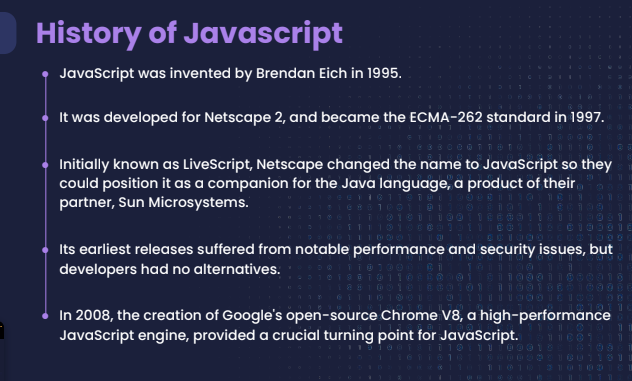
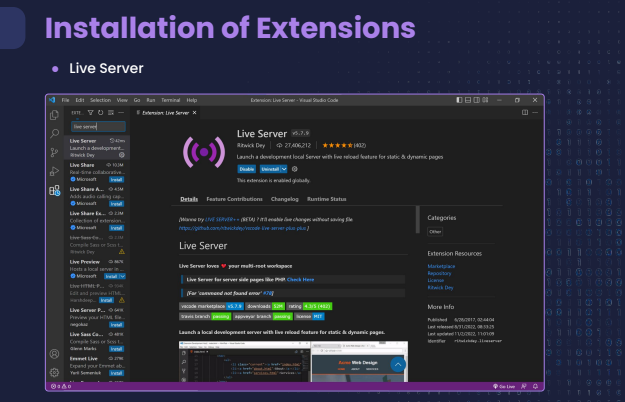
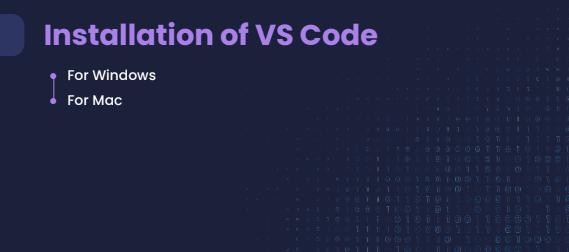
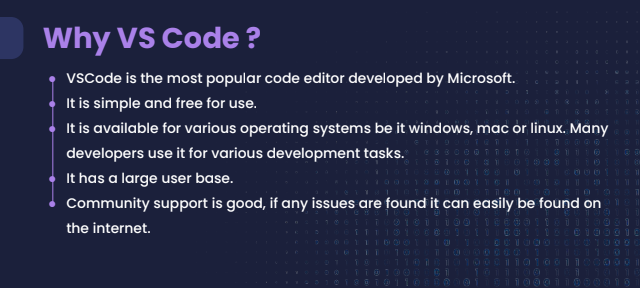
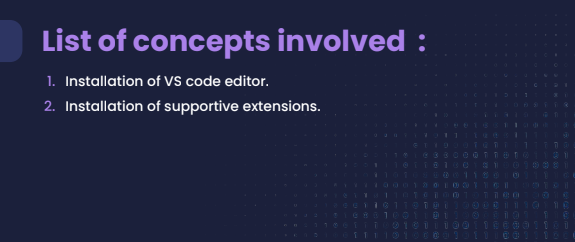
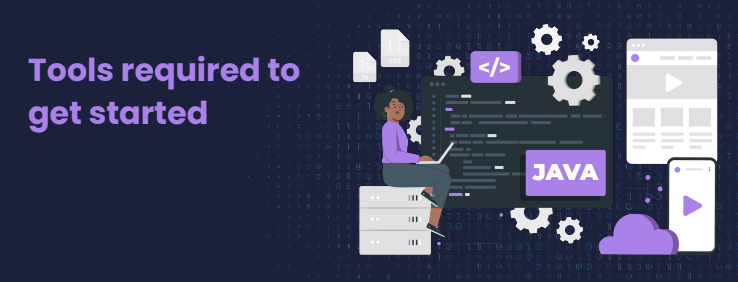
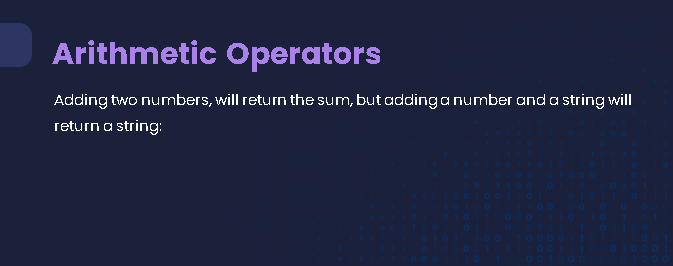
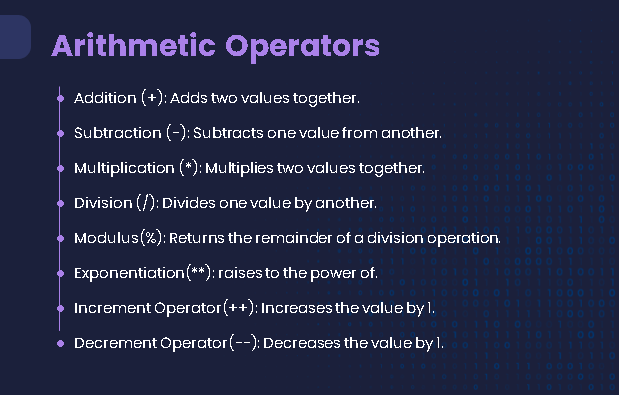
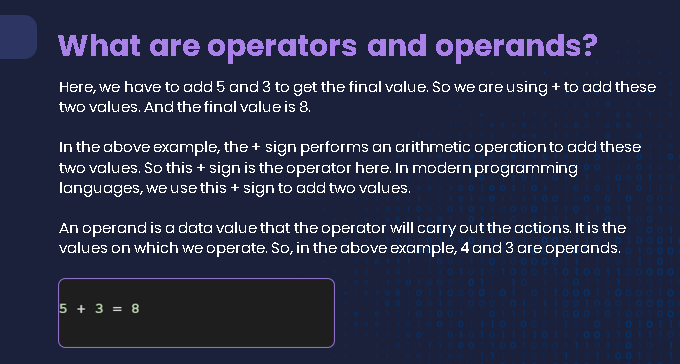


**Initially the name was Mocha (Based on a type of coffee)**







**// Adding two numbers**

**let num1 = 5;**

**let num2 = 10;**

**let result1 = num1 + num2;**

**console.log(result1); // Output: 15**

**// Adding a number and a string**

**let num3 = 7;**

**let str1 = " apples";**

**let result2 = num3 + str1;**

**console.log(result2); // Output: "7 apples"**

**In JavaScript, let is a keyword used to declare variables. It's an alternative to the var keyword but offers better scoping rules. Variables declared with let have block scope, which means they are only accessible within the block of code where they are declared. This helps avoid errors that can occur with var due to its function scope.**

**Here's an example to illustrate the difference between let and var:**

**// Using var**

**function exampleVar() {**

**if (true) {**

**var x = 10;**

**}**

**console.log(x); // Output: 10**

**}**

**// Using let**

**function exampleLet() {**

**if (true) {**

**let y = 20;**

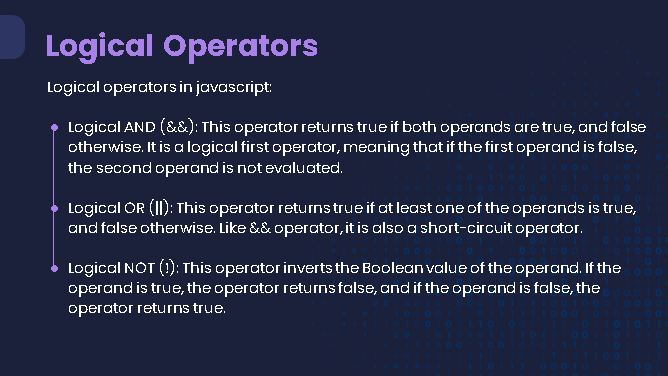
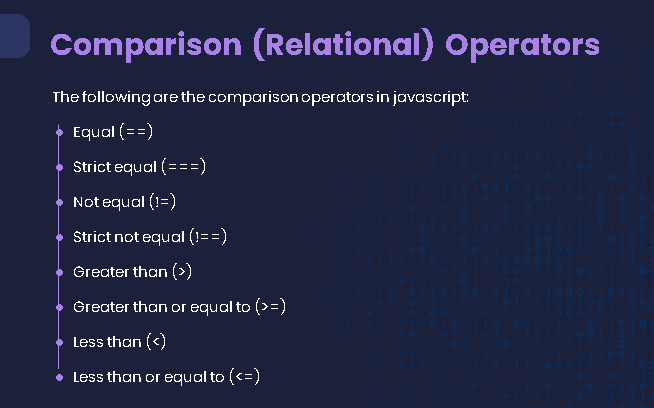
**}**

**console.log(y); // Output: ReferenceError: y is not defined**

**}**

**exampleVar();**

**exampleLet();**



Here are some examples of logical AND (&&), OR (||), and NOT (!) operators in JavaScript:

1. Logical AND (&&): This operator returns true if both operands are true.

javascript

let a = true;

let b = false;

console.log(a && b); // Output: false

// Another example

let age = 20;

let hasPermission = true;

if (age >= 18 && hasPermission) {

console.log("Access granted"); // Output: Access granted

} else {

console.log("Access denied");

}

1. Logical OR (||): This operator returns true if at least one of the operands is true.

javascript

let c = true;

let d = false;

console.log(c || d); // Output: true

// Another example

let isMember = false;

let hasGuestPass = true;

if (isMember || hasGuestPass) {

console.log("Welcome to the club"); // Output: Welcome to the club

} else {

console.log("Entry denied");

}

1. Logical NOT (!): This operator returns true if the operand is false, and vice versa.

javascript

let e = true;

console.log(!e); // Output: false

// Another example

let isLoggedIn = false;

if (!isLoggedIn) {

console.log("Please log in"); // Output: Please log in

} else {

console.log("Welcome back!");

}



Absolutely! Bitwise operators in JavaScript are used to perform operations on binary representations of numbers. Here are examples of commonly used bitwise operators:

1. **Bitwise AND (**&**)**: This operator performs a binary AND operation on each pair of corresponding bits of the operands.

javascript

let a = 5; // Binary: 0101

let b = 3; // Binary: 0011

let result = a & b; // Binary: 0001

console.log(result); // Output: 1

1. **Bitwise OR (**|**)**: This operator performs a binary OR operation on each pair of corresponding bits of the operands.

javascript

let c = 5; // Binary: 0101

let d = 3; // Binary: 0011

let result2 = c | d; // Binary: 0111

console.log(result2); // Output: 7

1. **Bitwise XOR (**^**)**: This operator performs a binary XOR operation on each pair of corresponding bits of the operands.

javascript

let e = 5; // Binary: 0101

let f = 3; // Binary: 0011

let result3 = e ^ f; // Binary: 0110

console.log(result3); // Output: 6

1. **Bitwise NOT (**~**)**: This operator inverts all the bits of the operand.

javascript

let g = 5; // Binary: 0101

let result4 = ~g; // Binary: 1010 (inverted)

console.log(result4); // Output: -6 (Two's complement of 1010)

1. **Bitwise Left Shift (**<<**)**: This operator shifts the bits of the operand to the left by the specified number of positions.

javascript

let h = 5; // Binary: 0101

let result5 = h << 1; // Binary: 1010

console.log(result5); // Output: 10

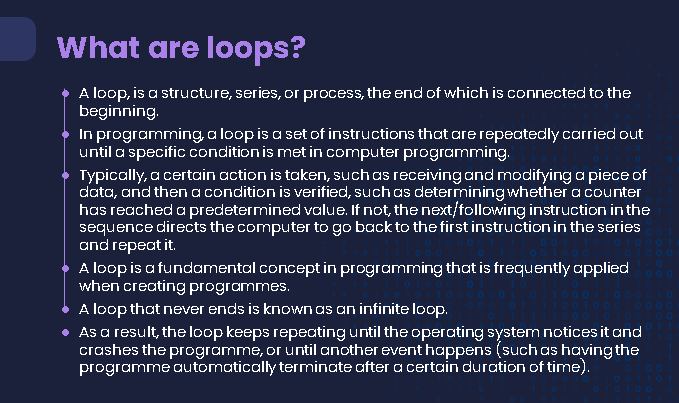
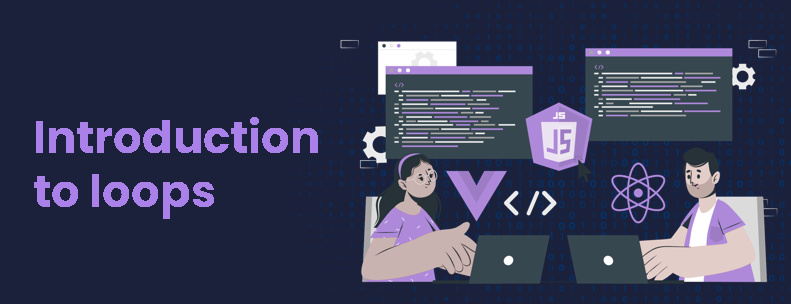
1. **Bitwise Right Shift (**>>**)**: This operator shifts the bits of the operand to the right by the specified number of positions.

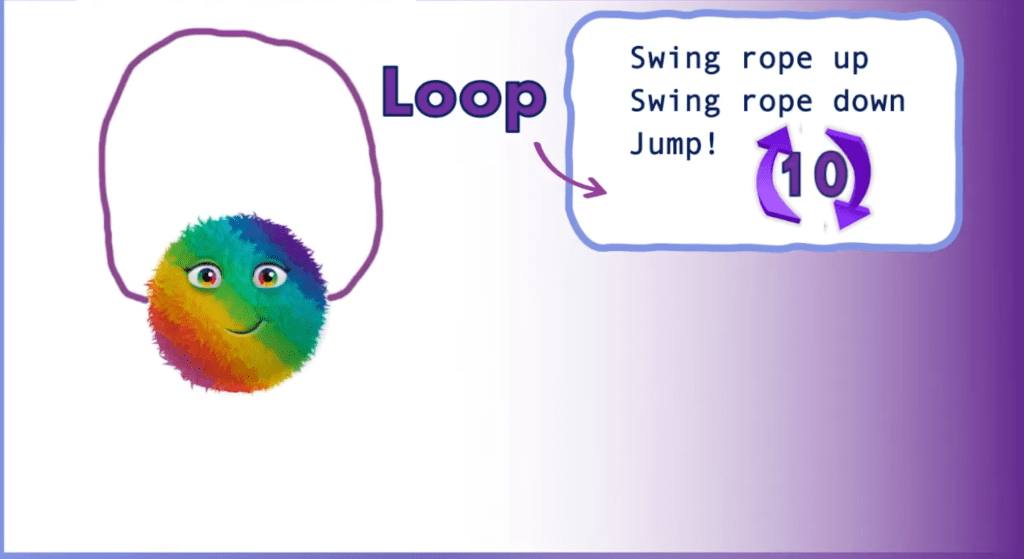
javascript

let i = 5; // Binary: 0101

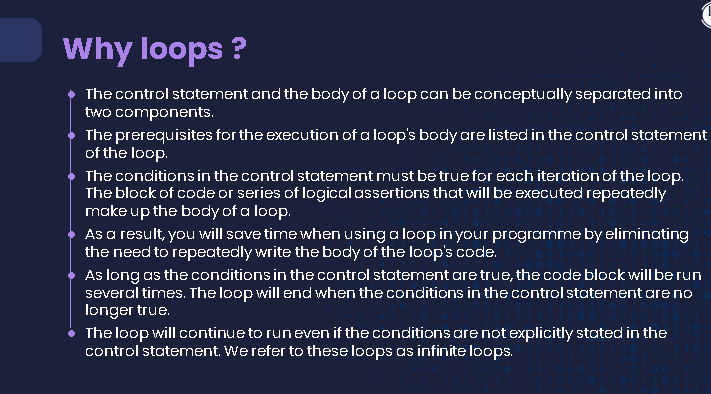
let result6 = i >> 1; // Binary: 0010

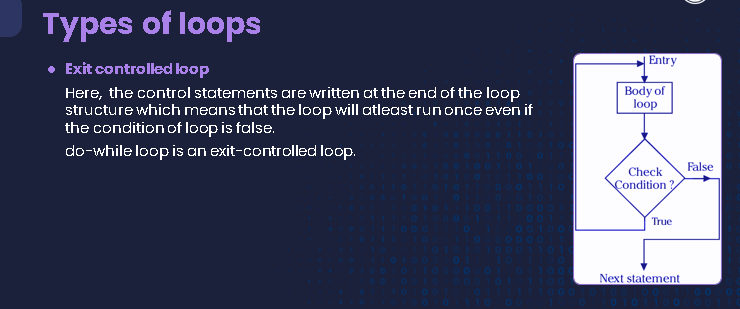
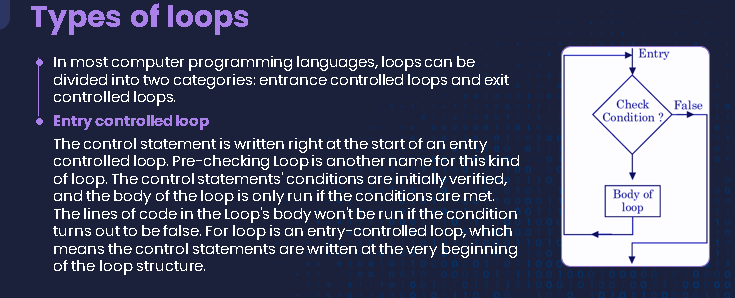
console.log(result6); // Output: 2











Let's consider a real-life example of a do-while loop to simulate a simple menu-driven program. Suppose you want to create a program that repeatedly displays a menu to the user until they choose to exit:

javascript

let choice;

do {

console.log("Menu:");

console.log("1. Check Balance");

console.log("2. Deposit Money");

console.log("3. Withdraw Money");

console.log("4. Exit");

choice = prompt("Enter your choice (1-4):");

switch (choice) {

case "1":

console.log("Your balance is $1000.");

break;

case "2":

console.log("Enter the amount to deposit:");

let deposit = prompt();

console.log(`You have deposited $${deposit}.`);

break;

case "3":

console.log("Enter the amount to withdraw:");

let withdraw = prompt();

console.log(`You have withdrawn $${withdraw}.`);

break;

case "4":

console.log("Exiting...");

break;

default:

console.log("Invalid choice. Please enter a number between 1 and 4.");

}

} while (choice !== "4");

In this example:

* The program displays a menu with options for checking balance, depositing money, withdrawing money, and exiting.
* The do block ensures that the menu is displayed at least once, regardless of the user's initial choice.
* The switch statement handles the user's choice and performs the corresponding action.
* The while condition checks if the user's choice is not "4" (exit). If it is not, the loop repeats, displaying the menu again.

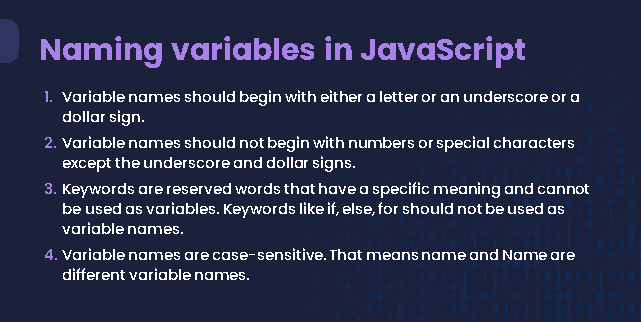
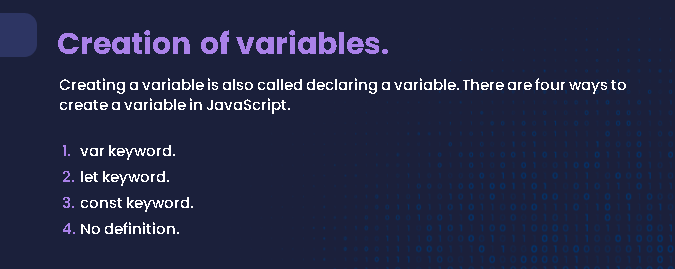
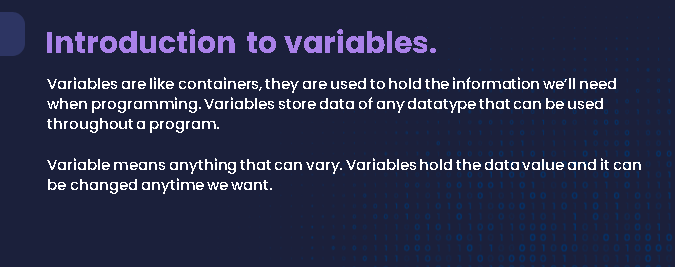
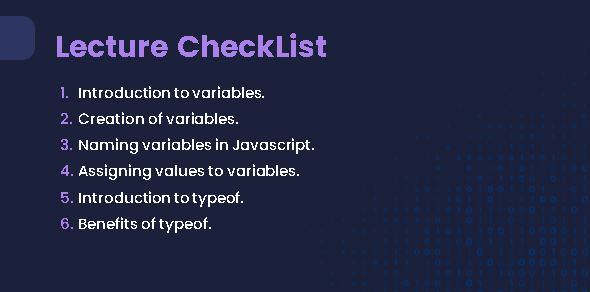
Example: Car's Fuel Level Monitoring

Imagine you have a smart car system that continuously monitors the car's fuel level. The system should alert the driver if the fuel level drops below a certain threshold. The check should be performed at least once and then continue to monitor the fuel level periodically.

In this scenario:

1. The fuel level check is initiated at least once (ensured by the "do" part of the loop).
2. The loop continues to monitor the fuel level and provide alerts as long as the car is running (handled by the "while" condition).

This type of loop ensures that the fuel level is always monitored, and the driver is alerted if they need to refuel, enhancing safety and convenience.



let bookTitle = "The Great Gatsby"; // Title of the book

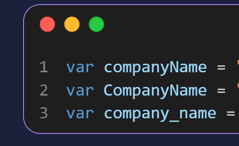
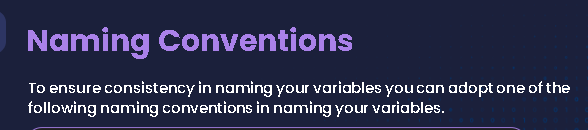
let authorName = "F. Scott Fitzgerald"; // Author's name

let publicationYear = 1925; // Year of publication

let isAvailable = true; // Availability status

let \_numberOfCopies = 3; // Number of copies available

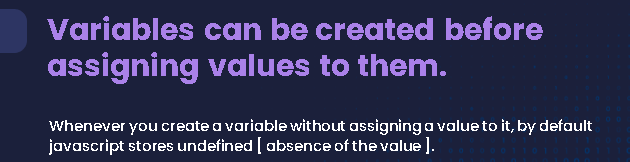
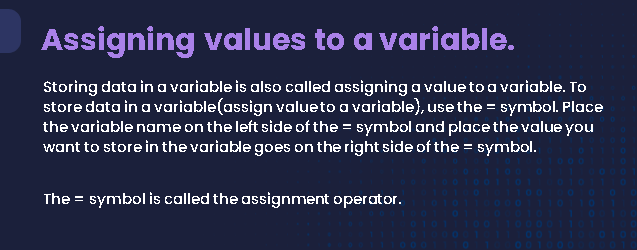
let $librarySection = "Fiction";



**// camel Case**

**// Pascal Case**

**// snake\_case**

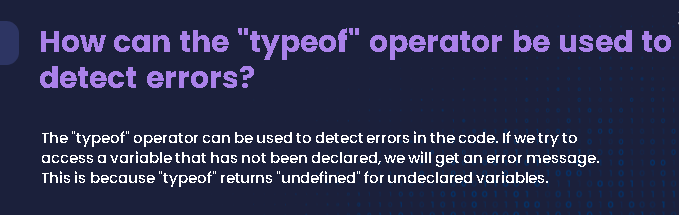
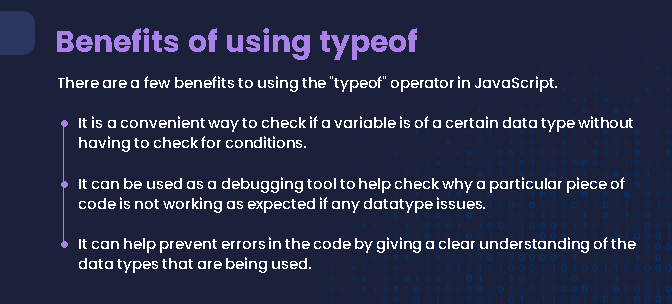
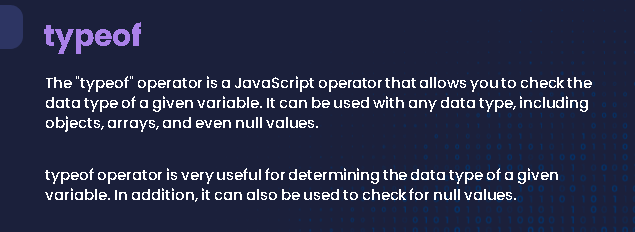


**var = name**

**name = “Hello World”**



**var name = “Hello World”;**

****

**let age = 25;**

**console.log(typeof age); // "number"**

**let name = "Alice";**

**console.log(typeof name); // "string"**

**let isStudent = true;**

**console.log(typeof isStudent); // "boolean"**

**let colors = ["red", "green", "blue"];**

**console.log(typeof colors); // "object"**

**let greet = function() {**

**return "Hello!";**

**};**

**console.log(typeof greet); // "function"**

**let x;**

**console.log(typeof x); // "undefined"**

**let y = null;**

**console.log(typeof y); // "object" (this is a special case in JavaScript)**