***Module 17) Javascript For Full Stack Assignment***

**Theory assignment**

1. **JavaScript introduction**

**Q.1**: **What is JavaScript? Explain the role of JavaScript in web development**

**Ans: What is JavaScript? :**

* JavaScript is a light-weight object-oriented programming language which is used by several websites for scripting the webpages.
* It is an interpreted, full-fledged programming language that enables interactivity on websites when applied to an HTML document.
* It was introduced in the year 1995 for adding programs to the webpages in the Netscape Navigator browser.
* With JavaScript, users can build modern web applications to interact direct without reloading the page every time.
* The traditional websites user JavaScript to provide several forms to interactivity and simplicity.
* Although, JavaScript has no connectivity with Java programming language. In addition to web browsers, database such as CouchDB, MongoDB uses JavaScript as their scripting and query language.

**Role of JavaScript in web development:**

JavaScript plays a **crucial role** in web development as the primary language for adding interactivity and dynamic behavior to websites. Here's a breakdown of its main roles:

**1. Client-Side Scripting:** JavaScript runs in the user's browser, enabling real-time interaction without needing to reload the page. Common uses include:

* Form validation
* Dynamic content updates (like showing/hiding elements)
* Interactive UI features (dropdowns, sliders, modals)

**2. Creating Rich User Interfaces:** JavaScript powers interactive and responsive UIs through:

* **DOM manipulation** – changing HTML and CSS on the fly
* **Event handling** – responding to user actions (clicks, keypresses, etc.)
* Libraries like **jQuery**, and frameworks like **React**, **Vue**, or **Angular** take this further by providing structure and reusable components.

**3. Asynchronous Communication:** With technologies like:

* **AJAX**
* **Fetch API**
* **Axios**

JavaScript can make background requests to a server (like retrieving data or submitting forms) without reloading the page – a key aspect of **single-page applications (SPAs)**.

**4. Integration with Backend Services:** JavaScript, especially with **Node.js**, can also run on the server side. This enables full-stack JavaScript development:

* Backend logic
* Database interaction
* API creation and consumption

**5. Progressive Web Apps (PWAs):** JavaScript enables web apps to behave like native apps using:

* **Service workers** for offline capabilities
* **Push notifications**
* **App-like caching and loading**

**Q.2 How is JavaScript different from other programming languages like Python or Java?**

**Ans:** JavaScript, Python, and Java are all powerful languages, but they serve different purposes and have distinct characteristics. Here's a breakdown of how **JavaScript** differs from **Python** and **Java**:

### ****Primary Use and Environment****

|  |  |  |  |
| --- | --- | --- | --- |
| Language | **Typical Use Case** | | **Runs On** | | --- | |
| JavaScript | Web development (front-end and back-end) | Browser (client-side), Node.js (server-side) |
| Python | General-purpose: scripting, AI, web, data science | Desktop/server (not in browser) |
| Java | |  | | --- | |  |  |  | | --- | | Enterprise apps, Android, backend systems | | Java Virtual Machine (JVM) |

### JavaScript is built for web interactivity, while Python is more of a general-purpose scripting language, and Java is a strongly typed language often used in large-scale systems.

**2. Syntax and Ease of Use**

* **JavaScript** has a syntax closer to C-style languages, can be a bit quirky, and has concepts like **prototypes** and **hoisting**.
* **Python** is known for its **clean, readable syntax** (indentation-based).
* **Java** is verbose with strict **object-oriented** structure and requires everything to be inside a class.

**3.Typing System**

|  |  |  |
| --- | --- | --- |
| **Language** | **Typing** | **Example** |
| JavaScript | Dynamically typed | let x=10,  x= ”hello” (allowed) |
| Python | Dynamically typed | x=5, x=”hii”  (allowed) |
| Java | Static typed | int x=5, x=”hii”  (error) |

* **Java** enforces type safety, which can prevent bugs but adds boilerplate. **JavaScript** and **Python** are more flexible but can lead to runtime errors.

### 4.Concurrency & Multithreading

* **JavaScript** uses a **single-threaded** event loop (non-blocking I/O, async/await).
* **Python** supports threads but has limitations (like the GIL).
* **Java** has robust multithreading support built into the language.

For real-time user interaction, JavaScript's event-driven model shines. For heavy parallel computing, **Java** often performs better.

**5. Speed & Performance**

* **Java** is fastest due to compiled bytecode and optimizations in the JVM.
* **JavaScript** is fast in the browser (thanks to engines like V8) but slower than Java.
* **Python** is typically slower, as it’s interpreted and prioritizes readability over speed.

**Q.3: Discuss the use of <script> tag in HTML. How can you link an external JavaScript file to an HTML document?**

**Ans:** The **HTML <script> tag** embeds client-side scripts or links to external JavaScript files, enabling dynamic content, form validation, and style manipulation. Attributes like async, defer, and src control script execution and loading, enhancing the interactivity and performance of web pages.

**Syntax:**

* **For Internal JavaScript linking**

<script> content </script>

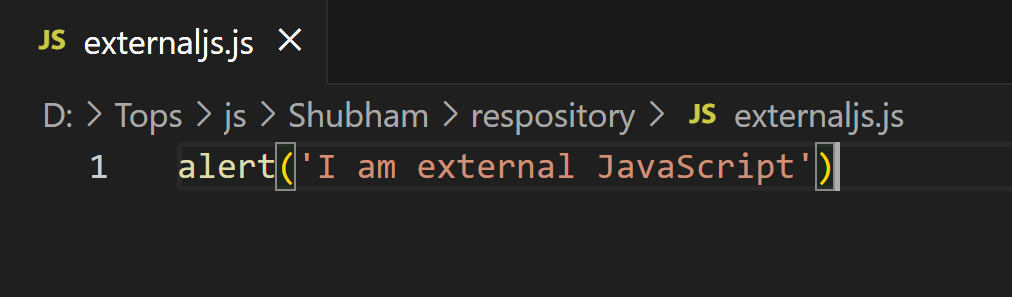
* **For External JavaScript linking**

<script src=”script.js”></script>

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| Src | It is used to specify the URL of an external script file. |
| type | It is used to specify the media type of the script. |
| async | It is used to specify the script is executed asynchronously. |
| cross-origin | It is used for loading an external script into their domain from a third-party server or another domain with the support of HTTP CORS Request. |
| defer | It is used to specify that the script is executed when the page has finished parsing. |
| integrity | It is used to give permission to the Browser to check the fetched script to make ensure the source code is never loaded. |
| referrerpolicy | It is used to specify the reference information that will be sent to the server when fetching the script. |

### How can you link an external JavaScript file to an HTML document?

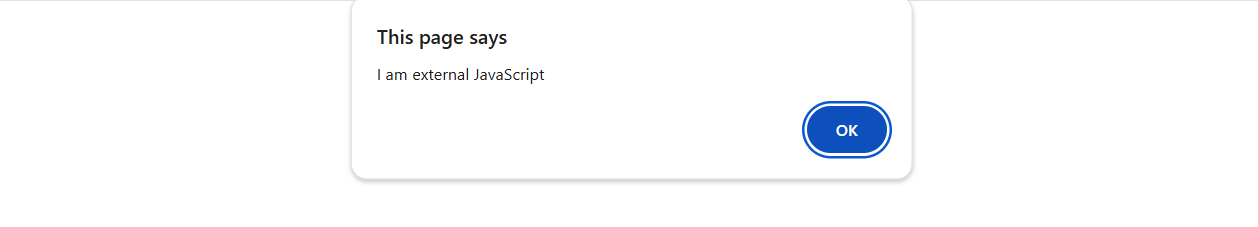
### Step 1: create JavaScript file with .js extension

****

**Step 2: link JavaScript file in HTML page**

****

**Output: when run HTML file in browser**

****

**2.Variables and Data Types**

**Q.1 What are variables in JavaScript? How do you declare a variable using var, let, and const?**

**Ans:** A JavaScript variable is simply a name of storage location. There are two types of variables in JavaScript: Local variable and Global variable.

* **Local variable:** Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.

Local variables are created when a function starts, and deleted when the function is completed.

* **Global variable:** Variables declared **Globally** (outside any function) have **Global Scope**.

**Global** variables can be accessed from anywhere in a JavaScript program.

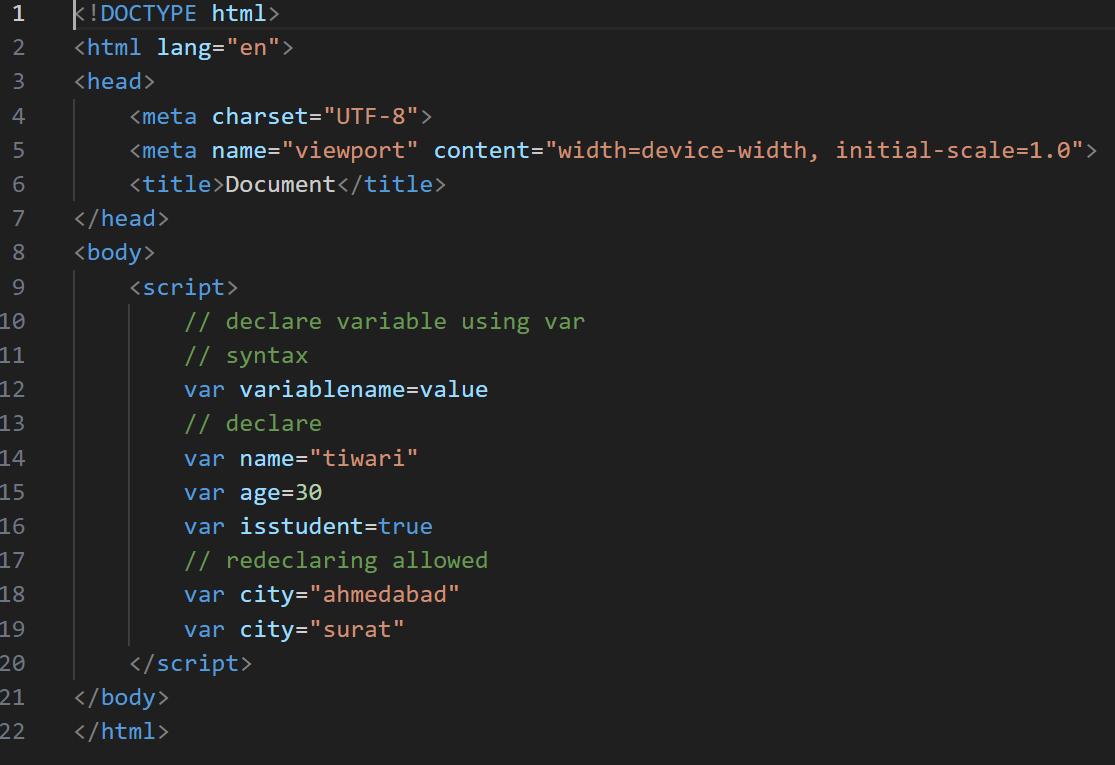
**Rules for declaring variables in JavaScript:**

* Must start with: letter (a-zA-Z), underscore (\_), dollar sign ($)
* Cannot be use Reserved keyword

**How do you declare a variable using var, let, and const?**

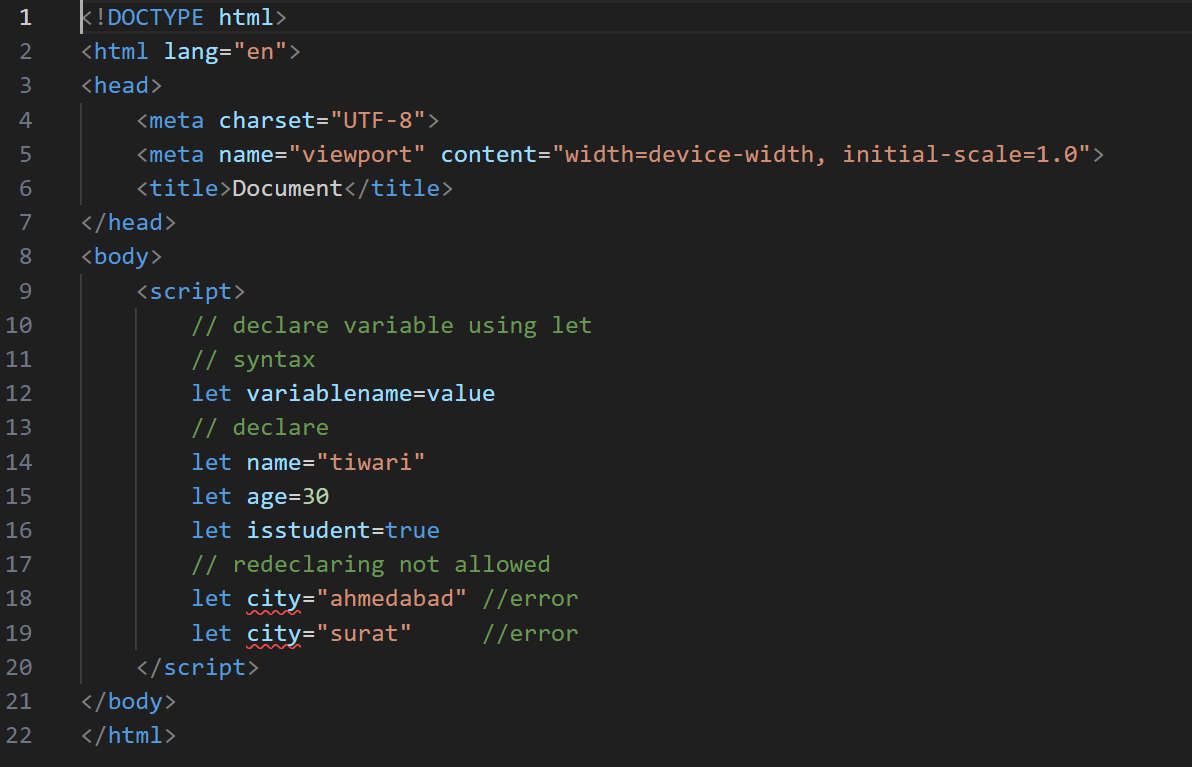
* **Using var:**
* Function-scoped, not block-scoped (unlike let or const)
* Can be re-declared and updated in the same scope
* Can lead to unexpected behavior due to hoisting

**Example:**



* **Using let:**
* It allows you to declare **block-scoped** variables (unlike var which is function-scoped).
* You **can** change the value of a let variable later:
* Not allowed redeclaration

Example:



* **Using const:**

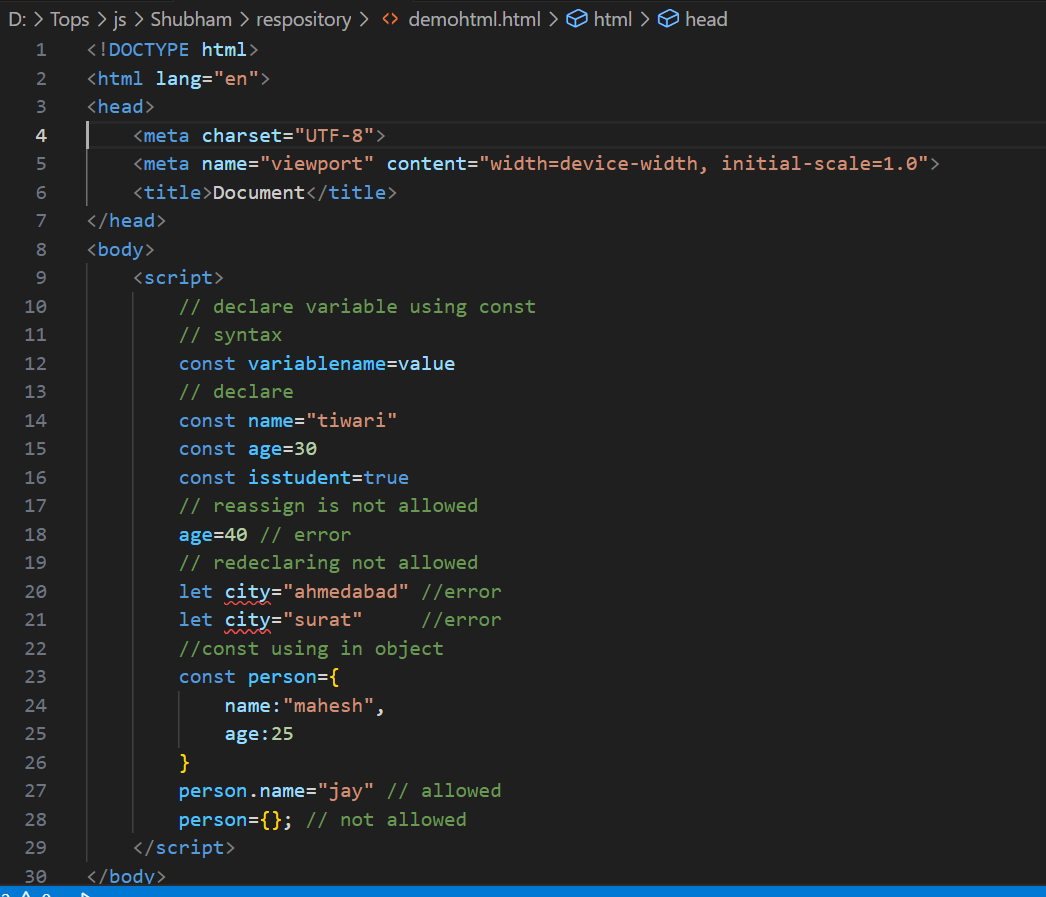
**Key points:**

* You **must** assign a value when declaring a const — you can't leave it undefined:

**Ex: const x; // error**

* const creates a block-scoped variable, just like let.
* You can't reassign a const variable:
* **However,** if the const is an object or array, you **can** modify the contents (just not reassign the variable)

Example:



**Q.2 How is JavaScript different from other programming languages like Python or Java?**

**Ans:**

**1.Syntax simplicity**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **JavaScript** | **Python** | **Java** |
| Semicolon | Optional | Not used | Required |
| Braces | Used for blocks | Uses indentation | Used for block |
| Variable Declaration | Let, const, var | No keywords needed | Must define types |

**2.Typing System**

|  |  |  |
| --- | --- | --- |
| **Language** | **Typing** | **Example** |
| **JavaScript** | **Dynamically typed** | **let x=10; x=”hii”** |
| **Python** | **Dynamically typed** | **x=10; x=”hii”** |
| **Java** | **Statically typed** | **int x=10;** |

**3.Compilation vs Interpretation**

|  |  |
| --- | --- |
| **Language** | **How it’s run** |
| JavaScript | Interpreted in browsers (or Node.js) |
| Python | Interpreted (via Cpython etc) |
| Java | Compiled to bycode, runs on JVM |

**4. Usage & Environment**

|  |  |
| --- | --- |
| **Language** | **Typically used for** |
| JavaScript | Web development (front-end and back-end) |
| Python | Data science, scripting, automation |
| Java | Enterprise apps, Android development |

### 5. ****Object-Oriented Programming****

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **JavaScript** | **Python** | **Java** |
| **classes** | ES6 + supports classes | Fully supports OOP | Strongly OOP |
| **Prototypes** | Uses prototypes under the hood | Not used | Class-based only |

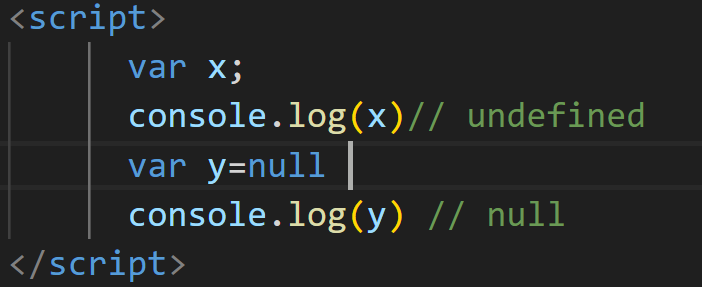
### 6. Asynchronous Programming

* **JavaScript**: Uses callbacks, promises, and async/await — very common.
* **Python**: Supports asyncio for async tasks.
* **Java**: Has threading and CompletableFuture for async tasks.

**Q.3 What is the difference between undefined and null in JavaScript?**

**Ans:**

* **Undefined**
* **Meaning**: A variable has been declared but has not been assigned a value.
* **Set by**: JavaScript automatically
* **Type**: undefined
* **Null**
* **Meaning**: A variable is explicitly set to "no value" or "empty".
* **Set by**: the developer
* **Type**: object (weird JavaScript quirk!)
* **Example:**

****

**3. JavaScript Operators**

**Q.1: What are the different types of operators in JavaScript? Explain with examples.**

**o Arithmetic operators**

**o Assignment operators**

**o Comparison operators**

**o Logical operators**

**Ans: 1. Arithmetic operator**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| **+** | **Addition** | **5+2=7** |
| **-** | **Subtraction** | **5-2=3** |
| **\*** | **Multiplication** | **5\*2=10** |
| **/** | **Division** | **10/2=5** |
| **%** | **Mudule (remainder)** | **5%2=1** |
| **\*\*** | **Exponentiation** | **2\*\*3=8** |
| **++** | **Increment** | **var a=10**  **a++ (post- increment)**  **++a (pre- increment)** |
| **--** | **Decrement** | **var a=10**  **a-- (post- Decrement)**  **--a (pre- Decrement)** |

1. **Assignment Operator**

**Used for assign values to variable**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same as** |
| = | var a=10 | assign value |
| += | a+=2 | a=a+2 |
| -+ | a-=2 | a=a-2 |
| \*= | a\*=2 | a=a\*2 |
| /= | a/=2 | a=a/2 |
| %= | a%=2 | a=a%2 |

1. **Comparison Operator**

**Used to compare two values**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| **==** | **Equal (loose)** | **5==”5” (return true)** |
| **===** | **Equal (strict)** | **5===”5” (return false)** |
| **!=** | **Not equal** | **5!=”5” (return false)** |
| **!==** | **Not equal (strict)** | **5!==”5” (return true)** |
| **>** | **Greater than** | **5>3** |
| **<** | **Less than** | **5<3** |
| **>=** | **Greater or Equal** | **5>=5** |
| **<=** | **Less or Equal** | **3<=5** |

1. **Logical operator**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| && | Logical AND  he && operator returns true if both expressions are true, otherwise it returns false. | (6 < 10) && (5>1) (is true) |
| || | Logical OR  The || operator returns true if one or both expressions are true, otherwise it returns false. | (15 < 10) || (5>1)  is true |
| ! | Logical NOT  The NOT operator (!) returns true for false statements and false for true statements. | !(5==10) is true |

1. **Conditional(Ternary) operator:**

Syntax: variablename=condition? value1: value2

Example:

let age=10;

let result=(age < 18)?”young”: ”old enough”;

above statement return young.

1. **String operator:**

+ is also used to concatenate strings

Example:

let str=”welcome to”+”Mumbai”;

**7.Bitwise operator**

Bit operators work on 32 bits numbers.

Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Same as** | **Result** | **Decimal** |
| & | AND | 5&1 | 0101 & 0001 | 0001 | 1 |
| | | OR | 5|1 | 0101 | 0001 | 0101 | 5 |
| ~ | NOT | ~5 | ~0101 | 1010 | 10 |
| ^ | XOR | 5^1 | 0101 ^ 0001 | 0100 | 4 |
| << | Left shift | 5<<1 | 0101 << 1 | 1010 | 10 |
| >> | Right shift | 5>>1 | 0101 >> 1 | 0010 | 2 |
| >>> | Unsigned right shift | 5>>>1 | 0101 >>> 1 | 0010 | 2 |

**8.Type Operator**

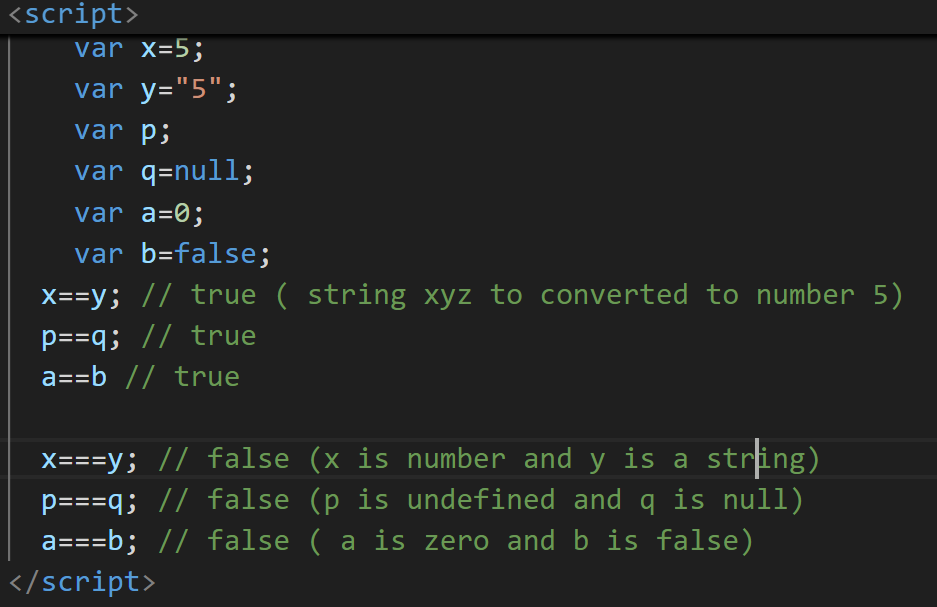
|  |  |
| --- | --- |
| **Operator** | **Description** |
| typeof | Returns the type of variable |
| instanceof | Returns true if an object is an instance of an object type |

**Q.2 What is the difference between == and === in JavaScript?**

**Ans:**

* **== (double equal):**
* Compares two values after type conversion.
* It will try to convert different data types to be the same before comparing.
* **=== (Triple Equals) — *Strict Equality***
  + Compares both value and type.
  + No type conversion. Values must be exactly the same.

**Example:**

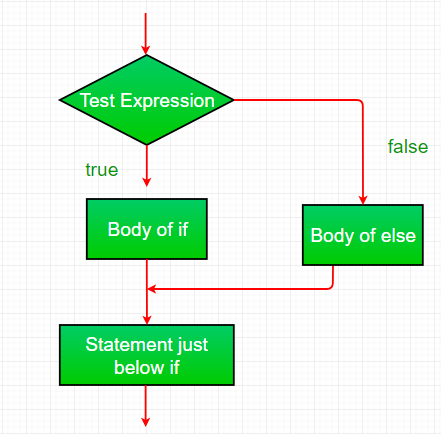


**4. Control flow (if-else, switch)**

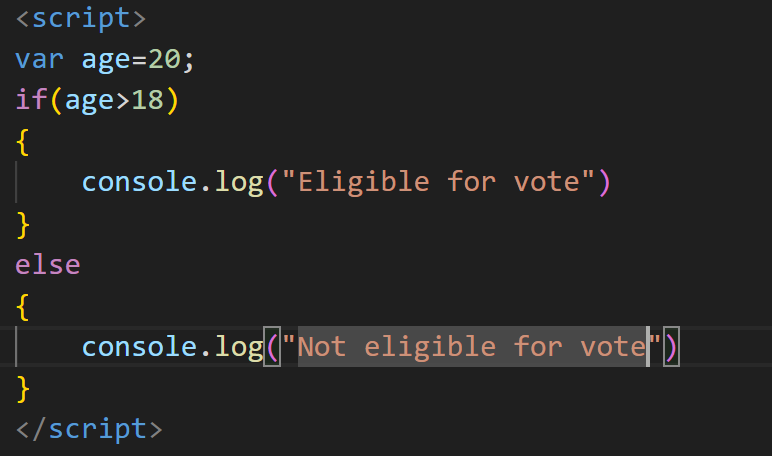
**Q.1: What is control flow in JavaScript? Explain how if-else statements work with an example.**

**Ans**: JavaScript **control statement** is used to control the execution of a program based on a specific condition. If the condition meets then a particular block of action will be executed otherwise it will execute another block of action that satisfies that particular condition.

**If-else statement:**

****

**Example:**

****

**Q.2: Describe how switch statements work in JavaScript. When should you use a switch statement instead of if-else?**

**Ans:** A switch statement is used to **perform different actions based on different values** of a variable or expression — like a cleaner alternative to multiple if...else if blocks.

**Syntax:**

Switch(expression){

case value1: // code of block for value1

break;

case value2: // code of block for value2

break;

default : // Code block if no case matches

}

**How switch statement work:**

1. JavaScript evaluates the expression.
2. It compares the result to each case value using === (strict equality).
3. If a match is found, it runs the matched block.
4. The break statement is used to exit the switch after executing a case.
5. If no match is found, the default block (optional) runs.

**When should you use a switch statement instead of if-else**?

**Use switch when:**

* You're checking **one variable against multiple exact values**

Best for **menu options, days of the week, user roles,** etc.

* You want **cleaner code than many** else if **block**

if...else if...else gets messy with lots of conditions.

switch looks cleaner and is easier to scan.

Don’t use switch when:

* + You need to check **complex conditions or ranges**
    - switch only checks **strict equality (**===**)**, so it won't handle:

**5. Loops (For, While, Do-While)**

**Q.1: Explain the different types of loops in JavaScript (for, while, do-while). Provide a basic example of each.**

**Ans:**

* **for loop:**

Used when you know **how many times** you want to run the loop.

**Syntax:**

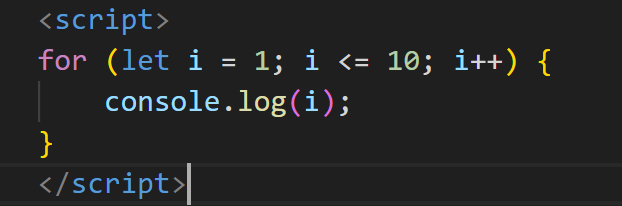
for(initialization; condition; increment/decrement)

{

// statements

}

**Example:**

****

* **while :**

Used when the number of repetitions is **not known in advance** — runs **as long as the condition is true.** It is also known as entry control loop.

**Syntax:**

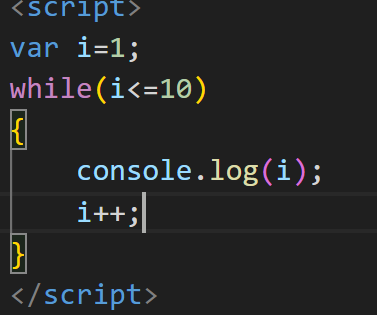
while(condition)

{

// code to run

}

**Example:**

****

* **Do-while:**

Like a while loop, but it **runs the code block at least once**, even if the condition is false. It is also known as exit control loop.

**Syntax:**

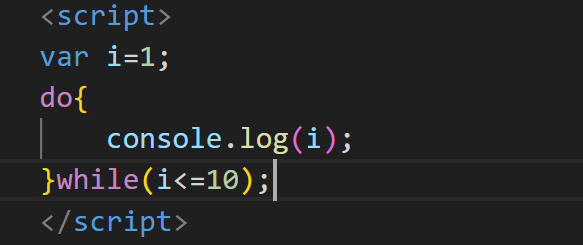
do

{

// code to run

}while(condition);

**Example:**

****

**Q.2: What is the difference between a while loop and a do-while loop?**

**Ans:**

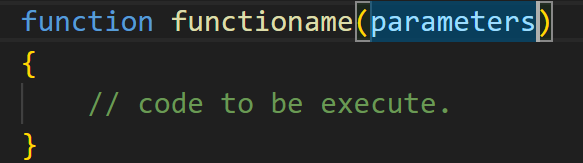
|  |  |  |
| --- | --- | --- |
| **Feature** | **While loop** | **do-while loop** |
| Syntax | While(condition){  //code  } | do{  //code  }while(condition); |
| First execution | Condition is checked before the loop block is executed. | Loop block is executed at least once before checking the condition. |
| Use cases | Suitable when the loop block should be executed only if the condition is initially true. | Useful when the loop block must be executed at least once, regardless of the initial condition. |
| **Initialization and Update** | Initialization and update need to be handled explicitly before and inside the loop. | Initialization (if needed) is done before the loop; update (if needed) is placed inside the loop. |

**6.Functions**

**Q.1: What are functions in JavaScript? Explain the syntax for declaring and calling a function.**

**Ans:** A function is a block of reusable code designed to perform a specific task. Instead of writing the same code again and again, you **define** a function once and call it whenever needed.

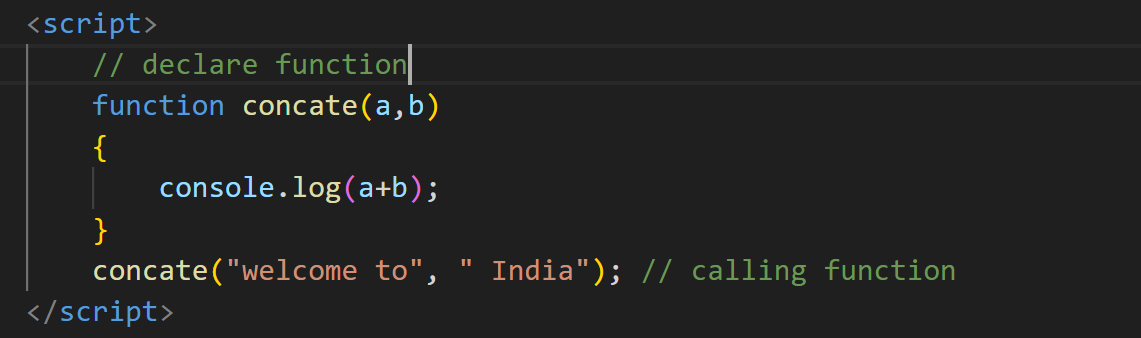
**Syntax for declaring function:**

****

**Syntax for Calling function:**

functionname(arguments);

**Example:**



**Q.2: What is the difference between a function declaration and a function expression?**

**Ans:**

|  |  |
| --- | --- |
| **Function declaration** | **Function expression** |
| A function declaration must have a function name. | A function expression is similar to a function declaration without the function name. |
| Function declaration does not require a variable assignment. | Function expressions can be stored in a variable assignment. |
| These are executed before any other code. | Function expressions load and execute only when the program interpreter reaches the line of code**.** |
| The function in function declaration can be accessed before and after the function definition. | The function in function expression can be accessed only after the function definition. |
| Function declarations are hoisted | Function expressions are not hoisted |
| **Syntax:** function functionname(paramA, paramB) { // Set of statements } | **Syntax:** var a= function functionname(paramA, paramB) { // Set of statements } |

**Q.3:** **Discuss the concept of parameters and return values in functions.**

**Ans:**

**1.function parameters**

Parameters are **placeholders** for values a function expects when it is called.  
They allow functions to be **dynamic** and **reusable**.

Syntax:

function greet(name) {

console.log("Hello, " + name + "!");

}

Here, name is a **parameter**.

**Calling the Function:**

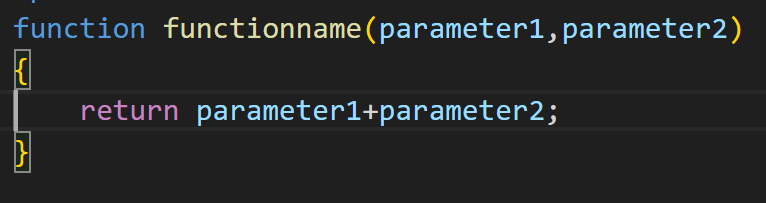
greet("Ayan"); // Output: Hello, Ayan!

Here, "Ayan" is the **argument** passed to the name parameter.

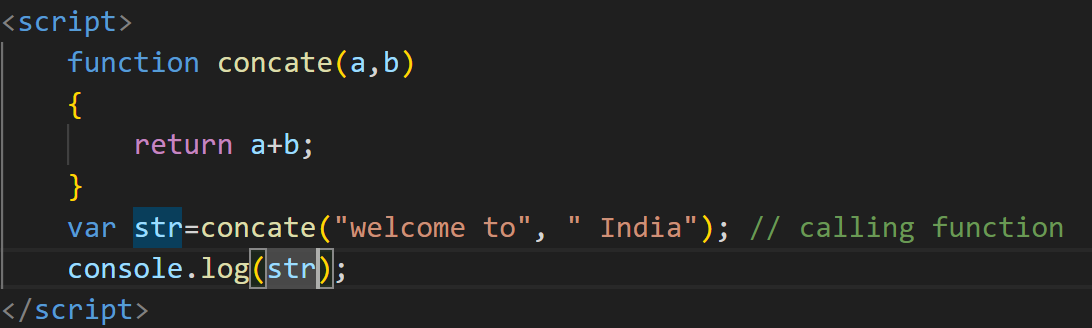
**2.Return values**

Functions can **return a result** back to the place where they were called using the return statement.

**Syntax:**

****

**Example:**

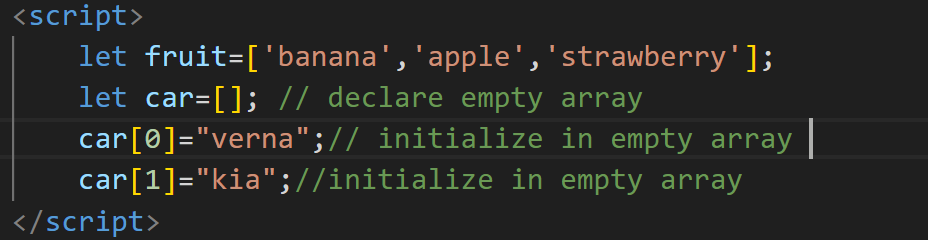
****

**7.Array**

**Q.1:** **What is an array in JavaScript? How do you declare and initialize an array?**

**Ans:** An **array** is a **special type of object** used to store **multiple values** in a single variable.  
Each value in an array is called an **element**, and each element has a **numeric index** starting from 0.

**Declaring and initialize an array:**

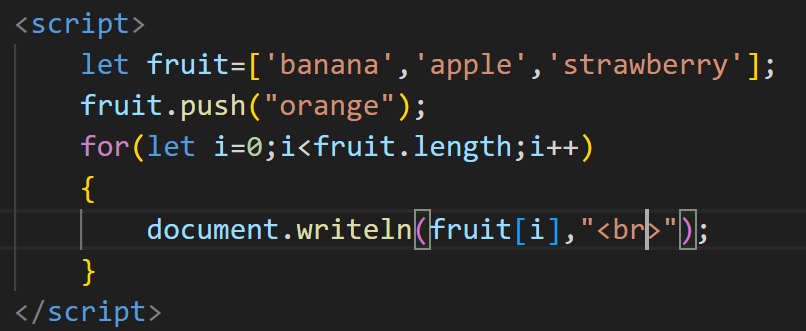
****

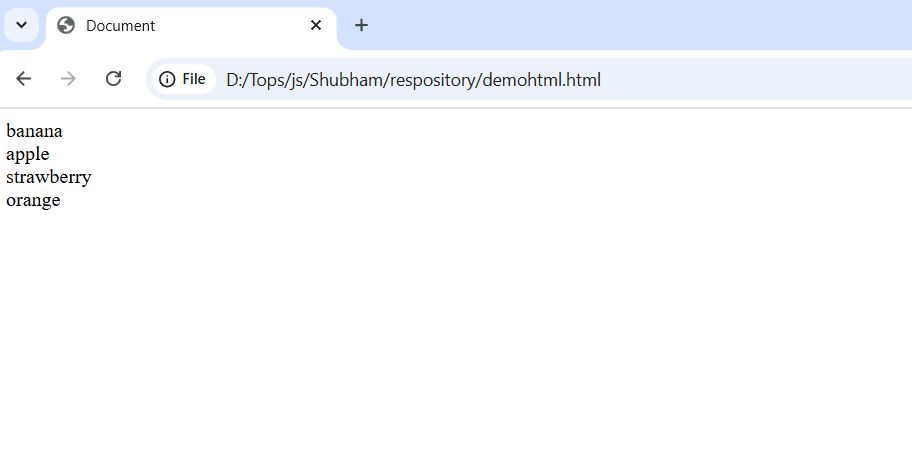
**Q.2: Explain the methods push(), pop(), shift(), and unshift() used in arrays.**

**Ans:1.push() :**

Adds **one or more elements** to the **end** of an array.

Example:

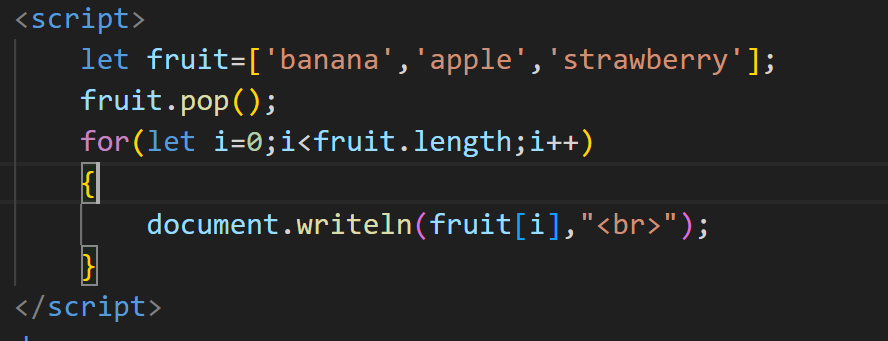
****

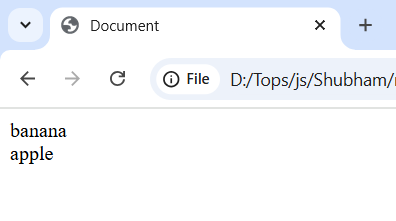
****

**2. pop():**

Removes the **last element** from the array.

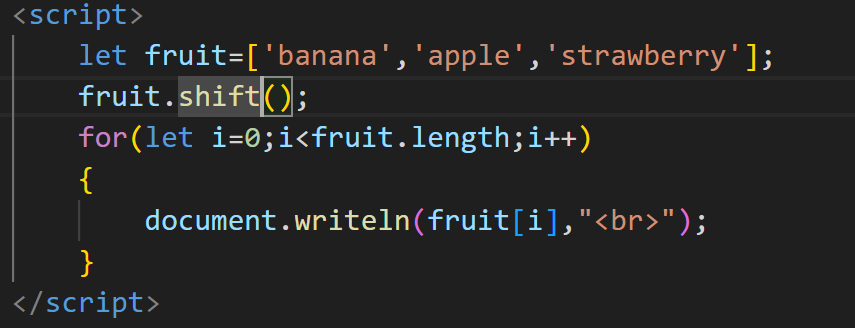
**Example:**

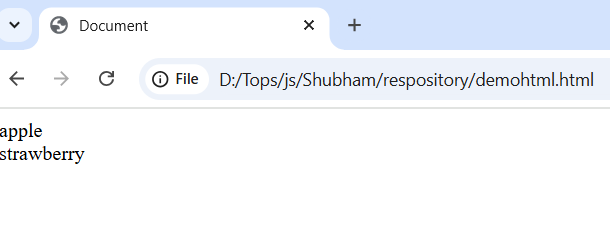
****

****

**3.shift():**Removes the **first element** of the array.

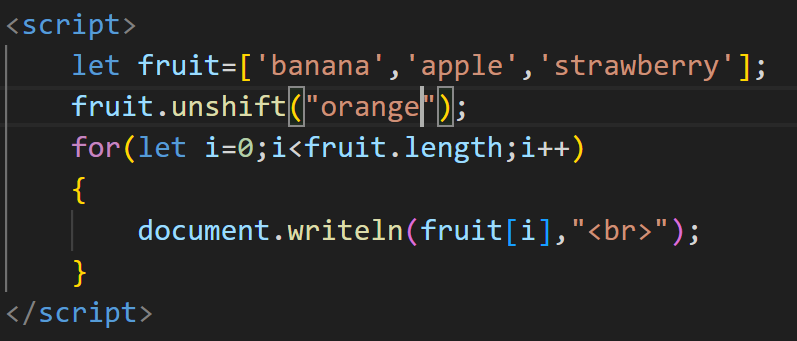
**Example:**

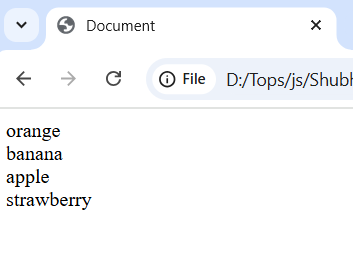
****

****

**4.unshift():** Adds **one or more elements** to the **beginning** of the array.

**Example:**

****

****

**8.Objects**

**Q.1: What is an object in JavaScript? How are objects different from arrays?**

**Ans:** In JavaScript, an object is an unordered collection of key-value pairs. Each key-value pair is called a property. The key of a property can be a string. The value of a property can be any value, e.g., a string, a number, an array, and even a function.

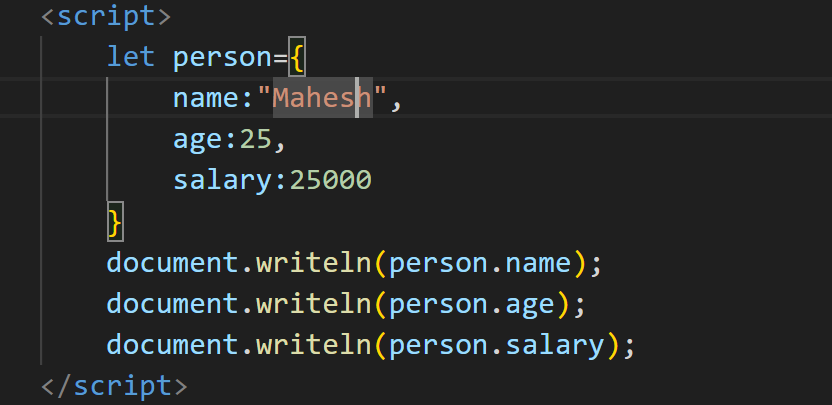
**How are objects different from arrays:**

|  |  |  |
| --- | --- | --- |
| Features | Array | Objects |
| Use case | Ordered list of items | Unordered data with named properties |
| Access key | Index(0,1,2, n) | Key name(name,age..) |
| Syntax | [ ] (square brackets) | { } (curly braces) |
| Best for | Lists, collections | Structured data (like a “thing”) |

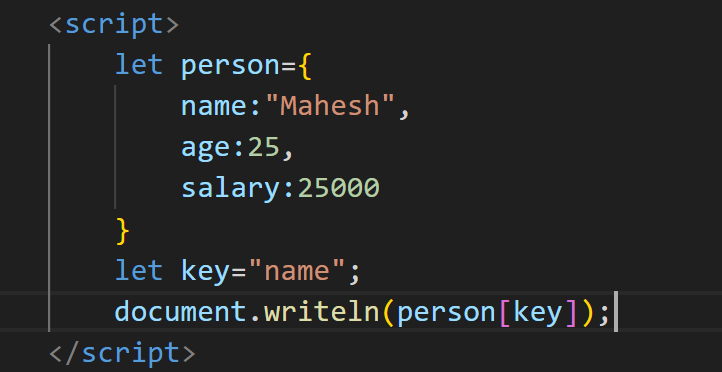
**Q.2: Explain how to access and update object properties using dot notation and bracket notation.**

**Ans:1. Accessing properties:**

* Dot Notation:

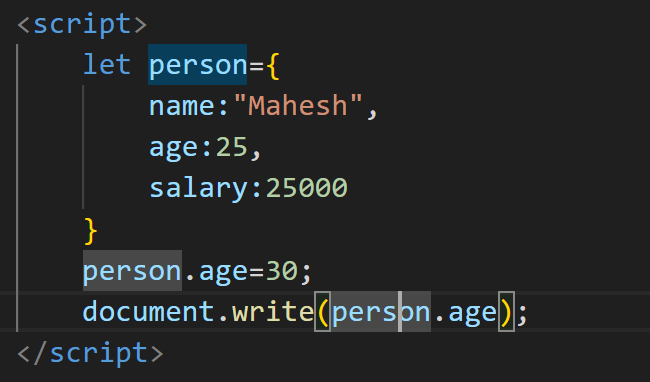


* Bracket Notation



**2.Update Properties**

* Dot Notation



* Bracket Notation



**9. JavaScript Events**

**Q.1: What are JavaScript events? Explain the role of event listeners.**

**Ans:** **JavaScript Events** are**actions or occurrences**that happen in the browser. They can be triggered by various user interactions or by the browser itself.

**Role of event listener:**

 **Detects Events:**

* Listens for user actions like **click**, **hover**, **keydown**, **submit**, etc.

 **Handles Interactivity:**

* Makes a website interactive by running a function when the event occurs.

 **Separates JavaScript from HTML:**

* Promotes clean code by keeping behavior (JS) separate from structure (HTML).

 **Dynamic Behavior:**

* You can attach or remove listeners at runtime based on conditions.

 **Supports Multiple Events:**

* You can attach different functions to the same element for different events.

**Q.2: How does the addEventListener() method work in JavaScript? Provide an example.**

**Ans:** The addEventListener() method is used to **attach an event handler** to a DOM element without overwriting existing event handlers.

**Syntax:** element.addEventListener(event, listener, useCapture);

* + **Parameters:**
    - **event:**event can be any valid JavaScript event. Events are used without “on” prefixes like using “click” instead of “onclick” or “mousedown” instead of “onmousedown”.
    - **listener(handler function):**It can be a JavaScript function that responds to the event occurring.
    - **useCapture:**It is an optional parameter used to control event propagation. A boolean value is passed where “*true*” denotes the capturing phase and “*false*” denotes the bubbling phase.

**How does addEventListener() method work:**

 **Select the HTML element** you want to listen to.

 **Call addEventListener()** on that element.

 **Pass the event type** you want to listen for (e.g., "click").

 **Provide a callback function** – this is the code that will run when the event happens.

 The browser **waits** for that event to happen on that element.

 When the event happens, **JavaScript automatically calls your function**.

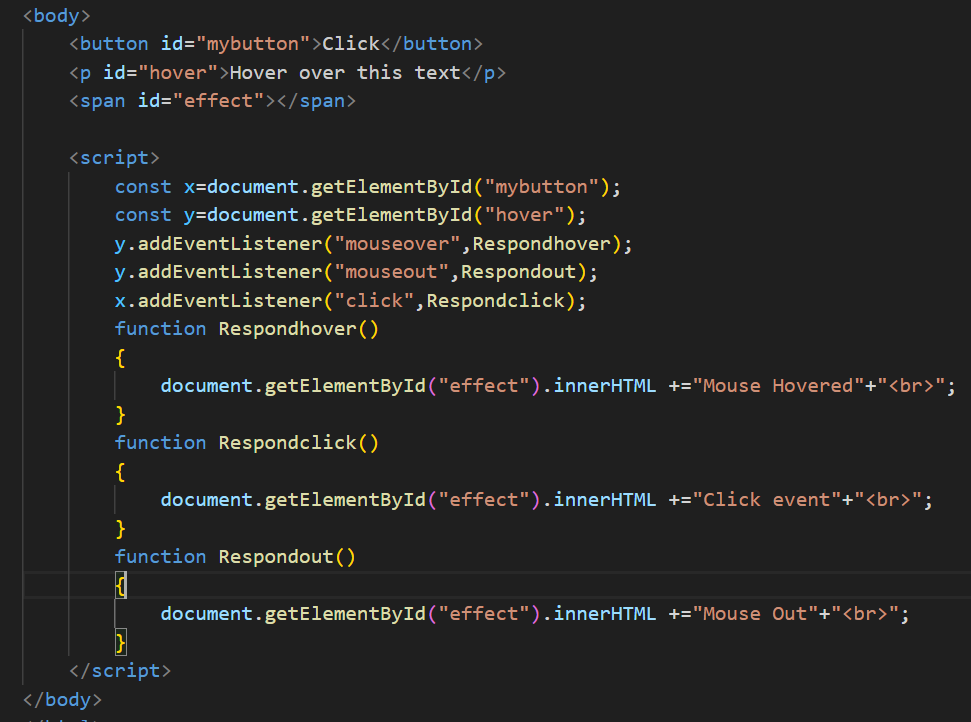
**Example 1:** In this example, we will display text on the webpage after clicking on the button.



**How It Works:**

1. JavaScript selects the button with id="myButton".
2. addEventListener("click", function...) tells the browser to **watch for a click** on that button.
3. When clicked, the **anonymous function** runs and displays an alert.

**Example 2:** In this example, two events “mouseover” and “mouseout” are added to the same element. If the text is hovered over then “mouseover” event occurs and the RespondMouseOver function is invoked, similarly for “mouseout” event RespondMouseOut function is invoked.

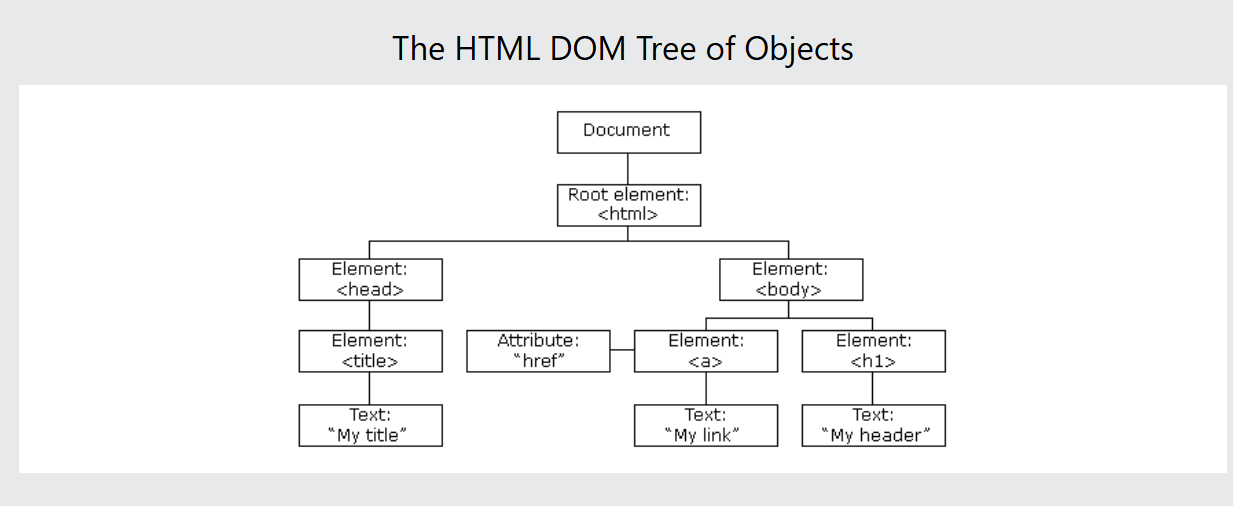


**10. DOM Manipulation**

**Q.1: What is the DOM (Document Object Model) in JavaScript? How does JavaScript interact with the DOM?**

**Ans:** The **DOM (Document Object Model)** is a way for **JavaScript to interact with HTML and CSS**. It turns the entire webpage into a **JavaScript object** that you can read and change.

In short, The **DOM** is like a **bridge** between JavaScript and the webpage.  
It lets JavaScript **see**, **access**, and **change** elements, content, styles, and structure of the page.



**How does JavaScript interact with the DOM?**

JavaScript interacts with the DOM (Document Object Model) through a set of built‑in objects, methods and properties that let you **select**, **traverse**, **manipulate**, and **listen** to changes in the page’s structure and content. Here’s how it works:

**1.The ‘document’ Object:**

At the top of the DOM API sits the global document object. It represents the entire HTML page and is your entry point into the DOM.

**// Access the <html> root element**

**console.log(document.documentElement);**

**// Access the <head> and <body>**

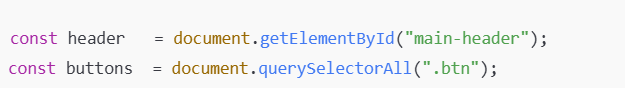
**console.log(document.head, document.body);**

**2. Selecting Elements:**

Before you can do anything, you need to grab references to one or more elements:

|  |  |
| --- | --- |
| **Method** | **What it does** |
| getElementById("id") | Returns one element with the given id |
| getElementsByClassName("class") | Returns a live HTMLCollection by class |
| getElementsByTagName("tag") | Returns a live HTMLCollection by tag name |
| querySelector("selector") | Returns the first element matching CSS selector |
| querySelectorAll("selector") | Returns a static NodeList of all matches |

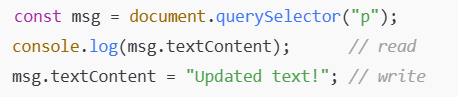
**Ex:**

****

**3. Reading & Changing Content**

Once you have an element, you can read or change its:

* + **Text content:**



* + **HTML content:**

msg.innerHTML = "<strong>Bold text</strong>";

* + **Attributes:**



**4. Changing Styles & Classes**

You can dynamically adjust CSS via the style object or by toggling classes:

****

**5. Creating & Inserting Elements**

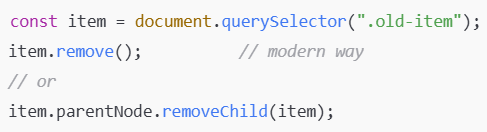
To build new DOM nodes:

****

You can also use methods like .insertBefore(), .prepend(), or .replaceChild() for more precise control.

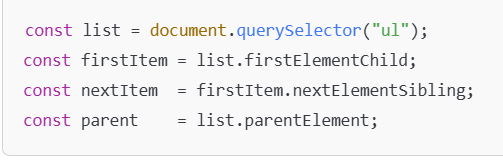
1. **Removing Elements:**

To remove a node from a tree

****

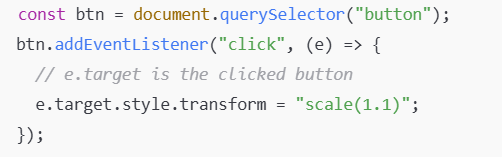
1. **Traversing the DOM:**

You can navigate between related nodes:

****

1. **Responding to Events:**

Interactivity comes from listening for user or browser events:



You can listen for hundreds of events: mouse (click, mouseover), keyboard (keydown, keyup), forms (submit, change), window (load, resize), and many more.

**Q.2: Explain the methods getElementById(), getElementsByClassName(), and querySelector() used to select elements from the DOM.**

**Ans:** **1. getElementById()**

**Syntax:**

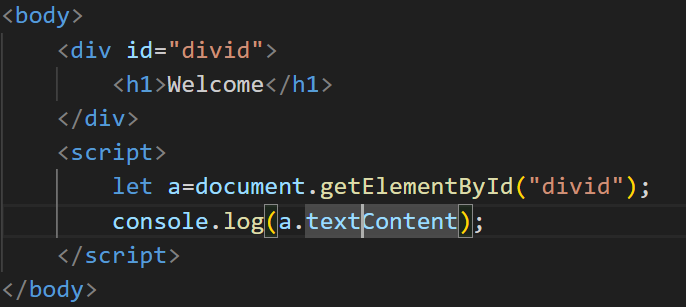
let a=document.getElementById(“some-id“);

**What it does:**  
Finds the *one* element whose id attribute exactly matches the string.

**Return value:**

* The matching Element object, or
* null if no element has that ID.

**Use‑case:**  
Fast, direct lookup when you know the unique id of an element.



**2. getElementsByClassName()**

**Syntax:**

let a=document.getElementsByClassName(“nameofclass”);

**What it does:**  
Finds *all* elements that have the given class name (exact match in their class list).

**Return value:**  
A **live** HTML Collection of zero or more elements. “Live” means it auto‑updates if you later add/remove elements with that class.

**Use‑case:**   
When you want to grab multiple elements sharing the same class and possibly respond to dynamic changes.



**3. querySelector()**

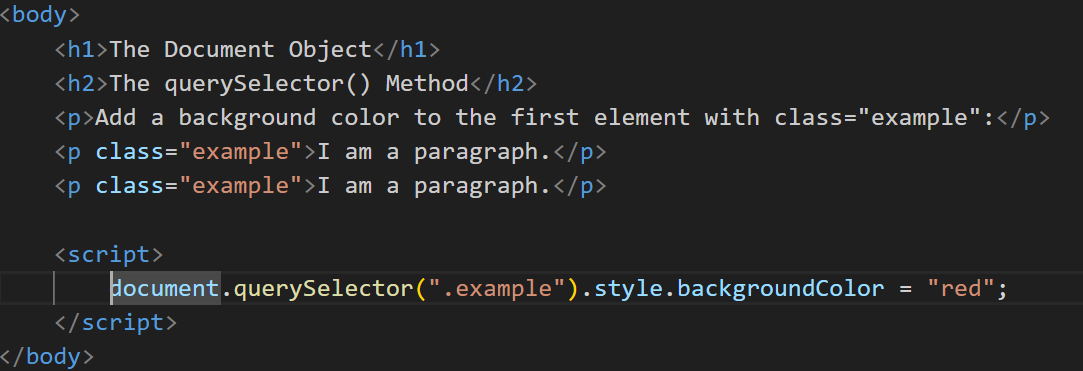
**Syntax:**

const firstmatch=document.querySelector(selectorstring);

**What it does:**  
Uses any **CSS selector** (ID, class, tag, attribute, combinators, pseudo‑classes, etc.) and returns the **first** element in document order that matches.

**Return value:**  
The matching Element, or null if nothing matches.

**Use‑case:**  
Flexible single‑element lookup when you need more complex targeting than just ID or class.



**11. JavaScript Timing Events (setTimeout, setInterval)**

**Q.1** **Explain the setTimeout() and setInterval() functions in JavaScript. How are they used for timing events?**

**Ans:** In JavaScript, setTimeout() and setInterval() are built-in functions used to execute code after a certain delay or repeatedly at specified intervals. These are commonly used for **timing events**, animations, polling data, etc.

* **setTimeout()**

**Definition:** setTimeout() is used to **execute a function once after a specified delay** (in milliseconds).

**Syntax:** setTimeout(function, delay, param1, param2, ...)

**Example:**



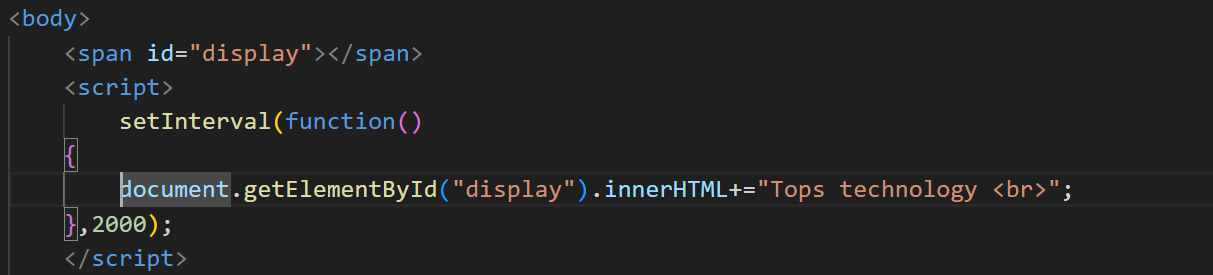
Note: in above example the message display in 2 seconds when page loaded.

* **setInterval()**

**Definition:** setInterval() is used to **repeatedly execute a function** at every given time interval (in milliseconds).

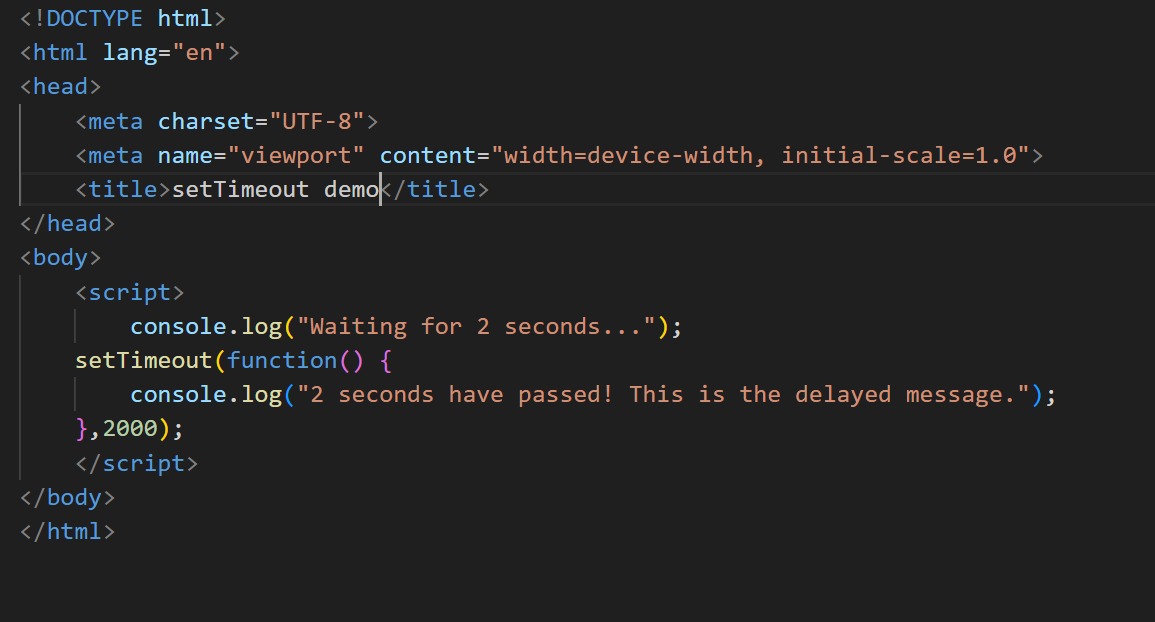
**Syntax: setInterval(function,interval,param1,param2,…);**

**Example:**



Note: in above example message display every 2 seconds.

**Q.2: Provide an example of how to use setTimeout() to delay an action by 2 seconds.**

**Ans: **

**Explanation:**

* setTimeout() takes two arguments:
  + A **function** to run
  + A **delay** in milliseconds (2000 ms = 2 seconds)

In this case:

* "Waiting for 2 seconds..." prints immediately
* After **2 seconds**, "2 seconds have passed!..." is printed

**12. JavaScript Error Handling**

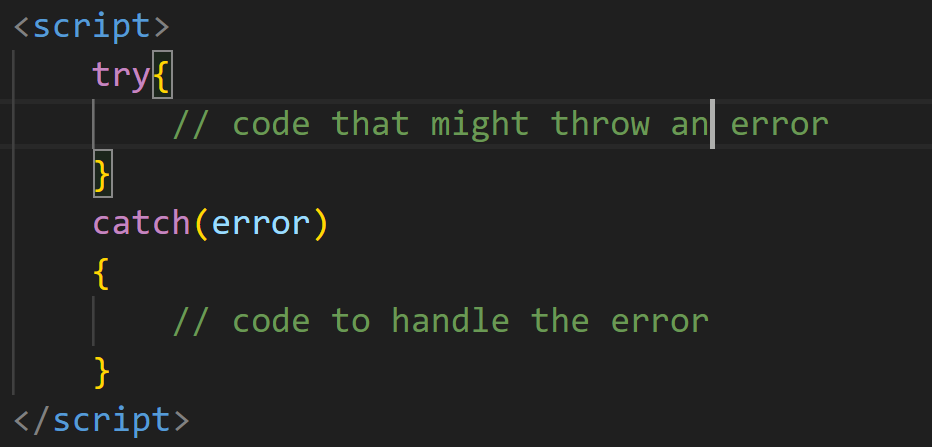
**Q.1What is error handling in JavaScript? Explain the try, catch, and finally blocks with an example.**

**Ans:** In JavaScript, error and exception handling allows you to manage unexpected issues that occur during the execution of your code. By using tools like try, catch, throw, and finally, you can handle errors easily and prevent the program from crashing. This enables you to provide meaningful error messages, debug your code efficiently, and maintain smooth execution.

**Error handling in JavaScript** refers to the process of managing errors that occur while a program is running, so that the program doesn't crash unexpectedly and can respond gracefully.

**try block:** A try block is used to wrap code that **might throw an error** while running. If an error occurs inside the try block, JavaScript immediately **stops executing** that block and jumps to the catch block.

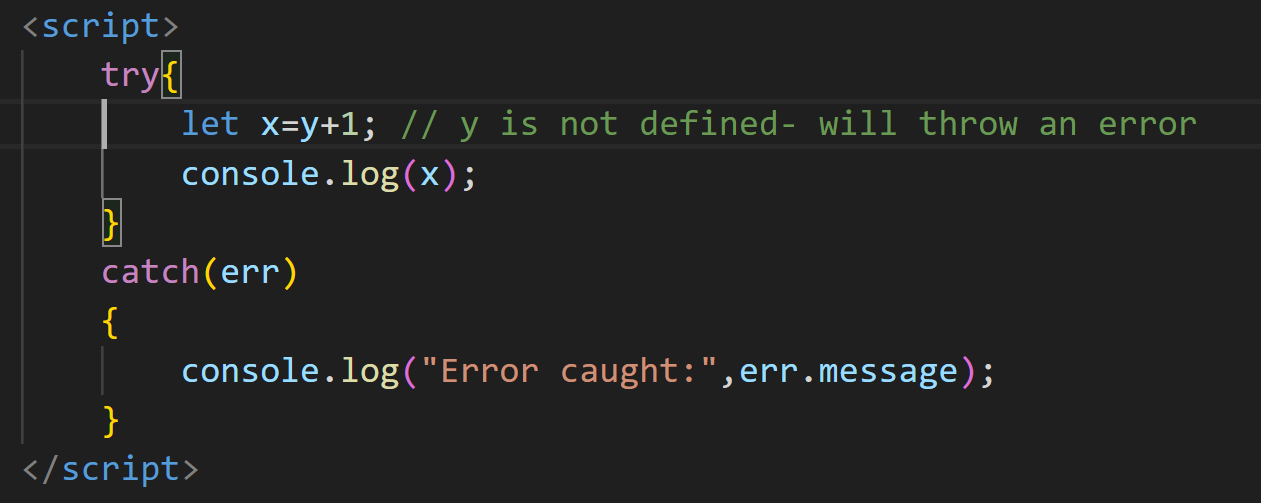
**Syntax:**



**How it’s work:**

1. JavaScript runs the code inside the try block.
2. If everything works fine, it skips the catch block.
3. If there’s an error, it **jumps to the catch block** and handles it without crashing the whole program.

**Example:**

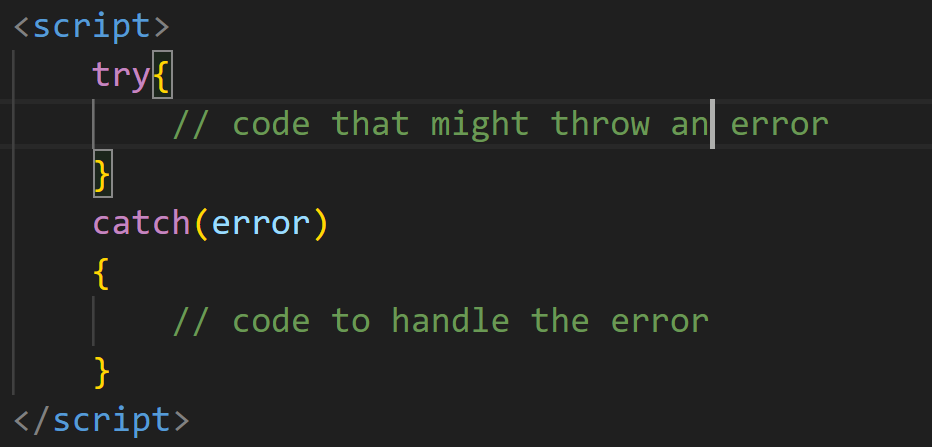


**catch block:**

The catch block is used to handle errors that occur in the try block.  
If an error happens in the try, the code jumps to catch, and you can write code here to:

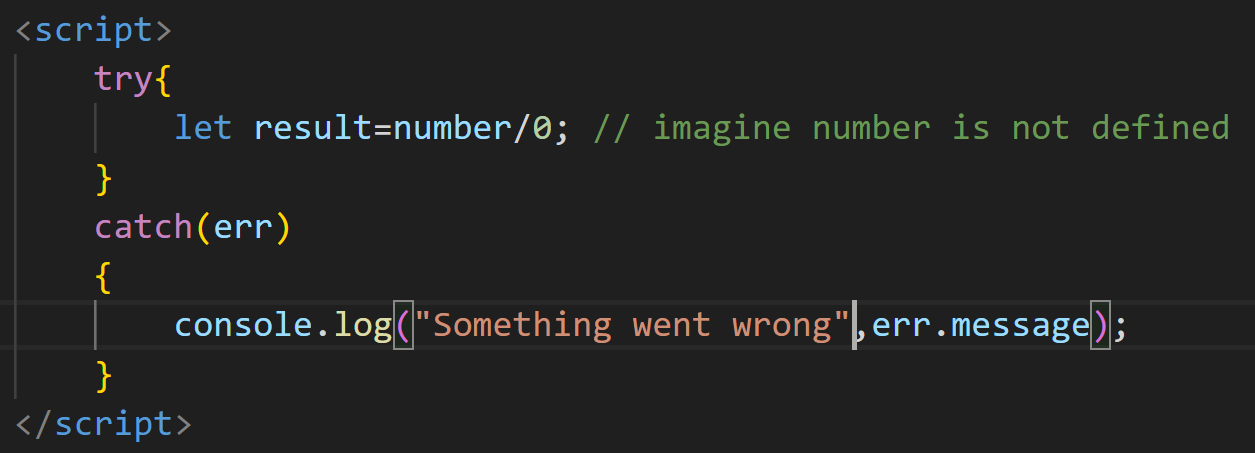
* Show an error message
* Log the error
* Prevent the app from crashing
* Take corrective action

**Syntax:**



The error is a variable (you can name it anything like e, err, etc.) that holds the **error object** with information about what went wrong.

**Example:**

****

**Finally block:**

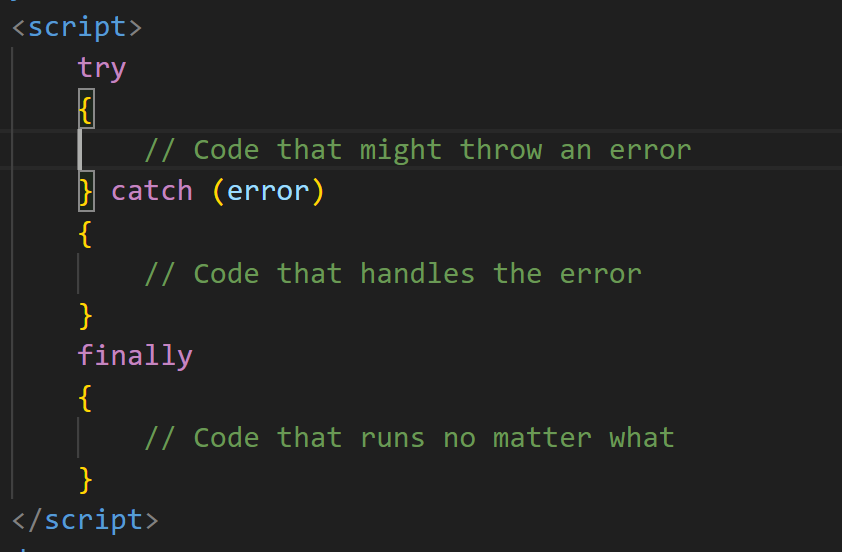
The finally block is used **to run code no matter what happens** whether there was an error or not.

It is placed **after try and catch** blocks and is **always executed**, even if:

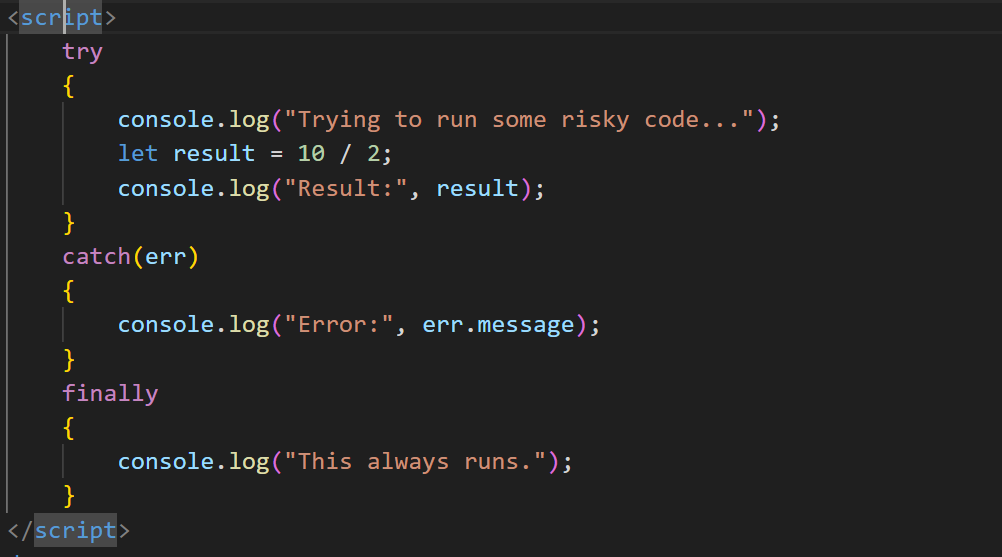
* There’s no error
* An error is caught
* There's a return inside try or catch

In short, the finally statement always executes after try and catch block regardless of if there was an error or not.

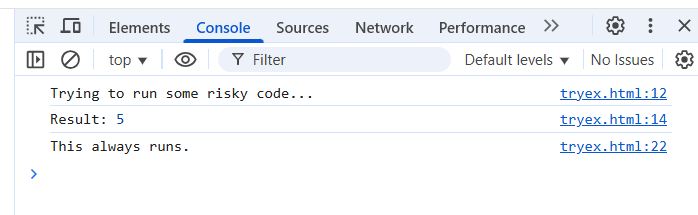
**Syntax:**

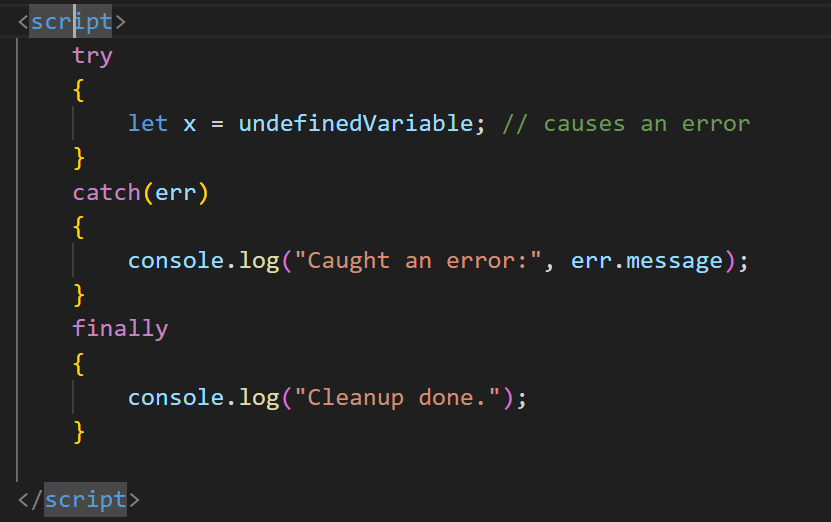
****

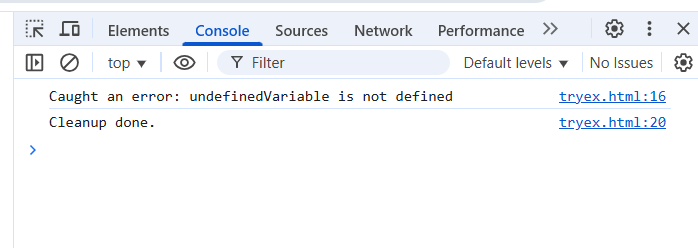
**Example:**

****

**Output:**



**Even if there's an error: **

**Output: Why use finally block:**

To clean up resources, close files/connections, stop loaders, or reset UI—even if something failed.

So, the finally block is super useful when you want to **guarantee some code runs**, like cleaning up or showing a message to the user.