‘this’ Keyword In a Function

The **‘this’ keyword** refers to an object. But exactly which object it’ll refer to depends on the context in which ‘this’ is **called.**

1. **Printing ‘this’ in the global scope**

```
> console.log(this);
  ▶ Window {window: Window, self: Window, document: document, name: '', location: Location, …}
```

The ‘this’ keyword refers to the window object in the global scope.

2. **Printing ‘this’ inside a function**

Consider the following ways of calling a function:

1. *demo()* is being **called in the global scope**, therefore the value of ‘this’ inside *demo()* becomes the window object.

```
> function demo(){
      this.demoName = "name" ;
      console.log(this) ;
  }

  demo();
  ▶ Window {window: Window, self: Window, document: document, name: '', location: Location, …}
```

2. The *demo()* function is being **called using the ‘new’ keyword,** therefore the value of ‘this’ inside *demo()* becomes a reference to a newly created object which contains the function object. (You’ll learn about how ‘new’ keyword works in the next video lecture)

```
> function demo(){
      this.demoName = "name" ;
      console.log(this) ;
  }

  new demo();
  ▶ demo {demoName: 'name'}
<· ▶ demo {demoName: 'name'}
```

3. *printName()* is being called in the global scope. But it is **called using the *demoObject* object**, therefore the value of ‘this’ inside *printName()* becomes the *demoObject* itself.

```
> var demoObject = {
      name: "I am demo",
      type: "I am an object",
      printName: function() {
          console.log(this) ;
      }
  }

  demoObject.printName();
  ▶ {name: 'I am demo', type: 'I am an object', printName: ƒ}
```

4. Here *demoObject.printName* is NOT calling the function, **but it only contains a reference to the function**. This reference is stored inside *globalDemoFunction.* Now *globalDemoFunction* is being **called in the global scope**, hence the value of 'this' inside this function is the window object.

```
> var demoObject = {
      name: "I am demo",
      type: "I am an object",
      printName: function() {
          console.log(this) ;
      }
  }

  var globalDemoFunction = demoObject.printName ;
  globalDemoFunction();
  ▶ Window {window: Window, self: Window, document: document, name: '', location: Location, …}
```

Conclusion:

Every function while executing has a reference to its current execution context, which can be referenced by 'this'. This execution context gets created when a function is called. **Therefore the value of 'this' inside a function depends on how and where the function is called.**

Implicit Binding	implicitly decides the value of 'this'.
Explicit Binding	explicitly specifies what the value of 'this' will be.
New Keyword	'this' takes up the value of a newly created object which refers to another object. This other object refers to the prototype of the function being called.
Default binding	when none of the above rules applies, the 'this' takes the value of the window object.