1st Video about -> **History of JavaScript?**

JavaScript was created in 1995 by Brendan Eich while working at Netscape Communications Corporation. Initially named Mocha, the language was designed to add interactive elements to websites, something that was lacking in the early web. It was later renamed to LiveScript and eventually to JavaScript, aligning with the popularity of Java at the time, despite the two languages having little in common beyond their name.

JavaScript was first implemented in Netscape Navigator, the leading web browser of the time. Microsoft quickly recognized the potential of this new scripting language and developed its own version called JScript, which was released with Internet Explorer 3.0. This move by Microsoft created a rivalry, leading to compatibility issues as each browser had its own implementation of the language.

To address this fragmentation, standardization efforts began, leading to the formation of ECMAScript, a standardized version of JavaScript, which was released in 1997. Over time, both Netscape's JavaScript and Microsoft's JScript adhered to this standard, which allowed JavaScript to evolve into a robust, widely-supported language that powers interactive content on the web.

JavaScript has since become one of the most important programming languages in the world, enabling dynamic content, interactivity, and complex web applications across all modern web browsers.

2nd video about -> **Data Types in JavaScript - 1**

Data plays a vital role in software development, enabling storage, processing, and the extraction of meaningful insights.

- **Value and Type:** In JavaScript, each value has a specific type, which is not tied to the variable itself but to the value it holds.
- **Primitive Data Types:** JavaScript includes several primitive data types such as number, string, boolean, null, undefined, and symbol.
- **Object Data Type:** The object is the sole non-primitive data type in JavaScript, capable of holding multiple values and complex structures.
- **Numbers in JavaScript:** The largest integer that can be safely represented is 9007199254740991. For even larger numbers, JavaScript provides the BigInt type.

```
let bigNumber = 9007199254740992n; // Using BigInt for large numbers
console.log(bigNumber); // 9007199254740992n
```

- **Floating-Point Representation:** JavaScript supports numbers in exponential notation, and underscores can be added to improve readability.

```
let largeFloat = 1.23e5; // Exponential notation
let readableNumber = 1_000_000; // Underscore for readability
console.log(largeFloat); // 123000
console.log(readableNumber); // 1000000
```

- **Infinity and NaN:** JavaScript recognizes special values such as Infinity, -Infinity, and NaN (Not a Number).

```
console.log(1 / 0); // Infinity
console.log(0 / 0); // NaN
```

- **BigInt:** For handling exceptionally large integers, append an n to the number to utilize the BigInt type.

```
let veryLargeNumber = 12345678901234567890123456789012345678901234567890n;
console.log(veryLargeNumber);//12345678901234567890123456789012345678901234567890n
```

3rd video about -> **Data Types in JavaScript-2**

String Literals in JavaScript:

Strings in JavaScript are enclosed in quotes, and special characters can be represented using escape sequences.

```
//Double Quotes:
let greeting = "Good morning!";
console.log(greeting); // Output: Good morning!
// Single Quotes:
let phrase = 'Don\'t worry, be happy!';
console.log(phrase); // Output: Don't worry, be happy!
// Escape Characters:
let story = "He whispered, \"Keep going.\" \nThe next line starts here.";
console.log(story);
// Output:
// He whispered, "Keep going."
// The next line starts here.
```

● Combining Strings: Strings can be concatenated using the + operator.

```
let firstName = "John";
let lastName = "Doe";
let fullName = firstName + " " + lastName;
console.log(fullName); // Output: John Doe
```

Example : Adding Numbers as Strings

```
let height = "180";
let info = "Height: " + height + " cm";
console.log(info); // Output: Height: 180 cm
```

Example : Concatenating Strings with Other Data Types

```
let fruit = "banana";
let count = 3;
let sentence = "I bought " + count + " " + fruit + "s.";
console.log(sentence); // Output: I bought 3 bananas.
```

Example : Boolean Expression Evaluation

```
let x = 15;
let y = 20;
let isLess = x < y;
console.log(isLess); // Output: true
```

Null and Undefined in JavaScript:

- Null signifies the intentional absence of any value, while undefined indicates that a variable has been declared but hasn't been assigned any value yet.

```
let product;
console.log(product); // Output: undefined
```

```
let customer = null;
console.log(customer); // Output: null
```

NaN: Represents a value that is not a valid number.

```
let Result = 100 / "apple";
console.log(Result); // Output: NaN
```

4th video about -> **Type Conversion and Coercion**

Type Conversion: This process involves transforming data from one type to another, such as changing a number into a string or vice versa. This can be done explicitly using functions like String(), Number() etc.

Changing number to string

```
let a = 98;
let b = String(a);
console.log(b); // Output: "98"
console.log(typeof b); // Output: string
```

Change String to number

```
let age = "30";
let ageToNumber = Number(age);
console.log(ageToNumber); // Output: 30
console.log(typeof ageToNumber); // Output: number
```

Type Coercion: This happens when JavaScript automatically converts a value from one data type to another to complete an operation.

```
let number = 5;
let text = "The number is " + number;
console.log(text); // Output: "The number is 5"
console.log(typeof text); // Output: string
```

5th video is about -> Arithmetic Operators

Basic Arithmetic Operations: JavaScript allows you to perform fundamental arithmetic operations such as addition, subtraction, multiplication, and division.

```
// Addition
let a = 10;
let b = 5;
let sum = a + b;
console.log(sum); // Output: 15
// Subsctraction
let c = 20;
let d = 8;
let difference = c - d;
console.log(difference); // Output: 12
// Multiplication
let e = 4;
let f = 7;
let product = e * f;
console.log(product); // Output: 28
// Division
let g = 40;
let h = 8;
let quotient = g / h;
console.log(quotient); // Output: 5
```

Increment and Decrement: The ++ operator increases a number by 1, while the -- operator decreases it by 1.

```
let count = 5;
count++;
console.log(count); // Output: 6

count--;
console.log(count); // Output: 5
```

Exponentiation: To perform exponentiation, you can use the ** operator or the Math.pow() function.

```
let base = 3;
let exponent = 4;
let result = base ** exponent; // Using ** operator
console.log(result); // Output: 81

result = Math.pow(base, exponent); // Using Math.pow() function
console.log(result); // Output: 81
```

6th video is about -> Relational Operators

Comparison of Numbers: Relational operators are used to compare two numerical values, yielding a boolean result.

```
let num1 = 10;
let num2 = 15;
console.log(num1 < num2); // Output: true
```

Comparison of Strings: JavaScript compares strings based on their ASCII values in a lexicographical order.

```
let str1 = "apple";
let str2 = "banana";
console.log(str1 < str2); // Output: true
```

Comparison of Mixed Types: When comparing values of different types, JavaScript performs type coercion to attempt to convert them to a common type.

```
let value = 10;
let text = "10";
console.log(value == text); // Output: true
```

Strict Equality (===) vs. Loose Equality (==): The === operator checks for both value and type equality, while the == operator performs type coercion before comparison.

```
let num = 5;
let str = "5";
console.log(num === str); // Output: false
console.log(num == str);  // Output: true
```

7th video is about -> Logical Operators

Logical Operators: Logical operators are used to combine or invert boolean values, resulting in a boolean outcome.

Example 1: Logical AND (&&)

```
let isSunny = true;
let isWeekend = false;
console.log(isSunny && isWeekend); // Output: false
```

Example 2: Logical OR (||)

```
let isSunny = true;
let isWeekend = false;
console.log(isSunny || isWeekend); // Output: true
```

Example 3: Logical NOT (!)

```
let isSunny = true;
console.log(!isSunny); // Output: false
```

8th video is about -> Ternary operations

Ternary Operators: The ternary operator (? :) is a shorthand for an if-else statement that evaluates a condition and returns one of two values based on the result.

Ternary Operator Syntax:

condition ? valueIfTrue : valueIfFalse;

```
let age = 18;
let canVote = age >= 18 ? "Yes" : "No";
console.log(canVote); // Output: "Yes"
```

Example 2 –

```
let number = 10;
let result = number % 2 === 0 ? "Even" : "Odd";
console.log(result); // Output: "Even"
```

9th video is about -> Template literals

Template Literals: Template literals are a way to embed expressions and variables within strings, using backticks (``) and ${} syntax.

```
let name = "Alice";
let greeting = `Hello, ${name}!`;
console.log(greeting); // Output: "Hello, Alice!"
```

Example 2: Expression Evaluation

```
let a = 5;
let b = 10;
let result = `The sum of ${a} and ${b} is ${a + b}.`;
console.log(result); // Output: "The sum of 5 and 10 is 15."
```