Функция sorted

Функция `sorted` возвращает новый отсортированный список, который получен из итерируемого объекта, который был передан как аргумент. Функция также поддерживает дополнительные параметры, которые позволяют управлять сортировкой.

Первый аспект, на который важно обратить внимание - sorted всегда возвращает список.

Если сортировать список элементов, то возвращается новый список:

```
In [1]: list_of_words = ['one', 'two', 'list', '', 'dict']

In [2]: sorted(list_of_words)
Out[2]: ['', 'dict', 'list', 'one', 'two']
```

При сортировке кортежа также возвращается список:

```
In [3]: tuple_of_words = ('one', 'two', 'list', '', 'dict')

In [4]: sorted(tuple_of_words)
Out[4]: ['', 'dict', 'list', 'one', 'two']
```

Сортировка множества:

```
In [5]: set_of_words = {'one', 'two', 'list', '', 'dict'}

In [6]: sorted(set_of_words)
Out[6]: ['', 'dict', 'list', 'one', 'two']
```

Сортировка строки:

```
In [7]: string_to_sort = 'long string'
```

```
In [8]: sorted(string_to_sort)
Out[8]: [' ', 'g', 'g', 'i', 'l', 'n', 'n', 'o', 'r', 's', 't']
```

Если передать sorted словарь, функция вернет отсортированный список ключей:

```
In [9]: dict_for_sort = {
   ...:          'id': 1,
   ...:          'name': 'London',
   ...:          'IT_VLAN': 320,
   ...:          'User_VLAN': 1010,
   ...:          'Mngmt_VLAN': 99,
   ...:          'to_name': None,
   ...:          'to_id': None,
   ...:          'port': 'G1/0/11'
   ...: }

In [10]: sorted(dict_for_sort)
Out[10]:
['IT_VLAN',
 'Mngmt_VLAN',
 'User_VLAN',
 'id',
 'name',
 'port',
 'to_id',
 'to_name']
```

reverse

Флаг reverse позволяет управлять порядком сортировки. По умолчанию сортировка будет по возрастанию элементов.

Указав флаг reverse, можно поменять порядок:

```
In [11]: list_of_words = ['one', 'two', 'list', '', 'dict']

In [12]: sorted(list_of_words)
Out[12]: ['', 'dict', 'list', 'one', 'two']
```

```
In [13]: sorted(list_of_words, reverse=True)
Out[13]: ['two', 'one', 'list', 'dict', '']
```

key

С помощью параметра key можно указывать, как именно выполнять сортировку. Параметр key ожидает функцию, с помощью которой должно быть выполнено сравнение.

Например, таким образом можно отсортировать список строк по длине строки:

```
In [14]: list_of_words = ['one', 'two', 'list', '', 'dict']

In [15]: sorted(list_of_words, key=len)
Out[15]: ['', 'one', 'two', 'list', 'dict']
```

Если нужно отсортировать ключи словаря, но при этом игнорировать регистр строк:

```
In [16]: dict_for_sort = {
    ...:          'id': 1,
    ...:          'name':'London',
    ...:          'IT_VLAN':320,
    ...:          'User_VLAN':1010,
    ...:          'Mngmt_VLAN':99,
    ...:          'to_name': None,
    ...:          'to_id': None,
    ...:          'port':'G1/0/11'
    ...: }

In [17]: sorted(dict_for_sort, key=str.lower)
Out[17]:
['id',
 'IT_VLAN',
 'Mngmt_VLAN',
 'name',
 'port',
 'to_id',
 'to_name',
 'User_VLAN']
```

Параметру key можно передавать любые функции, не только встроенные. Также тут удобно использовать анонимную функцию lambda.

С помощью параметра key можно сортировать объекты не по первому элементу, а по любому другому. Но для этого надо использовать или функцию lambda, или специальные функции из модуля operator.

Например, чтобы отсортировать список кортежей из двух элементов по второму элементу, надо использовать такой прием:

```
In [18]: from operator import itemgetter

In [19]: list_of_tuples = [('IT_VLAN', 320),
    ...:  ('Mngmt_VLAN', 99),
    ...:  ('User_VLAN', 1010),
    ...:  ('DB_VLAN', 11)]

In [20]: sorted(list_of_tuples, key=itemgetter(1))
Out[20]: [('DB_VLAN', 11), ('Mngmt_VLAN', 99), ('IT_VLAN', 320),
('User_VLAN', 1010)]
```

Пример сортировки разных объектов

Сортировка выполняется по первому элементу, например, по первому символу в списке строк, если он одинаковый, по второму и так далее. Сортировка выполняется по коду Unicode символа. Для символов из одного алфавита, это значит что сортировка по сути будет по алфавиту.

Пример сортировки списка строк:

```
In [6]: data = ["test1", "test2", "text1", "text2"]

In [7]: sorted(data)
Out[7]: ['test1', 'test2', 'text1', 'text2']
```

Некоторые данные будут сортироваться неправильно, например, список IP-адресов:

```
    In [11]: ip_list = ["10.1.1.1", "10.1.10.1", "10.1.2.1",
"10.1.11.1"]

    In [12]: sorted(ip_list)
    Out[12]: ['10.1.1.1', '10.1.10.1', '10.1.11.1', '10.1.2.1']
```

Это происходит потому используется лексикографическая сортировка. Чтобы в данном случае сортировка была нормальной, надо или использовать отдельный модуль с натуральной сортировкой (модуль natsort) или сортировать, например, по двоичному/десятичному значению адреса.

Пример сортировки IP-адресов по двоичному значению. Сначала создаем функцию, которая преобразует IP-адреса в двоичный формат:

```
    In [15]: def bin_ip(ip):
        ...:     octets = [int(o) for o in ip.split(".")]
        ...:     return ("{:08b}"*4).format(*octets)
        ...:

    In [16]: bin_ip("10.1.1.1")
    Out[16]: '00001010000000010000000100000001'

    In [17]: bin_ip("160.1.1.1")
    Out[17]: '10100000000000010000000100000001'
```

Сортировка с использованием функции bin_ip:

```
    In [18]: ip_list = ["10.1.1.1", "10.1.10.1", "10.1.2.1",
"10.1.11.1"]

    In [19]: sorted(ip_list, key=bin_ip)
    Out[19]: ['10.1.1.1', '10.1.2.1', '10.1.10.1', '10.1.11.1']
```

> **ПРИМЕЧАНИЕ**

> Также дальше будет рассматриваться модуль ipaddress, который позволит создавать специальные объекты, которые соответствуют IP-адресу и они уже сортируются правильно по десятичному значению.

Теги: functions sorted

Отредактировать эту страницу

*Последнее обновление **4 сент. 2023 г.** от **Stavis***