Python: Scalanie

Witajcie! Jest to kolejny z serii tutoriali uczący Pythona, a w przyszłości być może nawet Cythona i Numby. Jeśli chcesz nauczyć się nowych, zaawansowanych konstrukcji to spróbuj rozwiązać kilka następnych zadań. Powodzenia!

Zadanie

Sortowanie jest fundamentalnym problemem w informatyce. W tym zadaniu nauczysz się prostej metody sortowania. Co więcej, w kolejnym zadaniu będziesz testować wadliwe implementacje sortowania. Motywem przewodnim w kolejnych kilku zadaniach jest testowanie prostych ale błędnych implementacji algorytmów. Najpierw nauczysz się jak rozwiązać dany problem, a później będziesz testował rozwiązania innych.

Najpierw proste wyjaśnienie w jaki sposób odbywa się sortowanie:

1. Początkowe dzielenie

Na początku posiadamy listę bliżej nieokreślonych elementów. Elementy nie mają żadnego porządku. Kolejny etap będzie potrzebował uporządkowanych list. Zastosujemy więc prosty zabieg: stworzymy listę pojedynczych elementów. Dla przykładu:

```
divide([1,2,5,7])
```

```
[[1], [2], [5], [7]]
```

2. Scalanie

Dwie uporządkowane listy możemy scalić w jedną, także uporządkowaną. Zasada jest prosta: Wybieramy mniejszy element na początku obu list, wyjmujemy go i przenosimy na koniec wynikowej listy. Dla przykładu:

```
merge([1,2,5,7,8], [0,1,3,4,6])
```

```
[1, 2, 5, 7, 8] [0, 1, 3, 4, 6] -> []
[1, 2, 5, 7, 8] [1, 3, 4, 6] -> [0]
[1, 2, 5, 7, 8] [3, 4, 6] -> [0, 1]
[2, 5, 7, 8] [3, 4, 6] -> [0, 1, 1]
[5, 7, 8] [3, 4, 6] -> [0, 1, 1, 2]
[5, 7, 8] [4, 6] -> [0, 1, 1, 2, 3]
[5, 7, 8] [6] -> [0, 1, 1, 2, 3, 4]
[7, 8] [6] -> [0, 1, 1, 2, 3, 4, 5]
[7, 8] [] -> [0, 1, 1, 2, 3, 4, 5, 6]
[8] [] -> [0, 1, 1, 2, 3, 4, 5, 6, 7]
[] [] -> [0, 1, 1, 2, 3, 4, 5, 6, 7, 8]
```

3. Podziel i zwyciężaj

Powyższe dwie metody wykorzystamy do zaimplementowania sortowania. Na początku, dzielimy początkową listę na listę pojedynczych elementów. Następnie parami scalamy listy. Powstanie w ten sposób dwa razy mniej list o dwa razy więcej elementach. Ponownie scalamy parami listy. Ponownie, powstanie dwa razy mniej list. Na samym końcu

pozostanie tylko jedna lista. Dla przykładu:

```
sorting([3,5,0,1,6,3,5,4])
```

```
[3] [5] [0] [1] [6] [3] [5] [4]
[0] [1] [6] [3] [5] [4] [3, 5]
[6] [3] [5] [4] [3, 5] [0, 1]
[5] [4] [3, 5] [0, 1] [3, 6]
[3, 5] [0, 1] [3, 6] [4, 5]
[3, 6] [4, 5] [0, 1, 3, 5]
[0, 1, 3, 5] [3, 4, 5, 6]
[0, 1, 3, 3, 4, 5, 5, 6]
```

4. Testowanie

Twoim zadaniem jest zaimplementować funkcje o nazwach divide, merge i sorting. Pobierz szablon kodu, po uzupełnieniu odeślij jako zgłoszenie. Szablon zawiera doklejony kod, który sam pobierze dane wejściowe i wypisze dane wyjściowe. Musisz jedynie zaimplementować trzy metody.

```
def divide(inputlist):
    """Dzieli liste na wejsciu na liste list pojedynczych elementow.
    >>> divide([1,2,3])
    [[1],[2],[3]]
    """
    pass

def merge(onelist, seclist):
    """Scala dwie posortowane listy w jedna posortowana liste.
    >>> merge([1,2,4],[2,3,5])
    [1,2,2,3,4,5]
    """
    pass

def sorting(inputlist):
    """Zwraca kolejne stany algorytmu sortowania. Nalezy uzyc dwoch funkcji powyzej.
    Po kazdym scaleniu zwroc liste list (uzyj yield).
    >>> sorting([0,5,3,1,2])
    **yield dla kazdej linii**
    [[0], [5], [3], [1], [2]]
    [[3], [1], [2], [0, 5]]
    [[2], [0, 5], [1, 3]]
    [[1, 3], [0, 2, 5]]
    [[0, 1, 2, 3, 5]]
    """
    pass
```

Przykładowe dane oraz wyniki są dostępne do pobrania.