Arrow Functions

Arrow functions are anonymous functions

- one reason is for conciseness (less typing)
- lexical `this`

Using the old function syntax can cause `this` to forget what context we're referring to And arrow functions won't allow `this` to **not forget**

In the old `function` we would need to use `bind()` or `apply()` to keep track of `this`

Arrow functions & Lexical `this`

```
var workshop = {
  teacher: "Kyle",
  ask(question) {
    setTimeout(() => {
      console.log(this.teacher,question);
    }, 100)
  },
};

workshop.ask("Is this lexical 'this'?");
// Kyle Is this lexical 'this'
```

The `this` keyword, when the arrow function is invoked, is correctly pointing at the workshop object.

This is what we refer to as lexical `this` behavior

So what does lexical `this` mean?

Many people have in their minds a mental model that a arrow function is essentially a hardbound function to the parent `this` **this is not accurate**!

The proper way to think about what an arrow function is.. **an arrow function does not define a `this` keyword at all**! There is no such thing as a `this` keyword in an arrow function It will behave like any other variable, meaning it will lexically resolve to some enclosing scope that does define the `this` keyword

So in this particular case, the arrow function, when we say `console.log(this.teacher,question);` There is no `this` in that arrow function! No matter how it gets invoked So we lexically go up one level of scope which is, which function? `ask()` function.. It goes out from the callback fnuction, the arrow function, that scope to the enclosing scope, which is the `ask()` function

Remember global is a red marble... workshop is a red marble, but we create a blue bucket ask is a blue marble, but we create a green bucket our arrow function is a green marble

So since there is no `this` in the arrow function we out to our enclosing scope, and that's the green bucket (aka `ask` function scope)

And what is `ask`s definition of the `this` keyword?

We determine that by **how we call the function `ask`:** `workshop.ask("Is this lexical 'this'?");` We attached a context object to our function invocation... Therefore we look in that object for `this`

Because the `ask` functions `this` keyword gets set by the call site And then when that callback gets later invoked, it's essentially closed over. That parent scope that had a `this` keyword pointing at the workshop object That's what we mean by lexical this

We're not hard-binding the fnuction It's a function that doesn't have a `this` **at all** And so **it resolves lexically**, that means if it had to go up, five levels because you had 5 nested arrow functions, it just keeps going and going and going up the building elevator until it finds a function that defines a `this` keyword...

And whatever the `this` keyword points at for that function, that's what it uses...

So arrow functions don't have a `this` keyword, so it doesn't even seach there, just goes up 1 layer at a time searching for a function that **does** have a `this` keyword and whatever its pointing to thats what it uses

Remember: **you're not allowed to call `new` on an arrow function**

Arrow functions are not hard bound functions, **it is a function that doesn't define a `this` keyword**

Resolving `this` in arrow functions

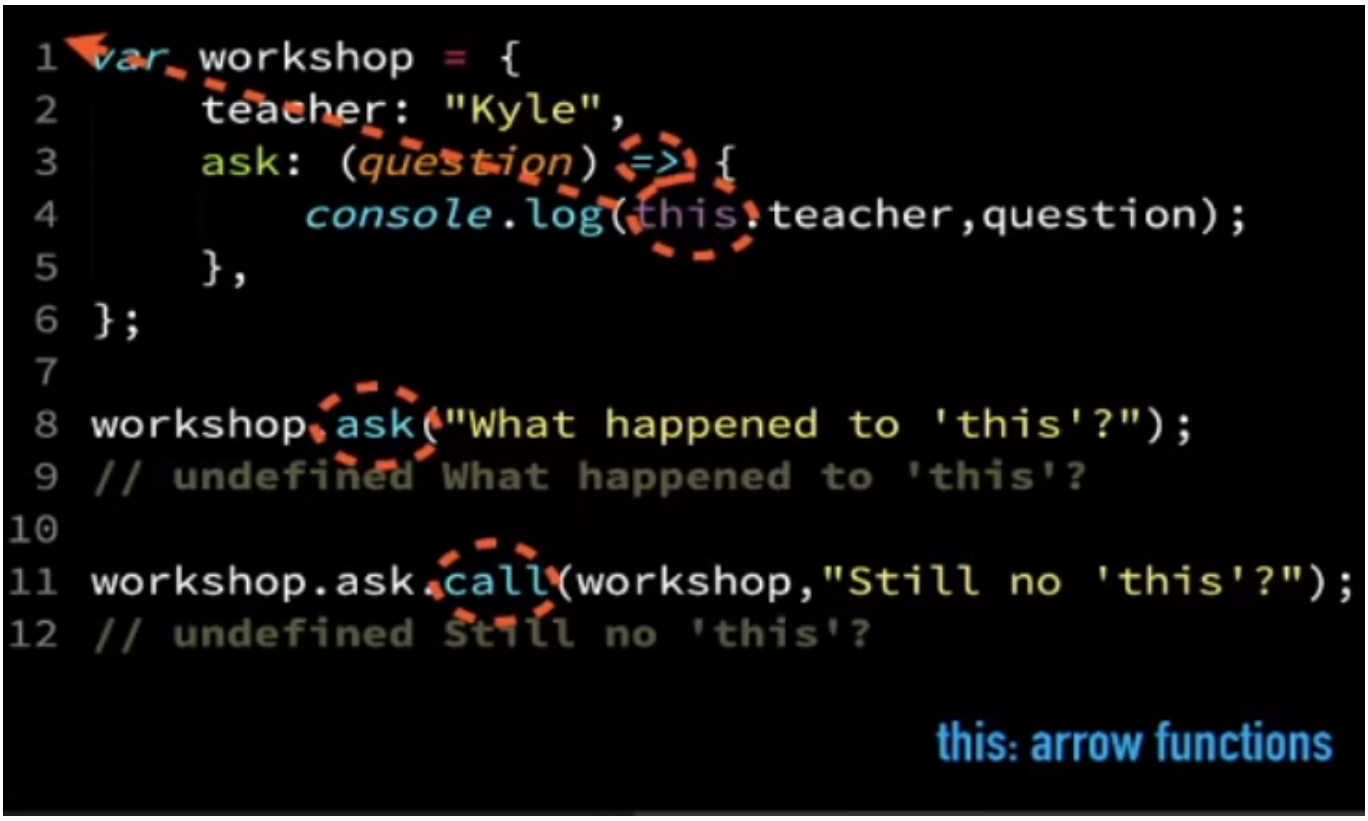
```
var workshop = {
  teacher: "Kyle",
  ask: (question) => {
    console.log(this.teacher,question);
  },
};

workshop.ask("What happened to 'this'?");
// undefined What happened to 'this'

workshop.ask.call(workshop, "Still no this?");
// undefined Still no this?
```

We tend to think that curly braces must be scopes... They're blocks, they're function bodies, they must be scopes...

But the arrow function that `ask` is referencing... what is the parent lexical scope from which that arrow function will go up one level to resolve the `this` keyword? -> **global scope**

Just because there are curly braces encapsulating the workshop object **DOES NOT MAKE IT** a scope! **Objects are NOT scopes**

There are 2 scopes in our code snippet! The scope of the `ask` function, which an arrow function, and the global scope



The only time you should ever use an arrow function is when **you're going to benefit from lexical `this` behavior**

you must remember that the `this` keyword **never ever ever** under any circumstances points at the function itself, **it points at a context**

Also properties arent scoped, properties aren't lexical identifiers, they're members of an object value... That's why objects don't have a scope, and hence our code snippet fails

The only thing you ever need to do to under the `this` keyword is look for the call site of a function that defines the `this` keyword **AND** ask those 4rules