# **1 ПРАКТИЧЕСКАЯ.**

Шаг 1: Скачивание и распаковка файлов

Шаг 2: Создание доп. Папок

Шаг 3: В файле httpd.conf через блокнот отредактировать:  
1. Define SRVROOT "c:/Apache24" в Define SRVROOT "c:/Server/bin/Apache24"

2. #ServerName [www.example.com:80](http://www.example.com:80) в ServerName localhost

3. DocumentRoot "${SRVROOT}/htdocs" в DocumentRoot "c:/Server/data/htdocs"

4. <Directory "${SRVROOT}/htdocs"> в <Directory "c:/Server/data/htdocs">

5. DirectoryIndex index.html в DirectoryIndex index.php index.html index.htm

6. AllowOverride None поменяйте состояние на All

7. #LoadModule rewrite\_module modules/mod\_rewrite.so нужно убрать #

Шаг 4: Windows PowerShell (администратор)

Ввести команды:

1- c:\Server\bin\Apache24\bin\httpd.exe -k install

2 - c:\Server\bin\Apache24\bin\httpd.exe –k start

3 - Появится оповещение Безопасности Windows, которое следует подтвердить, нажав «Разрешить доступ».

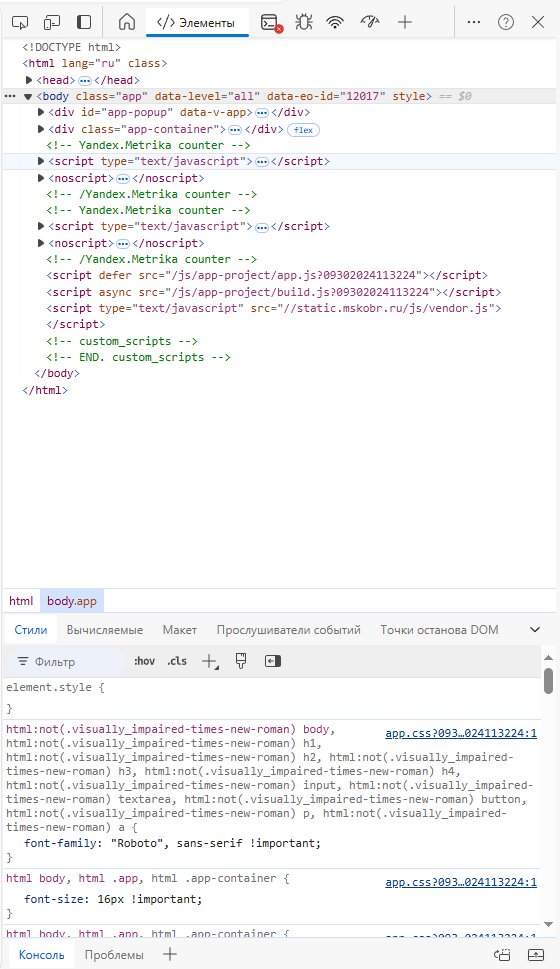
Шаг 5: Откройте любой браузер и перейдите по адресу <http://localhost/>.

# 2 ПРАКТИЧЕСКАЯ.

Выберите сайт для оптимизации

Нажмите F12

Откроется окно разработчика:



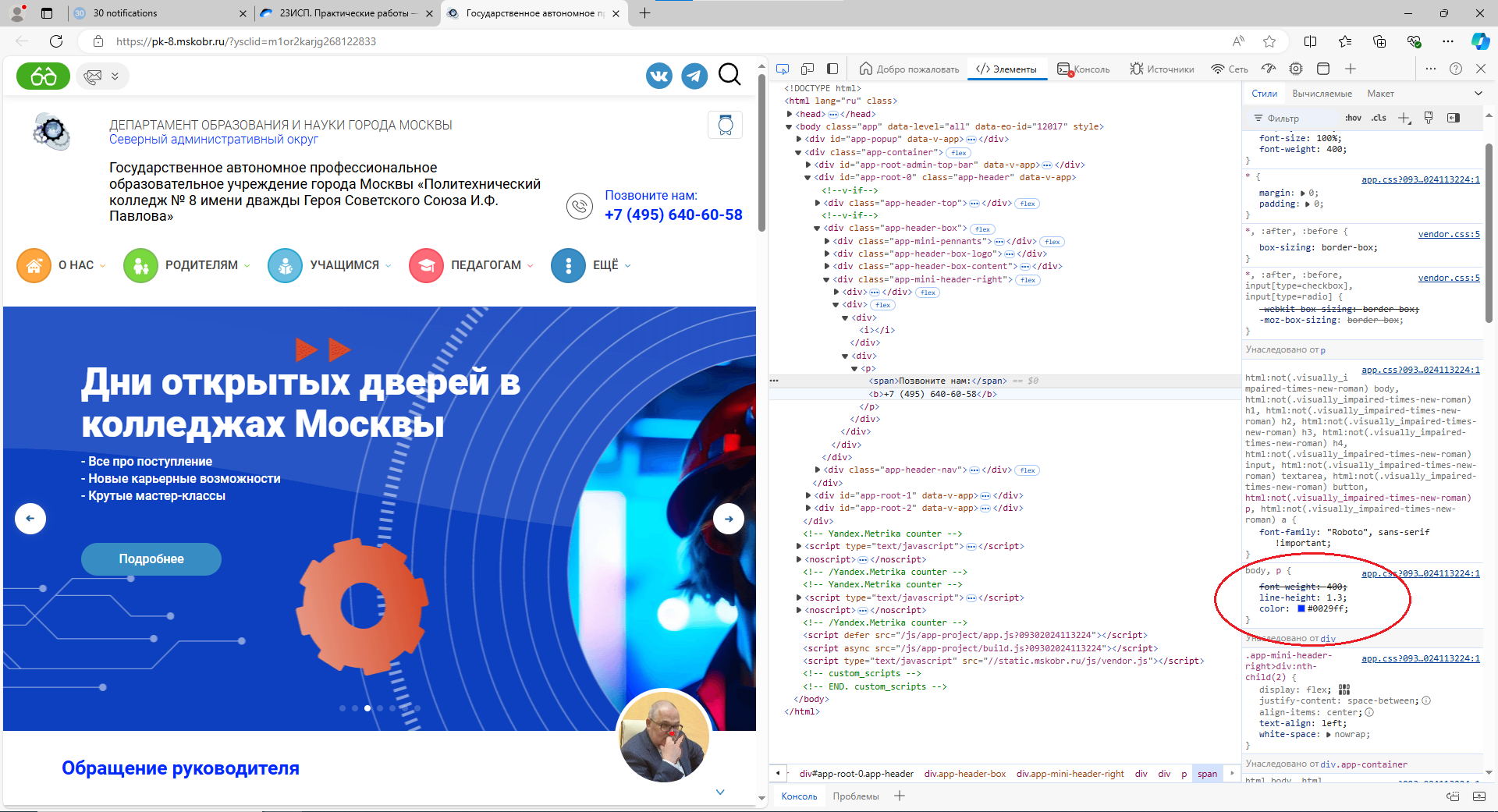
Выберете массив (Нажав Ctrl + Shift + C) с лишним наполнением сайта (изображение, текст и т.д.)

Изменим цвет текста:

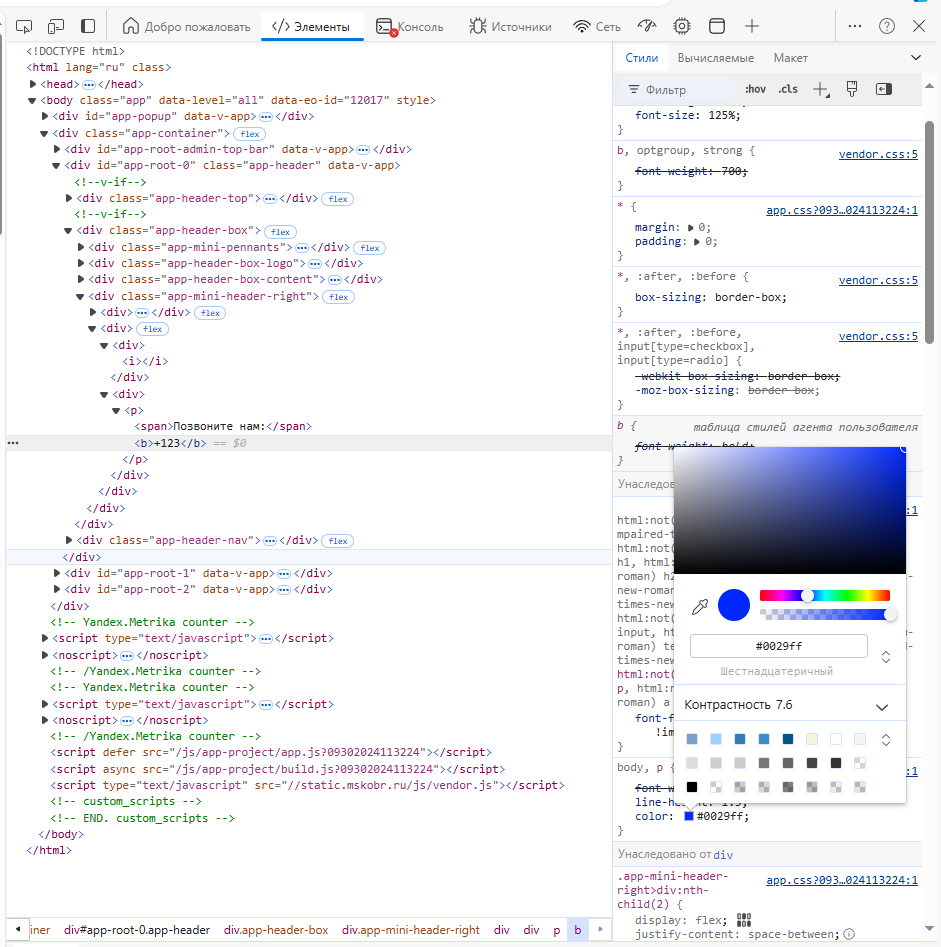
Наведитесь на текст

Нажмите ЛКМ

Откроется окно “Стили”



Измените цвет шрифта



Далее делайте по аналогии

# 3 ПРАКТИЧЕСКАЯ.

# 4 ПРАКТИЧЕСКАЯ.

КОД:   
import math

# Функция для расчета площади круга

def circle\_area(radius):

return math.pi \* (radius \*\* 2)

# Функция для расчета площади квадрата

def square\_area(side):

return side \*\* 2

# Функция для расчета площади прямоугольного треугольника

def triangle\_area(base, height):

return 0.5 \* base \* height

# Процедура для проверки, является ли число положительным

def is\_positive(number):

if number > 0:

print(f"{number} является положительным числом.")

else:

print(f"{number} не является положительным числом.")

# Процедура для проверки, могут ли два числа быть разделены без остатка

def delen(num1, num2):

if num2 == 0:

print("Нельзя делить на ноль!")

elif num1 % num2 == 0:

print(f"{num1} может быть разделен на {num2} без остатка.")

else:

print(f"{num1} не может быть разделен на {num2} без остатка.")

# Получение ввода от пользователя

radius = float(input("Введите радиус круга: "))

side = float(input("Введите сторону квадрата: "))

Osn = float(input("Введите основание треугольника: "))

height = float(input("Введите высоту треугольника: "))

num1 = float(input("Введите первое число: "))

num2 = float(input("Введите второе число: "))

# Вызов функций и процедур

print("Площадь круга:", circle\_area(radius))

print("Площадь квадрата:", square\_area(side))

print("Площадь треугольника:", triangle\_area(Osn, height))

is\_positive(radius)

is\_positive(side)

is\_positive(Osn)

is\_positive(height)

delen(num1, num2)

delen(num2, num1)  
  
Импорт модуля math:

import math — эта строка импортирует модуль math, который предоставляет доступ к математическим константам и функциям, таким как число pi.

Функции для расчета площадей:

circle\_area(radius):

Принимает радиус круга в качестве аргумента.

Вычисляет площадь круга по формуле math.pi \* (radius \*\* 2).

Возвращает рассчитанную площадь круга.

square\_area(side):

Принимает длину стороны квадрата в качестве аргумента.

Вычисляет площадь квадрата по формуле side \*\* 2.

Возвращает рассчитанную площадь квадрата.

triangle\_area(base, height):

Принимает основание и высоту треугольника в качестве аргументов.

Вычисляет площадь треугольника по формуле 0.5 \* base \* height.

Возвращает рассчитанную площадь треугольника.

Процедуры для проверки свойств чисел:

is\_positive(number):

Принимает число в качестве аргумента.

Проверяет, является ли число положительным (больше нуля).

Выводит соответствующее сообщение на экран.

delen(num1, num2):

Принимает два числа в качестве аргументов.

Проверяет, можно ли разделить num1 на num2 без остатка.

Выводит соответствующее сообщение на экран, включая проверку на деление на ноль.

Получение ввода от пользователя:

Код использует функцию input() для получения значений от пользователя для радиуса круга, стороны квадрата, основания и высоты треугольника, а также двух чисел для проверки делимости.

Функция float() преобразует введенные значения в числа с плавающей точкой.

Вызов функций и процедур:

После получения ввода от пользователя код вызывает функции для расчета площадей фигур и процедуры для проверки свойств чисел.

Результаты вычислений и проверки выводятся на экран с помощью функции print()

# 5 ПРАКТИЧЕСКАЯ.

a = 10

b = 5

print("Сложение:", a + b)

print("Вычитание:", a - b)

print("Умножение:", a \* b)

print("Деление:", a / b)

print("Модуль (остаток):", a % b)

print("Возведение в степень:", a \*\* b)

x = True

y = False

print("Логическое И (AND):", x and y)

print("Логическое ИЛИ (OR):", x or y)

print("Логическое НЕ (NOT):", not x)

num = int(input("Введите число: "))

if num % 2 == 0:

print("Число четное")

else:

print("Число нечетное")

num = int(input("Введите число: "))

if num > 10:

print("Число больше 10")

elif num < 5:

print("Число меньше 5")

else:

print("Число между 5 и 10")

num = int(input("Введите число: "))

if num % 3 == 0 and num % 5 == 0:

print("Число кратно 3 и 5")

else:

print("Число не кратно 3 и 5")  
  
Арифметические: Сложение, вычитание, умножение, деление, остаток от деления, возведение в степень.

Логические:

И (AND): возвращает True, если оба условия истинны.

ИЛИ (OR): возвращает True, если хотя бы одно условие истинно.

НЕ (NOT): инвертирует логическое значение.

Условные: if, elif, else: выполняют определенный код, если условие истинно.

Пример:

a = 10 - создается переменная a со значением 10.

if num % 2 == 0: - проверяет, является ли число четным.

# 6 ПРАКТИЧЕСКАЯ.

a = 10

b = 5

# Арифметические операции

addition = a + b

subtraction = a - b

multiplication = a \* b

division = a / b

modulus = a % b

# Вывод результатов

print(f"Сложение: {addition}")

print(f"Вычитание: {subtraction}")

print(f"Умножение: {multiplication}")

print(f"Деление: {division}")

print(f"Остаток от деления: {modulus}")

# Логические операторы

is\_greater = a > b

is\_equal = a == b

print(f"a больше b: {is\_greater}")

print(f"a равно b: {is\_equal}")

# Идентификаторы

x = 15

y = 25

name = "Пример"

sum\_xy = x + y

print(f"Сумма {name}: {sum\_xy}")

# Метки и операторы перехода

for i in range(10):

if i == 5:

print("Прыжок на 5, продолжаем...")

continue

if i == 8:

print("Выход из цикла на 8...")

break

print(i)

Арифметика: сложение, вычитание, умножение, деление, остаток от деления.

Сравнение: проверка, больше ли одно число другого, равно ли они.

Цикл: for повторяет код 10 раз.

Условные операторы: if проверяет условие, continue пропускает итерацию цикла, break прерывает цикл.

# 7 ПРАКТИЧЕСКАЯ.

my\_list = [15]

# Заполнение массива

my\_list.append(10)

my\_list.append("Hello")

my\_list.append(True)

# Перебор массива

for item in my\_list:

print(item)

# Вывод результата:

# 10

# Hello

# True

Объяснение:

Создание массива (списка):

my\_list = [] создает пустой список (массив) с именем my\_list.

Заполнение массива:

my\_list.append(10) добавляет число 10 в конец списка my\_list.

my\_list.append("Hello") добавляет строку “Hello” в конец списка.

my\_list.append(True) добавляет логическое значение True в конец списка.

Перебор массива:

for item in my\_list: - цикл for, который перебирает каждый элемент item в списке my\_list.

print(item) - выводит на экран текущий элемент item.

Другие способы заполнения массива:

Инициализация при создании:

my\_list = [10, "Hello", True]

Использование диапазона:

my\_list = list(range(5)) # Создает список [0, 1, 2, 3, 4]

Использование генератора списков:

my\_list = [i \* 2 for i in range(5)] # Создает список [0, 2, 4, 6, 8]

# 8 ПРАКТИЧЕСКАЯ.

matrix = []

print("Введите элементы матрицы 4x4:")

for i in range(4):

row = list(map(int, input().split()))

matrix.append(row)

# Вычисление сумм и произведений

sum\_second\_row = 0

product\_third\_column = 1

sum\_main\_diagonal = 0

product\_below\_main\_diagonal = 1

sum\_above\_main\_diagonal = 0

for i in range(4):

for j in range(4):

if i == 1: # Вторая строка

sum\_second\_row += matrix[i][j]

if j == 2: # Третий столбец

product\_third\_column \*= matrix[i][j]

if i == j: # Главная диагональ

sum\_main\_diagonal += matrix[i][j]

if i > j: # Элементы ниже главной диагонали

product\_below\_main\_diagonal \*= matrix[i][j]

if i < j: # Элементы выше главной диагонали

sum\_above\_main\_diagonal += matrix[i][j]

# Вывод результатов

print("Сумма элементов второй строки:", sum\_second\_row)

print("Произведение элементов третьего столбца:", product\_third\_column)

print("Сумма элементов главной диагонали:", sum\_main\_diagonal)

print("Произведение элементов ниже главной диагонали:", product\_below\_main\_diagonal)

print("Сумма элементов выше главной диагонали:", sum\_above\_main\_diagonal)

**Ввод матрицы:**

matrix = []: создается пустой список, который будет хранить матрицу.

print("Введите элементы матрицы 4x4:"): выводится сообщение для пользователя.

for i in range(4): цикл, который повторяется 4 раза (по количеству строк матрицы).

row = list(map(int, input().split())):

input(): считывает строку с клавиатуры, где пользователь вводит элементы одной строки матрицы, разделенные пробелами.

Инициализация переменных:

sum\_second\_row = 0: переменная для хранения суммы элементов второй строки (строка с индексом 1).

product\_third\_column = 1: переменная для хранения произведения элементов третьего столбца (столбец с индексом 2).

sum\_main\_diagonal = 0: переменная для хранения суммы элементов главной диагонали (элементы с одинаковыми индексами строк и столбцов).

product\_below\_main\_diagonal = 1: переменная для хранения произведения элементов ниже главной диагонали (элементы, где номер строки больше номера столбца).

sum\_above\_main\_diagonal = 0: переменная для хранения суммы элементов выше главной диагонали (элементы, где номер строки меньше номера столбца).

Вложенные циклы:

for i in range(4): внешний цикл по строкам матрицы (i - индекс строки).

for j in range(4): внутренний цикл по столбцам матрицы (j - индекс столбца).

Условия:

if i == 1: если номер строки (i) равен 1 (вторая строка), то добавляем элемент matrix[i][j] к сумме sum\_second\_row.

if j == 2: если номер столбца (j) равен 2 (третий столбец), то умножаем элемент matrix[i][j] на произведение product\_third\_column.

if i == j: если номер строки (i) равен номеру столбца (j) (главная диагональ), то добавляем элемент matrix[i][j] к сумме sum\_main\_diagonal.

if i > j: если номер строки (i) больше номера столбца (j) (ниже главной диагонали), то умножаем элемент matrix[i][j] на произведение product\_below\_main\_diagonal.

if i < j: если номер строки (i) меньше номера столбца (j) (выше главной диагонали), то добавляем элемент matrix[i][j] к сумме sum\_above\_main\_diagonal.

# 9 ПРАКТИЧЕСКАЯ.

import cmath # Импортируем модуль cmath для работы с комплексными числами

def solve\_quadratic\_equation(a, b, c):

if a == 0:

raise ValueError("Уравнение линейное, а не квадратное.")

discriminant = b\*\*2 - 4\*a\*c

if discriminant >= 0:

# Решение для действительных корней

x1 = (-b + discriminant\*\*0.5) / (2\*a)

x2 = (-b - discriminant\*\*0.5) / (2\*a)

return [x1, x2]

else:

# Решение для комплексных корней

x1 = (-b + cmath.sqrt(discriminant)) / (2\*a)

x2 = (-b - cmath.sqrt(discriminant)) / (2\*a)

return [x1, x2]

# Ввод коэффициентов уравнения

a = float(input("Введите коэффициент a: "))

b = float(input("Введите коэффициент b: "))

c = float(input("Введите коэффициент c: "))

# Вычисление и вывод корней

try:

roots = solve\_quadratic\_equation(a, b, c)

print("Корни уравнения:", roots)

except ValueError as e:

print(e)

Объяснение кода:

Импорт модуля cmath: Мы импортируем модуль cmath для работы с комплексными числами, поскольку дискриминант может быть отрицательным.

Функция solve\_quadratic\_equation:

Проверка на линейное уравнение: Если a равен 0, то функция вызывает исключение ValueError с сообщением об ошибке.

Вычисление дискриминанта: discriminant = b\*\*2 - 4\*a\*c.

Решения для действительных корней: Если дискриминант больше или равен 0, то функция вычисляет два действительных корня.

Решения для комплексных корней: Если дискриминант меньше 0, то функция вычисляет два комплексных корня с помощью модуля cmath.

Ввод коэффициентов: Программа запрашивает у пользователя коэффициенты a, b и c.

Вычисление корней: Программа пытается вычислить корни с помощью функции solve\_quadratic\_equation.

Обработка исключений: Если введенное значение a равно 0, программа выведет сообщение об ошибке.

Вывод корней: Программа выводит список корней уравнения, которые были вычислены функцией solve\_quadratic\_equation.

# 10 ПРАКТИЧЕСКАЯ.

from math import sin, cos, tan, radians, log10, log

print("Выберите операцию:")

print("1. Сложение")

print("2. Вычитание")

print("3. Умножение")

print("4. Деление")

print("5. Возведение в степень")

print("6. Вычисление синуса")

print("7. Вычисление косинуса")

print("8. Вычисление тангенса")

print("9. Десятичный логарифм")

print("10. Натуральный логарифм")

operation = int(input())

if operation in [1, 2, 3, 4, 5]:

a = float(input("Введите первое число: "))

b = float(input("Введите второе число: "))

if operation == 1:

result = a + b

print(f"Результат: {result}")

elif operation == 2:

result = a - b

print(f"Результат: {result}")

elif operation == 3:

result = a \* b

print(f"Результат: {result}")

elif operation == 4:

if b != 0:

result = a / b

print(f"Результат: {result}")

else:

print("Ошибка: Деление на ноль!")

elif operation == 5:

result = a \*\* b

print(f"Результат: {result}")

elif operation in [6, 7, 8]:

angle = float(input("Введите угол в градусах: "))

if operation == 6:

result = sin(radians(angle))

print(f"Синус угла: {result}")

elif operation == 7:

result = cos(radians(angle))

print(f"Косинус угла: {result}")

elif operation == 8:

if cos(radians(angle)) != 0:

result = tan(radians(angle))

print(f"Тангенс угла: {result}")

else:

print("Ошибка: Тангенс не определен (деление на ноль)!")

elif operation == 9:

x = float(input("Введите число для вычисления десятичного логарифма: "))

if x > 0:

result = log10(x)

print(f"Десятичный логарифм: {result}")

else:

print("Ошибка: Логарифм не определен для отрицательных чисел или нуля!")

elif operation == 10:

x = float(input("Введите число для вычисления натурального логарифма: "))

if x > 0:

result = log(x)

print(f"Натуральный логарифм: {result}")

else:

print("Ошибка: Логарифм не определен для отрицательных чисел или нуля!")

else:

print("Неверный выбор операции.")

Импорт библиотек

from math import sin, cos, tan, radians, log10, log

sin, cos, tan — для вычисления синуса, косинуса и тангенса угла.

radians — для преобразования градусов в радианы (так как тригонометрические функции в Python работают с радианами).

log10 — для вычисления десятичного логарифма.

log — для вычисления натурального логарифма.

Определение функции калькулятора

def calculator():

Мы определяем функцию calculator(), которая будет содержать весь код нашего калькулятора.

Вывод меню операций

print("Выберите операцию:")

print("1. Сложение")

print("2. Вычитание")

print("3. Умножение")

print("4. Деление")

print("5. Возведение в степень")

print("6. Вычисление синуса")

print("7. Вычисление косинуса")

print("8. Вычисление тангенса")

print("9. Десятичный логарифм")

print("10. Натуральный логарифм")

Здесь мы выводим список доступных операций для пользователя.

Ввод операции

operation = int(input())

Пользователь вводит номер операции, которую он хочет выполнить. Мы преобразуем ввод в целое число с помощью int().

Обработка арифметических операций

if operation in [1, 2, 3, 4, 5]:

a = float(input("Введите первое число: "))

b = float(input("Введите второе число: "))

Если пользователь выбрал одну из арифметических операций (сложение, вычитание, умножение, деление или возведение в степень), программа запрашивает два числа у пользователя и преобразует их в тип float.

Выполнение выбранной операции

if operation == 1:

result = a + b

# 11 ПРАКТИЧЕСКАЯ.

Создание аккаунта

Ссылка на репозиторий: [asernon/TEST\_SHARAGA: TEST (github.com)](https://github.com/asernon/TEST_SHARAGA)

# 12 ПРАКТИЧЕСКАЯ.

# 13 ПРАКТИЧЕСКАЯ.

Шаг 1: Инициализация локального репозитория

Создайте директорию:

mkdir my-project  
cd my-project

Инициализация репозитория:

git init

Шаг 2: Создание файлов и изменения

Создайте несколько файлов:

touch file1.txt  
touch file2.txt

Измените файл:

echo "Привет, мир!" > file1.txt

Шаг 3: Использование git status

git status

Вы должны увидеть вывод, похожий на этот:

On branch master  
No commits yet  
Changes not staged for commit:  
 (use "git add <file>..." to update what will be committed)  
 (use "git checkout -- <file>..." to discard changes in working directory)  
 modified: file1.txt  
Untracked files:  
 (use "git add <file>..." to include in what will be committed)  
 file2.txt

Шаг 4: Использование git diff

git diff

Вы увидите различия между текущим состоянием file1.txt и версией, которая хранится в репозитории (поскольку пока что коммитов нет, различия будут по сравнению с пустым файлом).

Шаг 5: Создание коммита

Добавление файлов в индекс:

git add .

(точка “.” означает “все файлы”)

Создание коммита:

git commit -m "Первый коммит"

Теперь у вас есть первый коммит в вашем репозитории.

git log: Используйте git log, чтобы просмотреть историю коммитов.

git reset HEAD~1: Откатитесь на один коммит назад (если нужно).

git branch: Создайте ветки для работы над различными частями проекта.

git checkout <branch\_name>: Переключитесь между ветками.

git merge: Объедините ветки

# 14 ПРАКТИЧЕСКАЯ.

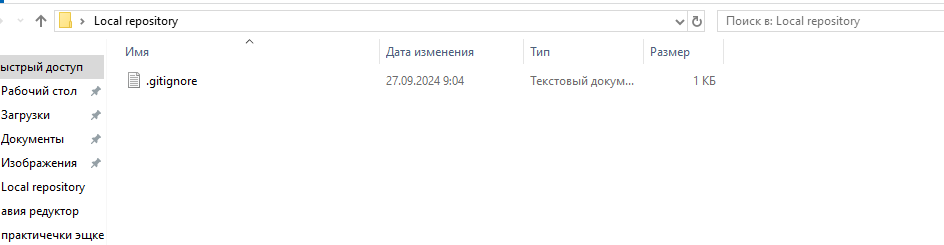
Шаг 1: Инициализируем репозиторий

Создайте новую директорию (папку) на вашем компьютере для

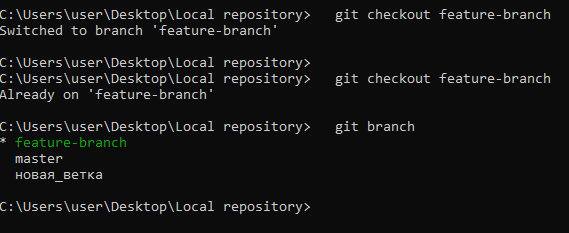
проекта, если она еще не создана.

Откройте командную строку или терминал в этой директории.

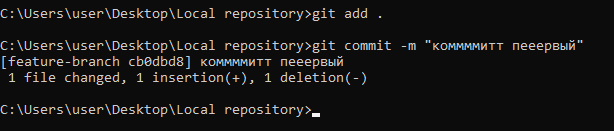
Инициализируйте новый локальный репозиторий с помощью команды:

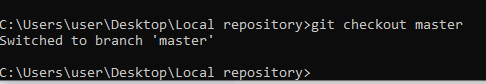
git init. 

Шаг 2: Создаем ветку и переключаемся на нее с помощью команд git branch feature-branch



Шаг 3. Создаем коммит

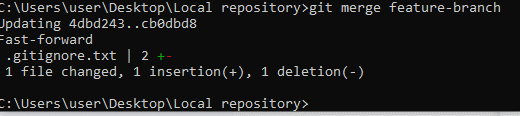


Шаг 4. Переключаемся на главную ветку 

Шаг 5. Объединение веток

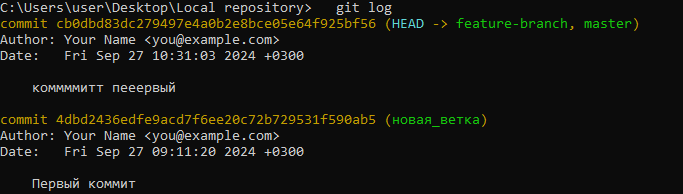
Используем команду git merge для объединения созданной ветки с

основной веткой (git merge feature-branch).

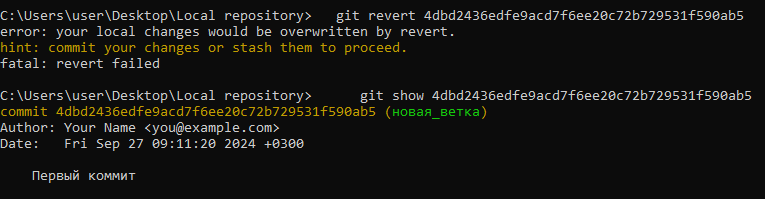


# 15 ПРАКТИЧЕСКАЯ.

Шаг 1: Создание нового коммита и получения списков коммитов



Шаг 2: Откат изменений



Шаг 3: Сброс изменений



ИТОГ

успешно выполнили задание по работе с Git, включая создание коммитов, откат изменений с помощью git revert и сброс изменений с помощью git reset --hard.

# 16 ПРАКТИЧЕСКАЯ.

**Шаг 1: Создание Cookie**

**HTML (index.html):**

<!DOCTYPE html>  
<html>  
<head>  
 <title>Создание Cookie</title>  
</head>  
<body>  
  
 <h1>Введите ваше имя:</h1>  
  
 <form id="user-form">  
 <input type="text" id="username" name="username" required>  
 <button type="submit">Отправить</button>  
 </form>  
  
 <p id="user-info"></p>  
  
 <script>  
 const form = document.getElementById('user-form');  
 const userInfo = document.getElementById('user-info');  
  
 form.addEventListener('submit', (event) => {  
 event.preventDefault();  
 const username = document.getElementById('username').value;  
  
 *// Создание Cookie*  
 document.cookie = `username=${username}; path=/;`;  
  
 *// Отображение имени в параграфе*  
 userInfo.textContent = `Привет, ${username}!`;  
 });  
 </script>  
  
</body>  
</html>

**Объяснение:**

1. **HTML-форма:** Форма содержит поле для ввода имени (username) и кнопку отправки.
2. **JavaScript:** При отправке формы:
   1. Получается значение из поля username.
   2. Создается Cookie с именем username и полученным значением.
   3. Выводится приветствие с именем пользователя в параграф user-info.

**Шаг 2: Изменение HTTP-заголовков ответа сервера**

**Node.js (server.js) с Express:**

const express = require('express');  
const app = express();  
  
app.get('/', (req, res) => {  
 *// Отключение кэширования*  
 res.setHeader('Cache-Control', 'no-store');   
 res.send('<h1>Привет, мир!</h1>');  
});  
  
app.listen(3000, () => {  
 console.log('Сервер запущен на порту 3000');  
});

**Объяснение:**

* Используем Express для создания веб-сервера.
* При обработке запроса GET на корневой маршрут (/):
  + Устанавливаем HTTP-заголовок Cache-Control со значением no-store. Это предотвращает кэширование ответа в браузере.

**Шаг 3: Работа с сессиями в веб-приложении**

**Node.js (server.js) с Express и сессиями:**

const express = require('express');  
const session = require('express-session');  
const app = express();  
  
app.use(session({  
 secret: 'my-secret-key', *// Используйте более надежный секретный ключ*  
 resave: false,  
 saveUninitialized: false  
}));  
  
app.get('/', (req, res) => {  
 res.send(`  
 <h1>Вход</h1>  
 <form action="/login" method="POST">  
 <input type="text" name="username" placeholder="Имя пользователя">  
 <button type="submit">Войти</button>  
 </form>  
 `);  
});  
  
app.post('/login', (req, res) => {  
 const username = req.body.username;  
 req.session.user = username; *// Сохранение имени в сессии*  
 res.redirect('/profile');  
});  
  
app.get('/profile', (req, res) => {  
 if (req.session.user) {  
 res.send(`  
 <h1>Профиль</h1>  
 <p>Привет, ${req.session.user}!</p>  
 <a href="/logout">Выйти</a>  
 `);  
 } else {  
 res.redirect('/');  
 }  
});  
  
app.get('/logout', (req, res) => {  
 req.session.destroy((err) => {  
 if (err) {  
 console.error(err);  
 }  
 res.redirect('/');  
 });  
});  
  
app.listen(3000, () => {  
 console.log('Сервер запущен на порту 3000');  
});

**Объяснение:**

1. **Инициализация сессий:** Используем express-session для работы с сессиями.
2. **Страница входа (/):** Отображаем форму входа.
3. **Обработка входа (/login):**
   1. Получаем имя пользователя из формы.
   2. Сохраняем имя в сессии (req.session.user).
   3. Перенаправляем пользователя на страницу профиля.
4. **Страница профиля (/profile):**
   1. Проверяем, существует ли req.session.user.
   2. Если да, то отображаем информацию о пользователе.
   3. Если нет, то перенаправляем на страницу входа.
5. **Выход (/logout):**
   1. Удаляем сессию.
   2. Перенаправляем пользователя на страницу входа.

**Запуск:**

1. Установите Node.js и npm.
2. Создайте файл server.js и скопируйте в него код.
3. Запустите сервер: node server.js.

# 17 ПРАКТИЧЕСКАЯ.

**Шаг 1: Чтение файла**

**HTML (index.html):**

<!DOCTYPE html>  
<html>  
<head>  
 <title>Чтение файла</title>  
</head>  
<body>  
  
 <h1>Чтение файла</h1>  
  
 <button id="read-file-button">Прочитать файл</button>  
 <p id="file-content"></p>  
  
 <script>  
 const readFileButton = document.getElementById('read-file-button');  
 const fileContent = document.getElementById('file-content');  
  
 readFileButton.addEventListener('click', () => {  
 fetch('/read-file')  
 .then(response => response.text())  
 .then(data => {  
 fileContent.textContent = data;  
 })  
 .catch(error => {  
 console.error('Ошибка при чтении файла:', error);  
 });  
 });  
 </script>  
  
</body>  
</html>

**Node.js (server.js) с Express:**

const express = require('express');  
const fs = require('fs');  
const app = express();  
  
app.get('/read-file', (req, res) => {  
 fs.readFile('path/to/your/file.txt', 'utf8', (err, data) => {  
 if (err) {  
 console.error('Ошибка при чтении файла:', err);  
 res.status(500).send('Ошибка при чтении файла');  
 } else {  
 res.send(data);  
 }  
 });  
});  
  
app.listen(3000, () => {  
 console.log('Сервер запущен на порту 3000');  
});

**Объяснение:**

* **HTML:** Кнопка “Прочитать файл” и параграф для вывода содержимого.
* **JavaScript:** При нажатии на кнопку:
  + Отправляется запрос GET на маршрут /read-file.
  + Полученный ответ (содержимое файла) отображается в параграфе.
* **Node.js:**
  + Используется модуль fs для работы с файлами.
  + fs.readFile() читает файл file.txt (замените на имя вашего файла).
  + Ответ отправляется клиенту.

**Шаг 2: Вывод содержимого файла**

Это уже выполнено в коде выше (содержимое файла выводится в параграф fileContent.)

**Шаг 3: Запись файла**

**HTML (index.html):**

<!DOCTYPE html>  
<html>  
<head>  
 <title>Запись файла</title>  
</head>  
<body>  
  
 <h1>Запись файла</h1>  
  
 <form id="file-upload-form" method="POST" enctype="multipart/form-data">  
 <input type="file" id="file-input" name="file">  
 <button type="submit">Отправить</button>  
 </form>  
  
</body>  
</html>

**Node.js (server.js) с Express:**

const express = require('express');  
const fs = require('fs');  
const multer = require('multer'); *// Для загрузки файлов*  
const app = express();  
  
const upload = multer({ dest: 'uploads/' }); *// Создаем папку для загрузок*  
  
app.post('/upload-file', upload.single('file'), (req, res) => {  
 const file = req.file;  
 if (file) {  
 fs.rename(file.path, `uploads/${file.originalname}`, (err) => {  
 if (err) {  
 console.error('Ошибка при сохранении файла:', err);  
 res.status(500).send('Ошибка при сохранении файла');  
 } else {  
 res.send('Файл успешно сохранен');  
 }  
 });  
 } else {  
 res.status(400).send('Файл не выбран');  
 }  
});  
  
app.listen(3000, () => {  
 console.log('Сервер запущен на порту 3000');  
});

**Объяснение:**

* **HTML:** Форма с полем file для выбора файла и кнопкой “Отправить”.
* **Node.js:**
  + Используем multer для обработки загрузок файлов.
  + upload.single('file') обрабатывает одно поле file в форме.
  + Файл сохраняется в папку uploads.
  + Переименовываем файл в оригинальное имя.
  + Отправляем ответ клиенту.

**Шаг 4: Удаление файла**

**HTML (index.html):**

<!DOCTYPE html>  
<html>  
<head>  
 <title>Удаление файла</title>  
</head>  
<body>  
  
 <h1>Удаление файла</h1>  
  
 <button id="delete-file-button">Удалить файл</button>  
  
 <script>  
 const deleteFileButton = document.getElementById('delete-file-button');  
  
 deleteFileButton.addEventListener('click', () => {  
 fetch('/delete-file')  
 .then(response => {  
 if (response.ok) {  
 alert('Файл удален');  
 } else {  
 console.error('Ошибка при удалении файла:', response.statusText);  
 }  
 })  
 .catch(error => {  
 console.error('Ошибка при удалении файла:', error);  
 });  
 });  
 </script>  
  
</body>  
</html>

**Node.js (server.js) с Express:**

*// ... (предыдущий код)*  
  
app.delete('/delete-file', (req, res) => {  
 fs.unlink('uploads/file.txt', (err) => { *// Удалить файл*  
 if (err) {  
 console.error('Ошибка при удалении файла:', err);  
 res.status(500).send('Ошибка при удалении файла');  
 } else {  
 res.send('Файл удален');  
 }  
 });  
});  
  
*// ... (остальной код)*

**Объяснение:**

* **HTML:** Кнопка “Удалить файл”.
* **JavaScript:** При нажатии на кнопку:
  + Отправляется запрос DELETE на маршрут /delete-file.
  + В ответ отображается сообщение об успешном удалении или об ошибке.
* **Node.js:**
  + Используется fs.unlink() для удаления файла.

**Шаг 5: Создание и управление каталогами**

**Node.js (server.js) с Express:**

*// ... (предыдущий код)*  
  
app.post('/create-directory', (req, res) => {  
 const directoryName = req.body.directoryName;  
 fs.mkdir(`uploads/${directoryName}`, { recursive: true }, (err) => { *// Создать каталог*  
 if (err) {  
 console.error('Ошибка при создании каталога:', err);  
 res.status(500).send('Ошибка при создании каталога');  
 } else {  
 res.send('Каталог успешно создан');  
 }  
 });  
});  
  
app.post('/move-directory', (req, res) => {  
 const oldPath = req.body.oldPath;  
 const newPath = req.body.newPath;  
 fs.rename(oldPath, newPath, (err) => { *// Переместить каталог*  
 if (err) {  
 console.error('Ошибка при перемещении каталога:', err);  
 res.status(500).send('Ошибка при перемещении каталога');  
 } else {  
 res.send('Каталог успешно перемещен');  
 }  
 });  
});  
  
app.delete('/delete-directory', (req, res) => {  
 const directoryPath = req.body.directoryPath;  
 fs.rmdir(directoryPath, { recursive: true }, (err) => { *// Удалить каталог*  
 if (err) {  
 console.error('Ошибка при удалении каталога:', err);  
 res.status(500).send('Ошибка при удалении каталога');  
 } else {  
 res.send('Каталог успешно удален');  
 }  
 });  
});  
  
*// ... (остальной код)*

**Объяснение:**

* **Node.js:**
  + fs.mkdir() создает каталог.
  + fs.rename() перемещает каталог.
  + fs.rmdir() удаляет каталог.

**HTML (index.html):**

Вам потребуется добавить HTML-формы для отправки запросов на создание, перемещение и удаление каталогов.

# 18 ПРАКТИЧЕСКАЯ.

**Шаг 1: Создание базы данных и таблицы**

**Пример с SQLite:**

*-- Создание базы данных (если не существует)*  
CREATE DATABASE IF NOT EXISTS my\_database;  
  
*-- Переключение на созданную базу данных*  
USE my\_database;  
  
*-- Создание таблицы users*  
CREATE TABLE users (  
 id INT PRIMARY KEY AUTO\_INCREMENT,  
 name VARCHAR(255) NOT NULL,  
 email VARCHAR(255) UNIQUE NOT NULL,  
 password VARCHAR(255) NOT NULL  
);

**Пример с MySQL:**

*-- Создание базы данных (если не существует)*  
CREATE DATABASE IF NOT EXISTS my\_database;  
  
*-- Переключение на созданную базу данных*  
USE my\_database;  
  
*-- Создание таблицы users*  
CREATE TABLE users (  
 id INT AUTO\_INCREMENT PRIMARY KEY,  
 name VARCHAR(255) NOT NULL,  
 email VARCHAR(255) UNIQUE NOT NULL,  
 password VARCHAR(255) NOT NULL  
);

**Шаг 2: Чтение данных**

**HTML (index.html):**

<!DOCTYPE html>  
<html>  
<head>  
 <title>Чтение данных из БД</title>  
</head>  
<body>  
  
 <h1>Список пользователей</h1>  
  
 <form id="user-search-form" method="GET">  
 <input type="text" name="search" placeholder="Поиск по имени">  
 <button type="submit">Искать</button>  
 </form>  
  
 <table id="user-table">  
 <thead>  
 <tr>  
 <th>ID</th>  
 <th>Имя</th>  
 <th>Email</th>  
 </tr>  
 </thead>  
 <tbody>  
 </tbody>  
 </table>  
  
 <script>  
 const userTable = document.getElementById('user-table').getElementsByTagName('tbody')[0];  
 const searchForm = document.getElementById('user-search-form');  
  
 function displayUsers(users) {  
 userTable.innerHTML = ''; *// Очищаем таблицу*  
 users.forEach(user => {  
 const row = userTable.insertRow();  
 row.insertCell().textContent = user.id;  
 row.insertCell().textContent = user.name;  
 row.insertCell().textContent = user.email;  
 });  
 }  
  
 searchForm.addEventListener('submit', (event) => {  
 event.preventDefault();  
 const searchTerm = event.target.elements.search.value;  
 fetch(`/users?search=${searchTerm}`)  
 .then(response => response.json())  
 .then(users => displayUsers(users))  
 .catch(error => console.error('Ошибка при чтении данных:', error));  
 });  
  
 *// Вывод всех пользователей при загрузке страницы*  
 fetch('/users')  
 .then(response => response.json())  
 .then(users => displayUsers(users))  
 .catch(error => console.error('Ошибка при чтении данных:', error));  
 </script>  
  
</body>  
</html>

**Node.js (server.js) с Express и SQLite:**

const express = require('express');  
const sqlite3 = require('sqlite3');  
const app = express();  
  
const db = new sqlite3.Database('./my\_database.db'); *// Подключение к базе*  
  
app.get('/users', (req, res) => {  
 const searchTerm = req.query.search;  
 const sql = searchTerm ? `SELECT \* FROM users WHERE name LIKE '%${searchTerm}%'` : 'SELECT \* FROM users';  
 db.all(sql, (err, rows) => {  
 if (err) {  
 console.error('Ошибка при чтении данных:', err);  
 res.status(500).send('Ошибка при чтении данных');  
 } else {  
 res.json(rows);  
 }  
 });  
});  
  
app.listen(3000, () => {  
 console.log('Сервер запущен на порту 3000');  
});

**Node.js (server.js) с Express и MySQL:**

const express = require('express');  
const mysql = require('mysql2');  
const app = express();  
  
const db = mysql.createConnection({  
 host: 'localhost',  
 user: 'your\_user',  
 password: 'your\_password',  
 database: 'my\_database'  
});  
  
db.connect((err) => {  
 if (err) {  
 console.error('Ошибка подключения к базе данных:', err);  
 } else {  
 console.log('Успешное подключение к базе данных!');  
 }  
});  
  
app.get('/users', (req, res) => {  
 const searchTerm = req.query.search;  
 const sql = searchTerm ? `SELECT \* FROM users WHERE name LIKE '%${searchTerm}%'` : 'SELECT \* FROM users';  
 db.query(sql, (err, rows) => {  
 if (err) {  
 console.error('Ошибка при чтении данных:', err);  
 res.status(500).send('Ошибка при чтении данных');  
 } else {  
 res.json(rows);  
 }  
 });  
});  
  
app.listen(3000, () => {  
 console.log('Сервер запущен на порту 3000');  
});

**Объяснение:**

* **HTML:** Форма для поиска пользователей по имени и таблица для вывода результатов.
* **JavaScript:**
  + При загрузке страницы и при отправке формы делается запрос к серверу.
  + Полученный ответ (массив пользователей) отображается в таблице.
* **Node.js:**
  + Подключение к базе данных SQLite (sqlite3) или MySQL (mysql2).
  + Выполнение запроса SELECT для получения пользователей.
  + Отправка данных в формате JSON клиенту.

**Шаг 3: Запись данных**

**HTML (index.html):**

<!DOCTYPE html>  
<html>  
<head>  
 <title>Запись данных в БД</title>  
</head>  
<body>  
  
 <h1>Добавить пользователя</h1>  
  
 <form id="add-user-form" method="POST">  
 <input type="text" name="name" placeholder="Имя" required>  
 <input type="email" name="email" placeholder="Email" required>  
 <input type="password" name="password" placeholder="Пароль" required>  
 <button type="submit">Добавить</button>  
 </form>  
  
 <script>  
 const addUserForm = document.getElementById('add-user-form');  
  
 addUserForm.addEventListener('submit', (event) => {  
 event.preventDefault();  
 const name = event.target.elements.name.value;  
 const email = event.target.elements.email.value;  
 const password = event.target.elements.password.value;  
  
 fetch('/users', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json'  
 },  
 body: JSON.stringify({ name, email, password })  
 })  
 .then(response => {  
 if (response.ok) {  
 alert('Пользователь добавлен');  
 addUserForm.reset(); *// Очищаем форму*  
 } else {  
 console.error('Ошибка при добавлении пользователя:', response.statusText);  
 }  
 })  
 .catch(error => console.error('Ошибка при добавлении пользователя:', error));  
 });  
 </script>  
  
</body>  
</html>

**Node.js (server.js) с Express и SQLite:**

*// ... (предыдущий код)*  
  
app.post('/users', (req, res) => {  
 const name = req.body.name;  
 const email = req.body.email;  
 const password = req.body.password;  
 const sql = 'INSERT INTO users (name, email, password) VALUES (?, ?, ?)';  
 db.run(sql, [name, email, password], (err) => {  
 if (err) {  
 console.error('Ошибка при добавлении пользователя:', err);  
 res.status(500).send('Ошибка при добавлении пользователя');  
 } else {  
 res.send('Пользователь добавлен');  
 }  
 });  
});  
  
*// ... (остальной код)*

**Node.js (server.js) с Express и MySQL:**

*// ... (предыдущий код)*  
  
app.post('/users', (req, res) => {  
 const name = req.body.name;  
 const email = req.body.email;  
 const password = req.body.password;  
 const sql = 'INSERT INTO users (name, email, password) VALUES (?, ?, ?)';  
 db.query(sql, [name, email, password], (err) => {  
 if (err) {  
 console.error('Ошибка при добавлении пользователя:', err);  
 res.status(500).send('Ошибка при добавлении пользователя');  
 } else {  
 res.send('Пользователь добавлен');  
 }  
 });  
});  
  
*// ... (остальной код)*

**Объяснение:**

* **HTML:** Форма для ввода имени, email и пароля.
* **JavaScript:**
  + При отправке формы делается запрос POST на маршрут /users.
  + Данные из формы отправляются в формате JSON.
  + В ответ отображается сообщение об успешном добавлении или об ошибке.
* **Node.js:**
  + Используется db.run() (SQLite) или db.query() (MySQL) для выполнения запроса INSERT.
  + Данные из запроса отправляются в базу данных.

**Шаг 4: Обновление и удаление данных**

**HTML (index.html):**

<!DOCTYPE html>  
<html>  
<head>  
 <title>Обновление и удаление данных</title>  
</head>  
<body>  
  
 <h1>Список пользователей</h1>  
  
 <table id="user-table">  
 <thead>  
 <tr>  
 <th>ID</th>  
 <th>Имя</th>  
 <th>Email</th>  
 <th>Действия</th>  
 </tr>  
 </thead>  
 <tbody>  
 </tbody>  
 </table>  
  
 <script>  
 *// ... (предыдущий код)*  
  
 function displayUsers(users) {  
 userTable.innerHTML = ''; *// Очищаем таблицу*  
 users.forEach(user => {  
 const row = userTable.insertRow();  
 row.insertCell().textContent = user.id;  
 row.insertCell().textContent = user.name;  
 row.insertCell().textContent = user.email;  
 const actionsCell = row.insertCell();  
  
 *// Кнопка обновления*  
 const updateButton = document.createElement('button');  
 updateButton.textContent = 'Обновить';  
 updateButton.addEventListener('click', () => {  
 *// Открыть модальное окно или форму для обновления данных*  
 *// ...*  
 });  
 actionsCell.appendChild(updateButton);  
  
 *// Кнопка удаления*  
 const deleteButton = document.createElement('button');  
 deleteButton.textContent = 'Удалить';  
 deleteButton.addEventListener('click', () => {  
 if (confirm(`Удалить пользователя ${user.name}?`)) {  
 fetch(`/users/${user.id}`, { method: 'DELETE' })  
 .then(response => {  
 if (response.ok) {  
 alert('Пользователь удален');  
 displayUsers(users.filter(u => u.id !== user.id)); *// Обновить таблицу*  
 } else {  
 console.error('Ошибка при удалении пользователя:', response.statusText);  
 }  
 })  
 .catch(error => console.error('Ошибка при удалении пользователя:', error));  
 }  
 });  
 actionsCell.appendChild(deleteButton);  
 });  
 }  
  
 *// ... (остальной код)*  
 </script>  
  
</body>  
</html>

**Node.js (server.js) с Express и SQLite:**

*// ... (предыдущий код)*  
  
app.put('/users/:id', (req, res) => {  
 const id = req.params.id;  
 const name = req.body.name;  
 const email = req.body.email;  
 const password = req.body.password;  
 const sql = 'UPDATE users SET name = ?, email = ?, password = ? WHERE id = ?';  
 db.run(sql, [name, email, password, id], (err) => {  
 if (err) {  
 console.error('Ошибка при обновлении пользователя:', err);  
 res.status(500).send('Ошибка при обновлении пользователя');  
 } else {  
 res.send('Пользователь обновлен');  
 }  
 });  
});  
  
app.delete('/users/:id', (req, res) => {

# 19 ПРАКТИЧЕСКАЯ.

**Шаг 1: Создание таблицы**

**Пример с SQLite:**

*-- Создание базы данных (если не существует)*  
CREATE DATABASE IF NOT EXISTS my\_database;  
  
*-- Переключение на созданную базу данных*  
USE my\_database;  
  
*-- Создание таблицы users*  
CREATE TABLE users (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 name VARCHAR(255) NOT NULL,  
 surname VARCHAR(255) NOT NULL,  
 email VARCHAR(255) UNIQUE NOT NULL,  
 birthdate DATE  
);

**Пример с MySQL:**

*-- Создание базы данных (если не существует)*  
CREATE DATABASE IF NOT EXISTS my\_database;  
  
*-- Переключение на созданную базу данных*  
USE my\_database;  
  
*-- Создание таблицы users*  
CREATE TABLE users (  
 id INT AUTO\_INCREMENT PRIMARY KEY,  
 name VARCHAR(255) NOT NULL,  
 surname VARCHAR(255) NOT NULL,  
 email VARCHAR(255) UNIQUE NOT NULL,  
 birthdate DATE  
);

**Объяснение:**

* **CREATE DATABASE IF NOT EXISTS my\_database:** Создает базу данных с именем my\_database, если она еще не существует.
* **USE my\_database:** Переключает текущую базу данных на my\_database.
* **CREATE TABLE users ( ... ):** Создает таблицу с именем users, описывая ее столбцы:
  + id - целое число, первичный ключ (AUTO\_INCREMENT в MySQL), автоматически увеличивается при добавлении новых записей.
  + name - строка (VARCHAR) длиной до 255 символов, обязательное поле.
  + surname - строка (VARCHAR) длиной до 255 символов, обязательное поле.
  + email - строка (VARCHAR) длиной до 255 символов, обязательное поле, уникальное (не допускает повторения).
  + birthdate - дата.

**Шаг 2: Вставка данных**

**Пример с SQLite:**

INSERT INTO users (name, surname, email, birthdate) VALUES   
 ('Иван', 'Петров', 'ivan.petrov@example.com', '2000-01-15'),  
 ('Мария', 'Иванова', 'maria.ivanova@example.com', '1998-05-22'),  
 ('Алексей', 'Сидоров', 'alexey.sidirov@example.com', '1995-10-08');

**Пример с MySQL:**

INSERT INTO users (name, surname, email, birthdate) VALUES   
 ('Иван', 'Петров', 'ivan.petrov@example.com', '2000-01-15'),  
 ('Мария', 'Иванова', 'maria.ivanova@example.com', '1998-05-22'),  
 ('Алексей', 'Сидоров', 'alexey.sidirov@example.com', '1995-10-08');

**Объяснение:**

* **INSERT INTO users ( ... ) VALUES ( ... ):** Вставляет данные в таблицу users.
* name, surname, email, birthdate - имена столбцов, в которые вставляются данные.
* Значения в скобках после VALUES должны соответствовать типам данных в соответствующих столбцах.

**Как выполнить:**

1. **Подключитесь** к базе данных. (Инструкции по подключению зависит от выбранной СУБД).
2. **Выполните** запросы CREATE TABLE и INSERT INTO в консоли СУБД или через клиент СУБД.

# 20 ПРАКТИЧЕСКАЯ.

**Шаг 1: Подготовка базы данных**

**Пример с SQLite:**

*-- Создание базы данных (если не существует)*  
CREATE DATABASE IF NOT EXISTS my\_database;  
  
*-- Переключение на созданную базу данных*  
USE my\_database;  
  
*-- Создание таблицы products*  
CREATE TABLE products (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 name VARCHAR(255) NOT NULL,  
 category VARCHAR(255) NOT NULL,  
 price DECIMAL(10,2) NOT NULL  
);  
  
*-- Вставка тестовых данных*  
INSERT INTO products (name, category, price) VALUES  
 ('Apple', 'Fruit', 1.50),  
 ('Banana', 'Fruit', 0.75),  
 ('Milk', 'Dairy', 2.50),  
 ('Bread', 'Bakery', 1.20),  
 ('Cheese', 'Dairy', 4.00),  
 ('Orange', 'Fruit', 0.90);

**Пример с MySQL:**

*-- Создание базы данных (если не существует)*  
CREATE DATABASE IF NOT EXISTS my\_database;  
  
*-- Переключение на созданную базу данных*  
USE my\_database;  
  
*-- Создание таблицы products*  
CREATE TABLE products (  
 id INT AUTO\_INCREMENT PRIMARY KEY,  
 name VARCHAR(255) NOT NULL,  
 category VARCHAR(255) NOT NULL,  
 price DECIMAL(10,2) NOT NULL  
);  
  
*-- Вставка тестовых данных*  
INSERT INTO products (name, category, price) VALUES  
 ('Apple', 'Fruit', 1.50),  
 ('Banana', 'Fruit', 0.75),  
 ('Milk', 'Dairy', 2.50),  
 ('Bread', 'Bakery', 1.20),  
 ('Cheese', 'Dairy', 4.00),  
 ('Orange', 'Fruit', 0.90);

**Шаг 2: Выполнение запросов на выборку данных**

**Пример запросов:**

*-- 1. Вывести все продукты*  
SELECT \* FROM products;  
  
*-- 2. Вывести продукты категории 'Fruit'*  
SELECT \* FROM products WHERE category = 'Fruit';  
  
*-- 3. Вывести продукты с ценой ниже 2.00*  
SELECT \* FROM products WHERE price < 2.00;

**Шаг 3: Выполнение запросов на вставку, обновление и удаление данных**

**Пример запросов:**

*-- 1. Вставить новый продукт*  
INSERT INTO products (name, category, price) VALUES ('Yogurt', 'Dairy', 3.00);  
  
*-- 2. Обновить цену продукта с id = 3*  
UPDATE products SET price = 2.75 WHERE id = 3;  
  
*-- 3. Удалить продукт с id = 1*  
DELETE FROM products WHERE id = 1;

**Как выполнить:**

1. **Подключитесь** к базе данных. (Инструкции по подключению зависят от выбранной СУБД).
2. **Выполните** SQL-запросы в консоли СУБД или через клиент СУБД.

# 21 ПРАКТИЧЕСКАЯ.

**Шаг 1: Создание сессии на сервере**

**Пример на PHP:**

<?php  
  
session\_start(); *// Инициализируем сессию*  
  
*// Проверка, залогинен ли пользователь*  
if (isset($\_SESSION['user\_id'])) {  
 *// Пользователь уже залогинен, делаем что-то...*  
} else {  
 *// Пользователь не залогинен, показываем форму входа*  
 echo '<form method="POST" action="login.php">';  
 echo '<input type="text" name="username" placeholder="Имя пользователя">';  
 echo '<button type="submit">Войти</button>';  
 echo '</form>';  
}  
  
*// Обработка входа*  
if ($\_SERVER['REQUEST\_METHOD'] === 'POST') {  
 $username = $\_POST['username'];  
  
 *// Проверка имени пользователя (в реальном проекте используйте базу данных)*  
 if ($username === 'admin') {  
 $\_SESSION['user\_id'] = 1; *// Сохраняем идентификатор пользователя в сессии*  
 $\_SESSION['username'] = $username; *// Сохраняем имя пользователя в сессии*  
 echo 'Успешный вход!';  
 } else {  
 echo 'Неверный логин.';  
 }  
}  
  
?>

**Шаг 2: Отправка и использование cookie**

**PHP:**

<?php  
  
*// ... (предыдущий код)*  
  
*// Отправка cookie с идентификатором сессии после успешного входа*  
if (isset($\_SESSION['user\_id'])) {  
 setcookie('session\_id', session\_id(), time() + 3600, '/'); *// Cookie с истечением через 1 час*  
}  
  
*// ... (остальной код)*  
  
?>

**JavaScript:**

*// ... (код для работы с cookie)*

**Шаг 3: Проверка сессии и cookie**

**PHP:**

<?php  
  
session\_start();  
  
*// Проверка наличия сессии и cookie*  
if (isset($\_SESSION['user\_id']) && isset($\_COOKIE['session\_id']) && $\_SESSION['user\_id'] == 1) {  
 *// Пользователь залогинен, разрешаем доступ к защищенным страницам*  
 echo 'Привет, ' . $\_SESSION['username'] . '! Вы залогинены.';  
} else {  
 *// Пользователь не залогинен, перенаправляем на страницу входа*  
 header('Location: login.php');  
 exit();  
}  
  
?>

**Объяснение:**

* **Шаг 1:**
  + session\_start() инициализирует сессию на сервере.
  + $\_SESSION - массив, в котором хранятся данные сессии.
  + При входе создается уникальный идентификатор сессии session\_id().
* **Шаг 2:**
  + setcookie() отправляет cookie с именем session\_id и значением session\_id().
* **Шаг 3:**
  + session\_start() снова инициализирует сессию.
  + Проверяется наличие $\_SESSION['user\_id'] (идентификатора пользователя) и cookie session\_id.
  + Если сессия действительна, то выводится информация о пользователе, иначе перенаправляется на страницу входа.