



“Разработка интернет-приложений”
«Python. Функциональные возможности»

Лабораторная работа № 4

Студент группы ИУ5 -53 _____ Бабин В.Е.
Преподаватель _____ Гапанюк Е.Ю.

Москва 2017

Задание

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 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 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`

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 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.

Код программы: ctmng.rs.py

```
import contextlib import
time
```

```
@contextlib.contextmanager
def timer():      t =
time.clock()      yield
print(time.clock() - t)
```

decorators.py

```
def print_result (some_func):      def
treatment(*args, **kwargs):
print(some_func.__name__)          if
type(some_func(*args,**kwargs)) == list:
for i in some_func(*args, **kwargs):
print(i)
return some_func(*args, **kwargs)
elif type(some_func(*args, **kwargs)) == dict:
for i in some_func(*args, **kwargs).keys():
print('{} = {}'.format(i, some_func(*args, **kwargs)[i]))
else:
print(some_func(*args, **kwargs))
```

return treatment gens.py

```
import random
def field(items, *args):      assert
len(args) > 0      if len(args) == 1:
for stroka in items:          if
args[0] in stroka:
yield stroka[args[0]]      elif
len(args) > 1:      for stroka in
items:
dict = {}
for arg in args:
if arg in stroka:
dict[arg] = stroka[arg]
if dict:
yield dict
def gen_random(begin, end,
num_count):
pass      for i in range(num_count):
yield random.randint(begin, end)
```

iterators.py

```
class Unique(object):      def
__init__(self, items, **kwargs):
self.items = []
if type(items) == list:
self.items = items
else:
self.items = [x for x in items]
self.index = 0
self.ignore_case = kwargs
self.length = len(self.items)      pass
```

```

        def __next__(self):
            if self.index == self.length - 1:
                raise StopIteration
            arr = []
            if self.ignore_case:
                arr = [i.lower() for i in self.items]
                if self.ignore_case and arr.count(self.items[self.index].lower())
!=
1:
            del self.items[self.index]
            self.length -= 1
            elif self.items.count(self.items[self.index]) != 1:
                del self.items[self.index]
            self.length -= 1
            else:
                self.index += 1
            return self.items[self.index]
        pass

    def __iter__(self):
        return self
ex_1.py
from librip.gens import field
goods =
[
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': None, 'color': 'black'},
    {'title': 'Стелаж', 'price': 7000, 'color': 'white'},
    {'color': 'red'},
    {'title': 'Нечто'},
    {'title': 'Вешалка для одежды', 'price': 800, 'color': 'white'}
]

```

```

# Реализация задания 1
def print_list(list):
    for x in range(len(list)):
        print(list[x], end='')
        if x != len(list)-1:
            print(end=', ')
    new_items = []
    for x in field(goods, 'color1', 'price'):
        new_items.append(x)
    print_list(new_items)

```

Результат работы:

```

C:\Users\Вкусик\AppData\Local\Programs\Python\Python37\python.exe (
{'price': 2000}, {'price': None}, {'price': 7000}, {'price': 800}
Process finished with exit code 0
|

```

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) data3
= ['a', 'b', 'A', 'D', 'A', 'b'] data4
= ['a', 'b', 'c', 'd', 'A', 'B', 'C',
'D', 'a', 'a', 'a', 'a', 'b', 'B',
'B']

# Реализация задания 2
uni = Unique(data4, ignore_case = True)
for i in uni:
    pass
print(uni.items)

```

Результат работы:

```
C:\Users\Вкусик\AppData\Local\Programs\Python\Python37\python.exe  
['C', 'D', 'a', 'B']
```

```
Process finished with exit code 0
```

ex_3.py

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

Результат работы:

```
C:\Users\Вкусик\AppData\Local\Programs\Python\Python37\python.exe (  
[0, 1, -1, 4, -4, -30, 100, -100, 123]
```

```
Process finished with exit code 0
```

```
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()
```

Результат работы:

```
C:\Users\Вкусик\AppData\Local\Programs\Python\Python37\p
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.ctxmgrs import timer
with timer():
    sleep(5.5)
```

Результат работы:

```
C:\Users\Вкусик\AppData\Local\Programs\Python\Python37\p
5.5303142105675125
```

Process finished with exit code 0

ex_6.py

```
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_cp1251.json'

with open(path) as f:
    data = json.load(f)

@print_result def
f1(arg):
    uni = unique([i for i in field(arg, 'job-name')], ignore_case=True)
    for i in uni:
        pass
        return uni.items

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

@print_result def
f3(arg):
```

```
return
list(map(lambda
x: x + ' с опытом
Python', arg))

@print_result def
f4(arg):
    salary = [x for x in gen_random(100000, 200000, len(arg))]    arg =
[x + ', зарплата {} руб'.format(y) for x, y in zip(arg, salary)]
return arg
    with
timer():
    f4(f3(f2(f1(data))))
```

Результат работы:

f1

Администратор на телефоне
Охранник сутки-день-ночь-вахта
теплотехник
Электро-газосварщик
Водитель Gett/Гетт и Yandex/Яндекс такси на личном автомобиле
Монолитные работы
Организатор - тренер
Врач ультразвуковой диагностики в детскую поликлинику
Менеджер по продажам ИТ услуг (B2B)
Аналитик
Воспитатель группы продленного дня

Итого по качеству

f2

Программист C++/C#/Java
Программист-разработчик информационных систем
Программист C++
Программист/ Junior Developer
Программист / Senior Developer
Программист/ технический специалист
программист
программист 1C
Программист C#

f3

Программист C++/C#/Java с опытом Python
Программист-разработчик информационных систем с опытом Python
Программист C++ с опытом Python
Программист/ Junior Developer с опытом Python
Программист / Senior Developer с опытом Python
Программист/ технический специалист с опытом Python
программист с опытом Python
программист 1C с опытом Python
Программист C# с опытом Python

f4

Программист C++/C#/Java с опытом Python, зарплата 151169 руб
Программист-разработчик информационных систем с опытом Python, за
Программист C++ с опытом Python, зарплата 180726 руб
Программист/ Junior Developer с опытом Python, зарплата 116291 руб
Программист / Senior Developer с опытом Python, зарплата 163405 руб
Программист/ технический специалист с опытом Python, зарплата 19918
программист с опытом Python, зарплата 179868 руб
программист 1C с опытом Python, зарплата 139098 руб
Программист C# с опытом Python, зарплата 186091 руб
60.06159429926719