



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

Лабораторная работа № 4

Студент группы ИУ5 -53_____Костенкова Ю.В.

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 повторов (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.

Исходный код:

```
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
    otv = []
    if len(args) == 1:
        for x in items:
            y = x.get(args[0])
            if y != None:
                otv.append(y)
    else:
        for x in items:
            z = []
            for y in args:
                q = []
                q.append(y)
                q.append(x.get(y))
                z.append(q)
            count = 0
            for i in z:
                if i[1] == None:
                    count += 1
            if (count != len(z)):
                for y in z:
                    if y[1] == None:
                        z.remove(y)
                otv1 = {}
                otv1.update(z)
                otv.append(otv1)
    return otv

# Необходимо реализовать генератор

# 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):

    return [random.choice([i for i in range(begin, end+1)]) for j in
```

```

range(num_count)]
    # Необходимо реализовать генератор

# 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 выводиться 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(function_to_decorate):
    # Внутри себя декоратор определяет функцию-"обёртку". Она будет обёрнута
    # вокруг декорируемой,
    # получая возможность исполнять произвольный код до и после неё.
    def the_wrapper_around_the_original_function(*a):
        otv = function_to_decorate(*a) # Сама функция
        print(function_to_decorate.__name__)
        if type(otv) == list:
            for i in otv:
                print(i)
        elif type(otv) == dict:
            for i, j in otv.items():
                print(i, '=', j)
        else:
            print(otv)
        return otv
    # Вернём эту функцию
    return the_wrapper_around_the_original_function

```

```

# 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.t = time.clock()

    def __exit__(self, exp_type, exp_value, traceback):
        print(time.clock() - self.t)
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()

```

Дополнительное задание:

```
lst = [1, 3, 5]
```

```
guard1 = [x * x for x in lst]
```

```
guard2 = list(map(lambda x: x * x, lst))
```

```
print(guard1);
```

```
print(guard2);
```

Результаты работы:

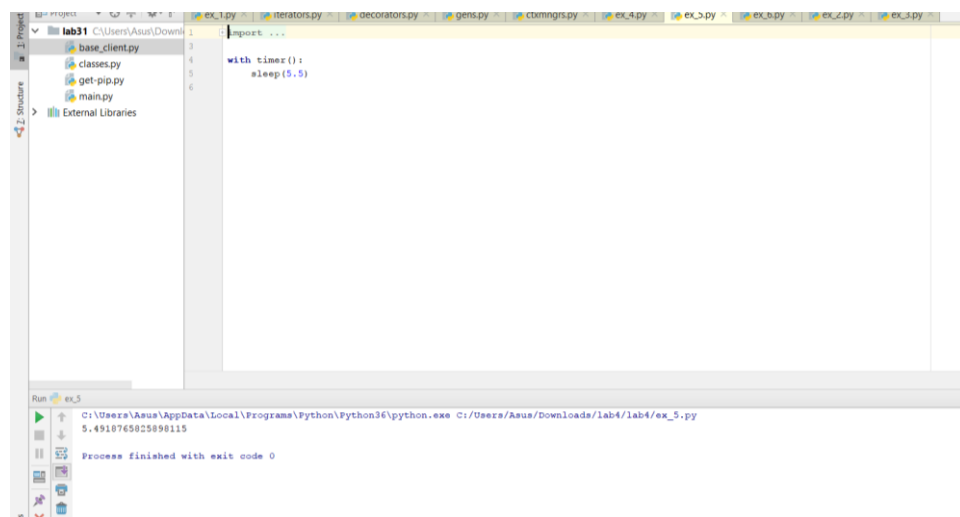


The screenshot shows an IDE with a project named 'lab31'. The file explorer on the left shows files: 'base_client.py', 'classes.py', 'get-pip.py', 'main.py', and 'External Libraries'. The main editor displays a Python script with the following code:

```
#!/usr/bin/env python3
import ...
goods = [
    {'title': 'Косеп', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
    {'title': 'Столешня', 'price': 7000, 'color': 'white'},
    {'title': 'Вешалка для одежды', 'price': 800, 'color': 'white'}
]

# Реализация задания 1
#print()
print(field(goods, 'title', 'price'), gen_random(1, 5, 5))
```

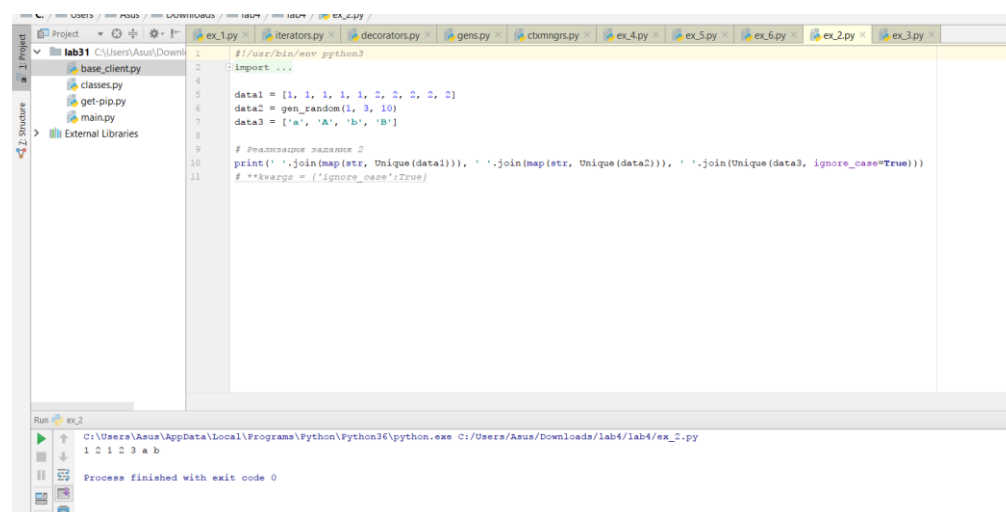
The output window at the bottom shows the command: `C:\Users\Asus\AppData\Local\Programs\Python\Python36\python.exe C:/Users/Asus/Downloads/lab4/lab4/ex_1.py` and the output: `[{'title': 'Косеп', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}, {'title': 'Столешня', 'price': 7000}, {'title': 'Вешалка для одежды', 'price': 800}] [3, 3, 3, 4, 4]`. The process finished with exit code 0.



The screenshot shows the same IDE with a different Python script. The file explorer shows the same files. The main editor displays the following code:

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

The output window shows the command: `C:\Users\Asus\AppData\Local\Programs\Python\Python36\python.exe C:/Users/Asus/Downloads/lab4/lab4/ex_5.py` and the output: `5.4918769825898115`. The process finished with exit code 0.



The screenshot shows the same IDE with a third Python script. The file explorer shows the same files. The main editor displays the following code:

```
#!/usr/bin/env python3
import ...
data1 = [1, 1, 1, 1, 1, 1, 0, 2, 2, 2, 2]
data2 = gen_random(1, 3, 10)
data3 = ['a', 'A', 'b', 'B']

# Реализация задания 2
print(' '.join(map(str, Unique(data1))), ' '.join(map(str, Unique(data2))), ' '.join(Unique(data3, ignore_case=True)))
# **kwargs = {'ignore_case': True}
```

The output window shows the command: `C:\Users\Asus\AppData\Local\Programs\Python\Python36\python.exe C:/Users/Asus/Downloads/lab4/lab4/ex_5.py` and the output: `1 2 1 0 3 a b`. The process finished with exit code 0.