this, Factories, Constructors and OLOO

Let's get this over with...

What is this?

- One of the most confusing mechanisms in JavaScript
- A special identifier keyword that's automatically defined for us
- It bedevils even senior JavaScript developers
- It can seem downright magical

It all comes back to the call site

To understand the keyword **this**, we need to understand where the function was called - and how it was called. There are 4 ways it can be defined:

1. Global Binding
2. Implicit Binding
3. Explicit Binding
4. **new** Binding

Global Binding

```
console.log( this );
// => window

var randomFunction = function () {
    console.log( this );
    // => window
};

randomFunction();
```

Implicit Binding

```
var person = {
    name: "Groucho",
    speak: function () {
        console.log( this );
        // => person

        console.log( this.name );
        // => "Groucho"
    }
};

person.speak();
```

Explicit Binding (Ahh!)

```
var sayHello = function () {
    console.log( "Hello, " + this.name );
};

var person = {
    name: "Zeppo"
};

sayHello.call( person );
sayHello.apply( person );

var personsHello = sayHello.bind( person );
personsHello();
```

new Binding

```
var Person = function (name) {
    this.name = name;
    console.log( this );
    // => { name: "Roger" }
};

var roger = new Person( "Roger" );
```

Factories, Constructors and OLOO

What are they?

Well, they stem from:

- Objects are annoying to create
- Objects are often inconsistent
- Inheritance with objects are difficult

They make all of these things "easier"

Factory Pattern

```javascript
var DogFactory = function (name, breed) {
    var dog = {};

    dog.name = name;
    dog.breed = breed;

    return dog;
};

var tamaskan = DogFactory("Tammy", "Tamaskan");
var buddy = DogFactory("Buddy", "Labrador");
```

Inheritance with Factories

```
var AnimalFactory = function (name) {
    var animal = {};
    animal.alive = true;
    return animal;
}

var DogFactory = function (name, breed) {
    var dog = AnimalFactory( name );

    dog.name = name;
    dog.breed = breed;

    return dog;
};

var tamaskan = DogFactory("Tammy", "Tamaskan");
```

Factory Readings

- ATEN Design
- Ilya Kantor's Version

Constructor Pattern

```
var Dog = function ( name, breed ) {
    this.name = name;
    this.breed = breed;
    this.bark = function () {
        console.log( "Woof!" );
    }
};

var tamaskan = new Dog( "Tammy", "Tamaskan" );
var buddy = new Dog( "Buddy", "Labrador" );
```

Inheritance with Constructors

```
var Animal = function () {};
Animal.prototype.breathe = function(){
    console.log( "Breathe" );
}

var Cat = function () {};
Cat.prototype = new Animal();
Cat.prototype.constructor = Cat;

Cat.prototype.somethingCatSpecific = true;
```

Inheritance with Constructors

```
var LivingThing = function (name) {
  this.name = name;
};
LivingThing.prototype.beBorn = function () {
  this.alive = true;
  console.log( "Alive!" );
};

var Wolf = function (name) {
  this.name = name;
};
Wolf.prototype = new LivingThing();
Wolf.prototype.constructor = Wolf;
Wolf.prototype.howl = function () {
  console.log("https://www.youtube.com/watch?v=5T-ZThSE5rQ");
};

var grey = new Wolf( "Hunter" );
grey.name; // => "Hunter"
grey.beBorn();
grey.howl();
```

Constructor Readings

- Pivotal
- Toby Ho
- Phrogz
- CSS Tricks

Objects Linked to Other Objects: OLOO

My preferred method.

OLOO

```
var Animal = {
    init: function (name) {
        this.name = name;
        console.log( "Born!" );
        this.alive = false;
    },
    die: function () {
        this.alive = false;
        console.log( "Dead!" );
    }
};

var Wolf = Object.create( Animal );
Wolf.howl = function () {
    console.log( "Howl" );
};

var tamaskan = Object.create( Wolf );
```

OLOO Readings

- Stack Overflow: OLOO
- Getify Gist: OLOO
- John Dugan: OLOO
- You don't know JS: Behaviour Delegation