


ХОЧУ ПОМОЧЬ ПРОЕКТУ

Операции сравнения в Python, цепочки сравнений



mango-office.ru

РЕКЛАМА

Виртуальная АТС Расширенная

2 000 ₽

Узнать больше

Бесплатное занятие английским в Яндекс Практикуме

Полноценное занятие с преподавателем, а не презентация курсов

Устный тест на уровень языка

Практика английского

Узнать больше

Операции сравнения в Python, цепочки сравнений

х сравнения;

в операциях сравнения;

эпочек сравнения);

ры сравнения;

в выражении;

сравнения;

чек сравнения;

Операторах сравнения.

авнения. Все они имеют одинаковый приоритет, который выше, чем у [логических операций](#).

Разрешенные операции сравнения:

- `x < y` - строго `x` меньше `y`,
- `x <= y` - `x` меньше или равно `y`,
- `x > y` - строго `x` больше `y`,
- `x >= y` - `x` больше или равно `y`,
- `x == y` - `x` равно `y`,
- `x != y` - `x` не равно `y`.

**Внимание!** [Комплексные числа \(тип `complex`\)](#) не поддерживают сравнение порядка.

Сравнения могут быть связаны произвольно и записаны в [цепочки сравнений](#), в которых для соединения сравнений используются неявные логические операторы `and`.

```
x < y <= z
# эквивалентно
x < y and y <= z
```

В примере выше `y` вычисляется только один раз. Если `x < y` оказывается ложным, то в обоих случаях, приведенных выше `z` не оценивается вообще.

Еще пример:

```
a < b <= c < d
# эквивалентно
a < b and b <= c and c < d
```

В такой форме сравнения легче читаются, и каждое подвыражение вычисляется по крайней мере один раз.

Объекты разных типов, за исключением различных числовых типов, никогда не будут равными.

Оператор `==` всегда определен, но для некоторых типов объектов, например объектов класса, эквивалентен [оператору `is`](#).

https://docs-python.ru/tutorial/operatsii-sravneniya-python/

1/5

Операторы `<`, `<=`, `>` и `>=` применяются только там, где они имеют смысл, например они вызывают исключение `TypeError`, когда один из аргументов является комплексным числом.

Неидентичные экземпляры класса обычно при сравнении будут неравны, если только класс не определяет метод `__eq__()`.

РЕКЛАМА · 18+

упорядочены относительно других экземпляров того же класса или других типов. Для сравнения требуется достаточное количество методов `__lt__()`, `__le__()`, `__gt__()` и `__ge__()`. В общем случае для этих целей бывает достаточно.

Числа `decimal.Decimal` являются десятичными дробями ([decimal.Decimal](#)) и используют двоичную арифметику, что позволяет им [вычисляться с ничтожно малыми погрешностями](#). Из-за этих погрешностей операции с ними требуются дополнительные проверки.

```
from decimal import Decimal

Decimal('0.5') - Decimal('0.3') == 0
Decimal('0.5') - Decimal('0.3')
```

### ЧИСЛОВЫХ ТИПОВ в операциях сравнения:

Числовые типы `int`, `float`, `complex` и стандартных библиотечных типов `fractions.Fraction` и `decimal.Decimal` можно сравнивать внутри и между их типами, с ограничением, что комплексные числа не поддерживают сравнение порядка. В пределах задействованных типов они сравнивают математически (алгоритмически) правильно без потери точности.

Нечисловые значения `float('NaN')` и `decimal.Decimal('NaN')` являются особыми. Любое упорядоченное сравнение числа с нечисловым значением неверно. Нечисловые значения не равны самим себе. Например, если `x = float('NaN')`, `3 < x`, `x < 3` и `x == x` все ложны, а `x != x` истинно. Это поведение соответствует стандарту IEEE 754.

- `None` и `NotImplemented` являются одиночными. PEP 8 советует, что сравнения для одиночных экземпляров всегда должны выполняться с использованием `is` или `is not`, а не с операторами равенства.
- Двоичные последовательности (экземпляры `bytes` или `bytearray`) можно сравнивать внутри и между их типами. Они сравнивают лексикографически, используя числовые значения своих элементов.
- Строки (экземпляры `str`) сравниваются лексикографически с использованием числовых кодовых точек `Unicode` (результат [встроенной функции ord\(\)](#)) их символов.

Строки и двоичные последовательности напрямую сравнивать нельзя.

- Последовательности (экземпляры `tuple`, `list` или `range`) можно сравнивать только в пределах каждого из их типов с ограничением, что диапазоны `range` не поддерживают сравнение порядка (сортировку). Оператор `==` между этими типами приводит к неравенству, а сравнение порядка между этими типами вызывает [исключение TypeError](#).

Последовательности сравнивают лексикографически с помощью сравнения соответствующих элементов. Встроенные контейнеры обычно предполагают, что идентичные объекты равны самим себе. Это позволяет им обходить тесты на равенство для идентичных объектов, чтобы повысить производительность и сохранить свои внутренние инварианты.

Лексикографическое сравнение встроенных коллекций работает следующим образом:

- Чтобы две коллекции были равными, они должны быть одного типа, иметь одинаковую длину и каждая пара соответствующих элементов должна быть равной. Например `[1,2] == (1,2)` ложно, потому что типы последовательностей разные.
- Коллекции, поддерживающие сравнение порядка ([сортировку](#)), упорядочиваются также, как их первые неравные элементы, например `[1,2, x] <= [1,2, y]` имеет то же значение, что и `x <= y`. Если соответствующий элемент не существует, то более короткая коллекция при сортировке встанет первой, например `[1,2] < [1,2,3]`


Вверх

истинно).

- Множества (экземпляры [set](#) или [frozenset](#)) можно сравнивать внутри и между их типами.

Они определяют операторы сравнения порядка для обозначения тестов подмножества и надмножества. Эти отношения не

РЕКЛАМА · 18+ · 9



ОФ

practicum.yandex.ru

**Бесплатное занятие английским в Яндекс Практикуме**

Полноценное занятие с преподавателем, а не презентация курсов

Устный тест на уровень языка

Практика английского

Узнать больше

Например два множества {1,2} и {2,3} не равны, ни подмножества друг друга, ни соответственно, наборы не являются подходящими аргументами для функций, которые принимают множества. Например [min\(\)](#), [max\(\)](#) и [sorted\(\)](#) дают неопределенные результаты при наличии множеств в качестве входных данных.

Типы данных, не имеющие реализованных методов сравнения, поэтому они наследуют поведение

## Ловушки цепочек сравнения).

Цепочки сравнения выглядят очень разумно, есть пара подводных камней, на которые необходимо

### Ловушки.

При проверке равенства `a == b == c`, можно использовать цепочку сравнения `a == b == c`. А как проверить, ВСЕ ли значения равны? Это `a != b != c`, и попадаем в первую ловушку!

Но это не так просто, как кажется. Проблема здесь в том, что `a != b != c` - это `a != b and b != c`, т.е. проверяет, что `b` отличается от `a` и от `c`, но ничего не говорит о том, как связаны `a` и `c`.

С математической точки зрения, `!=` не является транзитивным, т. е. знание того, как `a` относится к `b`, и знание того, как `b` относится к `c`, не говорит о том, как `a` относится к `c`. Что касается транзитивного примера, можно взять оператор равенства `==`. Если `a == b and b == c`, то также верно, что `a == c`.

## Непостоянное значение в выражении.

Напомним, что в цепочке сравнений, таких как `a < b < c`, значение `b` в середине выражения вычисляется только один раз, тогда как в расширенном выражении `a < b and b < c` значение `b` вычисляется дважды.

Если `b` содержит что-то непостоянное или выражение с побочными эффектами, то эти два выражения не эквивалентны.

Этот пример показывает разницу в количестве оценок значения в середине выражения:

```
def f():
    print("run")
    return 3

>>> 1 < f() < 5
# run
# True
>>> 1 < f() and f() < 5
# run
# run
# True
```

Следующий пример показывает, что выражение типа `1 < f() < 0` может принимать значение `True`, когда оно записано развернуто:

```
lst = [-2, 2]

def f():
    global lst
    Вверх lst[::-1]
    return lst[0]
```

```
>>> 1 < f() and f() < 0
# True
>>> 1 < f() < 0
```

РЕКЛАМА · 18+

practicum.yandex.ru

**Бесплатное занятие английским в Яндекс Практикуме**

Полноценное занятие с преподавателем, а не презентация курсов

Устный тест на уровень языка

Практика английского

Узнать больше

из, который переворачивает список.

е будет быть истинным, пример просто показывает, что цепочка сравнения и развернутое ы.

и сравнения.

твительно естественно, но в некоторых конкретных случаях она не так хороша. Это о лучше избегать цепочки, в которых операторы не "выровнены", например:

а < b > c читается как "проверим, b больше, чем a и c?", но лучше эту цепочку b > max(a, c).

которые просто сбивают с толку:

in и not in являются операторами сравнения, следовательно их также можно связать с другими операторами. Это создаст странные ситуации, такие как:

```
>>> a = 3
>>> lst = [3, 5]
>>> a in lst == True
# False
```

Более подробно, читайте материал ["Смешивание операторов в цепочках сравнений в Python"](#).

Примеры использования цепочек сравнения.

```
>>> a = 1
>>> b = 2
>>> c = 3
>>> a < b < c
# True
```

Когда Python видит два оператора сравнения подряд, как в a < b < c, он ведет себя так, как если бы было написано что-то вроде a < b and b < c, за исключением того, что b вычисляется только один раз. Такое поведение актуально, если, например, b является выражением, подобным вызову функции.

Другой пример использования - когда необходимо убедиться, что все три значения одинаковы:

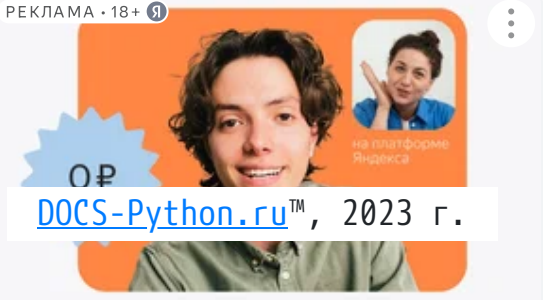
```
>>> a = b = 1
>>> c = 2
>>> if a == b == c:
...     print("все равны")
... else:
...     print("некоторые отличаются")

# некоторые отличаются

>>> c = 1
>>> if a == b == c:
...     print("все равны")
...     print("некоторые отличаются")
```


# все равны

На самом деле можно связать произвольное количество операторов сравнения в цепочку? Например, `a == b == c == d == e` проверяет, совпадают ли все пять переменных, в то время как `a < b < c < d < e` проверяет, есть ли строго .



(Внимание! При копировании материала ссылка на источник обязательна)

[@docs\\_python\\_ru](#)

 practicum.yandex.ru

**Бесплатное  
занятие  
английским  
в Яндекс  
Практикуме**

Полноценное занятие  
с преподавателем, а не  
презентация курсов

Устный тест на уровень  
языка

>

Практика английского

>

Узнать больше

Вверх