4

Python 基础

在这一章中，将获得 Python 语言及其核心理念的高层次概述，如何设置 Python 开发环境，以及围绕 Python 编程的关键概念，让你开始学习基础知识。本章对于非 Python 用户来说是一个额外的步骤或前提步骤。如果你对 Python 已经很熟悉了，我建议你快速浏览一下内容，确保你了解所有的关键概念。

4.1. 概述

俗话说，”生活中最好的东西是免费的！”Python 是一种开源的、高级的、面向对象的、解释性的、通用的动态编程语言。它有一个基于社区的开发模式。它的核心设计理论强调代码的可读性，与其他高级编程语言如 Java、C 或 C++ 相比，它的编码结构使程序员能够用较少的代码行阐述计算概念。

Python 的设计理念在文件”The Zen of Python”（Python 增强建议，信息条目编号 20）中得到了很好的总结，以下是总结中的摘要：

- 美的比丑的好–要一致。- 复杂的比复杂的好–使用现有的库。- 简单比复杂好–保持简单和愚蠢（KISS）。- 扁平比嵌套好–避免嵌套 ifs。- 明确的比隐含的好–要清楚。- 稀疏的比密集的好–把代码分成模块。- 可读性很重要–缩进是为了便于阅读。- 特殊情况不足以打破规则 - 一切都是对象。- 错误不应该无声地通过–好的异常处理程序。- 尽管实用性胜过纯洁性–如果需要，就打破规则。- 除非明确地沉默–错误记录和可追溯性。- 在模棱两可的情况下，拒绝猜测的诱惑–Python 的语法比较简单；但是，很多时候我们可能需要花更长的时间来破译它。- 尽管这种方式一开始可能并不明显，除非你是荷兰人–实现某件事情的方式并非只有一种。- 最好是只有一种明显的方法–使用现有的库。- 如果实现方式很难解释，那就是一个坏主意–如果你不能用简单的术语解释，那么你就没有很好地理解它。- 现在做总比不做好–有一些快速/肮脏的方法来完成工作，而不是过多地去尝试优化。- 虽然永远不会比现在 * 好–虽然有快速/肮脏的方法，但不要走那些不允许优雅地回头的路。- 命名空间是一个伟大的想法，所以让我们做更多这样的事情吧 - 要有针对性。- 如果实现起来很容易解释，这可能是一个好主意–简单性。

4.2. 简介

Python 于 1991 年 2 月 20 日正式诞生，版本号为 0.9.0，并在过去 5 年（2012 年至 2016 年）连续成为最受欢迎的语言，走上了巨大的发展道路。它的应用贯穿于各个领域，如网站开发、移动应用开发、科学和数字计算、桌面图形用户界面和复杂的软件开发。尽管 Python 是一种更通用的编程和脚本语言，但在过去 5 年里，它在数据科学家和机器学习工程师中越来越受欢迎。见图4.1。

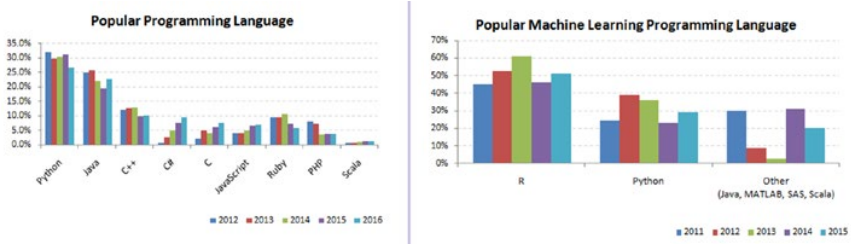

图 4.1: 流行的编码语言（来源：codeeval.com）和流行的机器学习编程语言（来源：KDD poll）。

有一些精心设计的开发环境，如 IPython Notebook 和 Spyder，它们可以快速反省数据，并能以交互方式开发机器学习模型。

强大的模块，如 NumPy 和 Pandas，可以有效地使用数字数据。科学计算通过 SciPy 包变得简单。一些主要的机器学习算法已经在 scikit-learn（也被称为 sklearn）中得到有效实现。HadooPy、PySpark 提供了与大数据技术栈的无缝工作体验。Cython 和 Numba 模块允许以与 C 代码相同的速度执行 Python 代码。nosetest 等模块强调高质量、持续集成测试和自动部署。

综合以上因素，许多机器学习工程师将 Python 作为探索数据、识别模式、建立和部署模型到生产环境的首选语言。最重要的是，各种关键的 Python 软件包的商业友好许可正在鼓励企业和开源社区的合作，使双方都受益。总的来说，Python 编程生态系统允许快速的结果和快乐的程序员。我们已经看到开发人员成为开源社区的一部分的趋势，为全球社区使用的错误修复和新算法做出贡献，同时保护他们各自工作的公司的核心知识产权。

4.3. Python

Python 3.4.x is the latest version and comes with nicer, consistent functionalities! However, there is very limited third-party module support for it, and this will be the trend for at least a couple of more years. However, all major frameworks still run on version 2.7.x and are likely to continue to do so for a significant amount of time. Therefore, it is advised to start with Python 2, for the fact that it is the most widely used version for building machine learning systems as of today.

For an in-depth analysis of the differences between python 2 vs. 3, you can refer to Wiki.python.org (https://wiki.python.org/moin/Python2orPython3v), which says that there are benefits to each.

I recommend Anaconda (Python distribution), which is BSD licensed and gives you permission to use it commercially and for redistribution. It has around 270 packages including the most important ones for most scientific applications, data analysis, and machine learning such as NumPy, SciPy, Pandas, IPython, matplotlib, and scikit-learn. It also provides a superior environment tool conda that allows you to easily switch between environments, even between Python

2 and 3 (if required). It is also updated very quickly as soon as a new version of a package is released and you can just use conda update <packagename> to update it. You can download the latest version of Anaconda from their official website at https://www.continuum.io/downloads and follow the installation instructions.

Python 3.11.x 是最新的版本，并且具有更好的、一致的功能！Python 3.11 的具体改进主要表现在：更详实的 Error Tracebacks、更快的代码执行、更好的异步任务语法、改进类型变量、支持 TOML 配置解析以及一些其他非常酷的功能（包括快速启动、Zero-Cost 异常处理、异常组等）。

但是，对它的第三方模块支持非常有限。然而，对它的第三方模块支持非常有限，这将是至少几年内的趋势。然而，所有主要的框架仍然运行在 2.7.x 版本上，并可能在相当长的时间内继续这样做。因此，我们建议从 Python 2 开始，因为它是截至目前用于构建机器学习系统的最广泛的版本。[1]

推荐 Anaconda（Python 发行版），它是 BSD 授权的，允许你在商业上和再发行上使用它。它有大约 270 个软件包，包括大多数科学应用、数据分析和机器学习的最重要的软件包，如 NumPy、SciPy、Pandas、IPython、matplotlib 和 scikit-learn。它还提供了一个卓越的环境工具 conda，使你能够在不同的环境中轻松切换，甚至在 Python 2 和 3 之间（如果需要）。一旦有新版本的软件包发布，它的更新速度也非常快，你只需使用 conda update <packagename> 来更新它。你可以从他们的官方网站 https://www.continuum.io/downloads 下载最新版本的 Anaconda，并按照安装说明操作。

4.3.1. Windows 平台的安装

- 根据你的系统配置（32 或 64 位），下载安装程序。
- 双击.exe 文件来安装 Anaconda，并按照屏幕上的安装向导操作。

4.3.2. OSX 平台的安装

对于 Mac OS，你可以通过图形安装程序或从命令行进行安装。

图形化安装程序

- 下载图形化的安装程序。
- 双击下载的.pkg 文件，按照屏幕上的安装向导指示操作。

或者

命令行安装程序

- 下载命令行安装程序。
- 在你的终端窗口中键入以下其中一个，并按照提示：bash <Anaconda2-x.x.x-MacOSX-x86_64.sh>。

4.3.3. Linux 平台的安装

- 根据你的系统配置（32 或 64 位），下载安装程序。- 在你的终端窗口中键入以下内容之一，并按照指示操作：bash Anaconda2-x.x.x-Linux-x86_xx.sh。

[1] 关于 Python 2 与 3 之间的差异的深入分析，你可以参考 Wiki. python.org (https://wiki.python.org/moin/Python2orPython3v)，每一种都有好处。

4.3.4. 官方网站的 Python

出于某种原因，如果你不想使用 Anaconda 编译包，你也可以去 Python 的官方网站 https://www.python.org/dow 并浏览相应的操作系统部分，下载安装程序。请注意，OSX 和大多数 Linux 都预装了 Python，所以不需要额外的配置。

为 Windows 设置 PATH。当你运行安装程序时，确保勾选" 将 Python 添加到 PATH 选项"。这将使我们能够从任何目录中调用 Python 解释器。

如果你错过了勾选" 将 Python 添加到 PATH 选项"，请按照以下说明操作。- 右键点击" 我的电脑"。- 点击" 属性"。- 在侧板中点击" 高级系统设置"。- 点击" 环境变量"。- 点击系统变量下面的" 新建"。- 对于名称，输入 pythonexe（或任何你想要的）。- 对于值，输入 Python 的路径（例如：C:32。- 现在编辑 Path 变量（在系统部分），并在已经存在的内容的末尾添加%pythonexe%; 。

4.3.5. 运行 Python

在命令行中，输入"Python " 来打开交互式解释器。一个 Python 脚本可以用这里的语法在命令行上执行。python <scriptname.py>。本书中使用的所有代码都可作为 IPython 笔记本（现在称为 Jupyter 笔记本），它是一个交互式的计算环境，你可以在其中结合代码执行、丰富的文本、数学、绘图和丰富的媒体。你可以通过点击开始菜单中 Anaconda 安装的图标（Windows）或在终端中输入"jupyter notebook"（Windows 中为 cmd）来启动 Jupyter Notebook。然后浏览你想使用的相关 IPython 笔记本文件。请注意，代码可能会随着软件包版本的变化而中断，因此，为了重现性，我已经分享了我的软件包版本号，请参考 Module_Versions IPython Notebook。

4.4. 关键概念

在 Python 中，有几个基本概念，理解这些概念对于初学者来说是必不可少的。下面将对这些概念进行简要介绍。

4.4.1. Python 的标识符

顾名思义，标识符帮助我们将一个实体与另一个实体区分开来。Python 实体如类、函数和变量被称为标识符。- 它可以是大写字母或小写字母的组合（a 到 z 或 A 到 Z）。- 它可以是任何数字（0 到 9）或下划线（□□−□$Python$□□□□□□□□□□□□□□□□□□□□−□□□□□□□□□□□□□□1$variable$□□□□□□$variable$1□□□□□ $Python$□□□□□□□□□□□1−1□□□□□□□□□□□□□−□□□□□□()□□□□□□□□□□@□□$□%□□□□□□□□□□□□□□□□

4.4.2. 第一个 Python 程序

在命令提示符上启动 Python 交互式，然后输入以下文字并按回车键。

Listing 4.1: 第一个 Python 程序

```
1 >>> print "Hello, Python World!"
```

如果你正在运行来自 Anaconda 编译包的 Python 2.7.x，那么你也可以使用带括号的 print 语句，如 print ("Hello, Python World!")，这将产生以下结果。Hello, Python World!

4.4.3. 代码块（缩进套件）

了解如何在 Python 中编写代码块是非常重要的。让我们看一下围绕代码块的两个关键概念。

缩进

Python 最独特的特征之一是它使用缩进来标记代码块。在 Python 中，每一行代码都必须缩进相同的量来表示一个代码块。与其他大多数编程语言不同，缩进不是为了让代码看起来漂亮。缩进是用来表示一个代码或语句属于哪个代码块的。

套件

在 Python 中，构成单个代码块的单个语句的集合被称为套件。对于复合或复杂的语句，如 if、while、def 和 class，需要在标题行后跟上一个套件 (我们将在后面的章节中详细介绍这些语句)。标题行以一个关键字开始，以冒号 (:) 结束，后面是构成套件的一个或多个行。

Listing 4.2: 示例程序 1

```
Listing 1-1. Example of correct indentation
# Correct indentation
print ("Programming is an important skill for Data Science")
print ("Statistics is a important skill for Data Science")
print ("Business domain knowledge is a important skill for Data Science")
# Correct indentation, note that if statement here is an example of suites x= 1
if x == 1:
    print ('x has a value of 1')
else:
    print ('x does NOT have a value of 1')
```

Listing 4.3: 示例程序 2

```
Listing 1-2. Example of incorrect indentation
# incorrect indentation, program will generate a syntax error
# due to the space character inserted at the beginning of second line
print ("Programming is an important skill for Data Science")
print ("Statistics is a important skill for Data Science")
print ("Business domain knowledge is a important skill for Data Science") 3
# incorrect indentation, program will generate a syntax error
# due to the wrong indentation in the else statement
x= 1
if x == 1:
    print ('x has a value of 1')
else:
 print ('x does NOT have a value of 1')
```

基本对象类型

根据 Python 数据模型参考，对象是 Python 对数据的概念。Python 程序中的所有数据都由对象或对象之间的关系来表示。从某种意义上说，与冯-诺依曼的" 存储程序计算机" 模型一致，代码也由对象表示。

Listing 4.4: 示例代码

```
Listing 1-3. Code For Basic Object Types
none = None # singleton null object
boolean = bool(True)
integer = 1
Long = 3.14
# float
Float = 3.14
```

```
Float_inf = float('inf')
Float_nan = float('nan')
# complex object type, note the usage of letter j
Complex = 2+8j
# string can be enclosed in single or double quote
string = 'this is a string'
me_also_string = "also me"
List = [1, True, 'ML'] # Values can be changed
Tuple = (1, True, 'ML') # Values can not be changed
Set = set([1,2,2,2,3,4,5,5]) # Duplicates will not be stored
# Use a dictionary when you have a set of unique keys that map to values
Dictionary = {'a':'A', 2:'AA', True:1, False:0}
# lets print the object type and the value
print type(none), none
print type(boolean), boolean
print type(integer), integer
print type(Long), Long
print type(Float), Float
print type(Float_inf), Float_inf
print type(Float_nan), Float_nan
print type(Complex), Complex
print type(string), string
print type(me_also_string), me_also_string
print type(Tuple), Tuple
print type(List), List
print type(Set), Set
print type(Dictionary), Dictionary
```

Listing 4.5: 示例代码

```
<type 'NoneType'> None
<type 'bool'> True
<type 'int'> 1
<type 'float'> 3.14
<type 'float'> 3.14
<type 'float'> inf
<type 'float'> nan
<type 'complex'> (2+8j)
<type 'str'> this is a string
<type 'str'> also me
<type 'tuple'> (1, True, 'ML')
<type 'list'> [1, True, 'ML']
<type 'set'> set([1, 2, 3, 4, 5])
<type 'dict'> {'a': 'A', True: 1, 2: 'AA', False: 0}
```

使用列表、图元、集合、字典

- 列表：当你需要一个同质集合的有序序列时使用，其值可以在程序中稍后改变。- 元组（Tuple）：当你需要一个异质集合的有序序列时使用，其值不需要在程序中稍后改变。- 集合：当你不需要存储重复的东西，而且你不关心顺序或项目时，它是理想的使用方式。你只想知道一个特定的值是否已经存在。- 词典：当你需要将值与键联系起来时，它是理想的选择，以便使用一个键有效地查找它们。

Python 中的注释

单行注释：任何由 (hash) 后面的字符和到行尾的字符都被认为是注释的一部分，Python 解释器会忽略它们。多行注释：字符串”” 之间的任何字符 (称为多行字符串) 之间的任何字符，也就是说，在你的注释的开头和结尾都会被 Python 解释器所忽略。

Listing 4.6: 注解示例

```
# This is a single line comment in Python
print "Hello Python World" # This is also a single line comment in Python
""" This is an example of a multi line
comment that runs into multiple lines.
Everything that is in between is considered as comments
"""
```

4.4.4. 多行声明

Python 在小括号、大括号和大括号内的斜线续行是最受欢迎的长线续行方式。使用反斜线表示续行，使可读性更好；但是如果需要，你可以在表达式周围再加一对小括号。正确缩进你的代码的续行是很重要的。注意，在二进制运算符周围缩进的首选位置是运算符之后，而不是运算符之前。参见清单 1-5。

Listing 4.7: 多行声明示例 1

```
# Example of implicit line continuation
x = ('1' + '2' +
'3' + '4')
# Example of explicit line continuation
y = '1' + '2' + \
'11' + '12'
weekdays = ['Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday']
weekend = {'Saturday',
           'Sunday'}
print ('x has a value of', x)
print ('y has a value of', y)
print days
print weekend
```

Listing 4.8: 多行声明示例 2

```
('x has a value of', '1234')
('y has a value of', '1234')
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
set(['Sunday', 'Saturday'])
```

单行多语句

通过使用分号 (;)，Python 也允许在一行中出现多个语句，前提是该语句不开始一个新的代码块。

Listing 4.9: 单行多语句示例

```
Listing 1-6. Code example for multistatements on a single line import os; x = 'Hello'; print x
```

4.4.5. 基本操作

在 Python 中，运算符是可以操作操作数的值的特殊符号。例如，让我们考虑表达式 1 + 2 = 3。这里，1 和 2 被称为操作数，是运算符操作的值，符号 + 被称为运算符。Python 语言支持以下类型的运算符。- 算术运算符 - 比较或关系运算符 - 赋值运算符 - 位操作符 - 逻辑运算符 - 成员操作符 - 身份操作符让我们通过实例来逐一学习所有的操作符。

算术运算符

算术运算符用于对数字进行数学运算，如加法、减法、乘法、除法，等等。

Listing 4.10: 算术运算符示例 1

```
Listing 1-7. Example code for arithmetic operators
# Variable x holds 10 and variable y holds 5 x = 10
y= 5
# Addition
print "Addition, x(10) + y(5) = ", x + y
# Subtraction
print "Subtraction, x(10) - y(5) = ", x - y
# Multiplication
print "Multiplication, x(10) * y(5) = ", x * y
# Division
print "Division, x(10) / y(5) = ",x / y
# Modulus
print "Modulus, x(10) % y(5) = ", x % y
# Exponent
print "Exponent, x(10)**y(5) = ", x**y
# Integer division rounded towards minus infinity
print "Floor Division, x(10)//y(5) = ", x//y
-------- output --------
Addition, x(10) + y(5) =  15
Subtraction, x(10) - y(5) =  5
Multiplication, x(10) * y(5) =  50
Divions, x(10) / y(5) =  2
Modulus, x(10) % y(5) =  0
Exponent, x(10)**y(5) =  100000
Floor Division, x(10)//y(5) =  2
```

比较或关系操作符

顾名思义，比较运算符或关系运算符是用来比较数值的。对于一个给定的条件，它将返回 True 或 False 作为结果。

Listing 4.11: 比较或关系操作符示例

```
# Variable x holds 10 and variable y holds 5 x = 10
y= 5
# Equal check operation
print "Equal check, x(10) == y(5) ", x == y
# Not Equal check operation
print "Not Equal check, x(10) != y(5) ", x != y
# Not Equal check operation
print "Not Equal check, x(10) <>y(5) ", x<>y
# Less than check operation
print "Less than check, x(10) <y(5) ", x<y
# Greater check operation
```

```
print "Greater than check, x(10) >y(5) ", x>y
# Less than or equal check operation
print "Less than or equal to check, x(10) <= y(5) ", x<= y
# Greater than or equal to check operation
print "Greater than or equal to check, x(10) >= y(5) ", x>= y
```

Listing 4.12: 比较或关系操作符输出

```
-------- output --------
Equal check, x(10) == y(5)  False
Not Equal check, x(10) != y(5)  True
Not Equal check, x(10) <>y(5)  True
Less than check, x(10) <y(5)  False
Greater than check, x(10) >y(5)  True
Less than or equal to check, x(10) <= y(5)  False
Greater than or equal to check, x(10) >= y(5)  True
```

赋值操作

在 Python 中，赋值运算符被用来给变量赋值。例如，考虑 $x = 5$；这是一个简单的赋值运算符，它把运算符右边的数值 5 赋给运算符左边的变量 x。在 Python 中有一系列的复合运算符，如 $x+ = 5$，对变量进行加法运算，然后再进行相同的赋值。它和 $x = x + 5$ 一样好。

Listing 4.13: 赋值操作代码示例

```
Listing 1-9. Example code for assignment operators
# Variable x holds 10 and variable y holds 5 x= 5
y = 10
x += y
print "Value of a post x+=y is ", x
x *= y
print "Value of a post x*=y is ", x
x /= y
print "Value of a post x/=y is ", x
x %= y
print "Value of a post x%=y is ", x
x **= y
print "Value of x post x**=y is ", x
x //= y
print "Value of a post x//=y is ", x
```

Listing 4.14: 输出

```
-------- output --------
Value of a post x+=y is  15
Value of a post x*=y is  150
Value of a post x/=y is  15
Value of a post x%=y is  5
Value of a post x**=y is  9765625
Value of a post x//=y  is  976562
```

位操作符

你可能知道，计算机中的一切都由比特表示，也就是一系列的 0（零）和 1（一）。位操作符使我们能够直接操作或操纵位。让我们了解一下基本的位操作。位操作符的主要用途之一是用于解析十六进制的颜色。

众所周知，对于 Python 编程的新手来说，位操作符是令人困惑的，所以如果你一开始不理解可用性，也不要着急。事实上，在日常的机器学习编程中，你并不真正会看到位运算符。然而，对这些运算符有所了解是很好的。

例如，我们假设 $x = 10$（二进制 0000 1010），$y = 4$（二进制 0000 0100）。

Listing 4.15: 位操作符示例代码 1

```
Listing 1-10. Example code for bitwise operators
# Basic six bitwise operations
# Let x = 10 (in binary0000 1010) and y = 4 (in binary0000 0100) x = 10
y= 4
print x >> y # Right Shift print x << y # Left Shift print x& y # Bitwise AND print x| y # Bitwise
    OR print x ^ y # Bitwise XOR print ~x # Bitwise NOT
```

Listing 4.16: 位操作符示例代码 2

```
-------- output --------
0
160
0
14
14
-11
```

逻辑运算符

AND, OR, NOT 运算符被称为逻辑运算符。这些运算符对于根据给定条件检查两个变量是非常有用的，其结果将适当地显示为真或假。

Listing 4.17: 逻辑运算符

```
Listing 1-11. Example code for logical operators
var1 = True
var2 = False
print('var1 and var2 is',var1and var2)
print('var1 or var2 is',var1 or var2)
print('not var1 is',not var1)
```

Listing 4.18: 逻辑运算符输出

```

-------- output --------
('var1 and var2 is', False)
('var1 or var2 is', True)
('not var1 is', False)
```

成员运算符

成员运算符对于测试一个值是否在一个序列，即字符串、列表、元组、集合或字典中找到很有用。在 Python 中有两个成员运算符，'in' 和'not in'。注意，在字典的情况下，我们只能测试是否存在键 (而不是值)。

Listing 4.19: 成员运算符代码示例

```
var1 = 'Hello world'          # string
```

```
var1 = {1:'a',2:'b'}# dictionary
print('H' in var1)
print('hello' not in var2)
print(1 in var2)
print('a' in var2)
```

Listing 4.20: 成员运算符代码示例输出结果

```
-------- output --------
True
True
True
False
```

身份运算符

身份运算符对于测试两个变量是否存在于内存的同一区域非常有用。在 Python 中有两个身份运算符，”是”和”不是”。注意，两个变量的值相等并不意味着它们是相同的。

Listing 4.21: 身份运算符示例代码

```
Listing 1-13.Example code for identity operators
var1 = 5
var1 = 5
var2 = 'Hello'
var2 = 'Hello'
var3 = [1,2,3]
var3 = [1,2,3]
print(var1 is not var1)
print(var2 is var2)
print(var3 is var3)
```

Listing 4.22: 身份运算符示例代码输出

```
-------- output --------
False
True
False
```

控制结构

控制结构是编程中的基本选择或决策过程。它是一大块代码，用来分析变量的值，并根据给定的条件决定一个方向。在 Python 中，主要有两种类型的控制结构。(1) 选择和 (2) 迭代。

选择语句

选择语句允许程序员检查一个条件，并根据结果执行不同的操作。这种有用的结构有两个版本：（1）if 和（2）if...else。

Listing 4.23: 选择语句示例 1

```
Listing 1-14. Example code for a simple ‘if’ statement
var = -1
if var < 0:
    print var
    print("the value of var is negative")
```

```
# If there is only a signle cluse then it may go on the same line as the
header statement
if ( var  == -1 ) : print "the value of var is negative"
Listing 1-15. Example code for  'if else'  statement var = 1
if var < 0:
    print "the value of var is negative"
    print var
else:
    print "the value of var is positive"
    print var
```

Listing 4.24: 选择语句示例 2

```
Listing 1-16. Example code for nested if else statements Score = 95
if score >= 99:
    print('A')
elif score >=75:
    print('B')
elif score >= 60:
    print('C')
elif score >= 35:
    print('D')
else:
    print('F')
```

迭代

循环控制语句使我们能够多次执行一个或一组编程语句，直到一个给定的条件得到满足。Python 提供了两个基本的循环语句。(1) for (2) while 语句。For 循环。它允许我们对代码块执行特定的次数或针对特定的条件，直到它得到满足。

Listing 4.25: 迭代示例

```
Listing 1-17. Example codes for a  'for loop'  statement
# First Example
print "First Example"
for item in [1,2,3,4,5]:
    print 'item :', item
# Second Example
print "Second Example"
letters = ['A', 'B', 'C']
for letter in letters:
    print ' First loop letter :', letter
# Third Example - Iterating by sequency index
print "Third Example"
for index in range(len(letters)):
    print 'First loop letter :', letters[index]
# Fourth Example - Using else statement
print "Fourth Example"
for item in [1,2,3,4,5]:
    print 'item :', item
else:
    print 'looping over item complete!'
----- output ------
First Example
item : 1
```

```
item : 2
item : 3
item : 4
item : 5
Second Example
First loop letter : A
First loop letter : B
First loop letter : C
Third Example
First loop letter : A
First loop letter : B
First loop letter : C
Fourth Example
item : 1
item : 2
item : 3
item : 4
item : 5
looping over item complete!
```

While 循环。while 语句重复一组代码，直到条件为真。

Listing 4.26: While 循环示例代码 t

```
Listing 1-18. Example code for while loop statement
count = 0
while (count <3):
    print 'The count is:', count
    count = count + 1
```

一个 else 语句可以和 while 循环一起使用，当条件变为假时，else 将被执行。

Listing 4.27: else 语句示例语句

```
count = 0
while count <3:
    print count, " is  less than 5"
    count = count + 1
else:
    print count, " is not less than 5"
```

列表

Python 的列表是最灵活的数据类型。它可以通过在方括号之间写一个以逗号分隔的值的列表来创建。注意，列表中的项目不一定是相同的数据类型。

Listing 4.28: 列表代码示例 1

```
Listing 1-20. Example code for accessing lists
# Create lists
list_1 = ['Statistics', 'Programming', 2016, 2017, 2018];
list_2 = ['a', 'b', 1, 2, 3, 4, 5, 6, 7 ];
# Accessing values in lists
print "list_1[0]: ", list_1[0]
print "list2_[1:5]: ", list_2[1:5]
---- output ----
list_1[0]:  Statistics
list2_[1:5]:  ['b', 1, 2, 3]
```

Listing 4.29: 列表代码示例 2

```
1
2 Listing 1-21. Example code for adding new values to lists print "list_1 values: ", list_1
3 # Adding new value to list
4 list_1.append(2019)
5 print "list_1 values post append: ", list_1
6 ---- output ----
7 list_1 values:  ['c', 'b', 'a', 3, 2, 1]
8 list_1 values post append:  ['c', 'b', 'a', 3, 2, 1, 2019]
```

Listing 4.30: 列表代码示例 3

```
1 Listing 1-22. Example code for updating existing values of lists print "Values of list_1: ", list_1
2 # Updating existing value of list
3 print "Index 2 value : ", list_1[2]
4 list_1[2] = 2015;
5 print "Index 2's new value : ", list_1[2]
6 ---- output ----
7 Values of list_1:  ['c', 'b', 'a', 3, 2, 1, 2019]
8 Index 2 value :  a
9 Index 2's new value :  2015
```

Listing 4.31: 列表代码示例 4

```
1 Listing 1-23. Example code for deleting a list element Print "list_1 values: ", list_1
2 # Deleting list element
3 del list_1[5];
4 print "After deleting value at index 2 : ", list_1
5 ---- output ----
6 list_1 values: ['c', 'b', 2015, 3, 2, 1, 2019]
7 After deleting value at index 2 :  ['c', 'b', 2015, 3, 2, 2019]
```

Listing 4.32: 列表代码示例 5

```
1 Listing 1-24. Example code for basic operations on lists
2 print "Length: ", len(list_1)
3 print "Concatenation: ", [1,2,3] + [4, 5, 6]
4 print "Repetition :", ['Hello'] * 4
5 print "Membership :", 3 in [1,2,3]
6 print "Iteration :"
7 for x in [1,2,3]: print x
8 # Negative sign will count from the right
9 print "slicing :", list_1[-2]
10 # If you dont specify the end explicitly, all elements from the specified
11 start index will be printed
12 print "slicing range: ", list_1[1:]
13 # Comparing elements of lists
14 print "Compare two lists: ", cmp([1,2,3, 4], [1,2,3])
15 print "Max of list: ", max([1,2,3,4,5])
16 print "Min of list: ", min([1,2,3,4,5])
17 print "Count number of 1 in list: ", [1,1,2,3,4,5,].count(1)
18 list_1.extend(list_2)
19 print "Extended :", list_1
20 print "Index for Programming : ", list_1.index( 'Programming')
21 print list_1
22 print "pop last item in list: ", list_1.pop()
```

```
print "pop the item with index 2: ", list_1.pop(2)
list_1.remove('b')
print "removed b from list: ", list_1
list_1.reverse()
print "Reverse: ", list_1
list_1 = ['a', 'b','c',1,2,3]
list_1.sort()
print "Sort ascending: ", list_1
list_1.sort(reverse = True)
print "Sort descending: ", list_1
---- output ----
Length:  5
Concatenation:  [1, 2, 3, 4, 5, 6]
Repetition : ['Hello', 'Hello', 'Hello', 'Hello']
Membership : True
Iteration :
1
2
3
slicing : 2017
slicing range: ['Programming', 2015, 2017, 2018]
Compare two lists: 1
Max of list: 5
Min of list: 1
Count number of 1 in list: 2
Extended : ['Statistics', 'Programming', 2015, 2017, 2018, 'a', 'b', 1, 2, 3, 4, 5, 6, 7]
Index for Programming : 1
['Statistics', 'Programming', 2015, 2017, 2018, 'a', 'b', 1, 2, 3, 4, 5, 6, 7] pop last item in
    list: 7
pop the item with index 2: 2015
removed b from list: ['Statistics', 'Programming', 2017, 2018, 'a', 1, 2, 3, 4, 5, 6]
Reverse: [6, 5, 4, 3, 2, 1, 'a', 2018, 2017, 'Programming', 'Statistics'] Sort ascending: [1, 2, 3,
     'a', 'b', 'c']
Sort descending: ['c', 'b', 'a', 3, 2, 1]
```

元素

Python 元组是一个序列或一系列不可变的 Python 对象，与列表非常相似。然而，在列表和元组之间存在一些本质的区别，

1. 与列表不同，图元的对象不能被改变。2. 2. 图元是用小括号定义的，但列表是用方括号定义的。

Listing 4.33: 创建元组的示例代码 # 创建一个元组

```
%Listing 1-25. Example code for creating tuple # Creating a tuple
Tuple = ()
print "Empty Tuple: ", Tuple
Tuple = (1,)
print "Tuple with single item: ", Tuple
Tuple = ('a','b','c','d',1,2,3)
print "Sample Tuple :", Tuple
```

Listing 4.34: 输出结果

```
---- output ----
Empty Tuple:  ()
```

```
Tuple with single item:  (1,)
Sample Tuple : ('a', 'b', 'c', 'd', 1, 2, 3)
```

Listing 4.35: 访问元组的代码示例 # 访问元组中的项目

```
%Listing 1-26. Example code for accessing tuple # Accessing items in tuple
Tuple = ('a', 'b', 'c', 'd', 1, 2, 3)
print "3rd item of Tuple:", Tuple[2]
print "First 3 items of Tuple", Tuple[0:2]
---- output ----
3rd item of Tuple: c
First 3 items of Tuple ('a', 'b')
```

Listing 4.36: 删除元组的代码示例 # 删除元组

```
%Listing 1-27. Example code for deleting tuple # Deleting tuple
print "Sample Tuple: ", Tuple
del Tuple
print Tuple # Will throw an error message as the tuple does not exist
```

Listing 4.37: 删除元组的代码示例 # 删除元组的输出

```
---- output ----
Sample Tuple:  ('a', 'b', 'c', 'd', 1, 2, 3)
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-35-6a0deb3cfbcf> in <module>()
      3 print "Sample Tuple: ", Tuple
4 del Tuple
----> 5 print Tuple # Will throw an error message as the tuple does not exist
NameError: name 'Tuple' is not defined
```

Listing 4.38: 对元组进行基本操作的代码示例（并非详尽无遗）# 基本元组操作

```
%Listing 1-28. Example code for basic operations on tupe (not exhaustive) # Basic Tuple operations
Tuple = ('a','b','c','d',1,2,3)
print "Length of Tuple: ", len(Tuple)
Tuple_Concat = Tuple + (7,8,9)
print "Concatinated Tuple: ", Tuple_Concat
print "Repetition: ", (1, 'a',2, 'b') * 3
print "Membership check: ", 3 in (1,2,3)
# Iteration
for x in (1, 2, 3): print x
print "Negative sign will retrieve item from right: ", Tuple_Concat[-2]
print "Sliced Tuple [2:] ", Tuple_Concat[2:]
# Comparing two tuples
print "Comparing tuples (1,2,3) and (1,2,3,4): ", cmp((1,2,3), (1,2,3,4))
print "Comparing tuples (1,2,3,4) and (1,2,3): ", cmp((1,2,3,4), (1,2,3))
# Find max
print "Max of the Tuple (1,2,3,4,5,6,7,8,9,10): ",
max((1,2,3,4,5,6,7,8,9,10))
print "Min of the Tuple (1,2,3,4,5,6,7,8,9,10): ",
min((1,2,3,4,5,6,7,8,9,10))
print "List [1,2,3,4] converted to tuple: ", type(tuple([1,2,3,4]))
```

Listing 4.39: 评估模型在测试集上的表现

```
---- output ----
 Length of Tuple:  7
Concatinated Tuple:  ('a', 'b', 'c', 'd', 1, 2, 3, 7, 8, 9)
Repetition:  (1, 'a', 2, 'b', 1, 'a', 2, 'b', 1, 'a', 2, 'b')
Membership check:  True
1
2
3
Negative sign will retrieve item from right:  8
Sliced Tuple [2:]  ('c', 'd', 1, 2, 3, 7, 8, 9)
Comparing tuples (1,2,3) and (1,2,3,4):  -1
Comparing tuples (1,2,3,4) and (1,2,3):  1
Max of the Tuple (1,2,3,4,5,6,7,8,9,10):  10
Min of the Tuple (1,2,3,4,5,6,7,8,9,10):  1
List [1,2,3,4] converted to tuple:  <type 'tuple'>
```

4.4.6. 集合

Sets

顾名思义，集合是数学集合的实现。集合的三个关键特征如下。1. 项目的集合是无序的。2. 没有重复的项目被存储，这意味着每个项目都是唯一的。3. 集合是可变的，这意味着它的项目可以被改变。

一个项目可以被添加或从集合中删除。数学上的集合操作，如联合、相交等，可以在 Python 集合上执行。

Listing 4.40: 创建集合的示例代码

```
Listing 1-29. Example code for creating sets
# Creating an empty set
languages = set()
print type(languages), languages
languages = {'Python', 'R', 'SAS', 'Julia'}
print type(languages), languages
# set of mixed datatypes
mixed_set = {"Python", (2.7, 3.4)}
print type(mixed_set), languages
---- output ----
<type 'set'> set([])
<type 'set'> set(['SAS', 'Python', 'R', 'Julia'])
<type 'set'> set(['SAS', 'Python', 'R', 'Julia'])
```

访问集合元素

Accessing Set Elements

如下：

Listing 4.41: 访问集合元素的代码示例

```
Listing 1-30. Example code for accessing set elements
print list(languages)[0]
print list(languages)[0:3]
```

Listing 4.42: 输出

```
---- output ----
```

```
C
['C', 'Java', 'Python']
```

在 Python 中改变一个集合

尽管集合是可变的，但由于它们是无序的，对它们的索引将没有任何意义。所以集合不支持使用索引或分片来访问或改变一个项目/元素。add() 方法可以用来添加一个元素，update() 方法可以用来添加多个元素。注意，update() 方法可以接受图元、列表、字符串或其他集合格式的参数。然而，在所有情况下，重复的部分都会被忽略。

Listing 4.43: 改变集合元素的代码示例

```
Listing 1-31. Example code for changing set elements
# initialize a set
languages = {'Python', 'R'}
print(languages)
# add an element
languages.add('SAS')
print(languages)
# add multiple elements
languages.update(['Julia','SPSS'])
print(languages)
# add list and set
languages.update(['Java','C'], {'Machine Learning','Data Science','AI'})
print(languages)
```

Listing 4.44: 输出

```
---- output ----
set(['Python', 'R'])
set(['Python', 'SAS', 'R'])
set(['Python', 'SAS', 'R', 'Julia', 'SPSS'])
set(['C', 'Java', 'Python', 'Data Science', 'Julia', 'SPSS', 'AI', 'R',
'SAS', 'Machine Learning'])
```

移除集合中的项目

discard() 或 remove() 方法可以用来从一个集合中删除一个特定的项目。discard() 和 remove() 的根本区别在于，如果项目不存在于集合中，前者不会采取任何行动，而 remove() 在这种情况下会引发一个错误。

Listing 4.45: 从集合中删除项目的代码示例

```
%Listing 1-32. Example code for removing items from set
# remove an element
languages.remove('AI')
print(languages)
# discard an element, although AI has already been removed discard will not
throw an error
languages.discard('AI')
print(languages)
# Pop will remove a random item from set
print "Removed:", (languages.pop()), "from", languages
```

Listing 4.46: 输出

```
---- output ----
set(['C', 'Java', 'Python', 'Data Science', 'Julia', 'SPSS', 'R', 'SAS',
'Machine Learning'])
set(['C', 'Java', 'Python', 'Data Science', 'Julia', 'SPSS', 'R', 'SAS',
'Machine Learning'])
Removed: C from set(['Java', 'Python', 'Data Science', 'Julia', 'SPSS', 'R',
'SAS', 'Machine Learning'])
```

集合操作

如前所述，集合允许我们使用数学集合操作，如并集、交集、差集和对称差集。我们可以在运算符或方法的帮助下实现这一点。

Set Union

两个集合 A 和 B 的联合将产生一个由两个集合的所有项目组成的集合。有两种方法来执行并集操作。1) 使用 | 操作符 2) 使用 union() 方法。

Listing 4.47: 集合操作的示例代码

```
Listing 1-33. Example code for set union operation
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
# use | operator
print "Union of A | B", A|B
# alternative we can use union()
A.union(B)
```

Listing 4.48: 输出

```
---- output ----
Union of A | B set([1, 2, 3, 4, 5, 6, 7, 8])
```

集合交叉点

两个集合 A 和 B 的相交将产生一个存在于或共同存在于两个集合中的项目集合。有两种方法来实现相交操作。1）使用 & 运算符 2）使用 intersection（）方法。

Listing 4.49: 集合交叉点示例代码

```
# use & operator
print "Intersection of A & B", A & B
# alternative we can use intersection()
print A.intersection(B)
```

Listing 4.50: 输出

```
---- output ----
Intersection of A & B set([4, 5])
```

集合差异

两个集合 A 和 B 的差值（即 A-B）将产生一个只存在于 A 中而不存在于 B 中的项目集合。有两种方法可以进行差值操作。1）使用'-' 运算符，和 2）使用 difference() 方法。

Listing 4.51: 集合差分操作的代码示例在 A 上使用-运算符

```
Listing 1-35. Example code for set difference operation # use - operator on A
print "Difference of A - B", A - B
# alternative we can use difference()
print A.difference(B)
```

Listing 4.52: 输出

```
---- output ----
Difference of A - B set([1, 2, 3])
```

集合对称性差异

两个集合 A 和 B 的对称差是指两个集合中不共同的项目的集合。有两种方法可以进行对称差分。1）使用□□□□2□□□$symmetric\_difference()$□□□

Listing 4.53: 设置对称性差分操作的示例代码 1

```
Listing 1-36. Example code for set symmetric difference operation
# use ^ operator
print "Symmetric difference of A ^ B", A ^ B
# alternative we can use symmetric_difference()
A.symmetric_difference(B)
```

Listing 4.54: 输出

```
---- output ----
Symmetric difference of A ^ B set([1, 2, 3, 6, 7, 8])
```

Listing 4.55: 设置对称性差分操作的示例代码 2

```
Basic Operations
Let's look at fundamental operations that can be performed on Python sets. See Listing 1-37.
Listing 1-37. Example code for basic operations on sets
# Return a shallow copy of a set
lang = languages.copy()
print languages
print lang
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
print A.isdisjoint(B) # True, when two sets have a null intersection
print A.issubset(B) # True, when another set contains this set
print A.issuperset(B) # True, when this set contains another set
sorted(B) # Return a new sorted list
print sum(A) # Retrun the sum of all items
print len(A) # Return the length
print min(A) # Return the largestitem
print max(A) # Return the smallest item
```

Listing 4.56: 输出

```
---- output ----
set(['C', 'Java', 'Python', 'Data Science', 'Julia', 'SPSS', 'AI', 'R',
'SAS', 'Machine Learning'])
set(['C', 'Java', 'Python', 'Data Science', 'Julia', 'SPSS', 'AI', 'R',
'SAS', 'Machine Learning'])
False
False
False
15
5
1
5
```

字典

Python 字典中的每一个项目都有一个 key 和 value 对，这是它的一部分。key 和 value 应该用大括号括起来。每个 key 和 value 用冒号 (:) 分隔，而每个项目则用逗号 (,) 分隔。注意，键在特定的字典中是唯一的，而且必须是不可改变的数据类型，如字符串、数字或图元，而值可以采取任何类型的重复数据。

Listing 4.57: 创建字典的示例代码创建字典

```
Listing 1-38. Example code for creating dictionary # Creating dictionary
dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
print "Sample dictionary: ", dict
```

Listing 4.58: 输出

```
---- output ----
Sample dictionary:  {'Age': 6, 'Name': 'Jivin', 'Class': 'First'}
Listing 1-39. Example code for accessing dictionary # Accessing items in dictionary
print "Value of key Name, from sample dictionary:", dict['Name']
---- output ----
Value of key Name, from sample dictionary: Jivin
```

Listing 4.59: 删除字典的例子

```
Listing 1-40. Example for deleting dictionary
# Deleting a dictionary
dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
print "Sample dictionary: ", dict
del dict['Name'] # Delete specific item
print "Sample dictionary post deletion of item Name:", dict
dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
dict.clear() # Clear all the contents of dictionary
print "dict post dict.clear():", dict
dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
del dict # Delete the dictionary
```

Listing 4.60: 输出

```
---- output ----
Sample dictionary:  {'Age': 6, 'Name': 'Jivin', 'Class': 'First'}
Sample dictionary post deletion of item Name: {'Age': 6, 'Class': 'First'}
dict post dict.clear(): {}
```

Listing 4.61: 更新字典的示例代码更新字典

```
Listing 1-41. Example code for updating dictionary # Updating dictionary
dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
print "Sample dictionary: ", dict
dict['Age'] = 6.5
print "Dictionary post age value update: ", dict
---- output ----
Sample dictionary:  {'Age': 6, 'Name': 'Jivin', 'Class': 'First'}
Dictionary post age value update:  {'Age': 6.5, 'Name': 'Jivin', 'Class':
'First'}
```

Listing 4.62: 对字典进行基本操作的示例代码基本操作

```
Listing 1-42. Example code for basic operations on dictionary # Basic operations
dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
print "Length of dict: ", len(dict)
dict1 = {'Name': 'Jivin', 'Age': 6};
dict2 = {'Name': 'Pratham', 'Age': 7};
dict3 = {'Name': 'Pranuth', 'Age': 7};
dict4 = {'Name': 'Jivin', 'Age': 6};
print "Return Value: dict1 vs dict2", cmp (dict1, dict2)
print "Return Value: dict2 vs dict3", cmp (dict2, dict3)
print "Return Value: dict1 vs dict4", cmp (dict1, dict4)
# String representation of dictionary
dict = {'Name': 'Jivin', 'Age': 6}
print "Equivalent String: ", str (dict)
# Copy the dict
dict1 = dict.copy()
print dict1
# Create new dictionary with keys from tuple and values to set value
seq = ('name', 'age', 'sex')
dict = dict.fromkeys(seq)
print "New Dictionary: ", str(dict)
dict = dict.fromkeys(seq, 10)
print "New Dictionary: ", str(dict)
# Retrieve value for a given key
dict = {'Name': 'Jivin', 'Age': 6};
print "Value for Age: ", dict.get('Age')
# Since the key Education does not exist, the second argument will be
returned
print "Value for Education: ", dict.get('Education', "First Grade")
# Check if key in dictionary
print "Age exists? ", dict.has_key('Age')
print "Sex exists? ", dict.has_key('Sex')
# Return items of dictionary
print "dict items: ", dict.items()
# Return items of keys
print "dict keys: ", dict.keys()
# return values of dict
print "Value of dict: ",  dict.values()
# if key does not exists, then the arguments will be added to dict and
returned
print "Value for Age : ", dict.setdefault('Age', None)
print "Value for Sex: ", dict.setdefault('Sex', None)
# Concatenate dicts
dict = {'Name': 'Jivin', 'Age': 6}
dict2 = {'Sex': 'male' }
```

```
dict.update(dict2)
print "dict.update(dict2) = ",  dict
```

Listing 4.63: 输出

```
---- output ----
Length of dict:  3
Return Value: dict1 vs dict2 -1
Return Value: dict2 vs dict3 1
Return Value: dict1 vs dict4 0
Equivalent String:  {'Age': 6, 'Name': 'Jivin'}
{'Age': 6, 'Name': 'Jivin'}
New Dictionary:  {'age': None, 'name': None, 'sex': None}
New Dictionary:  {'age': 10, 'name': 10, 'sex': 10}
Value for Age:  6
Value for Education:  First Grade
Age exists?  True
Sex exists?  False
dict items:  [('Age', 6), ('Name', 'Jivin')]
dict keys:  ['Age', 'Name']
Value for Age :  6
Value for Sex:  None
dict.update(dict2) =  {'Age': 6, 'Name': 'Jivin', 'Sex': 'male'}
Value of dict:  [6, 'Jivin', 'male']
```

4.4.7. 用户定义的函数

一个用户定义的函数是一个相关的代码语句块，它被组织起来以实现一个相关的动作。用户定义的函数概念的关键目标是鼓励模块化和实现代码的可重复使用。定义一个函数函数需要被定义，下面是在 Python 中定义一个函数需要遵循的一系列规则。- 关键字 def 表示一个函数块的开始，后面是函数的名称和开括号、闭括号。在这之后要放一个冒号 (:) 来表示函数头的结束。- 函数可以接受参数。任何这样的输入参数或参数都应放在参数头的括号内。- 主代码语句要放在函数头的下面，并且要缩进，这表示代码是同一个函数的一部分。- 函数可以向调用者返回一个表达式。如果在函数的末尾没有使用返回方法，它将作为一个子过程。函数和子程序之间的关键区别是，函数将总是返回表达式，而子程序则不会。

创建不带参数的函数的语法

语法如下：def functoin_name(): 1st block line 2nd block line ...

Listing 4.64: 创建不带参数的函数的语法示例

```
%Listing 1-43. Example code for creating functions without argument
# Simple function
def someFunction():
    print "Hello World"
# Call the function
someFunction()
```

Listing 4.65: 输出

```
----- output -----
Hello world
```

创建带参数的函数的语法

语法如下：def functoin$_name(parameters)$：$1stblockline2ndblockline...return[expression]$

Listing 4.66: 创建带参数的函数的代码示例

```
%Listing 1-44. Example code for creating functions with arguments
# Simple function to add two numbers
def sum_two_numbers(x, y):
return x + y
# after this line x will hold the value 3
print sum_two_numbers(1,2)
```

Listing 4.67: 输出

```
----- output -----
3
```

变量的范围

一个变量或标识符在程序执行中和执行后的可用性是由变量的作用域决定的。在 Python 中，有两个基本的变量作用域。1. 全局变量 2. 本地变量

请注意，Python 确实支持全局变量，而不需要你明确表示它们是全局变量。

Listing 4.68: 定义变量作用域的代码示例

```
Listing 1-45. Example code for defining variable scopes
# Global variable
x = 10
# Simple function to add two numbers
def sum_two_numbers(y):
return x + y
# Call the function and print result
print sum_two_numbers(10)
```

Listing 4.69: 输出

```
----- output -----
20
```

默认参数

你可以为函数的一个参数定义一个缺省值，这意味着如果在函数调用中没有为该参数提供任何值，函数将假定或使用缺省值。

Listing 4.70: 带有默认参数的函数的代码示例

```
Listing 1-46. Example code for function with default argument
# Simple function to add two number with b having default value of 10
def sum_two_numbers(x, y = 10):
return x + y
# Call the function and print result
print sum_two_numbers(10)
20
print sum_two_numbers(10, 5)
15
```

可变长度的参数

有些情况下，你在定义函数时不知道参数的确切数目，而希望能够动态地处理所有的参数。Python 对这种情况的回答是可变长度参数，它使我们能够处理比你在定义函数时指定的更多的参数。*args 和 **kwargs 是一个常见的习惯，允许动态的参数数量。

*args 将以一个元组的形式提供所有的函数参数

See Listing 1-47.

Listing 4.71: 以 *args 形式传递参数的示例代码

```
%Listing 1-47. Example code for passing argumens as *args
# Simple function to loop through arguments and print them
def sample_function(*args):
    for a in args:
        print a
# Call the function
Sample_function(1,2,3)
1
2
3
```

**kwargs 将使你有能力处理你没有事先定义的命名参数或关键字参数关键词

Listing 4.72: 将参数作为 ** 空项传递的代码示例

```
%Listing 1-48. Example code for passing argumens as **kwargs
# Simple function to loop through arguments and print them
def sample_function(**kwargs):
    for a in kwargs:
        print a, kwargs[a]
# Call the function
sample_function(name='John', age=27)
age 27
name 'John'
```

模块

一个模块是一个逻辑上有组织的多个独立但相关的代码或功能或类的集合。创建模块的关键原则是它更容易理解、使用，并具有高效的可维护性。你可以导入一个模块，Python 解释器将按以下顺序搜索感兴趣的模块。1. 目前直接活动的，也就是调用 Python 你的程序的目录。2. 如果在当前活动的目录中没有找到该模块，Python 将在路径变量 PYTHONPATH 中搜索每个目录。如果搜索失败，则在默认的软件包安装路径中搜索。

注意，模块的搜索路径以 sys.path 变量的形式保存在系统模块中，它包含了当前目录、PYTHONPATH 和依赖安装的默认路径。当你导入一个模块时，它只被加载一次，不管它被导入多少次。你也可以从你的模块中导入特定的元素（函数、类等）到当前命名空间。

Listing 4.73: 导入模块的示例代码

```
%Listing 1-49. Example code for importing modules
# Import all functions from a module
import module_name
from modname import*
# Import specific function from module
from module_name import function_name
```

Python 有一个被称为名字空间的内部字典，它将每个变量或标识符的名字作为键来存储，它们的对应值是各自的 Python 对象。有两种类型的名字空间，本地和全局。本地命名空间在 Python 程序的执行过程中被创建，以保存所有由程序创建的对象。本地变量和全局变量有相同的名字，本地变量对全局变量有阴影。每个类和函数都有自己的局部命名空间。Python 假定任何在函数中赋值的变量都是局部的。对于全局变量，你需要明确地指定它们。

另一个关键的内置函数是 dir()，运行这个函数将返回一个排序的字符串列表，其中包含一个模块中定义的所有模块、变量和函数的名称。见清单 1-50。

Listing 4.74: 示例代码 dir() 操作导入 os

```
Listing 1-50. Example code dir() operation Import os
content = dir(os)
print content
```

Listing 4.75: 输出

```
---- output ----
['F_OK', 'O_APPEND', 'O_BINARY', 'O_CREAT', 'O_EXCL', 'O_NOINHERIT',
'O_RANDOM', 'O_RDONLY', 'O_RDWR', 'O_SEQUENTIAL', 'O_SHORT_LIVED',
'O_TEMPORARY', 'O_TEXT', 'O_TRUNC', 'O_WRONLY', 'P_DETACH', 'P_NOWAIT',
'P_NOWAITO', 'P_OVERLAY', 'P_WAIT', 'R_OK', 'SEEK_CUR', 'SEEK_END', 'SEEK_
SET', 'TMP_MAX', 'UserDict', 'W_OK', 'X_OK', '_Environ', '__all__', '__
builtins__', '__doc__', '__file__', '__name__', '__package__', '_copy_reg',
'_execvpe', '_exists', '_exit', '_get_exports_list', '_make_stat_result',
'_make_statvfs_result', '_pickle_stat_result', '_pickle_statvfs_result',
'abort', 'access', 'altsep', 'chdir', 'chmod', 'close', 'closerange',
'curdir', 'defpath', 'devnull', 'dup', 'dup2', 'environ', 'errno', 'error',
'execl', 'execle', 'execlp', 'execlpe', 'execv', 'execve', 'execvp','execvpe', 'extsep', 'fdopen',
    'fstat', 'fsync', 'getcwd', 'getcwdu',
'getenv', 'getpid', 'isatty', 'kill', 'linesep', 'listdir', 'lseek',
'lstat', 'makedirs', 'mkdir', 'name', 'open', 'pardir', 'path', 'pathsep',
'pipe', 'popen', 'popen2', 'popen3', 'popen4', 'putenv', 'read', 'remove',
'removedirs', 'rename', 'renames', 'rmdir', 'sep', 'spawnl', 'spawnle',
'spawnv', 'spawnve', 'startfile', 'stat', 'stat_float_times', 'stat_
result', 'statvfs_result', 'strerror', 'sys', 'system', 'tempnam', 'times',
'tmpfile', 'tmpnam', 'umask', 'unlink', 'unsetenv', 'urandom', 'utime',
'waitpid', 'walk', 'write']
```

看一下上面的输出，$_{n}ame_{\square}{}_{f}ile_{\square\square\square\square\square\square\square\square\square\square}$

文件输入/输出

Python 提供了简单的函数来读取和写入文件信息。为了对文件进行读写操作，我们需要首先打开它们。一旦所需的操作完成，就需要关闭它，以便释放与该文件绑定的所有资源。

打开文件

在打开一个文件时，access_mode 将决定文件的打开模式，即读、写、追加等。读取（r）模式是默认的文件访问模式，这是一个可选参数。

Listing 4.76: 下面的代码将创建一个名为车辆的文件，并添加项目。
n 是一个换行符

```
%# Below code will create a file named vehicles and add the items. \n is a newline character
vehicles = ['scooter\n', 'bike\n', 'car\n']
f = open('vehicles.txt', 'w')
```

```
f.writelines(vehicles)
# Reading from file
f = open('vechicles.txt')
print f.readlines()
f.close()
```

Listing 4.77: 输出

```
---- output ----
['scooter\n', 'bike\n', 'car\n']
```

异常处理

任何在 Python 程序执行过程中发生的、会中断程序预期流程的错误都被称为异常。你的程序应该被设计成可以处理预期的和意外的错误。

Python 有丰富的内置异常集，当你的程序出错时，它可以强迫你的程序输出一个错误。

你可以在你的 Python 程序中使用 try、raise、except 和 final 语句来处理异常。try 和 except：try 子句可以用来在你的程序中放置任何可以引发异常的关键操作，异常子句应该有处理引发的异常的代码。

Listing 4.78: 异常处理的示例代码

```
%Listing 1-52. Example code for exception handling
try: x= 1 y= 1
    print "Result of x/y: ", x / y
except (ZeroDivisionError):
    print("Can not divide by zero")
except (TypeError):
    print("Wrong data type, division is allowed on numeric data type only")
except:
    print "Unexpected error occurred", '\n', "Error Type: ", sys.exc_info()
[0], '\n', "Error Msg: ", sys.exc_info()[1]
```

Listing 4.79: 输出

```
---- output ----
Result of x/y:  1
```

注 1）将上述代码中 b 的值改为 0，将打印出” 不能除以 0 ” 的语句。2) 在除法语句中用’a’ 替换’a’，将打印以下输出。发生了意外的错误错误类型。< 类型’exceptions.nameerror’>。错误信息：名称’a’ 未被定义最后：这是一个可选的子句，目的是定义在任何情况下都必须执行的清理动作。

Listing 4.80: 文件操作的异常处理示例代码

```
%Listing 1-53. Example code for exception handling with file operations
# Below code will open a file and try to convert the content to integer
try:
    f = open('vechicles.txt')
    print f.readline()
    i = int(s.strip())
except IOError as e:
    print "I/O error({0}): {1}".format(e.errno, e.strerror)
except ValueError:
    print "Could not convert data to an integer."
except:
```

```
12      print "Unexpected error occurred", '\n', "Error Type: ", sys.exc_info()
13      [0], '\n', "Error Msg: ", sys.exc_info()[1]
14  finally:
15      f.close()
16      print "file has been closed"
```

Listing 4.81: 输出

```
17  ---- output ----
18  scooter
19  Could not convert data to an integer.
20  file has been closed
```

Python 总是在离开 try 语句之前执行 final 子句，而不管是否有异常发生。如果在 try 子句中出现了没有被设计用来处理的异常，那么在执行了 final 子句之后，同样的异常会被重新提出来。如果使用诸如 break、continue 或 return 等语句迫使程序退出 try 子句，仍然会在离开时执行 finally。参见图4.2。

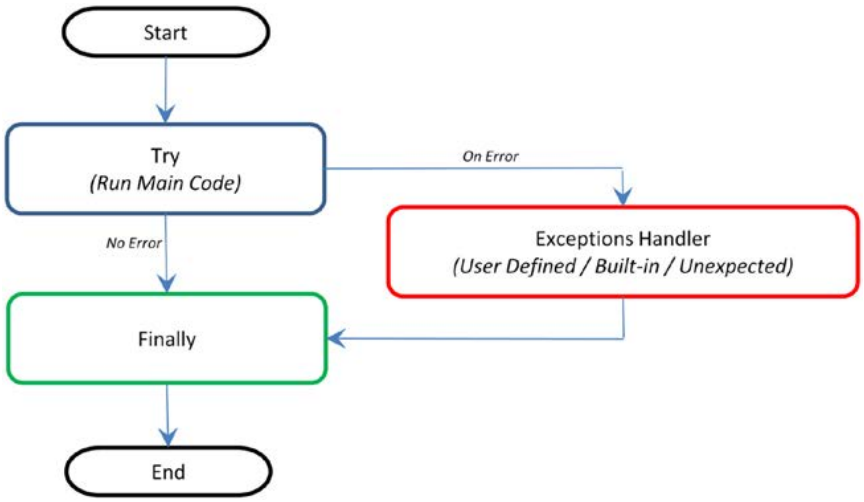

图 **4.2:** Code Flow for Error Handler

注意，一般来说，最好的做法是通过使用 finally 来遵循单一的退出点原则。这意味着在你的主代码成功执行后，或者你的错误处理程序处理完一个错误后，它应该通过 finally，这样代码在任何情况下都会在同一个点退出。

4.5. 总结

到目前为止，我已经尽力涵盖了让你开始学习 Python 的基础知识和基本主题，而且有大量的在线/离线资源可以用来增加你对 Python 这种编程语言的知识深度。同样，我想给你留下一些有用的资源供你将来参考。

4.6. 参考文献

http://docs.python-guide.org/en/latest/intro/learning/

http://awesome-python.com/