

Common Mistakes and Gotchas

CSS variables are sweet to work with but there are a few gotchas. You don't want to be making these mistakes and struggling with needless bugs.

Resolving Multiple Declarations

As with other properties, multiple declarations are resolved with the standard cascade.

Let's see an example

```
/*define the variables*/
:root {
  --color: blue;
}

div {
  --color: green;
}

#alert {
  --color: red;
}

/*use the variable on all elements*/
* {
  color: var(--color);
}
```

With the variable declarations above, what will be the color of the following elements?

```
<p>What is my color? </p>
<div>and me? </div>
<div id='alert'>
  What is my color too?
```

```
what is my color too?  
<p>color? </p>  
  
</div>
```

Can you figure it out?

The Solution

The first paragraph will be **blue**. There is no direct **--color** definition set on a **p** selector, so it inherits the value from **:root**

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

The first **div** will be **green**. That's kind of obvious. There's a direct variable definition set on the **div**

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

The **div** with the ID of **alert** will NOT be green. It will be **red**

```
#alert { --color: red; }
```

The ID has a direct variable scoping. As such, the value within the definition will override the others. The selector **#alert** is more specific.

Finally, the **p** within the **#alert** will be ... **red**

There's no variable declaration on the paragraph element. You may have expected the color to be **blue** owing to the declaration on the **:root** element.

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

As with other properties, CSS variables are inherited. The value is inherited from the parent, **#alert**

```
#alert { --color: red; }
```



Resolving Cyclic Dependencies

A cyclic dependency occurs in the following ways: (a) when a variable depends on itself i.e uses a `var()` that refers to itself

```
:root {
  --m: var(--m)
}

body {
  margin: var(--m)
```

```
margin: var(--m);  
}
```

(b) When two or more variables refer to each other

```
:root {  
  --one: calc(var(--two) + 10px);  
  --two: calc(var(--one) - 10px);  
}
```


Do not create cyclic dependencies within your code.

What Happens with Invalid Variables?

Syntax errors are discarded, but invalid `var()` substitutions default to either the initial or inherited value of the property in question .

Consider the following:

```
:root { --color: 20px; }  
p { background-color: red; }  
p { background-color: var(--color); }
```

```
:root { --color: 20px; }  
p { background-color:  red; }  
p { background-color: var(--color); }
```

Awful attempt to
Set a non-color value
Background-color: 20px

As expected, `--color` is substituted Into `var()` but the property value, `background-color: 20px` is invalid after the substitution. Since `background-color` isn't an inheritable property, the value will default to its `initial` value of `transparent`

```
:root { --color: 20px; }  
p { background-color: red; }  
p { background-color: var(--color); }
```

Background-color: 20px
Since you're invalid, I'll default to the initial value of `transparent`



Note that if you had written `background-color: 20px` without any variable substitutes, the particular background declaration would have been invalid. The previous declaration will then be used.

```
:root { --color: 20px; }  
p { background-color: red; }  
p { background-color: 20px; }
```

If you wrote this, the declaration would be invalid. Ignored. Thus, background-color will take the previous value of red. Ooops.



Be Careful While Building Single Tokens

When you set the value of a property like so:

```
font-size: 20px
```

The `20px` is interpreted as a single token. A simple way to put that is, the value `20px` is seen as a single 'entity'

You need to be careful when building single tokens with CSS variables.

For example, consider the following block of code:

```
:root {
```

```
--size: 20  
}  
  
div {  
  font-size: var(--size)px  
}
```

You may have expected the value of `font-size` to yield `20px`, but that is wrong. The browser interprets this as `20 px`. Note the space after the `20`.

Thus, if you must create single tokens, have a variable represent the entire token e.g. `--size: 20px`, or use the `calc` function e.g. `calc(--size) * 1px` where `--size` is equal to `20`.

Don't worry if you don't get this. I explain it better in a coming practical lessons.

Let's Build Stuff

This is the part I've been waiting for. It's likely the same for you.

I'll walk you through practical applications of the concepts discussed by building a few useful projects.

Let's get started in the next lesson.