Class and ID Selectors

CSS classes can be reusable and applied to many elements. Class selectors are denoted with a period `.` followed by the class name. CSS ID selectors should be unique and used to style only a single element. ID selectors are denoted with a hash sign `#` followed by the id name.

```
/* Selects all elements with class="column" */
.column {
}

/* Selects element with id="first-item" */
#first-item {
}
```

Write CSS in Separate Files

CSS code can be written in its own files to keep it separate from the HTML code. The extension for CSS files is **.css**. These can be linked to an HTML file using a `<link>` tag in the `<head>` section.

```
<head>
  <link href="style.css" type="text/css" rel="stylesheet">
</head>
```

Groups of CSS Selectors

Match multiple selectors to the same CSS rule, using a comma-separated list. In this example, the text for both `h1` and `h2` is set to red.

```
h1, h2 {
  color: red;
}
```

Write CSS in HTML File

CSS code can be written in an HTML file by enclosing the code in `<style>` tags. Code surrounded by `<style>` tags will be interpreted as CSS syntax.

```
<head>
  <style>
    h1 {
      color: blue;
    }
  </style>
</head>
```

CSS {selectors: cheat-sheet}

By Web Dev Simplified https://courses.webdevsimplified.com

Basic

Name	CSS	Description	Results
Universal Selector	*	Select all elements	a b c d
Type Selector	div	Select elements of that type Select div elements	a div c div
Class Selector	.c	Select elements with that class Select elements with the c class	.a .b .c .d
Id Selector	#i	Select elements with that id Select elements with the i id *It is best practice to not use ids in CSS	#a #b #i #d

Selector Chaining

CSS *selectors* define the set of elements to which a CSS rule set applies. For instance, to select all `<p>` elements, the `p` selector can be used to create style rules.

`!important` Rule

The CSS `!important` rule is used on declarations to override any other declarations for a property and ignore selector specificity. `!important` rules will ensure that a specific declaration always applies to the matched elements. However, generally it is good to avoid using `!important` as bad practice.

```
#column-one {
  width: 200px !important;
}

.post-title {
  color: blue !important;
}
```

Chaining Selectors

CSS selectors can be chained so that rule sets apply only to elements that match all criteria. For instance, to select `<h3>` elements that also have the `section-heading` class, the selector `h3.section-heading` can be used.

```
/* Select h3 elements with the section-heading class */
h3.section-heading {
  color: blue;
}

/* Select elements with the section-heading and button class */
.section-heading.button {
  cursor: pointer;
}
```

CSS Type Selectors

CSS *type selectors* are used to match all elements of a given type or tag name. Unlike for HTML syntax, we do not include the angle brackets when using type selectors for tag names. When using type selectors, elements are matched regardless of their nesting level in the HTML.

```
/* Selects all <p> tags */
p {
}
```

Inline Styles

CSS styles can be directly added to HTML elements by using the `style` attribute in the element's opening tag. Each style declaration is ended with a semicolon. Styles added in this manner are known as *inline styles*.

```
<h2 style="text-align: center;">Centered text</h2>

<p style="color: blue; font-size: 18px;">Blue, 18-point text</p>
```

CSS class selectors

The CSS class selector matches elements based on the contents of their `class` attribute. For selecting elements having `calendar-cell` as the value of the `class` attribute, a `.` needs to be prepended.

```
.calendar-cell {
  color: #fff;
}
```

HTML attributes with multiple values

Some HTML attributes can have multiple attribute values. Multiple attribute values are separated by a space between each attribute.

```
<div class="value1 value2 value3"></div>
```

Selector Specificity

Specificity is a ranking system that is used when there are multiple conflicting property values that point to the same element. When determining which rule to apply, the selector with the highest specificity wins out. The most specific selector type is the ID selector, followed by class selectors, followed by type selectors. In this example, only `color: blue` will be implemented as it has an ID selector whereas `color: red` has a type selector.

```
h1#header {
  color: blue;
} /* implemented */

h1 {
  color: red;
} /* Not implemented */
```

CSS ID selectors

The CSS ID selector matches elements based on the contents of their `id` attribute. The values of `id` attribute should be unique in the entire DOM. For selecting the element having `job-title` as the value of the `id` attribute, a `#` needs to be prepended.

```
#job-title {
  font-weight: bold;
}
```

CSS descendant selector

The CSS *descendant* selector combinator is used to match elements that are descended from another matched selector. They are denoted by a single space between each selector and the descended selector. All matching elements are selected regardless of the nesting level in the HTML.

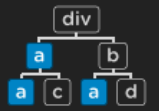
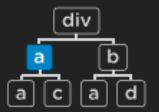
```
div p { }

section ol li { }
```

CSS declarations

In CSS, a *declaration* is the key-value pair of a CSS property and its value. CSS declarations are used to set style properties and construct rules to apply to individual or groups of elements. The property name and value are separated by a colon, and the entire declaration must be terminated by a semi-colon.

```
/*
CSS declaration format:
property-name: value;
*/

/* CSS declarations */
text-align: center;
color: purple;
width: 100px;
```

Combination

Name	CSS	Description	Results
Descendant Selector	`div a`	Select elements that are descendants of the first element Select anchors that are inside a div	
Direct Child Selector	`div > a`	Select elements that are direct children of the first element Select anchors that are direct children of a div	
General Sibling Selector	`div ~ a`	Select elements that are siblings of the first element and come after the first element Selects all anchors that are siblings of a div and come after the div	
Adjacent Sibling Selector	`div + a`	Select elements that are siblings of the first element and come directly after the first element Selects all anchors that are siblings of a div and come directly after the div	
Or Selector	`div, a`	Select elements that match any selector in the list Selects all anchors and all divs	
And Selector	`div.c`	Select elements that match all the selector combinations Selects all divs with the class c	

Attribute

Name	CSS	Description	Results
Has Attribute	[a]	Select elements that have that attribute Select elements with the a attribute	[a] [a="1"] [c] d
Exact Attribute	[a="1"]	Select elements that have that attribute with exactly that value Select elements with the a attribute with a value of 1	[a] [a="1"] [c] d
Begins With Attribute	[a^="1"]	Select elements that have that attribute which start with that value Select elements with the a attribute with a value that starts with 1	[a="12"] [a="21"]
Ends With Attribute	[a$="1"]	Select elements that have that attribute which end with that value Select elements with the a attribute with a value that ends with 1	[a="12"] [a="21"]
Substring Attribute	[a*="1"]	Select elements that have that attribute which contain that value anywhere Select elements with the a attribute with a value that contains a 1	[a="12"] [a="21"]

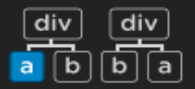
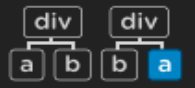
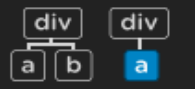
Pseudo Element

Name	CSS	Description	Results
Before Selector	`div::before`	Creates an empty element directly before the children of selected element	
After Selector	`div::after`	Creates an empty element directly after the children of selected element	

Pseudo Class State

Name	CSS	Description
Hover Selector	`button:hover`	**Select elements that are hovered by the mouse** Select buttons that are being hovered
Focus Selector	`button:focus`	**Select elements that are focused.** Select buttons that are being focused *Focus is set by either tabbing to an element or clicking an element such as a button or anchor tag
Required Selector	`input:required`	**Select inputs that are required** Select inputs with the required attribute
Checked Selector	`input:checked`	**Select checkboxes/radio buttons that are checked** Select inputs that are checked
Disabled Selector	`input:disabled`	**Select inputs that are disabled** Select inputs with the disabled attribute

Pseudo Class Position/Other

Name	CSS	Description	Results
First Child Selector	a:first-child	Select elements that are the first child inside a container Select anchors that are the first child	
Last Child Selector	a:last-child	Select elements that are the last child inside a container Select anchors that are the last child	
Nth Child Selector	a:nth-child(2n)	Select elements that are the nth child inside a container based on the formula Select anchors that are even numbered children	
Nth Last Child Selector	a:nth-last-child(3)	Select elements that are the nth child inside a container based on the formula counting from the end Select anchors that are the third to last child	
Only Child Selector	a:only-child	Select elements that are the only child inside a container Select anchors that are the only child	
First Of Type Selector	a:first-of-type	Select elements that are the first of a type inside a container Select the first anchor in a container	
Last Of Type Selector	a:last-of-type	Select elements that are the last of a type inside a container Select the last anchor in a container	
Nth Of Type Selector	a:nth-of-type(2n)	Select elements that are the nth of a type inside a container based on the formula Select every second anchor	
Nth Last Of Type Selector	a:nth-last-of-type(2)	Select elements that are the nth of a type inside a container based on the formula counting from the end Select the second to last anchor	
Only Of Type Selector	a:only-of-type	Select elements that are the only of a type inside a container Select anchors that are the only anchor in a container	
Not Selector	a:not(.c)	Select all elements that do not match the selector inside the not selector Select all anchor tags that do not have the c class	

Font Size

The `font-size` CSS property is used to set text sizes. Font size values can be many different units or types such as pixels.

```
font-size: 30px;
```

Background Color

The `background-color` CSS property controls the background color of elements.

```
background-color: blue
```

Opacity

The `opacity` CSS property can be used to control the transparency of an element. The value of this property ranges from `0` (transparent) to `1` (opaque).

```
opacity: 0.5;
```

Font Weight

The `font-weight` CSS property can be used to set the weight (boldness) of text. The provided value can be a keyword such as `bold` or `normal`.

```
font-weight: bold;
```

Text Align

The `text-align` CSS property can be used to set the text alignment of inline contents. This property can be set to these values: `left`, `right`, or `center`.

```
text-align: right;
```

CSS Rule Sets

A CSS rule set contains one or more selectors and one or more declarations. The selector(s), which in this example is `h1`, points to an HTML element. The declaration(s), which in this example are `color: blue` and `text-align: center` style the element with a property and value. The rule set is the main building block of a CSS sheet.

```
h1 {
  color: blue;
  text-align: center;
}
```

Setting foreground text color in CSS

Using the `color` property, foreground text color of an element can be set in CSS. The value can be a valid color name supported in CSS like `green` or `blue`. Also, 3 digit or 6 digit color code like `#22f` or `#2a2aff` can be used to set the color.

```
p {
  color : #2a2aff ;
}

span {
  color : green ;
}
```

Resource URLs

In CSS, the `url()` function is used to wrap resource URLs. These can be applied to several properties such as the `background-image`.

```
background-image: url("../resources/image.png");
```

Font Family

The `font-family` CSS property is used to specify the typeface in a rule set. Fonts must be available to the browser to display correctly, either on the computer or linked as a web font. If a font value is not available, browsers will display their default font. When using a multi-word font name, it is best practice to wrap them in quotes.

```
h2 {
  font-family: Verdana;
}

#page-title {
  font-family: "Courier New";
}
```

Color Name Keywords

Color name keywords can be used to set color property values for elements in CSS.

```
h1 {
  color: aqua;
}

li {
  color: khaki;
}
```

Background Image

The `background-image` CSS property sets the background image of an element. An image URL should be provided in the syntax `url("moon.jpg")` as the value of the property.

```
background-image: url("nyan-cat.gif");
```

CSS Margin Collapse

CSS *margin collapse* occurs when the top and bottom margins of blocks are combined into a single margin equal to the largest individual block margin.

Margin collapse only occurs with vertical margins, not for horizontal margins.

```
/* The vertical margins will collapse to 30 pixels
instead of adding to 50 pixels. */
.block-one {
  margin: 20px;
}

.block-two {
  margin: 30px;
}
```

CSS `auto` keyword

The value `auto` can be used with the property `margin` to horizontally center an element within its container. The `margin` property will take the width of the element and will split the rest of the space equally between the left and right margins.

```
div {
  margin: auto;
}
```

Dealing with `overflow`

If content is too large for its container, the CSS `overflow` property will determine how the browser handles the problem.

By default, it will be set to `visible` and the content will take up extra space. It can also be set to `hidden`, or to `scroll`, which will make the overflowing content accessible via scroll bars within the original container.

```
small-block {
  overflow: scroll;
}
```

Height and Width Maximums/Minimums

The CSS `min-width` and `min-height` properties can be used to set a minimum width and minimum height of an element's box. CSS `max-width` and `max-height` properties can be used to set maximum widths and heights for element boxes.

```
/* Any element with class "column" will be at most 200 pixels
wide, despite the width property value of 500 pixels. */

.column {
  max-width: 200px;
  width: 500px;
}
```

The `visibility` Property

The CSS `visibility` property is used to render `hidden` objects invisible to the user, without removing them from the page. This ensures that the page structure and organization remain unchanged.

```
.invisible-elements {
  visibility: hidden;
}
```

The property `box-sizing` of CSS *box model*

The CSS *box model* is a box that wraps around an HTML element and controls the design and layout. The property `box-sizing` controls which aspect of the box is determined by the `height` and `width` properties. The default value of this property is `content-box`, which renders the actual size of the element including the content box; but not the paddings and borders. The value `border-box`, on the other hand, renders the actual size of an element including the content box, paddings, and borders.

```
.container {
  box-sizing: border-box;
}
```

CSS `box-sizing: border-box`

The value `border-box` of the `box-sizing` property for an element corresponds directly to the element's total rendered size, including padding and border with the `height` and `width` properties.

```
#box-example {
  box-sizing: border-box;
}
```

The default value of the `border-box` property is `content-box`. The value `border-box` is recommended when it is necessary to resize the `padding` and `border` but not just the content. For instance, the value `border-box` calculates an element's `height` as follows: `height = content height + padding + border`.

Fixed CSS Positioning

Positioning in CSS provides designers and developers options for positioning HTML elements on a web page. The CSS `position` can be set to `static`, `relative`, `absolute` or `fixed`. When the CSS position has a value of `fixed`, it is set/pinned to a specific spot on a page. The fixed element stays the same regardless of scrolling. The navigation bar is a great example of an element that is often set to `position:fixed;`, enabling the user to scroll through the web page and still access the navigation bar.

```
navbar {
postion : fixed;
 }
```

CSS display property

The CSS `display` property determines the type of render block for an element. The most common values for this property are `block`, `inline`, and `inline-block`.

Block-level elements take up the full width of their container with line breaks before and after, and can have their height and width manually adjusted.

Inline elements take up as little space as possible, flow horizontally, and cannot have their width or height manually adjusted.

Inline-block elements can appear next to each other, and can have their width and height manually adjusted.

```
.container1 {
  display: block;
}

.container2 {
  display: inline;
}

.container3 {
  display: inline-block;
}
```

CSS position: absolute

The value `absolute` for the CSS property `position` enables an element to ignore sibling elements and instead be positioned relative to its closest parent element that is positioned with `relative` or `absolute`. The `absolute` value removes an element entirely from the document flow. By using the positioning attributes `top`, `left`, `bottom` and `right`, an element can be positioned anywhere as expected.

```
.element {
  position: absolute;
}
```

CSS position: relative

The value `relative` of the CSS `position` property enables an element to be positioned relative to where it would have originally been on a web page. The offset properties can be used to determine the actual position of the element relative to its original position. Without the offset properties, this declaration will have no effect on its positioning, it will act as the default value `static` of the `position` property.

```
.element {
  position: relative;
}
```

CSS float property

The CSS `float` property determines how far left or how far right an element should float within its parent element. The value `left` floats an element to the left side of its container and the value `right` floats an element to the right side of its container. For the property `float`, the `width` of the container must be specified or the element will assume the full width of its containing element.

```
/* The content will float to the left side of the container. */
.left {
  float: left;
}

/* The content will float to the right side of the container.
*/
.right {
  float: right;
}
```

The CSS clear property

The CSS `clear` property specifies how an element should behave when it bumps into another element within the same containing element.The `clear` is usually used in combination with elements having the CSS `float` property. This determines on which sides floating elements are allowed to float.

```
/*This determines that no other elements within the same
containing element are allowed to float on the left side of
this element.*/
.element {
  clear: left;
}

/*This determines that no other elements within the same
containing element are allowed to float on the right side of
this element.*/
.element {
  clear: right;
}

/*This determines that no elements within the same containing
element are allowed to float on either side of this element.*/
.element {
  clear: both;
}

/*This determines that other elements within the same
containing element are allowed to float on both side of this
element.*/
.element {
  clear: none;
}
```

CSS Color Alpha Values

Alpha values determine the transparency of colors in CSS. Alpha values can be set for both RGB and HSL colors by using `rgba()` and `hsla()` and providing a fourth value representing alpha. Alpha values can range between `0.0` (totally transparent) and `1.0` (totally opaque).

The CSS `transparent` value can also be used to create a fully transparent element.

```
.midground {
  background-color: rgba(0, 255, 0, 0.5);
}

.foreground {
  background-color: hsla(34, 100%, 50%, 0.1);
}

.transparent {
  color: transparent;
}
```

CSS Hexadecimal Colors

CSS colors can be represented in *hexadecimal* (or *hex*) notation. Hexadecimal digits can represent sixteen different values using `0` - `9` and `a` - `f`.

Hexadecimal colors are composed of 6 characters– each group of two represents a value between 0 and 255 for red, green, or blue. For example `#ff0000` is all red, no green, and no blue.

When both characters of all three colors are repeated, hex colors can be abbreviated to only three values, so `#0000ff` could also be represented as `#00f`.

```
.red {
  color: #ff0000;
}

.short-blue {
  color: #00f;
}
```

CSS HSL Colors

CSS colors can be declared with the *HSL* color system using `hsl()` syntax. This syntax contains three values: *hue* (the color value itself), *saturation* (intensity), and *lightness*.

Hue values range from 0 to 360 while saturation and lightness values are represented as percentages.

```
.light-blue {
  background-color: hsl(200, 70%, 50%);
}
```

CSS rgb() Colors

CSS colors can be declared with *RGB colors* using `rgb()` syntax.

`rgb()` should be supplied with three values representing red, green, and blue. These values range can from 0 to 255.

```
.hot-pink {
  color: rgb(249, 2, 171);
}

.green {
  color: rgb(0, 255, 0);
}
```

Color Name Keywords

Color name keywords can be used to set color property values for elements in CSS.

```
h1 {
  color: aqua;
}

li {
  color: khaki;
}
```

CSS font-weight Property

The CSS `font-weight` property declares how thick or thin should be the characters of a text. Numerical values can be used with this property to set the thickness of the text. The numeric scale range of this property is from 100 to 900 and accepts only multiples of 100. The default value is `normal` while the default numerical value is `400`. Any value less than `400` will have text appear lighter than the default while any numerical value greater than the `400` will appear bolder.

```
/* Sets the text as bolder. */
p {
  font-weight: 700;
}
```

In the given example, all the `<p>` elements will appear in a bolder font.

CSS font-style property

The CSS `font-style` property determines the font style in which text will appear.

It accepts `italic` as a value to set the font style to italic.

```
.text {
  font-style: italic;
}
```

CSS *@font-face* rule

The CSS *@font-face* rule allows external fonts or font files to be imported directly into stylesheets.The location of the font file must be specified in the CSS rule so that the files can be loaded from that location. This rule also allows locally hosted fonts to be added using a relative file path instead of a web URL.

```
@font-face {
  font-family: 'Glegoo';
  src: url('../fonts/Glegoo-Regular.ttf') format('truetype');
}
```

CSS Fallback Fonts

The CSS `font-family` property can have multiple fonts declared in order of preference. In this case the fonts following the initial font are known as the *fallback fonts*.

If the initial value of the property `font-family` fails to load to the webpage, the fallback fonts will be used.

```
/* Here `Arial` is the fallback font for <p> tags */
p {
  font-family: "Helvetica", "Arial";
}
```

The CSS `line-height` property

The CSS `line-height` property declares the vertical spacing between lines of text. It accepts both unitless numbers as a ratio (eg. `2`) and numbers specified by unit as values (eg. `12px`) but it does not accept negative numbers. A unitless number is an absolute value that will compute the line height as a ratio to the font size and a unit number can be any valid CSS unit (eg. pixels, percents, ems, rems, etc.). To set the `line-height` of the `<p>` elements to `10px`, the given CSS declaration can be used.

```
p {
line-height: 10px;
}
```

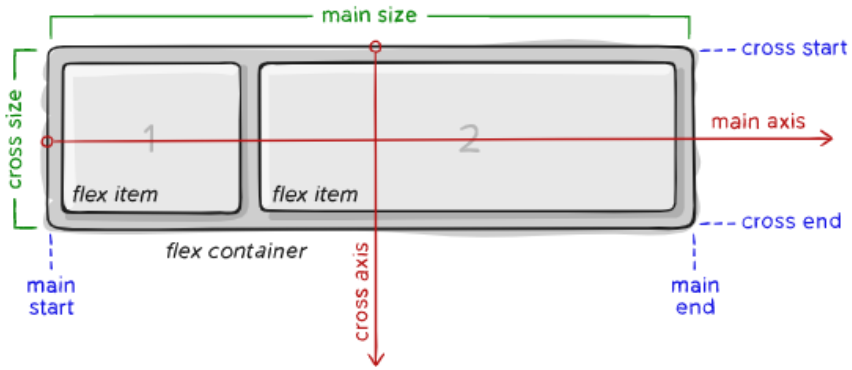
CSS *Linking fonts*

Linking fonts allow user to use web fonts in the document. They can be imported in an HTML document by using the `<link>` tag. Once the web font URL is placed within the `href` attribute, the imported font can then be used in CSS declaration.

```
<head>
  <link href="https://fonts.googleapis.com/css?
family=Droid%20Serif" rel="stylesheet">
</head>
```

Since flexbox is a whole module and not a single property, it involves a lot of things including its whole set of properties. Some of them are meant to be set on the container (parent element, known as "flex container") whereas the others are meant to be set on the children (said "flex items").

If "regular" layout is based on both block and inline flow directions, the flex layout is based on "flex-flow directions". Please have a look at this figure from the specification, explaining the main idea behind the flex layout.



flex-container properties

display	flex-wrap
flex · inline-flex	*nowrap* · wrap · wrap-reverse

flex-direction
row · row-reverse · column · column-reverse

justify-content
flex-start · flex-end · center · space-between · space-around

align-items
flex-start · flex-end · center · *stretch* · baseline

align-content
flex-start · flex-end · center · *stretch* · space-between · space-around

flex-item properties

flex-grow	flex-shrink	flex-basis
0 · *1* · *2* · …	0 · *1* · *2* · …	*auto* · value · fill · min-content · …

flex (= flex-grow flex-shrink flex-basis)
flex: initial; ⇢ *flex: 0 1 auto;*
flex: auto; ⇢ flex: 1 1 auto;
flex: none; ⇢ flex: 0 0 auto;
flex: value; ⇢ flex: value 1 0;

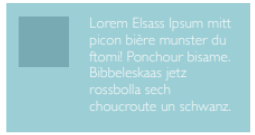
order
0 · *1* · *2* · …

align-self
auto · flex-start · flex-end · center · stretch · baseline

Items will be laid out following either the `main axis` (from `main-start` to `main-end`) or the cross axis (from `cross-start` to `cross-end`).

- **main axis** – The main axis of a flex container is the primary axis along which flex items are laid out. Beware, it is not necessarily horizontal; it depends on the `flex-direction` property (see below).
- **main-start | main-end** – The flex items are placed within the container starting from main-start and going to main-end.
- **main size** – A flex item's width or height, whichever is in the main dimension, is the item's main size. The flex item's main size property is either the 'width' or 'height' property, whichever is in the main dimension.
- **cross axis** – The axis perpendicular to the main axis is called the cross axis. Its direction depends on the main axis direction.
- **cross-start | cross-end** – Flex lines are filled with items and placed into the container starting on the cross-start side of the flex container and going toward the cross-end side.
- **cross size** – The width or height of a flex item, whichever is in the cross dimension, is the item's cross size. The cross size property is whichever of 'width' or 'height' that is in the cross dimension.

media object




```
.parent {
  display: flex;
  align-items: flex-start;
}
```

```
.content {
  flex: 1 1 0%;
}
```

modifiers:

```
.parent--reverse {
  flex-direction: row-reverse;
}
```

```
.figure--center {
  align-self: center;
}
```

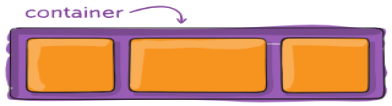
center everything

```
.parent {
  display: flex;
  justify-content: center;
  align-items: center;
}
```

alternative:

```
.parent {
  display: flex;
}
```

```
.children {
  margin: auto;
}
```



Properties for the Parent (flex container)

display

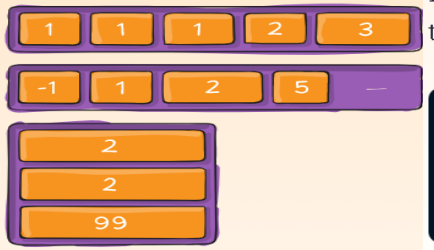
This defines a flex container; inline or block depending on the given value. It enables a flex context for all its direct children.

```css
.container {
  display: flex; /* or inline-flex */
}
```

Note that CSS columns have no effect on a flex container.

Properties for the Children (flex items)

order



By default, flex items are laid out in the source order. However, the `order` property controls the order in which they appear in the flex container.

```css
.item {
  order: 5; /* default is 0 */
}
```

flex Property

The `flex` CSS property specifies how a flex item will grow or shrink so as to fit within the space available in its `flex` container. This is a shorthand property that declares the following properties in order on a single line:

- `flex-grow`
- `flex-shrink`
- `flex-basis`

```
/* Three properties declared on three lines: */
.first-flex-item {
  flex-grow: 2;
  flex-shrink: 1;
  flex-basis: 150px;
}

/* Same three properties declared on one line: */
.first-flex-item {
  flex: 2 1 150px;
}
```

CSS Flexbox

The CSS `display: flex` property sets an HTML element as a block level flex container which takes the full width of its parent container. Any child elements that reside within the flex container are called flex items.

```
div {
  display: flex;
}
```

Flex items change their size and location in response to the size and position of their parent container.

mono-line grid

option 1 : flexible children

without gutters:

```
.parent {
  display: flex;
}
```

```
.children {
  flex: 1 1 0%;
}
```

with gutters:

```
.parent {
  display: flex;
}
```

```
.children {
  flex: 1 1 0%;
}
```

```
.parent > *:not(:first-child) {
  margin-left: gutter;
}
```

option 2 : space-between

without gutters:

```
.parent {
  display: flex;
}
```

```
.children {
  width: calc(100% / 3);
}
```

with gutters:

```
.parent {
  display: flex;
  justify-content: space-between;
}
.children {
  width: calc(100% / 3 - gutter / 2);
}
```

`flex-direction` Property

The `flex-direction` CSS property specifies how flex items are placed in the flex container - either vertically or horizontally. This property also determines whether those flex items appear in order or in reverse order.

```
div {
  display: flex;
  flex-direction: row-reverse;
}
```

`align-content` Property

The `align-content` property modifies the behavior of the flex-wrap property. It determines how to space rows from top to bottom (ie. along the cross axis). Multiple rows of items are needed for this property to take effect.

`flex-grow` Property

The CSS `flex-grow` property allows flex items to grow as the parent container increases in size horizontally. This property accepts numerical values and specifies how an element should grow relative to its sibling elements based on this value.

The default value for this property is `0`.

```
.panelA {
  width: 100px;
  flex-grow: 1;
}

/* This panelB element will stretch twice wider than the panelA element */
.panelB {
  width: 100px;
  flex-grow: 2;
}
```

flex-direction

This establishes the main-axis, thus defining the direction flex items are placed in the flex container. Flexbox is (aside from optional wrapping) a single-direction layout concept. Think of flex items as primarily laying out either in horizontal rows or vertical columns.

```css
.container {
  flex-direction: row | row-reverse | column | column-reve
}
```

- `row` (default): left to right in `ltr`; right to left in `rtl`
- `row-reverse`: right to left in `ltr`; left to right in `rtl`
- `column`: same as `row` but top to bottom
- `column-reverse`: same as `row-reverse` but bottom to top

flex-grow

This defines the ability for a flex item to grow if necessary. It accepts a unitless value that serves as a proportion. It dictates what amount of the available space inside the flex container the item should take up.

If all items have `flex-grow` set to `1`, the remaining space in the container will be distributed equally to all children. If one of the children has a value of `2`, the remaining space would take up twice as much space as the others (or it will try to, at least).

```css
.item {
  flex-grow: 4; /* default 0 */
}
```

Negative numbers are invalid.

`flex-shrink` Property

The CSS `flex-shrink` property determines how an element should shrink as the parent container decreases in size horizontally. This property accepts a numerical value which specifies the ratios for the shrinkage of a flex item compared to its other sibling elements within its parent container.

The default value for this property is `1`.

```
.container {
  display: flex;
}

.item-a {
  flex-shrink: 1;
  /* The value 1 indicates that the item should shrink. */
}

.item-b {
  flex-shrink: 2;
  /* The value 2 indicates that the item should shrink twice than the element item-a. */
}
```

Css flex-basis property

The `flex-basis` CSS property sets the initial base size for a flex item before any other space is distributed according to other flex properties.

```
// Default Syntax
flex-basis: auto;
```

The CSS `flex-flow` property

The CSS property `flex-flow` provides a shorthand for the properties `flex-direction` and `flex-wrap`. The value of the `flex-direction` property specifies the direction of the flex items and the value of the `flex-wrap` property allows flex items to move to the next line instead of shrinking to fit inside the flex container. The `flex-flow` property should be declared on the flex container.

```
// In this example code block, "column" is the value of the property "flex-direction" and "wrap" is the value of the property "flex-wrap".

.container {
  display: flex;
  flex-flow: column wrap;
}
```

flex-flow

This is a shorthand for the `flex-direction` and `flex-wrap` properties, which together define the flex container's main and cross axes. The default value is `row nowrap`.

```
.container {
  flex-flow: column wrap;
}
```

```
// In this example code block, "column" is the value of the property "flex-direction" and "wrap" is the value of the property "flex-wrap".

.container {
  display: flex;
  flex-flow: column wrap;
}
```

The CSS `flex-flow` property

The CSS property `flex-flow` provides a shorthand for the properties `flex-direction` and `flex-wrap`. The value of the `flex-direction` property specifies the direction of the flex items and the value of the `flex-wrap` property allows flex items to move to the next line instead of shrinking to fit inside the flex container. The `flex-flow` property should be declared on the flex container.

justify-content Property

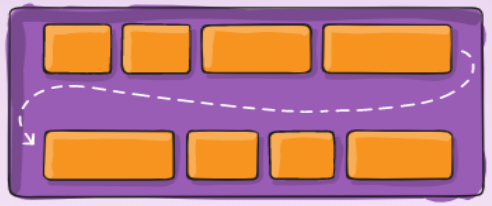
The CSS `justify-content` flexbox property defines how the browser distributes space between and around content items along the main-axis of their container. This is when the content items do not use all available space on the major-axis (horizontally).

`justify-content` can have the values of:

- `flex-start`
- `flex-end`
- `center`
- `space-between`
- `space-around`

```
/* Items based at the center of the parent container: */
div {
  display: flex;
  justify-content: center;
}

/* Items based at the upper-left side of the parent container: */
div {
  display: flex;
  justify-content: flex-start;
}
```

flex-wrap



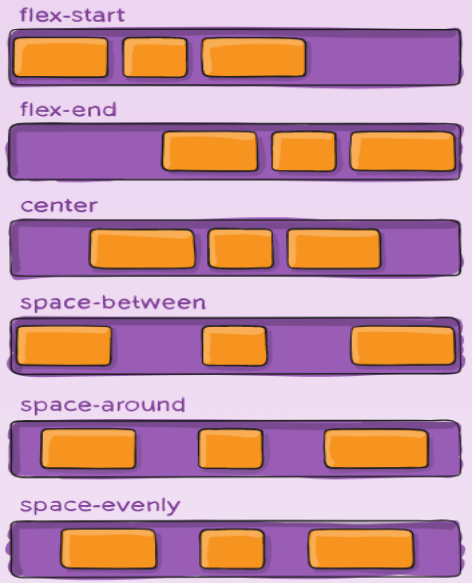
By default, flex items will all try to fit onto one line. You can change that and allow the items to wrap as needed with this property.

```css
.container {
  flex-wrap: nowrap | wrap | wrap-reverse;
}
```

- `nowrap` (default): all flex items will be on one line
- `wrap` : flex items will wrap onto multiple lines, from top to bottom.
- `wrap-reverse` : flex items will wrap onto multiple lines from bottom to top.

There are some visual demos of `flex-wrap` here.

justify-content



This defines the alignment along the main axis. It helps distribute extra free space leftover when either all the flex items on a line are inflexible, or are flexible but have reached their maximum size. It also exerts some control over the alignment of items when they overflow the line.

```css
.container {
  justify-content: flex-start | flex-end | center | space-
}
```

- `flex-start` (default): items are packed toward the start of the flex-direction.
- `flex-end` : items are packed toward the end of the flex-direction.
- `start` : items are packed toward the start of the `writing-mode` direction.
- `end` : items are packed toward the end of the `writing-mode` direction.
- `left` : items are packed toward left edge of the container, unless that doesn't make sense with the `flex-direction` , then it behaves like `start` .
- `right` : items are packed toward right edge of the container, unless that doesn't make sense with the `flex-direction` , then it behaves like `start` .
- `center` : items are centered along the line
- `space-between` : items are evenly distributed in the line; first item is on the start line, last item on the end line
- `space-around` : items are evenly distributed in the line with equal space around them. Note that visually the spaces aren't equal, since all the items have equal space on both sides. The first item will have one unit of space against the container edge, but two units of space between the next item because that next item has its own spacing that applies.
- `space-evenly` : items are distributed so that the spacing between any two items (and the space to the

flex-shrink

This defines the ability for a flex item to shrink if necessary.

```css
.item {
  flex-shrink: 3; /* default 1 */
}
```

Negative numbers are invalid.

HTML CSS Result EDIT ON CODEPEN

Css flex-wrap property

The `flex-wrap` property specifies whether flex items should wrap or not. This applies to flex items only. Once you tell your container to `flex-wrap`, wrapping become a priority over shrinking. Flex items will only begin to wrap if their combined `flex-basis` value is `greater` than the current size of their flex container.

```
.container {
  display: flex;
  flex-wrap: wrap;
  width: 200px;
}
```

Let's start with a very very simple example, solving an almost daily problem: perfect centering. It couldn't be any simpler if you use flexbox.

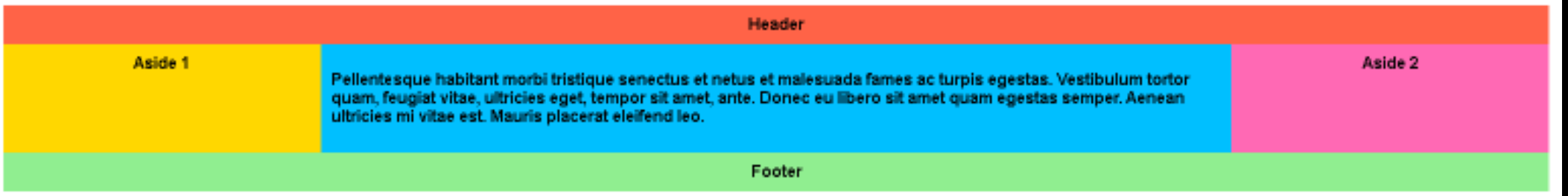
```
.parent {
  display: flex;
  height: 300px; /* Or whatever */
}

.child {
  width: 100px;  /* Or whatever */
  height: 100px; /* Or whatever */
  margin: auto;  /* Magic! */
}
```

CSS display: inline-flex property

The CSS `display: inline-flex` property sets an HTML element as an inline flex container which takes only the required space for the content. Any child elements that reside within the flex container are caleld flex items. Flex items change their size and location in response to the size and position of their parent container.

```
.container{
  display: inline-flex;
}
```

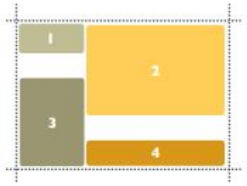
```
.flex-container {
  /* We first create a flex layout context */
  display: flex;

  /* Then we define the flow direction
     and if we allow the items to wrap
   * Remember this is the same as:
   * flex-direction: row;
   * flex-wrap: wrap;
   */
  flex-flow: row wrap;

  /* Then we define how is distributed the remaining space */
  justify-content: space-around;
}
```



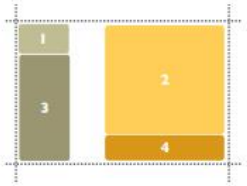

```css
.wrapper {
    display: flex;
    flex-flow: row wrap;
  }
  /* We tell all items to be 100% width, via flex-basis */
  .wrapper > * {
    flex: 1 100%;
  }
  /* We rely on source order for mobile-first approach
   * in this case:
   * 1. header
   * 2. article
   * 3. aside 1
   * 4. aside 2
   * 5. footer
   */
  /* Medium screens */
  @media all and (min-width: 600px) {
    /* We tell both sidebars to share a row */
    .aside { flex: 1 auto; }
  }
  /* Large screens */
  @media all and (min-width: 800px) {
    /* We invert order of first sidebar and main
     * And tell the main element to take twice as much width as the other two sidebars
     */
    .main { flex: 2 0px; }
    .aside-1 { order: 1; }
    .main    { order: 2; }
    .aside-2 { order: 3; }
    .footer  { order: 4; }
  }
```

gaps

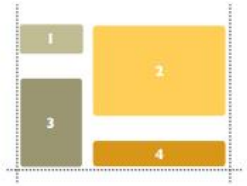
specifying size of row and column gutters



```
.parent {
  row-gap: 50px;
}
```



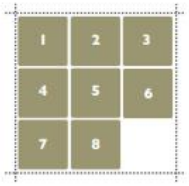
```
.parent {
  column-gap: 50px;
}
```



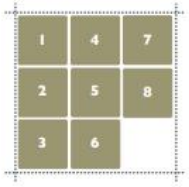
```
.parent {
  gap: 50px 10px;
}
```

auto placement

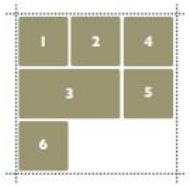
automatic placement for items that aren't placed



```
.parent {
  grid-auto-flow: row;
}
```



```
.parent {
  grid-auto-flow: column;
}
```



```
.parent {
  grid-auto-flow: dense;
}
```

placing items

grid-row-start
auto · line name · line number

grid-row-end
auto · line name · line number

grid-row
shortcut for grid-row-start|end

grid-column-start
auto · line name · line number

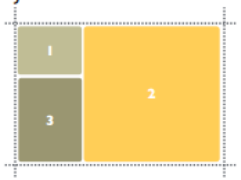
grid-column-end
auto · line name · line number

grid-column
shortcut for grid-column-*

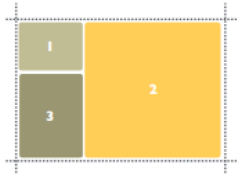
grid-area
shortcut for… everything

span *N*
keyword for row or column spanning

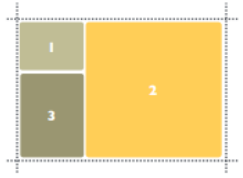
```
.two {
  grid-row-start: 1;
  grid-row-end: 3;
}
```



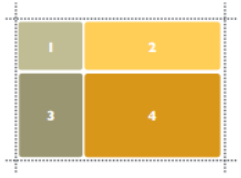
```
.two {
  grid-row: 1 / 3;
}
```



```
.two {
  grid-row: 1 / span 2;
}
```



```
.four {
  grid-area: 2 / 2 / 3 / 3;
}
```



Grid Template Columns

To specify the number of columns of the grid and the widths of each column, the CSS property `grid-template-columns` is used on the grid container. The number of width values determines the number of columns and each width value can be either in pixels(`px`) or percentages(%).

```
#grid-container {
  display: grid;
  width: 100px;
  grid-template-columns: 20px 20% 60%;
}
```

fr Relative Unit

The CSS grid relative sizing unit `fr` is used to split rows and/or columns into proportional distances. Each `fr` unit is a fraction of the grid's overall length and width. If a fixed unit is used along with `fr` (like pixels for example), then the `fr` units will only be proportional to the distance left over.

```
/*
In this example, the second column take 60px of the avaiable
100px so the first and third columns split the remaining
available 40px into two parts (`1fr` = 50% or 20px)
*/

.grid {
  display: grid;
  width: 100px;
  grid-template-columns: 1fr 60px 1fr;
}
```

Grid Gap

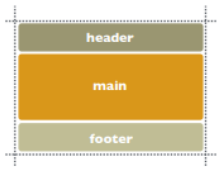
The CSS `grid-gap` property is a shorthand way of setting the two properties `grid-row-gap` and `grid-column-gap`. It is used to determine the size of the gap between each row and each column. The first value sets the size of the gap between rows and while the second value sets the size of the gap between columns.

```
// The distance between rows is 20px
// The distance between columns is 10px

#grid-container {
  display: grid;
  grid-gap: 20px 10px;
}
```

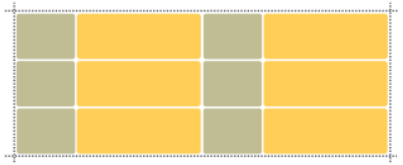
patterns

```
.parent {
  align-content: center;
  justify-content: center;
}
```

```
.parent {
  grid-auto-rows:
  auto 1fr auto;
}
```




```
.parent {
  grid-template-columns: repeat(2, 100px 1fr);
}
```



```
.parent {
  grid-template-columns: repeat(4, 22%);
  justify-content: space-between;
}
```

CSS grid-auto-rows

The CSS `grid-auto-rows` property specifies the height of implicitly added grid rows or it sets a size for the rows in a grid container. This property is declared on the grid container.

`grid-auto-columns` provides the same functionality for columns. Implicitly-added rows or columns occur when there are more grid items than cells available.

The CSS `justify-self` property is used to set how an individual grid item positions itself along the row or inline axis.

By default grid items inherit the value of the `justify-items` property on the container. So if the `justify-self` value is set, it would over-ride the inherited `justify-items` value.

The value `start` positions grid items on the left side of the grid area.

The value `end` positions the grid items on the right side of the grid area.

The value `center` positions grid items on the center of the grid area.

The value `stretch` positions grid items to fill the grid area (default).

```
// The grid items are positioned to the
right (end) of the row.

#grid-container {
  display: grid;
  justify-items: start;
}

.grid-items {
  justify-self: end;
}
```

Grid Gap

The CSS `grid-gap` property is a shorthand way of setting the two properties `grid-row-gap` and `grid-column-gap`. It is used to determine the size of the gap between each row and each column. The first value sets the size of the gap between rows and while the second value sets the size of the gap between columns.

```
// The distance between rows is 20px
// The distance between columns is 10px

#grid-container {
  display: grid;
  grid-gap: 20px 10px;
}
```

CSS Block Level Grid

CSS Grid is a two-dimensional CSS layout system. To set an HTML element into a block-level *grid container* use `display: grid` property/value. The nested elements inside this element are called *grid items*.

```
#grid-container {
  display: block;
}
```

CSS grid-row

The CSS `grid-row` property is shorthand for the `grid-row-start` and `grid-row-end` properties specifying a grid item's size and location within the grid row. The starting and ending row values are separated by a `/`. There is a corresponding `grid-column` property shorthand that implements the same behavior for columns.

```
/*CSS Syntax */
grid-row: grid-row-start / grid-row-end;

/*Example*/
.item {
  grid-row: 1 / span 2;
}
```

CSS Inline Level Grid

CSS Grid is a two-dimensional CSS layout system. To set an HTML element into a inline-level *grid container* use `display: inline-grid` property/value. The nested elements inside this element are called *grid items*.

The difference between the values `inline-grid` and `grid` is that the `inline-grid` will make the element inline while `grid` will make it a block-level element.

```
#grid-container {
  display: inline-grid;
}
```

Grid Container

The element on which `display: grid` is applied. It's the direct parent of all the grid items. In this example `container` is the grid container.

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

Grid Item

The children (i.e. *direct* descendants) of the grid container. Here the `item` elements are grid items, but `sub-item` isn't.

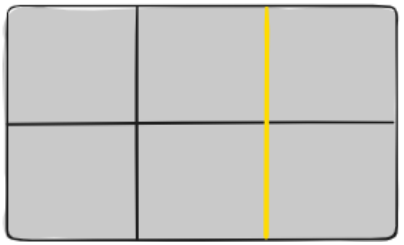
```HTML
<div class="container">
  <div class="item"> </div>
  <div class="item">
    <p class="sub-item"> </p>
  </div>
  <div class="item"> </div>
</div>
```

Grid Track

The space between two adjacent grid lines. You can think of them like the columns or rows of the grid. Here's the grid track between the second and third row grid lines.

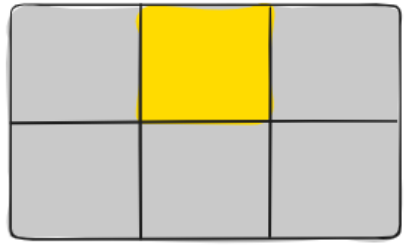
Grid Line

The dividing lines that make up the structure of the grid. They can be either vertical ("column grid lines") or horizontal ("row grid lines") and reside on either side of a row or column. Here the yellow line is an example of a column grid line.



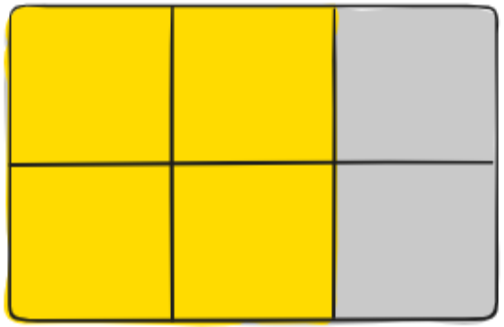
Grid Cell

The space between two adjacent row and two adjacent column grid lines. It's a single "unit" of the grid. Here's the grid cell between row grid lines 1 and 2, and column grid lines 2 and 3.



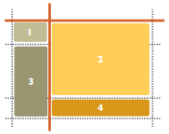
Grid Area

The total space surrounded by four grid lines. A grid area may be composed of any number of grid cells. Here's the grid area between row grid lines 1 and 3, and column grid lines 1 and 3.

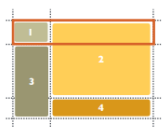


grid concepts and example

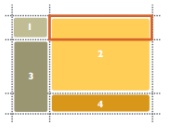
grid line



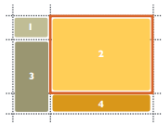
grid track



grid cell



grid area



```
.parent {
  display: grid;
  grid-template-columns: 100px 1fr;
  grid-template-rows: auto 1fr auto;
}

.one {
  grid-row: 1;
  grid-column: 1;
}

.two {
  grid-row: 1 / 3;
  grid-column: 2;
}

.three {
  grid-row: 2 / 4;
  grid-column: 1;
}

.four {
  grid-row: 3;
  grid-column: 2;
}
```

defining the grid

display
grid · inline-grid · subgrid

grid-template-rows
none · line name · track size · auto

grid-template-columns
none · line name · track size · auto

grid-template-areas
none · string

grid-template shortcut for grid-template-rows|-column
grid shortcut for… everything

display

Defines the element as a grid container and establishes a new grid formatting context for its contents.

Values:

- **grid** – generates a block-level grid
- **inline-grid** – generates an inline-level grid

```css
.container {
  display: grid | inline-grid;
}
```

CSS Inline Level Grid

CSS Grid is a two-dimensional CSS layout system. To set an HTML element into a inline-level *grid container* use `display: inline-grid` property/value. The nested elements inside this element are called *grid items*.

The difference between the values `inline-grid` and `grid` is that the `inline-grid` will make the element inline while `grid` will make it a block-level element.

grid-column-start
grid-column-end
grid-row-start
grid-row-end

Determines a grid item's location within the grid by referring to specific grid lines. `grid-column-start` / `grid-row-start` is the line where the item begins, and `grid-column-end` / `grid-row-end` is the line where the item ends.

Values:

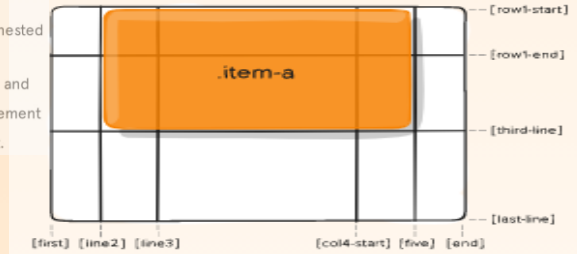
- **<line>** – can be a number to refer to a numbered grid line, or a name to refer to a named grid line
- **span <number>** – the item will span across the provided number of grid tracks
- **span <name>** – the item will span across until it hits the next line with the provided name
- **auto** – indicates auto-placement, an automatic span, or a default span of one

```
#grid-container {
  display: inline-grid;
}
```

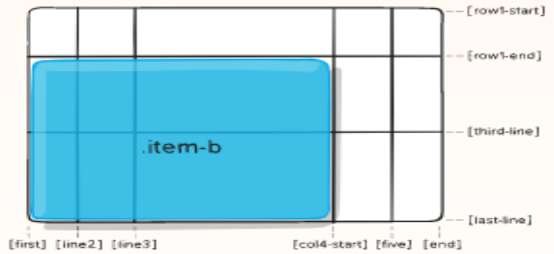
```css
.item {
  grid-column-start: <number> | <name> | span <number> | spa
  grid-column-end: <number> | <name> | span <number> | span
  grid-row-start: <number> | <name> | span <number> | span <
  grid-row-end: <number> | <name> | span <number> | span <na
}
```

Examples:

```css
.item-a {
  grid-column-start: 2;
  grid-column-end: five;
  grid-row-start: row1-start;
  grid-row-end: 3;
}
```




```css
.item-b {
  grid-column-start: 1;
  grid-column-end: span col4-start;
  grid-row-start: 2;
  grid-row-end: span 2;
}
```



If no `grid-column-end` / `grid-row-end` is declared, the item will span 1 track by default.

Items can overlap each other. You can use `z-index` to control their stacking order.

CSS grid-template-areas

The CSS `grid-template-areas` property allows the naming of sections of a webpage to use as values in the `grid-row-start`, `grid-row-end`, `grid-column-start`, `grid-column-end`, and `grid-area` properties. They specify named grid areas within a CSS grid.

```css
/* Specify two rows, where "item" spans the first two columns in the first two rows (in a four column grid layout)*/
.item {
  grid-area: nav;
}
.grid-container {
  display: grid;
  grid-template-areas:
    'nav nav . .'
    'nav nav . .';
}
```

CSS grid-row-gap

The CSS `grid-row-gap` property determines the amount of blank space between each row in a CSS grid layout or in other words, sets the size of the gap (gutter) between an element's grid rows. The `grid-column-gap` provides the same functionality for space between grid columns.

```
/*CSS syntax */
grid-row-gap: length; /*Any legal length value, like px or %. 0 is the default value*/
```

grid-template-columns grid-template-rows

Defines the columns and rows of the grid with a space-separated list of values. The values represent the track size, and the space between them represents the grid line.

Values:

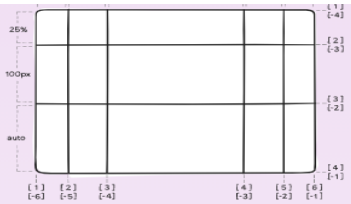
- **<track-size>** – can be a length, a percentage, or a fraction of the free space in the grid (using the `fr` unit)
- **<line-name>** – an arbitrary name of your choosing

```css
.container {
  grid-template-columns:  ... |  ...;
  grid-template-rows:  ... |  ...;
}
```

Examples:

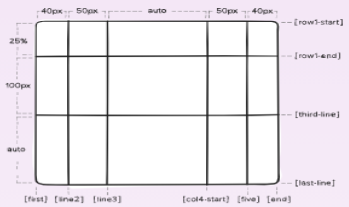
When you leave an empty space between the track values, the grid lines are automatically assigned positive and negative numbers:

```css
.container {
  grid-template-columns: 40px 50px auto 50px 40px;
  grid-template-rows: 25% 100px auto;
}
```



But you can choose to explicitly name the lines. Note the bracket syntax for the line names:

```css
.container {
  grid-template-columns: [first] 40px [line2] 50px [line3]
  grid-template-rows: [row1-start] 25% [row1-end] 100px [th
}
```



The `fr` unit allows you to set the size of a track as a fraction of the free space of the grid container. For example, this will set each item to one third the width of the grid container:

```
.container {
  grid-template-columns: 1fr 1fr 1fr;
}
```

The free space is calculated *after* any non-flexible items. In this example the total amount of free space available to the `fr` units doesn't include the 50px:

```
.container {
  grid-template-columns: 1fr 50px 1fr 1fr;
}
```

Note that a line can have more than one name. For example, here the second line will have two names: row1-end and row2-start:

```
.container {
  grid-template-rows: [row1-start] 25% [row1-end row2-start] 25% [
}
```

If your definition contains repeating parts, you can use the `repeat()` notation to streamline things:

```
.container {
  grid-template-columns: repeat(3, 20px [col-start]);
}
```

Which is equivalent to this:

```
.container {
  grid-template-columns: 20px [col-start] 20px [col-start] 20px [
}
```

If multiple lines share the same name, they can be referenced by their line name and count.

```
.item {
  grid-column-start: col-start 2;
}
```

CSS grid-area

The CSS `grid-area` property specifies a grid item's size and location in a grid layout and is a shorthand property for the `grid-row-start`, `grid-column-start`, `grid-row-end`, and `grid-column-end` in that order. Each value is separated by a `/`.

In the included example, `item1` will start on row 2 and column 1, and span 2 rows and 3 columns

```
.item1 {
  grid-area: 2 / 1 / span 2 / span 3
}
```

CSS grid-area

The CSS `grid-area` property allows for elements to overlap each other by using the `z-index` property on a particular element which tells the browser to render that element on top of the other elements.

grid-column
grid-row

Shorthand for `grid column start` + `grid column end`, and `grid row start` + `grid row end`, respectively.

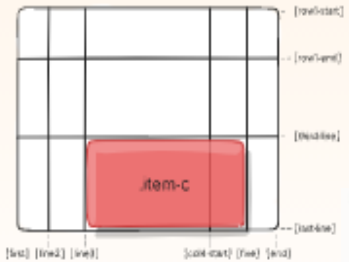
Values:

- <start-line> / <end-line> – each one accepts all the same values as the longhand version, including span

```
.item {
  grid-column: <start-line> / <end-line> | <start-line> / span <v
  grid-row: <start-line> / <end-line> | <start-line> / span <valu
}
```

Example:

```
.item-c {
  grid-column: 3 / span 2;
  grid-row: third-line / 4;
}
```



If no end line value is declared, the item will span 1 track by default.

grid-area

Gives an item a name so that it can be referenced by a template created with the `grid template areas` property. Alternatively, this property can be used as an even shorter shorthand for `grid row start` + `grid column start` + `grid row end` + `grid column end`.

Values:

- <name> – a name of your choosing
- <row-start> / <column-start> / <row-end> / <column-end> – can be numbers or named lines

```
.item {
  grid-area: <name> | <row-start> / <column-start> / <row-end> / <
}
```

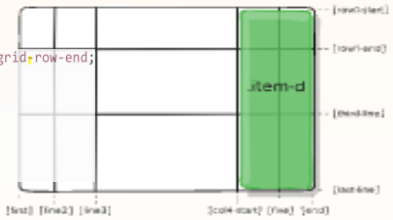
Examples:

As a way to assign a name to the item:

```
.item-d {
  grid-area: header;
}
```

As the short-shorthand for `grid row start` + `grid column start` + `grid row end` + `grid column end`:

```
.item-d {
  grid-area: 1 / col4-start / last-line / 6;
}
```




```
grid-row: grid-row-start / grid-row-end;

/*Example*/
.item {
  grid-row: 1 / span 2;
}
```

grid-template-areas

Defines a grid template by referencing the names of the grid areas which are specified with the `grid area` property. Repeating the name of a grid area causes the content to span those cells. A period signifies an empty cell. The syntax itself provides a visualization of the structure of the grid.

Values:

- <grid-area-name> – the name of a grid area specified with `grid area`
- . – a period signifies an empty grid cell
- none – no grid areas are defined

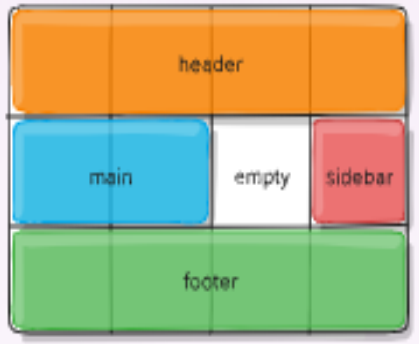
```
.container {
  grid-template-areas:
    " | . | none | ..."
    "...";
}
```

Example:

```
.item-a {
  grid-area: header;
}
.item-b {
  grid-area: main;
}
.item-c {
  grid-area: sidebar;
}
.item-d {
  grid-area: footer;
}

.container {
  display: grid;
  grid-template-columns: 50px 50px 50px 50px;
  grid-template-rows: auto;
  grid-template-areas:
    "header header header header"
    "main main . sidebar"
    "footer footer footer footer";
}
```

```
/* Specify two rows, where "item" spans the first two columns
in the first two rows (in a four column grid layout)*/
.item {
  grid-area: nav;
}
.grid-container {
  display: grid;
  grid-template-areas:
    'nav nav . .'
    'nav nav . .';
}
```

That'll create a grid that's four columns wide by three rows tall. The entire top row will be composed of the header area. The middle row will be composed of two main areas, one empty cell, and one sidebar area. The last row is all footer.



Each row in your declaration needs to have the same number of cells.

You can use any number of adjacent periods to declare a single empty cell. As long as the periods have no spaces between them they represent a single cell.

Notice that you're not naming lines with this syntax, just areas. When you use this syntax the lines on either end of the areas are actually getting named automatically. If the name of your grid area is *foo*, the name of the area's starting row line and starting column line will be *foo*-start, and the name of its last row line and last column line will be *foo*-end. This means that some lines might have multiple names, such as the far left line in the above example, which will have three names: header-start, main-start, and footer-start.

justify-self

Aligns a grid item inside a cell along the *inline (row)* axis (as opposed to `align self` which aligns along the *block (column)* axis). This value applies to a grid item inside a single cell.

Values:

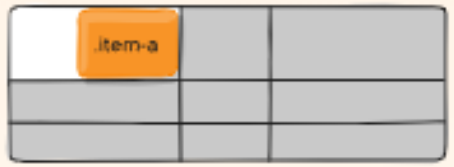
- start – aligns the grid item to be flush with the start edge of the cell
- end – aligns the grid item to be flush with the end edge of the cell
- center – aligns the grid item in the center of the cell
- stretch – fills the whole width of the cell (this is the default)

```
.item {
  justify-self: start | end | center | stretch;
}
```

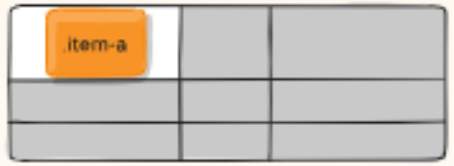
Examples:

```
.item-a {
  justify-self: start;
}
```

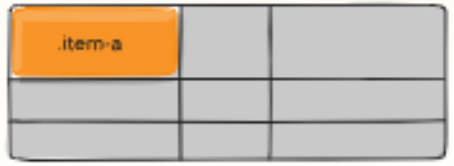

```
.item-a {
  justify-self: end;
}
```




```
.item-a {
  justify-self: center;
}
```




```
.item-a {
  justify-self: stretch;
}
```



align-self

Aligns a grid item inside a cell along the *block (column)* axis (as opposed to `justify self` which aligns along the *inline (row)* axis). This value applies to the content inside a single grid item.

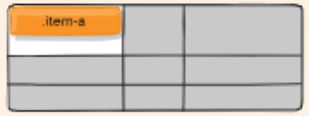
Values:

- start – aligns the grid item to be flush with the start edge of the cell
- end – aligns the grid item to be flush with the end edge of the cell
- center – aligns the grid item in the center of the cell
- stretch – fills the whole height of the cell (this is the default)

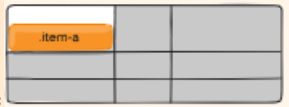
```
.item {
  align-self: start | end | center | stretch;
}
```

Examples:

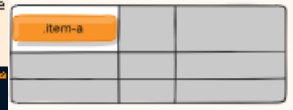
```
.item-a {
  align-self: start;
}
```



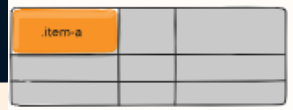

```
.item-a {
  align-self: end;
}
```




```
.item-a {
  align-self: center;
}
```




```
.item-a {
  align-self: stretch;
}
```



To align *all* the items in a grid, this behavior can also be set on the grid container via the `align items` property.

container alignment

align-content

vertical alignment for whole grid

start · end · center · stretch · space-around
space-between · space-evenly

justify-content

horizontal alignment for whole grid

start · end · center · stretch · space-around
space-between · space-evenly

place-content

shortcut for align-content and justify-content

align-items

vertical alignment for items in their cells

start · end · center · stretch

justify-items

horizontal alignment for items in their cells

start · end · center · stretch

place-items

shortcut for align-items and justify-items

```
#container {
  display: grid;
  justify-items: center;
  grid-template-columns: 1fr;
  grid-template-rows: 1fr 1fr 1fr;
  grid-gap: 10px;
}
```

justify-items

Aligns grid items along the *inline (row)* axis (as opposed to `align-items` which aligns along the *block (column)* axis). This value applies to all grid items inside the container.

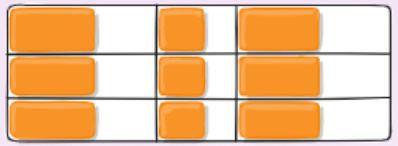
Values:

- **start** – aligns items to be flush with the start edge of their cell
- **end** – aligns items to be flush with the end edge of their cell
- **center** – aligns items in the center of their cell
- **stretch** – fills the whole width of the cell (this is the default)

```css
.container {
  justify-items: start | end | center | stretch;
}
```

Examples:

```css
.container {
  justify-items: start;
}
```



```css
.container {
  justify-items: end;
}
```

```css
.container {
  justify-items: center;
}
```

```css
.container {
  justify-items: stretch;
}
```

This behavior can also be set on individual grid items via the `justify-self` property.

align-items

Aligns grid items along the *block (column)* axis (as opposed to `justify-items` which aligns along the *inline (row)* axis). This value applies to all grid items inside the container.

Values:

- **start** – aligns items to be flush with the start edge of their cell
- **end** – aligns items to be flush with the end edge of their cell
- **center** – aligns items in the center of their cell
- **stretch** – fills the whole height of the cell (this is the default)

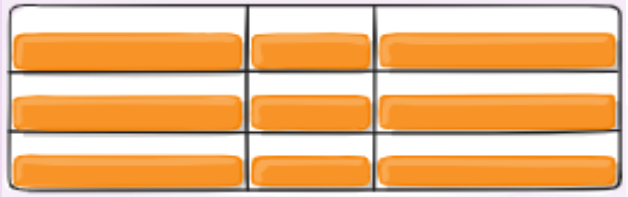
```css
.container {
  align-items: start | end | center | stretch;
}
```

Examples:

```css
.container {
  align-items: start;
}
```

```
#container {
  display: grid;
  align-items: start;
  grid-template-columns: 1fr;
  grid-template-rows: 1fr 1fr 1fr;
  grid-gap: 10px;
}
```

```
.container {
  align-items: end;
}
```



```
.container {
  align-items: center;
}
```

Align Items

The `align-items` property is used on a grid container. It's used to determine how the grid items are spread out along the column by setting the default `align-self` property for all child grid items.

The value `start` aligns grid items to the top side of the grid area.

The value `end` aligns grid items to the bottom side of the grid area.

The value `center` aligns grid items to the center of the grid area.

The value `stretch` stretches all items to fill the grid area.

```
.container {
  align-items: stretch;
}
```

justify-content

Sometimes the total size of your grid might be less than the size of its grid container. This could happen if all of your grid items are sized with non-flexible units like `px`. In this case you can set the alignment of the grid within the grid container. This property aligns the grid along the *inline (row)* axis (as opposed to `align-content` which aligns the grid along the *block (column)* axis).

Values:

- **start** – aligns the grid to be flush with the start edge of the grid container
- **end** – aligns the grid to be flush with the end edge of the grid container
- **center** – aligns the grid in the center of the grid container
- **stretch** – resizes the grid items to allow the grid to fill the full width of the grid container
- **space-around** – places an even amount of space between each grid item, with half-sized spaces on the far ends
- **space-between** – places an even amount of space between each grid item, with no space at the far ends
- **space-evenly** – places an even amount of space between each grid item, including the far ends

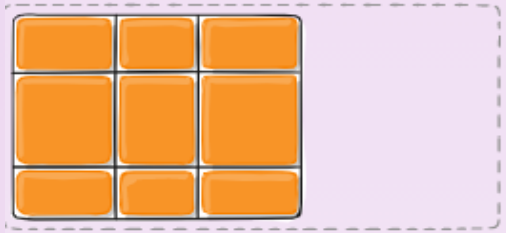
```
// The grid items are positioned to the right (end) of the row.

#grid-container {
  display: grid;
  justify-items: start;
}

.grid-items {
  justify-self: end;
}
```

```
.container {
  justify-content: start | end | cen
}
```

Examples:

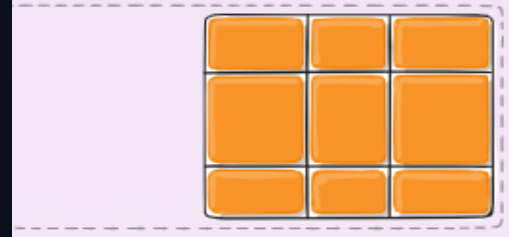
```
.container {
  justify-content: start;
}
```

grid container



```
.container {
  justify-content: end;
}
```

grid container



```
.container {
  justify-content: center;
}
```

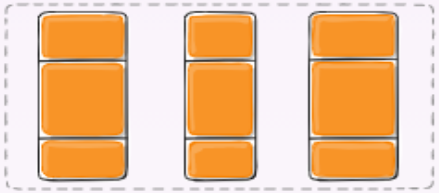
grid container

```
.container {
  justify-content: stretch;
}
```

grid container

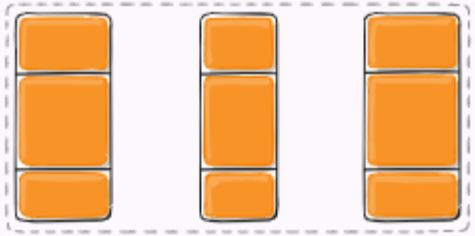
```
.container {
  justify-content: space-around;
}
```

grid container



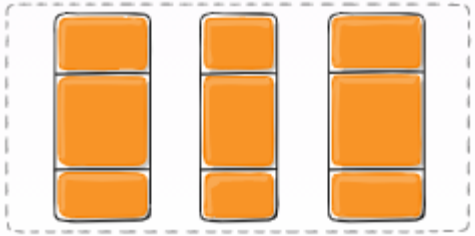
```
.container {
  justify-content: space-between;
}
```

grid container



```
.container {
  justify-content: space-evenly;
}
```

grid container



align-content

Sometimes the total size of your grid might be less than the size of its grid container. This could happen if all of your grid items are sized with non-flexible units like `px`. In this case you can set the alignment of the grid within the grid container. This property aligns the grid along the *block (column)* axis (as opposed to `justify-content` which aligns the grid along the *inline (row)* axis).

Values:

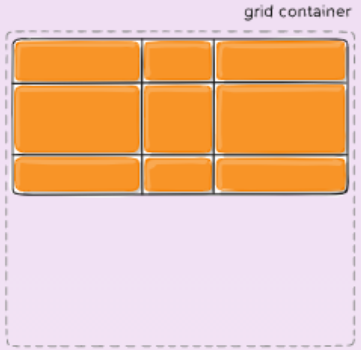
- **start** – aligns the grid to be flush with the start edge of the grid container
- **end** – aligns the grid to be flush with the end edge of the grid container
- **center** – aligns the grid in the center of the grid container
- **stretch** – resizes the grid items to allow the grid to fill the full height of the grid container
- **space-around** – places an even amount of space between each grid item, with half-sized spaces on the far ends
- **space-between** – places an even amount of space between each grid item, with no space at the far ends
- **space-evenly** – places an even amount of space between each grid item, including the far ends

```css
.container {
  align-content: start | end | center | stretch | space-ar
}
```

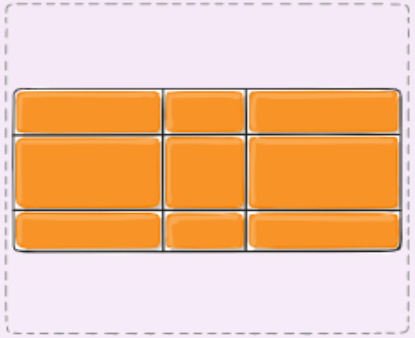
```
#container {
  display: grid;
  align-items: start;
  grid-template-columns: 1fr;
  grid-template-rows: 1fr 1fr 1fr;
  grid-gap: 10px;
}
```

Examples:

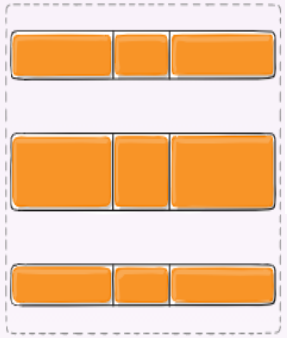
```
.container {
  align-content: start;
}
```



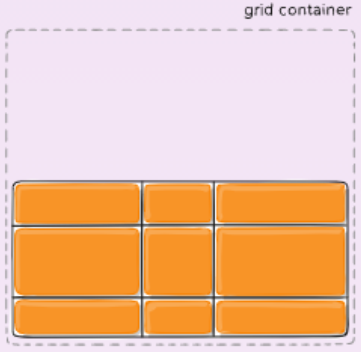

```
.container {
  align-content: center;
}
```



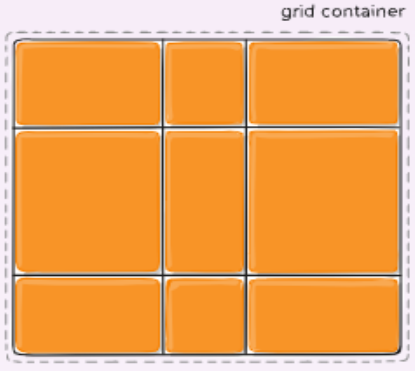

```
.container {
  align-content: space-around;
}
```



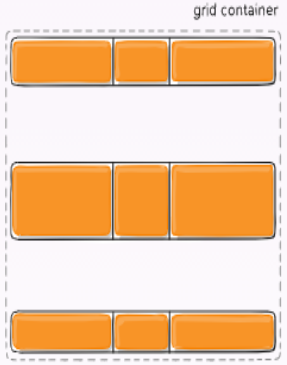

```
.container {
  align-content: end;
}
```




```
.container {
  align-content: stretch;
}
```




```
.container {
  align-content: space-between;
}
```



Align Items

The `align-items` property is used on a grid container. It's used to determine how the grid items are spread out along the column by setting the default `align-self` property for all child grid items.

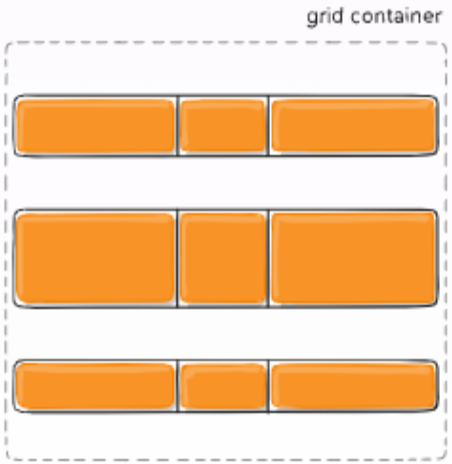
The value `start` aligns grid items to the top side of the grid area.

The value `end` aligns grid items to the bottom side of the grid area.

The value `center` aligns grid items to the center of the grid area.

The value `stretch` stretches all items to fill the grid area.

```css
.container {
  align-content: space-evenly;
}
```



CSS grid-auto-flow

The CSS `grid-auto-flow` property specifies whether implicity-added elements should be added as rows or columns within a grid or, in other words; it controls how auto-placed items get inserted in the grid and this property is declared on the grid container.

The value `row` specifies the new elements should fill rows from left to right and create new rows when there are too many elements (default).

The value `column` specifies the new elements should fill columns from top to bottom and create new columns when there are too many elements.

The value `dense` invokes an algorithm that attempts to fill holes earlier in the grid layout if smaller elements are added.

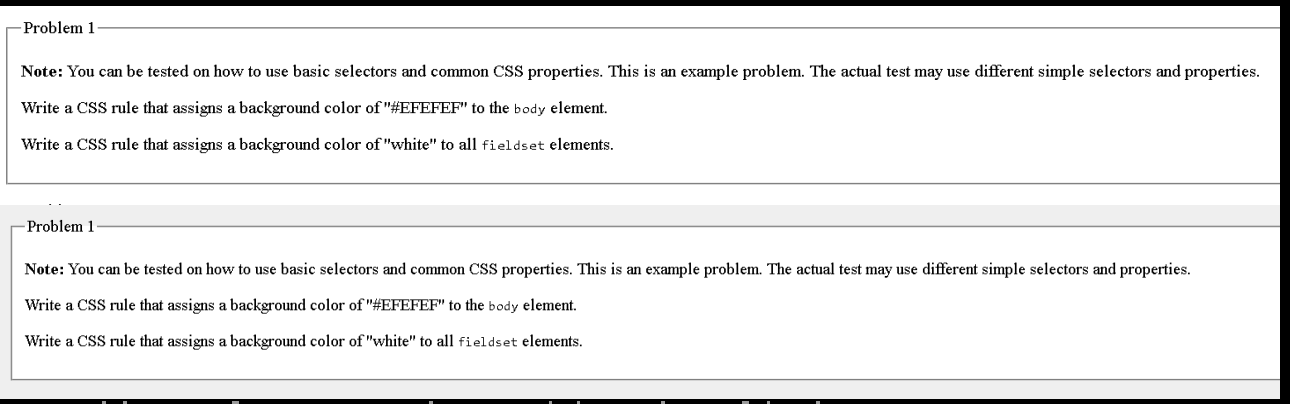
```
/*CSS Syntax */
grid-auto-flow: row|column|dense|row
dense|column dense;
```

```
const handleResponse= async ( res ) => {
    stopLoader();
    clearError();
    if ( !res.ok ) {
        throw res;
    }
    const data = await res.json()
    return data
};
const fetchImage= async () => {
    startLoader();
    try {
        const res = await fetch( 'http://localhost:3000/kitten/image'
        const data = await handleResponse( res )
        document.querySelector( '.cat-pic' ).src = data.src;
        document.querySelector( '.score' ).innerHTML= data.score;
        document.querySelector( '.comments' ).innerHTML= '';
    } catch ( e ) {
        handleError( e )
    };
};
const startLoader= () => {
    document.querySelector( '.loader' ).innerHTML= 'Loading...';
};
const stopLoader= () => {
    document.querySelector( '.loader' ).innerHTML= '';
};
const handleError= async ( error ) => {
    const errJSON= await error.json()
    if ( errJSON ) {
        document.querySelector( '.error' ).innerHTML= `Error occured:${errJSON.message}`;
    } else {
        console.error( error );
        alert( 'Something went wrong. Please try again');
    }
};
```

```
const clearError = () => {
    document.querySelector( '.error' ).innerHTML = '';
};

const vote = async ( e ) => {
    try {
        const res = await fetch( `http://localhost:3000/kitte${e.target.id}`, {
            method:'PATCH'
        } );
        const data = await handleRespons( res )
        const {
            score
        } = data;
        document.querySelector( '.score' ).innerHTML = score;
    } catch ( e ) {
        handleError( e )
    }
}
```

```
const receiveComments= ( data ) => {
    const comments = document.querySelector( '.comments' );
    comments.innerHTML = '';
    data.comments.forEach( ( comment, i ) => {
        const newCommentContainer= document.createElement( 'div' );
        newCommentContainer.className = 'comment-container';
        const newComment = document.createElement( 'p' );
        newComment.appendChild( document.createTextNode( comment ) );
        const deleteButton= document.createElement( 'button' );
        deleteButton.appendChild( document.createTextNode( 'Delete' ) );
        deleteButton.className = 'delete-button';
        deleteButton.setAttribute( 'id', i );
        newCommentContainer.appendChild( newComment );
        newCommentContainer.appendChild( deleteButton);
        comments.appendChild( newCommentContainer);
    } );
};
```

```
const commentForm= document.querySelecto( '.comment-form');
const submitComment= async ( event ) => {
    event.preventDefaul();
    const formData = new FormData( commentForm );
    const comment = formData.get( 'user-comment');
    try {
        const res = await fetch( 'http://localhost:3000/kitten/comments',{
            method:'POST',
            headers: {
                'Content-Type' 'application/json'
            },
            body:JSON.stringify( {
                comment
            } )
        } );
        const data = await handleRespons( res )
        commentForm.reset();
        receiveComment( data );
    } catch ( e ) {
        handleError( e )
    }
}
        }
        catch ( e ) {
            handleError( e )
        }
}
window.addEventListener( 'DOMContentLoaded', fetchImage );
document.querySelector( '#new-pic' ).addEventListener( 'click', fetchImage );
document.querySelector( '#upvote' ).addEventListener( 'click', vote );
document.querySelector( '#downvote' ).addEventListener( 'click', vote );
commentForm.addEventListener( 'submit', submitComment );
document.querySelector( '.comments' ).addEventListener( 'click', deleteComment );
```



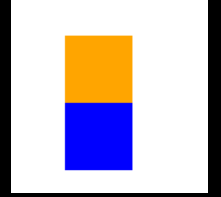
Problem 1
Note: You can be tested on how to use basic selectors and common CSS properties. This is an example problem. The actual test may use different simple selectors and properties.
Write a CSS rule that assigns a background color of "#EFEFEF" to the body element.
Write a CSS rule that assigns a background color of "white" to all fieldset elements.

You, a few seconds ago | 1 author (You)

```
1  body{
2      background-color: #efefef;
3  }
4
5  fieldset{
6      background-color: white;
7  }
8
```

You, a few seconds ago • Uncomm

```css
.outer-layer-2 {
    display: flex;
    flex-direction: column-reverse;
}
```



Problem 2

Note: You can be tested on how to use positioning. This is an example problem. The actual test may use different positioning and properties to affect layout.

There are three elements, here: An element with a class of "outer-layer-2" that has two children elements, one with a class of "blue-box" and the other with the class "orange-box".

Make the "blue-box" a square 40 pixels in width and height with a background color of "blue".

Make the "orange-box" the same size as the blue box. Make the "orange-box" have a background color of "orange".

Position the "orange-box" above the "blue-box".

Make sure you use "border-box" box sizing for "orange-box" and "blue-box".

Problem 2

Note: You can be tested on how to use positioning. This is an example problem. The actual test may use different positioning

There are three elements, here: An element with a class of "outer-layer-2" that has two children elements, one with a class o

Make the "blue-box" a square 40 pixels in width and height with a background color of "blue".

Make the "orange-box" the same size as the blue box. Make the "orange-box" have a background color of "orange".

Position the "orange-box" above the "blue-box".

Make sure you use "border-box" box sizing for "orange-box" and "blue-box".

```css
.blue-box,
.orange-box {
    width: 40px;
    height: 40px;
    box-sizing: border-box;
}

.blue-box {
    background-color: blue;
}

.orange-box {
    background-color: orange;
}
```

```css
/* Example using absolute positioning */
.outer-layer-2 {
    position: relative;
}
.orange-box,
.blue-box {
    position: absolute;
}
.orange-box {
    top: -40px;
}

/* Problem 3 */
.outer-layer-3 {
    display: flex;
    flex-direction: column;
    align-items: flex-end;
}
```

HTML

```html
<div class="content-box">Content box</div>
<br>
<div class="border-box">Border box</div>
```

```css
div {
  width: 160px;
  height: 80px;
  padding: 20px;
  border: 8px solid red;
  background: yellow;
}

.content-box {
  box-sizing: content-box;
  /* Total width: 160px + (2 * 20px) + (2 * 8px) = 216px
     Total height: 80px + (2 * 20px) + (2 * 8px) = 136px
     Content box width: 160px
     Content box height: 80px */
}

.border-box {
  box-sizing: border-box;
  /* Total width: 160px
     Total height: 80px
     Content box width: 160px - (2 * 20px) - (2 * 8px) = 104px
     Content box height: 80px - (2 * 20px) - (2 * 8px) = 24px */
}
```

border-box

The `width` and `height` properties include the content, padding, and border, but do not include the margin. Note that padding and border will be inside of the box. For example, `.box {width: 350px; border: 10px solid black;}` renders a box that is 350px wide, with the area for content being 330px wide. The content box can't be negative and is floored to 0, making it impossible to use `border-box` to make the element disappear.

Here the dimensions of the element are calculated as: *width = border + padding + width of the content*, and *height = border + padding + height of the content*.

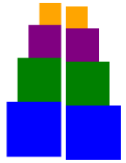
inline elements do NOT start on a new line and only takes up as much width as its content. So, if you try to set any width and height, it will have NO effects.

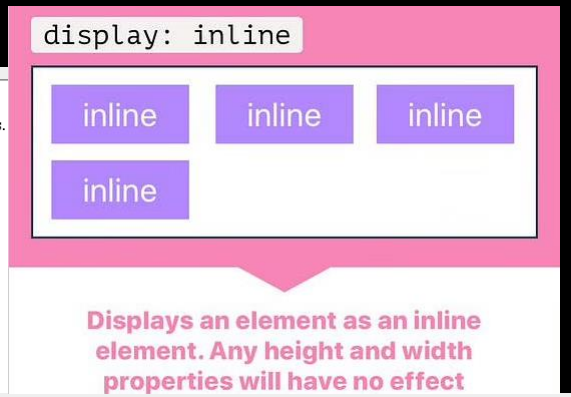
Problem 3

Note: You can be tested on how to use Flexible Box Model to affect layout. This is an example problem. The actual test may use different properties and selectors.

The element with the class "outer-layer-3" contains the already styled elements in this problem.

Apply the Flexible Box Model to "outer-layer-3" such that its internal elements are laid out vertically and right-justified.



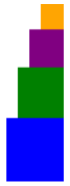


Problem 3

Note: You can be tested on how to use Flexible Box Model to affect layout. This is an example problem. The actual test may use different properties and selectors.

The element with the class "outer-layer-3" contains the already styled elements in this problem.

Apply the Flexible Box Model to "outer-layer-3" such that its internal elements are laid out vertically and right-justified.



```
<element class='outer-layer-3'>
.outer-layer-3 {
    display: flex;
    flex-direction: column;
    align-items: flex-end;
}
```

You, a few seconds ag

Problem 4

Note: You can be tested on **any and all** of the combinators and selectors. This one uses the adjacent sibling combinator.

There is an HTML element with a class "outer-layer-4". It has three paragraph child elements.

Use the *adjacent sibling combinator* + to make the text in the last two P tags the color orange.

Leave me alone

Orange me

Orange me

Leave me alone

Orange me

Orange me

```
.outer-layer-4 p + p {
    color: orange;
}
```

You, a few seconds a

Leave me alone

Orange me

Orange me

CSS

```
li:first-of-type + li {
  color: red;
}
```

HTML

```
<ul>
  <li>One</li>
  <li>Two!</li>
  <li>Three</li>
</ul>
```

Result

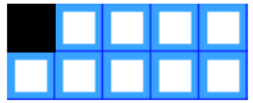
- One
- Two!
- Three

Problem 5

Note: You can be tested on any of the things that you can do to affect the layout of elements inside a Grid Layout.

The black square has the "inner-layer-5" class assigned to it. It lives in a grid of five columns and two rows.

Create a CSS rule that targets "inner-layer-5" to have it cover the last four squares of the first row.



Create a CSS rule that targets "inner-layer-5" to have it cover the last four squares of the first row.

```
.inner-layer-5 {
    grid-column: 2 / span 4;
}
```

You, a few seconds ago

HTML

```html
<div id="grid">
  <div id="item1"></div>
  <div id="item2"></div>
  <div id="item3"></div>
</div>
```

CSS

```css
#grid {
  display: grid;
  height: 100px;
  grid-template-columns: repeat(6, 1fr);
  grid-template-rows: 100px;
}

#item1 {
  background-color: lime;
}

#item2 {
  background-color: yellow;
  grid-column: 2 / 4;
}

#item3 {
  background-color: blue;
  grid-column: span 2 / 7;
}
```

Result

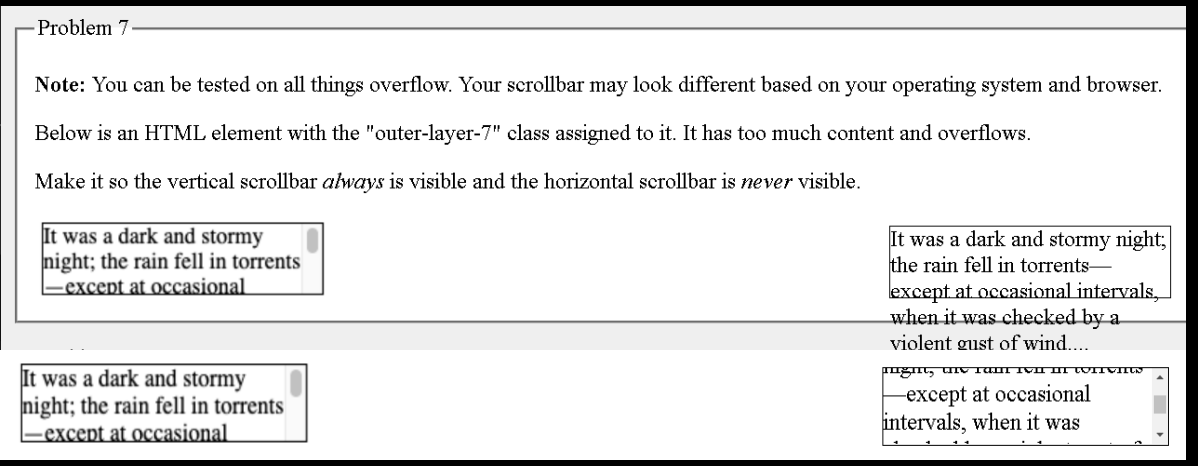
Problem 6

Note: You can be tested on all things box shadow.

Below is an HTML element with the class "outer-layer-6" assigned to it with a background color of "cyan".

Apply a shadow to the element that has a blur radius of four pixels, and a background color of blue with an opacity value of 0.6. (Use the `rgba` function to make that color.)

```
/*/ ---------------------- #6 ---------------------- */
.outer-layer-6 {
    box-shadow: 0 0 10px rgba(0, 0, 255, 0.6);
}
```




```css
/* Problem 7 */
.outer-layer-7 {
    overflow-x: hidden;
    overflow-y: scroll;
}
```

Problem 8

Note: You can be tested on **any and all** of the combinators and selectors. This one uses a compound class selector.

Note: The cursive font on your computer may look different based on your operating system and browser. Don't fret the cursive.

Below are two elements. The first has both the "cursive" and "bold" classes assigned to it. The second has both the "uppercase" and "bold" classes assigned to it.

Write a CSS rule that uses a *compound class selector* to target only the element that is both uppercase and bold to set the font family to "Courier New".

Cursive and bold
UPPERCASE AND BOLD

Cursive and bold
UPPERCASE AND BOLD

Cursive and bold
UPPERCASE AND BOLD

Cursive and bold
UPPERCASE AND BOLD

```
/* Problem 8 */
.uppercase.bold {
    font-family: 'Courier New';
}
```

HIDE GOT IT! How do you import other CSS files into your CSS file?

```
@import 'custom.css';
```

HIDE GOT IT! What two attributes must a link element have to link a stylesheet in your HTML page?

```
<link rel="stylesheet" href="file.css" />
```

HIDE GOT IT! Order the specificity of the following:

1. Last style for that element read in the browser
2. The ID tag (eg: #main-header)
3. The most element tags (eg: li, p, span)
4. The most classes (eg: .myClass)

1. The most ID tags (eg: #main-header)
2. The most classes (eg: .myClass)
3. The most element tags(eg: p, h1, li)
4. The last style for that element read in the browser.

HIDE GOT IT! Which of the following has the most specificity?

1. #main-header.large.on
2. div#main-header.header

#main-header.large.on has more specificity:

1. #main-header.large.on: 1-2
2. div#main-header.header: 1-1

HIDE GOT IT! Write a selector combinator that that creates the following style:

- First Child
    1. Different parent - an ol
    2. Different parent - an ol
- Second Child
- Third Child

All decendants selector: space

```
ul li {
    color: red;
}
```

- Second Child
- Third Child

All decendants selector: space

```
ul li {
    color: red;
}
```

HIDE GOT IT! Write a selector that will apply the same style to your body, div, paragraph, span, unordered list and line item elements.

The multiple items selection: comma

```
body, div, p, span, ul, li {
    color: red;
}
```

HIDE GOT IT! What selector combinator would you use to create the following style:

- First Child
    1. Different parent - ol
    2. Different parent - ol
- Second Child
- Third Child

The direct child selector: >

```
ul > li {
    color: red;
}
```

HIDE GOT IT! What is a pseudo-class?

A pseudo-class is a selector that selects elements that are in a specific state. e.g. :hover or :first-child

SHOW GOT IT! What are the following called?

1. ::first-line
2. ::before
3. ::after

SHOW GOT IT! Define what the following do and their parameters:

- color:
- background-color:

HIDE GOT IT! What types of shadows can be applied?

text-shadow or box-shadow

HIDE GOT IT! What does opacity do?

It sets the element's opacity. The lower the value the more transparent the element becomes. Values range from 0 (completely transparent) to 1;

HIDE GOT IT! What property do you need to set the image in the background?

background-image: url(<relative path or url>) or rel path to image

HIDE GOT IT! Why is using web fonts valuable?

Without loading web fonts, you are depending upon the browser to have the font you specify -- which may not be the case. Either downloading a font, or using a web font alleviates that problem and you can be assured the font will always be available.

HIDE GOT IT! What is a rem?

This is a relative length unit in CSS. It is based upon the root's value.

HIDE GOT IT! What is an em?

This is a relative length unit in CSS. It is based relative to its parent.

HIDE GOT IT! What are the 3 absolute length measures in CSS?

- pt: point
- px: pixel
- cm: centimeter

GOT IT! What does AJAX stand for?

Asynchronous JavaScript and XML

HIDE GOT IT! What is an AJAX request?

AJAX is a group of different technologies that work together to allow a website to communicate with a server in the background without requiring the website to reload in order to display new changes.

HIDE GOT IT! What are the advantages of using an AJAX request?

- We don't have to refresh the entire HTML page.
- It's a smaller amount of data that needs to be transferred.

HIDE GOT IT! Describe how AJAX relates to modern web programming.

- Asychronous: We don't lock up the page when we're waiting on a response. We're still able to interact and the response's data will be handled whenever it returns.
- JavaScript: The engine behind AJAX. We use JavaScript to make the request to the server, then we also use it to process the response and make any updates to the DOM that are needed based on this new data.
- XML: The original format of the data that was sent back on the response. Nowadays we will amost always be using JSON as the format.

HIDE GOT IT! Describe the different steps in an AJAX request/response cycle

- An event listener is set up to wait for a specific action that will trigger a request to our server. Clicking on a button or submitting a form would be a popular example
- When the event is triggered, we use JavaScript to formulate an appropriate request to a server. In our project we used fetch in order to send a request to a specific route on our server, along with an options object to indicate the methods, headers, etc., that differ from the default values, a body with necessary data, etc..
- The request is sent asynchronously to the server. The user is still able to interact with our application since the request is not blocking the call stack
- The server receives the request and does whatever it needs to do on its end to create/read/update/destroy data related to the request. After it performs the requested action, it creates a response and sends it back to the client. This is almost always going to be in a JSON format.
- The client receives the response and is able to parse the data and do any updates that it needs to do in the DOM. In our project we use a .then on our call to fetch, which allowed us to then convert the response's JSON into a usable POJO when the response came back. The data inside of this object is then accessible and used to manipulate the DOM.

HIDE GOT IT! Write an event listener in JavaScript that will fetch to 'http://localhost:3000/kitten/downvote', with the 'PATCH' method, handle the response and catch any errors.

```
document.querySelector('#downvote').addEventListener('click', () => {
    fetch('http://localhost:3000/kitten/downvote', {method: 'PATCH' })
        .then(handleResponse)
        .then(updateImageScore)
        .catch(handleError);
});
```

HIDE GOT IT! Write another event listener to the above, but this time use the try/catch syntax.

```
document.querySelector('#downvote').addEventListener('click', async () =>
{
    try {
        const resJSON = await fetch('http://localhost:3000/kitten/downvote', {method: 'PATCH' })
        const resObj = await handleResponse(resJSON);
        updateImageScore(resObj);
    } catch (e) {
        handle (e);
    }
```

HIDE GOT IT! Write the handleResponse for the fetch calls above.

```
const handleResponse = (response) => {
    stopLoader();
    clearError();

    if (!response.ok) {
        throw response;
    }
    return response.json();
};
```

HIDE GOT IT! Write the handleError function for the code above.

```
const handleError = (error) => {
    if (error.json) {
        document.querySelector('.error').innerHTML = `Error occured: ${errorJSON.message}`;
    } else {
        console.error(error);
        alert('Something went wrong. Please try again!');
    }
};
```

HIDE GOT IT! Write the updateImageScore for the fetch call above.

```
const updateImageScore = (data) => {
    const { score } = data;
    document.querySelector('.score').innerHTML = score;
};
```

HIDE GOT IT! In CSS, what are the different types of media that a media query can target?

- all: Able to work on every device
- print: Anytime a document is set to print mode, with the intention of printing
- screen: Any phone, tablet, computer, smart device, etc.
- speech: For use with speech synthesizers

HIDE GOT IT! Given the following CSS rule, write a media query that would change the product-index container so the items appear in a vertical fashion for a viewport width less than or equal to 300p

```
.product-index {
    display: flex;
}

.product-index__item {
    background-color: blue;
}

@media screen and (max-width: 300px) {
    .product-index {
        flex-direction: column;
    }
}
```

HIDE GOT IT! Describe padding

The innermost part of the box model -- creating space around an element's content (like putting on a puffy jacket)

HIDE GOT IT! Describe margin

The space between one html element and another html element

HIDE GOT IT! What is the different between `box-sizing: border-box;` and `box-sizing: content-box`

border-box includes the border and padding in the width and height; content-box does not;

```
div {
    box-sizing: border-box;
    width: 200px; /* including the padding and border */
    padding: 50px;
    border: 5px solid black;
    /* total width of div is 200 px. */
}

div {
    box-sizing: content-box;
    width: 200px; /* not including the padding and border */
    padding: 50 px;
    border: 5px solid black;
    /* total width of content is 200px */
    /* total width of div is 310 px; -- 200px + 2 * 50px (padding) + 2 * 5px (border) */
```

HIDE GOT IT! Which is the default box-sizing: border-box or content-box?

content-box: Where the border and padding are NOT included in the width and height;

HIDE GOT IT! Describe `position: fixed`

Always relative to the document, not any particular parent, and are unaffected by scrolling.

HIDE GOT IT! Describe `position: relative`

Positioned relative to itself; i.e. the element is still in the flow of the document, but now left/right/top/bottom/z-index will work.

HIDE GOT IT! Describe `position: absolute`

Element is removed from the flow of the document and other elements will behave as if it's not even there whilst all the other positional properties will work on it.

HIDE GOT IT! Describe `position: static`

The default positioning of all elements

HIDE GOT IT! What are the 5 display property values?

1. inline
2. inline-block
3. block
4. grid
5. flex

HIDE GOT IT! Which of the following has block as their default display value:

- div
- span
- p
- a
- button
- ul
- textarea

- div
- p
- ul

HIDE GOT IT! Describe the z-index

z-index refers to the "third dimension" i.e. stacking elements on top of each other. A higher z-index means bringing the element to the top.

HIDE GOT IT! How does flex-box lay out elements?

Flex box allows for easy responsive design by displaying items in a flexible container, where a container's height/width will adjust to the viewport.

HIDE GOT IT! What are the properties that control the distribution of elements in a flex-box?

- justify-content: alignment of items along a main axis; distributes extra space around/between items
- align-items: justify content for the cross axis

HIDE GOT IT! What properties specify the layout of children elements of a flex container?

- flex-grow: determines how much available space the element will take up (1*, 2*, 3*) So if you have 3 elements, and one is set to flex-grow: 3, this element gets 3 times the size, and the other elements get smushed to fit.
- flex-basis: initial item size; can set to auto which will evenly distribute the elements within the flex box
- flex-shrink: determines how much the flex item can shrink relative to the rest.

HIDE GOT IT! What does flex-wrap do?

Defines whether the flex items are forced in a single line, or can be flowed into multiple lines. Options: nowrap(default), wrap, wrap-reverse

HIDE GOT IT! What does align-self do?

It makes it possible to override the align-items value for a specific flex item. It accepts the same 5 values as align-items: flex-start, flex-end, center, baseline, stretch.

HIDE GOT IT! What does justify-content do in a flex-box?

It defines the alignment along the main axis. The values are: flex-start; flex-end; center; space-between; space-around; space-evenly.

HIDE GOT IT! What does align-items do in a flex-box?

It defines the default behaviro for how items are laid out along the cross axis. Options are: flex-start, flex-end, center, baseline, stretch.

HIDE GOT IT! What does order do in a flex box?

Sets the order of each item. For instance, if you have 5 boxes, you can set one box to be the 5th box which will push the first 4 boxes to the left and this box will be the 5th box.

HIDE GOT IT! How does grid layout lay out elements?

This layout style sections off your document into smaller sections that can be organized and customized via the css grid layout properties. Number of rows and columns, and their respective dimensions can be set.

HIDE GOT IT! Explain and use the shorthand versions of grid-column and grid-row to define how an element will span a grid layout

- grid-column: grid-column-start / grid-column-end
- grid-row: grid-row-start / grid-row-end
- can use grid-area with grid-template-areas to visualize grid layout
    - grid-template-areas: each string represents one row
    - use grid-areas name to define region element will take up
    - grid-area: grid-row-start / grid-column-start / grid-row-end / grid-column-end

HIDE GOT IT! Explain the "fr" unit of measure

fr: fraction unit of measure used for creating grid layout

HIDE GOT IT! How do you specify the layout of the grid?

Use grid-template-columns, grid-template-rows, and grid-template properties to specify the layout of the grid using relative and absolute measures.

HIDE GOT IT! How do you label areas of a grid?

Use grid-template-areas

HIDE GOT IT! How do you assign an element to a grid area?

Use grid-area (with the grid template area name you previously assigned)

HIDE GOT IT! How do you create spans across multiple columns and rows?

Use grid-column-start / grid-column-end and grid-row-start / grid-row-end

HIDE GOT IT! How do you adjust items along the main axis (left, right, center, etc.)

Use justify-content: flex-start; flex-end; center

HIDE GOT IT! How do you adjust items along the cross-axis (up, down, center)

Use align-itmes: flex-start, flex-end, center, etc.

HIDE GOT IT! Describe in BEM terminology what the B stands for.

B: Block : standalone entity that is meaningful on its own

HIDE GOT IT! What are some examples of a block?

header, container, menu, navbar, input

HIDE GOT IT! What can a block identifier contain (ie letters, numbers, etc.)

Latin letters (X, M, IX, C, etc), digits, dashes

HIDE GOT IT! What does the "E" of BEM stand for?

Element - part of a block, no meaning on its own

HIDE GOT IT! Give some examples of an "E" element

Element: menu item, list-item, header title, input label

HIDE GOT IT! What letters, numbers, etc, can make up an element identifier?

Latin letters, digits, dashes, underscores.

HIDE GOT IT! What does the "M" of BEM stand for?

Modifier: A flag on a block or element used to change appearance or behavior

HIDE GOT IT! How do you compose a BEM CSS class?

.block__elem

HIDE GOT IT! How would you compose a CSS BEM class with a modifier?

.block__elem--mod or .block--color-red

HIDE GOT IT! Create a BEM class name for a nav list-container with a shadow

nav__list-container--shadow

HIDE GOT IT! Create a BEM class name for a form that has modifiers theme-xmas and simple

form--theme-xmas and form--simple

HIDE GOT IT! Create a BEM CSS class for form that has elements input and submit. Submit has its own modifier disabled

form__input, form__submit, form__submit--disabled

HIDE GOT IT! Use the "hover" pseudo-class to make changes to a button, making the border 1px solid #386a7a and the color = #65c547b

```
div {
    width: 100px;
    height: 100px;
    background: red;
    transition-property: width;
    transition-duration: 2s;
    transition-delay: 1s;
}
div:hover {
    width: 300px;
```

HIDE GOT IT! How would you make an element show animated changes?

Use the transition-property and transition-duration

```
.button {
    transition: margin-top -5 2s;
}
```

HIDE GOT IT! What does the overflow CSS property do?

It specifies what should happen if content overflows an element's box. This property specifies whether to clip content or to add scrollbars when an element's content is too big to fit in a specified area.

HIDE GOT IT! Does overflow work for all block elements?

No. It only works for block elements with a specified height.

HIDE GOT IT! What is a default overflow value?

visible: The overflow is not clipped. It renders outside the element's box if it overflows. This is the default.

HIDE GOT IT! What does overflow: hidden do?

If the content is too large to fit in the box, the content that does not fit will not be displayed.

HIDE GOT IT! What are the overflow properties and values?

- visible: The overflow is not clipped. It renders outside the element's box. This is default
- hidden: The overflow is clipped, and the rest of the content will be invisible
- scroll: The overflow is clipped, but a scroll-bar is added to see the rest of the content
- auto: If overflow is clipped, a scroll-bar should be added to see the rest of the content.
- initial: Sets this property to its default value.
- inherit: Inherits this property from its parent element.

HIDE GOT IT! What is the overflow-x property?

erflow-x specifies whether to clip the content, add a scroll bar, or display overflow content of a block-level element, when it overflows at the left and right edges.

HIDE GOT IT! What is the overflow-y property?

The overflow-y property specifies whether to clip the content, add a scroll bar, or display overflow content of a block-level element, when it overflows at the top and bottom edges.

HIDE GOT IT! What effect does `position: fixed` have on an element?

The element is removed from the page flow and is typically positioned relative to the document itself and not to an ancestor element. It is positioned using the top, right, bottom and left properties. The element remains in the same spot on the page, regardless of viewport size or scrolling.

SHOW GOT IT! Which of the following diagrams shows the correct layout of the elements given their margin and padding settings shown in the CSS below:

```
<body>
  <div class="container">
    <div class="element"></div>
  </div>
</body>

CSS
.container {
  background-color: #2B2D2F;
  width: 300px;
  height: 100px;
}

.element {
  background-color: hotpink;
  width: 100px;
  height: 80px;
  padding: 10px;
  margin: 0 auto;
}
```

Diagram A

Diagram B

Diagram C

Diagram D

SHOW | GOT IT! Given the following HTML and CSS, which of the following diagrams shows the correct layout of the absolutely positioned element?

```
<body>
  <div class="container">
    <div class="element" id="element-1">1</div>
    <div class="element" id="element-2">2</div>
    <div class="element" id="element-3">3</div>
  </div>
</body>
```

```
CSS
.container {
  background-color: #2B2D2F;
}

.element {
  display: inline-block;
  width: 100px;
  height: 280px;
  font-size: 36px;
}

#element-1 {
  background-color: #FFFF00;
}

#element-2 {
  background-color: #FF69B4;
  position: absolute;
  top: 60px;
}

#element-3 {
  background-color: #00EEEE;
}
```

Diagram A

Diagram B

Diagram C

Diagram D

HIDE GOT IT! Given the following HTML and CSS, which of the following diagrams shows the correct layout of the absolutely positioned element contained in a relatively positioned element?

HTML

```
<body>
  <div class="container">
    <div class="element" id="element-1">1</div>
    <div class="element" id="element-2">2</div>
    <div class="element" id="element-3">3</div>
```

```
  </div>
</body>

CSS
.container {
  background-color: #2B2D2F;
  position: relative;
}

.element {
  display: inline-block;
  width: 100px;
  height: 280px;
  font-size: 36px;
}

#element-1 {
  background-color: #FFFF00;
}

#element-2 {
  background-color: #FF69B4;
  position: absolute;
  bottom: 60px;
}

#element-3 {
  background-color: #00EEEE;
}
```

Diagram A

Diagram B

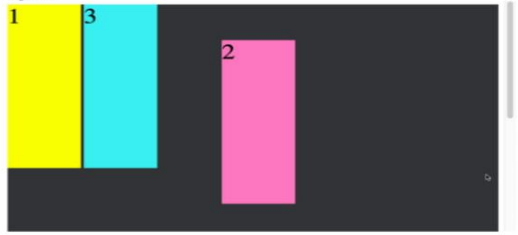


Diagram C

Diagram D

Diagram C

HIDE GOT IT! What position setting completely removes itself from the document flow, and what is that position dependent upon?

position: absolute;

- Completely removes the element from the document flow.
- Elements around it position themselves as though the element didn't exist.
- Absolute in conjunction with top, bottom, left, right, reference the parent element who's position is NOT static.
- If there is no parent element that is NOT static, then the position of top, bottom, etc, will be relative to the root document.
- Position absolute will scroll out of view.

HIDE GOT IT! What position setting will allow you to position the element with (top, bottom, left, right) relative to its normal position?

position: relative

position: relative does not change anything on its own, without setting top, bottom, left, and right.

HIDE GOT IT! What is the most common use for position: relative?

Setting the parent element to a non-static value, such that child elements may be positioned relative to this parent. Any non-static setting will allow this, but relative is the most common choice.

HIDE GOT IT! If you want to set an element to a specific position on the page, never to have its position interrupted by any other element, what position setting would you choose?

position: absolute

HIDE GOT IT! What would you use if you want to lock an element into a specific position, not dependent upon a parent element, which will remain in the same position regardless of scrolling?

position: fixed

Again note: Position fixed is similar to position absolute BUT rather than positioning itself relative to a parent element, it positions itself relative to the document and a specific on-screen position. However, unlike position absolute, when scrolling begins, this element will not move with the scrolling.

HIDE GOT IT! What is the default positioning setting?

position: static is default. However, the element set to position static will not be able to be used to position its children.

HIDE GOT IT! What position setting initially is relative, but as you scroll it becomes fixed to the top of the page?

position: stickey

This is a combination of both relative and fixed position. While you scroll, this position remains relative. But as you scroll the parent out of view this becomes fixed position, never scrolling out of view.

HIDE GOT IT! How do you create a grid?

Set the grid container class's display to grid or inline-grid

HIDE GOT IT! How do you put items into the grid?

All direct children of the grid container automatically become grid items.

HIDE GOT IT! What are grid container grid settings?

```
.grid__container {
    display: grid;
    grid-column-gap: 50px;
    grid-row-gap: 50px;
    grid-gap: 50px 100px; /* row, column */
    grid-template-columns:  repeat(4 1fr) /*can use px, or auto or combined to specify number of cols */
    grid-template-rows: 80px 200px /* ditto above */
    justify-content: space-between; /*space-around, center, start, end, space-evenly */
    align-content: space-around; /* space-evenly, space-around, space-betwen, center, start, end */
    grid-template-areas: 'area area . . .'  /* (template area "area" uses two of 5 cols) */
```

HIDE GOT IT! **What are the rules for using a flex box?**

- A flexible layout must have a parent element with the display property set to flex.
- Direct child elements of the flexible container automatically become flexible items.

HIDE GOT IT! **What properties are available on a flexible container and on a flex item?**

```
.class__container {
    display: flex;
    flex-direction: row | row-reverse | column | column-reverse
    flex-wrap: nowrap | wrap | wrap-reverse;
    flex-flow: column wrap; //shorthand for flex-direction and flex-wrap
    justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly | left | start
    align-items: flex-start | flex-end | center | stretch | baseline
    align-content: flex-start | flex-end ... same as justify content = aligns container's lines within when there is extra space in cross-asix
}
```

```
.class_childItem {
    order: 1 | 2 | 3 | 5 | ... 99 //default: 0;
    flex-grow: 1;  //grow twice as large as other containers; 3 times, 4 times, etc.
    flex-shrink: 3; //default 1;
    flex-basis:  | auto /* default auto:  default size of elt before remaining space is distrib.  20%, 5rem, auto, etc.
    flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ] //shorthand for flex-grow, flex-shrink and flex-basis; 2nd & 3rd properties are optional. Defualt is 0 1 auto;
    align-self: flex-start | flex-end | auto | center | baseline
}
```

1. What is a CSS rule?
    - A CSS rule consists of a selector and curly braces around property-value combinations.
2. How to import other CSS files into your CSS file
   How to link a stylesheet into an HTML page
    - css into css : with use of the `@import` tag!
    - css into html : by using the `link` HTML element.
        - The link element has to have the `rel` and `href` attributes configured like so:

```
<link rel="stylesheet" href="file.css">
```

3. Explain how CSS rules are applied based on their order and specificity, and be able to calculate the specificity of CSS rules and determine which rule override the properties of another.
    - `IDs` are unique, thus are considered highly specific since they always target one element.
    - Calculating the `score specificity` of selectors, and the one with the highest score is the most specific.
    - `in-line styling > id > pseudo-class selectors > class > tag name`
4. Write "combinators" to create compound selector statements to target specific elements
    - *Direct child selector*: the carrot, `>` means a **direct** child.
    - *All descendants/children selector*: denotes with a `space` in between selectors
    - *Multiple items selection*: `comma` dileniated selectors (ex-> body, div, html {})
5. Explain and apply pseudo selectors for specific elements in specific states (i.e. :hover)
    - See example from Monday's code demo:
      html:

```
<a href="https://google.com">Link</a>
```

      css:

```
a:hover {
    font-family: "Roboto Condensed", sans-serif;
    color: #4fc3f7;
    text-decoration: none;
    border-bottom: 2px solid #4fc3f7;
}
```

- The **not** pseudo-class selector:
  html:

```
<div class="hello">Hello</div>
<div class="world">World</div>
<div>Random</div>
```

```
div:not(.world) {
    background-color: red;
}
```

Only `div`s containing the text `"Hello"` and `"Random"` will have a `background-color` of `red`.

6. Explain and apply the `::before` and `::after` pseudo elements, & Use the content CSS property to define the content of an element:
   - See exerpt from W9D1 demo:
     html:

```
<p> this is a paragraph tag</p>
```

     css:

```
p::after {
    background-color: lightblue;
    border-right: 1px solid violet;
    content: ':-D';
    margin-right: 4px;
    margin-left: 4px;
}
```

7. What are these ?!
   - `font-size` : size of letters
   - `font-weight` : boldness of letters
   - `font-style` : italicization
   - `font-family` : actualy font
     - Some general font families: [sans-serif, serif, cursive]
   - `text-transform` : text casing
   - `text-decoration` : underlining
   - `text-align` : text justification (left, right, etc.)
8. Colors expressed as names, hexadecimal RGB values, and decimal RGB values
   - The `color` css property changes the color of the *text*.
   - The `background-color` css property does just what it says
   - `rgb()` - takes in 3 integer values denoting levels of `red, green and blue`

- `rgba()` - Same as above but with additional argument (the 'a') called **alpha** which represents how **transparent** the color will be and is on a scale from 0 to 1 where 0 is transparent and 1 is opaque.

9. Everything about borders, Shadows, & Opacity
   - takes in three arguments: thickness, line style, line color
   - Text and box shadows
   - Opacity set to 0 can make an element completely transparent.

```
h2 {
    border: 4px solid red;
    text-shadow: 0px 2px 10px orange;
    box-shadow: 5px 5px yellow;
    opacity: 0.5;
}
```

10. Covering an element with a background image

```
#picture-here {
    background-image: url(https://appacademy.github.io/styleguide/assets/logo/logo-emble
    background-size: cover;
    height: 100px;
    width: 100px;
}
```

11. `Explain why using Web fonts helps with consistent experience across viewing devices:
    - `Your explanation here.`
    - Web fonts are good to use because every browser has different default font families, but there is a drawback. Google tracks when you use their fonts using google fonts
    - Will be the same in any browser.
12. Explain absolute and relative length units in CSS
    - Relative :
      - `rem` - relative to root
      - `em` - relative to parent
    - Absolute Measure :
      - `pt`
      - `px`
      - `cm`

Wednesday

1. `display` property:
   - `inline`

- `inline-block`
- `block`
- `flex`
- `grid`

2. Identify the different types of media that a media query can target
   - all
   - print
   - screen
   - speech
3. Know the Box Model

1. Describe the following
   - padding
   - margin
   - `position: fixed` -
   - `position: relative` - positioned relative to closest parent ancestor
   - `position: absolute` - removes from regular flow of the page.
   - `position: static` - the default positioning of all elements.
   - The MC quiz from earlier in the week is good for positioning stuff.
2. Identify elements rendered with specific padding and margin settings
3. Apply padding and margins to HTML elements to achieve a desired layout
4. Apply positioning settings to elements (fixed, relative, and absolute) to HTML elements to achieve a desired layout
5. Identify which HTML elements have a default "inline" display value
   - `fill in`
6. Identify which HTML elements have a default "block" display value
   - `fill in`
7. Describe and use z-index positioning of elements
   - `z-index` : refers to the "third dimension" i.e. stacking elements on top of each other. a higher z-index means bringing the element to the top.
8. Explain how flexible box layout lays out elements
   - displays items in flexible container so layout is responsive
   - containers height/width adjust to fit viewport
   - `justify-content`
     - alignment of items along main axis
     - distributes extra space around/between items
   - `align-items`

        - justify content for cross axis
9. For the following LOs please revisit your project work and project solutions (i.e. AA Times, Wednesday EOD demo, etc.) for how you've done the following:
    - Use the flex property to specify `grow`, `shrink`, and `basis` values.
    - `grow` determines how much available space it will take up
    - `shrink` determines how much the element can shrink
    - `basis` is considered to be the default size of the element
    - Use the `flex-direction` property to direct the layout of the content.
    - Use the `flex-wrap` property to affect the wrap of content layout within an element using flexible box layout.
    - Use `align-self`, `justify-content`, and `align-items` to change the way that children elements are laid out in a flexible box layout.
    - Use the `order` property to change the order in which elements will appear in a flexible box layout.
    - Use the `grid-template-columns`, `grid-template-rows`, and `grid-template` properties to specify the layout of the grid using relative and absolute measures.
    - Use `grid-template-areas` to label areas of a grid and `grid-area` to assign an element to the area.
    - Use `grid-column-gap`, `grid-row-gap`, and `grid-gap` to set the "gutter" areas between elements in a grid layout.
    - Use `grid-column-start` / `grid-column-end` and `grid-row-start` / `grid-row-end` to create spans across multiple columns and rows with positive integers, negative integers, and in conjunction with the "span" operator.
    - Use `justify-items`, `align-items`, `justify-content` and `align-content` to layout items in each grid area.
    - Use the `order` property to change the default order in which items are laid out
10. Explain how grid layout lays out elements
    - Sections off your document into smaller sections that can be organized and customised via the
      css grid layout properties.
11. Explain and use the shorthand versions of `grid-column` and `grid-row` to define how an element will span a grid layout
    - `grid-column: grid-column-start / grid-column-end`
    - `grid-row: grid-row-start / grid-row-end`
    - can use grid-area with grid-template-areas to visualize grid layout
        - grid-template-areas: each string represents one row
        - use grid-areas name to define region element will take up
        - `grid-area: grid-row-start / grid-column-start / grid-row-end / grid-column-end`
12. Explain and use the "fr" unit of measure

- `fr` : fraction unit of measure used for creating grid layout

Thursday

1. Describe what `Block` , `Element` , and `Modifier` means in BEM
   - `block__element--modifier` `element` has `n` for `underscore` and `modifier` has `d` for `dash`
   - Block: standalone entity that is meaningful on its own
     - header, container, menu, navbar, input
     - name can contain latin letters, digits, dashes
   - Element - part of a block, no meaning on its own
     - menu item, list-item, header title, input label
     - name can contain latin letters, digits, dashes, underscore
     - class formed by block name + two underscores + element name
   - Modifier - flag on a block or element used to change appearance or behavior
     - disabled, color yellow, size big, fixed
     - add modifier class to blocks/el they modify and keep original class
     - css class formed as block or el name plus two dashes
2. Identify CSS class names that follow the BEM principle.
   - Go to wednesday EOD code demo for more (W9D3)

     ```
     <header class="header header--color>
     ```

3. Describe and use the transition property show animated changes due to class and pseudo-class CSS rule application
   - `transition-property`
     - name/names of css props to which transitions should apply
     - transition effect starts when specified CSS prop changes
   - `transition-duration`
     - duration over which transition occurs
   - `transition-delay`
     - how long to wait between time prop is changed and transition begins
   - `transition`
     - shorthand: property, duration, delay
4. Describe and use the overflow, overflow-x, and overflow-y properties to effect clipping and scrolling on elements
   - `overflow: auto`
   - `overflow: scroll`
   - `overflow: hidden`

Tuesday - AJAX

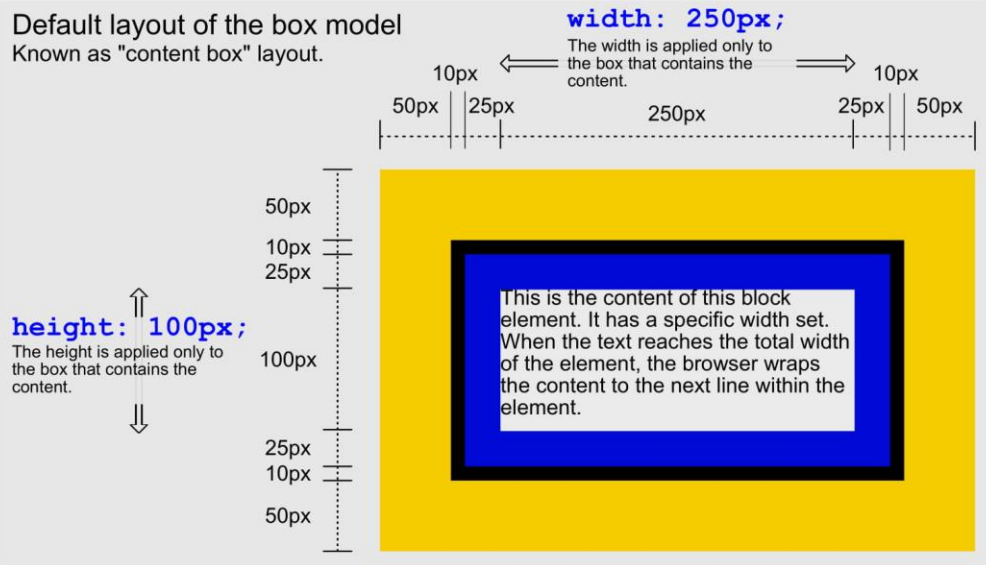
1. Explain what an AJAX request is
2. Identifying the advantages of using an AJAX request.
    - We don't have to refresh the entire HTML page.
    - It's a smaller amount of data that needs to be transferred.
3. Identify what the acronym AJAX means and how it relates to modern Web programming
    - Asynchronous JavaScript and XML
    - Asynchronous: We don't lock up the page when we are waiting on a response. We are still able to interact and the response's data will be handled whenever it returns.
    - JavaScript: The engine behind AJAX. We use JavaScript to make the request to the server, then we also use it to process the response and make any updates to the DOM that are needed based on this new data.
    - XML: The original format of the data that was sent back on the response. Nowadays we will almost always be using JSON as the format.
4. Describe the different steps in an AJAX request/response cycle
    - An event listener is set up to wait for an specific action that will trigger a request to our server. Clicking on a button or submitting a form would be a popular example.
    - When the event is triggered, we use JavaScript to formulate an appropriate request to a server. In our project we used `fetch` in order to send a request to a specific route on our server, along with an options object to indicate the methods, headers, etc., that differ from the default values, a body with necessary data, etc..
    - The request is sent asynchronously to the server. The user is still able to interact with our application since the request is not blocking the call stack.
    - The server receives the request and does whatever it needs to do on its end to create/read/update/destroy data related to the request. After it performs the requested action, it creates a response and sends it back to the client. This is almost always going to be in a JSON format.
    - The client receives the response and is able to parse the data and do any updates that it needs to do to the DOM. In our project, we used a `.then` on our call to `fetch`, which allowed us to then convert the response's JSON into a usable POJO when the response came back. The data inside of this object is then accessible and used to manipulate the DOM.
5. Fully use the fetch API to make dynamic Web pages without refreshing the page
    - Look over the AJAX project from Friday. Be comfortable with creating many different request types, such as `GET`, `PATCH`, `POST`, and `DELETE`.
    - Be comfortable with using both the `.then` promise chains that we used in the project as well as how we could convert them into an async/await format:

```
// Using Promise chains for .then and .catch
document.querySelector('#downvote').addEventListener('click', () => {
        fetch('http://localhost:3000/kitten/downvote', { method: 'PATCH' })
                .then(handleResponse) // handleResponse defined below for reference
    .then(updateImageScore) // updateImageScore defined below for reference
    .catch(handleError); // handleError defined below for reference
});


// Using async/await
document.querySelector('#downvote').addEventListener('click', async () => { // Notice the async
// We create a standard try/catch block
  try {
    // We await each asynchronous function call
    const resJSON = await fetch('http://localhost:3000/kitten/downvote', { method: 'PATCH' });
    const resObj = await handleResponse(resJSON);
    // updateImageScore is synchronous, so we do not have to await its response
    updateImageScore(resObj);
  } catch (e) {
    handleError(e)
  }
});


// Functions used above, for reference
const handleResponse = (response) => {
        stopLoader();
        clearError();

        if (!response.ok) {
                throw response;
        }
        return response.json();
};

const handleError = (error) => {
        if (error.json) {
                error.json().then((errorJSON) => {
                        document.querySelector('.error').innerHTML = `Error occured: ${errorJSON
                });
        } else {
                console.error(error);
                alert('Something went wrong. Please try again!');
        }
};

const updateImageScore = (data) => {
        const { score } = data;
        document.querySelector('.score').innerHTML = score;
};
```

- For items that are using `inline` as it's display, the browser will follow these rules to layout the element:
  - Each box appears in a single line until it fills up the space.
  - Width and height are **not** respected.
  - Padding, Margin, and Border are applied but they **do not** push other elements away from the box.
- Certain elements have `inline` as their default display, such as: span tags, anchors, and images.

Standard Box Model vs Border-Box

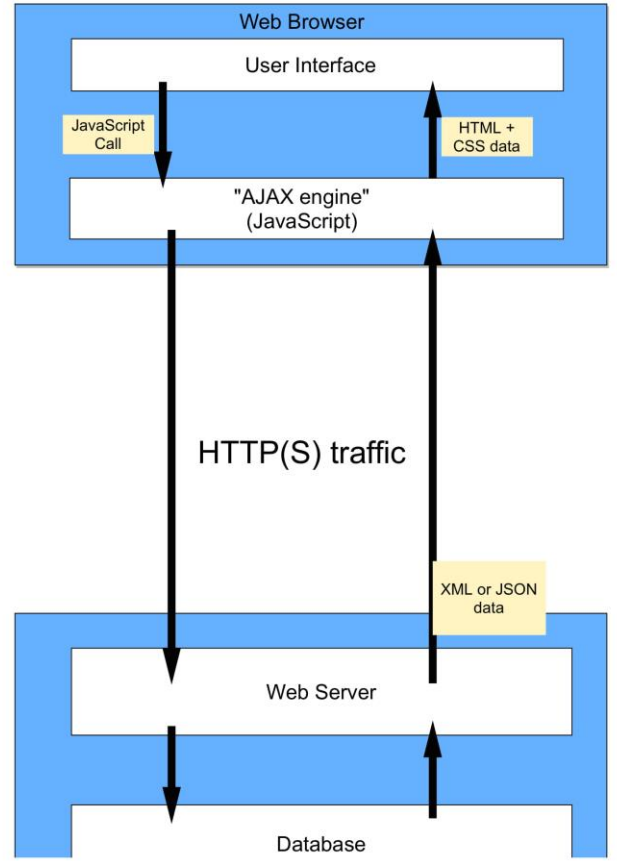
- As per the standard Box Model, the width and height pertains to the content of the box and excludes any additional padding, borders, and margins.



- This bothered many programmers so they created the **border box** to include the calculation of the entire box's height and width.


```
<button class="want-to-read">Want to Read</button>

<script>
  document.querySelector(".want-to-read").addEventListener("click", function() {
    fetch(`https://api.goodreads.com/books/${BOOK_ID}/update-status`, {
      method: "PATCH", // using PATCH since we'll just be modifying the book's status
      headers: {
        "Content-Type": "application/json"
      },
      body: JSON.stringify({
        status: "Want to Read"
      })
    })
      .then(function(res) {
        if (!res.ok) {
          throw Error(res.statusText); // handle any potential server errors
        }
        return res.json(); // extract JSON from the response
      })
      .then(function(data) {
        document.querySelector(".want-to-read").innerHTML = "✓ Want To Read";
      })
      .catch(function(error) {
        const errorMessage = document.createElement("p");
        errorMessage.appendChild(
          document.createTextNode("Something went wrong. Please try again!")
        );

        // This example appends an error message to the body for simplicity's sake.
        // Please do not copy this kind of DOM manipulation in your own projects:
        document.querySelector("body").appendChild(errorMessage);
      });
  });
</script>
```

1. When we first send out the event listener and fetch with filled out options this is ths segment where we are conducting the `Javascript Call` .
2. When the request is sent out it is the arrow leading from the AJAX engine to the Web Server.
3. The arrow from the Web Server back to the AJAX engine is the response from the Server in either XML or JSON format.
4. The response is handled within the AJAX engine and returns the new HTML & CSS for the UI.




```
fetch("https://jservice.xyz/api/categories", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({
    title: "ACTORS & THEIR FILMS",
  }),
})
  .then(function (res) {
    console.log(res);
    if (!res.ok) {
      throw Error(res.statusText);
    }
    return res.json();
  })
  .then(function (data) {
    console.log(data);
  })
  .catch(function (error) {
    console.log(error);
  });
```

- IMPORTANT! Fetch API will not reject HTTP status codes between 400 & 600 - it will only reject on errors like network errors.
  - Instead, Fetch REQUIRES you to check the `ok` key inside of the Response Object, and if the key is false then the fetch can properly handle the error.
  - TI;DR Don't forget to check the `ok` key and place a catch error handler at the end of the promise chain.