Булевы переменные. Прерывания и продолжения циклов

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`?

Логический 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 еще 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. Использование флагов

Обычно переменные с булевым значением используются в качестве флагов.

Флаг

Изначально флаг устанавливается в False, потом программа как-то работает, а при наступлении определенного события флаг устанавливается в True. После идет проверка, поднят ли флаг. В зависимости от ее 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 тренажер 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 еще ничего не успел сказать, поэтому

флаг установлен в False. Далее на каждой итерации цикла, если пользователь сказал запретное слово, флаг устанавливается в True и остается в таком состоянии (при необходимости флаг можно и «опустить»). Как только флаг оказывается равен True, поведение программы меняется: перед каждым вводом выдается предупреждение, а в конце выдается 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. Операторы break и continue. Бесконечные циклы

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 заключен 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.

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.')
```

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 так же 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.