Javascript

Luca Pau
luca.pau82@gmail.com

- Object Oriented
- Dynamic and lose typed
- No classes, no interfaces, no abstraction
- Prototypical inheritance
- No privacy control at object level
- Object as collection of properties
- Function scope

Variable

- Local variable declaration by **var** keyword
    - var foo = ‘some’;
- Global variable declaration, without **var**
- Dynamic type

Variable types

- Number, String, Boolean
- Object, function, Array, Date, RegExp
- Null, Undefined

TYPEOF - ???

```
console.log(typeof "string"); // returns "string"
console.log(typeof 1); // returns "number"
console.log(typeof {}); // returns "object"
console.log(typeof undefined); // returns "undefined"
console.log(typeof function(){}); // returns "function"

but...

console.log(typeof []); // returns "object"
console.log(typeof null); // returns "object"
console.log(typeof NaN); // returns "number"
```

Create a function

```
var foo = function(arg) {
    ...
}
```

function is not a function

Function in javascript

- Is an object
- Can have properties and methods
- Is a Closure
- Reference can be stored in variable
- Can create other functions
- Have own scope
- Have special property - prototype
- Native methods bind, apply, call

Function arguments

- Are stored in pseudo-array **arguments**
- Primitives are always passed by value
- Objects are passed by reference

A sample function

```
var multiplier = function(val) {
    this.result = (this.result || 1) * val;
    return this.result;
};

console.log(multiplier(2));                2
console.log(multiplier(4));                8
console.log(multiplier(2));               16
console.log(multiplier.result);   undefined
console.log(result);                      16
```

A sample function

```
var multiplier = function(val) {
    multiplier.result = (multiplier.result || 1) * val;
    return multiplier.result;
};

console.log(multiplier(2));                         2
console.log(multiplier(4));                         8
console.log(multiplier(2));                        16
console.log(multiplier.result);                    16
```

It works but still have privacy problem

THIS IS NOT THIS

"This" by default refers to the object that a function is a member of

```
{
  var multiplier = function(val) {
      this.result = (this.result || 1) * val;
      return this.result;
  };
}
```

In this case, function is member of default global object so "this" refers to global object

Scope

```
var foo = function(val) {
     // new scope
};
```

```
(function() {
    // new scope
}());
```

Scope

- Is static
- Is created only by function
- Function arguments becomes part of the scope
- Child scope have reference to parent scope (scope chain)
- **this** is not scope

Scope

```
var myVar = 'global var';
var foo = function () {
    var myVar = 'internal var';
    console.log(myVar);
};
console.log(myVar); // global var
foo();              // internal var
```

Scope based privacy

Creator function

Initialize variable in creator function scope

Do operation on parent scope's variable "result"

```
var multiplier = (
    function() {
        var result = 1;
        return function(val) {
            result = result  * val;
            return result;
        };
    }()
);

console.log(multiplier(2));            2
console.log(multiplier(4));            8
console.log(multiplier(2));           16
console.log(multiplier.result);   undefined
```

Do operation on parent scope's variable "result"

.BIND

```
var foo = function(param1) {
    console.log(param1);
};

var bar = foo.bind(null, 'bar');

foo(1);       1
bar();       bar
```

Object reference wich will be bound to “this”

value bound to function’s parameters

.BIND

```
var foo = function() {
    console.log(this.name);
};

var goo = {'name' : 'jack'}
var bar = foo.bind(goo);

foo();      undefined
bar();         jack
```

Create an object

```
var obj = {};
```

Where is the class??

Object-oriented, NOT Class-oriented

Javascript Object

- Dynamic, not ordered, key-value
- Collection of properties
- Array access or object access
- Iterable
- Created in runtime
- Object literal {}
- No privacy control at object level
- Prototypical inheritance
- Constructor functions

Javascript Object

```
var obj = {
    property1 : ''
};

obj.method1 = function() {
    console.log('foo');
};

obj.method1();        // foo
obj['method1']();     // foo
```

Iterate Object

```
for(var i in obj) {
    ....
}
```

It doesn’t guarantee order

Constructor

Constructor is not magic object method.
It's a function which returns new objects.

```
var HelloWorld = function() {
    return {
        param : 'foo',
        print : function() {
            console.log(this.param)
        }
     }
};

var printer = HelloWorld();
printer.print();    //foo
```

New

- Have to be used with constructor function
- Binds new object to constructor’s **this**
- Executes constructor function

New

- Have to be used with constructor function
- Uses **prototype** function property reference as __proto_of new object
- Binds new object to constructor’s **this**
- Executes constructor function

Inheritance

```
var Person = function(name) {
    this.name = name;
};

Person.prototype.callMe = function() {
    console.log(this.name);
}

var Child = function(name) {
    Person.call(this, name)
}

Child.prototype = new Person();
Child.prototype.logMeNow = function() {
    console.log('Hello I\'m ' + this.name);
}

var c = new Child('mike');
c.callMe();
c.logMeNow();
```

Call parent constructor

Connect parent prototype