What is **this**?

The JavaScript `this` keyword refers to the object it belongs to.

> It has different values depending on where it is used:
>
> In a method, `this` refers to the **owner object**.
>
> Alone, `this` refers to the **global object**.
>
> In a function, `this` refers to the **global object**.
>
> In a function, in strict mode, `this` is `undefined`.
>
> In an event, `this` refers to the **element** that received the event.
>
> Methods like `call()`, and `apply()` can refer `this` to **any object**.

this in a Method

The **person** object is the **owner** of the **fullName** method.

```
<!DOCTYPE html>
<html>
<body>

<h2>The JavaScript <b>this</b> Keyword</h2>

<p>In this example, <b>this</b> represents the <b>person</b> object.</p>
<p>Because the person object "owns" the fullName method.</p>

<p id="demo"></p>

<script>
// Create an object:
var person = {
  firstName: "John",
  lastName : "Doe",
  id     : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};

// Display data from the object:
document.getElementById("demo").innerHTML = person.fullName();
</script>

</body>
</html>
```

Output:

The JavaScript this Keyword

In this example, **this** represents the **person** object.

Because the person object "owns" the fullName method.

John Doe

this Alone

When used alone, the **owner** is the Global object, so `this` refers to the Global object.

In a browser window the Global object is `[object Window]`:

```
<!DOCTYPE html>
<html>
<body>

<h2>The JavaScript <b>this</b> Keyword</h2>

<p>In this example, <b>this</b> refers to the window Object:</p>

<p id="demo"></p>

<script>
var x = this;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Output:

The JavaScript this Keyword

In this example, **this** refers to the window Object:

[object Window]

In **strict mode**, when used alone, `this` also refers to the Global object `[object Window]`:

```
<!DOCTYPE html>
<html>
<body>

<h2>The JavaScript <b>this</b> Keyword</h2>

<p>In this example, <b>this</b> refers to the window Object:</p>

<p id="demo"></p>

<script>
"use strict";
var x = this;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Output:

The JavaScript this Keyword

In this example, **this** refers to the window Object:

[object Window]

this in a Function (Default)

In a JavaScript function, the owner of the function is the **default** binding for `this`.

So, in a function, `this` refers to the Global object `[object Window]`.

```
<!DOCTYPE html>
<html>
<body>

<h2>The JavaScript <b>this</b> Keyword</h2>

<p>In this example, <b>this</b> represents the object that "owns" myFunction:
</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = myFunction();
function myFunction() {
  return this;
}
</script>

</body>
</html>
```

Output:

The JavaScript this Keyword

In this example, **this** represents the object that "owns" myFunction:

[object Window]

this in a Function (Strict)

JavaScript **strict mode** does not allow default binding.

So, when used in a function, in strict mode, `this` is `undefined`.

```
<!DOCTYPE html>
<html>
<body>

<h2>The JavaScript <b>this</b> Keyword</h2>

<p>In a function, by default, <b>this</b> refers to the Global object.</p>
<p>In strict mode, <b>this</b> is <b>undefined</b>, because strict mode does
not allow default binding:</p>

<p id="demo"></p>

<script>
"use strict";
document.getElementById("demo").innerHTML = myFunction();
function myFunction() {
  return this;
}
</script>

</body>
</html>
```

Output:

The JavaScript this Keyword

In a function, by default, **this** refers to the Global object.

In strict mode, **this** is **undefined**, because strict mode does not allow default binding:

undefined

this in Event Handlers

In HTML event handlers, `this` refers to the HTML element that received the event:

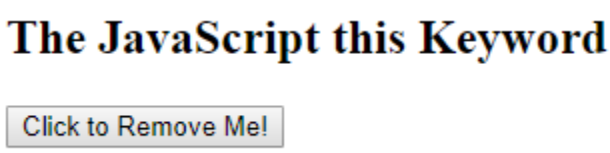
```
<!DOCTYPE html>
<html>
<body>

<h2>The JavaScript <b>this</b> Keyword</h2>

<button onclick="this.style.display='none'">Click to Remove Me!</button>

</body>
</html>
```

Output:



Object Method Binding

In these examples, `this` is the **person** object (The person object is the "owner" of the function):

```
<!DOCTYPE html>
<html>
<body>

<h2>The JavaScript <b>this</b> Keyword</h2>

<p>In this example, <b>this</b> represents the person object that "owns" the
fullName method.</p>

<p id="demo"></p>

<script>
// Create an object:
var person = {
  firstName  : "John",
  lastName   : "Doe",
  id     : 5566,
  myFunction : function() {
    return this;
  }
};

// Display data from the object:
document.getElementById("demo").innerHTML = person.myFunction();
</script>

</body>
</html>
```

Output:

The JavaScript this Keyword

In this example, **this** represents the person object that "owns" the fullName method.

[object Object]

Explicit Function Binding

The `call()` and `apply()` methods are predefined JavaScript methods.

They can both be used to call an object method with another object as argument.

In the example below, when calling person1.fullName with person2 as argument, `this` will refer to person2, even if it is a method of person1:

```
<!DOCTYPE html>
<html>
<body>

<h2>The JavaScript this Keyword</h2>
<p>In this example <strong>this</strong> refers to person2, even if it is a
method of person1:</p>

<p id="demo"></p>

<script>
var person1 = {
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
}
var person2 = {
  firstName:"John",
  lastName: "Doe",
}
var x = person1.fullName.call(person2);
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Output:

The JavaScript this Keyword

In this example **this** refers to person2, even if it is a method of person1:

John Doe