
Яндекс

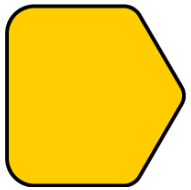
Академия Яндекса

Булевы переменные. Прерывания и продолжения циклов

Логический тип данных

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?



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 Джорджа 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`.

Приведение к логическому типу

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
```

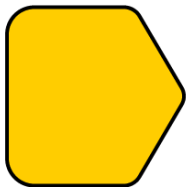
Пример

```
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.')
```

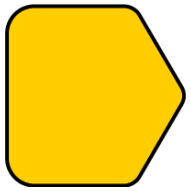
Использование флагов

Использование флагов

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



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.



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.

Использование флагов

```
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)
```

Операторы break и continue. Бесконечные циклы

Операторы `break` и `continue`

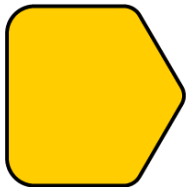
Если нужно прекратить работу цикла как только случится некое событие, то кроме флага есть и другой способ — оператор разрыва цикла `break` (он работает и для цикла `for`).

```
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.')
```

Операторы `break` и `continue`

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`**).

```
while True:
    word = input()
    if word == 'стоп':
        break
    print('Вы ввели:', word)
print('Конец.')
```



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.

Операторы **`break`** и **`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.')
```

```
Итерация номер 0 начинается...
Итерация номер 0 успешно завершена.
Итерация номер 1 начинается...
Итерация номер 1 успешно завершена.
Итерация номер 2 начинается...
Итерация номер 2 успешно завершена.
Итерация номер 3 начинается...
...но её окончание таинственно пропадает.
Итерация номер 4 начинается...
Итерация номер 4 успешно завершена.
Итерация номер 5 начинается...
Итерация номер 5 успешно завершена.
Итерация номер 6 начинается...
Итерация номер 6 успешно завершена.
Итерация номер 7 начинается...
Итерация номер 7 успешно завершена.
Итерация номер 8 начинается...
Итерация номер 8 успешно завершена.
Итерация номер 9 начинается...
Итерация номер 9 успешно завершена.
Цикл завершён.
```

Пример

```
count = 1
while count < 100:
    if count % 5 == 0:
        continue
    print(count)
    count += 1
```

Что выведет эта программа?

Пример

```
count = 1
while count < 100:
    if count % 5 == 0:
        continue
    print(count)
    count += 1
```

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?

Измените программу так, чтобы она работала правильно.

Подведение итогов

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.

```
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)")
```

```
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)")
```

Уменьшение количества используемых переменных и вложенных блоков улучшают читабельность и понимание кода больше, чем **break** или **continue** могут нанести вред.

Преобразование bool к int

Преобразование `bool` к `int`

Возможность представить логическое выражение числом позволяет сильно упростить решение задач на сравнение значений, поскольку можно обойтись без условного оператора.

Рассмотрим задачу, в которой нужно вывести бОльшее число из двух введенных. Вот так можно ее решить, не используя ветвление:

```
n = int(input())
m = int(input())
print(n * int(n >= m) + m * int(m > n))
```


Яндекс