

4

Fundamentals of Grammar in Python

Python 语法，边学边用

吸取英语学习失败的教训，不能死磕语法



当你建造空中楼阁时，它不会倒塌；空中楼阁本应属于高处。现在，撸起袖子把地基夯实。

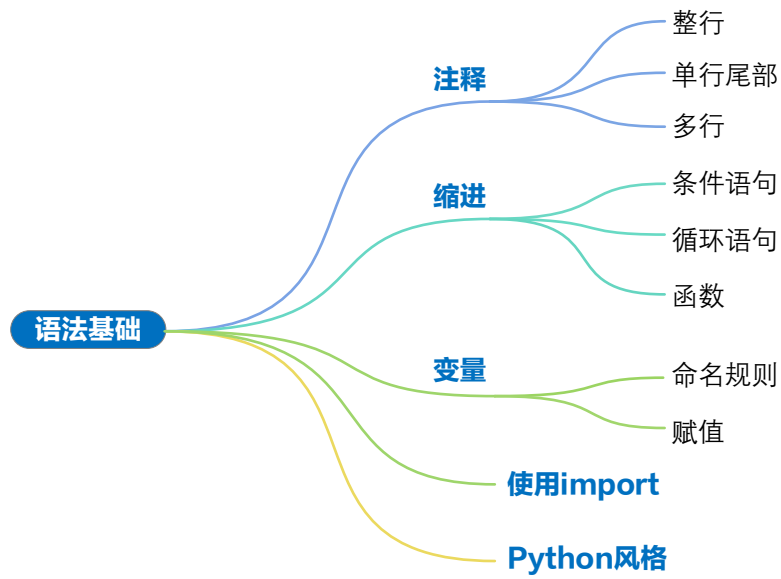
If you have built castles in the air, your work need not be lost; that is where they should be. Now put the foundations under them.

—— 亨利·戴维·梭罗 (Henry David Thoreau) | 作家、诗人 | 1817 ~ 1862



- ◀ float() Python 内置函数，将指定的参数转换为浮点数类型，如果无法转换则会引发异常
- ◀ for ... in ... Python 循环结构，用于迭代遍历一个可迭代对象中的元素，每次迭代时执行相应的代码块
- ◀ from numpy import * 从 NumPy 库中导入了所有函数和对象，使得我们可以直接使用 NumPy 的所有功能，无需使用前缀 "numpy." 来调用。不建议使用这种方法
- ◀ from numpy import array 从 NumPy 库中导入了 array 函数，使得我们可以直接使用 array 函数而无需使用 "numpy.array" 来创建数组
- ◀ if ... elif ... else Python 条件语句，用于根据多个条件之间的关系执行不同的代码块，如果前面的条件不满足则逐个检查后续的条件
- ◀ if ... else ... Python 条件语句，用于在满足 if 条件时执行一个代码块，否则执行另一个 else 代码块
- ◀ import numpy as np 将 NumPy 库导入为别名 np，使得我们可以使用 np 来调用 NumPy 的函数和方法
- ◀ import numpy 将 NumPy 库导入到当前的 Python 环境中，调用时使用完整的 numpy 作为前缀
- ◀ input() Python 内置函数，用于从用户处接收输入
- ◀ int() Python 内置函数，用于将指定的参数转换为整数类型，如果无法转换则会引发异常
- ◀ list() Python 内置函数，将元组、字符串等等转换为列表
- ◀ numpy.arange() 创建一个包含给定范围内等间隔的数值的数组
- ◀ numpy.array() 输入数据转换为 NumPy 数组，从而方便进行数值计算和数组操作
- ◀ numpy.random.rand() 在 [0, 1) 区间，即 0（包含）到 1（不包含）之间，内生成特定形状满足连续均匀的随机数
- ◀ print() Python 内置函数，将指定的内容输出到控制台或终端窗口，方便用户查看程序的运行结果或调试信息
- ◀ range() Python 内置函数，用于生成一个整数序列，可用于循环和迭代操作
- ◀ set() Python 内置函数，创建一个无序且不重复元素的集合，可用于去除重复元素或进行集合运算
- ◀ str() Python 内置函数，用于将指定的参数转换为字符串类型





本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：jiang.visualize.ml@gmail.com

4.1 Python 也有语法?

和汉语、英语、法语等人类语言一样，Python 也是语言。只不过 Python 是编程语言，是人和计算机交互语言。凡是语言就有语法——一套约定成俗交流规则。

有了类似 ChatGPT 这样的自然语言处理工具，人类的确可以直接使用人类语言和机器交流。但是，考虑到 ChatGPT 也是用 Python 开发而成，Python 不过是退隐幕后罢了。

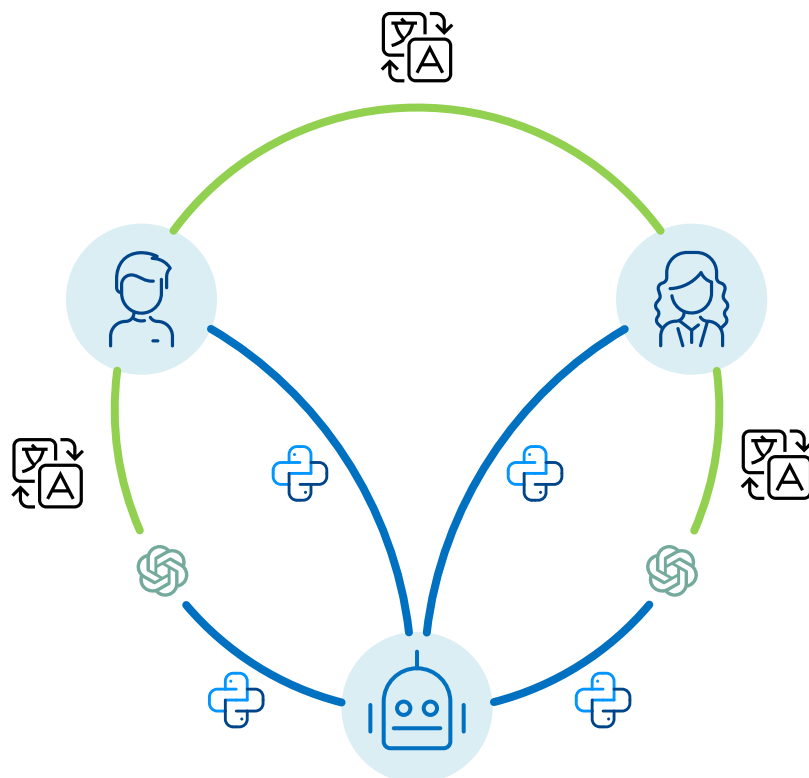


图 1. Python 也是语言

Python 语法使用数量极少的英文词汇，而且都是很基本的词汇。

Python 和英语都有一些关键词，例如 Python 中的 `if`、`else`、`for`、`while` 等关键词，和英语中的 `if`、`else`、`for`、`while` 等单词是一样的。

Python 和英语都有语法结构，例如 Python 中的 `if` 语句和英语中的条件句都是用来表示条件语句的结构。

Python 和英语都有一些语法规则，例如 Python 中的缩进规则和英语中的句子结构规则都是用来规范语法的。

Python 语法相对来说比英语语法容易掌握，因为 Python 语法的规则和规范性更强。

表 1 总结 Python 中常用英文关键词，并不需要大家背诵，浏览一遍就好。本书后续会介绍常用关键词。

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：jiang.visualize.ml@gmail.com

⚠ 请大家注意大小写，特别是 True、False、None 需要首字母大写。

表 1. Python 中常用英语关键词

英语	汉语	介绍
and	和	逻辑操作符，要求两个条件都满足时才返回 True
argument	参数	print('Hey you!') 中的 'Hey you' 是函数 print() 的输入参数
as	作为	用于别名，可以给模块、函数或类指定另一个名称，比如 import numpy as np
assert	断言	用于测试代码的正确性，如果条件不成立则会引发异常
boolean	布尔值	True 和 False 是两个布尔值
break	中断	用于跳出循环语句
class	类	定义一个类，包含属性和方法
complex	复数	8 + 8j 是一个复数
condition	条件	if x > 0: 是一个条件语句
continue	继续	用于跳过当前循环的剩余部分，继续执行下一次循环
def	定义	定义一个函数
del	删除	用于删除变量或对象
dictionary	字典	{'name': 'James', 'age': 18} 是一个字典
elif	否则如果	用于在 if 语句中添加多个条件判断
else	否则	用于 if 语句中，当所有条件都不满足时执行
except	除外	用于捕获异常。
False	假	表示布尔值为假。
finally	最后	用于定义无论是否发生异常都要执行的代码块
float	浮点数	3.14 是一个浮点数
for	循环	用于迭代遍历序列、集合或其他可迭代对象
from	来自	用于从模块中导入特定函数、类或变量，比如 from numpy import random
function	函数	print() 是一个函数
global	全局	用于在函数中引用全局变量
if	如果	用于条件判断，比如 if x > 0:
import	导入	用于导入模块，比如 import numpy
in	在	用于检查元素是否存在于序列、集合或其他可迭代对象中
integer	整数	88 是一个整数
is	是	用于检查两个对象是否相同
lambda	匿名	定义一个匿名函数
list	列表	[1, 2, 3] 是一个列表
loop	循环	for i in range(10): 是一个循环语句
module	模块	import math 导入了 Python 的 math 模块
None	空	表示一个空值或缺少值
not	非	逻辑操作符，将 True 变为 False，将 False 变为 True
object	对象	my_object = MyClass() 中 my_object 是一个 MyClass 类的对象
or	或	逻辑操作符，只要一个条件满足就返回 True
package	包	import numpy 导入了 Python 的 numpy 包
pass	跳过	用于占位符，不执行任何操作
raise	引发异常	用于引发异常，比如，raise ValueError("Invalid value.")

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：jiang.visualize.ml@gmail.com

return	返回	用于从函数中返回值
set	集合	{1, 2, 3} 是一个集合
statement	语句	x = 88.8 是一个赋值语句
string	字符串	Hey you! 是一个字符串
True	真	表示布尔值为真
try	尝试	用于包含可能引发异常的代码块，比如 try: except ValueError:
tuple	元组	(1, 2, 3) 是一个元组
variable	变量	x = 8 中 x 是一个变量
while	当	用于创建循环，只要条件为真就重复执行代码块
with	使用	用于自动管理资源，例如文件句柄或数据库连接
yield	产生	用于生成器函数，暂停函数执行并返回一个值

Python vs C 语言

Python 是一种高级的面向对象编程语言。C 语言是一种编译型语言，非常适合编写底层的系统软件，例如操作系统、编译器和设备驱动程序等。C 语言的优势在于其对硬件和操作系统的底层控制，而这也是 Python 所缺乏的。Python 在处理复杂的数据结构和算法时，通常比 C 语言慢得多。

Python 的优势主要是其强大的第三方库和工具生态系统，使得 Python 可以用于更高层次的机器控制和自动化任务，例如数据处理、机器学习和自然语言处理等。



什么是面向对象编程语言？什么是编译型语言？

面向对象编程语言是一种编程范式，它将现实世界中的概念和模型转化为计算机程序中的类和对象。面向对象编程中的核心概念包括封装、继承和多态性。

编译型语言是指需要先通过编译器将源代码转换成可执行代码的编程语言。在编译过程中，编译器会对代码进行语法分析、词法分析、语义分析、优化等操作，将源代码转换成二进制可执行文件。编译型语言的执行速度更快，但开发效率较低，因为需要编写和编译源代码。

学习板块

本书有关 Python 语法主要包括以下几个板块：

- ▶ 基础语法（本章）：注释、缩进、变量、包、代码风格等。
- ▶ 数据类型（第 5 章）：数字、字符串、列表、元组、字典等。
- ▶ 运算符（第 6 章）：算术运算符、比较运算符、逻辑运算符、位运算符等。
- ▶ 控制结构（第 7 章）：条件语句、循环语句、异常处理语句等。
- ▶ 函数和模块（第 8 章）：函数和模块的定义和使用。
- ▶ 面向对象编程（第 9 章）：定义类、对象、方法、属性等。

4.2 注释：不被执行，却很重要

Python **注释** (comment) 就是在写 Python 代码时，为了方便自己和别人理解代码，添加的文字说明。这些文字说明不会被 Python **解释器** (interpreter) 执行，只是为了让代码更易读懂和更易维护。

在 Python 代码中，我们可以使用 **#** (hash, hashtag, hashmark) 符号来添加注释。

当 Python 解释器读取代码时，如果遇到 **#** 符号，它就会将 **#** 所在行后面的内容视为注释，而不是代码的一部分。

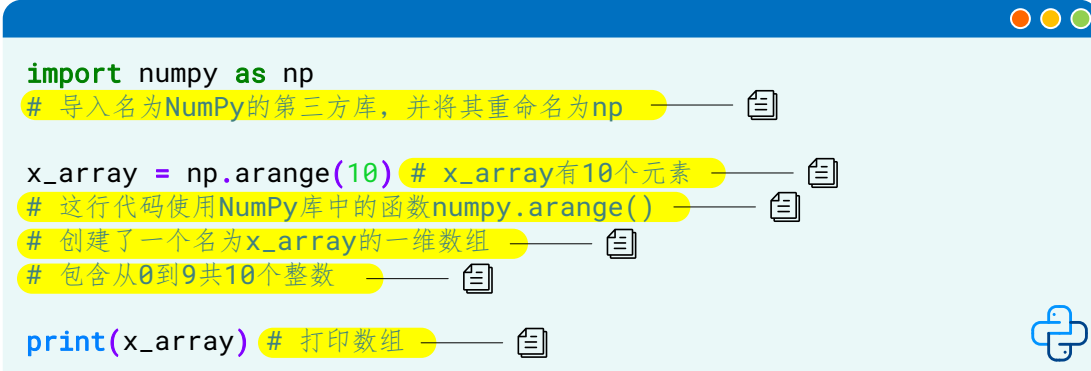
⚠ 注意，**#** 后面的字符开始直到该行的结尾都被认为是注释。

⚠ 注意，好记性不如烂笔头！对于初学者，特别是零基础读者，逐行注释是快速学习掌握编程的小窍门！关键语句记不住的话，在不同位置反复注释。

#注释：整行、单行尾部

如代码 1 所示，可以把注释（图中高亮部分）看作是给代码添加的“贴纸”，用来解释代码的用途、原理、变量的含义等等。机器遇到图中高亮部分文字就自然跳过。

代码 1 展示了两种注释：1) 整行注释；2) 单行尾部注释。



```

a import numpy as np
   # 导入名为NumPy的第三方库，并将其重命名为np

b x_array = np.arange(10) # x_array有10个元素
   # 这行代码使用NumPy库中的函数numpy.arange()
   # 创建了一个名为x_array的一维数组
   # 包含从0到9共10个整数

c print(x_array) # 打印数组

```

代码 1. 举例说明 Python 代码中的注释 | Bk1_Ch04_01.ipynb

下面简单讲解代码 1。

a 将 **numpy**（正式名称为 **NumPy**）导入到当前 Python 环境中，并给 **numpy** 一个别名 **np**。这样我们可以使用 **np** 来调用 **NumPy** 的函数和方法。这是一种在 Python 中较为常用的导入第三方库的方法。本章后文还会介绍其他几种导入库的方法。

b **np.arange()** 调用 **numpy**（别名 **np**）库中的 **arange()** 函数。如图 2 所示，**np.arange(10)** 产生 0 ~ 9 这 10 个整数构成的数组，**array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])**，并赋值给变量 **x_array**。

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：jiang.visualize.ml@gmail.com

➔ 本书第 13 到 18 章专门介绍 NumPy。

③ 利用 `print()` 函数打印变量 `x_array` 中保存的 `array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])`。

请大家在 JupyterLab 中练习代码 1。

```
numpy.arange(10)
```




图 2. 一维 NumPy 数组

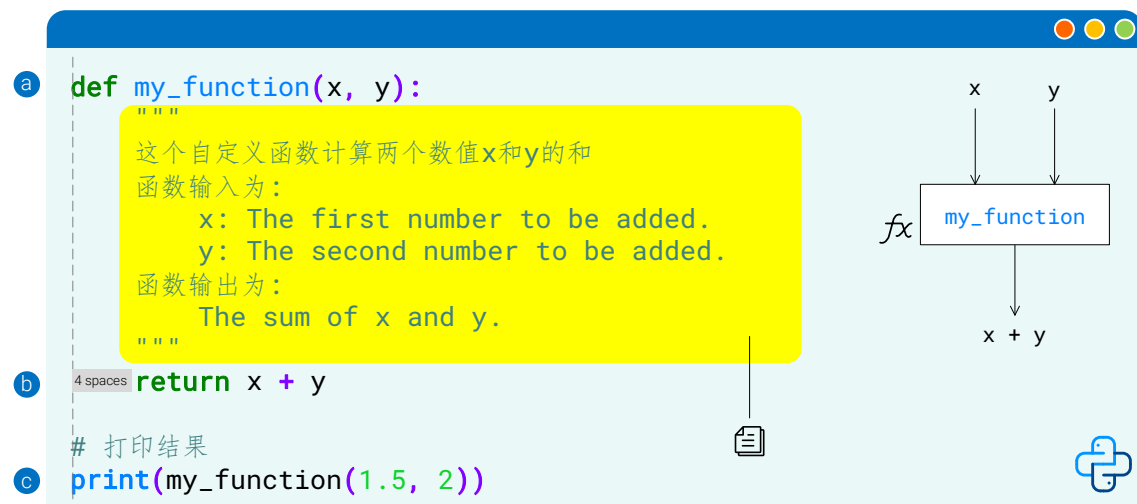
在 Jupyter Notebook 中，markdown 的功能和 comment 显然不同。Markdown 相当于笔记，可以是标题、文本段落、列表、图片、链接等等。而 comment 是在代码块中添加对具体代码的说明和解释。

⚠ 再次提醒，JupyterLab 中 comment 和 uncomment 默认快捷键为 `ctrl + /`。

'''或'''注释：多行

此外，我们还可以用成对三个引号（'''或'''）来添加多行注释。

比如，要在 Python 代码中添加一段多行注释，来描述一个函数的功能和用法，那么可以使用三个引号来实现。代码 2 是一个例子。



代码 2. 用三个引号来添加多行注释 | Bk1_Ch04_01.ipynb

代码 2 中，③ 利用 `def` 定义了一个名为“`my_function`”的函数，然后使用三个引号来添加多行注释。“`my_function`”是个自定义函数，括号（）内有两个输入 `x` 和 `y`。

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：jiang.visualize.ml@gmail.com

在 Python 中，自定义函数是一种将一段可重用代码封装起来的方法。大家可能会好奇，Python 各种库已经提供大量函数，我们为什么还需要自定义函数？

首先，除了通用函数之外，我们需要各种满足个人定制化要的函数。自定义函数让代码模块化，便于管理和维护。

一旦创建了一个函数，我们可以在不同的地方多次调用它，而不必重复编写相同的代码。将部分代码封装在自定义函数中，还可以提高代码的可读性，让代码更简洁，方便调试，降低错误。



本书第 8 章将专门讲解自定义函数。

▲ 注意，**a** 这句以**冒号** (colon) : 结束。一般情况下，Python 代码语句不需要用**分号** (semi-colon) ; 结束。但是，分号 ; 可以让我们在一行中写几句（短）代码。

表 2 列出 Python 中使用冒号的几种常见情况，本书后文都会涉及到。

b 用 return 返回自定义函数的输出——x 和 y 的和。

b 一句在 return 之前有 4 个空格，叫做**缩进** (indentation)。本章后文将专门介绍缩进的作用。

▲ 注意，中文输入法下的单、双引号都是“全角引号”，Python 解释器会抛出语法错误。在 Python 中，只有半角引号 (') 和双半角引号 (") 才可以用来定义字符串，而全角引号则不能用于字符串的定义。此外，使用圆括号、中括号等符号时也要注意全角、半角问题，避免语法错误。

c 利用自定义函数“my_function”计算 1.5 和 2 之和，然后用 print() 打印结果。

在上面的例子中，我们使用了三个引号来包裹函数的注释文字，这个注释可以跨越多行，并且被 Python 解释器忽略掉，不会被当作代码执行。这样，其他程序员在阅读我们的代码时，就可以清晰地了解这个函数的作用、输入和输出参数、以及函数的返回值。

前文提过，为了保证字母、数字、符号、空格等显示时宽度一致，本书正文和图片中示例代码采用的字体为 Roboto Mono Light。

表 2. Python 使用冒号的常见情况

情况	语法
索引和切片	<pre>string_obj[start:end:step_size] # 字符串 list_obj[start:end:step_size] # 列表 tuple_obj[start:end:step_size] # 元组 numpy_array[start:end:step_size] # NumPy array</pre>
字典键值对	<pre>dict_obj{key:value} # 字典</pre>
条件语句	<pre>if condition_1: # 代码块，注意缩进 elif condition_2: # 代码块，注意缩进 else: # 代码块，注意缩进</pre>
循环语句	<pre>for element in iterable: # 代码块，注意缩进 while condition: # 代码块，注意缩进</pre>

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：jiang.visualize.ml@gmail.com

定义函数	<code>def function_name(arguments):</code> # 代码块，注意缩进
lambda 函数	<code>lambda variables: expression</code>
定义类	<code>class ClassName:</code> # 代码块，注意缩进
异常处理	<code>try:</code> # 代码块，注意缩进 <code>except SomeException:</code> # 代码块，注意缩进 <code>finally:</code> # 代码块，注意缩进
上下文管理	<code>with context_manager:</code> # 代码块，注意缩进；第 35 和 36 章中使用 <code>streamlit</code> 库时会用到

4.3 缩进：四个空格，标识代码块

相信大家已经在代码 2 和表 2 发现了**缩进** (indentation)。

在 Python 中，缩进是非常重要的。缩进是指在代码行前面留出的**空格** (space) 或制表符 (tab →)，它们用于表示代码块的开始和结束。换句话说，缩进用于指示哪些代码行属于同一个代码块。

在其他编程语言中，通常使用花括号或关键字来表示代码块的开始和结束（比如 MATLAB 用 `end` 表示代码块结束）。但在 Python 中，使用缩进来代替。

⚠ 注意，在 Python 中，缩进的大小没有严格规定，一般情况下建议使用四个空格作为缩进，并不鼓励用制表符 `tab` 缩进。特别反对混用四个空格、`tab` 缩进。

Python 中常见的需要缩进的场合包括 `for` 循环，`while ...` 循环，`if ... else ...` 判断语句，函数定义以及类的定义等。同一缩进级别里的代码属于同一逻辑块。这些需要使用缩进的场合往往都是需要使用冒号 `:` 来表示下一行需要使用缩进。

⚠ 注意，如果缩进有误编译器会报错，报错内容为 `IndentationError: unindent does not match any outer indentation level`。

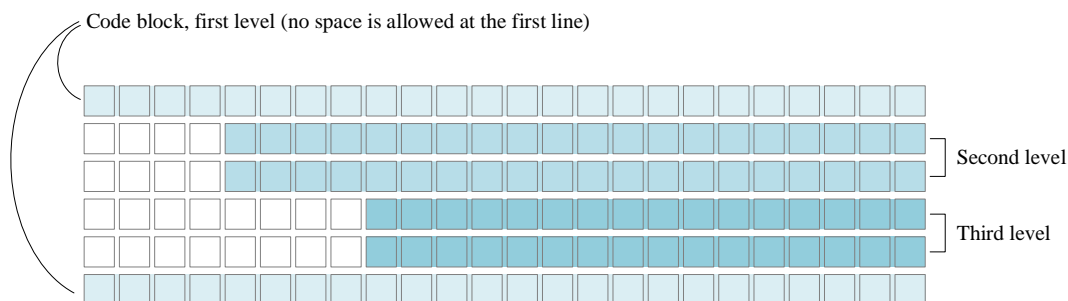


图 3. 缩进形成不同的代码级别

条件语句

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：jiang.visualize.ml@gmail.com

在 `if ... elif ... else ...` 语句中，它们所控制的代码块需要缩进，以表示它们属于条件语句。
代码 3 用 `if ... elif ... else ...` 语句判断输入数值正负。图 4 所示为代码的流程图。

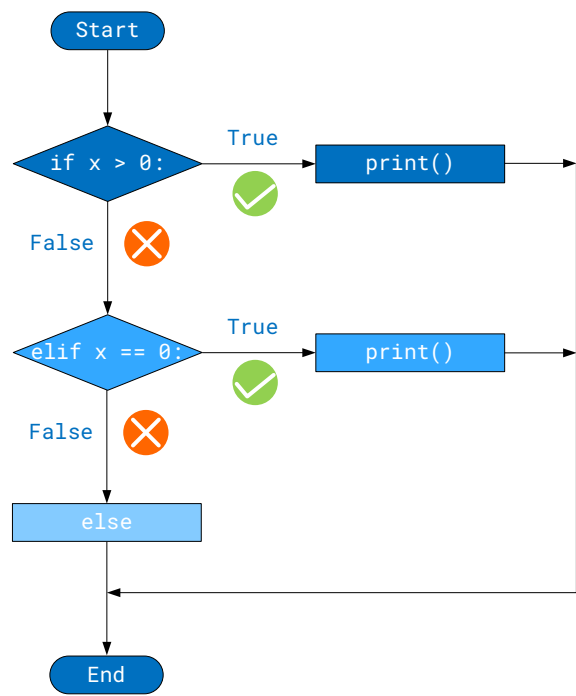


图 4. 条件判断流程图

编程时，**流程图**（flowchart）用于表示算法的逻辑结构和程序的执行流程。它可以帮助我们更好地理解代码的执行顺序、条件分支和循环结构。

从这个流程图结果来看，三个条件分支实际上将图 5 **实数轴**（number line）分为三个部分。

表 3. 流程图中最常用标识

名称	标识	解释
开始 (start)		流程的起始点
结束 (end, terminal)		流程的结束点
箭头 (flowline, arrowhead)		用来表达过程的次序
流程 (process)		表示一个操作、任务或活动的步骤
判断 (decision)		菱形，根据条件的不同，决定不同的流程走向

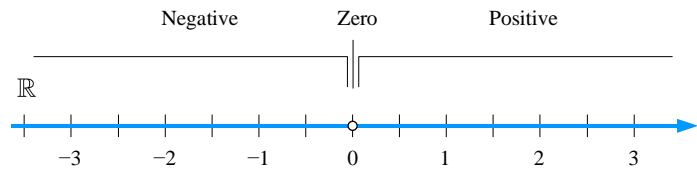


图 5. 将一根实数轴分为三个部分

下面介绍代码 3 中重要语句。

a 首先利用 Python 内置 `input()` 输入从用户获取数值输入，然后用 `float()` 将其转化为浮点数 (`float`) 并存在变量 `x` 中。简单来说，浮点数是在计算机中用以近似表示任意实数，基于科学计数法 (`scientific notation`)。

b 是条件语句的开始，它使用关键字 `if` 来引导一个条件的判断。在这里，条件是检查变量 `x` 是否大于零。大于号 `>` 用来判断。



本书第 6 章会专门讲解用于判断的运算。

如果这个条件为真 `True`，即 `x` 确实大于零，则下面缩进的代码块会被执行。

如图 6 (a) 所示，当输入为 8 时 (`x = 8`)，`x > 0` 结果为 `True`，则执行缩进中的代码块 `print("x is positive")`，打印消息。

如果 `x > 0` 判断结果为 `False`，则不执行缩进中代码，直接进入 **c**。



注意，在一个条件语句中，可以只有 `if` 分支，没有 `elif` 或 `else` 分支。

c 是条件语句中的 `elif` (`else if` 的缩写) 分支，在之前的条件 `if x > 0`：不满足时执行。这句用于检查变量 `x` 是否等于零，如果满足条件，则打印另一条消息。

如图 6 (b) 所示，当输入为 0 时 (`x = 0`)，`x == 0` 结果为 `True`，则执行缩进中的代码块 `print("x is zero")`，打印消息。两个相连等号 `==` 用来判断是否相等。

在一个条件语句中，可以没有 `elif`，也可以有若干 `elif`。

d 条件语句的 `else` 分支，用于处理在之前的条件不满足时的情况。之前条件包括 `if`，可能没有、也可能若干 `elif` 分支。

如图 6 (c) 所示，当输入为 -8 时，则执行 `else` 缩进中的代码块 `print("x is positive")`，打印消息。

e 这一句也在 `else` 分支中。如果 `x` 为负数，对 `x` 变号计算绝对值 (`absolute value`)，并赋值给 `abs_x`。

此外，还请大家注意 `if`、`elif`、`else` 最后需要以半角冒号 `:` 结束。这个冒号还在英文输入法下的半角冒号。



本书第 7 章将专门讲解几种常见控制结构。

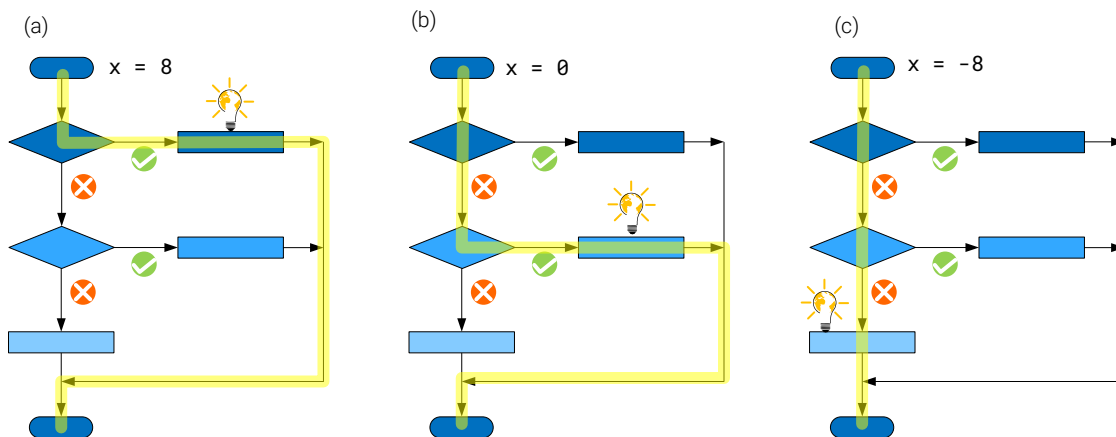


图 6. 三条不同路径

```

# 定义变量x，从用户输入中获取数值
x = float(input("请输入一个数值: "))

# 定义变量abs_x，用来存放绝对值
abs_x = x

# 如果x为正数
if x > 0:
    print("x is positive")

# 如果x为零
elif x == 0:
    print("x is zero")

# 如果x为负数
else:
    print("x is negative")

    # 计算负数绝对值
    abs_x = -x

print("该数值的绝对值为: ", abs_x)

```

代码 3. 条件语句中使用缩进 | Bk1_Ch04_02.ipynb

循环语句

在 `for`、`while` 等循环语句中，循环体内的代码块需要缩进，以表示它们属于循环语句。

在 Python 中，`for` 循环是一种迭代结构，用于遍历（iterate）可迭代对象（iterator），如列表、元组、字符串等，中的元素，执行特定的操作。

如代码 4 所示，^a 定义了一个字符串，赋值给变量 `x_string`。在 Python 中，**字符串** (string) 是一种数据类型，用于表示文本数据。字符串是由一系列字符组成的，可以包含字母、数字、符号以及空格等字符。定义字符串时，可以使用单引号 `' '` 或双引号 `" "` 包裹起来，两种方式是等效的。



本书第 5 章将专门介绍各种常见数据类型，比如字符串、列表、字典等等。

^b 在每次迭代时，`i_str` 会依次取得可迭代对象 `x_string` 中的元素，然后执行循环体内的 `print()` 操作。当可迭代对象中的所有元素都被遍历完毕，循环就会结束。

代码 4 的流程图如图 7 所示。



注意，`for ... in ...` 最后也需要以半角冒号 `:` 结束。

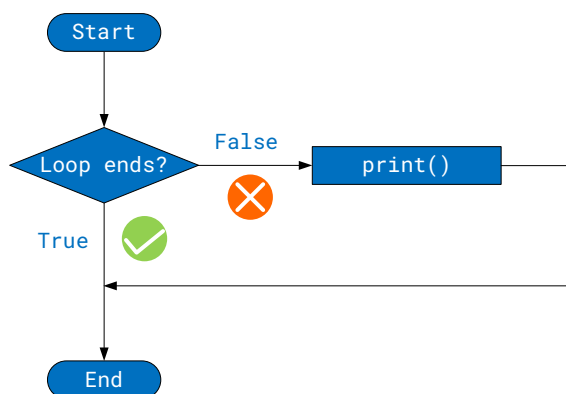


图 7. for 循环流程图

```

a x_string = 'Python is FUN!'
# 利用for循环打印每个字符
b for i_str in x_string:
    print(i_str)
  
```

代码 4. for 循环语句中使用缩进 | Bk1_Ch04_03.ipynb

4.4 变量：一个什么都能装的箱子

在 Python 中，**变量** (variable) 是用于存储数据值的标识符。本章开始到现在大家已经在不同代码中看到变量的影子。这些变量用于引用内存中的值，这些值可以是数字、字符串、列表、字典、函数等各种类型的数据。

如图 8 所示，简单来说，变量就是个“箱子”。

表 4 为 Python 中常见数据类型。下一章将专门介绍 Python 中各种常用数据类型。

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：jiang.visualize.ml@gmail.com

表 4. Python 中常见数据类型

数据类型	type()	特点	举例
数字 (number)	int float complex	包括整数、浮点数、复数等	x = 8 y = 88.8 z = 8 + 8j
字符串 (string)	str	一系列字符的序列	s = 'hello world'
列表 (list)	list	一组有序的元素，可以修改	a = [1, 2, 3, 4] b = ['apple', 'banana', 'orange']
元组 (tuple)	tuple	一组有序的元素，不能修改	c = (1, 2, 3, 4) d = ('apple', 'banana', 'orange')
集合 (set)	set	一组无序的元素，不允许重复	e = {1, 2, 3, 4} f = {'apple', 'banana', 'orange'}
字典 (dictionary)	dict	一组键-值对，键必须唯一	g = {'name': 'Tom', 'age': 18}
布尔 (boolean)	bool	代表 True 和 False 两个值	x = True y = False
None 类型	NoneType	代表空值或缺失值	z = None

在 Python 中，变量是动态类型的，这意味着我们可以在运行时为变量分配不同类型的值。不需要提前声明变量的类型，Python 会根据所赋予的值自动确定其类型。

也就是说，这个 Python 中的箱子什么都能装。在 Python 中，可以使用内置的 type() 函数来判定数据的类型。type() 函数返回一个表示对象类型的值。

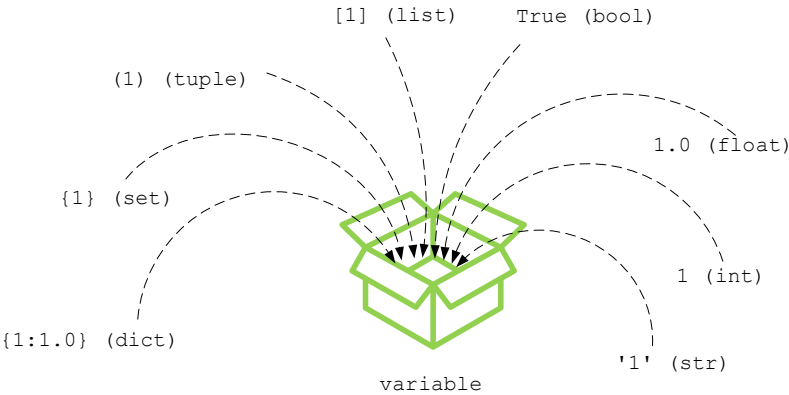


图 8. Python 变量就是个“箱子”，什么都能装



什么是动态类型语言？

动态类型语言是指在运行时可以自动判断变量的数据类型的编程语言。动态类型语言不需要在编写代码时显式地指定变量的数据类型，而是在程序运行时自动进行类型检查。

与之相对的是静态类型语言。静态类型语言中，每个变量都必须在声明时指定其数据类型，编译器会在编译时检查变量是否被正确使用。比如，C 语言是一种静态类型语言。`int x = 10; int y = 20;` `x` 和 `y` 都被声明为整数类型 (`int`)，编译器会在编译时检查它们是否被正确使用。

变量命名规则

Python 中的变量命名规则和建议如下。

- ▶ 变量名必须是一个合法的标识符，即由字母、数字和下划线组成，且不能以数字开头。例如，`x`、`my_var`、`var_1` 等都是合法的标识符。注意，变量名不能以数字开头，比如 `1_variable` 作为变量名不合法。
- ▶ 变量名区分大小写。例如，`my_var` 和 `My_var` 是不同的变量名。
- ▶ 变量名应该具有描述性，能够清晰地表达其所代表的内容。例如，`name` 可以代表人名，`age` 可以代表年龄等。
- ▶ 变量名应该尽量简洁明了，但不要过于简短或过于复杂。避免使用单个字母或缩写作为变量名，除非上下文明确。
- ▶ 变量名不应该与 Python 中的保留函数（关键字）重名，否则会导致语法错误。例如，不能使用 `if`、`else`、`while` 等关键字作为变量名。
- ▶ 在特定的上下文中，可以使用特定的命名约定。例如，类名应该使用驼峰命名法（`camelCase`），函数名和变量名应该使用下划线分隔法（`snake_case`）等。

有关 Python 内置函数用法，请参考。

<https://docs.python.org/zh-cn/3/library/functions.html>

驼峰、蛇形命名法

常见有两种变量命名法——`camel case`、`snake case`。下面简单比较两者。

- ▶ **驼峰命名法**（`camel case`）得名于其类似于骆驼背部的形状，其中单词之间的空格被移除，而每个单词首字母一般大写。在驼峰命名法中，通常有两种常见的变体：**小驼峰命名法**（`lower camel case`），比如 `firstName`、`totalAmount`，和**大驼峰命名法**（`upper camel case`），比如 `FirstNames`、`TotalAmount`。大驼峰命名法也叫**帕斯卡命名法**（`Pascal case`）。`Pascal case` 在 C# 中应用更多。
- ▶ **蛇形命名法**（`snake case`）以其类似于蛇的形状而得名，其中单词之间用下划线 _ 分隔，而且所有字母都是小写，例如 `first_name` 或 `total_amount`。

⚠ 注意，Python 社区变量名一般普遍采用蛇形命名法；Python **面向对象编程**（`Object-Oriented Programming, OOP`）中的**类**（`class`）定义一般采用驼峰命名法。而 Java 和 JavaScript 等语言则更常使用驼峰命名法。

变量赋值

本章读到这里，相信大家都已经清楚，我们可以使用等号 `=` 将一个值赋给一个变量。

可以同时给多个变量赋值，用逗号分隔每个变量，并使用等号将值分配给变量。例如：

```
x, y, z = 1, 2, 3 # x = 1; y = 2; z = 3
```

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：jiang.visualize.ml@gmail.com

可以使用链式赋值的方式给多个变量赋相同的值。例如：

```
x = y = z = 0
```

可以使用增量赋值的方式对变量进行递增或递减。例如：

```
x += 1 # 等价于 x = x + 1
```

4.5 使用 import 导入包

在代码 1 中，我们已经用过 import 导入 numpy 包。

在 Python 中，包是一组相关的模块和函数的集合，用于实现特定的功能或解决特定的问题。包通常由一个顶层目录和一些子目录和文件组成，其中包含了实现特定功能的模块和函数。

Python 中有很多常用的包，包括数据处理和可视化、机器学习和深度学习、网络编程、Web 开发等。其中，常用的可视化包包括 Matplotlib、Seaborn、Plotly 等，机器学习常用的包包括 NumPy、Pandas、Statsmodels、Scikit-learn、TensorFlow、Streamlit 等。

Matplotlib 是 Python 中最流行的绘图库之一，可用于创建各种类型的静态图形，如线图、散点图、柱状图、等高线图等。

Seaborn 是基于 Matplotlib 的高级绘图库，提供了更美观、更丰富的图形元素和绘图样式。

Plotly 是一款交互式绘图库，可用于创建各种类型的交互式图形，如散点图、热力图、面积图、气泡图等，支持数据可视化的各个方面，包括统计学可视化、科学可视化、金融可视化等。

NumPy 是 Python 中常用的数值计算库，提供了数组对象和各种数学函数，用于高效地进行数值计算和科学计算。

Pandas 是 Python 中常用的数据处理库，提供了高效的数据结构和数据分析工具，可用于数据清洗、数据处理和数据可视化。

Scikit-learn 是 Python 中常用的机器学习库，提供了各种常见的机器学习算法和模型，包括分类、回归、聚类、降维等。

TensorFlow 是谷歌开发的机器学习框架，提供了各种深度学习模型和算法，可用于构建神经网络、卷积神经网络、循环神经网络等深度学习模型。

Streamlit 可以通过简单的 Python 脚本快速构建交互式数据分析、机器学习应用程序。

本书前文介绍过如何安装、更新、删除某个具体包，下面我们聊一聊如何在 Python 中导入包。

导入包

下面以 NumPy 为例介绍几种常用的导入包的方式。

第一种，直接导入整个 NumPy 包：

```
import numpy
```

这种方式会将整个 NumPy 包导入到当前的命名空间中，需要使用完整的包名进行调用，例如：

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：jiang.visualize.ml@gmail.com

```
a = numpy.array([1, 2, 3])
```

第二种，导入 NumPy 包并指定别名：

```
import numpy as np
```

这种方式会将 NumPy 包导入到当前的命名空间中，并使用别名 np 来代替 NumPy，例如：

```
a = np.array([1, 2, 3])
```

第三种，导入 NumPy 包中的部分模块或函数：

```
from numpy import array
```

这种方式会将 NumPy 包中的 array 函数导入到当前的命名空间中，可以直接调用该函数，例如：

```
a = array([1, 2, 3])
```

第四种，导入 NumPy 包中的所有模块或函数：

```
from numpy import *
```

这种方式会将 NumPy 包中的所有函数和模块导入到当前的命名空间中，可以直接调用任意函数或模块，例如：

```
a = array([1, 2, 3])
```

```
b = random.rand(3, 3)
```

在实际应用中，可以根据需要选择和使用适当的导入方式。一般来说，建议使用第二种（导入 NumPy 包并指定别名）或第三种方式（导入部分模块或函数），这样既可以简化代码，又不会导入太多无用的函数或模块，从而提高代码的可读性和性能。


 注意，请大家采用常用 Python 库/模块的约定成俗的简称，特别不建议大家自由发挥。

表 5. 常用 Python 包的名称及简称

库名	IDE 中库/模块全称	简称	导入	关键词
NumPy	numpy	np	import numpy as np	多维数组、线性代数运算
Pandas	pandas	pd	import pandas as pd	数据帧、数据处理、数据分析
Matplotlib	matplotlib.pyplot	plt	import matplotlib.pyplot as plt	绘图、美化
Seaborn	seaborn	sns	import seaborn as sns	统计可视化
Plotly	plotly.express	px	import plotly.express as px	交互可视化
Streamlit	streamlit	st	import streamlit as st	应用 App

4.6 Pythonic: Python 风格

"Pythonic" 翻译成中文可以是 "符合 Python 风格的"、"Python 风格的" 等。让 Python 代码 Pythonic 是指遵循 Python 社区的最佳实践和代码风格，使代码更加易读、易维护、易扩展和高效。

以下是一些让 Python 代码 Pythonic 的方法：

- ▶ 遵循 PEP8 规范：PEP8 是 Python 社区的代码风格指南，包括缩进、命名、代码结构、注释等。编写符合 PEP8 规范的代码可以提高代码的可读性和可维护性。
- ▶ 使用 Python 内置函数和数据结构：Python 提供了许多内置函数和数据结构，如列表、字典、集合、生成器、装饰器、lambda 表达式等。使用这些功能可以使代码更加简洁、高效和易于理解。
- ▶ 使用异常处理机制：Python 的异常处理机制可以使代码更加健壮和容错。在编写代码时应该预见到可能的异常情况，并使用 try/except 块来处理这些异常情况。
- ▶ 避免使用全局变量：全局变量可以使代码更加难以理解和维护，因为它们可能会被其他代码意外修改。应该尽量避免使用全局变量，而是使用函数或类来封装状态和行为。
- ▶ 使用函数式编程风格：函数式编程风格强调函数的不可变性和无状态性，使得代码更加简洁、高效和易于测试。应该尽可能使用纯函数，避免使用副作用和可变状态。
- ▶ 使用面向对象编程风格：面向对象编程风格可以使代码更加模块化和易于扩展。使用类和对象可以封装状态和行为，使代码更加结构化和易于维护。
- ▶ 编写文档和测试：编写文档和测试可以使代码更加易读、易于理解和易于维护。

有关 PEP8，请参考：

<https://peps.python.org/pep-0008/>

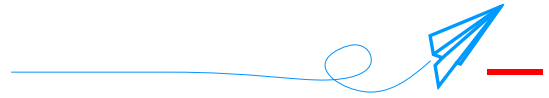
如果在 Python 编程中遇到问题或者 bug，可以去以下几个地方寻求帮助：

- ▶ 官方文档：Python 官方文档提供了丰富的资源，包括语言参考手册、标准库参考手册、教程、示例代码等。可以先在官方文档中查找相关信息，寻找解决问题的方法。
- ▶ <https://stackoverflow.com/>：这是一个广泛使用的程序员问答社区，拥有庞大的用户群体和丰富的问题解答资源。可以在这里提出你的问题，或者搜索其他人遇到的类似问题的解决方法。
- ▶ 此外，ChatGPT 之类的助手工具也可以帮助我们解决编程中遇到的问题。



本章题目仅是请大家在 JupyterLab 中复刻所有示例代码，并逐行注释加强理解。

* 题目不提供答案。



直言不讳地说，对于初学者来说，如果一本 Python 编程教材整本都是基本语法，这本书大概率的命运就是躺在书架上吃灰。Python 初学者最想看的是怎么让 Python 代码跑起来，用 Python 工具解决实际问题，而不是翻一本味同嚼蜡的 Python 词典。

如果遇到具体的 Python 语法问题，大家可以求助 Python 官网、社区、ChatGPT、Stack overflow (<https://stackoverflow.com/>) 等等资源。

如果数学工具有问题建议大家求助：<https://mathworld.wolfram.com/>；也可以参考如下社区 <https://math.stackexchange.com/>。

对于我们，Python 是解决各种问题的工具。希望大家学习 Python 时，一定要吸取英语学习失败的教训，千万不能死磕 Python 语法。死记硬背要不得，千万别把 Python 当成“文科”来学。要用为主、学为辅，边学边用，活学活用。

先让代码“跑”起来，有了成就感、获得感之后，内生的兴趣就会推着大家一路狂奔。