**What is javascript?**

JavaScript is a **high-level**, **interpreted** programming language commonly used to create and control **dynamic website content.**

Alongside HTML and CSS, it is one of the core technologies of the World Wide Web.

Here are some key aspects of JavaScript:

1. **Client-Side Scripting:**
   1. JavaScript is primarily used for client-side scripting, allowing developers to create interactive web pages.
   2. This means it runs in the user's browser and can respond to user actions like clicks, form submissions, and other events.
2. **Dynamic Content:**
   1. JavaScript can change HTML content, styles, and attributes on the fly, making web pages more dynamic and interactive.
   2. This includes tasks like updating the content of a web page without reloading it, creating interactive forms, and implementing animations.
3. **Libraries and Frameworks:**
   1. JavaScript has a rich ecosystem of libraries (like jQuery) and frameworks (like React, Angular, and Vue.js) that simplify complex tasks and enhance development efficiency.
   2. These tools provide pre-built functions and components that speed up development.
4. **Server-Side Development:**
   1. With the introduction of Node.js, JavaScript can also be used for server-side development.
   2. This means developers can use JavaScript to write code that runs on servers, handling tasks like database interactions, server-side logic, and more.
5. **Asynchronous Programming:**
   1. JavaScript supports asynchronous programming through callbacks, promises, and async/await syntax.
   2. This allows developers to write non-blocking code that can handle multiple tasks simultaneously, improving performance and user experience.
6. **Versatility:**
   1. JavaScript is versatile and can be used for various tasks beyond web development, such as game development, mobile app development (using frameworks like React Native), and even desktop applications (using Electron).
7. **Event-Driven:**
   1. JavaScript is event-driven, meaning it can listen for and respond to events (e.g., user clicks, form submissions) in real-time, providing a responsive and interactive experience.
8. **Standardized:**
   1. JavaScript is standardized through the ECMAScript specification, which ensures consistency and compatibility across different browsers and platforms.

Overall, JavaScript is an essential language for modern web development, enabling developers to create rich, interactive experiences on the web.

**Contents of JavaScript**

1. **Introduction to JavaScript**
   * What is JavaScript?
   * Where to use JavaScript (Client-side, Server-side with Node.js)
   * JavaScript Execution Environment (Browser, Node.js)
   * Setting up Development Environment (Browser Developer Tools, Editors like VS Code)
2. **Data Types and Variables**
   * Primitive Data Types (String, Number, Boolean, Null, Undefined, Symbol, BigInt)
   * Complex Data Types (Objects, Arrays, Functions)
   * Variables (let, const, var)
   * Type Conversion and Coercion
3. **Operators**
   * Arithmetic, Comparison, Logical, Assignment, Bitwise Operators
   * Conditional (Ternary) Operator
   * Precedence and Associativity of Operators
4. **Control Structures**
   * Conditional Statements (if, else, switch)
   * Loops (for, while, do-while, for...of, for...in)
   * Iteration Protocols (Iterator, Iterable)
5. **Functions**
   * Function Declarations and Expressions
   * Function Parameters (default, rest parameters)
   * Return Statements and undefined
   * **Closures** (Lexical Environment, Access to Outer Scope)
   * Arrow Functions (=>)
   * Higher-order Functions (Functions that return or accept functions)
   * **Immediately Invoked Function Expressions (IIFE)**
6. **Objects and Arrays**
   * Object Literals and Constructors
   * Accessing, Adding, and Removing Properties
   * **Object Methods (this keyword)** and Method Binding
   * Arrays: Creating, Accessing, Modifying, Looping through Arrays
   * **Array Methods** (push, pop, map, filter, reduce, find, some, every, etc.)
   * Destructuring Assignment (Objects and Arrays)
   * **Spread Operator** and **Rest Operator**
   * **Shallow Copy vs. Deep Copy** of Objects and Arrays
7. **Prototype and Inheritance**
   * Prototypes and Prototype Chain
   * Inheritance in JavaScript
   * Object.create() and Prototype-based Inheritance
   * **Class Syntax** in ES6+
   * Constructors and super()
   * Inheritance in Classes
8. **DOM Manipulation**
   * Introduction to the DOM
   * Selecting Elements (getElementById, querySelector, querySelectorAll)
   * Modifying Element Content (innerHTML, textContent)
   * Modifying Element Attributes (setAttribute, removeAttribute)
   * Modifying Styles with JavaScript (style object)
   * Event Handling (addEventListener, Event Bubbling, Event Delegation)
   * Creating and Removing Elements (createElement, appendChild, removeChild)
9. **Forms and Input Validation**
   * Handling Forms in JavaScript
   * Retrieving Form Data
   * Validating Input (Basic Validation, Regular Expressions)
   * Preventing Default Form Submissions (preventDefault())
   * Creating Custom Validation Messages
10. **Object-Oriented Programming (OOP)**

* Constructors and Instances
* Class-Based vs. Prototype-Based Inheritance
* Encapsulation, Polymorphism, and Inheritance
* Getters and Setters
* Static Methods
* Composition vs. Inheritance

1. **Asynchronous JavaScript**

* **Callbacks**
* Promises (resolve, reject, then, catch, finally)
* async / await
* Error Handling in Asynchronous Code (try, catch)
* **Fetching Data from APIs** (using fetch and XMLHttpRequest)
* **AJAX** and HTTP Requests (GET, POST, PUT, DELETE)

1. **Error Handling**

* try, catch, finally Blocks
* Throwing Custom Errors
* **Creating Custom Error Types**

1. **JavaScript Modules**

* **ES6 Modules** (import/export)
* Default and Named Exports
* **CommonJS Modules** (used in Node.js)
* Bundling Modules (with tools like Webpack)

1. **Event Loop and JavaScript Engine**

* JavaScript's Single Threaded Nature
* Event Loop (Call Stack, Task Queue)
* **Microtasks** and **Macrotasks**
* Non-blocking I/O with Event Loop

1. **Local Storage, Session Storage, and Cookies**

* **localStorage** and **sessionStorage**
* Storing and Retrieving Data from Browser Storage
* JSON Storage in Local/Session Storage
* **Cookies** and Managing Cookies with JavaScript

1. **Regular Expressions**

* Regular Expressions Syntax (/pattern/flags)
* Commonly Used Methods (test, match, replace, search)
* Validating Patterns in Strings (e.g., emails, phone numbers)

1. **JSON and AJAX**

* Introduction to JSON (JavaScript Object Notation)
* Parsing JSON with JSON.parse()
* Stringifying JavaScript Objects with JSON.stringify()
* AJAX Requests and Interacting with REST APIs

1. **JavaScript Performance**

* Optimizing JavaScript Code (Execution, Memory, Reflows)
* **Debouncing** and **Throttling**
* Managing Heavy Computations in JavaScript (e.g., Web Workers)

1. **JavaScript Design Patterns**

* Singleton Pattern
* Factory Pattern
* Observer Pattern
* Module Pattern
* Revealing Module Pattern

1. **Testing JavaScript**

* **Unit Testing** (Jest, Mocha, Jasmine)
* Writing Testable Code
* Asynchronous Testing

1. **Advanced Topics**

* **Proxies** and Reflect API
* Generators and yield
* Iterators and Iterables
* Symbol and its use cases
* Web Workers for Multithreading
* **WeakMap** and **WeakSet**

1. **JavaScript in Front-End Frameworks**

* Introduction to Modern JavaScript Frameworks (React, Angular, Vue.js)
* Setting up Frameworks with JavaScript
* Component-Based Architecture in Frameworks

**Basics of Javascript**

**1. Syntax and Variables**

**Variables** are used to store data values.

You can declare a variable using **var**, **let**, or **const**.

var name = "John"; // Declares a variable that can be reassigned

let age = 25; // Declares a block-scoped variable that can be reassigned

const pi = 3.14; // Declares a block-scoped variable that cannot be reassigned

**2. Data Types**

JavaScript supports various data types, including:

* **Primitive Types:**
  + String: "Hello"
  + Number: 42
  + Boolean: true or false
  + Null: null
  + Undefined: undefined
  + Symbol: Symbol('id')
  + BigInt: 123n
* **Composite Types:**
  + Object: { name: "John", age: 25 }
  + Array: [1, 2, 3]

**3. Operators**

JavaScript provides various operators:

* **Arithmetic Operators:** +, -, \*, /, %
* **Assignment Operators:** =, +=, -=, \*=, /=
* **Comparison Operators:** ==, ===, !=, !==, >, <, >=, <=
* **Logical Operators:** &&, ||, !

**4. Control Structures(decisions)**

**Conditional Statements** allow you to execute code based on conditions.

if (age > 18) {

console.log("Adult");

} else {

console.log("Minor");

}

**Loops** allow you to execute code multiple times.

* **For Loop:**

for (let i = 0; i < 5; i++) {

console.log(i);

}

* **While Loop:**

let i = 0;

while (i < 5) {

console.log(i);

i++;

}

**5. Functions**

Functions are reusable blocks of code.

function greet(name) {

return "Hello, " + name;

}

console.log(greet("Alice"));

**6. Objects and Arrays**

**Objects** store key-value pairs.

let person = {

name: "John",

age: 25,

greet: function() {

console.log("Hello, " + this.name);

}

};

person.greet();

**Arrays** store ordered collections of values.

let numbers = [1, 2, 3, 4, 5];

console.log(numbers[0]); // Outputs 1

numbers.push(6); // Adds a new element to the array

console.log(numbers);

**7. Event Handling**

JavaScript can handle events like user actions.

document.getElementById("myButton").addEventListener("click", function() {

alert("Button was clicked!");

});

**8. DOM Manipulation**

JavaScript can manipulate the Document Object Model (DOM) to change the content of a webpage.

document.getElementById("myElement").innerHTML = "New Content";

document.getElementById("myElement").style.color = "red";

**9. Promises and Async/Await**

JavaScript supports asynchronous programming using Promises and the async/await syntax.

* **Promise:**

let promise = new Promise(function(resolve, reject) {

// Some async operation

if (success) {

resolve("Success");

} else {

reject("Error");

}

});

promise.then(function(result) {

console.log(result);

}).catch(function(error) {

console.error(error);

});

* **Async/Await:**

async function fetchData() {

try {

let response = await fetch('https://api.example.com/data');

let data = await response.json();

console.log(data);

} catch (error) {

console.error('Error:', error);

}

}

fetchData();

These basics provide a foundation for learning JavaScript. As you practice and build more projects, you'll become more comfortable with its syntax and capabilities.

**document.querySelector()**

The document.querySelector() method in JavaScript is used to select the first element in the DOM that matches a specified CSS selector.

It allows you to find and manipulate specific HTML elements based on their tag name, class, ID, or other attributes.

**Syntax:**

let element = document.querySelector(selector);

**Parameters:**

* **selector**: A string containing one or more CSS selectors to match the desired element. It can be:
  + **Tag** **name**: "div", "h1", etc.
  + **Class** **name**: ".class-name"
  + **ID**: "#id-name"
  + **Attribute**: "[attribute=value]"

**Example Usage:**

1. **Select an element by class**:

let element = document.querySelector('.myClass');

1. **Select an element by ID**:

let element = document.querySelector('#myId');

1. **Select a nested element**:

let element = document.querySelector('div > p');

1. **Select an input with a specific attribute**:

let element = document.querySelector('input[type="text"]');

**Key Points:**

* **First Match**: It returns only the first element that matches the selector. If no match is found, it returns null.
* **CSS Selectors**: It supports complex CSS selectors, so you can target elements more precisely.

**Example:**

<!DOCTYPE html>

<html>

<head>

<title>querySelector Example</title>

</head>

<body>

<div class="container">

<p id="first-paragraph">This is the first paragraph.</p>

<p>This is the second paragraph.</p>

</div>

<script>

let firstParagraph = document.querySelector('#first-paragraph');

console.log(firstParagraph.textContent); // Outputs: "This is the first paragraph."

</script>

</body>

</html>

In this example, document.querySelector('#first-paragraph') selects the paragraph with the id="first-paragraph" and logs its content.

**Decisions in Javascript**

In JavaScript, decision-making is primarily handled through conditional statements.

These statements allow you to execute different blocks of code based on certain conditions.

The main types of conditional statements in JavaScript are if, else if, else, and switch.

Here’s a detailed look at each:

**1. if Statement**

The if statement is used to execute a block of code if a specified condition is true.

let age = 20;

if (age >= 18) {

console.log("You are an adult.");

}

**2. else Statement**

The else statement is used to execute a block of code if the condition in the if statement is false.

let age = 15;

if (age >= 18) {

console.log("You are an adult.");

} else {

console.log("You are a minor.");

}

**3. else if Statement**

The else if statement is used to specify a new condition if the first condition is false.

let score = 85;

if (score >= 90) {

console.log("Grade: A");

} else if (score >= 80) {

console.log("Grade: B");

} else if (score >= 70) {

console.log("Grade: C");

} else {

console.log("Grade: F");

}

**4. switch Statement**

The switch statement is used to perform different actions based on different conditions. It is a good alternative when you need to compare the same value to several different possible matches.

let day = 3;

let dayName;

switch (day) {

case 1:

dayName = "Monday";

break;

case 2:

dayName = "Tuesday";

break;

case 3:

dayName = "Wednesday";

break;

case 4:

dayName = "Thursday";

break;

case 5:

dayName = "Friday";

break;

case 6:

dayName = "Saturday";

break;

case 7:

dayName = "Sunday";

break;

default:

dayName = "Invalid day";

}

console.log(dayName); // Outputs: Wednesday

**5. Ternary Operator**

The ternary operator is a shorthand for the if-else statement. It is useful for making simple decisions.

let age = 20;

let access = (age >= 18) ? "Granted" : "Denied";

console.log(access); // Outputs: Granted

**Example: Combining Conditions**

You can combine multiple conditions using logical operators like && (AND), || (OR), and ! (NOT).

let age = 25;

let hasPermission = true;

if (age >= 18 && hasPermission) {

console.log("Access granted.");

} else {

console.log("Access denied.");

}

if (age < 18 || !hasPermission) {

console.log("Access denied.");

} else {

console.log("Access granted.");

}

**Summary**

* Use if to specify a block of code to be executed if a condition is true.
* Use else to specify a block of code to be executed if the same condition is false.
* Use else if to specify a new condition to test if the first condition is false.
* Use switch to specify many alternative blocks of code to be executed.
* Use the ternary operator for simple conditional expressions.

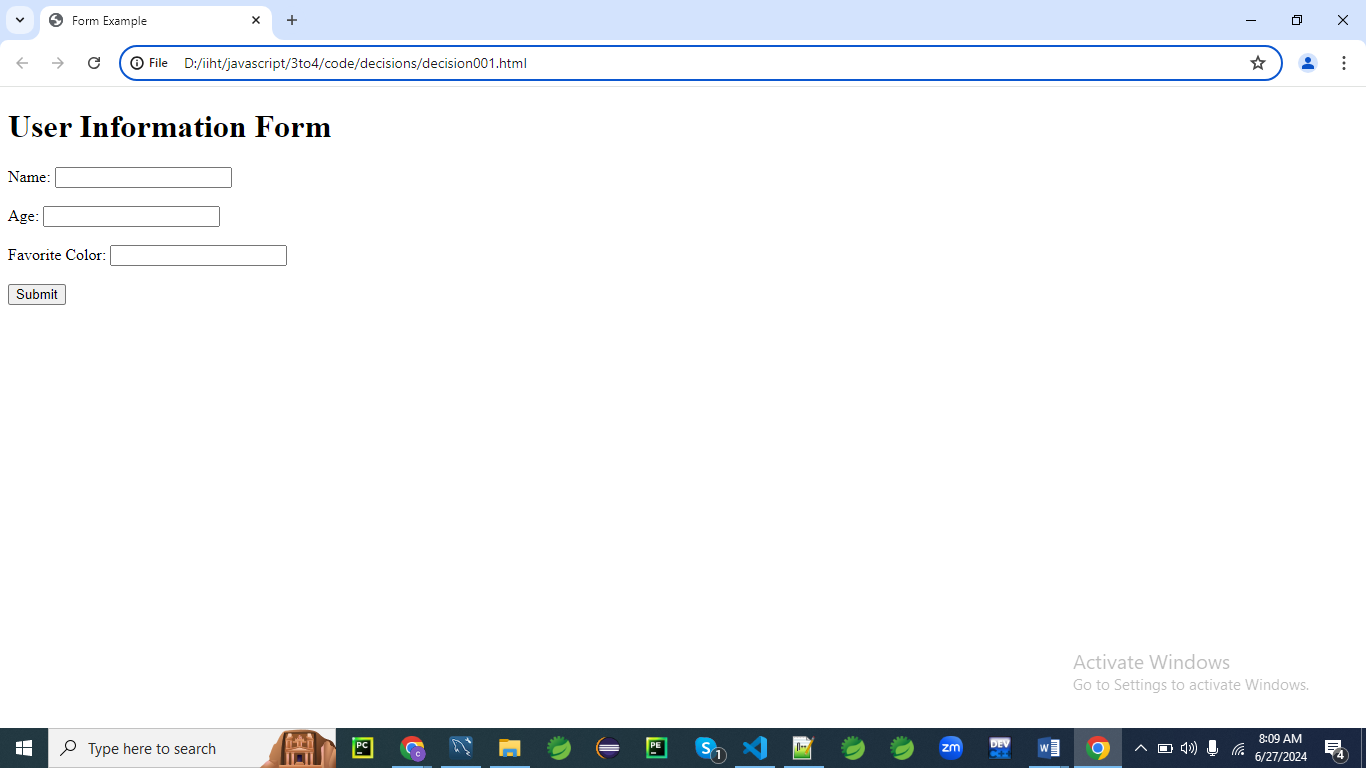
Understanding these basic decision-making structures will allow you to write more dynamic and responsive JavaScript code.

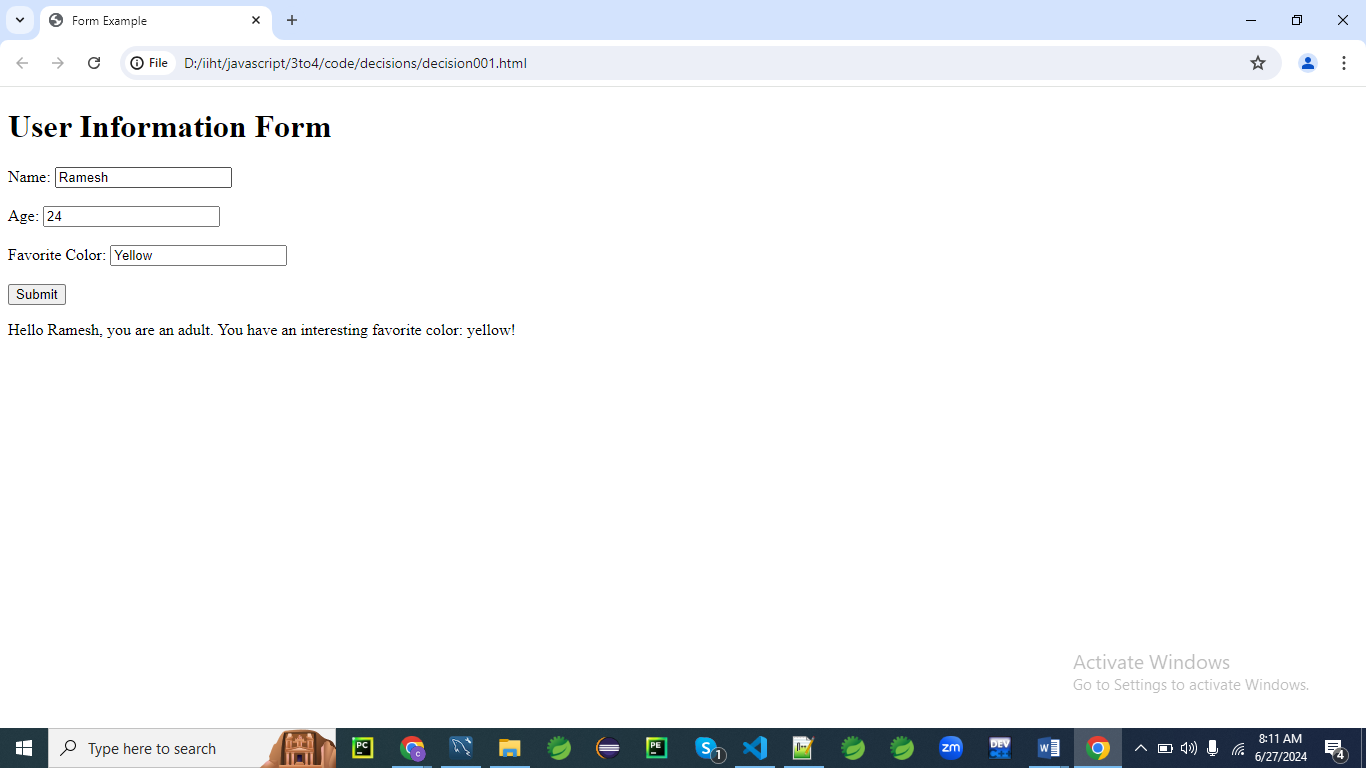
**Example 01**

Here's an example program in JavaScript that uses a form to demonstrate decision-making.

This program will validate user input from a form, checking the age and favorite color provided by the user.

Depending on the input, it will display different messages.





**HTML Form**

First, let's create an HTML form with fields for name, age, and favorite color.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Form Example</title>

</head>

<body>

<h1>User Information Form</h1>

<form id="userForm">

<label for="name">Name:</label>

<input type="text" id="name" name="name" required><br><br>

<label for="age">Age:</label>

<input type="number" id="age" name="age" required><br><br>

<label for="color">Favorite Color:</label>

<input type="text" id="color" name="color" required><br><br>

<button type="submit">Submit</button>

</form>

<p id="message"></p>

<script src="script001.js"></script>

</body>

</html>

**JavaScript Code(script001.js)**

Now, let's write the JavaScript code that handles the form submission and includes various decision-making structures.

document.getElementById('userForm').addEventListener('submit', function(event) {

event.preventDefault(); // Prevent the form from submitting the traditional way

// Get form values

let name = document.getElementById('name').value;

let age = parseInt(document.getElementById('age').value);

let color = document.getElementById('color').value.toLowerCase();

// Decision-making logic

let message = '';

if (age >= 0 && age < 18) {

message = `Hello ${name}, you are a minor.`;

}

else if (age >= 18 && age < 60) {

message = `Hello ${name}, you are an adult.`;

}

else if (age >= 60) {

message = `Hello ${name}, you are a senior citizen.`;

}

else {

message = `Invalid age entered.`;

}

// Using switch-case for favorite color

switch (color) {

case 'red':

message += ' Your favorite color is Red!';

break;

case 'blue':

message += ' Your favorite color is Blue!';

break;

case 'green':

message += ' Your favorite color is Green!';

break;

default:

message += ` You have an interesting favorite color: ${color}!`;

}

// Display the message

document.getElementById('message').innerText = message;

});

**Explanation**

1. **HTML Form:**
   * The form contains three input fields: name, age, and favorite color.
   * When the form is submitted, the submit event is triggered.
2. **JavaScript Code:**
   * An event listener is added to handle the form submission.
   * The event.preventDefault() method prevents the form from submitting in the traditional way, allowing us to handle the submission with JavaScript.
   * The values from the form fields are retrieved and stored in variables.
   * Conditional statements (if, else if, else) are used to determine the message based on the user's age.
   * A switch statement is used to add a specific message based on the user's favorite color.
   * The final message is displayed in the paragraph with the id message.

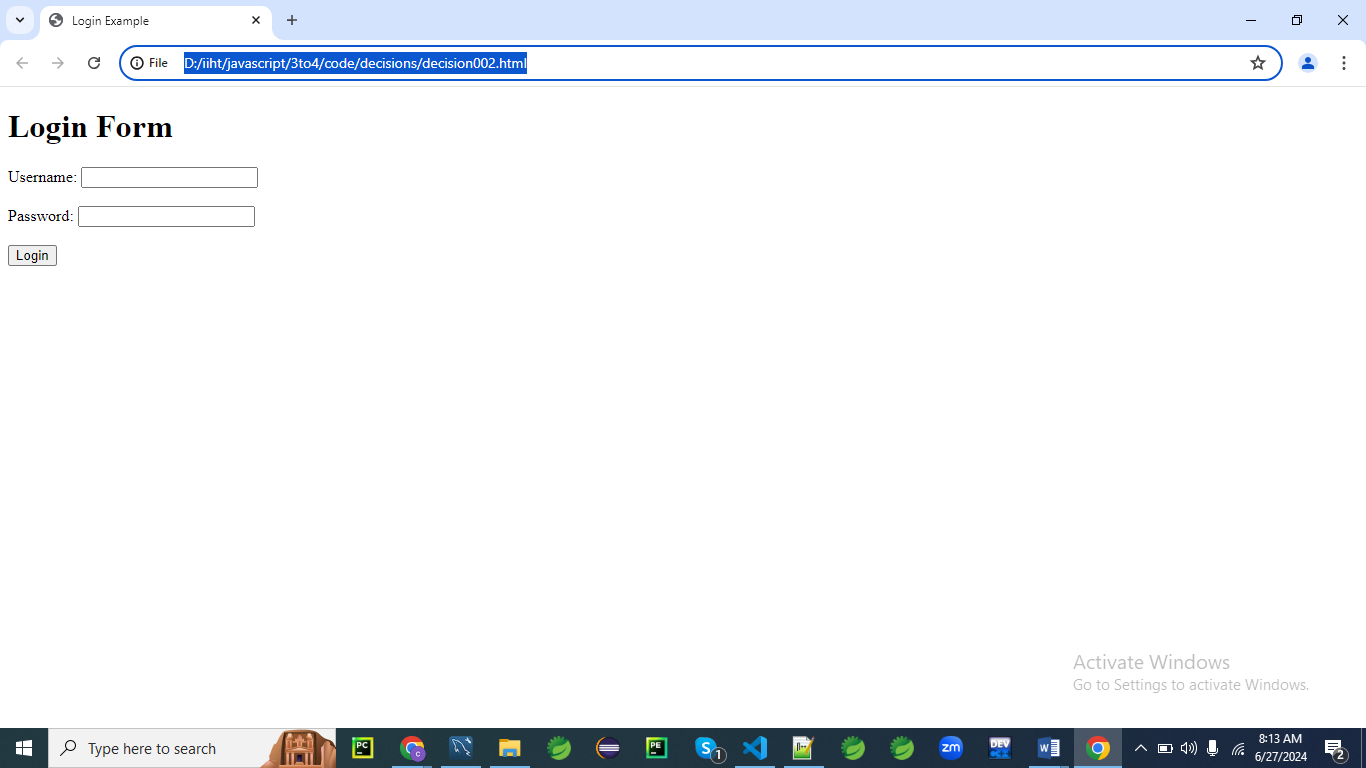
This example demonstrates how to use various decision-making structures in JavaScript to handle and validate form input.

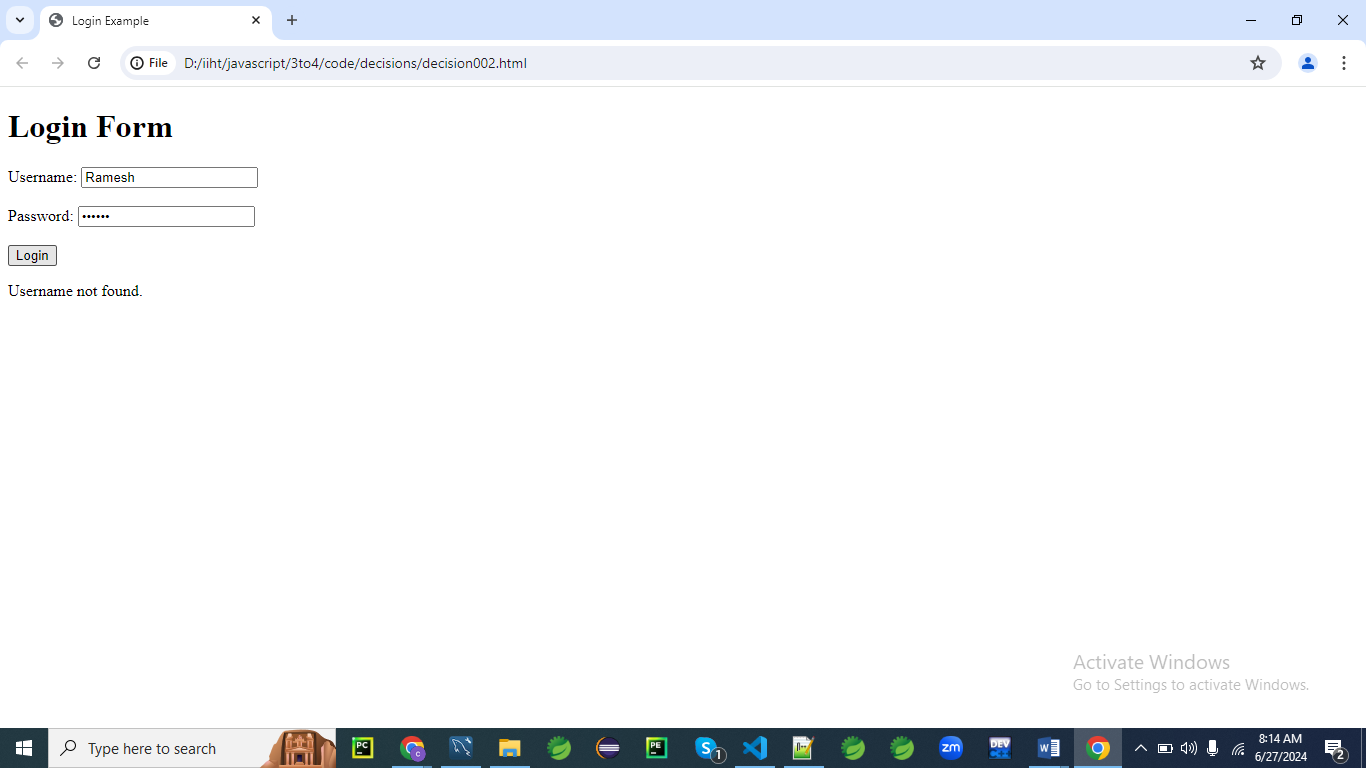
**Example 02**

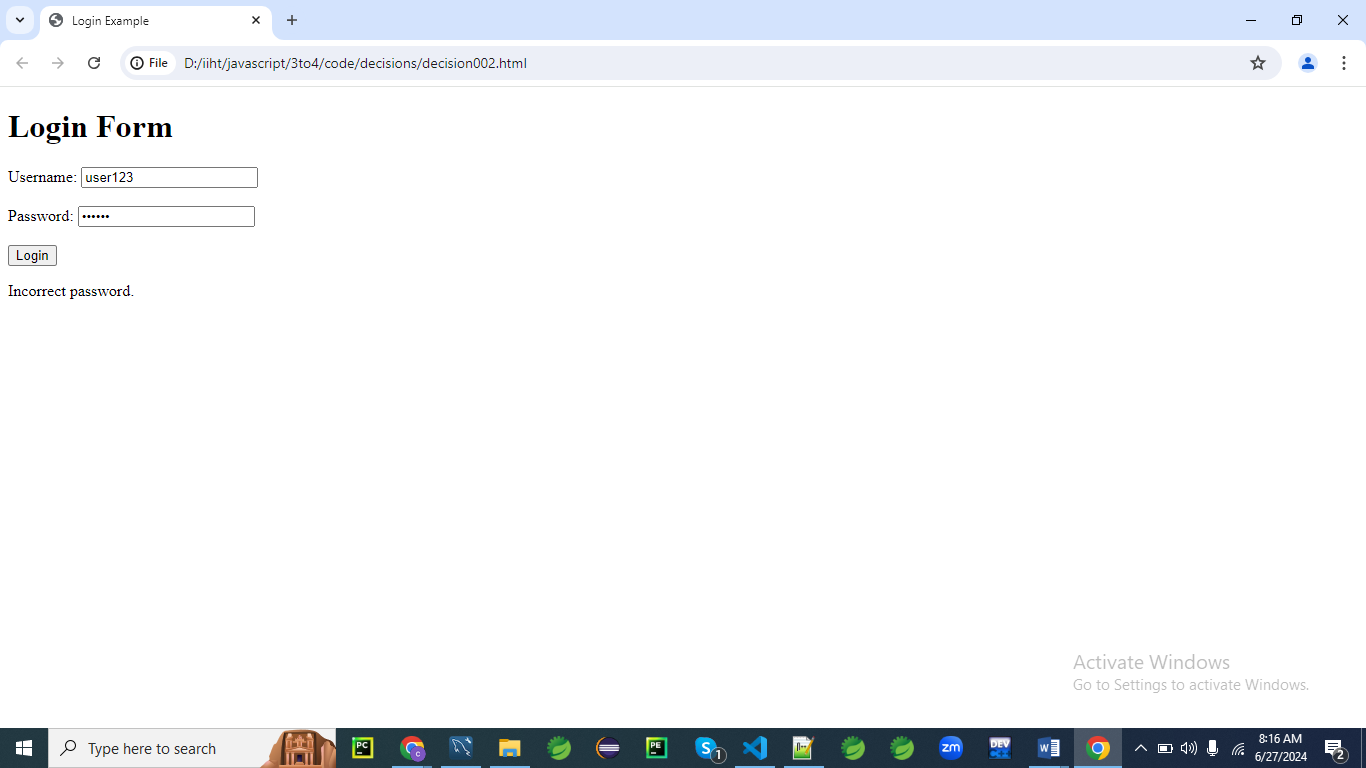
Here's another example program in JavaScript that uses a form to demonstrate decision-making.

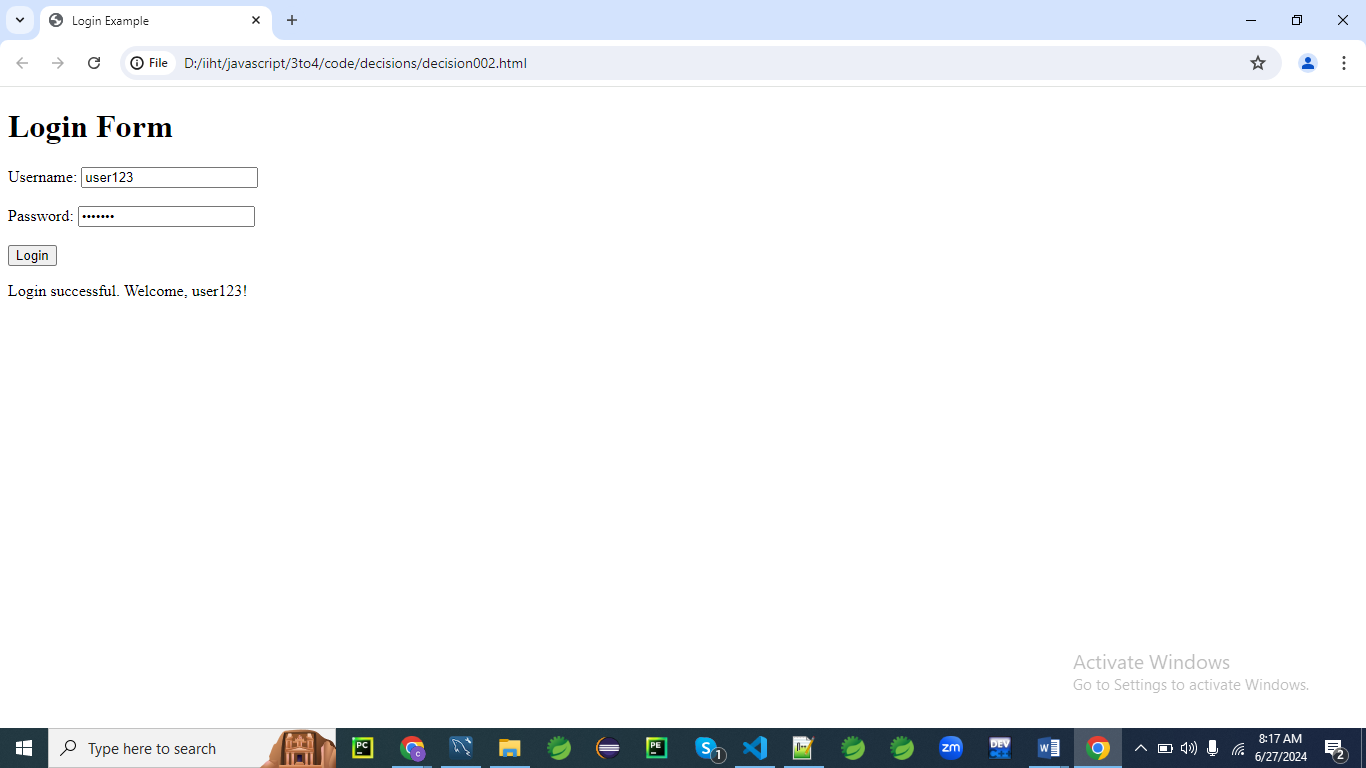
This example will include a login form where the user must enter a username and password.

The program will check the input against predefined values and display appropriate messages based on the validity of the credentials.









**HTML Form**

First, let's create an HTML form with fields for username and password.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Login Example</title>

</head>

<body>

<h1>Login Form</h1>

<form id="loginForm">

<label for="username">Username:</label>

<input type="text" id="username" name="username" required><br><br>

<label for="password">Password:</label>

<input type="password" id="password" name="password" required><br><br>

<button type="submit">Login</button>

</form>

<p id="message"></p>

<script src="script002.js"></script>

</body>

</html>

**JavaScript Code**

Now, let's write the JavaScript code that handles the form submission and includes various decision-making structures.

document.getElementById('loginForm').addEventListener('submit', function(event) {

event.preventDefault(); // Prevent the form from submitting the traditional way

// Get form values

let username = document.getElementById('username').value;

let password = document.getElementById('password').value;

// Predefined valid credentials

let validUsername = "user123";

let validPassword = "pass123";

// Decision-making logic

let message = '';

// Check if the username is correct

if (username === validUsername) {

// Check if the password is correct

if (password === validPassword) {

message = "Login successful. Welcome, " + username + "!";

} else {

message = "Incorrect password.";

}

} else {

message = "Username not found.";

}

// Display the message

document.getElementById('message').innerText = message;

});

**Explanation**

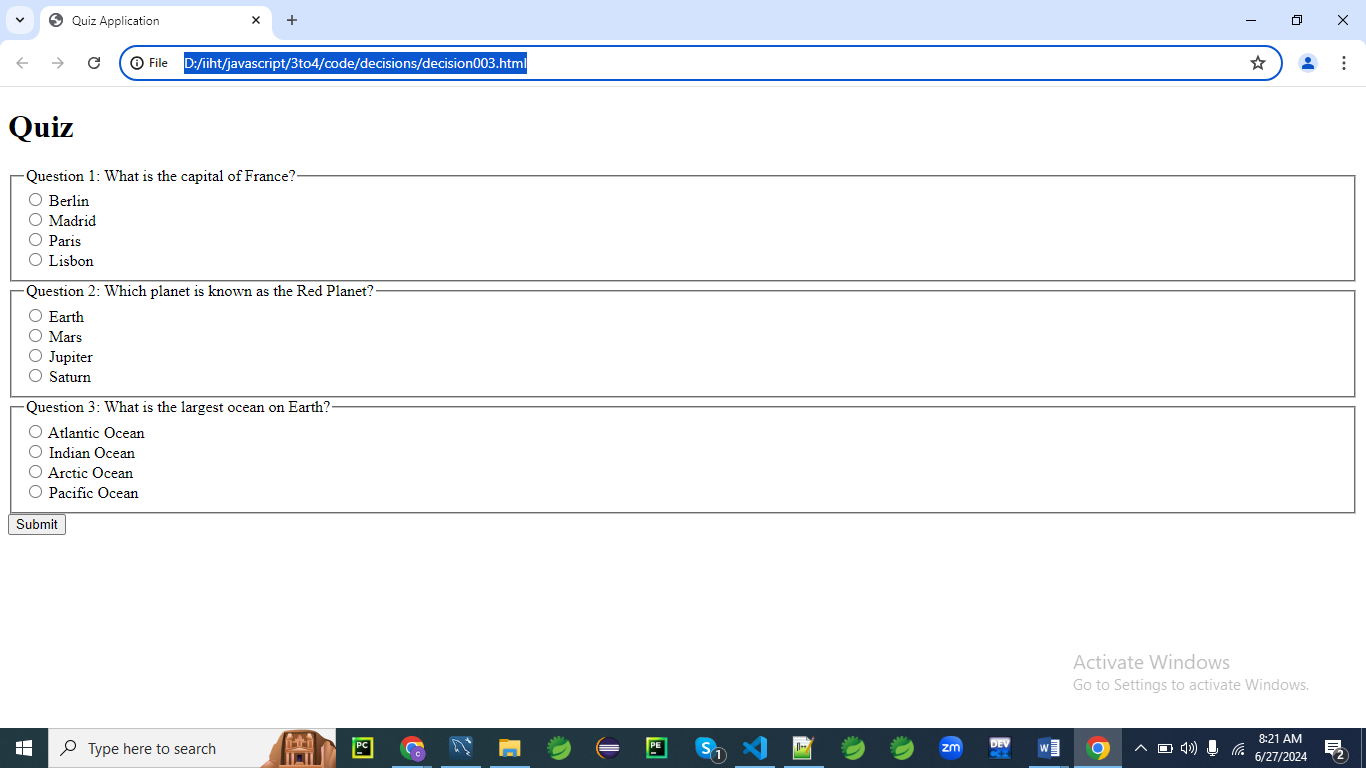
1. **HTML Form:**
   * The form contains two input fields: username and password.
   * When the form is submitted, the submit event is triggered.
2. **JavaScript Code:**
   * An event listener is added to handle the form submission.
   * The event.preventDefault() method prevents the form from submitting in the traditional way, allowing us to handle the submission with JavaScript.
   * The values from the form fields are retrieved and stored in variables.
   * Predefined valid credentials are defined for comparison.
   * Conditional statements (if, else) are used to validate the username and password.
     + First, the username is checked. If it matches the predefined valid username, the password is checked next.
     + If both the username and password are correct, a success message is displayed.
     + If the username is correct but the password is incorrect, an appropriate error message is displayed.
     + If the username is incorrect, a different error message is displayed.
   * The final message is displayed in the paragraph with the id message.

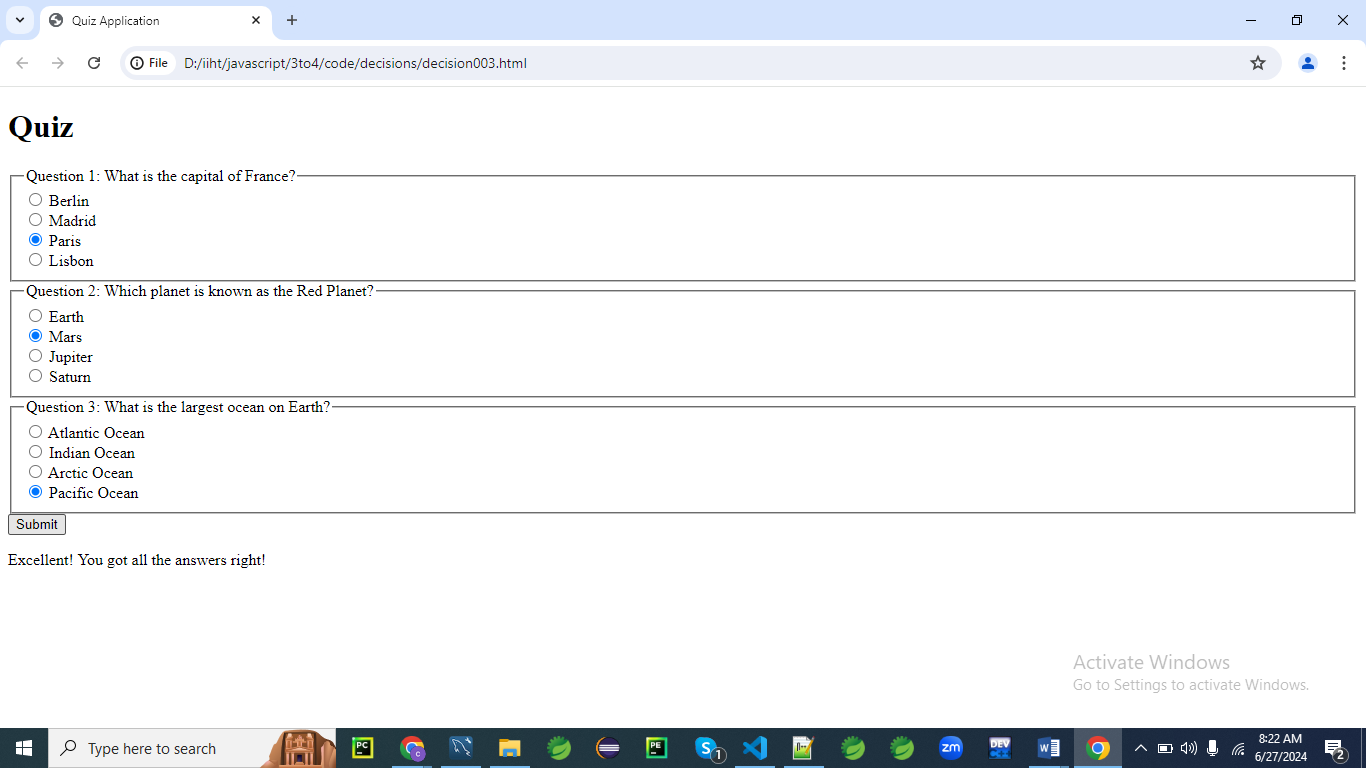
This example demonstrates how to use nested decision-making structures in JavaScript to handle and validate login credentials

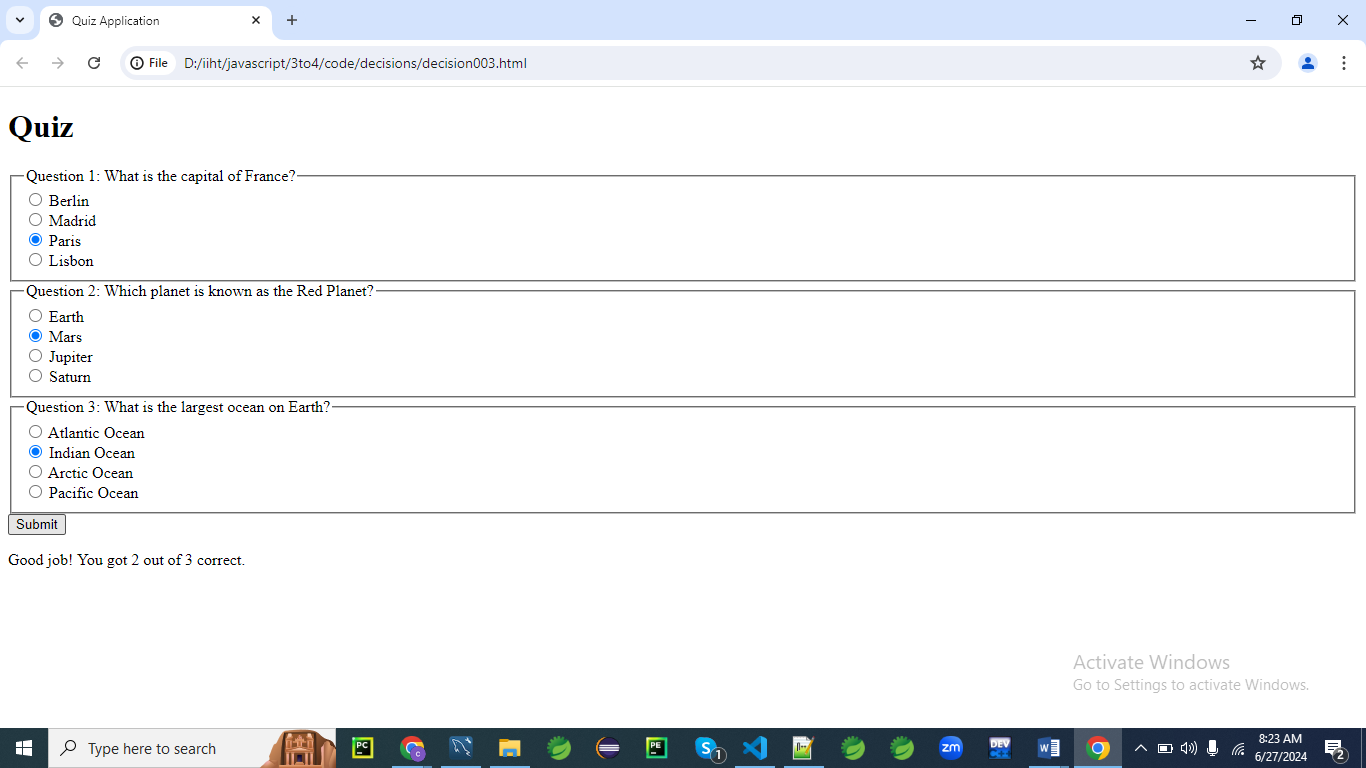
**Example 03**

Let's create a more comprehensive example program in JavaScript that demonstrates a variety of decision-making structures.

This example will be a simple quiz application where users can answer multiple-choice questions, and the application will provide feedback based on their answers.







**HTML Form**

First, let's create an HTML form for the quiz with multiple-choice questions.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Quiz Application</title>

</head>

<body>

<h1>Quiz</h1>

<form id="quizForm">

<fieldset>

<legend>Question 1: What is the capital of France?</legend>

<label>

<input type="radio" name="q1" value="Berlin" required> Berlin

</label><br>

<label>

<input type="radio" name="q1" value="Madrid"> Madrid

</label><br>

<label>

<input type="radio" name="q1" value="Paris"> Paris

</label><br>

<label>

<input type="radio" name="q1" value="Lisbon"> Lisbon

</label><br>

</fieldset>

<fieldset>

<legend>Question 2: Which planet is known as the Red Planet?</legend>

<label>

<input type="radio" name="q2" value="Earth" required> Earth

</label><br>

<label>

<input type="radio" name="q2" value="Mars"> Mars

</label><br>

<label>

<input type="radio" name="q2" value="Jupiter"> Jupiter

</label><br>

<label>

<input type="radio" name="q2" value="Saturn"> Saturn

</label><br>

</fieldset>

<fieldset>

<legend>Question 3: What is the largest ocean on Earth?</legend>

<label>

<input type="radio" name="q3" value="Atlantic" required> Atlantic Ocean

</label><br>

<label>

<input type="radio" name="q3" value="Indian"> Indian Ocean

</label><br>

<label>

<input type="radio" name="q3" value="Arctic"> Arctic Ocean

</label><br>

<label>

<input type="radio" name="q3" value="Pacific"> Pacific Ocean

</label><br>

</fieldset>

<button type="submit">Submit</button>

</form>

<p id="message"></p>

<script src="script.js"></script>

</body>

</html>

**JavaScript Code**

Now, let's write the JavaScript code that handles the form submission and includes various decision-making structures.

document.getElementById('quizForm').addEventListener('submit', function(event) {

event.preventDefault(); // Prevent the form from submitting the traditional way

// Get form values

let q1 = document.querySelector('input[name="q1"]:checked').value;

let q2 = document.querySelector('input[name="q2"]:checked').value;

let q3 = document.querySelector('input[name="q3"]:checked').value;

// Correct answers

// Create an object correctAnswers

let correctAnswers = {

q1: "Paris",

q2: "Mars",

q3: "Pacific"

};

// Decision-making logic

let score = 0;

if (q1 === correctAnswers.q1) {

score++;

}

if (q2 === correctAnswers.q2) {

score++;

}

if (q3 === correctAnswers.q3) {

score++;

}

// Using switch-case for feedback based on score

let feedback = '';

switch (score) {

case 3:

feedback = "Excellent! You got all the answers right!";

break;

case 2:

feedback = "Good job! You got 2 out of 3 correct.";

break;

case 1:

feedback = "You got 1 out of 3 correct. Keep trying!";

break;

case 0:

feedback = "You got all the answers wrong. Better luck next time!";

break;

default:

feedback = "An unexpected error occurred.";

}

// Display the feedback

document.getElementById('message').innerText = feedback;

});

**Explanation**

1. **HTML Form:**
   * The form contains three multiple-choice questions, each with four possible answers.
   * Each question uses radio buttons to allow the user to select one answer.
   * When the form is submitted, the submit event is triggered.
2. **JavaScript Code:**
   * An event listener is added to handle the form submission.
   * The event.preventDefault() method prevents the form from submitting in the traditional way, allowing us to handle the submission with JavaScript.
   * The values of the selected radio buttons are retrieved and stored in variables.
   * Correct answers are predefined in an object.
   * Conditional statements (if) are used to compare the user's answers with the correct answers and calculate the score.
   * A switch statement is used to provide feedback based on the score.
   * The feedback message is displayed in the paragraph with the id message.

This example demonstrates how to use various decision-making structures in JavaScript to handle and validate user input in a quiz application. It includes if statements for condition checking and a switch statement for providing different feedback messages based on the user's score.