

# Module 2: Website Layouts

## Video 2.3: Holy Grail

### CSS Grid: Holy Grail

The "Holy Grail" layout is a classic web design concept that aims to achieve a specific type of page structure with a header, footer, and three main content columns, all of which are fluid and responsive. This layout is considered a design challenge because it requires creating a complex structure while maintaining flexibility and adaptability across different screen sizes. The goal is to create a layout that remains functional and visually pleasing regardless of the device being used to view the website.

### Components of the Holy Grail layout:

**Header:** The header is typically the top section of a webpage and often contains branding elements, navigation menus, and other essential content. It provides users with a quick overview of the website's purpose and offers easy access to various sections.

**Footer:** The footer is located at the bottom of the webpage and usually contains copyright information, contact details, links to important pages, and sometimes additional navigation options. It offers closure to the page and allows users to find crucial information without scrolling back up.

**Three Columns:** The central challenge of the Holy Grail layout lies in arranging three main content columns in a way that they are equally spaced and responsive to different screen sizes. These columns can hold various types of content, such as articles, sidebars, advertisements, or widgets.

- **Left Column:** This column is often used for secondary content, such as navigation menus, related links, or additional information. It can be narrower than the other two columns.

- **Center Column:** The center column holds the primary content of the page, such as articles, blog posts, or the main text of the website. It usually has the most significant width among the columns.
- **Right Column:** Similar to the left column, the right column is often utilized for additional content, such as advertisements, featured content, or widgets. It can also be narrower than the center column.

To achieve the Holy Grail layout, web developers traditionally relied on techniques such as floats and clearfix hacks, which could be challenging to manage and might lead to layout inconsistencies. However, with the advent of modern CSS, particularly the Flexbox and CSS Grid layout systems, creating the Holy Grail layout has become more accessible and efficient.

## HTML Structure for Holy Grail Layout



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="styles.css" />
    <!-- Link to your CSS file -->
    <title>Holy Grail Layout</title>
  </head>
  <body>
    <header>
      <!-- Header content goes here -->
      <nav>
        <ul>
          <li><a href="#">Home</a></li>
          <li><a href="#">About</a></li>
          <li><a href="#">Services</a></li>
          <li><a href="#">Contact</a></li>
        </ul>
      </nav>
    </header>
    <main>
      <div class="left-column">
        <!-- Left column content goes here -->
      </div>
      <div class="center-column">
        <!-- Center column content goes here -->
      </div>
      <div class="right-column">
        <!-- Right column content goes here -->
      </div>
    </main>
    <footer>
      <!-- Footer content goes here -->
      <p>&copy; 2024 Your Company. All rights reserved.</p>
    </footer>
  </body>
</html>
```



In this structure, we've used semantic HTML elements to represent the header, main content, and footer sections. We've also included placeholders for the actual content within each section. You would typically replace the placeholder content with your actual content, such as text, images, and other elements.

## Dividing the Layout

Let's break down the layout into its sections (header, main content, and footer) while using CSS Grid for positioning.

HTML:



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="styles.css" />
    <!-- Link to your CSS file -->
    <title>Holy Grail Layout</title>
  </head>
  <body>
    <div class="wrapper">
      <header class="header">
        <nav>
          <ul>
            <li><a href="#">Home</a></li>
            <li><a href="#">About</a></li>
            <li><a href="#">Services</a></li>
            <li><a href="#">Contact</a></li>
          </ul>
        </nav>
      </header>
      <main class="main-content">
        <div class="left-column">
          <!-- Left column content goes here -->
        </div>
        <div class="center-column">
          <!-- Center column content goes here -->
        </div>
        <div class="right-column">
          <!-- Right column content goes here -->
        </div>
      </main>
      <footer class="footer">
        <p>&copy; 2024 Your Company. All rights reserved.</p>
      </footer>
    </div>
  </body>
</html>
```

CSS:

```
/* Reset some default styles for consistency */
body,
h1,
h2,
h3,
p,
ul,
li {
    margin: 0;
    padding: 0;
}
/* Basic styling for the whole page */
.wrapper {
    display: grid;
    grid-template-rows: auto 1fr auto;
    min-height: 100vh;
}
/* Header styles */
.header {
    background-color: #333;
    color: white;
    padding: 10px;
}
/* Main content styles */
.main-content {
    display: grid;
    grid-template-columns: 1fr 3fr 1fr;
    gap: 20px;
    padding: 20px;
}
/* Column styles */
.left-column,
.center-column,
.right-column {
    background-color: #f5f5f5;
    padding: 10px;
}
```

```
/* Footer styles */
.footer {
  background-color: #333;
  color: white;
  text-align: center;
  padding: 10px;
}
```

In this example, we're using CSS Grid to define the layout structure. The wrapper element contains the header, main content, and footer sections. The `grid-template-rows` property ensures that the header and footer take up the necessary space, while the main content takes the remaining available space. Within the main content, the three columns are created using the `grid-template-columns` property.

## Unequal Column Height Challenge

Unequal column heights are a common challenge when working with multi-column layouts, such as the Holy Grail layout. This issue occurs because the content within each column varies in length, causing the columns to have different heights. The result can be an unbalanced or uneven appearance, which can negatively impact the overall design and user experience.

There are a few reasons why unequal column heights occur:

1. **Dynamic Content:** When the content within columns is generated dynamically, such as through user-generated posts, comments, or database entries, the length of the content can vary significantly. This leads to columns with different heights.
2. **Content Overflow:** If the content within a column exceeds the available space, it might overflow and cause that column to become taller than the others. This can happen with long paragraphs, images, or other media that don't fit within the designated column width.
3. **CSS Box Sizing:** The box-sizing property determines how the width and height of elements are calculated. If elements have different box-sizing settings, it can affect their overall dimensions and contribute to unequal column heights.

4. **Margins and Padding:** If you apply margins or padding to elements within columns, they can impact the overall height of the columns, especially if these margins or padding are not consistent across all columns.
5. **Floats and Clearfix Issues:** In traditional layouts using floats, clearing floated elements can sometimes lead to unintended differences in column heights. Clearing floats incorrectly can cause columns to behave unpredictably.
6. **Positioning and Overlapping:** If you're using absolute or relative positioning for certain elements within columns, they might overlap with other content, leading to differences in heights.

Addressing the issue of unequal column heights is crucial for maintaining a visually pleasing and consistent design. There are several techniques you can use to address this problem:

1. **CSS Grid or Flexbox:** Modern layout systems such as CSS Grid and Flexbox can help automatically align columns with equal heights, even if their content varies. These layout systems are responsive and handle dynamic content well.
2. **Faux Columns:** Using background images or pseudo-elements, you can create the illusion of equal column heights by applying a background to the container element that stretches the full height of the tallest column.
3. **Equalizing Content:** Adjust the content within the columns to be roughly the same length or use techniques such as truncation or ellipsis for long text content.
4. **JavaScript:** In cases where CSS solutions aren't sufficient, you can use JavaScript to dynamically adjust column heights based on the tallest column's height.

By understanding the causes of unequal column heights and employing appropriate techniques to mitigate them, you can create a more polished and balanced layout for your web design projects.



## Implicit and Explicit Grid Tracks

In CSS Grid layouts, grid tracks refer to the rows and columns that make up the grid. These tracks can be explicitly defined by the developer or generated automatically based on the content and layout requirements. The concept of implicit and explicit grid tracks plays a significant role in understanding how the grid system works and how elements are placed within it.

### Explicit Grid Tracks:

Explicit grid tracks are those rows and columns that you define explicitly in your CSS using properties such as `grid-template-rows` and `grid-template-columns`. These properties allow you to specify the size and number of tracks in your grid. For example:

```
.grid-container {  
  display: grid;  
  grid-template-rows: 100px 200px; /* Defines two rows with specific heights */  
  grid-template-columns: 1fr 2fr; /* Defines two columns with a ratio of 1:2 */  
}
```

In this example, the `grid-template-rows` and `grid-template-columns` properties explicitly define the size and number of rows and columns in the grid. Elements placed in the grid will follow these explicit tracks.

### Implicit Grid Tracks:

Implicit grid tracks are generated automatically by the browser based on the content you place within the grid. If there are more grid items than explicitly defined tracks, the browser creates additional tracks to accommodate the items. This is particularly useful when dealing with dynamic content or when you don't want to define an exact number of tracks. Implicit tracks are created based on the `grid-auto-rows` and `grid-auto-columns` properties:

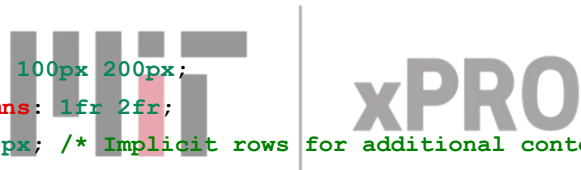
```
.grid-container {  
  display: grid;  
  grid-template-columns: 1fr 2fr;  
  grid-auto-rows: 100px; /* Defines the height for implicitly generated rows */  
}
```

In this example, the `grid-auto-rows` property specifies the height for implicitly generated rows that result from having more items than the initially defined rows. Implicit grid tracks can also be used for column sizing when `grid-template-columns` doesn't explicitly define all columns.

### Combining Explicit and Implicit Grids:

You can combine explicit and implicit grid tracks to create flexible layouts. Explicit tracks define a structured grid, while implicit tracks accommodate overflow content:

```
.grid-container {  
  display: grid;  
  grid-template-rows: 100px 200px;  
  grid-template-columns: 1fr 2fr;  
  grid-auto-rows: 100px; /* Implicit rows for additional content */  
}
```



### Using grid-auto-rows for Unequal Column Heights

Using `grid-auto-rows` can help ensure equal heights for columns in a CSS Grid layout, particularly when dealing with dynamic content that might cause unequal column heights. Here's how you can use `grid-auto-rows` to achieve equal heights for columns:

Let's consider a scenario with a Holy Grail layout where you have a header, footer, and three content columns. We'll focus on using `grid-auto-rows` to maintain equal heights for the columns.

HTML:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="styles.css" />
    <!-- Link to your CSS file -->
    <title>Equal Height Columns with CSS Grid</title>
  </head>
  <body>
    <div class="wrapper">
      <header class="header">
        <!-- Header content goes here -->
      </header>
      <main class="main-content">
        <div class="left-column">
          <!-- Left column content goes here -->
        </div>
        <div class="center-column">
          <!-- Center column content goes here -->
        </div>
        <div class="right-column">
          <!-- Right column content goes here -->
        </div>
      </main>
      <footer class="footer">
        <!-- Footer content goes here -->
      </footer>
    </div>
  </body>
</html>
```

CSS:

```
/* Reset some default styles for consistency */
body,
h1,
h2,
h3,
p,
ul,
li {
    margin: 0;
    padding: 0;
}
/* Basic styling for the whole page */
.wrapper {
    display: grid;
    grid-template-rows: auto 1fr auto;
    min-height: 100vh;
}
/* Header and footer styles */
/* Main content styles */
.main-content {
    display: grid;
    grid-template-columns: 1fr 1fr 1fr; /* Equal columns */
    gap: 20px;
    padding: 20px;
    grid-auto-rows: minmax(
        100px,
        auto
    ); /* Minimum height of 100px, expand as needed */
}
/* Column styles */
.left-column,
.center-column,
.right-column {
    background-color: #f5f5f5;
    padding: 10px;
}
```

```
/* Footer styles */  
.footer {  
  background-color: #333;  
  color: white;  
  text-align: center;  
  padding: 10px;  
}
```

In this example, the critical part is the `grid-auto-rows` property set on the `.main-content` element. This property specifies the minimum and maximum height for implicitly generated rows (i.e., rows created for items that don't fit within the explicitly defined rows).

By setting `grid-auto-rows` to `minmax(100px, auto)`, you're ensuring that the rows will have a minimum height of 100px. This helps maintain a uniform appearance for the columns, ensuring that they expand to accommodate content without drastically varying in height.

## Implementing `align-self: stretch;` to stretch column content

Using the `"align-self: stretch;"` property can help stretch column content to equal heights within a CSS Grid layout. Here's how you can implement it:

HTML:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="styles.css" />
    <!-- Link to your CSS file -->
    <title>Equal Height Columns with align-self: stretch;</title>
  </head>
  <body>
    <div class="wrapper">
      <header class="header">
        <!-- Header content goes here -->
      </header>
      <main class="main-content">
        <div class="left-column">
          <section>
            <h2>Left Column</h2>
            <!-- Left column content goes here -->
          </section>
        </div>
        <div class="center-column">
          <section>
            <h2>Center Column</h2>
            <!-- Center column content goes here -->
          </section>
        </div>
        <div class="right-column">
          <section>
            <h2>Right Column</h2>
            <!-- Right column content goes here -->
          </section>
        </div>
      </main>
      <footer class="footer">
        <!-- Footer content goes here -->
      </footer>
    </div>
  </body>
</html>
```

CSS:

```
/* Reset some default styles for consistency */
body,
h1,
h2,
h3,
p,
ul,
li {
    margin: 0;
    padding: 0;
}
/* Basic styling for the whole page */
.wrapper {
    display: grid;
    grid-template-rows: auto 1fr auto;
    min-height: 100vh;
}
/* Header and footer styles */
/* Main content styles */
.main-content {
    display: grid;
    grid-template-columns: 1fr 1fr 1fr; /* Equal columns */
    gap: 20px;
    padding: 20px;
}
/* Column styles */
.left-column,
.center-column,
.right-column {
    background-color: #f5f5f5;
    padding: 10px;
    display: flex;
    flex-direction: column; /* Ensure content stacks vertically */
    align-self: stretch; /* Stretch column to equal height */
}
```

```
/* Footer styles */  
.footer {  
  background-color: #333;  
  color: white;  
  text-align: center;  
  padding: 10px;  
}
```

In this example, the key is using `align-self: stretch;` in combination with `display: flex;` on each column element (`.left-column`, `.center-column`, `.right-column`). The `align-self: stretch;` property stretches the column to match the height of the tallest column in the row.

## Holy Grail Layout for Different Screen Sizes

The Holy Grail layout is designed to be responsive, adapting to various screen sizes and devices. This adaptability is achieved using CSS Grid, media queries, and flexible units. Here's how the Holy Grail layout can adapt to different screen sizes.

**Media Queries:** Media queries allow you to apply different styles based on the screen's width or other characteristics. By defining breakpoints and adjusting your layout accordingly, you can create a seamless transition between different screen sizes.

**Responsive Columns:** In a Holy Grail layout, the three main content columns can be adjusted using relative units such as percentages or `fr` (fractional units). This ensures that the columns proportionally adjust to the available screen width.

**Fluid Header and Footer:** The header and footer can also be designed to be responsive. You might adjust font sizes, spacing, and other elements to ensure they look balanced and legible on various screen sizes.



Here's an example of how media queries might be applied to adapt the Holy Grail layout for different screen sizes:

```
/* Adjust columns for larger screens */
@media (min-width: 768px) {
  .main-content {
    grid-template-columns: 1fr 2fr 1fr;
  }
}

/* Stack columns on smaller screens */
@media (max-width: 767px) {
  .main-content {
    grid-template-columns: 1fr;
  }
}
```

In this example, when the screen width is at least 768px, the columns retain their original layout. However, when the screen width is 767px or smaller, the columns stack vertically, creating a more user-friendly experience for narrow screens.

## The Importance of Spacing Between Columns and Rows

The spacing between columns and rows in a layout, such as the Holy Grail layout, is of paramount importance for several reasons.

**Visual Separation:** Adequate spacing helps visually separate different sections of the layout, making it easier for users to distinguish between header, content columns, and footer. This separation improves the overall readability and organization of the content.

**Readability:** White space, or negative space, between elements contributes to better readability. It prevents content from feeling crowded and overwhelming, allowing users to focus on individual elements without being distracted.

**Touch-Friendly Design:** In mobile and touch-based interfaces, sufficient spacing ensures that users can interact with elements accurately. Buttons, links, and other interactive elements should have enough padding to prevent accidental taps or clicks.

**User Experience:** Proper spacing enhances the user experience by creating a comfortable and pleasant visual flow. It avoids a cramped appearance that could lead to user frustration or confusion.

`grid-column-gap` and `grid-row-gap` are CSS properties used to create gutters, or spacing, between columns and rows in a CSS Grid layout. These properties help you control the amount of space between grid tracks, enhancing the readability and visual separation of your layout. Here's how you can use them:

```
.grid-container {  
  display: grid;  
  grid-template-columns: repeat(3, 1fr); /* Three equal columns */  
  grid-column-gap: 20px; /* Space between columns */  
  grid-row-gap: 20px; /* Space between rows */  
}
```

In this example, the `grid-column-gap` property sets the space between columns, while the `grid-row-gap` property sets the space between rows. The value `20px` can be adjusted to your desired spacing.

Alternatively, you can use the shorthand property `grid-gap` to set both column and row gaps at once:

```
.grid-container {  
  display: grid;  
  grid-template-columns: repeat(3, 1fr);  
  grid-gap: 20px; /* Sets both column and row gaps */  
}
```

## Spanning and Alignment in Grid Containers

### Spanning:

CSS Grid allows you to control the placement and size of grid items using the `grid-column` and `grid-row` properties. You can use the `span` keyword to make an item span across multiple columns or rows. This is particularly useful for creating layouts with varying column or row requirements.

### Example:

```
.grid-item {  
  grid-column: 2 / span 2; /* The item spans from column 2 to 3, occupying 2 columns */  
  grid-row: 1 / span 1; /* The item spans 1 row */  
}
```

### Alignment:

CSS Grid provides powerful alignment and justification properties to control the positioning of items within grid cells. The main alignment properties are `justify-items` and `align-items`, which control how items align along the column and row axes, respectively. There are also corresponding properties for aligning the entire grid (`justify-content` and `align-content`) and for self-alignment within individual grid items (`justify-self` and `align-self`).

### Example:

```
.grid-container {  
  display: grid;  
  justify-items: center; /* Aligns items horizontally in the center */  
  align-items: center; /* Aligns items vertically in the center */  
}
```

These CSS Grid features give you precise control over the placement, size, and alignment of grid items, enabling you to create complex and responsive layouts with ease.

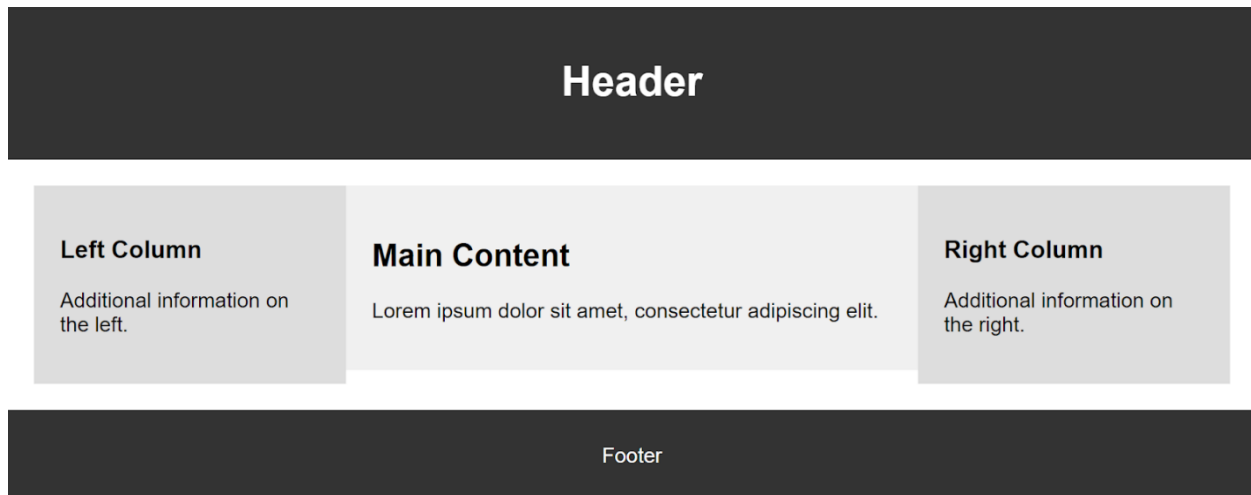
## Exercises

### Task 1: Basic Holy Grail Layout

Create a Holy Grail layout with a header, footer, and three content columns. The center column should have the main content, while the left and right columns contain additional information.

Starter Code (HTML):

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="styles.css" />
    <!-- Replace with the path to your CSS file -->
    <title>Basic Holy Grail Layout</title>
  </head>
  <body>
    <header>
      <h1>Header</h1>
    </header>
    <div class="wrapper">
      <aside class="left">
        <h3>Left Column</h3>
        <p>Additional information on the left.</p>
      </aside>
      <main>
        <h2>Main Content</h2>
        <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</p>
      </main>
      <aside class="right">
        <h3>Right Column</h3>
        <p>Additional information on the right.</p>
      </aside>
    </div>
    <footer>
      <p>Footer</p>
    </footer>
  </body>
</html>
```

**Desired Output:****Instructions:**

1. This layout is called "Holy Grail" because it's a common design pattern used in web development.
2. Start with a basic HTML structure containing a header, footer, and a container for the main content area with three columns.
3. Define a grid container for the layout. In this case, the container element wraps around the header, main content, left column, right column, and footer.
4. The properties you need to use are:
  - a. "display: grid;" turns the .container into a grid container.
  - b. "grid-template-columns" defines the column structure. In this example, we have three columns: one for the left column, one for the main content, and one for the right column. 1fr indicates that the columns should take up an equal portion of the available space.
  - c. "grid-template-rows: auto;" makes the rows automatically adjust their height based on their content.
  - d. "grid-gap" sets the spacing between grid items (columns and rows) within the container.
5. Define the styles for each grid item, which in this case are the header, main content, left column, right column, and footer.

6. The header and footer are styled with background color, text color, centered text, and padding to create the header and footer appearance.
7. "aside.left" and "aside.right" are the left and right columns. They have a background color and padding to distinguish them from the main content.
8. "main" represents the main content area, which has its own background color and padding.
9. This media query targets screens with a maximum width of 768px.
10. Inside the media query, change the grid-template-columns to 1fr, making the layout a single column for smaller screens.

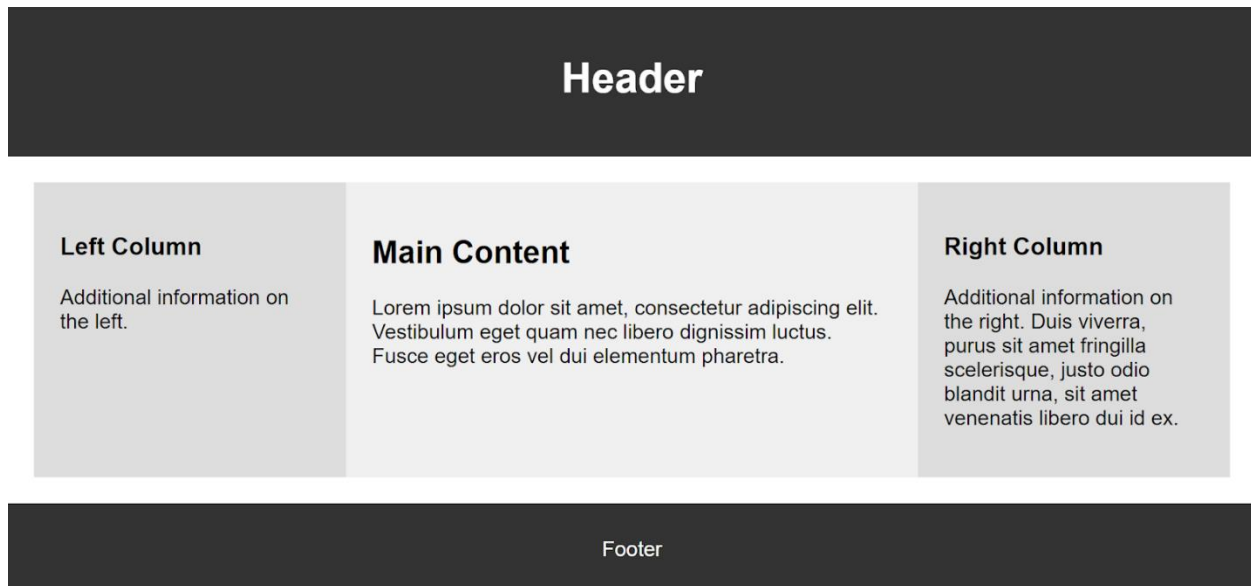
Use grid-template-areas to define the layout structure by naming the areas and assigning them to the different sections of the layout. Don't jump directly to the solution — try it out yourself first.

## Task 2: Equal Height Columns

Build a Holy Grail layout with equal-height columns, where columns adjust their height based on the tallest column's content.

Starter Code (HTML):

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="styles.css" />
    <!-- Replace with the path to your CSS file -->
    <title>Equal Height Columns</title>
  </head>
  <body>
    <header>
      <h1>Header</h1>
    </header>
    <div class="wrapper">
      <aside class="left">
        <h3>Left Column</h3>
        <p>Additional information on the left.</p>
      </aside>
      <main>
        <h2>Main Content</h2>
        <p>
          Lorem ipsum dolor sit amet, consectetur adipiscing elit.
          Vestibulum eget quam nec libero dignissim luctus. Fusce eget
          eros vel dui elementum pharetra.
        </p>
      </main>
      <aside class="right">
        <h3>Right Column</h3>
        <p>
          Additional information on the right. Duis viverra, purus sit
          amet fringilla scelerisque, justo odio blandit urna, sit
          amet venenatis libero dui id ex.
        </p>
      </aside>
    </div>
    <footer>
      <p>Footer</p>
    </footer>
  </body>
</html>
```

**Desired Output:****Instructions:**

1. Start with a basic HTML structure containing a header, footer, and a container for the main content area with three columns.
2. Define a grid container for the layout. In this case, the container element wraps around the header, main content, left column, right column, and footer.
3. The header and footer are styled with background color, text color, centered text, and padding to create the header and footer appearance.
4. Styles Required:
  - a. "grid-template-columns" defines three equal columns. The "grid-template-columns" property in CSS Grid is used to define the columns of a grid container. It specifies the size, width, and layout of each column within the grid.
  - b. "grid-gap" sets the spacing between grid items.
5. Main Content Styles:
  - a. "main" represents the main content area, which also has a background color and padding.
  - b. "grid-column" and "grid-row" properties ensure it spans the appropriate column and row.



6. In the media query for smaller screens, change the grid-template-columns to a single column layout.
7. Adjust the grid-column property for the main content to ensure it spans the first column in this case.

Use grid-auto-rows to set a minimum height for implicitly created rows. Don't jump directly to the solution — try it out yourself first.

### Task 3: Responsive Holy Grail Layout

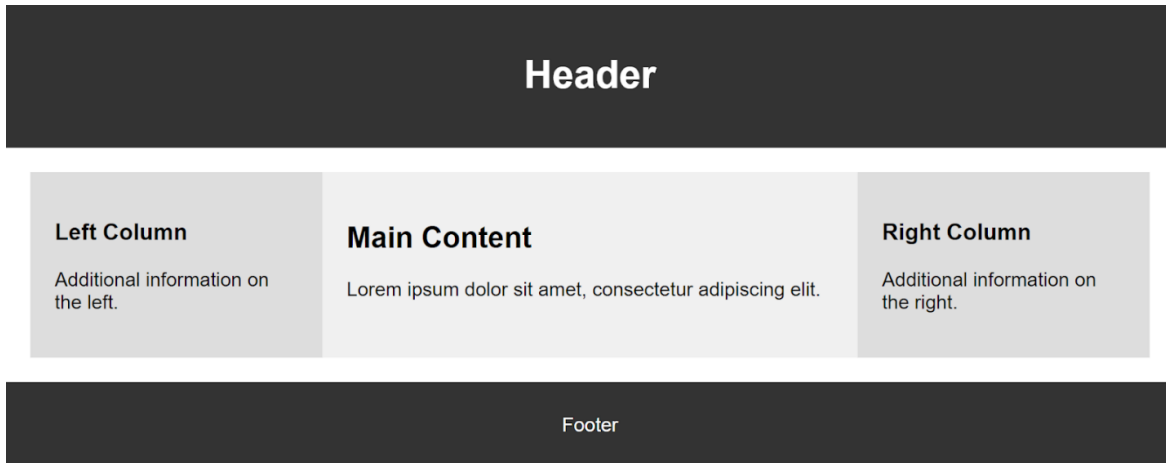
Create a Holy Grail layout that is responsive, adapting to different screen sizes using media queries.

Starter Code (HTML):



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="styles.css" />
    <!-- Replace with the path to your CSS file -->
    <title>Responsive Holy Grail Layout</title>
  </head>
  <body>
    <header>
      <h1>Header</h1>
    </header>
    <div class="wrapper">
      <aside class="left">
        <h3>Left Column</h3>
        <p>Additional information on the left.</p>
      </aside>
      <main>
        <h2>Main Content</h2>
        <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</p>
      </main>
      <aside class="right">
        <h3>Right Column</h3>
        <p>Additional information on the right.</p>
      </aside>
    </div>
    <footer>
      <p>Footer</p>
    </footer>
  </body>
</html>
```

## Desired Output:



## On Small Screens:



**Instructions:**

1. The Holy Grail layout typically consists of a header, a footer, a main content area, and two sidebars.
2. Define the overall layout using CSS Grid.
3. The `grid-template-columns` and `grid-template-rows` properties create a 3x3 grid layout with areas for the header, main content, sidebars, and footer.
4. Style the header, main content, and footer areas with padding and center-aligned text.
5. Apply styles to the sidebars, giving them a background color and padding.
6. Media Query for Screens 768px and Above: Inside this media query, make adjustments to the layout and styles to better suit screens that are 768px wide or wider.
  - a. `@media`: This is the beginning of a media query declaration.
  - b. `screen`: This specifies that these styles should only apply to screens, not print or other media types.
  - c. `and`: Combines conditions. In this case, use "and" to specify both screen type and screen width.
  - d. `(min-width: 768px)`: This is the condition for the media query. It targets screens with a minimum width of 768 pixels.
7. Change the `grid-template-columns` property to give the main content area more space, making it 2fr wide instead of 3fr.
8. Media queries ensure that the layout adapts as the screen width increases.

Use media queries to adjust the layout's structure for different screen widths. Don't jump directly to the solution — try it out yourself first.