Function's

A function is a block of reusable code designed to perform a specific task or a set of tasks. Functions allow you to break down a program into smaller, manageable pieces, making the code more modular and easier to maintain.

Here's a basic structure of a function in JavaScript:

```
function functionName(parameters) {
  // code to be executed
  return result; // optional, specifies the value to be returned
}
```

Let's break down the components:

- function: Keyword indicating the start of the function declaration.

- functionName: The name given to the function, which you can use to call the function later.

- parameters: Input values that the function accepts. These are optional; a function can have zero or more parameters.

- { ... }: The curly braces contain the code block or body of the function.

- return: Keyword used to specify the value that the function should return. This part is optional, and a function may not always return a value.

Here's a function Declarations simple example:

```
function addNumbers(a, b) {
  var sum = a + b;
  return sum;
}

var result = addNumbers(5, 3);
console.log(result); // Output: 8
```

In this example, the `addNumbers` function takes two parameters (`a` and `b`), adds them together, and returns the result. When called with `addNumbers(5, 3)`, it returns 8, which is then printed to the console.

JavaScript also supports anonymous functions (functions without a name) and function expressions, providing flexibility in how you define and use functions in your code.

function expression

A function expression in JavaScript is a way to define a function using an expression rather than a declaration. In a function expression, you create a function as part of an assignment or within an expression. The basic syntax looks like this:

```
var functionName = function(parameters) {
  // code to be executed
  return result; // optional
};
```

Let's break down the components:

- var functionName: The variable declaration where you assign the function to a variable.

- function(parameters): The actual function expression, with optional parameters.

- { ... }: The curly braces contain the code block or body of the function.

- return: An optional keyword used to specify the value that the function should return.

Here's a simple example:

```
var addNumbers = function(a, b) {
  var sum = a + b;
  return sum;
};

var result = addNumbers(5, 3);
console.log(result); // Output: 8
```

IIFE stands for "Immediately Invoked Function Expression." It is a design pattern in JavaScript where you define and execute a function immediately after its creation. This pattern is often used to create a private scope for variables, preventing them from polluting the global scope.

The basic syntax for an IIFE looks like this:

```
(function() {
  // code to be executed immediately
})();
```

Here's a breakdown of the components:

- The function is enclosed within parentheses `(function() {...})`. This is necessary to tell the JavaScript parser that the function is going to be part of an expression.

- The opening parenthesis before `function` and the closing parenthesis at the end `})();` create the function expression and invoke it immediately.

This pattern is useful for encapsulating code and preventing variables defined inside the function from interfering with variables in the global scope. For example:

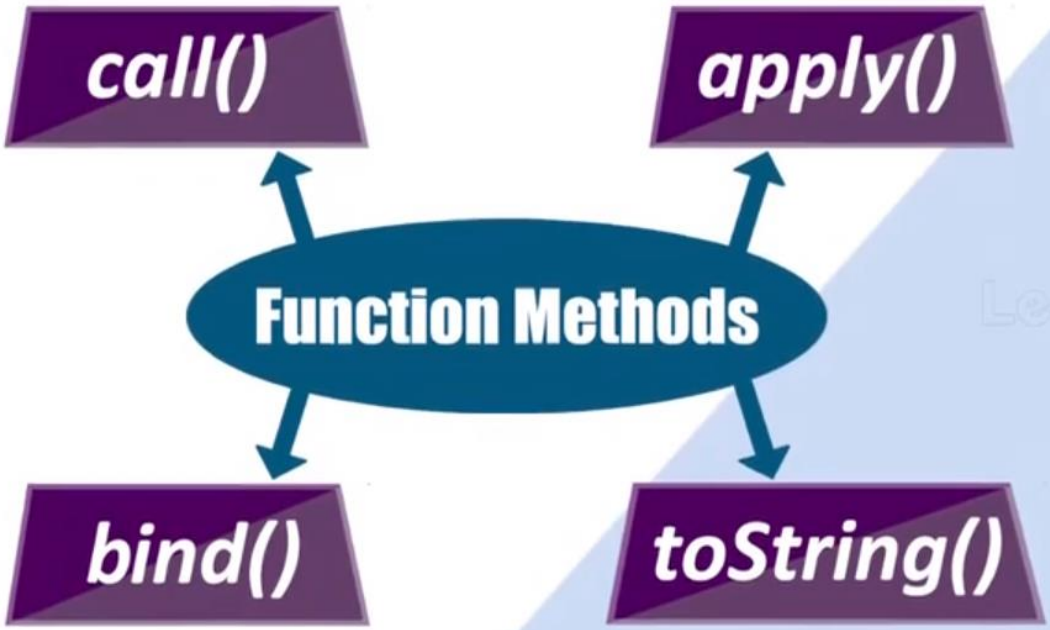
```
(function() {
  var privateVariable = "I am private";

  // The privateVariable is only accessible within this function scope

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

Function Methods

JavaScript mein, har ek function ek object hota hai, aur is object ke sath kuch methods judi hoti hain.



Method	Description
call()	It is used to call a function contains this value and an argument list.
apply()	It is used to call a function contains this value and a single array of arguments.
bind()	It is used to create a new function.
toString()	It returns the result in a form of a string.

1. call(): `call` method ka istemal kisi function ko ek specific value ke sath call karne ke liye hota hai. Yeh method function ko ek nirdharit value ke sath execute karta hai.

It is use to call a function contains this value and an argument list.

Example →

```
function greet(name) {
  console.log(`Hello, ${name}!`);
}
greet.call(null, "John");
```



```
function Emp(id,name){
        this.id = id;
        this.name = name;
}

function PermanentEmp(id,name){
    Emp.call(this,id,name);
}
function TemporaryEmp(id,name){
    Emp.call(this,id,name);
}

let PEmp = new PermanentEmp(1,"Ganesh dutt");

let TEmp = new TemporaryEmp(2,"motu");

document.writeln(PEmp.id + " " + PEmp.name);
document.writeln("<br>");

document.writeln(TEmp.id + " " + TEmp.name);
```

2. apply():

`apply` method bhi ek function ko call karne ke liye hota hai, lekin yahan parameters ek array ke roop mein diye jaate hain.

The javaScript function apply() method is used to call a function contains this value and an argument contains elements of an array. Unlike call() methos , it contains the single array of arguments.

Syntax

Function.apply(thisArg,[array])

Parameter:

thisArg – It is optional . The this value is given for the call to a function.

array – It is optional. It is an array-like object.

Example ->

```
function greet(name, age) {
  console.log(`Hello, ${name}! You are ${age} years old.`);
}
greet.apply(null, ["John", 25]);
```

```
var arr = [5,7,3,2];
        var max = Math.max.apply(this ,arr);
        var min = Math.min.apply(this ,arr);

        document.writeln(max);
        document.writeln(min);
```

3. bind():

`bind` method ek naya function create karta hai, jo original function ko ek specific context ke sath call karta hai. Yeh method ek function ka copy banata hai, jismein this value set hoti hai.

The javaScript function bind() method is used to create a new function. When a function is called, it has its own this keyword set to the provided value, with a given sequence of arguments.

Syntax:

function.bind(thisArg[,arg1[,arg2[, ...]]])

Parameter :

thisArg – The this value passed to the target function .

arg1 , arg2,...,argn – It represents the arguments for the function.

Example ->

```
const greetJohn = greet.bind(null, "John");
greetJohn(25);
```

```
var website = {
        name: "JavaScript",
        getName:function(){
            return this.name;
        }
    }
    var website2 = {
        name: "React",

    }

    var unboundGetName = website.getName;
      var boundGetName = unboundGetName.bind(website2);
    document.writeln(boundGetName());
```

4. toString():

`toString` method ka istemal function ko string mein convert karne ke liye hota hai.

The javaScript function toString() method retirns a string. Here , string represents the source code of the function.

Syntax:

function.toString()

Example →

```
function myFunction() {
  console.log("Hello, World!");
}

console.log(myFunction.toString());
```

```
function add(num1, num2){
 return num1+num2;
   }

   // document.writeln(add(10,40));
   document.writeln(add(10,40).toString());
   document.writeln(typeof add(10,40).toString());
```

5. length: `length` property function ke parameters ki sankhya ko represent karti hai.

```
javascript

function exampleFunction(param1, param2, param3) {
  // Some code
}


console.log(exampleFunction.length); // Output: 3
```

Yeh kuch common methods hain jo JavaScript functions ke sath judi hui hain. Har ek method ka apna apna istemal hota hai aur specific use cases ke liye design kiye gaye hote hain.