Искусство написания циклов на Python

[Блог компании VDSina.ru](https://habr.com/ru/company/vdsina/blog/)[Python](https://habr.com/ru/hub/python/)[Программирование](https://habr.com/ru/hub/programming/)

Перевод

[Автор оригинала: Yang Zhou](https://medium.com/techtofreedom/the-art-of-writing-loops-in-python-68e9869e4ed4)

Цикл for — самый базовый инструмент потока управления большинства языков программирования. Например, простой цикл for на C выглядит так:

**int** i;

**for** (i=0;i<N;i++)

{

//do something

}

Не существует более изящного способа написания цикла for на C. В сложных случаях обычно приходится писать уродливые вложенные циклы или задавать множество вспомогательных переменных (например, как i в показанном выше коде).  
  
К счастью, в Python всё более удобно. В этом языке есть множество хитростей, позволяющих писать более изящные циклы, которые упрощают нашу жизнь. В Python вполне можно избежать вложенных циклов и вспомогательных переменных, и мы даже можем самостоятельно настраивать цикл for.  
  
Эта статья познакомит вас с самыми полезными трюками по написанию циклов на Python. Надеюсь, она поможет вам ощутить красоту этого языка.

Одновременно получаем значения и индексы

Частым примером использования цикла for является получение индексов и значений из списка. Когда я начинал изучать Python, то писал свой код таким образом:

**for** i **in** range(len(my\_list)):

print(i, my\_list[i])

Разумеется, он работал. Но это решение не в стиле Python. Спустя несколько месяцев я узнал стандартный способ реализации в стиле Python:

**for** i, v **in** enumerate(my\_list):

print(i, v)

Как мы видим, встроенная функция enumerate упрощает нам жизнь.

Как избежать вложенных циклов с помощью функции Product

Вложенные циклы — это настоящая головная боль. Они могут снизить читаемость кода и усложнить его понимание. Например, [прерывание вложенных циклов](https://medium.com/techtofreedom/5-ways-to-break-out-of-nested-loops-in-python-4c505d34ace7) обычно реализовать не так просто. Нам нужно знать, где прерван самый внутренний цикл, второй по порядку внутренний цикл, и так далее.  
  
К счастью, в Python существует потрясающая функция product из встроенного модуля itertools. Мы можем использовать её, чтобы не писать множество вложенных циклов.  
  
Давайте убедимся в её полезности на простом примере:

list\_a = [1, 2020, 70]

list\_b = [2, 4, 7, 2000]

list\_c = [3, 70, 7]

**for** a **in** list\_a:

**for** b **in** list\_b:

**for** c **in** list\_c:

**if** a + b + c == 2077:

print(a, b, c)

# 70 2000 7

Как мы видим, нам требуется три вложенных цикла, чтобы получить из трёх списков три числа, сумма которых равна 2077. Код не очень красив.  
  
А теперь давайте попробуем использовать функцию product.

**from** itertools **import** product

list\_a = [1, 2020, 70]

list\_b = [2, 4, 7, 2000]

list\_c = [3, 70, 7]

**for** a, b, c **in** product(list\_a, list\_b, list\_c):

**if** a + b + c == 2077:

print(a, b, c)

# 70 2000 7

Как мы видим, благодаря использованию функции product достаточно всего одного цикла.  
  
Так как функция product генерирует прямое произведение входящих итерируемых данных, она позволяет нам во многих случаях избежать вложенных циклов.

Используем модуль Itertools для написания красивых циклов

На самом деле, функция product — это только вершина айсберга. Если вы изучите встроенный модуль Python itertools, то перед вами откроется целый новый мир. Этот набор инструментов содержит множество полезных методов, покрывающих наши потребности при работе с циклами. Их полный список можно найти в [официальной документации](https://docs.python.org/3/library/itertools.html). Давайте рассмотрим несколько примеров.

Создаём бесконечный цикл

Существует не меньше трёх способов создания бесконечного цикла:  
  
1. При помощи функции count:

natural\_num = itertools.count(1)

**for** n **in** natural\_num:

print(n)

# 1,2,3,...

2. Функцией cycle:

many\_yang = itertools.cycle('Yang')

**for** y **in** many\_yang:

print(y)

# 'Y','a','n','g','Y','a','n','g',...

3. Через функцию repeat:

many\_yang = itertools.repeat('Yang')

**for** y **in** many\_yang:

print(y)

# 'Yang','Yang',...

Комбинируем несколько итераторов в один

Функция chain() позволяет объединить несколько [итераторов](https://medium.com/techtofreedom/iterable-and-iterator-in-python-dbe7011d1ff7) в один.

**from** itertools **import** chain

list\_a = [1, 22]

list\_b = [7, 20]

list\_c = [3, 70]

**for** i **in** chain(list\_a, list\_b, list\_c):

print(i)

# 1,22,7,20,3,70

Выделяем соседние дублирующиеся элементы

Функция groupby используется для выделения соседних дублирующихся элементов в итераторе и их соединения.

**from** itertools **import** groupby

**for** key, group **in** groupby('YAaANNGGG'):

print(key, list(group))

# Y ['Y']

# A ['A']

# a ['a']

# A ['A']

# N ['N', 'N']

# G ['G', 'G', 'G']

Как показано выше, соседние одинаковые символы соединены вместе. Более того, мы можем указать функции groupby способ определения идентичности двух элементов:

**from** itertools **import** groupby

**for** key, group **in** groupby('YAaANNGGG', **lambda** x: x.upper()):

print(key, list(group))

# Y ['Y']

# A ['A', 'a', 'A']

# N ['N', 'N']

# G ['G', 'G', 'G']

Самостоятельно настраиваем цикл

Изучив представленные выше примеры, давайте задумаемся о том, почему циклы for в Python настолько гибки и изящны. Насколько я понимаю, это из-за того, что мы можем применять в итераторе цикла for функции. Во всех рассмотренных выше примерах в итераторе всего лишь используются специальные функции. Все трюки имеют одинаковый шаблон:

**for** x **in** function(iterator)

Сам встроенный модуль itertools всего лишь реализует за нас самые распространённые функции. Если мы забудем функцию или не сможем найти нужную нам, то можем просто написать её самостоятельно. Если быть более точным, то эти функции являются [генераторами](https://medium.com/techtofreedom/generators-in-python-f0e59784b3b7). Именно поэтому мы можем генерировать с их помощью бесконечные циклы.  
  
По сути, мы можем настроить цикл for под себя, как сделали бы это с настраиваемым генератором.  
  
Давайте рассмотрим простой пример:

**def** **even\_only**(num):

**for** i **in** num:

**if** i % 2 == 0:

**yield** i

my\_list = [1, 9, 3, 4, 2, 5]

**for** n **in** even\_only(my\_list):

print(n)

# 4

# 2

Как видно из приведённого выше примера, мы определили генератор под названием even\_only. Если мы используем этот генератор в цикле for, итерация будет происходить только для чётных чисел из списка.  
  
Разумеется, этот пример приведён только для объяснения. Существуют и другие способы выполнения тех же действий, например, использование [представления списков](https://medium.com/techtofreedom/8-levels-of-using-list-comprehension-in-python-efc3c339a1f0).

my\_list = [1, 9, 3, 4, 2, 5]

**for** n **in** (i **for** i **in** my\_list **if** **not** i % 2):

print(n)

# 4

# 2

Вывод

Задачу создания циклов на Python можно решать очень гибко и изящно. Чтобы писать удобные и простые циклы, мы можем использовать встроенные инструменты или даже самостоятельно определять генераторы.

На правах рекламы

[Надёжный сервер в аренду](https://vdsina.ru/cloud-servers?partner=habr412), создавайте свою конфигурацию в пару кликов и начинайте работать уже через минуту. Всё будет работать без сбоев и с очень высоким uptime!  
  
Присоединяйтесь к [нашему чату в Telegram](https://t.me/vdsina).