
Яндекс

Академия Яндекса

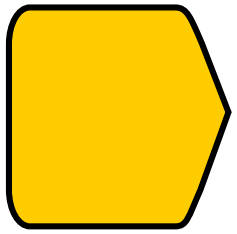
Цикл с предусловием

Цикл **while**

Цикл **while**

Сегодня мы научимся повторять заданные действия несколько раз. Для этого существуют операторы циклов.
Мы разберем оператор цикла **`while`**. Он выполняет блок кода, **пока истинно** какое-то условие.

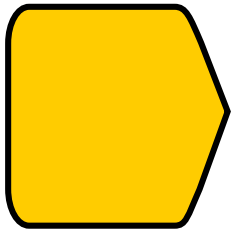
Напомним, условный оператор **`if`** проверяет условие и, в зависимости от того, истинно оно или ложно, выполняет либо не выполняет следующий записанный с отступом блок. После этого программа в любом случае выполняется дальше



Оператор **`while`** («пока») тоже проверяет условие и тоже в случае его истинности выполняет следующий блок кода («**тело цикла**»). Однако после выполнения этого блока кода выполняется не то, что идёт после него, а снова проверяется условие, записанное после **`while`**.

```
while условие:
    Блок кода (тело цикла)
```

Цикл **while**



Один шаг цикла (выполнение тела цикла) ещё называют **итерацией**.
Используйте цикл **`while`** всегда, когда какая-то часть кода должна выполниться несколько раз — причём невозможно заранее сказать, сколько именно.

Давайте посмотрим программу, в которой цикл будет выполняться пока не введут число меньше 0:

```
number = int(input())
while number > 0:
    print('Вы ввели положительное число! Вводите дальше.')
    number = int(input())
    print('Так-так, что тут у нас...')
print('Вы ввели отрицательное число или ноль. Всё.')
```

Пример

```
number = int(input())
while number > 0:
    print('Вы ввели положительное число! Вводите дальше.')
    number = int(input())
    print('Так-так, что тут у нас...')
print('Вы ввели отрицательное число или ноль. Всё.')
```

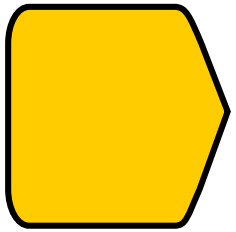
Шаг	Действие	Пояснение	Цикл
1	`number = int(input())`	`number = 10`	
2	`while number > 0:`	10 > 0 (Истина)	Входим в цикл, 1 итерация
3	`print('Вы ввели положительное число! Вводите дальше.')`	Вывод	
4	`number = int(input())`	`number = 2`	
5	`print('Так-так, что тут у нас...')`	Вывод	
6	`while number > 0:`	2 > 0 (Истина)	2 итерация
7	`print('Вы ввели положительное число! Вводите дальше.')`	Вывод	
8	`number = int(input())`	`number = 3`	
9	`print('Так-так, что тут у нас...')`	Вывод	
10	`while number > 0:`	3 > 0 (Истина)	3 итерация
11	`print('Вы ввели положительное число! Вводите дальше.')`	Вывод	
12	`number = int(input())`	`number = -1`	
13	`print('Так-так, что тут у нас...')`	Вывод	
14	`while number > 0:`	-1 > 0 (Ложь)	Выход из цикла
15	`print('Вы ввели отрицательное число или ноль. Всё.')`	Вывод	

Составной оператор присваивания

Составной оператор присваивания

Напомним, что в операторе присваивания одно и то же имя переменной может стоять и справа (в составе какого-то выражения), и слева. В этом случае сначала вычисляется правая часть со старым значением переменной, после чего результат становится новым значением этой переменной. Ни в коем случае не воспринимайте такой оператор присваивания как уравнение!

```
number = int(input())   # например, 5
number = number + 1   # тогда здесь number становится равным 6
print(number)
```



Для конструкций вида **`number = number + 1`**, также существует сокращенная форма записи оператора присваивания: **`number += 1`**.
Аналогично оператор **`x = x + y`**
можно записать как **`x += y`**,
оператор **`x = x * y`**
— как **`x *= y`**,
и так для любого из семи арифметических действий.

Подсчет количества элементов, удовлетворяющих условию

А теперь рассмотрим задачу.

Пользователь вводит целые числа. Ввод чисел прекращается, если введено число 0. Необходимо определить сколько чисел среди введенных оканчивались на 2 и были кратны числу 4. Теперь нам надо проверять последовательность чисел.

Для каждого введенного числа надо делать проверку, соответствует ли оно условию. Если оно подходит под условие, то увеличиваем счетчик таких чисел. Если нет — цикл останавливается, а интерпретатор переходит на строку, следующую за циклом. Такое условие называется **сигналом остановки**.
И уже после цикла, когда остановился ввод чисел — выводим результат — посчитанное количество нужных чисел.

```
count = 0
number = int(input())
while number != 0:
    if number % 10 == 2 and number % 4 == 0:
        count += 1
    number = int(input())
print('Количество искомых чисел:', count)
```

Подсчет количества элементов, удовлетворяющих условию

Шаг	Действие	Пояснение	Цикл
1	`count = 0`	`count = 0`	
2	`number = int(input())`	`number = 12`	
3	`while number != 0:`	12 != 0 (Верно)	Вход в цикл, 1 итерация
4	`if number % 10 == 2 and number % 4 == 0:`	`12 % 10 == 2 and 12 % 4 == 0` (Верно)	Заходим в if
5	`count += 1`	`count = 1`	
6	`number = int(input())`	`number = 3`	
7	`while number != 0:`	3 != 0 (Верно)	2 итерация
8	`if number % 10 == 2 and number % 4 == 0:`	`3 % 10 == 2 and 3 % 4 == 0` (Ложно)	Пропускаем if
9	`number = int(input())`	`number = 32`	
10	`while number != 0:`	32 != 0 (Верно)	3 итерация
11	`if number % 10 == 2 and number % 4 == 0:`	`32 % 10 == 2 and 32 % 4 == 0` (Верно)	Заходим в if
12	`count += 1`	`count = 2`	
13	`number = int(input())`	`number = 14`	
14	`while number != 0:`	14 != 0 (Верно)	4 итерация
15	`if number % 10 == 2 and number % 4 == 0:`	`14 % 10 == 2 and 14 % 4 == 0` (Ложно)	Пропускаем if
16	`number = int(input())`	`number = 0`	
17	`while number != 0:`	0 != 0 (Ложно)	Выход из цикла
18	`print('Количество искомых чисел:', count)`	Вывод вычисленного count	

```
count = 0
number = int(input())
while number != 0:
    if (number % 10 == 2 and
        number % 4 == 0):
        count += 1
    number = int(input())
print('Количество искомых чисел:',
      count)
```

Сигнал остановки

Пример задачи

Рассмотрим такую задачу: пользователь вводит числа.

Пусть это будут цены на купленные в магазине товары, а наша программа — часть программного обеспечения кассового аппарата. Ввод "-1" — сигнал остановки. Нужно сосчитать сумму всех введённых чисел (сумму чека).

Поскольку требуется повторить нечто (ввод очередной цены) неизвестное количество раз, потребуется цикл **`while`**. Нам понадобится как минимум две переменные: **`price`** для цены очередного товара и **`total`** для общей суммы.

Если бы мы знали точно, что пользователю надо купить ровно три товара, то цикл (и ввод -1 как условие его прерывания) был бы не нужен. Тогда программа могла бы выглядеть так:

```
total = 0
price = float(input())
total = total + price
price = float(input())
total = total + price
price = float(input())
total = total + price
print('Сумма введённых чисел равна', total)
```

Пример задачи

Если бы мы хотели сократить запись, можно было бы организовать цикл, который выполнился бы ровно три раза. Для этого нам потребуется переменная счетчик, которая внутри цикла будет считать каждую итерацию цикла. А условием выхода — поставим выполнение нужного количества итераций:

Обратите внимание, что **total** и **count** должны обнуляться до цикла.

```
count = 0
total = 0
while count < 3:
    price = float(input())
    total = total + price
    count = count + 1
print('Сумма введённых чисел равна', total)
```

Пример задачи

Однако у нас в задаче количество товаров неизвестно, поэтому понадобится цикл до ввода сигнала остановки (-1). С учётом сказанного выше программа будет выглядеть так:

```
total = 0
print('Вводите цены; для остановки введите -1.')
price = float(input())
while price > 0:
    total = total + price  # можно заменить на аналогичную запись  total += price
    price = float(input())
print('Общая стоимость равна', total)
```

Поиск максимума и минимума

Поиск максимума и минимума

Рассмотрим алгоритм в общем виде:

1. заведем отдельную переменную для хранения максимума и минимума. В качестве начального значения можно задать:

- заведомо малое для анализируемых данных значения, для максимума – это будет очень маленькое число, например, если мы вычисляем максимальный балл за экзамен, то можно взять `maximum = 0`, тогда гарантированно произойдет замена максимума. Минимуму же наоборот присваивается заведомо большое значение;
- первый элемент данных;

2. в теле цикла каждый подходящий элемент данных обрабатывается операторами по принципу:

- если текущий элемент больше максимума, меняем максимум;
- если текущий элемент меньше минимума, заменяем минимум.

Поиск максимума и минимума

Рассмотрим пример. Витя анализировал список литературы и решил, что хочет начать с самой большой по количеству страниц книги. Напишем программу, которая поможет Вите определить сколько страниц ему предстоит прочитать. Витя последовательно вводит количество страниц каждой книги из списка, а окончанием ввода служит ввод любого отрицательного числа.

```
biggest_book = 0
n = int(input())
while n > 0:
    if n > biggest_book:
        biggest_book = n
    n = int(input())
print(biggest_book)
```

Шаг	Действие	Пояснение	Цикл
1	biggest_book = 0	Задание начального значения	
2	n = int(input())	n = 148	
3	while n > 0:	148 > 0 (ИСТИНА)	Вход в цикл, 1 итерация
4	if n > biggest_book:	148 > 0 (ИСТИНА)	
5	biggest_book = n	biggest_book = 148	
6	n = int(input())	n = 120	
7	while n > 0:	120 > 0 (ИСТИНА)	2 итерация
8	if n > biggest_book:	120 > 148 (ЛОЖЬ)	
9	n = int(input())	n = 486	
10	while n > 0:	486 > 0 (ИСТИНА)	3 итерация
11	if n > biggest_book:	486 > 148 (ИСТИНА)	
12	biggest_book = n	biggest_book = 486	
13	n = int(input())	n = -1	
14	print(biggest_book)	Вывод	

«Моржовый» оператор

Моржовый оператор

Начиная с версии **Python 3.8** появилась возможность одновременно решать две задачи: присвоить переменной значение и сразу вернуть его. Это позволяет сократить код, сделать его более читаемым, а в ряде случаев и более эффективным. Для этого используется моржовый оператор (**walrus, :=**).

В отличие от простого присваивания, которое просто связывает переменную и ее значение, моржовый оператор позволяет присвоить значение переменной в процессе выполнения какой-то инструкции:

```
assignment = 2 * 3 + 1  # обычное присваивание
print(assignment)

print(walrus := 2 * 3 + 1)  # присваивание и возвращение значения
```

Моржовый оператор

В этом примере оператор присваивания будет выполнен **последним**, поскольку у него самый низкий приоритет. И действительно, сначала надо получить результат операции сравнения и уже потом этот результат надо занести в переменную **`line`**, которая получит тип **`bool`**. На экран при этом будет выведено:

```
while line := input() != "":
    print(line)
```

```
Hello  # True
world  # True
```

Моржовый оператор

В этом примере оператор присваивания будет выполнен **последним**, поскольку у него самый низкий приоритет. И действительно, сначала надо получить результат операции сравнения и уже потом этот результат надо занести в переменную `line`, которая получит тип `bool`. На экран при этом будет выведено:

```
while line := input() != "":
    print(line)
```

```
Hello   # True
world   # True
```

А в этом примере будет иное поведение, поскольку мы использовали скобки, изменив порядок выполнения операций:

```
while (line := input()) != "":
    print(line)
```

Моржовый оператор

Еще пример. Различный вывод при выполнении или невыполнении условия можно записать в одну строку:

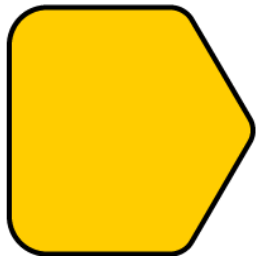
```
x = int(input())
print((x + 1) ** 2 if x + 1 < 0 else x + 1)
```

Однако, в этом случае нам три раза пришлось посчитать выражение **`x + 1`**. Теперь можно сделать проще: один раз посчитать выражение, записать его в переменную и оперировать уже посчитанным значением:

```
x = int(input())
print(y ** 2 if (y := x + 1) < 0 else y)
```

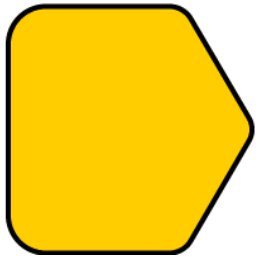
Алгоритм Евклида. Наибольший общий делитель

Алгоритм Евклида. Наибольший общий делитель



Наибольший общий делитель (НОД, GCD (**Greatest common divisor**)) двух целых чисел — это наибольшее число, на которое можно разделить оба числа.
Алгоритм нахождения НОД назван по имени древнегреческого математика Евклида Александрийского, предложившего данный метод. Существует две реализации — вычитанием и делением. Рассмотрим обе.

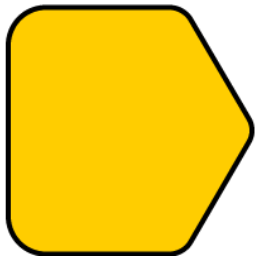
Алгоритм Евклида (вычитанием)



- Если числа равны, то любое из них является НОДом.
- Если нет, то выберем большее и вычтем из большего числа меньшее.
- Вернемся к пункту 1, только теперь будем работать с меньшим числом и результатом вычитания.

```
a = int(input())
b = int(input())
while a != b:
    if a > b:
        a -= b
    else:
        b -= a
print(b)
```

Алгоритм Евклида (делением)



- Большее число делим на меньшее
- Если делится без остатка – меньшее число и есть НОД, выходим из цикла
- Если остаток не 0, большее число заменяем на этот остаток
- Возвращаемся к пункту 1

Попробуйте написать программу, реализующую данный метод. В данном случае условием выхода из цикла будет равенство нулю какого-либо из чисел.

Алгоритм Евклида (делением)

```
a = int(input())
b = int(input())
while a * b != 0:
    if a > b:
        a %= b
    else:
        b %= a
print(a + b)
```

или

```
a = int(input())
b = int(input())
while a * b != 0:
    a, b = b, a % b
print(a + b)
```


Яндекс