|  |  |  |
| --- | --- | --- |
|  |  |  |

**Отчет по лабораторной работе № 4 по курсу**

**РИП**

#### Тема работы: "Python. Функциональные возможности."

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

Студент группы ИУ 5-51ц

Щипицин Р.А.

Москва, МГТУ - 2016

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Описание задания лабораторной работы.

# 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 в​ озвращаемые значения.

# Пример: data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2] Unique(data) будет последовательно возвращать только 1 и 2МГТУ 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 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 1МГТУ 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 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

**3. Исходный код**

**Задача 1 (ex\_1.py)**

*#!/usr/bin/env python3***from** librip.gen **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(list(field (goods, **'title'**)))  
print(**'\n'**)  
print(list(field(goods, **'title'**, **'color'**)))  
print(**'\n'**)  
print(list(gen\_random(1, 3, 5)))

**Задача 2 (ex\_2.py)**

*#!/usr/bin/env python3***from** librip.gen **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*print(list(Unique(data1)))  
  
print(list(Unique(data2)))  
  
print(list(Unique(data3)))  
  
print(list(Unique(data3, ignore\_case=**True**)))

**Задача 3 (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)))

**Задача 4 (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\_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()

**Задача 5 (ex\_5.py)**

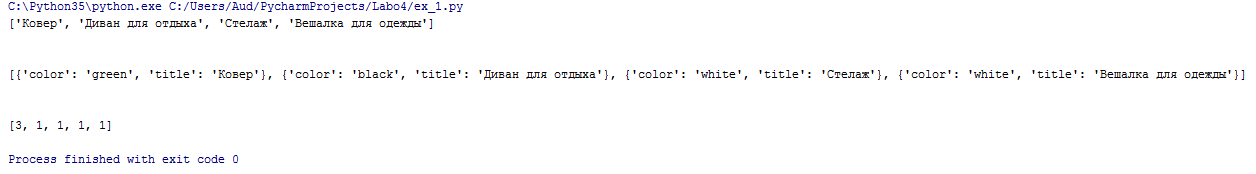
**from** time **import** sleep  
**from** librip.ctxmngrs **import** timer  
  
**with** timer():  
 sleep(5.5)

**Задача 6 (ex\_6.py)**

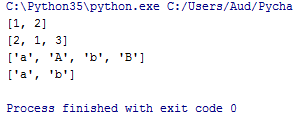
*#!/usr/bin/env python3***import** json  
**import** xml  
**import** sys  
**from** librip.ctxmngrs **import** timer  
**from** librip.decorators **import** print\_result  
**from** librip.gen **import** \*  
**from** librip.iterators **import** Unique  
  
path = **"data\_light.json"***# Здесь необходимо в переменную path получить  
# путь до файла, который был передан при запуске***with** open(path, encoding=**"utf8"**) **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** list(Unique(list(field(arg, **"job-name"**)), ignore\_case=**True**))  
  
  
@print\_result  
**def** f2(arg):  
 **return** list(filter(**lambda** \_: **"Программист" in** \_, arg))  
  
  
@print\_result  
**def** f3(arg):  
 **return** list(map(**lambda** x: x + **" с опытом Python"**, arg))  
  
  
@print\_result  
**def** f4(arg):  
 **return** list(map(**lambda** x: **"{}, зарплата {} руб."**.format(x[0], x[1]),  
 zip(arg, gen\_random(100000, 200000, len(arg)))))  
  
  
**with** timer():  
 f4(f3(f2(f1(data))))

**4. Скриншоты с результатами выполнения**

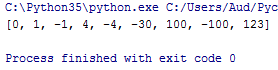
**Задача 1 (ex\_1.py)**

****

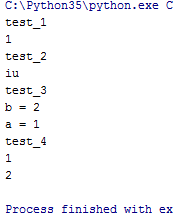
**Задача 2 (ex\_2.py)**

****

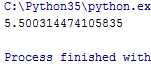
**Задача 3 (ex\_3.py)**

****

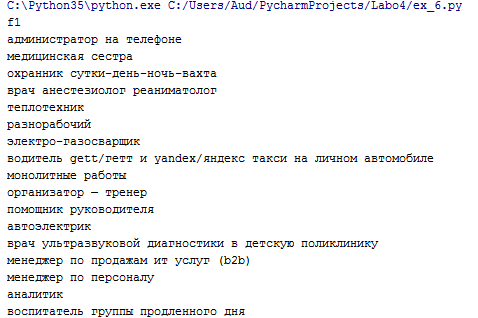
**Задача 4 (ex\_4.py)**

****

**Задача 5 (ex\_5.py)**

****

**Задача 6 (ex\_6.py)**

****