**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

**БРЯНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**Кафедра «Информатика и программное обеспечение»**

**РАСЧЁТНО–ГРАФИЧЕСКАЯ РАБОТА**

**по дисциплине «Введение в современные информационные технологии»**

**Тема: «Модуль Python itertools»**

Студент гр. О – 21 – ИВТ2 – ПО – Б

Бахтин Г. А.

№ зач. книжки 21.0285

Преподаватель

Подвесовская М.А.

**БРЯНСК 2022**

Содержание

[Введение 3](#_Toc104908804)

[1. Функции 4](#_Toc104908805)

[1.1. Бесконечные итераторы 4](#_Toc104908806)

[1.1.1. Пример 1 4](#_Toc104908807)

[1.2. Итераторы, завершающиеся на кратчайшей входной последовательности 5](#_Toc104908808)

[1.2.1. Функция accumulate 6](#_Toc104908809)

[1.2.2. Функция chain 6](#_Toc104908810)

[1.2.3. Функция chain.from\_iterable 7](#_Toc104908811)

[1.3. Комбинационные итераторы 7](#_Toc104908812)

[1.3.1. Пример 1 7](#_Toc104908813)

[1.3.2. Пример 2 8](#_Toc104908814)

[1.3.3. Пример 3 9](#_Toc104908815)

[1.3.4. Пример 4 10](#_Toc104908816)

[Заключение 13](#_Toc104908817)

[Список Литературы 14](#_Toc104908818)

# Введение

Основанный на итераторах код обеспечивает лучшие характеристики использования памяти, чем код, основанный на использовании списков. Поскольку итераторы не возвращает данные до тех пор, пока они не потребуются, отпадает необходимость в хранении всего набора данных в памяти. Такая модель отложенной обработки сглаживает отрицательное влияние подкачки и других побочных эффектов, связанных с обработкой больших объёмов данных, на производительность.

itertools – стандартный модуль Python, включающий в себя ряд функций, предназначенных для обработки последовательностей.

itertools относят к функциональным модулям, ведь он был вдохновлён такими языками программирования, как APL, Haskell, SML.

# Функции

## Бесконечные итераторы

Бесконечные итераторы – бесконечные последовательности, которые имеют паттерн (см. табл.таблица 1).

Таблица 1

Бесконечные итераторы

|  |  |  |  |
| --- | --- | --- | --- |
| **Итератор** | **Аргументы** | **Результат** | **Пример** |
| count() | start, [step] | start, start+step, start+2\*step, … | count(10) ––> 10 11 12 13 14 ... |
| cycle() | p | p0, p1, … plast, p0, p1, … | cycle('строка') ––> с т р о к а с т р о к а ... |
| repeat() | elem [,n] | elem, elem, elem, … бесконечно или столько сколько указано в количестве | repeat(10, 3) ––> 10 10 10 |

### Пример 1

count используется, когда необходим вечный цикл со знанием количествa пройденных итераций.

Ниже представлено сравнение скорости работы скрипта с count() и без него.

Без count() скорость выполнения составит 0.0007404560001305072 секунды (см. листинг 1).

Листинг 1

Пример без count

i = 0  
while True:  
 if i == 200:  
 break  
 print(i)  
 i += 1

С count() скорость выполнения составит 0.0006238849996407225 секунды (см. листинг 2).

Листинг 2

Пример с count

import itertools  
for i in itertools.count():  
 if i == 200:  
 break  
 print(i)

## Итераторы, завершающиеся на кратчайшей входной последовательности

Ниже представлены итератора, завершающиеся на кратчайшей входной последовательности (см. табл.таблица 2).

Таблица 2

Итераторы, завершающиеся на кратчайшей входной последовательности

|  |  |  |  |
| --- | --- | --- | --- |
| **Итератор** | **Аргументы** | **Результат** | **Пример** |
| accumulate() | p [,func] | p0, p0+p1, p0+p1+p2, … | accumulate([1, 2, 3, 4, 5]) ––> 1 3 6 10 15 |
| chain() | p, …, q | p0, p1, … plast, q0, q1, … | chain('abc', 'абв') ––> a b c а б в |
| chain.from\_iterable() | iterable | p0, p1, … plast, q0, q1, … | chain.from\_iterable(['abc', 'абв']) ––> a b c а б в |
| compress() | data, selectors | (d[0] если s[0]), (d[1] если s[1]), … | compress('компр', [1, 0, 1]) ––> к м |
| dropwhile() | pred, seq | seq[n], seq[n+1], начинается при ложном предикате | dropwhile(lambda x: x<5), [1, 4, 7, 8, 9]) ––> 7, 8, 9 |
| filterfalse() | pred, seq | элементы последовательности, где предикат ложь | filterfalse(lambda x: x%2, range(10)) ––> 0 2 4 6 8 |
| takewhile() | pred, seq | seq[0], seq[1]? пока предикат ложь | takewhile(lambda x: x<5, [1, 4, 5]) ––> 1 4 |
| groupby() | iterable[, key] | подитераторы, сгруппированные по значению ключа |  |
| islice() | seq, [start,] stop [, step] | элементы последовательности[начало:конец:шаг] | islice('ABCDEFG', 2, None) ––> C D E F G |
| pairwise() | iterable | (p[0], p[1]), (p[2], p[3]), … | pairwise('мишка') ––> ми иш шк ка |
| starmap() | func, seq | func(*seq[0]), func(*seq[1]), … | starmap(pow, [(2, 5), (3, 2)]) ––> 32 9 |
| tee() | it, n | it1, it2, … tin splits one iter |  |
| zpi\_longest() | p, q, … | (p[0], q[0]), (p[1]), (p[1], q[1]), … | itertools.zip\_longest('abcd', 'xy', fillvalue) ––> ax by c– d– |

### Функция accumulate

accumulate – очень полезная функция. Разберёмся, как можно её использовать в наших программах.

Для начала нужно импортировать обозреваемый модуль itertools. Затем инициализируем список с нашими значениям, например: l = [1, 3, 4, 5, 10]. Теперь применим функцию к списку: itertools.accumulate(l). Результат действия: <itertools.accumulate object at 0x00000176CD> – accumulate объект, с ним можно взаимодействовать как с итератором или как с последовательностью (для этого нужно преобразовать объект).

См. листинг 3 – пример вышесказанного.

Листинг 3

Пример с функцией accumulate

from itertools import accumulate  
l = [1, 3, 4, 5, 10]  
print(tuple(accumulate(l))) # (1, 4, 8, 13, 23)  
print(accumulate(l)) # <itertools.accumulate object at 0x00000176CD>  
a = accumulate(l)  
print(next(a)) # 1  
print(next(a)) # 4

### Функция chain

chain ещё одна полезная функция, из названия можно определить её суть. Занесённые строки в качестве аргументов chain, будут соединены между собой как цепи (см. листинг 4).

Листинг 4

Пример с функцией chain

from itertools import chain  
print(list(chain('це', 'пь'))) # ['ц', 'е', 'п', 'ь']

### Функция chain.from\_iterable

chain.from\_iterable выполняет то же самое что и chain, но только для таких последовательностей как: списки и кортежи (см. листинг 5).

Листинг 5

Пример с функцией chain.from\_iterable

from itertools import chain  
matrix = [[1, 2, 3,], [3, 2, 1]]  
print(list(chain.from\_iterable(matrix))) # [1, 2, 3, 3, 2, 1]

## Комбинационные итераторы

Комбинационные итераторы – последовательности, которые перебирают комбинации (см. таблица 3).

Таблица 3

Комбинационные итераторы

|  |  |  |
| --- | --- | --- |
| **Итератор** | **Аргументы** | **Результат** |
| product() | p, q, … [repeat=1] | декартово произведение, эквивалентное вложенному циклу for |
| permutations() | p, [r] | кортеж длиной r, все возможные порядки без повторяющихся элементов, позиция важна |
| combinations\_with\_replacement() | p, r | кортеж длиной r, все возможные порядки c повторяющимися элементами, позиция не важна |

### Пример 1

Ниже приведена таблица с примерами комбинационных итераторов.

Таблица 4

Примеры комбинационных итераторов

|  |  |
| --- | --- |
| **Пример** | **Результат** |
| product(‘abcd’, repeat=2) | aa ab ac ad ba bb bc bd ca cb cc cd da db dc dd |
| permutations(‘abcd’, 2) | ab ac ad ba bc bd ca cb cd da db dc |
| combinations(‘abcd’, 2) | ab ac ad bc bd cd |
| combinations\_with\_replacement(‘abcd’, 2) | aa ab ac ad bb bc bd cc cd dd |

### Пример 2

Получить последовательность, состоящую из последовательностей, в которых первый элемент – масть карты, второй элемент её достоинство.

Решение (см. листинг 6):

Листинг 6

Пример использования product

import itertools  
  
dignities = [\*map(str, range(1, 11)), 'валет', 'дама', 'король', 'туз']  
suits = ['пики', 'трефы', 'черви', 'бубны']  
  
print(\*itertools.product(suits, dignities))  
  
"""  
Результат:  
 ('пики', '1') ('пики', '2') ('пики', '3')  
 ('пики', '4') ('пики', '5') ('пики', '6')  
 ('пики', '7') ('пики', '8') ('пики', '9')  
 ('пики', '10') ('пики', 'валет') ('пики', 'дама')  
 ('пики', 'король') ('пики', 'туз') ('трефы', '1')  
 ...  
 ('бубны', 'дама') ('бубны', 'король') ('бубны', 'туз')  
"""

\*map(str, range(1, 11)) – это выражение означает, что мы создаём диапазон чисел от 1 до 10, затем используем функцию map, которая преобразовывает все элементы последовательности согласно правилу, которое задаётся первым аргументом. Если мы добавим это в наш список, мы получим map объект, поэтому мы используем \* – оператор распаковки. В данном случае произойдёт распаковка элементов map объекта.

itertools.product(suits, dignities) – это itertools объект, который осуществляет произведение элементов друг над другом. Можно провести аналогию с таблицей умножения: у нас есть числа от 1 до 10 по горизонтали и вертикали, применяя product, мы получим таблицу умножения. Важно понимать, что, например, при пересечении 3 и 2 мы получим не 6, а последовательность (3, 2).

### Пример 3

Решим задание 8 задание ЕГЭ, используя разбираемый модуль.

Задание: (№ 3538) (Е. Джобс) Все 4–буквенные слова, составленные из букв П, Р, В, Д, А, записаны в алфавитном порядке и пронумерованы. Вот начало списка:

1. АААА  
2. АААВ  
3. АААД  
4. АААП  
5. АААР  
6. ААВА  
...

Найдите номер первого слова в этом списке, которое не содержит гласных и одинаковых букв.

Решение (см. листинг 7):

Листинг 7

Пример использования combinations

import itertools  
  
seq = 'авдпр'  
without\_repeat = tuple(itertools.combinations(seq, 4))  
  
for idx, elem in enumerate(itertools.product(seq, repeat=4), 1):  
 if 'а' not in elem and elem in without\_repeat:  
 print(idx, elem)  
 break

Результат: 195 ('в', 'д', 'п', 'р').

Для задачи требуется объявить переменные seq и without\_repeat. авдпр записывается в том же порядке, что и идут последние буквы с 1 по 5 слов. Выражением tuple(itertools.combinations(seq, 4)) создаются комбинации, где исключаются повторы букв.

itertools.product(seq, repeat=4) перемножает буквы между собой, получая слово с длиной равной 4.

enumerate(itertools.product(seq, repeat=4), 1) – итератор, возвращающий два элемента: индекс и значение. Индекс начинается со значения, которое задаётся вторым аргументом, то есть у нас с 1.

Заключительным шагом является итерирование enumerate объекта и проверка условия, если оно выполняется, то выводим в терминал индекс и значение элемента. Дальше обрываем цикл.

### Пример 4

Задание: сформировать таблицу истинности, исходя из введённых значений пользователем, и загрузить её в csv файл.

Решение:

Импортируем необходимые библиотеки: itertools, csv.

Инициализируем переменные и булеву функцию.

variables = 'a b c d e'  
func = 'not(a + b) \* (c + e) <= d \* b'

Здесь variables задаётся как строка для того, чтобы, если нужно будет, использовать input, и тем самым предоставить пользователю возможность задовать переменные. Тоже самое относится и к func.

Открываем запись файла, задав при этом символ новой строки (это важно, так как MS Excel может прочитать стандартный newline символ не так, как нам бы этого хотелось).

with open('test.csv', 'w', newline='\n') as csvfile:

Инициализируем переменную, к которой присвоим csv.writer, с аргументом рабочего файла.

writer = csv.writer(csvfile)

Записываем переменные и обозначение функции в первую строку файла.

writer.writerow(variables.replace(' ', '') + 'f')

Результатом variables.replace(' ', '') будет являться строка abcd, к которой мы прибавляем f. Так как строка – последовательность, то каждый её символ будет записан в отдельную ячейку.

Итерируем элементы itertools.product, аргументами которого 01 – правда и ложь и количество повторов, которое считается длиной строки variables без пробелов.

for row in itertools.product('01', repeat=len(variables.replace(' ', '')):

Выполняем строку f'{variables.replace(" ", ", ")} = map(int, row)', как выражение Питона с помощью функции exec. Другими словами мы выполняем a, b, c, d, e = 0, 0, 0, 0, 0.

exec(f'{variables.replace(" ", ", ")} = map(int, row)')

Теперь остаётся только записать строку, как список, куда мы кладём значения строки: 0, 0, 0, 0, 0 с помощью [\*row] и прибавляем значение функции с помощью тернарного оператора ['1' if eval(func) else '0'] – если выполнение выражения eval(func) равно 1, то кладём в список '1', иначе '0'.

writer.writerow([\*row] + ['1' if eval(func) else '0'])

Ниже представлен описанный код (см. листинг 8).

Листинг 8

Программа реализующая запись таблицы истинности по формуле

import itertools  
import csv  
  
variables = 'a b c d e'  
func = 'not(a + b) \* (c + e) <= d \* b'  
  
with open('test.csv', 'w', newline='\n') as csvfile:  
 writer = csv.writer(csvfile)  
 writer.writerow(variables.replace(' ', '') + 'f')  
  
 for row in itertools.product('01', repeat=len(variables.replace(' ', '')):  
 exec(f'{variables.replace(" ", ", ")} = map(int, row)')  
 writer.writerow([\*row] + ['1' if eval(func) else '0'])

Результат в качестве csv файла представлен на рис. 1.

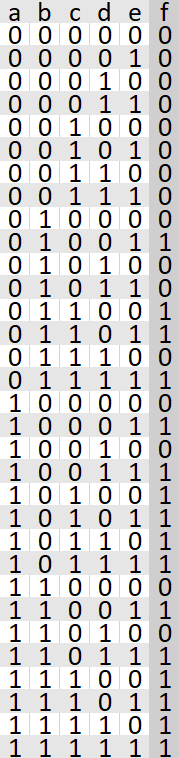


Рис. 1. Результат программы в листинге 15

# Заключение

Мы познакомились с замечательным, очень мощным Python модулем itertools, который позволяет упростить реализации некоторых алгоритмов. Знание этого модуля выводит код программиста на средний уровень, так как зачастую код становится более понятным и читаемым, увеличивая при этом скорость.

В реферате мы рассмотрели самые полезные и чаще встречающиеся функции этого модуля и написали практический скрипт.

# Список Литературы

1. Официальный сайт Python. [Электронный ресурс]. – Режим доступа: <https://www.python.org>. (Дата обращения 15.05.2022).
2. Веб–сервис «kpolyakov.spb.ru». [Электронный ресурс]. – Режим доступа: https://kpolyakov.spb.ru/school/ege/generate.htm. (Дата обращения 15.05.2022).