

**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 №3–4.**

Выполнил:  
студент группы ИУ5-34Б:  
Белозеров Дмитрий Сергеевич  
Подпись и дата:

Проверил:  
преподаватель каф. ИУ5  
Гапанюк Юрий Евгеньевич  
Подпись и дата:

## Задание:

Задание лабораторной работы состоит из решения нескольких задач.

Файлы, содержащие решения отдельных задач, должны располагаться в пакете lab\_python\_fr. 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', 'price': 5300}

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(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.
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, Абв и АБВ - разные строки
        #                 ignore_case = False, Абв и АБВ - 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, 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))))

```

## Решение задач

### field.py

```

def field(goods, *args):
    assert len(args) > 0
    line = {}
    for i in goods:
        for v,b in i.items():
            if v in args:
                safe_v = v
                line[v] = b
        if len(line) == 0:
            yield None
        if len(line) == 1:
            yield "" + line[safe_v] + ""
        else:
            yield line

if __name__ == '__main__':
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'color': 'black'}
    ]
    for i in field(goods, 'title'):
        print(i)
    for i in field(goods, 'title', 'price'):
        print(i)

```

### Результат

```
C:\Users\beloz\PycharmProjects\Lab_3\venv\Script
'Ковер'
'Диван для отдыха'
{'title': 'Ковер', 'price': 2000}
{'title': 'Диван для отдыха', 'price': 2000}

Process finished with exit code 0
```

## gen\_random.py

```
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(5, 1, 3):
        print(i, end = " ")
```

## Результат

```
C:\Users\beloz\PycharmProjects\Lab_3\venv\Script
2 1 2 1 1
Process finished with exit code 0
```

## unique.py

```
import gen_random
class Unique(object):
    def __init__(self, items, **kwargs):
        if 'ignore_case' in kwargs:
            items = list(map(str, items))
            items = [x.lower() for x in items]
            self.items = list(set(items))
        else:
            self.items = list(set(items))
    def __next__(self):
        self.items = iter(self.items)
        return next(self.items)
    def __iter__(self):
        return self

if __name__ == '__main__':
    #####
    print("1st test")
    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    t = Unique(data)
    print(next(t))
    print(next(t))
    print(next(t))
    print(next(t))
    #####
    print("2nd test")
    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
```



```

t = Unique(data, ignore_case=True)
print(next(t))
print(next(t))
#####
print("3rd test")
data = [1,1,1,1,1,1,1,2,2,2,2,2,2]
t = Unique(data)
print(next(t))
print(next(t))
#####
print("4th test")
data = gen_random.gen_random(10, 1, 3)
t = Unique(data)
print(next(t))
print(next(t))
print(next(t))
#####

```

## Результат

```

1st test
A
b
a
B
2nd test
b
a
3rd test
1
2
4th test
1
2
3

Process finished with exit code 0

```

## sort.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))
    print(result_with_lambda)

```

## Результат

```
C:\Users\beloz\PycharmProjects\Lab_3\ve
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]

Process finished with exit code 0
```

### print\_result.py

```
def print_result(func):
    def inner(*args, **kwargs):
        print(func.__name__)
        res = func(*args, **kwargs)
        if type(res) == list:
            for i in res:
                print(i)
        elif type(res) == dict:
            for key in res.keys():
                print(f'''{key} = {res[key]}''')
        else:
            print(res)
        return res
    return inner

'''
def print_result(func):
    def wrapper(lis=[], *args, **kwargs):
        print(f"Название функции: {func.__name__}")
        if len(lis) == 0:
            line = func(*args, **kwargs)
        else:
            line = func(lis, *args, **kwargs)
        if type(line) == list:
            for i in line:
                print(i)
        if type(line) == dict:
            for key, value in line.items():
                print("{} = {}".format(key, value))
        if type(line) == int:
            print(line)
        if type(line) == str:
            print(line)
        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]

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

```

## Результат

```

C:\Users\det02\PycharmProjects\Lab_
!!!!!!!
test_1
1
test_2
105
test_3
a = 1
b = 2
test_4
1
2

Process finished with exit code 0

```

## cm\_timer.py

```

from contextlib import contextmanager
import time

@contextmanager
def cm_timer_1():
    start_time = time.time()
    yield
    print("time: ", time.time() - start_time)

class cm_timer_2:
    def __init__(self):
        self.start_time = 0
    def __enter__(self):
        self.start_time = time.time()
    def __exit__(self, type, value, traceback):
        print("time: ", time.time() - self.start_time)

if __name__ == "__main__":
    with cm_timer_1():
        time.sleep(5.5)

```

```
with cm_timer_2():  
    time.sleep(5.5)
```

## Результат

```
C:\Users\beloz\PycharmProjects\Lab_3  
time: 5.50301718711853  
time: 5.502365827560425  
  
Process finished with exit code 0
```

## process\_data.py

```
import json  
from gen_random import gen_random  
from unique import Unique  
from print_result import print_result  
import cm_timer  
  
path = 'data_light.json'  
  
with open(path, encoding="utf8") as json_file:  
    data = json.load(json_file)  
  
@print_result  
def f1(arg):  
    return list(Unique([x['job-name'] for x in arg]))  
  
@print_result  
def f2(arg):  
    return list(filter(lambda s: s.startswith('Программист'), arg))  
  
@print_result  
def f3(arg):  
    return list(map(lambda s: s + ' с опытом Python', arg))  
  
@print_result  
def f4(arg):  
    return list(zip(arg, gen_random(len(arg), 100000, 200000)))  
  
if __name__ == '__main__':  
    with cm_timer.cm_timer_1():  
        f4(f3(f2(f1(data))))
```

## Результат

f2

Программист

Программист/ Junior Developer

Программист/ технический специалист

Программист C++/C#/Java

Программист 1C

Программист C++

Программист / Senior Developer

Программист C#

Программист-разработчик информационных систем

f3

Программист с опытом Python

Программист/ Junior Developer с опытом Python

Программист/ технический специалист с опытом Python

Программист C++/C#/Java с опытом Python

Программист 1C с опытом Python

Программист C++ с опытом Python

Программист / Senior Developer с опытом Python

Программист C# с опытом Python

Программист-разработчик информационных систем с опытом Python

f4

('Программист с опытом Python', 163239)

('Программист/ Junior Developer с опытом Python', 136184)

('Программист/ технический специалист с опытом Python', 137892)

('Программист C++/C#/Java с опытом Python', 158356)

('Программист 1C с опытом Python', 197020)

('Программист C++ с опытом Python', 146746)

('Программист / Senior Developer с опытом Python', 180084)

('Программист C# с опытом Python', 102984)

('Программист-разработчик информационных систем с опытом Python', 146284)

time: 0.03265881538391113