**Python: Sorting**

Sorting a list be like:

list\_name.sort()

list\_name.sort(reverse = True)

The same cannot be performed for a tuple because they are ***immutable*** objects as the elements inside cannot be changed.

While checking help(list) the IN PLACE term means that a second data structure is not created and modifications are made to the existing one.

Consider having a list of tuples with planet details, they are arranged in order of distance from the sun:

planets = [('Mercury', 2440, 5.43, 0.395), ('Venus', 6052, 5.24, 0.723), ('Earth', 6378, 5.52, 1.000), ('Mars', 3396, 3.93, 1.530), ('Jupiter', 71492, 1.33, 5.210)]

We need to order them based on the size (descending).

size = lambda planet:planet[1]

planets.sort(key=size, reverse=True)

returns:

[('Jupiter', 71492, 1.33, 5.21),

('Earth', 6378, 5.52, 1.0),

('Venus', 6052, 5.24, 0.723),

('Mars', 3396, 3.93, 1.53),

('Mercury', 2440, 5.43, 0.395)]

Now let’s sort them based on density:

density = lambda planet:planet[2]

planets.sort(key=density, reverse=True)

planets returns:

[('Earth', 6378, 5.52, 1.0),

('Mercury', 2440, 5.43, 0.395),

('Venus', 6052, 5.24, 0.723),

('Mars', 3396, 3.93, 1.53),

('Jupiter', 71492, 1.33, 5.21)]

list.sort() alters the list itself.

But if you want:

* A sorted copy of the list
* To sort a tuple

We use sorted()

sorted() can be used on iterables like tuples, lists and strings.

It creates a new copy of the sorted list while unfaltering the original collection