Python (BSU FAMCS Fall'19)

Семинар 3

Преподаватель: Дмитрий Косицин

Задание 1. (0.5 балла). Реализуйте функцию `transpose`, которая транспонирует *iterable* вложенных *iterable*. Предполагайте, что количество элементов во всех вложенных *iterable* одинаково. Другими словами, транспонирует прямоугольный двухмерный массив.

Воспользуйтесь функциями из модуля `itertools` и *built-in* функциями. Использовать циклы и явно приводить *iterable* к спискам или кортежам не разрешается.

Функцию сохраните в файле *iter_helpers.py*.

Пример

```
expected = [[1, 2], [-1, 3]]
actual = transpose([[1, -1], [2, 3]])
assert expected == list(map(list, actual))
```

Задание 2. (1 балл). Реализуйте функцию `scalar_product`, которая считает скалярное произведение двух *iterable*.

Элементы могут иметь тип `int` или `float`, а также быть строками. Строки могут быть:

- представлением целых чисел, в том числе в двоичной и шестнадцатиричной системе счисления,– используйте *built-in* функцию `int` для преобразования их к числу,
- состоять из букв – в таком случае результат вычисления всего выражения полагайте `None`.

Воспользуйтесь функциями из модуля **itertools** и *built-in* функциями. Использовать циклы не разрешается.

Функцию сохраните в файле *iter_helpers.py*.

Пример

```
expected = 1
actual = scalar_product([1, '2'], [-1, 1])
assert expected == actual

actual = scalar_product([1, 'xyz'], [-1, 1])
assert actual is None
```

Задание 3. (0.5 балла). Реализуйте декоратор `profile`, который при вызове функции подсчитывает время выполнения этой функции, выводит его на экран и возвращает результат вызова функции.

Рассмотрите стандартный модуль `timeit` и, в частности, функцию `default_timer` для измерения времени выполнения.

Декоратор сохраните в файле *utils.py*.

Пример

```
@profile
def some_function():
    return sum(range(1000))

result = some_function()  # return a value and print execution time
```

Задание 4. (0.5 балла). Реализуйте менеджер контекста `timer`, который посчитает время выполнения блока и выведет его на экран.

Сохраните менеджер контекста в файле *utils.py*.

Пример

```
with timer():
    print(sum(range(1000)))
    # print execution time when calculation is over
```

Задание 5. (1.5 балла). Пусть у вас есть класс, представляющий собой узел односвязного списка `Node` (см. ниже).

Тогда односвязный список будет представлять собой последовательность узлов `Node`, где `next_` либо является объектом типа `Node`, либо равно `None` (конец списка).

Списки могут быть вложенными – в этом случае значение `value` будет иметь тип `Node`.

Реализуйте следующее:

- Добавьте в класс проверки типов в конструкторе (используйте `assert`);
- Добавьте также свойства (property), позволяющие взять или изменить значения `_value` и `_next`;
- Добавьте «магический» метод `__iter__`, позволяющий проитерироваться по списку;
- Напишите функцию `flatten_linked_list`, которая разворачивает список *inplace*: модифицирует переданный список так, что в нем `_value` не могут быть `Node`, но при итерировании последовательность значений получается та же – и возвращает тот же объект, который был ей передан (см. пример *r3* ниже).

Код сохраните в файле *linked_list.py*.

Пример

```
class Node(object):
    def __init__(self, value, next_=None):
        self._value = value
        self._next = next_


r1 = Node(1)  # 1 -> None - just one node

r2 = Node(7, Node(2, Node(9)))  # 7 -> 2 -> 9 -> None

# 3 -> (19 -> 25 -> None) -> 12 -> None
r3 = Node(3, Node(Node(19, Node(25)), Node(12)))
r3_flattenned = flatten_linked_list(r3)  # 3 -> 19 -> 25 -> 12 -> None
r3_expected_flattenned_collection = [3, 19, 25, 12]
assert r3_expected_flattenned_collection == list(r3_flattenned)
```