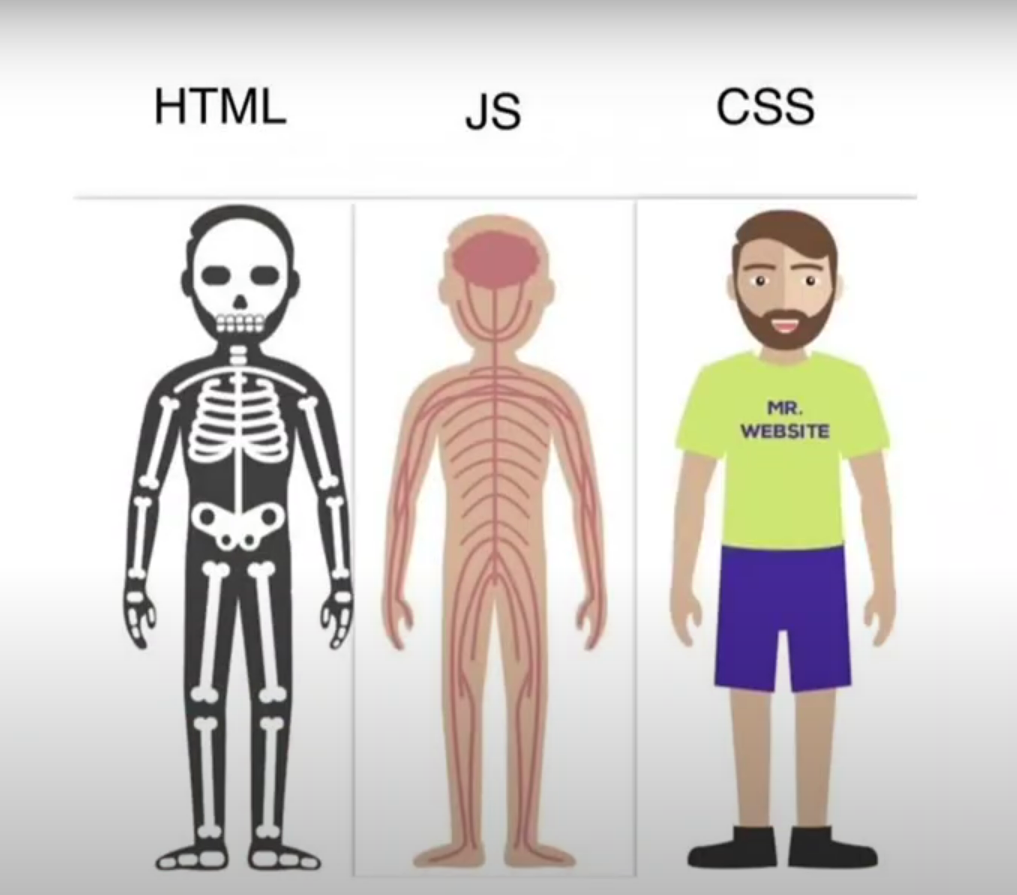
**Java Script**

1. **why we need java script?**

Developers use JavaScript in web development to add interactivity and features to [improve the user experience](https://blog.hubspot.com/website/usability-vs-user-experience?__hstc=45788219.f59dd9dc41b459e719c5b4cd9a768a27.1646249529534.1646249529534.1646249529534.1&__hssc=45788219.1.1646249529535&__hsfp=1603130137&_ga=2.168554793.901349043.1646249529-1254543002.1646249529&hubs_content=blog.hubspot.com/website/what-is-javascript&hubs_content-cta=improve%20the%20user%20experience) and make the internet much more enjoyable.



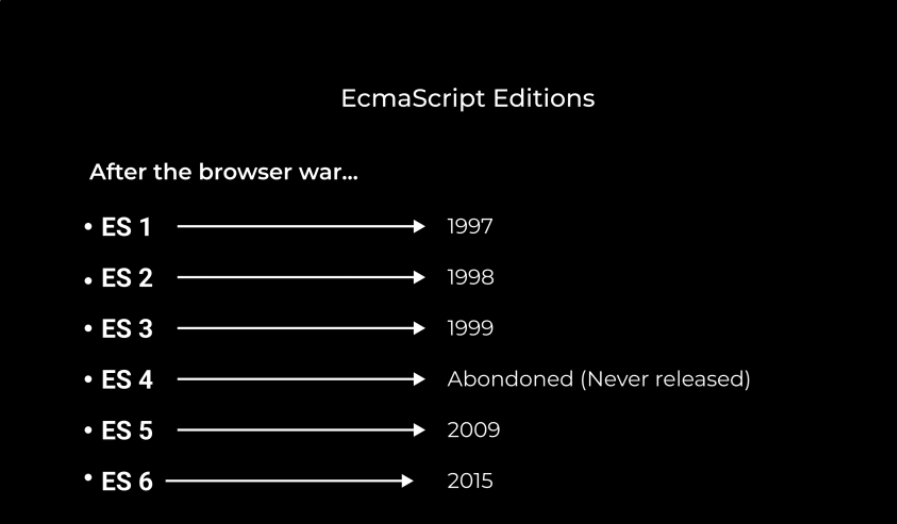
1. **History of java script?**

**Netscape programmer** named **Brendan Eich** developed a new scripting language in just **10 days**. It was originally called **Mocha**, but quickly became known as **LiveScript** and, later, **JavaScript**.

**What Is ECMAScript?**

When JavaScript was first introduced by Netscape, there was a war going on between all the browser vendors on the market at the time. Microsoft and several other browser vendors implemented their own versions of JavaScript (with different names and syntax) in their respective browsers. This created a huge headache for developers, as code that worked fine on one browser was a total waste on another. This went on for a while till they all agreed to use the same language (JavaScript) in their browsers.

As a result, Netscape submitted JavaScript to the [European Computer Manufacturers Association](https://www.ecma-international.org/) (ECMA) for standardization in order to ensure proper maintenance and support of the language. Since JavaScript was standardized by ECMA, it was officially named ECMAScript.

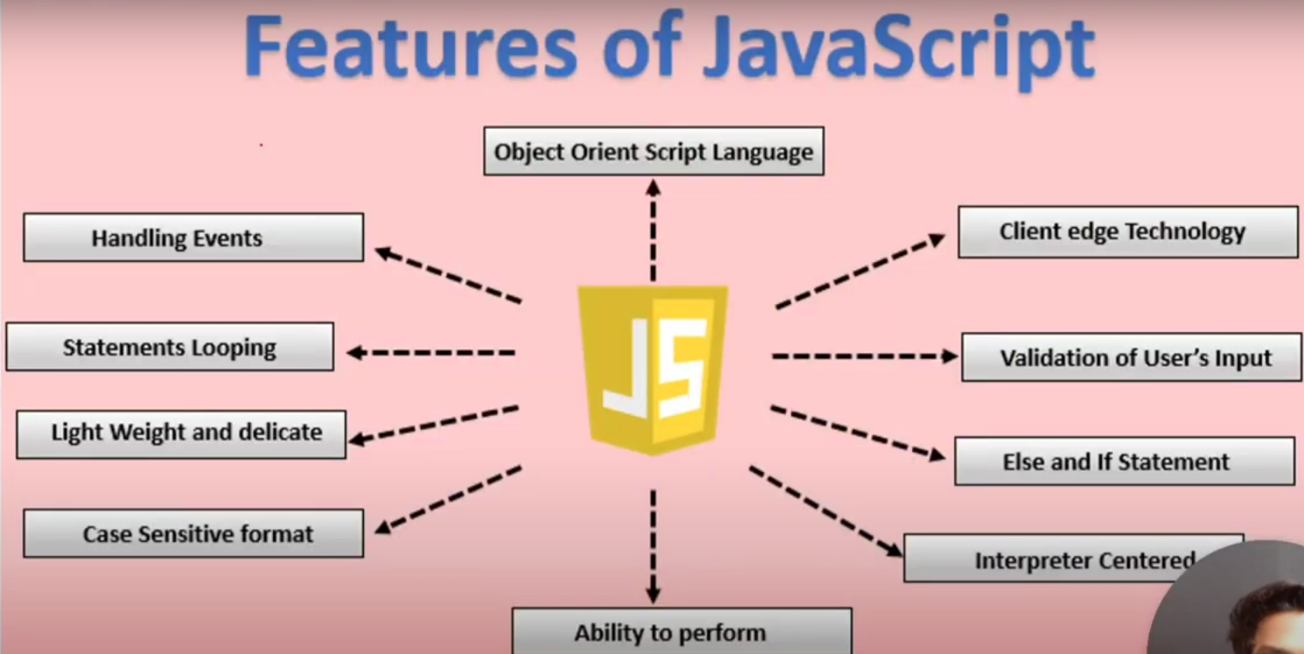


Originally, the name ECMAScript was just the formalization of JavaScript, but now languages like JScript and ActionScript are also based on the ECMAScript standard.

1. **What is java script?**

Java Script is a High level Programing Language that is primarly used to enhance the interactivity and dynamic behaviour of web sites

Java Script is also a light weight, cross platforms, Single threaded and High level Interpreted compiled programming language. It is knowns as Scripting Languages for websites.



* **High-Level Language**

High-level languages are programming languages that are used for writing programs or software that can be understood by humans and computers. High-level languages are easier to understand for humans because they use a lot of symbols letters phrases to represent logic and instructions in a program. It contains a high level of abstraction compared to low-level languages.

* **Cross-platform**

It is a software or applications that can operate on multiple operating system

* **Single-theaded**

It is the only programming language that can run natively in a browser, making it an instrumental part of web development. However, one critical feature of JavaScript is that it is single-threaded. This means that it can only execute one task at a time.

* **Asynchronous**

how it can be used to effectively handle potential blocking operations, such as fetching resources from a server.

* **Synchronous**

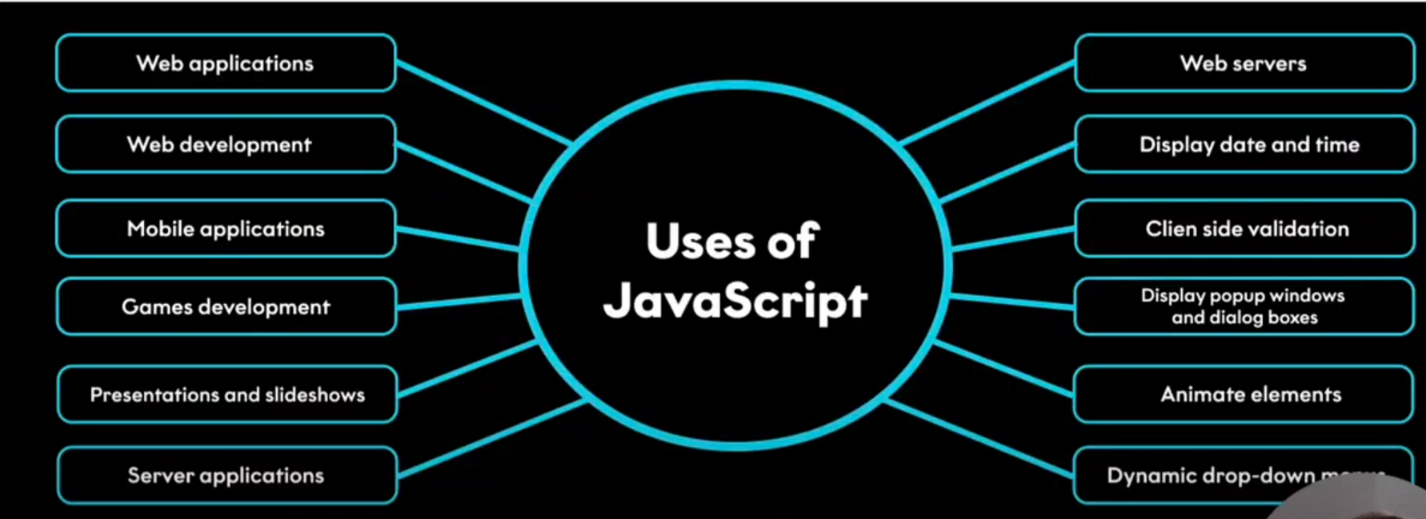
Synchronous means the code runs in a particular sequence of instructions given in the program. Each instruction waits for the previous instruction to complete its execution.

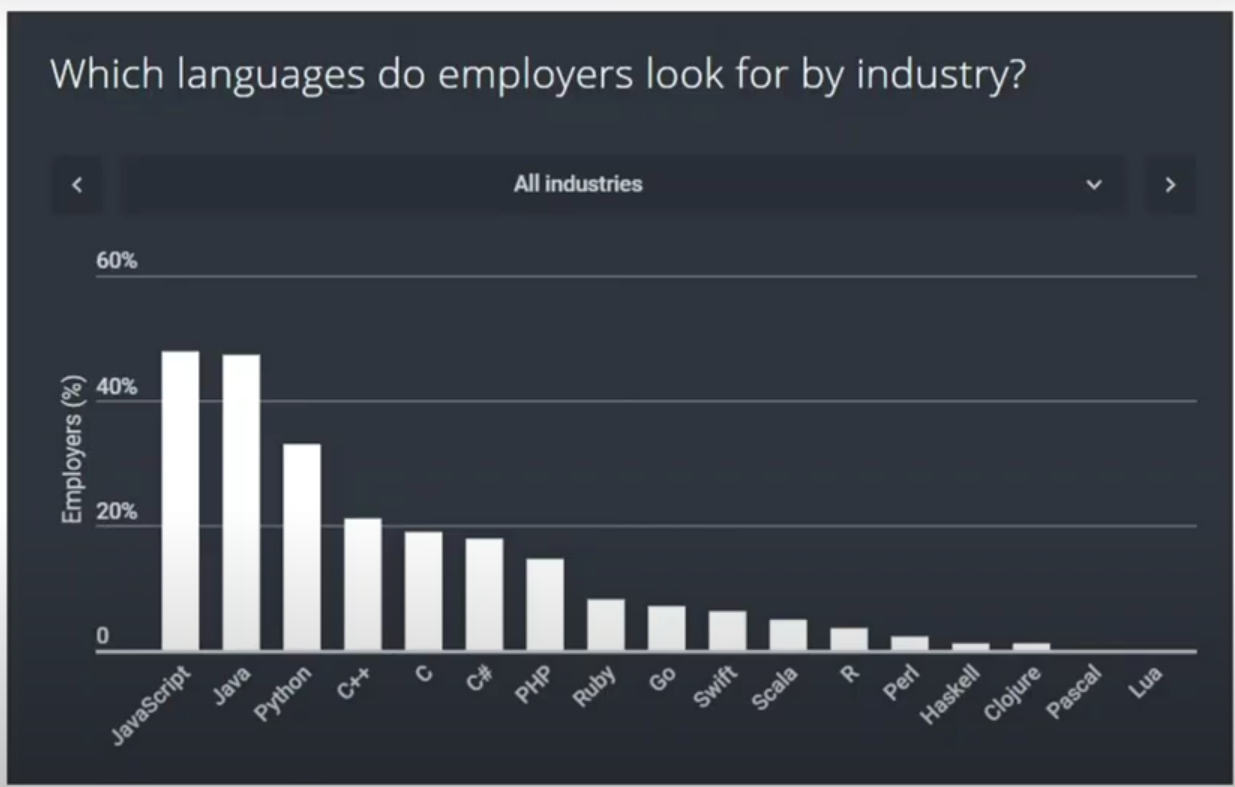
* **Obeject Oriented**

It provides an overview of the basic concepts of OOP.

* **Scripting**

Scripting is used to automate tasks on a website. It can respond to any specific event, like button clicks, scrolling, and form submission. It can also be used to generate dynamic content. and JavaScript is a widely used scripting language.





1. **What is Vanilla JavaScript**

The term vanilla script is used to refer to the pure JavaScript (or we can say plain JavaScript) without any type of additional library. Sometimes people often used it as a joke"nowadays several things can also be done without using any additional JavaScript libraries".

The vanilla script is one of the lightest weight frameworks ever. It is very basic and straightforward to learn as well as to use. You can create significant and influential applications as well as websites using the vanilla script.

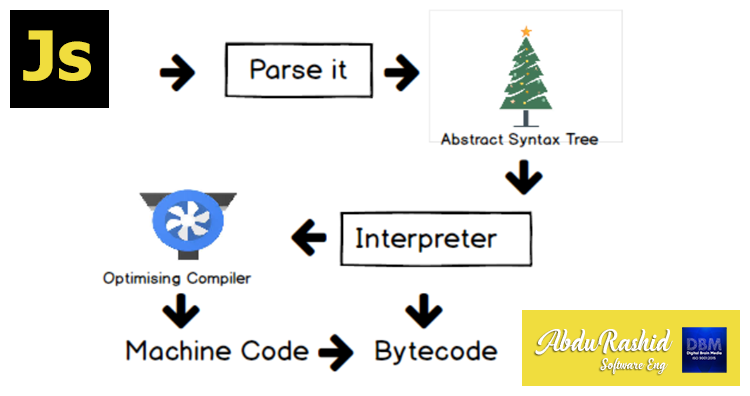
The team of developers that created the vanilla JavaScript is continuously working on it to improve it and make it more useful for the web-developers.

1. **why JavaScript programming language?**

It supplies objects relevant to running JavaScript on a server. For if the server-side extensions allow an application to communicate with a database, and provide continuity of information from one invocation to another of the application, or perform file manipulations on a server. The useful framework which is the most famous these days is [**node.js**](https://www.geeksforgeeks.org/introduction-to-nodejs/).

1. **What is a JavaScript Engine?**

A JavaScript engine is a program that compiles JavaScript code and executes it. It is a software that runs inside a web browser or on a server that interprets JavaScript code and executes it. The engine is responsible for parsing the JavaScript code, compiling it into machine code, and executing it.



**Parsing**

The first stage of a JavaScript engine is parsing. It is the process of breaking down the source code into its individual components, such as keywords, variables, and operators. The parser creates a tree structure called the Abstract Syntax Tree (AST), which represents the structure of the code.

**Compilation**

The next stage is compilation. The compiler takes the AST and converts it into machine code. The machine code is optimized to run efficiently on the target platform. The compilation process includes several optimizations, such as inlining, loop unrolling, and dead code elimination.

**Execution**

The final stage is execution. The compiled code is executed by the JavaScript engine. The engine executes the code line by line, keeping track of variables and function calls. It also manages memory allocation and deallocation.

1. **Java script Run time environment?**

JavaScript works on a environment called **JavaScript Runtime Environment**. To use JavaScript you basically install this environment and than you can simply use JavaScript.

So in order to use JavaScript you install a **Browser** or **NodeJS** both of which are JavaScript Runtime Environment.

**call stack:**

The call stack is used to store information about function calls, including local variables, parameters, and the point of execution.

**Heap:**

The heap is a region of memory used for dynamic memory allocation. It stores objects, arrays, and other complex data structures that are created and managed at runtime.

**Web API:**

A Web API (Application Programming Interface) is a set of rules and tools that allows different software applications to communicate with each other over the web. It acts as an intermediary, enabling one application to interact with another application's data or functionality using standard web protocols, usually HTTP.

**Event loop:**

The event loop is a fundamental concept in asynchronous programming, especially in environments like JavaScript. It enables non-blocking operations, allowing code to execute asynchronously while ensuring that tasks are handled in an orderly manner. Here's a closer look at how synchronous and asynchronous operations interact with the event loop:

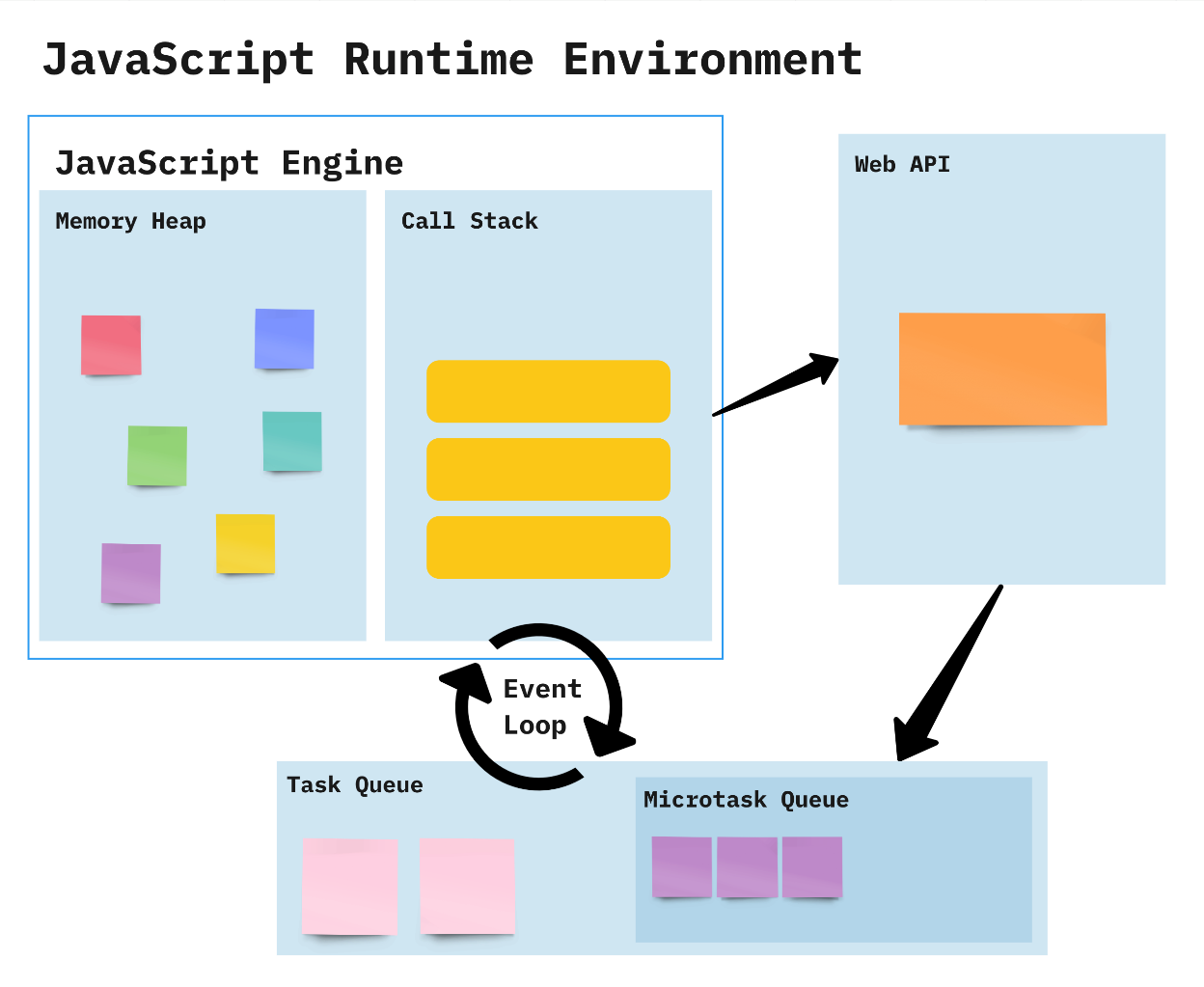
**Synchronous vs. Asynchronous**

1. **Synchronous:**

Synchronous operations are executed sequentially, one after the other. Each operation must complete before the next one begins.

1. **Asynchronous:**

Asynchronous operations allow code to execute without waiting for previous operations to complete. This is useful for tasks that involve waiting, such as network requests or timers.



**How it Works:**

* Constantly checks whether or not the call stack is empty
* When the call stack is empty, all queued up Microtasks from Microtask Queue are popped onto the callstack
* If both the call stack and Microtask Queue are empty, the event loop dequeues tasks from the Task Queue and calls them
* Starved event loop

**Difference Between Compiler and Interpreter**

| **Compiler** | **Interpreter** |
| --- | --- |
| **Steps of Programming:**   * Program Creation. * Analysis of language by the compiler and throws errors in case of any incorrect statement. * In case of no error, the Compiler converts the source code to Machine Code. * Linking of various code files into a runnable program. * Finally runs a Program. | **Steps of Programming:**   * Program Creation. * Linking of files or generation of Machine Code is not required by Interpreter. * Execution of source statements one by one. |
| The compiler saves the Machine Language in form of Machine Code on disks. | The Interpreter does not save the Machine Language. |
| Compiled codes run faster than Interpreter. | Interpreted codes run slower than Compiler. |
| The compiler generates an output in the form of (.exe). | The interpreter does not generate any output. |
| Errors are displayed in Compiler after Compiling together at the current time. | Errors are displayed in every single line. |

**How Many Ways To Insert JS**

JavaScript, also known as JS, is one of the scripting (client-side scripting) languages, that is usually used in web development to create modern and interactive web-pages. The term "script" is used to refer to the languages that are not standalone in nature and here it refers to JavaScript which run on the client machine.

1. **internal JS:**

By using script tag at the bottom of the document

**Syntax:**

<body>

<script>

Console.log(“hello world”);

</script>

</body>

1. **inline JS:**

we can apply inline js within the element.

**Syntax:**

<body>

<button onclick=”alert(‘hello world’)”>click</button>

</body>

1. **external JS:**

By creating a js file with .js extention we can insert js file into the html document. That js file is linked in the script tag.

**Syntax:**

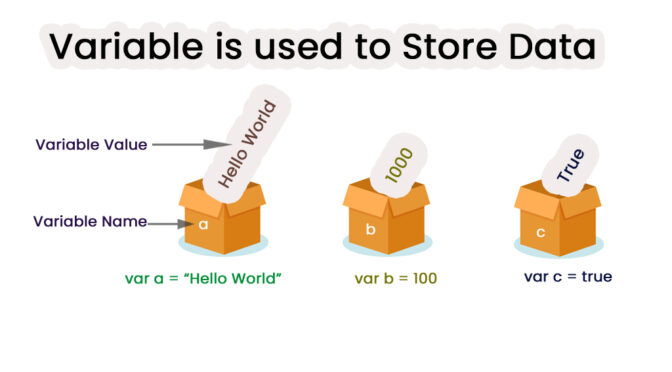
<body>

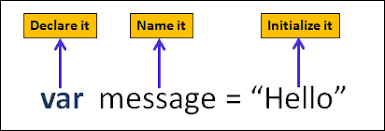
<script src=”js file with .js extention”></script>

</body>

**Variables:**

Variables are used to store data in JavaScript. Variables are used to store reusable values. The values of the variables are allocated using the assignment operator(“=”).





JavaScript Variables can be declared in 4 ways:

* **Automatically**
* **Using var**
* **Using let**
* **Using const**

Example:

a=10;

Var b=20;

let c=5;

Const d=15;

console.log(a); //10

console.log(a); //20

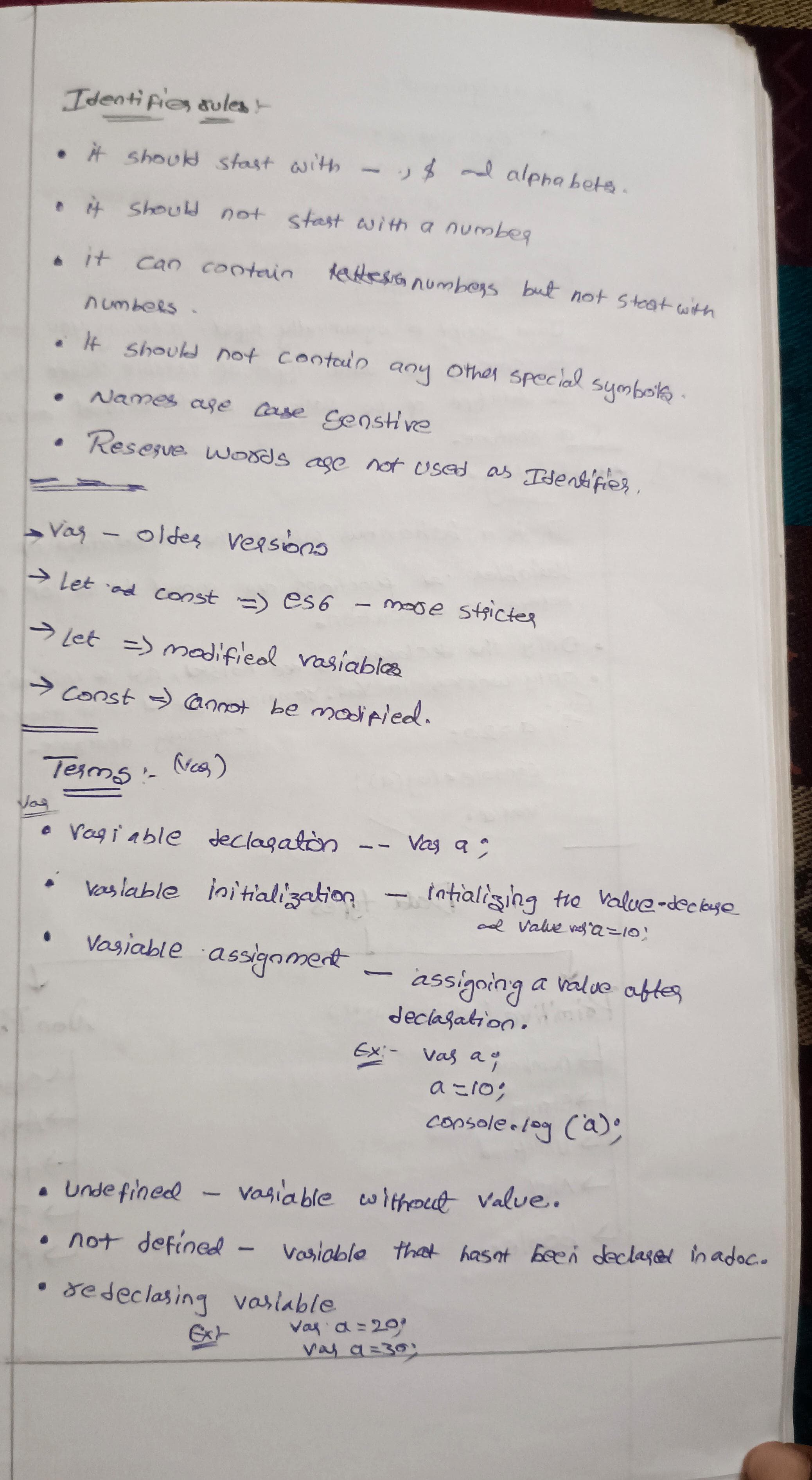
console.log(a); //5

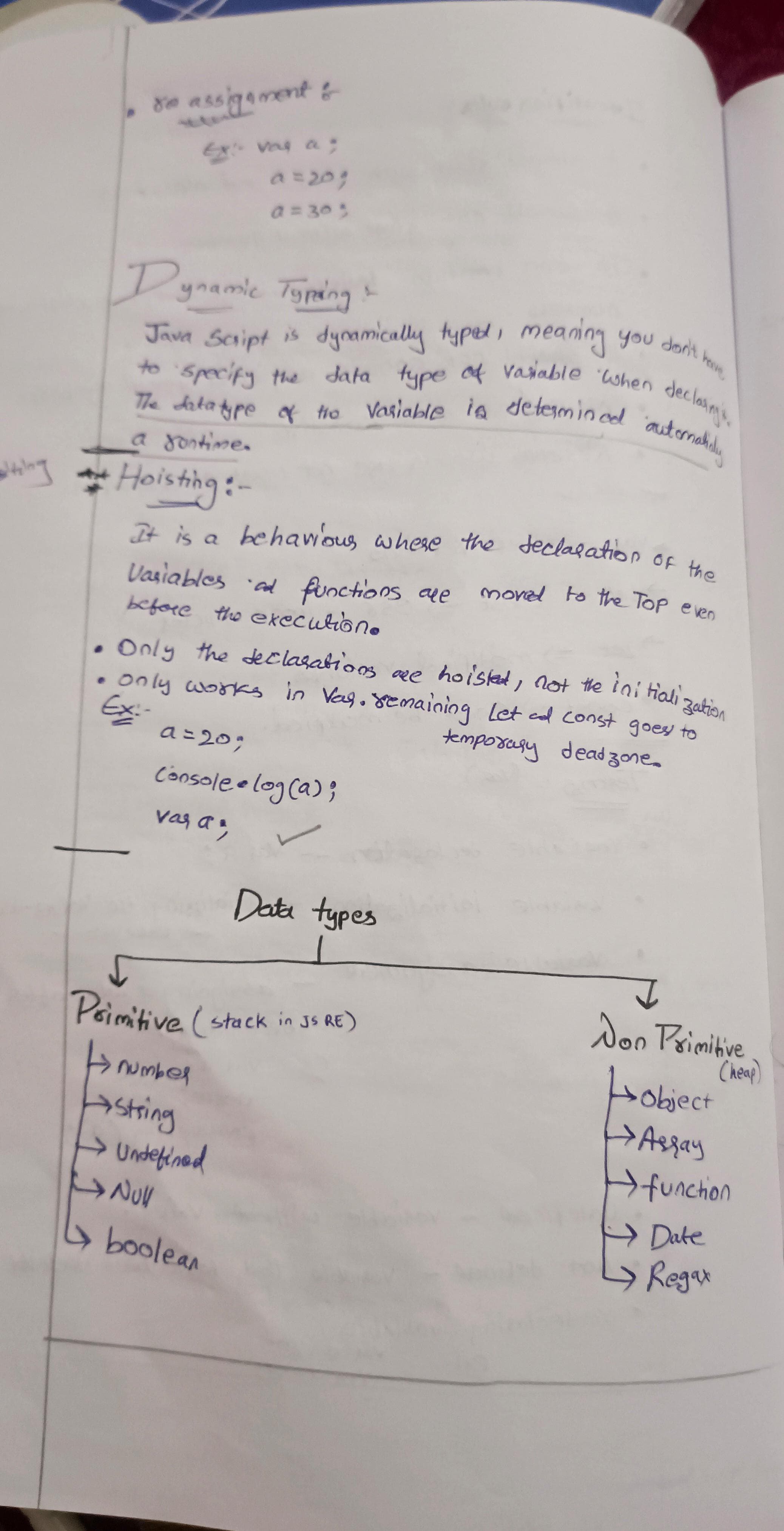
console.log(a); //15

**Rules for Identifiers:**

* Names can contain letters, digits, underscores, and dollar signs
* Identifier should not start with number
* Names must begin with a letter or \_ or $
* Names are case-sensitive
* Reserved words cannot be used as Identifier.

**Terms:**





**Dynamic typing:**

Js is a dynamically typed, meaning you do not have to specify the datatype of the variable when declared. The Data type of the variable is determined automatically in a runtime.

**Hoisting**

* It is a behaviour where the declaration of the variable and functions are moved to the top even before the execution.
* Only Declaration is hoisted not the Initialization
* Only works in **var** remaining **let** and **const** goes to temporary deadzone

Example:

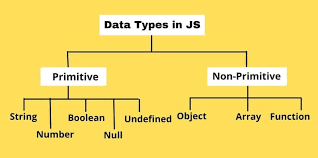
a=20;

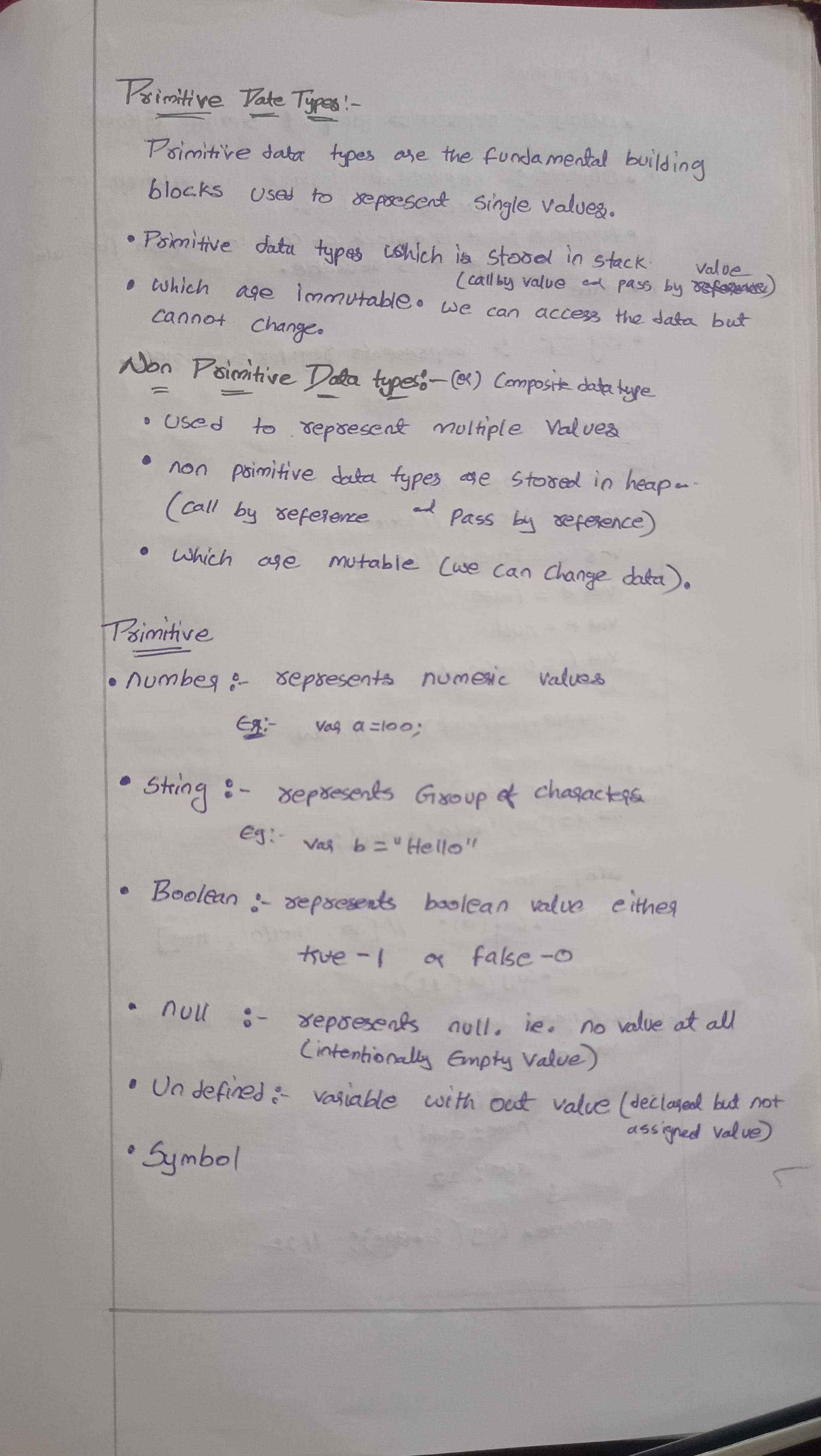
Console.lob(a); //20

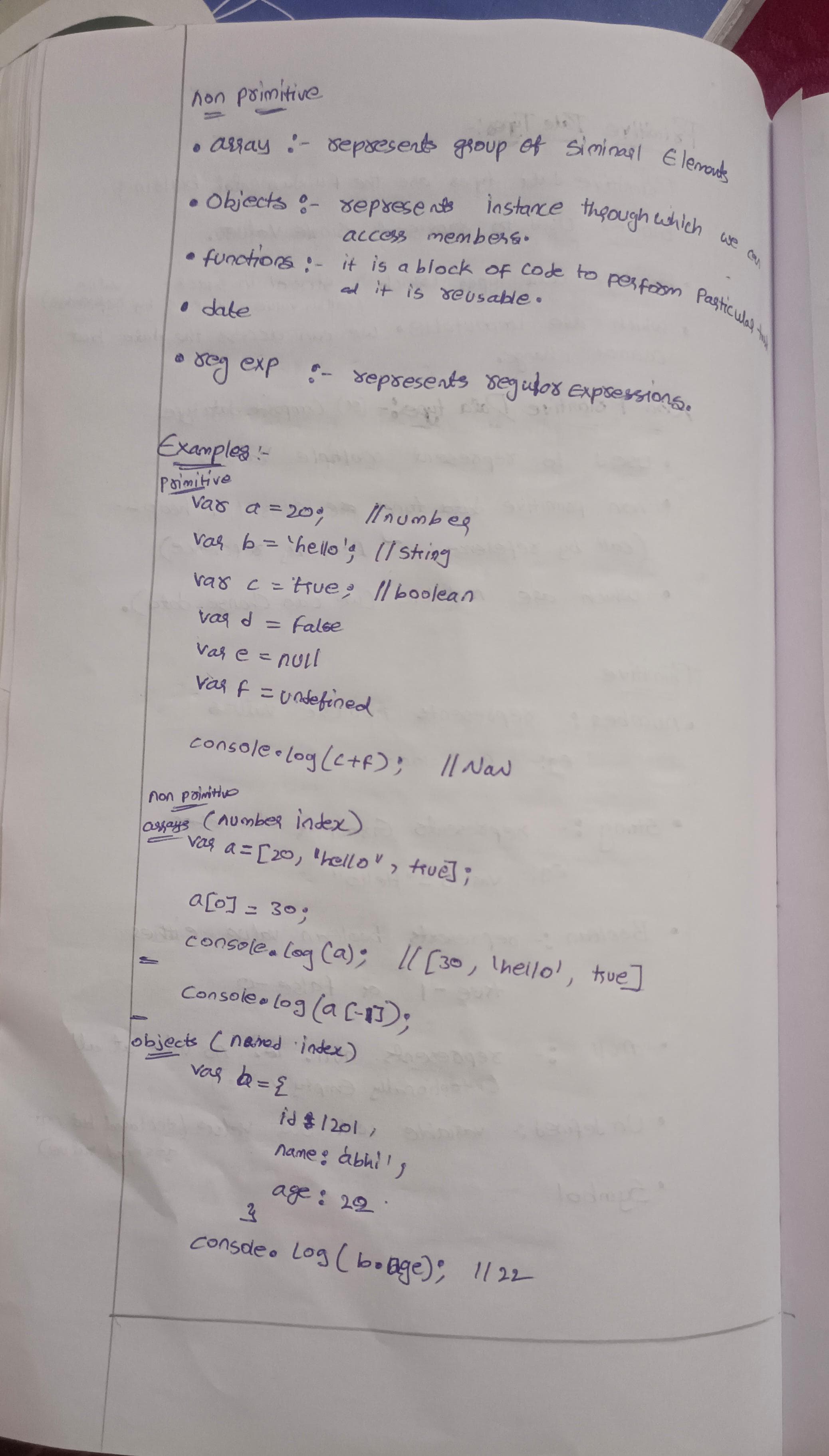
var a;

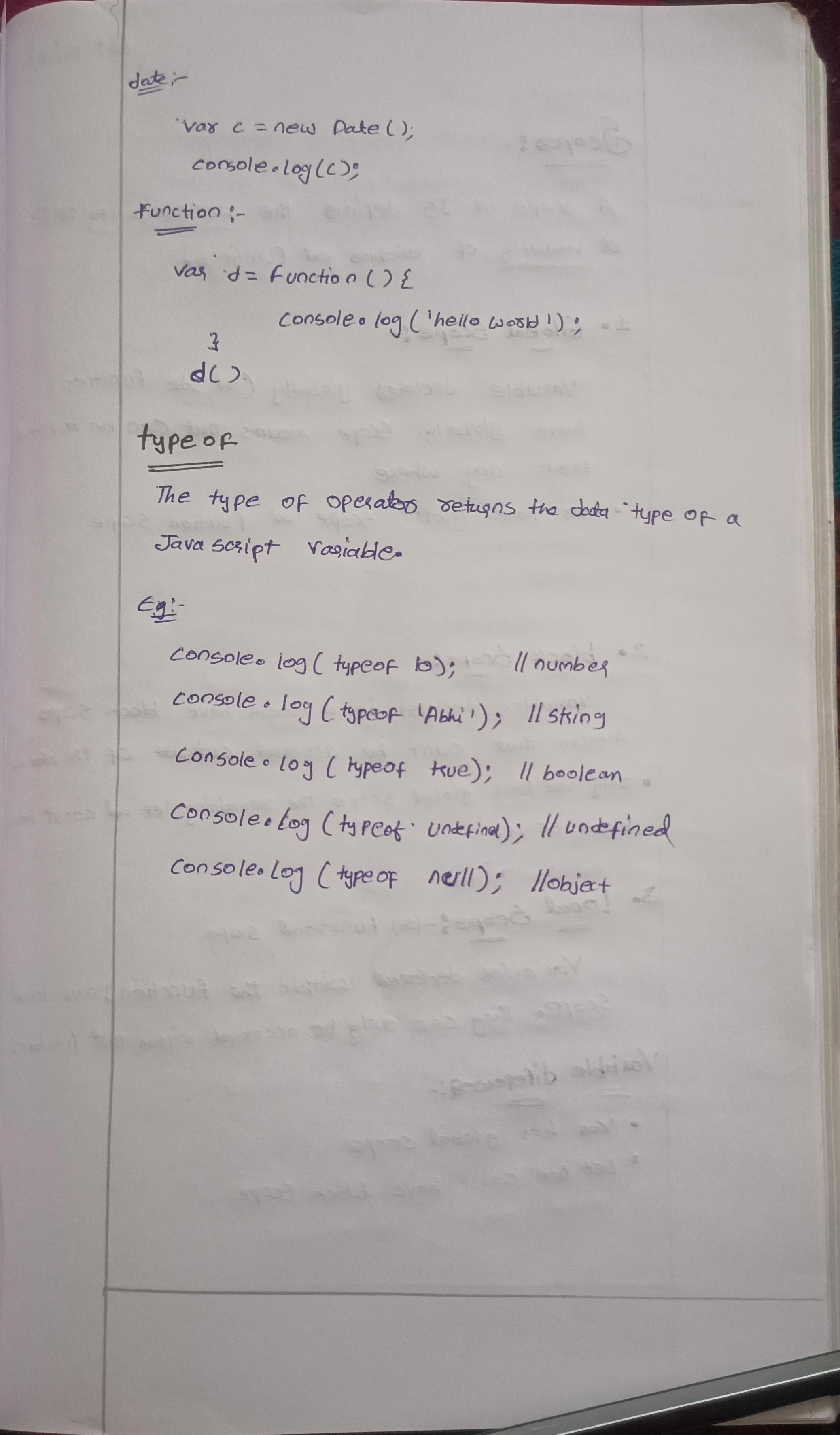
**Data types**

JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript.



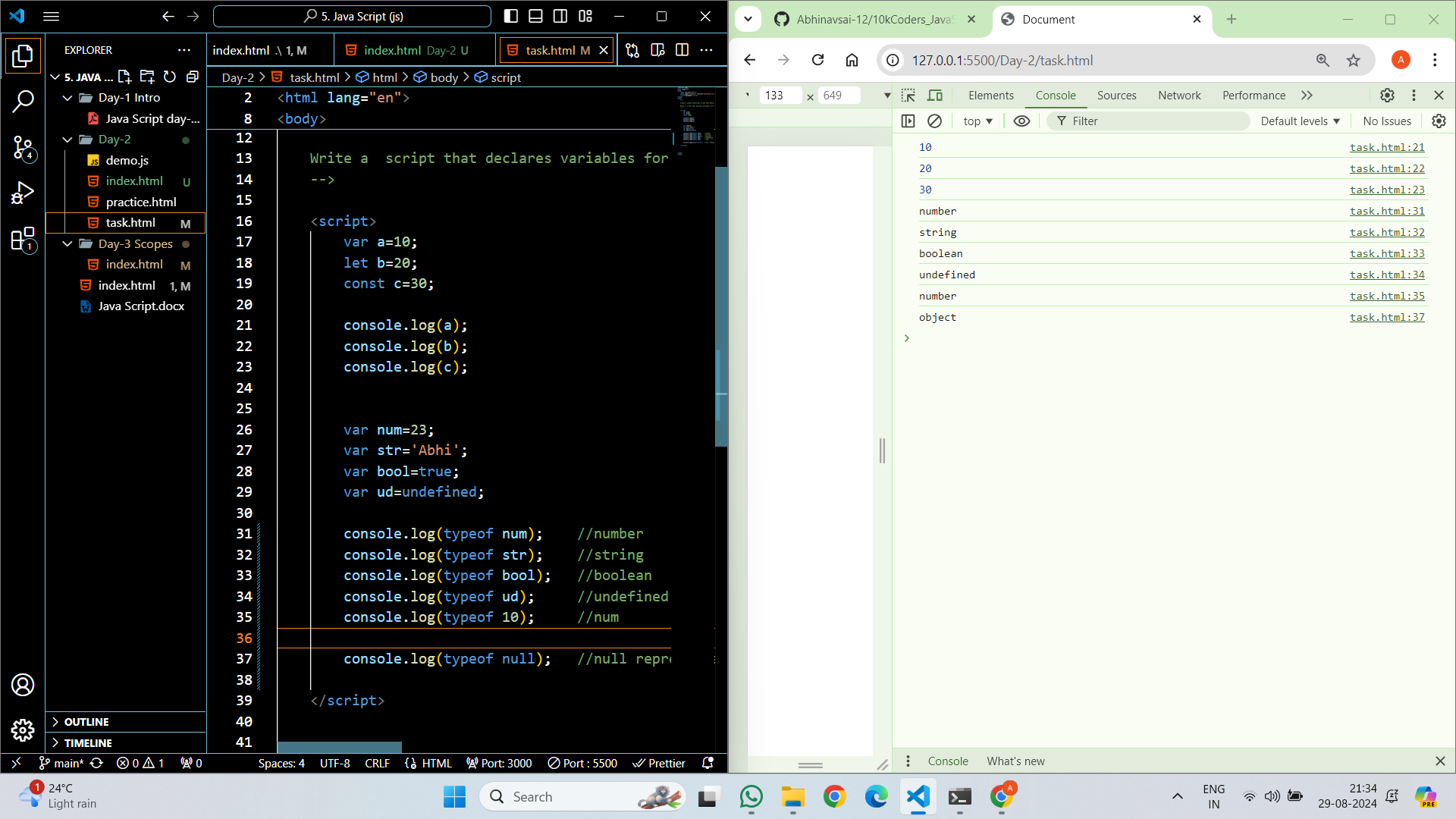






**Typeof:**  
The typeof operator returns the data type of a variable.

The JavaScript typeof operator returns the data type of a variable or expression. It’s a unary operator placed before its operand and returns a string indicating the data type, such as “number”, “string”, “boolean”, “object”, “undefined”, “function”, or “symbol”.



**Scopes**

A Scope in JS defines the Accessibility or life or Visibility of Variables and Functions.

* 1. Global Scope:

Variable declared globally (out side function) have globally. Scope means can be access from any where.

Var have global Scope and Function Scope.

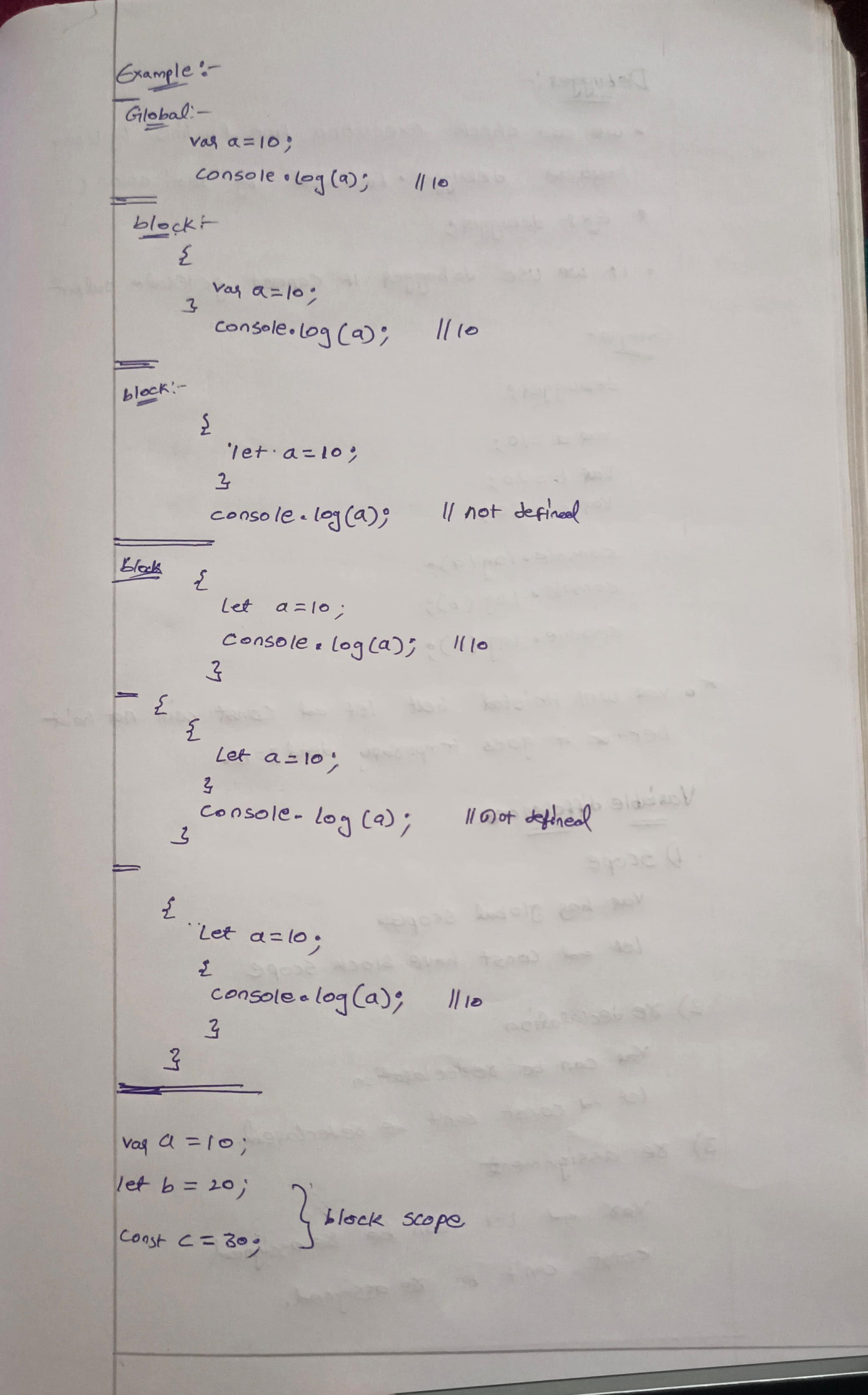
* 1. Block Scope:

Variables declared in a block have block scope means that can’t be accessed outside of the block.

Only var have global scope the remaining let and const have block scope.

* 1. Local Scope:

Variables declared within the function have local scope. They can only be accessed with in the function.



**Debugger**

We can check Execution line by line by using keyword debugger followed by semi colon (:)

**syntax:**

debugger;

**Example:**

    <script>

debugger;

        var a=10;

        let b=20;

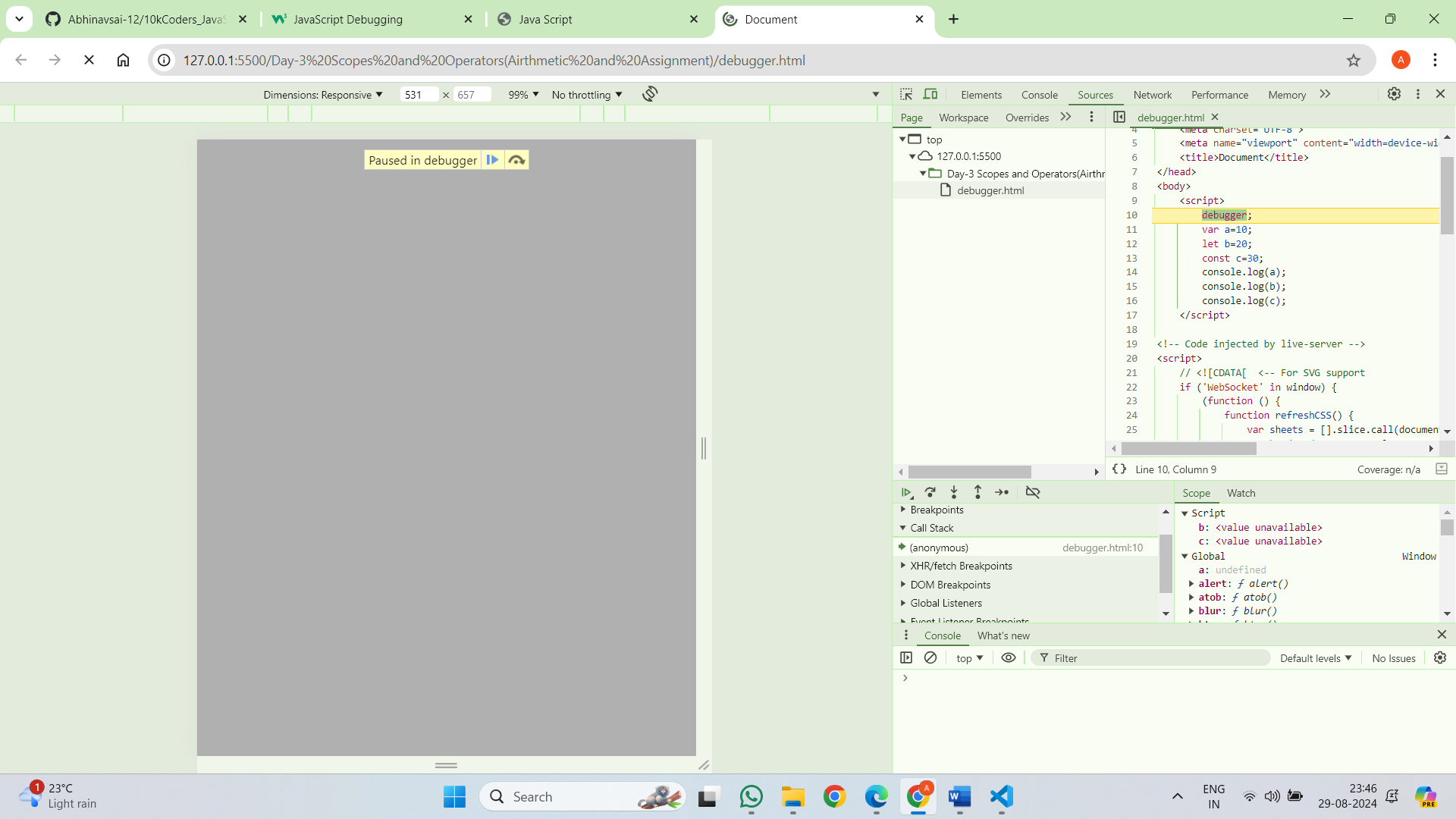
        const c=30;

        console.log(a);

        console.log(b);

        console.log(c);

    </script>



**Variable Difference:**

1. Scope  
   var has global scope

Let and const have block scope

1. Re declaration

Var can be re declared

Let and const can’t be re declared

1. Re assignment

Var and let can be re assigned

Const can’t be re assigned

**Operators**

Javascript operators are used to perform different types of mathematical and logical computations.

(or)

In JavaScript, an **operator** is a symbol that performs an operation on one or more operands, such as variables or values, and returns a result. Let us take a simple expression **4 + 5** is equal to 9. Here 4 and 5 are called **operands**, and ‘+’ is called the **operator**.

Types:

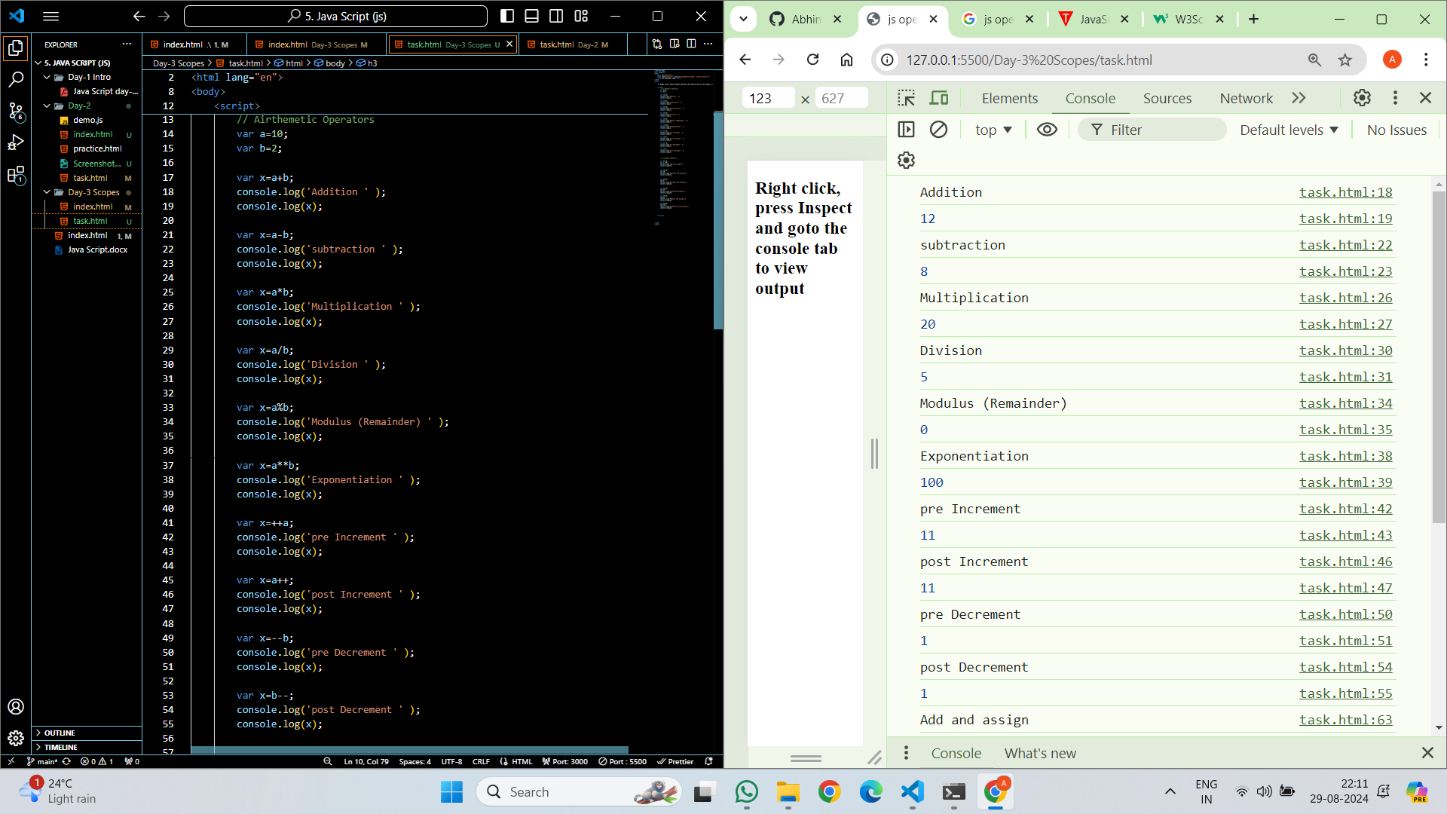
1. Airthmetic Operators
2. Assignment Operator
3. Comparision Operator
4. Logical Operator
5. Ternary Operator
6. Bitwise Oberator
7. String Operator
8. Typeof Operator

**Arithmetic operators**

Arithmetic operators are used to perform **arithmetic operations** between variables or values.



**Example:**

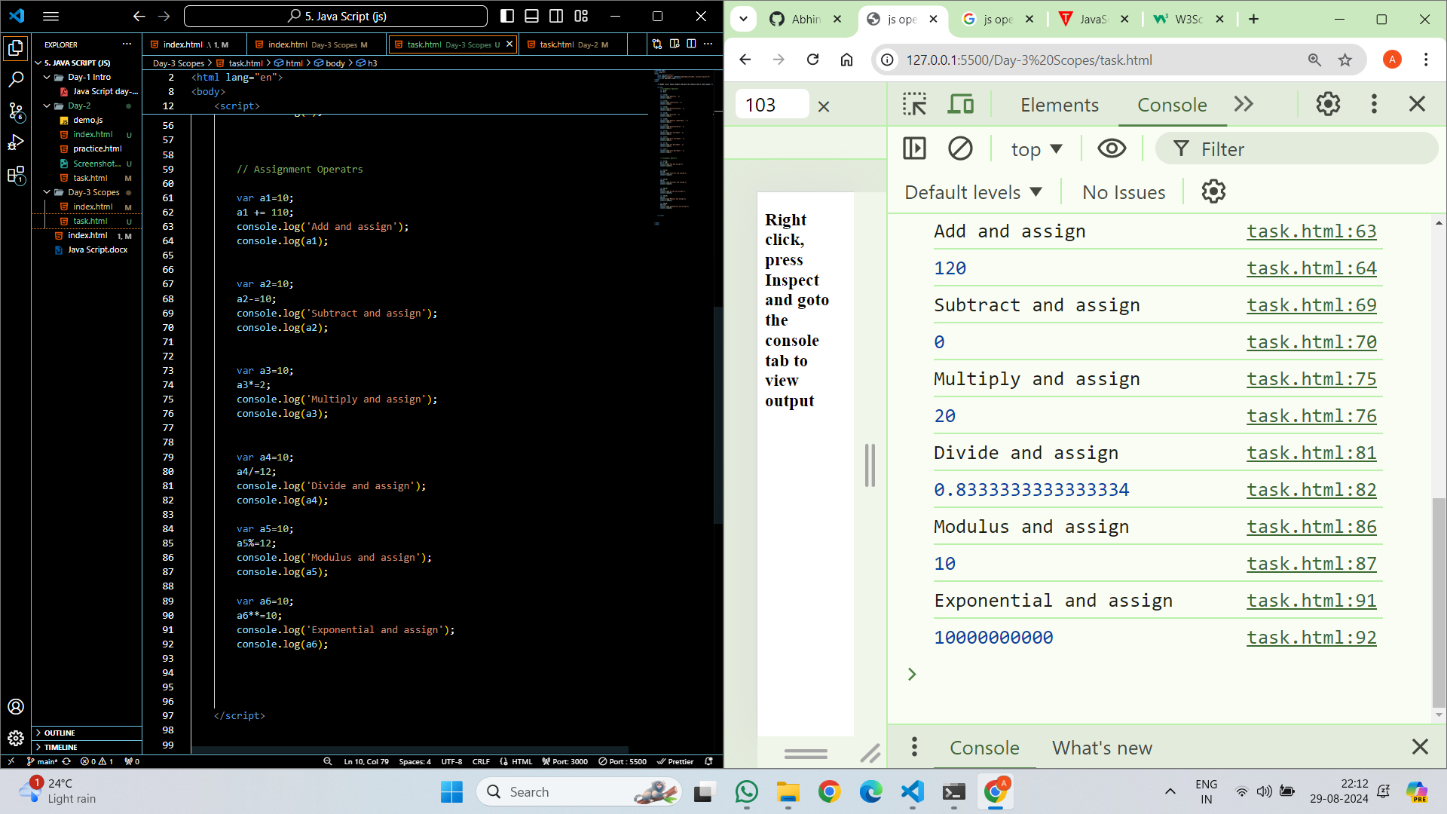


**Assignment Operators:**

We use assignment operators to **assign** values to variables.



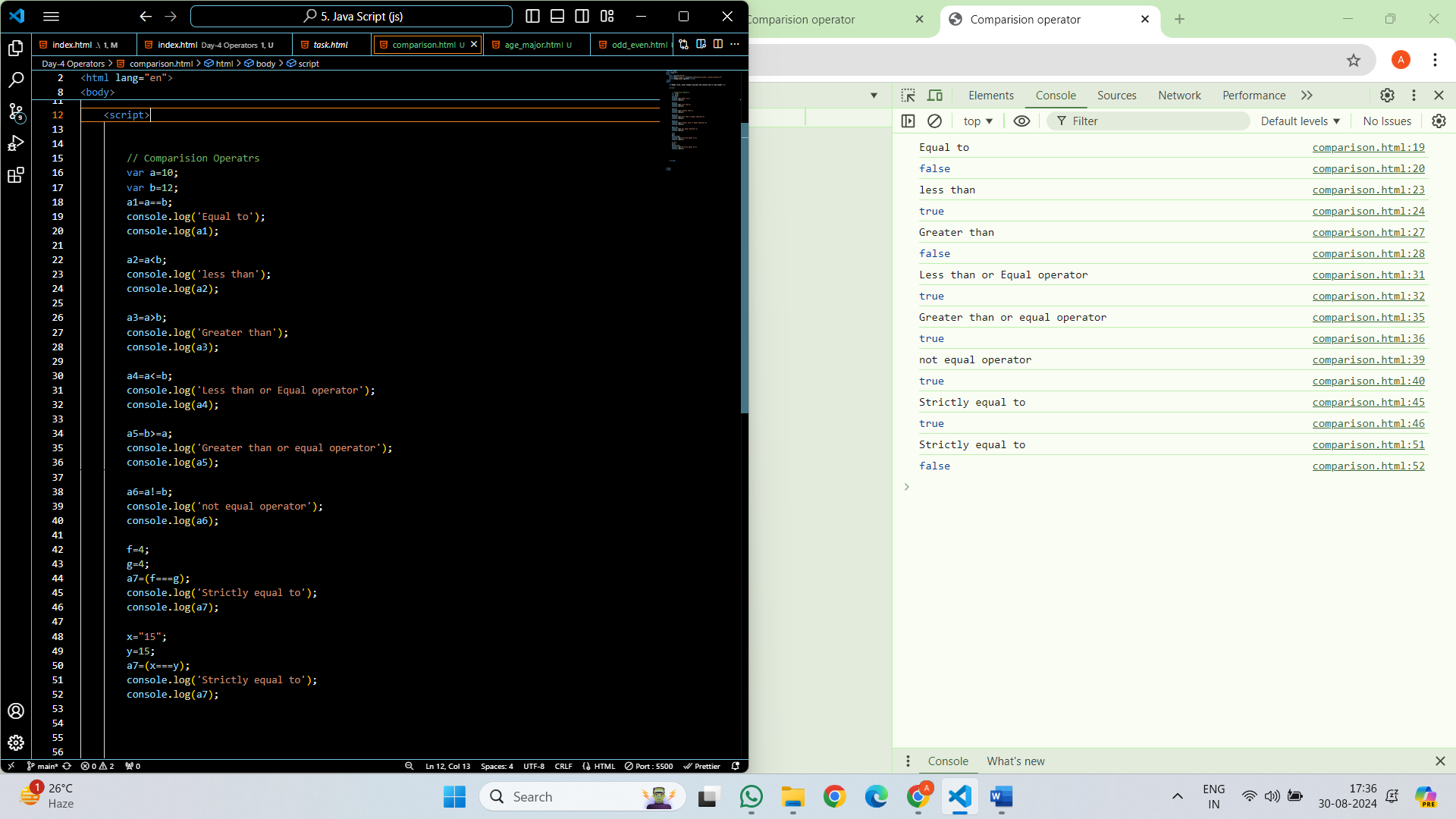
**Example:**



**Comparision Operators**

Comparison operators are used in logical statements to determine equality or difference between variables or values.



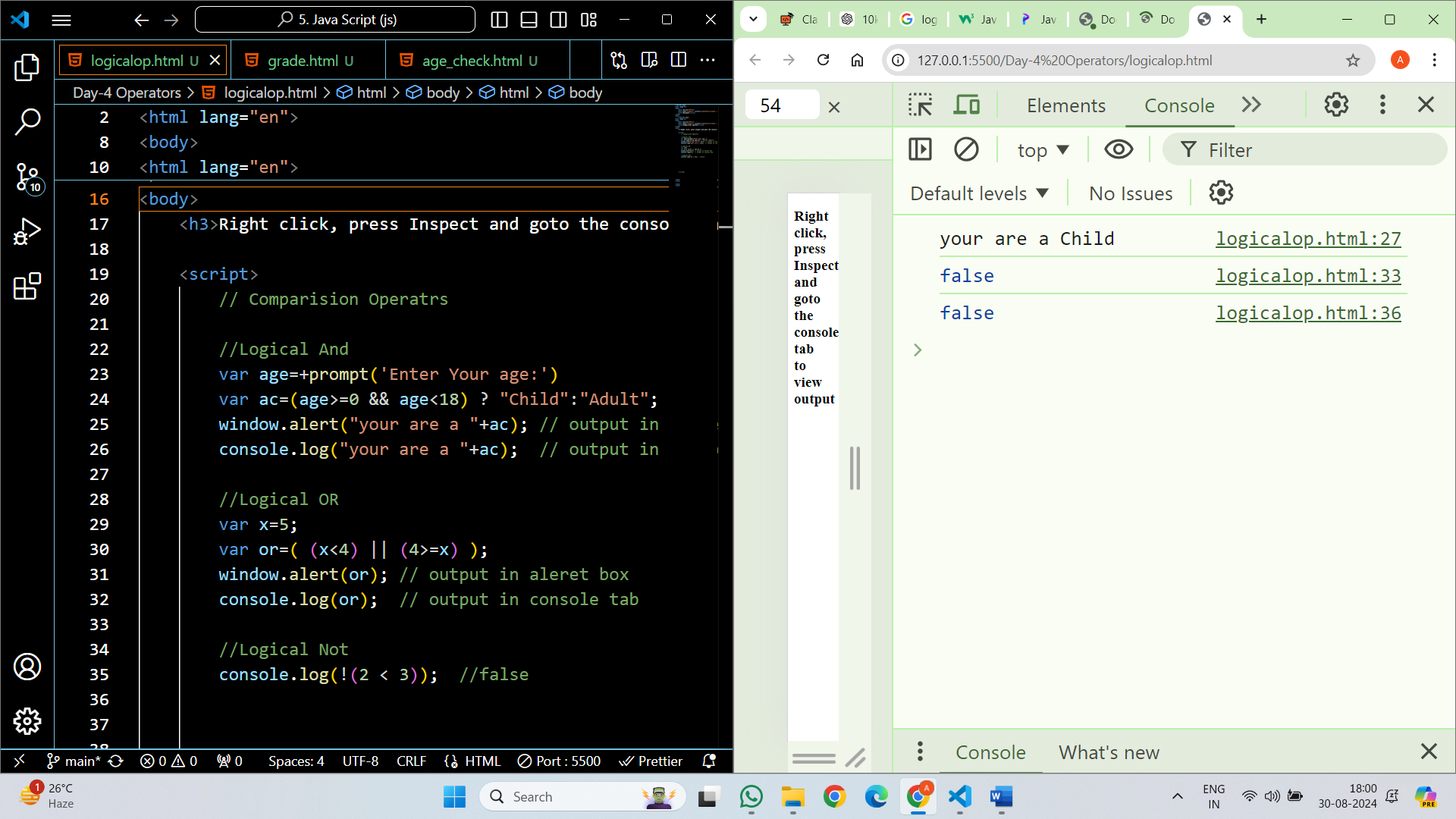


**Logical Operator:**

Logical operators return a boolean value by evaluating boolean expressions.

1. Logical And Operator: The **logical AND** operator **&&** returns true if both the expressions are true.
2. Logical OR Operator: The logical OR operator **||** returns true if at least one expression is true.
3. Logical Not Operator: The logical NOT operator **!** returns true if the specified expression is false and vice versa.





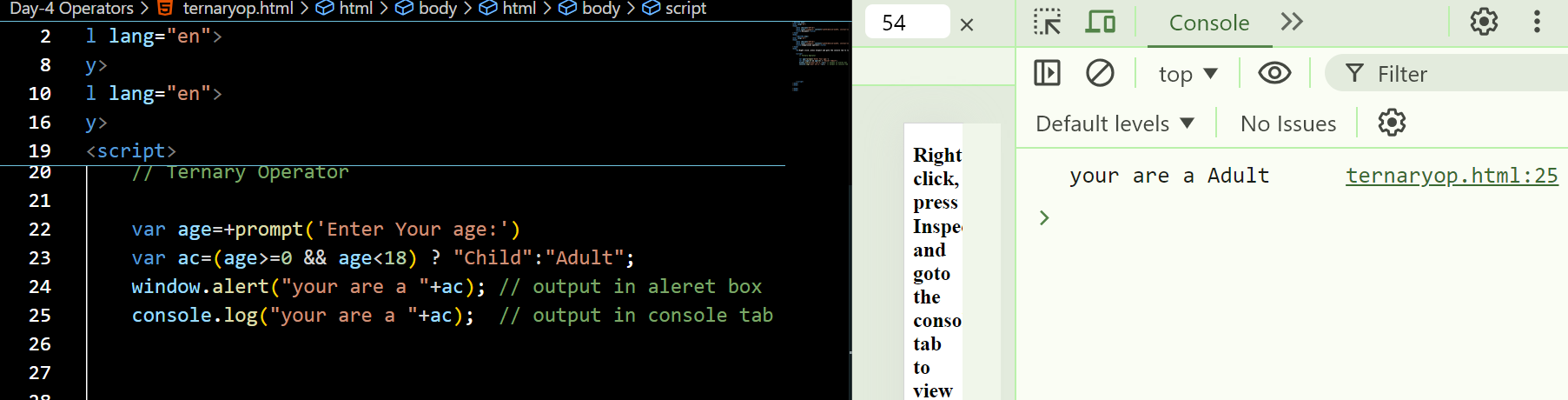
**Ternary operator:**

The Ternary Operator in JavaScript is a shortcut for writing simple if-else statements. It’s also known as the Conditional Operator because it works based on a condition. The ternary operator allows you to quickly decide between two values depending on whether a condition is true or false.

**Syntax:**

condition ? trueExpression : falseExpression

**Example:**



**Nullish coalescing operator (??)**

is a logical operator that returns its right-hand side operand when its left-hand side operand is null or undefined, and otherwise returns its left-hand side operand. It’s commonly used to provide default values for variables.

**Example:**

<script>

        //Nulish Coalescing Operator

         var a=null;

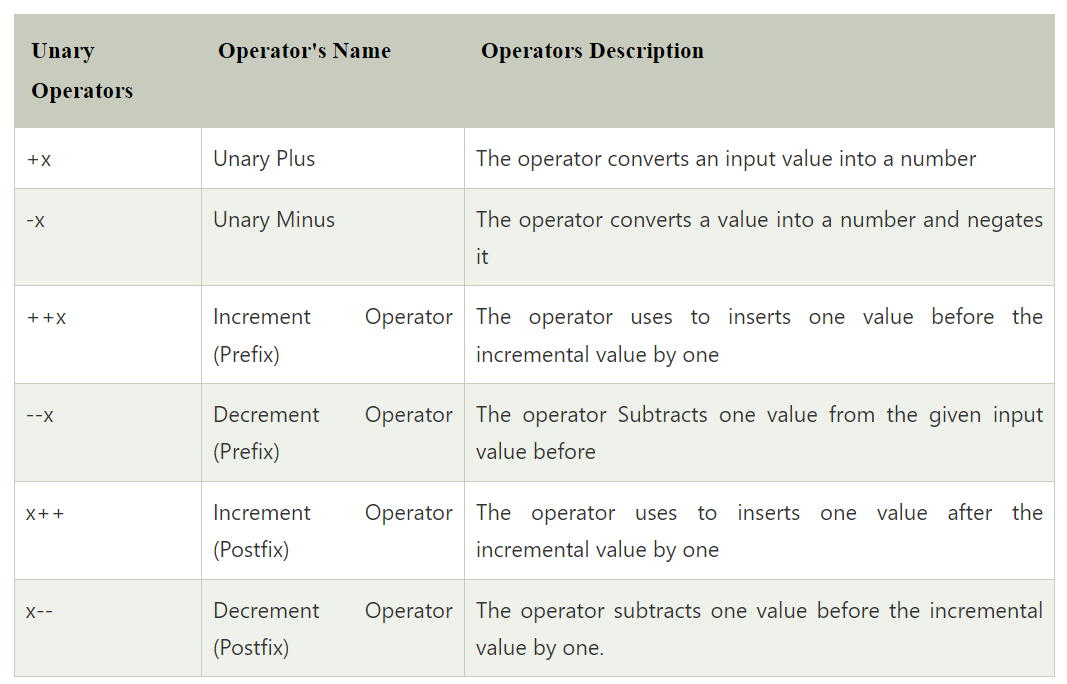
         var b=a ?? "Some Content";

         console.log(b); // Some Content

    </script>

**Unary Operator:**

* Unary operators in JavaScript are unique operators that consider a single input and carry out all possible operations.
* The Unary plus, unary minus, prefix increments, postfix increments, postfix decrements, and prefix decrements are examples of these operators. These operators are either put before or after the operand.
* The unary operators are more effective in executing functions than JavaScript; they are more popular. Unary operators are flexible and versatile since they cannot be overridden.



Example:

    <script>

            // Using unary plus to convert string to number

            let str1 = "12";

            let num = +str1;

            console.log(num);

            console.log(typeof (num))  // Here we are using typeof operator

            // "Abhinav" cannot be converted to a number

            let str2 = +"Abhinav";

            console.log(str2);

            console.log(typeof (str2))

            let s1='2'

            let n1 = -s1;

            console.log(n1);

            console.log(typeof (n1))

            let s2='3'

            let n2 = ++s2;

            console.log(n2);

            console.log(typeof (n2))

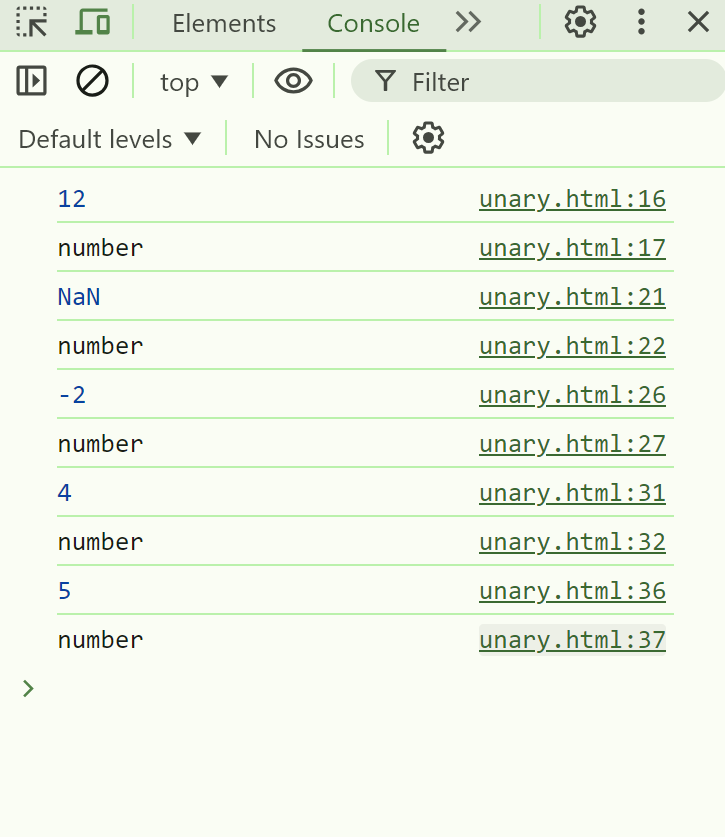
            let s3='5'

            let n3 = s3++;

            console.log(n3);

            console.log(typeof (n3))

    </script>



**Type Coercion**

Type coercion refers to the automatic or implicit conversion of values from one data type to another.

In programming, type conversion is the process of converting data of one [type](https://www.programiz.com/javascript/data-types) to another. For example, converting [string](https://www.programiz.com/javascript/string) data to [number](https://www.programiz.com/javascript/numbers).

There are two types of type conversion in JavaScript:

* **Implicit Conversion** - Automatic type conversion.
* **Explicit Conversion** - Manual type conversion.

**Explicit Type Conversion**

JavaScript type conversion, allowing you to convert values from one data type to another.

1.   **String()**: Converts a value to a string.

let num = 123;

let str = String(num);

console.log(str);

// Output: "123"

2.   **Number()**: Converts a value to a number.

let str = "123";

let num = Number(str);

console.log(num); // Output: 123

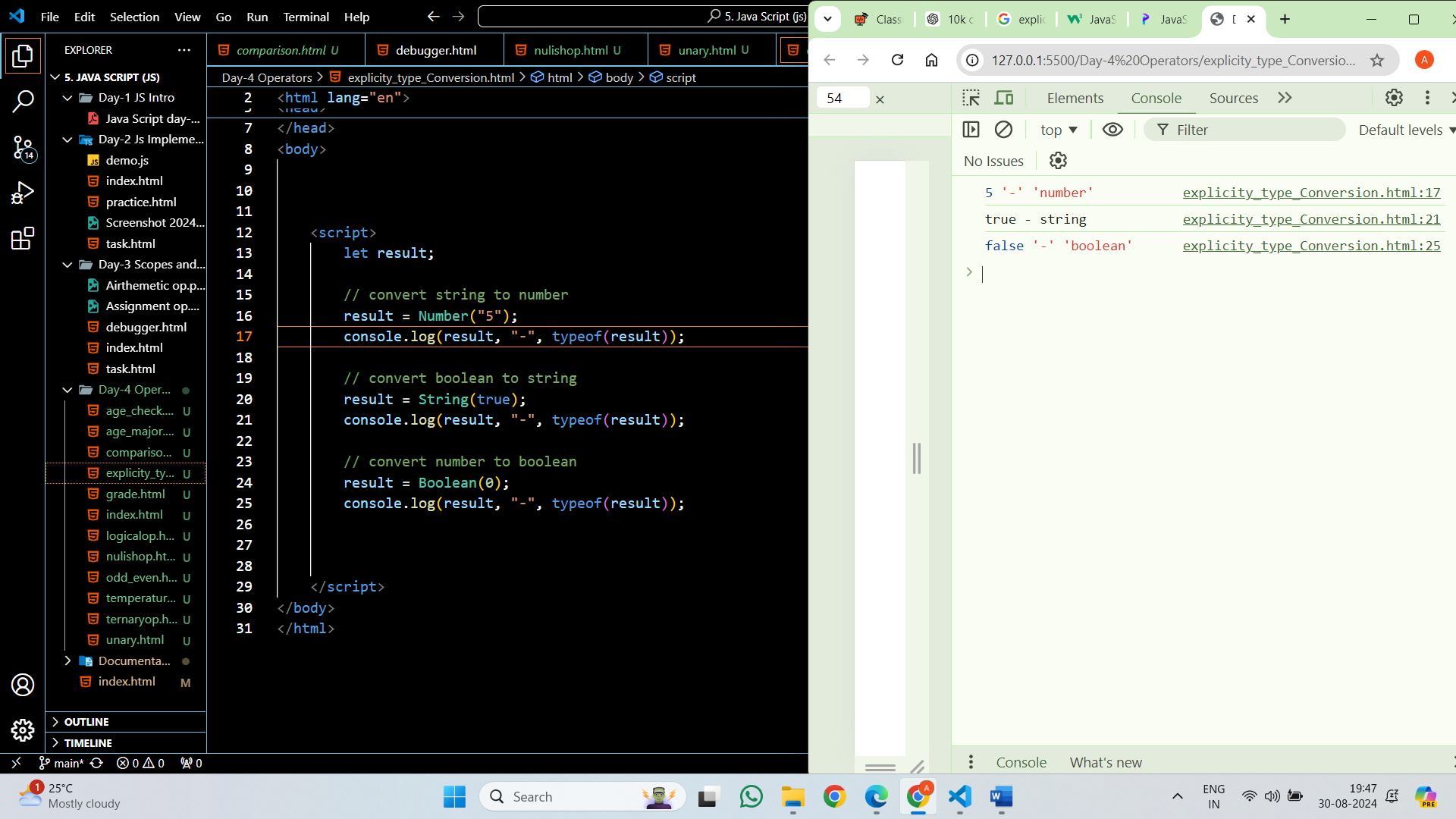
3. **Boolean()**: Converts a value to a boolean.

let num = 0;

let bool = Boolean(num);

console.log(bool); // Output: false

Example:



**How to take or get input from Users:**

Var a= +prompt(‘Enter Your Data’);

**In JavaScript, values are categorized as either "truthy" or "falsy"**

**Falsy Values:**

1.   **false**: The boolean value false itself.

2.   **0**: The number zero.

3.   **""**: Empty string.

4.   **null**: The absence of any value.

5.   **undefined**: A variable that has not been assigned a value or a property that does not exist.

6.   **NaN**: Not-a-Number.

**Truthy Values:**

1.   **true**: The boolean value true itself.

2.   **Non-zero numbers**: Any number other than 0 (including negative numbers and decimals).

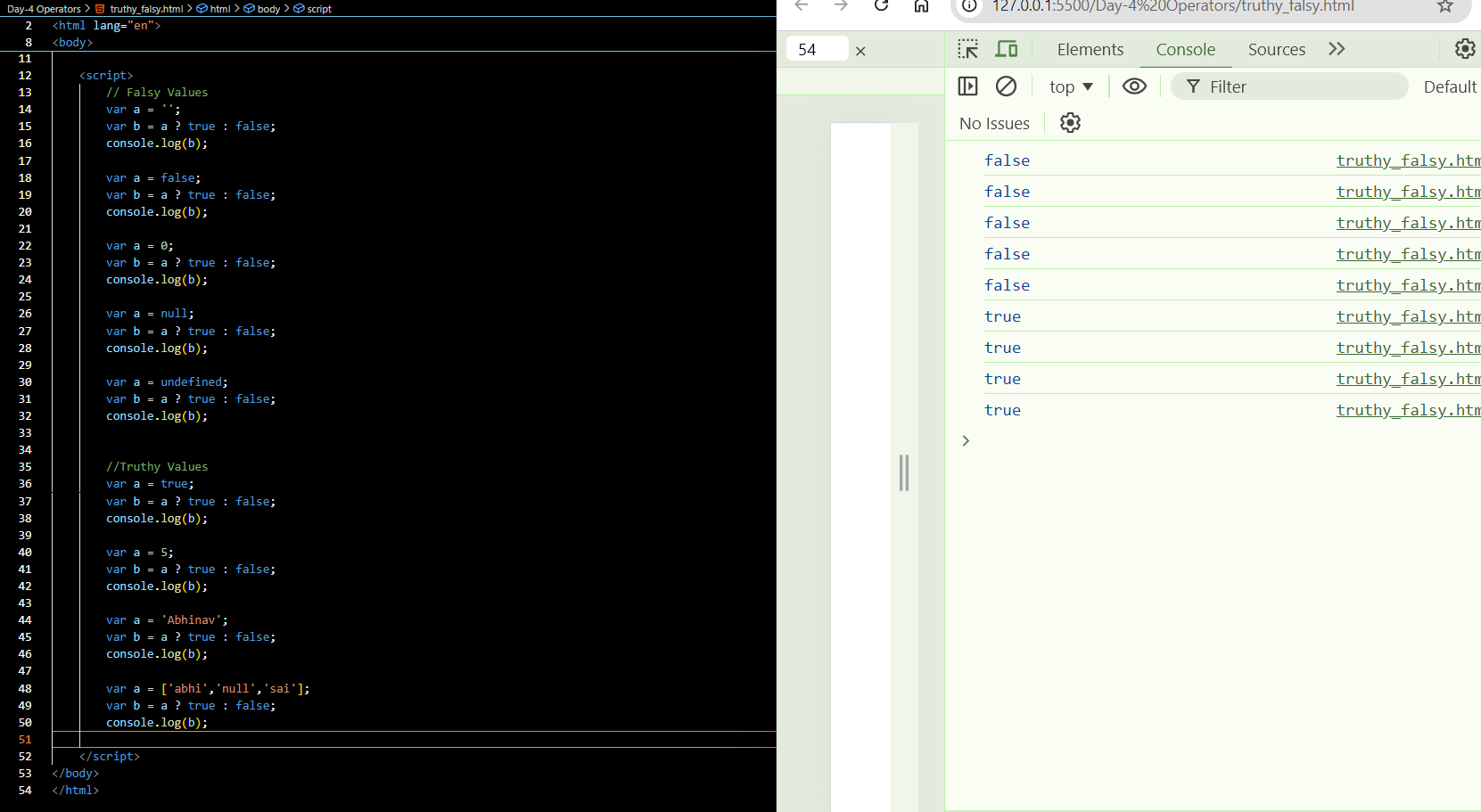
3.   **Non-empty strings**: Any string with at least one character.

4.   **Non-empty arrays**: Arrays with at least one element.

5.   **Objects**: Any object (including functions and arrays) is truthy, even if it's empty.

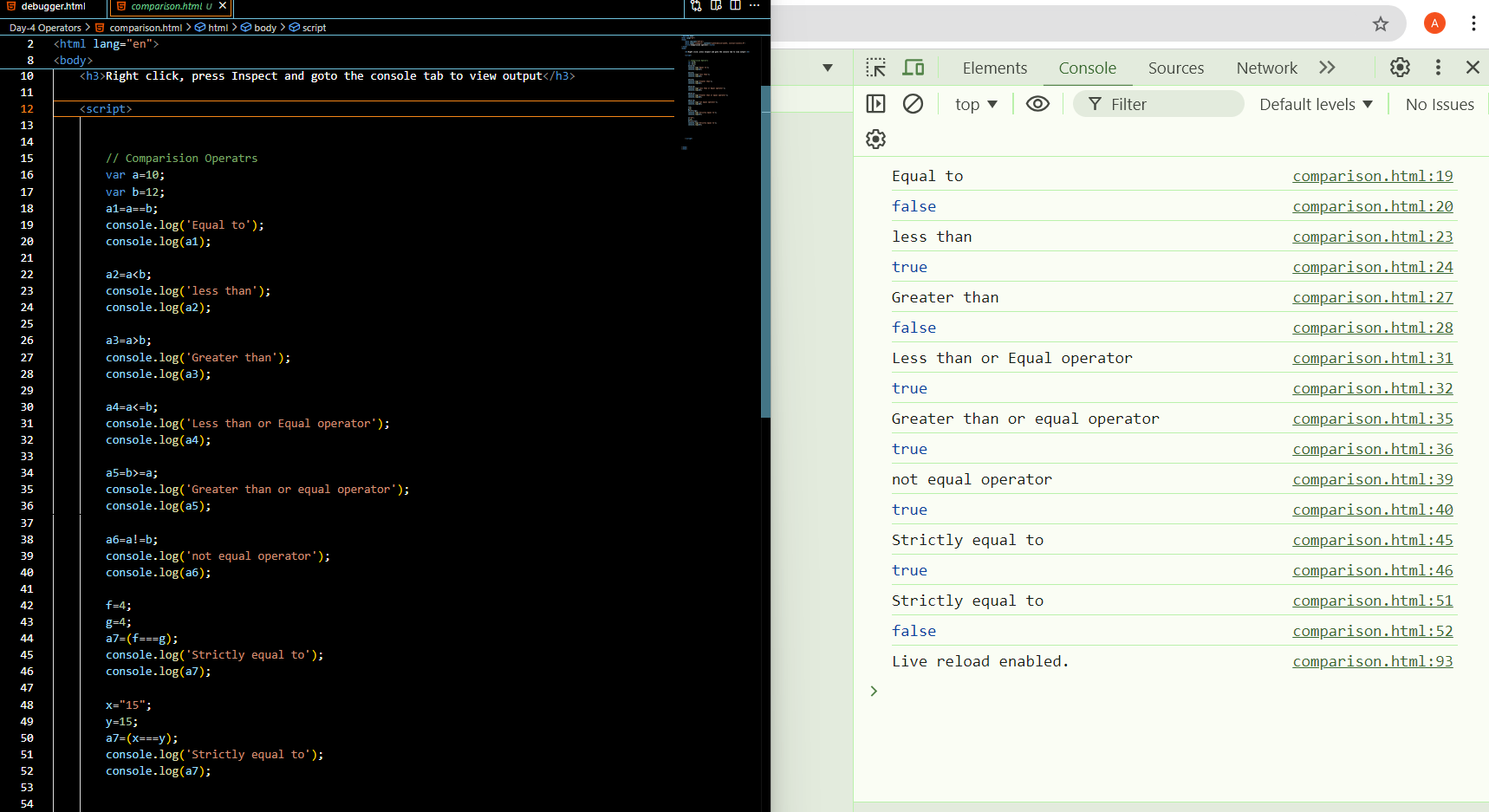
6.   **Functions**: Any function is truthy, even if it doesn't return anything.

**Check Truthy, Falsy values using ternary operator:**

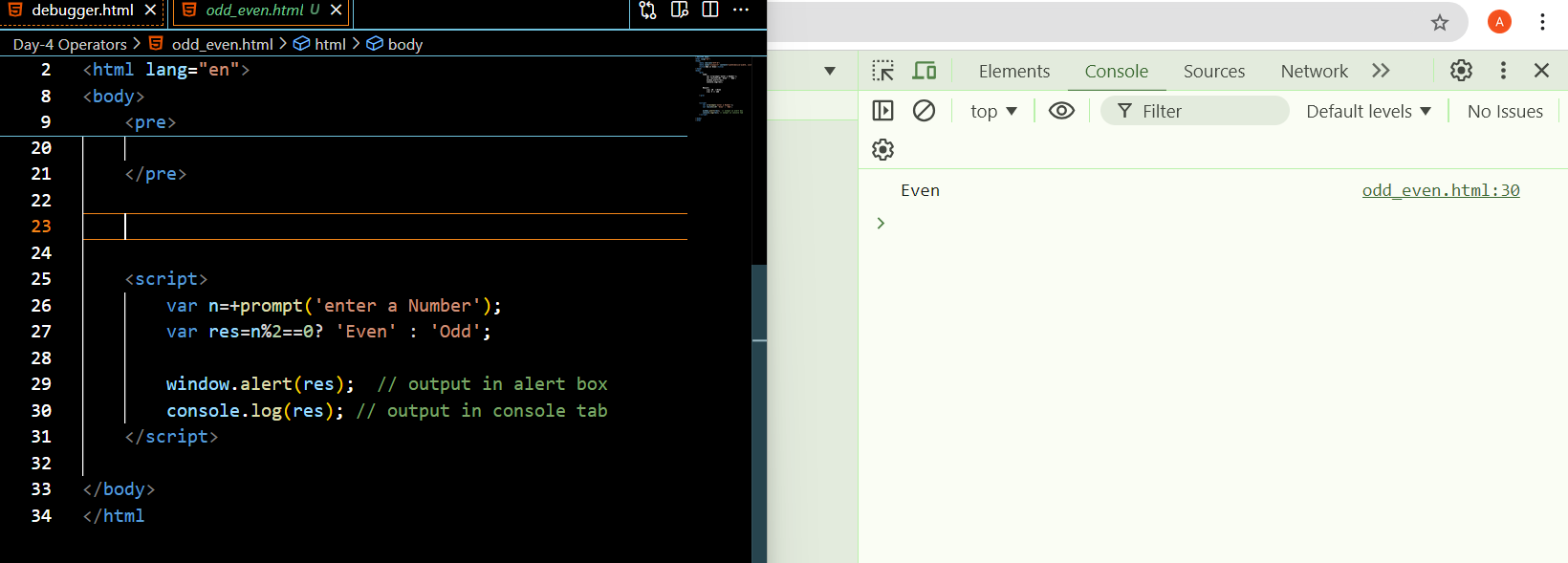


**Tasks:**

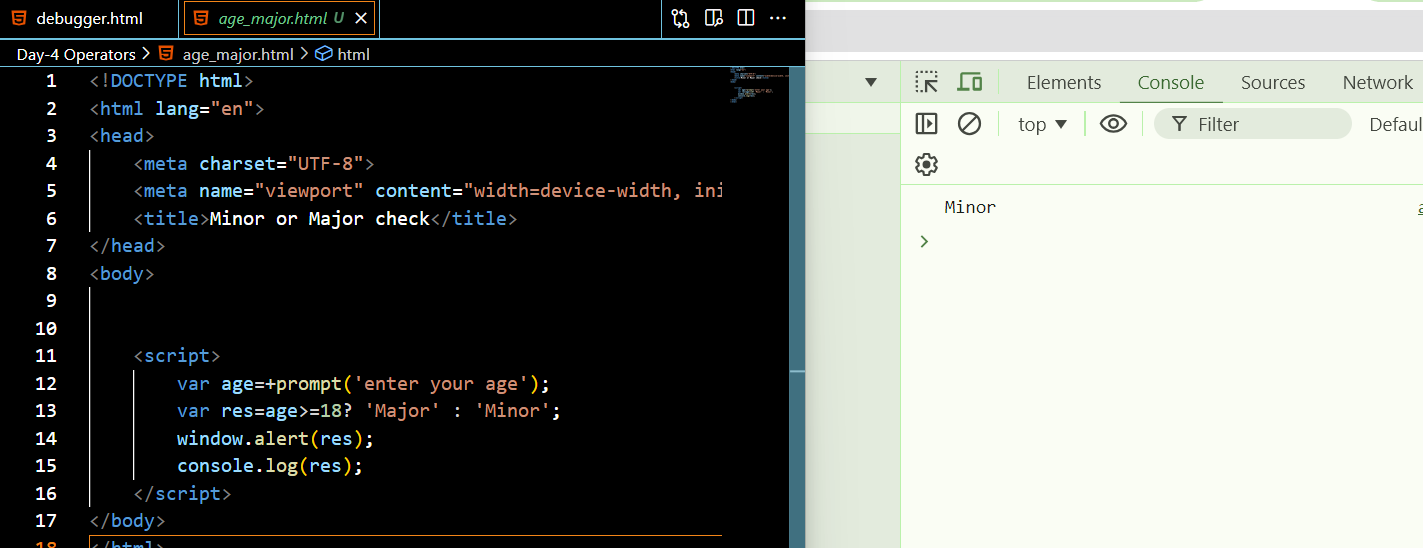
1. Write a JavaScript script that compares two variables using different comparison operators (==, ===, !=, !==, >, <, >=, <=) and prints the results.



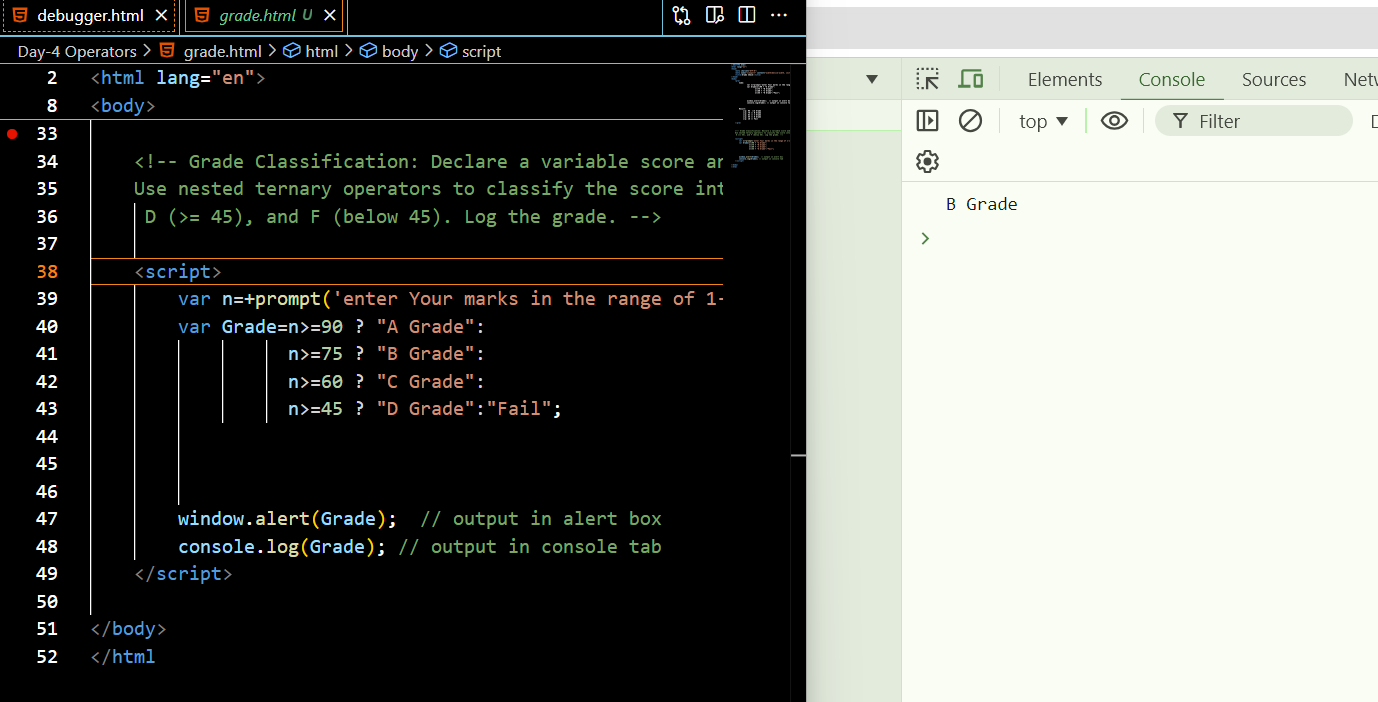
1. Write a JavaScript script that uses the ternary operator to determine if a number is even or odd.



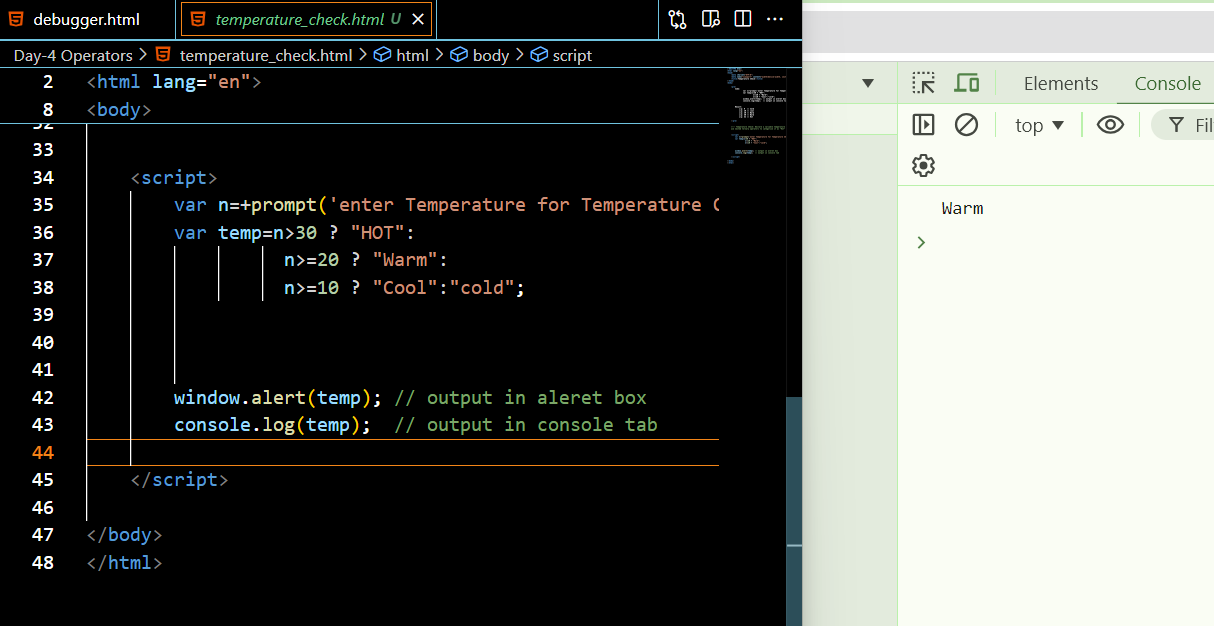
1. Expand the script to include a ternary operation that checks if a user is an adult (18+) or a minor.



1. Grade Classification: Declare a variable score and set it to a value between 0 and 100. Use nested ternary operators to classify the score into grades: A (>= 90), B (>= 75), C (>= 60), D (>= 45), and F (below 45). Log the grade.



1. Temperature Check: Declare a variable temperature and use nested ternary operators to categorize it as "Hot" (above 30), "Warm" (20-30), "Cool" (10-19), and "Cold" (below 10). Log the result.



1. Age Group: Declare a variable age and use the ternary operator to classify the age into "Child" (0-12), "Teen" (13-19), "Adult" (20-64), and "Senior" (65 and above). Log the result.

