МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет “Информатика и системы управления”
Кафедра “Системы обработки информации и управления”

Дисциплина “ Парадигмы и конструкции языков программирования”

Отчет по лабораторной работе №3-4

Выполнила:
Студент группы ИУ5-34Б
Глозман В.А.
Преподаватель:
Гапанюк Ю.Е.

Москва 2025

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. 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.

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 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, 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 выводить 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.

data_light.json:

```
[
  {"job-name": "Программист C++", "salary": 50000},
  {"job-name": "Менеджер", "salary": 30000},
  {"job-name": "программист Java", "salary": 60000}
]
```

field.py:

```
def field(items, *args):
    assert len(args) > 0
    if len(args) == 1:
        for item in items:
            val = item.get(args[0])
            if val is not None:
                yield val
    else:
        for item in items:
            current_dict = {key: item.get(key) for key in args if item.get(key) is not None}
            if len(current_dict) > 0:
                yield current_dict


if __name__ == '__main__':
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
```

```
        {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
        {'title': 'Стелаж', 'price': None, 'color': 'white'}
    ]
    print(list(field(goods, 'title')))
    print(list(field(goods, 'title', 'price')))
```

gen_random.py:

```
import random

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

if __name__ == '__main__':
    print(list(gen_random(5, 1, 3)))
```

unique.py:

```
class Unique(object):
    def __init__(self, items, **kwargs):
        self.ignore_case = kwargs.get('ignore_case', False)
        self.items = iter(items)
        self.used_elements = set()

    def __next__(self):
        while True:
            current = next(self.items)

            check_val = current
            if self.ignore_case and isinstance(current, str):
                check_val = current.lower()
```

```
            if check_val not in self.used_elements:
                self.used_elements.add(check_val)
                return current

    def __iter__(self):
        return self

if __name__ == '__main__':
    data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    print(list(Unique(data)))

    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    print(list(Unique(data)))
    print(list(Unique(data, ignore_case=True)))
```

sort.py:

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

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

    result = sorted(data, key=abs, reverse=True)
    print(result)
```

print_result.py:

```
def print_result(func):
    def wrapper(*args, **kwargs):
        print(func.__name__)
        result = func(*args, **kwargs)
```

```
        if isinstance(result, list):
            for item in result:
                print(item)
        elif isinstance(result, dict):
            for key, value in result.items():
                print(f'{key} = {value}')
        else:
            print(result)

        return result
    return wrapper

if __name__ == '__main__':
    @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]
```

```
    test_1()
    test_2()
    test_3()
    test_4()
```

cm_timer.py:

```
import time
from contextlib import contextmanager

class cm_timer_1:
    def __enter__(self):
        self.start_time = time.time()
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        end_time = time.time()
        print(f'time: {end_time - self.start_time}')

@contextmanager
def cm_timer_2():
    start_time = time.time()
    yield
    end_time = time.time()
    print(f'time: {end_time - start_time}')

if __name__ == '__main__':
    with cm_timer_1():
        time.sleep(0.5)

    with cm_timer_2():
```

```
    time.sleep(0.5)
```

process_data.py:

```
mport json
import sys
from lab_python_fp.print_result import print_result
from lab_python_fp.cm_timer import cm_timer_1
from lab_python_fp.unique import Unique
from lab_python_fp.field import field
from lab_python_fp.gen_random import gen_random

path = 'data_light.json'

with open(path, encoding='utf-8') as f:
data = json.load(f)
@print_result
def f1(arg):
return sorted(Unique(field(arg, 'job-name'), ignore_case=True), key=lambda x:
x.lower())
@print_result
def f2(arg):
return list(filter(lambda x: x.lower().startswith('программист'), arg))
@print_result
def f3(arg):
return list(map(lambda x: x + ' с опытом Python', arg))
@print_result
def f4(arg):
salaries = gen_random(len(arg), 100000, 200000)
return list(zip(arg, salaries))
if __name__ == '__main__':
```

with cm_timer_1():

f4(f3(f2(f1(data))))

Результат выполнения: