**Сравнение в последовательностях**

Сравнение идет в лексикографическом порядке (как в словаре). Т.е. сравниваются первые элементы, если они разные, то больше та последовательность, у которой сравниваемый элемент больше. Если элементы равны, переходим к сравнению следующих элементов.

Численные типы сравниваются по их значению чисел. Т.е. при сравнении 1 и 1.5, рассматриваем 1 как 1.0.

Сравнение прочих разных типов порождает исключение TypeError[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/TypeError?topicparent=Cintro.PythonSort).

(1, 2, 3) < (1, 2, 4)

[1, 2, 3] < [1, 2, 4]

'ABC' < 'C' < 'Pascal' < 'Python'

(1, 2, 3, 4) < (1, 2, 4)

(1, 2) < (1, 2, -1)

(1, 2, 3) == (1.0, 2.0, 3.0)

(1, 2, ('aa', 'ab')) < (1, 2, ('abc', 'a'), 4)

**Сортировка**

* <https://developers.google.com/edu/python/sorting>

**sort и sorted**

**sort(\*, *key=None, reverse=False*)** - стабильная сортировка самого списка

**sorted(\*, *key=None, reverse=False*)** - создается новый отсортированный список, старый остается без изменения

По умолчанию используются стандартные операторы сравнения и сортирует по возрастанию.

a = [3, 6, 8, 2, 78, 1, 23, 45, 9]

b = sorted(a)

print(a) # [3, 6, 8, 2, 78, 1, 23, 45, 9]

print(b) # [1, 2, 3, 6, 8, 9, 23, 45, 78]

a.sort()

print(a) # [1, 2, 3, 6, 8, 9, 23, 45, 78]

**reverse=True - в обратном порядке**

>>> sorted(a, reverse=True)

[78, 45, 23, 9, 8, 6, 3, 2, 1]

>>> a.sort(reverse=True)

>>> a

[78, 45, 23, 9, 8, 6, 3, 2, 1]

**key=функция - как сравнивать объекты**

Отсортируем список по значению модуля каждого числа. Модуль берется функцией **abs()**

a = [3, 6, -8, 2, -78, 1, 23, -45, 9]

b = sorted(a, key=abs)

print(a) # [3, 6, -8, 2, -78, 1, 23, -45, 9]

print(b) # [1, 2, 3, 6, -8, 9, 23, -45, -78]

a.sort(key=abs)

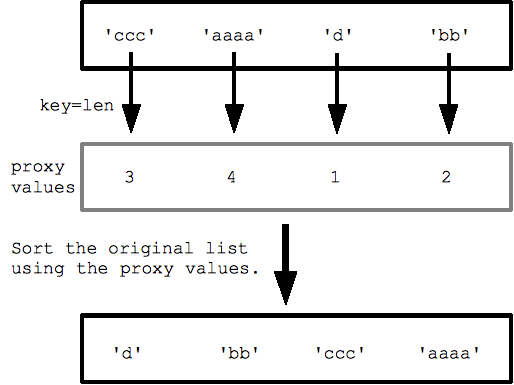
print(a) # [1, 2, 3, 6, -8, 9, 23, -45, -78]

Другой пример. Отсортируем строки по длине.

strs = ['ccc', 'aaaa', 'd', 'bb']

print sorted(strs, key=len) # ['d', 'bb', 'ccc', 'aaaa']

Ключевая функция (имя которой передается в аргументе key) должна принимать 1 значение (value) и возвращать 1 значение (proxy value). По этому proxy value проходит сортировка.



Отсортируем список строк БЕЗ учета регистра. Для этого будем сортировать строки, которые приведены к нижнему регистру.

strs = ['aa', 'BB', 'zz', 'CC']

print sorted(strs) # ['BB', 'CC', 'aa', 'zz'] (case sensitive)

print sorted(strs, key=str.lower) # ['aa', 'BB', 'CC', 'zz']

Напишем функцию сами:

# Say we have a list of strings we want to sort by the last letter of the string.

strs = ['xc', 'zb', 'yd' ,'wa']

# Write a little function that takes a string, and returns its last letter.

# This will be the key function (takes in 1 value, returns 1 value).

def MyFn(s):

return s[-1]

# Now pass key=MyFn to sorted() to sort by the last letter:

print sorted(strs, key=MyFn) ## ['wa', 'zb', 'xc', 'yd']

Еще пример: отсортируем разнотипные числа как числа.

sorted(["1.3", 7.5, "5", 4, "2.4", 1], key=float)

**Как при сортировке узнают что больше, а что меньше?**

При сортировке объектов для пары объектов вызывается функция **\_\_cmp\_\_** (о ней расскажем позже, как и о том, почему в названии функции \_\_).

Если функция возвратила 0, то считается, что объекты равны. Если 1, то первый объект больше второго, а если -1, то меньше.

**Алгоритм сортировки**

Саммерфильд, стр 172.

В языке Python реализован адаптивный алгоритм устойчивой сортировки со слиянием, который отличается высокой скоростью и интеллектуальностью и особенно хорошо подходит для сортировки частично отсортированных списков, что встречается достаточно часто.

«адаптивный» - алгоритм сортировки адаптируется под определенные условия, например, учитывает наличие частичной сортировки данных.

«устойчивый» - одинаковые элементы не перемещаются относительно друг друга (в конце концов, в этом нет никакой необходимости)

«сортировка со слиянием» – это общее название используемых алгоритмов сортировки.

Алгоритм был создан Тимом Петерсом (Tim Peters). Интересное описание и обсуждение алгоритма можно найти в файле listsort.txt, который поставляется в составе исходных программных кодов Python.

**Сравнение и сортировка в python 3.x**

Лутц, стр 261

Сравнивание и сортировка в Python 3.0: В Python 2.6 и в более ранних версиях сравнивание выполняется по-разному для объектов разных типов (например, списков и строк) – язык задает способ упорядочения различных типов, который можно признать скорее детерминистским, чем эстетичным. Этот способ упорядочения основан на именах типов, вовлеченных в операцию сравнения, например любые целые числа всегда меньше любых строк, потому что строка "int" меньше, чем строка "str".

При выполнении операции сравнения никогда не выполняется преобразование типов объектов, за исключением сравнивания объектов числовых типов.

В Python 3.0 такой порядок был изменен: попытки сравнивания объектов различных типов возбуждают исключение – вместо сравнивания по названиям типов. Так как метод сортировки использует операцию сравнения, это означает, что инструкция [1, 2, ‘spam’].sort() будет успешно выполнена в Python 2.x, но возбудит исключение в версии Python 3.0 и выше. Кроме того, в версии Python 3.0 больше не поддерживается возможность передачи методу sort произвольной функции сравнения, для реализации иного способа упорядочения. Чтобы обойти это ограничение, предлагается использовать именованный аргумент key=func, в котором предусматривать возможность трансформации значений в процессе сортировки, и применять именованный аргумент reverse=True для сортировки по убыванию. То есть фактически выполнять те же действия, которые раньше выполнялись внутри функции сравнения.

**Задачи**

**По убыванию модуля числа**

Отсортировать целые числа по убыванию модуля числа

|  |  |
| --- | --- |
| [**Вход**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonSort?sortcol=0&table=1&up=0#sorted_table) | [**Выход**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonSort?sortcol=1&table=1&up=0#sorted_table) |
| [3, 6, -8, 2, -78, 1, 23, -45, 9] | [-78, -45, 23, 9, -8, 6, 3, 2, 1] |

**Координаты точек на плоскости по удалению от центра координат**

Даны точки на плоскости ХУ. Отсортировать по удалению от центра координат. При одинаковом расстоянии сначала идет та точка, у которой меньше координата х. При равных координатах х, сначала идет точка с меньшей координатой у.

**90-60-90**

На факультетском конкурсе красоты юные физики решили подойти к проблеме нахождения самой красивой девушки с точки зрения точных наук. У каждой участницы были измерены три параметра: обхват груди (ОГ), обхват талии (ОТ), обхват бедер (ОБ). Было решено за идеал взять значения 90-60-90 (ОГи, ОТи,ОБи), а участниц отранжировать по мере удаления от идеала.  
Мерой расхождения с идеалом служит сумма квадратов отклонений (ОГ-ОГи)2+(ОТ-ОТи)2+(ОБ-ОБи)2.  
При равной мере отклонения ближе к идеалу считаются участницы с большей грудью.  
Если и обхват груди оказался одинаковым, предпочтение отдается меньшему обхвату талии.  
При равных ОГ и ОТ предпочтение отдается большему ОБ.

|  |  |
| --- | --- |
| [**Вход**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonSort?sortcol=0&table=2&up=0#sorted_table) | [**Выход**](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonSort?sortcol=1&table=2&up=0#sorted_table) |
| 90 62 92 90 60 88 90 60 92 90 58 92 91 61 91 | 91 61 91 90 60 92 90 60 88 90 58 92 90 62 92 |

-- [TatyanaDerbysheva](http://acm.mipt.ru/twiki/bin/view/Main/TatyanaDerbysheva) - 23 Sep 2017