**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет Радиотехнический

Кафедра “Системы обработки информации и управления”

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторным работам №3-4

«Функциональные возможности языка Python»

Вариант №12

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы РТ5-31Б: |  | преподаватель каф. ИУ5 |
| Мусаев Максим  Шахинович |  | Гапанюк Юрий  Евгеньевич |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2025 г.

**Постановка задачи**

Реализовать набор функций и инструментов для обработки данных: генератор field для извлечения значений из словарей с обработкой None-значений, генератор случайных чисел gen\_random, итератор Unique для фильтрации дубликатов с поддержкой игнорирования регистра, два способа сортировки по модулю (с lambda и без), декоратор print\_result для форматированного вывода результатов функций (списков и словарей), контекстные менеджеры cm\_timer для замера времени выполнения, а также цепочку функций (f1-f4) для обработки JSON-данных: получения уникальных профессий без учета регистра, фильтрации по слову "программист", добавления фразы "с опытом Python" и генерации зарплат с последующим выводом результатов и замерем времени выполнения всей цепочки.

Текст программы

field.py

def field(data:list[dict], \*args):

    assert len(args) > 0

    if len(args) == 1:

        return [item[args[0]] for item in data if item[args[0]]]

    else:

        return [{arg: item[arg] for arg in args if item[arg]} for item in data]

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

    goods = [

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

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

    ]

    print(\*field(goods, 'title'), sep=', ')

    print(\*field(goods, 'title', 'price'), sep=', ')

gen\_random.py

import random

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

    return [random.randint(begin, end) for \_ in range(num\_count)]

def main():

    print(gen\_random(5, 1, 3))

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

    main()

unique.py

from gen\_random import gen\_random

class Unique(object):

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

        self.ignore\_case = kwargs.get('ignore\_case', False)

        self.items = iter(items)

        self.seen = set()

    def \_\_next\_\_(self):

        while True:

            item = next(self.items)

            if self.ignore\_case and type(item) == str:

                key = item.lower()

            else:

                key = item

            if key not in self.seen:

                self.seen.add(key)

                return item

    def \_\_iter\_\_(self):

        return self

def main():

    data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]

    print(\*Unique(data, ignore\_case=True))

    data = gen\_random(10, 1, 3)

    print(\*Unique(data, ignore\_case=True))

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

    print(\*Unique(data, ignore\_case=True))

    print(\*Unique(data))

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

    main()

sort.py

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

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

    result = [x for \_, x in sorted([(abs(x), x) for x in data], reverse=True)]

    print(result)

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

    print(result\_with\_lambda)

print\_result.py

def print\_result(func):

    def wrapper(\*args, \*\*kwargs):

        print(func.\_\_name\_\_)

        result = func(\*args, \*\*kwargs)

        if type(result) == list:

            print(\*result, sep='\n')

        elif type(result) == dict:

            print(\*[f'{key} = {value}' for key, value in result.items()], sep='\n')

        else:

            print(result)

        return result

    return wrapper

@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()

cm\_timer.py

import time

import contextlib

class cm\_timer\_1:

    def \_\_enter\_\_(self):

        self.start = time.time()

        return self

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

        end\_time = time.time()

        elapsed\_time = end\_time - self.start

        print(f'time: {elapsed\_time:.1f}')

@contextlib.contextmanager

def cm\_timer\_2():

    start\_time = time.time()

    yield

    end\_time = time.time()

    elapsed\_time = end\_time - start\_time

    print(f'time: {elapsed\_time:.1f}')

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

    with cm\_timer\_1():

        time.sleep(5.5)

    with cm\_timer\_2():

        time.sleep(5.5)

process\_data.py

import json

import sys

from print\_result import print\_result

from cm\_timer import cm\_timer\_1

from unique import Unique

from field import field

import random

path = r'lab\_python\_fp\data\_light.json'

# Необходимо в переменную path сохранить путь к файлу, который был передан при запуске сценария

with open(path) as f:

    data = json.load(f)

# Далее необходимо реализовать все функции по заданию, заменив `raise NotImplemented`

# Предполагается, что функции f1, f2, f3 будут реализованы в одну строку

# В реализации функции f4 может быть до 3 строк

@print\_result

def f1(arg):

    return sorted(Unique(sorted(field(arg, 'job-name')), ignore\_case=True), key=lambda x: x.lower())

@print\_result

def f2(arg):

    return list(filter(lambda x: x.lower()[:11] == 'программист', arg))

@print\_result

def f3(arg):

    return list(map(lambda x: x + ' с опытом Python', arg))

@print\_result

def f4(arg):

    return [item + f', зарплата {salary} руб.' for item, salary in zip(arg, [random.randint(100000, 200000) for \_ in range(len(arg))])]

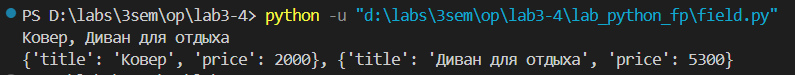
if \_\_name\_\_ == '\_\_main\_\_':

    with cm\_timer\_1():

        f4(f3(f2(f1(data))))

Анализ результатов

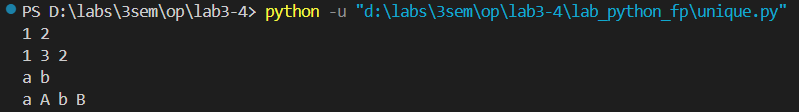
1. Работа файла field.py



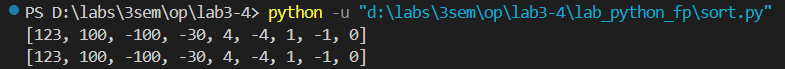
1. Работа файла gen\_random.py



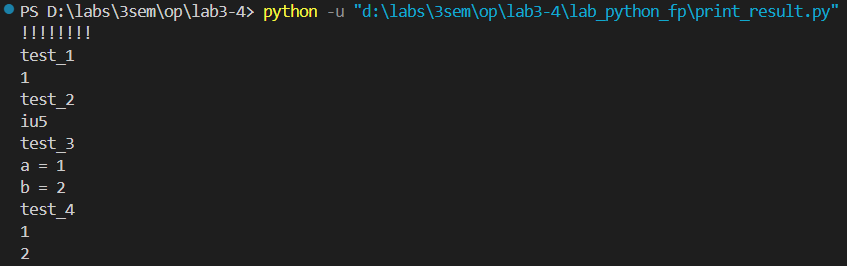
1. Работа файла unique.py



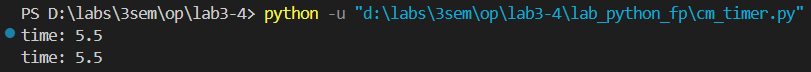
1. Работа файла sort.py



1. Работа файла print\_result.py



1. Работа файла cm\_timer.py



1. Работа файла process\_data.py

