Sorting HOW TO

Author: Andrew Dalke and Raymond Hettinger
Release: 0.1

Python lists have a built-in :meth:`list.sort` method that modifies the list in-place. There is also a :func:`sorted` built-in function that builds a new sorted list from an iterable.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 10); *backlink***
>
> Unknown interpreted text role "meth".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 10); *backlink***
>
> Unknown interpreted text role "func".

In this document, we explore the various techniques for sorting data using Python.

Sorting Basics

A simple ascending sort is very easy: just call the :func:`sorted` function. It returns a new sorted list:

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 20); *backlink***
>
> Unknown interpreted text role "func".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 23)**
>
> Unknown directive type "doctest".
>
> ```
> .. doctest::
>
>     >>> sorted([5, 2, 3, 1, 4])
>     [1, 2, 3, 4, 5]
> ```

You can also use the :meth:`list.sort` method. It modifies the list in-place (and returns `None` to avoid confusion). Usually it's less convenient than :func:`sorted` - but if you don't need the original list, it's slightly more efficient.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 28); *backlink***
>
> Unknown interpreted text role "meth".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 28); *backlink***
>
> Unknown interpreted text role "func".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 33)**
>
> Unknown directive type "doctest".
>
> ```
> .. doctest::
>
>     >>> a = [5, 2, 3, 1, 4]
>     >>> a.sort()
>     >>> a
>     [1, 2, 3, 4, 5]
> ```

Another difference is that the :meth:`list.sort` method is only defined for lists. In contrast, the :func:`sorted` function accepts any iterable.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 40); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 40); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 43)

Unknown directive type "doctest".

```
.. doctest::

    >>> sorted({1: 'D', 2: 'B', 3: 'B', 4: 'E', 5: 'A'})
    [1, 2, 3, 4, 5]
```

Key Functions

Both :meth:`list.sort` and :func:`sorted` have a *key* parameter to specify a function (or other callable) to be called on each list element prior to making comparisons.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 51); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 51); *backlink*

Unknown interpreted text role "func".

For example, here's a case-insensitive string comparison:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 57)

Unknown directive type "doctest".

```
.. doctest::

    >>> sorted("This is a test string from Andrew".split(), key=str.lower)
    ['a', 'Andrew', 'from', 'is', 'string', 'test', 'This']
```

The value of the *key* parameter should be a function (or other callable) that takes a single argument and returns a key to use for sorting purposes. This technique is fast because the key function is called exactly once for each input record.

A common pattern is to sort complex objects using some of the object's indices as keys. For example:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 70)

Unknown directive type "doctest".

```
.. doctest::

    >>> student_tuples = [
    ...     ('john', 'A', 15),
    ...     ('jane', 'B', 12),
    ...     ('dave', 'B', 10),
    ... ]
    >>> sorted(student_tuples, key=lambda student: student[2])   # sort by age
    [('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
```

The same technique works for objects with named attributes. For example:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 82)

Unknown directive type "doctest".

```
.. doctest::

    >>> class Student:
    ...     def __init__(self, name, grade, age):
    ...         self.name = name
    ...         self.grade = grade
    ...         self.age = age
    ...     def __repr__(self):
    ...         return repr((self.name, self.grade, self.age))

    >>> student_objects = [
    ...     Student('john', 'A', 15),
    ...     Student('jane', 'B', 12),
    ...     Student('dave', 'B', 10),
    ... ]
    >>> sorted(student_objects, key=lambda student: student.age)   # sort by age
    [('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
```

Operator Module Functions

The key-function patterns shown above are very common, so Python provides convenience functions to make accessor functions easier and faster. The :mod:`operator` module has :func:`~operator.itemgetter`, :func:`~operator.attrgetter`, and a :func:`~operator.methodcaller` function.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 103); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 103); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 103); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 103); *backlink*

Unknown interpreted text role "func".

Using those functions, the above examples become simpler and faster:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 110)

Unknown directive type "doctest".

```
.. doctest::

    >>> from operator import itemgetter, attrgetter

    >>> sorted(student_tuples, key=itemgetter(2))
    [('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]

    >>> sorted(student_objects, key=attrgetter('age'))
    [('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
```

The operator module functions allow multiple levels of sorting. For example, to sort by *grade* then by *age*:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 123)

Unknown directive type "doctest".

```
.. doctest::

    >>> sorted(student_tuples, key=itemgetter(1,2))
    [('john', 'A', 15), ('dave', 'B', 10), ('jane', 'B', 12)]

    >>> sorted(student_objects, key=attrgetter('grade', 'age'))
    [('john', 'A', 15), ('dave', 'B', 10), ('jane', 'B', 12)]
```

Ascending and Descending

Both :meth:`list.sort` and :func:`sorted` accept a *reverse* parameter with a boolean value. This is used to flag descending sorts. For example, to get the student data in reverse *age* order:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 134); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 134); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 138)

Unknown directive type "doctest".

```
.. doctest::

    >>> sorted(student_tuples, key=itemgetter(2), reverse=True)
    [('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)]

    >>> sorted(student_objects, key=attrgetter('age'), reverse=True)
    [('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)]
```

Sort Stability and Complex Sorts

Sorts are guaranteed to be stable. That means that when multiple records have the same key, their original order is preserved.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 153)

Unknown directive type "doctest".

```
.. doctest::

    >>> data = [('red', 1), ('blue', 1), ('red', 2), ('blue', 2)]
    >>> sorted(data, key=itemgetter(0))
    [('blue', 1), ('blue', 2), ('red', 1), ('red', 2)]
```

Notice how the two records for *blue* retain their original order so that `('blue', 1)` is guaranteed to precede `('blue', 2)`.

This wonderful property lets you build complex sorts in a series of sorting steps. For example, to sort the student data by descending *grade* and then ascending *age*, do the *age* sort first and then sort again using *grade*:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 166)

Unknown directive type "doctest".

```
.. doctest::

    >>> s = sorted(student_objects, key=attrgetter('age'))     # sort on secondary key
    >>> sorted(s, key=attrgetter('grade'), reverse=True)       # now sort on primary key, descendi
    [('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
```

This can be abstracted out into a wrapper function that can take a list and tuples of field and order to sort them on multiple passes.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-

main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 175)

Unknown directive type "doctest".

```
.. doctest::

    >>> def multisort(xs, specs):
    ...     for key, reverse in reversed(specs):
    ...         xs.sort(key=attrgetter(key), reverse=reverse)
    ...     return xs

    >>> multisort(list(student_objects), (('grade', True), ('age', False)))
    [('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
```

The Timsort algorithm used in Python does multiple sorts efficiently because it can take advantage of any ordering already present in a dataset.

The Old Way Using Decorate-Sort-Undecorate

This idiom is called Decorate-Sort-Undecorate after its three steps:

- First, the initial list is decorated with new values that control the sort order.
- Second, the decorated list is sorted.
- Finally, the decorations are removed, creating a list that contains only the initial values in the new order.

For example, to sort the student data by *grade* using the DSU approach:

```
>>> decorated = [(student.grade, i, student) for i, student in enumerate(student_objects)]
>>> decorated.sort()
>>> [student for grade, i, student in decorated]               # undecorate
[('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)]
```

This idiom works because tuples are compared lexicographically; the first items are compared; if they are the same then the second items are compared, and so on.

It is not strictly necessary in all cases to include the index *i* in the decorated list, but including it gives two benefits:

- The sort is stable -- if two items have the same key, their order will be preserved in the sorted list.
- The original items do not have to be comparable because the ordering of the decorated tuples will be determined by at most the first two items. So for example the original list could contain complex numbers which cannot be sorted directly.

Another name for this idiom is Schwartzian transform, after Randal L. Schwartz, who popularized it among Perl programmers.

Now that Python sorting provides key-functions, this technique is not often needed.

The Old Way Using the *cmp* Parameter

Many constructs given in this HOWTO assume Python 2.4 or later. Before that, there was no :func:`sorted` builtin and :meth:`list.sort` took no keyword arguments. Instead, all of the Py2.x versions supported a *cmp* parameter to handle user specified comparison functions.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 233); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 233); *backlink*

Unknown interpreted text role "meth".

In Py3.0, the *cmp* parameter was removed entirely (as part of a larger effort to simplify and unify the language, eliminating the conflict between rich comparisons and the :meth:`__cmp__` magic method).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 238); *backlink*

Unknown interpreted text role "meth".

In Py2.x, sort allowed an optional function which can be called for doing the comparisons. That function should take two arguments to be compared and then return a negative value for less-than, return zero if they are equal, or return a positive value for greater-than. For example, we can do:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 247)

Unknown directive type "doctest".

```
.. doctest::

    >>> def numeric_compare(x, y):
    ...     return x - y
    >>> sorted([5, 2, 4, 1, 3], cmp=numeric_compare) # doctest: +SKIP
    [1, 2, 3, 4, 5]
```

Or you can reverse the order of comparison with:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 256)

Unknown directive type "doctest".

```
.. doctest::

    >>> def reverse_numeric(x, y):
    ...     return y - x
    >>> sorted([5, 2, 4, 1, 3], cmp=reverse_numeric) # doctest: +SKIP
    [5, 4, 3, 2, 1]
```

When porting code from Python 2.x to 3.x, the situation can arise when you have the user supplying a comparison function and you need to convert that to a key function. The following wrapper makes that easy to do:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 267)

Unknown directive type "testcode".

```
.. testcode::

    def cmp_to_key(mycmp):
        'Convert a cmp= function into a key= function'
        class K:
            def __init__(self, obj, *args):
                self.obj = obj
            def __lt__(self, other):
                return mycmp(self.obj, other.obj) < 0
            def __gt__(self, other):
                return mycmp(self.obj, other.obj) > 0
            def __eq__(self, other):
                return mycmp(self.obj, other.obj) == 0
            def __le__(self, other):
                return mycmp(self.obj, other.obj) <= 0
            def __ge__(self, other):
                return mycmp(self.obj, other.obj) >= 0
            def __ne__(self, other):
                return mycmp(self.obj, other.obj) != 0
        return K
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 288)

Unknown directive type "doctest".

```
.. doctest::
    :hide:

    >>> sorted([5, 2, 4, 1, 3], key=cmp_to_key(reverse_numeric))
    [5, 4, 3, 2, 1]
```

To convert to a key function, just wrap the old comparison function:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 296)

Unknown directive type "testsetup".

```
.. testsetup::
```

```
from functools import cmp_to_key
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 300)

Unknown directive type "doctest".

```
.. doctest::

    >>> sorted([5, 2, 4, 1, 3], key=cmp_to_key(reverse_numeric))
    [5, 4, 3, 2, 1]
```

In Python 3.2, the :func:`functools.cmp_to_key` function was added to the :mod:`functools` module in the standard library.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 305); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 305); *backlink*

Unknown interpreted text role "mod".

Odd and Ends

- For locale aware sorting, use :func:`locale.strxfrm` for a key function or :func:`locale.strcoll` for a comparison function.

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 311); *backlink*

  Unknown interpreted text role "func".

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 311); *backlink*

  Unknown interpreted text role "func".

- The *reverse* parameter still maintains sort stability (so that records with equal keys retain the original order). Interestingly, that effect can be simulated without the parameter by using the builtin :func:`reversed` function twice:

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 314); *backlink*

  Unknown interpreted text role "func".

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 319)

  Unknown directive type "doctest".

  ```
  .. doctest::

    >>> data = [('red', 1), ('blue', 1), ('red', 2), ('blue', 2)]
    >>> standard_way = sorted(data, key=itemgetter(0), reverse=True)
    >>> double_reversed = list(reversed(sorted(reversed(data), key=itemgetter(0))))
    >>> assert standard_way == double_reversed
    >>> standard_way
    [('red', 1), ('red', 2), ('blue', 1), ('blue', 2)]
  ```

- The sort routines are guaranteed to use :meth:`__lt__` when making comparisons between two objects. So, it is easy to add a standard sort order to a class by defining an :meth:`__lt__` method:

  **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 328); backlink

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 328); backlink

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 332)

Unknown directive type "doctest".

```
.. doctest::

  >>> Student.__lt__ = lambda self, other: self.age < other.age
  >>> sorted(student_objects)
  [('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
```

- Key functions need not depend directly on the objects being sorted. A key function can also access external resources. For instance, if the student grades are stored in a dictionary, they can be used to sort a separate list of student names:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\howto\[cpython-main][Doc][howto]sorting.rst, line 343)

Unknown directive type "doctest".

```
.. doctest::

  >>> students = ['dave', 'john', 'jane']
  >>> newgrades = {'john': 'F', 'jane':'A', 'dave': 'C'}
  >>> sorted(students, key=newgrades.__getitem__)
  ['jane', 'dave', 'john']
```