

CSS Selectors and Styles



CSS selectors: element, class, ID, Color, fonts, text alignment, font size, Box model: padding, margin, border, Applying styles: inline, internal, external,

Here are **clear, exam-ready notes** on **CSS Selectors, Styling properties, Box Model, and Ways to apply CSS**, explained simply with examples

1. CSS Selectors

CSS selectors are used to **select HTML elements** that you want to style.

1.1 Element Selector

Targets HTML elements by their tag name.

Syntax

```
css
p {
  color: blue;
}
```

Explanation

- Applies to all <p> elements
- No symbol is used

1.2 Class Selector

Targets elements with a specific class attribute.

Syntax

```
css
.myClass {
  color: green;
}
```

tram

HTML

```
html
<p class="myClass">Hello</p>
```

Key Point

- Uses a **dot (.)**
 - Can be used on **multiple elements**
-

1.3 ID Selector

Targets a single element with a unique ID.

Syntax

```
css
#myId {
  color: red;
}
✓
```

HTML

```
html
<p id="myId">Hello</p>
```

Key Point

- Uses **#**
 - ID must be **unique**
-

2. CSS Properties

2.1 Color

Used to change text color.

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

Ways to define color:

- Color name: **red**
 - HEX: **#ff0000**
 - RGB: **rgb(255, 0, 0)**
-

2.2 Fonts

Font Family

```
css
p {
  font-family: Arial, sans-serif;
}
```

Font Size

```
css
p {
  font-size: 16px;
}
```

Units:

- px (pixels)
 - %
 - em, rem
-

2.3 Text Alignment

Controls horizontal alignment of text.

```
css
h1 {
  text-align: center;
}
```

Values:

- left
 - right
 - center
 - justify
-

3. CSS Box Model

The **box model** defines the layout of elements.

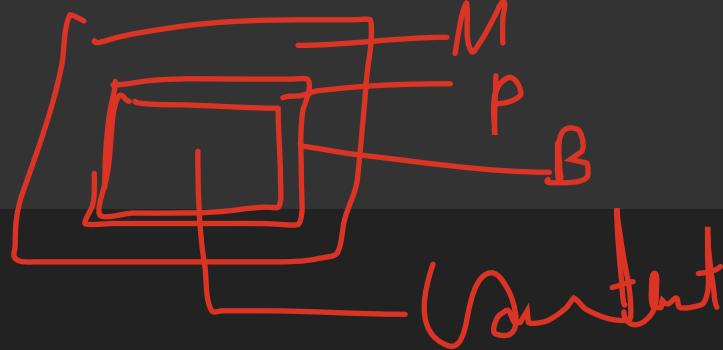
Components of Box Model:

1. **Content** – actual text or image
2. **Padding** – space inside border
3. **Border** – surrounds padding
4. **Margin** – space outside border

Example

css

```
div {
  margin: 20px;
  padding: 10px;
  border: 2px solid black;
}
```



📌 Order (inside → outside):

Content → Padding → Border → Margin

4. Applying CSS Styles

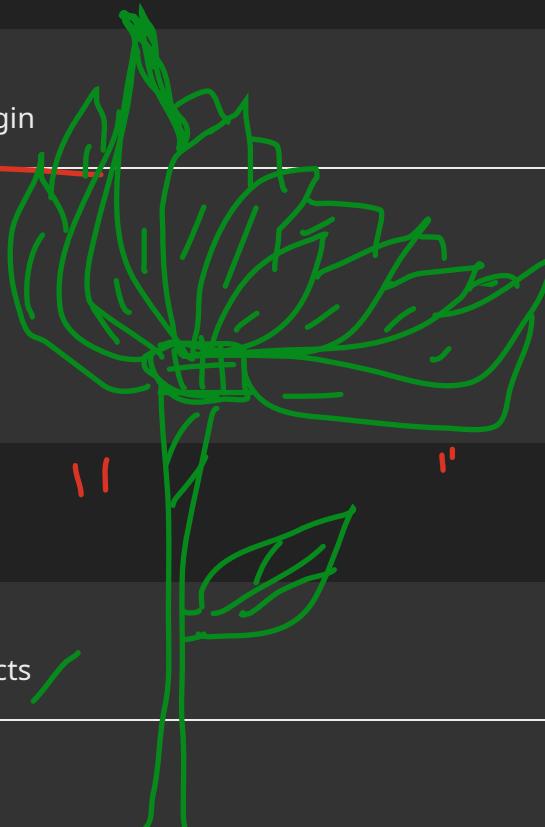
4.1 Inline CSS

Written directly inside HTML tags.

```
html
<p style="color: blue;">Hello</p>
```

✓ Highest priority

✗ Not recommended for large projects



4.2 Internal CSS

Written inside `<style>` tag in `<head>`.

```
html
<style>
p {
  color: green;
}
</style>
```

✓ Good for single-page styling

4.3 External CSS

Written in a separate `.css` file.

`style.css`

css

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

HTML

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

- ✓ Best practice
- ✓ Reusable
- ✓ Easy maintenance

5. Comparison Table (Exam Friendly)

Type	Location	Priority	Usage
Inline	Inside tag	Highest	Small changes
Internal	<head>	Medium	Single page
External	.css file	Lowest	Large projects

If you want, I can also give:

- **1-mark / 2-mark exam answers**
- **MCQs**
- **Short notes format**
- **Practical HTML + CSS example**

Just tell me 



display: block, inline, inline-block, float and clear

Introduction to Flexbox: justify-content, align-items, flex-direction, Units: px, %, em, rem, vh, vw
explain

Here are **clean, exam-oriented notes** on **CSS Display properties, Float & Clear, Flexbox basics, and CSS Units**, explained simply with examples 

1. CSS display Property

The **display** property defines **how an element is shown on the page**.

1.1 display: block

- Takes **full width**
- Starts on a **new line**
- Width & height can be set

Examples: `<div>`, `<p>`, `<h1>`

css

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

`<div>`
`<p>`
`<h1>` to `<h6>`
`<section>`
`<article>`
``
`<table>`

1.2 display: inline

- Takes **only required width**
- Does **not start** on new line
- **Width & height cannot be set**

Examples: ``, `<a>`, ``

css

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

``
`<a>`
``
``
``
`<label>`

1.3 display: inline-block

- Behaves like inline
- Allows **width & height**
- Does not **start on new line**

`<button>`
`<input>`
`<select>`

css

```
button {
  display: inline-block;
  width: 100px;
}
```

📌 **Best of both inline and block**

Comparison

Property	New Line	Width/Height	Full Width
block	Yes	Yes	Yes
inline	No	No	No

Property	New Line	Width/Height	Full Width
inline-block	No ↗	Yes ↗	No

2. Float and Clear

2.1 float

Used to **move elements left or right**.

```
css
img {
  float: left;
}
```

Values:

- **left**
- **right**
- **none**

✖ Used earlier for layouts (now mostly replaced by Flexbox)

2.2 clear

Used to **stop floating effect**.

```
css
div {
  clear: both;
}
```

Values:

- **left**
- **right**
- **both**

✖ Prevents elements from wrapping around floated elements

3. Introduction to Flexbox

Flexbox is a **one-dimensional layout system** for aligning items in rows or columns.

To activate Flexbox:

css

```
.container {  
  display: flex;  
}
```

3.1 flex-direction

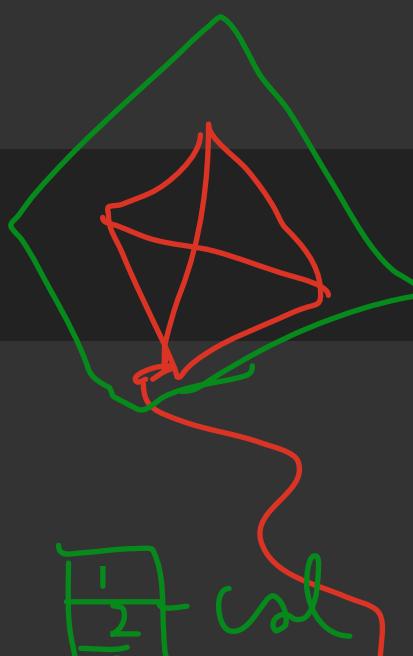
Defines the **direction of items**.

css

```
.container {  
  flex-direction: row;  
}
```

Values:

- **row** (default)
- **row-reverse**
- **column**
- **column-reverse**



3.2 justify-content

Aligns items **horizontally (main axis)**.

css

```
.container {  
  justify-content: center;  
}
```

Values:

- **flex-start**
- **flex-end**
- **center**
- **space-between**
- **space-around**
- **space-evenly**

3.3 align-items

Aligns items **vertically** (cross axis).

css

```
.container {
  align-items: center;
}
```

Values:

- stretch (default)
- center
- flex-start
- flex-end
- baseline



🧭 Axis Understanding

- **Main axis** → controlled by justify-content
- **Cross axis** → controlled by align-items



4. CSS Units

Units define **measurement sizes** in CSS.

4.1 Absolute Unit

px (Pixel)

- Fixed size
- Not responsive

css

```
p {
  font-size: 16px;
}
```

4.2 Relative Units

%

- Relative to parent element

css

```
div {
  width: 50%;
```

em

- Relative to parent's font size

18px

css

```
p {
  font-size: 2em;
```

36px

rem

- Relative to root (html) font size

~~fixed root~~

css

```
p {
  font-size: 1.5rem;
```

📌 rem is more predictable than em

vh (Viewport Height)

- 1vh = 1% of viewport height

css

```
section {
  height: 100vh;
```

vw (Viewport Width)

- 1vw = 1% of viewport width

css

```
div {
  width: 50vw;
```

5. Units Comparison (Exam Friendly)

Unit	Relative To	Responsive
px	Fixed .	✗
%	Parent ↗	✓
em	Parent font ↘	✓
rem	Root font ↙	✓✓
vh	Viewport height	✓
vw	Viewport width	✓

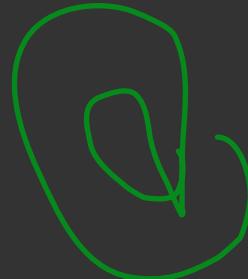
6. One-Line Exam Answers

- **display:** Controls how an element is displayed.
- **float:** Positions elements left or right.
- **clear:** Stops floating elements.
- **Flexbox:** Layout model for alignment in row/column.
- **justify-content:** Aligns items horizontally.
- **align-items:** Aligns items vertically.

If you want next:

- **Flexbox full layout example**
- **MCQs**
- **Practical lab program**
- **Grid vs Flexbox difference**

Just say 



Media queries and breakpoints, Mobile-first design, Responsive navigation concepts, Project planning and design,

Below are **clear, structured, exam-ready notes** on **Media Queries & Breakpoints, Mobile-First Design, Responsive Navigation, and Project Planning & Design**, explained simply with examples 

1. Media Queries and Breakpoints

1.1 Media Queries

Media queries allow CSS to **apply styles based on screen size, device, or resolution.**

Syntax

```
css

@media (max-width: 768px) {
  body {
    background-color: lightgray;
  }
}
```

👉 The above CSS applies **only when screen width \leq 768px**

1.2 Common Media Features

- `max-width`
- `min-width`
- `orientation`
- `resolution`

1.3 Breakpoints

Breakpoints are **screen widths** where layout **changes**.

320-480
481-768
769-1024
1024 and more

Common Breakpoints (Industry Standard)

Device	Width
Mobile	320px - 480px
Tablet	481px - 768px
Laptop	769px - 1024px
Desktop	1025px and above

👉 Breakpoints are **design decisions**, not fixed **rules**

2. Mobile-First Design

2.1 What is Mobile-First Design?

Mobile-first design means:

👉 **Designing for small screens first**, then enhancing for larger **screens**.

2.2 How Mobile-First Works

- Default CSS → Mobile
- Use min-width media queries → Tablet & Desktop

Example

css

```
/* Mobile first */
body {
  font-size: 14px;
}

/* Tablet and above */
@media (min-width: 768px) {
  body {
    font-size: 16px;
  }
}
```

2.3 Advantages of Mobile-First

- Better performance
- Improved user experience
- Easier scalability
- Google-friendly (SEO)

better perform
improve user exp
easy scalable
seo friwndly

3. Responsive Navigation Concepts

Responsive navigation adapts menus for different screen sizes

3.1 Desktop Navigation

- Horizontal menu
- Always visible

css

```
nav ul {
  display: flex;
```

3.2 Mobile Navigation

- Hamburger menu (Ξ)
- Hidden by default
- Shown on click

Common Techniques

- Toggle menu using CSS/JavaScript
- Use Flexbox for layout
- Hide/show using display or transform.

📌 Example concept:

```
css

@media (max-width: 768px) {
  nav ul {
    display: none;
  }
}
```

3.3 Key Responsive Nav Features

- Touch-friendly buttons
- Vertical layout on mobile
- Smooth transitions
- Accessible menu icons



4. Project Planning and Design

Project planning is the process of organizing and designing a web project before development.

4.1 Steps in Project Planning

1 Requirement Analysis

- Understand project goals
- Identify target users
- Decide features

2 Content Planning

- Pages needed
- Text, images, media

3 Wireframing

- Basic page structure (layout sketch)

tea

- No colors or styling
- ✖ Tools: Paper, Figma, Balsamiq
-

4 UI/UX Design

- Color scheme
- Typography
- User interaction

5 Technology Selection

- HTML, CSS, JavaScript
- Frameworks (Bootstrap, React – if needed)

6 Development

- Coding the website
- Responsive design implementation

req analysis
content planning
wireframing
uiux designing
tech selection
development
test
deployment

7 Testing

- Cross-browser testing
- Mobile responsiveness
- Bug fixing

8 Deployment & Maintenance

- Hosting
- Domain setup
- Updates

saves time
saves cost
improve team coordination
quality output
reduces error

4.2 Importance of Project Planning

- Saves time and cost
- Reduces errors
- Improves team coordination
- Ensures quality output

cost nd time
reduce error
team coordination
quality output

5. Short Exam Notes (1-2 Marks)

- **Media Query:** CSS technique to apply styles based on screen conditions.
 - **Breakpoint:** Screen width where layout changes.
 - **Mobile-First Design:** Designing for mobile before desktop.
 - **Responsive Navigation:** Menu adapting to screen sizes.
 - **Wireframe:** Basic visual guide of layout.
-

If you want next:

- **Responsive website mini-project**
- **Media query practical program**
- **MCQs + viva questions**
- **Full project report format (BCA/BBA)**

Just tell me 



Version control in large projects, Deploy using GitHub Pages, Final portfolio polishing

Below are **clear, practical, exam-ready notes** on **Version Control in Large Projects, Deploying with GitHub Pages, and Final Portfolio Polishing**, written in simple language with real-world relevance 

1. Version Control in Large Projects

1.1 What is Version Control?

Version control is a system that **tracks changes in code over time**, allowing multiple people to work on the same project safely.

Popular Tool: Git

Popular Platform: GitHub

1.2 Why Version Control is Critical in Large Projects

Large projects usually have:

- Multiple developers
- Thousands of files
- Frequent updates

collab
history of changes
parallel dev
backup nd safety of code
better proj management

Benefits

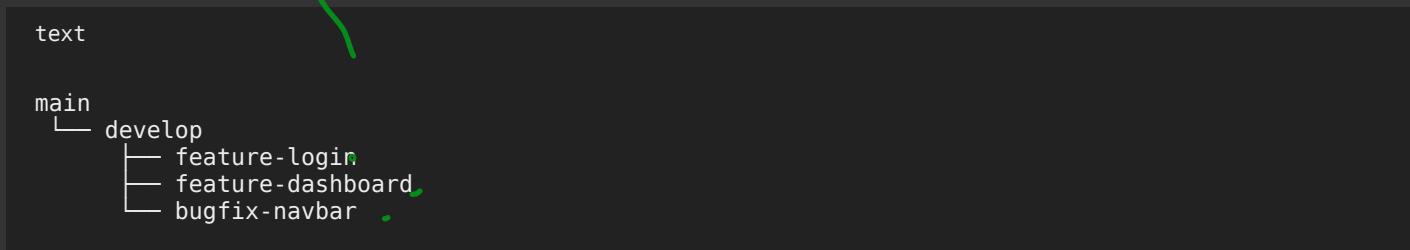
- Team collaboration
- History of changes (rollback possible)
- Parallel development
- Code safety & backup
- Better project management

team collab
history of changes rollback
parallel dev\\
code safety nd backup
large better managemnt

1.3 Common Git Workflow in Large Projects

Branching Strategy

- **main / master** → Stable production code
- **develop** → Ongoing development
- **feature branches** → New features
- **bugfix branches** → Fixing issues



1.4 Pull Requests (PR)

- Developers push code to their branch
- Create a **Pull Request**
- Team reviews code
- Merge into main/develop

👉 Ensures **code quality and fewer bugs**

1.5 Version Control Best Practices

- Commit small changes
- Write meaningful commit messages
- Never commit passwords or keys
- Pull before pushing
- Use .gitignore

commit small
commit freq
commit msg
never commit pass ✓
pull before push
use .gitignore

2. Deploy Using GitHub Pages

GitHub Pages allows you to **host static websites for free** using a GitHub repository.

2.1 What Can Be Deployed?

- HTML
- CSS
- JavaScript
- Portfolio websites
- Project demos

✗ Backend apps not supported

2.2 Steps to Deploy Using GitHub Pages

Step 1: Create a GitHub Repository

- Repository name (e.g., `portfolio`)
- Upload your project files
- Ensure `index.html` exists

Step 2: Go to Repository Settings

- Click **Settings**
- Go to **Pages** section

Step 3: Configure GitHub Pages

- Source: main branch
- Folder: /root
- Save

Step 4: Get Live URL

GitHub gives a link like:

```
arduino  
https://username.github.io/portfolio/
```

✗ Changes update automatically after git push

2.3 Common Deployment Errors

- Missing index.html
- Wrong branch selected
- Broken file paths
- Case-sensitive filenames

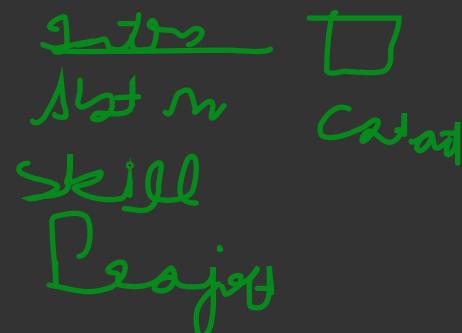
3. Final Portfolio Polishing

Portfolio polishing means **refining your website before submission or interviews.**

3.1 Content Polishing

- ✓ Clear introduction
 - ✓ Professional photo
 - ✓ About Me section
 - ✓ Skills section
 - ✓ Projects with description
 - ✓ Contact details
- ✗ Avoid filler text like "*Lorem Ipsum*"

clear intro
about me
photo
skill section
projects
contact details



3.2 UI & Design Improvements

- Consistent colors
- Proper spacing
- Readable fonts
- Responsive layout
- Smooth navigation

consistent color
proper spacing
readable fonts
responsive layout
smooth navigation

✗ First impression matters in **5 seconds**

3.3 Technical Polishing

Checklist

- ✓ Mobile responsive
- ✓ No console errors
- ✓ Working links
- ✓ Optimized images
- ✓ Clean folder structure

mobile responsive
working links
no error
optimized images
clean folder structure

mobile responsive
no error console
working links
optimized imgs
clean folder structure

3.4 Performance & SEO Basics

- Proper heading structure (h1 → h2 → h3)
- Meta tags
- Fast loading
- Compressed images

3.5 Final Review Before Submission

- Test on mobile & desktop
- Test on Chrome & Firefox
- Re-read all text
- Ask a friend to review
- Check GitHub Pages link

test on mobile
test on chrome
re read content
ask a frnd to review
check github pages link

4. One-Line Exam Answers

- **Version Control:** System to track and manage code changes.
- **GitHub Pages:** Free hosting service for static websites.
- **Branch:** Separate line of development.
- **Pull Request:** Request to merge code into main branch.
- **Portfolio Polishing:** Final improvements to design, content, and performance.

5. Real-World Tip (Important for You)

Since you've worked on **large academic and GitHub projects (like Retro Game Zone)**, showcasing:

- Proper commits
- Clean repo structure
- Live GitHub Pages demo

👉 gives you a **strong edge in interviews and evaluations**.

If you want next:

- **Complete portfolio project structure**
- **README.md professional template**
- **Viva questions & answers**

- Mini project report format

Just say 