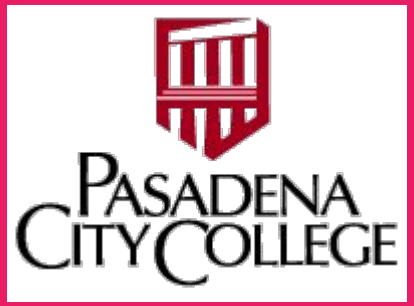



# CIS 192

## Intro to Website Development

Module 6

# Flexbox

# Flexbox

```
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>
```

Before the Flexbox Layout module, there were four layout modes:

- **Block**, for sections in a webpage
- **Inline**, for text
- **Table**, for two-dimensional table data
- **Positioned**, for explicit position of an element

The Flexible Box Layout Module, makes it easier to design flexible responsive layout structure without using float or positioning.

Flexbox is a CSS layout mode that provides an efficient way to lay out elements in a container so the **elements behave predictably** when the **container is resized** or viewed on **different screen sizes**.

# Flexbox introduced

**Flexbox**, or the Flexible Box Layout, is a powerful layout model in CSS that allows you to design flexible and responsive layouts more easily than traditional methods. It provides a way to arrange and distribute space among items within a container, even when their size is unknown or dynamic. Here's a breakdown of how Flexbox works:

- **Container and Items**: In Flexbox, you have a container element (parent) and its child elements (items) that you want to layout. The container is designated as a flex container by applying **`display: flex;`** or **`display: inline-flex;`** to it.
- **Main and Cross Axes**: Flexbox introduces two axes: the main axis and the cross axis. The main axis is defined by the **`flex-direction`** property, which can be set to **`row`**, **`row-reverse`**, **`column`**, or **`column-reverse`**. The cross axis is perpendicular to the main axis.

# Flex container

- Flex Container Properties:
  - **display**: Defines a flex container.
  - **flex-direction**: Establishes the main axis direction. This property defines in which direction the container wants to stack the flex items.
  - **flex-wrap**: Controls whether items are forced onto a single line or can wrap onto multiple lines.
    - The **flex-flow** property is a shorthand property for setting both the **flex-direction** and **flex-wrap** properties.
  - **justify-content:** Aligns the flex items along the main axis.
  - **align-items**: Aligns items along the cross axis.
  - **align-content**: Aligns content lines along the cross axis.

```
.flex-container {
  display: flex;
  height: 300px;
  justify-content: center;
  align-items: center;
}
```

# Flex Items

```
<div class="flex-container">
  <div>1</div>
  <div style="align-self: flex-start">2</div>
  <div style="align-self: flex-end">3</div>
  <div>4</div>
</div>
```

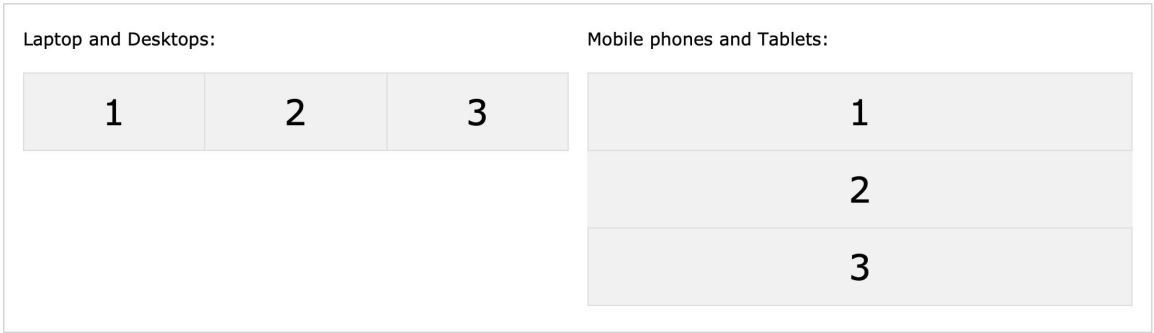
- Child Elements (Items)
  - The direct child elements of a flex container automatically becomes flexible (flex) items.
    - Styling for this is **inline on the html page**
- Specific Styling elements
  - **order:** Specifies the order of the flex item inside the same container.
  - **Align-self** Specifies the alignment for a flex item (overriding the flex container's align-items property)
  - **Flex** A shorthand property for the flex-grow, flex-shrink, and the flex-basis properties
    - **Flex-basis** Specifies the initial size of a flex item
    - **Flex-grow** Specifies how much a flex item will grow relative to the rest of the flex items inside the same container
    - **Flex-shrink** Specifies how much a flex item will shrink relative to the rest of the flex items inside the same container

# Flexbox Responsiveness

- You can use media queries to create different layouts for different screen sizes and devices.
  - For example, if you want to create a two-column layout for most screen sizes, and a one-column layout for small screen sizes (such as phones and tablets), you can change the flex-direction from row to column at a specific breakpoint (800px in the example below):
  - Another way is to change the percentage of the flex property of the flex items to create different layouts for different screen sizes. Note that we also have to include flex-wrap: wrap; on the flex container for this example to work:

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

/* Responsive layout - makes a one column
@media (max-width: 800px) {
  .flex-container {
    flex-direction: column;
  }
}
```



# Let’s do an example!

---

- Let’s do an example demonstrating a simple website layout using Flexbox.
  - The **.container** div uses **display: flex** to create a flex container.
  - The flex items inside (**.item**) are set to grow and shrink with **flex: 1**, and a **basis of 200px** in this case, but they can change size depending on available space.
  - The **justify-content: space-between** property evenly distributes the items along the main axis with space between them.
  - The **.header** and **.footer** divs are styled similarly, with a different background color and centered text.
  - Additionally, there's a **media** query included to adjust the layout for smaller screens.

# Grid Layout

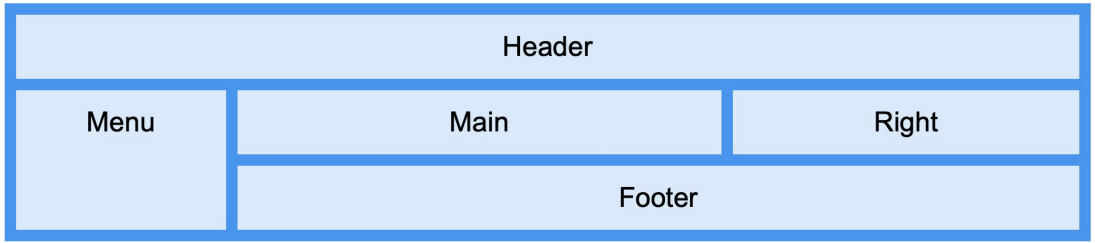
# Intro to Grid Layout

CSS Grid Layout is a powerful two-dimensional layout system that allows you to create complex grid-based layouts with ease. It provides a way to design both rows and columns in a flexible and responsive manner. Here's an overview of how CSS Grid Layout works:

- **Grid Container and Items**: Similar to Flexbox, in CSS Grid Layout, you have a grid container (parent) and its child elements (items) that you want to layout. The container is designated as a grid container by applying **display: grid**; to it.
- **Grid Lines and Track**s: CSS Grid Layout introduces the concept of grid lines, which are imaginary lines that divide the grid container into rows and columns.
    - These lines create tracks, which are the spaces between the grid lines where grid items are placed. You can define the size of these tracks using various length units like pixels, percentages, or fractions of available space.
- **Grid Rows and Columns**: You can define the number and size of rows and columns in the grid using properties like **grid-template-rows** and **grid-template-columns**. These properties allow you to specify the size of each track explicitly or use keywords like **auto**, **fr** (fractional unit), or **minmax**() to create flexible layouts.
- **Grid Item Placement**: Grid items can be placed onto the grid using line-based placement or grid area placement.
    - With line-based placement, you can **specify the start and end grid lines for both rows and columns where the item should be placed**.
    - With grid area placement, you can define named g**rid areas within the grid container and place items into these areas using the grid-area property.**
- **Grid Gap**: You can define the spacing between grid items using the grid-gap property, which sets the gap between rows and columns uniformly.
- **Alignment and Sizing:** CSS Grid Layout provides various properties for aligning and sizing grid items and the grid itself. Properties like **justify-items, align-items, justify-content**, and **align-content** allow you to control the alignment of items within their grid cells and the alignment of the grid within its container.
- **Responsive Design:** CSS Grid Layout is inherently responsive, allowing you to create layouts that adapt to different screen sizes and orientations. You can use media queries to change the grid structure or item placement based on viewport dimensions.

# Grid

The CSS Grid Layout Module offers a grid-based layout system, with rows and columns, making it easier to design web pages without having to use floats and positioning.




- The vertical lines of grid items are called columns.
- The horizontal lines of grid items are called rows.
- The spaces between each column/row are called gaps.
- The lines between columns are called column lines.
- The lines between rows are called row lines.

# Grid Container properties

To make an HTML element behave as a grid container, you have to set the display property to **grid** or **inline-grid**.

Grid containers consist of grid items, placed inside columns and rows

Sampling of Grid container level properties

- **Gap** A shorthand property for the row-gap and the column-gap properties
  - **Column-gap** Specifies the gap between the columns
  - **Row-gap** Specifies the gap between the grid rows
- **Grid** A shorthand property for the grid-template-rows, grid-template-columns, grid-template-areas, grid-auto-rows, grid-auto-columns, and the grid-auto-flowproperties
  - **Grid-area** Either specifies a name for the grid item, or this property is a shorthand property for the grid-row-start, grid-column-start, grid-row-end, and grid-column-end properties
  - **Grid-row** A shorthand property for the grid-row-start and the grid-row-end properties
  - **Grid-template** A shorthand property for the grid-template-rows, grid-template-columns and grid-areas properties
    - **grid-template-row** Specifies the size of the rows in a grid layout

# Grid Items

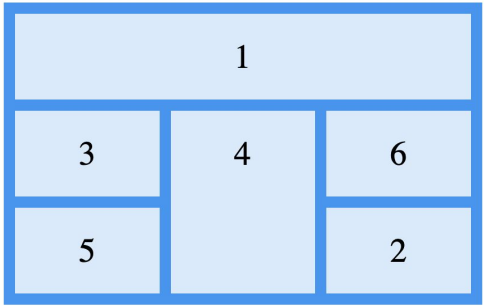
- A grid container contains grid items.
  - By default, a container has one grid item for each column, in each row, but you can style the grid items so that they will span multiple columns and/or rows.
- The **grid-column** property defines on which column(s) to place an item.
  - You define where the item will start, and where the item will end.
- The **grid-column** property defines on which column(s) to place an item.
  - You define where the item will start, and where the item will end.
- The **grid-area** property can be used as a shorthand property for the grid-row-start, grid-column-start, grid-row-end and the grid-column-end properties.

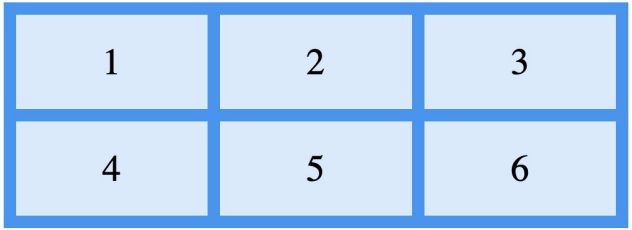
```
@media only screen and (max-width: 500px) {
  .item1 { grid-area: 1 / span 3 / 2 / 4; }
  .item2 { grid-area: 3 / 3 / 4 / 4; }
  .item3 { grid-area: 2 / 1 / 3 / 2; }
  .item4 { grid-area: 2 / 2 / span 2 / 3; }
  .item5 { grid-area: 3 / 1 / 4 / 2; }
  .item6 { grid-area: 2 / 3 / 3 / 4; }
}
</style>
</head>
<body>

<h1>Re-arange the Order on Small Devices</h1>

<p>Resize the window to 500 pixels see the effect.</p>

<div class="grid-container">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
  <div class="item4">4</div>
  <div class="item5">5</div>
  <div class="item6">6</div>
</div>
```



# Now let’s redo that example as a CSS Grid Layout

———

- The **.container** div is set as a grid container using **display: grid**.
- **grid-template-columns** defines the number and size of columns in the grid.
  - In this case, we’re using **repeat(auto-fill, minmax(200px, 1fr))**, which means the grid will automatically fill the available space with columns of at least 200px width.
  - The **grid-gap** property sets the gap between grid items.
- The **.header** and **.footer** divs have a background color, and their text is centered.
- The **.footer** div spans across all columns using **grid-column: 1 / -1**, which ensures it stretches the full width of the grid.
- In the **media quer**y, we adjust the grid template columns to repeat(**auto-fill, minmax(100%, 1fr)**), which makes the grid items stack vertically on smaller screens.

# Which to use?

---

**Flexbox vs. Grid:** Flexbox is best suited for laying out items in one dimension (either as a row or a column), whereas CSS Grid Layout is better for two-dimensional layouts. Often, they are used together for complex layouts.

# Positioning Elements

# Positioning

---

### The position property

The CSS ***position*** property gives developers more control over where elements should appear in the browser. position has four possible values:

- static - ***Static positioning*** is the default positioning
- relative - ***Relative positioning*** positions the element relative to the element's default position
- fixed - ***Fixed positioning*** positions the element relative to the viewport in a fixed location
- absolute - ***Absolute positioning*** positions the element relative to the nearest positioned ancestor

**Absolute** positioning is similar to **fixed** positioning except:

1. The position is based on the nearest positioned ancestor element that uses fixed, absolute, or relative positioning. If no positioned ancestor element exists, the element is positioned relative to the document body.
2. An absolute-positioned element scrolls with the document unless an ancestor element is using fixed positioning.