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 ИУ5-36Б
Юдин Григорий Олегович
Проверил:
Гапанюк Юрий Евгеньевич

Москва, 2025

Code cm_timer.py

```
# cm_timer.py

import time
from contextlib import contextmanager


class cm_timer_1:
    """Контекстный менеджер на основе класса."""

    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:.4f}")


@contextmanager
def cm_timer_2():
    """Контекстный менеджер с использованием contextlib."""
    start_time = time.time()
    try:
        yield
    finally:
        elapsed_time = time.time() - start_time
        print(f"time: {elapsed_time:.4f}")


if __name__ == '__main__':
    print("--- Тестирование cm_timer_1 (класс) ---")
    with cm_timer_1():
        time.sleep(1.2)

    print("\n--- Тестирование cm_timer_2 (contextlib) ---")
    with cm_timer_2():
        time.sleep(1.8)
```

Code field.py

```
def field(items, *args):
    assert len(args) > 0
    if len(args) == 1:
        key = args[0]
        for i in items:
            value = i.get(key)
            if value is not None:
                yield value
    else:
        for i in items:
            filtered = {k: v for k, v in i.items() if k in args and v is not None}
            if filtered:
                yield filtered

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

    print("--- Тест field с одним аргументом ('title') ---")
    for title in field(goods, 'title'):
        print(title)

    print("\n--- Тест field с одним аргументом ('price'), пропуская None
---")
    for price in field(goods, 'price'):
        print(price)

    print("\n--- Тест field с несколькими аргументами ('title', 'price',
'color') ---")
    for item in field(goods, 'title', 'price', 'color'):
        print(item)
```

Code gen_random.py

```
def field(items, *args):
    assert len(args) > 0
    if len(args) == 1:
        key = args[0]
        for i in items:
            value = i.get(key)
            if value is not None:
                yield value
    else:
        for i in items:
            filtered = {k: v for k, v in i.items() if k in args and v is not None}
            if filtered:
                yield filtered

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

    print("--- Тест field с одним аргументом ('title') ---")
    for title in field(goods, 'title'):
        print(title)

    print("\n--- Тест field с одним аргументом ('price'), пропуская None
---")
    for price in field(goods, 'price'):
        print(price)

    print("\n--- Тест field с несколькими аргументами ('title', 'price',
'color') ---")
    for item in field(goods, 'title', 'price', 'color'):
        print(item)
```

Code print_result.py

```
# print_result.py

from functools import wraps


def print_result(func):
    """
    Декоратор, который печатает имя функции и ее результат.
    Форматирует вывод для списков и словарей.
    """

    @wraps(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


# Блок main для проверки декоратора
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]

print("--- Запуск тестов для декоратора print_result ---\n")
test_1()
print("-" * 20)
test_2()
print("-" * 20)
test_3()
print("-" * 20)
test_4()
print("\n--- Тесты завершены ---")
```

Code process_data.py

```
# process_data.py

# Импортируем все необходимые заготовки из предыдущих задач
from cm_timer import cm_timer_1
from field import field
from gen_random import gen_random
from print_result import print_result
from unique import Unique


@print_result
def f1(arg):
    """
    Возвращает отсортированный список уникальных профессий (без учета регистра).
    """
    return sorted(list(Unique(field(arg, "job-name"), ignore_case=True)), key=str.lower)


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


@print_result
def f3(arg):
    """
    Добавляет к каждой профессии "с опытом Python".
    """
    return list(map(lambda x: x + " с опытом Python", arg))


@print_result
```

```
def f4(arg):
    """
    Добавляет к каждой профессии случайную зарплату от 100 000 до 200 000 руб.
    """
    salaries = gen_random(len(arg), 100000, 200000)
    return [f"{prof}, зарплата {salary} руб." for prof, salary in zip(arg, salaries)]


if __name__ == "__main__":
    mock_data = [
        {"job-name": "Программист Python", "salary": "150000"},
        {"job-name": "Аналитик данных", "salary": "120000"},
        {"job-name": "программист C++", "salary": "180000"},
        {"job-name": "Инженер-программист", "salary": "170000"},
        {"job-name": "Программист Java", "salary": "190000"},
        {"job-name": "Программист Python", "salary": "160000"},  # Дубликат
    ]

    print("--- Запуск цепочки обработки с тестовыми данными ---")
    with cm_timer_1():
        f4(f3(f2(f1(mock_data))))
    print("--- Обработка завершена ---")
```

Code sort.py

```
# sort.py

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

# Блок main уже был в этом задании, он выполняет проверку
if __name__ == '__main__':
    print(f"Исходные данные: {data}\n")

    print("--- Сортировка по модулю (без lambda) ---")
    result = sorted(data, key=abs, reverse=True)
    print(result)

    print("\n--- Сортировка по модулю (с lambda) ---")
    result_with_lambda = sorted(data, key=lambda x: abs(x),
reverse=True)
    print(result_with_lambda)
```

Code unique.py

```
# unique.py

# Для теста импортируем генератор из предыдущего задания
from gen_random import gen_random


class Unique(object):
    """
    Итератор, который убирает дубликаты из итерируемого объекта.

    Поддерживает опцию ignore_case для строк.
    """

    def __init__(self, items, **kwargs):
        self._items = iter(items)
        self._ignore_case = kwargs.get('ignore_case', False)
        self._seen = set()

    def __next__(self):
        while True:
            i = next(self._items)
            check_item = i
            if self._ignore_case and isinstance(i, str):
                check_item = i.lower()

            if check_item not in self._seen:
                self._seen.add(check_item)
                return i

    def __iter__(self):
        return self


if __name__ == '__main__':
    print("--- Тест Unique с числами ---")
    data1 = [1, 1, 1, 2, 2, 3, 1, 2]
    print(f"Исходные данные: {data1}")
    print("Уникальные:", end=' ')
```

```
for item in Unique(data1):
    print(item, end=' ')
print("\n")

print("--- Тест Unique с генератором ---")
data2_gen = gen_random(15, 1, 5)
data2_list = list(data2_gen)
print(f"Исходные данные (случайные): {data2_list}")
print("Уникальные:", end=' ')
for item in Unique(data2_list):
    print(item, end=' ')
print("\n")

print("--- Тест Unique со строками (ignore_case=False) ---")
data3 = ['a', 'A', 'b', 'B', 'a', 'A']
print(f"Исходные данные: {data3}")
print("Уникальные:", end=' ')
for item in Unique(data3, ignore_case=False):
    print(item, end=' ')
print("\n")

print("--- Тест Unique со строками (ignore_case=True) ---")
print(f"Исходные данные: {data3}")
print("Уникальные:", end=' ')
for item in Unique(data3, ignore_case=True):
    print(item, end=' ')
print()
```

Code Test

```
(.venv) gringo@gringos-MacBook-Pro Lab_3 % python3 gen_random.py
--- Тест gen_random (5 чисел от 1 до 3) ---
[3, 2, 2, 3, 2]

--- Тест gen_random (10 чисел от 50 до 100) ---
[76, 95, 57, 86, 60, 82, 93, 72, 71, 96]
(.venv) gringo@gringos-MacBook-Pro Lab_3 % python3 field.py
--- Тест field с одним аргументом ('title') ---
Ковер
Диван для отдыха

--- Тест field с одним аргументом ('price'), пропуская None ---
2000

--- Тест field с несколькими аргументами ('title', 'price', 'color') ---
{'title': 'Ковер', 'price': 2000, 'color': 'green'}
{'title': 'Диван для отдыха', 'color': 'black'}
(.venv) gringo@gringos-MacBook-Pro Lab_3 % python3 cm_timer.py
--- Тестирование cm_timer_1 (класс) ---
time: 1.2052

--- Тестирование cm_timer_2 (contextlib) ---
time: 1.8024
(.venv) gringo@gringos-MacBook-Pro Lab_3 % python3 unique.py
--- Тест Unique с числами ---
Исходные данные: [1, 1, 1, 2, 2, 3, 1, 2]
Уникальные: 1 2 3

--- Тест Unique с генератором ---
Исходные данные (случайные): [4, 5, 4, 5, 5, 4, 2, 1, 4, 1, 1, 5, 3, 3, 4]
Уникальные: 4 5 2 1 3

--- Тест Unique со строками (ignore_case=False) ---
Исходные данные: ['a', 'A', 'b', 'B', 'a', 'A']
Уникальные: a A b B

--- Тест Unique со строками (ignore_case=True) ---
Исходные данные: ['a', 'A', 'b', 'B', 'a', 'A']
Уникальные: a b
```

```
(.venv) gringo@gringos-MacBook-Pro Lab_3 % python3 print_result.py
--- Запуск тестов для декоратора print_result ---

test_1
1
--------------------
test_2
iu5
--------------------
test_3
a = 1
b = 2
--------------------
test_4
1
2

--- Тесты завершены ---
(.venv) gringo@gringos-MacBook-Pro Lab_3 % python3 process_data.py
--- Запуск цепочки обработки с тестовыми данными ---
f1
Аналитик данных
Инженер-программист
программист C++
Программист Java
Программист Python
f2
программист C++
Программист Java
Программист Python
f3
программист C++ с опытом Python
Программист Java с опытом Python
Программист Python с опытом Python
f4
программист C++ с опытом Python, зарплата 106979 руб.
Программист Java с опытом Python, зарплата 175556 руб.
Программист Python с опытом Python, зарплата 139117 руб.
time: 0.0001
--- Обработка завершена ---
```

```
(.venv) gringo@gringos-MacBook-Pro Lab_3 % python3 sort.py
Исходные данные: [4, -30, 100, -100, 123, 1, 0, -1, -4]

--- Сортировка по модулю (без lambda) ---
[123, 100, -100, -30, 4, -4, 1, -1, 0]

--- Сортировка по модулю (с lambda) ---
[123, 100, -100, -30, 4, -4, 1, -1, 0]
```