**Отчёт по лабораторным работам №5-6**

**Функциональные возможности языка Python.**

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

**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.
* Конструктор итератора также принимает на вход именованный bool-параметр ignore\_case, в зависимости от значения которого будут считаться одинаковыми строки в разном регистре. По умолчанию этот параметр равен False.
* При реализации необходимо использовать конструкцию \*\*kwargs.
* Итератор должен поддерживать работу как со списками, так и с генераторами.
* Итератор не должен модифицировать возвращаемые значения.

**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/ugapanyuk/BKIT_2021/blob/main/notebooks/fp/files/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.
* Предполагается, что функции f1, f2, f3 будут реализованы в одну строку. В реализации функции f4 может быть до 3 строк.
* Функция f1 должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна игнорировать регистр. Используйте наработки из предыдущих задач.
* Функция f2 должна фильтровать входной массив и возвращать только те элементы, которые начинаются со слова “программист”. Для фильтрации используйте функцию filter.
* 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.
* 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.

**Текст программы**

goods = [

    {'title': 'Ковер', 'price': 2000, 'color': 'green'},

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

]

def field(items, \*args):

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

    if len(args) == 1:

        for i in range(len(items)):

            if args[0] in items[i] and items[i].get(args[0]) is not None:

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

    else:

        for i in range(len(items)):

            s = {}

            for j in range(len(args)):

                if args[j] in items[i] and items[i].get(args[j]) is not None:

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

            yield s

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()

import random

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

    ans = []

    for i in range(num\_count):

        ans.append(random.randint(begin, end))

    return ans

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

if \_\_name\_\_ == "\_\_main\_\_":

    result = []

    temp = 0

    for i in range(len(data)):

        for j in range(len(data)):

            if (abs(data[i]) > abs(data[j])):

                temp = data[j]

                data[j] = data[i]

                data[i] = temp

    result = data.copy()

    print(result)

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

    print(result\_with\_lambda)

class Unique(object):

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

        self.used\_elements = set()

        self.items = items

        self.counter = 0

        if len(kwargs) != 0:

            self.ignore\_case = kwargs

        else:

            self.ignore\_case = False

    def \_\_next\_\_(self):

        while True:

            for item in self.items:

                temp\_item = item

                self.counter += 1

                if (temp\_item not in self.used\_elements):

                  if not(self.ignore\_case):

                      self.used\_elements.add(temp\_item)

                      return temp\_item

                  else:

                   if not(temp\_item.swapcase() in self.used\_elements):

                       self.used\_elements.add(temp\_item)

                       return temp\_item

            else:

                raise StopIteration

    def \_\_iter\_\_(self):

        return self

def print\_result(func):

    def decorated\_func(\*args):

        print(func.\_\_name\_\_)

        result = func(\*args)

        if type(result) is list:

            for i in result:

                print(i)

        elif type(result) is dict:

            for i in result:

                print(i, result.get(i), sep = ' = ')

        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]

#if \_\_name\_\_ == '\_\_main\_\_':

#   print('!!!!!!!!')

#    test\_1()

#    test\_2()

#    test\_3()

#    test\_4()

from contextlib import contextmanager

import time

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):

        if exc\_type is not None:

            print(exc\_type, exc\_val, exc\_tb)

        else:

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

@contextmanager

def cm\_timer\_2():

    begin\_time = time.time()

    yield 1

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

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

path = "lab3/data\_light.json"

with open(path) as f:

    data = json.load(f)

@print\_result

def f1(arg):

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

@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):

    price = gen\_random(len(arg), 100000, 200000)

    res = list(zip(arg, (list(map(lambda x: ', зарплата ' + x + ' руб', ''.join(str(list(price)))[1:-1].split(', '))))))

    return [''.join(i) for i in res]

def main():

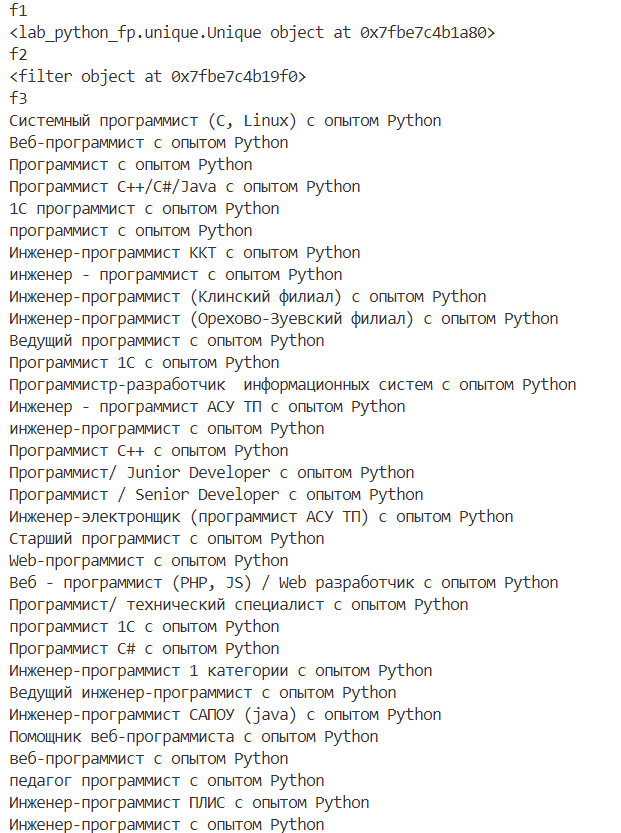
    with cm\_timer\_1():

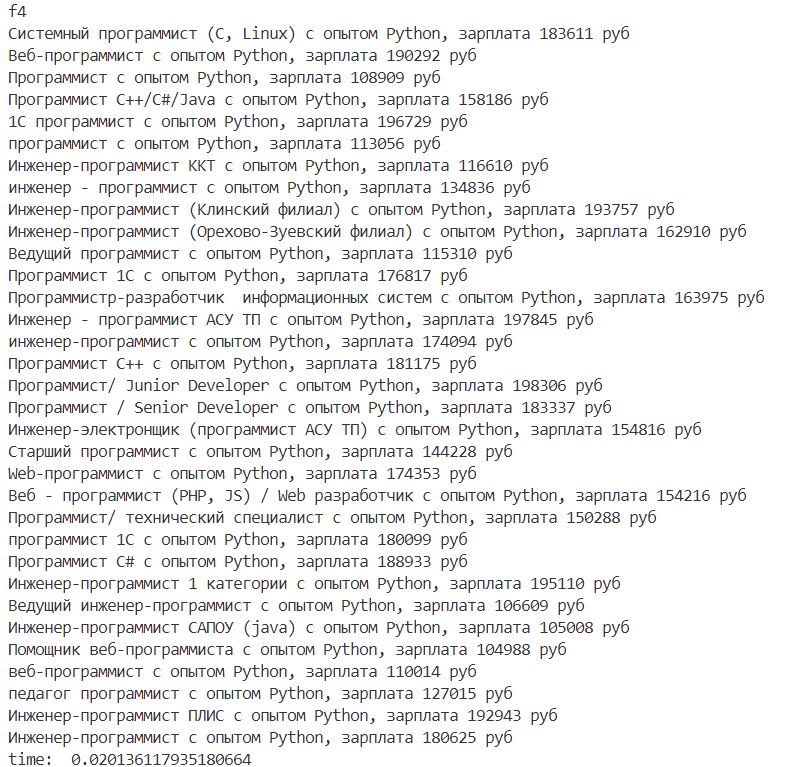
        f4(f3(f2(f1(data))))

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Пример выполнения программы**

****

****