[13] This (BIND, CALL, APPLY)

In JavaScript, the keyword "this" is a special identifier that refers to the current object or context within which a function is executed. JavaScript provides three methods—bind, call, and apply—that allow you to explicitly control the value of "this" within a function.

https://www.youtube.com/watch?v=bS71_...

"this" in Regular Functions

In regular functions, the value of "this" depends on how the function is called. There are several rules that determine the value of "this":

- Global Context: In a function that's not part of an object or class method, "this" refers to the global object, which is window in a browser environment or global in Node.js.
- Object Methods: In a function that's a method of an object, "this" refers to the object on which the method was called.
- Constructor Functions: In a constructor function (created using the new keyword), "this" refers to the newly created object.

1. Global Context
    - Anything which is defined in the global scope can be accessed by `this`. By default, '`this`' always refers to the global object (so if you are writing code for a browser, `this` will typically be the `window` object).

```
---Example[1]:
var color = 'red';
console.log(this.color); //red
console.log(window.color); //red
console.log(this) // window
console.log(this===window) // True
```

2. Function Context
    - In a function, `this` will default to the global object, which is `window` in a browser.

```
---Example[1]:
var color = "red";
function printColor() {
var color = "blue";
console.log(this.color)
}
printColor(); // Red will be returned.
```

3. Object Context
    - `this` in an object points to the object always and works pretty much as you would expect it to.
    - The magic of objects is you can pass objects (and their methods — which is what you call a function when it is inside an object) as parameters to functions. So here we are passing in `obj.getColor()` into `console.log`.

```
---Example[1]:
var color = "red";
const obj = {
color: 'orange',
getColor : function() {
console.log(this.color)
}
}
console.log(obj.getColor()); // Orange will be returned
```

---

call()

The call method allows you to invoke a function with a specific value of "this" and pass arguments individually. It immediately executes the function.

```
---Example[1]:
const person1 = { name: 'Alice' };
const person2 = { name: 'Bob' };
function sayHello() {
console.log(`Hello, my name is ${this.name}`);
}
sayHello.call(person1); // Output: Hello, my name is Alice
sayHello.call(person2); // Output: Hello, my name is Bob
```

apply()

The apply method is similar to call, but it accepts arguments as an array. It also immediately executes the function.

```
---Example[1]:
const person1 = { name: 'Alice' };
const person2 = { name: 'Bob' };
function sayHello(greeting) {
console.log(`${greeting}, my name is ${this.name}`);
}
sayHello.apply(person1, ['Hi']); // Output: Hi, my name is Alice
sayHello.apply(person2, ['Hello']); // Output: Hello, my name is Bob
```

bind()

The bind method is used to create a new function with a specified value of "this" that is permanently fixed. It doesn't execute the function immediately but returns a new function that you can call later with the specified "this" value.

```
---Example[1]:
const person = {
name: 'John',
sayHello: function() {
console.log(`Hello, my name is ${this.name}`);
},
};
const greet = person.sayHello.bind(person);
greet(); // Output: Hello, my name is John
```