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»

ЛР4. Python. Функциональные возможности

Исполнитель:

Студент группы РТ5-51

Разуваев К. А.

Преподаватель:

Гапанюк Ю. Е.

«___»_____

Москва 2017 г.

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 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`. 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`

`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 *о дной 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 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
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 вывестись 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**. 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). П ример: *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.

Исходный код: **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
   # Необходимо реализовать генератор
if len(args) == 1:        for dic in items:
if dic[args[0]] != None:
            yield dic[args[0]]
else:
      temp_dic = {}        for
dic in items:            for arg
in args:               if dic[arg]
!= None:
                temp_dic[arg] = dic[arg]
if len(temp_dic) > 0:
            yield temp_dic
            temp_dic = {}


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

iterators.py:

```
# Итератор для удаления дубликатов class
Unique(object):    def __init__(self, items,
ignore_case=False):
      # Нужно реализовать конструктор
      # В качестве ключевого аргумента, конструктор должен принимать bool-параметр ignore_case,
      # в зависимости от значения которого будут считаться одинаковые строки в разном регистре
      # Например: ignore_case = True, Абв и АБВ разные строки
      #                ignore_case = False, Абв и АБВ одинаковые строки, одна из них удалится
      # По-умолчанию ignore_case = False
      # self.len_items = len(items)
# assert self.len_items > 0
self.items = items
```

```
self.new_items = []         self.index
= -1
        self.ignore_case = ignore_case

        if ignore_case:             for item in
self.items:                 if item not in
self.new_items:
self.new_items.append(item)         else:
            for item in self.items:
                if isinstance(item, str):                     if item.lower() not in
[x.lower() for x in self.new_items]:
self.new_items.append(item)                 else:                     if item not
in self.new_items:                          self.new_items.append(item)

    def __next__(self):
        # Нужно реализовать __next__
if self.index == len(self.new_items)-1:
            raise StopIteration
self.index += 1
        return self.new_items[self.index]

    def __iter__(self):
        return self
```

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 выводить 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


def print_result(func_to_decorate):     def
decorated_func(*args, **kwargs):         result =
func_to_decorate(*args, **kwargs)
print(func_to_decorate.__name__)         if
isinstance(result, list):
            for x in result:
                if isinstance(x, tuple):
                    if len(x) == 2:
                        print("{}, зарплата {}".format(x[0], x[1]))
else:
                    print(x)
            # print(*(x for x in result))
elif isinstance(result, dict):
for key in result.keys():
                print(key, "=", result[key])
            # print(*(("\n{} = {}".format(key, result[key])) for key in result.keys()))
else:
            print(result)
        return result
return decorated_func
```

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

ex_1.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(*('\" + x + '\" for x in field(goods, 'title')))
print(*(x for x in field(goods, 'title', 'price'))) print(*(x
for x in gen_random(1, 3, 5)))
```

```
C:\Users\Deny\AppData\Local\Programs\Python\Python36\python.exe C:/Users/Deny/PycharmProjects/lab4/ex_1.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', 'price': 800}
1 2 1 2 3

Process finished with exit code 0
```

ex_2.py:

```
#!/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 = ['a', 'A', 'b', 'B']
data4 = []
data5 = ["Andrey", "andrey", "alExey", "Mikhail", "mikhail", "alexey"]

# Реализация задания 2

print(*(x for x in Unique(data1))) print(*(x
for x in Unique(data2))) print(*(x for x in
Unique(data3))) print(*(x for x in
Unique(data3, True))) print(*(x for x in
Unique(data4))) print(*(x for x in
Unique(data5))) print(*(x for x in
Unique(data5, True)))
```

```
C:\Users\Deny\AppData\Local\Programs\Python\Python36\python.exe C:/Users/Deny/PycharmProjects/lab4/ex_2.py
1 2
1 3 2
a b
a A b B

Andrey alExey Mikhail
Andrey andrey alExey Mikhail mikhail alexey

Process finished with exit code 0
```

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 number: abs(number)))
```

```
C:\Users\Deny\AppData\Local\Programs\Python\Python36\python.exe C:/Users/Deny/PycharmProjects/lab4/ex_3.py
[0, 1, -1, 4, -4, -30, 100, -100, 123]

Process finished with exit code 0
```

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_resul
t def test_2():
return        'iu'
@print_resul
t def test_3():
   return {'a': 1, 'b': 2}


@print_result def
test_4():     return
[1, 2]


test_1() test_2()
test_3() test_4()
```

```
C:\Users\Deny\AppData\Local\Programs\Python\Python36\python.exe C:/Users/Deny/PycharmProjects/lab4/ex_4.py
test_1
1
test_2
iu
test_3
a = 1
b = 2
test_4
1
2

Process finished with exit code 0
```

ex_5.py:

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

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

```
C:\Users\Deny\AppData\Local\Programs\Python\Python36\python.exe C:/Users/Deny/PycharmProjects/lab4/ex_5.py
5.4998942957223935

Process finished with exit code 0
```

ex_6.py:

```
#!/usr/bin/env python3
import json import sys
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
= "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 дожны 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(job for job in unique(field(arg, "job-name"), True))
```

```
@print_result def f2(arg):     return list(filter(lambda job:
job.startswith("рограммист", 1), arg))


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


@print_result def
f4(arg):
    return list(zip(arg, gen_random(100000, 200000, len(arg))))


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

```
C:\Users\Deny\AppData\Local\Programs\Python\Python36\python.exe C:/Users/Deny/PycharmProjects/lab4/ex_6.py
f1
1С программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
ASIC специалист
JavaScript разработчик
RTL специалист
Web-программист
Web-разработчик
[химик-эксперт
web-разработчик
Автожестянщик
Автоинструктор
Автомаляр
```

...

```
юрист
f2
Программист
Программист / Senior Developer
Программист 1С
Программист C#
Программист C++
Программист C++/C#/Java
Программист/ Junior Developer
Программист/ технический специалист
Программистр-разработчик  информационных систем
программист
программист 1С
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
программист с опытом Python
программист 1С с опытом Python
f4
Программист с опытом Python, зарплата 184252
Программист / Senior Developer с опытом Python, зарплата 190861
Программист 1С с опытом Python, зарплата 170863
Программист C# с опытом Python, зарплата 148392
Программист C++ с опытом Python, зарплата 196715
Программист C++/C#/Java с опытом Python, зарплата 149492
Программист/ Junior Developer с опытом Python, зарплата 187121
Программист/ технический специалист с опытом Python, зарплата 132678
Программистр-разработчик  информационных систем с опытом Python, зарплата 137371
программист с опытом Python, зарплата 138497
программист 1С с опытом Python, зарплата 199902
0.08602884587931361

Process finished with exit code 0
```