Photo by Charles Deluvio 🇵🇭🇨🇦 on Unsplash

HOW TO: Pure CSS pie charts w/ CSS variables

Create customisable pie charts in ~30 lines of CSS 😎

Jhey Tompkins · Follow

Published in codeburst · 8 min read · Nov 7, 2018

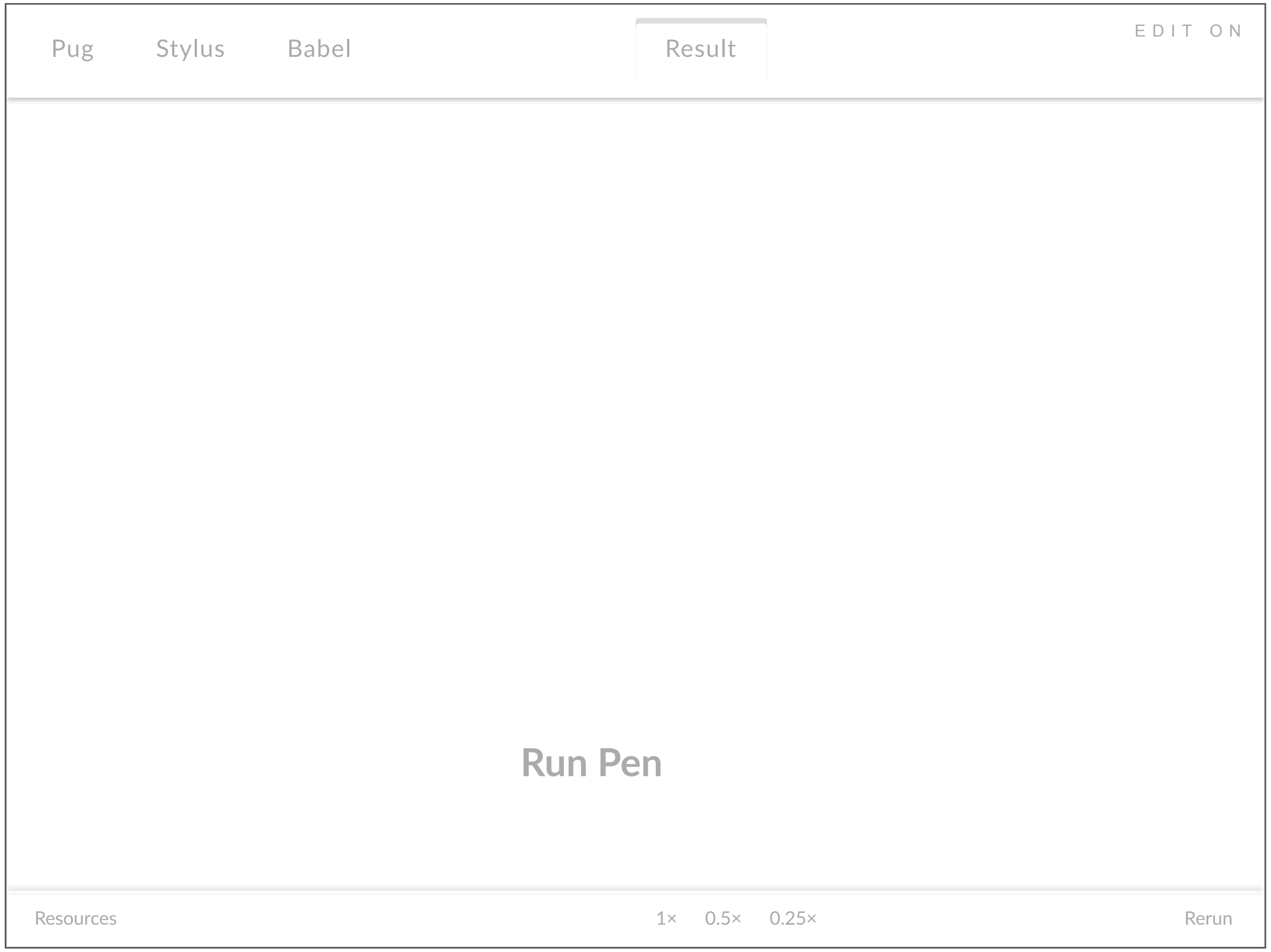
380 7

Ever needed to create a pie chart for your app but didn't want to pull in an entire library? I've got you covered. In around 30 lines of vanilla CSS we can create our very own pie charts 🙌

We are only creating the pie. We won't be creating the legend to go with it or any interactive features. These wouldn't be tricky to add though and I will touch on how to go about doing so.

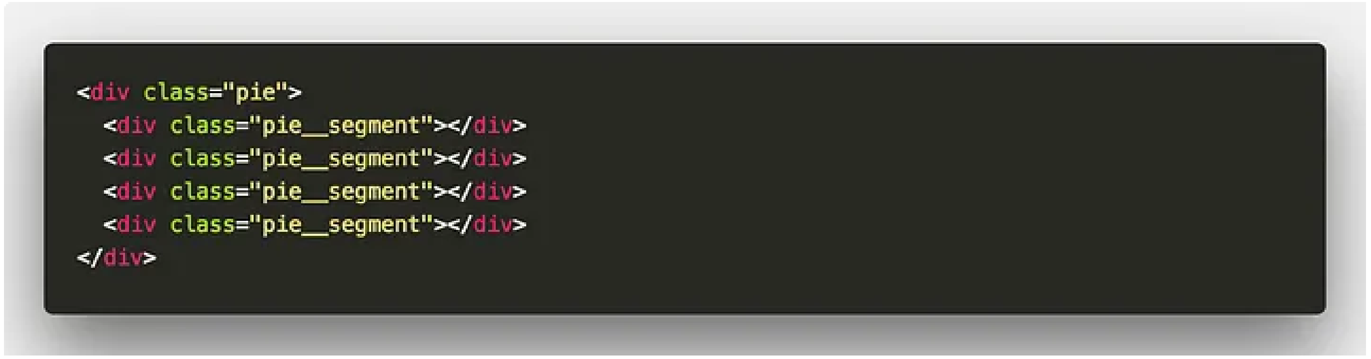
The trick to creating our charts will be to leverage `css` variables and `calc` along with `clip-path`. We will also use a `css` variable to act as a conditional 🤓 (I've listed some resources below if you're unfamiliar with `variables`, `calc` or `clip-path` 📚 👍)

For those in camp **TL;DR**, here's a demo to play with!



The Markup

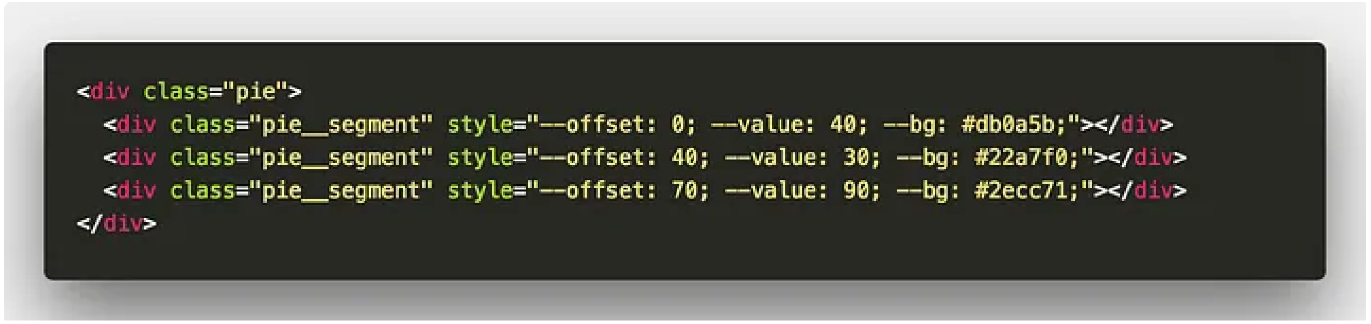
Let’s start with the markup. We want simple markup. Something like;



```
<div class="pie">
  <div class="pie__segment"></div>
  <div class="pie__segment"></div>
  <div class="pie__segment"></div>
  <div class="pie__segment"></div>
</div>
```

Basic pie chart markup

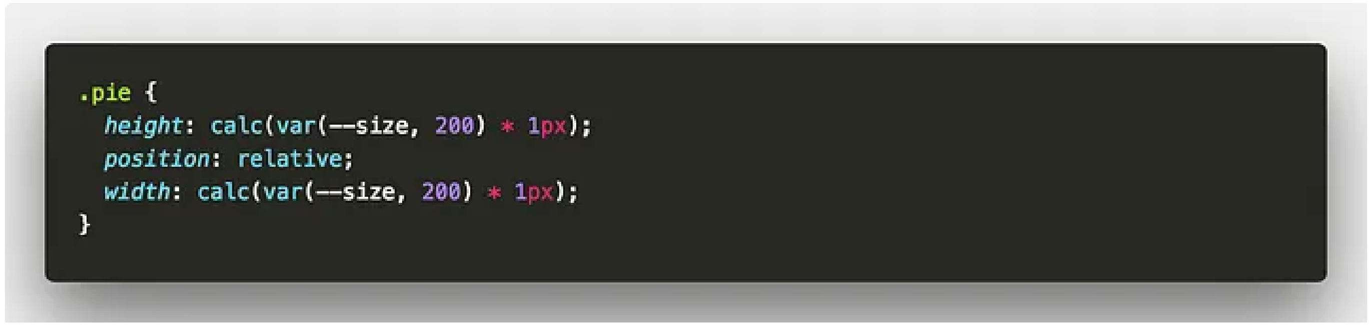
But how do we define values? Values such as start, end, color etc. We could use inline CSS variables on the segment!



```
<div class="pie">
  <div class="pie__segment" style="--offset: 0; --value: 40; --bg: #db0a5b;"></div>
  <div class="pie__segment" style="--offset: 40; --value: 30; --bg: #22a7f0;"></div>
  <div class="pie__segment" style="--offset: 70; --value: 90; --bg: #2ecc71;"></div>
</div>
```

The CSS

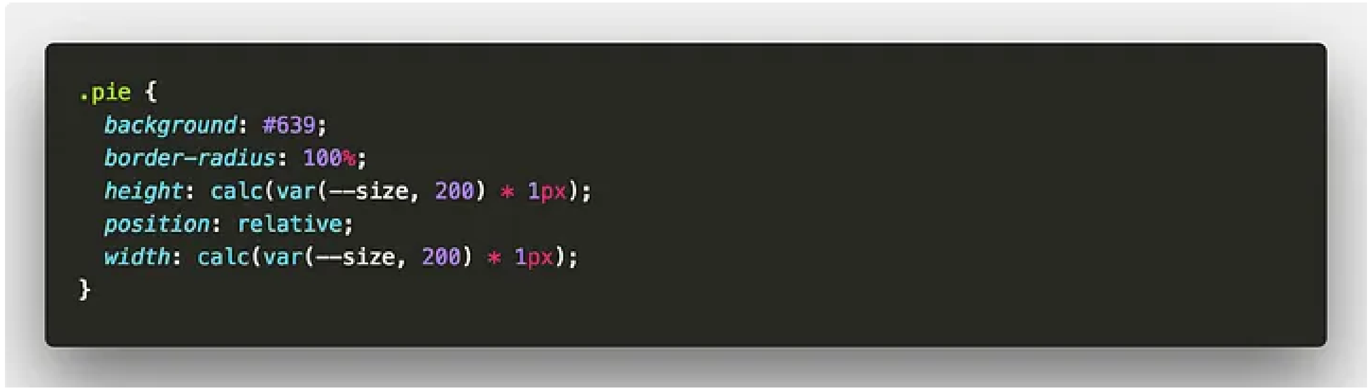
Let’s start with our pie element. It starts as a square. We want the size to be user configured. Let’s use a `size` variable. We will set the position to `relative` as our segments will have `absolute` positioning.



```css
.pie {
  height: calc(var(--size, 200) * 1px);
  position: relative;
  width: calc(var(--size, 200) * 1px);
}
```

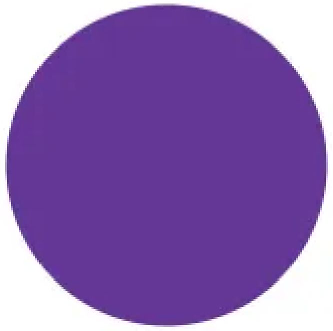
Configurable sizing

To make it a circle, we will use the `border-radius` property. Let's give it a `background-color` too so we can see what we are working with.



```css
.pie {
  background: #639;
  border-radius: 100%;
  height: calc(var(--size, 200) * 1px);
  position: relative;
  width: calc(var(--size, 200) * 1px);
}
```

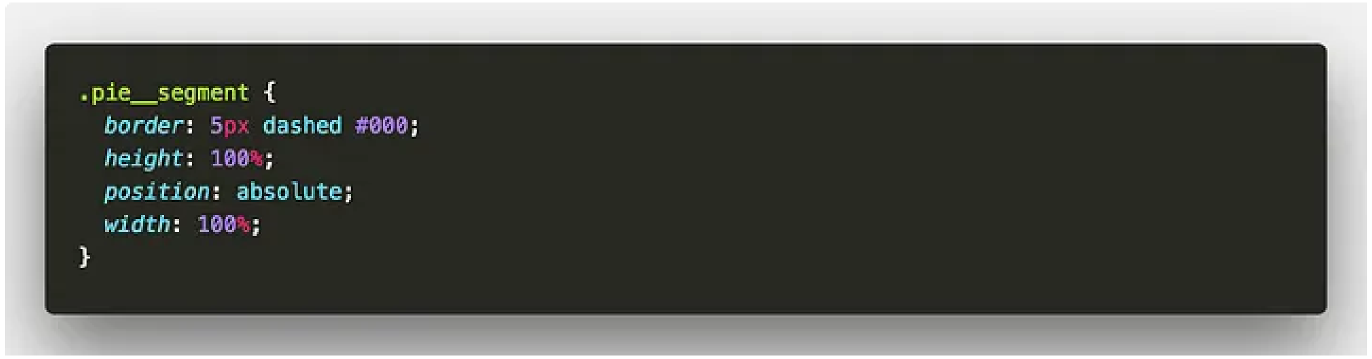
And this is what we have. The start of our pie.



The foundations of pie 😋

Segments

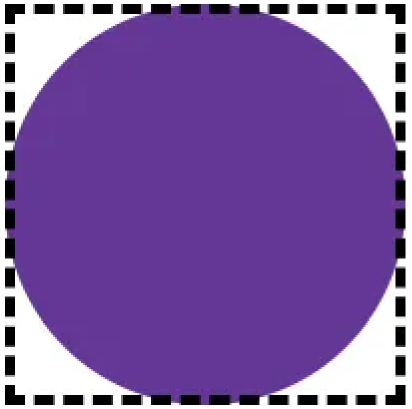
So how do we tackle segments? Let’s start by adding an element that’s `100%` `height` and `width` of the pie. Let’s give it a `border` so we can see what we are working with. Segments are going to have `absolute` positioning so they start in the same spot.



```css
.pie__segment {
  border: 5px dashed #000;
  height: 100%;
  position: absolute;
  width: 100%;
}
```

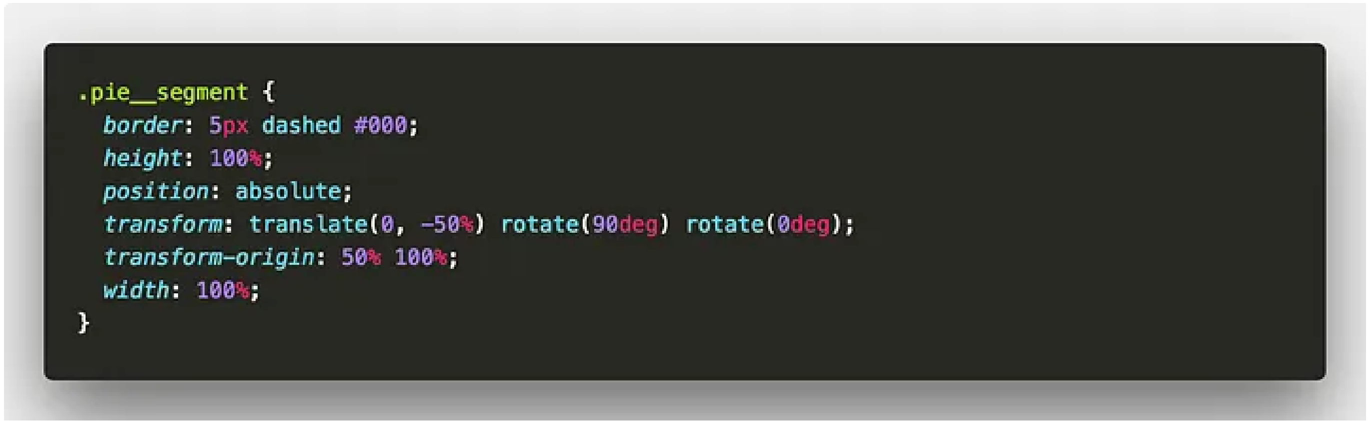
The start of our segment code

That gives us the following;



That does not look like a pie chart 👎

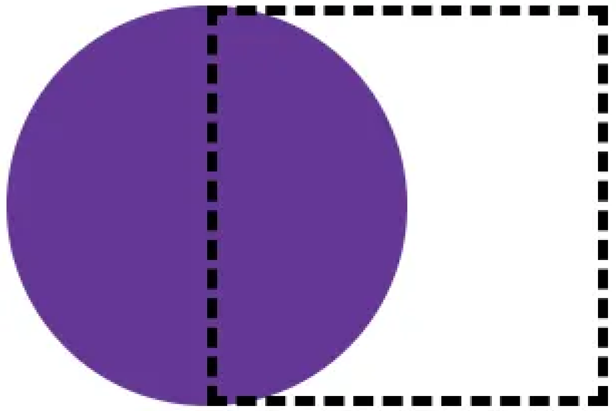
That looks nothing like a pie segment. Our segments are going to rotate around the center point of the pie starting at 12 o'clock. Let's apply a `transform` to the segment.



```
.pie__segment {
  border: 5px dashed #000;
  height: 100%;
  position: absolute;
  transform: translate(0, -50%) rotate(90deg) rotate(0deg);
  transform-origin: 50% 100%;
  width: 100%;
}
```

Apply a transform reset for segments

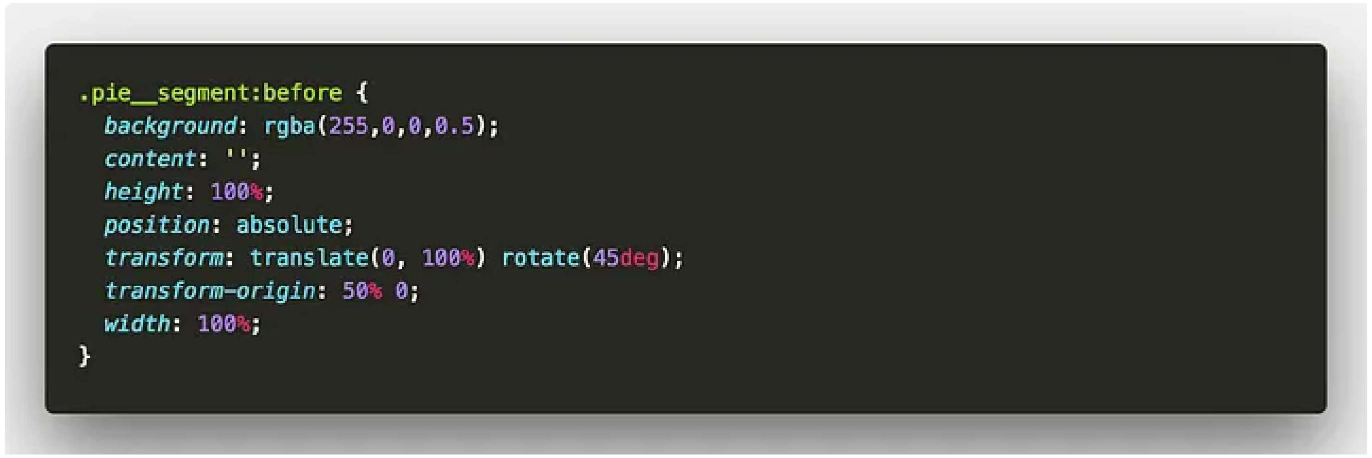
This gives us;



Let your imagination run and you can see where we are going 😉

This still might not look like it’s going to be a pie segment but believe 😃

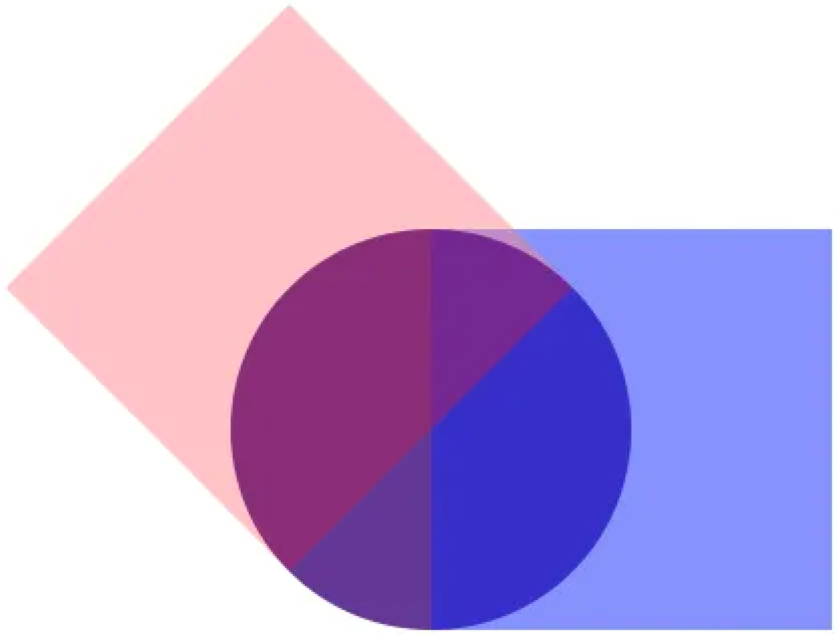
Let’s start by creating a hard coded segment. It starts at `0` and uses `45` degrees of the pie, that’s `12.5%`. Let’s leverage pseudo elements. We will also remove the segment border and add some hard coded colors and opacity to show the concept.



```css
.pie__segment:before {
  background: rgba(255,0,0,0.5);
  content: '';
  height: 100%;
  position: absolute;
  transform: translate(0, 100%) rotate(45deg);
  transform-origin: 50% 0;
  width: 100%;
}
```

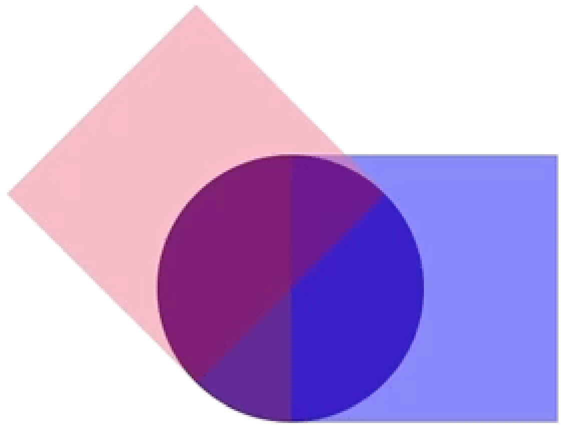
Starting to play with pseudo elements

This gives us



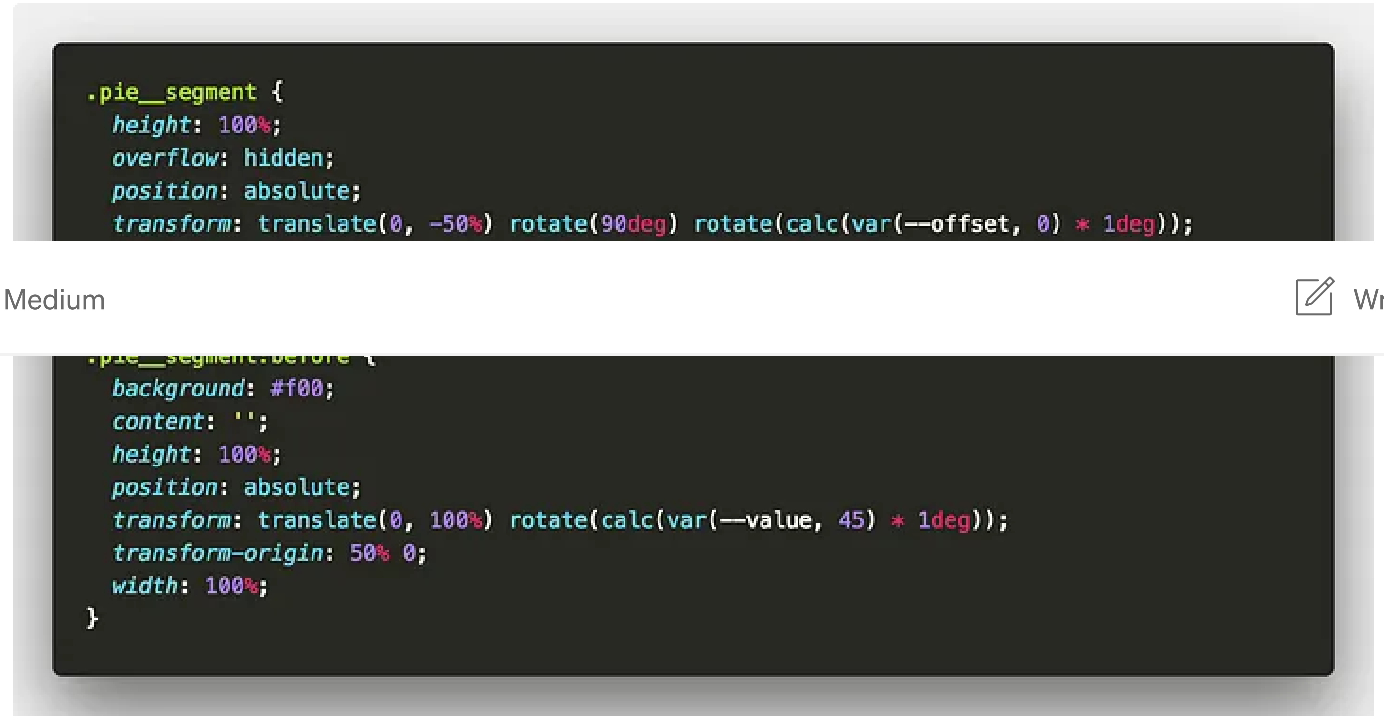
I think I see a segment hiding in there 👀

Now we can actually make out a segment. Removing the color and hiding the `overflow` on the pie and segment will give us something like this;



Ahhh, now I see it!

Let's apply that `overflow` to the segment and the pie. Let's also make the `offset` and `value` a `CSS` variable. We can use `calc` to determine the correct rotation.



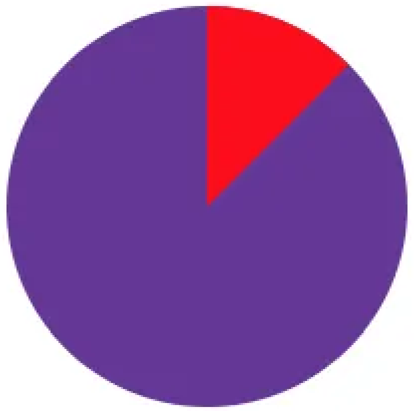
```css
.pie__segment {
  height: 100%;
  overflow: hidden;
  position: absolute;
  transform: translate(0, -50%) rotate(90deg) rotate(calc(var(--offset, 0) * 1deg));
```

Search Medium

Write

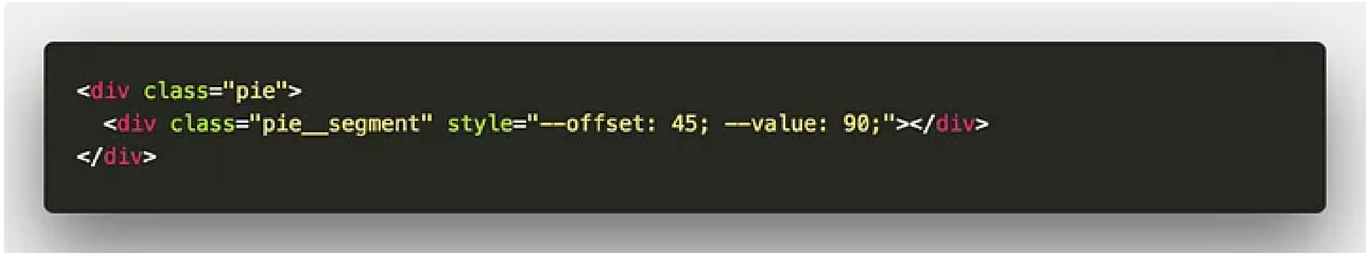
```css
.pie__segment:before {
  background: #f00;
  content: '';
  height: 100%;
  position: absolute;
  transform: translate(0, 100%) rotate(calc(var(--value, 45) * 1deg));
  transform-origin: 50% 0;
  width: 100%;
}
```

Introduce calc so we can be flexible 🏅



Our first true hard coded segment 🙌

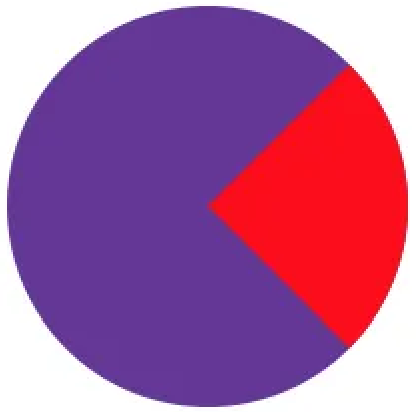
Now we can update our markup to update the value and offset of our segment!



```
<div class="pie">
  <div class="pie__segment" style="--offset: 45; --value: 90;"></div>
</div>
```

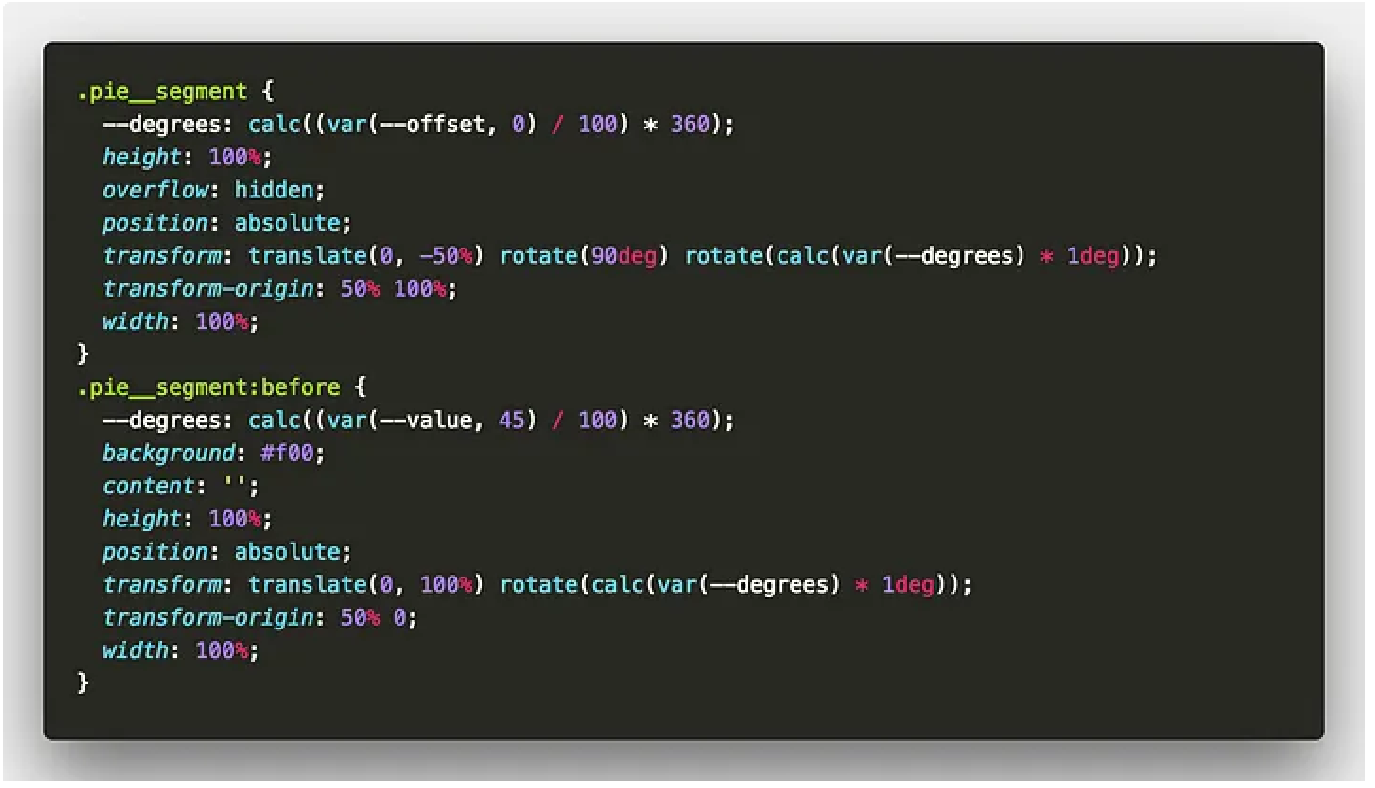
Taking it dynamic 👟

Now we have a segment that is offset by `45` degrees and has a value of `25%` or `90` degrees of the pie.



The result of calc! 🔩

It's not going to be ideal defining the `offset` and `value` in degrees. So let's update our calculation so it accepts a percentage representation of the pie 🤓



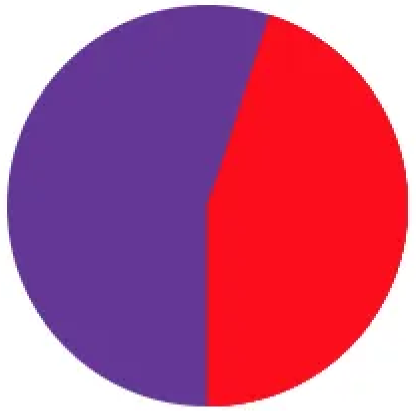
```
.pie__segment {
  --degrees: calc((var(--offset, 0) / 100) * 360);
  height: 100%;
  overflow: hidden;
  position: absolute;
  transform: translate(0, -50%) rotate(90deg) rotate(calc(var(--degrees) * 1deg));
  transform-origin: 50% 100%;
  width: 100%;
}
.pie__segment:before {
  --degrees: calc((var(--value, 45) / 100) * 360);
  background: #f00;
  content: '';
  height: 100%;
  position: absolute;
  transform: translate(0, 100%) rotate(calc(var(--degrees) * 1deg));
  transform-origin: 50% 0;
  width: 100%;
}
```

Convert the percentage value to degrees offset and value 🤓

For our last example, we can have a value of `25%` at an offset of `12.5%`.

An Issue

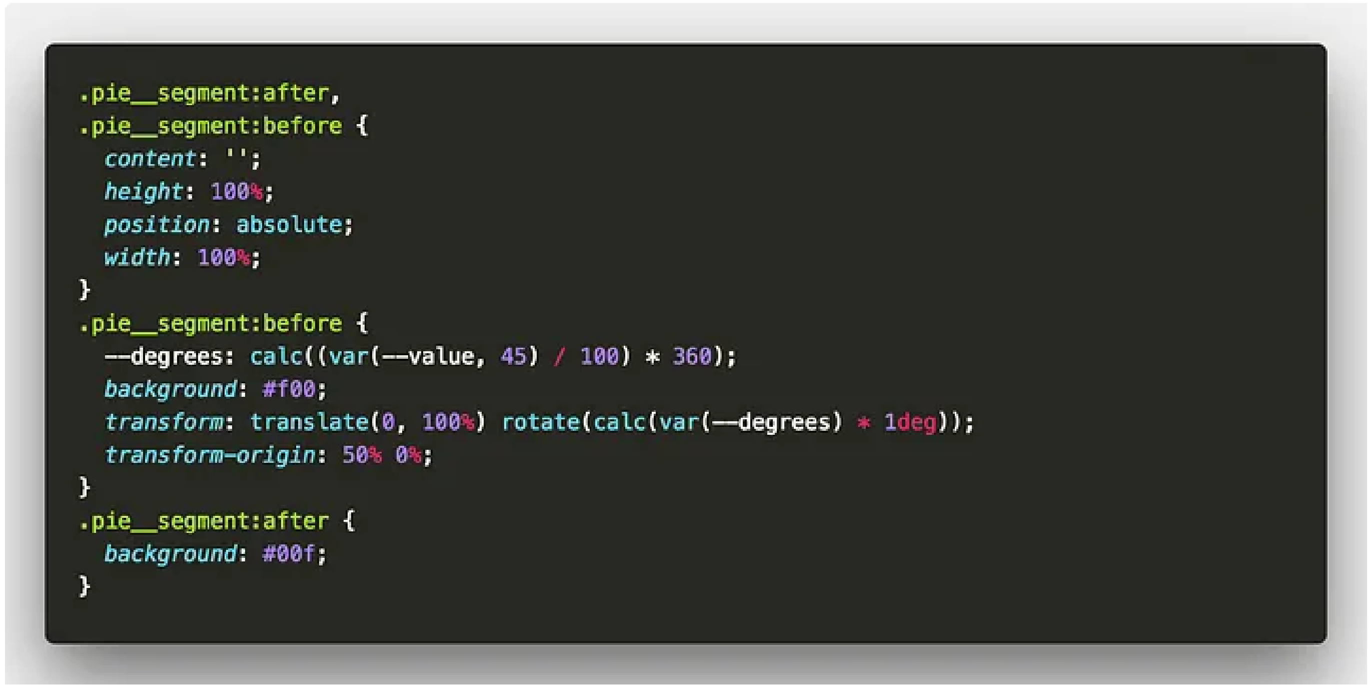
We have an issue. We can add segments and have dynamic values. But we our limited to a value of `50%` 😫. Any value over 50 breaks the segment. That won't do.



This segment should show 55%, instead it's like 45% at offset 5% 😭

We need to show more than what we have available. We do have another pseudo element at our disposal 🤔. Let's work on a segment that is `55%` percent at an offset of `0`.

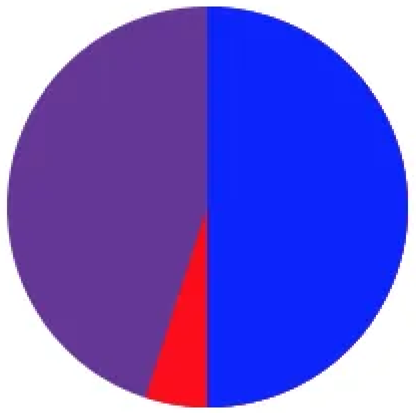
We can use the `:after` pseudo element to represent the first `50%`. We can do this by filling the segment. We can remove the `overflow: hidden` on our segment to show the extra value.



```css
.pie__segment:after,
.pie__segment:before {
  content: '';
  height: 100%;
  position: absolute;
  width: 100%;
}
.pie__segment:before {
  --degrees: calc((var(--value, 45) / 100) * 360);
  background: #f00;
  transform: translate(0, 100%) rotate(calc(var(--degrees) * 1deg));
  transform-origin: 50% 0%;
}
.pie__segment:after {
  background: #00f;
}
```

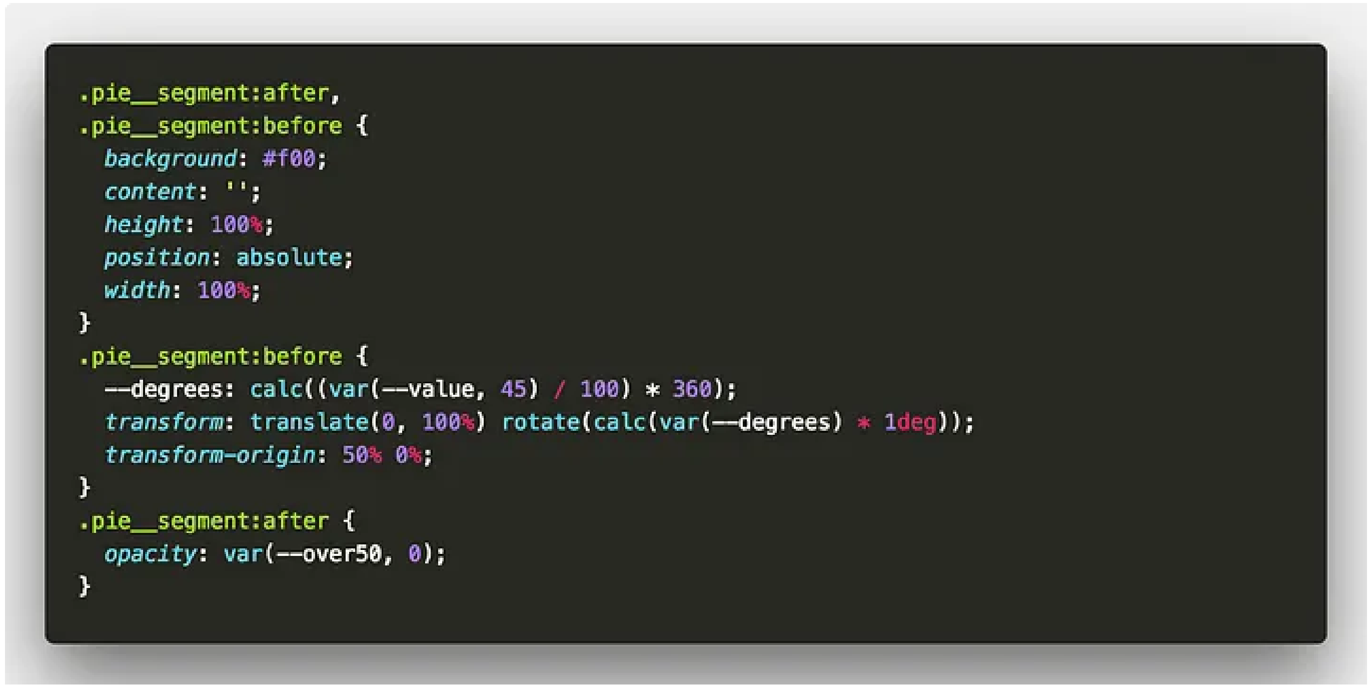
Bringing in :after to save the day!

But now there's another issue. The `:after` element always shows and we don't want it to. But how can we show it based on the value?



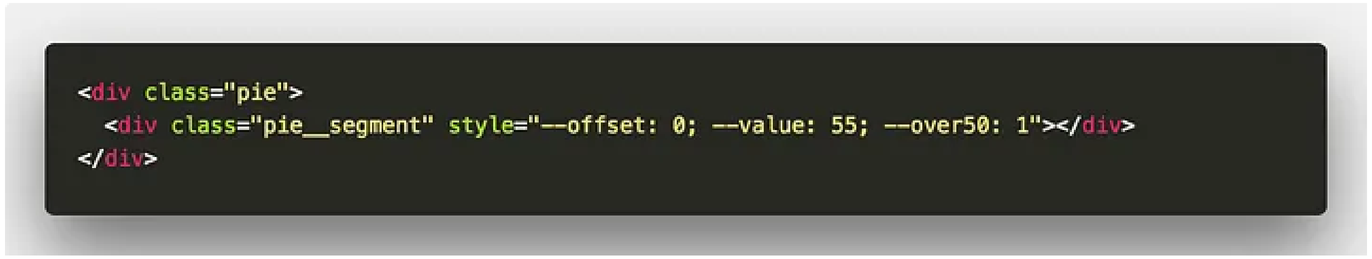
:after element is represented by blue for demo purposes!

What we can do is introduce a new `CSS` variable that will act as a conditional. Let's call it `over50`. If `over50` is `1`, then show the `:after` element. If it's `0`, then don't. We can use this to set the `opacity` of the after element. Make sure to set the default value to `0` also.



```css
.pie__segment:after,
.pie__segment:before {
  background: #f00;
  content: '';
  height: 100%;
  position: absolute;
  width: 100%;
}
.pie__segment:before {
  --degrees: calc((var(--value, 45) / 100) * 360);
  transform: translate(0, 100%) rotate(calc(var(--degrees) * 1deg));
  transform-origin: 50% 0%;
}
.pie__segment:after {
  opacity: var(--over50, 0);
}
```

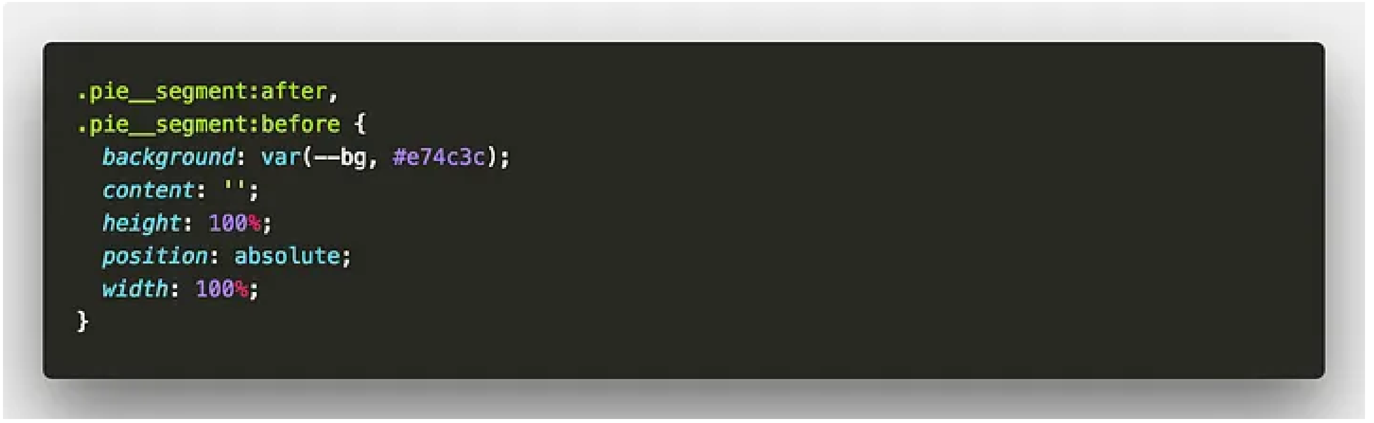
Make both pseudo elements the same color and only show :after when necessary 👍



```html
<div class="pie">
  <div class="pie__segment" style="--offset: 0; --value: 55; --over50: 1"></div>
</div>
```

Making use of the over50 variable 🔨

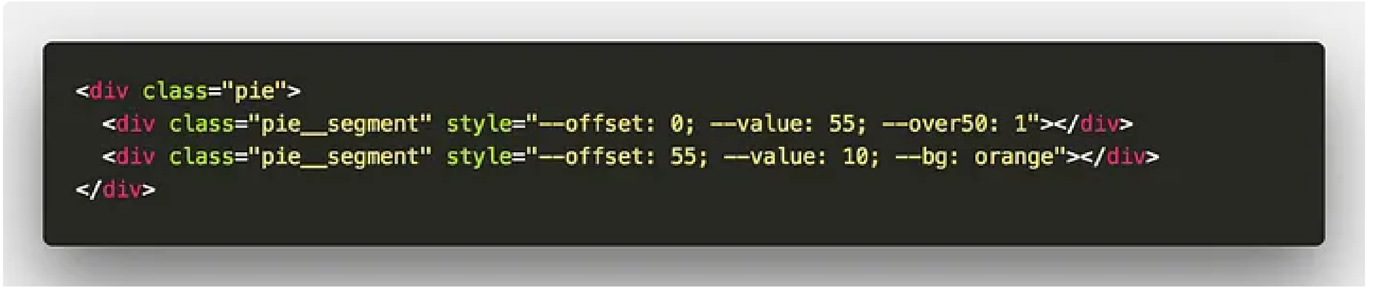
Let’s add color support so our segments can be different colors based on a variable.



```css
.pie__segment:after,
.pie__segment:before {
  background: var(--bg, #e74c3c);
  content: '';
  height: 100%;
  position: absolute;
  width: 100%;
}
```

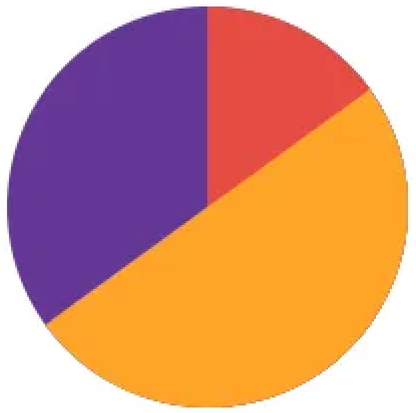
Sweet color support 🍭

Awesome! We are getting somewhere now. Let’s test it out by adding another segment.



```html
<div class="pie">
  <div class="pie__segment" style="--offset: 0; --value: 55; --over50: 1"></div>
  <div class="pie__segment" style="--offset: 55; --value: 10; --bg: orange"></div>
</div>
```

Introducing another segment 🍕

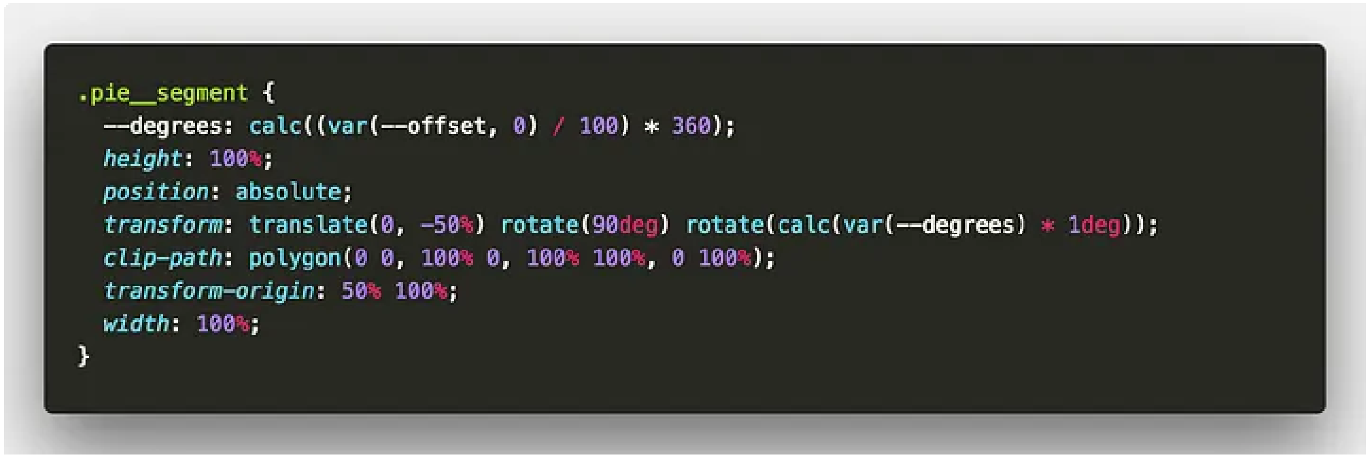


That's not right 👎

Well that doesn't look right! It seems we need that `overflow: hidden` on the segment. But only when the value is under `50`. Looks like we are going to need that `over50` variable some more.

How could we use a `CSS` variable to calculate the `overflow` though? With `clip-path` is the answer! (You can get familiar with `clip-path` here 📚)

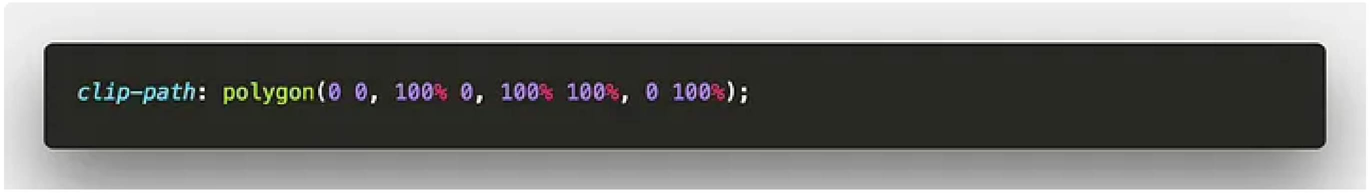
By default we want our segment to have no `overflow`. Using `clip-path` we can do this with polygon.



```css
.pie__segment {
  --degrees: calc((var(--offset, 0) / 100) * 360);
  height: 100%;
  position: absolute;
  transform: translate(0, -50%) rotate(90deg) rotate(calc(var(--degrees) * 1deg));
  clip-path: polygon(0 0, 100% 0, 100% 100%, 0 100%);
  transform-origin: 50% 100%;
  width: 100%;
}
```

Introduce a basic clip-path that equates to overflow: hidden

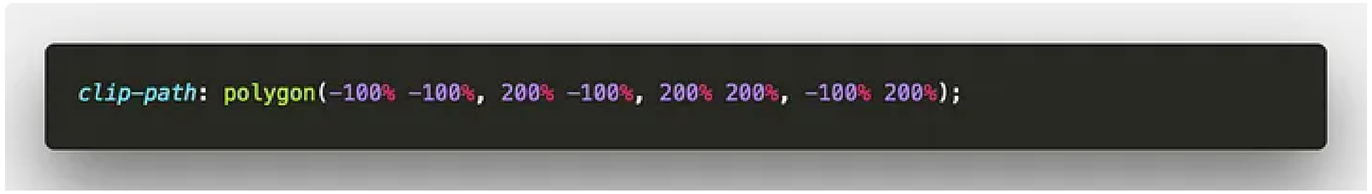
This use of `clip-path` gives the same result as `overflow: hidden`. The trick is that when the `over50` variable is set to `1` we want to show outside those bounds.



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

This is the smaller clip for overflow: hidden

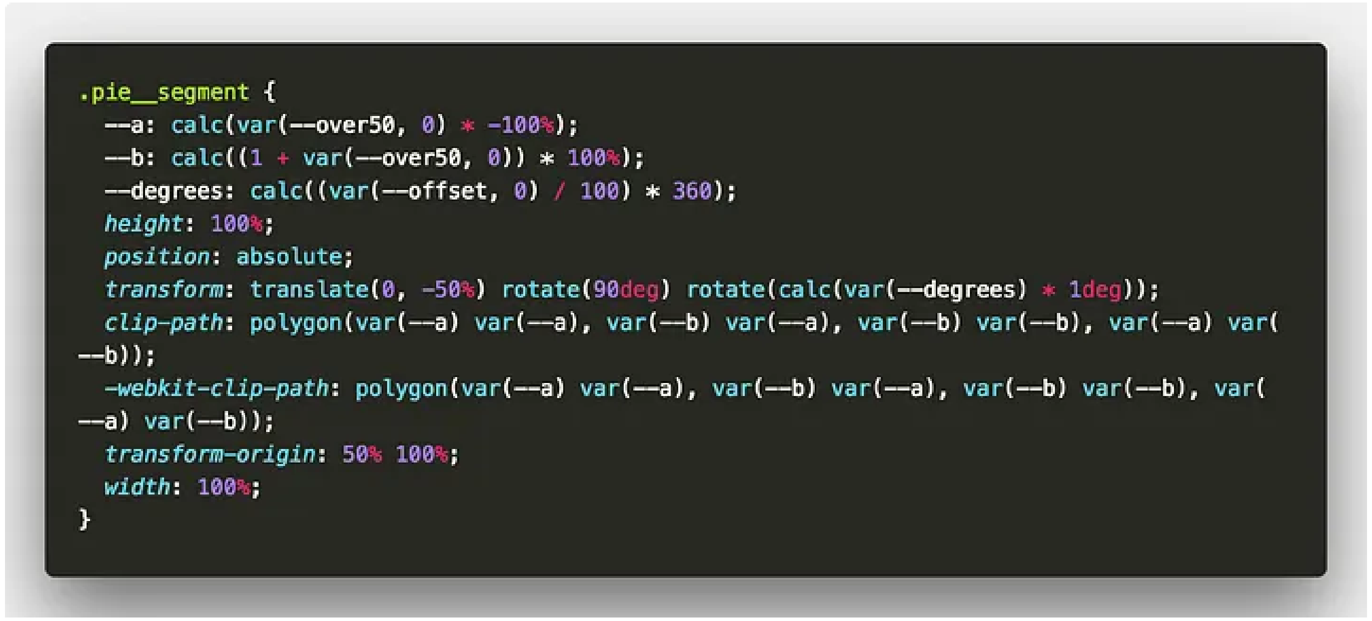
We can do this by multiplying the `over50` value against `-100%` and `100%` where appropriate. This will give us either a small clip or a big clip that allows us to show the overflow.



```css
clip-path: polygon(-100% -100%, 200% -100%, 200% 200%, -100% 200%);
```

The desired big clip for showing bigger segments 🍕

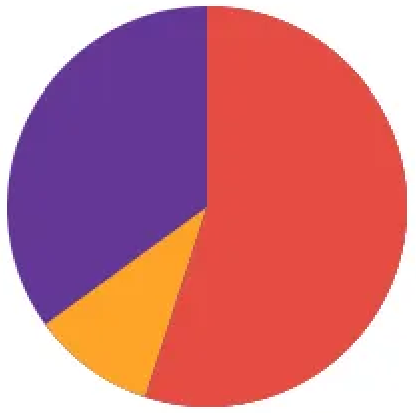
To generate the clip, we create two variables that will go into our `polygon` `clip-path`.



```css
.pie__segment {
  --a: calc(var(--over50, 0) * -100%);
  --b: calc((1 + var(--over50, 0)) * 100%);
  --degrees: calc((var(--offset, 0) / 100) * 360);
  height: 100%;
  position: absolute;
  transform: translate(0, -50%) rotate(90deg) rotate(calc(var(--degrees) * 1deg));
  clip-path: polygon(var(--a) var(--a), var(--b) var(--a), var(--b) var(--b), var(--a) var(
--b));
  -webkit-clip-path: polygon(var(--a) var(--a), var(--b) var(--a), var(--b) var(--b), var(
--a) var(--b));
  transform-origin: 50% 100%;
  width: 100%;
}
```

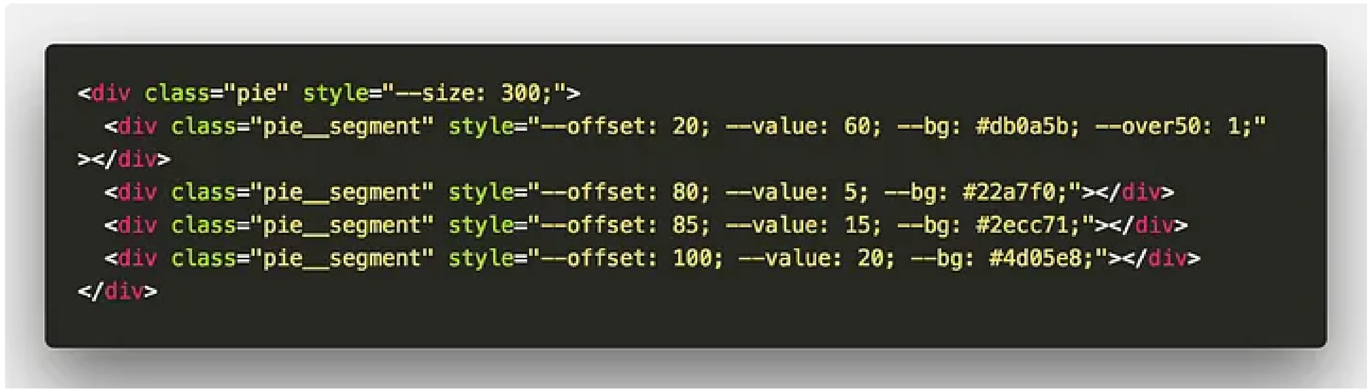
Generating the clip-path based on the value of over50 🤓

Putting that altogether and reloading the page now gives use the desired result 🎉



That's how it's meant to look!

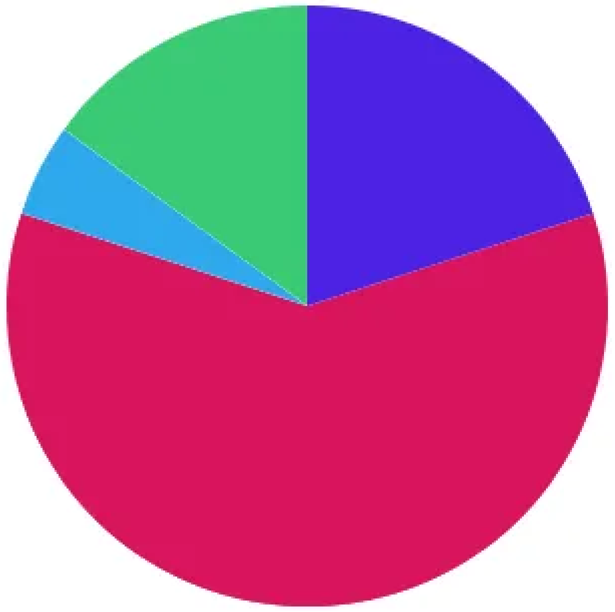
Now all that's left to do is remove the pie background and add some more segments ✊



```
<div class="pie" style="--size: 300;">
  <div class="pie__segment" style="--offset: 20; --value: 60; --bg: #db0a5b; --over50: 1;"
></div>
  <div class="pie__segment" style="--offset: 80; --value: 5; --bg: #22a7f0;"></div>
  <div class="pie__segment" style="--offset: 85; --value: 15; --bg: #2ecc71;"></div>
  <div class="pie__segment" style="--offset: 100; --value: 20; --bg: #4d05e8;"></div>
</div>
```

Let's add some extra segments and make something interesting!

And here's the result!



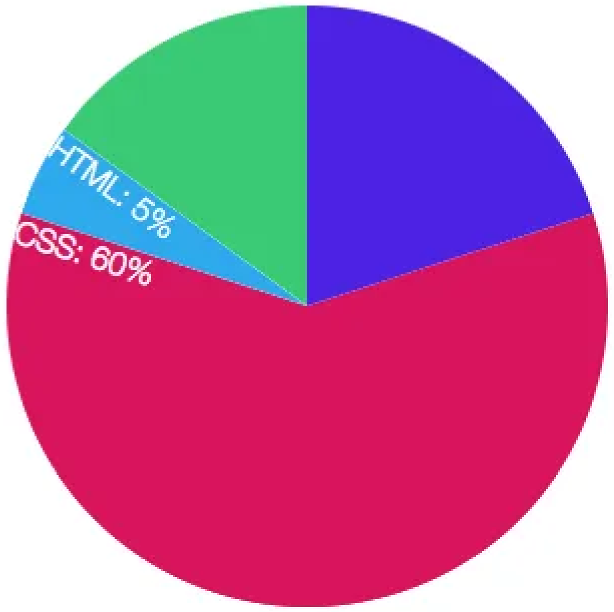
Sweet 🎉 A working pie chart with just CSS!

That’s pretty sweet 🍭

Extras

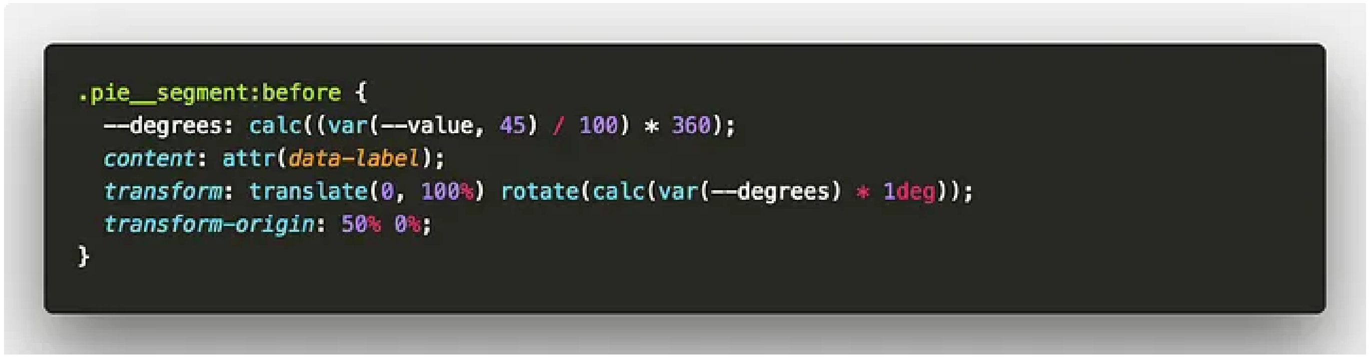
I did say I would touch on extras before we wrap things up. It’s likely if you did use these pie charts, you’d want some other features such as a legend, labels or hover effects. There are many possibilities. Let’s run through some.

Simple labels. Let’s start with applying a simple label to each segment.



Simple labels using attr()

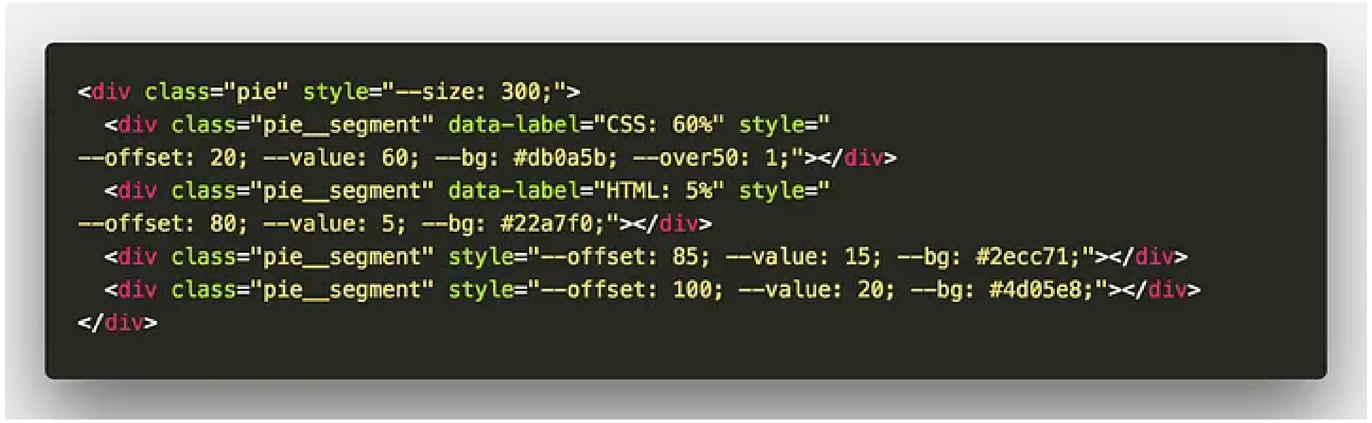
To do this we could make of the `content` property on our `:before` pseudo element.



```
.pie__segment:before {
  --degrees: calc((var(--value, 45) / 100) * 360);
  content: attr(data-label);
  transform: translate(0, 100%) rotate(calc(var(--degrees) * 1deg));
  transform-origin: 50% 0%;
}
```

Leveraging attr() on content 👍

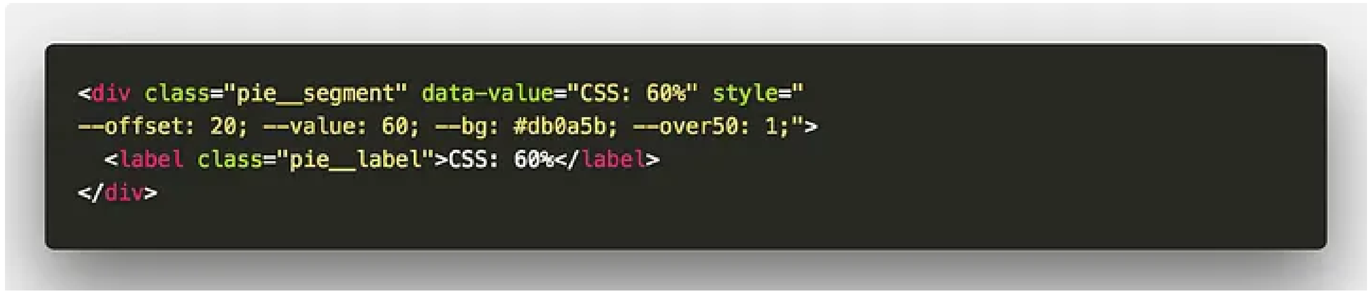
Then all we need do is send a value into the markup attribute.



```
<div class="pie" style="--size: 300;">
  <div class="pie__segment" data-label="CSS: 60%" style="
--offset: 20; --value: 60; --bg: #db0a5b; --over50: 1;"></div>
  <div class="pie__segment" data-label="HTML: 5%" style="
--offset: 80; --value: 5; --bg: #22a7f0;"></div>
  <div class="pie__segment" style="--offset: 85; --value: 15; --bg: #2ecc71;"></div>
  <div class="pie__segment" style="--offset: 100; --value: 20; --bg: #4d05e8;"></div>
</div>
```

Applying a label through the markup attribute

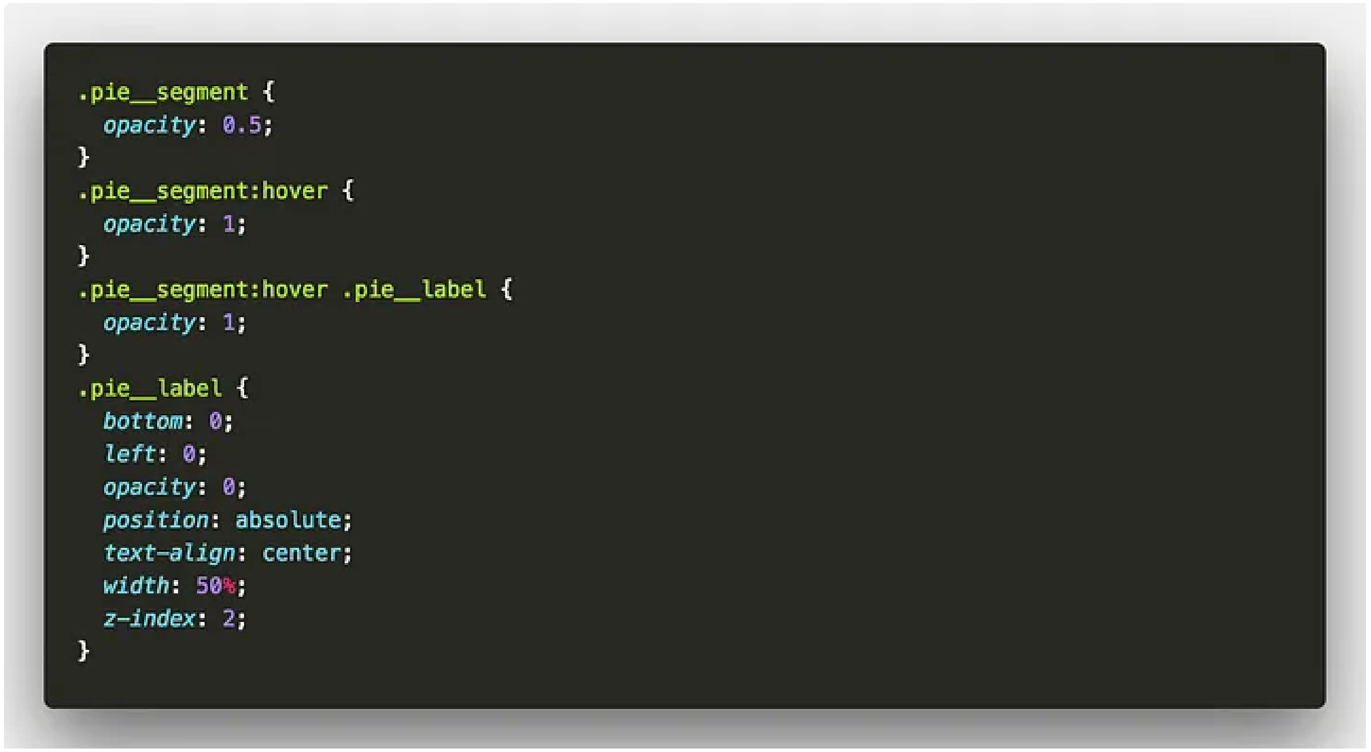
How about an actual element that shows on `:hover`. Here's some markup



```
<div class="pie__segment" data-value="CSS: 60%" style="
--offset: 20; --value: 60; --bg: #db0a5b; --over50: 1;">
  <label class="pie__label">CSS: 60%</label>
</div>
```

A simple pie label element

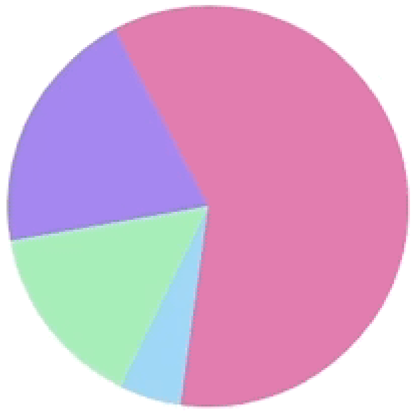
And then we apply some basic styles for it.



```css
.pie__segment {
  opacity: 0.5;
}
.pie__segment:hover {
  opacity: 1;
}
.pie__segment:hover .pie__label {
  opacity: 1;
}
.pie__label {
  bottom: 0;
  left: 0;
  opacity: 0;
  position: absolute;
  text-align: center;
  width: 50%;
  z-index: 2;
}
```

Only show that label when I hover over the segment 👀

And we get a label on hover



Show segment label on :hover 🐝

These short examples are a starting point for customisation.

That’s it!

We’ve created pure CSS pie charts in around 30 lines of `css` leveraging `calc` and variables 🙌 🎉

As always, any questions or suggestions, please feel free to leave a response or tweet me 🐦!

RESOURCES 📚

- `CSS clip-path` — MDN

- `CSS calc()` — MDN
- `CSS variables` — MDN

CSS Web Development Web Design Front End Development React

Written by Jhey Tompkins

Follow

1.6K Followers · Writer for codeburst

I make awesome things for awesome people!