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  
“ Функциональные возможности языка Python”

**Выполнил:**Студент группы ИУ5Ц-54Б  
Цурин А.П.  
**Преподаватель:**Гапанюк Ю.Е.

Москва 2025

# Задания для выполнения

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.

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.

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.

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.

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.

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.

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
* 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 программы

**файл field.py**

from typing import List, Dict, Any, Union

def field(items: List[Dict], \*args: str) -> Union[List, List[Dict]]:

    """

    Генератор для выборки полей из словарей

    Args:

        items: Список словарей

        \*args: Названия полей для выборки

    Yields:

        Если передан один аргумент - значения поля

        Если несколько аргументов - словари с выбранными полями

    """

    if len(args) == 0:

        return

    for item in items:

        if not isinstance(item, dict):

            continue

        if len(args) == 1:

            field\_name = args[0]

            if field\_name in item and item[field\_name] is not None:

                yield item[field\_name]

        else:

            result = {}

            has\_valid\_fields = False

            for field\_name in args:

                if field\_name in item and item[field\_name] is not None:

                    result[field\_name] = item[field\_name]

                    has\_valid\_fields = True

            if has\_valid\_fields:

                yield result

**файл gen\_random.py**

import random

from typing import List

def gen\_random(num\_count: int, min\_value: int, max\_value: int) -> List[int]:

    """

    Генератор случайных чисел

    Args:

        num\_count: Количество чисел

        min\_value: Минимальное значение

        max\_value: Максимальное значение

    Returns:

        Список случайных чисел

    """

    return [random.randint(min\_value, max\_value) for \_ in range(num\_count)]

# Альтернативная реализация как генератор

def gen\_random\_generator(num\_count: int, min\_value: int, max\_value: int):

    """

    Генератор случайных чисел (реализация как генератор)

    """

    for \_ in range(num\_count):

        yield random.randint(min\_value, max\_value)

**файл unique.py**

from typing import List, Any, Callable

def Unique(arr: List[Any], ignore\_case: bool = False) -> List[Any]:

    """

    Генератор для получения уникальных элементов из списка

    Args:

        arr: Входной список

        ignore\_case: Игнорировать регистр для строк

    Yields:

        Уникальные элементы в порядке первого появления

    """

    seen = set()

    for item in arr:

        if ignore\_case and isinstance(item, str):

            key = item.lower()

        else:

            key = item

        if key not in seen:

            seen.add(key)

            yield item

# Альтернативная реализация как класс

class unique:

    def \_\_init\_\_(self, items, \*\*kwargs):

        self.items = iter(items)

        self.ignore\_case = kwargs.get('ignore\_case', False)

        self.seen = set()

    def \_\_iter\_\_(self):

        return self

    def \_\_next\_\_(self):

        while True:

            item = next(self.items)

            key = item.lower() if self.ignore\_case and isinstance(item, str) else item

            if key not in self.seen:

                self.seen.add(key)

                return item

**файл sort.py**

from typing import List, Any, Callable

def sort(arr: List[Any], key: Callable = None, reverse: bool = False) -> List[Any]:

    """

    Функция сортировки с использованием быстрой сортировки

    Args:

        arr: Список для сортировки

        key: Функция ключа для сравнения

        reverse: Обратный порядок сортировки

    Returns:

        Отсортированный список

    """

    if len(arr) <= 1:

        return arr

    pivot = arr[len(arr) // 2]

    if key:

        pivot\_val = key(pivot)

        left = [x for x in arr if key(x) < pivot\_val]

        middle = [x for x in arr if key(x) == pivot\_val]

        right = [x for x in arr if key(x) > pivot\_val]

    else:

        left = [x for x in arr if x < pivot]

        middle = [x for x in arr if x == pivot]

        right = [x for x in arr if x > pivot]

    result = sort(left, key, reverse) + middle + sort(right, key, reverse)

    return result if not reverse else result[::-1]

# Альтернативная реализация как класс

class sort\_by:

    def \_\_init\_\_(self, items, \*\*kwargs):

        self.key = kwargs.get('key', None)

        self.reverse = kwargs.get('reverse', False)

        self.sorted\_items = self.\_quicksort(list(items))

        self.index = 0

    def \_quicksort(self, arr):

        if len(arr) <= 1:

            return arr

        pivot = arr[len(arr) // 2]

        if self.key:

            pivot\_val = self.key(pivot)

            left = [x for x in arr if self.key(x) < pivot\_val]

            middle = [x for x in arr if self.key(x) == pivot\_val]

            right = [x for x in arr if self.key(x) > pivot\_val]

        else:

            left = [x for x in arr if x < pivot]

            middle = [x for x in arr if x == pivot]

            right = [x for x in arr if x > pivot]

        result = self.\_quicksort(left) + middle + self.\_quicksort(right)

        return result if not self.reverse else result[::-1]

    def \_\_iter\_\_(self):

        return self

    def \_\_next\_\_(self):

        if self.index < len(self.sorted\_items):

            result = self.sorted\_items[self.index]

            self.index += 1

            return result

        raise StopIteration

**файл print\_result.py**

from functools import wraps

import inspect

def print\_result(func):

    """

    Декоратор для вывода результата функции

    Выводит название функции и её результат в отформатированном виде

    """

    @wraps(func)

    def wrapper(\*args, \*\*kwargs):

        print(f"\nНазвание функции: {func.\_\_name\_\_}")

        # Получаем сигнатуру функции для красивого вывода аргументов

        sig = inspect.signature(func)

        bound\_args = sig.bind(\*args, \*\*kwargs)

        bound\_args.apply\_defaults()

        print("Аргументы:", dict(bound\_args.arguments))

        result = func(\*args, \*\*kwargs)

        print("Результат:")

        if isinstance(result, (list, tuple, set)):

            for item in result:

                print(f"  {item}")

        elif isinstance(result, dict):

            for key, value in result.items():

                print(f"  {key}: {value}")

        else:

            print(f"  {result}")

        return result

    return wrapper

**файл cm\_timer.py**

import time

from contextlib import contextmanager

class cm\_timer\_1:

    """

    Класс-контекстный менеджер для измерения времени выполнения

    """

    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"Время выполнения: {elapsed\_time:.4f} секунд")

@contextmanager

def cm\_timer\_2():

    """

    Функция-контекстный менеджер для измерения времени выполнения

    """

    start\_time = time.time()

    yield

    elapsed\_time = time.time() - start\_time

    print(f"Время выполнения: {elapsed\_time:.4f} секунд")

**файл process\_data.py**

from print\_result import print\_result

from field import field

from gen\_random import gen\_random

from cm\_timer import cm\_timer\_1

import json

import sys

# Пример данных по умолчанию

DEFAULT\_DATA = [

    {'name': 'Ksenia', 'age': 25, 'city': 'Moscow', 'salary': 50000},

    {'name': 'Ivan', 'age': 30, 'city': 'St Petersburg', 'salary': 60000},

    {'name': 'Maria', 'age': 28, 'city': 'Moscow', 'salary': 55000},

    {'name': 'Alexey', 'age': 35, 'city': 'Kazan', 'salary': 70000},

    {'name': 'Olga', 'age': 22, 'city': 'Moscow', 'salary': 45000},

    {'name': 'Anna', 'age': 29, 'city': 'Moscow', 'salary': 48000},

    {'name': 'Andrey', 'age': 32, 'city': 'St Petersburg', 'salary': 65000}

]

@print\_result

def f1(arg):

    """Получить уникальные имена, отсортированные по алфавиту"""

    return sorted(list(set(field(arg, 'name'))), key=lambda x: x.lower())

@print\_result

def f2(arg):

    """Отфильтровать имена, начинающиеся на 'А' или 'A'"""

    return list(filter(lambda x: x.lower().startswith('а'), arg))

@print\_result

def f3(arg):

    """Добавить фразу к каждому имени"""

    return list(map(lambda x: x + " смотрит в будущее", arg))

@print\_result

def f4(arg):

    """Сопоставить имена со случайными зарплатами"""

    return dict(zip(arg, gen\_random(len(arg), 100000, 200000)))

def main():

    # Инициализация данных

    data = DEFAULT\_DATA

    # Чтение данных из файла, если он указан

    if len(sys.argv) > 1:

        try:

            with open(sys.argv[1], 'r', encoding='utf-8') as f:

                data = json.load(f)

            print(f"Данные загружены из файла: {sys.argv[1]}")

        except FileNotFoundError:

            print(f"Файл {sys.argv[1]} не найден. Используются данные по умолчанию.")

        except json.JSONDecodeError:

            print(f"Ошибка чтения JSON из файла {sys.argv[1]}. Используются данные по умолчанию.")

        except Exception as e:

            print(f"Ошибка при загрузке файла: {e}. Используются данные по умолчанию.")

    else:

        print("Используются данные по умолчанию.")

    print(f"Всего записей: {len(data)}")

    # Выполнение пайплайна с замером времени

    with cm\_timer\_1():

        result = f4(f3(f2(f1(data))))

    return result

if \_\_name\_\_ == "\_\_main\_\_":

    main()

# Результаты работы программы

PS C:\Users\student\lab\_python\_fp> python.exe .\main.py

Лабораторная работа: Функциональное программирование в Python

============================================================

=== Тестирование unique ===

Исходные данные: [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]

Уникальные: [1, 2]

Исходные строки: ['a', 'A', 'b', 'B', 'a', 'A', 'c']

Уникальные (без ignore\_case): ['a', 'A', 'b', 'B', 'c']

Уникальные (с ignore\_case): ['a', 'b', 'c']

Класс unique: ['a', 'b', 'c']

=== Тестирование sort ===

Исходные данные: [5, 2, 8, 1, 9]

Отсортированные: [1, 2, 5, 8, 9]

Обратная сортировка: [9, 8, 1, 2, 5]

Исходные строки: ['banana', 'apple', 'cherry']

Отсортированные: ['apple', 'banana', 'cherry']

По длине: ['apple', 'banana', 'cherry']

Класс sort\_by: ['apple', 'banana', 'cherry']

=== Тестирование field ===

Только названия: ['Ковер', 'Диван для отдыха', 'Стул']

Названия и цены: [{'title': 'Ковер', 'price': 2000}, {'title': 'Диван для отдыха', 'price': 5300}, {'price': 1500}, {'title': 'Стул'}]

Все поля: [{'title': 'Ковер', 'price': 2000, 'color': 'green'}, {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'}, {'price': 1500, 'color': 'blue'}, {'title': 'Стул', 'color': 'white'}]

=== Тестирование gen\_random ===

5 случайных чисел от 1 до 10: [2, 10, 1, 9, 6]

3 случайных числа от 100 до 200: [135, 158, 130]

=== Тестирование print\_result ===

Название функции: test\_function

Аргументы: {'x': 2, 'y': 5}

Результат:

2

4

6

8

10

=== Тестирование cm\_timer ===

Время выполнения: 0.5005 секунд

Время выполнения: 0.3002 секунд

============================================================

Тестирование основного пайплайна:

============================================================

Используются данные по умолчанию.

Всего записей: 7

Название функции: f1

Аргументы: {'arg': [{'name': 'Ksenia', 'age': 25, 'city': 'Moscow', 'salary': 50000}, {'name': 'Ivan', 'age': 30, 'city': 'St Petersburg', 'salary': 60000}, {'name': 'Maria', 'age': 28, 'city': 'Moscow', 'salary': 55000}, {'name': 'Alexey', 'age': 35, 'city': 'Kazan', 'salary': 70000}, {'name': 'Olga', 'age': 22, 'city': 'Moscow', 'salary': 45000}, {'name': 'Anna', 'age': 29, 'city': 'Moscow', 'salary': 48000}, {'name': 'Andrey', 'age': 32, 'city': 'St Petersburg', 'salary': 65000}]}

Результат:

Alexey

Andrey

Anna

Ivan

Ksenia

Maria

Olga

Название функции: f2

Аргументы: {'arg': ['Alexey', 'Andrey', 'Anna', 'Ivan', 'Ksenia', 'Maria', 'Olga']}

Результат:

Название функции: f3

Аргументы: {'arg': []}

Результат:

Название функции: f4

Аргументы: {'arg': []}

Результат:

Время выполнения: 0.0021 секунд