FSD ASSIGNMENT - 3

1. History of JavaScript

JavaScript is a high-level, dynamic programming language that I primarily use to create interactive and responsive elements on websites. It runs in web browsers, allowing me to control multimedia, animate images, and handle user inputs. One of the key strengths of JavaScript is its versatility, enabling its use on both the client-side and server-side.

For example, I use JavaScript to:

1. **Validate user input in real-time on interactive forms before submission.**
2. **Create animations and transitions to make web pages more engaging.**
3. **Display real-time data updates like live sports scores or chat messages without requiring page reloads.**

2. Data Types in JavaScript - Part 1

Primitive Data Types are the most basic types of data in JavaScript. They include:

- **String: This represents textual data, such as "Hello, World!".**
- **Number: This represents numerical data, including both integers and floating-point numbers, like 42 or 3.14.**
- **Boolean: This represents logical values, either true or false.**
- **Undefined: This represents a variable that has been declared but has not been assigned a value.**
- **Null: This represents an intentional absence of any object value.**
- **Symbol: This is a unique and immutable primitive data type, often used as keys for object properties to avoid name collisions.**

Examples:-

```
console.log(typeof "Hello, World!"); // string
console.log(typeof true); // boolean
console.log(typeof [1, 2, 3]); // object
console.log(typeof function() {}); // function
console.log(typeof { name: "Alice", age: 25 }); // object
```

3. Data Types in JavaScript - 2
String: Text data enclosed in quotes

```
let string1 = "Hello, world,";
let string2 = "This is a new line.\nHere is the next line.";
 let result = string1 + string2; console.log(result);

 string1 = "Name \t Rakesh"; string2 = "  Age \t 20";
result = string1 + string2; console.log(result); // Name    Rakesh Age    20

let firstName = "Rakesh";
 let lastName = "Kurra";
let fullName = firstName + " " + lastName; console.log(fullName); // "Rakesh Kurra"
```

- **Boolean: A value that is either true or false.**

Examples:

```
 console.log(3>2)
 console.log(9<10)
 console.log(45>90)
```

Null: A special value that means "no value."
Example: let object=null console.log(typeof object)

Undefined: A value that means a variable has been declared but not yet assigned a value.
Example: let object console.log(object)

4. Type Conversion and Coercion

Type Conversion Definition: Type conversion, also known as type casting, is the explicit conversion of a value from one type to another. It is performed by the programmer using functions or methods.

Examples

```
let str = "987";
let num = Number(str);
 console.log(num); // Output: 987
```

```
let num = 789;
let str = String(num);
console.log(str); // Output: "789"
```

```
let num = 0;
 let bool = Boolean(num);
console.log(bool); // Output: false
```

5. Arithmetic Operators

Arithmetic operators are used to perform mathematical operations such as addition, subtraction, multiplication, division, and modulus. They work with numerical values to produce results based on the operation performed.

```
// Exponentiation
 let power = 3 ** 4;
 console.log(power); // Output: 81
```

```
// Pre-increment
let x = 7; console.log(++x);
 // Output: 8 (value of `x` is incremented first, then used)
```

```
 // Post-increment
 x = 7;
 console.log(x++);
// Output: 7 (value of `x` is used first, then incremented)
console.log(x); // Output: 8 (value of `x` after increment)
```

6. Relational Operators

Relational operators are symbols I use to compare two values or expressions. They help determine the relationship between these values, such as whether one is greater than, less than, equal to, or not equal to the other. The result of using a relational operator is always a boolean value: `true` if the specified condition is met, and `false` if it is not.

Examples:

Here are some examples of relational operators in JavaScript:

1. `5 > 3` **// true - Checks if 5 is greater than 3.**
2. `7 == 7` **// true - Checks if 7 is equal to 7.**
3. `4 <= 2` **// false - Checks if 4 is less than or equal to 2.**

7 . Logical Operators

Logical operators are crucial for performing logical operations on boolean values (`true` or `false`). They allow me to combine multiple conditions, which is essential for controlling the flow of a program, especially in decision-making structures like `if` statements. The three main logical operators are:

1. **&& (Logical AND): Returns `true` if both operands are true. If either operand is false, it returns `false`.**
2. **|| (Logical OR): Returns `true` if at least one of the operands is true. If both are false, it returns `false`.**
3. **! (Logical NOT): Inverts the boolean value of its operand. If the operand is true, it returns `false`, and vice versa.**

Examples:- AND

```
let isAdult = true;

 let hasPermission = true;

console.log(isAdult && hasPermission);
```

2. OR

```
let isWeekend = true;

 let isHoliday = false;

 console.log(isWeekend || isHoliday); // true - At least one condition is true
```

3. NOT

```
let isRaining = false;

 console.log(!isRaining); // true - Inverts the boolean value.
```

8. Ternary Operators

The ternary operator is a compact way to handle conditional expressions in programming. It allows me to evaluate a condition and return one of two possible values based on whether the condition is true or false. The ternary operator is named "ternary" because it involves three parts: the condition, the result if the condition is true, and the result if the condition is false.

Examples

```
 let age = 18;

 let canVote = age >= 18 ? "Yes, can vote" : "No, cannot vote";

 console.log(canVote); // Output: "Yes, can vote"
```

```
let temperature = 30;

let weather = temperature > 25 ? "Hot" : "Cool";

console.log(weather); // Output: "Hot"
```

```
let score = 85;

 let grade = score >= 90 ? "A" : score >= 80 ? "B" : "C";

console.log(grade); // Output: "B"
```

9. Template Literals

Template literals are a powerful feature in JavaScript that make working with strings more versatile. Unlike traditional quotes, template literals are enclosed in backticks (`). They support embedding expressions, multi-line strings, and string interpolation using `${expression}`.

Examples

```
let name = "Rakesh";

 let age = 30;

let message = `My name is ${name} and I am ${age} years old.`;

console.log(message); // Output: "My name is Rakesh and I am 30 years old."
```

```
let a = 8; let b = 12;
 let result = `The sum of ${a} and ${b} is ${a + b}. This result is calculated using template literals.`;
 console.log(result); // Output: // "The sum of 8 and 12 is 20. // This result is calculated using template literals."
```

```
let fruit = "banana"; let quantity = 6;
 let message = `I have ${quantity} ${fruit}s in my basket.`;
 console.log(message); // Output: "I have 6 bananas in my basket."
```