

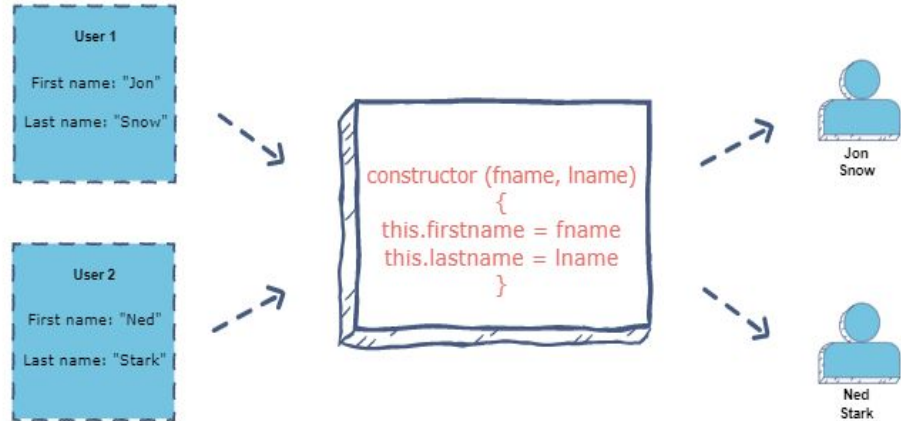



Constructor function

Key points to remember

What is the constructor function?


In simple words constructor function is used to create custom objects.





It is similar to creating any other function in JavaScript: it can have parameters, has a name, and is declared with the **function** keyword.


```
function Car(make,model,bodyStyle,year){  
    // some properties  
}
```



We can set properties to our custom object by defining them on **this**, which will point to the new object that is created

```
function Car(make,model,bodyStyle,year){  
  this.make=make;  
  this.model=model;  
  this.bodystyle=bodyStyle;  
  this.year=year;  
}
```

Note- just remember constructor function is just a function. We know that in a function we need to return something, but here we are not returning anything. So what's happening here? To understand this, let's look at how we create an instance of our example(creating a car object).



In order to create a object(car object) is like calling a normal function, but in constructor function we use the **new** keyword before the function name.

```
var honda= new Car("Honda","Accord","sedan","1999");
```



Now let's see our custom car object.

```
function Car(make,model,bodyStyle,year){  
  this.make=make;  
  this.model=model;  
  this.bodystyle=bodyStyle;  
  this.year=year;  
}  
  
var honda= new Car("Honda","Accord","sedan","1999");  
  
var kia = new Car("KIA","seltos","suv","2022")  
  
console.log(honda)  
console.log(kia)
```

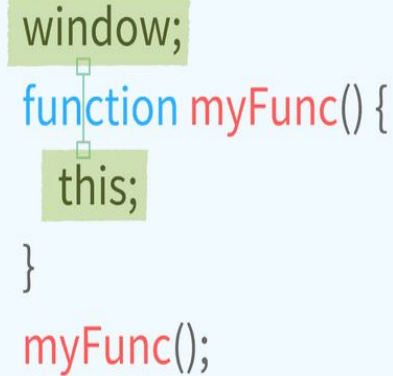
output

```
day1.js:60  
▶ Car {make: 'Honda', model: 'Accord', bodystyle: 'sedan', year: '1999'}  
day1.js:61  
▶ Car {make: 'KIA', model: 'seltos', bodystyle: 'suv', year: '2022'}
```



**“this” points to
global object in a
function invocation**

this in function invocation



```
window;  
function myFunc() {  
  this;  
}  
myFunc();
```

“this” points to
object that owns the
method in the
method invocation

this in method invocation

```
var myObject = {  
  myMethod: function() {  
    this;  
  }  
};  
myObject.myMethod();
```


The diagram illustrates the scope of the 'this' keyword. A green box highlights the 'this;' line inside the 'myMethod' function. A vertical line with a small square at the top and bottom connects this 'this;' to another green box highlighting 'myObject' in the 'myObject.myMethod();' line. This visualizes that 'this' refers to the object that owns the method being invoked, which is 'myObject'.



**“this” points to newly
created object in a
constructor
invocation**

this in constructor invocation

```
function Constructor() {  
  this;  
}  
var object = new Constructor();
```



The diagram illustrates the relationship between the `this` keyword and the object created during a constructor invocation. A green box highlights the `this;` statement inside the `Constructor` function. Another green box highlights the `object` variable in the `var object = new Constructor();` line. A vertical line with small squares at both ends connects the `this` box to the `object` box, indicating that `this` refers to the newly created object.



Generally “this” point to the owner object.

But we can use call, apply, bind to point this to different owner object.

Call and apply.

Call and apply are same almost but call takes arguments with commas, but apply takes arguments in a single array.

```
function myFunc() {  
  this;  
}  
var object = { name: 'My Obj' };  
myFunc.apply(object);  
myFunc.call(object);
```



Example for call and apply

```
var a= { name : "masai" };  
var b= { name : "school" };  
  
function printName(){  
    console.log(this.name);  
}  
  
printName.call(a);  
printName.apply(b)
```

output

```
masai  
school  
>
```

bind

In bind we need to manually call it as a function, because it returns a function

```
function myFunc() {  
  this;  
}  
var object = { name: 'My Obj' };  
var bound = myFunc.bind(object);  
bound();
```

Example for bind

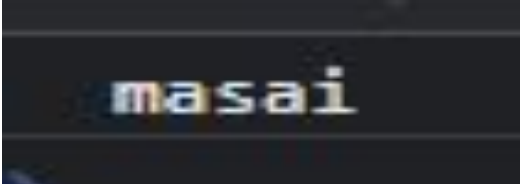
```
var a= { name : "masai" };

function printName(){
    console.log(this.name);
}

var res= printName.bind(a);

res();
```

output

A screenshot of a console log showing the output 'masai' in a monospace font.