Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ	**«Информатики и систем управления»**
КАФЕДРА	Системы обработки информации и управления

Дисциплина «Разработка Интернет-Приложений»

ЛАБОРАТОРНАЯ РАБОТА №3

Функциональные возможности языка Python

Студент	Сахарова Е. К. ИУ5-52Б
Преподаватель	Гапанюк Ю. Е.

Цель лабораторной работы: изучение возможностей функционального программирования в языке Python.

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 располагаться 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. 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 количество 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

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

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.

Пример:

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

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

data = gen_random(1, 3, 10)

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.

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. 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.

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.

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. 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.

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.

Код программы:

field.py

```
def field(items, *args):
    assert len(args) > 0
    if len(args) == 1:
        return_list = [item[key] for item in items for key in item if key in args]
        #print(*[item[key] for item in items for key in item if key in args], sep=',
')
    else:
        return_list = []
        for item in items:
            return_list.append({key: item[key] for key in args})
        #print(return_list)
    return return_list


if __name__ == '__main__':
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}
     ]

    print(*field(goods, 'title'), sep=', ')  # должен выдавать 'Ковер', 'Диван для
отдыха'
    print(*field(goods, 'title', 'price'), sep=', ')
    # должен выдавать {'title': 'Ковер', 'price': 2000}, {'title': 'Диван для
отдыха', 'price': 5300}
```

gen_random.py

```
import random


# Hint: типовая реализация занимает 2 строки
def gen_random(num_count, begin, end):
    list_ans = [random.randint(begin, end) for _ in range(num_count)]
    # print(*list_ans, sep=', ')
    return list_ans


if __name__ == '__main__':
```

```
    print(*gen_random(5, 1, 3), sep=', ')  # должен выдать выдать 5 случайных чисел
    # в диапазоне от 1 до 3, например 2, 2, 3, 2, 1
```

unique.py

```
from github.lab3.lab_python_fp.gen_random import gen_random


class Unique:
    """Итератор, оставляющий только уникальные значения."""
    def __init__(self, items, **kwargs):
        self.used_elements = set()
        self.items = items
        self.index = 0
        self.ignore_case = [kwargs[key] for key in kwargs] if len(kwargs) > 0 and
kwargs['ignore_case'] is True \
            else False

    def __iter__(self):
        return self

    def __next__(self):
        while True:
            if self.index >= len(self.items):
                raise StopIteration
            else:
                #if self.ignore_case:
                    #self.items = list(map(lambda _: str(_).lower(), self.items))
                if self.ignore_case:
                    current = str(self.items[self.index]).lower()
                else:
                    current = self.items[self.index]
                self.index += 1
                if current not in self.used_elements:
                    # Добавление в множество производится
                    # с помощью метода add
                    self.used_elements.add(current)
                    return current


if __name__ == '__main__':
    print('Task 1')
    data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    for item in Unique(data):
        print(item, end=' ')

    print('\nTask 2')
    data = gen_random(5, 3, 10)
    print(*data, sep=', ')
    for item in Unique(data):
        print(item, end=' ')

    print('\nTask 3')
    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    for item in Unique(data):
        print(item, end=' ')

    print('\nTask 4')
    for item in Unique(data, ignore_case=True):
        print(item, end=' ')
```

sort.py

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

if __name__ == '__main__':
    result = sorted(data, reverse=True, key=abs)
    print(result)

    result_with_lambda = sorted(data, reverse=True, key=lambda _: abs(_))
    print(result_with_lambda)
```

print_result.py

```
def print_result(func_to_decorate):

    def wrapped_func(*args, **kwargs):
        print(func_to_decorate.__name__)
        result = func_to_decorate(*args, **kwargs)
        if type(result) == dict:
            for key in result:
                # print('{} = {}'.format(key, result[key]))
                print('{}, зарплата {} руб.'.format(key, result[key]))
        elif type(result) == list:
            for value in result:
                print(value)
        else:
            print(result)
        return result

    return wrapped_func


@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()
```

cm_timer.py

```
import time
from contextlib import contextmanager


class cm_timer1:
    def __init__(self):
        self.start = time.time()

    def __enter__(self):
        return

    def __exit__(self, *args):
        print('Прошло времени: ', time.time() - self.start)


@contextmanager
def cm_timer2():
    start = time.perf_counter()
    yield lambda: time.perf_counter() - start


if __name__ == '__main__':
    with cm_timer1():
        time.sleep(2)
    with cm_timer2() as t:
        time.sleep(2)
    print('Прошло времени: ', t())
```

process_data.py

```
import json
from github.lab3.lab_python_fp.cm_timer import cm_timer1
from github.lab3.lab_python_fp.field import field
from github.lab3.lab_python_fp.gen_random import gen_random
from github.lab3.lab_python_fp.print_result import print_result
from github.lab3.lab_python_fp.unique import Unique

path = 'data_light.json'

with open(path, 'r', encoding='utf8') as f:
    data = json.load(f)


@print_result
def f1(arg):
    return sorted(Unique(field(arg, 'job-name'), ignore_case=True), key=lambda _:
str(_).lower())


@print_result
def f2(arg):
    return list(filter(lambda _: str(_).startswith('программист'), arg))


@print_result
def f3(arg):
    return list(map(lambda _: str(_).title() + ' с опытом Python', arg))
```

```python
@print_result
def f4(arg):
    return dict(zip(arg, gen_random(len(arg), 100000, 200000)))


if __name__ == '__main__':
    with cm_timer1():
        f4(f3(f2(f1(data))))
```

Экранные формы с примерами выполнения программы:

```
C:\Programs\Python\python.exe C:/Users/sakha/Downloads/Универ/RIP/github/lab3/lab_python_fp/field.py
Ковер, Диван для отдыха
{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}

Process finished with exit code 0
```

```
C:\Programs\Python\python.exe C:/Users/sakha/Downloads/Универ/RIP/github/lab3/lab_python_fp/gen_random.py
3, 3, 2, 2, 1

Process finished with exit code 0
```

```
C:\Programs\Python\python.exe C:/Users/sakha/Downloads/Универ/RIP/github/lab3/lab_python_fp/unique.py
Task 1
1 2
Task 2
4, 5, 3, 9, 4
4 5 3 9
Task 3
a A b B
Task 4
a b
Process finished with exit code 0
```

```
C:\Programs\Python\python.exe C:/Users/sakha/Downloads/Универ/RIP/github/lab3/lab_python_fp/sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]

Process finished with exit code 0
```

```
C:\Programs\Python\python.exe C:/Users/sakha/Downloads/Универ/RIP/github/lab3/lab_python_fp/print_result.py
!!!!!!!!
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2

Process finished with exit code 0
```

```
C:\Programs\Python\python.exe C:/Users/sakha/Downloads/Универ/RIP/github/lab3/lab_python_fp/cm_timer.py
Прошло времени:  2.0003929138183594
Прошло времени:  1.9999820000000001

Process finished with exit code 0
Программист/ Технический Специалист с опытом Python
Программистр-Разработчик  Информационных Систем с опытом Python
f4
Программист с опытом Python, зарплата 154794 руб.
Программист / Senior Developer с опытом Python, зарплата 155097 руб.
Программист 1С с опытом Python, зарплата 144634 руб.
Программист C# с опытом Python, зарплата 152930 руб.
Программист C++ с опытом Python, зарплата 120880 руб.
Программист C++/C#/Java с опытом Python, зарплата 164047 руб.
Программист/ Junior Developer с опытом Python, зарплата 118099 руб.
Программист/ Технический Специалист с опытом Python, зарплата 142672 руб.
Программистр-Разработчик  Информационных Систем с опытом Python, зарплата 168707 руб.
Прошло времени:  0.07328653335571289

Process finished with exit code 0
```