**‘this’ in JavaScript**

In most cases, the value of *this*is determined by how a function is called. It can’t be set by assignment during execution, and it may be different each time the function is called. You can change *this*context through *.call()*, *.apply()*and *.bind().* Value of *this* is equal to the value of the object which invokes the function. *this* is not assigned a value until an object invokes the function where *this* is defined.

**Global Context**

When functions are executed in the global scope, value of *this* is *windows*object. This is because when we call a function in global scope by default they are invoked on the Window object. In strict mode, value of *this* in the global context will be undefined. Consider the below example:

The above function was not executed inside any other function or object hence, by default *myFunction*was called on the global object.

*myFunction()* is equivalent to w*indow.myFuntion()*

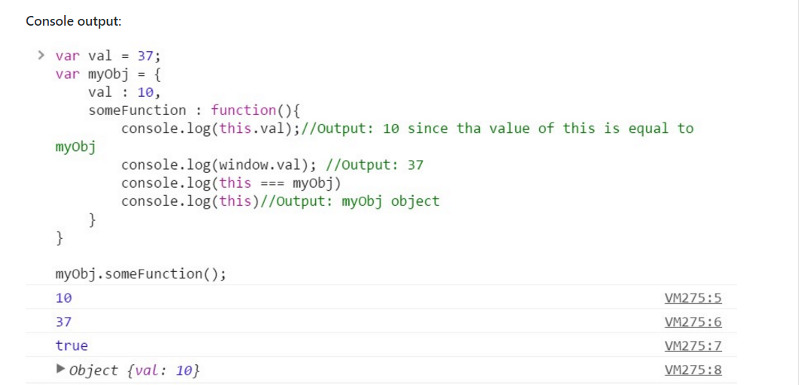
Hence the value of this is ‘*window*’. In strict mode, value of *this*would be *undefined*.



**As an object method**

When a function is called as a method of an object, it’s ***this***is set to the ***object the method is called on***.

Example:



In the above example, we have defined two *val*variables, one in the global scope with value**37**and another inside **myObj**object with value **10**.

When we call ***myObj.someFunction(),***as *someFunction()* was called by *myObj*object, value of ***this***inside *someFunction*will become equal to *myObj.*

Hence, inside *someFunction*when we do **console.log(this.val)**, it outputs **myObj’s val** variable value i.e 10.

When we do **window.val,** we get value 37 as global value for *val*is 37.

Another example:

In the above example, we have defined *someFunction*outside *myObj*object.

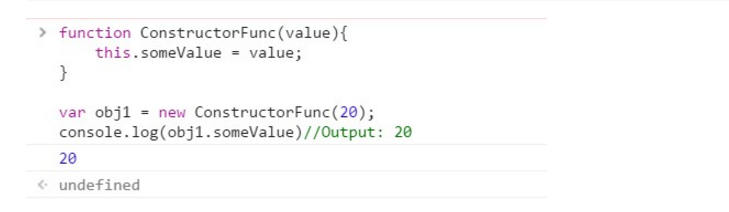
First we call *someFunction()* on *myObj*object, hence inside *someFunction*value of ***this***is equal to *myObj* object.

In second case, we have called *someFunction()* in the global scope which is same as writing *window.someFunction()*. Here, as *someFunction()*has been called on *window*object, value of *this*is equal to *window*object.

First we call *someFunction()* in the context of *myObj*. Later we called the function in the global context and the value of *this* inside the *someFunction()*changed accordingly. This shows that the value of *this* was determined dynamically on the basis of execution context.

**As a constructor**

When a function is used as a constructor (with the new keyword), its *this* is bound to the new object being constructed. To understand what are constructor function, please go through [this article](https://medium.com/@happymishra66/create-objects-in-javascript-10924cfa9fc7).



Here, we have defined ConstructorFunc to create new objects. When

**var obj1 = new ConstructorFunction(20)** is executed, value of *this*will be equal to the new object that is created.

**‘this’ in Immediately Invoked Function expression (IIFE)**

In *IIFE*, value of *this*is always equal to Window object. Let’s see an example:

**Console Output:**

**First example:**



In the first example, *IIFE outside any function*, value of *this*inside IIFE will be *Window* object.

**Second Example:**



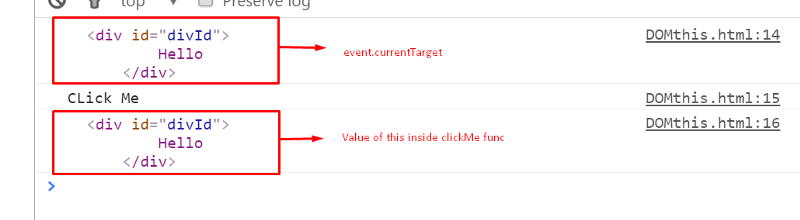
In the second example, when we have called *IIFE*inside *someFunc*still the value of *this* is *Window*object not *obj*object.

This is because, value of *this*inside a function is equal to the object on which it is called. *someFunc* is called on *obj*, hence value of *this*inside *someFunc*is *obj*. But, *IIFE*is self invoked, it has not been called by any object. Hence, the value of *this*inside *IIFE*is *Window*object.

**Event handler in JavaScript**

Inside event handler, value of *this*is equal to the element on which the event is fired. Let’s see this with an example of click event.

In the above example, we have added an *on click event handler* on the *div*element which calls the *clickMe*function.



When we click on the *div*, from the above console output we can see that value of *this*is equal to the *div*element which we clicked.

We can change the value of *this*at run time using **call, apply** and **bind**function. We will discuss call, apply and bind in another article.

**Other articles**

1. [Virtual DOM in ReactJS](https://medium.com/@happymishra66/virtual-dom-in-reactjs-43a3fdb1d130)
2. [Execution Context in JavaScript](https://medium.com/@happymishra66/execution-context-in-javascript-319dd72e8e2c)
3. [Prototypes in JavaScript](https://medium.com/@happymishra66/prototypes-in-javascript-5bba2990e04b)
4. [Inheritance in JavaScript](https://medium.com/@happymishra66/inheritance-in-javascript-21d2b82ffa6f)
5. [Create objects in JavaScript](https://medium.com/@happymishra66/create-objects-in-javascript-10924cfa9fc7)
6. [Objects in JavaScript](https://medium.com/@happymishra66/objects-in-javascript-2980a15e9e71)