Python 代码压缩指南

By *灰尘疾客*

2023.07.02

前言

因一偶念[1]，让我突然想去压缩 Python 代码（像 html，js，css 一样）。我尝试搜之于茫茫网络，但并网上并没有很多相关资料。或许那些 pythonors 更喜欢 pythonic 风格的编码方式吧！但是这影响不了我的 **min.py** 计划。

为何要压缩 Python 代码？

答曰：因为没人做，所以我想做！

注意，所有的这些操作都多少有点违反*《Python 之禅》*及其它编写规范。此外，代码可读性也会显著降低。如果不是为了方便储存，我绝对不建议这么做，因为这对后续维护影响很大；而若你想给那些只会说『多说无益，放「码」过来』的伸手党一个『惊喜』，那么我建议在此之外再加一层代码混淆，一切只为整蛊。

如果你认为它有用，请~~熟读并背诵~~，把它**分享**给别人，给更多 Pythonors，至少它是为数不多关于手动压缩 Python 代码的教程。

如果你有任何疑问，可以通过邮箱联系我：gkcoll@111.com。

主体

现在我将分级展示一些压缩 Python 代码的技巧。

总共包括这些骚操作：

- **删除注释和空行**
- **重构**
- **隐藏的语法特性**（*分号*）

建议操作顺序：根据本文从上到下从头到尾。

一、删除注释和空行

难度：★☆☆☆☆

在程序员交接工作时，有注释是最好的，这样方便接收者快速适应工作。但如果不需要工作交接，而且工作已经完结（或者想在别人伸手要代码的时候搞事情），那么直接删除掉所有注释和空行，可以大幅度压缩你的代码行数。

找到所有井号（#）后的内容或三引号（''' 或 """）之间的内容，连着符号一起把它们删除，再删除所有空行。如你所见，你的代码也已经压缩得七七八八了，您的代码看起来已经非常紧凑，可读性也已显著降低。

二、重构

在一些关于 Python 的教程书里，重构都是保证代码可读性和减少后续维护成本的很重要的方式。

但在这一部分，重构有一个更重要的作用：**抽象**。

（一）合并导入语句

难度：★☆☆☆☆

当我们进行 Python 编程时，我们常常需要导入各种包。就像这样：

```
from time import sleep as s
from time import time as t
import requests
import json
from langid import classify as c
```

在很多时候，导入表达式都可以被压缩：

```
from time import sleep as s, time as t
import requests, json
from langid import classify as c
```

虽然这看起来好像不够短，但是如果所有 import 语句都形如 `import 模块`（或 `import 模块 as 别名`），则它们都可以被压缩成一行。

```
import time, requests, json, langid
```

（二）合并赋值表达式

难度：☆☆☆☆☆

这个方法应该在课堂中或自学中都有所了解和熟练使用：

```
a = 1
b = 2
c = 3
```

直接写成 `a, b, c = 1, 2, 3`。

（三）合并 with 语句[2]

难度：☆☆☆☆☆

当你看到这种结构：

```
with a:
    with b:
        pass
```

很好，直接替换成 `with a,b: pass` 就行了。

（四）修改变量名

难度：★★☆☆☆

忽略意义不讲，为了保证代码压缩程度，建议把超长变量名替换成毫无可读性的 `a,b,c`。

（五）优化 For 循环结构

1. 生成器

难度：★★★★★

在这一部分，我希望你有一些关于生成器（Generator）的知识储备。

就像下方示例代码：

```
lst = list()
for i in range(1, 6):
    lst.append(i)
```

我们可以试试这样写（运行效果一致）：

```
lst = [i for i in range(1, 6)]
```

上面的示例或许影响不明显，但是在某些情况，一个生成器可以大幅度压缩你的代码。

举个夸张的例子（一个特别的计算字符串长度的函数，与内置 len 函数不同的是它会识别中文字符并将其视为 2 个单位长度）：

```
def l(s: str) -> int:
    return sum([2 if '\u4e00' <= i <= '\u9fff' else 1 for i in s])
```

2. 笛卡尔积[3]

难度：★★★☆☆

有时，我们可能需要同时遍历多个可迭代对象，这可能会导致令人头皮发麻的 for 循环嵌套。

对于涉及遍历同一类或类似功能的多个对象，使用 for 循环嵌套往往是多余的。此外，对于像 Python 这样不需要 for 循环退出声明的语言，因为没有类似 JS 的三角法则，代码也可能非常难看。

就像这样：

```
# 非原创
phone = ['iPhone', 'HuaWei', 'Mi']
number = [1, 2, 3]
color = ['White', 'Black']
for p in phone:
    for n in number:
        for c in color:
            print(f'{p}{n}{c}', end=' ')
```

在给出更多建议前，让我们先了解一个数学知识——笛卡尔积。

笛卡尔积（Cartesian product）是指数学中将两个集合 X 和 Y 中的对象组合，第一个对象是 X 中的成员而第二个对象是 Y 中的成员，组合完所有可能的有序对。笛卡尔积又称为直积，表示为 X×Y 。例如，假设集合 X={a, b} ，集合 Y={0, 1, 2} ，则两个集合的笛卡尔积为 {(a, 0), (a, 1), (a, 2), (b, 0), (b, 1), (b, 2)} 。

在Python的内置模块 ~~functools~~ itertools 中，提供了高阶类 product() ，用于实现多个可迭代对象（列表、字符串等）中元素的组合，返回可迭代对象中元素组合的笛卡尔积，效果相当于嵌套的循环。为了更直观地看出效果，下面用代码对比来看 product 的效果。

```
# 调用 `product` 实现元素组合，替代 for 循环
import itertools
for p, n, c in itertools.product(phone, number, color):
    print(f'{p}{n}{c}', end=' ')
```

运行结果是一样的：

```
iPhone1White iPhone1Black iPhone2White iPhone2Black iPhone3White
iPhone3Black HuaWei1White HuaWei1Black HuaWei2White HuaWei2Black
HuaWei3White HuaWei3Black Mi1White Mi1Black Mi2White Mi2Black Mi3White
Mi3Black RedMi1White RedMi1Black RedMi2White RedMi2Black RedMi3White
RedMi3Black
```

可以看到，实现相同的功能，for 循环使用了三层嵌套，而 product() 只需要一层循环。

便利自然不用说，主要是这有助于我们进一步压缩 Python 代码。至于其他高级用法，请自行参考相关资料。

（六）优化函数定义

难度：★★★★☆

在很多时候，定义一个 Python 函数需要：

```
def func():
    """一个冤种函数。"""
    # 做些动作...
    pass
```

如果你是个高级 Python 程序员，你应该知道 lambda 表达式[4]。它是创建**匿名函数**的一种方式。它只包含函数主体，没有函数名、定义标识符和返回关键字。事实上，lambda 函数可以像 def 定义的函数一样传递。

就像这个一次函数：def f(x): return 2*x + 1，如果函数只会在一个地方被调用，那么你可以在那将其替换为 lambda x: 2*x + 1。

举个栗子：

```
def calc(func, x):
    """
    传入一个函数和一个参数 x，
    返回由 `func` 计算的结果。
    """
    return func(x)

y = calc(lambda x: 2*x + 1, 2)
print(y)  # 5
```

更高级些，这是一个用于计算十六位 MD5 值的函数：

```
import hashlib

def md5(content):
    hash = hashlib.md5(content.encode('utf-8'))
    return hash.hexdigest()
```

压缩后，我们可以使用 lambda 函数，一行代码实现定义：

```
md5 = lambda s: __import__("hashlib").md5(s.encode('utf-8')).hexdigest()
```

注意

根据 PEP8[5] 官方说明，并不建议这样定义 lambda 函数。

- Always use a def statement instead of an assignment statement that binds a lambda expression directly to an identifier:

```
# Correct:
def f(x): return 2*x
```

```
# Wrong:
f = lambda x: 2*x
```

The first form means that the name of the resulting function object is specifically 'f' instead of the generic '<lambda>'. This is more useful for tracebacks and string representations in general. The use of the assignment statement eliminates the sole benefit a lambda expression can offer over an explicit def statement (i.e. that it can be embedded inside a larger expression)

因为这不仅有失 lambda 函数的匿名性（示例中将其赋值给了变量，也就是有了名），也使其难以维护，因为当出现错误时，解释器显示的错误是不完整的，并且不能追溯到函数（名），它只会告诉你这是由 lambda 引起的。

体现其**不够灵活，可读性差，不适合复杂的逻辑**[6]。故，非必要不要滥用，尤其是对于正在维护的项目。

因此，如果您确信您编写的函数在整个代码文件中只使用一次，请确保只在需要的地方使用 lambda 函数。例如上面的代码，您可以直接在需要的位置传递它，而无需为将其赋值给变量：

```
print((lambda s: __import__("hashlib").md5(s.encode('utf-8')).hexdigest())
(input('>>>')))
```

在其它情况下，尽管是定义一个仅包含一条表达式的函数，但是只要该函数会在不同地方被调用多次，我就建议你继续使用 def 关键字（def func():pass）。

整蛊别人固然爽，但别以治人之道还治己人之身。

三、隐藏的语法特性

难度：★★★☆☆

众所周知，Python 利用缩进以确保代码可读性。我们也知道很多情况下，缩进错误会影响运行结果或成功操作。但是这不是大问题，我们的代码仍需压缩，而 Python 也确实提供了一个让我们可以像 JavaScript 一样压缩代码的方法。这就是分号（;）。

与其他语言不同，在一行**表达式**后使用分号运行 Python 是没有意义的。在 Python 中，分号仅用于一行中的拆分表达式。

举个栗子：

```
print('你好世界！');print('极客藏源万岁！')
```

我们升级一下难度：这是一个演示如何读取文件并逐行打印内容的示例代码：

```
with open('test.txt', 'r', encoding='utf-8') as f:
    for line in f:
        print(f)
```

我猜你应该知道接下来该怎么做了，但是你可能很快就会得到一个 SyntaxError（语法错误）。

现在，让我们压缩它：

```
with open('test.txt', 'r', encoding='utf-8') as f:for line in f:print(f)
```

跑一下……好吧，预期中的 BUG 出现了。

在解决问题之前，你应该注意到上面文本中最接近这个的粗体字。对的，是**表达式**。我没有告诉你**控制流**不能用这种方式压缩。

事实上，所有的控制流（包括 **if-elif-else** 结构、for 循环、while 循环、**with** 结构和 **try-except-finally** 结构）都不能通过此方法直接压缩。除非它们都在一行之首。

所以示例应该这么改：

```
with open('test.txt', 'r', encoding='utf-8') as f:print("\n".join(l for l
in f)
# 或
f=open('test.txt', 'r', encoding='utf-8');print("\n".join(l for l in
f));f.close()
```

总结一下，就是

- 所有**同级**表达式都可以通过分号直接合并；
- 当遇到循环等控制流结构时，需要**换行**（确保它们在一行之首），控制流内的表达式**像上面一样添加**，控制流外的表达式需要**换行**；
- 如果存在控制流结构**嵌套**，则请保持原来的**缩进**原则。

难以理解？看看这个示例：

```
if x>y:x+=y;print(y);
else:y+=x;print(x);
```

尾声

一、练一练

难度：★★★☆☆

在完美的结束之前，作者我想给大家留道题：

这是一个用于将输入的英文文本处理为标题化文本的程序，请压缩它：

```
import nltk # 自然语言工具包
import string

CAPITALIZE_TAGS = set(['NN', 'PRP', 'VB', 'RB', 'JJ', 'WDT', 'WP', 'WRB'])
SPECIAL_SITUATION_TAGS = set(["IN", "UH", "CC", "DT", "TO"])

def titleize(obj: str) -> str:
    """
    核心函数：标题化传入的内容
    :param obj: 待处理的文字。
    """
    words = nltk.pos_tag(nltk.word_tokenize(obj))
    newwords = []
    for i, word in enumerate(words):
        if i == 0 or i == len(words) - 1 or word[0][-1] in
string.punctuation:
            newwords.append(word[0].capitalize())
        elif word[1] in CAPITALIZE_TAGS:
            newwords.append(word[0].capitalize())
        else:
            if word[1] in SPECIAL_SITUATION_TAGS and len(word[0]) < 5:
                newwords.append(word[0])
            else:
                newwords.append(word[0].capitalize())
    return " ".join(newwords)
```

其实有一点我上面说漏了，那就是在压缩代码之前，你可以尝试重新仔细阅读代码，看看有没有什么逻辑可以被优化的，这对压缩效果也很重要。

就像上面函数中 for 循环的内容显得多余一样，因为它可以直接这样写：

```
def titleize(obj: str) -> str:
    """
    核心函数: 标题化传入的内容
    :param obj: 待处理的文字。
    """
    words = nltk.pos_tag(nltk.word_tokenize(obj))
    newwords = []

    for word in words:
        if word[1] in SPECIAL_SITUATION_TAGS and len(word[0]) < 5:
            newwords.append(word[0])
        else:
            newwords.append(word[0].capitalize())

    return " ".join(newwords)
```

然后你会注意到一些细节——变量 CAPITALIZE_TAGS 和模块 string 现在无用了，我们可以直接把它们删除。另外，类型为 set （集合）的变量——SPECIAL_SITUATION_TAGS 其实也不是很有用，因为毕竟都是要被遍历的（直接用元组或列表就行了）。

结合全文，我们大概可以将它改成这样：

```
from nltk import pos_tag as a, word_tokenize as b;s,l=["IN", "UH", "CC",
"DT", "TO"],len  # 第一行
def t(obj):return " ".join([w[0] if w[1] in s and l(w[0]) < 5 else
w[0].title() for w in a(b(obj))])  # 第二行
```

二、结语

现在，通过阅读本文献，你已经学习了所有压缩 Python 代码的技能。但是事实上，作者我还是一个新手——一个已经入门三年的新手！所以如果你发现有任何不合理的内容或明显的错误，请千万提出，我将学习并逐一修改。如果有任何其它相关建议，你也可以贡献，让我们一起维护好这份**革命性**的文献！

[1] performance - Can Python code be compressed like Javascript? - Stack Overflow ↩

[2] 参阅文献：mnfy — minify/obfuscate Python 3 source code — mnfy 33.0.0 documentation ↩

[3] 参阅文献：详解 Python 中的排列组合生成器 ↩

[4] 参阅 Lambda 函数用法总结 - 知乎 和 Python Lambda 函数深度总结 - 稀土掘金 ↩

[5] 详情请见：https://pep8.org/#programming-recommendations ↩

[6] 参阅文献：Python 中 lambda 表达式的优缺点及使用场景 - 51CTO 博客 ↩