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
"Функциональные возможности языка Python"

Выполнил:
Студент группы ИУ5-35Б
Королев М.О.
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 располагаться в отдельном файле.

При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

https://github.com/koromax/Korolev-PCPL-Labs-2025/tree/main/lab3

Задача 1

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.

Листинг кода:

```python
# lab_python_fp/field.py
def field(items, *args):
    assert len(args) > 0, "Введите аргументы"

    out = []

    if len(args) == 1:
        out = [e[args[0]] for e in items if args[0] in e.keys()]
    else:
        out = [{arg:e[arg] for arg in e.keys() if arg in args} for e in items if any(arg in e.keys() for arg in args)]

    return out

if __name__ == "__main__":
    goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'color': 'black'},
    {'color': 'blue'}
    ]

    print(*field(goods, "title"), sep='\n')
    print(*field(goods, "title", "price"), sep='\n')
```

Результат выполнения программы:

```
10:Korolev-PCPL-Labs-2025 koromax$ python3 ./lab3/lab_python_fp/field.py
Ковер
Диван для отдыха
{'title': 'Ковер', 'price': 2000}
{'title': 'Диван для отдыха'}
10:Korolev-PCPL-Labs-2025 koromax$
```

Задача 2

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.

Листинг кода:

```python
# lab_python_fp/gen_random.py
import random

def gen_random(n, min, max):
    return [random.randint(min, max) for i in range(n)]

if __name__ == "__main__":
    print(*gen_random(10, 1, 10))
```

Результат выполнения программы:

```
10:Korolev-PCPL-Labs-2025 koromax$ python3 ./lab3/lab_python_fp/gen_random.py
1 4 6 8 2 3 4 8 8 6
10:Korolev-PCPL-Labs-2025 koromax$
```

Задача 3

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.

Листинг кода:

```python
# lab_python_fp/gen_random.py
class Unique:
    def __init__(self, items, **kwargs):
        Unique._data = items
        Unique._used = set()
        Unique._index = 0
        Unique.ignore_case = kwargs["ignore_case"] if (len(kwargs) > 0 and "ignore_case" in
kwargs.keys()) else False

    def __iter__(self):
        return self
```

```
    def __next__(self):
        while self._index < len(self._data):
            item = str(self._data[self._index])
            self._index += 1

            if self.ignore_case and item.lower() in [e.lower() for e in self._used]:
                continue
            if not self.ignore_case and item in self._used:
                continue

            self._used.add(item)
            return item

        raise StopIteration


if __name__ == "__main__":
    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']

    print(list(Unique(data)))
    print(list(Unique(data, ignore_case=True)))
```

Результат выполнения программы:

```
10:Korolev-PCPL-Labs-2025 koromax$ python3 ./lab3/lab_python_fp/unique.py
['a', 'A', 'b', 'B']
['a', 'b']
10:Korolev-PCPL-Labs-2025 koromax$
```

Задача 4

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.

Листинг кода:

```
# lab_python_fp/sort.py
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]

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)
```

Результат выполнения программы:

```
10:Korolev-PCPL-Labs-2025 koromax$ python3 ./lab3/lab_python_fp/sort.py
[123, 100, -100, -30, 4, -4, 1, -1, 0]
[123, 100, -100, -30, 4, -4, 1, -1, 0]
10:Korolev-PCPL-Labs-2025 koromax$
```

Задача 5

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.

Листинг кода:

```python
# lab_python_fp/print_result.py
def print_result(func):
    def wrapper(*args, **kwargs):
        print(func.__name__)
        out = func(*args, **kwargs)

        if isinstance(out, list):
            print(*out, sep='\n')
        elif isinstance(out, dict):
            print(*[f"{key} = {out[key]}" for key in out.keys()], sep='\n')
        else:
            print(out)

        return out

    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]
```

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

Результат выполнения программы:

```
10:Korolev-PCPL-Labs-2025 koromax$ python3 ./lab3/lab_python_fp/print_result.py
!!!!!!!!
test_1
1
test_2
iu5
test_3
a = 1
b = 2
test_4
1
2
10:Korolev-PCPL-Labs-2025 koromax$
```

Задача 6

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).

Листинг кода:

```python
# lab_python_fp/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):
        print(f"time: {time.time() - self.start_time}")

@contextmanager
def cm_timer_2():
    start_time = time.time()

    try:
        yield start_time
    finally:
```

```
        print(f"time: {time.time() - start_time}")

if __name__ == "__main__":
    with cm_timer_1():
        time.sleep(1.5)
    with cm_timer_2():
        time.sleep(1.5)
```

Результат выполнения программы:

```
10:Korolev-PCPL-Labs-2025 koromax$ python3 ./lab3/lab_python_fp/cm_timer.py
time: 1.5012917518615723
time: 1.5050599575042725
10:Korolev-PCPL-Labs-2025 koromax$
```

Задача 7

- 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.

Листинг кода:

```
# lab_python_fp/process_data.py
import json

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

path = "lab3/lab_python_fp/data_light.json"

# 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(data):
    return list(Unique(field(data, "job-name"), ignore_case=True))


@print_result
def f2(data):
    return list(filter(lambda x: x.lower().startswith("программист"), data))


@print_result
def f3(data):
    return list(map(lambda x: f"{x} с опытом Python", data))


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

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

Результат выполнения программы:

```
Оператор склада
Специалист по электромеханическим испытаниям аппаратуры бортовых космических систем
Заведующий музеем в д.Копорье
Документовед
Специалист по испытаниям на электромагнитную совместимость аппаратуры бортовых космических систем
Менеджер (в промышленности)
f2
Программист
Программист C++/C#/Java
Программист 1C
Программистр-разработчик  информационных систем
Программист C++
Программист/ Junior Developer
Программист / Senior Developer
Программист/ технический специалист
Программист C#
f3
Программист с опытом Python
Программист C++/C#/Java с опытом Python
Программист 1C с опытом Python
Программистр-разработчик  информационных систем с опытом Python
Программист C++ с опытом Python
Программист/ Junior Developer с опытом Python
Программист / Senior Developer с опытом Python
Программист/ технический специалист с опытом Python
Программист C# с опытом Python
f4
Программист с опытом Python, зарплата 197089 руб
Программист C++/C#/Java с опытом Python, зарплата 108727 руб
Программист 1C с опытом Python, зарплата 194556 руб
Программистр-разработчик  информационных систем с опытом Python, зарплата 143164 руб
Программист C++ с опытом Python, зарплата 120191 руб
Программист/ Junior Developer с опытом Python, зарплата 122860 руб
Программист / Senior Developer с опытом Python, зарплата 193550 руб
Программист/ технический специалист с опытом Python, зарплата 185726 руб
Программист C# с опытом Python, зарплата 118780 руб
time: 0.7860190868377686
10:Korolev-PCPL-Labs-2025 koromax$
```