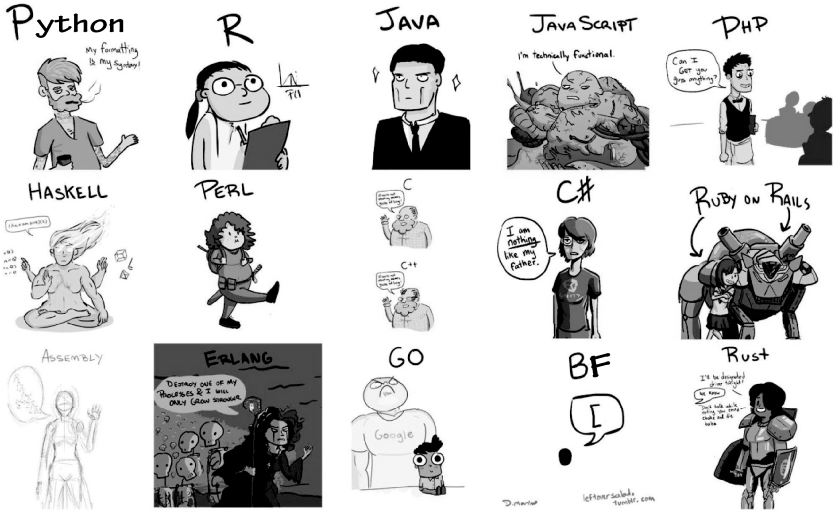


图 1-1-2　如果编程语言是人

（源自 http://kokizzu.blogspot.com/2017/01/if-programming-language-were-humans.html
以一种娱乐的心态看看编程语言，让枯燥的编程工作也变得愉悦）

本“编程语言极简史”就停止在了 2014 年，但是这并不意味着以后没有新的语言出现。还有很多语言没有被写在上述列表中，并不是它不重要或者没用途，而是使用了很“世俗”的观点选择了所谓的“主流语言”罢了。事实上，每种编程语言都有其存在的合理性，也有其应用的领域。

有些机构还会给出“编程语言排行榜”。或许每个人对这种排行榜有不同的解读，并不意味着排名靠后的是“劣等”语言。学习者不能将排行榜作为选择学习某种语言的依据。

那么，根据什么来选择学习某种编程语言呢？

本书作者提供如下参考：依据一，项目需要；依据二，时代发展需要。

依据一就不需要阐述了。依据二貌似有点“空泛”，事实上静心思考，就能理解。如今是什么时代？可能有各种回答方式，从靠近编程的角度来看，可以用“人工智能”时代来概括。

问：在“人工智能”时代，程序开发工作是否重要？

答：当然重要，虽然有媒体热炒“机器人替代程序员”，但“机器人”的程序是谁写的？追根溯源都是要人来做，“机器人”的智能还要靠“人工”。

问：学什么语言能参与这项工作？

答：Python。因为目前它是人工智能领域应用最多的语言。

决定了，学 Python。

“历史是过去的现实，现实是未来的历史”。编程语言的发展史也紧扣社会的发展。如果读者把“编程语言极简史”与相应的社会经济发展状况对应，更能理解如何选择学习某种语言了。

编程语言除了跟时代相关，其实还有“高低”之分，但无“贵贱”之别。

1.2 编程语言分类

在 1.1 节的“编程语言极简史”中列出的语言都是所谓的“高级语言”。这仅仅是编程语言的一类，并且不是“历史悠久”的那一族，也不是计算机能够直接识别和运行的。面向计算机的语言是“史前时代”就已经产生的“机器语言”和“汇编语言”。

1.2.1 机器语言

机器语言（Machine Language）是用二进制代码表示的计算机能够直接识别和执行的机器指令集合。

如果读者学习过有关计算机硬件知识，就知道计算机内部是由集成电路（Integrated Circuit，简称 IC）组成的，包括 CPU 和内存等。IC 有很多引脚（见图 1-2-1），只有直流电压 0V 或 5V 两个状态。也就是说，IC 的一个引脚只能表示两个状态。

IC 的这种特性正好与二进制相对应。计算机就将一系列的二进制数字转变为对应电压，从而使计算机的电子器件受到驱动，完成指定运算（见图 1-2-2）。

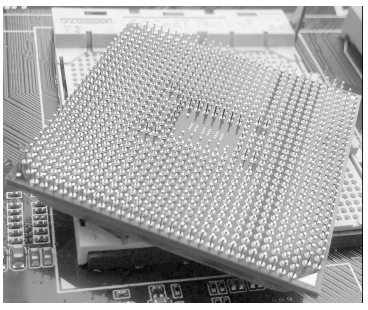


图 1-2-1　CPU 的引脚

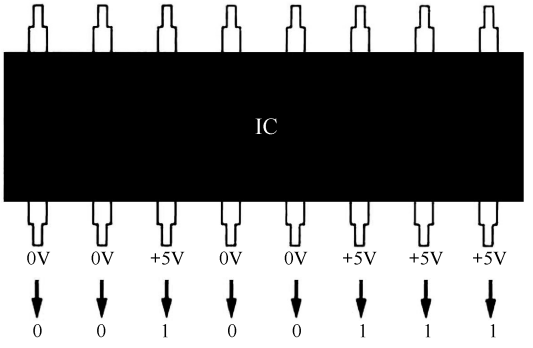


图 1-2-2　IC 的引脚和二进制

“史前时代”的程序员不得不使用机器语言来工作。他们将用 0、1 编写的程序打在纸带或卡片上，1 打孔，0 不打孔，再将程序通过纸带机或卡片机输入到计算机中进行运算。

这是一件多么富有挑战性的事情。

比如，“Hello World”，如果用机器语言表示，即二进制代码，应该是这样的：

```
01001000 01100101 01101100 01101100 01101111 00100000 01010111 01101111 01110010
01101100 01100100
```

如果写成了下面这样，打印的就不是“Hello World”了。

```
01001000 01100101 01101100 01101100 01101111 00110000 01010111 01101111 01110010
01101100 01100100
```

读者是否可以找出错误来？

显然，机器语言对于人“不友好”。用机器语言写程序，或许可以献上敬意，但不值得效仿。

另外，因为机器语言是计算机的设计生产者通过硬件结构赋予计算机的操作功能，所以不同型号计算机的机器语言会有所差别。除了少数专业人员，绝大多数编程者不需要学习机器语言。

所以，对人“友好”的语言应运而生。

1.2.2 汇编语言

汇编语言（Assembly Language）是二进制代码的文本形式，即使用便于记忆的书写格式表达机器语言的指令。

图 1-2-3 所示的汇编语言示例就是在 64 位 Linux 操作系统上运行的。

```
; ------------------------------------------------------------------------------
; Writes "Hello, World" to the console using only system calls. Runs on 64-bit Linux only.
; To assemble and run:
;
;     nasm -felf64 hello.asm && ld hello.o && ./a.out
; ------------------------------------------------------------------------------

          global    _start

          section   .text
_start:   mov       rax, 1                  ; system call for write
          mov       rdi, 1                  ; file handle 1 is stdout
          mov       rsi, message            ; address of string to output
          mov       rdx, 13                 ; number of bytes
          syscall                           ; invoke operating system to do the write
          mov       rax, 60                 ; system call for exit
          xor       rdi, rdi                ; exit code 0
          syscall                           ; invoke operating system to exit

          section   .data
message:  db        "Hello, World", 10      ; note the newline at the end
```

这里是对左边指令的说明
这是给人看的

图 1-2-3 汇编语言示例

看不懂图 1-2-3 中的代码也没有关系。在此只是请读者看一看汇编语言的基本“模样”，并非要求理解它。

每种汇编语言专用于某种计算机系统，不能在不同系统之间移植。

汇编语言相对于机器语言而言，已经是人可读、可编写的一种编程语言了。但它还非常靠近机器语言，用汇编语言“告诉”计算机干什么和计算机所干的之间（几乎）是一一对应的，即汇编语言的一条指令对应着一条机器指令。所以，汇编语言依然属于“低级语言”。

尽管如此，汇编语言现在依然有用武之地，因为它有自身的特点，比如目标程序占用内存少、运行效率高等——当然，这些优点的代价就是开发效率低。在某些特定任务中，还是少不了汇编语言的。

但，本书不以此为内容，而是介绍“高级语言”。

1.2.3 高级语言

“高级语言”（High-level Programming Language）是面向人的语言——It is for Humans（见

图 1-2-4）。

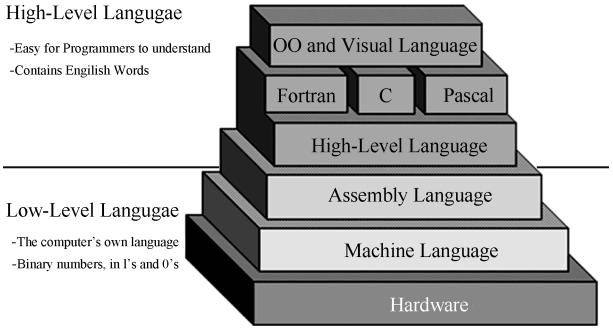


图 1-2-4　高级语言和低级语言

（源自 http://justcode.me/assembly/introduction-assembly-language-examples/）

之所以冠以“高级”之名，是因为这种语言使用了大量的英语单词，对开发者而言，更容易理解。最重要的是，高级语言摆脱了“硬件的拖累”，不需要与机器语言的指令对应，借助操作系统实现了对硬件的抽象。即使开发者“对硬件一窍不通”，也能利用高级语言开发程序。

图 1-2-5 中展示了一段 Python 程序，其作用是创建一个名为 hello.txt 的文件，并向这个文件中写入了“Hello World”。这个操作如果直接面对硬件，需要向磁盘的 I/O 指定扇区位置写入数据。但在这段代码中，丝毫没有扇区的影子。这是因为 Python 语言借助操作系统，将这个流程抽象为创建文件的 open 函数和写入文件的 write 函数，并且使用 with 实现自动关闭的功能。

```
>>> with open("hello.txt", "w") as f:
...     f.write("Hello World")
...
11
```

创建文件

向文件中写入此内容

图 1-2-5　创建并写入文件内容

所以，此语言才“高级”（见图 1-2-6）。

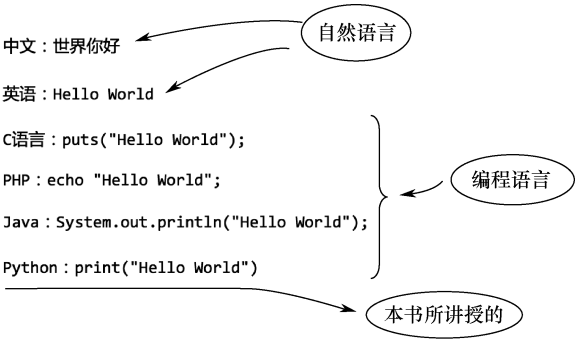


图 1-2-6　自然语言和不同的高级语言比较

自 20 世纪 50 年代的 Fortran 语言开始，人类已经发明了好多种高级语言，它们各有千秋，就如同人类自然语言五花八门那样。每种高级语言都是针对某种“编程”行为，兼顾开发者的知识结构，以及程序的应用场景，制订了特有的语法规则。

虽然高级语言的语法规则各异，但它们都必须使用“语句”进行表达。高级语言中的“语

句”是对计算机指令的抽象（见图 1-2-5），不与机器语言一一对应，“是给人看的”，计算机不能直接识别和执行。比如“a = b + 1”，要让计算机能够执行，必须“翻译”为机器语言的三条指令（见图 1-2-7）。

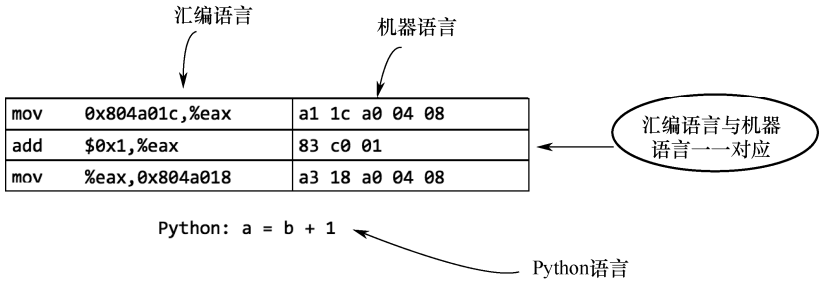


图 1-2-7　汇编语言与高级语言比较

计算机是如何“翻译”的？

1.3　程序简介

计算机程序（Computer Program），或称为程序（Program），是一组指示计算机或其他具有信息处理能力的设备的每一步操作的指令集合。通常，程序由某种编程语言编写而成。

如前所述，如果使用机器语言编写程序，那么计算机就能直接“认识”了。但是，编程者就痛苦了。

用汇编语言，对编程者而言，痛苦程度下降了，但是计算机“不认识”汇编语言，还需要将汇编语言所编写的程序翻译为机器语言程序，这个过程被称为“汇编”，用于执行此过程的程序被称为“汇编器”。

用汇编语言编写程序也不是快乐的事情，因为它与机器语言一样，同属低级语言。

现在，更多的程序是用高级语言开发的。

计算机更“不认识”高级语言了。所以，用高级语言所编写的程序同样要被“翻译”为机器语言程序，才能被计算机识别和执行。

1.3.1　程序“翻译”方式

程序的“翻译”方式有两种：编译和解释。

在说明这两种“翻译”方式之前，先定义如下专有术语。

源代码：用某种高级语言写的程序就称为“源代码”。

源文件：保存源代码的文件称为源文件。

本地代码：计算机（具体就是 CPU）能直接执行的机器语言的程序。用任何编程语言编写的源代码，最后都要被翻译为本地代码，否则 CPU 不能理解。

（1）编译

用编译器（complier，也是一种程序）将源代码全部翻译为本地代码的过程，就是“编译”。所谓编译器，则是执行这一过程的程序。

某些编程语言写的程序需要编译之后才能被执行，这类语言常被称为“编译型语言”，

如 C、C++、Pascal、Object-C、Swift、Rust 等。

（2）解释

有的程序不需要编译，在运行它的时候，直接用解释器（interpreter，也是一种程序）对源代码进行解释和执行。

同样，用于编写这类程序的编程语言被称为“解释型语言”，如 BASIC、PHP 等。

不论程序用哪种方式被翻译为本地代码，程序中的内容都需要按一定规则来编写，其中最重要的规则就是算法。

1.3.2 算法

算法，比计算机还要古老，虽然它现在常常被放在计算机或者软件专业来学习，事实上算法的历史可以上溯到早期文字出现的时候。

其实，读者应该对算法不陌生。比如，小学数学中的竖式加法（见图 1-3-1）就是一种算法。

编程序如同写文章，语句相当于文章中的句子，算法则与篇章结构类似。

算法是编写程序不可或缺的，而且普遍存在于编程过程中。或许因为它广泛存在，才使得定义它越发困难，所以迄今还没有一个严格的、大家公认的定义，但对它的基本含义还是有一些共识的。

算法（algorithm）是一系列解决问题的清晰指令。也就是说，对于符合一定规范的输入，程序能够在有限时间内获得所要求的输出（见图 1-3-2）。

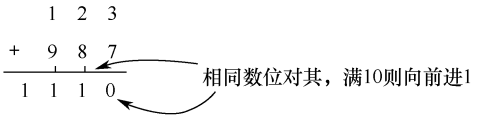


图 1-3-1 加法的竖式算法

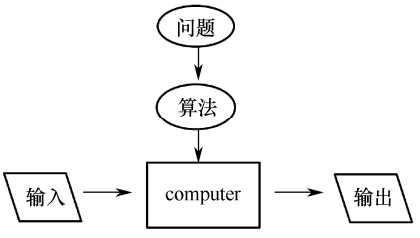


图 1-3-2 算法定义

作为“指令”集合的算法传给 computer，computer 必须理解此指令，并能按照指令要求进行操作。而 computer 在很早的时候并不是现在所说的计算机（从构词法就可以推断，可能是从事 compute 工作的人）。或者说，算法不是必须与编程绑定的。

在著名的《几何原本》（公元前 3 世纪欧几里得著）中记载了最大公约数的算法。其现代方式的表述如下。

假设两个不全为 0 的非负整数 m 和 n（$m>n$）的最大公约数记为 gcd(m, n)，其计算方法如下。

第一步：如果 n=0，返回 m 的值作为结果，同时结束计算；否则进入第二步。

第二步：m 除以 n，将余数赋给 r，即 $r = m \% n$（%是 Python 中计算余数的符号）。

第三步：令 $m = n$，$n = r$；然后返回第一步。

对于算法的特征，业界基本认同高德纳在他的著作《计算机程序设计艺术》一书中的归纳。

① 输入：一个算法必须有两个或以上输入量。

② 输出：一个算法应有一个或以上输出量，输出量是算法计算的结果。

③ 明确性：算法的描述必须无歧义，以保证算法的实际执行结果是精确地符合要求或期望，通常要求实际执行结果是确定的。

④ 有限性：依据图灵的定义，一个算法是能够被任何图灵完备系统模拟的一串运算，而图灵机只有有限个状态、有限个输入符号和有限个转移函数（指令）。一些定义更规定，算法必须在有限个步骤内完成任务。

⑤ 有效性：一个算法的任何计算步骤都可以被分解为基本可执行的操作，每个操作都能够在有限的时间内完成。

算法对于程序开发而言，其重要性是不言而喻的。但因为本书的定位不是算法专门教程，所以，建议读者可以通过阅读算法类的专门书籍来学习有关算法知识。

本书的重点还是讲述如何用 Python 语言编写程序，其间会自然而然地用到算法，不过读者不必为此担心。

1.3.3 Hello World

解决同一个问题的程序，通常可以用不同编程语言实现，但是受限于客观条件，最终会选择某种语言。在某些项目中也会使用多种语言，不同语言所编写的程序之间通过“接口”实现数据和业务互通。

因为每种语言都有自己的语法规则，所以程序样式各异。图 1-3-3 展示了利用几种高级语言打印“Hello World”的程序。

```
#include <stdio.h>
int main(void) {
    printf("Hello World!\n");
    return 0;
}
```

这是C语言

```
public class HelloWorld {
    public static void main(String[] args){
        System.out.println("Hello World!");
    }
}
```

Java

```
using System;
class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
```

C#，是不是跟Java有点类似
C#是站在Java肩膀上

```
print("Hello World")
```

这是Python
就这么简洁，简洁到了令人惊讶的程度

图 1-3-3　不同高级语言输出“Hello World”

不要根据代码行数评价语言的“好坏”，尽管 Python 的“Hello World”最简短，也不意味着它就具有某种“可炫耀的优越”。因为不同类别的语言有不同的用武之地。

然而，Python 的简洁还是吸引人的。

1.4 Python 概要

作为高级语言之一的 Python，日益受开发者瞩目。特别是随着大数据、人工智能等相关技术的发展，Python 几乎成为了这种“高科技”领域的必学语言。

这颗新星是如何升起的？它有什么特征？

1.4.1 发展历程

Python 语言是“人工语言”，就有一个创造它的人——Python 之父，此人名为吉多 • 范罗苏姆（Guido van Rossum）（见图 1-4-1）。

图 1-4-1 Guido van Rossum
（源自 https://zh.wikipedia.org/wiki/Python）

向此人献上崇拜和敬意，非常感谢他创造了 Python，世界上又多了一个可用的高级语言——并且那么好用。

关于 Python 的诞生，流传着这样一个故事（来自《维基百科》中文的“Python”词条，如图 1-4-2 所示）。

如果读者阅读《维基百科》中的英文词条（https://en.wikipedia.org/wiki/Python_(programming_language)），则没有这种春秋笔法了。

无论如何，Python 诞生了。

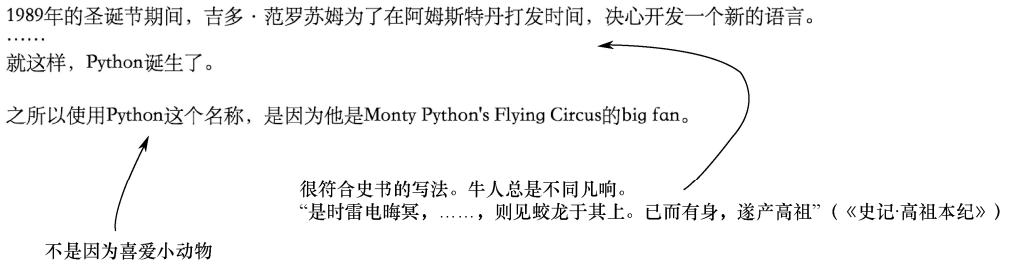


图 1-4-2 Python 诞生的故事节选

Python 自诞生以来，遵循着“开源、开放”的原则，得到了快速的发展和广泛的应用，包括一些大型公司或者大型项目（见图 1-4-3）。

目前，用 Python 语言可以做的事情已经很多了，包括但不限于以下所列：

- Web 开发。通常使用一些 Web 框架，如 Django、Flask 等。
- 网络爬虫。Python 对于各种网络协议的支持很完善，用之做网络爬虫非常便捷。
- GUI 开发。Python 中有非常好的支持桌面软件开发的工具，如 Tkinter、wxPtyhon 等。
- 数据分析和机器学习。NumPy、Pandas、SciPy、Scikit Learn 等工具让 Python 在数据分析和机器学习领域成为翘楚。
- 神经网络。TensorFlow 是 Google 公司推出的神经网络库，从开始就支持 Python API。此外，类似的库还有 PyTorch 等。

故，Python 值得拥有。

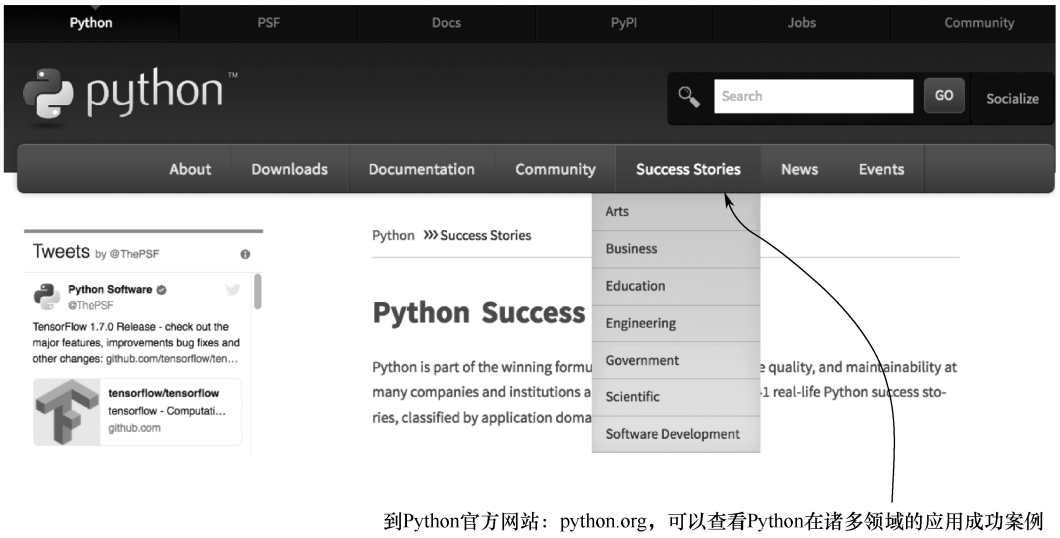


图 1-4-3　Python 官方网站上的案例

还因为，它简单易学。

1.4.2　从 Python 开始

本书是专门讲授 Python 的教程，当然要“从 Python 开始”了。但，这还不是充足的理由。

如果选择一个可以适应更多行业的人学习的编程语言，那么非 Python 莫属。因为它“简单易学”——初步感受请见图 1-3-3。据悉，国内外都有很多中小学生开始学习 Python 了。

可能有人担心，“简单”的 Python 会不会“简陋”呢？

绝不会！

Google 等国际大公司的大项目已经做出了回答。请参照图 1-4-3 所示的 Python 官方网站的诸多案例介绍。

所以，读者不论是否有计算机软件开发基础，都可以“从 Python 开始”学习编程。

另外，Python 语言因为“开源、开放”而得到了众多支持，形成了完善的“生态环境”。在这个环境中有众多“轮子”——这是一种比喻说法，意思是有很多开发支持工具，提高软件开发效率。例如，PyPI（Python Package Index）就是开发者发布各种工具模块的地方（见图 1-4-4）。

Python 还有一个绰号：“胶水语言”，因为它能轻松地实现与其他高级语言对接。例如在 Google 内部，据说就有很多项目使用 Python 调用 C++的程序。

Python 除了具有上述特征，以下各项也是它的特征，同样可以作为“从 Python 开始”学编程的重要理由。

- ❖ Python 是多范式编程语言，全面支持面向对象编程和结构化编程，也能实现函数式编程。
- ❖ Python 以“优雅、明确、简单”为设计哲学，倡导“最好只有一种方法做一件事”的编程思想，拒绝花哨的语法，使用明确且没有歧义的语法。

最后的结论是：马上准备开始学 Python。

图 1-4-4　PyPI 首页

练习和编程 1

1．计算机能够识别和执行的语言是什么？

2．说明“编译”和“解释”的含义。

3．浏览 Python 官方网站，根据网站提供的信息，并运用搜索引擎，写一篇关于 Python 特点和应用领域的综述短文。

4．浏览网站 stackoverflow.com，了解网站的功能，并且在网站注册账号。

5．浏览网站 github.com，借助搜索引擎，完成如下操作：

（1）通过搜索引擎了解“源码管理”和 git。

（2）在本地计算机上安装 git。

（3）在 github.com 上注册个人账号，并建立个人代码仓库。

（4）在 github.com 上创建个人博客，并将第 3 题中的短文发布到个人博客（提示：运用搜索引擎，查找能够在 github.com 上应用的开源代码以及代码的部署方法）。