academeez

this

Execution Context

this

- **this** in Javascript works very differently from most of the programming languages
- **this** represents the execution context we are in
- Execution context means from which object did our function run from.
- **this** is everywhere not only in **class**
- **this** is dynamic and can change, I can call the same function multiple time where every time there is a different context

this - global

- **this** is available everywhere even when it's not wrapped at all in function or class
- When it's not wrapped, **this** will be equal to **window** in the browser or **module.exports** in node

```
// in the browser
console.log(this === window)

// in node
console.log(this === module.exports);
```

this - function

- **this** - will change in function according to whomever is running the function
- Without a certain context it will be set to the global or undefined in strict mode
- With a context it will be set to that context

```
function returnThis() {
    return this;
}

// in browser it will equal window
// in strict mode it will equal undefined
console.log(returnThis() === global);

// running the same function from an object
// this change to the object
const a = {returnThis};
console.log(a.returnThis() === a);
```

this - lambda functions

academeez

- In lambda function this has a fixed value
- The value is binded to the same value of this when the lambda is created

```
const returnThis = () => {
    return this;
}

// in browser it will equal window
console.log(returnThis() === module.exports);

// running the same function from an object
// this remains the same
const a = {returnThis};
console.log(a.returnThis() === module.exports);
```

this - in class

academeez

- Class in Javascript is a syntax sugar-coat around functions
- The behaviour is similar to functions but with use strict

```
class Person{
    returnThis() { return this; }
    returnThis2 = () => this;
}

// calling from instance
const me = new Person();
console.log(me.returnThis() === me); // true
console.log(me.returnThis2() === me); // true

// calling from the root context
const {returnThis, returnThis2} = me;
console.log(returnThis() === undefined); // true
console.log(returnThis2() === me); // true

// calling from an object
const a = {returnThis: me.returnThis, returnThis2: me.returnThis2}
console.log(a.returnThis() === a); // true
```

bind / apply / call

academeez

- **bind** - will return a function whose this is fixed
- **apply / call** - we can use to call a function while setting it's this
- They will not work on lambda function

```
function returnThis() {
    return this;
}

const a = {hello: 'world'};

// bind
console.log(returnThis.bind(a)() === a);

// call
console.log(returnThis.call(a) === a);

// apply
console.log(returnThis.apply(a) === a);
```

Summary

- this in Javascript is dynamic and is determined by the execution context - who is running the function

academeez

Thank You

Next Lesson: Promise