**Lecture 1: Flexbox Layouts, Aligning Items, Containers**

**1. What is Flexbox?**

Flexbox (Flexible Box Layout) is a **CSS layout model** used to arrange items in a **one-dimensional** row or column. It makes it **easy to align, distribute, and space elements** inside a container, even when the sizes are unknown or dynamic.

**2. Basic Concepts of Flexbox**

**a. Flex Container**

Apply display: flex to a container:

.container {

display: flex;

}

This makes **all children of .container become flex items**.

**b. Flex Direction**

* flex-direction: row (default): items go left to right
* flex-direction: column: items go top to bottom

.container {

display: flex;

flex-direction: row; /\* or column, row-reverse, column-reverse \*/

}

**c. Aligning Items**

* justify-content: aligns items **horizontally** (main axis)
* align-items: aligns items **vertically** (cross axis)

.container {

display: flex;

justify-content: center; /\* flex-start | center | space-between | space-around | space-evenly \*/

align-items: center; /\* flex-start | center | stretch | baseline \*/

}

**3.Advanced Flexbox Techniques**

**Container Properties:**

**flex-wrap:** Allows items to wrap to the next line.

.container {

display: flex;

flex-wrap: wrap; /\* or nowrap, wrap-reverse \*/

}

**align-content:** Aligns **multiple lines** (only works when flex-wrap is used)

.container {

align-content: space-between;

}

**Item Properties:**

These are applied to individual items inside the container.

**a. flex-grow:**  
How much the item should **grow** compared to others.

.item {

flex-grow: 1;

}

**b. flex-shrink:**  
How much the item should **shrink** if space is tight.

.item {

flex-shrink: 1;

}

**c. flex-basis:**  
Initial size of the item **before** growing or shrinking.

.item {

flex-basis: 200px;

}

Shorthand:

.item {

flex: 1 1 200px; /\* grow shrink basis \*/

}

**4. Building Complex Layouts**

**a. Responsive Grid System**

You can create a **multi-column layout** that adjusts to screen size using Flexbox:

.portfolio-container {

display: flex;

flex-wrap: wrap;

gap: 20px;

}

.portfolio-item {

flex: 1 1 calc(33.33% - 20px); /\* 3 columns \*/

}

**5. Practical Task – Multi-column Portfolio Layout**

**Task: Create a responsive portfolio layout using Flexbox**

**HTML**

<div class="portfolio-container">

<div class="portfolio-item">Item 1</div>

<div class="portfolio-item">Item 2</div>

<div class="portfolio-item">Item 3</div>

<div class="portfolio-item">Item 4</div>

<div class="portfolio-item">Item 5</div>

<div class="portfolio-item">Item 6</div>

</div>

**CSS**

.portfolio-container {

display: flex;

flex-wrap: wrap;

gap: 20px;

padding: 20px;

}

.portfolio-item {

background-color: #f3f3f3;

padding: 20px;

box-shadow: 0 0 10px #ccc;

text-align: center;

font-weight: bold;

}

**Lecture 2: CSS Grid, Building Complex Layouts**

**1. What is CSS Grid?**

**CSS Grid** is a two-dimensional layout system in CSS. It lets you **arrange content in rows and columns**. Unlike Flexbox (1D layout), Grid works in **both directions: horizontal and vertical**.

**2. CSS Grid Basics**

**a. Grid Container**

To start using Grid, make a container a grid:

.container {

display: grid;

}

All direct children become **grid items**.

**b. grid-template-columns & grid-template-rows**

These define how many **columns and rows** you want, and their sizes.

.container {

display: grid;

grid-template-columns: 200px 1fr 1fr; /\* 3 columns \*/

grid-template-rows: 100px auto; /\* 2 rows \*/

}

* 1fr means “one fraction” of available space.
* You can mix fixed and flexible units.

**c. grid-area & grid-template-areas**

You can name sections of the grid for easier layout.

.container {

display: grid;

grid-template-areas:

"header header"

"sidebar content"

"footer footer";

grid-template-columns: 200px 1fr;

grid-template-rows: auto 1fr auto;

}

.header { grid-area: header; }

.sidebar { grid-area: sidebar; }

.content { grid-area: content; }

.footer { grid-area: footer; }

This makes layouts more **readable and manageable**.

**3. Advanced Grid Techniques**

**Alignment and Justification**

* justify-items: aligns **items horizontally** (inside the cell)
* align-items: aligns **items vertically**
* place-items: shorthand for both

.container {

justify-items: center;

align-items: center;

/\* OR simply: \*/

place-items: center;

}

}

**grid-gap (or gap)**

Adds space between rows and columns:

.container {

gap: 20px; /\* or row-gap / column-gap \*/

}

**4. Hands-on Exercise: Dashboard Layout Using CSS Grid**

**Goal:** Create a dashboard with header, sidebar, content, and footer sections.

**Lecture 3: Media Queries, Responsive Design, Mobile-First Approach**

**1. What is Responsive Design?**

**Responsive Web Design** means your website should:

* **Look good on all devices** (mobile, tablet, desktop)
* **Automatically adjust layout and size** based on screen size

It ensures **better user experience** and **accessibility**.

**2. What Are Media Queries?**

**Media Queries** are a CSS feature that allows you to apply **specific styles depending on the device’s screen size, resolution, or orientation**.

**Syntax:**

@media (condition) {

/\* CSS rules here \*/

}

**Example:**

@media (max-width: 768px) {

body {

background-color: lightblue;

}

}

This means: **If the screen width is 768px or less**, apply these styles.

**3. Common Breakpoints (Not fixed, but common)**

| **Device** | **Width (approx.)** |
| --- | --- |
| Mobile | 0 – 767px |
| Tablet | 768px – 1023px |
| Desktop | 1024px and up |

**4. Mobile-First Approach**

**Mobile-First** means:

* Start designing for the **smallest screens first** (mobile),
* Then use media queries to adjust for **larger screens**.

**How?**

Write base styles for mobile, then override with media queries:

]

/\* Mobile First (default) \*/

.container {

font-size: 14px;

flex-direction: column;

}

/\* Tablet and above \*/

@media (min-width: 768px) {

.container {

font-size: 16px;

flex-direction: row;

}

}

/\* Desktop \*/

@media (min-width: 1024px) {

.container {

font-size: 18px;

}

}

**5. Practical Examples**

**Example 1: Change layout on smaller screens**

/\* Mobile layout \*/

.navbar {

flex-direction: column;

}

/\* Tablet and up \*/

@media (min-width: 768px) {

.navbar {

flex-direction: row;

}

}

**Example 2: Hide image on mobile**

.image {

display: block;

}

@media (max-width: 767px) {

.image {

display: none;

}

}

**6. Hands-on Practice Task – Responsive Card Layout**

**HTML:**

<div class="card-container">

<div class="card">Card 1</div>

<div class="card">Card 2</div>

<div class="card">Card 3</div>

</div>

**CSS (Mobile-first):**

.card-container {

display: flex;

flex-direction: column;

gap: 10px;

padding: 20px;

}

.card {

background: #eee;

padding: 20px;

border: 1px solid #ccc;

}

/\* Tablet and up: 2 cards per row \*/

@media (min-width: 768px) {

.card-container {

flex-direction: row;

flex-wrap: wrap;

}

.card {

flex: 1 1 calc(50% - 20px);

}

}

/\* Desktop: 3 cards per row \*/

@media (min-width: 1024px) {

.card {

flex: 1 1 calc(33.33% - 20px);

}

}