

**Московский государственный технический  
университет им. Н.Э. Баумана**

**Факультет «Информатика и системы управления»  
Кафедра ИУ5 «Системы обработки информации и управления»**

**Курс «Базовые компоненты интернет-технологий»**

**Лабораторная работа №3**

Выполнил:

студент группы ИУ5-34Б  
Гордеев Никита

Подпись и дата:

Проверил:

Гапанюк Ю. Е.

Подпись и дата:

Москва, 2021 г.

## Постановка задачи

### 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.

Шаблон для реализации генератора:

```
# 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', 'pri
```

```
def field(items, *args):
    assert len(args) > 0
    # Необходимо реализовать генератор
```

## 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`

Шаблон для реализации генератора:

```
# Пример:
# gen_random(5, 1, 3) должен 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):
    pass
    # Необходимо реализовать генератор
```

## 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.

Шаблон для реализации класса-итератора:

```
# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, **kwargs):
        # 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, Абв и АВВ - одинаковые строки, одна из которых удалится
        # По-умолчанию ignore_case = False
        pass

    def __next__(self):
        # Нужно реализовать __next__
        pass

    def __iter__(self):
        return self
```

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.

Шаблон реализации:

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

if __name__ == '__main__':
    result = ...
    print(result)

    result_with_lambda = ...
    print(result_with_lambda)
```

## 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 выводиться в столбик через знак равенства.

Шаблон реализации:

```
# Здесь должна быть реализация декоратора

@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()

```

результат выполнения:

```

test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2

```

## 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 (на основе класса и с использованием библиотеки `contextlib`).

## 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.

Шаблон реализации:

```
import json
import sys
# Сделаем другие необходимые импорты

path = None

# Необходимо в переменную path сохранить путь к файлу, который был передан при запуске сценария

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):
    raise NotImplemented

@print_result
def f2(arg):
    raise NotImplemented

@print_result
def f3(arg):
    raise NotImplemented

@print_result
def f4(arg):
    raise NotImplemented

if __name__ == '__main__':
    with cm_timer_1():
        f4(f3(f2(f1(data))))
```

**Текст программы**



Файлы: lab\_3.py и unique.py

```
field.py  cm_timer.py  sort.py  gen_random.py  print_result.py  lab_3.py  unique.py

# f3
import json
import sys
from lab_python_fp.field import field
from lab_python_fp.gen_random import gen_random
from lab_python_fp.unique import Unique
from lab_python_fp.print_result import print_result
from lab_python_fp.cm_timer import cm_timer_1

# Сделаем другие необходимые импорты
path = r"C:\Users\nagor\Documents\python\lab_3\data_light.json"

# Необходимо в переменную path сохранить путь к файлу, который был передан при
# запуске сценария
with open(path, 'r') 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):
    return sorted(list(Unique(field(arg, "job-name"))), key=str.lower)
    #return list(Unique(sorted(list(field(arg, "job-name")), key=str.lower)))
    #return list(Unique(field(arg, "job-name")))

@print_result
def f2(arg):
    return list(filter(lambda x: x.split()[0].lower() == "программист", arg))

@print_result
def f3(arg):
    return list(map(lambda x: x + " с опытом Python", arg))

@print_result
def f4(arg):
    out = list(zip(arg, list(gen_random(len(arg), 100000, 200000))))
    return list(map(lambda x: x[0] + ", зарплата " + str(x[1]) + " py6.", out))

if __name__ == '__main__':
    with cm_timer_1():
        f4(f3(f2(f1(data))))
```

```
# Итератор для удаления дубликатов
class Unique(object):
    def __init__(self, items, **kwargs):
        self.data = set()
        self.iter = iter(items)
        self.ignore_case = kwargs.get('ignore_case', True)
        self.lower_set = set()

    def __next__(self):
        current = None
        while True:
            current = str(next(self.iter))
            if not self.ignore_case and current not in self.data:
                self.data.add(current)
                return current
            elif self.ignore_case and current.lower() not in self.lower_set:
                self.data.add(current)
                self.lower_set.add(current.lower())
                return current

    def __iter__(self):
        return self

if __name__ == '__main__':
    b = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    alist = [1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2]

    for i in Unique(b): print(i)
    print()
    for i in Unique(b, ignore_case=False): print(i)
```

Файл: print\_result.py и gen\_random.py

```
field.py  cm_timer.py  sort.py  print_result.py  lab_3.py  unique.py  gen_random.py

def print_result(func):
    def wrapper(self=None):
        print(func.__name__)
        if self: self = func(self)
        else: self = func()
        if isinstance(self, dict):
            for key, val in self.items():
                print(key, "-", val, end='\n')
        elif isinstance(self, list):
            print(*self, sep='\n')
        else:
            print(self)
        return self
    return wrapper

@print_result
def test_1():
    return 1

@print_result
def test_2():
    return 'ius'

@print_result
def test_3():
    return {'a': 1, 'b': 2}

@print_result
def test_4():
    return [1, 2]

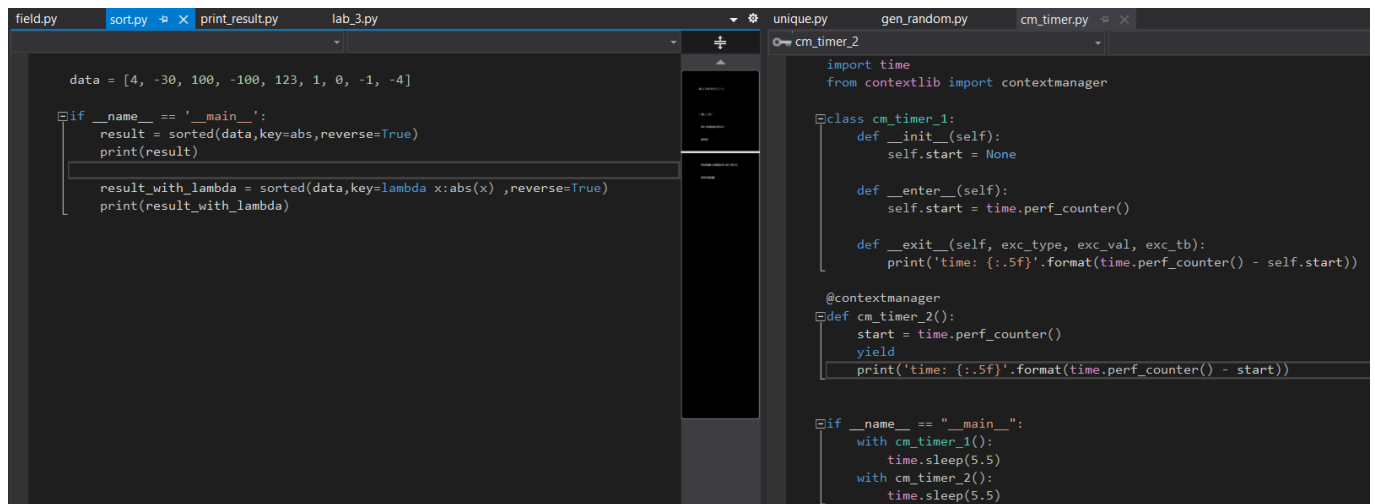
if __name__ == '__main__':
    print('!!!!!!!')
    test_1()
    test_2()
    test_3()
    test_4()
```

```
# Пример:
# gen_random(5, 1, 3) должен выдать 5 случайных чисел
# в диапазоне от 1 до 3, например 2, 2, 3, 2, 1
# Hint: типовая реализация занимает 2 строки
import random

def gen_random(num_count, begin, end):
    for i in range(num_count):
        yield random.randint(begin, end)

if __name__ == '__main__':
    for i in gen_random(4, 1, 100): print(i)
```

Файлы: sort.py и cm\_timer.py



```
field.py  sort.py  print_result.py  lab_3.py  unique.py  gen_random.py  cm_timer.py

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

if __name__ == '__main__':
    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)

import time
from contextlib import contextmanager

class cm_timer_1:
    def __init__(self):
        self.start = None

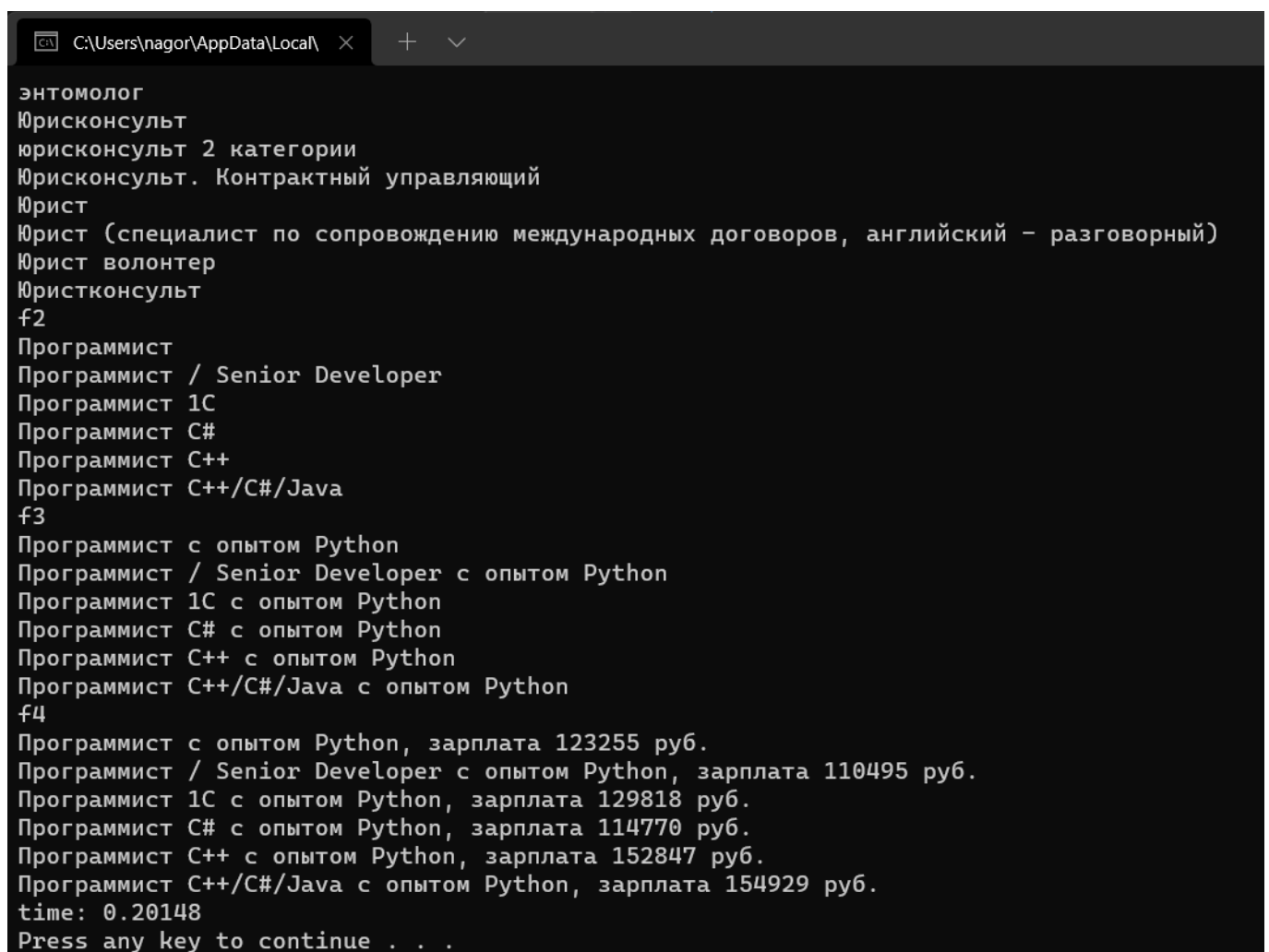
    def __enter__(self):
        self.start = time.perf_counter()

    def __exit__(self, exc_type, exc_val, exc_tb):
        print('time: {:.5f}'.format(time.perf_counter() - self.start))

@contextmanager
def cm_timer_2():
    start = time.perf_counter()
    yield
    print('time: {:.5f}'.format(time.perf_counter() - start))

if __name__ == '__main__':
    with cm_timer_1():
        time.sleep(5.5)
    with cm_timer_2():
        time.sleep(5.5)
```

## Результат выполнения



```
C:\Users\nagor\AppData\Local\  +  v

ЭНТОМОЛОГ
Юрисконсульт
юрисконсульт 2 категории
Юрисконсульт. Контрактный управляющий
Юрист
Юрист (специалист по сопровождению международных договоров, английский – разговорный)
Юрист волонтер
Юристконсульт
f2
Программист
Программист / Senior Developer
Программист 1С
Программист C#
Программист C++
Программист C++/C#/Java
f3
Программист с опытом Python
Программист / Senior Developer с опытом Python
Программист 1С с опытом Python
Программист C# с опытом Python
Программист C++ с опытом Python
Программист C++/C#/Java с опытом Python
f4
Программист с опытом Python, зарплата 123255 руб.
Программист / Senior Developer с опытом Python, зарплата 110495 руб.
Программист 1С с опытом Python, зарплата 129818 руб.
Программист C# с опытом Python, зарплата 114770 руб.
Программист C++ с опытом Python, зарплата 152847 руб.
Программист C++/C#/Java с опытом Python, зарплата 154929 руб.
time: 0.20148
Press any key to continue . . .
```