# Циклы

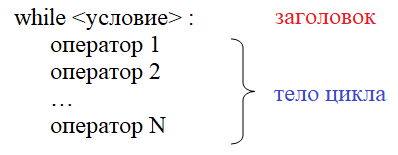
## Понятие итерации

**Итерация** — отдельное повторение однотипного действия, под которым понимается блок выполнения цикла («тела цикла»).

## Цикл while

Ни одна сколь-нибудь серьезная программа на Python не обходится без циклов.

на уровне языка Python выполняет оператор цикла while, имеющий следующий синтаксис:



Смотрите как в Python записывается группа операторов (тело цикла): вся группа должна иметь один и тот же сдвиг относительно оператора while Причем такой отступ строго обязателен – это элемент синтаксиса python. И благодаря этому текст программы становится наглядным и хорошо читаемым. Это, безусловно, один из плюсов данного языка.

Однократное выполнение тела цикла называется **итерацией**. То есть, может быть первая итерация, вторая итерация, N-я итерация и так далее.

Давайте в качестве примера с помощью оператора цикла while вычислим вот такую вот сумму: Расписывать все это через тысячу слагаемых не очень то удобно. И к тому же число слагаемых может зависеть от значения переменной и быть неопределенным.



В таких задачах без циклов не обойтись.

И программа будет выглядеть так:

S=0; i=1

**while** i <= 1000:

    S += 1/i

    i += 1

**print**(S)

В качестве выражения в цикле while можно писать все те же самые условия, что и в условном операторе if. Например, можно вычислять сумму S пока либо i<=1000, либо S < 5. Такое условие запишется так:

**while** i <= 1000 **and** S < 5:

здесь цикл будет работать пока i<=1000 и S<5 как только одно из подусловий станет ложным, все составное условие становится ложным и цикл завершит свою работу.

проверка условия завершения происходит только на начальной отметке, то есть, спортсмен должен добежать круг целиком и только потом проверить: прошел час или нет.

Другими словами, пока целиком не выполнится текущая итерация тела цикла, оператор while продолжает свою работу. И как только условие цикла становится ложным, цикл завершает свою работу.

А что будет, если условие в цикле while будет истинным всегда? В этом случае мы получим «вечный» цикл, программа фактически зависнет и наш спортсмен будет обречен на бесконечный бег по кругу.

S=0; i=1

**while** 1 : S += 1

**print**(S)

Далее, цикл while может иметь необязательный блок else, который идет после цикла:

Это, вроде как естественный выход из оператора цикла. И здесь часто возникает вопрос: а чем блок else отличается от блока операторов, просто идущих после блока while? Ведь когда цикл while завершится, мы так и так перейдем к последующим операторам! Однако, тут есть один нюанс. Любой оператор цикла в Python может быть досрочно прерван с помощью оператора

**break**

Как только он встречается в теле цикла, цикл (в данном случае while) завершает свою работу. То есть, при досрочном прерывании работы цикла while, конструкция else не выполняется и управление переходит на последующие операторы. Вот в чем отличие блока else от операторов, стоящих непосредственно после while. Например:

z = 0  
v = -10  
  
while v < 100:  
 if v == 0:  
 break  
 z += 1/v  
 v = v + 1  
else:  
 print("Сумма вычислена корретно")  
print(z)

Если здесь при вычислении суммы ожидается деление на 0, то срабатывает break и цикл досрочно прерывается. В этом случае блок else не срабатывает и мы не видим сообщения, что сумма вычислена корректно. Если же все проходит штатно (без вызова break), то в консоли появляется сообщение - Сумма вычислена корректно - означающее выполнение блока else.

**continue**

Этот оператор позволяет пропускать тело цикла и перейти к следующей итерации, не прерывая работу самого цикла. Например, мы хотим перебрать все целые значения от -4 до 4, исключая значение 0. Такую программу можно реализовать так:

a = 0  
b = -5  
  
while b < 4:  
 b = b + 1  
 if b == 0:  
 continue  
 print(b)  
 a += 1/b  
print(a)

При выполнении этой программы увидим, что в консоль выведены все значения кроме нуля. Так как при i=0 срабатывает условие и выполняется оператор continue. Все что находится после этого оператора пропускается и цикл продолжается уже со значением i=1.

Вот так работают эти два управляющих оператора break и continue, которые можно использовать во всех операторах циклов.

## Один

## Вводятся два целых положительных числа n и m, причем, n < m

## Вывести в строку через пробел квадраты целых чисел в диапазоне [n; m]. Программу реализовать при помощи цикла while.

## Два

Вводится стоимость одной книги x рублей (вещественное число).

Необходимо вывести на экран в строчку через пробел стоимости 2, 3, ... 10 таких книг.

Программу реализовать при помощи цикла while.

**Sample Input:**

34.6

**Sample Output:**

69.2 103.8 138.4 173.0 207.6 242.2 276.8 311.4 346.0

## Три

## Одноклеточная амеба каждые 3 часа делится на 2 клетки.

## Определить, сколько клеток будет через n часов (n - целое положительное число, вводимое с клавиатуры).

## Считать, что изначально была одна амеба.

## Результат вывести на экран.

## Задачу необходимо решить с использованием цикла while.

## Четыре

## Составить программу поиска всех трехзначных чисел, которые при делении на 47 дают в остатке 43 и кратны 3.

## Вывести найденные числа в строчку через пробел.

## Программу реализовать при помощи цикла while.

**Sample Output:**

231 372 513 654 795 936

Ежемесячная стипендия студента составляет educational\_grant руб., а расходы на проживание превышают стипендию и составляют expenses руб. в месяц. Рост цен ежемесячно увеличивает расходы на 3%, кроме первого месяца.

Составьте программу расчета суммы денег, которую необходимо единовременно попросить у родителей, чтобы можно было прожить учебный год (10 месяцев), используя только эти деньги и стипендию. 37566.55 рублей.

ed, exp = 10000, 12000  
  
m = 1  
d = exp - ed  
while m < 10:  
 m += 1  
 d += exp - ed  
 exp \*= 1.03  
d = round (d, 2)  
print ('Студенту надо попросить', d, 'рублей.')

## Цикл for

Следующий и, наверное, самый часто используемый оператор цикла – это оператор for, который имеет такой синтаксис:

Например,

**for** x **in** 1,5,2,4:

**print**(x\*\*2)

выведет в консоль квадраты соответствующих чисел. Но что, если мы хотим перебрать значения по порядку в соответствии с правилом: начальное значение, шаг, конечное значение

Для этого используется генератор последовательностей range(start, stop, step)

Например, если мы запишем его вот так:

**for** x **in** range(1,5,1):

**print**(x)

то в консоли увидим числа от 1 до 4 с шагом 1. То есть, range генерирует последовательность в интервале[1;5]

Последнее значение не входит в интервал. Если в нашем примере поставить шаг отрицательный -1, то конечное значение 5 не может быть достигнуто и в этом случае Python возвратит пустую последовательность:

**for** x **in** range(1,5,-1):

**print**(x)

Если нам нужны числа от 5 до 1, то следует записывать range в таком виде:

**for** x **in** range(5,0,-1):

**print**(x)

Причем, в range можно записывать только целые числа, с вещественными он не работает.

Давайте перепишем нашу программу подсчета суммы



с помощью цикла for, получим:

S=0

**for** i **in** range(1, 1001, 1):

    S += 1/i

**print**(S)

Здесь весь цикл записан буквально в одну строчку, а тело цикла состоит из одного оператора – подсчета суммы ряда.

Вторым примером рассмотрим задачу вычисления значений линейной функции



Программа будет выглядеть так:

k = 0.5; b = 2

lst = [0, 0.1, 0.2, 0.3, 0.4, 0.5]

**for** x **in** lst:

**print**(x\*k+b)

Этот пример показывает, что для перебора значений счетчика x можно использовать списки, сформированные ранее в программе. (О списках мы подробнее будем говорить на последующих занятиях).

Циклы типа «for», в отличие от «while», повторяются не в зависимости от выполнения условия, а для каждого элемента в списке, множества, кортежа или другой совокупности элементов. Другими словами, для каждой итерации цикла используется один из элементов совокупности. К примеру, чтобы проверить пригодность продуктов питания в магазине, человек должен по очереди взять каждый продукт из своей корзины и убедиться, что срок годности еще не истек. Так, корзина с продуктами выступает в качестве совокупности элементов, продукты являются самими элементами, а проверка срока годности одного продукта — итерацией цикла.

В качестве совокупности элементов может выступать строка, поскольку она является совокупностью символов. Для примера попробуем вывести все элементы строки отдельно:

|  |
| --- |
|  |
| line = input("Enter some string: ") |
| for c in line: |
| print(c) |
|  |

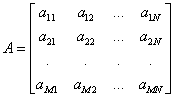
## 

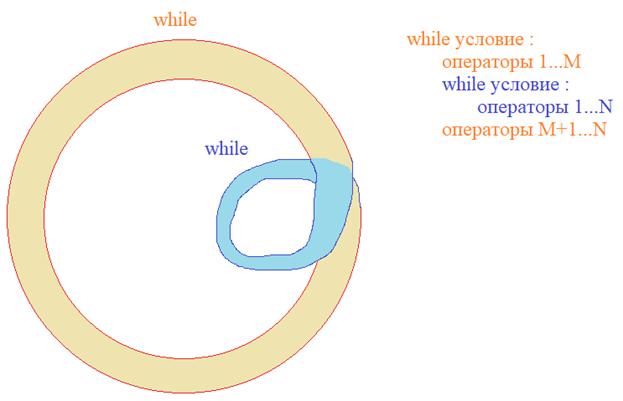
**Вложенные циклы**

Итак, мы с вами рассмотрели два оператора циклов: while и for. Все эти циклы можно комбинировать друг с другом. То есть, создавать вложенные циклы (цикл внутри цикла).

Как это работает? Представьте, что бегун начинает бежать по большому кругу, но затем, для продолжения бега, ему необходимо сделать еще несколько вложенных кругов, после чего он возвращается на большой круг и продолжает свой бег.

В частности, такие вложенные циклы очень полезны для перебора элементов матрицы/





Подобно условным конструкциям, циклы так же могут содержать в себе другие циклы и конструкции. Рассмотрим на примере вывода таблицы умножения:

|  |
| --- |
|  |
| for i in range(1, 10): |
| for j in range(1, 10): |
| print(I \* j, end=”\t”) |
| print(«\n») |

Здесь внутренний цикл выполняется 9 раз на каждой итерации внешнего цикла, которых тоже 9. Таблица умножения получается, поскольку в теле внутреннего цикла мы выводим произведение элементов обоих циклов.

**САМОСТОЯТЕЛЬНО**

Рассмотрим еще один пример: человеку необходимо подняться на 5-й этаж. Сейчас он находится на первом этаже, между этажами по 20 ступенек. Пройдя 70 ступенек подряд, человек устает и останавливается, чтобы отдохнуть:

floor = 1  
energy = 70  
print(f'Я на {floor} этаже')  
  
while floor != 5:  
 step = 0  
 while step != 20:  
 step += 1  
 energy -= 1  
 if energy == 0:  
 print('Я устал, я немного отдохну')  
 energy += 70  
 floor += 1  
 print(f'Сейчас я на {floor} этаже')  
print('Ghbitk')

Приведенный код демонстрирует, что каждая итерация цикла «while floor != 5:» выполняет вложенный цикл, а вложенный цикл выполняет конструкцию «if».

## 

## Управляющие операторы continue, break и else

Для управления выполнением цикла используются операторы «continue», «break» и «else», которые позволяют пропустить итерацию или выполнить набор инструкций после успешного завершения цикла соответственно.

Рассмотрим примеры их использования. ***break***

В приведенном примере так же видно, что при помощи «break» можно прервать бесконечный цикл. Эта особенность может быть полезна для создания циклов с постусловием. Вы уже могли заметить, что цикл «while» выполняет проверку условия перед выполнением тела цикла, поэтому возможна ситуация, когда условие будет изначально ложно и тело цикла не выполнится ни разу.

В других языках для избежания подобной ситуации можно использовать цикл «do … while» («цикл с постусловием»), в котором сначала выполняется тело цикла, а потом проверяется условие. В Python такого цикла нет, но его несложно составить самому. Допустим, мы хотим сделать цикл с постусловием из другого, ранее созданного цикла «while»:

|  |
| --- |
|  |
| number = 1 |
| while number < 5: |
|  |
| print(number) |
| number += 1 |
|  |

Для этого заменим условие цикла на «while», что сделает его бесконечным, и поместим оператор «break» в условную конструкцию, использующую условие, противоположное изначальному, в конце тела цикла:

|  |
| --- |
|  |
| number = 1 |
| while True: |
| print(number) |
| number += 1 |
| if number >= 5: |
| break |
|  |

***continue***

Оператор «continue» используется для прерывания текущей итерации цикла. Это полезно, если необходимо опустить некоторые итерации, и позволяет избежать вложенности кода. Рассмотрим обе ситуации. Допустим у нас есть цикл, который выводит пять последовательных чисел:

Этот результат можно изменить таким образом, чтобы на экран было выведено 5 последовательных четных чисел, используя «continue» при условии, что остаток от деления числа на два не равен нулю (т.е. число нечетное):

**САМОСТОЯТЕЛЬНО**

Для рассмотрения избегания вложенности кода предположим, что необходимо определить, какие из чисел от 0 до 300 делятся на 3, на 3 и 5 или на 3, 5 и 7 одновременно. Тогда код будет выглядеть следующим образом:

|  |
| --- |
|  |
| number = 0 |
| while number < 300: |
|  |
| number += 1 |
|  |
| if number % 3 == 0 and number % 5 != 0: |
|  |
| print(number, «divides by 3») |
|  |
| elif number % 3 == 0: |
|  |
| if number % 5 == 0 and number % 7 != 0: |
|  |
| print(number, “divides by 3 and 5”) |
|  |
| elif number % 5 == 0 and number % 7 == 0: |
| print(number, “divides by 3 and 5 and 7”) |
|  |

Здесь последняя инструкция находится на 3-м уровне вложенности, т. Е. он вложен одновременно в 3 конструкции. Используя «continue», можно разделить эти конструкции, заканчивая итерацию, если условие ложно:

number = 0  
while number < 300:  
 number += 1  
 if number % 3 != 0:  
 continue  
 elif number % 5 != 0:  
 print(number, "divides by 3")  
 elif number % 7 != 0:  
 print(number, "divides by 3 and 5")  
 else:  
 print(number, "divides by 3 and 5 and 7")

Преимущество такой конструкции в том, что она позволяет одновременно разделить условия, упростив их, и понизить уровень вложенности кода, тем самым упростить его прочтение.

Как и оператор «break», «continue» завершает итерацию только для вложенного цикла, в то время как итерация внешнего цикла продолжается.

***else***

В отличие от условных конструкций, в которых «else» выполняется только если условие ложно, «else» в циклах выполняется в том случае, когда цикл был успешно завершен. Например, ребенок, посчитавший от 1 до 5, может сказать, что он посчитал от 1 до 5: