HTML

1) **Doctype:**

HTML Doctype (<!DOCTYPE html>) is a declaration placed at the very top of an HTML document.
It tells the browser **which version of HTML** the page is written in so the browser can render the webpage correctly.

✓ What is <!DOCTYPE html>?

- It is **not** an HTML tag.
- It is simply an **instruction** for the browser.
- It tells the browser to use **HTML5**, which is the latest version of HTML.

✓ Why is Doctype important?

1. Helps the browser display the page in **standard mode** (correct mode).
2. Avoids **quirks mode**, where the browser behaves like old versions and may break your layout.
3. Ensures **better CSS and JavaScript compatibility**.

✓ Where to place it?

Always at the **top of the HTML file**, before <html> tag:

```
<!DOCTYPE html>
<html>
<head>
 <title>My Page</title>
</head>
<body>
 Hello!
</body>
</html>
```

✓ Does Doctype have closing tag?

No. There is **no closing tag** for the doctype.

2)HTML basic structure:

Every HTML document follows a common structure so the browser understands the page correctly.

1. Doctype

Tells the browser the version of HTML (HTML5).

<!DOCTYPE html>

2. <html> Tag

The root element that contains the entire HTML page.

<html>

3. <head> Section

This part contains **information about the webpage**, not visible to the user.
Examples:

- Page title
- CSS links
- Meta tags
- Scripts

<head>

<title>My Webpage</title>

</head>

4.<title>Section

Placed inside the <head> section of the HTML document.

Sets the text shown in the browser tab, in bookmarks, and as the default title in search engine results. Not displayed inside the page body.

<head>

<title>My Webpage</title>

</head>

5. <body> Section

This part contains **all visible content** shown on the webpage.
Examples:

- Text
- Buttons
- Images
- Tables
- Forms

```
<body>
  <h1>Hello!</h1>
</body>
```

6. Closing the <html> Tag

```
</html>
```

✓ Complete HTML Basic Structure

```
<!DOCTYPE html>
<html>
<head>
  <title>My Webpage</title>
</head>
<body>
  <h1>Welcome to HTML</h1>
</body>
</html>
```

3)Symentic structure:

Semantic HTML means using tags that describe the meaning and purpose of the content.
These tags help:
✓ Browsers
✓ Developers
✓ Search engines
✓ Screen readers (for accessibility)
understand what each part of the webpage represents.
Why Semantic HTML is Important?

- Improves SEO (Search Engine Optimization)
- Makes code more readable and organized
- Helps assistive technologies understand the structure
- Enhances overall user experience

Common Semantic Tags

1. <header>
Represents the top section of a page or section.
Usually contains logo, navigation, title, etc.
2. <nav>
Defines navigation links.
3. <main>
Contains the main content of the webpage.
Only one <main> per page.
4. <section>
Represents a thematic grouping of content.
Used to divide content into meaningful sections.
5. <article>
Used for self-contained content like blog posts, news, cards, comments.
6. <aside>
Represents side content such as ads, sidebars, or additional info.
7. <footer>
Bottom part of a page or section.
Contains copyright, links, contact info, etc.
8. <figure> & <figcaption>
Used for images with captions.
9. <details> & <summary>
Used to create expandable/collapsible content.

Simple Semantic Layout Example (Structure Only)

```
<!DOCTYPE html>
<html>
<head>
  <title>Semantic HTML Page</title>
</head>
<body>

  <header>Header Area</header>

  <nav>Navigation Menu</nav>
<main>
```

```
    <section>
     <article>
      Article Content
     </article>
    </section>
<aside>Sidebar Content</aside>
 </main>
<footer>Footer Information</footer>
</body>
</html>
```

4)Block and Inline elements in html:

Block Elements in HTML

Definition:

Block elements always start on a new line and take the full width of the page (or parent).

Key Points:

- Start from a new line
- Take 100% width
- You can set width, height, margin, padding
- They stack one below another

Common Block Elements:

-
- <p>
- <h1> to <h6>
- <header>
- <footer>
- <section>
- <article>
- <nav>
- , <ol>,
- <table>

- <form>

Inline Elements in HTML

Definition:

Inline elements do not start on a new line and take only the required width based on their content.

Key Points:

- Do not start a new line
- Take only content width
- Cannot set width or height
- Appear within a line side-by-side

Common Inline Elements:

-
- <a>
- <img>
-
-
- <label>
- <input>
- <textarea>
- <select>
- <button> (inline-block type)
- <b>, <i>, <u>

5) Tags (header,footer,main,div,span,form,input,anchar,p,section,img,article,aside):

HTML Tags and Their Meanings

1. <header>

Used for the top section of a page or section (logo, title, navigation).

2. <footer>

Represents the bottom section of a page or section (copyright, links, contact info).

3. <main>

Contains the main and unique content of the webpage (only one per page).

**4. **

A block-level container used to group elements for layout or styling.

**5. **

An inline container used for styling small pieces of text.

6. <form>

Used to create a form for user input (login, signup, search etc.).

7. <input>

Used inside a form to take user input (text, password, email, checkbox, etc.).

8. <a> (anchor tag)

Used to create hyperlinks to other pages or sections.

9. <p> (paragraph)

Used to represent a paragraph of text.

10. <section>

Represents a thematic section of a webpage (like chapters or grouped content).

11. <img>

Used to display images on a webpage.

12. <article>

Defines self-contained content like blog posts, news items, or cards.

13. <aside>

Represents side content like ads, sidebars, or extra information.

Attributes in HTML

Definition: Attributes are extra information added inside an HTML tag to modify, control, or provide details about an element.They are always written inside the opening tag.

Format:

<tagname attribute="value">Content</tagname>

Common HTML Attributes

1. id

Gives a unique name to an element.

2. class

Used to group elements for styling or JavaScript.

3. style

Applies inline CSS to an element.

4. src

Used in <img> to specify the image path.

5. href

Used in <a> tag to specify the link URL.

6. alt

Used in <img> to describe the image.

7. type

Used in <input> to specify the input type (text, password, email, etc.).

8. value

Sets the initial value of form elements.

9. name

Gives a name to form inputs for backend identification.

10. placeholder

Shows a hint inside text input boxes.

11. title

Shows tooltip text when you hover over an element.

12. width / height

Used in <img>, <video>, <canvas> etc.

13. disabled

Makes an input uneditable.

Important Points

- Attributes are always written in the opening tag.
- Most attributes follow name="value" format.
- Some attributes like disabled, checked, readonly work without value (boolean).

Events in HTML:

Definition:

HTML events are actions or activities that happen in the browser (like clicking, typing, loading, etc.) and you can run JavaScript when these events occur.

Example actions:

- User clicks a button
- User types in a textbox
- Page loads
- Mouse moves
- Form submits

Categories of HTML Events

1. Mouse Events

Triggered by mouse actions.

Event	Meaning
onclick	When an element is clicked
ondblclick	When element is double-clicked
onmouseover	When mouse pointer enters an element
onmouseout	When mouse pointer leaves an element
onmousemove	When mouse is moved
onmousedown	Mouse button pressed
onmouseup	Mouse button released

2. Keyboard Events

Triggered when the user types.

Event	Meaning
onkeydown	Key is pressed down
onkeyup	Key is released
onkeypress	Key is pressed and held

3. Form Events

Triggered by form elements.

Event	Meaning
onsubmit	When form is submitted
onreset	When form is reset
onchange	When value changes (dropdown, input)
oninput	When user types input
onfocus	When element gets focus
onblur	When element loses focus

4. Window / Page Events

Event	Meaning
onload	When page finishes loading
onunload	When page is closed or refreshed
onresize	When window is resized
onscroll	When user scrolls the page

5. Clipboard Events

Event	Meaning
oncopy	When user copies
oncut	When user cuts

6)Web Storage in HTML:

Definition:

Web Storage is a feature in HTML5 that allows websites to store data in the browser safely and easily.

It stores data locally on the user's browser (not on the server).

Types of Web Storage

HTML provides two types of storage:

1. localStorage

- Stores data permanently (until manually cleared).
- Data remains even after the browser is closed.
- Storage limit: around 5MB per site.
- Works on key–value pairs.

Example meaning:

localStorage.setItem("name", "John");

(This stores the data permanently in browser storage.)

2. sessionStorage

- Stores data temporarily.
- Data is deleted when the browser tab is closed.
- Also uses key–value pairs.
- Limit: around 5MB.

Example meaning:

sessionStorage.setItem("age", "25");

(Data will be lost after closing the tab.)

7) including CSS in HTML:

There are three ways to include CSS inside an HTML page:

Inline CSS

CSS is written inside the HTML tag using the style attribute.

Syntax:

This is a paragraph

Use when:

You want to style only one specific element.

Internal CSS

CSS is written inside the <style> tag in the <head> section of the HTML file.

Syntax:

```
<head>
 <style>
  p {
   color: red;
  }
 </style>
</head>
```

Use when:

You want styles for one single page.

External CSS

CSS is written in a separate .css file, and linked using the <link> tag.

Syntax (HTML):

```
<link rel="stylesheet" href="styles.css">
```

Syntax (styles.css):

```
p {
 color: green;
}
```

Use when:

You want styling for multiple pages – best and professional method.

8)including script in HTML:

Including Script in HTML (JavaScript)

There are three main ways to include JavaScript in an HTML page:

Inline Script

JavaScript is written inside an HTML tag using the onclick, onchange, etc. event attributes.

Syntax:

<button onclick="alert('Hello!')">Click Me</button>

Use when:

Very small scripts or simple actions.

Internal Script

JavaScript is written inside the <script> tag within the HTML file (usually inside <head> or at the bottom of <body>).

Syntax:

<script>

alert("Page Loaded!");

</script>

Use when:

Small or medium JavaScript for a single page.

External Script

JavaScript is written in a separate .js file, and linked using the <script src=""> tag.

Syntax (HTML):

<script src="script.js"></script>

Syntax (script.js):

console.log("Hello from external file!");

Use when:

Large projects, clean code, reusable scripts — best and professional method.

Where to place <script>?

Option 1: In <head>

Script loads before content → page may load slower.

Option 2: At the bottom of <body> (recommended)

Loads HTML first → faster page load.

```
<body>
  ...
  <script src="app.js"></script>
</body>
```

CSS

1)CSS Box Model

The Box Model in CSS describes how every HTML element is treated as a box.
This box is made up of:

1. Content (width, height)
2. Padding
3. Border
4. Margin

Width & Height

- These define the size of the content area.
- Content is the actual text, image, or element inside the box.

```
width: 200px;
height: 100px;
```

Padding

- Space inside the box, between the content and border.
- It increases the total size of the element.

```
padding: 20px;
```

Border

- A line around the padding and content.
- It also adds extra size to the box.

```
border: 2px solid black;
```

Margin

- Space outside the border, used to create distance between elements.

margin: 20px;

Total Element Size Formula:

Total Width = width + padding-left + padding-right + border-left + border-right + margin-left + margin-right

Total Height = height + padding-top + padding-bottom + border-top + border-bottom + margin-top + margin-bottom

Short Summary

- **Width/Height** = content size
- **Padding** = space inside
- **Border** = line around the box
- **Margin** = space outside

2)Position in CSS:

The **position** property is used to control **how an element is placed** on a webpage.

There are **5 types**:

1. static
2. relative
3. absolute
4. fixed
5. sticky

Let's understand each with **meaning + example**.

position: static (default)

✓ Every element by default is static
✓ It **cannot be moved** using top, left, etc.

Example

```
.box {
  position: static;
}
```

position: relative

✓ Element stays in normal position
✓ But **you can move it** using top, left, right, bottom
✓ Movement happens **relative to its original position**

Example

```
.box {
  position: relative;
  top: 20px;
  left: 30px;
}
```

position: absolute

✓ Moves the element **anywhere on the page**
✓ Positioned relative to the **nearest positioned parent**
(parent must have position: relative or similar)
✓ If no parent is positioned → it uses the **page (body)**

Example

```
.parent {
  position: relative;
}


.child {
  position: absolute;
  top: 20px;
  left: 40px;
}
```

position: fixed

✓Sticks the element to the **browser window**
✓ Does not move even when scrolling
✓ Used for: header bars, chat buttons, ads, etc.

Example

```
.box {
  position: fixed;
  bottom: 10px;
  right: 10px;
}
```

position: sticky

✓ Behaves like **relative**
✓ But **sticks** to the top when user scrolls
✓ Best for sticky headers

Example

```
.box {
  position: sticky;
  top: 0;
}
```

CSS Position Examples

```
<!DOCTYPE html>
<html>
<head>
<style>
.box {
  width: 150px;
  height: 50px;
  background: lightblue;
  margin: 10px;
  padding: 10px;
  border: 2px solid black;
}
```

```
/* 1) STATIC */
.static-box {
  position: static;
}
/* 2) RELATIVE */
.relative-box {
  position: relative;
  top: 20px;   /* moved down */
  left: 30px;  /* moved right */
}
/* 3) ABSOLUTE */
.parent {
  position: relative; /* important */
  height: 150px;
  background: #f6e58d;
}
.absolute-box {
  position: absolute;
  top: 20px;
  left: 40px;
  background: orange;
}
/* 4) FIXED */
.fixed-box {
  position: fixed;
  bottom: 10px;
  right: 10px;
  background: lightgreen;
```

```
}
/* 5) STICKY */
.sticky-box {
  position: sticky;
  top: 0;
  background: pink;
}
</style>
</head>
<body>
<h2>Position Examples</h2>
<!-- Static -->
<div class="box static-box">Static Position</div>
<!-- Relative -->
<div class="box relative-box">Relative Position</div>
<!-- Absolute -->
<div class="parent">
  <div class="box absolute-box">Absolute Position</div>
</div>
<!-- Sticky -->
<div class="box sticky-box">Sticky Position (Scroll Down)</div>
<p style="height:600px;">Scroll to see sticky and fixed effect...</p>
<!-- Fixed -->
<div class="box fixed-box">Fixed Position</div>
</body>
</html>
```

What You Will See

✓ Static → normal

✓ Relative → slightly moved down & right

✓ Absolute → placed inside the yellow parent box

✓ Sticky → sticks at top while scrolling

✓ Fixed → stays at bottom-right even when scrolling

3)Borders in CSS:

The border property is used to add an outline around an HTML element.

A border has 3 main parts:

1. Border Width
2. Border Style
3. Border Color

You can write them together or separately.

Border (Shorthand):

You can set width, style, color in one line.

Syntax

border: 2px solid red;

Border Width:

Controls how thick the border is.

Example

border-width: 5px;

Border Style

Defines how the border looks.

Common Border Styles

- solid
- dashed
- dotted
- double
- groove

- ridge
- inset
- outset
- none

Example

border-style: dashed;

Border Color

Sets the color of the border.

Example

border-color: blue;

Rounded Border (border-radius)

Used to make rounded corners.

Example

border-radius: 10px;

Applying Border to Specific Sides

You can apply border to one side:

Example

border-top: 3px solid green;

border-right: 2px dashed red;

border-bottom: 4px dotted blue;

border-left: 5px double black;

Full Example (Simple Box)

```
.box {
  border: 3px solid blue;
  border-radius: 10px;
  width: 200px;
  padding: 10px;
}
```

4)boxsizing in CSS:

The box-sizing property controls how the width and height of an element are calculated.

There are mainly two values:

1. content-box (default)
2. border-box (most commonly used)

box-sizing: content-box (Default)

✓ Width & height apply only to the content
✓ Padding and border are added outside the width and height
✓ Final size becomes bigger than the given width

Example

```
.box {
  width: 200px;
  padding: 20px;
  border: 5px solid black;
  box-sizing: content-box;
}
```

Final Width Calculation:

200 (content) + 40 (padding) + 10 (border) = 250px total width

box-sizing: border-box (Recommended)

✓ Width & height include:

- Content
- Padding
- Border

✓ Final size does NOT increase

Example

```
.box {
  width: 200px;
```

```
 padding: 20px;
 border: 5px solid black;
 box-sizing: border-box;
}
```

Final Width Calculation:

Always 200px (padding and border included)

5)float in CSS:

The float property is used to move an element to the left or right of its container, allowing text or other elements to wrap around it.

Values of float

1. float: left;

Moves the element to the left, and other content flows on the right.

2. float: right;

Moves the element to the right, and other content flows on the left.

3. float: none;

(Default) No floating is applied.

Basic Example

```
img {
 float: left;
 width: 150px;
 margin-right: 10px;
}
```

Meaning:

The image moves to the left, and text wraps around the right side.

Example with Right Float

```
img {
 float: right;
```

```
  width: 150px;
  margin-left: 10px;
}
```

Clearing Float

Sometimes floated elements cause layout problems.
Use clear to stop wrapping.

```
.clear {
  clear: both;
}
<div class="clear"></div>
```

Full Example

```
<!DOCTYPE html>
<html>
<head>
<style>

.box {
  width: 200px;
  height: 100px;
  background: lightblue;
  margin: 10px;
  float: left;
}

</style>
</head>

<body>
```

```
<div class="box">Box 1</div>

<div class="box">Box 2</div>

<div class="box">Box 3</div>


<div style="clear: both;"></div>


<p>After clearing float, normal content continues.</p>


</body>

</html>
```

What happens?

✓ All boxes line up horizontally
✓ Next content starts below only after clear: both

6)Background in CSS:

The **background** properties in CSS are used to style the **background** of an element using:

✓ Color
✓ Image
✓ Position
✓ Repeat
✓ Size
✓ Attachment

background-color

Sets the **background color** of an element.

Example:

```
div {
  background-color: lightblue;
}
```

background-image

Adds an **image** as the background.

Example:

```
div {
  background-image: url("image.jpg");
}
```

background-repeat

Controls whether the image should **repeat or not**.

Values:

- repeat (default)
- no-repeat
- repeat-x (repeat horizontally)
- repeat-y (repeat vertically)

Example:

```
div {
  background-image: url("bg.png");
  background-repeat: no-repeat;
}
```

background-position

Sets the **starting position** of the background image.

Values:

- left, center, right
- top, bottom
- 50px 20px (x and y)

Example:

```
div {
  background-position: center;
}
```

background-size

Controls **how large** the background image should be.

Common values:

- cover → fills entire area (crop possible)
- contain → full image visible (may leave gaps)
- 100% 100% → stretch fully

Example:

```
div {
  background-size: cover;
}
```

background-attachment

Controls whether the image **scrolls** with the page or stays **fixed**.

Values:

- scroll (default)
- fixed (stays in one place)
- local

Example:

```
div {
  background-attachment: fixed;
}
```

background (Shorthand)

You can write all background properties **in one line**.

Example:

```
div {
  background: url("pic.jpg") no-repeat center/cover fixed;}
```

Javascript

1)Data Types in JavaScript:

JavaScript has **8 official data types** (as per ECMAScript).

They are divided into:

1. Primitive Data Types (7 types)

2. Non-Primitive Data Type (1 type: Object)

Let's understand them deeply.

Primitive Data Types (Immutable)

Primitive means:

✓ Stored directly in memory
✓ Hold a **single value**
✓ **Immutable** (cannot be changed, only replaced)
✓ Stored in **stack memory**

JavaScript primitives:

1. Number

Represents **integer** and **floating-point** numbers.

Examples:

10, 15.5, -20

Special number values:

- Infinity
- -Infinity
- NaN (Not a Number)

2. String

A sequence of characters.

"hello"

'hello'

`hello`

Strings are immutable.

3. Boolean

Represents either:

true or false

4. Undefined

A variable declared but **not assigned** a value.

let x; // undefined

5. Null

Represents **empty value**.

Important fact:

typeof null → "object" (bug in JS, but kept for backward compatibility)

6. BigInt

Used for **very large numbers** beyond Number limit.

let a = 12345678901234567890n;

7. Symbol

Provides a **unique value**, mainly used as object keys.

let id = Symbol("key");

Non-Primitive Data Type

8. Object

Objects store **multiple values** in **key–value pairs**.

Objects are **mutable** and stored in **heap memory**.

Examples:

{ name: "John", age: 20 }

Objects include:

- Arrays: [10,20,30]
- Functions: function(){}

- Dates: new Date()
- Maps, Sets, WeakMap, WeakSet
- RegExp

All are objects.

2)Hoisting in js:

Hoisting is not just “moving declarations to the top.”
It is actually the behavior of the **JavaScript engine** during the **memory creation phase** of the Execution Context.

How JavaScript Executes Code

JavaScript runs code in **two phases**:

Memory Creation Phase (Hoisting phase)

- Variables and functions are **registered in memory**
- No code is executed yet

Execution Phase

- Code runs **line by line**

Hoisting happens **during the Memory Creation Phase**.

What Actually Gets Hoisted?

✓ Function Declarations → Fully hoisted

✓ var variables → Hoisted but set to undefined

✓ let & const → Hoisted but NOT initialized (TDZ)

✓ Class declarations → Hoisted but in TDZ

Hoisting with var (Memory Behavior)

Code:

```
console.log(a);

var a = 5;
```

Memory Creation Phase:

a → undefined (hoisted)

Execution Phase:

console.log(a); // undefined

a = 5;

- ❖ No error
- ❖ But value is undefined before assignment

Hoisting with let and const (TDZ)

let and const are hoisted but are placed in the:

Temporal Dead Zone (TDZ)

A region between the start of the scope and the actual declaration.

Code:

console.log(x);

let x = 10;

Memory Creation Phase:

x → uninitialized (TDZ)

Execution:

Trying to access it before the declaration → **ReferenceError**

Why TDZ Exists?

To avoid common bugs like using variables before they are ready.

Hoisting with Function Declarations

Code:

greet();

function greet() {

  console.log("Hello");

}

Memory Creation Phase:

greet → entire function stored

Execution Phase:

```
greet();  // "Hello"
```

✓ Works perfectly
✓ Full function hoisted

Hoisting with Function Expressions

Example 1: Using var

```
greet();

var greet = function() {
  console.log("Hi");
};
```

Memory Creation Phase:

greet → undefined

Execution:

```
greet(); // TypeError: greet is not a function
```

Because greet is undefined at that moment.

Example 2: Using let/const

```
greet();

let greet = function() {};
```

Memory:

greet → uninitialized (TDZ)

Execution:

ReferenceError

Important: Function Hoisting Priority

Function declarations are hoisted **before** var declarations.

Example:

```
console.log(test);

function test() {}

var test = 10;
```

Memory Creation:

test → function test(){}

test → undefined (var re-declared but ignored)

Execution Output:

function test(){}

Hoisting in Block Scope

let & const are block scoped.

Example:

```
{
  console.log(a);
  let a = 20;
}
```

ReferenceError (TDZ inside block)

Hoisting with Class

Classes are hoisted but also placed in TDZ like let/const.

Example:

```
let obj = new Person(); //
class Person {}
```

Error: **ReferenceError**

3)Variable declarions and scope :

JavaScript has three ways to declare variables:

1. var

- Old way of declaring variables.
- Function-scoped.
- Can be redeclared and updated.
- Not recommended in modern JavaScript.

2. let

- Modern way (ES6).

- Block-scoped.
- Cannot be redeclared in the same block.
- Can be updated.

3. const

- Also ES6.
- Block-scoped.
- Cannot be updated or redeclared.
- Must be assigned a value when declared.

Scope in JavaScript:

Scope means where a variable is accessible.

There are three types:

1. Block Scope (for let and const)

A block is anything inside { }.

Example blocks:

- if block
- for loop block
- function block

Variables declared using let and const are only available inside that block.

2. Function Scope (for var)

If a variable is declared with var inside a function, it can be accessed only inside that function.

3. Global Scope

If a variable is declared:

- Outside any function
- OR without var/let/const (not recommended)

Then it becomes globally accessible.

Quick Summary Table

4)Conditional Statements in JavaScript:

Conditional statements are used to make decisions in your program.
They check a condition and run code only if the condition is true.

JavaScript has these main conditional statements:

if statement

Runs a block of code only if the condition is true.

```
if (condition) {
  // code
}
```

if...else statement

Runs one block if the condition is true,
and another block if the condition is false.

```
if (condition) {
  // code if true
} else {
  // code if false
}
```

if...else if...else

Used when you have multiple conditions.

```
if (condition1) {
  // code
} else if (condition2) {
  // code
} else {
  // final code if none true
}
```

switch statement

Used when you want to compare one value against many cases.

```
switch(value) {
```

```
  case 1:
    // code
    break;
  case 2:
    // code
    break;
  default:
    // code if no case matches
}
```

Ternary Operator (?:)

Short form of if...else.
Used for simple one-line conditions.

```
condition ? valueIfTrue : valueIfFalse;
```

5)Loops:

for loop

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

```
for (initialization; condition; increment) {
  // code
}
```

Example

```
for (let i = 1; i <= 5; i++) {
  console.log(i);
}
```

Output: 1 2 3 4 5

while loop

Runs a block of code as long as the condition is true.

```
while (condition) {
  // code
}
```

Example:

```
let i = 1;

while (i <= 5) {
  console.log(i);
  i++;
}
```

Output: 1 2 3 4 5

do...while loop

Similar to while loop,
but executes the code at least once, even if the condition is false.

```
do {
  // code
} while (condition);
```

Example:

```
let i = 1;

do {
  console.log(i);
  i++;
} while (i <= 5);
```

Output: 1 2 3 4 5

(Always runs at least one time)

for...in loop

Used to loop through object properties.

```
for (let key in object) {
   // code
}
```

Example:

```
let person = { name: "Rahul", age: 22, city: "Delhi" };


for (let key in person) {
   console.log(key, person[key]);
}
```

Output:

name Rahul

age 22

city Delhi

for...of loop

Used to loop through iterables like arrays, strings, sets, maps.

```
for (let element of array) {
   // code
}
```

Example:

```
let numbers = [10, 20, 30];


for (let num of numbers) {
   console.log(num);
}
```

Output: 10 20 30

forEach()

Used to loop through arrays only.

```
array.forEach(function(item) {
   // code
});
```

Example:

```
let fruits = ["apple", "banana", "mango"];


fruits.forEach(function(fruit) {
   console.log(fruit);
});
```

Output: apple banana mango

6) Arrays methods:

Array Methods in JavaScript

1. push()

Adds an element to the end of an array.

```
let arr = [1, 2];
arr.push(3);
console.log(arr); // [1, 2, 3]
```

2. pop()

Removes the last element.

```
let arr = [1, 2, 3];
arr.pop();
console.log(arr); // [1, 2]
```

3. unshift()

Adds an element to the beginning.

```
let arr = [2, 3];
```

```
arr.unshift(1);

console.log(arr); // [1, 2, 3]
```

4. shift()

Removes the first element.

```
let arr = [1, 2, 3];

arr.shift();

console.log(arr); // [2, 3]
```

5. concat()

Joins two arrays.

```
let a = [1, 2];

let b = [3, 4];

let c = a.concat(b);

console.log(c); // [1, 2, 3, 4]
```

6. slice()

Returns a portion of array (does NOT change original array).

```
let arr = [10, 20, 30, 40];

let part = arr.slice(1, 3);

console.log(part); // [20, 30]
```

7. splice()

Adds/Removes elements (changes original array).

```
let arr = [10, 20, 30, 40];

arr.splice(1, 2);

console.log(arr); // [10, 40]
```

8. indexOf()

Returns index of value.

```
let arr = ["a", "b", "c"];

console.log(arr.indexOf("b")); // 1
```

9. includes()

Checks if value exists in array.

```
let arr = [1, 2, 3];

console.log(arr.includes(2)); // true
```

10. join()

Converts array to string.

```
let arr = ["apple", "banana"];

console.log(arr.join(" - ")); // apple – banana
```

11. reverse()

Reverses array.

```
let arr = [1, 2, 3];

arr.reverse();

console.log(arr); // [3, 2, 1]
```

12. sort()

Sorts array.

```
let arr = [30, 5, 10];

arr.sort();

console.log(arr); // [10, 30, 5]
```

13. map()

Creates a new array by applying a function to each element.

```
let arr = [1, 2, 3];

let doubled = arr.map(x => x * 2);

console.log(doubled); // [2, 4, 6]
```

14. filter()

Returns elements that pass a condition.

```
let arr = [10, 5, 30];

let big = arr.filter(x => x > 10);

console.log(big); // [30]
```

15. reduce()

Reduces array to a single value.

```
let arr = [1, 2, 3];

let sum = arr.reduce((total, x) => total + x);

console.log(sum); // 6
```

16. forEach()

Loops through each element.

```
let arr = [1, 2, 3];

arr.forEach(x => console.log(x));
```

17. find()

Returns the first matching element.

```
let arr = [10, 20, 30];

console.log(arr.find(x => x > 15)); // 20
```

18. findIndex()

Returns index of first match.

```
let arr = [10, 20, 30];

console.log(arr.findIndex(x => x > 15)); // 1
```

19. some()

Returns true if at least one element passes a condition.

```
let arr = [1, 2, 3];

console.log(arr.some(x => x > 2)); // true
```

20. every()

Returns true if all elements pass.

```
let arr = [2, 4, 6];

console.log(arr.every(x => x % 2 === 0)); // true
```

21. flat()

Flattens nested arrays.

```
let arr = [1, [2, 3], [4]];

console.log(arr.flat()); // [1, 2, 3, 4]
```

7) es6 in Javascript:

ES6 (ECMAScript 2015) – Clear Explanation

ES6 is an updated version of JavaScript that introduced new features to make coding easier, faster, and cleaner.

1. let and const

Used to declare variables.

let

- Can change value
- Block scoped

```
let age = 20;

age = 25;   // allowed
```

const

- Cannot change value
- Block scoped

```
const pi = 3.14;

// pi = 3.15     // not allowed
```

2. Arrow Functions (=>)

Short way to create functions.

Before ES6:

```
function add(a, b) {
  return a + b;
}
```

With ES6:

```
const add = (a, b) => a + b;
```

✓ Short
✓ Clean
✓ Used everywhere

3. Template Literals ( )

Use backticks instead of quotes
Allows you to easily insert variables inside strings.

```
let name = "Ravi";

console.log(`Hello, ${name}!`);
```

4. Default Parameters

Function parameters get a default value.

```
function greet(name = "Guest") {
  console.log("Hello " + name);
}
```

5. Destructuring

Extract values from arrays or objects into variables easily.

Array destructuring:

```
let arr = [10, 20];

let [a, b] = arr;  // a=10, b=20
```

Object destructuring:

```
let person = { name: "Asha", age: 22 };

let { name, age } = person;
```

6. Spread Operator (...)

Used to spread array or object values.

```
let a = [1, 2];

let b = [...a, 3, 4];
```

Result: [1, 2, 3, 4]

7. Rest Operator (...)

Opposite of spread – it collects values.

```
function sum(...numbers) {
  console.log(numbers);
}
```

If you call:

sum(10, 20, 30);

Output: [10, 20, 30]

8. Classes

Way to create objects (OOP style).

```
class Student {
  constructor(name) {
    this.name = name;
  }
}
let s = new Student("Ravi");
```

9. Promises

Used for asynchronous work, like fetching data from a server.

```
let promise = new Promise((resolve, reject) => {
  resolve("Success");
});
```

10. Modules (import / export)

Allows us to split code into multiple files.

```
// file1.js
export let num = 10;


// file2.js
import { num } from "./file1.js";
```

8)DOM manipulation:

What is DOM?

DOM = Document Object Model

When a webpage loads, the browser converts the HTML into a tree-like structure called the DOM.

Example:

```
<html>
 <body>
  <h1>Hello</h1>
  <p>Welcome</p>
 </body>
</html>
```

The browser creates a DOM tree where each tag becomes a node/object.

JavaScript can select, change, add, or remove these HTML elements using DOM manipulation.

1. Selecting Elements (Very Important)

These methods help you get HTML elements so you can work with them.

✓ getElementById()

Selects element by ID.

```
let title = document.getElementById("myTitle");
```

✓ getElementsByClassName()

Returns all elements with the same class.

```
let items = document.getElementsByClassName("item");
```

✓ getElementsByTagName()

Select by tag name.

```
let buttons = document.getElementsByTagName("button");
```

✓ querySelector()

Selects the first matching element.

```
let element = document.querySelector(".box");
```

✓ querySelectorAll()

Selects all matching elements.

let allItems = document.querySelectorAll("li");

2. Changing HTML Content

✓ innerHTML

Changes or retrieves HTML inside a tag.

document.getElementById("msg").innerHTML = "Hello World!";

✓ textContent

Changes only text.

element.textContent = "Welcome!";

3. Changing CSS Styles

Use .style to modify CSS.

document.getElementById("box").style.backgroundColor = "red";

document.getElementById("box").style.fontSize = "20px";

4. Changing Attributes

Example: href, src, id, value, etc.

✓ setAttribute()

document.getElementById("img").setAttribute("src", "photo.jpg");

✓ getAttribute()

document.getElementById("img").getAttribute("src");

5. Adding Elements (Create)

✓ createElement()

let newPara = document.createElement("p");

newPara.textContent = "This is new!";

✓ Add the element to the webpage:

document.body.appendChild(newPara);

6. Removing Elements

✓ remove()

document.getElementById("title").remove();

✓ Remove child:

```
parent.removeChild(child);
```

7. Event Handling (Important in DOM)

Add events like click, mouseover, keypress.

✓ addEventListener()

```
document.getElementById("btn").addEventListener("click", function() {
   alert("Button clicked!");
});
```

8. Changing Form Values

```
document.getElementById("name").value = "Sumalatha";
```

9. Class Manipulation

✓ Add class

```
element.classList.add("active");
```

✓ Remove class

```
element.classList.remove("active");
```

✓ Toggle class

```
element.classList.toggle("dark-mode");
```

9) events in Js:

What Are Events in JavaScript?

An event is something that happens on a webpage.

Examples:

- Clicking a button
- Typing in an input box
- Moving the mouse
- Loading the page
- Submitting a form

JavaScript can detect these events and run functions when they happen.

This process is called event handling.

Common Events in JavaScript (VERY IMPORTANT)

Mouse Events

Event	Meaning
click	When user clicks
dblclick	Double click
mouseover	Mouse enters element
mouseout	Mouse leaves element
mousedown	Mouse button pressed
mouseup	Mouse button released
mousemove	Moving mouse

Keyboard Events

Event	Meaning
keydown	Key pressed
keyup	Key released
keypress	Key pressed (character keys only)

Form Events

Event	Meaning
submit	Form is submitted
change	Value changes (dropdown, input)
input	User types
focus	Input gets focus
blur	Input loses focus

Window Events

Event	Meaning
load	Page fully loaded
resize	Browser window resized
scroll	User scrolls the page

How to Add Events in JavaScript

There are 3 main ways:

✓ 1. HTML Event Attribute (Old method)

```
<button onclick="alert('Clicked!')">Click me</button>
```

✓ 2. JavaScript Property Method

```
let btn = document.getElementById("myBtn");
btn.onclick = function() {
  console.log("Button clicked!");
};
```

✓ 3. addEventListener() (Best & Modern Method)

```
let btn = document.getElementById("myBtn");

btn.addEventListener("click", function() {
  console.log("Button clicked!");
});
```

✓ Modern
✓ Can add multiple events
✓ Recommended

Event Object (event)

Every event gives an event object with information.

```
element.addEventListener("click", function(event) {
  console.log(event.type);  // click
});
```

Important Event Examples

✓ Click Event

```
btn.addEventListener("click", () => {
  console.log("Button clicked!");
});
```

✓ Mouseover Event

```
box.addEventListener("mouseover", () => {
  box.style.backgroundColor = "yellow";
});
```

✓ Keydown Event

```
document.addEventListener("keydown", (event) => {
  console.log("You pressed:", event.key);
});
```

✓ Input Event

```
input.addEventListener("input", () => {
  console.log("Typing...");
});
```

✓ Form Submit Event

```
form.addEventListener("submit", (e) => {
  e.preventDefault(); // stops page reload
  console.log("Form submitted");
});
```

10) Read and Write from Tags:

READ and WRITE From Tags in JavaScript (DOM Manipulation)

JavaScript interacts with HTML using the DOM (Document Object Model).

You can READ (get data) and WRITE (change data) from any HTML tag.

1. Reading From HTML Tags (GET)

You read values/content using:

A) innerHTML

Gets the HTML content inside a tag.

```
<p id="msg">Hello World</p>
<script>
 let text = document.getElementById("msg").innerHTML;
 console.log(text);  // Output: Hello World
</script>
```

B) innerText

Gets only text (no HTML tags).

```
<p id="p1">This is <b>bold</b></p>
<script>
 let t = document.getElementById("p1").innerText;
 console.log(t);  // Output: This is bold
</script>
```

C) value

Used for input, textarea, select elements.

```
<input id="name" value="Sumalatha">
<script>
 let n = document.getElementById("name").value;
 console.log(n); // Output: Sumalatha
</script>
```

2. Writing to HTML Tags (SET / UPDATE)

You can update/change the content using:

A) innerHTML

Sets new HTML content.

```
<p id="msg"></p>
<script>
 document.getElementById("msg").innerHTML = "Welcome!";
</script>
```

B) innerText

Changes only text.

```
<p id="para"></p>
<script>
 document.getElementById("para").innerText = "Hello User";
</script>
```

C) value

Updates input field value.

```
<input id="email">
<script>
 document.getElementById("email").value = "example@gmail.com";
</script>
```

3. Reading and Writing Attributes

getAttribute()

Read an attribute:

```
<img id="pic" src="photo.png">
<script>
 let src = document.getElementById("pic").getAttribute("src");
 console.log(src); // photo.png
</script>
```

setAttribute()

Write/change an attribute:

```
<script>
 document.getElementById("pic").setAttribute("src", "newImage.jpg");
```

</script>

4. Reading and Writing Style (CSS)

Read CSS style

```
<p id="p" style="color: red;"></p>
<script>
  console.log(document.getElementById("p").style.color);
</script>
```

Write CSS style

```
<script>
  document.getElementById("p").style.backgroundColor = "yellow";
  document.getElementById("p").style.fontSize = "20px";
</script>
```

5. Reading and Writing Class Names

Read class

```
document.getElementById("box").className;
```

Write class

```
document.getElementById("box").className = "active highlight";
```

6. Reading and Writing Using querySelector

Read

```
document.querySelector("#title").innerText;
```

Write

```
document.querySelector(".btn").innerText = "Clicked";
```