Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ**

Отчет по лабораторной работе №3

«Объектно-ориентированные возможности»
по курсу “Разработка интернет приложений”

Студент: Федорова Антонина

Группа: ИУ5-53

_____ 08.10.2018

Москва, 2018

Задание лабораторной работы

Подготовительный этап

1. Зайти на github.com и выполнить `fork` проекта с заготовленной структурой https://github.com/iu5team/ex-lab4
2. Переименовать репозиторий в `lab_3`
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

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
В `ex_1.py` нужно вывести на экран то, что они выдают, с помощью кода в *одну строку*
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`. Итератор **не должен модифицировать** возвращаемые значения.
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`

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 выводить в столбик через знак равно
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

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**. Используйте наработки из предыдущих заданий.
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. Для фильтрации используйте функцию `filter`.
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*. Для модификации используйте функцию `map`.

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.* Используйте `zip` для обработки пары специальность - зарплата.

Исходный код:

Librib:

gens.py:

```
import random


def field(items, *args):
    assert len(args) > 0
    if len(args) == 1:
        for x in items:
            for a in args:
                yield x[a]
    else:
        for x in items:
            for a in args:
                if x[a] is not None:
                    yield {a: x[a]}
    # реализовала генератор


def gen_random(begin, end, num_count):
    for _ in range(num_count):
        yield random.randint(begin, end)
    # Реализовала генератор
```

iterators.py:

```
# Итератор для удаления дубликатов
class Unique(object):
```

```
    def __init__(self, items, **kwargs):
        self.lst = items
        self.index = 0
        self.pre = 0
        if kwargs.__len__() == 0:
            self.ignore_case = False
        else:
            self.ignore_case = kwargs

    def __next__(self):
        if self.index == (len(self.lst)):
            raise StopIteration
        else:
            if self.index == 0:
                self.index = self.index + 1
                return self.lst[self.pre]
            else:
                if self.ignore_case is False:
                    if self.lst[self.pre] != self.lst[self.index]:
                        self.pre = self.index
                        self.index = self.index + 1
                        return self.lst[self.pre]
                    else:
                        self.pre = self.index
                        self.index = self.index + 1
                        return self.__next__()
                else:
                    if self.lst[self.pre].lower() !=
self.lst[self.index].lower():
                        self.pre = self.index
                        self.index = self.index + 1
                        return self.lst[self.pre]
                    else:
                        self.pre = self.index
                        self.index = self.index + 1
                        return self.__next__()

    def __iter__(self):
        return self
```

decorators.py:

```
def print_result(fn):
    def inside(*args):
        print(fn.__name__)
        if len(args) == 0:
            fun = fn()
        else:
            fun = fn(args[0])
        if type(fun) == list:
            for i in fun:
                print(i)
        elif type(fun) == dict:
            for i in fun:
                print(i, "=", fun[i])
        else:
            print(fun)
        return fun
    return inside
```

ctxmngrs.py:

```
import time


# 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
class timer:
    def __enter__(self):
        self.start = time.clock()

    def __exit__(self, exc_type, exc_val, exc_tb):
        print(time.clock() - self.start)
```

Не librib:

ex1.py:

```
#!/usr/bin/env python3
from librip.gens import field, gen_random

goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
    {'title': 'Стелаж', 'price': 7000, 'color': 'white'},
    {'title': 'Вешалка для одежды', 'price': 800, 'color': 'white'}
]

# Реализация задания 1
print(list(field(goods, 'title')))
print(list(field(goods, 'title', 'price')))
print(list(field(goods, 'title', 'price', 'color')))

print(list(gen_random(1, 10, 5)))
```

Ex2.py

```
#!/usr/bin/env python3
from librip.gens import gen_random
from librip.iterators import Unique

data1 = [1, 1, 3, 5, 2, 2, 2, 4, 2, 2]
data2 = list(gen_random(1, 3, 10))

# Реализация задания 2

my_iter = Unique(data1)
for i in my_iter:
    print(i)
print('\n')

my_iter2 = Unique(data2)
print(data2)
for i in my_iter2:
    print(i)
print('\n')

data3 = ['A', 'a', 'B', 'b']
my_iter3 = Unique(data3, ig=True)
for i in my_iter3:
    print(i)
print('\n')

my_iter4 = Unique(data3)
```

```
for i in my_iter4:
    print(i)
```

Ex3.py:

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

data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
# Реализация задания 3
data = sorted(data, key=lambda x: abs(x))
print(data)
```

Ex4.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()
```

ex5.py

```
from time import sleep
from librip.ctxmngrs import timer

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

ex6.py

```
#!/usr/bin/env python3
import json
from librip.ctxmngrs import timer
from librip.decorators import print_result
from librip.gens import field, gen_random
from librip.iterators import Unique as Unique

path = "C:/Users/User/Desktop/Любимый/РИП/lab3_RIP/data_light_cp1251.json"

# Здесь необходимо в переменную path получить
# путь до файла, который был передан при запуске

with open(path) 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 list(Unique(sorted(field(arg, "job-name")), ig=True))


@print_result
def f2(arg):
    return list(filter(lambda x: "программист" in x, arg))


@print_result
def f3(arg):
    return list(map(lambda x: x+" с опытом python", arg))


@print_result
def f4(arg):
    return list("{}, зарплата {} руб.".format(x, y)for x, y in zip(arg,
list(gen_random(100000, 200000, len(arg)))))


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

Скриншоты выполнения работы

Ex1.py

```
['Ковер', 'Диван для отдыха', 'Стелаж', 'Вешалка для одежды']
[{'title': 'Ковер'}, {'price': 2000}, {'title': 'Диван для отдыха'}, {'price': 5300}, {'title': 'Стелаж'}, {'price': 7000}, {'title': 'Вешалка для одежды'}, {'
[{'title': 'Ковер'}, {'price': 2000}, {'color': 'green'}, {'title': 'Диван для отдыха'}, {'price': 5300}, {'color': 'black'}, {'title': 'Стелаж'}, {'price': 70
[10, 5, 3, 3, 5]
```

Ex2.py

```
1
3
5
2
4
2


[2, 3, 2, 3, 1, 1, 2, 3, 1, 2]
2
3
2
3
1
2
3
1
2


A
B


A
a
B
b
```

Ex3.py:

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

Process finished with exit code 0
```

Ex4.py:

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

Process finished with exit code 0
```

Ex5.py:

```
5.500463858494652

Process finished with exit code 0
```

Ex6.py:

```
1С программист с опытом python, зарплата 155796 руб.
Web-программист с опытом python, зарплата 133450 руб.
Веб - программист (PHP, JS) / Web разработчик с опытом python, зарплата 180714 руб.
Веб-программист с опытом python, зарплата 153718 руб.
Ведущий инженер-программист с опытом python, зарплата 144581 руб.
Ведущий программист с опытом python, зарплата 186820 руб.
Инженер - программист АСУ ТП с опытом python, зарплата 189963 руб.
Инженер-программист с опытом python, зарплата 135817 руб.
Инженер-программист (Клинский филиал) с опытом python, зарплата 140141 руб.
Инженер-программист (Орехово-Зуевский филиал) с опытом python, зарплата 177332 руб.
Инженер-программист 1 категории с опытом python, зарплата 151230 руб.
Инженер-программист ККТ с опытом python, зарплата 166441 руб.
Инженер-программист ПЛИС с опытом python, зарплата 121418 руб.
Инженер-программист САПОУ (java) с опытом python, зарплата 178494 руб.
Инженер-электронщик (программист АСУ ТП) с опытом python, зарплата 179198 руб.
Помощник веб-программиста с опытом python, зарплата 140938 руб.
Системный программист (С, Linux) с опытом python, зарплата 143821 руб.
Старший программист с опытом python, зарплата 163648 руб.
веб-программист с опытом python, зарплата 160839 руб.
инженер - программист с опытом python, зарплата 100266 руб.
инженер-программист с опытом python, зарплата 196397 руб.
педагог программист с опытом python, зарплата 170343 руб.
программист с опытом python, зарплата 169455 руб.
программист 1С с опытом python, зарплата 169230 руб.
0.04635262232025908

Process finished with exit code 0
```