

Отчет по лабораторной работе №4
по курсу «Разработка интернет-приложений»
«Python. Функциональные возможности»

Выполнил:

Калиниченко Ирина, ИУ5-52

Преподаватель:

Гапанюк Ю.Е.

2016 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](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. Итератор не должен модифицировать в озращаемые 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 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 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). Пример: Программист С# 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 С# с опытом Python, зарплата 137287 руб. Используйте zip для обработки пары специальность — зарплата

Код программы

Файл gens.py:

```
import random

def field(items, *args):
    assert len(args) > 0
    if len(args) == 1:
        for el in items:
            yield el.get(args[0])
    else:
        for el in items:
            a = {}
            for key in args:
                if not el.get(key) is None:
                    a[key] = el[key]
            else:
                break
            if len(a) != 0:
                yield a

def gen_random(begin, end, num_count):
    pass
    for i in range(num_count):
        yield random.randint(begin, end)
```

Файл decorators.py:

```

def print_result(func):
    def decorated_func(*args):
        print(func.__name__)
        if len(args)>0:
            a=func(args[0])
        else:
            a=func()
        if type(a) is list:
            print("\n".join(map(str,a)))
        elif type(a) is dict:
            print('\n'.join([str(x)+"="+str(a[x]) for x in a]))
        else:
            print(a)
        return a
    return decorated_func

```

Файл iterators.py:

```

# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, **kwargs):
        self.ignore_case = 0
        self.items = iter(items) if isinstance(items, list) else items
        for k, v in kwargs.items():
            if k == 'ignore_case':
                if v:
                    self.ignore_case = 1
                else:
                    self.ignore_case = 0
        self.lst = []
        pass

    def __next__(self):
        while True:
            el = next(self.items)
            if self.ignore_case:
                if self.lst.count(el.lower()) == 0:
                    self.lst.append(el.lower())
                    return el
                else:
                    continue
            else:
                if self.lst.count(el) == 0:
                    self.lst.append(el)
                    return el
                else:
                    continue
        raise StopIteration

    def __iter__(self):
        return self

```

Файл ctxmgrs.py:

```

import time
class timer:
    def __enter__(self):
        self.t = time.clock()
    def __exit__(self, exp_type, exp_value, traceback):
        print(time.clock()-self.t)

```

Файл ex_1.py:

```
#!/usr/bin/env python3
from librip.gens import field
from librip.gens import gen_random
goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
    {'title': None, 'price': 7000, 'color': 'white'},
    {'title': None, 'price': None, 'color': 'white'}
]

# Реализация задания 1
g = field(goods, 'title', 'price')
for i in g:
    print(i, end=" ")

print()

num = gen_random(1, 5, 3)
for i in num:
    print(i, end=" ")
```

Файл 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']

# Реализация задания 2
first = Unique(data1)
for i in first:
    if i is None:
        continue
    else:
        print(i, end=' ')
print()

second = Unique(list(data2))
for i in second:
    if i is None:
        continue
    else:
        print(i, end=' ')
print()

third = Unique(data3)
for i in third:
    if i is None:
        continue
    else:
        print(i, end=' ')
print()

fourth = Unique(data3, ignore_case=1)
for i in fourth:
```

```

        if i is None:
            continue
        else:
            print(i, end=' ')
    print()

```

Файл 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)), end=" ")

```

Файл ex_4.py:

```

from librip.decorators import print_result

@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()

```

Файл ex_5.py:

```

from time import sleep
from librip.ctxmgrs import timer

with timer():
    sleep(5.5)

```

Файл 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
print(sys.getdefaultencoding())

path = sys.argv[1]

with open(path, encoding="cp1251") as f:

```



```

data = json.load(f)

@print_result
def f1(arg):
    return sorted(unique([i for i in field(arg, 'job-name')],
ignore_case=True), key=lambda x: x.lower())

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

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

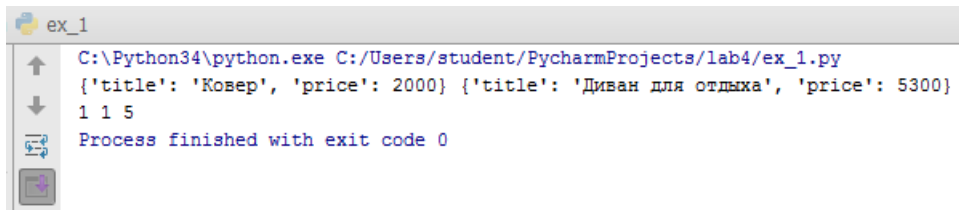
@print_result
def f4(arg):
    sal = list(gen_random(100000, 200000, len(arg)))
    return list('{} , зарплата {} руб.'.format(r, sal) for r, sal in zip(arg,
sal))

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

```

Результат работы программы

ex_1.py:

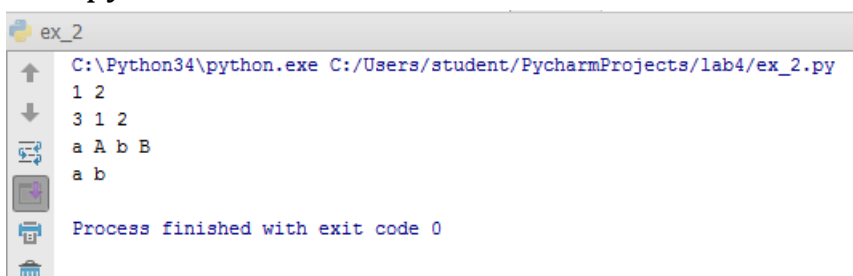


```

ex_1
C:\Python34\python.exe C:/Users/student/PycharmProjects/lab4/ex_1.py
{'title': 'Ковёр', 'price': 2000} {'title': 'Диван для отдыха', 'price': 5300}
1 1 5
Process finished with exit code 0

```

ex_2.py:

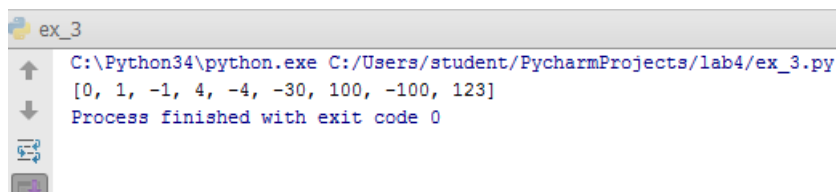


```

ex_2
C:\Python34\python.exe C:/Users/student/PycharmProjects/lab4/ex_2.py
1 2
3 1 2
a A b B
a b
Process finished with exit code 0

```

ex_3.py:



```

ex_3
C:\Python34\python.exe C:/Users/student/PycharmProjects/lab4/ex_3.py
[0, 1, -1, 4, -4, -30, 100, -100, 123]
Process finished with exit code 0

```

ex_4.py:

```
ex_4
C:\Python34\python.exe C:/Users/student/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:

```
ex_5
C:\Python34\python.exe C:/Users/student/PycharmProjects/lab4/ex_5.py
5.501127336669006

Process finished with exit code 0
```

ex_6.py:

```
ex_6
C:\Python34\python.exe C:/Users/student/PycharmProjects/lab4/ex_6.py data_light_cp1251.json
utf-8
f1
1С программист
2-ой механик
3-ий механик
4-ый механик
4-ый электромеханик
[химик-эксперт
ASIC специалист
JavaScript разработчик
RTL специалист
Web-программист
web-разработчик
Автожестящик
Автоинструктор
Автомаляр
Автомойщик
Автор студенческих работ по различным дисциплинам
автослесарь
Автослесарь - моторист
Автоэлектрик
Агент
Агент банка
Агент нпф
Агент по гос. закупкам недвижимости
Агент по недвижимости
Агент по недвижимости (стажер)
Агент по недвижимости / Риэлтор
Агент по привлечению юридических лиц
...
```

ex_6	
↑	Электрослесарь по ремонту оборудования в карьере
↓	Электроэрозионист
↕	Эндокринолог
↕	Энергетик
↕	Энергетик литейного производства
↕	энтомолог
↕	Юрисконсульт
↕	юрисконсульт 2 категории
↕	Юрисконсульт. Контрактный управляющий
↕	Юрист
↕	Юрист (специалист по сопровождению международных договоров, английский - разговорный)
↕	Юрист волонтер
↕	Юристконсульт
↕	f2
↕	Программист
↕	Программист / Senior Developer
↕	Программист 1C
↕	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 1C 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 159982 руб.
↕	Программист / Senior Developer с опытом Python, зарплата 142376 руб.
↕	Программист 1C с опытом Python, зарплата 167856 руб.
↕	Программист C# с опытом Python, зарплата 170371 руб.
↕	Программист C++ с опытом Python, зарплата 124523 руб.
↕	Программист C++/C#/Java с опытом Python, зарплата 135968 руб.
↕	Программист/ Junior Developer с опытом Python, зарплата 112285 руб.
↕	Программист/ технический специалист с опытом Python, зарплата 184196 руб.
↕	Программист-разработчик информационных систем с опытом Python, зарплата 193621 руб.
↕	0.1071409057634691