Python (BSU FAMCS Fall'18)

Семинар 5

Преподаватели: Дмитрий Косицин, Светлана Боярович

Общие замечания ко всем заданиям

Свой тестирующий код можно размещать под условием `if __name__ == '__main__'` внизу файла. Такой код выполнится только если запустить этот файл, но не импортировать его.

Просьба использовать те имена для функций и файлов, которые указаны в замечаниях к заданиям.

Для отправки заданий выберите в anytask нужную задачу и там к сообщению прикрепите свое решение (один или несколько .py файлов).

Просьба также не оставлять debug print'ы в ваших программах, которые нужны исключительно для вывода отладочной информации.

Задание 1. (0.5 балла). Напишите функцию, которая при каждом вызове возвращает количество раз, которое она была вызвана. Использовать глобальные переменные не допускается, иначе говоря, в файле должно быть только определение функции (с ключевым словом `def`).
Замечание. Функцию назовите `smart_function`. Программу сохраните в файле *functional.py*.

Пример

```
for real_call_count in range(1, 5):
  assert f() == real_call_count
```

Задание 2. (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 не разрешается.

Функцию сохраните в файле *iterators.py*.

Пример

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

Задание 3. (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 либо представлением целых чисел (в том числе в двоичной или шестнадцатиричной системе счисления – используйте *built-in* функцию `int`), либо состоять из букв. Обработайте этот случай с помощью исключений, результатом вычисления в таком случае считайте `None`.

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

Функцию сохраните в файле *iterators.py*.

Пример

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

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

Задание 4. (1 балл). Реализуйте функцию-генератор `flatten`, которая разворачивает вложенные итерируемые объекты в один *iterable*.

Обратите внимание, что строки, хоть они и являются итерируемыми, распаковывать не нужно. Рекурсивно вызывать функцию не следует. Преобразовывать итерируемые объекты к спискам запрещено.

Функцию сохраните в файле *functional.py*.

Пример

```
expected = [1, 2, 0, 1, 1, 2, 1, 'ab']
actual = flatten([1, 2, range(2), [[], [1], [[2]]], (x for x in [1]), 'ab'])
assert expected == list(actual)
```

Задание 5. (1 балл). Реализуйте метакласс `BoundedMeta`, который контролирует количество созданных объектов классов, которые имеют данный метакласс. Допустимое количество объектов задайте параметром (по умолчанию 1).

В случае превышения бросайте исключение `TypeError`. Если значение параметра равно `None`, то ограничений нету.

Другими словами, у класса `C` с метаклассом `BoundedMeta` должно быть создано не более 2 экземпляров.

Класс сохраните в файле *semaphore.py*.

Заготовка метакласса BoundedMeta

```
class C(metaclass=BoundedMeta, max_instance_count=2):
    pass

c1 = C()
c2 = C()

try:
    c3 = C()
except TypeError:
    print('everything works fine!')
else:
    print('something goes wrong!')
```

Задание 6. (1 балл). Реализуйте класс `BoundedBase`, в котором определен абстрактный метод класса `get_max_instance_count`, возвращающий максимальное количество экзмепляров, которые можно создать.

Не допускайте создания объекта, если данное значение превышено – бросайте исключение `TypeError`. Значение, равное `None` – без ограничений.

Класс сохраните в файле *semaphore.py*.

Заготовка класса BoundedBase

```
class D(BoundedBase):
    @classmethod
    def get_max_instance_count(cls):
        return 1

d1 = D()

try:
    d2 = D()
except TypeError:
    print('everything works fine!')
else:
    print('something goes wrong!')
```