Rabbit test subjects

The diabolical Professor Boolean has captured you and a group of your hapless rabbit kin as test subjects for his terrible experiments! You're not sure what his real plans are, but currently it seems he's trying to make everyone faster and smarter? He's exposing rabbit test subjects to novel chemicals, genetic manipulations, and pathogens; then measuring their completion time for various puzzles and exercises. Then again, there's a rumor he's developing a kind of zombie rabbit. You don't want to become a zombit!

Unfortunately, due to insubordination and laziness, Professor Boolean just "eliminated" the lab assistant tracking all data from this research. Now, he's forcing you to sort through the notes and find something useful from the chaos. You have no choice but to abide by your captors evil rules. For now.

Of the subjects that have survived, each has a distinct file, with anywhere from 1 to 100 measurements of completion time for the tests. The measurements from the before and after cases are listed separately, but the ordering has been mixed up. You have to figure out the degree of improvement (0% to 99%, rounded to the nearest whole number) based on the two lists of results.

For example, if the first list of times is [22.2, 46, 100.8] and the second list is [23, 11.1, 50.4] you would return 50, because the times got 50% shorter: the 22.2 entry improved to 11.1, the 46 improved to 23, and the 100.8 improved to 50.4. Even though the data points are in different order, each improves by the same amount.

Write a function answer(x, y) which takes two lists of floating point performance scores and returns the improvement percentage, rounded to the nearest integer.

Test cases

Inputs: (double list) y = [1.0] (double list) x = [1.0] Output: (int) 0

Inputs: (double list) y = [2.2999999999999998, 15.0, 102.40000000000001, 3486.8000000000002] (double list) x = [23.0, 150.0, 1024.0, 34868.0] Output: (int) 90

Constraints

Python

Your code will run inside a Python 2.7.6 sandbox. (KN edit: v3 is OK too)

Standard libraries are supported except for bz2, crypt, fcntl, mmap, pwd, pyexpat, select, signal, termios, thread, time, unicodedata, zipimport, zlib.

Solved with a For Loop:

In [4]:
```
list1 = [22.2, 46, 100.8]
list2 = [23, 11.1, 50.4]

def sort_percent_for(list1, list2):
    assert len(list1) == len(list2)
    ans = []
    list1 = sorted(list1)
    list2 = sorted(list2)
    for x in range(len(list1)):
        diff = list1[x] - list2[x]
        pct = diff / list1[x]
        pct *= 100
        pct = int(pct)
        ans.append(pct)
    return ans
```

In [5]:
```
sort_percent_for(list1, list2)
```

Out[5]:
```
[50, 50, 50]
```

Solved with a List Comprehension

In [7]:
```
list1 = [22.2, 46, 100.8]
list2 = [23, 11.1, 50.4]

def sort_percent(list1, list2):
    #Sort both lists so they can be compared elementwise.  Two different
ways to sort:
    list1 = sorted(list1) #can be used on any iterable
    list2.sort()
    #Calculate the difference elementwise across the two lists and divid
e by the original value.  Then scale to percent.
    percents = [int((list1[x] - list2[x]) / list1[x] * 100) for x in ran
ge(len(list1))]
    return percents
```

In [8]:
```
sort_percent(list1, list2)
```

Out[8]:
```
[50, 50, 50]
```

Solved using NumPy

In [9]:
```
import numpy as np
```

In [10]:
```
def sort_percent_np(list1, list2):
    list1.sort()
    list2.sort()
    ans = (list1 - list2) / list1 * 100
    return ans.astype(int)
```

In [11]:
```
list1 = np.array([22.2, 46, 100.8])
list2 = np.array([23, 11.1, 50.4])

sort_percent_np(list1, list2)
```

Out[11]:
```
array([50, 50, 50])
```

Test Cases

In [12]:
```
sort_percent([1.0], [1.0])
```

Out[12]:
```
[0]
```

In [13]:
```
sort_percent([23.0, 150.0, 1024.0, 34868.0],[2.2999999999999998, 15.0, 1
02.40000000000001, 3486.8000000000002])
```

Out[13]:
```
[90, 90, 90, 90]
```