

CUSTOM PROPERTIES

CSS Custom Properties are variables within CSS.

They are similar to pre-processor variables,
but they have advantages over them.

We can create a variable, and then references it over and over again in our file, making it much easier to make quick changes.

They take a little more setup, so it helps to set them up from the start of a project (more on this later).

THE SYNTAX

```
--name: value;
```

Unlike pre-processors, we have to declare custom properties within a CSS rule.



```
.selector {  
  --name: value;  
}
```

Custom properties take part in the cascade.

In general, it's very useful when they are globally accessible.



```
:root {  
  --name: value;  
}
```

The **:root** is the root of our document, which is the same as the **html** element.

We use **:root** instead because it has higher specificity. Chrome has optimized rendering for custom properties that are declared on the **:root** as well.

It's also a nice way to differentiate our variables and our actual styles.

HOW DO WE USE THEM?



```
:root {  
  --name: value;  
}
```



```
:root {  
  --clr-accent: #ff0;  
}
```



```
:root {  
  --clr-accent: #ff0;  
}  
  
a {  
  color: var(--clr-accent);  
  text-decoration: none;  
  font-weight: bold;  
}
```


Custom properties are inherited, so you can use a custom property in one of its children.

You can't, however, use a custom property defined in one place somewhere else.

```
:root {  
  --clr-accent: #ee3265;  
}
```

```
h1 {  
  color: var(--clr-accent);  
}
```

```
.btn {  
  background: var(--clr-accent);  
}
```

```
h1 {  
  --clr-accent: #ee3265;  
  color: var(--clr-accent);  
}
```

```
.btn {  
  background: var(--clr-accent);  
}
```

Because custom properties take part in the Cascade, you can overwrite them in a similar way.



```
:root {  --color: blue; }
```

```
div {  --color: green; }
```

```
.example {  --color: red; }
```

**BUT BE CAREFUL
WITH THIS!**

```
:root {  
  --font-size: 1.125rem;  
}  
  
body {  
  font-size: var(--font-size);  
}  
  
.call-to-action {  
  --font-size: 1.5rem;  
}
```

```
:root {  
  --font-size: 1.125rem;  
}  
  
body {  
  font-size: var(--font-size);  
}  
  
.call-to-action {  
  --font-size: 1.5rem;  
}
```



```
:root {  
  --font-size: 1.125rem;  
}  
  
body {  
  font-size: var(--font-size);  
}  
  
.call-to-action {  
  --font-size: 1.5rem;  
  font-size: var(--font-size);  
}
```



If you use a custom property that was never defined, the browser simply ignores it, the same way it'll ignore an invalid property.

This might seem useless, but when you use a custom property, you can provide a fallback.

This can be used in very powerful ways.



```
:root {  
  --clr-primary: #345;  
  --clr-secondary: #479;  
  --clr-accent: #ff0;  
}
```



```
.button {  
  display: inline-block;  
  padding: .5em 1.25em;  
  text-decoration: none;  
  font-weight: 700;  
  
  color: var(--btn-fg, #fff);  
  background: var(--btn-bg, #333);  
}
```

Q&A

I was wondering about the colour system you use. I've noticed from your recent live streams that your custom properties for colour follow a `--clr-primary-400` naming convention.

I was wondering what the 400 (number) part represents? I've noticed that the higher the number, the darker the colour (hue?). Is there a method behind this?

NAMING CUSTOM PROPERTIES.

```
50 :root {
51   --clr-primary: #fff;
52   --clr-primary-300: rgba(255,255,255,.8);
53   --clr-accent-300: #a9f0e2;
54   --clr-accent-400: #0fc5b0;
55   --clr-accent-500: #00937e;
56   --clr-accent-600: #0a4039;
57   --clr-secondary-200: #434956;
58   --clr-secondary-300: #272a2e;
59   --clr-secondary-400: #222529;
60   --clr-secondary-500: #1d1e20;
61
62   --ff-accent: atrament-web,sans-serif;
63   --ff-primary: basic-sans,sans-serif;
64
65   --fw-300: 300;
66   --fw-400: 400;
67   --fw-700: 700;
68   --fs-100: .64rem;
69   --fs-200: .8rem;
70   --fs-300: 1.125rem;
71   --fs-400: 1.325rem;
72   --fs-500: 1.563rem;
73   --fs-600: 1.563rem;
74   --fs-700: 2.0rem;
75   --fs-800: 2.5rem;
76   --fs-900: 3.2rem;
77   --fs-xl: clamp(4.5rem,1rem + 8vw,9rem);
78
79   --spacer: 2rem;
80
81   --linear-gradient: linear-gradient(90deg,var(--clr-secondary-300),var(--clr-secondary-400));
82   --glow-gradient: radial-gradient(var(--clr-secondary-300),var(--clr-secondary-400));
83 }
```

Q&A

I struggle with keeping **font-size from getting too small or too large** inside a container. I have your scrimba course and you go into viewport widths and when to use em or rem; but it just isn't intuitive to me. Is there perhaps a newer way?

**CUSTOM PROPERTIES
HAVE A SUPER POWER
OVER PREPROCESSORS**

**YOU CAN REDEFINE A
CUSTOM PROPERTY
INSIDE A MEDIA QUERY**

Q&A

It's always been figuring out **how selectors map to actual usage in the HTML** after you create them and then start applying them.

At the same time, **the interaction between two selectors and unintended side effects** that require you to debug the design to see how class/selectors/etc. have now been combined.

Do you still have to use hover when designing for mobile?

I want to know the behaviour's difference between Float and Absolute position.

Second, I want to know how to manage the Layout's Height, we see a lot of about the Width but anything about Height.

Re **naming conventions** - I'm trying to work up a plug and play set of layout, element and utility custom css classes that I can just apply for each project.

I've looked at a number of them BEM CUBE etc and all look a little ott for a simple project. Any pointers to implement action and to stop things getting messy and confused?