Московский государственный технический университет им. Н.Э. Баумана

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

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №3-4

«Функциональные возможности языка Python»

Выполнил:
студент группы ИУ5 - 32Б:
Васильев Н. Д.
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю. Е.
Подпись и дата:

Москва, 2025 г.

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.

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.

Код:

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

    if len(args) == 1:
        field_name = args[0]
        for item in items:
            if field_name in item and item[field_name] is not None:
                yield item[field_name]
    else:
        for item in items:
            result = {}
            for field_name in args:
                if field_name in item and item[field_name] is not None:
                    result[field_name] = item[field_name]
            if result:
                yield result


if __name__ == "__main__":
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'color': 'black'},
        {'title': None, 'price': 3000},
        {'price': 2500}
    ]

    print("Тест field:")
```

```
    print("Один аргумент:", list(field(goods, 'title')))
    print("Несколько аргументов:", list(field(goods, 'title', 'price')))
```

```
C:\Users\Николай\Desktop\Python_Course_3Sem\.venv\Scripts\python.exe C:\Users\Николай\Desktop\Python_Course_3Sem\Lab3-3Sem\src\field.py
Тест field:
Один аргумент: ['Ковер', 'Диван для отдыха']
Несколько аргументов: [{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха'}, {'price': 3000}, {'price': 2500}]

Process finished with exit code 0
```

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

Код:

```
import random

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


if __name__ == "__main__":
    print("Тест gen_random:", list(gen_random(8, 1, 78)))
```

```
C:\Users\Николай\Desktop\Python_Course_3Sem\.venv\Scripts\python.exe C:\Users\Николай\Desktop\Python_Course_3Sem\Lab3-3Sem\src\gen_random.py
Тест gen_random: [75, 30, 37, 65, 64, 62, 54, 24]

Process finished with exit code 0
```

```
C:\Users\Николай\Desktop\Python_Course_3Sem\.venv\Scripts\python.exe C:\Users\Николай\Desktop\Python_Course_3Sem\Lab3-3Sem\src\gen_random.py
Тест gen_random: [69, 5, 2, 74, 67, 50, 49, 13]

Process finished with exit code 0
```

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.

Код:

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

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

            if self.ignore_case and isinstance(item, str):
                key = item.lower()
            else:
                key = item

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

    def __iter__(self):
        return self


if __name__ == "__main__":
    data = [1, 1, 2, 2, 4, 3]
    print("Тест Unique:", list(Unique(data)))
```

```
C:\Users\Николай\Desktop\Python_Course_3Sem\.venv\Scripts\python.exe C:\Users\Николай\Desktop\Python_Course_3Sem\Lab3-3Sem\src\unique.py
Тест Unique: [1, 2, 4, 3]


Process finished with exit code 0
```

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.

Код:

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

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

    print("Без lambda:", result)
    print("С lambda:", result_with_lambda)
```

```
C:\Users\Николай\Desktop\Python_Course_3Sem\.venv\Scripts\python.exe C:\Users\Николай\Desktop\Python_Course_3Sem\Lab3-3Sem\src\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]

Process finished with exit code 0
```

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.

Код:

```
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


if __name__ == '__main__':
    @print_result
    def test_1():
        return 1


    @print_result
    def test_2():
        return 'iu5'
```

```python
@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()
```

```
C:\Users\Николай\Desktop\Python_Course_3Sem\.venv\Scripts\python.exe C:\Users\Николай\Desktop\Python_Course_3Sem\Lab3-3Sem\src\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
```

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).

Код:

```
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):
        elapsed_time = time.time() - self.start_time
        print(f"time: {elapsed_time:.1f}")


@contextmanager
def cm_timer_2():
    start_time = time.time()
    try:
        yield
    finally:
        elapsed_time = time.time() - start_time
        print(f"time: {elapsed_time:.1f}")


if __name__ == "__main__":
    with cm_timer_1():
        time.sleep(1)

    with cm_timer_2():
        time.sleep(1)
```

```
C:\Users\Николай\Desktop\Python_Course_3Sem\.venv\Scripts\python.exe C:\Users\Николай\Desktop\Python_Course_3Sem\Lab3-3Sem\src\cm_timer.py
time: 1.0
time: 1.0

Process finished with exit code 0
```

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:

```
import json
import sys
import os

# Добавляем путь к src для импорта модулей
sys.path.append(os.path.join(os.path.dirname(__file__), '..'))

from src.field import field
from src.gen_random import gen_random
from src.unique import Unique
from src.print_result import print_result
from src.cm_timer import cm_timer_1

# Путь к данным
```

```python
path = os.path.join(os.path.dirname(__file__), '..', 'data', 'data_light.json')

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

@print_result
def f1(arg):
    return sorted(list(Unique(field(arg, 'job-name'), ignore_case=True)),
key=str.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: f"{x} с опытом Python", arg))

@print_result
def f4(arg):
    salaries = list(gen_random(len(arg), 100000, 200000))
    return list(map(lambda x: f"{x[0]}, зарплата {x[1]} руб.", zip(arg, salaries)))


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

```
C:\Users\Николай\Desktop\Python_Course_3Sem\.venv\Scripts\python.exe C:\Users\Николай\Desktop\Python_Course_3Sem\Lab3-3Sem\src\process_data.py
f1
Аналитик данных
Дизайнер
Менеджер проекта
Программист C#
программист C++
программист Java
Программист JavaScript
Программист Python
f2
Программист C#
программист C++
программист Java
Программист JavaScript
Программист Python
f3
Программист C# с опытом Python
программист C++ 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 192443 руб.
программист C++ с опытом Python, зарплата 176827 руб.
программист Java с опытом Python, зарплата 185119 руб.
Программист JavaScript с опытом Python, зарплата 177753 руб.
Программист Python с опытом Python, зарплата 133917 руб.
time: 0.0

Process finished with exit code 0
```

Файл main.py:

```
import os
import sys

# Добавляем src в путь для импорта
sys.path.append(os.path.join(os.path.dirname(__file__), 'src'))


def run_all_tests():
    """Запуск всех тестов"""
    print("=" * 50)
    print("ЗАПУСК ВСЕХ ТЕСТОВ ЛАБОРАТОРНОЙ РАБОТЫ")
    print("=" * 50)

    # Импортируем и запускаем каждый модуль
    from src import field, gen_random, unique, sort, print_result, cm_timer,
process_data

    print("\n1. Тестирование field:")
    import src.field

    print("\n2. Тестирование gen_random:")
    import src.gen_random

    print("\n3. Тестирование unique:")
    import src.unique

    print("\n4. Тестирование sort:")
    import src.sort

    print("\n5. Тестирование print_result:")
    import src.print_result

    print("\n6. Тестирование cm_timer:")
    import src.cm_timer

    print("\n7. Тестирование process_data:")
    import src.process_data


if __name__ == "__main__":
    run_all_tests()
```