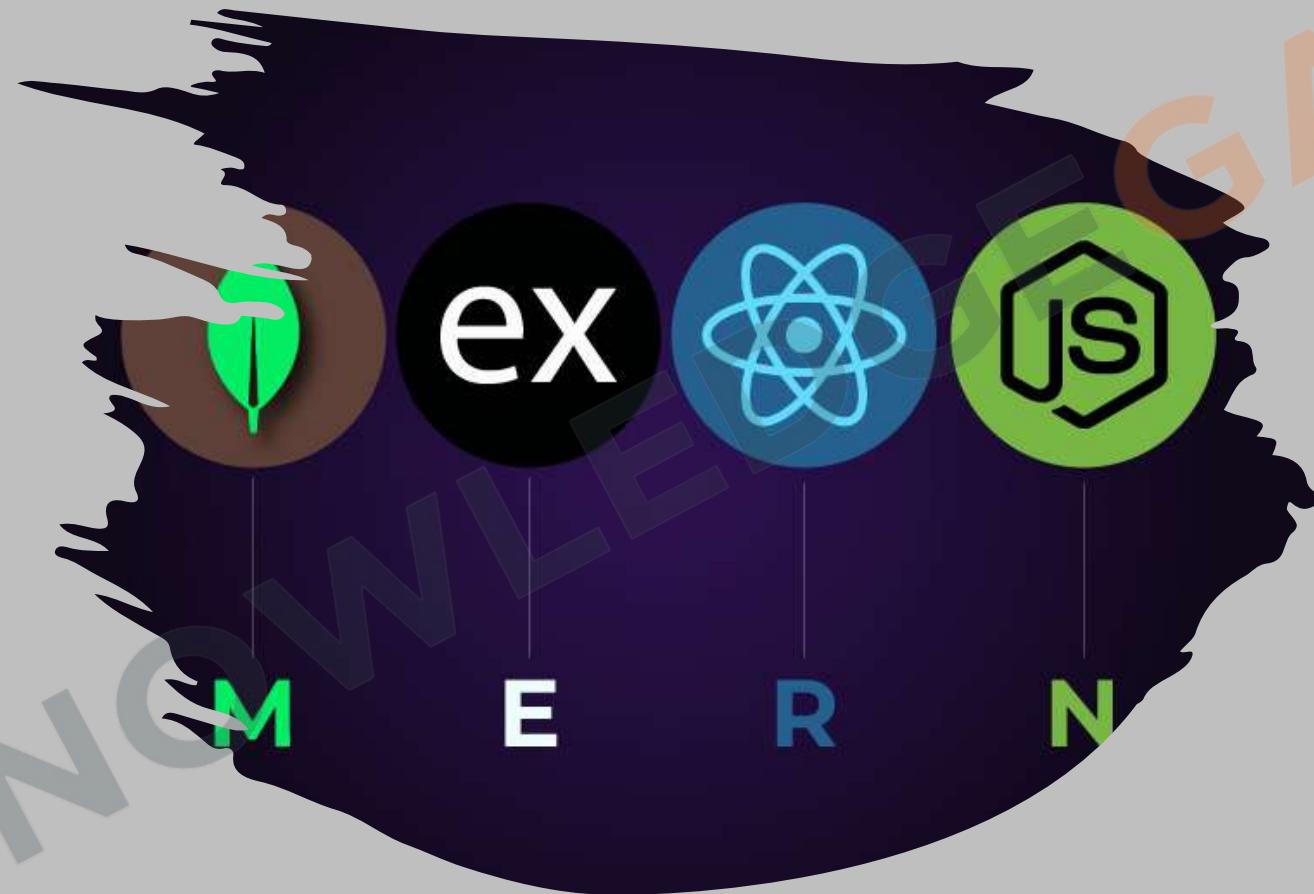


Introduction to **MERN Architecture**



e-KNOWLEDGE GATE

MERN



E

express



KNOWLEDGATE

FrontEnd / BackEnd / FullStack



**Client Side / Front-End
Web Development**

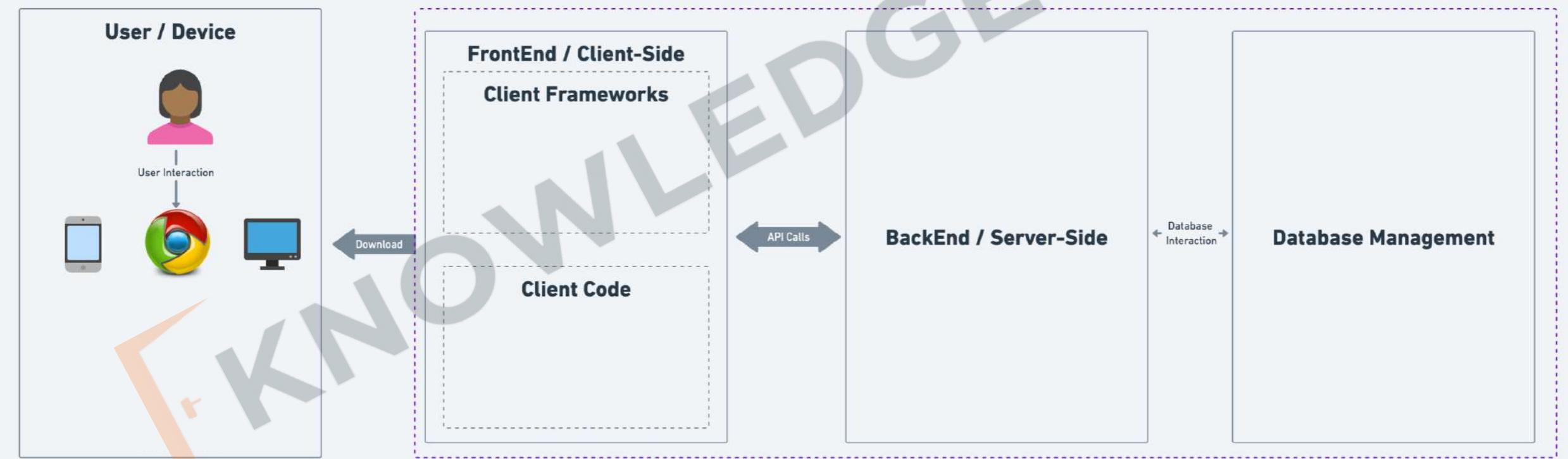


**Server Side
Back-End**



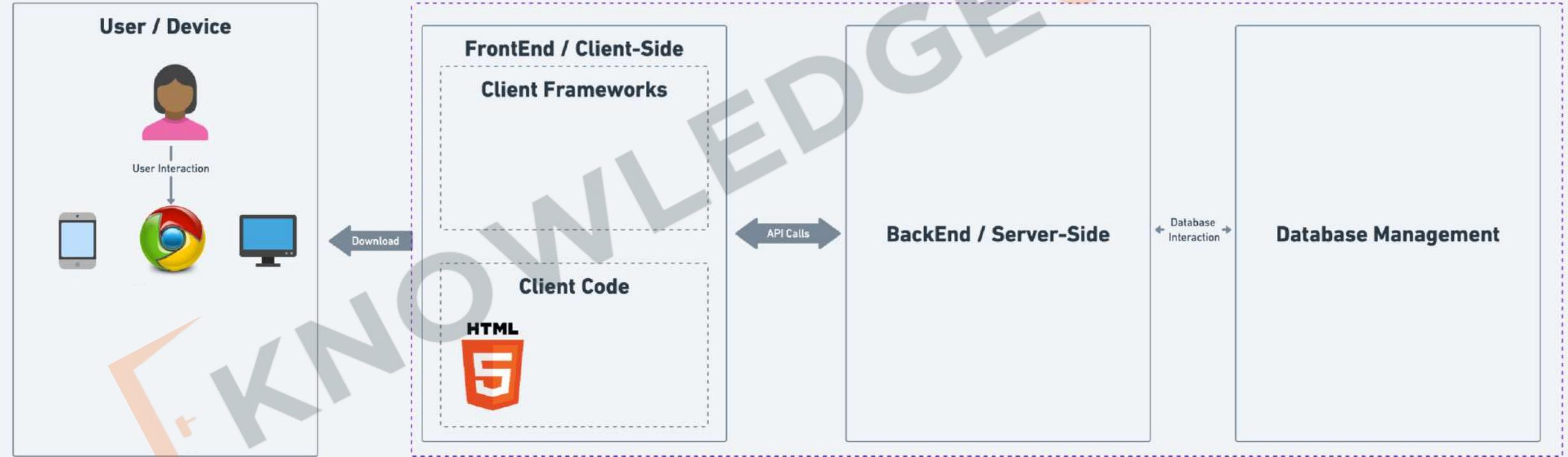
Full Stack

MERN Architecture



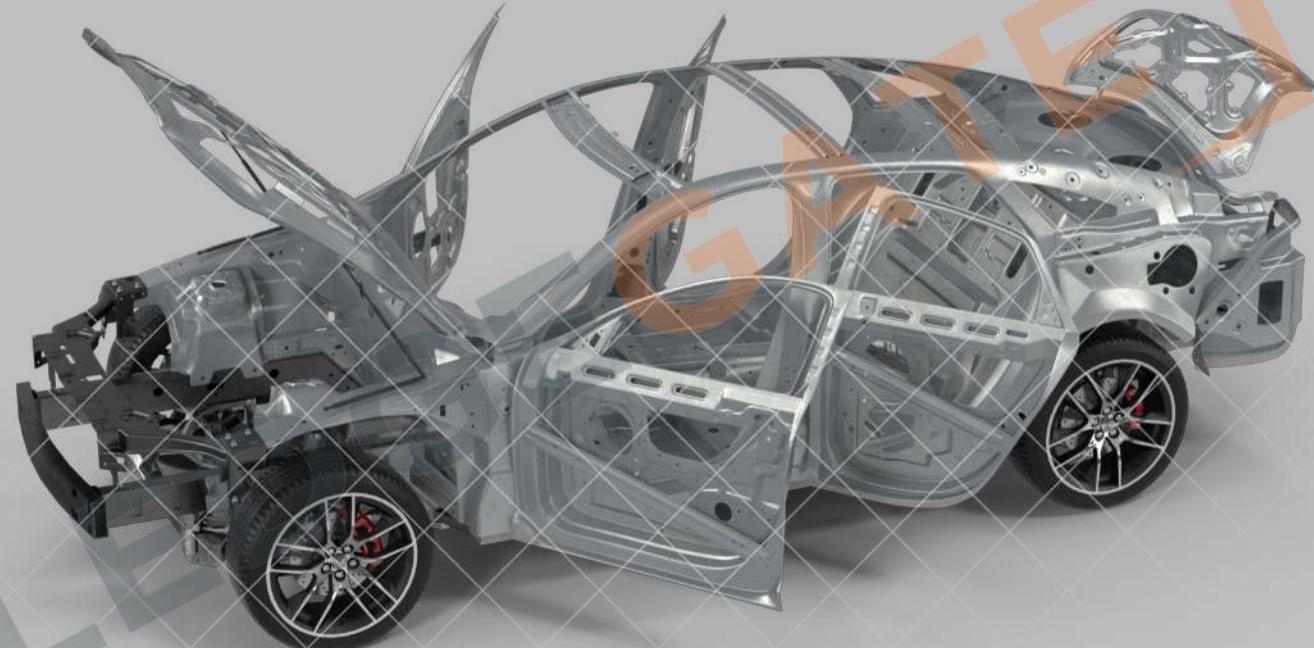
MERN Architecture

M E R N





(Hypertext Markup Language)



1. **Structure:** Sets up the layout.
2. **Content:** Adds text, images, links.
3. **Tags:** Uses elements like <p>, <a>.
4. **Hierarchy:** Organizes elements in a tree.

myntra.com

Mynta Insider New Gift CardTrack OrdersContact Us
Men

- [Topwear](#)
- [T-Shirts](#)
- [Casual Shirts](#)
- [Formal Shirts](#)
- [Sweatshirts](#)
- [Sweaters](#)
- [Jackets](#)
- [Blazers & Coats](#)
- [Suits](#)
- [Rain Jackets](#)
- [Indian & Festive Wear](#)
- [Kurtas & Kurta Sets](#)
- [Sherwanis](#)
- [Nehru Jackets](#)
- [Dhotis](#)

- [Bottomwear](#)
- [Jeans](#)
- [Casual Trousers](#)
- [Formal Trousers](#)
- [Shorts](#)
- [Track Pants & Joggers](#)
- [Innerwear & Sleepwear](#)
- [Briefs & Trunks](#)
- [Boxers](#)
- [Vests](#)
- [Sleepwear & Loungewear](#)
- [Thermals](#)
- [Plus Size](#)

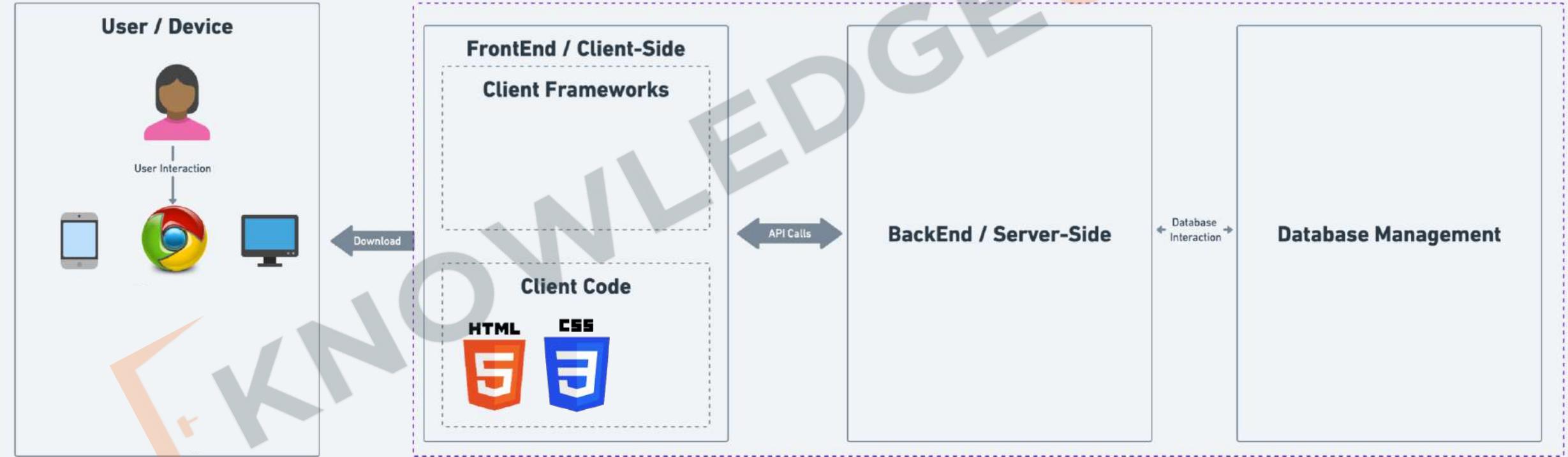


Myntra

only HTML

KNOWLEDGE GATE

MERN Architecture



css



(Cascading Style Sheets)

1. **Style:** Sets the look and feel.
2. **Colors & Fonts:** Customizes text and background.
3. **Layout:** Controls position and size.
4. **Selectors:** Targets specific **HTML** elements.



MEN WOMEN KIDS HOME & LIVING BEAUTY STUDIO NEW

Search for products, brands and more

Profile Wishlist Bag

FLAT ₹400 OFF

On Your 1st Purchase
Via Myntra App!

Her ➤

 100% ORIGINAL PRODUCTS ₹ EASY RETURNS & REFUNDS 100% SECURE PAYMENTSFASHION
CARNIVAL

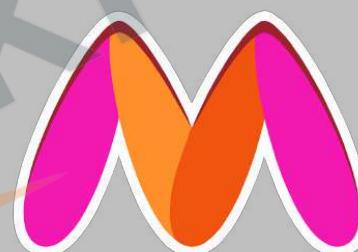
MAY 2-9

Your Summer Style Refresh

50-80% OFF

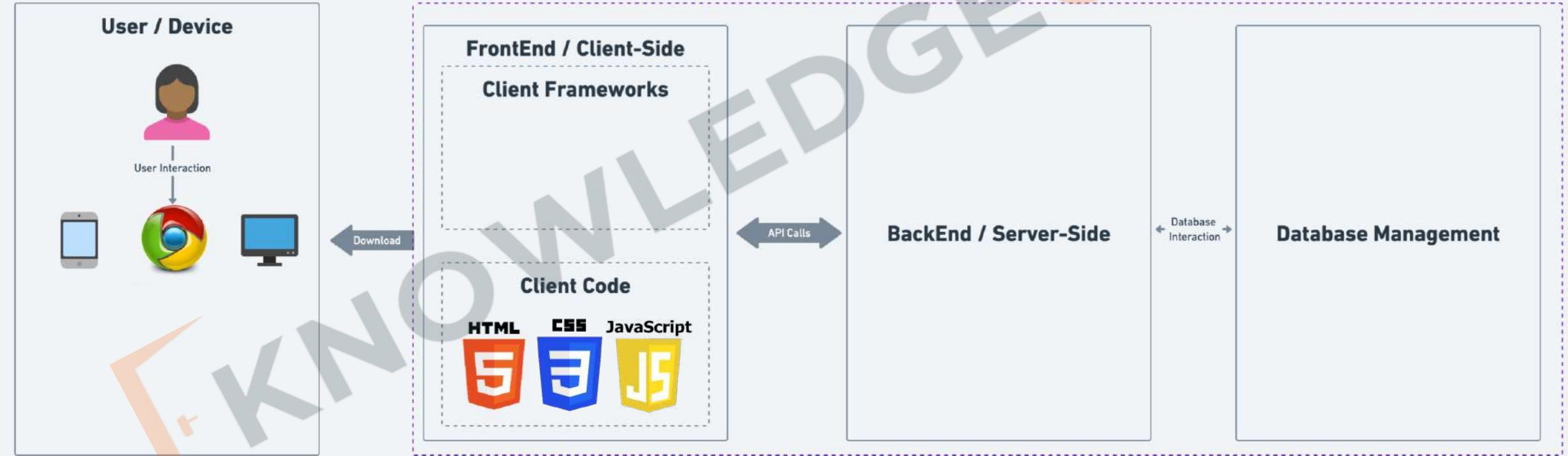
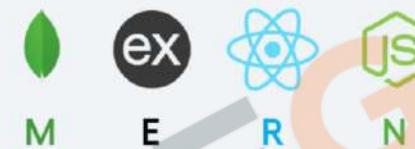
Him ➤

Hot Coupons Alert!

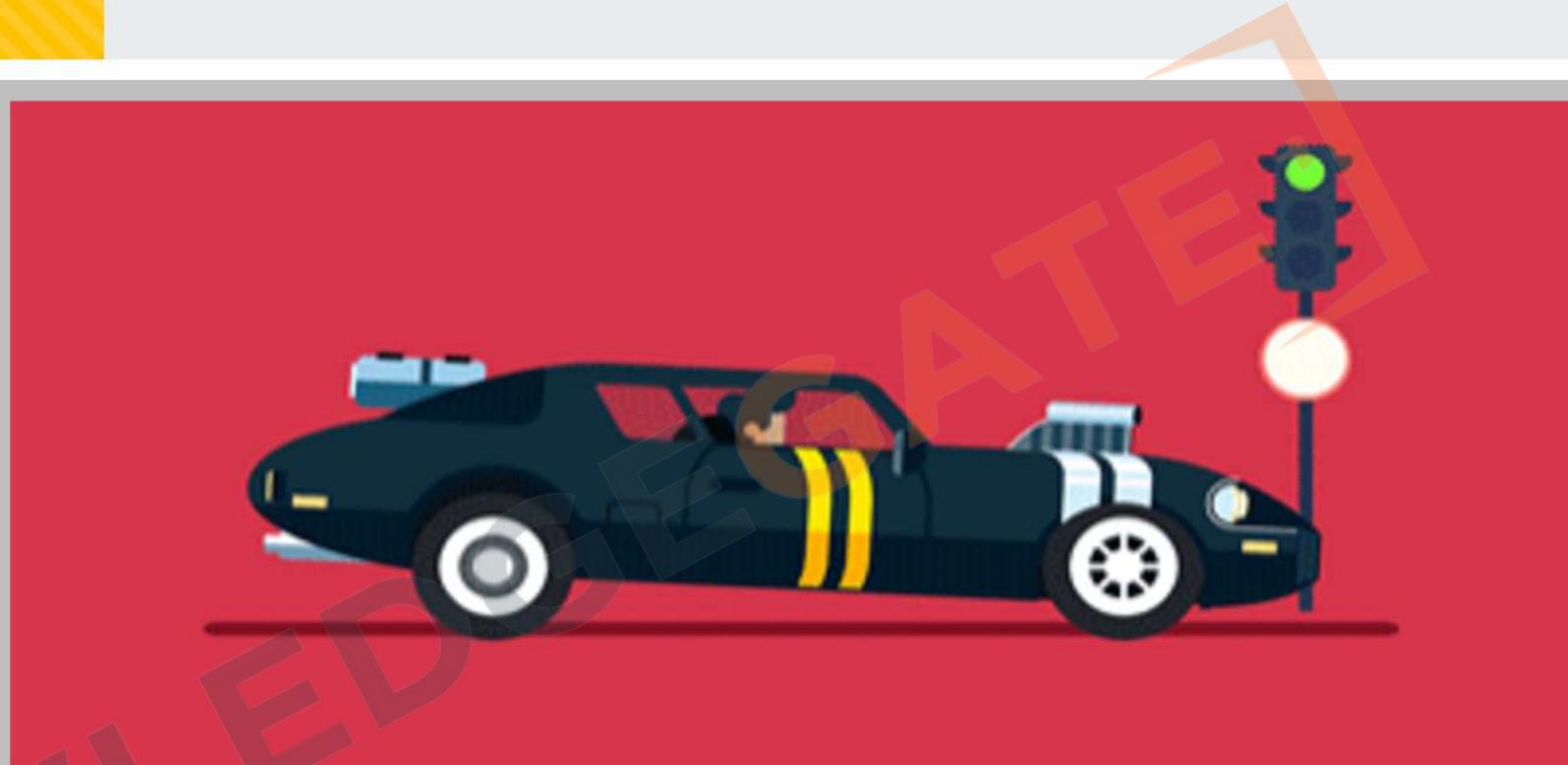
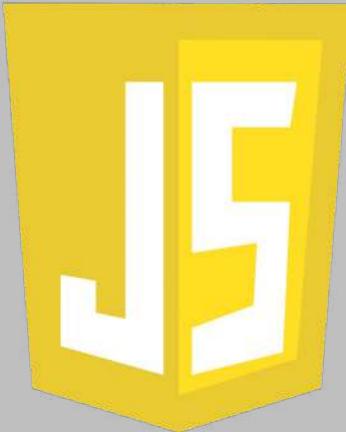


Myntra with CSS

MERN Architecture



JavaScript



1. JavaScript has nothing to do with Java
2. **Actions:** Enables interactivity.
3. **Updates:** Alters page without reloading.
4. **Events:** **Responds** to user actions.
5. **Data:** Fetches and sends info to server.

HTML 10%

CSS 10%

JavaScript 20%

myntra.com

MEN WOMEN KIDS HOME & LIVING BEAUTY STUDIO NEW

FLAT ₹400 OFF



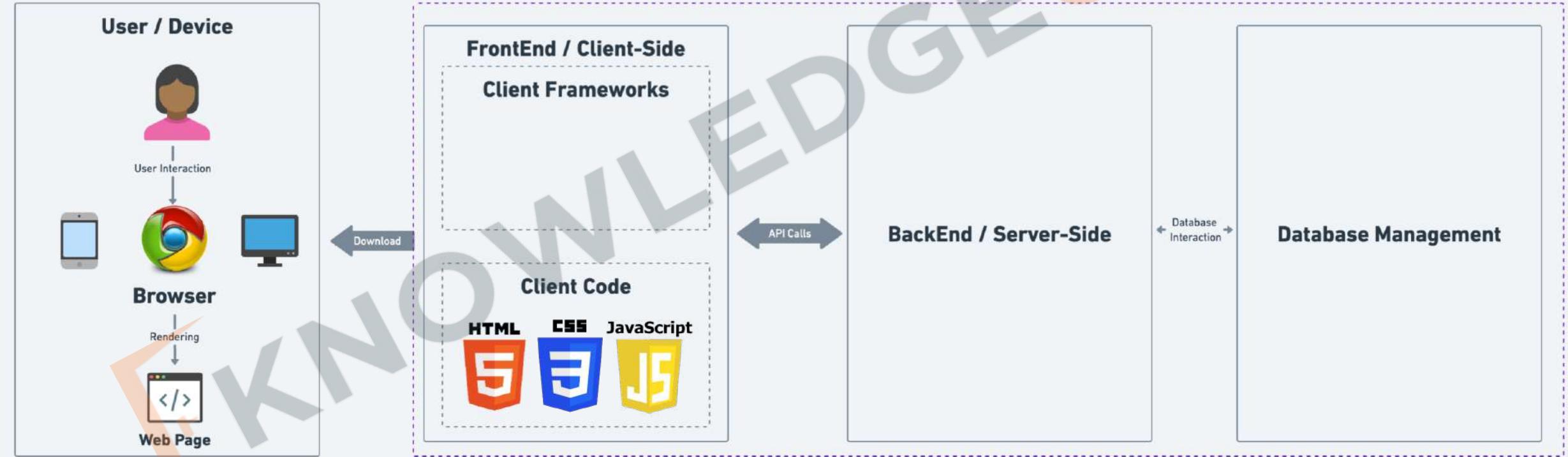
Myntra with JS

Role of Browser



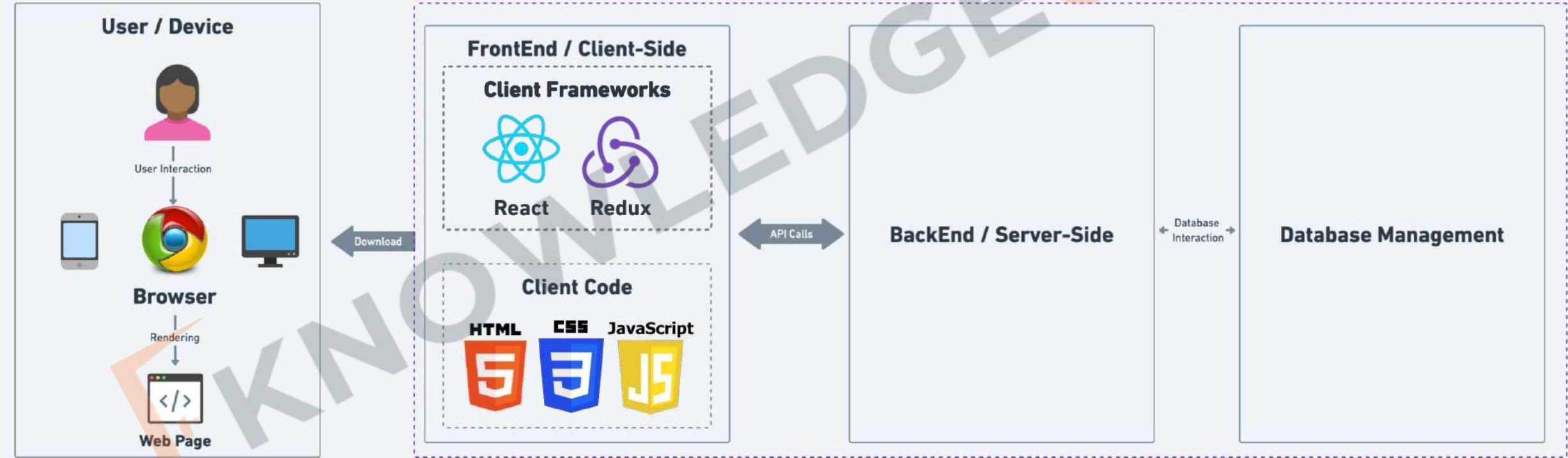
1. **Displays Web Page:** Turns HTML code into what you see on screen.
2. **User Clicks:** Helps you interact with the web page.
3. **Updates Content:** Allows changes to the page using JavaScript.
4. **Loads Files:** Gets HTML, images, etc., from the server.

MERN Architecture



MERN Architecture

M E R N



What is ReactJS



1. **JavaScript** library to build **Dynamic** and **interactive** user interfaces
2. Developed at **Facebook** in **2011**.
3. Currently **most widely used JS library** for front-end development.
4. **Used** to create **single page application** (page does not re-load).

HTML 10%

CSS 10%

JavaScript 20%

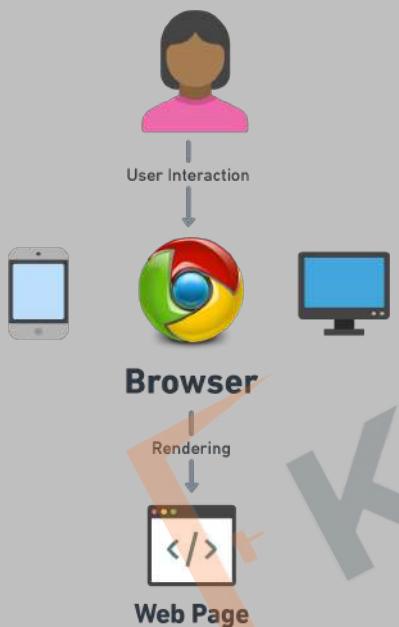
ReactJS 20%

NodeJS 20%

MERN Architecture



User / Device



FrontEnd / Client-Side

Client Frameworks



React Redux

Client Code



HTML CSS JavaScript



HTML CSS JavaScript

BackEnd / Server-Side

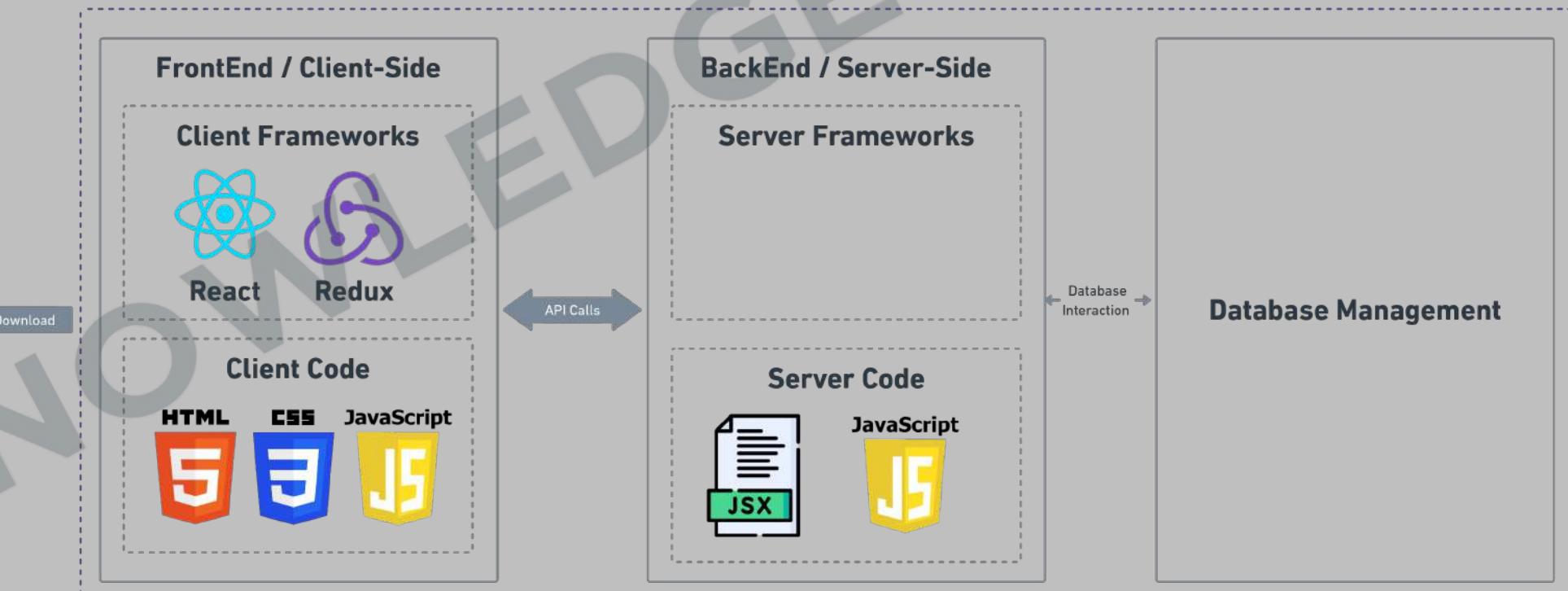
Server Frameworks



JSX JavaScript

Database Management

Database
Interaction



HTML 10%

CSS 10%

JavaScript 20%

ReactJS 20%

NodeJS 20%

MERN Architecture



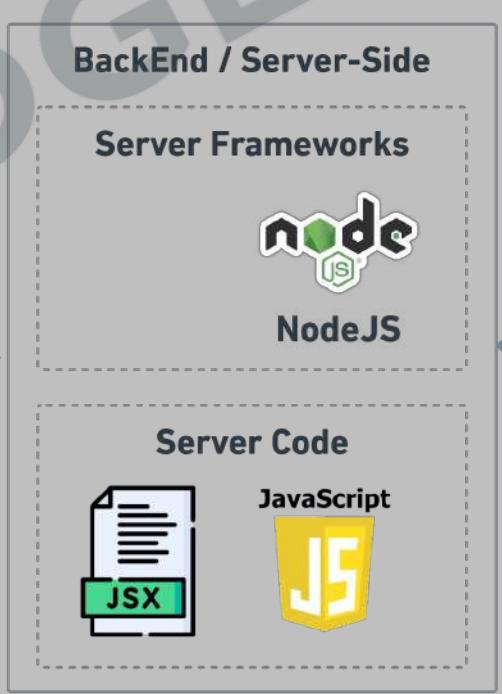
User / Device



FrontEnd / Client-Side

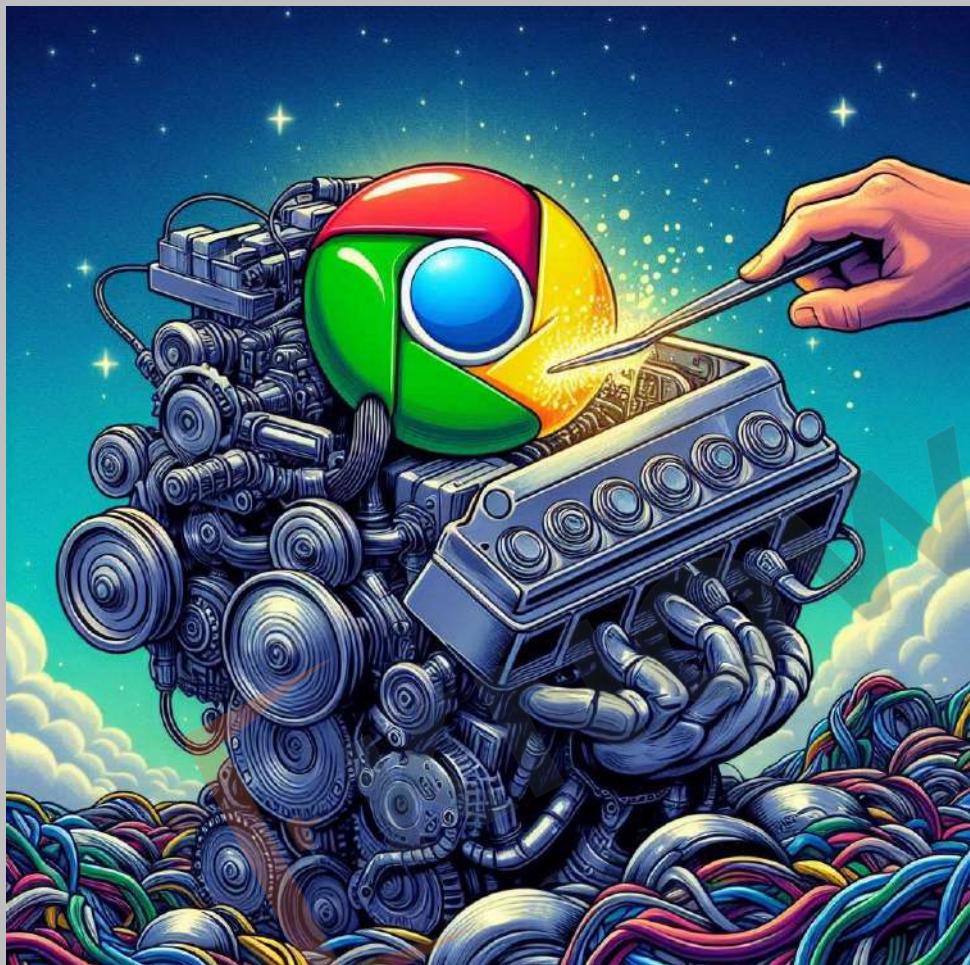


BackEnd / Server-Side



Database Management

What is NodeJs



- 1. JavaScript Runtime:** Node.js is an [open-source, cross-platform](#) runtime environment for executing JavaScript code outside of a browser.
- 2. Built on Chrome's V8 Engine:** It runs on the V8 engine, which [compiles JavaScript directly to native machine code](#), enhancing performance.
- 3. Design:** Features an event-driven, non-blocking I/O model for efficiency.
- 4. Full-Stack JavaScript:** Allows using JavaScript on both server and client sides.
- 5. Scalability:** Ideal for scalable network applications due to its architecture.
- 6. Versatility:** Suitable for web, real-time chat, and REST API servers.

HTML 10%

CSS 10%

JavaScript 20%

ReactJS 20%

NodeJS 20%

ExpressJS 10%

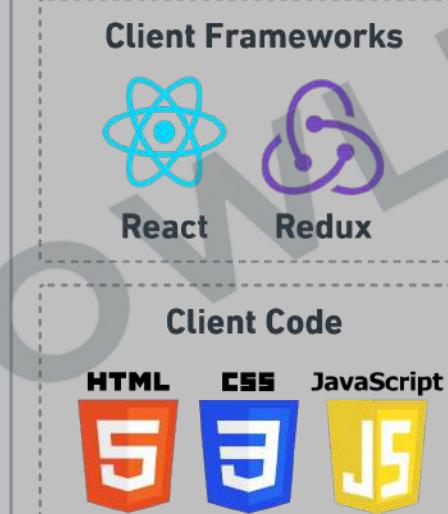
MERN Architecture



User / Device



FrontEnd / Client-Side



BackEnd / Server-Side



Database Management

What is ExpressJS

- 
- 1. Web Framework:** Express.js is a minimalist web framework for Node.js, designed for building web applications and APIs.
 - 2. Function:** Utilizes [middleware](#) for flexible request handling.
 - 3. Simplicity:** Streamlines web application development.
 - 4. Flexibility:** Customizable with extensive middleware support.
 - 5. Routing:** Efficient management of [HTTP routes](#).
 - 6. Speed:** Offers performance without compromise.

HTML 10%

CSS 10%

JavaScript 20%

ReactJS 20%

NodeJS 20%

ExpressJS 10%

MongoDB 10%

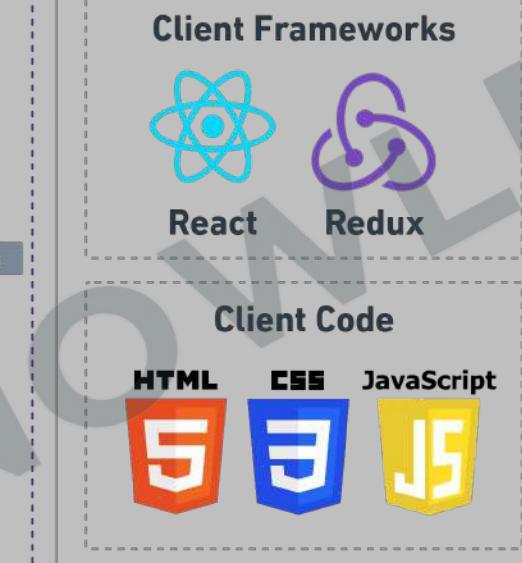
MERN Architecture



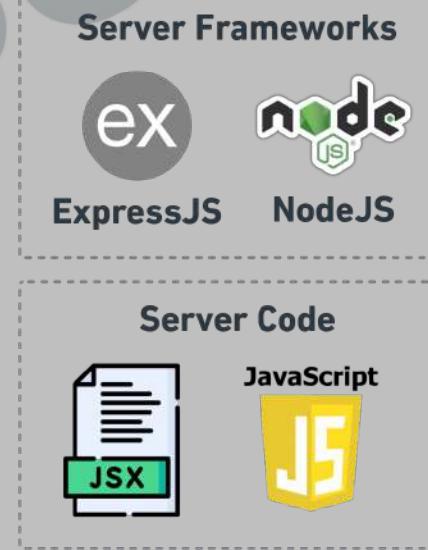
User / Device



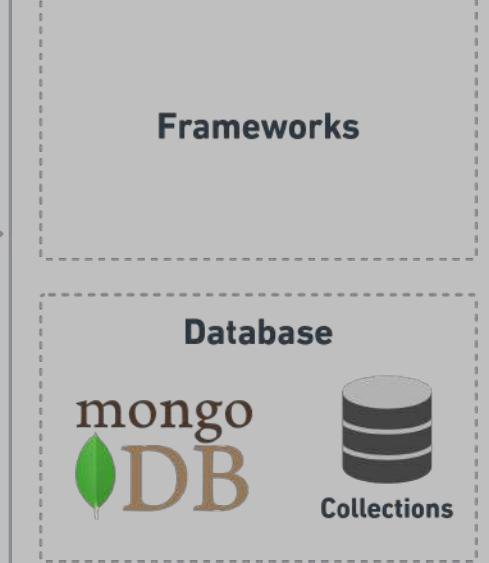
FrontEnd / Client-Side



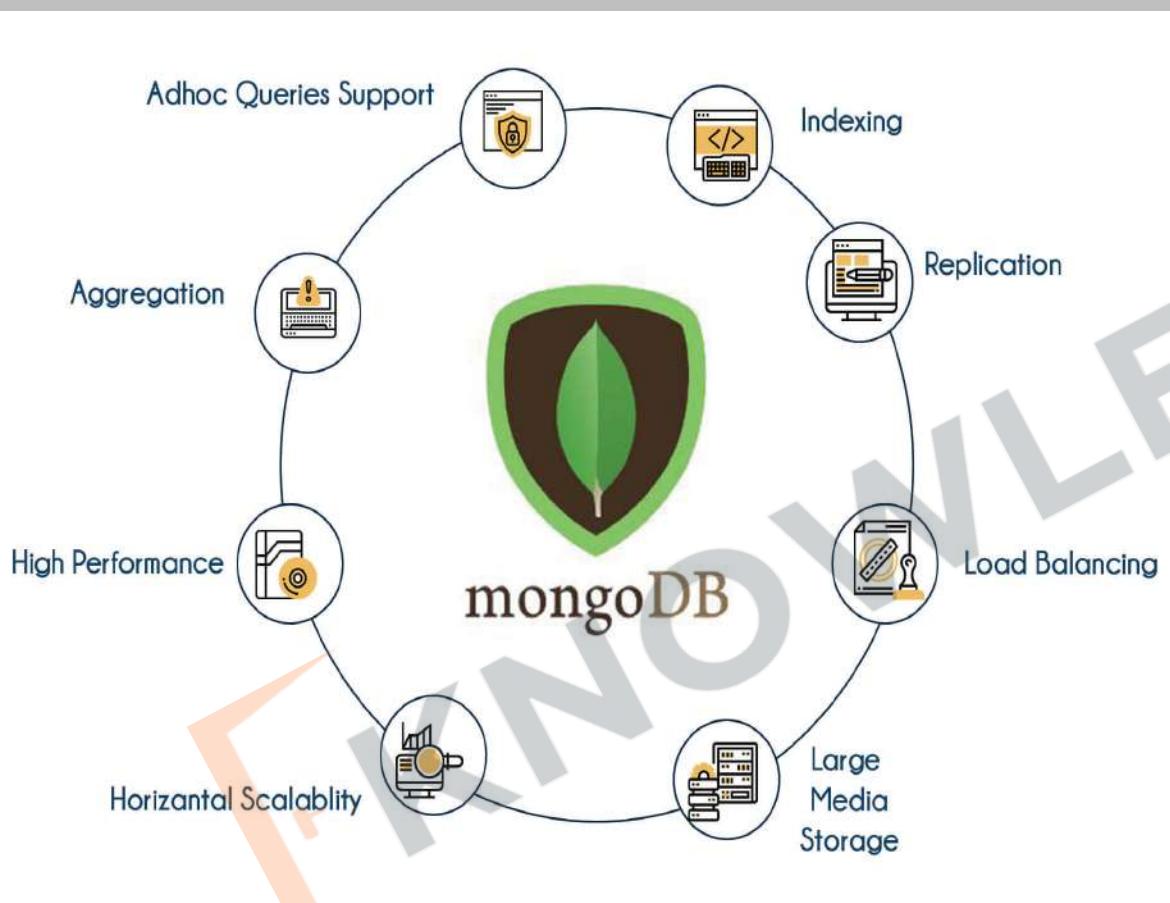
BackEnd / Server-Side



Database Management

Database
Interaction

What is MongoDB



- 1. NoSQL Database:** MongoDB is a **document-oriented NoSQL database** used for high volume data storage.
- 2. Document Model:** Instead of tables and rows, MongoDB uses collections and documents that provide flexibility and allow data to be stored in **JSON-like formats**.
- 3. Scalability:** Supports **horizontal scaling** with sharding.
- 4. Flexible Schema:** Adapts easily to **varied data structures**.
- 5. Performance:** Optimized for **quick reads and writes**.
- 6. Querying:** Offers advanced querying capabilities.

HTML 10%

CSS 10%

JavaScript 20%

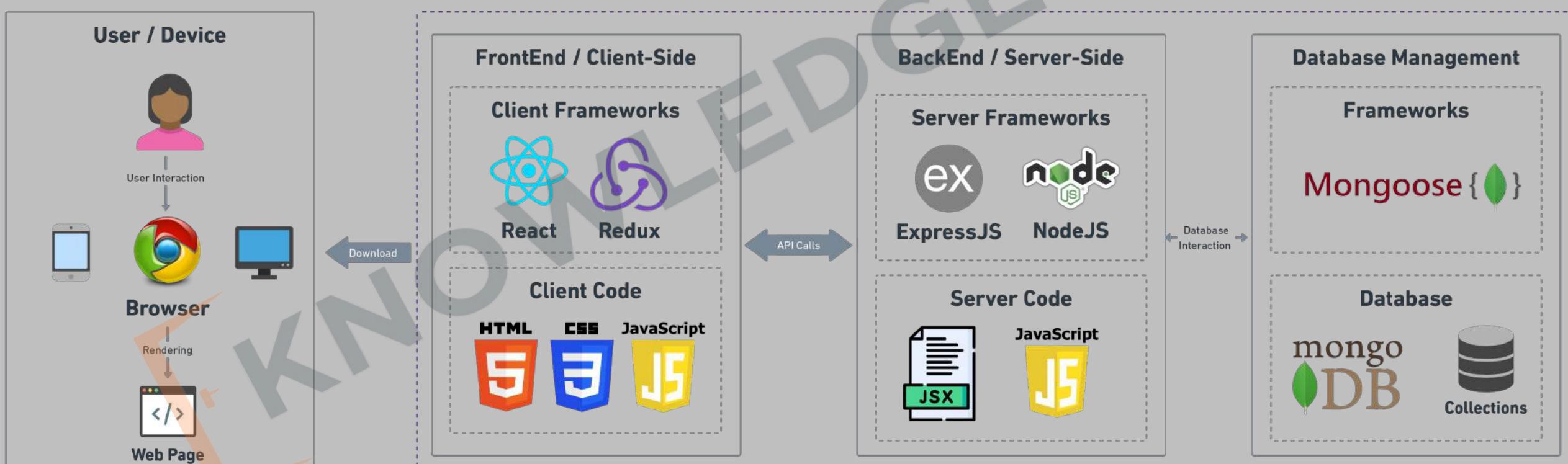
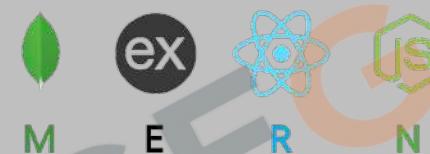
ReactJS 20%

NodeJS 20%

ExpressJS 10%

MongoDB 10%

MERN Architecture



Software Deployments



- Process:** Software deployment involves **delivering a software product** to a user or system environment.
- Stages:** Includes **all stages from release to active use**, encompassing installation, configuration, running, and updating of software.
- Availability:** Ensures software is **available for use by target users** or systems.

EVOLUTION OF SOFTWARE DEPLOYMENT

Requires manual configuration and can be costly and inflexible, making it challenging for larger, complex applications.



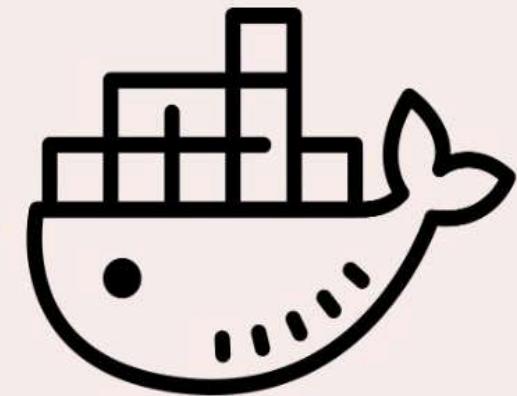
Physical machine

Allow scalability and isolated deployment of applications, but can be resource-intensive and time-consuming when boot up.



Virtual machines

Providing a flexible way to deploy, scale and manage resources. Challenges include networking and additional tools.

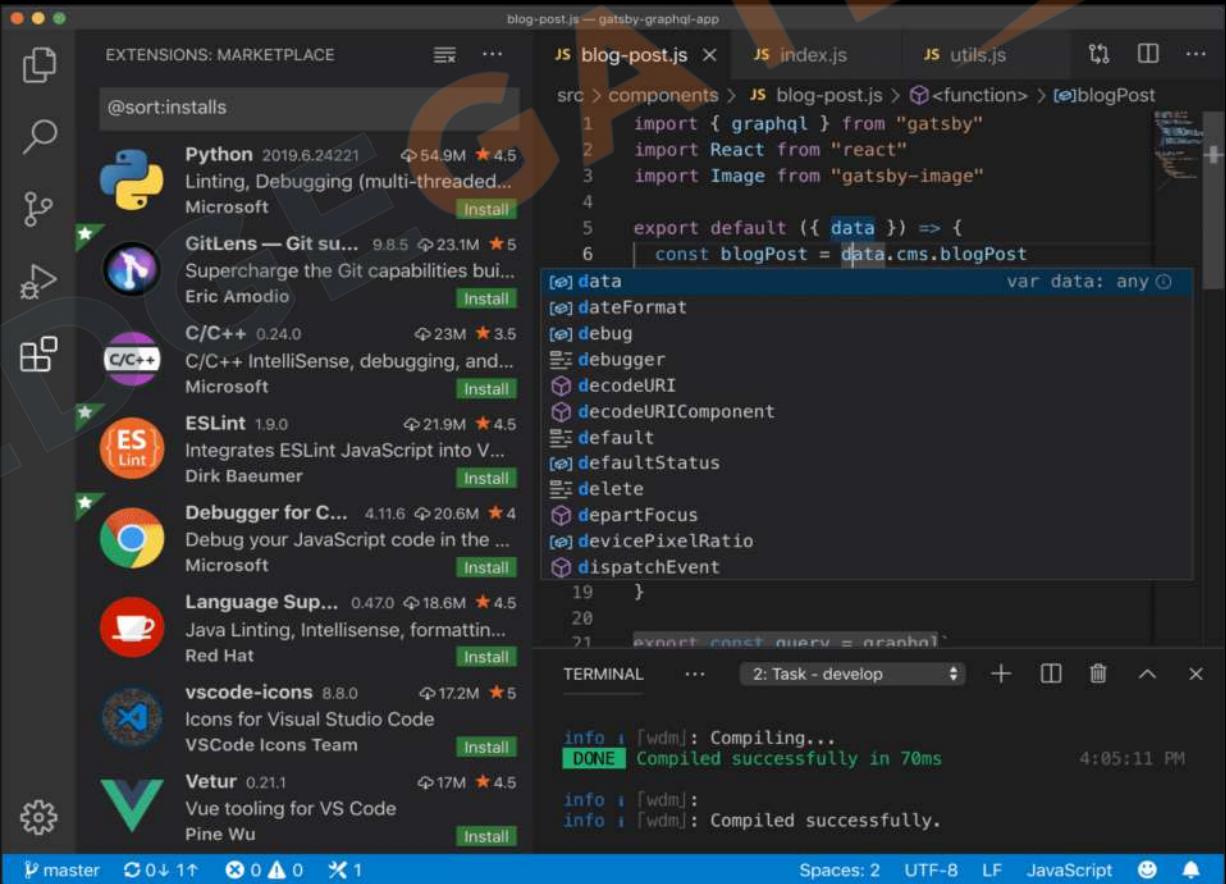


Containerization



IDE Setup

IDE
OR
Code Editor



What is IDE

1. IDE stands for Integrated Development Environment.
2. Software suite that consolidates basic tools required for software development.
3. Central hub for coding, finding problems, and testing.
4. Designed to improve developer efficiency.



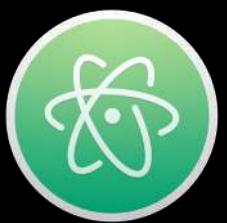
Need of IDE

1. Streamlines development.
2. Increases productivity.
3. Simplifies complex tasks.
4. Offers a unified workspace.
5. IDE Features
 1. Code Autocomplete
 2. Syntax Highlighting
 3. Version Control
 4. Error Checking



```
MainActivity.kt
```

```
@Composable
fun MessageCard(msg: Message) {
    Row(modifier = Modifier.padding(all = 8.dp)) {
        Image(
            painter = painterResource(R.drawable.android_studio_logo),
            contentDescription = "Profile Picture",
            modifier = Modifier
                .size(45.dp)
        )
        Spacer(modifier = Modifier.width(8.dp))
        Column(modifier = Modifier
            .background(color = Color.White)) {
            Text(text = msg.author, color = Color.Black)
            Spacer(modifier = Modifier.height(1.dp))
            Text(text = msg.body, color = Color.Black)
        }
    }
}
```



IDE Selection

1. Sublime Text
2. Atom
3. VS Code
4. Github CodeSpaces



The screenshot shows the Visual Studio Code interface. On the left, the Extensions Marketplace sidebar is open, displaying a list of extensions such as Python, GitLens, C/C++, ESLint, Debugger for Chrome, Language Support, vscode-icons, and Vetur. The main code editor area shows a file named 'blog-post.js' with some JavaScript code. Below the editor, the terminal window shows the output of a compilation process, indicating success. The status bar at the bottom provides information about the current branch ('master'), file counts, and other settings.



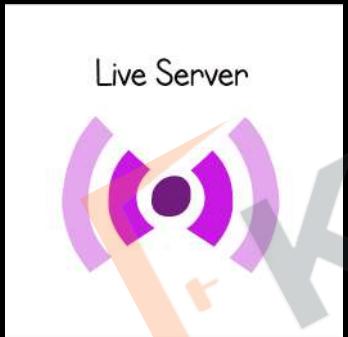
Installation & Setup

1. Search VS Code
2. Keep Your **Software** up to date



VsCode Extensions

1. Live Server / Live Preview
2. Prettier (Format on Save)
3. Line Wrap
4. Tab Size from 4 to 2



Client Side vs Server Side

	Client Side	Server Side
Execution Location	Executes on user's device.	Executes on a remote machine.
Languages	Primarily JavaScript, HTML, CSS.	PHP, Python, Java, Node.js, etc.
Main Job	Makes clicks and scrolls work	Manages saved information
Access Level	Can't access server data directly	Can read/write files, interact with databases.
Speed	Quicker for UI	Slower due to network latency

KNOWLEDGEAGATE

HTML Basics



Starting
Up

KNOWLEDGE RATE

First file using Text Editor

1. Create a folder with name **First Project** on your Desktop.
2. Open **Notepad**.
3. Create a file and save it as **index.html**
4. Copy Sample code
5. Open **Browser** and Check.



File Extension

HTML

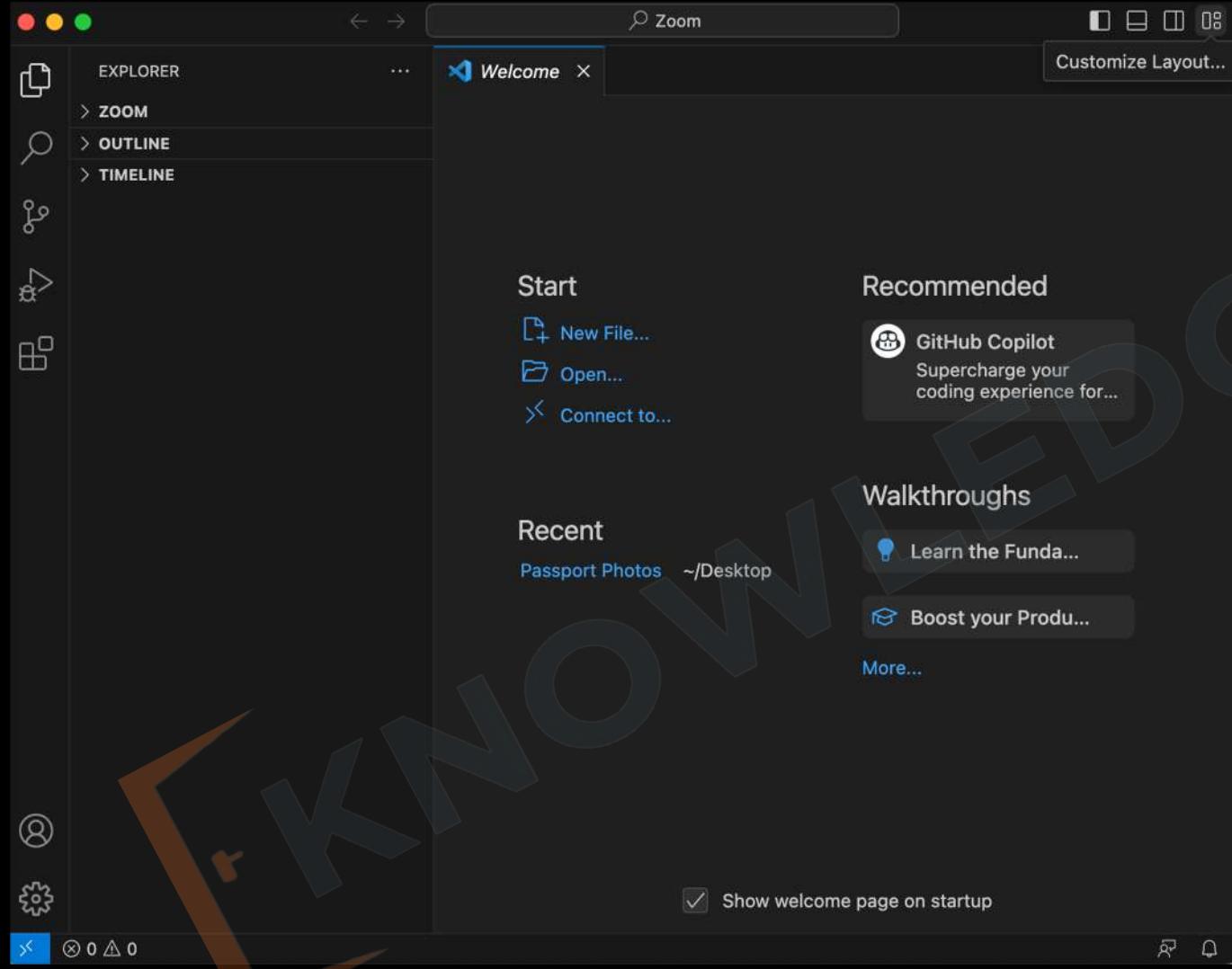
1. Most commonly used.
2. Works across all browsers.
3. Widely recognized and supported.
4. Typically saved as .html.



HTM

1. Less commonly used.
2. Originated for compatibility with older systems.
3. Works same as .html.
4. Typically saved as .htm.

Opening project in VsCode



Importance of index.html

1. Default name of a website's homepage.
2. First page users see when visiting a website
3. Important for SEO (Search Engine Optimization)
4. Provides uniform starting point across servers
5. Serves as fallback when no file is specified in URL



HTML Basics

Basics of
HTML



What are Tags

1. Elements that are used to create a website are called HTML Tags.
2. Tags can contain content or other HTML tags.
3. Define elements like text, images, links



Using Emmet ! to generate code

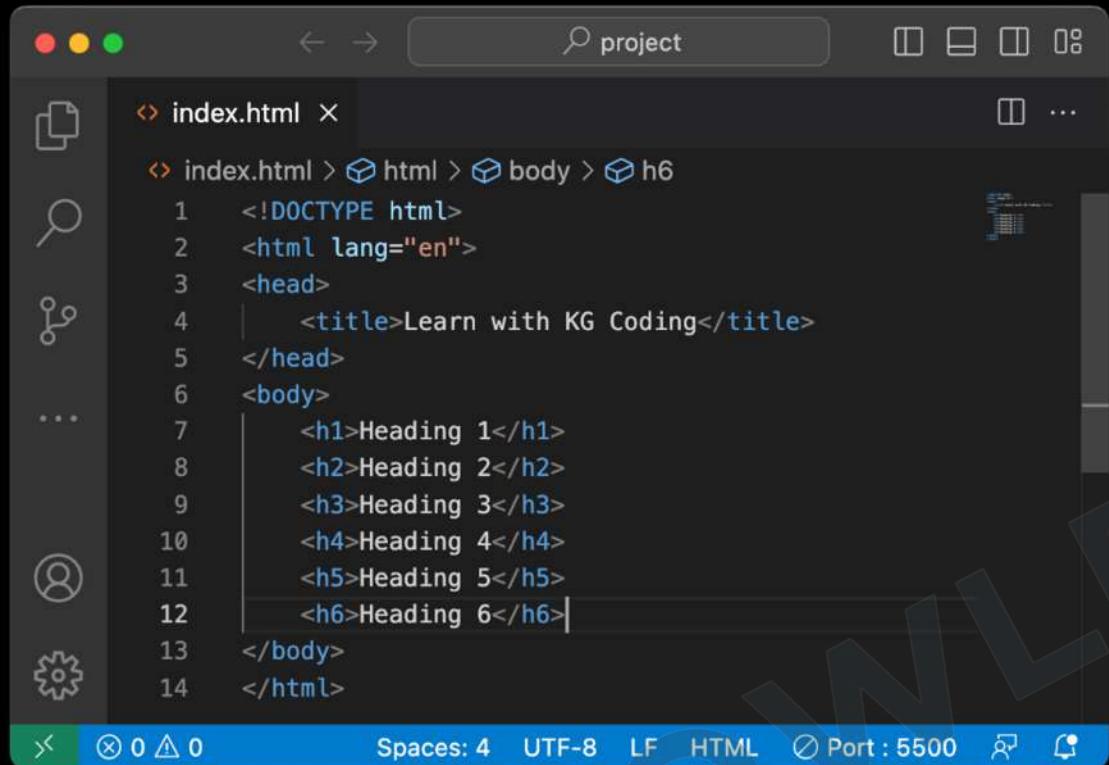


1. Type ! and wait for suggestions.

Basic HTML Page

```
<!DOCTYPE html>           Defines the HTML Version  
  
<html lang="en">          Parent of all HTML tags / Root element  
  
    <head>                  Parent of meta data tags  
        <title>My First Webpage</title>  Title of the web page  
    </head>  
  
    <body>                  Parent of content tags  
        <h1>Hello World!</h1>  Heading tag  
    </body>  
</html>
```

Heading Tag



A screenshot of a code editor window titled "project". The file "index.html" is open, showing the following code:

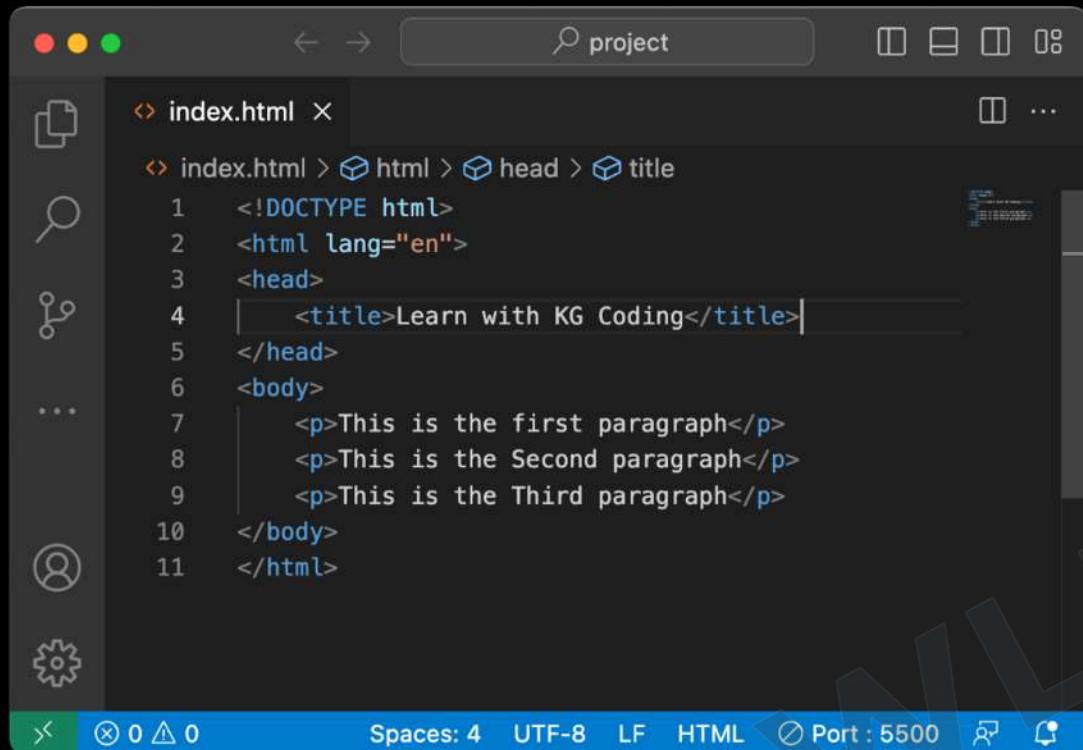
```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <h1>Heading 1</h1>
    <h2>Heading 2</h2>
    <h3>Heading 3</h3>
    <h4>Heading 4</h4>
    <h5>Heading 5</h5>
    <h6>Heading 6</h6>
</body>
</html>
```

The code editor interface includes a sidebar with icons for file operations, a search bar, and a bottom status bar indicating "Spaces: 4", "UTF-8", "LF", "HTML", "Port : 5500", and other developer tools.



1. Defines **headings** in a document
2. Ranges from **<h1>** to **<h6>**
3. **<h1>** is most important, **<h6>** is least
4. Important for **SEO**
5. Helps in structuring content

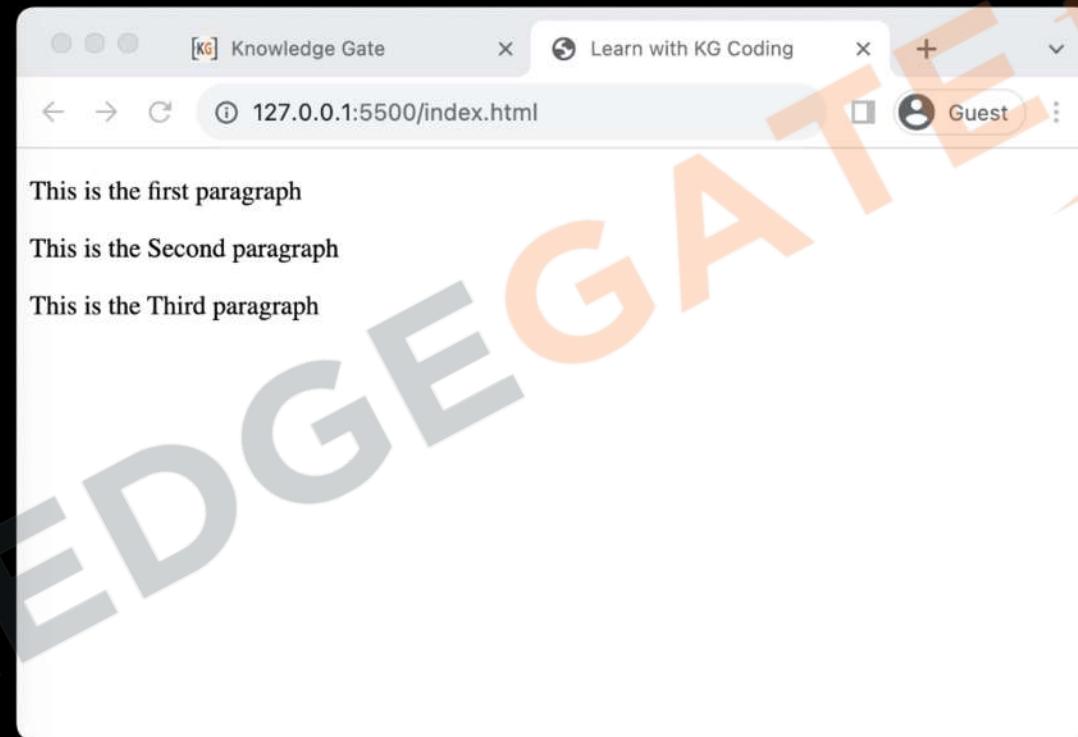
Paragraph Tag



A screenshot of a code editor window titled "project". The file "index.html" is open, showing the following code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <p>This is the first paragraph</p>
    <p>This is the Second paragraph</p>
    <p>This is the Third paragraph</p>
</body>
</html>
```

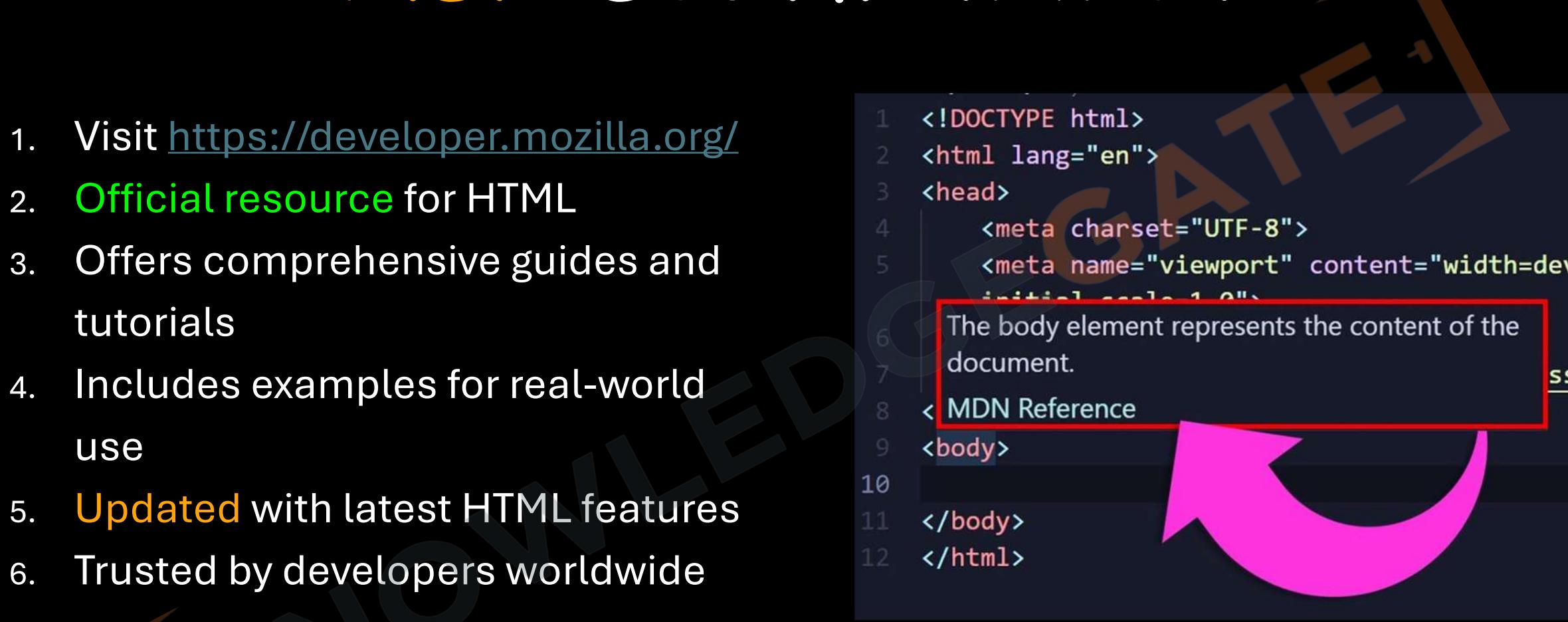
The code editor has a dark theme with light-colored syntax highlighting. The status bar at the bottom shows "Spaces: 4", "UTF-8", "LF", "HTML", "Port : 5500", and a refresh icon.



1. Used for defining **paragraphs**
2. Enclosed within **<p>** and **</p>** tags
3. Adds **automatic spacing** before and after
4. **Text wraps** to next line inside tag
5. Common in text-heavy content

MDN Documentation

1. Visit <https://developer.mozilla.org/>
2. Official resource for HTML
3. Offers comprehensive guides and tutorials
4. Includes examples for real-world use
5. Updated with latest HTML features
6. Trusted by developers worldwide



```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <!-- Global site tag (gtag.js) - Google Analytics -->
7      <!-- Global site tag (gtag.js) - Google Analytics -->
8      <!-- Global site tag (gtag.js) - Google Analytics -->
9      <!-- Global site tag (gtag.js) - Google Analytics -->
10     <!-- Global site tag (gtag.js) - Google Analytics -->
11     </body>
12     </html>
```

The body element represents the content of the document.

MDN Reference



Comments

1. Used to add **notes** in HTML code
2. **Not displayed** on the web page
3. Syntax: **<!-- Comment here -->**
4. Helpful for code organization
5. Can be **multi-line** or **single-line**

Writing comments in HTML

Single-line Comment

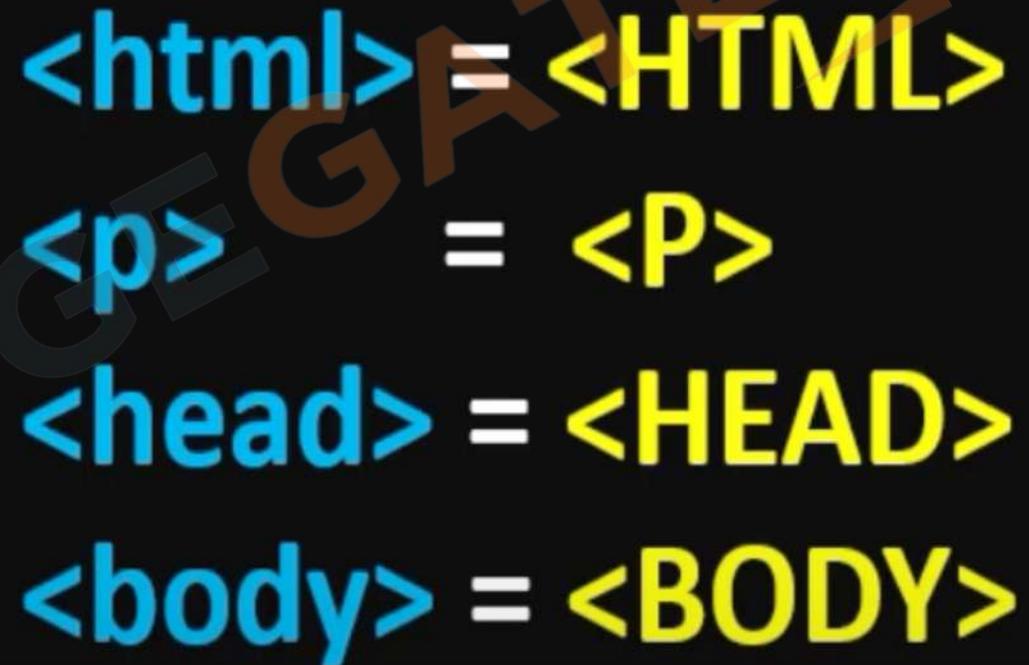
```
1 <!--This is a single line  
comment in HTML. You cannot  
see it on a webpage. Click  
on view-source to see a  
message I left just for you.  
-->
```

Multi-line Comment

```
1 <!-- This is a multi-line  
comment in HTML.  
2 You cannot see it on a  
webpage.  
3 If you view-source on the  
browser you can see the  
comment there.-->
```

Case Sensitivity

1. HTML is case-insensitive for tag names
2. Attribute names are also be case-insensitive
3. Best practice: use lowercase for consistency



`<html> = <HTML>`
`<p> = <P>`
`<head> = <HEAD>`
`<body> = <BODY>`

Practice Exercise

HTML Basics

1. Create a new project with `Index.html`
2. Generate boilerplate code using Emmet
3. Write “I am learning with **Prashant sir**”
4. Use **comments**
5. Also use Case insensitive tags



Practice Exercise

HTML Basics

Assignment: Adding Comments and Case Sensitivity

- **Objective:** Use comments and demonstrate case sensitivity in HTML.
- **Instructions:**
 - Open your index.html file in VS Code.
 - Add a comment at the top of the file: <!-- This is my first HTML project -->.
 - Use various tags in different cases (e.g., <H1>, <p>, <DIV>,).
 - Ensure that the HTML is still valid and displays correctly in the browser.
 - Add another comment explaining why HTML is case-insensitive but best practice is to use lowercase.

Practice Exercise

HTML Basics

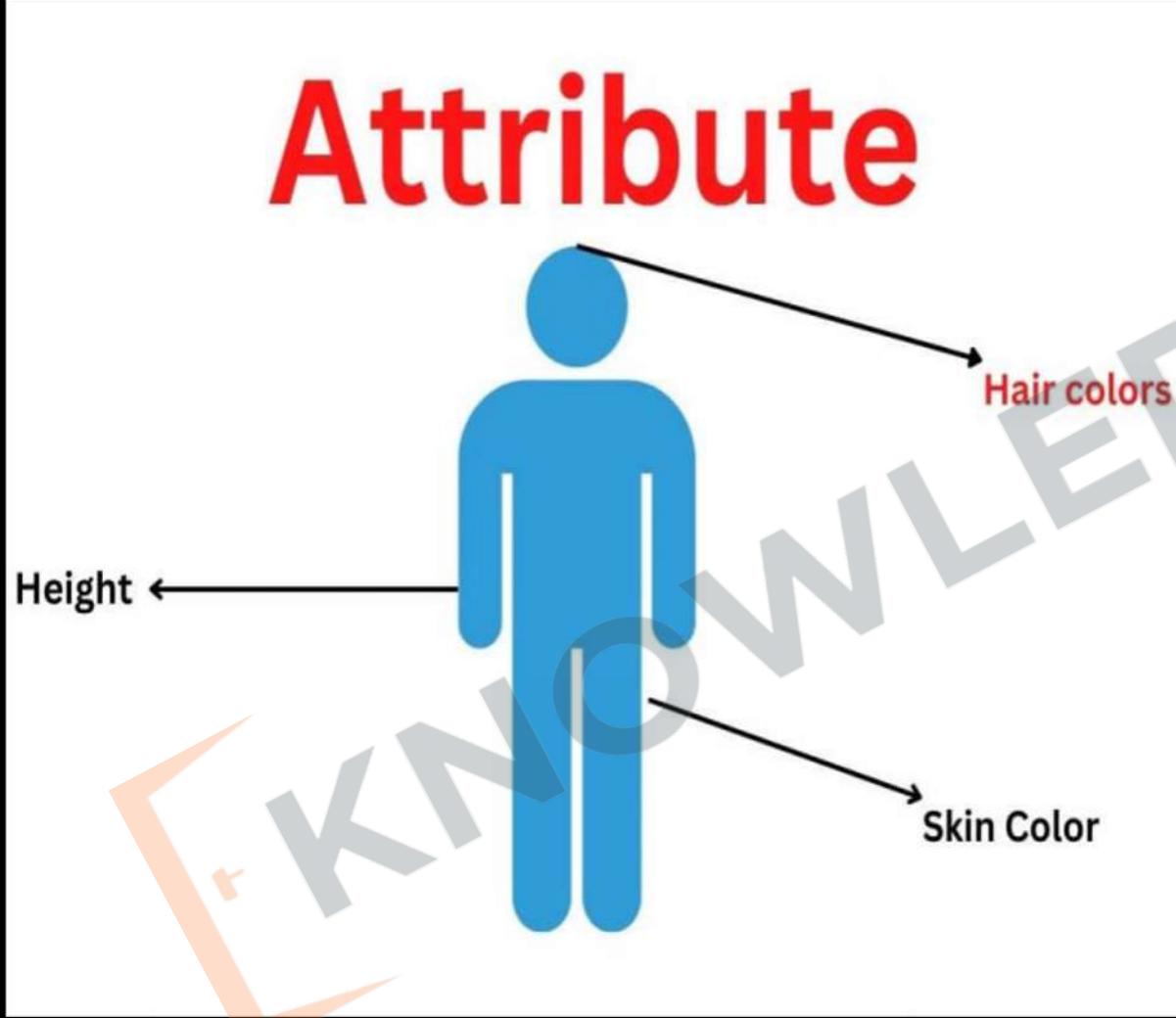
Assignment:

- Objective: Utilize MDN documentation for HTML.
- Instructions:
 - Visit [MDN Web Docs](#).
 - Find the documentation for the `<h1>` tag.
 - Write a short summary of what you learned about the `<h1>` tag from the documentation.
 - Include an example usage of the `<h1>` tag in your `index.html` file.



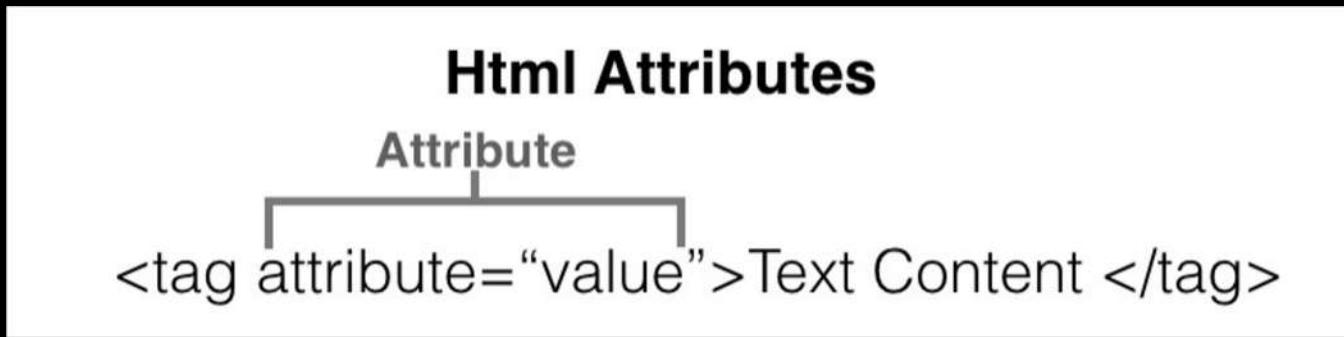
Must-Use HTML Tags

Attribute



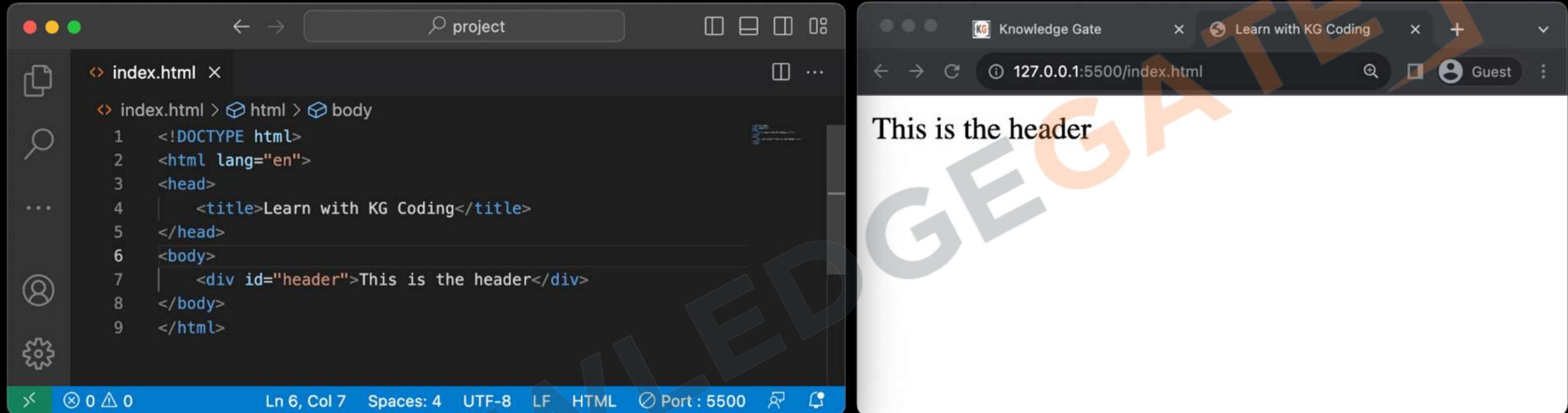
HTML
Attributes

What are HTML Attributes?



1. Provides additional information about elements
2. Placed within opening tags
3. Common examples: href, src, alt
4. Use name=value format
5. Can be single or multiple per element

id property



The image shows a code editor on the left and a browser window on the right. The code editor displays the following HTML code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <div id="header">This is the header</div>
</body>
</html>
```

The browser window shows the rendered HTML with the text "This is the header" displayed inside a red-bordered div element.

- **Unique Identifier:** Each id should be unique within a page.
- **Anchoring:** Allows for direct links to sections using the `#id` syntax in URLs.
- **CSS & JavaScript:** Used for selecting elements for styling or scripting.

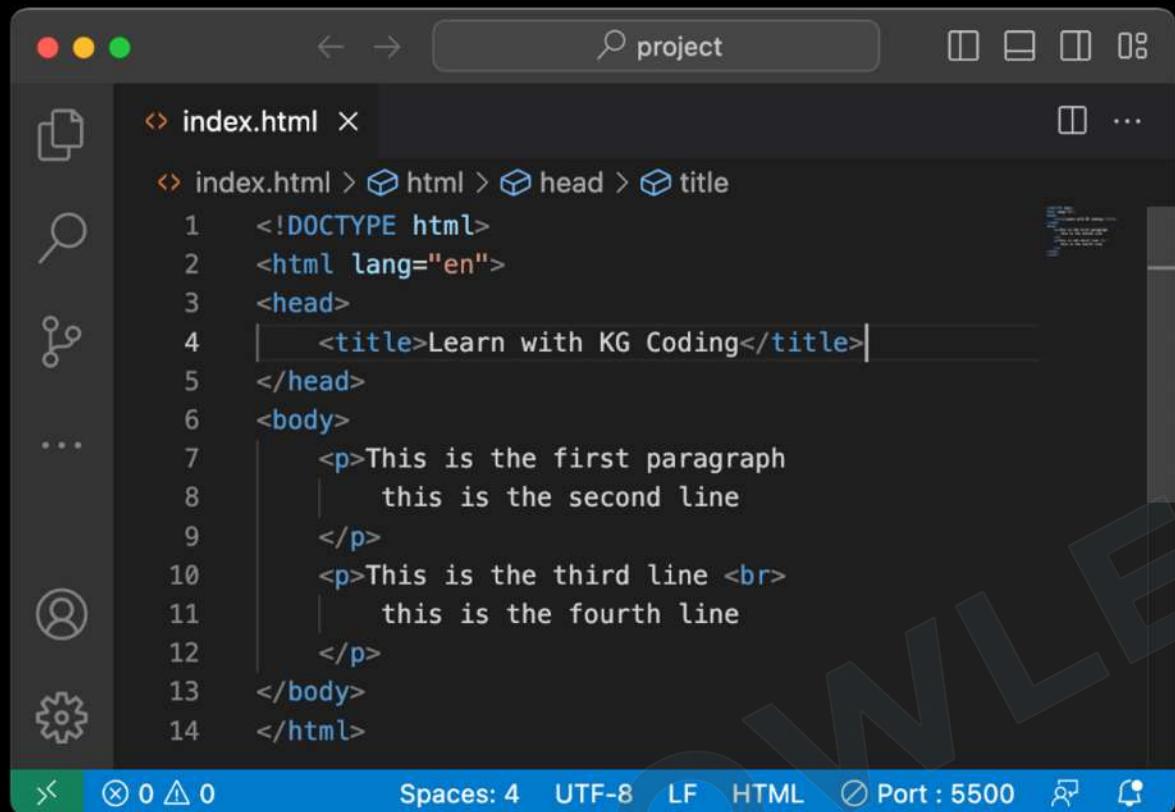
Must-Use HTML Tags



HTML Tags

EDGEGATE

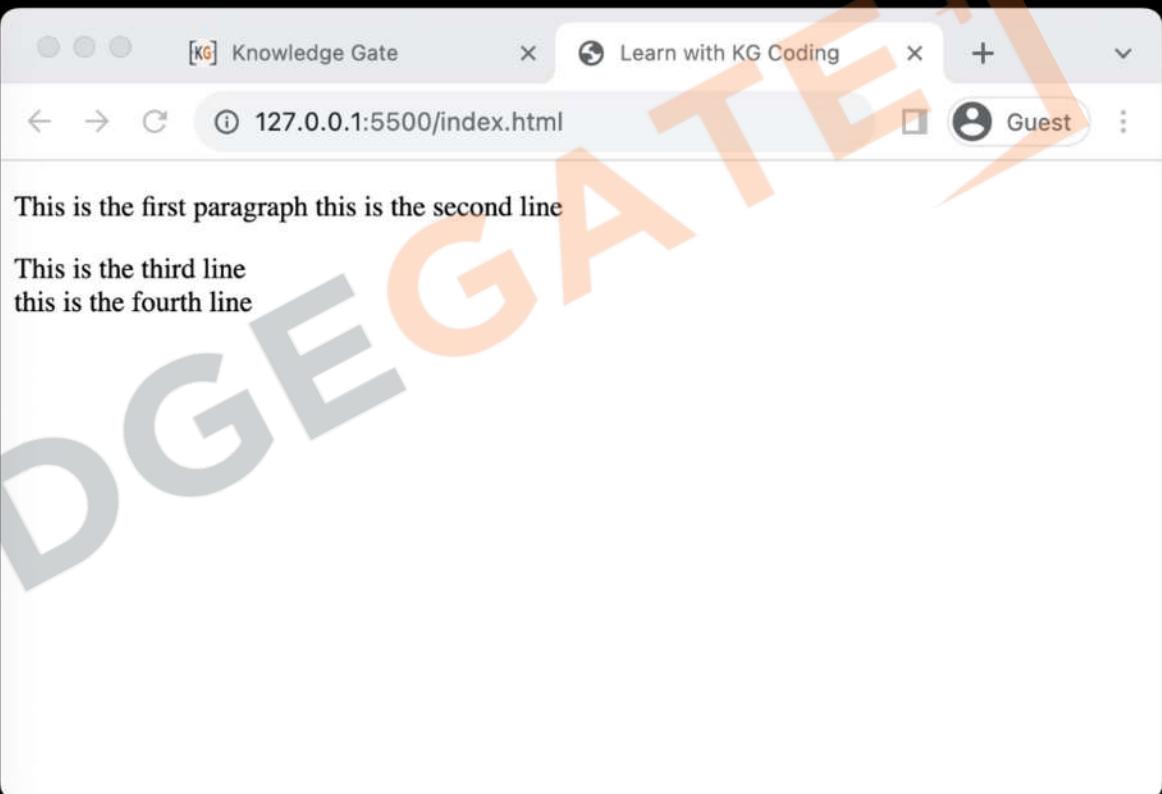
 Tag



A screenshot of a code editor window titled "project". The file "index.html" is open, showing the following code:

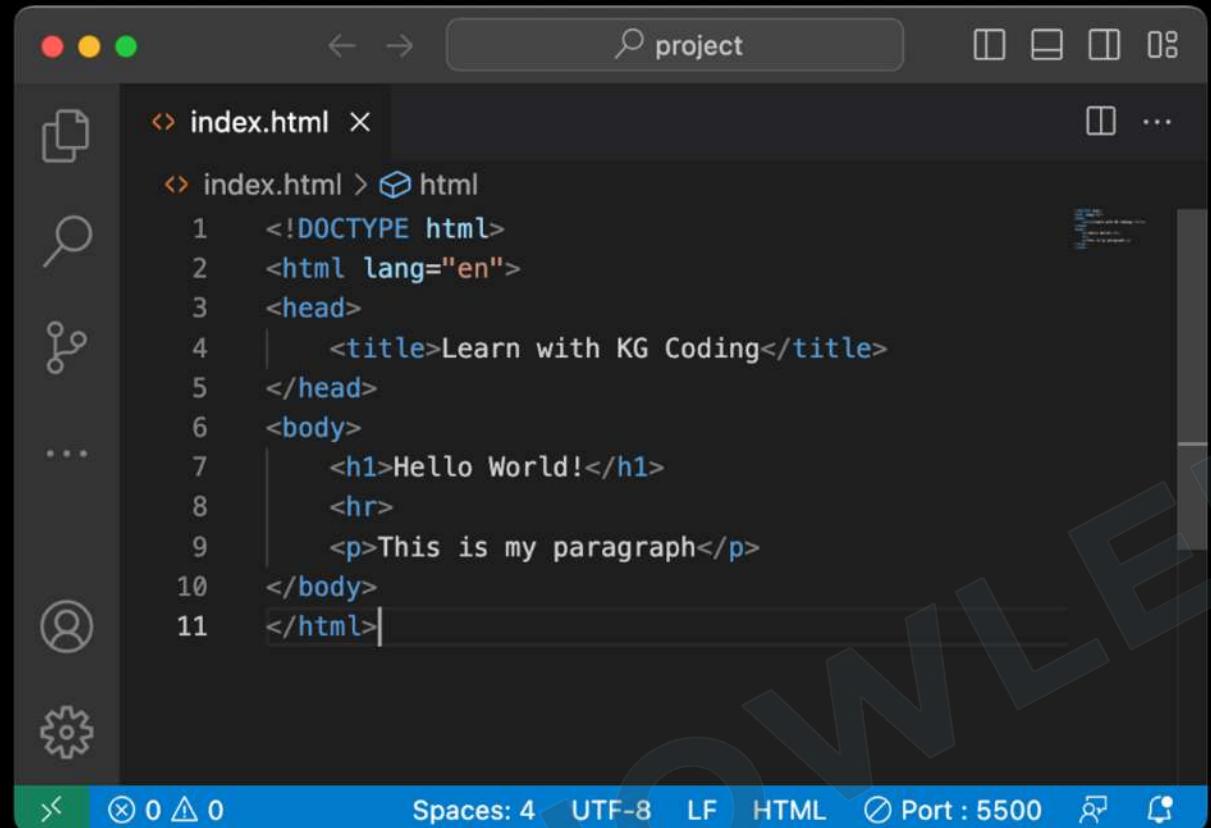
```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <title>Learn with KG Coding</title>
5  </head>
6  <body>
7      <p>This is the first paragraph
8          this is the second line
9      </p>
10     <p>This is the third line <br>
11         this is the fourth line
12     </p>
13 </body>
14 </html>
```

The code editor has a dark theme with light-colored text. The status bar at the bottom shows "Spaces: 4", "UTF-8", "LF", "HTML", and "Port : 5500".



1.
 adds a line break within text
2.
 is empty, no closing tag needed
3.
 and
 are both valid

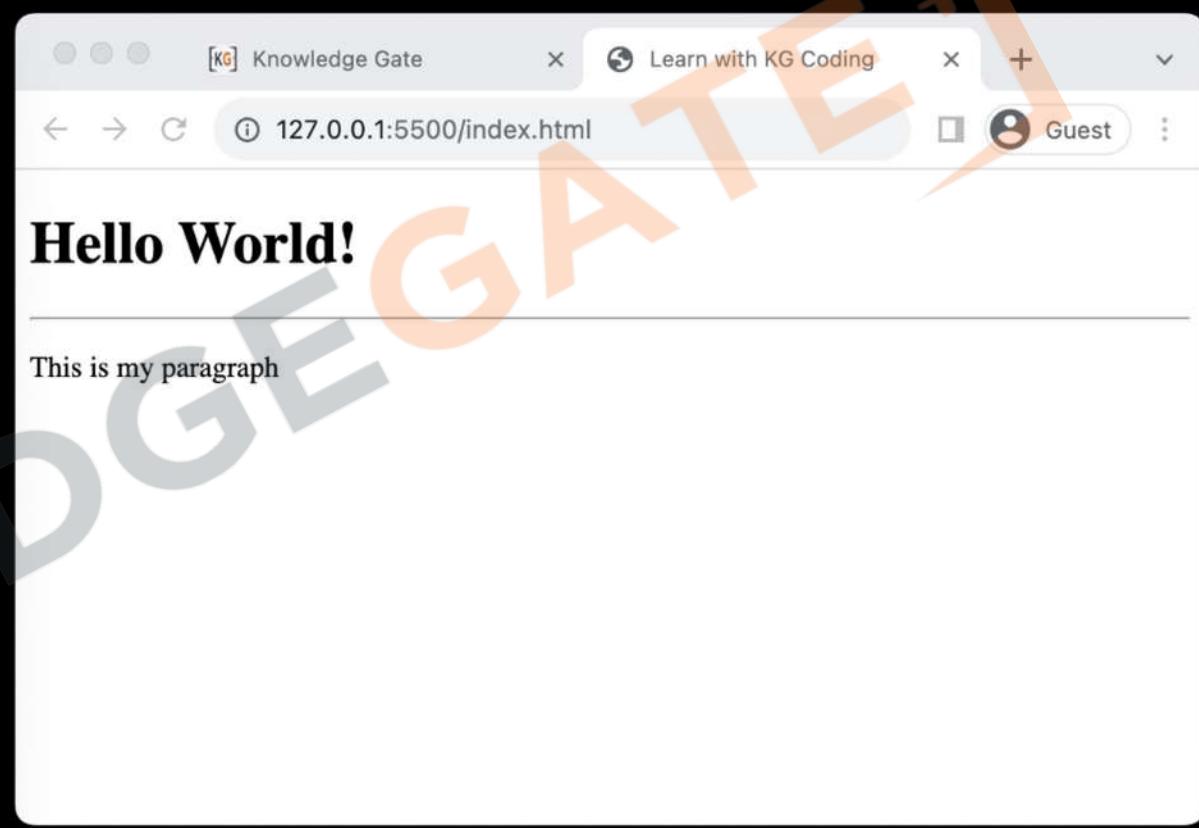
<HR> Tag



A screenshot of a code editor window titled "project". The file "index.html" is open, showing the following code:

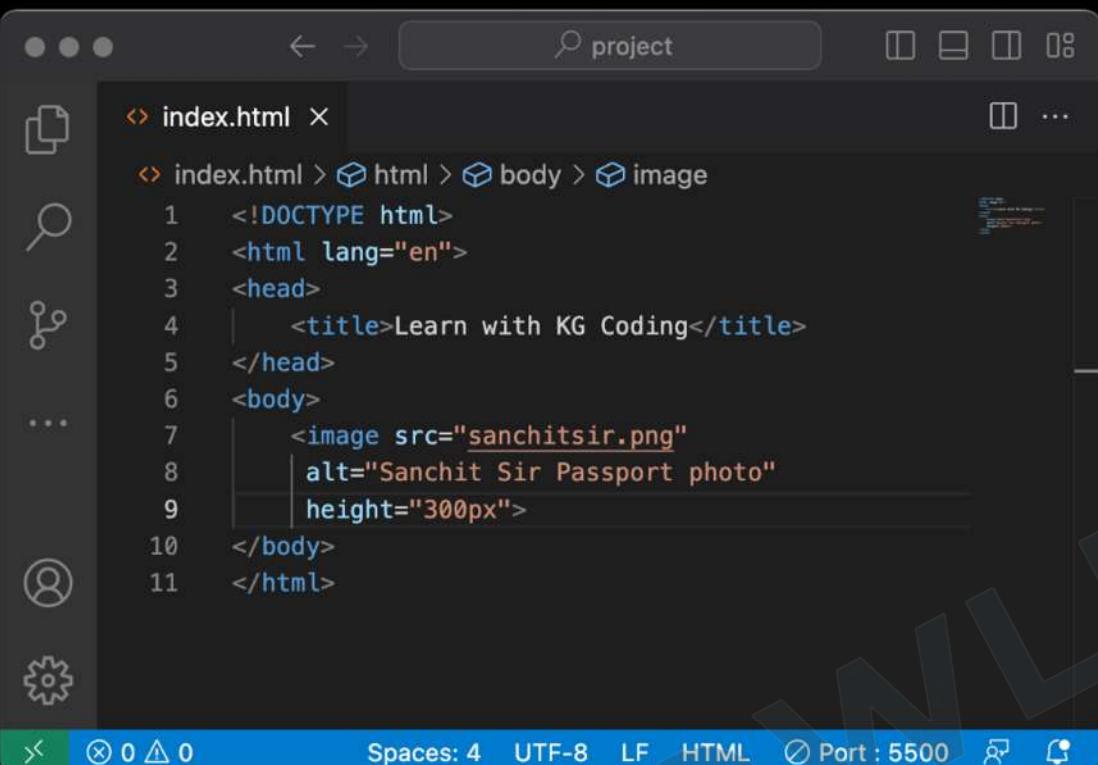
```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <h1>Hello World!</h1>
    <hr>
    <p>This is my paragraph</p>
</body>
</html>
```

The code editor has a dark theme with light-colored syntax highlighting. The status bar at the bottom shows "Spaces: 4", "UTF-8", "LF", "HTML", "Port : 5500", and a refresh icon.



1. <hr> creates a horizontal rule or line
2. <hr> also empty, acts as a divider

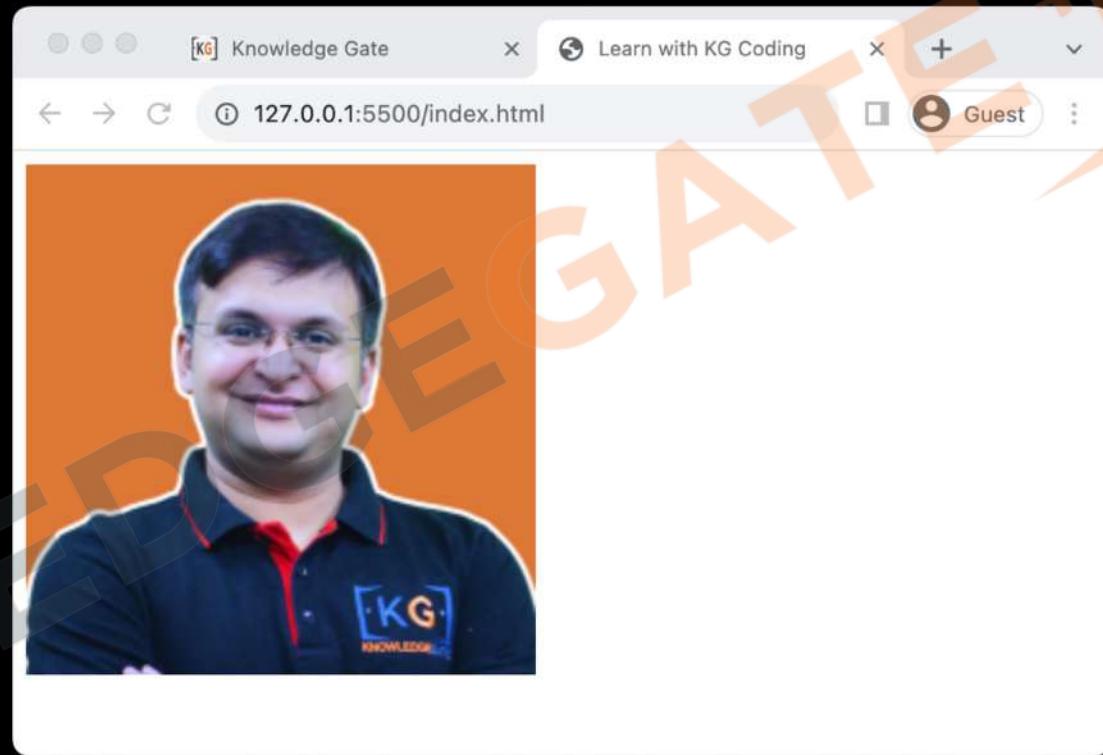
Image Tag



The screenshot shows a code editor interface with a dark theme. On the left, there's a sidebar with various icons. The main area displays an HTML file named 'index.html'. The code is as follows:

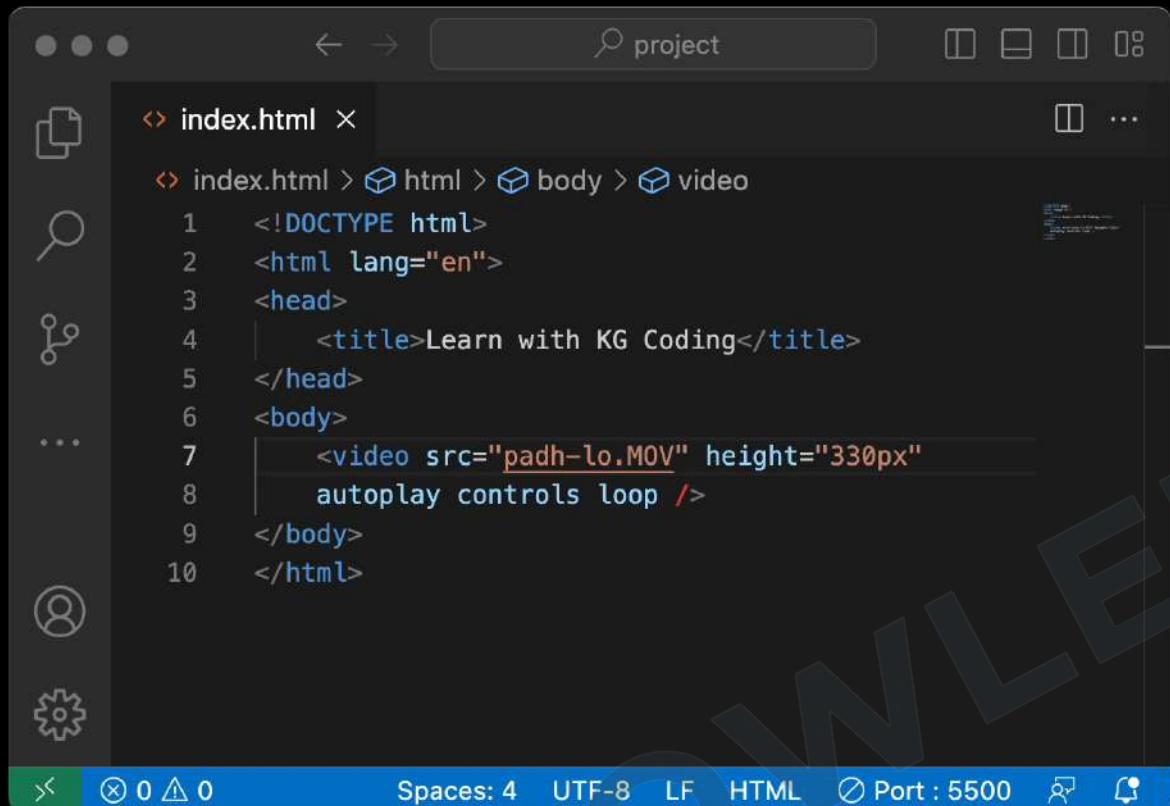
```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <image src="sanchitsir.png"
           alt="Sanchit Sir Passport photo"
           height="300px">
</body>
</html>
```

The status bar at the bottom indicates 'Spaces: 4' and 'Port : 5500'.



1. Used to embed **images**
2. Utilizes the **src** attribute for image URL
3. **alt** attribute for alternative text
4. Can be resized using **width** and **height**
5. **Self-closing**, doesn't require an end tag

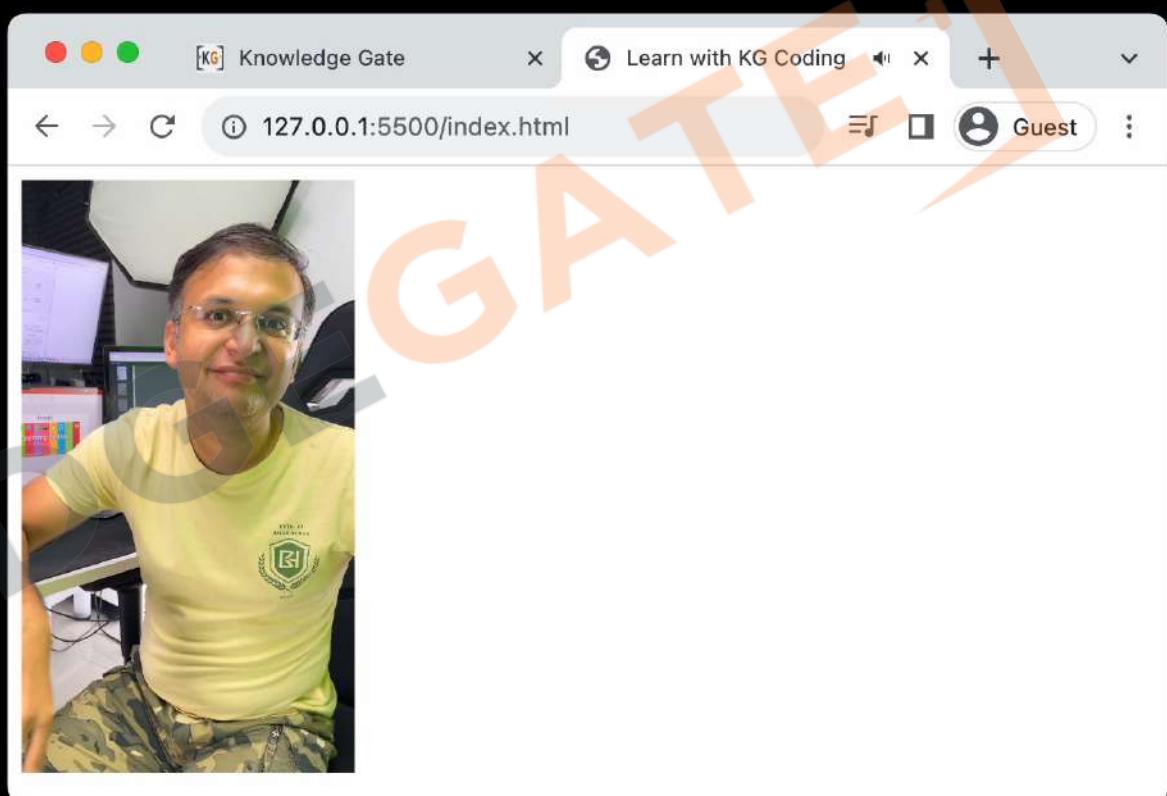
Video Tag



A screenshot of a code editor window titled "project". The file "index.html" is open, showing the following HTML code:

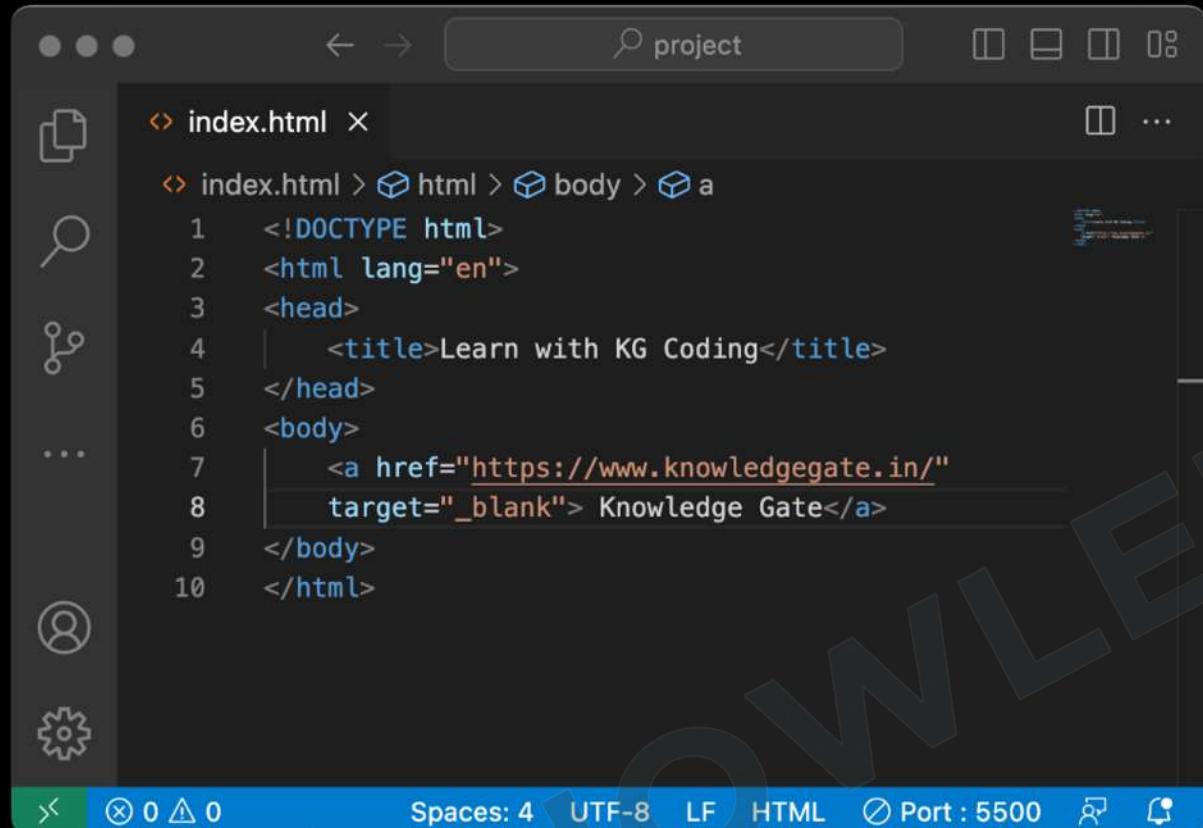
```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <video src="padh-lo.MOV" height="330px"
        autoplay controls loop />
</body>
</html>
```

The code editor has a dark theme with light-colored syntax highlighting. A sidebar on the left contains icons for file operations like new, open, save, and search.



1. Embeds video files on a page
2. Uses **src** attribute for video URL
3. Supports **multiple formats** like MP4, WebM
4. Allows for built-in controls via attributes like **autoplay, controls, loop**

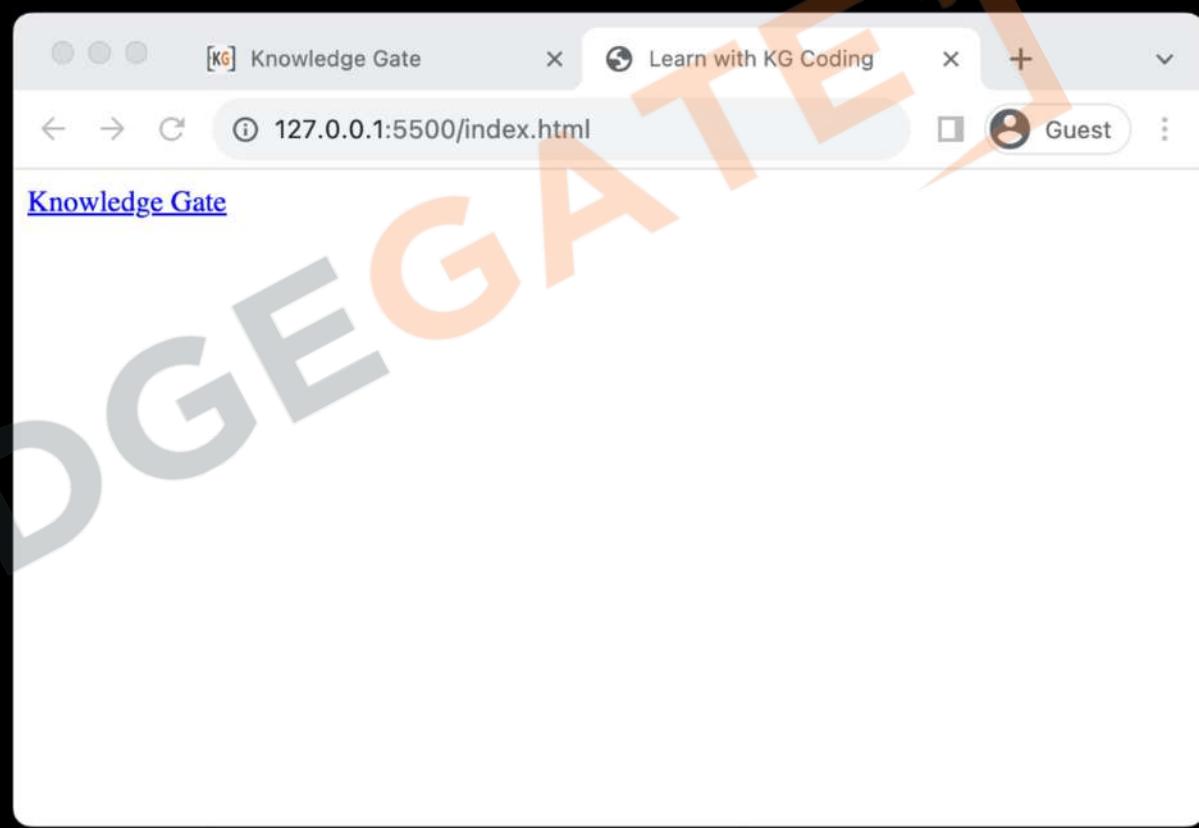
Anchor Tag



A screenshot of a code editor window titled "project". The file "index.html" is open, showing the following HTML code:

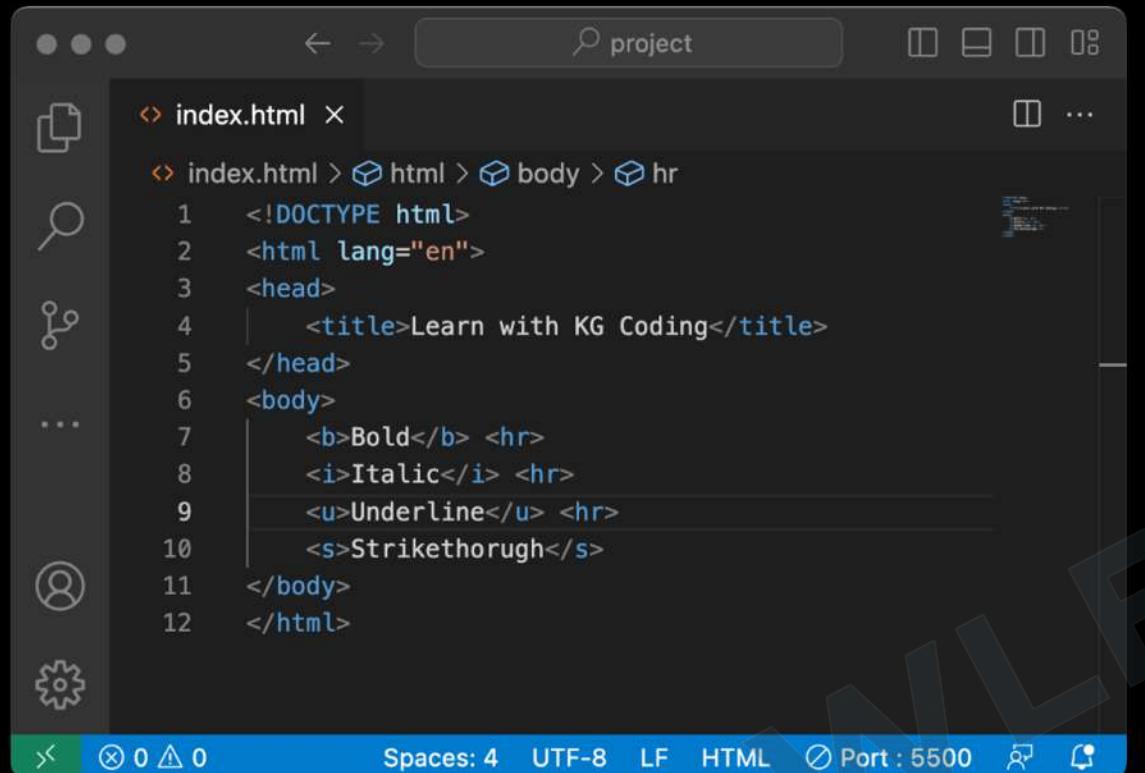
```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <a href="https://www.knowledgegate.in/" target="_blank"> Knowledge Gate</a>
</body>
</html>
```

The code editor has a dark theme with light-colored syntax highlighting. A sidebar on the left contains icons for file operations like new, open, save, and search.



1. Used for creating hyperlinks
2. Requires **href** attribute for URL
3. Can link to external sites or internal pages
4. Supports **target** attribute to control link behavior

Bold/Italic/Underline/Strikethrough Tag



A screenshot of a code editor window titled "project". The file "index.html" is open, displaying the following HTML code:

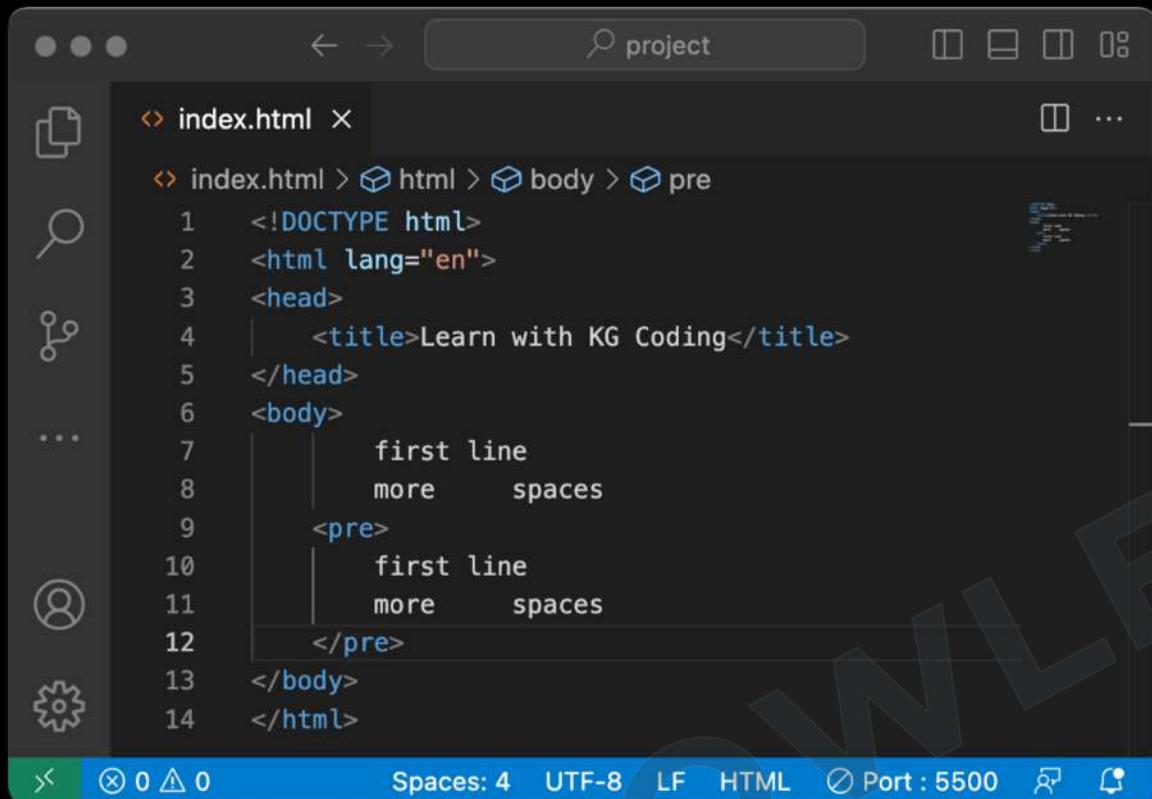
```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <b>Bold</b> <hr>
    <i>Italic</i> <hr>
    <u>Underline</u> <hr>
    <s>Strikethrough</s>
</body>
</html>
```

The code editor has a dark theme with light-colored syntax highlighting. The status bar at the bottom shows "Spaces: 4", "UTF-8", "LF", "HTML", "Port: 5500", and a refresh icon.



1. **** makes text bold
2. *<i>* makes text italic
3. <u> underlines text
4. ~~<s>~~ or ~~<strike>~~ applies strikethrough
5. Primarily used for text styling and emphasis

Pre Tag



A screenshot of a code editor window titled "project". The file "index.html" is open, showing the following HTML code:

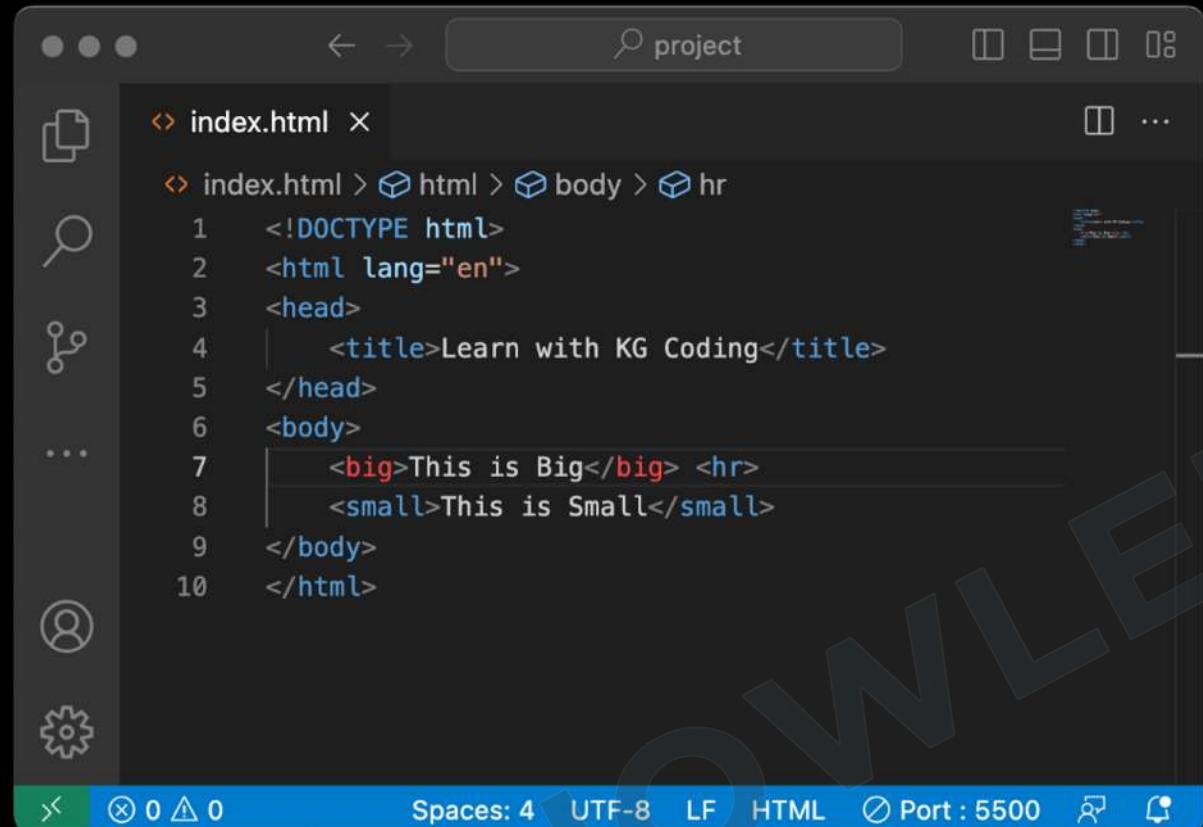
```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    first line
    more spaces
    <pre>
        first line
        more spaces
    </pre>
</body>
</html>
```

The code editor interface includes a sidebar with icons for file, search, and user, and a bottom bar with status indicators and settings.

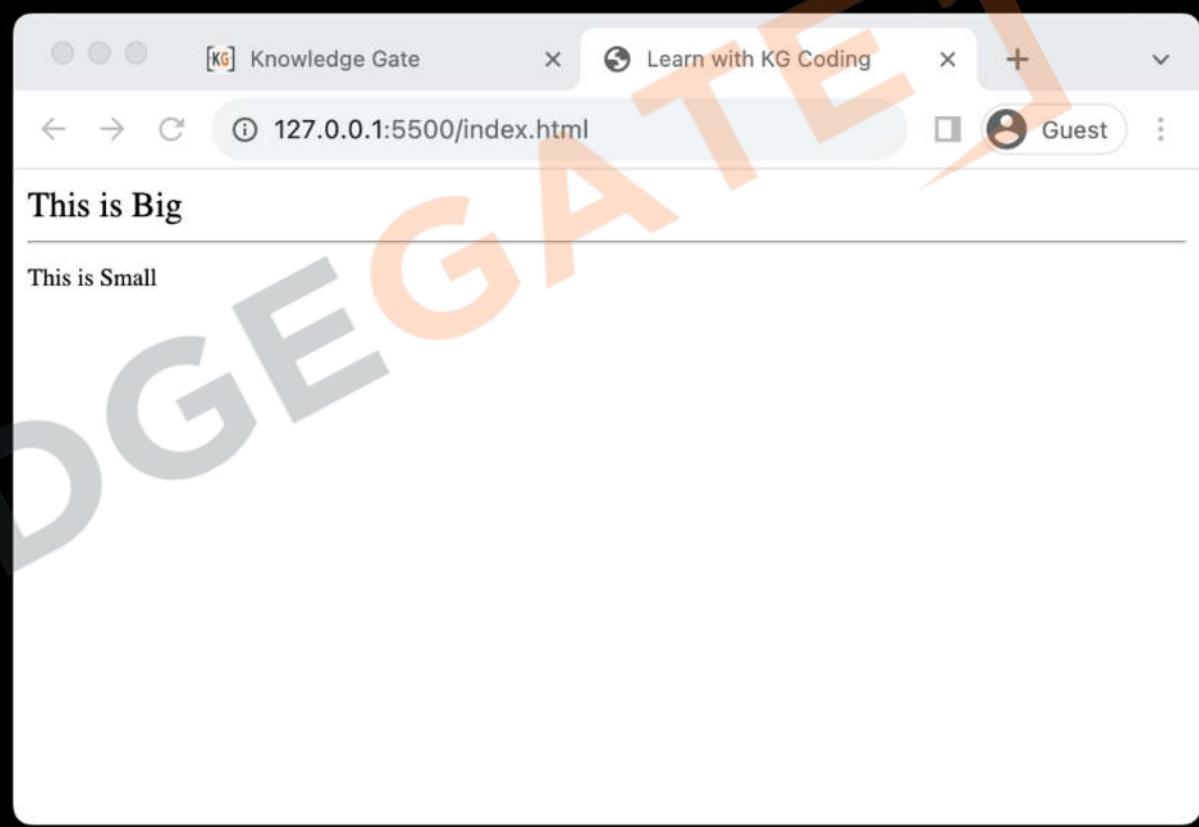


1. Preserves text formatting
2. Maintains whitespace and line breaks
3. Useful for displaying code
4. Enclosed within <pre> and </pre> tags

Big/Small Tag

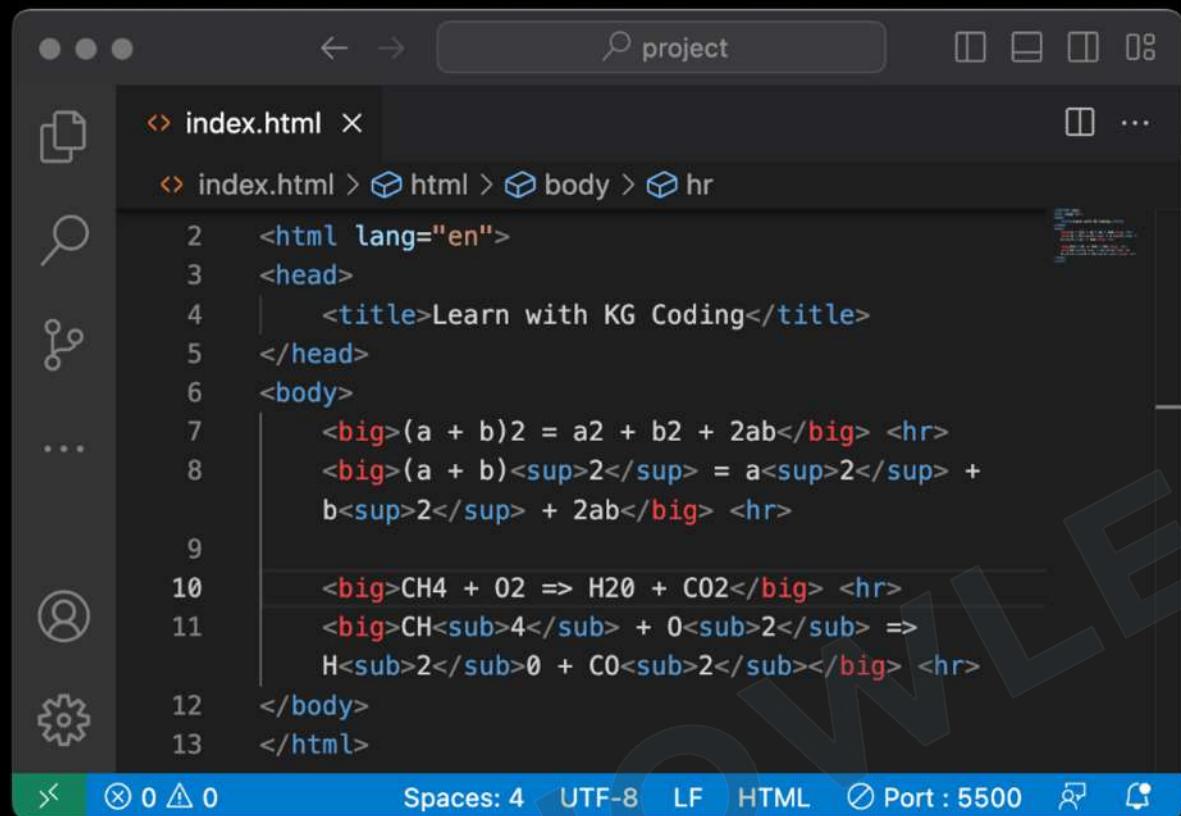


```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <big>This is Big</big> <hr>
    <small>This is Small</small>
</body>
</html>
```



1. `<big>` increases text size
2. `<small>` decreases text size
3. Less common due to CSS alternatives

Superscript/Subscript Tag

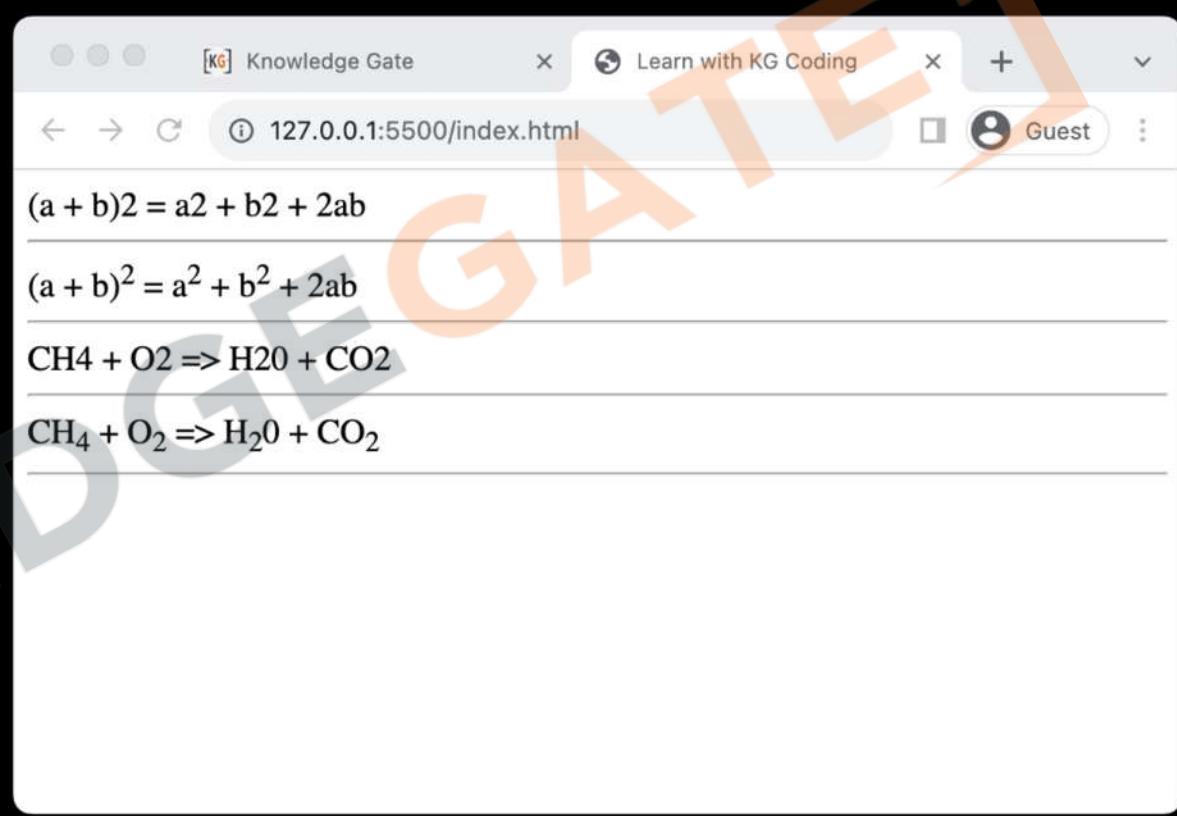


A screenshot of a code editor window titled "project". The file "index.html" is open, showing the following code:

```
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <big>(a + b)2 = a2 + b2 + 2ab</big> <hr>
    <big>(a + b)2 = a2 + b2 + 2ab</big> +
    b2 + 2ab</big> <hr>

    <big>CH4 + O2 => H2O + CO2</big> <hr>
    <big>CH4 + O2 => H2O + CO2</big> <hr>
</body>
</html>
```

The code uses the `<big>` tag to wrap mathematical equations. The browser's developer tools sidebar is visible on the left.



A screenshot of a web browser window titled "Knowledge Gate". The URL is "127.0.0.1:5500/index.html". The page displays the following mathematical equations:

$$(a + b)^2 = a^2 + b^2 + 2ab$$
$$(a + b)^2 = a^2 + b^2 + 2ab$$
$$\text{CH}_4 + \text{O}_2 \Rightarrow \text{H}_2\text{O} + \text{CO}_2$$
$$\text{CH}_4 + \text{O}_2 \Rightarrow \text{H}_2\text{O} + \text{CO}_2$$

1. `<sup>` makes text superscript
2. `<sub>` makes text subscript
3. Used for mathematical equations, footnotes
4. Does not change font size, just position

Must-Use HTML Tags

Character Entity Reference

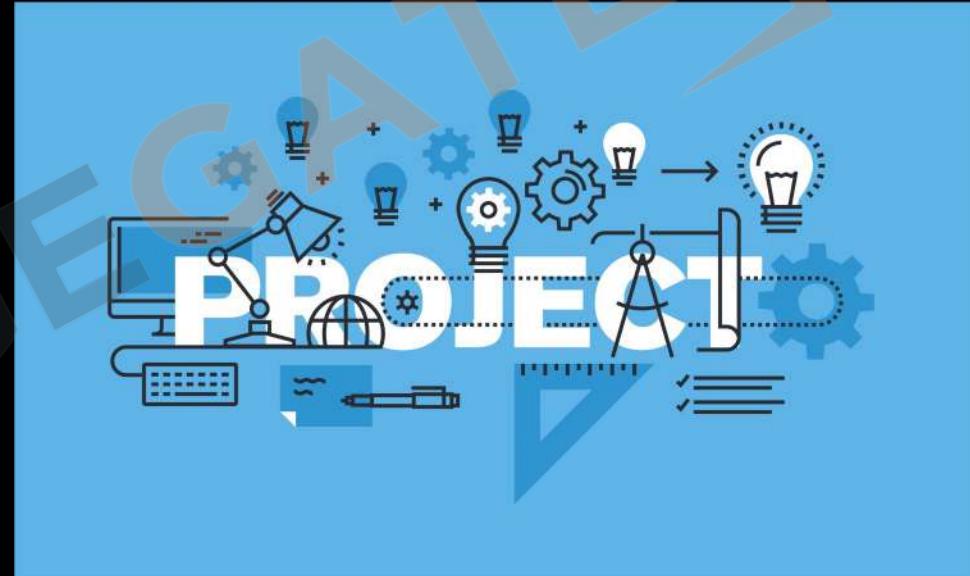
Character Entity Reference

1. Used to display reserved or special characters
2. Syntax often starts with & and ends with ; (e.g., & for &)

 	-	–	-	−	°	°	Δ	Δ	α	&al;	
€	€	—	—	±	±	°	º	Λ	Λ	β	&be;
ƒ	¢	…	…	√	√	ª	ª	Θ	Θ	γ	&ga;
£	£	§	§	∞	∞	¹	¹	Ξ	Ξ	δ	&de;
¥	¥	¶	¶	∞	∝	²	²	Π	Π	ε	&ep;
¤	¤	†	†	×	×	³	³	Σ	Σ	ζ	ζ
f	ƒ	‡	‡	÷	÷	¼	¼	Φ	Φ	η	&et;
©	©		¡	~	∼	½	½	Ψ	Ψ	θ	&th;
®	®	⌚	¿	≈	≈	¾	¾	Ω	Ω	ι	&io;
™	™	%	‰	≈	≅	∴	∴	∇	∇	κ	κ

Practise Set

1. Create a page with **heading**, **paragraph**, **line breaks** and **separators**.
2. Use an **image** with height 300, which is a **link** to another page.
3. Use **bold**, **italic**, **underline** and **strike through** in one line.
4. Write third equation of motion using **superscript** and **subscript**.







Importance of CSS



Premium Website



Premium Brand

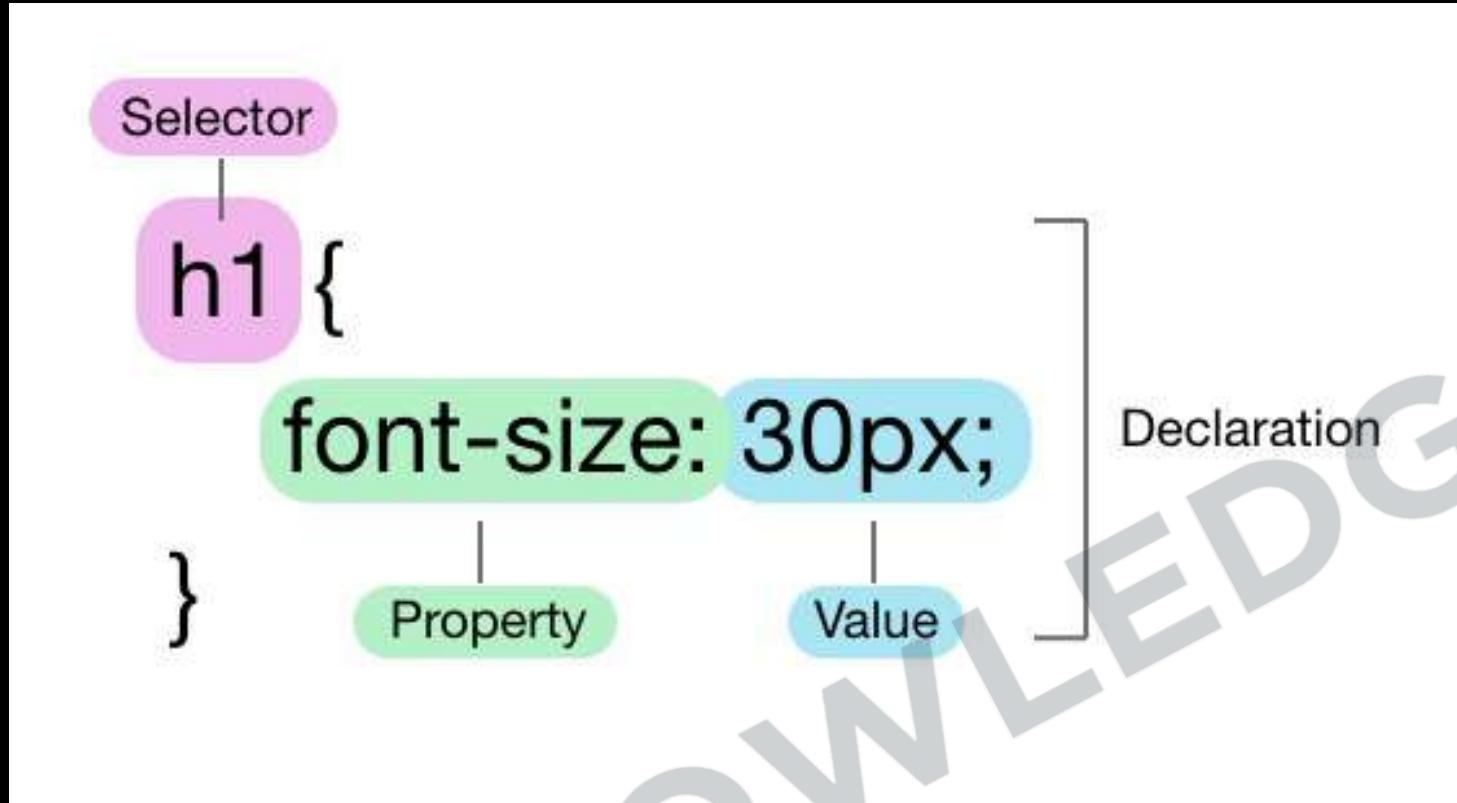


Premium Customer



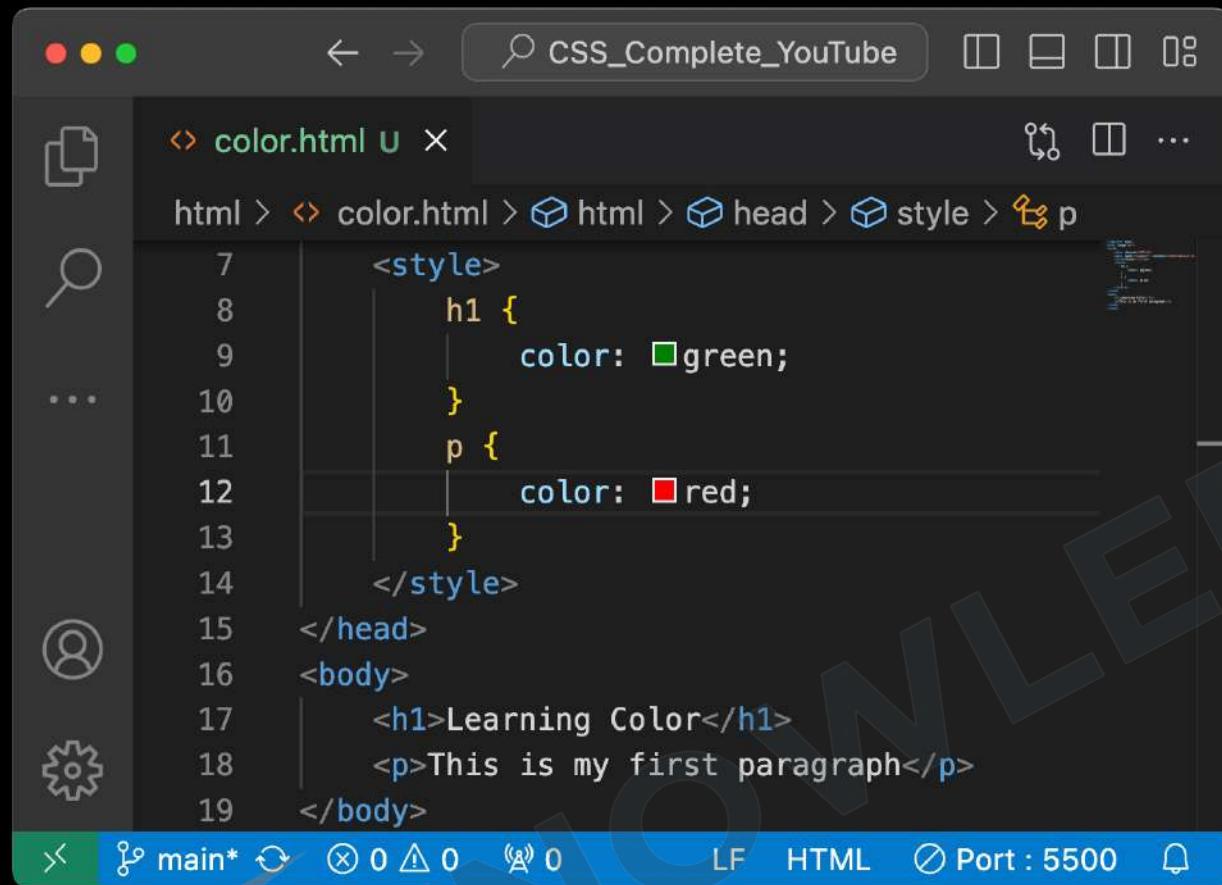
High Salary Developer

CSS Basic Syntax



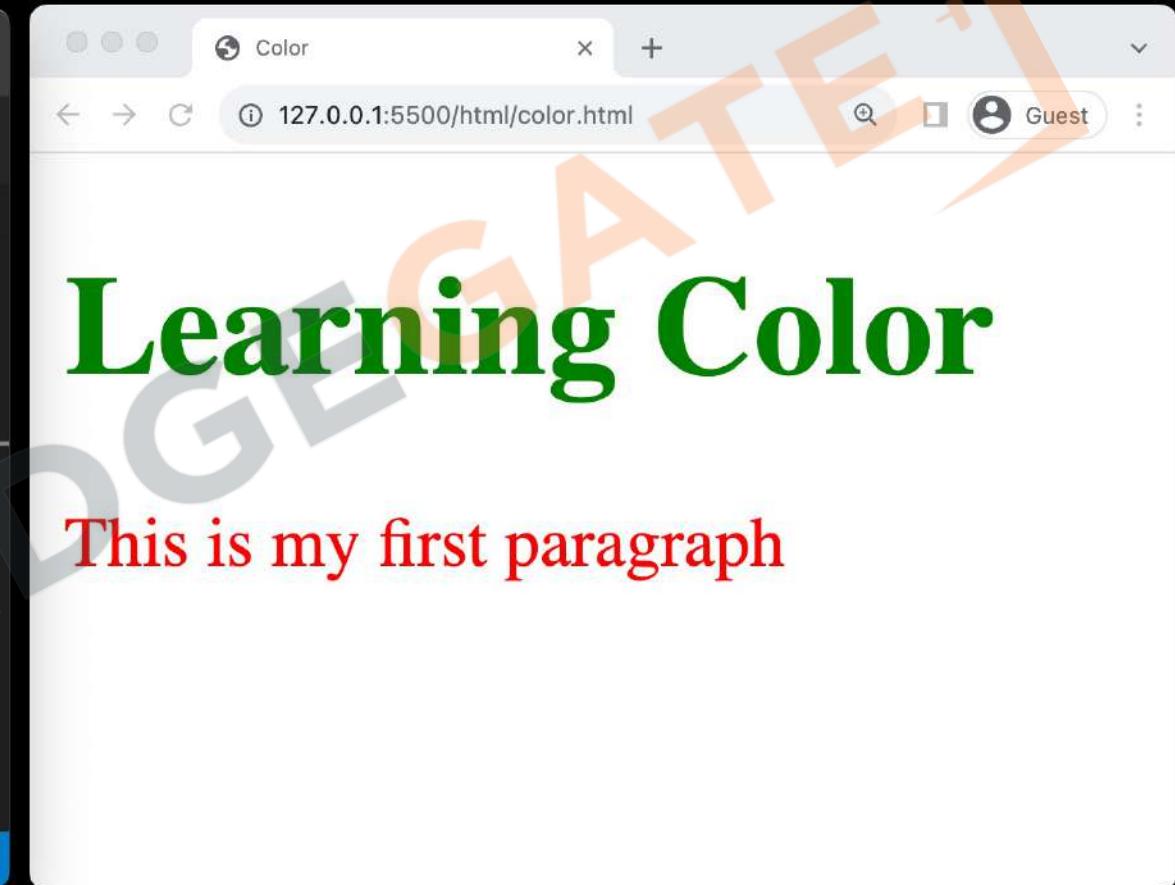
- **Selector**: The HTML element that you want to style.
- **Property**: The attribute you want to change (like font, color, etc.).
- **Value**: The specific style you want to apply to the property (like red, bold, etc.).

Color Property



A screenshot of a code editor window titled "css_Complete_YouTube". The file being edited is "color.html". The code shows a CSS style block with rules for an h1 element (color: green) and a p element (color: red). The HTML part contains an h1 with the text "Learning Color" and a p with the text "This is my first paragraph". The code editor has a dark theme with syntax highlighting.

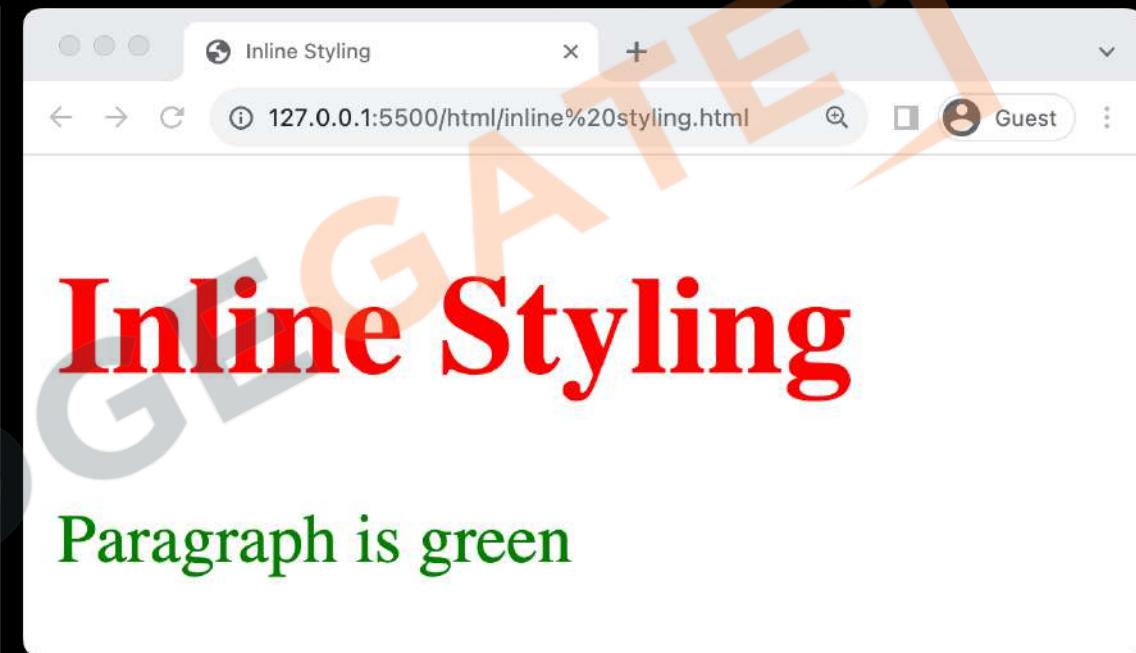
```
html > color.html U X
  html > color.html > head > style > p
    7   <style>
    8     h1 {
    9       color: green;
   10    }
   11    p {
   12      color: red;
   13    }
   14  </style>
   15 </head>
   16 <body>
   17   <h1>Learning Color</h1>
   18   <p>This is my first paragraph</p>
   19 </body>
```



- **Definition:** The *CSS* color property defines the text color or foreground color in an *HTML* element.
- **Enhancement:** Use it to emphasize sections and elevate webpage aesthetics.

Including Styles (Inline Styling)

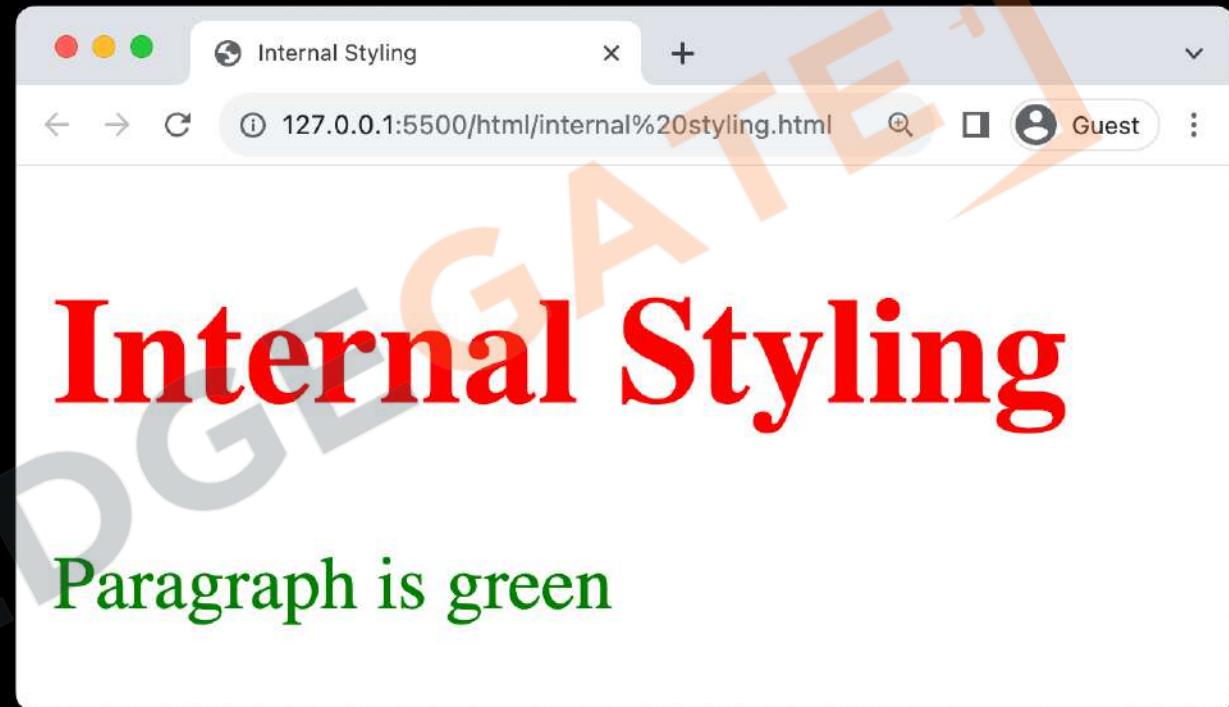
```
html > inline styling.html > html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <title>Inline Styling</title>
5  </head>
6  <body>
7      <h1 style="color: red;">Inline Styling</h1>
8      <p style="color: green;">Paragraph is green</p>
9  </body>
10 </html>
```



- **Direct Application:** Apply styles directly to **HTML** elements using the **style** attribute.
- **One-off Changes:** Ideal for **single**, **unique style** alterations.
- **Can Be Cluttered:** May lead to **cluttered** **HTML** if used extensively.
- **Limited Reusability:** Reduces the **reusability** of **CSS** rules in larger projects.

Including Styles (Internal Styling)

```
3 <head>
4     <title>Internal Styling</title>
5     <style>
6         h1 {color: red;}
7         p {color: green;}
8     </style>
9 </head>
10 <body>
11     <h1>Internal Styling</h1>
12     <p>Paragraph is green</p>
13 </body>
14 </html>
```

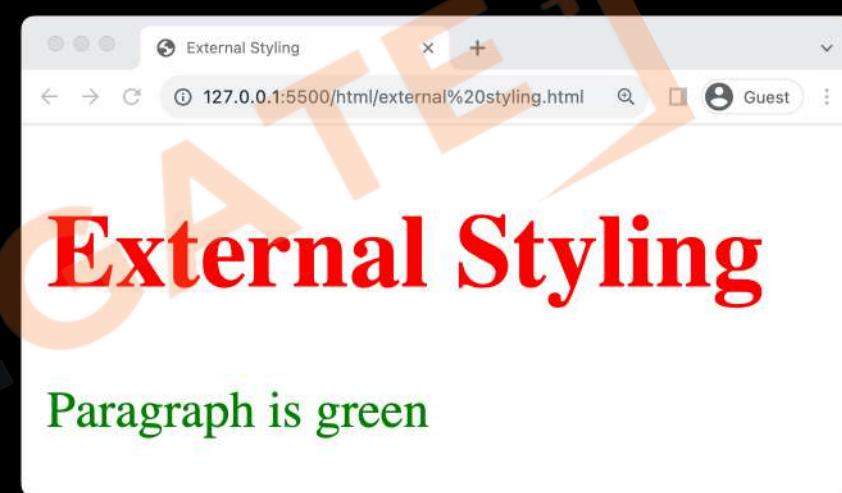


- **Embedded CSS:** Styles are placed within **<style>** tags in the HTML head section.
- **Cleaner than Inline:** More organized compared to inline styles.
- **Reusable Styles:** Allows for some reuse of styles across the page.

Including Styles (External Styling)

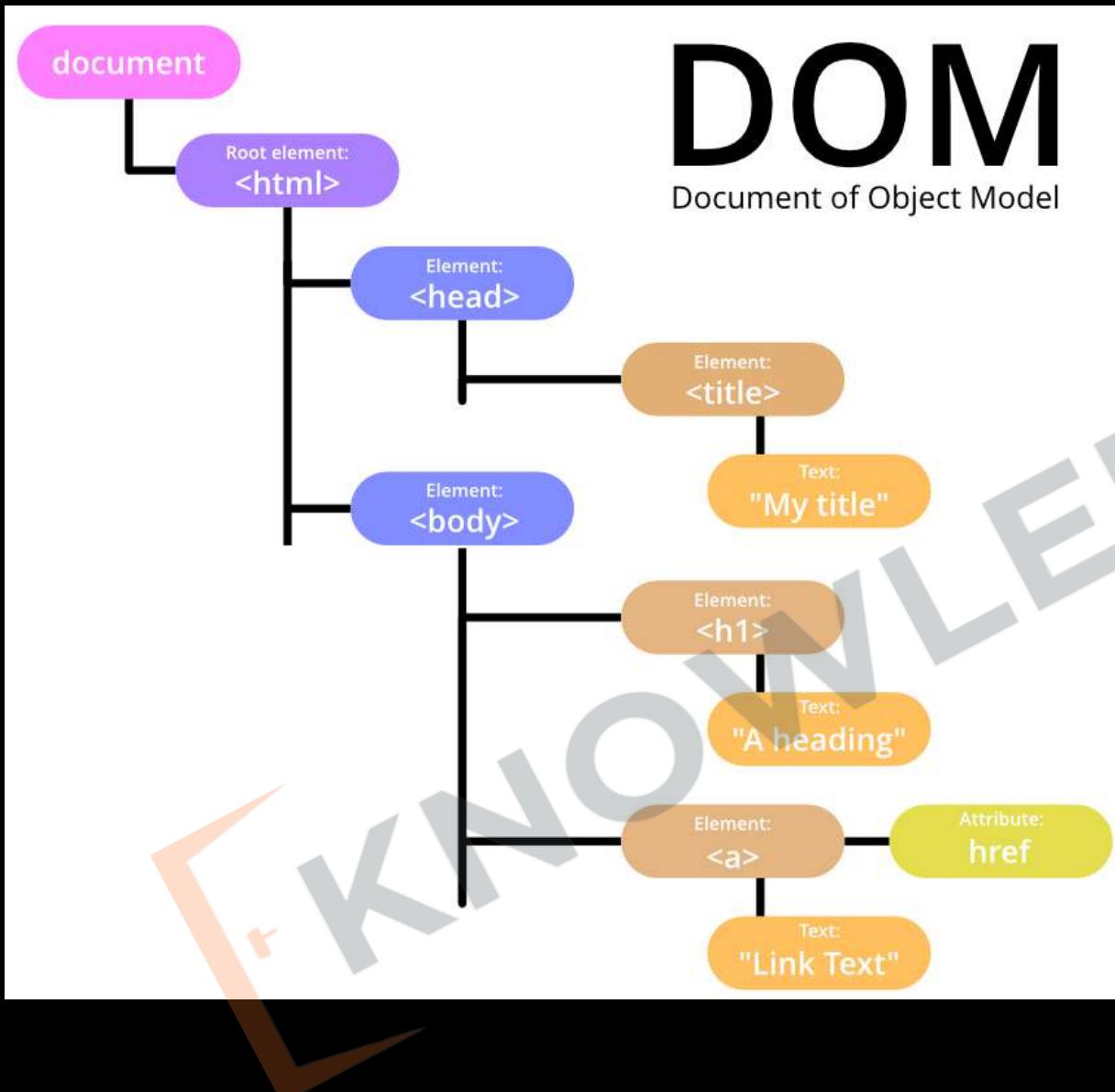
```
3 <head>
4     <title>External Styling</title>
5     <link rel="stylesheet" href="../css/
6         external styling.css">
7 </head>
8 <body>
9     <h1>External Styling</h1>
10    <p>Paragraph is green</p>
11 </body>
```

```
css > # external styling.css > p
1   <h1>
2       color: red;
3   </h1>
4
5   <p>
6       color: green;
7   </p>
```



- **Separate CSS File:** Stores styles in a **separate .css file**, linked to HTML.
- **Reusable:** Enables style **reuse** across multiple webpages.
- **Link in HTML:** Use the **<link>** tag within the **<head>** section to link the CSS.
- **Relative or Absolute Path:** The href attribute can contain a relative or absolute path to the CSS file.

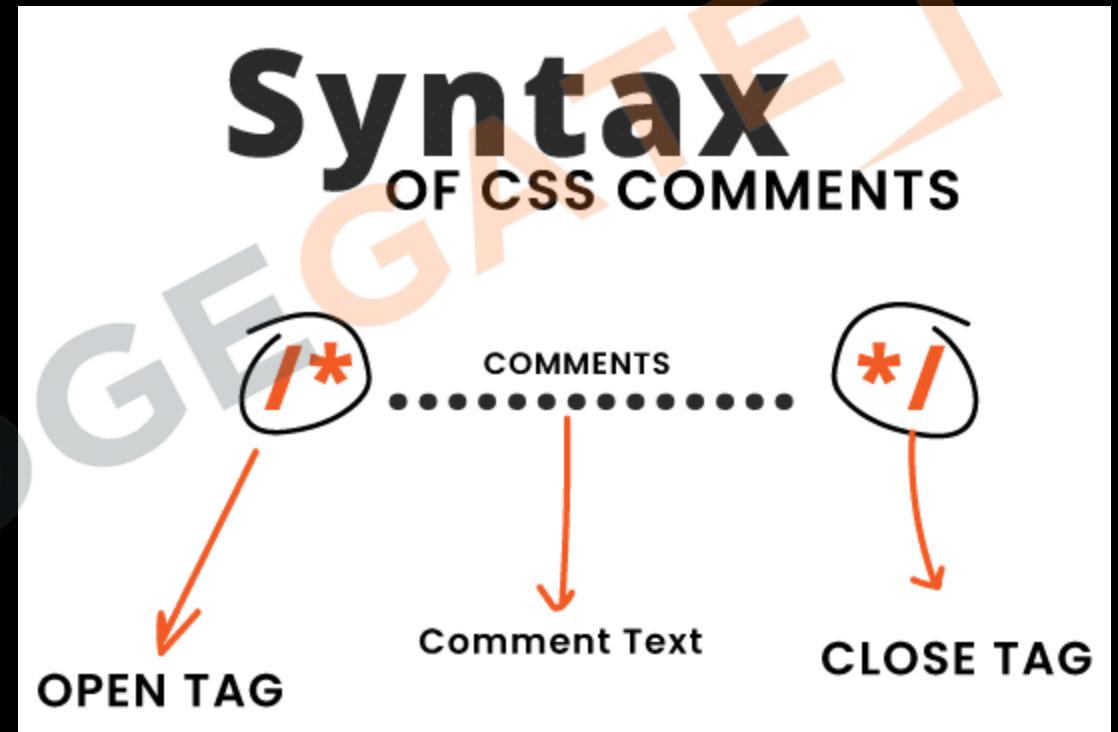
HTML Refresher (DOM)



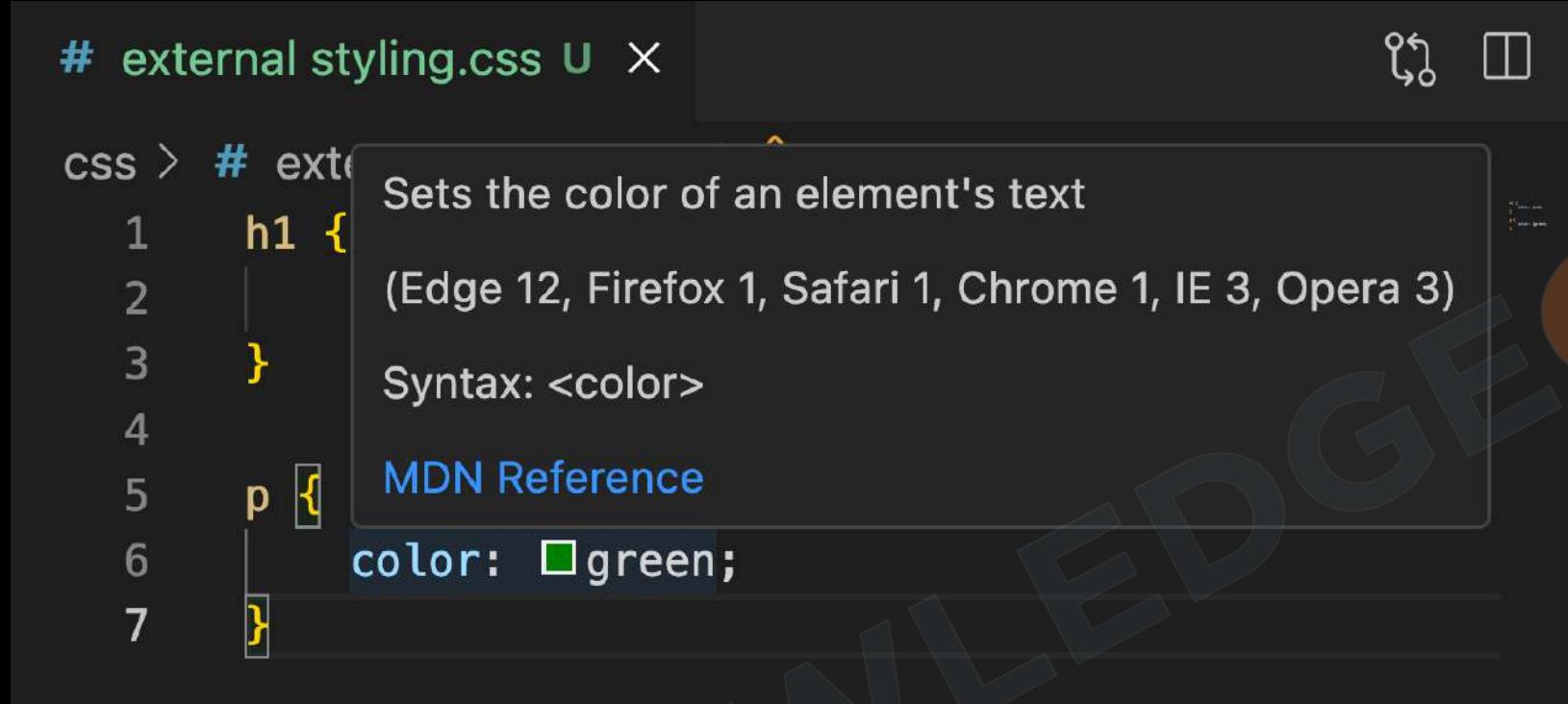
- 1. Structure Understanding:** Helps in understanding the **hierarchical structure** of a webpage, crucial for applying targeted **CSS styles**.
- 2. Dynamic Styling:** Enables learning about **dynamic styling**, allowing for **real-time changes** and **interactivity** through CSS.

CSS Comments

1. Used to add **notes** in HTML or CSS code
2. **Not displayed** on the web page
3. Syntax: `/* */`
4. Helpful for **code organization**
5. Can be **multi-line** or **single-line**



MDN Documentation



A screenshot of a code editor window titled '# external styling.css'. The code is as follows:

```
css > # external styling.css
1  h1 {
2    |
3  }
4
5  p {
6    color: green;
7 }
```

A tooltip is displayed over the 'color' declaration in the fifth line. The tooltip contains the following information:

- Sets the color of an element's text
- (Edge 12, Firefox 1, Safari 1, Chrome 1, IE 3, Opera 3)
- Syntax: <color>
- [MDN Reference](#)

1. For Official resource for CSS, visit
<https://developer.mozilla.org/>
2. Includes examples for real-world use
3. Updated with latest CSS-3 features
4. Trusted by developers worldwide

Selectors (Element selector)

```
3 <head>
4     <title>Element Selector</title>
5     <style>
6         h1 {
7             color: red
8         }
9     </style>
10    </head>
11    <body>
12        <h1>Universal Selector</h1>
13        <p>Lorem ipsum dolor sit amet consectetur
14            adipisicing elit. Nulla, nisi.</p>
15    </body>
```



- **Targets Elements:** Selects HTML elements based on their **tag name**.
- **Syntax:** Simply use the **element's name**
- **Uniform Styling:** Helps in applying **consistent styles** to all instances.
- **Ease of Use:** Straightforward and **easy** to implement for basic styling.

Selectors (Universal selector)

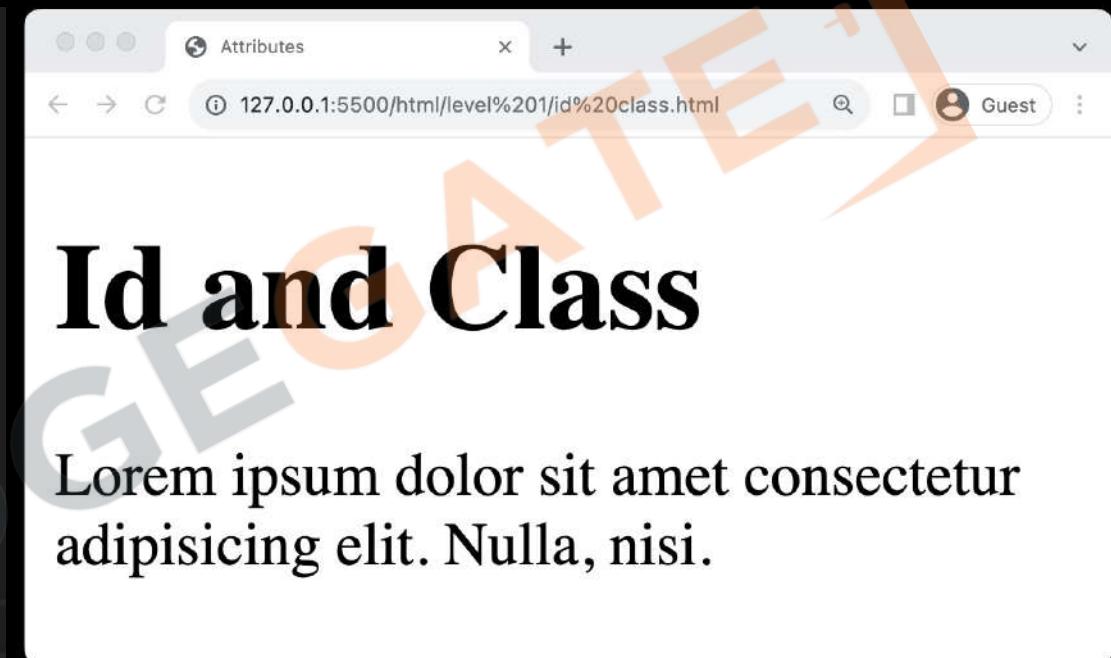
```
3 <head>
4     <title>Universal Selector</title>
5     <style>
6         * {
7             color: red
8         }
9     </style>
10    </head>
11    <body>
12        <h1>Universal Selector</h1>
13        <p>Lorem ipsum dolor sit amet consectetur
14            adipisicing elit. Nulla, nisi.</p>
```



- **Matches All:** Targets and styles **all elements** on a webpage.
- **Syntax:** Utilized as an **asterisk (*)**.
- **Resets Styles:** Commonly used to **reset margins** and paddings globally.
- **Broad Styling:** Useful for setting universal attributes like **font or color**.
- **Usage Caution:** Can cause style **conflicts** due to its wide-reaching effects.

Selectors (id & class property)

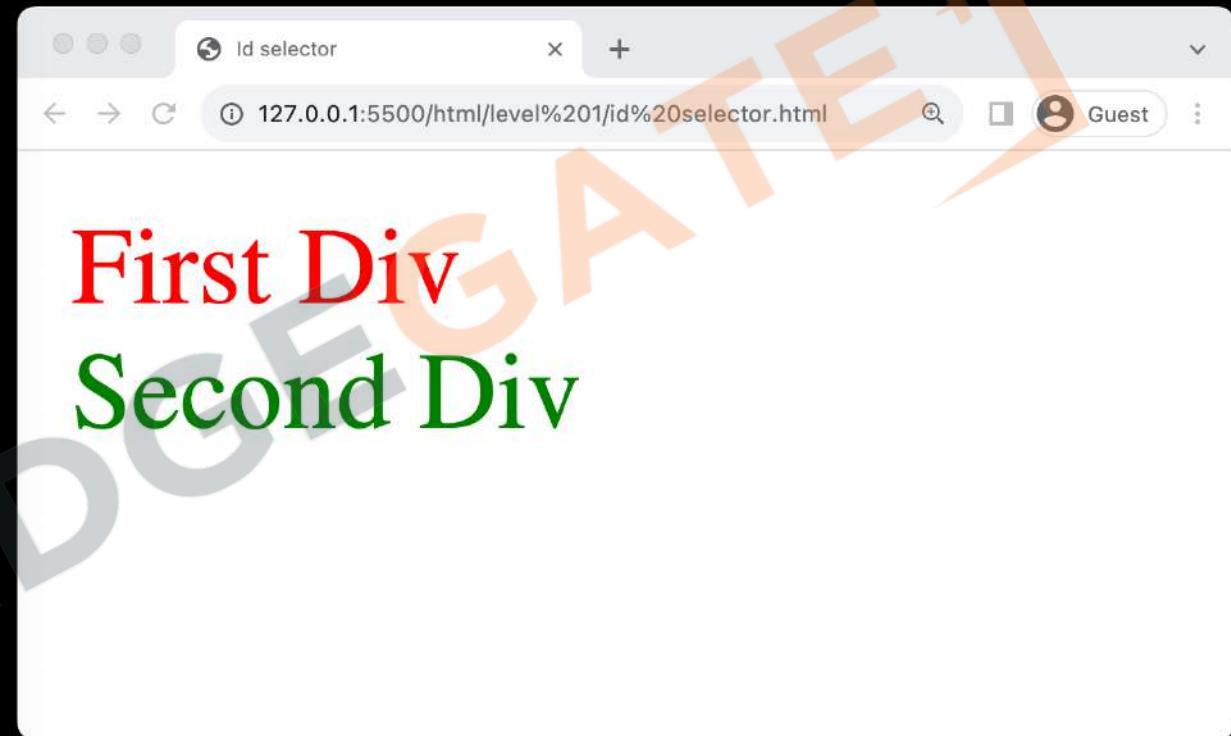
```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <title>Attributes</title>
5  </head>
6  <body>
7      <h1 id="top_heading">Id and Class</h1>
8      <p class="article">Lorem ipsum dolor sit amet
9         consectetur adipisicing elit. Nulla, nisi.</p>
10 </body>
11 </html>
```



- **ID Property:** Assigns a unique identifier to a single HTML element.
- **Class Property:** Allows grouping of multiple HTML elements to style them collectively.
- **Reusable Classes:** Class properties can be reused across different elements for consistent styling.
- **Specificity and Targeting:** Both properties assist in targeting specific elements or groups of elements for precise styling.

Selectors (Id selector)

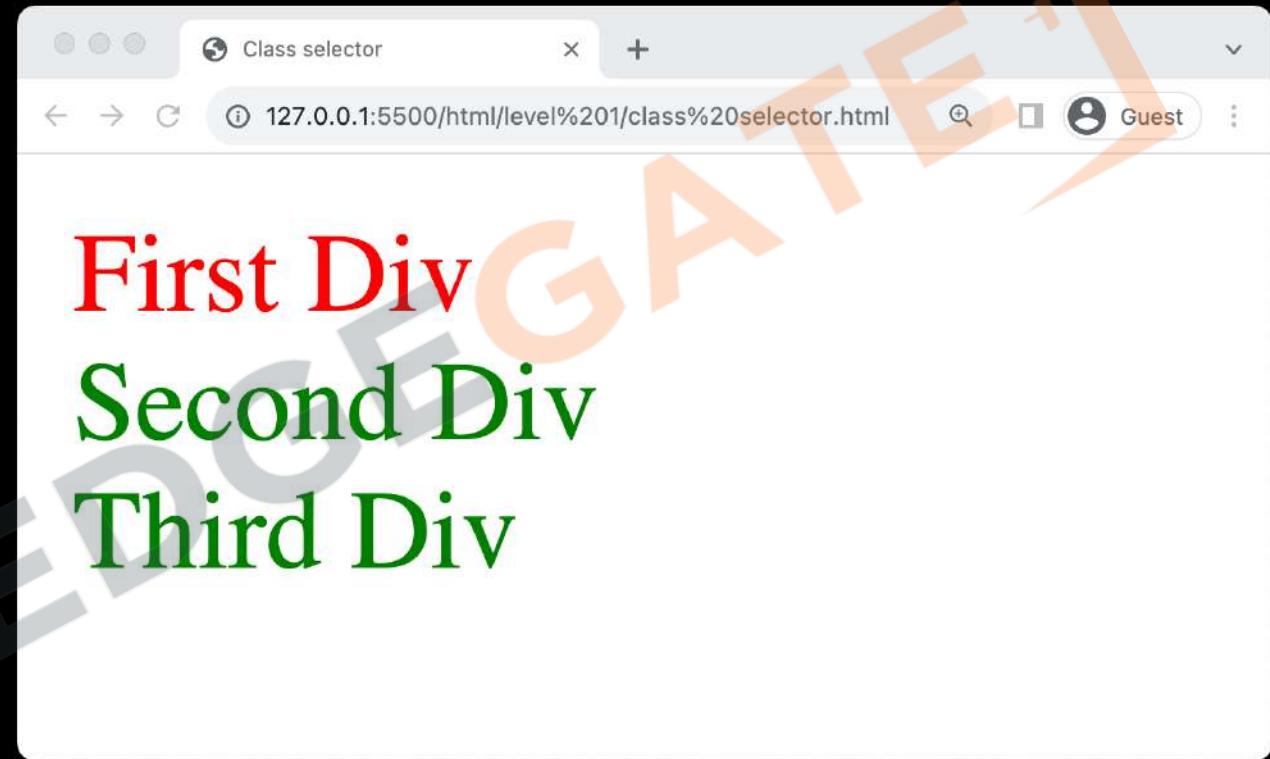
```
3 <head>
4     <title>Id selector</title>
5     <style>
6         #first { color: red; }
7         #second { color: green; }
8     </style>
9 </head>
10 <body>
11     <div id="first">First Div</div>
12     <div id="second">Second Div</div>
13 </body>
```



- **Unique Identifier:** Targets a specific element with a **unique ID** attribute.
- **Syntax:** Uses the **hash (#)** symbol
- **Single Use:** Each ID should be used **once per page** for uniqueness.
- **Specific Targeting:** Ideal for styling **individual, distinct** elements.

Selectors (Class selector)

```
<head>
    <title>Class selector</title>
    <style>
        #first { color: red; }
        .second { color: green; }
    </style>
</head>
<body>
    <div id="first">First Div</div>
    <div class="second">Second Div</div>
    <div class="second">Third Div</div>
</body>
```



Selectors (Group selector)

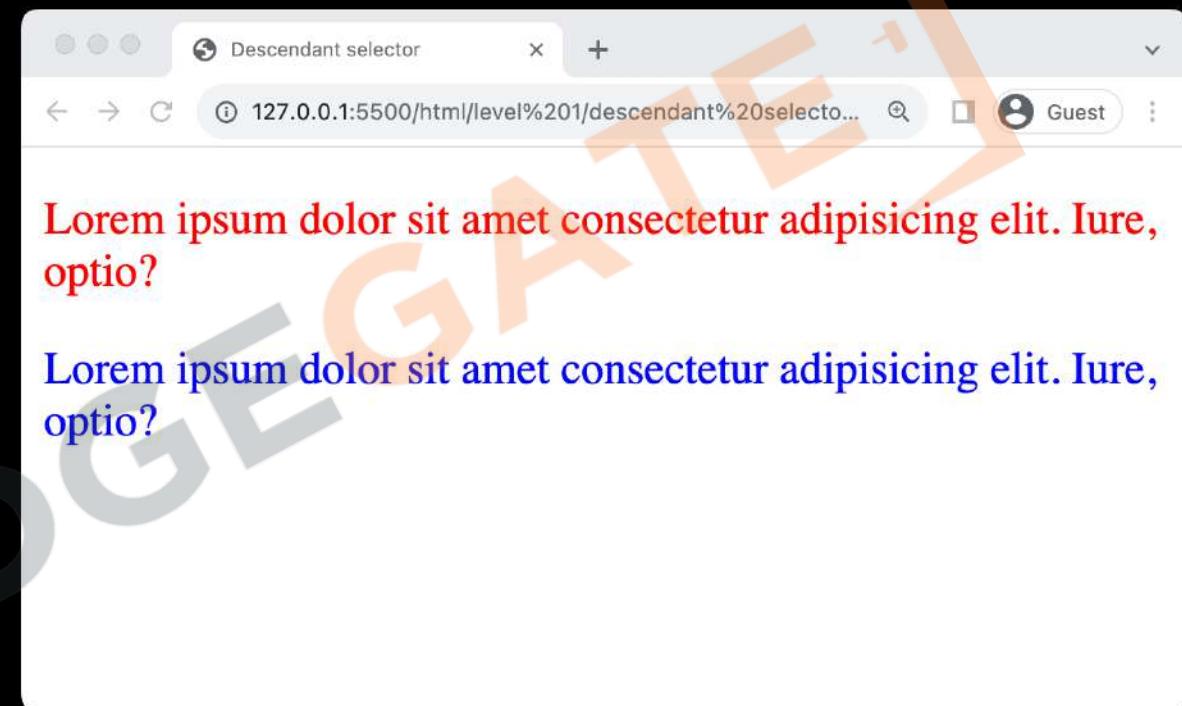
```
<head>
  <title>Group selector</title>
  <style>
    h1, h2, h3 {
      color: red
    }
  </style>
</head>
<body>
  <h1>Heading 1</h1>
  <h2>Heading 2</h2>
  <h3>Heading 3</h3>
  <h4>Heading 4</h4>
</body>
```



- **Multiple Elements:** Styles **multiple** elements simultaneously.
- **Syntax:** Separates selectors with **commas**.
- **Efficiency:** Reduces code **redundancy** and saves time.

Selectors (Descendant selector)

```
<head>
  <title>Descendant selector</title>
  <style>
    div p { color: red }
    p { color: blue }
  </style>
</head>
<body>
  <div>
    <p>
      Lorem ipsum dolor sit amet consectetur adipisicing elit. Iure, optio?
    </p>
  </div>
  <p>
    Lorem ipsum dolor sit amet consectetur adipisicing elit. Iure, optio?
  </p>
</body>
```

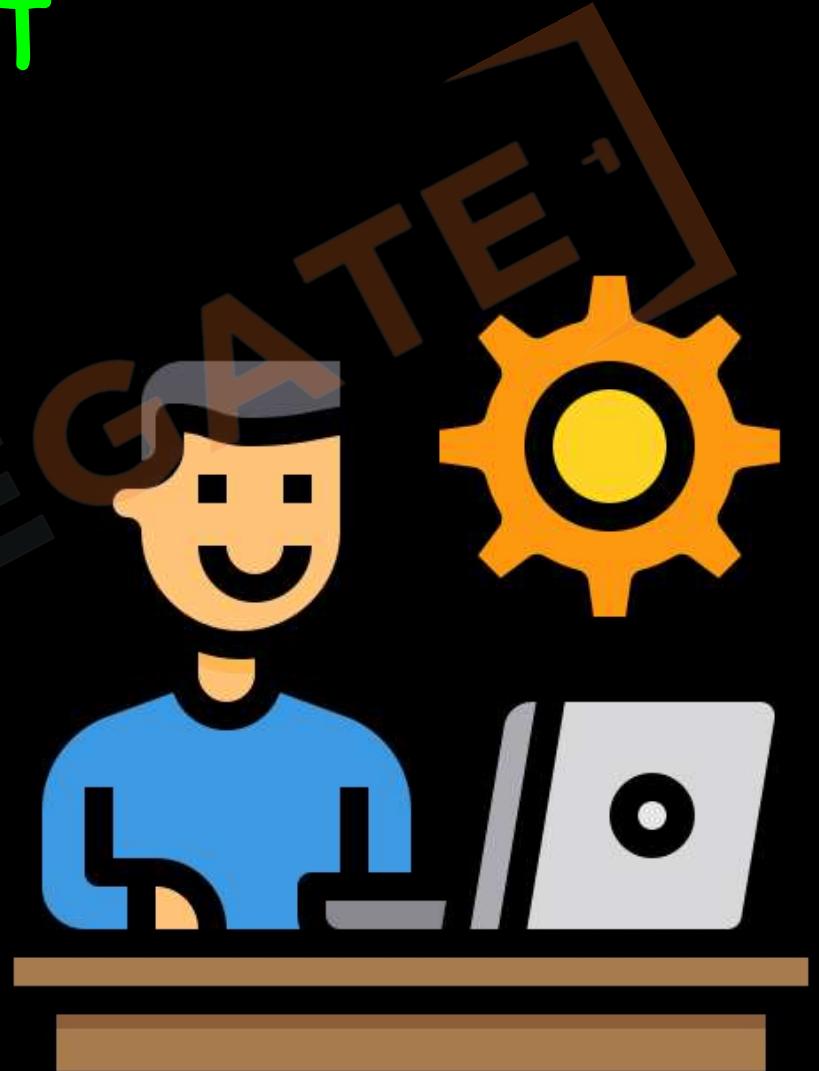


- **Nested Targeting:** Styles elements **nested** within a specified element.
- **Syntax:** Separate selectors with **spaces**.
- **Hierarchy-Based:** Allows styling based on the **hierarchical** structure of HTML.
- **Specific Styling:** Facilitates more **targeted** and specific styling of elements.

Practice Set

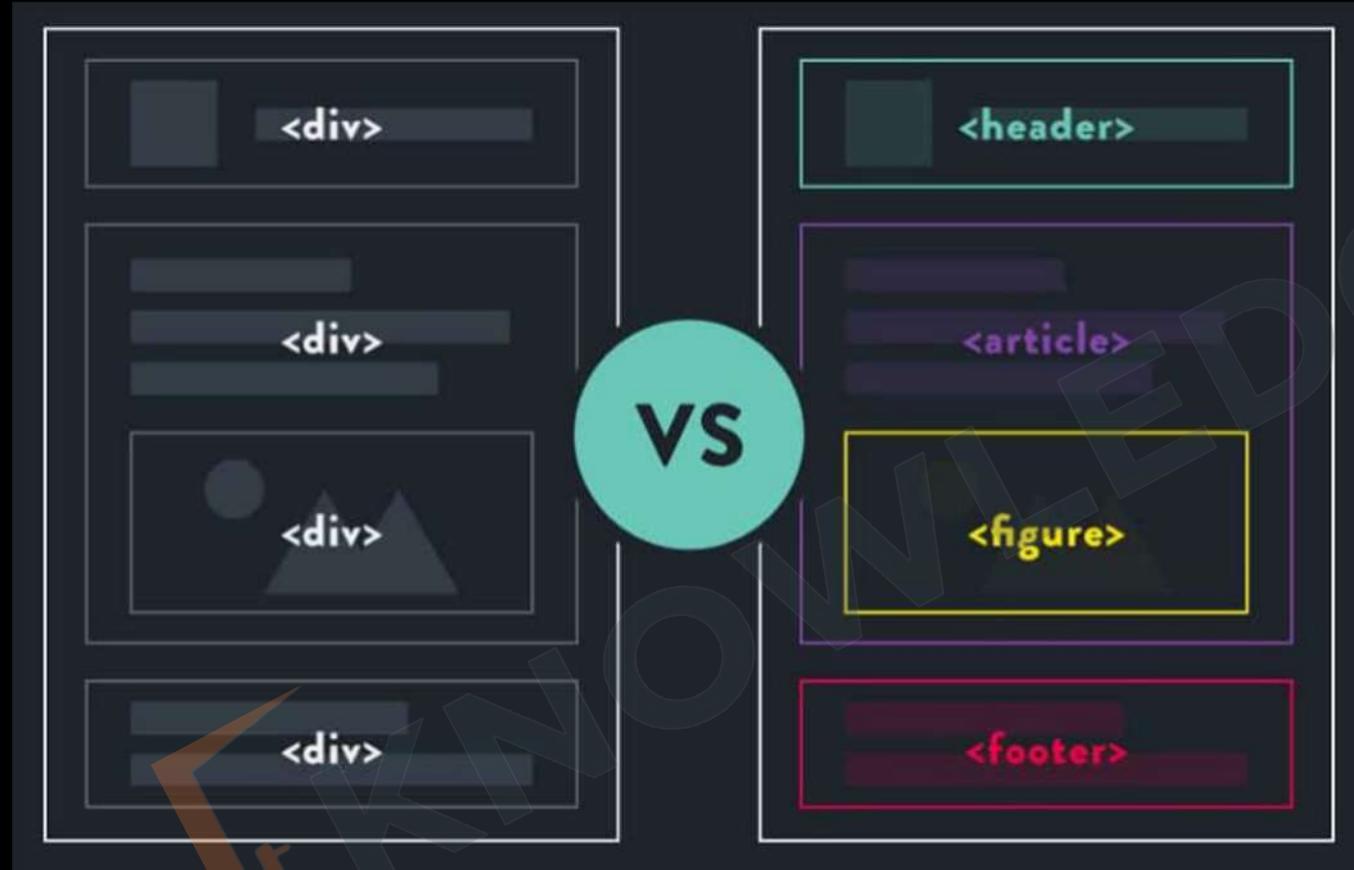
CSS Basics

- Create a heading and set the text color red.
- Create a div with id `#heading`, include CSS using all 3 ways line, style tag and external, and observe priority.
- Add comments to your `CSS` class
- Create a div, paragraph and heading and use id Selector, element selector and class selector for them.
- Create two divs with id `first` and `second` and define color for both using group selector.





HTML Core Concepts



Semantic
Tags

Semantic/Non-Semantic Tags

Semantic Tags

- Meaningful: Describe content.
- SEO: Good for search engines.
- Accessibility: Useful for screen readers.
- Examples: <header>, <footer>, <article>, <section>, <nav>.

Non-Semantic Tags

- Generic: No specific meaning.
- For Styling: Used for layout.
- No SEO: Not SEO-friendly.
- Examples: <div>, , <i>, .

Div Tags

The image shows a code editor on the left and a web browser on the right. The code editor displays the following HTML code:

```
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <div>
        <p> Lorem ipsum dolor sit amet
        consectetur adipisicing elit. Itaque quae
        veritatis, repellendus nam adipisci fuga
        nulla eos nobis.</p>
    </div>
</body>
</html>
```

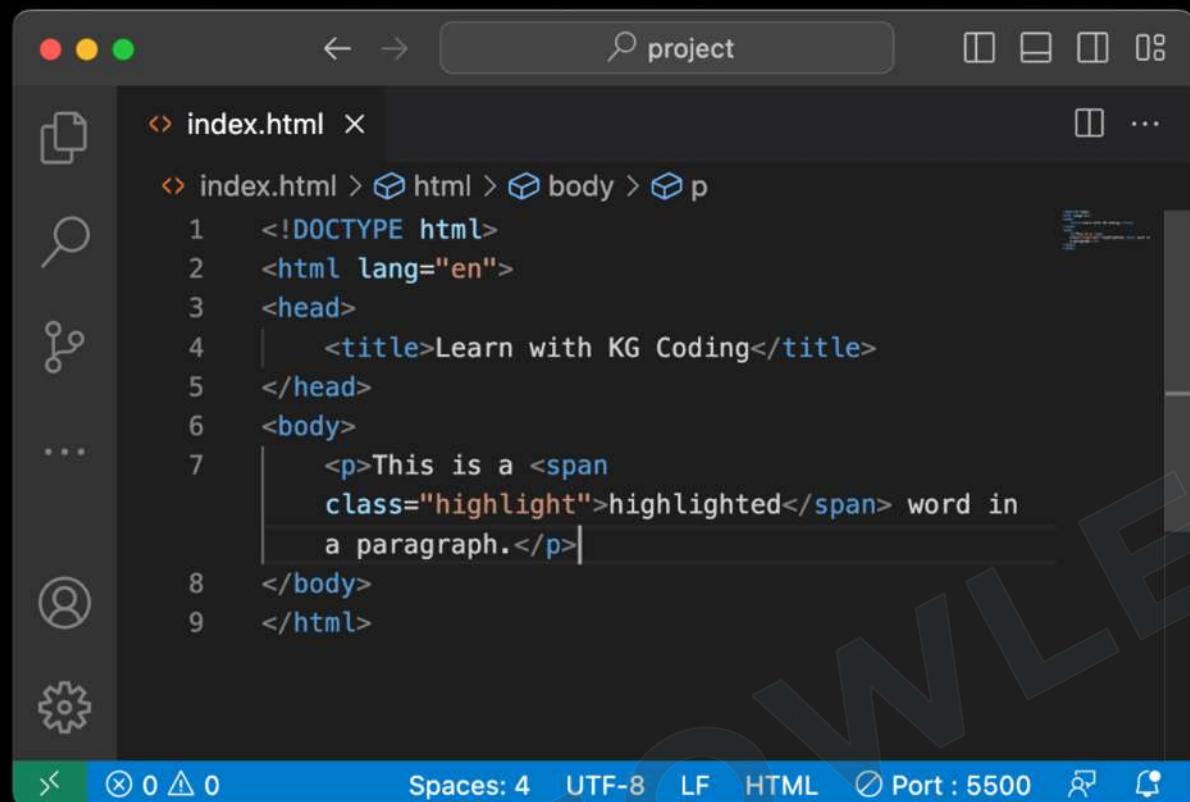
The browser window shows the rendered content of the page:

Learn with KG Coding

Lore ipsum dolor sit amet consectetur adipisicing elit. Itaque quae veritatis, repellendus nam adipisci fuga nulla eos nobis.

1. **Purpose:** Acts as a container for other **HTML** elements.
2. **Non-Semantic:** Doesn't provide inherent meaning to enclosed content.
3. **Styling:** Commonly used for layout and styling via **CSS**.
4. **Flexibility:** Highly versatile and can be customized using classes or IDs.

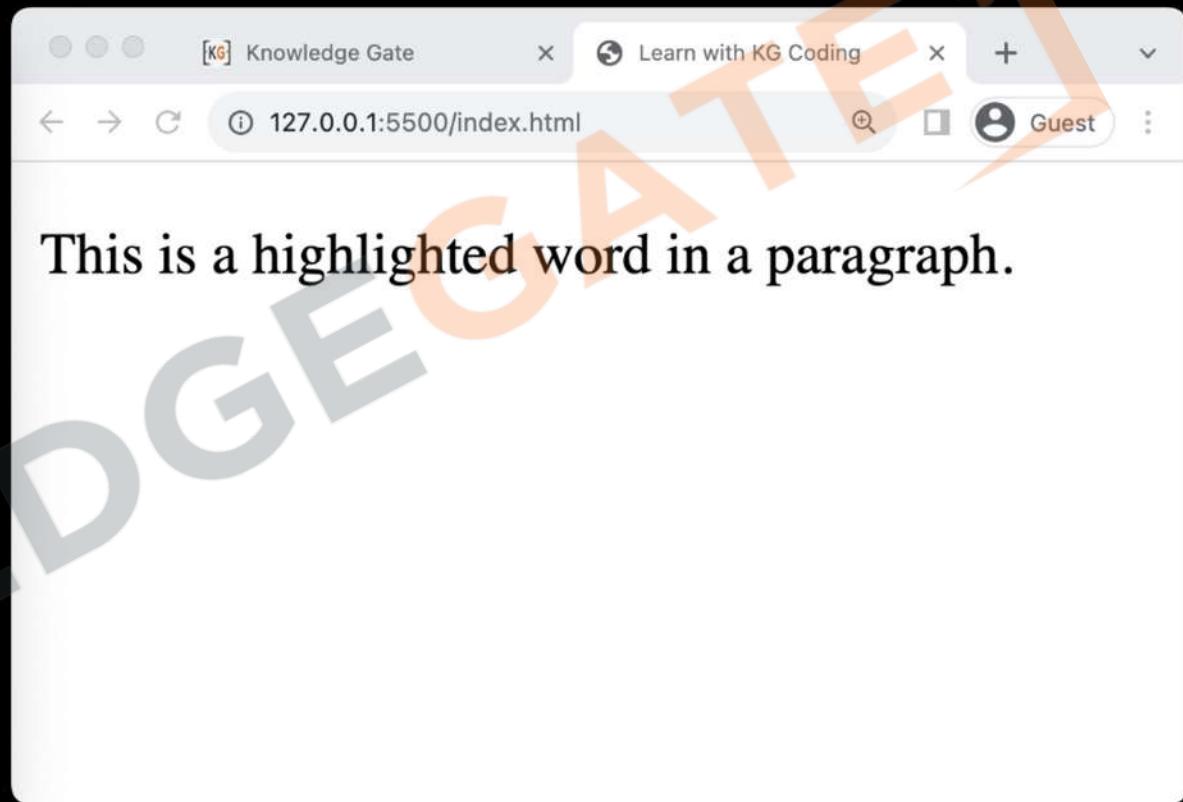
Span Tags



A screenshot of a code editor window titled "project". The file "index.html" is open, showing the following code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <p>This is a <span class="highlight">highlighted</span> word in a paragraph.</p>
</body>
</html>
```

The word "highlighted" is enclosed in a `` tag with the class attribute set to "highlight". This code is displayed in a dark-themed code editor.



1. **Purpose:** Used for inline elements to style or manipulate a portion of text.
2. **Non-Semantic:** Doesn't add specific meaning to the enclosed text.
3. **Styling:** Commonly used for changing color, font, or adding effects via CSS.
4. **Inline Nature:** Doesn't break text flow or create a new block-level element.

HTML and Project Structure



Body
Tags

Header Tag

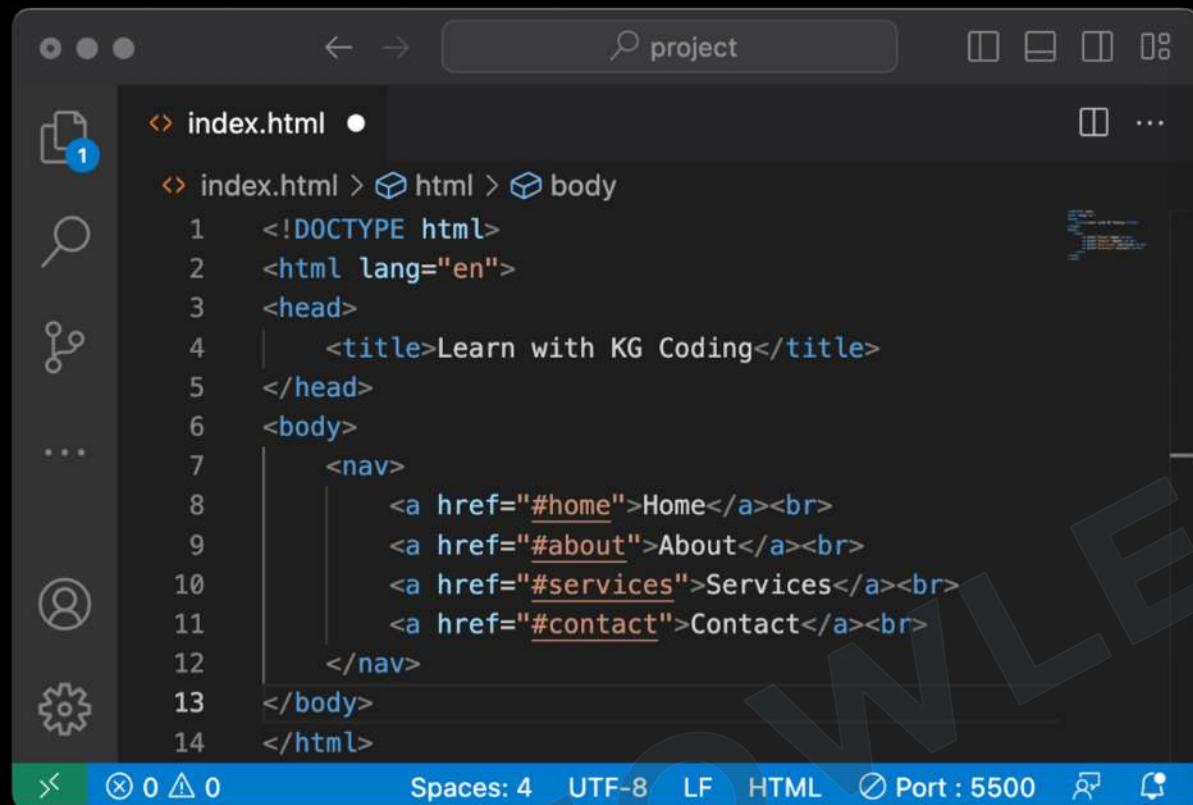
The image shows a code editor on the left and a browser window on the right. The code editor displays the following HTML code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <header><big>This is the Header</big></header>
</body>
</html>
```

The browser window shows the rendered output: "This is the Header".

1. **Purpose:** Used to contain introductory content or navigation links.
2. **Semantic:** It's a semantic tag, providing meaning to the enclosed content.
3. **Location:** Commonly found at the top of web pages, but can also appear within `<article>` or `<section>` tags.
4. **Multiple Instances:** Can be used more than once on a page within different sections.

Navigation Tags

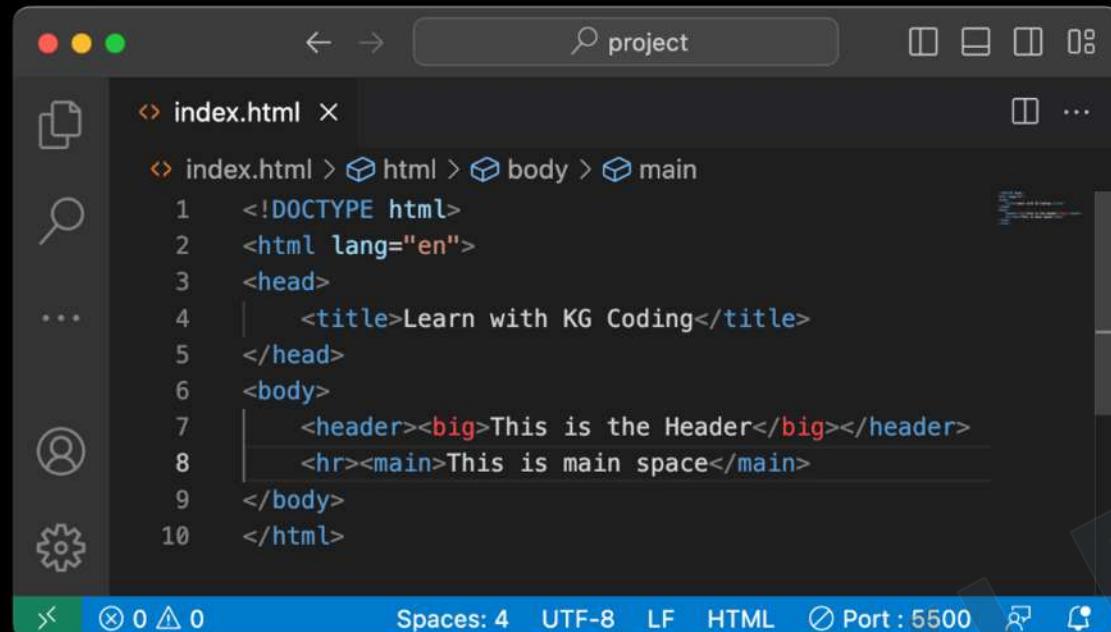


```
<> index.html •  
<> index.html > html > body  
1  <!DOCTYPE html>  
2  <html lang="en">  
3  <head>  
4  |   <title>Learn with KG Coding</title>  
5  </head>  
6  <body>  
7  |   <nav>  
8  |   |   <a href="#home">Home</a><br>  
9  |   |   <a href="#about">About</a><br>  
10 |   |   <a href="#services">Services</a><br>  
11 |   |   <a href="#contact">Contact</a><br>  
12 |   </nav>  
13 </body>  
14 </html>
```



1. **Purpose:** Encloses navigation links or menus.
2. **Semantic:** Signals that the content is meant for navigating the site.
3. **Common Content:** Usually contains lists ``, `` of links `<a>`.
4. **Accessibility:** Aids screen readers in identifying site navigation.

Main Tag

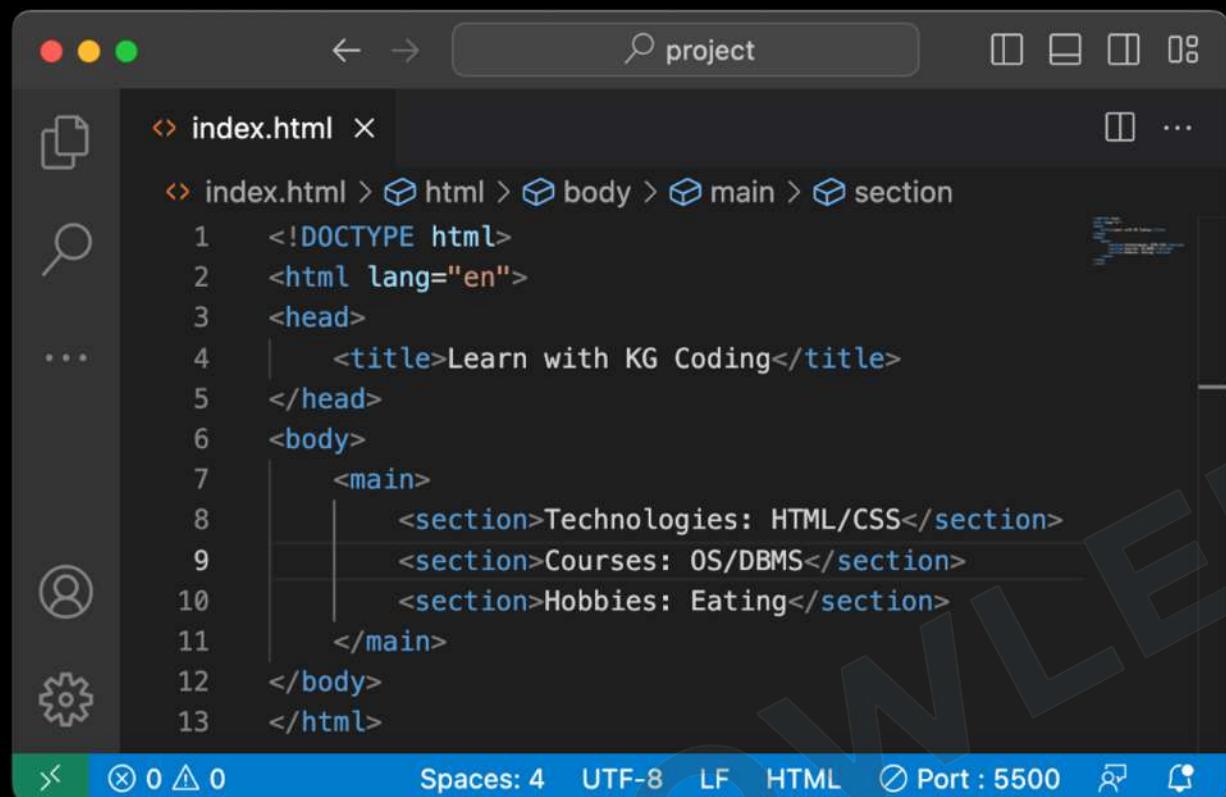


```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <header><big>This is the Header</big></header>
    <hr><main>This is main space</main>
</body>
</html>
```



1. **Purpose:** Encloses the primary content of a webpage.
2. **Semantic:** Adds meaning, indicating the main content area.
3. **Unique:** Should appear only once per page.
4. **Accessibility:** Helps screen readers identify key content.
5. **Not for Sidebars:** Excludes content repeated across multiple pages like site navigation or footer.

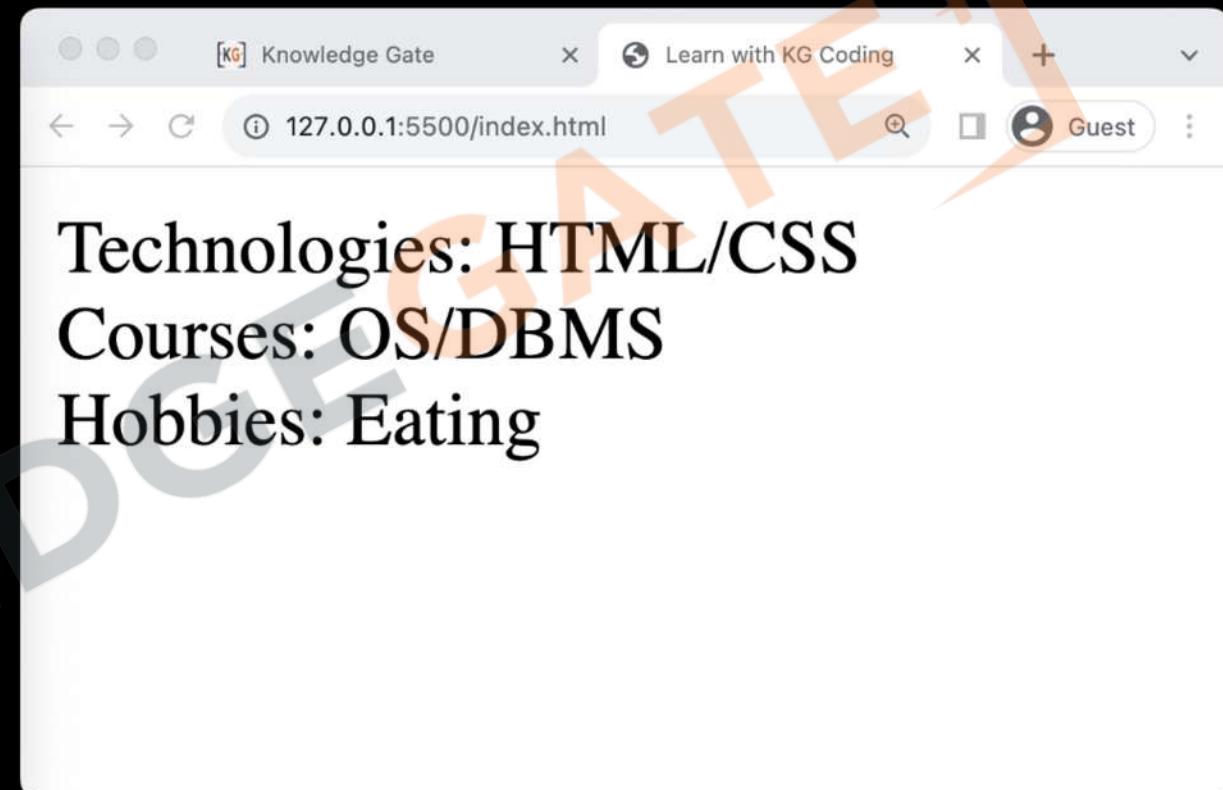
Section Tag



A screenshot of a code editor window titled "project". The file "index.html" is open, showing the following code:

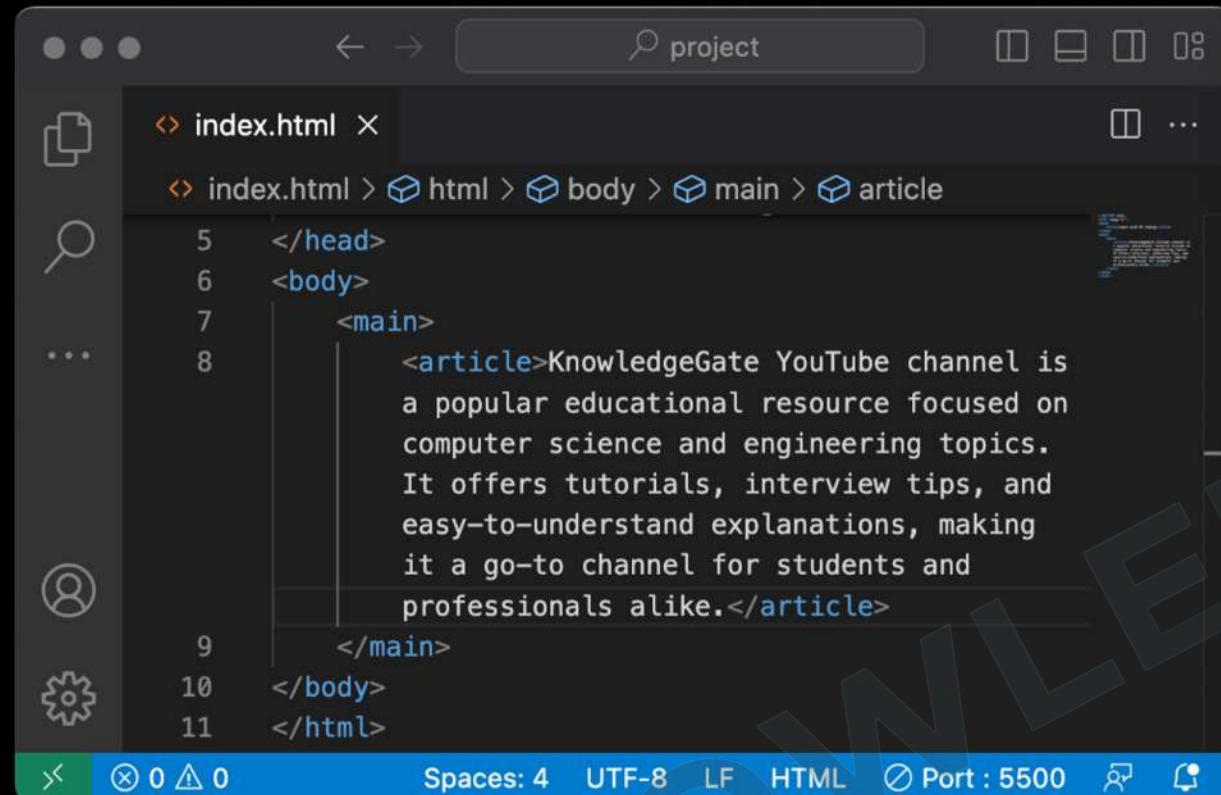
```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <main>
        <section>Technologies: HTML/CSS</section>
        <section>Courses: OS/DBMS</section>
        <section>Hobbies: Eating</section>
    </main>
</body>
</html>
```

The code editor has a dark theme with light-colored syntax highlighting. A sidebar on the left contains icons for file, search, and settings. The bottom bar shows "Spaces: 4", "UTF-8", "LF", "HTML", "Port : 5500", and other status indicators.



1. **Purpose:** Groups related content in a distinct section.
2. **Semantic:** Adds structure and meaning.
3. **Headers:** Often used with a heading `<h1>` to `<h6>` to indicate section topic.
4. **Nested:** Can be nested within other `<section>` or `<article>` tags.

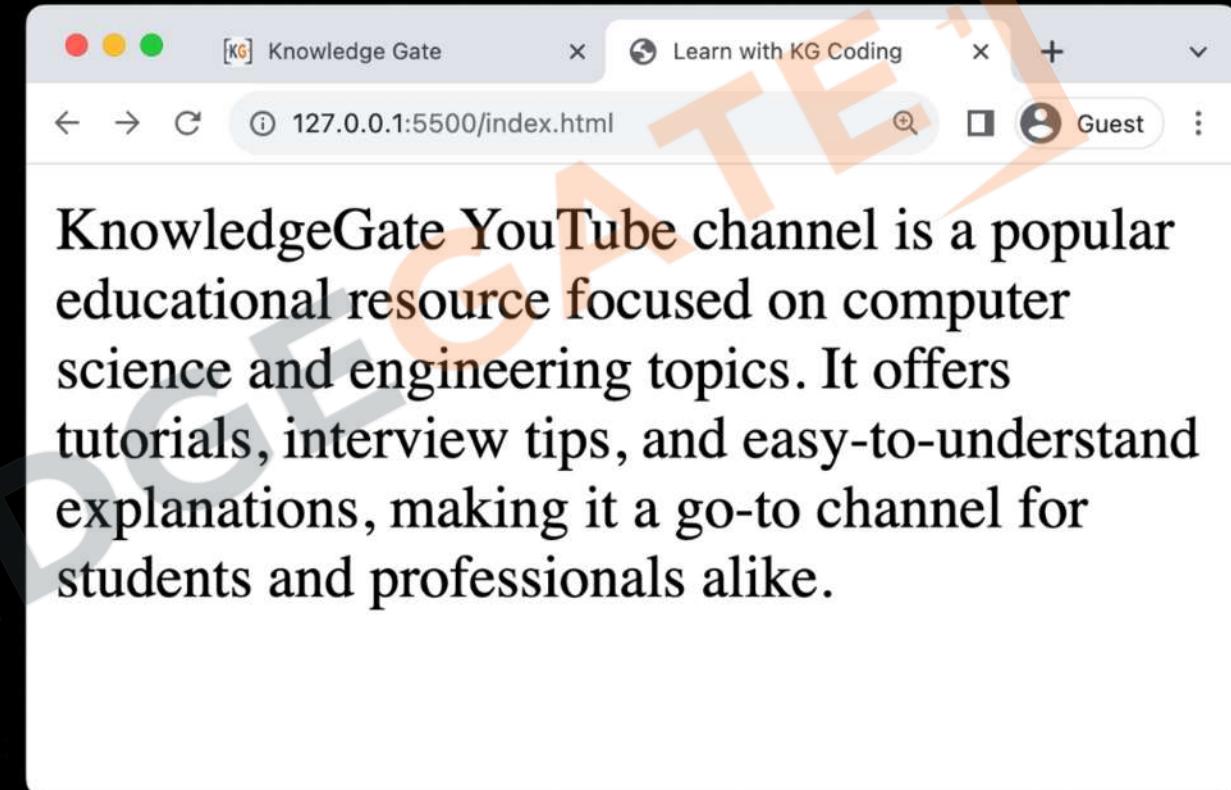
Article Tag



A screenshot of a code editor window titled "project". The file "index.html" is open, showing the following HTML code:

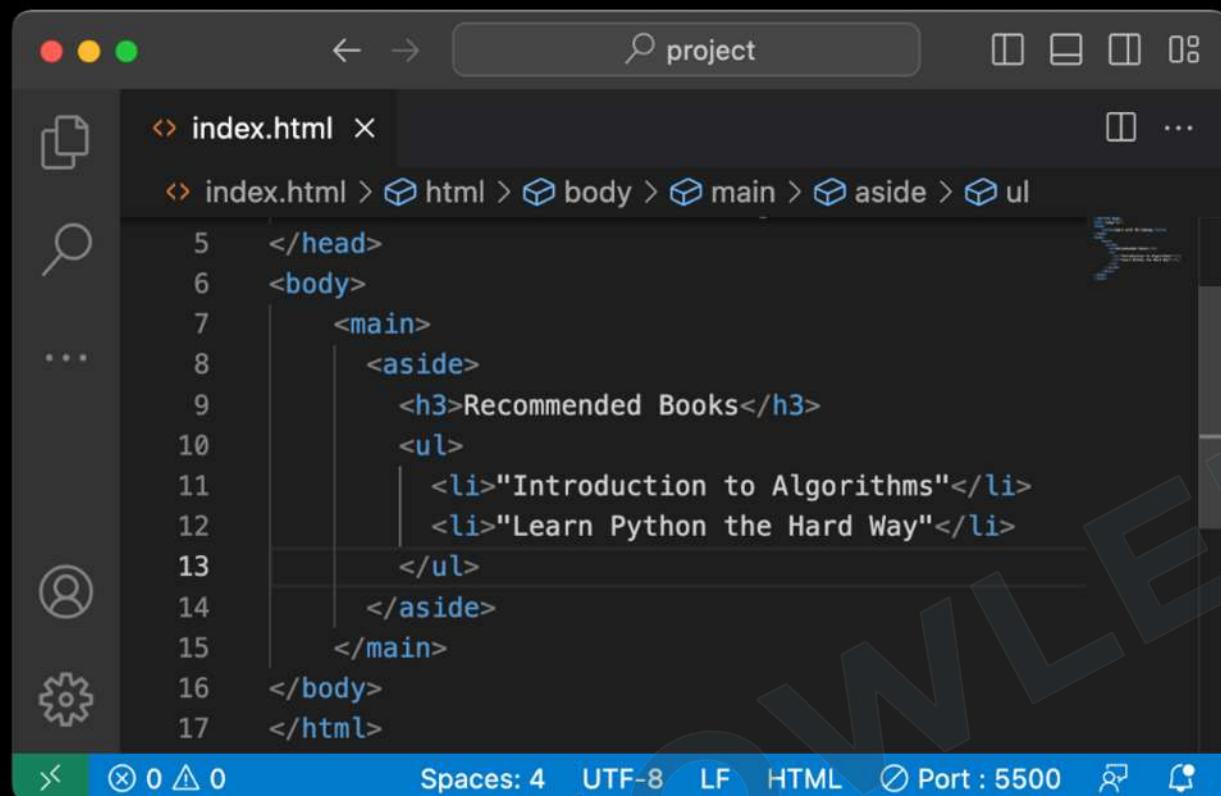
```
<index.html>
<html>
  <head>
    ...
  </head>
  <body>
    <main>
      <article>KnowledgeGate YouTube channel is a popular educational resource focused on computer science and engineering topics. It offers tutorials, interview tips, and easy-to-understand explanations, making it a go-to channel for students and professionals alike.</article>
    </main>
  </body>
</html>
```

The "article" tag is highlighted in blue. The status bar at the bottom shows "Spaces: 4" and "Port : 5500".

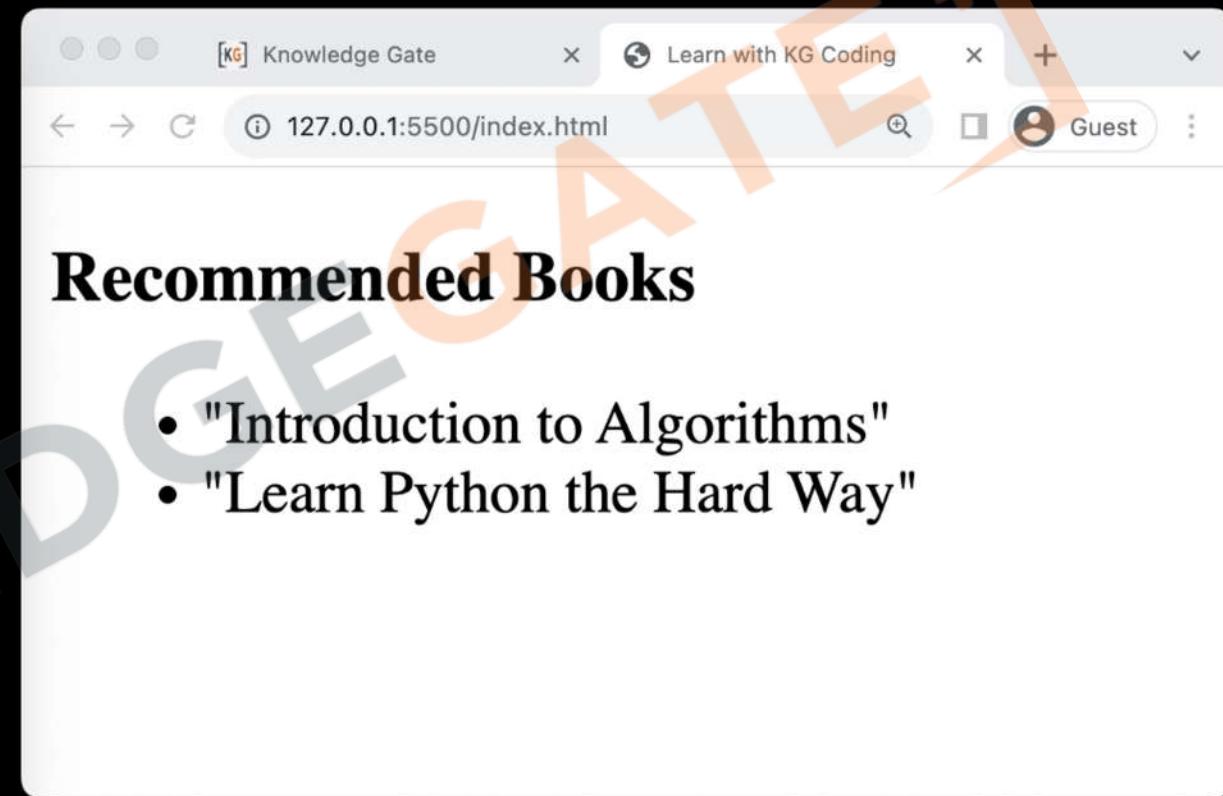


1. **Purpose:** Encloses content that stands alone, like a **blog post or news** story.
2. **Semantic:** Provides contextual meaning.
3. **Independence:** Content should make sense even if taken out of the page context.
4. **Multiple Instances:** Can be used multiple times on the same page

Aside Tag

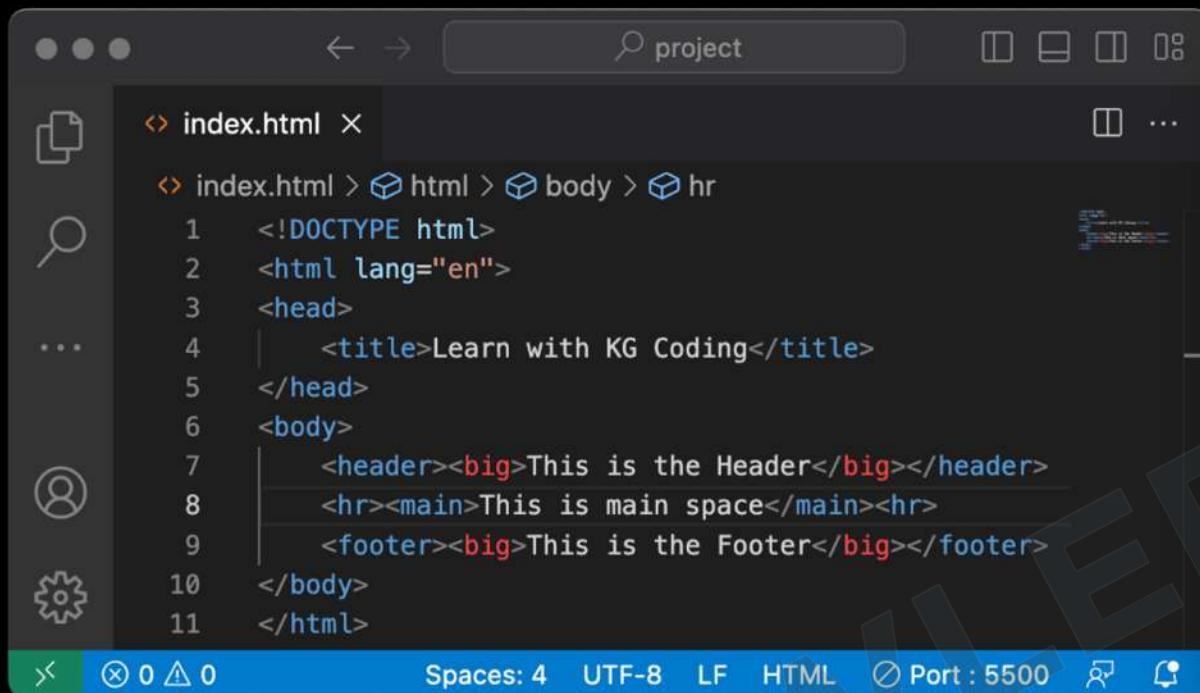


```
</head>
<body>
  <main>
    <aside>
      <h3>Recommended Books</h3>
      <ul>
        <li>"Introduction to Algorithms"</li>
        <li>"Learn Python the Hard Way"</li>
      </ul>
    </aside>
  </main>
</body>
</html>
```

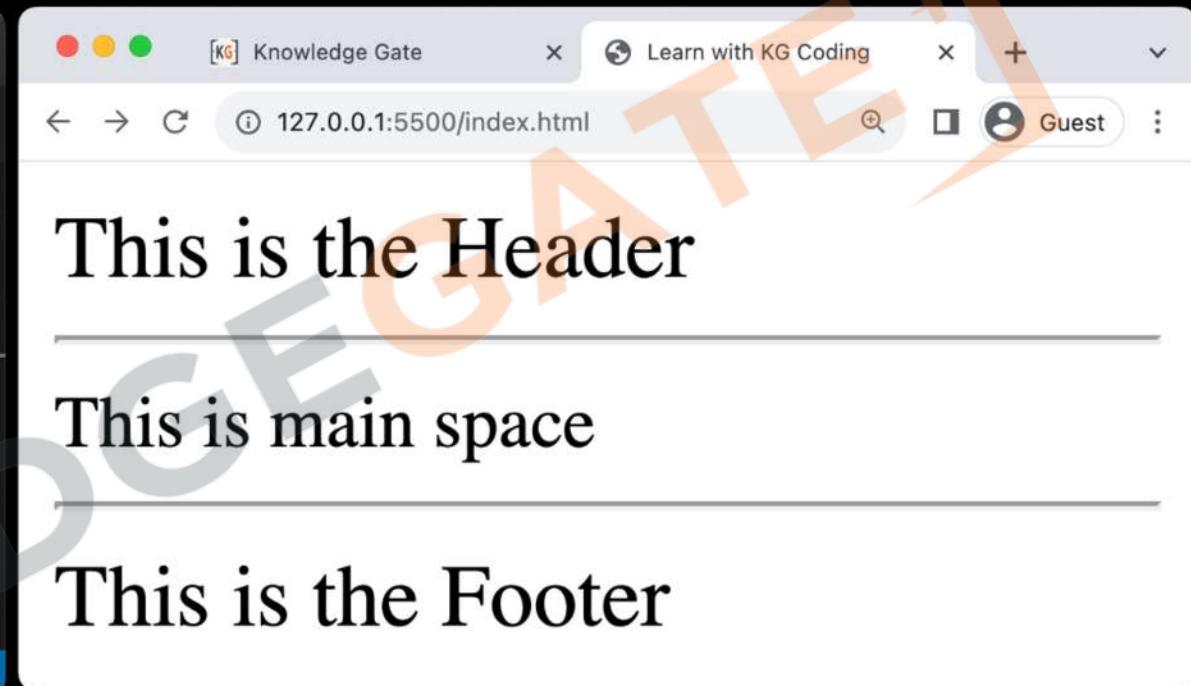


1. **Purpose:** Contains sidebar or supplementary content.
2. **Semantic:** Indicates content tangentially related to the main content.
3. **Not Crucial:** Content is not essential to understanding the main content.
4. **Examples:** Could hold **widgets, quotes, or ads**.

Footer Tag



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <header><big>This is the Header</big></header>
    <hr><main>This is main space</main><hr>
    <footer><big>This is the Footer</big></footer>
</body>
</html>
```



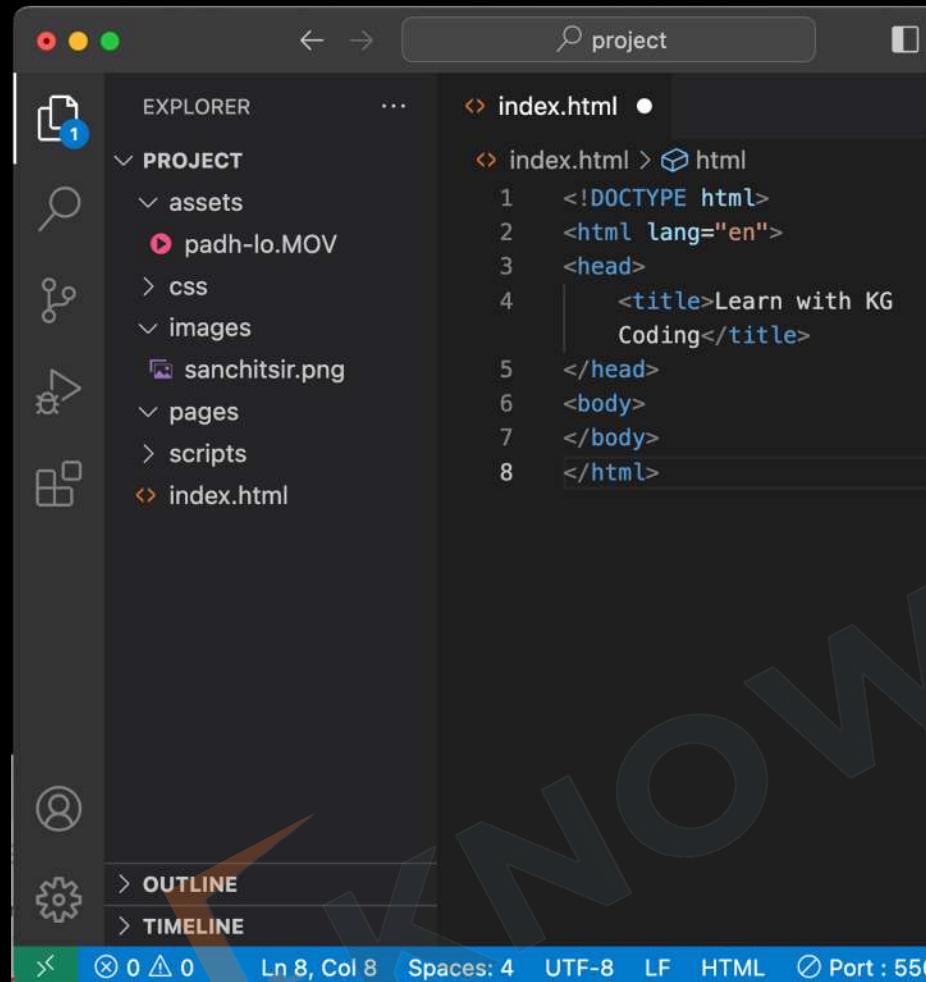
1. **Purpose:** For footer content like extra info or links.
2. **Semantic:** Provides meaning to enclosed content.
3. **Location:** Typically at the **bottom** of pages or sections.
4. **Content:** Includes copyrights, contact info, and social links.
5. **Multiple Instances:** Can be used more than once on a page.

HTML Core Concepts

Folder
Structure



Recommended Folder Structure



The screenshot shows a code editor interface with the following details:

- EXPLORER** view: Shows a project structure with a file named "padh-lo.MOV".
- PROJECT** view: Shows sub-folders "assets", "css", "images", "pages", "scripts", and the main file "index.html".
- index.html** content (Preview):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
</body>
</html>
```
- OUTLINE** and **TIMELINE** views are also visible.
- Bottom status bar: Line 8, Col 8, Spaces: 4, UTF-8, LF, HTML, Port: 550.

1. **Root Directory:** Main folder containing all website files.
2. **HTML Files:** Store main .html files at the root level for easy access.
3. **CSS Folder:** Create a css/ folder for all Cascading Style Sheets.
4. **JS Folder:** Use a scripts/ folder for JavaScript files.
5. **Images Folder:** Store images in an images/ or images/ folder.
6. **Assets:** Other assets like fonts can go in an assets/ folder.
7. **Sub-directories:** For multi-page websites, use sub-folders to categorize content.

Block / Inline Elements

Block Elements

- New Line: Start on a new line.
- Full Width: Take up all horizontal space.
- Styling: Can have margins and padding.
- Size: Width and height can be set.
- Examples: <div>, <p>, <h1>, , .

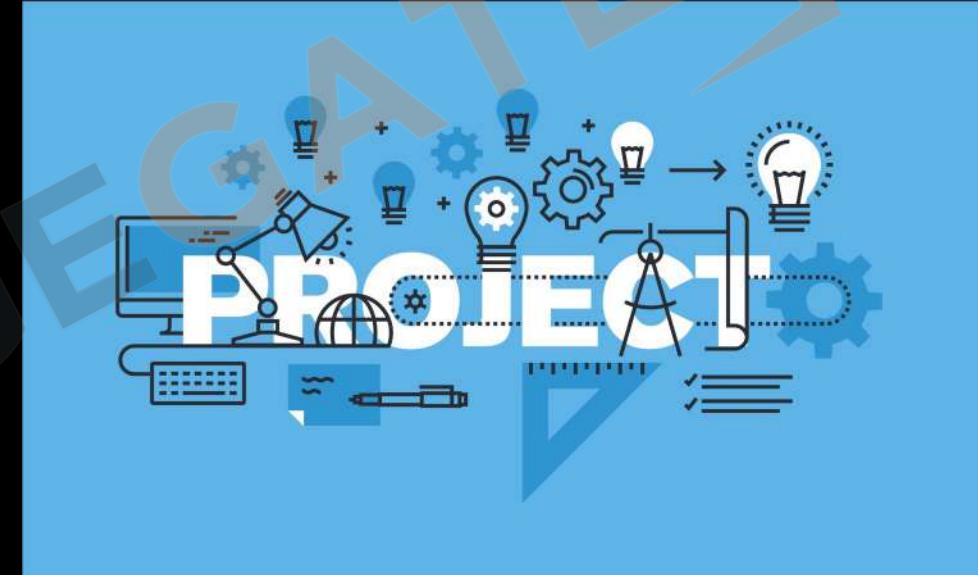
Inline Elements

- Flow: Stay in line with text.
- Width: Just as wide as the content.
- No Break: No new line between elements.
- Limited Styling: Can't set size easily.
- Examples: , <a>, , , .

Practise Exercise

HTML Core Concepts

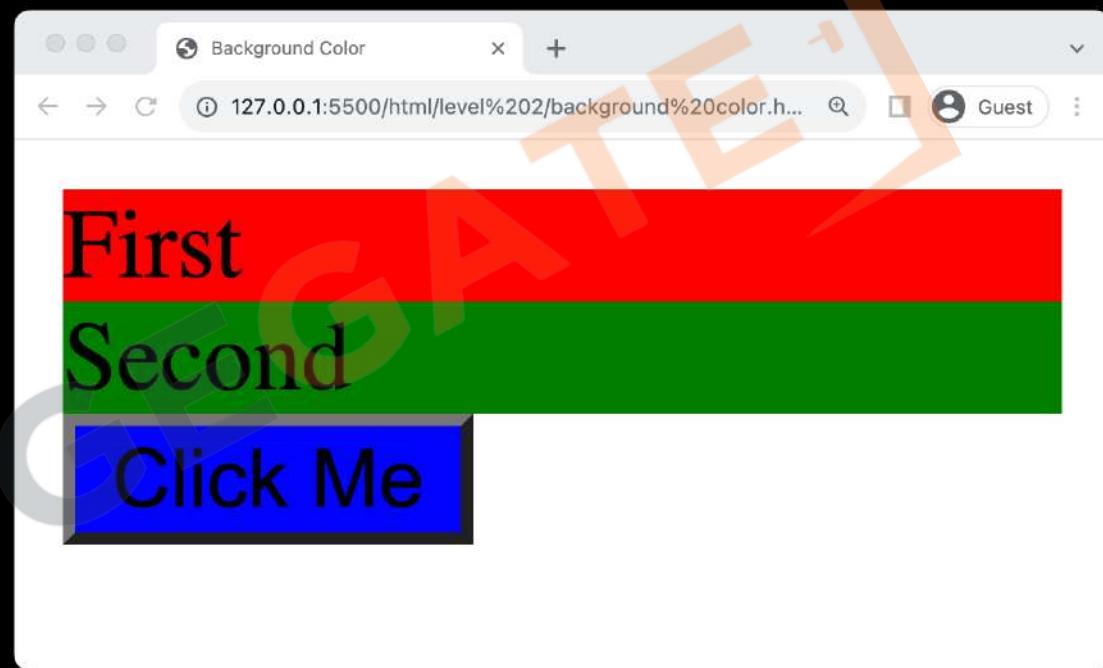
1. Create a **page** with header, footer, main(section, article, aside tag).
2. Make sure the project from level 3 has correct **folder structure**.
3. Create **groupings** of multiple tags using div.
4. Create **navigation** to important sections of your page.





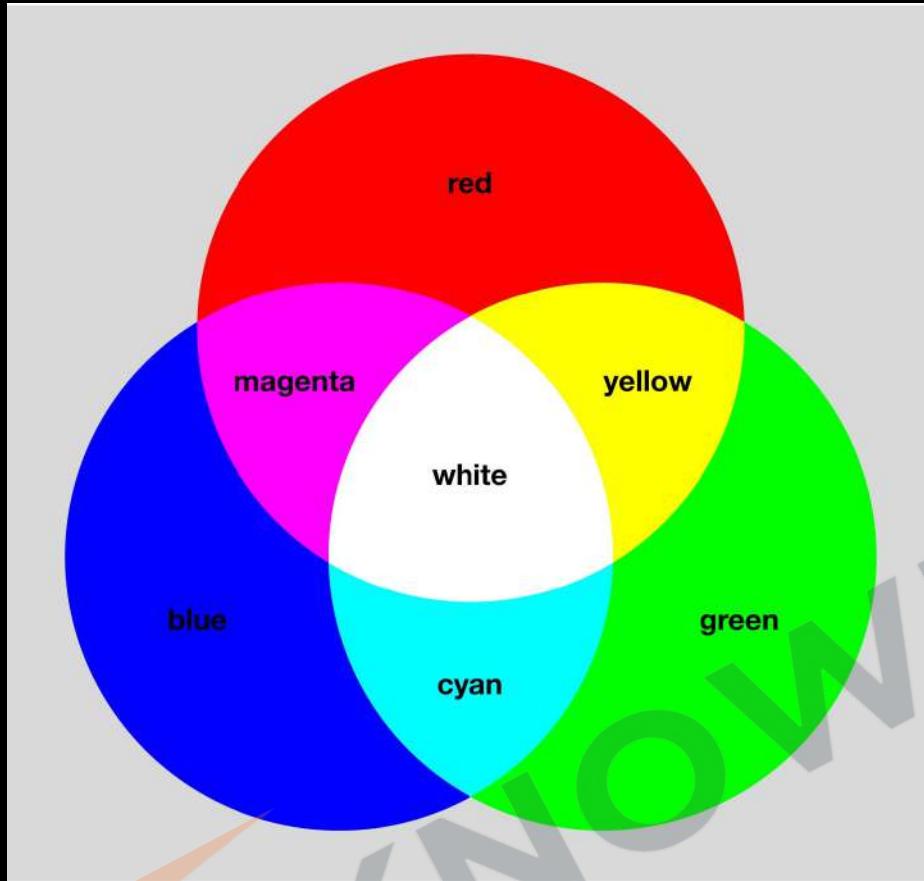
Background Color

```
<head>
    <title>Background Color</title>
    <style>
        #first { color: black; background-color: red; }
        #second { color: black; background-color: green; }
        button { color: black; background-color: blue; }
    </style>
</head>
<body>
    <div id="first">First</div>
    <div id="second">Second</div>
    <button>Click Me</button>
</body>
```



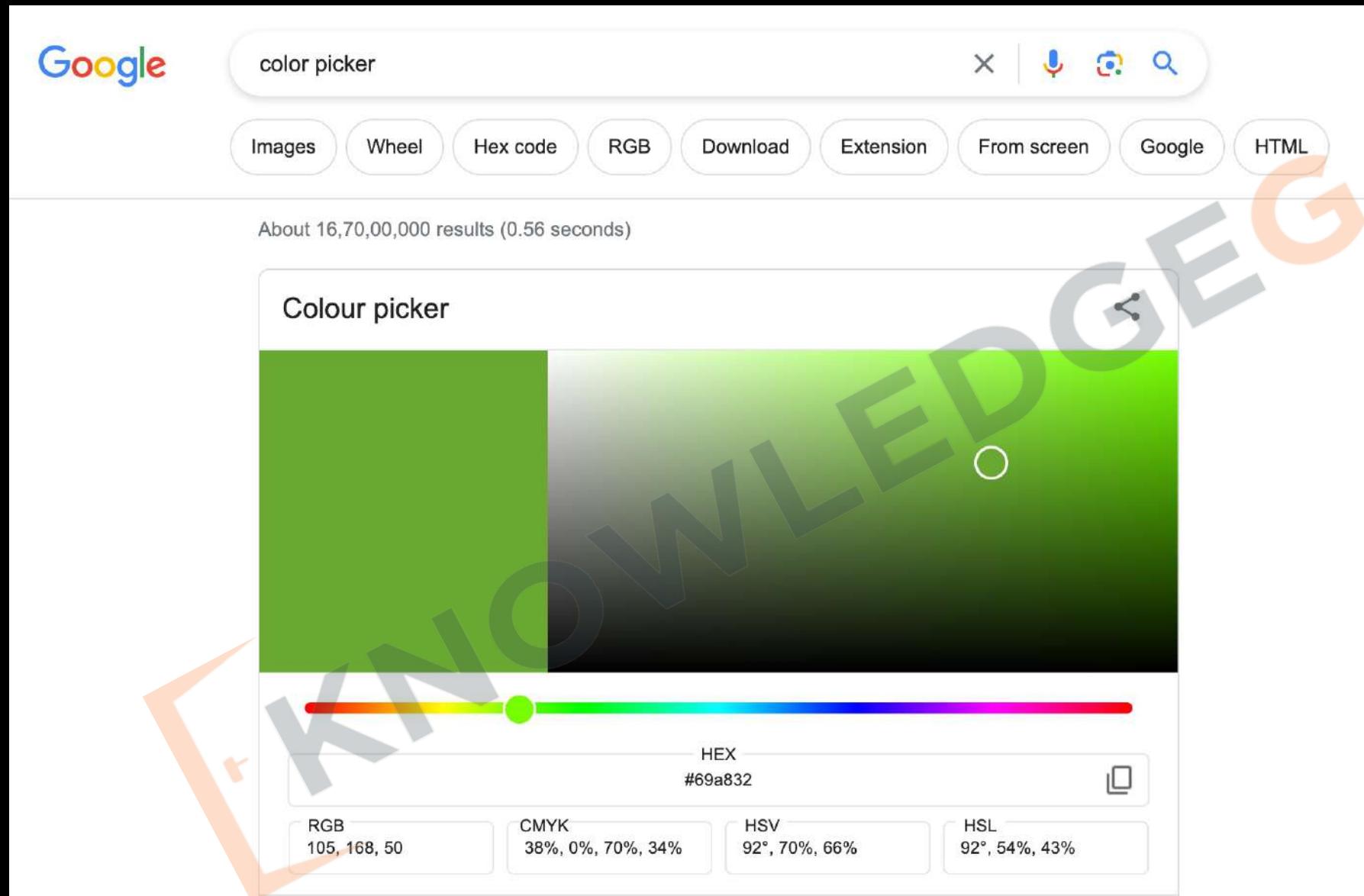
- **Definition:** Sets the background color of an element.
- **Syntax:** Utilized as `background-color: color;`
- **Visual Appeal:** Enhances the visual appeal and contrast of webpage elements.

Color System (Color Theory)



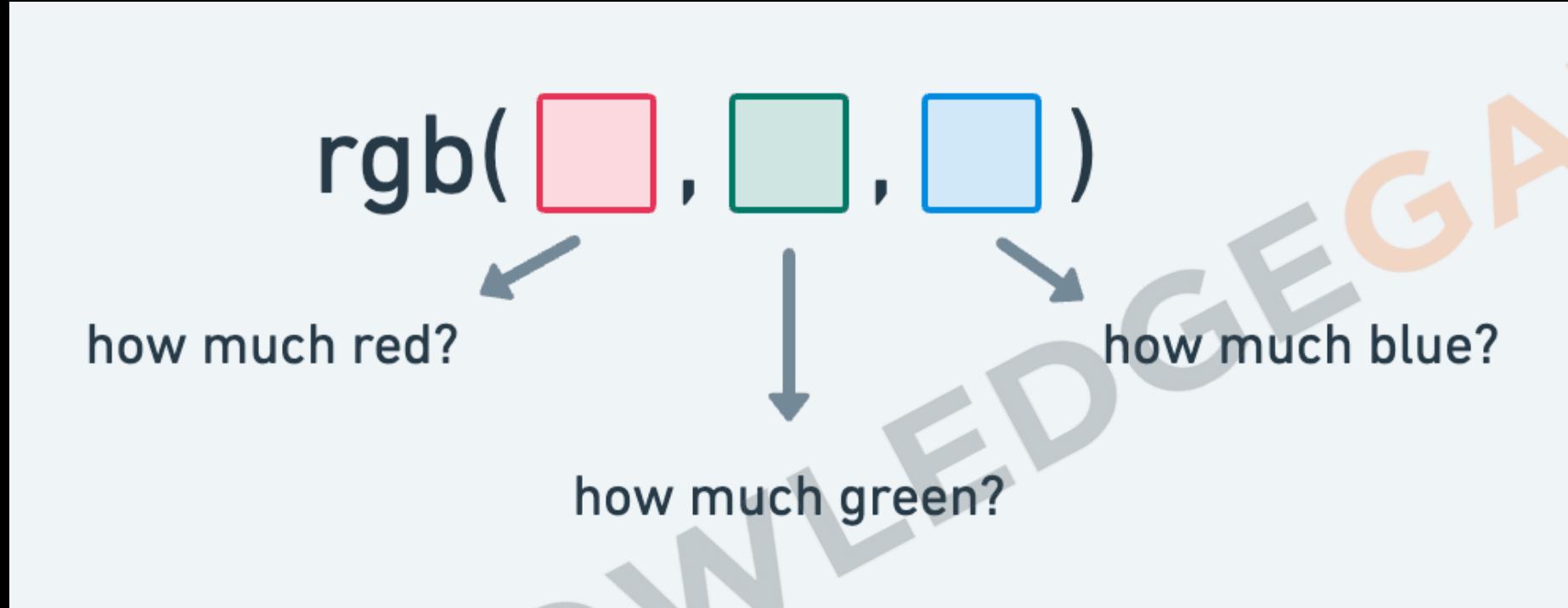
- **RGB Model:** Creates colors by mixing Red (R), Green (G), and Blue (B) light sources.
- **Additive Model:** More light means increased brightness.
- **Primary Colors:** R, G, and B are the foundational colors.
- **White & Black:** All combined yield white; absence equals black.
- **Color Depth:** Allows for millions of color variations.

Color System (color picker)



KNOWLEDGE GATE

Color System (RGB Color Model)



- **Three Channels:** Consists of Red (R), Green (G), and Blue (B) channels to create a variety of colors.
- **Syntax:** Utilized as `rgb(r, g, b)` where r, g, and b are values between 0 and 255.

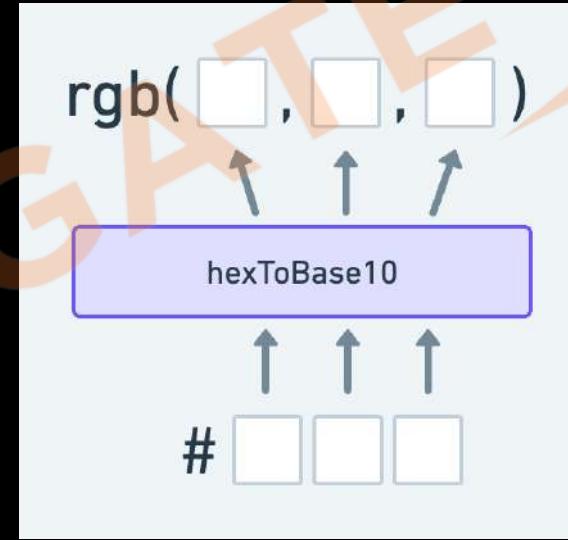
Color System (RGB Color Model)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>RGB Color</title>
</head>
<body>
    <div style="background-color: red; color: white; padding: 10px; margin-bottom: 5px;">First</div>
    <div style="background-color: green; color: white; padding: 10px; margin-bottom: 5px;">Second</div>
    <div style="background-color: blue; color: white; padding: 10px; margin-bottom: 5px;">Third</div>
    <div style="background-color: #1e8449; color: white; padding: 10px; margin-bottom: 5px;">Fourth</div>
</body>
</html>
```



Color System (HEX Color Model)

- **Hexadecimal Codes:** Represents colors using hexadecimal values, consisting of **6 digits** combined from numbers and letters **(A-F)**.
- **Syntax:** Written as **#RRGGBB**
- **Easy Color Matching:** Facilitates easy color matching with graphic design tools and branding colors.
- **Web Standards:** Widely supported and a common standard for defining colors in web design



Color System (HEX Color Model)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Hex Color</title>
</head>
<body>
    <div style="background-color: #ff0000">First</div>
    <div style="background-color: #00ff00">Second</div>
    <div style="background-color: #0000ff">Third</div>
    <div style="background-color: #402ae9">Fourth</div>
</body>
</html>
```



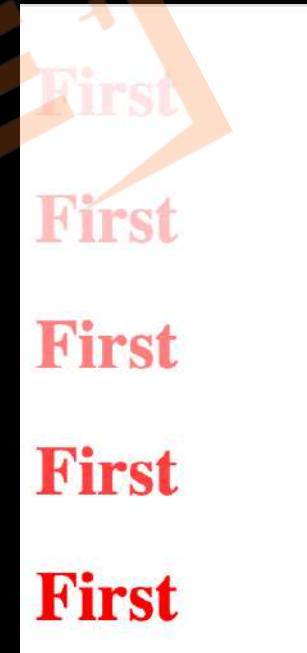
Color System (Alpha Channel)

- **RGBA:** RGB's extension, includes alpha for opacity control (0-1 range).
- **Transparency Control:** Facilitates the adjustment of transparency levels in colors.
- **Visual Effects:** Enables the creation of visual effects like shadows and overlays.
- **Layering:** Assists in layering elements with varying degrees of visibility.



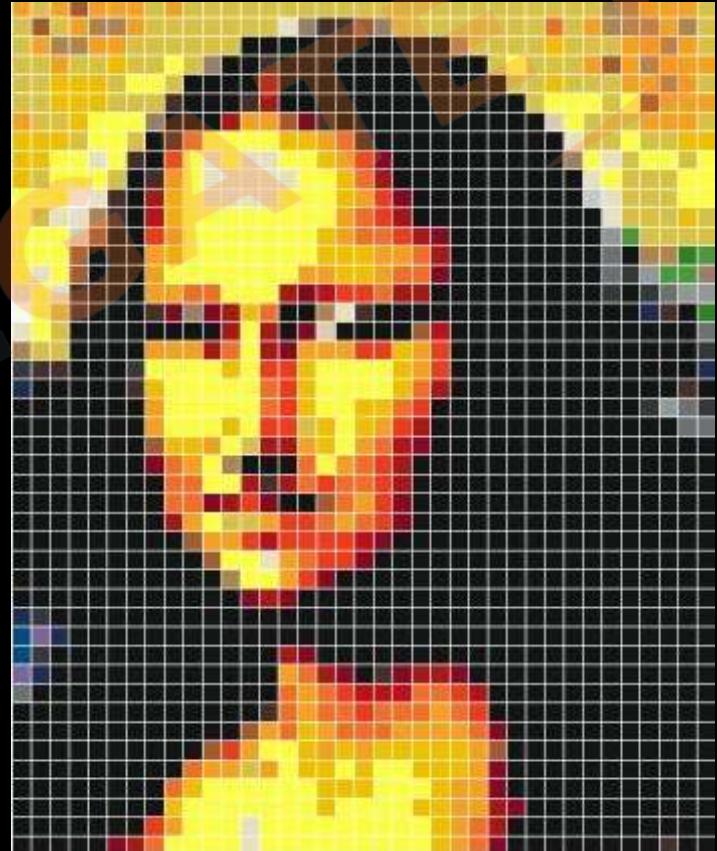
Color System (Alpha Channel)

```
<h1 style="color: □rgb(255,0,0,0.1);">First</div>
<h1 style="color: □rgb(255,0,0,0.25);">First</div>
<h1 style="color: □rgb(255,0,0,0.5);">First</div>
<h1 style="color: □rgb(255,0,0,0.75);">First</div>
<h1 style="color: □rgb(255,0,0,1.0);">First</div>
```



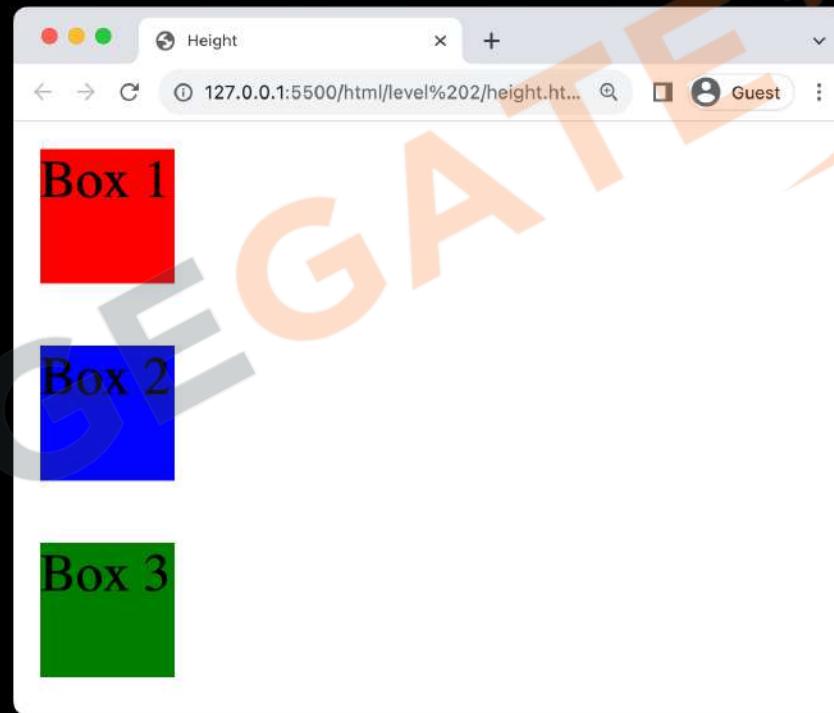
Absolute Units

- **Definition:** Pixels (px) are fixed-size units, representing a dot on a computer screen.
- **Precision:** Allows for precise control over element dimensions.
- **Graphics & Web Design:** Commonly used in graphics and web design for setting font sizes, margins, and more.
- **Cross-Browser Consistency:** Provides consistency across different browsers.
- **High-DPI Displays:** Can vary in appearance on high-DPI (dots per inch) displays.



Height & Width Property

```
<head>
  <title>Height</title>
  <style>
    .box { height: 40px; width: 40px; }
    #box1 {background-color: red;}
    #box2 {background-color: blue;}
    #box3 {background-color: green;}
  </style>
</head>
<body>
  <div id="box1" class="box">Box 1</div> <br>
  <div id="box2" class="box">Box 2</div> <br>
  <div id="box3" class="box">Box 3</div>
</body>
```



- **Dimensions Control:** Used to specify the **height** and **width** of elements.
- **Unit Variability:** Can use units like pixels (px)
- **Box Model Component:** Influences padding, border, and margin.
- **Min and Max Values:** Can utilize min-height, max-height, min-width, and max-width to set restrictions on dimensions.

Background image Property

- **Usage:** Adds an **image** as a background to elements.
- **Syntax:** Defined using **background-image:**
`url('path/to/image');`
- **Repetition:** Control image repetition using **background-repeat**.
- **Positioning:** Adjust image position using **background-position**.
- **Size Control:** Manipulate image size using **background-size**.
- **Background-Attachment:** Sets whether the background image **scrolls** with the element or remains fixed.
- Shorthand (**color, image, repeat, attachment, position**)

```
<head>
  <title>Background Image</title>
  <style>
    #box1 {
      background-image: url(../../../images/css.png);
      height: 500px;
      width: 500px;
    }
  </style>
</head>
<body>
  <div id="box1"></div>
</body>
```



Background image Shorthand

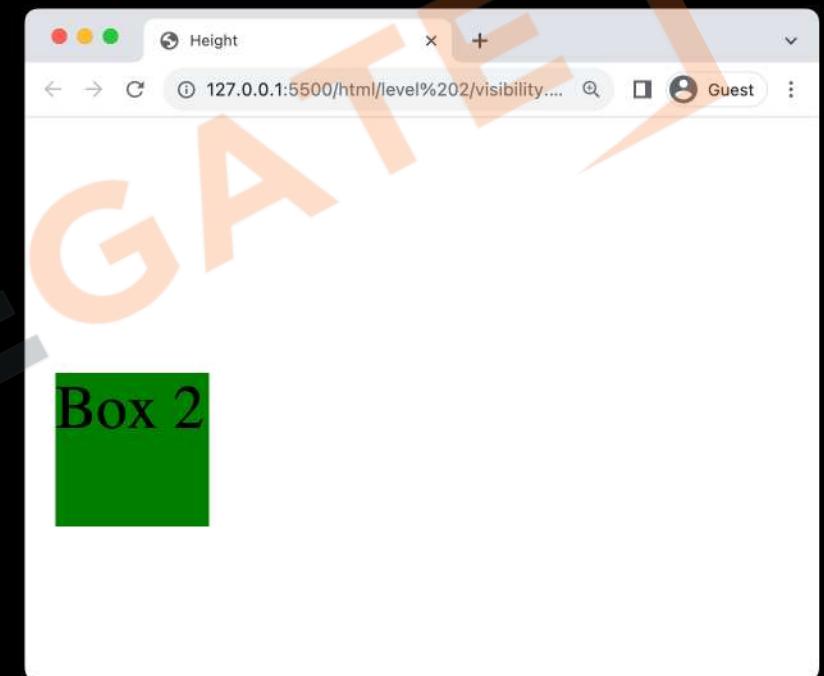
```
#my-div {  
    height: 150px;  
    width: 150px;  
    background: #00ff00 url(html.png)  
    no-repeat center/contain;  
}
```



- **Usage:** shorthand is used to combine different background properties.
- **Syntax**
`background: [background-color] [background-image] [background-position] / [background-size] [background-repeat];`
- The values in the background shorthand **can be in any order**, but the browser assigns them to properties based on their types and order.

Visibility Property

```
<head>
  <title>Visibility</title>
  <style>
    .box { height: 40px; width: 40px; }
    #box1 {background-color: red; visibility: hidden;}
    #box2 {background-color: green; visibility: visible;}
  </style>
</head>
<body>
  <div id="box1" class="box">Box 1</div> <br>
  <div id="box2" class="box">Box 2</div> <br>
</body>
```



- **Usage:** Controls the **visibility** of elements without changing the layout.
- **Values:** Can take **visible**, **hidden**, or **collapse** as values.
- **Space Occupancy:** Even when hidden, the element **occupies space**.
- **Interactivity:** Hidden elements are not accessible to **user interactions**.

Practice Set

Color System, Background & Text

- Create a div bar with text and background color with opacity
- Change the color of the main content
- Add background image to one div
- Use background shorthand property





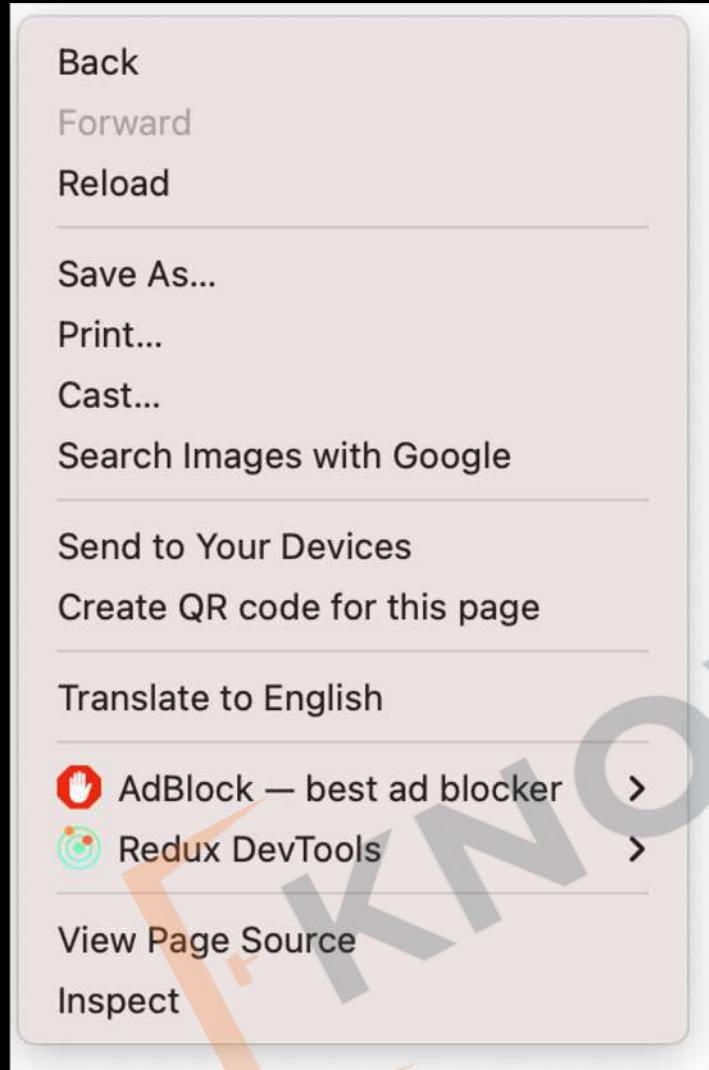
Browser Tools



Browser
Tools

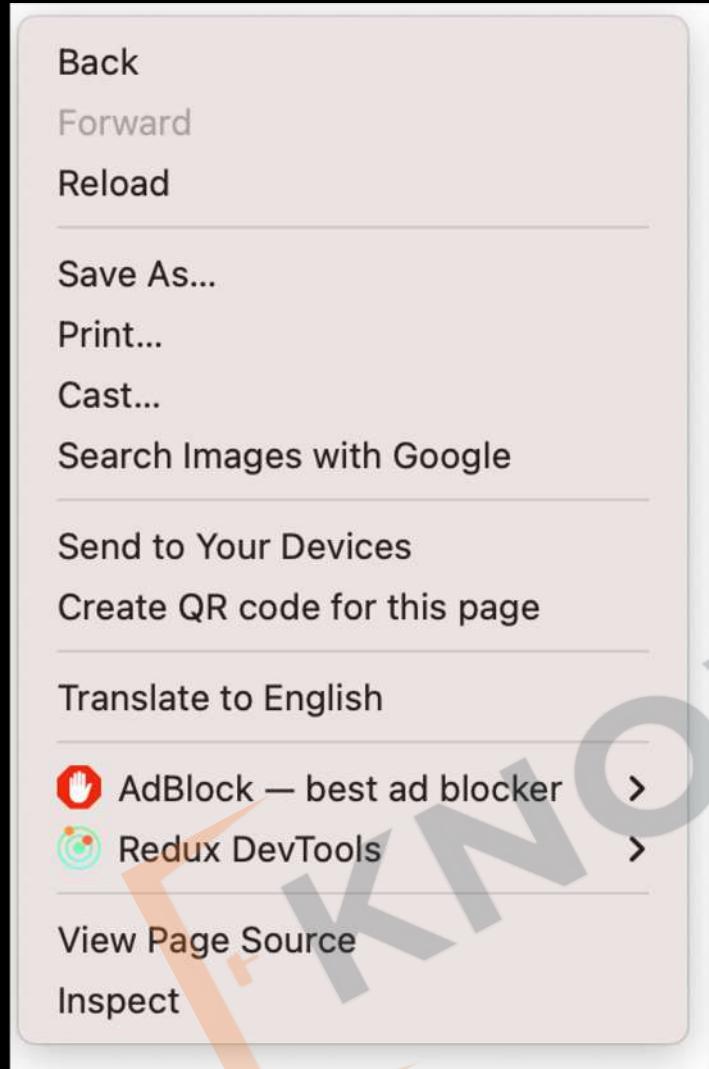
EDGE CREATE

View Page Source



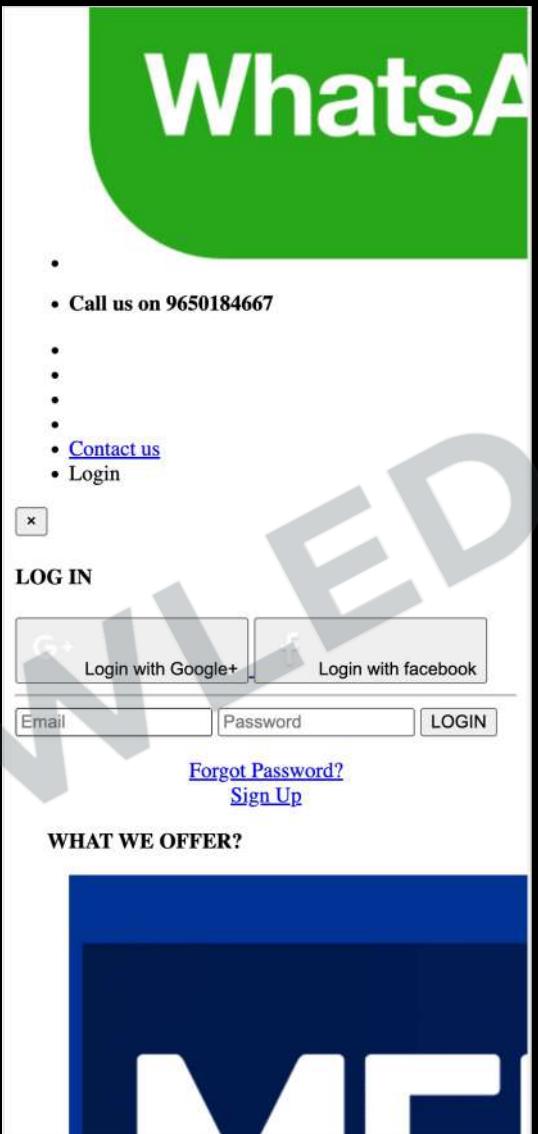
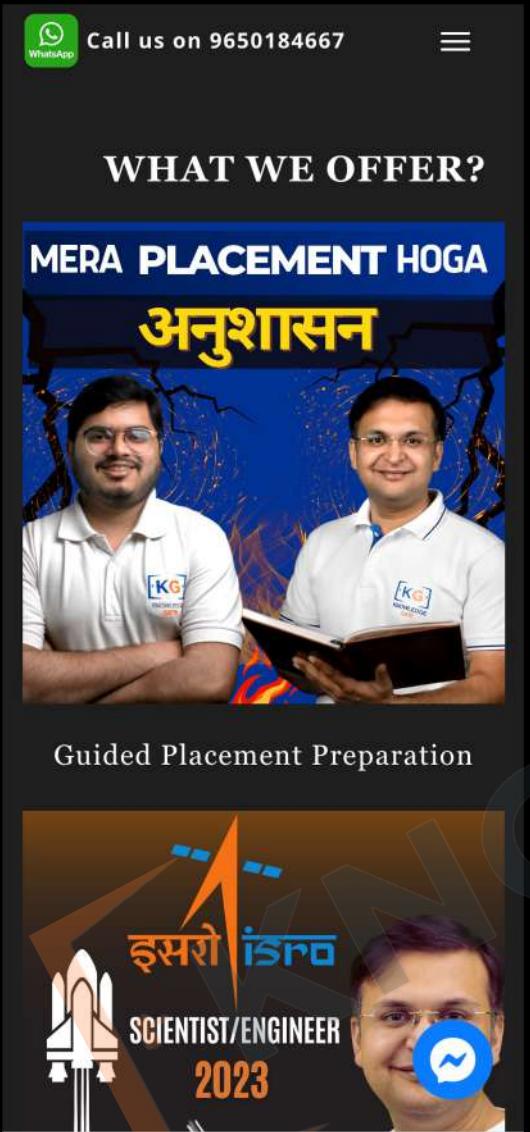
1. Displays raw **HTML** and **CSS**
2. Useful for debugging and learning
3. Shows external files like
JavaScript links

Inspect Element



1. Allows real-time editing of HTML/CSS
2. Useful for debugging and testing
3. Shows element hierarchy and layout
4. Includes console for JavaScript
5. Highlights selected elements on page

HTML without CSS



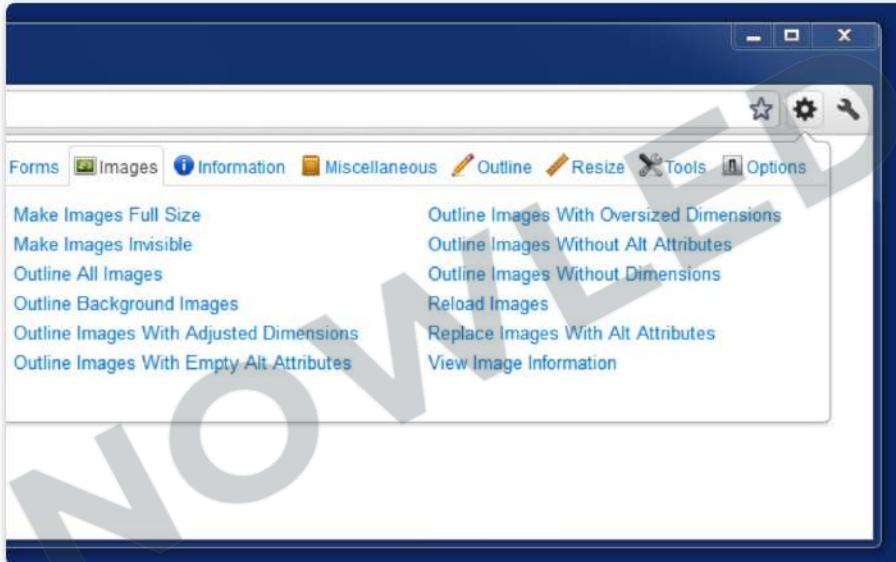
HTML without CSS

 **Web Developer**

 chrispederick.com 4.5 ★ (2.8K ratings)

Extension Developer Tools 1,000,000 users

Remove from Chrome



A screenshot of the Web Developer extension interface. It features a toolbar with tabs for Forms, Images, Information, Miscellaneous, Outline, Resize, Tools, and Options. Below the toolbar is a list of tools: Make Images Full Size, Make Images Invisible, Outline All Images, Outline Background Images, Outline Images With Adjusted Dimensions, Outline Images With Empty Alt Attributes, Outline Images With Oversized Dimensions, Outline Images Without Alt Attributes, Outline Images Without Dimensions, Reload Images, Replace Images With Alt Attributes, and View Image Information.

< >



KNOWLEDGE GATE

Browser Tools



KNOWLEDGE
CREATE

**Responsive
Design**

A large, stylized text graphic on the right side of the image. It features the words "KNOWLEDGE" and "CREATE" in a dark brown, sans-serif font, each enclosed in a brown chevron-shaped border. Overlaid on these words are the words "Responsive" and "Design" in a larger, bold font. "Responsive" is green and "Design" is orange, matching the colors of the devices in the collage.

Different Screen Sizes



1. Adapts layout for different screen sizes
2. Flexible layouts
3. Optimizes images and assets
4. Enhances user experience on mobile and desktop

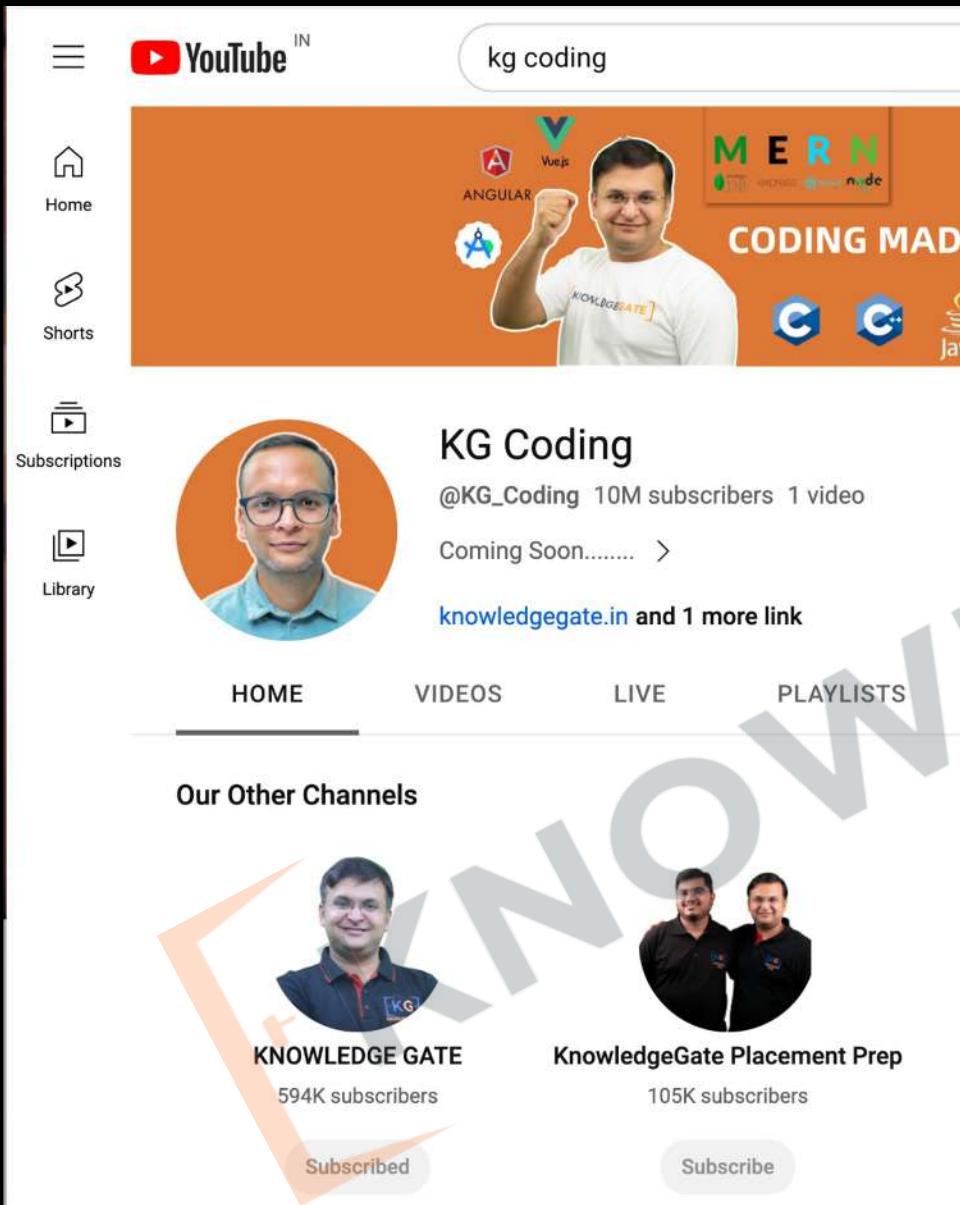
Browser Tools



Live Edit
Code

ADVANCED
ROBOTIC
SURGERY

Live edit HTML



```
> renderer>
  > <yt-channel-name id="channel-name" wrap-text class="style-scope ytd-c4-tabbed-header-renderer">...
```

```
</yt-channel-name>
  > <p class="style-scope ytd-c4-tabbed-header-renderer" hidden>...</p>
  > <span class="meta-item style-scope ytd-c4-tabbed-header-renderer">...</span>
  > <span class="meta-item style-scope ytd-c4-tabbed-header-renderer" hidden>...</span>
  > <span class="meta-item style-scope ytd-c4-tabbed-header-renderer">...
    <yt-formatted-string id="subscriber-count" class="style-scope ytd-c4-tabbed-header-renderer" aria-label="13.8K subscribers">10M subscribers</yt-formatted-string> == $0
    <span aria-hidden="true" class="delimiter style-scope ytd-c4-tabbed-header-renderer">·</span>
  </span>
  > <span class="meta-item style-scope ytd-c4-tabbed-header-renderer">...</span>
  > <div id="channel-tagline" class="style-scope ytd-c4-tabbed-header-renderer">...</div>
  > <div id="channel-header-links" class="style-scope ytd-c4-tabbed-header-renderer">...</div>
</div>
  > <div id="buttons" class="style-scope ytd-c4-tabbed-header-renderer">...</div> flex
</div>
</div>
<div id="links-holder" class="style-scope ytd-c4-tabbed-header-renderer">...</div> flex
</div>
<tp-yt-app-toolbar sticky class="style-scope ytd-c4-tabbed-header-renderer" style="transform: translate3d(0px, 0px, 0px);> ...</tp-yt-app-toolbar> flex
iv>
yt-app-header>
id="contentContainer" class="style-scope tp-yt-app-header-l" style="padding-top: 364px;"></div>

app-header-layout>
tabbed-header-renderer>

rts class="style-scope ytd-browse"></div>
legal-info-renderer class="style-scope ytd-browse" disable-en></yt-channel-legal-info-renderer>
t-sidebar-renderer class="style-scope ytd-browse" disable-en></yt-playlist-sidebar-renderer>
t-header-renderer class="style-scope ytd-browse" disable-en></yt-playlist-header-renderer>
s-sidebar-renderer class="style-scope ytd-browse" disable-en></yt-dispatches-sidebar-renderer>
um-browse-results-renderer class="style-scope ytd-browse g" columns="page-subtype="channels" style="touch-action: pan-y;">

ild:shady-->
ild:shady-->
primary class="style-scope ytd-two-column-browse-results-re
```

Changed Subscriber count

Live edit CSS

The screenshot shows the KG Coding YouTube channel page. At the top, the channel name 'kg coding' is visible in the search bar. Below it, the channel banner features a man in a white t-shirt with 'KNOWLEDGE GATE' on it, surrounded by various technology logos like Angular, Vue.js, MERN, Express, Node.js, C, C++, and Java. The channel title 'KG Coding' is displayed in large red letters. The channel stats show 10M subscribers and 1 video. A 'Coming Soon..... >' message is present. Below the stats, there are links to 'knowledgegate.in' and '1 more link'. Navigation tabs include HOME, VIDEOS, LIVE, and PLAYLISTS. A section titled 'Our Other Channels' lists 'KNOWLEDGE GATE' with 594K subscribers and a 'Subscribed' button, and 'KnowledgeGate Placement Prep' with 105K subscribers and a 'Subscribe' button.

```
<div id="channel-name" class="style-scope ytd-c4-tabbed-header-renderer">
<!---->
<!---->
<div id="container" class="style-scope ytd-channel-name">
<div id="text-container" class="style-scope ytd-channel-name">
<yt-formatted-string id="text" link-inherit-color title class="style-scope ytd-channel-name">KG Coding</yt-formatted-string> == $0
</div>
<tp-yt-paper-tooltip fit-to-visible-bounds class="style-scope ytd-channel-name" role="tooltip" tabindex="-1"></tp-yt-paper-tooltip>
</div>
<ytd-badge-supported-renderer class="style-scope ytd-channel-name" disable-upgrade hidden></ytd-badge-supported-renderer>
</ytd-channel-name>
<p class="style-scope ytd-c4-tabbed-header-renderer" hidden>...</p>
<span class="meta-item style-scope ytd-c4-tabbed-header-renderer"></span>
<span class="meta-item style-scope ytd-c4-tabbed-header-renderer" hidden></span>
<span class="meta-item style-scope ytd-c4-tabbed-header-renderer">
<yt-formatted-string id="subscriber-count" class="style-scope ytd-c4-tabbed-header-renderer" aria-label="13.8K subscribers">10M subscribers</yt-formatted-string>
<span aria-hidden="true" class="delimiter style-scope ytd-c4-tabbed-header-renderer"></span>
<span class="meta-item style-scope ytd-c4-tabbed-header-renderer"></span>
<div id="channel-tagline" class="style-scope ytd-c4-tabbed-header-renderer"></div>
<div id="channel-header-links" class="style-scope ytd-c4-tabbed-header-renderer"></div>
</div>
<div id="buttons" class="style-scope ytd-c4-tabbed-header-renderer"></div>
</div>
<div id="links-holder" class="style-scope ytd-c4-tabbed-header-renderer"></div>
</div>
<tp-yt-app-toolbar sticky class="style-scope ytd-c4-tabbed-header-renderer" style="transform: translate3d(0px, 0px, 0px); "></tp-yt-app-toolbar>
<div id="contentContainer" class="style-scope tp-yt-app-header-layout" style="padding-top: 364px;"></div>
</div>
<tp-yt-app-header-layout>
</ytd-c4-tabbed-header-renderer>
</div>
<div id="alerts" class="style-scope ytd-browse"></div>
<ytd-channel-legal-info-renderer class="style-scope ytd-browse" disable-upgrade hidden></ytd-channel-legal-info-renderer>
<ytd-playlist-sidebar-renderer class="style-scope ytd-browse" disable-upgrade hidden></ytd-playlist-sidebar-renderer>
```

Changed Channel Name color

Changes happening at Client

1. Changes made are **temporary**
2. Affect **only** the current session
3. **Not saved** to the server
4. Reset upon page **reload**
5. Useful for **testing**, not permanent fixes

Like: If you change the question in your question paper that has no effect on actual exam.



Browser Tools (CSS Specific element)

The screenshot shows the Chrome DevTools interface with the 'Elements' tab selected. In the top navigation bar, the 'Elements' tab is highlighted. Below the tabs, the DOM tree is visible, showing the structure of the page. The 'Styles' tab in the bottom navigation bar is circled in red. The main content area displays the CSS styles for the selected element, with two sections: 'external styling.css:1' and 'user agent stylesheet'. The 'external styling.css:1' section contains the rule `h1 { color: red; }`. The 'user agent stylesheet' section contains rules for the user agent, including margin, border, and padding properties. A large red arrow points from the 'Styles' tab in the DevTools to a detailed 'Box Model' diagram below. The 'Box Model' diagram illustrates the layout of an `h1` element with a red border and a green background. It shows the outermost box with a width of 249.200 and a height of 36.800, and inner boxes for margin, border, and padding.

```
<!DOCTYPE html>
<html lang="en">
  <head></head>
  <body>
    <h1>External Styling</h1> == $0
  </body>
</html>
```

element.style {

h1 {
 color: red;
}
h1 {
 display: block;
 font-size: 2em;
 margin-block-start: 0.67em;
 margin-block-end: 0.67em;
 margin-inline-start: 0px;
 margin-inline-end: 0px;
 font-weight: bold;
}

margin 21.440
border -
padding -
249.200x36.800 -
-
-
-
margin 21.440

- Styles Panel
- Box Model
- Changes happening only at client

Browser Tools (Source tab)



A screenshot of the Chrome DevTools Sources tab. The tab bar includes icons for Find, Select, Sources, Elements, Console, Network, and a settings gear. The Sources tab is selected. The left sidebar shows a file tree with 'Page' and 'Filesystem' sections, and a '127.0.0.1:5500' network entry containing 'css' and 'html' folders. The 'external styling.css' file is selected in the 'css' folder, and its content is displayed in the main pane:

```
1 h1 {  
2   color: red;  
3 }  
4  
5 p {  
6   color: green;  
7 }
```

The bottom right of the interface shows a 'Coverage: n/a' status.

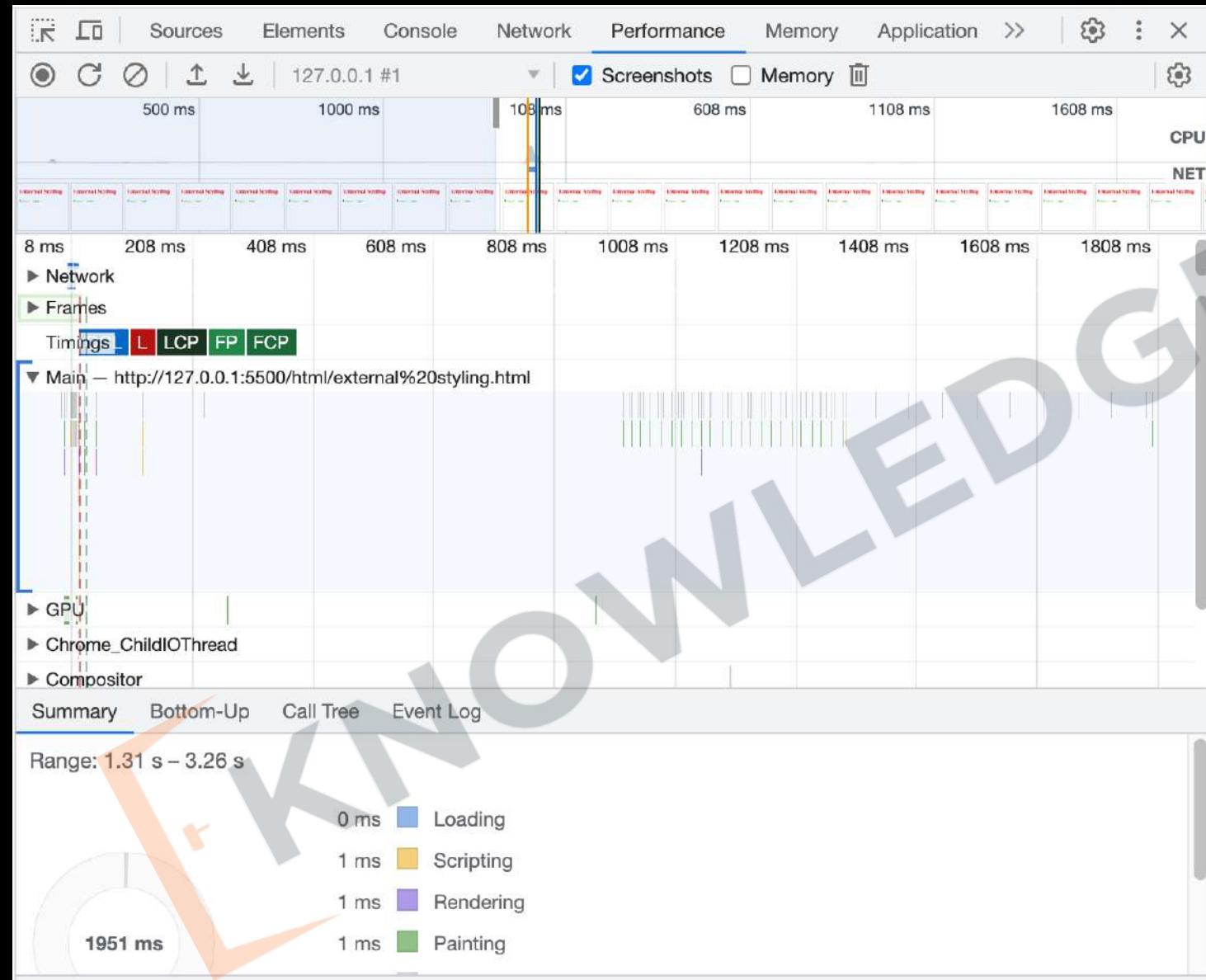
KNOWLEDGE AGATE

Browser Tools (Network tab)

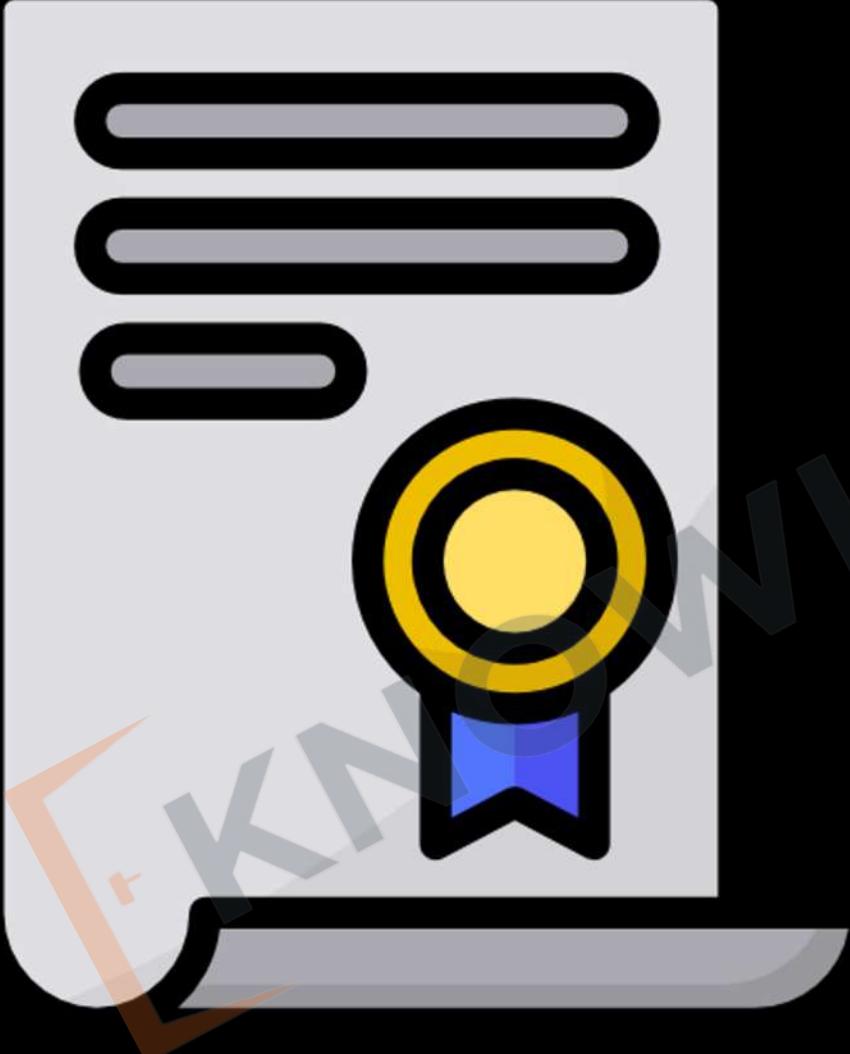
The screenshot shows the Network tab of a browser's developer tools. At the top, there are tabs for Sources, Elements, Console, Network (which is selected), Performance, Memory, and Application. Below the tabs are various controls: a red square icon, a circular icon with a slash, a magnifying glass icon, checkboxes for 'Preserve log' and 'Disable cache', a dropdown for 'No throttling' with options like 'Mobile 3G', 'Mobile 4G', 'Laptop', 'Tablet', and 'Smartphone', and download/upload icons. A large watermark 'KNOWLEDGE' is diagonally across the interface.

Name	Status	Type	Initiator	Size	Time	Waterfall
external%20styling.html	304	docu...	Other	295 B	4 ms	
ws	101	webso...	external styling...	0 B	Pending	

Browser Tools (Performance tab)



Browser Tools

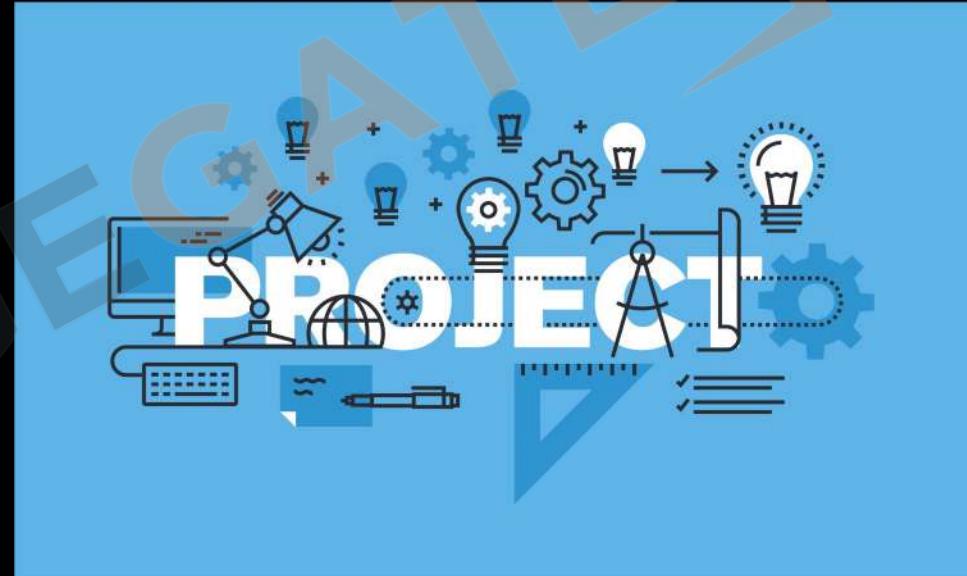


Validating
Web Pages
using
ChatGPT

Practice Set

Browser Tools

1. Save Source of **Instagram** in a file and check the render.
2. **Inspect** the likes element on the page and read the code to understand.
3. Change number of likes on Your **Instagram** post
4. **Validate** the page we created in last project.



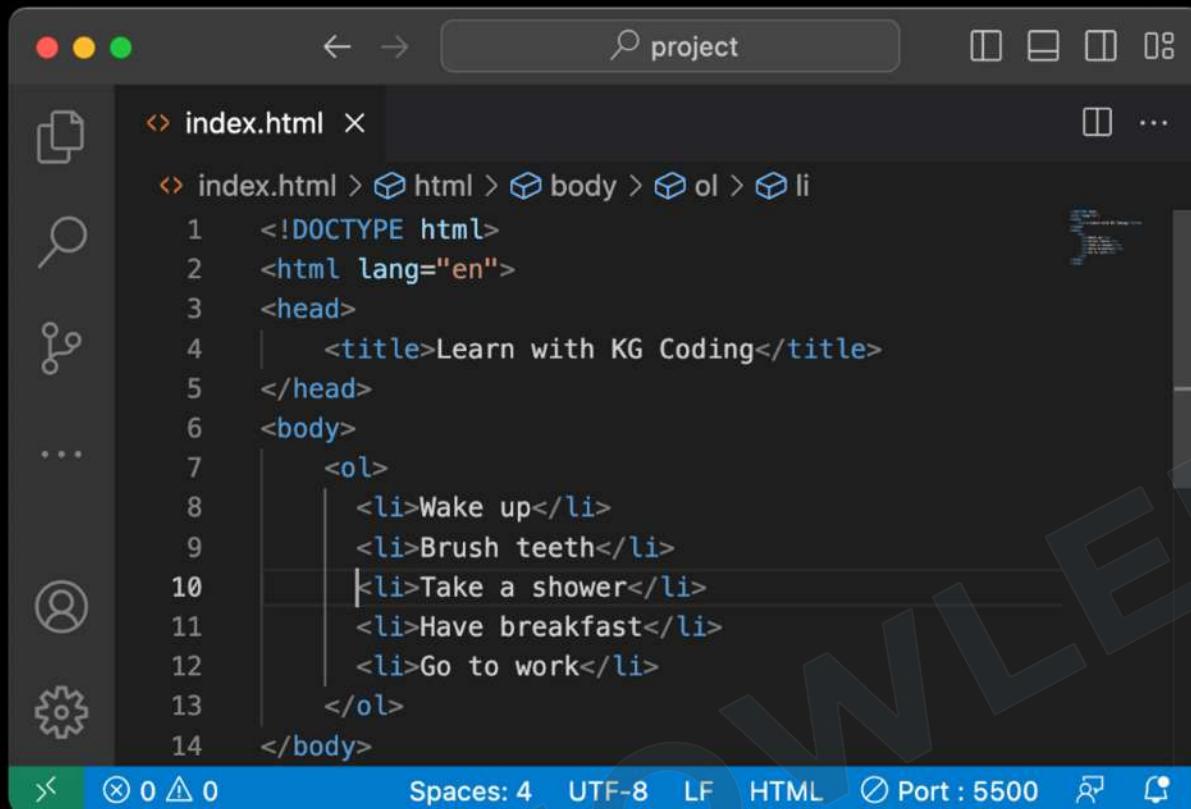


List, Tables & Forms



LIST Tag

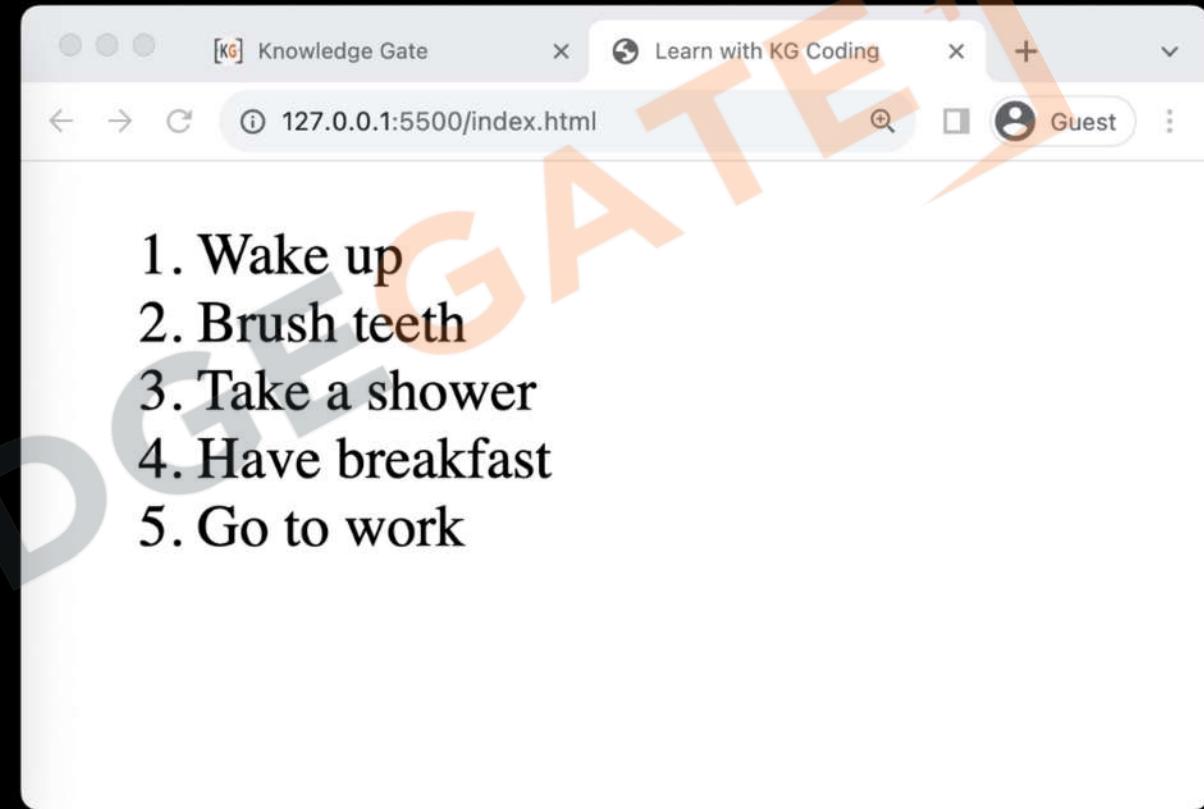
Ordered Lists



A screenshot of a code editor window titled "project". The file "index.html" is open, showing the following code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <ol>
        <li>Wake up</li>
        <li>Brush teeth</li>
        <li>Take a shower</li>
        <li>Have breakfast</li>
        <li>Go to work</li>
    </ol>
</body>
```

The code editor interface includes a sidebar with icons for file, search, and user, and a bottom bar with status indicators.



1. **Purpose:** Used for creating lists with items that have a specific order.
2. **Default:** Items are automatically numbered.
3. **Nesting:** Can be nested within other lists.
4. **Reversed** attribute can be used to show the numbering in reversed.

Types of Ordered Lists

Ordered Lists

- **Numeric:** Default type, (1, 2, 3, ...)
Attribute: `type="1"`
- **Uppercase Letters:** (A, B, C, ...)
Attribute: `type="A"`
- **Lowercase Letters:** (a, b, c, ...)
Attribute: `type="a"`
- **Uppercase Roman:** (I, II, III, ...)
Attribute: `type="I"`
- **Lowercase Roman:** (i, ii, iii, ...)
Attribute: `type="i"`

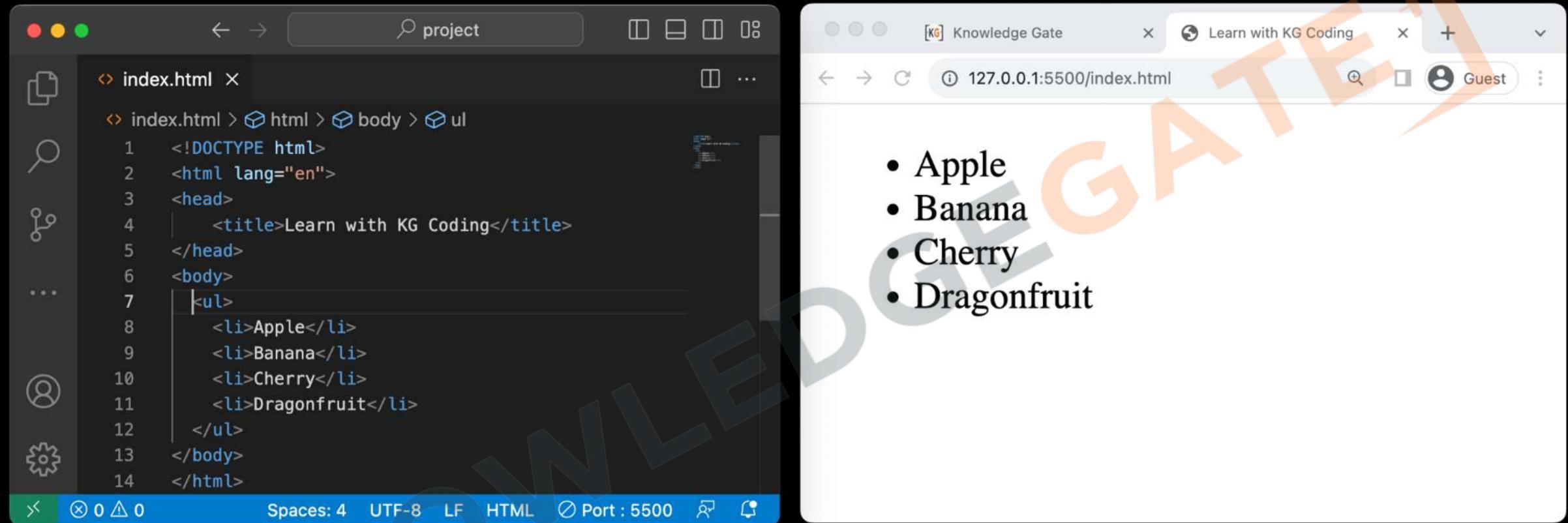
A. Apple
B. Banana
C. Cherry
D. Dragonfruit

a. Apple
b. Banana
c. Cherry
d. Dragonfruit

I. Apple
II. Banana
III. Cherry
IV. Dragonfruit

i. Apple
ii. Banana
iii. Cherry
iv. Dragonfruit

Unordered Lists



The image shows a code editor on the left and a web browser on the right. The code editor displays the following HTML code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <ul>
        <li>Apple</li>
        <li>Banana</li>
        <li>Cherry</li>
        <li>Dragonfruit</li>
    </ul>
</body>
</html>
```

The browser window shows the rendered output of the code:

- Apple
- Banana
- Cherry
- Dragonfruit

1. **Purpose:** Used for lists where the order of items doesn't matter.
2. **Default:** Items are usually bulleted.
3. **Nesting:** Can be nested within other lists.

Types of Unordered Lists

```
<ul type="disc">
    <li>Disc item 1</li>
    <li>Disc item 2</li>
</ul>
```

```
<ul type="circle">
    <li>Circle item 1</li>
    <li>Circle item 2</li>
</ul>
```

```
<ul type="square">
    <li>Square item 1</li>
    <li>Square item 2</li>
</ul>
```

- Disc item 1
- Disc item 2
- Circle item 1
- Circle item 2
- Square item 1
- Square item 2

List, Tables & Forms



Table Tag

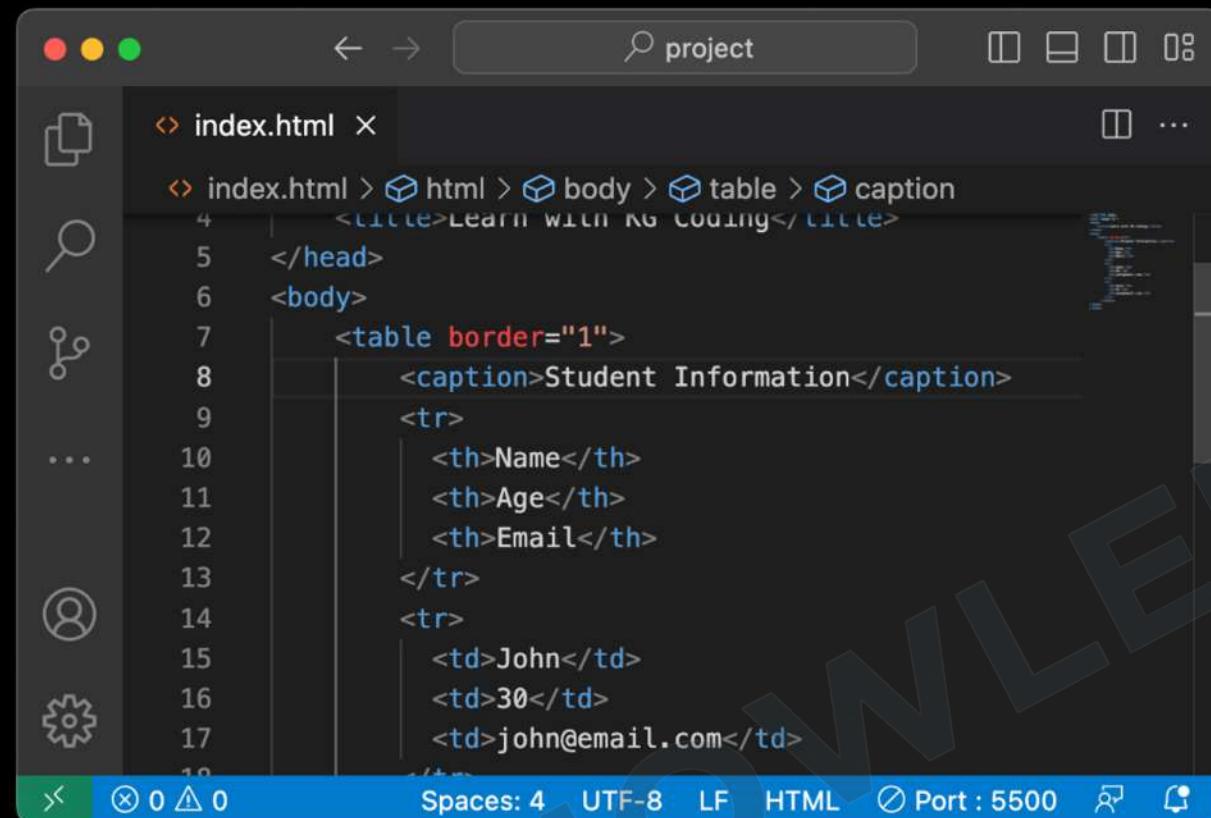
<tr>, <td>, <th> Tags

```
<!--> project <--> □ □ □ 0%  
index.html X  
index.html > html > body > table  
4 | <title>Learn with RG Coding</title>  
5 | </head>  
6 | <body>  
7 |   <table border="1">  
8 |     <tr>  
9 |       <th>Name</th>  
10 |       <th>Age</th>  
11 |       <th>Email</th>  
12 |     </tr>  
13 |     <tr>  
14 |       <td>John</td>  
15 |       <td>30</td>  
16 |       <td>john@email.com</td>  
17 |     </tr>  
18 |</table>  
...  
Spaces: 4 UTF-8 LF HTML Port: 5500 ✖ ⊗ 0 △ 0
```

Name	Age	Email
John	30	john@email.com
Jane	25	jane@email.com

1. **<tr>** Table Row : Used to define a row in an HTML table.
 2. **<th>** Table Header : Used for header cells within a row. Text is bold and centered by default.
 3. **<td>** Table Data : This Holds the actual data.
 4. Adding border="1" or simply border adds a border
 5. The recommended approach is to use CSS to add borders.

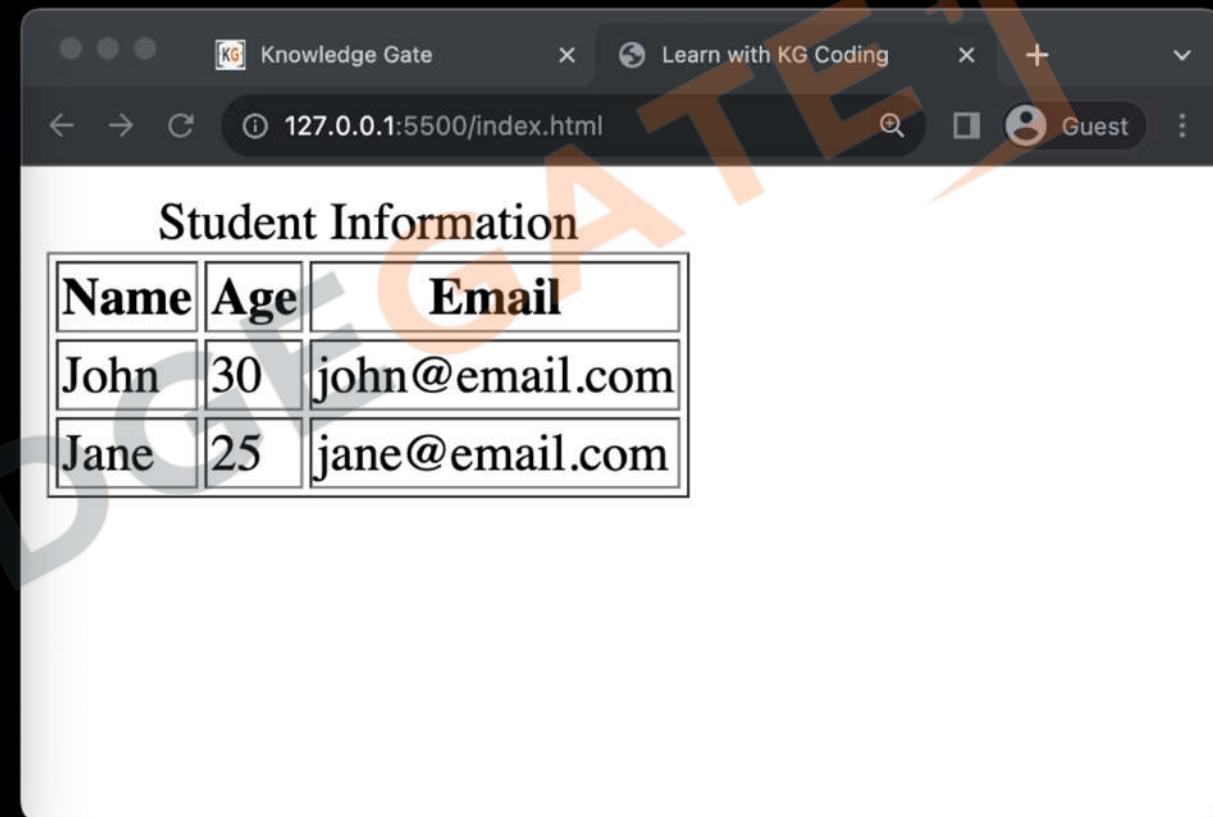
Captions



A screenshot of a code editor window titled "project". The file "index.html" is open. The code displays a table with a caption:

```
<caption>Student Information</caption>
<table border="1">
  <thead>
    <tr>
      <th>Name</th>
      <th>Age</th>
      <th>Email</th>
    </tr>
  <tbody>
    <tr>
      <td>John</td>
      <td>30</td>
      <td>john@email.com</td>
    </tr>
  </tbody>
</table>
```

The code editor interface includes a sidebar with icons for file operations, a search bar, and a status bar at the bottom indicating "Spaces: 4", "UTF-8", "LF", "HTML", "Port : 5500", and a refresh icon.



A screenshot of a web browser window titled "Knowledge Gate" showing the URL "127.0.0.1:5500/index.html". The page title is "Student Information". A table is displayed with the following data:

Name	Age	Email
John	30	john@email.com
Jane	25	jane@email.com

1. **Purpose:** Provides a title or description for a table.
2. **Placement:** Must be inserted **immediately after** the `<table>` opening tag.
3. **Alignment:** **Centered** above the table by default.
4. **Accessibility:** Helps screen readers understand the table's purpose.

Col Spans

The image shows a code editor on the left and a browser window on the right. The code editor displays the following HTML code:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html>
    <head>
        <title>Student Information</title>
    </head>
    <body>
        <table border="1">
            <caption>Student Information</caption>
            <thead>
                <tr>
                    <th>Name</th>
                    <th>Age</th>
                    <th>Email</th>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <td>John</td>
                    <td>30</td>
                    <td>john@email.com</td>
                </tr>
                <tr>
                    <td colspan="3">This cell spans 3 columns</td>
                </tr>
            </tbody>
        </table>
    </body>
</html>
```

The browser window shows the rendered table with the following data:

Name	Age	Email
John	30	john@email.com
This cell spans 3 columns		

1. **Attribute:** Uses the `colspan` attribute in `<td>` or `<th>` tags.
2. **Purpose:** Allows a cell to **span multiple columns** horizontally.
3. **Alignment:** Takes the space of the **specified number of columns**.
4. **Layout:** Useful for combining cells to create complex table layouts.

Col Group

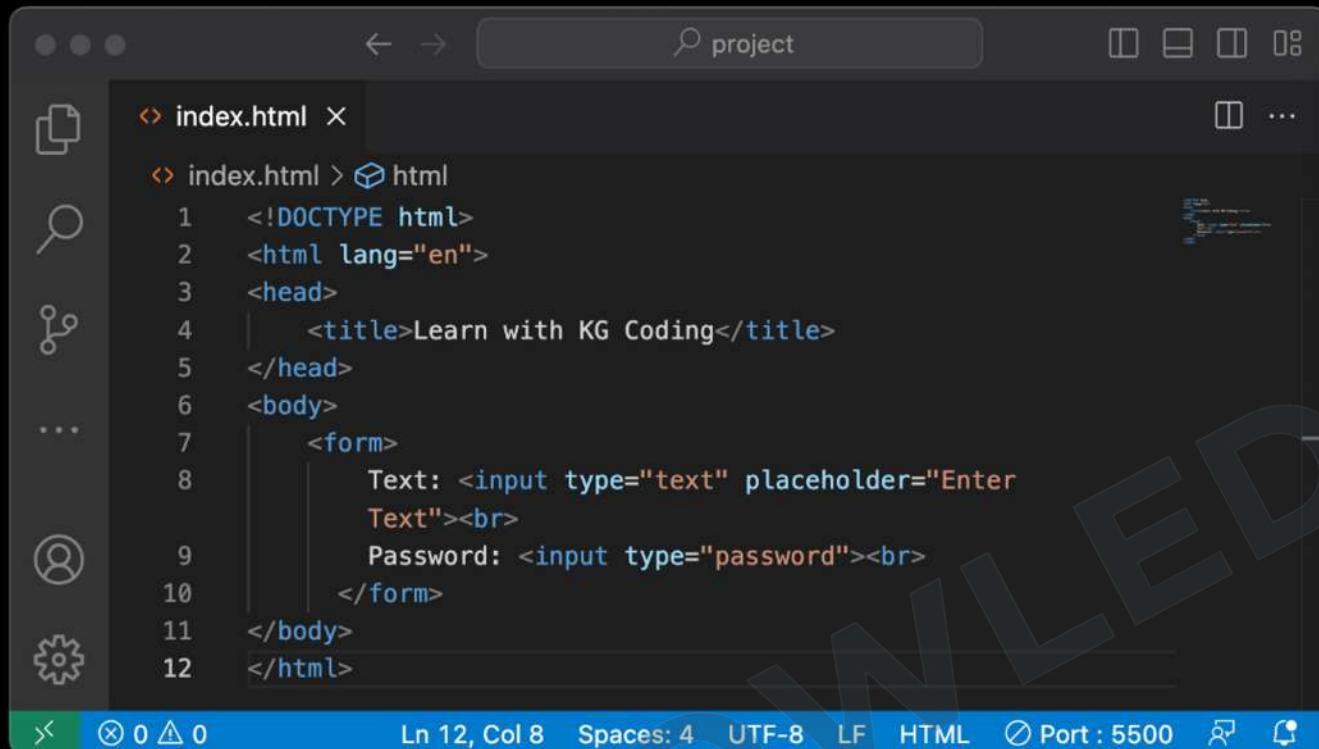
1. Attribute:

List, Tables & Forms



Forms

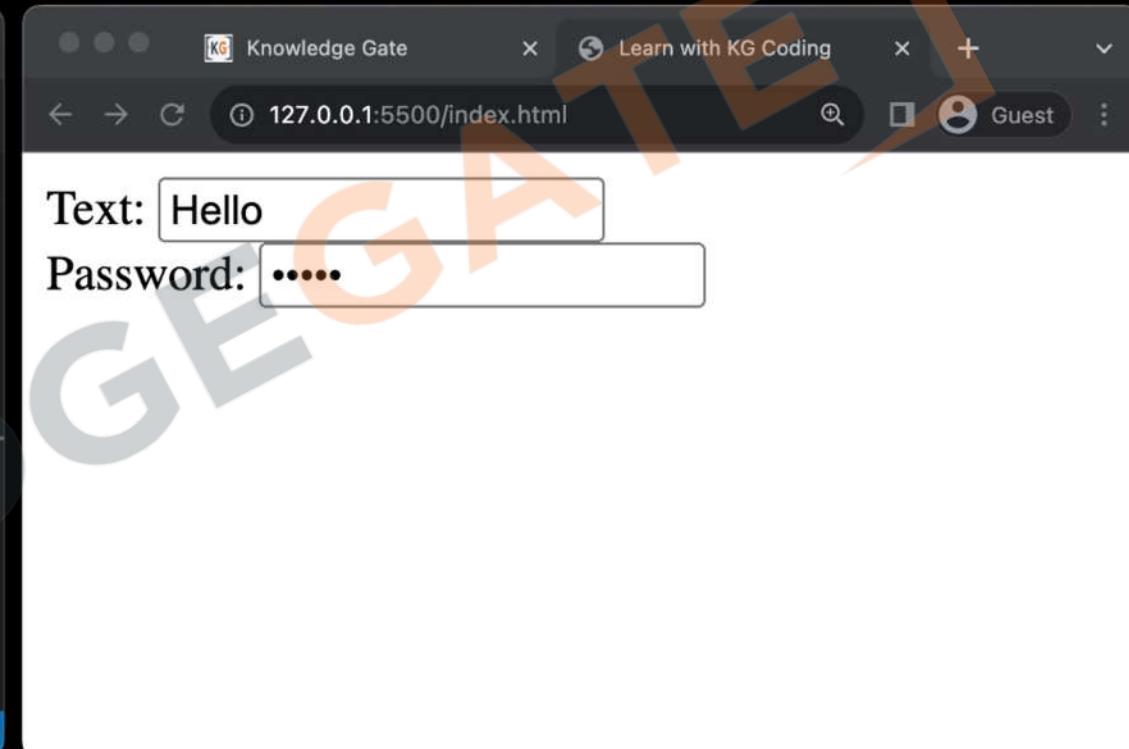
Input Tag



A screenshot of a code editor window titled "index.html". The code is as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <form>
        Text: <input type="text" placeholder="Enter Text"><br>
        Password: <input type="password"><br>
    </form>
</body>
</html>
```

The code editor has a dark theme with light-colored syntax highlighting. The status bar at the bottom shows "Ln 12, Col 8" and "Port : 5500".



A screenshot of a web browser window titled "Knowledge Gate". The URL is "127.0.0.1:5500/index.html". The page displays two input fields:

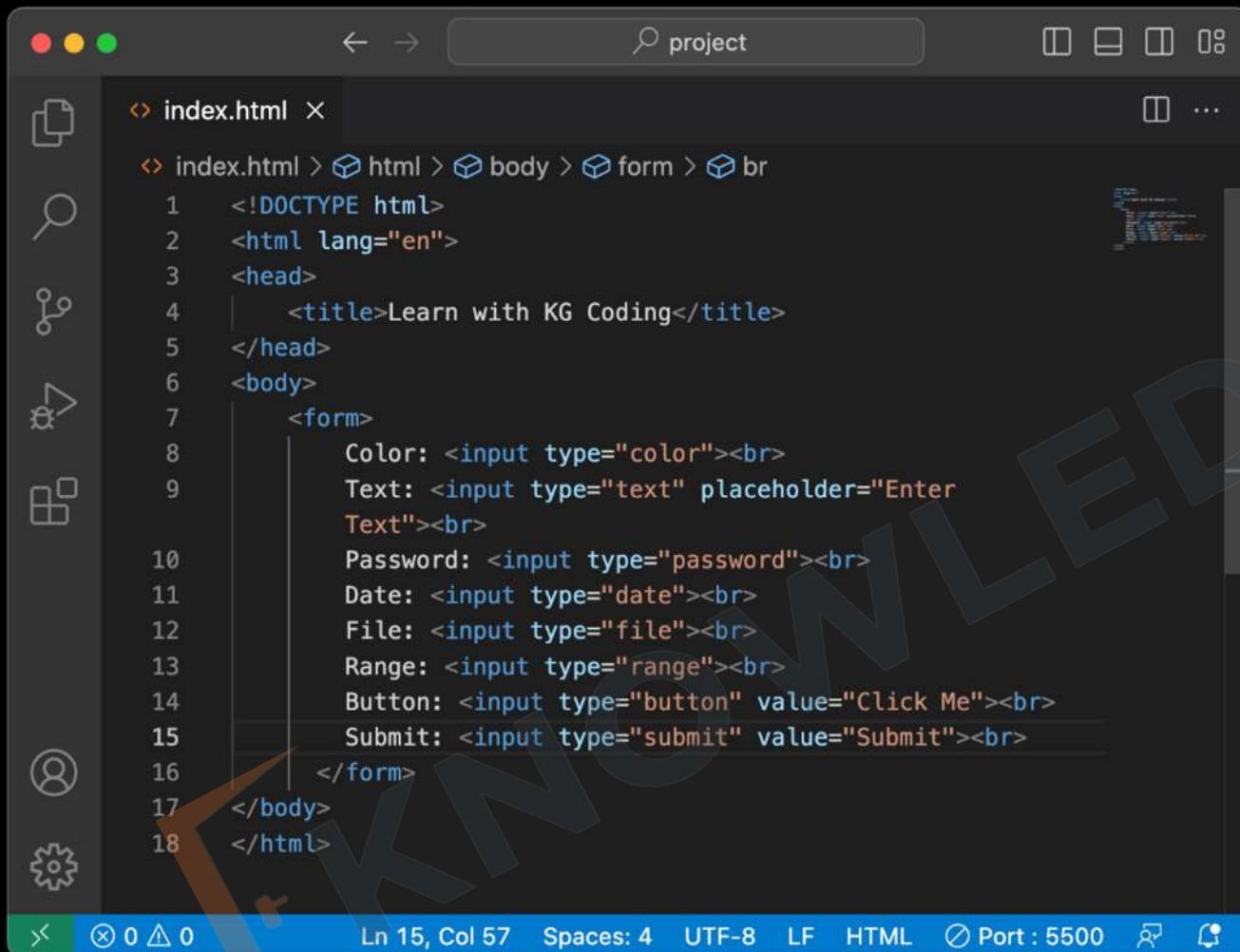
Text:

Password:

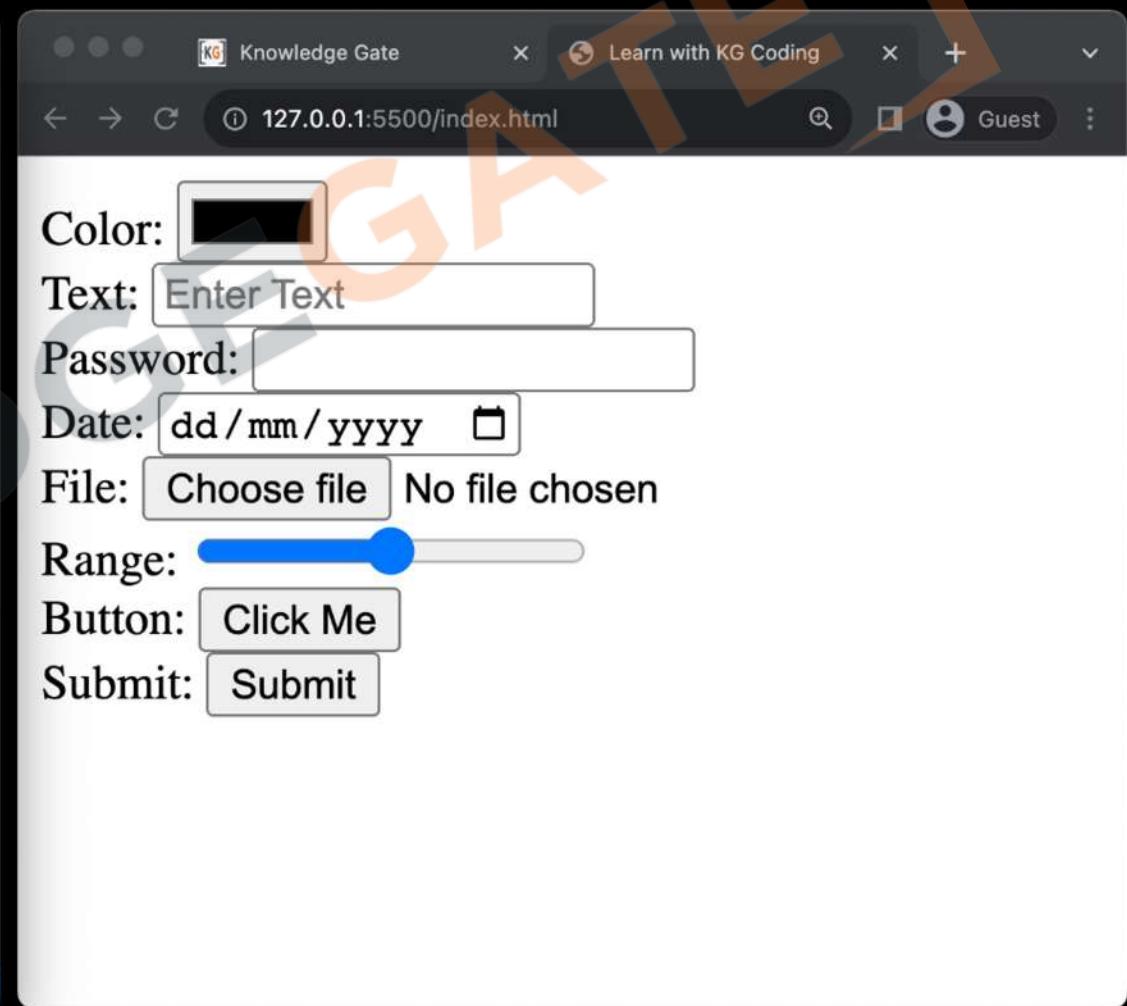
The browser interface includes a search bar, a guest user icon, and a menu icon.

1. **Purpose:** Used within a `<form>` element to collect user input.
2. **Self-Closing:** The `<input>` tag is self-closing; doesn't require a closing tag.
3. **Attributes:** Common attributes are `name`, `value`, `placeholder`, and `required`.

Input type: Submit



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <form>
        Color: <input type="color"><br>
        Text: <input type="text" placeholder="Enter Text"><br>
        Password: <input type="password"><br>
        Date: <input type="date"><br>
        File: <input type="file"><br>
        Range: <input type="range"><br>
        Button: <input type="button" value="Click Me"><br>
        Submit: <input type="submit" value="Submit"><br>
    </form>
</body>
</html>
```



Color:

Text:

Password:

Date: dd / mm / yyyy

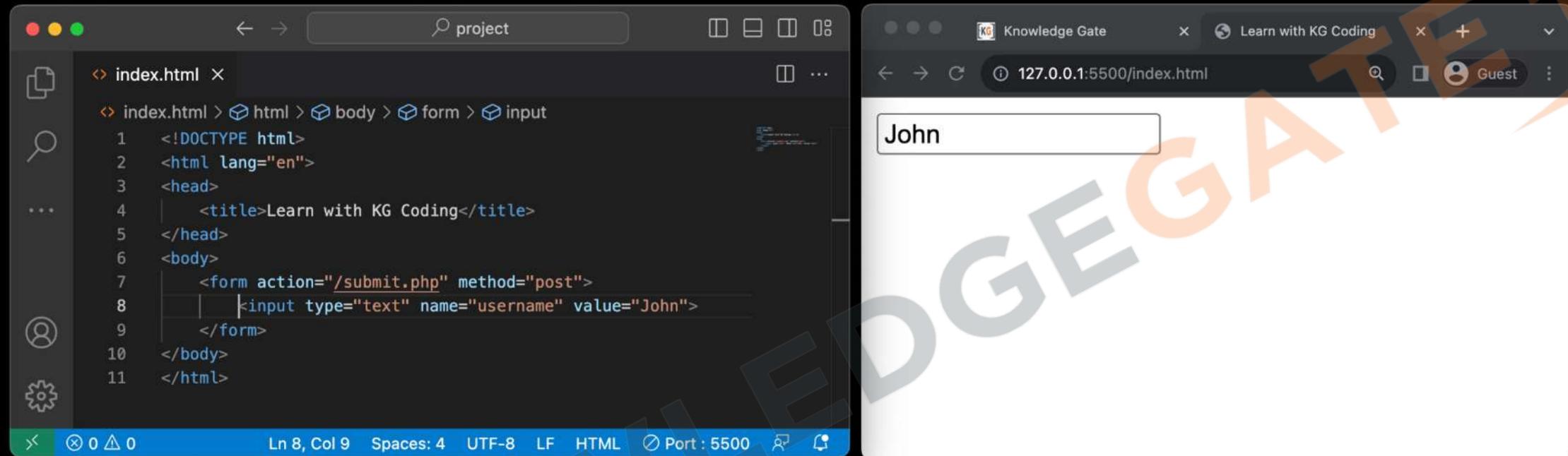
File: Choose file No file chosen

Range:

Button:

Submit:

Name and Value property



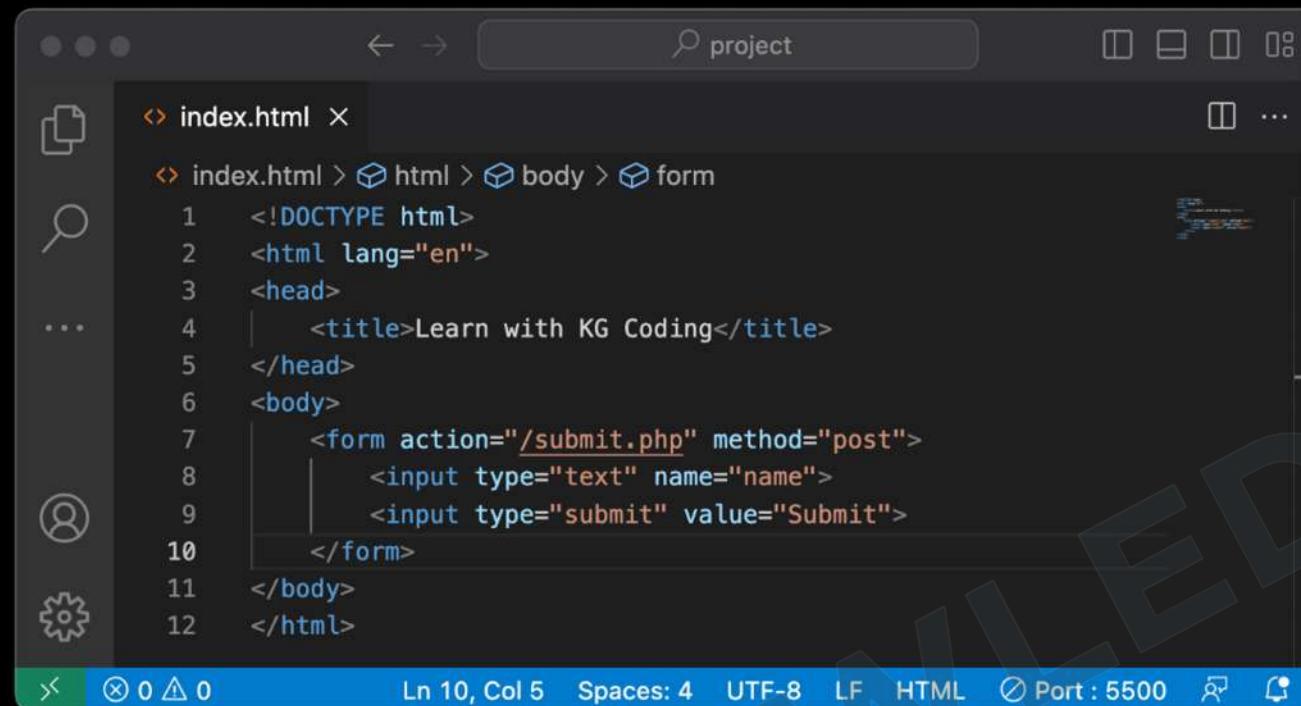
The image shows a code editor window on the left and a browser window on the right. The code editor displays the file 'index.html' with the following content:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <form action="/submit.php" method="post">
        <input type="text" name="username" value="John">
    </form>
</body>
</html>
```

The browser window shows the URL '127.0.0.1:5500/index.html'. Inside the browser, there is a single input field containing the text 'John'.

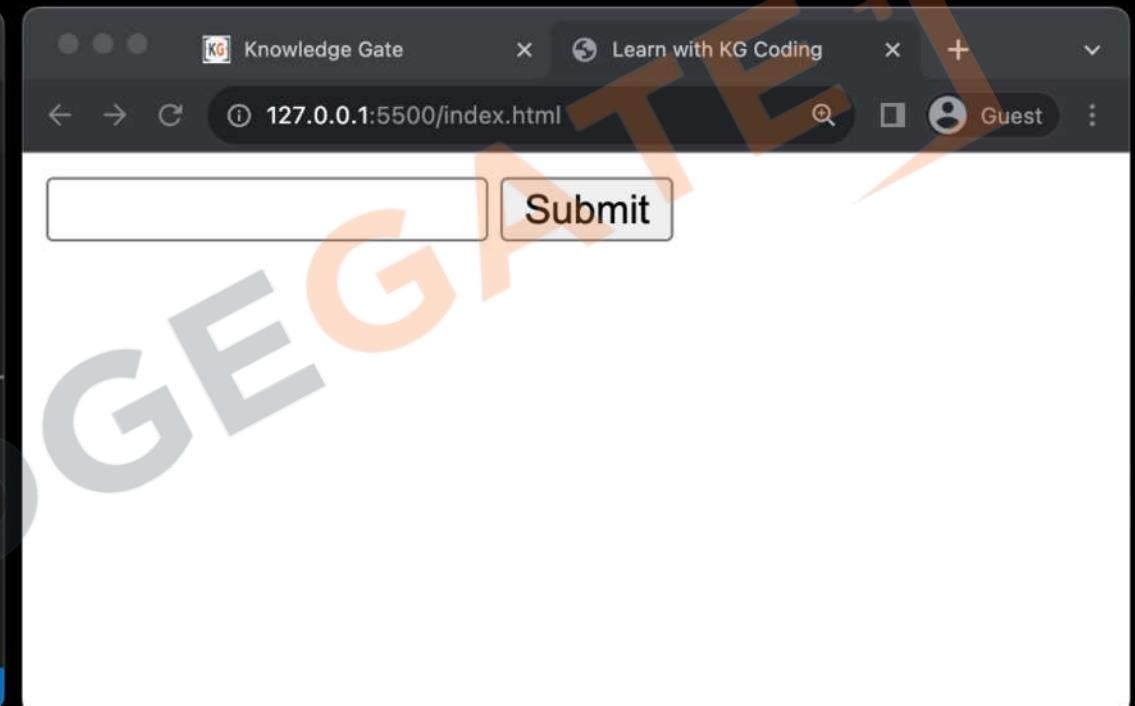
- `name` Property:
 - **ID for Data:** Identifies form elements when submitting.
 - **Unique:** Should be unique to each element for clarity.
- `value` Property:
 - **Default Data:** Sets initial value for input elements.
 - **Sent to Server:** This is the data sent when form is submitted.

Action attribute



A screenshot of a code editor showing the file `index.html`. The code defines a simple HTML page with a title and a form. The form has an `action` attribute pointing to `/submit.php`, and a `method` attribute set to `post`. The code editor interface includes a sidebar with icons for file operations, a search bar at the top, and various status indicators at the bottom.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <form action="/submit.php" method="post">
        <input type="text" name="name">
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```



1. **Purpose:** Specifies the URL to which the form data should be sent when submitted.
2. **Default:** If not specified, the form will be submitted to the **current page's URL**.
3. **Server-Side:** Usually points to a server-side script (like **PHP**, **Python**, etc.) that processes the form data.

Label Tag

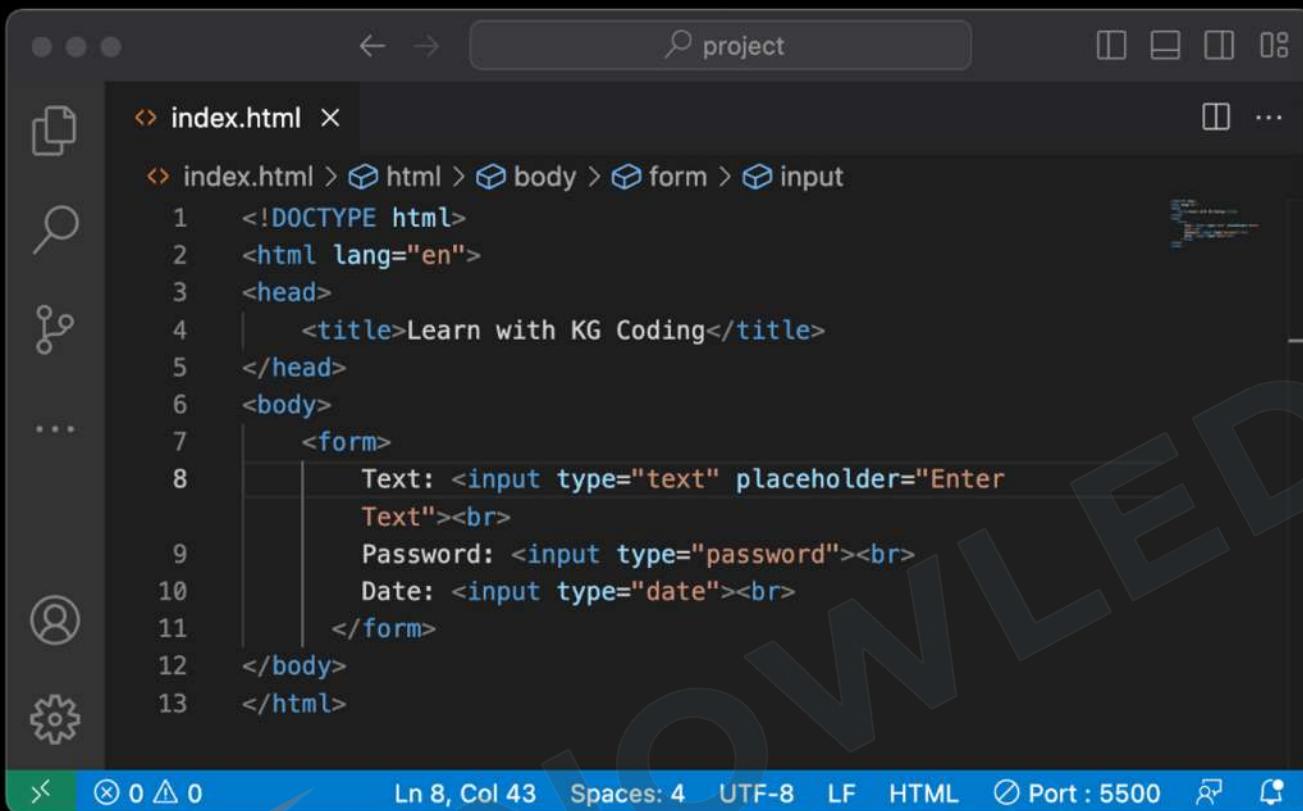
The image shows a code editor on the left and a web browser on the right. The code editor displays the file 'index.html' with the following content:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <form action="/submit.php" method="post">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username">
    </form>
</body>
</html>
```

The browser window shows the rendered HTML with the text 'Username:' followed by an empty text input field.

- **Purpose:** Adds a text description to form elements.
- **for Attribute:** Connects the label to a specific **form element** using the element's id.
- **Accessibility:** Makes the form more accessible.
- **Readability:** Enhances form readability and usability.

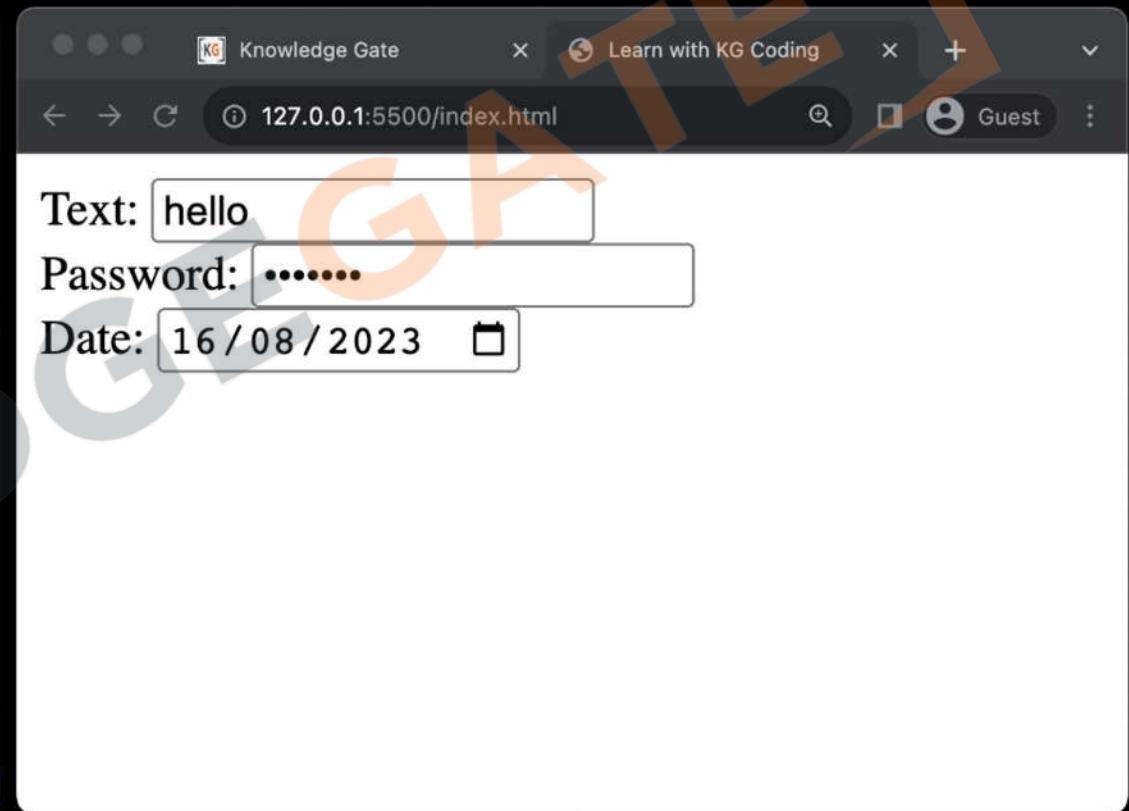
Input type: Date



A screenshot of a code editor window titled "index.html". The code displays a simple HTML form with three fields: a text input, a password input, and a date input. The date input is highlighted with a red box.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <form>
        Text: <input type="text" placeholder="Enter Text"><br>
        Password: <input type="password"><br>
        Date: <input type="date"><br>
    </form>
</body>
</html>
```

The status bar at the bottom shows the following information: Line 8, Column 43, Spaces: 4, UTF-8, LF, HTML, Port: 5500.



A screenshot of a web browser window titled "Knowledge Gate". The URL is "127.0.0.1:5500/index.html". The page contains a form with three fields: "Text" (value: "hello"), "Password" (value: "*****"), and "Date" (value: "16/08/2023"). The date input field has a small calendar icon to its right.

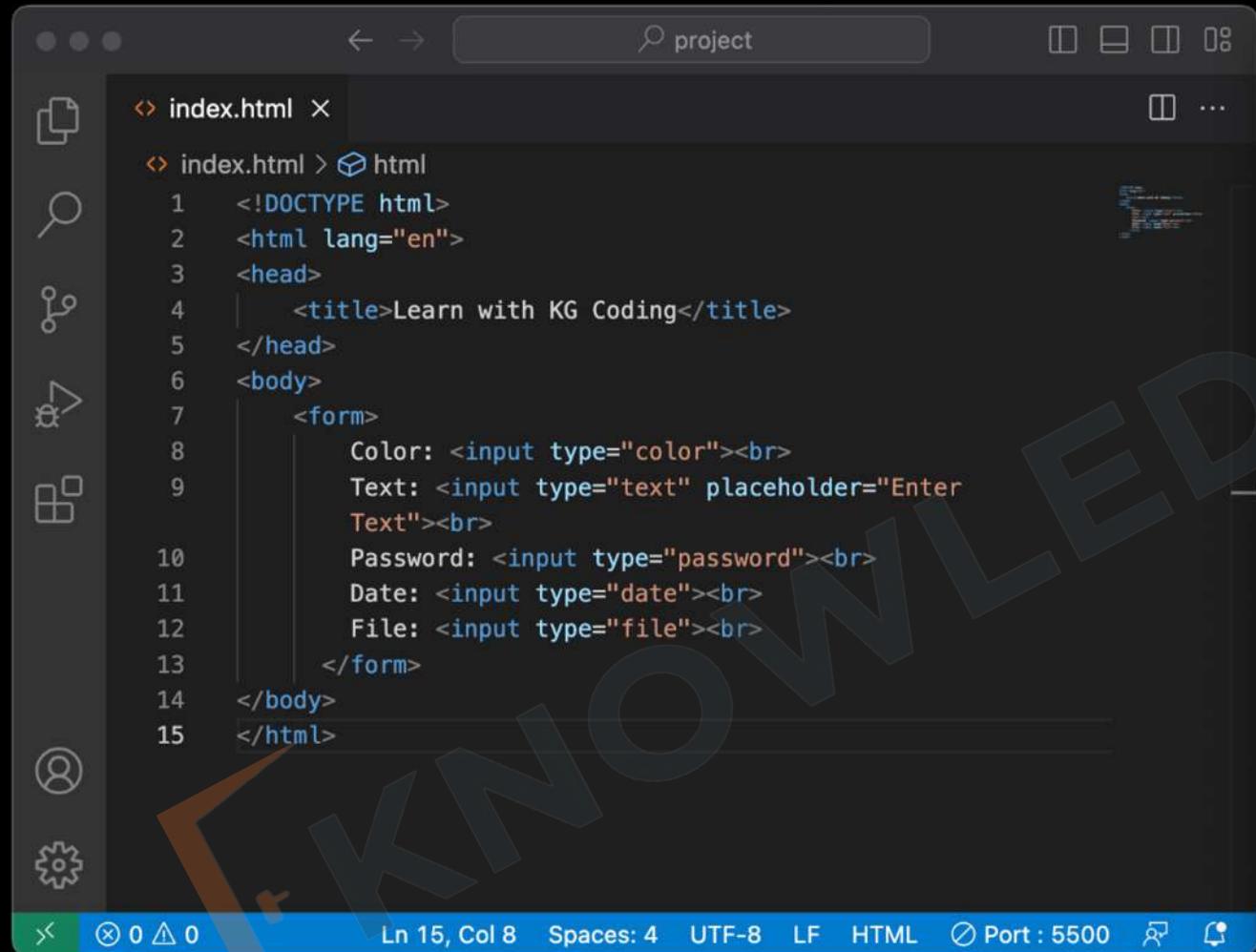
Input type: File

The image shows a code editor on the left and a web browser on the right. The code editor displays the contents of index.html, which includes a form with four input fields: text, password, date, and file.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <form>
        Text: <input type="text" placeholder="Enter Text"><br>
        Password: <input type="password"><br>
        Date: <input type="date"><br>
        File: <input type="file"><br>
    </form>
</body>
</html>
```

The browser window shows the rendered HTML. It has four labels: "Text:", "Password:", "Date:", and "File:". Next to each label is an input field. The "Text:" field contains "Enter Text". The "Password:" field is empty. The "Date:" field contains "dd/mm/yyyy". The "File:" field has a button labeled "Choose file" and the text "No file chosen".

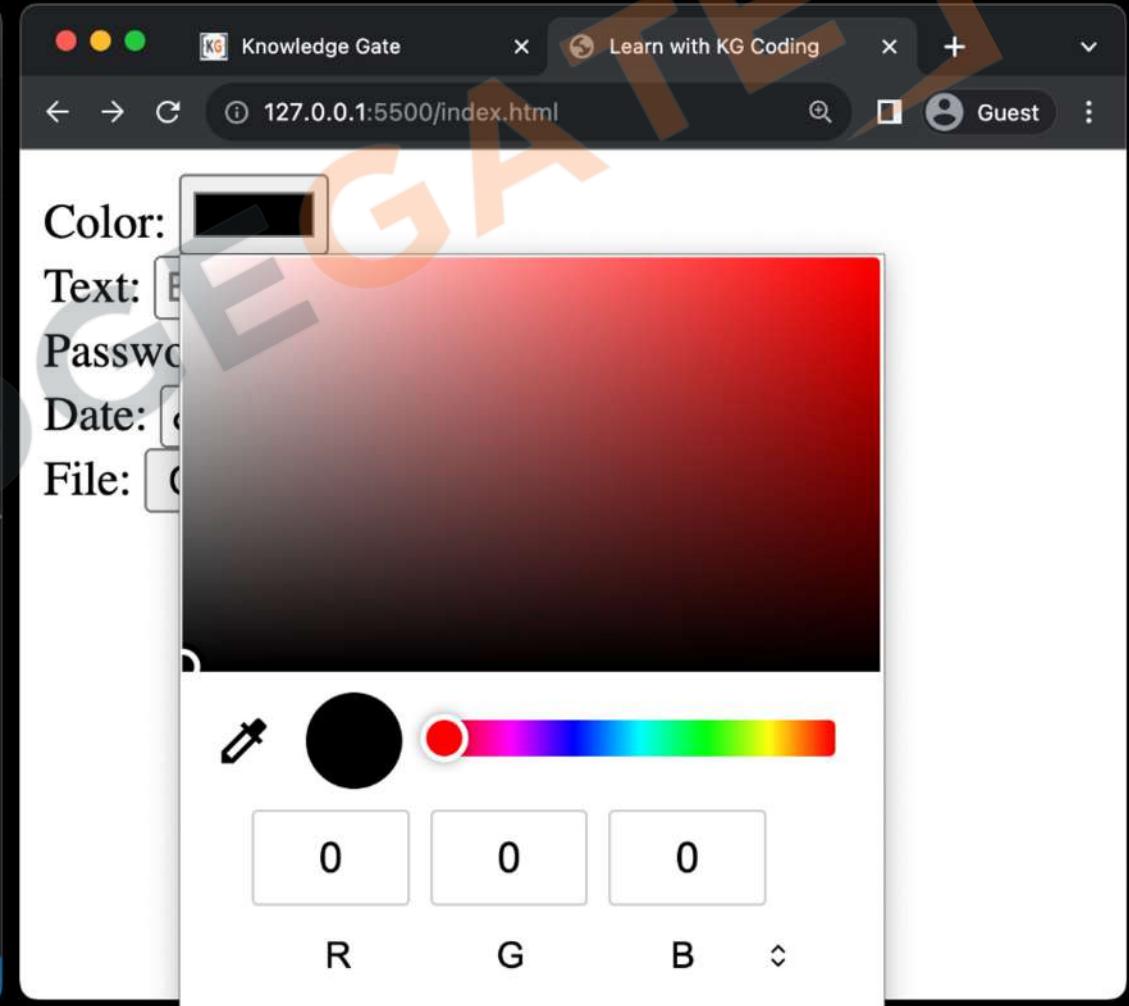
Input type: Color



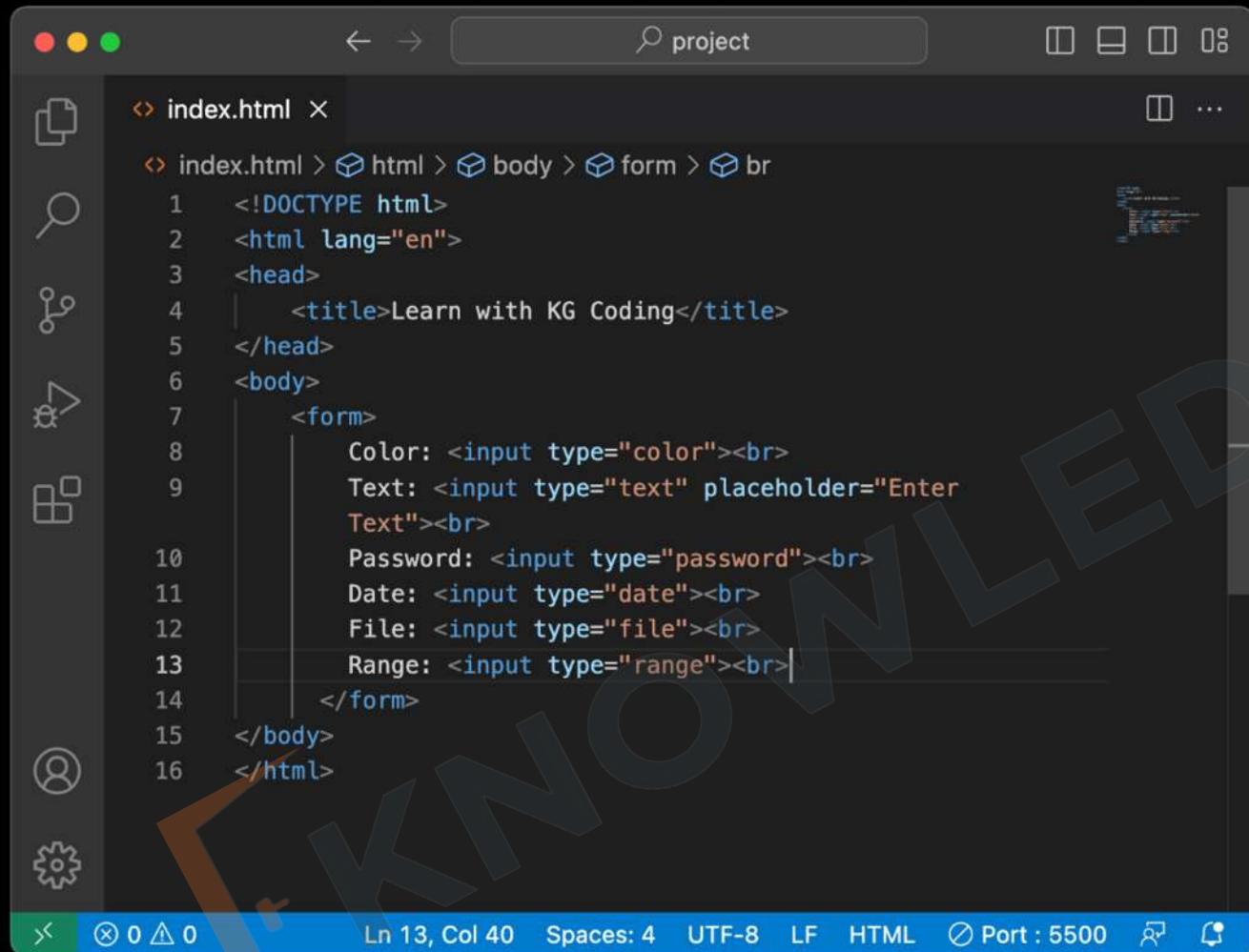
A screenshot of a code editor window titled "index.html". The code editor shows the following HTML structure:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <form>
        Color: <input type="color"><br>
        Text: <input type="text" placeholder="Enter Text"><br>
        Password: <input type="password"><br>
        Date: <input type="date"><br>
        File: <input type="file"><br>
    </form>
</body>
</html>
```

The status bar at the bottom indicates the current line is "Ln 15, Col 8" and the port is "Port : 5500".



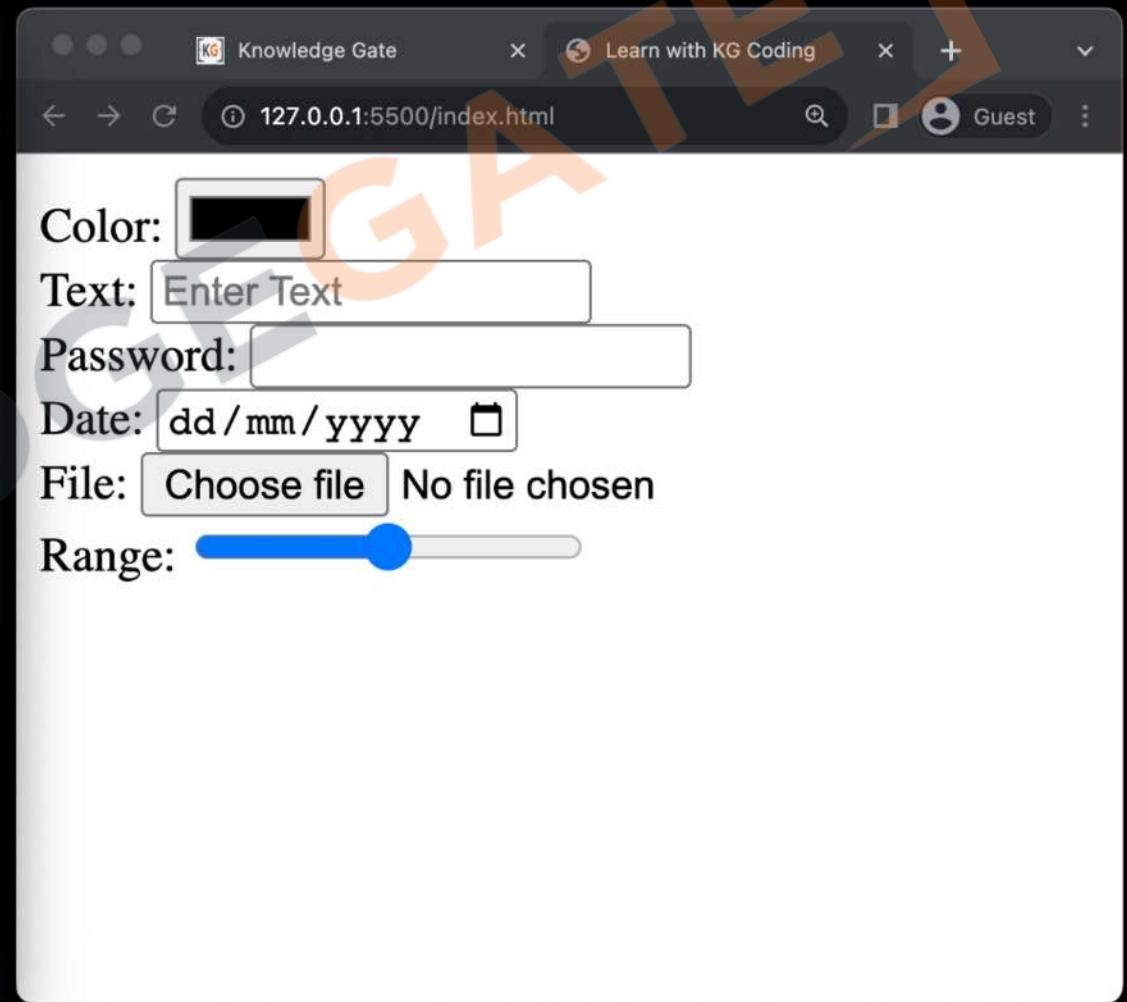
Input type: Range



A screenshot of a code editor window titled "index.html". The code displays a simple HTML form with several input fields:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <form>
        Color: <input type="color"><br>
        Text: <input type="text" placeholder="Enter Text"><br>
        Password: <input type="password"><br>
        Date: <input type="date"><br>
        File: <input type="file"><br>
        Range: <input type="range"><br>
    </form>
</body>
</html>
```

The status bar at the bottom shows the following information: Line 13, Column 40, Spaces: 4, UTF-8, LF, HTML, Port: 5500.



A screenshot of a web browser window titled "Knowledge Gate" showing the rendered HTML from the code editor. The page contains the following form elements:

Color:

Text:

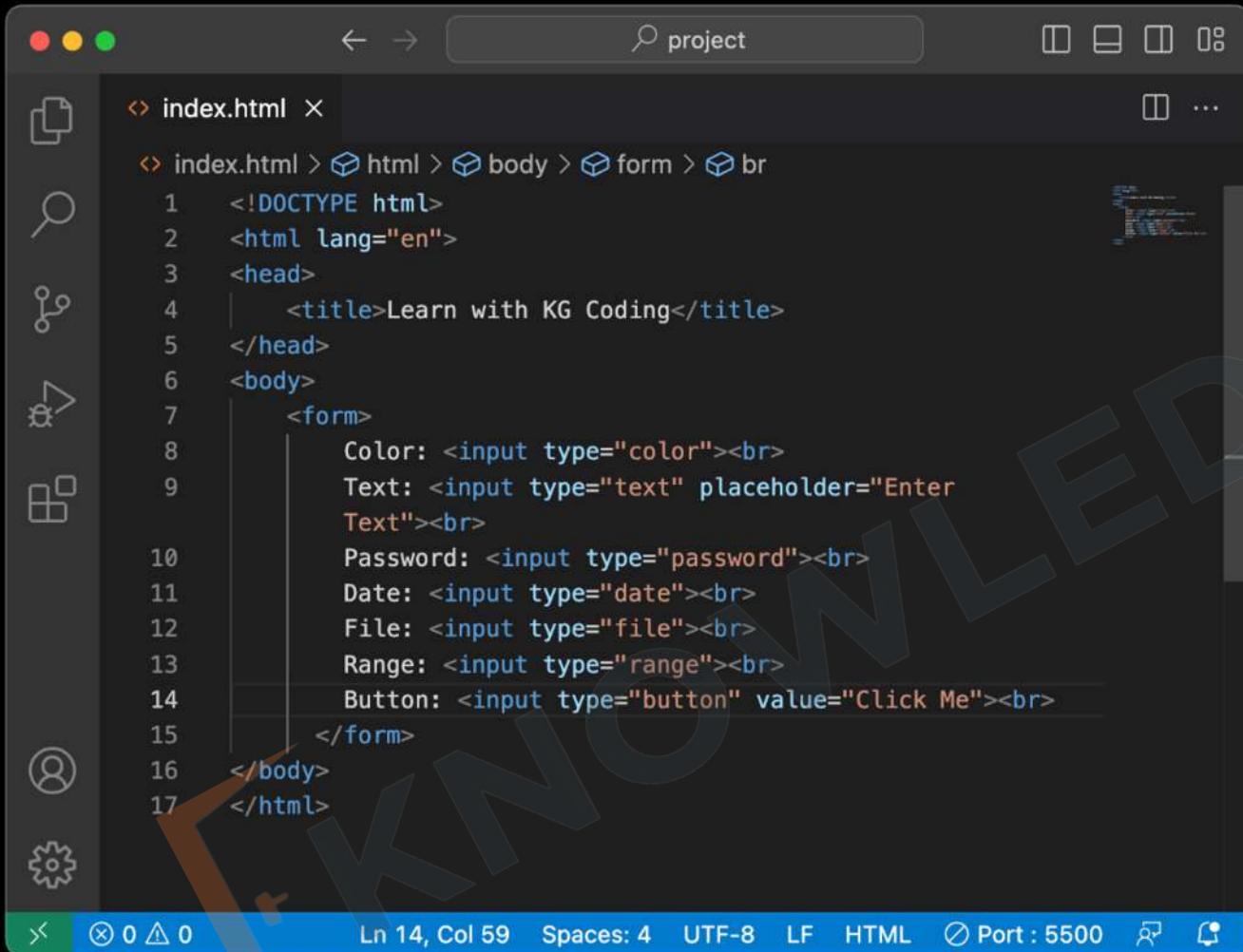
Password:

Date:

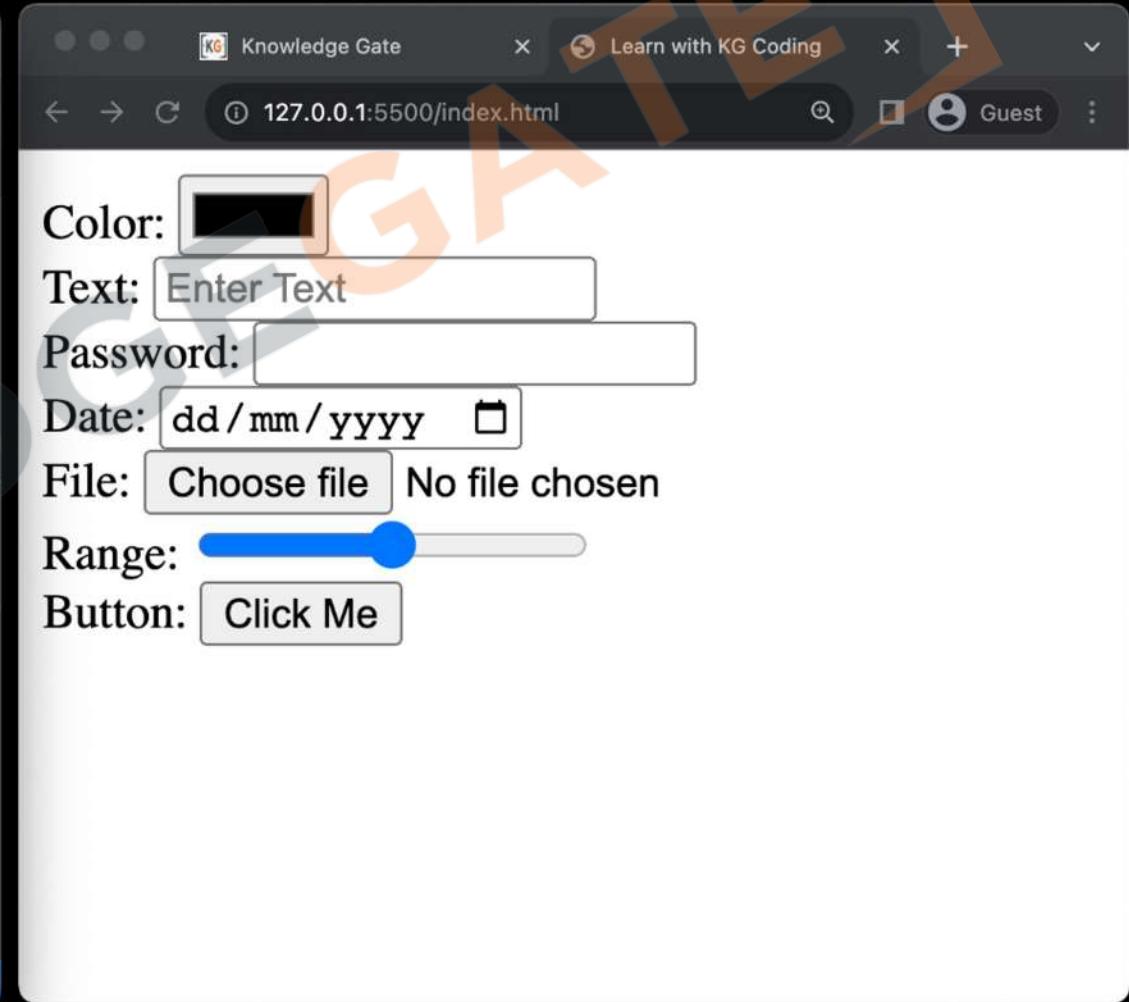
File: Choose file No file chosen

Range:

Input type: Button



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <form>
        Color: <input type="color"><br>
        Text: <input type="text" placeholder="Enter Text"><br>
        Password: <input type="password"><br>
        Date: <input type="date"><br>
        File: <input type="file"><br>
        Range: <input type="range"><br>
        Button: <input type="button" value="Click Me"><br>
    </form>
</body>
</html>
```



Color:

Text:

Password:

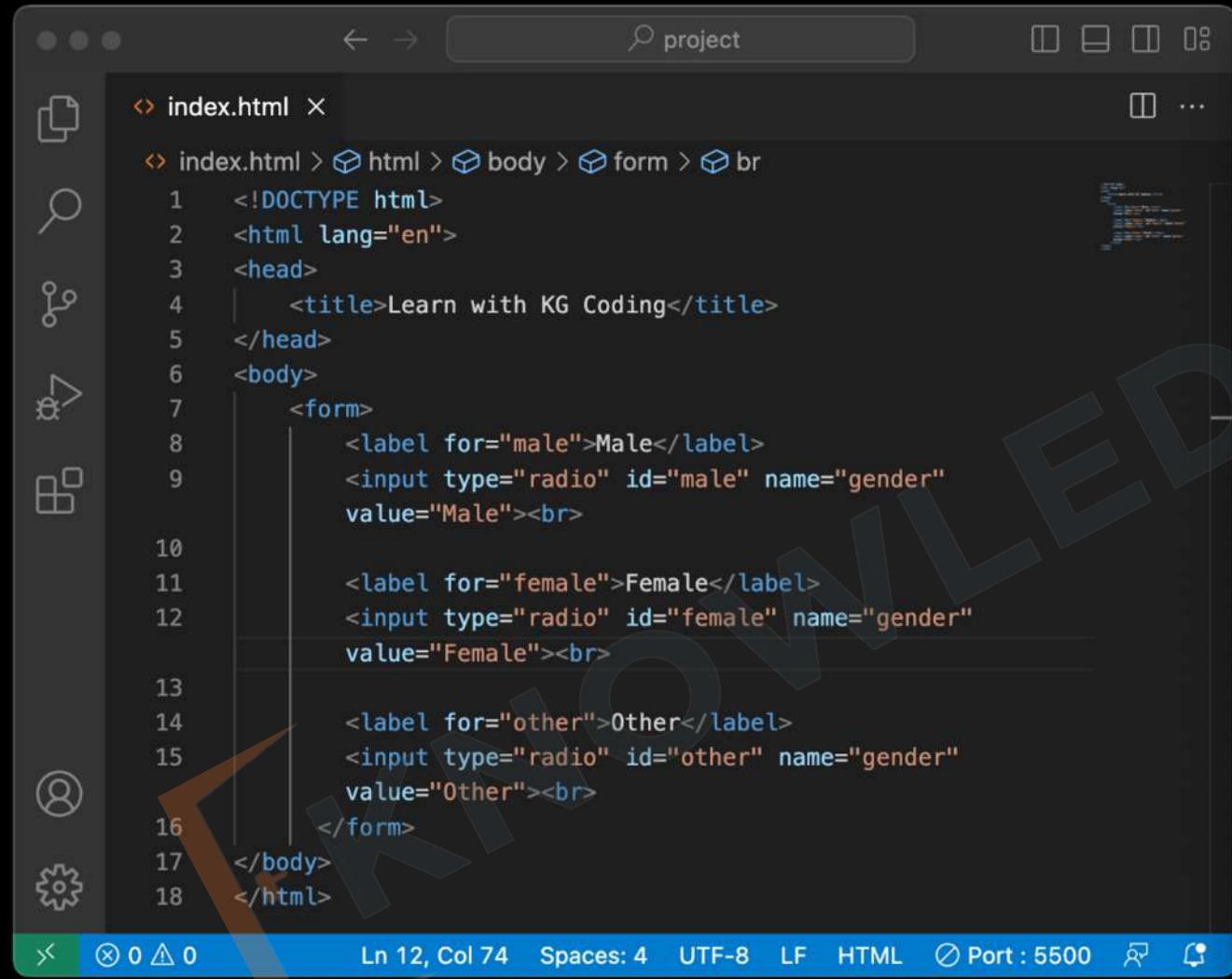
Date:

File: No file chosen

Range:

Button:

Input type: Radio

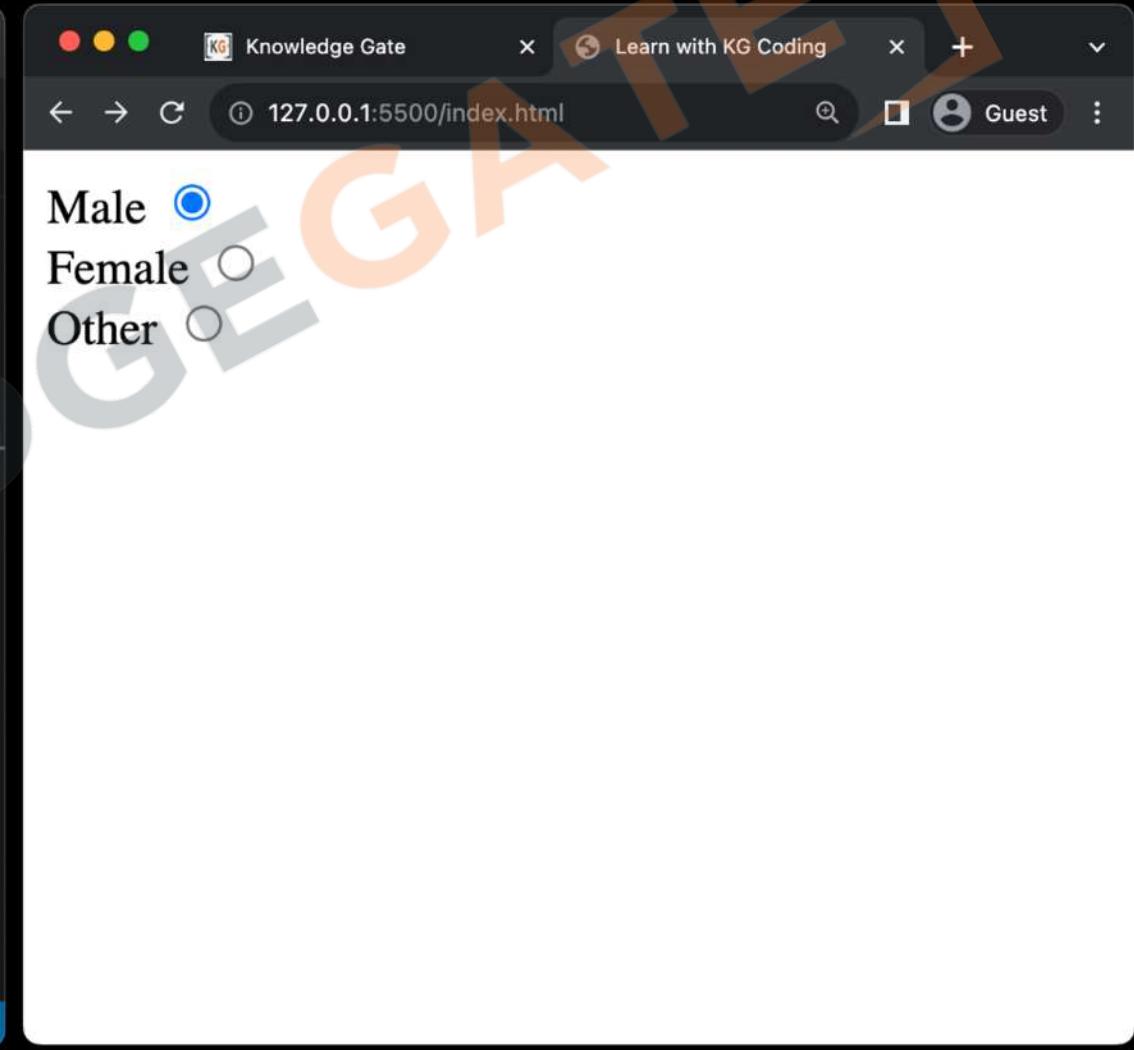


A screenshot of a code editor window titled "index.html". The code displays a simple HTML form with three radio button options: Male, Female, and Other. The "Male" option is selected, indicated by a blue circle next to the label. The "Female" and "Other" options are represented by empty circles.

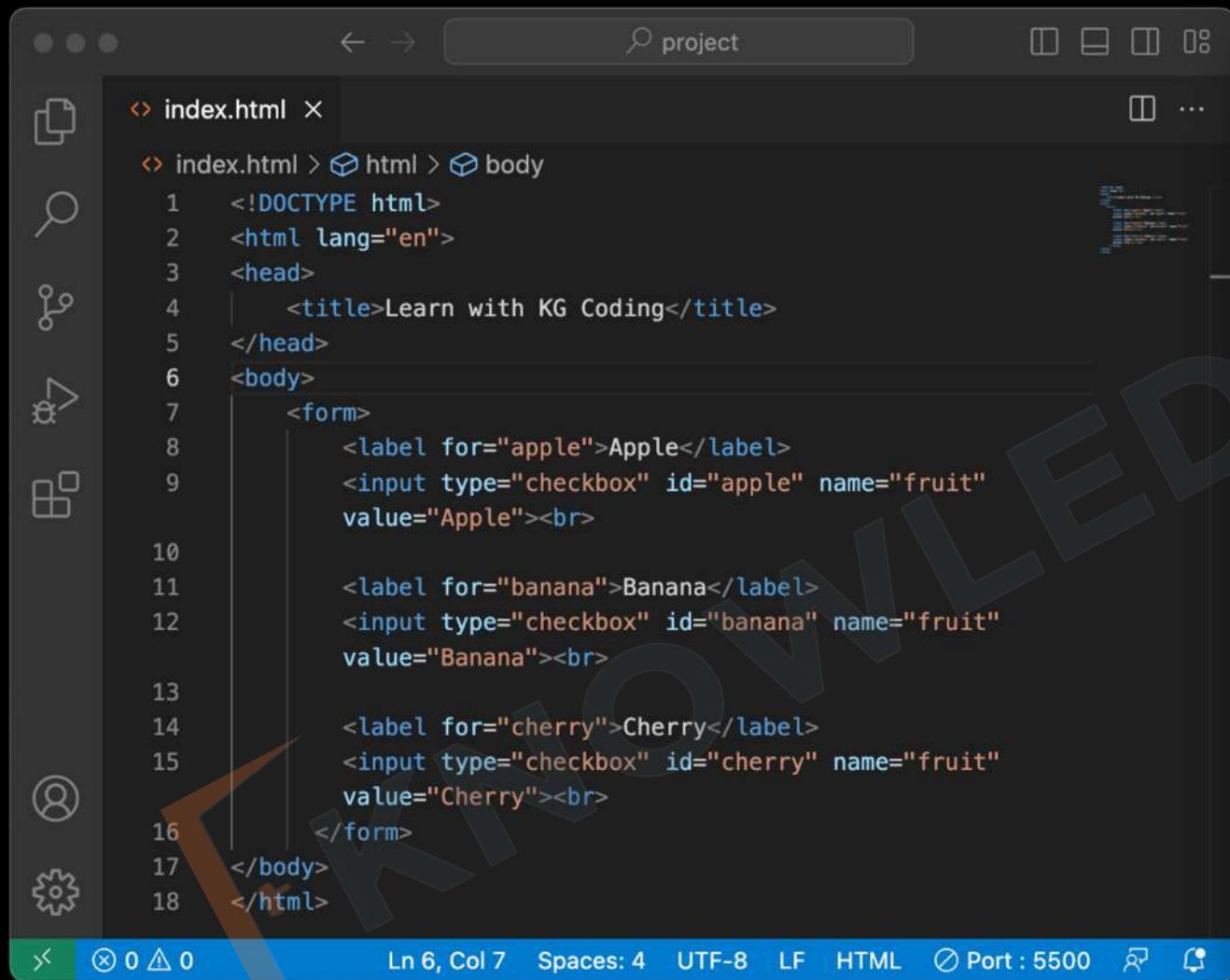
```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <form>
        <label for="male">Male</label>
        <input type="radio" id="male" name="gender" value="Male"><br>

        <label for="female">Female</label>
        <input type="radio" id="female" name="gender" value="Female"><br>

        <label for="other">Other</label>
        <input type="radio" id="other" name="gender" value="Other"><br>
    </form>
</body>
</html>
```



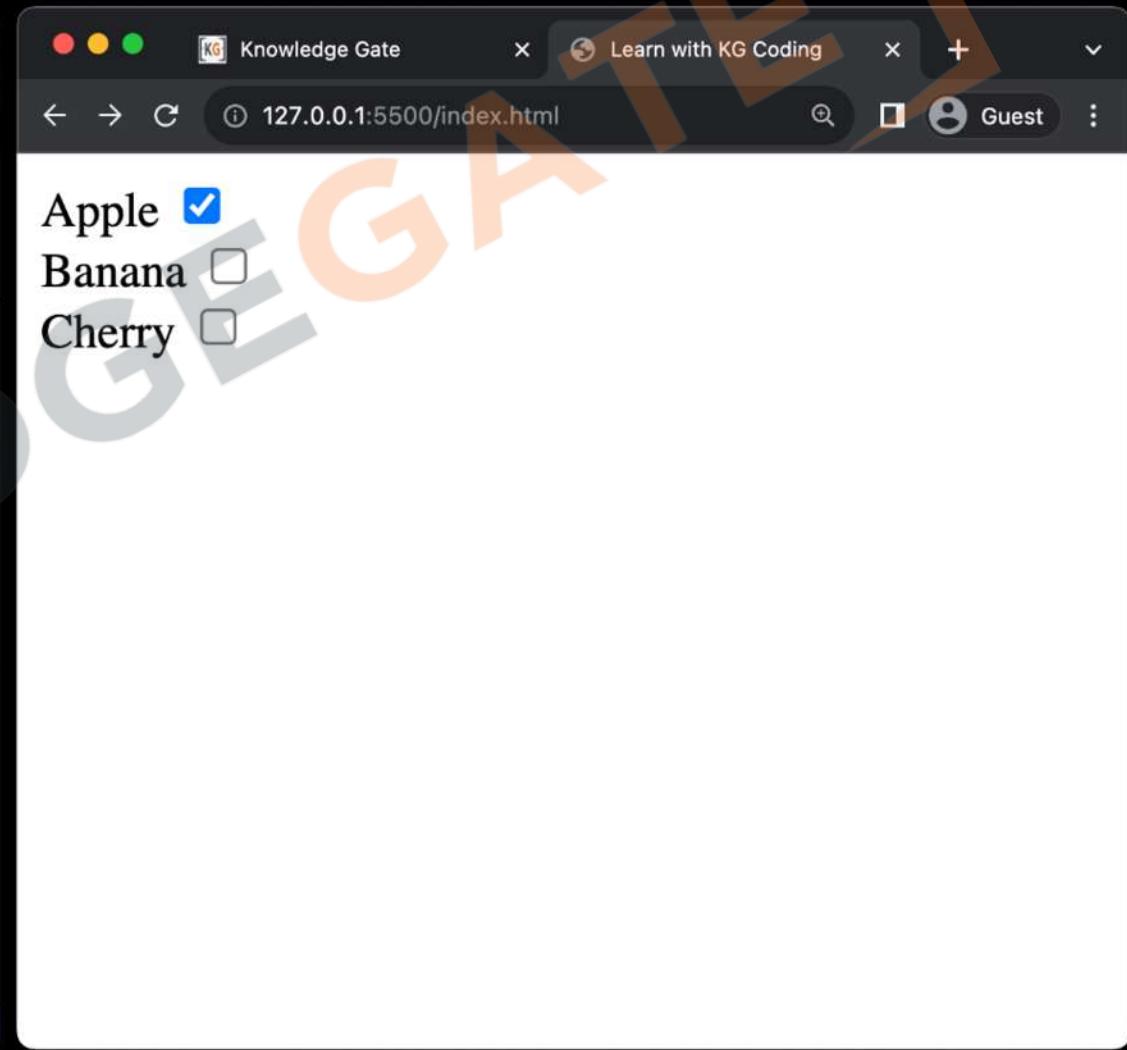
Input type: Checkbox



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <form>
        <label for="apple">Apple</label>
        <input type="checkbox" id="apple" name="fruit" value="Apple"><br>

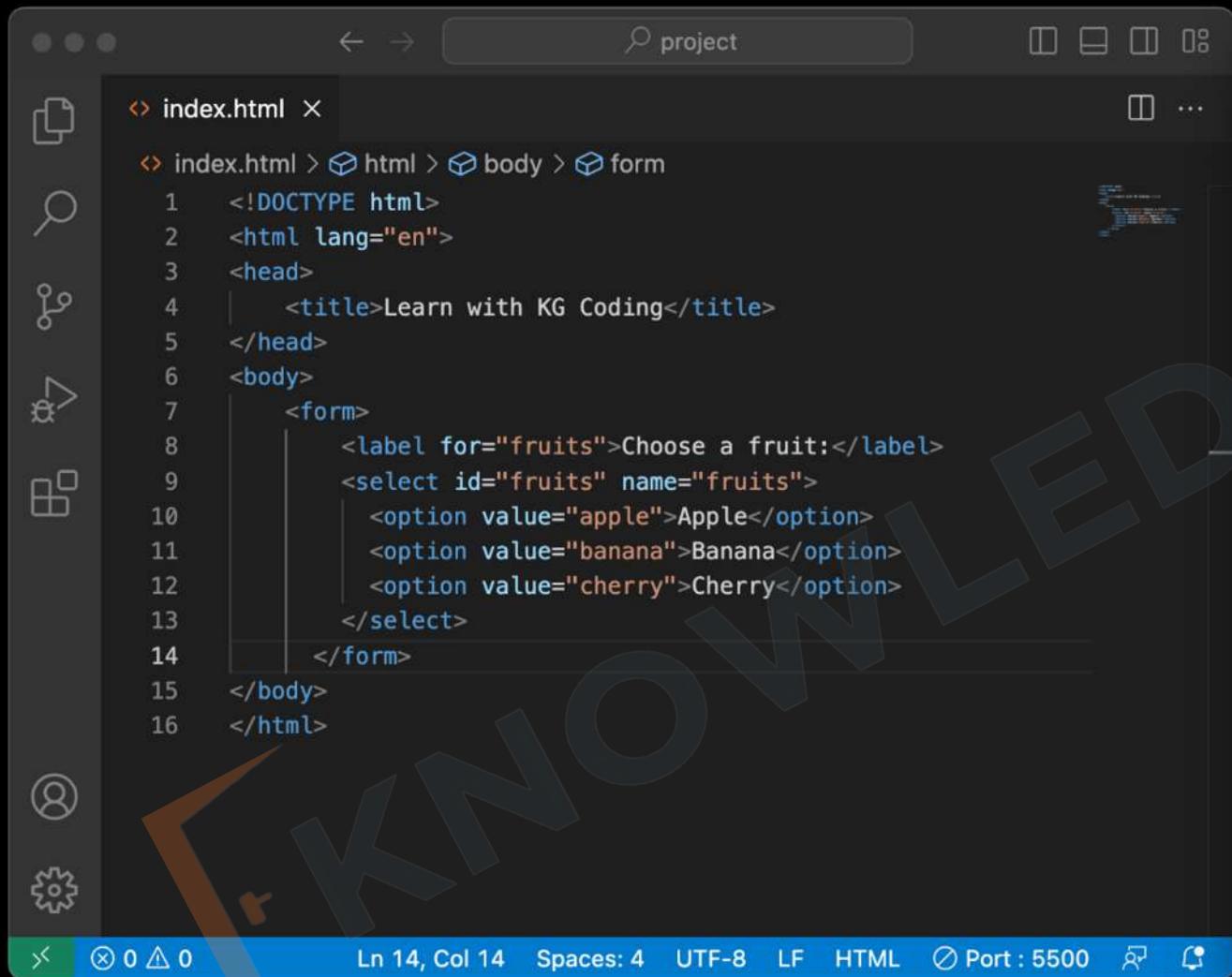
        <label for="banana">Banana</label>
        <input type="checkbox" id="banana" name="fruit" value="Banana"><br>

        <label for="cherry">Cherry</label>
        <input type="checkbox" id="cherry" name="fruit" value="Cherry"><br>
    </form>
</body>
</html>
```



Apple Banana Cherry

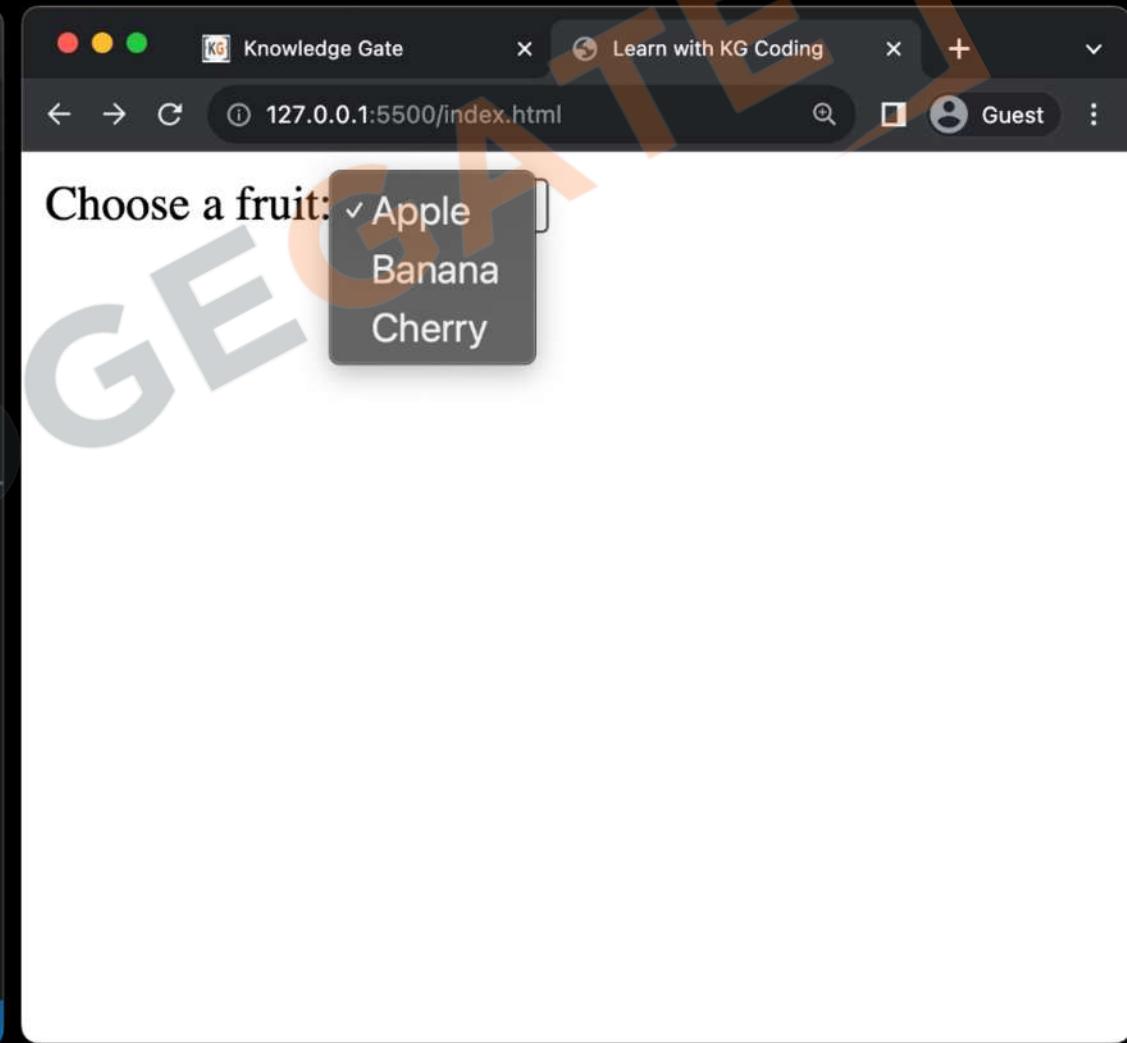
Input type: Select



A screenshot of a code editor window titled "project". The file "index.html" is open, showing the following HTML code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <form>
        <label for="fruits">Choose a fruit:</label>
        <select id="fruits" name="fruits">
            <option value="apple">Apple</option>
            <option value="banana">Banana</option>
            <option value="cherry">Cherry</option>
        </select>
    </form>
</body>
</html>
```

The code editor has a dark theme with light-colored syntax highlighting. The status bar at the bottom shows "Ln 14, Col 14", "Spaces: 4", "UTF-8", "LF", "HTML", "Port : 5500", and a refresh icon.



A screenshot of a web browser window titled "Knowledge Gate" with the URL "127.0.0.1:5500/index.html". The page content is:

Choose a fruit: ✓ Apple
Banana
Cherry

The "Apple" option is selected, indicated by a checked checkbox icon and the word "✓". The browser interface includes standard navigation buttons and a user profile icon.

Input type: TextArea

The image shows a code editor on the left and a browser window on the right. The code editor displays the file 'index.html' with the following content:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <textarea rows="4" cols="40">
        Default text here.
    </textarea>
</body>
</html>
```

The browser window shows the rendered HTML. It features a large text area with a gray border. Inside the text area, the text "Default text here." is displayed in a black font. The browser's title bar reads "Knowledge Gate" and the address bar shows "127.0.0.1:5500/index.html".

1. **Purpose:** `<textarea>` is used for multi-line text input in forms.
 1. **rows Property:** Specifies the visible number of lines in the textarea.
 2. **cols Property:** Sets the visible width measured in average character widths.
2. **Resizable:** Some browsers allow users to manually resize the textarea.

iFrame Tag



Using iFrames

The image shows a code editor on the left and a web browser on the right. The code editor displays the following HTML code:

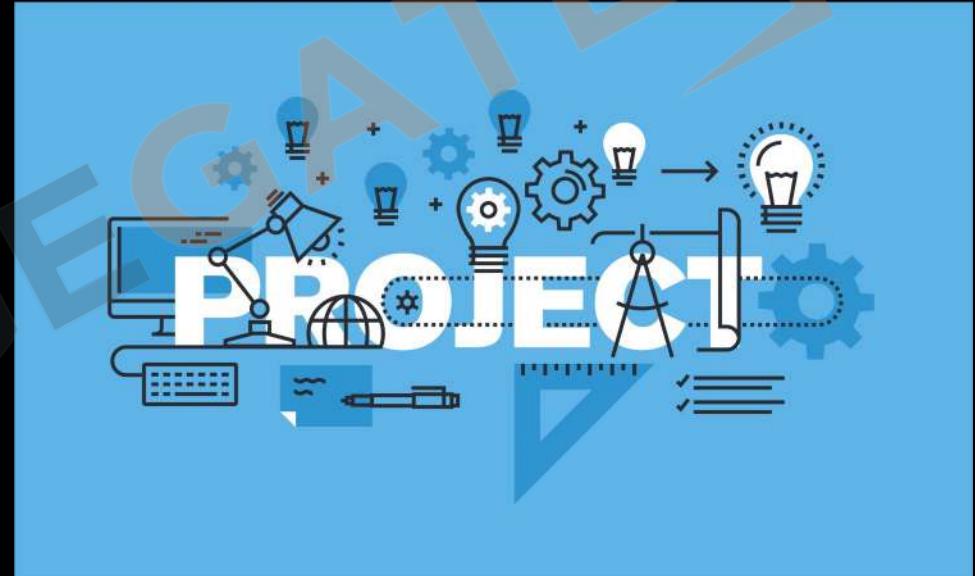
```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Learn with KG Coding</title>
</head>
<body>
    <iframe width="300" height="200" src="https://en.wikipedia.org/wiki/Main_Page"></iframe>
</body>
</html>
```

The browser window shows the rendered output of the code, which is a Wikipedia page with the title "WIKIPEDIA The Free Encyclopedia" and a link to "Main Page".

1. **Embedded Content:** Allows you to embed another webpage or multimedia content within a webpage.
2. **src Attribute:** Specifies the URL of the content to be embedded.
3. **Dimensions:** Width and height can be set using width and height attributes.

List, Tables & Forms

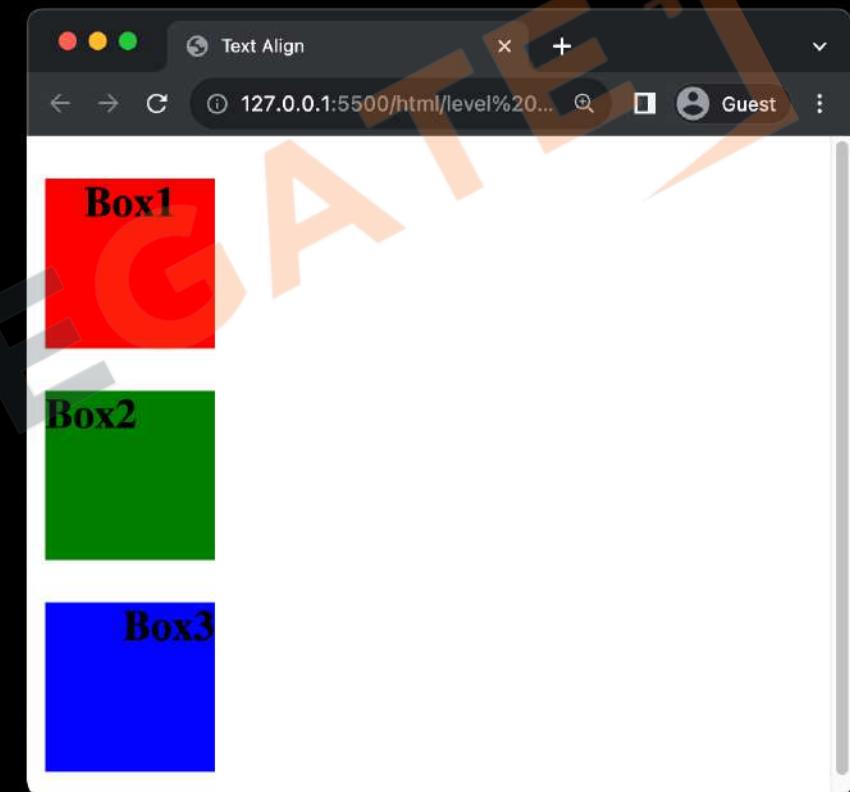
1. Create a **page** with all type of ordered list and one unordered list.
2. Create a **table** with headings, captions and a few rows. One of heading should take at least 3 columns.
3. Create a **contact me** form with relevant details for your resume website.
4. Use iFrame to add a YouTube video to your page.





Text-Align Property

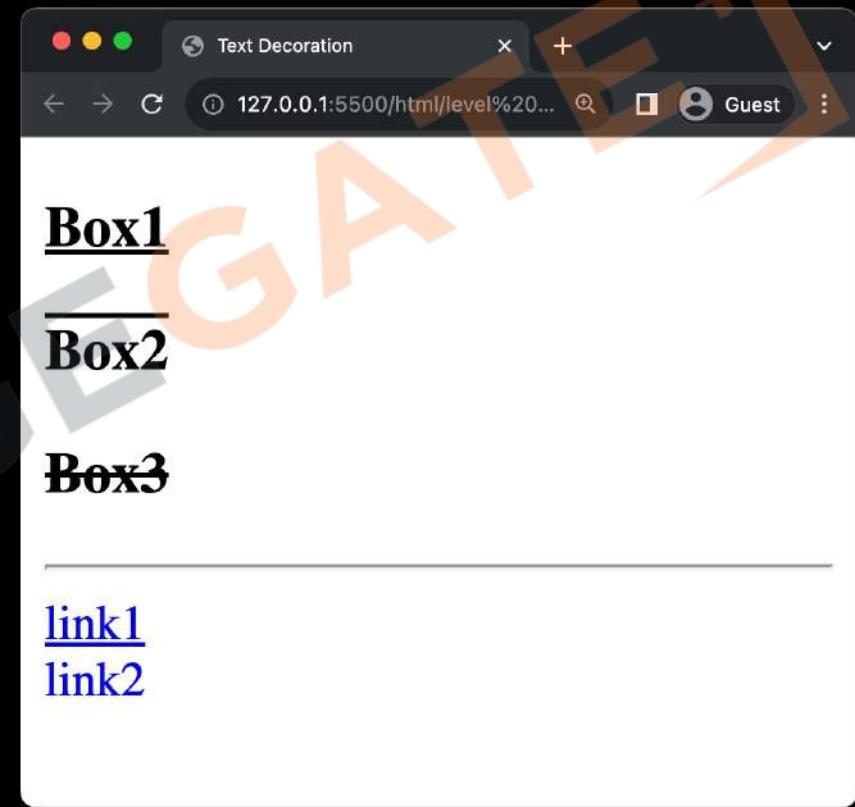
```
<head>
  <title>Text Align</title>
  <style>
    .box {height: 100px; width: 100px;}
    #box1 {background-color: red; text-align: center;}
    #box2 {background-color: green; text-align: left;}
    #box3 {background-color: blue; text-align: right;}
  </style>
</head>
<body>
  <h3 id="box1" class="box">Box1</h3>
  <h3 id="box2" class="box">Box2</h3>
  <h3 id="box3" class="box">Box3</h3>
</body>
```



- **Usage:** Controls the horizontal alignment of text within an element.
- **Values:** Can take values like left, right, center, and justify.
- **Visual Appeal:** Enhances readability and visual appeal by organizing text neatly.

Text-Decoration Property

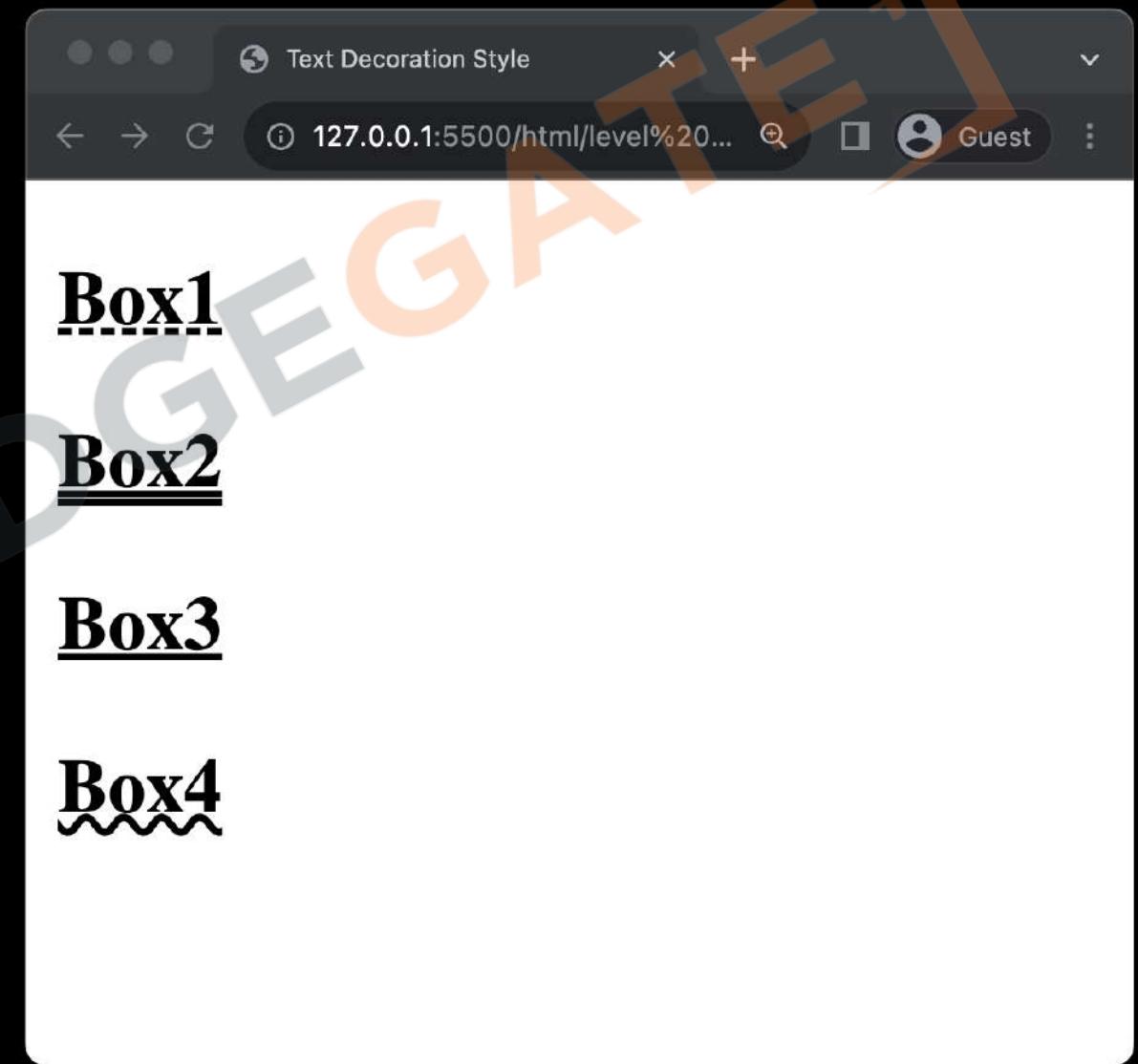
```
<head>
  <title>Text Decoration</title>
  <style>
    #box1 {text-decoration: underline;}
    #box2 {text-decoration: overline;}
    #box3 {text-decoration: line-through;}
  </style>
</head>
<body>
  <h3 id="box1" class="box">Box1</h3>
  <h3 id="box2" class="box">Box2</h3>
  <h3 id="box3" class="box">Box3</h3> <hr>
  <a href="#">link1</a> <br>
  <a href="#" style="text-decoration: none;">link2</a>
</body>
```



- **Usage:** Modifies the appearance of inline text.
- **Values:** Options include **none**, **underline**, **overline**, and **line-through**.
- **Hyperlinks:** Commonly used to remove underlines from hyperlinks for aesthetic purposes.

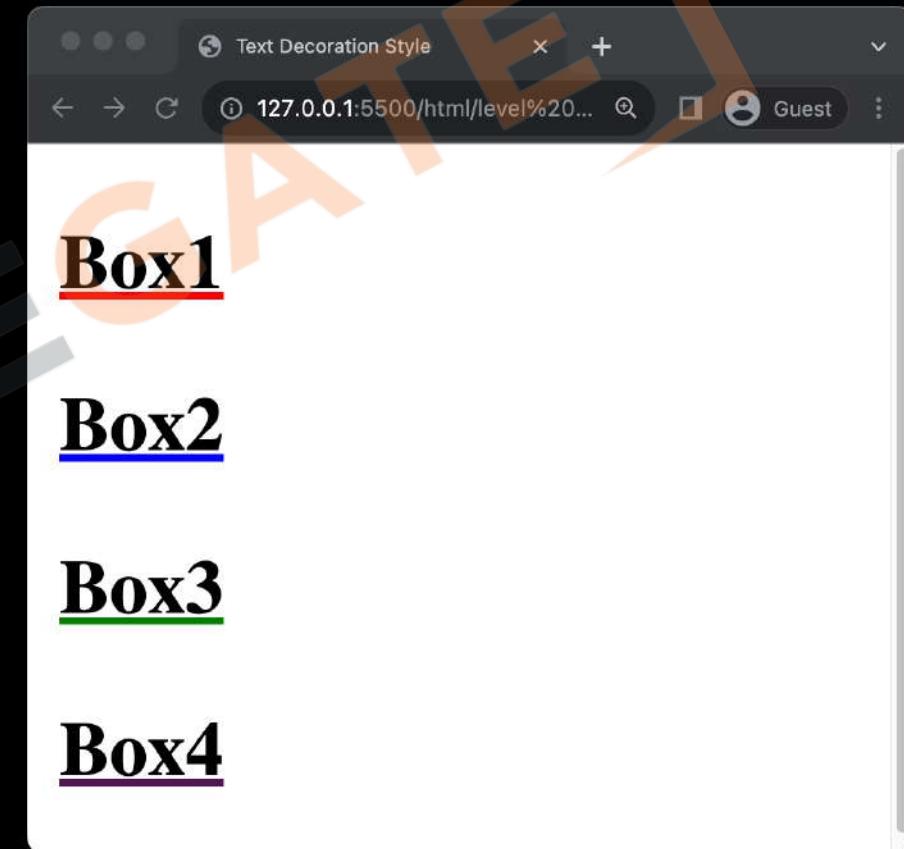
Text-Decoration Property (style)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Text Decoration Style</title>
    <style>
        .box {text-decoration: underline;}
        #box1 {text-decoration-style: dashed;}
        #box2 {text-decoration-style: double;}
        #box3 {text-decoration-style: solid;}
        #box4 {text-decoration-style: wavy;}
    </style>
</head>
<body>
    <h3 id="box1" class="box">Box1</h3>
    <h3 id="box2" class="box">Box2</h3>
    <h3 id="box3" class="box">Box3</h3>
    <h3 id="box4" class="box">Box4</h3>
</body>
</html>
```



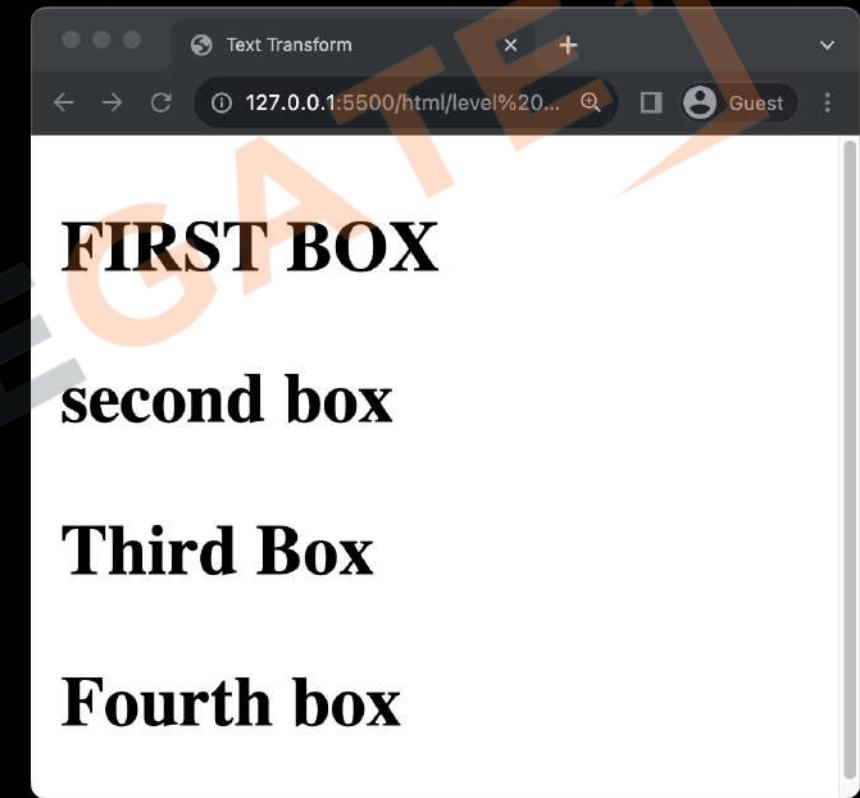
Text-Decoration Property (color)

```
<head>
  <title>Text Decoration Style</title>
  <style>
    .box {text-decoration: underline;}
    #box1 {text-decoration-color: red;}
    #box2 {text-decoration-color: blue;}
    #box3 {text-decoration-color: green;}
    #box4 {text-decoration-color: rgb(86, 20, 86);}
  </style>
</head>
<body>
  <h3 id="box1" class="box">Box1</h3>
  <h3 id="box2" class="box">Box2</h3>
  <h3 id="box3" class="box">Box3</h3>
  <h3 id="box4" class="box">Box4</h3>
</body>
```



Text-Transform Property

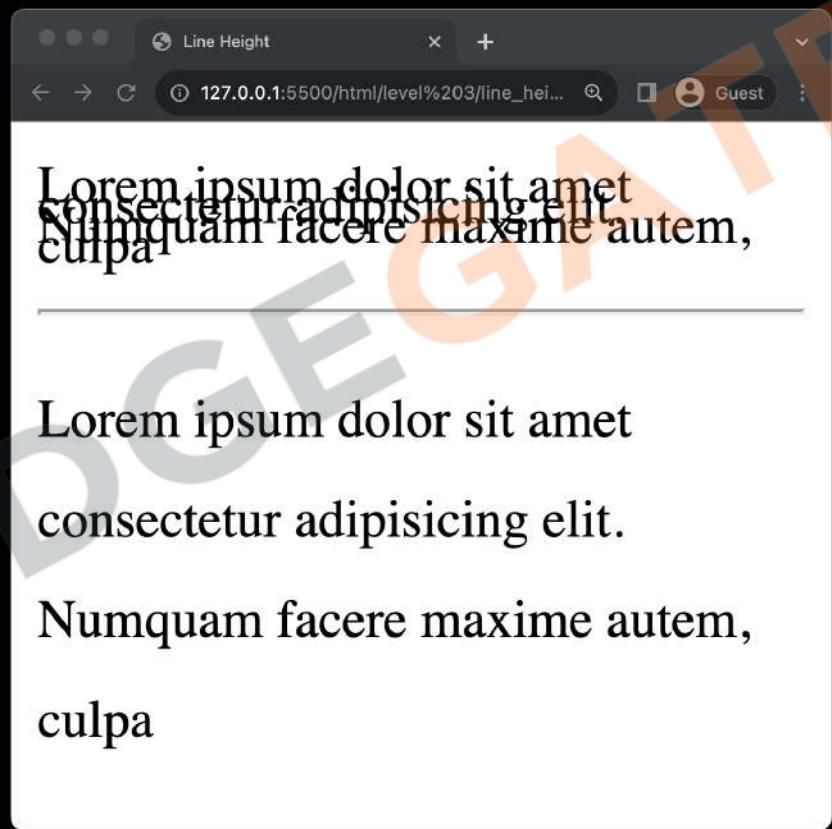
```
<head>
  <title>Text Transform</title>
  <style>
    #box1 {text-transform: uppercase;}
    #box2 {text-transform: lowercase;}
    #box3 {text-transform: capitalize;}
    #box4 {text-transform: none;}
  </style>
</head>
<body>
  <h3 id="box1" class="box">First box</h3>
  <h3 id="box2" class="box">Second box</h3>
  <h3 id="box3" class="box">Third box</h3>
  <h3 id="box4" class="box">Fourth box</h3>
</body>
```



- **Usage:** Controls the **capitalization** of text.
- **Common Values:** Can be **uppercase**, **lowercase**, or **capitalize**.
- **None Value:** none value **disables** text transformations.
- **Typography:** Useful for setting text style and improving **typography**

Line Height

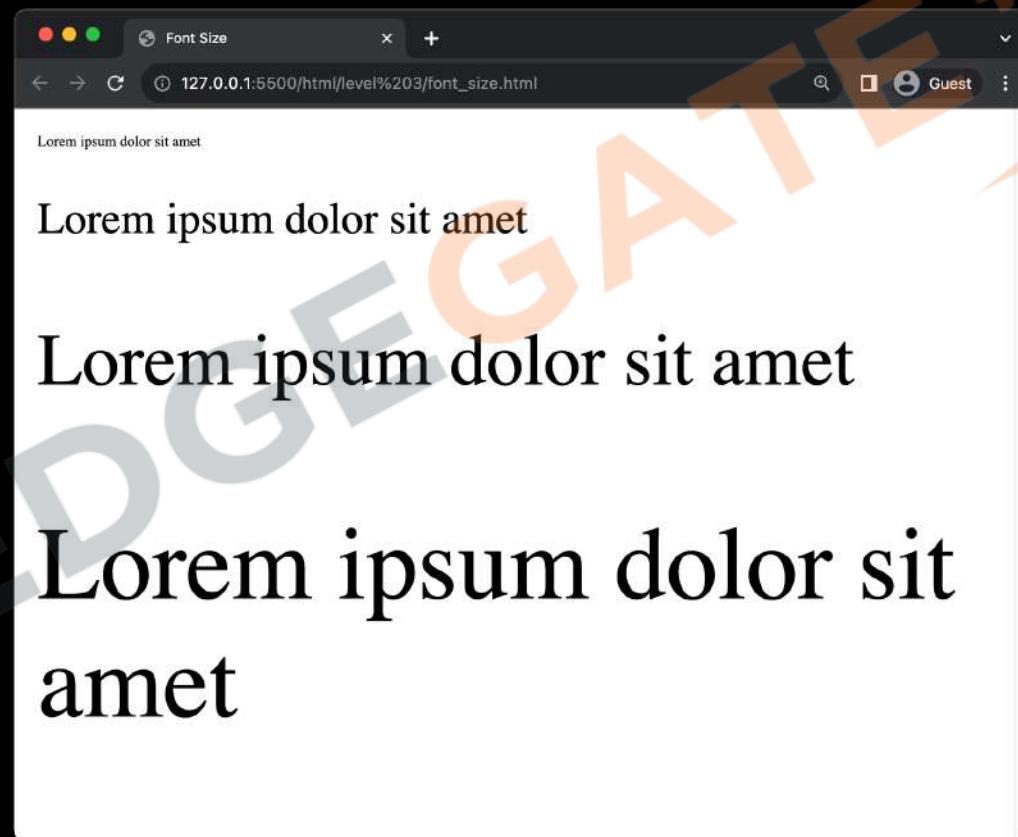
```
<head>
  <title>Line Height</title>
  <style>
    #first { line-height: 6px; }
    #second { line-height: 30px; }
  </style>
</head>
<body>
  <p id="first">Lorem ipsum dolor sit amet
consectetur adipisicing elit. Numquam facere
maxime autem, culpa</p> <hr>
  <p id="second">Lorem ipsum dolor sit amet
consectetur adipisicing elit. Numquam facere
maxime autem, culpa</p>
</body>
```



- **Usage:** Adjusts the amount of **space above and below** inline elements.
- **Readability:** Enhances text readability by preventing overcrowding.
- **Vertical Spacing:** Useful for **controlling vertical spacing** between lines of text.

Font Property (font-size)

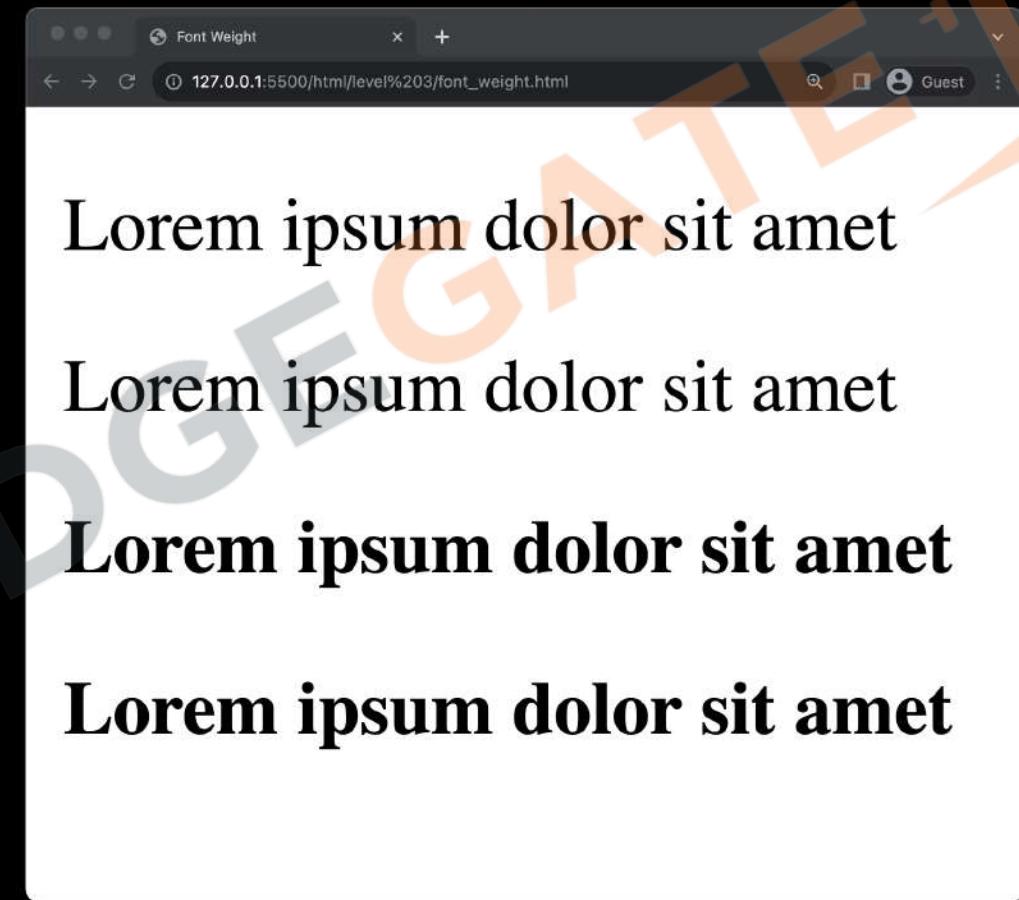
```
<head>
  <title>Font Size</title>
  <style>
    #first {font-size: 5px;}
    #second {font-size: 15px;}
    #third {font-size: 25px;}
    #fourth {font-size: 35px;}
  </style>
</head>
<body>
  <p id="first">Lorem ipsum dolor sit amet</p>
  <p id="second">Lorem ipsum dolor sit amet</p>
  <p id="third">Lorem ipsum dolor sit amet</p>
  <p id="fourth">Lorem ipsum dolor sit amet</p>
</body>
```



- **Usage:** Sets the **size of the font** in web content.
- **Responsiveness:** Helps in creating **responsive designs** adaptable to various screen **sizes**.
- **Readability:** Crucial for ensuring the **readability** of text on websites.

Font Property (font-weight)

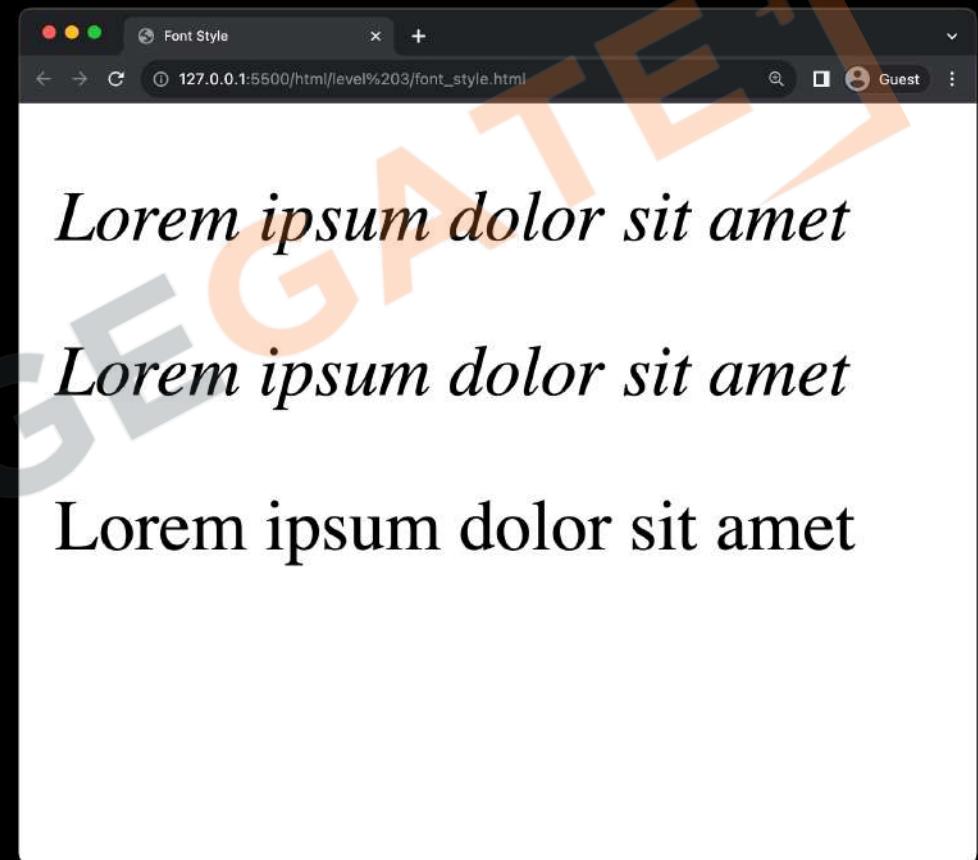
```
<head>
    <title>Font Weight</title>
    <style>
        #first {font-weight: 100;}
        #second {font-weight: 400;}
        #third {font-weight: 600;}
        #fourth {font-weight: 900;}
    </style>
</head>
<body>
    <p id="first">Lorem ipsum dolor sit amet</p>
    <p id="second">Lorem ipsum dolor sit amet</p>
    <p id="third">Lorem ipsum dolor sit amet</p>
    <p id="fourth">Lorem ipsum dolor sit amet</p>
</body>
```



- **Usage:** Defines the **thickness** of characters in a font.
- **Values:** Can take values like **normal**, **bold**, **bolder**, or numeric values (100 to 900).
- **Text Emphasis:** Utilized to emphasize text or **create contrast**.

Font Property (font-style)

```
<head>
  <title>Font Style</title>
  <style>
    #first {font-style: italic;}
    #second {font-style: oblique;}
    #third {font-style: normal;}
  </style>
</head>
<body>
  <p id="first">Lorem ipsum dolor sit amet</p>
  <p id="second">Lorem ipsum dolor sit amet</p>
  <p id="third">Lorem ipsum dolor sit amet</p>
</body>
```



- **Usage:** Controls the **style** of the font, mainly affecting its inclination.
- **Values:** Common values are **normal**, **italic**, and **oblique**.
- **Text Formatting:** Useful for **highlighting** or distinguishing

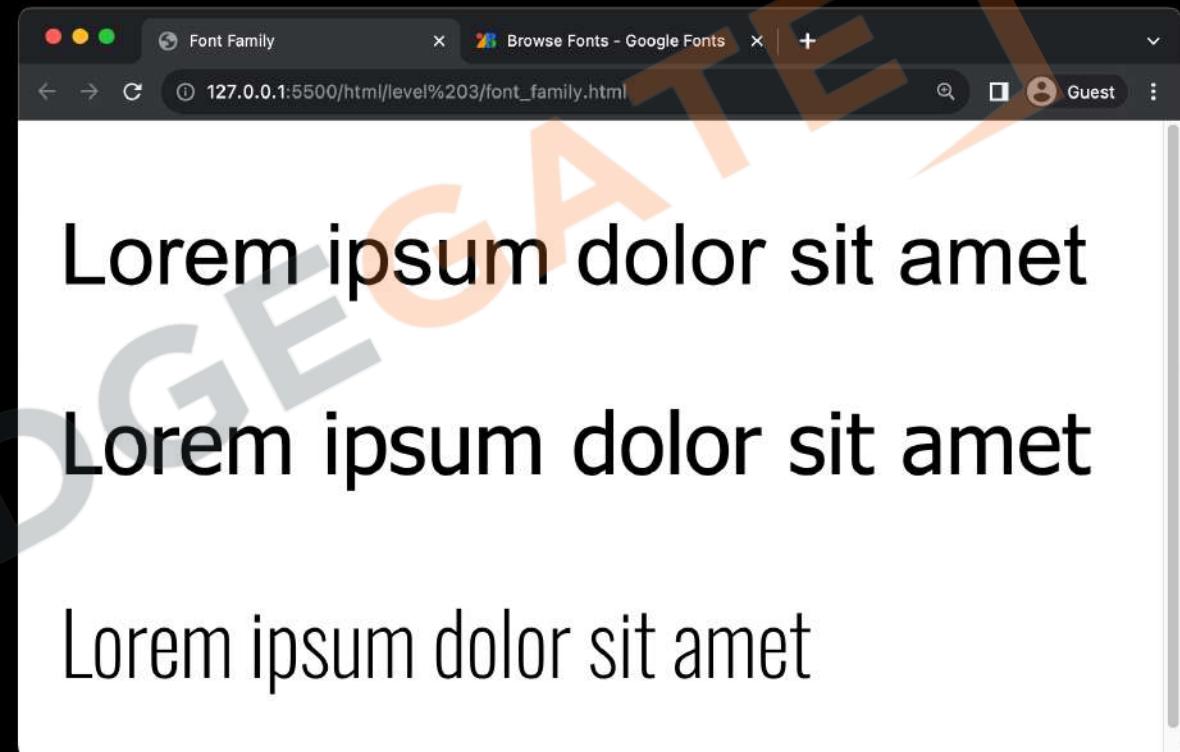
Font Family

- **Usage:** Defines which font should be used for text within an element.
- **Specific Fonts:** Common choices include Arial, Segoe UI, Times New Roman, and others.
- **Fallback Mechanism:** Incorporate a fallback font family in case the primary font is unavailable; helps in maintaining the site aesthetics.
- **Web Safe Fonts:** Employ web-safe fonts to ensure consistency across different browsers and operating systems.
- **Generic Family:** Always end the font family list with a generic family like serif or sans-serif as a last resort option.

Arial Narrow
Book Antiqua
Cambria
Century Gothic
Consolas
COPPERPLATE
Georgia
Impact
Lucida Sans Unicode
Papyrus
Script MT Bold
Tahoma
Times New Roman
Verdana

Font Family

```
<head>
  <title>Font Family</title>
  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <link href="https://fonts.googleapis.com/css2?family=Oswald:wght@200&
display=swap" rel="stylesheet">
  <style>
    #first {font-family: Arial, Helvetica, sans-serif;}
    #second {font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
    sans-serif;}
    #third {font-family: 'Oswald', sans-serif;}
  </style>
</head>
<body>
  <p id="first">Lorem ipsum dolor sit amet</p>
  <p id="second">Lorem ipsum dolor sit amet</p>
  <p id="third">Lorem ipsum dolor sit amet</p>
</body>
```



Font Family (Fallbacks)

```
font-family: "Open Sans", "Helvetica Neue", "Arial", sans-serif;
```

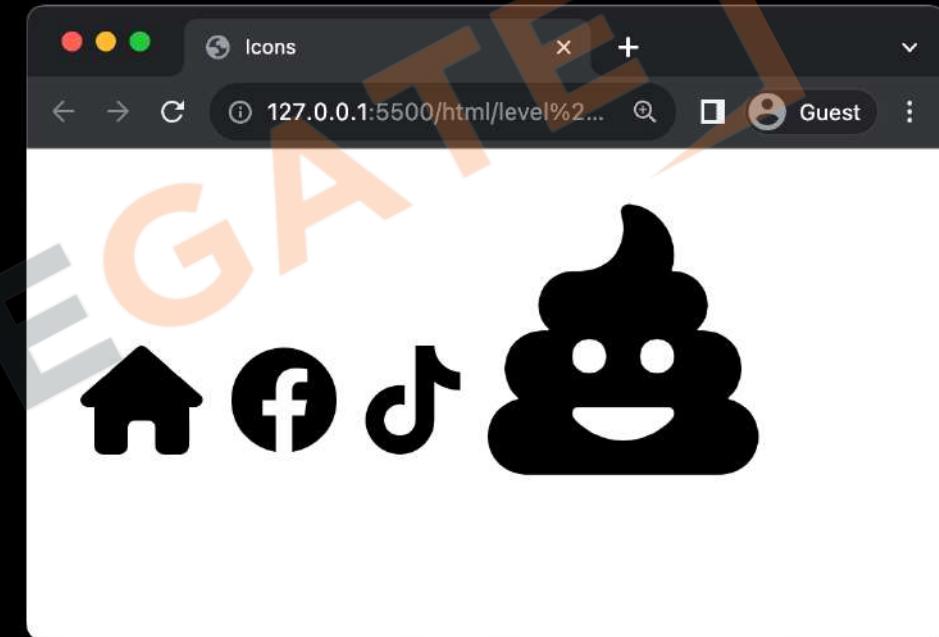
When defining a font-family in CSS, you can list multiple fallback fonts. There's no strict limit, but keep it reasonable for maintainability and readability.

In this example:

- If "Open Sans" is not available, the browser will try "Helvetica Neue".
- If "Helvetica Neue" is not available, the browser will try "Arial".
- If none of the specified fonts are available, the browser will use the default sans-serif font.

Icons using Fonts

```
<head>
  <title>Icons</title>
  <script src="https://kit.fontawesome.com/43290fa92d.js" crossorigin="anonymous"></script>
</head>
<body>
  <i class="fa-solid fa-house"></i>
  <i class="fa-brands fa-facebook"></i>
  <i class="fa-brands fa-tiktok"></i>
  <i class="fa-solid fa-poo" style="font-size: 40px;"></i>
</body>
```



Using <https://fontawesome.com>

Practice Set

Text Properties

- Create one div inside another div.
Set id and text **outer** to outer div,
set id and text **inner** to inner div.
Set outer div text size to 25px.
Set inner to 10px.
- Use icons from fontawesome.com and
use icons of LinkedIn and GitHub
- Create an **Heading** at the centre and
make capitalized
- Use Font family for the whole page
to **TimeNewRoman**





What is Box Model

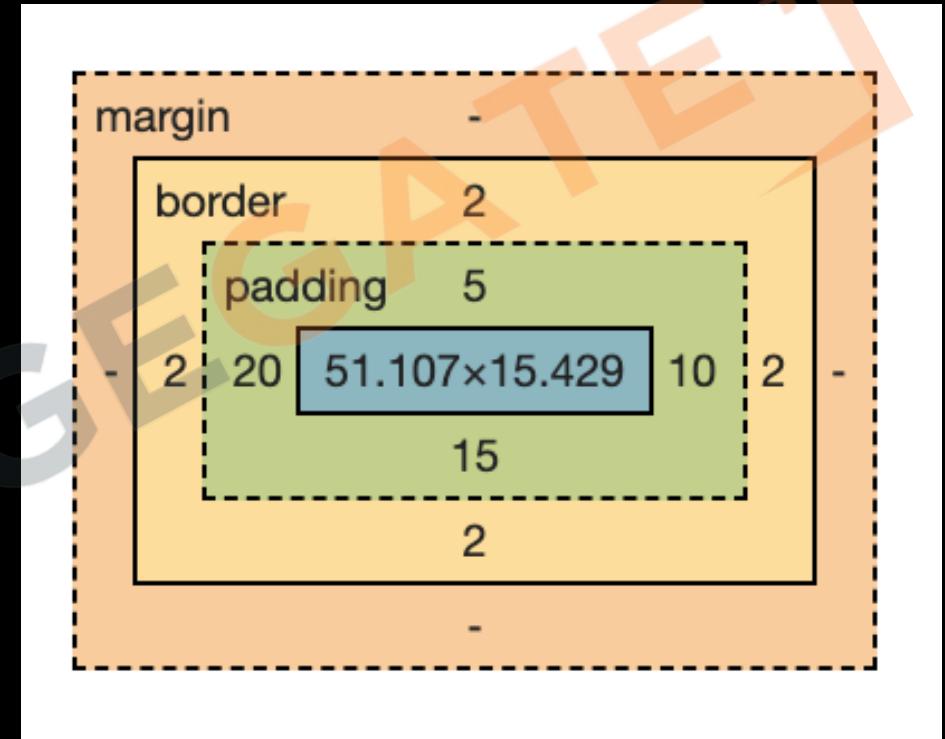
ਮਿਠਾਈ ਮਣਡਾਰ ਪੇ ਚਲੋ

- **Core Concept:** Central concept in CSS that outlines the design and layout of elements on the web page.
- **Components:** Consists of four main components
 - margin, border, padding, and content.
- **Margin:** The space outside the border, separating the element from others.
- **Border:** The outline that encapsulates the padding and content.
- **Padding:** The space between the border and the actual content, providing a buffer.
- **Content:** The innermost layer where text, images, or other media are housed.



Padding Property

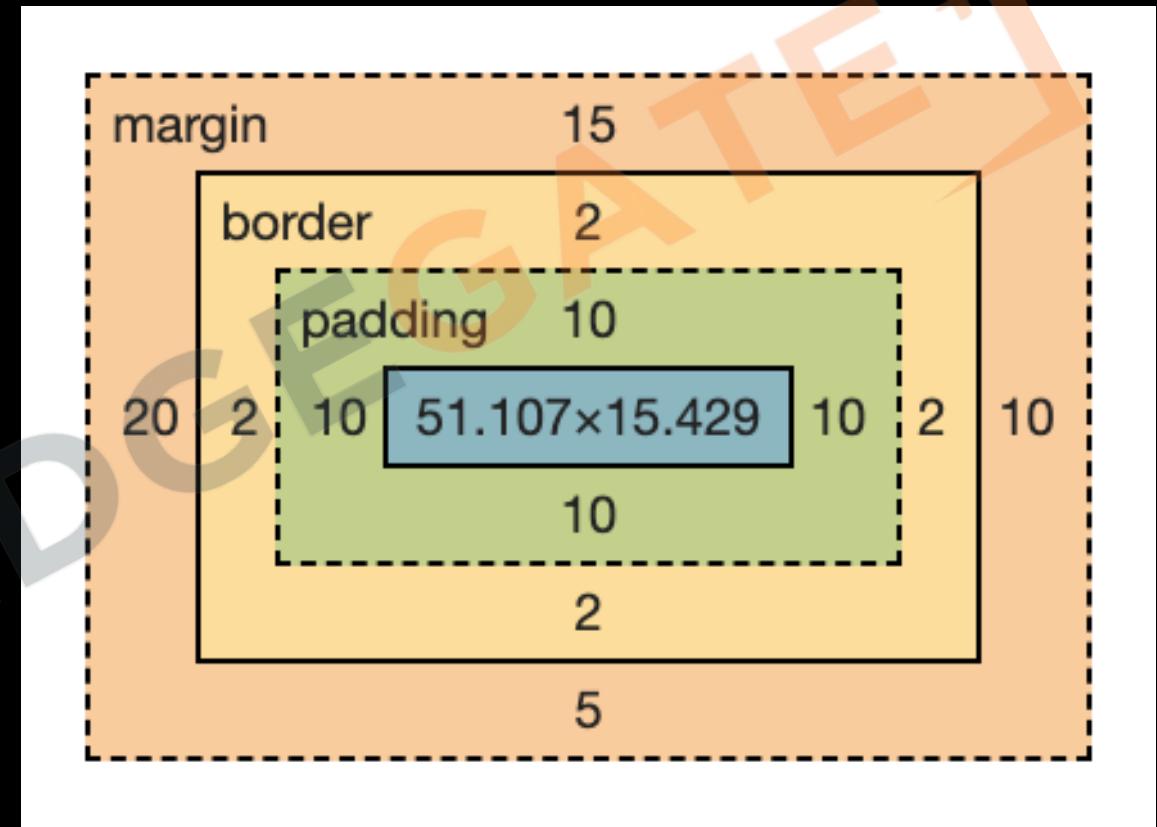
```
<head>
  <title>Padding</title>
  <style>
    * { margin: 0; padding: 0; }
    #button1 {
      padding: 5px 10px 15px 20px;
      background-color: aquamarine;
    }
  </style>
</head>
<body>
  <button id="button1">Click Me</button>
</body>
```



- **Usage:** Defines the space between the content of an element and its border.
- **Individual Sides:** Allows setting padding for individual sides using `padding-top`, `padding-right`, `padding-bottom`, and `padding-left`.
- **Shorthand:** Can use shorthand property `padding` to set all sides at once, e.g., `padding: 10px 20px 10px 20px`.

Margin Property

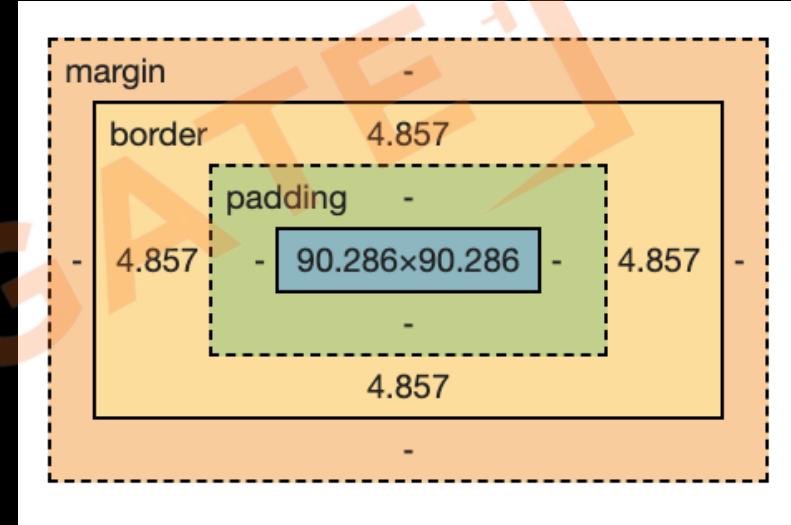
```
<head>
  <title>margin</title>
  <style>
    * { margin: 0; padding: 0; }
    #button1 {
      margin: 15px 10px 5px 20px;
      padding: 10px;
      background-color: aquamarine;
    }
  </style>
</head>
<body>
  <button id="button1">Click Me</button>
</body>
```



- **Functionality:** Sets the space around elements, separating them from others.
- **Individual Sides:** Customizable for **top**, **right**, **bottom**, and **left** sides.
- **Shorthand:** Allows quick setup, e.g., **margin: 10px 20px**. (clockwise)
- **Auto Value:** Can be used for central alignment with **auto** value.

Border Property

```
<head>
  <title>Border</title>
  <style>
    * { margin: 0; padding: 0; }
    #button1 {
      height: 100px; width: 100px;
      border: 5px dashed black;
      background-color: aquamarine;
    }
  </style>
</head>
<body>
  <button id="button1">Click Me</button>
</body>
```



- **Usage:** Creates an outline around **HTML** elements.
- **Components:** Defined by **width**, **style**, and **color** attributes.
- **Styles:** Includes options like **solid**, **dashed**, and **dotted**.
- **Shorthand:** Can set attributes at once, e.g., `border: 2px solid black`.

Border Property (border radius)

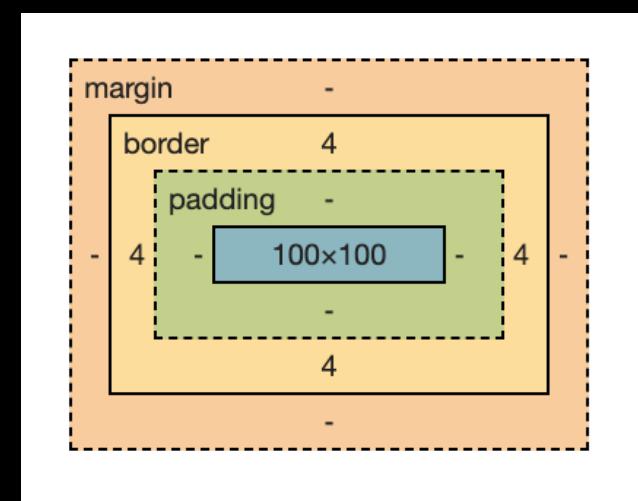
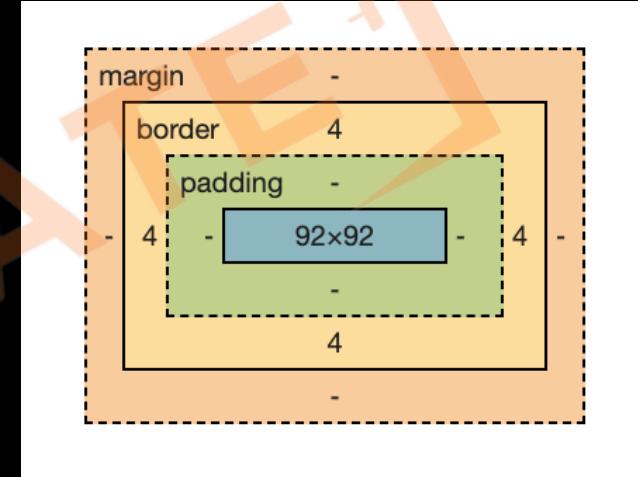
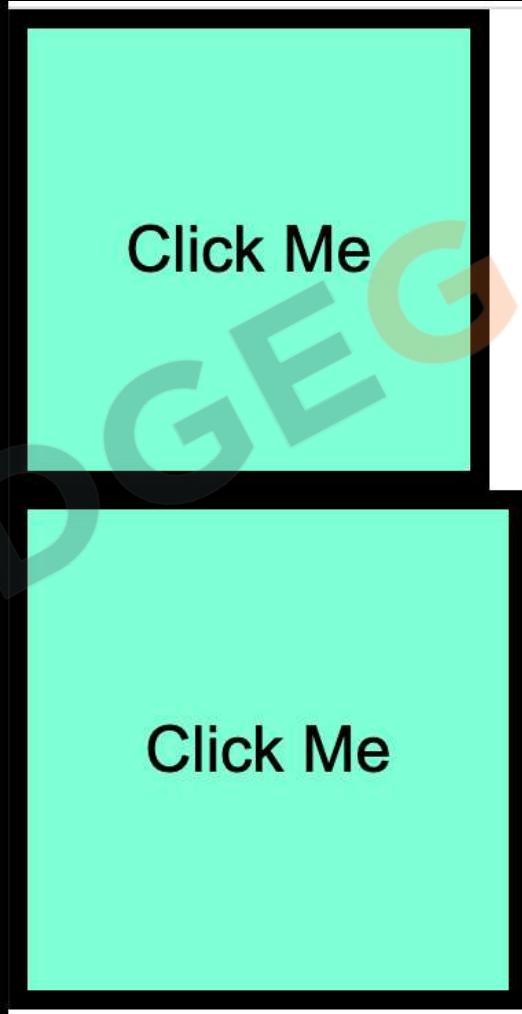
```
<head>
  <title>Border Radius</title>
  <style>
    * { margin: 0; padding: 0; }
    #button1 {
      height: 100px; width: 100px;
      border: 5px solid black;
      border-radius: 30px;
      background-color: aquamarine;
    }
  </style>
</head>
<body>
  <button id="button1">Click Me</button>
</body>
```



- **Usage:** Used to **create rounded corners** for elements.
- **Individual Corners:** Allows setting different radii for each corner.
- **Shorthand:** e.g., border-radius: 10px 20px.

Border Property (box sizing)

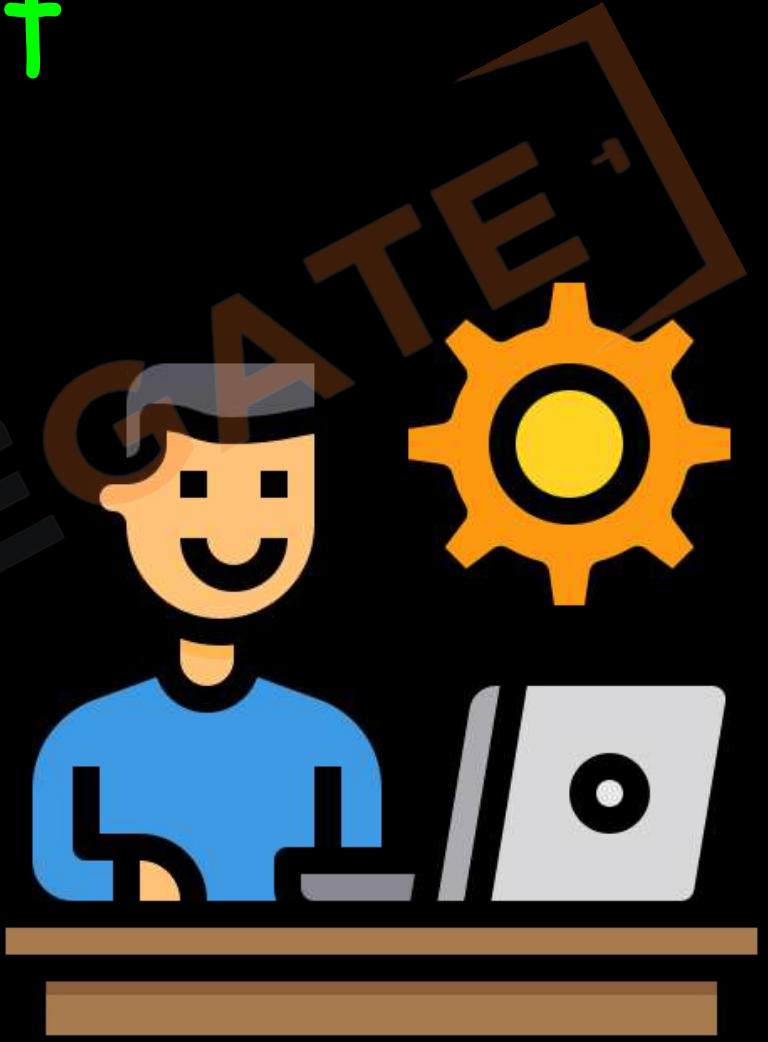
```
<head>
  <title>Box Sizing</title>
  <style>
    * { margin: 0; padding: 0; }
    button {
      height: 100px; width: 100px;
      border: 4px solid black;
      background-color: aquamarine;
    }
    #button1 { box-sizing: border-box; }
    #button2 { box-sizing: content-box; }
  </style>
</head>
<body>
  <button id="button1">Click Me</button>
  <button id="button2">Click Me</button>
</body>
```



Practice Set

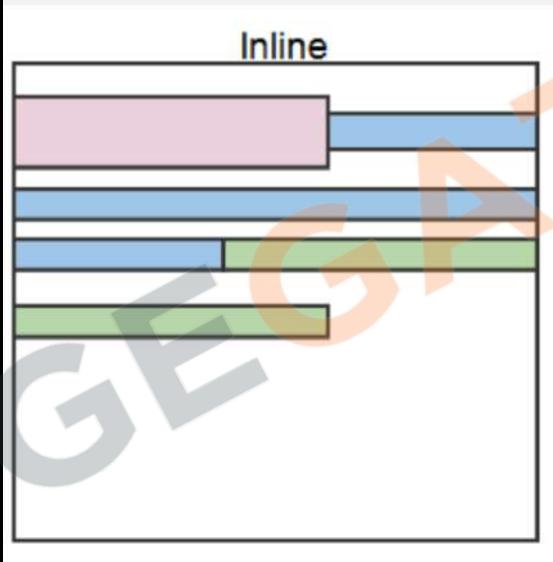
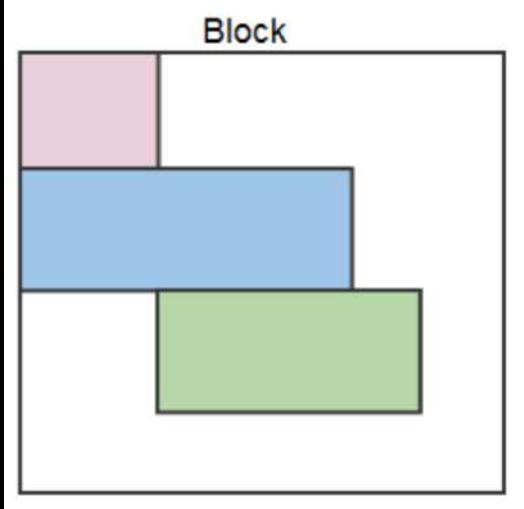
Box Model

- Create a div with height and width **200px** and observe different values in the box model by inspecting.
- Create a **button** and give **10px padding** to top and bottom and **15px** on the sides.
- Add **100px margin** to the button on all sides.
- Add dotter border with color **red** and so much that the button becomes a **circle**.
- Create two **boxes** with different box-sizing values and observe changes in box model.





Display Property (Block / Inline Elements)



Block Elements

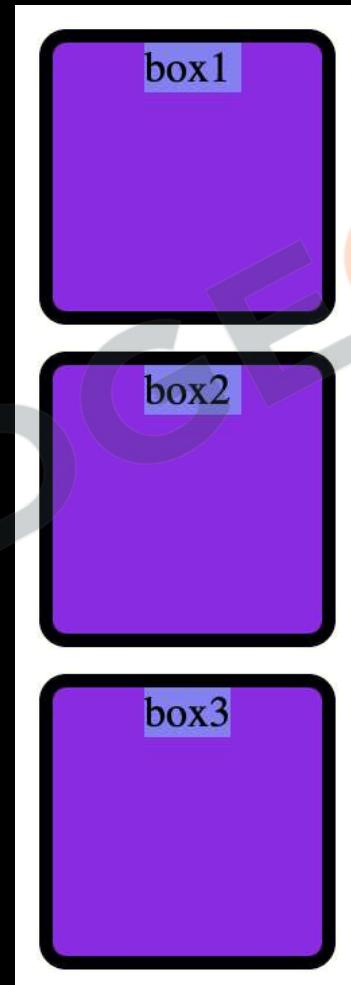
- New Line: Start on a new line.
- Full Width: Take up all horizontal space.
- Styling: Can have margins and padding.
- Size: Width and height can be set.
- Examples: <div>, <p>, <h1>, , .

Inline Elements

- Flow: Stay in line with text.
- Width: Just as wide as the content.
- No Break: No new line between elements.
- Limited Styling: Can't set size easily.
- Examples: , <a>, , .

Display Property (Block)

```
.box {  
    height: 100px;  
    width: 100px;  
    background-color: blueviolet;  
    margin: 10px;  
    text-align: center;  
    border: 5px solid black;  
    border-radius: 10px;  
  
    display: block;  
}
```



```
<head>  
    <title>Display Block</title>  
    <link rel="stylesheet" href="../../css/  
    level 5/display.css">  
</head>  
<body>  
    <div id="parent">  
        <div id="div1" class="box">box1</div>  
        <div id="div2" class="box">box2</div>  
        <div id="div3" class="box">box3</div>  
    </div>  
</body>
```

Display Property (Inline)

```
.box {  
    height: 100px;  
    width: 100px;  
    background-color: □blueviolet;  
    margin: 10px;  
    text-align: center;  
    border: 5px solid □black;  
    border-radius: 10px;  
  
    display: inline;  
}
```

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <title>Display Inline</title>  
    <link rel="stylesheet" href="../../css/  
        level 5/display_inline.css">  
</head>  
<body>  
    <div id="parent">  
        <div id="div1" class="box">box1</div>  
        <div id="div2" class="box">box2</div>  
        <div id="div3" class="box">box3</div>  
    </div>  
</body>  
</html>
```

box1

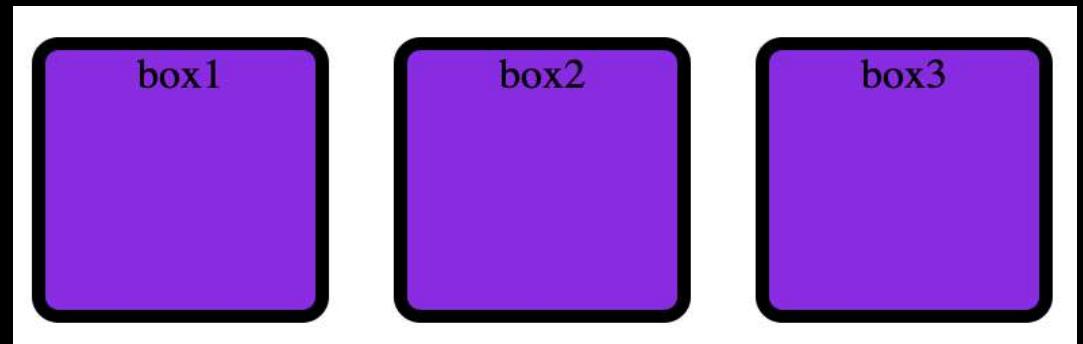
box2

box3

Display Property (Inline-Block)

```
.box {  
    height: 100px;  
    width: 100px;  
    background-color: #blueviolet;  
    margin: 10px;  
    text-align: center;  
    border: 5px solid #black;  
    border-radius: 10px;  
  
    display: inline-block;  
}
```

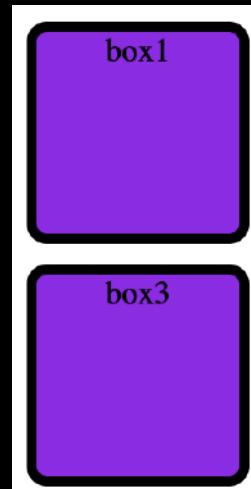
```
<head>  
    <title>Display Inline Block</title>  
    <link rel="stylesheet" href="../../css/  
        level 5/display_inline_block.css">  
</head>  
<body>  
    <div id="parent">  
        <div id="div1" class="box">box1</div>  
        <div id="div2" class="box">box2</div>  
        <div id="div3" class="box">box3</div>  
    </div>  
</body>
```



Display Property (None)

```
.box {  
    height: 100px;  
    width: 100px;  
    background-color: blueviolet;  
    margin: 10px;  
    text-align: center;  
    border: 5px solid black;  
    border-radius: 10px;  
}  
  
#div2 {  
    display: none;  
}
```

```
<head>  
    <title>Display None</title>  
    <link rel="stylesheet" href="../../css/  
        level 5/display_none.css">  
</head>  
<body>  
    <div id="parent">  
        <div id="div1" class="box">box1</div>  
        <div id="div2" class="box">box2</div>  
        <div id="div3" class="box">box3</div>  
    </div>  
</body>
```



Relative Units



CSS Units Cheat Sheet

px

Absolute pixel value

%

A percentage of the parent element.
100% is the width of the parent element

em

Relative to the font size of the element

vh

Relative to 1% of the viewport's height

rem

Relative to the font size of
the root element

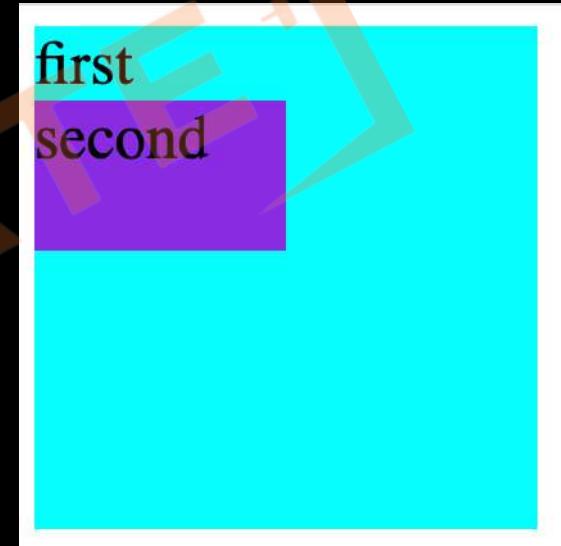
vw

Relative to 1% of the viewport's width

Relative Units (Percentage)

```
#first {  
    height: 200px;  
    width: 200px;  
    background-color: #aqua;  
    font-size: 25px;  
}  
  
#second {  
    background-color: #blueviolet;  
    width: 50%;  
    height: 30%;  
}
```

```
<body>  
    <div id="first">  
        first  
        <div id="second">  
            second  
        </div>  
    </div>  
</body>
```

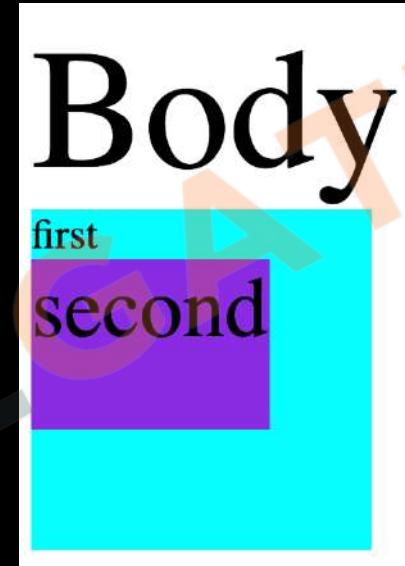


- **Relative Sizing:** Facilitates dynamic sizing **relative to parents**.
- **Adaptability:** Ensures **responsiveness** across various screens.
- **Dimensions:** Quickly set width and height as a percentage.

Relative Units (EM)

```
body {  
    font-size: 100px;  
}  
  
#first {  
    height: 200px;  
    width: 200px;  
    background-color: #aqua;  
    font-size: 25px;  
}  
  
#second {  
    background-color: #blueviolet;  
    width: 70%;  
    height: 50%;  
    font-size: 2em;  
}
```

```
<body>  
    Body  
        <div id="first">  
            first  
            <div id="second">  
                second  
            </div>  
        </div>  
</body>
```

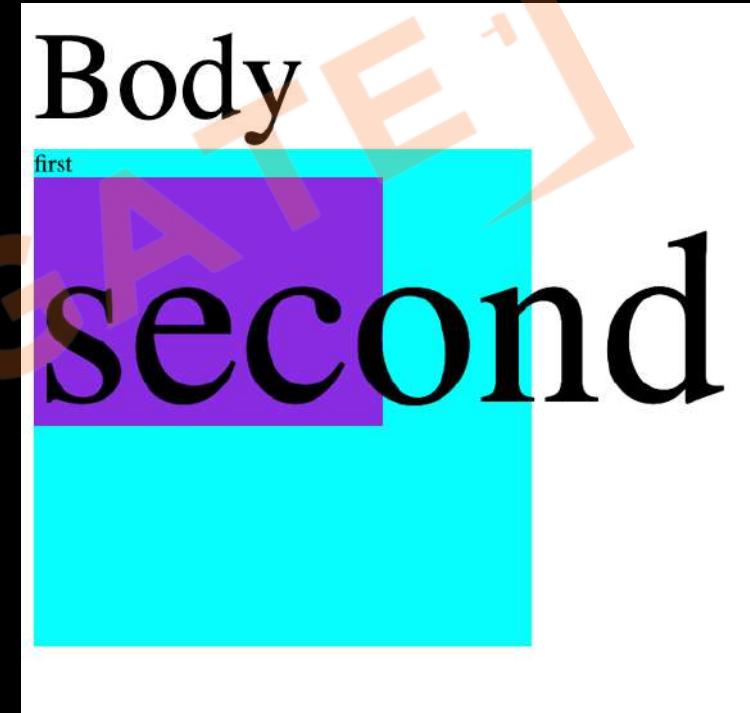


- **Relative Unit:** Sized relative to the parent element's font size.
- **Scalability:** Facilitates easy scaling of elements for responsive design.
- **Font Sizing:** Commonly used for setting font sizes adaptively.

Relative Units (REM)

```
* {  
    font-size: 50px;  
}  
  
#first {  
    height: 200px;  
    width: 200px;  
    background-color: #aqua;  
    font-size: 10px;  
}  
  
#second {  
    background-color: #blueviolet;  
    width: 70%;  
    height: 50%;  
    font-size: 2rem;  
}
```

```
<body>  
    Body  
    <div id="first">  
        first  
        <div id="second">  
            |   second  
            </div>  
        </div>  
    </body>
```



- **Relative Sizing:** Facilitates dynamic sizing **relative to root element**.
- **Adaptability:** Ensures **responsiveness** across various screens.
- **Dimensions:** Quickly set width and height as a percentage.

Relative Units (vw/vH)

```
<head>
  <title>Unit VH and VW</title>
  <style>
    #first {
      height: 50vh;
      width: 90vw;
      background-color: red;
    }
  </style>
</head>
<body>
  <div id="first"></div>
</body>
```



- **Viewport Relative Units:** Units based on **viewport's width (vw)** or **height (vh)** for responsive design.
- **Responsive Layouts:** Essential for creating adaptive layouts; e.g., `height: 100vh` for full-screen sections.
- **Element Sizing:** Useful for defining heights and widths that **scale**

Position Property

STATIC



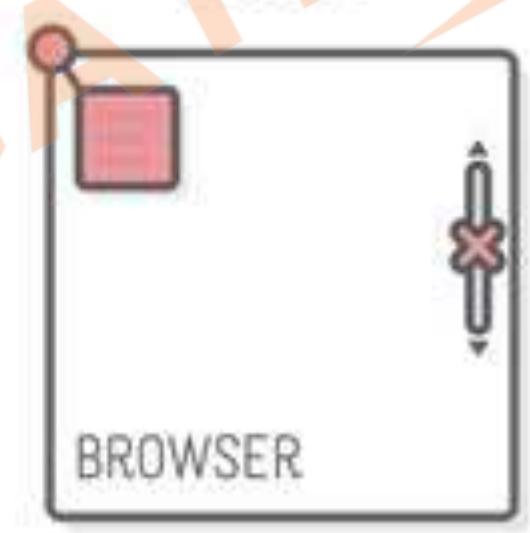
RELATIVE



ABSOLUTE



FIXED

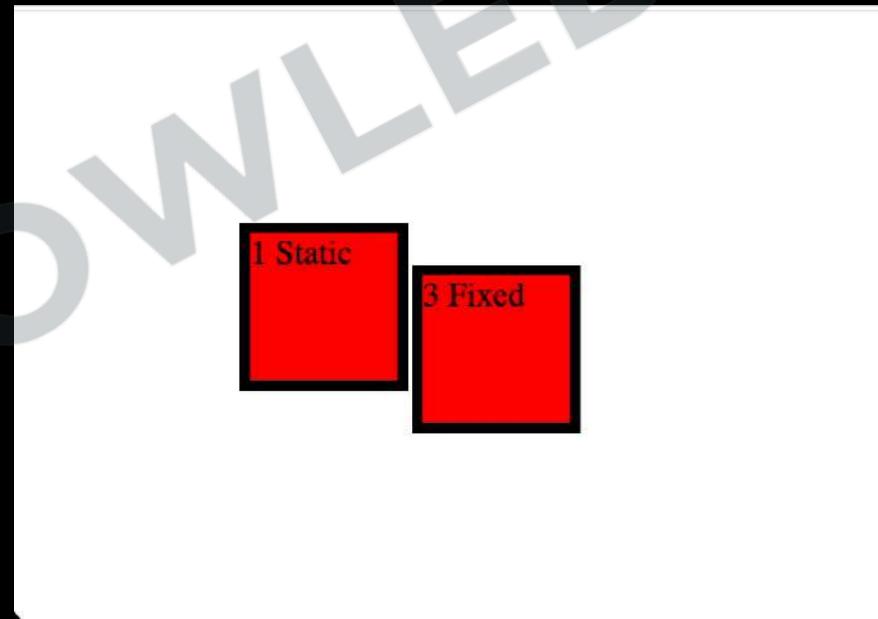


- **Static (default)** : Elements follow the normal document flow. (`top`, `right`, `bottom`, `left`, `z-index` would not work)
- **Relative**: Element's position adjusted from its normal position.
- **Absolute**: Positions element relative to the nearest positioned ancestor.
- **Fixed**: Element positioned relative to the `viewport`, does not move on scroll.

Position Property

```
div {  
    height: 70px;  
    width: 70px;  
    background-color: red;  
    border: 5px solid black;  
    margin: 20px;  
}  
  
#div1 {  
    position: static;  
}  
  
#div2 {  
    position: relative;  
    top: 20px;  
    left: 90px;  
}  
  
#div3 {  
    position: fixed;  
    top: 20px;  
    left: 90px;  
}  
  
#div4 {  
    position: absolute;  
    top: 200px;  
    left: 200px;  
}
```

```
<head>  
    <title>Position</title>  
    <link rel="stylesheet" href="../../  
        css/level 5/positions.css">  
</head>  
<body>  
    <div id="div1">1 Static</div>  
    <div id="div2">2 Relative</div>  
    <div id="div3">3 Fixed</div>  
    <div id="div4">4 Absolute</div>  
</body>
```

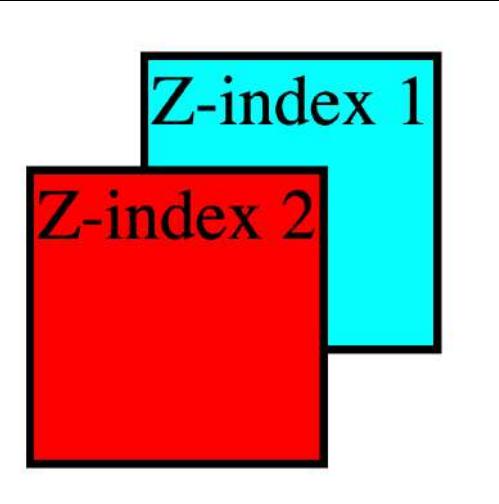


Position Property (z index)

```
.container {  
    position: relative;  
}  
  
.box1, .box2 {  
    position: absolute;  
    border: 3px solid black;  
    width: 100px;  
    height: 100px;  
    text-align: center;  
    font-size: 25px;  
}  
  
.box1 {  
    background-color: red;  
    left: 20px;  
    top: 60px;  
    z-index: 2;  
}  
  
.box2 {  
    background-color: aqua;  
    left: 60px;  
    top: 20px;  
    z-index: 1;  
}
```

- **Stacking Order:** Determines the **stacking order** of elements along the Z-axis.
- **Position Context:** Only applies to elements with position set to relative, absolute, fixed, or sticky.
- **Integer Values:** Accepts **integer values**, including negative numbers.
- **Higher Values:** An element with a **higher z-index value appears above others.**

```
<head>  
    <title>Z-Index</title>  
    <link rel="stylesheet" href="../css/  
        level 5/z-index.css">  
</head>  
<body>  
    <div class="container">  
        <div class="box1">Z-index 2</div>  
        <div class="box2">Z-index 1</div>  
    </div>  
</body>
```

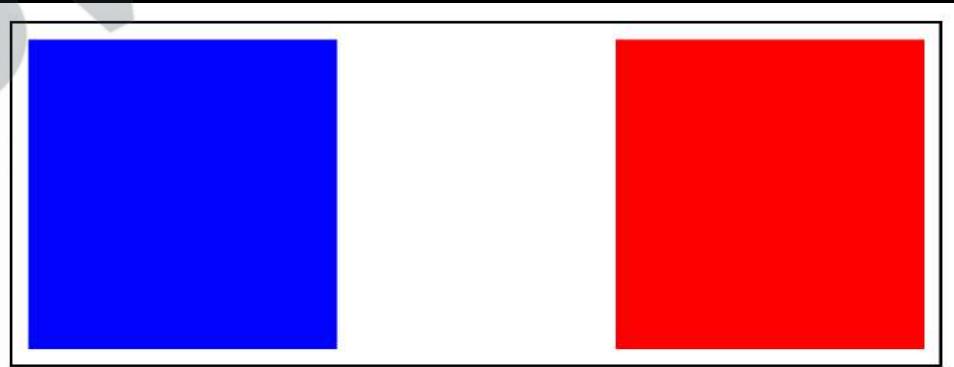


Float Property

```
.container {  
    height: 110px;  
    width: 300px;  
    border: 1px solid #000;  
}  
  
.box {  
    width: 100px;  
    height: 100px;  
    margin: 5px;  
}  
  
.box1 {  
    background-color: red;  
    float: right;  
}  
  
.box2 {  
    background-color: blue;  
    float: left;  
}
```

- **Element Alignment:** Allows elements to be aligned to the left or right within their containing element.
- **Values:** Can take values like "left", "right", or "none" to determine the floating direction.
- **Old Layout Technique:** Less commonly used with the advent of Flexbox.

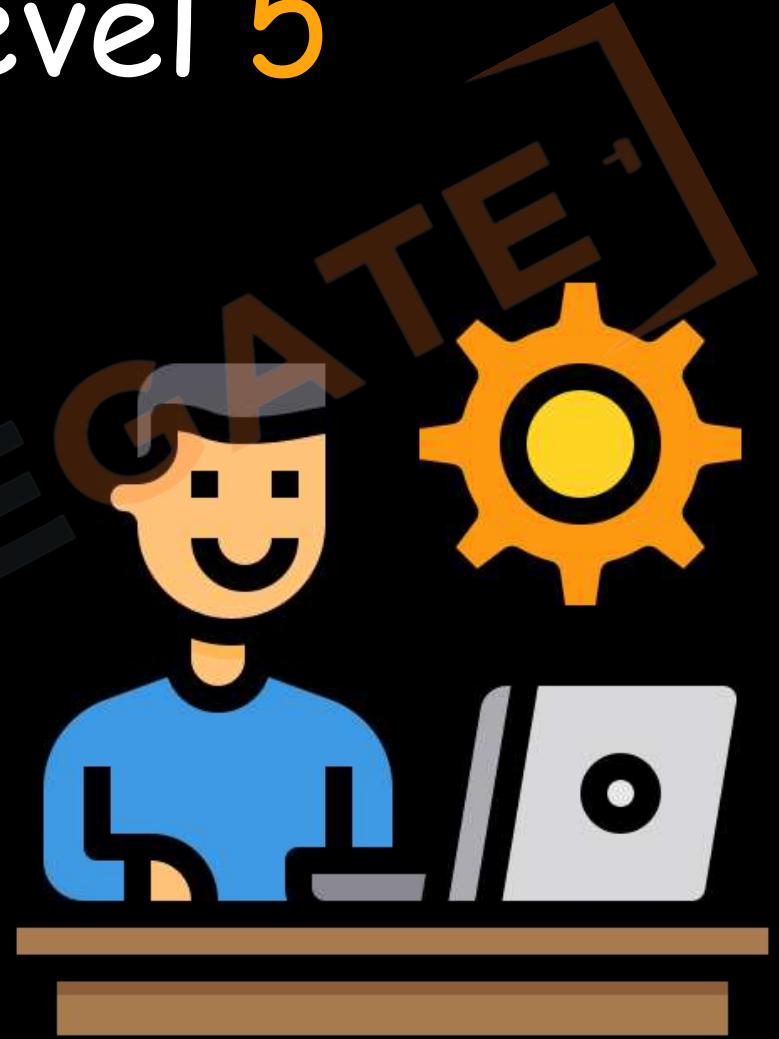
```
<head>  
    <title>Float Property</title>  
    <link rel="stylesheet" href="../../css/level 6/float.css">  
</head>  
<body>  
    <div class="container">  
        <div class="box box1"></div>  
        <div class="box box2"></div>  
    </div>  
</body>
```



Practice Set Level 5

Display and Position

- Create a webpage with header, footer, and a content area.
 - Header
 - Create a nav bar with links.
 - Main
 - Create a div with width and height, background green and border radius 50%
 - Create Three divs with container height and width 100px. Display inline block.
 - Set the correct position property for the single div element to ensure it remains at the right side of the page and does not shift when scrolling.
 - Use z-index to place the div on top of another div.
 - Footer
 - Add text in footer.





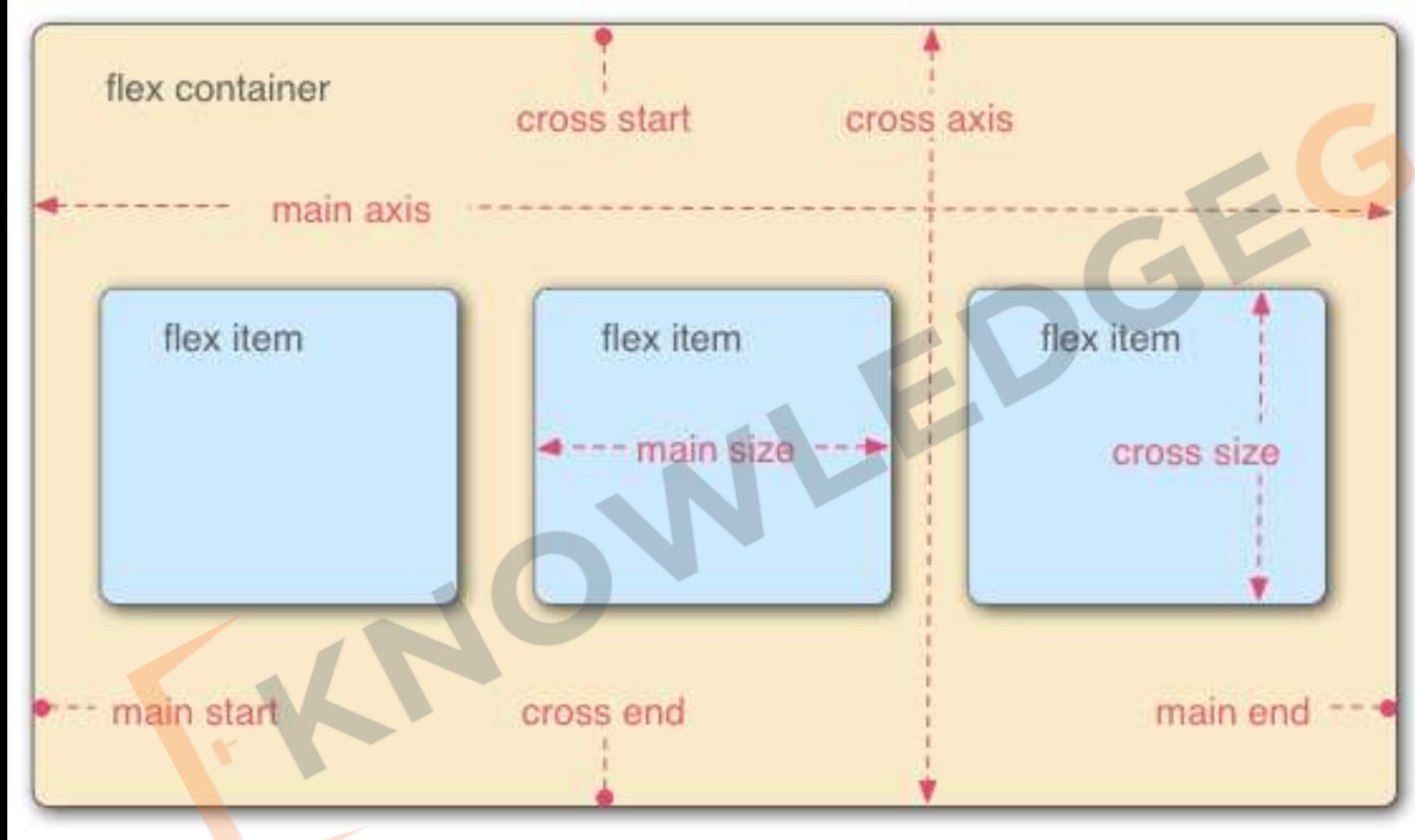
What is Flexbox?

Flexbox is a one-dimensional layout method for arranging items in rows or columns.

Items *flex* (expand) to fill additional space or shrink to fit into smaller spaces.



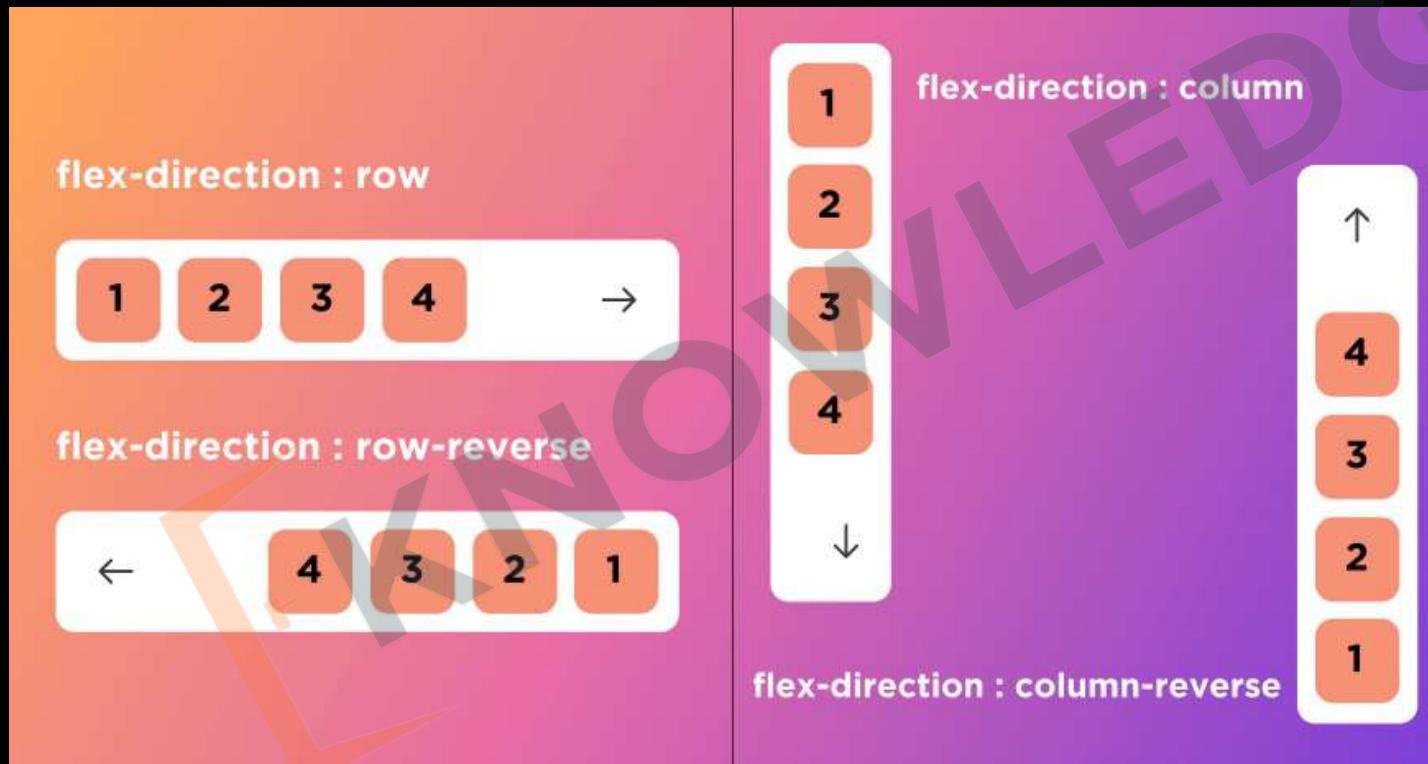
Flex Model



display: flex

Flexbox Direction

- **Property Name:** `flex-direction` is the property used to define the direction in a flex container.
- **Row Layout:** `row` value aligns the flex items **horizontally**, in a left-to-right fashion.
- **Column Layout:** `column` value stacks the flex items **vertically**, from top to bottom.
- **Reverse Direction:** Adding `-reverse` to row or column (as in `row-reverse` or `column-reverse`) reverses the order of the items.



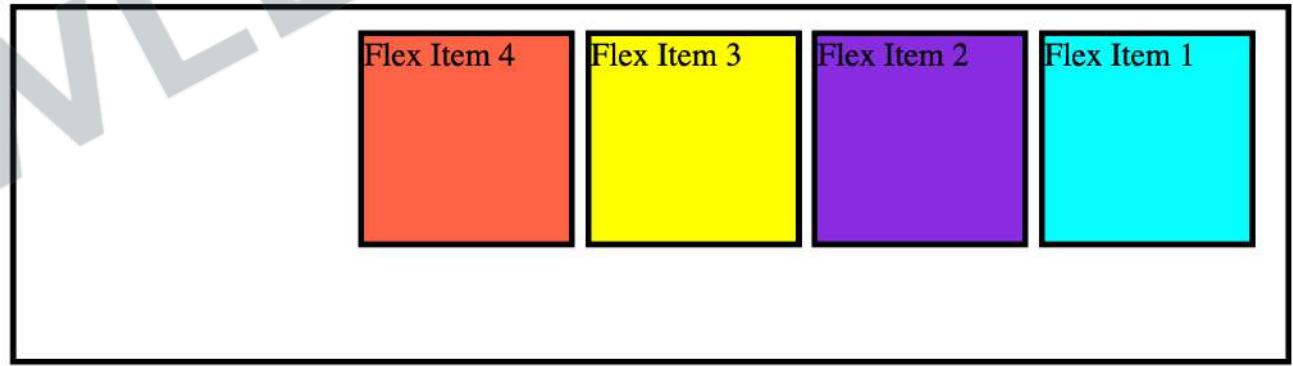
Flexbox Direction

```
* {  
    margin: 0;  
    padding: 0;  
}  
  
.box {  
    height: 100px;  
    width: 100px;  
    border: 3px solid black;  
    margin-right: 5px;  
}  
  
#heading {margin-left: 200px;}  
  
#container {  
    height: 150px;  
    width: 600px;  
    padding: 10px;  
    margin: 20px;  

```

```
<body>  
    <h1 id="heading">Flex Container</h1>  
    <div id="container">  
        <div id="box1" class="box">Flex Item 1</div>  
        <div id="box2" class="box">Flex Item 2</div>  
        <div id="box3" class="box">Flex Item 3</div>  
        <div id="box4" class="box">Flex Item 4</div>  
    </div>  
</body>
```

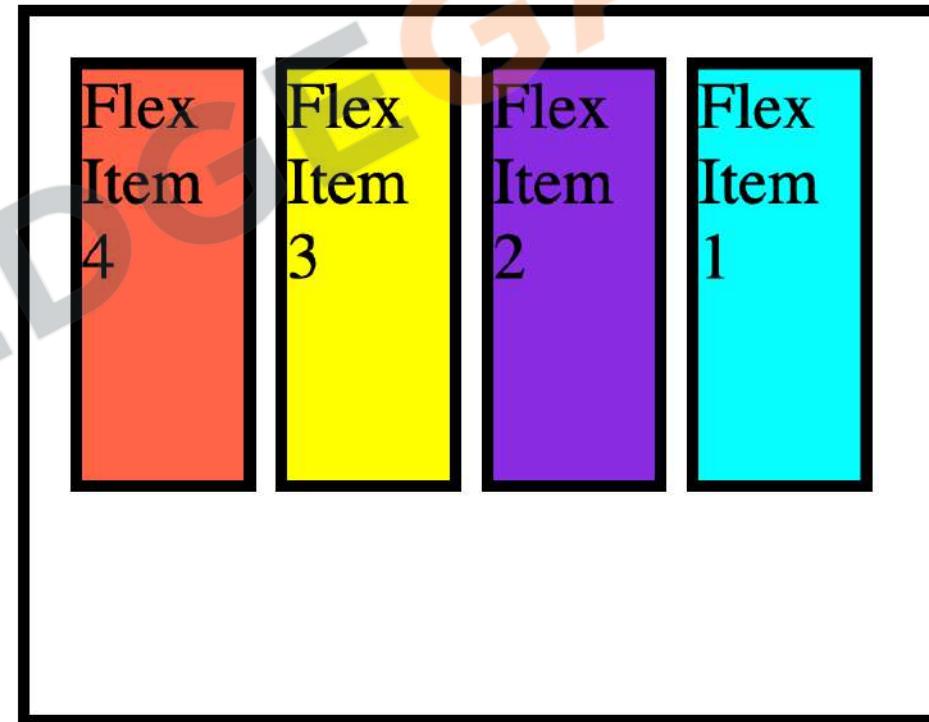
Flex Container



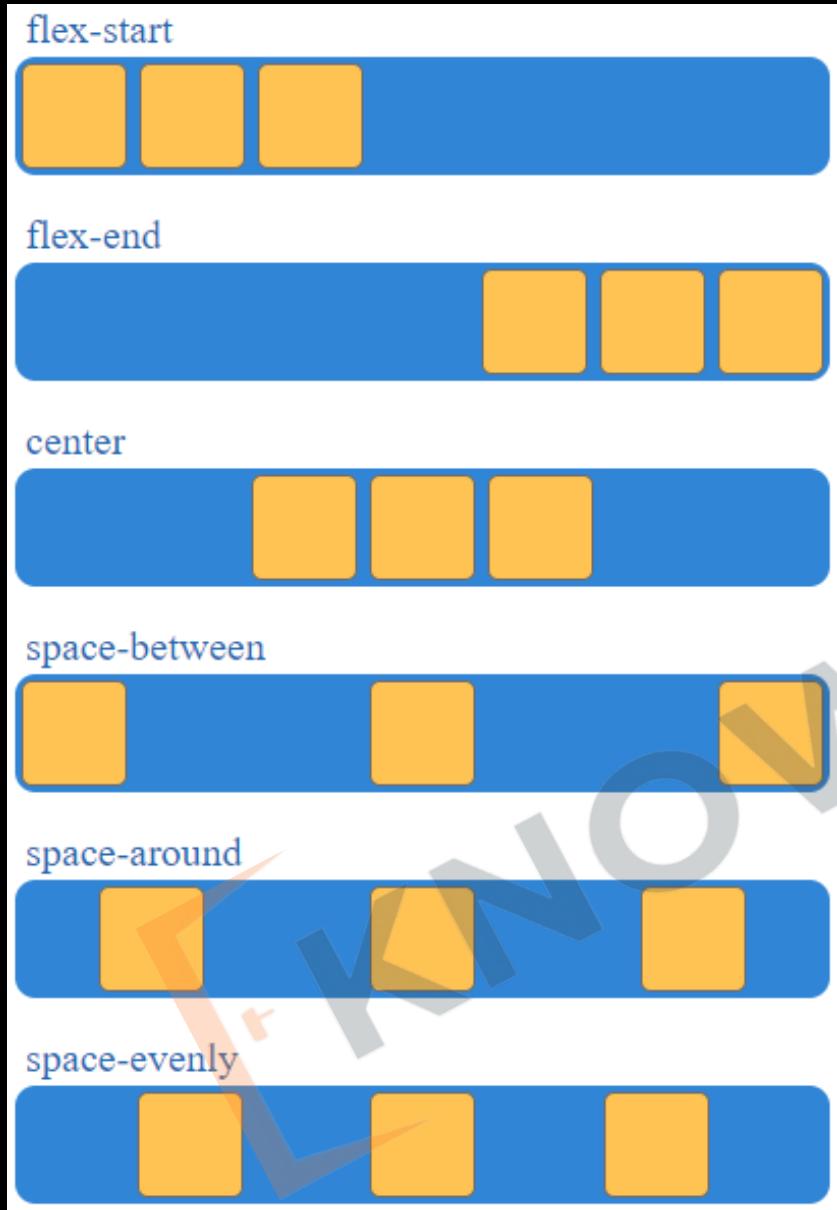
Flexbox Direction

```
.box {  
    height: 100px;  
    width: 100px;  
    border: 3px solid black;  
    margin-right: 5px;  
}  
  
#heading {margin-left: 50px;}  
  
#container {  
    height: 150px;  
    width: 200px; outline: 2px solid red;  
    padding: 10px;  
    margin: 20px;  
    border: 3px solid black;  
  
    display: flex;  
    flex-direction: row-reverse;  
}
```

Flex Container



Properties: Flexbox container (Justify Content)



- **Alignment**: Aligns flex items along the **main axis**.
- **flex-start**: Items align to the start of the flex container.
- **flex-end**: Items align to the end of the flex container.
- **Center**: Items are centered within the flex container.
- **space-between/space-around/space-evenly**: Distributes space between items evenly.

Properties: Flexbox container (Justify Content)

Flexbox Container

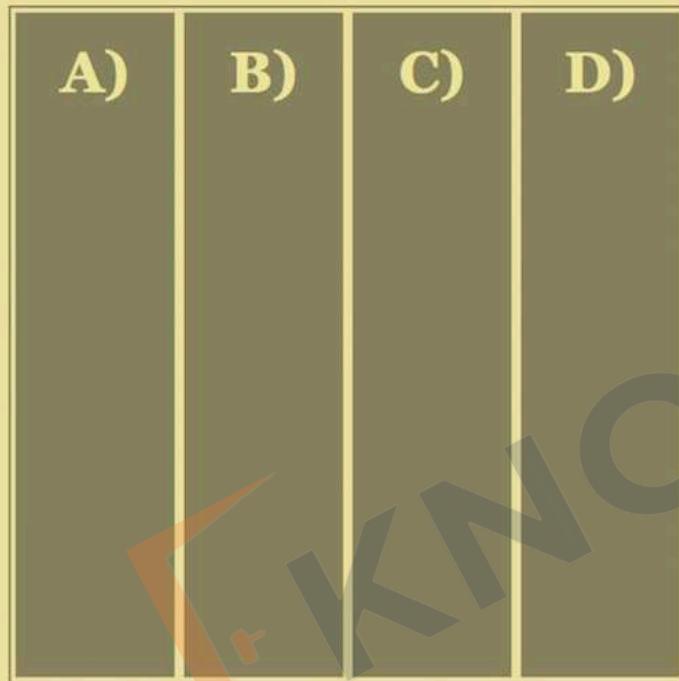


```
display: flex;  
justify-content: space-between;
```

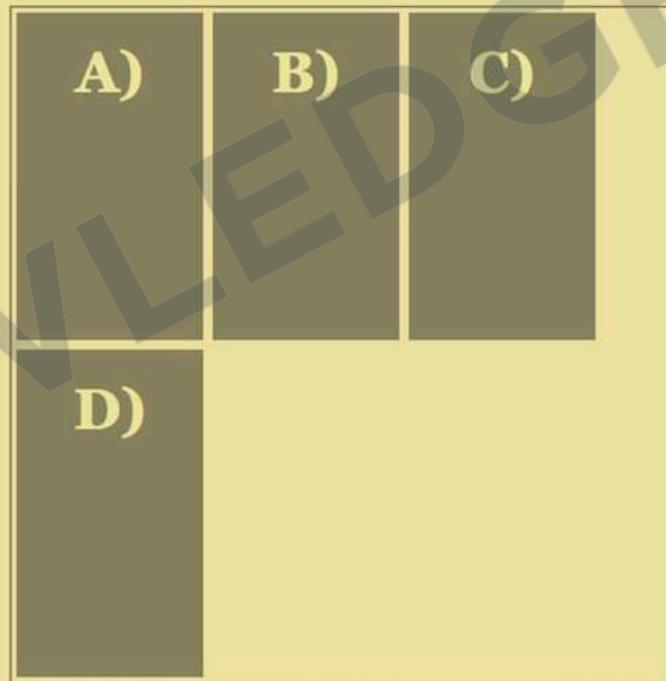
Properties: Flexbox container (Flex Wrap)

`flex-wrap:`

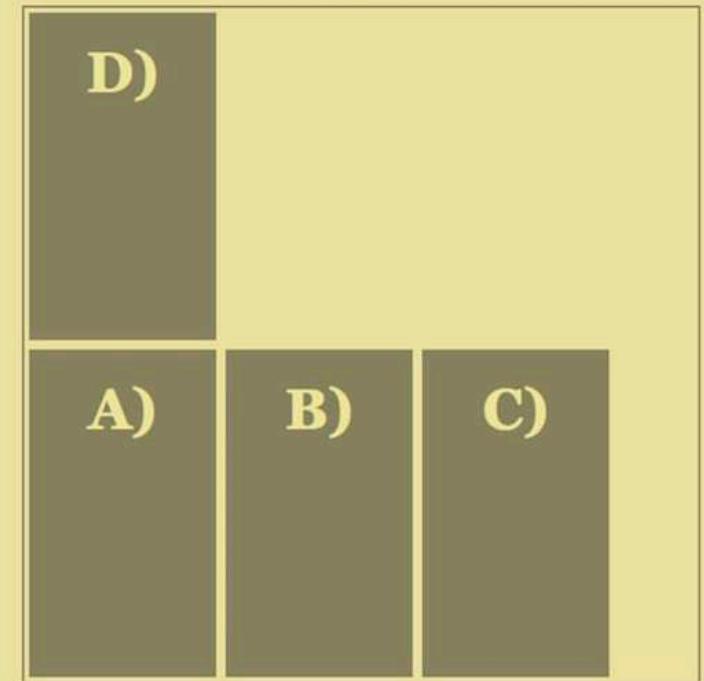
`nowrap` (*default*)



`wrap`



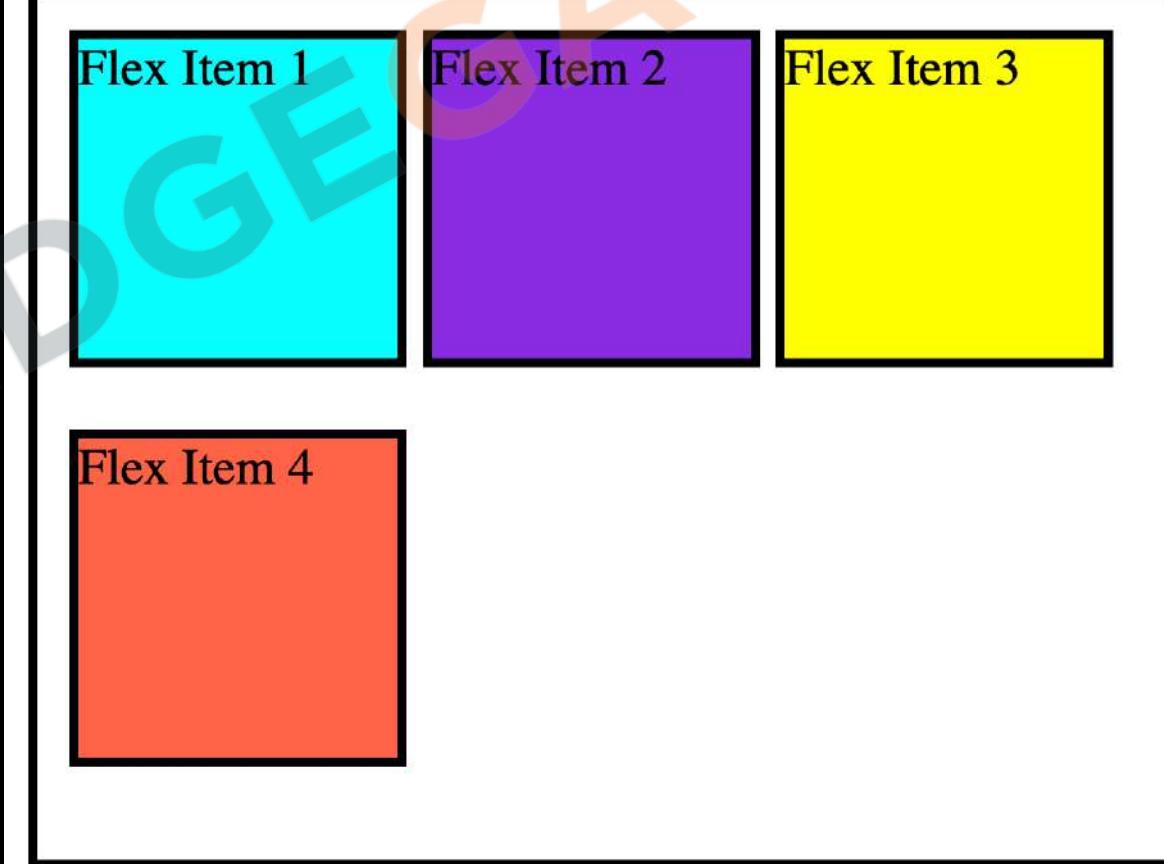
`wrap-reverse`



Properties: Flexbox container (Flex Wrap)

```
#container {  
    height: 250px;  
    width: 335px;  
    padding: 10px;  
    margin: 20px;  
    border: 3px solid black;  
  
    display: flex;  
    flex-wrap: wrap;  
}
```

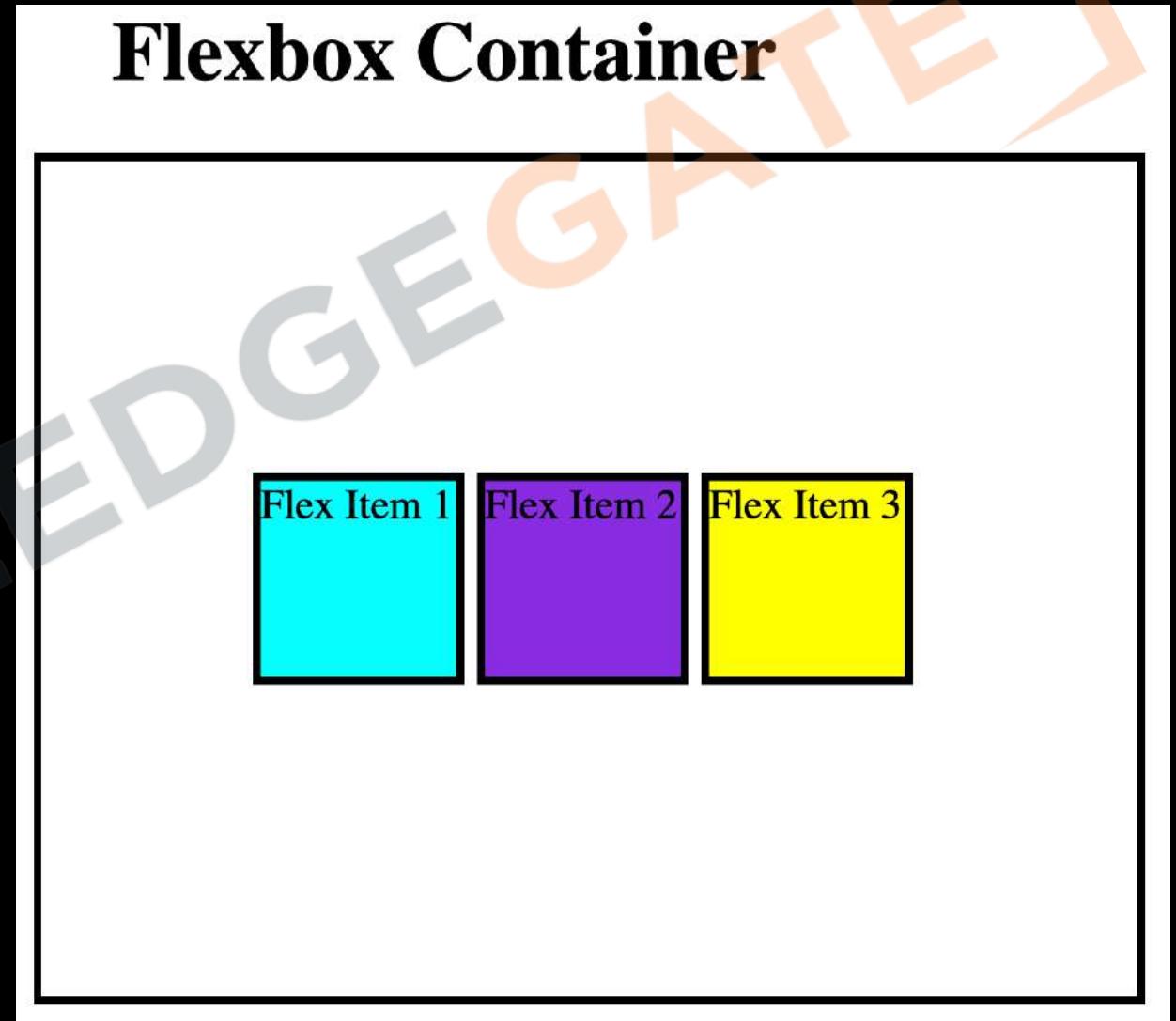
Flex Container



Properties: Flexbox container (Align Items)

This property is used to **align** the flex container's items **along the cross-axis**, which is perpendicular to the main axis.

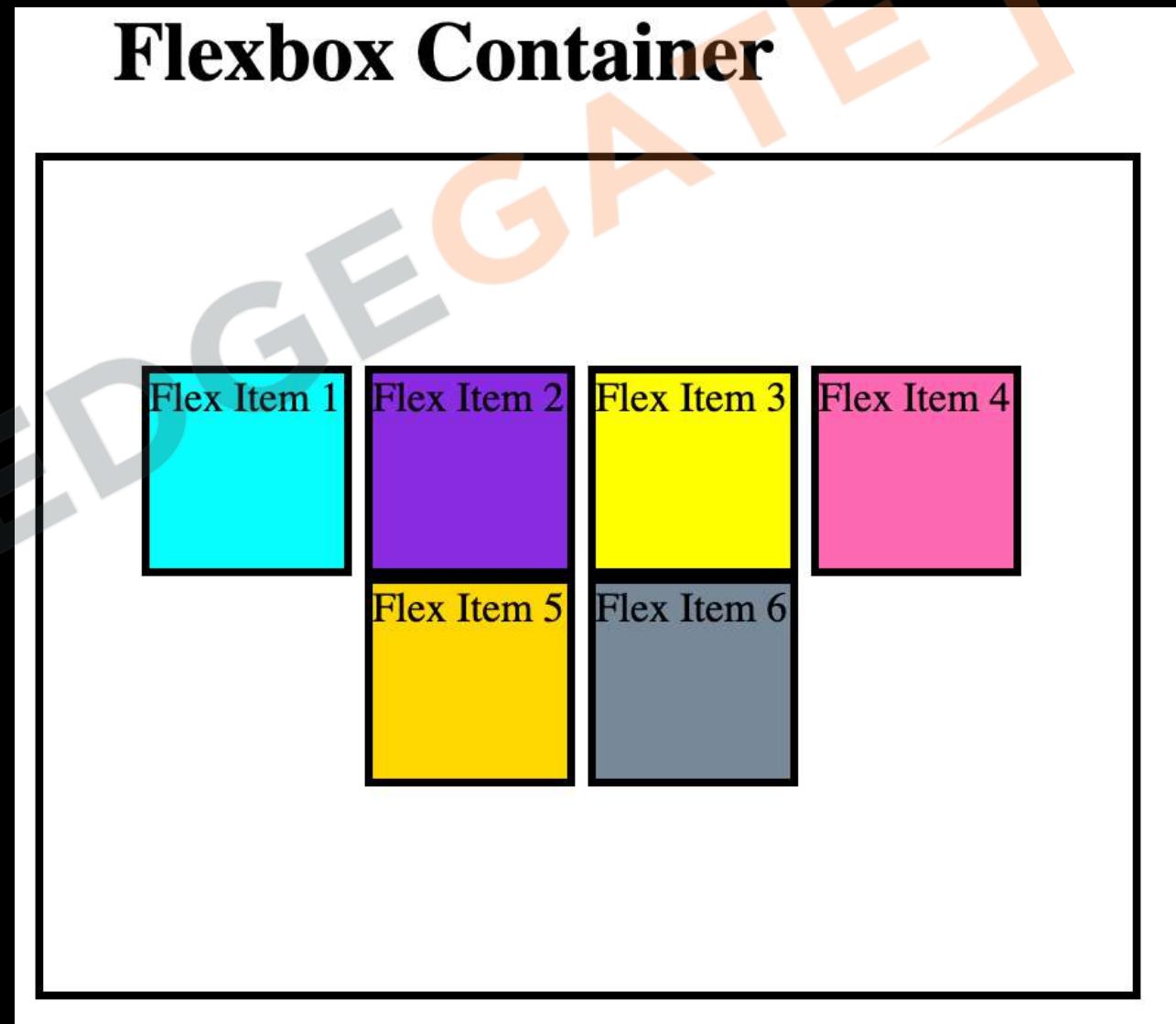
```
display: flex;  
flex-direction: row;  
justify-content: center;  
align-items: center;
```



Properties: Flexbox container (Align Content)

It is utilized to adjust the spacing between flex lines within a flex container, particularly when there is extra space along the **cross-axis**.

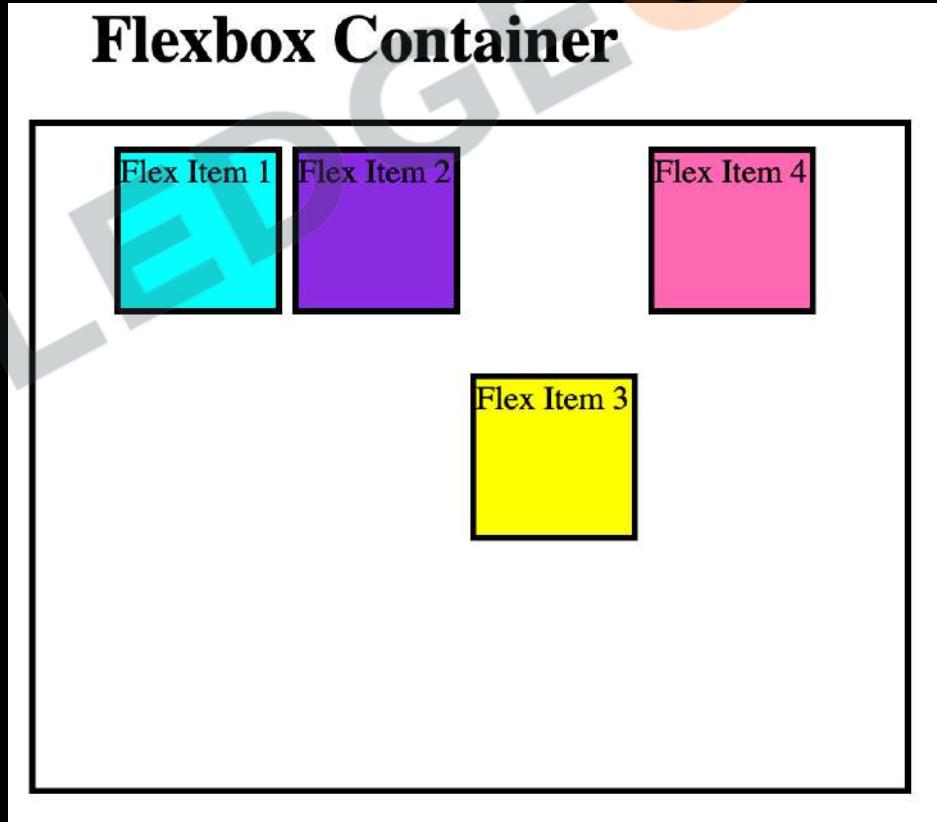
```
display: flex;  
flex-direction: row;  
justify-content: center;  
flex-wrap: wrap;  
align-content: center;
```



Properties: Flex Items (Align Self)

```
#container {  
    height: 300px;  
    width: 400px;  
    padding: 10px;  
    margin: 20px;  
    border: 3px solid black;  
  
    display: flex;  
    flex-direction: row;  
    justify-content: center;  
    flex-wrap: wrap;  
    align-items: start;  
}  
  
#box1 { background-color: aqua; }  
#box2 { background-color: blueviolet; }  
#box4 { background-color: hotpink; }  
#box3 {  
    background-color: yellow;  
    align-self: center;  
}
```

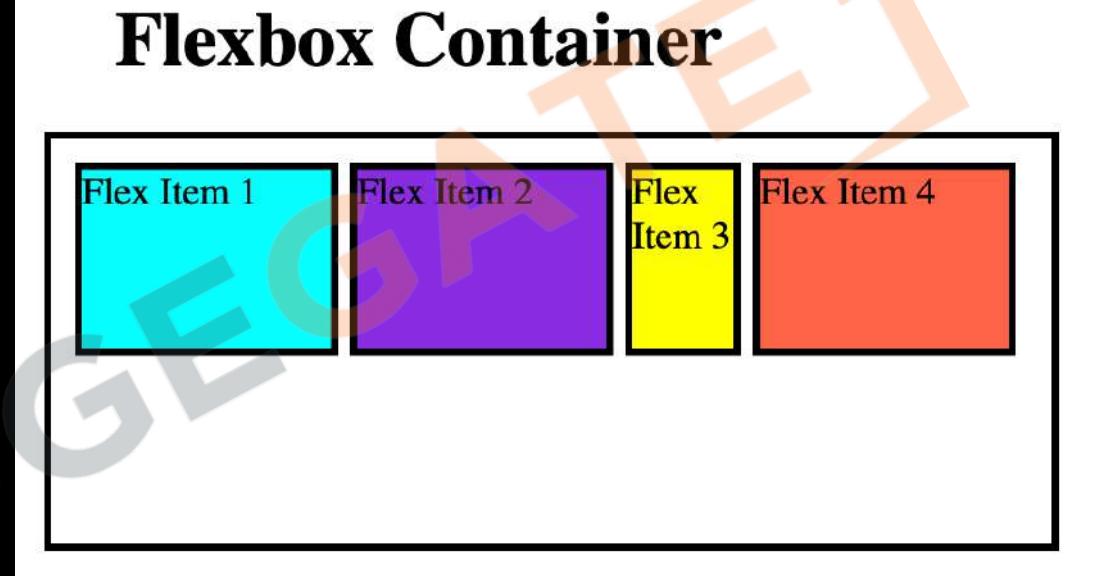
Allows individual flex items to override the container's align-items property, aligning them differently along the cross-axis.



Properties: Flex Items (Flex Shrink)

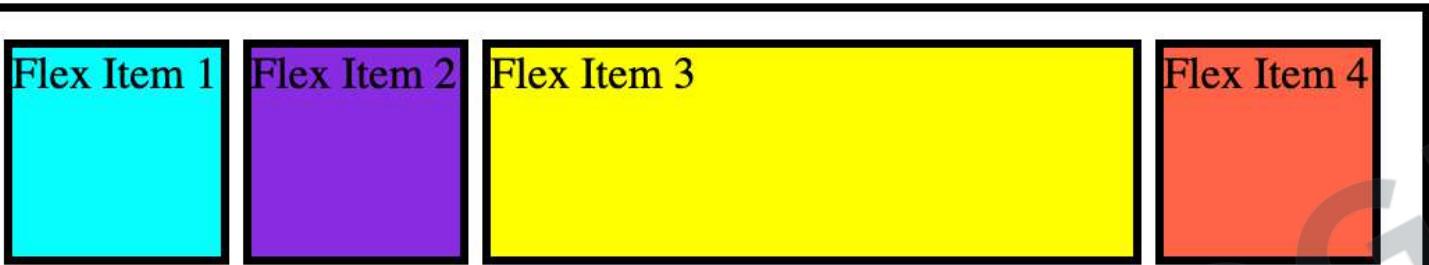
```
#box3 {  
    background-color: yellow;  
    flex-shrink: 4;  
}
```

The "flex-shrink" property in CSS determines how much a **flex item will shrink relative to other items** in the flex container if there is insufficient space.



Properties: Flex Items (Flex Grow)

Flexbox Container



```
#box3 {  
    background-color: yellow;  
    flex-grow: 1;  
}
```

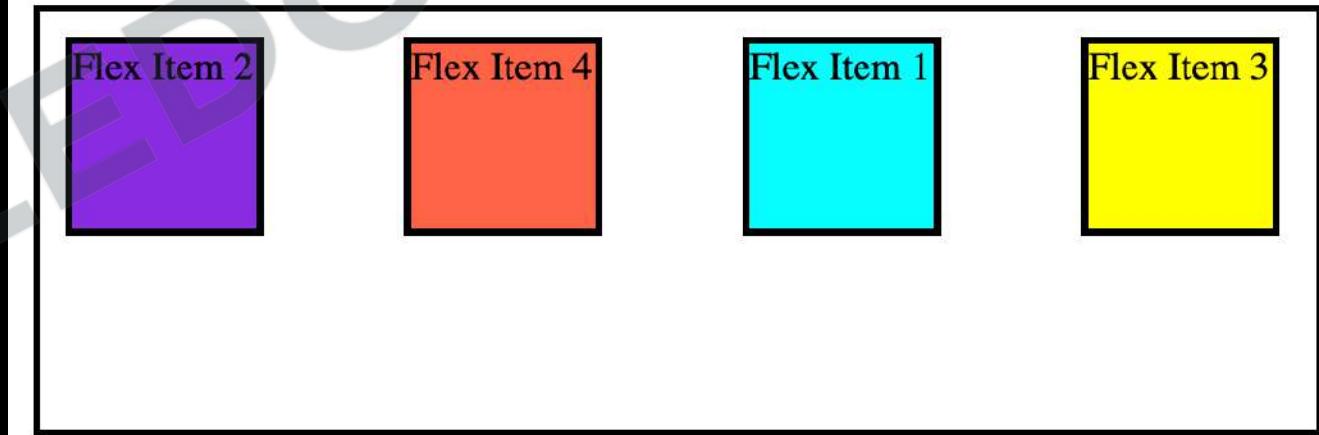
The "flex-grow" property in CSS specifies how much a **flex item will grow relative to other items** in the flex container when additional space is available.

Properties: Flex Items (Order)

```
#box1 {  
    background-color: #aqua;  
    order: 3  
}  
  
#box2 {  
    background-color: #blueviolet;  
    order: 1;  
}  
  
#box3 {  
    background-color: #yellow;  
    order: 4;  
}  
  
#box4 {  
    background-color: #tomato;  
    order: 2;  
}
```

The "order" property in CSS allows you to define the sequence in which flex items appear within the flex container, **overriding their original order** in the HTML.

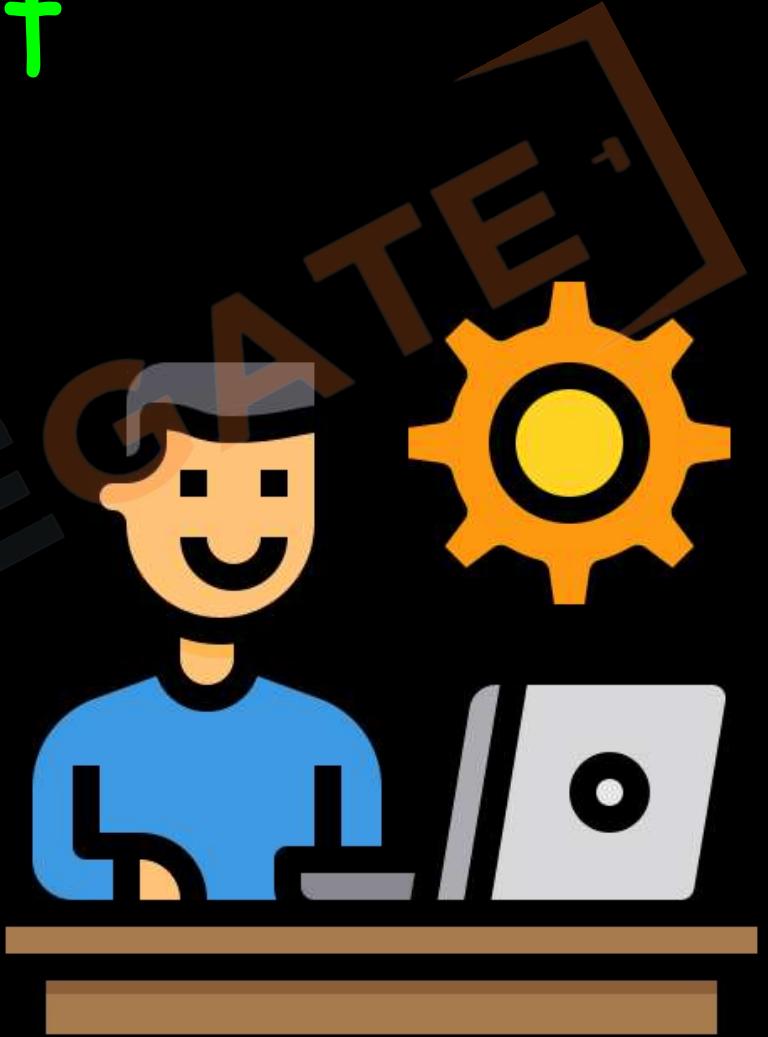
Flexbox Container



Practice Set

Flex Box

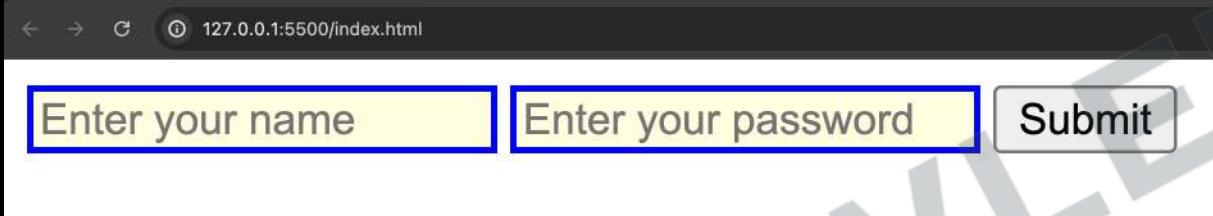
- Create the same nav bar created in the last Practice Set but do the spacing using flexbox.
- Use **flexbox** to centre an image inside a div.
- Create a div with three boxes where two boxes have fixed size, but the third box grows and shrinks with container size.





Selectors (Attribute selector)

```
<input type="text" placeholder="Enter  
your name" class="input-field" />  
<input type="password" placeholder="Enter  
your password" class="input-field" />  
<input type="submit" value="Submit" />
```



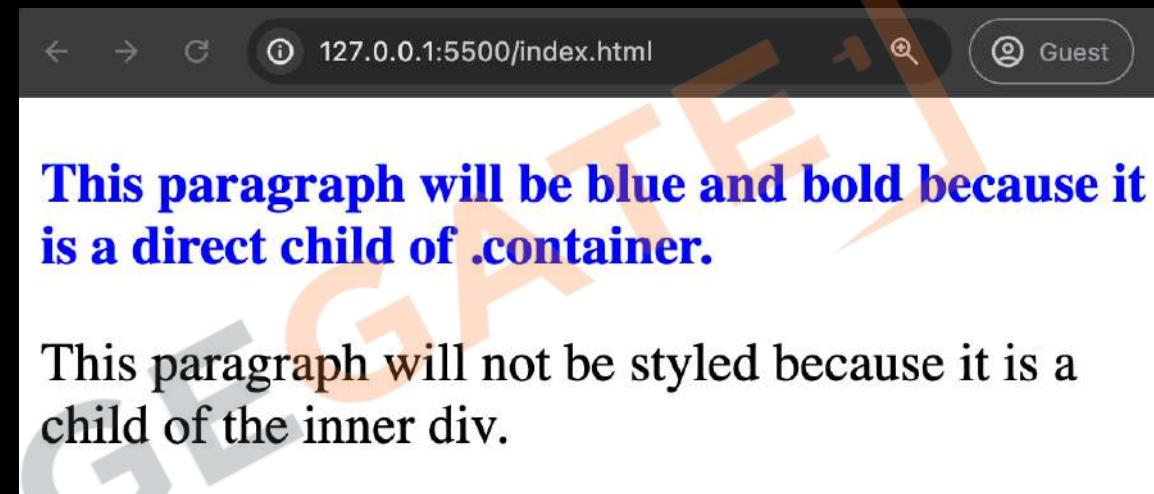
Syntax:

- Basic Attribute: [attribute]
- Exact match: [attribute="value"]
- Starts with: [attribute^="value"]
- Ends with: [attribute\$="value"]
- Contains: [attribute*="value"]

- Attribute selectors are used to select elements based on their attribute values.
- Attribute values are case-sensitive.
- Useful for selecting elements without adding additional classes or IDs.

Selectors (Child selector)

```
<head>
  <title>Child Selector Example</title>
  <style>
    /* Using child selector to style only direct children */
    .container > p {
      color: blue;
      font-weight: bold;
    }
  </style>
</head>
<body>
  <div class="container">
    <p>This paragraph will be blue and bold because it is a direct
       child of .container.</p>
    <div>
      <p>This paragraph will not be styled because it is a child of
         the inner div.</p>
    </div>
  </div>
</body>
```



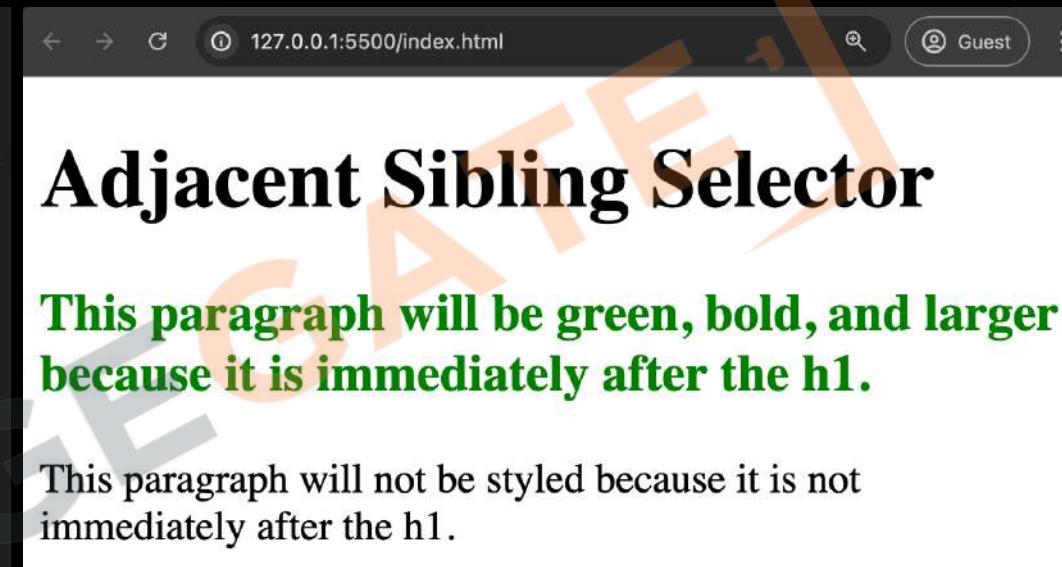
This paragraph will be blue and bold because it is a direct child of .container.

This paragraph will not be styled because it is a child of the inner div.

- The child selector in CSS is used to select only the **direct children** of a specified element.
- The child selector is denoted by the **>** symbol.
- It targets **immediate children** elements, not grandchildren or other descendants.
- The child selector is **more specific than the descendant selector** (space), which selects all descendants regardless of their depth in the hierarchy.

Selectors (Adjacent Sibling selector)

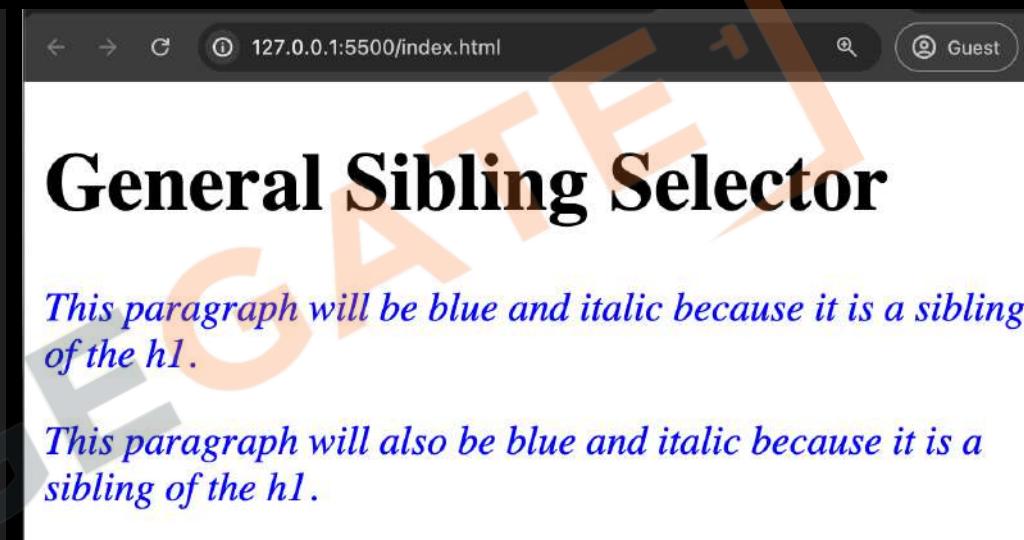
```
<head>
  <title>Adjacent Sibling Selector Example</title>
  <style>
    /* Using adjacent sibling selector to style the immediate sibling */
    h1 + p {
      color: green;
      font-size: 20px;
      font-weight: bold;
    }
  </style>
</head>
<body>
  <h1>Adjacent Sibling Selector</h1>
  <p>This paragraph will be green, bold, and larger because it is
  immediately after the h1.</p>
  <p>This paragraph will not be styled because it is not immediately
  after the h1.</p>
</body>
```



- The adjacent sibling selector in CSS selects an element that is **immediately preceded** by a specified element.
- It targets the element that comes **directly after** the specified element.
- The adjacent sibling selector is specific to the immediate following sibling and does not affect any other siblings.

Selectors (General Sibling selector)

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>General Sibling Selector Example</title>
  <style>
    /* Using general sibling selector to style all following siblings */
    h1 ~ p {
      color: blue;
      font-style: italic;
    }
  </style>
</head>
<body>
  <h1>General Sibling Selector</h1>
  <p>This paragraph will be blue and italic because it is a sibling of the h1.</p>
  <p>This paragraph will also be blue and italic because it is a sibling of the h1.</p>
</body>
```

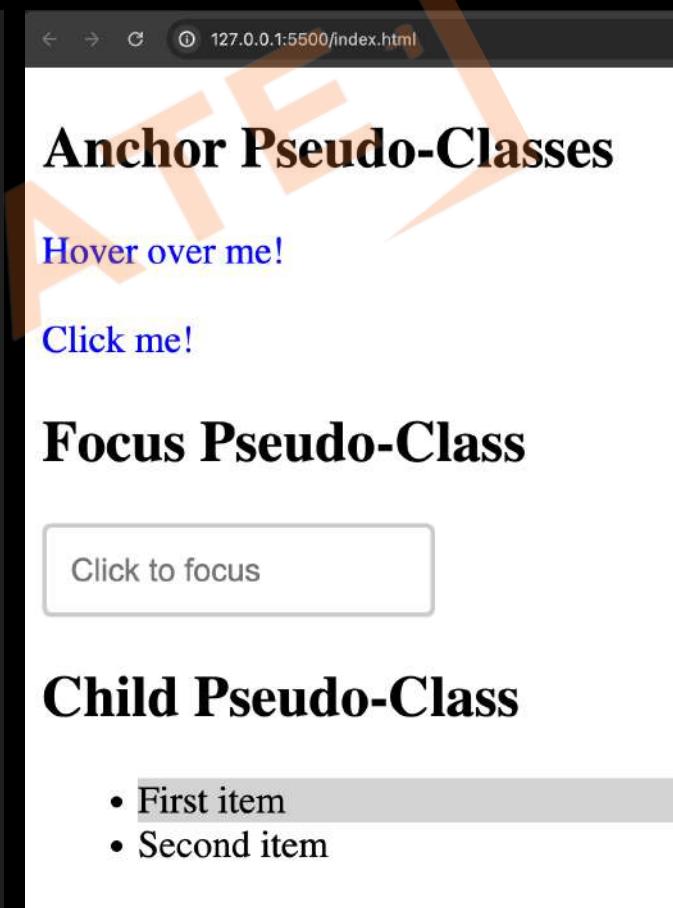


- The **general sibling selector** in CSS selects all elements that are siblings of a specified element, regardless of their position.
- It targets **all siblings** that follow the specified element, not just the immediate one.
- Currently, there is **no direct CSS selector** to target **previous siblings**.

Pseudo Classes

- Pseudo-classes are keywords added to selectors that specify a special state of the selected elements.
- They help to style elements based on their state, such as when an element is hovered over or when a form input is checked.
- Syntax: **selector:pseudo-class { styles }**.
- Common examples: **:hover, :active, :focus, :first-child**.

```
<title>Pseudo-Classes Example</title>
<style>
  a {
    color: blue;
    text-decoration: none;
  }
  a:hover {
    color: red;
  }
  a:active {
    color: green;
  }
  input {
    padding: 10px;
    border: 2px solid #ccc;
    border-radius: 4px;
    outline: none; /* Remove default focus outline */
  }
  input:focus {
    border-color: orange;
    box-shadow: 0 0 5px orange;
  }
  ul li:first-child {
    background-color: lightgray;
  }
</style>
```



Practice Set

Advance Selectors

```
1 <div class="store">
2   <header>
3     <h1>Welcome to MyStore</h1>
4   <nav>
5     <ul class="main-nav">
6       <li>Home</li>
7       <li>About</li>
8       <li>
9         Products
10        <ul>
11          <li>Electronics</li>
12          <li>Books</li>
13        </ul>
14      </li>
15      <li>Contact</li>
16    </ul>
17  </nav>
18 </header>
```

```
19 <section>
20   <h2>Featured Products</h2>
21   <div class="product" data-category="electronics">
22     
23     <p>Description of the product goes here.</p>
24     <button type="submit">Buy Now</button>
25   </div>
26   <div class="product" data-category="books">
27     
28     <p>Description of another product.</p>
29     <button type="submit">Buy Now</button>
30   </div>
31 </section>
32 <footer>
33   <p>Copyright 2023 by MyStore. All rights reserved.</p>
34   <p>Follow us on <a href="#" class="social">Social Media</a></p>
35 </footer>
36 </div>
```

Practice Set

Advance Selectors

1. Attribute Selector:

- Style all `div.product` elements with a `border: 1px solid gray;`.
- Highlight products where the `data-category` attribute contains “electronics” with a `background-color: lightblue;`.

2. Child Selector:

- Style direct children `li` elements of `ul.main-nav` with `color: darkblue;`.
- Increase the left margin for the nested `ul` inside `li.Products` to `20px`.

3. Adjacent Sibling Selector:

- Style the paragraph immediately following the `h2` tag in the section with `font-weight: bold;`.

4. General Sibling Selector:

- Apply a `font-size: 12px;` to all paragraphs that are siblings of an `img` element within `div.product`.

5. Pseudo-Classes:

- Style the button when hovered over with `background-color: green; color: white;`.
- Style the first `li` in `ul.main-nav` to be `font-weight: bold;`.

Overflow Property

- **Purpose:** Controls how content is handled when it overflows an element's box.
- **visible:** Default; content is not clipped and may overflow the element's box.
- **hidden:** Content is clipped and not visible beyond the element's box.
- **scroll:** Content is clipped, but scrollbars are added to allow scrolling.
- **auto:** Scrollbars are added only when necessary to see the overflowing content.

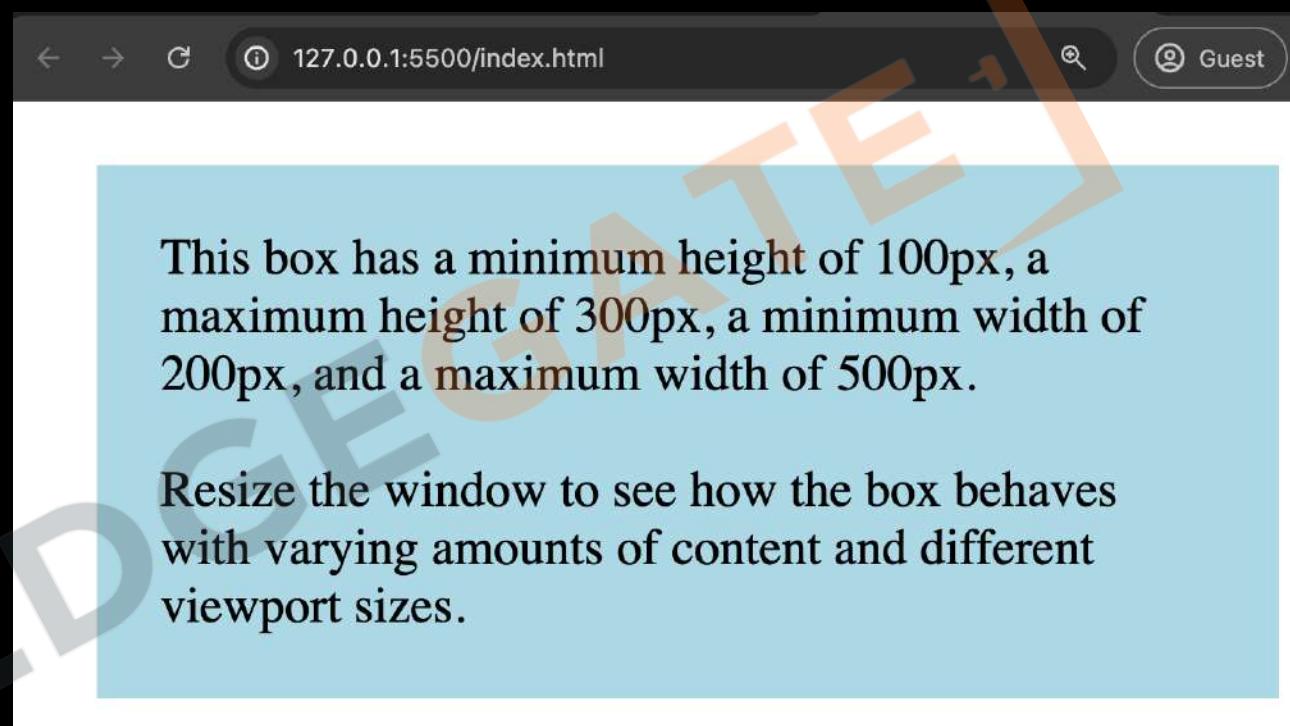
```
<style>
  .container {
    width: 240px;
    height: 120px;
    border: 1px solid black;
    margin: 20px;
  }
  .visible {
    overflow: visible;
    background-color: lightcoral;
  }
  .hidden {
    overflow: hidden;
    background-color: lightgreen;
  }
  .scroll {
    overflow: scroll;
    background-color: lightblue;
  }
  .auto {
    overflow: auto;
    background-color: lightyellow;
  }
</style>
```

The screenshot shows a browser window with four separate div elements, each demonstrating a different value for the overflow property:

- visible:** The first div has a red background and contains text that extends beyond its container boundaries.
- hidden:** The second div has a green background and contains text that is completely clipped by the container's boundaries.
- scroll:** The third div has a blue background and contains text, with a vertical scrollbar appearing on the right side to allow viewing of the overflowed content.
- auto:** The fourth div has a yellow background and contains text, with a vertical scrollbar appearing on the right side only if the content exceeds the container's height.

Min-Max Height and Width

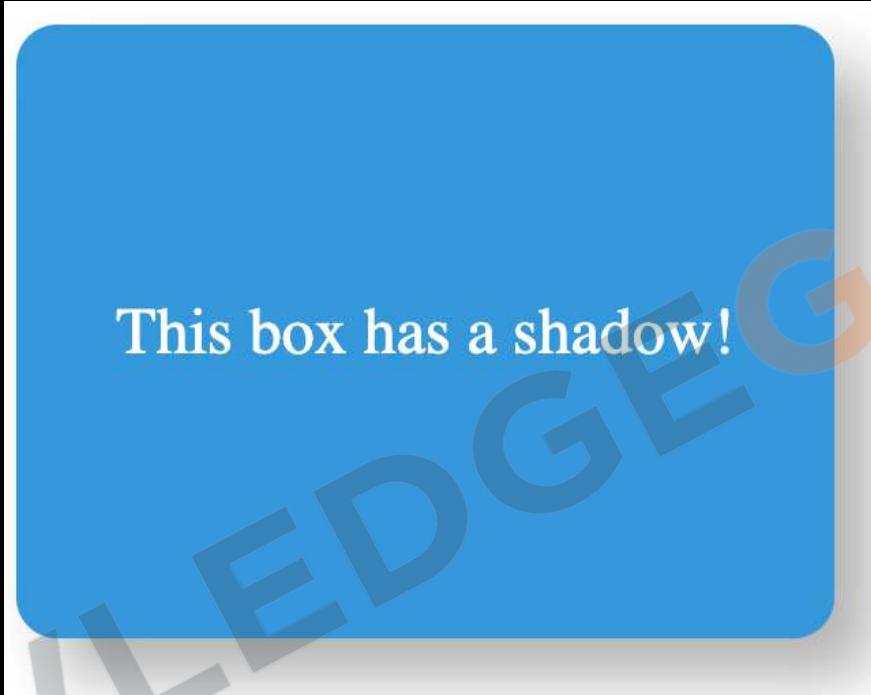
```
<head>
  <title>Min/Max Height and Width Example</title>
  <style>
    .box {
      background-color: lightblue;
      min-height: 100px;
      max-height: 300px;
      min-width: 200px;
      max-width: 500px;
      overflow: auto; /* To handle overflow content */
      padding: 20px;
      margin: 20px;
    }
  </style>
</head>
<body>
  <div class="box">
    This box has a minimum height of 100px, a maximum height of 300px,
    a minimum width of 200px, and a maximum width of 500px.
    <br><br>
    Resize the window to see how the box behaves with varying amounts
    of content and different viewport sizes.
  </div>
</body>
```



- **min-height:** Sets the minimum height an element can be.
- **max-height:** Sets the maximum height an element can be.
- **min-width:** Sets the minimum width an element can be.
- **max-width:** Sets the maximum width an element can be.

Box-Shadow Property

```
<head>
  <title>Box-Shadow Example</title>
  <style>
    .box {
      width: 200px;
      height: 150px;
      background-color: #3498db;
      color: white;
      display: flex;
      align-items: center;
      justify-content: center;
      border-radius: 10px;
      box-shadow: 10px 10px 15px rgba(0, 0, 0, 0.3);
    }
  </style>
</head>
<body>
  <div class="box">
    This box has a shadow!
  </div>
</body>
```



The **box-shadow** property in CSS adds shadow effects around an element's frame.

Syntax:

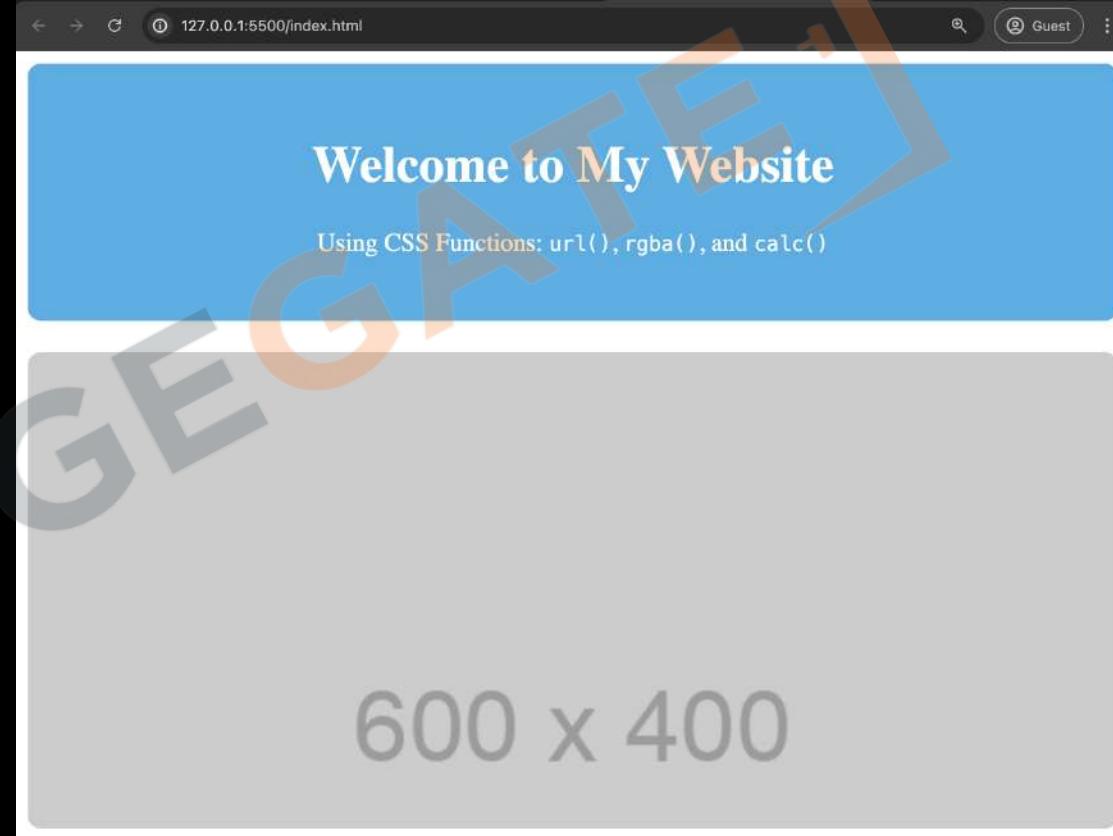
- **box-shadow: h-offset v-offset blur spread color;**
- **h-offset:** Horizontal offset of the shadow.
- **v-offset:** Vertical offset of the shadow.
- **blur:** Optional. Blur radius of the shadow.
- **spread:** Optional. Spread radius of the shadow.
- **color:** Color of the shadow.

CSS Functions

```
<head>
  <title>CSS Functions Example</title>
  <style>
    .header {
      background-color: rgba(52, 152, 219, 0.8); /* rgba function */
      padding: calc(10px + 2vw); /* calc function */
      color: white;
      text-align: center;
      border-radius: 8px;
    }

    .content {
      background-image: url('https://via.placeholder.com/600x400'); /* url function */
      /*
      background-size: cover;
      height: 300px;
      margin-top: 20px;
      border-radius: 8px;
    }
  </style>
</head>
<body>
  <div class="header">
    <h1>Welcome to My Website</h1>
    <p>Using CSS Functions: <code>url()</code>, <code>rgba()</code>, and <code>calc()</code></p>
  </div>

  <div class="content">
  </div>
</body>
```



- CSS functions are **special constructs** that perform specific **operations or calculations** and return values that **can be used** in CSS properties.

CSS Variables

- CSS variables, also known as custom properties, are entities defined by CSS authors that contain specific values to be reused throughout a document.
- Declared using the -- prefix, e.g., --main-color: #3498db;.
- Accessed using the var() function, e.g., color: var(--main-color);.
- var() function can accept a fallback value if the variable is not defined, e.g., color: var(--secondary-color, #2ecc71);

```
<body>
  <h1>Welcome to My Website</h1>
  <p>This is an example of using CSS variables to manage colors and spacing.</p>

  <div class="card">
    <h2>Card Title</h2>
    <p>This card uses CSS variables for styling.</p>
  </div>

  <p class="fallback-example">This text uses a fallback value for an undefined variable.</p>
</body>
```



```
<title>CSS Variables Example</title>
<style>
  html {
    --main-bg-color: #3498db;
    --main-text-color: #ffffff;
    --secondary-bg-color: #2ecc71;
    --secondary-text-color: #ff6347;
    --padding: 10px;
    --border-radius: 5px;
  }

  body {
    background-color: var(--main-bg-color);
    color: var(--main-text-color);
    font-family: Arial, sans-serif;
    padding: var(--padding);
  }

  .card {
    background-color: var(--secondary-bg-color);
    color: var(--secondary-text-color);
    padding: var(--padding);
    border-radius: var(--border-radius);
    margin: var(--padding) 0;
  }

  .fallback-example {
    /* Uses fallback color */
    color: var(--undefined-variable, #ff6347);
  }
</style>
```

Practice Set

CSS Advanced

```
1 <body>
2   <header>
3     <h1>My Blog</h1>
4   </header>
5   <section class="content">
6     <article class="post">
7       <h2>First Post</h2>
8       <div class="post-content">
9         <p>
10        Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed do
11          eiusmod tempor incididunt ut labore et dolore magna aliqua.
12        </p>
13      </div>
14    </article>
15    <article class="post">
16      <h2>Second Post</h2>
17      <div class="post-content">
18        <p>
19        Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed do
20          eiusmod tempor incididunt ut labore et dolore magna aliqua.
21        </p>
22      </div>
23    </article>
24  </section>
25  <footer>
26    <p>© 2023 My Blog</p>
27  </footer>
28</body>
```

Practice Set

CSS Advanced

1. Overflow Property:

- Ensure that the .post-content divs have a maximum height of 100px and set the overflow to auto to handle any overflowing content.

2. Min-Max Height and Width:

- Set the header to have a minimum height of 80px and a maximum height of 150px.
- Ensure the section.content has a minimum width of 300px and a maximum width of 800px.

3. Box-Shadow Property:

- Apply a box shadow to the .post elements with the following properties: horizontal offset 5px, vertical offset 5px, blur radius 10px, spread radius 0, and color rgba(0, 0, 0, 0.1).

4. CSS Functions:

- Use the calc function to set the padding of the .post-content divs to calc(10px + 2%).

5. CSS Variables:

- Declare a CSS variable --main-bg-color with the value #f0f0f0.
- Apply this variable to the background color of the body.
- Declare another CSS variable --header-color with the value #333.
- Use this variable to set the color of the h1 in the header.

Media Queries

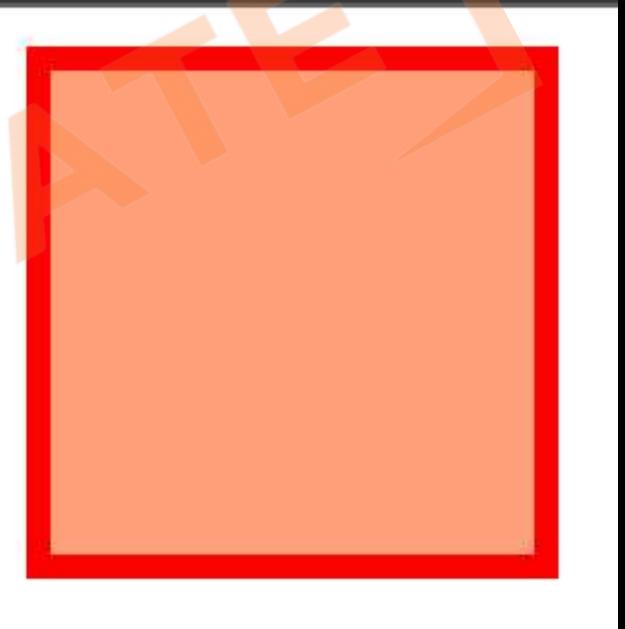
declaration



- Tailor styles for specific device characteristics.
- Use to create **responsive** web designs.
- Apply styles based on conditions like screen size.
- Syntax: `@media (condition) { CSS rules }`.
- Can **combine** multiple conditions using and, or.

Media Queries (width)

```
.box {  
    height: 100px;  
    width: 100px;  
    background-color: lightsalmon;  
}  
  
@media screen and (width: 250px) {  
    .box {  
        border: 5px solid red;  
    }  
}
```



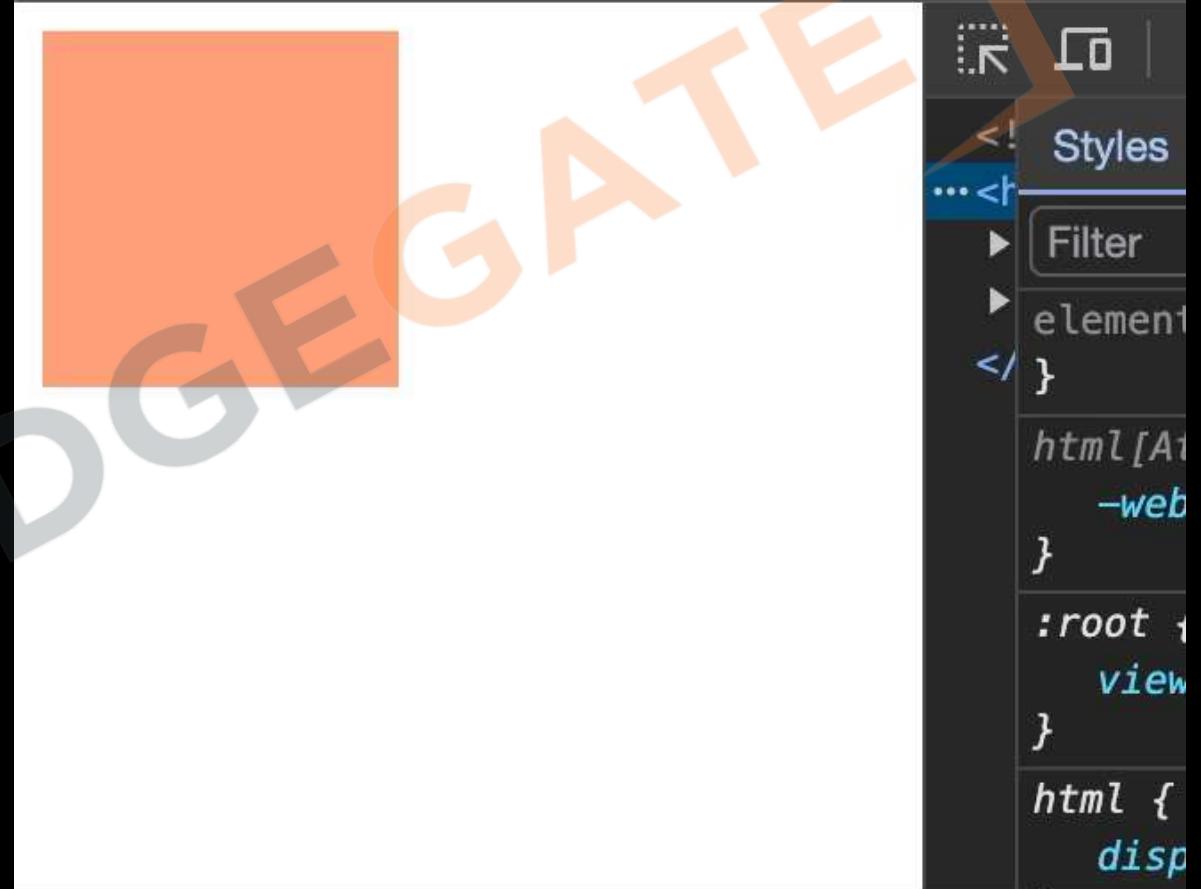
Media Queries (min-width)

```
.box {  
    height: 100px;  
    width: 100px;  
    background-color: lightcoral;  
}  
  
@media screen and (min-width: 300px) {  
    .box {  
        height: 150px;  
        width: 150px;  
    }  
}
```



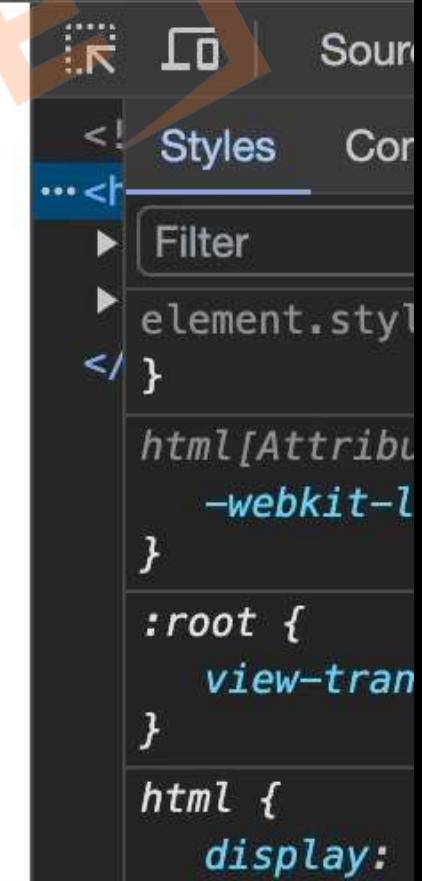
Media Queries (max-width)

```
.box {  
    height: 100px;  
    width: 100px;  
    background-color: lightsalmon;  
}  
  
@media screen and (max-width: 250px) {  
    .box {  
        height: 50px;  
        width: 50px;  
    }  
}
```



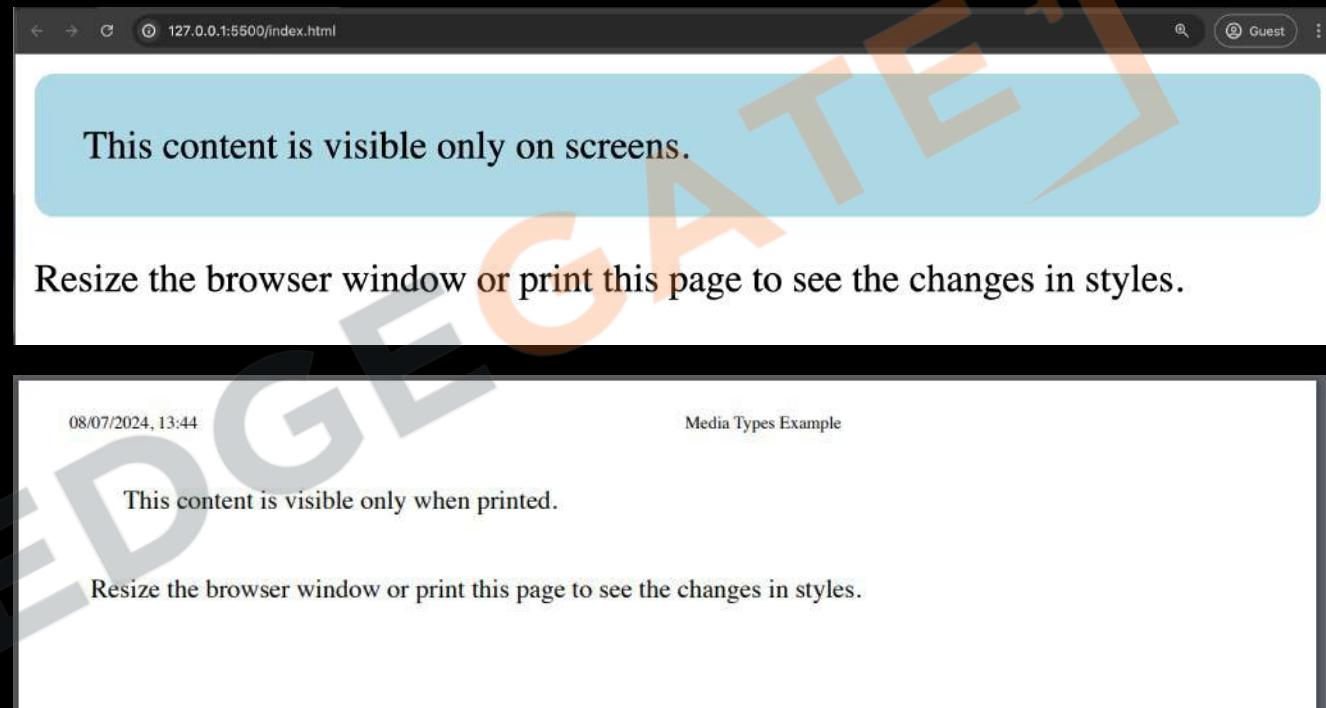
Media Queries (combination)

```
.box {  
    height: 100px;  
    width: 100px;  
    background-color: lightsalmon;  
}  
  
@media screen and (min-width: 250px)  
and (max-width: 300px) {  
    .box {  
        border-radius: 50%;  
    }  
}
```

A screenshot of a browser's developer tools, specifically the 'Styles' panel. It shows the CSS rules applied to the element. The rules include the base styles for '.box' and the media query rule that adds 'border-radius: 50%' to the element. Other rules listed in the panel include 'background-color: lightsalmon;', '@media screen and (min-width: 250px) and (max-width: 300px)', and various vendor-specific and root-level declarations.

Media Queries (different media types)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Media Types Example</title>
    <style>
        .screen-only {
            background-color: #lightblue;
            padding: 20px;
            border-radius: 8px;
        }
        .print-only {
            display: none;
        }
        @media print {
            .screen-only {
                display: none;
            }
            .print-only {
                display: block;
                background-color: #lightgray;
                padding: 20px;
                border-radius: 8px;
            }
        }
    </style>
</head>
<body>
    <div class="screen-only">
        This content is visible only on screens.
    </div>
    <div class="print-only">
        This content is visible only when printed.
    </div>
    <p>Resize the browser window or print this page to see the changes in styles.</p>
</body>
</html>
```



Different Media types are:

- All
- Print
- Screen
- Speech
- Braille, embossed

Media Queries (different condition types)

1. **min-width**: Applies styles when the **viewport width** is greater than or equal.
2. **max-width**: Applies styles when the **viewport width** is less than or equal.
3. **min-height**: Applies styles when the **viewport height** is greater than or equal.
4. **max-height**: Applies styles when the **viewport height** is less than or equal.
5. **orientation**: Applies styles based on the **device's orientation**, either portrait or landscape.
6. **min-resolution**: Applies styles when the **device's resolution** is greater than or equal.
7. **max-resolution**: Applies styles when the **device's resolution** is less than or equal.
8. **min-aspect-ratio**: Applies styles when the **device's aspect ratio** is greater than.
9. **max-aspect-ratio**: Applies styles when the **device's aspect ratio** is less than.
10. **min-device-width**: Applies styles when the **device's width** is greater than.
11. **max-device-width**: Applies styles when the **device's width** is less than or equal.
12. **min-device-height**: Applies styles when the **device's height** is greater than or equal.
13. **max-device-height**: Applies styles when the **device's height** is less than or equal.

Practice Set

Media Queries

- Create a div of 50px by 50px with color green. It should respond to following conditions:
 - Keep color **green** under 300px of view port size
 - Change color to **red** from 300px to 400px
 - Change color to **blue** after 400px



Practice Set Advanced

(Media Queries)

```
<header>
  <h1>My Portfolio</h1>
  <nav>
    <ul class="main-nav">
      <li>Home</li>
      <li>About</li>
      <li>Projects</li>
      <li>Contact</li>
    </ul>
  </nav>
</header>
<section class="intro">
  <h2>Welcome to My Portfolio</h2>
  <p>This is a brief introduction about myself.</p>
</section>
<section class="projects">
  <h2>Projects</h2>
  <div class="project">
    <h3>Project One</h3>
    <p>Description of project one.</p>
  </div>
  <div class="project">
    <h3>Project Two</h3>
    <p>Description of project two.</p>
  </div>
</section>
<footer>
  <p>&copy; 2023 My Portfolio</p>
</footer>
```

- 1. Basic Media Query:** Apply a background-color: lightblue; to the body when the screen width is less than 600px.
- 2. Min-Width Media Query:** Style the header with background-color: darkblue; and color: white; when the screen width is at least 600px.
- 3. Max-Width Media Query:** Set the header font-size to 1.2em when the screen width is at most 800px.
- 4. Orientation Media Query:** Apply a background-color: lightgreen; to the .projects section when the device orientation is landscape.
- 5. Min-Height Media Query:** Set the .intro section's padding: 20px; when the viewport height is at least 400px.
- 6. Combination Media Query:** Style the .project divs with a border: 2px solid black; and padding: 10px; when the screen width is between 600px and 1200px.
- 7. Different Media Types:** Apply a font-size: 12pt; to the body when printing.
- 8. Min-Resolution Media Query:** Apply a font-size: 1.5em; to the body when the device resolution is at least 2dppx.



Transitions



CSS transition is a property that enables smooth animation between changes in CSS property values

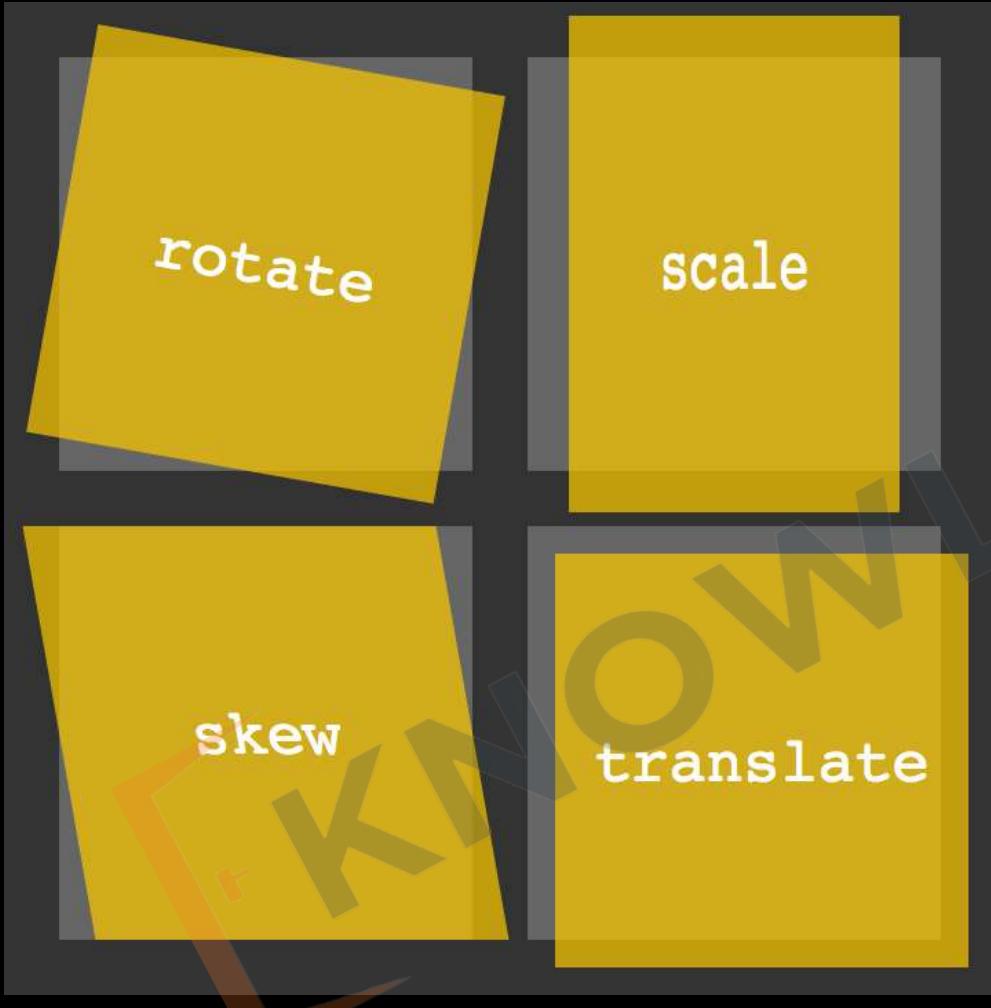
- **transition-property:** Defines which CSS properties will transition.
- **transition-duration:** Sets how long the transition lasts.
- **transition-timing-function:** Controls the speed curve of the transition.
- **transition-delay:** Specifies a delay before the transition starts.

Transitions

```
.btn {  
    height: 20px;  
    width: 70px;  
    border: 1px solid blue;  
    border-radius: 5px;  
    background-color: aliceblue;  
    transition-property: all;  
    transition-duration: 1s;  
    transition-timing-function: ease-in-out;  
    transition-delay: 1s;  
    /*transition: all 1s ease-in-out 1s;*/  
}  
  
.btn:hover {  
    height: 25px;  
    width: 80px;  
    border: 1px solid red;  
}  
  
.btn:active {  
    height: 25px;  
    width: 80px;  
    border: 1px solid red;  
    background-color: indianred;  
}
```



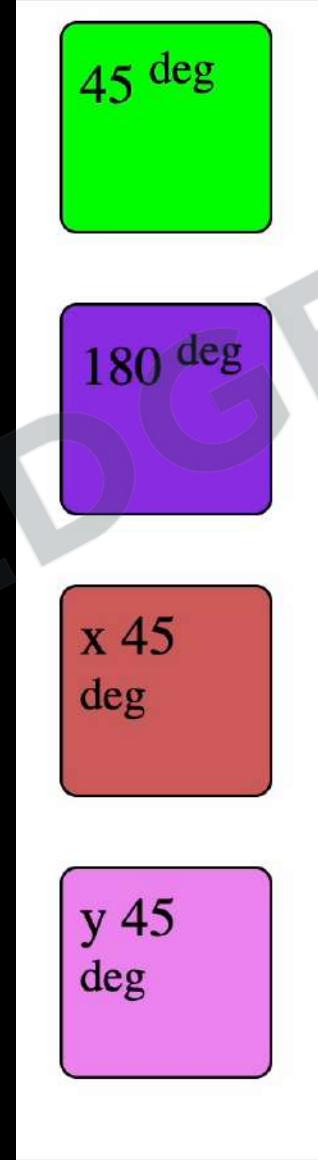
CSS Transform



- Allows modification of an element's shape and position.
- Can perform operations like `rotate`, `scale`, and `translate`.
- Does not affect the layout of surrounding elements.
- Used to create visual effects like 3D space transformations.
- Implemented with functions like `rotate()`, `scale()`, and `translate()`.

CSS Transform (Rotate)

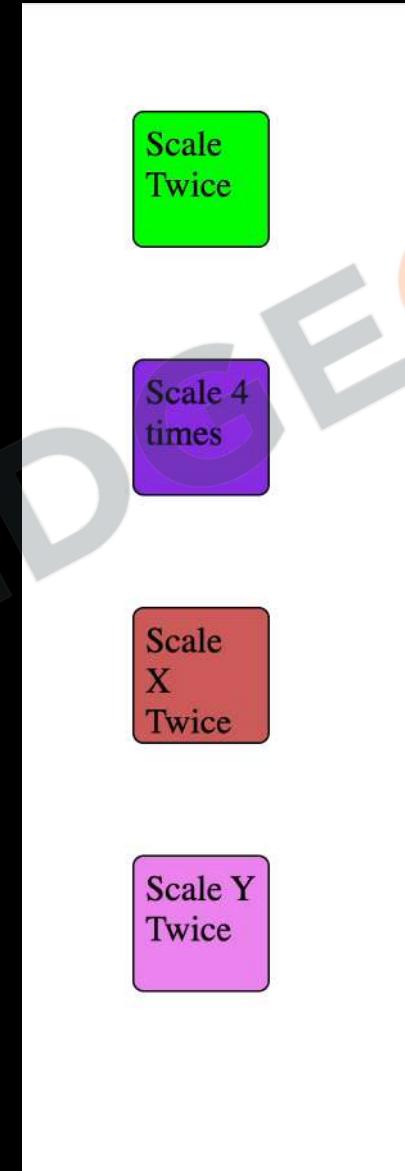
```
.box {  
    height: 50px;  
    width: 50px;  
    padding: 5px;  
    margin: 20px;  
    border: 1px solid black;  
    border-radius: 5px;  
    transition-property: all;  
    transition-duration: 1s;  
    transition-timing-function: ease-in-out;  
}  
  
#box1 { background-color: lime; }  
#box1:hover { transform: rotate(45deg); }  
  
#box2 { background-color: blueviolet; }  
#box2:hover { transform: rotate(180deg); }  
  
#box3 { background-color: indianred; }  
#box3:hover { transform: rotatex(45deg); }  
  
#box4 { background-color: violet; }  
#box4:hover { transform: rotatey(45deg); }
```



- Rotates an element around a fixed point.
- Defined using the `rotate()` function within the `transform` property.
- Default rotation point is the element's center.

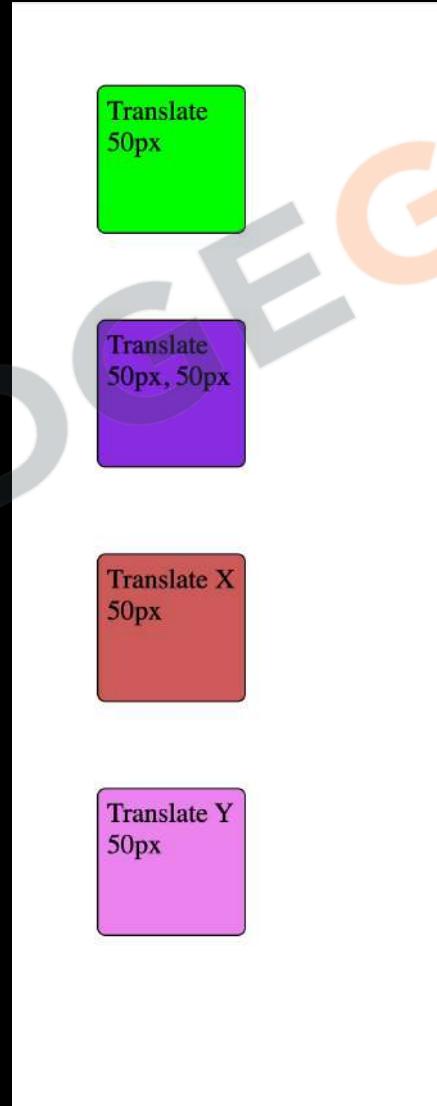
CSS Transform (Scale)

```
.box {  
    height: 50px;  
    width: 50px;  
    padding: 5px;  
    margin: 50px;  
    border: 1px solid black;  
    border-radius: 5px;  
    transition-property: all;  
    transition-duration: 1s;  
    transition-timing-function: ease-in-out;  
}  
  
#box1 { background-color: lime; }  
#box1:hover { transform: scale(2); }  
  
#box2 { background-color: blueviolet; }  
#box2:hover { transform: scale(4); }  
  
#box3 { background-color: indianred; }  
#box3:hover { transform: scalex(2); }  
  
#box4 { background-color: violet; }  
#box4:hover { transform: scaley(3); }
```



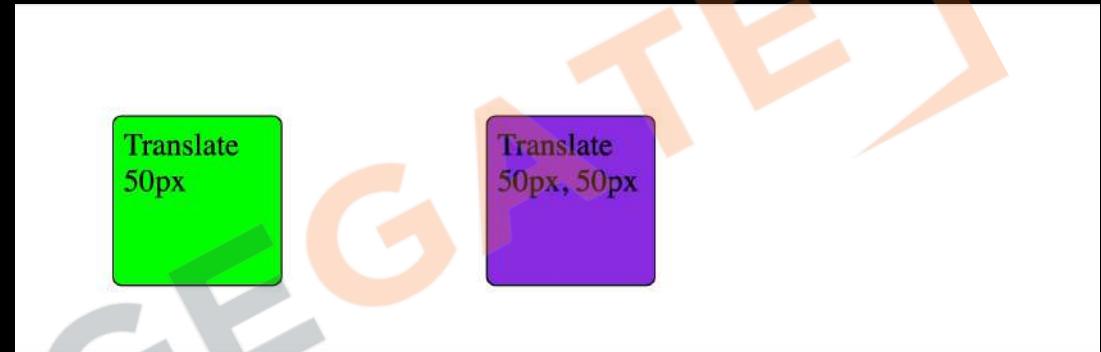
CSS Transform (Translate)

```
.box {  
    height: 75px;  
    width: 75px;  
    padding: 5px;  
    margin: 50px;  
    border: 1px solid black;  
    border-radius: 5px;  
    transition-property: all;  
    transition-duration: 1s;  
    transition-timing-function: ease-in-out;  
}  
  
#box1 { background-color: lime; }  
#box1:hover { transform: translate(50px); }  
#box2 { background-color: blueviolet; }  
#box2:hover { transform: translate(50px, 50px); }  
#box3 { background-color: indianred; }  
#box3:hover { transform: translateX(50px); }  
#box4 { background-color: violet; }  
#box4:hover { transform: translateY(50px); }
```



CSS Transform (Skew)

```
.box {  
    display: inline-block;  
    height: 75px;  
    width: 75px;  
    padding: 5px;  
    margin: 50px;  
    border: 1px solid black;  
    border-radius: 5px;  
    transition-property: all;  
    transition-duration: 1s;  
    transition-timing-function: ease-in-out;  
}  
  
#box1 { background-color: lime; }  
#box1:hover { transform: skew(45deg); }  
  
#box2 { background-color: blueviolet; }  
#box2:hover { transform: skew(90deg); }
```



Animation

The keyframes at rule rule start with "@keyframes" keyword

animation name which is specified in animation-name property



```
/* CSS code */  
@keyframes animation-name {  
    from { /* CSS code */ }  
    to { /* CSS code */ }  
}
```

specify where animation should end. You can write 100% as well instead of "to"

Specify when the style change will happen. You can write 0% as well, which is same as "from"

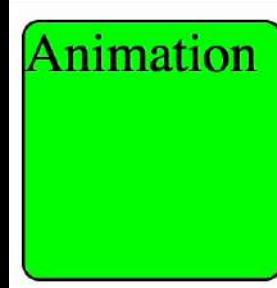
CSS properties.

Animation Properties

- **animation-name:** Specifies the name of the @keyframes defined animation.
- **animation-duration:** Defines the total time the animation takes to complete one cycle.
- **animation-timing-function:** Controls the pacing of the animation (e.g., linear, ease-in).
- **animation-delay:** Sets a delay before the animation starts, allowing for a pause before initiation.
- **animation-iteration-count:** Indicates the number of times the animation should repeat.
- **animation-direction:** Specifies the direction of the animation, allowing for reverse or alternate cycles.

Animation

```
.box {  
    height: 75px;  
    width: 75px;  
    border: 1px solid black;  
    border-radius: 5px;  
    position: absolute;  
    left: 10;  
    background-color: lime;  
  
    animation-name: ghumakkad;  
    animation-duration: 4s;  
    animation-timing-function: ease-in-out;  
    animation-delay: 0s;  
    animation-iteration-count: 4;  
  
    animation-direction: alternate;  
  
    /* animation: ghumakkad 4s ease-in-out 0s 4  
    alternate; */  
}  
  
@keyframes ghumakkad {  
    from {left: 10px}  
    to {left: 300px}  
}
```



Animation

Animation

```
@keyframes ghumakkad {  
    0% {left: 10px; top: 0px}  
    50% { left: 150px; top: 100px }  
    100% {left: 300px; top: 0px}  
}
```

Practice Set

Animation, Transition & Transform

- Create a webpage with a progress bar that showcases a smooth loading animation. The progress bar should fill up from 0 to 100% with a smooth transition effect and a slight bounce when it reaches 100%.



Advanced Practice Set

Animation, Transition & Transform

```
<head>
  <title>Interactive Card</title>
  <style>
    /* Your CSS code here */
  </style>
</head>
<body>
  <div class="card">
    <div class="card-front">
      <h2>Front Side</h2>
    </div>
    <div class="card-back">
      <h2>Back Side</h2>
    </div>
  </div>
</body>
```

1. **CSS Transition:** Apply a transition to the .card to smoothly transform the card when hovered. Use transform 0.6s ease-in-out.
2. **CSS Transform (Rotate):** Rotate the .card by 180 degrees when hovered.
3. **CSS Transform (Scale):** Scale the .card to 1.1 times its original size when hovered.
4. **CSS Transform (Translate):** Translate the .card 10px to the right and 20px down when hovered.
5. **CSS Transform (Skew):** Skew the .card by 10 degrees along the X-axis and 5 degrees along the Y-axis when hovered.
6. **CSS Animation:** Create a keyframe animation named flip that rotates the .card from 0 degrees to 180 degrees.
7. **Animation Properties:** Apply the flip animation to the .card with a duration of 2 seconds, an ease-in timing function, a delay of 0.5 seconds, 3 iterations, and alternate direction.

Advanced Practice Set (Solution)

Animation, Transition & Transform

```
1 /* Base styles for the card */
2 .card {
3   width: 200px;
4   height: 300px;
5   perspective: 1000px;
6 }
7
8 .card-front, .card-back {
9   width: 100%;
10  height: 100%;
11  position: absolute;
12  backface-visibility: hidden;
13 }
14
15 .card-front {
16   background-color: #fff;
17   color: #000;
18 }
19
20 .card-back {
21   background-color: #000;
22   color: #fff;
23   transform: rotateY(180deg);
24 }
25
26 /* CSS Transition */
27 .card {
28   transition: transform 0.6s ease-in-out;
29 }
30
31 /* CSS Transform (Rotate) */
32 .card:hover {
33   transform: rotateY(180deg);
34 }
35
36 /* CSS Transform (Scale) */
37 .card:hover {
38   transform: scale(1.1);
39 }
40
41 /* CSS Transform (Translate) */
42 .card:hover {
43   transform: translate(10px, 20px);
44 }
45
46 /* CSS Transform (Skew) */
47 .card:hover {
48   transform: skew(10deg, 5deg);
49 }
50
51 /* CSS Animation */
52 @keyframes flip {
53   from {
54     transform: rotateY(0deg);
55   }
56   to {
57     transform: rotateY(180deg);
58   }
59 }
60
61 .card {
62   animation-name: flip;
63   animation-duration: 2s;
64   animation-timing-function: ease-in;
65   animation-delay: 0.5s;
66   animation-iteration-count: 3;
67   animation-direction: alternate;
68 }
```



Grid

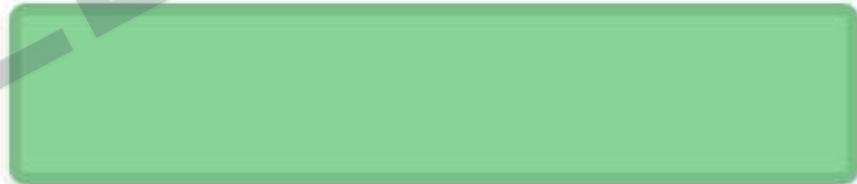
Flexbox

One-dimensional layout



Grid

Multi-dimensional layout



KNOWLEDGE GATE

Grid template rows & Columns

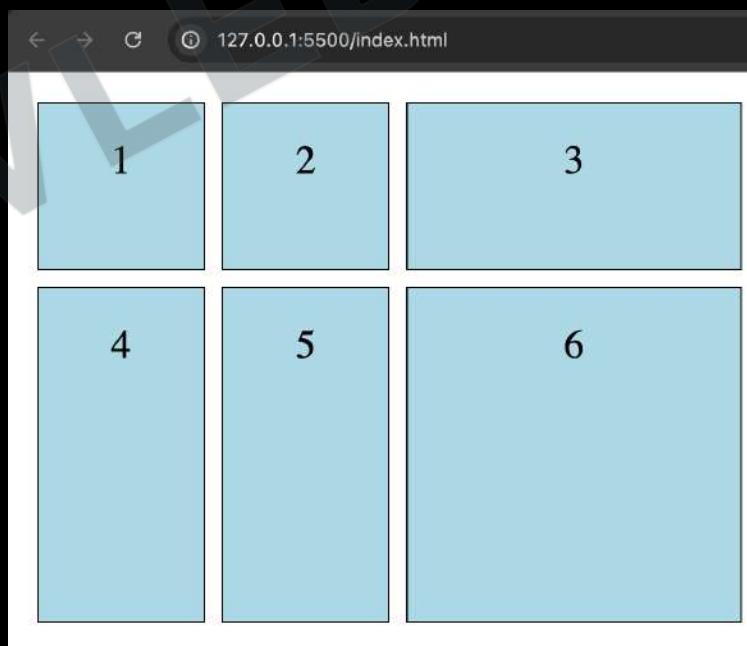
```
<head>
  <title>Grid Template Example</title>
  <style>
    .grid-container {
      display: grid;
      grid-template-rows: 100px 200px;
      grid-template-columns: 100px 100px 200px;
      gap: 10px;
      padding: 10px;
    }
    .grid-item {
      background-color: lightblue;
      border: 1px solid #000;
      text-align: center;
      padding: 20px;
      font-size: 1.5em;
    }
  </style>
</head>
<body>
  <div class="grid-container">
    <div class="grid-item">1</div>
    <div class="grid-item">2</div>
    <div class="grid-item">3</div>
    <div class="grid-item">4</div>
    <div class="grid-item">5</div>
    <div class="grid-item">6</div>
  </div>
</body>
```

- **CSS Grid Template Rows:**

- Defines the **rows** of the grid with specific sizes.
- Accepts values such as pixels (px), percentages (%), fractions (fr), or the auto keyword.
- Can create fixed or flexible grid layouts.

- **Grid Template Columns:**

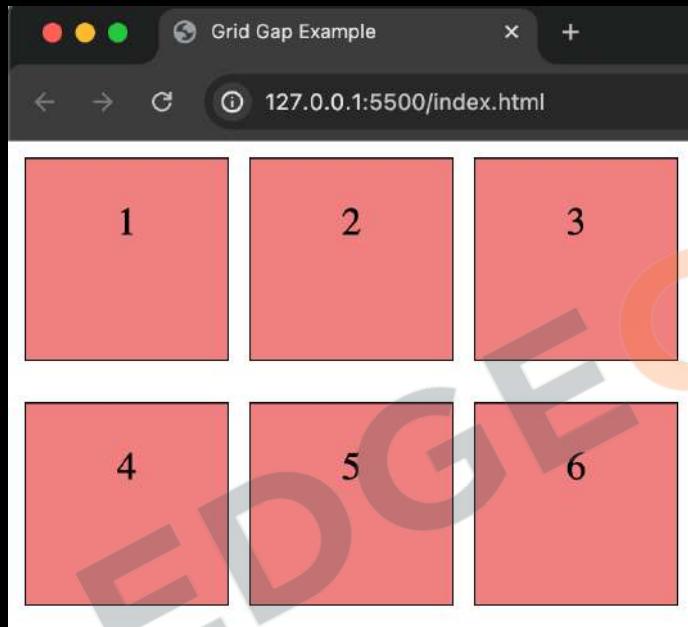
- Defines the **columns of the grid** with specific sizes.
- Similar to rows, it accepts pixels, percentages, fractions, or the auto keyword.
- Helps in designing the layout structure by specifying the width of each column.



Grid-Gap

```
<head>
<title>Grid Gap Example</title>
<style>
.grid-container {
  display: grid;
  grid-template-rows: 100px 100px 100px;
  grid-template-columns: 100px 100px 100px;
  gap: 20px 10px; /* 20px row gap, 10px column gap */
}

.grid-item {
  background-color: lightcoral;
  border: 1px solid #000;
  text-align: center;
  padding: 20px;
  font-size: 1.2em;
}
</style>
</head>
<body>
<div class="grid-container">
  <div class="grid-item">1</div>
  <div class="grid-item">2</div>
  <div class="grid-item">3</div>
  <div class="grid-item">4</div>
  <div class="grid-item">5</div>
  <div class="grid-item">6</div>
</div>
</body>
```



Definition:

- **grid-gap**: A shorthand property for setting the gaps (spaces) between rows and columns in a grid layout.
- **gap**: The **modern shorthand** for setting both row and column gaps, replacing grid-gap.

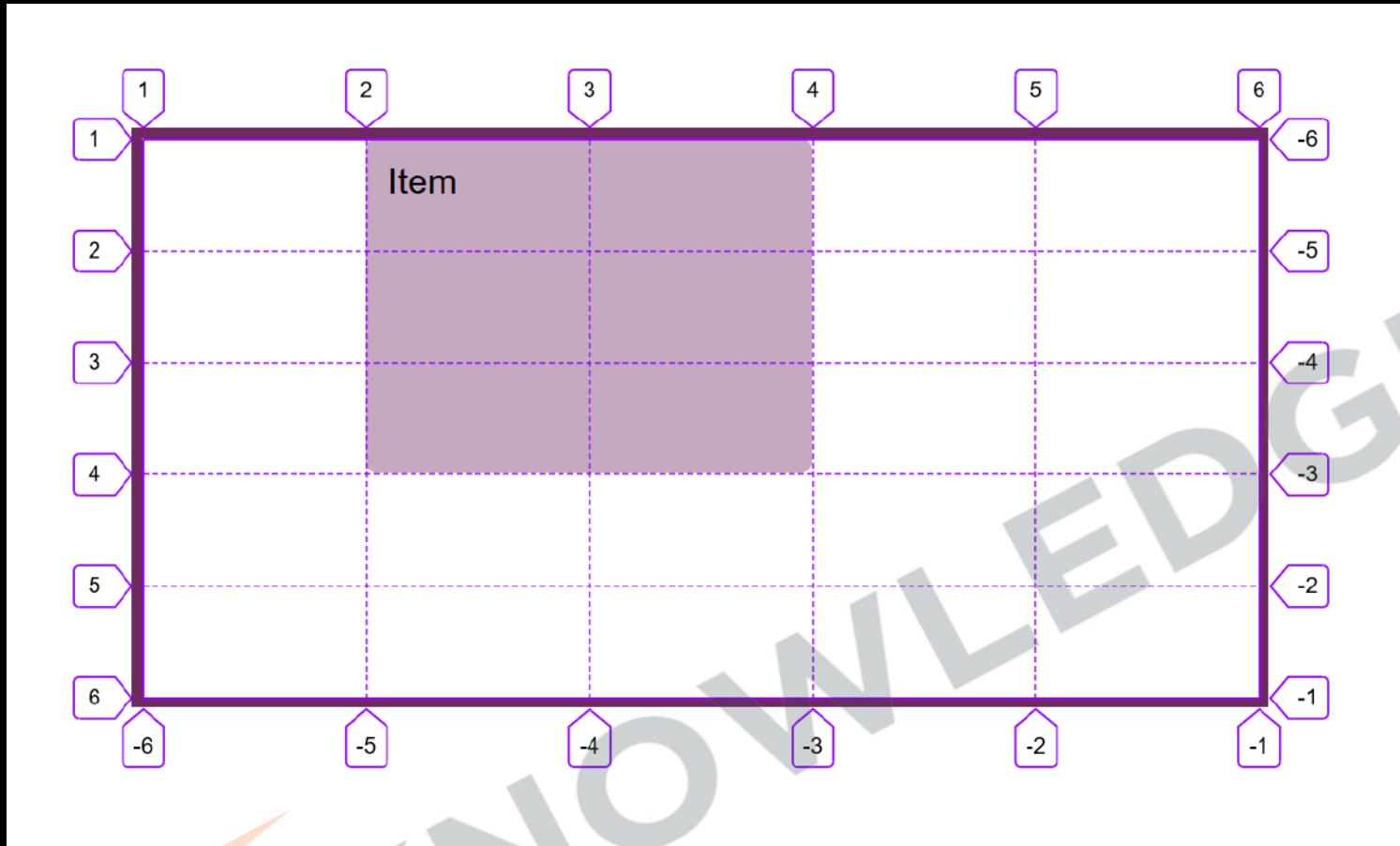
Components:

- **row-gap**: Specifies the size of the gap between grid rows.
- **column-gap**: Specifies the size of the gap between grid columns.

Syntax:

- **gap: <row-gap> <column-gap>;**
- If only one value is provided, it applies to both row and column gaps.

Grid Lines

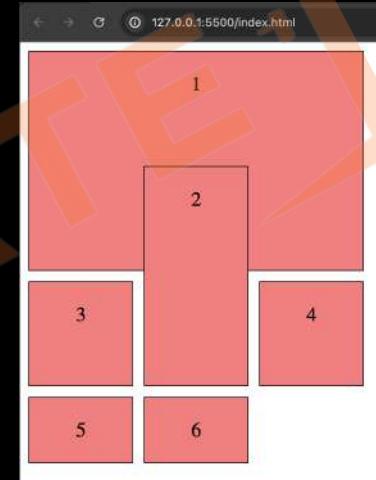


- Grid lines are the **horizontal and vertical lines** that **divide the grid into cells**, defined by the grid rows and columns.
- **Numbering:** Grid lines are numbered **starting from 1**, with the **first line before the first row/column** and the last line after the last row/column.
- **Usage:** Grid lines are used to **place and span grid items precisely** within the grid layout.
- Negative numbers can be used to reference grid lines, starting from the end of the grid. -1 refers to the last grid line, -2 to the second-last, and so on.

Grid row/column start/end

```
.grid-container {  
    display: grid;  
    grid-template-rows: 100px 100px 100px;  
    grid-template-columns: 100px 100px 100px;  
    gap: 10px;  
}  
  
.grid-item {  
    background-color: lightcoral;  
    border: 1px solid #000;  
    text-align: center;  
    padding: 20px;  
    font-size: 1.2em;  
}  
  
.item1 {  
    grid-row-start: 1;  
    /* Spans from row line 1 to row line 3 */  
    grid-row-end: 3;  
    grid-column-start: 1;  
    /* Spans from column line 1 to column line 4 */  
    grid-column-end: 4;  
}  
  
.item2 {  
    grid-row-start: 2;  
    /* Spans from row line 2 to row line 4 */  
    grid-row-end: 4;  
    grid-column-start: 2;  
    /* Spans from column line 2 to column line 3 */  
    grid-column-end: 3;  
}
```

```
<div class="grid-container">  
    <div class="grid-item item1">1</div>  
    <div class="grid-item item2">2</div>  
    <div class="grid-item">3</div>  
    <div class="grid-item">4</div>  
    <div class="grid-item">5</div>  
    <div class="grid-item">6</div>  
</div>
```



• Grid Row Start/End:

- **grid-row-start:** Specifies the **starting grid line** for a grid item on the **row axis**.
- **grid-row-end:** Specifies the **ending grid line** for a grid item on the **row axis**.
- **Usage:** Defines the **vertical position** and **span** of a grid item.

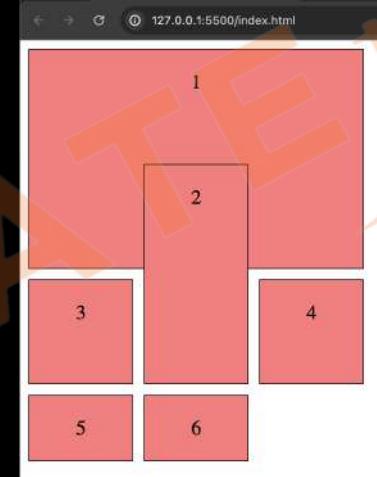
• Grid Column Start/End:

- **grid-column-start:** Specifies the **starting grid line** for a grid item on the **column axis**.
- **grid-column-end:** Specifies the **ending grid line** for a grid item on the **column axis**.
- **Usage:** Defines the **horizontal position** and **span** of a grid item.

Grid-row & Grid-column

```
.grid-container {  
    display: grid;  
    grid-template-rows: 100px 100px 100px;  
    grid-template-columns: 100px 100px 100px;  
    gap: 10px;  
}  
.grid-item {  
    background-color: lightcoral;  
    border: 1px solid #000;  
    text-align: center;  
    padding: 20px;  
    font-size: 1.2em;  
}  
.item1 {  
    /* Equivalent to grid-row-start: 1;  
     * grid-row-end: 3; */  
    grid-row: 1 / 3;  
    /* Equivalent to grid-column-start:  
     * 1; grid-column-end: 4; */  
    grid-column: 1 / 4;  
}  
.item2 {  
    /* Equivalent to grid-row-start: 2;  
     * grid-row-end: 4; */  
    grid-row: 2 / 4;  
    /* Equivalent to grid-column-start: 2;  
     * grid-column-end: 3; */  
    grid-column: 2 / 3;  
}
```

```
<div class="grid-container">  
    <div class="grid-item item1">1</div>  
    <div class="grid-item item2">2</div>  
    <div class="grid-item">3</div>  
    <div class="grid-item">4</div>  
    <div class="grid-item">5</div>  
    <div class="grid-item">6</div>  
</div>
```



- **Grid Row:**

- **grid-row:** A shorthand property for setting both grid-row-start and grid-row-end.
- **Syntax:** `grid-row: <start-line> / <end-line>;`
- **Usage:** Simplifies the declaration by combining the start and end lines of a grid item on the row axis.

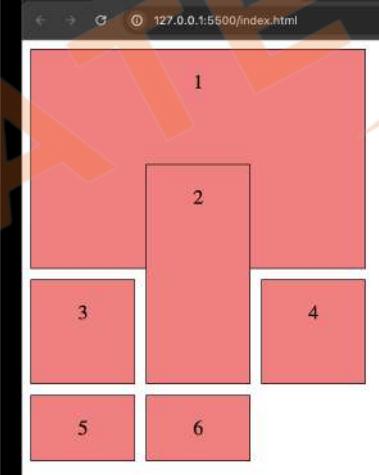
- **Grid Column:**

- **grid-column:** A shorthand property for setting both grid-column-start and grid-column-end.
- **Syntax:** `grid-column: <start-line> / <end-line>;`
- **Usage:** Simplifies the declaration by combining the start and end lines of a grid item on the column axis.

Span Keyword

```
.grid-container {  
    display: grid;  
    grid-template-rows: 100px 100px 100px;  
    grid-template-columns: 100px 100px 100px;  
    gap: 10px;  
}  
  
.grid-item {  
    background-color: lightcoral;  
    border: 1px solid black;  
    text-align: center;  
    padding: 20px;  
    font-size: 1.2em;  
}  
  
.item1 {  
    /* Spans 2 rows starting from row line 1 */  
    grid-row: 1 / span 2;  
    /* Spans 3 columns starting from column line 1 */  
    grid-column: 1 / span 3;  
}  
  
.item2 {  
    /* Spans 2 rows starting from row line 2 */  
    grid-row: 2 / span 2;  
    /* Spans 1 column starting from column line 2 */  
    grid-column: 2 / span 1;  
}
```

```
<div class="grid-container">  
    <div class="grid-item item1">1</div>  
    <div class="grid-item item2">2</div>  
    <div class="grid-item">3</div>  
    <div class="grid-item">4</div>  
    <div class="grid-item">5</div>  
    <div class="grid-item">6</div>  
</div>
```



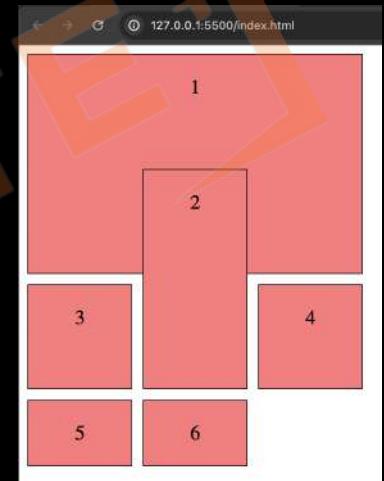
- Grid Row and Column with span:

- span: A keyword used with `grid-row` and `grid-column` to span a grid item across multiple rows or columns.
- Syntax: `grid-row: <start-line> / span <number-of-rows>;` and `grid-column: <start-line> / span <number-of-columns>;`
- Usage: Provides a more intuitive way to specify the number of rows or columns a grid item should span, starting from a given line.

Grid-Area

```
.grid-container {  
    display: grid;  
    grid-template-rows: 100px 100px 100px;  
    grid-template-columns: 100px 100px 100px;  
    gap: 10px;  
}  
.grid-item {  
    background-color: lightcoral;  
    border: 1px solid #000;  
    text-align: center;  
    padding: 20px;  
    font-size: 1.2em;  
}  
.item1 {  
/* Spans from row 1 to 3 and column 1 to 4 */  
    grid-area: 1 / 1 / 3 / 4;  
}  
.item2 {  
/* Spans from row 2 to last row and  
column 2 to second last column */  
    grid-area: 2 / 2 / -1 / -2;  
}
```

```
<div class="grid-container">  
    <div class="grid-item item1">1</div>  
    <div class="grid-item item2">2</div>  
    <div class="grid-item">3</div>  
    <div class="grid-item">4</div>  
    <div class="grid-item">5</div>  
    <div class="grid-item">6</div>  
</div>
```

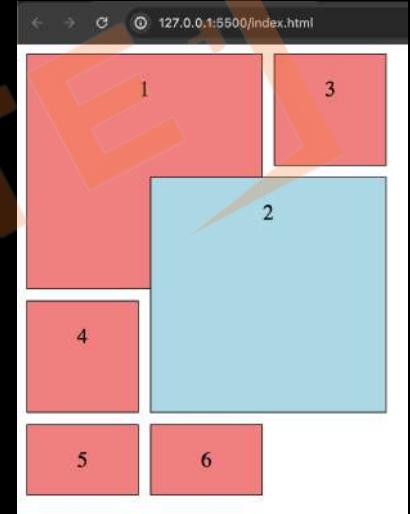


- Grid Area:
 - Definition: The grid-area property is used to **define a grid item's size and location** within a grid by specifying the start and end lines for both rows and columns.
 - Syntax: `grid-area: <row-start> / <column-start> / <row-end> / <column-end>;`
 - Usage: Combines `grid-row-start`, `grid-row-end`, `grid-column-start`, and `grid-column-end` into a single **property** for more concise code.

Layering with Z-index

```
.grid-container {  
    display: grid;  
    grid-template-rows: 100px 100px 100px;  
    grid-template-columns: 100px 100px 100px;  
    gap: 10px;  
}  
  
.grid-item {  
    background-color: lightcoral;  
    border: 1px solid #000;  
    text-align: center;  
    padding: 20px;  
    font-size: 1.2em;  
}  
  
.item1 {  
/* Spans from row 1 to 3 and column 1 to 3 */  
    grid-area: 1 / 1 / 3 / 3;  
/* Lower z-index */  
    z-index: 1;  
}  
  
.item2 {  
/* Spans from row 2 to 4 and column 2 to 4 */  
    grid-area: 2 / 2 / 4 / 4;  
    z-index: 2; /* Higher z-index */  
    background-color: lightblue;  
}
```

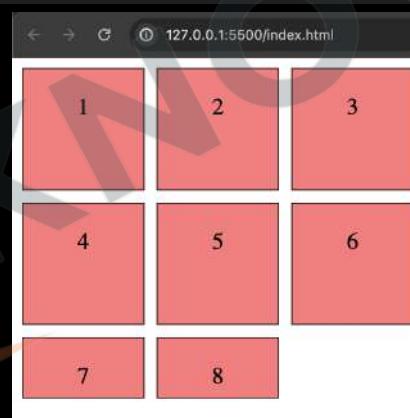
```
<div class="grid-container">  
    <div class="grid-item item1">1</div>  
    <div class="grid-item item2">2</div>  
    <div class="grid-item">3</div>  
    <div class="grid-item">4</div>  
    <div class="grid-item">5</div>  
    <div class="grid-item">6</div>  
</div>
```



- Using z-index with CSS Grid
- Definition: A CSS property that specifies the stack order of elements. Elements with higher z-index values are rendered on top of elements with lower values.
- Usage in Grid: When working with CSS Grid, you can use z-index to control the layering of grid items. This is particularly useful when grid items overlap, either due to negative margins or explicitly overlapping grid areas.

Grid Overflow & Grid-Auto-Rows

```
.grid-container {  
    display: grid;  
    grid-template-rows: 100px 100px;  
    grid-template-columns: 100px 100px 100px;  
    /* Automatically added rows will be 50px high */  
    grid-auto-rows: 50px;  
    /* Adds scrollbars if content overflows */  
    overflow: auto;  
    gap: 10px;  
    /* Limits the height of the grid container */  
    max-height: 350px;  
}  
  
.grid-item {  
    background-color: lightcoral;  
    border: 1px solid #000;  
    text-align: center;  
    padding: 20px;  
    font-size: 1.2em;  
}  
  
<body>  
<div class="grid-container">  
    <div class="grid-item">1</div>  
    <div class="grid-item">2</div>  
    <div class="grid-item">3</div>  
    <div class="grid-item">4</div>  
    <div class="grid-item">5</div>  
    <div class="grid-item">6</div>  
    <div class="grid-item">7</div>  
    <!-- Implicit row -->  
    <div class="grid-item">8</div>  
    <!-- Implicit row -->  
</div>  
</body>
```



Grid Overflow:

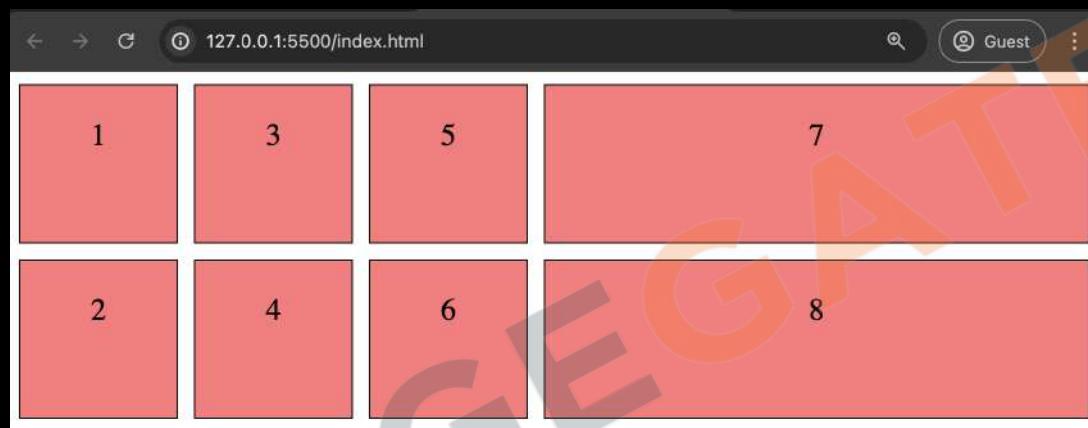
- **Definition:** Determines how content that overflows the boundaries
- **Properties:**
 - **overflow:** Controls both horizontal and vertical overflow.
 - **overflow-x:** Controls horizontal overflow.
 - **overflow-y:** Controls vertical overflow.
- **Values:**
 - **visible:** Default. Content is not clipped and may be rendered outside the container.
 - **hidden:** Content is clipped and not visible outside the container.
 - **scroll:** Content is clipped, but scrollbars are provided to view the hidden content.
 - **auto:** Similar to scroll, but scrollbars are only provided when necessary.

Grid-Auto-Rows:

- **Definition:** Defines the size of implicitly created rows in a grid layout when the number of items exceeds the explicitly defined grid structure.
- **Usage:** Ensures that additional rows created dynamically have a consistent size.

Grid-Auto-Flow

```
<head>
  <title>Grid Auto Flow Example</title>
  <style>
    .grid-container {
      display: grid;
      grid-template-rows: 100px 100px;
      grid-template-columns: 100px 100px 100px;
      grid-auto-flow: column; /* Items will flow in columns */
      gap: 10px;
    }
    .grid-item {
      background-color: lightcoral;
      border: 1px solid #000;
      text-align: center;
      padding: 20px;
      font-size: 1.2em;
    }
  </style>
</head>
<body>
  <div class="grid-container">
    <div class="grid-item">1</div>
    <div class="grid-item">2</div>
    <div class="grid-item">3</div>
    <div class="grid-item">4</div>
    <div class="grid-item">5</div>
    <div class="grid-item">6</div>
    <div class="grid-item">7</div>
    <div class="grid-item">8</div>
  </div>
</body>
```



grid-auto-flow: Controls how **auto-placed items** are inserted into the grid.

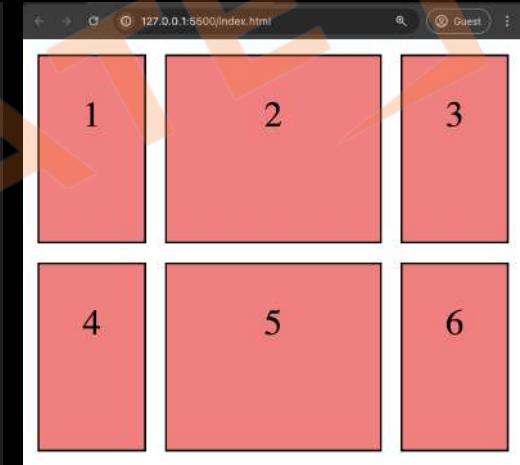
Values:

- **row:** Places items in **rows**, filling each row before moving to the next.
- **column:** Places items in **columns**, filling each column before moving to the next.
- **dense:** Tries to **fill in gaps** in the grid, ensuring items are placed in the smallest available spot.
- **row dense:** Combines row and dense, placing items in rows and trying to fill gaps.
- **column dense:** Combines column and dense, placing items in columns and trying to fill gaps.

Fractional Units

```
<style>  
  .grid-container {  
    display: grid;  
    grid-template-rows: 100px 100px;  
    /* Columns will take 1/4, 1/2, and 1/4  
     * of the space respectively */  
    grid-template-columns: 1fr 2fr 1fr;  
    gap: 10px;  
  }  
  
  .grid-item {  
    background-color: lightcoral;  
    border: 1px solid black;  
    text-align: center;  
    padding: 20px;  
    font-size: 1.2em;  
  }  
</style>
```

```
<div class="grid-container">  
  <div class="grid-item">1</div>  
  <div class="grid-item">2</div>  
  <div class="grid-item">3</div>  
  <div class="grid-item">4</div>  
  <div class="grid-item">5</div>  
  <div class="grid-item">6</div>  
</div>
```



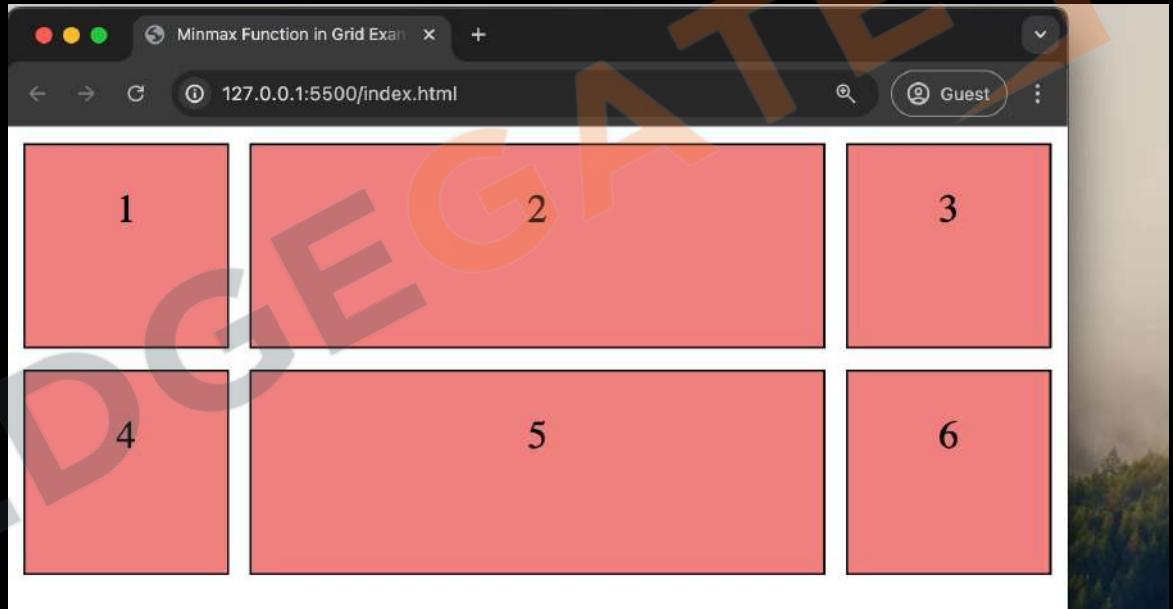
Fractional Units (fr): A flexible unit that represents a fraction of the available space in the grid container.

Usage: Allows for responsive and flexible grid layouts by dividing the remaining space among grid tracks.

min-max Function

```
<style>
  .grid-container {
    display: grid;
    grid-template-rows: 100px 100px;
    grid-template-columns: 100px minmax(150px, 1fr) 100px;
    /* Second column has a minimum of 150px and a maximum of 1fr */
    gap: 10px;
  }

  .grid-item {
    background-color: lightcoral;
    border: 1px solid #000;
    text-align: center;
    padding: 20px;
    font-size: 1.2em;
  }
}
```

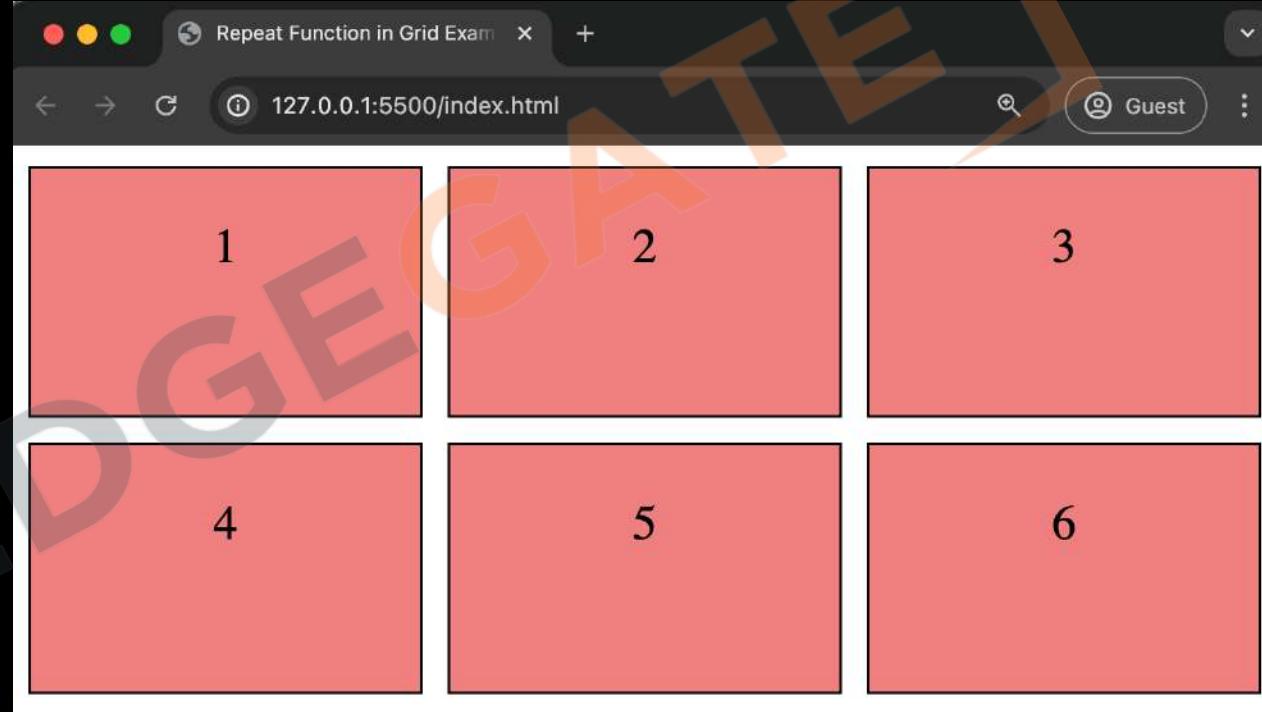


minmax(min, max): A CSS Grid function that defines a track size range, setting a minimum and maximum size for grid tracks (rows or columns).

Usage: Ensures grid tracks are flexible within specified constraints, improving layout responsiveness.

repeat Function

```
<style>
  .grid-container {
    display: grid;
    grid-template-rows: repeat(2, 100px);
    /* Creates 2 rows, each 100px high */
    grid-template-columns: repeat(3, 1fr);
    /* Creates 3 columns, each 1 fraction of the available space */
    gap: 10px;
  }
  .grid-item {
    background-color: lightcoral;
    border: 1px solid #000;
    text-align: center;
    padding: 20px;
    font-size: 1.2em;
  }
}
```

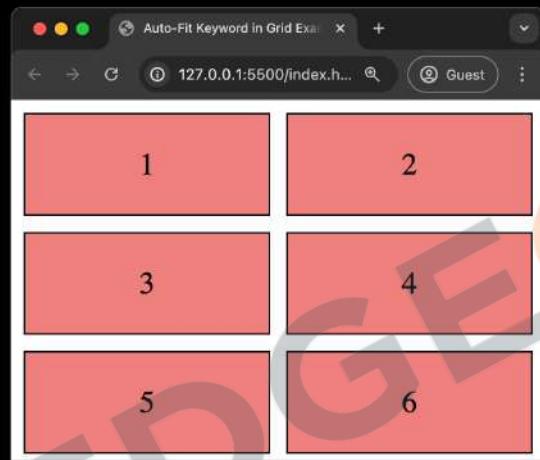


repeat(count, size): A CSS Grid function that allows you to repeat grid track definitions (rows or columns) a specified number of times.

- **Usage:** Simplifies the definition of repetitive grid structures, reducing code redundancy and improving readability.

auto-fit Keyword

```
<head>
  <title>Auto-Fit Keyword in Grid Example</title>
  <style>
    .grid-container {
      display: grid;
      grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));
      /* Auto-fit as many 150px columns as possible */
      gap: 10px;
    }
    .grid-item {
      background-color: lightcoral;
      border: 1px solid #000;
      text-align: center;
      padding: 20px;
      font-size: 1.2em;
    }
  </style>
</head>
<body>
  <div class="grid-container">
    <div class="grid-item">1</div>
    <div class="grid-item">2</div>
    <div class="grid-item">3</div>
    <div class="grid-item">4</div>
    <div class="grid-item">5</div>
    <div class="grid-item">6</div>
  </div>
</body>
```



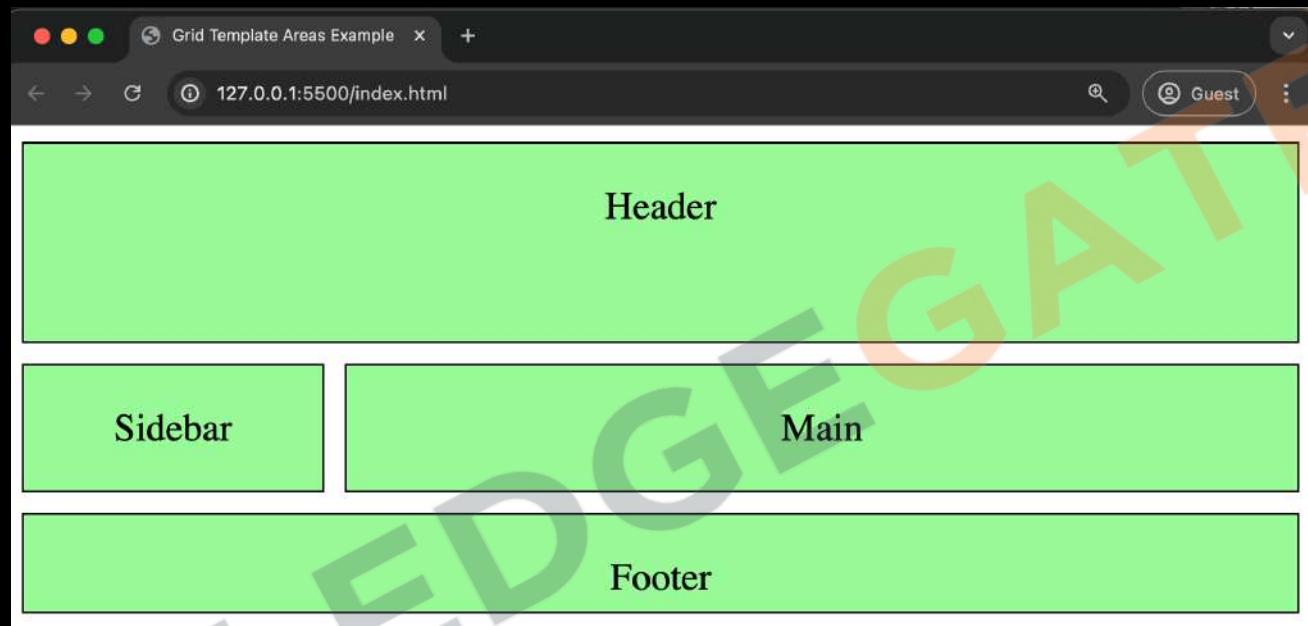
auto-fit: A keyword used in the `repeat()` function to automatically fit as many columns (or rows) as possible into the available space. The columns (or rows) will expand to fill the space.

Advantages:

- Creates responsive layouts that adapt to different screen sizes.
- Automatically adjusts the number of columns (or rows) based on the available space, ensuring optimal use of space.

Grid-Template-Areas

```
.grid-container {  
    display: grid;  
    grid-template-areas:  
        "header header header"  
        "sidebar main main"  
        "footer footer footer";  
    grid-template-rows: 100px 1fr 50px;  
    grid-template-columns: 150px 1fr 1fr;  
    gap: 10px;  
}  
  
.grid-item {  
    background-color: palegreen;  
    border: 1px solid #000;  
    text-align: center;  
    padding: 20px;  
    font-size: 1.2em;  
}  
  
.header { grid-area: header; }  
.sidebar { grid-area: sidebar; }  
.main { grid-area: main; }  
.footer { grid-area: footer; }  
  
<div class="grid-container">  
    <div class="grid-item header">Header</div>  
    <div class="grid-item sidebar">Sidebar</div>  
    <div class="grid-item main">Main</div>  
    <div class="grid-item footer">Footer</div>  
</div>
```



grid-template-areas: A CSS Grid property that defines a grid template by specifying named grid areas. These named areas can then be used to place grid items within the grid layout.

Advantages:

- Simplifies the layout design by using named areas.
- Makes the grid layout more readable and maintainable.

Syntax:

- Define areas using strings, with each string representing a row in the grid.
- Use the same name for grid cells that should be merged into a single area.
- Use a period (.) to represent an empty cell.

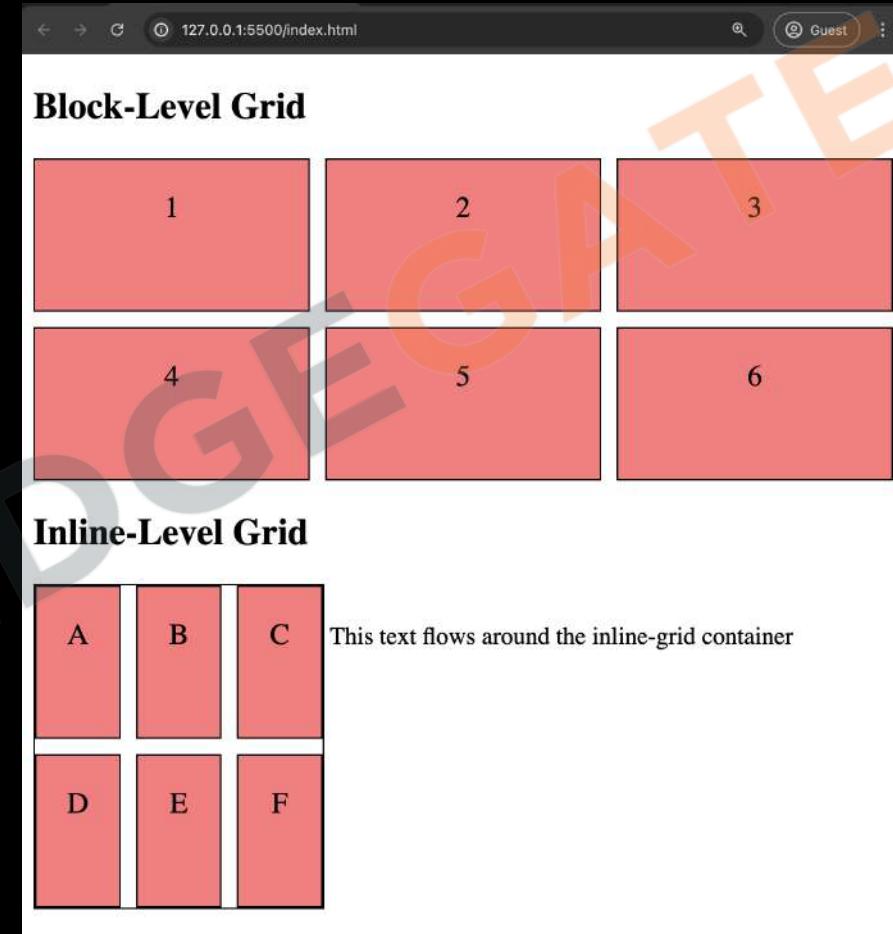
Grid vs Inline-Grid

```
<style>
  .grid-container {
    display: grid;
    grid-template-rows: 100px 100px;
    grid-template-columns: 1fr 1fr 1fr;
    gap: 10px;
    margin-bottom: 20px;
  }
  .inline-grid-container {
    display: inline-grid;
    grid-template-rows: 100px 100px;
    grid-template-columns: 1fr 1fr 1fr;
    gap: 10px;
    border: 1px solid #000;
  }
  .grid-item {
    background-color: lightcoral;
    border: 1px solid #000;
    text-align: center;
    padding: 20px;
    font-size: 1.2em;
  }
</style>
```

```
<body>
  <h2>Block-Level Grid</h2>
  <div class="grid-container">
    <div class="grid-item">1</div>
    <div class="grid-item">2</div>
    <div class="grid-item">3</div>
    <div class="grid-item">4</div>
    <div class="grid-item">5</div>
    <div class="grid-item">6</div>
  </div>

  <h2>Inline-Level Grid</h2>
  <div class="inline-grid-container">
    <div class="grid-item">A</div>
    <div class="grid-item">B</div>
    <div class="grid-item">C</div>
    <div class="grid-item">D</div>
    <div class="grid-item">E</div>
    <div class="grid-item">F</div>
  </div>

  <span>
    This text flows around the
    inline-grid container
  </span>
</body>
```

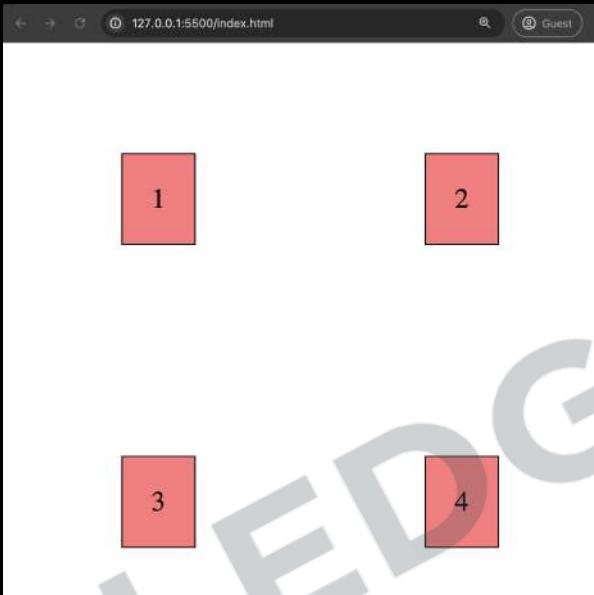


display: grid;: Establishes a **block-level** grid container.

display: inline-grid;: Establishes an **inline-level grid container**, making the grid container behave like an inline element while retaining grid layout capabilities.

Justify-Items & Align-Items

```
<head>
  <title>Justify-Items and Align-Items Example</title>
  <style>
    .grid-container {
      display: grid;
      grid-template-rows: 200px 200px;
      grid-template-columns: 200px 200px;
      gap: 10px;
      /* Centers items horizontally */
      justify-items: center;
      /* Centers items vertically */
      align-items: center;
    }
    .grid-item {
      background-color: lightcoral;
      border: 1px solid #000;
      text-align: center;
      padding: 20px;
      font-size: 1.2em;
    }
  </style>
</head>
<body>
  <div class="grid-container">
    <div class="grid-item">1</div>
    <div class="grid-item">2</div>
    <div class="grid-item">3</div>
    <div class="grid-item">4</div>
  </div>
</body>
```



justify-items and align-items in CSS Grid

Definition:

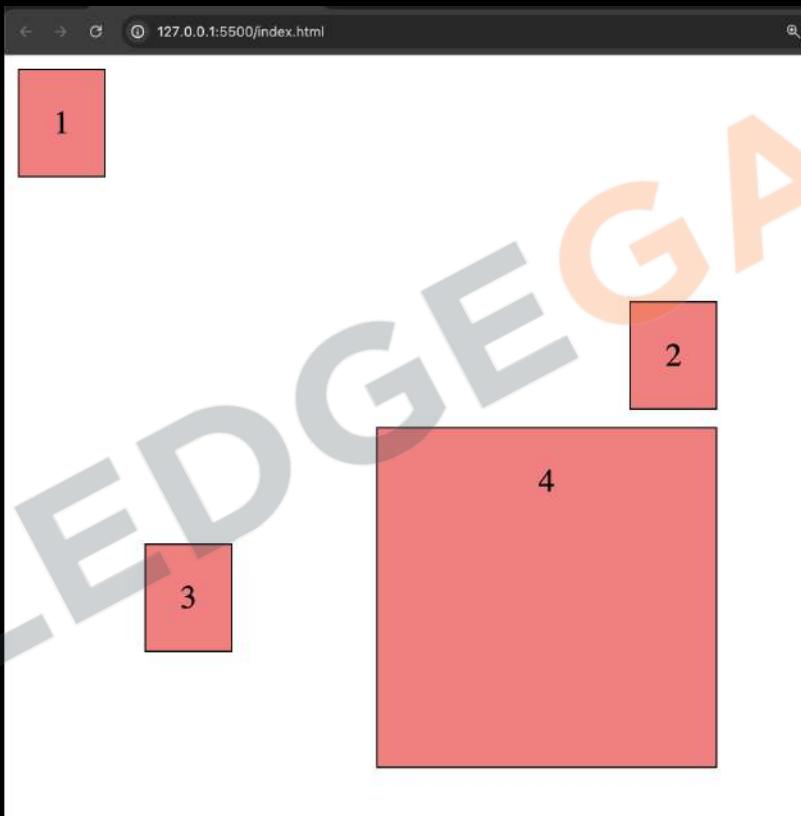
- **justify-items:** Aligns grid items along the inline (row) axis.
- **align-items:** Aligns grid items along the block (column) axis.

Values:

- **start:** Aligns items at the **start** of the grid area.
- **end:** Aligns items at the **end** of the grid area.
- **center:** Aligns items in the **center** of the grid area.
- **stretch:** Stretches items to **fill the grid area** (default).

Justify-Self & Align-Self

```
.grid-container {  
  display: grid;  
  grid-template-rows: 200px 200px;  
  grid-template-columns: 200px 200px;  
  gap: 10px;  
  justify-items: center; /* Centers all items horizontally by default */  
  align-items: center; /* Centers all items vertically by default */  
}  
  
.grid-item {  
  background-color: lightcoral;  
  border: 1px solid #000;  
  text-align: center;  
  padding: 20px;  
  font-size: 1.2em;  
}  
  
.item1 {  
  justify-self: start; /* Aligns item 1 to the start horizontally */  
  align-self: start; /* Aligns item 1 to the start vertically */  
}  
  
.item2 {  
  justify-self: end; /* Aligns item 2 to the end horizontally */  
  align-self: end; /* Aligns item 2 to the end vertically */  
}  
  
.item3 {  
  justify-self: center; /* Aligns item 3 to the center horizontally */  
  align-self: center; /* Aligns item 3 to the center vertically */  
}  
  
.item4 {  
  justify-self: stretch; /* Stretches item 4 to fill horizontally */  
  align-self: stretch; /* Stretches item 4 to fill vertically */  
}
```



```
<div class="grid-container">  
  <div class="grid-item item1">1</div>  
  <div class="grid-item item2">2</div>  
  <div class="grid-item item3">3</div>  
  <div class="grid-item item4">4</div>  
</div>
```

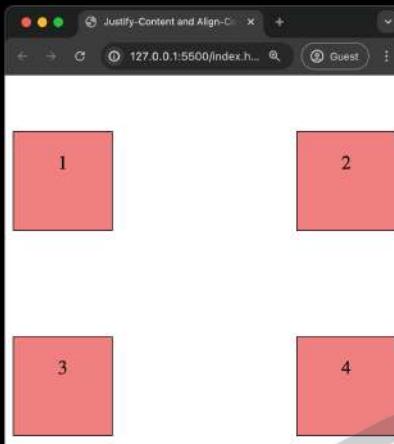
justify-self and align-self in CSS Grid

- **justify-self:** Aligns a single grid item along the inline (row) axis.
- **align-self:** Aligns a single grid item along the block (column) axis.

Justify-Content & Align-Content

```
<head>
  <title>Justify-Content and Align-Content Example</title>
  <style>
    .grid-container {
      display: grid;
      grid-template-rows: 100px 100px;
      grid-template-columns: 100px 100px;
      gap: 10px;
      /* Distributes space between columns */
      justify-content: space-between;
      /* Distributes space around rows */
      align-content: space-around;
      /* Added height to demonstrate align-content */
      height: 400px;
    }

    .grid-item {
      background-color: lightcoral;
      border: 1px solid #000;
      text-align: center;
      padding: 20px;
      font-size: 1.2em;
    }
  </style>
</head>
<body>
  <div class="grid-container">
    <div class="grid-item">1</div>
    <div class="grid-item">2</div>
    <div class="grid-item">3</div>
    <div class="grid-item">4</div>
  </div>
</body>
```



grid-template-areas: A CSS justify-content and align-content in CSS Grid

- **justify-content:** Aligns the entire grid along the inline (row) axis.
 - **align-content:** Aligns the entire grid along the block (column) axis.
- Values:**
- **start:** Aligns the grid at the start of the container.
 - **end:** Aligns the grid at the end of the container.
 - **center:** Aligns the grid in the center of the container.
 - **stretch:** Stretches the grid to fill the container.
 - **space-between:** Distributes the grid items with space between them.
 - **space-around:** Distributes the grid items with space around them.
 - **space-evenly:** Distributes the grid items with equal space around them.

Practice Set

Grid

Instructions:

Below is an HTML structure for a simple grid layout. Using the specified CSS properties, create a responsive grid layout and apply basic transitions.

```
<!DOCTYPE html>
<html lang="en">
<head><title>Easy Grid Layout</title>
<style>
/* Your CSS code here */
</style>
</head>
<body>
<div class="container">
<div class="item item1">Item 1</div>
<div class="item item2">Item 2</div>
<div class="item item3">Item 3</div>
<div class="item item4">Item 4</div>
</div>
</body>
</html>
```

CSS Tasks:

- **Grid Template Rows & Columns:** Define the grid container with 2 rows and 2 columns, each taking up 1fr of space.
- **Grid Gap:** Set a gap of 10px between rows and columns.
- **Grid Item Placement:** Place each item in a separate cell.
- **Transitions:** Apply a transition to change the background color of .item when hovered.

Practice Set (Solution)

Grid

```
/* Grid Template Rows & Columns */
.container {
display: grid;
grid-template-rows: 1fr 1fr;
grid-template-columns: 1fr 1fr;
gap: 10px;
}

/* Grid Item Placement */
.item1 {
grid-row: 1 / 2;
grid-column: 1 / 2;
}

.item2 {
grid-row: 1 / 2;
grid-column: 2 / 3;
}

.item3 {
grid-row: 2 / 3;
grid-column: 1 / 2;
}

.item4 {
grid-row: 2 / 3;
grid-column: 2 / 3;
}

/* Transitions */
.item {
background-color: lightblue;
transition: background-color
0.5s ease-in-out;
}

.item:hover {
background-color: lightcoral;
}
```

Advanced Practice Set

Grid

Instructions:

Below is an HTML structure for a more complex grid layout with additional elements for using advanced grid properties. Using the specified CSS properties, create an advanced responsive grid layout.

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Advanced Grid Layout</title>
<style>
/* Your CSS code here */
</style>
</head>
<body>
<div class="grid-container">
<div class="grid-item item1">Item 1</div>
<div class="grid-item item2">Item 2</div>
<div class="grid-item item3">Item 3</div>
<div class="grid-item item4">Item 4</div>
<div class="grid-item item5">Item 5</div>
<div class="grid-item item6">Item 6</div>
</div>
</body>
</html>
```

Grid Template Areas: Define a grid layout with the following template areas:

"item1 item1 item2"

"item3 item4 item4"

"item5 item5 item6"

Grid Gap: Set a gap of **15px** between rows and columns.

Grid Item Placement: Place each item in its respective grid area.

Grid Span: Use the **grid-column** and **grid-row** properties with the **span** keyword to make items span multiple rows or columns.

Grid Overflow: Set the **overflow** property to handle content that overflows the boundaries of the grid items.

Advanced Practice Set (Solution)

Grid

```
/* Grid Template Areas */
.grid-container {
  display: grid;
  grid-template-areas:
    "item1 item1 item2"
    "item3 item4 item4"
    "item5 item5 item6";
  gap: 15px;
  width: 100%;
  height: 100vh;
}

/* Grid Overflow */
.grid-item {
  background-color: lightblue;
  padding: 20px;
  text-align: center;
  overflow: auto;
}

/* Grid Item Placement */
.item1 {
  grid-area: item1;
}

.item2 {
  grid-area: item2;
}

.item3 {
  grid-area: item3;
}

.item4 {
  grid-area: item4;
}

.item5 {
  grid-area: item5;
}

.item6 {
  grid-area: item6;
}

/* Grid Span */
.item1 {
  grid-column: 1 / span 2;
}
.item2 {
  grid-column: 3 / span 1;
  grid-row: 1 / span 2;
}
.item3 {
  grid-row: 2 / span 1;
}
.item4 {
  grid-column: 2 / span 2;
  grid-row: 2 / span 2;
}
.item5 {
  grid-column: 1 / span 2;
  grid-row: 3 / span 1;
}
.item6 {
  grid-column: 3 / span 1;
  grid-row: 3 / span 1;
}
```



Introduction to JavaScript

- History of JavaScript
- What is JavaScript
- Popularity of JavaScript
- Applications of JavaScript
- Runtime Environment
- JavaScript vs ECMA
- JavaScript vs TypeScript
- JavaScript in Console
- JavaScript in Webpage
- DOM Manipulation
- JavaScript with Node



History of JavaScript



1. JavaScript was originally named Mocha, then renamed to LiveScript, and finally JavaScript to capitalize on the popularity of Java at the time.
2. JavaScript was created by Brendan Eich in 1995 while he was working at Netscape Communications Corporation.
3. JavaScript is an interpreted language, meaning it is executed line by line.

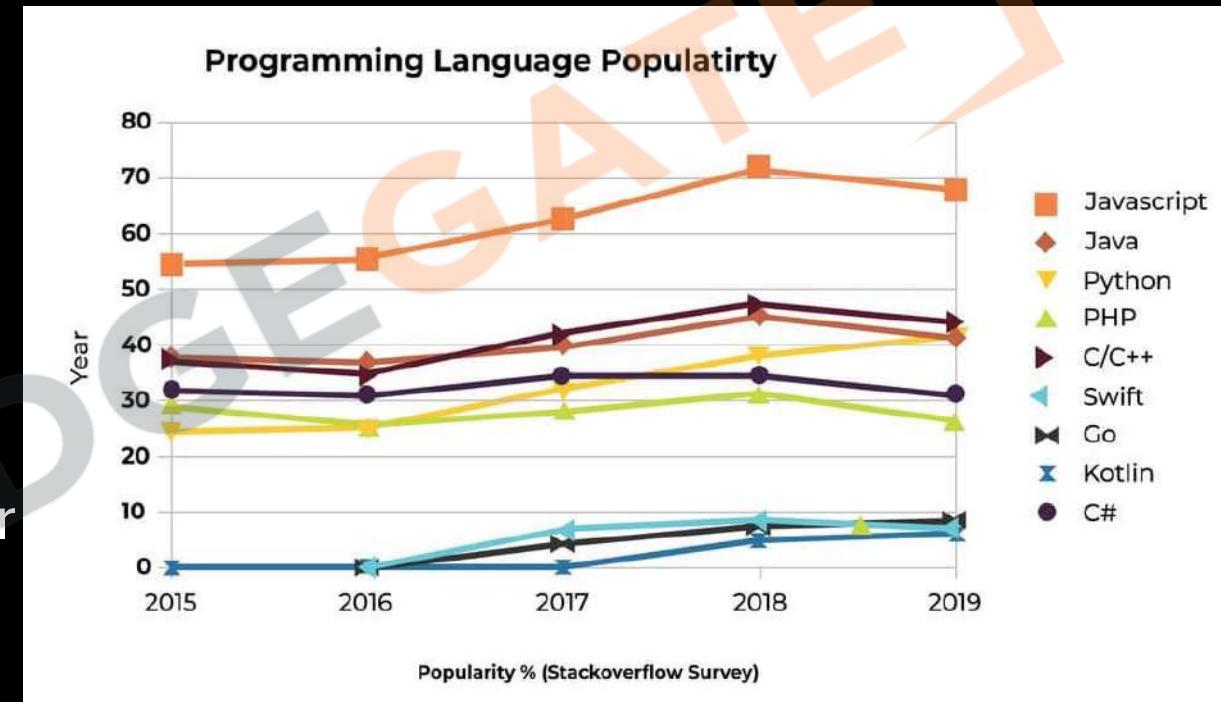
What is JavaScript

1. **JavaScript** is a high-level, **dynamic** programming language commonly used for creating interactive effects within web browsers.
2. **Actions:** Enables **interactivity**.
3. **Updates:** Alters page **without** reloading.
4. **Events:** Responds to user actions.
5. **Data:** Fetches and sends info to server.



Popularity of JavaScript

1. **JavaScript** is one of the most popular programming languages in the world, consistently ranking at the top in surveys and job listings.
2. Average **JavaScript** Dev Salary in India:
 - Entry-Level (0-1 year): Around ₹3,50,000 per annum.
 - Mid-Level (2-5 years): Approximately ₹6,00,000 to ₹10,00,000 per annum.
 - Experienced (5+ years): Can exceed ₹10,00,000 per annum, potentially reaching up to ₹20,37,500.

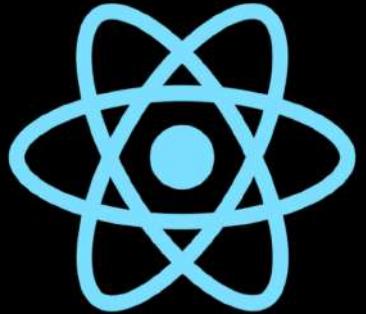


Applications of JavaScript



1. **HTML**: Defines the **structure and content** of the **website**.
2. **CSS**: Specifies the **appearance and layout** of the **website**.
3. **JavaScript**: Adds **interactivity and dynamic behavior** to the **website**.

Applications of JavaScript



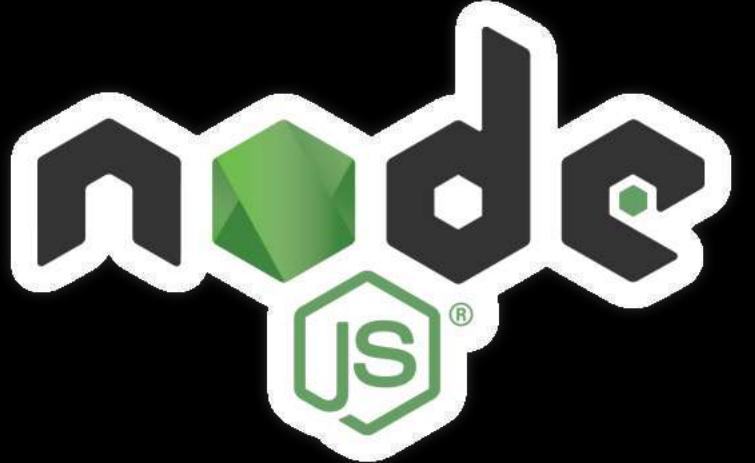
React JS



Web Applications:

- **React**: A library for **building user interfaces**, maintained by **Facebook**.
- **Angular**: A platform for building **mobile and desktop web applications**, maintained by **Google**.
- **Vue.js**: A progressive framework for building user interfaces.

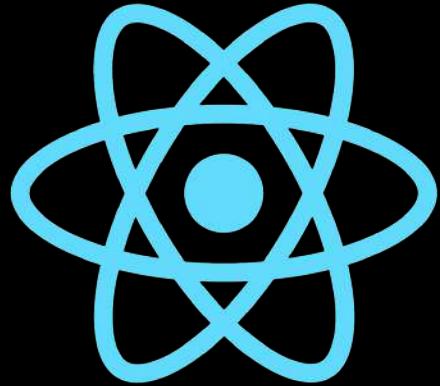
Applications of JavaScript



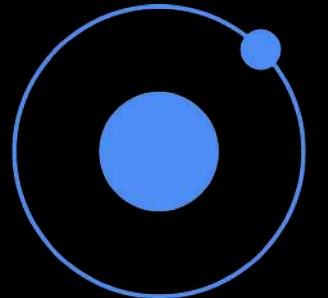
Server-Side:

- **Node.js:** Allows JavaScript to run on the server, used for building scalable network applications.
- **Express.js:** A minimal and flexible Node.js web application framework.

Applications of JavaScript



React Native



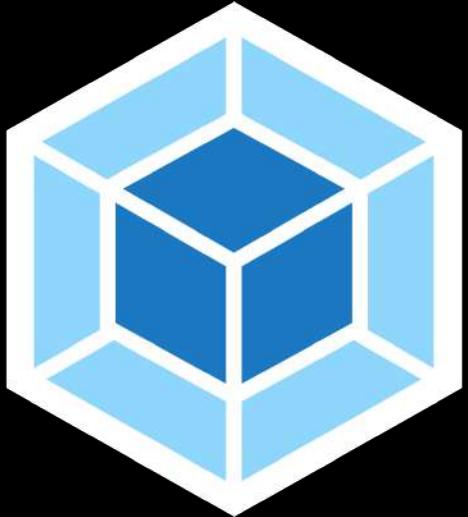
ionic



Mobile Applications:

- **React Native:** Builds **mobile apps** using **JavaScript** and **React**.
- **Ionic:** A framework for building **cross-platform mobile apps** with **web technologies** like **HTML**, **CSS**, and **JavaScript**.
- **NativeScript:** Allows building **native iOS** and **Android apps** using **JavaScript** or **TypeScript**.

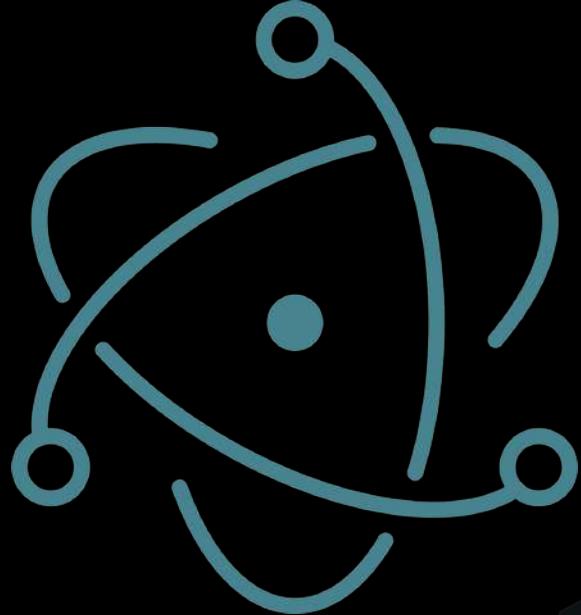
Applications of JavaScript



BuildTools:

- **Webpack**: A **module bundler** for JavaScript applications.
- **Parcel**: A fast, **zero-configuration** web application bundler.
- **Gulp**: A toolkit to **automate tasks** in your development workflow.

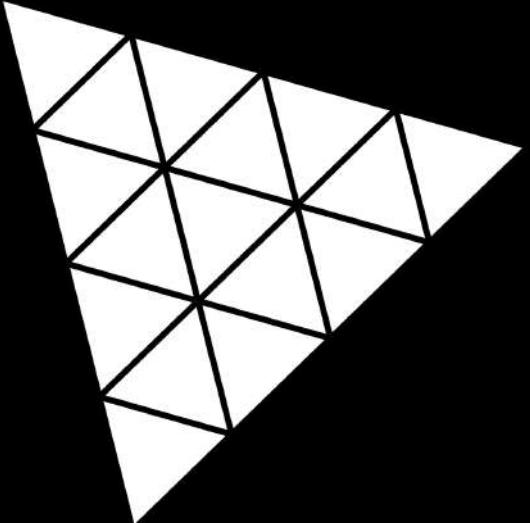
Applications of JavaScript



Desktop Applications:

- **Electron:** Allows building cross-platform desktop applications using HTML, CSS, and JavaScript.
- **NW.js:** A framework for building native applications with web technologies.

Applications of JavaScript

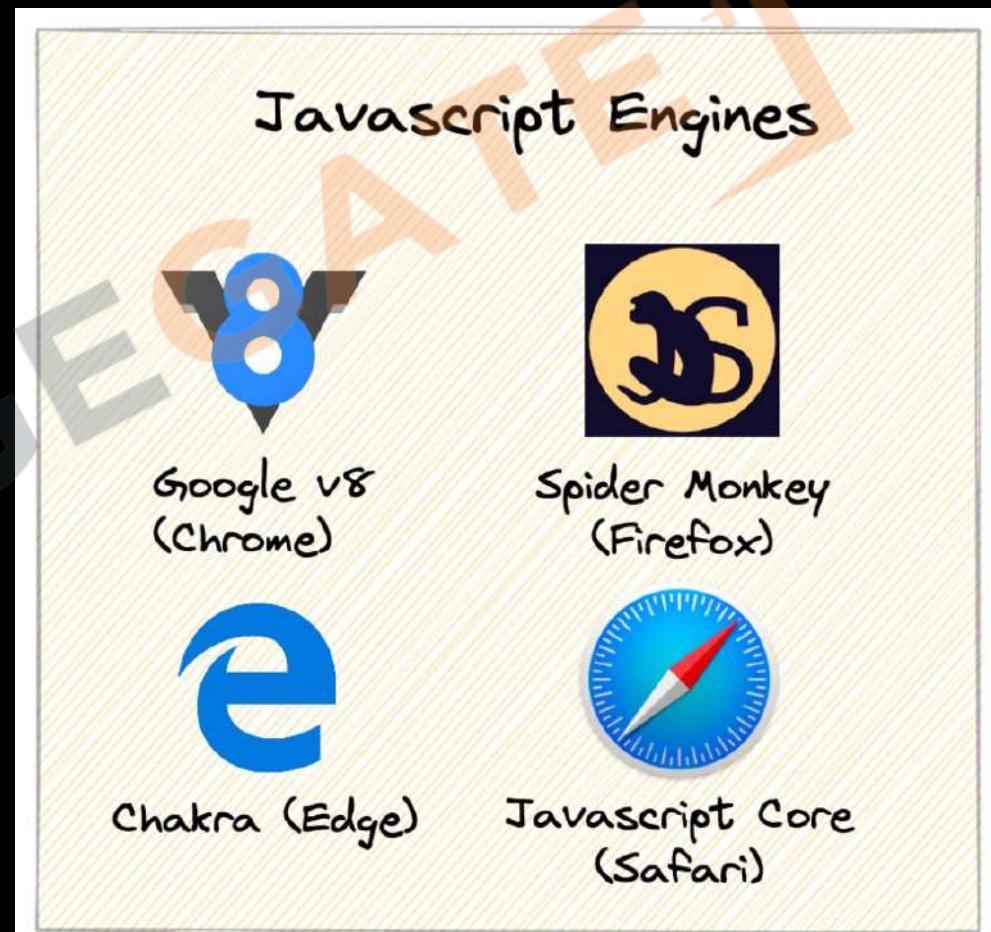


Cameras and Speakers:

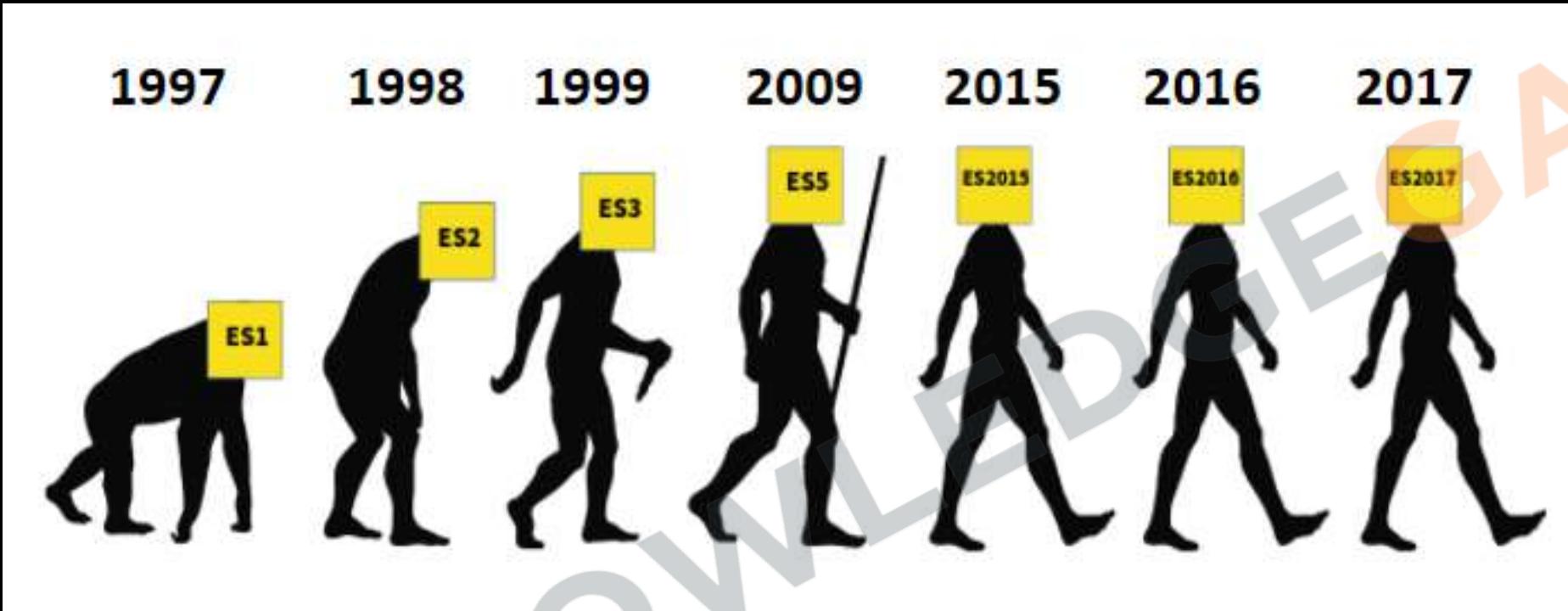
- **Three.js**: A library that makes **WebGL** - 3D programming for the web - easier to use.
- **WebRTC**: A technology that **enables peer-to-peer** audio, video, and data sharing.
- **Howler.js**: A JavaScript **audio library** for the modern web.

Runtime Environment

1. Provides infrastructure to execute JavaScript code.
2. Core: Includes a JavaScript engine (e.g., V8, SpiderMonkey).
3. Browser Environment: Offers APIs for DOM manipulation, events, and network requests.
4. Node.js: Extends JavaScript capabilities to server-side programming.
5. Asynchronous Support: Handles non-blocking operations with event loops, callbacks, and promises.

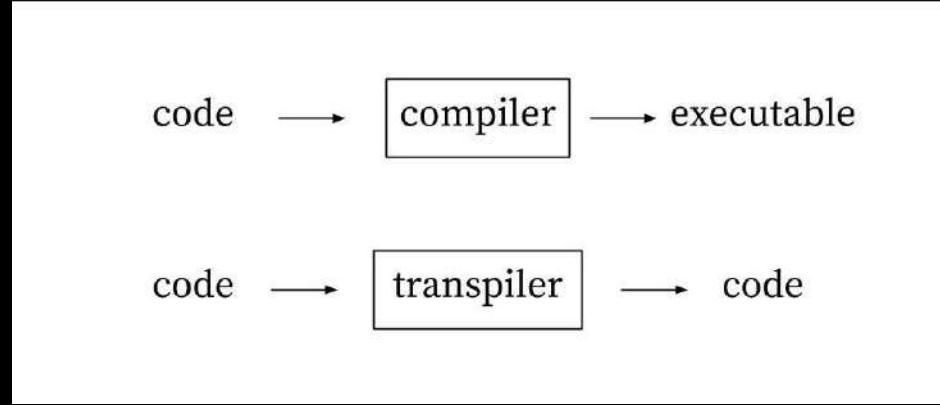


JavaScript vs ECMA

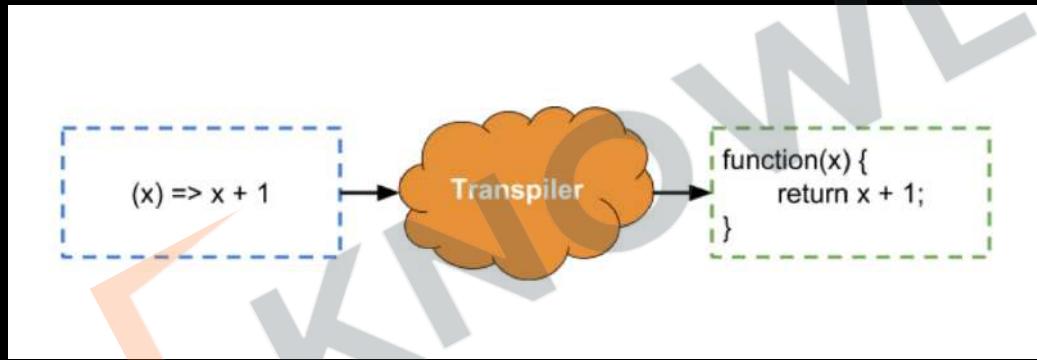


1. ECMAScript is the **standardized specification** developed by **ECMA International** that defines the **core features**, **syntax**, and **functionalities** of JavaScript and similar scripting languages.
2. JavaScript is the **actual language** implementation.

JavaScript vs TypeScript



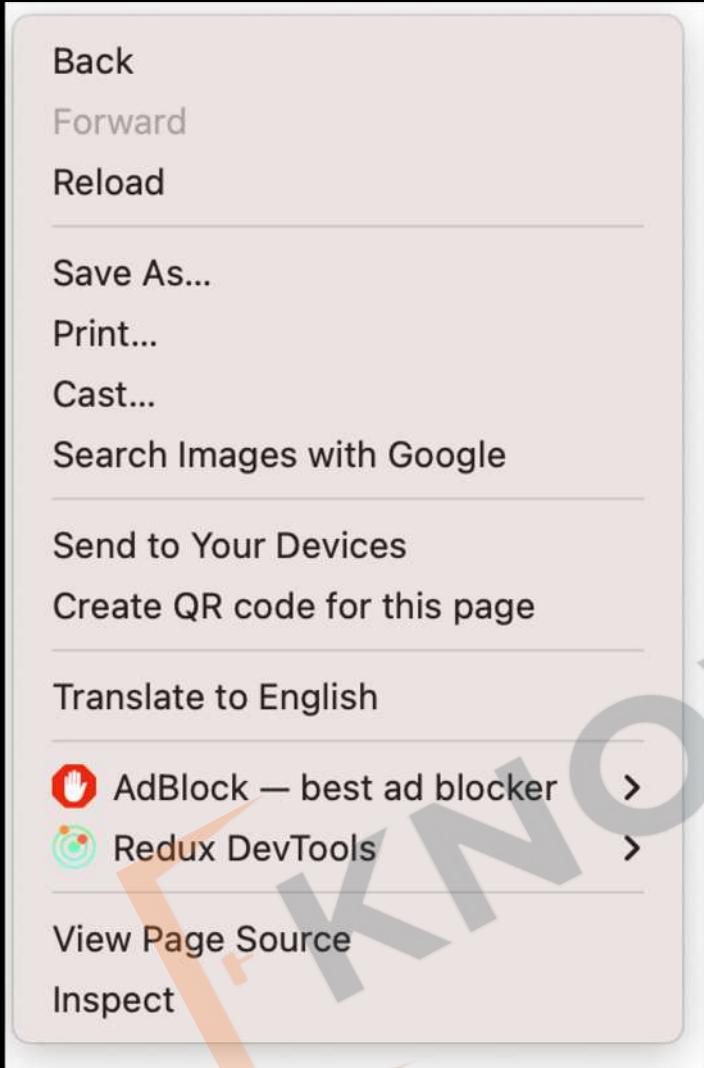
1. JavaScript runs at the **client** side in the browser.
2. Coffee Script / **TypeScript** are **transpiled** to JavaScript.



JavaScript vs TypeScript

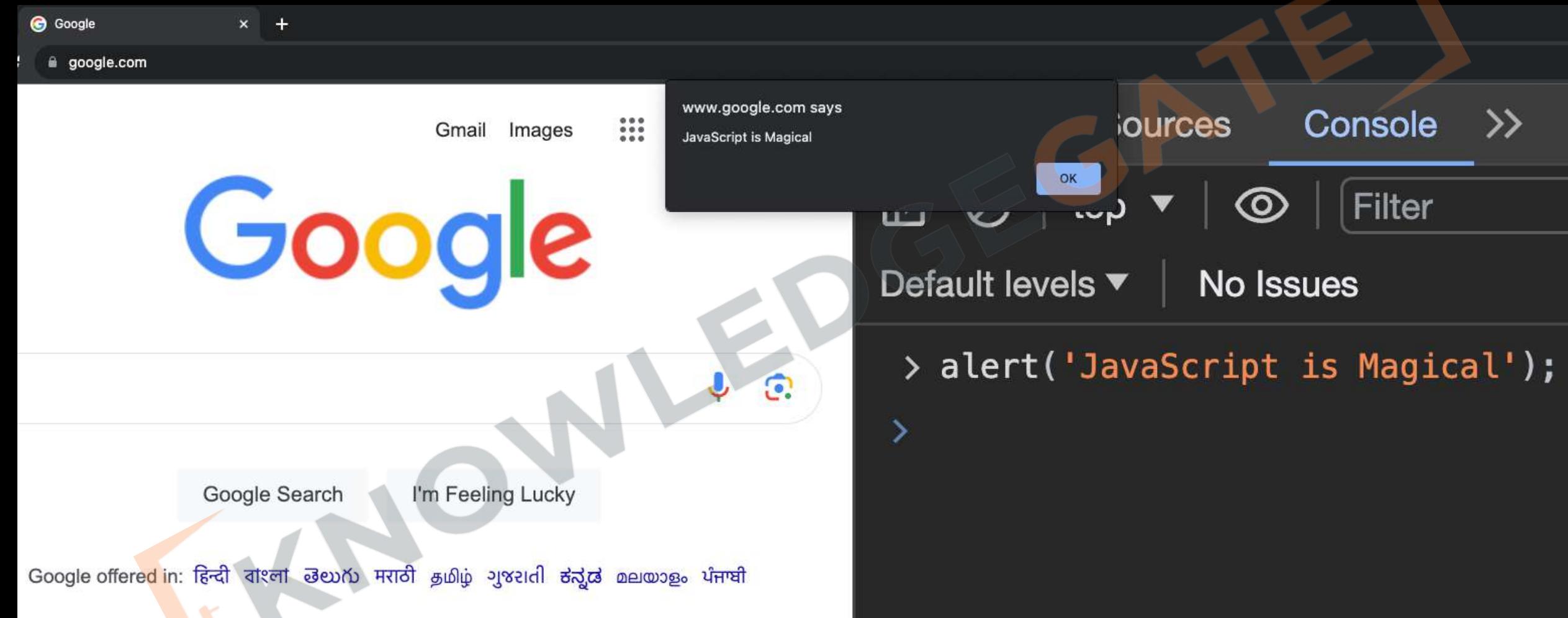
Feature	JavaScript (JS)	TypeScript (TS)
Definition	A dynamic, high-level scripting language.	A statically typed superset of JavaScript.
Typing	Dynamically typed.	Statically typed with optional type annotations.
Compilation	Interpreted by browsers.	Transpiles to JavaScript before execution.
Error Detection	Errors detected at runtime.	Errors caught at compile-time.
Tooling Support	Basic tooling, less support for large-scale projects.	Enhanced tooling support with features like IntelliSense.
Learning Curve	Easier to learn for beginners.	Slightly steeper learning curve due to static typing.
Code Maintenance	Can be harder to maintain and debug in large codebases.	Easier to maintain and refactor due to static types.
Development Speed	Faster for small projects and prototyping.	Potentially slower initial development but saves time in the long run with fewer bugs.
Community and Usage	Widely used, especially in web development.	Growing rapidly, especially in large-scale applications.
Example Usage	<code>var x = 10;</code>	<code>let x: number = 10;</code>

JavaScript in Console (Inspect)

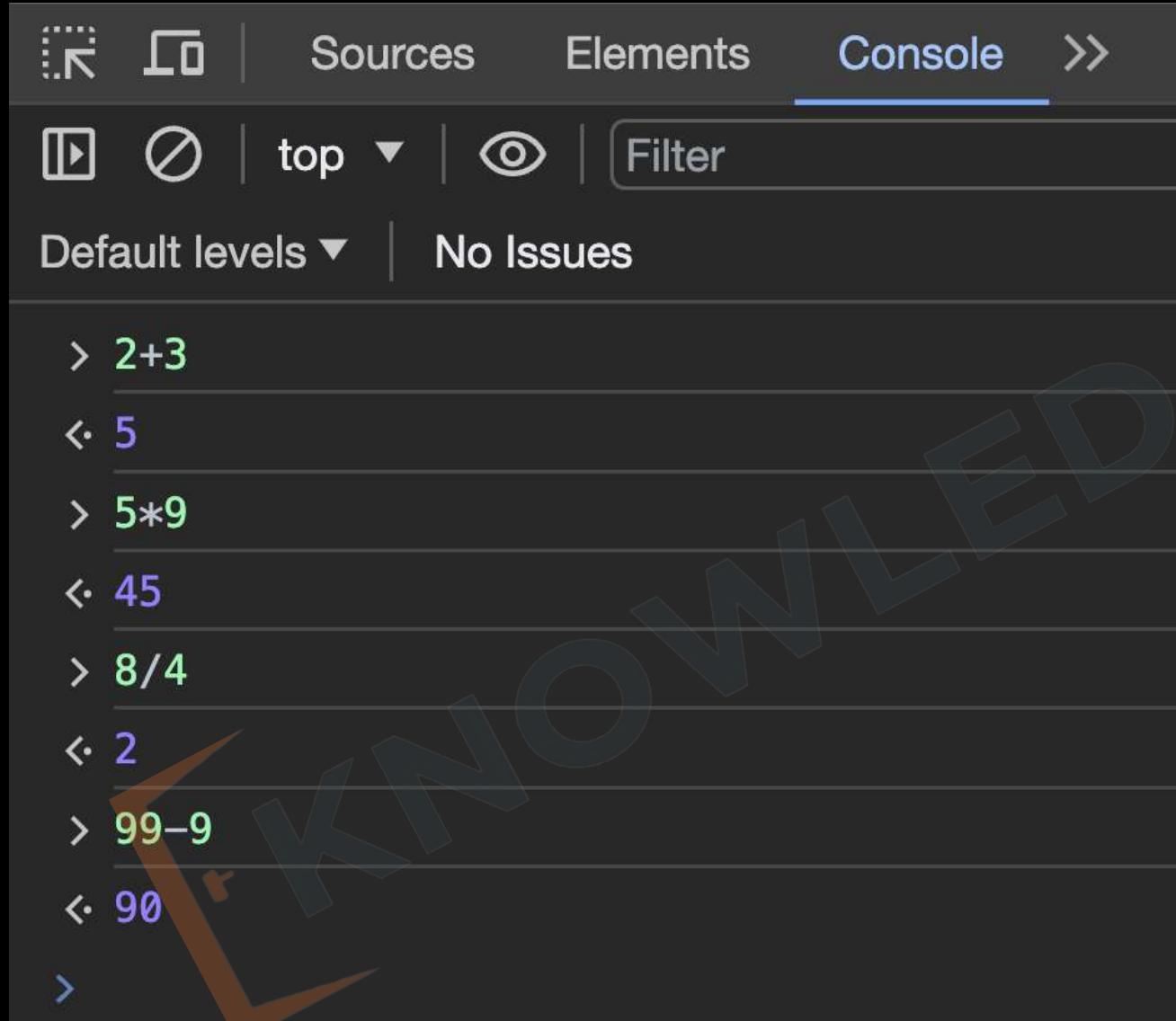


1. Allows real-time editing of **HTML/CSS/JS**
2. **Run Scripts:** Test code in console.
3. **Debug:** Locate and fix errors.
4. **Modify DOM:** Change webpage elements.
Errors: View error messages.

JavaScript in Console (Alert)



JavaScript in Console (Math)

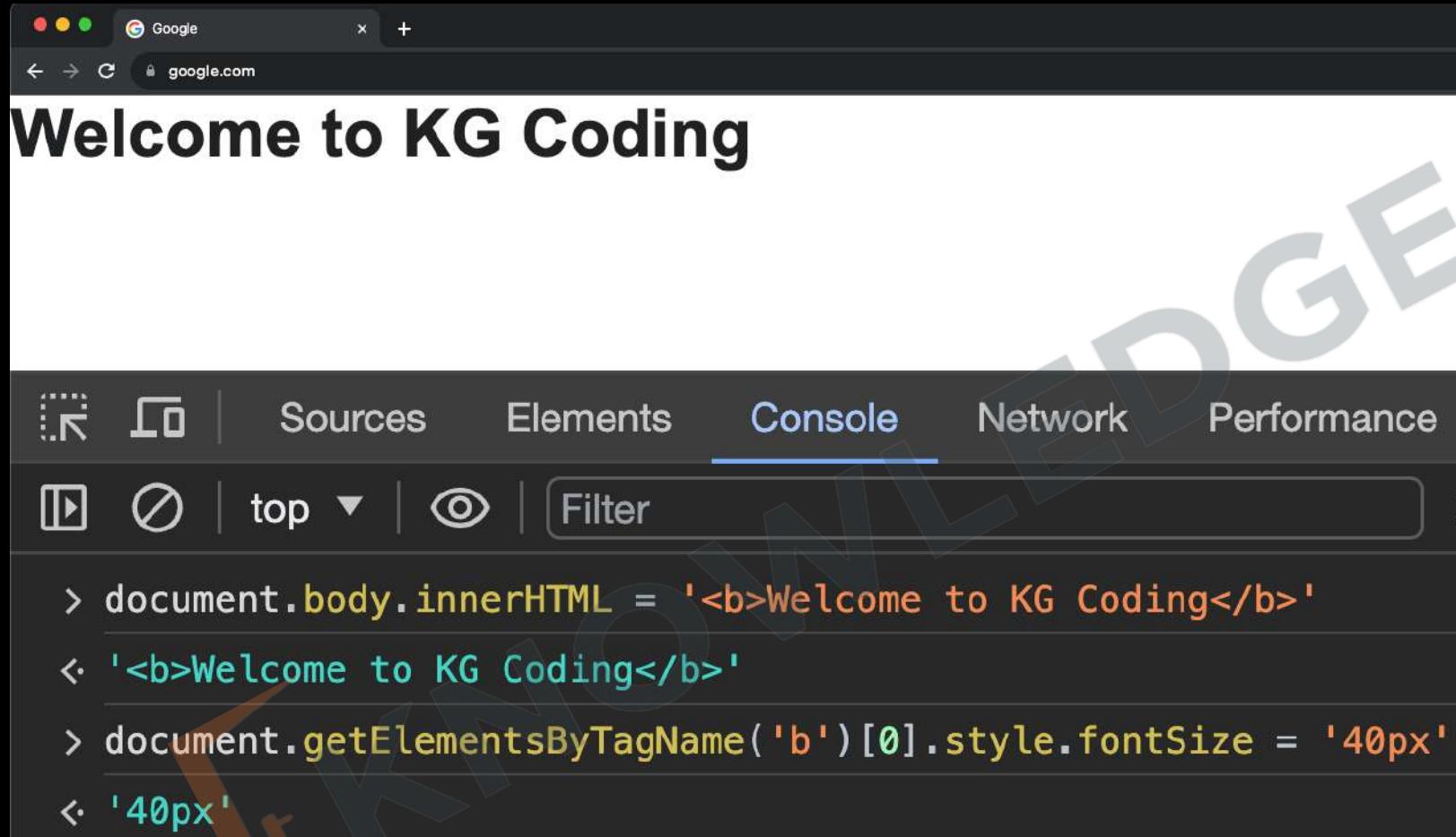


A screenshot of a browser's developer tools interface, specifically the 'Console' tab. The console window shows a series of JavaScript expressions and their results:

- > $2+3$
← 5
- > $5*9$
← 45
- > $8/4$
← 2
- > $99-9$
← 90

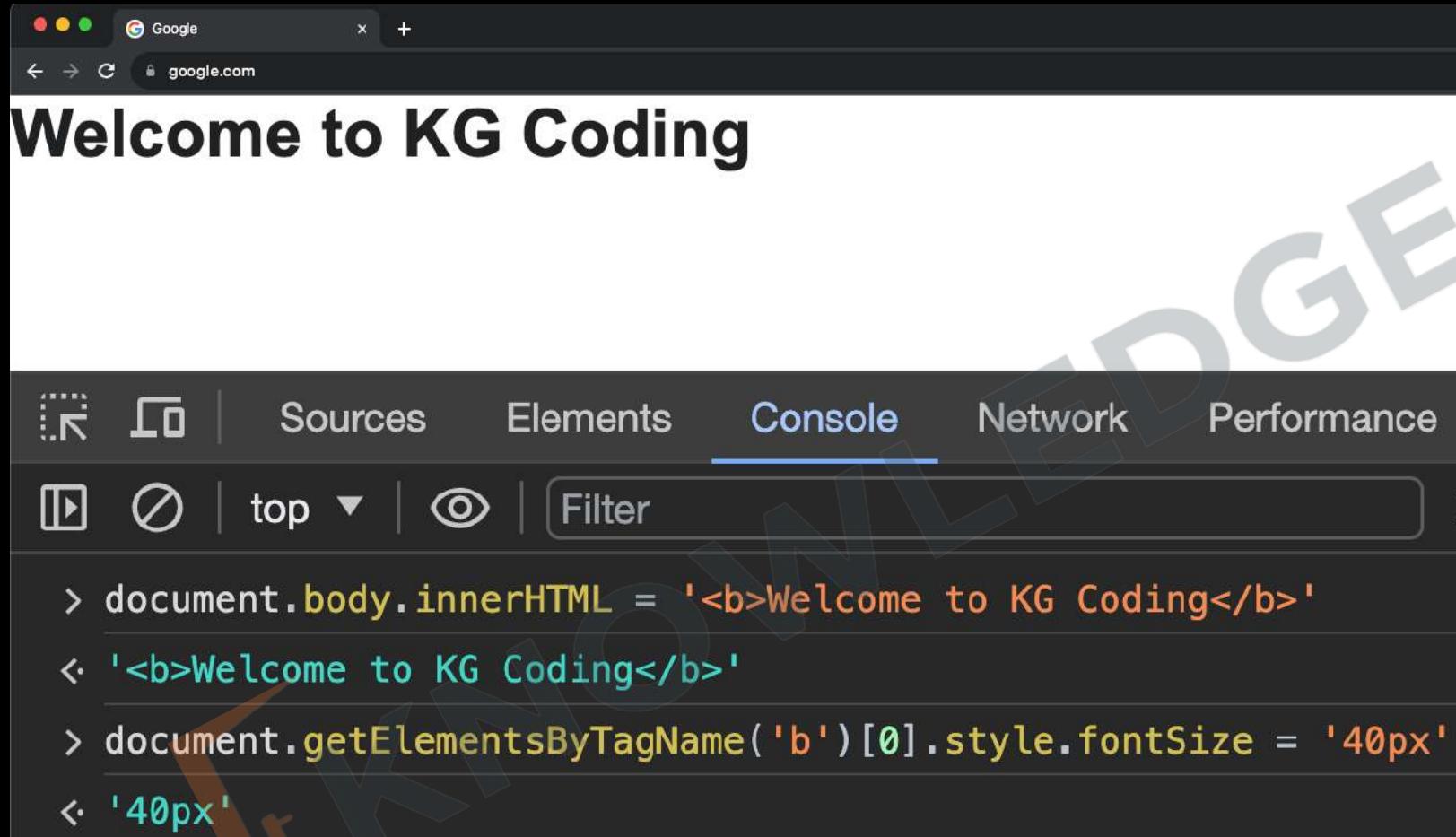
Console can be used
as a Calculator

DOM Manipulation

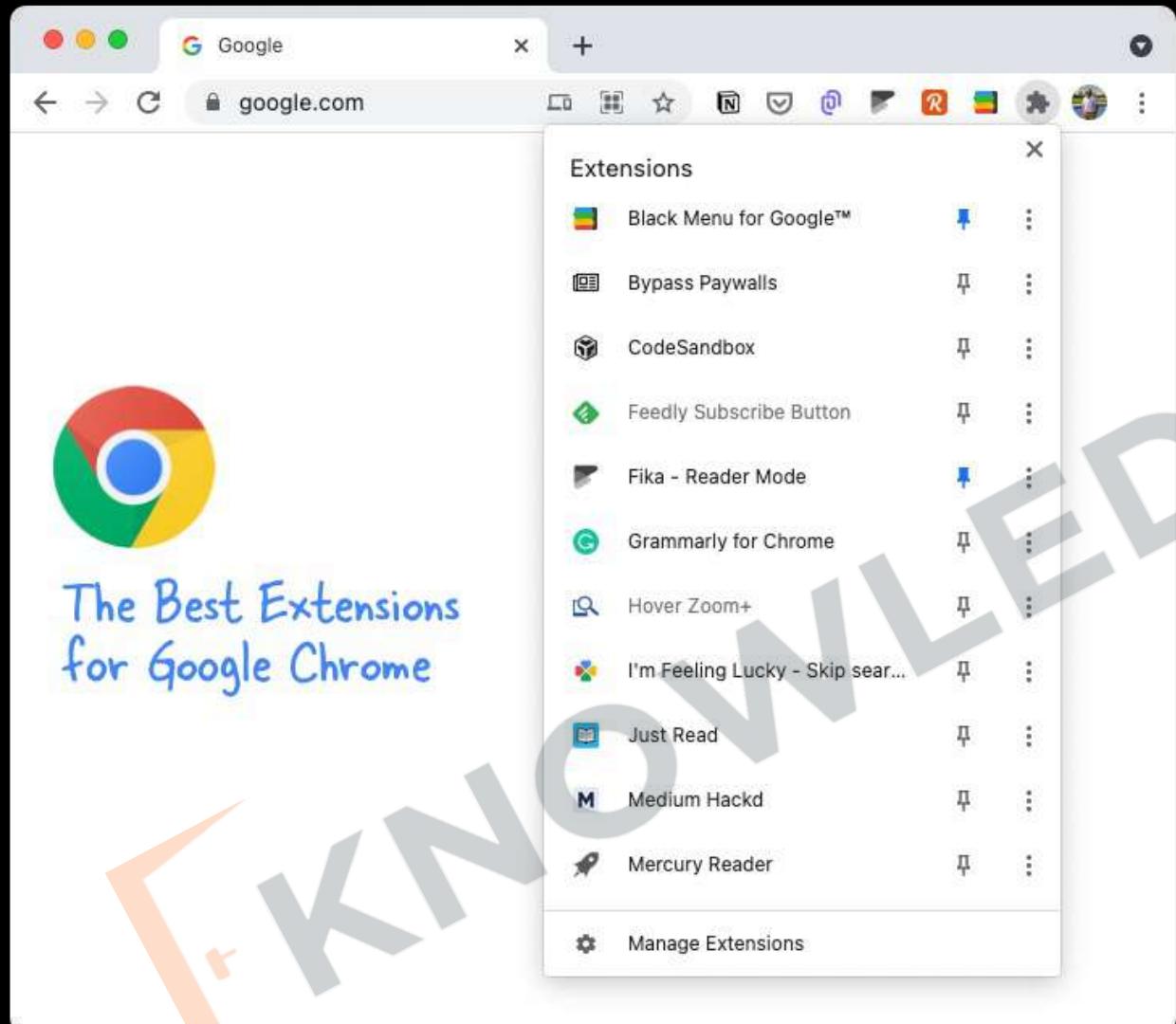


A screenshot of a browser window showing the developer tools console tab selected. The page content is "Welcome to KG Coding". The console shows the following JavaScript code:

```
> document.body.innerHTML = '<b>Welcome to KG Coding</b>'  
< <b>Welcome to KG Coding</b>  
> document.getElementsByTagName('b')[0].style.fontSize = '40px'  
< '40px'
```

- 
1. Change HTML
 2. Change CSS
 3. Perform Actions

Chrome Extensions



1. Create Features: Add new functionalities to Chrome.
2. Interact with Web: Modify or read webpage content.
3. API Access: Use Chrome's built-in functions.
4. User Experience: Enhance or customize browsing.

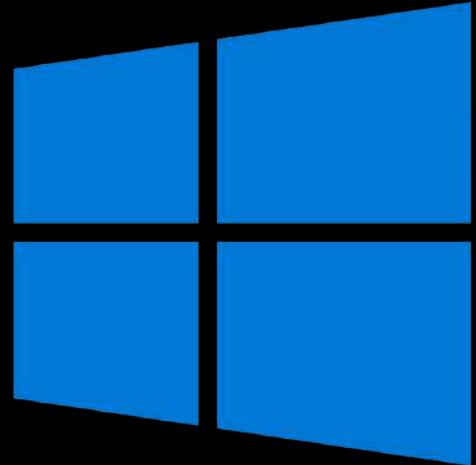
Practice Exercise

1. Use an **alert** to display Good Morning.
2. Display your name in a **popup**.
3. Using Math calculate the following:
=> **75-25**
=> **3+3-5**
4. Change **Facebook** page to display “I am Learning JS”



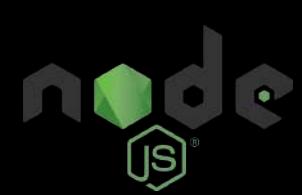


NodeJS Windows Setup



Windows





Windows Setup (Install latest Node)

Search Download NodeJS

nodejs.org/en/download

KG Tools youtube Resume ChatGPT GitHub whatsapp direct m... KG Coding LeetCode Morgan Stanley Lo... Notes KG Coding

Learn About Download Blog Docs ↗ Certification ↗

Download Node.js®

Download Node.js the way you want.

Prebuilt Installer Prebuilt Binaries Package Manager Source Code

I want the v22.1.0 (Current) version of Node.js for Windows running x64

[Download Node.js v22.1.0](#)

Node.js includes npm (10.7.0) ↗
Read the changelog for this version ↗
Read the blog post for this version ↗
Learn how to verify signed SHASUMS ↗
Check out all available Node.js download options ↗
Learn about Node.js Releases ↗

Windows Setup

(Install VsCode)



Search VS Code on Google

Code editing.
Redefined.

Free. Built on open source. Runs everywhere.

Download for Windows
Stable Build

Web, Insiders edition, or other platforms

By using VS Code, you agree to its
license and privacy statement.

A screenshot of the Visual Studio Code interface. The main area shows a file named 'serviceWorker.js' with code related to a service worker. The code includes imports from 'register.js' and 'navigator.serviceWorker'. A tooltip is shown over the 'serviceWorker' property. The left sidebar shows the 'EXTENSIONS: MARKETPLACE' section with various extensions listed, such as Python, GitLens, C/C++, ESLint, Debugger for Chrome, Language Support, vscode-icons, Vetur, and C#. The bottom right corner of the screen displays a watermark with the word 'KNOW'.

```
src > JS serviceWorker.js > register > window.addEventListener('load') callback
checkValidServiceWorker(swUrl, config);

// Add some additional logging to localhost, per
// service worker/PWA documentation.
navigator.serviceWorker.ready.then(() => {

    product
    productSub
    removeSiteSpecificTrackingException
    removeWebWideTrackingException
    requestMediaKeySystemAccess
    sendBeacon
    serviceWorker (property) Navigator.serviceWorker
    storage
    storeSiteSpecificTrackingException
    storeWebWideTrackingException
} userAgent
vendor

function registerValidSW(swUrl, config) {
    navigator.serviceWorker
        .register(swUrl)
        .then(registration => {
```

TERMINAL

```
1:node
```

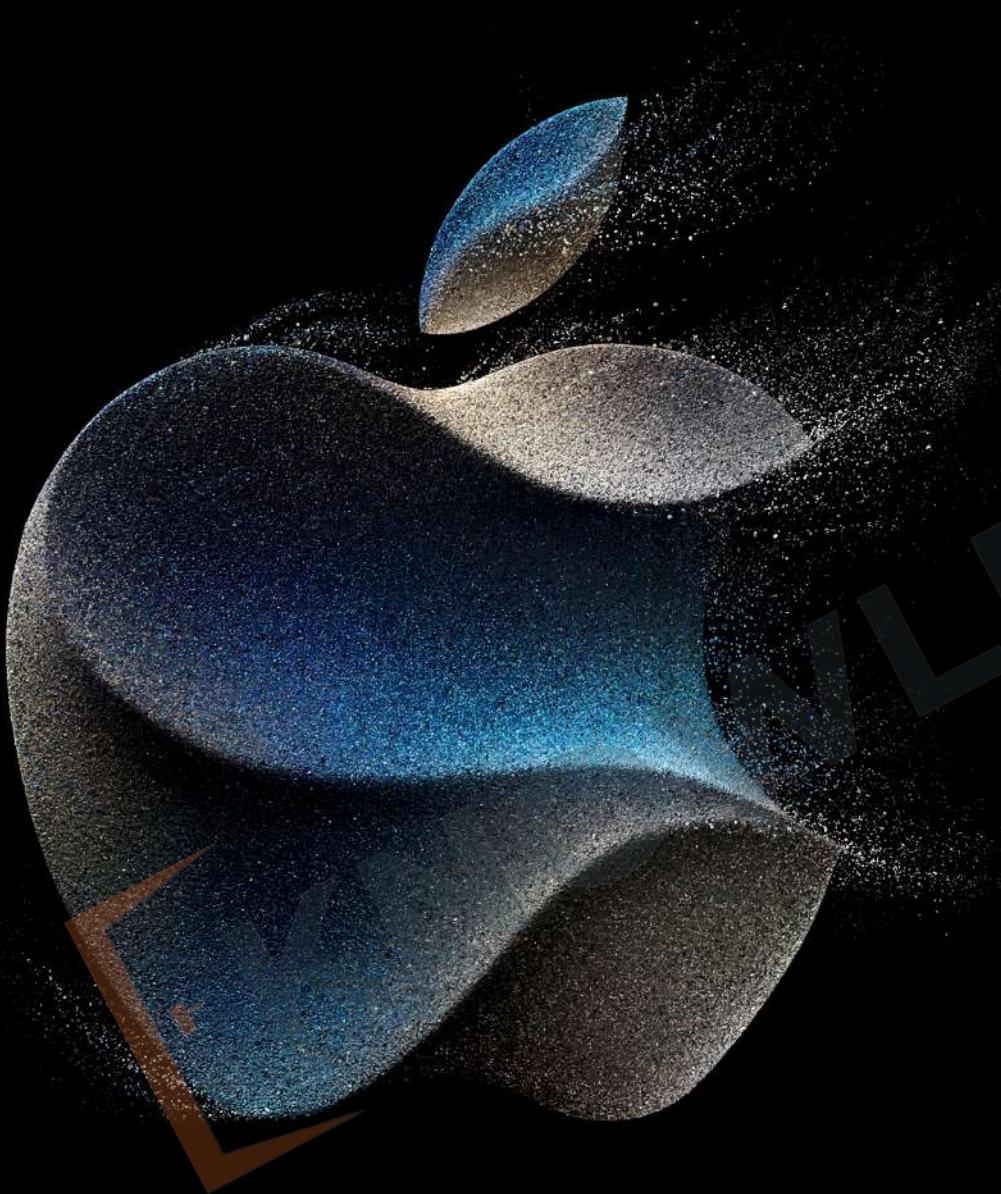
You can now view `create-react-app` in the browser.

Local: http://localhost:3000/
On Your Network: http://10.211.55.3:3000/

Note that the development build is not optimized.

In 43, Col 19 Spaces: 2 UTF-8 LF JavaScript

NodeJS MAC Setup





MAC Setup

(Install latest Node)

Search Download NodeJS

nodejs.org/en/download

Rare READ KG Tools youtube Resume ChatGPT GitHub whatsapp direct m... KG Coding LeetCode Morgan Stanley Lo... Notes KG Coding

node Learn About Download Blog Docs ↗ Certification ↗

Download Node.js®

Download Node.js the way you want.

Prebuilt Installer Prebuilt Binaries Package Manager Source Code

I want the v22.1.0 (Current) version of Node.js for macOS running on my Mac

[Download Node.js v22.1.0](#)

Node.js includes npm (10.7.0) ↗
Read the changelog for this version ↗
Read the blog post for this version ↗
Learn how to verify signed SHASUMS ↗
Check out all available Node.js download options ↗
Learn about Node.js Releases ↗

MAC Setup

(Install VsCode)



Search VS Code on Google

Code editing.
Redefined.

Free. Built on open source. Runs everywhere.

Download Mac Universal

Stable Build

Web, Insiders edition, or other platforms

By using VS Code, you agree to its
license and privacy statement.

The screenshot shows the Visual Studio Code interface. On the left, the Extensions Marketplace sidebar lists several extensions: Python, GitLens, C/C++, ESLint, Debugger for Chrome, Language Support for Java, vscode-icons, and Vetur. The main editor area displays a JavaScript file named 'blog-post.js' with code related to Gatsby. The bottom right corner shows the terminal window outputting build logs for a GraphQL compilation.

```
blog-post.js - density-graphql-app
src > components > JS blog-post.js > <function> > blogPost
  import { graphql } from "gatsby"
  import React from "react"
  import Image from "gatsby-image"

  export default ({ data }) => {
    const blogPost = data.cms.blogPost
    return (
      <div>
        <blogPost>
          <blogPos>
            <blogPos>
              <Imag>
                <decodeURIComponent>
                  <Imag>
                    <default>
                      <h1><blog>
                        <defaultStatus>
                          <div><Post>
                            <delete>
                              <div><dang>
                                <departFocus>
                                  <devicePixelRatio>
                                    <dispatchEvent>

```

PROBLEMS TERMINAL ... 2: Task - develop

```
Info : [wdm]: Compiling...
DONE! Compiled successfully in 26ms
3:57:58 PM
```

Info : [wdm]: Compiled successfully.

En 6, Col 21 Spaces: 2 UTF-8 LF JavaScript

Executing first .js file

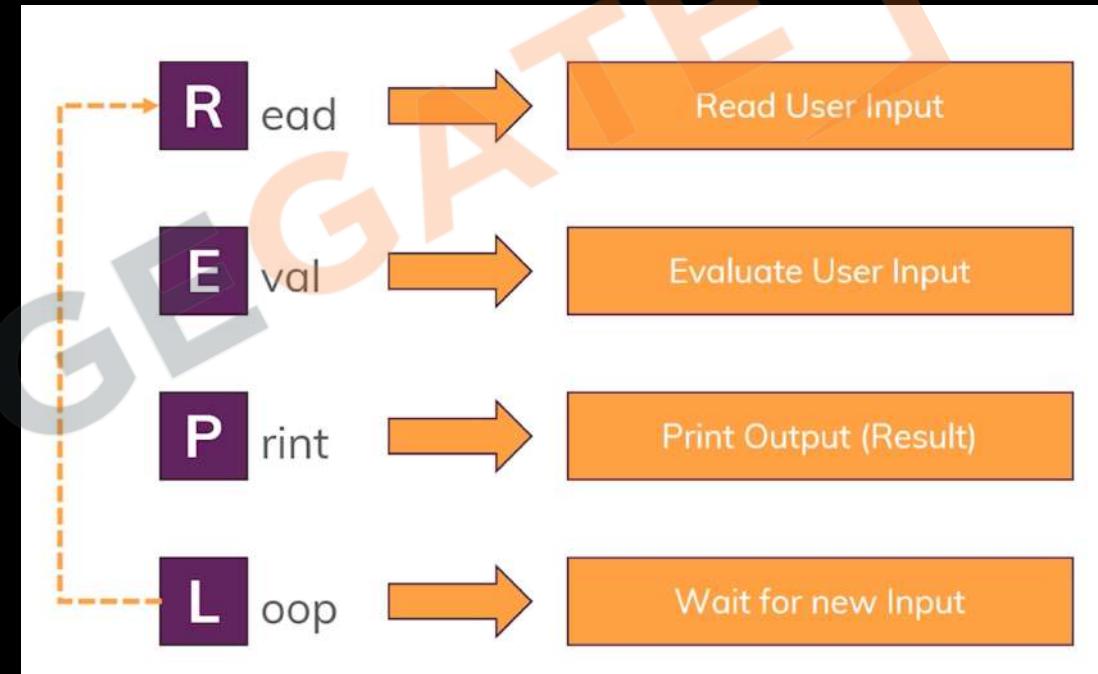
```
1 const fs = ...  
2  
3 // Define two variables  
4 let a = 10;  
5 let b = 5;  
6  
7 // Basic arithmetic operations  
8 let sum = a + b;  
9 let product = a * b;  
10  
11 // Prepare data to write  
12 let data = `Sum: ${sum}\nProduct: ${product}`;  
13 console.log(data);  
14  
15 // Write data to a local file  
16 fs.writeFile('output.txt', data, (err) => {  
17   if (err) throw err;  
18   console.log('Data written to file');  
19});
```

```
prashantjain@Mac-mini Desktop % node test.js  
Sum: 15  
Product: 50  
Data written to file
```

1. Streamlines Node Command: Use `node filename.js` to execute a JavaScript file in the Node.js environment.
2. Require Syntax: Use `require('module')` to include built-in or external modules, or other JavaScript files in your code.
3. Modular Code: require helps organize code into reusable modules, separating concerns and improving maintainability.
4. Caching: Modules loaded with require are cached, meaning the file is executed only once even if included multiple times.

What is REPL

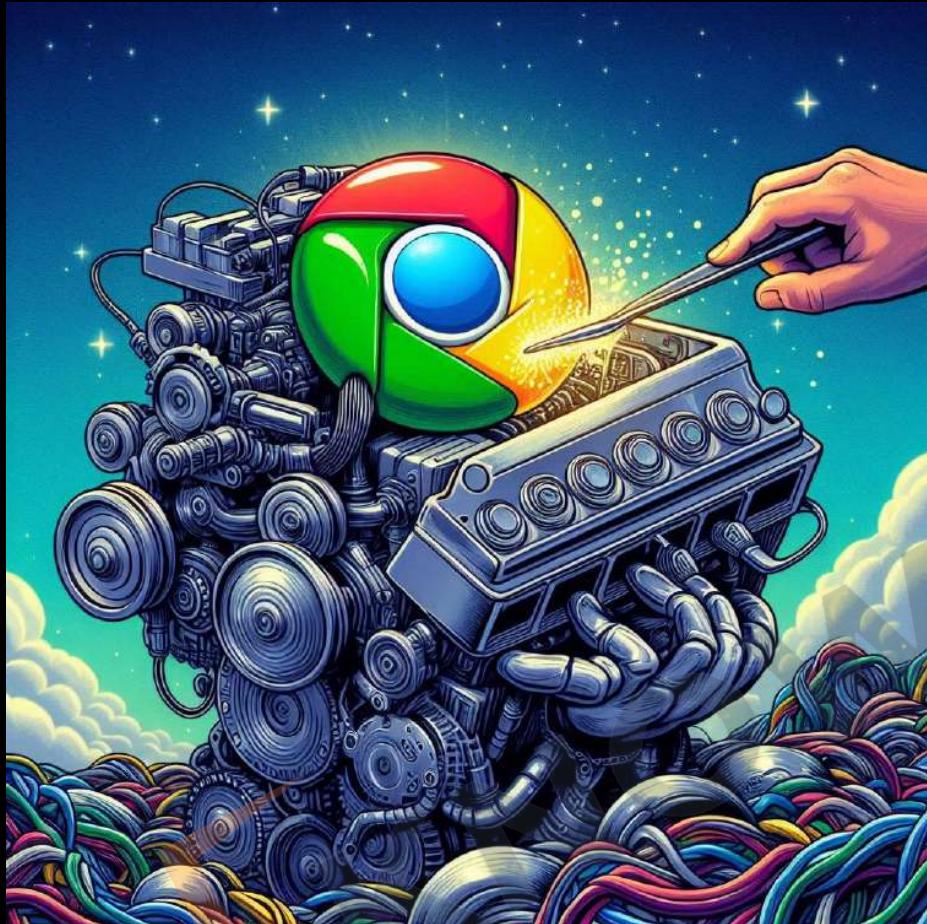
1. Streamlines Interactive Shell: Executes JavaScript code **interactively**.
2. Quick Testing: Ideal for **testing and debugging code snippets** on the fly.
3. Built-in Help: Offers help commands **via .help**.
4. Session Management: Supports saving (**.save**) and loading (**.load**) code sessions.
5. Node.js API Access: Provides **direct access** to Node.js APIs for experimentation.
6. Customizable: Allows **customization of prompt** and behaviour settings.



Executing Code via REPL

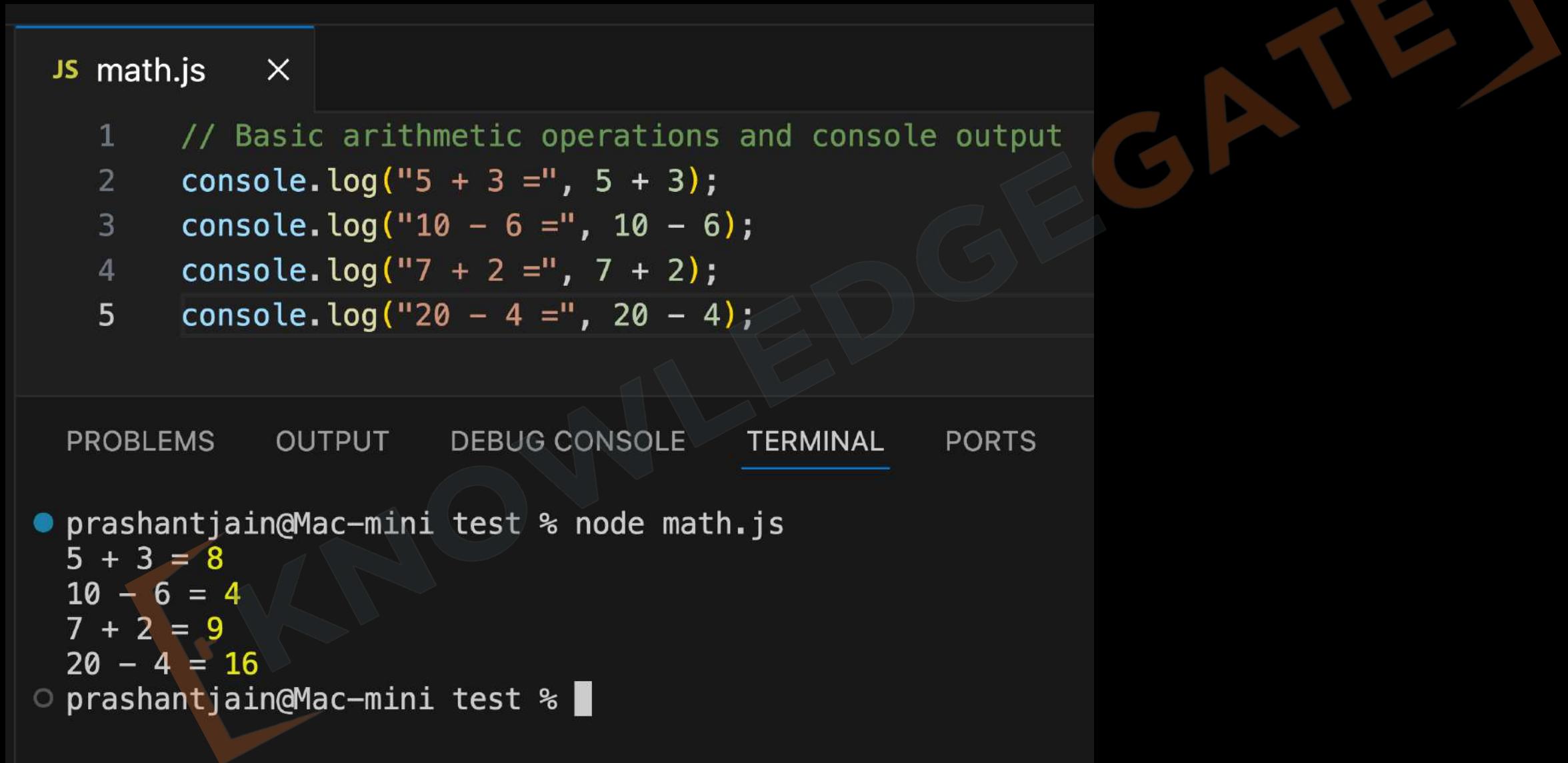
```
prashantjain@Mac-mini Desktop % node
Welcome to Node.js v20.9.0.
Type ".help" for more information.
> 5 + 6
11
> console.log('KG Coding is the best');
KG Coding is the best
undefined
> fs.writeFile('output.txt', 'Writing to file', (err) => {
...     if (err) throw err;
...     console.log('Data written to file');
... });
undefined
> Data written to file
```

JavaScript with Node



1. **JavaScript Runtime:** Node.js is an **open-source, cross-platform** runtime environment for executing **JavaScript code outside of a browser.**
2. **NodeJs** is a **JavaScript in a different environment** means **Running JS on the server or any computer.**
3. **Built on Chrome's V8 Engine:** It runs on the **V8 engine**, which **compiles JavaScript directly to native machine code**, enhancing performance.
4. **V8** is written in **C++** for speed.
5. **V8 + Backend Features = NodeJs**

JavaScript with Node



```
JS math.js ×  
1 // Basic arithmetic operations and console output  
2 console.log("5 + 3 =", 5 + 3);  
3 console.log("10 - 6 =", 10 - 6);  
4 console.log("7 + 2 =", 7 + 2);  
5 console.log("20 - 4 =", 20 - 4);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

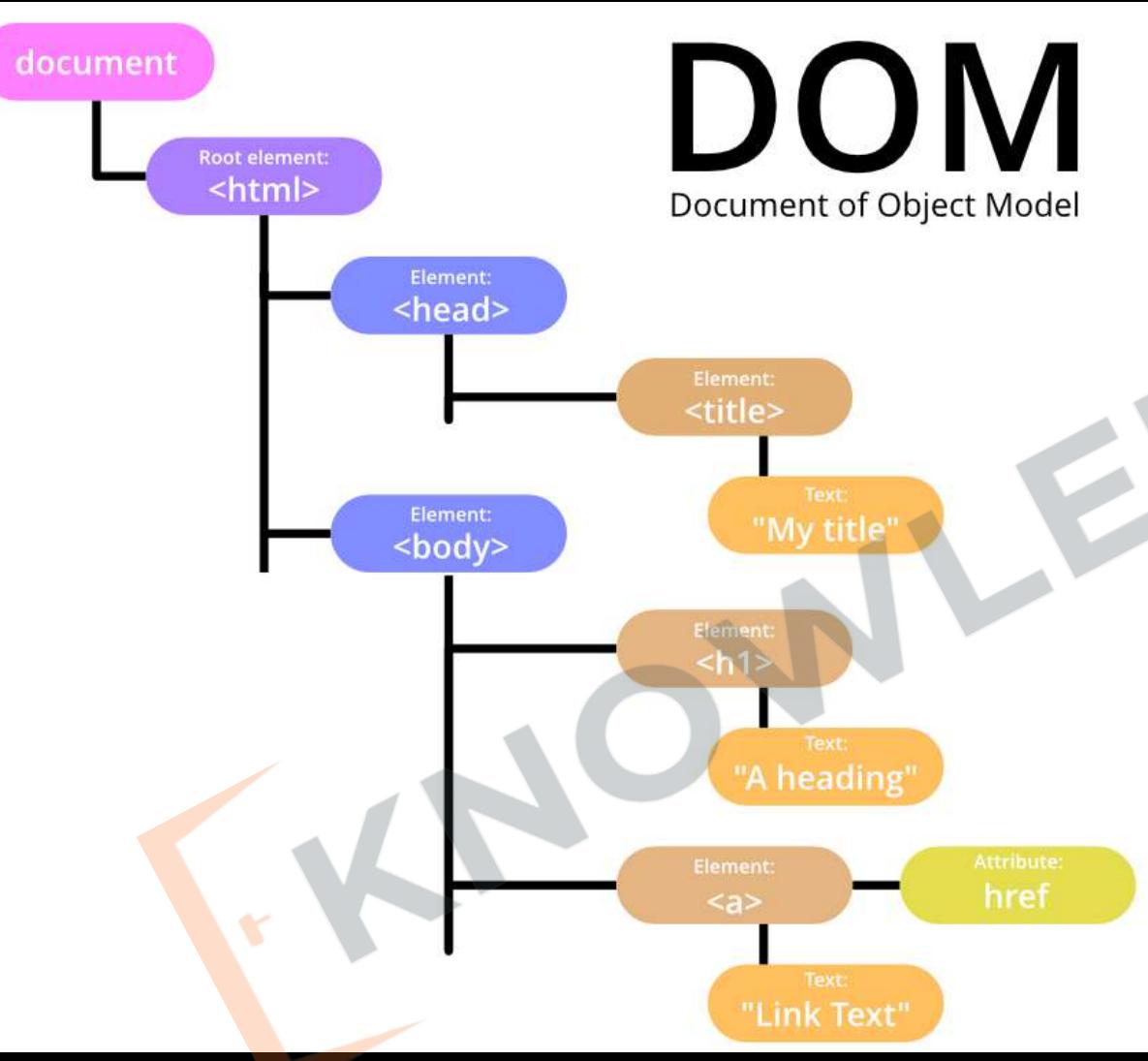
- prashantjain@Mac-mini test % node math.js
5 + 3 = 8
10 - 6 = 4
7 + 2 = 9
20 - 4 = 16
- prashantjain@Mac-mini test %



JavaScript in Webpage

```
<!DOCTYPE html>           Defines the HTML Version  
  
<html lang="en">          Parent of all HTML tags / Root element  
  
    <head>                  Parent of meta data tags  
        <title>My First Webpage</title>  Title of the web page  
    </head>  
  
    <body>                  Parent of content tags  
        <h1>Hello World!</h1>  Heading tag  
    </body>  
</html>
```

JavaScript in Webpage



- 1. Structure Understanding:** Helps in understanding the **hierarchical structure** of a webpage, crucial for applying targeted CSS styles.
- 2. Dynamic Styling:** Enables learning about dynamic styling, allowing for **real-time changes** and interactivity through CSS.

JavaScript in Webpage

(Script Tag)

1. **Embed Code:** Incorporates JavaScript into an HTML file, either **directly or via external files.**
2. **Placement:** Commonly placed in the **<head> or just before the closing </body> tag** to control when the script runs.
3. **External Files:** Use **src** attribute to link external JavaScript files, like **<script src="script.js"></script>**.
4. **Console Methods:** **log, warn, error, clear**

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Basic JavaScript Example</title>
    <script>
      // This function changes the content of the paragraph with id="demo"
      function changeContent() {
        document.getElementById("demo").innerHTML =
          "Content changed by JavaScript!";
      }
    </script>
  </head>
  <body>
    <h1>Welcome to My Web Page</h1>
    <p id="demo">JavaScript can change HTML content.</p>
    <button onclick="changeContent()">Click me to change content</button>
  </body>
</html>
```

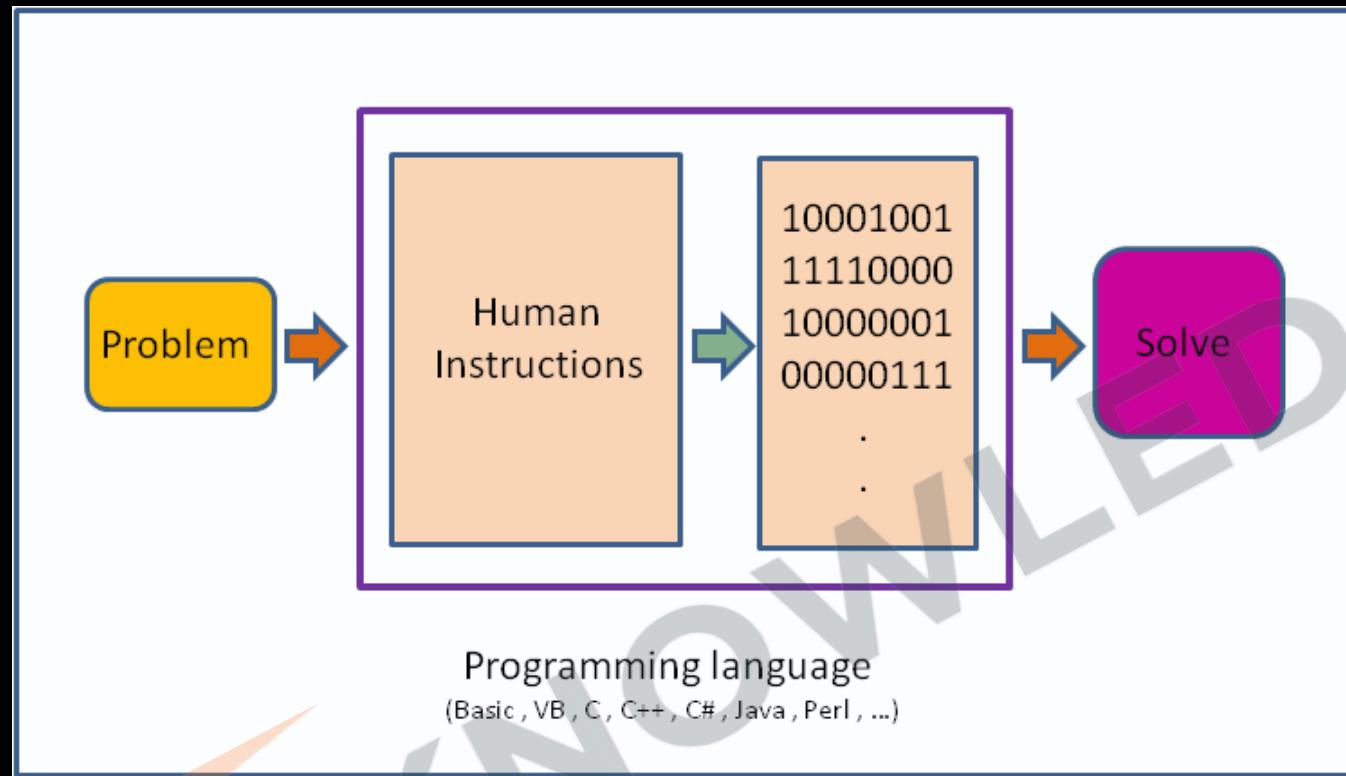
JavaScript in Webpage

(Comments)

```
/**  
 * * This is a important comment  
 * ! This is warning comment  
 * ? This is a question comment  
 * TODO: This is a todo comment  
 */
```

1. Used to add notes in source code in JavaScript or CSS.
2. Not displayed on the web page
3. Syntax: /* comment here */
4. Helpful for code organization
5. Can be multi-line or single-line

What is a Programming Language?



- Giving instructions to a computer
- **Instructions:** Tells computer what to do.
- These instructions are called **code**.

What is a Syntax?



- Structure of words in a sentence.
- Rules of the language.
- For programming exact syntax must be followed.

```
> alert 'hello world'  
✖ Uncaught SyntaxError: Unexpected string  
> |
```

FrontEnd / BackEnd / FullStack



Client Side / Front-End
Web Development



Server Side
Back-End

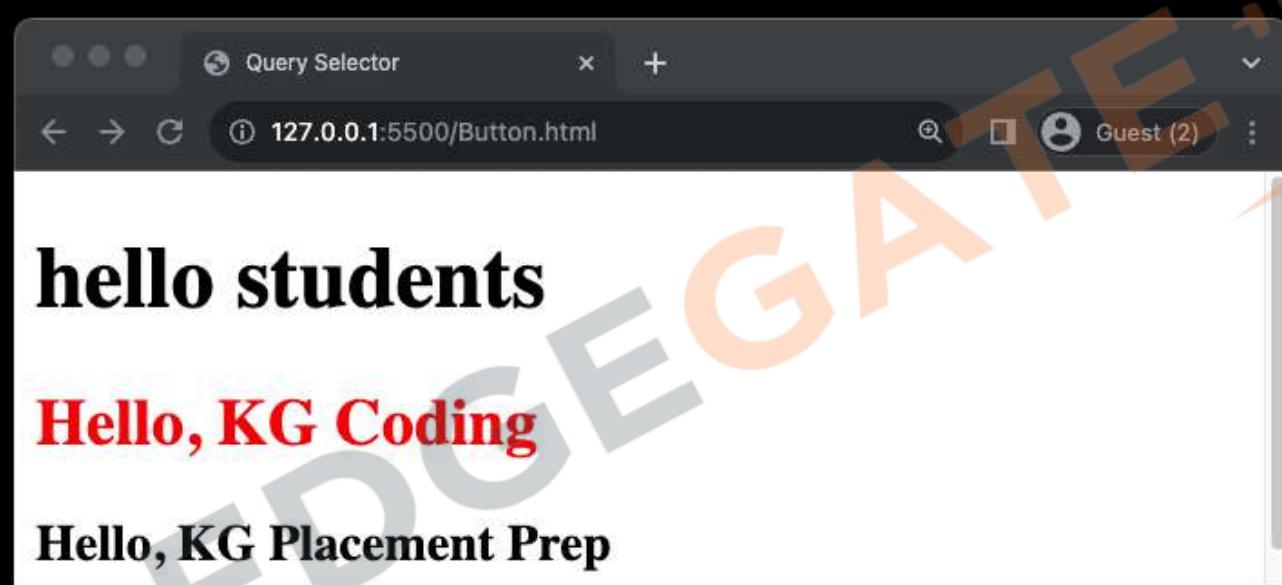
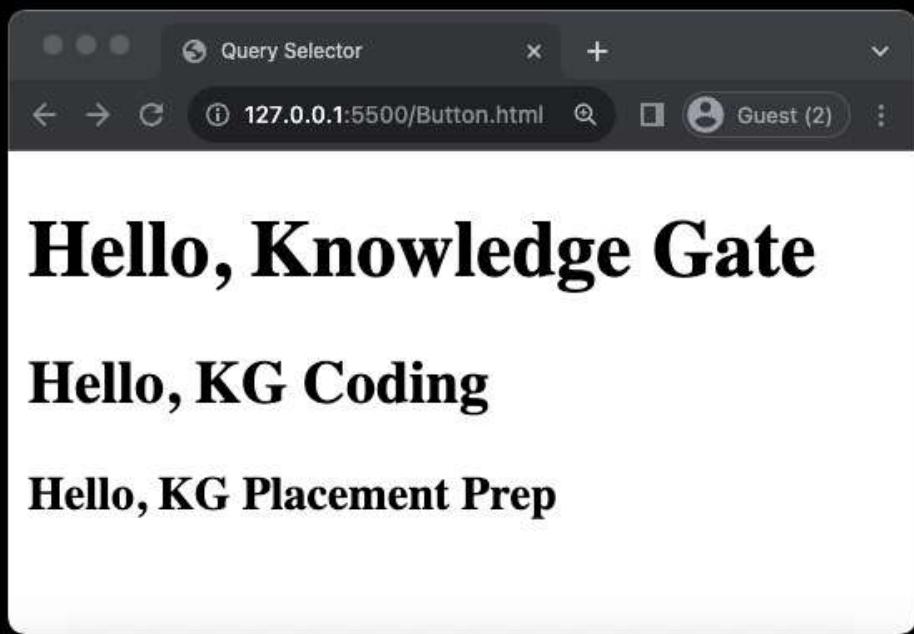


Full Stack

Query Selector

1. **getElementById**: Finds one element by its ID.
2. **getElementsByClassName**: Finds elements by their class, returns a list.
3. **querySelector**: Finds the first element that matches a CSS selector.
4. **Purpose**: To interact with or modify webpage elements.

Query Selector



A screenshot of a code editor window titled "Button.html – testing". The file content is as follows:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Query Selector</title>
  </head>
  <body>
    <h1>Hello, Knowledge Gate</h1>
    <h2 class="coding">Hello, KG Coding</h2>
    <h3 id="placement">Hello, KG Placement Prep</h3>
  </body>
</html>
```

A screenshot of a browser developer tools window with the "Console" tab selected. The console output shows the following JavaScript code:

```
> const hElement = document.querySelector('h1');
const hElementCode = document.querySelector('.coding');
const hElementPlacement =
document.querySelector('#placement');

hElement.textContent = 'hello students';
hElementCode.style.color = 'red';

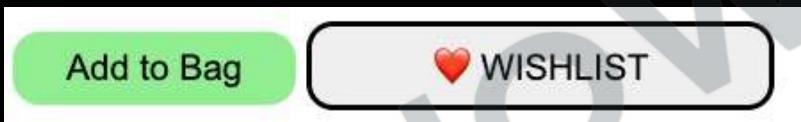
< 'red'

>
```

Practice Exercise

JavaScript With HTML & CSS

1. Create a button with text click
2. Create 2 buttons with class and id
3. Create a paragraph
4. Add colors to two buttons
5. Add proper html structure
6. Change page title
7. Try to copy the given design on the bottom



8. Add script element to page to show welcome alert
9. Add onclick alert on Add to Bag and ❤️ Wishlist buttons

KNOWLEDGEAGATE

Arithmetic Operators

Operators	Meaning	Example	Result
+	Addition	$4+2$	6
-	Subtraction	$4-2$	2
*	Multiplication	$4*2$	8
/	Division	$4/2$	2
%	Modulus operator to get remainder in integer division	$5\%2$	1

2/2 ITEMS SELECTED

REMOVE

MOVE TO WISHLIST



Allen Solly

Men Textured Slim Fit Mid-Rise Trousers
Sold by: Westbury Holdings Pvt Ltd_Madura

Size: 32 - Qty: 1 -

₹ 1,474 ₹2,499 41% OFF

Coupon Discount: ₹29

⌚ 14 days return available

✓ Delivery by 8 Oct 2023



GUESS

Brand Logo Woven Design Structured Handheld Bag
Sold by: Supercom Net

Size: Onesize - Qty: 1 - 9 left

₹ 14,039 ₹15,599 10% OFF

Coupon Discount: ₹272

⌚ 7 days return available

⌚ Delivery by 3 Oct 2023

X

X

COUPONS



1 Coupon applied

You saved additional ₹301

EDIT

GIFTING & PERSONALISATION



Yay! Gift Wrapping applied

Your order will be gift wrapped with your personalised message

REMOVE

EDIT MESSAGE

PRICE DETAILS (2 Items)

Total MRP ₹18,098

Discount on MRP -₹2,585

Coupon Discount ⓘ -₹301

Gift Wrap Charges ₹25

Convenience Fee [Know More](#) ₹20

Total Amount ₹15,257

PLACE ORDER

Order of Operations

B	O	D	M	A	S
Bracket	Order	Divide	Multiply	Add	Subtract
()	\sqrt{x}	x^2	\div or \times	$+$ or $-$	
Parentheses	Exponents	Multiply	Divide	Add	Subtract
P	E	M	D	A	S

$$9 \div 3 \times 2 \div 6$$

$$8 - 5 + 7 - 1$$



MYNTRA Cart

✓ 2/2 ITEMS SELECTED

REMOVE

MOVE TO WISHLIST



Allen Solly

Men Textured Slim Fit Mid-Rise Trousers
Sold by: Westbury Holdings Pvt Ltd_Madura

Size: 32 - Qty: 1 -

₹ 1,474 ₹2,499 41% OFF

Coupon Discount: ₹29

⌚ 14 days return available

✓ Delivery by 8 Oct 2023



GUESS

Brand Logo Woven Design Structured Handheld Bag
Sold by: Supercom Net

Size: Onesize - Qty: 1 - 9 left

₹ 14,039 ₹15,599 10% OFF

Coupon Discount: ₹272

⌚ 7 days return available

⌚ Delivery by 3 Oct 2023

COUPONS



1 Coupon applied

You saved additional ₹301

EDIT

GIFTING & PERSONALISATION



Yay! Gift Wrapping applied

Your order will be gift wrapped with your personalised message

REMOVE

EDIT MESSAGE

PRICE DETAILS (2 Items)

Total MRP ₹18,098

Discount on MRP -₹2,585

Coupon Discount ⓘ -₹301

Gift Wrap Charges ₹25

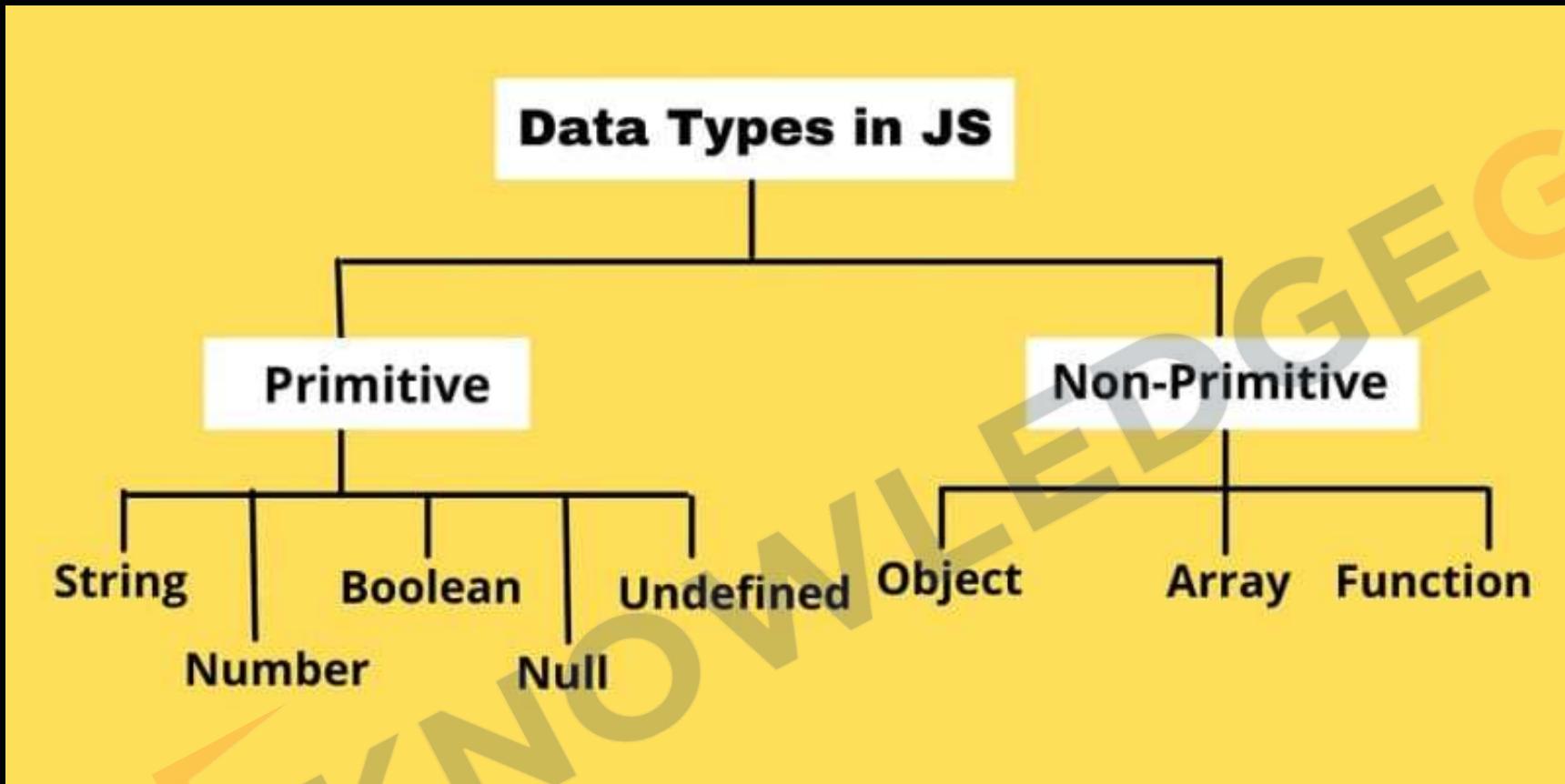
Convenience Fee [Know More](#) ₹20

Total Amount ₹15,257

PLACE ORDER

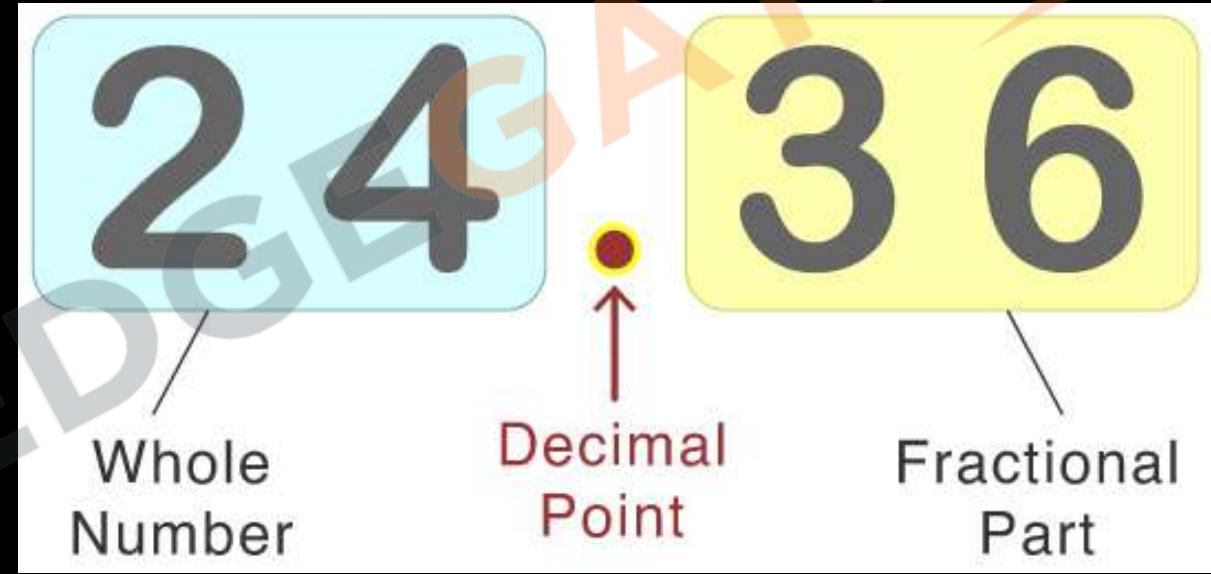
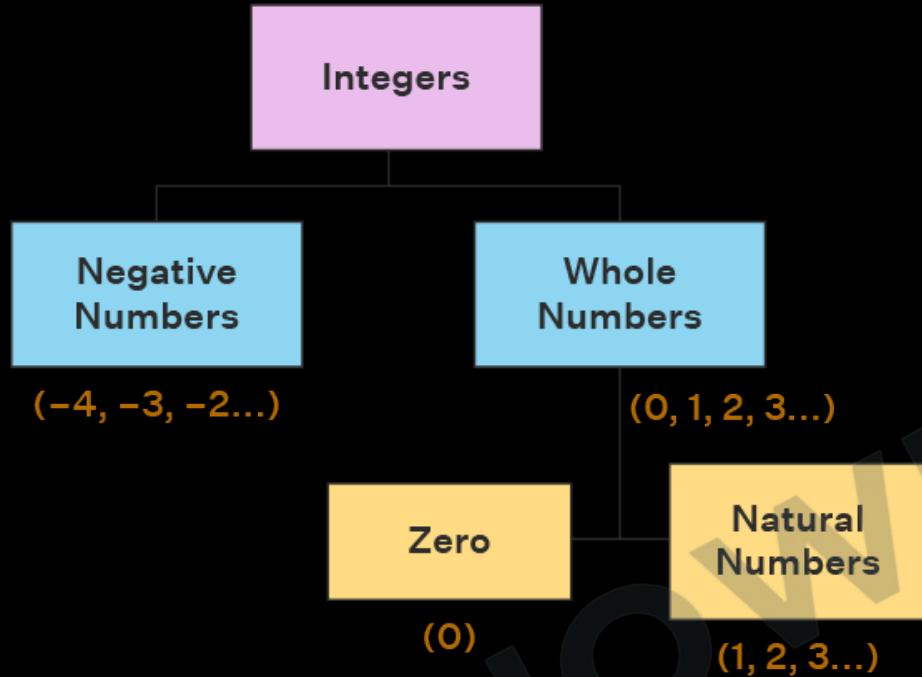
Primitive Types

(What are Data Types)

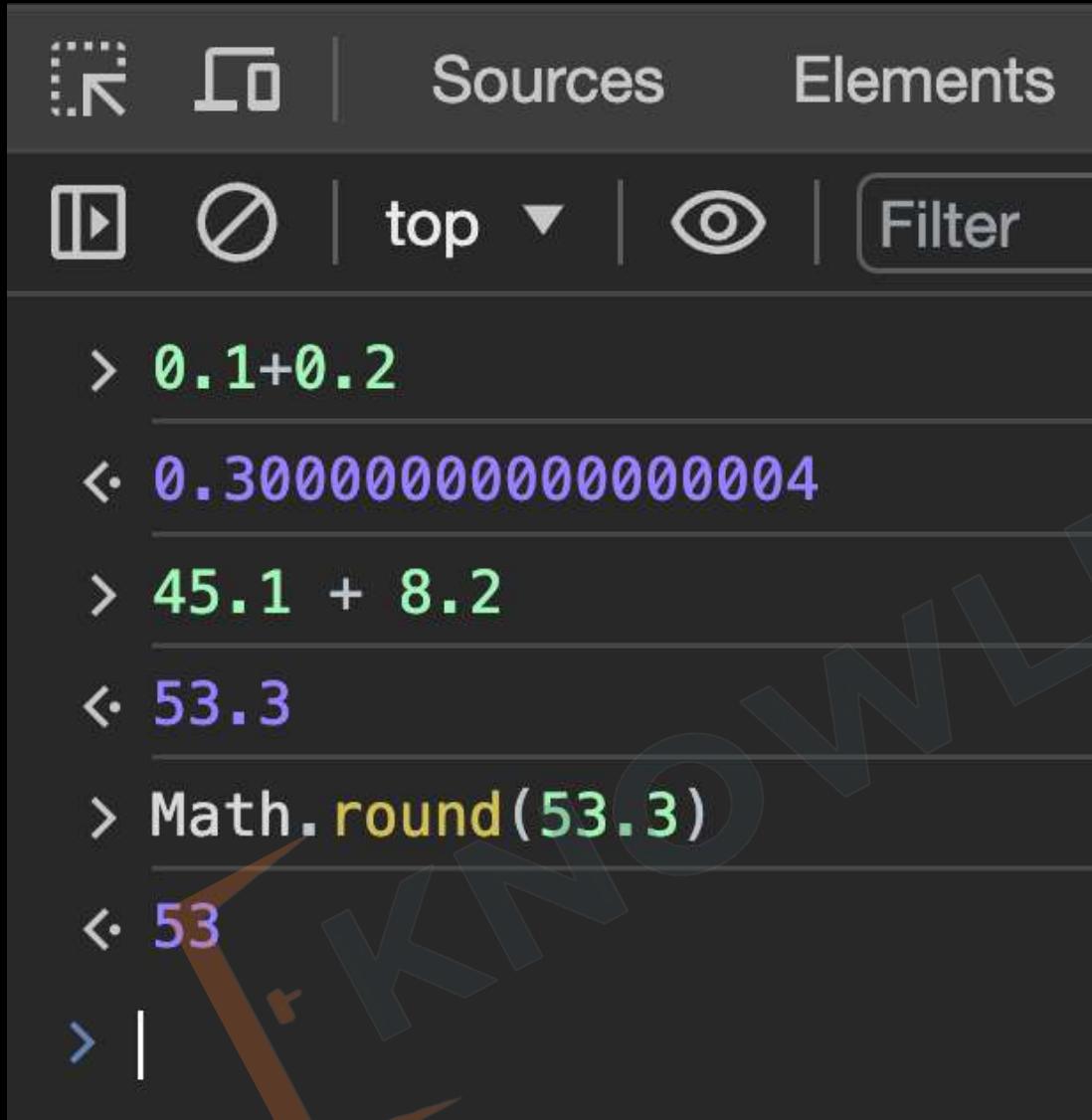


Primitive types in **JavaScript** are the most basic data types that are not objects and have no methods. They are **immutable**, meaning their **values** cannot be changed.

Types of Numbers (Integers & Floats)

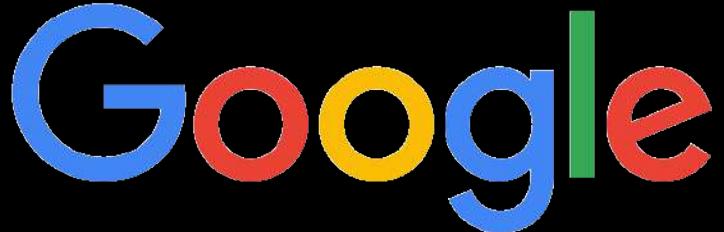


Types of Numbers (Float Problems)



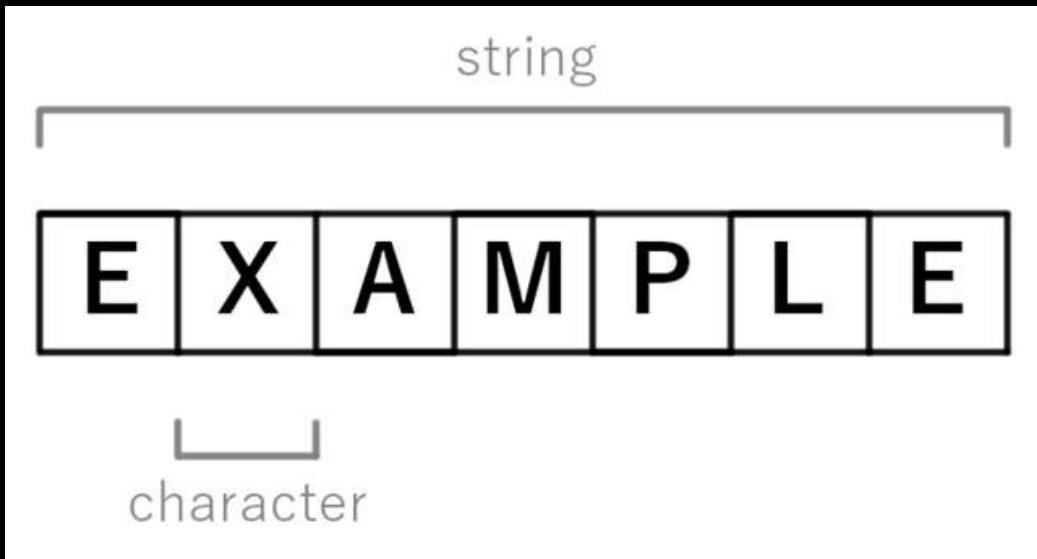
1. JavaScript has many problems with float operations.
2. We can use `Math.round()` to convert floats to integers.
3. Always do money calculation in Paisa instead of Rupees

Don't learn syntax



1. **Google:** Quick answers to coding problems.
2. **MDN:** In-depth guides and documentation.
<https://developer.mozilla.org/>
3. **ChatGPT:** Real-time assistance for coding queries.
4. **Focus:** Understand concepts, not just syntax.

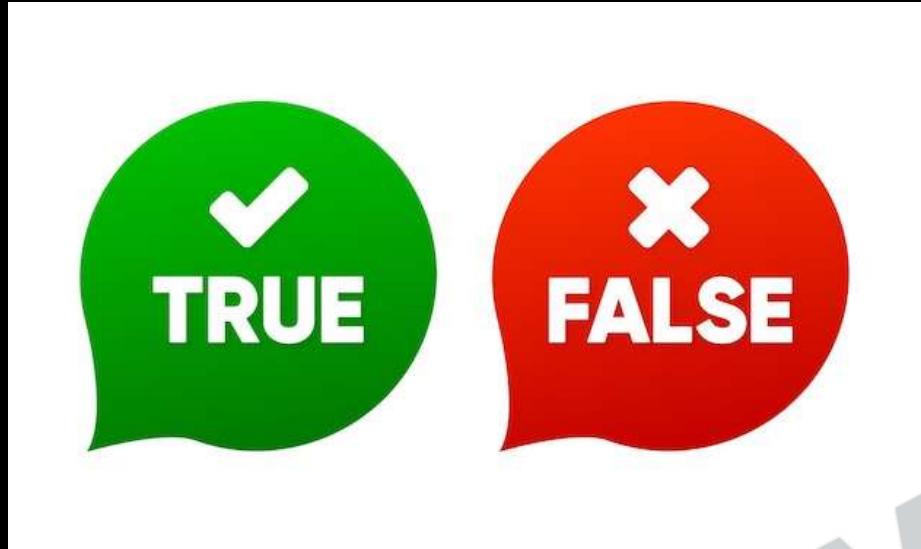
Strings



1. Strings hold **textual data**, anything from a single character to paragraph.
2. Strings can be defined using **single quotes ''**, **double quotes " "**, or **backticks ` `**. Backticks allow for template literals, which can include variables.
3. You can combine (**concatenate**) strings using the **+** operator. For example, **"Hello" + " World"** will produce **"Hello World"**.

Primitive Types

(Boolean)



1. Data Type: Booleans are a basic data type in JavaScript.
2. Two Values: Can only be `true` or `false`.
3. '`true`' is a String not a Boolean

Primitive Types

(Null vs Undefined)

⇒ NULL vs UNDEFINED

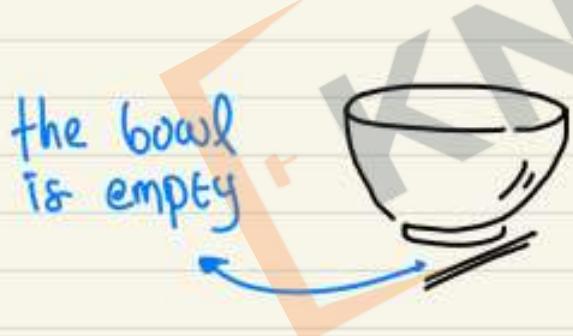
THOUGH BOTH ARE NULLISH & FALSEY VALUE

null !== undefined

• no value, on purpose

the type is object

equal to 0 (zero)



• declared, but not yet defined

the type is undefined

equal to NaN (Not A Number)

even, the bowl
is not exist

AGGREGATE

Type Of Operator

JS

typeof function

typeof 'this is a string' → 'string'

typeof 12345 → 'number'

typeof (11 === 11) → 'boolean'

1. **Check Type:** Tells you the data type of a variable.
2. **Syntax:** Use it like `typeof` variable.
3. **Common Types:** Returns "number," "string," "boolean," etc.

Practice Exercise

Numbers & Strings

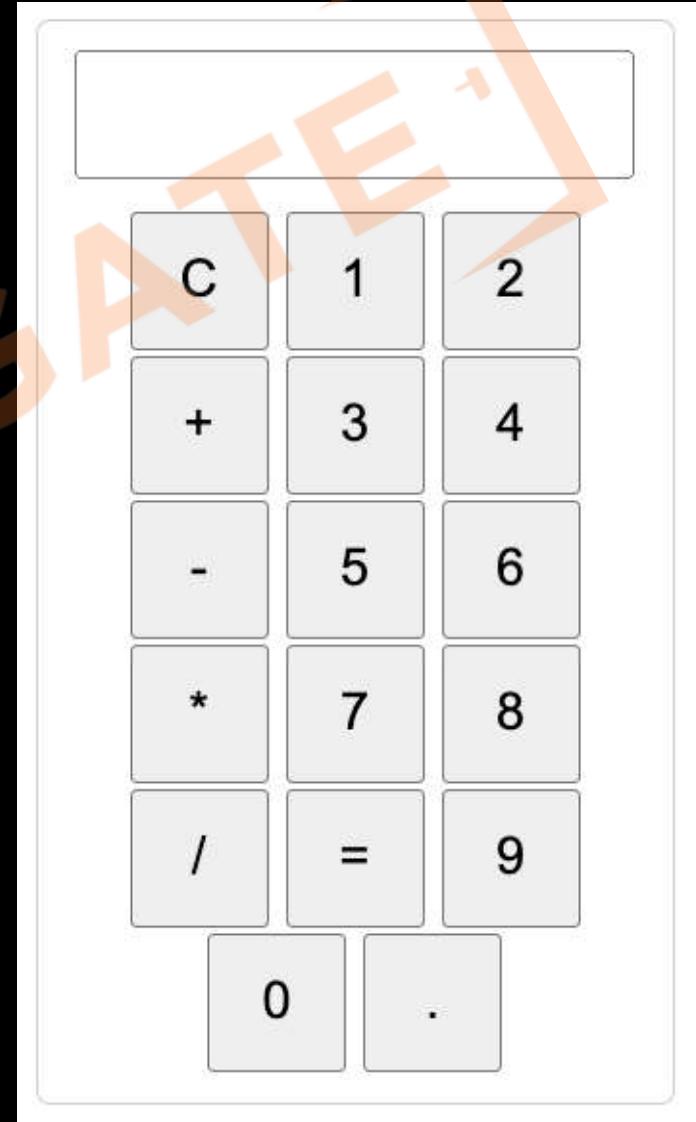
1. At a restaurant you ate: 1 Dal ₹ 100, 2 Roti ₹10 each, 1 Ice Cream ₹30, calculate and display the final bill amount.
2. Calculate 18% GST on iPhone15 ₹79,990 and 2 Air pods Pro ₹24990 each.
3. Create strings using all 3 methods.
4. Concatenate String with Strings, and String with numbers.
5. Create Order Summary String for our Myntra Cart.
6. Display order summary in a popup





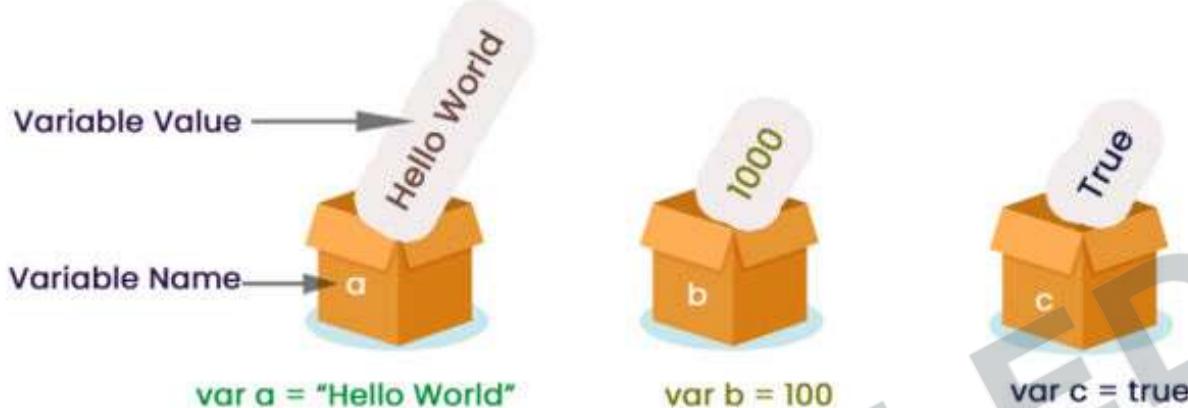
Variables

- What are Variables
- Syntax Rules
- Updating Values
- Myntra Bag Exercise
- Naming Conventions
- Ways to create Variables



What are Variables?

Variable is used to Store Data



Variables are like containers
used for storing data values.

Syntax Rules

```
1 // Defining a number variable  
2 let noOfStudents = 5;  
3 // Defining a String variable  
4 let welcomeMessage = "Hello Beta"
```

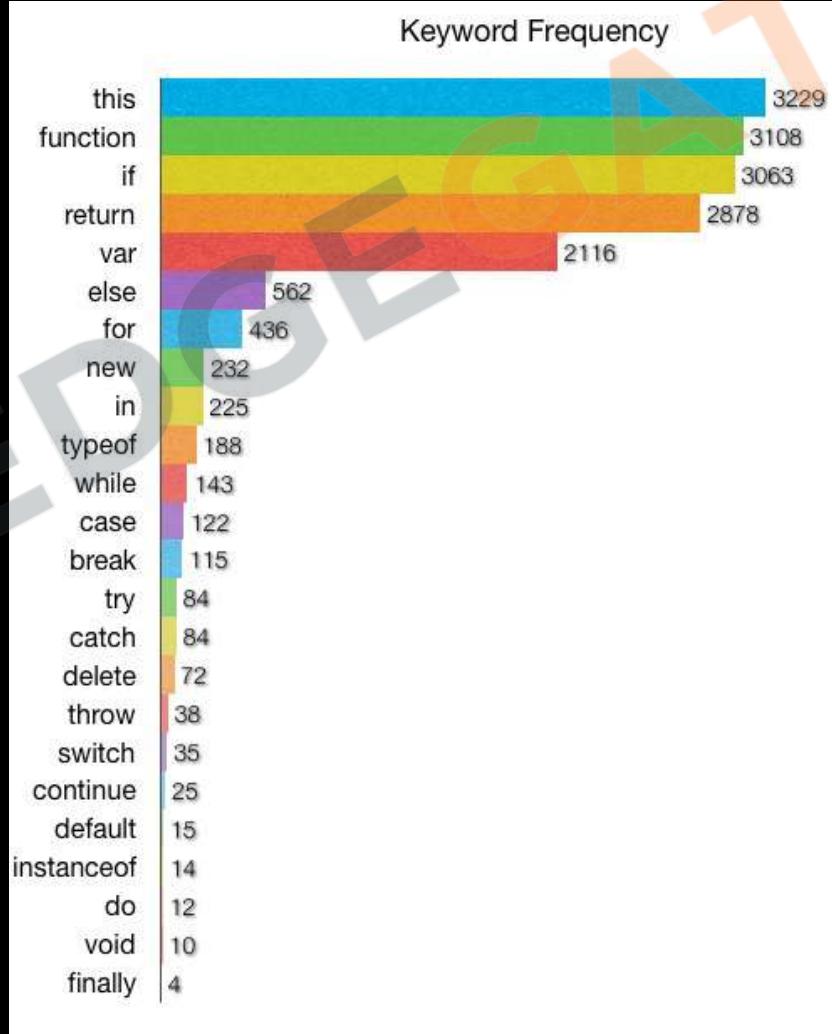
1. Can't use **keywords** or reserved words
2. Can't start with a **number**
3. **No** special characters other than **\$** and **_**
4. **=** is for **assignment**
5. **;** means end of **instruction**

JS Keywords

JS	• enum*	• null	• try
• arguments	• eval	• package	• typeof
• export*	• extends*	• private	• var
• await*	• false	• protected	• void
• break	• finally	• public	• while
• case	• for	• return	• with
• catch	• function		
• class*	• if		
• const	• implements		
• continue	• in	• static	JavaScript
• debugger	• instanceof	• super*	
• default	• import*	• switch	
• delete	• interface	• this	
• do	• let*	• throw	
• else	• new	• true	

Reserved Words

Words marked with* are new in ECMAScript 5 and 6



Naming Conventions

camelCase

- Start with a lowercase letter. Capitalize the first letter of each subsequent word.
- Example: `myVariableName`

snake_case

- Start with an lowercase letter. Separate words with `underscore`
- Example: `my_variable_name`

Kebab-case

- All lowercase letters. Separate words with `hyphens`. Used for HTML and CSS.
- Example: `my-variable-name`

Keep a Good and Short Name

- Choose names that are descriptive but not too long. It should make it easy to understand the variable's purpose.
- Example: `age`, `firstName`, `isMarried`



Practice Exercise

Variable Naming

- username
- totalAmount
- 99balls
- User3
- first name
- new
- function
- _score
- maxHeight
- user-name
- \$budget
- class
- 2024Year
- last_name
- Count99
- user@name
- boolean
- var
- indexValue
- tempVar

Practice Exercise (Solution)

Variable Naming

- `username`
- `totalAmount`
- `99balls` - Incorrect, cannot start with a number.
- `User3`
- `first name` - Incorrect, contains a space.
- `new` - Incorrect, uses a reserved keyword.
- `function` - Incorrect, uses a reserved keyword.
- `_score`
- `maxHeight`
- `user-name` - Incorrect, hyphens are not allowed.
- `$budget`
- `class` - Incorrect, uses a reserved keyword.
- `2024Year` - Incorrect, starts with a number.
- `last_name`
- `Count99`
- `user@name` - Incorrect, special characters like @ are not allowed.
- `boolean` - Incorrect, uses a reserved keyword.
- `var` - Incorrect, uses a reserved keyword.
- `indexValue`
- `tempVar`

Updating Values

```
let noOfStudents = 5;  
noOfStudents = noOfStudents + 1;  
  
let money = 1;  
money += 5; // money = 6  
money -= 2; // money = 4  
money *= 3; // money = 12  
money /= 4; // money = 3  
money++; // money = 4
```

1. Do not need to use **let** again.
2. Syntax: **variable = variable + 1**
3. **Assignment Operator** is used **=**
4. **Short Hand Assignment Operators:**
`+ =, - =, * =, / =, ++`

Myntre Bag Exercise

Add to Bag

MOVE TO WISHLIST

Add 1+1 sale item

Your Bag has 2 items

1. Implement the above three buttons
2. Add to Bag should add an item to the bag
3. Move to wishlist should remove an item from the bag
4. Add 1+1 sale item should add 2 items to the bag

Ways to Create Variables

var

var apple = 
 a thing in a box
named "apple"

apple = 
 you can swap
item later

let

let apple = 
 a thing in a box
named "apple" w/
protection shield

apple = 
 OK!
apple = 
you can swap item
only if you ask
inside of the shield

const

const apple = 
 a thing in
LOCKED cage
named "apple"

apple = 
 you can't
swap item
later.
apple.multiply(3)
OK!
... but you can ask
the item to change itself
(if the item has method
to do that)



Practice Exercise

Variables

1. Save your name in a **variable** inside script tag
2. Display name from the **variable** on the page
3. Calculate the cost of **Myntra Bag** and keep it in a **variable**
4. Show it to **console**
5. Keep **GST** percentage as **constant**
6. Use **eval** method from math to convert string calculation into result



Project: Calculator



KNOWLEDGE
AGGREGATE



Decision Control

- Comparison Operators
- if-else, ladder, nested-if
- Truthy and Falsy Values
- Logical Operator
- If alternates
- Scope
- Switch

Create

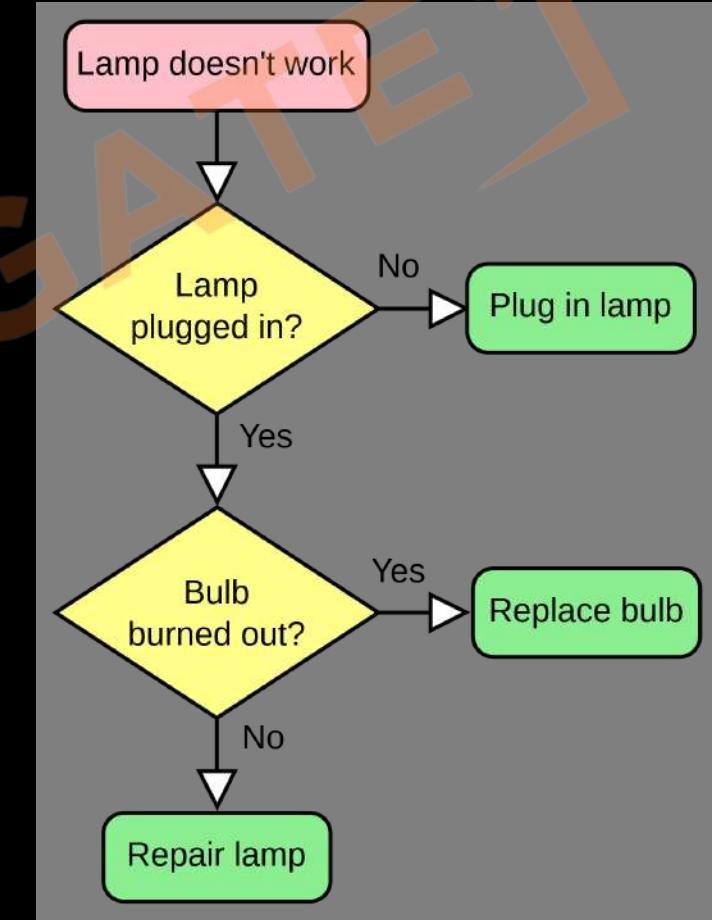
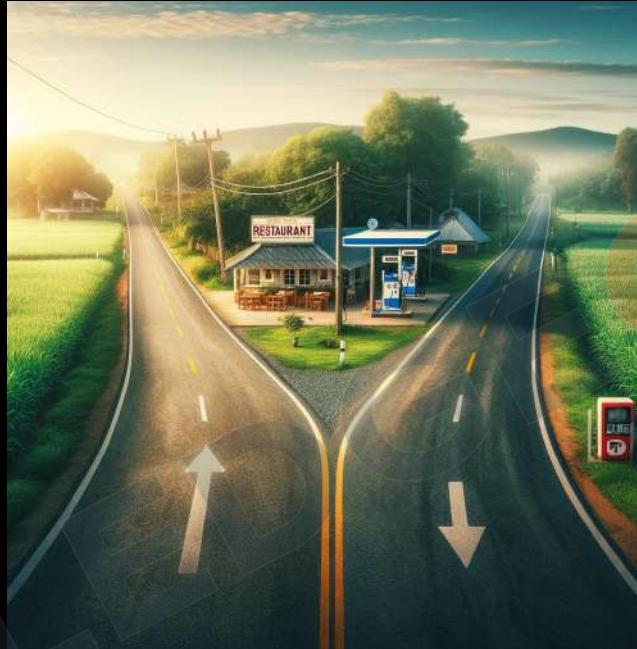


Project

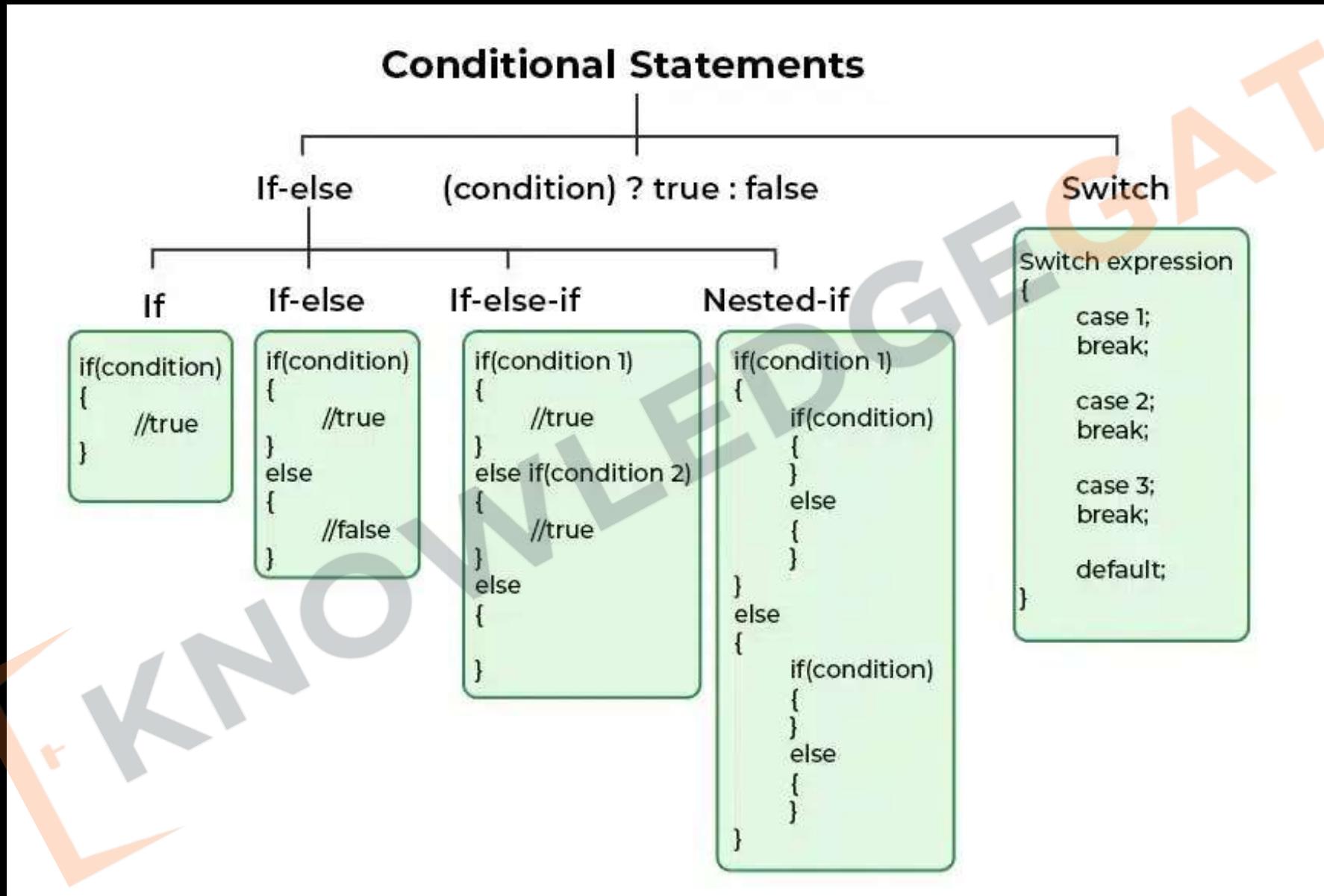
KNOWLEDGE GATE

What is Decision Control ?

1. Conditional Execution: They allow code to run **based on specific conditions**, making programs dynamic.
2. Handles Complexity: Enables handling **complex decisions** through nested statements.
3. Enhances Flexibility: Increases the adaptability of programs to **different scenarios**.



What is Decision Control ?



Comparison Operators

<, >, <= , >= , == , !=

- Equality

- == Checks value equality.
- === Checks value and type equality.

- Inequality

- != Checks value inequality.
- !== Checks value and type inequality.

- Relational

- > Greater than.
- < Less than.
- >= Greater than or equal to.
- <= Less than or equal to.

Order of comparison operators is less than arithmetic operators

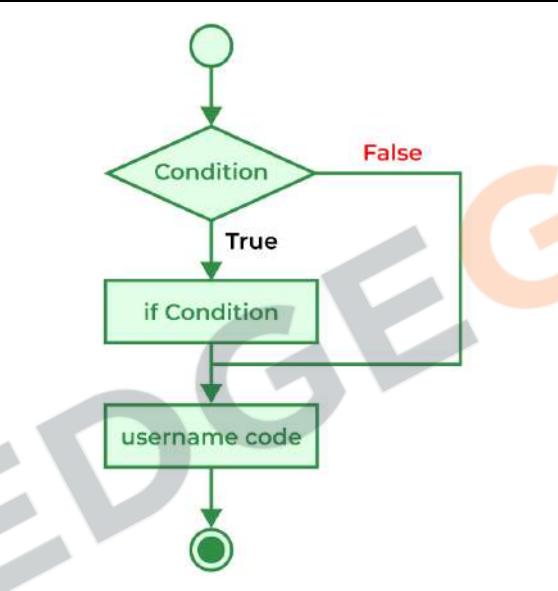
if-else

Condition is true

```
let number = 2;  
if (number > 0) {  
    // code  
}  
else {  
    // code  
}  
// code after if
```

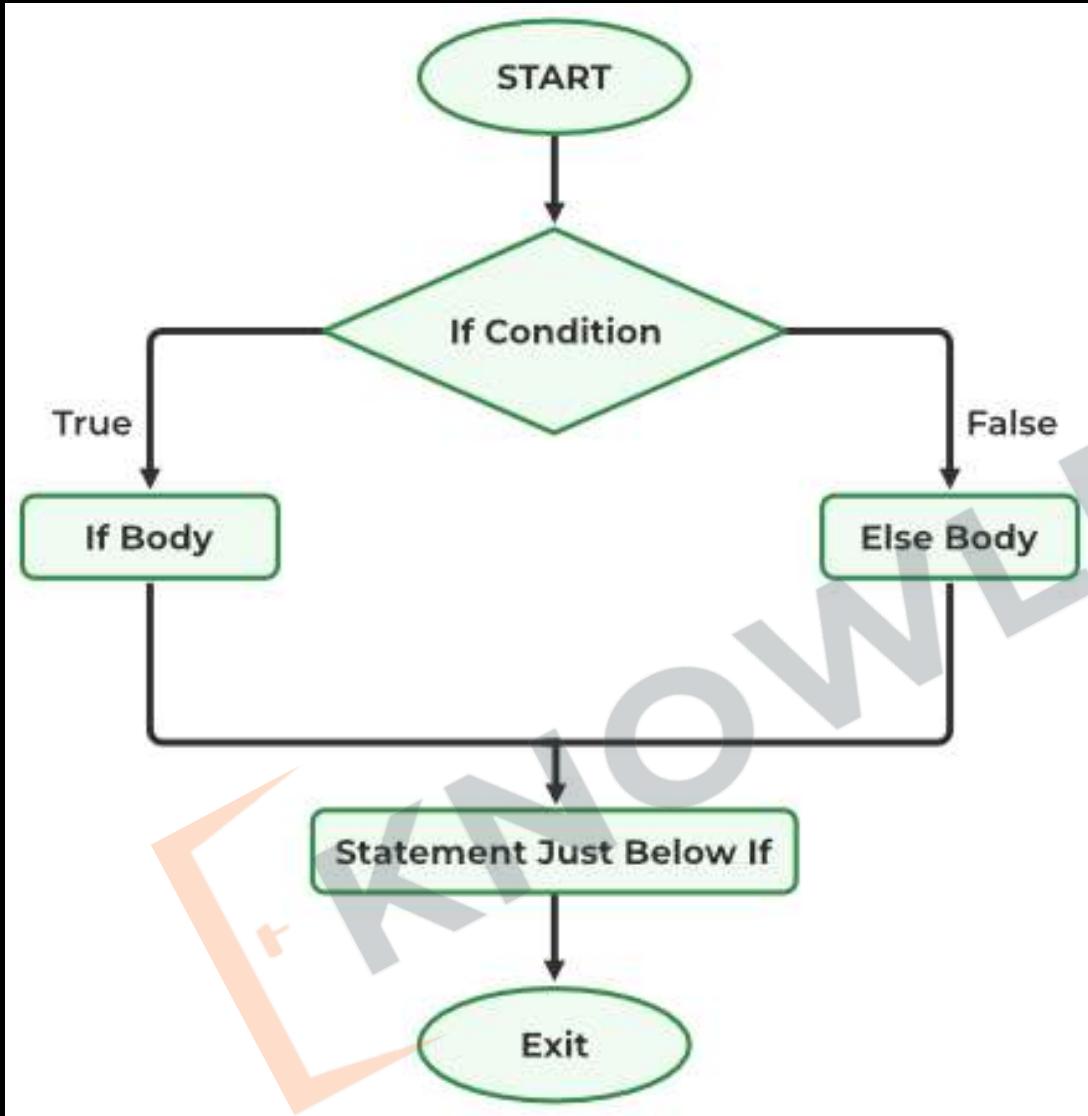
Condition is false

```
let number = -2;  
if (number > 0) {  
    // code  
}  
else {  
    // code  
}  
// code after if
```



1. Syntax: Uses `if () {}` to check a condition.
2. What is `if`: Executes block if condition is `true`, skips if `false`.
3. What is `else`: Executes a block when the if condition is `false`.
4. Curly Braces can be omitted for single statements, but not recommended.
5. Use Variables: Can store conditions in variables for use in if statements.

if-else



if thirsty {



} else {



What is **else**: Executes a block
when the **if** condition is **false**.



Truthy vs Falsy

```
if (x) {  
    // x is "truthy"  
} else {  
    // x is "falsy"  
}
```

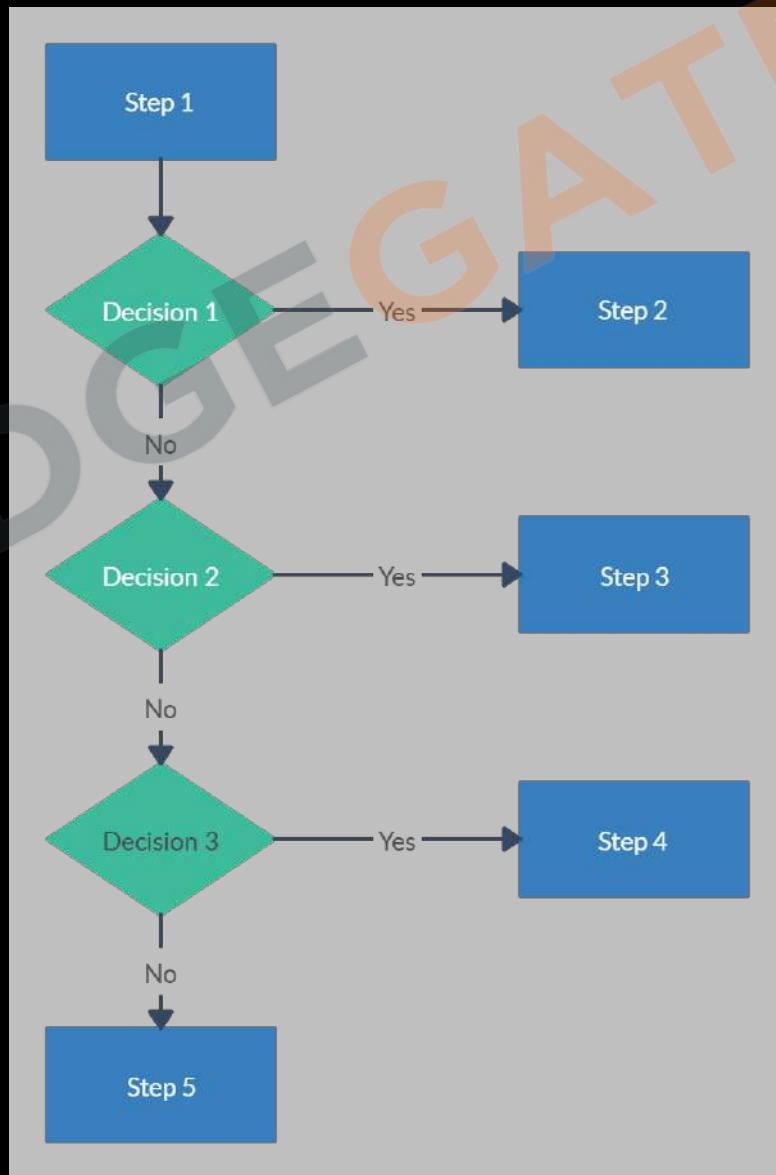
```
if (1) {  
    console.log("the value is truthy");  
}  
else {  
    console.log("the value is falsy");  
}
```



1. Falsy Values: 0, null, undefined, false, NaN, "" (empty string)
2. Truthy Values: All values **not** listed as falsy.
3. Used in **conditional** statements like **if**.
4. **Non-boolean** values are **auto-converted** in logical operations.
5. Be **explicit** in **comparisons** to avoid unexpected behaviour.

if-else-if Ladder

1. **Sequential Checking:** The if-else ladder checks **multiple conditions** one after the other, from top to bottom.
2. **First True Condition:** Executes the block of code associated with the **first true condition** it encounters.
3. **Exits After Execution:** Once a true condition's code block is executed, it **exits the ladder** and skips the remaining conditions.
4. **Fallback with Else:** If **none of the conditions** are true, the final else block (if present) executes as a default case.

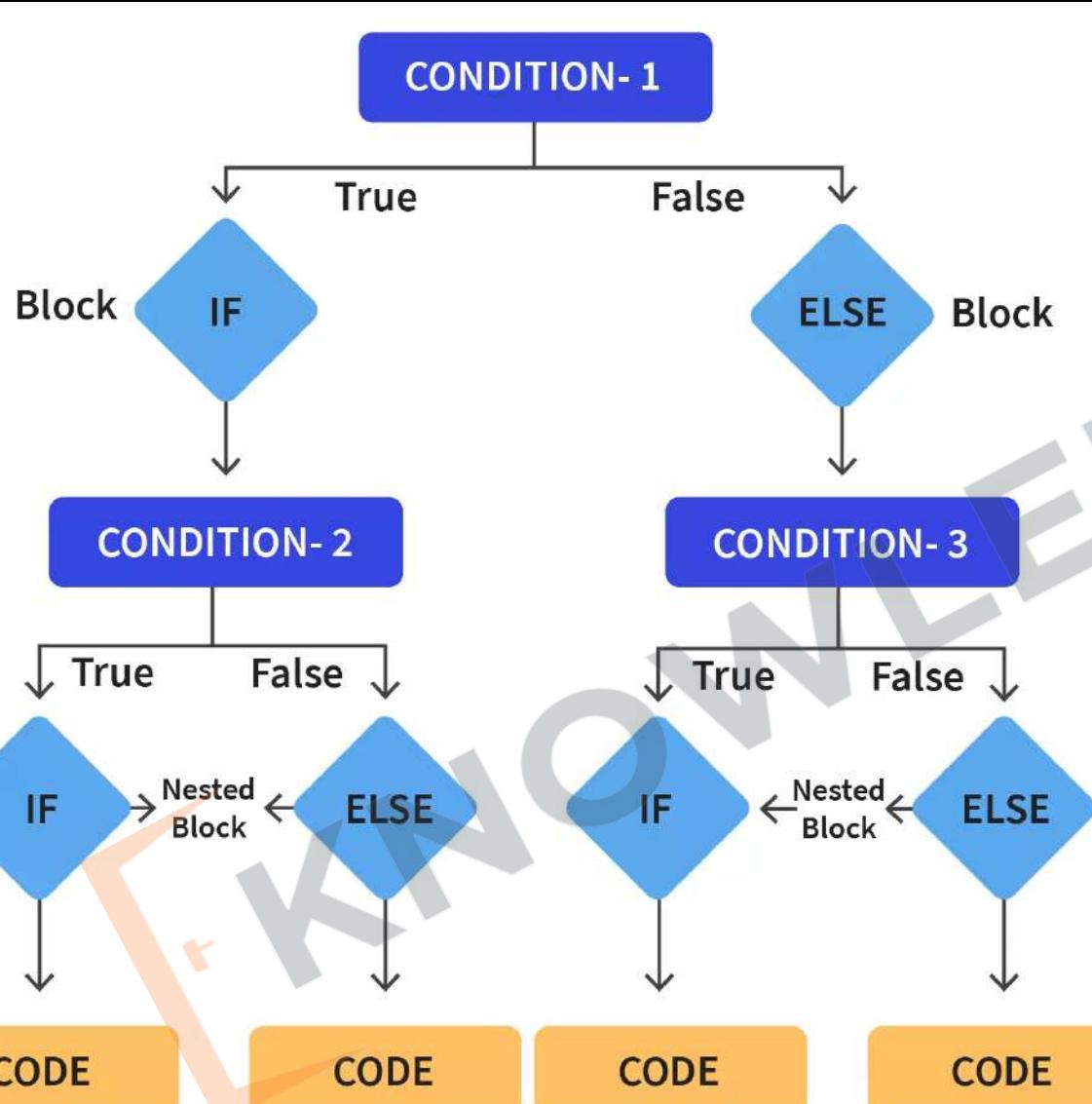


if-else-if Ladder

```
// Use of if-else ladder
let score = 85;
if (score >= 90) {
    console.log("Grade: A");
} else if (score >= 80) {
    console.log("Grade: B");
} else if (score >= 70) {
    console.log("Grade: C");
} else if (score >= 60) {
    console.log("Grade: D");
} else {
    console.log("Grade: F");
}
```

If-else Ladder: Multiple if and else if blocks; only one executes.

Nested if

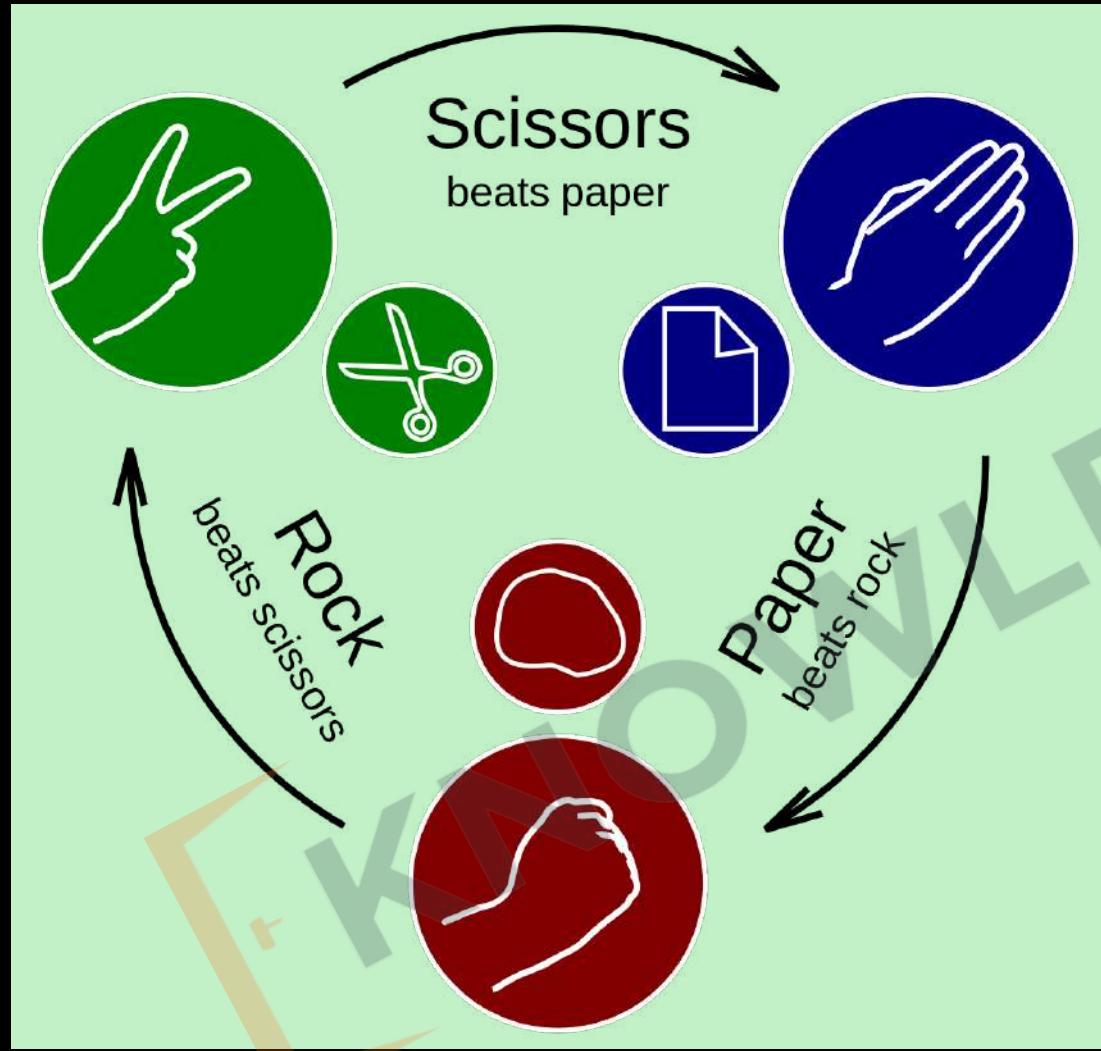


1. Condition Hierarchy: Enables hierarchical condition checks.
2. Complexity: Allows for detailed decision-making paths.
3. Syntax: An **if** inside another **if** or **else**.
4. Readability: Deep nesting can reduce code clarity.

Nested if

```
// Use of nested if-else
let number = 10;
if (number > 0) {
  if (number % 2 === 0) {
    console.log("The number is positive and even.");
  } else {
    console.log("The number is positive and odd.");
  }
} else if (number < 0) {
  console.log("The number is negative.");
} else {
  console.log("The number is zero.");
}
```

Project Rock-Paper-Scissor Game



Rock Paper Scissors Game

Click on one of the following to play the game:



- Create the UI
- Add click listeners to buttons to print the current choice

Logical Operators



AND

Or GEGAP not

1. Types: **&&** (AND), **||** (OR), **!** (NOT)
2. AND (**&&**): All conditions **must be true** for the result to be true.
3. OR (**||**): Only **one condition** must be true for the result to be true.
4. NOT (**!**): **Inverts** the Boolean value of a condition.
5. Lower Priority than **Math** and **Comparison** operators

Logical Operators

```
console.log("And Operator");
console.log(true && true);
console.log(true && false);
console.log(false && true);
console.log(false && false);
```

```
console.log("Or Operator");
console.log(true || true);
console.log(true || false);
console.log(false || true);
console.log(false || false);
```

```
console.log("Not Operator");
console.log(!true);
console.log(!false);
console.log (!!true);
```

```
let num = 5;
if (num > 0 && num % 2 === 0) {
  console.log("Positive and Even number");
} else if (num > 0) {
  console.log("Positive and Odd number");
} else if (num < 0) {
  console.log("Negative number");
} else {
  console.log("Number is Zero");
}
```

Including Script

1. Inline JavaScript:

- Place JavaScript code directly within the **HTML element** using the onclick, onload, or other event attributes.

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

2. Internal JavaScript:

- Embed JavaScript code within a **<script>** tag in the HTML document's **<head>** or **<body>**

```
<script>
  function showAlert() {
    alert("Hello, World!");
  }
</script>
<button onclick="showAlert()">Click me</button>
```

3. External JavaScript:

- Link to an external JavaScript file using the **<script>** tag with the src attribute.

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

Template Literals

1. **Syntax:** Enclosed by backticks (`) instead of single or double quotes.
2. **Multi-line Strings:** Allows creating multi-line strings without the need for escape characters.
3. **String Interpolation:** Embed expressions within a string using `${expression}` syntax.
4. **Expression Evaluation:** Supports embedding any valid JavaScript expression, including arithmetic operations, function calls, and ternary operations.

```
// Syntax  
let greeting = `Hello, World!`;  
  
// Multi-line Strings  
let multiLine = `This is a  
multi-line string.`;  
  
// String Interpolation  
let firstName = 'John';  
let hello = `Hello, ${firstName}`;  
  
// Expression Evaluation  
let a = 5;  
let b = 10;  
let result = `The sum of a and b is ${a + b}.`;
```

Revisiting Undefined & Null

1. undefined:

- A variable that has been declared but **has not yet** been assigned a value.
- Represents the **absence** of a value in a variable.
- Also returned when trying to access an object property that does not exist.

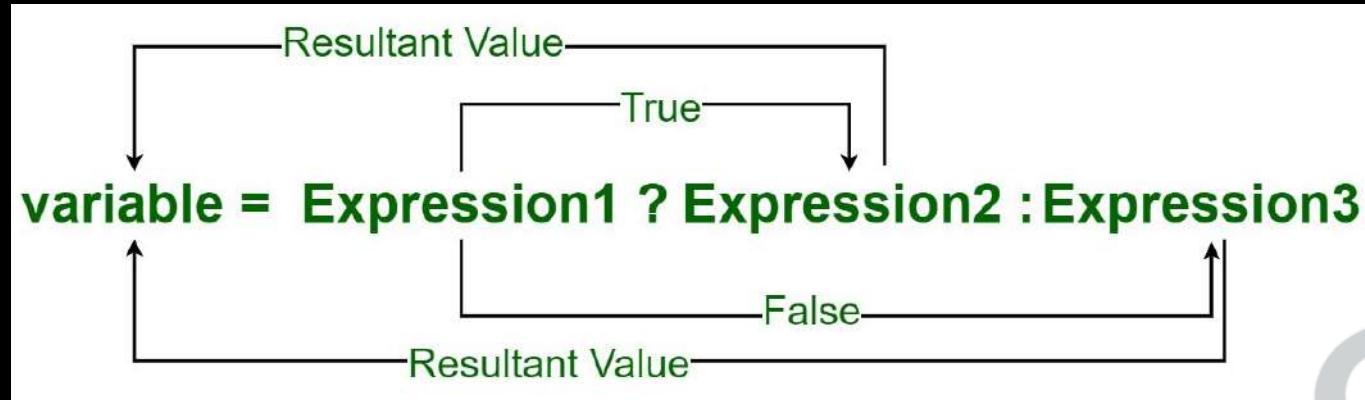
```
let a;  
console.log(a); // Output: undefined  
let obj = {};  
console.log(obj.property); // Output: undefined
```

2. null:

- Represents the **intentional absence** of any object value.
- Often used to explicitly **indicate that a variable should be empty**.
- Both undefined and null are loosely equal (==) but **not strictly equal** (==).

```
let b = null;  
console.log(b); // Output: null  
console.log(undefined == null); // Output: true  
console.log(undefined === null); // Output: false
```

If alternates



1. Ternary Operator: `condition ? trueValue : falseValue`

Quick one-line if-else.

2. Guard Operator: `value || defaultValue`

Use when a `fallback` value is needed.

3. Default Operator: `value ?? fallbackValue`

Use when you want to consider only null and undefined as falsy.

4. Simplifies conditional logic.

5. Use wisely to maintain readability.

If alternates

```
// Ternary Operator:
```

```
let age = 20;  
let canVote = (age >= 18) ? 'Yes' : 'No';  
console.log(canVote); // Output: Yes
```

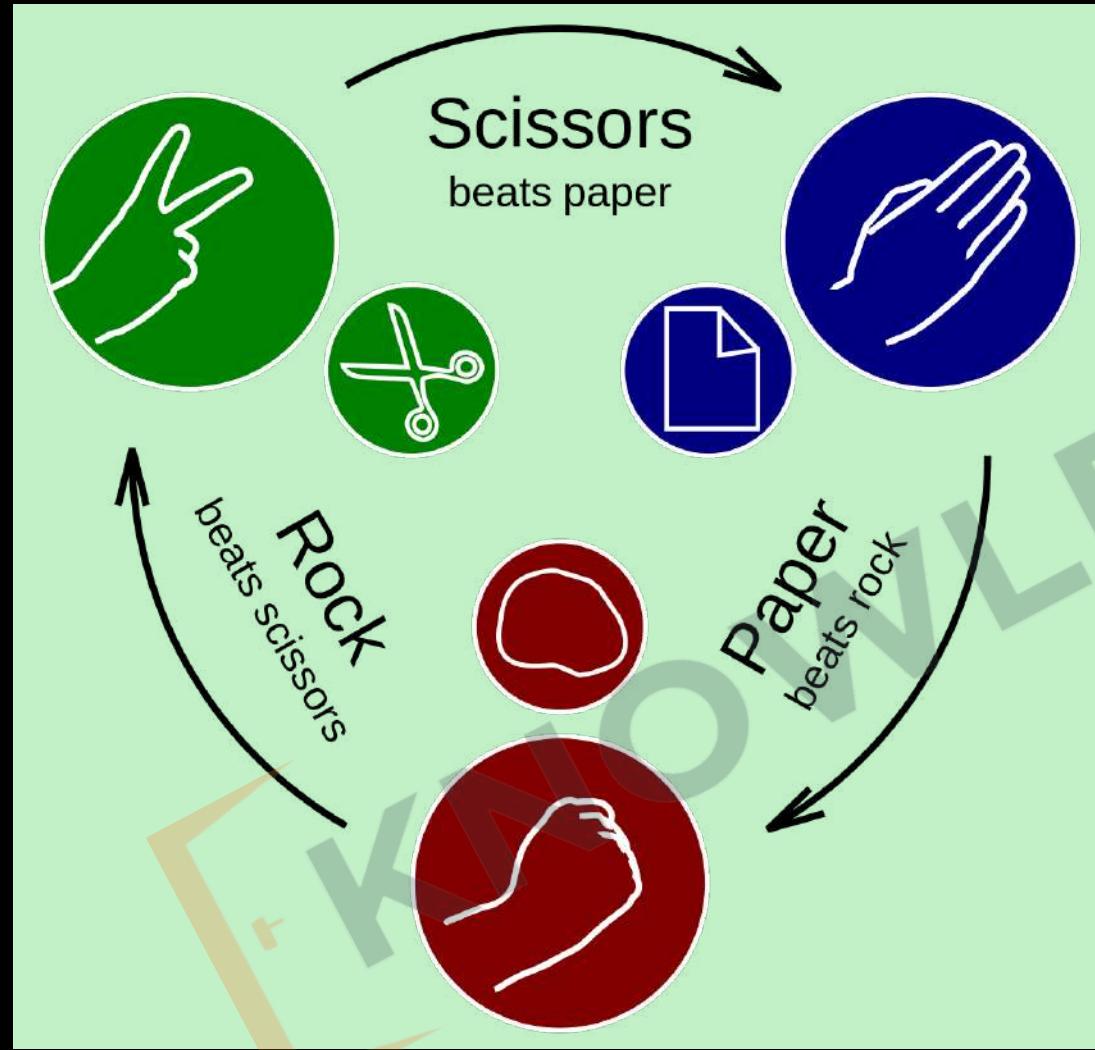
```
// Guard Operator:
```

```
let username = '';  
let defaultUsername = 'Guest';  
let displayName = username || defaultUsername;  
console.log(displayName); // Output: Guest
```

```
// Default Operator:
```

```
let userAge = null;  
let defaultAge = 18;  
let ageToDisplay = userAge ?? defaultAge;  
console.log(ageToDisplay); // Output: 18
```

Project Rock-Paper-Scissor Game



Rock Paper Scissors Game

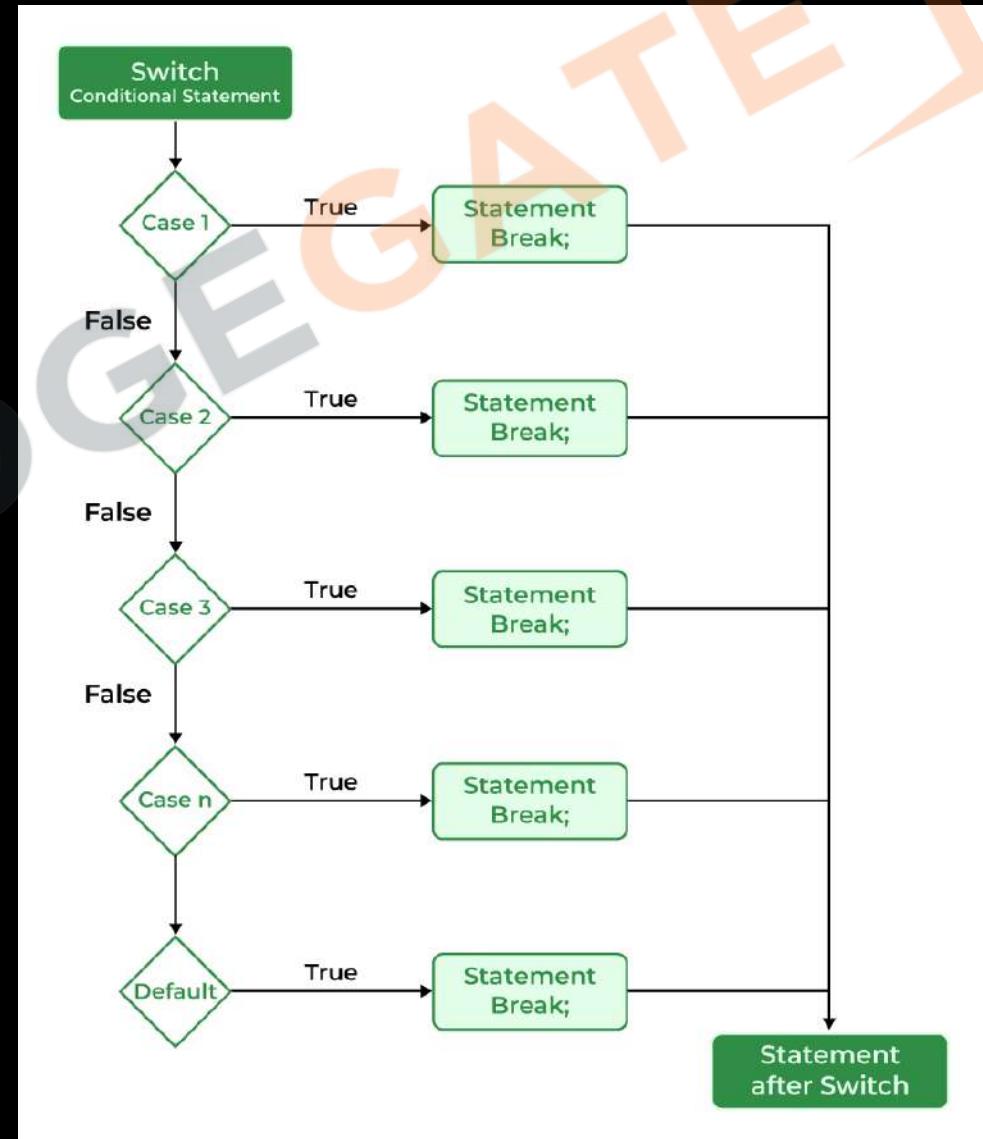
Click on one of the following to play the game:



- Use `Math.random` to do computer move.
- Use logical operators to compute the result

Switch

1. Multi-way Branching: switch provides a cleaner method for multi-way branching than multiple if-else statements when testing the same expression.
2. Case Labels: Represents individual branches. Execution jumps to the matching case label.
3. Break Statement: Typically used to exit the switch block after a case is executed to prevent "fall through" to subsequent cases.
4. Default Case: Optional. Executes if no case matches. Placed at the end of the switch block.
5. Enhances Readability: For certain types of conditional logic, switch can make the code more readable compared to nested if-else statements.
6. The switch statement compares the expression's value strictly (==) with the values of the case clauses.



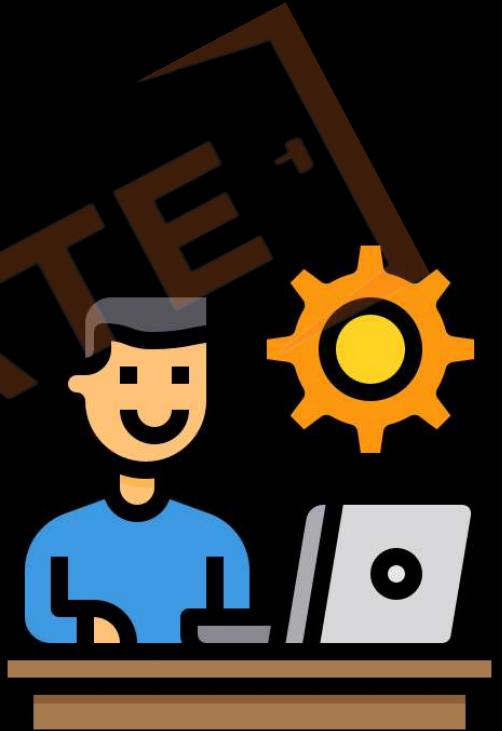
Switch

```
let day = 2;  
switch (dayNumber) {  
    case 1:  
        day = "Monday";  
        break;  
    case 2:  
        day = "Tuesday";  
        break;  
    case 3:  
        day = "Wednesday";  
        break;  
    case 4:  
        day = "Thursday";  
        break;  
    case 5:  
        day = "Friday";  
        break;  
    case 6:  
        day = "Saturday";  
        break;  
    case 7:  
        day = "Sunday";  
        break;  
    default:  
        day = "Invalid day";  
}  
console.log(day);
```

Practice Exercise

Decision Control

- If-Else Statement:** Write code that checks if a number is positive, negative, or zero. Use an `if-else` statement for this purpose.
- Nested If Statement:** Write code to determine the ticket price for a movie. The function should consider the following:
 - If the viewer is under 13, the ticket is free.
 - If the viewer is between 13 and 60, check if it's a weekend. If yes, the ticket price is Rs 500; otherwise, it's Rs 300.
 - If the viewer is over 60, the ticket price is Rs 250.
- If-Else If Ladder:** Write code to determine grades based on marks:
 - Above 90 is 'A'.
 - 80 to 89 is 'B'.
 - 70 to 79 is 'C'.
 - 60 to 69 is 'D'.
 - Below 60 is 'F'.
- Ternary Operator:** Use the ternary operator in JavaScript to assign a value to a variable named `status`. The value should be "Adult" if the age is 18 or above, and "Minor" otherwise.



Practice Exercise

Decision Control

5. **Switch Case:** Write a JavaScript switch case statement that evaluates the variable `day` and returns a specific greeting:
 - "Happy Monday!" for Monday.
 - "Terrific Tuesday!" for Tuesday.
 - "Wonderful Wednesday!" for Wednesday.
 - "Thriving Thursday!" for Thursday.
 - "Fun Friday!" for Friday.
 - "Super Saturday!" for Saturday.
 - "Serene Sunday!" for Sunday.
6. **Comparison and Logical Operators:** Write a JavaScript expression using logical operators that checks if a variable `age` is either below 13 or above 65. Also, use a comparison operator to determine if another variable `income` is greater than or equal to 50000.
7. **Guard Operator:** Demonstrate the use of the guard operator to handle null or undefined values. Write code where a variable `userInput` is checked. If `userInput` is null or undefined, assign "No input provided" to another variable `output`.



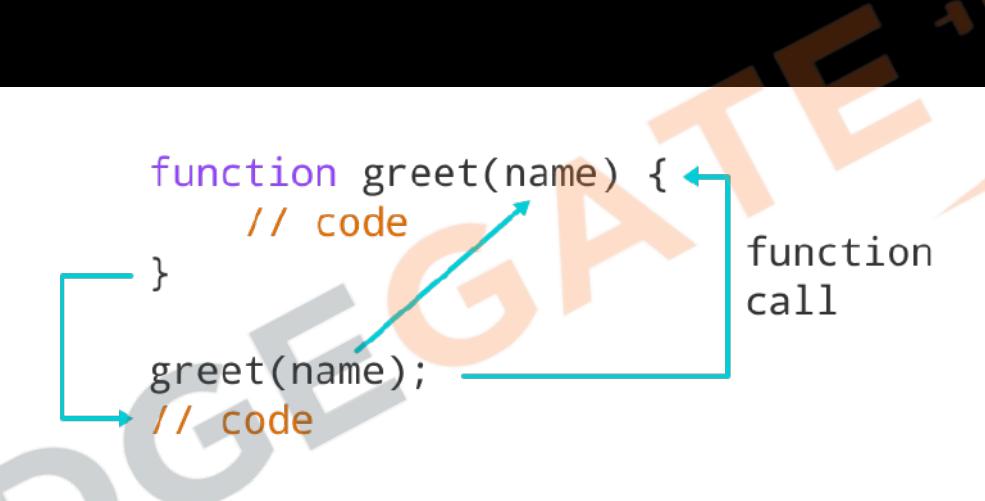
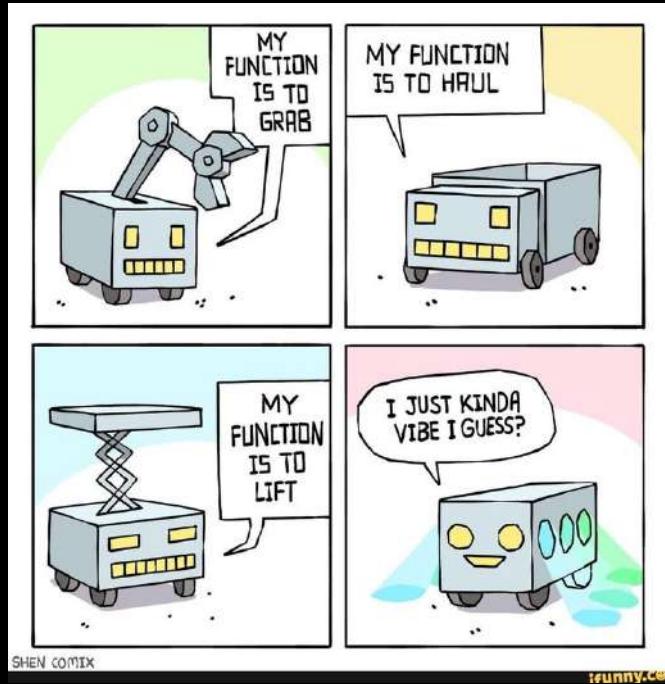


Functions

- What are Functions
- Function Syntax
- Return statement
- Function Parameters
- Call by Value
- Variable Scope

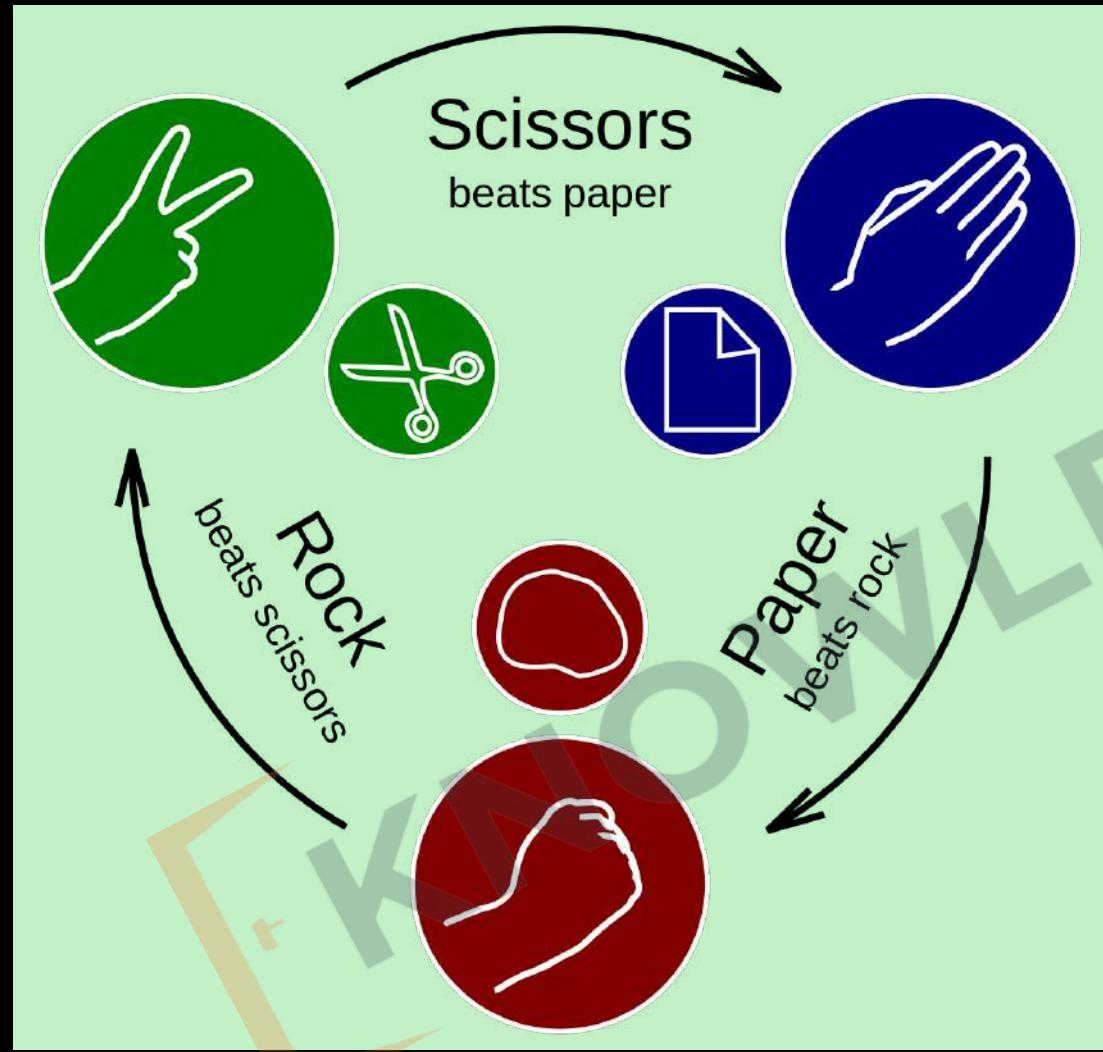
Improve    Project

What are Functions?



1. **Definition:** Blocks of **reusable** code.
2. **DRY Principle:** "Don't Repeat Yourself" it Encourages code reusability.
3. **Usage:** Organizes **code** and performs specific tasks.
4. **Naming Rules:** Same as **variable** names: **camelCase**
5. **Example:** "Beta Gas band kar de"

Project Rock-Paper-Scissor Game



Rock Paper Scissors Game

Click on one of the following to play the game:

✊ Rock

✋ Paper

✌ Scissors

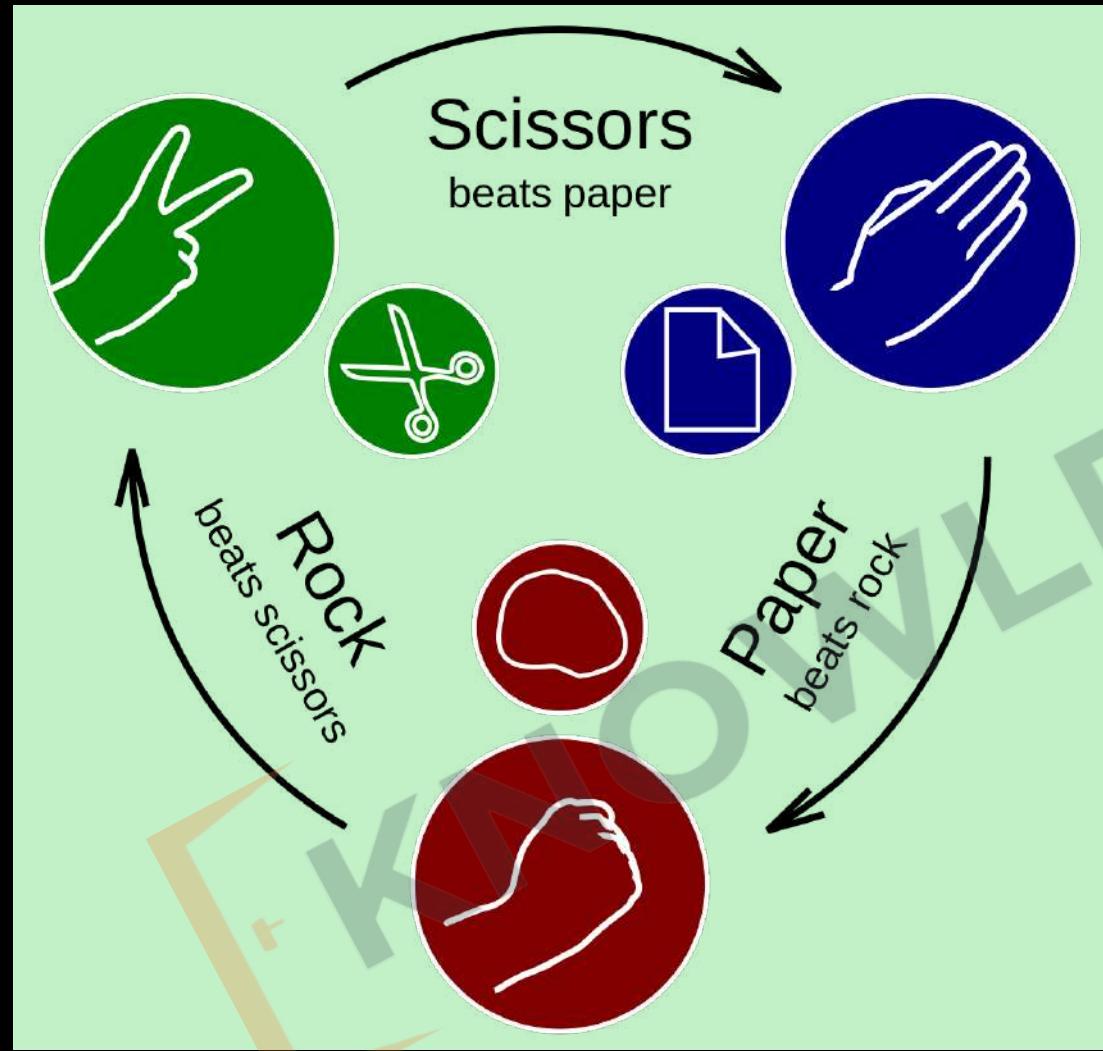
Create functions for onclick, for
Random Number & Computer Choice

Return statement



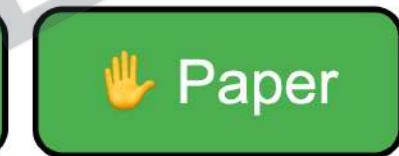
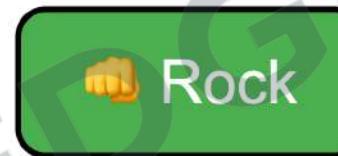
1. Sends a value back from a function.
2. Example: "Ek glass paani laao"
3. What Can Be Returned: Value, variable, calculation, etc.
4. Return ends the function immediately.
5. Function calls make code jump around.
6. Prefer returning values over using global variables.

Project Rock-Paper-Scissor Game



Rock Paper Scissors Game

Click on one of the following to play the game:



User return instead of Global Variable

Parameters



Parameter



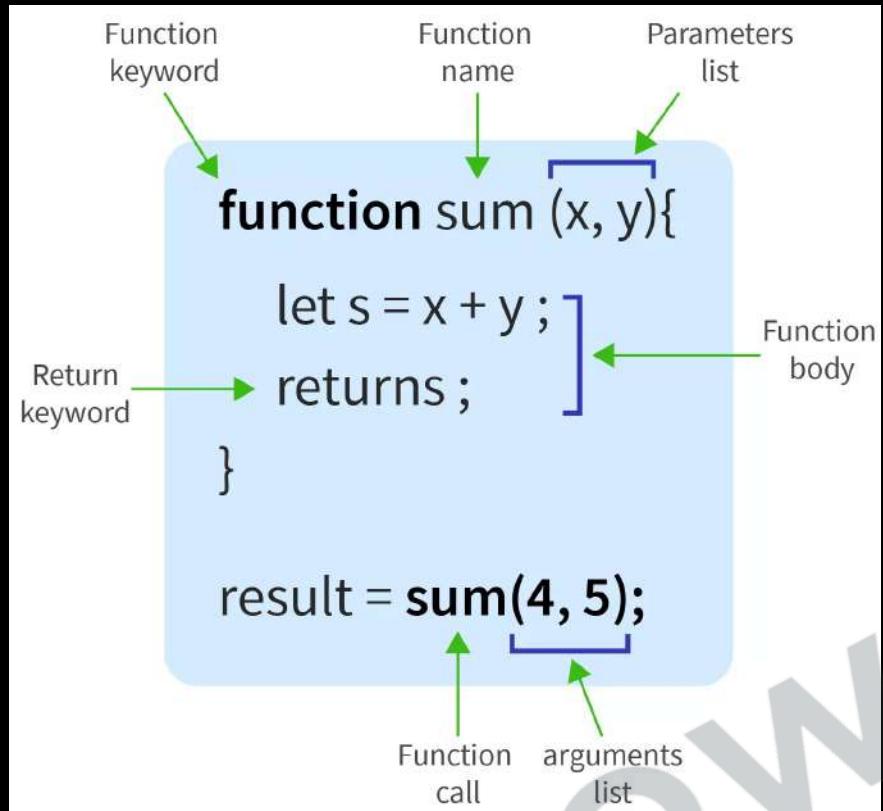
Function



Return

1. Input values that a **function** takes.
2. Parameters put value into function, while return gets value out.
3. Example: "Ek packet dahi laao"
4. Naming Convention: Same as **variable** names.
5. **Parameter** vs **Argument**
6. Examples: `alert`, `Math.round`, `console.log` are functions we have already used
7. Multiple Parameters: Functions can take **more than one**.
8. Default Value: Can set a **default** value for a parameter.

Functions Syntax



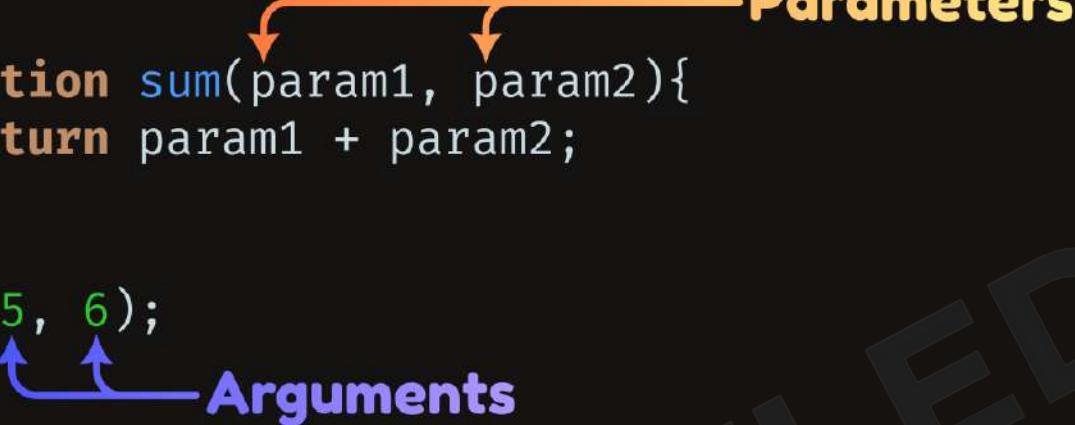
1. Use **function** keyword to declare.
2. Follows same rules as **variable names**.
3. Use **()** to contain parameters.
4. Invoke by using the **function name followed by ()**.
5. Fundamental for **code organization and reusability**.

Argument vs Parameter

```
function sum(param1, param2){  
    return param1 + param2;  
}  
  
sum(5, 6);
```

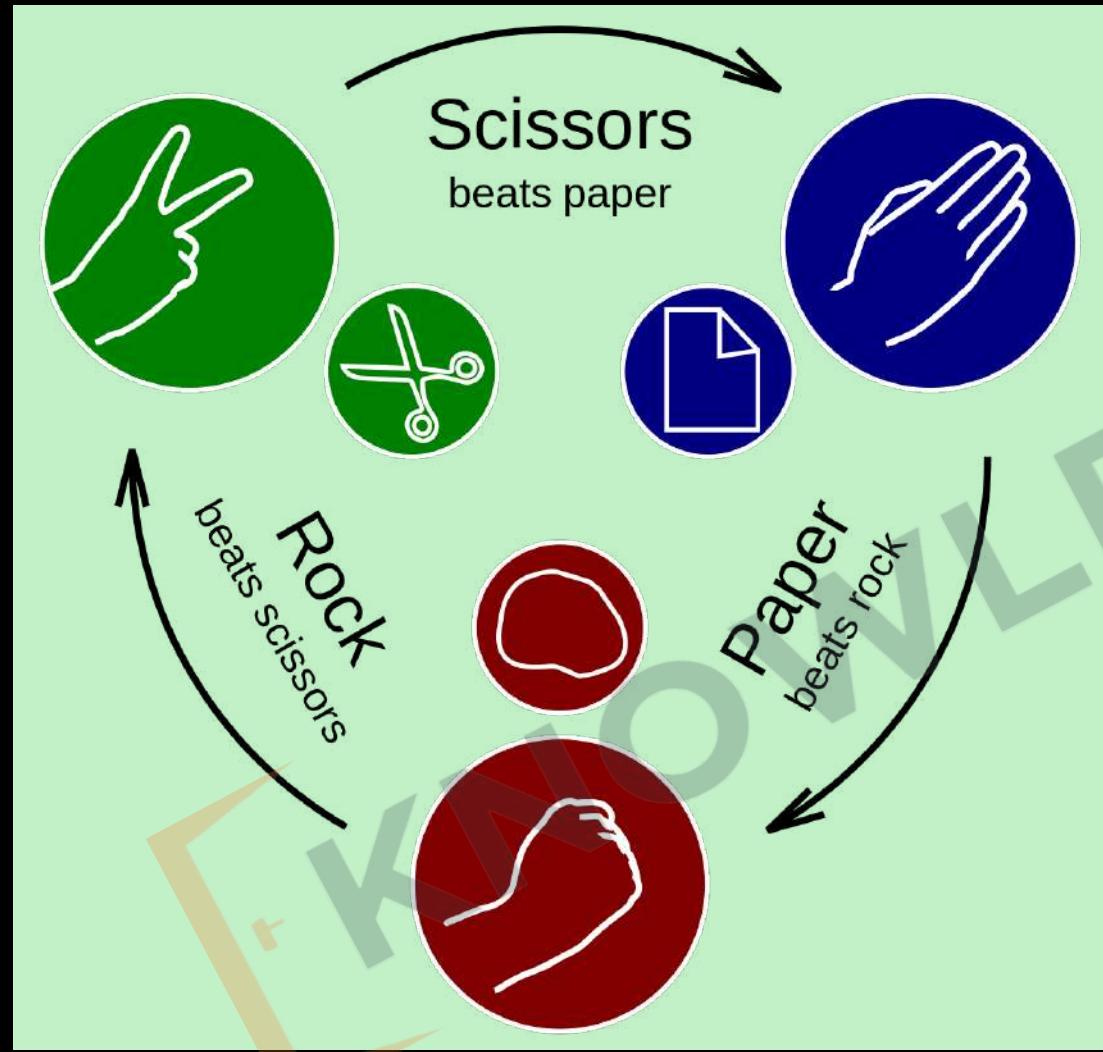
Parameters

Arguments



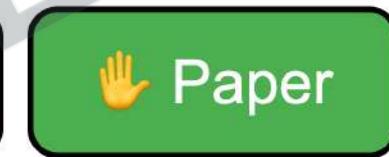
1. Parameters: Variables in a **function** definition, acting as placeholders.
2. Arguments: **Actual values** passed to a function at call time.

Project Rock-Paper-Scissor Game



Rock Paper Scissors Game

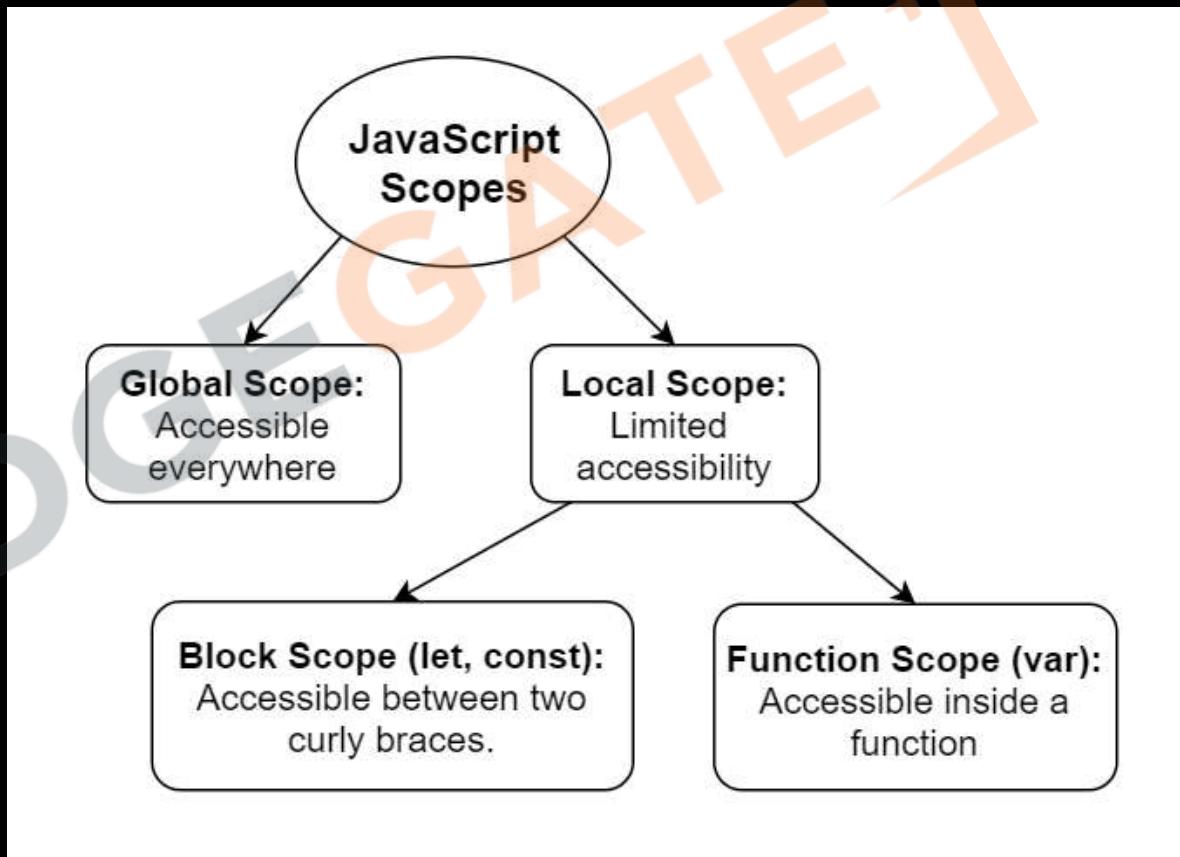
Click on one of the following to play the game:



Create function for comparing user choice
& Showing Result alert

Variable Scope

1. **No Keyword:** Declares a variable globally, regardless of block scope, often leading to **unintended consequences** if not managed carefully.
2. **var:** Declares a variable with **function scope**, subject to hoisting, allowing it to be accessed (as `undefined`) before its actual declaration line.
3. **let:** Declares a **block-scoped variable**, accessible only after its declaration and not subject to hoisting, preventing use before declaration.
4. **const:** Declares a block-scoped variable **that must be initialized at the time of declaration** and cannot be reassigned, ensuring immutable bindings (though not immutable values).
5. Declare variables in the **narrowest scope possible**.



Variable Scope

Feature	No Keyword	var	let	const
Scope	Global unless in a module	Function scope	Block scope	Block scope
Hoisting	Not Hoisted	Hoisted within function	Temporal Dead Zone (TDZ)	Temporal Dead Zone (TDZ)
Re-declaration	Allowed globally	Allowed within scope	Not allowed	Not allowed
Re-assignment	Allowed	Allowed	Allowed	Not allowed
Use in Strict Mode	Error if used	Allowed	Allowed	Allowed
Initialization Requirement	Not required	Not required	Not required	Required
Typical Use Case	Avoid in modern JS	Legacy JS code	General variables	Constants

Variable Scope

```
// Example of 'no keyword' declaration and redefining
function globalTestNoKeyword() {
    // Global variable are not hoisted
    //console.log(globalVar); // ReferenceError: globalVar is not defined
    globalVar = "I am global initially"; // Declared without any keyword, defaults to global
    scope
    console.log(globalVar); // Output: I am global initially
    globalVar = "I am globally redefined"; // Redefining the variable
    console.log(globalVar); // Output: I am globally redefined
}
globalTestNoKeyword();
console.log(globalVar); // Output: I am globally redefined

// Example of 'var' declaration, hoisting, and redefining
function varTest() {
    console.log(hoistedVar); // Output: undefined due to hoisting
    var hoistedVar = "I am hoisted";
    console.log(hoistedVar); // Output: I am hoisted
    var hoistedVar = "I am redefined"; // Redefining the variable
    console.log(hoistedVar); // Output: I am redefined
}
varTest();
```

Variable Scope

```
// Example of 'let' declaration and redefining within the same scope
function letTest() {
    //console.log(blockVar); // ReferenceError: Cannot access 'blockVar' before initialization
    let blockVar = "I am block-scoped";
    console.log(blockVar); // Output: I am block-scoped
    //let blockVar = "redefined"; // SyntaxError: Identifier 'blockVar' has already been declared
    console.log(blockVar); // Output: I am redefined
}
letTest();
```

```
// Example of 'const' declaration and attempted redefining
function constTest() {
    const constantVar = "I am constant";
    console.log(constantVar); // Output: I am constant
    // constantVar = "Attempt to redefine"; // TypeError: Assignment to constant variable
}
constTest();
```

Call by Value

```
// Function to try to swap two numbers
function trySwap(a, b) {
    let temp = a;
    a = b;
    b = temp;
    console.log(`Inside trySwap - a: ${a}, b: ${b}`);
}

function main() {
    let x = 10, y = 20;
    console.log(`Before trySwap - x: ${x}, y: ${y}`);
    trySwap(x, y); // Attempt to swap x and y
    // The original values are unchanged
    console.log(`After trySwap - x: ${x}, y: ${y}`);
}

main();
```

Before trySwap - x: 10, y: 20

Inside trySwap - a: 20, b: 10

After trySwap - x: 10, y: 20

1. **Value Copy:** Passes argument's copy, not the original.
2. **Separate Memory:** Parameters use distinct memory locations.
3. **Data Safety:** Original data remains unchanged by the function.
4. **Direct Modification:** Cannot modify original arguments directly.
5. **Efficiency:** Good for small data types, less so for large structures.
6. **Ease of Use:** Straightforward and safe for functions not altering inputs.

Practice Exercise

Functions

1. Create a function to check if a number is odd or even.
2. Create a function to return larger of the two numbers.
3. Create function to convert Celsius to Fahrenheit
 $F = (9/5) * C + 32$
4. Define a function square that takes a number and returns its square.
5. Demonstrate with a function increment that the original number passed to it does not change after incrementing it inside the function.
6. Define a function getAverage that takes five numbers and returns the average.
7. Create a function concatStrings that takes two strings and returns their concatenation.



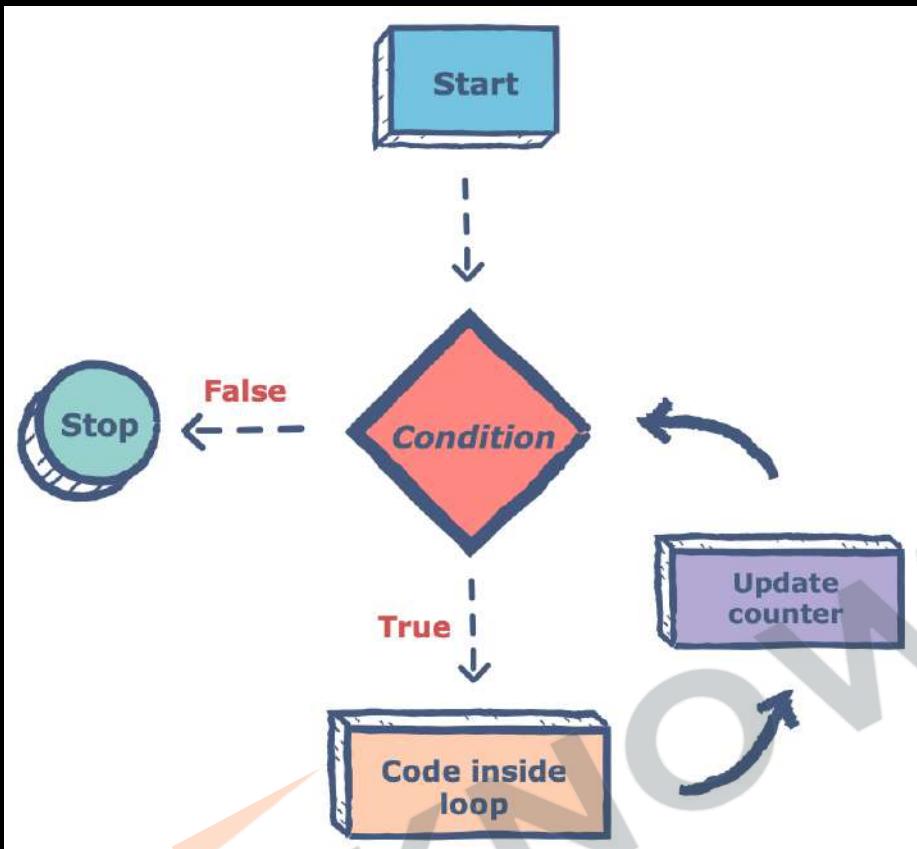


Iteration & Loop Control Structure

- Need of loops
- While Loop
- Do-while Loop
- For Loop
- Break statement
- Continue statement
- Odd Loop
- Infinite Loop
- Accumulator Pattern



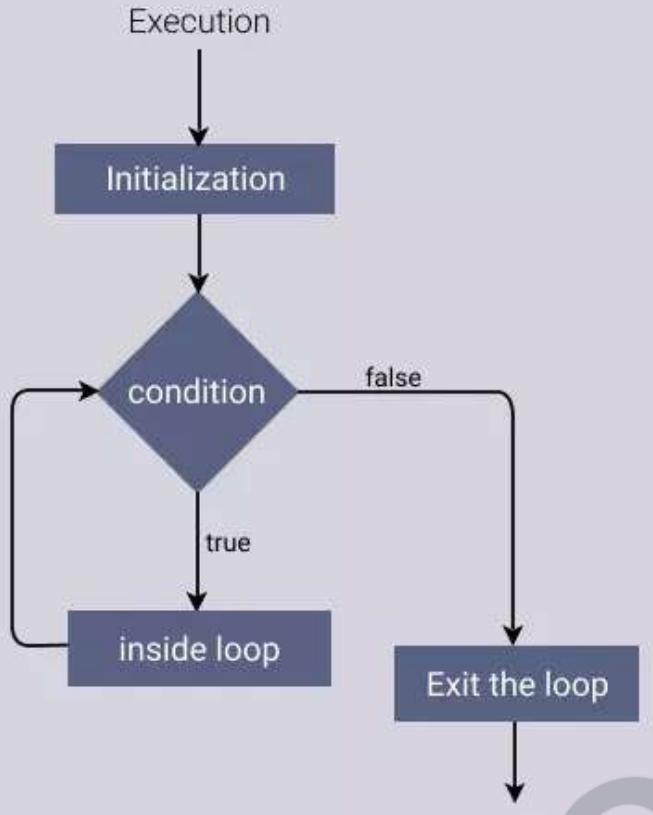
What is a Loop?



1. Code that runs **multiple times** based on a condition.
2. Loops also **alter the flow of execution**, similar to functions.
 - Functions: **Reusable** blocks of code.
 - Loops: **Repeated execution** of code.
3. Loops automate **repetitive** tasks.
4. Types of Loops: **for, while, do-while**.
5. Iterations: Number of **times** the loop runs.

While Loop

Execution



```
while (condition) {  
    // Body of the loop  
}
```

```
let i = 0;  
while (i < 5) {  
    console.log(i);  
    i++;  
}  
// Output: 0, 1, 2, 3, 4
```

1. Iterations: Number of times the loop runs.
2. Used for non-standard conditions.
3. Repeating a block of code while a condition is true.
4. Remember: Always include an update to avoid infinite loops.

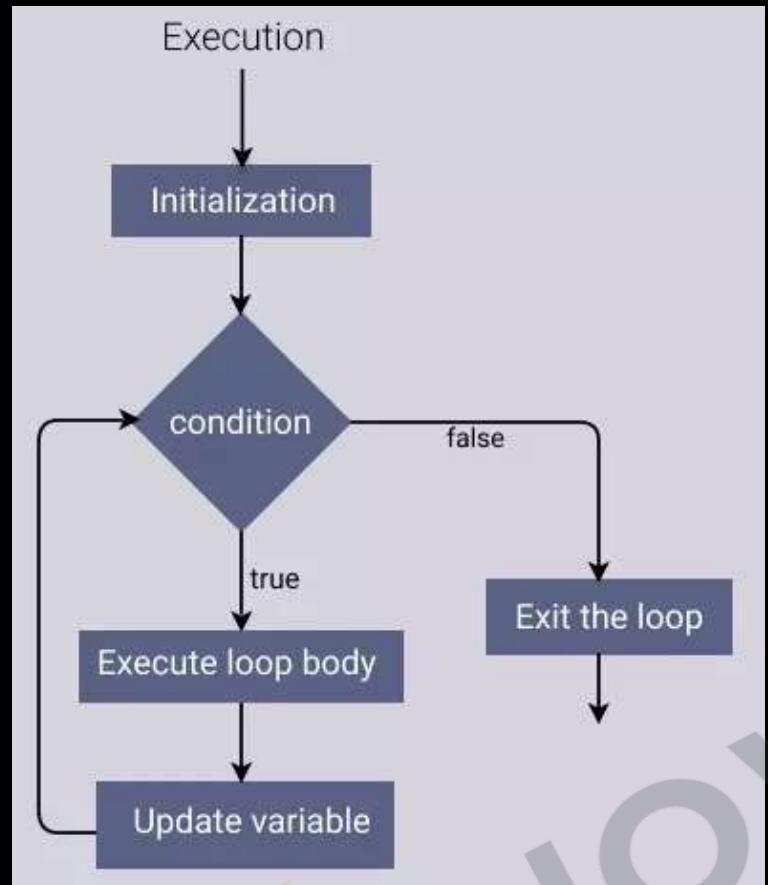
Practice Exercise

While Loop

- Develop a program using while that prints the multiplication table for a given number.
- Create a program to sum all odd numbers from 1 to a specified number N.
- Write a function that calculates the factorial of a given number.
- Create a program that computes the sum of the digits of an integer.
- Create a program to find the Least Common Multiple (LCM) of two numbers.
- Create a program to find the Greatest Common Divisor (GCD) of two integers.
- Create a program to check whether a given number is prime using while.



For Loop



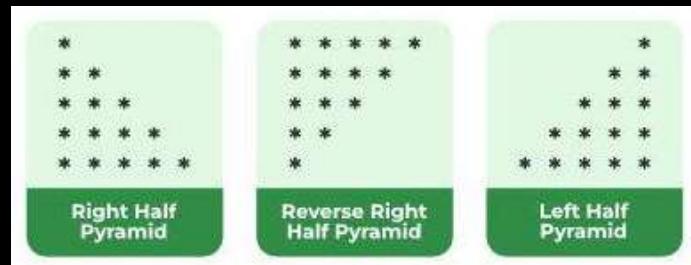
```
for (initialisation; condition; update) {  
    // Body of the loop  
}  
  
for (let i = 0; i< 5; i++) {  
    console.log(i);  
}  
// Output: 0, 1, 2, 3, 4
```

1. Standard loop for running code multiple times.
2. Generally preferred for counting iterations.

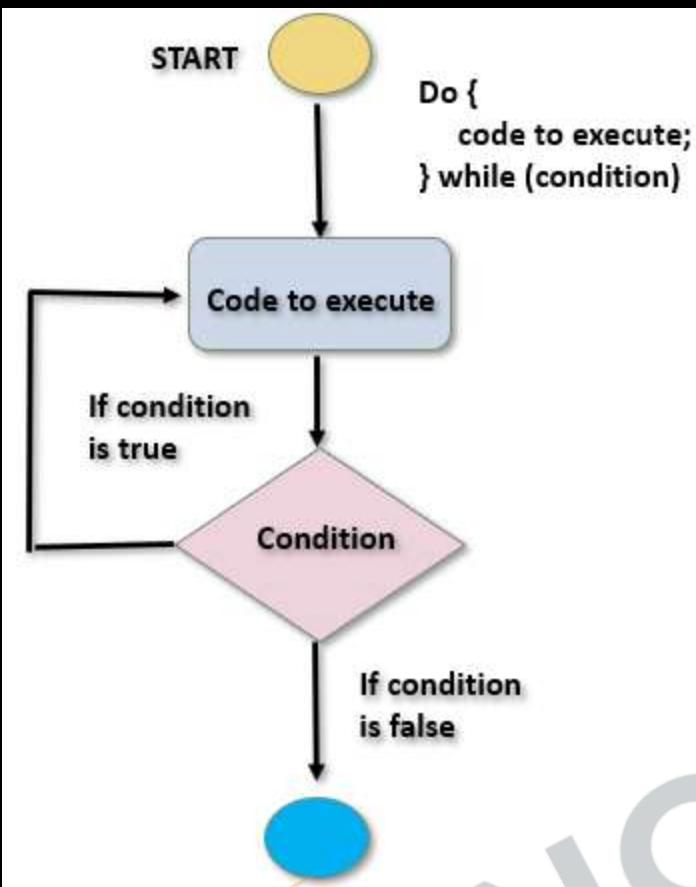
Practice Exercise

For Loop

- Create a program to reverse the digits of a number.
- Create a program to print the Fibonacci series up to a certain number.
- Create a program to check if a number is an Armstrong number.
- Create a program to verify if a number is a palindrome.
- Create a program using for loop multiplication table for a number.
- Create a program using for to display if a number is prime or not.
- Create a program that print patterns:



Do While Loop



```
let userInput;  
  
do {  
    | | userInput = prompt("Enter a number greater than 10:");  
} while (userInput <= 10);  
  
console.log("You entered a valid number: " + userInput);  
  
do {  
    // Body of the loop  
}  
while (condition);
```

1. Executes **block first**, then checks condition.
2. Guaranteed to run **at least one** iteration.
3. Unlike **while**, first iteration is **unconditional**.
4. Don't forget to update condition to avoid **infinite loops**.

Practice Exercise

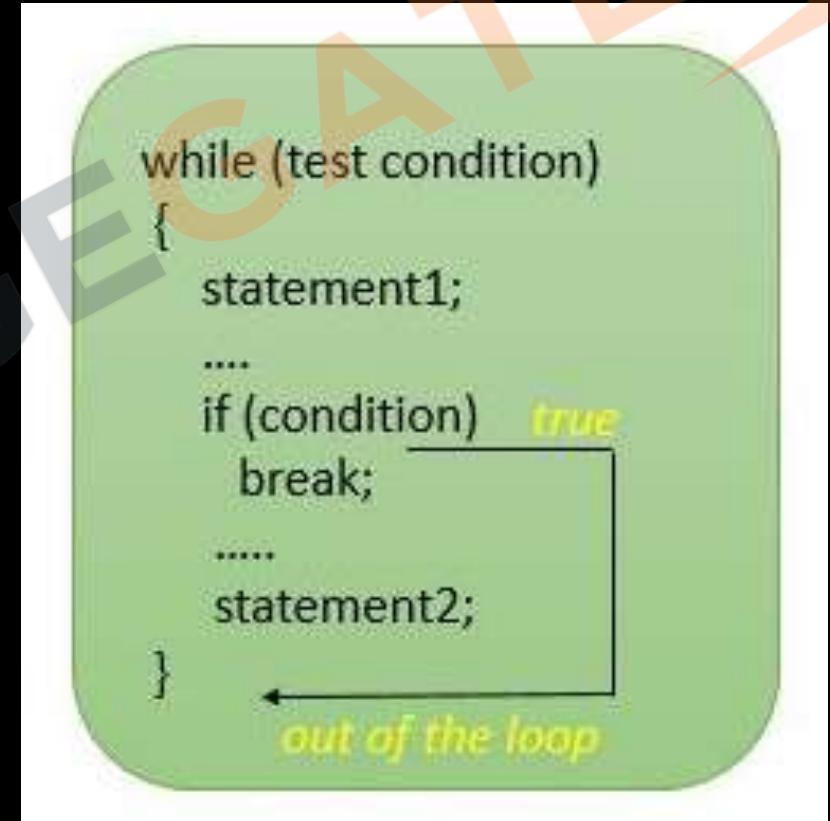
Do-While Loop

- Create a program that prompts the user to enter a positive number. Use a do-while loop to keep asking for the number until the user enters a valid positive number.
- Develop a program that calculates the sum of all numbers entered by a user until the user enters 0. The total sum should then be displayed.



Break statement

1. Break lets you stop a loop early, or break out of a loop
2. Exits Loops: Ends for, while, do-while loops early.
3. Ends Switch Cases: Prevents fall-through in switch cases.
4. Immediate Effect: Immediately leaves the loop/switch.
5. Controls Flow: Alters program flow for efficiency.
6. Use Wisely: Important for readability.

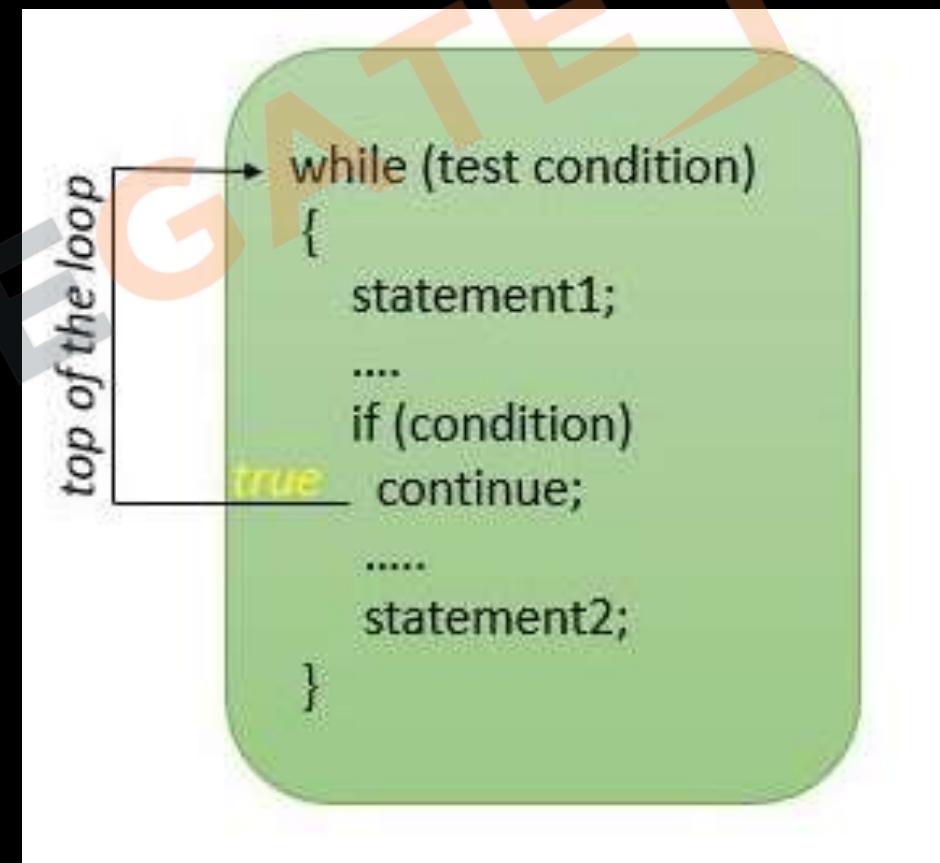


Break statement

```
for (let i = 0; i < 10; i++) {  
  if (i === 5) {  
    | | break; // Exit the loop when i is 5  
  }  
  console.log(i);  
}  
// Output: 0, 1, 2, 3, 4
```

Continue statement

1. Continue is used to skip one iteration or the current iteration
2. Next Iteration: Immediately starts the next iteration of the loop.
3. In while loop remember to do the increment manually before using continue.
4. Used in Loops: Works within for, while, do-while loops.
5. Not for switch: Unlike break, not used in switch statements.
6. Improves Logic: Helps in avoiding nested conditions within loops.



Continue statement

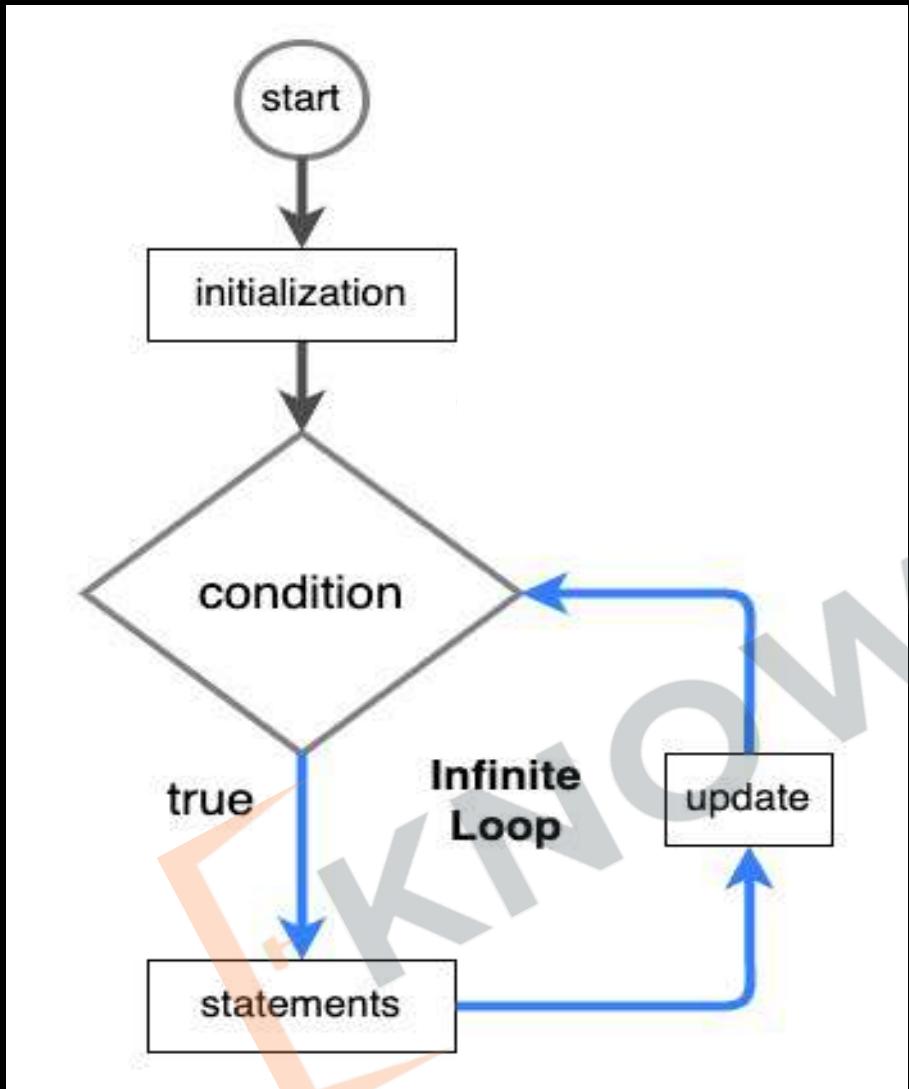
```
for (let i = 0; i < 10; i++) {  
  if (i === 5) {  
    | | continue; // Skip the iteration when i is 5  
  }  
  console.log(i);  
}  
// Output: 0, 1, 2, 3, 4, 6, 7, 8, 9
```

Odd Loop

```
let another, num;  
do {  
    num = parseInt(prompt("Enter a number: "), 10);  
    console.log(`Square of ${num} is ${num * num}`);  
    another = prompt("Want to enter another number (y/n)? ");  
} while (another === 'y');
```

1. Condition-Driven: Run until a specific condition is fulfilled.
2. While and Do-While: Commonly used for indeterminate iterations.
3. Dynamic Iteration: Iterations depend on changing conditions or input.
4. Break Usage: May use break for exit in any loop type.
5. Practical Use: Ideal for processing with unknown completion point.
6. Design Carefully: Important to avoid infinite loops by ensuring a valid exit condition.

Infinite Loop



```
let i = 1;  
while (true) {  
    console.log(i);  
    i++;  
}
```

1. **Endless Execution:** They run continuously.
2. **Purposeful or Accidental:** Used deliberately or by mistake.
3. **Exit Strategy:** Require **break** or similar statements for stopping.
4. **Resource Intensive:** May cause high CPU usage.

Accumulator Pattern

ACCUMULATE



1. A pattern to accumulate values through looping.
2. Common Scenarios:
 - Sum all the numbers.
 - Create a modified copy of an array.

Accumulator Pattern

```
let sum = 0;  
let input;  
  
do {  
    input = parseInt(prompt("Enter a number (or type 'stop' to  
    finish): "), 10);  
    if (!isNaN(input)) {  
        sum += input;  
    }  
} while (!isNaN(input));  
  
console.log("Total Sum:", sum);
```

Practice Exercise

Loops

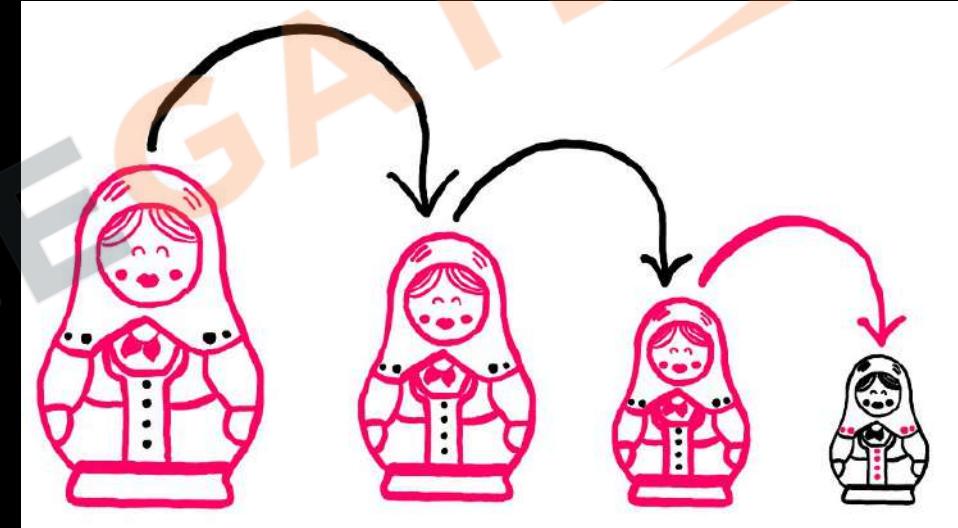
- Create a program using `continue` to sum all positive numbers entered by the user; skip any negative numbers.
- Create a program using `continue` to print only even numbers using `continue` for odd numbers.
- Write a program that continuously reads integers from the user and prints their squares. Use an infinite loop and a `break statement` to exit when a special number (e.g., -1) is entered.





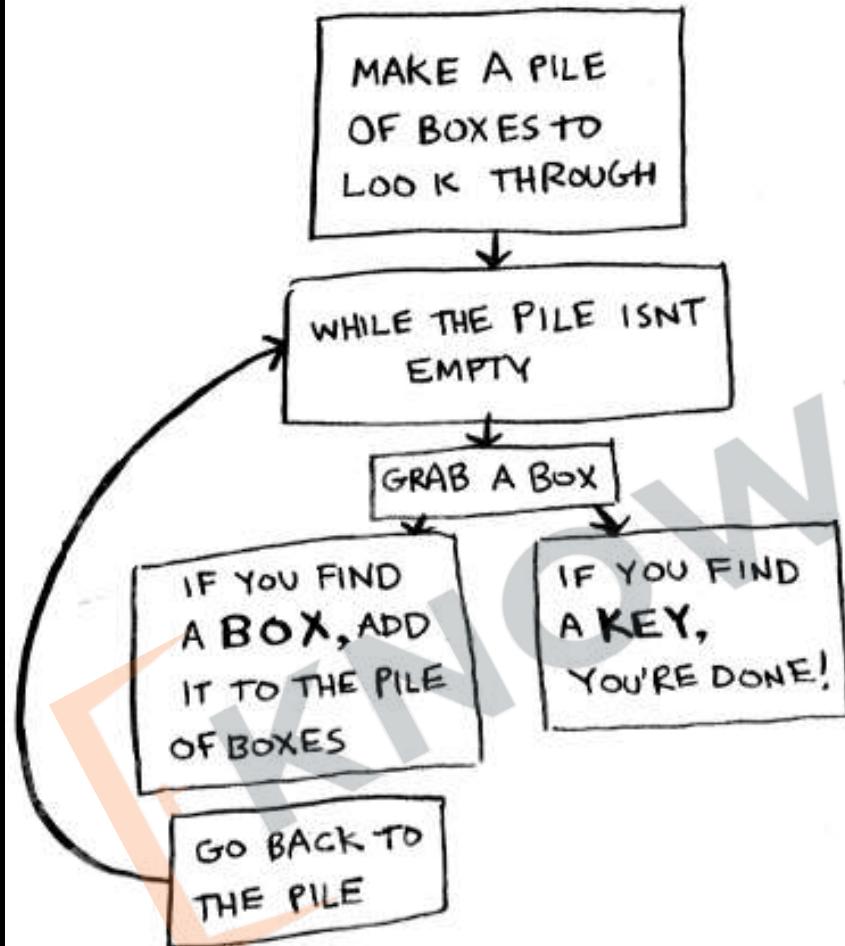
Recursion

1. **Self-Calling:** A **function that calls itself** to solve sub-problems.
2. **Base Case:** Essential to stop recursion and prevent infinite loops.
3. **Recursive Case:** The **condition** under which the function keeps calling itself.
4. **Stack Usage:** Consumes **stack space** with each **call**, risk of overflow.
5. **Applications:** Ideal for **divisible tasks**, tree/graph traversals, sorting algorithms.
6. **Memory Intensive:** More overhead than iterative solutions.
7. **Clarity:** Often **simplifies complex problems**.

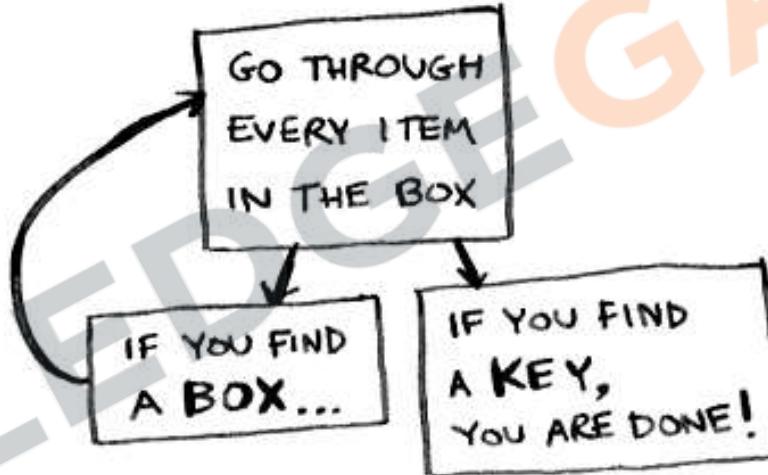


Iterative vs Recursive

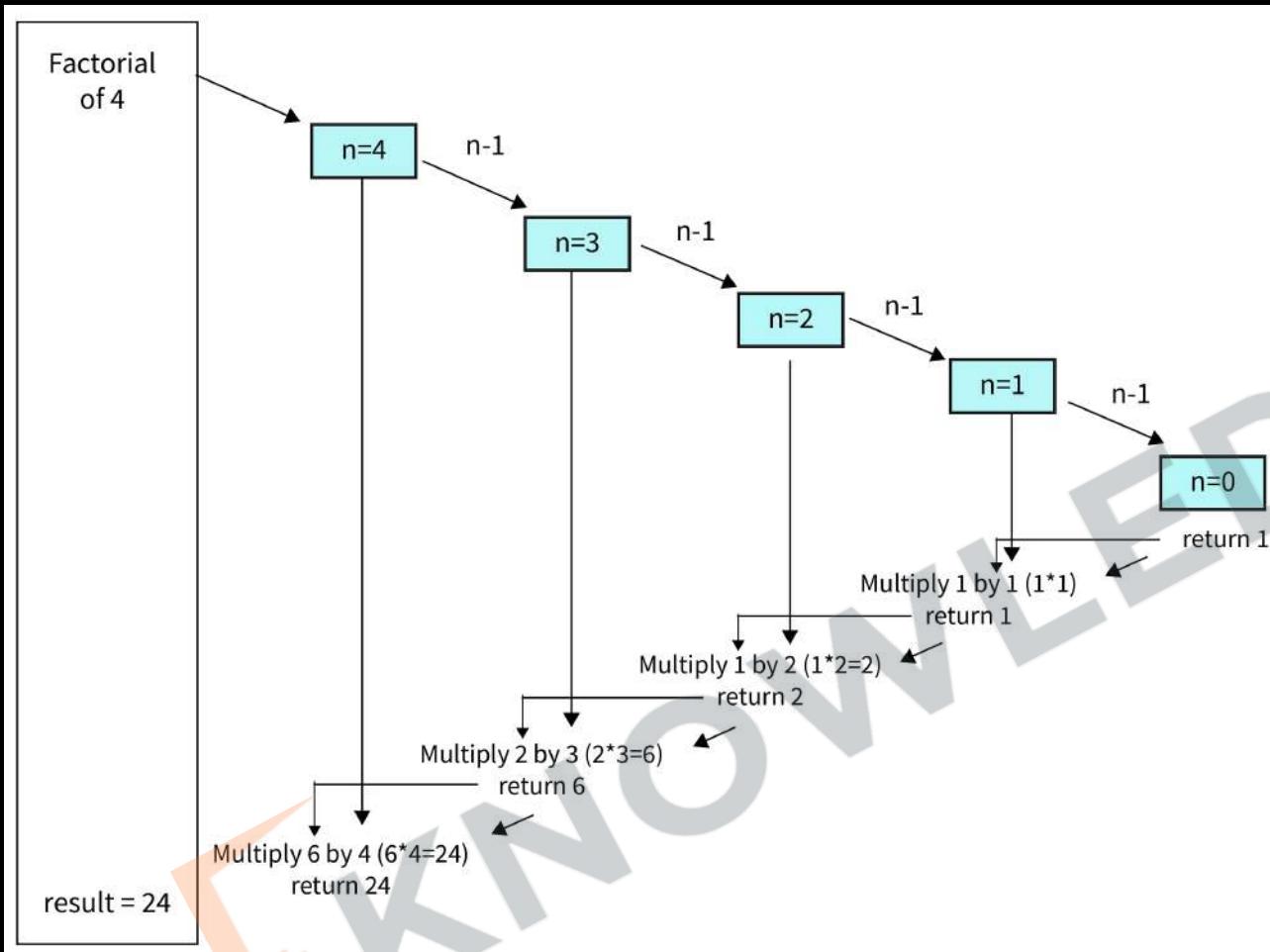
Iterative Approach



Recursive Approach

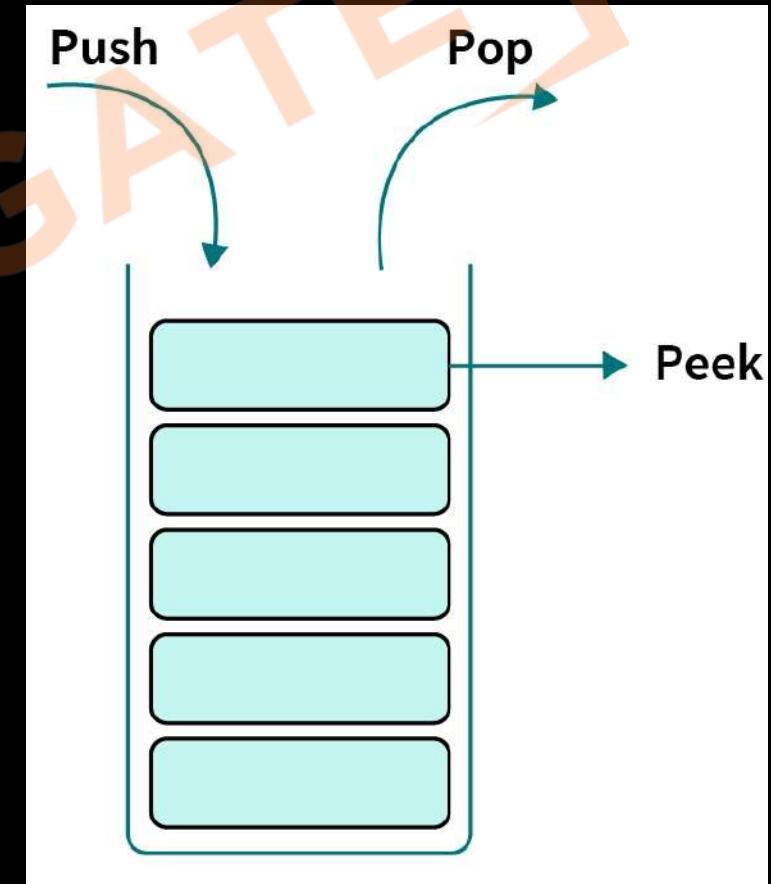
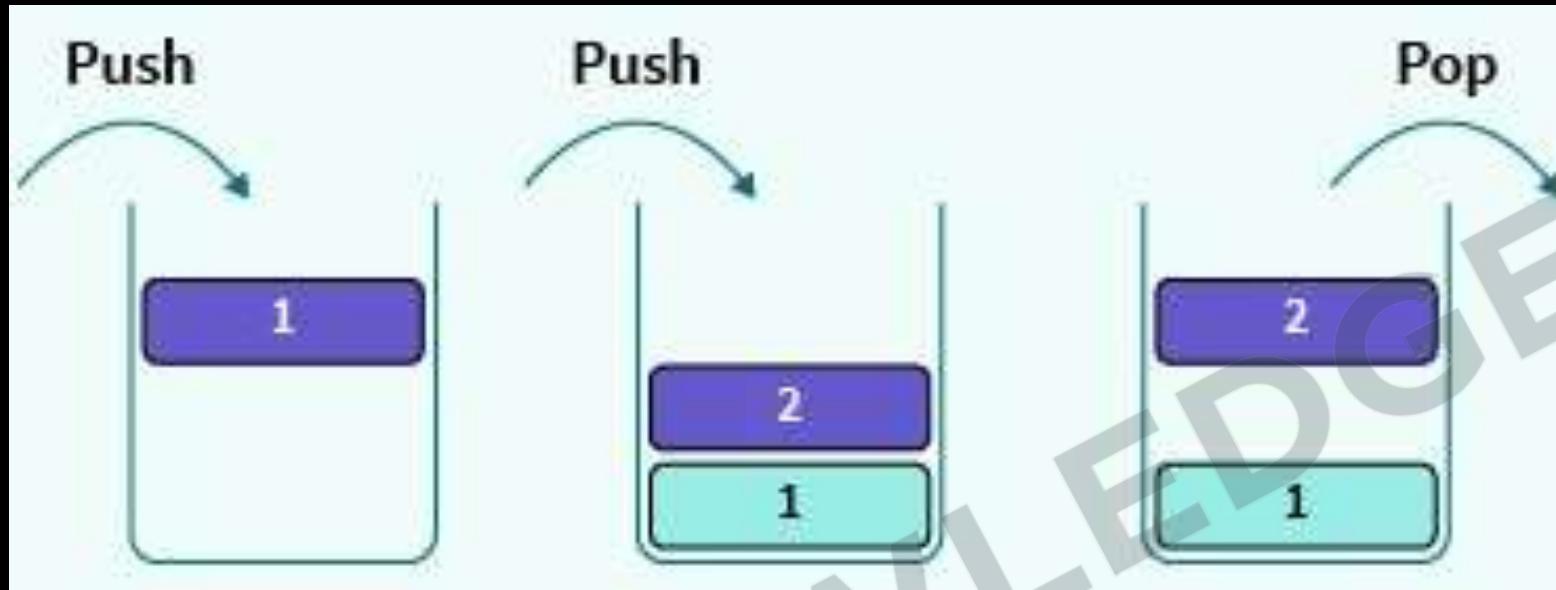


Factorial Using Recursion



```
function factorial(n) {  
    // Base case: factorial of 0 is 1  
    if (n === 0) {  
        return 1;  
    }  
    // Recursive case: n! = n * (n-1)!  
    return n * factorial(n - 1);  
}  
  
// Example usage:  
console.log(factorial(5)); // Output: 120  
console.log(factorial(0)); // Output: 1  
console.log(factorial(3)); // Output: 6
```

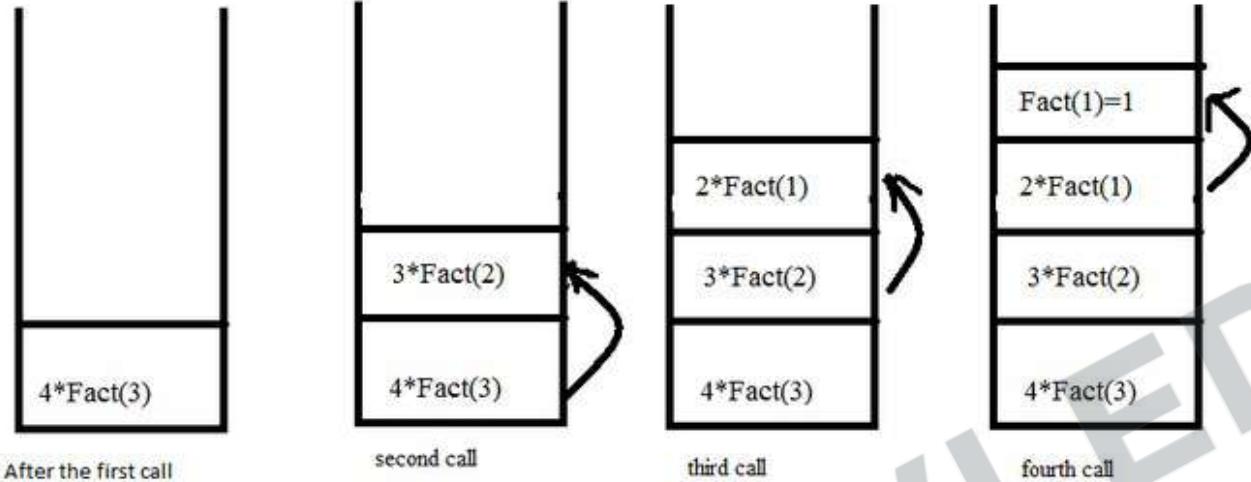
What is Stack



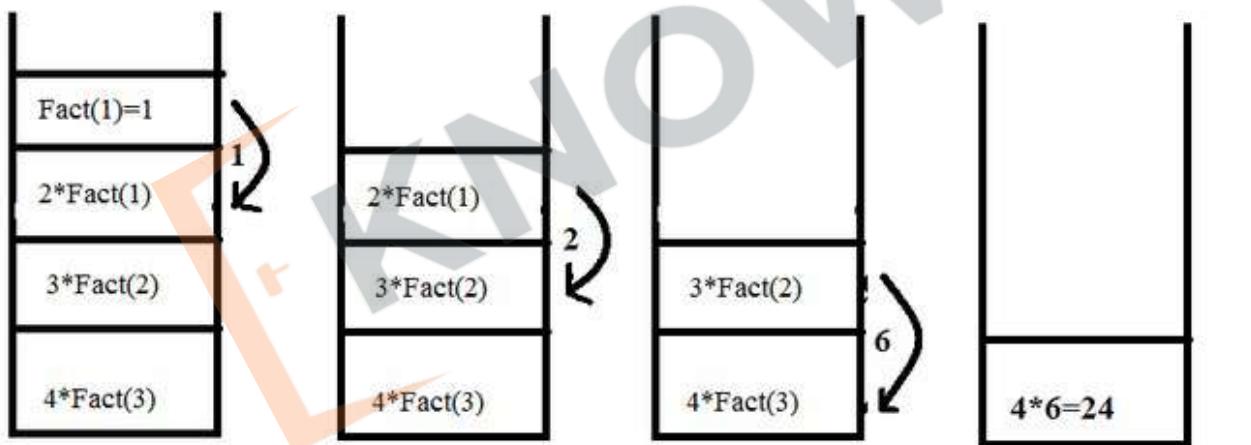
1. **LIFO:** Last-In, First-Out operation; the last element added is the first to be removed.
2. **Operations:** Mainly push (add an item) and pop (remove an item).
3. **Top Element:** Only the top element is accessible at any time, not the middle or bottom ones.
4. **Overflow and Underflow:** Overflow when full, underflow when empty during operations.

Recursion Stack

When function call happens previous variables gets stored in stack



Returning values from base case to caller function



- Memory Allocation:** Recursive calls add frames to the call stack for variables and return points.
- Growth with Depth:** Deeper recursion equals more stack space.
- Base Case:** Essential to limit recursion depth and prevent stack overflow.
- Stack Overflow Risk:** Excessive recursion depth can crash the program.
- Tail Recursion Optimization:** Can reduce stack usage for certain recursive patterns.
- Efficiency:** Iterative solutions may be more stack-efficient than recursion for some problems.

Practice Exercise

Recursion

1. Write a recursive function to print all numbers from 1 to N.
2. Create a function to calculate Fibonacci element using recursion.
3. Write a recursive function to find the sum of digits of a given positive integer n.
4. Write a recursive function to calculate x raised to the power of n (i.e., x^n).
5. Write a recursive function to find the greatest common divisor (GCD) of two numbers a and b.
6. Write a recursive function to count the number of times a specific character appears in a string.
7. Write a recursive function to check if a given string is a palindrome (reads the same forwards and backwards).





Objects

- What is an Object
- Objects Syntax
- Accessing Objects
- Inside Objects
- Autoboxing
- Object References
- Object Shortcuts

Maintain score of    Project

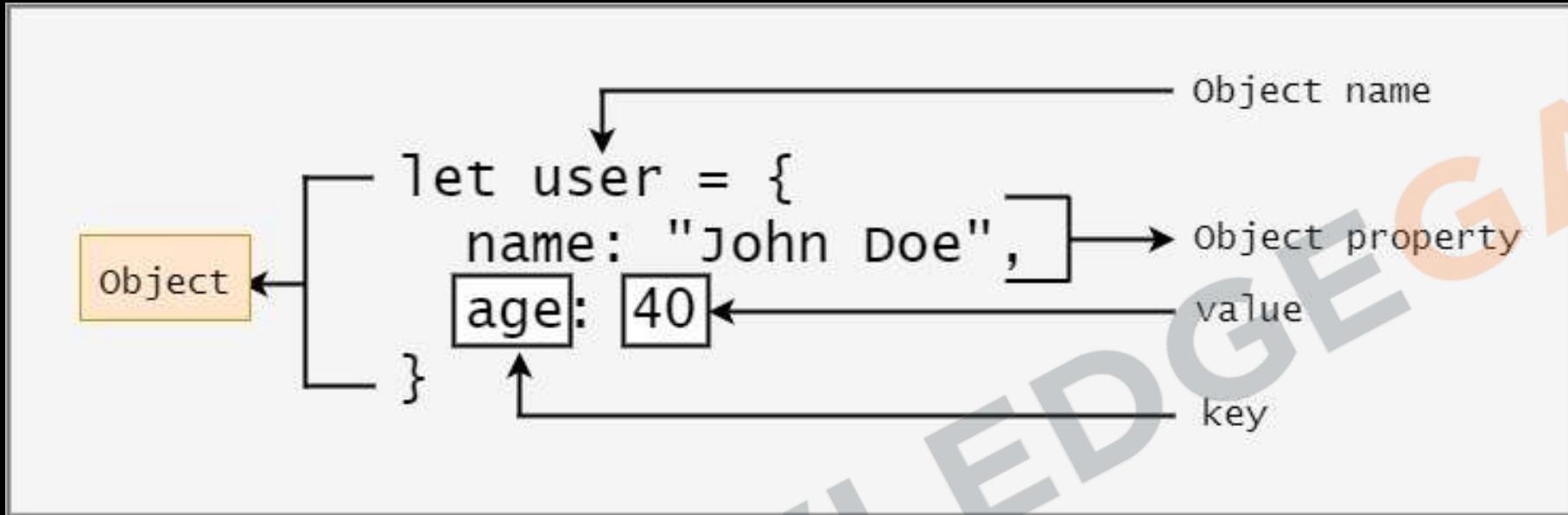
What is an Object?



```
let product = {  
    company: 'Mango',  
    item_name: 'Cotton striped t-shirt',  
    price: 861  
};
```

1. Groups multiple **values** together in **key-value** pairs.
2. **How to Define:** Use `{}` to enclose **properties**.
3. **Example:** `product {name, price}`
4. **Dot Notation:** Use `.` **operator** to access values.
5. **Key Benefit:** Organizes **related data** under a **single name**.

Object Syntax?



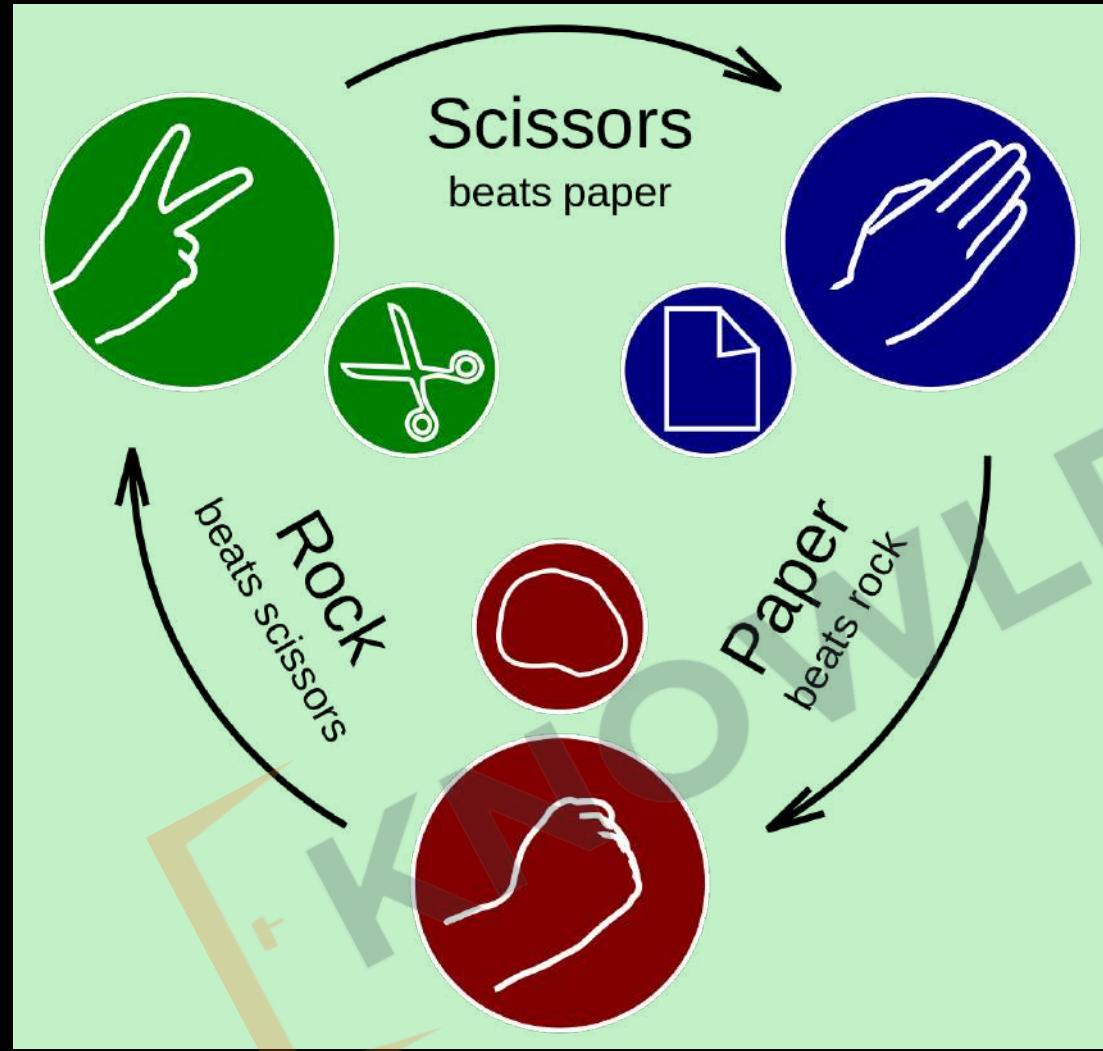
1. Basic Structure: Uses `{}` to enclose **data**.
2. Rules: **Property** and **value** separated by a **colon(:)**
3. Comma: Separates different **property-value** pairs.
4. Example: `{ name: "Laptop", price: 1000 }`

Accessing Objects



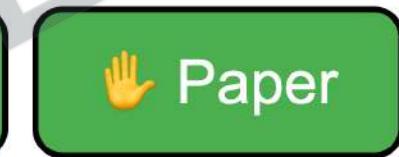
1. Dot Notation: Access **properties** using **. Operator** like `product.price`
2. Bracket Notation: Useful for **properties** with **special characters** `product["nick-name"]`. Variables can be used to access properties
3. `typeof` returns **object**.
4. Values can be **added** or **removed** to an object
5. Delete Values using `delete`

Project Rock-Paper-Scissor Game



Rock Paper Scissors Game

Click on one of the following to play the game:



Create object for
maintaining Score

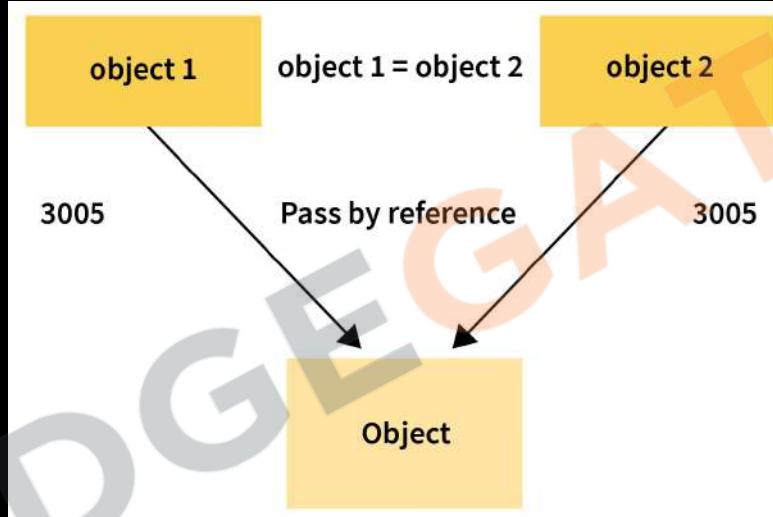
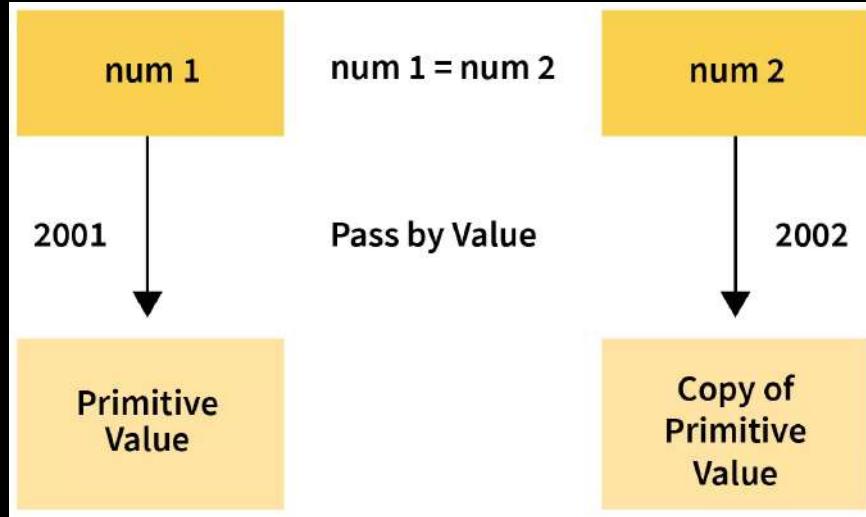
Inside Object

```
let product = {  
    company: 'Mango',  
    itemName: 'Cotton striped t-shirt',  
    price: 861,  
    rating: {  
        stars: 4.5,  
        noOfReviews: 87  
    },  
    displayPrice: function() {  
        return `$$ {this.price.toFixed(2)} `;  
    }  
};
```



1. Objects can contain Primitives like numbers and strings.
2. Objects can contain other objects and are called Nested Objects.
3. Functions can be object properties.
4. Functions inside an object are called methods.
5. Null Value: Intentionally leaving a property empty.

Object References



1. Objects work based on **references**, not actual data.
2. Copying an object copies the reference, not the actual object.
3. When comparing with `==`, you're comparing references, not content.
4. Changes to one reference affects all copies.

Call by Reference

```
// Function to swap two numbers using call by reference
function trySwap(obj) {
  let temp = obj.a;
  obj.a = obj.b;
  obj.b = temp;
  console.log(`Inside trySwap - a: ${obj.a}, b: ${obj.b}`);
}
```

```
function main() {
  let obj = {a: 10, b: 20};
  console.log(`Before trySwap - a: ${obj.a}, b: ${obj.b}`);
  trySwap(obj); // Swap x and y using reference
  // The original values are changed
  console.log(`After trySwap - a: ${obj.a}, b: ${obj.b}`);
}
```

```
main();
```

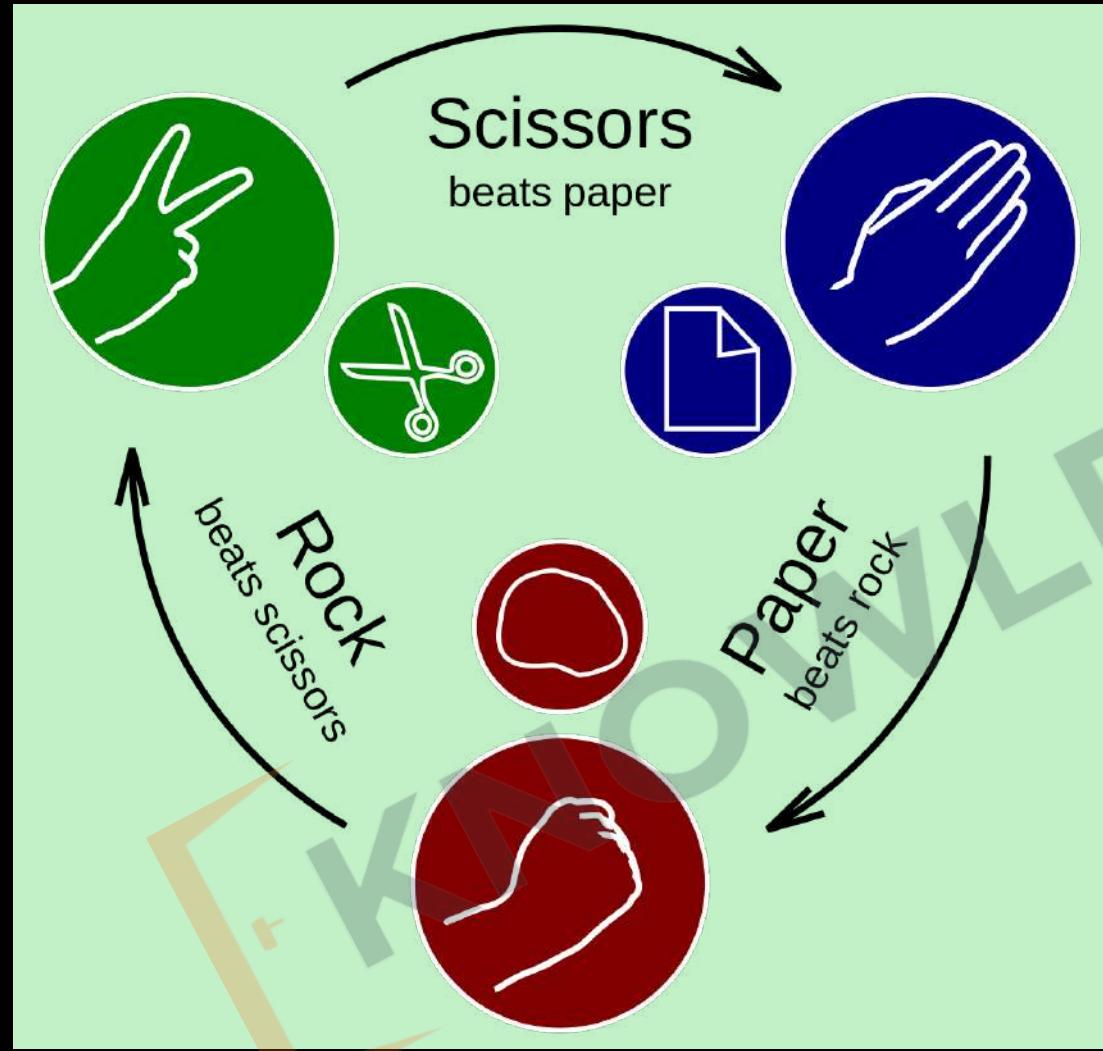
Before trySwap - a: 10, b: 20

Inside trySwap - a: 20, b: 10

After trySwap - a: 20, b: 10

1. **Direct Access:** In JavaScript, call by reference is achieved by **passing objects or arrays to functions**, allowing the function to directly modify the original object or array.
2. **Objects and Arrays:** Functions **receive a reference to the original object or array**, enabling them to alter the properties or elements within it.
3. **Memory Efficiency:** This method **avoids creating copies of the entire object or array**, thus saving memory and improving performance.
4. **Persistent Changes:** Any **modifications** made to the object or array within the function **persist outside the function**, reflecting the changes in the original object or array.

Project Rock-Paper-Scissor Game



Rock Paper Scissors Game

Click on one of the following to play the game:

✊ Rock

✋ Paper

✌ Scissors

Create method for showing result
into score Object.

Practice Exercise

Objects

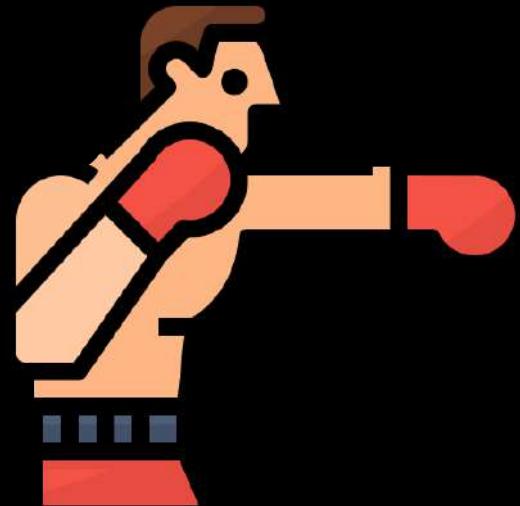
1. Create **object** to represent a **product** from **Myntra**
2. Create an **Object** with **two references** and log changes to one object by **changing** the other one.
3. Use bracket notation to display **delivery-time**.





Autoboxing

1. **Automatic Conversion:** JavaScript converts primitives to their corresponding object wrappers when methods or properties are accessed.
2. **Temporary Objects:** The conversion creates a temporary object that is discarded after the method or property access.
3. **Method Access:** Enables the use of methods like `.length` on strings, `.toFixed()` on numbers, and `.toString()` on booleans directly on primitive values.
4. Allows properties and methods to be used on primitives.
5. Example: Strings have properties and methods like `length`, `toUpperCase`, etc.



Autoboxing

```
// Primitive string
let primitiveString = "Hello, World!";
// JavaScript automatically converts the primitive string to a String object
let length = primitiveString.length;
console.log(`Length of string: ${length}`); // Output: Length of string: 13
```

```
// Primitive number
let primitiveNumber = 42;
// JavaScript automatically converts the primitive number to a Number object
let fixedNumber = primitiveNumber.toFixed(2);
console.log(`Fixed number: ${fixedNumber}`); // Output: Fixed number: 42.00
```

```
// Primitive boolean
let primitiveBoolean = true;
// JavaScript automatically converts the primitive boolean to a Boolean object
let booleanString = primitiveBoolean.toString();
console.log(`Boolean as string: ${booleanString}`); // Output: Boolean as string: true
```

Type Coercion

1. **Memory Implicit Conversion:** JavaScript automatically converts one data type to another when required.
2. **String Conversion:** Numbers and booleans can be implicitly converted to strings.
3. **Number Conversion:** Strings and booleans can be implicitly converted to numbers.
4. **Boolean Conversion:** Various data types can be converted to booleans (e.g., 0, "", null, undefined are false).

```
// String Conversion
let num = 10;
let str = "The number is " + num;
console.log(str); // Output: "The number is 10"
```

```
// Number Conversion
let strNum = "15";
let result = strNum - 5;
console.log(result); // Output: 10
```

```
// Boolean Conversion
let boolValue = true;
let numberValue = 1;
| // Output: true
console.log(boolValue == numberValue);
//true is converted to 1 before comparison
```

Type Coercion

1. Memory Addition (+)

- Number + Number: Adds the numbers.
- String + String: Concatenates the strings.
- String + Number: Converts the number to a string.
- Boolean + String: Converts the boolean to a string.

2. Subtraction (-), Multiplication (*), Division (/), Modulus (%)

- Number - Number: Subtracts the numbers.
- String - Number: Converts the string to a number.
- Boolean - Number: Converts the boolean to a number (true to 1, false to 0).

3. Equality (==)

- Number == String: Converts the string to a number.
- Boolean == Number: Converts the boolean to a number.
- Boolean == String: Converts the boolean to a number and the string to a number.

4. Strict Equality (===)

- No type coercion is performed. The types and values must be identical.

```
// Addition (+)
console.log(1 + 2);          // 3 (Number + Number)
console.log('Hi' + ' Sam'); // "Hi Sam" (String + String)
console.log(5 + '5');       // "55" (Number + String)
console.log(true + 'yes'); // "trueyes" (Boolean + String)

// Subtraction (-)
console.log(10 - 5); // 5 (Number - Number)
console.log('10' - 5); // 5 (String - Number, "10" converted to 10)
console.log(10 - '5'); // 5 (Number - String, "5" converted to 5)
console.log(true - 1); // 0 (Boolean - Number, true converted to 1)

// Multiplication (*)
console.log(2 * 3); // 6 (Number * Number)
console.log('2' * 3); // 6 (String * Number, "2" converted to 2)
console.log(2 * '3'); // 6 (Number * String, "3" converted to 3)
console.log(false * 10); // 0 (Boolean * Number, false converted to 0)

// Equality (==)
console.log(5 == '5'); // true (String converted to Number)
console.log(true == 1); // true (Boolean converted to Number)
console.log(false == 0); // true (Boolean converted to Number)

// Strict Equality ( === )
console.log(5 === '5'); // false (Different types)
console.log(true === 1); // false (Different types)
```

== VS ===

```
let num = 5;
let str = "5";

// true, because "5" == "5" after type coercion
console.log(num == str);
// false, because 5 (number) !== "5" (string)
console.log(num === str);
```

```
let booleanValue = true;
let numValue = 1;

// true, because true is converted to 1
console.log(booleanValue == numValue);
// false, because true (boolean) !== 1 (number)
console.log(booleanValue === numValue);
```

1. **==**: Performs **type coercion**, converting values to a **common type** before comparing.
2. **====**: Does **not** perform type coercion, compares **both value and type**.
3. **==**: Loosely checks for equality, can lead to **unexpected results**.
4. **====**: Strictly checks for equality, **more predictable and recommended**.

Memory Allocation

```
// number type  
let age = 25;  
  
// string type  
let name = 'John';
```

1. age Variable:

- **Memory Allocation:** Stack
- **Variable Memory:** Holds the reference (address) to the actual data.
- **Value Memory:** Typically, numbers occupy 8 bytes in memory.

2. name Variable:

- **Memory Allocation:** Stack
- **Variable Memory:** Holds the reference (address) to the actual data.
- **Value Memory:** Strings occupy memory based on their length. Each character typically uses 2 bytes (UTF-16 encoding).

```
// initially a number  
let data = 42;  
  
// reassign to a string  
data = 'Hello World';
```

1. Initial State (data as Number):

- **Memory Allocation:** Stack
- **Variable Memory:** Holds the reference to the number.
- **Value Memory:** 8 bytes for the number.

2. After Reassignment (data as String):

- **Memory Allocation:** Stack (reference), Heap (actual string data)
- **Variable Memory:** Holds the reference to the string.
- **Value Memory:** Length of "Hello World" (11 characters) x 2 bytes = 22 bytes on the heap.

Garbage Collection

1. Automatic memory management using garbage collection.
2. Reachability determines if an object is collected.
3. Performance optimization to reduce pauses.
4. Memory leaks can still occur due to lingering references.
5. Best practices include minimizing global variables and clearing unused references.

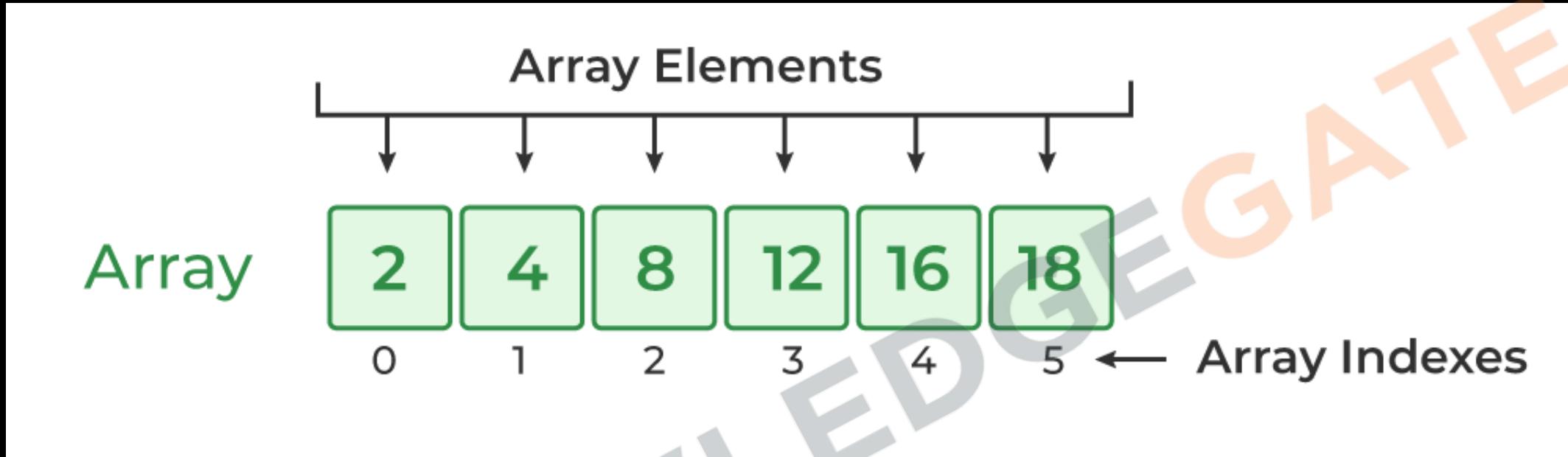
Common Algorithms:

- **Mark-and-Sweep:** The most common garbage collection algorithm. It marks all reachable objects and then sweeps or collects all unmarked objects, freeing up their memory.
- **Reference Counting:** Keeps track of the number of references to each object. An object is collected when its reference count drops to zero. However, it cannot handle circular references.





What is an **Array**?



1. An Array is just a **list** of values.
2. **Index:** Starts with **0**.
3. Arrays are used for **storing multiple values** in a single variable.

Array (Syntax & Values)

```
let myArray = [1, 'KG Coding', null, true,  
 {likes: '1 Million'}];
```

1. Use [] to create a new array, [] brackets enclose list of values
2. Arrays can be saved to a variable.
3. Accessing Values: Use [] with index.
4. Syntax Rules:
 - Brackets start and end the array.
 - Values separated by commas.
 - Can span multiple lines.
5. Arrays can hold any value, including arrays.
6. typeof operator on Array Returns Object.

Array

```
// Create an array of fruits
const fruits = ["Apple", "Banana", "Cherry"];

// Access elements in the array
console.log("First fruit:", fruits[0]); // Output: Apple
console.log("Second fruit:", fruits[1]); // Output: Banana
console.log("Third fruit:", fruits[2]); // Output: Cherry
```

Array

```
const fruits = ["Apple", "Banana", "Cherry"];
```

```
// Using while loop
```

```
console.log("Using while loop:");
```

```
let i = 0;
```

```
while (i < fruits.length) {
```

```
    console.log(fruits[i]);
```

```
    i++;
```

```
}
```

```
// Using a for loop
```

```
console.log("Using for loop:");
```

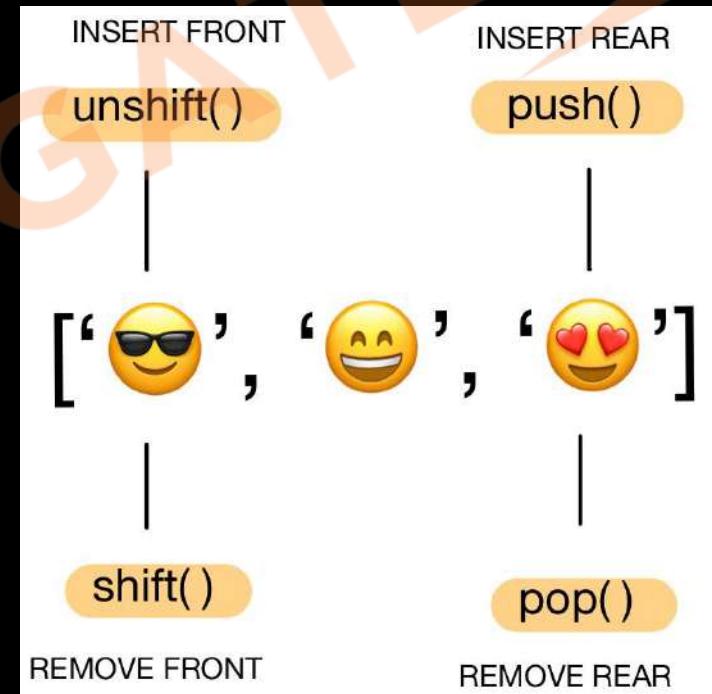
```
for (let i = 0; i < fruits.length; i++) {
```

```
    console.log(fruits[i]);
```

```
}
```

Array Methods

1. `Array.isArray()` checks if a variable is an array.
2. `Length` property holds the size of the array.
3. Common Methods:
 - `push/pop`: Add or remove to end.
 - `shift/unshift`: Add or remove from front.
 - `splice`: Add or remove elements.
 - `toString`: Convert to string.
 - `sort`: Sort elements.
 - `valueOf`: Get array itself.
4. Arrays also use `reference` like objects.
5. De-structuring also `works` for Arrays.



Array

```
const fruits = ["Apple", "Banana", "Cherry"];  
  
// Add elements to the end of the array using push  
fruits.push("Date");  
console.log("After push:", fruits); // Output: ["Apple", "Banana", "Cherry", "Date"]  
  
// Remove the last element using pop  
const lastFruit = fruits.pop();  
console.log("Popped fruit:", lastFruit); // Output: Date  
console.log("After pop:", fruits); // Output: ["Apple", "Banana", "Cherry"]  
  
// Add elements to the beginning using unshift  
fruits.unshift("Elderberry");  
console.log("After unshift:", fruits); // Output: ["Elderberry", "Apple", "Banana", "Cherry"]  
  
// Remove the first element using shift  
const firstFruit = fruits.shift();  
console.log("Shifted fruit:", firstFruit); // Output: Elderberry  
console.log("After shift:", fruits); // Output: ["Apple", "Banana", "Cherry"]
```

Array

```
const numbers = [1, 2, 3, 4, 5];

// Find the first element greater than 3
const firstGreaterThanThree = numbers.find((num) => num > 3);
console.log("First number greater than 3:", firstGreaterThanThree); // Output: 4

// Find the index of the number 3
const index = numbers.indexOf(3);
console.log("Index of 3:", index); // Output: 2
```

for-each Method

```
let foods = ['bread', 'rice', 'meat', 'pizza'];

foods.forEach(function(food) {
  console.log(food);
})
```

1. A method for **array iteration**, often preferred for readability.
2. Parameters: One for **item**, optional second for **index**.
3. Using **return** is similar to **continue** in traditional loops.
4. Not straightforward to **break** out of a **forEach** loop.
5. When you need to perform an action on each array element and don't need to break early.

Array

```
const fruits = ["Apple", "Banana", "Cherry"];  
  
// Using forEach method  
console.log("Using forEach:");  
fruits.forEach((fruit) => {  
  console.log(fruit);  
});
```

Array Advance Methods

```
[🐄,🥔,🐓,🥕].map(cook) => [🍔,🍟,🍗,🥤]  
[🍔,🍟,🍗,🥤].filter(isVegetarian) => [🍟,🥤]
```

1. Filter Method:

- Syntax: `array.filter((value, index) => return true/false)`
- Use: Filters elements based on condition.

2. Map Method:

- Syntax: `array.map((value) => return newValue)`
- Use: Transforms each element.

Array

```
const numbers = [1, 2, 3, 4, 5];

// Create a new array with each number doubled
const doubled = numbers.map((num) => num * 2);
console.log("Doubled:", doubled); // Output: [2, 4, 6, 8, 10]

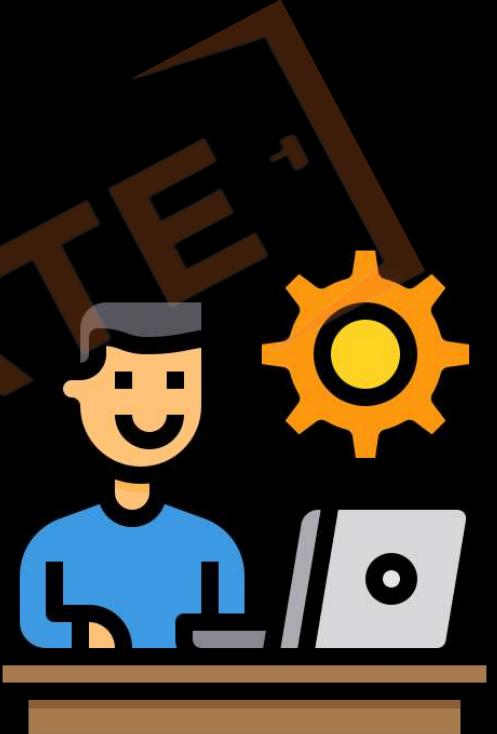
// Filter the array to include only even numbers
const evens = numbers.filter((num) => num % 2 === 0);
console.log("Evens:", evens); // Output: [2, 4]

// Calculate the sum of all numbers
const sum = numbers.reduce((accumulator, currentValue) => accumulator + currentValue, 0);
console.log("Sum:", sum); // Output: 15
```

Practice Exercise

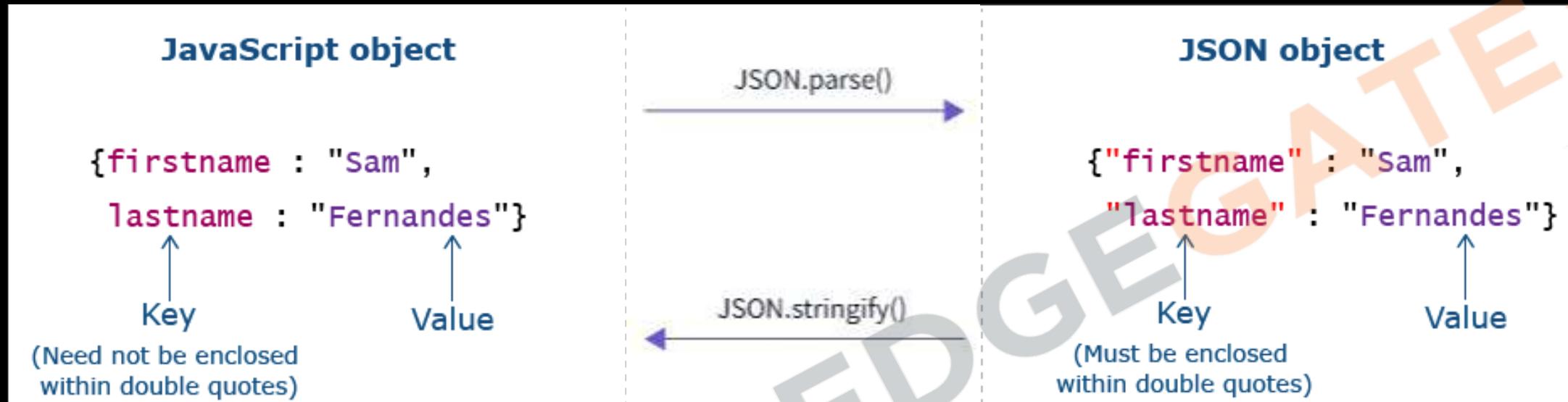
Arrays

1. Create an **array** of numbers [5,6]. Add 4 at the **beginning** and 7 at the **end** of the array.
2. Create a method to return an **element** at a particular **position** in the array.
3. Create an array copy using **slice** method.
4. Using accumulator pattern **concatenate** all the strings in the given array ['KG', 'Coding', 'Javascript', 'Course', 'is', 'Best']





What is JSON?



1. JavaScript Object Notation: Not the same as JS object, but similar.
2. Common in network calls and data storage.
3. `JSON.stringify()` and `JSON.parse()`
4. Strings are easy to transport over network.
5. JSON requires double quotes. Escaped as \".
6. JSON is data format, JS object is a data structure.

What is JSON?

```
// Define a JavaScript object
const person = {
  name: "Amit",
  age: 28,
  city: "New Delhi",
  skills: ["JavaScript", "Node.js", "React"]
};

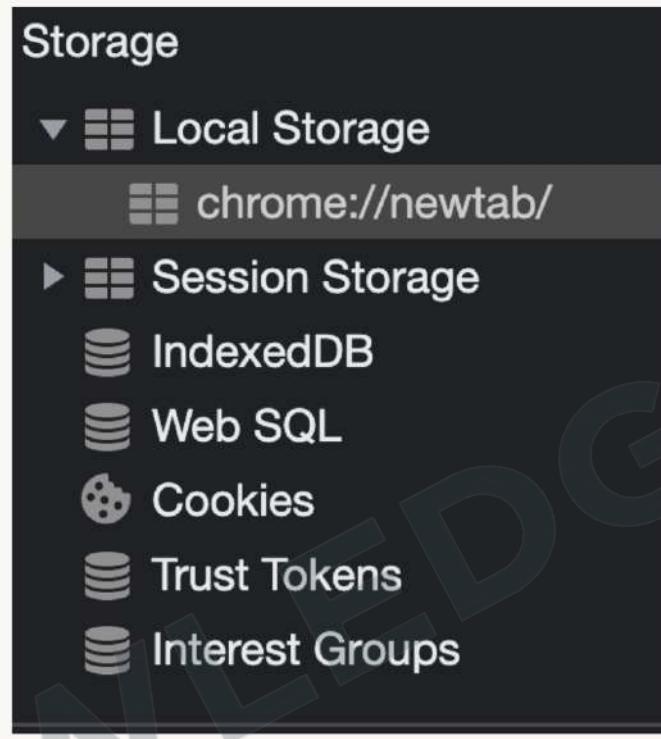
// Convert the JavaScript object to a JSON string
const jsonString = JSON.stringify(person);
console.log("JSON String:", jsonString);

// Convert the JSON string back to a JavaScript object
const jsonObject = JSON.parse(jsonString);
console.log("JavaScript Object:", jsonObject);

// Access properties from the parsed JavaScript object
console.log("Name:", jsonObject.name);
console.log("Age:", jsonObject.age);
console.log("City:", jsonObject.city);
console.log("Skills:", jsonObject.skills.join(", "))
```

Browser Local Storage

```
// store an object in Local Storage  
localStorage.setItem(  
  "user",  
  JSON.stringify({  
    name: "Gopi Gorantala"  
    age: 32  
});  
  
// retrieve an object in Local Storage  
const user = JSON.parse(  
  localStorage.getItem("user")  
);
```



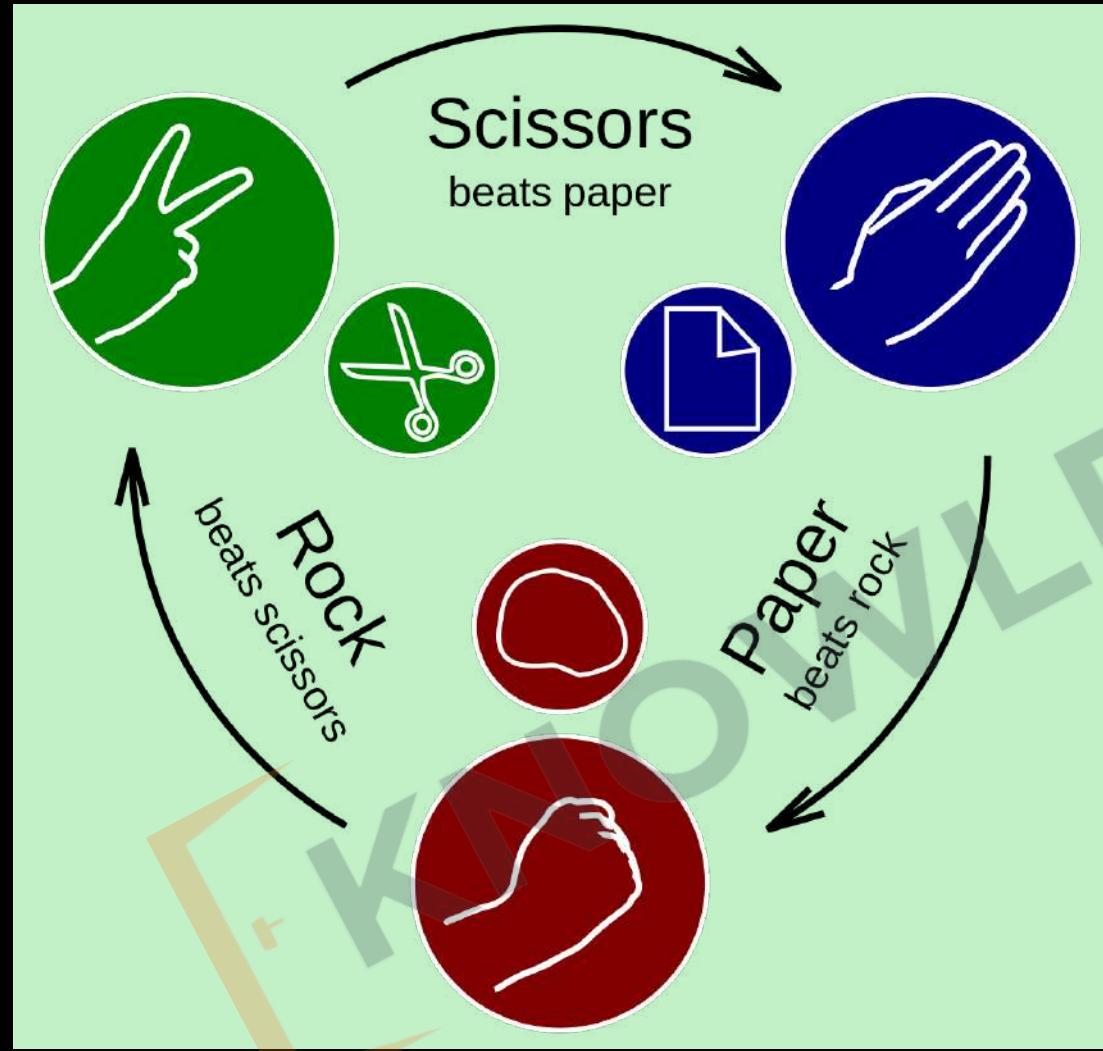
1. Persistent data storage in the browser.
2. `setItem`: Stores data as key-value pairs.
3. Only strings can be stored.
4. `getItem`: Retrieves data based on key.
5. Other Methods: `localStorage.clear()`, `removeItem()`.
6. Do not store sensitive information. Viewable in storage console.

Browser Local Storage

```
// Store data in localStorage using setItem  
localStorage.setItem("name", "Amit");  
localStorage.setItem("age", "28");  
localStorage.setItem("city", "New Delhi");  
  
// Retrieve data from localStorage using getItem  
const name = localStorage.getItem("name");  
const age = localStorage.getItem("age");  
const city = localStorage.getItem("city");  
  
console.log("Name:", name); // Output: Amit  
console.log("Age:", age); // Output: 28  
console.log("City:", city); // Output: New Delhi
```

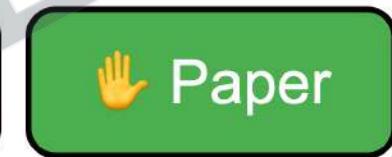
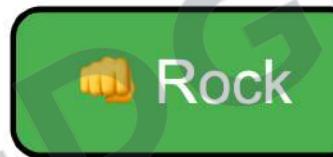
```
// Remove a specific item from localStorage  
localStorage.removeItem("age");  
  
// Attempt to retrieve the removed item  
const removedAge = localStorage.getItem("age");  
console.log("Removed Age:", removedAge); // Output: null  
  
// Clear all items from localStorage  
localStorage.clear();  
  
// Attempt to retrieve data after clearing localStorage  
const clearedName = localStorage.getItem("name");  
console.log("Cleared Name:", clearedName); // Output: null
```

Project Rock-Paper-Scissor Game



Rock Paper Scissors Game

Click on one of the following to play the game:



1. Score will survive browser refresh.
2. Add Reset Button To clear or reset stored data.

Date



1. new Date() Creates a new Date object with the current date and time.
2. Key Methods:
 - `getTime()`: Milliseconds since Epoch.
 - `getFullYear()`: 4-digit year
 - `getDay()`: Day of the week
 - `getMinutes()`: Current minute
 - `getHours()`: Current hour.
3. Crucial for timestamps, scheduling, etc.

Date

```
// Create a new Date object for the current date and time
const currentDate = new Date();

// Get the current timestamp in milliseconds since January 1, 1970
console.log("Current Time (ms since 1970):", currentDate.getTime());

// Get the current day of the week (0 = Sunday, ..., 6 = Saturday)
console.log("Day of the Week:", currentDate.getDay());

// Get the current year
console.log("Current Year:", currentDate.getFullYear());

// Get the current month (0 = January, ..., 11 = December)
console.log("Current Month:", currentDate.getMonth());

// Get the current date of the month
console.log("Date of the Month:", currentDate.getDate());

// Create a specific date (e.g., December 25, 2024)
const specificDate = new Date("2024-12-25");

// Log the specific date
console.log("Specific Date:", specificDate.toDateString());
```

DOM Properties & Methods

DOM and Element Properties

1. location
2. title
3. href
4. domain
5. innerHTML
6. innerText
7. classList

DOM and Element Methods

1. getElementById()
2. querySelector()
3. classList: add(), remove()
4. createElement()
5. appendChild()
6. removeChild()
7. replaceChild()

Practice Exercise

JSON ,Local Storage, Date & DOM

1. Display good morning, afternoon and night based on current hour.
2. Add the name to the output too.
3. Create a Button which shows the number how many times it has been pressed.
 - Also, it has different colors for when it has been pressed odd or even times.
 - The click count should also survive browser refresh.





Objects Equality (`==` & `===`)

```
const obj1 = { a: 1 };
const obj2 = { a: 1 };
const obj3 = obj1;
```

```
console.log(obj1 == obj2); // Output: false (different instances)
console.log(obj1 == obj3); // Output: true (same instance)
```

```
console.log(obj1 === obj2); // Output: false (different instances)
console.log(obj1 === obj3); // Output: true (same instance)
```

1. `==`: When comparing objects, `==` checks if the two operands refer to the same object in memory. It does not compare the contents of the objects. Therefore, even if two objects have the same properties and values, they will be considered unequal unless they refer to the same instance.
2. `===`: Like `==`, the `===` operator checks if the operands refer to the same object in memory. It does not consider the object's properties or values.

Objects Equality (Using JSON)

```
function jsonEqual(obj1, obj2) {  
  return JSON.stringify(obj1) === JSON.stringify(obj2);  
}  
  
const obj1 = { a: 1, b: 2 };  
const obj2 = { a: 1, b: 2 };  
  
console.log(jsonEqual(obj1, obj2)); // Output: true
```

For simple objects without circular references, functions, or undefined values, you can use JSON serialization as a quick comparison

Objects Equality (Shallow Comparison)

```
function shallowEqual(obj1, obj2) {  
  // Check if both are objects and not null  
  if (typeof obj1 !== 'object' || obj1 === null ||  
      typeof obj2 !== 'object' || obj2 === null) {  
    return false;  
  }  
  
  // Compare the number of properties  
  const keys1 = Object.keys(obj1);  
  const keys2 = Object.keys(obj2);  
  if (keys1.length !== keys2.length) {  
    return false;  
  }  
  
  // Compare each property value  
  for (let key of keys1) {  
    if (obj1[key] !== obj2[key]) {  
      return false;  
    }  
  }  
  
  return true;  
}
```

For a shallow comparison, you can check if two objects have the same set of properties with identical values. This approach doesn't compare nested objects.

```
const objA = { a: 1, b: 2 };  
const objB = { a: 1, b: 2 };  
const objC = { a: 1, b: 3 };  
  
console.log(shallowEqual(objA, objB)); // Output: true  
console.log(shallowEqual(objA, objC)); // Output: false
```

Objects Equality (Deep Comparison)

```
function deepEqual(obj1, obj2) {  
  if (obj1 === obj2) {  
    return true; // Same reference or both null  
  }  
  
  if (typeof obj1 !== 'object' || obj1 === null ||  
  | typeof obj2 !== 'object' || obj2 === null) {  
    return false;  
  }  
  
  const keys1 = Object.keys(obj1);  
  const keys2 = Object.keys(obj2);  
  
  if (keys1.length !== keys2.length) {  
    return false; // Different number of properties  
  }  
  
  for (let key of keys1) {  
    if (!keys2.includes(key) ||  
    | !deepEqual(obj1[key], obj2[key])) {  
      return false;  
    }  
  }  
  return true;  
}
```

For a deep comparison, you need to recursively compare properties that might themselves be objects.

```
const objD = { a: 1, b: { c: 3 } };  
const objE = { a: 1, b: { c: 3 } };  
const objF = { a: 1, b: { c: 4 } };  
  
console.log(deepEqual(objD, objE)); // Output: true  
console.log(deepEqual(objD, objF)); // Output: false
```

Object Copy (Shallow)

```
function shallowCopyObject(original) {  
    const copy = {};  
    for (let key in original) {  
        if (original.hasOwnProperty(key)) {  
            copy[key] = original[key];  
        }  
    }  
    return copy;  
}
```

Creates a shallow copy by spreading the properties of the original object into a new object.

```
const original = { a: 1, b: 2, c: 3 };  
const copy = newFunction(original);
```

Object Copy (Deep using JSON)

```
// Deep Copy using JSON Serialization
const original = { a: 1, b: { c: 2 } };
const deepCopy = JSON.parse(JSON.stringify(original));

console.log(deepCopy); // Output: { a: 1, b: { c: 2 } }
```

Object Copy (Deep using Recursive Copy)

```
// Deep Copy using Recursion
function deepCopy(obj) {
  if (obj === null || typeof obj !== 'object') {
    return obj;
  }
  const copy = Array.isArray(obj) ? [] : {};
  for (let key in obj) {
    if (obj.hasOwnProperty(key)) {
      copy[key] = deepCopy(obj[key]);
    }
  }
  return copy;
}

const original = { a: 1, b: { c: 2 } };
const deepCopyObj = deepCopy(original);

console.log(deepCopyObj); // Output: { a: 1, b: { c: 2 } }
```

Practice Exercise

Objects

1. Add **function** `isIdenticalProduct` to compare two product objects.
2. Write a function that takes **two objects** as input and returns true if they have identical contents using **JSON serialization**. Test your function with objects containing different data types and explain any limitations.
3. Create an **object** with nested objects. Write a function that performs a **shallow copy** of the object. **Modify a property in the nested object of the copy** and observe its effect on the original object.
4. Write a function that **merges two objects**. If the same property exists in both objects, use the value from the second object. Test this function with objects that have both overlapping and unique properties.





De-structuring

```
let product = {  
    company: 'Mango',  
    itemName: 'Cotton striped t-shirt',  
    price: 861  
};  
  
// Destructuring  
let company = product.company  
// is same as  
let { company } = product;
```

```
// Property shorthand  
let price = 861;  
let product = {  
    company: 'Mango',  
    itemName: 'Cotton striped t-shirt',  
    price: price  
};  
  
// is same as  
let product1 = {  
    company: 'Mango',  
    itemName: 'Cotton striped t-shirt',  
    price  
};
```

```
// Method shorthand  
let product = {  
    company: 'Mango',  
    itemName: 'Cotton striped t-shirt',  
    displayPrice: function() {  
        return `$$\{this.price.toFixed(2)\}`  
    }  
};  
  
// is same as  
let product1 = {  
    company: 'Mango',  
    itemName: 'Cotton striped t-shirt',  
    displayPrice() {  
        return `$$\{this.price.toFixed(2)\}`  
    }  
};
```

1. De-structuring: Extract properties from objects easily.
2. We can extract more than one property at once.
3. Shorthand Property: {message: message} simplifies to just message.
4. Shorthand Method: Define methods directly inside the object without the function keyword.

Spread & Rest Operator

(Spread)

```
1 // Array Expansion
2 const arr1 = [1, 2, 3];
3 const arr2 = [...arr1]; // [1, 2, 3]
4 // [1, 2, 3, 4, 5]
5 const arr3 = [...arr1, 4, 5];
6
7 // Object Expansion
8 const obj1 = { a: 1, b: 2 };
9 // { a: 1, b: 2, c: 3 }
10 const obj2 = { ...obj1, c: 3 };
11
12 // Function Arguments
13 function sum(a, b, c) {
14     return a + b + c;
15 }
16 const numbers = [1, 2, 3];
17 console.log(sum(...numbers)); // 6
```

1. Represented by three dots (...), the spread operator is used to expand elements of an iterable (like an array or string) into individual elements.
2. It is used to make shallow copy of arrays or objects.
3. Useful for copying arrays and objects without modifying the original.
4. Ensures immutability in functions where modification of inputs is not desired.

Spread & Rest Operator

(Rest)

```
1 // Function Parameters
2 function sum(...numbers) {
3   return numbers.reduce((acc, curr) => acc + curr, 0);
4 }
5 console.log(sum(1, 2, 3, 4)); // 10
6
7 // Array Destructuring
8 const [first, second, ...rest] = [1, 2, 3, 4, 5];
9 console.log(rest); // [3, 4, 5]
10
11 // Object destructuring
12 const { a, b, ...rest } = { a: 1, b: 2, c: 3, d: 4 };
13 console.log(rest); // { c: 3, d: 4 }
```

- Used to collect the remaining elements of an array after extracting some elements.
- Used to collect the remaining properties of an object after extracting some properties.

1. Represented by three dots (...), the rest operator is used to collect multiple elements into a single array or object.
2. Allows a function to accept an indefinite number of arguments as an array.

Practice Exercise

Objects

- Given an object `{message: 'good job', status: 'complete'}`, use de-structuring to create two variables `message` and `status`.





Callbacks

```
// Define a callback function
function greeting(name) {
  console.log('Hello, ' + name);
}

// Define a function that takes a callback
function processUserInput(callback) {
  var name = prompt('Please enter your name.');
  callback(name);
}

// Call the function with the callback
processUserInput(greeting);
```

1. A **callback** is a **function** passed as an **argument** to another **function**, **which** is then **invoked inside the outer function** to complete some kind of routine or action.
2. Usage: Callbacks are commonly used in asynchronous programming to **execute code after an asynchronous operation has completed**.

Anonymous Functions as Values

```
// Assigned to a variable
const add = function(a, b) {
  return a + b;
}
console.log(add(2, 3)); // Outputs: 5
```

```
// Example as a callback
setTimeout(function() {
  console.log("This is anonymous");
}, 1000);
```

1. Anonymous functions are functions without a name.
2. They are often used as arguments to other functions or assigned to a variable.
3. Useful for creating function scopes and avoiding global variables.

Arrow Functions

```
let sum = function(num1, num2) {  
    return num1 + num2;  
}  
  
let Sum1 = (num1, num2) => {  
    return num1 + num2;  
}  
  
let Sum2 = (num1, num2) => num1 + num2;  
  
let square = num => num * num;
```

1. A concise way to write **anonymous** functions.
2. For Single Argument: **Round brackets** optional.
3. For Single Line: **Curly brackets** and return optional.
4. Often used when passing **functions as arguments**.

Arrow Functions

(Anonymous & Arrow Callbacks)

```
const numbers = [1, 2, 3, 4];

// Callback function without arrow function
function double(num) {
  return num * 2;
}

// Using map with a regular function
const doubled = numbers.map(double);

console.log('Doubled with regular:', doubled);
// Output: [2, 4, 6, 8]

// Using map with an arrow function
const doubledArrow = numbers.map((num) => num * 2);

console.log('Doubled with arrow:', doubledArrow);
// Output: [2, 4, 6, 8]
```

1. Instead of naming the callback function, you can define it directly within the argument list.
2. ES6 arrow functions can also be used as callbacks for a more concise syntax.

Higher-Order Functions

(Function as Argument)

```
const numbers = [1, 2, 3, 4, 5];  
  
// Using map, a higher-order function  
const doubled = numbers.map(num => num * 2);  
console.log(doubled); // [2, 4, 6, 8, 10]
```

1. Functions that can take other functions as arguments or return functions as their result.
2. Higher-order functions can accept functions as parameters, allowing you to pass behavior as data.
3. Example: `Array.prototype.map()`, `Array.prototype.filter()`, and `Array.prototype.reduce()` are higher-order functions.

Higher-Order Functions

(Return Functions)

```
function createAdder(x) {  
  return function(y) {  
    return x + y;  
  };  
}
```

```
const addFive = createAdder(5);  
console.log(addFive(10)); // 15
```

1. Higher-order functions can **return new functions**, enabling the creation of **function factories** or function composition.
2. Higher-order functions allow you to **encapsulate behavior** and create **abstractions for common patterns**, making your code more reusable and modular.

Closures (Lexical Scoping)

```
var age = 21;

function init() {
  var name = "Mozilla";

  function displayName() {
    console.log(name);
    console.log(age);
  }

  displayName();
}

init();
```

The diagram illustrates the lexical environment and scope in the provided JavaScript code. The entire code block is enclosed in a red box labeled 'Lexical Environment'. Inside, a green box labeled 'Lexical Scope' encloses the inner function 'displayName'. A blue bracket labeled 'Lexical Scope' points to the inner function. The variable 'name' is shown with a red line pointing to its declaration in the 'init' scope, and 'age' is shown with a red line pointing to its declaration in the 'init' scope.

1. JavaScript uses **lexical scoping**, which means that the scope of a variable is determined by its position within the source code.
2. Functions can access variables from their own scope, the scope of the parent function, and the global scope.

Closures

(Closure Creation)

```
function outerFunction() {  
    const outerVariable = 'I am outside!';  
    function innerFunction() {  
        console.log(outerVariable); // Accesses outerVariable from outerFunction's scope  
    }  
    return innerFunction;  
}  
  
const closureFunction = outerFunction();  
closureFunction(); // Output: "I am outside!"
```

1. A closure is created when a function is defined inside another function, and the inner function captures variables from the outer function.
2. The inner function retains access to these variables even after the outer function has finished executing.

Closures

(Maintaining State)

```
function makeCounter() {  
  let count = 0; // Private variable  
  
  return function () {  
    count += 1;  
    return count;  
  };  
}
```

```
const counter = makeCounter();  
console.log(counter()); // Output: 1  
console.log(counter()); // Output: 2  
console.log(counter()); // Output: 3
```

1. `makeCounter` returns a function that increments the `count` variable.
2. The `count` variable is **preserved between calls to `counter`** because the inner function forms a closure with it.



setTimeout & setInterval



1. Functions for executing code **asynchronously** after a delay.
2. `setTimeout` runs **once**; `setInterval` runs **repeatedly**
3. **setTimeout:**
 - Syntax: `setTimeout(function, time)`
 - Cancel: `clearTimeout(timerID)`
4. **setInterval:**
 - Syntax: `setInterval(function, time)`
 - Cancel: `clearInterval(intervalID)`

setTimeout & setInterval

```
// Example of setTimeout  
function greet() {  
  console.log("Hello, World!");  
}  
  
// Set a timeout to execute the greet function after  
// 2 seconds (2000 milliseconds)  
setTimeout(greet, 2000);  
console.log("This message will display first.");
```

```
// Example of setInterval  
function printTime() {  
  const now = new Date();  
  console.log(`Current time: ${now.toLocaleTimeString()}`);  
}  
  
// Set an interval to execute the printTime function  
// every second (1000 milliseconds)  
const intervalId = setInterval(printTime, 1000);  
  
// Stop the interval after 5 seconds  
setTimeout(() => {  
  clearInterval(intervalId);  
  console.log("Stopped printing the time.");  
}, 5000);
```

Event Handling

```
// HTML: <button id="myButton">Click Me</button>
```

```
// JavaScript
```

```
const button = document.getElementById('myButton');
```

```
function handleClick() {  
  alert('Button clicked!');  
}
```

```
// Attach event handler  
button.onclick = handleClick;
```

1. It involves responding to **events** triggered by **user actions** such as **clicks**, **key presses**, or **mouse movements**.
2. **Handlers** are functions that are **executed in response to an event**.

Event Listeners

```
// JavaScript
const button = document.getElementById('myButton');

function handleClick() {
  alert('Button clicked!');
}

// Attach event listener
button.addEventListener('click', handleClick);

// Remove event listener
button.removeEventListener('click', handleClick);
```

1. Event listeners are a **more flexible way to handle events**, allowing multiple handlers for the same event and easier removal of handlers.
2. The `addEventListener` method is used to **attach** event listeners to elements.

Event Propagation (Bubbling)

```
// HTML:  
// <div id="outerDiv">  
//   <button id="innerButton">Click Me</button>  
// </div>  
  
// JavaScript  
const outerDiv = document.getElementById('outerDiv');  
const innerButton = document.getElementById('innerButton');  
  
outerDiv.addEventListener('click', () => {  
  console.log('Outer DIV clicked (bubbling).');  
});  
  
innerButton.addEventListener('click', () => {  
  console.log('Inner Button clicked.');//  
});  
  
// Clicking the button will log:  
// "Inner Button clicked."  
// "Outer DIV clicked (bubbling)."
```

1. Event propagation determines the order in which event handlers are executed when an event occurs on an element nested within other elements.
2. The event starts from the target element and bubbles up to the outer elements.
3. By default, events propagate in the bubbling phase.

Event Propagation (Stopping Propagation)

```
// JavaScript
innerButton.addEventListener('click', (event) => {
  console.log('Inner Button clicked.');
  event.stopPropagation(); // Stop the event from bubbling up
});

outerDiv.addEventListener('click', () => {
  console.log('Outer DIV clicked.');
});

// Clicking the button will log only:
// "Inner Button clicked."
```

You can **stop the propagation of an event** using the `stopPropagation` method.

Event Propagation (Capturing)

```
// JavaScript
const outerDiv = document.getElementById('outerDiv');
const innerButton = document.getElementById('innerButton');

outerDiv.addEventListener(
  'click',
  () => {
    console.log('Outer DIV clicked (capturing).');
  },
  true // Enable capturing phase
);

innerButton.addEventListener('click', () => {
  console.log('Inner Button clicked.');
});

// Clicking the button will log:
// "Outer DIV clicked (capturing)."
// "Inner Button clicked."
```

1. The event starts from the **outermost** element and **captures** down to the target element.
2. Use the third parameter of `addEventListener` to specify capturing mode.
3. In this phase, the event starts from the window object and propagates down through the DOM tree to the target element.
4. Capturing phase is useful when you want to handle events at a higher level in the DOM hierarchy before they reach their target.

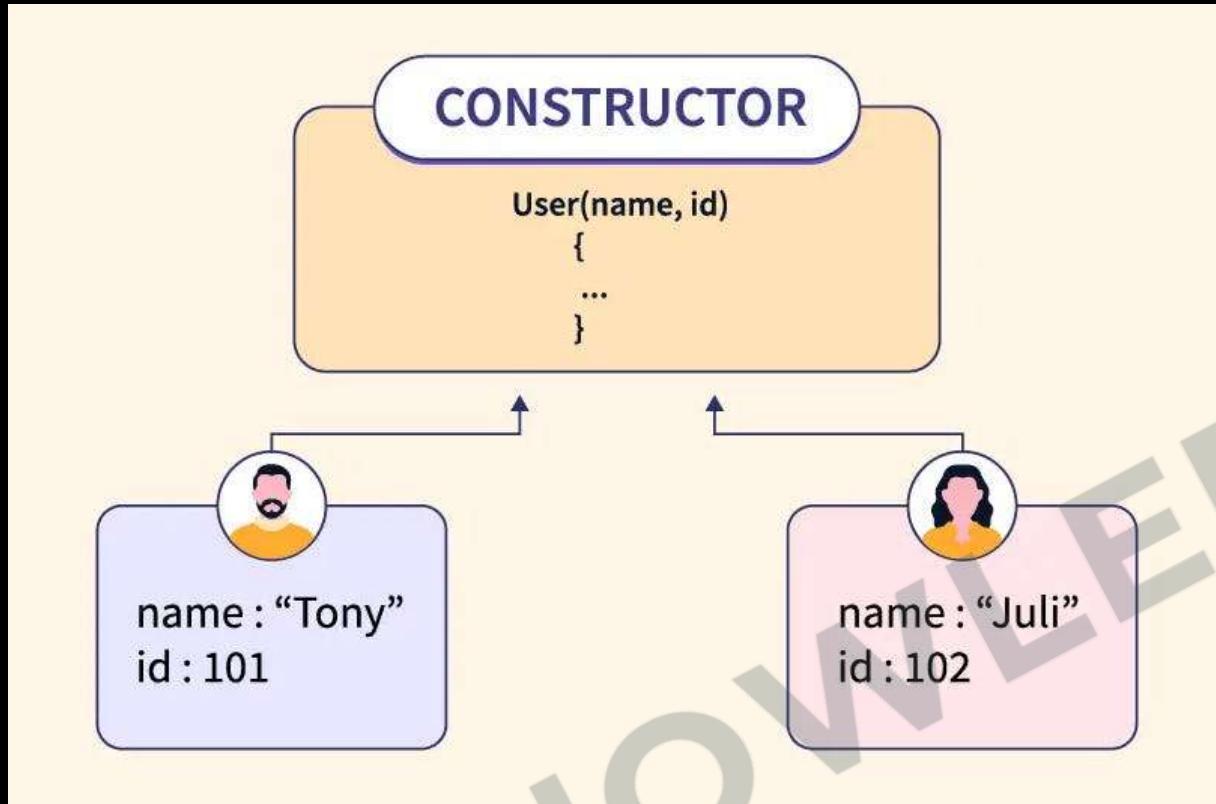


Constructor vs Object



Constructor is a blueprint; Objects are real values in memory.

Constructors



1. **Constructors** are special functions in **JavaScript** used to create and initialize objects.
2. They serve as **blueprints** for creating **instances** of specific types, making **code organized and reusable**.

Without Constructors

```
// User object with properties
const user1 = {
  name: "Prashant Jain",
  age: 32,
  email: "prashant@example.com",
  isAdmin: false,
};

const user2 = {
  name: "Shiv",
  age: 19,
  email: "shiv@example.com",
  isAdmin: false,
};

// Function to update user's admin status
function makeUserAdmin(user) {
  user.isAdmin = true;
}

makeUserAdmin(user1); // Make user an admin
```

1. Object literals are used to define objects, leading to **code duplication and inefficiency**.
2. Ensuring consistency in object structure requires manual validation, **increasing the risk of errors**.
3. Functions like **makeUserAdmin** are separate from objects, leading to **less organized code**.

Using Constructors

```
// User constructor function
function User(name, age, email, isAdmin) {
  this.name = name;
  this.age = age;
  this.email = email;
  this.isAdmin = isAdmin;

  // Method to update user's admin status
  this.makeUserAdmin = function() {
    this.isAdmin = true;
  };

// Create user1 object using the User constructor
const user1 = new User("Prashant Jain", 32,
  "prashant@example.com", false);

// Create user2 object using the User constructor
const user2 = new User("Shiv", 19,
  "shiv@example.com", false);

// Make user1 an admin
user1.makeUserAdmin();
```

1. By convention, constructor functions are named with an initial capital letter to distinguish them from regular functions.
2. new Keyword: When a function is called with new, it becomes a constructor that creates a new object instance. Inside the constructor, this refers to the new object.
3. Constructors automatically return the new object unless they explicitly return a different object.
4. Reusability: Constructors like Task provide a consistent structure for creating objects, reducing duplication and enhancing maintainability.
5. Consistency: Constructors ensure all objects have the same properties and methods, enforcing a common structure.
6. Encapsulation: Methods are encapsulated within objects, improving code organization and readability.

Constructor Prototype

Properties and methods can be added to a constructor's prototype to be shared across all instances created by that constructor.

```
// User constructor function
function User(name, age, email, isAdmin) {
  this.name = name;
  this.age = age;
  this.email = email;
  this.isAdmin = isAdmin;
}

// Method to update user's admin status using prototype
User.prototype.makeUserAdmin = function() {
  this.isAdmin = true;
};

// Create user1 object using the User constructor
const user1 = new User("Prashant Jain", 32,
  "prashant@example.com", false);

// Create user2 object using the User constructor
const user2 = new User("Shiv", 19,
  "shiv@example.com", false);

// Make user1 an admin
user1.makeUserAdmin();
```

Class (ES6 Convention)

```
// User constructor function
class User {
  constructor(name, age, email, isAdmin) {
    this.name = name;
    this.age = age;
    this.email = email;
    this.isAdmin = isAdmin;

    // Method to update user's admin status
    this.makeUserAdmin = function () {
      this.isAdmin = true;
    };
  }
}

// Create user1 object using the User constructor
const user1 = new User("Prashant Jain", 32,
  "prashant@example.com", false);

// Create user2 object using the User constructor
const user2 = new User("Shiv", 19,
  "shiv@example.com", false);

// Make user1 an admin
user1.makeUserAdmin();
```

- In ES6, the `class` keyword was introduced as syntactic sugar over `constructor functions` to define classes more succinctly.
- The `instanceof` operator can be used to check if an object is an instance of a particular constructor.

```
console.log(person instanceof Person); // Output: true
```

Class (ES6 Convention)

```
// Define a class
class Animal {
  // Constructor method
  constructor(name) {
    this.name = name;
  }

  // Method
  speak() {
    console.log(` ${this.name} makes a noise.`);
  }
}

// Create an instance of the class
const animal = new Animal('Dog');
animal.speak(); // Output: "Dog makes a noise."
```

- Classes in JavaScript are a template for creating objects. They encapsulate data with code to work on that data.
- Introduced in ECMAScript 2015 (ES6), classes provide a clearer syntax to create and manage objects and handle inheritance.

this Keyword

	STRICT MODE	NON-STRICK MODE
NODE.js	<code>console.log(this)</code> <code>function printthis() { console.log(this) }</code>	<code>{} <code>console.log(this)</code> <code>function printthis() { console.log(this) }</code></code>
BROWSER	<code>console.log(this)</code> <code>function printthis() { console.log(this) }</code>	<code>Window Object <code>console.log(this)</code> <code>function printthis() { console.log(this) }</code></code>

1. **Context-Sensitive:** The value of `this` depends on **how a function is called, not where it is defined.**
2. **Global Context:** In the **global execution context** (outside any function), `this` refers to the global object (`window` in browsers).

this Keyword

(Object Method)

```
const obj = {  
    name: "KGCoding",  
    greet: function() {  
        console.log(this.name);  
    }  
};  
obj.greet(); // Output: KGCoding
```

Object Method: When a function is called as a method of an object, this refers to the object the method is called on.

this Keyword

(Constructor Function)

```
function Person(name) {  
  this.name = name;  
}  
  
const person = new Person("KGCoding");  
console.log(person.name); // Output: KGCoding
```

Constructor Function: When a function is used as a constructor with `new`, this refers to the newly created instance.

this Keyword

(Event Handler)

```
<button id="myButton">Click me</button>
<script>
  document.getElementById("myButton").addEventListener("click", function() {
    console.log(this.id); // Output: myButton
  });
</script>
```

Event Handler: In an event handler, **this** refers to the element that received the event.

this Keyword

(Arrow Functions)

```
const obj = {  
    name: "KGCoding",  
    greet: () => {  
        console.log(this.name);  
    }  
};  
obj.greet(); // Output: undefined (inherited from global context)
```

Arrow Functions: Arrow functions do not have their own this.

They inherit this from the enclosing object.

Class (Inheritance)

```
// Define a class that extends another class
class Dog extends Animal {
  constructor(name, breed) {
    super(name); // Call the parent class constructor
    this.breed = breed;
  }

  // Method overriding
  speak() {
    console.log(`"${this.name}, the ${this.breed}, barks.`);
  }
}

// Create an instance of the subclass
const myDog = new Dog('Buddy', 'Golden Retriever');
myDog.speak(); // Output: "Buddy, the Golden Retriever, barks."
```

- Inheritance allows one class to inherit properties and methods from another class. This is achieved using the `extends` keyword.
- The `super` keyword is used to call the constructor of the parent class and to access its methods.

Class (Inheritance)

```
class MathUtils {  
    // Static method  
    static add(a, b) {  
        return a + b;  
    }  
}  
  
// Call the static method  
console.log(MathUtils.add(5, 3)); // Output: 8
```

- Static methods are defined on the class itself, rather than on instances of the class. They can be used to create utility functions related to the class.
- Static methods are called on the class directly, without needing to create an instance.



Error Handling

(Try-Catch Statements)

```
try {  
  const data = JSON.parse('Invalid JSON');  
} catch (error) {  
  console.error('Failed to parse JSON:', error.message);  
}
```

- The try...catch block is used to handle exceptions in synchronous code.
- The try block contains code that might throw an error, while the catch block handles the error.

Error Handling

(Error Objects)

```
const error = new Error('Something went wrong');
console.log(error.name); // "Error"
console.log(error.message); // "Something went wrong"
```

- JavaScript provides the `Error` object for creating and handling errors.
- The `Error` object includes properties like `name` and `message` to describe the error.

Error Handling

(Throwing Errors)

```
function divide(a, b) {  
  if (b === 0) {  
    throw new Error('Division by zero is not allowed');  
  }  
  return a / b;  
}  
  
try {  
  console.log(divide(4, 0));  
} catch (error) {  
  console.error(error.message); // "Division by zero is not allowed"  
}
```

- You can use the `throw` statement to **throw an error manually**.
- Thrown errors can be **caught and handled** by a try...catch block.

Error Handling

(Finally Block)

```
try {  
    console.log('Trying to execute');  
    // Some code that may throw an error  
} catch (error) {  
    console.error('Caught an error:', error.message);  
} finally {  
    console.log('This will always execute');  
}
```

- The **finally** block is used to **execute code regardless of whether an error occurred or not.**
- It is typically used for **cleanup tasks** like closing connections or releasing resources.

Error Handling

(Extending Errors)

```
class ValidationError extends Error {  
  constructor(message) {  
    super(message);  
    this.name = 'ValidationError';  
  }  
  
  function validate(input) {  
    if (!input) {  
      throw new ValidationError('Input is required');  
    }  
  }  
  
  try {  
    validate('');  
  } catch (error) {  
    if (error instanceof ValidationError) {  
      console.error('Validation Error:', error.message);  
    }  
  }  
}
```

- You can create custom error types by extending the built-in Error class.
- This is useful for defining specific error types in your application.



Promises

(Need: Callback Hell)

```
function step1(callback) {  
  setTimeout(() => {  
    console.log('Step 1');  
    callback();  
  }, 1000);  
}  
  
function step2(callback) {  
  setTimeout(() => {  
    console.log('Step 2');  
    callback();  
  }, 1000);  
}
```

```
function step3(callback) {  
  setTimeout(() => {  
    console.log('Step 3');  
    callback();  
  }, 1000);  
}  
  
step1(() => {  
  step2(() => {  
    step3(() => {  
      console.log('All steps completed');  
    });  
  });  
});
```

When multiple asynchronous operations need to be performed in sequence, callbacks can lead to deeply nested and hard-to-read code, often referred to as “callback hell.”

Promises

(States of Promise)



1. Definition: A promise is an **object** representing the eventual **completion** or failure of an asynchronous operation.
2. States of a Promise:
 - Pending: Initial state, **neither fulfilled nor rejected**.
 - Fulfilled: Operation **completed successfully**.
 - Rejected: Operation **failed**.

Promises

(Creation of Promise)

```
// Creating a Promise
let promise = new Promise(resolve, reject) => {
  // Asynchronous operation
  if (result()) {
    resolve('Success');
  } else {
    reject('Error');
  }
};
```

Promises are created using the **Promise constructor**, which takes an **executor function** with two arguments: **resolve** and **reject**.

Promises

(Handling of Promise)

```
// Handling a Promise: handle value
promise.then(value => {
  console.log(value); // 'Success'
});

// Handling a Promise: handle rejection
promise.catch(error => {
  console.error(error); // 'Error'
});

/* Handling a promise: Executes a block of
code regardless of the promise's outcome.*/
promise.finally(() => {
  console.log('Operation completed');
});
```

Promises have `then`, `catch`, and `finally` methods for handling the results of the asynchronous operation.

- `then()`: Used to handle fulfilment.
- `catch()`: Used to handle rejection.
- `finally()`: Executes a block of code regardless of the promise's outcome.

Promises

(Error handling)

```
// Using Promises
fetch('https://api.example.com/data')
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error('Fetch error:', error));

fetchData();
```

Errors in **promises** are handled using **.catch()** or by chaining **.then()** with a second callback for error handling.

Promises

(Solving Callback Hell)

```
function step1() {  
  return new Promise((resolve) => {  
    setTimeout(() => {  
      console.log('Step 1');  
      resolve();  
    }, 1000);  
  });  
  
function step2() {  
  return new Promise((resolve) => {  
    setTimeout(() => {  
      console.log('Step 2');  
      resolve();  
    }, 1000);  
  });  
}
```

```
function step3() {  
  return new Promise((resolve) => {  
    setTimeout(() => {  
      console.log('Step 3');  
      resolve();  
    }, 1000);  
  });  
  
step1()  
  .then(() => step2())  
  .then(() => step3())  
  .then(() => {  
    console.log('All steps completed');  
  });  
}
```

In this version, each step returns a Promise that resolves after a timeout. The steps are chained together using `.then()`, making the code more readable and easier to maintain.

Fetch API

```
fetch('https://jsonplaceholder.typicode.com/posts')
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok' + response.statusText);
    }
    return response.json();
  })
  .then(data => console.log(data))
  .catch(error => console.log(error));
```

- The Fetch API provides a **modern way** to make HTTP requests in **JavaScript**.
- It is a **promise-based API**, making it **easier** to handle asynchronous requests.

Async / Await

```
// using async
async function myFunction() {
  return 'Hello';
}

// using await
async function fetchData() {
  let response = await fetch('https://api.example.com/data');
  let data = await response.json();
  return data;
}
```

1. Syntax Sugar for Promises: `async/await` is built on top of promises, providing a cleaner and more readable way to work with asynchronous code.
2. Defining Async Functions: An `async` function is declared using the `async` keyword before the `function` definition. This function always returns a promise.
3. The `await` keyword is used to pause the execution of an `async` function until a promise is resolved. It can only be used inside an `async` function.

Async / Await

(Handling Exceptions)

```
async function getData() {  
  try {  
    let response = await fetch('https://api.example.com/data');  
    let data = await response.json();  
    return data;  
  } catch (error) {  
    console.error('Error:', error);  
  }  
}
```

Errors in `async` functions can be handled using `try...catch` blocks, making error management straightforward and consistent with synchronous code.

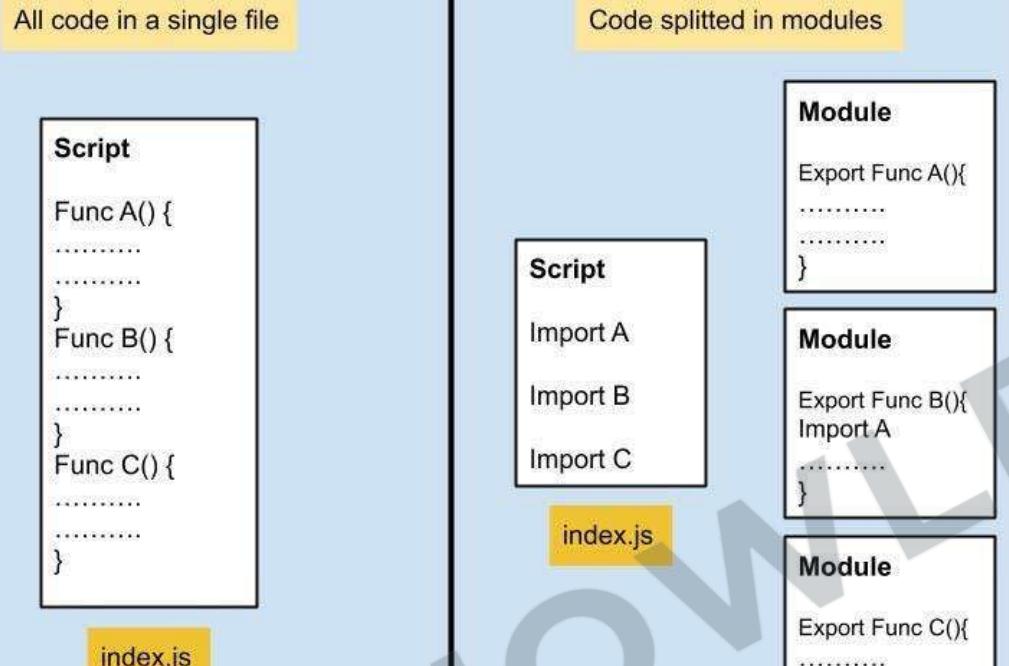
Async / Await

(Fetch API using `async/await`)

```
async function fetchData(url) {  
  try {  
    const response = await fetch(url);  
    if (!response.ok) {  
      throw new Error('Network response was not ok ' + response.statusText);  
    }  
    const data = await response.json();  
    console.log(data);  
  } catch (error) {  
    console.log(error);  
  }  
}  
  
fetchData('https://jsonplaceholder.typicode.com/posts');
```



Modules



- **Modules** are used to **organize and manage code** by dividing it into separate files or modules.
- This modular approach **enhances code maintainability, reusability, and scalability**.
- Modules can be **imported and exported** using the import and export statements.
- Modules are **JavaScript files that encapsulate code** and expose specific parts using the export keyword.
- The **import keyword** is used to **bring in the exported features** from one module to another.

Modules (Named Exports)

Named Exports

Module File: `mathUtils.js`

```
// Named exports
export const PI = 3.14159;

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

export function subtract(a, b) {
  return a - b;
}
```

Importing Named Exports: `main.js`

```
import { PI, add, subtract } from './mathUtils.js';

console.log(`The value of PI is ${PI}`);
console.log(`2 + 3 = ${add(2, 3)}`);
console.log(`5 - 2 = ${subtract(5, 2)})`);
```

Wildcard Import: Import all named exports as an object.

```
import * as mathUtils from './mathUtils.js';

console.log(`The value of PI is ${mathUtils.PI}`);
console.log(`2 + 3 = ${mathUtils.add(2, 3)}`);
console.log(`5 - 2 = ${mathUtils.subtract(5, 2)})`);
```

Allow you to export multiple values from a module. Each export must be imported using its exact name.

Modules (Default Exports)

Default Exports

Module File: `greet.js`

```
// Default export
export default function greet(name) {
  return `Hello, ${name}!`;
}
```

Importing Default Export: `main.js`

```
import greet from './greet.js';
console.log(greet('Alice'));
```

Allow you to export a single default value from a module. The importing module can choose any name for the default export.

Modules (Together)

Named and Default Imports Together:

Module File: shapes.js

```
// Named export
export const squareArea = (side) => side * side;

// Default export
export default function circleArea(radius) {
  return Math.PI * radius * radius;
}
```

Importing Both: main.js

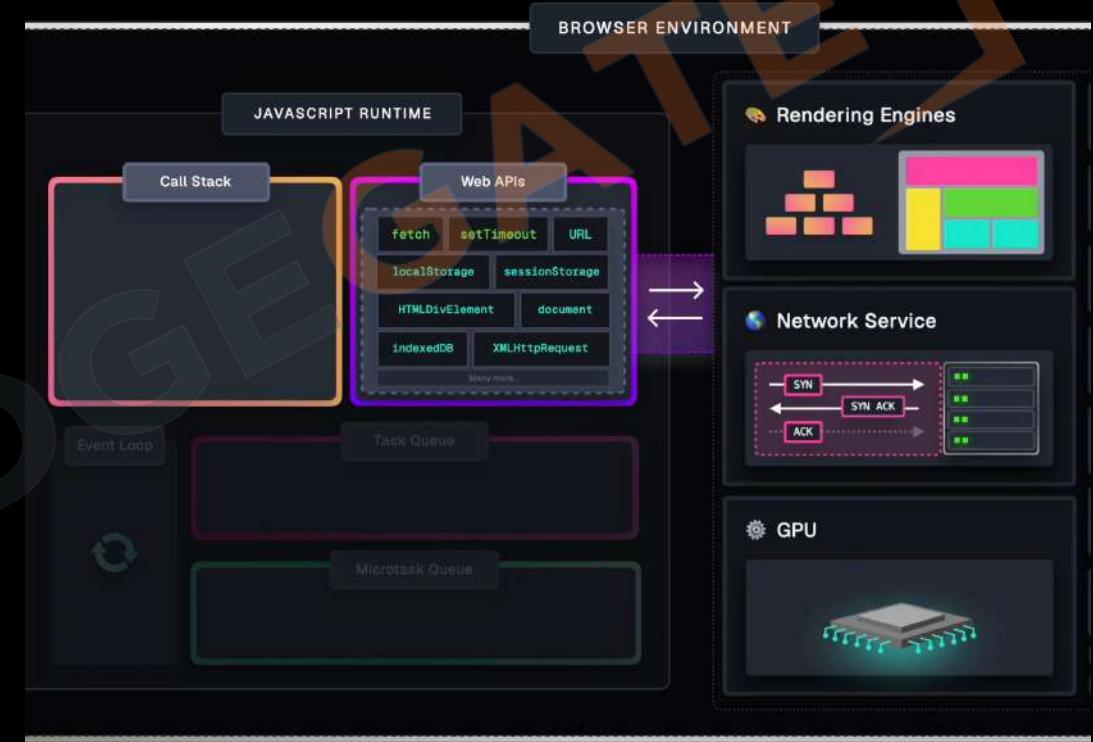
```
import circleArea, { squareArea } from './shapes.js';

console.log(`Circle area with radius 3: ${circleArea(3)})`);
console.log(`Square area with side 4: ${squareArea(4)})`);
```

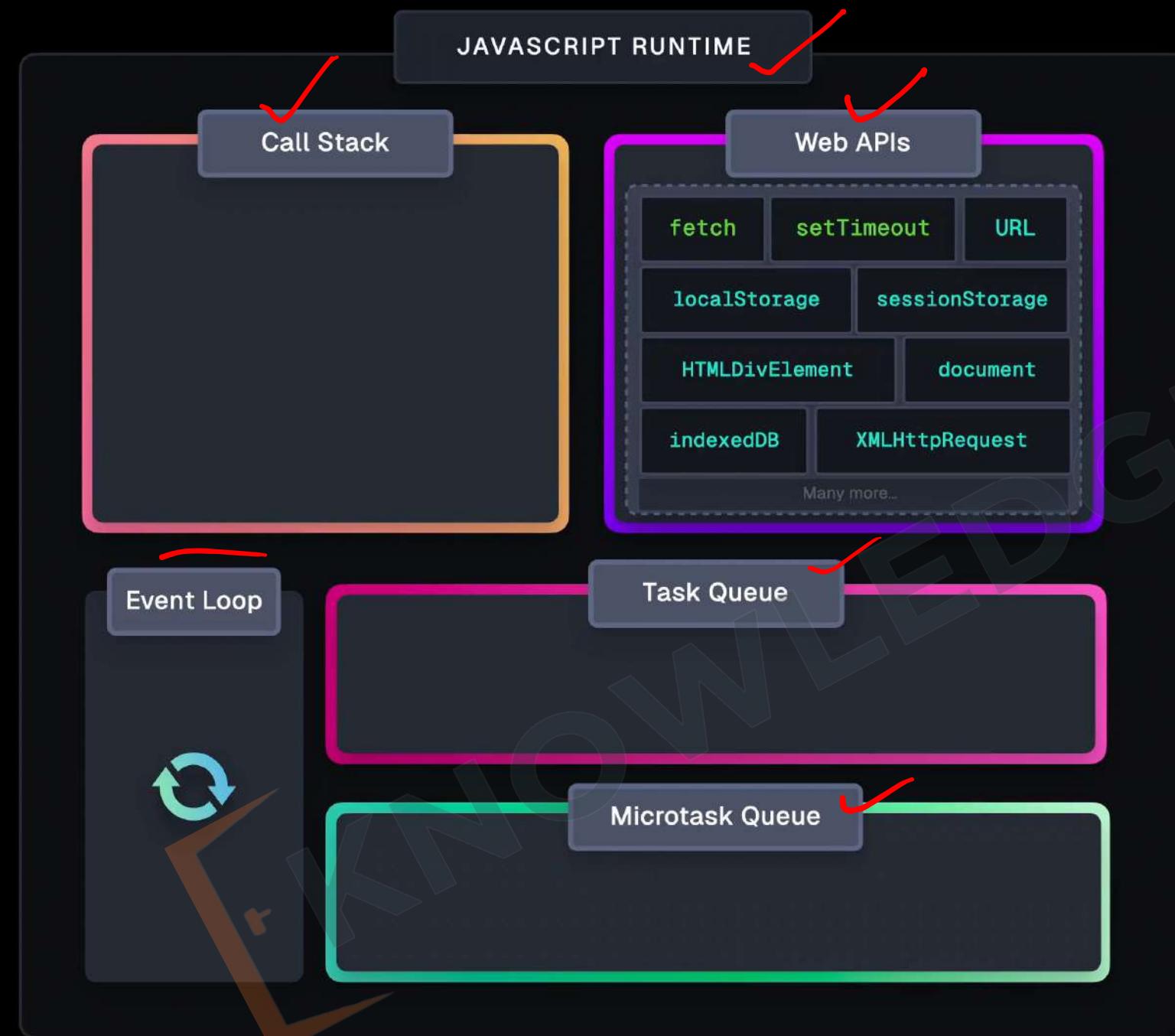


How JavaScript Works?

- Event Loop
- Call Stack
- Web APIs & Browser Env
- Async Task API
- Popular Web APIs
- Geo Location (Callback based API)
- Task Queue
- setTimeout (Callback based API)
- Microtask Queue
- Fetch (Promise based API)



Event Loop

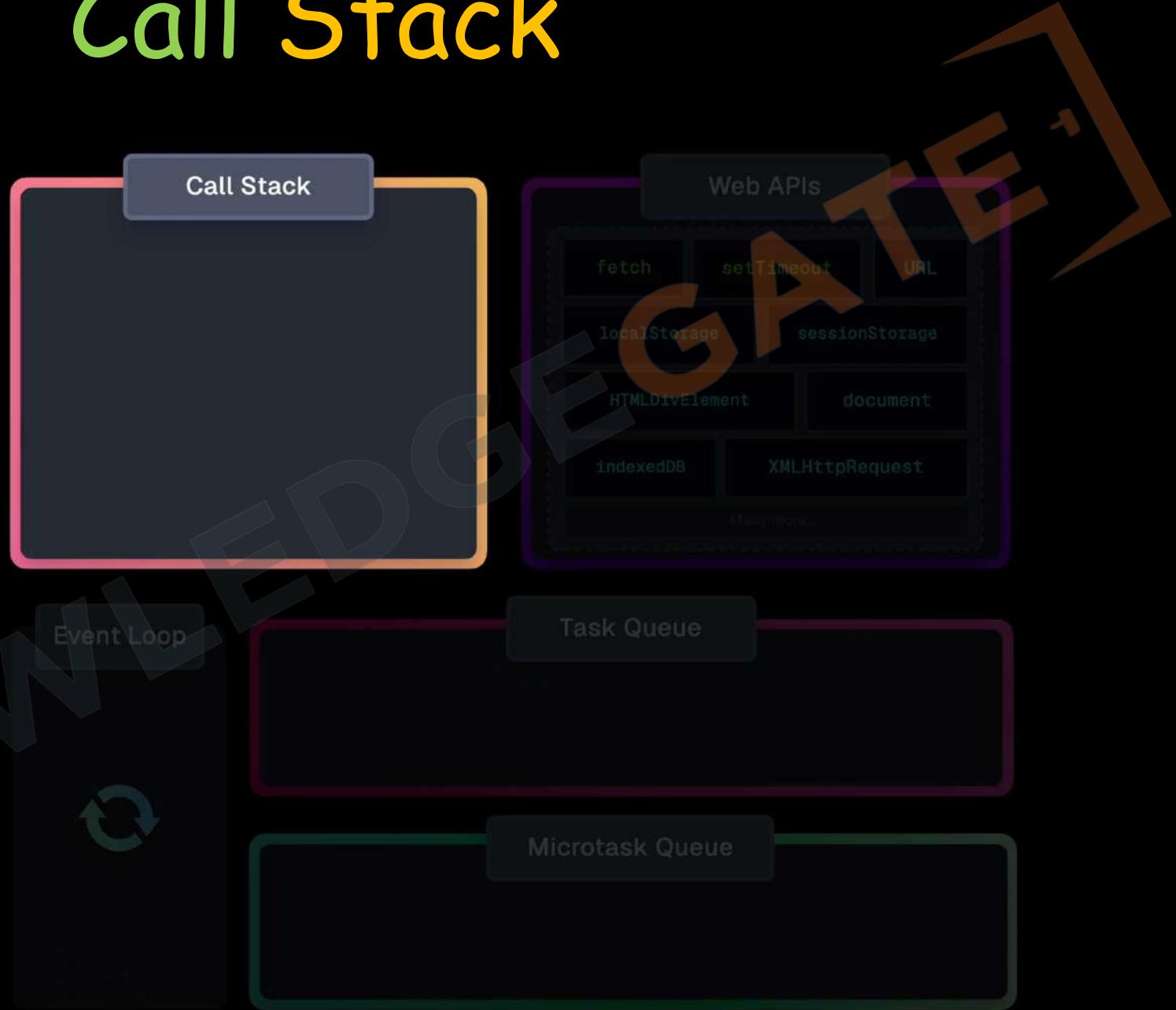


1. it's just a tiny component within the JavaScript runtime!
2. The event loop is a mechanism in JavaScript that handles asynchronous operations, ensuring that non-blocking tasks are executed efficiently.
3. JavaScript runs on a single thread, meaning it can only perform one operation at a time. The event loop helps manage multiple tasks without blocking the main thread.

Call Stack



```
1 console.log("One! ");
2
3 console.log("Two! ");
4
5 function logThree() {
6   console.log("Three! ");
7 }
8
9 function logThreeAndFour() {
10   logThree();
11   console.log("Four! ");
12 }
13
14 logThreeAndFour();
```



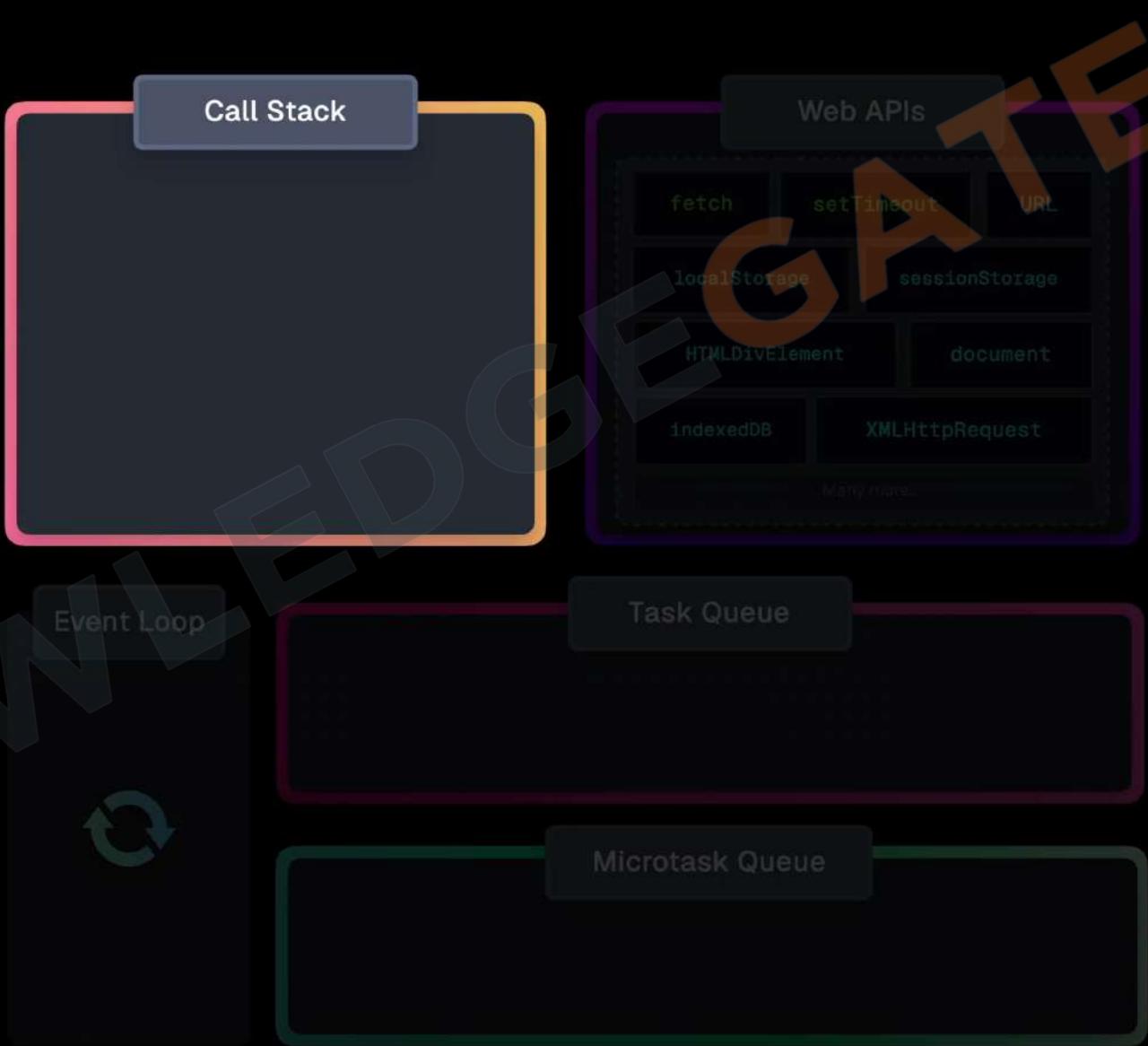
Call Stack

(Problem with only one task at a time)



```
1 function longRunningTask() {  
2     let count = 0;  
3     for (let j = 0; j < 1e9; j++) {  
4         count++  
5     }  
6     console.log("Long task done!");  
7 }  
8  
9 function importantTask() {  
10    console.log("Important!");  
11 }  
12  
13 longRunningTask();  
14 importantTask();
```

console

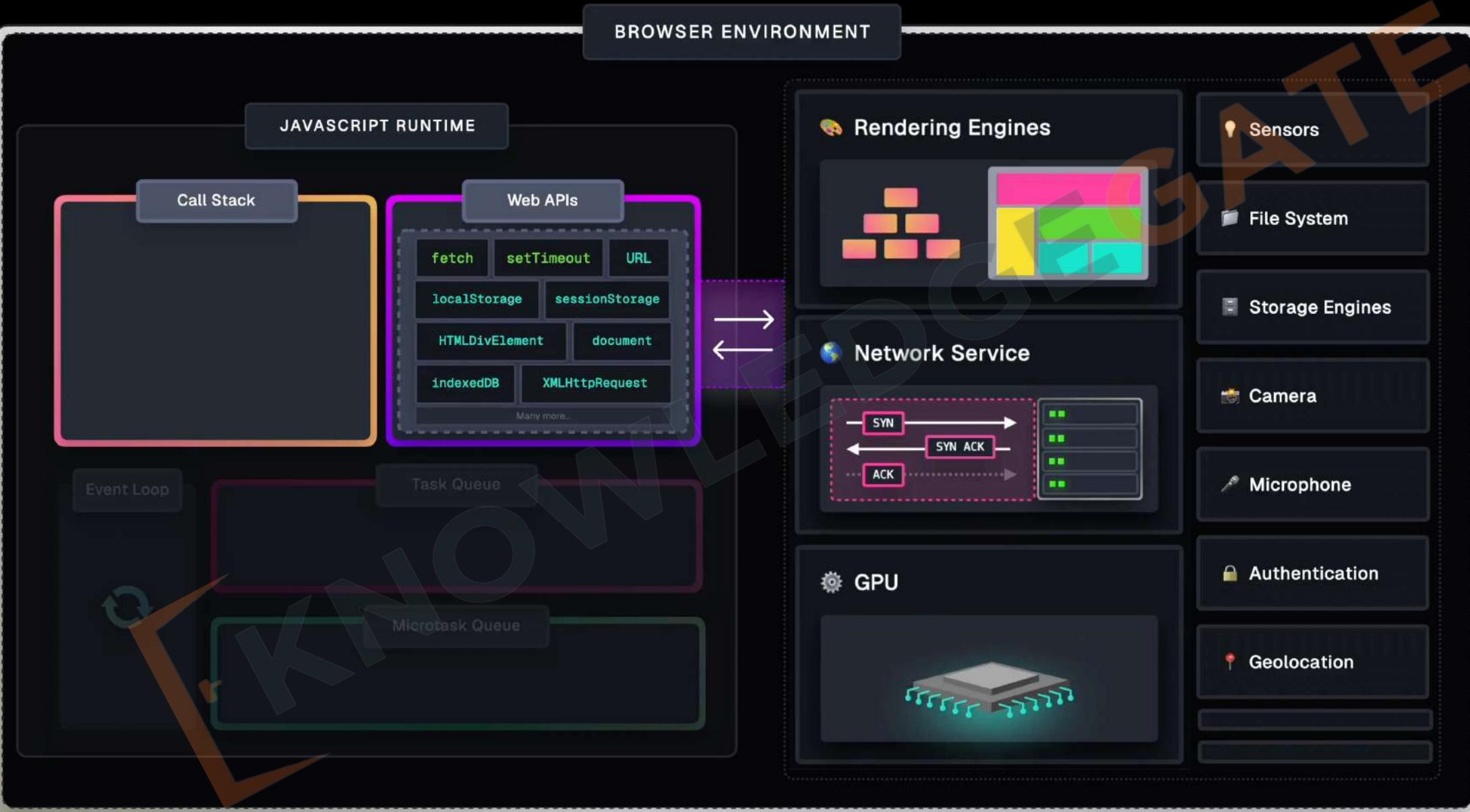


Web APIs & Browser Environment



Solution of only one task at a time.

Web APIs & Browser Environment



Async Task API



```
1  asyncTask((result) => console.log(result));
```

BROWSER ENVIRONMENT

Call Stack

Web APIs

Example Web API

Placeholder for any asynchronous API

asyncTask

callback

Browser Feature



-KNOC-

DELEGATE

Popular Web APIs

CALLBACKS, FOR EXAMPLE:



```
navigator.geolocation.getCurrentPosition(  
  position => console.log(position),  
  error => console.error(error)  
)
```



```
setTimeout(() => console.log("Done"), 2000)
```



```
const request = indexedDB.open("myDb");  
  
request.onsuccess = event => {  
  console.log(event)  
}  
  
request.onerror = error => {  
  console.log(error)  
}
```

PROMISES, FOR EXAMPLE:



```
fetch("")  
.then(res => ...)
```



```
const [fileHandle] = await window.showOpenFilePicker();  
const file = await fileHandle.getFile();
```

Geo Location

(Callback based API)

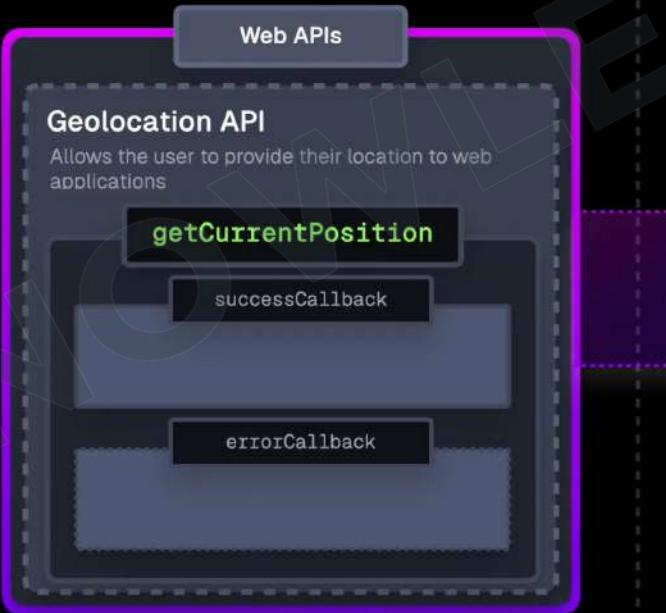
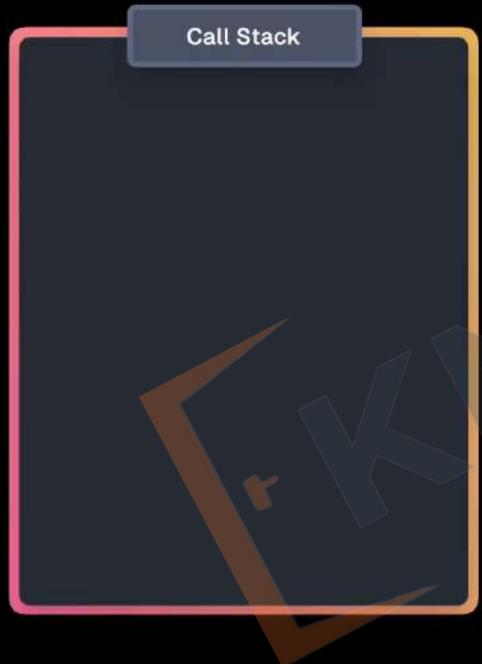


```
navigator.geolocation.getCurrentPosition(  
  position => console.log(position),  
  error => console.error(error)  
)
```

Geo Location

(Call Initiation)

```
1 navigator.geolocation  
2   .getCurrentPosition(  
3     (position) => console.log(position),  
4     (error) => console.error(error)  
5   );
```



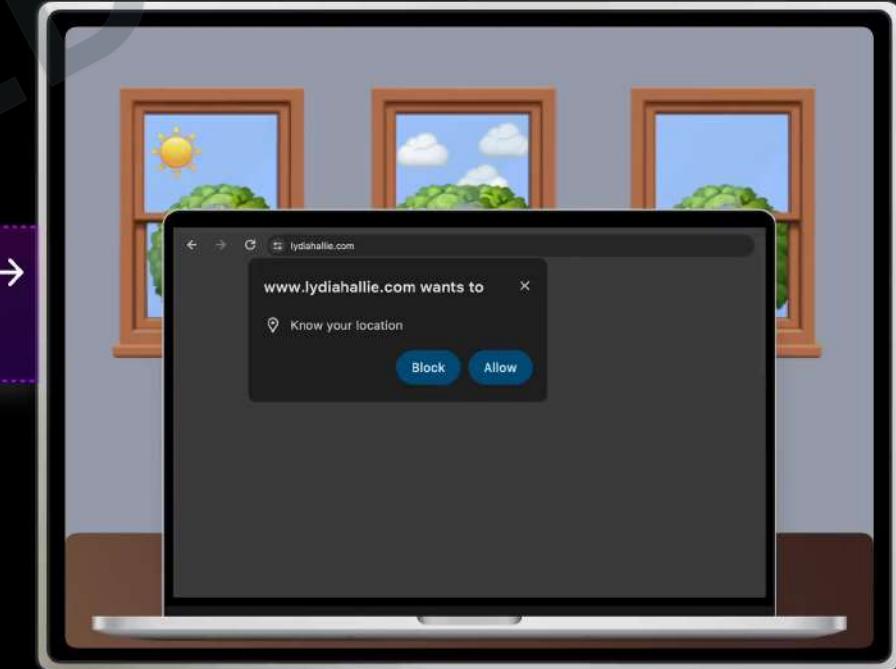
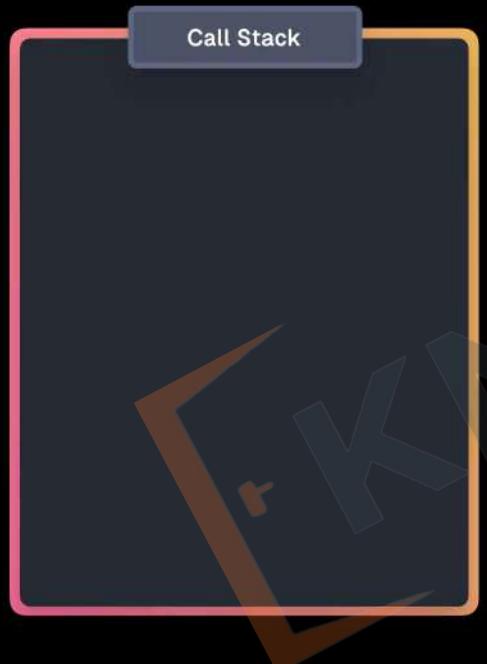
EDGE
DELEGATE

Geo Location

(Call stack executing other tasks)



```
1 navigator.geolocation  
2   .getCurrentPosition(  
3     (position) => console.log(position),  
4     (error) => console.error(error)  
5   );
```



REDUCE

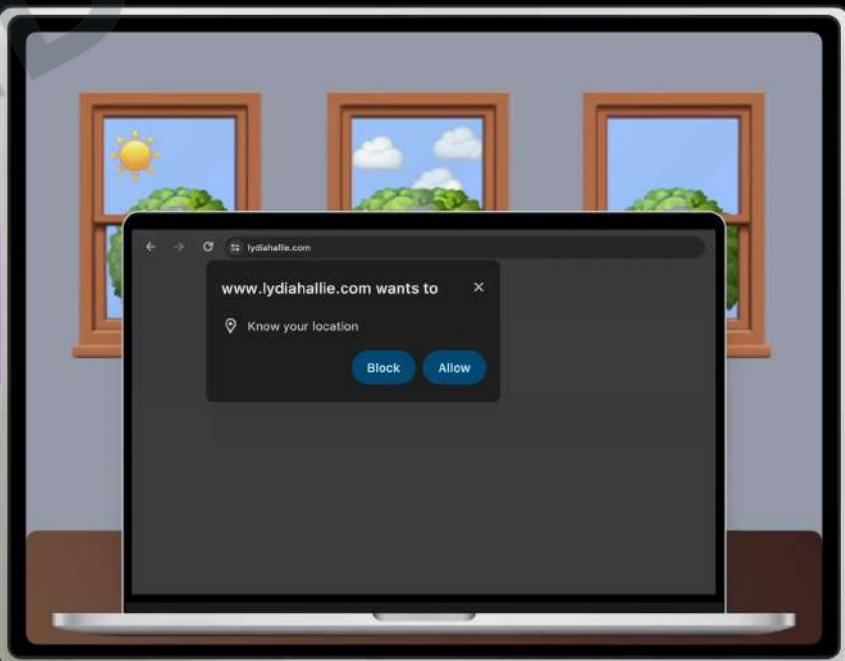
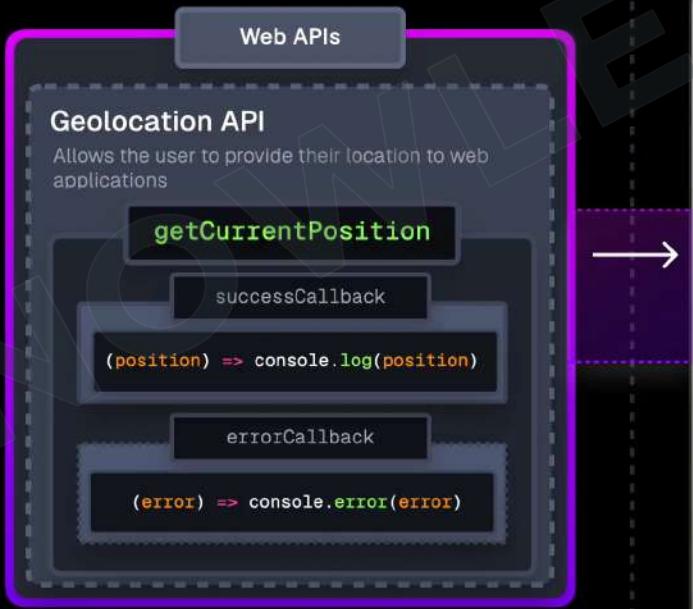
DELEGATE

Geo Location

(Execution of Callback)



```
1 navigator.geolocation  
2   .getCurrentPosition(  
3     (position) => console.log(position),  
4     (error) => console.error(error)  
5   );
```

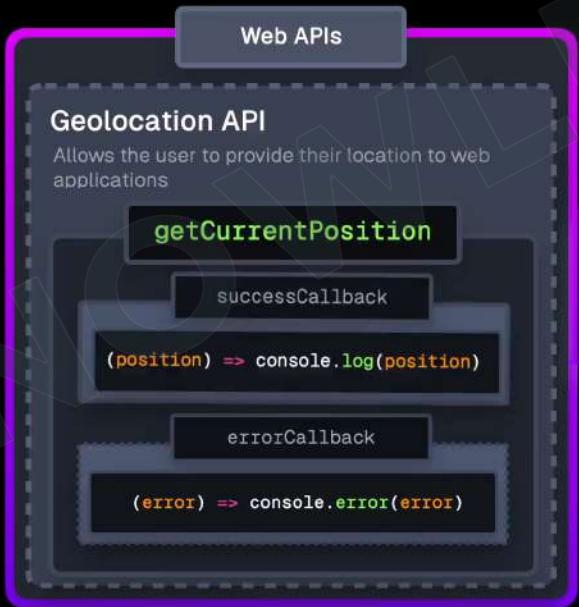


EDGE DELEGATE

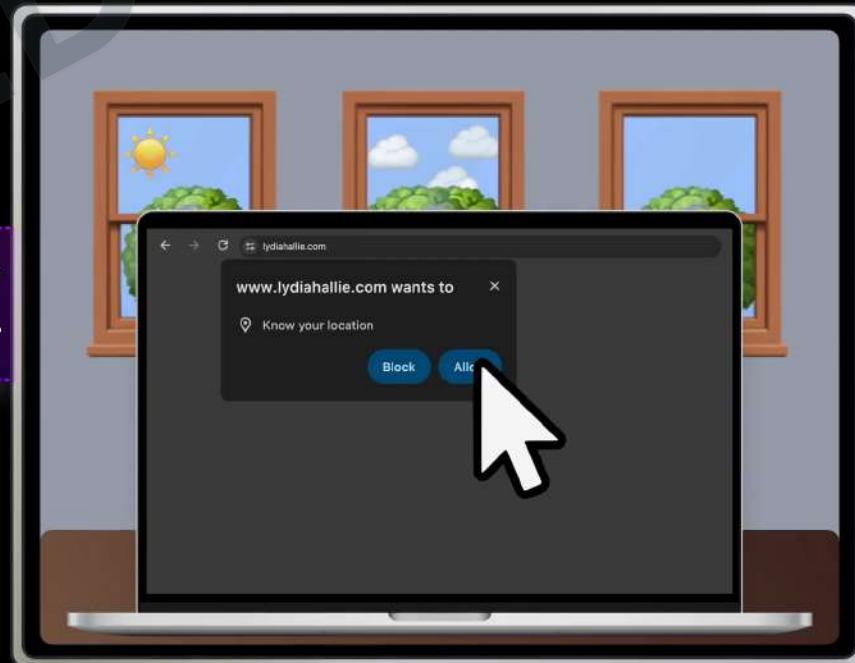
Task Queue



```
1 navigator.geolocation  
2   .getCurrentPosition(  
3     (position) => console.log(position),  
4     (error) => console.error(error)  
5   );
```



EDGEGATE

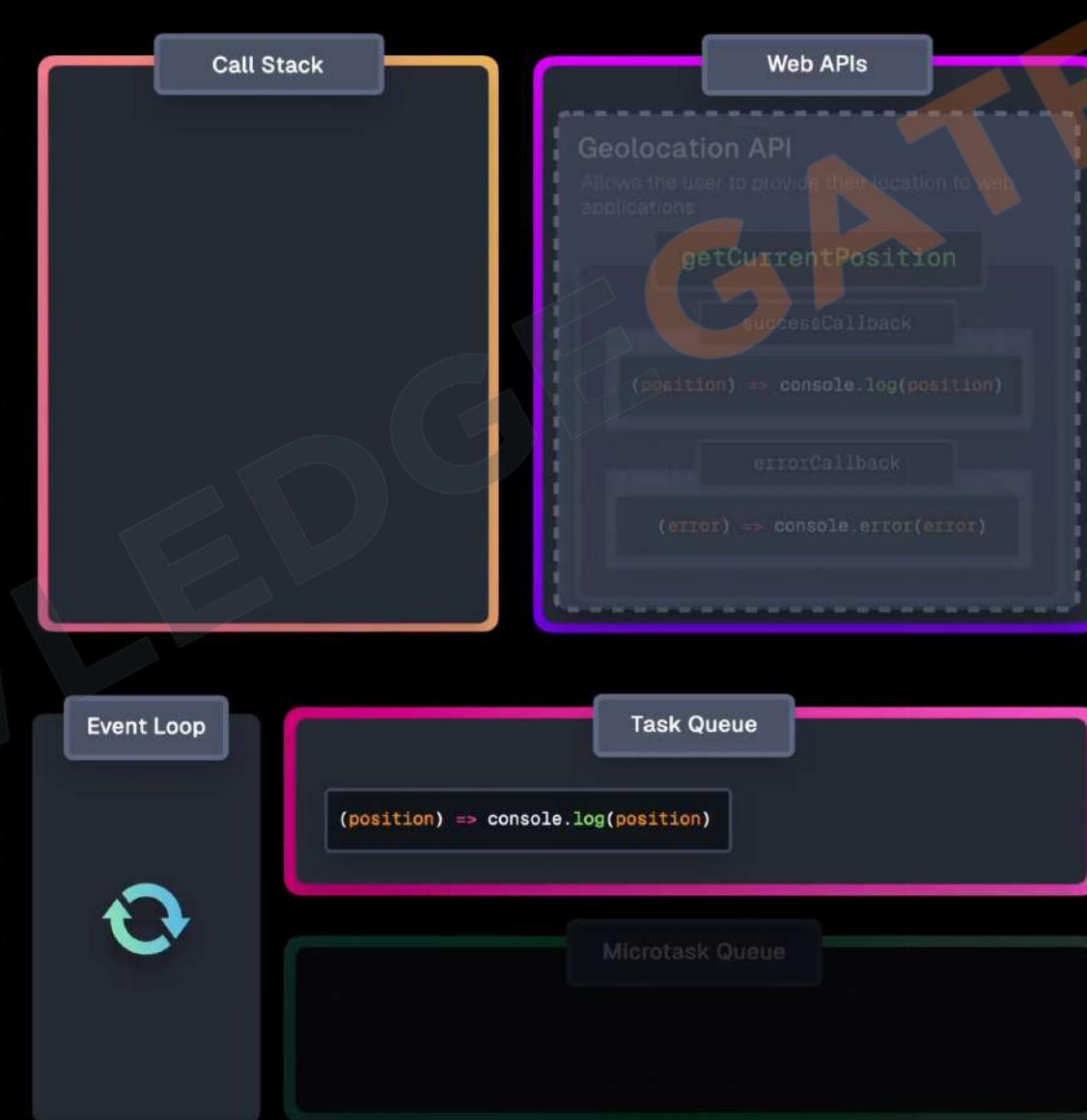


Task Queue & Event Loop



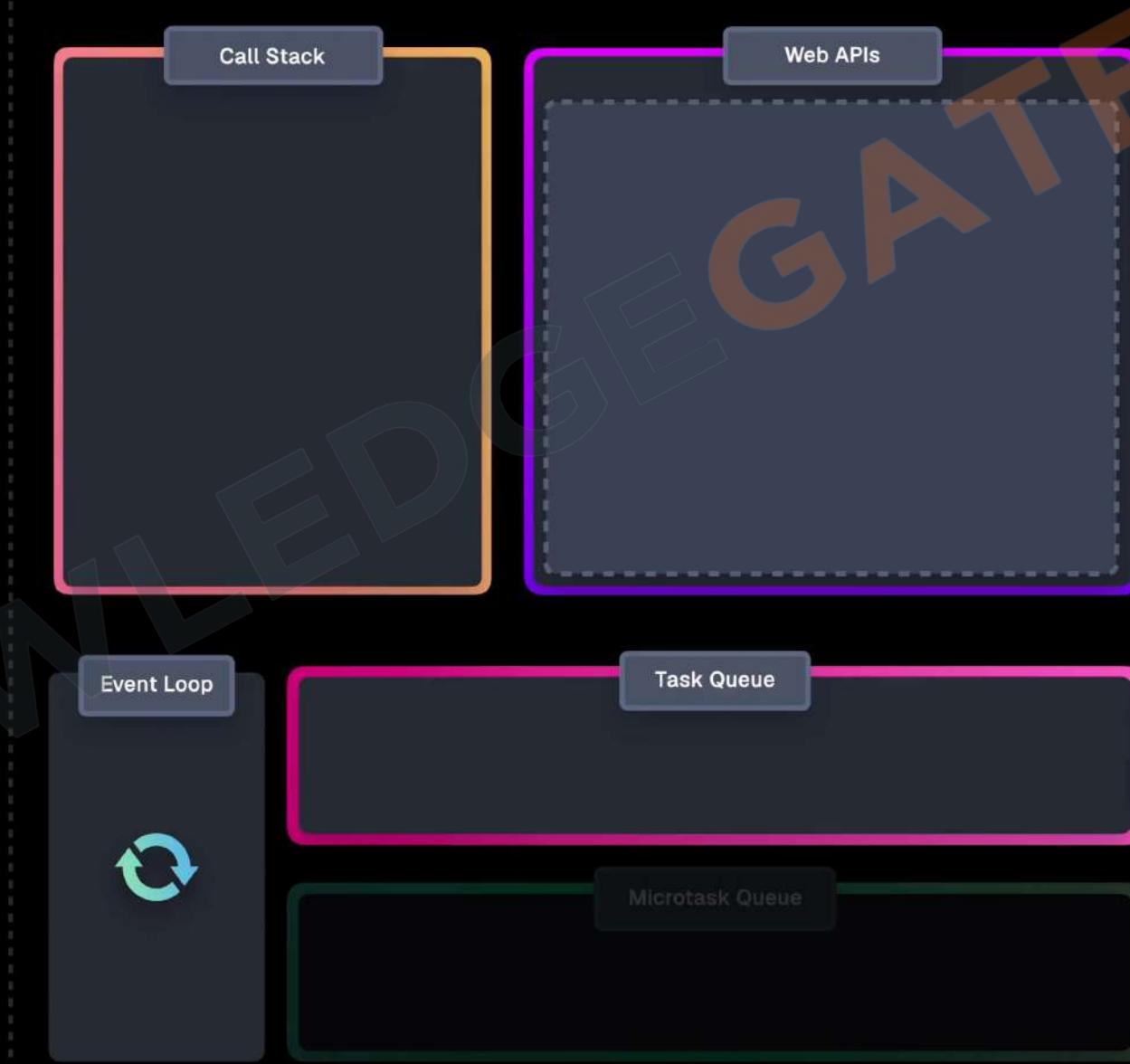
```
1 navigator.geolocation  
2   .getCurrentPosition(  
3     (position) => console.log(position),  
4     (error) => console.error(error)  
5   );
```

console

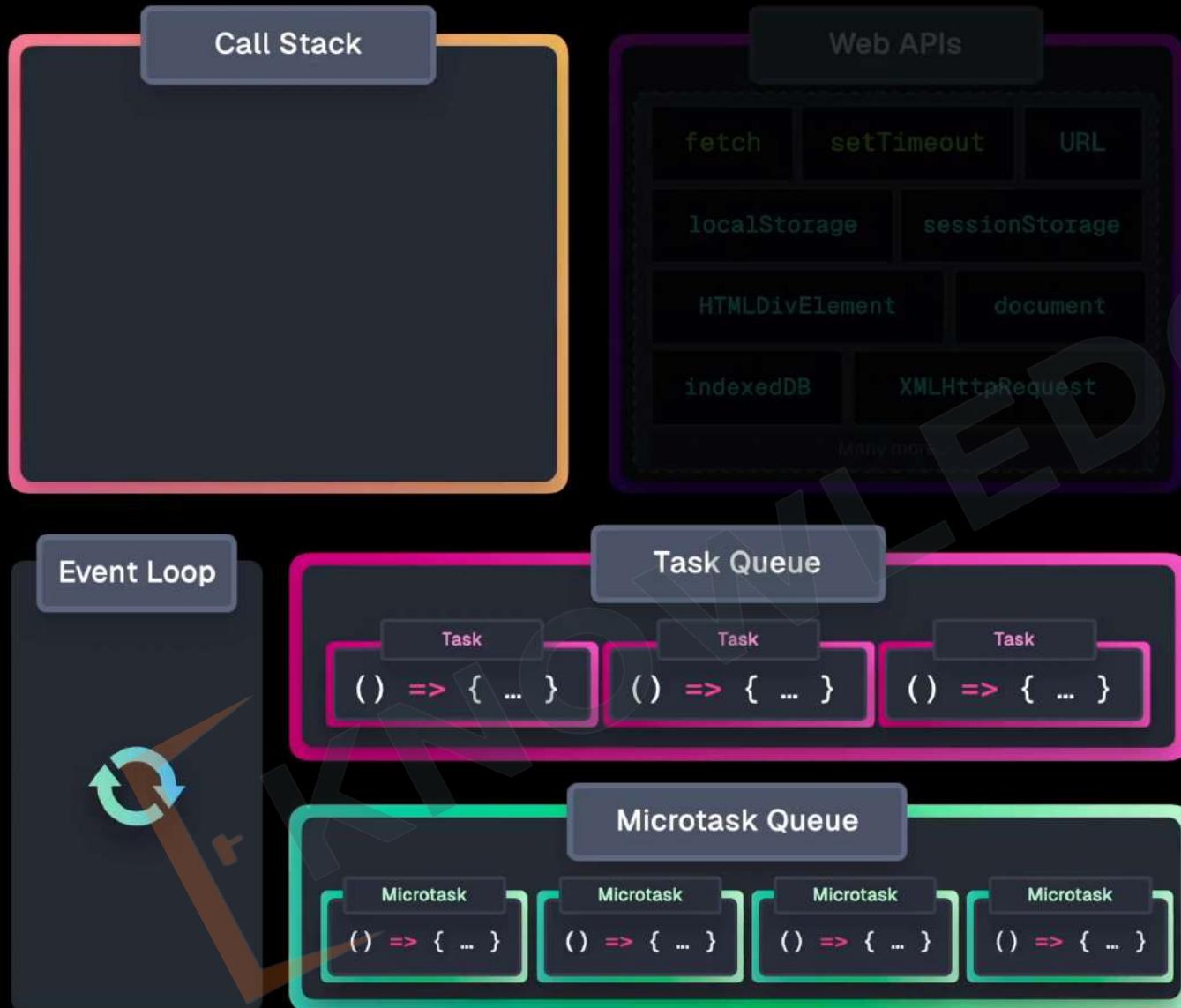


setTimeout (Callback based API)

```
1  setTimeout(() => {  
2      console.log("2000ms")  
3  }, 2000);  
4  
5  setTimeout(() => {  
6      console.log("100ms")  
7  }, 100);  
8  
9  console.log("End of script")
```



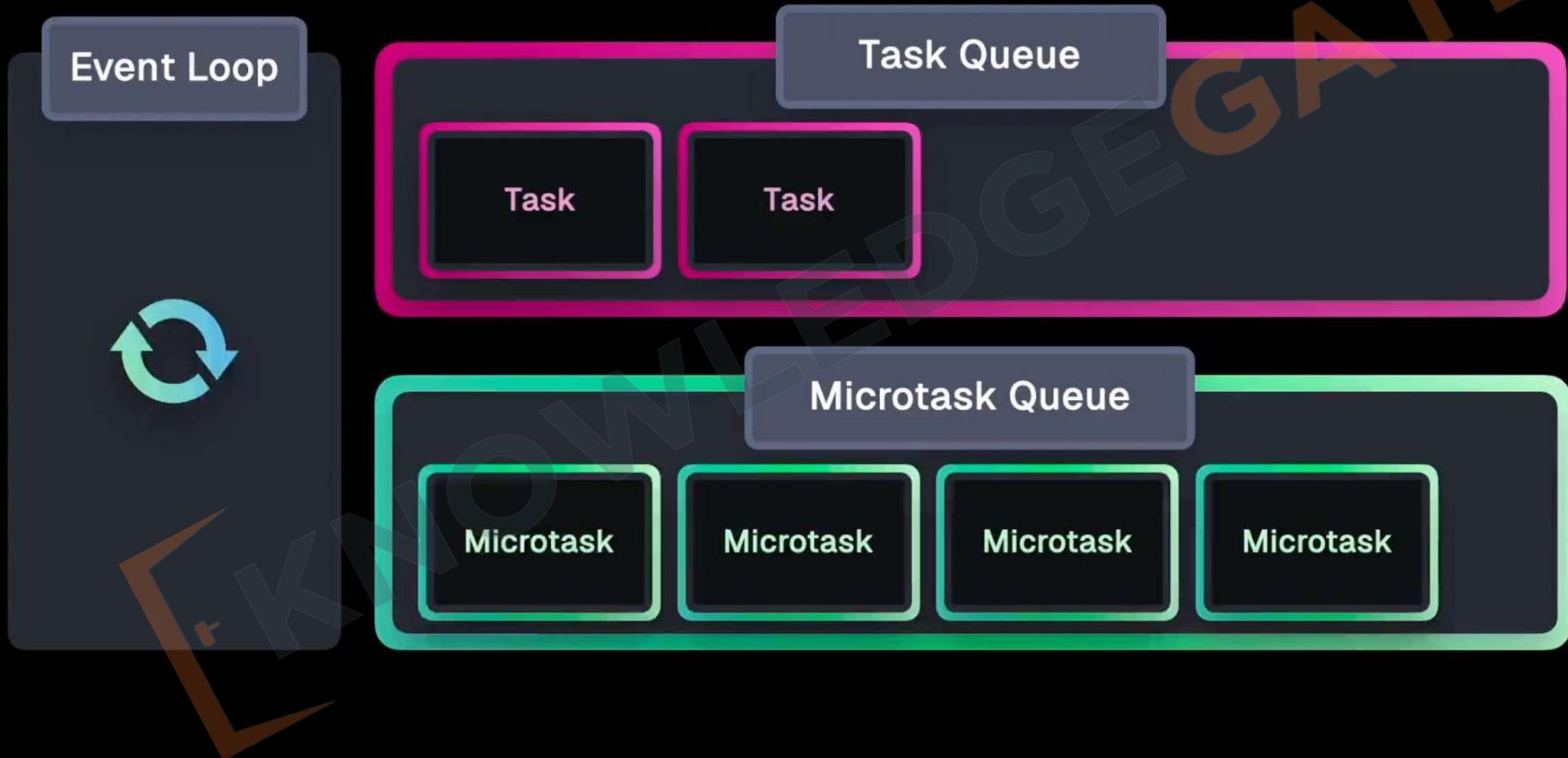
Microtask Queue



The **Microtask Queue** is another queue in the runtime with a **higher priority** than the **Task Queue**. This queue is specifically dedicated to:

- **Promise** handler callbacks (`then(callback)`, `catch(callback)`, and `finally(callback)`)
- Execution of **async** function bodies following **await**
- **MutationObserver** callbacks
- **queueMicrotask** callbacks

Microtask Queue (Infinite Loop)



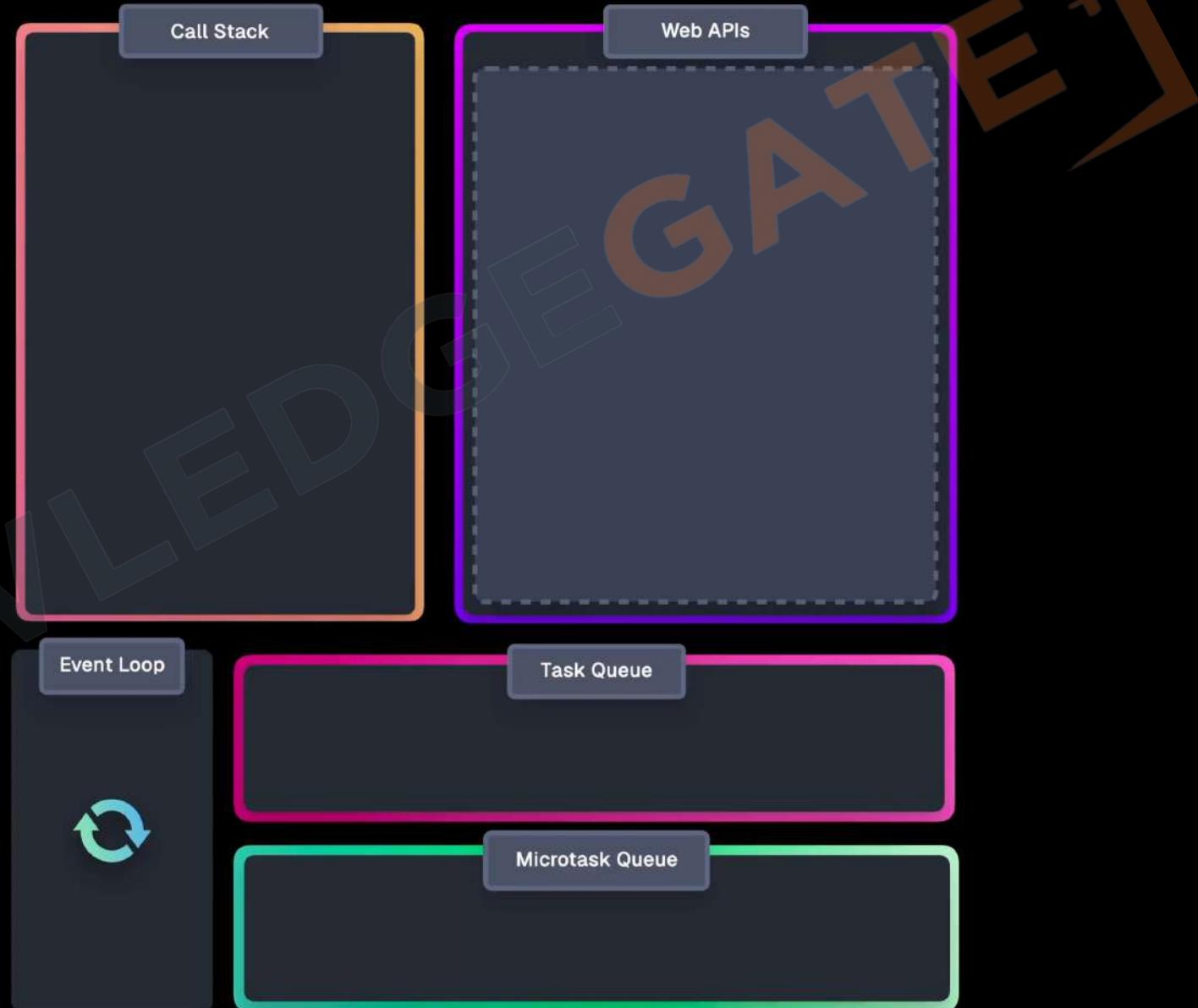
Fetch

(Promise based API)

```
1  fetch("https://api.site.com/posts")
2    .then(res => console.log(res))
3
4  console.log("End of script")
```

console

.KNOWLEDGE



Fetch (Promise Reaction Record)

```
1 fetch("https://api.site.com/posts")
2 .then(res => console.log(res))
3
4 console.log("End of script")
```

console



Fetch

(Normal Execution Continues)

```
1 fetch("https://api.site.com/posts")
2 .then(res => console.log(res))
3
4 console.log("End of script")
```

console



Fetch (Promise Reaction pushed to Microtask queue)

```
1 fetch("https://api.site.com/posts")  
2   .then(res => console.log(res))  
3  
4 console.log("End of script")
```

console
End of script



Concurrency

- Concurrency allows multiple operations to overlap in execution, but not necessarily run simultaneously.
- JavaScript can achieve parallelism through web workers, which allow scripts to run in background threads.
- Concurrency in JavaScript is handled through the event loop and asynchronous programming.
- Asynchronous constructs like callbacks, promises, and async/await allow non-blocking operations.
- Web APIs and timers enable asynchronous task execution without blocking the main thread.
- Understanding concurrency is essential for writing efficient, responsive JavaScript applications, especially for handling I/O operations and user interactions.

