**Сортировка**

**Источники**

* (Google for Education / Python Course / Sorting)[<https://developers.google.com/edu/python/sorting>]
* (Sorting HowTo)[<https://docs.python.org/3/howto/sorting.html>]

**Сравнение**

Чтобы отсортировать объекты, для них нужно знать какой из них больше, какой меньше.

Числа сравнивают как числа:

3 > 1

-3.5 < 5.24

-1 < 1.5

Строки сравнивают, как слова в словаре. Какое слово в словаре идет раньше, то и меньше.

'Abc' < 'abc'

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

То есть сравнивают по символам, начиная с первого.

Списки (list) и кортежи (tuple) сравнивают по элементам.

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

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

(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)

Если сравнивать типы, которые не сравниваются, получается исключение TypeError.

**Функции sort и sorted**

Для сортировки используют стандартные функции sort (сортировка списка) и sorted (сортировка последовательности)

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

()[../images/sorted-key.png]

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']

**Как написать функцию для сравнения**

Пишем функцию, которая для 1 объекта (аргумента) возвращает proxy value, по которым этот объект будут сравнивать в сортировке

# 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)

**Дополнительные материалы**

**Как обеспечить сравниение объектов классов**

Об этом будет рассказано позже, когда будем говорить о классах и объектах.

**Алгоритмы 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**

**1. Отсортировать последовательность целых чисел по возрастанию**

**2. Отсортировать последовательность целых чисел по убыванию**

**3. Отсортировать последовательность целых чисел по убыванию модуля числа**

| **Input** | **Output** |
| --- | --- |
| 3 6 -8 2 -78 1 23 -45 9 | -78 -45 23 9 -8 6 3 2 1 |

**4. Даты**

На каждой строке написана дата в формате dd/mm/yyyy. Выведите даты в порядке возрастания в формате yyyy/mm/dd.

Input:

01/12/1910

31/05/0861

22/06/2014

Output:

0861/05/31

1910/12/01

2014/06/22

**5. Рост, вес**

Даны рос и вес каждого человека. Отсортировать по уменьшению роста. При одинаковом росте сначала печатать больший вес.

Input:

156 66.2

178 66.8

178 56.4

Output:

178 66.8

178 56.4

156 66.2

**6. Студенты**

Дан список студентов и их средний балл. Отсортировать по среднему баллу по убыванию, при одинаковом среднем балле - по алфавиту.

Input:

alla 7.5

mike 8.2

john 7.5

boris 9

Output:

boris 9.0

mike 8.2

alla 7.5

john 7.5

**7. Точки по удалению от начала координат**

Даны координаты точек. Отсортировать их по удалению от точки (0,0).

Если две точки находятся на одном расстоянии от начала координат, то сначала выведите точку с меньшим значением координаты х. Если и совпали и расстояния, и значения х-координаты, сначала выведите точку с меньшим значением у-координаты.

**8 Позиция**

Даны числа. Напечатайте их по возрастанию четных чисел, нечетные остаются на своих местах

+

Input:

5 122 3 26 48

Output:

5 26 3 48 122