

Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Автоматизированные системы обработки информации и управления»



Отчет по лабораторной работе № 4

«Python. Функциональные возможности»

по курсу

“Разработка Интернет-приложений”

ИСПОЛНИТЕЛЬ:

Березин И.С.

Группа ИУ5-53

"__" _____ 2016 г.

ПРЕПОДАВАТЕЛЬ:

Гапанюк Ю.Е.

"__" _____ 2016 г.

Москва 2016

Задание

Важно выполнять все задачи последовательно. С 1 по 5 задачу формируется модуль librip, с помощью которого будет выполняться задание 6 на реальных данных из жизни. Весь вывод на экран (даже в столбик) необходимо реализовывать одной строкой.

Подготовительный этап

1. Зайти на [github.com](https://github.com/iu5team/ex-lab4) и выполнить fork проекта с заготовленной структурой <https://github.com/iu5team/ex-lab4>
2. Переименовать репозиторий в lab_4
3. Выполнить git clone проекта из вашего репозитория

Задача 1 (ex_1.py)

Необходимо реализовать генераторы field и gen_random. Генератор field последовательно выдает значения ключей словарей массива Пример:

```
goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'color': 'black'}
]
field(goods, 'title') должен выдавать 'Ковер', 'Диван для отдыха'
field(goods, 'title', 'price')
должен выдавать {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}
```

1. В качестве первого аргумента генератор принимает list, дальше через *args генератор принимает неограниченное кол-во аргументов.
2. Если передан один аргумент, генератор последовательно выдает только значения полей, если поле равно None, то элемент пропускается
3. Если передано несколько аргументов, то последовательно выдаются словари, если поле равно None, то оно пропускается, если все поля None, то пропускается целиком весь элемент

Генератор gen_random последовательно выдает заданное количество случайных чисел в заданном диапазоне Пример: gen_random(1, 3, 5) должен выдать 5 чисел от 1 до 3, т.е. примерно 2, 2, 3, 2, 1

В ex_1.py нужно вывести на экран то, что они выдают *одной строкой*. Генераторы должны располагаться в librip/gen.py

Задача 2 (ex_2.py)

Необходимо реализовать итератор, который принимает на вход массив или генератор и итерируется по элементам, пропуская дубликаты. Конструктор итератора также принимает на вход именной bool-параметр ignore_case, в зависимости от значения которого будут считаться одинаковыми строки в разном регистре. По умолчанию этот параметр равен False

Пример:

```
data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
```

Unique(data) будет последовательно возвращать только 1 и 2

```
data = gen_random(1, 3, 10)
```

unique(gen_random(1, 3, 10)) будет последовательно возвращать только 1, 2 и 3

```
data = ['a', 'A', 'b', 'B']
```

Unique(data) будет последовательно возвращать только a, A, b, B

```
data = ['a', 'A', 'b', 'B']
```

Unique(data, ignore_case=True) будет последовательно возвращать только a, b

В ex_2.py нужно вывести на экран то, что они выдают *одной строкой*. **Важно** продемонстрировать работу как с массивами, так и с генераторами (gen_random). Итератор должен располагаться в librip/iterators.py

Задача 3 (ex 3.py)

Дан массив с положительными и отрицательными числами. Необходимо одной строкой вывести на экран массив, отсортированный по модулю. Сортировку осуществлять с помощью функции `sorted`

Пример: `data = [4, -30, 100, -100, 123, 1, 0, -1, -4]`

Вывод: `[0, 1, -1, 4, -4, -30, 100, -100, 123]`

Задача 4 (ex 4.py)

Необходимо реализовать декоратор `print_result`, который выводит на экран результат выполнения функции. Файл `ex_4.py` **не нужно** изменять. Декоратор должен принимать на вход функцию, вызывать её, печатать в консоль имя функции, печатать результат и возвращать значение.

Если функция вернула список (`list`), то значения должны выводиться в столбик. Если функция вернула словарь (`dict`), то ключи и значения должны выводиться в столбик через знак равно. Пример:

```
@print_result
```

```
def test_1():
```

```
    return 1
```

```
@print_result
```

```
def test_2():
```

```
    return 'iu'
```

```
@print_result
```

```
def test_3():
```

```
    return {'a': 1, 'b': 2}
```

```
@print_result
```

```
def test_4():
```

```
    return [1, 2]
```

```
test_1()
```

```
test_2()
```

```
test_3()
```

```
test_4()
```

На консоль выведется:

```
test_1
```

```
1
```

```
test_2
```

```
iu
```

```
test_3
```

```
a = 1
```

```
b = 2
```

```
test_4
```

```
1
```

```
2
```

Декоратор должен располагаться в `librip/decorators.py`

Задача 5 (ex 5.py)

Необходимо написать контекстный менеджер, который считает время работы блока и выводит его на экран. Пример:

```
with timer():
```

```
    sleep(5.5)
```

После завершения блока должно выводиться в консоль примерно 5.5

Задача 6 (ex_6.py)

Мы написали все инструменты для работы с данными. Применим их на реальном примере, который мог возникнуть в жизни. В репозитории находится файл `data_light.json`. Он содержит облегченный список вакансий в России в формате json (ссылку на полную версию размером ~ 1 Гб. в формате xml можно найти в файле `README.md`).

Структура данных представляет собой массив словарей с множеством полей: название работы, место, уровень зарплаты и т.д. В `ex_6.py` дано 4 функции. В конце каждая функция вызывается, принимая на вход результат работы предыдущей. За счет декоратора `@print_result` печатается результат, а контекстный менеджер `timer` выводит время работы цепочки функций.

Задача реализовать все 4 функции по заданию, ничего не изменяя в файле-шаблоне.

Функции `f1-f3` должны быть реализованы в 1 строку, функция `f4` может состоять максимум из 3 строк. Что функции должны делать:

1. Функция `f1` должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна игнорировать регистр.

2. Функция `f2` должна фильтровать входной массив и возвращать только те элементы, которые начинаются со слова “программист”. Иными словами нужно получить все специальности, связанные с программированием

3. Функция `f3` должна модифицировать каждый элемент массива, добавив строку “с опытом Python” (все программисты должны быть знакомы с Python). Пример:

Программист C# с опытом Python

4. Функция `f4` должна сгенерировать для каждой специальности зарплату от 100 000 до 200 000 рублей и присоединить её к названию специальности. Пример:

Программист C# с опытом Python, зарплата 137287 руб.

?

ex_1.py

```
#!/usr/bin/env python3
```

```
from librip.gens import field, gen_random
```

```
goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Кресло', 'price': 1500, 'color': 'brown'},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
    {'title': 'Стелаж', 'price': 7000, 'color': 'white'},
    {'title': None, 'price': 100, 'color': 'blue'},
    {'title': 'Вешалка для одежды', 'price': 800, 'color': 'white'}
]
```

```
# Реализация задания 1
```

```
print(list(field(goods, 'title')))
print(list(field(goods, 'title', 'price')))
```

```
print(list(gen_random(1,3,5)))
```

ex_2.py

```
from librip.gens import gen_random
from librip.iterators import Unique
```

```
data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
data2 = gen_random(1, 3, 10)
```

```
# Реализация задания 2
```

```

print(list(Unique(data1)))
print(list(Unique(data2)))
print(list(Unique(['a', 'A', 'b', 'B'])))
print(list(Unique(['a', 'A', 'b', 'B'], ignore_case=True)))

```

ex 3.py

```
#!/usr/bin/env python3
```

```
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
```

```
# Реализация задания 3
```

```
print(sorted(data, key=lambda x: abs(x)))
```

ex 4.py

```
from librip.decorators import print_result
```

```
# Необходимо верно реализовать print_result  
# и задание будет выполнено
```

```
@print_result
```

```
def test_1():  
    return 1
```

```
@print_result
```

```
def test_2():  
    return 'iu'
```

```
@print_result
```

```
def test_3():  
    return {'a': 1, 'b': 2}
```

```
@print_result
```

```
def test_4():  
    return [1, 2]
```

```
test_1()
```

```
test_2()
```

```
test_3()
```

```
test_4()
```

ex 5.py

```
from time import sleep
```

```
from librip.ctxmgrs import timer
```

```
with timer():  
    sleep(5.5)
```

ex 6.py

```
#!/usr/bin/env python3
```

```
import json
```

```
import sys
```

```
from librip.ctxmgrs import timer
```

```
from librip.decorators import print_result
```

```
from librip.gens import field, gen_random
```

```
from librip.iterators import Unique as unique
```

```
path = "data_light.json"
```

```
# Здесь необходимо в переменную path получить
```

```

# путь до файла, который был передан при запуске

with open(path, encoding="utf8") as f:
    data = json.load(f)

# Далее необходимо реализовать все функции по заданию, заменив `raise
NotImplemented`
# Важно!
# Функции с 1 по 3 должны быть реализованы в одну строку
# В реализации функции 4 может быть до 3 строк
# При этом строки должны быть не длиннее 80 символов

@print_result
def f1(arg):
    return sorted(unique(field(arg, 'job-name'), ignore_case=True),
key=str.lower)

@print_result
def f2(arg):
    return list(filter(lambda x: str(x).lower().startswith('программист'),
arg))

@print_result
def f3(arg):
    return ["{} с опытом Python".format(i) for i in arg]

@print_result
def f4(arg):
    return ["{}", зарплата {} руб.".format(work, salary) for (work, salary) in
zip(arg, gen_random(100000, 200000, len(arg)))]

with timer():
    f4(f3(f2(f1(data))))

```

librip/gens.py

```

import random

# Генератор вычленения полей из массива словарей
# Пример:
# goods = [
#     {'title': 'Ковер', 'price': 2000, 'color': 'green'},
#     {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
# ]
# field(goods, 'title') должен выдавать 'Ковер', 'Диван для отдыха'
# field(goods, 'title', 'price') должен выдавать {'title': 'Ковер', 'price':
2000}, {'title': 'Диван для отдыха', 'price': 5300}

def field(items, *args):
    assert len(args) > 0
    for item in items:
        assert isinstance(item, dict)
        if len(args) == 1:
            val = item.get(args[0])
            if val is not None:
                yield val
        else:
            res_dict = {key: item.get(key) for key in (set(item.keys()) &

```

```

set(args)) if item.get(key) is not None}
    if len(res_dict) > 0:
        yield res_dict

# Необходимо реализовать генератор

# Генератор списка случайных чисел
# Пример:
# gen_random(1, 3, 5) должен выдать примерно 2, 2, 3, 2, 1
# Hint: реализация занимает 2 строки
def gen_random(begin, end, num_count):
    for i in range(num_count):
        yield random.randint(begin, end)
# Необходимо реализовать генератор

```

librip/iterators.py

```

# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, **kwargs):
        self.ignore_case = kwargs.get('ignore_case') is not None and
kwargs.get('ignore_case')
        self.items = iter(items)
        self.seen = set()

    def __next__(self):
        while True:
            val = self.items.__next__()
            val_compare = str(val).lower() if self.ignore_case else val

            if val_compare not in self.seen:
                self.seen.add(val_compare)
                return val

    def __iter__(self):
        return self

```

librip/decorators.py

```

# Здесь необходимо реализовать декоратор, print_result который принимает на
вход функцию,
# вызывает её, печатает в консоль имя функции, печатает результат и
возвращает значение
# Если функция вернула список (list), то значения должны выводиться в столбик
# Если функция вернула словарь (dict), то ключи и значения должны выводиться в
столбик через знак равно
# Пример из ex_4.py:
# @print_result
# def test_1():
#     return 1
#
# @print_result
# def test_2():
#     return 'iu'
#
# @print_result
# def test_3():
#     return {'a': 1, 'b': 2}
#
# @print_result
# def test_4():
#     return [1, 2]
#

```

```

# test_1()
# test_2()
# test_3()
# test_4()
#
# На консоль выведется:
# test 1
# 1
# test_2
# iu
# test 3
# a = 1
# b = 2
# test_4
# 1
# 2

import functools

def print_result(func):
    @functools.wraps(func)
    def result(*args, **kwargs):
        val = func(*args, **kwargs)

        print(func.__name__)
        if isinstance(val, list):
            print("\n".join(map(str, val)))
        elif isinstance(val, dict):
            print("\n".join(map(lambda x: "{} = {}".format(x[0], x[1]),
                                val.items())))
        else:
            print(val)

        return val

    return result

```

librip/ctxmngers.py

```

# Здесь необходимо реализовать
# контекстный менеджер timer
# Он не принимает аргументов, после выполнения блока он должен вывести время
# выполнения в секундах
# Пример использования
# with timer():
#     sleep(5.5)
#
# После завершения блока должно вывестись в консоль примерно 5.5

```

```

import time

class timer:
    @staticmethod
    def get_time():
        return time.time()

    def __enter__(self):
        self.time_start = __class__.get_time()

    def __exit__(self, exc_type, exc_val, exc_tb):
        print(__class__.get_time() - self.time_start)

```


Результаты выполнения:

ex 1.py

```
['Ковер', 'Кресло', 'Диван для отдыха', 'Стелаж', 'Вешалка для одежды']  
[{'price': 2000, 'title': 'Ковер'}, {'price': 1500, 'title': 'Кресло'},  
[1, 1, 2, 3, 3]
```

ex 2.py

```
[1, 2]  
[3, 1, 2]  
['a', 'A', 'b', 'B']  
['a', 'b']
```

ex 3.py

```
[0, 1, -1, 4, -4, -30, 100, -100, 123]
```

ex 4.py

```
test_1  
1  
test_2  
iu  
test_3  
a = 1  
b = 2  
test_4  
1  
2
```

ex 5.py

```
5.512930393218994
```

ex 6.py

```
f1  
1С программист  
2-ой механик  
3-ий механик  
4-ый механик  
4-ый электромеханик  
[химик-эксперт  
ASIC специалист  
JavaScript разработчик
```

```
f2  
Программист  
Программист / Senior Developer  
Программист 1С  
Программист C#  
Программист C++  
Программист C++/C#/Java  
Программист/ Junior Developer  
Программист/ технический специалист  
Программист-разработчик информационных систем  
f3  
Программист с опытом Python  
Программист / Senior Developer с опытом Python  
Программист 1С с опытом Python  
Программист C# с опытом Python  
Программист C++ с опытом Python  
Программист C++/C#/Java с опытом Python  
Программист/ Junior Developer с опытом Python  
Программист/ технический специалист с опытом Python  
Программист-разработчик информационных систем с опытом Python  
f4  
Программист с опытом Python, зарплата 190885 руб.  
Программист / Senior Developer с опытом Python, зарплата 125032 руб.  
Программист 1С с опытом Python, зарплата 187153 руб.  
Программист C# с опытом Python, зарплата 158246 руб.  
Программист C++ с опытом Python, зарплата 139092 руб.  
Программист C++/C#/Java с опытом Python, зарплата 116105 руб.  
Программист/ Junior Developer с опытом Python, зарплата 137791 руб.  
Программист/ технический специалист с опытом Python, зарплата 131259 руб.  
Программист-разработчик информационных систем с опытом Python, зарплата 115054 руб.  
0.10937690734863281
```