МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
им. Н.Э. Баумана

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

A logo with a gold star and blue ribbon

Description automatically generated

Дисциплина «Парадигмы и конструкции языков программирования»

## Отчет по лабораторной работе №3-4 «Функциональные возможности языка Python»

**Выполнил:**Студент группы ИУ5-34Б  
Буш И. Е.  
**Преподаватель:**Нардид Анатолий Николаевич

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', 'price': 5300}**

**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, Aбв и АБВ - разные строки**

**# ignore\_case = False, Aбв 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:**

**С использованием lambda-функции.**

**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))))**

Листинг кода

Biquadratic.py

import math

class CoefficientError(Exception):

    pass

def read\_coefficient(value):

    try:

        return float(value)

    except ValueError:

        raise CoefficientError(f"Неверный формат коэффициента: {value}")

def solve\_biquadratic(a, b, c):

    if a == 0:

        raise CoefficientError("Коэффициент A не должен быть равен нулю.")

    d = b \*\* 2 - 4 \* a \* c

    if d < 0:

        return []

    y1 = (-b + math.sqrt(d)) / (2 \* a)

    y2 = (-b - math.sqrt(d)) / (2 \* a)

    roots = []

    for y in (y1, y2):

        if y > 0:

            roots.extend([math.sqrt(y), -math.sqrt(y)])

        elif y == 0:

            roots.append(0.0)

    return sorted(set(roots))

test\_bdd.py

import pytest

from lab3.biquadratic import solve\_biquadratic

def describe\_biquadratic\_solver():

    def test\_when\_discriminant\_positive():

        def it\_finds\_four\_roots():

            assert solve\_biquadratic(1, -5, 4) == [-2, -1, 1, 2]

    def test\_when\_discriminant\_negative():

        def it\_returns\_empty\_list():

            assert solve\_biquadratic(1, 2, 5) == []

    def test\_when\_root\_zero():

        def it\_contains\_zero():

            # x^4 + 0\*x^2 + 0 = x^4 = 0 → x = 0

            assert solve\_biquadratic(1, 0, 0) == [0.0]

from lab4.main import User

def test\_user\_creation\_bdd():

    # Given

    name = "Alice"

    email = "alice@example.com"

    age = 25

    # When

    user = User(name, email, age)

    # Then

    assert user.name == name

    assert user.email == email

    assert user.age == age

test\_tdd.py

import unittest

from lab3.biquadratic import solve\_biquadratic, read\_coefficient, CoefficientError

class TestBiquadraticTDD(unittest.TestCase):

    def test\_read\_coefficient\_valid(self):

        self.assertEqual(read\_coefficient("2.5"), 2.5)

    def test\_read\_coefficient\_invalid(self):

        with self.assertRaises(CoefficientError):

            read\_coefficient("abc")

    def test\_solve\_positive\_discriminant(self):

        # x^4 - 5x^2 + 4 = 0 → корни ±1, ±2

        roots = solve\_biquadratic(1, -5, 4)

        self.assertEqual(sorted(roots), [-2, -1, 1, 2])

    def test\_solve\_zero\_discriminant(self):

        # x^4 + 2x^2 + 1 = 0 → (x²+1)²=0 → нет действительных корней

        self.assertEqual(solve\_biquadratic(1, 2, 1), [])

    def test\_a\_equals\_zero(self):

        with self.assertRaises(CoefficientError):

            solve\_biquadratic(0, 1, 1)

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

    unittest.main()

import unittest

from lab4.main import User

class TestUser(unittest.TestCase):

    def test\_creation(self):

        user = User("Alice", "alice@example.com", 25)

        self.assertEqual(user.name, "Alice")

        self.assertEqual(user.email, "alice@example.com")

        self.assertEqual(user.age, 25)

    def test\_immutable(self):

        user = User("Alice", "alice@example.com")

        with self.assertRaises(AttributeError):

            user.name = "Bob"

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

    unittest.main()

test\_mock.py

import sys

from unittest.mock import patch

import pytest

from lab3.biquadratic import solve\_biquadratic, CoefficientError

from lab3.main import main

def test\_solve\_biquadratic\_a\_zero\_raises\_error():

    with pytest.raises(CoefficientError):

        solve\_biquadratic(0, 1, 1)

from unittest.mock import Mock

from lab4.main import User

class EmailService:

    def send\_email(self, user: User, message: str):

        print(f"Sending email to {user.email}: {message}")

def test\_email\_service\_mock():

    service = EmailService()

    service.send\_email = Mock()

    user = User("Alice", "alice@example.com")

    service.send\_email(user, "Welcome!")

    service.send\_email.assert\_called\_once\_with(user, "Welcome!")

Примеры работы программы

