* [Сортировка](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonSorTurtle" \l "_q__k_q_mc)
  + [Сравнение объектов](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonSorTurtle#__c_pgpkg_qd_gm_q_)
  + [Функции sort и sorted](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonSorTurtle#__pmekk_sort_k_sorted)
    - [reverse=True - в обратном порядке](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonSorTurtle#reverse_True___qd_c_pqo_rq_sfmg)
    - [key=функция - как сравнивать объекты](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonSorTurtle#key_h_pmeks_mcm___c_pk_c___qd_gm)
    - [Пример: Сортируем по росту и весу](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonSorTurtle#__kog___q__k__go_rq__q____k__g__)
* [Задачи](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonSorTurtle#_cfc_k)
  + [Задача 1. Отсортировать отрезки по возрастанию длины](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonSorTurtle#_cfc_c_1____q__k_q_c___q__g_mk_r)
  + [Задача 3a. По возрастанию цветные отрезки](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonSorTurtle#_cfc_c_3a__q__q__c__cpk__e_g_p_g)
  + [Задача 4. z-буфер для квадратов](http://acm.mipt.ru/twiki/bin/view/Cintro/PythonSorTurtle#_cfc_c_4_z_d_hg__fns_m_cf_c_q_)

**Сортировка**

**Сравнение объектов**

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[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/TypeError?topicparent=Cintro.PythonSorTurtle).

**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]

По умолчанию значение reverse=False.

**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']

Еще пример: отсортируем разнотипные числа как числа.

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

key=float позволяет все объекты сортировать как float.

**Как написать функцию для сравнения**

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

**Пример: Пример: сортируем строки по последним буквам**

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']

Написали функцию MyFn[?](http://acm.mipt.ru/twiki/bin/edit/Cintro/MyFn?topicparent=Cintro.PythonSorTurtle), которая по переданному объекту сортировке (строка) возвращает proxy value (последний символ), которое используется в сортировке.

**Пример: Сортируем по росту и весу**

Даны рост (см) и вес (кг) каждого человека. По одному человеку на строку. Отсортируем людей.

a = [(166, 55.2), (157, 55.2), (170, 55.2), (175, 90), (166, 73), (180, 73)]

print(a)

b = sorted(a)

print(b)

# [(157, 55.2), (166, 55.2), (166, 73), (170, 55.2), (175, 90), (180, 73)]

Получилась сортировка по возрастанию роста, при одинаковом росте сортируем по весу.

Теперь напишем функцию, которая будет смотреть только на вес и игнорировать рост.

def weight(t):

h = t[0]

w = t[1]

return w # или return t[1]

b = sorted(a, key=weight)

print(b)

# # [(166, 55.2), (157, 55.2), (170, 55.2), (166, 73), (180, 73), (175, 90)]

Заметим, что люди с одинаковым весом идут в том же порядке, что и в списке до сортировки: при весе 55.2 сначала 166, потом 157. При весе 73 сначала 166, потом 180.

Сортировка в питоне стабильная, то есть равные по одному признаку элементы будут идти в том же порядке, что и до сортировки.

Отсортируем по весу по убыванию. Не будем использовать reverse = True, сделаем это через функцию.

def weight\_decr(t):

h = t[0]

w = t[1]

return -w # или return -t[1]

b = sorted(a, key=weight\_decr)

print(b)

# [(175, 90), (166, 73), (180, 73), (166, 55.2), (157, 55.2), (170, 55.2)]

Отсортируем по весу (по возрастанию), а при равном весе - по росту (по возрастанию).

Для этого в функции, которая возвращает proxy values вернем (вес, рост).

def wh1(t):

h = t[0]

w = t[1]

return (w, h) # обязательно возвратить tuple

b = sorted(a, key=wh1)

print(b)

# [(157, 55.2), (166, 55.2), (170, 55.2), (166, 73), (180, 73), (175, 90)]

Как отсортировать по возрастанию веса и при равном весе по\_убыванию роста?

def wh2(t):

h = t[0]

w = t[1]

return (w, -h) # обязательно возвратить tuple

b = sorted(a, key=wh2)

print(b)

# [(170, 55.2), (166, 55.2), (157, 55.2), (180, 73), (166, 73), (175, 90)]

То же самое через lambda-функцию (Мьянме не читать!!!):

b = sorted(a, key=lambda t: (t[1], -t[0]))

print(b)

# [(170, 55.2), (166, 55.2), (157, 55.2), (180, 73), (166, 73), (175, 90)]

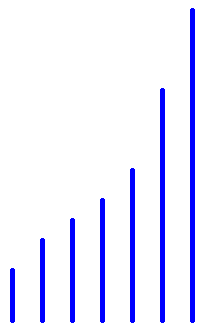
**Задачи**

**Задача 1. Отсортировать отрезки по возрастанию длины**

Даны длины отрезков (целые числа) в одну строку через пробел.

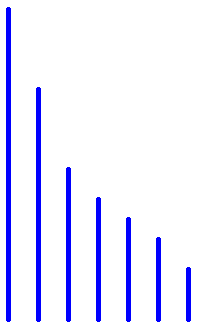
Нарисовать эти отрезки вертикально от меньшего к большему.

150 230 50 100 80 120 310



**Задача 2. Отсортировать отрезки по убыванию длины**

150 230 50 100 80 120 310



**Задача 3. По возрастанию цветные отрезки**

Дано количество отрезков. Далее на строке написана длина отрезка и его цвет. Нарисуйте эти отрезки по возрастанию.

7

150 blue

230 green

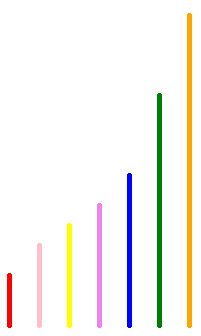
50 red

100 yellow

80 pink

120 violet

310 orange



**Задача 3a. По возрастанию цветные отрезки**

Дано количество отрезков. Далее на строке написана длина отрезка и его цвет. Нарисуйте эти отрезки по возрастанию.

7

blue 150

green 230

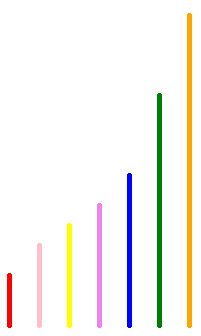
red 50

yellow 100

pink 80

violet 120

orange 310



**Задача 4. z-буфер для квадратов**

-- [TatyanaDerbysheva](http://acm.mipt.ru/twiki/bin/view/Main/TatyanaDerbysheva) - 21 Mar 2019