ПРАКТИЧЕСКОЕ ЗАДАНИЕ

Настройка Git-аккаунта, создание репозитория, разработка и публикация консольной формы регистрации (Python)

Цель

- завести аккаунт на Git-сервере (https://magicgit.sytes.net/users/sign_in),
- создать удалённый репозиторий,
- связать его с локальным проектом на ПК,
- настроить файл. gitignore
- реализовать консольную программу «Регистрация пользователя» с валидацией полей и обработкой исключений,
- фиксация зависимостей pip freeze > requirements.txt и опубликовать проект в удалённый репозиторий, после чего проект будет проверен преподавателем.

1. Аккаунт и удалённый репозиторий

Заведите/подтвердите аккаунт на Git-сервере (https://magicgit.sytes.net/users/sign_in).

Пример для GitLab: создайте проект («New project/repository») https://docs.gitlab.com/user/project/

Скопируйте URL удалённого репозитория (HTTPS).

2. Локальный проект и привязка к Git

В папке, где будут храниться проекты:

1) Клонирование пустого удалённого репозитория ИЛИ

git clone <URL_удалённого_репозитория>

2) (Альтернатива) Локальная инициализация и привязка к уже созданному удалённому

git init

git remote add origin <URL_удалённого_репозитория>

Справка по git clone и базовым операциям, см. файл «Менеджер пакетов и гит.docx»

3. Виртуальное окружение и зависимости

Можно либо использовать программные функции VSCode или Pycharm, либо в корне проекта выполнить следующие команды:

Создать и активировать виртуальное окружение (Windows, Git Bash)

python -m venv .venv

source .venv/Scripts/activate # для Git Bash/PowerShell: .venv\Scripts\activate

Важно: по завершении работы зафиксируйте зависимости:

pip freeze > requirements.txt

4. Задание по программированию

Файл register.py (в корне проекта).

Задача реализовать «форму регистрации» в консоли с пошаговыми подсказками и валидацией полей

Ник**,** Почта**,** Телефон**,** Пароль

Все проверки должны быть изолированы в try/except, с понятными пользователю сообщениями об ошибке и повторным запросом ввода до тех пор, пока значение не пройдёт валидацию.

Программа при запуске:

- Описывает требования к каждому полю.
- Запрашивает данные по шагам, повторяет запросы до валидного ввода.
- Выводит «чистое» резюме с маскировкой почты/телефона.
- Не показывает трейсбеки; все ошибки перехватываются и транслируются в понятные сообщения.

4.1. Требования к полям

Ник (username)

- Длина от **3 до 20** символов.
- **Первый символ** — буква латиницы или подчёркивание (_).
- Остальные символы — **только** латинские буквы, цифры или подчёркивание.
- **Без пробелов** и других спецсимволов.
- Не входит в список зарезервированных: {"admin", "root", "system"}.

Почта (email) (в задании оставил подсказки по применяемым функциям)

- Нет пробельных символов: ни ' ' ни \t, \n **(проверка any(ch.isspace() for ch in s) == False).**
- Ровно **одна** @: **s.count("@") == 1.**
- До @ есть непустая локальная часть, после @ — непустой домен.
- В домене есть точка. не в начале/конце домена и не две подряд **(простая проверка на '.' in domain и что ни одна часть после split('.') не пустая).**
- Общая длина ≤ **254** символов.
- Запрет служебных символов в простом приближении: "<>()[]{},;: \\" не встречаются (по желанию).

Телефон (в задании оставил подсказки по применяемым функциям)

- Строка начинается с '+'.
- После + — **только цифры** (str.isdigit() для хвоста).
- Количество цифр после +: **10–14**.
- **Не** начинается с +0.
- Не все цифры одинаковые (отсечь +1111111111): множество цифр имеет размер> 1.

Пароль

- Есть **строчная** буква (any(c.islower() for c in pwd)).
- Есть **заглавная** буква (any(c.isupper() for c in pwd)).

- Есть **цифра** (any(c.isdigit() for c in pwd)).
- Есть **спецсимвол** из заданного набора, например: !@#$%^&*()-_=+[]{};:,.? (проверка через in).
- **Нет пробельных символов** (not any(ch.isspace() for ch in pwd)).
- Пароль **не совпадает** с ником или почтой (в нижнем регистре для честного сравнения).

4.2. Обработка исключений (изоляция)

Для каждого поля реализовать функцию вида prompt_username(), prompt_email() и т. д., где внутри выполняется цикл while True, валидация и **локальный try/except**.

При ошибке — вывод понятного сообщения и повторный запрос.

4.3. Удобство для пользователя (UX в консоли)

1) Чёткие подсказки перед вводом каждого поля (укажите требования кратко).
2) После успешной регистрации — сводка введённых данных (email маскировать: u***@d***.com, телефон маскировать: +7***...).
3) Сообщения об ошибках — на «человеческом» языке, без трассбеков.
4) Короткая памятка
5) Пользуйтесь методами строк: str.isalpha(), str.isalnum(), str.isdigit(), str.isspace(), str.startswith(), in, count, split.
6) Стройте проверку как **последовательность простых условий** с ранним выходом: как только правило нарушено — выбрасывайте ValidationError с понятным текстом.
7) Для email делаем **разумный минимум**: одна @, есть точка в домене, никаких пробелов и пустых частей.

5. Первые коммиты и публикация

В корне проекта создайте файлы:

register.py — код,

README.md — краткая инструкция запуска (см. Ниже «Ожидаемый результат»),

.gitignore — добавьте .venv/, __pycache__/, .idea/ и т. п.

Далее:

git add .

git commit -m "Init: консольная форма регистрации с валидацией"

pip freeze > requirements.txt # фиксация зависимостей

git add requirements.txt

git commit -m "chore: add requirements.txt"

git push -u origin main # или master в зависимости от настройки

Ожидаемый результат (артефакты)

Удалённый репозиторий с минимальным набором файлов:

register.py

README.md (содержит: требования, как активировать venv, как запустить python register.py)

requirements.txt (получен командой pip freeze > requirements.txt)

.gitignore

README.md должен включать:

Registration App (CLI)

Требования

- Python 3.10+

- Windows + Git Bash

Установка

```bash

python -m venv .venv

source .venv/Scripts/activate    # Windows

pip install -r requirements.txt
```