This

Understanding the "this" keyword in JavaScript

Table of Content

- What Is "This"?
- Usages Of "This" Keyword
  - In Objects
  - In Browser
  - In Events
- "This" In Functions
- Explicit Binding

Have a Question?

#js-advanced

this

Introduction to "this"

Interpreter and Execution Context

- Interpreter **reads** and **executes** code **line by line**
- Execution Context
  - The **scope** in which the line is being executed
- The JavaScript runtime **maintains** a **stack** of these execution contexts
  - The execution context present at **the top** of this **stack** is currently being executed

What is Function Context?

- The **function context** is the object that **owns** the currently executed code
- Function context === **`this`** object
- Depends on how the function is invoked
  - Global invoke: **`func()`**
  - **`object.function()`**
  - **`domElement.event()`**
  - Using **`call()`** / **`apply()`** / **`bind()`**

this

- Special keyword in JavaScript
- Its value is **based** on the **context**
- There are differences in **strict mode**
- The **object** that **`this`** refers to **changes** every time execution **context** is **changed**

"This" Refers to The Global Object

- When used alone, the **owner** is the **`Object [global]`**

```
console.log(this === global); // false

function solve() {
  return this;
}

console.log(solve() === global) // true
```

"This" Keyword in the Browser

```
let b = "b";
console.log(this.b); //undefined
```

var a = "a";

console.log(this.a); *//a*

```
function foo() {
  console.log("Simple function call");
  console.log(this === window);  // true
}
foo();
```

"This" in Strict Mode

- Strict Mode

```
function solve() {
  "use strict";
  console.log(this);
}
solve(); // undefined
```

- No Strict Mode

```
function solve() {
  console.log(this);
}
solve();
// Object [global]
```

"This" in Different Context

"This" in a Method

- Refers to the **owner** of the method

```
let person = {
  firstName: "Peter",
  lastName: "Ivanov",
  fullName: function(){
    return this.firstName + " " + this.lastName
  },
  whatIsThis: function(){ return this }
}
console.log(person.fullName());  // Peter Ivanov
console.log(person.whatIsThis()); // person
```

"This" Refers to the Parent Object

```
function foo() {
  console.log(this === global);
}
let user = {
  count: 10,
  foo: foo,
  bar: function () { console.log(this === global); }
}
user.foo()  // false
let func = user.bar;
func() // true
user.bar()  // false
```

In Events

- In event handlers, `this` is set to the **element** the **event fired from**

```
element.addEventListener("click", function(e) {
  console.log(this === e.currentTarget); // Always true
});
```

"This" in Classes

- The value of `this` **refers** to the **newly created instance**

```
class Person {
  constructor(fn, ln) {
    this.first_name = fn;
    this.last_name = ln;
    this.displayName = function () {
      console.log(`Name: ${this.first_name} ${this.last_name}`);
    } } };
let person = new Person("John", "Doe");
person.displayName();  // John Doe
```

"This" in Functions

"This" in Functions

"This" with Inner Functions

- **`this`** variable is **accessible** only by the **function itself**

```
function outer() {
  console.log(this); // Object {name: "Peter"}
  function inner() {
    console.log(this); // Window
  }
  inner();
}
const obj = { name: 'Peter', func: outer }
obj.func();
```

"This" with Arrow Functions

- **`this`** retains the value of the **enclosing lexical context**

```
function outer() {
  const inner = () => console.log(this);
  inner();
}
const obj = {
  name: 'Peter',
  func: outer
};
obj.func(); // Object {name: "Peter"}
```

Explicit Function Binding

call, apply, bind

Explicit Binding

- Occurs when **`call()`**, **`apply()`**, or **`bind()`** are used on a function
- **Forces** a **function** call to **use** a particular **object** for this binding

```
function greet() {
  console.log(this.name);
}
let person = { name:'Alex' };
greet.call(person, arg1, arg2, arg3, ...); // Alex
```

Changing the Context: Call

- Calls a function with a given **`this`** value and **`arguments`** provided individually

```
const sharePersonalInfo = function (...activities) {

  let info = `Hello, my name is ${this.name} and`+
+ `I'm a ${this.profession}.\n`;

  info += activities.reduce((acc, curr) => {

      let el  = `--- ${curr}\n`;

      return acc + el;

  }, "My hobbies are:\n").trim();
  return info;

}
// Continues on the next slide...
```

Changing the Context: Call

```
const firstPerson = { name: "Peter", profession: "Fisherman" };

console.log(sharePersonalInfo.call(firstPerson, 'biking',
'swimming','football'));

// Hello, my name is Peter.

// I'm a Fisherman.

// My hobbies are:

// --- biking

// --- swimming

// --- football
```

Changing the Context: Apply

- Calls a function with a **given this value**, and **arguments** provided as an **array**
- **`apply()`** accepts a **single array** of arguments, while **`call()`** accepts an **argument list**
- If the first argument is **undefined** or **null** a similar outcome can be achieved using the array **spread syntax**

Apply() - Example

```
const firstPerson = {
  name: "Peter",
  prof: "Fisherman",
  shareInfo: function () {
    console.log(`${this.name} works as a ${this.prof}`);
  }
};
const secondPerson = { name: "George", prof: "Manager" };
firstPerson.shareInfo.apply(secondPerson);
// George works as a Manager
```

Changing the Context: Bind

- The **`bind()`** method creates a **new function**
- Has its **`this`** keyword **set** to the **provided** value, with a given sequence of arguments preceding any provided when the **new function** is called
- Calling the bound function generally results in the **execution** of its **wrapped function**

Bind - Example

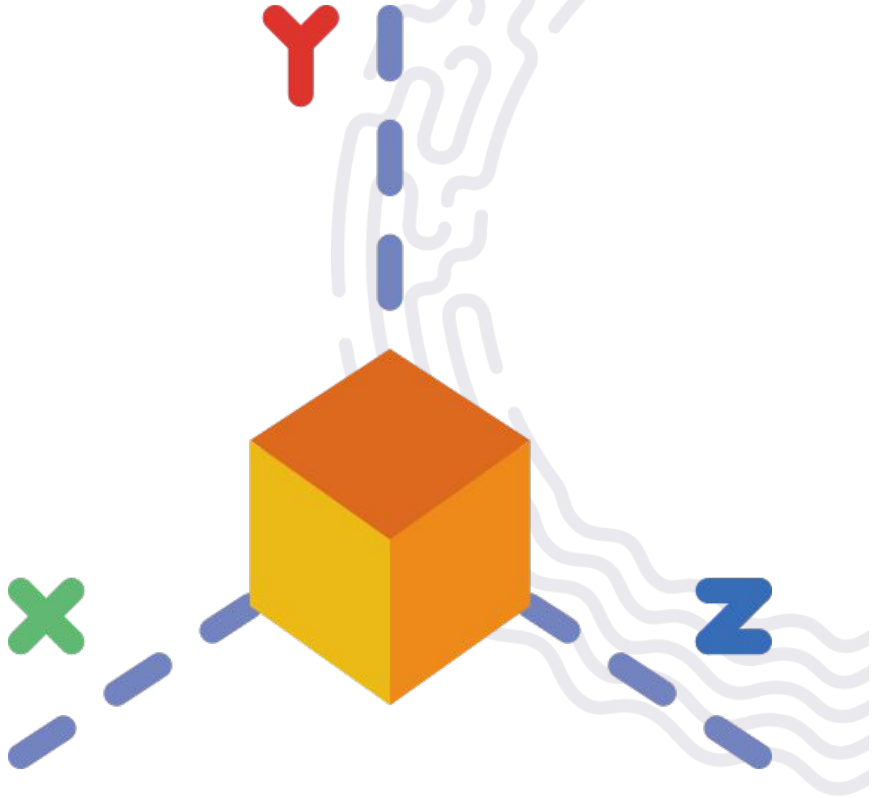
```
const x = 42;
const getX = function () {
  return this.x;
}
const module = {x , getX };
const unboundGetX = module.getX;
console.log(unboundGetX()); // undefined
const boundGetX = unboundGetX.bind(module);
console.log(boundGetX()); // 42
```

Problem: Area and Volume Calculator

- The functions **`area`** and **`vol`** are **passed as parameters** to your function

```
function area() {
  return this.x * this.y;
};
```

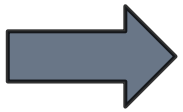
```
function vol() {
  return this.x * this.y * this.z;
};
```



Problem: Area and Volume Calculator

- Calculate the **area** and the **volume** of figures, which are defined by their coordinates (**x**, **y** and **z**), **using** the **provided functions**

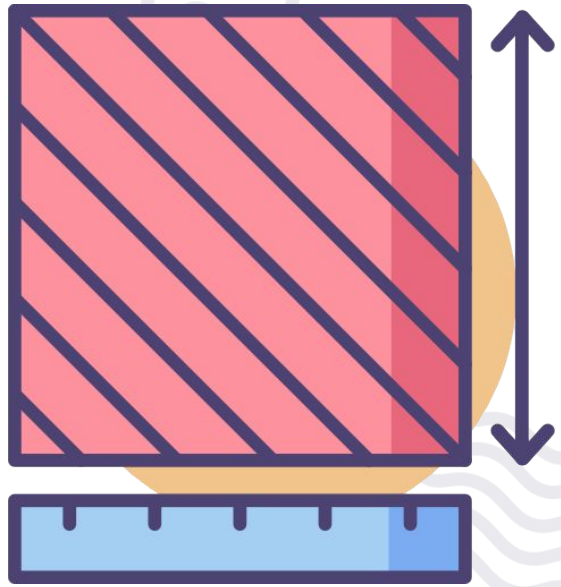
```
'[
{"x":"1","y":"2","z":"10"},
{"x":"7","y":"7","z":"10"},
{"x":"5","y":"2","z":"10"}
]'
```



```
[
  { area: 2, volume: 20 },
  { area: 49, volume: 490 },
  { area: 10, volume: 100 }
]
```

Solution: Area and Volume Calculator

```
function solve(area, vol, input) {
  let objects = JSON.parse(input);
  function calc(obj) {
    let areaObj = Math.abs(area.call(obj));
    let volumeObj = Math.abs(vol.call(obj));
    return { area: areaObj, volume: volumeObj }
  }
  return objects.map(calc);
}
```



Problem: Person

- Return an object with **`firstName`**, **`lastName`** and **`fullName`**
  - If `firstName` or `lastName` are **changed**, then `fullName` should **also** be changed
  - If `fullName` is changed, then `firstName` and `lastName` should also be changed

```
let person = new Person("Albert", "Simpson");
console.log(person.fullName);//Albert Simpson
person.firstName = "Simon";
console.log(person.fullName);//Simon Simpson
```

Solution: Person

```
function Person(first, last) {
  this.firstName = first;
  this.lastName = last;
  Object.defineProperty(this, "fullName", {
      set: function(value) {
        // ToDo
      },
      get: function() {
        // ToDo
      }
    }
  );
}
```

Practice

Live Exercise in Class (Lab)

Summary

- Functional Context
- What `this` refers to depends on where and how the function that is being executed is called
- bind, apply and call are all functions that can be used to explicitly set the value of `this`

Questions?

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © Kingsland University – https://kingslanduniversity.com


KINGSLAND
UNIVERSITY
THANK YOU