如果你让一个 Python 程序员说一下 Python 的优势，他会说简洁以及高可读是最有影响力的优势。为证明上述两点，在这个 Python 教程里，我们将聊聊许多基本的 Python 建议和技巧。

我们从开始使用 Python 便收集这些有用的捷径（贴士与技巧）。分享一些我们知道，同时又能造福于人的知识，有什么事情比这更棒吗？

过去我们分享过 [一些给初学者的 Python 编程贴士](http://www.techbeamers.com/top-10-python-coding-tips-for-beginners/) 旨在优化代码并且减少编码工作，我们的读者仍旧很乐意阅读它。

所以今天我们带来另一些基本的 Python 贴士与技巧，所有的这些技巧都能帮助你压缩代码并且优化运行。此外，你可以在日常工作中很容易地在真实项目中使用他们。

每一个技巧都有一个示例并且给出简短的解释，如果要测试这些代码片段，你可以看一下这些 [在线虚拟 Python 运行终端](http://www.techbeamers.com/top-7-online-virtual-terminal-for-python-coding-practice/)。

最近我们发布的另两个必看的 Python 资源：

💡 [9 中优化 Python 代码的主要方式](http://www.techbeamers.com/python-code-optimization-tips-tricks/)

💡 [发现在编程时应该避免的最常见的错误](http://www.techbeamers.com/python-programming-mistakes/)

使用下面的 TOC 来快速地浏览这些 Python 贴士与技巧。

**给程序员的 30 个基本 Python 贴士与技巧**

**贴士#1. 原地交换两个数字**

Python 提供了一个直观的在一行代码中赋值与交换（变量值）的方法，请参见下面的示例：



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | x, y = 10, 20  print(x, y)    x, y = y, x  print(x, y)    #1 (10, 20)  #2 (20, 10) |

赋值的右侧形成了一个新的元组，左侧立即解析（unpack）那个（未被引用的）元组到变量 <a> 和 <b>。

一旦赋值完成，新的元组变成了未被引用状态并且被标记为可被垃圾回收，最终也完成了变量的交换。

**贴士#2. 链状比较操作符**

比较操作符的聚合是另一个有时很方便的技巧：



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | n = 10  result = 1 < n < 20  print(result)    # True    result = 1 > n <= 9  print(result)    # False |

**贴士#3. 使用三元操作符来进行条件赋值**

三元操作符是 if-else 语句也就是条件操作符的一个快捷方式：



|  |  |
| --- | --- |
| 1 | [表达式为真的返回值] if [表达式] else [表达式为假的返回值] |

这里给出几个你可以用来使代码紧凑简洁的例子。下面的语句是说“如果 y 是 9，给 x 赋值 10，不然赋值为 20”。如果需要的话我们也可以延长这条操作链。



|  |  |
| --- | --- |
| 1 | x = 10 if (y == 9) else 20 |

同样地，我们可以对类做这种操作：



|  |  |
| --- | --- |
| 1 | x = (classA if y == 1 else classB)(param1, param2) |

在上面的例子里 classA 与 classB 是两个类，其中一个类的构造函数会被调用。

下面是另一个多个条件表达式链接起来用以计算最小值的例子：



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | def small(a, b, c):      return a if a <= b and a <= c else (b if b <= a and b <= c else c)    print(small(1, 0, 1))  print(small(1, 2, 2))  print(small(2, 2, 3))  print(small(5, 4, 3))    #Output  #0 #1 #2 #3 |

我们甚至可以在列表推导中使用三元运算符：



|  |  |
| --- | --- |
| 1  2  3 | [m\*\*2 if m > 10 else m\*\*4 for m in range(50)]    #=> [0, 1, 16, 81, 256, 625, 1296, 2401, 4096, 6561, 10000, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 784, 841, 900, 961, 1024, 1089, 1156, 1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764, 1849, 1936, 2025, 2116, 2209, 2304, 2401] |

**贴士#4. 多行字符串**

基本的方式是使用源于 C 语言的反斜杠：



|  |  |
| --- | --- |
| 1  2  3  4  5 | multiStr = "select \* from multi\_row  where row\_id < 5"  print(multiStr)    # select \* from multi\_row where row\_id < 5 |

另一个技巧是使用三引号：



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | multiStr = """select \* from multi\_row  where row\_id < 5"""  print(multiStr)    #select \* from multi\_row  #where row\_id < 5 |

上面方法共有的问题是缺少合适的缩进，如果我们尝试缩进会在字符串中插入空格。所以最后的解决方案是将字符串分为多行并且将整个字符串包含在括号中：



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | multiStr= ("select \* from multi\_row "  "where row\_id < 5 "  "order by age")  print(multiStr)    #select \* from multi\_row where row\_id < 5 order by age |

**贴士#5. 存储列表元素到新的变量中**

我们可以使用列表来初始化多个变量，在解析列表时，变量的数目不应该超过列表中的元素个数：【译者注：元素个数与列表长度应该严格相同，不然会报错】



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | testList = [1,2,3]  x, y, z = testList    print(x, y, z)    #-> 1 2 3 |

**贴士#6. 打印引入模块的文件路径**

如果你想知道引用到代码中模块的绝对路径，可以使用下面的技巧：



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | import threading  import socket    print(threading)  print(socket)    #1- <module 'threading' from '/usr/lib/python2.7/threading.py'>  #2- <module 'socket' from '/usr/lib/python2.7/socket.py'> |

**贴士#7. 交互环境下的 “\_” 操作符**

这是一个我们大多数人不知道的有用特性，在 Python 控制台，不论何时我们测试一个表达式或者调用一个方法，结果都会分配给一个临时变量： \_（一个下划线）。



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | >>> 2 + 1  3  >>> \_  3  >>> print \_  3 |

“\_” 是上一个执行的表达式的输出。

**贴士#8. 字典/集合推导**

与我们使用的列表推导相似，我们也可以使用字典/集合推导，它们使用起来简单且有效，下面是一个例子：



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | testDict = {i: i \* i for i in xrange(10)}  testSet = {i \* 2 for i in xrange(10)}    print(testSet)  print(testDict)    #set([0, 2, 4, 6, 8, 10, 12, 14, 16, 18])  #{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81} |

注：两个语句中只有一个 <:> 的不同，另，在 Python3 中运行上述代码时，将 <xrange>改为 <range>。

**贴士#9. 调试脚本**

我们可以在 <pdb> 模块的帮助下在 Python 脚本中设置断点，下面是一个例子：



|  |  |
| --- | --- |
| 1  2 | import pdb  pdb.set\_trace() |

我们可以在脚本中任何位置指定 <pdb.set\_trace()> 并且在那里设置一个断点，相当简便。

**贴士#10. 开启文件分享**

Python 允许运行一个 HTTP 服务器来从根路径共享文件，下面是开启服务器的命令：

**# Python 2**



|  |  |
| --- | --- |
| 1 | python -m SimpleHTTPServer |

**# Python 3**



|  |  |
| --- | --- |
| 1 | python3 -m http.server |

上面的命令会在默认端口也就是 8000 开启一个服务器，你可以将一个自定义的端口号以最后一个参数的方式传递到上面的命令中。

**贴士#11. 检查 Python 中的对象**

我们可以通过调用 dir() 方法来检查 Python 中的对象，下面是一个简单的例子：



|  |  |
| --- | --- |
| 1  2 | test = [1, 3, 5, 7]  print( dir(test) ) |



|  |  |
| --- | --- |
| 1 | ['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_delslice\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getslice\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_iadd\_\_', '\_\_imul\_\_', '\_\_init\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_reversed\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_setslice\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort'] |

**贴士#12. 简化 if 语句**

我们可以使用下面的方式来验证多个值：



|  |  |
| --- | --- |
| 1 | if m in [1,3,5,7]: |

而不是：



|  |  |
| --- | --- |
| 1 | if m==1 or m==3 or m==5 or m==7: |

或者，对于 in 操作符我们也可以使用 '{1,3,5,7}' 而不是 '[1,3,5,7]'，因为 set 中取元素是 O(1) 操作。

**贴士#13. 运行时检测 Python 版本**

当正在运行的 Python 低于支持的版本时，有时我们也许不想运行我们的程序。为达到这个目标，你可以使用下面的代码片段，它也以可读的方式输出当前 Python 版本：



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | import sys    #Detect the Python version currently in use.  if not hasattr(sys, "hexversion") or sys.hexversion != 50660080:      print("Sorry, you aren't running on Python 3.5n")      print("Please upgrade to 3.5.n")      sys.exit(1)    #Print Python version in a readable format.  print("Current Python version: ", sys.version) |

或者你可以使用 sys.version\_info >= (3, 5) 来替换上面代码中的 sys.hexversion != 50660080，这是一个读者的建议。

在 Python 2.7 上运行的结果：

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | Python 2.7.10 (default, Jul 14 2015, 19:46:27)  [GCC 4.8.2] on linux    Sorry, you aren't running on Python 3.5    Please upgrade to 3.5. |

在 Python 3.5 上运行的结果：

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Python 3.5.1 (default, Dec 2015, 13:05:11)  [GCC 4.8.2] on linux    Current Python version:  3.5.2 (default, Aug 22 2016, 21:11:05)  [GCC 5.3.0] |

**贴士#14. 组合多个字符串**

如果你想拼接列表中的所有记号，比如下面的例子：



|  |  |
| --- | --- |
| 1 | >>> test = ['I', 'Like', 'Python', 'automation'] |

现在，让我们从上面给出的列表元素新建一个字符串：



|  |  |
| --- | --- |
| 1 | >>> print ''.join(test) |

**贴士#15. 四种翻转字符串/列表的方式**

**# 翻转列表本身**



|  |  |
| --- | --- |
| 1  2  3  4  5 | testList = [1, 3, 5]  testList.reverse()  print(testList)    #-> [5, 3, 1] |

**# 在一个循环中翻转并迭代输出**



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | for element in reversed([1,3,5]):      print(element)    #1-> 5  #2-> 3  #3-> 1 |

**# 一行代码翻转字符串**



|  |  |
| --- | --- |
| 1 | "Test Python"[::-1] |

输出为 “nohtyP tseT”

**# 使用切片翻转列表**



|  |  |
| --- | --- |
| 1 | [1, 3, 5][::-1] |

上面的命令将会给出输出 [5,3,1]。

**贴士#16. 玩转枚举**

使用枚举可以在循环中方便地找到（当前的）索引：



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | testlist = [10, 20, 30]  for i, value in enumerate(testlist):      print(i, ': ', value)    #1-> 0 : 10  #2-> 1 : 20  #3-> 2 : 30 |

**贴士#17. 在 Python 中使用枚举量**

我们可以使用下面的方式来定义枚举量：



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | class Shapes:      Circle, Square, Triangle, Quadrangle = range(4)    print(Shapes.Circle)  print(Shapes.Square)  print(Shapes.Triangle)  print(Shapes.Quadrangle)    #1-> 0  #2-> 1  #3-> 2  #4-> 3 |

**贴士#18. 从方法中返回多个值**

并没有太多编程语言支持这个特性，然而 Python 中的方法确实（可以）返回多个值，请参见下面的例子来看看这是如何工作的：



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | # function returning multiple values.  def x():      return 1, 2, 3, 4    # Calling the above function.  a, b, c, d = x()    print(a, b, c, d)    #-> 1 2 3 4 |

**贴士#19. 使用 \* 运算符（splat operator）来 unpack 函数参数**

\* 运算符（splat operator）提供了一个艺术化的方法来 unpack 参数列表，为清楚起见请参见下面的例子：



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | def test(x, y, z):      print(x, y, z)    testDict = {'x': 1, 'y': 2, 'z': 3}  testList = [10, 20, 30]    test(\*testDict)  test(\*\*testDict)  test(\*testList)    #1-> x y z  #2-> 1 2 3  #3-> 10 20 30 |

**贴士#20. 使用字典来存储选择操作**

我们能构造一个字典来存储表达式：



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | stdcalc = {      'sum': lambda x, y: x + y,      'subtract': lambda x, y: x - y  }    print(stdcalc['sum'](9,3))  print(stdcalc['subtract'](9,3))    #1-> 12  #2-> 6 |

**贴士#21. 一行代码计算任何数的阶乘**

**Python 2.x.**



|  |  |
| --- | --- |
| 1  2  3  4 | result = (lambda k: reduce(int.\_\_mul\_\_, range(1,k+1),1))(3)  print(result)    #-> 6 |

**Python 3.x.**



|  |  |
| --- | --- |
| 1  2  3  4  5 | import functools  result = (lambda k: functools.reduce(int.\_\_mul\_\_, range(1,k+1),1))(3)  print(result)    #-> 6 |

**贴士#22. 找到列表中出现最频繁的数**



|  |  |
| --- | --- |
| 1  2  3  4 | test = [1,2,3,4,2,2,3,1,4,4,4]  print(max(set(test), key=test.count))    #-> 4 |

**贴士#23. 重置递归限制**

Python 限制递归次数到 1000，我们可以重置这个值：



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | import sys    x=1001  print(sys.getrecursionlimit())    sys.setrecursionlimit(x)  print(sys.getrecursionlimit())    #1-> 1000  #2-> 1001 |

请只在必要的时候采用上面的技巧。

**贴士#24. 检查一个对象的内存使用**

在 Python 2.7 中，一个 32 比特的整数占用 24 字节，在 Python 3.5 中利用 28 字节。为确定内存使用，我们可以调用 getsizeof 方法：

**在 Python 2.7 中**



|  |  |
| --- | --- |
| 1  2  3  4  5 | import sys  x=1  print(sys.getsizeof(x))    #-> 24 |

**在 Python 3.5 中**



|  |  |
| --- | --- |
| 1  2  3  4  5 | import sys  x=1  print(sys.getsizeof(x))    #-> 28 |

**贴士#25. 使用 \_\_slots\_\_ 来减少内存开支**

你是否注意到你的 Python 应用占用许多资源特别是内存？有一个技巧是使用 \_\_slots\_\_ 类变量来在一定程度上减少内存开支。



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | import sys  class FileSystem(object):        def \_\_init\_\_(self, files, folders, devices):          self.files = files          self.folders = folders          self.devices = devices  print(sys.getsizeof( FileSystem ))    class FileSystem1(object):        \_\_slots\_\_ = ['files', 'folders', 'devices']      def \_\_init\_\_(self, files, folders, devices):          self.files = files          self.folders = folders          self.devices = devices    print(sys.getsizeof( FileSystem1 ))  #In Python 3.5  #1-> 1016  #2-> 888 |

很明显，你可以从结果中看到确实有内存使用上的节省，但是你只应该在一个类的内存开销不必要得大时才使用 \_\_slots\_\_。只在对应用进行性能分析后才使用它，不然地话，你只是使得代码难以改变而没有真正的益处。

【译者注：在我的 win10 python2.7 中上面的结果是：



|  |  |
| --- | --- |
| 1  2  3 | #In Python 2.7 win10  #1-> 896  #2-> 1016 |

所以，这种比较方式是不那么让人信服的，使用 \_\_slots\_\_ 主要是用以限定对象的属性信息，另外，当生成对象很多时花销可能会小一些，具体可以参见 [python 官方文档](https://docs.python.org/3/reference/datamodel.html?highlight=__slots__#object.__slots__):

The **slots** declaration takes a sequence of instance variables and reserves just enough space in each instance to hold a value for each variable. Space is saved because **dict** is not created for each instance. 】

**贴士#26. 使用 lambda 来模仿输出方法**



|  |  |
| --- | --- |
| 1  2  3  4  5 | import sys  lprint=lambda \*args:sys.stdout.write(" ".join(map(str,args)))  lprint("python", "tips",1000,1001)    #-> python tips 1000 1001 |

**贴士#27. 从两个相关的序列构建一个字典**



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | t1 = (1, 2, 3)  t2 = (10, 20, 30)    print(dict (zip(t1,t2)))    #-> {1: 10, 2: 20, 3: 30} |

**贴士#28. 一行代码搜索字符串的多个前后缀**



|  |  |
| --- | --- |
| 1  2  3  4  5 | print("http://www.google.com".startswith(("http://", "https://")))  print("http://www.google.co.uk".endswith((".com", ".co.uk")))    #1-> True  #2-> True |

**贴士#29. 不使用循环构造一个列表**



|  |  |
| --- | --- |
| 1  2  3  4  5 | import itertools  test = [[-1, -2], [30, 40], [25, 35]]  print(list(itertools.chain.from\_iterable(test)))    #-> [-1, -2, 30, 40, 25, 35] |

**贴士#30. 在 Python 中实现一个真正的 switch-case 语句**

下面的代码使用一个字典来模拟构造一个 switch-case。



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | def xswitch(x):      return xswitch.\_system\_dict.get(x, None)    xswitch.\_system\_dict = {'files': 10, 'folders': 5, 'devices': 2}    print(xswitch('default'))  print(xswitch('devices'))    #1-> None  #2-> 2 |

学习资料或者学习班咨询若兰老师QQ:2463598866