


ХОЧУ ПОМОЧЬ
ПРОЕКТИ

mango-office.ru

Виртуальная АТС Расширенная

До 120 функций. Когда нужно распределять звонки по группам сотрудников и по типам...

Узнать больше

РЕКЛАМА

[Справочник по языку Python3.](#) / Смешивание операторов в цепочках сравнений в Python

Содержание:

- [Общие приемы использования операторов: is, in, not, and, or:](#)
 - [Применение оператора in,](#)
 - [Применение оператора is,](#)
 - [Применение операторов сравнения,](#)
 - [Применение булевых операторов.](#)
- [Смешивание операторов в цепочках сравнений:](#)
 - [Связывание цепочек при помощи is,](#)
 - [Связывание цепочек при помощи in.](#)

Использование операторов is, in, not, and, or и операторов сравнения.

Условия, используемые в операторах [while](#) и [if...else](#) могут содержать любые операторы, а не только операторы сравнения.

Применение оператора in:

[Операторы in и not in](#) проверяют, входит/не входит ли значение в последовательность.

```
>>> basket = ['orange', 'banana', 'pear', 'apple']

# вхождение/наличие элемента в списке
>>> 'orange' in basket
# True
>>> 'crabgrass' in basket
# False
>>> 'banana' not in basket
# False
```

Применение оператора is:

[Операторы is и is not](#) сравнивают, действительно ли два объекта являются одним и тем же объектом (идентичным). Это имеет значение только для изменяемых объектов, таких как списки.

```
var = None

>>> var is None
# True
>>> var is not None
# False
```

Применение операторов сравнения:

Все [операторы сравнения](#) имеют одинаковый приоритет. Сравнения могут быть записаны как цепочки сравнений. Например выражение `A < B == C` проверяет, меньше ли A, чем B, и кроме того сравнивает равно ли B и C.

Вверх

Применение булевых операторов:

Сравнения могут быть объединены с помощью булевых операторов `and` и `or`, а результат сравнения или любого другого булева выражения может быть сведен на НЕТ с помощью оператора `not`. Они имеют [более низкие приоритеты](#), чем [операторы `ср`](#)^{РЕКЛАМА 1}. Из булевых операторов, `not` имеет самый высокий приоритет, а `or` самый низкий, так что `A and not B or C` эквивалентно `(A and (not B)) or C`. Как всегда, скобки могут быть использованы для выражения желаемого приоритета в операциях.

[Булевы операторы](#) `and` и `or` являются так называемыми операторами замыкания: их аргументы вычисляются слева направо и вычисление прекращается, как только результат определен. Например, если `A` и `C` истинны `True`, а `B` ложно `False`, то выражение `A and B and C` не вычисляет `C`.

Можно присвоить переменной [результат сравнения](#) или другое логическое выражение (типа тернарных операций в языке `C` - но это не то же самое!!!). При использовании логического выражения в качестве получения значения для присваивания переменной, возвращаемое значение будет равно последнему вычисляемому аргументу в логическом выражении. За подробностями обратитесь к [описанию логических операторов](#).

Например:

```
>>> string1, string2, string3 = '', 'Trondheim', 'Hammer Dance'
>>> non_null = string1 or string2 or string3
>>> non_null
# 'Trondheim'
```

Обратите внимание, что в Python, в отличие от языка `C`, присваивание внутри выражений должно выполняться явно с помощью [выражения присваивания \(моржового оператора\)](#) `walrus:=`. Это позволяет избежать распространенного класса проблем, встречающихся в программах на языке `C`: `typing = in an expression when == was intended.vv`.

Смешивание операторов в цепочках сравнений.

Немного о неожиданных результатах при добавлении оператора идентичности `is` и оператора проверки вхождения `in` в [цепочки сравнения](#).

Связывание цепочек сравнений при помощи оператора `is`.

В цепочках сравнения можно связать все выше перечисленные операторы сравнения Python. Это может **привести к неожиданному поведению**:

```
>>> a = 0
>>> a is a < 1
# True
>>> (a is a) < 1
# False
>>> a is (a < 1)
# False
```

Так как `a < 1` - это цепочка сравнения, то она оценивается как `True`. Разорвем цепочку на части:

- Выражение `a is a` истинно `True`, как и для любого значения, оцениваемого по отношению к самому себе.
- Выражение `a < 1` истинно `True`, так как `0` меньше `1`.

Следовательно обе части истинны, цепочка вычисляется как истинная.

Некоторые программисты, могут предположить, что в цепочках сравнения можно указывать приоритет выполнения того или иного сравнения как в выражениях, включающие несколько операторов `(1 + 2) * 3`. В случае с цепочками сравнений ни один из способов вставки скобок не вернет `True`.

Разберем, почему обе цепочки сравнения с круглыми скобками оцениваются как `False`. Если разбить первое выражение `(a is a) < 1`, то получится следующее:

```
>>> a = 0
>>> a is a
# True
>>> (a is a) == 1
# False
```

```
>>> (a is a) < 1
# False

# т.е. в итоге сравниваются
>>> True < True
# False
```

Из кода выше видно, что выражение `a is a` возвращает `True`. Это означает, что выражение `(a is a) < 1` будет преобразовано в выражение `True < 1`. Логические значения являются числовыми типами, а `True` равно 1. Таким образом, `True < 1` совпадает с `1 < 1`. Поскольку это строгое неравенство, то оно возвращает `False`.

Второе выражение `a is (a < 1)` с круглыми скобками работает иначе:

```
>>> a = 0
>>> a < 1
# True
>>> 0 is True
# False
```

Поскольку 0 меньше 1, то выражение `a < 1` возвращает `True`. Логические значения являются числовыми типами и `True` равно 1, следовательно 0 ну ни как не будет идентичен 1.

**Самый важный урок, который можно извлечь из этого, заключается в том, что связывание сравнений с помощью [оператора is](#) не является хорошей идеей. Это сбивает с толку и, вероятно, в этом нет необходимости.*

Связывание цепочек сравнений при помощи оператора in.

Подобно оператору `is`, [оператор in и его противоположность not in](#), часто могут давать удивительные результаты при объединении в цепочку сравнений:

```
>>> "b" in "aba" in "cabad" < "cabae"
# True

# так как приоритет оператора '<' выше
# чем 'in', то 1 выполниться выражение
>>> "cabad" < "cabae"
# True

# далее выражение будет выполняться
# как обычно с лева на право, следовательно
# вторым выполниться
"b" in "aba"
# True

# и последним выполняется проверка вхождения
>>> True in True
# True
```

Чтобы избежать путаницы, в этом примере сравниваются цепочки с разными операторами и используются строки для проверки подстрок. Опять же, это ПЛОХОЙ пример написанного кода! Все же важно уметь прочитать этот пример и понять, почему он возвращает `True`.

Наконец, можно связать цепочку `is not` с `not in`:

```
>>> greeting = "hello"
>>> quality = "good"
>>> end_greeting = "farewell"
>>> greeting is not quality not in end_greeting
# True
```

Обратите внимание, что порядок оператора `not` в двух операторах не одинаков! Отрицательными операторами являются `is not` и `not in`. Это соответствует обычному использованию таких оборотов в английском языке, но легко ошибиться при изменении кода.

- [КРАТКИЙ ОБЗОР МАТЕРИАЛА.](#)
- [Сравнение последовательностей и других типов](#)

РЕКЛАМА



DOCS-Python.ru™, 2023 г.

(Внимание! При копировании материала ссылка на источник обязательна)

[@docs_python_ru](#)

Вверх