

Лабораторная работа №4 по дисциплине РИП
"Python. Функциональные возможности"

ИСПОЛНИТЕЛЬ:

студент группы ИУ5-53
Пьянзин С.А.

"__" _____ 2016 г.

1. Описание задания лабораторной работы

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

4. *Задача 1 (ex_1.py)*

Необходимо реализовать генераторы `field` и `gen_random`

Генератор `field` последовательно выдает значения ключей словарей массива

Генератор `gen_random` последовательно выдает заданное количество случайных чисел в заданном диапазоне

5. *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`.

6. *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`.

7. *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.

8. *Задача 5 (ex_5.py)*

Необходимо написать контекстный менеджер, который считает время работы блока и выводит его на экран.

9. *Задача 6 (ex_6.py)*

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:

- Функция `f1` должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна игнорировать регистр.
- Функция `f2` должна фильтровать входной массив и возвращать только те элементы, которые начинаются со слова "программист".
- Функция `f3` должна модифицировать каждый элемент массива, добавив строку "с опытом Python" (все программисты должны быть знакомы с Python).

Функция `f4` должна сгенерировать для каждой специальности зарплату от 100 000 до 200 000 рублей и присоединить её к названию специальности.

2. Исходный код

2.1. ctxmgrs.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
```

```
# Класс контекстного менеджера
class timer:
    def __enter__(self):
        self.start = time.clock()

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

2.2. 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 выводиться в столбик
# через знак равно
```

```
def print_result(func_arg):
    def decorated_func(*args):
        print(func_arg.__name__)
        # Если возвращает список - печатать в столбик
        if type(func_arg(*args)) == list:
            for i in func_arg(*args):
                print(i)
        # Если словарь - печатать парами ключ-значение
        elif type(func_arg(*args)) == dict:
            for key, val in func_arg(*args).items():
                print('{} = {}'.format(key, val))
        # Во всех прочих случаях - выводить результат как есть
        else:
            print(func_arg(*args))
    return decorated_func
```

2.3. gen.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
    if len(args) == 1:
        for i in items:
            if args[0] in i: yield i[args[0]]
    else:
        for i in items:
            res = {}
            for j in args:
                if j in i:
                    res[j] = i[j]
            yield res

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

2.4. iterators.py

```
# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, **kwargs):
        # 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

        # Проверка наличия ключевого аргумента 'ignore_case' и его значения
        if ('ignore_case' in kwargs.keys()) and (kwargs['ignore_case']):
            # Игнорирование регистра - приведение всех строк из списка к нижнему
            регистру
            self.items = [str(i).lower() for i in items]
        else:
            self.items = items
        self.index = 0
        self.used = []
```

```

def __next__(self):
    # Нужно реализовать __next__
    # Проходим по списку использованных элементов - если текущего элемента в нём
нет, то добавляем его и выводим
    while self.items[self.index] in self.used:
        if self.index == len(self.items) - 1:
            raise StopIteration
        self.index += 1

    self.used.append(self.items[self.index])
    return self.items[self.index]

def __iter__(self):
    return self

```

2.5. ex_6.py

```

#!/usr/bin/env python3
import os.path
import json
import sys
from librip.ctxmgrs import timer
from librip.decorators import print_result
from librip.gen import field, gen_random
from librip.iterators import Unique as unique

path = os.path.abspath(sys.argv[1])

# Здесь необходимо в переменную path получить
# путь до файла, который был передан при запуске

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

# Функция для вывода отсортированного списка профессий без повторений
def f1(arg):
    return(sorted([i for i in unique([j['job-name'] for j in arg], ignore_case =
True)]))

# Функция для отбора профессий со словом "программист" в начале
def f2(arg):
    return(filter(lambda x: (x.lower().find('программист') == 0), arg))

# Функция модификации профессии
def f3(arg):
    return(["{} {}".format(x, "с опытом Python") for x in arg])

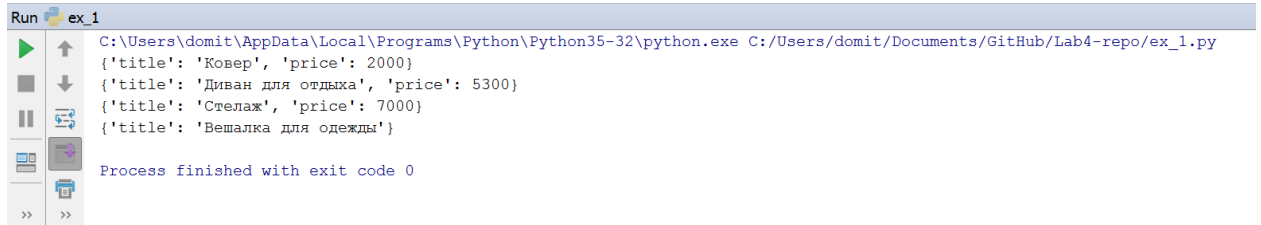
# Функция генерации размера зарплаты для профессий
@print_result
def f4(arg):
    return(["{}", {} {} {}".format(x, "зарплата", y, "руб.") for x, y in zip(arg,
list(gen_random(100000, 200000, len(arg))) )])

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

```

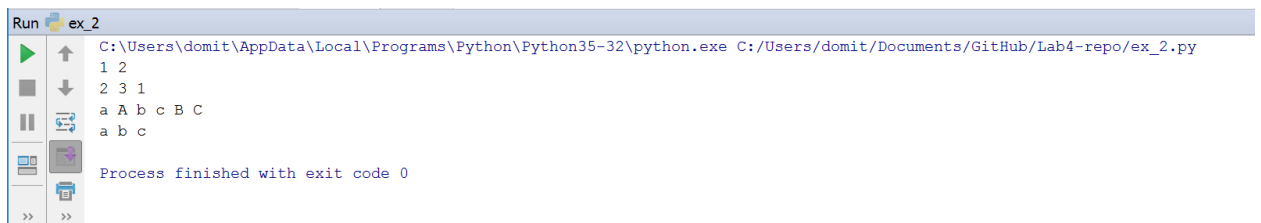
3. Снимки экрана с результатом выполнения работы программы

3.1. ex_1.py



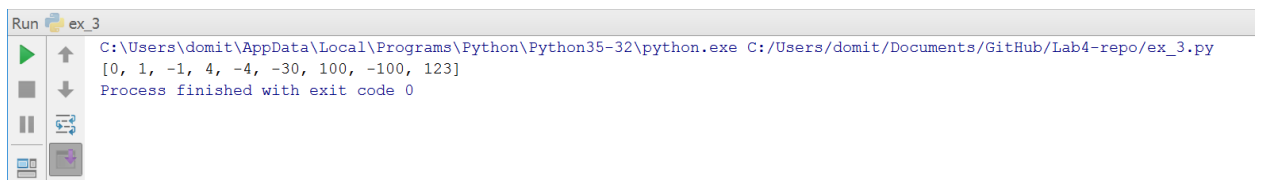
```
Run ex_1
C:\Users\domit\AppData\Local\Programs\Python\Python35-32\python.exe C:/Users/domit/Documents/GitHub/Lab4-repo/ex_1.py
{'title': 'Ковёр', 'price': 2000}
{'title': 'Диван для отдыха', 'price': 5300}
{'title': 'Стелаж', 'price': 7000}
{'title': 'Вешалка для одежды'}
Process finished with exit code 0
```

3.2. ex_2.py



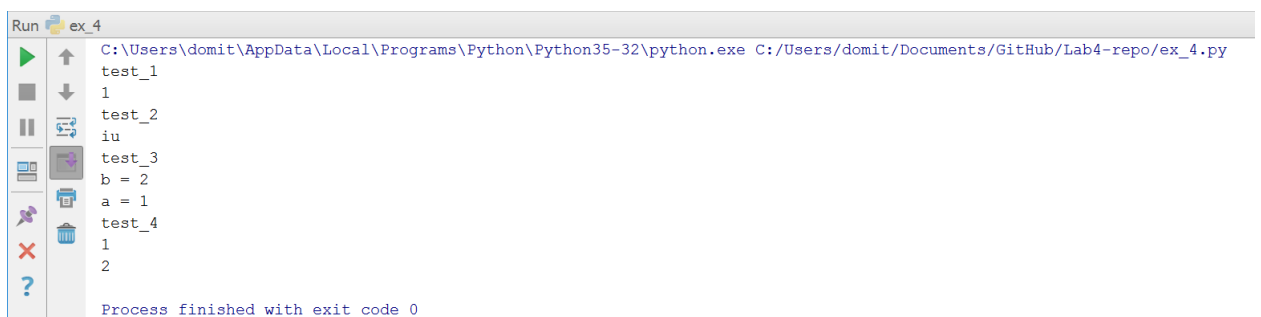
```
Run ex_2
C:\Users\domit\AppData\Local\Programs\Python\Python35-32\python.exe C:/Users/domit/Documents/GitHub/Lab4-repo/ex_2.py
1 2
2 3 1
a A b c B C
a b c
Process finished with exit code 0
```

3.3. ex_3.py



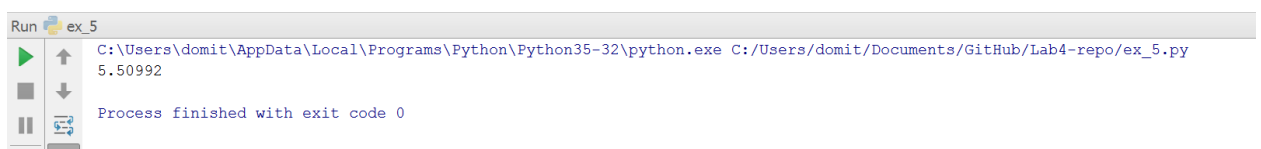
```
Run ex_3
C:\Users\domit\AppData\Local\Programs\Python\Python35-32\python.exe C:/Users/domit/Documents/GitHub/Lab4-repo/ex_3.py
[0, 1, -1, 4, -4, -30, 100, -100, 123]
Process finished with exit code 0
```

3.4. ex_4.py



```
Run ex_4
C:\Users\domit\AppData\Local\Programs\Python\Python35-32\python.exe C:/Users/domit/Documents/GitHub/Lab4-repo/ex_4.py
test_1
1
test_2
iu
test_3
b = 2
a = 1
test_4
1
2
Process finished with exit code 0
```

3.5. ex_5.py



```
Run ex_5
C:\Users\domit\AppData\Local\Programs\Python\Python35-32\python.exe C:/Users/domit/Documents/GitHub/Lab4-repo/ex_5.py
5.50992
Process finished with exit code 0
```

3.6. ex_6.py

```
Run ex_6
C:\Users\domit\AppData\Local\Programs\Python\Python35-32\python.exe C:/Users/domit/Documents/GitHub/Lab4-repo/ex_6.py data_light.json
f4
программист с опытом Python, зарплата 189138 руб.
программист / senior developer с опытом Python, зарплата 104199 руб.
программист 1с с опытом Python, зарплата 147148 руб.
программист с# с опытом Python, зарплата 198263 руб.
программист с++ с опытом Python, зарплата 136287 руб.
программист с++/с#/java с опытом Python, зарплата 151881 руб.
программист/ junior developer с опытом Python, зарплата 175331 руб.
программист/ технический специалист с опытом Python, зарплата 109504 руб.
программистр-разработчик информационных систем с опытом Python, зарплата 110315 руб.
0.12772661728395063

Process finished with exit code 0
```