# 6 Useful Python Sorting Examples Using sorted Function

*by* AARON TABOR *on* JUNE 16, 2014



A common idiom in programming is sorting a list. Python makes this a very simple task.

Python provides a built-in sorted() function that accepts an iterable type, and return a sorted list:

## 1. Standard Sorting

First, try sorting a list:

```
>>> l = [3, 2, 5 ,4, 7, 1]
>>> sorted(l)
[1, 2, 3, 4, 5, 7]
```

Next, sort a tuple:

```
>>> t = ('Zane', 'Bob', 'Janet')
>>> sorted(t)
['Bob', 'Janet', 'Zane']
```

Finally, sort a dictionary:

```
>>> d = {1:'a', 2:'b', 3:'c'}
>>> sorted(d)
[1, 2, 3]
```

Notice how every time, the sorted() function returns a list, even if that is not the type

that was passed in. In the dictionary case, it returns a sorted list of the dictionaries keys.

## 2. Sorting Complex Structures using a Key

This is fine when we're working with things that have a natural sorting order like numbers or strings, but what happens when we need to sort more complex structures?

This is where the sorted() function shines. The sorted() function accepts a key as an optional named parameter.

This key should be, itself, a function which accepts one parameter and will be used by the sorted() function to determine a value to sort on.

Let's take a look at an example. Say we have a Person class with attributes name and age:

```
>>> class Person(object):
...     def __init__(self, name, age):
...             self.name = name
...             self.age = age
...     def __repr__(self):
...             return "<name: %s, age: %s>" % (self.name, self.age)
...
>>>
```

(The __repr__ function is a special function used to override how the object will be represented when displayed by the python interpreter.

The reason I've defined the function is to highlight the sort order.

By default, the representation of user defined objects looks something like this: "<__main__.Person object at 0xb7083ccc>". Leaving the respresentation like this would have made it harder for us to distinguish between different instances in the examples to come.)

Now let's make a list of people:

```
>>> jack = Person('Jack', 19)
>>> adam = Person('Adam', 43)
>>> becky = Person('Becky', 11)
>>> people = [jack, adam, becky]
```

By itself, sorted() doesn't know what to do with the list of people:

```
>>> sorted(people)
[<name: Jack, age: 19>, <name: Adam, age: 43>, <name: Becky, age: 11>]
```

However, we can tell the sorted() function which attribute to sort on by specifying a key to be used. Let's define one:

```
>>> def byName_key(person):
...     return person.name
...
```

A key function must accept a single argument and returns a value on which to base the sorting (sorted() will call the key function on each element of the iterable it is given, and use this return value when sorting the list.)

```
>>> sorted(people, key = byName_key)
[<name: Adam, age: 43>, <name: Becky, age: 11>, <name: Jack, age: 19>]
```

Notice we are handing in a reference to the function itself, not calling it and handing in a reference to its return value. This is an important distinction. Remember, sorted() will make use of the key function, calling it on each element of the iterable.

Let's write another one, this time specifying age as the value to sort on:

```
>>> def byAge_key(person):
...     return person.age
...
>>> sorted(people, key = byAge_key)
[<name: Becky, age: 11>, <name: Jack, age: 19>, <name: Adam, age: 43>]
```

## 3. Reverse Sorting

The sorted() function makes it easy to sort in reverse order too. The function accepts an optional, named parameter ‘reverse’ which should be a boolean.

```
>>> sorted(l, reverse = True)
[7, 5, 4, 3, 2, 1]
>>> sorted(t, reverse = True)
['Zane', 'Janet', 'Bob']
```

By default, ‘reverse’ is set to False.

## 4. Sort using attrgetter Function

This time, the list returned is sorted by age, just as we expected. In fact, sorting by a certain attribute of an object is such a common task in python that the standard library provides a function that can generate key functions for you:

```
>>> from operator import attrgetter
```

The return from a call to attrgetter() is a function similair to the 2 we just wrote. We specify the name of the attribute to be fetched, and attrgetter generates a function that accepts an object and return the specified attribute from that object.

```
>>> getName = attrgetter('name')
>>> getName(jack)
'jack'
```

So attrgetter('name') returns a function which behaves like our previously defined byName_key():

```
>>> sorted(people, key = attrgetter('name'))
[<name: Adam, age: 43>, <name: Becky, age: 11>, <name: Jack, age: 19>]
```

And attrgetter('age') returns a function which behaves like our previously defined byAge_key():

```
 >>> sorted(people, key = attrgetter('age'))
[<name: Becky, age: 11>, <name: Jack, age: 19>, <name: Adam, age: 43>]
```

## 5. Advance Usage of Key in Sorted Function

So far, our key functions have been simple attribute readers, but they can also compute a value to sort on. Let's look at another example. This time we'll define a Snake class:

```
>>> class Snake(object):
...     def __init__(self, name, toxicity, aggression):
...             self.name = name
...             self.toxicity = toxicity
...             self.aggression = aggression
...     def __repr__(self):
...             return "<%s>" % self.name
...
```

Our Snake has a name, a toxicity (which is a measure of how poisonous the snake's venom is), and aggression (which will be a number between 0 and 1, indicating the likelihood of the snake attacking).

```
>>> gardenSnake = Snake('gardenSnake', 10, 0.1)
>>> rattleSnake = Snake('rattleSnake', 100, 0.25)
>>> kingCobra = Snake('kingCobra', 50, 1.0)
```

```
>>> snakes = [rattleSnake, kingCobra, gardenSnake]
```

Now assuming that we can calculate how dangerous a snake is based on the product of its toxicity and aggression levels, we can sort the list of snakes by how dangerous they are.

```
>>> def byDangerous_key(snake):
...     return snake.toxicity * snake.aggression
...
>>> sorted(snakes, key = byDangerous_key)
[<gardenSnake>, <rattleSnake>, <kingCobra>]
```

The snakes come out in the order we expect (even though the rattleSnake was more venomous, the kingCobra aggression made it more dangerous)

## 6. Random Sorting

The keys don't even have to have any relation with elements being sorted (however, generally this is not a useful way to 'sort' things). We can create a random ordering with the following key:

```
>>> from random import random
>>> def randomOrder_key(element):
...     return random()
...
>>>
```

random() is a function from the standard library that returns a random number between 0 and 1. Sorting using this key returns, as you would expect, random ordering:

```
>>> sorted(snakes, key = randomOrder_key)
[<kingCobra>, <gardenSnake>, <rattleSnake>]
>>> sorted(snakes, key = randomOrder_key)
```

```
[<rattleSnake>, <kingCobra>, <gardenSnake>]
```

In this tutorial we've taken a look at how python makes sorting lists (and other iterables) easy. By default, the sorted() function will return a list sorted in its natural sort order (which is generally what we want for numbers and strings). For more information about the sorted() function, visit the python official docs here.

We've also learned how to specify our own sort order by passing in a key function to sorted(). Our key functions can return any value that we desire, but a lot of the time we want to sort on some attribute belonging to each element of the list. In fact, this is so often the case that Python provides a standard library function, operator.getattr(), to generate key functions of this type for us.



> Add your comment

## If you enjoyed this article, you might also like..


1. 50 Linux Sysadmin Tutorials
2. 50 Most Frequently Used Linux Commands (With Examples)
3. Top 25 Best Linux Performance Monitoring and Debugging Tools
4. Mommy, I found it! – 15 Practical Linux Find Command Examples
5. Linux 101 Hacks 2nd Edition eBook

- Awk Introduction – 7 Awk Print Examples
- Advanced Sed Substitution Examples
- 8 Essential Vim Editor Navigation Fundamentals
- 25 Most Frequently Used Linux IPTables Rules Examples
- Turbocharge PuTTY with 12 Powerful Add-Ons


Tagged as: Python attrgetter Example, Python Sort by Key, Python Sort Dictionary, Python Sort List, Python Sort Reverse, Python Sort Tuples, Python Sorted Key

## { 14 comments... add one }

**Jalal Hajigholamali** June 17, 2014, 12:53 am

Hi,

Very nice article.

Thanks...

LINK

---

**Bartłomiej Burek / furas** June 17, 2014, 4:40 am

I prefer `lambda` than `attrgetter`.

sorted(people, key=lambda x:x.age)

sorted(people, key=lambda x:x.name)

---