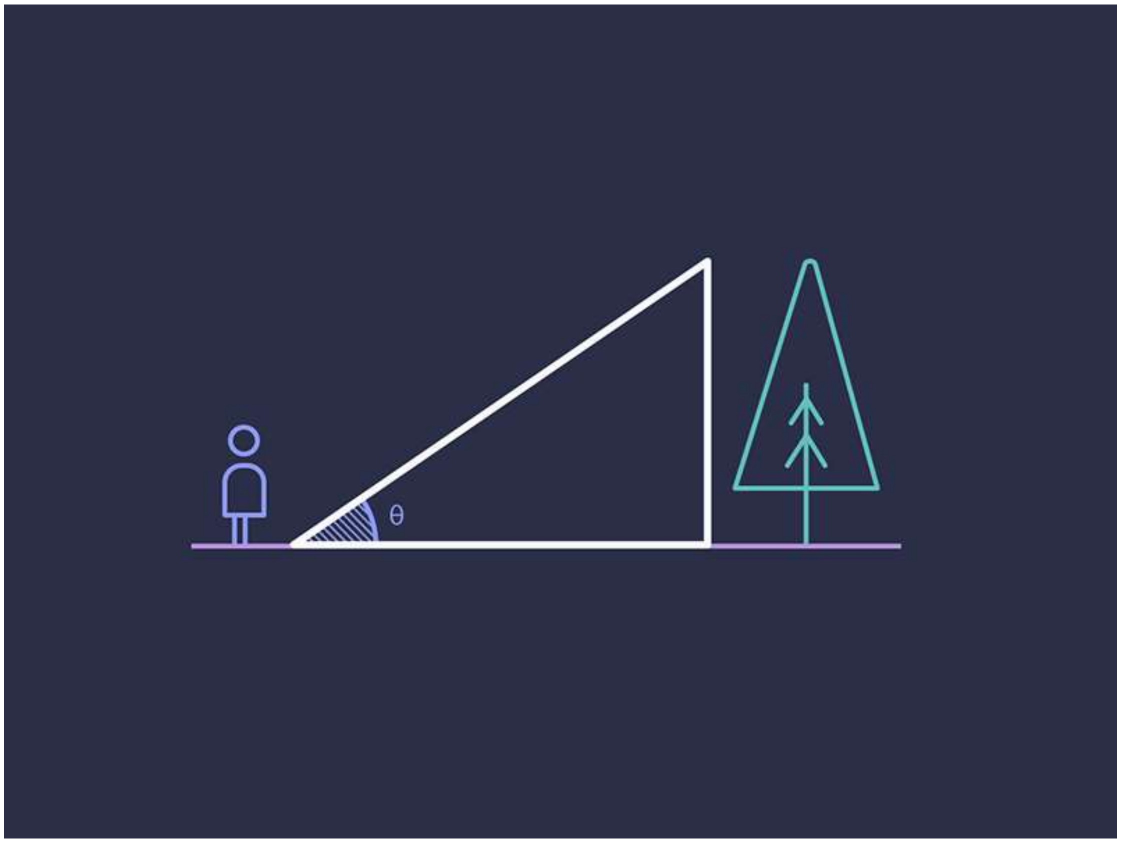
Trigonometry in CSS and JavaScript: Introduction to Trigonometry

In this series of articles we'll get an overview of trigonometry, understand how it can be useful, and delve into some creative applications in CSS and JavaScript.

By Michelle Barker in Tutorials on June 1, 2021



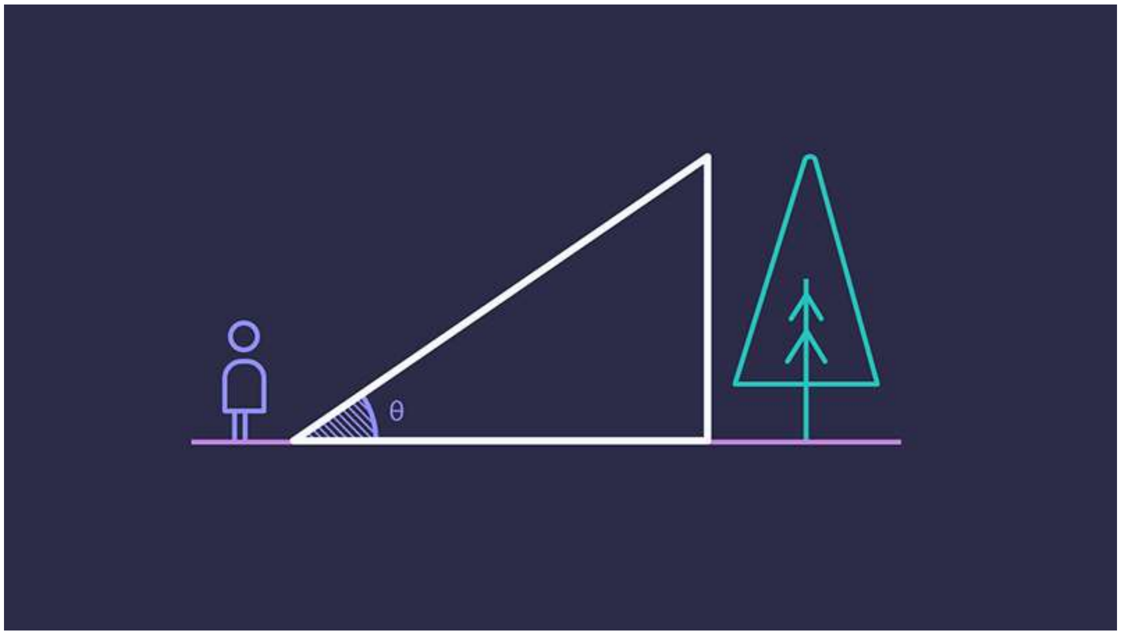
Understanding trigonometry can give us super powers when it comes to creative coding. But to the uninitiated, it can seem a little intimidating. In this 3-part series of articles we'll get an overview of trigonometry, understand how it can be useful, and delve into some creative applications in CSS and JavaScript.

1. **Introduction to Trigonometry** (this article)
2. Getting Creative with Trigonometric Functions
3. Beyond Triangles

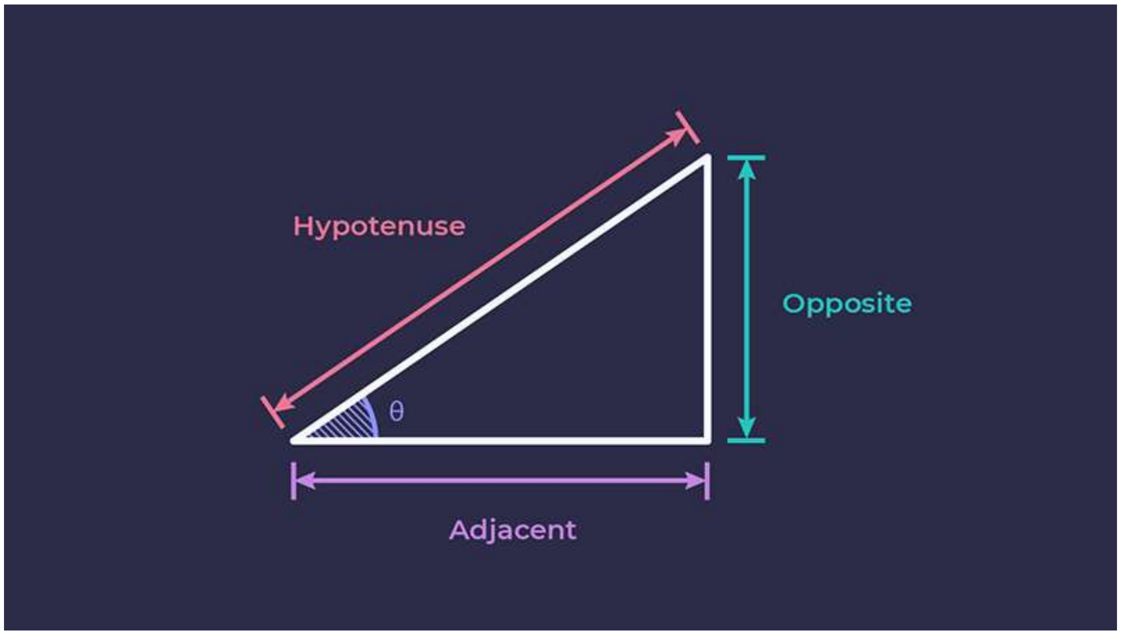
Trigonometry basics

If, like me, you've rarely used trigonometry outside of the classroom, let's take a trip back to school and get ourselves reacquainted.

Trigonometric functions allow us to calculate unknown values of a right-angled triangle from known parameters. Imagine you're standing on the ground, looking up at a tall tree. It would be very difficult to measure the height of the tree from the ground. But if we know the angle at which we look up at the top of the tree, and we know the distance from ourselves to the tree, we can infer the height of the tree itself.



If we imagine this scene as a triangle, the known length (from us to the tree) is known as the *adjacent* side, the tree is the *opposite* side (it's *opposite* the angle), and the longest side – from us to the top of the tree – is called the *hypotenuse*.

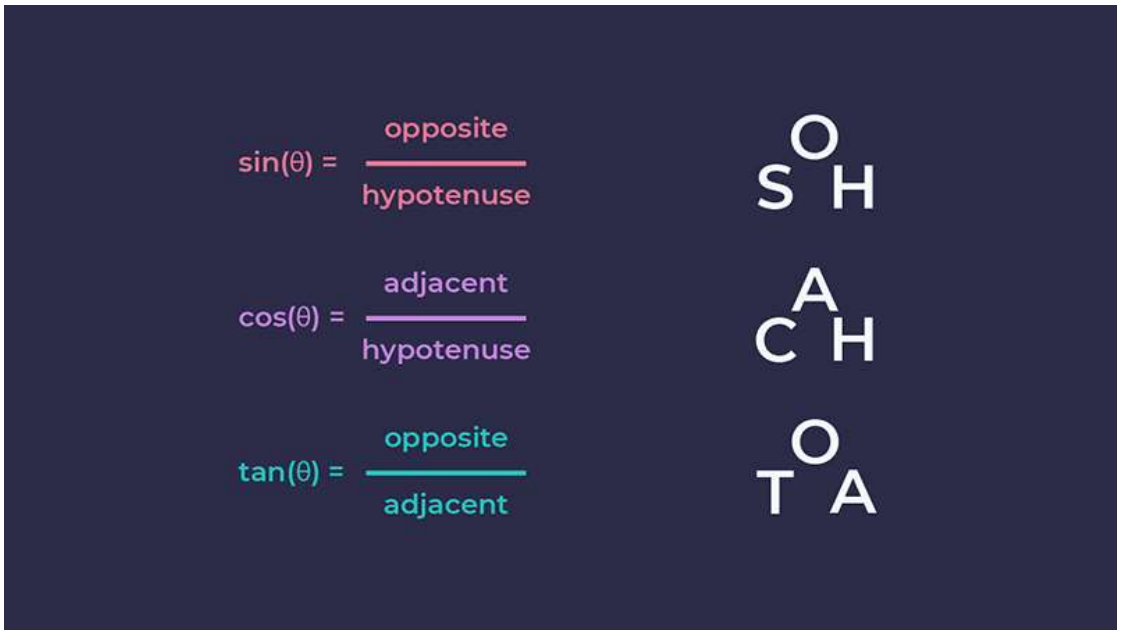


Sine, Cosine and Tangent

There are three main functions to remember in trigonometry: *Sine*, *Cosine* and *Tangent* (abbreviated to *sin*, *cos* and *tan*). They are expressed as the following formulae:

```
sin(angle) = opposite / hypotenuse
cos(angle) = adjacent / hypotenuse
tan(angle) = opposite / adjacent
```

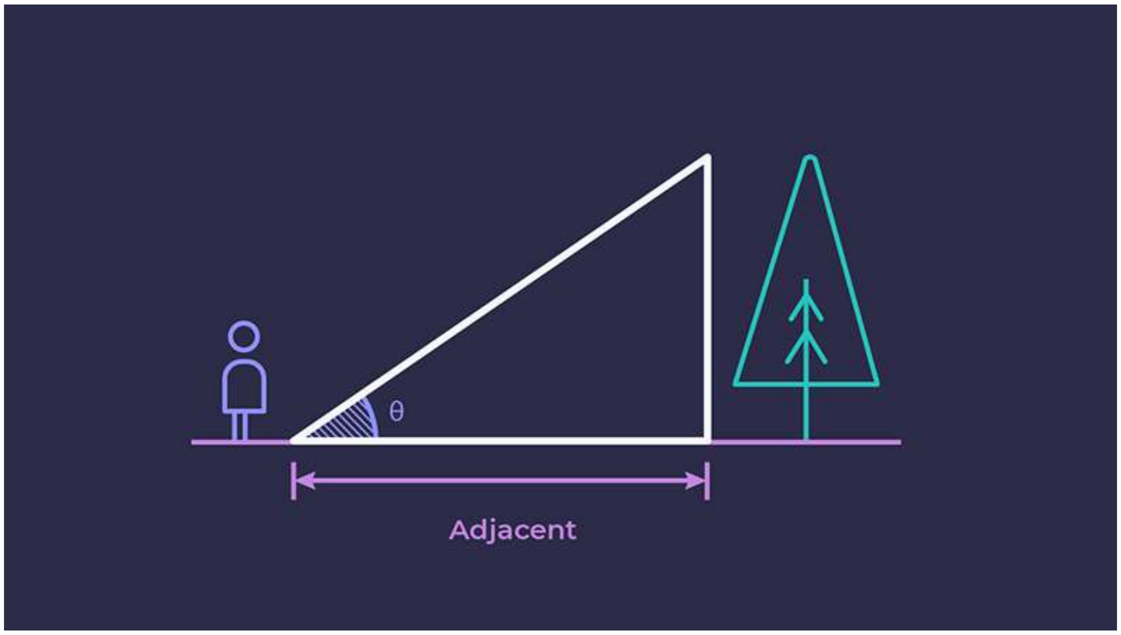
The angle is usually written as the Greek *theta* (*θ*) symbol.



$\sin(\theta) = \frac{\text{opposite}}{\text{hypotenuse}}$
$\cos(\theta) = \frac{\text{adjacent}}{\text{hypotenuse}}$
$\tan(\theta) = \frac{\text{opposite}}{\text{adjacent}}$
S O H
C A H
T O A

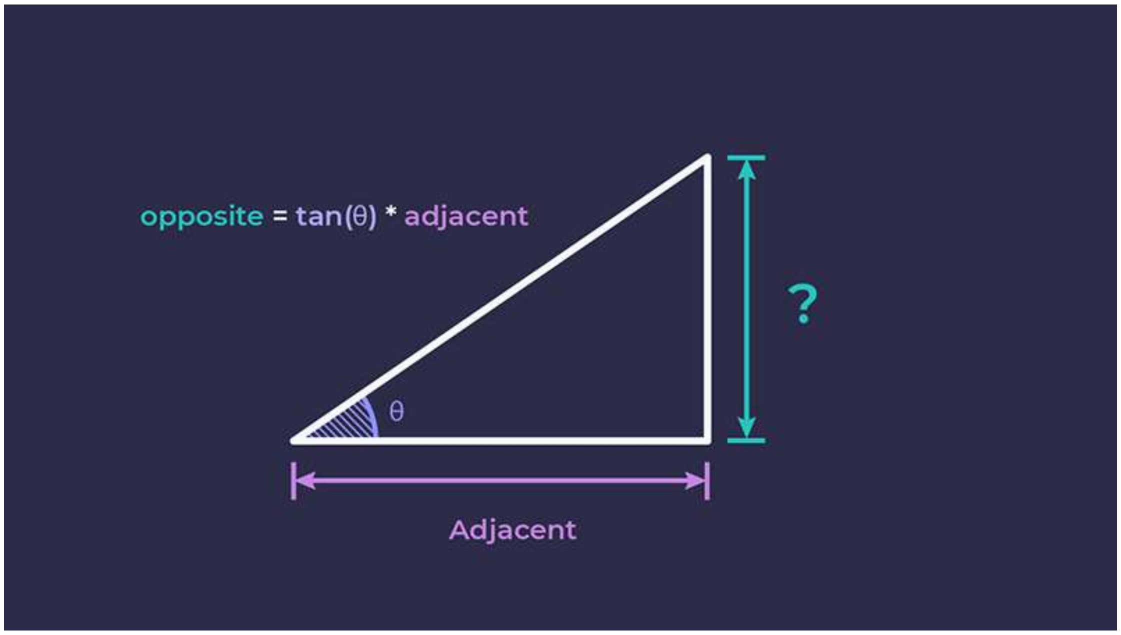
The acronym SOHCAHTOA can help us remember the formulae

We can use these equations to calculate the unknown values of our triangle from the known ones. To measure the height of the tree in the example, we know the angle (θ) and the *adjacent* side.



To calculate the *opposite* side we would need the *tangent* function. We would need to switch around the formula:

```
opposite = tan(angle) * adjacent
```



How do we get *tan(θ)*? We could use a scientific calculator (type *tan* and then the angle), or we could use code! Sass and JavaScript both include trigonometric functions, and we'll look at some ways to use these in this article and the following ones.

Sass functions

If we're working with predetermined values, we could use the trigonometric functions built into Sass (the CSS preprocessor).

To include the Math module we need the following line in our Sass file:

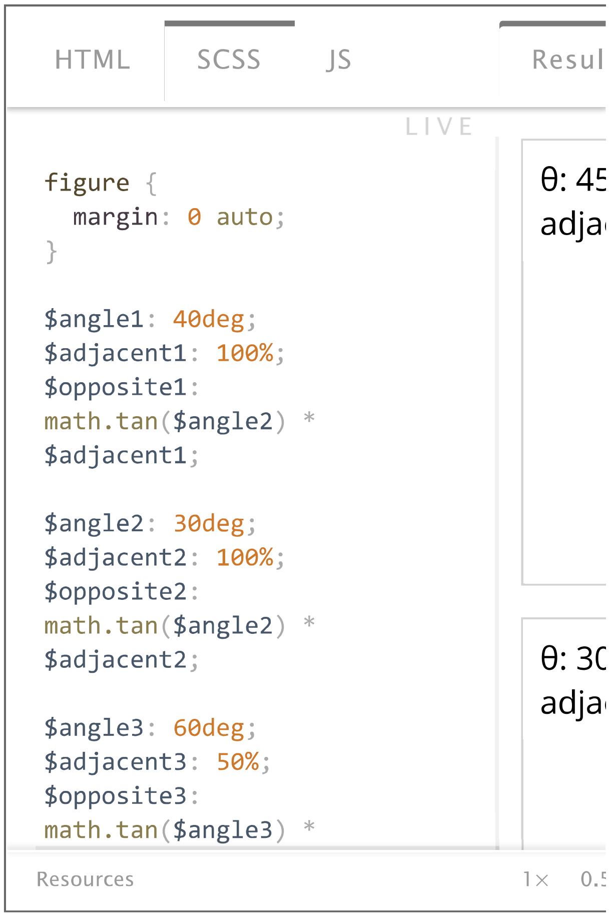
```
@use "sass:math";
```

We can use variables to calculate the *opposite* side from the angle and *adjacent* side values.

```
$angle: 45deg;
$adjacent: 100%;
$opposite: math.tan($angle) * $adjacent;
```

The *tan* function in Sass can use radians or degrees — if using degrees, the units must be specified. Without units, radians will be used by default (more on these later).

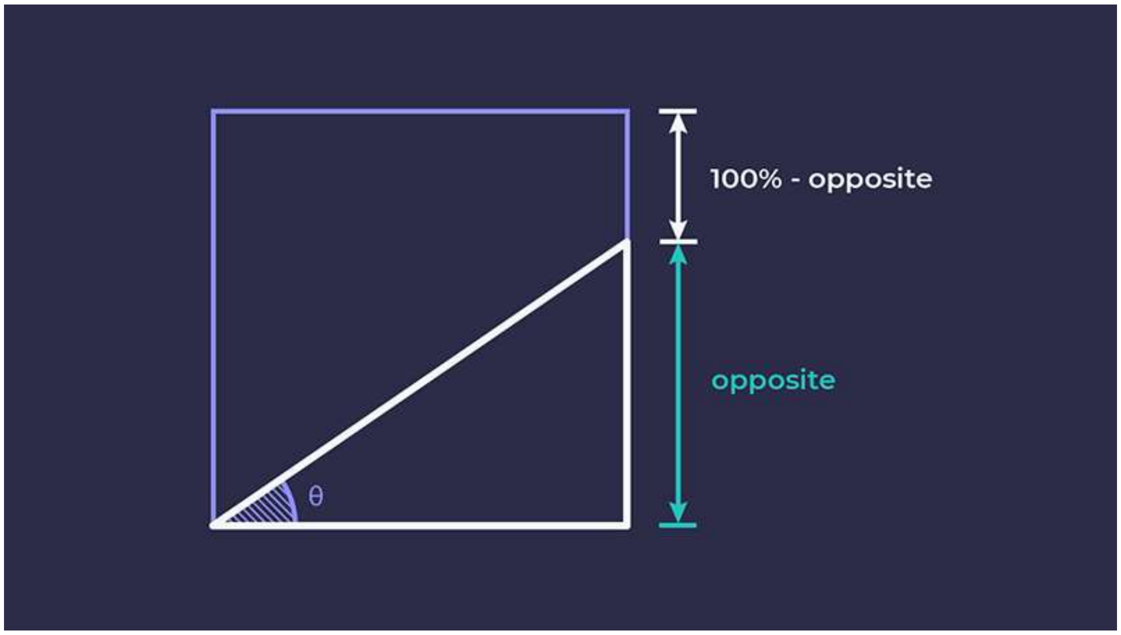
In the following demo we're using these in the `clip-path` property to determine the coordinates of the polygon points, similar to calculating the height of a tree.



```
figure {
  margin: 0 auto;
}

$angle1: 40deg;
$adjacent1: 100%;
$opposite1:
math.tan($angle2) *
$adjacent1;

$angle2: 30deg;
$adjacent2: 100%;
$opposite2:
math.tan($angle2) *
$adjacent2;

$angle3: 60deg;
$adjacent3: 50%;
$opposite3:
math.tan($angle3) *
```

We need to subtract the `$opposite` variable from the height of the element in order to get the *y* coordinate — as clip-path coordinates are plotted along the *y* axis increasing from top to bottom.

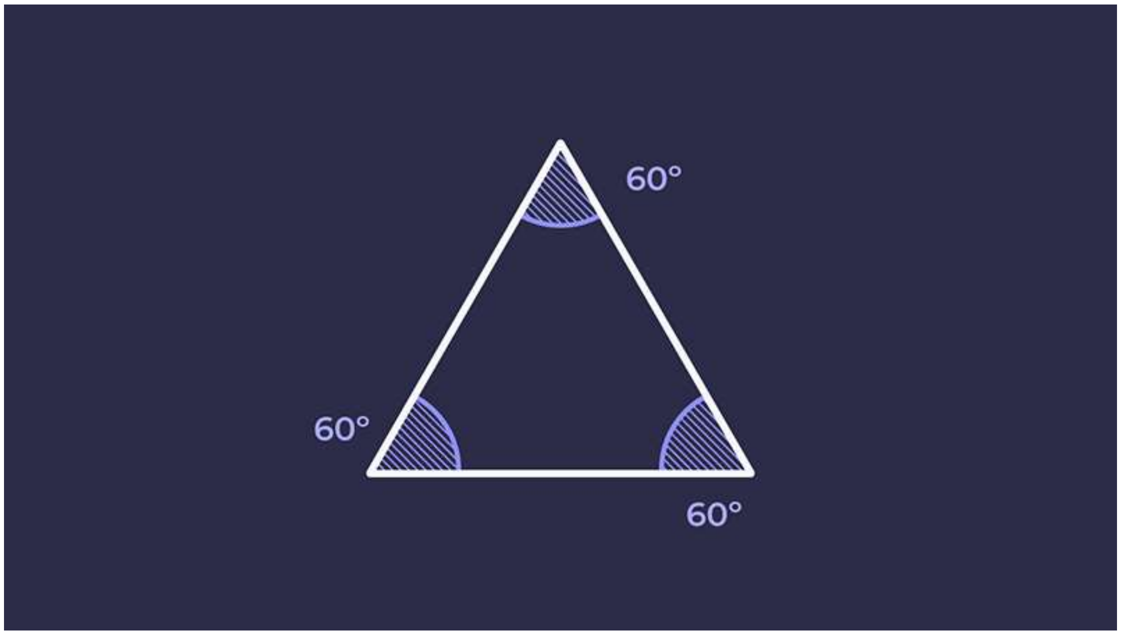
```
.element {
    clip-path: polygon(0 100%, $adjacent (100% - $opposite), $ac
}
```



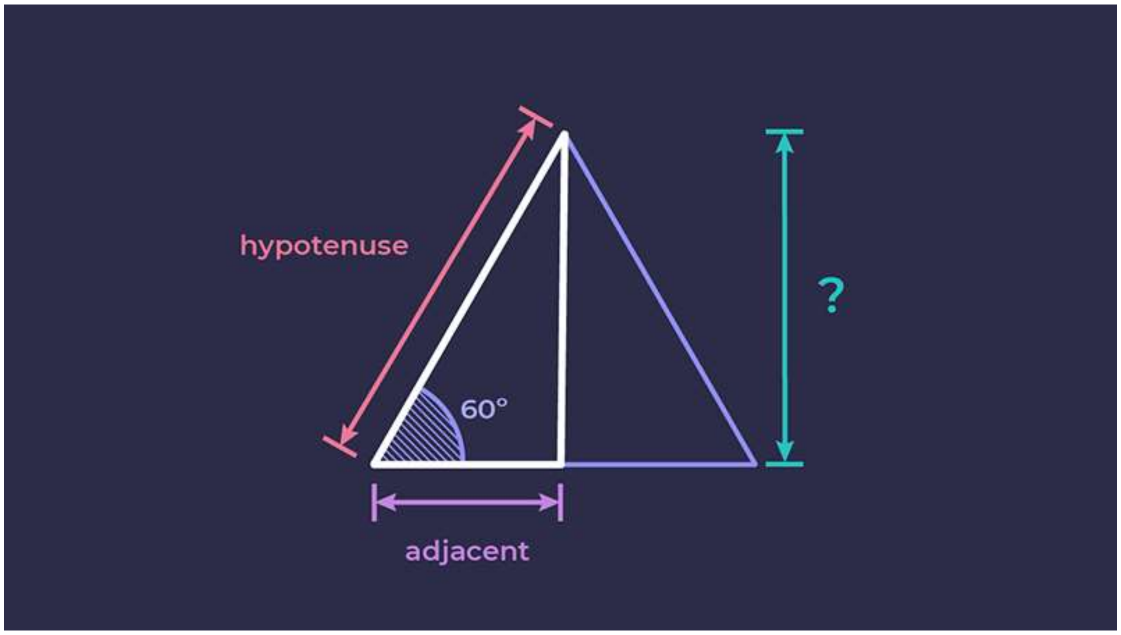
Clipping an equilateral triangle

A right-angled triangle is the simplest use of trigonometry. But we can work out the coordinates of more complex shapes by splitting them up into right-angled triangles.

An equilateral triangle is a triangle with three sides of the same length. Perhaps you remember from school that the angles in a triangle add up to 180º? That means each angle in an equilateral triangle is 60º.



If we draw a line down the middle of an equilateral triangle, we split it into (you guessed it) two right-angled triangles. So, for a triangle with sides of a given length, we know the angle (60º), the length of the *hypotenuse*, and the length of the *adjacent* side (half the length of the *hypotenuse*).

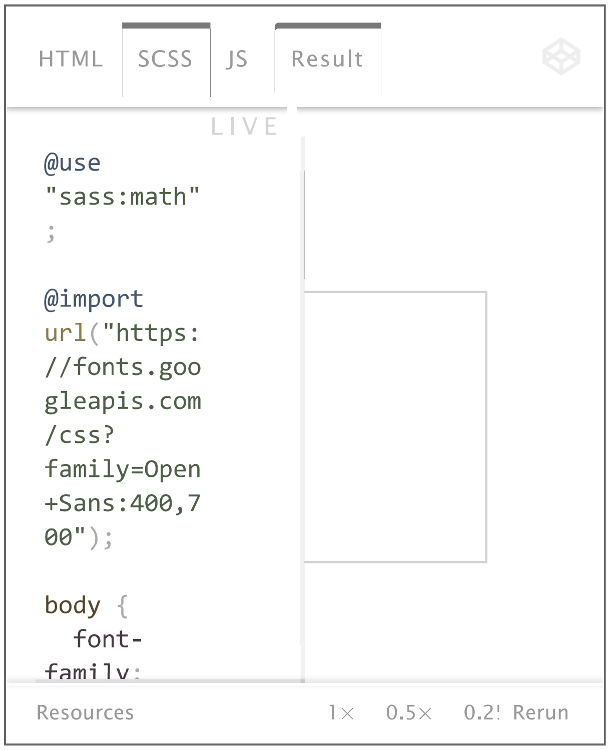


What we *don't* know is the height of the triangle — once again, the *opposite* side of the right-angled triangle. To plot the clip-path coordinates, this is what we need to work out. This time, as we know the angle and the length of the *hypotenuse*, we can use the *sine* function:

```
$hypotenuse: 60%; // side length
$angle: 60deg;
$opposite: math.sin($angle) * $hypotenuse;
```

(It would also be possible for us to use the *tangent* function instead, as we know that the length of the *adjacent* side is half of the *hypotenuse*.) Then we can use those values for our clip-path polygon points:

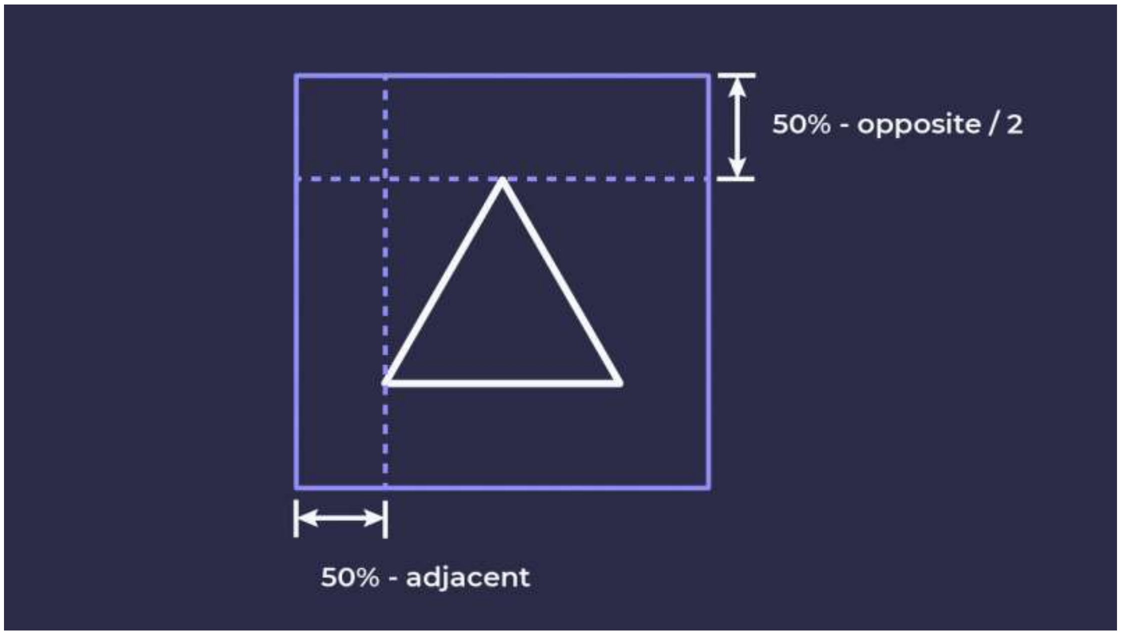
```
.element {
    clip-path: polygon(
        0 $opposite,
        ($hypotenuse / 2) 0,
        $hypotenuse $opposite
    );
}
```



```
@use
"sass:math"
;

@import
url("https:
//fonts.goo
gleapis.com
/css?
family=Open
+Sans:400,7
00");

body {
  font-
family:
```

As you can see in the demo, the element is clipped from the top left corner. This might not be completely satisfactory: it's more likely we'd want to clip from the center, especially if we're clipping an image. We can adjust our clip-path coordinates accordingly. To make this more readable, we can assign some additional variables for the *adjacent* side length (half the hypotenuse), and the start and end position of the triangle:

```
$hypotenuse: 60%; //side length
$angle: 60deg;
$opposite: math.sin($angle) * $hypotenuse;
$adjacent: $hypotenuse / 2;
$startPosX: (50% - $adjacent);
$startPosY: (50% - $opposite / 2);
$endPosX: (50% + $adjacent);
$endPosY: (50% + $opposite / 2);

.element {
    clip-path: polygon(
        $startPosX $endPosY,
        50% $startPosY,
        $endPosX $endPosY
    );
}
```



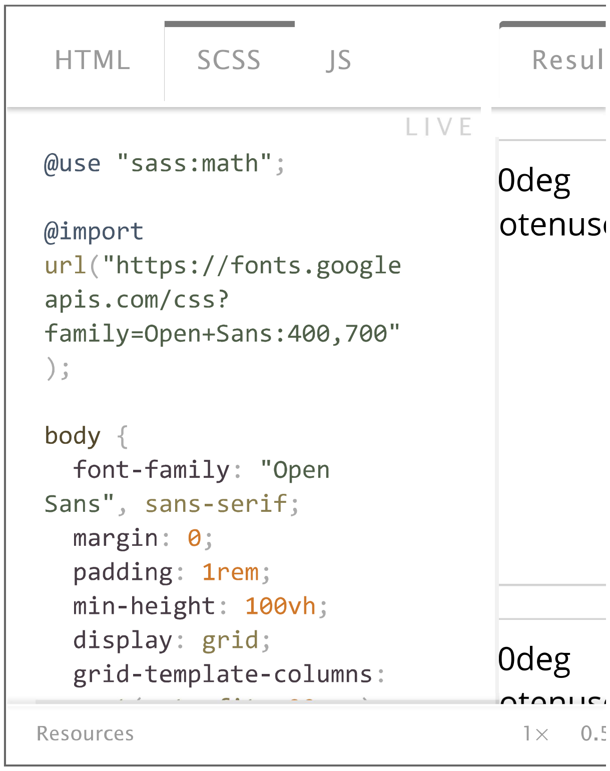
Creating a mixin for reuse

This is quite a bit of complex code to write for a single triangle. Let's create a Sass mixin, allowing us to clip a triangle of any size on any element we like. As `clip-path` still needs a prefix in some browsers, our mixin covers that too:

```
@mixin triangle($sideLength) {
    $hypotenuse: $sideLength;

    $angle: 60deg;
    $opposite: math.sin($angle) * $hypotenuse;
    $adjacent: $hypotenuse / 2;
    $startPosX: (50% - $adjacent);
    $startPosY: (50% - $opposite / 2);
    $endPosX: (50% + $adjacent);
    $endPosY: (50% + $opposite / 2);

    $clip: polygon(
                $startPosX $endPosY,
                50% $startPosY,
                $endPosX $endPosY
            );

    -webkit-clip-path: $clip;
    clip-path: $clip;
}
```

To clip a centred equilateral triangle from any element, we can simply include the mixin, passing in the length of the triangle's sides:

```
.triangle {
    @include triangle(60%);
}
```



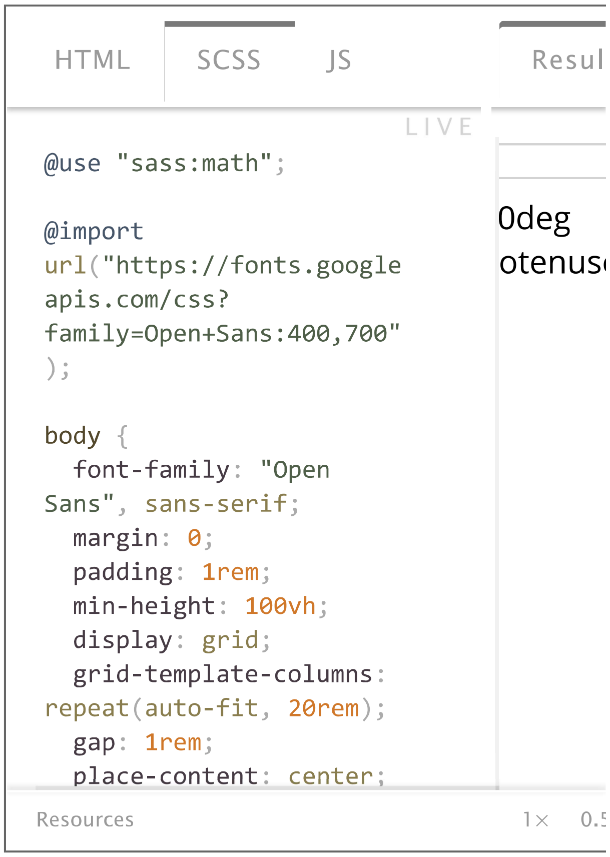
```
@use "sass:math";

@import
url("https://fonts.google
apis.com/css?
family=Open+Sans:400,700"
);

body {
  font-family: "Open
Sans", sans-serif;
  margin: 0;
  padding: 1rem;
  min-height: 100vh;
  display: grid;
  grid-template-columns:
```

Limitations of Sass functions

Our use of Sass functions has some limitations:

1. It assumes the `$sideLength` variable is known at compile time, and doesn't allow for dynamic values.
2. Sass doesn't handle mixing units all that well for our needs. In the last demo, if you switch out the percentage-based side length to a fixed length (such as rems or pixels), the code breaks.

The latter is because our calculations for the `$startPos` and `$endPos` variables (to position the clip-path centrally) depend on subtracting the side length from a percentage. Unlike in regular CSS (using *calc()*), Sass doesn't allow for that. In the final demo, I've adjusted the mixin so that it works for any valid length unit, by passing in the size of the clipped element as a parameter. We'd just need to ensure that the values for the two parameters passed in have identical units.



```
@use "sass:math";

@import
url("https://fonts.google
apis.com/css?
family=Open+Sans:400,700"
);

body {
  font-family: "Open
Sans", sans-serif;
  margin: 0;
  padding: 1rem;
  min-height: 100vh;
  display: grid;
  grid-template-columns:
repeat(auto-fit, 20rem);
  gap: 1rem;
  place-content: center;
```

CSS trigonometric functions

CSS has a proposal for trigonometric functions as part of the CSS Values and Units Module Level 4 (currently in working draft). These could be extremely useful, especially when used alongside custom properties. Here's how we could rewrite our CSS to use native CSS trigonometric functions. Changing the size of the clip path is as simple as updating a single custom property:

```css
.triangle {
    --hypotenuse: 8rem;
    --opposite: calc(sin(60deg) * var(--hypotenuse));
    --adjacent: calc(var(--hypotenuse) / 2);
    --startPosX: calc(var(--size) / 2 - var(--adjacent));
    --startPosY: calc(var(--size) / 2 - var(--opposite) / 2);
    --endPosX: calc(var(--size) / 2 + var(--adjacent));
    --endPosY: calc(var(--size) / 2 + var(--opposite) / 2);

    --clip: polygon(
            var(--startPosX) var(--endPosX),
            50% var(--startPosY),
            var(--endPosX) var(--endPosY)
        );

    -webkit-clip-path: var(--clip);
    clip-path: var(--clip);
}

.triangle:nth-child(2) {
    --hypotenuse: 3rem;
}

.triangle:nth-child(2) {
    --hypotenuse: 50%;
}
```

Dynamic variables

Custom properties can be dynamic too. We can change them with JS and the values dependant on them will be automatically recalculated.

```
triangle.style.setProperty('--hypotenuse', '5rem')
```

CSS trigonometric functions have a lot of potential when they finally land, but sadly they're not yet supported in any browsers. To use trigonometry with dynamic variables right now, we need JavaScript.

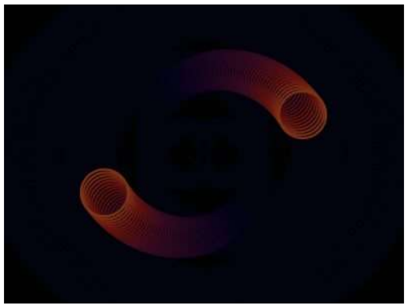
We'll take a look at some of the possibilities in the next article.

Michelle Barker

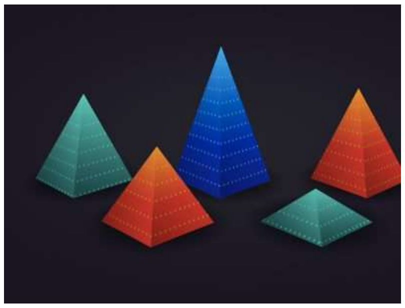
Michelle is a Lead Front End Developer at Bristol web agency Atomic Smash and author of front-end blog CSS { In Real Life }. She has written articles for CSS Tricks, Smashing Magazine, and Web Designer Magazine, to name a few. She enjoys experimenting with new CSS features and helping others learn about them.

https://michellebarker.co.uk

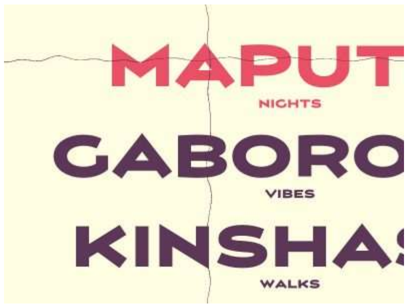
Twitter Instagram Codepen Github



Trigonometry in CSS and JavaScript: Beyond Triangles



Trigonometry in CSS and JavaScript: Getting Creative with Trigonometric Functions



How to Code a Crosshair Mouse Cursor with Distortion Hover Effect