**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №3-4

«Функциональные возможности языка Python»

|  |  |  |
| --- | --- | --- |
| Выполнила: |  | Проверил: |
| Студентка группы ИУ5-33Б |  | преподаватель каф. ИУ5 |
| Коренева София |  | Гапанюк Ю. Е. |
| Подпись и дата: |  | Подпись и дата: |

2023 г.

**Общее описание задания**

Задание лабораторной работы состоит из решения нескольких задач.

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

При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

**Задача 1 (файл field.py)**

**Описание задачи**

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

goods = [

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

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

]

field(goods, 'title') должен выдавать 'Ковер', 'Диван для отдыха'

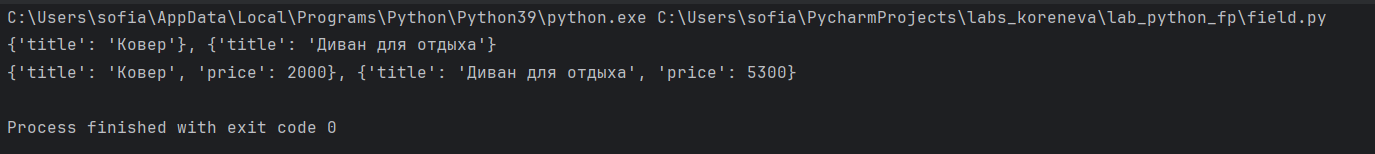
field(goods, 'title', 'price') должен выдавать {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}

* 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  
 res = []  
 for item in items:  
 tmp = {}  
 for arg in args:  
 if item[arg] is not None:  
 tmp[arg] = item[arg]  
 if tmp:  
 res.append(tmp)  
 return tuple(res)  
  
  
def task1():  
 goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}  
 ]  
 print(\*[el for el in field(goods, 'title')], sep=', ')  
 print(\*field(goods, 'title', 'price'), sep=', ')  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 task1()

**Примеры выполнения программы**

****

**Задача 2 (файл gen\_random.py)**

**Описание задачи**

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. Пример:

gen\_random(5, 1, 3) должен выдать 5 случайных чисел в диапазоне от 1 до 3, например 2, 2, 3, 2, 1.

**Текст программы**

from random import randint  
  
  
def gen\_random(num\_count, begin, end):  
 for i in range(num\_count):  
 yield randint(begin, end)  
  
  
def task2():  
 print(\*gen\_random(5, 1, 3), sep=', ')  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 task2()

**Примеры выполнения программы**



[**Задача 3 (файл unique.py)**](https://github.com/ugapanyuk/BKIT_2022/wiki/lab_python_fp#%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B0-3-%D1%84%D0%B0%D0%B9%D0%BB-uniquepy)

**Описание задачи**

* 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.

Пример:

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

Unique(data) будет последовательно возвращать только 1 и 2.

data = gen\_random(10, 1, 3)

Unique(data) будет последовательно возвращать только 1, 2 и 3.

data = [‘a’, ‘A’, ‘b’, ‘B’, ‘a’, ‘A’, ‘b’, ‘B’]

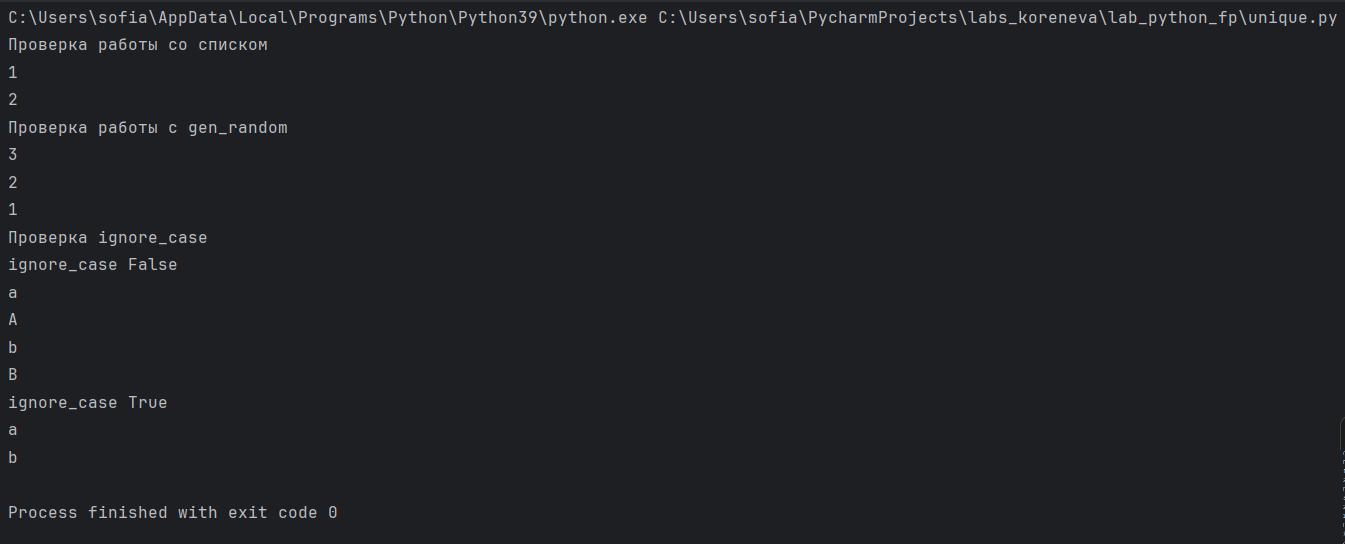
Unique(data) будет последовательно возвращать только a, A, b, B.

Unique(data, ignore\_case=True) будет последовательно возвращать только a, b.

**Текст программы**

from lab\_python\_fp.gen\_random import gen\_random  
  
  
class Unique(object):  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 self.res = []  
 for key, value in kwargs.items():  
 if key == 'ignore\_case' and value is True:  
 items = [i.lower() for i in items]  
 for item in items:  
 if item not in self.res:  
 self.res.append(item)  
 pass  
  
 def \_\_next\_\_(self):  
 try:  
 x = self.res[self.begin]  
 self.begin += 1  
 return x  
 except:  
 raise StopIteration  
  
 def \_\_iter\_\_(self):  
 self.begin = 0  
 return self  
  
  
def task3():  
 print('Проверка работы со списком')  
 data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
 for item in Unique(data):  
 print(item)  
 print('Проверка работы с gen\_random')  
 data = gen\_random(10, 1, 3)  
 for item in Unique(data):  
 print(item)  
 print('Проверка ignore\_case')  
 data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
 print('ignore\_case False')  
 for item in Unique(data):  
 print(item)  
 print('ignore\_case True')  
 for item in Unique(data, ignore\_case=True):  
 print(item)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 task3()

**Примеры выполнения программы**



**Задача 4 (файл sort.py)**

**Описание задачи**

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. Пример:

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

Вывод: [123, 100, -100, -30, 30, 4, -4, 1, -1, 0]

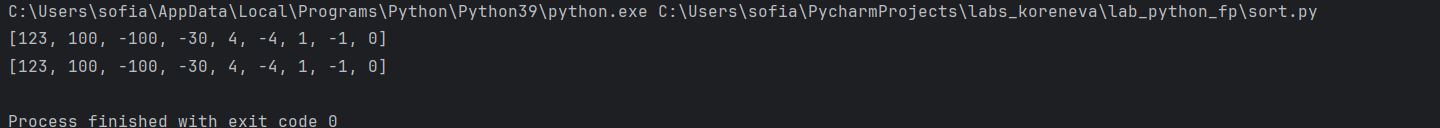
Необходимо решить задачу двумя способами:

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

**Текст программы**

import math  
  
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]  
  
result = sorted(list(x for x in data), key=abs, reverse=True)  
print(result)  
  
result\_with\_lambda = sorted(data, key = lambda x: abs(x), reverse=True)  
print(result\_with\_lambda)

**Примеры выполнения программы**



**Задача 5 (файл print\_result.py)**

**Описание задачи**

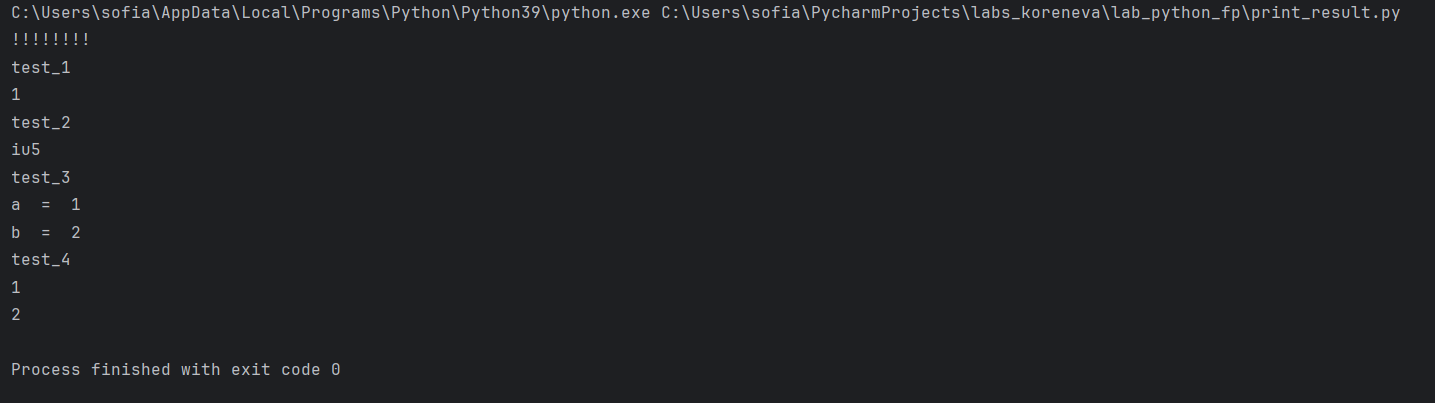
Необходимо реализовать декоратор print\_result, который выводит на экран результат выполнения функции.

* 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):  
 res = func(\*args, \*\*kwargs)  
 print(f"{func.\_\_name\_\_}")  
 if isinstance(res, list):  
 for r in res:  
 print (r)  
 elif isinstance(res, dict):  
 for x, y in res.items():  
 print (x, " = ", y)  
 else:  
 print(res)  
 return res  
 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()

**Примеры выполнения программы**

****

**Задача 6 (файл cm\_timer.py)**

**Описание задачи**

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

with cm\_timer\_1():

sleep(5.5)

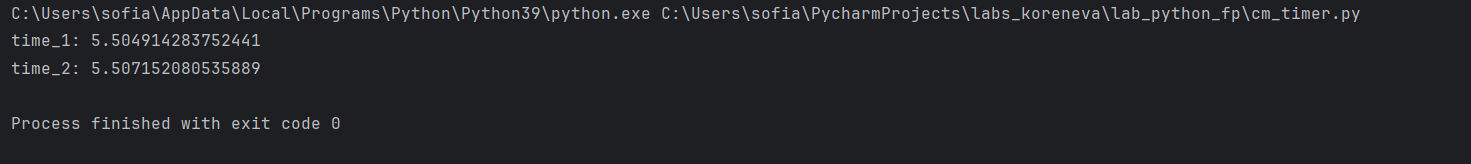
После завершения блока кода в консоль должно вывестись time: 5.5 (реальное время может несколько отличаться).

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()  
   
 def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):  
 elapsed\_time = time.time() - self.start\_time  
 print(f"time\_1: {elapsed\_time}")  
  
  
@contextmanager  
def cm\_timer\_2():  
 start\_time = time.time()  
 yield  
 elapsed\_time = time.time() - start\_time  
 print(f"time\_2: {elapsed\_time}")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 time.sleep(5.5)  
 with cm\_timer\_2():  
 time.sleep(5.5)

**Примеры выполнения программы**



**Задача 7 (файл process\_data.py)**

**Описание задачи**

* 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**

import json  
from gen\_random import gen\_random  
from cm\_timer import cm\_timer\_1  
from print\_result import print\_result  
  
path = "C:/Users/sofia/PycharmProjects/labs\_koreneva/lab\_python\_fp/data\_light.json"  
  
with open(path, encoding='utf-8') as f:  
 data = json.load(f)  
  
@print\_result  
def f1(arg):  
 return sorted(set(item['job-name'].lower() for item in arg))  
  
@print\_result  
def f2(arg):  
 return list(filter(lambda s: s.startswith('программист'), arg))  
  
@print\_result  
def f3(arg):  
 return list(map(lambda s: s + ' с опытом Python', arg))  
  
@print\_result  
def f4(arg):  
 salaries = gen\_random(len(arg), 100000, 2000000)  
 return ['{} зарплата {}'.format(job, salary) for job, salary in zip(arg, salaries)]  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 f4(f3(f2(f1(data))))

**Примеры выполнения программы**

