ANIMATION FOR THE WEB: A GUIDE TO ADVANCED CSS AND ANIMATION TECHNIQUES

Rosemary Barke
HACC, Central Pennsylvania's Community College

LibreTexts™

HACC, Central Pennsylvania’s Community College

Animation for the Web: A Guide to Advanced CSS and Animation Techniques

Rosemary Barker

HACC, Central Pennsylvania’s Community College

Animation for the Web: A Guide to Advanced CSS and Animation Techniques

Rosemary A. Barker

Version 1.0

The LibreTexts libraries are Powered by NICE CXone Expert and was supported by the Department of Education Open Textbook Pilot Project, the California Education Learning Lab, the UC Davis Office of the Provost, the UC Davis Library, the California State University Affordable Learning Solutions Program, and Merlot. This material is based upon work supported by the National Science Foundation under Grant No. 1246120, 1525057, and 1413739.

This text was compiled on 02/23/2026

TABLE OF CONTENTS

Licensing

A detailed breakdown of this resource's licensing can be found in ***Back Matter/Detailed Licensing****.*

Dedication

Dedication

So many people have helped me in this journey to create this textbook. I would like to thank the following people for inspiration and encouragement:

- My students in the Web Department at the Harrisburg Area Community College. You guys make me a better teacher because you are wonderful people!
- My Dean, Dr. Jennie Baar, for her support and encouragement during this process.
- My mentor, Carl Petersheim, for his help in starting my journey in learning and teaching Web Design.
- My patient husband, Patrick Barker, that puts up with my long hours at the computer while writing this magnum opus. Your darling wifey loves you!
- My sweet son that I love, dearly! I love you to your eyeballs and back.
- My cheerleader and friend, my mother. Thank you for everything!

1: An Introduction to Animating with CSS

An Introduction to Animating with CSS

Learning Objectives

- Discover how animations work on websites

Animations on the Web

> ***Web animation*** *is "motion on the web," which is achieved through different technologies and techniques, or by the combination of those. Motion graphics, Sprite animations, CSS animations, JavaScript animation, SVG animations, and WebGL animations are among the most common techniques, but we could extend this to include UI animations, web-based Game animations, and animated Data Visualizations. Other ways to conceive Web animation is through the use of GIFs, Java Applets, Flash, and Shockwave, but these technologies are being less and less used over the last years, or even deprecated.* [1]

Web animations are everywhere! You'll find animations used to call attention to a particular item on a page or to confirm an action the viewer just took. Animations can give the viewer the sense of a real-world experience (like flipping the page in a book), or they can quickly explain how to bake a cake.

In the following sections, we'll look at how these animations work a bit more closely. Let's go!

Animations that Draw Attention

Websites frequently use a marketing technique called a **call to action** or a **CTA** for short. CTAs are little bits of text that encourage site visitors to take an action.

For example, you'll see a button or a text link that reads "Purchase Tickets" or "Sign Up."

Developers have noticed that, over time, site visitors get used to the layout of a page. The visitors don't interact with CTAs like the developers would hope they would. So what is to be done? Why animate the CTA, of course! Motion draws the eye and encourages interaction. Watch the video below to see an example of a CTA. There is no audio in the video.

Footer Animation from Magnetism.fr

Here’s a footer animation from Magnetism.fr . Magnetism is a Web agency that creates everything from websites and apps to content. Their animation draws the eye down to the footer in a fun and unique way.

Nowy Teatr Calendar Animation

The following animation we’ll be looking at is a calendar animation from Nowy Teatr . The calendar reveals an image for the event you have hovered over. Displaying an image is a great way to catch a site visitor’s attention. Let’s see it in action below:

Confirming an Action with Animations

CodePen.io has a ton of outstanding animation examples, along with the code you would need to implement them. A lovely example of an elegant submit button with a confirmation action was created by Valentin Galmand . In the Live Code Editor below, hover over the submit button and click it to see the animation. *Note - If the animation is sluggish, rerun it or view it on Codepen by clicking the “Edit On CodePen” link.*

Upload Progress Microinteraction

This intriguing little upload progress animation caught my attention because it does three things. First, it shows that you have clicked the upload button by changing its shape to a circle. Then, you see a progress bar that lets you know how far along the upload is. Finally, a green check shows that the upload is complete. This combination of three animations enables the site to visitors understand what is happening. *Note - If the animation is clipped on either side, hit the “0.5x” button to make it smaller and then press the upload button again.*

Cause and Effect Illusions: Real World Experience in the Digital Format

Cause and effect illusions in the digital world come in many forms. For example, have you ever seen a page-turn animation? Turning the page of a book in the real world has a distinct look and feel.

Page Flip Animation

Let’s take a look at an example of a digital page turn. Pizzabote’s “ Book Page Flip Animation ” is a beautiful example of the technique. Be sure to view it at the 0.5 size for the full effect and click on the right half of the ‘book.’

Cause and effect illusions are found frequently in mobile devices. Any time you swipe or pinch, you are using one!

“One of the best benefits of motion graphics is communicating cause-and-effect to users, particularly with gesture controls on a mobile device. Think of Tinder’s iconic swiping—the animation of the page being dragged left or right strengthens the illusion of interactivity. It feels like you’re actually moving something. Imagine if, when you swiped, the screen just blinked and loaded a new profile. The app wouldn’t be anywhere near as fun." [11]

Swipe Illusion

Here’s a great little example of a swipe illusion for a camera app on the phone by Vincent Tantardini . You can see the swipe effect when the viewer chooses the camera speed. Be sure to save it on Dribble!

Pinch and Pan

The last example we will see of the cause and effect illusions is the “Pinch and Pan” created by Benjamin den Boer for Framer. Check out his work on Dribble.

Explainer Videos

“As the name suggests, an **explainer video** is a type of video that aims to inform the viewer in a short time, typically no more than 2 minutes. It uses various creative techniques to capture and keep attention, from colorful animation to storytelling. In marketing, they are an invaluable tool for building brand awareness in an easy-to-watch and memorable way.” [13]

The animation examples we saw previously were implemented through HTML, CSS, and sometimes even a little JavaScript. Explainer videos are created with high-powered software like Adobe After Effects, Adobe Animate, and Adobe Premier Pro. These videos can be fully animated in 2D or 3D, or they can utilize video shot in the real world. Let’s look at two 2D animation examples.

The Google Wifi explainer video below is a great example of a short and sweet explainer video. Take a look.

Loading Screens

Loading screens are little animations you view while waiting for something else to happen. A great example of this would be the first screen you see when a video game is loading. [14]

You can also find these screens after you click a button to login into a private section of a website. An excellent way to visualize this is to think of the process that happens when you log into your online banking.

Look at a few of these SVG Loader Animations from Nikhil Krishnan . Be sure to drag the gray scroll bar on the right of the embedded content below.

Pencil Pre-loader by Mantas Bačiuška

Pre-loaders are loading animations that happen before you enter a website.

This mesmerizing pre-loader animation works quite well because it can be fast and light which won’t require too much from the server.

Transitions

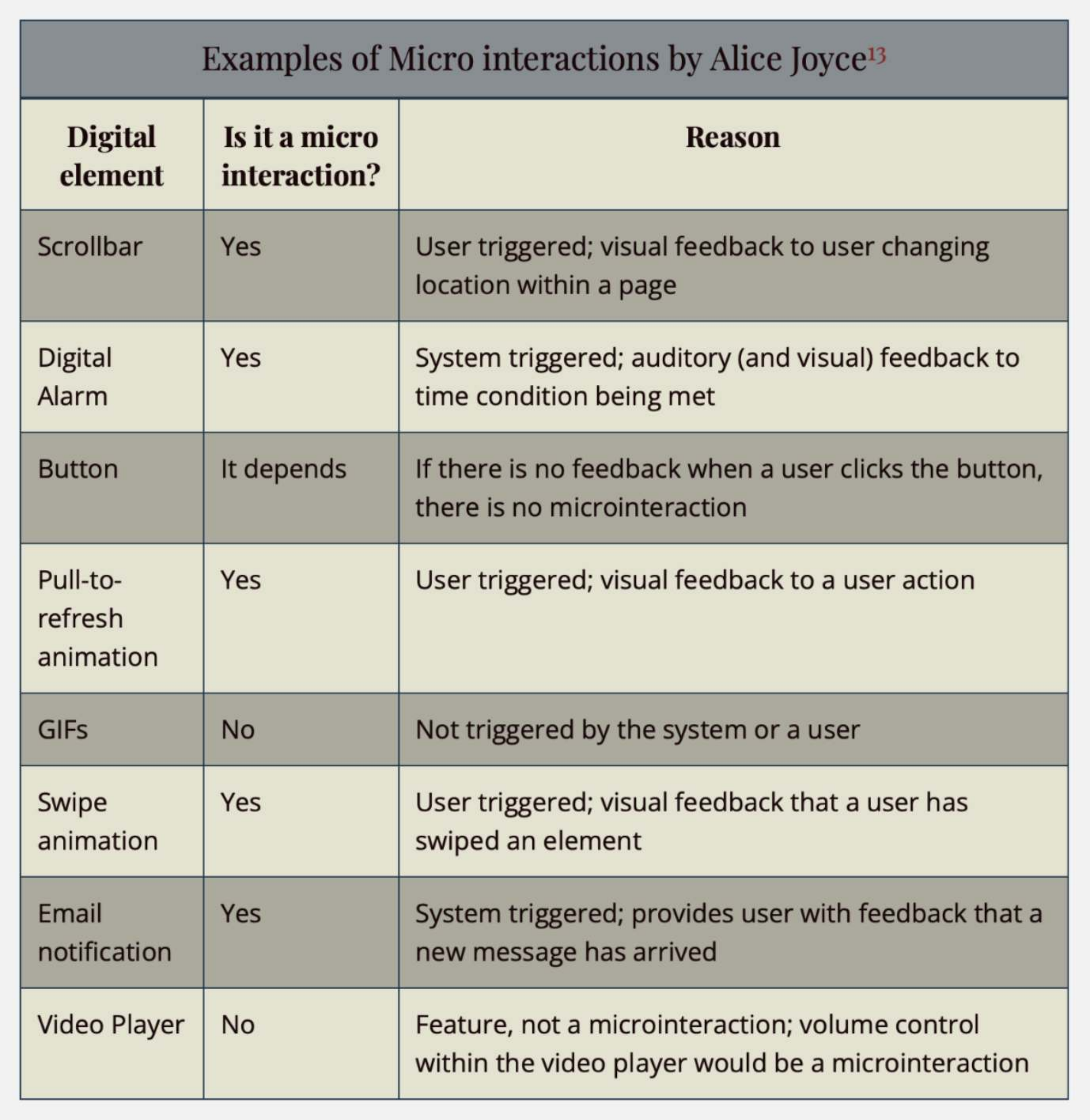
Transitions are the animated changes between two pages, states, or views to provide visual continuity to the user interface. [14] The most creative Web transitions are inspired by what is happening in the video today. Let’s look at some ideas below:

The following example, the portfolio of Hisami Kurita , shows some lovely wipe transitions that use circles. I was also delighted by how the cursor interacts with items on the page. Take a look.

Micro Interactions

“**Micro interactions** are trigger-feedback pairs in which (1) the trigger can be a user action or an alteration in the system’s state; (2) the feedback is a narrowly targeted response to the trigger and is communicated through small, highly contextual (usually visual) changes in the user interface.” [13] They can be used to encourage engagement, display system status, help with error prevention, and communicate the brand to the consumer as well. [13]

Here are some examples of what a micro interaction is (or is not as the case may be).



Examples of Micro interactions by Alice Joyce[13]		
Digital element	**Is it a micro interaction?**	**Reason**
Scrollbar	Yes	User triggered; visual feedback to user changing location within a page
Digital Alarm	Yes	System triggered; auditory (and visual) feedback to time condition being met
Button	It depends	If there is no feedback when a user clicks the button, there is no microinteraction
Pull-to-refresh animation	Yes	User triggered; visual feedback to a user action
GIFs	No	Not triggered by the system or a user
Swipe animation	Yes	User triggered; visual feedback that a user has swiped an element
Email notification	Yes	System triggered; provides user with feedback that a new message has arrived
Video Player	No	Feature, not a microinteraction; volume control within the video player would be a microinteraction

Micro interaction Example 1: Tab Bar by Abron Studios

This next short video shows a great little micro interaction when the tab bar is clicked. Notice the animation of the colors and tab name. Also, see how the entire tab bar wobbles slightly? That gives it a real-world illusion of being pressed, too. (No audio is present in the video.)

Micro interaction Example 2: Booking Interaction by Mauricio Bucardo

There is so much to love in this button micro interaction ! First, we see the button become slightly smaller when clicked. That click then launches the animation, and finally, we see the visual confirmation that the purchaser’s booking is now complete. This animation is gorgeous and very inventive.

Animated Logos

The last things that I want to show you all today are some animated logos. These are self-explanatory as to what they are at this point in your career here at HACC. So, let’s dive into the examples!

Animated Logo Example 1: Sello Logo by Latham Arnott

Sello is a Swedish company that offers many different products from other companies. The logo reflects this by showing a wide variety of items that could be purchased. Take a look below:

Animated Logo Example 2: Tyton Logo by Sava Stoic

Tyton offers government services. Sava Stoic did an excellent job on this animation by drawing the shield first, then emphasizing the ‘T’ in the middle of the shield. After the main animation, the company’s shield is revealed, and the shield does an outward warp. Take a look:

Summary

I hope you enjoyed looking at the real-world examples of web animations! I am passionate about bringing you the best examples to help you understand what can be done.

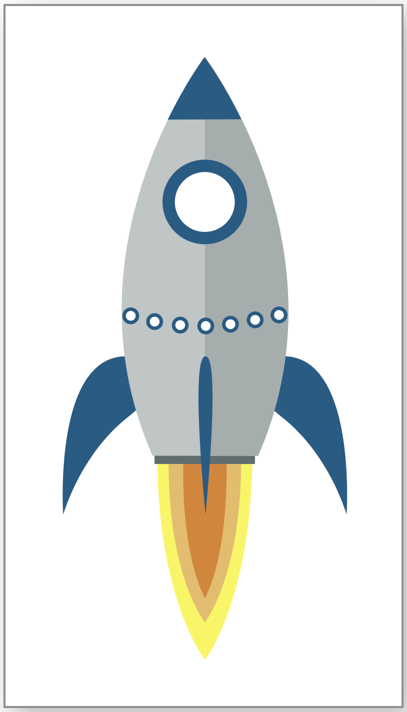
"See" you in the next chapter!

1.1: Introduction to CSS Animation

Overall Learning Objectives for Chapter 1

- Overview of the textbook
- How to Use CodePen
- Understanding Cartesian Coordinates

Introduction to CSS Animation



Welcome to the wonderful world of CSS Animations! In this textbook, we will be looking at manipulating HTML objects in 2D. These topics will be explored:

- Transforms
- Transitions and Eases
- Keyframe Animations
- Clip-paths and Shapes
- Variables and SVG Graphics
- Tools of the Trade
- Using Greensock (a JavaScript Library) for Animations

Before we begin the official journey on those topics, let's explore two concepts that will help you get the most out of this course: CodePen and Cartesian Coordinates. CodePen will be used to allow you to experiment with live code right here inside of our textbook. Cartesian Coordinates will be used to help us understand where to place our objects when we animate them.

Let's get started!

1.2: Becoming Familiar with CodePen

1.2 Learning Objectives

- Discovering CodePen
- Understanding How to Use CodePen in This Course

Becoming Familiar with CodePen

As we start working with CSS Animations, you will see many examples of code that I have created on CodePen. Some of you may not have had experience with working with CodePen and are curious about what CodePen is.

> ***"CodePen is a social development environment**. At its heart, it allows you to write code in the browser, and see the results of it as you build. A useful and liberating online code editor for developers of any skill, and particularly empowering for people learning to code. We focus primarily on front-end languages like HTML, CSS, JavaScript, and preprocessing syntaxes that turn into those things."*[1]

CodePen is a repository of coding projects from around the world. Some of the best minds put their work on there for all to see! I would strongly urge you to visit the site and browse through the many examples that can be found there. You will definitely pick up some inspiration for your future projects.

Walkthrough of CodePen

CodePen is a very simple live text editor that is fairly intuitive. When you are using the editor on Libretexts, it can look a bit different than what you see on CodePen's external site. I have a short walkthrough to help you with some of the differences.

One thing I would like to warn you about is that, for the best coding experience, I would strongly recommend that you use a Chrome browser on a desktop machine. Chrome does the best job with rendering the CodePen we will be working with.

> *Be sure that you are using the Chrome browser to view this textbook. Other browsers may or may not support CodePen well on this platform.*

CodePen is much more than what I showed you up above. Want to learn even more about CodePen? Take a look at this video from one of CodePen's founders, Chris Coyier.

Footnotes

1. CodePen. (n.d.). *About CodePen*. CodePen. Retrieved March 2, 2022, from https://codepen.io/about/

1.3: Working with Cartesian Coordinates

1.3 Learning Objectives

- Overview of Cartesian Coordinates
- Finding Coordinates in Adobe Software and Chrome

Overview of Coordinates

When coding for Web animations, it's important to know how a browser will calculate where objects are in its window. All browsers use a Cartesian coordinate system to do this.

Definition: Cartersian Coordinates

"**Cartesian coordinates** allow one to specify the location of a point in the plane, or in three-dimensional space. The Cartesian coordinates (also called rectangular coordinates) of a point are a pair of numbers (in two-dimensions) or a triplet of numbers (in three-dimensions) that specified signed distances from the coordinate axis."[1]

The Cartesian coordinate system consists of X, Y, and Z axes. As shown in figure 1 below, the X axis runs parallel to the bottom of your screen, the Y axis runs perpendicular to the X axis, and the Z axis provides a depth of field in moving away from the viewer and toward the viewer. You can see this in Figure 1 below.

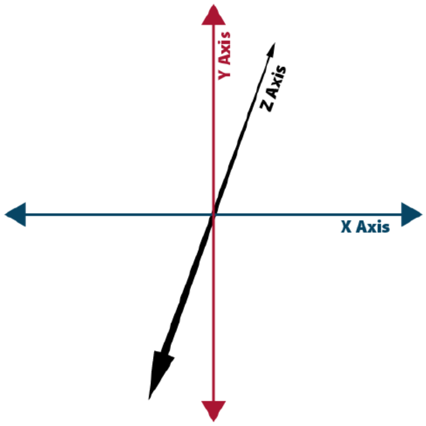


Figure 1: *X, Y, and Z Axes in a Cartesian Coordinate System*

Coordinates in a Web browser work much in the same way that coordinates in your Math classes do. The first number will be the X value and the second number will be the Y value. So, if you want a spot at 30 on the X axis and 20 on the Y axis you would notate that as "30, 20." So far, so good!

Web browsers use the lower right-hand quadrant of the X and Y axis AND **they use positive numbers on the Y axis.**

However, there is one big difference. Web browsers use the lower right-hand quadrant of the X and Y axis AND **they use positive numbers on the Y axis.** You can see this in Figure 2 below. If you remember your Math class, you will remember that that particular Y axis is typically denoted in negative numbers. Make sure that you remember this difference when we start using coordinates later in the course.

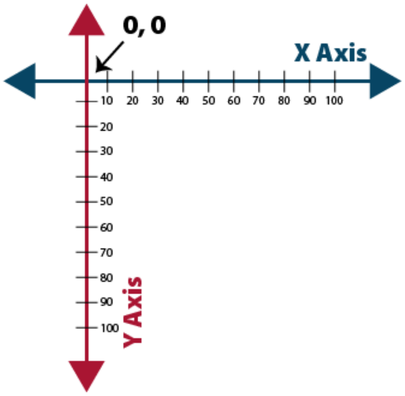


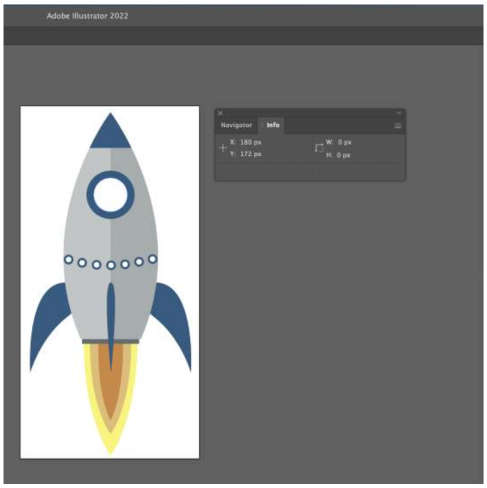
Figure 2: *How Coordinates Work in Web Browsers*

How to Find Coordinates in Adobe Software and Chrome

As you're animating, from time to time you will need to know the exact coordinates of an object so that you can calculate a path for it to follow. How do you do this? Luckily, there are some pretty great options to help you do this.

First, in Adobe programs like Illustrator and Photoshop, there is an information panel of some kind that will show you the location of where your mouse is on the artboard. To see this window in either Photoshop or Illustrator, go to Window > Info. Then, just look for the "X:" and "Y:"coordinates that are listed in the panel.

coo



X: 180 px
Y: 172 px
W: 0 px
H: 0 px

Figure 3: *How Coordinates Work in Web Browsers*

Another nice tool to keep in mind is the Coordinates Extension. This extension allows you to see coordinates as you mouse over an HTML page when you have it open in Chrome. Sometimes, it's just nice to see things in a live environment!

Footnotes

[1] *"Cartesian Coordinates." Math Insight, Math Insight, https://mathinsight.org/cartesian_coordinates*.

2: Understanding 2D CSS Transforms

Understanding CSS Transforms

Chapter 2: Overall Learning Objectives

- Understand what CSS Transforms are
- Learn to Apply 2D CSS Transforms
- Work with best practices for 2D CSS Transforms

Introduction to CSS Transforms

What are CSS Transforms?

The CSS transform property was a part of the first draft of the CSS3 Animations specification that was introduced by the W3C on March 20, 2009.[1] Transforms can be applied in both the 2D and 3D spaces.

Transforms are like snapshots of where an object is or how it looks at a given moment in time rather than the object fluidly going from one point to another point. They're a great place to start our journey of learning animation together.

Definition: CSS Transform

"The **transform** CSS property lets you rotate, scale, skew, or translate an element. It modifies the coordinate space of the CSS visual formatting model."[2]

Concept

Transforms are like snapshots of where an object is or how it looks at a given moment in time rather than the object fluidly going from one point to another point.

Let's look at the different types of 2D transforms that can be used: rotate, scale, skew, translate, transform-origin, and transform style.

Footnotes

1. Jackson, Dean, et al. “CSS 3D Transforms Module Level 3.” *CSS 3D Transforms Module Level 3*, Word Wide Web Consortium (W3C), 20 Mar. 2009, https://www.w3.org/TR/2009/WD-css3-3d-transforms-20090320/.
2. “Transform - CSS: Cascading Style Sheets: MDN.” *CSS: Cascading Style Sheets | MDN*, MDN Web Docs, https://developer.mozilla.org/en-US/docs/Web/CSS/transform.

2.1: 2D CSS Transforms - rotate()

Learning Objectives

- Gain an understanding of rotate()

2D CSS Transforms - rotate()

The first of our CSS Transforms that we will be looking at today is **rotate()**.

Definition: rotate()

When you **rotate** an element, you are spinning it in either a clockwise or counterclockwise motion around the object's center point.

Here are some commonly used angles for rotation:

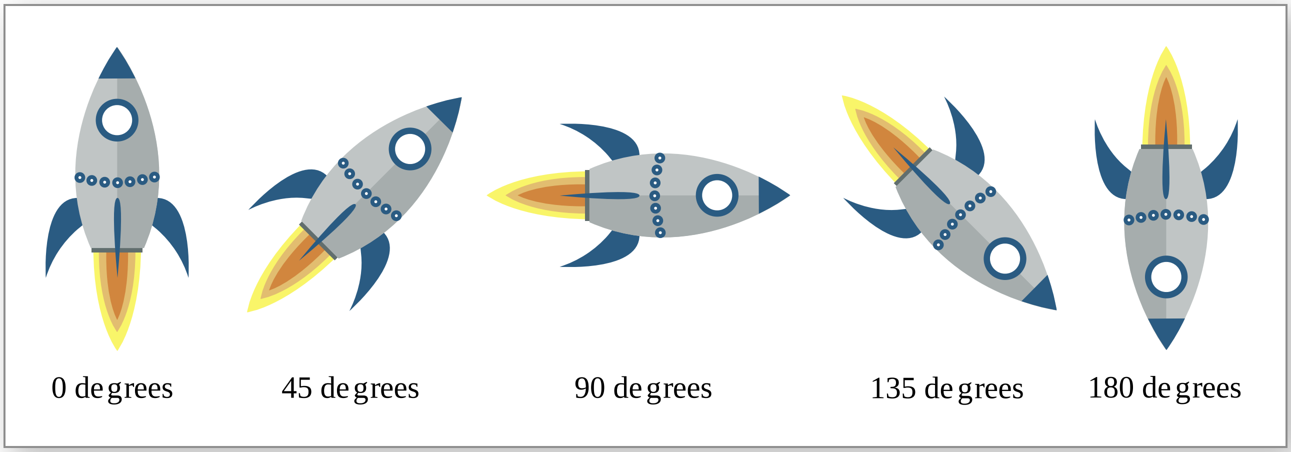


Figure 1: Common angles of rotation

The way you set up this CSS property value is like this:

Concept 2.1.1

```
selector {
        transform: rotate();
}
```

You set the rotation value by inserting a number from 0 through 180 followed by 'deg' like this:

Example 2.1.1

```
selector {
        transform: rotate(45deg);
}
```

Definition: Point Reflection

A rotation by 180° is called **point reflection**.

Ready to play around with some code? I have included a live coding experience from CodePen for you below. Follow the instructions in the exercise as you type in the code editor.

Exercise 2.1.1

1. Open this page in Chrome if you have not done so already.
2. Toggle back and forth between the CSS and HTML buttons. Note the code in each of those windows.
3. In the CSS panel, create a transform property on the rotate ID. Set the value of the rotation property on the rocket to 90 degrees. (**Note:** You may or may not be able to see the cursor in the CodePen. If this happens, just go ahead and type like normal. If that is too confusing, click on the "Edit on CodePen" button to be taken to their site for easier editing.)
4. Then, add a '-' sign in front of the number so that it reads like this: -90deg. Notice what happened to the rocket.
5. Play around with two or three more values that you make up but keep those values between 0 and 180.
6. Need help? Click on the "Answer" link below the CodePen to expand more information.

Answer

After step #4, you would have this in the CSS code:

```
#rotate {
        transform: rotate(-90deg);
}
```

Footnotes

1. MDN Web Docs. (2022, January 31). *Rotate() - CSS: Cascading style sheets: MDN*. CSS: Cascading Style Sheets | MDN. Retrieved March 1, 2022, from https://developer.mozilla.org/en-US/docs/Web/CSS/transform-function/rotate().
2. MDN Web Docs. (2022, January 31). *Rotate() - CSS: Cascading style sheets: MDN*. CSS: Cascading Style Sheets | MDN. Retrieved March 2, 2022, from https://developer.mozilla.org/en-US/docs/Web/CSS/transform-function/rotate()#syntax.

2.2: 2D CSS Transforms - scale()

Learning Objectives

- Gain understanding of scale(), scaleX(), and scaleY().

2D CSS Transforms - scale()

Now, let's look at our next 2D Transform: scale().

Definition: scale()

When you **scale** an element, you increase or decrease the size of an element (according to the parameters given for the width and height).[1]

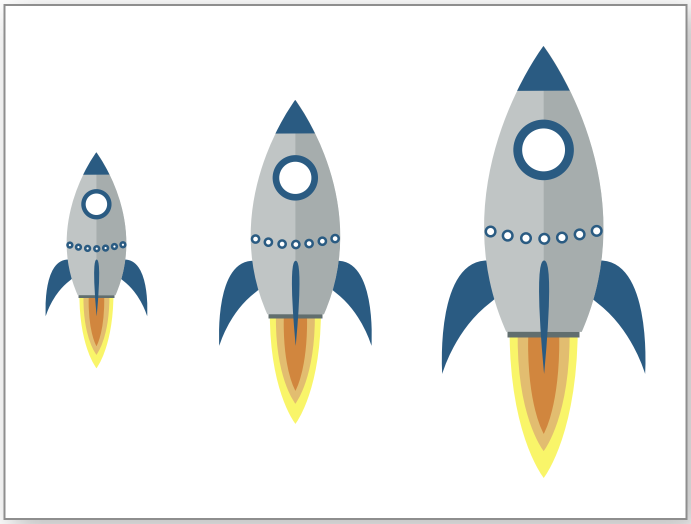


Figure 1: Scaling an element

The way you set up this CSS property value is like this:

Concept 2.2.1

selector {

transform: scale(*width*, *height*);

}

You set the scale value by inserting a whole number, decimal value, or percentage. The value in the example below will make the object 25% of its regular size.

Concept 2.2.2

selector {

transform: scale(*0.25*, *0.25*);

}

The values in this example will increase the size of the object by 5 times for width and 6 times for the height:

✓ Example 2.2.1

```
selector {
transform: scale(5, 6);
}
```

Let's give this a go in CodePen:

? Exercise 2.2.1

1. Open this page in Chrome if you have not done so already.
2. Toggle back and forth between the CSS and HTML buttons. Note the code in each of those windows.
3. In the CSS panel, create a transform property on the scale ID. Set the value of the scale property to .75 on the width and .25 on the height.
4. Notice what happened to the rocket.
5. Play around with two or three more values that you make up that are whole numbers. (I'd advise keeping these numbers on the smaller side because you may crash the browser if the image is too large.)
6. Need help? Click on the "Answer" link below the CodePen to expand more information.

Answer

```
#scale {
transform: scale(.75, .25);
}
```

Did you notice how squashed the rocket looked because of the height being set at a different size than the width? This mismatch of values altered the aspect ratio of the rocket.

Definition: Aspect Ratio

"The **aspect ratio** is the proportional relationship between an images with and height. Essentially, it describes an image's shape."[2]

The trick with using scale() with two different values is to pick the graphic you use it upon very carefully. For example, a plain box would work very well whereas our rocket did not work well at all. Our eyes expect the rocket to be taller than it is wide, so the squashed version doesn't feel natural to us at all.

However, we have seen rectangles in all kinds of different shapes and orientations both vertical and horizontal throughout our lives. Our eyes don't view the squashing of the rectangle as anything out of the ordinary due to this.

> *The trick with using scale() with two different values is to pick the graphic you use it upon very carefully.*

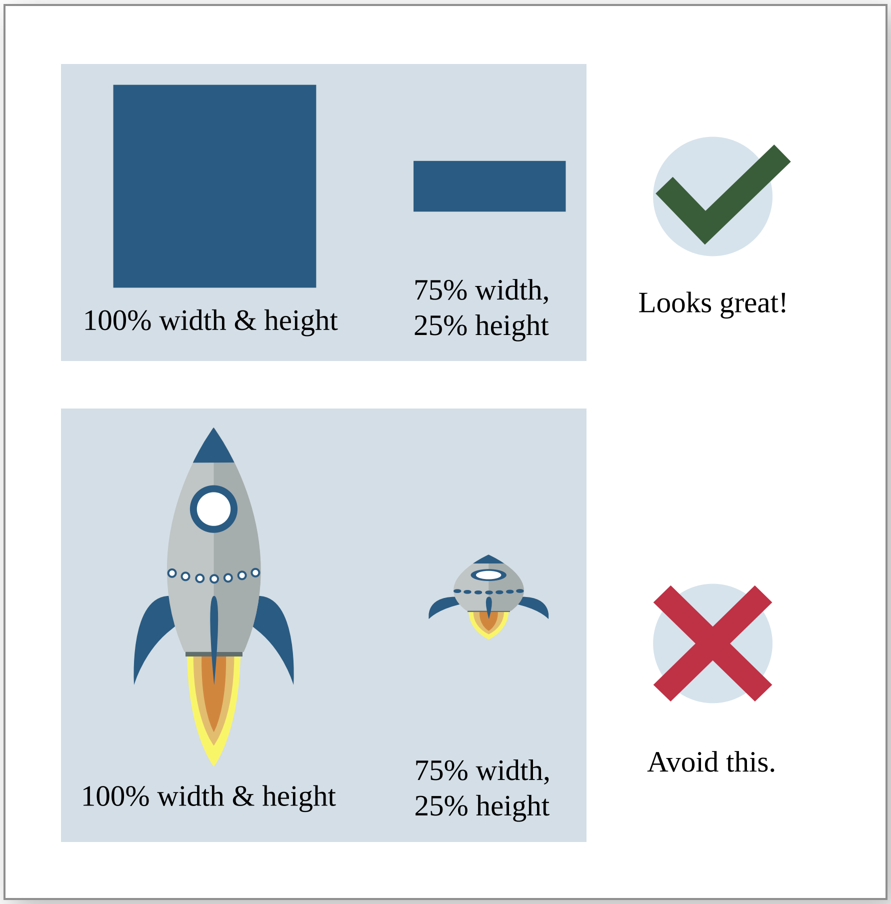


Figure 2: Changing the aspect ratio with scale()

So, how do you change the scale of the rocket and keep the aspect ratio intact? Easy! You just specify one number rather than two.

Example 2.2.4

```
selector {
transform: scale(5);
}
```

Now, give it a whirl in our CodePen again:

Exercise 2.2.2

1. In the CSS panel, create a transform property on the scale ID. Set a singular value of the scale property to 5.
2. Notice what happened to the rocket.
3. Play around with two or three more values that you make up.
4. Need help? Click on the "Answer" link below the CodePen to expand more information.

Answer

```
#scale {

transform: scale(5);

}
```

Now that we have looked at scale() in depth, let's explore two other types that are available to us: scaleX() and scaleY().

2D CSS Transforms - scaleX() and scaleY()

The scaleX() and scaleY() functions in CSS both change the size of an object on an HTML page. However, they both work with just one axis on the Cartesian Coordinate System. The scaleX() function works on the horizontal axis like you see below:

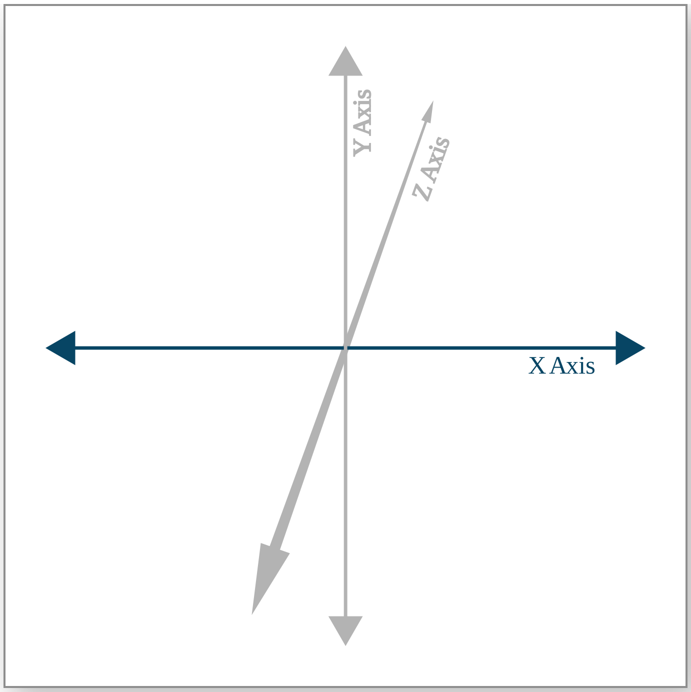


Figure 3: The X Axis (Horizontal)

Definition: scaleX()

The scaleX() CSS function defines a transformation that resizes an element along the x-axis (horizontally).[3]

The code for working with the X axis is very similar to what we saw above with scale(). The only difference is that you will have the letter 'X' just before the parentheses like what you see in the example below. The 'X' is marked in bold font to make it easier to see.

✓ Example 2.2.5

selector {

transform: scale**X**(5);

}

And, you guessed it, the scaleY() function will work on the vertical axis like this:

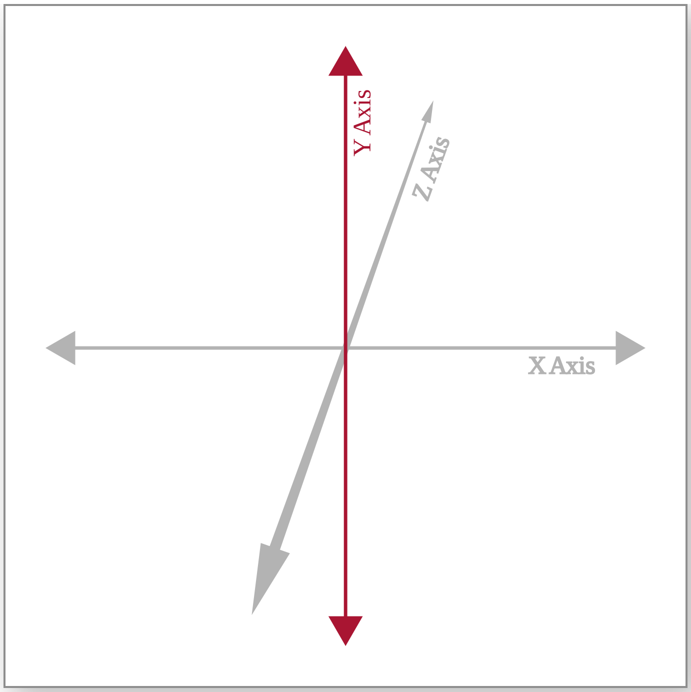


Figure 4: The Y Axis (Vertical)

Definition: scaleY()

The scaleY() CSS function defines a transformation that resizes an element along the y-axis (vertically).[4]

Like the scale() function, the scaleX() and scaleY() functions also take whole numbers, decimal values, or percentages. Let's try this out in the exercise below.

? Exercise 2.2.3

Reminder - CodePen works best in the Chrome browser.

1. Analyze the code you see below. Notice that scaleX() is the item we are working with first.

2. Change the value inside of the parentheses from '1' to '1.5.' What happened to the rocket?

3. Change the value inside of the parentheses from '1.5' to '50%.' What happened to the rocket this time?

4. Now, let's switch functions from scaleX() to scaleY(). Upate the code now.

5. In the new scaleY() function, set the value inside of the parentheses to '150%.' What happened to the rocket?

6. Try a few more whole numbers, fractions, and percentages. What worked well and what didn't?

7. Need help? Click on the "Answer" link below the CodePen to expand more information.

Answers

Step 2: #scaleXY {
transform: scaleX(1.5);
}

Step 3: #scaleXY {
transform: scaleX(1.5);
}

Step 5: #scaleXY {
transform: scaleY(150%);
}

Footnotes

1. "The Scale() Method." *w3schools.Com*, w3schools, https://www.w3schools.com/css/css3_2dtransforms.asp.
2. "Understanding Aspect Ratios." *Squarespace Help Center*, https://support.squarespace.com/hc/en-us/articles/115008538927-Understanding-aspect-ratios.
3. "ScaleX() - CSS: Cascading Style Sheets: MDN." *CSS: Cascading Style Sheets | MDN*, MDN Web Docs, https://developer.mozilla.org/en-US/docs/Web/CSS/transform-function/scaleX.
4. "ScaleY() - CSS: Cascading Style Sheets: MDN." *CSS: Cascading Style Sheets | MDN*, MDN Web Docs, https://developer.mozilla.org/en-US/docs/Web/CSS/transform-function/scaleY.

2.3: 2D CSS Transforms - skew()

Learning Objectives

- Gain an understanding of skew() in the horizontal, vertical, and all directions.

2D CSS Transforms - skew()

When you skew an object in CSS you introduce a slant to it. You can slant it right, left, up, and down! You can even slant it in two directions at once.

Definition: Term

The skew() "transformation is a shear mapping that distorts each point within an element by a certain angle in the horizontal and vertical directions. The effect is as if you grabbed each corner of the element and pulled them along a certain angle."[1]

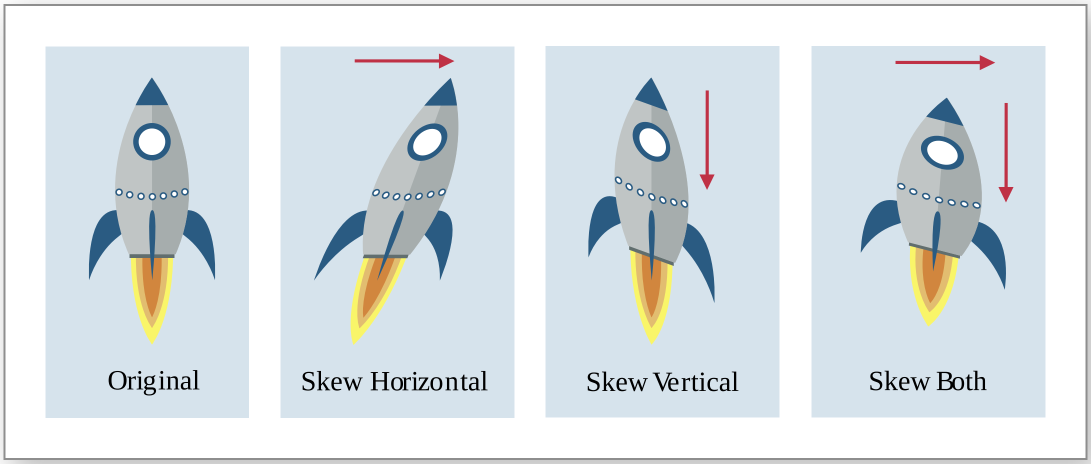


Figure 1: Skewing on Horizontal, Vertical, and Both Axes

Skewing on the X Axis (Horizontal)

In an animation environment, this is a truly useful property when it comes to making something wobble especially from side to side. The skew() function can be set with either one or two values measured in degrees. If you use just one value, the object will be affected on the x-axis only. Let's look at how to code this.

Concept 2.3.1

selector {

transform: skew(Xdeg);

}

**In this example, there are two things being represented by "Xdeg." X is equal to the numerical amount and degrees will be represented by the abbreviation 'deg' in your code.

So, if we put this into code, it would look like what you see below. I have entered '20' for the numerical amount of degrees to skew the rocket. This will slant the rocket 20 degrees to the left.

Example 2.3.1

```
.rocket {
transform: skew(20deg);
}
```

Now, it's your turn to play with the code. Here's our next CodePen.

Exercise 2.3.1

1. In a Chrome browser window, type "transform: skew(20deg);" between the open and closing curly braces for the #skew selector. Notice how the rocket leans to the left.

2. Next, let's try a negative value and see how that affects the rocket. Type "transform: skew(-20deg);" between the open and closing curly braces for the #skew selector. The rocket now leans to the right instead of the left.

3. Play with a few more values and see what they do. Use both positive and negative numbers.

Answers

1. .rocket {

```
transform: skew(20deg);
}
```

2. .rocket {

```
transform: skew(-20deg);
}
```

Skewing on the Y Axis (Vertical)

Skewing an object on just the Y Axis will require us to use two values for the skew() function. It will look like this:

Concept 2.3.2

```
selector {
transform: skew(0deg, Ydeg);
}
```

Did you see how the 'X' value changed in this one? There is a zero replacing the number there. That is because we don't want the X value to change. Only the Y value should change this time around. Let's see it in real CSS code.

Example 2.3.2

```
.rocket {
transform: skew(0deg, 20deg);
}
```

Here's the CodePen for you to try it out on.

Exercise 2.3.2

1. In a Chrome browser window, type "transform: skew(0deg, 20deg);" between the open and closing curly braces for the #skew selector. Notice how the rocket slants downward.

2. Next, let's try a negative value and see how that affects the rocket. Type "transform: skew(0deg, -20deg);" between the open and closing curly braces for the #skew selector. The rocket now upward instead of downward.

3. Play with a few more values and see what they do. Use both positive and negative numbers.

Answer

1. .rocket {

transform: skew(0deg, 20deg);

}

2. .rocket {

transform: skew(0deg, -20deg);

```
}
```

Skewing on Both the X and Y Axes

Skewing on both the X and Y axes is quite easy. You just change both of the values for the parameters rather than just one.

Concept 2.3.3

```
selector {
transform: skew(Xdeg, Ydeg);
}
```

Here is how it would look in CSS code:

✓ Example 2.3.3

```
.rocket {
transform: skew(30deg, 30deg);
}
```

Here's the CodePen for you to try it out on.

? Exercise 2.3.3

1. In a Chrome browser window, type "transform: skew(30deg, 30deg);" between the open and closing curly braces for the #skew selector. Notice how the rocket slants downward.

2. Next, let's try a negative value and see how that affects the rocket. Type "transform: skew(30deg, -30deg);" between the open and closing curly braces for the #skew selector. The rocket now faces upward instead of downward.

3. Play with a few more values and see what they do. Use both positive and negative numbers.

Answer

1. .rocket {

   transform: skew(30deg, 30deg);

```
}

2. .rocket {

transform: skew(30deg, -30deg);

}
```

Footnotes

[1] "Skew() - CSS: Cascading Style Sheets: MDN." *CSS: Cascading Style Sheets | MDN*, MDN Web Docs, https://developer.mozilla.org/en-US/docs/Web/CSS/transform-function/skew.

2.4: 2D CSS Transforms - translate()

Learning Objectives

- Gain an understanding of translate() along the X and Y axes.

2D CSS Transforms - Translate() on the X Axis

When an object is translated, that means that it is moved from one point to another point along a specific axis.

Definition: Translate

The **translate() function** allows you to transfer an element from one place to another along the X (horizontal) axis and the Y (vertical) axis. This is similar to how you might think of moving an element using offsets, like left, right, up, and down.[1]

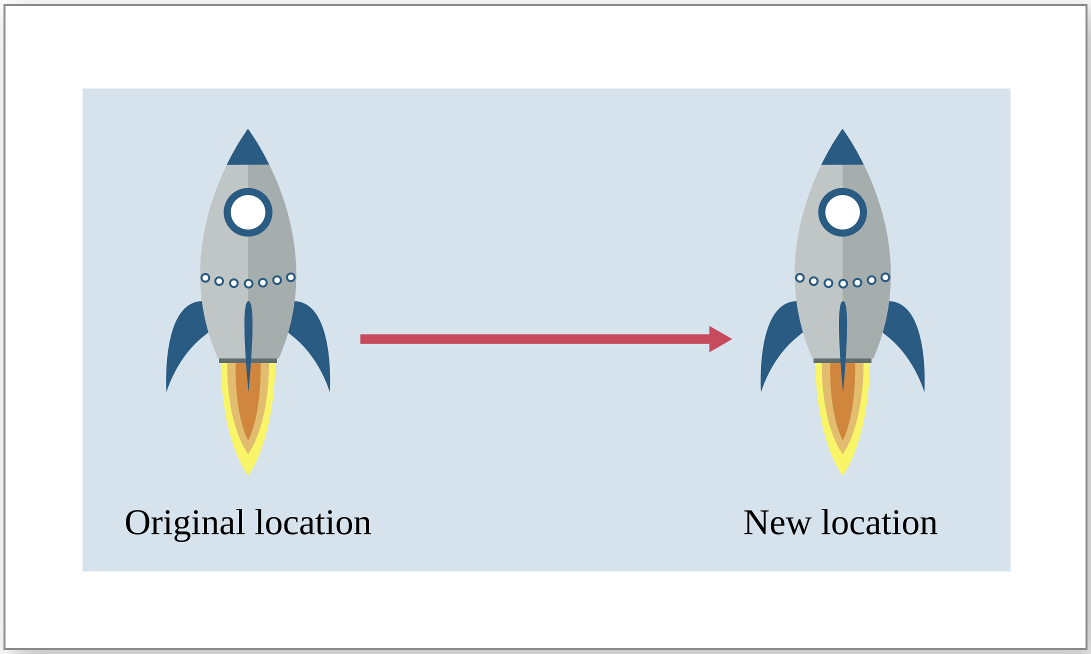


Figure 1: Translating on the X axis.

There are two varieties of translate that you will encounter in CSS: the property and the function. When using translate as a **CSS property**, it will be used as you see below:

Concept - Translate Property

```
selector {
        translate: someValue;
}
```

Notice that "translate" is to the left of the colon in this instance. This is what differentiates it from the translate() function. The "someValue" can be replaced with a size of your choice.

The translate() function looks like what we have been working with in previous parts of this chapter. You'll see "transform" for the property rather than "translate."

Concept - Translate() Function on X Axis

```
selector {
        transform: translate(someValue);
}
```

The key thing to remember about these two ways of working with Translate is that they both do exactly the same thing. So why have two different ways of working with translation? There must be a difference, right? Correct! There is a difference. That difference comes from the fact that the translate() function does not support working with the Z axis.[1]

However, there is a drawback to using the translate property instead of using it as a function of transform. Not all browsers support this feature just yet. To see what browsers do support using the translate property, be sure to check out the "CSS Property: Translate" chart from CanIUse.com.

> *Not all browsers support using translate as a property just yet. Use it as a function for now.*

Let's work with the code a bit now.

Exercise 2.4.1

1. Open Chrome and then follow the steps below.

2. Next, let's use the **translate() function.** Insert 200px for the value inside of the parentheses and notice what happens.

3. Then, put -200px as the value inside of the parentheses. The rocket is now sitting off the left hand side of the browser window where you can't see it! This can be useful when you need an object to be 'off stage' before it starts to animate onscreen.

4. Try a few more locations on your own.

Answers

```
2. .rocket {
        transform: translate(200px);
   }
3. .rocket {
        transform: translate(-200px);
    }
```

Translate() on the Y Axis

To move an object on the Y Axis, you simply specify two, comma-separated values inside of the parentheses of the translate() function. It will look like this:

Concept - Translate() function on Y Axis

```
selector {
        transform: translate(xValue, yValue);
}
```

**The 'xValue' is for the X axis amount and the 'yValue' is for the Y axis amount.

Let's try this out in the CodePen.

Exercise 2.4.2

1. In Chrome, add transform: translate(*0px, 200px*); between the two curly braces. Notice how the rocket goes straight down.

2. Next, change the Y value to -200 px. What happens? Where do you expect it to be? That's right! It moved 200px straight up off screen. (Negative number move objects off screen.)

3. Play around with a few more positive and/or negative numbers on your own.

Answer

```
1. .rocket {
transform: translate(100px, 200px);
}
```

```
2. .rocket {
transform: translate(200px, -200px);
}
```

Footnotes

[1] Coyier, Chris, et al. “Translate: CSS-Tricks.” *CSS Tricks*, CSS Tricks, 9 Nov. 2021, https://css-tricks.com/almanac/properties/t/translate/.

2.5: Transform-origin Property

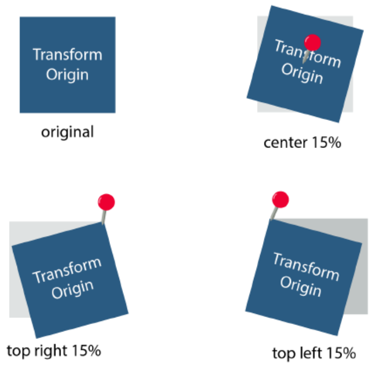
Transform-origin

> The **transform-origin property** is a helper property. It is used in conjunction with other functions like rotate().

Imagine this scenario. You’ve got a square that you put on a corkboard with a pushpin. But you don’t push the pin all the way into the board…you let the square have a little space to rotate.

Your square will then move around the pushpin (the origin point) as you move the square on the board. If you pick up the pin and move it to a different location, the square will then rotate around that new location.

In the example to the below, you'll see four different graphics. The 'original' is our square without any rotation applied to it. Hidden behind this original blue square is a gray square. This gray square will stay in place as we rotate the blue one. Because the gray square doesn't move, you can use this as a visual reference to see what the blue square will be doing.



original

center 15%

top right 15%

top left 15%

In the top right of the example, there is a red push pin in the center of the boxes. This shows the location of the origin point. *(Note - This is also called the 'transform origin' in some literature. We'll be using "origin point" for this class.)*

So, the transform-origin property spins an object around its origin point.

✎ Definition: Transform-origin property

*The **transform-origin property** spins an object around its origin point.*

✎ Definition: Origin Point (aka transform origin)

*The **origin point** (aka transform origin) is the point around which a transformation is applied.*[1]

By the way, *the default origin point is the center of any object.*[1] So, the only time you'll define that an object should rotate at its center point is after you made it rotate at a different point, and then you need to have it return to the center point.

✎ Definition: Default Origin Point

*The **default origin point** is the center of any object.*[1]

Here's the syntax for defining the properties of transform-origin:[2]

Concept: Transform-origin syntax

Basic syntax:

transform-origin: x-axis value | y-axis value | z-axis value;

You can use keywords like this:

transform-origin: top left; (Just the x and y are set)

Plus, you can use percentages and specific lengths like this:

transform-origin: 50% 75% 5cm; (All three values are set)

Both the x and y values can be set with keywords, a specific length, or a percentage.[2] Here is what you can use for these:[2]

X-axis values:

- Keywords: Left, Center, or Right
- Specific Lengths: mm, cm, inches, etc.. See the article from Mozilla for more.
- Percentages

Y-axis values:

- Keywords: Top, Center, or Bottom (notice how these are different than the x-axis values!)
- Specific Lengths: mm, cm, inches, etc.. See the article from Mozilla for more.
- Percentages

Z-axis values: The z axis value is different. This one only has a length that is specified.[2]

- The z axis is a little different. It can only be set with a length.
- Also, this value can be left off if you don't need to work in 3D.[2]

> *The z axis value is different. This one only has a length that is specified and that value can be left off if you are working in 3D.*

It's time to work with CSS! Here is our first exercise.

Exercise 2.5.1

1. Set the transform-origin with the following values: x-axis = top, y-axis = right.
2. Set the transform-origin with the following values: x-axis = 20%, y-axis = 60%.
3. Set the transform-origin with the following values: x-axis = 1cm, y-axis = 2cm

Answer

1. transform-origin: top left;
2. transform-origin: 20% 60%;
3. transform-origin: 1cm 2cm;

Footnotes

1. “Transform-Origin - CSS: Cascading Style Sheets: MDN.” CSS: Cascading Style Sheets | MDN, MDN Web Docs, https://developer.mozilla.org/en-US/...ansform-origin.

2. "CSS Transform-Origin Property." W3 Schools, https://www.w3schools.com/csSref/css...orm-origin.php.

3: CSS Transitions and Eases - Creating Motion on the HTML Page

Introduction to Transitions and Eases

Everything up until this point has been static animation; animation that jumps from one point to another point without a smooth transition between the two stopping points. In this chapter, we'll take a look at how to create transitions with keyframes that will animate properties over time on our HTML pages. Then, we'll talk about using one of the finer points of animation techniques: easing. Let's go!

3.1: CSS Transitions - Creating Motion on the HTML Page

CSS Transitions - Creating Motion on the HTML Page

Ahhhh, transitions! This is where we get into seeing some real motion on our HTML pages by animating over time. Transitions and keyframes, which come a little later, really start making the animations come alive.

> **Transition property**
>
> ***Transition*** *properties allow elements to change values over a specified duration, animating the property changes, rather than having them occur immediately.*[1]

The syntax for transitions is as follows:[1]

> *transition: property | duration | timing-function | delay;*

Let's look at all the values to the right of transition to see what they mean.

Property

The **property** value of a transition is the name of the CSS property that you want to animate over time. This would be something like opacity, color, or 'all.' Here is a complete list of properties that can be animated.

> **Definition: Property**
>
> The **property** value of a transition is the name of the CSS property that you want to animate over time.[1]

Mozilla keeps an updated list of all of the possible animatable CSS properties. Be sure to go check out their page.

Duration

This is how long you want the animation to take to finish. This is typically denoted in seconds (i.e., 2s).[1]

> **Definition: Duration**
>
> How long it takes for the animation to finish.

Timing-function

The timing function has to do with making an animation look more organic...more like what you would see in nature. In animation, we use ideas from how real-world objects act.

A rock that rolls down a hill takes a little time to pick up speed...until it comes to a dead stop after hitting a tree. So, you could say that the motion of the rock eases in at the beginning (starts slowly), goes fast during the central part of its motion, and then stops suddenly.

> *In traditional animations,* ***ease*** *defines the starting and stopping motion of an animation.*

Eases need a bit more explanation because there is a lot to them. Let's look at these items next.

> **Definition: Timing-function**
>
> The timing-function has to do with the style of the animation. This is where easing comes into play.

Delay

The amount of time that the animation should wait before it starts to move.

Definition: Delay

The delay is the pause that the animation has before it begins.

Footnotes

[1] "Using CSS Transitions - CSS: Cascading Style Sheets: MDN." *CSS: Cascading Style Sheets | MDN*, MDN Web Docs, https://developer.mozilla.org/en-US/...SS_transitions.

3.2: Eases - Adding Character and Style to Motion

Eases

In traditional animations, **ease** defines an animation's starting and stopping motion. If an animation starts slowly and then speeds up, that is called **ease in**. If an animation decelerates toward the end of its motion, that animation is said to **ease out**. Eases are what gives an animation more character and style.

Definition: Ease

The character with which an animation moves.

Definition: Ease-in

An animation that starts slowly and then speeds up.

Definition: Ease-out

An animation that slows down at the end of its motion.

The nice thing about the following video is that it compares no easing at all to other easing types found in Adobe Animate, which we will be using a bit later in the semester. Don't get too wrapped up in the technical details just yet...the takeaway here is to compare no easing to some of the other easing types.

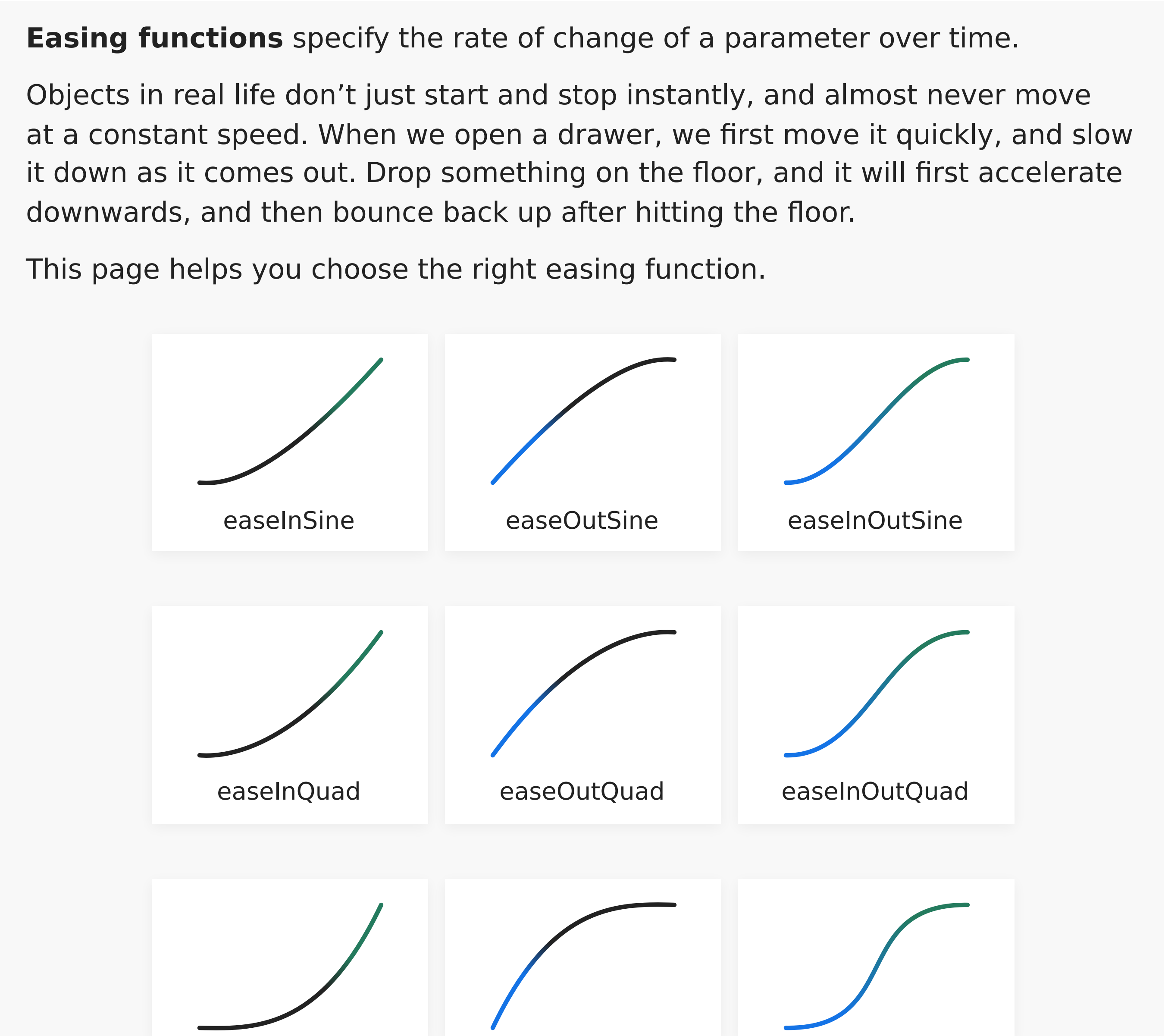
Please watch the first four minutes or so to get a feel for what eases do.

Let's explore the keywords that are associated with the timing-function in the Easing Sampler below. If you have trouble viewing this, feel free to see the original one on my website. All of the coding for the eases below came from the "Easing-function" article on the MDN Web docs site. Play around with the animations below and then go read the article, please.

The very last timing function in the example above was the **cubic-bezier**. The cubic-bezier function provides the most flexibility and customization in the motion of an animation. There are four numbers that can be tweaked in setting this up.[2] Take a look at the article from W3Schools on the CSS cubic-bezier() Function.

> The **cubic-bezier timing function** provides the most flexibility and customization in the motion of an animation.

Now that you know how the cubic-bezier function works, there are tons of resources where you can see your animations in action. I'm going to share two of these today: Easings.net and Cubic-bezier.com. Take a look at this page from Easings.net. I like the way they visualize the cubic bezier curves.



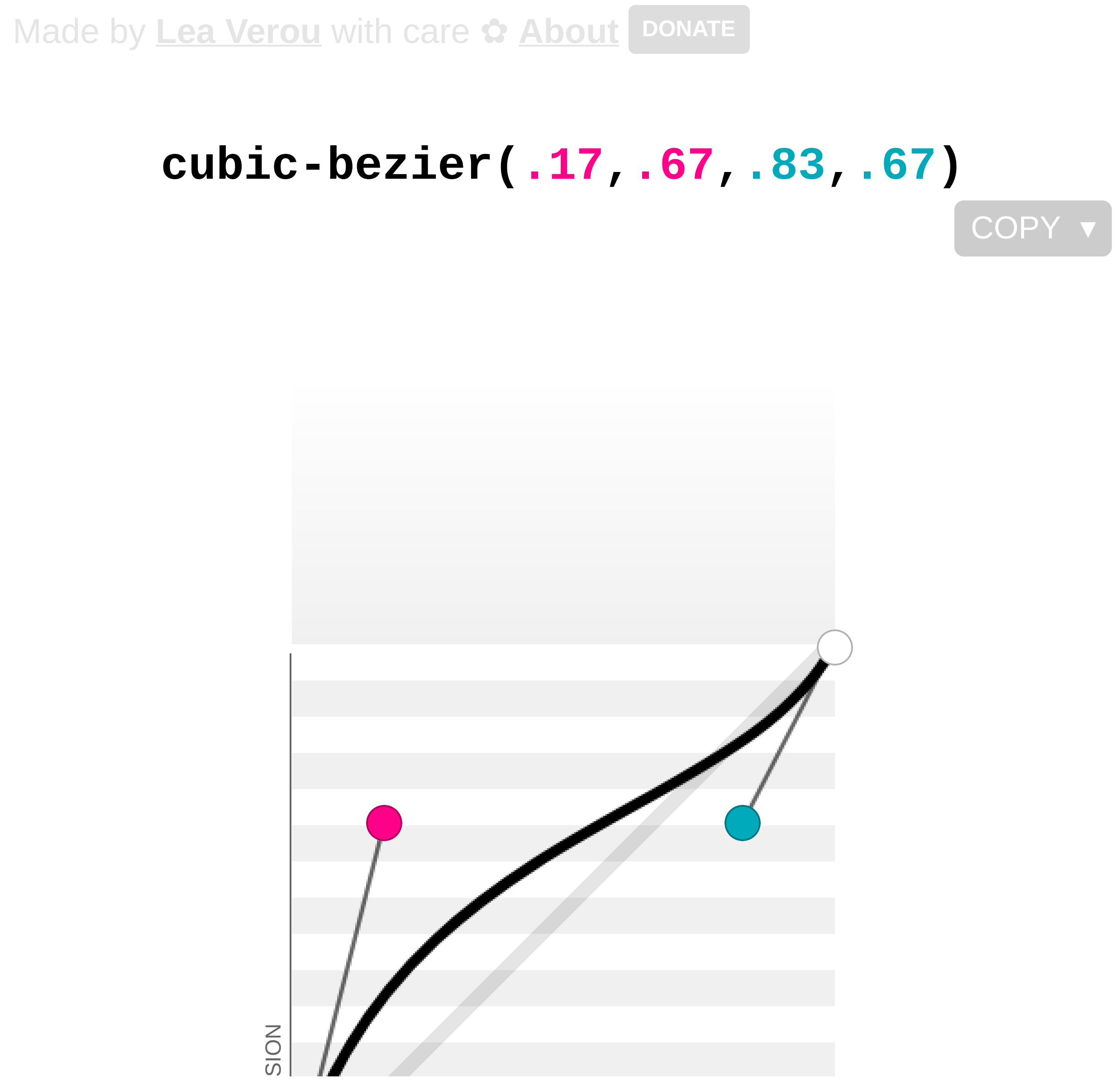
Easing functions specify the rate of change of a parameter over time.

Objects in real life don’t just start and stop instantly, and almost never move at a constant speed. When we open a drawer, we first move it quickly, and slow it down as it comes out. Drop something on the floor, and it will first accelerate downwards, and then bounce back up after hitting the floor.

This page helps you choose the right easing function.

Warning - Do not use their terms (like easeInQuad) as a keyword in your CSS files. These terms are descriptive only. However, they do have a post where you can copy the correct CSS code, so be sure to look for that instead.

I'd like to show you the final resource for bezier curves: cubic-bezier.com. The site allows you to drag the handles on the bezier curve much like you would do in the pen tool in Illustrator or Photoshop. Use it to tweak your easings.


Made by Lea Verou with care ✿ About

```
cubic-bezier(.17,.67,.83,.67)
```

There are a couple of timing functions that I did not cover today: Step in and Step out. If you'd like to see information on those, check out the w3schools article on CSS transition-timing functions.

? Exercise 3.2.1

We're going to do the exercise for this section a little differently. Open up the video below in a different browser window and resize it so that you can see both the video and the codepen below it simultaneously. Then, follow along with my demonstration by watching the video and typing along with me.

Footnotes

1. “ CSS Eases.” *CSS: Cascading Style Sheets | MDN*, MDN Web Docs, https://developer.mozilla.org/en-US/...asing-function.

2. *CSS Cubic-Bezier() Function*, W3 Schools, https://www.w3schools.com/cssref/fun...bic-bezier.php.

4: CSS Keyframe Animations - Animating at Points in Time

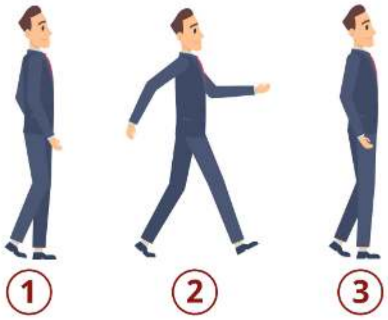
Keyframes

Back in the Golden Age of animation, head animators would lead a team of junior animators as they worked on a new cartoon or animated feature film. The head animators would create **keyframes**. Keyframes are the critical points or poses in an animation.

Definition: Keyframes

Keyframes are the critical points or poses in an animation.

If a head animator drew keyframes for a man's walking pose, they would probably draw him standing still, in full stride, and at rest. You can see what this might look like in the below. I have added numbers below the poses drawn by the original artist because they will come in handy later.



Example of Keyframes

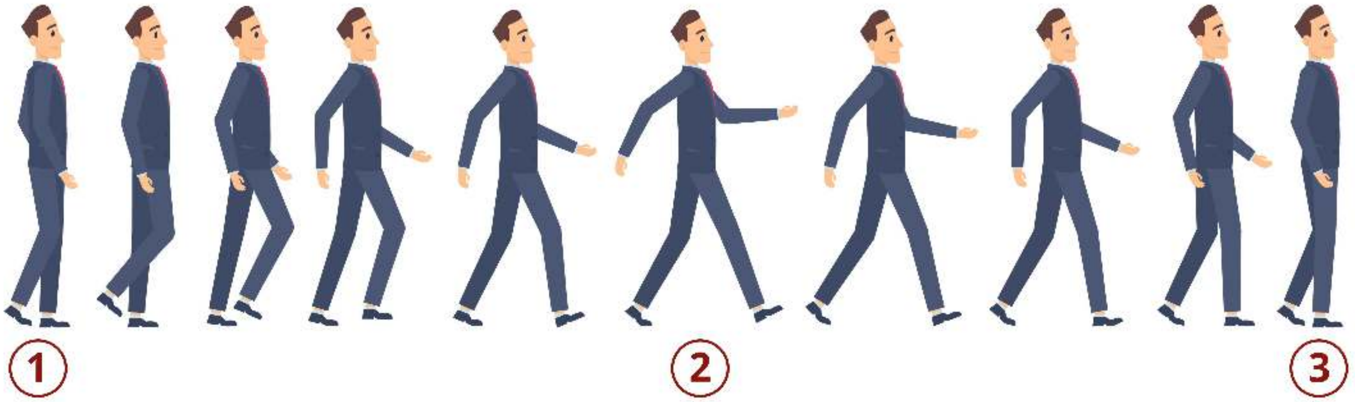
Image credit: Valerii Egorov from Alamy Stock Vector

The head animator would then pass those keyframes off to the junior animators. The junior people would create the frames that were in between the keyframes. This process is known as **tweening**.

Definition: Tweening

The creation of frames between important poses.

In the image below, you can see the original keyframes that are marked with the numbers. However, you can also see all of the new 'tween' poses between each keyframe to make the animation sequence smooth.



Example of Tweening

Image credit: Valerii Egorov from Alamy Stock Vector

In digital animations, we still use those exact words for the process. However, instead of a team of junior animators that create the 'tweens,' we now have digital applications that can help us with that.

In CSS animations, keyframes still have much of that idea from the Golden Age of animation. We have specific points at which we want things to happen. The syntax is as follows:[1]

Syntax 1

```
@keyframes myName {

keyframes-selector {css-styles;}

}
```

Let's talk about what the syntax means.[1]

- @keyframes - This is the required rule name.
- myName - This is something that is totally made up by you. Make it something that you will remember easily.
- keyframes-selector - This will define the time when you want the action to happen. It is specified in percentages or in words like "from" and "to."[1] You must have at least one of these.
- css-styles - The CSS style properties and values that you want to animate. Again, you must have at least one of these.

Below is what one keyframe rule might look like. The 'moveDown' text was the name that I made up for the example. 0% identifies the starting position. 50% identifies the mid-way point. 100% shows what should happen at the end of the animation. Best practice dictates that you must have at least the 0% and the 100% positions defined.

There is a second part that you must have to bind it to the element you want to animate. It looks like what you see below. Notice the 'moveDown' name is present. This name must match whatever you make up in the @keyframes rule exactly.

Syntax 2

div { position: relative; animation: **moveDown** 5s infinite; }

Here is what this means:[1]

- **div** - The name of the element you want to animate.
- **position: relative**; - The position property must be defined to get the animation to work properly if you need to move it across the stage in some fashion. The property's value doesn't matter as much as just having it on the page.
- **animation** - This is where you tie the keyframes rule to the specific element being animated, set how long the overall animation should take, and how many times the animation should loop (aka - animation iteration count). You can set it to a specific number or have it loop infinitely.

Exercise 4.1

1. In between the curly braces for the div, add the following:
   1. position: fixed;
   2. animation: diagonalMove 5s infinite;
2. In the @keyframes rule, do the following steps
   1. Set the name of the animation to diagonalMove. (Check the spelling carefully!)
   2. Create a keyframe at the beginning (0%). Add values so that the square starts at the top left corner, has a yellow background color, and cannot be seen.
   3. Add a second keyframe midway through the animation (50%). Add values so that the square is now 100px from the top, 100px from the left, and has a red background color.
   4. Add a final keyframe at the end of the animation (100%). Add values so that the square returns to the top left corner, has a blue background color, and can be fully seen on the page.

Answer

1. div { position: fixed; animation: diagonalMove 5s infinite; }
2. @keyframes diagonalMove {
   0% {top: 0px; left: 0px; background: yellow; opacity: 0;}
   50% {top: 100px; left: 100px; background: red;}
   100% {top: 0px; left: 0px; background: blue; opacity: 1;}
   }

Footnotes

[1] “CSS @Keyframes Rule.” *CSS @Keyframes Rule*, W3Schools, https://www.w3schools.com/cssref/css...-keyframes.php.

5: CSS3 Clip-paths and Shapes

Introduction

Today, we'll be continuing our work with CSS by looking at clip-paths. There are a couple of things that I just can't pass up on, too, even though they aren't technically animations per se. CSS Shapes can do some beautiful things with how text looks and yes, they can even be 'technically' animated. We'll talk about what that means later in this chapter.

Some of the techniques in this chapter are still being implemented by browsers and are not yet fully supported. Be sure to check out CanIUse.com to see the level of support for each technique.

5.1: Clip-path Property

"Arch" by Franz Bachinger from Pixabay

The Clip-path Property

The clip-path property reminds me of looking through a view port on a ship or a window in a house. Only a portion of the landscape is visible through the hull or the wall. The **clip-path property** in CSS allows you to specify a specific region of an element to display, with the rest being hidden (or "clipped") away.[2]

Definition: Clip-path property

The **clip-path property** in CSS allows you to specify a specific region of an element to display, with the rest being hidden (or "clipped") away.[2]

It's very common to use an image with a clip-path, but this isn't the only thing it can be used with. Try it out with some text or other inline elements.[2] For animation purposes, Clip-path can be used on any basic shape.[3] Basic shapes are inset(), circle(), ellipse(), and polygon().[1]

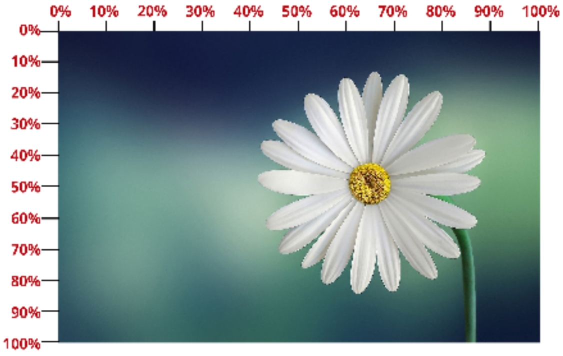
Definition: Basic Shapes

Basic shapes are inset(), circle(), ellipse(), and polygon().[1]

As we go forward through these four basic shapes, keep in mind that positions can be set with pixels or percentages.

> ***Positions** can be set with pixels or percentages.*

In the image below, if you wanted to set a positions that is in the complete lower right hand corner, you would state that as 100% 100%. If you wanted something in the middle of the bottom edge of the image, that would be 50% 100%. The first number is the X value and the second number is the Y value.



Example of Using Percentages to Set Positions

Values for the Clip-path Property

Let's look now at the different values that you can set on the clip-path property: inset(), circle(), ellipse(), and polygon().

> The values that you can set on clip-path are inset(), circle(), ellipse(), and polygon().

Inset()

Inset() will define a **rectangle-shaped vector path**. [1] The code would look like this:

> 📌 Inset() Syntax
>
> *clip-path: inset(50px 100px);*

The first number defines the rectangle's upper left corner in pixels, and the second one defines the lower right corner. Keep in mind that you could use percentages here instead of pixels, too. Play around with the numbers in the CodePen below and see how the rectangle changes.

Circle()

Circle() defines a **circle-shaped vector path.** [1] Here is what that code will look like:

> 📌 Circle() Syntax
>
> clip-path: circle(30% at 50% 50%);

The first number defines the radius, while the last two numbers show where the circle should be located. [1]

Ellipse()

Ellipse() functions much like the circle(). The one thing that sets this shape apart is that **you define two radii** rather than just one. [1]

Ellipse() Syntax

```
clip-path: ellipse(130px 140px at 10% 20%);
```

Polygon()

This defines a polygon "using an SVG filling rule and a set of vertices." [1] You can have as many vertices (points) as you want! Just keep defining them in a clockwise motion around the graphic.

Polygon() Syntax

clip-path: polygon(50% 0, 100% 50%, 50% 100%, 0 50%);

The last thing to do for clip-paths is to head over to Clippy and play! Try several different clip-paths on their backgrounds. You'll have fun!

Bibliography

1. “Clip-Path - CSS: Cascading Style Sheets: MDN.” *Cascading Style Sheets | MDN*, MDN Web Docs, https://developer.mozilla.org/en-US/.../CSS/clip-path.

2. Cope, Sarah. “Clip-Path: CSS-Tricks.” *CSS Tricks*, CSS Tricks, 27 Nov. 2020, https://css-tricks.com/almanac/prope...s/c/clip-path/.

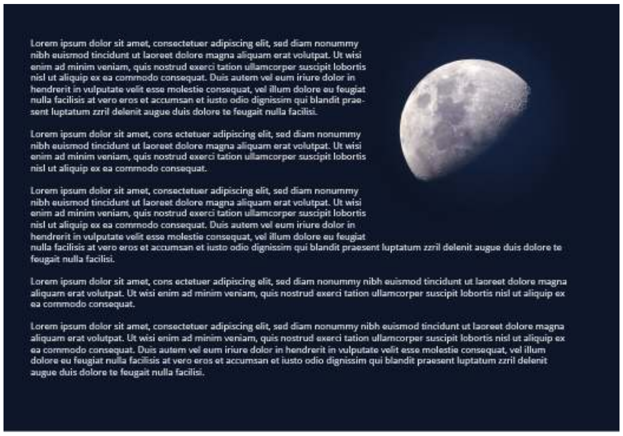
3. *CSS Clip-Path Property*, W3Schools.Com, https://www.w3schools.com/cssref/css3_pr_clip-path.asp .

Footnotes

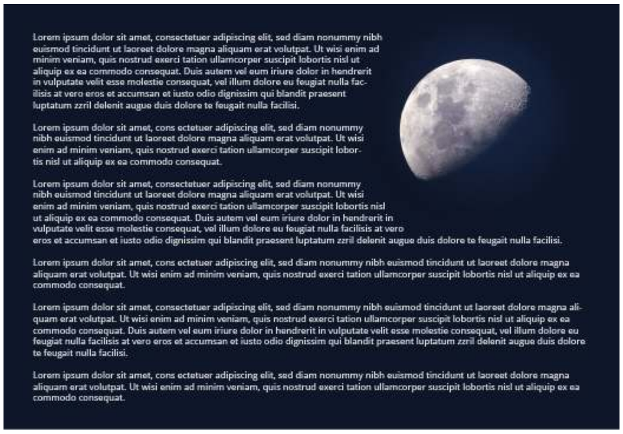
5.2 CSS Shapes Introduction

CSS Shapes Introduction

Whenever we create an object in HTML, by default, that object is some kind of rectangle. The image below shows this in detail. Notice how the text wraps itself around the Moon in a square shape.



For years, printed media could curved text around a round image or graphic like the second photo you see. See how much more elegant this is?



Lucky for us, Web development has finally caught up with the printed graphics technique in the past few years by implementing **CSS3 Shapes**. All major browsers now support this feature.[1] **Click the HTML page below and then hover over the flower** for an idea of how this could be useful. See how the text accentuates the motion of the flower?

On the next page, we'll discuss the syntax that will make this happen.

Footnotes

[1] “‘CSS Shapes’: Can I Use... Support Tables for HTML5, CSS3, Etc.” *"CSS Shapes" | Can I Use... Support Tables for HTML5, CSS3, Etc*, CanIUse.com, https://caniuse.com/?search=css+shapes.

5.3: Shape-outside

Shape-outside

There are several properties that can be set as a part of CSS Shapes 1, like shape-image-threshold and shape-margin , however, we'll just be using shape-outside in the interest of time. Feel free to peruse the links above for more information on those topics.

"The **shape-outside** CSS property defines a shape—which may be non-rectangular—around which adjacent inline content should wrap. By default, inline content wraps around its margin box; shape-outside provides a way to customize this wrapping, making it possible to wrap text around complex objects rather than simple boxes." [1]

Definition: Shape-outside

> The **shape-outside** CSS property defines a shape—which may be non-rectangular—around which adjacent inline content should wrap.

So, basically, shape-outside will wrap text around the outside of a shape. Four shapes can be called upon with the shape-outside property: circle(), ellipse(), inset(), polygon().[1]

> Four shapes can be called upon with the shape-outside property: circle(), ellipse(), inset(), polygon().[1]

So, using our syntax from above, it would look something like this in practice. Here is the syntax:

Shape-outside Syntax

> shape-outside: property-name(values);

To get the shape to work correctly, you'll also need to implement a float on the element and the element must have a width and a height set on it. Keep that in mind!

Remember the example of the spinning flower from Chapter 5.2: Introduction to CSS Shapes? Take a moment and download the code for that demonstration, please. Unfortunately, this platform is a bit wonky when it comes to directly downloading a zip file, so, you'll have to do this on your own. Here is the link. Copy and paste it into your browser.

http://www.rosemarybarker.net/web_23...hapes_demo.zip

In the text editor of your choice, open up both the index.html and master.css files from my CSS Shapes Code Demo above. Look through both the HTML and CSS files. In the HTML file on line 10, notice that I've given the flower image a class of "myImage."

Now, switch over to the master.css file. Notice this code on starting on line 5.

```
.myImage {
width: 200px; /*required*/
height: auto; /*required*/
float: left; /*required*/
shape-outside: circle(55%);
margin: 20px;
}
```

? Exercise 5.3.1

Please read the following article and follow along with the instructor.

https://tympanus.net/codrops/2018/11/29/an-introduction-to-css-shapes/

The final thing to do is to watch the following video on today's topic:

Footnotes

1. “Shape-Outside.” *Cascading Style Sheets* | *MDN*, MDN Web Docs, https://developer.mozilla.org/en-US/.../shape-outside.

6: CSS Variables and SVG Graphics

Introduction to CSS Variables and SVG Graphics

Let's take a quick peek at one of the new-ish techniques in CSS, the CSS Variable, and explore the animation of SVG graphics. Let's go!

6.1 CSS Variables

CSS Variables

CSS Variables is a new-er "kid on the block" but you should know about it and implement it because it has good support on the major browsers.[1] While this isn't technically an animation technique, you can leverage it if you have many different animations and want to be able to update them in one convenient place. Check it out!

Footnotes

[1] "'CSS Variables': Can I Use... Support Tables for HTML5, CSS3, Etc." *"CSS Variables" | Can I Use... Support Tables for HTML5, CSS3, Etc*, CanIUse.com, https://caniuse.com/?search=css+variables.

6.2 Working with SVGs

Working with SVGs

This video will help you understand why SVGs are so important for today's Web environments.

6.3 SVG Mask Animations

SVG Mask Animations

Now, it's time to put that new-found SVG knowledge to work while you use an SVG mask! Enjoy!

7: Tools of the Trade - Exporting and Optimizing SVGs and Using the Greensock Library for Animations

Tools of the Trade - Exporting and Optimizing SVGs and Using the Greensock Library for Animations

This is the week to pull out all the stops on the final project in the CSS Animations series. In an in-person classroom, today's lecture would consist of a small lecture and lots of time to work on your projects in a lab. Although we aren't together in a physical classroom, I did want you to have the same opportunity to have plenty of time to work on your projects. In light of this, today's lecture will be brief and no homework will be assigned.

Let's look at a few techniques that will help you with your projects. First, we'll explore some tips for exporting SVGs from Illustrator. Then, we'll hit a simple introduction to using the Greensock Animation Library. Let's go!

7.1 Exporting SVGs from Illustrator

Exporting SVGs from Illustrator

Please watch the following video for some great in-depth tips on exporting SVGs from Illustrator.

Styling: Internal CSS
Font: Convert To Outlines
Images: Preserve
Object IDs: Layer Names
Decimal: 1
□ Minify ☑ Responsive

7.2 Optimizing SVG Files

Optimizing SVG Files

Any time you work with graphics on the Web, you need those files to be as small as possible so that the site is performant. A great little Web app to help optimize SVG files is SVG OMG . I'll walk you through how to use this in the video below.

7.3 Using the GSAP (aka Greensock) Animation Library

Using the GSAP (aka Greensock) Animation Library

The last item on today's agenda is getting a look at how to use the GSAP (aka Greensock) Animation Library. Greensock is based on JavaScript and it works really well witih Adobe Animate.[1] It has over 8 million users, too![1]

Watch the video below to get started with learning about Greensock. When you start your projects, you'll want to have the link to the CDN .

Footnotes

[1] Priya, C.S. “10+ Best Javascript Animation Libraries to Use in 2023.” *CodeinWP*, VertiStudio, 1 Jan. 2023, https://www.codeinwp.com/blog/best-j...ibraries/#gref.

Index

Detailed Licensing

Overview

Title: Animation for the Web: A Guide to Advanced CSS and Animation Techniques (Barker)

Webpages: 38

Applicable Restrictions: Noncommercial

All licenses found:

- CC BY-NC 4.0: 92.1% (35 pages)
- Undeclared: 7.9% (3 pages)

By Page

- Animation for the Web: A Guide to Advanced CSS and Animation Techniques (Barker) - *CC BY-NC 4.0*
  - Front Matter - *CC BY-NC 4.0*
    - Title Page - *CC BY-NC 4.0*
    - TitlePage - *Undeclared*
    - InfoPage - *CC BY-NC 4.0*
    - Table of Contents - *Undeclared*
    - Licensing - *CC BY-NC 4.0*
    - Dedication - *CC BY-NC 4.0*
  - 1: An Introduction to Animating with CSS - *CC BY-NC 4.0*
    - 1.1: Introduction to CSS Animation - *CC BY-NC 4.0*
    - 1.2: Becoming Familiar with CodePen - *CC BY-NC 4.0*
    - 1.3: Working with Cartesian Coordinates - *CC BY-NC 4.0*
  - 2: Understanding 2D CSS Transforms - *CC BY-NC 4.0*
    - 2.1: 2D CSS Transforms - rotate() - *CC BY-NC 4.0*
    - 2.2: 2D CSS Transforms - scale() - *CC BY-NC 4.0*
    - 2.3: 2D CSS Transforms - skew() - *CC BY-NC 4.0*
    - 2.4: 2D CSS Transforms - translate() - *CC BY-NC 4.0*
    - 2.5: Transform-origin Property - *CC BY-NC 4.0*
  - 3: CSS Transitions and Eases - Creating Motion on the HTML Page - *CC BY-NC 4.0*
    - 3.1: CSS Transitions - Creating Motion on the HTML Page - *CC BY-NC 4.0*
    - 3.2: Eases - Adding Character and Style to Motion - *CC BY-NC 4.0*
  - 4: CSS Keyframe Animations - Animating at Points in Time - *CC BY-NC 4.0*
  - 5: CSS3 Clip-paths and Shapes - *CC BY-NC 4.0*
    - 5.1: Clip-path Property - *CC BY-NC 4.0*
    - 5.2 CSS Shapes Introduction - *CC BY-NC 4.0*
    - 5.3: Shape-outside - *CC BY-NC 4.0*
  - 6: CSS Variables and SVG Graphics - *CC BY-NC 4.0*
    - 6.1 CSS Variables - *CC BY-NC 4.0*
    - 6.2 Working with SVGs - *CC BY-NC 4.0*
    - 6.3 SVG Mask Animations - *CC BY-NC 4.0*
  - 7: Tools of the Trade - Exporting and Optimizing SVGs and Using the Greensock Library for Animations - *CC BY-NC 4.0*
    - 7.1 Exporting SVGs from Illustrator - *CC BY-NC 4.0*
    - 7.2 Optimizing SVG Files - *CC BY-NC 4.0*
    - 7.3 Using the GSAP (aka Greensock) Animation Library - *CC BY-NC 4.0*
  - Back Matter - *CC BY-NC 4.0*
    - Index - *Undeclared*
    - Glossary - *CC BY-NC 4.0*
    - Detailed Licensing - *CC BY-NC 4.0*