*осударственное образовательное учреждение высшего профессионального образования*



ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Системы обработки информации и управления» (ИУ-5)

**ЛАБОРАТОРНАЯ РАБОТА**

**№ 3-4**

\_\_\_\_\_\_\_\_\_\_\_\_\_Функциональные возможности языка Python\_\_\_\_\_\_\_\_\_\_\_\_\_

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

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

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

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

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

Группа ИУ5-35Б

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **16.12.2024** /**А. А. Торопыгин/** 

(Подпись, дата) (И.О.Фамилия)

Преподаватель \_\_\_\_\_\_\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** /**Ю. Е. Гапанюк/**

(Подпись, дата) (И.О.Фамилия)

2024

**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 раcполагаться 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.

* В качестве первого аргумента генератор принимает список словарей, дальше через \*args генератор принимает неограниченное количествово аргументов.
* Если передан один аргумент, генератор последовательно выдает только значения полей, если значение поля равно None, то элемент пропускается.
* Если передано несколько аргументов, то последовательно выдаются словари, содержащие данные элементы. Если поле равно None, то оно пропускается. Если все поля содержат значения None, то пропускается элемент целиком.

**Текст программы:**

def field(items, \*args):

assert len(args) > 0

result = []

if len(args) == 1:

for item in items:

result.append(item[args[0]])

else:

for item in items:

result.append({i: item[i] for i in args if i in item.keys()})

return result

goods = [

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

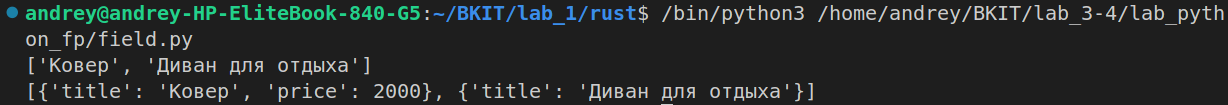
{'title': 'Диван для отдыха', 'color': 'black'}

]

print(field(goods, 'title'))

print(field(goods, 'title', 'price'))

**Вывод:**



### **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.

**Текст программы:**

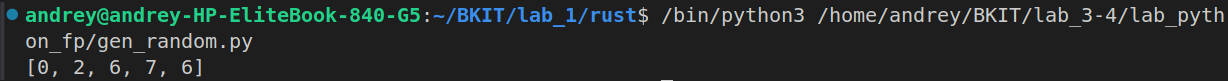
import random as rd

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

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

print(gen\_random(5, 0, 10))

**Вывод:**

****

### **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.

**Текст программы:**

class Unique(object):

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

self.items = []

if 'ignore\_case' in kwargs.keys() and kwargs['ignore\_case']:

self.items = list(set(map(lambda x: x.lower() if type(x) == str else x, items)))

else:

self.items = list(set(items))

self.start = 0

self.stop = len(self.items)

self.position = -1

def \_\_next\_\_(self):

self.position += 1

if self.position < self.stop:

return self.items[self.position]

else:

return ''

def \_\_iter\_\_(self):

return self

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

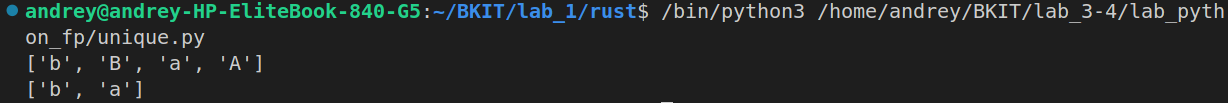
Udata = Unique(data)

print(Udata.items)

Udata = Unique(data, ignore\_case = True)

print(Udata.items)

**Вывод:**

****

**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.

Необходимо решить задачу двумя способами:

1. С использованием lambda-функции.
2. Без использования lambda-функции.

**Текст программы:**

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

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

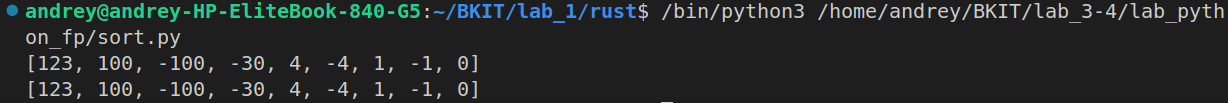
result = sorted(data, key=abs, reverse=True)

print(result)

result\_with\_lambda = sorted(data, key=lambda x: x if x > 0 else -x, reverse=True)

print(result\_with\_lambda)

**Вывод:**

****

**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.

**Текст программы:**

def print\_result(func):

def wrapper(\*args, \*\*kwargs):

print(func.\_\_name\_\_)

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

if type(result) == list:

print("\n".join(map(str, result)))

elif type(result) == dict:

for i in result.keys():

print(i, '=', result[i])

else:

print(result)

return func(\*args, \*\*kwargs)

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]

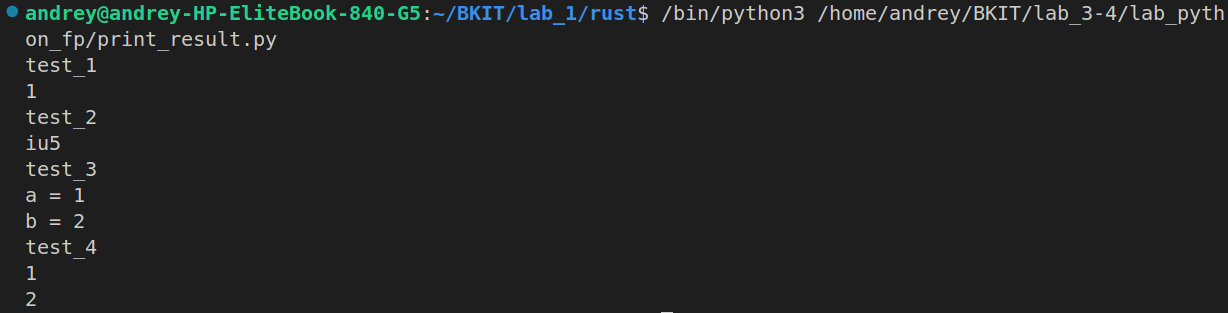
test\_1()

test\_2()

test\_3()

test\_4()

**Вывод:**

****

### **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.

cm\_timer\_1 и cm\_timer\_2 реализуют одинаковую функциональность, но должны быть реализованы двумя различными способами (на основе класса и с использованием библиотеки contextlib).

**Текст программы:**

import time

from contextlib import contextmanager

class cm\_timer\_1:

def \_\_enter\_\_(self):

self.start\_time = time.time()

return self

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

print(time.time() - self.start\_time)

@contextmanager

def cm\_timer\_2():

start\_time = time.time()

try:

yield

finally:

print(time.time() - start\_time)

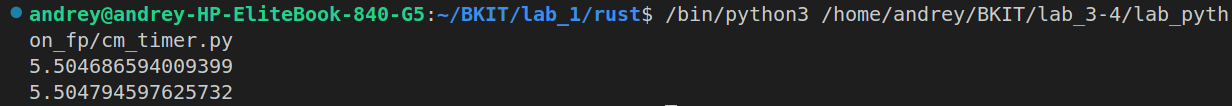
with cm\_timer\_1():

time.sleep(5.5)

with cm\_timer\_2():

time.sleep(5.5)

**Вывод:**

****

**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 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:**

import json

from field import field

from gen\_random import gen\_random

from unique import Unique

from print\_result import print\_result

from cm\_timer import cm\_timer\_1

path = '/home/andrey/BKIT/lab\_3-4/lab\_python\_fp/data\_light.json'

with open(path) as f:

data = json.load(f)

def f1(arg):

return sorted(Unique(field(arg, 'job-name')).items)

@print\_result

def f2(arg):

return list(filter(lambda x: True if x.lower().startswith('программист') else False, arg))

@print\_result

def f3(arg):

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

@print\_result

def f4(arg):

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

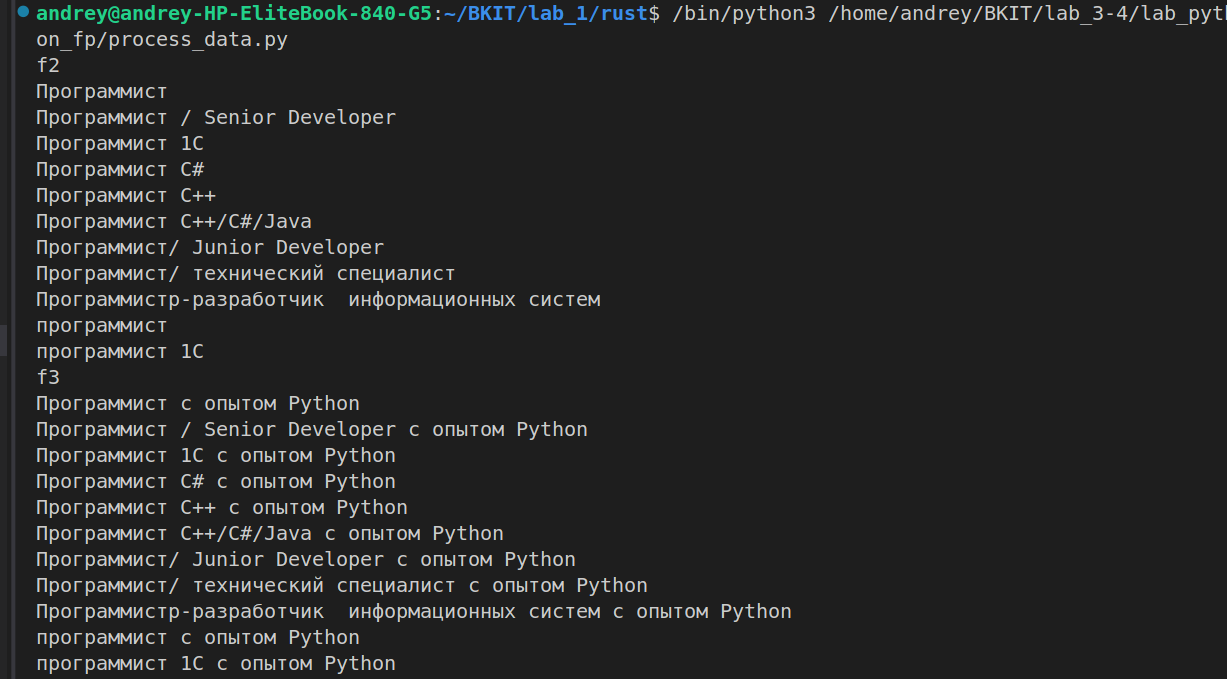
return [i + ', зарплата {} руб.'.format(j) for i, j in zip(arg, salaries)]

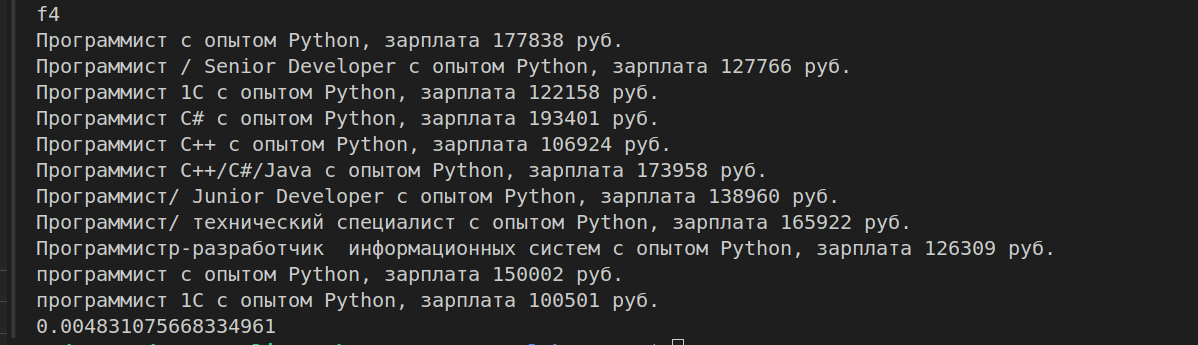
if \_\_name\_\_ == '\_\_main\_\_':

with cm\_timer\_1():

f4(f3(f2(f1(data))))

**Вывод:**

****

****