

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



```
◀ XXXXX
◀ XXXXX
◀ XXXXX
◀ XXXXX
◀ XXXXX
◀ XXXXX
```



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。

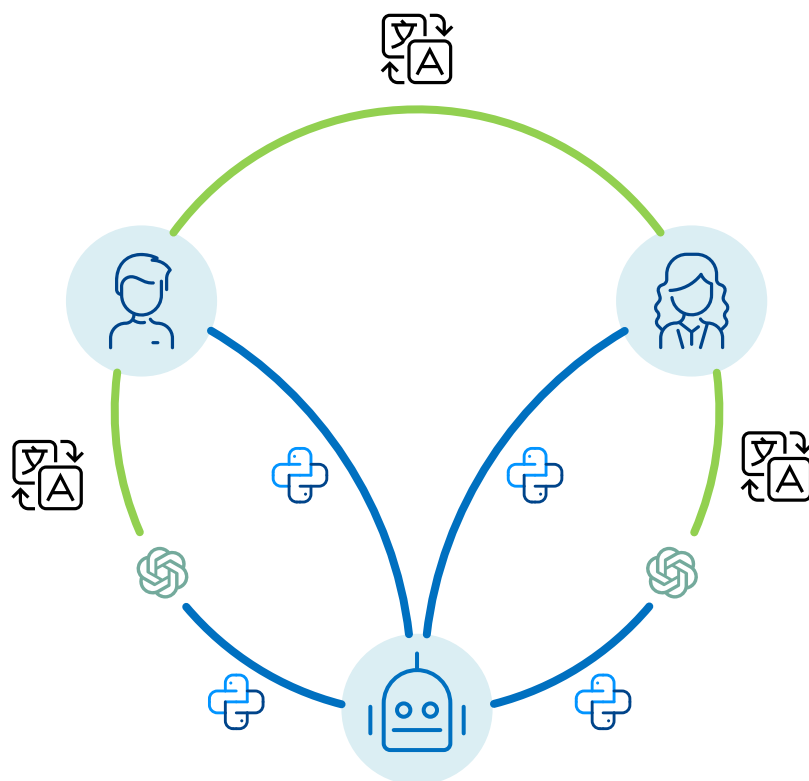


图 1. Python 也是语言

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。

⚠ 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	<code>print('Hey you!')</code> 中的 'Hey you' 是函数 <code>print()</code> 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 <code>import numpy as np</code>
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	<code>3 + 4j</code> 是一个复数
condition	条件	<code>if x > 0:</code> 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	<code>{'name': 'James', 'age': 18}</code> 是一个字典
elif	否则如果	用于在 <code>if</code> 语句中添加多个条件判断
else	否则	用于 <code>if</code> 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	<code>3.14</code> 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 <code>from numpy import random</code>
function	函数	<code>print()</code> 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 <code>if x > 0:</code>
import	导入	用于导入模块，比如 <code>import numpy</code>
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	<code>3</code> 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	<code>[1, 2, 3]</code> 是一个列表
loop	循环	<code>for i in range(10):</code> 是一个循环语句
module	模块	<code>import math</code> 导入了 Python 的 <code>math</code> 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	<code>my_object = MyClass()</code> 中 <code>my_object</code> 是一个 <code>MyClass</code> 类的对象

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

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.")
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 = 5 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 = 5 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。

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

► 面向对象编程 (第 9 章): 定义类、对象、方法、属性等。

4.2 注释

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。

#注释: 整行、单行尾部

如图 2 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。

图 2 展示了两种注释: 1) 整行注释; 2) 单行尾部注释。

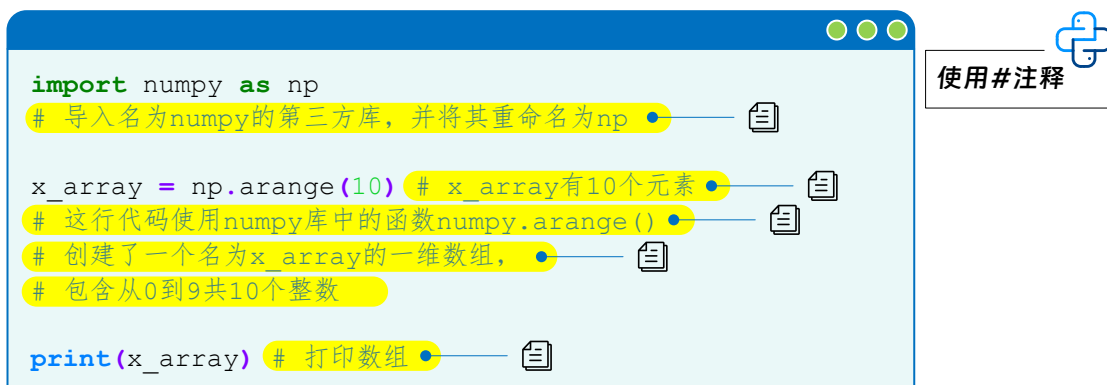


图 2. 举例说明 Python 代码中的注释

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。以下是一个例子，我们定义了一个名为“my_function”的函数，使用三个引号来添加多行注释。

▲ 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。

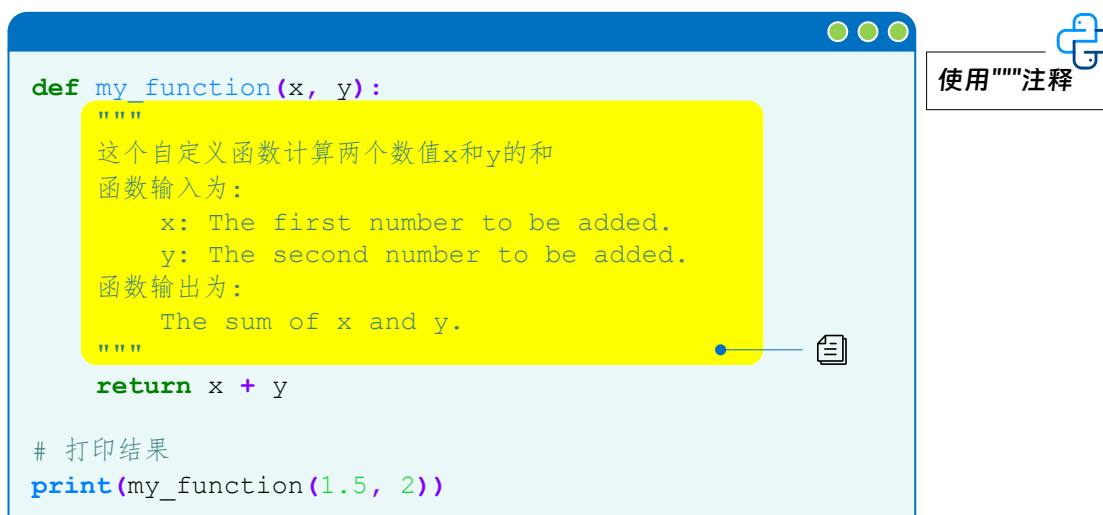


图 3. 用三个引号来添加多行注释

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。



本书第 8 章将介绍自定义函数。

4.3 缩进

相信大家已经在图 3 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，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，为了保证字母、数字、符号等显示时宽度一致，本书正文示例代码采用的字体为 Courier New。

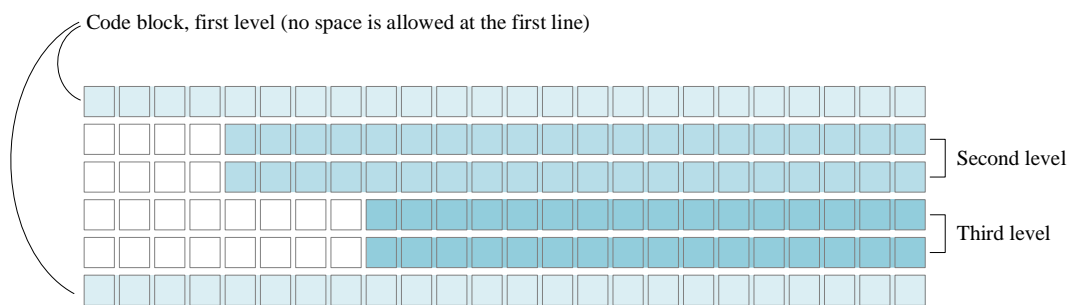


图 4. 缩进形成不同的代码级别

条件语句

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。

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`。



```

# 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)


```

条件语句中
使用缩进

图 5. 条件语句中使用缩进

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。



```

x_string = 'Python 3.X is easy!'

# 利用for循环打印每个字符
for i_str in x_string:
    print(i_str)

```

for循环语
句中使
用缩进

图 6. for 循环语句中使用缩进

函数定义

在函数定义时，函数体内的代码块需要缩进，以表示该代码块属于函数体。

在这个例子中，我们定义 `fibonacci()` 函数生成斐波那契数列 (Fibonacci sequence) 接受一个整数 `n`，它返回 Fibonacci 数列的第 `n` 项。如果 `n` 小于或等于 1，它将直接返回 `n`。否则，它将调用两次自己，并将 `n-1` 和 `n-2` 作为参数传递给它们。最终，当 `n` 达到 0 或 1 时，递归将停止，返回相应的值。

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

通过使用 `for` 循环来输出 Fibonacci 数列的前 10 项，可以看到这个函数在工作时是如何递归调用自己的。

《数学要素》将专门介绍斐波那契数列，《矩阵力量》讲解如何用线性代数工具求解斐波那契数列通项公式。

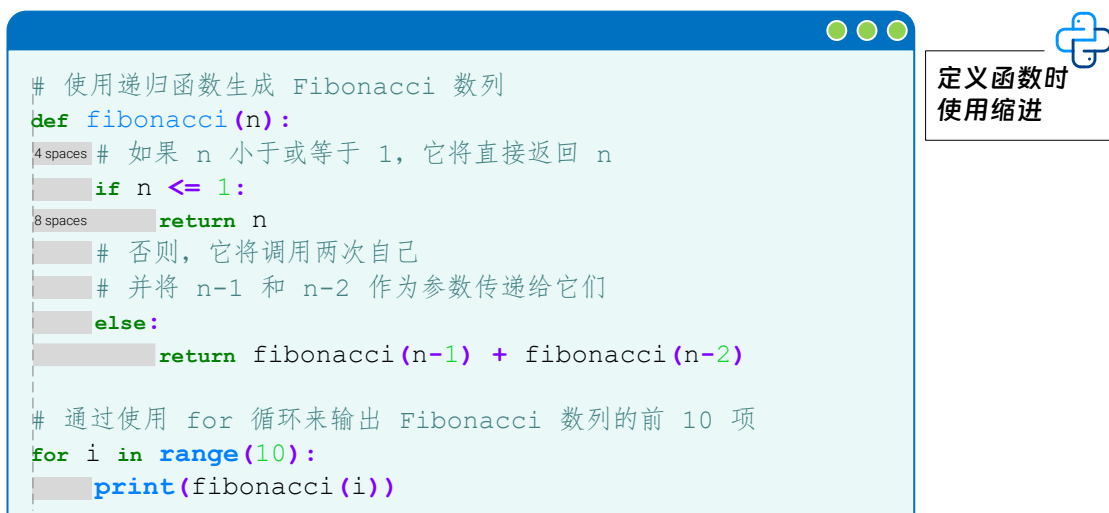


图 7. 定义函数时使用缩进



什么是斐波那契数列？

斐波那契数列是一组数字，其中每个数字都是前两个数字的和。斐波那契数列的前几个数字是 0、1、1、2、3、5、8、13、21、34 等等。斐波那契数列是计算机科学中常用的例子，用于介绍递归和动态规划等概念。在植物学中，叶子、花瓣和果实的排列顺序可以遵循斐波那契数列。许多音乐家和作曲家使用斐波那契数列的规律来创建旋律和和弦。

4.4 变量

在 Python 中，变量 (variable) 是用于存储数据值的标识符。它们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”。下表为 Python 中常见数据类型。本书第 5 章专门介绍数据类型。

数据类型	type()	特点	举例
数字 (Number)	int float	包括整数、浮点数等	x = 10 y = 3.14
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。

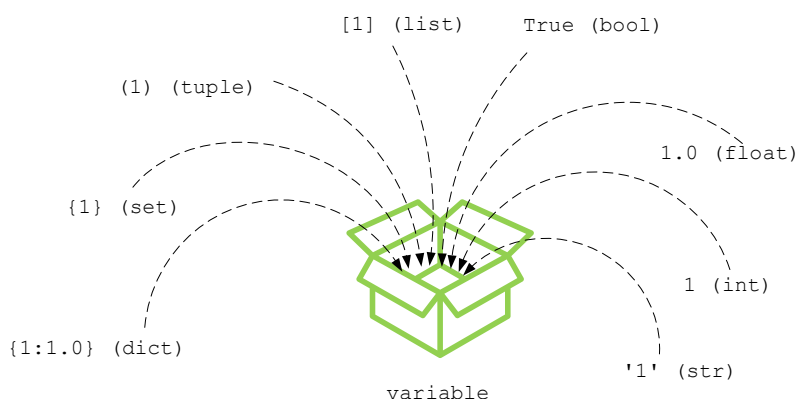


图 8. Python 变量就是个“箱子”，什么都能装



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`。Python 社区普遍采用蛇形命名法，而 Java 和 JavaScript 等语言则更常使用驼峰命名法。

变量赋值

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
```

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 导入包

Python 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 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。

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。

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：

```
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。

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。

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。要用为主、学为辅，边学边用，活学活用。