Module 2: Website Layouts

Video 2.2: Understanding Grid Lines

CSS Grid Lines

CSS Grid is a powerful layout system in web development that enables you to create complex two-dimensional layouts with ease. It allows you to define a grid container, which is the parent element that holds a set of grid items (child elements). This system provides precise control over the placement, sizing, and alignment of these grid items within the grid, making it an excellent tool for creating responsive and dynamic layouts.

Grid Container and Grid Items:

To start using CSS Grid, you designate an element as a grid container by applying the display: grid; property to it. The children of the grid container become grid items.

Defining the Grid:

By using properties like grid-template-columns and grid-template-rows, you define the structure of the grid in terms of columns and rows. You can specify fixed sizes (e.g., pixels) for columns and rows, use relative units (e.g., percentages or fractions), or even a combination of both.

Placing Grid Items:

You control the placement of grid items within the grid using properties like grid-column and grid-row. For example, "grid-column: 2/4;" places an item in columns 2 to 3 (spanning two columns) and "grid-row: 1 / span 2;" places an item in row 1, spanning two rows.

Grid Line Numbers and Naming:

CSS Grid assigns numbers to grid lines (the lines that separate columns and rows) to make referencing them easier. You can also name specific grid lines to improve readability. For instance, you can name columns and rows like grid-template-columns: [col1] 100px [col2] 1fr [col3] and reference them using the names.

Grid Areas:

CSS Grid introduces the concept of grid areas, which are defined using the grid-template-areas property on the grid container. With grid areas, you can give names to rectangular areas of the grid and place grid items into these named areas. This approach provides a very intuitive way to design layouts, as you can visualize the entire layout structure at a glance.

Alignment and Justification:

CSS Grid provides properties such as justify-items, align-items, justify-content, and align-content to control the alignment and distribution of grid items within the grid cells. This allows for precise control over the horizontal and vertical positioning of items.

Features of CSS Grid

1. **Two-Dimensional Layouts:** Unlike many other layout systems that focus on either rows or columns, CSS Grid lets you design layouts in two dimensions simultaneously. This means that you can precisely position items both horizontally and vertically, accommodating a wide range of design requirements.

2. **Spanning and Overlapping:** You can create items that span multiple rows or columns, or even overlap other items if needed. This level of control is incredibly useful for crafting unique and creative designs.

3. **Flexible Sizing:** Grid items can have fixed sizes, be sized based on a fraction of available space, or take up as much space as their content requires (known as "auto" sizing). This flexibility accommodates various content types and design preferences.

4. **Responsive Design:** With CSS Grid, adapting layouts for different screen sizes becomes seamless. By altering the grid structure or adjusting item placements through media queries, you can create layouts that gracefully adapt to various devices and orientations.

5. **Alignment and Justification:** CSS Grid provides a comprehensive set of alignment and justification properties. You can control the alignment of items within their grid cells as well as the distribution of extra space along both axes. This ensures that your design looks polished and well proportioned.

6. **Layering and Z-Index:** While primarily a layout tool, CSS Grid can also be used in conjunction with other CSS properties to control layering and z-index, enabling more complex visual arrangements.

CSS Grid Properties

display: grid;

This property is applied to a container element to establish it as a grid container. It creates a new grid formatting context for its child elements.

grid-template-columns

This property defines the sizing and structure of columns within the grid. You can set the column sizes using various units such as pixels, percentages, fractions, or the auto keyword. For example:

```
grid-template-columns: 1fr 2fr 1fr;
```

grid-template-rows

Similar to grid-template-columns, this property defines the sizing and structure of rows within the grid. It follows the same syntax and units. For example: `grid-template-rows: 100px auto 50px;`

grid-template-areas

This property assigns names to grid areas, creating a visual representation of the layout. It allows you to place items by referring to the named areas. For example:

```
grid-template-areas:
  "header header"
  "sidebar main"
  "footer footer";
```

grid-column and grid-row

These properties specify where an item should start and end within the grid columns and rows, respectively. You can use line numbers or named lines to define their positions. For example:

```
grid-column: 2 / 4;
grid-row: 1 / span 2;
```

These properties, among others provided by CSS Grid, give you the power to define, structure, and arrange your layout items in a precise and controlled manner. By using them effectively, you can create intricate and visually appealing grid-based designs for your web applications.

Example of CSS Grid:

HTML Code (index.html):

```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>Interactive Form Exercise</title>
        <link rel="stylesheet" href="styles.css" />
    </head>
    <body>
        <div class="grid-container">
            <div class="item">1</div>
            <div class="item">2</div>
            <div class="item">3</div>
            <div class="item">4</div>
            <div class="item">5</div>
            <div class="item">6</div>
        </div>
        <script src="script.js"></script>
    </body>
</html>
```

CSS Code (styles.css):

```css
.grid-container {
    display: grid;
    grid-template-columns: repeat(
        3,
        100px
    ); /* Three columns, each 100px wide */
    grid-template-rows: 150px 150px; /* Two rows, each 150px tall */
    gap: 10px; /* Gap between grid items */
}
.item {
    background-color: #3498db;
    color: white;
    font-size: 24px;
    display: flex;
    align-items: center;
    justify-content: center;
}
```

In this example, you have a grid container with three columns, each 100 pixels wide, and two rows, each 150 pixels tall. The gap property adds a 10px gap between grid items. The grid items with class "item" are placed within this grid. They have a blue background color, white text, and are centered both vertically and horizontally using Flexbox properties.

Grid Areas

Grid areas are a powerful feature in CSS Grid that allow you to define named regions within the grid layout. These named regions can then be used to place grid items, making the process of creating complex layouts more intuitive and readable.

Defining Grid Areas:

To define grid areas, you use the grid-template-areas property within the grid container's CSS. This property accepts a grid template represented as a string containing space-separated values. Each value corresponds to a named area.

Naming Grid Areas:

The names you use for the grid areas can be chosen freely, but they must be unique within the grid template. Names are typically represented as strings. You can assign the same name to multiple cells if you want them to share the same area.

Creating the Grid Layout:

The grid template defined with grid-template-areas should represent the structure of your grid layout, indicating where you want specific named areas to be placed. It uses line breaks to indicate row changes and periods (.) to indicate empty cells or cells that are not part of any named area.

Placing Grid Items Using Area Names:

Once you've defined the grid template areas, you can use these area names to place grid items within the layout. To do this, you apply the grid-area property to the grid items and set its value to the name of the desired grid area.

The following is an example of using a grid:

HTML Code (index.html):

```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>Interactive Form Exercise</title>
        <link rel="stylesheet" href="styles.css" />
    </head>
    <body>
        <div class="grid-container">
            <div class="item a">A</div>
            <div class="item b">B</div>
            <div class="item c">C</div>
            <div class="item d">D</div>
        </div>
        <script src="script.js"></script>
    </body>
</html>
```

CSS Code (styles.css):

```
.grid-container {
    display: grid;
    grid-template-columns: 1fr 1fr;
    grid-template-rows: auto;
    grid-template-areas:
        "a b"
        "c d";
    gap: 10px;
}
.item {
    background-color: #3498db;
    color: white;
    font-size: 24px;
    display: flex;
    align-items: center;
    justify-content: center;
}
.a {
    grid-area: a;
}
.b {
    grid-area: b;
}
.c {
    grid-area: c;
}
.d {
    grid-area: d;
}
```

In this example, you have a 2x2 grid defined by grid-template-areas. The grid items are then placed within the specified areas using the grid-area property. The layout is structured in a way that "A" occupies the top-left area, "B" the top-right area, "C" the bottom-left area, and "D" the bottom-right area.

How Grid Areas Are used to target specific sections of the layout?

Named grid areas are incredibly useful for targeting specific sections of a CSS Grid layout. They provide a more intuitive and readable way to define and manipulate the placement of grid items within the layout.

Defining Named Grid Areas:

In the CSS of the grid container, you define named grid areas using the grid-template-areas property. You lay out the grid template by representing the areas with strings, using names you've assigned to those areas.

```
.grid-container {
    display: grid;
    grid-template-columns: 1fr 1fr;
    grid-template-areas:
        "header header"
        "sidebar main"
        "footer footer";
}
```

Placing Grid Items with Named Areas:

Once you have defined the named grid areas, you can place grid items within these areas using the grid-area property applied to each grid item. Use the names you defined earlier to specify where each item should be placed.

```
.item-header {
    grid-area: header;
}
.item-sidebar {
    grid-area: sidebar;
}
.item-main {
    grid-area: main;
}
.item-footer {
    grid-area: footer;
}
```

Benefits of Using Named Grid Areas:

- **Readability:** Using area names instead of line numbers for item placement makes the layout code more readable and self-explanatory.
- **Maintainability:** Changes to the layout structure can be easily made by adjusting the grid template areas, without needing to update every item's placement explicitly.
- **Flexibility:** You can reorder and rearrange the areas, and the grid items will automatically adjust their positions accordingly.
- **Communication:** Named areas provide a clear way to communicate layout intentions to other developers or designers.

Responsive Design:

Named grid areas also work well with responsive design. As the grid adjusts based on media queries or viewport sizes, the placement of items in named areas remains intact, making it easier to maintain the layout's intended structure across different screen sizes.

Giving Styles to Specific Areas

Applying background colors, borders, and other styles to specific areas in a CSS Grid layout can be achieved using the named grid areas concept.

Define Named Grid Areas:

In your CSS, define the named grid areas using the grid-template-areas property. Each name corresponds to a specific section of your layout.

```
.grid-container {
    display: grid;
    grid-template-columns: 1fr 2fr;
    grid-template-areas: "sidebar main";
}
```

Placing Grid Items with Named Areas:

Place grid items within the named areas using the grid-area property on each item.

```
.item-sidebar {
    grid-area: sidebar;
}
.item-main {
    grid-area: main;
}
```

Styling Different Sections:

Apply styles to the named areas in your CSS to achieve the desired visual effects.

```
.grid-container {
    /* Set background colors and borders for specific areas */
    grid-template-areas: "sidebar main";
    border: 1px solid #ccc;
    padding: 10px;
}
.item-sidebar {
    /* Sidebar styles */
    background-color: #f1c40f;
    padding: 20px;
}
.item-main {
    /* Main content styles */
    background-color: #3498db;
    color: white;
    padding: 20px;
}
```

Responsive Styling:

Utilize media queries to adjust the styles based on different screen sizes or orientations.

```
@media (max-width: 768px) {
    .grid-container {
        grid-template-areas:
            "main"
            "sidebar";
    }
}
```

By following these steps, you can apply background colors, borders, padding, and other styles to specific sections of your CSS Grid layout.

Using Media Queries with Grid Layout

Using media queries to adjust styles based on screen size is a crucial aspect of responsive web design. CSS Grid's flexibility makes it an excellent tool for adapting grid layout and styling for different devices. Here's how you can achieve this:

Define Base Grid Layout:

Begin by defining the base grid layout that serves as the foundation for your design. This layout should work well on larger screens.

```css
.grid-container {
    display: grid;
    grid-template-columns: 1fr 2fr;
    grid-template-areas: "sidebar main";
    gap: 20px;
}
.item-sidebar {
    grid-area: sidebar;
}
.item-main {
    grid-area: main;
}
```

Apply Media Queries:

Use media queries to adjust the layout and styles for smaller screens. In this example, we'll make changes for screens with a maximum width of 768px.

```
@media (max-width: 768px) {
    .grid-container {
        grid-template-columns: 1fr;
        grid-template-areas:
            "main"
            "sidebar";
    }
    .item-sidebar,
    .item-main {
        /* Adjust individual item styles */
        padding: 10px;
    }
}
```

Adjust Styling and Layout:
Within the media query, you can modify individual item styles, grid template areas, and other properties to suit the smaller screen size. The example above rearranges the grid areas and reduces padding for the grid items.

By using media queries, you can adapt the grid layout and styling to different devices and screen sizes. CSS Grid's ability to handle complex layouts and its flexible nature make it a great choice for responsive design. This ensures that your web pages look and function well across a variety of devices, from large desktop monitors to tablets and mobile phones.

Reusability and Maintainability of Grid Layouts

Reusability:
Named grid areas promote reusability by abstracting layout details into named regions. Here's how it helps:

- **Consistent Naming:** By naming layout sections descriptively (e.g., "header," "sidebar," "main"), you create a standardized vocabulary for your layout structure. This consistent naming makes it easier to understand and reuse the layout in different parts of your website.
- **Component-Based Design:** Named areas are especially useful when creating reusable components. You can encapsulate the grid structure and named areas within a component, allowing you to drop the component into various parts of your application while maintaining a consistent layout structure.
- **Scaling and Maintenance:** As your project grows, reusing named grid areas helps maintain a consistent visual identity and layout across different pages or sections. You don't need to recreate the entire layout every time; you can just reuse the named areas and adjust content as needed.

Maintainability:

Named grid areas contribute to better maintainability by simplifying layout adjustments and updates:

- **Separation of Concerns:** By abstracting layout details into named areas, you separate the concerns of layout from other styling concerns. This isolation makes it easier to focus on specific areas of your design without affecting the overall layout structure.
- **Layout Changes:** If you need to modify the layout, you can update the grid template areas in one place. This change will automatically propagate across all items that use those named areas. This approach significantly reduces the chances of introducing errors during updates.
- **Collaboration:** Named areas provide a clear visual representation of your layout's structure. This visualization improves collaboration among designers and developers, making it easier to discuss and iterate on layout adjustments.
- **Responsive Design:** When adapting layouts for different screen sizes, adjusting the grid template areas in media queries maintains a consistent structure while allowing individual items to adapt to the new layout.

Writing Semantic HTML with Grid Layouts

Semantic HTML elements carry meaning, making it easier for assistive technologies like screen readers to understand and convey the content's structure. When you use named grid areas in conjunction with semantic elements, you create a more accessible experience for all users. If you have a layout area named "main," using the <main> element for that section adds semantic value, helping screen readers identify the primary content of the page.

They inherently communicate the purpose of different sections on your webpage. Combining these tags with named grid areas provides a clear and intuitive understanding of the layout's structure. If you have a section named "header," using the <header> element not only represents the layout area but also communicates the role of that section in the overall design.

These elements provide a natural context for applying styles. When paired with named grid areas, your CSS becomes more semantic and self-explanatory. For example, if you use a <nav> element with the named grid area "navigation," it's immediately clear which area the navigation styles correspond to in your CSS.

Some Naming Conventions for Grid Layouts

Use Descriptive and Intuitive Names:

Choose names that clearly indicate the purpose or content of each grid area. The names should be intuitive and immediately understandable, even to someone new to the project.

```
grid-template-areas:
  "header header"
  "sidebar main"
  "footer footer";
```

Use Hyphen or Underscore Separators:

To improve readability, consider using hyphens or underscores to separate words within your area names.

```
grid-template-areas:
  "header header"
  "side-bar main"
  "footer footer";
```

Reflect Layout Structure:

If your layout follows a certain structure, consider reflecting that in your naming convention. For instance, if you have multiple rows, you can include row numbers in the naming.

```
grid-template-areas:
  "header-1 header-2"
  "sidebar-1 main-1"
  "footer-1 footer-2";
```

Exercises:

Task 1: Basic Grid Layout

- Create a basic grid layout with three rows and three columns.
- Apply different background colors to each grid cell.

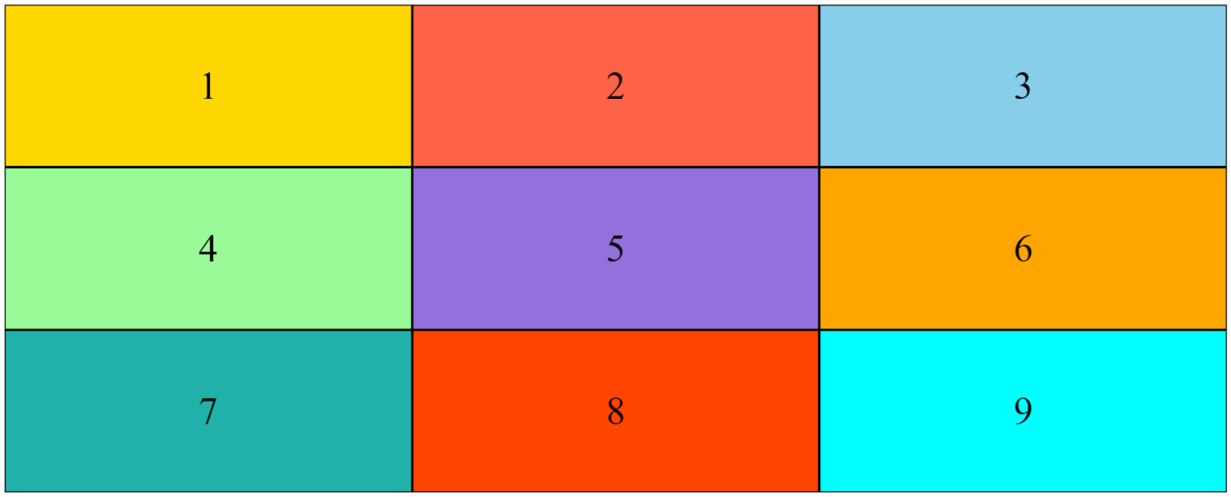
Starter Code (HTML):

```
<!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" />
        <title>Grid Exercise 1</title>
    </head>
    <body>
        <div class="grid-container">
            <div class="item">1</div>
            <div class="item">2</div>
            <div class="item">3</div>
            <div class="item">4</div>
            <div class="item">5</div>
            <div class="item">6</div>
            <div class="item">7</div>
            <div class="item">8</div>
            <div class="item">9</div>
        </div>
    </body>
</html>
```

Starter Code (styles.css):

```
.grid-container {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    grid-template-rows: repeat(3, 100px);
}
.item {
    border: 1px solid black;
}
```

Desired Output:



Instructions:

1. Start by creating an HTML file. You can name it something like "grid-layout.html."
2. In your HTML file, create a structure for the grid layout using the <div> element. Refer to the starter code.
3. Have a grid-container div that contains nine grid-item divs. This represents a 3x3 grid layout.
4. Create a CSS file named "styles.css" (or any other name you prefer) in the same directory as your HTML file.
5. Use the "display: grid property" to create a grid container with three columns and three rows.
6. Set the display property of the grid-container element to grid. This will tell the browser to use CSS grid layout for this element.
7. Set the grid-template-columns property of the grid-container element to repeat(3, 1fr). This will create three columns of equal width.
8. Set the grid-template-rows property of the grid-container element to repeat(3, 1fr). This will create three rows of equal height.
9. Each grid item (.grid-item) should be given a different background color using nth-child selectors.

10. Also set common styles for all grid items to make them take up the full width and height of their container.

Start by understanding the basic structure of CSS Grid and how it works. Break down the layout into smaller components and build step by step. Don't jump directly to the solution — try it out yourself first.

Task 2: Responsive Grid Layout

- Create a responsive grid layout with four columns. The layout should change to two columns on smaller screens.
- Apply different background colors to even and odd grid cells.

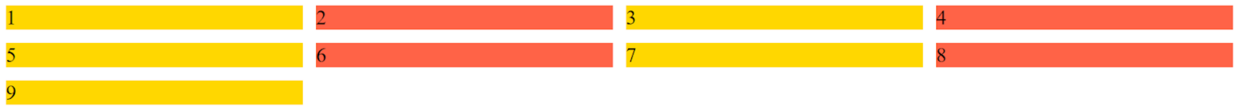
Starter Code (HTML):

```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" />
        <title>Grid Exercise 1</title>
    </head>
    <body>
        <div class="grid-container">
            <div class="item">1</div>
            <div class="item">2</div>
            <div class="item">3</div>
            <div class="item">4</div>
            <div class="item">5</div>
            <div class="item">6</div>
            <div class="item">7</div>
            <div class="item">8</div>
            <div class="item">9</div>
        </div>
    </body>
</html>
```

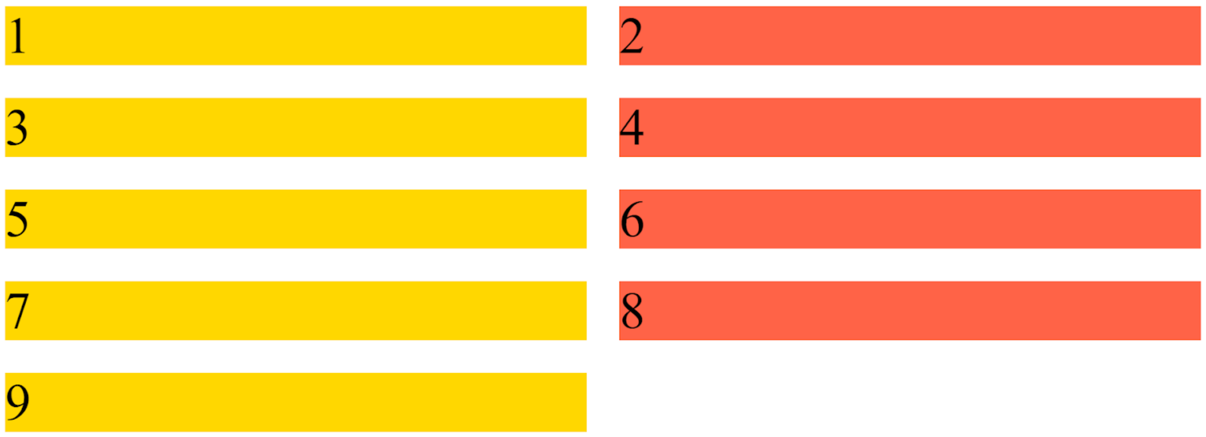
Starter Code (CSS):

```
.grid-container {
    display: grid;
    grid-template-columns: repeat(4, 1fr);
    grid-gap: 10px;
}
.item {
    border: 1px solid black;
}
```

Desired Output:



On Smaller Screen:



Instructions:

1. Use CSS Grid for the layout.
2. Set the container's display property to grid.
3. display: grid: This tells the browser to use CSS grid layout for the .grid-container element.

4. Define the number of columns for both large screens (four columns) and smaller screens (two columns).
5. grid-template-columns: repeat(4, 1fr): This creates four columns of equal width in the .grid-container element.
6. Adjust the gap between grid items as needed for spacing.
7. Implement media queries to make the grid responsive.
8. Set a breakpoint at the desired screen width where the layout switches from four columns to two columns.
9. @media (max-width: 992px) {}: This is a media query that applies the following CSS rules when the screen width is less than 992px.
    a. .grid-container { grid-template-columns: repeat(2, 1fr) }: This changes the number of columns in the .grid-container element to two when the screen width is less than 992px.
10. Apply different background colors to even and odd grid cells.
11. Utilize the :nth-child pseudo-class selector to target alternating grid items.
12. .grid-item:nth-child(even) { background-color: red }: This applies a red background color to all even grid items in the .grid-container element.
13. .grid-item:nth-child(odd) { background-color: green }: This applies a green background color to all odd grid items in the .grid-container element.

Start by understanding the basic structure of CSS Grid and how it works. Break down the layout into smaller components and build step by step. Don't jump directly to the solution — try it out yourself first.

Task 3: Image Gallery Grid

- Create an image gallery grid with a dynamic number of columns based on the screen width. The images should maintain their aspect ratio and be centered within each cell.

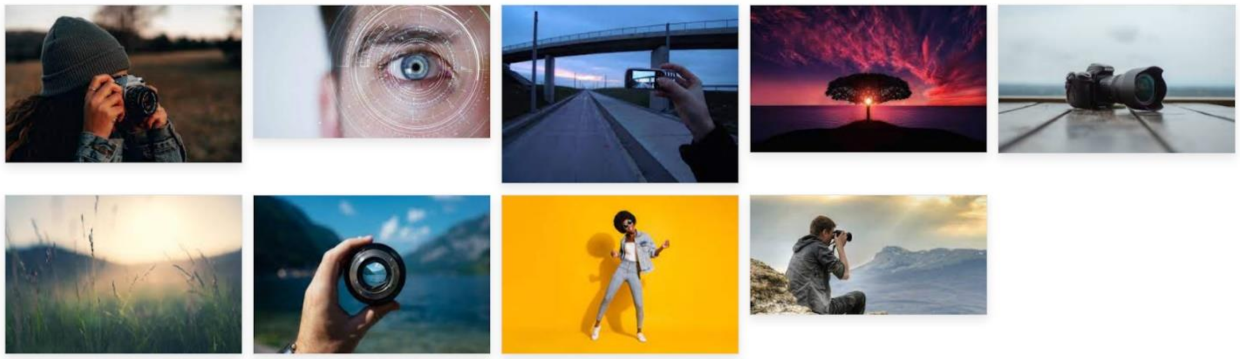
Starter Code (HTML):

```
<!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" />
        <title>Grid Exercise 1</title>
    </head>
    <body>
        <div class="grid-container">
            <div class="item">1</div>
            <div class="item">2</div>
            <div class="item">3</div>
            <div class="item">4</div>
            <div class="item">5</div>
            <div class="item">6</div>
            <div class="item">7</div>
            <div class="item">8</div>
            <div class="item">9</div>
        </div>
    </body>
</html>
```

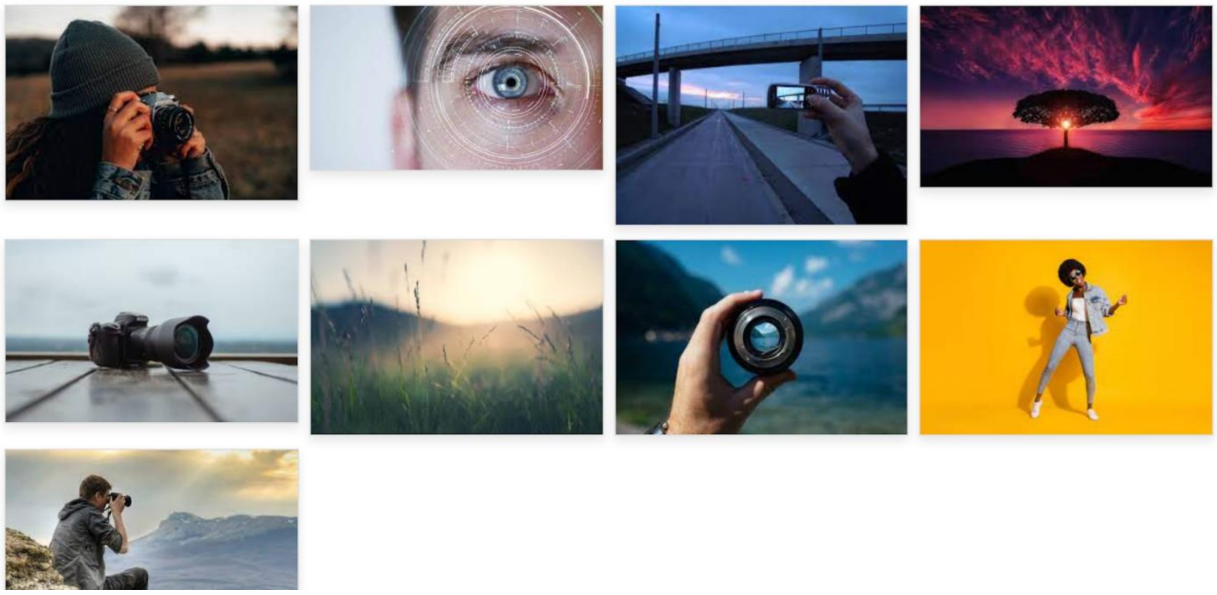
Starter Code (CSS):

```
.grid-container {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
    grid-gap: 10px;
}
.item img {
    max-width: 100%;
    height: auto;
    display: block;
    margin: 0 auto;
}
```

Desired Output:



Upon Zoom:



Replace the numbers in the div with class=”item” tag with img tags.

Instructions:

1. Use CSS Grid to create a flexible grid layout that adjusts to the screen width.
2. Define the number of columns you want for your grid.

3. You can use the repeat(auto-fit, minmax(200px, 1fr)) syntax to create a responsive grid where columns are at least 200px wide but adapt to the available space.
4. To maintain the aspect ratio of images and center them within each cell, apply the following styles to the images:

```
.gallery-item img {
    max-width: 100%;
    height: auto;
    display: block;
    margin: 0 auto; /* Center the image horizontally */
  }
```

5. Your grid will now dynamically adjust the number of columns based on the screen width while maintaining image aspect ratios and centering them within each cell.
6. Adjust the minmax value in the grid-template-columns property to set the minimum and maximum width for your columns as needed.

Start by understanding the basic structure of CSS Grid and how it works. Break down the layout into smaller components and build step by step. Don't jump directly to the solution — try it out yourself first.