# CSS Essentials

*CSS Layout*

# Overview

1. Overview of CSS Layout
2. Normal flow
3. Float
4. Flex
5. Grid
6. Position
7. Responsive

Section 1

# Overview of CSS Layout

# 1. Overview of CSS Layout

- **CSS page layout** techniques allow us to take elements contained in a web page and control where they are positioned relative to their default position in normal layout flow, the other elements around them, their parent container, or the main viewport/window.
- **The page layout techniques:**
  - Normal flow
  - The display property
  - Flexbox
  - Grid
  - Multiple-column layout
  - Floats
  - Positioning
  - Table layout

Section 2

# Normal flow

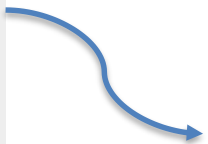
# 2. Normal flow

- **Normal flow** is how the browser lays out HTML pages by default when you do nothing to control page layout.

```
<p>I love my cat.</p>

<ul>
  <li>Buy cat food</li>
  <li>Exercise</li>
  <li>Cheer up friend</li>
</ul>

<p>The end!</p>
```



I love my cat.

- Buy cat food
- Exercise
- Cheer up friend

The end!

Section 3

# Float

# 3. Float

- The **float** property specifies how an element should float.
- **Floating** an element changes the behavior of that element and the block level elements that follow it in **normal flow**.
- The element is moved to the **left** or **right** and removed from **normal flow**, and the surrounding content floats around the floated item.

❖ **Note:**

- Absolutely positioned elements ignore the float property
- Elements after a floating element will flow around it. To avoid this, use the clear property or the clearfix hack.

# 3. Float

- The CSS **clear** property specifies what elements can float beside the cleared element and on which side.
- The **clear** property can have one of the following values:
  - none - Allows floating elements on both sides. This is default
  - left - No floating elements allowed on the left side
  - right- No floating elements allowed on the right side
  - both - No floating elements allowed on either the left or the right side
  - inherit - The element inherits the clear value of its parent

# 3. Float

- The **float** property has four possible values:
  - *left* — Floats the element to the left.
  - *right* — Floats the element to the right.
  - *none* — Specifies no floating at all. This is the default value.
  - *inherit* — Specifies that the value of the float property should be inherited from the element's parent element.

# 3. Float

- The **clearfix Hack:** If an element is taller than the element containing it, and it is floated, it will "overflow" outside of its container
- Then we can add ***overflow: auto****;* to the containing element to fix this problem

Section 4

# Flexbox

# 4. Flexbox

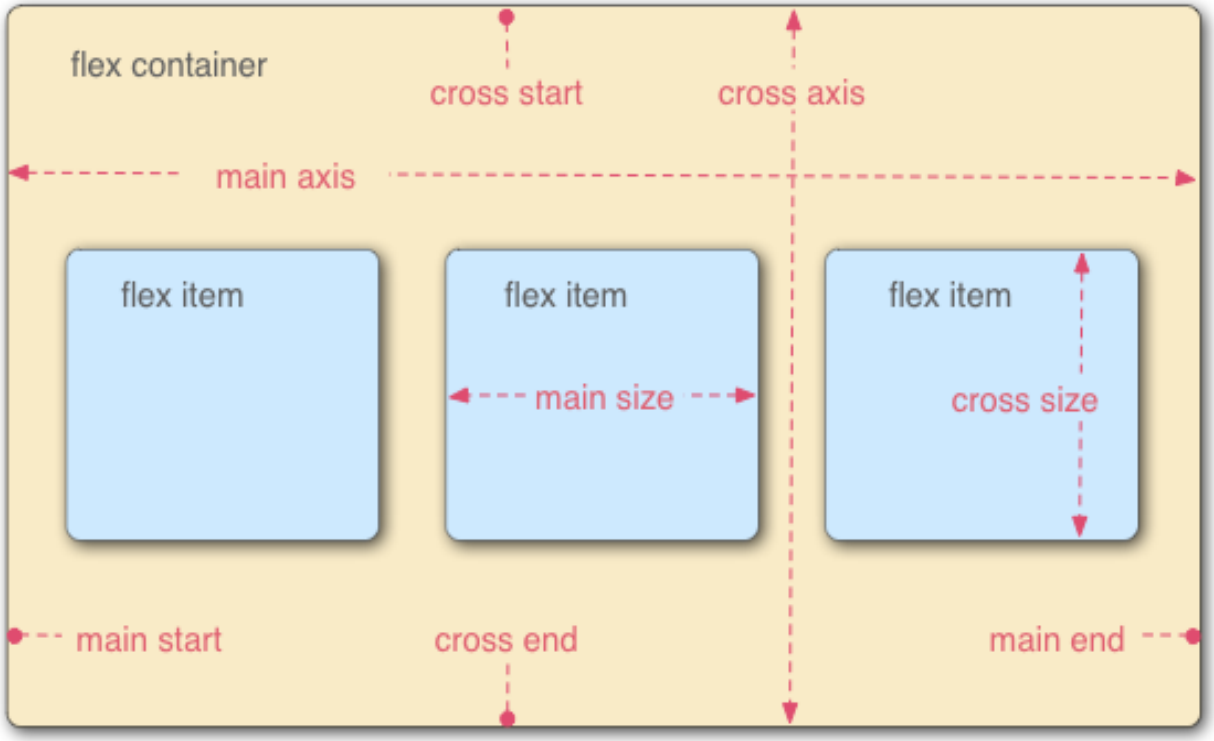
## ➢ Why flexbox?

The following simple layout requirements are either difficult or impossible to achieve with such tools, in any kind of convenient, flexible way:

- Vertically centering a block of content inside its parent.
- Making all the children of a container take up an equal amount of the available width/height, regardless of how much width/height is available.
- Making all columns in a multiple column layout adopt the same height even if they contain a different amount of content.

# 4. Flexbox

## An aside on the flex model



# 4. Flexbox

## ➢ How to use flexbox?

- To start using the Flexbox model, you need to first define a **flex container**.
- The flex container becomes flexible by setting the **display property** to **flex**.

# 4. Flexbox

## How to use flexbox?

- The **flex container properties** are:
  - flex-direction
  - flex-wrap
  - flex-flow
  - justify-content
  - align-items
  - align-content

# 4. Flexbox

- **Flex direction:** The flex-direction property specifies the direction of the flexible items - by default this is set to row
- Below are values of the **flex-direction** property

| Value | Description |
|---|---|
| row | Default value. The flexible items are displayed horizontally, as a row |
| row-reverse | Same as row, but in reverse order |
| column | The flexible items are displayed vertically, as a column |
| column-reverse | Same as column, but in reverse order |
| initial | Sets this property to its default value. Read about *initial* |
| inherit | Inherits this property from its parent element. Read about *inherit* |

# 4. Flexbox

- **Flex wrap:** The flex-wrap property specifies whether the flexible items should wrap or not.
- Below are values of the **flex-wrap** property

| Value | Description |
|---|---|
| nowrap | Default value. Specifies that the flexible items will not wrap |
| wrap | Specifies that the flexible items will wrap if necessary |
| wrap-reverse | Specifies that the flexible items will wrap, if necessary, in reverse order |
| initial | Sets this property to its default value. Read about *initial* |
| inherit | Inherits this property from its parent element. Read about *inherit* |

# 4. Flexbox

- **Flex flow:** The flex-flow property is a shorthand property for:
  - flex-direction
  - Flex-wrap
- **Syntax:** *flex-flow: flex-direction flex-wrap|initial|inherit;*

- **Flex - shorthand versus longhand:** flex is a shorthand property that can specify up to three different values
  - *flex-grow*
  - *flex-shrink*
  - *flex-basis*

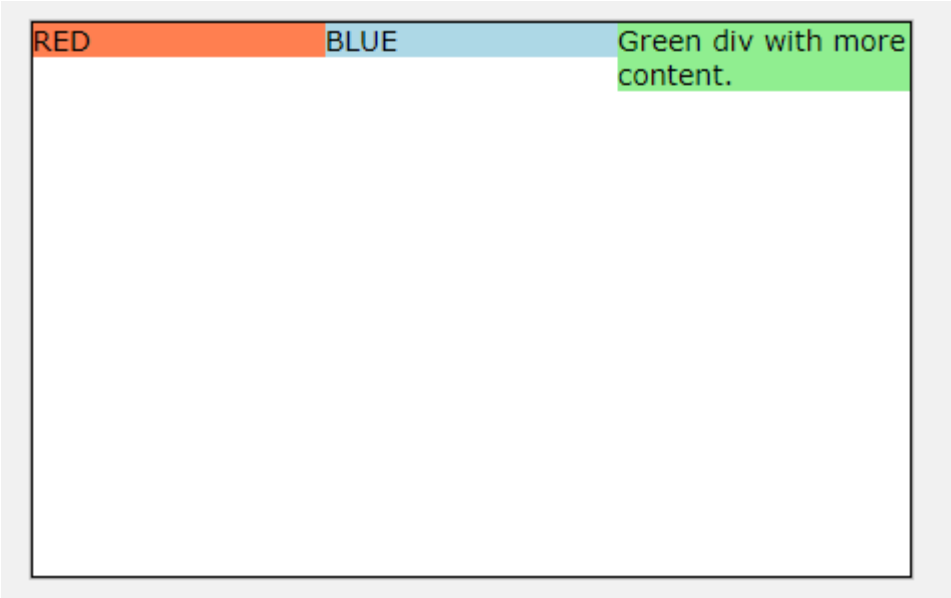
# 4. Flexbox

- **Align items:** The align-items property specifies the default alignment for items inside the flexible container.
- Below are values of the **align-items** property:

| Value | Description |
|---|---|
| stretch | Default. Items are stretched to fit the container |
| center | Items are positioned at the center of the container |
| flex-start | Items are positioned at the beginning of the container |
| flex-end | Items are positioned at the end of the container |
| baseline | Items are positioned at the baseline of the container |
| initial | Sets this property to its default value. Read about *initial* |
| inherit | Inherits this property from its parent element. Read about *inherit* |

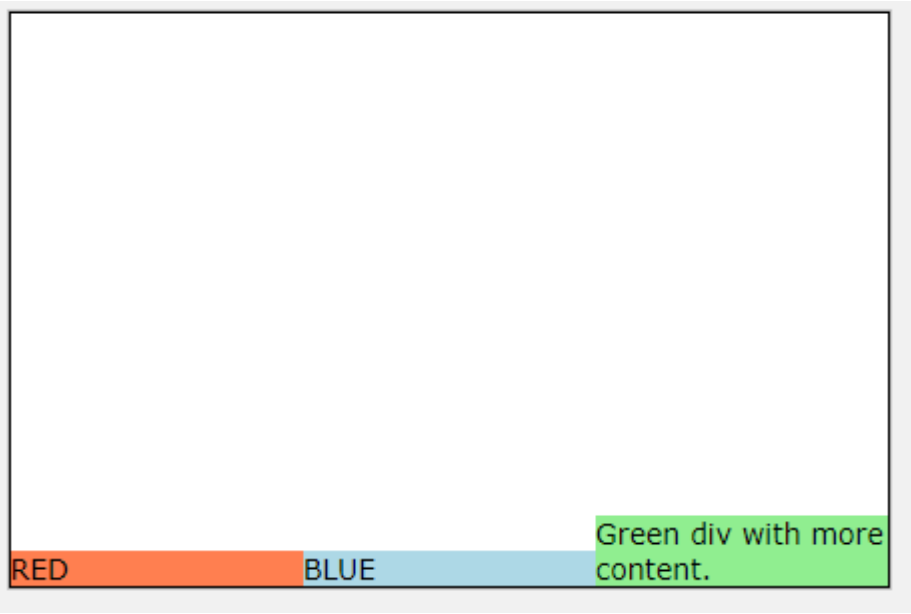
# 4. Flexbox

- **align-items**
  - **Flex-start:**



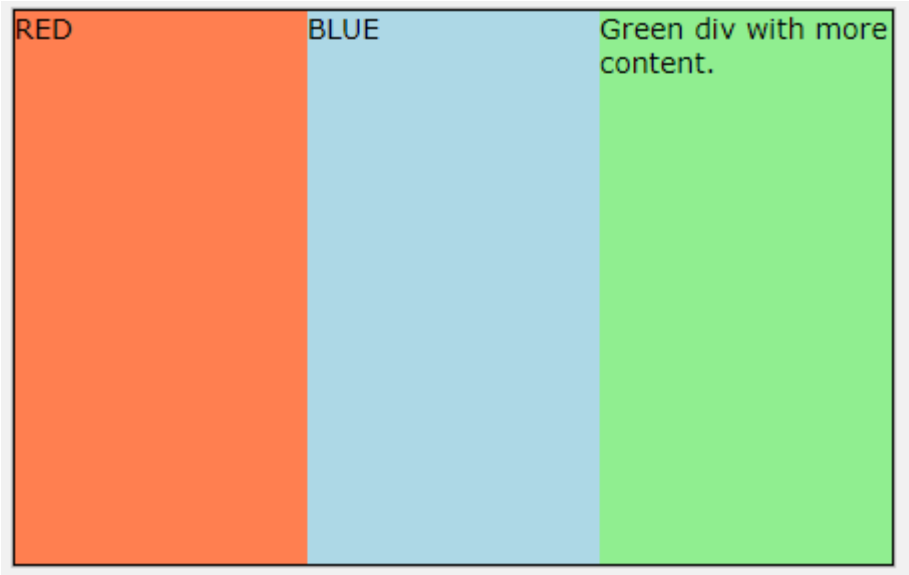
# 4. Flexbox

- **align-items**
  - **Flex-end:**



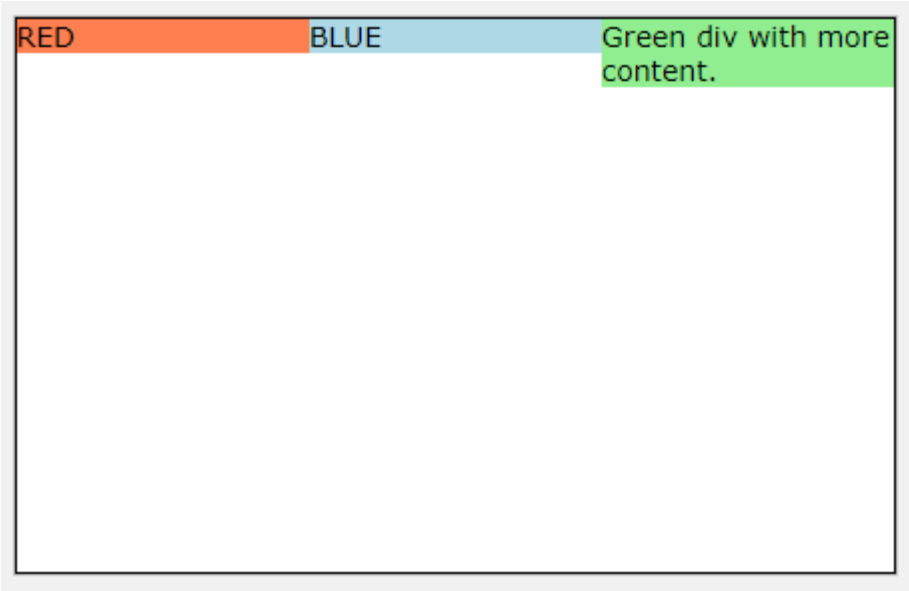
# 4. Flexbox

- **align-items**
  - **stretch:**



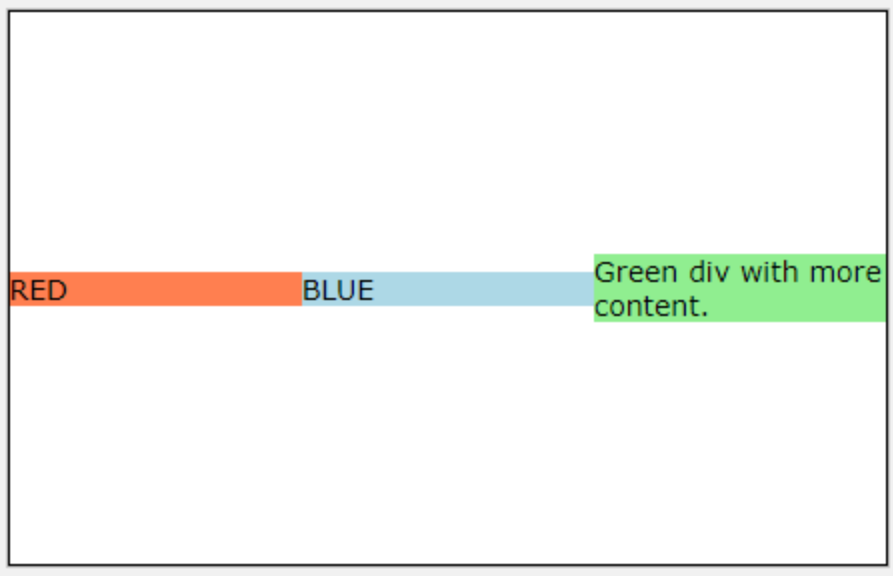
# 4. Flexbox

- **align-items**
  - **Base-line:**



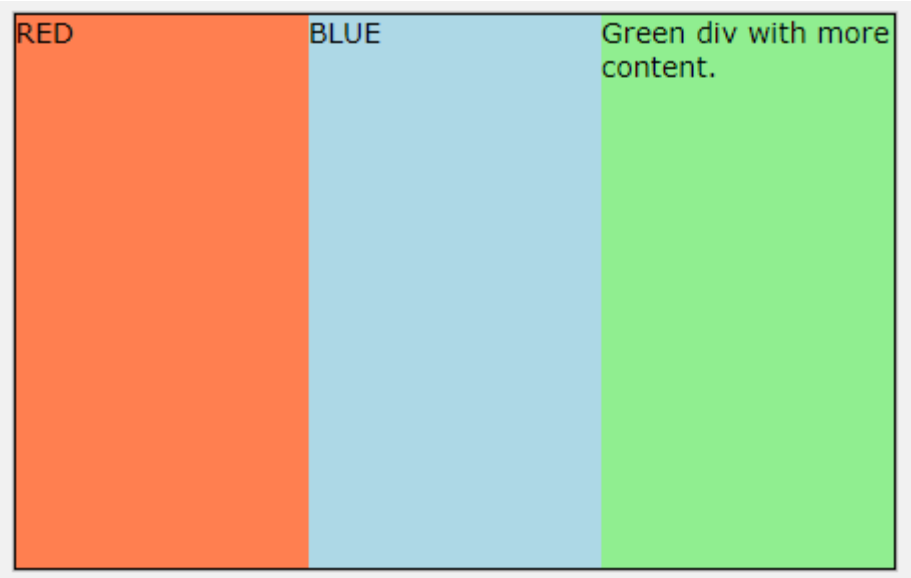
# 4. Flexbox

- **align-items**
  - **center:**



# 4. Flexbox

- **align-items**
  - **initial:**



# 4. Flexbox

- **Align content:** The align-content property modifies the behavior of the flex-wrap property. It is similar to align-items, but instead of aligning flex items, it aligns flex lines.
- Below are values of the **align-items** property:

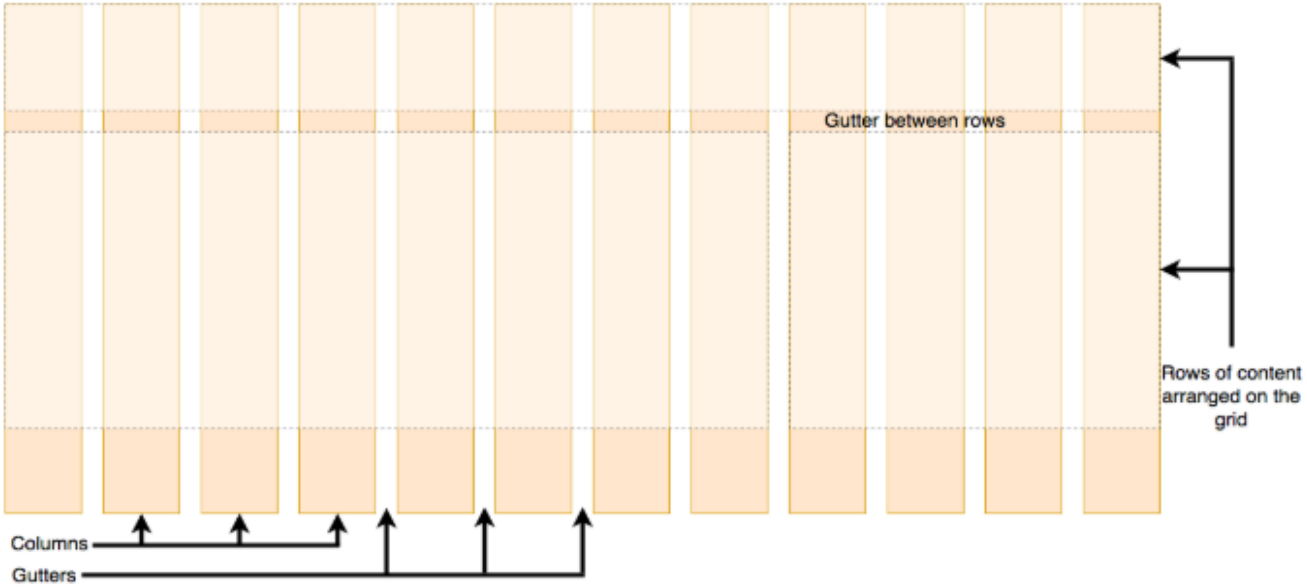
| Value | Description |
|---|---|
| stretch | Default value. Lines stretch to take up the remaining space |
| center | Lines are packed toward the center of the flex container |
| flex-start | Lines are packed toward the start of the flex container |
| flex-end | Lines are packed toward the end of the flex container |
| space-between | Lines are evenly distributed in the flex container |
| space-around | Lines are evenly distributed in the flex container, with half-size spaces on either end |
| initial | Sets this property to its default value. Read about *initial* |
| inherit | Inherits this property from its parent element. Read about *inherit* |

Section 5

# Grid

# 5. Grid

- **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.



# 5. Grid

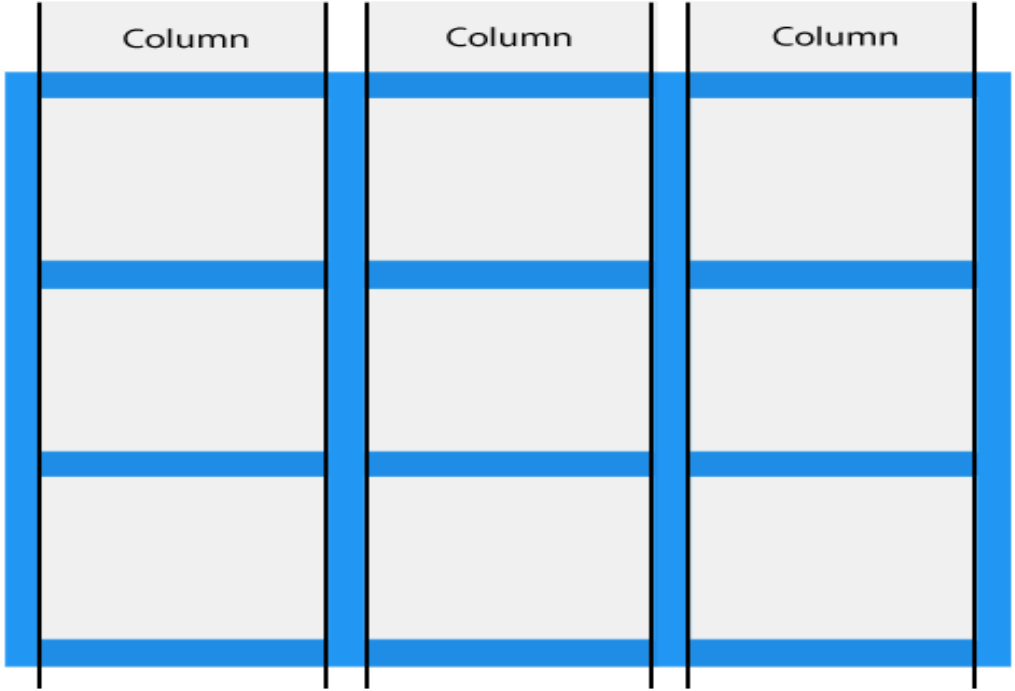
- A grid layout consists of a parent element, with one or more child elements.

```
<div class="grid-container">
  <div class="grid-item">1</div>
  <div class="grid-item">2</div>
  <div class="grid-item">3</div>
  <div class="grid-item">4</div>
  <div class="grid-item">5</div>
  <div class="grid-item">6</div>
  <div class="grid-item">7</div>
  <div class="grid-item">8</div>
  <div class="grid-item">9</div>
</div>
```

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

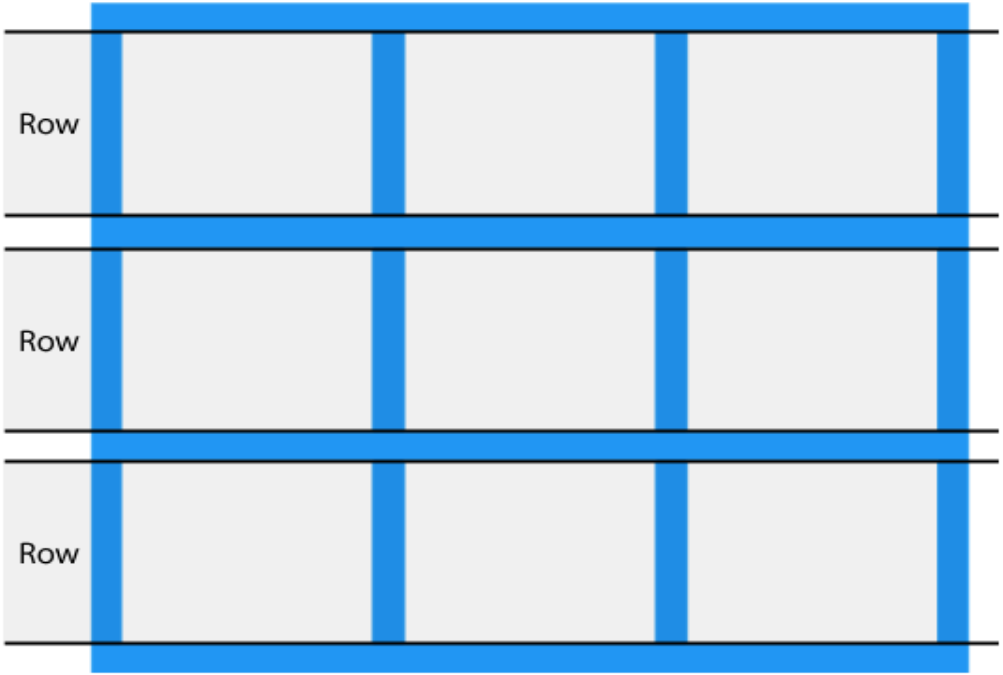
# 5. Grid

- **Grid Columns:** The vertical lines of grid items are called *columns*.



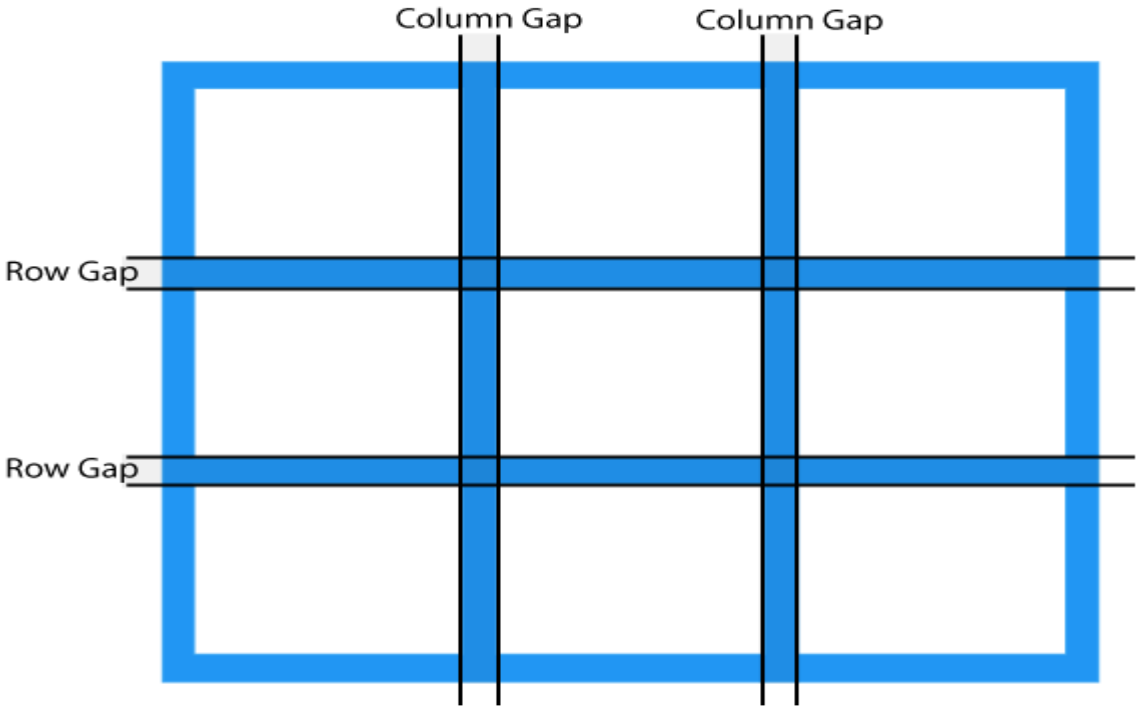
# 5. Grid

- **Grid Rows:** The horizontal lines of grid items are called *rows.*



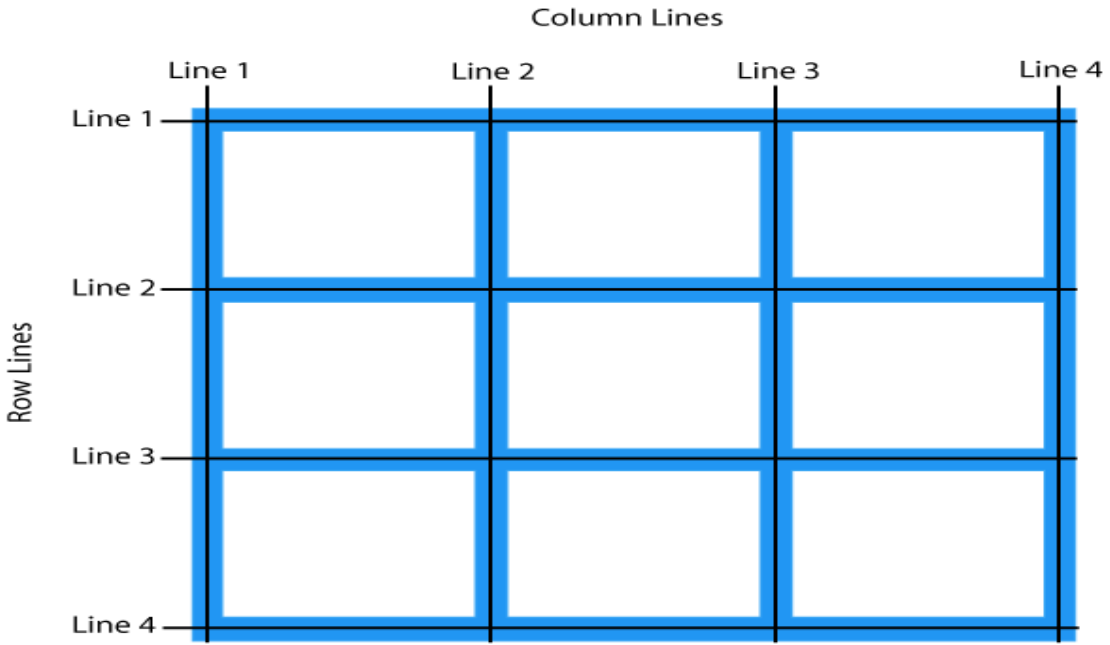
# 5. Grid

- **Grid Gaps:** The spaces between each column/row are called *gaps.*



# 5. Grid

- **Grid Lines:** The lines between columns are called column lines. The lines between rows are called row lines.



# 5. Grid

- The **grid-template-columns** property defines the number of columns in your grid layout, and it can define the width of each column.

The value is a space-separated-list, where each value defines the length of the respective column.

- The **grid-template-rows** property defines the height of each row.

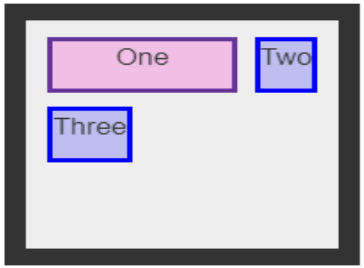
The value is a space-separated-list, where each value defines the height of the respective row

- The **align-content** property is used to vertically align the whole grid inside the container.

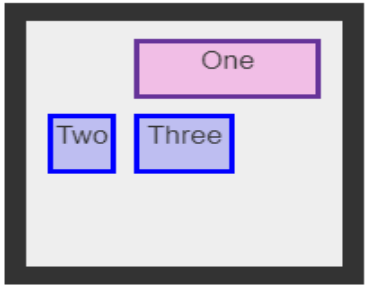
# 5. Grid

- The **grid-column** property defines on which column(s) to place an item.The value is a space-separated-list, where each value defines the length of the respective column.

```
grid-column: 1 / 3;
```



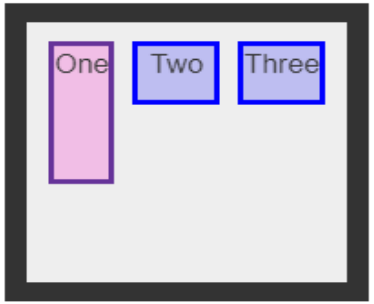

```
grid-column: 2 / -1;
```



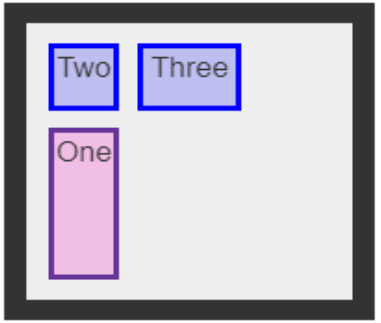
# 5. Grid

- The **grid-row** property defines on which row to place an item.

```
grid-row: 1 / 3;
```



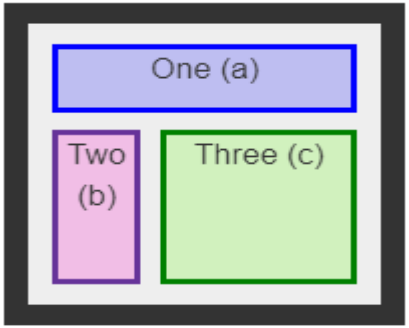

```
grid-row: 2 / -1;
```



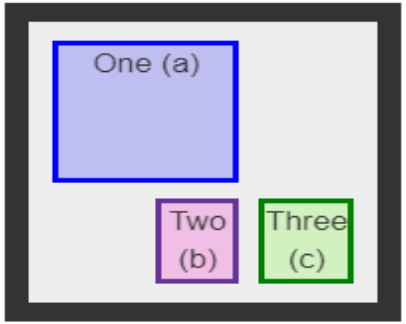
# 5. Grid

- **Grid template area**: The **grid-template-areas** property specifies areas within the grid layout.

```
grid-template-areas:
            "a a a"
            "b c c"
            "b c c";
```




```
grid-template-areas:
            "a a ."
            "a a ."
            ". b c";
```



Section 6

# Positioning

# 6. Positioning

- The **position** property specifies the type of positioning method used for an element.
- There are five different position values:
  - *Static*
  - *Relative*
  - *Fixed*
  - *Absolute*
  - *Sticky*

# 6. Positioning

- **static:**
  - HTML elements are positioned **static** by default.
  - **Static positioned elements** are not affected by the *top*, *bottom*, *left*, and *right* properties.
  - An element with *“position: static;”* is not positioned in any special way; it is always positioned according to the normal flow of the page.

# 6. Positioning

- **relative:**
  - An element with *“position: relative;”* is positioned relative to its normal position.
  - Setting the top, right, bottom, and left properties of **a relatively-positioned element** will cause it to be adjusted away from its normal position. Other content will not be adjusted to fit into any gap left by the element.

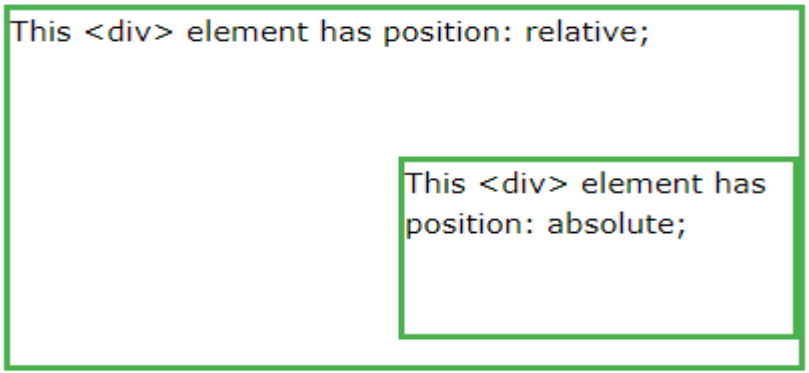
# 6. Positioning

- **fixed:**
  - An element with *“position: fixed;”* is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled.
  - The *top*, *right*, *bottom*, and *left* properties are used to position the element.
  - **A fixed element** does not leave a gap in the page where it would normally have been located.

# 6. Positioning

## ➢ absolute:

- An element with *“position: absolute;”* is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).
- If an **absolute positioned element** has no positioned ancestors, it uses the document body, and moves along with page scrolling.



# 6. Positioning

- **sticky:**
  - An element with position: sticky; is positioned based on the user's scroll position.
  - **A sticky element** toggles between relative and fixed, depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like position:fixed).

Section 6

# Responsive Design

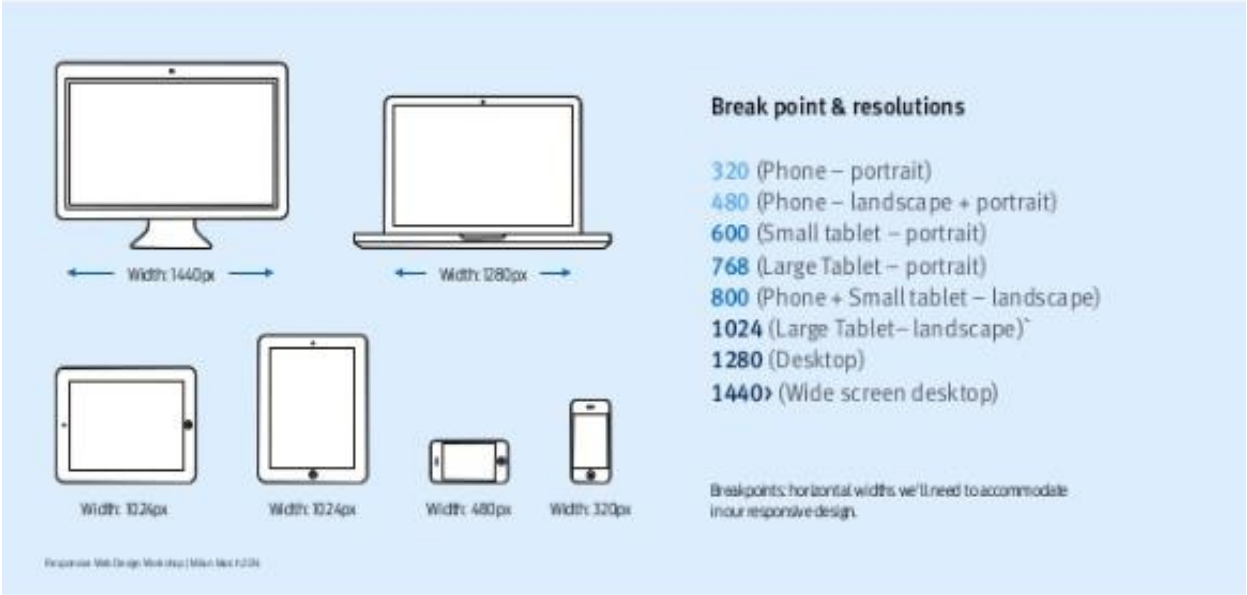
# 7. Responsive Design

- **Responsive** web design allows you to use CSS and HTML to resize, hide, shrink, enlarge, or move the content to make it look good on any screen.
- The **viewport** is the user's visible area of a web page.
- HTML5 introduced a method to let web designers take control over the viewport, through the <*meta*> tag.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

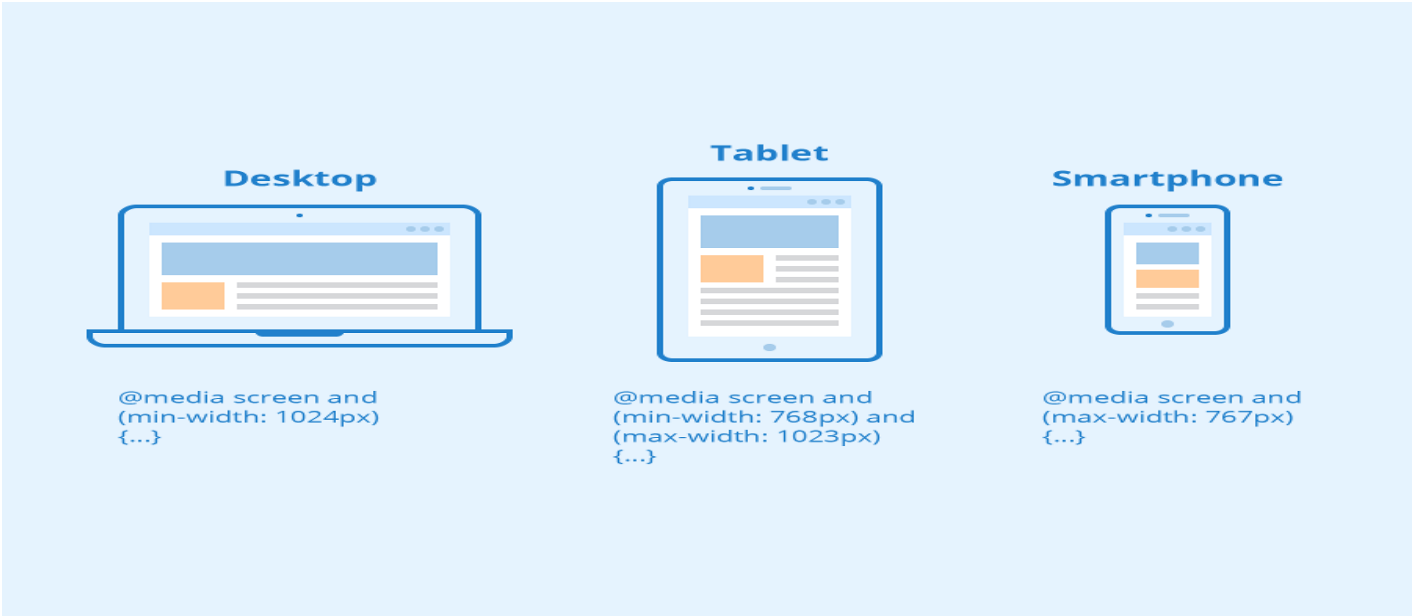
# 7. Responsive Design

- **Break point:** CSS breakpoints are points where the website content responds according to the device width, allowing you to show the best possible layout to the user.



# 7. Responsive Design

- **Media Queries** is a CSS technique introduced in CSS3. It uses the **@media** rule to include a block of CSS properties only if a certain condition is true.



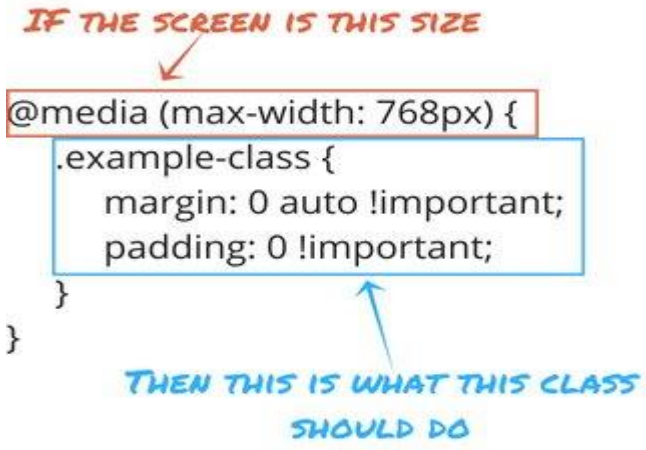
# 7. Responsive Design

- **Media Queries** is a CSS technique introduced in CSS3. It uses the **@media** rule to include a block of CSS properties only if a certain condition is true.
- The **@media** rule is used in media queries to apply different styles for different media types/devices.
- **Media queries** can be used to check many things, such as:
  - width and height of the viewport
  - width and height of the device
  - orientation (is the tablet/phone in landscape or portrait mode?)
  - resolution

# 7. Responsive Design

## ➢ Syntax of media query

```
@media not|only mediatype and (mediafeature and|or|not
mediafeature) {
  CSS-Code;
}
```



# 7. Responsive Design

## ➢ Media types

| Value | Description |
|---|---|
| all | Default. Used for all media type devices |
| print | Used for printers |
| screen | Used for computer screens, tablets, smart-phones etc. |
| speech | Used for screenreaders that "reads" the page out loud |

# 7. Responsive Design

## ➢ Media features

| Value | Description |
|---|---|
| all | Default. Used for all media type devices |
| print | Used for printers |
| screen | Used for computer screens, tablets, smart-phones etc. |
| speech | Used for screenreaders that "reads" the page out loud |

# Thank you