**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет “Информатика и системы управления”

Кафедра ИУ-5 “Системы обработки информации и управления”

Курс “Разработка интернет-приложений”

Отчет по лабораторной работе №3

“Функциональные возможности языка Python”

Выполнила:

Студентка группы ИУ5-53Б

Елхимова Ирина

**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 раполагаться 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.

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

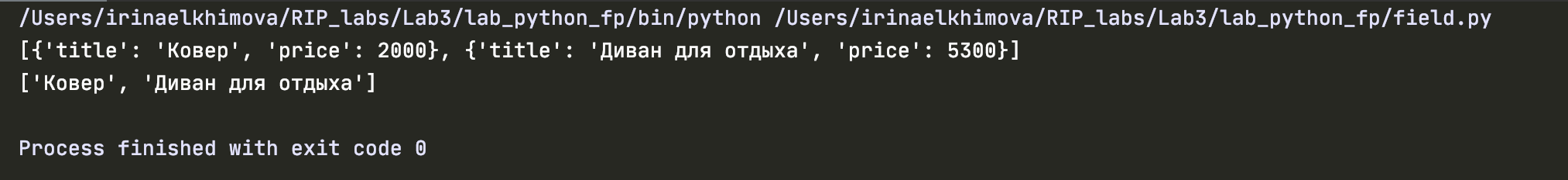
field(goods, 'title', 'price') должен выдавать {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}

* 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:**

#!/Users/irinaelkhimova/RIP\_labs/Lab3/lab\_python\_fp/bin/python  
# -\*- coding: utf-8 -\*-  
  
  
  
goods = [  
 {'title': "Ковер", 'price': 2000, 'color': "green"},  
 {'title': "Диван для отдыха", 'price': 5300, 'color': "black"}  
]  
  
  
*def* field(*items*, *\*args*):  
 *assert* len(*args*) > 0  
 *if* len(*args*) == 1:  
 *for* item *in items*:  
 *for* arg *in args*:  
 *if* arg *in* item:  
 *yield* item[arg]  
 *else*:  
 *for* item *in items*:  
 new\_item = {}  
 *for* arg *in args*:  
 *if* arg *in* item:  
 new\_item[arg] = item[arg]  
 *if* len(new\_item.keys()) > 0:  
 *yield* new\_item  
print(list(field(goods, 'title', 'price')))  
print(list(field(goods, 'title')))

**Экранные формы:**

****

**Задача 2 (файл gen\_random.py)**

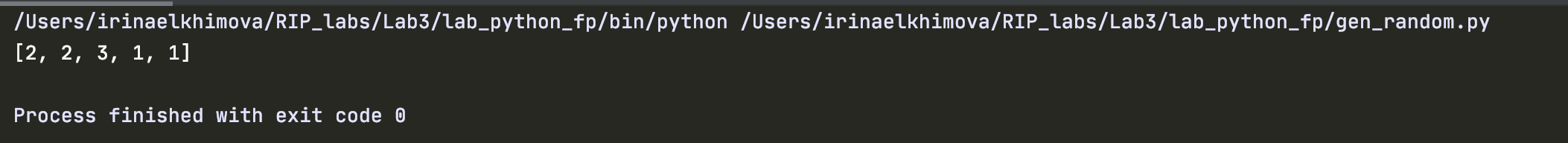
Необходимо реализовать генератор gen\_random(количество, минимум, максимум), который последовательно выдает заданное количество случайных чисел в заданном диапазоне от минимума до максимума, включая границы диапазона. Пример:

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

**Текст программы:**

#!/Users/irinaelkhimova/RIP\_labs/Lab3/lab\_python\_fp/bin/python  
# -\*- coding: utf-8 -\*-  
  
*import* random  
  
  
*def* gen\_random(*num\_count*, *begin*, *end*):  
 *for* \_ *in* range(*num\_count*):  
 *yield* random.randint(*begin*, *end*)  
  
  
  
print(list(gen\_random(5, 1, 3)))

**Экранные формы:**

****

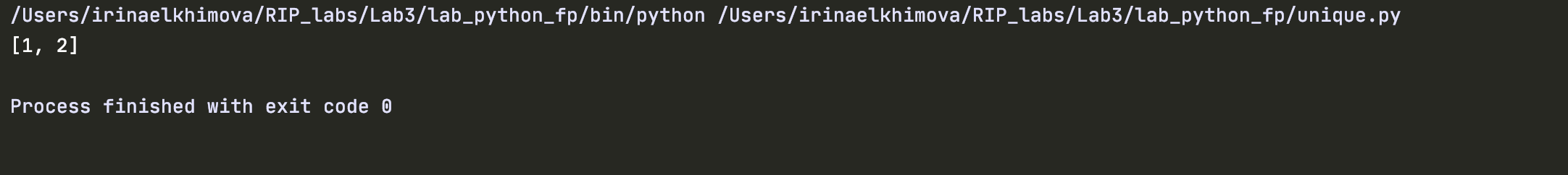
**Задача 3 (файл unique.py)**

* 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.unique\_items = []  
 self.items = iter(*items*)  
 *if* 'ignore\_case' *not in kwargs*:  
 self.ignore\_case = *False  
 else*:  
 self.ignore\_case = *kwargs*['ignore\_case']  
  
 *def \_\_next\_\_*(self):  
 # Нужно реализовать \_\_next\_\_  
  
 *while True*:  
 item = self.items.\_\_next\_\_()  
 compare\_item = *None  
 if* self.ignore\_case *and* type(item) *is* str:  
 compare\_item = item.lower()  
 *else*:  
 compare\_item = item  
 *if* compare\_item *not in* self.unique\_items:  
 self.unique\_items.append(compare\_item)  
 *return* item  
  
 *def \_\_iter\_\_*(self):  
 *return* self  
  
data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
print(list(Unique(data)))

**Экранные формы:**

****

**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.

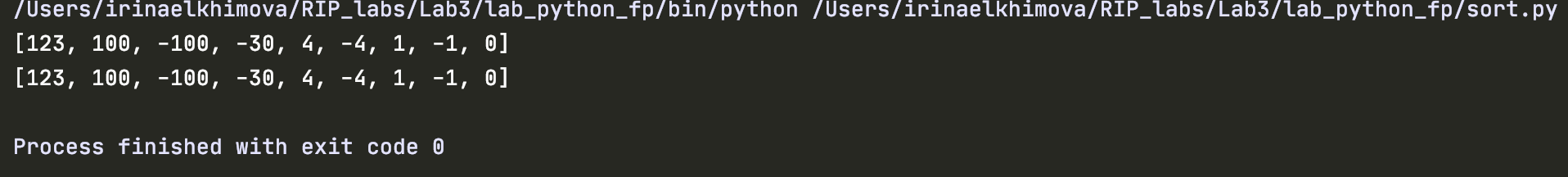
Необходимо решить задачу двумя способами:

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

**Текст программы:**

*from* math *import* fabs  
  
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 result = sorted(data, key=fabs, reverse=*True*)  
 print(result)  
  
 result\_with\_lambda = sorted(data, key=*lambda* i: fabs(i), reverse=*True*)  
 print(result\_with\_lambda)

**Экранные формы:**

****

**Задача 5 (файл print\_result.py)**

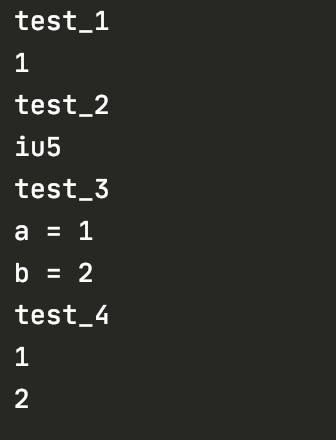
Необходимо реализовать декоратор print\_result, который выводит на экран результат выполнения функции.

* 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:**

#!/Users/irinaelkhimova/RIP\_labs/Lab3/lab\_python\_fp/bin/python  
# -\*- coding: utf-8 -\*-  
  
*def* print\_result(*func*, \*arg):  
 *def* decor(*\*arg*):  
 print(func.\_\_name\_\_)  
 result = func(\**arg*)  
 *if* type(result) *is* list:  
 *for* item *in* result:  
 print(item)  
 *elif* type(result) *is* dict:  
 *for* key, value *in* result.items():  
 print(str(key) + ' = ' + str(value))  
 *else*:  
 print(result)  
 *return* result  
 *return* decor  
  
  
@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()

**Экранные формы:**



**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.

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.

**Текст программы:**

*import* time  
  
  
*class* cm\_timer\_1:  
  
 *def \_\_enter\_\_*(self):  
 self.time = time.time()  
  
 *def \_\_exit\_\_*(self, *value*, *key*, *traceback*):  
 print(time.time()-self.time)  
  
  
*class* cm\_timer\_2:  
 *def \_\_init\_\_*(self):  
 self.\_start\_time = *None  
  
 def \_\_enter\_\_*(self):  
 self.\_start\_time = time.perf\_counter()  
  
 *def \_\_exit\_\_*(self, *value*, *key*, *traceback*):  
 elapsed\_time = time.perf\_counter() - self.\_start\_time  
 self.\_start\_time = *None* print(f"Elapsed time: {elapsed\_time:0.4f} seconds")

**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](https://github.com/iu5team/iu5web-fall-2020/tree/master/labs/lab3/data/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.