

**Министерство образования и науки Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего профессионального образования**

**«Московский государственный технический университет имени Н.Э. Баумана»**

**(МГТУ им. Н.Э. Баумана)**

|  |  |
| --- | --- |
| **ФАКУЛЬТЕТ** | **«Информатики и систем управления»** |
| КАФЕДРА | ИУ5 |

Дисциплина «Разработка интернет-приложений»

Отчет по лабораторной работе №3

«Функциональные возможности языка Python.»

|  |  |  |
| --- | --- | --- |
| Студент | группы ИУ5-52Б | Гришин Илья |
| Преподаватель |  | Гапанюк Ю.Е. |

**Цель лабораторной работы:** изучение возможностей функционального программирования в языке Python.

1. **Задание**

Задание лабораторной работы состоит из решения нескольких задач.

Файлы, содержащие решения отдельных задач, должны располагаться в пакете lab\_python\_fp. Решение каждой задачи должно раполагаться в отдельном файле.

При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

**Задача 1 (файл field.py)**

Необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря.

**Задача 2 (файл gen\_random.py)**

Необходимо реализовать генератор gen\_random(количество, минимум, максимум), который последовательно выдает заданное количество случайных чисел в заданном диапазоне от минимума до максимума, включая границы диапазона.

**Задача 3 (файл unique.py)**

Необходимо реализовать итератор Unique(данные), который принимает на вход массив или генератор и итерируется по элементам, пропуская дубликаты.

**Задача 4 (файл sort.py)**

Дан массив 1, содержащий положительные и отрицательные числа. Необходимо **одной строкой кода** вывести на экран массив 2, которые содержит значения массива 1, отсортированные по модулю в порядке убывания. Сортировку необходимо осуществлять с помощью функции sorted.

**Задача 5 (файл print\_result.py)**

Необходимо реализовать декоратор print\_result, который выводит на экран результат выполнения функции.

Декоратор должен принимать на вход функцию, вызывать её, печатать в консоль имя функции и результат выполнения, после чего возвращать результат выполнения.

Если функция вернула список (list), то значения элементов списка должны выводиться в столбик.

Если функция вернула словарь (dict), то ключи и значения должны выводить в столбик через знак равенства.

**Задача 6 (файл cm\_timer.py)**

Необходимо написать контекстные менеджеры cm\_timer\_1 и cm\_timer\_2, которые считают время работы блока кода и выводят его на экран.

**Задача 7 (файл process\_data.py)**

В предыдущих задачах были написаны все требуемые инструменты для работы с данными. Применим их на реальном примере.

В файле [data\_light.json](https://github.com/iu5team/iu5web-fall-2020/tree/master/labs/lab3/data/data_light.json) содержится фрагмент списка вакансий.

Структура данных представляет собой список словарей с множеством полей: название работы, место, уровень зарплаты и т.д.

Необходимо реализовать 4 функции - f1, f2, f3, f4. Каждая функция вызывается, принимая на вход результат работы предыдущей. За счет декоратора @print\_result печатается результат, а контекстный менеджер cm\_timer\_1 выводит время работы цепочки функций.

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 содержат слово “программист”.

Функция f3 должна модифицировать каждый элемент массива, добавив строку “с опытом Python”.

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

1. **Текст программ**

field.py

goods = [

    {'title': 'Ковер', 'price': 3000, 'color': 'green'},

    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},

    {'title': 'Стул', 'price': None, 'color': 'green'},

    {'title': None, 'price': None, 'color': 'green'},

]

def field(items,\*args):

    assert len(args) > 0, 'Не переданы аргументы полей словаря'

    if len(args) == 1:

        for i in range(len(items)):

            if (items[i].get(args[0])!=None):

                yield items[i].get(args[0])

    else:

        for i in range(len(items)):

            d={}

            for j in range(len(args)):

                if (items[i].get(args[j])!=None):

                    d.update({args[j]: items[i].get(args[j])})

            if (d!={}):

                yield d

def main():

    f=field(goods, 'title')

    for i in f:

        print(i, end=';  ')

    print('\n',end='')

    f=field(goods, 'title', 'price')

    for i in f:

        print(i, end=';  ')

if \_\_name\_\_ == "\_\_main\_\_":

   main()

    main()

gen\_random.py

import random

def gen\_random(num\_count, begin, end):

    for i in range(num\_count):

        yield random.randint(begin, end)

def main():

    gen = gen\_random(5, 1, 3)

    for i in gen:

        print(i, end=' ')

    print('\n',end='')

    gen = gen\_random(10, 0, 10)

    for i in gen:

        print(i, end=' ')

if \_\_name\_\_ == "\_\_main\_\_":

   main()

unique.py

from gen\_random import gen\_random

class Unique(object):

    def \_\_init\_\_(self, items, \*\*kwargs):

        self.set\_unique = set()

        self.items = items

        if len(kwargs) != 0:

            self.ignore\_case = kwargs

        else:

            self.ignore\_case = False

    def \_\_iter\_\_(self):

            return self

    def \_\_next\_\_(self):

        while True:

            for item in self.items:

                temp\_item = item

                if (temp\_item not in self.set\_unique) and not(self.ignore\_case and temp\_item.swapcase() in self.set\_unique):

                    self.set\_unique.add(temp\_item)

                    return temp\_item

            else:

                raise StopIteration

def main():

    data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]

    print(data1)

    itr1 = Unique(data1)

    for i1 in itr1:

        print(i1, end=' ')

    print('\n', end='')

    data2 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']

    print(data2)

    itr2 = Unique(data2)

    for i2 in itr2:

        print(i2, end=' ')

    print('\n', end='')

    print(data2)

    itr3 = Unique(data2, ignor\_case=True)

    for i3 in itr3:

        print(i3, end=' ')

    print('\n', end='')

    data3 = gen\_random(5, 1, 3)

    for i4 in data3:

        print(i4, end=' ')

    print('\n', end='')

    itr4 = Unique(data3)

    for i5 in itr4:

        print(i5, end=' ')

if \_\_name\_\_ == "\_\_main\_\_":

    main()

sort.py

def main():

    data = [4, -30, 100, -100, 123, 1, 0, -1, -4]

    result = sorted(data, key=abs, reverse=True)

    print(result)

    result\_with\_lambda = sorted(data, key=lambda x: abs(x), reverse=True)

    print(result\_with\_lambda)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

print\_result.py

def print\_result(func):

    def decorated\_func(\*args):

        print(func.\_\_name\_\_)

        result = func(\*args)

        if (type(result)==list):

            for i in result:

                print(i)

        else:

            if (type(result)==dict):

                for i in result:

                    print(i, '-->', result.get(i))

            else:

                print(result)

        return result

    return decorated\_func

@print\_result

def test\_1():

    return 1

@print\_result

def test\_2():

    return 'iu5'

@print\_result

def test\_3():

    return {'a': 1, 'b': 2}

@print\_result

def test\_4():

    return [1, 2]

def main():

    print('!!!!!!!!')

    test\_1()

    test\_2()

    test\_3()

    test\_4()

if \_\_name\_\_ == '\_\_main\_\_':

    main()

cm\_timer.py

import time

from contextlib import contextmanager

class cm\_timer\_1:

    def \_\_init\_\_(self):

        self.begin\_time = time.time()

    def \_\_enter\_\_(self):

        pass

    def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):

        print('time: ', time.time() - self.begin\_time)

@contextmanager

def cm\_timer\_2():

    begin\_time = time.time()

    yield 1

    print('time: ', time.time() - begin\_time)

def main():

    with cm\_timer\_1():

        time.sleep(5.5)

    with cm\_timer\_2():

        time.sleep(2)

if \_\_name\_\_ == '\_\_main\_\_':

    main()

process\_data.py

from lab\_python\_fp.cm\_timer import cm\_timer\_1

from lab\_python\_fp.print\_result import print\_result

from lab\_python\_fp.unique import Unique

from lab\_python\_fp.field import field

from lab\_python\_fp.gen\_random import gen\_random

import re

import json

import sys

path = 'data\_light.json'

with open(path, encoding='utf-8') as f:

    data = json.load(f)

@print\_result

def f1(arg):

    return Unique(field(arg, 'job-name'), ignore\_case=True)

@print\_result

def f2(arg):

    return filter(lambda x: re.search('Программист', x) or re.search('программист', x), 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: x + ", зарплата " + str(\*gen\_random(1, 100000, 200000)) + " руб", arg))

def main():

    with cm\_timer\_1():

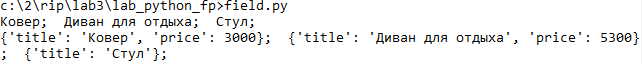
        f4(f3(f2(f1(data))))

if \_\_name\_\_ == "\_\_main\_\_":

    main()

1. **Результаты**

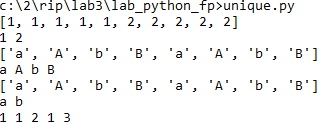
field.py



gen\_random.py



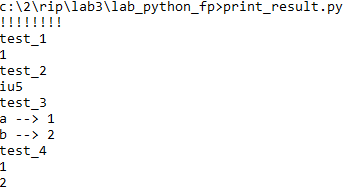
unique.py



sort.py



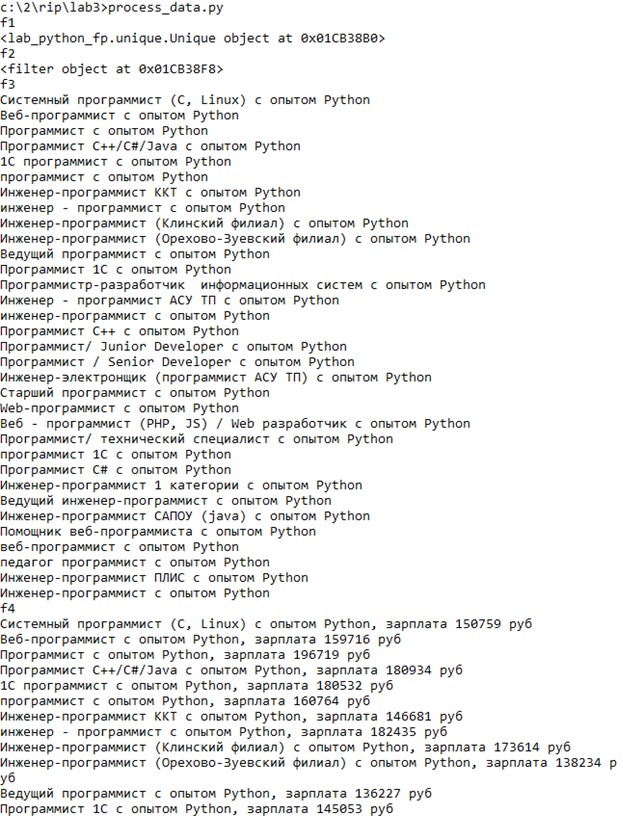
print\_result.py

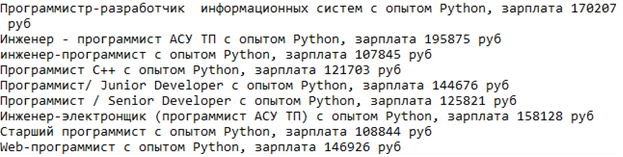


cm\_timer.py



process\_data.py



****

