Lesson:

More on functions call, apply, bind

PW SKILLS

Topics

- Revisit Functions
- Function method - call()
- Function method - apply()
- Function method - bind()
- Interview Point

Revisit Functions

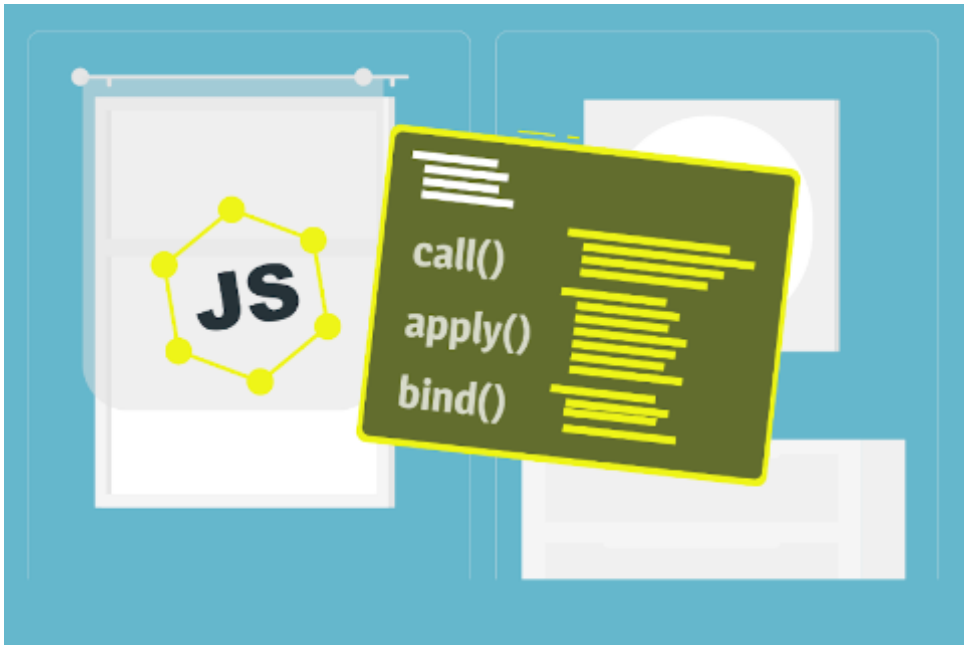
Like we have seen, in case of arrays and objects we can create them using **Array** and **Object** constructors.

```
const arr = new Array();
const object = new Object();
```

Similarly, In Javascript functions are also objects too, and so can create them using **Function** constructor.

```
const newFunction = new Function();

// EQUIVALENT TO
function newFunction(){}
```

If functions are also objects, then it can have methods too.



In this lecture, we will talk about some important methods of functions by which function work internally,

- **call()**
- **apply()**
- **bind()**

call () in JavaScript

The call method in JavaScript helps us to call a function and control what “this” keyword means inside it. It also lets us pass arguments one by one. It's useful when we want to borrow a function from one object and use it with another object.

In simple words, we can say it is used to invoke a function / method belonging to one object by another object.

Let’s take an example to understand this case:
Assume we have two objects named person1 and person2 with the following data in it-

```
let person1 = {
  firstName: "PW",
  lastName: "Skills",
  printFullName: function () {
    console.log(this.firstName + " " + this.lastName);
   }
 };

let person2 = {
  firstName: "Physics",
  lastName: "Wallah",
};

person1.printFullName();

// output
PW Skills
```

In the above example, inside the person1 there is method to log the full name, where it is using the this keyword to get the firstName and lastName of the person from the object as the this keyword over there is referring to the person1 object.

Now we also have person2 and we want to log the full name of it, so creating a method inside in it will not be a good idea as we already have a method inside the person1 to do the same task, so what we can do is to use that method (borrow the method).
This is known as the **function borrowing** in javascript to which we can do using the call method.

call method syntax

Before moving forward let's understand the syntax of call method to have better understanding of it.

```
functionName.call(thisArg, arg1, arg2, ...);
```

functionName: It refers to the function on that we want to use the call method.

call: It is the keyword that invokes the call method.

thisArg: It is the value that will be assigned to the this keyword inside the function when it will be executed.

arg1, arg2, ...: These are the optional arguments that we can pass individually to the function separated by the comma.

Example 1: Function borrowing with call method

```
let person1 = {
  firstName: "PW",
  lastName: "Skills",
  printFullName: function () {
    console.log(this.firstName + " " + this.lastName);
  }
};

let person2 = {
  firstName: "Alakh",
  lastName: "Pandey",
};

person1.printFullName.call(person2);


// Output
Alakh Pandey
```

Now we can see that we are able to print the full name of person2 also by using the method of the person1.

As discussed in the syntax we can see person1.printFullName is the function, call is to invoke call method and person2 is the value which will be assigned to the this keyword thats why it is printing the 'Alakh Pandey' instead of 'PW Skills' as the this keyword is now pointing to the person2 instead of person1.

Example 2: Removing the printFullName method from person1

As we had seen that person1 and person2 both needs the method printFullName, so lets make it an independent function and use the call method to print the full name from both the objects.

```
function printFullName() {
  console.log(this.firstName + " " + this.lastName);
}

let person1 = {
  firstName: "PW",
  lastName: "Skills",
};
```

```
let person2 = {
  firstName: "Alakh",
  lastName: "Pandey",
};

printFullName.call(person1);
printFullName.call(person2);

// Output
PW Skills
Alakh Pandey
```

Now in this example, printFullName is the function, call is used to invoke the call method and the person1 and person2 is the value which will be assigned to the this keyword. So for first function call this keyword will be referring to the person1 and for the second one it is referring to the person2.

Example 3: Passing the arguments to the function while using call method.

```
function printFullName(personName)  {
  console.log("Welcome " + personName + " to " + this.firstName + " " + this.lastName);
 };

let person1 = {
  firstName: "PW",
  lastName: "Skills",
}

let person2 = {
  firstName: "Physics",
  lastName: "Wallah",
};


printFullName.call(person1, "Vishwa Mohan");
printFullName.call(person2, "Pratik Maheswari");


 // Output

Welcome Vishwa Mohan to PW Skills
Welcome Pratik Maheswari to Physics Wallah
```

So in the above example we can clearly see that we can use comma to give arguments to which we want to pass to the function as per our need. In the above example we had passed the person name while using the call method after mentioning the value for this keyword.

apply() in JavaScript

apply method in completely similar to that of the call method with a single difference to it, which is in the terms of how arguments are passed to the function.

apply method syntax

Before moving forward let's understand the syntax of apply method to have better understanding of it.

```
functionName.apply(thisArg, [argsArray]);
```

functionName: It refers to the function on that we want to use the apply method.

apply: It is the keyword that invokes the call method.

thisArg: It is the value that will be assigned to the this keyword inside the function when it will be executed.

argsArray: An array-like object or an actual array containing the arguments to be passed to the function.

Now lets again use those three examples of call method and use the apply method on them to understand the working of the apply method one by one.

Example 1: Function borrowing with call method

```
let person1 = {
  firstName: "PW",
  lastName: "Skills",
  printFullName: function () {
    console.log(this.firstName + " " + this.lastName);
   }
 };

let person2 = {
  firstName: "Physics",
  lastName: "Wallah",
 };
person1.printFullName.apply(person2);


// Output
Physics Wallah
```

We are getting the same result as we were getting with the call method.

Example 2: Removing the printFullName method from person1

```
function printFullName() {
  console.log(this.firstName + " " + this.lastName);
};

let person1 = {
  firstName: "PW",
   lastName: "Skills",
};

let person2 = {
  firstName: "Alakh",
  lastName: "Pandey",
};

printFullName.apply(person1);
printFullName.apply(person2);

//Output
PW Skills
Alakh Pandey
```

We are getting the same result as we were getting with the call method.

Example 3: Passing the arguments to the function while using apply method.

```
function printFullName(personName)  {
  console.log("Welcome " + personName + " to " + this.firstName + " " + this.lastName);
 };

let person1 = {
  firstName: "PW",
  lastName: "Skills",
}

let person2 = {
  firstName: "Physics",
  lastName: "Wallah",
};

printFullName.apply(person1, ["Vishwa Mohan"]);
printFullName.apply(person2, ["Pratik Maheswari"]);


 // Output

Welcome Vishwa Mohan to PW Skills
Welcome Pratik Maheswari to Physics Wallah
```

Note: apply method is quite useful when we have an array like object or an actual array and we want to pass its elements as individual argument to a function.

Example 4: Working on array of data with apply method

```
let numbers = [10, 2, 60, 50, 32, 29, 6];
let maxNumber =  Math.max.apply(null, numbers);
console.log(maxNumber);

// Output
// 60
```

Here we are using math's max method to find the number having the maximum value in the array.
In this example, Math.max is the function, apply is to invoke the apply method, null is there because we do not need to pass any value to this, and numbers array as the argument which will be passed. So apply is quite useful while passing the array as an argument.

bind() in JavaScript

It is used to create a new function that when invoked has its 'this' keyword set to a specific value. It allows you to explicitly bind the context (this keyword) of a function, regardless of how it is called later.
bind method is also similar to call method, but it does not invoke the function instantly like the call method, instead it stores the function in the variable to which we can invoke when we want.

We can also pass the arguments one by one by separating them using the comma similar to the call method.

bind method syntax

Before moving forward let's understand the syntax of bind method to have better understanding of it.

```
functionName.bind(thisArg, arg1, arg2, ...);
```

functionName: It refers to the function on that we want to use the bind method.

bind: It is the keyword that invokes the bind method.

thisArg: It is the value that will be assigned to the this keyword inside the function when it will be executed.

arg1, arg2, ...: These are the optional arguments that we can pass individually to the function separated by the comma.

Now lets again use those three examples of call method and use the bind method on them to understand the working of the bind method one by one.

Example 1: Function borrowing with bind method

```
let person1 = {
  firstName: "PW",
  lastName: "Skills",
  printFullName: function () {
    console.log(this.firstName + " " + this.lastName);
   }
 };

let person2 = {
  firstName: "Physics",
  lastName: "Wallah",
};


const fullName = person1.printFullName.bind(person2);
console.log(fullName);
fullName();


 // Output
[Function: bound printFullName]
Physics Wallah
```

Now in the above example, if we will not store our result inside the fullName variable then there will not be any output as bind method does not invoke the function directly over there, it needs to be invoked from the variable in which we had stored wherever we want it.

Also we can see that when we had logged the fullName we are getting the function and after invoking it in next line we are able to get the same output as we were getting in the call and apply method.

Example 2: Removing the printFullName method from person1

```
function printFullName() {
  console.log(this.firstName + " " + this.lastName);
};

let person1 = {
  firstName: "PW",
   lastName: "Skills",
};

let person2 = {
  firstName: "Alakh",
  lastName: "Pandey",
};

const firstPersonFullName = printFullName.bind(person1);
const secondPersonFullName = printFullName.bind(person2);

console.log(firstPersonFullName);
console.log(secondPersonFullName);


firstPersonFullName();
secondPersonFullName();

//Output
[Function: bound printFullName]
[Function: bound printFullName]
PW Skills
Alakh Pandey
```

Now we can see that both the firstPersonFullName and secondPersonFullName is displaying that both of them is bound to the printFullName with their this value, thats why we are getting two different output as per the values of person1 and person2. Once we are invoking those two we are getting the same result as we were getting for the call and apply method

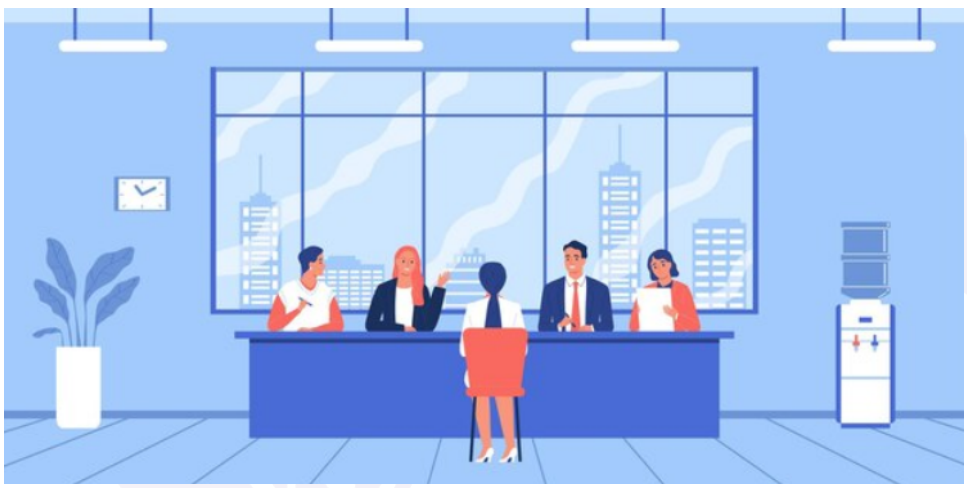
Example 3: Passing the arguments to the function while using bind method.

```
function printFullName(personName)  {
  console.log("Welcome " + personName + " to " + this.firstName + " " + this.lastName);
 };

let person1 = {
  firstName: "PW",
  lastName: "Skills",
}

let person2 = {
  firstName: "Physics",
  lastName: "Wallah",
};
```

```
const firstPersonFullName =  printFullName.bind(person1, ["Vishwa Mohan"]);
const secondPersonFullName = printFullName.bind(person2, ["Pratik Maheswari"]);

console.log(firstPersonFullName);
console.log(secondPersonFullName);

firstPersonFullName();
secondPersonFullName();

// Output
[Function: bound printFullName]
[Function: bound printFullName]
Welcome Vishwa Mohan to PW Skills
Welcome Pratik Maheswari to Physics Wallah
```

As we can see that we are getting the same result as we were getting in the call method. We had passed the argument by apply comma after passing the value to which we want to bind, we can pass as many arguments simply by separating them by using comma.

Interview Point



Q. Explain the difference between call, apply, and bind.

Ans. The main difference between call(), apply(), and bind() lies in how they handle function invocation and the “this” value. The call() function immediately invokes the function it is called on and allows you to pass arguments individually. The apply() function is similar but expects arguments to be passed as an array. Both call() and apply() execute the function immediately.

On the other hand, the bind() function returns a new function with the same body as the original function but with a fixed “this” value and, optionally, some initial arguments. Unlike call() and apply(), bind() doesn't invoke the function immediately. Instead, it creates a new function that can be invoked at a later time, often used to preserve the context in event handlers or callbacks.

Q. What will be the output of the following code

```
const person = { name: 'Subham' };

function sayHi(age) {
  return `${this.name} is ${age} years`;
}

console.log(sayHi.call(person, 19));
console.log(sayHi.bind(person, 19));
```

Ans.
OUTPUT

```
Subham is 19 years
ƒ sayHi(age) {
  return `${this.name} is ${age} years`;
}
```