**What is diff between event, bind**

[**http://www.adequatelygood.com/JavaScript-Scoping-and-Hoisting.html**](http://www.adequatelygood.com/JavaScript-Scoping-and-Hoisting.html) **(example site )**

You can use a function **declaration** or a function **expression**.----

[**http://www.w3schools.com/Js/js\_function\_definition.asp**](http://www.w3schools.com/Js/js_function_definition.asp)

myFunction(2,3); // it return output

function myFunction(a, b) {  
    return a \* b;  
}

function declaration is hoisted . but function expression is not hoisted .

Semicolons are used to separate executable JavaScript statements.  
Since a function **declaration** is not an executable statement, it is not common to end it with a semicolon.

A function expression can be stored in a variable: function expression is not hoisted .

fnName(2,3);// not return output;

Var fnName = function (a, b) {  
    return a \* b;  
};

fnName(2,3);// it return output;

After a function expression has been stored in a variable, the variable can be used as a function:

**JavaScript Scope and Scope Chains----**

<https://kamlouis.wordpress.com/2016/04/02/javascript-scope-and-scope-chaining/>

We should understand that the JavaScript engine views all variables as two separate statements. A variable declaration and a variable assignment. For example: **var b = 10;**would be viewed as the variable declaration**var b;**and the variable assignment**b = 10;**

All variables form part of the global or local scope but no matter where variables are declared within a particular scope their variable declaration (not assignment) will be moved to the top of their scope, this is known as hoisting.

[Variable scope and Scope chains](https://kamlouis.wordpress.com/2016/04/02/javascript-scope-and-scope-chaining/#scope)

* [Shadowing](https://kamlouis.wordpress.com/2016/04/02/javascript-scope-and-scope-chaining/#shadowing)--[Shadowing](https://en.wikipedia.org/wiki/Variable_shadowing) (also known as variable shadowing) is a term used when a variable declared in a scope has the same name as a variable declared in an outer scope. It is perfectly valid to define two different variables, in different scopes, with the same name. As they are located in different scopes they do not affect each other but if they are located in the same scope they will overwrite each other
* [**Hoisting**](https://kamlouis.wordpress.com/2016/04/02/javascript-scope-and-scope-chaining/#hoisting)

JavaScript Variable Scope and Hoisting Explained----

<http://javascriptissexy.com/javascript-variable-scope-and-hoisting-explained/>

local variables---JavaScript has function-level scope. Variables declared within a function are local variables and are only accessible within that function or by functions inside that function.

var name = "Richard";

function showName () {

var name = "Jack"; // local variable; only accessible in this showName function​

console.log (name); // Jack​

}

console.log (name); // Richard: the global variable

Create Chainable (Cascading) Methods for Expressiveness

1. **Local Variables Have Priority Over Global Variables in Functions—**

If you declare a global variable and a local variable with the same name, the local variable will have priority when you attempt to use the variable inside a function (local scope):

**var name = "Paul";**

**​function users () {**

**// Here, the name variable is local and it takes precedence over the same name variable in the global scope​**

**​var name = "Jack";**

**​// The search for name starts right here inside the function before it attempts to look outside the function in the global scope​**

**console.log (name);**

**}**

**users (); // Jack**

1. **If You Don’t Declare Your Local Variables, Trouble is Nigh—**

Global Variables--Any variable declared or initialized outside a function is a global variable, and it is therefore available to the entire application. For example:// To declare a global variable, you could do any of the following:​

​var myName = "Richard";

​// or even​

firstName = "Richard";

​// or ​

​var name; //​

name;

​If a variable is initialized (assigned a value) without first being declared with the var keyword, it is automatically added to the global context and it is thus a global variable:

function showAge () {

// Age is a global variable because it was not declared with the var keyword inside this function​

age = 90;

console.log(age);// ​

}

showAge (); // 90​

​// Age is in the global context, so it is available here, too​

console.log(age); // 90

**Variable Hoisting---**It is important to know that only variable declarations are hoisted to the top, not variable initialization or assignments (when the variable is assigned a value).

Function Declaration Overrides Variable Declaration When Hoisted----

Both **function declaration and variable** declarations are hoisted to the top of the containing scope. And function declaration takes precedence over variable declarations (but not over variable assignment).

As is noted above, variable assignment is not hoisted, and neither is function assignment.

**Chaining Methods**, also known as **Cascading**, refers to repeatedly calling one method after another on an object, in one continuous line of code.

To use cascading, we have to return this (the object we want subsequent methods to operate on) in each method.

Operator precedence: it means which operator function gets called first. Function are called in order of precedence. If we have more than one operator the javascript call operator which has higher precendece. In order.

[**https://www.quora.com/What-are-the-advantages-of-closures**](https://www.quora.com/What-are-the-advantages-of-closures)

[**http://www.masterjavascript.io/blog/2016/04/24/understanding-closures/**](http://www.masterjavascript.io/blog/2016/04/24/understanding-closures/)

[**https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures)

What is lexical scope-----

<http://javascriptissexy.com/understand-javascript-closures-with-ease/>

## JavaScript’s this Keyword Basics

The this reference ALWAYS refers to (and holds the value of) an object—a singular object—and it is usually used inside a function or a method, although it can be used outside a function in the global scope. Note that when we use **strict mode**, this holds the value of undefined in global functions and in anonymous functions that are not bound to any object.

## The use of this in the global scope—

## Fix this inside closure—

## What is a Callback or Higher-order Function?

<http://javascriptissexy.com/understand-javascript-callback-functions-and-use-them/>

## How Callback Functions Work?

**Use Named OR Anonymous Functions as Callbacks---** Another popular pattern is to declare

## Basic Principles when Implementing Callback Functions---

**Use Named OR Anonymous Functions as Callbacks----**

**Problem When Using Methods With The** this **Object as Callbacks----**

<http://javascriptissexy.com/understand-javascript-callback-functions-and-use-them/>

<http://knowledgehills.com/javascript/callback-functions.htm>

**Use the Call or Apply Function To Preserve** this in case of callbacks--------

Understand JavaScript Closures With Ease----

<http://javascriptissexy.com/understand-javascript-closures-with-ease/>

**WHAT ABOUT ASYNCHRIOUS CALLBACKS—It means more than one at a time.**

Q- what is difference between callback and promice?

Both are same or difference

JavaScript Objects in Detail—

<http://javascriptissexy.com/javascript-objects-in-detail/>

**JSON AND OBJECT ----**

**Json – javascript object notation**

**In case of JSON property name is inside the {“ property name”:”value”}**

**But in object like as -- { property name:”value”}**

var **mango** = {color: "yellow",shape:”circle”} // this is object

Now convert object into JSON

Console.log(JSON.stringfy(**mango**));

// {“color”: "yellow",”shape”:”circle”} // this is JSON

Now convert JSON inot object literal

Var jsonData= {“color”: "yellow",”shape”:”circle”} ;

Console.log(JSON.parse(jsonData)); // {color: "yellow",shape:”circle”} // this is object

**Function and Object ---**

**Function is the object in javascrit so we can add property and function inside the functiuon like as object.**

**Function greet(){**

**Console.log(‘hi’);}**

**Greet.language=’English’;**

**Console.log(greet.language); /// English**

**Son function is a object**

**Muatate--- we can change object value**

**.inmutable --- we can not change the value**

**Object Constructor-----**

<http://javascriptissexy.com/javascript-objects-in-detail/>

<https://www.youtube.com/watch?v=2DvQUUbNsIo> (best for oops)

<https://www.youtube.com/watch?v=yiyXTc90aBc> ()

<https://www.youtube.com/watch?v=xizFJHKHdHw>

<https://www.youtube.com/watch?v=c0mLRpw-9rI> (call,aply, bind)

<http://www.w3schools.com/jsref/jsref_obj_string.asp>

<http://www.w3schools.com/jsref/jsref_obj_number.asp>

<https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Object/prototype>

<http://code.tutsplus.com/tutorials/prototypes-in-javascript--net-24949>

<http://phrogz.net/JS/Classes/OOPinJS.html>

## 12-How to check that which object belongs to which cunstructor function in javascript?

## <script>

## function Rabbit() { };

## function Rabbit2() { };

## var rabbit1 = new Rabbit();

## var rabbit2 = new Rabbit2();

## var hh=rabbit1.constructor;

## var hh2=rabbit2.constructor;

## alert(hh) // function Rabbit() { }

## alert(hh2) // function Rabbit2() { }

## </script>

## hasOwnProperty---All objects have hasOwnProperty method which allows to check if a property belongs to the object or its prototype. IN this case we check that which property belong to which object. And it return true or false.

function Rabbit(name) {

this.name = name

}

Rabbit.prototype = { eats: true }

var rabbit = new Rabbit('John')

var rabbit2 = new Rabbit('John')

alert( rabbit.hasOwnProperty('eats') ) // false, in prototype

alert(rabbit.hasOwnProperty('name') ) // true, in object

**instanceof ----** it is use to find that cretated object belongs to which constructor function. It return true/false.

Document.write(rabbit2 instanceof Rabbit) // return true.

isPrototypeOf----

<http://javascriptissexy.com/javascript-prototype-in-plain-detailed-language/>

<http://javascriptissexy.com/oop-in-javascript-what-you-need-to-know/>

<http://javascriptissexy.com/javascript-apply-call-and-bind-methods-are-essential-for-javascript-professionals/>

…………………………………………………………………………………………………………………………………………………………………….

First let us see few terminologies that we use in object-oriented programming.

**Class**: Defines the characteristics of the Object.  
**Constructor**: A method called at the moment of instantiation.

<http://toranbillups.com/blog/archive/2013/05/15/Basic-javascript-inheritance-and-polymorphism/>

*anakin.hasOwnProperty('skill')*

*anakin.constructor.prototype.hasOwnProperty('skill')*  
**Object**: An Instance of a Class.  
**Method**: An Object capability as walk.  
**Property**: An Object characteristic, such as color  
**Inheritance**: A Class can inherit characteristics from another Class.

**Overrides------**

[**http://stackoverflow.com/questions/15497259/overriding-methods-in-javascript**](http://stackoverflow.com/questions/15497259/overriding-methods-in-javascript)

**what is the difference between overriding methods with prototypes and without prototypes.--**

**Example 1:**

function Animal() {

this.sleep = function () {

alert("animal sleeping");

};

this.eat = function () {

alert("animal eating"); //// IT **Overrides eat(); method**

};

}

function Dog() {

this.eat = function () {

alert("Dog eating");

};

}

Dog.prototype = new Animal(); // set up inheritance relation

var dog = new Dog;

dog.eat();

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**Example 2:**

**function Animal() {**

**// code here**

**}**

**function Dog() {**

**//code here**

**}**

**Animal.prototype.sleep = function () { // add new method property sleep() using prototype in Animal function.**

**alert("animal sleeping");**

**};**

**Animal.prototype.eat = function () { // add new method property eat() using prototype in Animal function.**

**alert("animal eating");**

**};**

**Dog.prototype = new Animal();//set up inheritance relation to use method property of Animal cunstractor function inside the Dog function.**

**Dog.prototype.eat = function () { // it is concept of override eat()**

**alert("Dog eating");**

**};**

**var dog = new Dog;**

**var animal = new Animal();**

**animal.eat(); // Animal eating**

**dog.eat(); // Dog eating**

[**http://javascript.crockford.com/private.html**](http://javascript.crockford.com/private.html)

**Polymorphism**: Different Classes might define the same method or property.

(means we create different object by instance of same constrctor function )

Polymorphism in Object-Oriented Programming is the ability to create a variable, a function, or an object that has more than one form

The primary usage of Polymorphism in Object-Oriented Programming is the ability of objects belonging to different types to respond to methods, fields, or property calls of the same name, each one according to an appropriate type-specific behaviour.

<http://toranbillups.com/blog/archive/2013/05/15/Basic-javascript-inheritance-and-polymorphism/>

JavaScript Encapsulation using Anonymous Functions

<https://www.codeproject.com/Articles/757949/JavaScript-Encapsulation-using-Anonymous-Functions>

## Creating a Constructor Function

<https://www.codeproject.com/Articles/757949/JavaScript-Encapsulation-using-Anonymous-Functions>

<http://aboutcode.net/2011/10/04/efficient-encapsulation-of-javascript-objects.html>

**Abstraction**: The conjunction of complex inheritance, methods, properties of an Object must be able to simulate a reality model.

<http://www.guru99.com/java-data-abstraction.html>

<http://whatis.techtarget.com/definition/abstraction>

<http://ramanshankar.blogspot.in/2016/01/javascript-abstraction-in-javascript.html>

<http://www.techumber.com/javascript-object-oriented-programming-tutorial/>

**interface----**

## The Class in JavaScript—

<https://www.youtube.com/watch?v=coIsvOMYEi0>

🡪>Create external class file in javascript

<https://www.youtube.com/watch?v=YbSSvTTBc6E>

In JavaScript we can use function as a class and name of function is the class name. means creating the function is known as class.

JavaScript does not contains class statement. JavaScript is a prototype-based language. JavaScript uses functions as classes.

//Define the class Car

function Car() { }

For creating instances of any class i.e. objects use new keyword. For example, in below code snippet we created two instances of class Car.

|  |
| --- |
| //Define the class Car  function Car() { }  var car1 = new Car();  var car2 = new Car(); |

.

## Polymorphism—

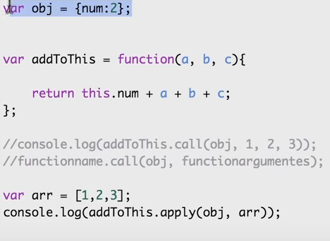
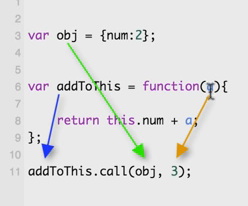
**Polymorphism** (objects can share the same interface)

The word Polymorphism in OOPs means having more than one form. In JavaScript a Object, Property, Method can have more than one form.

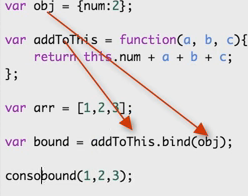
<http://www.techumber.com/2013/08/javascript-object-oriented-programming-tutorial.html>

**javaScript call ,apply and bind-----**

<https://www.youtube.com/watch?v=c0mLRpw-9rI>



Bind return function.



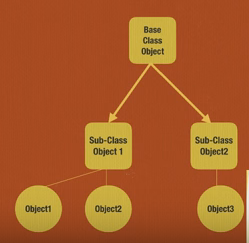
Javascript Closure tutorial ( Closures Explained )

**javaScript Method chaining tutorial ----**

<https://www.youtube.com/watch?v=5rwuKH-zdos>

**javascript prototype inheritance explained**

<https://www.youtube.com/watch?v=uIlj6_z_wL8>



The above screens shot are use to define that how we can create sub class from base class and also inherit the base class prototype method into subclass. And set sub class as a constructor. So that we can create new different -2 object.

Create sub class from base class and also inherit prototype from base class into subclass prototype…..

1-to create subclass from base class we have to use call() or apply() method into subclass. So that we can use property or method of base class into sub class.

2- To use prototype method property from base class into sub class we can use—

TechJob.prototype=Object.create(job.prototype);

//or

// TechJob.prototype = new job ();

//inheritPrototype(TechJob, job);

// base class

var app=[3,4];

var job= function(pay,pay2){

this.pay=pay;

this.pay2=pay2;

};

// prototype method into base class

job.prototype.print=function(){

return 'abhi'+''+this.pay;

};

// sub Class

var TechJob=function(title,pays){

job.apply(this,app); // this is use to inherit job function into TechJob function.

//job.call(this,3,4);

this.title=title;

this.pays=pays;

};

// thihs is use to inherit the prototype property from job method to TechJob method.

TechJob.prototype=Object.create(job.prototype);

3-- and in last create different -2 object from sub class-

// create object from sub class

var myFather = new TechJob

**Objects** can be thought of as the main actors in an application, or simply the main “things” or building blocks that do all the work. As you know by now, objects are everywhere in JavaScript since every component in JavaScript is an Object, including Functions, Strings, and Numbers. .

<http://javascriptissexy.com/oop-in-javascript-what-you-need-to-know/>

function. ----for that we can use call/apply method follow the following code.

<http://softwareengineering.stackexchange.com/questions/270645/what-are-the-benefits-of-using-new-over-closures>