**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Программирование на основе классов и шаблонов»

Отчет по лабораторным работам №3-4

«Функциональные возможности языка Python»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Выполнил: | |  | | Проверил: | |
|  | студент группы ИУ5-31Б | |  | | преподаватель каф. ИУ5 | |
|  | Хомутинников Александр | |  | |  | |
|  | Подпись и дата: | |  | | Подпись и дата: | |
|  | |  | |  | |  |

Москва, 2023 г.

**Задача 1.**

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'}

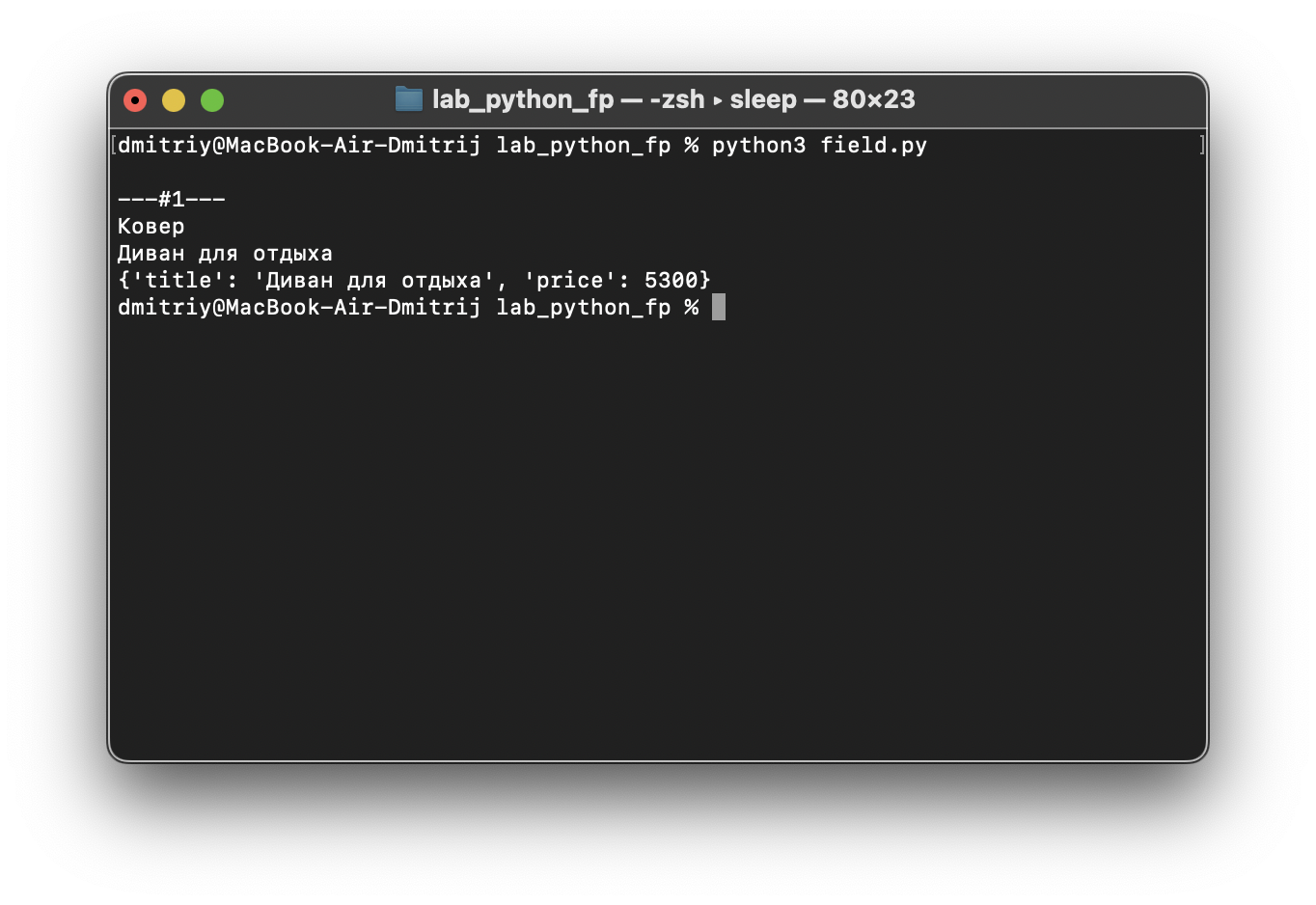
- В качестве первого аргумента генератор принимает список словарей, дальше через \*args генератор принимает неограниченное количествово аргументов.

- 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.

**Field.py**  
*""" Task #1. """  
  
  
def* field(items, \*args):  
 *assert* len(args) > 0, 'The parameter "args" is empty!'  
 *if* len(args) == 1:  
 *return* (item[el] *for* item *in* items *for* el *in* item *if* el == args[0] *and* item[el] *is not None*)  
 *else*:  
 *return* {el: item[el] *for* item *in* items *for* el *in* item *for* argument *in* args *if* el == argument *and* item[argument] *is not None*}  
  
  
*def* main1():  
 print('\n---#1---')  
  
 goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}  
 ]  
 # должен выдавать 'Ковер', 'Диван для отдыха'.  
 res = (field(goods, 'title'))  
 *for* el *in* res:  
 print(el)  
 # должен выдавать {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}.  
 print(field(goods, 'title', 'price'))  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 main1()

**Результат:**



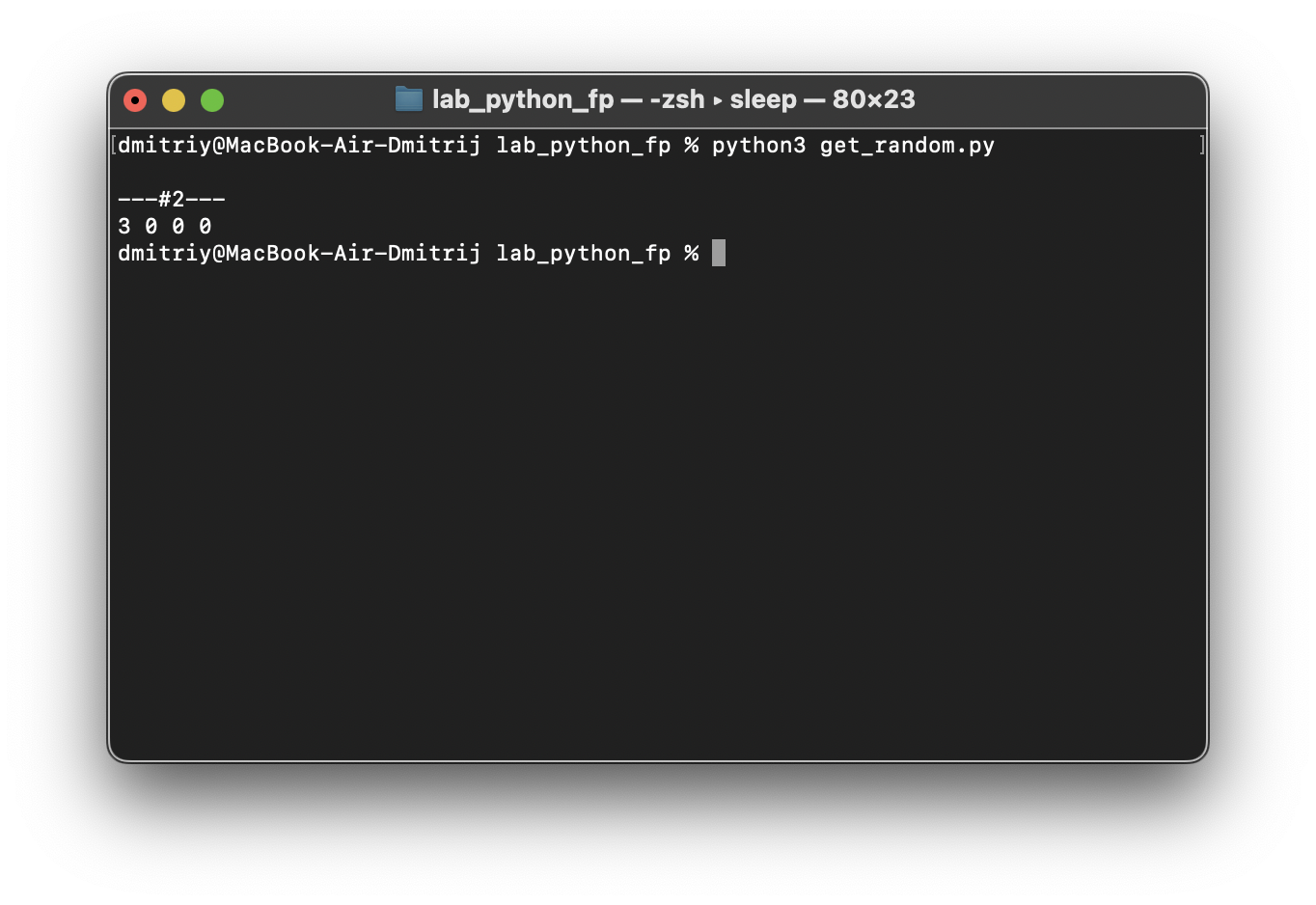
**Задача 2.**

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

**Get\_random.py**  
*""" Task #2. """  
from* random *import* randint  
  
  
*def* get\_random(num\_count, begin, end):  
 *"""  
 Рандомные числа* ***:param*** *num\_count: число случайных чисел* ***:param*** *begin: с какого числа* ***:param*** *end: по какое* ***:return****: картеж чисел  
 """  
 return* (randint(begin, end) *for* \_ *in* range(num\_count))  
  
  
*def* main2():  
 print('\n---#2---')  
 numbers = get\_random(4, 0, 5)  
 *for* el *in* numbers:  
 print(el, end=' ')  
 print()  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 main2()

**Результат:**

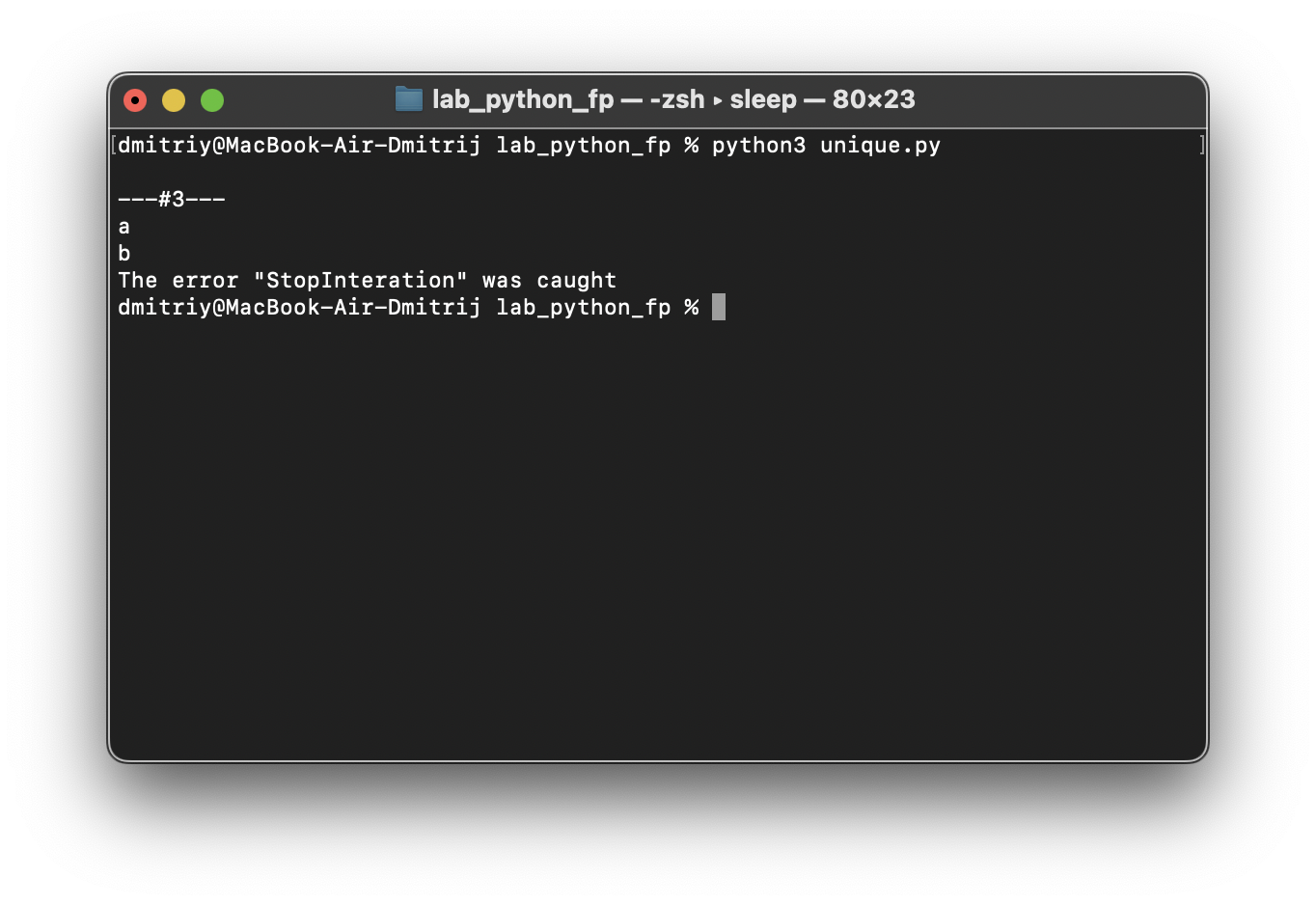


**Задача 3.**

* 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.

**Unique.py**  
*""" Task #3. """*# Итератор для удаления дубликатов  
*class* Unique(object):  
 *def* \_\_init\_\_(self, items, ignore\_case=*False*, \*\*kwargs):  
 self.\_data = items  
 self.\_ignore\_case = ignore\_case  
 self.\_\_used\_data = set()  
 self.\_\_index = 0  
  
 *def* \_\_next\_\_(self):  
 # Если игнорируем капс, то пробегаемся по списку и приводим всё к общему капсу.  
 *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  
 # если текущего числа ещё не было, добавляем и возвращаем.  
 *if* current *not in* self.\_\_used\_data:  
 self.\_\_used\_data.add(current)  
 *return* current  
  
 *def* \_\_iter\_\_(self):  
 *return* self  
  
  
*def* main3():  
 print('\n---#3---')  
 data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']  
 it = Unique(data, ignore\_case=*True*)  
 *try*:  
 *while True*:  
 print(it.\_\_next\_\_())  
 *except* StopIteration:  
 print('The error "StopInteration" was caught')  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 main3()

**Результат:**

****

**Задача 4.**

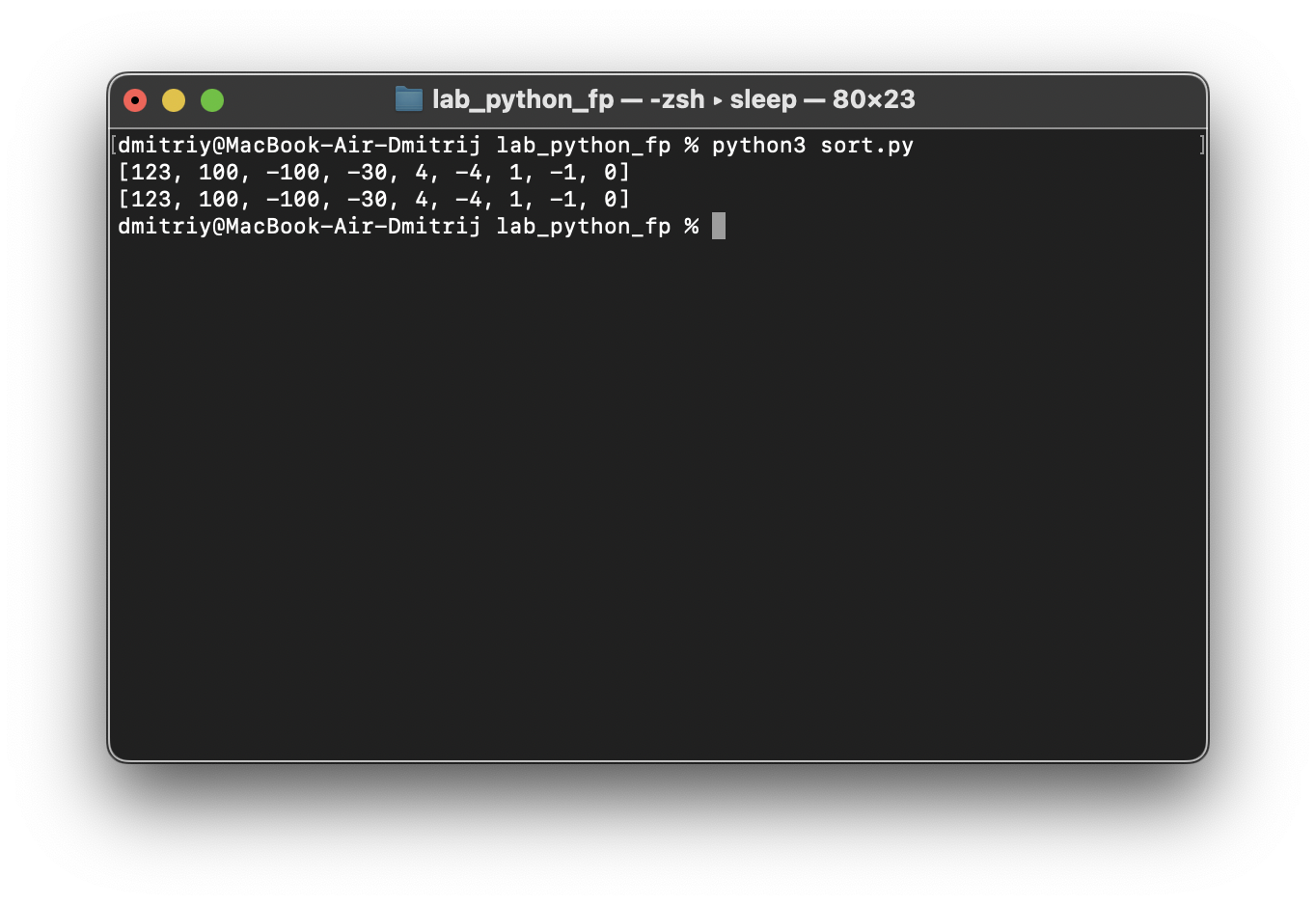
Дан массив 1, содержащий положительные и отрицательные числа. Необходимо **одной строкой кода** вывести на экран массив 2, которые содержит значения массива 1, отсортированные по модулю в порядке убывания. Сортировку необходимо осуществлять с помощью функции sorted.

Необходимо решить задачу двумя способами:

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

**Sort.py**  
*""" Task #4 """  
  
  
def* main4():  
 data = [4, -30, 100, -100, 123, 1, 0, -1, -4]  
 result = sorted(data, key=abs, reverse=*True*)  
 print(result)  
  
 result\_with\_lambda = sorted(data, key=*lambda* a: abs(a), reverse=*True*)  
 print(result\_with\_lambda)  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 main4()

**Результат:**



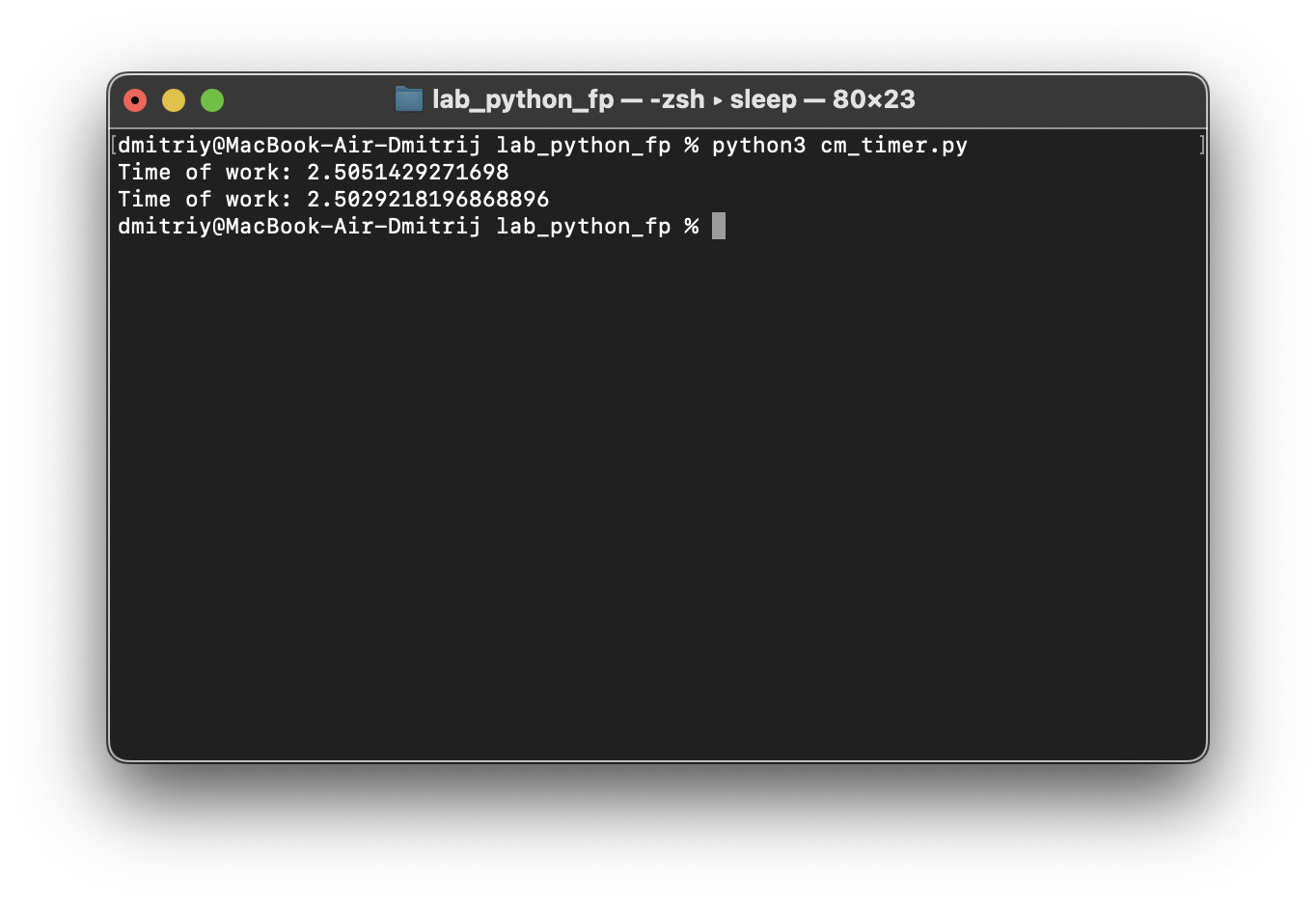
**Задача 5.**

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.

**Cm\_timer.py**  
*""" Task #6 """  
from* time *import* sleep, time  
*from* contextlib *import* contextmanager  
  
  
*class* cm\_timer\_1:  
 *def* \_\_init\_\_(self):  
 self.\_\_start = 0  
 self.\_\_finish = 0  
  
 *def* \_\_enter\_\_(self):  
 self.\_\_start = time()  
 # return self.\_\_start  
  
 *def* \_\_exit\_\_(self, exp\_type, exp\_value, traceback):  
 self.\_\_finish = time()  
 print(f'Time of work: {self.\_\_finish - self.\_\_start}')  
  
  
@contextmanager  
*def* cm\_timer\_2():  
 st = time()  
 *yield None* en = time()  
 print(f'Time of work: {en - st}')  
  
  
*def* main6():  
 *with* cm\_timer\_1():  
 sleep(2.5)  
  
 *with* cm\_timer\_2():  
 sleep(2.5)  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 main6()

**Результат:**

****

**Задача 6.**

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():

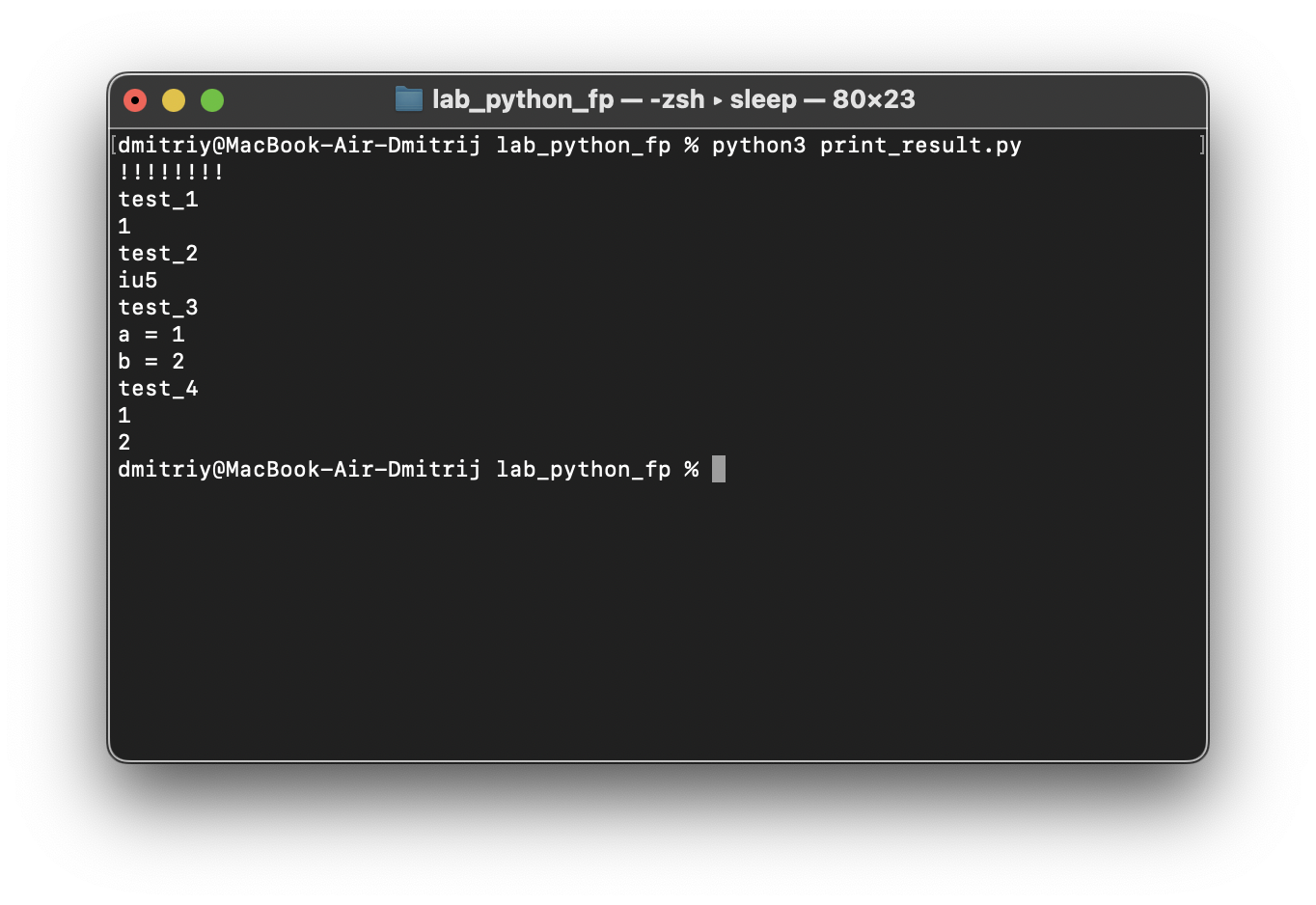
sleep(5.5)

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).

**Print\_result.py**  
*""" Task #5 """  
  
  
def* print\_result(func):  
 *def* wrapper(lst=[], \*args, \*\*kwargs):  
 print(func.\_\_name\_\_)  
  
 *if* len(lst) == 0:  
 result = func(\*args, \*\*kwargs)  
 *else*:  
 result = func(lst, \*args, \*\*kwargs)  
  
 *if* type(result) *is* dict:  
 *for* key, el *in* result.items():  
 print(f'{key} = {el}')  
  
 *elif* type(result) *is* list:  
 print('\n'.join(map(str, result)))  
  
 *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]  
  
  
*def* main5():  
 print('!!!!!!!!')  
 test\_1()  
 test\_2()  
 test\_3()  
 test\_4()  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 main5()

**Результат:**

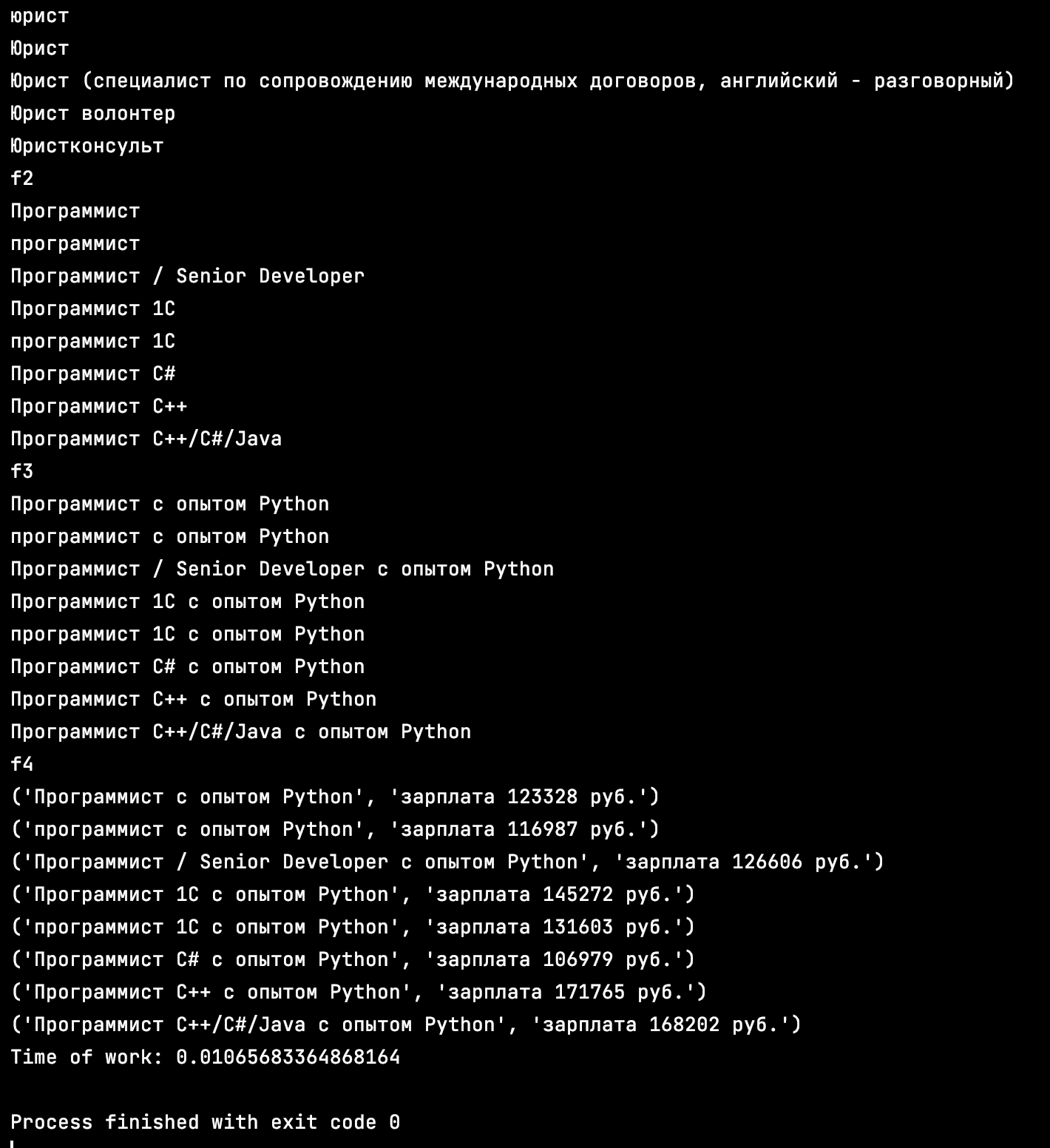


**Задача 7.**

* 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.

**process\_data.py**  
*from* Code.lab\_python\_fp.print\_result *import* print\_result  
*from* Code.lab\_python\_fp.cm\_timer *import* cm\_timer\_1  
*from* Code.lab\_python\_fp.get\_random *import* get\_random  
*import* json  
*import* sys  
  
*try*:  
 path = sys.argv[1]  
 print(path)  
*except*:  
 path = '../file.json'  
  
*with* open(path) *as* f:  
 data = json.load(f)  
  
  
@print\_result  
*def* f1(arg) -> list:  
 *"""  
 Выводит отсортированный список профессий без повторений.  
 Сортировка должна игнорировать регистр.* ***:param*** *arg: список* ***:return****: отсортированный список  
 """  
 return* sorted(list(set([el['job-name'] *for* el *in* arg])), key=*lambda* x: x.lower())  
  
  
@print\_result  
*def* f2(arg) -> list:  
 *"""  
 Фильтрует входной массив и возвращает только те элементы, которые начинаются со слова “программист”* ***:param*** *arg:* ***:return****:  
 """  
 return* list(filter(*lambda* text: (text.split())[0].lower() == 'программист', arg))  
  
  
@print\_result  
*def* f3(arg) -> list:  
 *"""  
 Функция f3 должна модифицировать каждый элемент массива, добавив строку “с опытом Python”* ***:param*** *arg:* ***:return****:  
 """  
 return* list(map(*lambda* lst: lst + ' с опытом Python', arg))  
  
  
@print\_result  
*def* f4(arg) -> list:  
 *"""  
 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.* ***:param*** *arg:* ***:return****:  
 """  
 return* list(zip(arg, ['зарплата ' + str(el) + ' руб.' *for* el *in* get\_random(len(arg), 100000, 200000)]))  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 *with* cm\_timer\_1():  
 f4(f3(f2(f1(data))))

**Результат:**



**Дополнительное задание:**

*import* numpy  
  
  
*def* print\_result(arr):  
 *for* el *in* arr:  
 print(el)  
 print('---')  
  
  
*def* main():  
 lst = [3, 4, 5, 6, 7]  
  
 res1 = zip(lst, map(*lambda* num: num \*\* 2, lst))  
 res2 = [(el, el \*\* 2) *for* el *in* lst]  
 res3 = zip(lst, numpy.array(lst) \*\* 2)  
  
 print\_result(res1)  
 print\_result(res2)  
 print\_result(res3)  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 main()

**Результаты:**

