ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
Кафедра Информатики и информационных технологий

направление подготовки
09.03.02 «Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № _8_

Дисциплина: Backend-разработка

Тема: Строки

Выполнил(а): студент(ка) группы __221-379__

_____Кодиров Жамшид Мурод угли_____
(Фамилия И.О.)

Проверил: _______ _____
(Фамилия И.О., степень, звание) **(Оценка)**

Дата, подпись _____ _____
(Дата) (Подпись)

Замечания:_____

_____

_____

Москва

2025

Задание

Цель работы: Написать код, который демонстрирует работу со строками в Python.

Должны быть реализованы следующие конструкции:

1. Функция, принимающая на вход некий набор параметров (минимум 3 параметра).
Внутри себя эта функция содержит заранее определённую строку, в которую можно подставлять значения переменных.
Функция возвращает строку, в которую вставлены значения, такие что
1.1. Как минимум, одно значение - это просто строка
1.2. Как минимум, одно значение - это результат арифметической операции
1.3. Как минимум, одно значение - это результат вызова другой функции

2. Функция, которая формирует строку, состоящую из повторений комбинации других строк.
Эта функция выводит получившуюся строку, где каждое повторение выводится на отдельной строке.

3. Функция, которая считает количество вхождений подстроки в строку без учёта регистра.

4. Функция, принимающая на вход строку и выводящая подстроку, содержащуюся между двумя индексами.
Индексы ДОЛЖНЫ быть больше нуля и меньше длины строки минус 1.
Тело функции ДОЛЖНО быть написано в одну строку.

5. Функция, принимающая на вход произвольное количество разных строк, где содержатся любые кириллические буквы, а также могут содержаться латинские буквы, но только такие, которые визуально неотличимы от кириллических. Регистр букв произвольный.
Эта функция ищет слова, в которых содержатся латинские буквы.
На выход возвращаются строки, где были обнаружены латинские символы и количество слов, в которых была обнаружена хотя бы одна латинская буква.

6. Функция, определяющая, является ли строка палиндромом (одинаково читается с начала и с конца).
Строка МОЖЕТ содержать как цифры, так и буквы.

7. Функция, принимающая на вход строку, содержащую несколько слов, которые разделены одним или несколькими пробелами.
У входной строки могут быть несколько пробелов в начале и в конце.
Функция убирает лишние пробелы: то есть все пробелы в начале и в конце

строки, а между словами оставляет только один пробел.
Функция возвращает длину строки после удаления лишних пробелов.

8. Функция, принимающая на вход строку, содержащую текст из нескольких предложений.
Функция заменяет символы окончания предложения на символ переноса строки.
Функция возвращает получившуюся строку.

9. Минимум 3 функции, содержащие произвольные алгоритмы работы со строками.
Функции ДОЛЖНЫ решать алгоритмы, отличные от реализованных в п. 1-8

10. Функция, которая последовательно вызывает ВСЕ вышесозданные функции.
Функция ДОЛЖНА завершаться корректно и НЕ ДОЛЖНА иметь необработанных исключений

Дополнительные требования:

Функции, созданные в шагах 1-9 ДОЛЖНЫ быть размещены в одном или нескольких отдельных файлах.

Функция из шага 10 ДОЛЖНА быть размещена в файле main.py.

В файле main.py ДОЛЖНА быть конструкция if __name__ == "__main__", внутри которой ДОЛЖНА вызываться функция из шага 10.

В комментариях к каждой функции ДОЛЖНО быть отмечено, к какому шагу относится эта функция.

КРАЙНЕ ЖЕЛАТЕЛЬНО, чтобы реализуемые функции имитировали какую-то реальную логику, были как-нибудь связаны между собой и содержали как можно меньше искусственных примеров.

ЗАПРЕЩАЕТСЯ полностью копировать примеры из лекции.

РАЗРЕШАЕТСЯ использовать примеры из лекции за основу.

Ход работы

1. **Создание проекта FastAPI**
   - Установлены необходимые зависимости:

```
pip install fastapi uvicorn
```

2. **Создание основного файла проекта (main.py)**

- Создан файл main.py со следующим содержимым:

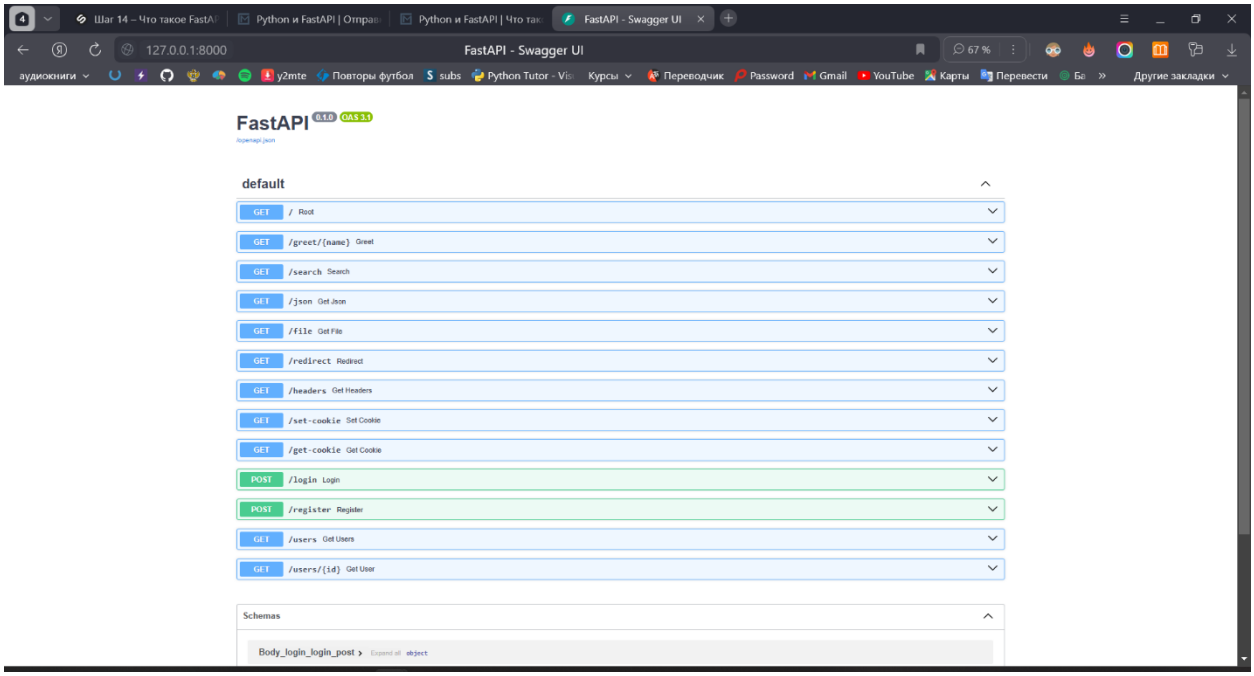
```
from fastapi import FastAPI, Form, Request, Response
from fastapi.responses import JSONResponse,
FileResponse, RedirectResponse
from pydantic import BaseModel
from typing import List
import os

app = FastAPI()

# Базовый маршрут
@app.get("/")
async def root():
    return {"message": "Hello, World!"}

# Обработка параметров
@app.get("/greet/{name}")
async def greet(name: str = "Alice"):
    return {"message": f"Hello, {name}!"}

@app.get("/search")
async def search(query: str = "python"):
    return {"message": f"You searched for: {query}"}

# Различные типы данных
@app.get("/json")
async def get_json():
    return {
        "name": "Alex Smith",
        "age": 25,
        "hobbies": ["coding", "hiking", "photography"]
    }

@app.get("/file")
async def get_file():
    with open("sample.txt", "w") as f:
        f.write("Welcome to FastAPI tutorial\nLine 2:
Example text")
    return FileResponse("sample.txt")
```

```python
@app.get("/redirect")
async def redirect():
    return RedirectResponse(url="/")

# Работа с заголовками и куками
@app.get("/headers")
async def get_headers(request: Request):
    return dict(request.headers)

@app.get("/set-cookie")
async def set_cookie():
    response = JSONResponse(content={"message": "Cookie
has been set successfully"})
    response.set_cookie(key="username",
value="alex123")
    return response

@app.get("/get-cookie")
async def get_cookie(request: Request):
    username = request.cookies.get("username", "guest")
    return {"username": username}

# Обработка данных запроса
@app.post("/login")
async def login(username: str = Form(default="john"),
password: str = Form(default="secret123")):
    return {"message": f"Welcome, {username}!"}

class RegisterData(BaseModel):
    username: str = "newuser"
    email: str = "newuser@example.com"
    password: str = "mypassword123"

@app.post("/register")
async def register(data: RegisterData):
    return {"message": f"User {data.username}
registered successfully!"}

# Работа с классами
```

```python
class User(BaseModel):
    id: int
    username: str
    email: str
    password: str

users = [
    User(id=1, username="mary",
email="mary@example.com", password="pass123"),
    User(id=2, username="peter",
email="peter@example.com", password="secure456")
]

@app.get("/users", response_model=List[User])
async def get_users():
    return users

@app.get("/users/{id}", response_model=User)
async def get_user(id: int = 1):
    for user in users:
        if user.id == id:
            return user
    return {"error": "User not found"}

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="127.0.0.1", port=8000)
```

3. **Запуск сервера**
   - Проверена доступность API по адресу http://127.0.0.1:8000/ через браузер.
4. **Тестирование маршрутов**
   - Проверен базовый маршрут: GET http://127.0.0.1:8000/ → {"message": "Hello, World!"}
   - Проверен маршрут с параметром пути: GET http://127.0.0.1:8000/greet/Bob → {"message": "Hello, Bob!"}
   - Проверен маршрут с параметром запроса: GET http://127.0.0.1:8000/search?query=fastapi → {"message": "You searched for: fastapi"}
   - Проверен JSON-ответ: GET http://127.0.0.1:8000/json → {"name": "Alex Smith", "age": 25, "hobbies": ["coding", "hiking", "photography"]}

- Проверен возврат файла: GET http://127.0.0.1:8000/file → скачан файл sample.txt
- Проверено перенаправление: GET http://127.0.0.1:8000/redirect → перенаправление на /
- Проверены заголовки: GET http://127.0.0.1:8000/headers → возвращены все заголовки запроса
- Установлена кука: GET http://127.0.0.1:8000/set-cookie → установлена кука username=alex123
- Проверена кука: GET http://127.0.0.1:8000/get-cookie → {"username": "alex123"}
- Проверен POST-запрос формы: POST http://127.0.0.1:8000/login с данными username=john&password=secret123 → {"message": "Welcome, john!"}
- Проверен POST-запрос JSON: POST http://127.0.0.1:8000/register с телом {"username": "newuser", "email": "newuser@example.com", "password": "mypassword123"} → {"message": "User newuser registered successfully!"}
- Проверен список пользователей: GET http://127.0.0.1:8000/users → возвращен список из двух пользователей
- Проверен конкретный пользователь: GET http://127.0.0.1:8000/users/1 → {"id": 1, "username": "mary", "email": "mary@example.com", "password": "pass123"}



Вывод

В ходе выполнения лабораторной работы был создан и протестирован проект на основе FastAPI. Реализованы все требуемые маршруты, включая обработку параметров пути и запросов, работу с различными типами ответов (JSON, файлы, перенаправления), управление заголовками и куками, а также

обработку данных формы и JSON. Использование классов Pydantic позволило структурировать данные пользователей и обеспечить их типизацию. API успешно запущен и протестирован на локальном сервере, все маршруты работают корректно с заданными примерами значений. FastAPI показал себя как удобный и мощный инструмент для быстрого создания веб-приложений с автоматической генерацией документации и встроенной поддержкой асинхронности.

Ссылка на Github: https://github.com/QodirovJM/BackendPython