* [ОБЗОР КУРСА](https://lyceum.yandex.ru/courses/123/groups/568)

[Урок True и False](https://lyceum.yandex.ru/courses/123/groups/568/lessons/648)

**True, False, break, continue**

**План урока**

1

[Логический тип данных](https://lyceum.yandex.ru/courses/123/groups/568/lessons/648/materials/836#1)

2

[Флаги](https://lyceum.yandex.ru/courses/123/groups/568/lessons/648/materials/836#2)

3

[Оператор break](https://lyceum.yandex.ru/courses/123/groups/568/lessons/648/materials/836#3)

4

[Оператор continue](https://lyceum.yandex.ru/courses/123/groups/568/lessons/648/materials/836#4)

5

[Бесконечные циклы](https://lyceum.yandex.ru/courses/123/groups/568/lessons/648/materials/836#5)

**Аннотация**

*Этот урок посвящен условиям выхода из циклов. Рассматривается булев тип, даются задачи на использование флагов. Затем рассматриваются операторы break и continue, позволяющие в некоторых случаях избавиться от флагов.*

**1. Логический тип данных**

Если a и b — числа (допустим, действительные), то у выражения a+b есть какое-то значение (зависящее от значений a и b) и тип — тоже действительное число. Как вы думаете, можно ли сказать, что у выражения a==b есть значение и тип? Или это просто конструкция, которая всегда должна стоять в условии if или while?

**Тип bool**

На самом деле, такое выражение имеет и тип под названием bool, и значение: **True** (истина) или **False** (ложь). По-русски bool — это булев тип или булево значение (в честь математика Джона Буля), иногда его ещё называют «логический тип».

Логический тип может иметь только два значения, а над переменными логического типа можно выполнять логические операции **not**, **and**, **or**.

Также для приведения к логическому типу можно использовать функцию **bool**, которая для ненулевого значения вернет истину.

k = True

**print**(k) *# выведет True*

**print**(**not** k) *# выведет False*

k = 5 > 2

**print**(k) *# выведет True*

k = bool(0)

**print**(k) *# выведет False*

k = bool("")

**print**(k) *# выведет False*

k = bool(13)

**print**(k) *# выведет True, т.к. число не 0*

k = bool("q")

**print**(k) *# выведет True, т.к. строка не пустая*

k = bool("False")

**print**(k) *# выведет True, т.к. строка не пустая*

Или вот еще пример:

**if** True:

**print**('Эта строка будет выведена на экран.')

**else**:

**print**('Эта строка никогда не будет выведена на экран.')

**print**(2 \* 2 == 4) *# выведет True*

a = input()

b = input()

*# Теперь переменная equal равна True, если строки a и b равны,  
# и False в противном случае*

equal = (a == b)

**if** equal **and** len(a) < 6:

**print**('Вы ввели два коротких одинаковых слова.')

**2. Флаги**

Обычно переменные с булевым значением используются в качестве **флагов**.

**Важно**

Изначально флаг устанавливается в **False**, потом программа как-то работает, а при наступлении определённого события флаг устанавливается в **True**. После идёт проверка, «поднят» ли флаг. В зависимости от её результата выполняется то или иное действие. Иными словами, флаг — это переменная с булевым значением, которая показывает, наступило ли некое событие.

В примере ниже (эта программа — терапевтический тренажёр для избавления физиков-экспериментаторов от синхрофазотронозависимости) имеется флаг said\_forbidden\_word, который означает «сказал ли пользователь запретное слово „синхрофазотрон“». Флаг равен True, если сказал, и False, если нет.

В самом начале пользователь ещё ничего не успел сказать, поэтому флаг установлен в False. Далее на каждой итерации цикла, если пользователь сказал запретное слово, флаг устанавливается в True и остаётся в таком состоянии (при необходимости флаг можно и «опустить»). Как только флаг оказывается равен True, поведение программы меняется: перед каждым вводом выдаётся предупреждение, а в конце выдаётся другое сообщение.

**Важно**

Переменным-флагам особенно важно давать осмысленные имена (обычно — утверждения вроде said\_forbidden\_word, found\_value, mission\_accomplished, mission\_failed), ведь флагов в программе бывает много.

forbidden\_word = 'синхрофазотрон'

*# можно было использовать и sep='', чтобы кавычки не отклеились от слова*

**print**('Введите десять слов, но постарайтесь случайно не ввести слово "' + forbidden\_word + '"!')

said\_forbidden\_word = False

**for** i **in** range(10):

**if** said\_forbidden\_word:

**print**('Напоминаем, будьте осторожнее, не введите снова слово "' + forbidden\_word + '"!')

word = input()

**if** word == forbidden\_word:

said\_forbidden\_word = True

*# вместо предыдущих двух строк также можно написать:*

*# said\_forbidden\_word = (said\_forbidden\_word or word == forbidden\_word)*

**if** said\_forbidden\_word:

**print**('Вы нарушили инструкции.')

**else**:

**print**('Спасибо, что ни разу не упомянули', forbidden\_word)

**3. Оператор break**

Если нужно прекратить работу цикла как только случится некое событие, то кроме флага есть и другой способ — оператор разрыва цикла **break** (он работает и для цикла for). Это не функция и не заголовок блока, а оператор, который состоит из одного слова. Он немедленно прерывает выполнение цикла for или while.

**for** i **in** range(10):

**print**('Итерация номер', i, 'начинается...')

**if** i == 3:

**print**('Ха! Внезапный выход из цикла!')

**break**

**print**('Итерация номер', i, 'успешно завершена.')

**print**('Цикл завершён.')

В частности, нередко встречается такая конструкция: цикл, выход из которого происходит не по записанному в заголовке цикла условию (это условие делается всегда истинным — как правило, просто True), а по оператору break, который уже заключён в какой-то условный оператор:

**while** True:

word = input()

**if** word == 'стоп':

**break**

**print**('Вы ввели:', word)

**print**('Конец.')

**Важно**

Впрочем, злоупотреблять этой конструкцией и вообще оператором break не стоит. Когда программист читает ваш код, он обычно предполагает, что после окончания цикла while условие в заголовке этого цикла ложно. Если же из цикла можно выйти по команде break, то это уже не так. Логика кода становится менее ясной.

**4. Оператор continue**

Оператор **continue** немедленно завершает текущую итерацию цикла и переходит к следующей.

**for** i **in** range(10):

**print**('Итерация номер', i, 'начинается...')

**if** i == 3:

**print**('...но её окончание таинственно пропадает.')

**continue**

**print**('Итерация номер', i, 'успешно завершена.')

**print**('Цикл завершён.')

**5. Бесконечные циклы**

Рассмотрим еще один пример:

count = 1

**while** count < 100:

**if** count % 5 == 0:

**continue**

**print**(count)

count += 1

Что будет напечатано в процессе выполнения программы?

Предполагается, что программа выведет все числа от 1 до 100, не кратные 5. Но на самом деле, если вы запустите программу в режиме трассировки, на экран выведется 1 2 3 4, а потом программа уйдет в бесконечный цикл. Почему это происходит?

Когда переменная count станет равна 5, записанное в операторе if условие станет истинным, и выполнится оператор continue. Т.е. мы немедленно перейдем к следующей итерации цикла, пропуская вывод числа и увеличение счетчика count.

Переменная count так и не увеличится и по-прежнему останется со значением 5. Значит, условие в if будет все также равно True, и цикл станет бесконечным.

Иными словами, часто использовать break и continue не рекомендуют, поскольку они приводят к произвольному перемещению точки выполнения программы по всему коду, что усложняет понимание и следование логике. Тем не менее, разумное использование этих операторов может улучшить читабельность циклов в программе, уменьшив при этом количество вложенных блоков и необходимость в сложной логике выполнения цикла.

Например, рассмотрим следующую программу:

count = 0

exitLoop = False

**while** (**not** exitLoop):

**print**("Введите 'e' для выхода и любую другую клавишу для продолжения")

sm = input()

**if** (sm == 'e'):

exitLoop = True

**else**:

count += 1

**print**("Вы зашли в цикл ", count, " раз(а)")

А теперь ту же самую программу напишем с использованием оператора break:

exitLoop = False

**while** (**not** exitLoop):

**print**("Введите 'e' для выхода и любую другую клавишу для продолжения")

sm = input()

**if** (sm == 'e'):

**break**

count += 1

**print**("Вы зашли в цикл ", count, " раз(а)")

Чего нам удалось добиться? Во-первых, мы избежали использования как логической переменной, так и оператора else. Уменьшение количества используемых переменных и вложенных блоков улучшают читабельность и понимание кода больше, чем break или continue могут нанести вред.

[Справка](https://yandex.ru/support/lyceum-students)

Исключительное право на учебную программу и все сопутствующие ей учебные материалы, доступные в рамках проекта «Яндекс.Лицей», принадлежат АНО ДПО «ШАД». Воспроизведение, копирование, распространение и иное использование программы и материалов допустимо только с предварительного письменного согласия АНО ДПО «ШАД».

© 2018 – 2020  ООО «[Яндекс](https://yandex.ru/)»

Чаты