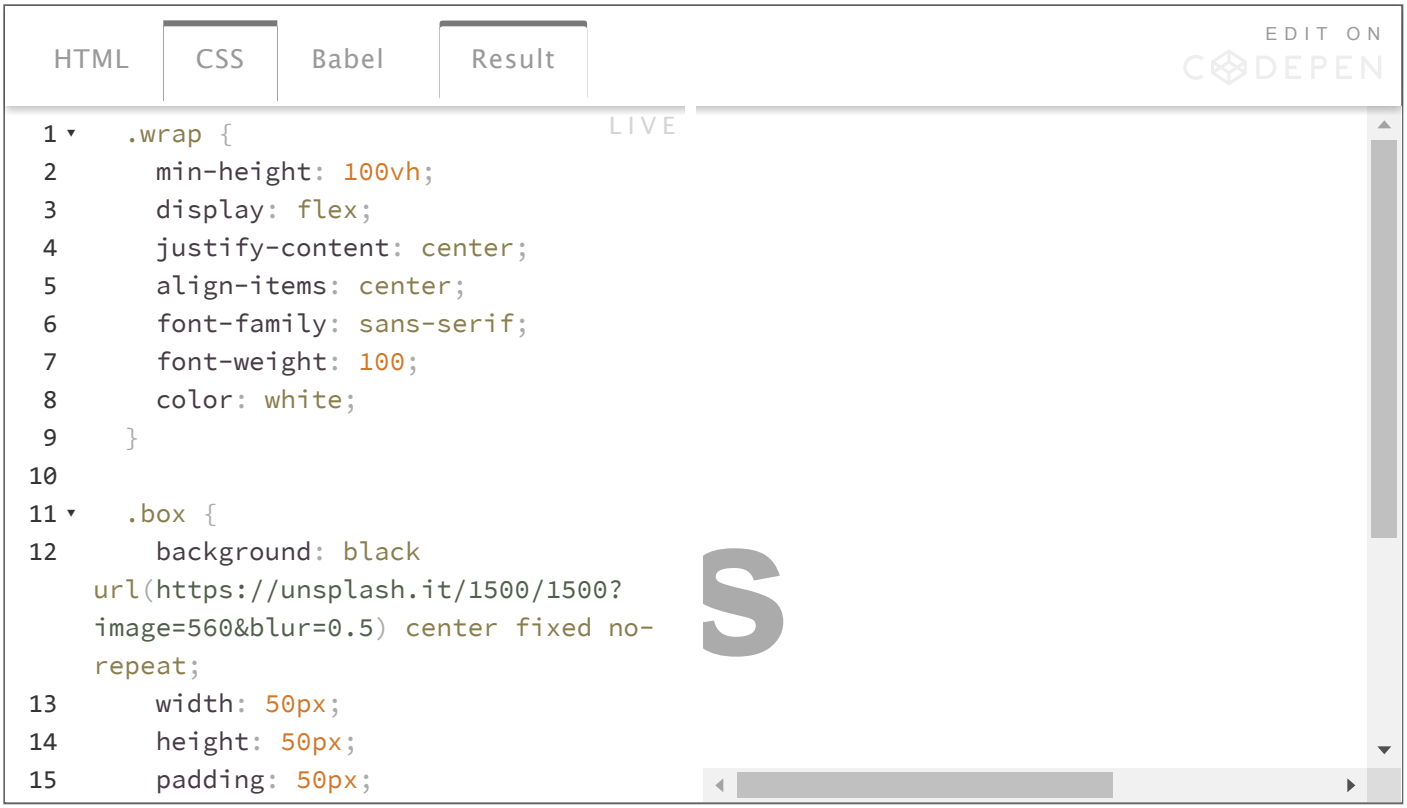
SEP 10 2016

We've learned about the concise syntax of arrow functions. We've learned about the implicit return of arrow functions. The last thing we're going to learn about and probably the most important is that the fact that `this` keyword **does not get rebound**.

What does that mean? Let's show an example as to when you might run into this. This is a pretty visual one so you might be better off watching the corresponding ES6.io video. We will be creating this:



What I have here is I've got this `div` with the class of `box` right here.

Markup

```
<div class="wrap">
    <div class="box">
        <h2>Wes Bos</h2>
        <p class="social">@wesbos</p>
    </div>
</div>
```

When you click that box what's going to happen is a two-stage animation. You click it, and it grows. Then it animates in the `h2`, and the `social` paragraph, from the left and from the right.

With the source files, you can try this out in your browser's element inspector. First add a class of `opening` to the `div`:

Markup

```
<div class="wrap">
    <div class="box opening">
        <h2>Wes Bos</h2>
        <p class="social">@wesbos</p>
    </div>
</div>
```

What that does it actually grows it.

Then if you add a class of `open()` to it, that will bring in the text.

Markup

```
<div class="wrap">
    <div class="box opening open">
        <h2>Wes Bos</h2>
        <p class="social">@wesbos</p>
    </div>
</div>
```

It's always on the `<div class="box">`, adding `opening`, and then after a couple of seconds I have a class of `open` being added to it.

I've given you all of the CSS that comes along with it. Nothing too exciting here, and that's a whole another course together.

Essentially the way it works is when it has a class of `opening` I just change the width and the height. Then, when it has a class of `open`, I bring all of the text in. Then I've got transitions on everything so it goes has some funky effects.

If you open it in the browser and click it, though, you'll see that nothing works.

We need to select that element. So in the `<script>` tags, we're going to use `const`, just because we won't want the reference to the box to change.

JavaScript

```
const box = document.querySelector('.box');
console.log(box);
```

If you run that, you can see that the `box` is logged to the console.

Here we are going to type a regular function, and include a `console.log` so we can take a look at what's happening:

JavaScript

```
const box = document.querySelector('.box');
box.addEventListener('click', function() {
    console.log(this);
});
```

It logs `this` as the same as `box`. The way you can think about this is you look at what got called `addEventListener`, and you look at the thing to the left of it, `box`. What's `box`? That's the `div` with the class of `box` that we have.

That's good. But, if you swap this out with an arrow function here, watch what happens:

JavaScript

```
const box = document.querySelector('.box');
box.addEventListener('click', () => {
```

```
    console.log(this);
});
```

We get `Window`. Why do we get `Window` here?

That's because when you use an arrow function, the value of `this` is **not rebound** inside of that function. It is just inherited form whatever the parent scope is.

What's the parent scope of this? It's `Window`, as in the browser window.

If you use your console here and type `this`, you'll see that `this` is equal to `Window`, because it's not being bound to anything. It ends up by the window.

You don't just want to go willy-nilly using arrow functions everywhere, because it's just less to type. You need to know what the benefits and the drawbacks of them are. In this case I don't want an arrow function, because I need the keyword to reference the actual box that got clicked. That would be even more important if I had a whole bunch of them.

We can't use an arrow function there. I'm going to bring that back to regular function.

JavaScript

```
const box = document.querySelector('.box');
box.addEventListener('click', function() {
    console.log(this);
});
```

That's what we want. You generally want these functions for your top level ones. Then inside of that, we're going to replace `console.log` with:

JavaScript

```
const box = document.querySelector('.box');
box.addEventListener('click', function() {
    this.classList.toggle('opening');
});
```

Let's see how that works. The box should animate itself in and out, in and out. If it does that, good.

After maybe 500 milliseconds or so, I want to also toggle `open`, because that's the final stage. Remember, it's a two-stage animation here.

I think we'll use a timeout for that:

JavaScript

```
const box = document.querySelector('.box');
box.addEventListener('click', function() {
    this.classList.toggle('opening');
    setTimeout(function() {
       this.classList.toggle('open');
    });
});
```

OK, so does that work? No.

Our console says `Uncaught type error cannot read property toggle of undefined`. That's weird. How do I debug that?

Let's take a look using `console.log` to help us out:

JavaScript

```
const box = document.querySelector('.box');
box.addEventListener('click', function() {
    this.classList.toggle('opening');
    setTimeout(function() {
       console.log(this.classList);
       this.classList.toggle('open');
    });
});
```

If we run that, we see that `this.classList` comes back as `undefined`. It's nothing. Why is there no `classList` on the box?

Let's take another look, this time we'll just `console.log(this)` to get a little deeper:

JavaScript

```
const box = document.querySelector('.box');
box.addEventListener('click', function() {
    this.classList.toggle('opening');
    setTimeout(function() {
       console.log(this);
       this.classList.toggle('open');
    });
});
```

We'll see that it's targeting `Window`. Why's that?

If `this` inside the `opening` toggle function here was equal to `box`, why the heck is it not equal to `box` for the open function?

That's because we've entered a new function, and `this` inside the function has **not been bound** to anything, which means that, if it's not

bound to anything, it's going to be equal to the window. That's a pain in the ass.

A lot of people would solve this by adding a `self` variable and use it to trigger the class `open` like this:

JavaScript

```
const box = document.querySelector('.box');
box.addEventListener('click', function() {
    var self = this;
    this.classList.toggle('opening');
    setTimeout(function() {
       console.log(this);
       self.classList.toggle('open');
    });
});
```

That's works, but not the greatest, because we have this weird or some of you like to say `var that = this;` and etc.

Fortunately, we don't need to do that anymore if I bring that back to this. What we need to do is just simply make it an arrow function:

JavaScript

```
const box = document.querySelector('.box');
box.addEventListener('click', function() {
    this.classList.toggle('opening');
    setTimeout(() => {
       console.log(this);
       this.classList.toggle('open');
    });
});
```

Why? Because when you have an arrow function, **it does not change the value** of `this`. It inherits the value of this from the **parent**. We don't have to worry about the scope changing or anything like that.

We can just go ahead and keep working using this as if it was scoped to this actual function here, just great.

JavaScript

```
const box = document.querySelector('.box');
box.addEventListener('click', function() {
    this.classList.toggle('opening');
    setTimeout(() => {
       console.log(this);
       this.classList.toggle('open');
    }, 500);
});
```

We should probably put 500 milliseconds in here.

So let's run that, and click it again and it closes. That's a little bit funky. I'm going to fix this to be a little bit nicer.

None of this has anything to do with arrow functions. If you're interested in seeing how I might figure this out, you can stay on with me.

The problem with this right here is that when you toggle an `open`, it adds a class of `opening`, then after 500 milliseconds it adds a class of `open`. When you close it down, you want it to be the opposite.

The way I would probably solve this is to first make two variables.

JavaScript

```
const box = document.querySelector('.box');
box.addEventListener('click', function() {
    let first = 'opening';
    let second ='open';
    this.classList.toggle(first);
    setTimeout(() => {
       console.log(this);
       this.classList.toggle(second);
    }, 500);
});
```

I'm just using variables so we still have this problem.

Let's add an `if` statement to switch our variables around. This is going to look forward into our destructuring exercise, but as little hot tip. If we want to switch two variables with ES6 you can simply put them in an array:

JavaScript

```
const box = document.querySelector('.box');
box.addEventListener('click', function() {
    let first = 'opening';
    let second = 'open';

    if (this.classList.contains(first)) {
        [first, second] = [second, first];
    }
    this.classList.toggle(first);
    setTimeout(() => {
       console.log(this);
       this.classList.toggle(second);
    }, 500);
});
```

Because we've switched the variables, we should be able to get this to animate in the correct way.

Check this out. Click it open, click it closed, and you have a fun little animation that we've made there.

The big takeaway here is that we can use an arrow function for things inside of a normal function and it's going to inherit the value of `this`.

This entry was posted in ES6, JavaScript. Bookmark the permalink.

One Response to *JavaScript Arrow Functions and this scoping*

Jon Beck *says:*

September 12, 2016 at 8:59 am

Wes – what does this mean in terms of Garbage collection?

"The big takeaway here is that we can use an arrow function for things inside of a normal function and it's going to inherit the value of this."

Does that mean I've made in memory, attached to the Window, a copy of whatever this is? And because it's attached to the Window, it will never be freed by JS garbage collectors?

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment

Notify me of follow-up comments by email.

Notify me of new posts by email.

Follow @wesbos 116K followers