**freeCodeCamp**

**Part 1 – Responsive Web Design**

**FONTS**

* In addition to specifying common fonts that are found on most operating systems, we can also specify non-standard, custom web fonts for use on our website. Google fonts.

<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet" type="text/css">

Now you can use the Lobster font in your CSS by using Lobster as the FAMILY\_NAME as in the following example:

p {  
  font-family: Helvetica, sans-serif;  
}

**Borders:**

CSS borders have properties like style, color and width

For example, if we wanted to create a red, 5 pixel border around an HTML element, we could use this class:

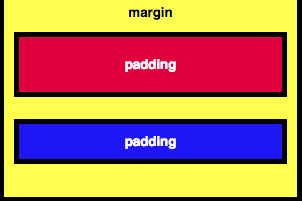
<style>  
  .thin-red-border {  
    border-color: red;  
    border-width: 5px;  
    border-style: solid;  
  }  
</style>

Your cat photo currently has sharp corners. We can round out those corners with a CSS property called border-radius.

You can set an element's background color with the background-color property.

In addition to pixels, you can also specify the border-radius using a percentage.

An idalso has a higher specificity (importance) than a class so if both are applied to the same element and have conflicting styles, the styles of the idwill be applied.



You may have already noticed this, but all HTML elements are essentially little rectangle

Three important properties control the space that surrounds each HTML element: padding, margin, and border.

Padding:

An element's padding controls the amount of space between the element's content and its border. When you increase the blue box's padding, it will increase the distance(padding) between the text and the border around it.

CSS allows you to control the padding of all four individual sides of an element with the padding-top, padding-right, padding-bottom, and padding-leftproperties.

Margin:

An element's margin controls the amount of space between an element's border and surrounding elements. If you set an element's marginto a negative value, the element will grow larger.

CSS allows you to control the marginof all four individual sides of an element with the margin-top, margin-right, margin-bottom, and margin-leftproperties.

Instead of specifying an element's padding-top, padding-right, padding-bottom, and padding-leftproperties individually, you can specify them all in one line, like this:

Clockwise Notation for top, left, bottom, right

E.g. padding: 10px 20px 10px 20px; OR margin 40

**Attribute Selectors**

There are other CSS Selectors you can use to select custom groups of elements to style.

For this challenge, you will use the [attr=value]. For example, the below code changes the margins of all elements with the attribute type and a corresponding value of radio:

[type='radio'] {  
  margin: 20px 0px 20px 0px;  
}

ABOSLUTE AND RELATIVE UNITS

The two main types of length units are absolute and relative.

Relative units, such as em or rem, are relative to another length value. For example, em is based on the size of an element's font. If you use it to set the font-size property itself, it's relative to the parent's font-size.

**Override Styles in Subsequent CSS**

However, the order of the class declarations in the <style>section are what is important. The second declaration will always take precedence over the first. Because .blue-text is declared second, it overrides the attributes of .pink-text

Why would you want to override CSS? In many situations, you will use CSS libraries. These may accidentally override your own CSS. So when you absolutely need to be sure that an element has specific CSS, you can use !important E.g. color: red !important;

**HEXADECIMALS**

Hexadecimals(or hex) are base 16 numbers. This means it uses sixteen distinct symbols. Like decimals, the symbols 0-9 represent the values zero to nine. Then A,B,C,D,E,F represent the values ten to fifteen. Altogether, 0 to F can represent a digit in hexadecimal

In CSS, we can use 6 hexadecimal digits to represent colors, two each for the red (R), green (G), and blue (B) components. For example, #000000is black and is also the lowest possible value.

The digit 0is the lowest number in hex code, and represents a complete absence of color.The digit Fis the highest number in hex code, and represents the maximum possible brightness.

 it's difficult to remember hex code. Fortunately, you can shorten it. For example, red's hex code #FF0000can be shortened to #F00. This shortened form gives one digit for red, one digit for green, and one digit for blue.

**RGB Colors**

Instead of using six hexadecimal digits like you do with hex code, with RGByou specify the brightness of each color with a number between 0 and 255.

Body {

Background-color: rgb(0, 0, 0)

}

**Use CSS Variables to change several elements at once**

*CSS Variables* are a powerful way to change many CSS style properties at once by changing only one value.

To create a CSS Variable, you just need to give it a name with two dashes in front of it and assign it a value like this:

--penguin-skin: gray;

Now you can use that variable elsewhere in your CSS to change the value of other elements to gray. After you create your variable, you can assign its value to other CSS properties by referencing the name you gave it.

background: var(--penguin-skin);

When using your variable as a CSS property value, you can attach a fallback value that your page will revert to if for some reason it can't get your variable to work. It could be that someone is using an older browser that hasn't yet adopted CSS Variables, or perhaps their device doesn't support the value you gave the variable. Useful for debugging.

background: var(--penguin-skin, black);

When you create a variable, it becomes available for you to use inside the element in which you create it. It also becomes available within any elements nested within it. This effect is known as *cascading*.

Because of cascading, CSS variables are often defined in the *:root* element. You can think of the :root element as a container for your entire HTML document, in the same way that an htmlelement is a container for the body element. By creating your variables in :root, they will be available throughout the whole web page.



:root {

--penguin-belly: pink;

}You can then over-write these variables by setting them again within a specific element.



## Use a media query to change a variable

CSS Variables can simplify the way you use media queries.

For instance, when your screen is smaller or larger than your media query break point, you can change the value of a variable, and it will apply its style wherever it is used.

## Introduction to the Applied Visual Design Challenges

Visual Design in web development is a broad topic. It combines typography, color theory, graphics, animation, and page layout to help deliver a site's message. This section covers some of the basic tools developers use to create their own visual designs.

**Text Align**

* text-align: justify;causes all lines of text except the last line to meet the left and right edges of the line box.
* text-align: center;centers the text
* text-align: right;right-aligns the text
* And text-align: left;(the default) left-aligns the text.

Width: You can specify the width of an element using the widthproperty in CSS. Values can be given in relative length units (such as em), absolute length units (such as px), or as a percentage of its containing parent element.

* With the strong tag, the browser applies the CSS of font-weight: bold;
* With the u tag, the browser applies the CSS of text-decoration: underline;to the element.
* To emphasize text, you can use the emtag. This displays text as italicized, as the browser applies the CSS of font-style: italic; to the element.
* To strikethrough text, which is when a horizontal line cuts across the characters, you can use the deltag
* You can use the hrtag to add a horizontal line across the width of its containing element.
* The box-shadow property applies one or more shadows to an element.
* line-height property to change the height of each line in a block of text

**Background Color**

rgba stands for:  
  r = red  
  g = green  
  b = blue  
  a = alpha/level of opacity

The RGB values can range from 0 to 255. The alpha value can range from 1, which is fully opaque or a solid color, to 0, which is fully transparent or clear.

**Adding box shadows to Card like Elements**

The box-shadowproperty takes values(in the below order)

* offset-x(how far to push the shadow horizontally from the element),
* offset-y(how far to push the shadow vertically from the element),
* blur-radius, spread-radius

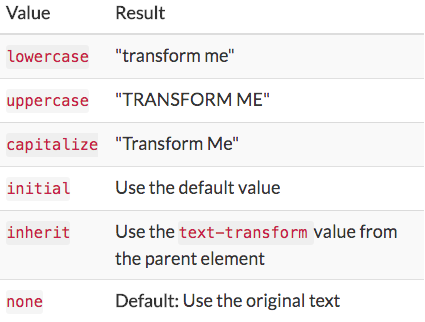
box-shadow: 0 10px 20px rgba(0,0,0,0.19), 0 6px 6px rgba(0,0,0,0.23);

OPACITY

The opacityproperty in CSS is used to adjust the opacity, or conversely, the transparency for an item.

A value of 1 is opaque, which isn't transparent at all.  
A value of 0.5 is half see-through.  
A value of 0 is completely transparent.

The text-transform property in CSS a convenient way to make sure text on a webpage appears consistently, without having to change the text content of the actual HTML elements.



Pseudo states

* A pseudo-class is a keyword that can be added to selectors, in order to select a specific state of the element.
* For example, the styling of an anchor tag can be changed for its hover state using the :hover pseudo-class selector. Here's the CSS to change the colorof the anchor tag to red during its hover state:
* a:hover {  
    color: red;

## Change an Element's Relative Position

CSS treats each HTML element as its own box, which is usually referred to as the CSS Box Model

Block-level items automatically start on a new line (think headings, paragraphs, and divs) while inline items sit within surrounding content (like images or spans). The default layout of elements in this way is called the normal flow of a document, but CSS offers the position property to override it.

When the position of an element is set to relative, it allows you to specify how CSS should move it *relative* to its current position in the normal flow of the page. It pairs with the CSS offset properties of left or right, and top or bottom.

**These say how many pixels, percentages, or ems to move the item *away* from where it is normally positioned**. The following example moves the paragraph 10 pixels away from the bottom:

p {  
  position: relative;  
  bottom: 10px;  
}

Changing an element's position to relative does not remove it from the normal flow - other elements around it still behave as if that item were in its default position.

E.g.: Change the position of the h2 to relative, and use a CSS offset to move it 15 pixels away from the top of where it sits in the normal flow. Notice there is no impact on the positions of the surrounding h1 and p elements.

The CSS offsets of top or bottom, and left or right tell the browser how far to offset an item relative to where it would sit in the normal flow of the document. You're offsetting an element away from a given spot, which moves the element away from the referenced side (effectively, the opposite direction).

## ABSOLUTE

The next option for the CSS position property is absolute, which locks the element in place relative to its parent container. Unlike the relative position, this removes the element from the normal flow of the document, so surrounding items ignore it.

One nuance with absolute positioning is that it will be locked relative to its closest *positioned* ancestor. If you forget to add a position rule to the parent item, (this is typically done using position: relative;), the browser will keep looking up the chain and ultimately default to the body tag.

E.g. Lock the #searchbar element to the top-right of its section parent by declaring its position as absolute. Give it topand right offsets of 50 pixels each

<style>

section {

position: relative;

}

#searchbar {

position: absolute;

top: 50px;

right: 50px;

}

## 

FIXED POSITIONING

The next layout scheme that CSS offers is the fixed position, which is a type of absolute positioning that locks an element relative to the browser window. Similar to absolute positioning, it's used with the CSS offset properties and also removes the element from the normal flow of the document. Other items no longer "realize" where it is positioned, which may require some layout adjustments elsewhere.

*One key difference from the absolute position is that the element won't move when the user scrolls.*

*E.g.:* The navigation bar in the code is labeled with an id of navbar. Change its positionto fixed, and offset it 0 pixels from the topand 0 pixels from the left. Notice the (lack of) impact to the h1position, it hasn't been pushed down to accommodate the navigation bar and would need to be adjusted separately.

Float.

Elemnts pushed to either the leftor right of their containing parent element. It's commonly used with the width property to specify how much horizontal space the floated element requires.

#left {

float: left;

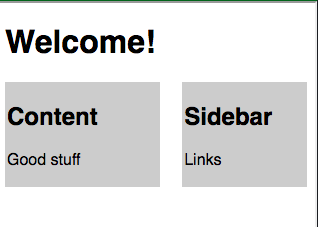
width: 50%;

}

#right {

float: right;

width: 40%;



OVERLAPPING

When elements are positioned to overlap, the element coming later in the HTML markup will, by default, appear on the top of the other elements. However, the z-index property can specify the order of how elements are stacked on top of one another. It must be an integer.

.first {

background-color: red;

position: absolute;

z-index: 1;

}

.second {

background-color: blue;

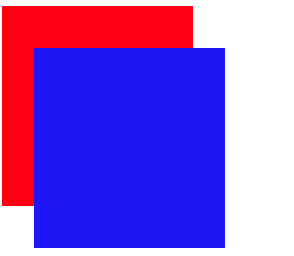
position: absolute;

left: 40px;

top: 50px;

z-index: 1;

}



Margin auto to center

Another positioning technique is to center a block element horizontally. One way to do this is to set its margin to a value of auto.

This method works for images, too. Images are inline elements by default, but can be changed to block elements when you set the display property to block.

VISAUL DESIGN: COLORS

When two colors are opposite each other on the wheel, they are called complementary colors. They have the characteristic that if they are combined, they "cancel" each other out and create a gray color.  However, when placed side-by-side, these colors appear more vibrant and produce a strong visual contrast.

**Aside**: There;s ~3:1 ratio between width and height sizes. E.g. a boxed div element is

Height: 25px;

Width: 75px;

* Use to denote code: <code>fetch Pringles</code>
* Use <br> for line break

COLOR HUE

* Colors have several characteristics including hue, saturation, and lightness. CSS3 introduced the hsl()property as an alternative way to pick a color by directly stating these characteristics. where the angle of the color on the circle is given as a value between 0 and 360

|  |
| --- |
| Examples below: |
| **Color** | **HSL** |
| red | hsl(0, 100%, 50%) |
| yellow | hsl(60, 100%, 50%) |
| cyan | hsl(180, 100%, 50%) |
| blue | hsl(240, 100%, 50%) |
| magenta |  |
|  |  |

* **Saturation** is the amount of gray in a color. A fully saturated color has no gray in it, and a minimally saturated color is almost completely gray. This is given as a percentage with 100% being fully saturated.
* **Lightness** is the amount of white or black in a color. A percentage is given ranging from 0% (black) to 100% (white), where 50% is the normal color.

COLOR TONE

The hsl()option in CSS also makes it easy to adjust the tone of a color. Mixing white with a pure hue creates a tint of that color, and adding black will make a shade. *This is useful when you have a base hue you like, but need different variations of it.*

E.g. Header and nav bar have same base but different saturation and lightness.

header {

background-color: hsl(180, 90%, 35%);

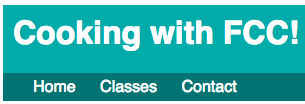
color: #FFFFFF;

}

nav {

background-color: hsl(180, 80%, 25%);

}



Gradients

Applying a color on HTML elements is not limited to one flat hue. CSS provides the ability to use color transitions, otherwise known as gradients.This is accessed through the background property's linear-gradient()function. Here is the general syntax:

background: linear-gradient(gradient\_direction, color 1, color 2, color 3, ...);

background: linear-gradient(90deg, red, yellow, rgb(204, 204, 255));

The first argument specifies the direction from which color transition starts - it can be stated as a degree, where 90deg makes a vertical gradient and 45deg is angled like a backslash. The following arguments specify the order of colors used in the gradient.

REPEATING

The repeating-linear-gradient()function is very similar to linear-gradient()with the major difference that it repeats the specified gradient pattern.

Background Patterns

The backgroundproperty supports the url()function in order to link to an image of the chosen texture or pattern. The link address is wrapped in quotes inside the parentheses.

background: url("https://i.imgur.com/MJAkxbh.png")

Transform scale to change Element Size, on hover etc…

To change the scale of an element, CSS has the transform property, along with its scale()function. The following code example doubles the size of all the paragraph elements on the page:

p {  
  transform:scale(2);  
}

The transform property has a variety of functions that lets you scale, move, rotate, skew, etc., your elements. When used with pseudo-classes such as :hover that specify a certain state of an element, the transform property can easily add interactivity to your elements.

Here's an example to scale the paragraph elements to 2.1 times their original size when a user hovers over them:

p:hover {  
  transform: scale(2.1);  
}

Skew X and Y

The next function of the transformproperty is skewX(), which skews the selected element along its X (horizontal) axis by a given degree.

The following code skews the paragraph element by -32 degrees along the X-axis.

p {  
  transform: skewX(-32deg);  
}

BEFORE and AFTER

You need to understand the :before and :after selectors. These selectors are used to add something before or after a selected element.

For the :before and :after selectors to function properly, they must have a defined content property. It usually has content such as a photo or text. When the :beforea nd :after selectors add shapes, the content property is still required, but it's set to an empty string.

In the following example, a :before selector is used to add a rectangle to an element with the class heart:

.heart:before {  
  content: "";  
  background-color: yellow;  
  border-radius: 25%;  
  position: absolute;  
  height: 50px;  
  width: 70px;  
  top: -50px;  
  left: 5px;  
}

KEYFRAMES & ANIMATIONS

To animate an element, you need to know about the animation properties and the @keyframes rule. The animation properties control how the animation should behave and the @keyframes rule controls what happens during that animation.

 There are eight animation properties in total. This challenge will keep it simple and cover the two most important ones first:

1. animation-name sets the name of the animation, which is later used by @keyframes to tell CSS which rules go with which animations.
2. animation-duration sets the length of time for the animation.

@keyframes is how to specify exactly what happens within the animation over the duration. This is done by giving CSS properties for specific "frames" during the animation, with percentages ranging from 0% to 100%. If you compare this to a movie, the CSS properties for 0% is how the element displays in the opening scene. The CSS properties for 100% is how the element appears at the end, right before the credits roll.

<style>

button {

border-radius: 5px;

color: white;

background-color: #0F5897;

padding: 5px 10px 8px 10px;

}

button:hover {

animation-name: background-color;

animation-duration: 500ms;

/\* add your code below this line \*/

animation-fill-mode: forwards;

/\* add your code above this line \*/

}

@keyframes background-color {

100% {

background-color: #4791d0;

}

}

</style>

<button>Register</button>

When elements have a specified position, such as fixed or relative, the CSS offset properties right, left, top, and bottom can be used in animation rules to create movement.

As shown in the example below, you can push the item downwards then upwards by setting the topproperty of the 50%keyframe to 50px, but having it set to 0px for the first (0%) and the last (100%) keyframe. (see Atom keygrames animation)

ANIMATION ITERATION

Another animation property is the animation-iteration-count, which allows you to control how many times you would like to loop through the animation. Here's an example:

animation-iteration-count: 3;

In the previous challenge, you changed the animation rates for two similarly animated elements by altering their @keyframes rules. You can achieve the same goal by manipulating the animation-duration of multiple elements

ANIMATION TIMING

In CSS animations, the animation-timing-function property controls how quickly an animated element changes over the duration of the animation. If the animation is a car moving from point A to point B in a given time (your animation-duration), the animation-timing-function says how the car accelerates and decelerates over the course of the drive.

There are a number of predefined keywords available for popular options. For example, the default value is ease, ease out, ease-in, linear

BEZIER CURVES

CSS offers an option other than keywords that provides even finer control over how the animation plays out, through the use of Bezier curves. here's an example of a Bezier curve in CSS code:

Animation-timing-function: cubic-bezier(0.25, 0.25, 0.75, 0.75);

**ACCESSIBILITY**

The section element is also new with HTML5, and has a slightly different semantic meaning than article. An article is for standalone content, and a section is for grouping thematically related content. They can be used within each other, as needed. For example, if a book is the article, then each chapter is a section. When there's no relationship between groups of content, then use a div.

<div> - groups content  
<section> - groups related content  
<article> - groups independent, self-contained content

The next HTML5 element that adds semantic meaning and improves accessibility is the header tag. It's used to wrap introductory information or navigation links for its parent tag, and works well around content that's repeated at the top on multiple pages.

The nav element is another HTML5 item with the embedded landmark feature for easy screen reader navigation. This tag is meant to wrap around the main navigation links in your page.

The audio tag supports the controls attribute. This shows the browser default play, pause, and other controls, and supports keyboard functionality. This is a boolean attribute, meaning it doesn't need a value, its presence on the tag turns the setting on.

Here's an example:

<audio id="meowClip" controls>  
  <source src="audio/meow.mp3" type="audio/mpeg" />  
  <source src="audio/meow.ogg" type="audio/ogg" />  
</audio>

HTML5 introduced the figure element, along with the related figcaption. Used together, these items wrap a visual representation (like an image, diagram, or chart) along with its caption.

FORMS(date)

Forms often include the input field, which can be used to create several different form controls. The type attribute on this element indicates what kind of input will be created.

You may have noticed the textand submit input types in prior challenges, and HTML5 introduced an option to specify a date field, which makes filling in a form easier for all users

<label for="pickdate">Preferred Date:</label

<input type="date" id="pickdate" name="date">

## Introduction to the Responsive Web Design Challenges

 Responsive Web Design is an approach to designing web content that responds to the constraints of different devices. In general, design the page's CSS to your target audience.

Media Queries

Media Queries are a new technique introduced in CSS3 that change the presentation of content based on different viewport sizes. The viewport is a user's visible area of a web page, and is different depending on the device used to access the site.

Media Queries consist of a media type, and if that media type matches the type of device the document is displayed on, the styles are applied. You can have as many selectors and styles inside your media query as you want

E.g. Add a media query, so that the p tag has a font-sizeof 10px when the device's height is less than or equal to 800px.

@media screen and (max-height: 800px) {

p {

font-size: 10px

}

}

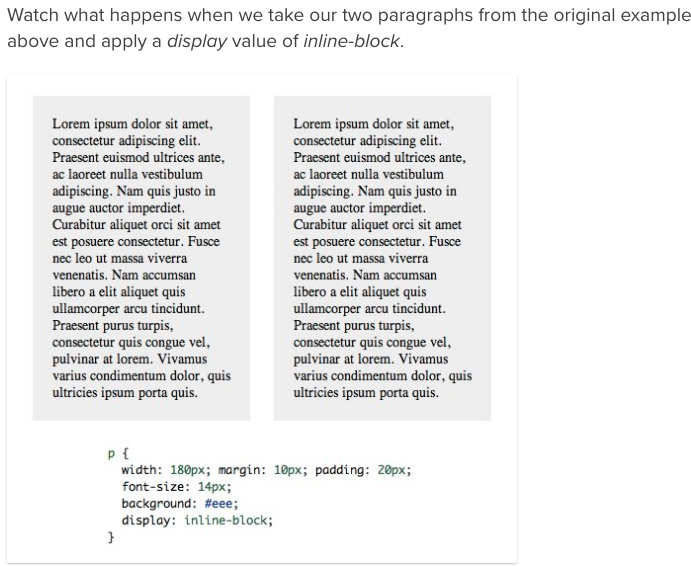
**Making Images Responsive**

Making images responsive with CSS is actually very simple. Instead of applying an absolute width to an element, you can use:

img {  
  max-width: 100%;  
  display: block;  
  height: auto;  
}

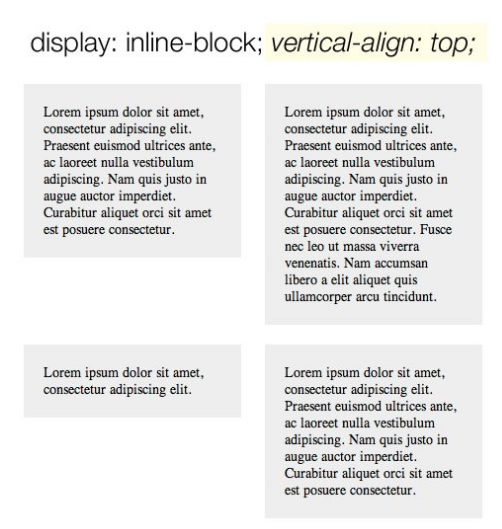
The max-width property of 100% scales the image to fit the width of its container, but the image won't stretch wider than its original width. Setting the display property to block changes the image from an inline element (its default), to a block element on its own line. The height property of auto keeps the original aspect ratio of the image.

Display: inline-block;



we’re telling the browser to display the paragraphs inline, but allow them to retain their block-level characteristics. This means we can set a width and height manually and have the two elements remain distinct, but also have them appear next to each other in the document flow

Vertical-align: top; (to be used in combo with the display-inline-block declaration.)



**Optimize for Retina Screens**

The simplest way to make your images appear "retina" (and optimize them for retina displays) is to define their width and height values as *only half of what the original file is*.

E.g.

<style>  
  img { height: 250px; width: 250px; }  
</style>  
<img src="coolPic500x500" alt="A most excellent picture">

Making Typography Responsive

Instead of using em or px to size text, you can use viewport units for responsive typography. Viewport units, like percentages, are relative units, but they are based off different items. Viewport units are relative to the viewport dimensions (width or height) of a device, and percentages are relative to the size of the parent container element.

The four different viewport units are:

* vw: 10vw would be 10% of the viewport's width.
* vh: 3vh would be 3% of the viewport's height.
* vmin: 70vmin would be 70% of the viewport's smaller dimension (height vs. width).
* vmax: 100vmax would be 100% of the viewport's bigger dimension (height vs. width).

E.g.:

h2 {

width: 80vw;

}

p {

width: 75vmin;

}

## Introduction to the CSS Flexbox Challenges

Intro:A website's User Interface ("UI") has two components. First, there are the visual elements, such as colors, fonts, and images. Second, there is the placement or positioning of those elements. In Responsive Web Design, a UI layout must accommodate many different browsers and devices accessing the content.  
  
CSS3 introduced Flexible Boxes, or flexbox, to create page layouts for a dynamic UI. It is a layout mode that arranges elements in a predictable way for different screen sizes and browsers. While somewhat new, all popular modern browsers support flexbox. This section covers how to use flexbox and the different layout options it offers.

FLEX

Placing the CSS property display: flex; on an element allows you to use other flex properties to build a responsive page.

Adding display: flex to an element turns it into a flex container. This makes it possible to align any children of that element into rows or columns. You do this by adding the flex-direction property to the parent item and setting it to row or column. Creating a row will align the children horizontally, and creating a column will align the children vertically.

Other options for flex-direction are row-reverse and column-reverse.

**Note**  
The default value for the flex-direction property is row.

E.g. header, footer {

display: flex;

flex-direction: row;(or could be row-reverse, column-reverse, column)

}

**Justify-content & Align-items for Flexbox**

Sometimes the flex items within a flex container do not fill all the space in the container. It is common to want to tell CSS how to align and space out the flex items a certain way. Fortunately, the justify-content property has several options to do this. But first, there is some important terminology to understand before reviewing those options.

There are several options for how to space the flex items along the line that is the main axis. One of the most commonly used is justify-content: center;, which aligns all the flex items to the center inside the flex container. Others options include:

* flex-start: aligns items to the start of the flex container. For a row, this pushes the items to the left of the container. For a column, this pushes the items to the top of the container.
* flex-end: aligns items to the end of the flex container.
* space-between: aligns items to the center of the main axis, with extra space placed between the items. The first and last items are pushed to the very edge of the flex container. For example, in a row the first item is against the left side of the container, the last item is against the right side of the container, then the other items between them are spaced evenly.
* space-around: similar to space-between but the first and last items are not locked to the edges of the container, the space is distributed around all the items

The align-items property is similar to justify-content. Recall that the justify-content property aligned flex items along the main axis. For rows, the main axis is a horizontal line and for columns it is a vertical line.

* Flex containers also have a **cross axis** which is the opposite of the main axis. For rows, the cross axis is vertical and for columns, the cross axis is horizontal.
* CSS offers the align-items property to align flex items along the cross axis. For a row, it tells CSS how to push the items in the entire row up or down within the container. And for a column, how to push all the items left or right within the container

Options:

* Flex-start, flex-end, center
* stretch: stretch the items to fill the flex container. For example, rows items are stretched to fill the flex container top-to-bottom.
* baseline: align items to their baselines. Baseline is a text concept, think of it as the line that the letters sit on.

FLEX-WRAP property

CSS flexbox has a feature to split a flex item into multiple rows (or columns). By default, a flex container will fit all flex items together. For example, a row will all be on one line.

However, using the flex-wrap property, it tells CSS to wrap items. This means extra items move into a new row or column. The break point of where the wrapping happens depends on the size of the items and the size of the container.

CSS also has options for the direction of the wrap:

* nowrap: this is the default setting, and does not wrap items.
* wrap: wraps items from left-to-right if they are in a row, or top-to-bottom if they are in a column.
* wrap-reverse: wraps items from right-to-left if they are in a row, or bottom-to-top if they are in a column.

SHRINK ITEMS in the Flex Container

So far, all the properties in the challenges apply to the flex container (the parent of the flex items). However, there are several useful properties for the flex items.

The first is the flex-shrink property. When it's used, it allows an item to shrink if the flex container is too small. Items shrink when the width of the parent container is smaller than the combined widths of all the flex items within it.

The flex-shrink property takes numbers as values. The higher the number, the more it will shrink compared to the other items in the container. For example, if one item has a flex-shrink value of 1 and the other has a flex-shrink value of 3, the one with the value of 3 will shrink three times as much as the other.

The opposite of flex-shrink is the flex-grow property. The flex-grow property controls the size of items when the parent container expands.

The flex-basis property specifies the initial size of the item before CSS makes adjustments with flex-shrink or flex-grow.

The units used by the flex-basis property are the same as other size properties (px, em, %, etc.). The value auto sizes items based on the content.

SHORTCUT

There is a shortcut available to set several flex properties at once. The flex-grow, flex-shrink, and flex-basis properties can all be set together by using the flex property.

For example, flex: 1 0 10px; will set the item to flex-grow: 1;, flex-shrink: 0;, and flex-basis: 10px;.

The default property settings are flex: 0 1 auto;

ORDER property

The order property is used to tell CSS the order of how flex items appear in the flex container. By default, items will appear in the same order they come in the source HTML. The property takes numbers as values, and negative numbers can be used.

SELF-ALIGN

The final property for flex items is align-self. This property allows you to adjust each item's alignment individually, instead of setting them all at once. *This is useful since other common adjustment techniques using the CSS properties float, clear, and vertical-align do not work on flex items.*

align-self accepts the same values as align-items and will override any value set by the align-items property.

## Introduction to the CSS Grid Challenges

CSS Grid helps you easily build complex web designs. It works by turning an HTML element into a grid container with rows and columns for you to place children elements where you want within the grid.

CSS Grid

Turn any HTML element into a grid container by setting its display property to grid. This gives you the ability to use all the other properties associated with CSS Grid.

**Note**  
In CSS Grid, the parent element is referred to as the container and its children are called items.

GRID COLUMNS & ROWS

To add some columns to the grid, use the grid-template-columns property on a grid container as demonstrated below:

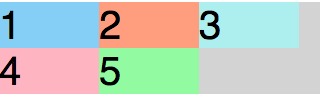
.container {  
  display: grid;  
  grid-template-columns: 50px 50px;  
}

This will give your grid two columns that are 50px wide each.

BEFORE grid-templates



AFTER grid templates-columns



The grid you created in the last challenge will set the number of rows automatically. To adjust the rows manually, use the grid-template-rows property in the same way you used grid-template-columns in previous challenge.

.container {

font-size: 40px;

width: 100%;

background: LightGray;

display: grid;

grid-template-columns: 100px 100px 100px;

grid-template-rows: 50px 50px;

}

CHANGING THE SIZE OF ROWS & COLUMNS

* fr: sets the column or row to a fraction of the available space,
* auto: sets the column or row to the width or height of its content automatically,
* %: adjusts the column or row to the percent width of its container.

Here's the code that generates the output in the preview:

grid-template-columns: auto 50px 10% 2fr 1fr;

This snippet creates five columns. The first column is as wide as its content, the second column is 50px, the third column is 10% of its container, and for the last two columns; the remaining space is divided into three sections, two are allocated for the fourth column, and one for the fifth.

Give Columns a GAP

To add a gap between the columns, use the grid-column-gap or between rows with grid-row-gap property like this:

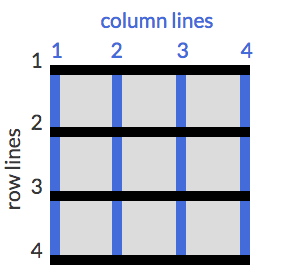
grid-column-gap: 10px;

grid-row-gap: 10px

grid-gap is a shorthand property for grid-row-gap and grid-column-gap from the previous two challenges that's more convenient to use. If grid-gap has one value, it will a create a gap between all rows and columns. However, if there are two values, it will use the first one to set the gap between the rows and the second value for the columns.

GRIDCOLUMN TO CONTROL SPACING (within the container)

Up to this point, all the properties that have been discussed are for grid containers. The grid-column property is the first one for use on the grid items themselves.



grid-column: 1 / 3;

This will make the item start at the first vertical line of the grid on the left and span to the 3rd line of the grid, consuming two columns.



OR grid-row: 3: 2

In CSS Grid, the content of each item is located in a box which is referred to as a *cell*. You can align the content's position within its cell horizontally using the justify-self property on a grid item. By default, this property has a value of stretch, which will make the content fill the whole width of the cell. This CSS Grid property accepts other values as well. (Use align-self to do it horizontally.)

* start: aligns the content at the left of the cell,
* center: aligns the content in the center of the cell,
* end: aligns the content at the right of the cell.

You can align them all at once horizontally by using justify-items

AREAS

You can group cells of your grid together into an *area* and give the area a custom name. Do this by using grid-template-areas on the container like this:

grid-template-areas:  
  "header header header"  
  "advert content content"  
  "footer footer footer";

The code above merges the top three cells together into an area named header, the bottom three cells into a footerarea, and it makes two areas in the middle row; advertand content.

**Note**  
Every word in the code represents a cell and every pair of quotation marks represent a row.

In addition to custom labels, you can use a period (.) to designate an empty cell in the grid.

After creating an areas template for your grid container, as shown in the previous challenge, you can place an item in your custom area by referencing the name you gave it. To do this, you use the grid-area property on an item like this:

.item1 { grid-area: header; }

This lets the grid know that you want the item1 class to go in the area named header. In this case, the item will use the entire top row because that whole row is named as the header area.

The grid-area property you learned in the last challenge can be used in another way. If your grid doesn't have an areas template to reference, you can create an area on the fly for an item to be placed like this:

item1 { grid-area: 1/1/2/4; }

When you used grid-template-columns and grid-template-rows to define the structure of a grid, you entered a value for each row or column you created.

REPEAT FUNCTION(skipped challenge)

Lets say you want a grid with 100 rows of the same height. It isn't very practical to insert 100 values individually. Fortunately, there's a better way - by using the repeatfunction to specify the number of times you want your column or row to be repeated, followed by a comma and the value you want to repeat.

Here's an example that would create the 100 row grid, each row at 50px tall.

grid-template-rows: repeat(100, 50px);

MinMAx Function

There's another built-in function to use with grid-template-columns and grid-template-rows called minmax. It's used to limit the size of items when the grid container changes size. To do this you need to specify the acceptable size range for your item. Here is an example:

grid-template-columns: 100px minmax(50px, 200px);

In the code above, grid-template-columns is set to create two columns; the first is 100px wide, and the second has the minimum width of 50px and the maximum width of 200px.

AUTO FILL

The repeat function comes with a option called *auto-fill*. This allows you to automatically insert as many rows or columns of your desired size as possible depending on the size of the container. You can create flexible layouts when combining auto-fillwith minmax.

In the preview, grid-template-columns is set to

repeat(auto-fill, minmax(60px, 1fr));

auto-fit works almost identically to auto-fill. The only difference is that when the container's size exceeds the size of all the items combined, auto-fill keeps inserting empty rows or columns and pushes your items to the side, while auto-fit collapses those empty rows or columns and stretches your items to fit the size of the container.