**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

«Функциональные возможности языка Python»

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

2023 год

**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'

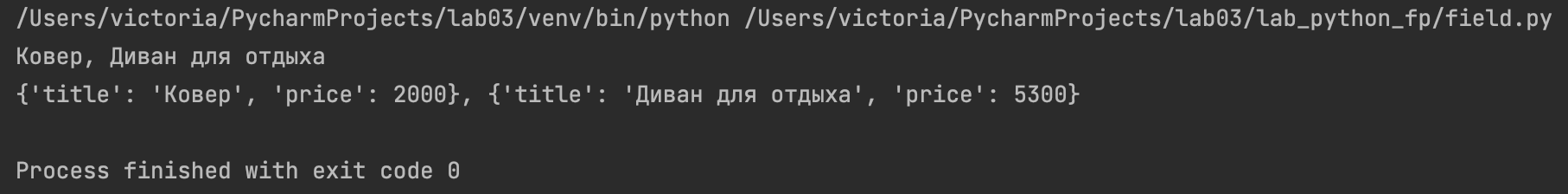
field(goods, 'title', 'price') должен выдавать {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}

* 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**

*""" Задача №1 """*# Пример:  
# goods = [  
# {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
# {'title': 'Диван для отдыха', 'price': 5300, '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', 'price': 5300}  
  
  
def field(items, \*args):  
 *"""* ***:param*** *items: список словарей* ***:param*** *args: неограниченное количество аргументов* ***:returns*** *кортеж или список словарей  
 """* assert len(args) > 0, 'Не передано ни одного аргумента в "args"!'  
 if len(args) == 1:  
 return (item[item\_key] for item in items for item\_key in item if  
 item[item\_key] is not None and item\_key == args[0])  
 else:  
 ans = []  
 for item in items:  
 tmp = {}  
 for arg in args:  
 if not item[arg] is None: tmp[arg] = item[arg]  
 if tmp: ans.append(tmp)  
 return tuple(ans)  
  
  
def mainTask1():  
 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\_\_':  
 mainTask1()

**Примеры выполнения программы**

****

**Задача 2 (файл gen\_random.py)**

**Описание задачи**

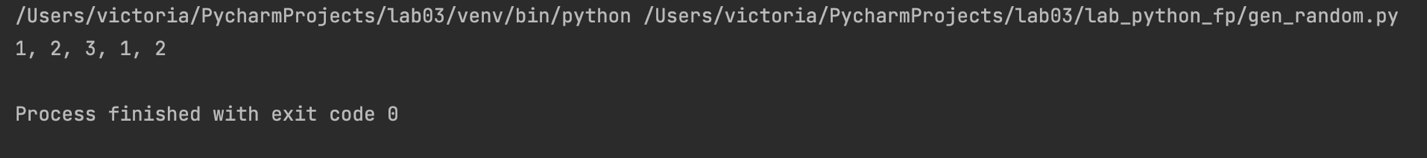
Необходимо реализовать генератор gen\_random(количество, минимум, максимум), который последовательно выдает заданное количество случайных чисел в заданном диапазоне от минимума до максимума, включая границы диапазона. Пример:

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

**Текст программы**

*""" Задача 2 """*from random import randint  
  
  
# Пример:  
# gen\_random(5, 1, 3) должен выдать выдать 5 случайных чисел  
# в диапазоне от 1 до 3, например 2, 2, 3, 2, 1  
# Hint: типовая реализация занимает 2 строки  
  
def gen\_random(num\_count, begin, end):  
 *"""  
 Генерация случайных чисел* ***:param*** *num\_count: количество случайных чисел* ***:param*** *begin: минимальное число* ***:param*** *end: максимальное число* ***:return****: кортеж случайных чисео  
 """* return [randint(begin, end) for \_ in range(num\_count)]  
  
  
def mainTask2():  
 print(\*gen\_random(5, 1, 3), sep=', ')  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 mainTask2()

**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’]

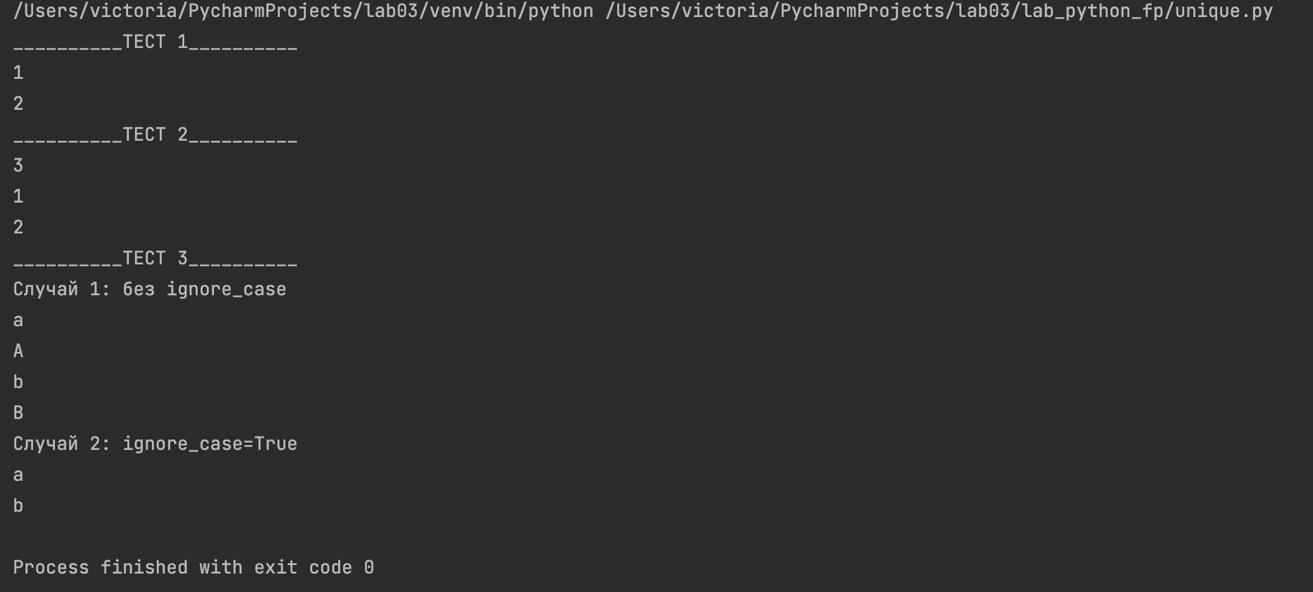
Unique(data) будет последовательно возвращать только a, A, b, B.

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

**Текст программы**

**Текст программы** *""" Задача 3 """*# Нужно реализовать конструктор  
# В качестве ключевого аргумента, конструктор должен принимать bool-параметр ignore\_case,  
# в зависимости от значения которого будут считаться одинаковыми строки в разном регистре  
# Например: ignore\_case = True, Aбв и АБВ - разные строки  
# ignore\_case = False, Aбв и АБВ - одинаковые строки, одна из которых удалится  
# По-умолчанию ignore\_case = False  
from lab\_python\_fp.gen\_random import gen\_random  
  
  
# Итератор для удаления дубликатов  
class Unique(object):  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 self.\_used\_data = set()  
 self.\_data = items  
 self.\_index = 0  
  
 if 'ignore\_case' not in kwargs:  
 self.ignore\_case = False  
 else:  
 self.ignore\_case = kwargs['ignore\_case']  
  
 def \_\_next\_\_(self):  
  
 # Если bool-параметр ignore\_case(игнорирование регистра) истинен,  
 # то все элементы приводяся к нижнему регистру  
 if self.ignore\_case:  
 for counter, el in enumerate(self.\_data):  
 if type(el) is str:  
 self.\_data[counter] = el.lower()  
  
 while True:  
 if self.\_index >= len(self.\_data):  
 raise StopIteration  
 else:  
 current = self.\_data[self.\_index]  
 self.\_index += 1  
 if current not in self.\_used\_data:  
 self.\_used\_data.add(current)  
 return current  
  
 def \_\_iter\_\_(self):  
 return self  
  
  
def mainTask3():  
 print('\_\_\_\_\_\_\_\_\_\_ТЕСТ 1\_\_\_\_\_\_\_\_\_\_')  
 data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
 for item in Unique(data):  
 print(item)  
  
 print('\_\_\_\_\_\_\_\_\_\_ТЕСТ 2\_\_\_\_\_\_\_\_\_\_')  
 data = gen\_random(10, 1, 3)  
 for item in Unique(data):  
 print(item)  
  
 print('\_\_\_\_\_\_\_\_\_\_ТЕСТ 3\_\_\_\_\_\_\_\_\_\_')  
 data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
 print('Случай 1: без ignore\_case')  
 for item in Unique(data):  
 print(item)  
  
 print('Случай 2: ignore\_case=True')  
 for item in Unique(data, ignore\_case=True):  
 print(item)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 mainTask3()

**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]

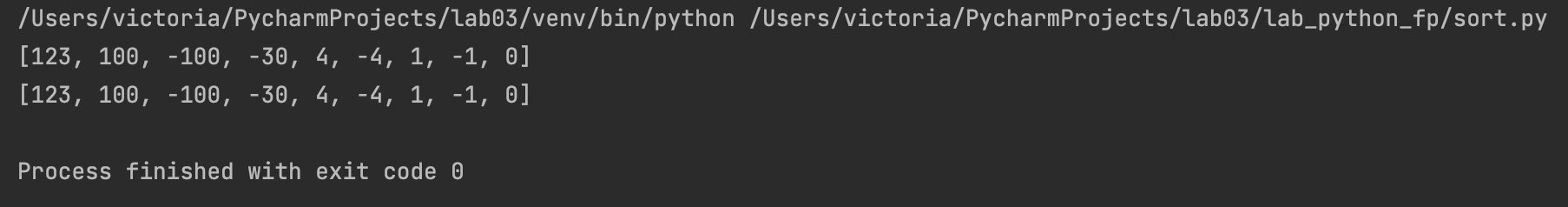
Вывод: [123, 100, -100, -30, 30, 4, -4, 1, -1, 0]

Необходимо решить задачу двумя способами:

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

**Текст программы** *""" Задача 4"""*data = [4, -30, 100, -100, 123, 1, 0, -1, -4]  
  
  
# Сортировка по модулю в порядке убывания  
def mainTask4():  
 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\_\_':  
 mainTask4()

**Примеры выполнения программы**



**Задача 5 (файл print\_result.py)**

**Описание задачи**

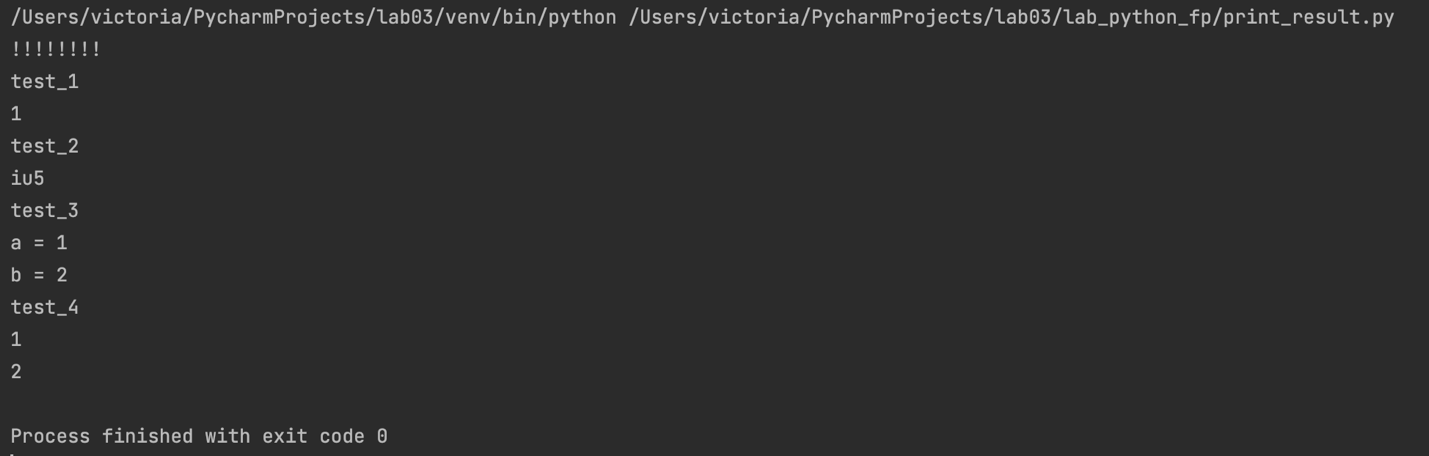
Необходимо реализовать декоратор print\_result, который выводит на экран результат выполнения функции.

* 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**

*""" Задача 5"""*# Реализация декоратора  
def print\_result(func):  
 def wrapper(\*args, \*\*kwargs):  
 # фактические и формальные парамтеры опциональны в данном случае  
 print(func.\_\_name\_\_)  
 original\_result = func(\*args, \*\*kwargs)  
 if isinstance(original\_result, list):  
 for \_ in original\_result:  
 print(\_)  
 return original\_result  
 elif isinstance(original\_result, dict):  
 for key, value in original\_result.items():  
 print('{} = {}'.format(key, value))  
 return original\_result  
 else:  
 print(original\_result)  
 return original\_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]  
  
  
def mainTask5():  
 print('!!!!!!!!')  
 test\_1()  
 test\_2()  
 test\_3()  
 test\_4()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 mainTask5()

**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)

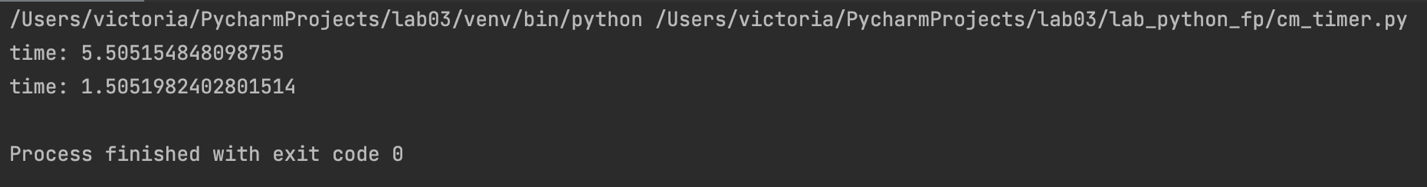
После завершения блока кода в консоль должно вывестись time: 5.5 (реальное время может несколько отличаться).

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**

*""" Задача 6"""*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\_value, exc\_tb):  
 print(f'time: {time.time() - self.start\_time}')  
  
  
@contextmanager  
def cm\_timer\_2():  
 start\_time = time.time()  
 yield  
 end\_time = time.time()  
 print(f'time: {end\_time - start\_time}')  
  
  
def mainTask6():  
 with cm\_timer\_1():  
 time.sleep(5.5)  
  
 with cm\_timer\_2():  
 time.sleep(1.5)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 mainTask6()

**Примеры выполнения программы**



**Задача 7 (файл process\_data.py)**

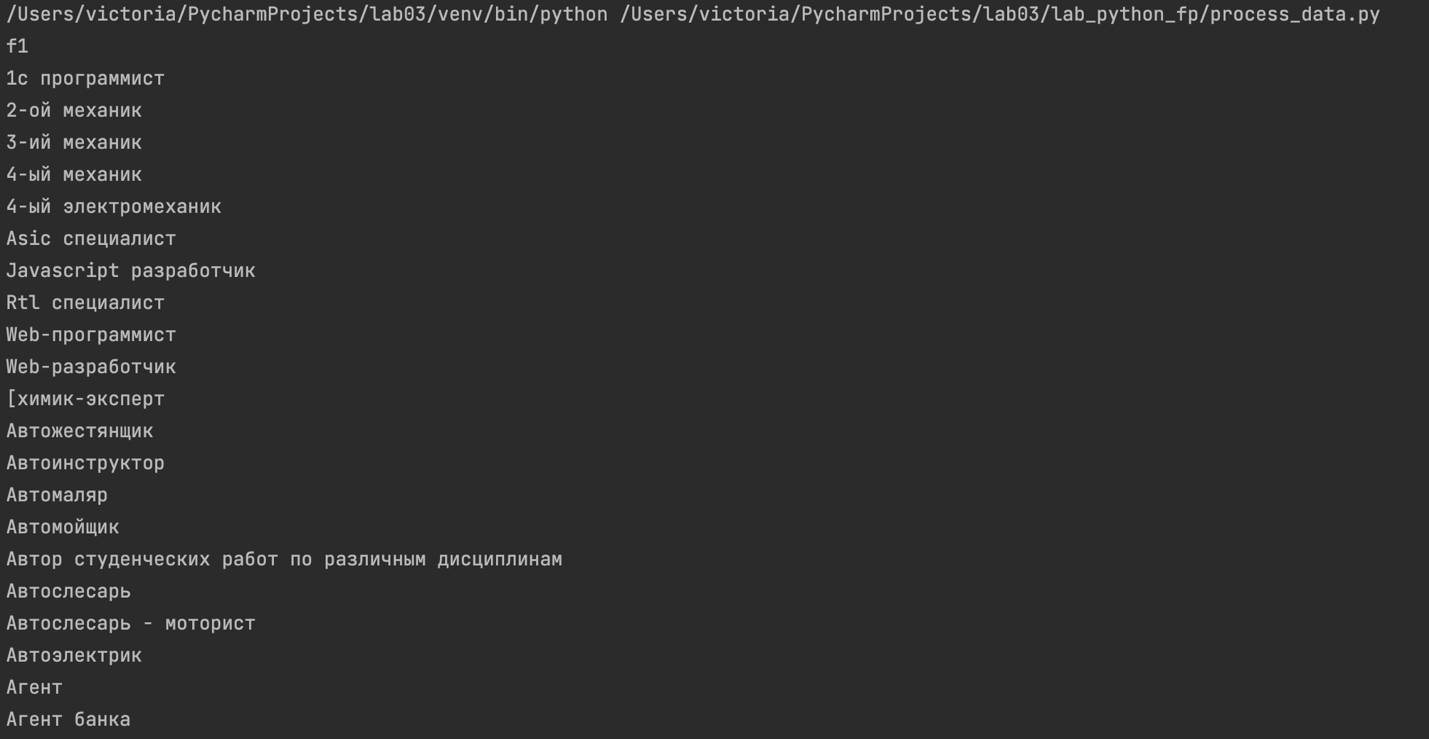
**Описание задачи**

* 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  
  
# Сделаем другие необходимые импорты  
from lab\_python\_fp.print\_result import print\_result  
from lab\_python\_fp.cm\_timer import cm\_timer\_1  
from lab\_python\_fp.gen\_random import gen\_random  
  
# Необходимо в переменную path сохранить путь к файлу, который был передан при запуске сценария  
path = "/Users/victoria/PycharmProjects/lab03/data\_light.json"  
  
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):  
 *"""  
 Выводит отсортированный список профессий без повторений.  
 Регистр игнорируется при сортировке.* ***:param*** *arg: json-файл* ***:return****: отсортированный список  
 """* return sorted(list(set([item['job-name'].capitalize() for item in arg])))  
  
  
@print\_result  
def f2(arg):  
 *"""  
 Фильтрует входной массив и возвращает только те элементы, которые начинаются со слова "программист"* ***:param*** *arg: список* ***:return****: отфильтрованный список  
 """* return list(filter(lambda text: (text.split())[0] == 'Программист', arg))  
  
  
@print\_result  
def f3(arg):  
 *"""  
 Модифицирует каждый элемент списка посредством добавления к нему строки “с опытом Python”* ***:param*** *arg: список* ***:return****: модифицированный список  
 """* return list(map(lambda text: text + ' с опытом Python', arg))  
  
  
@print\_result  
def f4(arg):  
 *"""  
 Генерирует для каждой специальности зарплату от 100 000 до 200 000 рублей и присоединяет её к названию специальности.  
 Пример: "Программист C# с опытом Python, зарплата 137287 руб"* ***:param*** *arg: список* ***:return****:список  
 """* tmp = list(zip(arg, [', зарплата ' + str(salary) + ' руб.' for salary in gen\_random(len(arg), 100000, 200000)]))  
 return [item[0] + item[1] for item in tmp]  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with cm\_timer\_1():  
 f4(f3(f2(f1(data))))

**Примеры выполнения программы**

****

