Web Class

Widuranga Dilruksha
(Software Engineer)
(BSc IT & M (Hons) (University of Moratuwa))
(MSc in IT(Reading) (University of Colombo)

Date: 06 / 08 / 2023

Day 5

JavaScript

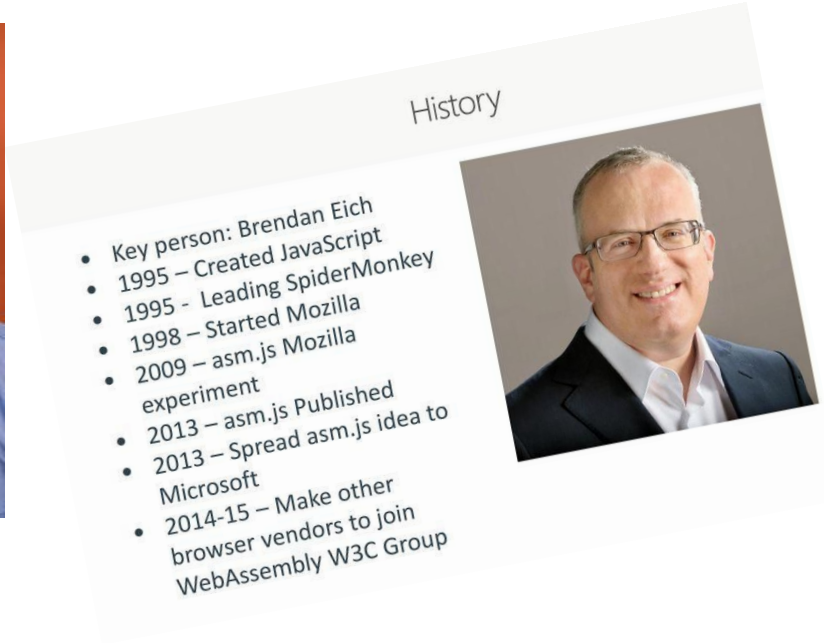
Aim : Learn JavaScript

File Folder Link:

https://drive.google.com/drive/folders/

Brendan Eich

- JavaScript is a powerful programming language that can add interactivity to a website. It was invented by Brendan Eich.

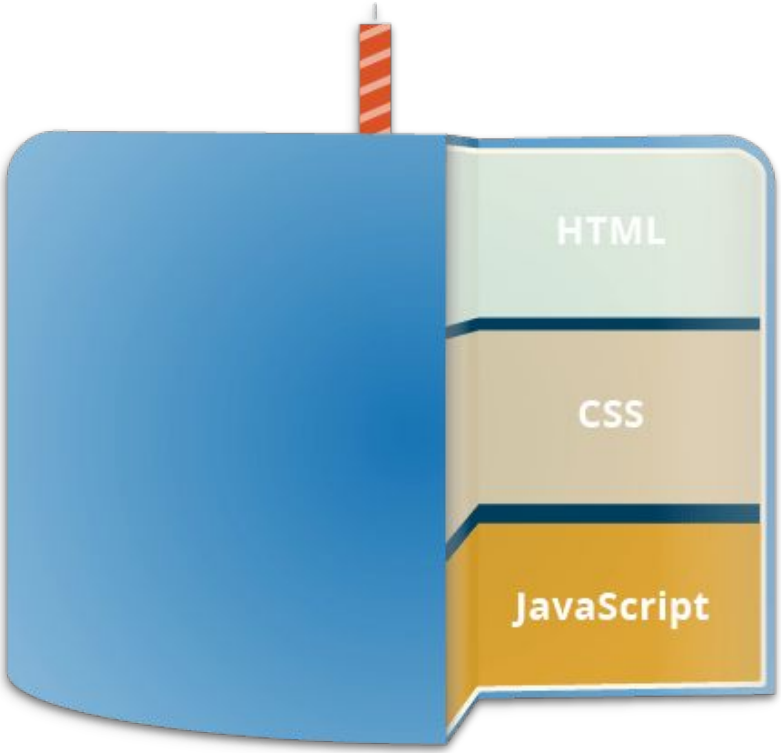


Why Study JavaScript?

- JavaScript is versatile and beginner-friendly. With more experience, you'll be able to create games, animated 2D and 3D graphics, comprehensive database-driven apps, and much more!
- JavaScript is one of the 3 languages all web developers must learn:

  1. **HTML** to define the content of web pages.
  2. **CSS** to specify the layout of web pages.
  3. **JavaScript** to program the behavior of web pages.

Web Cake



Basic syntax and variables

- JavaScript has a straightforward syntax that allows you to perform various operations. Here's an overview of some essential syntax elements:

- **Statements:** JavaScript code consists of statements, which are individual instructions that perform actions or calculations.

  Example:

  ```
  console.log("Hello, world!"); // This is a statement
  ```

Basic syntax and variables:

- **Comments:** Comments are used to add explanations or notes within your code. They are not executed by the computer.

Example:

```
// This is a single-line comment

/*
  This is a
  multi-line comment
*/
```

Basic syntax and variables:

- **Variables:** Variables are used to store data. You can declare a variable using let, const, or var (avoid using var in modern JavaScript).

  Example:

```
let age = 25; // A variable 'age' storing the value 25
const pi = 3.14159; // A constant variable 'pi' with a fixed value
```

Basic syntax and variables:

- Data Types: JavaScript has several data types, including:
  - **Numbers**: Used for numeric values, including integers and floating-point numbers.
  - **Strings**: Used for text and characters, enclosed in single or double quotes.
  - **Boolean**: Representing true or false values.
  - **Arrays**: Ordered collections of elements enclosed in square brackets.
  - **Objects**: Collections of key-value pairs enclosed in curly braces.

Basic syntax and variables:

- **Numbers:**

```
let num1 = 10; // Integer
let num2 = 3.14; // Floating-point number
```

- **Strings:**

```
let name = "John Doe"; // String
let message = 'Hello, world!'; // Strings can be enclosed in single or double quotes
```

Basic syntax and variables:

- **Boolean:**

```
let isLogged = true; // Boolean with value 'true'
let hasPermission = false; // Boolean with value 'false'
```

- **Arrays:**

```
let fruits = ['apple', 'orange', 'banana']; // An array of strings
let numbers = [1, 2, 3, 4]; // An array of numbers
```

Basic syntax and variables:

- **Objects:**

```
let person = {
  name: 'Alice',
  age: 30,
  isEmployed: true
};
```

Basic syntax and variables:

- Type Conversions:
  - JavaScript allows converting data from one type to another using various functions or methods.
- For example:
  - parseInt()
  - ParseFloat()
  - Number()

Basic syntax and variables:

```
let numString = "10"; // A string containing a number
let num = parseInt(numString); // Convert the string to an integer using parseInt()
console.log(num); // Output: 10

let numFloatString = "3.14"; // A string containing a floating-point number
let numFloat = parseFloat(numFloatString); // Convert the string to a floating-point number using parseFloat()
console.log(numFloat); // Output: 3.14

let isRaining = true;
let numFromBoolean = Number(isRaining); // Convert the boolean to a number (1 for true, 0 for false)
console.log(numFromBoolean); // Output: 1
```

Conditional statements

- Conditional statements are used to make decisions in your code based on certain conditions. In JavaScript, you can use the ‘if’, ‘else if’, and ‘else’ statements to create conditional blocks.
- Here's the syntax:

```
if (condition1) {
  // Code block executed if condition1 is true
} else if (condition2) {
  // Code block executed if condition1 is false and condition2 is true
} else {
  // Code block executed if both condition1 and condition2 are false
}
```

Conditional statements

- Simple if statement:

```
let age = 18;

if (age >= 18) {
  console.log("You are an adult.");
} else {
  console.log("You are a minor.");
}
```

Conditional statements

- Using else if:

```
let grade = 85;

if (grade >= 90) {
  console.log("A");
} else if (grade >= 80) {
  console.log("B");
} else if (grade >= 70) {
  console.log("C");
} else {
  console.log("F");
}
```

Loops

- Loops are used to repeat a block of code multiple times until a specific condition is met. JavaScript provides two main types of loops: the “for” loop and the “while” loop.
- **For Loop:**
    - The for loop is used when you know the number of iterations you want to execute. It has the following syntax:

```
for (initialization; condition; increment/decrement) {
  // Code block to be executed in each iteration
}
```

Loops

- **For Loop**:
  - Initialization: Setting up a counter or variable before the loop starts.
  - Condition: The loop continues as long as this condition is true.
  - Increment/Decrement: Updating the counter or variable after each iteration.
    - Example:

```
// Printing numbers from 1 to 5 using a for loop
for (let i = 1; i <= 5; i++) {
  console.log(i);
}
```

Loops

- **While Loop**:
  - The while loop is used when you don't know how many times the loop should execute but have a specific condition that must be true to continue the loop. It has the following syntax:

```
while (condition) {
  // Code block to be executed in each iteration
}
```

Loops

- Example:

```
// Printing numbers from 1 to 5 using a while loop
let count = 1;
while (count <= 5) {
  console.log(count);
  count++;
}
```

- **Note:**
  - Be cautious while using loops to avoid infinite loops. An infinite loop is a loop that never stops executing because the condition never becomes false. Make sure to have a clear exit condition or a way to break out of the loop.
  - Loops are powerful tools that allow you to efficiently perform repetitive tasks and handle data collections such as arrays. They are essential for iterating over elements in arrays or handling asynchronous tasks like fetching data from APIs.

Introduction to functions:

- In JavaScript, functions are blocks of code that perform a specific task or calculate a value. Functions help organize code into reusable units, making it easier to maintain and debug. They also promote the concept of modularity in programming.
- Here's the basic syntax of a function:

```
function functionName(parameters) {
  // Code block with the function's logic
  // It can perform tasks and/or return a value
}
```

Introduction to functions:

- Here's the basic syntax of a function:
  - functionName: This is the name of the function. You can choose any valid identifier as the function name.
  - parameters: These are optional. They represent the input values (arguments) that the function receives. Parameters are used to pass data into the function for processing.

Introduction to functions:

- Example of a simple function:

```
function greet(name) {
  console.log("Hello, " + name + "!");
}

greet("John"); // Output: Hello, John!
```

Introduction to functions:

- Functions can also have a return statement to send a value back to the caller:

```
function add(a, b) {
  return a + b;
}

let sum = add(3, 5);
console.log(sum); // Output: 8
```

Function Declarations vs. Function Expressions:

- Function Declarations:
  - A function declaration is a way to define a named function using the function keyword. Function declarations are hoisted in JavaScript, which means they can be called before they are defined in the code.

    - Example of a function declaration:

```
function square(x) {
  return x * x;
}
```

Function Declarations vs. Function Expressions:

- Function Expressions:
  - A function expression defines a function as part of a variable assignment. Instead of giving the function a name, it's assigned to a variable. Function expressions are not hoisted, so they can only be called after the assignment.
    - Example of a function expression:

```
const multiply = function(a, b) {
  return a * b;
};
```

- Function expressions are often used to create anonymous functions, which are functions without a specific name. They can be passed as arguments to other functions or used in various functional programming techniques.

Higher-order function

- Example of using an anonymous function in a higher-order function (map):

```
const numbers = [1, 2, 3, 4];
const squaredNumbers = numbers.map(function(num) {
  return num * num;
});

console.log(squaredNumbers); // Output: [1, 4, 9, 16]
```

Arrow Functions (ES6):

- In addition to function declarations and function expressions, ES6 introduced arrow functions, which provide a shorter syntax for writing functions.
    - Example of an arrow function:

```
const cube = (x) => {
  return x * x * x;
};
```

- When the function has only one expression, the curly braces and return keyword can be omitted:

```
const cube = x => x * x * x;
```

Arrow Functions (ES6):

- What About this?
  - The handling of this is also different in arrow functions compared to regular functions.
  - In short, with arrow functions there are no binding of this.
  - In regular functions the this keyword represented the object that called the function, which could be the window, the document, a button or whatever.
  - With arrow functions the this keyword always represents the object that defined the arrow function.