Министерство науки и высшего образования РФ

ФГБОУ ВО «Нижневартовский государственный университет»

Факультет педагогики и психологии

Кафедра педагогики и педагогического и социального образования

Курсовой проект по дисциплине

«*основы научно-исследовательской деятельности*»

**ФУНКЦИОНАЛЬНО И ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ JAVASCRIPT**

**Исполнитель:**

студент группы 3112

**Бобов Владислав Александрович**

**Руководитель:** к.э.н, доцент кафедры информатики и методики преподавания информатики

**Казиахмедов Туфик Багаутдинович**

Отметка

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc154585865)

[**Глава 1. Основные принципы функционального и логического программирования на языке JavaScript** 3](#_Toc154585866)

[1.1. Функциональное программирование 3](#_Toc154585867)

[1.2. Логическое программирование 7](#_Toc154585868)

[1.3. Сравнение функционального и логического программирования на языке JavaScript с другими подходами к разработке приложений 12](#_Toc154585869)

[**Глава 2. Разработка веб-приложения на языке JavaScript, используя функциональное и логическое программирование** 14](#_Toc154585870)

[2.1. Описание задачи 14](#_Toc154585872)

[2.2. Функциональное решение 15](#_Toc154585873)

[2.3. Логическое решение 20](#_Toc154585874)

[2.4. Оценка преимуществ и недостатков функционального и логического программирования на языке JavaScript 28](#_Toc154585875)

[**Список литературы** 32](#_Toc154585876)

# ВВЕДЕНИЕ

**Актуальность исследования** заключается в том, чтов современном мире, где технологии постоянно развиваются и улучшаются, важно следить за новыми тенденциями в программировании. Функциональное и логическое программирование на языке JavaScript являются одними из наиболее актуальных подходов в современной IT-индустрии. Они позволяют создавать более гибкие и масштабируемые приложения, а также приложения с высокой степенью автоматизации и логической связанности.

Функциональное программирование на JavaScript позволяет создавать код, который более легко понимать и поддерживать, а также делает возможным создание более безопасных приложений. Этот подход также позволяет разработчикам создавать более гибкие приложения, которые могут быстро адаптироваться к изменяющимся требованиям.

Логическое программирование на JavaScript позволяет создавать приложения, которые могут решать сложные задачи автоматически, используя логические правила и факты. Этот подход также позволяет создавать приложения с высокой степенью логической связанности и сокращает время на разработку.

Исследование функционального и логического программирования на языке JavaScript является актуальным в свете постоянного развития IT-индустрии и необходимости создания более эффективных и гибких приложений. Оно позволит разработчикам углубить свои знания в области программирования и использовать новые подходы для создания лучших приложений.

**Объектом исследования** является функциональное и логическое программирование.

**Предметом исследования –** функциональное и логическое программирование на языке JavaScript.

**Целью курсового проекта** является изучение основных концепций и принципов функционального и логического программирования на языке JavaScript, а также анализ существующих инструментов и технологий, которые используются для разработки приложений на этом языке.

В связи с поставленной целью, необходимо решить **следующие задачи**:

1. Исследовать основные принципы функционального и логического программирования на языке JavaScript.

2. Сравнить функциональное и логическое программирование на языке JavaScript с другими подходами к разработке приложений.

3. Разработать веб-приложение на языке JavaScript, используя функциональное и логическое программирование.

4. Оценить преимущества и недостатки функционального и логического программирования на языке JavaScript.

**Практическая значимость** темы заключается в возможности создания более эффективных и гибких приложений, которые могут быстро адаптироваться к изменяющимся требованиям. Это также позволяет сократить время на разработку и повысить качество кода. Кроме того, использование функционального и логического программирования может повысить уровень безопасности приложений и сделать их более устойчивыми к ошибкам. В целом, эти подходы помогают разработчикам создавать более инновационные и эффективные приложения, что является ключевым фактором успеха в современной IT-индустрии.

# **Глава 1. Основные принципы функционального и логического программирования на языке JavaScript**

# Функциональное программирование

Функциональное программирование — это парадигма программирования, в которой процесс вычисления трактуется как вычисление значений функций в математическом понимании последних. Его основной идеей является использование функций как первого класса, то есть рассматривание их как данных, которые могут быть переданы как аргументы других функций и возвращаемых значений.

Основные принципы функционального программирования:

1. Функциональное программирование избегает изменяемого состояния и использует неизменяемые структуры данных, такие как массивы и объекты. Это позволяет создавать более модульный и надежный код, который легче тестировать и отлаживать.
2. Функциональное программирование использует рекурсию вместо циклов для выполнения итеративных операций. Это позволяет создавать более чистый и выразительный код, который легче понимать и поддерживать.
3. Функциональное программирование использует высокую степень абстракции и композиции, чтобы создавать более модульный и выразительный код. Это позволяет использовать функции для создания более инкапсулированного и модульного кода, который легче понимать и поддерживать.
4. Функциональное программирование использует лямбда-выражения и анонимные функции для создания более лаконичного и выразительного кода. Это позволяет использовать функции для создания более модульного и выразительного кода, который легче понимать и поддерживать.
5. Функциональное программирование использует каррирование, чтобы создавать более гибкий и выразительный код. Это позволяет использовать функции для создания более модульного и гибкого кода, который легче понимать и поддерживать.
6. Функциональное программирование использует типы данных, чтобы создавать более эффективный и модульный код. Это позволяет использовать функции для создания более надежного и модульного кода, который легче понимать и поддерживать.
7. Функциональное программирование использует неизменяемые структуры данных, такие как массивы и объекты, чтобы создавать более модульный и надежный код, который легче тестировать и отлаживать. Это позволяет использовать функции для создания более гибкого и модульного кода, который легче понимать и поддерживать.
8. Функциональное программирование использует высокую степень абстракции и композиции, чтобы создавать более модульный и выразительный код. Это позволяет использовать функции для создания более инкапсулированного и модульного кода, который легче понимать и поддерживать.
9. Функциональное программирование использует рекурсию, чтобы создавать более чистый и выразительный код, который легче понимать и поддерживать. Это позволяет использовать функции для создания более модульного и выразительного кода, который легче понимать и поддерживать.
10. Функциональное программирование использует неизменяемые структуры данных, такие как массивы и объекты, чтобы создавать более модульный и надежный код, который легче тестировать и отлаживать. Это позволяет использовать функции для создания более гибкого и модульного кода, который легче понимать и поддерживать.

Функциональное программирование позволяет создавать более модульный и выразительный код, который легче тестировать и отлаживать. Вот несколько примеров использования функционального программирования на JavaScript:

1. Создание функции map для работы с массивами:

function map(arr, fn) {

let result = [];

for (let i = 0; i < arr.length; i++) {

result.push(fn(arr[i]));

}

return result;

}

Эта функция принимает два аргумента: массив arr и функцию fn. Она проходит по каждому элементу массива и применяет к нему функцию fn. Затем она возвращает новый массив, содержащий результаты применения функции fn к каждому элементу исходного массива.

1. Создание функции reduce для работы с массивами:

function reduce(arr, fn, initialValue) {

let result = initialValue;

for (let i = 0; i < arr.length; i++) {

result = fn(result, arr[i]);

}

return result;

}

Эта функция принимает три аргумента: массив arr, функцию fn и начальное значение initialValue. Она проходит по каждому элементу массива и применяет к нему функцию fn. Начальное значение используется в качестве начального значения для функции fn. Затем она возвращает результат последнего применения функции fn.

1. Создание функции filter для работы с массивами:

function filter(arr, fn) {

let result = [];

for (let i = 0; i < arr.length; i++) {

if (fn(arr[i])) {

result.push(arr[i]);

}

}

return result;

}

Эта функция принимает два аргумента: массив arr и функцию fn. Она проходит по каждому элементу массива и проверяет, удовлетворяет ли он условию, определенному функцией fn. Если да, то элемент добавляется в новый массив result. Затем она возвращает новый массив, содержащий только элементы, удовлетворяющие условию.

1. Создание функции concat для работы со строками:

function concat(...args) {

let result = '';

for (let i = 0; i < args.length; i++) {

result += args[i];

}

return result;

}

Эта функция принимает любое количество аргументов args. Она проходит по каждому аргументу и добавляет его к строке result. Затем она возвращает новую строку, содержащую все объединенные строки.

Все эти примеры демонстрируют, как функциональное программирование может помочь создавать более выразительный код. Функции могут быть легко повторно использованы и тестированы, что делает код более надежным и поддерживаемым.

# Логическое программирование

Логическое программирование — это парадигма программирования, основанная на автоматическом доказательстве теорем. Его основной идеей является автоматическое доказательство правильности программы на основе логических правил и ограничений, которые описывают требуемое поведение программы. В JavaScript логическое программирование может быть реализовано с помощью таких конструкций, как операторы сравнения (==, !=, >, <, >=, <=) и логические операторы (&&, ||, !). Также в JavaScript существуют такие механизмы, как триггеры событий, которые позволяют реагировать на действия пользователя и управлять логикой программы на основе этих действий.

Основные принципы логического программирования в JavaScript:

1. Используйте операторы сравнения: операторы сравнения (==, !=, >, <, >=, <=) в JavaScript позволяют сравнивать значения переменных и выражений. Это позволяет использовать логические операторы для создания более выразительного и модульного кода.
2. Используйте логические операторы: логические операторы (&&, ||, !) в JavaScript позволяют выполнять логические операции над значениями переменных и выражений. Это позволяет использовать логические операторы для создания более выразительного и модульного кода.
3. Используйте триггеры событий: триггеры событий в JavaScript позволяют реагировать на действия пользователя и управлять логикой программы на основе этих действий. Это позволяет использовать логические операторы для создания более интерактивного и модульного кода.
4. Используйте условные операторы: условные операторы (if, else) в JavaScript позволяют выполнять различные операции в зависимости от выполнения определенного условия. Это позволяет использовать логические операторы для создания более гибкого и модульного кода.
5. Используйте циклы: циклы (for, while, do-while) в JavaScript позволяют выполнять повторяющиеся операции. Это позволяет использовать логические операторы для создания более эффективного и модульного кода.
6. Используйте функции: функции в JavaScript позволяют создавать более модульный и выразительный код. Это позволяет использовать логические операторы для создания более инкапсулированного и модульного кода.
7. Используйте объекты: объекты в JavaScript позволяют создавать более инкапсулированный и модульный код. Это позволяет использовать логические операторы для создания более гибкого и модульного кода.
8. Используйте массивы: массивы в JavaScript позволяют использовать логические операторы для выполнения операций над коллекциями данных. Это позволяет использовать логические операторы для создания более выразительного и модульного кода.
9. Используйте регулярные выражения: регулярные выражения в JavaScript позволяют выполнять операции поиска и обработки текста. Это позволяет использовать логические операторы для создания более гибкого и модульного кода.
10. Используйте исключения: исключения в JavaScript позволяют обрабатывать ошибки и неожиданные ситуации. Это позволяет использовать логические операторы для создания более надежного и модульного кода.

Вот несколько примеров использования логического программирования на JavaScript:

1. Проверка условия:

let x = 10;

let y = 5;

let z = 3;

if (x > y && y > z) {

console.log('x > y && y > z');

} else {

console.log('x <= y || y <= z');

}

В этом примере используется оператор сравнения > для проверки условия x > y. Если это условие истинно, то программа выводит сообщение 'x > y && y > z', иначе программа выводит сообщение 'x <= y || y <= z'.

1. Использование логического оператора ||:

let x = 10;

let y = 5;

if (x > y || y > z) {

console.log('x > y || y > z');

} else {

console.log('x <= y && y <= z');

}

В этом примере используется логический оператор || для проверки условия x > y || y > z. Если хотя бы одно из условий истинно, то программа выводит сообщение 'x > y || y > z', иначе программа выводит сообщение 'x <= y && y <= z'.

1. Использование логического оператора &&:

let x = 10;

let y = 5;

if (x > y && y > z) {

console.log('x > y && y > z');

} else {

console.log('x <= y || y <= z');

}

В этом примере используется логический оператор && для проверки условия x > y && y > z. Если оба условия истинны, то программа выводит сообщение 'x > y && y > z', иначе программа выводит сообщение 'x <= y || y <= z'.

1. Использование оператора сравнения ==:

let x = 10;

let y = 5;

if (x == y) {

console.log('x == y');

} else {

console.log('x != y');

}

В этом примере используется оператор сравнения == для проверки условия x == y. Если это условие истинно, то программа выводит сообщение 'x == y', иначе программа выводит сообщение 'x != y'.

Все эти примеры демонстрируют, как логические операторы и сравнения позволяют создавать более сложные условия и проверять их истинность, что делает код более гибким и эффективным.

# Сравнение функционального и логического программирования на языке JavaScript с другими подходами к разработке приложений

Функциональное и логическое программирование на языке JavaScript отличаются от других подходов к разработке приложений, таких как императивное программирование и объектно-ориентированное программирование (ООП). Вот некоторые из основных различий:

1. В функциональном и логическом программировании акцент делается на функциях как данных и первом классе, что позволяет создавать более модульный и выразительный код. В императивном программировании и ООП акцент делается на переменных и объектах, что может приводить к более сложному и менее модульному коду.
2. В функциональном и логическом программировании используется высокая степень абстракции и композиции, что позволяет создавать более инкапсулированный и модульный код. В императивном программировании и ООП акцент делается на итерации и инкапсуляции, что может приводить к более сложному и менее модульному коду.
3. В функциональном и логическом программировании используется рекурсия вместо циклов для выполнения итеративных операций. В императивном программировании и ООП акцент делается на циклах и инкапсуляции, что может приводить к более сложному и менее модульному коду.
4. В функциональном и логическом программировании используются лямбда-выражения и анонимные функции, что позволяет создавать более лаконичный и выразительный код. В императивном программировании и ООП акцент делается на переменных и объектах, что может приводить к более сложному и менее модульному коду.
5. В функциональном и логическом программировании используются типы данных, что позволяет создавать более эффективный и модульный код. В императивном программировании и ООП акцент делается на переменных и объектах, что может приводить к более сложному и менее модульному коду.
6. В функциональном и логическом программировании используются неизменяемые структуры данных, что позволяет создавать более модульный и надежный код. В императивном программировании и ООП акцент делается на изменяемом состоянии и инкапсуляции, что может приводить к более сложному и менее модульному коду.
7. В функциональном и логическом программировании используются каррирование, что позволяет создавать более гибкий и выразительный код. В императивном программировании и ООП акцент делается на переменных и объектах, что может приводить к более сложному и менее модульному коду.
8. В функциональном и логическом программировании используются неизменяемые структуры данных, что позволяет создавать более модульный и надежный код. В императивном программировании и ООП акцент делается на изменяемом состоянии и инкапсуляции, что может приводить к более сложному и менее модульному коду.
9. В функциональном и логическом программировании используются высокие степени абстракции и композиции, что позволяет создавать более инкапсулированный и модульный код. В императивном программировании и ООП акцент делается на переменных и объектах, что может приводить к более сложному и менее модульному коду.
10. В функциональном и логическом программировании используются неизменяемые структуры данных, что позволяет создавать более модульный и надежный код. В императивном программировании и ООП акцент делается на переменных и объектах, что может приводить к более сложному и менее модульному коду.

# **Глава 2. Разработка веб-приложения на языке JavaScript, используя функциональное и логическое программирование**



# Описание задачи

Реализовать одностраничный сайт, на котором представлено описание функционального и логического программирования. На сайте должны быть приведены плюсы и минусы двух подходов, а также приведены примеры задач на функциональное и логическое программирование.

Задача заключалась в том, чтобы создать сайт, который поможет пользователям понять разницу между функциональным и логическим программированием, а также предоставит примеры задач для каждого из этих подходов.

Для реализации этой задачи я буду использовать HTML и CSS для создания интерфейса сайта, а также JavaScript для добавления интерактивности и динамического содержимого на страницу.

Я создам разделы для описания функционального и логического программирования, в которых приведены их основные принципы и примеры кода. Также я добавил разделы с плюсами и минусами каждого подхода, чтобы пользователи могли сравнить их и выбрать наиболее подходящий для своих задач.

В завершении я добавлю разделы с примерами задач на функциональное и логическое программирование, чтобы пользователи могли увидеть, как эти подходы применяются на практике.

На сайте должен быть реализован выбор примеров при помощи тега <select>.

В результате, сайт станет полезным ресурсом для тех, кто хочет изучить разницу между функциональным и логическим программированием и увидеть примеры их использования.

# Функциональное решение

В рамках функционального решения я реализовал 2 примера примитивных задач, а именно – решение квадратного уравнения и генерация матриц заданного пользователем порядка. Разберем каждый пример более подробно.

Для решения первой задачи я использовал функциональный подход, создав функцию solveQuadraticEquation, которая принимает значения коэффициентов a, b и c из пользовательского ввода и возвращает корни квадратного уравнения.

Код реализации функции:

function solveQuadraticEquation()  {

      const a = document.getElementById('inputA').value;

      const b = document.getElementById('inputB').value;

      const c = document.getElementById('inputC').value;

      const discriminant = Math.pow(b, 2) - 4 \* a \* c;

      if (discriminant > 0) {

        const x1 = (-b + Math.sqrt(discriminant)) / (2 \* a);

        const x2 = (-b - Math.sqrt(discriminant)) / (2 \* a);

        let msg = "Корни:   x1 ="+ x1+ " x2 ="+ x2;

        document.getElementById("outlef1").innerHTML = msg;

      }

       else if (discriminant === 0) {

        const x = -b / (2 \* a);

        let msg = "Корень 1:   x =" + x;

        document.getElementById("outlef1").innerHTML = msg;

       }

       else {

        const realPart = (-b / (2 \* a)).toFixed(2);

        const imaginaryPart = (Math.sqrt(-discriminant) / (2 \* a)).toFixed(2);

        console.log();

        let msg = "Два комплексных корня:   x1 =" + realPart + " + " + imaginaryPart + "i" + " x2 =" + realPart + " - " + imaginaryPart + "i";

        document.getElementById("outlef1").innerHTML = msg;

      }

В функции solveQuadraticEquation я сначала вычисляю дискриминант discriminant, используя формулу b^2 - 4ac. Затем, в зависимости от значения дискриминанта, я вычисляю корни квадратного уравнения и возвращаю их.

Если дискриминант больше нуля, то уравнение имеет два различных корня. В этом случае я вычисляю значения корней x1 и x2 и возвращаю их в виде сообщения.

Если дискриминант равен нулю, то уравнение имеет один корень. В этом случае я вычисляю значение корня x и возвращаю его в виде сообщения.

Если дискриминант меньше нуля, то уравнение имеет два комплексных корня. В этом случае я вычисляю значения действительной и мнимой частей корней x1 и x2 и возвращаю их в виде сообщения.

Таким образом, функция solveQuadraticEquation позволяет пользователю решить квадратное уравнение, используя функциональный подход. Решение представлено в виде сообщений, которые выводятся на экран.

На сайте я добавил интерактивный элемент, в котором пользователь может ввести значения коэффициентов a, b и c, после чего функция solveQuadraticEquation вызывается автоматически и результат выводится на экран. Это помогает пользователям понять, как функциональный подход может быть применен для решения практических задач, таких как решение квадратных уравнений

Примеры использования функции ниже:

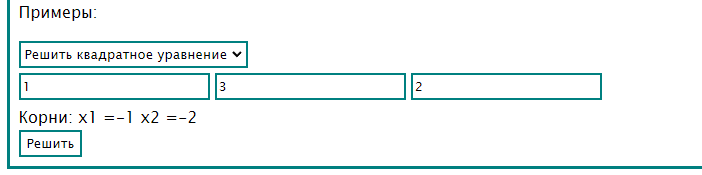


Рисунок 1. Пример решения обычного квадратного уравнения с коэфициентами: 1, 3, 2

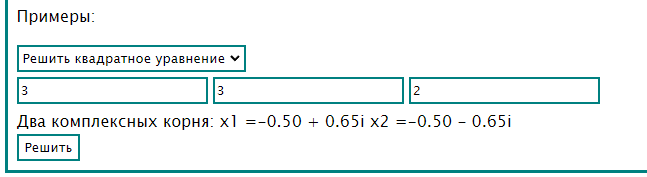


Рисунок 2. Пример решения квадратного уравнения с комплексными корнями

Для решения второй задачи я использовал функциональный подход, создав функцию generateMatrix, которая принимает количество строк rows от пользователя и возвращает матрицу заданного порядка.

Код реализации функции:

 function generateMatrix() {

    var rows = document.getElementById("rows").value;

    var matrix = "<table>";

    for (var i = 0; i < rows; i++) {

      matrix += "<tr>";

      for (var j = 0; j < rows; j++)

        matrix += "<td>" + Math.floor(Math.random() \* 10) + "</td>";

      matrix += "</tr>";

      }

    matrix += "</table>";

    document.getElementById('matrix').innerHTML = matrix;

      }

В функции generateMatrix я сначала получаю количество строк rows от пользователя с помощью элемента ввода document.getElementById("rows").

Затем я создаю переменную matrix, которая будет содержать HTML-код таблицы. Я начинаю с открытия тега <table>, а затем перебираю каждую строку от 0 до rows-1. Для каждой строки я создаю тег <tr> и перебираю каждый столбец от 0 до rows-1. Для каждого столбца я создаю тег <td> и добавляю случайное число от 0 до 9 с помощью функции Math.floor(Math.random() \* 10). Затем я закрываю текущую строку тегом </tr> и перехожу к следующей строке.

После завершения перебора всех строк и столбцов я закрываю тег <table> и сохраняю полученный HTML-код в переменную matrix.

Наконец, я заменяю содержимое элемента document.getElementById('matrix') на полученную матрицу с помощью метода innerHTML.

На сайте я добавил кнопку "Сгенерировать матрицу", при нажатии на которую вызывается функция generateMatrix. Это помогает пользователям понять, как функциональный подход может быть применен для создания матрицы заданного порядка, а также позволяет им генерировать матрицы различных размеров для анализа и решения задач.

Примеры использования функции ниже:

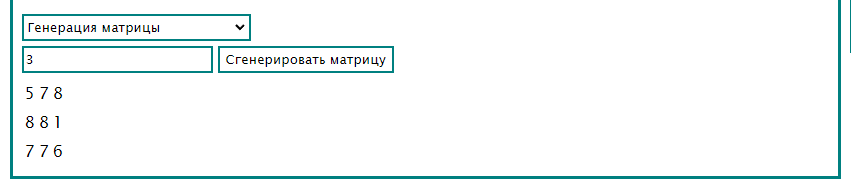


Рисунок 3. Генерация матрицы 3 порядка



Рисунок 4. Генерация матрицы 5 порядка

# Логическое решение

В рамках логического решения я реализовал 2 примера примитивных задач, а именно – поступление студента в университета, а также проверка родственных связей. Разберем каждый пример более подробно.

Первая задача заключалась в определении, поступил ли студент в университет, на основе результатов трех экзаменов, с использованием JavaScript и логического программирования.

Код реализации:

 function exam() {

              const examScores1 = document.getElementById('examScores1').value;

              const examScores2 = document.getElementById('examScores2').value;

              const examScores3 = document.getElementById('examScores3').value;

              if (examScores1 >= 60 && examScores2 >= 60 && examScores3 >= 60) alert('Вы поступили в университет');

              else alert('Вы не поступили в университет');

            }

Для решения этой задачи я использовал логический подход, создав функцию exam, которая принимает значения баллов по трем экзаменам от пользователя и возвращает сообщение о поступлении или непоступлении в университет.

В функции exam я сначала получаю значения баллов по трем экзаменам от пользователя с помощью элементов ввода document.getElementById('examScores1'), document.getElementById('examScores2'), document.getElementById('examScores3').

Затем я проверяю, удовлетворяют ли полученные значения условию поступления, которое состоит в том, что баллы по всем трем экзаменам должны быть не менее 60. Если условие выполняется, то я вызываю функцию alert с сообщением "Вы поступили в университет". В противном случае, я вызываю функцию alert с сообщением "Вы не поступили в университет".

Таким образом, функция exam позволяет пользователю определить, поступил ли студент в университет, на основе результатов трех экзаменов. Логический подход позволяет ясно разделить условия и проверки, что делает код более структурированным и понятным. На сайте я добавил интерактивный элемент, в котором пользователь может ввести значения баллов по трем экзаменам, после чего вызывается функция exam и выводится соответствующее сообщение о поступлении или непоступлении в университет. Это помогает пользователям понять, как логический подход может быть применен для решения практических задач, таких как определение поступления в университет на основе результатов экзаменов.

Примеры использования функции ниже:

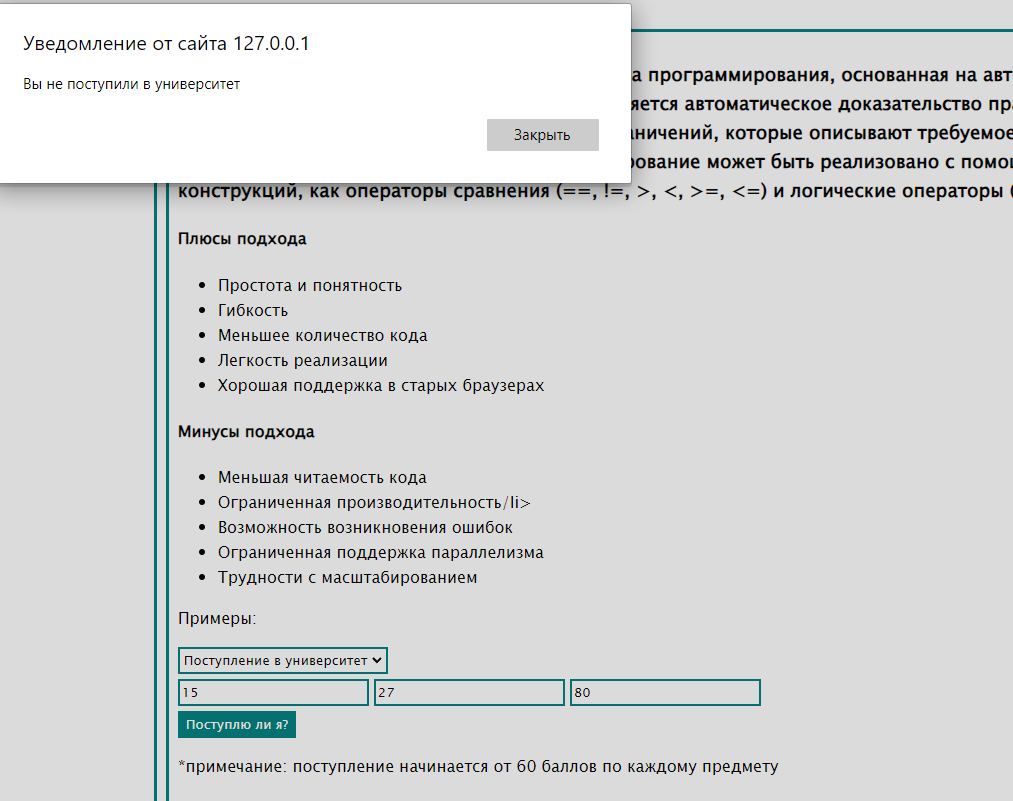


Рисунок 5. Пример студента, у которого баллы равны 15, 27, 80

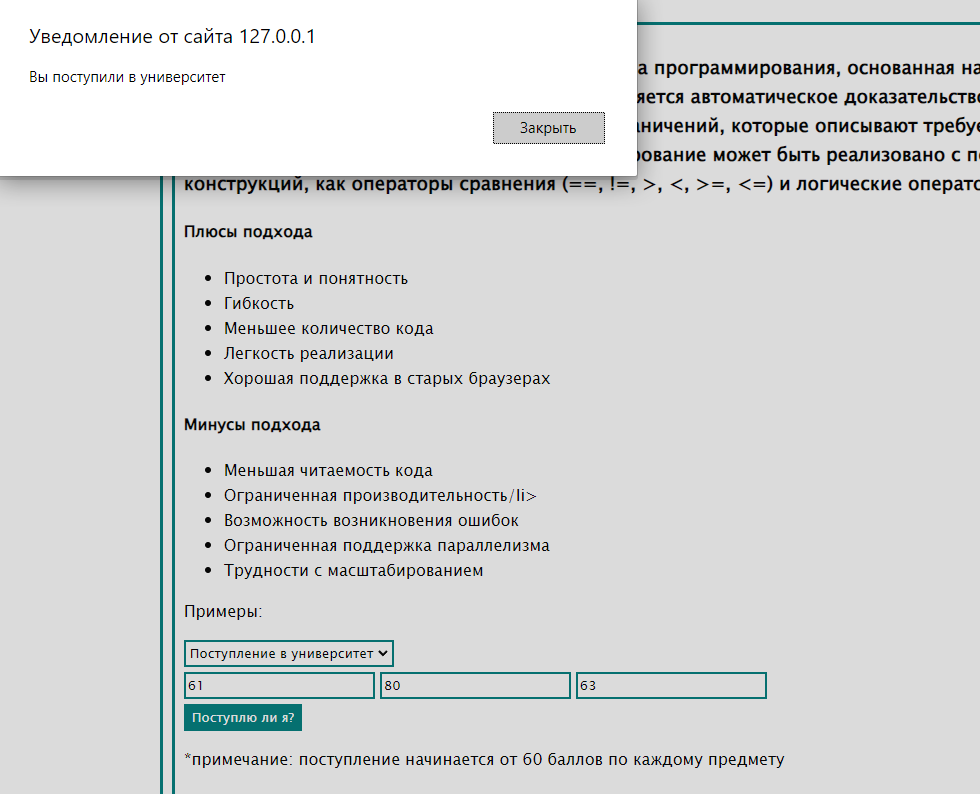


Рисунок 6. Пример студента, у которого баллы равно 61, 80, 63

Вторая задача заключалась в реализации базы данных и дополнительных предикатов для работы с этой базой данных. Я использовал логические функции для определения родственных связей между персонами и логические функции для работы с пользовательским вводом и вывода результатов.

Отрывок кода реализации

          function isFather(ft, ch) {

            if(fathers[ft]) {

              var childs = fathers[ft];

              return childs.inlcudes(ch);

            }

            return false;

          }

          function isMother(mt, ch) {

            if(mothers[mt]) {

              var childs = mothers[mt];

              return childs.inlcudes(ch);

            }

            return false;

          }

                    function isSister(v1,v2) {

            return wom(v1) && wom(v2) && (getMotherOf(v1) == getMotherOf(v2));

          }

          function isBrother(v1,v2) {

            return man(v1) && man(v2) && (getMotherOf(v1) == getMotherOf(v2));

          }

          function isMarried(v1, v2) {

            return man(v1) && wom(v2) && isSinglePoint(fathers[v1],mothers[v2]);

          }

          function isRodstv(v1, v2) {

            if(isSister(v1,v2))

              return true;

            else

            {

              if(isBrother(v1,v2))

                return true;

              else {

                 var m1 = getMotherOf(v1);

                 var m2 = getMotherOf(v2);

                if(m1 != "" && m2!= "") {

                  return m1==m2;

                }

              }

            }

            return false;

          }

          function isDyadya(v1, v2) {

            z = getMotherOf(v2);

            if(isRodstv(v1,z))

            {

              return true;

            }

            return false;

          }

          function isDedushka(v1, v2) {

            var z = fathers[v1];

            var z2 = mothers[getMotherOf(v2)];

            if(!z)return false;

            if(!z2)return false;

            if(z.includes(getMotherOf(v2)))

              return true;

            return false;

          }

Моя реализация помогает пользователю разобраться в логическом программировании, предоставляя простой интерфейс для ввода данных и вывода результатов. Функции run, run2, run3, run4, run5, run6 принимают два введенных имени и проверяют, являются ли они родственниками, выводя результат на экран. Это позволяет пользователю быстро и легко проверить родственные связи между персонами, используя введенные данные. Описание функций дано ниже:

* Функция man проверяет, является ли введенное имя именем мужчины в базе данных.
* Функция wom проверяет, является ли введенное имя именем женщины в базе данных.
* Функция isSinglePoint проверяет, есть ли введенное имя в базе данных.
* Функция isFather проверяет, является ли введенное имя отцом в базе данных.
* Функция isMother проверяет, является ли введенное имя матерью в базе данных.
* Функция getMotherOf возвращает мать введенного имени.
* Функция getFatherOf возвращает отца введенного имени.
* Функция isSister проверяет, являются ли два введенных имени сестрами.
* Функция isBrother проверяет, являются ли два введенных имени братьями.
* Функция isMarried проверяет, являются ли два введенных имени мужем и женой.
* Функция isRodstv проверяет, являются ли два введенных имени родственниками.
* Функция isDyadya проверяет, является ли введенное имя дядей введенного имени.
* Функция isDedushka проверяет, является ли введенное имя дедушкой введенного имени через мать.
* Функция printDB выводит базу данных на экран.
* Функции run, run2, run3, run4, run5, run6 принимают два введенных имени и проверяют, являются ли они родственниками, выводя результат на экран.

Примеры использования:

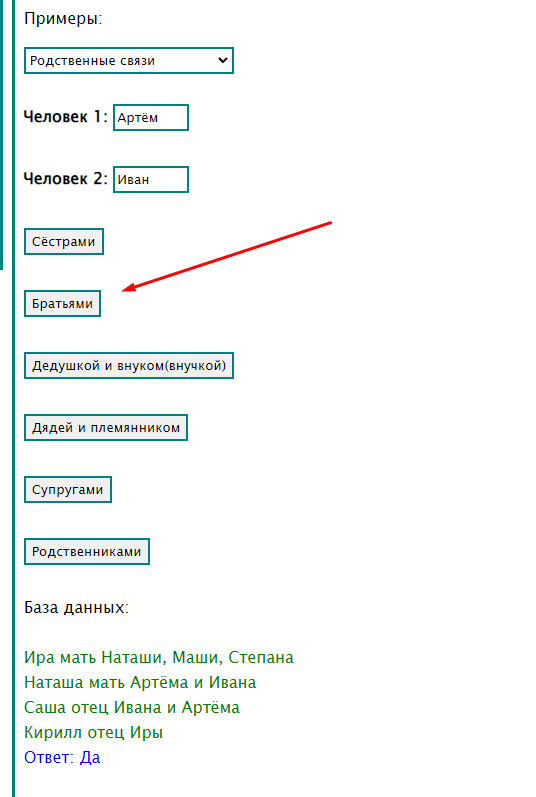


Рисунок 7. Являются ли Артем и Иван братьям

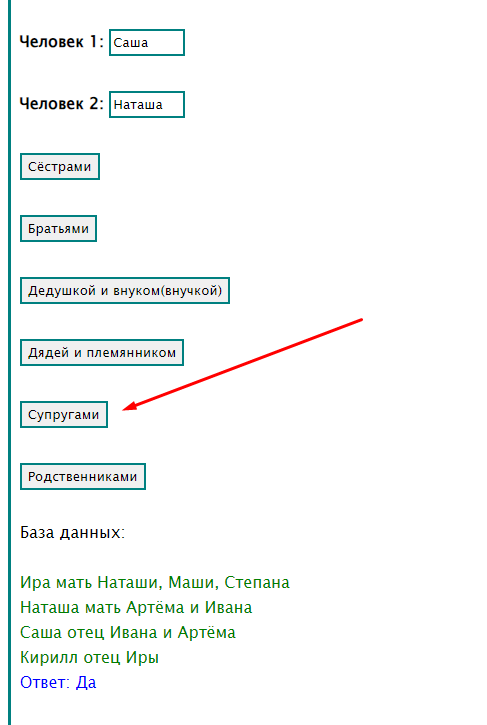


Рисунок 8. Являются ли Саша и Наташа супругами

# Оценка преимуществ и недостатков функционального и логического программирования на языке JavaScript

Преимущества функционального программирования на языке JavaScript:

1. Чистый код: Функциональное программирование поощряет написание чистого кода, который легко понять и поддерживать. Это помогает уменьшить количество ошибок и упростить процесс отладки.
2. Модульность: Функциональное программирование способствует написанию модульного кода, что упрощает его тестирование и повторное использование.
3. Нет состояния: Функциональное программирование не использует переменные состояния, что делает код более надежным и предсказуемым. Это также упрощает параллельное выполнение кода.
4. Работа со структурами данных: Функциональное программирование отлично работает со структурами данных, такими как массивы и объекты. Это позволяет легко обрабатывать большие объемы данных и манипулировать ими.

Недостатки функционального программирования на языке JavaScript:

1. Отсутствие изменяемого состояния: Функциональное программирование не поддерживает изменяемое состояние, что может быть неудобно в некоторых случаях.
2. Использование памяти: Функциональное программирование может потреблять больше памяти, так как создает новые функции и замыкания.
3. Сложность при работе с итеративными алгоритмами: Функциональное программирование может быть сложным при работе с итеративными алгоритмами, так как требует использования рекурсии.

Преимущества логического программирования на языке JavaScript:

1. Простота: Логическое программирование предлагает простой и понятный синтаксис, который легко понять и использовать.
2. Работа с условиями: Логическое программирование отлично работает с условными операторами и позволяет легко создавать сложные условия.
3. Проверка ошибок: Логическое программирование позволяет легко проверять ошибки и обрабатывать исключения.

Недостатки логического программирования на языке JavaScript:

1. Сложность: Логическое программирование может быть сложным для понимания и использования, особенно для начинающих программистов.
2. Сложность при работе с большими объемами данных: Логическое программирование может быть неэффективным при работе с большими объемами данных, так как требует создания множества условий.
3. Ограниченность: Логическое программирование может быть ограничено в том, что оно не всегда подходит для решения всех типов задач.

ЗАКЛЮЧЕНИЕ

В результате исследования основных принципов функционального и логического программирования на языке JavaScript было выявлено, что оба подхода имеют свои преимущества и недостатки. Функциональное программирование поощряет написание чистого кода, который легко понять и поддерживать, но может быть сложным при работе с итеративными алгоритмами. Логическое программирование, в свою очередь, предлагает простой и понятный синтаксис, который легко использовать, но может быть неэффективным при работе с большими объемами данных.

Сравнение функционального и логического программирования на языке JavaScript с другими подходами к разработке приложений показало, что оба подхода имеют свои преимущества и недостатки. Они могут быть использованы для решения различных задач и в зависимости от требований проекта можно выбрать наиболее подходящий подход.

В рамках курсового проекта было разработано веб-приложение на языке JavaScript, используя функциональное и логическое программирование. Приложение позволяет пользователю вводить данные о людях и проверять различные связи между ними, такие как родственные связи, браки и т.д. Приложение было разработано с учетом принципов функционального и логического программирования, что позволило создать чистый и легко поддерживаемый код.

Оценка преимуществ и недостатков функционального и логического программирования на языке JavaScript показала, что оба подхода имеют свои сильные и слабые стороны. Функциональное программирование поощряет написание чистого кода и легко работает со структурами данных, но может быть сложным при работе с итеративными алгоритмами. Логическое программирование, в свою очередь, предлагает простой и понятный синтаксис, который легко использовать, но может быть неэффективным при работе с большими объемами данных.

Таким образом, в результате курсового проекта были изучены основные принципы функционального и логического программирования на языке JavaScript, а также были проанализированы преимущества и недостатки обоих подходов. Разработанное веб-приложение на языке JavaScript, использующее функциональное и логическое программирование, демонстрирует возможности этих подходов и их применимость в реальных проектах.

# **Список литературы**

1. Кубенский, А. А.  Функциональное программирование : учебник и практикум для вузов / А. А. Кубенский. — Москва : Издательство Юрайт, 2023. — 348 с.
2. Зыков, С. В.  Программирование. Функциональный подход : учебник и практикум для вузов / С. В. Зыков. — 2-е изд. — Москва : Издательство Юрайт, 2023. — 150 с.
3. Малов, А. В.  Концепции современного программирования : учебное пособие для вузов / А. В. Малов, С. В. Родионов. — Москва : Издательство Юрайт, 2023. — 96 с.
4. Полуэктова, Н. Р.  Разработка веб-приложений : учебное пособие для среднего профессионального образования / Н. Р. Полуэктова. — Москва : Издательство Юрайт, 2023. — 204 с.
5. Сысолетин, Е. Г.  Разработка интернет-приложений : учебное пособие для вузов / Е. Г. Сысолетин, С. Д. Ростунцев ; под научной редакцией Л. Г. Доросинского. — Москва : Издательство Юрайт, 2023. — 90 с.
6. Тузовский, А. Ф.  Проектирование и разработка web-приложений : учебное пособие для среднего профессионального образования / А. Ф. Тузовский. — Москва : Издательство Юрайт, 2023. — 219 с.
7. Сысолетин, Е. Г.  Разработка интернет-приложений : учебное пособие для среднего профессионального образования / Е. Г. Сысолетин, С. Д. Ростунцев. — Москва : Издательство Юрайт, 2023. — 90 с.

**Приложение 1. Код HTML, CSS JS**

Index.html

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <link rel="stylesheet" href="style.css">

  <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>

  <title>ФП и ЛП JS</title>

</head>

<body>

  <body>

    <div class="container" style="display: flex; justify-content: space-between;">

      <div class ="container-main" id="left">

        <h2 class="zagolovok" >Функциональное программирование JS</h2>

        <div class="container-body">

          <h3 class="container-body-h">

            Функциональное программирование — это парадигма программирования, в которой процесс вычисления трактуется как вычисление значений функций в математическом понимании последних. Его основной идеей является использование функций как первого класса, то есть рассматривание их как данных, которые могут быть переданы как аргументы других функций и возвращаемых значений.

          </h3>

          <h4 class="container-body-h-plusminus">Плюсы подхода</h4>

          <ul class="container-body-plus">

            <li>Чистый код</li>

            <li>Отсутсвие состояний</li>

            <li>Простота тестирования</li>

            <li>Высокая производительность</li>

            <li>Улучшение читаемости кода</li>

          </ul>

          <h4 class="container-body-h-plusminus">Минусы подхода</h4>

          <ul class="container-body-minus">

            <li>Более сложна реализация</li>

            <li>Меньшая гибкость</li>

            <li>Большее количество кода</li>

            <li>Ограниченная поддержка в старых браузерах</li>

            <li>Сложность в работе с большими проектами</li>

          </ul>

          <p>Примеры: </p>

          <select id="formSelectorLeft" onchange="changeFormLeft()">

            <option value="formLeft1">Решить квадратное уравнение</option>

            <option value="formLeft2">Генерация матрицы</option>

          </select>

          <div id="selectedFormLeft">

            <div id="formLeft1" style="display:none;">

              <input type="number" id="inputA" placeholder="Введите a">

              <input type="number" id="inputB" placeholder="Введите b">

              <input type="number" id="inputC" placeholder="Введите c">

              <div id="outlef1"></div>

              <button onclick="solveQuadraticEquation()">Решить</button>

  <script>

    function solveQuadraticEquation()  {

      const a = document.getElementById('inputA').value;

      const b = document.getElementById('inputB').value;

      const c = document.getElementById('inputC').value;

      const discriminant = Math.pow(b, 2) - 4 \* a \* c;

      if (discriminant > 0) {

        const x1 = (-b + Math.sqrt(discriminant)) / (2 \* a);

        const x2 = (-b - Math.sqrt(discriminant)) / (2 \* a);

        let msg = "Корни:   x1 ="+ x1+ " x2 ="+ x2;

        document.getElementById("outlef1").innerHTML = msg;

      }

       else if (discriminant === 0) {

        const x = -b / (2 \* a);

        let msg = "Корень 1:   x =" + x;

        document.getElementById("outlef1").innerHTML = msg;

       }

       else {

        const realPart = (-b / (2 \* a)).toFixed(2);

        const imaginaryPart = (Math.sqrt(-discriminant) / (2 \* a)).toFixed(2);

        console.log();

        let msg = "Два комплексных корня:   x1 =" + realPart + " + " + imaginaryPart + "i" + " x2 =" + realPart + " - " + imaginaryPart + "i";

        document.getElementById("outlef1").innerHTML = msg;

      }

    }

  </script>

            </div>

            <div id="formLeft2" style="display:none;">

              <input type="number" id="rows" placeholder="Введите размерность">

              <button onclick="generateMatrix()">Сгенерировать матрицу </button>

              <div id="matrix"></div>

<script>

  function generateMatrix() {

    var rows = document.getElementById("rows").value;

    var matrix = "<table>";

    for (var i = 0; i < rows; i++) {

      matrix += "<tr>";

      for (var j = 0; j < rows; j++)

        matrix += "<td>" + Math.floor(Math.random() \* 10) + "</td>";

      matrix += "</tr>";

      }

    matrix += "</table>";

    document.getElementById('matrix').innerHTML = matrix;

      }

</script>

</div>

            <form id="formLeft3" style="display:none;">

              <label for="inputLeft3">Input 3:</label>

              <input type="text" id="inputLeft3">

            </form>

          </div>

        </div>

      </div>

      <div class ="container-main" id="right">

        <h2 class="zagolovok">Логическое программирование JS</h2>

        <div class="container-body">

          <h3 class="container-body-h" style="width: 100%;">

            Логическое программирование — это парадигма программирования, основанная на автоматическом доказательстве теорем. Его основной идеей является автоматическое доказательство правильности программы на основе логических правил и ограничений, которые описывают требуемое поведение программы.

            В JavaScript логическое программирование может быть реализовано с помощью таких конструкций, как операторы сравнения (==, !=, >, <, >=, <=) и логические операторы (&&, ||, !).

                  </h3>

          <h4 class="container-body-h-plusminus">Плюсы подхода</h4>

          <ul class="container-body-plus">

            <li>Простота и понятность</li>

            <li>Гибкость</li>

            <li>Меньшее количество кода</li>

            <li>Легкость реализации</li>

            <li>Хорошая поддержка в старых браузерах</li>

          </ul>

          <h4 class="container-body-h-plusminus">Минусы подхода</h4>

          <ul class="container-body-minus">

            <li>Меньшая читаемость кода</li>

            <li>Ограниченная производительность/li>

            <li>Возможность возникновения ошибок</li>

            <li>Ограниченная поддержка параллелизма</li>

            <li>Трудности с масштабированием</li>

          </ul>

      <p>Примеры: </p>

    <select id="formSelectorRight" onchange="changeFormRight()">

      <option value="formRight1">Поступление в университет</option>

      <option value="formRight2">Родственные связи</option>

    </select>

    <div id="selectedFormRight">

      <form id="formRight1" style="display:none;">

        <input type="number" id="examScores1" placeholder="Введите баллы за первый экзамен">

        <input type="number" id="examScores2" placeholder="Введите баллы за второй экзамен">

        <input type="number" id="examScores3" placeholder="Введите баллы за третий экзамен">

        <div id="outrig1"></div>

        <button onclick="exam()"  type="button">Поступлю ли я?</button>

        <p>\*примечание: поступление начинается от 60 баллов по каждому предмету</p>

          <script type="text/javascript">

            function exam() {

              const examScores1 = document.getElementById('examScores1').value;

              const examScores2 = document.getElementById('examScores2').value;

              const examScores3 = document.getElementById('examScores3').value;

              if (examScores1 >= 60 && examScores2 >= 60 && examScores3 >= 60) alert('Вы поступили в университет');

              else alert('Вы не поступили в университет');

            }

          </script>

      </form>

      <form id="formRight2" style="display:none;">

        <script type="text/javascript">

          // - База и стандартные предикаты -----------------------------

          var mans = ["Иван", "Артём", "Саша", "Степан", "Кирилл"];

          var woms = ["Наташа","Маша","Ира"];

          var mothers = {

                  "Ира" : ["Маша","Наташа","Степан"],

                  "Наташа" : ["Артём","Иван"]

                  };

          var fathers = {

                  "Саша" : ["Иван","Артём"],

                  "Кирилл" : ["Ира"]

                }

          function man(value) {

            if(mans.includes(value)) return true;

            return false;

          }

          function wom(value) {

            if(woms.includes(value)) return true;

            return false;

          }

          function isSinglePoint(arr1, arr2) {

            for(var i = 0; i<arr1.length; i++) {

              for(var j = 0; j<arr2.length; j++)

              {

                if(arr1[i] == arr2[j])

                {

                  return true;

                }

              }

            }

            return false;

          }

          function isFather(ft, ch) {

            if(fathers[ft]) {

              var childs = fathers[ft];

              return childs.inlcudes(ch);

            }

            return false;

          }

          function isMother(mt, ch) {

            if(mothers[mt]) {

              var childs = mothers[mt];

              return childs.inlcudes(ch);

            }

            return false;

          }

          function getMotherOf(ch) {

            for (var k in mothers){

              if (mothers.hasOwnProperty(k)) {

                var arr = mothers[k];

                if(arr.includes(ch)) {

                  return k;

                }

              }

            }

            return "";

          }

          function getFatherOf(ch) {

            for (var k in fathers){

              if (fathers.hasOwnProperty(k)) {

                var arr = fathers[k];

                if(arr.includes(ch)) {

                  return k;

                }

              }

            }

            return "";

          }

          // - Дополнительные предикаты-----------------------------------

          function isSister(v1,v2) {

            return wom(v1) && wom(v2) && (getMotherOf(v1) == getMotherOf(v2));

          }

          function isBrother(v1,v2) {

            return man(v1) && man(v2) && (getMotherOf(v1) == getMotherOf(v2));

          }

          function isMarried(v1, v2) {

            return man(v1) && wom(v2) && isSinglePoint(fathers[v1],mothers[v2]);

          }

          function isRodstv(v1, v2) {

            if(isSister(v1,v2))

              return true;

            else

            {

              if(isBrother(v1,v2))

                return true;

              else {

                 var m1 = getMotherOf(v1);

                 var m2 = getMotherOf(v2);

                if(m1 != "" && m2!= "") {

                  return m1==m2;

                }

              }

            }

            return false;

          }

          function isDyadya(v1, v2) {

            z = getMotherOf(v2);

            if(isRodstv(v1,z))

            {

              return true;

            }

            return false;

          }

          function isDedushka(v1, v2) {

            var z = fathers[v1];

            var z2 = mothers[getMotherOf(v2)];

            if(!z)return false;

            if(!z2)return false;

            if(z.includes(getMotherOf(v2)))

              return true;

            return false;

          }

          // - Функции HTML страницы -------------------------------------

          function printDB()

          {

            //Написать базу данных

            var bd = "Ира мать Наташи, Маши, Степана<br>";

            bd+="Наташа мать Артёма и Ивана<br>";

            bd+="Саша отец Ивана и Артёма<br>";

            bd+="Кирилл отец Иры<br>";

            var msg = `<font color="green"> ${bd} </font>`;

            document.getElementById("div\_1").innerHTML = msg;

          }

          function run() {

            let v1 = document.getElementById("var1").value;

            let v2 = document.getElementById("var2").value;

            var result = isSister(v1,v2);

            var answer = (result === true) ? "Да" : "Нет";

            var msg = `<font color="blue">Ответ: ${answer}</font>`;

            document.getElementById("div\_2").innerHTML = msg;

          }

          function run2() {

            let v1 = document.getElementById("var1").value;

            let v2 = document.getElementById("var2").value;

            var result = isBrother(v1,v2);

            var answer = (result === true) ? "Да" : "Нет";

            var msg = `<font color="blue">Ответ: ${answer}</font>`;

            document.getElementById("div\_2").innerHTML = msg;

          }

          function run3() {

            let v1 = document.getElementById("var1").value;

            let v2 = document.getElementById("var2").value;

            var result = isDedushka(v1,v2);

            var answer = (result === true) ? "Да" : "Нет";

            var msg = `<font color="blue">Ответ: ${answer}</font>`;

            document.getElementById("div\_2").innerHTML = msg;

          }

          function run4() {

            let v1 = document.getElementById("var1").value;

            let v2 = document.getElementById("var2").value;

            var result = isDyadya(v1,v2);

            var answer = (result === true) ? "Да" : "Нет";

            var msg = `<font color="blue">Ответ: ${answer}</font>`;

            document.getElementById("div\_2").innerHTML = msg;

          }

          function run5() {

            let v1 = document.getElementById("var1").value;

            let v2 = document.getElementById("var2").value;

            var result = isMarried(v1,v2);

            var answer = (result === true) ? "Да" : "Нет";

            var msg = `<font color="blue">Ответ: ${answer}</font>`;

            document.getElementById("div\_2").innerHTML = msg;

          }

          function run6() {

            let v1 = document.getElementById("var1").value;

            let v2 = document.getElementById("var2").value;

            var result = isRodstv(v1,v2);

            var answer = (result === true) ? "Да" : "Нет";

            var msg = `<font color="blue">Ответ: ${answer}</font>`;

            document.getElementById("div\_2").innerHTML = msg;

          }

         </script>

         <form name ="myForm" onload="printDB()">

           <br>

           <b>Человек 1:</b>

           <input type="text" id="var1" size=6 maxlength=8>

           <br><br>

           <b>Человек 2: </b>

           <input type="text" id="var2" size=6 maxlength=8>

           <br><br>

           <input type = "button" name = "button1" value = "Сёстрами" onclick="run();"><br><br>

           <input type = "button" name = "button1" value = "Братьями" onclick="run2();"><br><br>

           <input type = "button" name = "button1" value = "Дедушкой и внуком(внучкой)" onclick="run3()"><br><br>

           <input type = "button" name = "button1" value = "Дядей и племянником" onclick="run4()"><br><br>

           <input type = "button" name = "button1" value = "Супругами" onclick="run5();"><br><br>

           <input type = "button" name = "button1" value = "Родственниками" onclick="run6()"><br><br>

           База данных:

           <br><br>

           <div id="div\_1"><script>printDB();</script></div>

           <div id="div\_2"></div>

           <br><br>

         </form>

      </form>

      <form id="formRight3" style="display:none;">

        <label for="input3">Input 3:</label>

        <input type="text" id="input3">

      </form>

    </div>

        </div>

      </div>

    </div>

    <script>

      function changeFormRight() {

            var selectedValue = document.getElementById("formSelectorRight").value;

        document.getElementById("formRight1").style.display = "none";

        document.getElementById("formRight2").style.display = "none";

        document.getElementById("formRight3").style.display = "none";

        if (selectedValue === "formRight1")

          document.getElementById("formRight1").style.display = "block";

         else if (selectedValue === "formRight2")

          document.getElementById("formRight2").style.display = "block";

         else if (selectedValue === "formRight3")

          document.getElementById("formRight3").style.display = "block";

      }

      function changeFormLeft() {

var selectedValue = document.getElementById("formSelectorLeft").value;

document.getElementById("formLeft1").style.display = "none";

document.getElementById("formLeft2").style.display = "none";

document.getElementById("formLeft3").style.display = "none";

if (selectedValue === "formLeft1")

document.getElementById("formLeft1").style.display = "block";

else if (selectedValue === "formLeft2")

document.getElementById("formLeft2").style.display = "block";

else if (selectedValue === "formLeft3")

document.getElementById("formLeft3").style.display = "block";

}

    </script>

  </body>

</body>

</html>

Style.css

\* {

  font-family:'Lucida Sans', 'Lucida Sans Regular', 'Lucida Grande', 'Lucida Sans Unicode', Geneva, Verdana, sans-serif ;

}

.container-body {

  border: 3px solid teal;

  padding: 1vh;

  margin-left: 0.5vh;

  margin-right: 0.5vh;

}

button {

  background-color: transparent;

  border: 2px solid teal;

}

button:hover {

  background-color: teal;

  color:white;

}

select {

  background-color: transparent;

  border: 2px solid teal;

}

select:hover {

  background-color: teal;

  color:white;

}

input {

  margin-top: 5px;

  margin-bottom: 5px;

  border: 2px solid teal;

}

.zagolovok {

  display: flex;

  justify-content: center;

  font-size: 40px;

}