Московский государственный университет им. Н. Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»

Лабораторные работы по курсу:

«Разработка Интернет Приложений»

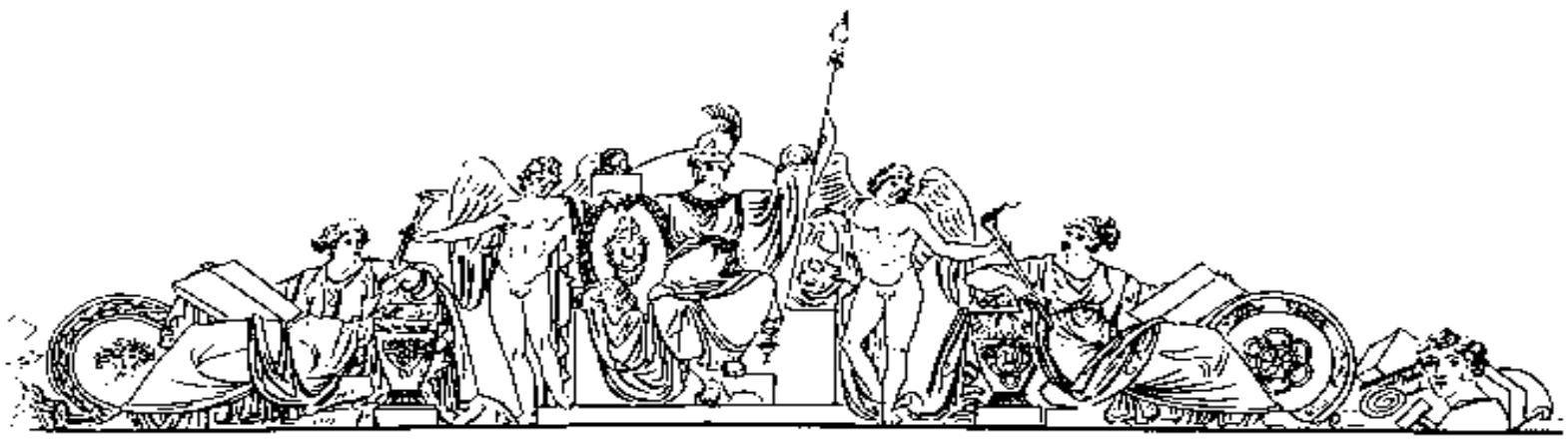
Лабораторная работа №4

Исполнитель:
Студентка группы ИУ5-54

Ворошилова М.А.

Преподаватель:

Гапанюк Ю.Е,



Москва 2017г.

Задание

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.

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

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 вывестись в консоль примерно 5.5

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
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
```

МГТУ им. Н. Э. Баумана, кафедра ИУ5, курс РИП
ЛР №4: Python, функциональные возможности

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

Декоратор должен располагаться в `librip/decorators.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 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.

Пример:

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

`Unique(data)` будет последовательно возвращать только `1` и `2`

МГТУ им. Н. Э. Баумана, кафедра ИУ5, курс РИП
ЛР №4: Python, функциональные возможности

```
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 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`

Листинг и вывод

ex_1.py:

```
from librip.gens import *

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'},
    {'title': 'Шкаф', 'price': 6500, 'color': None},
    {'title': None, 'price': None, 'color': None}
]

# Реализация задания 1

print(*((str(x) + '\n') for x in field(goods, 'title', 'color')))
print(*(str(x) + '  ' for x in gen_random(2, 5, 10)))
```

e

```
#!/usr/bin/env python3
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)
data3 = ["AAA", "aaa", "bb", "c", "c", "D"]
data4 = [1,1,2,3,4,4,4,3,5]

print(*Unique(data4, ignore_case=False))
# Реализация задания 2
print(*Unique(data1,False))
print(*(x for x in Unique(data1, False)))
print(*(x for x in Unique(list(data2), False)))
print(*(x for x in Unique(data3, False)))
```

e

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

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

e

```
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()
```

e

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

—

py:

```
with timer():
  sleep(3)
  print("Проснулся!")
  sleep(1)
  print("Ой, опять заснул!")
```

e

```
#!/usr/bin/env python3
import json
import sys
from librip.ctxmngrs import timer
from librip.decorators import print_result
```

```python
from librip.gens import field, gen_random
from librip.iterators import Unique as unique

path = "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):
    # Отсортированный без повторений Список названий профессий
    return sorted((x for x in unique(list(field(data, "job-name")), ignore_case=True)))


@print_result
def f2(arg):
    # Все профессии начинающиеся с "Программист"
    return list(filter(lambda x: x.startswith("Программист"), arg))


@print_result
def f3(arg):
    # Добавить строку " с опытом Python", map - обработка каждого элеммента массива
    return list(map(lambda x: x + " с опытом Python", arg))


@print_result
def f4(arg):
    salaries = list(gen_random(100000, 200000, len(arg)))
    for name, salary in zip(arg, salaries):
        print(name, ", зарплата ", salary, "руб. ")


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

ctxmngrs.py:

```python
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)
#


class timer:
    time = 0

    def __enter__(self):
        self.time = time.time()

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

decorators.py:

```python
# После завершения блока должно вывестись в консоль примерно 5.5
# 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 выводить в столбик через знак равно


def print_result(func):
    result = None

    def wrapper(*args, **kwargs):
        print(func.__name__)
        result = func(*args, **kwargs)
        if isinstance(result, list):
            for x in result:
                print(x)
        elif isinstance(result, dict):
            for key in result:
                print(key, " = ", result[key])
        else:
            print(result)
        return result
    return wrapper

# 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
```

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
    if len(args) == 1:
        for dic in items:
            if dic[args[0]] != None:
                yield dic[args[0]]
    else:
        temp = {}
        for dic in items:
            for arg in args:
                if dic[arg] != None:
                    temp[arg] = dic[arg]
            if len(temp) > 0:
                yield temp
                temp = {}


# 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 x in range(0, num_count):
        yield random.randint(begin, end)
    # Необходимо реализовать генератор
```

iterators.py:

```
# Итератор для удаления дубликатов
class Unique(object):
    _lst = []
    _new_lst = []
    _index = -1

    def __init__(self, items, ignore_case):
        assert len(items) > 0   # выход
        self._lst = list(items)
        if ignore_case & isinstance(self._lst[0], str):
            self._lst.sort(key=lambda st: st.lower())
        else:
            self. lst.sort()
        self._new_lst = []
        cur_in = 0
        self._new_lst.append(self._lst[0])
        for x in self._lst:
            if(ignore_case & isinstance(self._lst[0], str)):
                if(x.lower() != self._new_lst[cur_in].lower()):
                    self._new_lst.append(x)
                    cur_in += 1
            else:
                if(x != self._new_lst[cur_in]):
                    self. new lst.append(x)
                    cur in += 1
        self._index = -1

    def __next__(self):
        if self._index >= len(self._new_lst)-1:
            raise StopIteration
        self._index += 1
```

```
        return self._new_lst[self._index];

    def __iter__(self):
        return self
```

Вывод:

```
{'title': 'Ковер', 'color': 'green'}
 {'title': 'Диван для отдыха', 'color': 'black'}
 {'title': 'Стелаж', 'color': 'white'}
 {'title': 'Вешалка для одежды', 'color': 'white'}
 {'title': 'Шкаф'}

3   3   5   4   4   2   5   4   3   3

Process finished with exit code 0

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

Process finished with exit code 0
test_1
1
test_2
iu
test_3
a  =  1
b  =  2
test_4
1
2

Process finished with exit code 0
Проснулся!
Ой, опять заснул!
4.000855207443237

Process finished with exit code 0
```

```
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
Программистр-разработчик  информационных систем с опытом Python
f4
Программист с опытом Python , зарплата  109134 руб.
Программист / Senior Developer с опытом Python , зарплата  153328 руб.
Программист 1С с опытом Python , зарплата  102710 руб.
Программист C# с опытом Python , зарплата  105117 руб.
Программист C++ с опытом Python , зарплата  115390 руб.
Программист C++/C#/Java с опытом Python , зарплата  178228 руб.
Программист/ Junior Developer с опытом Python , зарплата  144509 руб.
Программист/ технический специалист с опытом Python , зарплата  164349 руб.
Программистр-разработчик  информационных систем с опытом Python , зарплата  156304 руб.
None
0.029074430465698242
```