# Welcome to lecture 8!

## Agenda

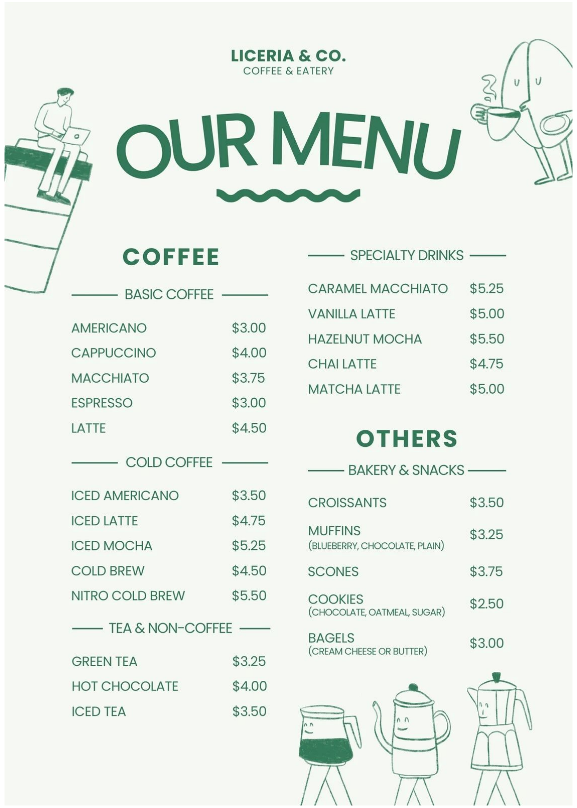
Session Objectives

- Review Homework
- Review CSS concepts
- Flexbox
  - Understanding Flexbox
  - Creating Flexible Layouts
  - Making Layouts Responsive with Media Queries
- Quiz
- Homework (will be shared by batch coordinator from today!)

NO

homework review!

# Preview: designing menus


LICERIA & CO.
COFFEE & EATERY

OUR MENU

COFFEE

BASIC COFFEE

| | |
|---|---|
| AMERICANO | $3.00 |
| CAPPUCCINO | $4.00 |
| MACCHIATO | $3.75 |
| ESPRESSO | $3.00 |
| LATTE | $4.50 |

COLD COFFEE

| | |
|---|---|
| ICED AMERICANO | $3.50 |
| ICED LATTE | $4.75 |
| ICED MOCHA | $5.25 |
| COLD BREW | $4.50 |
| NITRO COLD BREW | $5.50 |

TEA & NON-COFFEE

| | |
|---|---|
| GREEN TEA | $3.25 |
| HOT CHOCOLATE | $4.00 |
| ICED TEA | $3.50 |

SPECIALTY DRINKS

| | |
|---|---|
| CARAMEL MACCHIATO | $5.25 |
| VANILLA LATTE | $5.00 |
| HAZELNUT MOCHA | $5.50 |
| CHAI LATTE | $4.75 |
| MATCHA LATTE | $5.00 |

OTHERS

BAKERY & SNACKS

| | |
|---|---|
| CROISSANTS | $3.50 |
| MUFFINS (BLUEBERRY, CHOCOLATE, PLAIN) | $3.25 |
| SCONES | $3.75 |
| COOKIES (CHOCOLATE, OATMEAL, SUGAR) | $2.50 |
| BAGELS (CREAM CHEESE OR BUTTER) | $3.00 |

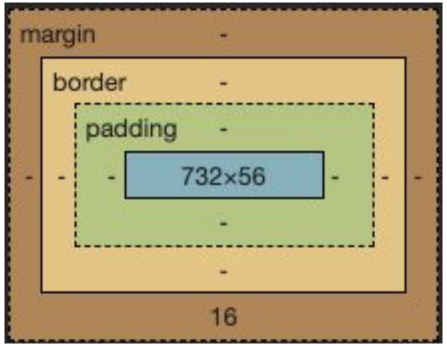
Cafe Coffee Day

District Cafe Bakery

## Prerequisites: A recap of CSS

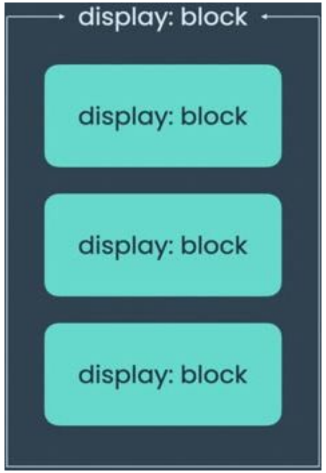
## The CSS Box Model

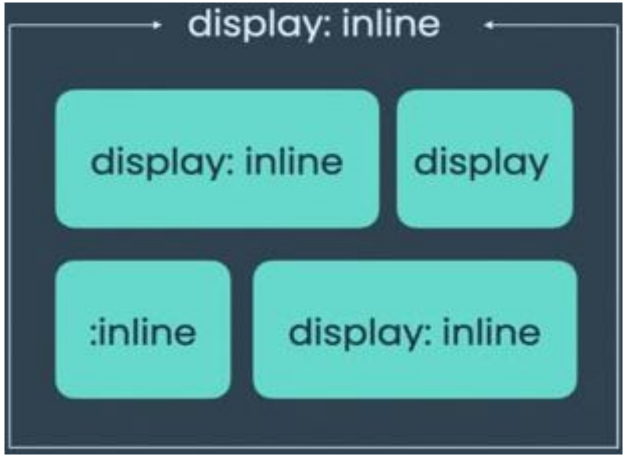
- Recall:
  - Every HTML element is treated as a box
  - The Box Model consists of
    - Content: The actual content (text, image)
    - Padding: Space inside the element
    - Border: A line surrounding the padding and content
    - Margin: Space outside the element

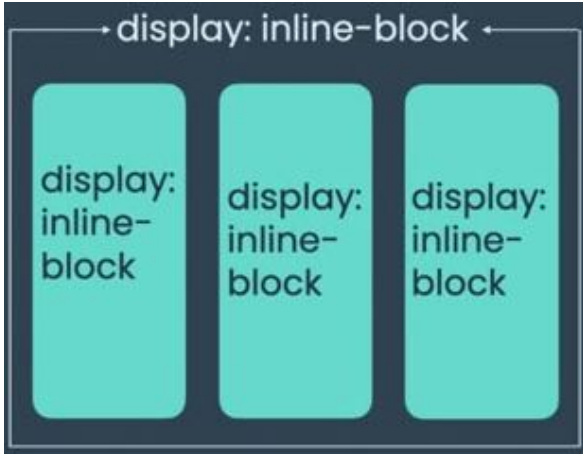


## Element Flow: display Property

- Recall
  - The display property determines how an element is rendered
- Key values
  - block: Element starts on a new line and takes up full width. (e.g., <div>, <p>, <h1>)
  - inline: Element flows with surrounding content. (e.g., <span>, <a>, <img>)
  - inline-block: Element is like inline but allows setting width and height

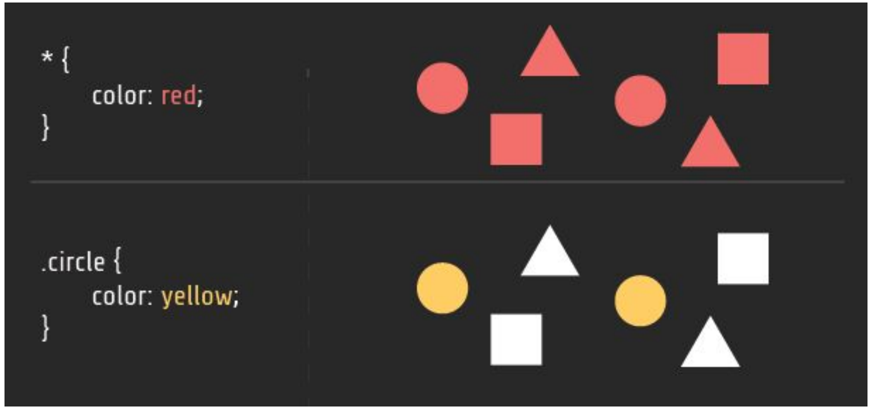




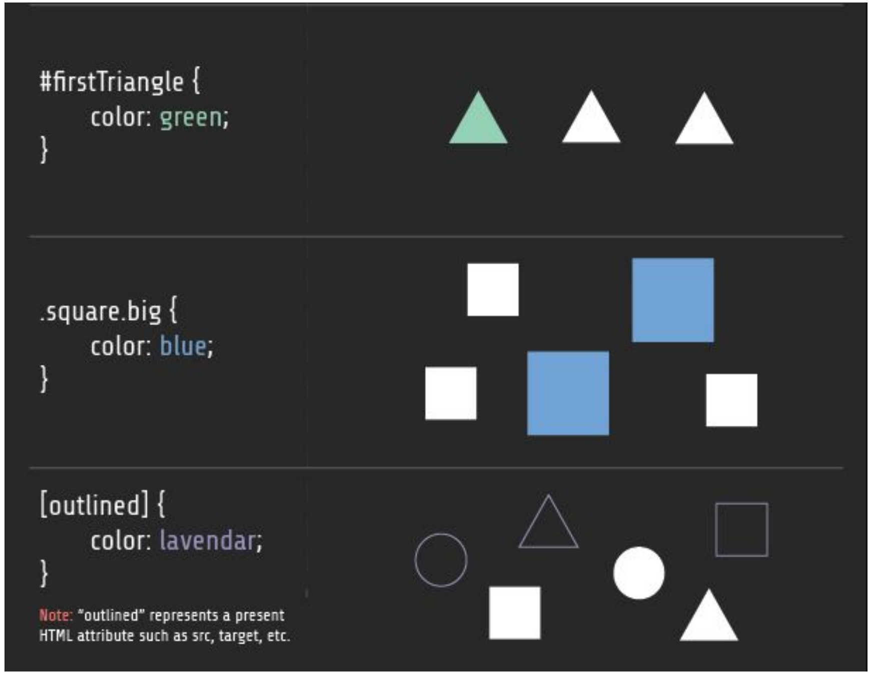


## CSS Selectors

- Recall
  - Selectors are used to target HTML elements for styling
- Types of selectors
  - Element (Tag) Selector: p, h1, div
  - Class Selector: .menu-item, .price
  - ID Selector: #menu, #submit-button



```css
* {
    color: red;
}

.circle {
    color: yellow;
}
```

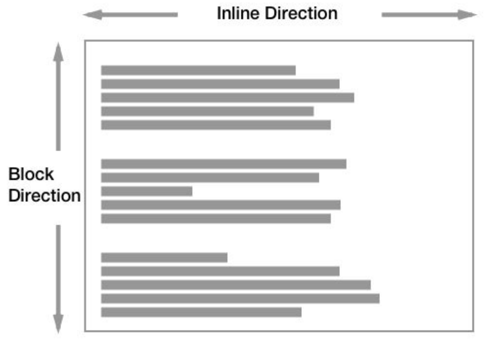


```css
#firstTriangle {
    color: green;
}

.square.big {
    color: blue;
}

[outlined] {
    color: lavendar;
}
```

Note: "outlined" represents a present HTML attribute such as src, target, etc.

# How Elements Render by Default

- Recall
  - Normal Flow is how browsers lay out elements by default
  - Block-level elements stack vertically
    - Width and height can be set
    - Can contain text , data, inline elements, or other block level elements
  - Inline elements flow horizontally
    - Width and height is determined by the content
    - Can contain text (paragraph, anchor tags, etc) or other inline elements
- Flexbox changes this normal flow



## Introduction to Flexbox

# Introduction to Flexbox

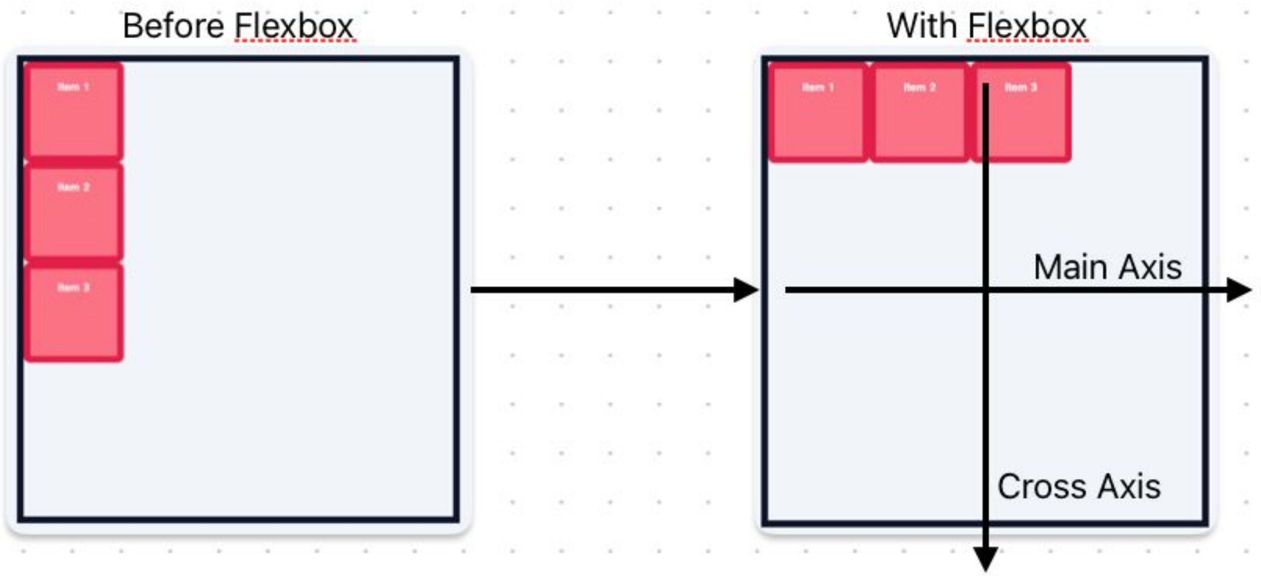
- What is flexbox?
    - Also known as flexible box layout module
    - A modern CSS layout model for organizing elements on a webpage
    - Allows browsers to display HTML elements as flexible box models
- Core Principles
    - Single Axis
        - Each flex container manages layout on a single axis (row or column) at a time
        - In fact, it cannot lay out box models in a row and column at the same time
    - Container & Items
        - A container may use flexbox items. As a result, the elements inside become flex items
    - Flexible Ordering
        - Use flexbox based properties to rearrange items without altering the HTML

## What problem did it solve?

- Easy Alignment
  - Before: Centering or aligning elements often required float hacks, negative margins, or display: table-cell
  - With Flexbox: Use properties like justify-content and align-items to easily align items horizontally or vertically
- Rigid Sizing & Distribution
  - Before: Manually juggling fixed widths, percentage widths, or complex media queries to handle changing layouts
  - With Flexbox: Harness flex-grow, flex-shrink, and flex-basis to dynamically distribute available space and respond to container resizing
- Reordering Content Without Altering HTML
  - Before: Changing element order typically meant editing the HTML source or using absolute positioning.
  - With Flexbox: The order property (and row-reverse / column-reverse) allows rearranging items purely via CSS—improving maintainability
- Built-In Responsiveness
  - Before: Maintaining multiple media queries with complex rules for different screen sizes.
  - With Flexbox: Flex items adapt automatically to container space; often fewer (or simpler) media queries are needed

# Creating a Flexbox Container

- The display: flex; property turns an element into a Flexbox container
- Direct children of this container become flex items
- The layout of these flex items is controlled by Flexbox properties
- Example:



## Setting the Main Axis

- The flex-direction property sets the main axis
  - Defines the direction flex items are placed in the container
- Values
  - row (default): Items are placed in a row (horizontally).
  - column: Items are placed in a column (vertically).
  - row-reverse: Items are placed in a row, reversed.
  - column-reverse: Items are placed in a column, reversed

## Aligning Items Along the Main Axis

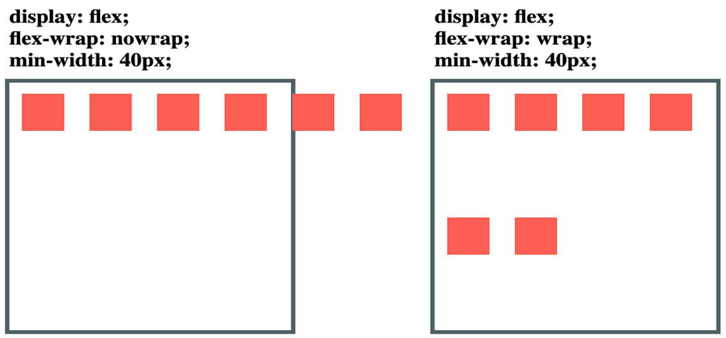
- The justify-content aligns flex items along the main axis
  - Defines the direction flex items are placed in the container
- Values
  - flex-start (default): Items are packed to the start of the main axis
  - flex-end: Items are packed to the end of the main axis
  - center: Items are centered along the main axis
  - space-between: Items are evenly distributed; the first item is at the start, the last at the end
  - space-around: Items are evenly distributed with equal space around each item
  - space-evenly: Items are evenly distributed with equal space between items, at the start, and at the end

## Aligning Items Along the Cross Axis

- The align-items aligns flex items along the *cross* axis
  - Defines the direction flex items are placed in the container
- Values
  - stretch (default): Items are stretched to fill the container's height (or width, depending on flex-direction).
  - flex-start: Items are aligned to the start of the cross axis.
  - flex-end: Items are aligned to the end of the cross axis.
  - center: Items are centered along the cross axis.
  - baseline: Items are aligned to their baselines

## Handling Overflow

- What’s overflow?
  - Situation where the content exceeds the containers height, usually set via the “height” property
  - Can be managed in CSS via the “overflow” property
- flex-wrap property controls what happens when flex items overflow the container
  - nowrap (default): Items are forced into a single line
  - wrap: Items wrap onto multiple lines
  - wrap-reverse: Items wrap onto multiple lines in reverse order


```
display: flex;
flex-wrap: nowrap;
min-width: 40px;
```

```
display: flex;
flex-wrap: wrap;
min-width: 40px;
```

## Advanced Container Control

- align-content Aligns lines of flex items when there is extra space in the cross axis and items wrap
    - When to use: Only effective when flex-wrap: wrap or flex-wrap: wrap-reverse is used and the flex container has multiple lines of items.
- flex-flow
    - A shorthand for flex-direction and flex-wrap
- Flexbox and auto margins
    - This can be used as an alignment techniques (e.g., pushing an item to the far right)
    - Example: Using margin-left: auto to push a navigation item to the right

## Let’s Code

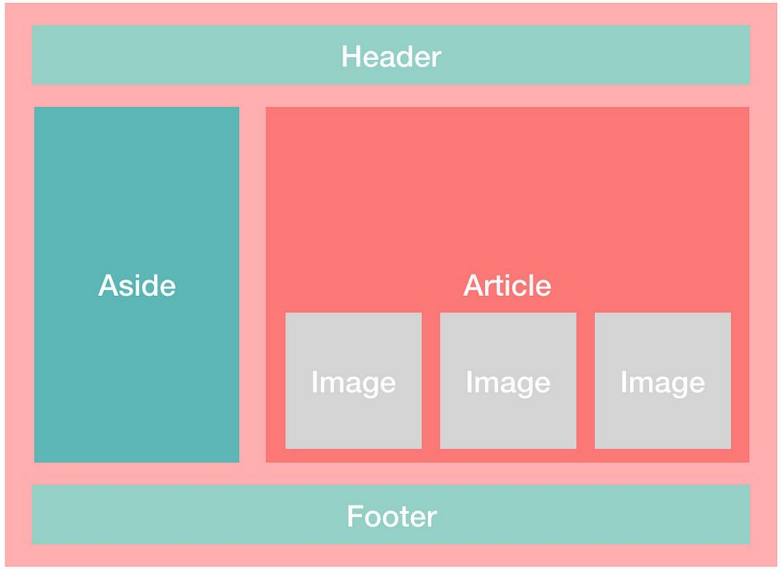
## Exercise

- I’ll share the restaurant menu code (HTML & CSS)
- Your task
    - Apply display: flex to the .menu-items container within each .menu-category
    - Try arranging the menu items in a row

## Creating Layouts with Flexbox

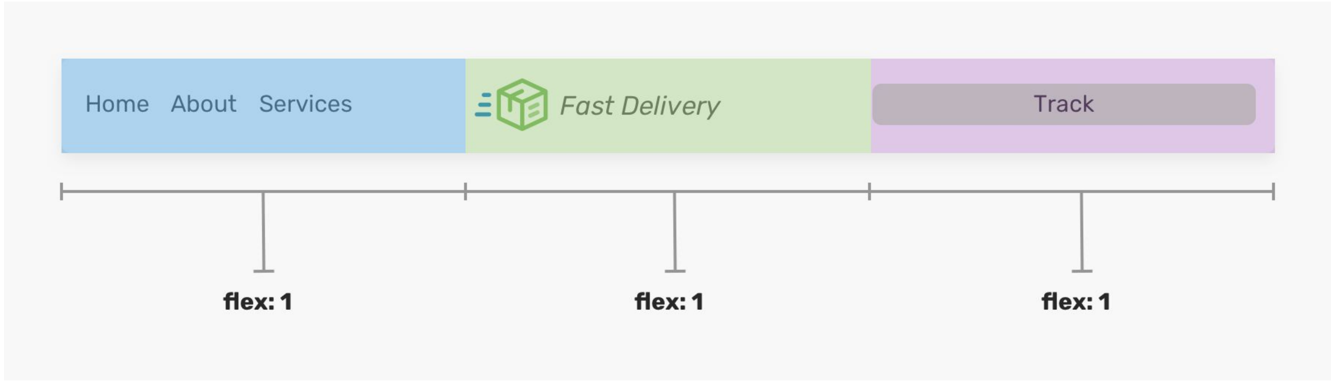
## Beyond Item Arrangement

- Flexbox is not just for small components; it can structure entire pages
- Common layout patterns
  - Header and Footer
  - Sidebar and Content
  - Grid-like structures



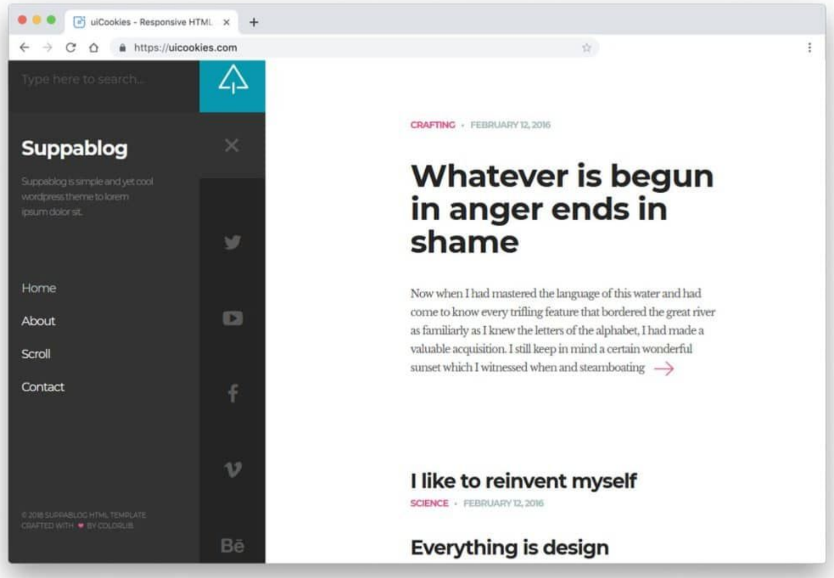
## Flexbox for Header and Footer

- Use Flexbox to:
  - Position navigation elements in the header
  - Align copyright information in the footer
  - Create a sticky footer that stays at the bottom of the page



# Flexbox for Sidebar and Content

- Use Flexbox to:
    - Create a two-column layout for the main content and sidebar
    - Control the width ratio of the sidebar and content area
    - Make the sidebar fixed or scrollable


uiCookies - Responsive HTML
https://uicookies.com
Type here to search...
Suppablog
Home
About
Scroll
Contact
© 2018 SUPPABLOG HTML TEMPLATE
CRAFTED WITH ♥ BY COLORLIB
CRAFTING · FEBRUARY 12, 2016
SCIENCE · FEBRUARY 12, 2016

Suppablog is simple and yet cool wordpress theme to lorem ipsum dolor sit.

Whatever is begun in anger ends in shame

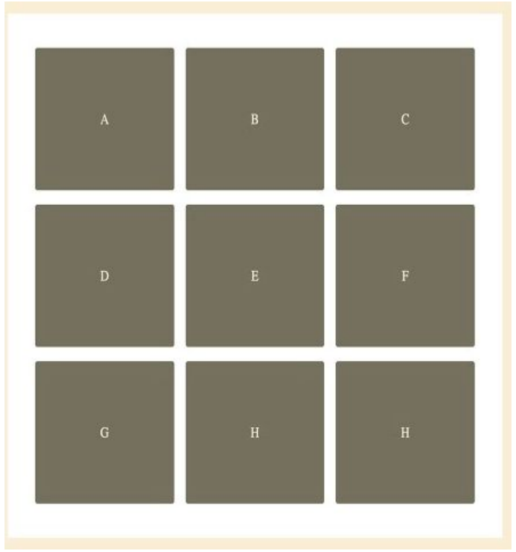
Now when I had mastered the language of this water and had come to know every trifling feature that bordered the great river as familiarly as I knew the letters of the alphabet, I had made a valuable acquisition. I still keep in mind a certain wonderful sunset which I witnessed when and steamboating

I like to reinvent myself

Everything is design

## Flexbox for Grid-like Layouts

- Flexbox can be used to create grid-like layouts
  - Note: CSS Grid is another way to layout elements and is more powerful for complex grids. We'll cover this later.
- Use flex-wrap and flex-basis to control item distribution in a "grid."
- Use Flexbox to:
  - Create a two-column layout for the main content and sidebar
  - Control the width ratio of the sidebar and content area
  - Make the sidebar fixed or scrollable



## Let’s Code

## Exercise: Restaurant Menu Layout

- Using the existing HTML and CSS
- Try to use display: flex and related properties to structure the <header> and <div class="menu"> elements
- Aim for a basic layout where the navigation is horizontal and the menu categories are arranged in columns
- Use flex-wrap and flex-basis to control item distribution in a "grid."
- Use Flexbox to:
  - Create a two-column layout for the main content and sidebar
  - Control the width ratio of the sidebar and content area
  - Make the sidebar fixed or scrollable

## Flexbox: Best Practices

- Use Flexbox for layout, not just alignment
- Understand the main axis and cross axis
- Consider flex-wrap for responsiveness
- Use flex-basis for content sizing
- Test across browsers (Chrome, Safari, etc) and screen types (desktop, mobile etc)

## Media Queries

## Introduction to Media Queries

- Media Queries allow you to apply CSS styles based on device characteristics
- Key characteristics
  - Viewport width and height
  - Device orientation (portrait or landscape)
  - Resolution (pixel density)
  - Media type (screen, print, speech)
- Essential for creating responsive designs that adapt to different screens

## How to Write Media Queries

- The @media rule is used to define Media Queries
- Basic syntax: @media (media feature) { /* CSS rules */ }
    - Common media features
        - max-width: Maximum width of the viewport.
        - min-width: Minimum width of the viewport.
        - orientation: portrait or landscape.
- Media Types
    - screen: For computer screens, tablets, smartphones, etc.
    - print: For printed documents.
    - speech: For screen readers.
    - all: For all media type
- Common media features
    - max-width: Maximum width of the viewport. Eg: @media screen and (max-width: 768px) { ... }
    - min-width: Minimum width of the viewport. Eg: @media screen and (min-width: 1024px) { ... }
    - orientation: portrait or landscape. Eg: @media screen and (orientation: portrait) { ... }

## Media Queries: Combining Features

- Combine multiple media features using logical operators:
  - and: Both conditions must be true
    - Example: @media screen and (max-width: 768px) and (orientation: portrait) { … }
  - , (comma): Either condition can be true. (OR)
    - Example: @media screen and (max-width: 480px), screen and (orientation: landscape) { … }
  - not: Negates a condition
    - Example: @media not all and (orientation: landscape) { … }

## Best Practices for Breakpoints

- Breakpoints are the screen widths where your layout changes
- Common breakpoint strategies
  - Mobile-first: Start with mobile styles and add styles for larger screens
  - Major devices: Target common screen sizes (e.g., phone, tablet, desktop)
- Common Breakpoint Examples (in pixels):
  - Small phones: 320px - 480px
  - Phones: 481px - 767px
  - Tablets: 768px - 1023px
  - Desktops: 1024px and up
- Avoid too many breakpoints

## Media Queries: Units (em vs. px)

- em units are relative to the font size of the parent element
- px units are absolute pixels
- Using em for Media Queries can create more scalable and accessible designs
  - Allows layouts to adapt if the user changes their default font size in the browser
  - Provides better consistency across devices and zoom levels
- Example: If the base font size is 16px, then 1em = 16px, 2em = 32px, etc

## Let’s Code

## Let's Code!

- Using the existing menu code:
  - Add a Media Query to change the layout when the screen width is less than 768px
- Try:
  - Making the menu categories stack vertically instead of being side-by-side.
  - Increasing the font size of the menu titles for better mobile readability."

# Media Queries: Advanced Techniques

- Using em for breakpoints
  - Add a Media Query to change the layout when the screen width is less than 768px
- srcset attribute for responsive images
  - Serve different image sizes based on screen resolution.
  - Improves page load times on smaller devices
- CSS variables with Media Queries
  - Change CSS variables at breakpoints for more dynamic styling
  - Example:
    - --main-font-size: 16px;
      - Declares a CSS variable that can set the font size for your text to 16 pixels
    - @media (max-width: 768px) { --main-font-size: 14px; }
      - For a device size less than or equal to 768 pixels, the font size is now reduced to 14 pixels. Note: this syntax isn’t complete!
    - body { font-size: var(--main-font-size); }
      - The declared variable is now applied to the HTML body

## Quiz!

## Question 1

- Which CSS property is used to enable Flexbox on a container element?
  - A) display: grid;
  - B) float: flex;
  - C) display: flex;
  - D) flex-layout: true;

Correct Answer: C

## Question 2

- Which flex-direction value arranges flex items vertically in a column?
  - A) row
  - B) column
  - C) row-reverse
  - D) vertical

Correct Answer: B

## Question 3

- Which CSS property aligns flex items along the main axis?
    - A) align-items
    - B) justify-content
    - C) align-content
    - D) place-content

Correct Answer: B

## Question 4

- What does the flex-wrap: wrap; property do?
    - A) Prevents flex items from wrapping to the next line.
    - B) Forces flex items to wrap to the next line.
    - C) Allows flex items to wrap onto multiple lines.
    - D) Reverses the order of flex items.

Correct Answer:  C

## Question 5

- Which value of the align-items property aligns flex items to the bottom of the flex container?
  - A) flex-start
  - B) flex-end
  - C) center
  - D) stretch

Correct Answer: B

## Question 6

- Which @media feature targets a maximum viewport width
  - A) min-width
  - B) max-height
  - C) max-width
  - D) min-height

Correct Answer: C

## Question 7

- Which media type is used for printed documents?
  - A) screen
  - B) all
  - C) speech
  - D) print

Correct Answer: D

## Question 8

- Which logical operator in Media Queries requires both conditions to be true?
    - A) and
    - B) or
    - C) , (comma)
    - D) not

Correct Answer: A

## Question 8

- What is a common breakpoint range for tablets?
    - A) 768px - 1023px
    - B) 481px - 767px
    - C) 320px - 480px
    - D) 1024px and up

Correct Answer: A

## Question 8

- Why are em units often preferred over px units in Media Queries?
    - A) em units are faster to render.
    - B) em units are absolute.
    - C) em units are relative to the viewport width.
    - D) em units are relative to the font size, providing better scalability and accessibility.

Correct Answer: D

## Homework: Your Menu Challenge

- Task: Design a responsive digital menu for a restaurant of your choice
- Requirements
  - Include at least 3 menu categories (e.g., Appetizers, Main Courses, Drinks)
  - For each category, display at least 4 menu items (dish name, description, price, image - optional).
  - Use Flexbox to structure the menu layout
  - Use Media Queries to make the menu adapt to different screen sizes (mobile, tablet, desktop).
- Bonus
  - Add a navigation bar to switch between categories (hint: using anchor)

## References

- Essential Resources
  - Mozilla Developer Network: CSS Media Queries and CSS Flexible Box Layout
- Additional Resources
  - CSS-Tricks: A Complete Guide to Flexbox
  - FreeCodeCamp: CSS Flexbox Guide