**Applied Visual Design Notes**

Adjust the background-color Property of Text

Instead of adjusting your overall background or the color of the text to make the foreground easily readable, you can add a background-color to the element holding the text you want to emphasize. This challenge uses rgba() instead of hex codes or normal rgb().

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. rgba() is great to use in this case, as it allows you to adjust the opacity. This means you don't have to completely block out the background.

You'll use background-color: rgba(45, 45, 45, 0.1) for this challenge. It produces a dark gray color that is nearly transparent given the low opacity value of 0.1.

To make the text stand out more, adjust the background-color of the h4 element to the given rgba() value.

Also for the h4, remove the height property and add padding of 10px.

h4 {

    text-align: center;

    padding: 10px;

    background-color: rgba(45,45,45,0.1)

  }

Add a box-shadow to a Card-like Element

The box-shadow property applies one or more shadows to an element.

The box-shadow property takes values for

* 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 and
* color, in that order.

The blur-radius and spread-radius values are optional.

Multiple box-shadows can be created by using commas to separate properties of each box-shadow element.

Here's an example of the CSS to create multiple shadows with some blur, at mostly-transparent black colors:

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

Decrease the Opacity of an Element

The opacity property 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 value given will apply to the entire element, whether that's an image with some transparency, or the foreground and background colors for a block of text.

Use the text-transform Property to Make Text Uppercase

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

The following table shows how the different text-transformvalues change the example text "Transform me".

| Value | Result |
| --- | --- |
| lowercase | "transform me" |
| uppercase | "TRANSFORM ME" |
| capitalize | "Transform Me" |
| initial | Use the default value |
| inherit | Use the text-transform value from the parent element |
| none | Default: Use the original text |

Adjust the Hover State of an Anchor Tag

This challenge will touch on the usage of pseudo-classes. 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 color of 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.

Note: Positioning gives you a lot of flexibility and power over the visual layout of a page. It's good to remember that no matter the position of elements, the underlying HTML markup should be organized and make sense when read from top to bottom. This is how users with visual impairments (who rely on assistive devices like screen readers) access your content.

Move a Relatively Positioned Element with CSS Offsets

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).

As you saw in the last challenge, using the top offset moved the h2 downwards. Likewise, using a left offset moves an item to the right.

Lock an Element to its Parent with Absolute Positioning

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. The CSS offset properties (top or bottom and left or right) are used to adjust the position.

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.

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

#searchbar {

    position:absolute;

    top: 50px;

    right:50px;

  }

Lock an Element to the Browser Window with 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 between the fixed and absolute positions is that an element with a fixed position won't move when the user scrolls.

The navigation bar in the code is labeled with an id of navbar. Change its position to fixed, and offset it 0 pixels from the top and 0 pixels from the left. After you have added the code, scroll the preview window to see how the navigation stays in place.

Push Elements Left or Right with the float Property

The next positioning tool does not actually use position, but sets the float property of an element. Floating elements are removed from the normal flow of a document and pushed to either the left or right of their containing parent element.

It's commonly used with the width property to specify how much horizontal space the floated element requires.

Change the Position of Overlapping Elements with the z-index Property

When elements are positioned to overlap (i.e. using position: absolute | relative | fixed | sticky), 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 (i.e. a whole number and not a decimal), and higher values for the z-index property of an element move it higher in the stack than those with lower values.

Add a z-index property to the element with the class name of first (the red rectangle) and set it to a value of 2 so it covers the other element (blue rectangle).

.first {

    background-color: red;

    position: absolute;

    z-index: 2;}

Center an Element Horizontally Using the margin Property

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.

Center the div on the page by adding a margin property with a value of auto.

Learn about Complementary Colors

Color theory and its impact on design is a deep topic and only the basics are covered in the following challenges. On a website, color can draw attention to content, evoke emotions, or create visual harmony.

Using different combinations of colors can really change the look of a website, and a lot of thought can go into picking a color palette that works with your content.

The color wheel is a useful tool to visualize how colors relate to each other - it's a circle where similar hues are neighbors and different hues are farther apart.

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.

Some examples of complementary colors with their hex codes are:

red (#FF0000) and cyan (#00FFFF)  
green (#00FF00) and magenta (#FF00FF)  
blue (#0000FF) and yellow (#FFFF00)

This is different than the outdated RYB color model that many of us were taught in school, which has different primary and complementary colors. Modern color theory uses the additive RGB model (like on a computer screen) and the subtractive CMY(K) model (like in printing). Read [here](https://en.wikipedia.org/wiki/Color_model) for more information on this complex subject.

There are many color picking tools available online that have an option to find the complement of a color.

Note: Using color can be a powerful way to add visual interest to a page. However, color alone should not be used as the only way to convey important information because users with visual impairments may not understand that content. This issue will be covered in more detail in the Applied Accessibility challenges.

Adjust the Hue of a Color

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.

Hue is what people generally think of as 'color'. If you picture a spectrum of colors starting with red on the left, moving through green in the middle, and blue on right, the hue is where a color fits along this line.

In hsl(), hue uses a color wheel concept instead of the spectrum, where the angle of the color on the circle is given as a value between 0 and 360.

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.

Here are a few examples of using hsl() with fully-saturated, normal lightness colors:

| Color | HSL |
| --- | --- |
| red | hsl(0, 100%, 50%) |
| yellow | hsl(60, 100%, 50%) |
| green | hsl(120, 100%, 50%) |
| cyan | hsl(180, 100%, 50%) |
| blue | hsl(240, 100%, 50%) |
| magenta | hsl(300, 100%, 50%) |

Example:

.cyan {

    background-color: hsl(180,100%,50%);

  }

Adjust the Tone of a Color

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. Alternatively, a tone is produced by adding gray or by both tinting and shading. Recall that the 's' and 'l' of hsl() stand for saturation and lightness, respectively.

The saturation percent changes the amount of gray and the lightness percent determines how much white or black is in the color. This is useful when you have a base hue you like, but need different variations of it.

Create a Gradual CSS Linear Gradient

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, on elements.

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, ...);

The first argument specifies the direction from which color transition starts - it can be stated as a degree, where 90deg makes a horizontal gradient (from left to right) and 45deg makes a diagonal gradient (from bottom left to top right). The following arguments specify the order of colors used in the gradient.

Example:

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

background: linear-gradient(35deg, #CCFFFF, #FFCCCC);

  }

Use a CSS Linear Gradient to Create a Striped Element

The repeating-linear-gradient() function is very similar to linear-gradient() with the major difference that it repeats the specified gradient pattern. repeating-linear-gradient() accepts a variety of values, but for simplicity, you'll work with an angle value and color stop values in this challenge.

The angle value is the direction of the gradient. Color stops are like width values that mark where a transition takes place, and are given with a percentage or a number of pixels.

In the example demonstrated in the code editor, the gradient starts with the color yellow at 0 pixels which blends into the second color blue at 40 pixels away from the start. Since the next color stop is also at 40 pixels, the gradient immediately changes to the third color green, which itself blends into the fourth color value red as that is 80 pixels away from the beginning of the gradient.

For this example, it helps to think about the color stops as pairs where every two colors blend together.

0px [yellow -- blend -- blue] 40px [green -- blend -- red] 80px

If every two color stop values are the same color, the blending isn't noticeable because it's between the same color, followed by a hard transition to the next color, so you end up with stripes.

Make stripes by changing the repeating-linear-gradient() to use a gradient angle of 45deg, then set the first two color stops to yellow, and finally the second two color stops to black.

Use the CSS Transform scale Property to Change the Size of an Element

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);

}

Use the CSS Transform scale Property to Scale an Element on Hover

The transform property has a variety of functions that let 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);

}

Note: Applying a transform to a div element will also affect any child elements contained in the div.

Use the CSS Transform scale Property to Scale an Element on Hover

The transform property has a variety of functions that let 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);

}

Note: Applying a transform to a div element will also affect any child elements contained in the div.

Use the CSS Transform Property skewX to Skew an Element Along the X-Axis

The next function of the transform property 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);

}

Use the CSS Transform Property skewY to Skew an Element Along the Y-Axis

Given that the skewX() function skews the selected element along the X-axis by a given degree, it is no surprise that the skewY() property skews an element along the Y (vertical) axis.

Create a Graphic Using CSS

By manipulating different selectors and properties, you can make interesting shapes. One of the easier ones to try is a crescent moon shape. For this challenge you need to work with the box-shadow property that sets the shadow of an element, along with the border-radius property that controls the roundness of the element's corners.

You will create a round, transparent object with a crisp shadow that is slightly offset to the side - the shadow is actually going to be the moon shape you see.

In order to create a round object, the border-radius property should be set to a value of 50%.

You may recall from an earlier challenge that the box-shadow property takes values for offset-x, offset-y, blur-radius, spread-radius and a color value in that order. The blur-radius and spread-radius values are optional.

Manipulate the square element in the editor to create the moon shape. First, change the background-color to transparent, then set the border-radius property to 50% to make the circular shape. Finally, change the box-shadow property to set the offset-x to 25px, the offset-y to 10px, blur-radius to 0, spread-radius to 0, and color to blue.

Create a More Complex Shape Using CSS and HTML

One of the most popular shapes in the world is the heart shape, and in this challenge you'll create one using pure CSS. But first, you need to understand the ::before and ::after pseudo-elements. These pseudo-elements are used to add something before or after a selected element. In the following example, a ::before pseudo-element 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;

}

For the ::before and ::after pseudo-elements to function properly, they must have a defined content property. This property is usually used to add things like a photo or text to the selected element.

When the ::before and ::after pseudo-elements are used to make shapes, the content property is still required, but it's set to an empty string. In the above example, the element with the class of heart has a ::before pseudo-element that produces a yellow rectangle with height and width of 50px and 70px, respectively.

This rectangle has round corners due to its 25% border-radius and is positioned absolutely at 5px from the left and 50px above the top of the element.

Learn How the CSS @keyframes and animation Properties Work

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:

animation-name sets the name of the animation, which is later used by @keyframes to tell CSS which rules go with which animations.

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.

Then CSS applies the magic to transition the element over the given duration to act out the scene. Here's an example to illustrate the usage of @keyframes and the animation properties:

#anim {

animation-name: colorful;

animation-duration: 3s;

}

@keyframes colorful {

0% {

background-color: blue;

}

100% {

background-color: yellow;

}

}

For the element with the anim id, the code snippet above sets the animation-name to colorful and sets the animation-duration to 3 seconds.

Then the @keyframes rule links to the animation properties with the name colorful. It sets the color to blue at the beginning of the animation (0%) which will transition to yellow by the end of the animation (100%).

You aren't limited to only beginning-end transitions, you can set properties for the element for any percentage between 0% and 100%.

Create an animation for the element with the id rect, by setting the animation-name to rainbow and the animation-duration to 4 seconds. Next, declare a @keyframes rule, and set the background-color at the beginning of the animation (0%) to blue, the middle of the animation (50%) to green, and the end of the animation (100%) to yellow.

Example:

div {

    height: 40px;

    width: 70%;

    background: black;

    margin: 50px auto;

    border-radius: 5px;

  }

  #rect {

    animation-name: rainbow;

    animation-duration: 4s;

  }

  @keyframes rainbow{

    0%{

      background-color: blue;

    }

    50%{

      background-color: green;

    }

    100%{

      background-color: yellow;

    }

  }

Modify Fill Mode of an Animation

That's great, but it doesn't work right yet. Notice how the animation resets after 500ms has passed, causing the button to revert back to the original color.

You want the button to stay highlighted.

This can be done by setting the animation-fill-mode property to forwards. The animation-fill-mode specifies the style applied to an element when the animation has finished. You can set it like so:

animation-fill-mode: forwards;

Set the animation-fill-mode property of button:hover to forwards so the button stays highlighted when a user hovers over it.

button:hover {

    animation-name: background-color;

    animation-duration: 500ms;

    animation-fill-mode: forwards;

  }

  @keyframes background-color {

    100% {

      background-color: #4791d0;

    }

  }

Create Movement Using CSS Animation

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 top property of the 50% keyframe to 50px, but having it set to 0px for the first (0%) and the last (100%) keyframe.

@keyframes rainbow {

0% {

background-color: blue;

top: 0px;

}

50% {

background-color: green;

top: 50px;

}

100% {

background-color: yellow;

top: 0px;

}

}

Add a horizontal motion to the div animation. Using the left offset property, add to the @keyframes rule so rainbow starts at 0 pixels at 0%, moves to 25 pixels at 50%, and ends at -25 pixels at 100%. Don't replace the top property in the editor - the animation should have both vertical and horizontal motion.

div {

    height: 40px;

    width: 70%;

    background: black;

    margin: 50px auto;

    border-radius: 5px;

    position: relative;

  }

  #rect {

    animation-name: rainbow;

    animation-duration: 4s;

  }

  @keyframes rainbow {

    0% {

      background-color: blue;

      top: 0px;

      left: 0px;

    }

    50% {

      background-color: green;

      top: 50px;

      left: 25px;

    }

    100% {

      background-color: yellow;

      top: 0px;

      left: -25px;

    }

  }

Create Visual Direction by Fading an Element from Left to Right

For this challenge, you'll change the opacity of an animated element so it gradually fades as it reaches the right side of the screen.

In the displayed animation, the round element with the gradient background moves to the right by the 50% mark of the animation per the @keyframes rule.

Target the element with the id of ball and add the opacity property set to 0.1 at 50%, so the element fades as it moves to the right.

#ball {

    width: 70px;

    height: 70px;

    margin: 50px auto;

    position: fixed;

    left: 20%;

    border-radius: 50%;

    background: linear-gradient(

      35deg,

      #ccffff,

      #ffcccc

    );

    animation-name: fade;

    animation-duration: 3s;

  }

  @keyframes fade {

    50% {

      left: 60%;

      opacity: 0.1;

    }

  }

Animate Elements Continually Using an Infinite Animation Count

The previous challenges covered how to use some of the animation properties and the @keyframes rule. 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 this case the animation will stop after running 3 times, but it's possible to make the animation run continuously by setting that value to infinite.

To keep the ball bouncing on the right on a continuous loop, change the animation-iteration-count property to infinite.

#ball {

    width: 100px;

    height: 100px;

    margin: 50px auto;

    position: relative;

    border-radius: 50%;

    background: linear-gradient(

      35deg,

      #ccffff,

      #ffcccc

    );

    animation-name: bounce;

    animation-duration: 1s;

    animation-iteration-count: infinite;

  }

  @keyframes bounce{

    0% {

      top: 0px;

    }

    50% {

      top: 249px;

      width: 130px;

      height: 70px;

    }

    100% {

      top: 0px;

    }

  }

Make a CSS Heartbeat using an Infinite Animation Count

Here's one more continuous animation example with the animation-iteration-count property that uses the heart you designed in a previous challenge.

The one-second long heartbeat animation consists of two animated pieces. The heart elements (including the :before and :after pieces) are animated to change size using the transform property, and the background div is animated to change its color using the background property.

Keep the heart beating by adding the animation-iteration-count property for both the back class and the heart class and setting the value to infinite. The heart:before and heart:after selectors do not need any animation properties.

.back {

    position: fixed;

    padding: 0;

    margin: 0;

    top: 0;

    left: 0;

    width: 100%;

    height: 100%;

    background: white;

    animation-name: backdiv;

    animation-duration: 1s;

    animation-iteration-count: infinite;

  }

  .heart {

    position: absolute;

    margin: auto;

    top: 0;

    right: 0;

    bottom: 0;

    left: 0;

    background-color: pink;

    height: 50px;

    width: 50px;

    transform: rotate(-45deg);

    animation-name: beat;

    animation-duration: 1s;

    animation-iteration-count: infinite;

  }

  .heart:after {

    background-color: pink;

    content: "";

    border-radius: 50%;

    position: absolute;

    width: 50px;

    height: 50px;

    top: 0px;

    left: 25px;

  }

  .heart:before {

    background-color: pink;

    content: "";

    border-radius: 50%;

    position: absolute;

    width: 50px;

    height: 50px;

    top: -25px;

    left: 0px;

  }

  @keyframes backdiv {

    50% {

      background: #ffe6f2;

    }

  }

  @keyframes beat {

    0% {

      transform: scale(1) rotate(-45deg);

   }

    50% {

      transform: scale(0.6) rotate(-45deg);

    }

  }

</style>

<div class="back"></div>

<div class="heart"></div>

Change Animation Timing with Keywords

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, which starts slow, speeds up in the middle, and then slows down again in the end. Other options include ease-out, which is quick in the beginning then slows down, ease-in, which is slow in the beginning, then speeds up at the end, or linear, which applies a constant animation speed throughout.

For the elements with id of ball1 and ball2, add an animation-timing-function property to each, and set #ball1 to linear, and #ball2 to ease-out. Notice the difference between how the elements move during the animation but end together, since they share the same animation-duration of 2 seconds.

.balls {

    border-radius: 50%;

    background: linear-gradient(

      35deg,

      #ccffff,

      #ffcccc

    );

    position: fixed;

    width: 50px;

    height: 50px;

    margin-top: 50px;

    animation-name: bounce;

    animation-duration: 2s;

    animation-iteration-count: infinite;

  }

  #ball1 {

    left:27%;

    animation-timing-function: linear;

  }

  #ball2 {

    left:56%;

    animation-timing-function: ease-out;

  }

  @keyframes bounce {

    0% {

      top: 0px;

    }

    100% {

      top: 249px;

    }

  }