**Оглавление**

[1 Дан текстовый файл, содержащий ключевые слова. Вывести список всех файлов в заданном каталоге, не содержащих ни одного ключевого слова. 3](#_Toc185990526)

[2 Дан текстовый файл, содержащий ключевые слова. Вывести список всех файлов в заданном каталоге, содержащих хотя бы одно из ключевых слов. 4](#_Toc185990527)

[3 Класс (структура) "Группа" содержит поля "Номер" "Курс" "ЧислоСтудентов". Разработать приложение для ведения файла студентов: изменение числа студентов, перевод на следующий курс, поиск по номеру группы. 7](#_Toc185990528)

[4 Класс (структура) "Студент" содержит поля "Фамилия" "Группа" "СтудБилет". Разработать приложение для ведения файла студентов: добавление, удаление, поиск по номеру билета. 10](#_Toc185990529)

[5 Разработать приложение для построения графика полинома n-й степени. Коэффициенты полинома считываются из PyQT QGridLayout или аналога 12](#_Toc185990530)

[6 Текстовый файл содержит слова, разделенные одним или несколькими пробелами. Разработать приложение для построения отсортированного словаря используемых слов. Вывести словарь в файл, используя tkinter filedialog или аналог. 14](#_Toc185990531)

[7 Разработать приложение для демонстрации работы с абстрактным типом данных "Стек". 16](#_Toc185990532)

[8 Разработать приложение для нахождения обратной матрицы для матрицы 3\*3. Коэффициенты матрицы считываются из PyQT QGridLayout или аналога. Коэффициенты обратной матрицы сохраняются при помощи PyQT QFileDialog или аналога. 19](#_Toc185990533)

[9 Разработать приложение для демонстрации работы с абстрактным типом данных "Дек" 20](#_Toc185990534)

[10 Разработать приложение для построения графика полинома n-й степени. Коэффициенты полинома считываются из PyQT QGridLayout или аналога 21](#_Toc185990535)

[11 Каталог содержит текстовые файлы с описанием компьютеров: производитель, быстродействие, оперативная память, внешняя память. Название каждого файла уникально и совпадает с гравировкой компьютера. Вывести информацию обо всех компьютерах в Excel-файл по указанной структуре, включая гравировку 23](#_Toc185990536)

[12 Разработать приложение для построения графика функции y=a\*sin(x)+b в координатных осях. Коэффициенты a и b считываются из текстового файла при помощи filedialog (tkinter) или аналога. 25](#_Toc185990537)

[13 Разработать приложение для демонстрации работы с абстрактным типом данных "Множество". 28](#_Toc185990538)

[14 Разработать приложение для демонстрации работы с абстрактным типом данных "Односвязный список". 29](#_Toc185990539)

[15 Разработать визуальное приложение «Таймер» с тремя кнопками «старт», «стоп», «сброс», которое бы показывало на экране прошедшее время с хранящегося момента времени до остановки. Кнопка «сброс» «сбрасывает» таймер 30](#_Toc185990540)

[16 В таблицах базы данных хранится информация о посещении студентами колледжа спортивных секций: фамилия, группа, секция, тренер, дата занятия. Провести нормализацию. Разработать оконное приложение для вывода в таблицу (grid) занятий группы 399ИС-22, отсортированных по дате 32](#_Toc185990541)

[17 Разработать программу с меню для ввода с клавиатуры целых чисел в массив (список), сортировки «обменом» и вывода в файл. Сортировку реализовать в виде функции в отдельном модуле. 34](#_Toc185990542)

[18 Разработать программу с меню для ввода из файла целых чисел в массив (список), сортировки «выбором» и вывода на экран. Сортировку реализовать в виде функции в отдельном модуле. 37](#_Toc185990543)

[19 В каждой строке текстового файла in.txt посещения библиотеки хранится запись: студент: дд.мм.гг, дд.мм.гг. … Сформировать файл out.txt, содержащий строки следующего вида: дд.мм.гг: Студент1, Студент2, … Использовать словари (хэш-таблицы). 38](#_Toc185990544)

[20 В базе данных хранятся сведения о заработной плате сотрудников отделов предприятия. Увеличить заработную плату сотрудникам отдела главного конструктора на 10%. Отобразить в табличном виде на экране 39](#_Toc185990545)

[21 Разработать программу для формирования в текущем каталоге системы каталогов. 41](#_Toc185990546)

[22 Разработать визуальное приложение с меню для выбора и шифрования текстового файла по методу Цезаря. 42](#_Toc185990547)

[23 Разработать программу для хранения полинома N-степени с использованием абстрактного типа данных «Односвязный список». 45](#_Toc185990548)

[24 Разработать рекурсивную функцию определения N-числа Фибоначчи. Программа должна печатать число рекурсивных вызовов 46](#_Toc185990549)

[25 Разработать функцию для печати суммы прописью до 1000000 рублей. Пример: для 123456 функция должна печатать Сто двадцать три тысяч четыреста пятьдесят шесть (рублей). 47](#_Toc185990550)

[26 Разработать класс Cart (Корзина). 47](#_Toc185990551)

[27 Разработать приложение с графическим интерфейсом для создания логина и пароля. 50](#_Toc185990552)

[28 Разработать программу для генерации последовательности случайных целых чисел в бинарном файле и сортировки файла. Саму сортировку реализовать в виде функции в отдельном модуле 52](#_Toc185990553)

# **1 Дан текстовый файл, содержащий ключевые слова. Вывести список всех файлов в заданном каталоге, не содержащих ни одного ключевого слова.**

import os  
  
  
def find\_files\_without\_keywords(directory: str, keywords\_file: str):  
 *"""  
 Возвращает список файлов в указанном каталоге, которые не содержат ни одного из ключевых слов.  
  
 :param directory: Каталог для поиска файлов.  
 :param keywords\_file: Путь к файлу с ключевыми словами.  
 :return: Список имен файлов, не содержащих ключевые слова.  
 """* # Чтение ключевых слов из файла  
 with open(keywords\_file, 'r', encoding='utf-8') as f:  
 keywords = [line.strip() for line in f.readlines()]  
  
 files\_without\_keywords = []  
  
 # Проход по всем файлам в указанной директории  
 for root, \_, files in os.walk(directory):  
 for file in files:  
 file\_path = os.path.join(root, file)  
  
 try:  
 # Чтение содержимого файла  
 with open(file\_path, 'r', encoding='utf-8') as f:  
 content = f.read()  
  
 # Проверка наличия хотя бы одного ключевого слова  
 if not any(keyword in content for keyword in keywords):  
 files\_without\_keywords.append(file\_path)  
 except (UnicodeDecodeError, IOError):  
 # Игнорируем файлы, которые не удалось прочитать  
 continue  
  
 return files\_without\_keywords  
  
  
# Пример использования  
directory\_to\_search = '/path/to/directory' # Укажите путь к каталогу  
keywords\_file\_path = '/path/to/keywords.txt' # Укажите путь к файлу с ключевыми словами  
  
result = find\_files\_without\_keywords(directory\_to\_search, keywords\_file\_path)  
print("Файлы, не содержащие ключевых слов:")  
for file in result:  
 print(file)

# **2 Дан текстовый файл, содержащий ключевые слова. Вывести список всех файлов в заданном каталоге, содержащих хотя бы одно из ключевых слов.**

**Пример содержания текстового файла с ключевыми словами (например, keywords.txt):**

python

data

algorithm

file

**Пример содержания файлов в каталоге:**

1. file1.txt: "This file contains python code for algorithm testing."
2. file2.txt: "This is a data file."
3. file3.txt: "Just some random text here."

**Код:**

import os  
  
# Функция для чтения ключевых слов из текстового файла  
def read\_keywords(file\_path):  
 with open(file\_path, 'r', encoding='utf-8') as file:  
 # Читаем все строки, удаляем лишние пробелы и возвращаем список ключевых слов  
 return [line.strip().lower() for line in file.readlines()]  
  
# Функция для проверки, содержит ли файл хотя бы одно из ключевых слов  
def contains\_keyword(file\_path, keywords):  
 try:  
 with open(file\_path, 'r', encoding='utf-8') as file:  
 content = file.read().lower() # Читаем содержимое файла в нижнем регистре для удобства  
 # Проверяем, содержит ли контент хотя бы одно из ключевых слов  
 for keyword in keywords:  
 if keyword in content:  
 return True # Если найдено хотя бы одно ключевое слово  
 except Exception as e:  
 print(f"Ошибка при чтении файла {file\_path}: {e}")  
 return False  
  
# Функция для поиска файлов с ключевыми словами в заданном каталоге  
def search\_files\_with\_keywords(directory, keywords):  
 matching\_files = [] # Список файлов, которые содержат хотя бы одно ключевое слово  
 for root, dirs, files in os.walk(directory): # Проходим по всем файлам в каталоге  
 for file\_name in files:  
 file\_path = os.path.join(root, file\_name) # Полный путь к файлу  
 if contains\_keyword(file\_path, keywords): # Проверяем файл на наличие ключевых слов  
 matching\_files.append(file\_path) # Добавляем файл в список  
 return matching\_files  
  
# Основная часть программы  
if \_\_name\_\_ == "\_\_main\_\_":  
 keywords\_file = 'keywords.txt' # Имя файла с ключевыми словами  
 directory\_to\_search = 'files' # Каталог для поиска файлов  
  
 # Шаг 1: Читаем ключевые слова из файла  
 keywords = read\_keywords(keywords\_file)  
  
 # Шаг 2: Ищем файлы, содержащие хотя бы одно ключевое слово  
 matching\_files = search\_files\_with\_keywords(directory\_to\_search, keywords)  
  
 # Шаг 3: Выводим результаты  
 if matching\_files:  
 print("Файлы, содержащие хотя бы одно из ключевых слов:")  
 for file in matching\_files:  
 print(file)  
 else:  
 print("Нет файлов, содержащих указанные ключевые слова.")

**Объяснение кода:**

1. **Чтение ключевых слов:** В функции read\_keywords мы открываем текстовый файл, читаем все строки и убираем лишние пробелы с помощью метода strip. Ключевые слова приводятся к нижнему регистру, чтобы поиск был регистронезависимым.
2. **Проверка содержимого файлов:** В функции contains\_keyword мы открываем файл и читаем его содержимое. Преобразуем текст в нижний регистр, чтобы избежать проблем с регистрами. Для каждого файла проверяем наличие хотя бы одного из ключевых слов.
3. **Поиск файлов в каталоге:** В функции search\_files\_with\_keywords мы используем os.walk, чтобы пройти по всем файлам в заданном каталоге. Для каждого файла вызываем функцию contains\_keyword и, если хотя бы одно ключевое слово найдено, добавляем файл в список.
4. **Вывод результатов:** После поиска выводим на экран список файлов, которые содержат хотя бы одно ключевое слово.

**Пример структуры каталогов и файлов:**

project/

│

├── keywords.txt

├── files/

│ ├── file1.txt

│ ├── file2.txt

│ └── file3.txt

**Пример содержания файла keywords.txt:**

python

data

algorithm

file

**Пример содержания файлов в каталоге files:**

* file1.txt: "This file contains python code for algorithm testing."
* file2.txt: "This is a data file."
* file3.txt: "Just some random text here."

**Ожидаемый вывод программы:**

Файлы, содержащие хотя бы одно из ключевых слов:

files/file1.txt

files/file2.txt

# **3 Класс (структура) "Группа" содержит поля "Номер" "Курс" "ЧислоСтудентов". Разработать приложение для ведения файла студентов: изменение числа студентов, перевод на следующий курс, поиск по номеру группы.**

import json  
  
class Group:  
 def \_\_init\_\_(self, number: int, course: int, num\_students: int):  
 self.number = number  
 self.course = course  
 self.num\_students = num\_students  
  
 def update\_students(self, num\_students: int):  
 *"""Изменяет количество студентов в группе."""* self.num\_students = num\_students  
  
 def next\_course(self):  
 *"""Переводит группу на следующий курс."""* self.course += 1  
  
 def to\_dict(self):  
 *"""Возвращает словарь с данными группы."""* return {  
 "number": self.number,  
 "course": self.course,  
 "num\_students": self.num\_students  
 }  
  
 @classmethod  
 def from\_dict(cls, data: dict):  
 *"""Создает объект Group из словаря."""* return cls(data["number"], data["course"], data["num\_students"])  
  
  
class GroupManager:  
 def \_\_init\_\_(self, file\_path: str):  
 self.file\_path = file\_path  
 self.groups = self.load\_groups()  
  
 def load\_groups(self):  
 *"""Загружает группы из файла."""* try:  
 with open(self.file\_path, 'r', encoding='utf-8') as f:  
 data = json.load(f)  
 return [Group.from\_dict(group) for group in data]  
 except FileNotFoundError:  
 return [] # Если файл не найден, возвращаем пустой список  
 except json.JSONDecodeError:  
 return [] # Если файл поврежден, возвращаем пустой список  
  
 def save\_groups(self):  
 *"""Сохраняет группы в файл."""* with open(self.file\_path, 'w', encoding='utf-8') as f:  
 json.dump([group.to\_dict() for group in self.groups], f, ensure\_ascii=False, indent=4)  
  
 def add\_group(self, number: int, course: int, num\_students: int):  
 *"""Добавляет новую группу."""* self.groups.append(Group(number, course, num\_students))  
 self.save\_groups()  
  
 def find\_group(self, number: int):  
 *"""Находит группу по номеру."""* for group in self.groups:  
 if group.number == number:  
 return group  
 return None  
  
 def update\_group\_students(self, number: int, num\_students: int):  
 *"""Изменяет количество студентов в группе."""* group = self.find\_group(number)  
 if group:  
 group.update\_students(num\_students)  
 self.save\_groups()  
 return True  
 return False  
  
 def promote\_group(self, number: int):  
 *"""Переводит группу на следующий курс."""* group = self.find\_group(number)  
 if group:  
 group.next\_course()  
 self.save\_groups()  
 return True  
 return False  
  
  
# Пример использования  
if \_\_name\_\_ == "\_\_main\_\_":  
 manager = GroupManager("groups.json")  
  
 while True:  
 print("\n1. Добавить группу")  
 print("2. Изменить количество студентов")  
 print("3. Перевести группу на следующий курс")  
 print("4. Найти группу по номеру")  
 print("5. Показать все группы")  
 print("6. Выход")  
  
 choice = input("Выберите действие: ")  
  
 if choice == "1":  
 number = input("Введите номер группы: ")  
 course = int(input("Введите курс: "))  
 num\_students = int(input("Введите число студентов: "))  
 manager.add\_group(number, course, num\_students)  
 print("Группа добавлена!")  
  
 elif choice == "2":  
 number = input("Введите номер группы: ")  
 num\_students = int(input("Введите новое количество студентов: "))  
 if manager.update\_group\_students(number, num\_students):  
 print("Количество студентов обновлено!")  
 else:  
 print("Группа не найдена!")  
  
 elif choice == "3":  
 number = input("Введите номер группы: ")  
 if manager.promote\_group(number):  
 print("Группа переведена на следующий курс!")  
 else:  
 print("Группа не найдена!")  
  
 elif choice == "4":  
 number = input("Введите номер группы: ")  
 group = manager.find\_group(number)  
 if group:  
 print(f"Группа {group.number}: курс {group.course}, студентов {group.num\_students}")  
 else:  
 print("Группа не найдена!")  
  
 elif choice == "5":  
 for group in manager.groups:  
 print(f"Группа {group.number}: курс {group.course}, студентов {group.num\_students}")  
  
 elif choice == "6":  
 print("Выход из программы.")  
 break  
  
 else:  
 print("Неверный выбор. Попробуйте снова.")

# **4 Класс (структура) "Студент" содержит поля "Фамилия" "Группа" "СтудБилет". Разработать приложение для ведения файла студентов: добавление, удаление, поиск по номеру билета.**

import json  
  
class Student:  
 def \_\_init\_\_(self, surname: str, group: int, ticket\_number: str):  
 self.surname = surname  
 self.group = group  
 self.ticket\_number = ticket\_number  
  
 def to\_dict(self):  
 *"""Возвращает словарь с данными студента."""* return {  
 "surname": self.surname,  
 "group": self.group,  
 "ticket\_number": self.ticket\_number  
 }  
  
 @classmethod  
 def from\_dict(cls, data: dict):  
 *"""Создает объект Student из словаря."""* return cls(data["surname"], data["group"], data["ticket\_number"])  
  
  
class StudentManager:  
 def \_\_init\_\_(self, file\_path: str):  
 self.file\_path = file\_path  
 self.students = self.load\_students()  
  
 def load\_students(self):  
 *"""Загружает список студентов из файла."""* try:  
 with open(self.file\_path, 'r', encoding='utf-8') as f:  
 data = json.load(f)  
 return [Student.from\_dict(student) for student in data]  
 except FileNotFoundError:  
 return []  
 except json.JSONDecodeError:  
 return []  
  
 def save\_students(self):  
 *"""Сохраняет список студентов в файл."""* with open(self.file\_path, 'w', encoding='utf-8') as f:  
 json.dump([student.to\_dict() for student in self.students], f, ensure\_ascii=False, indent=4)  
  
 def add\_student(self, surname: str, group: int, ticket\_number: str):  
 *"""Добавляет нового студента."""* self.students.append(Student(surname, group, ticket\_number))  
 self.save\_students()  
  
 def find\_student(self, ticket\_number: str):  
 *"""Ищет студента по номеру студенческого билета."""* for student in self.students:  
 if student.ticket\_number == ticket\_number:  
 return student  
 return None  
  
 def delete\_student(self, ticket\_number: str):  
 *"""Удаляет студента по номеру студенческого билета."""* student = self.find\_student(ticket\_number)  
 if student:  
 self.students.remove(student)  
 self.save\_students()  
 return True  
 return False  
  
  
# Пример использования  
if \_\_name\_\_ == "\_\_main\_\_":  
 manager = StudentManager("students.json")  
  
 while True:  
 print("\n1. Добавить студента")  
 print("2. Найти студента по номеру студенческого билета")  
 print("3. Удалить студента по номеру студенческого билета")  
 print("4. Показать всех студентов")  
 print("5. Выход")  
  
 choice = input("Выберите действие: ")  
  
 if choice == "1":  
 surname = input("Введите фамилию студента: ")  
 group = int(input("Введите номер группы: "))  
 ticket\_number = input("Введите номер студенческого билета: ")  
 manager.add\_student(surname, group, ticket\_number)  
 print("Студент добавлен!")  
  
 elif choice == "2":  
 ticket\_number = input("Введите номер студенческого билета: ")  
 student = manager.find\_student(ticket\_number)  
 if student:  
 print(f"Студент: {student.surname}, группа: {student.group}, билет: {student.ticket\_number}")  
 else:  
 print("Студент не найден!")  
  
 elif choice == "3":  
 ticket\_number = input("Введите номер студенческого билета: ")  
 if manager.delete\_student(ticket\_number):  
 print("Студент удален!")  
 else:  
 print("Студент не найден!")  
  
 elif choice == "4":  
 for student in manager.students:  
 print(f"{student.surname}, группа {student.group}, билет {student.ticket\_number}")  
  
 elif choice == "5":  
 print("Выход из программы.")  
 break  
  
 else:  
 print("Неверный выбор. Попробуйте снова.")

# **5 Разработать приложение для построения графика полинома n-й степени. Коэффициенты полинома считываются из PyQT QGridLayout или аналога**

import sys  
import numpy as np  
from PyQt5.QtWidgets import (  
 QApplication, QMainWindow, QWidget, QGridLayout, QLabel, QLineEdit, QPushButton  
)  
from matplotlib.backends.backend\_qt5agg import (  
 FigureCanvasQTAgg as FigureCanvas  
)  
from matplotlib.figure import Figure  
  
  
class PolynomialPlotter(QMainWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle("График полинома")  
 self.setGeometry(100, 100, 600, 500)  
  
 # Основной виджет  
 self.main\_widget = QWidget()  
 self.setCentralWidget(self.main\_widget)  
  
 # Сетка для интерфейса  
 self.grid\_layout = QGridLayout(self.main\_widget)  
  
 # Поля для ввода коэффициентов  
 self.inputs = []  
 self.degree\_label = QLabel("Степень полинома:")  
 self.grid\_layout.addWidget(self.degree\_label, 0, 0)  
 self.degree\_input = QLineEdit("2")  
 self.grid\_layout.addWidget(self.degree\_input, 0, 1)  
  
 # Кнопка для генерации полей ввода  
 self.generate\_button = QPushButton("Создать поля ввода")  
 self.generate\_button.clicked.connect(self.create\_input\_fields)  
 self.grid\_layout.addWidget(self.generate\_button, 1, 0, 1, 2)  
  
 # Кнопка для построения графика  
 self.plot\_button = QPushButton("Построить график")  
 self.plot\_button.clicked.connect(self.plot\_graph)  
 self.grid\_layout.addWidget(self.plot\_button, 2, 0, 1, 2)  
  
 # Поле для графика  
 self.figure = Figure()  
 self.canvas = FigureCanvas(self.figure)  
 self.grid\_layout.addWidget(self.canvas, 3, 0, 1, 2)  
  
 def create\_input\_fields(self):  
 *"""Создаёт поля ввода для коэффициентов полинома."""* # Очистка текущих полей  
 for input\_field in self.inputs:  
 self.grid\_layout.removeWidget(input\_field)  
 input\_field.deleteLater()  
 self.inputs = []  
  
 # Создание новых полей ввода  
 try:  
 degree = int(self.degree\_input.text())  
 if degree < 0:  
 raise ValueError("Степень должна быть неотрицательной.")  
 except ValueError:  
 self.degree\_input.setText("2")  
 degree = 2  
  
 for i in range(degree + 1):  
 label = QLabel(f"a{i}:")  
 input\_field = QLineEdit()  
 self.grid\_layout.addWidget(label, 4 + i, 0)  
 self.grid\_layout.addWidget(input\_field, 4 + i, 1)  
 self.inputs.append(input\_field)  
  
 def plot\_graph(self):  
 *"""Читает коэффициенты и строит график полинома."""* try:  
 coefficients = [  
 float(field.text()) if field.text() else 0  
 for field in self.inputs  
 ]  
 except ValueError:  
 print("Ошибка: все коэффициенты должны быть числами.")  
 return  
  
 # Построение графика  
 x = np.linspace(-10, 10, 500)  
 y = np.polyval(coefficients, x)  
  
 self.figure.clear()  
 ax = self.figure.add\_subplot(111)  
 ax.plot(x, y, label="Полином")  
 ax.axhline(0, color="black", linewidth=0.8, linestyle="--")  
 ax.axvline(0, color="black", linewidth=0.8, linestyle="--")  
 ax.set\_title("График полинома")  
 ax.set\_xlabel("x")  
 ax.set\_ylabel("y")  
 ax.legend()  
 self.canvas.draw()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Создание приложения  
 app = QApplication(sys.argv)  
 window = PolynomialPlotter()  
 window.show()  
 # Запуск приложения  
 sys.exit(app.exec\_())

# **6 Текстовый файл содержит слова, разделенные одним или несколькими пробелами. Разработать приложение для построения отсортированного словаря используемых слов. Вывести словарь в файл, используя tkinter filedialog или аналог.**

import tkinter as tk  
from tkinter import filedialog  
from tkinter import messagebox  
  
  
def load\_text\_file():  
 *"""Загружает текстовый файл с использованием диалогового окна."""* file\_path = filedialog.askopenfilename(title="Выберите текстовый файл",  
 filetypes=(("Текстовые файлы", "\*.txt"), ("Все файлы", "\*.\*")))  
 if not file\_path:  
 return None  
 try:  
 with open(file\_path, 'r', encoding='utf-8') as file:  
 return file.read()  
 except IOError:  
 messagebox.showerror("Ошибка", f"Не удалось открыть файл: {file\_path}")  
 return None  
  
  
def save\_text\_file(content):  
 *"""Сохраняет текстовый файл с использованием диалогового окна."""* file\_path = filedialog.asksaveasfilename(title="Сохранить словарь в файл",  
 defaultextension=".txt",  
 filetypes=(("Текстовые файлы", "\*.txt"), ("Все файлы", "\*.\*")))  
 if not file\_path:  
 return  
 try:  
 with open(file\_path, 'w', encoding='utf-8') as file:  
 file.write(content)  
 messagebox.showinfo("Успех", f"Словарь сохранен в файл: {file\_path}")  
 except IOError:  
 messagebox.showerror("Ошибка", f"Не удалось сохранить файл: {file\_path}")  
  
  
def process\_file():  
 *"""Обрабатывает текстовый файл и сохраняет отсортированный словарь."""* content = load\_text\_file()  
 if content is None:  
 return  
  
 # Разделяем слова, удаляем лишние пробелы и сортируем  
 words = content.split()  
 unique\_words = sorted(set(words), key=str.lower) # Сортировка без учета регистра  
  
 # Формируем строку для сохранения  
 result = "\n".join(unique\_words)  
 save\_text\_file(result)  
  
  
def main():  
 *"""Основная функция для запуска приложения."""* # Создаем окно  
 root = tk.Tk()  
 root.title("Словарь слов из файла")  
  
 # Создаем интерфейс  
 label = tk.Label(root, text="Нажмите кнопку для обработки текстового файла")  
 label.pack(pady=10)  
  
 process\_button = tk.Button(root, text="Обработать файл", command=process\_file)  
 process\_button.pack(pady=10)  
  
 # Запуск интерфейса  
 root.mainloop()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

# **7 Разработать приложение для демонстрации работы с абстрактным типом данных "Стек".**

**Описание приложения:**

1. Мы реализуем класс Stack, который будет хранить элементы стека и обеспечивать основные операции, такие как:
   * **push** — добавление элемента в стек.
   * **pop** — удаление и возврат верхнего элемента.
   * **peek** — просмотр верхнего элемента без его удаления.
   * **is\_empty** — проверка, пуст ли стек.
   * **size** — возвращение размера стека.
2. Для взаимодействия с пользователем создадим текстовое меню с возможностью выбора действий.

**Алгоритм:**

1. Реализуем класс Stack.
2. Создадим функции для работы с пользователем через текстовое меню.
3. Программа должна позволять пользователю:
   * Добавлять элемент в стек.
   * Удалять верхний элемент стека.
   * Просматривать верхний элемент стека.
   * Проверять, пуст ли стек.
   * Выходить из программы.

**Код:**

class Stack:  
 def \_\_init\_\_(self):  
 *"""Инициализация стека"""* self.stack = [] # Стек будет храниться в виде списка  
  
 def push(self, item):  
 *"""Добавить элемент в стек"""* self.stack.append(item)  
  
 def pop(self):  
 *"""Удалить верхний элемент из стека и вернуть его"""* if not self.is\_empty():  
 return self.stack.pop()  
 else:  
 return "Стек пуст!"  
  
 def peek(self):  
 *"""Вернуть верхний элемент стека без удаления"""* if not self.is\_empty():  
 return self.stack[-1]  
 else:  
 return "Стек пуст!"  
  
 def is\_empty(self):  
 *"""Проверить, пуст ли стек"""* return len(self.stack) == 0  
  
 def size(self):  
 *"""Возвращает размер стека"""* return len(self.stack)  
  
  
# Функция для отображения меню и взаимодействия с пользователем  
def menu():  
 stack = Stack() # Создаем экземпляр стека  
 while True:  
 print("\nМеню:")  
 print("1. Добавить элемент в стек")  
 print("2. Удалить верхний элемент из стека")  
 print("3. Просмотреть верхний элемент стека")  
 print("4. Проверить, пуст ли стек")  
 print("5. Размер стека")  
 print("6. Выйти")  
  
 # Ввод действия пользователя  
 choice = input("Выберите действие (1-6): ")  
  
 if choice == '1':  
 item = input("Введите элемент для добавления в стек: ")  
 stack.push(item)  
 print(f"Элемент '{item}' добавлен в стек.")  
 elif choice == '2':  
 popped\_item = stack.pop()  
 print(f"Удалённый элемент: {popped\_item}")  
 elif choice == '3':  
 top\_item = stack.peek()  
 print(f"Верхний элемент стека: {top\_item}")  
 elif choice == '4':  
 if stack.is\_empty():  
 print("Стек пуст!")  
 else:  
 print("Стек не пуст!")  
 elif choice == '5':  
 print(f"Размер стека: {stack.size()}")  
 elif choice == '6':  
 print("Выход из программы.")  
 break  
 else:  
 print("Неверный ввод. Пожалуйста, выберите действительное действие.")  
  
# Основная функция запуска программы  
if \_\_name\_\_ == "\_\_main\_\_":  
 menu()

**Объяснение кода:**

1. **Класс Stack:**
   * Метод \_\_init\_\_: инициализирует пустой список, который будет представлять наш стек.
   * Метод push: добавляет элемент в стек с помощью метода append.
   * Метод pop: удаляет и возвращает верхний элемент стека, если стек не пуст.
   * Метод peek: возвращает верхний элемент стека без удаления.
   * Метод is\_empty: проверяет, пуст ли стек.
   * Метод size: возвращает количество элементов в стеке.
2. **Функция menu:** создает цикл с текстовым меню, где пользователь может выбрать действие (добавление элемента в стек, удаление элемента, просмотр верхнего элемента, проверка, пуст ли стек, и выход).
3. **Основная часть программы:** запускает функцию menu, которая инициализирует объект стека и обрабатывает пользовательский ввод.

# **8 Разработать приложение для нахождения обратной матрицы для матрицы 3\*3. Коэффициенты матрицы считываются из PyQT QGridLayout или аналога. Коэффициенты обратной матрицы сохраняются при помощи PyQT QFileDialog или аналога.**

import sys  
import numpy as np  
from PyQt5.QtWidgets import (  
 QApplication, QWidget, QGridLayout, QPushButton, QLabel,  
 QFileDialog, QMessageBox  
)  
  
class MatrixApp(QWidget):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.initUI()  
  
 def initUI(self):  
 self.setWindowTitle('Обратная матрица 3x3')  
 self.layout = QGridLayout()  
  
 # Создание полей для ввода коэффициентов матрицы  
 self.entries = [[None for \_ in range(3)] for \_ in range(3)]  
 for i in range(3):  
 for j in range(3):  
 self.entries[i][j] = QLabel(f'[{i+1},{j+1}]:')  
 self.layout.addWidget(self.entries[i][j], i, j \* 2)  
 self.entries[i][j] = QLineEdit()  
 self.layout.addWidget(self.entries[i][j], i, j \* 2 + 1)  
  
 # Кнопка для вычисления обратной матрицы  
 self.calc\_button = QPushButton('Найти обратную матрицу')  
 self.calc\_button.clicked.connect(self.calculate\_inverse)  
 self.layout.addWidget(self.calc\_button, 3, 0, 1, 2)  
  
 # Кнопка для сохранения результата  
 self.save\_button = QPushButton('Сохранить результат')  
 self.save\_button.clicked.connect(self.save\_result)  
 self.layout.addWidget(self.save\_button, 3, 2)  
  
 self.setLayout(self.layout)  
  
 def calculate\_inverse(self):  
 try:  
 # Считывание значений из полей ввода  
 matrix = np.array([[float(self.entries[i][j].text()) for j in range(3)] for i in range(3)])  
 inverse\_matrix = np.linalg.inv(matrix)  
  
 # Отображение результата  
 result\_str = "Обратная матрица:\n" + str(inverse\_matrix)  
 QMessageBox.information(self, 'Результат', result\_str)  
 self.inverse\_matrix = inverse\_matrix # Сохраняем обратную матрицу для дальнейшего использования  
 except np.linalg.LinAlgError:  
 QMessageBox.warning(self, 'Ошибка', 'Матрица вырождена, обратная матрица не существует.')  
 except ValueError:  
 QMessageBox.warning(self, 'Ошибка', 'Пожалуйста, введите корректные числовые значения.')  
  
 def save\_result(self):  
 if hasattr(self, 'inverse\_matrix'):  
 options = QFileDialog.Options()  
 file\_name, \_ = QFileDialog.getSaveFileName(self, "Сохранить файл", "", "Text Files (\*.txt);;All Files (\*)", options=options)  
 if file\_name:  
 np.savetxt(file\_name, self.inverse\_matrix, fmt='%.4f')  
 QMessageBox.information(self, 'Успех', 'Обратная матрица успешно сохранена.')  
 else:  
 QMessageBox.warning(self, 'Ошибка', 'Сначала найдите обратную матрицу.')  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app = QApplication(sys.argv)  
 ex = MatrixApp()  
 ex.show()  
 sys.exit(app.exec\_())

# **9 Разработать приложение для демонстрации работы с абстрактным типом данных "Дек"**

from collections import deque  
  
class DequeDemo:  
 def \_\_init\_\_(self):  
 self.deque = deque()  
  
 def add\_to\_front(self, item):  
 *"""Добавляет элемент в начало дека."""* self.deque.appendleft(item)  
 print(f"Элемент {item} добавлен в начало.")  
  
 def add\_to\_rear(self, item):  
 *"""Добавляет элемент в конец дека."""* self.deque.append(item)  
 print(f"Элемент {item} добавлен в конец.")  
  
 def remove\_from\_front(self):  
 *"""Удаляет элемент из начала дека."""* if self.is\_empty():  
 print("Дек пуст. Нечего удалять.")  
 else:  
 item = self.deque.popleft()  
 print(f"Элемент {item} удалён из начала.")  
  
 def remove\_from\_rear(self):  
 *"""Удаляет элемент из конца дека."""* if self.is\_empty():  
 print("Дек пуст. Нечего удалять.")  
 else:  
 item = self.deque.pop()  
 print(f"Элемент {item} удалён из конца.")  
  
 def peek\_front(self):  
 *"""Возвращает элемент из начала дека без удаления."""* if self.is\_empty():  
 print("Дек пуст. Нечего просматривать.")  
 else:  
 print(f"Элемент на начале: {self.deque[0]}")  
  
 def peek\_rear(self):  
 *"""Возвращает элемент из конца дека без удаления."""* if self.is\_empty():  
 print("Дек пуст. Нечего просматривать.")  
 else:  
 print(f"Элемент на конце: {self.deque[-1]}")  
  
 def is\_empty(self):  
 *"""Проверяет, пуст ли дек."""* return len(self.deque) == 0  
  
 def size(self):  
 *"""Возвращает размер дека."""* print(f"Размер дека: {len(self.deque)}")  
  
 def display(self):  
 *"""Отображает содержимое дека."""* print(f"Содержимое дека: {list(self.deque)}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 deque\_app = DequeDemo()  
  
 while True:  
 print("\nМеню:")  
 print("1. Добавить в начало")  
 print("2. Добавить в конец")  
 print("3. Удалить из начала")  
 print("4. Удалить из конца")  
 print("5. Просмотреть начало")  
 print("6. Просмотреть конец")  
 print("7. Проверить, пуст ли дек")  
 print("8. Узнать размер дека")  
 print("9. Показать содержимое дека")  
 print("10. Выйти")  
  
 choice = input("Выберите действие: ")  
  
 if choice == "1":  
 item = input("Введите элемент: ")  
 deque\_app.add\_to\_front(item)  
 elif choice == "2":  
 item = input("Введите элемент: ")  
 deque\_app.add\_to\_rear(item)  
 elif choice == "3":  
 deque\_app.remove\_from\_front()  
 elif choice == "4":  
 deque\_app.remove\_from\_rear()  
 elif choice == "5":  
 deque\_app.peek\_front()  
 elif choice == "6":  
 deque\_app.peek\_rear()  
 elif choice == "7":  
 if deque\_app.is\_empty():  
 print("Дек пуст.")  
 else:  
 print("Дек не пуст.")  
 elif choice == "8":  
 deque\_app.size()  
 elif choice == "9":  
 deque\_app.display()  
 elif choice == "10":  
 break  
 else:  
 print("Неверный выбор. Попробуйте снова.")

# **10 Разработать приложение для построения графика полинома n-й степени. Коэффициенты полинома считываются из PyQT QGridLayout или аналога**

import sys  
import numpy as np  
from PyQt5.QtWidgets import (  
 QApplication, QMainWindow, QWidget, QGridLayout, QLabel, QLineEdit, QPushButton  
)  
from matplotlib.backends.backend\_qt5agg import (  
 FigureCanvasQTAgg as FigureCanvas  
)  
from matplotlib.figure import Figure  
  
  
class PolynomialPlotter(QMainWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle("График полинома")  
 self.setGeometry(100, 100, 600, 500)  
  
 # Основной виджет  
 self.main\_widget = QWidget()  
 self.setCentralWidget(self.main\_widget)  
  
 # Сетка для интерфейса  
 self.grid\_layout = QGridLayout(self.main\_widget)  
  
 # Поля для ввода коэффициентов  
 self.inputs = []  
 self.degree\_label = QLabel("Степень полинома:")  
 self.grid\_layout.addWidget(self.degree\_label, 0, 0)  
 self.degree\_input = QLineEdit("2")  
 self.grid\_layout.addWidget(self.degree\_input, 0, 1)  
  
 # Кнопка для генерации полей ввода  
 self.generate\_button = QPushButton("Создать поля ввода")  
 self.generate\_button.clicked.connect(self.create\_input\_fields)  
 self.grid\_layout.addWidget(self.generate\_button, 1, 0, 1, 2)  
  
 # Кнопка для построения графика  
 self.plot\_button = QPushButton("Построить график")  
 self.plot\_button.clicked.connect(self.plot\_graph)  
 self.grid\_layout.addWidget(self.plot\_button, 2, 0, 1, 2)  
  
 # Поле для графика  
 self.figure = Figure()  
 self.canvas = FigureCanvas(self.figure)  
 self.grid\_layout.addWidget(self.canvas, 3, 0, 1, 2)  
  
 def create\_input\_fields(self):  
 *"""Создаёт поля ввода для коэффициентов полинома."""* # Очистка текущих полей  
 for input\_field in self.inputs:  
 self.grid\_layout.removeWidget(input\_field)  
 input\_field.deleteLater()  
 self.inputs = []  
  
 # Создание новых полей ввода  
 try:  
 degree = int(self.degree\_input.text())  
 if degree < 0:  
 raise ValueError("Степень должна быть неотрицательной.")  
 except ValueError:  
 self.degree\_input.setText("2")  
 degree = 2  
  
 for i in range(degree + 1):  
 label = QLabel(f"a{i}:")  
 input\_field = QLineEdit()  
 self.grid\_layout.addWidget(label, 4 + i, 0)  
 self.grid\_layout.addWidget(input\_field, 4 + i, 1)  
 self.inputs.append(input\_field)  
  
 def plot\_graph(self):  
 *"""Читает коэффициенты и строит график полинома."""* try:  
 coefficients = [  
 float(field.text()) if field.text() else 0  
 for field in self.inputs  
 ]  
 except ValueError:  
 print("Ошибка: все коэффициенты должны быть числами.")  
 return  
  
 # Построение графика  
 x = np.linspace(-10, 10, 500)  
 y = np.polyval(coefficients, x)  
  
 self.figure.clear()  
 ax = self.figure.add\_subplot(111)  
 ax.plot(x, y, label="Полином")  
 ax.axhline(0, color="black", linewidth=0.8, linestyle="--")  
 ax.axvline(0, color="black", linewidth=0.8, linestyle="--")  
 ax.set\_title("График полинома")  
 ax.set\_xlabel("x")  
 ax.set\_ylabel("y")  
 ax.legend()  
 self.canvas.draw()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Создание приложения  
 app = QApplication(sys.argv)  
 window = PolynomialPlotter()  
 window.show()  
 # Запуск приложения  
 sys.exit(app.exec\_())

# **11 Каталог содержит текстовые файлы с описанием компьютеров: производитель, быстродействие, оперативная память, внешняя память. Название каждого файла уникально и совпадает с гравировкой компьютера. Вывести информацию обо всех компьютерах в Excel-файл по указанной структуре, включая гравировку**

import os  
from openpyxl import Workbook  
  
  
def read\_computer\_data(directory: str):  
 *"""  
 Считывает информацию о компьютерах из текстовых файлов в указанном каталоге.  
  
 :param directory: Каталог с текстовыми файлами.  
 :return: Список словарей с данными о компьютерах.  
 """* computer\_data = []  
  
 for filename in os.listdir(directory):  
 file\_path = os.path.join(directory, filename)  
  
 # Проверяем, что файл текстовый  
 if not os.path.isfile(file\_path) or not filename.endswith('.txt'):  
 continue  
  
 computer\_info = {"Гравировка": filename}  
 try:  
 with open(file\_path, 'r', encoding='utf-8') as file:  
 for line in file:  
 # Ожидаем формат "ключ: значение"  
 if ':' in line:  
 key, value = map(str.strip, line.split(':', 1))  
 computer\_info[key] = value  
 except IOError:  
 print(f"Ошибка чтения файла: {file\_path}")  
  
 computer\_data.append(computer\_info)  
  
 return computer\_data  
  
  
def write\_to\_excel(data, output\_file):  
 *"""  
 Записывает данные о компьютерах в Excel файл.  
  
 :param data: Список словарей с данными о компьютерах.  
 :param output\_file: Путь для сохранения Excel файла.  
 """* wb = Workbook()  
 ws = wb.active  
 ws.title = "Компьютеры"  
  
 # Заголовки  
 headers = ["Гравировка", "Производитель", "Быстродействие", "Оперативная память", "Внешняя память"]  
 ws.append(headers)  
  
 # Заполнение данных  
 for computer in data:  
 row = [  
 computer.get("Гравировка", ""),  
 computer.get("Производитель", ""),  
 computer.get("Быстродействие", ""),  
 computer.get("Оперативная память", ""),  
 computer.get("Внешняя память", ""),  
 ]  
 ws.append(row)  
  
 # Сохранение файла  
 wb.save(output\_file)  
 print(f"Данные успешно сохранены в {output\_file}")  
  
  
def main():  
 # Укажите путь к каталогу с текстовыми файлами  
 directory = input("Введите путь к каталогу с текстовыми файлами: ").strip()  
  
 # Проверяем существование каталога  
 if not os.path.exists(directory):  
 print("Указанный каталог не существует.")  
 return  
  
 # Считываем данные  
 computer\_data = read\_computer\_data(directory)  
  
 # Указываем имя Excel файла  
 output\_file = "computers.xlsx"  
 write\_to\_excel(computer\_data, output\_file)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

# **12 Разработать приложение для построения графика функции y=a\*sin(x)+b в координатных осях. Коэффициенты a и b считываются из текстового файла при помощи filedialog (tkinter) или аналога.**

**Алгоритм:**

1. Создадим графический интерфейс с использованием Tkinter.
2. Будем использовать диалоговое окно для выбора текстового файла с коэффициентами aa и bb.
3. Прочитаем коэффициенты из файла и построим график функции y=a⋅sin⁡(x)+by = a \cdot \sin(x) + b.
4. Для построения графика будем использовать библиотеку **matplotlib**.

**Код:**

import tkinter as tk  
from tkinter import filedialog  
import matplotlib.pyplot as plt  
import numpy as np  
  
def read\_coefficients\_from\_file(file\_path):  
 *"""Чтение коэффициентов a и b из файла"""* try:  
 with open(file\_path, 'r') as file:  
 # Предположим, что файл содержит два числа, разделенные пробелом или новой строкой  
 a, b = map(float, file.read().split())  
 return a, b  
 except Exception as e:  
 print(f"Ошибка при чтении файла: {e}")  
 return None, None  
  
def plot\_graph(a, b):  
 *"""Построение графика функции y = a\*sin(x) + b"""* # Генерация данных для x от -10 до 10 с шагом 0.1  
 x = np.linspace(-10, 10, 400)  
 y = a \* np.sin(x) + b # Вычисление y по формуле  
  
 # Построение графика  
 plt.plot(x, y)  
 plt.title(f"График функции: y = {a} \* sin(x) + {b}")  
 plt.xlabel("x")  
 plt.ylabel("y")  
 plt.grid(True)  
 plt.show()  
  
def open\_file():  
 *"""Открытие файла с коэффициентами и построение графика"""* # Открытие диалогового окна для выбора файла  
 file\_path = filedialog.askopenfilename(title="Выберите файл с коэффициентами", filetypes=[("Text files", "\*.txt")])  
 if file\_path:  
 # Чтение коэффициентов из выбранного файла  
 a, b = read\_coefficients\_from\_file(file\_path)  
 if a is not None and b is not None:  
 plot\_graph(a, b)  
  
# Создание главного окна  
root = tk.Tk()  
root.title("Построение графика функции y = a \* sin(x) + b")  
  
# Кнопка для выбора файла и построения графика  
button = tk.Button(root, text="Выбрать файл с коэффициентами", command=open\_file)  
button.pack(pady=20)  
  
# Запуск приложения  
root.mainloop()

**Объяснение кода:**

1. **Чтение коэффициентов из файла**:
   * Функция read\_coefficients\_from\_file открывает выбранный файл, читает значения aa и bb и возвращает их как числа с плавающей точкой. Мы предполагаем, что в файле содержатся два числа, разделенные пробелом или новой строкой.
2. **Построение графика**:
   * Функция plot\_graph использует библиотеку **matplotlib** для построения графика функции y=a⋅sin⁡(x)+by = a \cdot \sin(x) + b.
   * Мы генерируем массив xx от -10 до 10 и вычисляем соответствующие значения yy, используя формулу для синусоидальной функции.
3. **Графический интерфейс с Tkinter**:
   * В приложении используется библиотека **Tkinter** для создания окна с кнопкой, которая вызывает диалоговое окно для выбора файла.
   * Функция open\_file открывает диалоговое окно с помощью filedialog.askopenfilename, после чего считывает коэффициенты из выбранного файла и строит график.
4. **Запуск программы**:
   * Программа запускается через метод root.mainloop(), который поддерживает отображение графического интерфейса и обработку пользовательского ввода.

**Пример содержания текстового файла:**

Пример файла coefficients.txt:

2.5 1.0

где:

* 2.5 — это коэффициент aa,
* 1.0 — это коэффициент bb.

**Как использовать:**

1. Запустите программу. Откроется окно с кнопкой.
2. Нажмите кнопку "Выбрать файл с коэффициентами" и выберите текстовый файл с коэффициентами aa и bb.
3. После выбора файла будет построен график функции y=a⋅sin⁡(x)+by = a \cdot \sin(x) + b, где aa и bb считываются из файла.

# **13 Разработать приложение для демонстрации работы с абстрактным типом данных "Множество".**

import sys  
from PyQt5.QtWidgets import (  
 QApplication, QWidget, QVBoxLayout, QHBoxLayout,  
 QPushButton, QLabel, QLineEdit, QListWidget, QMessageBox  
)  
  
class SetApp(QWidget):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.initUI()  
 self.set\_data = set() # Инициализация множества  
  
 def initUI(self):  
 self.setWindowTitle('Демонстрация работы с множеством')  
 self.layout = QVBoxLayout()  
  
 # Поле для ввода элемента  
 self.input\_field = QLineEdit(self)  
 self.input\_field.setPlaceholderText('Введите элемент для добавления/удаления')  
 self.layout.addWidget(self.input\_field)  
  
 # Кнопки для операций  
 self.add\_button = QPushButton('Добавить', self)  
 self.add\_button.clicked.connect(self.add\_element)  
 self.layout.addWidget(self.add\_button)  
  
 self.remove\_button = QPushButton('Удалить', self)  
 self.remove\_button.clicked.connect(self.remove\_element)  
 self.layout.addWidget(self.remove\_button)  
  
 # Кнопка для отображения элементов множества  
 self.show\_button = QPushButton('Показать множество', self)  
 self.show\_button.clicked.connect(self.show\_set)  
 self.layout.addWidget(self.show\_button)  
  
 # Кнопки для операций над множествами  
 self.union\_button = QPushButton('Объединение с другим множеством', self)  
 self.union\_button.clicked.connect(self.union\_set)  
 self.layout.addWidget(self.union\_button)  
  
 self.intersection\_button = QPushButton('Пересечение с другим множеством', self)  
 self.intersection\_button.clicked.connect(self.intersection\_set)  
 self.layout.addWidget(self.intersection\_button)  
  
 # Список для отображения элементов множества  
 self.list\_widget = QListWidget(self)  
 self.layout.addWidget(self.list\_widget)  
  
 self.setLayout(self.layout)  
  
 def add\_element(self):  
 element = self.input\_field.text().strip()  
 if element:  
 self.set\_data.add(element)  
 QMessageBox.information(self, 'Успех', f'Элемент "{element}" добавлен.')  
 self.input\_field.clear()  
 else:  
 QMessageBox.warning(self, 'Ошибка', 'Введите элемент.')  
  
 def remove\_element(self):  
 element = self.input\_field.text().strip()  
 if element in self.set\_data:  
 self.set\_data.remove(element)  
 QMessageBox.information(self, 'Успех', f'Элемент "{element}" удален.')  
 self.input\_field.clear()  
 else:  
 QMessageBox.warning(self, 'Ошибка', f'Элемент "{element}" не найден в множестве.')  
  
 def show\_set(self):  
 self.list\_widget.clear()  
 for item in sorted(self.set\_data):  
 self.list\_widget.addItem(item)  
  
 def union\_set(self):  
 other\_set = self.get\_other\_set()  
 if other\_set is not None:  
 result\_set = self.set\_data.union(other\_set)  
 QMessageBox.information(self, 'Результат', f'Объединение: {result\_set}')  
  
 def intersection\_set(self):  
 other\_set = self.get\_other\_set()  
 if other\_set is not None:  
 result\_set = self.set\_data.intersection(other\_set)  
 QMessageBox.information(self, 'Результат', f'Пересечение: {result\_set}')  
  
 def get\_other\_set(self):  
 text, ok = QInputDialog.getText(self, 'Введите другое множество', 'Введите элементы через запятую:')  
 if ok and text:  
 return set(map(str.strip, text.split(',')))  
 return None  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app = QApplication(sys.argv)  
 ex = SetApp()  
 ex.show()  
 sys.exit(app.exec\_())

# **14 Разработать приложение для демонстрации работы с абстрактным типом данных "Односвязный список".**

class Node:  
 def \_\_init\_\_(self, data):  
 self.data = data  
 self.next = None  
  
class SinglyLinkedList:  
 def \_\_init\_\_(self):  
 self.head = None  
  
 def is\_empty(self):  
 return self.head is None  
  
 def append(self, data):  
 new\_node = Node(data)  
 if self.is\_empty():  
 self.head = new\_node  
 else:  
 current = self.head  
 while current.next:  
 current = current.next  
 current.next = new\_node  
  
 def prepend(self, data):  
 new\_node = Node(data)  
 new\_node.next = self.head  
 self.head = new\_node  
  
 def delete(self, data):  
 if self.is\_empty():  
 print("Список пуст. Удаление невозможно.")  
 return  
  
 if self.head.data == data:  
 self.head = self.head.next  
 return  
  
 current = self.head  
 while current.next and current.next.data != data:  
 current = current.next  
  
 if current.next is None:  
 print(f"Элемент {data} не найден в списке.")  
 else:  
 current.next = current.next.next  
  
 def display(self):  
 if self.is\_empty():  
 print("Список пуст.")  
 return  
  
 current = self.head  
 while current:  
 print(current.data, end=" -> ")  
 current = current.next  
 print("None")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 linked\_list = SinglyLinkedList()  
  
 while True:  
 print("\nМеню:")  
 print("1. Добавить в конец")  
 print("2. Добавить в начало")  
 print("3. Удалить элемент")  
 print("4. Отобразить список")  
 print("5. Выйти")  
  
 choice = input("Выберите действие: ")  
  
 if choice == "1":  
 data = input("Введите значение: ")  
 linked\_list.append(data)  
 elif choice == "2":  
 data = input("Введите значение: ")  
 linked\_list.prepend(data)  
 elif choice == "3":  
 data = input("Введите значение для удаления: ")  
 linked\_list.delete(data)  
 elif choice == "4":  
 linked\_list.display()  
 elif choice == "5":  
 break  
 else:  
 print("Неверный выбор. Попробуйте снова.")

# **15 Разработать визуальное приложение «Таймер» с тремя кнопками «старт», «стоп», «сброс», которое бы показывало на экране прошедшее время с хранящегося момента времени до остановки. Кнопка «сброс» «сбрасывает» таймер**

import sys  
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QPushButton, QLabel  
from PyQt5.QtCore import QTimer, QTime  
  
class TimerApp(QWidget):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
  
 self.initUI()  
  
 def initUI(self):  
 # Основные настройки  
 self.setWindowTitle("Таймер")  
 self.setGeometry(100, 100, 300, 150)  
  
 # Время и таймер  
 self.start\_time = QTime(0, 0, 0)  
 self.timer = QTimer()  
 self.timer.timeout.connect(self.update\_time)  
  
 # Элементы интерфейса  
 self.label = QLabel(self.start\_time.toString("hh:mm:ss"), self)  
 self.label.setStyleSheet("font-size: 24px; text-align: center;")  
  
 self.start\_button = QPushButton("Старт", self)  
 self.stop\_button = QPushButton("Стоп", self)  
 self.reset\_button = QPushButton("Сброс", self)  
  
 # Расположение элементов  
 layout = QVBoxLayout()  
 layout.addWidget(self.label)  
 layout.addWidget(self.start\_button)  
 layout.addWidget(self.stop\_button)  
 layout.addWidget(self.reset\_button)  
  
 self.setLayout(layout)  
  
 # Связи сигналов и слотов  
 self.start\_button.clicked.connect(self.start\_timer)  
 self.stop\_button.clicked.connect(self.stop\_timer)  
 self.reset\_button.clicked.connect(self.reset\_timer)  
  
 def start\_timer(self):  
 self.timer.start(1000)  
  
 def stop\_timer(self):  
 self.timer.stop()  
  
 def reset\_timer(self):  
 self.timer.stop()  
 self.start\_time = QTime(0, 0, 0)  
 self.label.setText(self.start\_time.toString("hh:mm:ss"))  
  
 def update\_time(self):  
 self.start\_time = self.start\_time.addSecs(1)  
 self.label.setText(self.start\_time.toString("hh:mm:ss"))  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = QApplication(sys.argv)  
 timer\_app = TimerApp()  
 timer\_app.show()  
 sys.exit(app.exec\_())

# **16 В таблицах базы данных хранится информация о посещении студентами колледжа спортивных секций: фамилия, группа, секция, тренер, дата занятия. Провести нормализацию. Разработать оконное приложение для вывода в таблицу (grid) занятий группы 399ИС-22, отсортированных по дате**

**Код нормализации SQL**

-- Таблица студентов  
CREATE TABLE Students (  
 StudentID INTEGER PRIMARY KEY,  
 Surname TEXT NOT NULL,  
 GroupID INTEGER NOT NULL,  
 FOREIGN KEY (GroupID) REFERENCES Groups(GroupID)  
);  
  
-- Таблица групп  
CREATE TABLE Groups (  
 GroupID INTEGER PRIMARY KEY,  
 GroupName TEXT NOT NULL UNIQUE  
);  
  
-- Таблица секций  
CREATE TABLE Sections (  
 SectionID INTEGER PRIMARY KEY,  
 SectionName TEXT NOT NULL UNIQUE,  
 CoachID INTEGER NOT NULL,  
 FOREIGN KEY (CoachID) REFERENCES Coaches(CoachID)  
);  
  
-- Таблица тренеров  
CREATE TABLE Coaches (  
 CoachID INTEGER PRIMARY KEY,  
 CoachName TEXT NOT NULL UNIQUE  
);  
  
-- Таблица занятий  
CREATE TABLE Sessions (  
 SessionID INTEGER PRIMARY KEY,  
 SectionID INTEGER NOT NULL,  
 Date TEXT NOT NULL,  
 FOREIGN KEY (SectionID) REFERENCES Sections(SectionID)  
);

**Приложение Python с Tkinter**

import sqlite3  
from tkinter import Tk, ttk, messagebox  
  
  
def get\_sessions\_for\_group(group\_name):  
 *"""  
 Получает информацию о занятиях группы, отсортированных по дате.  
 """* query = """  
 SELECT Surname, Groups.GroupName, Sections.SectionName, Coaches.CoachName, Sessions.Date  
 FROM Students  
 JOIN Groups ON Students.GroupID = Groups.GroupID  
 JOIN Sections ON Groups.GroupID = Sections.SectionID  
 JOIN Coaches ON Sections.CoachID = Coaches.CoachID  
 JOIN Sessions ON Sections.SectionID = Sessions.SectionID  
 WHERE Groups.GroupName = ?  
 ORDER BY Sessions.Date;  
 """  
 try:  
 with sqlite3.connect("college.db") as conn:  
 cursor = conn.cursor()  
 cursor.execute(query, (group\_name,))  
 return cursor.fetchall()  
 except sqlite3.Error as e:  
 messagebox.showerror("Ошибка базы данных", str(e))  
 return []  
  
  
def populate\_table(data, tree):  
 *"""  
 Заполняет таблицу данными.  
 """* for row in tree.get\_children():  
 tree.delete(row)  
  
 for row in data:  
 tree.insert("", "end", values=row)  
  
  
def main():  
 # Настраиваем окно  
 root = Tk()  
 root.title("Занятия группы 399ИС-22")  
  
 # Создаем таблицу  
 columns = ("Фамилия", "Группа", "Секция", "Тренер", "Дата")  
 tree = ttk.Treeview(root, columns=columns, show="headings")  
 tree.pack(fill="both", expand=True)  
  
 for col in columns:  
 tree.heading(col, text=col)  
 tree.column(col, width=150)  
  
 # Загружаем данные  
 group\_name = "399ИС-22"  
 data = get\_sessions\_for\_group(group\_name)  
 populate\_table(data, tree)  
  
 # Запуск GUI  
 root.mainloop()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

# **17 Разработать программу с меню для ввода с клавиатуры целых чисел в массив (список), сортировки «обменом» и вывода в файл. Сортировку реализовать в виде функции в отдельном модуле.**

**Код программы:**

**1. Модуль сортировки (sort\_module.py):**

def bubble\_sort(arr):  
 *"""Функция для сортировки массива методом пузырька."""* n = len(arr)  
 for i in range(n):  
 # Флаг для проверки, была ли совершена перестановка  
 swapped = False  
 for j in range(0, n-i-1):  
 if arr[j] > arr[j+1]:  
 # Обмен элементов  
 arr[j], arr[j+1] = arr[j+1], arr[j]  
 swapped = True  
 # Если за проход не было перестановок, выходим из цикла  
 if not swapped:  
 break

1. **Главная программа (main.py):**

import os  
from sort\_module import bubble\_sort  
  
def input\_numbers():  
 *"""Функция для ввода чисел с клавиатуры."""* numbers = []  
 print("Введите целые числа для массива. Введите 'stop' для завершения ввода.")  
 while True:  
 user\_input = input("Введите число: ")  
 if user\_input.lower() == 'stop':  
 break  
 try:  
 number = int(user\_input)  
 numbers.append(number)  
 except ValueError:  
 print("Пожалуйста, вводите только целые числа.")  
 return numbers  
  
def save\_to\_file(arr, filename):  
 *"""Функция для сохранения отсортированного списка в файл."""* with open(filename, 'w') as f:  
 for number in arr:  
 f.write(f"{number}\n")  
 print(f"Список сохранен в файл {filename}")  
  
def main():  
 *"""Главная функция программы с меню."""* while True:  
 print("\nМеню:")  
 print("1. Ввести числа в массив")  
 print("2. Сортировать массив методом обмена")  
 print("3. Сохранить массив в файл")  
 print("4. Выйти")  
  
 choice = input("Выберите действие (1-4): ")  
  
 if choice == '1':  
 # Ввод чисел  
 numbers = input\_numbers()  
 elif choice == '2':  
 # Сортировка массива  
 if 'numbers' in locals():  
 bubble\_sort(numbers)  
 print("Массив отсортирован.")  
 else:  
 print("Сначала введите массив чисел.")  
 elif choice == '3':  
 # Сохранение массива в файл  
 if 'numbers' in locals():  
 save\_to\_file(numbers, "sorted\_numbers.txt")  
 else:  
 print("Сначала введите массив чисел.")  
 elif choice == '4':  
 # Выход из программы  
 print("Выход из программы.")  
 break  
 else:  
 print("Неверный выбор. Пожалуйста, выберите пункт из меню.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Объяснение кода:**

1. **Модуль сортировки** (sort\_module.py):
   * Функция bubble\_sort реализует алгоритм сортировки пузырьком (обмен).
   * Этот алгоритм состоит в том, чтобы несколько раз пройти по списку, сравнивая соседние элементы и меняя их местами, если они расположены в неправильном порядке.
   * Процесс повторяется до тех пор, пока не останется элементов, которые нужно поменять местами.
2. **Главная программа** (main.py):
   * **Меню**: Программа имеет меню с четырьмя опциями:
     1. Ввод чисел с клавиатуры.
     2. Сортировка массива с использованием функции из модуля сортировки.
     3. Сохранение отсортированного массива в файл.
     4. Выход из программы.
   * **Ввод чисел**: Пользователь может вводить целые числа, пока не введет команду "stop". Введенные числа сохраняются в список.
   * **Сортировка**: После ввода чисел можно отсортировать список с помощью функции bubble\_sort, которая импортирована из модуля sort\_module.py.
   * **Сохранение в файл**: Отсортированный список сохраняется в файл sorted\_numbers.txt в той же директории.
3. **Запуск программы**:

Программа будет работать в консоли. Пользователь может вводить числа, сортировать их и сохранять в файл, выбирая соответствующие опции в меню.

# **18 Разработать программу с меню для ввода из файла целых чисел в массив (список), сортировки «выбором» и вывода на экран. Сортировку реализовать в виде функции в отдельном модуле.**

import os  
from sorting import selection\_sort  
  
  
def selection\_sort(arr):  
 *"""Сортировка методом выбора."""* n = len(arr)  
 for i in range(n):  
 # Найти минимальный элемент в оставшейся части массива  
 min\_idx = i  
 for j in range(i + 1, n):  
 if arr[j] < arr[min\_idx]:  
 min\_idx = j  
 # Обменять найденный минимальный элемент с первым элементом  
 arr[i], arr[min\_idx] = arr[min\_idx], arr[i]  
  
  
def read\_numbers\_from\_file(filename):  
 *"""Читает целые числа из файла и возвращает их в виде списка."""* if not os.path.isfile(filename):  
 print(f"Файл {filename} не найден.")  
 return []  
  
 with open(filename, 'r') as file:  
 numbers = []  
 for line in file:  
 try:  
 numbers.extend(map(int, line.split()))  
 except ValueError:  
 print(f"Некорректные данные в строке: {line.strip()}")  
 return numbers  
  
  
def main():  
 while True:  
 print("\nМеню:")  
 print("1. Ввести данные из файла")  
 print("2. Сортировать массив")  
 print("3. Вывести массив")  
 print("4. Выход")  
  
 choice = input("Выберите действие (1-4): ")  
  
 if choice == '1':  
 filename = input("Введите имя файла: ")  
 numbers = read\_numbers\_from\_file(filename)  
 print(f"Числа из файла {filename} были загружены.")  
  
 elif choice == '2':  
 if 'numbers' in locals():  
 selection\_sort(numbers)  
 print("Массив отсортирован.")  
 else:  
 print("Сначала загрузите массив из файла.")  
  
 elif choice == '3':  
 if 'numbers' in locals():  
 print("Текущий массив:", numbers)  
 else:  
 print("Сначала загрузите массив из файла.")  
  
 elif choice == '4':  
 print("Выход из программы.")  
 break  
  
 else:  
 print("Некорректный выбор. Пожалуйста, выберите 1-4.")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

# **19 В каждой строке текстового файла in.txt посещения библиотеки хранится запись: студент: дд.мм.гг, дд.мм.гг. … Сформировать файл out.txt, содержащий строки следующего вида: дд.мм.гг: Студент1, Студент2, … Использовать словари (хэш-таблицы).**

**Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**def process\_library\_visits(input\_file, output\_file):  
 visits = {}  
  
 # Чтение и обработка данных  
 with open(input\_file, "r", encoding="utf-8") as infile:  
 for line in infile:  
 parts = line.strip().split(":")  
 if len(parts) != 2:  
 continue # Пропуск некорректных строк  
  
 student, dates = parts  
 for date in dates.split(","):  
 date = date.strip()  
 if date not in visits:  
 visits[date] = []  
 visits[date].append(student.strip())  
  
 # Запись результата в выходной файл  
 with open(output\_file, "w", encoding="utf-8") as outfile:  
 for date, students in sorted(visits.items()):  
 outfile.write(f"{date}: {', '.join(sorted(students))}\n")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 input\_filename = "in.txt"  
 output\_filename = "out.txt"  
  
 process\_library\_visits(input\_filename, output\_filename)  
 print(f"Данные обработаны. Результат записан в {output\_filename}.")

**Изображение выглядит как текст, Шрифт, программное обеспечение, снимок экрана

Автоматически созданное описание**

# **20 В базе данных хранятся сведения о заработной плате сотрудников отделов предприятия. Увеличить заработную плату сотрудникам отдела главного конструктора на 10%. Отобразить в табличном виде на экране**

import sys  
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QPushButton, QLabel, QTableWidget, QTableWidgetItem  
import sqlite3  
  
class SalaryApp(QWidget):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
  
 self.initDB() # Инициализация базы данных должна быть до UI  
 self.initUI()  
  
 def initUI(self):  
 # Основные настройки  
 self.setWindowTitle("Заработная плата сотрудников")  
 self.setGeometry(100, 100, 600, 400)  
  
 # Элементы интерфейса  
 self.table = QTableWidget(self)  
  
 self.update\_button = QPushButton("Увеличить зарплату на 10%", self)  
 self.update\_button.clicked.connect(self.update\_salary)  
  
 # Расположение элементов  
 layout = QVBoxLayout()  
 layout.addWidget(self.table)  
 layout.addWidget(self.update\_button)  
  
 self.setLayout(layout)  
  
 self.display\_data()  
  
 def initDB(self):  
 # Подключение к базе данных  
 self.conn = sqlite3.connect('enterprise.db') # Файл базы данных  
 self.cursor = self.conn.cursor() # Создание курсора  
  
 # Создание таблицы сотрудников, если ее нет  
 self.cursor.execute('''  
 CREATE TABLE IF NOT EXISTS employees (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 name TEXT,  
 department TEXT,  
 salary REAL  
 )  
 ''')  
  
 # Очистка и заполнение тестовыми данными  
 self.cursor.execute('DELETE FROM employees')  
 employees = [  
 ("Иван Иванов", "Главный конструктор", 50000),  
 ("Петр Петров", "Главный конструктор", 60000),  
 ("Сергей Сергеев", "Отдел продаж", 45000),  
 ("Мария Иванова", "Отдел кадров", 40000)  
 ]  
 self.cursor.executemany('INSERT INTO employees (name, department, salary) VALUES (?, ?, ?)', employees)  
 self.conn.commit()  
  
 def display\_data(self):  
 # Отображение данных из базы в таблице  
 self.cursor.execute('SELECT \* FROM employees')  
 rows = self.cursor.fetchall()  
  
 self.table.setRowCount(len(rows))  
 self.table.setColumnCount(4)  
 self.table.setHorizontalHeaderLabels(["ID", "Имя", "Отдел", "Зарплата"])  
  
 for row\_idx, row\_data in enumerate(rows):  
 for col\_idx, col\_data in enumerate(row\_data):  
 self.table.setItem(row\_idx, col\_idx, QTableWidgetItem(str(col\_data)))  
  
 def update\_salary(self):  
 # Увеличение зарплаты сотрудников отдела "Главный конструктор" на 10%  
 self.cursor.execute('''  
 UPDATE employees  
 SET salary = salary \* 1.1  
 WHERE department = "Главный конструктор"  
 ''')  
 self.conn.commit()  
  
 self.display\_data()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = QApplication(sys.argv)  
 salary\_app = SalaryApp()  
 salary\_app.show()  
 sys.exit(app.exec\_())

# **21 Разработать программу для формирования в текущем каталоге системы каталогов.**

Название каждого подкаталога - строка входного файла in.txt. гарантируется, что строки уникальны. В каждом каталоге создать файл readme.md, который содержит время создания, через пробел - случайное целое число в диапазоне от 0 до 1000000

import os  
import random  
from datetime import datetime  
  
  
def create\_directories\_from\_file(file\_path):  
 # Читаем имена каталогов из файла  
 with open(file\_path, 'r') as file:  
 directory\_names = [line.strip() for line in file.readlines()]  
  
 # Создаем подкаталоги и файлы readme.md  
 for directory\_name in directory\_names:  
 # Создаем подкаталог  
 os.makedirs(directory\_name, exist\_ok=True)  
  
 # Создаем файл readme.md в подкаталоге  
 readme\_file\_path = os.path.join(directory\_name, 'readme.md')  
  
 # Генерируем случайное целое число  
 random\_number = random.randint(0, 1000000)  
  
 # Получаем текущее время  
 creation\_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')  
  
 # Записываем информацию в файл  
 with open(readme\_file\_path, 'w') as readme\_file:  
 readme\_file.write(f"{creation\_time} {random\_number}\n")  
  
  
# Указываем путь к файлу in.txt  
create\_directories\_from\_file('in.txt')

# **22 Разработать визуальное приложение с меню для выбора и шифрования текстового файла по методу Цезаря.**

Файл содержит слова, записанные строчными буквами, разделённые пробелами. Ключ вводится через поле ввода. Кнопки «выбрать файл», «зашифровать», «расшифровать»

**Код программы:**

import tkinter as tk  
from tkinter import filedialog, messagebox  
import os  
  
# Функция для шифрования текста методом Цезаря  
def caesar\_cipher(text, key):  
 result = []  
 for char in text:  
 if char.isalpha(): # Проверяем, является ли символ буквой  
 shift = key % 26 # Сдвиг по алфавиту (в пределах 26 букв)  
 if char.islower(): # Для строчных букв  
 new\_char = chr((ord(char) - ord('a') + shift) % 26 + ord('a'))  
 else: # Для заглавных букв (если нужно)  
 new\_char = chr((ord(char) - ord('A') + shift) % 26 + ord('A'))  
 result.append(new\_char)  
 else:  
 result.append(char) # Прочие символы остаются без изменений  
 return ''.join(result)  
  
# Функция для расшифровки текста методом Цезаря  
def caesar\_decipher(text, key):  
 return caesar\_cipher(text, -key) # Для расшифровки сдвиг отрицательный  
  
# Функция для выбора файла  
def select\_file():  
 file\_path = filedialog.askopenfilename(filetypes=[("Text files", "\*.txt")])  
 if file\_path:  
 with open(file\_path, 'r') as file:  
 file\_content = file.read()  
 text\_box.delete(1.0, tk.END) # Очищаем текстовое поле  
 text\_box.insert(tk.END, file\_content) # Загружаем содержимое файла в текстовое поле  
 global current\_file\_path  
 current\_file\_path = file\_path # Сохраняем путь к выбранному файлу  
  
# Функция для шифрования текста и сохранения его в файл  
def encrypt\_text():  
 key = int(key\_entry.get())  
 if current\_file\_path:  
 original\_text = text\_box.get(1.0, tk.END).strip()  
 encrypted\_text = caesar\_cipher(original\_text, key)  
 with open(current\_file\_path, 'w') as file:  
 file.write(encrypted\_text)  
 messagebox.showinfo("Успех", "Текст успешно зашифрован и сохранен в файл.")  
 else:  
 messagebox.showerror("Ошибка", "Выберите файл для шифрования.")  
  
# Функция для расшифровки текста и сохранения его в файл  
def decrypt\_text():  
 key = int(key\_entry.get())  
 if current\_file\_path:  
 original\_text = text\_box.get(1.0, tk.END).strip()  
 decrypted\_text = caesar\_decipher(original\_text, key)  
 with open(current\_file\_path, 'w') as file:  
 file.write(decrypted\_text)  
 messagebox.showinfo("Успех", "Текст успешно расшифрован и сохранен в файл.")  
 else:  
 messagebox.showerror("Ошибка", "Выберите файл для расшифровки.")  
  
# Основное окно приложения  
root = tk.Tk()  
root.title("Шифрование методом Цезаря")  
  
# Глобальная переменная для хранения пути к выбранному файлу  
current\_file\_path = ""  
  
# Фрейм для кнопок  
frame = tk.Frame(root)  
frame.pack(pady=10)  
  
# Кнопка выбора файла  
select\_button = tk.Button(frame, text="Выбрать файл", command=select\_file)  
select\_button.grid(row=0, column=0, padx=5)  
  
# Поле для ввода ключа  
key\_label = tk.Label(frame, text="Введите ключ (сдвиг):")  
key\_label.grid(row=0, column=1, padx=5)  
  
key\_entry = tk.Entry(frame, width=10)  
key\_entry.grid(row=0, column=2, padx=5)  
  
# Кнопка шифрования  
encrypt\_button = tk.Button(frame, text="Зашифровать", command=encrypt\_text)  
encrypt\_button.grid(row=1, column=0, padx=5, pady=5)  
  
# Кнопка расшифровки  
decrypt\_button = tk.Button(frame, text="Расшифровать", command=decrypt\_text)  
decrypt\_button.grid(row=1, column=1, padx=5, pady=5)  
  
# Текстовое поле для отображения/редактирования текста файла  
text\_box = tk.Text(root, height=10, width=50)  
text\_box.pack(pady=10)  
  
# Запуск главного цикла Tkinter  
root.mainloop()

**Объяснение кода:**

1. **Метод Цезаря**:
   * Шифрование текста осуществляется сдвигом каждой буквы на указанное количество позиций по алфавиту.
   * Для расшифровки текст сдвигается в противоположную сторону.
2. **Визуальный интерфейс**:
   * Используется библиотека tkinter для создания графического интерфейса.
   * Включает текстовое поле для отображения содержимого файла и поле ввода для ключа.
   * Кнопки для выбора файла, шифрования и расшифровки текста.
3. **Шифрование и расшифровка**:
   * Для шифрования и расшифровки текста применяются функции caesar\_cipher и caesar\_decipher.
   * Выбранный текст из файла зашифровывается или расшифровывается с помощью введённого ключа (сдвиг в алфавите).
4. **Функции**:
   * **select\_file**: Открывает диалоговое окно для выбора файла.
   * **encrypt\_text**: Шифрует текст в файле и сохраняет результат в тот же файл.
   * **decrypt\_text**: Расшифровывает текст в файле и сохраняет результат.
5. **Путь к файлу**:
   * Путь к выбранному файлу сохраняется в переменной current\_file\_path, что позволяет шифровать или расшифровывать только этот файл.

**Пример использования:**

1. **Выбор файла**:
   * Нажмите кнопку "Выбрать файл" и выберите файл с текстом для шифрования.
2. **Ввод ключа**:
   * Введите ключ сдвига (например, 3) в поле "Введите ключ (сдвиг)".
3. **Шифрование/Расшифровка**:
   * Нажмите на кнопку "Зашифровать" для шифрования текста или на кнопку "Расшифровать" для расшифровки текста.
4. **Результат**:
   * Программа перезапишет файл с зашифрованным или расшифрованным текстом.

# **23 Разработать программу для хранения полинома N-степени с использованием абстрактного типа данных «Односвязный список».**

Программа должна считывать из текстового файла in.txt два набора целых чисел и выполнить действия сложения, умножения, сравнения. Результат вывести на экран.

class Node:  
 *"""Узел односвязного списка для хранения коэффициента и степени."""* def \_\_init\_\_(self, coefficient, degree):  
 self.coefficient = coefficient  
 self.degree = degree  
 self.next = None  
  
  
class LinkedList:  
 *"""Односвязный список для представления полинома."""* def \_\_init\_\_(self):  
 self.head = None  
  
 def append(self, coefficient, degree):  
 *"""Добавить новый узел в конец списка."""* new\_node = Node(coefficient, degree)  
 if not self.head:  
 self.head = new\_node  
 return  
 current = self.head  
 while current.next:  
 current = current.next  
 current.next = new\_node  
  
 def display(self):  
 *"""Вывести полином на экран."""* current = self.head  
 terms = []  
 while current:  
 terms.append(f"{current.coefficient}x^{current.degree}")  
 current = current.next  
 print(" + ".join(terms))  
  
 def to\_list(self):  
 *"""Преобразовать полином в список кортежей (коэффициент, степень)."""* result = []  
 current = self.head  
 while current:  
 result.append((current.coefficient, current.degree))  
 current = current.next  
 return result  
  
 @staticmethod  
 def from\_list(lst):  
 *"""Создать полином из списка кортежей (коэффициент, степень)."""* poly = LinkedList()  
 for coeff, deg in lst:  
 poly.append(coeff, deg)  
 return poly  
  
  
def add\_polynomials(poly1, poly2):  
 *"""Сложить два полинома."""* result = LinkedList()  
 p1 = poly1.head  
 p2 = poly2.head  
  
 while p1 or p2:  
 if not p1:  
 result.append(p2.coefficient, p2.degree)  
 p2 = p2.next  
 elif not p2:  
 result.append(p1.coefficient, p1.degree)  
 p1 = p1.next  
 elif p1.degree > p2.degree:  
 result.append(p1.coefficient, p1.degree)  
 p1 = p1.next  
 elif p1.degree < p2.degree:  
 result.append(p2.coefficient, p2.degree)  
 p2 = p2.next  
 else: # p1.degree == p2.degree  
 result.append(p1.coefficient + p2.coefficient, p1.degree)  
 p1 = p1.next  
 p2 = p2.next  
  
 return result  
  
  
def multiply\_polynomials(poly1, poly2):  
 *"""Умножить два полинома."""* result = LinkedList()  
 p1 = poly1.head  
  
 while p1:  
 temp\_poly = LinkedList()  
 p2 = poly2.head  
 while p2:  
 temp\_poly.append(p1.coefficient \* p2.coefficient, p1.degree + p2.degree)  
 p2 = p2.next  
 result = add\_polynomials(result, temp\_poly)  
 p1 = p1.next  
  
 return result  
  
  
def compare\_polynomials(poly1, poly2):  
 *"""Сравнить два полинома."""* # Сравниваем по степени первого не нулевого коэффициента  
 max\_degree\_1 = poly1.head.degree if poly1.head else -1  
 max\_degree\_2 = poly2.head.degree if poly2.head else -1  
  
 if max\_degree\_1 > max\_degree\_2:  
 return 1 # poly1 больше  
 elif max\_degree\_1 < max\_degree\_2:  
 return -1 # poly2 больше  
 else:  
 return 0 # равны  
  
  
from linked\_list import LinkedList, add\_polynomials, multiply\_polynomials, compare\_polynomials  
  
  
def read\_polynomial\_from\_file(filename):  
 *"""Читать полином из файла."""* with open(filename, 'r') as file:  
 lines = file.readlines()  
  
 polynomials = []  
  
 for line in lines:  
 coefficients\_degrees = list(map(int, line.split()))  
 poly = LinkedList.from\_list(zip(coefficients\_degrees[::2], coefficients\_degrees[1::2]))  
 polynomials.append(poly)  
  
 return polynomials  
  
  
def main():  
 filename = 'in.txt'  
  
 # Чтение полиномов из файла  
 polynomials = read\_polynomial\_from\_file(filename)  
  
 if len(polynomials) < 2:  
 print("Недостаточно полиномов для выполнения операций.")  
 return  
  
 poly1, poly2 = polynomials[0], polynomials[1]  
  
 print("Полином 1:")  
 poly1.display()  
  
 print("Полином 2:")  
 poly2.display()  
  
 # Сложение  
 sum\_poly = add\_polynomials(poly1, poly2)  
 print("Сумма:")  
 sum\_poly.display()  
  
 # Умножение  
 product\_poly = multiply\_polynomials(poly1, poly2)  
 print("Произведение:")  
 product\_poly.display()  
  
 # Сравнение  
 comparison\_result = compare\_polynomials(poly1, poly2)  
  
 if comparison\_result == 0:  
 print("Полиномы равны.")  
 elif comparison\_result > 0:  
 print("Первый полином больше второго.")  
 else:  
 print("Второй полином больше первого.")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

# **24 Разработать рекурсивную функцию определения N-числа Фибоначчи. Программа должна печатать число рекурсивных вызовов**

def fibonacci(n, call\_count):  
  
 call\_count[0] += 1  
  
 if n == 0:  
 return 0  
 elif n == 1:  
 return 1  
 else:  
 return fibonacci(n - 1, call\_count) + fibonacci(n - 2, call\_count)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 n = int(input("Введите позицию числа Фибоначчи (n >= 0): "))  
  
 if n < 0:  
 print("Ошибка: n должно быть неотрицательным числом.")  
 else:  
 call\_count = [0] # Используем список, чтобы счётчик можно было изменять внутри функции  
 result = fibonacci(n, call\_count)  
 print(f"Число Фибоначчи на позиции {n}: {result}")  
 print(f"Количество вызовов функции: {call\_count[0]}")

# **25 Разработать функцию для печати суммы прописью до 1000000 рублей. Пример: для 123456 функция должна печатать Сто двадцать три тысяч четыреста пятьдесят шесть (рублей).**

def number\_to\_words(n):  
 # Словари для преобразования  
 units = ["", "один", "два", "три", "четыре", "пять", "шесть", "семь", "восемь", "девять"]  
 teens = ["", "одиннадцать", "двенадцать", "тринадцать", "четырнадцать", "пятнадцать",  
 "шестнадцать", "семнадцать", "восемнадцать", "девятнадцать"]  
 tens = ["", "десять", "двадцать", "тридцать", "сорок", "пятьдесят", "шестьдесят",  
 "семьдесят", "восемьдесят", "девяносто"]  
 hundreds = ["", "сто", "двести", "триста", "четыреста", "пятьсот",  
 "шестьсот", "семьсот", "восемьсот", "девятьсот"]  
 thousands = ["", "тысяча", "тысячи", "тысяч"]  
  
 def num\_to\_text(number):  
 if number == 0:  
 return "ноль"  
  
 parts = []  
  
 # Разделение на тысячи и сотни  
 thousands\_part = number // 1000  
 hundreds\_part = number % 1000  
  
 # Обработка тысяч  
 if thousands\_part > 0:  
 parts.append(hundreds[thousands\_part // 100]) # Сотни в тысячах  
 remainder = thousands\_part % 100  
 if 10 < remainder < 20:  
 parts.append(teens[remainder - 10])  
 else:  
 parts.append(tens[remainder // 10])  
 parts.append(units[remainder % 10])  
 # Определение правильного окончания "тысяч"  
 if remainder % 10 == 1 and remainder != 11:  
 parts.append(thousands[1])  
 elif 2 <= remainder % 10 <= 4 and not (12 <= remainder <= 14):  
 parts.append(thousands[2])  
 else:  
 parts.append(thousands[3])  
  
 # Обработка сотен  
 parts.append(hundreds[hundreds\_part // 100])  
 remainder = hundreds\_part % 100  
 if 10 < remainder < 20:  
 parts.append(teens[remainder - 10])  
 else:  
 parts.append(tens[remainder // 10])  
 parts.append(units[remainder % 10])  
  
 return " ".join(filter(None, parts))  
  
 return f"{num\_to\_text(n)} рублей"  
  
  
# Пример использования  
if \_\_name\_\_ == "\_\_main\_\_":  
 number = int(input("Введите число до 1000000: "))  
 if 0 <= number < 1000000:  
 print(number\_to\_words(number))  
 else:  
 print("Введите число от 0 до 999999.")

# **26 Разработать класс Cart (Корзина).**

Каждый объект класса Cart должен иметь локальное

свойство goods - список объектов для покупки (объекты классов Fruit, Vegetable, Bread и Tea).тИзначально этот список должен быть пустым. В классе Cart объявить методы:

add(self, gd) - добавление в корзину товара, представленного объектом gd;

remove(self, indx) - удаление из корзины товара по индексу indx;

get\_list(self) - получение из корзины товаров в виде списка из строк:

['<наименовние\_1>: <цена\_1>',

'<наименовние\_2>: <цена\_2>',

...

Разработать программу для демонстрации системы классов

class Product:  
 *"""Базовый класс для товаров."""* def \_\_init\_\_(self, name, price):  
 self.name = name  
 self.price = price  
  
 def \_\_str\_\_(self):  
 return f"{self.name}: {self.price}"  
  
  
class Fruit(Product):  
 *"""Класс для фруктов."""* def \_\_init\_\_(self, name, price):  
 super().\_\_init\_\_(name, price)  
  
  
class Vegetable(Product):  
 *"""Класс для овощей."""* def \_\_init\_\_(self, name, price):  
 super().\_\_init\_\_(name, price)  
  
  
class Bread(Product):  
 *"""Класс для хлеба."""* def \_\_init\_\_(self, name, price):  
 super().\_\_init\_\_(name, price)  
  
  
class Tea(Product):  
 *"""Класс для чая."""* def \_\_init\_\_(self, name, price):  
 super().\_\_init\_\_(name, price)  
  
  
class Cart:  
 *"""Класс для корзины."""* def \_\_init\_\_(self):  
 self.goods = []  
  
 def add(self, gd):  
 *"""Добавление товара в корзину."""* self.goods.append(gd)  
  
 def remove(self, indx):  
 *"""Удаление товара из корзины по индексу."""* if 0 <= indx < len(self.goods):  
 del self.goods[indx]  
 else:  
 print("Ошибка: индекс вне диапазона!")  
  
 def get\_list(self):  
 *"""Получение списка товаров в корзине."""* return [str(item) for item in self.goods]  
  
  
# Демонстрация работы системы классов  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Создаем объекты товаров  
 apple = Fruit("Apple", 1.5)  
 carrot = Vegetable("Carrot", 0.8)  
 baguette = Bread("Baguette", 2.0)  
 green\_tea = Tea("Green Tea", 3.5)  
  
 # Создаем корзину  
 my\_cart = Cart()  
  
 # Добавляем товары в корзину  
 my\_cart.add(apple)  
 my\_cart.add(carrot)  
 my\_cart.add(baguette)  
 my\_cart.add(green\_tea)  
  
 # Выводим список товаров в корзине  
 print("Товары в корзине:")  
 print("\n".join(my\_cart.get\_list()))  
  
 # Удаляем товар по индексу  
 my\_cart.remove(1)  
  
 # Выводим обновленный список товаров  
 print("\nОбновленный список товаров:")  
 print("\n".join(my\_cart.get\_list()))

Объяснение кода:

1. Базовый класс Product:
   * Содержит общие свойства и методы для всех товаров: название (name) и цена (price).
   * Метод \_\_str\_\_ возвращает строку вида <название>: <цена>.
2. Наследуемые классы (Fruit, Vegetable, Bread, Tea):
   * Наследуют базовые свойства и поведение от Product.
   * Каждый класс можно дополнительно расширить при необходимости.
3. Класс Cart:
   * Хранит товары в локальном свойстве goods, которое изначально пустое.
   * Методы:
     + add(self, gd): добавляет товар (gd) в корзину.
     + remove(self, indx): удаляет товар по индексу (indx), с проверкой диапазона индекса.
     + get\_list(self): возвращает список товаров в корзине в виде строк.
4. Демонстрация работы:
   * Создаются экземпляры товаров.
   * Товары добавляются в корзину.
   * Выводится список товаров, производится удаление по индексу, и список выводится снова.

Пример вывода программы:

Товары в корзине:

Apple: 1.5

Carrot: 0.8

Baguette: 2.0

Green Tea: 3.5

Обновленный список товаров:

Apple: 1.5

Baguette: 2.0

Green Tea: 3.5

# **27 Разработать приложение с графическим интерфейсом для создания логина и пароля.**

Приложение должно содержать 3 текстовых поля ввода: для имени, для пароля и для повторения пароля и кнопку «сохранить» (изначально – неактивную). Логин – строка строчных букв английского алфавита длиной не менее 5. Пароль должен включать строчные английские буквы, цифры, знаки «@-!» длиной не менее 8. Содержимое обоих полей пароля должно совпадать. При невыполнении какого-либо условия программа должна выдавать сообщение. Если ввод корректный, кнопка «Сохранить» становится активной. Логин и пароль должны дописываться с новой строки в файл out.txt

import tkinter as tk  
from tkinter import messagebox  
import re  
  
  
class LoginApp:  
 def \_\_init\_\_(self, master):  
 self.master = master  
 master.title("Создание логина и пароля")  
  
 # Метки и текстовые поля  
 self.label\_username = tk.Label(master, text="Имя пользователя:")  
 self.label\_username.pack()  
  
 self.entry\_username = tk.Entry(master)  
 self.entry\_username.pack()  
  
 self.label\_password = tk.Label(master, text="Пароль:")  
 self.label\_password.pack()  
  
 self.entry\_password = tk.Entry(master, show="\*")  
 self.entry\_password.pack()  
  
 self.label\_repeat\_password = tk.Label(master, text="Повторите пароль:")  
 self.label\_repeat\_password.pack()  
  
 self.entry\_repeat\_password = tk.Entry(master, show="\*")  
 self.entry\_repeat\_password.pack()  
  
 # Кнопка "Сохранить"  
 self.save\_button = tk.Button(master, text="Сохранить", command=self.save, state=tk.DISABLED)  
 self.save\_button.pack()  
  
 # Привязываем проверку ввода к изменениям в текстовых полях  
 self.entry\_username.bind("<KeyRelease>", self.validate\_input)  
 self.entry\_password.bind("<KeyRelease>", self.validate\_input)  
 self.entry\_repeat\_password.bind("<KeyRelease>", self.validate\_input)  
  
 def validate\_input(self, event=None):  
 username = self.entry\_username.get()  
 password = self.entry\_password.get()  
 repeat\_password = self.entry\_repeat\_password.get()  
  
 # Проверка логина  
 username\_valid = len(username) >= 5 and username.islower()  
  
 # Проверка пароля  
 password\_valid = (len(password) >= 8 and  
 re.search("[a-z]", password) and  
 re.search("[0-9]", password) and  
 re.search("[-!@]", password))  
  
 # Проверка совпадения паролей  
 passwords\_match = password == repeat\_password  
  
 # Активируем кнопку, если все условия выполнены  
 if username\_valid and password\_valid and passwords\_match:  
 self.save\_button.config(state=tk.NORMAL)  
 else:  
 self.save\_button.config(state=tk.DISABLED)  
  
 def save(self):  
 username = self.entry\_username.get()  
 password = self.entry\_password.get()  
  
 # Записываем логин и пароль в файл out.txt  
 with open("out.txt", "a") as f:  
 f.write(f"{username} {password}\n")  
  
 # Сообщение об успешном сохранении  
 messagebox.showinfo("Успех", "Логин и пароль успешно сохранены!")  
  
 # Очищаем поля ввода  
 self.entry\_username.delete(0, tk.END)  
 self.entry\_password.delete(0, tk.END)  
 self.entry\_repeat\_password.delete(0, tk.END)  
 self.save\_button.config(state=tk.DISABLED)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 root = tk.Tk()  
 app = LoginApp(root)  
 root.mainloop()

# **28 Разработать программу для генерации последовательности случайных целых чисел в бинарном файле и сортировки файла. Саму сортировку реализовать в виде функции в отдельном модуле**

generate\_numbers.py

import random  
import struct  
  
def generate\_random\_numbers(file\_name, count):  
 *"""Генерирует случайные целые числа и записывает их в бинарный файл."""* with open(file\_name, 'wb') as f:  
 for \_ in range(count):  
 number = random.randint(0, 1000000) # Генерация случайного числа  
 f.write(struct.pack('i', number)) # Запись числа в бинарный файл  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 generate\_random\_numbers('numbers.bin', 10) # Пример: генерируем 10 случайных чисел

sort\_numbers.py

import struct  
  
  
def sort\_binary\_file(file\_name):  
 *"""Сортирует целые числа в бинарном файле."""* # Чтение чисел из бинарного файла  
 with open(file\_name, 'rb') as f:  
 numbers = []  
 while byte := f.read(4): # Читаем по 4 байта (размер int)  
 number = struct.unpack('i', byte)[0]  
 numbers.append(number)  
  
 # Сортировка списка чисел  
 numbers.sort()  
  
 # Запись отсортированных чисел обратно в бинарный файл  
 with open(file\_name, 'wb') as f:  
 for number in numbers:  
 f.write(struct.pack('i', number))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 sort\_binary\_file('numbers.bin') # Сортируем числа в бинарном файле

**Объяснение программы:**

1. **Генерация случайных чисел**:

В модуле generate\_numbers.py используется функция generate\_random\_numbers, которая принимает имя файла и количество чисел для генерации.

Случайные числа генерируются с помощью random.randint(0, 1000000) и записываются в бинарный файл с помощью struct.pack('i', number), который упаковывает число в формат int.

1. **Сортировка чисел**:

В модуле sort\_numbers.py функция sort\_binary\_file открывает бинарный файл, читает целые числа по 4 байта (размер одного целого числа), распаковывает их и добавляет в список.

После этого список сортируется с помощью метода sort().

Отсортированные числа записываются обратно в тот же бинарный файл.

**Как использовать программу:**

1. Сохраните оба модуля (generate\_numbers.py и sort\_numbers.py) в одной директории.
2. Сначала запустите модуль генерации случайных чисел:

python generate\_numbers.py

Это создаст файл numbers.bin с 10 случайными целыми числами.

1. Затем запустите модуль сортировки:

python sort\_numbers.py

Это отсортирует числа в файле numbers.bin.