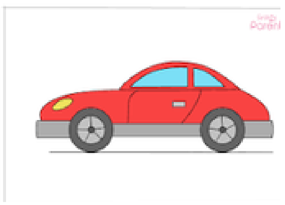
Javascript Fundamentals

- Hello World
  - console.log(“Hello World”);
  - alert(“Hello World”);
- Intro to JS
  - What is JS?
    - JavaScript is high level language that helps us to provide functionality to a website.
  - Role of JS?
    - Building web Application.
    - To Interact the content of the web page.
    - To manipulate the content of the web page.
    - Adding dynamic effects to a web page.
    - e.g a car.

HTML

It is the strucure



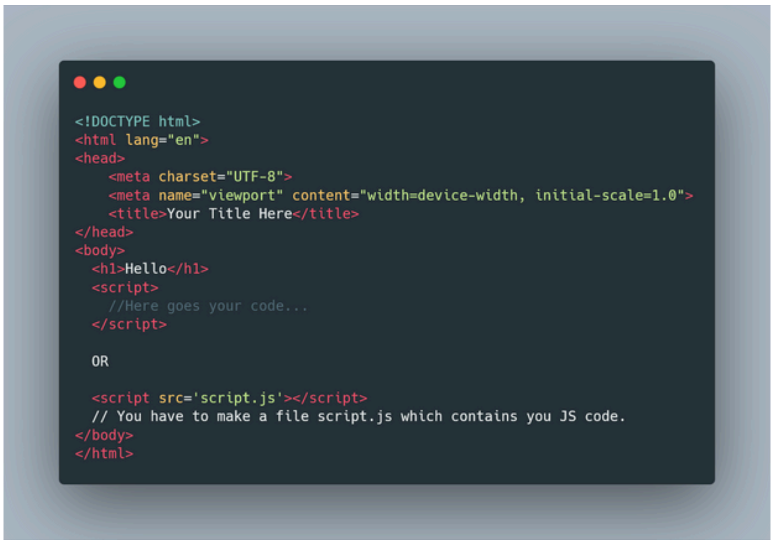
CSS

It provides the color and styling

JavaScript

It provides the fuctionality

- How to Link JS file/Run JS in a browser



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Your Title Here</title>
</head>
<body>
  <h1>Hello</h1>
  <script>
    //Here goes your code...
  </script>

  OR

  <script src='script.js'></script>
  // You have to make a file script.js which contains you JS code.
</body>
</html>
```

- Variable
  - Variable is container that help you to store your values
  - Conventions -
    - Your variable name should be self explanatory.
    - Variable should be named using camel case.
  - Rules -
    - Your variable only consist of _ , alphabets, numbers and $.
    - Your variable should not start with number like, const 1name = ‘rahul’

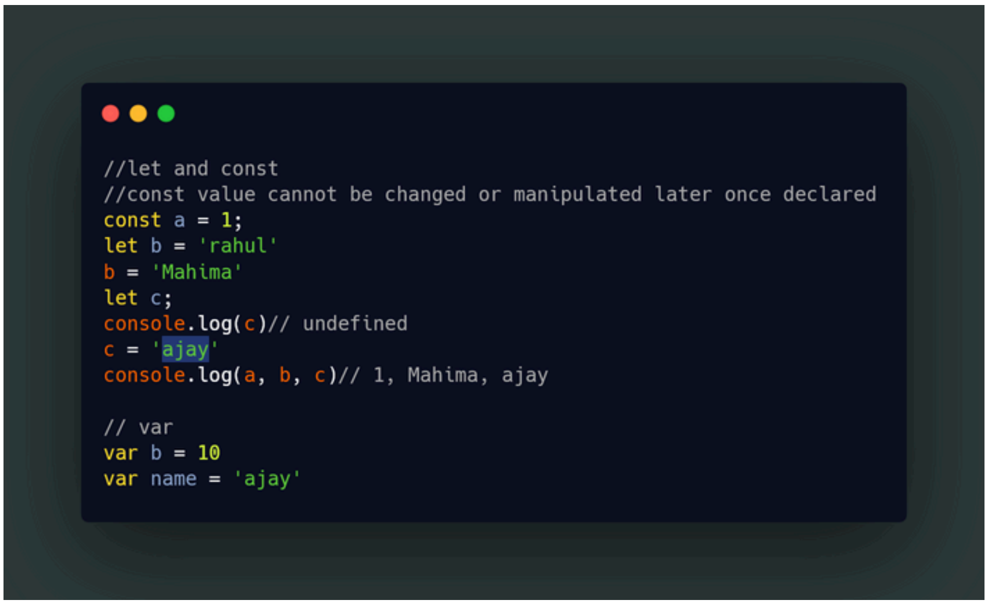
- Data Types
  - Primitives
    - Number: let a = 1;
    - String: let name = ‘dhrubBharwa’;
    - Boolean: let me = true;
    - Undefined: let b;
    - Null: It is a empty value. If you access a element from HTML that doesn’t exist it will return null. It is also a falsy value.
    - BigInt(Es2020): large numbers .
    - Symbol(Es2015):

Note: JavaScript is a dynamic type language, we don’t need to manually set the data type of a variable. Data type is automatically determined base on the value while run time.

typeOf null = object(Bug of JS)
typeOf undefined = undefined

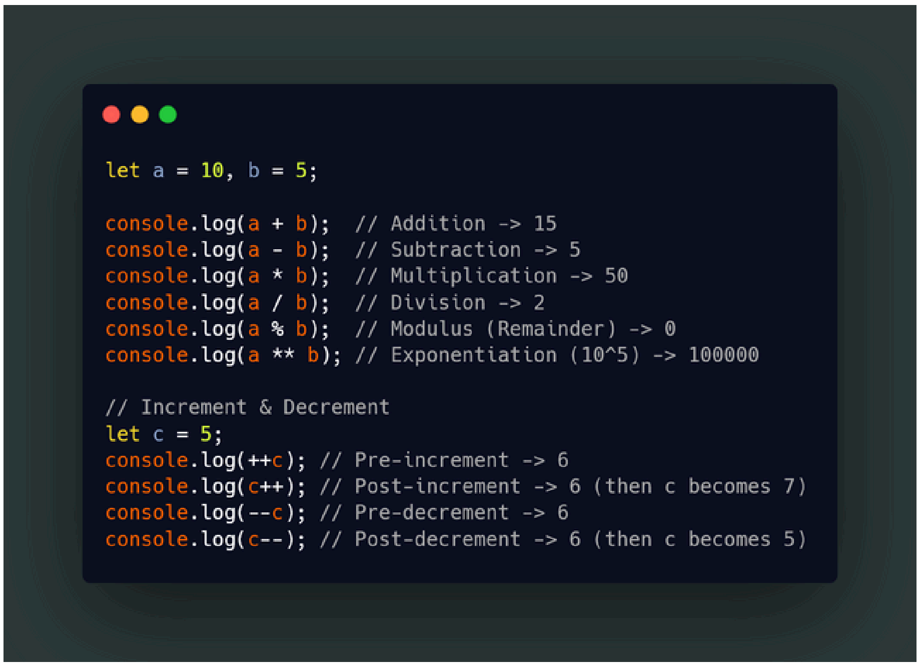
- Non-Primitive
  - Object{Explained Later in Detail}

- Let vs const vs var
  - let and const (Intoduce in ES6v).
  - const variable cannot be changed or manipulated once declared.
  - let variable can be changed later to any data type.
  - let and const is a block scope based. For e.g {it can only access here}
  - var is a function scope based. It can access anywhere in a function.



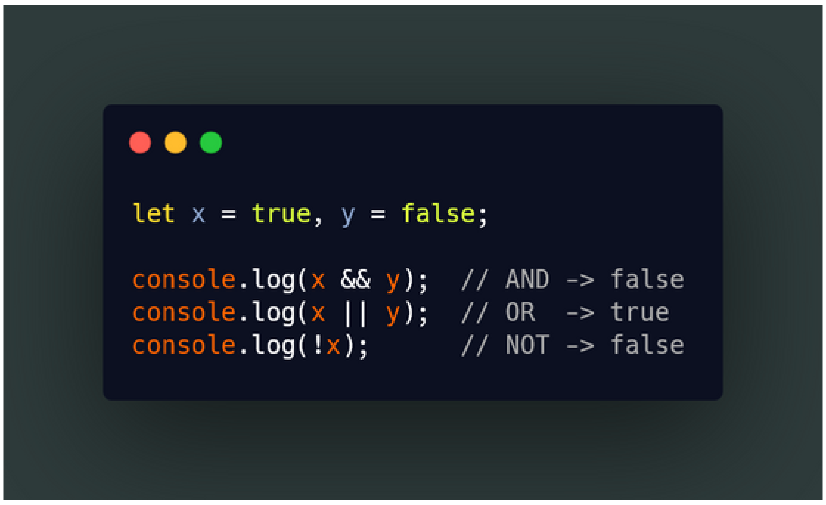
```
//let and const
//const value cannot be changed or manipulated later once declared
const a = 1;
let b = 'rahul'
b = 'Mahima'
let c;
console.log(c)// undefined
c = 'ajay'
console.log(a, b, c)// 1, Mahima, ajay

// var
var b = 10
var name = 'ajay'
```

- **Operators:**
  - **Mathematical Operator:**



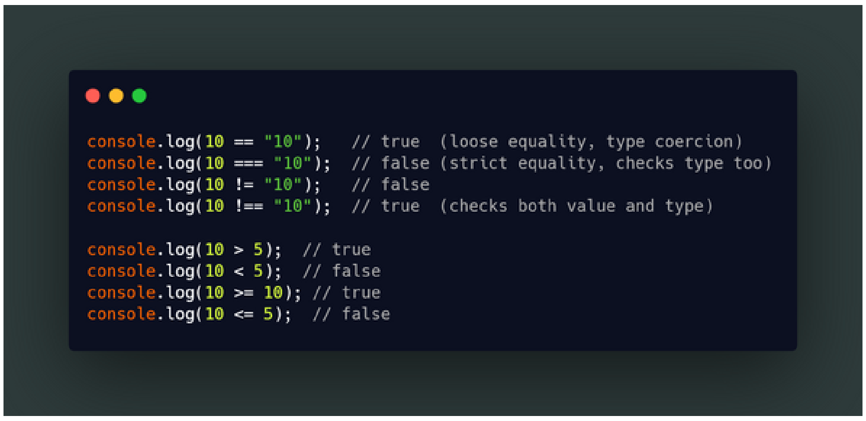
```
let a = 10, b = 5;

console.log(a + b);  // Addition -> 15
console.log(a - b);  // Subtraction -> 5
console.log(a * b);  // Multiplication -> 50
console.log(a / b);  // Division -> 2
console.log(a % b);  // Modulus (Remainder) -> 0
console.log(a ** b); // Exponentiation (10^5) -> 100000

// Increment & Decrement
let c = 5;
console.log(++c); // Pre-increment -> 6
console.log(c++); // Post-increment -> 6 (then c becomes 7)
console.log(--c); // Pre-decrement -> 6
console.log(c--); // Post-decrement -> 6 (then c becomes 5)
```

  - **Logical Operator**



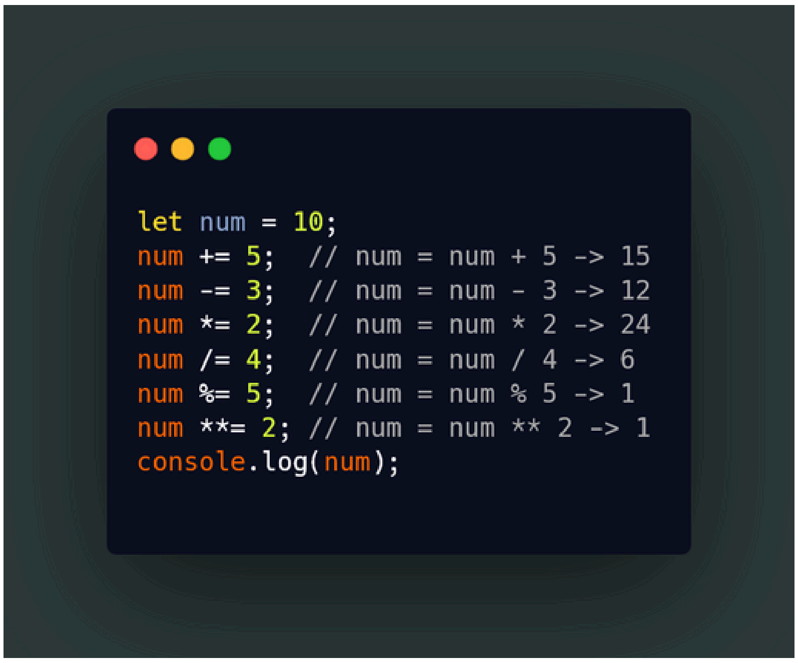
```
let x = true, y = false;

console.log(x && y);  // AND -> false
console.log(x || y);  // OR  -> true
console.log(!x);      // NOT -> false
```

  - **Comparison Operator**



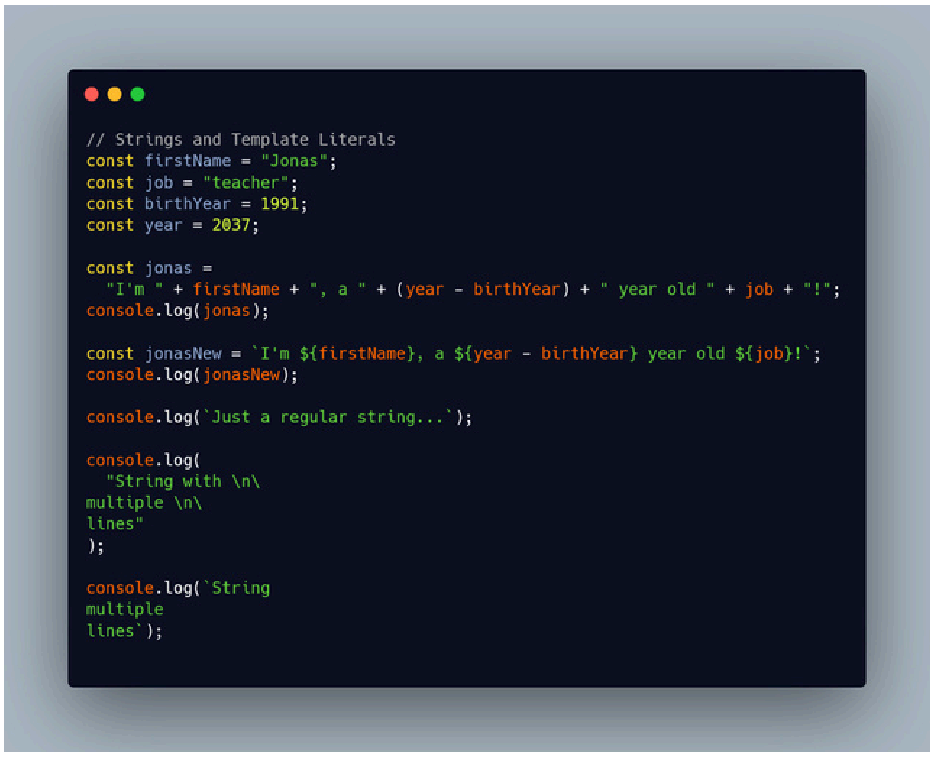
```
console.log(10 == "10");   // true  (loose equality, type coercion)
console.log(10 === "10");  // false (strict equality, checks type too)
console.log(10 != "10");   // false
console.log(10 !== "10");  // true  (checks both value and type)

console.log(10 > 5);  // true
console.log(10 < 5);  // false
console.log(10 >= 10); // true
console.log(10 <= 5);  // false
```

- **Assignment Operator**



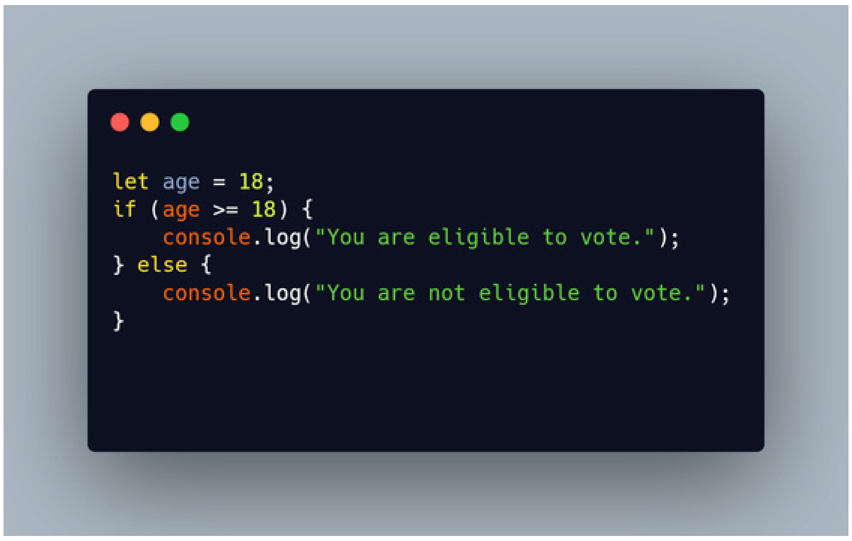
```javascript
let num = 10;
num += 5;  // num = num + 5 -> 15
num -= 3;  // num = num - 3 -> 12
num *= 2;  // num = num * 2 -> 24
num /= 4;  // num = num / 4 -> 6
num %= 5;  // num = num % 5 -> 1
num **= 2; // num = num ** 2 -> 1
console.log(num);
```

- **String & Template Literals**
  - **Template literals (also called template strings) are a way to work with strings in JavaScript using backticks (`) instead of quotes ("" or ''). They allow multi-line strings, interpolation, and embedded expressions.**



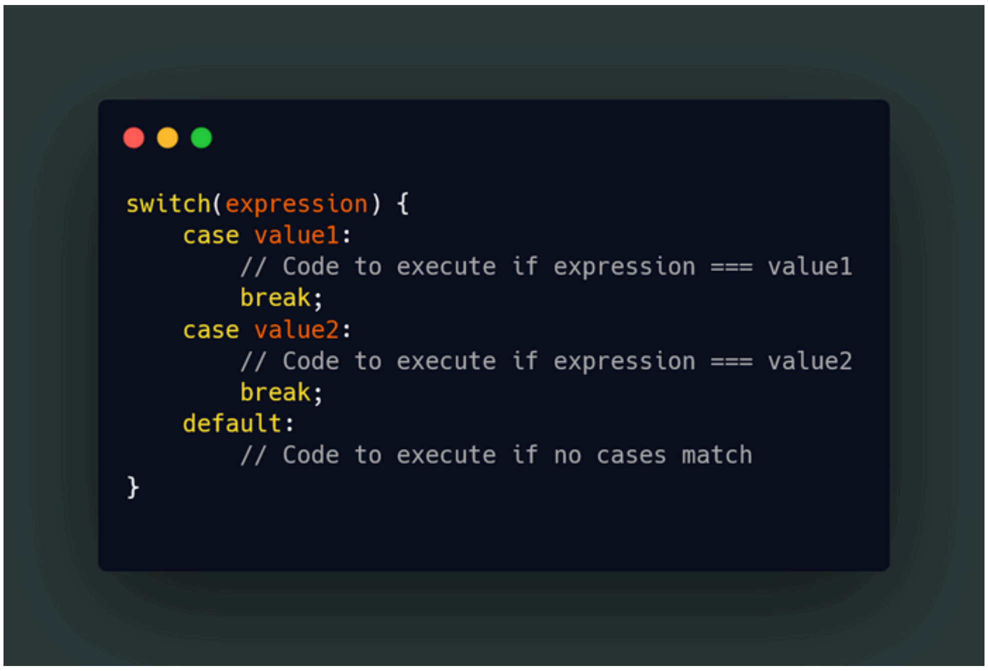
```javascript
// Strings and Template Literals
const firstName = "Jonas";
const job = "teacher";
const birthYear = 1991;
const year = 2037;

const jonas =
  "I'm " + firstName + ", a " + (year - birthYear) + " year old " + job + "!";
console.log(jonas);

const jonasNew = `I'm ${firstName}, a ${year - birthYear} year old ${job}!`;
console.log(jonasNew);

console.log(`Just a regular string...`);

console.log(
  "String with \n\
multiple \n\
lines"
);

console.log(`String
multiple
lines`);
```

- Decision Making(If - else)
  - A decision statement (if-else) in JavaScript is used to execute different blocks of code based on a given condition



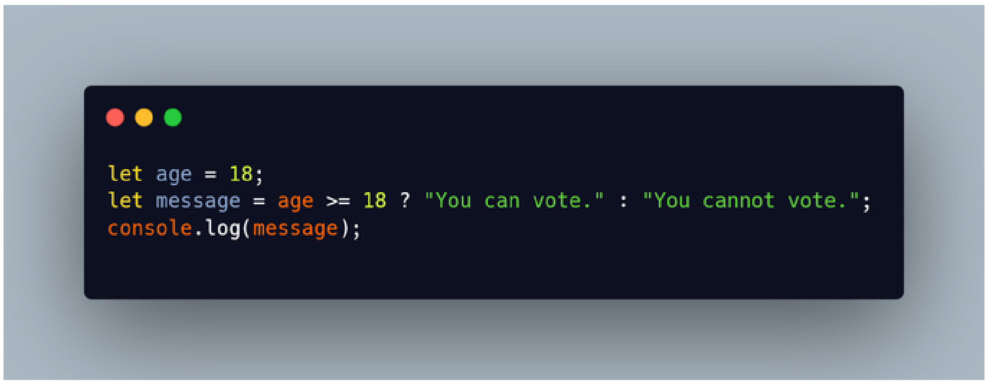
```
let age = 18;
if (age >= 18) {
    console.log("You are eligible to vote.");
} else {
    console.log("You are not eligible to vote.");
}
```

- Switch Statement
  - The switch statement is used for multi-way branching, meaning it allows you to execute different blocks of code based on the value of a variable.



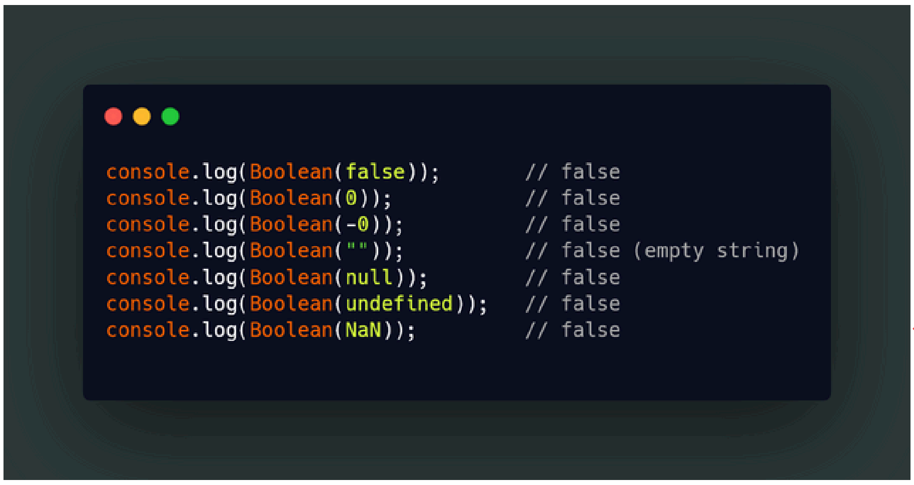
```
switch(expression) {
    case value1:
        // Code to execute if expression === value1
        break;
    case value2:
        // Code to execute if expression === value2
        break;
    default:
        // Code to execute if no cases match
}
```

- Ternary Operator
  - The ternary operator (? :) in JavaScript is a shorthand for if-else, used to make conditional decisions in a single line.



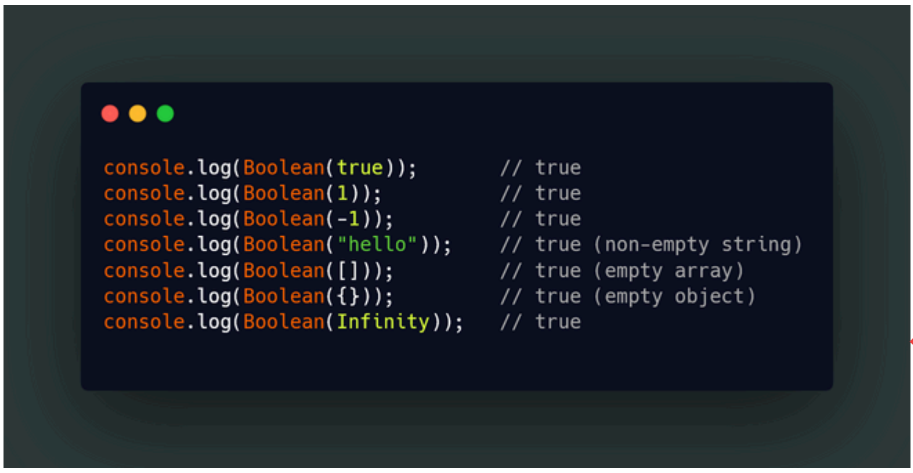
```
let age = 18;
let message = age >= 18 ? "You can vote." : "You cannot vote.";
console.log(message);
```

- Truthy and Falsy Value



```
console.log(Boolean(false));        // false
console.log(Boolean(0));            // false
console.log(Boolean(-0));           // false
console.log(Boolean(""));           // false (empty string)
console.log(Boolean(null));         // false
console.log(Boolean(undefined));    // false
console.log(Boolean(NaN));          // false
```

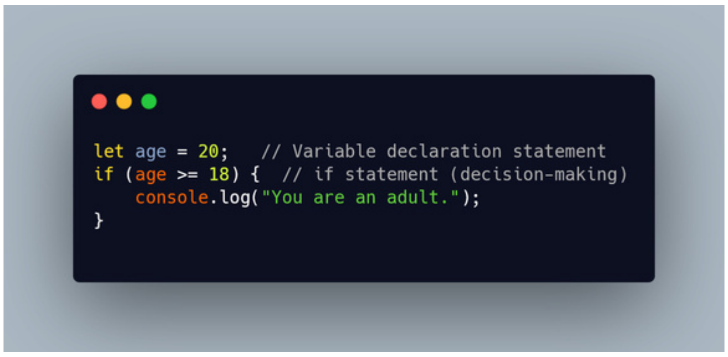
Falsy value



```
console.log(Boolean(true));        // true
console.log(Boolean(1));           // true
console.log(Boolean(-1));          // true
console.log(Boolean("hello"));     // true (non-empty string)
console.log(Boolean([]));          // true (empty array)
console.log(Boolean({}));          // true (empty object)
console.log(Boolean(Infinity));    // true
```

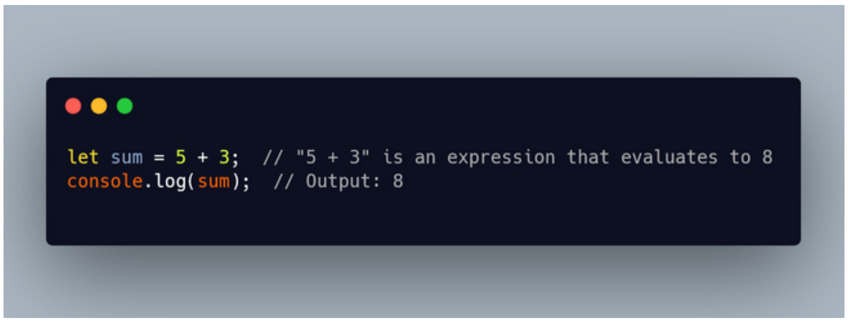
Truthy value

- Statement
  - A statement in JavaScript performs an action but does not necessarily return a value. Statements control the flow of the program.

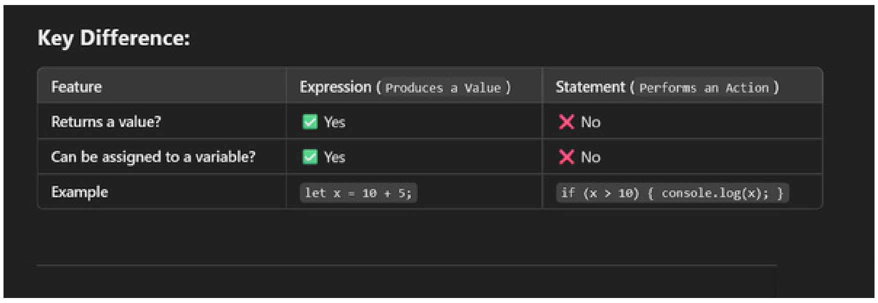


```
let age = 20;   // Variable declaration statement
if (age >= 18) {  // if statement (decision-making)
    console.log("You are an adult.");
}
```

- Expression
  - An expression in JavaScript produces a value and can be assigned to a variable. It can be a single value, a mathematical operation, or a function that returns something.



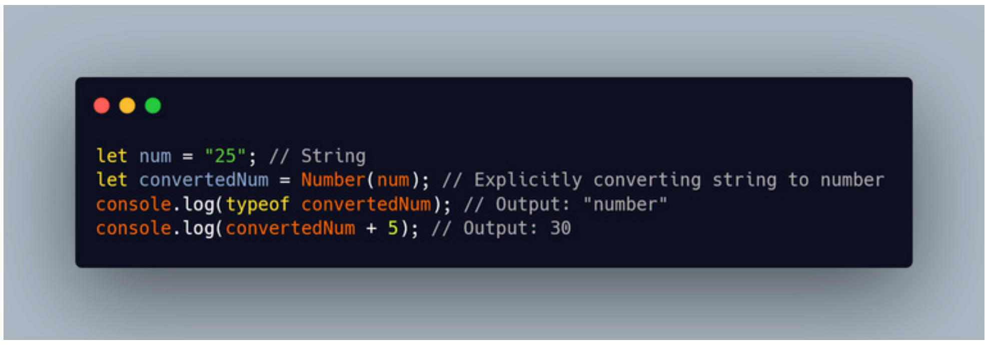
```
let sum = 5 + 3;  // "5 + 3" is an expression that evaluates to 8
console.log(sum);  // Output: 8
```



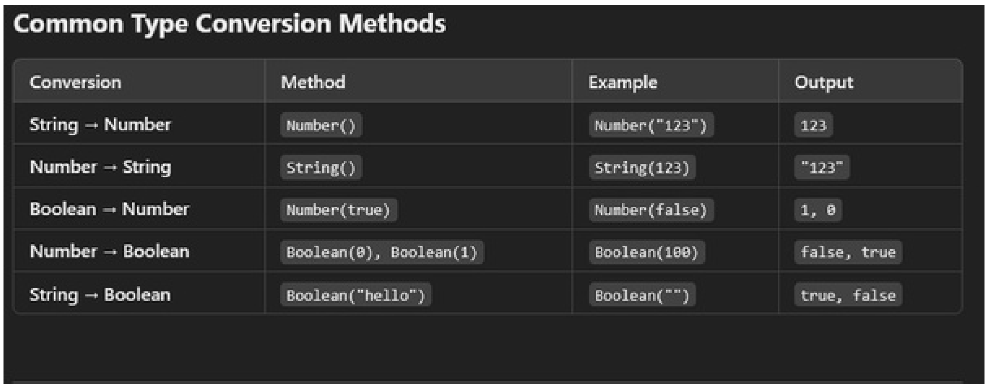
Key Difference:

Feature	Expression (Produces a Value)	Statement (Performs an Action)
Returns a value?	✅ Yes	❌ No
Can be assigned to a variable?	✅ Yes	❌ No
Example	`let x = 10 + 5;`	`if (x > 10) { console.log(x); }`

- **Type Conversion**
  - **Type conversion happens manually when we explicitly convert a value from one type to another using JavaScript functions.**



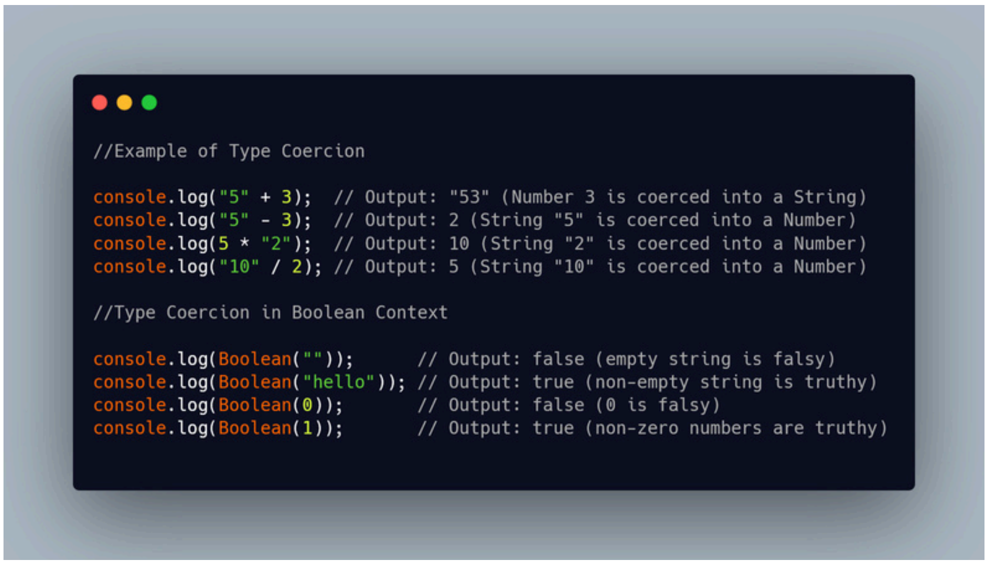
```
let num = "25"; // String
let convertedNum = Number(num); // Explicitly converting string to number
console.log(typeof convertedNum); // Output: "number"
console.log(convertedNum + 5); // Output: 30
```



Common Type Conversion Methods

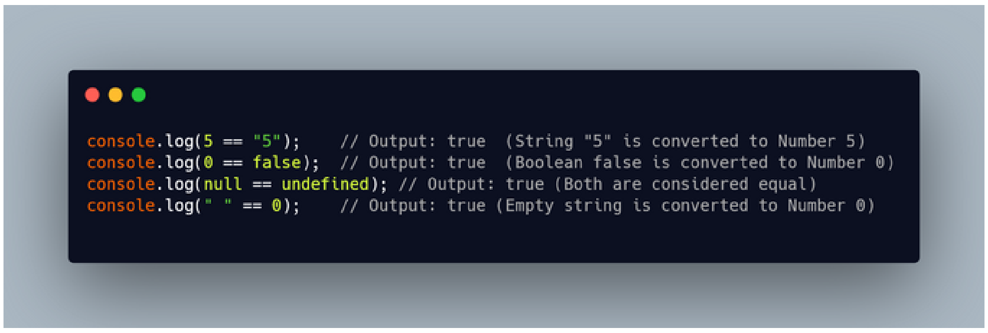
Conversion	Method	Example	Output
String → Number	`Number()`	`Number("123")`	`123`
Number → String	`String()`	`String(123)`	`"123"`
Boolean → Number	`Number(true)`	`Number(false)`	`1, 0`
Number → Boolean	`Boolean(0), Boolean(1)`	`Boolean(100)`	`false, true`
String → Boolean	`Boolean("hello")`	`Boolean("")`	`true, false`

- **Type Coercion**
  - **Type coercion happens automatically when JavaScript converts one data type to another behind the scenes.**



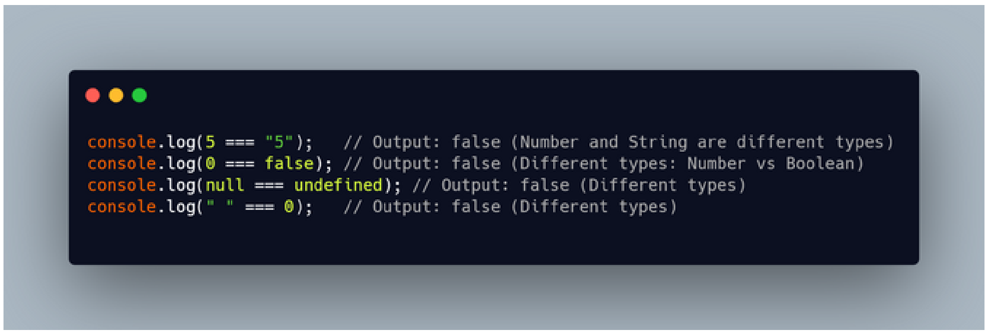
```
//Example of Type Coercion

console.log("5" + 3);  // Output: "53" (Number 3 is coerced into a String)
console.log("5" - 3);  // Output: 2 (String "5" is coerced into a Number)
console.log(5 * "2");  // Output: 10 (String "2" is coerced into a Number)
console.log("10" / 2); // Output: 5 (String "10" is coerced into a Number)

//Type Coercion in Boolean Context

console.log(Boolean(""));      // Output: false (empty string is falsy)
console.log(Boolean("hello")); // Output: true (non-empty string is truthy)
console.log(Boolean(0));       // Output: false (0 is falsy)
console.log(Boolean(1));       // Output: true (non-zero numbers are truthy)
```

- **== (Loose Equality)**
  - Compares only the values, ignoring the data type.
  - If the values are of different types, JavaScript performs type coercion before comparison.
  - Also known as "loose equality**".**



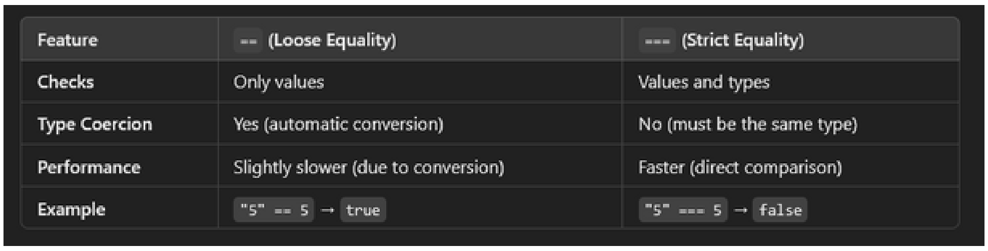
```
console.log(5 == "5");    // Output: true  (String "5" is converted to Number 5)
console.log(0 == false);  // Output: true  (Boolean false is converted to Number 0)
console.log(null == undefined); // Output: true (Both are considered equal)
console.log(" " == 0);    // Output: true (Empty string is converted to Number 0)
```

- **=== (Strict Equality)**
  - Compares both value and data type.
  - No type coercion happens.
  - If the values are not of the same type, it directly returns false.
  - Also known as "strict equality"



```
console.log(5 === "5");   // Output: false (Number and String are different types)
console.log(0 === false); // Output: false (Different types: Number vs Boolean)
console.log(null === undefined); // Output: false (Different types)
console.log(" " === 0);   // Output: false (Different types)
```

- ***Key Differences Between == and ===***



Feature	`==` (Loose Equality)	`===` (Strict Equality)
Checks	Only values	Values and types
Type Coercion	Yes (automatic conversion)	No (must be the same type)
Performance	Slightly slower (due to conversion)	Faster (direct comparison)
Example	`"5" == 5` → `true`	`"5" === 5` → `false`