МИНОБРНАУКИ РОССИИ

РГУ НЕФТИ И ГАЗА (НИУ) ИМЕНИ И.М. ГУБКИНА

|  |  |  |
| --- | --- | --- |
| Факультет | **Автоматики и вычислительной техники** |  |
| Кафедра | **Информатики** |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Оценка комиссии: |  | | | Рейтинг: |  |
| Подписи членов комиссии: | | | | | |
|  | |  | Сидоров В.В. | | |
| (подпись) | |  | (фамилия, имя, отчество) | | |
|  | |  |  | | |
| (подпись) | |  | (фамилия, имя, отчество) | | |
|  | | | | | |
| (дата) | | | | | |
|  | |  |  | | |

**КУРСОВАЯ РАБОТА**

|  |  |
| --- | --- |
| по дисциплине | Основы алгоритмизации и программирования |
|  | |

|  |  |
| --- | --- |
| на тему | Разработка программы регистрации и аутентификации |
| пользователей | |
|  | |

|  |  |  |  |
| --- | --- | --- | --- |
| «К ЗАЩИТЕ» |  | ВЫПОЛНИЛ: |  |
|  |  | Студент группы | **АА-24-10** |
|  |  |  | (номер группы) |
|  |  | Колесников С.И. | |
| (должность, ученая степень; фамилия, и.о.) |  | (фамилия, имя, отчество) | |
|  |  |  | |
| (подпись) |  | (подпись) | |
|  |  |  | |
| (дата) |  | (дата) | |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Москва, 20 | 25 |  |

Оглавление

[1. Постановка задачи 3](#_Toc197694777)

[2. Алгоритм работы программы 4](#_Toc197694778)

[3. Принцип работы XOR-шифрования 7](#_Toc197694779)

[4. Код программы с пояснениями 10](#_Toc197694780)

[4.1. Код 10](#_Toc197694781)

[4.2. Ссылка на файл .py 10](#_Toc197694782)

[5. Отдельная функция шифратора/дешифратора с Base64 10](#_Toc197694783)

[5.1. Код 11](#_Toc197694784)

[5.2. Ссылка на файл .py 11](#_Toc197694785)

[6. Скриншоты работы программы 12](#_Toc197694786)

[7. Литература 15](#_Toc197694787)

# 1. Постановка задачи

Задача состояла в разработке программы регистрации пользователя. Необходимо запросить информацию: Ф.И.О., дата рождения, пол, город, email, логин, пароль, подтверждение пароля. Проверить корректность введенных данных. Затем запомнить их в связном списке, при этом пароль должен быть закодирован с помощью битовой операции XOR (исключающее ИЛИ) (восстановление производится также с помощью этой операции). Реализовать возможность запроса логина и пароля пользователя и при корректном вводе, вывести сообщение о входе в личный кабинет. Список пользователей должен сохраняться в файле и загружаться из файла при запуске программы Программа должна предоставлять возможность запроса логина и пароля пользователя. При корректном вводе должно выводиться сообщение об успешном входе в личный кабинет, где отображается информация о пользователе. Список пользователей должен сохраняться в файле и загружаться из файла при запуске программы.

Графический интерфейс пользователя (GUI) реализован с использованием библиотеки Tkinter.

# 2. Алгоритм работы программы

Программа имеет графический интерфейс и управляется событиями (нажатия кнопок, ввод данных). Основные этапы работы:

1. **Инициализация программы:**
   * Загрузка списка зарегистрированных пользователей из файла users.data. Если файл отсутствует, он будет создан при первой регистрации.
   * Создание главного окна приложения и фреймов для различных экранов (вход, регистрация, информация о пользователе).
   * Отображение начального экрана – формы входа.
2. **Основной цикл (управляется Tkinter mainloop()):**
   * Программа ожидает действий пользователя.
3. **Процесс регистрации нового пользователя:**
   * Пользователь переходит на экран регистрации.
   * Пользователь вводит данные: Фамилия, Имя, Отчество, Дата рождения, Пол, Город, Email, Логин, Пароль, Подтверждение пароля.
   * **Валидация данных:**
     + Проверка на заполненность обязательных полей.
     + Проверка на совпадение пароля и его подтверждения.
     + Проверка на уникальность введенного логина среди уже зарегистрированных пользователей.
   * **Кодирование пароля:** Введенный пароль кодируется с помощью операции XOR с использованием предопределенного ключа.
   * **Сохранение пользователя:**
     + Данные нового пользователя (включая закодированный пароль) добавляются в общий список пользователей в памяти.
     + Обновленный список пользователей сохраняется в файл users.data, перезаписывая его.
   * Вывод сообщения об успешной регистрации.
   * Автоматический переход на экран входа.
4. **Процесс входа пользователя (аутентификация):**
   * Пользователь вводит логин и пароль на экране входа.
   * **Поиск пользователя:** Программа ищет пользователя с введенным логином в загруженном списке.
   * **Проверка пароля:**
     + Если пользователь найден, его сохраненный закодированный пароль извлекается.
     + К сохраненному закодированному паролю повторно применяется операция XOR с тем же ключом (для "дешифрования").
     + "Дешифрованный" пароль сравнивается с паролем, введенным пользователем.
   * **Результат входа:**
     + **Успешный вход:** Если пароли совпадают, выводится приветственное сообщение. Затем отображается экран с личной информацией пользователя (ФИО, дата рождения, город и т.д.).
     + **Неудачный вход:** Если пользователь не найден или пароли не совпадают, выводится сообщение об ошибке.
5. **Отображение информации о пользователе:**
   * После успешного входа открывается экран "Личный кабинет".
   * На экране отображаются данные вошедшего пользователя: ФИО, дата рождения, пол, город, email, логин.
   * Доступна кнопка "Выйти" для возврата на экран входа.
6. **Сохранение данных:**
   * Данные пользователей (список словарей) сохраняются в текстовый файл users.data в формате CSV (значения, разделенные точкой с запятой). Каждая строка файла представляет одного пользователя.
   * Пароль сохраняется в закодированном виде.
7. **Выход из программы:**
   * Закрытие главного окна приложения завершает выполнение программы.

# 3. Принцип работы XOR-шифрования

XOR (исключающее ИЛИ) — это битовая операция, которая является основой простого симметричного алгоритма шифрования.

**Принцип действия:**

1. **Представление данных:** Исходный текст (пароль) и ключ шифрования представляются в виде последовательностей битов. На практике это обычно означает работу с числовыми кодами символов (например, ASCII или Unicode).
2. **Операция XOR:** Для каждого символа исходного текста выполняется операция XOR с соответствующим символом ключа.

Таблица истинности XOR (A XOR B = C):

|  |  |  |
| --- | --- | --- |
| A | B | A XOR B = C |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

A ^ B = (!A and B) or (A or !B)

Результаты применения XOR:

* A ^ A = 0
* A ^ 0 = A
* (A ^ B) ^ C = A ^ (B ^ C)

1. **Результат:** Полученная последовательность битов (или символов, соответствующих новым кодам) является зашифрованным текстом.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Text | A1 | A2 | A3 | A4 |
| ^ | ↓ | ↓ | ↓ | ↓ |
| Key | B1 | B2 | B3 | B4 |
| = | ↓ | ↓ | ↓ | ↓ |
| Result | C1 | C2 | C3 | C4 |
| ^ | ↓ | ↓ | ↓ | ↓ |
| Key | B1 | B2 | B3 | B4 |
| = | ↓ | ↓ | ↓ | ↓ |
| Result | A1 | A2 | A3 | A4 |

Размер ключа обычно фиксирован, а также его размер часто меньше размера текста для шифрования. Разберём на примере. Предположим, у нас имеется текст для шифрования длиной в 4 байта. Индексы i этого массива меняются от 0 до 3. Ключ для шифрования имеет длину 2 байта. Таким образом к первым двум байтам текста применяется исключающее ИЛИ с первыми двумя байтами ключа. К третьему байту текста нужно применять XOR уже первый байт ключа и т.д. Определяется порядок по формуле: остаток от деления индекса на длину ключа.

i % len(key)

0 % 2 = 0

1 % 2 = 1

2 % 2 = 0

3 % 2 = 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 |
| Text | A1 | A2 | A3 | A4 |
| ^ | ↓ | ↓ |  |  |
| Key | B1 | B2 |  |  |
| = | ↓ | ↓ |  |  |
| Result | C1 | C2 | C3 | C4 |

1. **Дешифрование:** Уникальным свойством операции XOR является ее обратимость при повторном применении с тем же ключом:
   * C XOR B = (A XOR B) XOR B = A  
     Таким образом, для дешифрования зашифрованного текста достаточно снова применить к нему операцию XOR с тем же самым ключом, который использовался для шифрования.

**В данной программе:**

* Функция xor\_cipher(text, key) реализует этот принцип.
* Она берет каждый символ входной строки text и символ из строки key.
* Получает их числовые представления (коды Unicode) с помощью функции ord().
* Применяет операцию XOR (^ в Python) к этим кодам.
* Преобразует полученный код обратно в символ с помощью функции chr().
* Все полученные символы объединяются в итоговую строку (зашифрованную или расшифрованную).

**Важно:**

* XOR-шифрование является **симметричным**, т.е. один и тот же ключ используется как для шифрования, так и для дешифрования.
* Это очень **простой** алгоритм, но он **не является криптостойким** для защиты важной информации в реальных системах. Он уязвим для различных атак, особенно если ключ короткий или используется повторно. В данной программе он используется для демонстрации принципа и выполнения учебной задачи.
* Результатом операции XOR над кодами символов могут быть коды непечатных или управляющих символов. При сохранении в текстовый файл или выводе на экран это может приводить к некорректному отображению (пустые знаки или "красные значки").

# 4. Код программы с пояснениями

## 4.1. Код

import csv # Модуль для работы с CSV файлами (хранение данных пользователей)

import tkinter as tk # Модуль для создания графического интерфейса пользователя (GUI)

from tkinter import messagebox # Модуль для отображения стандартных диалоговых окон (сообщения, ошибки)

# --- Константы и глобальные переменные ---

FILENAME = "users.data" # Имя файла для хранения данных пользователей. Файл будет в папке запуска скрипта.

XOR\_KEY = "SimpleKey"   # Простой ключ для XOR "шифрования". В реальных системах использовать нельзя!

user\_list = []          # Глобальный список для хранения данных всех пользователей в памяти.

current\_logged\_in\_user = None # Переменная для хранения данных текущего вошедшего пользователя.

# --- Функции для работы с данными ---

def xor\_cipher(text, key):

  """

  Кодирует или декодирует текст с помощью битовой операции XOR.

  Принимает строку 'text' и строку 'key'.

  Возвращает строку, являющуюся результатом операции XOR.

  Внимание: результат может содержать непечатные символы,

  так как операция XOR применяется к кодам символов.

  """

  # Если ключ короче текста, он будет циклически повторяться.

  repeated\_key = (key \* (len(text) // len(key) + 1))[:len(text)]

  # Применяем операцию XOR к кодам Unicode каждого символа текста и ключа.

  # ord(char) - возвращает Unicode код символа.

  # chr(code) - возвращает символ по его Unicode коду.

  # ^ - оператор XOR.

  result = ''.join(chr(ord(a) ^ ord(b)) for a, b in zip(text, repeated\_key))

  return result

def load\_users(filename):

  """

  Загружает список пользователей из указанного файла.

  Если файл не существует, он будет создан при первой попытке записи (режим 'a+').

  """

  users = [] # Локальный список для загруженных пользователей

  # Открываем файл в режиме 'a+' (добавление и чтение).

  # Файл создается, если не существует.

  # newline='' и encoding='utf-8' важны для корректной работы с CSV и кириллицей.

  with open(filename, 'a+', newline='', encoding='utf-8') as file:

      file.seek(0) # Перемещаем курсор в начало файла для чтения существующих записей.

      reader = csv.reader(file, delimiter=';') # Создаем объект для чтения CSV, разделитель - точка с запятой.

      for row in reader: # Читаем файл построчно

          if row and len(row) == 9: # Проверяем, что строка не пустая и содержит 9 полей

              # Создаем словарь с данными пользователя

              user\_data = {

                  "last\_name": row[0],

                  "first\_name": row[1],

                  "middle\_name": row[2],

                  "birth\_date": row[3],

                  "gender": row[4],

                  "city": row[5],

                  "email": row[6],

                  "login": row[7],

                  "password\_xor\_encoded": row[8] # Пароль хранится в "зашифрованном" XOR виде

              }

              users.append(user\_data) # Добавляем пользователя в список

  return users

def save\_users(users, filename):

  """

  Сохраняет текущий список пользователей в указанный файл.

  Файл перезаписывается при каждом сохранении (режим 'w').

  """

  # Открываем файл в режиме 'w' (запись). Если файл существует, он будет перезаписан.

  with open(filename, 'w', newline='', encoding='utf-8') as file:

      writer = csv.writer(file, delimiter=';') # Создаем объект для записи CSV.

      for user\_data\_item in users: # Проходим по каждому пользователю в списке

          # Записываем данные пользователя в файл одной строкой

          writer.writerow([

              user\_data\_item["last\_name"],

              user\_data\_item["first\_name"],

              user\_data\_item["middle\_name"],

              user\_data\_item["birth\_date"],

              user\_data\_item["gender"],

              user\_data\_item["city"],

              user\_data\_item["email"],

              user\_data\_item["login"],

              user\_data\_item["password\_xor\_encoded"] # Сохраняем XOR "зашифрованный" пароль

          ])

def find\_user\_by\_login(users\_list\_to\_search, login\_to\_find):

  """

  Ищет пользователя в предоставленном списке по логину.

  Возвращает словарь с данными пользователя, если найден, иначе None.

  """

  for user\_data\_item in users\_list\_to\_search:

    if user\_data\_item["login"] == login\_to\_find:

      return user\_data\_item # Пользователь найден

  return None # Пользователь не найден

# --- Функции для GUI (Графического Интерфейса Пользователя) ---

def clear\_entries(entries\_list\_to\_clear):

    """Очищает содержимое полей ввода (tk.Entry) из предоставленного списка."""

    for entry\_widget in entries\_list\_to\_clear:

        if isinstance(entry\_widget, tk.Entry): # Проверяем, что это поле ввода

            entry\_widget.delete(0, tk.END) # Удаляем текст от начала (0) до конца (END)

    # Специальная очистка для Radiobutton выбора пола

    if 'gender\_var' in globals() and isinstance(gender\_var, tk.StringVar):

        gender\_var.set("") # Сбрасываем значение переменной, управляющей Radiobutton

def show\_frame(frame\_to\_display):

    """

    Показывает указанный фрейм (экран) и скрывает все остальные основные фреймы.

    """

    login\_frame.pack\_forget()      # Скрываем фрейм входа

    register\_frame.pack\_forget()   # Скрываем фрейм регистрации

    user\_info\_frame.pack\_forget()  # Скрываем фрейм информации о пользователе

    # Отображаем нужный фрейм, растягивая его на все доступное пространство

    frame\_to\_display.pack(fill="both", expand=True, padx=20, pady=20)

def display\_user\_info(user\_data\_to\_display):

    """

    Заполняет и отображает экран с информацией о вошедшем пользователе.

    """

    global current\_logged\_in\_user # Обращаемся к глобальной переменной

    current\_logged\_in\_user = user\_data\_to\_display # Сохраняем данные вошедшего пользователя

    # Очищаем предыдущие метки с данными пользователя, если они были

    # Это нужно, чтобы при повторном входе информация не дублировалась

    for widget in user\_info\_display\_frame.winfo\_children(): # Получаем всех потомков фрейма

        # Удаляем только те метки, которые мы пометили как 'is\_data\_label'

        if isinstance(widget, tk.Label) and hasattr(widget, 'is\_data\_label'):

            widget.destroy()

    # Список кортежей: (Текст метки, Значение из данных пользователя)

    info\_labels\_config = [

        ("Фамилия:", user\_data\_to\_display.get("last\_name", "N/A")),

        ("Имя:", user\_data\_to\_display.get("first\_name", "N/A")),

        ("Отчество:", user\_data\_to\_display.get("middle\_name", "N/A") or "Отсутствует"),

        ("Дата рождения:", user\_data\_to\_display.get("birth\_date", "N/A")),

        ("Пол:", user\_data\_to\_display.get("gender", "N/A")),

        ("Город:", user\_data\_to\_display.get("city", "N/A")),

        ("Email:", user\_data\_to\_display.get("email", "N/A")),

        ("Логин:", user\_data\_to\_display.get("login", "N/A")),

    ]

    # Динамически создаем и размещаем метки с информацией

    for i, (text, value) in enumerate(info\_labels\_config):

        # Метка с названием поля

        label\_text = tk.Label(user\_info\_display\_frame, text=text, font=("Arial", 11), anchor="w")

        label\_text.grid(row=i, column=0, padx=5, pady=2, sticky="w") # Размещаем в сетке

        label\_text.is\_data\_label = True # Помечаем метку для последующей очистки

        # Метка со значением поля

        label\_value = tk.Label(user\_info\_display\_frame, text=value, font=("Arial", 11, "bold"), anchor="w")

        label\_value.grid(row=i, column=1, padx=5, pady=2, sticky="w")

        label\_value.is\_data\_label = True # Помечаем метку

    show\_frame(user\_info\_frame) # Показываем фрейм с информацией

def handle\_login():

    """Обрабатывает нажатие кнопки 'Войти'."""

    login\_attempt = login\_entry\_login.get() # Получаем введенный логин

    password\_attempt = password\_entry\_login.get() # Получаем введенный пароль

    # Проверка, что поля не пустые

    if not login\_attempt or not password\_attempt:

        messagebox.showerror("Ошибка входа", "Пожалуйста, введите логин и пароль.")

        return

    # Ищем пользователя по логину

    user\_account = find\_user\_by\_login(user\_list, login\_attempt)

    if user\_account: # Если пользователь найден

        # "Дешифруем" сохраненный пароль (применяем XOR еще раз)

        stored\_password\_decoded = xor\_cipher(user\_account["password\_xor\_encoded"], XOR\_KEY)

        if password\_attempt == stored\_password\_decoded: # Сравниваем пароли

            # Формируем имя для приветствия

            user\_name\_for\_greeting = user\_account.get('first\_name', 'Пользователь')

            if user\_account.get('last\_name'):

                 user\_name\_for\_greeting = f"{user\_account.get('first\_name', '')} {user\_account.get('last\_name', '')}".strip()

            # Показываем сообщение об успешном входе

            messagebox.showinfo("Успешный вход", f"Добро пожаловать, {user\_name\_for\_greeting}!")

            clear\_entries([login\_entry\_login, password\_entry\_login]) # Очищаем поля ввода

            display\_user\_info(user\_account) # Показываем информацию о пользователе

        else:

            messagebox.showerror("Ошибка входа", "Неверный пароль.") # Пароли не совпали

    else:

        messagebox.showerror("Ошибка входа", "Пользователь с таким логином не найден.") # Пользователь не найден

def handle\_register():

    """Обрабатывает нажатие кнопки 'Зарегистрироваться' на форме регистрации."""

    # Получаем данные из всех полей ввода на форме регистрации

    last\_name = last\_name\_entry\_reg.get()

    first\_name = first\_name\_entry\_reg.get()

    middle\_name = middle\_name\_entry\_reg.get()

    birth\_date = birth\_date\_entry\_reg.get()

    gender = gender\_var.get() # Значение из Radiobutton

    city = city\_entry\_reg.get()

    email = email\_entry\_reg.get()

    login = login\_entry\_reg.get()

    password = password\_entry\_reg.get()

    password\_confirm = password\_confirm\_entry\_reg.get()

    # Проверка на заполненность обязательных полей

    if not all([last\_name, first\_name, birth\_date, gender, city, email, login, password, password\_confirm]):

        messagebox.showerror("Ошибка регистрации", "Пожалуйста, заполните все обязательные поля (кроме Отчества).")

        return

    # Проверка совпадения паролей

    if password != password\_confirm:

        messagebox.showerror("Ошибка регистрации", "Пароли не совпадают.")

        return

    # Проверка, не занят ли логин

    if find\_user\_by\_login(user\_list, login):

        messagebox.showerror("Ошибка регистрации", f"Пользователь с логином '{login}' уже существует.")

        return

    # "Шифрование" пароля с помощью XOR

    password\_xor\_encoded = xor\_cipher(password, XOR\_KEY)

    # Создание словаря с данными нового пользователя

    new\_user = {

        "last\_name": last\_name,

        "first\_name": first\_name,

        "middle\_name": middle\_name,

        "birth\_date": birth\_date,

        "gender": gender,

        "city": city,

        "email": email,

        "login": login,

        "password\_xor\_encoded": password\_xor\_encoded

    }

    user\_list.append(new\_user) # Добавляем нового пользователя в общий список

    save\_users(user\_list, FILENAME) # Сохраняем обновленный список в файл

    messagebox.showinfo("Успешная регистрация", "Пользователь успешно зарегистрирован!")

    # Очищаем все поля на форме регистрации

    clear\_entries([

        last\_name\_entry\_reg, first\_name\_entry\_reg, middle\_name\_entry\_reg,

        birth\_date\_entry\_reg, city\_entry\_reg,

        email\_entry\_reg, login\_entry\_reg, password\_entry\_reg,

        password\_confirm\_entry\_reg

    ])

    gender\_var.set("") # Сброс выбора пола

    show\_frame(login\_frame) # Переход на экран входа

def handle\_logout():

    """Обрабатывает нажатие кнопки 'Выйти' на экране информации о пользователе."""

    global current\_logged\_in\_user

    current\_logged\_in\_user = None # Сбрасываем данные о вошедшем пользователе

    show\_frame(login\_frame) # Показываем экран входа

# --- Основная функция для настройки GUI и запуска приложения ---

def main():

    # Объявляем глобальными переменные, которые используются в других функциях (обработчиках)

    # Это необходимо, так как эти виджеты и списки создаются в main(),

    # а используются в функциях, вызываемых по событиям.

    global window, login\_frame, register\_frame, user\_info\_frame, user\_info\_display\_frame

    global login\_entry\_login, password\_entry\_login

    global last\_name\_entry\_reg, first\_name\_entry\_reg, middle\_name\_entry\_reg

    global birth\_date\_entry\_reg, gender\_var, city\_entry\_reg, email\_entry\_reg

    global login\_entry\_reg, password\_entry\_reg, password\_confirm\_entry\_reg

    global user\_list # user\_list загружается здесь и используется глобально

    # --- Создание главного окна ---

    window = tk.Tk()

    window.title("Система регистрации и входа (Простой XOR)")

    window.geometry("450x650") # Устанавливаем размер окна

    # --- Создание основных фреймов (экранов) ---

    login\_frame = tk.Frame(window)      # Фрейм для экрана входа

    register\_frame = tk.Frame(window)   # Фрейм для экрана регистрации

    user\_info\_frame = tk.Frame(window)  # Фрейм для отображения информации о пользователе

    # --- Виджеты для экрана входа (login\_frame) ---

    login\_label\_login = tk.Label(login\_frame, text="Вход в систему", font=("Arial", 16, "bold"))

    login\_label\_login.pack(pady=20) # pack - менеджер геометрии, размещает виджеты друг под другом

    tk.Label(login\_frame, text="Логин:", font=("Arial", 12)).pack(pady=5)

    login\_entry\_login = tk.Entry(login\_frame, width=30, font=("Arial", 12))

    login\_entry\_login.pack(pady=5)

    tk.Label(login\_frame, text="Пароль:", font=("Arial", 12)).pack(pady=5)

    password\_entry\_login = tk.Entry(login\_frame, show="\*", width=30, font=("Arial", 12)) # show="\*" скрывает вводимые символы

    password\_entry\_login.pack(pady=10)

    login\_button = tk.Button(login\_frame, text="Войти", command=handle\_login, width=15, height=2, font=("Arial", 12))

    login\_button.pack(pady=10)

    go\_to\_register\_button = tk.Button(login\_frame, text="Зарегистрироваться", command=lambda: show\_frame(register\_frame), width=20, font=("Arial", 10))

    go\_to\_register\_button.pack(pady=5)

    # --- Виджеты для экрана регистрации (register\_frame) ---

    # Используем grid - менеджер геометрии, который размещает виджеты в виде таблицы (строки и столбцы)

    register\_label\_reg = tk.Label(register\_frame, text="Регистрация", font=("Arial", 16, "bold"))

    register\_label\_reg.grid(row=0, column=0, columnspan=3, pady=15, sticky="n") # columnspan=3 - занять 3 колонки

    current\_row = 1 # Переменная для отслеживания текущей строки в grid

    # Поля для ФИО

    tk.Label(register\_frame, text="Фамилия\*:", font=("Arial", 11)).grid(row=current\_row, column=0, padx=5, pady=5, sticky="w") # sticky="w" - прижать к западу (левому краю)

    last\_name\_entry\_reg = tk.Entry(register\_frame, width=30, font=("Arial", 11))

    last\_name\_entry\_reg.grid(row=current\_row, column=1, padx=5, pady=5)

    current\_row += 1

    tk.Label(register\_frame, text="Имя\*:", font=("Arial", 11)).grid(row=current\_row, column=0, padx=5, pady=5, sticky="w")

    first\_name\_entry\_reg = tk.Entry(register\_frame, width=30, font=("Arial", 11))

    first\_name\_entry\_reg.grid(row=current\_row, column=1, padx=5, pady=5)

    current\_row += 1

    tk.Label(register\_frame, text="Отчество:", font=("Arial", 11)).grid(row=current\_row, column=0, padx=5, pady=5, sticky="w")

    middle\_name\_entry\_reg = tk.Entry(register\_frame, width=30, font=("Arial", 11))

    middle\_name\_entry\_reg.grid(row=current\_row, column=1, padx=5, pady=5)

    current\_row += 1

    # Остальные поля

    tk.Label(register\_frame, text="Дата рождения\*:", font=("Arial", 11)).grid(row=current\_row, column=0, padx=5, pady=5, sticky="w")

    birth\_date\_entry\_reg = tk.Entry(register\_frame, width=30, font=("Arial", 11))

    birth\_date\_entry\_reg.grid(row=current\_row, column=1, padx=5, pady=5)

    tk.Label(register\_frame, text="(ДД.ММ.ГГГГ)", font=("Arial", 9)).grid(row=current\_row, column=2, padx=2, pady=5, sticky="w") # Подсказка формата

    current\_row += 1

    tk.Label(register\_frame, text="Пол\*:", font=("Arial", 11)).grid(row=current\_row, column=0, padx=5, pady=5, sticky="w")

    gender\_var = tk.StringVar(value="") # Переменная для хранения значения Radiobutton

    gender\_frame\_ui = tk.Frame(register\_frame) # Фрейм для группировки Radiobutton

    tk.Radiobutton(gender\_frame\_ui, text="Мужской", variable=gender\_var, value="Мужской", font=("Arial", 11)).pack(side=tk.LEFT)

    tk.Radiobutton(gender\_frame\_ui, text="Женский", variable=gender\_var, value="Женский", font=("Arial", 11)).pack(side=tk.LEFT)

    gender\_frame\_ui.grid(row=current\_row, column=1, padx=5, pady=5, sticky="w")

    current\_row += 1

    tk.Label(register\_frame, text="Город\*:", font=("Arial", 11)).grid(row=current\_row, column=0, padx=5, pady=5, sticky="w")

    city\_entry\_reg = tk.Entry(register\_frame, width=30, font=("Arial", 11))

    city\_entry\_reg.grid(row=current\_row, column=1, padx=5, pady=5)

    current\_row += 1

    tk.Label(register\_frame, text="Email\*:", font=("Arial", 11)).grid(row=current\_row, column=0, padx=5, pady=5, sticky="w")

    email\_entry\_reg = tk.Entry(register\_frame, width=30, font=("Arial", 11))

    email\_entry\_reg.grid(row=current\_row, column=1, padx=5, pady=5)

    current\_row += 1

    tk.Label(register\_frame, text="Логин\*:", font=("Arial", 11)).grid(row=current\_row, column=0, padx=5, pady=5, sticky="w")

    login\_entry\_reg = tk.Entry(register\_frame, width=30, font=("Arial", 11))

    login\_entry\_reg.grid(row=current\_row, column=1, padx=5, pady=5)

    current\_row += 1

    tk.Label(register\_frame, text="Пароль\*:", font=("Arial", 11)).grid(row=current\_row, column=0, padx=5, pady=5, sticky="w")

    password\_entry\_reg = tk.Entry(register\_frame, width=30, font=("Arial", 11)) # Пароль виден при регистрации

    password\_entry\_reg.grid(row=current\_row, column=1, padx=5, pady=5)

    current\_row += 1

    tk.Label(register\_frame, text="Подтвердите пароль\*:", font=("Arial", 11)).grid(row=current\_row, column=0, padx=5, pady=5, sticky="w")

    password\_confirm\_entry\_reg = tk.Entry(register\_frame, width=30, font=("Arial", 11)) # Пароль виден

    password\_confirm\_entry\_reg.grid(row=current\_row, column=1, padx=5, pady=5)

    current\_row += 1

    tk.Label(register\_frame, text="\* - обязательные поля", font=("Arial", 9)).grid(row=current\_row, column=0, columnspan=3, padx=5, pady=10, sticky="w")

    current\_row += 1

    register\_button\_reg = tk.Button(register\_frame, text="Зарегистрироваться", command=handle\_register, width=20, height=2, font=("Arial", 12))

    register\_button\_reg.grid(row=current\_row, column=0, columnspan=3, pady=15)

    current\_row += 1

    go\_to\_login\_button\_reg = tk.Button(register\_frame, text="Уже есть аккаунт? Войти", command=lambda: show\_frame(login\_frame), width=25, font=("Arial", 10))

    go\_to\_login\_button\_reg.grid(row=current\_row, column=0, columnspan=3, pady=5)

    # --- Виджеты для экрана информации о пользователе (user\_info\_frame) ---

    user\_info\_label = tk.Label(user\_info\_frame, text="Личный кабинет", font=("Arial", 16, "bold"))

    user\_info\_label.pack(pady=15)

    user\_info\_display\_frame = tk.Frame(user\_info\_frame) # Внутренний фрейм для динамического размещения меток с данными

    user\_info\_display\_frame.pack(pady=10, padx=10, fill="x")

    logout\_button = tk.Button(user\_info\_frame, text="Выйти", command=handle\_logout, width=15, height=2, font=("Arial", 12))

    logout\_button.pack(pady=20)

    # --- Загрузка пользователей и запуск главного цикла приложения ---

    user\_list = load\_users(FILENAME) # Загружаем пользователей из файла

    print(f"Загружено {len(user\_list)} пользователей из файла '{FILENAME}'.") # Информационное сообщение в консоль

    show\_frame(login\_frame) # Показываем начальный экран (вход)

    window.mainloop() # Запускаем главный цикл обработки событий Tkinter

# --- Точка входа в программу ---

# Этот блок гарантирует, что функция main() будет вызвана только тогда,

# когда скрипт запускается напрямую (а не импортируется как модуль).

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

    main()

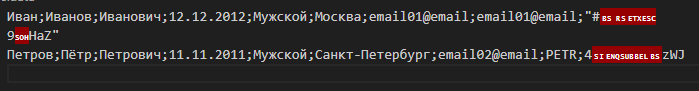
## 4.2. Ссылка на файл .py

[Ссылка](kursach_02.py)

# 5. Отдельная функция шифратора/дешифратора с Base64

В ходе улучшения хранения паролей была разработана отдельная функция, которая комбинирует XOR-шифрование с кодированием Base64. Цель Base64 — представить результат XOR-операции (который может содержать непечатные символы) в виде строки, состоящей только из безопасных для текстового хранения символов (латинские буквы, цифры, +, /, =). Это решает проблему "красных значков" и некорректного отображения зашифрованного пароля в текстовых файлах. Функция представляет собой полезное дополнение для понимания того, как происходит процесс отображения зашифрованных данных.

База данных без Base64:



Видны строки с данными пользователей, разделенные точкой с запятой. В поле пароля (последнее поле) видны символы, которые могут отображаться некорректно (например, "красные значки", как BS, NUL и т.д.), что демонстрирует результат прямого XOR-шифрования без последующего кодирования в Base64. Это подтверждает, что пароль хранится в "зашифрованном" виде, но его представление может быть нечитаемым для человека и содержать непечатные символы.

База данных с Base64:



**Принцип работы xor\_base64\_cipher:**

* **Шифрование (action='encode')**:
  1. Исходный текст и ключ преобразуются в байтовые последовательности (используя кодировку UTF-8).
  2. К байтам текста применяется операция XOR с байтами ключа.
  3. Полученная байтовая последовательность (результат XOR) кодируется в формат Base64. Base64 преобразует произвольные двоичные данные в текстовую строку, состоящую из ограниченного набора символов (A-Z, a-z, 0-9, +, /).
  4. Эта строка Base64 возвращается как результат.
* **Дешифрование (action='decode')**:
  1. Входная строка (предположительно, в формате Base64) преобразуется в байты.
  2. Байты декодируются из Base64, восстанавливая байтовую последовательность, которая была результатом исходной операции XOR.
  3. К этой восстановленной байтовой последовательности снова применяется операция XOR с тем же ключом.
  4. Полученные в результате байты (которые должны соответствовать исходному тексту) декодируются обратно в строку (используя UTF-8).

Эта комбинация позволяет хранить "зашифрованный" пароль в виде обычной текстовой строки, избегая проблем с непечатными символами.

## 5.1. Код

import base64

def xor\_base64\_cipher(text\_to\_process, key, action='encode'):

    """

    Шифрует или дешифрует текст, используя XOR, а затем Base64.

    Упрощенная версия без try-except.

    :param text\_to\_process: Строка для шифрования или строка Base64 для дешифрования.

    :param key: Ключ для операции XOR (строка).

    :param action: 'encode' для шифрования, 'decode' для дешифрования.

    :return: Зашифрованная строка Base64 или расшифрованная исходная строка.

             Вернет None или вызовет ошибку, если действие некорректно или данные неверны.

    """

    # Преобразуем ключ в байты один раз

    key\_bytes = key.encode('utf-8')

    key\_len = len(key\_bytes)

    if action == 'encode':

        # 1. Преобразовать исходный текст в байты

        text\_bytes = text\_to\_process.encode('utf-8')

        # 2. Применить XOR. Создаем список байтов результата XOR

        xor\_result\_bytes\_list = []

        for i, b in enumerate(text\_bytes):

            xor\_result\_bytes\_list.append(b ^ key\_bytes[i % key\_len])

        xor\_result\_bytes = bytes(xor\_result\_bytes\_list) # Преобразуем список в байтовую строку

        # 3. Закодировать результат XOR в Base64

        base64\_encoded\_bytes = base64.b64encode(xor\_result\_bytes)

        # 4. Преобразовать байты Base64 в строку для возврата

        return base64\_encoded\_bytes.decode('utf-8')

    elif action == 'decode':

        # 1. Преобразовать строку Base64 (которая является text\_to\_process) в байты

        base64\_to\_decode\_bytes = text\_to\_process.encode('utf-8')

        # 2. Декодировать из Base64 (результат - байты после XOR)

        # Если text\_to\_process не является корректной Base64 строкой, здесь будет ошибка.

        xor\_result\_bytes\_after\_b64decode = base64.b64decode(base64\_to\_decode\_bytes)

        # 3. Применить XOR еще раз к результату (байты)

        # Создаем список байтов результата XOR

        original\_bytes\_list = []

        for i, b in enumerate(xor\_result\_bytes\_after\_b64decode):

            original\_bytes\_list.append(b ^ key\_bytes[i % key\_len])

        original\_bytes\_after\_xor = bytes(original\_bytes\_list) # Преобразуем список в байтовую строку

        # 4. Преобразовать исходные байты обратно в строку

        # Если байты не являются корректной UTF-8 последовательностью, здесь будет ошибка.

        return original\_bytes\_after\_xor.decode('utf-8')

    else:

        print(f"Ошибка: Некорректное действие '{action}'. Используйте 'encode' или 'decode'.")

        return None

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

    my\_secret\_text = "ghfhjkm123"

    my\_key = "SimpleKey"

    print(f"Исходный текст: {my\_secret\_text}")

    print(f"Ключ: {my\_key}\n")

    # Шифрование

    encrypted\_text = xor\_base64\_cipher(my\_secret\_text, my\_key, action='encode')

    if encrypted\_text: # Проверяем, что шифрование вернуло результат

        print(f"Зашифрованный текст (XOR + Base64): {encrypted\_text}")

        # Дешифрование

        decrypted\_text = xor\_base64\_cipher(encrypted\_text, my\_key, action='decode')

        if decrypted\_text: # Проверяем, что дешифрование вернуло результат

            print(f"Расшифрованный текст: {decrypted\_text}\n")

            if my\_secret\_text == decrypted\_text:

                print("Проверка: Шифрование и дешифрование прошли успешно!")

            else:

                print("Проверка: Ошибка! Исходный и расшифрованный тексты не совпадают.")

        else:

            print("Ошибка при дешифровании.")

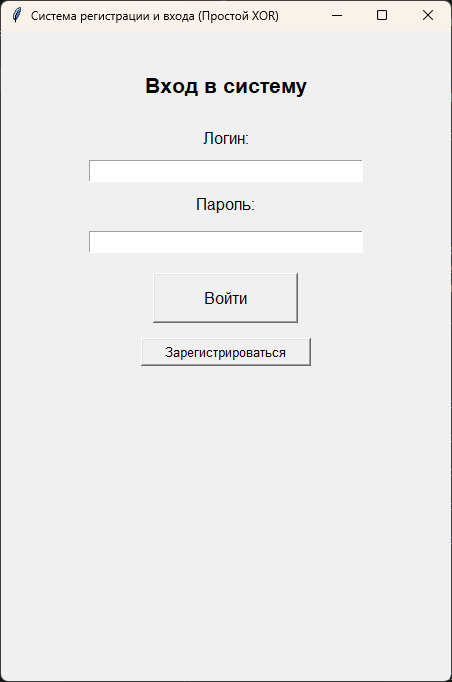
    else:

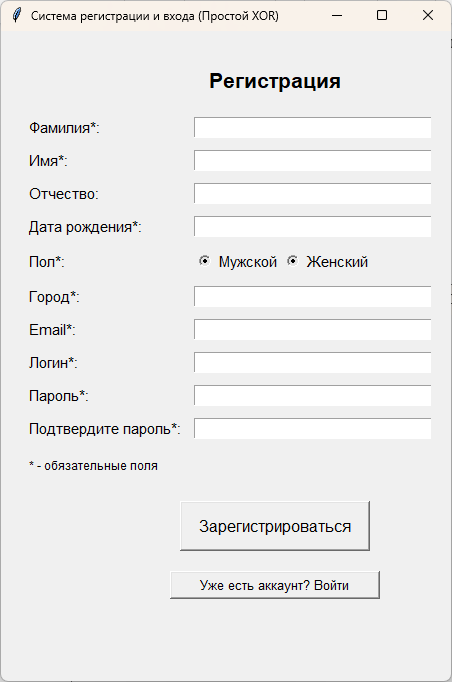
        print("Ошибка при шифровании.")

## 5.2. Ссылка на файл .py

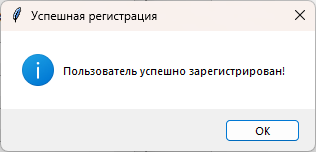
[Ссылка](xor_cipher.py)

# 6. Скриншоты работы программы

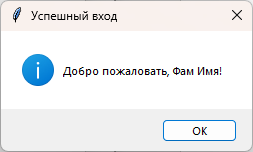
1. Окно входа в систему
2. Окно регистрации нового пользователя



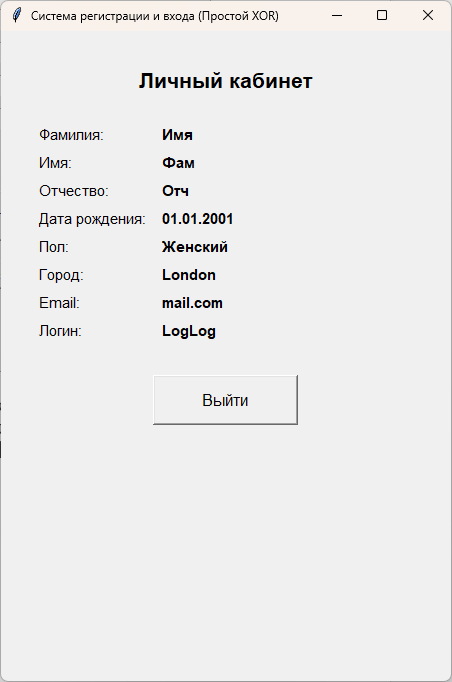
1. Сообщение об успешной регистрации



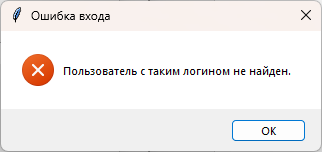
1. Сообщение об успешном входе

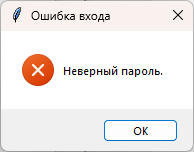


1. Окно «Личный кабинет» с информацией о пользователе



1. Сообщения об ошибках





# 7. Литература

1. Книги
   * Мэтиз, Эрик. *Изучаем Python: программирование игр, визуализация данных, веб-приложения.* 3-е изд. – СПб.: Питер, 2023. – 688 с.: ил.
2. Интернет-ресурсы (дата обращения 09.05.2025)
   * Официальная документация Python: <https://docs.python.org/3/>
   * Документация по модулю tkinter: <https://docs.python.org/3/library/tkinter.html>
   * Документация по модулю csv: <https://docs.python.org/3/library/csv.html>
   * Документация по модулю base64 (для отдельной функции): <https://docs.python.org/3/library/base64.html>
   * Видео по XOR шифрованию: <https://www.youtube.com/watch?v=f8HIUXkAHWo>