BY PRAKASH

JAVASCRIPT .JS

COMPLETE GUIDE

Overview

Introduction to JavaScript:

- What is JavaScript?
- Why use JavaScript?
- Brief history and evolution.

Basic Syntax:

- Variables (var, let, const).
- Data types (Primitive and Reference).
- Operators (+, -, *, /, etc.).
- Control structures (if, else, switch, loops).

Functions:

- Function declaration vs. expression.
- Parameters and arguments.
- Return statements.
- Arrow functions.

Arrays and Objects:

- Array creation and manipulation.
- Object creation and manipulation.
- Accessing properties.

DOM Manipulation:

- Introduction to the DOM.

- Accessing and modifying HTML elements.
- Event handling.

Asynchronous JavaScript:

- Callback functions.
- Promises.
- Async/await.

Error Handling:

- The try...catch statement.
- Throwing errors with throw.

ES6 Features:

- Let and const.
- Arrow functions.
- Template literals.
- Destructuring assignment.
- Spread and rest operators.
- Classes and inheritance.

Modules:

- Exporting and importing modules.
- Default and named exports.

Browser APIs:

- Local Storage.
- Fetch API for making HTTP requests.
- Geolocation API.

Regular Expressions:

- Creating and using regular expressions.
- Methods like test() and exec().

Strict Mode:

- Introduction to strict mode.
- Benefits and limitations.

Topic Explanations with Examples:

Introduction to JavaScript:

- Explanation: JavaScript is a high-level, interpreted programming language primarily used for client-side web development. It enables dynamic content and interactivity in web pages.

Basic Syntax:

- Explanation: JavaScript syntax includes variables, data types, operators, and control structures.

```
let x = 5;

if (x > 0) {

    console.log("Positive");

} else {

    console.log("Non-positive");

}
```

Functions:

- Explanation: Functions are blocks of reusable code. They can take parameters and return values.
- CODE :

```
function greet(name) {

   return "Hello, " + name + "!";

}

console.log(greet("John"));
```

Arrays and Objects:

- Explanation: Arrays and objects are data structures used to store collections of values.
- CODE :

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

let person = { name: "John", age: 30 };
```

DOM Manipulation:

- Explanation: The DOM represents the structure of HTML documents. JavaScript can manipulate it to change content, style, and behavior.
- CODE :

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

    document.getElementById("myDiv").innerHTML = "Button clicked!";

});
```

Asynchronous JavaScript:

- Explanation: Asynchronous operations allow non-blocking execution, commonly used for fetching data from servers or handling user interactions.
- Example using Promise:
- CODE :

```
function fetchData() {

    return new Promise((resolve, reject) => {

        setTimeout(() => {

            resolve("Data fetched successfully");

        }, 2000);

    });

}
```

```
fetchData().then(data => {
    console.log(data);
});
```

Error Handling:

- Explanation: Error handling in JavaScript involves catching and handling exceptions to prevent program crashes.
- CODE :

```
try {
 throw new Error("Something went wrong");
 } catch (error) {
  console.error(error.message);
 }
```

ES6 Features:

- Explanation: ES6 introduced new features like let and const for variable declaration, arrow functions, template literals, etc.
- Example using arrow function:
- CODE :

```
const square = (x) => x * x;
console.log(square(5)); // Output: 25
```

Modules:

- Explanation: ES6 modules allow splitting code into reusable modules.

CODE :

```
export function greet(name) {
        return "Hello, " + name + "!";
```

```
}
```

```
import { greet } from './module.js';

console.log(greet("Alice")); // Output: Hello, Alice!
```

Browser APIs:

- Explanation: Browser APIs provide additional functionality to JavaScript for interacting with the browser environment.
- Example using Local Storage:

CODE :

```
localStorage.setItem("username", "John");

console.log(localStorage.getItem("username")); // Output: John
```

Regular Expressions:

- Explanation: Regular expressions are patterns used to match character combinations in strings.

CODE :

```
const regex = /[0-9]+/;

console.log(regex.test("abc123")); // Output: true
```

Strict Mode:

- Explanation: Strict mode enforces stricter parsing and error handling, helping to write cleaner and more secure
- x = 3.14; // Throws an error in strict mode

Iterators and Generators:

- **Explanation**: Iterators are objects that provide a way to iterate over data collections. Generators are functions that enable easy iteration with custom-defined sequences.
- **Example using Iterators**:

```
let arr = [1, 2, 3];

let iterator = arr[Symbol.iterator]();

console.log(iterator.next()); // Output: { value: 1, done: false }

console.log(iterator.next()); // Output: { value: 2, done: false }

console.log(iterator.next()); // Output: { value: 3, done: false }

console.log(iterator.next()); // Output: { value: undefined, done: true }
```

Example using Generators:

```
function* generator() {
    yield 1;
    yield 2;
    yield 3;
}

let gen = generator();

console.log(gen.next()); // Output: { value: 1, done: false }

console.log(gen.next()); // Output: { value: 2, done: false }

console.log(gen.next()); // Output: { value: 3, done: false }

console.log(gen.next()); // Output: { value: undefined, done: true }
```

Higher-order Functions:

- **Explanation**: Higher-order functions are functions that can take other functions as arguments or return them as results.
- **Example**:

```
function higherOrderFunction(callback) {
```

```
    callback();
}

function callbackFunction() {
    console.log("Callback function called");
}

higherOrderFunction(callbackFunction); // Output: Callback function called
```

Promises:

- **Explanation**: Promises represent a value that may be available now, in the future, or never. They are used for handling asynchronous operations.
- **Example**:

```
let promise = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve("Promise resolved");
    }, 2000);
});

promise.then(result => {
    console.log(result); // Output: Promise resolved
});
```

Async/Await:

- **Explanation**: Async/await is a syntax for handling asynchronous operations in a more synchronous-like manner, introduced in ES8 (ES2017).
- **Example**:

```
async function fetchData() {
```

```
    let promise = new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve("Data fetched successfully");
        }, 2000);
    });
    let result = await promise;
    console.log(result); // Output: Data fetched successfully
}
fetchData();
```

Closures:

- **Explanation**: Closures are functions that have access to variables from their outer scope even after the outer function has finished executing.
- **Example**:

```
function outerFunction() {
    let outerVariable = "I'm outer";
    function innerFunction() {
        console.log(outerVariable);
    }
    return innerFunction;
}
let closure = outerFunction();
```

```
closure(); // Output: I'm outer
```

Event Loop:

- **Explanation**: The event loop is JavaScript's mechanism for handling asynchronous operations. It allows non-blocking I/O operations to be performed despite JavaScript being single-threaded.
- **Example**:

```
console.log("Start");

setTimeout(() => {
    console.log("Timeout");
}, 0);

Promise.resolve().then(() => {
    console.log("Promise");
});

console.log("End");
```

Output:

```
Start
End
Promise
Timeout
```

THANK YOU ....