HTML & CSS Interview Questions with clear, beginner-to-intermediate explanations.
These are very common in internships, and junior web dev roles.

□ HTML Interview Questions (with explanations)

1. What is HTML?

HTML (HyperText Markup Language) is the standard language used to structure content on the web.
It tells the browser *what* things are (headings, paragraphs, images), not *how they look*.

Example:

```
<h1>Hello World</h1>
<p>This is a paragraph.</p>
```

---

2. What is the difference between HTML and HTML5?

HTML5 is the latest version of HTML and adds:

- Semantic tags (`<header>`, `<footer>`, `<section>`)
- Audio & video support
- Better form controls
- Local storage (with JavaScript)

HTML (older) relied heavily on `<div>` and needed plugins for media.

---

3. What are semantic elements?

Semantic elements clearly describe their meaning to browsers and developers.

Examples:

- `<header>` – page header
- `<nav>` – navigation links
- `<article>` – independent content
- `<footer>` – page footer

Why important?

- Better accessibility

- Better SEO
- Cleaner code

---

4. Difference between `<div>` and `<span>`?

`<div>`	`<span>`
Block-level	Inline
Takes full width	Takes only needed width
Used for layout	Used for small text styling

---

5. What are attributes in HTML?

Attributes provide extra information about elements.

Example:

```
<img src="image.jpg" alt="A photo">
```

- `src` → image source
- `alt` → alternative text (important for accessibility)

---

6. What is the purpose of the `alt` attribute?

- Helps screen readers
- Shows text if image fails to load
- Improves SEO

---

7. What are empty (void) elements?

Elements that don't need a closing tag.

Examples:

```
<img>
<br>
<hr>
<input>
```

---

8. What is the difference between `id` and `class`?

id	class
Unique	Can be reused
Used once	Used multiple times
`#id` in CSS	`.class` in CSS

---

9. What is the `<meta>` tag used for?

Provides metadata about the page.

Common uses:

- Character set
- Viewport for mobile
- SEO description

```
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

---

10. What is accessibility in HTML?

Accessibility means making websites usable for everyone, including people with disabilities.

Examples:

- Use semantic tags
- Add `alt` text
- Label form inputs

□ CSS Interview Questions (with explanations)

1. What is CSS?

CSS (Cascading Style Sheets) controls the appearance of HTML elements (colors, layout, fonts).

HTML = structure
CSS = style

---

2. What are the different types of CSS?

1. **Inline CSS**

```
<p style="color:red;">Text</p>
```

2. **Internal CSS**

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

3. **External CSS** □ (best practice)

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

---

3. What is the CSS box model?

Every element is made of:

- **Content**
- **Padding**
- **Border**
- **Margin**

Understanding this is key to layout design.

---

4. Difference between `margin` and `padding`?

- **Padding** → space *inside* the element
- **Margin** → space *outside* the element

---

5. What is `display` property?

Controls how elements appear.

Common values:

- `block`
- `inline`
- `inline-block`
- `none`
- `flex`

---

6. What is `position` in CSS?

Defines how an element is positioned.

Values:

- `static` (default)
- `relative`
- `absolute`
- `fixed`
- `sticky`

Example:

```
div {
  position: relative;
  top: 10px;
}
```

---

7. What is Flexbox?

Flexbox is a layout system for aligning items in one direction (row or column).

Key properties:

- `display: flex`
- `justify-content`
- `align-items`

Used for responsive layouts.

---

8. Difference between `class` and `id` selectors?

```
.className { }
#idName { }
```

- `class` → reusable
- `id` → unique

---

9. What is specificity in CSS?

Specificity decides **which CSS rule wins** when multiple rules apply.

Order:

1. Inline styles
2. ID selectors
3. Class selectors
4. Element selectors

---

10. What is responsive design?

Design that works on all screen sizes (mobile, tablet, desktop).

Techniques:

- Media queries
- Flexible layouts
- Relative units (`%`, `em`, `rem`)

Example:

```
@media (max-width: 600px) {
  body {
    background: lightblue;
  }
}
```

What is the `display` property?

The **`display` property** defines **how an HTML element is shown on the page** and **how it behaves in the layout**.

It controls:

- Whether the element starts on a new line
- How much width it takes
- How it aligns with other elements
- Whether it is visible or hidden

`display: block`

✔ What it does

- Element starts on a **new line**
- Takes **full available width**
- You can set **width, height, margin, padding**

✔ Examples of block elements

- `<div>`
- `<p>`
- `<h1>` to `<h6>`
- `<section>`

✔ Example

```
div {
  display: block;
}
```

✔ Key points

- Stacks vertically
- Next element appears on a new line

 Interview tip:
Most layout elements are block by default.

□ `display: inline`

✔ What it does

- Does **not start a new line**
- Takes **only the space it needs**
- **Width and height cannot be set**
- Top and bottom margin don't work

✔ Examples of inline elements

- `<span>`
- `<a>`
- `<strong>`

✔ Example

```
span {
  display: inline;
}
```

✔ Key points

- Flows inside text
- Good for small content

□ **Important:**
Inline elements cannot be resized.

□ `display: inline-block`

✔ What it does

- Behaves like inline (stays in same line)
- Behaves like block (supports width & height)

✔ Example

```
.button {
  display: inline-block;
  width: 120px;
  height: 40px;
}
```

✔ Key points

- Can sit side by side
- Can be sized and spaced

Common use case:
Buttons, navigation items

`display: none`

✔ What it does

- Completely **removes the element from the page**
- Element takes **no space**
- Different from `visibility: hidden`

✔ Example

```
.hide {
  display: none;
}
```

✔ Difference from `visibility: hidden`

display: none	visibility: hidden
Element removed	Element invisible
No space	Space remains

Common use case:
Hide/show elements using JavaScript

`display: flex`

✔ What it does

- Turns an element into a **flex container**
- Child elements become **flex items**
- Allows easy alignment and spacing

✔ Example

```
.container {
  display: flex;
  justify-content: center;
  align-items: center;
}
```

✔ Key points

- Layout in one direction (row or column)
- Makes responsive design easier

 Common use case:
Navigation bars, cards, layouts

□ Comparison Table (Very Useful in Interviews □)

Display Value	New Line	Width Control	Height Control	Use Case
block	□ Yes	□ Yes	□ Yes	Layout sections
inline	□ No	□ No	□ No	Text elements
inline-block	□ No	□ Yes	□ Yes	Buttons, menus
none	□ No	□ No	□ No	Hide element
flex	□ Yes	□ Yes	□ Yes	Modern layouts

□ One-Line Interview Answers

- **block** → Full-width, new line
- **inline** → Same line, no size control
- **inline-block** → Same line + size control
- **none** → Removes element
- **flex** → Flexible layout system

What is `position` in CSS?

The **`position` property** controls **how an element is placed in the document** and **how it behaves when you move it using**:

- `top`
- `right`
- `bottom`
- `left`
- `z-index`

It decides **whether an element follows the normal page flow or is taken out of it**.

□ `position: static` (default)

✔ What it does

- Default position for all elements
- Element follows **normal document flow**
- `top`, `left`, `right`, `bottom`, and `z-index` **do NOT work**

✔ Example

```
div {
  position: static;
}
```

✔ Key points

- Cannot be moved manually
- Stacks normally from top to bottom

□ **Interview tip:**
If position is not mentioned, it's always `static`.

□ `position: relative`

✔ What it does

- Element stays in **normal flow**

- Can be moved **relative to its original position**
- Other elements are **NOT affected**

✔ Example

```
.box {
  position: relative;
  top: 20px;
  left: 10px;
}
```

✔ Key points

- Space is still reserved for the element
- Often used as a **reference for absolute elements**

 Common use case:
Parent container for `position: absolute`

 `position: absolute`

✔ What it does

- Element is **removed from normal flow**
- Positioned relative to the **nearest positioned ancestor**
- If no positioned ancestor → relative to `<body>`

✔ Example

```
.child {
  position: absolute;
  top: 0;
  right: 0;
}
.parent {
  position: relative;
}
```

✔ Key points

- Other elements act like it doesn't exist
- Moves freely inside its parent

 Common use case:
Dropdown menus, tooltips, modals

 `position: fixed`

✔ What it does

- Element is **fixed relative to the viewport**
- Does NOT move when scrolling
- Removed from normal flow

✔ Example

```
.navbar {
  position: fixed;
  top: 0;
  width: 100%;
}
```

✔ Key points

- Always stays in the same place on screen
- Ignores parent elements

 Common use case:
Sticky headers, chat buttons, floating icons

 `position: sticky`

✔ What it does

- Acts like `relative` **until a scroll point**
- Then behaves like `fixed`
- Requires `top`, `left`, etc.

✔ Example

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

✔ Key points

- Scroll-dependent
- Stays within its parent container

 Common use case:
Sticky table headers, section titles

□ Comparison Table (Interview Gold □)

Position	In Flow	Moves With Scroll	Reference
static	□ Yes	□ Yes	Normal flow
relative	□ Yes	□ Yes	Itself
absolute	□ No	□ Yes	Nearest positioned parent
fixed	□ No	□ No	Viewport
sticky	□ Yes	□ (after stick)	Parent + viewport

□ One-Line Interview Answers

- **static** → Default, no movement
- **relative** → Moves relative to itself
- **absolute** → Positioned inside parent
- **fixed** → Fixed to screen
- **sticky** → Relative + fixed on s

What are the types of CSS?

CSS can be applied to HTML in **three different ways**:

1. **Inline CSS**
2. **Internal CSS**
3. **External CSS** □ *(Best practice)*

Each method controls how styles are written and applied.

1□ Inline CSS

✔ What is Inline CSS?

Inline CSS is written **directly inside an HTML element** using the `style` attribute.

✔ Example

```
<p style="color:red;">Text</p>
```

✔ How it works

- The style applies **only to that specific element**
- It has **very high priority (specificity)**

✔ Advantages

- Quick for small changes
- Useful for testing or one-time styling

✔ Disadvantages

- Hard to maintain
- Makes HTML messy
- Not reusable
- Breaks separation of structure and style

✔ When to use

- Rarely
- Temporary fixes

 Interview note:
Inline CSS has the highest priority (except `!important`).

---

2 Internal CSS

✔ What is Internal CSS?

Internal CSS is written inside a `<style>` tag within the `<head>` section of an HTML document.

✔ Example

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

✔ How it works

- Styles apply to **the entire page**
- Rules affect all matching elements on that page

✔ Advantages

- Better than inline CSS
- Useful for single-page styling
- No extra file needed

✔ Disadvantages

- Styles cannot be reused across multiple pages
- Not good for large projects

✔ When to use

- Single page websites
- Page-specific styles

---

3 External CSS (Best Practice)

✔ What is External CSS?

External CSS is written in a **separate `.css` file** and linked to the HTML file.

✔ Example

HTML file

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

style.css

```
p {
  color: red;
}
```

✔ How it works

- One CSS file can style **multiple HTML pages**
- Browser can cache the file (faster loading)

✔ Advantages

- Clean HTML
- Easy maintenance
- Reusable styles
- Best for large projects
- Faster website performance