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

Выполнил:

Студент группы ИУ5-35Б

Костылев М.С.

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', '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 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.

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))))
```

Код программы:

cm_timer.py:

```
import time
from contextlib import contextmanager


# Реализация на основе класса
class cm_timer_1:
    def __enter__(self):
        self.start_time = time.time()
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        elapsed_time = time.time() - self.start_time
        print(f"time: {elapsed_time:.1f}")


# Реализация с использованием contextlib
@contextmanager
def cm_timer_2():
    start_time = time.time()
    yield
    elapsed_time = time.time() - start_time
    print(f"time: {elapsed_time:.1f}")


if __name__ == '__main__':
```

```
    print("Test cm_timer_1:")
    with cm_timer_1():
        time.sleep(0.5)

    print("\nTest cm_timer_2:")
    with cm_timer_2():
        time.sleep(0.5)
```

data_light.json:

```
[
    {"job-name": "Программист C#"},
    {"job-name": "Программист Python"},
    {"job-name": "Аналитик данных"},
    {"job-name": "Менеджер проекта"},
    {"job-name": "Программист Java"},
    {"job-name": "Дизайнер UI/UX"},
    {"job-name": "Программист JavaScript"},
    {"job-name": "Тестировщик"},
    {"job-name": "DevOps инженер"},
    {"job-name": "программист Python"},
    {"job-name": "Data Scientist"},
    {"job-name": "Программист C++"},
    {"job-name": "Системный администратор"},
    {"job-name": "Программист Go"}
]
```

field.py:

```
def field(items, *args):
    assert len(args) > 0

    # Если передан один аргумент
    if len(args) == 1:
        key = args[0]
        for item in items:
            if key in item and item[key] is not None:
                yield item[key]
    # Если передано несколько аргументов
    else:
        for item in items:
            result = {}
            has_valid_field = False
            for key in args:
                if key in item and item[key] is not None:
                    result[key] = item[key]
                    has_valid_field = True

            if has_valid_field:
                yield result


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

```
        {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
        {'title': None, 'price': 1000, 'color': 'white'},
        {'title': 'Стул', 'price': None, 'color': 'brown'}
    ]

    print("Test 1 - один аргумент:")
    for title in field(goods, 'title'):
        print(title)

    print("\nTest 2 - несколько аргументов:")
    for item in field(goods, 'title', 'price'):
        print(item)
```

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("Test gen_random:")
    random_numbers = list(gen_random(5, 1, 3))
    print(f"gen_random(5, 1, 3): {random_numbers}")
```

print_result.py:

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

        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


@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()
```

process_data.py:

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

```
# Импортируем все нужные функции
try:
    path = sys.argv[1]
except IndexError:
    path = "data_light.json"

with open(path, encoding='utf-8') as f:
    data = json.load(f)



@print_result
def f1(arg):
    # Сортированный список профессий без повторений (игнорируя регистр)
    professions = list(field(arg, 'job-name'))
    unique_professions = list(Unique(professions, ignore_case=True))
    return sorted(unique_professions, key=lambda x: x.lower())



@print_result
def f2(arg):
    # Фильтруем только профессии, начинающиеся со слова "программист"
    return list(filter(lambda x: x.lower().startswith('программист'), arg))



@print_result
def f3(arg):
```

```
    # Добавляем "с опытом Python" к каждой профессии
    return list(map(lambda x: f"{x} с опытом Python", arg))


@print_result
def f4(arg):
    # Генерируем зарплаты и объединяем с профессиями
    salaries = list(gen_random(len(arg), 100000, 200000))
    return [f"{profession}, зарплата {salary} руб." for profession, salary in zip(arg, salaries)]


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

sort.py:

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

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

    # С lambda
    result_with_lambda = sorted(data, key=lambda x: abs(x), reverse=True)
    print("\nС lambda:")
```

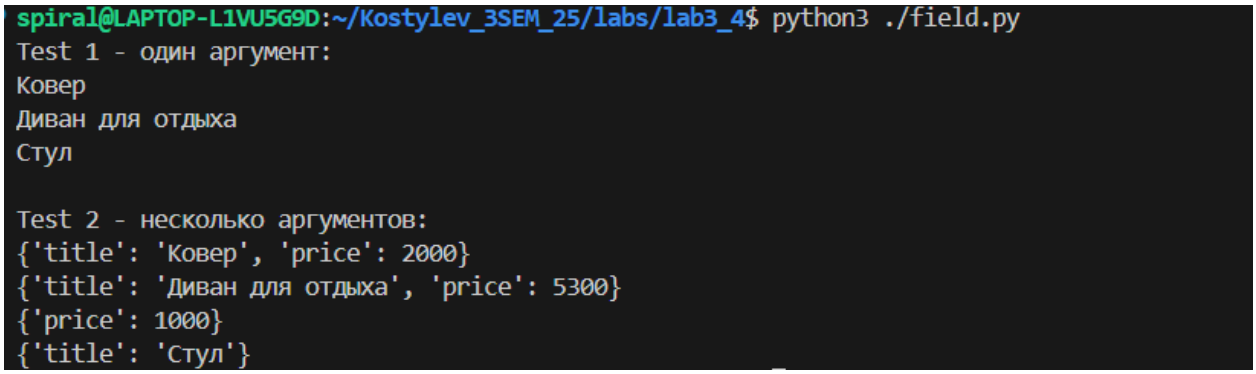
```
    print(result_with_lambda)
```

unique.py:

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

    def _unique_generator(self):
        for item in self.items:
            # Для строк с учетом регистра
            if isinstance(item, str) and self.ignore_case:
                key = item.lower()
            else:
                key = item

            if key not in self.seen:
                self.seen.add(key)
                yield item

    def __next__(self):
        return next(self.iterator)

    def __iter__(self):
        return self
```

```
if __name__ == '__main__':
    print("Test 1 - числа:")
    data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    for item in Unique(data1):
        print(item, end=' ')
    print()

    print("\nTest 2 - строки без ignore_case:")
    data2 = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    for item in Unique(data2):
        print(item, end=' ')
    print()

    print("\nTest 3 - строки с ignore_case=True:")
    for item in Unique(data2, ignore_case=True):
        print(item, end=' ')
    print()
```

Работа программы:



```
spiral@LAPTOP-L1VU5G9D:~/Kostylev_3SEM_25/labs/lab3_4$ python3 ./gen_random.py
Test gen_random:
gen_random(5, 1, 3): [3, 1, 2, 1, 3]
```

```
spiral@LAPTOP-L1VU5G9D:~/Kostylev_3SEM_25/labs/lab3_4$ python3 unique.py
Test 1 - числа:
1 2

Test 2 - строки без ignore_case:
a A b B

Test 3 - строки с ignore_case=True:
a b
```

```
spiral@LAPTOP-L1VU5G9D:~/Kostylev_3SEM_25/labs/lab3_4$ python3 print_result.py
!!!!!!!!
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
```

```
spiral@LAPTOP-L1VU5G9D:~/Kostylev_3SEM_25/labs/lab3_4$ python3 sort.py
Без lambda:
[123, 100, -100, -30, 30, 4, -4, 1, -1, 0]

С lambda:
[123, 100, -100, -30, 30, 4, -4, 1, -1, 0]
```

```
spiral@LAPTOP-L1VU5G9D:~/Kostylev_3SEM_25/labs/lab3_4$ python3 cm_timer.py
Test cm_timer_1:
time: 0.5

Test cm_timer_2:
time: 0.5
spiral@LAPTOP-L1VU5G9D:~/Kostylev_3SEM_25/labs/lab3_4$
```

```
spiral@LAPTOP-L1VU5G9D:~/Kostylev_3SEM_25/labs/lab3_4$ python3 process_data.py data_light.json
f1
Data Scientist
DevOps инженер
Аналитик данных
Дизайнер UI/UX
Менеджер проекта
Программист C#
Программист C++
Программист Go
Программист Java
Программист JavaScript
Программист Python
Системный администратор
Тестировщик
f2
Программист C#
Программист C++
Программист Go
Программист Java
Программист JavaScript
Программист Python
f3
Программист C# с опытом Python
Программист C++ с опытом Python
Программист Go с опытом Python
Программист Java с опытом Python
Программист JavaScript с опытом Python
Программист Python с опытом Python
f4
Программист C# с опытом Python, зарплата 126177 руб.
Программист C++ с опытом Python, зарплата 143662 руб.
Программист Go с опытом Python, зарплата 178217 руб.
Программист Java с опытом Python, зарплата 137462 руб.
Программист JavaScript с опытом Python, зарплата 122334 руб.
Программист Python с опытом Python, зарплата 125784 руб.
time: 0.0
spiral@LAPTOP-L1VU5G9D:~/Kostylev_3SEM_25/labs/lab3_4$
```